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

Rajesh Siraskar

July 14, 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 not been studied so far. This article is an empirical study, aimed at industrial practitioners not accustomed to 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 understand the performance of *untuned* algorithms under various scenarios: (1) simulation tool-wear data (2) real tool-wear data (benchmark IEEE NUAA Ideahouse dataset) (3) added noise levels and a random chance of break-down.

Model performance was measured by how accurately the predictive maintenance agent suggested tool replacement when compared to a deterministic preventive maintenance rule based on the tool-wear threshold. Across variants of the environment, 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, when compared to the average of the three advanced algorithms. Comparing the *best model* over 10 rounds of training produced surprisingly larger 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. Our findings indicate that the computationally lightweight REINFORCE performs significantly well for this particular problem. Consequently, for this particular problem, selecting the naïve REINFORCE could be a more suitable and effective policy generating alternative to more advanced complex algorithms.

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	Multi-variate state
TP	True positive	TN	True negative
FP	False positive	FN	False negative
RL	Reinforcement Learning	SB3	Stable-Baselines3
PHM	The Prognostics and Health Management Society		

List of Figures

List of Tables

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 in 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.

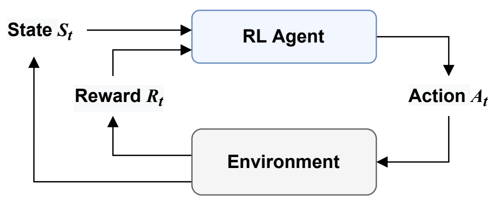


Figure 1: Reinforcement Learning

Introduced in 1992, the REINFORCE algorithm (Williams, 1992) is considered as a basic reinforcement learning 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).

While most studies on 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 the earliest RL algorithm, the REINFORCE. This is often neglected by studies, as it is considered an early and weak algorithm. In the industry the milling tool is replaced after a set threshold. We use this "deterministic preventive maintenance" policy as the baseline for comparing the RL algorithm policy.

Our systematic evaluation, based on levels of environment difficulty, different bench-mark data sets 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 F1-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. Research is targeted toward the industrial practitioner not accustomed to complex hyper-parameter tuning of RL algorithms
2. Design and implement an RL environment for predictive maintenance of a milling machine.
3. Rigorous evaluation of four standard RL algorithms.
4. 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 presents the results of the experiments, that are discussed in Section 5. 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 (Erhan et al., 2021; Panzer and Bender, 2021; Siraskar et al., 2023), in general. However, we did not find[1] any work done for predictive maintenance of *milling machines* across three major journal index services[2]. While RL is not, traditional machine learning methods *have* been applied; for example Qin et al. (2023) and Qiang et al. (2023) apply tool-wear law and physics based models before applying ML. Twardowski et al. (2023) and Denkena et al. (2023) use data gathered by sensors. While Twardowski et al. (2023) use two different classification trees, Denkena et al. (2023) use data recorded on other similar machines for building ML models. Oshida et al. (2023) proposes real-time tool wear detection using a stacked LSTM[3] encoder-decoder model for anomaly detection as a mechanism to address predictive maintenance.

Research work focusing of comparing experimental and empirical 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 OpenAI Gym environments such as the Pendulum, Mountain Car, Bipedal Walker, Lunar Landing and Atari 2600 game

[1]Search terms: `"reinforcement learning AND tool wear AND maintenance"`. As of: 10-Jul-2023.

[2]IEEE Xplore™, Scopus™ and Web Of Science™

[3]long short-term memory networks

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[4]. 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[5] environments such as the HalfCheetah, Hopper, Walker and Swimmer. As in our research they used the OpenAI baseline implementations of RL algorithms for the experiments and evaluation. Ford and Ritchie (2022) is experimental evaluation we found that was based on real-world application. They compare DQN, A2C and PPO 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.

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

[4]Link to RWRL » link

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

($s \in \mathcal{S}$). 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.

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

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 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 to update the action probabilities.

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

Stable-Baselines3

Stable-Baselines3 (SB3) Raffin et al. (2021) provides robust open source implementations of many reinforcement learning algorithms. The core algorithms of Stable-Baselines3 are completely refactored PyTorch implementations of Open AI baselines (Dhariwal et al., 2017). Huang (2022) describes the 37 PPO implementation details that SB3 carried out and demonstrates the complexity depth of their implementations.

As of 10-Jul-2023, 13 algorithms have been implemented (SB3-Algorithms, 2022). REINFORCE has *not* been implemented. For this research we use the SB3 implementations of DQN, A2C and PPO and compare its performance to a naïve custom implementation of the REINFORCE algorithm.

3 Methodology

3.1 Implementation details

xx

3.1.1 Data description

- Simulated
- PHM IEEE NUAA Ideahouse dataset has been used in this paper for the RUL estimation of the milling cutter. Yingguang et al. (2021)

3.1.2 Environment design

we follow the procedure in Graesser and Keng (2019) pg 289 Part iv, ch14-15.

3.1.3 simulated data

(Dašić, 2006). eq 16 from the paper parameters b0, b1, b2 and a under the form of a complex power-exponential regression equation power functions as approximating functions of cutting tool wear, we proposed is powerexponential function form

tool flank wear VB

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

b0 -2.4941 b1 0.3342 b2 0.03147 a 0.08257 Determination of Complex Power- Exponential Regression Equation for Functional Dependence between Tool Wear and Cutting Time for Cutting Speed v=79.2 [m/min] Parameter calculation b0, b1 and b2 of the linear regression for the mentioned example is consisted in the solving of the normal system equation (12) with the following shape:

Implementation details: Normalization

init
reset
reward function

degradation model as exponential

$$H(t) = 1 - D_0 - \mathrm{e}^{(at^b)}, \tag{10}$$

where, D_0 is the initial degradation state while a and b are wear-rate coefficients that depend on the effect of temperature, vibration and other system stress parameters.

$$R_t = \begin{cases} 0, & \text{if } U(s_t) = U(s_{t+1}) = 0 \\ -1.0, & \text{if } U(s_t) = 0, U(s_{t+1}) = 1 \\ R_{ff}, & \text{if } U(s_t) = 1 \end{cases} \tag{11}$$

3.1.4 SS

```
def _get_observation(self, index):
next_state = np.array([
self.df['time'][index],
self.df['tool_wear'][index]
], dtype=np.float32)

return next_state
```

3.1.5 MS

```
def _get_observation(self, index):
  next_state = np.array([
  self.df['tool_wear'][index],
  self.df['force_x'][index],
  self.df['force_y'][index],
  self.df['force_z'][index],
  self.df['vibration_x'][index],
```

```python
    self.df['vibration_y'][index],
    self.df['vibration_z'][index],
    self.df['acoustic_emission_rms'][index]
    ], dtype=np.float32)
  return next_state
```

```python
def _get_observation(self, index):
next_state = np.array([
self.df['tool_wear'][index],
self.df['force_x'][index],
self.df['force_y'][index],
self.df['force_z'][index],
self.df['vibration_x'][index],
self.df['vibration_y'][index],
self.df['vibration_z'][index],
self.df['acoustic_emission_rms'][index]
], dtype=np.float32)
return next_state

# Machine data frame properties
self.df = df
self.df_length = len(self.df.index)
self.df_index = 0

# Milling operation and tool parameters
self.wear_threshold = wear_threshold
self.max_operations = max_operations
self.breakdown_chance = breakdown_chance
self.add_noise = add_noise
self.R1 = R1
self.R2 = R2
self.R3 = R3
self.reward = 0.0
```

3.1.6 main env

```
env = MillingTool_SS_NT(df_train, WEAR_THRESHOLD_NORMALIZED, MILLING_OPERATIONS_
```

3.1.7 test env

```
env_test = MillingTool_SS_NT(df_test, WEAR_THRESHOLD_ORG_NORMALIZED, MILLING_OPE
```

- SS
- MS
- States: We normalize the tool wear and other state features, $x \in [0,\ 1] \subset \mathbb{R}$. This allows for adding white noise of similar magnitudes across experiments of different data-sets
- Actions
- Reward signal

- R1 = +1
- R2 = -1
- R3 = -100 => higher neg. Improve recall. Lower neg. Improve precision.
- Hyper-parameters for Precision or Recall control:
- Precision => low FP => False replacement-action reduced. *** Unnecessary tool replacement reduced. Tool life maximized, down time minimized, production disruption minimized.
- - Recall => low FN => False declaration of normal operation reduced. Reduce missed replacements. Tool replacements increased. *** Product quality not compromised.
- **LOOK AHEAD PARAM**:

- Transition function (Graesser and Keng, 2019): pg333 "Transition function - also known as model. can be learnt or programmed programmable rules

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

are common and lead to increasing complexity. *** We use a hybrid model - partly learnt and partly programmed": **Noise and breakdown**

3.1.8 Network architecture and basic hyper-parameters

3.1.9 PPO hyperparms

- soure of ppo implementation details https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/
- source of SB3 network mp : https://github.com/openai/baselines/blob/ea25b9e8b234e6ee1bca43083f8f3cf974143998/baselines/common/models.py#L75-L103

implementation guide source » https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/

By default, PPO uses a simple MLP network consisting of two layers of

64 neurons and Hyperbolic Tangent as the activation function. Then PPO builds a policy head and value head that share the outputs of the MLP network. Below is a pseudocode:

3.1.10 dqn hyperparms

default hyperparms : https://stable-baselines3.readthedocs.io/en/master/_modules/stable_baselines3/common/policies.html

overridden in indiv policies for example SB3 DQN hypoerparms for example were taken from ; Paper: https://arxiv.org/abs/1312.5602, https://www.nature.com/articles/ Default hyperparameters are taken from the Nature paper, except for the optimizer and learning rate that were taken from Stable Baselines defaults

3.1.11 Hyper-parameters for Precision or Recall control

- xxx

*The methodology explains in detail what the researcher did to undertake the research. Various aspects of the research have to be outlined: The overall structure and operation of the experiment or observational experience. The groups studied in the research including the size of each group and any features of the subjects which may be relevant to the topic being researched. The variables that were changed between groups and the variables measured as a result of the changes. The conditions under which the research was undertaken and any factors or variations in conditions which may have an impact on the results. The methods of data analysis used in order to analyse and collate the results. ***Any limitations of the data collected.*

3.2 Training

training

- Training: SB3 - 10 k eps. 3 times. Average their outputs
- Testing:

  - Avg. over 5 rounds.
  - Each round - avg over 40 test cases x 10 test rounds
  - Total: 40 x 10 x 5 = 2000 cases
  - Avgs over: 10 rounds (of 40 cases each) X 5 rounds of **re-trained** SB3 agents = 50 rounds

– selecting the model
– conducting the experiments

3.3 Testing

- Testing `env-test = wear threshold original`
- Experiment file
- Rounds

3.4 Evaluation method

-pr, rc, f1, f1-beta

- why classifction metrics
- why F1beta

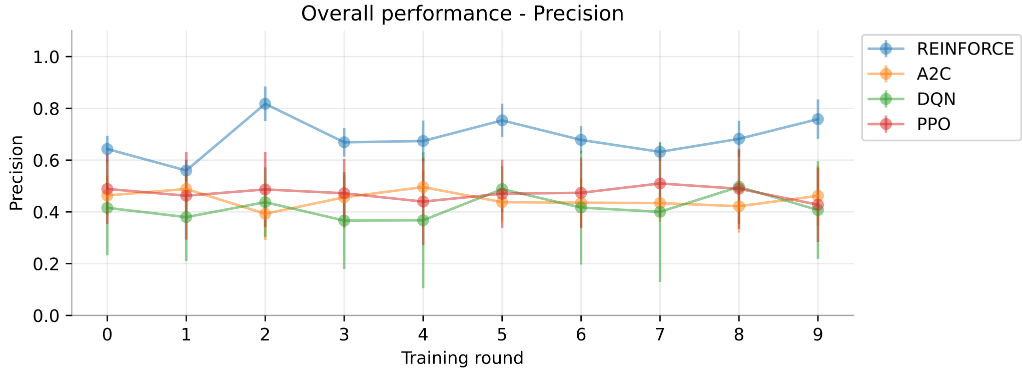
3.5 Inference

- Training: SB3 is also unstable - show examples of results such as A2C/DQN 0.00
- Training: SB3 is also unstable - SHOW SB3 tensorboard plots
- Training: SB3 is also unstable - EXCEL plots of results over the 10 rounds

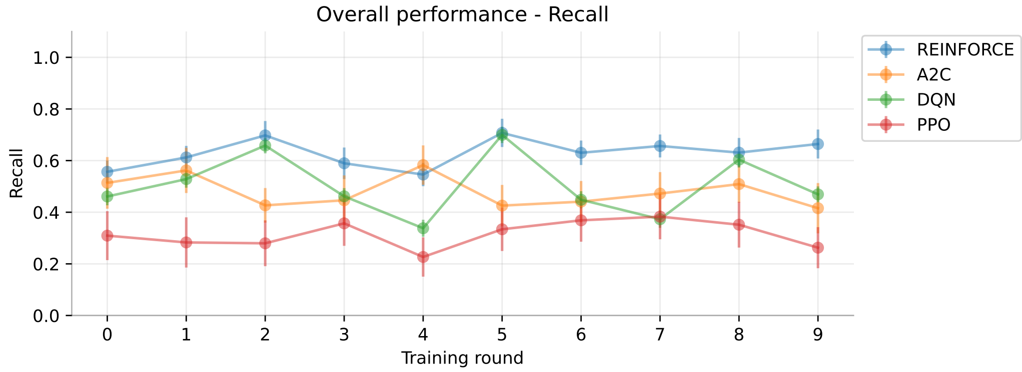
4 Results

4.1 Detailed metrics

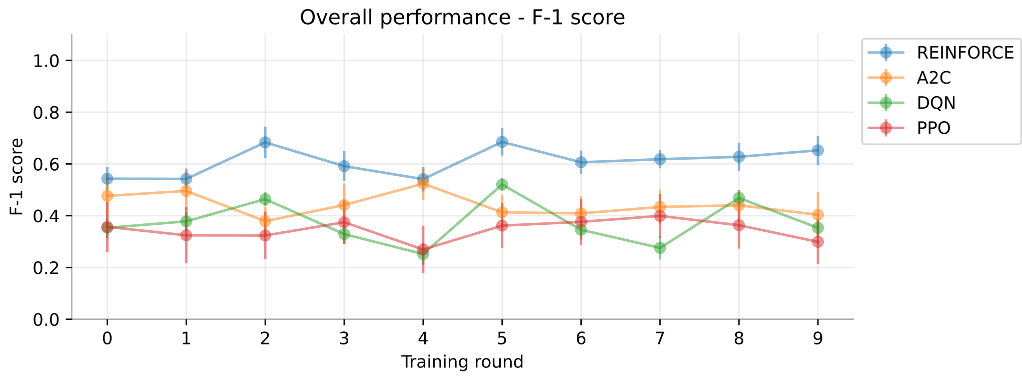
xx



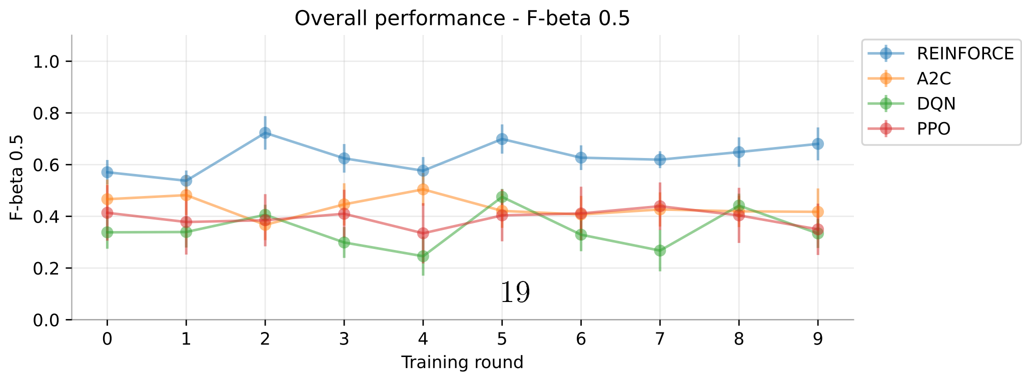
(a) Precision



(b) Recall

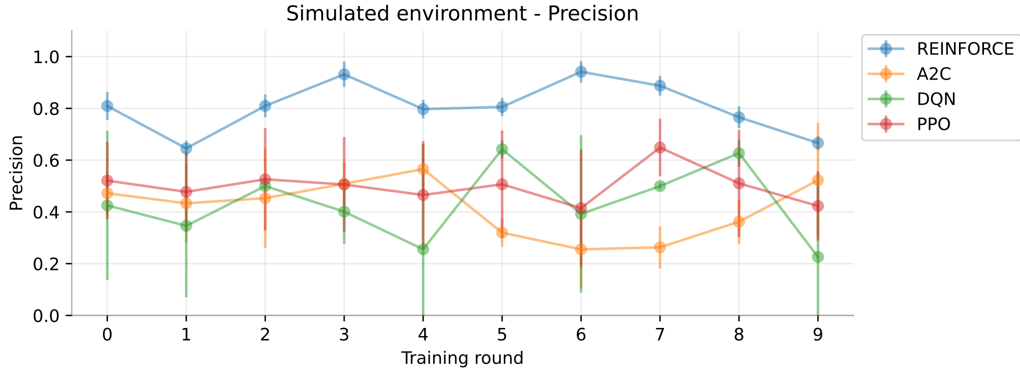


(c) F1-score

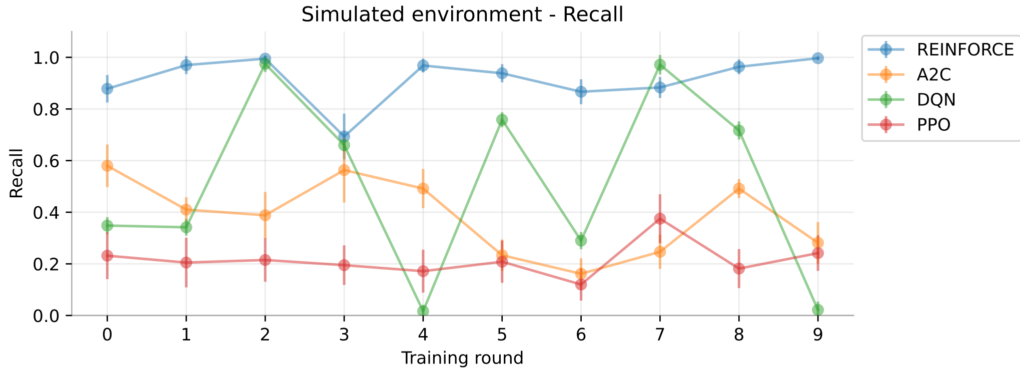


(d) F1-beta (0.5)

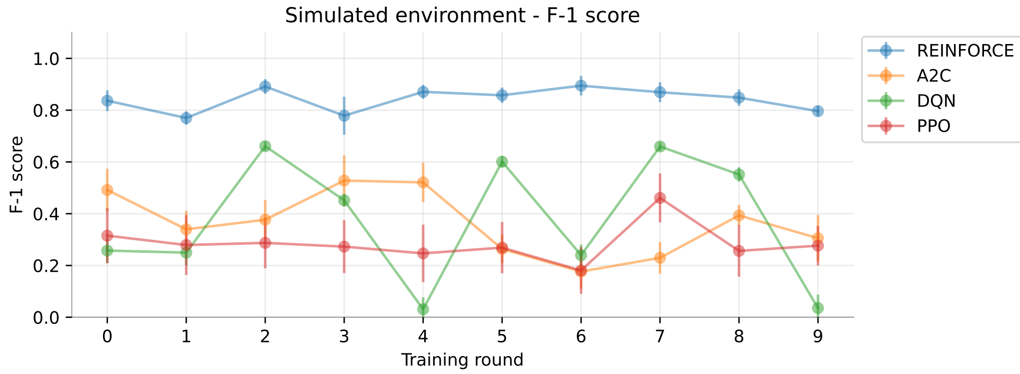
Figure 2: Overall performance – Avg. performance over 10 rounds of model training



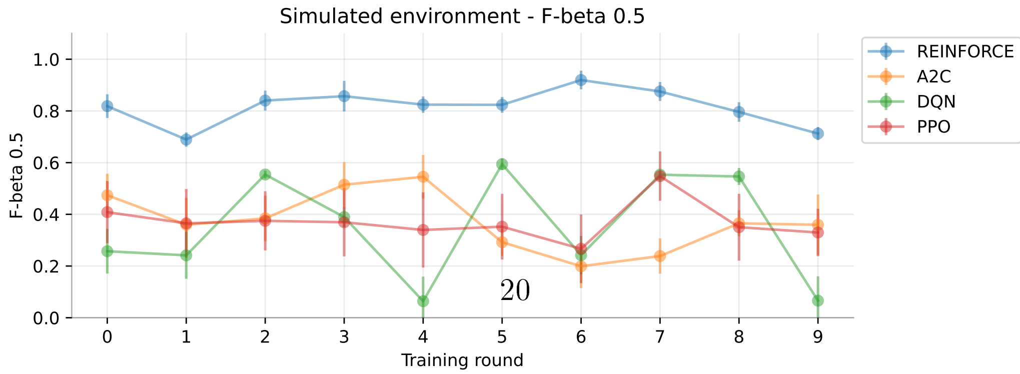
(a) Precision



(b) Recall

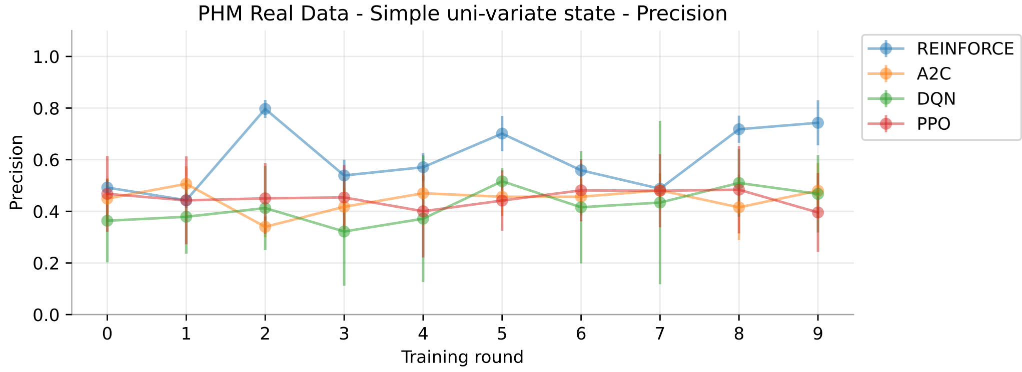


(c) F1-score

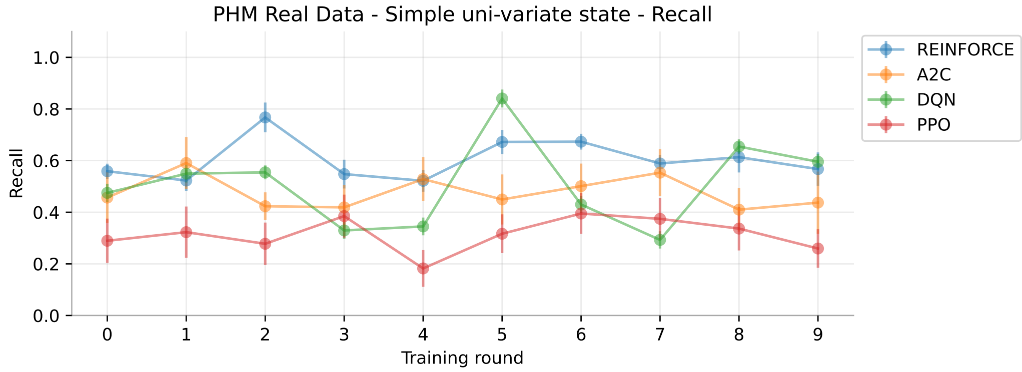


(d) F1-beta (0.5)

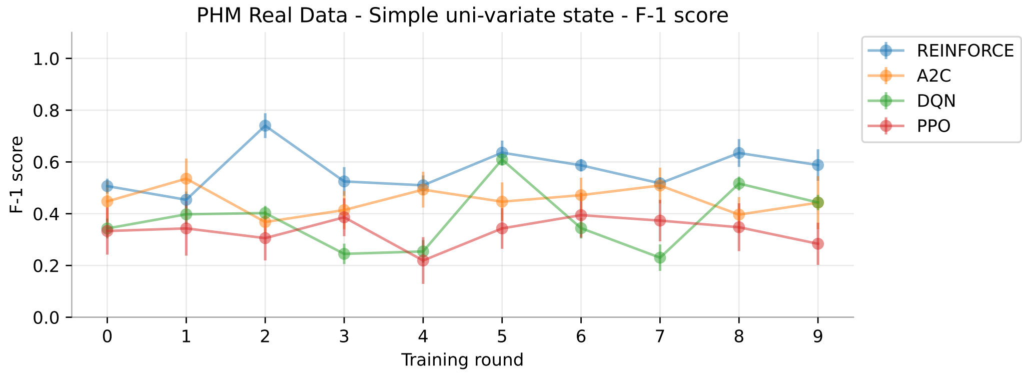
Figure 3: Simulated environment – Avg. performance over 10 rounds of model training



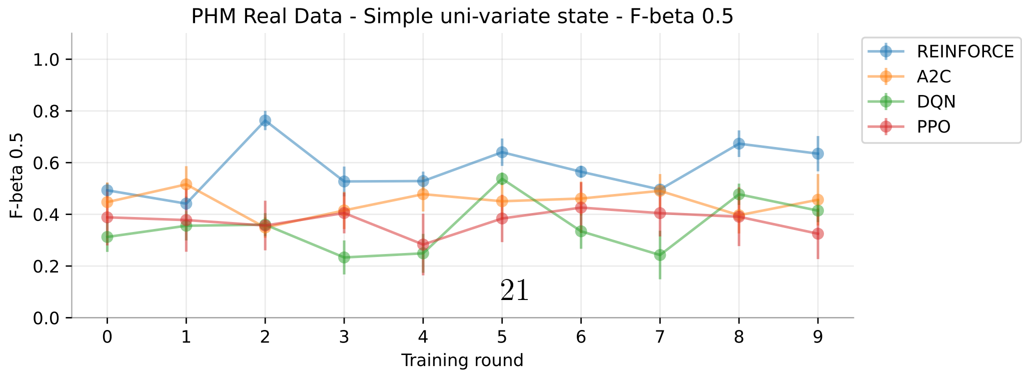
(a) Precision



(b) Recall

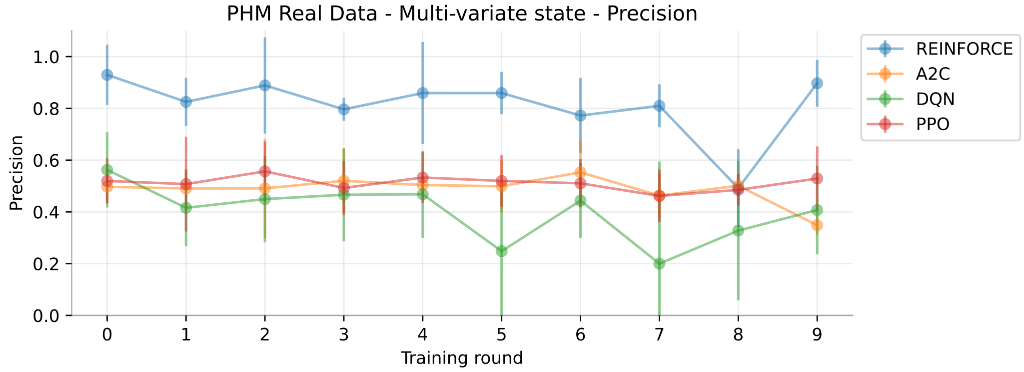


(c) F1-score

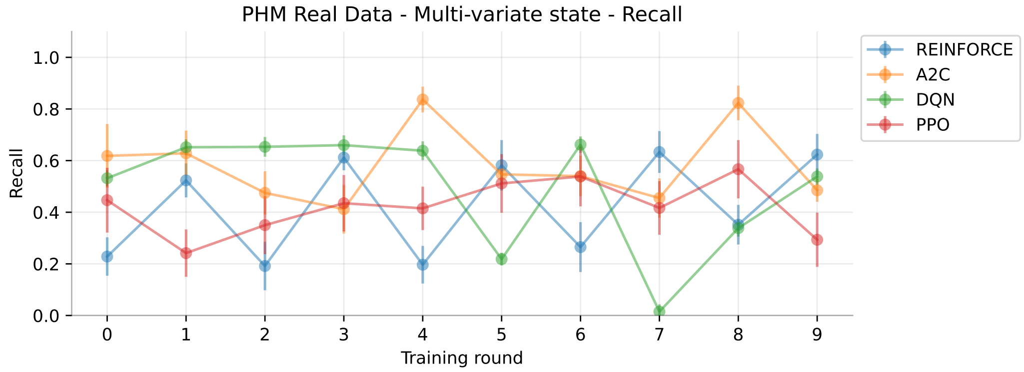


(d) F1-beta (0.5)

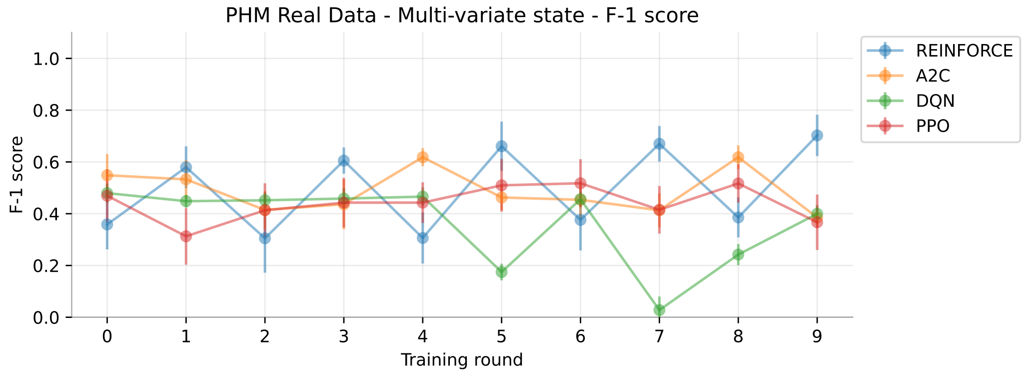
Figure 4: Singe-variable state environment – Avg. performance over 10 rounds of model training



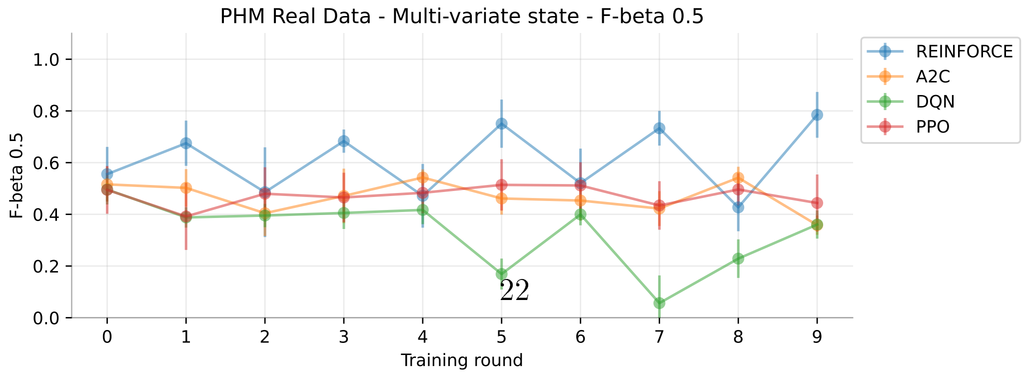
(a) Precision



(b) Recall



(c) F1-score



(d) F1-beta (0.5)

Figure 5: Multi-variate state environment – Avg. performance over 10 rounds of model training

	REINFORCE				A2C				DQN				PPO			
Environment	Prec.	Recall	F1	F0.5	Prec.	Recall	F1	F0.5	Prec.	Recall	F1	F0.5	Prec.	Recall	F1	F0.5
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 2: Model performance comparison all variants of the environments, over 10 rounds of training.

4.2 Super models metrics

	REINFORCE				A2C				DQN				PPO			
Environment	Prec.	Recall	F1	F0.5	Prec.	Recall	F1	F0.5	Prec.	Recall	F1	F0.5	Prec.	Recall	F1	F0.5
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 3: Super Models: Best models selected over 10 rounds of training.

4.3 Overall summary performance

	Precision		Recall		F1-score		F1-beta score	
	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 environment.

4.4 Simulated environment

	Precision		Recall		F1-score		F1-beta score	
	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.

4.5 Real data – simple uni-variate environment

	Precision		Recall		F1-score		F1-beta score	
	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.

4.6 Real data – complex multi-variate state

	Precision		Recall		F1-score		F1-beta score	
	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 multi-variate environment.

4.7 Hypothesis testing

Notes:

- Definition of 1 data point
- Single training round (5th of the 10 rounds)
- Single dataset: PHM C01

- Single env: Simple uni-variate state
- Single noise setting: HBD = high
- File has 40 rows = 4 algos x 10 test rounds
- 10 test rounds of 40 random sampled points
- 1 point = 1 row = 1 classification test with 40 samples
- E.g. Dasic = 10 training rounds x 10 test rounds x 3 noise settings = 300 sample points
- E.g. PHM SS = 3 data-sets x 10 training rounds x 10 test rounds x 3 noise settings = 900 sample points
- Technically Dasic = 300 points for metrics = 300 * 40 sampled data points of wear data = 12000

$$\left.\begin{aligned} H_0 &: \mu_{RF} - \mu_{AA} = 0, \\ H_a &: \mu_{RF} - \mu_{AA} > 0, \end{aligned}\right\} \qquad \forall \;\; AA \in [A2C, DQN, PPO] \tag{12}$$

4.8 Training times

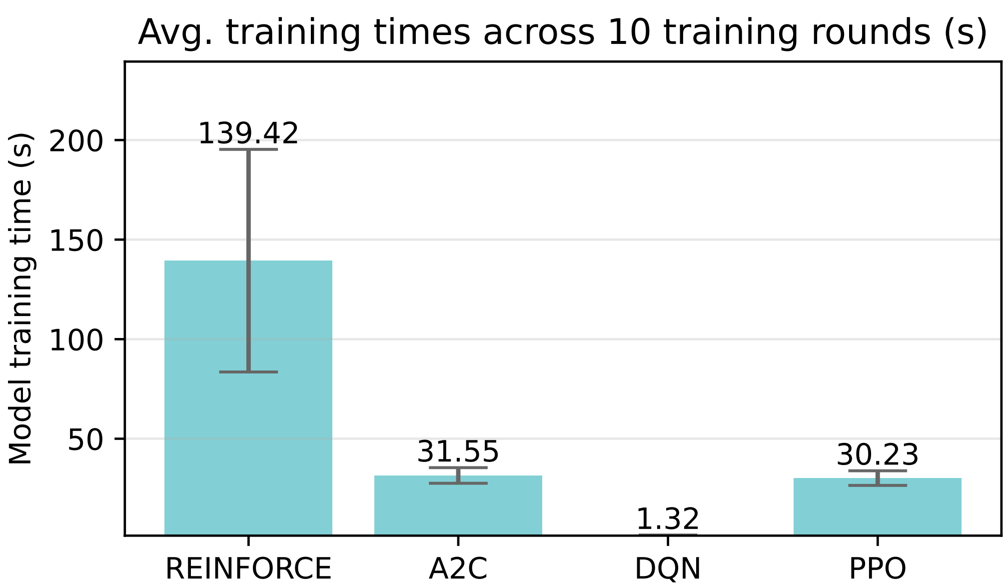


Figure 6: Training time. Across 10 rounds and all environment variants.

	p Value			t Statistic		
Metric	RF $\overset{H_a}{\underset{H_0}{\geqq}}$ A2C	RF $\overset{H_a}{\underset{H_0}{\geqq}}$ DQN	RF $\overset{H_a}{\underset{H_0}{\geqq}}$ PPO	RF $\overset{H_a}{\underset{H_0}{\geqq}}$ A2C	RF $\overset{H_a}{\underset{H_0}{\geqq}}$ DQN	RF $\overset{H_a}{\underset{H_0}{\geqq}}$ 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 uni-variate 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 multi-variate 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 8: One-tail t-test - Ho: No difference in metrics. Ha: REINFORCE metric $>$ Advanced algorithm metric

5 Discussion

Duan et al. (2016) the environment.*OBSERVATION**REINFORCE: Despite its simplicity, REINFORCE is an effective algorithm in optimizing deep neural network policies in most basic and locomotion tasks. Even for high-DOF tasks like Ant, REINFORCE can achieve competitive results. However we observe that REINFORCE sometimes suffers from premature convergence to local optima as noted by Peters Schaal (2008), which explains the performance gaps between REINFORCE and TNPG on tasks such as Walker (Figure 3(a)). By visualizing the final policies, we can see that REINFORCE results in policies that tend to jump forward and fall over to maximize short-term return instead of acquiring a stable walking gait to maximize long-term return. In Figure 3(b), we can observe that even with a small learning rate, steps taken by REINFORCE can sometimes result in large changes to policy distribution, which may explain the fast convergence to local optima.

5.1 Precision or Recall?

- Precision => low FP => False replacement-action reduced. *** Unnecessary tool replacement reduced. Tool life maximized, down time minimized, production disruption minimized
- Recall => low FN => False declaration of normal operation reduced. Reduce missed replacements. Tool replacements increased. *** Product quality not compromised.

limitiongs of the slecting model for evaluation

Raffin et al. (2021) – quote from paper - intro section – "A major challenge is that small implementation details can have a substantial effect on performance often greater than the difference between algorithms (Engstrom et al., 2020). It is particularly important that implementations used as experimental baselines are reliable; otherwise, novel algorithms compared to weak baselines lead to inated estimates of performance improvements"

DPO paper - Rafael Rafailov May-2023 — Direct Preference Optimization: Your Language Model is Secretly a Reward Model

5.2 Instability of Actor-Critic Algorithms We can also use our framework to diagnose instabilities with standard actor-critic algorithms used for the RLHF, such as PPO. We follow the RLHF pipeline and focus on the RL fine-tuning step outlined in Section 3. We can draw connections to the control as inference framework [18] for the constrained RL problem outlined in 3. We assume a parameterized model and minimize This is the same objective optimized in prior works [45, 35, 1, 23] using the DPO-equivalent reward for the reward class of In this setting, we can interpret the normalization term in as the soft value function of the reference policy . While this term does not affect the optimal solution, without it, the policy gradient of the objective could have high variance, making learning unstable. We can accommodate for the normalization term using a learned value function, but that can also be difficult to optimize. Alternatively, prior works have normalized rewards using a human completion baseline, essentially a single sample Monte-Carlo estimate of the normalizing term. In contrast the DPO reparameterization yields a reward function that does not require any baselines.

6 Conclusion

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.

Berend Denkena, Heinrich Klemme, and Tobias H. Stiehl. Tool wear monitoring using process data of multiple machine tools by means of machine learning. *ZWF Zeitschrift fuer Wirtschaftlichen Fabrikbetrieb*, 118(5):298 – 301, 2023. doi: 10.1515/zwf-2023-1059.

Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. https://github.com/openai/baselines, 2017.

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.

Laura Graesser and Wah Loon Keng. *Foundations of deep reinforcement learning*. Addison-Wesley Professional, 2019.

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.

Rousslan Fernand Julien; Raffin Antonin; Kanervisto Anssi; Wang Weixun Huang, Shengyi; Dossa. The 37 implementation details of proximal policy optimization. March 2022. Accessed: 2023-06-27.

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.

Taisuke Oshida, Tomohiro Murakoshi, Libo Zhou, Hirotaka Ojima, Kazuki Kaneko, Teppei Onuki, and Jun Shimizu. Development and implementation of real-time anomaly detection on tool wear based on stacked lstm encoder-decoder model. *The International Journal of Advanced Manufacturing Technology*, pages 1–16, 2023.

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

Biyao Qiang, Kaining Shi, Ning Liu, Junxue Ren, and Yaoyao Shi. Integrating physics-informed recurrent gaussian process regression into instance transfer for predicting tool wear in milling process. *Journal of Manufacturing Systems*, 68:42 – 55, 2023. doi: 10.1016/j.jmsy.2023.02.019.

Bo Qin, Yongqing Wang, Kuo Liu, Shaowei Jiang, and Qi Luo. A novel online tool condition monitoring method for milling titanium alloy with consideration of tool wear law. *Mechanical Systems and Signal Processing*, 199, 2023. doi: 10.1016/j.ymssp.2023.110467.

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.

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.

Paweł Twardowski, Jakub Czyżycki, Agata Felusiak-Czyryca, Maciej Tabaszewski, and Martyna Wiciak-Pikuła. Monitoring and forecasting of tool wear based on measurements of vibration accelerations during cast iron milling. *Journal of Manufacturing Processes*, 95:342 – 350, 2023. doi: 10.1016/j.jmapro.2023.04.036.

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.

LI Yingguang, LIU Changqing, LI Dehua, HUA Jiaqi, and WAN Peng. Tool wear dataset of nuaa ideahouse, 2021. URL https://dx.doi.org/10.21227/3aa1-5e83.