Highlights

An empirical study of the naïve REINFORCE algorithm for predictive maintenance

Rajesh Siraskar, Satish Kumar, Shruti Patil, Arunkumar Bongale, Ketan Kotecha

- Contributes to the broader goal of AutoML and AutoRL for PdM
- Targeted toward industrial practitioners not accustomed to RL hyperparameter tuning
- Design and implement an RL environment for PdM of a milling machine.
- Rigorous evaluation of four standard, untuned, RL algorithms.
- Simple performance evaluation measures and plots used

An empirical study of the naïve REINFORCE algorithm for predictive maintenance

Rajesh Siraskar[a,c,1], Satish Kumar[a,b,*], Shruti Patil[a,b], Arunkumar Bongale[a], Ketan Kotecha[a,b]

[a]*Symbiosis Institute of Technology, Symbiosis International (Deemed University), Pune, 412115, Maharashtra, India*
[b]*Symbiosis Centre for Applied Artificial Intelligence, Symbiosis International (Deemed University), Pune, 412115, Maharashtra, India*
[c]*Birlasoft Ltd., , Pune, 411057, Maharashtra, India*

Abstract

Industrial systems are highly complex and dynamic electro-mechanical systems. Reinforcement Learning (RL) offers an autonomous learning framework to generate optimal predictive maintenance (PdM) policies for such systems. RL algorithms are highly sensitive to hyper-parameter tuning and network architecture, and this is where Automated Reinforcement Learning, AutoRL, can offer industrial practitioners a platform to encourage the application of RL to their problems. AutoRL applied to the field of PdM, has yet to be studied. Aimed at industrial practitioners unfamiliar with complex RL tuning, we undertake an empirical study to appreciate the effects of *untuned* RL algorithms for generating an optimal tool replacement policy for a milling machine.

We compare a naive implementation of REINFORCE against the policies of industry-grade implementations of three advanced algorithms – Deep Q-Network (DQN), Advantage Actor-Critic (A2C), and Proximal Policy Optimization (PPO). Our broad goal was to study model performance under four scenarios: (1) simulated tool-wear data, (2) actual tool-wear data (benchmark IEEE*DataPort* PHM Society datasets), (3) univariate state with added noise levels and a random chance of break-down, and finally (4) complex multivariate state. Performance was measured by how accurately the agent suggested tool replacement compared to a deterministic preventive maintenance rule based on tool-wear threshold. Across

*Corresponding author
Email address: satishkumar.vc@gmail.com (Satish Kumar)
[1]ORCID 0000-0003-2341-8787

15 environment variants, REINFORCE models demonstrated a tool replacement precision of 0.687 against 0.449 for A2C, 0.418 for DQN, and 0.472 for PPO. The F1 scores were 0.609, 0.442, 0.374, and 0.345, respectively. Variability in precision and F1 was lower for REINFORCE. Comparing the best *auto-selected* model, over ten rounds of training produced unusually wider gaps in performance. REINFORCE precision/F1 stood at 0.884/0.873. The best A2C, DQN, and PPO models produced 0.520/0.639, 0.651/0.740, and 0.558/0.580, respectively. While this study is a first tiny step toward AutoRL for PdM, our findings, surprisingly, indicate that the computationally lightweight REINFORCE performs significantly well for this particular problem.

For reproducibility – model training and testing code, data and the trained REINFORCE models have been uploaded to https://github.com/to-be-provided

Keywords: Reinforcement Learning, Predictive maintenance, REINFORCE

1. Introduction

Milling machines are highly versatile, ubiquitous tools serving a variety of industries. A milling machine removes metal from the work piece by rotating and driving a cutting device into it. Abrasive forces cause tool wear, and optimal tool replacement reduces direct costs and optimizes the machines' downtime. This is an important goal, considering that the global milling machine market is valued at USD 68.3 billion (Future Market Insights, 2023). The cutting tool experiences multiple types of wear as it cuts through metal. Tool wear depends on several factors such as the cutting speed, force applied to the tool, lubrication and materials of the work piece and cutting tool.

Reinforcement learning (RL) is an artificial intelligence technique inspired by nature. Fig. 1 (Sutton and Barto, 2018) shows the RL learning feedback loop. An actor or "agent" interacts with an environment and learns via "trial-and-error". It acts based on stimuli or feedback received from the environment after performing a certain action. Actions that help in achieving the learning goal receive a reward while actions that do not, are punished. Repeating this loop over thousands of episodes "reinforces" good actions thereby building a "policy" that is optimized for that goal. In the case of predictive maintenance for milling machines, the agent is the "planner" with a goal of learning an optimal tool replacement

policy. The environment is technically anything other than the agent – sensors attached to the machine, job specifications, ambient conditions etc.

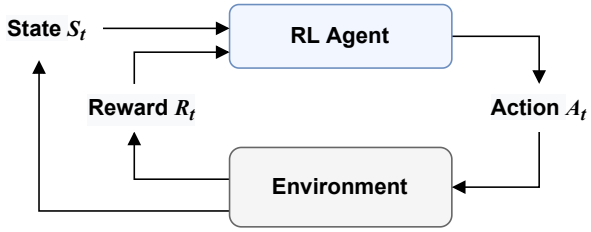


Figure 1: Reinforcement Learning

AutoML offers automated machine learning (ML) platforms. Once the user feeds in data and sets an objective, they detect data types of features and response variables, select suitable ML techniques from a library, run techniques like grid-search on the hyperparameter space, train the models, select evaluation metrics, evaluate the models using techniques such as hold-out or cross-validation and finally select the best model. Such automated pipelines are often available for supervised and unsupervised ML fields. AutoML for RL, called AutoRL, is conceptually similar and intended for automating RL development. However RL is extremely sensitive to hyperparameter and network architectures and this prevents practical out-of-the-box use of RL, (Shala et al., 2022; Afshar et al., 2022). Successful RL agents must therefore be manually trained by experts (Parker-Holder et al., 2022), which unfortunately hampers the spread of these learning methods across the wide industrial domain.

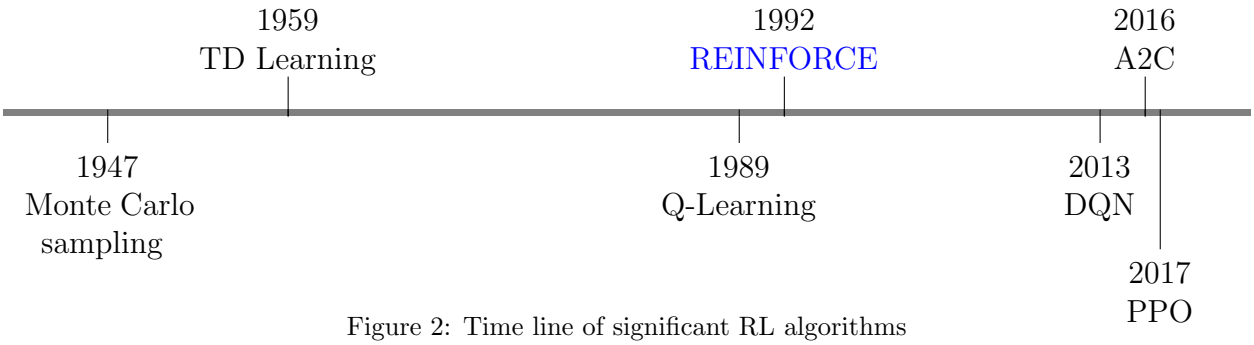


Figure 2: Time line of significant RL algorithms

Introduced in 1992, the REINFORCE algorithm (Williams, 1992) is an early policy based RL algorithm, capable of handling both discrete and continuous observation and action spaces. In practice the REINFORCE algorithm is considered a "weak" algorithm and super-

seded by several algorithms developed since. Most notably the deep-neural network version of Q-Learning, the Deep Q-Network (DQN) (Mnih et al., 2013), followed by Actor-Critic (Mnih et al., 2016) and one of the most robust modern-day algorithms, the Proximal Policy Optimization (PPO) (Schulman et al., 2017), Fig. 2. Our study takes a fresh look at REINFORCE, an otherwise neglected algorithm comparing it against three advanced algorithms, namely, DQN, Advantage Actor-Critic (A2C), and PPO. Secondly, while most RL studies are evaluated on Open AI Gym environments, our experiments cover the PdM problem using a custom built environment. In practice the milling tool is replaced after a set threshold. We use this "deterministic preventive maintenance" policy as the baseline for comparing the various policies. Our systematic evaluation, based on levels of environment difficulty, different bench-mark datasets and varying noise levels allow a broader comparison of the algorithms. Finally, we conduct statistical tests to ensure a robust statistical-evidence based conclusion.

The main **contributions** of this research are:

1. Contributes to the broader goal of AutoML and AutoRL for PdM.
2. Targeted toward industrial practitioners not accustomed to RL hyperparameter tuning.
3. Design and implement an RL environment for PdM of a milling machine.
4. Rigorous evaluation of four standard, untuned, RL algorithms.
5. Use of simple performance evaluation statistical measures and plots that industrial practitioners are normally used to.

The rest of the paper is structured as follows: In the next section we survey some related work and provide the necessary technical background describing the algorithms studied in this research. Section 3 discusses implementation details of the REINFORCE algorithm and the PdM environment followed by the methodology adopted for training, testing and evaluation. Section 4 and 5 present and discuss, the results of experiments. Finally, we summarize and draw conclusions in Section 6.

2. Related work and background

In this section we survey the intersection of past work related to RL for PdM with experimental evaluation of real world applications, specifically milling machines and AutoRL.

2.1. Literature Review

Application of RL for PdM has been surveyed by Panzer and Bender (2021); Erhan et al. (2021); Siraskar et al. (2023) however none of the RL research surveyed covered milling machines[2]. Similarly, AutoML for PdM was covered in only 9 research articles, with coverage limited to supervised machine learning and therefore excluded RL and therefore AutoRL.

Experimental comparison of RL algorithms, has been limited to using standard benchmark OpenAI Gym environments: Sandeep Varma et al. (2022) documents experimental evaluation of four policy-gradient and actor-critic algorithms PPO, Soft Actor Critic (SAC), Deep Deterministic Policy Gradient (DDPG) and A2C using the Pendulum, Mountain Car, Bipedal Walker, Lunar Landing and Atari 2600 game environments. Velivela and Yarram (2020) evaluate DQN, DoubleDQN, A2C, REINFORCE and PPO using Cartpole, Space Invaders and the Lunar Lander. Dulac-Arnold et al. (2021, 2020) are significant contributions toward analyzing *real-world* challenges. They performed empirical studies on Cartpole, humanoid and walker environments[3] by embedding them with real-world design concepts and then evaluating REINFORCE, Trust Region Policy Optimization (TRPO) and DDPG.

Dulac-Arnold et al. (2021); Henderson et al. (2018) tackle RL for continuous control. Henderson et al. (2018) evaluate DDPG, ACKTR, TRPO and PPO, using, like we do, the OpenAI baseline implementations, on complex MuJoCo[4] environments. Ford and Ritchie (2022) is one experimental evaluation we found based on a *real-world* application where DQN, A2C and PPO are applied for choosing the operational radio frequency (RF) mode for a multi-function RF system and go on to suggest that PPO is the best.

Our survey shows that most existing work use standard OpenAI Gym environments, which although necessary for bench-marking performance, do not provide coverage of industrial PdM. In Section 3.1 we attempt to bridge this gap by implementing a custom built environment.

[2] As of: 20-Aug-2023, across IEEE Xplore™, Scopus™ and Web Of Science™
[3] From the Real-World Reinforcement Learning (RWRL) Suite
[4] MuJoCo: 'Multi-Joint dynamics with Contact' e.g. HalfCheetah, Hopper, Walker and Swimmer

2.2. Technical Background

Key concepts of RL

A task is a goal we set for our agent to learn. In our case the agent must learn to optimally predict the replacement of the tool. Frequent tool replacements increases down-time while delaying it results in inferior work piece quality. In Fig. 1 the agent interacts with the environment by performing an action ($a \in \mathcal{A}$), which then alters the state of the environment to one of many states ($s \in \mathcal{S}$). The resulting state is determined by state-transition probabilities ($\mathcal{P}$) as governed by Markov Decision Process (MDP) theory. The new state provides a feedback via a reward ($r \in \mathcal{R}$). Higher positive rewards "reinforce" good behavior. Performing this over thousands of episodes with the objective of maximizing the total rewards R, enables the agent to develop a policy π which is essentially a mapping of the optimal action to perform given a certain state.

A **value function** computes how good a state or an action is by predicting future rewards, also known as a "return" $G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$. $\gamma \in [0, 1]$ facilitates discounting i.e. applying less weight to future rewards. Value functions can be represented by **state-value** $V_\pi(s)$ of a state s, as the expected return: $V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s]$; or an **action-value** function of a state-action pair as $Q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$.

With this brief overview of RL, we look at the core ideas of the four algorithms we experimented with.

Deep Q-Network (DQN)

Deep Q-Network (Mnih et al., 2013) significantly improved the earliest RL algorithm, Q-learning, by introducing neural networks to learn policies for high-dimension environments with two novel strategies to significantly stabilize learning – an "experience replay buffer" and a target network that was frozen and only periodically updated. Equation (1) shows the DQN loss function where D is the replay memory, sampled using a uniform distribution $U(D)$, $Q(s, a; \theta)$ parameterized with θ, helps compute the Q values and θ^- represents parameters of the frozen target Q-network. The objective then is to minimize this loss.

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\Big[\big(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\big)^2\Big] \tag{1}$$

Advantage Actor Critic (A2C)

A2C is a variant of Asynchronous Advantage Actor Critic (A3C) (Mnih et al., 2016), and uses multiple computation workers to avoid the use of a replay buffer. A2C is a policy-gradient actor-critic algorithm. Policy-gradient algorithms strive to model and optimize the policy directly. Actor-critic structures consist of two networks – a critic that updates function parameters w of the value function (i.e either $Q_w(a|s)$ or $V_w(s)$); and an actor that updates the policy parameters θ for $\pi_\theta(a|s)$, following the direction computed by critic. Actors therefore learn the parameterized policy π_θ using the policy-gradient as shown in (2).

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_t \left[A_t^\pi \nabla_\theta \ln \pi_\theta(a_t|s_t) \right] \tag{2}$$

Where the advantage function $A_t^\pi(s_t, a_t)$ measures how good or bad the action is w.r.t. policy's average, for a particular state, using (3).

$$A_t^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t) \tag{3}$$

Proximal Policy Optimization (PPO)

Schulman et al. (2017) formulated PPO which is often considered as the most robust of the RL algorithms. PPO is a policy-gradient method based on TRPO (Trust Region Policy Optimization) by Schulman et al. (2015), where the core idea is the use of a trust region to improve training stability by avoiding updates to parameters that vastly change the policy at a particular time step. TRPO ensures this by using a divergence constraint on the magnitude of policy update. If $r(\theta)$ (4) represents the ratio of probabilities between policies of previous and current iteration, then the objective function of TRPO is given by (5), where $\hat{A}$ represents the estimated advantage function.

$$r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} \tag{4}$$

$$J^{\text{TRPO}}(\theta) = \mathbb{E}[r(\theta)\hat{A}_{\theta_{\text{old}}}(s, a)] \tag{5}$$

PPO extends TRPO by additionally imposing a regional constraint. It prevents large

updates by forcing the ratio $r(\theta)$ to stay within a small interval $[1-\epsilon, 1+\epsilon]$, around 1.0, by use of a hyper-parameter ϵ.

$$J^{CLIP}(\theta) = \mathbb{E}_t\ [\ min(r_t(\theta)A_t^{\pi_{\theta_{old}}}, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t^{\pi_{\theta_{old}}})] \tag{6}$$

REINFORCE

The REINFORCE algorithm, invented by Williams (1992), directly learns a policy π_θ to produce action probabilities from states. True to the fundamental RL concept, actions that cause favorable states, are positively reinforced thereby increasing their probability of occurrence, while those resulting in unfavorable states are penalized. Equation (7) is the objective function the agent maximizes and is defined as the expected return over many trajectories sampled from the policy. Comparing this to the objective functions of the more advanced algorithms above, we see just how very simple it indeed is.!

$$\max_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}[R(\tau)] \tag{7}$$

REINFORCE uses policy gradient (8) to update the action probabilities.

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_t\ [\ R_t(\tau)\ \nabla_\theta \ln \pi_\theta(a_t|s_t)\ ] \tag{8}$$

3. Methodology

3.1. Implementation details

RL requires an environment to function. We built a custom "milling" environment for the agent to learn a policy for tool replacement. Fig. 3 shows the stepped approach used to generate the fifteen environments, of varying complexity. We first describe the simulation-based environment followed by the real-data based family, describing the child-variants along the way.

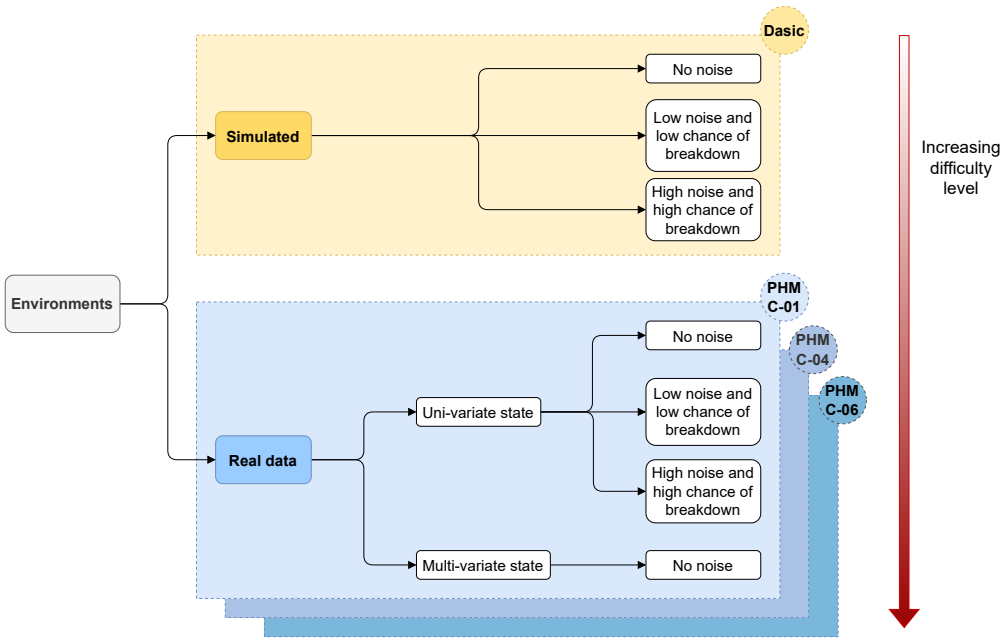
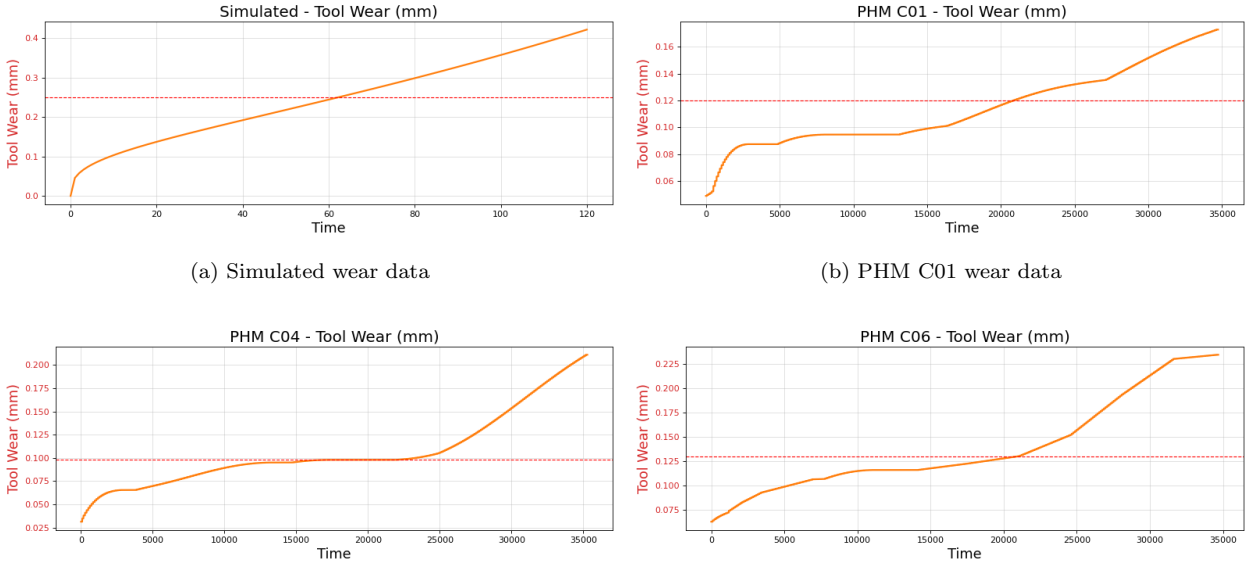


Figure 3: The fifteen different environments used for evaluation.



(a) Simulated wear data

(b) PHM C01 wear data

(c) PHM C04 wear data

(d) PHM C06 wear data

Figure 4: Tool wear data. Red dotted line indicates the wear threshold beyond which tool is replaced.

Simulating tool wear

Dašić (2006) provides a parameterized power-exponential function for modeling tool wear (9), where VB represents the flank wear in mm.

$$VB = a \cdot t^{b_1} \cdot e^{b_2 \cdot t} \Big|_{t=t_0}^{t=t_1} \tag{9}$$

We used the parameter values provided in the paper $a = 0.08257$, $b1 = 0.3342$ and $b2 = 0.03147$ to simulate 120 data points. Fig. 4a shows the tool wear simulated using (9), with the red dotted-line indicating the wear threshold beyond which tool is replaced. This provided the mechanism to simulate the basic variant of tool-wear based **univariate state**. We then added two further levels of increased complexity using noise and a chance of breakdown, giving us three distinct environments.

Actual tool wear data

The IEEE*DataPort* hosts the "2010 PHM Society" tool-wear data obtained from a high-speed CNC milling machine, (Li, 2021). C01, C04 and C06 datasets are suggested as benchmarks to be used for machine learning research and were the ones we used[5]. The data is from seven sensors – dynamometer measuring forces in X, Y and Z dimensions, measured in N; accelerometer measuring vibration in X, Y and Z dimensions, measured in g and finally acoustic emission data as AE-RMS, in V. A separate file contains tool wear data in mm. Figures 4b, 4c and 4d show the tool wear for the three datasets. We use real data to create two state designs, an univariate state consisting of only the tool wear and a **multivariate** state designed using all the additional seven sensor values. Just as we did for the simulated case, the complexity of the univariate state is increased using two levels of noise and breakdown parameters. The multivariate state is complex in itself, Fig. 5, and we use the natural (without noise) form.

In practice, state variables are often normalized to improve stability and convergence. Both the simulated and real data was normalized using min-max scaling such that the tool wear and other state features, $x \in [0,\ 1] \subset \mathbb{R}$. We will see next how this allows adding white noise of similar magnitudes across different PHM datasets.

Adding noise and chance of break-down

Fig. 6 shows the effect of adding two levels of noise; a "low" level by adding Gaussian noise of order -3, $(0.0, 0.001]$ and "high" of order -2, $(0.0, 0.01]$. Since the tool wear is less than 0.24 mm, this adds significant perturbations as seen in Fig. 6b and 6c. The noise affects

[5]In the article, we often refer to these datasets as "PHM-" data.

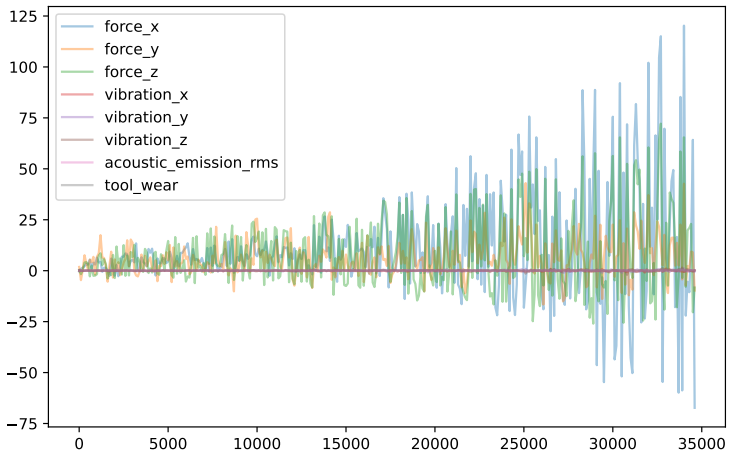


Figure 5: PHM C06: multivariate data

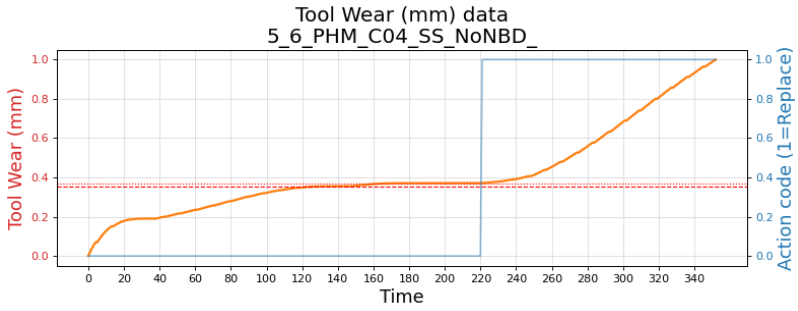
the tool replacement decision (solid blue line) around the replacement threshold (dotted red line). The *human* preventive maintenance policy replaces the tool if the wear exceeds the threshold and this decision boundary oscillates due to the noise. One can see that in the case of no noise (Fig. 6a), the decision boundary is clean.

Break down occurs due to excessive tool use and can often occur randomly. In conjunction with Guassian noise this complexity is added for the univariate state based environments. For the low-noise variant we add a low 5% chance of break down and for the high noise variant we add a higher chance of 10%. The "milling" episode is terminated if a probability, sampled from a uniform distribution is less than this "chance" threshold.

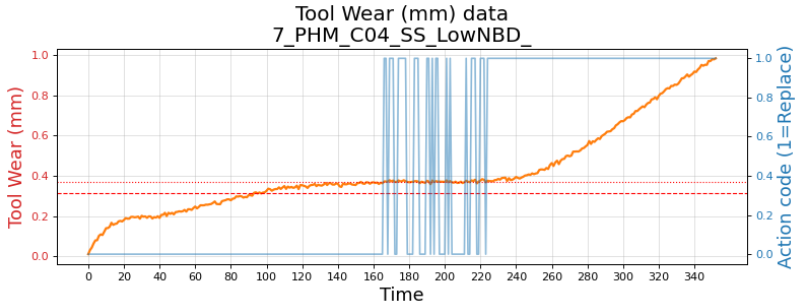
Table 1 summarizes the 15 environment variants and their three logical groups: (1) Simulated 1-3 (2) Real data – simple univariate environment (4-12) and Real data – complex multivariate (13-15).

Tool wear as a Markov Decision Processes (MDP)

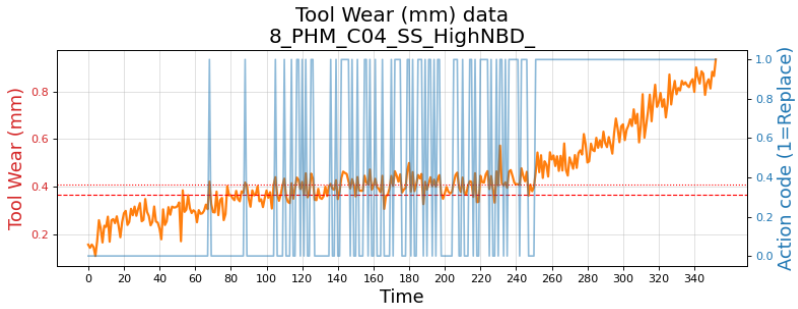
Formulating our problem to be solved by RL requires us to assume that the wear process satisfies the Markov property – which implies that the transition of tool wear to another state is dependent only on the current state and not on any previous states. MDPs are defined by the tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$. We will define the elements of state space $\mathcal{S}$, action space $\mathcal{A}$ and reward function $\mathcal{R}$, next.



(a) No noise



(b) Low noise



(c) High noise

Figure 6: PHM C04 tool wear data (normalized) and the effect of noise. Blue line is the replacement action decision.

State and environment elements

There are two basic state definitions across the 15 environment variants, the "simple univariate" and the "complex multivariate".

The elements of simple univariate state vector are $S_t = [w_t]$, where w_t is the current tool wear. As part of the environment other elements that are sensed by the agent are $[W_\tau, N_f, P_{bd}]$, where W_τ is the wear threshold, N_f is the noise factor and is one of [0, 1e-3, 1e-2], P_{bd} is the chance of tool breakdown and is one of [0, 0.05, 0.10]. The complex multivariate state $S_t = [f_x, f_y, f_z, v_x, v_y, v_z, ae_{rms}, w_t]$ where, as mentioned in Section 3.1,

	Environment variant	Noise factor	Breakdown chance
	Simulated		
1	Simulated - No noise	None	None
2	Simulated - Low noise	1e-3	0.05
3	Simulated - High noise	1e-2	0.10
	Real data – simple univariate		
4	PHM C01 SS (simple, univariate) - No noise	None	None
5	PHM C01 SS (simple, univariate) - Low noise	1e-3	0.05
6	PHM C01 SS (simple, univariate) - High noise	1e-2	0.10
7	PHM C04 SS (simple, univariate) - No noise	None	None
8	PHM C04 SS (simple, univariate) - Low noise	1e-3	0.05
9	PHM C04 SS (simple, univariate) - High noise	1e-2	0.10
10	PHM C06 SS (simple, univariate) - No noise	None	None
11	PHM C06 SS (simple, univariate) - Low noise	1e-3	0.05
12	PHM C06 SS (simple, univariate - High noise	1e-2	0.10
	Real data – complex multivariate		
13	PHM C01 MS (complex, multivariate) - No noise	None	None
14	PHM C04 MS (complex, multivariate) - No noise	None	None
15	PHM C06 MS (complex, multivariate) - No noise	None	None

Table 1: List of the fifteen environments and their categorization

(f_x, f_y, f_z) represents the force along the 3 axes, similarly v represents the vibration, ae_{rms} the acoustic emission.

Actions

The action space is binary, $\mathcal{A} \in [0, 1]$, where 0 (`CONTINUE`) represents continuation of milling operation and 1 (`REPLACE_TOOL`) represents the action of replacing the tool.

Environment feedback

"Feedback" generated by the action an agent takes, is the central mechanism by which agents learn. This is implemented via the `step()` function in the environment code and is

outlined in Algorithm 1. At every time step an action A_t is taken and the resulting state is evaluated for terminating conditions or assigning a reward and continuing.

Reward function

$$R_t = \begin{cases} +R_1 \times t, & if \;\; w_t < W_\tau \\ -R_2 \times t, & if \;\; w_t \geq W_\tau \\ += -R_3, & if \;\; \text{ACTION} = \text{REPLACE\_TOOL} \end{cases} \tag{10}$$

In the reward function (10) there are three distinct phases where a reward (or penalty) is offered as feedback. R_1, R_2 and R_3 are constants that determine the magnitude of reward. When the current state of wear w_t is less than the threshold W_τ we have a desirable condition, and we award the agent a positive reward. The formulation (10) allows a higher reward to be collected the closer it is to threshold so as to maximize tool usage, but not allowing it to cross it. If it does, the agent is penalized (negative reward) by a magnitude of R_2, and once again the farther away it is from the threshold i.e. a highly deteriorated tool the larger the penalty. To avoid this "bad" state, the agent must learn to replace the tool; represented by the third condition. Tool replacement implies a direct cost (that of the tool) and a much higher and significant downtime "cost". To ensure the agent does not learn to replace unnecessarily, we "penalize" it. It is important to note that the last condition in (10) is an "incremental addition", the first two conditions are mutually exclusive and evaluated first, followed by the last condition which is incrementally added on whatever reward is collected in the previous condition. The agent then tries to balance these three conditions, such that it maximizes its total return, over time.

Of the final two elements of the MDP quintuple, $\mathcal{P}$ represents the probability transition matrix and is usually not known and we will therefore use *model-free* RL techniques to learn that from "experiences"; γ enables the agent to learn long-term impact of its actions i.e. what is the long term impact of replacing the tool now or that of delaying the replacement, γ is set to 0.99 to facilitate this farsightedness.

Algorithm 1 Agent class `step()` method – Reward handling mechanism

procedure STEP CLASS METHOD: REWARD HANDLING

Class attributes: Wear-threshold W_τ; reward accumulated so far *Reward*; reward function parameters $R1$, $R2$ and $R3$; random chance of breakdown P_{bd}; tool replacements made so far *tool_replacements*; maximum allowable operations T

Input: Current length of episode t; current tool-wear w_t; current policy action A_t; current index of training data *data_index*

Initialize p from uniform probability distribution
if $t \geq T$ **then**:
    ▷ Termination based on length of episode.
    $Terminate \leftarrow True$
    $Reward \leftarrow 0.0$
    $data\_index \leftarrow 0$ ▷ Reset the training data-frame index to the beginning.
else if $w_t > W_\tau$ **and** $p < P_{bd}$ **then**:
    ▷ Termination based on chance of breakdown.
    $Terminate \leftarrow True$
    $Reward \leftarrow 0.0$
    $data\_index \leftarrow 0$
else
    $Terminate \leftarrow False$
    if $w_t < W_\tau$ **then**:
        ▷ Healthy tool
        $Reward \leftarrow Reward + t \cdot R1$
    else
        ▷ Tool deteriorating
        $Reward \leftarrow Reward - t \cdot R2$
    if A_t is $REPLACE\_TOOL$ **then**:
        ▷ Tool is replaced
        $Reward \leftarrow Reward - R3$
        $data\_index \leftarrow 0$ ▷ Tool replaced, therefore roll back tool life
        Increment *tool_replacements*

Network architecture and basic hyper-parameters

Stable-Baselines3 (SB3) provides robust open source implementations of many RL algorithms, (Raffin et al., 2021). As of 20-Aug-2023, 13 algorithms have been implemented

however REINFORCE has *not* been implemented, (SB3-Algorithms, 2022). For this research we use the *default* SB3 implementations of DQN, A2C and PPO and compare its performance to a custom implementation of the REINFORCE algorithm using a very simple network architecture. Table 2 shows a comparison of the architecture and the basic common hyper-parameters. The REINFORCE uses a *single* internal layer of 64 units.

	A2C	**DQN**	**PPO**	**REINFORCE**
Network architecture	input dim x [64—Tanh x 64—Tanh] x output dim	input dim x [64—Tanh x 64—Tanh] x output dim	input dim x [64—Tanh x 64—Tanh] x output dim	input dim x [64—ReLU] x output dim
Layers	2	2	2	1
Units	64 x 64	64 x 64	64 x 64	64
Activation	Tanh, Tanh	Tanh, Tanh	Tanh, Tanh	ReLU
Optimizer	RMSprop	Adam	Adam	Adam
Learning rate	0.0007	0.0001	0.0003	0.01
Gamma	0.99	0.99	0.99	0.99

Table 2: Comparing the network architecture and basic hyper-parameters across algorithms

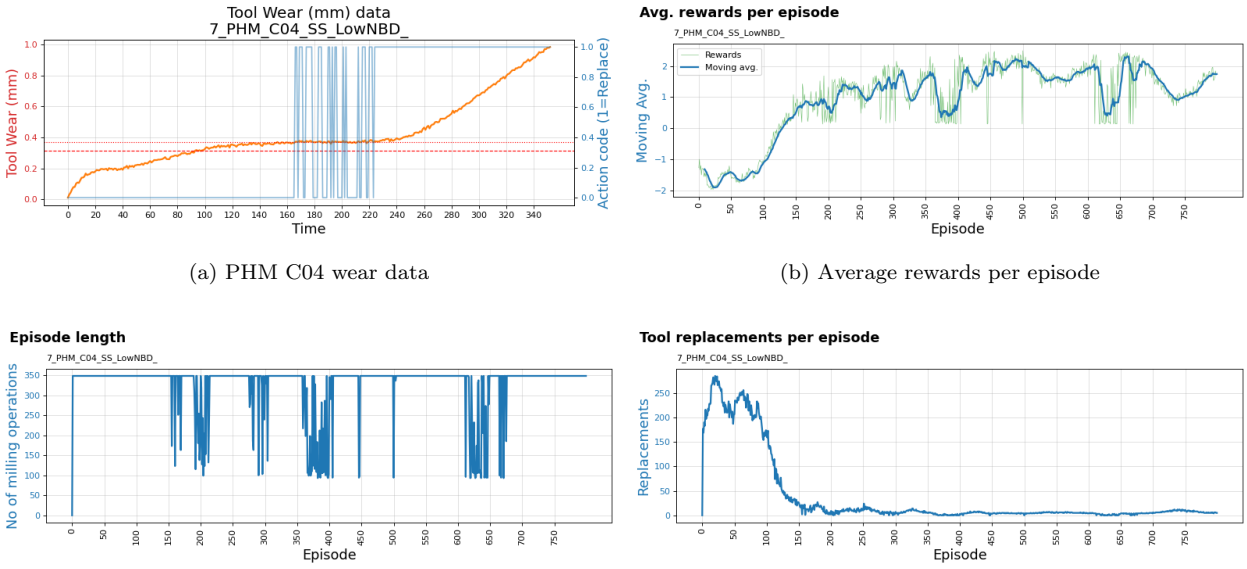
The default architecture for all three SB3 algorithms (A2C/DQN/PPO) consists of *two* fully connected layers with 64 units per layer (SB3-Default Network Architecture, 2022). While REINFORCE used ReLU (rectified linear unit) as the activation function, the other three algorithms used hyperbolic tangent (Tanh). Finally, REINFORCE's learning-rate is much larger.

3.2. Training

The training strategy must ensure a uniform comparison of the algorithms. We maintained the exact *same* environment variant, wear dataset, noise parameter, probability of breakdown parameter, and the three reward function parameters; across all four algorithms during a single training round. As the wear data is time-series data, the training and test

sets are created by systematic sampling. Simulated data and real tool wear data (PHM) was randomly sampled at a certain frequency and down sampled into training and test sets.

The REINFORCE was trained for 800 episodes for the simulated and PHM univariate variants, for all three noise and breakdown levels (none, low and high) – Table 1 items 1-12. For the PHM multivariate variant, Table 1 items 13-15, REINFORCE was trained for 1000 episodes. SB3 algorithms were trained for 10,000 episodes for all variants. We ran ten rounds of training, tested each generated model and averaged results over the 10 rounds. Testing is explained in the next section while results are presented in Section 4.



(a) PHM C04 wear data

(b) Average rewards per episode

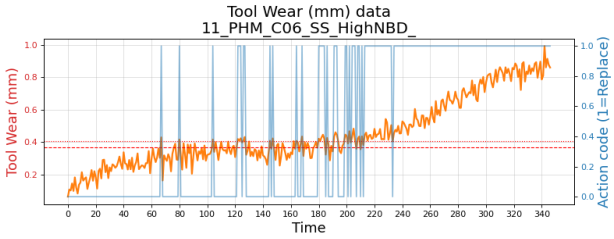
(c) Episode length completed per episode

(d) Tool replacements per episode

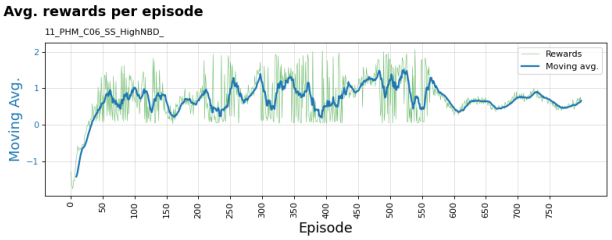
Figure 7: Training plots of REINFORCE. Dataset: PHM-C04. Variant: Univariate state, low-noise and low chance of breakdown.

Fig. 7 shows the training plots for the algorithm of our interest – REINFORCE. It displays how the wear plot looked for C04 with low noise and low chance of breakdown settings. The average rewards increase over the course of 800 episodes (Fig. 7b). The episode length (Fig. 7c) demonstrates the complexity introduced by random breakdown (which abruptly terminates the episode). It is the tool replacement policy that is of interest to the industrial practitioner – Fig. 7d shows that it decreases to optimal levels as the agent learns over time. Similarly, Fig. 8 demonstrates the training for the PHM C06 dataset affected by high noise and higher breakdown probability. Finally, for the more complex multivariate state variant, the training plots are as seen in Fig. 9; as we do not introduce

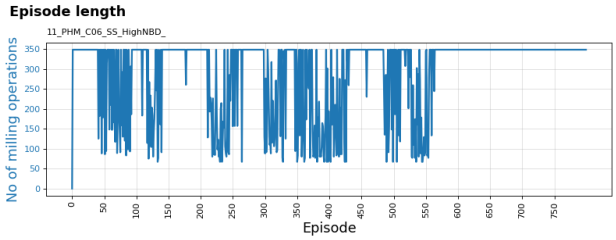
noise or breakdown here, the episodes are always completed (Fig. 9c).



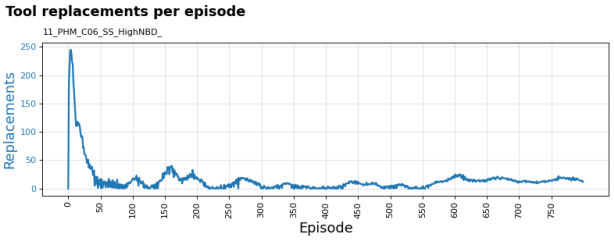
(a) PHM C06 wear data



(b) Average rewards per episode

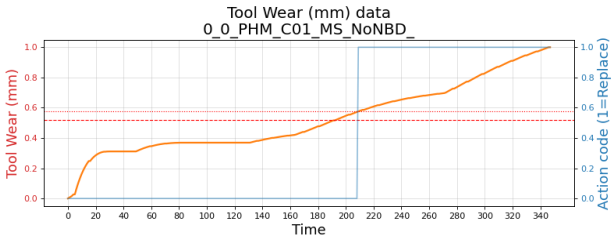


(c) Episode length completed per episode

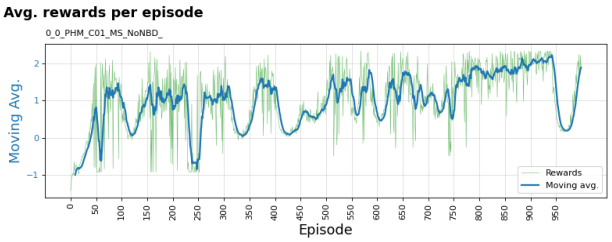


(d) Tool replacements per episode

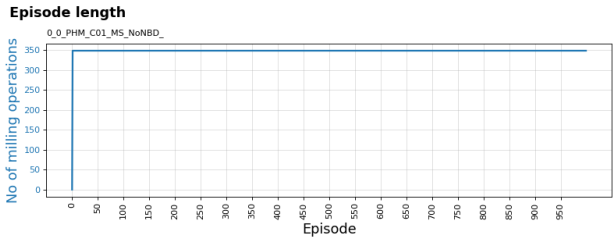
Figure 8: Training plots of REINFORCE. Dataset: PHM-C06. Variant: Univariate state, high-noise and high chance of breakdown.



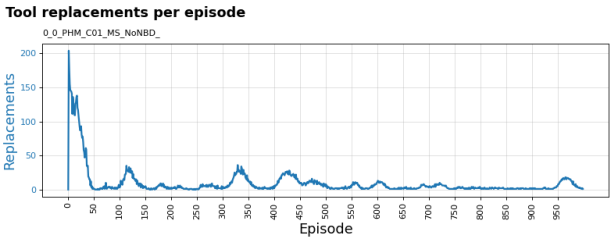
(a) PHM C01 wear data



(b) Average rewards per episode



(c) Episode length completed per episode



(d) Tool replacements per episode

Figure 9: Training plots of REINFORCE. Dataset: PHM-C01. Variant: Multivariate state, No noise or breakdown.

3.3. Testing and performance evaluation

Testing was performed with data separate from the training data. 10 rounds of testing are performed, with a *new* set of 40 test cases randomly sampled and frozen across all four algorithms, during each round.

Evaluation metrics

The human decision is based on "preventive maintenance" – replace the tool against a predefined wear threshold. In our data a tool replacement is represented as 1 and a normal operation as 0. We applied classification metrics to evaluate the RL agent decisions of tool replacement. It is worth noting that while we have selected an arbitrary wear threshold as this serves our purpose for algorithm comparison; in reality the threshold is based on several factors like materials of tool and work-piece, the duration of continuous operation, ambient conditions, "cost" of production downtime etc. Threshold could therefore vary significantly from one case to another.

		Human decision	
		Replace tool	Continue milling
Agent decision	Replace tool	TP	FP
	Continue milling	FN	TN

Figure 10: Confusion matrix: Human versus Agent decisions

Classification metrics are based on the confusion matrix shown in Fig. 10. TP represents true positive cases, where both the agent and human agree on replacing the tool. False positive FP cases denote the agent falsely suggesting replacements, while false negatives FN are cases where continuation of milling is suggested, when in fact a tool replacement would have helped. Precision (Pr), Recall (Rc) and F1-score metrics can then be computed as shown in (11).

$$\text{Pr} = \frac{TP}{TP + FP}, \quad \text{Rc} = \frac{TP}{TP + FN}, \quad \text{F1-score} = 2 \times \frac{Pr \times Rc)}{(Pr + Rc)} \tag{11}$$

Tool replacement precision: Timely replacements, TP, ensure work piece quality. While we desire high TPs, we do not want *unnecessary* replacements that cause production downtime, we therefore want lower FPs. The *precision* metric is therefore an ideal performance metric. While we prefer a high precision, we do want a reasonably high recall i.e. do not want to miss replacement opportunities (low FNs). The F1-score (true harmonic mean of precision and recall) gives us a *balanced* measure. Equation (12) provides a *weighted* mechanism to provide a higher F-score for higher precision, by setting $\beta < 1.0$. For our evaluation we set β to 0.5.

$$F_\beta = (1 + \beta^2) \cdot \frac{Pr \times Rc)}{(\beta^2 \cdot Pr + Rc)} \tag{12}$$

4. Results

We present the summarized results in this section accompanied by commentary referring to detailed results made available in Appendix A. Since there are several tables and figures, for reference, we created a cross-linked Table 3. Generally, blue text is used in tables to highlight prominent values. Summary performance and $F_\beta 0.5$ plots accompany tables to assist in visualizing the comparative performance. The $F_\beta 0.5$ plots show *individual* performance over ten rounds of training-validation and help infer training stability. Similar plots for precision, recall and F1 have been relegated to Appendix A.

4.1. Overall performance

At an aggregated level, Table 4 along with Fig. 11, show that the REINFORCE scores better than the other algorithms, on all four metrics. Tool replacement precision at 0.687, is highest of the four algorithms and better by 0.215 in absolute terms when compared to the next best, PPO. The standard deviation for precision is the lowest at 0.059. On recall, F1 and $F_\beta 0.5$, REINFORCE is better by 0.125, 0.168 and 0.195 compared to the next best.

These metrics were averaged over 10 rounds of training followed by validation. We visualize the behavior of our primary metric $F_\beta 0.5$, *individually* across the ten rounds in Fig. 12. The blue line floating above the rest of the algorithms, for all metrics, is that of REINFORCE and appears fairly stable across the ten rounds. The error-bars are also pretty

	Item	Reference
	Summary Results	
1	Overall summary – All 15 environments, averaged over 10 rounds	Table 4, Fig. 11
	–"– $F_\beta 0.5$ behavior plot over 10 rounds	Fig. 12
2	Simple univariate state – Simulated data, 3 variants (3 noise settings), averaged over 10 rounds	Table 5, Fig. 13
	–"– $F_\beta 0.5$ behavior plot over 10 rounds	Fig. 14
3	Simple univariate state – Real data, 9 variants (3 datasets $\times$ 3 noise settings), averaged over 10 rounds	Table 6, Fig. 15
	–"– $F_\beta 0.5$ behavior plot over 10 rounds	Fig. 16
4	Complex multivariate state – Real data, 3 variants (3 datasets), averaged over 10 rounds	Table 7, Fig. 17
	–"– $F_\beta 0.5$ behavior plot over 10 rounds	Fig. 18
	Super Models	
5	Super Models – Best model, *averaged* over all 15 environments	Table 8 Fig. 19
	Super Models – Best model, *details* of all 15 environments	Appendix A - Table 11
	Hypothesis tests and Training time results	
6	Hypothesis tests – p-value and t-statistic of REINFORCE versus other algorithms	Table 9
7	Training time – Averaged over 10 rounds and all 15 variants	Fig. 20
	Detailed Results - Appendix A	
8	Complete detailed results – All 15 environments, averaged over 10 rounds	Appendix A - Table 10
9	Stability behavior plots over 10 rounds, Precision, Recall and F1	
	Overall (all variants)	Apx. A - Fig. 21
	Simulated state	Apx. A - Fig. 22
	Simple univariate state	Apx. A - Fig. 23
	Complex multivariate state	Apx. A - Fig. 24

Table 3: Reference table for results.

	Precision		Recall		F1-score		$F_\beta 0.5$	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
A2C	0.449	0.088	0.480	0.084	0.442	0.070	0.436	0.071
DQN	0.418	0.185	0.504	0.032	0.374	0.035	0.348	0.058
PPO	0.472	0.144	0.316	0.087	0.345	0.091	0.393	0.105
REINFORCE	0.687	0.059	0.629	0.051	0.609	0.050	0.631	0.052

Table 4: Model performance summary - averaged over all environments.

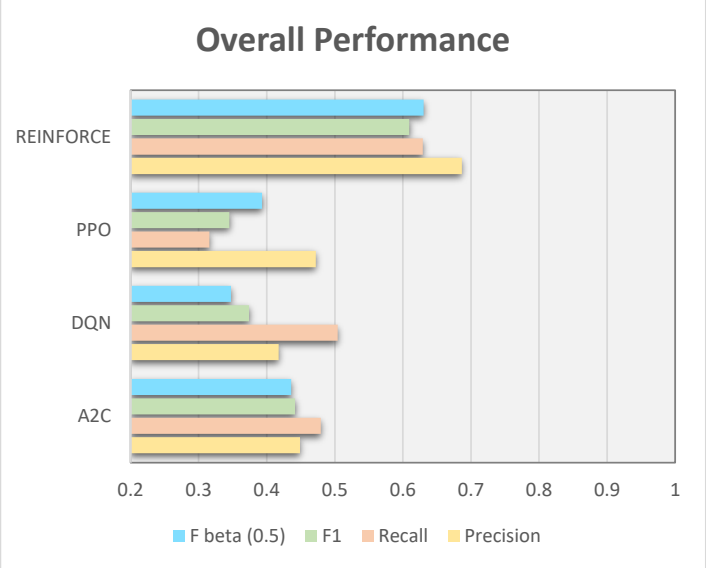


Figure 11: Overall model performance summary

small, indicating lower uncertainty. We notice that the other three algorithms are all centered closely around, 0.4.

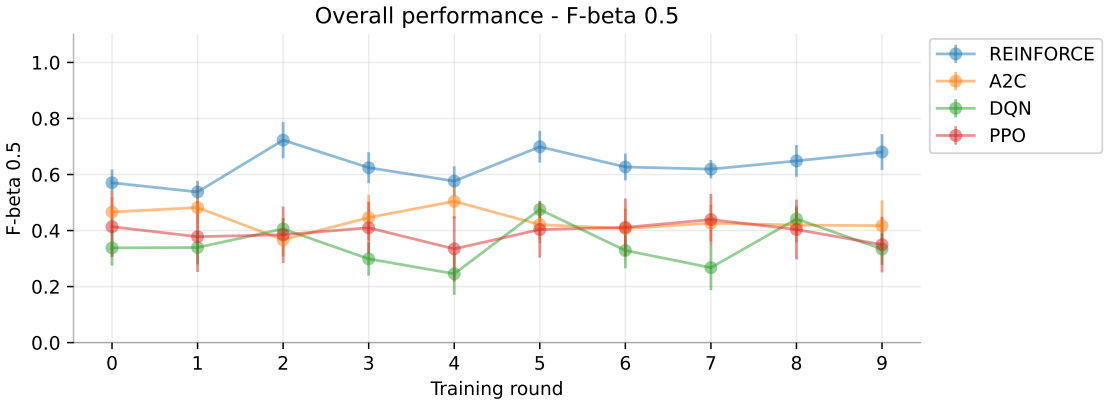


Figure 12: Overall performance comparison of the $F_\beta 0.5$ metric, over 10 rounds of training and testing.

We now dig into the detailed metrics Appendix A - Table 10 to better understand the aggregated summary. The largest values are typeset in blue. And a quick visual glance shows the dominance of REINFORCE. The notable exceptions are the PHM C01 single-state and no-noise variant where A2C and DQN do better, followed by PHM C01 and PHM C06 univariate-state, low-noise variants where A2C performs better in every aspect and finally the PHM C04 and C06 multivariate variants where A2C again performs better on recall and which in turns drives the F1 score up. Barring these three complete cases and two cases where A2C recall was better, the REINFORCE performs best in ten variants and in two cases its precision, and therefore $F_\beta 0.5$, is highest. This corroborates the overall performance observed in Table 4. Finally, Fig. 21 shows that the stability across multiple rounds is evident for other metrics as well.

4.2. Simulated environment

	Precision		Recall		F1-score		$F_\beta 0.5$	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
A2C	0.416	0.120	0.385	0.073	0.363	0.072	0.373	0.082
DQN	0.432	0.184	0.510	0.031	0.374	0.034	0.351	0.056
PPO	0.500	0.178	0.215	0.081	0.285	0.099	0.370	0.122
REINFORCE	0.806	0.040	0.915	0.038	0.841	0.035	0.816	0.037

Table 5: Simulated environments - model performance summary.

The simulated tool-wear environment is relatively the simplest for the agent to learn, even with added noise and chance of breakdown. Table 5 shows the averaged performance over the three variants. The REINFORCE performs the best and in absolute terms it is better than the next best advanced algorithm by very high margins: precision by 0.306, recall by 0.405, F1 by 0.468 and $F_\beta 0.5$ by 0.442, with standard deviation lower or marginally higher than others. $F_\beta 0.5$ plot Fig. 14 shows an exceptionally consistent performance, over all ten trained models, vis-a-vis the other algorithms. Plot Fig. 22 shows DQN fluctuating heavily, occasionally showing recalls at the REINFORCE levels (rounds 2 and 8).

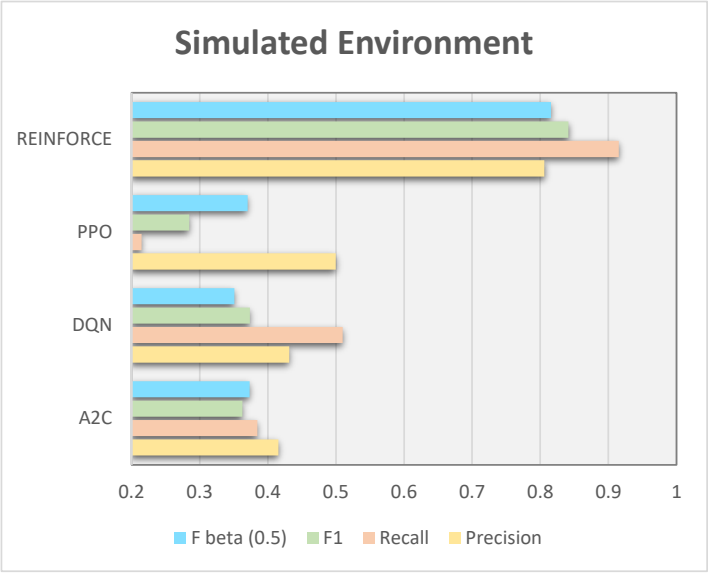


Figure 13: Simulated environments - model performance summary.

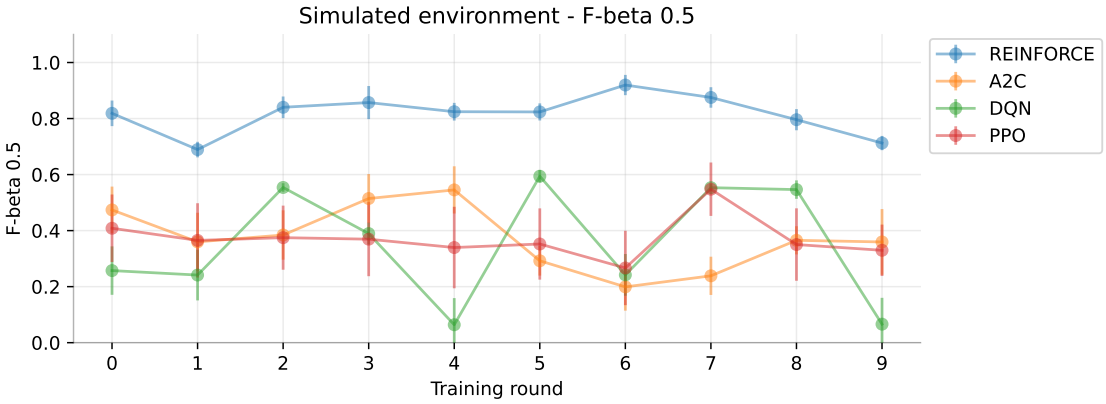


Figure 14: Simulated environments - $F_\beta 0.5$ metric, over 10 rounds of training and testing.

	Precision		Recall		F1-score		$F_\beta 0.5$	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
A2C	0.447	0.077	0.477	0.091	0.452	0.072	0.446	0.070
DQN	0.419	0.179	0.507	0.032	0.379	0.036	0.352	0.057
PPO	0.450	0.146	0.314	0.082	0.333	0.087	0.374	0.102
REINFORCE	0.605	0.046	0.603	0.046	0.570	0.041	0.576	0.040

Table 6: Model performance summary - averaged over PHM-2010 environments with simple single-variable environment.

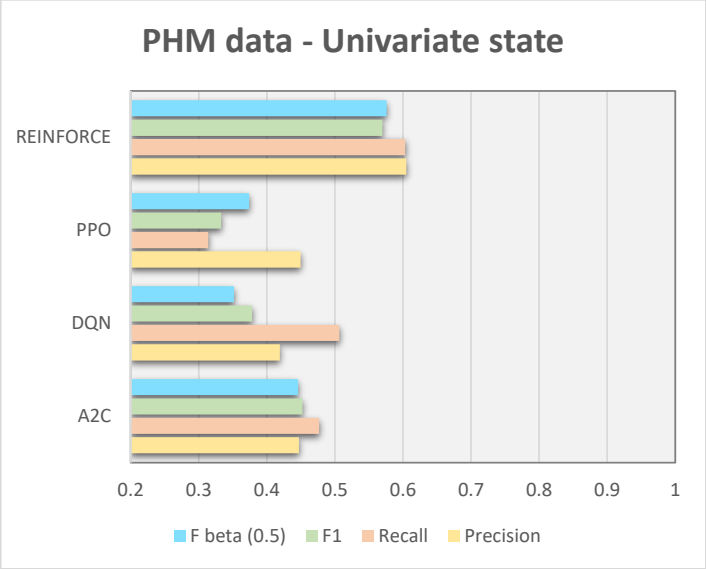


Figure 15: PHM real data - Model performance summary for the simple single-variable environment.

4.3. Real data - simple univariate environment

Real data offers a more challenging environment. Despite this, in Table 6, we notice that REINFORCE still performs better than the other algorithms. The margins are understandably lower: precision by 0.155, recall by 0.097, F1 by 0.117 and $F_\beta 0.5$ by 0.130. For these variants the $F_\beta 0.5$ plot Fig. 14 shows a considerably higher performance over 9 of the 10 training rounds. Behavior of the other metrics is seen in Fig. 23. While the REINFORCE precision is higher for most rounds, the recall seems to be occasionally surpassed slightly by DQN's (1, 8 and 9) and by a larger margin once (5).

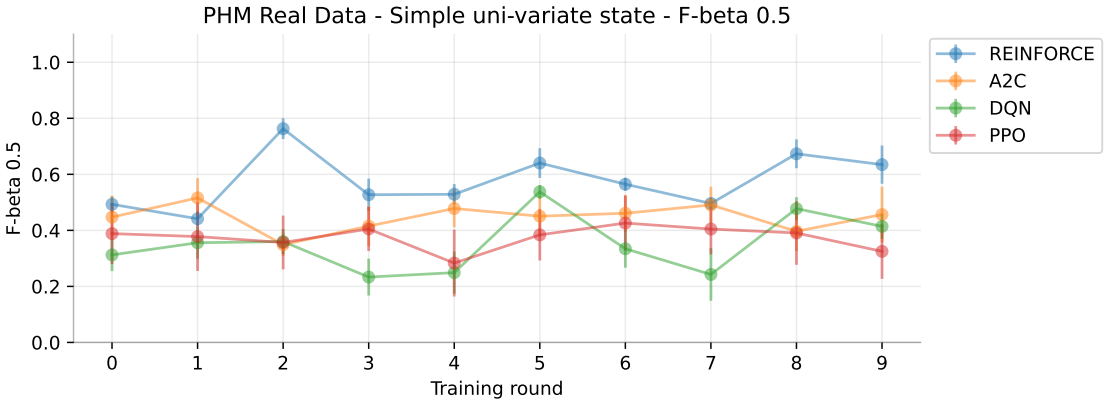


Figure 16: Univariate simple state environments - $F_\beta 0.5$ metric, over 10 rounds of training and testing.

	Precision		Recall		F1-score		$F_\beta 0.5$	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
A2C	0.487	0.086	0.582	0.075	0.488	0.063	0.467	0.065
DQN	0.399	0.204	0.491	0.032	0.361	0.035	0.332	0.060
PPO	0.512	0.107	0.422	0.107	0.441	0.096	0.472	0.096
REINFORCE	0.813	0.119	0.421	0.079	0.495	0.090	0.609	0.101

Table 7: Model performance summary - averaged over PHM-2010 environments with complex multivariate environment.

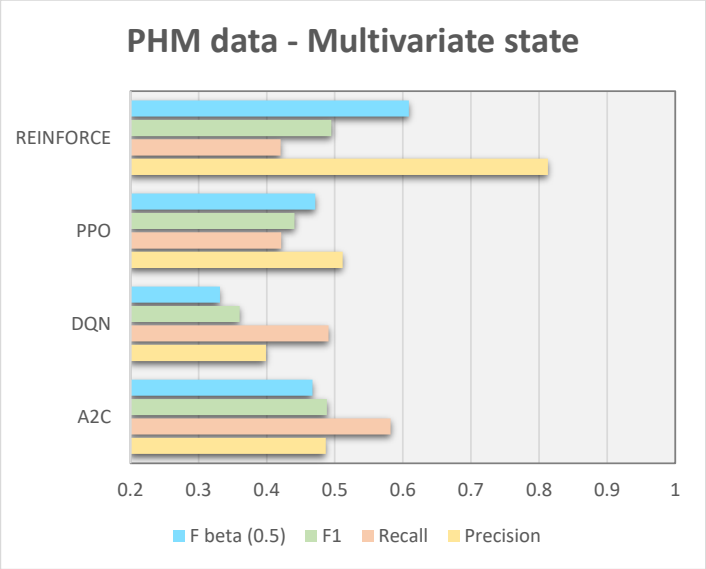


Figure 17: PHM real data - Model performance summary for the complex multivariate environment.

4.4. Real data - complex multivariate state

This environment offers the highest difficulty. We use real data, from *multiple* sensors. As with other scenarios, we used untuned, default settings of all algorithms. In Table 6, we see that the REINFORCE has the poorest recall at 0.421, however it demonstrates a surprisingly high tool replacement precision at 0.813, which drives the F1 and $F_\beta 0.5$ to the top of the table. $F_\beta 0.5$ performance is reflected in the Fig. 14; where it remains high for most rounds. The plot, in Appendix A, Fig. 24 shows REINFORCE's high precision behavior. However, it is important to note that the error-bars are occasionally larger (rounds 0, 2, 4 and 6). The recall seemed higher at some points but lower half the times and this manifests in the F1 behavior.

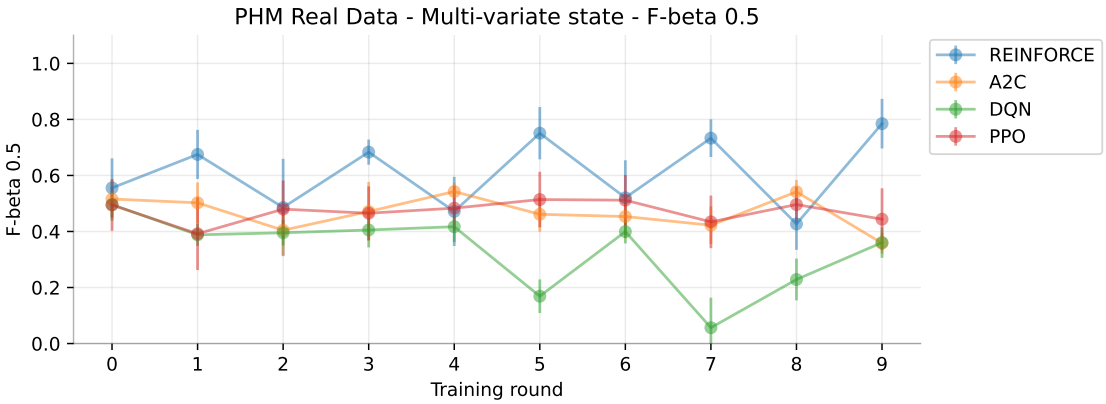


Figure 18: Multivariate complex state environments - $F_\beta 0.5$ metric, over 10 rounds of training and testing.

4.5. Super models

Finally, we look at performance of "select" models. Logically, one would choose the best model from a set of 10 trained models, just as an AutoML or AutoRL framework would. Selecting models based on the highest F1 with a *minimum* satisfying criteria for precision and recall allowed us to evaluate performance of "super-models" for each algorithm. At an overall averaged level Table 8 demonstrates the remarkable performance of the simple REINFORCE algorithm. It does show a lower recall when compared to the DQN, however it is a much more *balanced* model on an overall basis. Appendix A Table 11 shows details where the REINFORCE performs better than the other three algorithms by a huge margin, for 14 of the 15 variants. For PHM C06, univariate environment, it was the DQN that performed best, with extremely high metrics throughout and an F1 of 0.969 to 0.831 of REINFORCE.

	Precision		Recall		F1-score		$F_\beta 0.5$	
	Mean	SD	Mean	SD	Mean	SD	Mean	SD
A2C	0.520	0.031	0.859	0.053	0.639	0.036	0.560	0.032
DQN	0.651	0.022	0.937	0.031	0.740	0.022	0.678	0.021
PPO	0.558	0.076	0.643	0.097	0.580	0.079	0.562	0.075
REINFORCE	0.884	0.042	0.884	0.042	0.873	0.034	0.876	0.036

Table 8: Super models: Best of 10 rounds; performance averaged over all 15 environments.

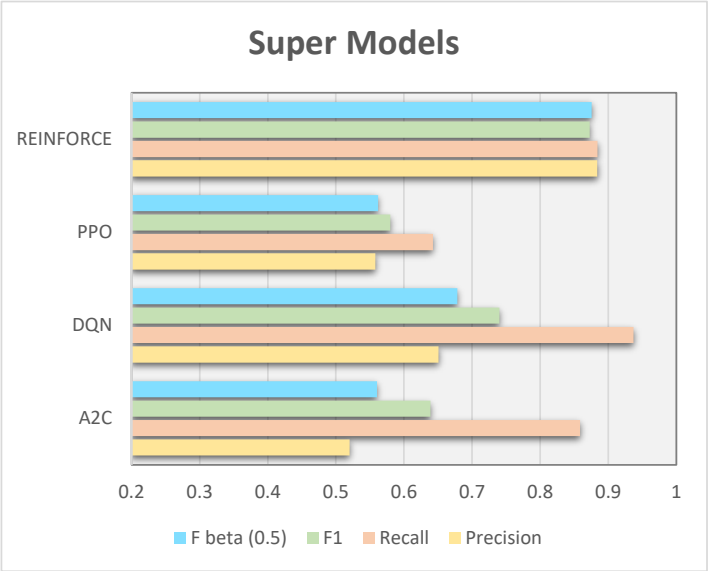


Figure 19: Super models: Best of 10 rounds; performance averaged over all 15 environments.

4.6. Hypothesis testing

We undertake a statistical analysis of the results, postulated by the hypothesis (13). Table 9 shows the result of a one-sided, two-sample t-test, conducted for a significance level $\alpha = 0.05$. Sample size of the test are mentioned against each category[6].

$$\left.\begin{aligned} H_0 : \mu_{RF} - \mu_{AA} = 0, \\ H_a : \mu_{RF} - \mu_{AA} > 0, \end{aligned}\right\} \quad \forall \;\; AA \in [A2C, DQN, PPO] \tag{13}$$

We see that the p-values are extremely low and test statistic positive, for all cases except for the *recall* in the PHM multivariate state case. We can thus reject H_0.

4.7. Training times

Our final set of results are related to the training time required for each algorithm. The naïve REINFORCE algorithm is extremely slow, with a very high variance in training time, as seen in Fig. 20.

[6]What does a single data point indicate?: Consider the "Simulated" category – 10 training rounds × 10 test rounds × 3 noise settings = 300 sample points. Note that a single test round is conducted with 40 randomly sampled wear datapoints and therefore 300 samples = 300 ×40 i.e. 12,000 samples of wear data. Similarly, for the real-data univariate state: 3 datasets × 10 training rounds × 10 test rounds ×3 noise settings = 900 sample points.

Metric	p Value			t Statistic		
	RF $\underset{H_0}{\overset{H_a}{\geqq}}$ A2C	RF $\underset{H_0}{\overset{H_a}{\geqq}}$ DQN	RF $\underset{H_0}{\overset{H_a}{\geqq}}$ PPO	RF $\underset{H_0}{\overset{H_a}{\geqq}}$ A2C	RF $\underset{H_0}{\overset{H_a}{\geqq}}$ DQN	RF $\underset{H_0}{\overset{H_a}{\geqq}}$ PPO
Overall (1500 samples)						
Precision	4.31E-126	2.17E-109	2.81E-106	25.071	23.170	22.804
Recall	4.20E-35	3.37E-16	4.36E-150	12.522	8.206	27.650
F1 score	1.99E-64	1.46E-88	5.29E-155	17.364	20.634	28.160
Simulated environment (300 samples)						
Precision	3.20E-98	1.69E-63	2.65E-81	25.611	19.032	22.427
Recall	8.12E-104	2.56E-41	1.57E-264	26.665	14.558	62.541
F1 score	9.60E-134	8.56E-99	2.96E-242	32.402	25.719	56.575
PHM Real data - Simple univariate state (900 samples)						
Precision	2.27E-32	7.29E-31	9.95E-31	12.082	11.770	11.742
Recall	1.27E-16	1.55E-06	8.19E-71	8.357	4.821	18.607
F1 score	1.94E-19	4.67E-34	2.19E-67	9.121	12.423	18.098
PHM Real data - Complex multivariate state (300 samples)						
Precision	1.64E-60	3.34E-54	7.88E-59	18.451	17.207	18.122
Recall	2.69E-10	2.69E-02	9.68E-01	-6.425	-2.219	-0.041
F1 score	7.27E-01	1.44E-08	1.35E-03	0.349	5.748	3.220

Table 9: Statistical test: One-sided two-sample t-tests. $H_0 : \mu_{RF} - \mu_{AA} = 0; H_a : \mu_{RF} - \mu_{AA} > 0$, where $AA$is one of A2C, DQN or PPO

5. Discussion

In this study we use the raw data, with no feature engineering. This is in tune with the "untuned" algorithms concept. Sophisticated feature engineering techniques exist, allowing for superior results; for example for a similar milling machine PdM problem, Sayyad et al. (2023) used time and frequency based techniques such as Fourier and wavelet transforms. With AutoRL one could automate such feature engineering techniques, followed by automated selection of significant features.

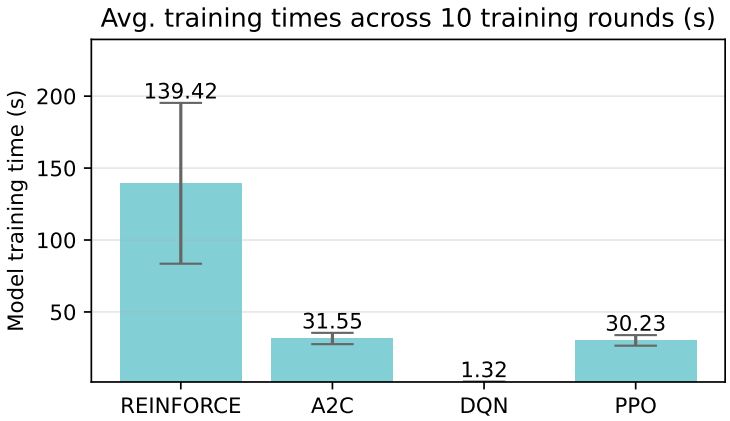


Figure 20: Training time, averaged over 10 rounds and all 15 variants.

The inventor of the REINFORCE algorithm, in his original paper Williams (1992), mentions the inability to predict the convergent and asymptotic properties of REINFORCE algorithms using analytical means. The author mentions that *simulation studies* are the primary source of understanding its behavior. 27 years later, Peck and Renaux (2019) notes that rigorous convergence proofs are *not* available for many algorithms based on REINFORCE and in their paper provide an intuition for convergence properties. This supports the extensive experimental studies on REINFORCE performed by Zhang et al. (2021a,b); Peck and Renaux (2019); Duan et al. (2016) as well our empirical research.

Both Duan et al. (2016) and Zhang et al. (2021b) studied the behavior on multiple OpenAI Gym environments, while Peck and Renaux (2019) provide empirical results using MuJuCo. Duan et al. (2016) observe that, despite its simplicity it is an effective algorithm in optimizing policies based on deep neural networks. On the other hand, their empirical study concluded that REINFORCE is occasionally trapped in premature convergence to local optima. REINFORCE suffers from high variance (Peck and Renaux, 2019) and converges very slowly when it does. High variance implies that a large sample of environment interactions are needed to obtain a reasonable approximation of the actual gradient.

Zhang et al. (2021a) provide the first known global convergence and sample efficiency results for the REINFORCE algorithm. Their study involved controlling the number of "bad" episodes as well as applying a "doubling" trick for studying the global convergence. Interestingly our training rounds resulted in multiple situations of "zero" performance ("bad"

episodes) for REINFORCE (18 instances) as also for Stable-Baseline's A2C implementation (12 instances); DQN and PPO did not show any occurrences.

There could be two possible reasons for the positive performance demonstrated by REINFORCE. We used ReLU (rectified linear unit) as the activation function, while the other three algorithms used hyperbolic tangent (Tanh). Henderson et al. (2018) studied four policy-gradient methods, including the PPO and TRPO, and observed that ReLU activation performs best. Duan et al. (2016) observed that *even* with a small learning rate REINFORCE sometimes resulted in large changes to policy distribution, which possibly explains the fast convergence to local optima. On the other hand we used a larger learning rate (almost 50:1) and this could have assisted in reaching global optimas when it did.

RL algorithms are incredibly sensitive to hyperparameters and the neural network architectures they employ. Working implementations need several decisions to be made during high-, as well as low- level design; these strongly impact the performance, (Henderson et al., 2018; Andrychowicz et al., 2021). In-fact the same algorithm could produce very different results depending on the code base one uses, (Henderson et al., 2018; Raffin et al., 2021). These research pointers could additionally be the reason for the superior performance of the REINFORCE, but more significantly they collectively must be considered before AutoRL platforms become successful.

While there is no doubt that this REINFORCE naïve implementation will be unstable and occasionally produce learning patterns that are stuck in local optimas and demonstrate premature convergence, our empirical studies indicate that it did perform well for this problem and at the least deserves further research.

On the evaluation criteria: It must be noted that the classification metrics we used assume that the human preventive maintenance policy, based on experience, is ideal. In practice this might not always be the case. The RL agent is trained to replace the tool optimally i.e. maximize use before reaching the threshold wear, and replace only when necessary so as to maintain lowest possible cost of replacement. In our dataset we assume the threshold to be ideally set and base the "human" action against that, rather than a real human action, which incidentally is not available for the PHM dataset anyway.

6. Conclusion

The REINFORCE is a simple algorithm and our naïve implementation can be improved drastically, for example with the implementation of a baseline to reduce variance. We evaluated the REINFORCE along with industry grade implementations of more advanced algorithms, DQN, A2C and PPO, over 15 environment variants (simulated and real data). Despite its simplicity, known variance and convergence issues, it performed surprisingly well, as observed through numerical results, plots and statistical tests.

Implementing robust RL algorithms is complex and Raffin et al. (2021) puts this beautifully, mentioning how small implementation details can significantly impact performance *"that is often greater than the difference between the algorithms themselves"*.

We hope our research contributes to empirical evidence related to the application of REINFORCE to predictive maintenance. This contribution aims to play a vital role in advancing the prospects of automated reinforcement learning i.e. AutoRL to Industry 4.0 revolution.

Appendix A Detailed results

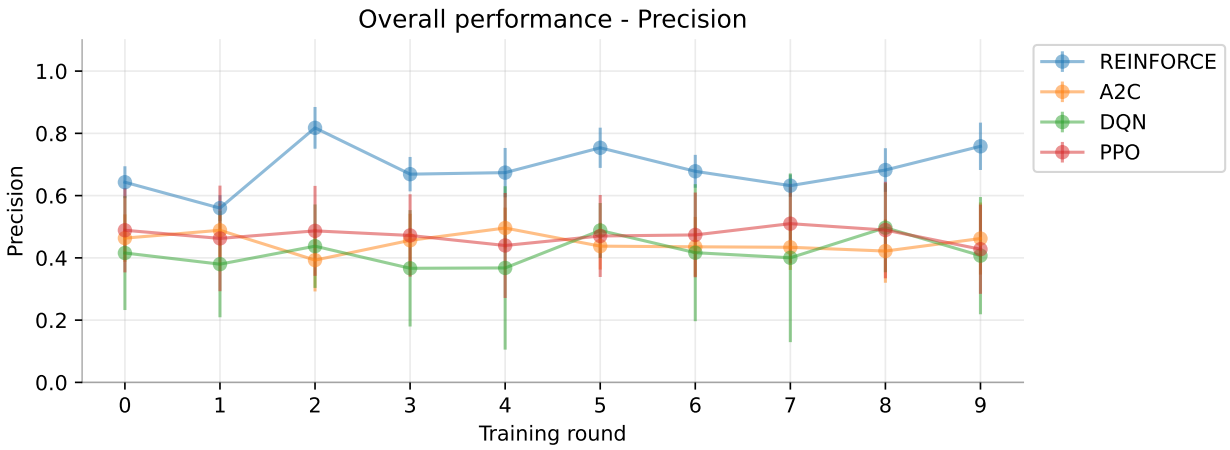
The Appendix contains detailed tabulated results as well as validation plots.

	REINFORCE				A2C				DQN				PPO		
Environment	Prec.	Recall	F1	$F_\beta 0.5$	Prec.	Recall	F1	$F_\beta 0.5$	Prec.	Recall	F1	$F_\beta 0.5$	Prec.	Recall	F1
Simulated - No noise	0.842	0.878	0.838	0.834	0.424	0.451	0.423	0.421	0.426	0.674	0.471	0.410	0.504	0.200	0.271
Simulated - Low noise	0.777	0.929	0.834	0.796	0.465	0.423	0.409	0.427	0.421	0.338	0.270	0.283	0.482	0.236	0.296
Simulated - High noise	0.798	0.940	0.851	0.816	0.358	0.281	0.256	0.272	0.447	0.519	0.380	0.360	0.514	0.207	0.286
PHM C01 SS - No noise	0.478	0.363	0.400	0.439	0.501	0.500	0.493	0.496	0.472	0.807	0.568	0.490	0.440	0.417	0.387
PHM C01 SS - Low noise	0.507	0.311	0.332	0.383	0.503	0.598	0.535	0.513	0.393	0.502	0.351	0.317	0.522	0.338	0.388
PHM C01 SS - High noise	0.693	0.562	0.579	0.623	0.266	0.282	0.267	0.262	0.458	0.525	0.400	0.384	0.456	0.369	0.372
PHM C04 SS - No noise	0.751	0.878	0.784	0.757	0.487	0.442	0.449	0.463	0.439	0.684	0.472	0.411	0.500	0.510	0.469
PHM C04 SS - Low noise	0.662	0.756	0.672	0.657	0.409	0.455	0.428	0.416	0.411	0.500	0.370	0.341	0.488	0.280	0.324
PHM C04 SS - High noise	0.611	0.713	0.620	0.598	0.518	0.607	0.552	0.530	0.358	0.451	0.325	0.294	0.428	0.262	0.286
PHM C06 SS - No noise	0.830	0.726	0.754	0.792	0.517	0.509	0.507	0.511	0.360	0.309	0.256	0.258	0.409	0.248	0.275
PHM C06 SS - Low noise	0.205	0.279	0.228	0.212	0.510	0.577	0.530	0.516	0.434	0.266	0.266	0.296	0.417	0.181	0.232
PHM C06 SS - High noise	0.709	0.843	0.759	0.726	0.316	0.324	0.311	0.308	0.449	0.518	0.400	0.375	0.388	0.222	0.265
PHM C01 MS - No noise	0.835	0.652	0.656	0.716	0.461	0.444	0.397	0.404	0.384	0.558	0.393	0.348	0.513	0.383	0.416
PHM C04 MS - No noise	0.739	0.255	0.359	0.494	0.498	0.589	0.490	0.470	0.323	0.209	0.160	0.168	0.499	0.393	0.421
PHM C06 MS - No noise	0.864	0.356	0.469	0.616	0.501	0.713	0.578	0.527	0.489	0.705	0.529	0.479	0.523	0.488	0.485

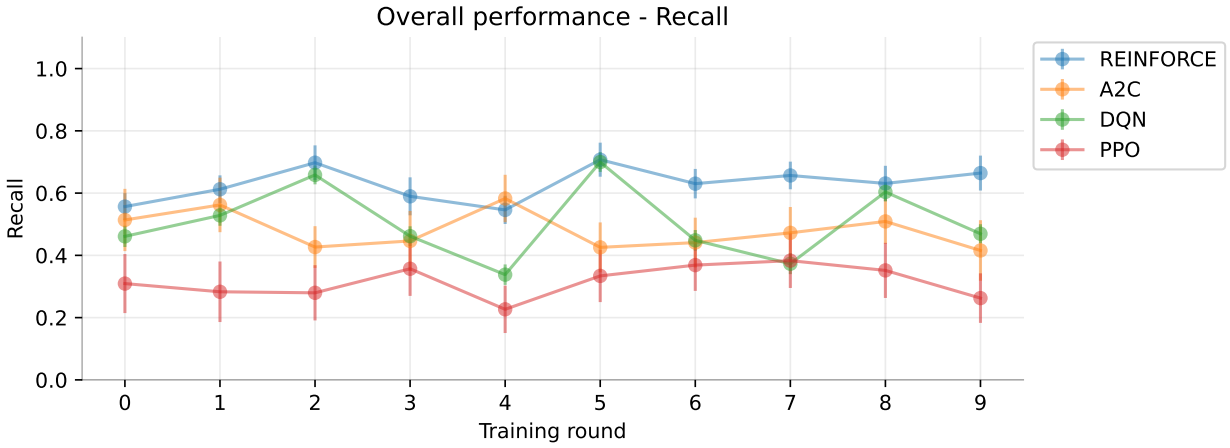
Table 10: Model performance comparison all variants of the environments, over 10 rounds of training. Maximum values indicated in blue.

	REINFORCE				A2C				DQN				PPO		
Environment	Prec.	Recall	F1	$F_\beta 0.5$	Prec.	Recall	F1	$F_\beta 0.5$	Prec.	Recall	F1	$F_\beta 0.5$	Prec.	Recall	F1
Simulated - No noise	0.897	0.960	0.926	0.908	0.500	1.000	0.667	0.556	0.505	0.980	0.667	0.560	0.669	0.430	0.518
Simulated - Low noise	0.960	0.945	0.952	0.957	0.516	1.000	0.680	0.571	0.500	0.980	0.662	0.554	0.633	0.460	0.530
Simulated - High noise	0.922	0.990	0.955	0.935	0.503	1.000	0.669	0.558	0.504	0.990	0.668	0.559	0.569	0.355	0.434
PHM C01 SS - No noise	0.889	0.995	0.939	0.908	0.586	0.625	0.603	0.592	0.647	0.970	0.776	0.693	0.543	1.000	0.703
PHM C01 SS - Low noise	0.988	0.765	0.861	0.932	0.499	0.995	0.664	0.554	0.504	0.990	0.668	0.559	0.623	0.740	0.675
PHM C01 SS - High noise	0.850	0.970	0.905	0.871	0.521	0.680	0.588	0.546	0.505	0.985	0.668	0.560	0.520	0.725	0.604
PHM C04 SS - No noise	0.811	1.000	0.895	0.842	0.536	0.645	0.583	0.554	0.501	0.965	0.660	0.554	0.579	0.895	0.702
PHM C04 SS - Low noise	0.798	0.980	0.879	0.829	0.556	0.665	0.603	0.573	0.734	0.990	0.843	0.774	0.546	0.660	0.596
PHM C04 SS - High noise	0.708	0.840	0.767	0.730	0.521	0.835	0.641	0.563	0.511	0.985	0.672	0.565	0.517	0.820	0.633
PHM C06 SS - No noise	1.000	0.895	0.944	0.977	0.520	0.680	0.587	0.545	0.935	0.975	0.954	0.942	0.587	0.650	0.615
PHM C06 SS - Low noise	0.943	0.795	0.861	0.908	0.501	1.000	0.668	0.557	0.961	0.725	0.826	0.901	0.552	0.370	0.438
PHM C06 SS - High noise	0.821	0.845	0.831	0.825	0.540	0.755	0.628	0.572	0.980	0.960	0.969	0.976	0.521	0.615	0.564
PHM C01 MS - No noise	0.827	0.995	0.903	0.856	0.500	1.000	0.667	0.556	0.505	0.985	0.668	0.560	0.512	0.595	0.549
PHM C04 MS - No noise	0.910	0.425	0.577	0.738	0.500	1.000	0.667	0.556	0.501	0.975	0.662	0.555	0.501	0.635	0.558
PHM C06 MS - No noise	0.934	0.865	0.896	0.918	0.500	1.000	0.667	0.556	0.969	0.600	0.741	0.863	0.497	0.690	0.577

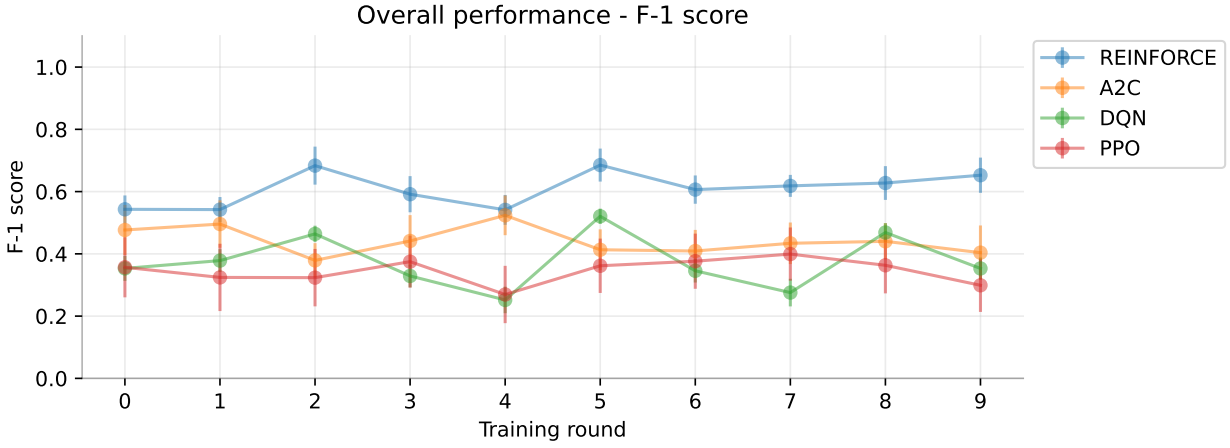
Table 11: Super Models: Best models selected over 10 rounds of training. Maximum performance values indicated in blue.



(a) Precision

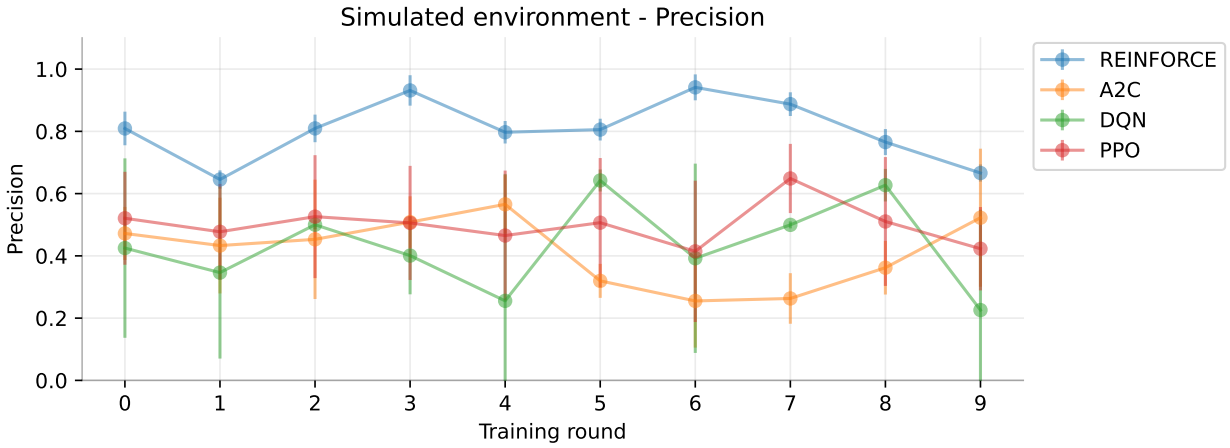


(b) Recall

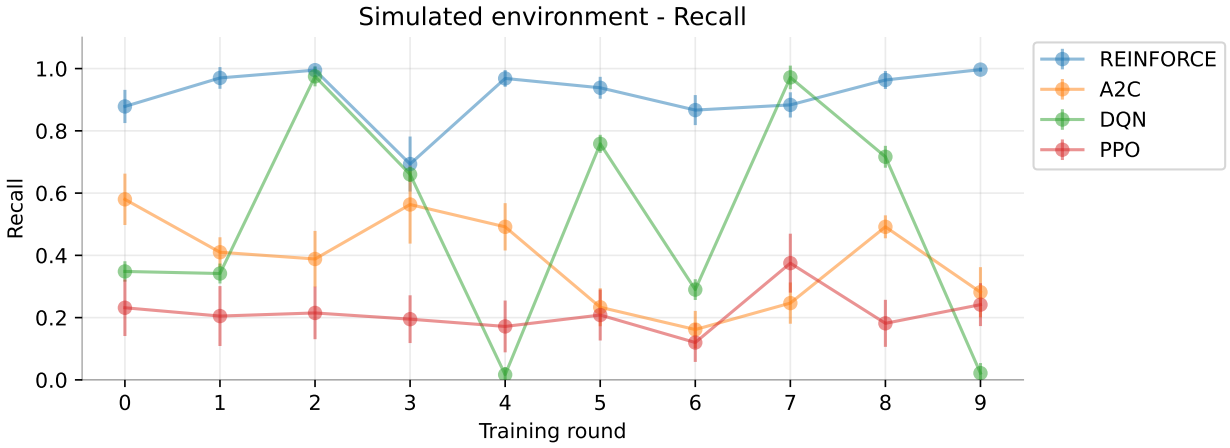


(c) F1-score

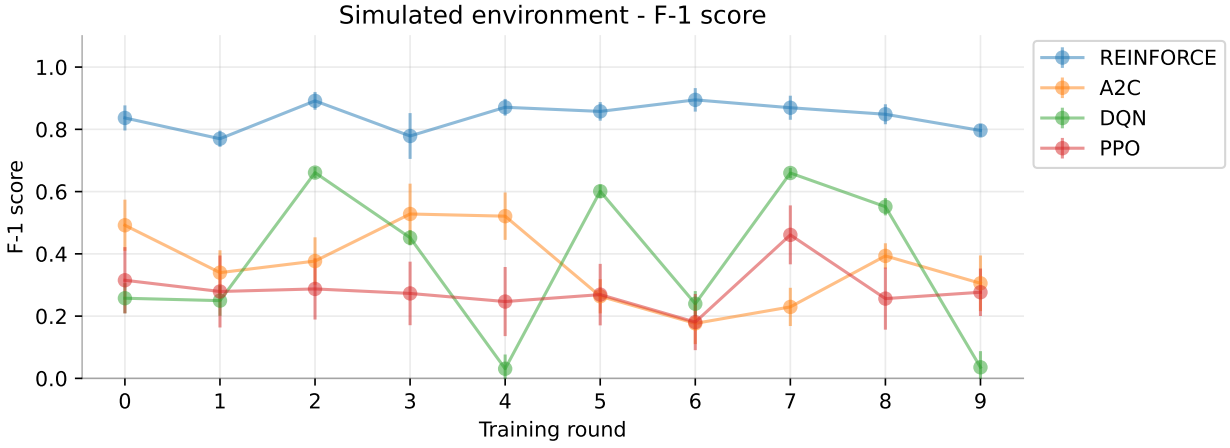
Figure 21: Overall performance – Average over 10 models



(a) Precision

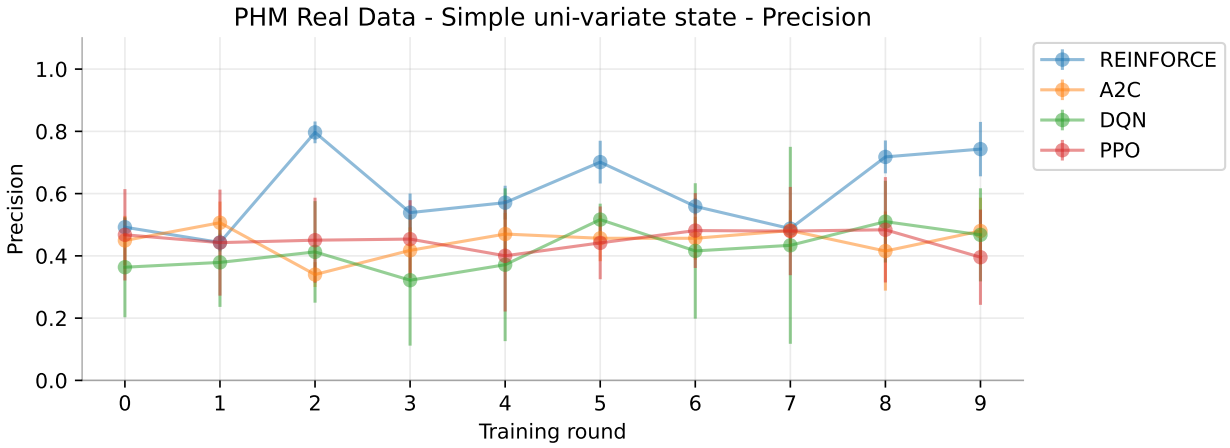


(b) Recall

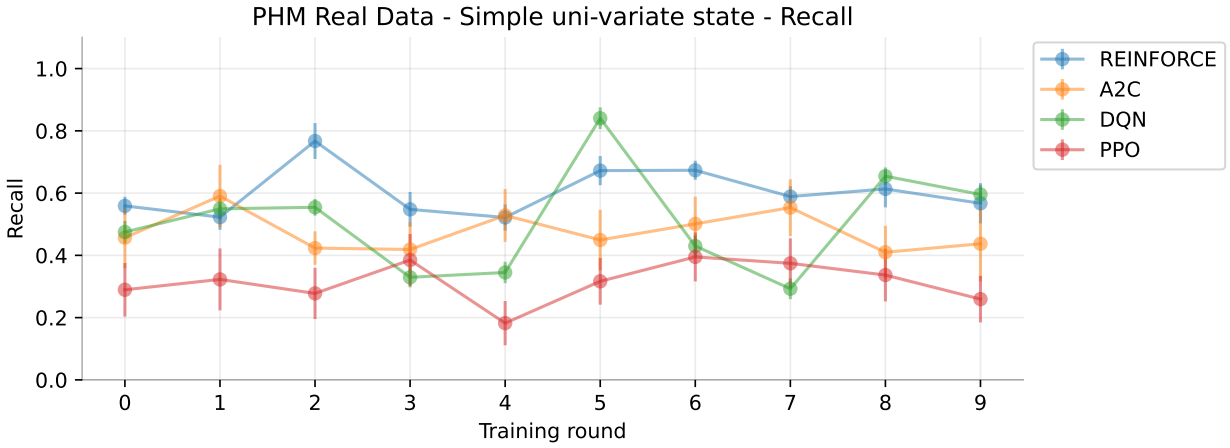


(c) F1-score

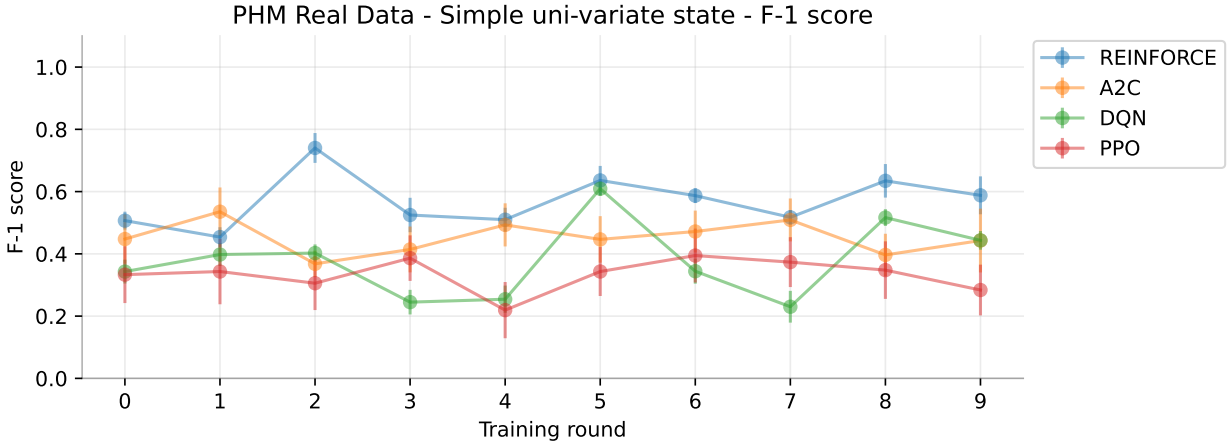
Figure 22: Simulated environment – Average over 10 models



(a) Precision

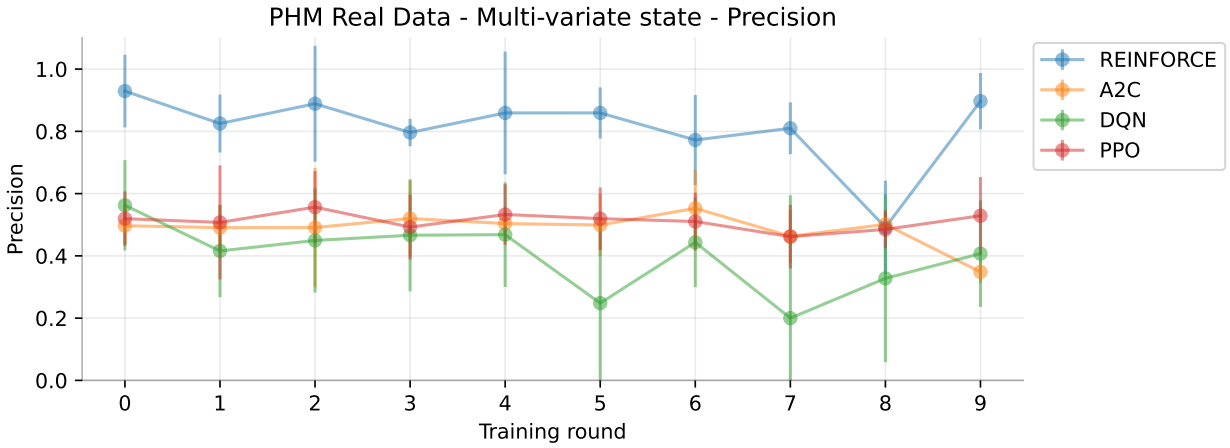


(b) Recall



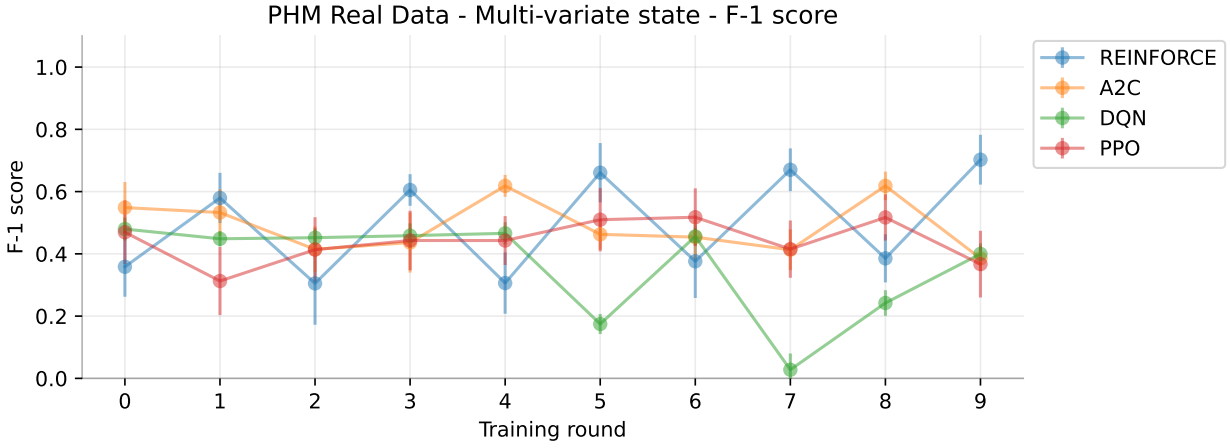
(c) F1-score

Figure 23: Univariate state environment – Average over 10 models



(a) Precision

(b) Recall



(c) F1-score

Figure 24: Multivariate state environment – Average over 10 models

References

Afshar, R.R., Zhang, Y., Vanschoren, J., Kaymak, U., 2022. Automated reinforcement learning: An overview. arXiv preprint .

Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., et al., 2021. What matters in on-policy reinforcement learning? a large-scale empirical study, in: ICLR 2021-Ninth International Conference on Learning Representations.

Dašić, P., 2006. Analysis of wear cutting tools by complex power-exponential function for finishing turning of the hardened steel 20crmo5 by mixed ceramic tools. Fascicle VIII Tribology 12, 54–60.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., Abbeel, P., 2016. Benchmarking deep reinforcement learning for continuous control, in: International conference on machine learning, PMLR. pp. 1329–1338.

Dulac-Arnold, G., Levine, N., Mankowitz, D.J., Li, J., Paduraru, C., Gowal, S., Hester, T., 2020. An empirical investigation of the challenges of real-world reinforcement learning. arXiv preprint arXiv:2003.11881 .

Dulac-Arnold, G., Levine, N., Mankowitz, D.J., Li, J., Paduraru, C., Gowal, S., Hester, T., 2021. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. Machine Learning 110, 2419–2468.

Erhan, L., Ndubuaku, M., Di Mauro, M., Song, W., Chen, M., Fortino, G., Bagdasar, O., Liotta, A., 2021. Smart anomaly detection in sensor systems: A multi-perspective review. Information Fusion 67, 64–79. doi:10.1016/j.inffus.2020.10.001.

Ford, S., Ritchie, M., 2022. Cognitive radar mode control: a comparison of different reinforcement learning algorithms, in: International Conference on Radar Systems (RADAR 2022), IET. pp. 107–112.

Future Market Insights, 2023. Milling machine market outlook (2023 to 2033). URL: https://www.futuremarketinsights.com/reports/milling-machine-market. accessed: 2023-06-23.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D., 2018. Deep reinforcement learning that matters, in: Proceedings of the AAAI conference on artificial intelligence.

Li, X., 2021. 2010 phm society conference data challenge. URL: https://dx.doi.org/10.21227/jdxd-yy51, doi:10.21227/jdxd-yy51.

Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning, in: International conference on machine learning, PMLR. pp. 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M., 2013. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 .

Panzer, M., Bender, B., 2021. Deep reinforcement learning in production systems: a systematic literature review. International Journal of Production Research .

Parker-Holder, J., Rajan, R., Song, X., Biedenkapp, A., Miao, Y., Eimer, T., Zhang, B., Nguyen, V., Calandra, R., Faust, A., et al., 2022. Automated reinforcement learning (autorl): A survey and open problems. Journal of Artificial Intelligence Research 74, 517–568.

Peck, R., Renaux, L., 2019. A review of reinforce algorithms .

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., Dormann, N., 2021. Stable-baselines3: Reliable reinforcement learning implementations. J. Mach. Learn. Res. 22.

Sandeep Varma, N., Sinha, V., Pradyumna Rahul, K., 2022. Experimental evaluation of reinforcement learning algorithms, in: International Conference on Computational Intelligence and Data Engineering, Springer. pp. 469–484.

Sayyad, S., Kumar, S., Bongale, A., Kotecha, K., Abraham, A., 2023. Remaining useful-life prediction of the milling cutting tool using time-frequency-based features and deep learning models. Sensors 23. URL: https://www.mdpi.com/1424-8220/23/12/5659, doi:10.3390/s23125659.

SB3-Algorithms, 2022. Stable-baselines3 - master list of algorithms. URL: https://stable-baselines3.readthedocs.io/en/master/guide/algos.html. accessed: 2023-06-27.

SB3-Default Network Architecture, 2022. Stable-baselines3 - default network architecture. URL: https://stable-baselines3.readthedocs.io/en/master/guide/custom_policy.html#default-network-architecture. accessed: 2023-06-27.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P., 2015. Trust region policy optimization, in: International conference on machine learning, PMLR. pp. 1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O., 2017. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 .

Shala, G., Arango, S.P., Biedenkapp, A., Hutter, F., Grabocka, J., 2022. AutoRL-bench 1.0, in: Sixth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems. URL: https://openreview.net/forum?id=RyAl6OVhTcG.

Siraskar, R., Kumar, S., Patil, S., Bongale, A., Kotecha, K., 2023. Reinforcement learning for predictive maintenance: a systematic technical review. Artificial Intelligence Review , 1–63.

Sutton, R., Barto, A., 2018. Reinforcement Learning: An Introduction. 2nd. edition ed., The MIT Press, Cambridge, England.

Velivela, K., Yarram, S., 2020. Comparison of reinforcement learning algorithms.

Williams, R.J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Reinforcement learning , 5–32.

Zhang, J., Kim, J., O'Donoghue, B., Boyd, S., 2021a. Sample efficient reinforcement learning with reinforce, in: Proceedings of the AAAI conference on artificial intelligence, pp. 10887–10895.

Zhang, J., Ni, C., Szepesvari, C., Wang, M., et al., 2021b. On the convergence and sample efficiency of variance-reduced policy gradient method. Advances in Neural Information Processing Systems 34, 2228–2240.