An empirical study of the naïve REINFORCE algorithm for predictive maintenance of industrial milling machines

Rajesh Siraskar

August 1, 2023

Abstract

Industrial systems tend to be highly complex and dynamic. Reinforcement learning (RL) offers a mechanism for creating optimal predictive maintenance policies for such systems. RL is known to be an extremely complex exercise in hyper-parameter tuning. Automated machine learning (AutoML), applied to the combined fields of predictive maintenance (PdM) and RL, has yet to be studied. This article is an empirical study aimed at industrial practitioners unfamiliar with complex RL tuning. We study the effects of *untuned* RL algorithms for generating an optimal tool replacement policy for a milling machine. We compare a naïve implementation of REINFORCE against the policies of industry-grade implementations of three advanced algorithms, namely, Deep Q-Network (DQN), Advantage Actor-Critic (A2C), and Proximal Policy Optimization (PPO). Our broad goal was to study model performance under various scenarios: (1) simulated tool-wear data, (2) real tool-wear data (benchmark IEEE NUAA Ideahouse dataset), (3) univariate state with added noise levels and a random chance of break-down (4) complex multivariate state.

Performance was measured by how accurately the predictive maintenance agent suggested tool replacement compared to a deterministic preventive maintenance rule (based on the tool-wear threshold). Across 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 by 0.08 and 0.016 compared to the average of the three advanced algorithms. 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 AutoML for PdM using RL, 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/Link

Keywords: Predictive maintenance, milling machines, Reinforcement Learning, REINFORCE

Abbreviations

A2C	Advantage Actor-Critic	DQN	Deep Q-Network
PPO	Proximal Policy Optimization	RF	REINFORCE
SS	Single-variable state	MS	Multivariate state
TP	True positive	TN	True negative
FP	False positive	FN	False negative
RL	Reinforcement Learning	SB3	Stable-Baselines3
PdM	Predictive maintenance	PHM	The Prognostics and Health Management Society

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. With the 2023 milling machine market valued at USD 68.3 billion (Future Market Insights, 2023), this is an important goal for the industry. 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, good actions are "reinforced", 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 consists of sensors attached to the machine and related information such as job specifications, environment conditions etc.

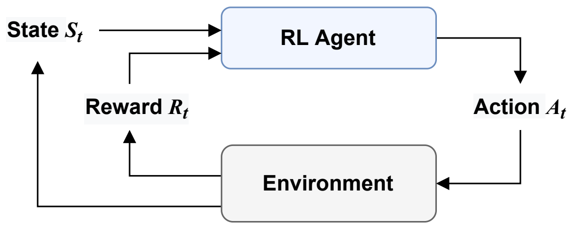


Figure 1: Reinforcement Learning

Introduced in 1992, the REINFORCE algorithm (Williams, 1992) is considered as a basic RL algorithm. It is a policy-based, on-policy algorithm, capable of handling both discrete and continuous observation and action domains. In practice the REINFORCE algorithm is considered as a "weak" algorithm and superseded by several algorithms developed since. Most notably the Q-Learning and its deep-neural network version, the DQN (Mnih et al., 2013), followed by Actor-Critic (Mnih et al., 2016) and one of the most robust modern-day algorithms, the PPO (Schulman et al., 2017), Fig. 2.

While most studies of RL algorithms are evaluated on Open AI Gym environments, our experiments cover the predictive maintenance problem using a custom built environment. Also, our study focuses on REINFORCE otherwise an often neglected algorithm since it considered an early, naïve algorithm.

In practice the milling tool is replaced after a set threshold. We use this "determin-

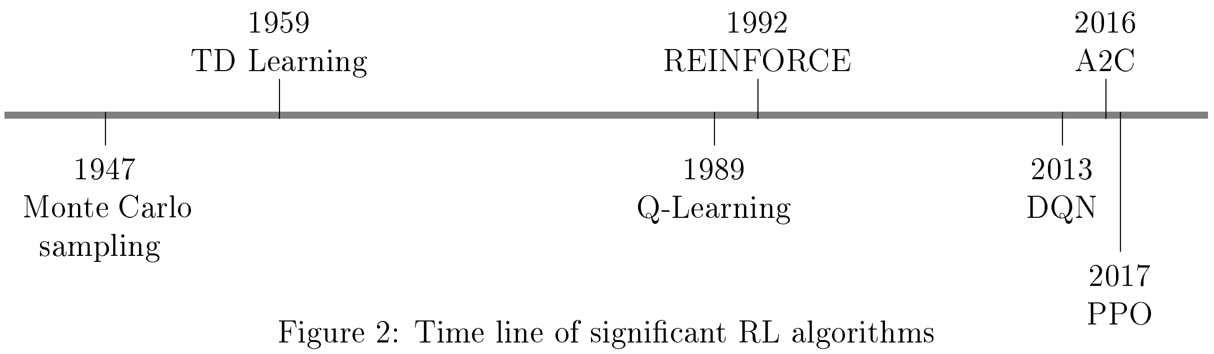


Figure 2: Time line of significant RL algorithms

istic 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, more robust, comparison of the algorithms. Finally, we conduct statistical tests to ensure a robust statistical-evidence based conclusion. Based on experiments we show that REINFORCE works surprisingly well on tool-replacement precision and F-beta (0.5), over all the variants. The recall and F1-score are better or at-par with the advanced algorithms.

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

1. Contributes to the broader goal of AutoML for PdM using RL.
2. Research targeted toward the industrial practitioner not accustomed to complex hyper-parameter tuning of RL algorithms.
3. Design and implement an RL environment for predictive maintenance of a milling machine.
4. Rigorous evaluation of four standard 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 predictive maintenance 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

2.1 Literature Review

Significant work has been conducted in the application of RL for predictive maintenance in general (Erhan et al., 2021; Panzer and Bender, 2021; Siraskar et al., 2023); however none of them covered[1] PdM for *milling machines*. Similarly, automated machine learning (AutoML) for predictive maintenance was presented in only 9 research articles and limited coverage to supervised machine learning – none of them studied RL.

Research work focusing of experimental comparison and analysis of various 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, SAC, 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. (2020, 2021) are significant contributions toward analyzing empirical studies directed toward *real-world* challenges. They apply real-world design concepts on the Cartpole and other complex environments such as humanoid and walkers from the Real-World Reinforcement Learning (RWRL) Suite[2]. These environments are then evaluated for multiple RL algorithms such as, REINFORCE, Trust Region Policy Optimization (TRPO) and Deep Deterministic Policy Gradient (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 on complex MuJoCo[3] environments such as the HalfCheetah, Hopper, Walker and Swimmer. Similar to our research, they used the OpenAI baseline implementations of RL algorithms for the experiments and evaluation. 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.

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

[1]Query: `(milling OR "tool wear") AND "reinforcement learning" AND maintenance`. As of: 10-Jul-2023, across IEEE Xplore™, Scopus™ and Web Of Science™

[2]Link to RWRL » link

[3]MuJoCo provids environments for studying Multi-Joint dynamics with Contact

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 replacement 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 a state-transition probabilities ($\mathcal{P}$) since RL is founded on 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 a brief overview of RL, we now briefly touch upon 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 and is sampled using a uniform distribution $U(D)$, $Q(s, a; \theta)$ is the function parameterized with θ, that helps compute the Q values and θ^- represents parameters of the frozen target Q-network.

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

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. The policy-gradient family of 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 main 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.

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

REINFORCE

The REINFORCE algorithm, invented by Williams (1992), is an early algorithm that directly learns a policy π_θ to produce action probabilities from states. Actions that cause favorable states, are positively reinforced thereby increasing their probability of occurrence. Conversely, those resulting in unfavorable states are penalized.

The objective function (7) that the agent attempts to maximize is defined as the expected return over many trajectories sampled from the policy.

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

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

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

3 Methodology

3.1 Implementation details

Reinforcement learning requires an environment to function. In this section we describe the custom milling environment we built, which allows our agent to learn a policy for tool replacement. Fig. 3 shows the stepped approach used to generate the *fifteen* environments, of varying complexity, to evaluate the agents. We first describe the approach used to model and simulate tool wear and hence generate wear data followed by a description of real tool wear data. We then describe the elements of environment design i.e. state space, action space and reward functions. The Python code and data used for this research is uploaded to GitHub at URL.

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 tool wear based **univariate state**. We then add two further levels of increased complexity using noise and a chance of breakdown. This gives 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[4]. The

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

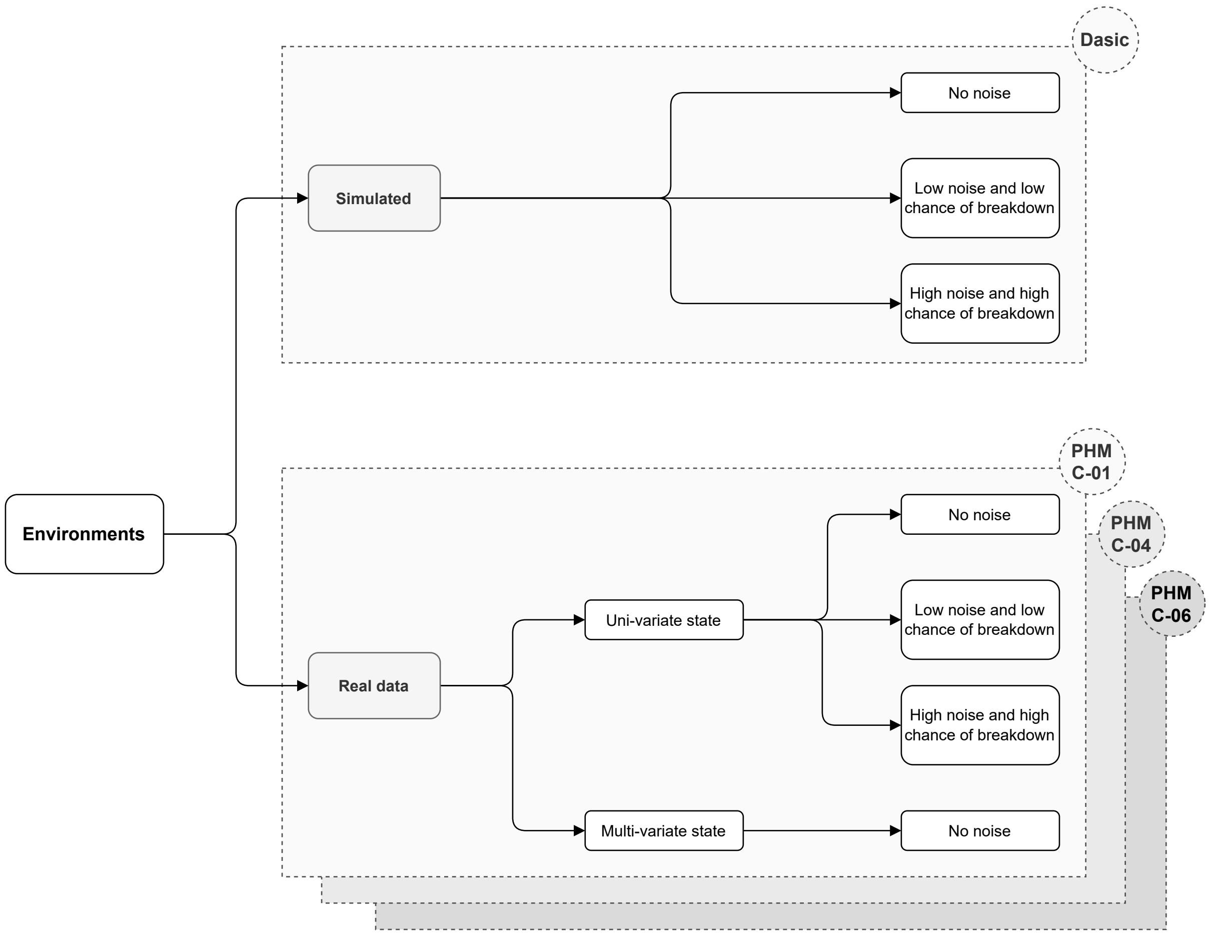

Dasic
Environments
Simulated
No noise
Low noise and low chance of breakdown
High noise and high chance of breakdown
PHM C-01
PHM C-04
PHM C-06
Real data
Uni-variate state
Multi-variate state
No noise
Low noise and low chance of breakdown
High noise and high chance of breakdown
No noise

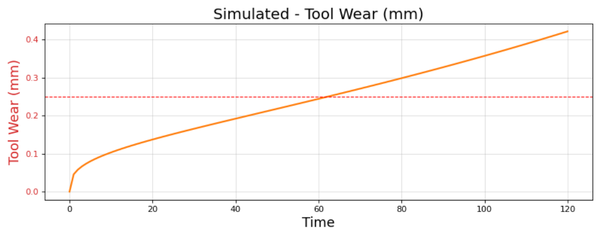
Figure 3: The fifteen different environments used for evaluation

data is from seven sensors – dynamometer measuring force (N) in X, Y and Z dimensions; accelerometer measuring vibration (g) in X, Y and Z dimensions and finally acoustic emission data as AE-RMS (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 the 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. As for the simulated case, the complexity of the *univariate* state is increased using two levels of noise and break-down parameters. The multivariate state is complex in itself, Fig. 5, and we use the natural (without noise) form.

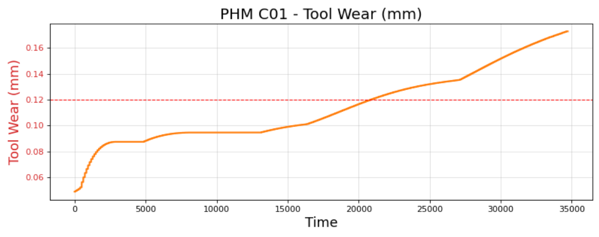
In RL state variables are often normalized to improve stability and convergence. Both the simulated and real data was normalized using min-max scaling such that the tool wear and other state features, $x \in [0,\ 1] \subset \mathbb{R}$. We will see next how this allows adding white noise of similar magnitudes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

Adding noise and chance of break-down

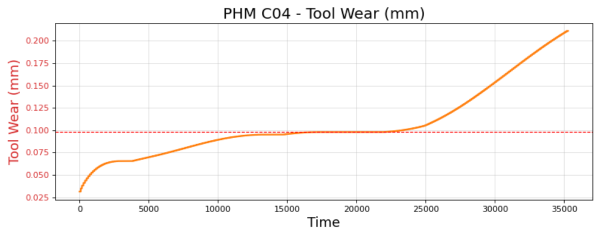
Fig. (6) shows the effect of adding two levels of noise. “Low noise” is obtained by adding Gaussian noise with an order of magnitude of -3 i.e. between $[0.0, 0.001]$ and



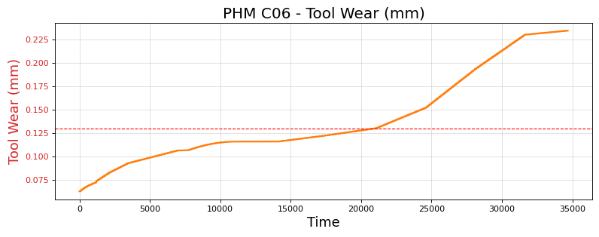
(a) Simulated wear data



(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.

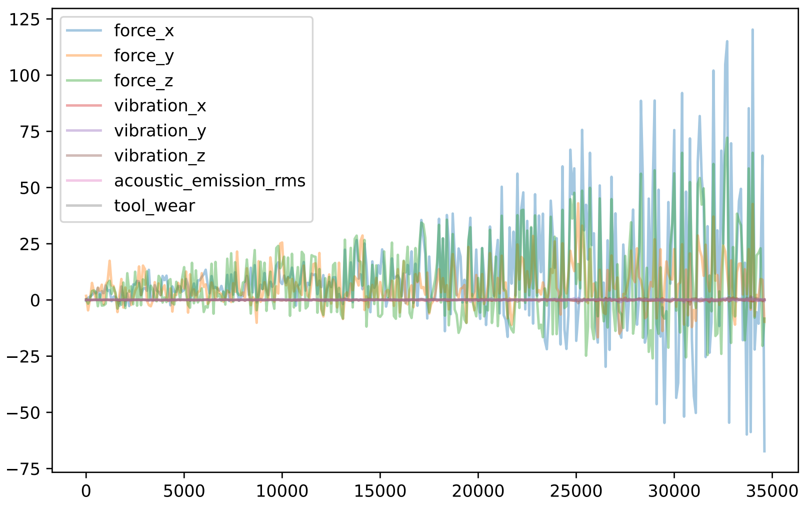
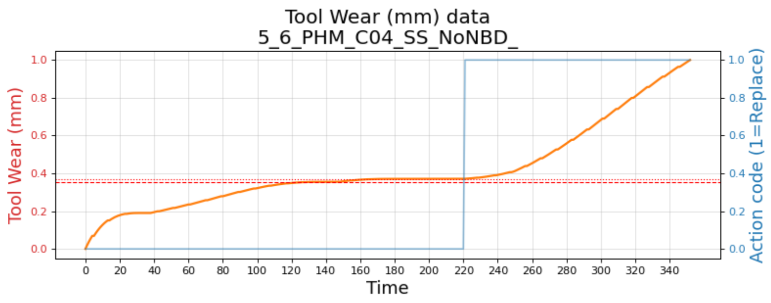


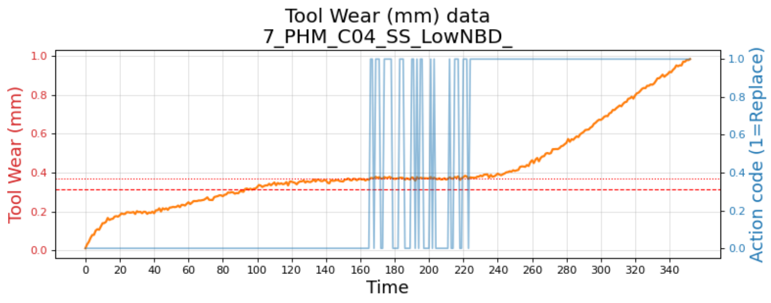
Figure 5: PHM C06: multivariate data

“high noise” is of order -2 i.e. between $[0.0, 0.01]$. The tool wear is less than 0.24 mm, this adds significant perturbations as seen in Fig. (6b) and (6c). The noise affects 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.

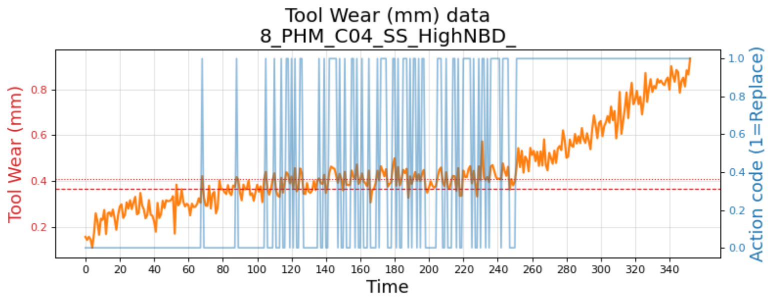
Break down occurs due to excessive tool use and can often occur randomly. In con-



(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.

junction 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; see line 9 of the code, Listing 1.

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 – simple 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

	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

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.

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, (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 choice of action is binary and represented by ACTION $\in [0, 1]$, where 0 represents continuation of milling operation (CONTINUE), while 1 represents the action of replacing the tool (REPLACE_TOOL).

Reward function

The reward function (10) drives the learning process. There are three important phases where reward (or penalty) is provided as feedback. Code implementation is seen in Listing 1. R_1, R_2 and R_3 are constants that determine the magnitude of reward. The first condition is when the current state of wear w_t is less than the threshold W_τ, and a desirable condition, hence the reward is positive. The formulation allows higher reward to be collected the closer it is to threshold, 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.

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

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 long-sightedness.

Environment feedback

Fig. 1 shows "feedback" that is important for the agent to learn from the action it takes. This is performed by the `step()` function in the environment code and the main parts are shown in Listing 1. At every time step an action is taken, the conditions (lines 2 and

8) of the resulting state evaluated for assigning a reward and then either proceeding or terminating the episode (lines 9 and 11).

```
 1 # Termination condition
 2 if self.ep_length >= self.max_operations:
 3   terminated = True
 4   self.reward = 0.0
 5   self.df_index = -1
 6   info = {'termination':'Max. milling operations crossed'}
 7
 8 elif (tool_wear > self.wear_threshold and
 9   np.random.uniform() < self.breakdown_chance):
10   terminated = True
11   self.reward = 0.0
12   self.df_index = -1
13   info = {'termination':'Tool breakdown'}
14
15 else:
16   terminated = False
17   info = {'action':'Continue'}
18   reward = 0.0
19
20   if tool_wear < self.wear_threshold:
21     reward += self.R1*self.df_index
22   else:
23     # Threshold breached. Farther away from threshold => more penalty
24     reward += self.R2*self.df_index
25
26   # Based on the action = 1 replace the tool or if 0, continue with
      normal operation
27   if action:
28     reward += self.R3
29     self.df_index = -1 # Tool replaced - so roll back tool life
30     self.ep_tool_replaced += 1 # Increment tool replaced count
31     info = {'action':'Tool replaced'}
```

Listing 1: Environment: Important implementation details of the 'step' function

Network architecture and basic hyper-parameters

Stable-Baselines3 (SB3) Raffin et al. (2021) provides robust open source implementations of many RL algorithms. As of 10-Jul-2023, 13 algorithms have been implemented (SB3-Algorithms, 2022) however REINFORCE has *not* been implemented. 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

	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

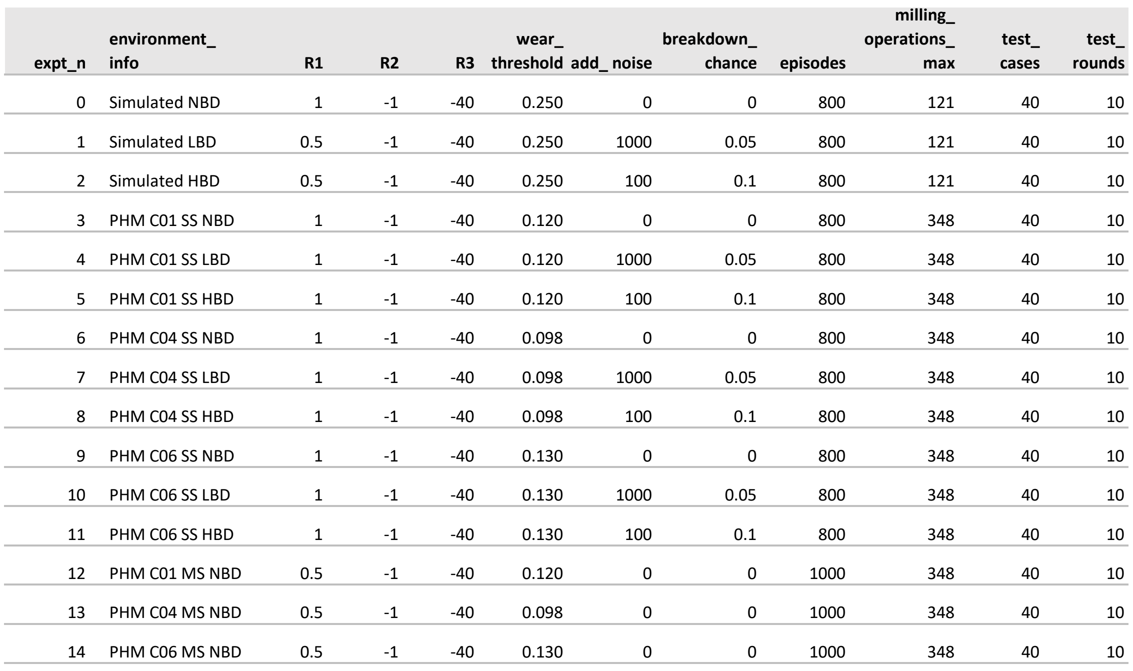
hyper-parameters. The REINFORCE uses a *single* internal layer of 64 units. The default architecture for all three SB3 algorithms (A2C/DQN/PPO) consists of *two* fully connected layers with 64 units per layer (SB3-Default Network Architecture, 2022). While REINFORCE used ReLU (rectified linear unit) as the activation function, the other three algorithms used hyperbolic tangent (Tanh). Finally, REINFORCE's learning-rate is also 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 reward function parameters R_1, R_2 and R_3; across all four algorithms during a single training round. Our code is fairly automated and we used a configuration file to configure the training experiments and record the test results. Fig. 7 shows the main columns of the configuration file we used.

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.

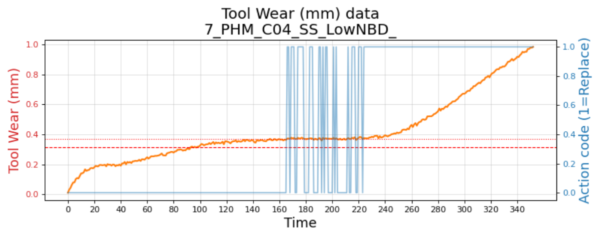
The REINFORCE was trained for 800 episodes for the simulated and PHM univariate variants, for all three noise and breakdown levels (none, low and high) – Table 1 items 1-12. For the PHM multivariate variant Table 1 items 13-15, REINFORCE was trained



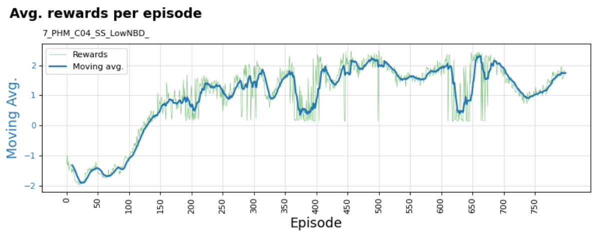
expt_n	environment_ info	R1	R2	R3	wear_ threshold	add_ noise	breakdown_ chance	episodes	milling_ operations_ max	test_ cases	test_ rounds
0	Simulated NBD	1	-1	-40	0.250	0	0	800	121	40	10
1	Simulated LBD	0.5	-1	-40	0.250	1000	0.05	800	121	40	10
2	Simulated HBD	0.5	-1	-40	0.250	100	0.1	800	121	40	10
3	PHM C01 SS NBD	1	-1	-40	0.120	0	0	800	348	40	10
4	PHM C01 SS LBD	1	-1	-40	0.120	1000	0.05	800	348	40	10
5	PHM C01 SS HBD	1	-1	-40	0.120	100	0.1	800	348	40	10
6	PHM C04 SS NBD	1	-1	-40	0.098	0	0	800	348	40	10
7	PHM C04 SS LBD	1	-1	-40	0.098	1000	0.05	800	348	40	10
8	PHM C04 SS HBD	1	-1	-40	0.098	100	0.1	800	348	40	10
9	PHM C06 SS NBD	1	-1	-40	0.130	0	0	800	348	40	10
10	PHM C06 SS LBD	1	-1	-40	0.130	1000	0.05	800	348	40	10
11	PHM C06 SS HBD	1	-1	-40	0.130	100	0.1	800	348	40	10
12	PHM C01 MS NBD	0.5	-1	-40	0.120	0	0	1000	348	40	10
13	PHM C04 MS NBD	0.5	-1	-40	0.098	0	0	1000	348	40	10
14	PHM C06 MS NBD	0.5	-1	-40	0.130	0	0	1000	348	40	10

Figure 7: Configuring the experiments

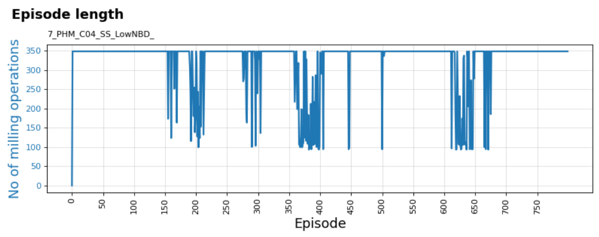
for 1000 episodes. SB3 algorithms were trained for 10,000 episodes for all variants. We ran ten rounds of training, tested each model generated and then averaged their results. Testing is explained in the next section while results are presented in Section 4.



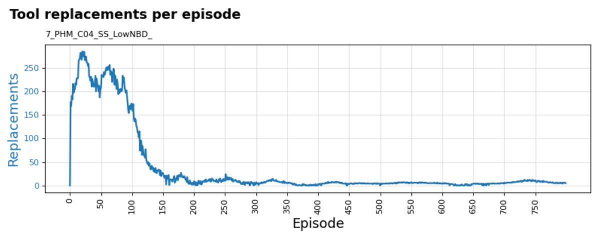
(a) PHM C04 wear data



(b) Average rewards per episode



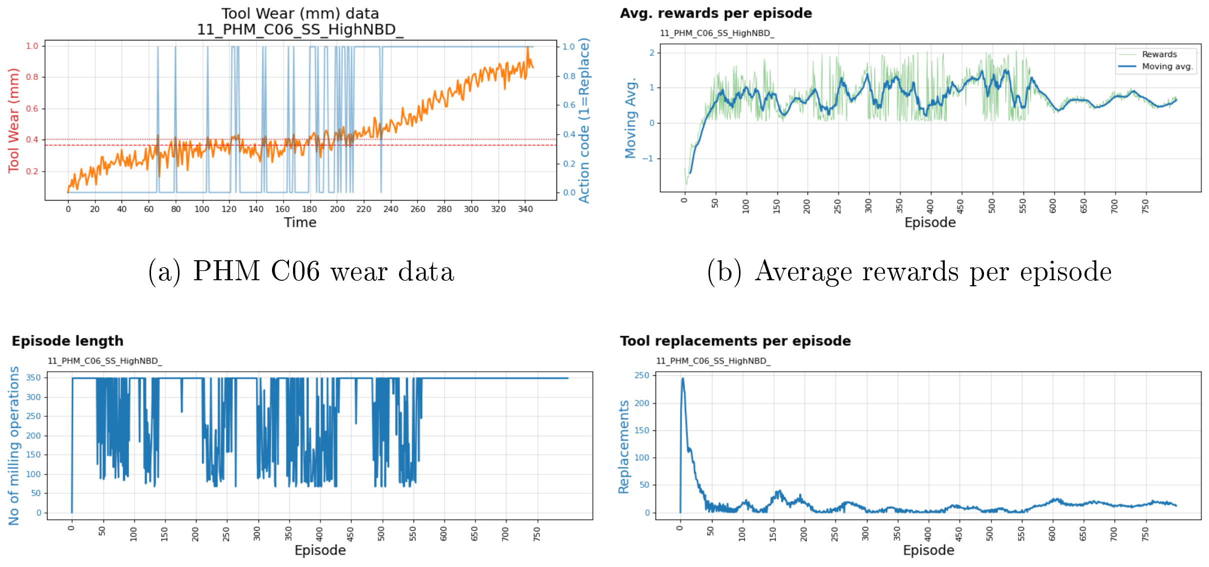
(c) Episode length completed per episode



(d) Tool replacements per episode

Figure 8: Training plots. Algorithm: REINFORCE. Dataset: PHM-C04. Variant: Univariate state, low-noise and low chance of breakdown.

Fig. 8 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. 8b), the episode length demonstrates the complexity introduced by random breakdown (which abruptly terminates the episode). It is the tool replacement policy that is essentially of interest to the industrial practitioner, this decreases to optimal levels as the agent learns over time, as seen in Fig. 8d. Similarly, Fig. 9 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. 10; as we do not introduce noise or breakdown here, the episodes are always completed (Fig. 10c).



(a) PHM C06 wear data

(b) Average rewards per episode

(c) Episode length completed per episode

(d) Tool replacements per episode

Figure 9: Training plots. Algorithm: REINFORCE. Dataset: PHM-C06. Variant: Univariate state, high-noise and high chance of breakdown.

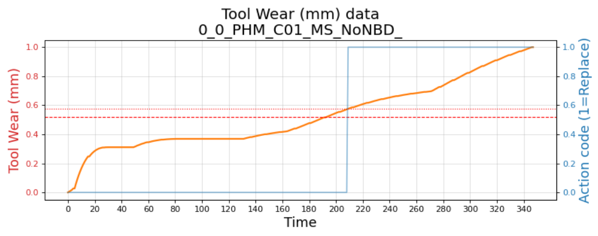
3.3 Testing and performance evaluation

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.

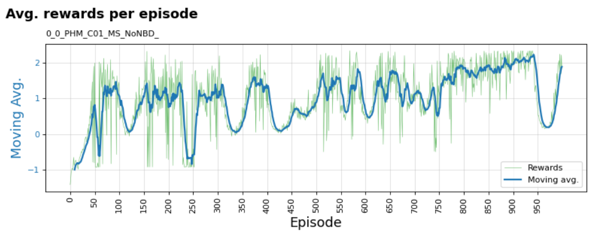
Evaluation metrics

RL algorithms are often evaluated based on the learning efficiency by using metrics such as average reward and the number of episodes to reach maximum reward. However for the industrial practitioner this is of secondary significance and it is the tool replacement precision that is of prime importance.

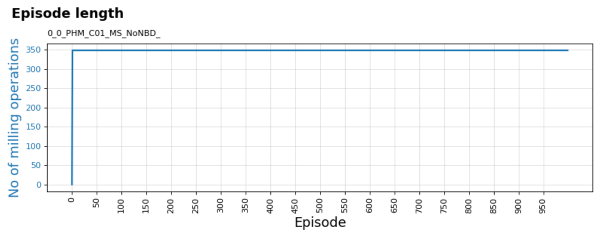
In our data, we indicate a tool replacement suggestion by a human expert as 1, and a normal operation as 0. The human decision is based on "preventive maintenance" against a set threshold. In reality the threshold is based on several factors like materials of



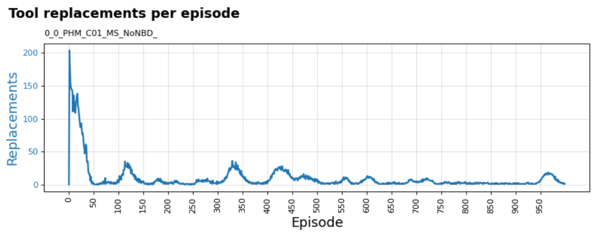
(a) PHM C01 wear data



(b) Average rewards per episode



(c) Episode length completed per episode



(d) Tool replacements per episode

Figure 10: Training plots. Algorithm: REINFORCE. Dataset: PHM-C01. Variant: Multivariate state, No noise or breakdown.

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

Figure 11: Confusion matrix: Human versus Agent decisions

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. We applied classification metrics to evaluate the RL agent decisions that suggest similar actions. These are simple to understand for the general industrial practitioners.

Classification metrics are based on the confusion matrix shown in Fig. 11. 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. With false negative FN, the agent suggests to continue milling operation when 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 want to simultaneously drive down *unnecessary* replacements and reduce costly 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 first present averaged metrics and averaged standard deviation of *all* 15 variants in Table 3, followed by category wise summary of results. These summaries are followed by error-bar plots over the 10 training rounds. Selecting the best of 10 models obtained, for each algorithm, offers a comparison of select models, we called "super models".

1. Overall performance: Averaged over all 15 environments (Table 1, items 1–15)
2. Simulated environment: Averaged over 3 noise setting variants (Table 1, items 1–3)
3. Real data - simple univariate state: Averaged over 9 variants i.e. 3 PHM datasets (C01, C04 and C06) $\times$ 3 noise settings each (Table 1, items 4–12)
4. Real data - complex multivariate state: Averaged over 3 PHM dataset variants i.e. C01, C04 and C06 (Table 1, items 13–15)

	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	$F_\beta 0.5$
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	0.360
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	0.369
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	0.382
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	0.395
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	0.448
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	0.400
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	0.473
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	0.386
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	0.333
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	0.321
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	0.294
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	0.317
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	0.460
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	0.457
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	0.498

Table 3: Model performance comparison all variants of the environments, over 10 rounds of training.

4.1 Overall summary performance

	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.

At an overall level, Table 4, the REINFORCE scores much higher than the other algorithms, on all the 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.

Fig. 12 shows the error-bar plot of the overall metrics, across the 10 rounds of training. The blue line floating above the rest of the algorithms, for all metrics, is that of REINFORCE. The error-bars are also pretty small, indicating a comparatively small uncertainty. We notice that the other three algorithms are all centered closely around, roughly, 0.5.

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: Model performance summary - averaged over simulated environments.

The simulated tool-wear environment is relatively the simplest for the agent to learn. We added two levels of noise and chance of breakdown and averaged the performance over the three variants, Table 5. 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. Plot Fig. 13 shows DQN having very high fluctuations, occasionally showing recalls at the REINFORCE levels (rounds 2 and 8).

4.3 Real data - simple univariate environment

	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.

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. Fig. 14 shows the plots for the simple univariate state. 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).

4.4 Real data - complex multivariate state

	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.

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 to the top of the table. The plot, Fig. 15 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. The F1 appears to in range of all the other algorithms.

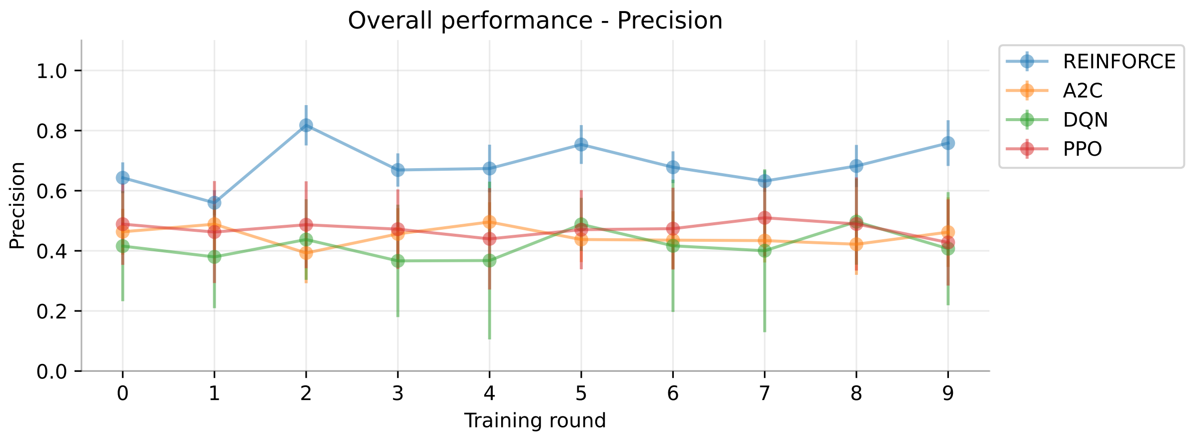
4.5 Super models

Finally, we look at performance of "select" models. Logically, one would choose the best model from a set of 10 trained models. Selecting models that produce the highest F1 with a minimum criteria for precision and recall allowed us to evaluate performance of "super-models" for each algorithm. In Table 9, the REINFORCE performs better than the other three algorithms by a huge margin, for 14 of the 15 variants. For the PHM C06, univariate environment, the DQN performs the best, with extremely high metrics throughout and an F1 of 0.969 to 0.831 of REINFORCE.

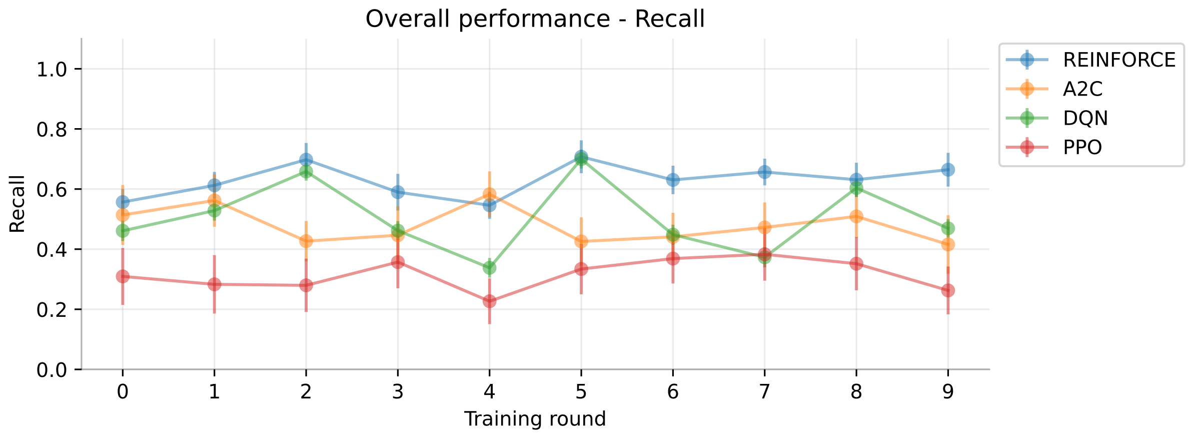
As an overall average performance, Table 8 demonstrates the remarkable performance of the simple REINFORCE algorithm. It does show a lower recall when compared to the DQN, however it is a much more *balanced* model on an overall basis.

	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

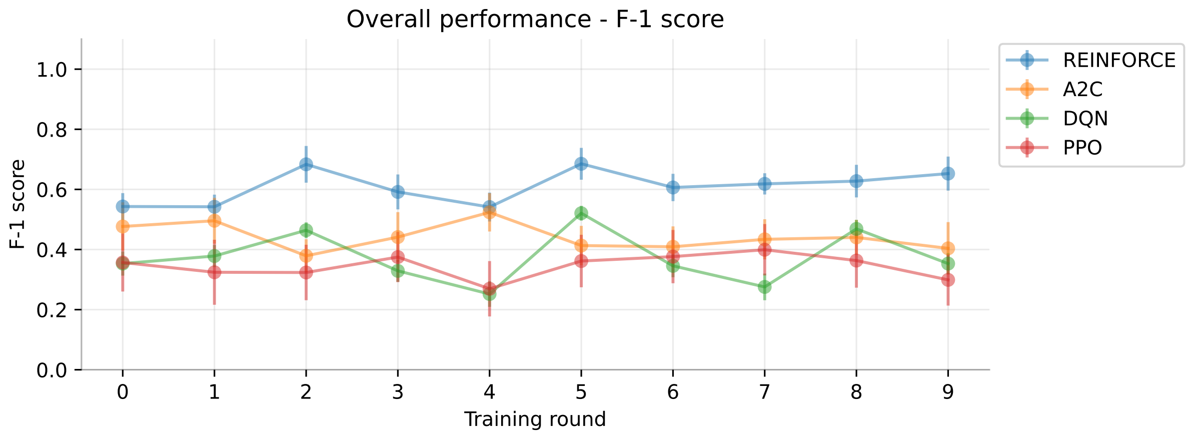
Table 8: Super models: Best of 10 rounds; performance averaged over all 15 environments.



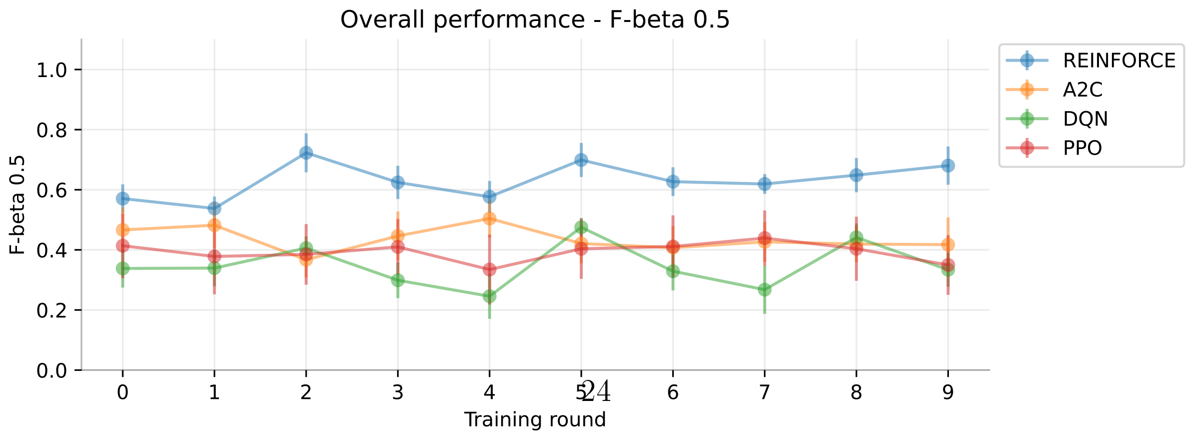
(a) Precision



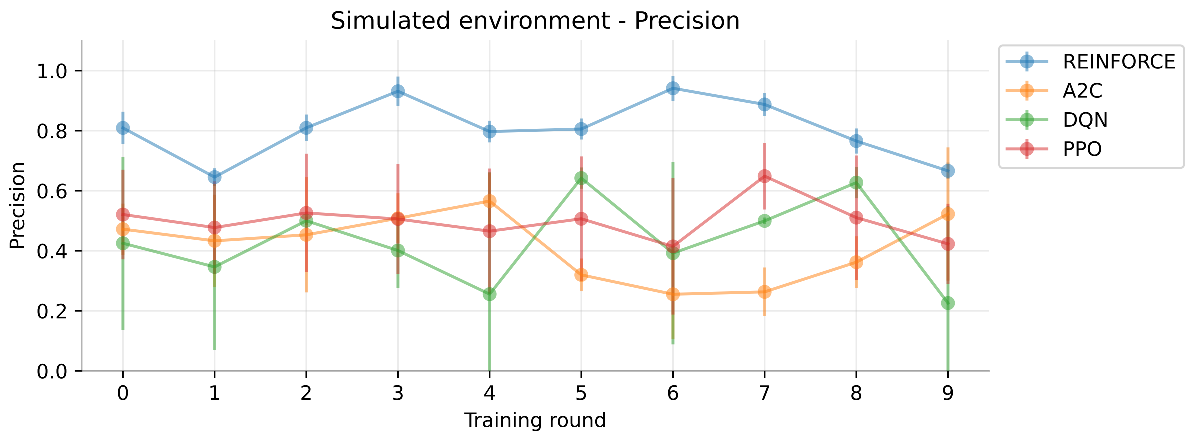
(b) Recall



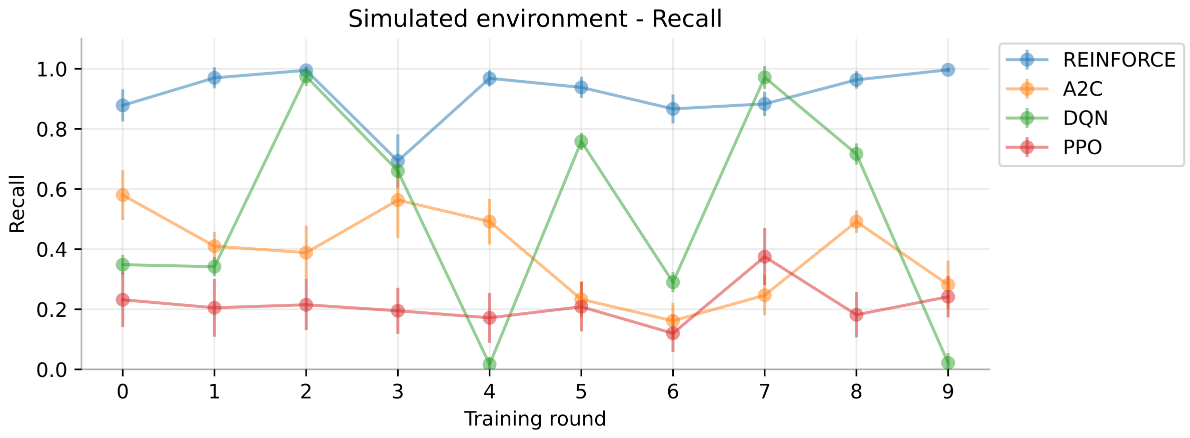
(c) F1-score



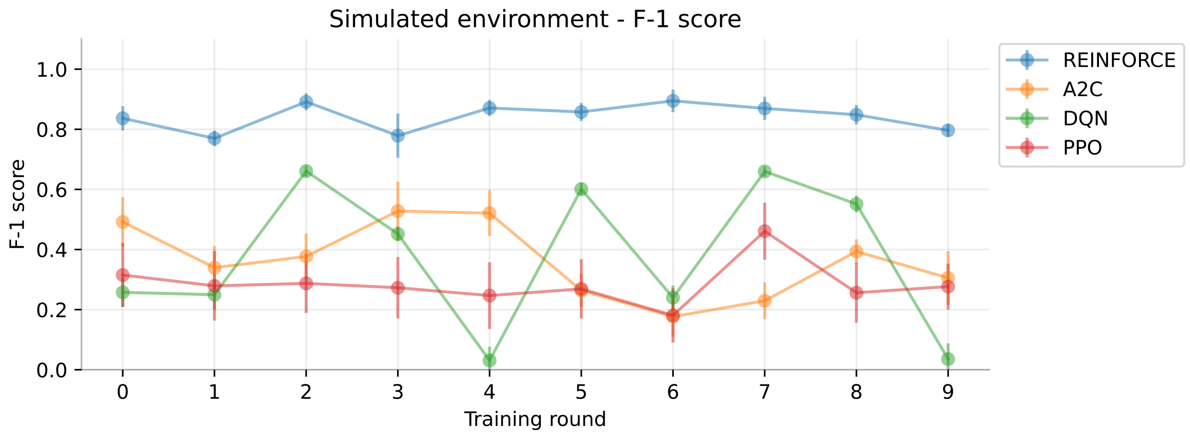
(d) F-beta (0.5)



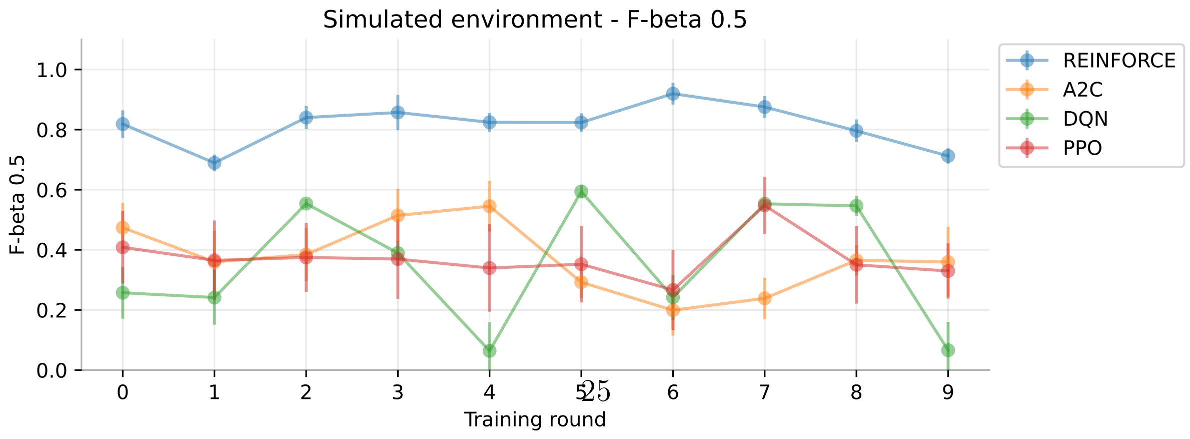
(a) Precision



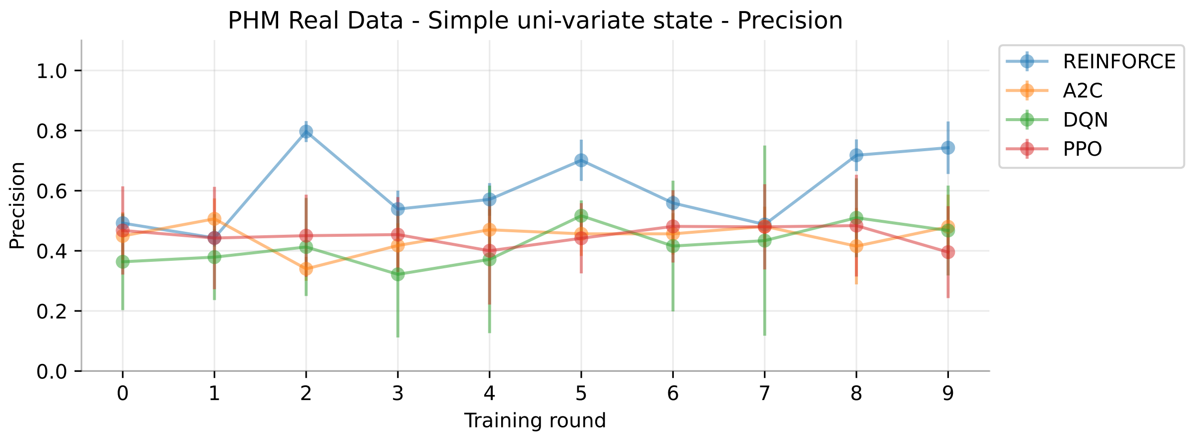
(b) Recall



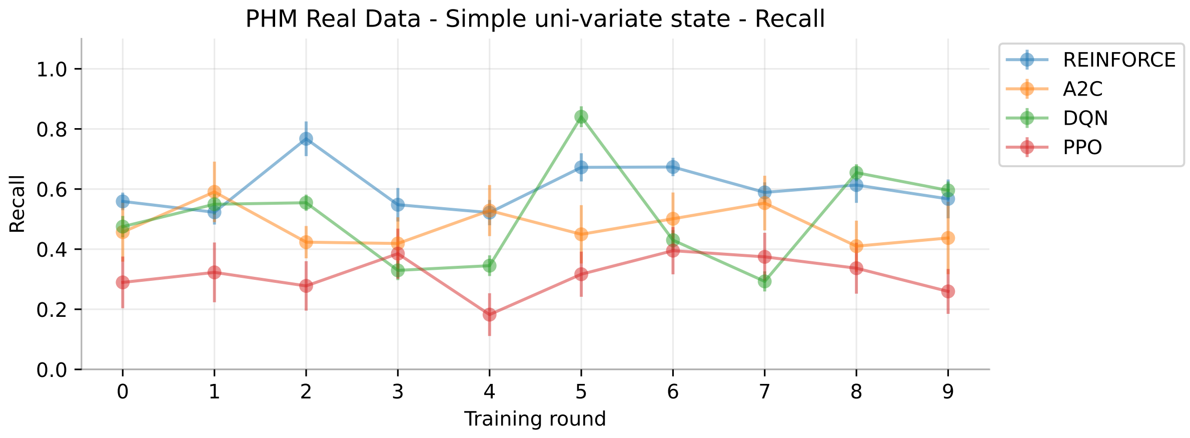
(c) F1-score



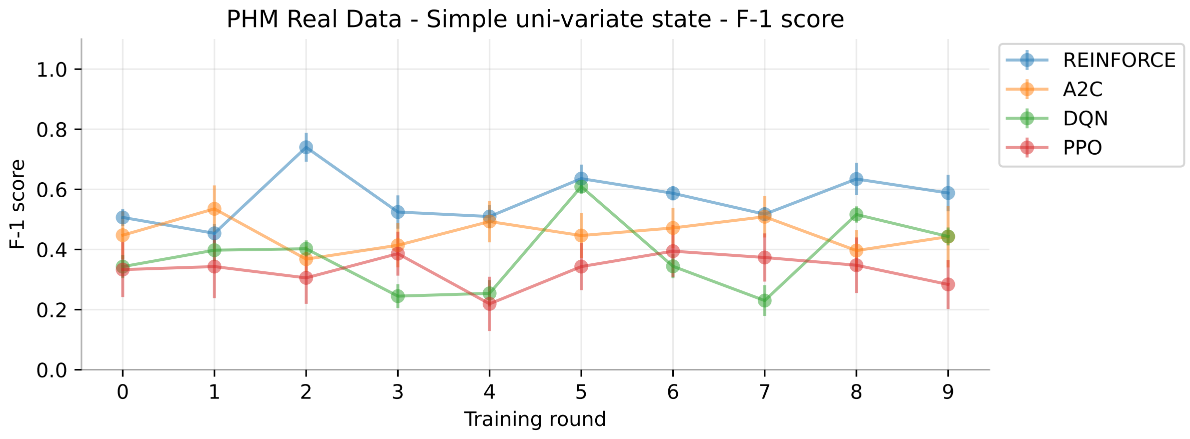
(d) F-beta (0.5)



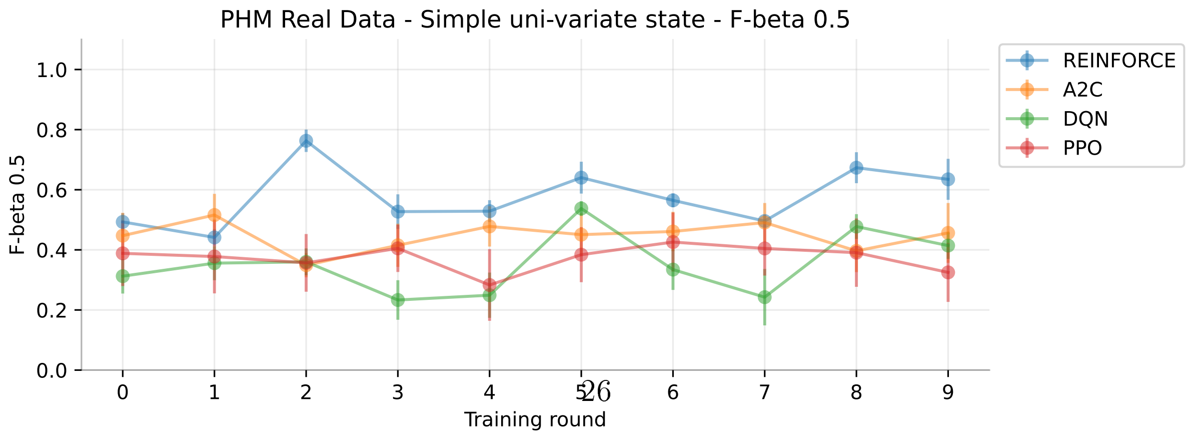
(a) Precision



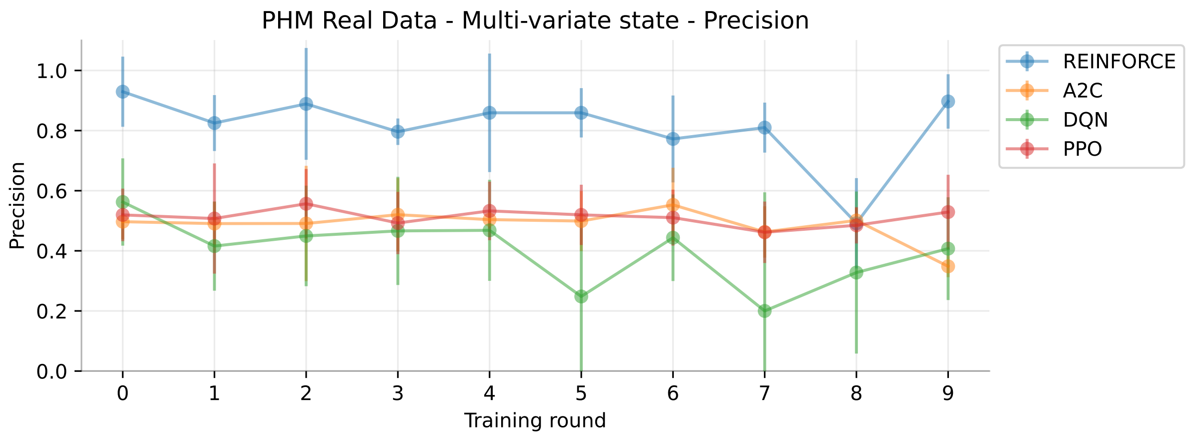
(b) Recall



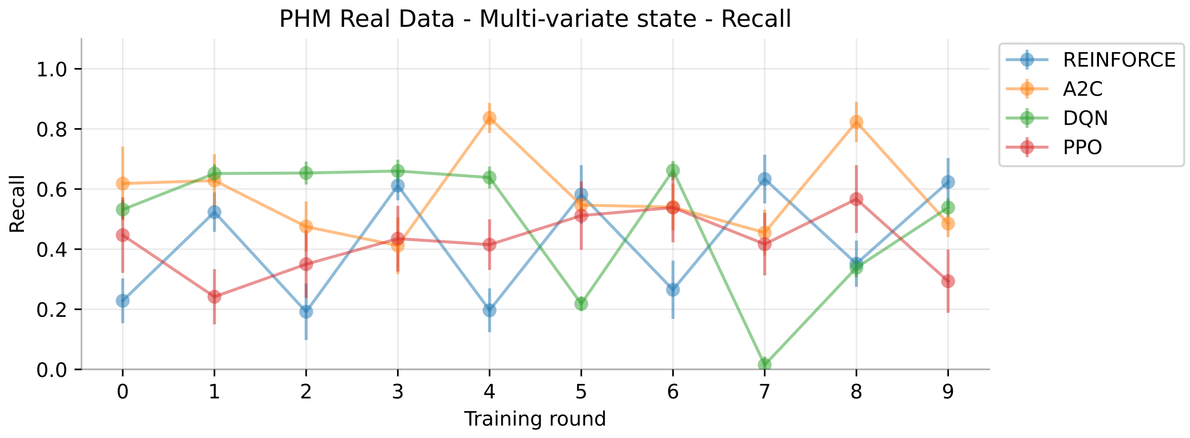
(c) F1-score



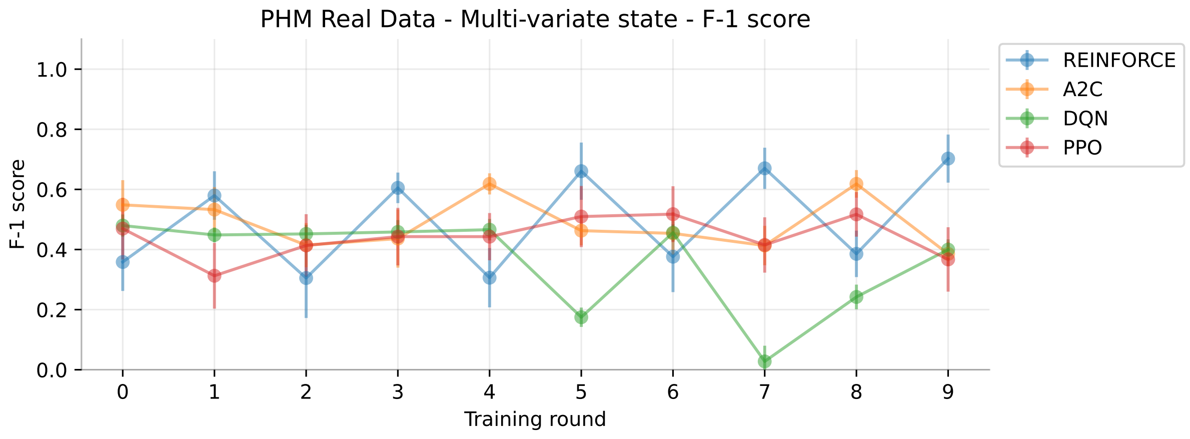
(d) F-beta (0.5)



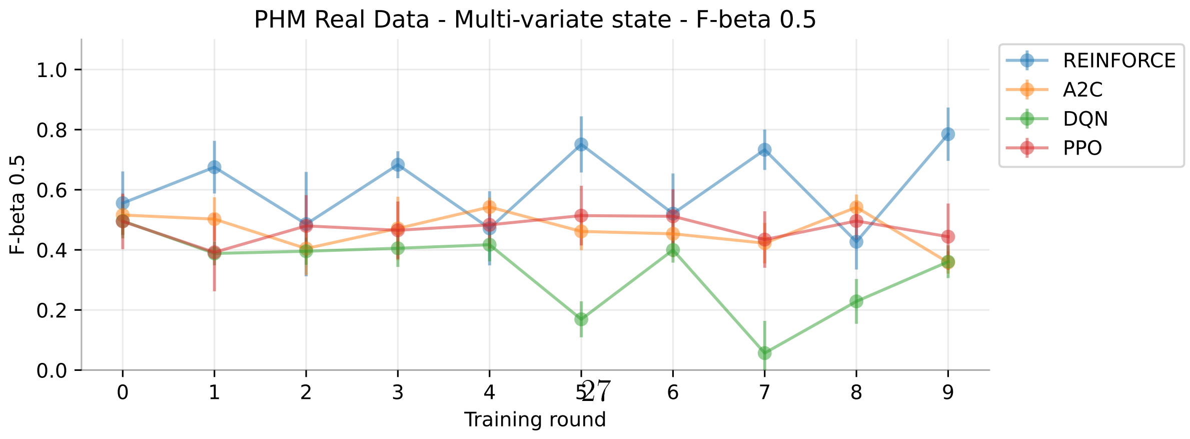
(a) Precision



(b) Recall



(c) F1-score



(d) F-beta (0.5)

	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	$F_\beta 0.5$
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	0.597
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	0.586
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	0.505
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	0.597
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	0.643
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	0.551
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	0.622
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	0.565
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	0.558
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	0.597
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	0.497
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	0.537
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	0.526
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	0.522
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	0.526

Table 9: Super Models: Best models selected over 10 rounds of training.

4.6 Hypothesis testing

We undertake a statistical analysis of the results, postulated by the hypothesis (13). Table 10 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[5].

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

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. 16.

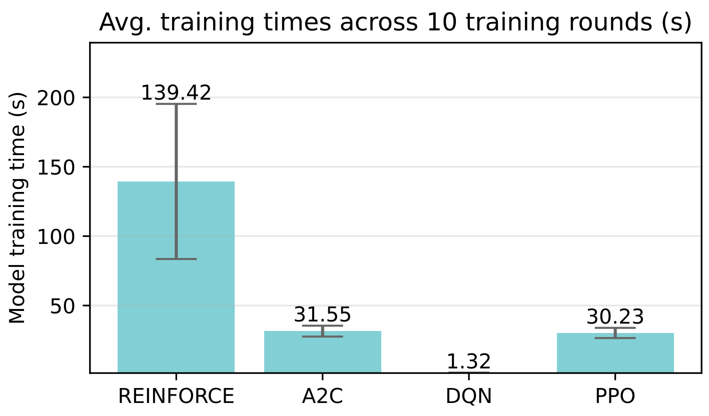


Figure 16: Training time, averaged over 10 rounds and all 15 variants.

5 Discussion

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

[5]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.

	p Value				t Statistic		
Metric	RF $\overset{H_a}{\underset{H_0}{\geq}}$ A2C	RF $\overset{H_a}{\underset{H_0}{\geq}}$DQN	RF $\overset{H_a}{\underset{H_0}{\geq}}$PPO		RF $\overset{H_a}{\underset{H_0}{\geq}}$ A2C	RF $\overset{H_a}{\underset{H_0}{\geq}}$DQN	RF $\overset{H_a}{\underset{H_0}{\geq}}$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 10: 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

REINFORCE and in their paper provide an intuition for convergence properties. This supports the extensive experimental studies on REINFORCE performed by Duan et al. (2016); Peck and Renaux (2019); Zhang et al. (2021a,b) as well our empirical research.

Both Duan et al. (2016) and Zhang et al. (2021a) studied the behavior on multiple OpenAI Gym environments, while Peck and Renaux (2019) provide empirical results for MuJuCo environments. 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. (2021b) 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 perform 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.

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 variants of environment. 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) mention how small implementation details can significantly impact performance that is often greater than the difference between the algorithms themselves. We hope our research will contribute to empirical evidence in the study of REINFORCE and therefore contribute in an infinitesimal way to enable AutoML for the field of predictive maintenance using RL.

References

Predrag Dašić. 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, 2006.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International conference on machine learning*, pages 1329–1338. PMLR, 2016.

Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. An empirical investigation of the challenges of real-world reinforcement learning. *arXiv preprint arXiv:2003.11881*, 2020.

Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021.

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

SA Ford and M Ritchie. Cognitive radar mode control: a comparison of different reinforcement learning algorithms. In *International Conference on Radar Systems (RADAR 2022)*, volume 2022, pages 107–112. IET, 2022.

Future Market Insights. Milling machine market outlook (2023 to 2033), 1 2023. URL https://www.futuremarketinsights.com/reports/milling-machine-market. Accessed: 2023-06-23.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

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

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

Marcel Panzer and Benedict Bender. Deep reinforcement learning in production systems: a systematic literature review. *International Journal of Production Research*, 2021.

Ryan Peck and Louis Renaux. A review of reinforce algorithms. 2019.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *J. Mach. Learn. Res.*, 22(1), jan 2021. ISSN 1532-4435.

N Sandeep Varma, Vaishnavi Sinha, and K Pradyumna Rahul. Experimental evaluation of reinforcement learning algorithms. In *International Conference on Computational Intelligence and Data Engineering*, pages 469–484. Springer, 2022.

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

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

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Rajesh Siraskar, Satish Kumar, Shruti Patil, Arunkumar Bongale, and Ketan Kotecha. Reinforcement learning for predictive maintenance: a systematic technical review. *Artificial Intelligence Review*, pages 1–63, 2023.

Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, England, 2nd. edition edition, 2018.

Krishna Velivela and Sudhir Yarram. Comparison of reinforcement learning algorithms, Dec 2020.

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

Junyu Zhang, Chengzhuo Ni, Csaba Szepesvari, Mengdi Wang, et al. On the convergence and sample efficiency of variance-reduced policy gradient method. *Advances in Neural Information Processing Systems*, 34:2228–2240, 2021a.

Junzi Zhang, Jongho Kim, Brendan O'Donoghue, and Stephen Boyd. Sample efficient reinforcement learning with reinforce. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 10887–10895, 2021b.