Proceedings of Machine Learning Research **318** (2026)
The 39th Canadian Conference on Artificial Intelligence

Towards Optimizing Proximal Policy Optimization PPO through Supervised Model-Support

Abdallah Alfaham[‡,◇,*],
[‡] Dept. of Electrical Engineering and Computer Science, University of Liège, Liège, Belgium
[◇] AI Engineer, SmartEye BV, Telecommunications service provider, Antwerp, Belgium

Abstract

Reinforcement Learning enables agents to learn behaviors by interacting with the environment and maximizing cumulative rewards. Model-free methods are widely used for their simplicity and flexibility, but often suffer from slow convergence as they solely rely on trial-and-error learning without knowledge of environment dynamics. Proximal Policy Optimization (PPO) is a popular on-policy algorithm that collects data using its current policy. However, because PPO relies on freshly sampled trajectories, it has limited ability to reuse past experiences, which can lead to repeatedly exploring suboptimal behaviors and slow policy improvement. To address this, we present Model-Support (MS), a supervised assistant that maintains model-free learning principles while improving efficiency. MS learns state-action pairs from high-return trajectories and serves as a supplementary policy that clones high-performing behaviors. While the agent explores broadly, the MS policy samples meaningful actions based on those behaviors. This combination leads to greater diversity of actions by mixing broad sampling from the agent's actor with focused sampling from the MS policy. MS acts as a form of local memory, capturing high-reward trajectories and guiding exploration toward promising regions that the agent policy might overwrite or miss. Although PPO uses advantage estimates to emphasize better actions within sampled data, it does not explicitly prioritize high-return trajectories. Consequently, suboptimal experiences still influence learning, weakening valuable signals and slowing convergence. This highlights the role of MS in preserving and cloning high-return behaviors to guide exploration and accelerate convergence.

Keywords: Deep RL, Model-Free methods, Proximal Policy Optimization (PPO) , Supervise Learning

1. Introduction

Scientific research constantly aims to automate software systems using machine learning to develop data-driven models capable of solving complex problems without human intervention. These models ultimately generate their own rules and make precise decisions[1]. The key factors for achieving effective ML models include data efficiency, fast algorithms enabled by parallel training, and well-designed architectures[2]. Our approach focuses on sampling efficient actions during agent-environment interaction, collecting high-return trajectories for model-support (MS) which is trained in parallel to accelerate the learning process and improve convergence.

The increasing interest and recent studies in the field of RL have led to implementing many methods that are able to interact with the environment with high efficiency. In addition, the integration of this approach with deep learning led to the emergence of the concept deep reinforcement learning (deep RL)[3]. These algorithms are generally classified into two main categories: model-free methods[4] and model-based methods[5].

In model-free methods, the agent starts with random actions and learns a policy solely through direct interaction with the environment by maximizing expected returns[6]. While in model-based methods, the agent can benefit from prior information about the environment's dynamics to plan its actions[5]. This prior information can come from learned transition models[7], expert knowledge or mathematical equations describing the experiment[8].

[*] abdallah.alfaham@uliege.be, abdallah.alfaham@smarteye.eu

However, building transition models remains challenging, computationally intensive, and time-consuming, particularly when dealing with high-dimensional dynamics.

In this paper, we introduce a new approach to improve the performance of model-free methods without building transition models, knowing environment dynamics, or using expert knowledge. Instead, we enhance the exploration efficiency of the behavior policy by integrating a model-support policy that emphasizes high-return trajectories from agent-environment interactions. At each state, one of the two policies, i.e. the agent or the MS is selected to sample an action for the current environment state. Unlike ensemble learning that uses voting or aggregation mechanisms, which can lead to preferential bias toward one dominated behavior, we sample directly from the selected policy to preserve exploration diversity.

2. **Related Work**

In the literature, various studies on behavior cloning have been introduced to improve the learning speed of model-free methods, such as the work of Goecks[9] and Lee[10]. A common theme in these studies is the use of expert knowledge, where human input is used to enhance the training of RL agents. In these approaches, the agent's loss function is often modified by measuring the discrepancy between the actions sampled by the agent and those provided by the expert[9].

Integrating behavior cloning with expert input can be viewed as a fusion of Human-Centric (HC) and Robot-Centric sampling[11]. Studies like DAgger[10, 11] implement this fusion by employing a supervised learning approach to clone the behavior of an expert, where dataset aggregation (DAgger) method iteratively trains a policy on state-action pairs collected from an expert, then queries the expert for corrections.

Inspired by DAgger technique, in which an expert provides data to train a policy through a supervised manner, our approach aims to build a model that clones the high-performance of the RL agent, guided by the rewards associated with those performed trajectories.

SVRPG study[12] (Stochastic Variance-Reduced Policy Gradient) is an on-policy reinforcement learning algorithm for improving sampling by reducing the variance of policy gradient estimates through a combination of full-gradient computations and corrected mini-batch updates, resulting in unbiased and more stable policy updates. Nevertheless, SVRPG is computationally costly due to the full gradient computations and lacks the simplicity and robustness of PPO, which achieves stable performance using lightweight, bias-tolerant clipping.

The study by Meng[13] aims to enhance PPO performance by changing its behavior policy to an off-policy one, thereby reducing interaction with the environment. It relies on pre-collected transitions under a behavior policy $\mu(a|s)$, and uses importance sampling to stabilize learning for the target policy $\pi(a \mid s)$. This approach shifts PPO away from its original on-policy formulation and alters its foundational principles. In comparison, our MS improves PPO without abandoning its on-policy core, instead augmenting the behavior policy with a supervised assistant policy trained on high-return trajectories. Importantly, the PPO implementation remains unchanged, and when MS is inactive, PPO_{MS} behaves exactly like the original PPO.

Other studies highlight the promising potential of integrating supervised learning technique into reinforcement learning context[14]. The main challenge was to select the suitable model capacity in terms of regularization and architecture which corresponds to the pre-collected transition data and choosing information critical for performance to condition on (e.g., goals or rewards). The study[15] builds upon this concept by incorporating immediate targets to guide the agent towards optimal trajectories. Similarly, Lee's work[16] highlights as well the ability to learn in-context for decision-making problems.

These studies rely on pre-collected transition data from RL experiments and focus on substituting the RL agent with a surrogate supervised model. In comparison, our approach strives to incorporate supervised learning within model-free reinforcement learning to improve performance and enhance the sampling process without relying on pre-collected transitions.

3. Background

We begin with the Markov Decision Process (MDP) as a formal framework used to model decision-making problems under uncertainty. It is defined by a tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ denotes the set of all possible states, and $\mathcal{A}$ represents the set of available actions. The function $P(s'|s, a)$ defines the probability of transitioning to state s' given that the agent is in state s and takes action a. The reward function $r(s, a)$ specifies the immediate reward received after performing action a in state s. Lastly, the discount factor $\gamma \in [0, 1[$ determines the importance of future rewards in the agent's decision-making process[17]. At each time step t, the agent observes a state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$, receives a reward $r(s_t, a_t)$, and transitions to a new state s_{t+1} according to $P(s_{t+1}|s_t, a_t)$. The objective is to learn a policy $\pi(a|s)$ that maximizes the expected cumulative discounted reward as shown in 3.1:

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) \right] \tag{3.1}$$

Policy-based methods optimize this objective by adjusting the parameters θ of the stochastic policy $\pi_\theta(a|s)$. One widely used approach is PPO, which employs a *clipped surrogate objective* to constrain abrupt policy updates and promote more stable training[18].

The clipped objective of PPO is defined as follows 3.2:

$$L_t^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t,\, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{3.2}$$

Where, $r_t(\theta)$ is the probability ratio between the new (target) policy and the old (behavior) policy 3.3:

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} \tag{3.3}$$

θ refer to the parameters of the *agent*, $\hat{A}_t$ is an estimate of the advantage function at time t, and ϵ is a small constant $= 0.2$ determines the clipping range[18].

The clipping mechanism prevents drastic policy changes by restricting the ratio $r_t(\theta)$ within $[1-\epsilon, 1+\epsilon]$ to improve stability.

In practice, PPO combines this clipped objective with a value function loss and an entropy bonus to form the full training objective 3.4:

$$L_t^{\text{PPO}}(\theta) = L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t) \tag{3.4}$$

The term $L^{\text{VF}}(\theta)$ is the squared error of the agent$_{critic}$ between the predicted state value and the target value 3.5:

$$L_t^{\text{VF}} = \left(V_\theta(s_t) - V_t^{\text{targ}}\right)^2 \tag{3.5}$$

$S[\pi_\theta]$ is the entropy of the policy. The constants $c_1 = 0.5$ and $c_2 = 0.01$ are weighting coefficients that balance the contribution of the value function loss and entropy bonus[18].

4. Methodology

Our approach seeks to enhance the performance of model-free methods by promoting more efficient exploration within the behavior policy through the integration of Model Support (MS), an auxiliary policy designed for sampling actions. MS is a neural network trained in

a supervised manner to associate state-action pairs and provides a distribution for sampling actions based on the agent's high-yield behavior.

The optimal MS policy is the one that best mimics the mean actions in high-return trajectories and minimizes the error between its predicted actions and those taken in the best-performing episodes 4.1:

$$\pi^*_{\theta'} = \arg\min_{\pi_{\theta'}} \mathbb{E}_{(s,a)\sim Dataset_{MS}} \left[\mathcal{L}\left(\pi_{\theta'}(a \mid s), a\right)\right] \tag{4.1}$$

Where θ' refers to the parameters of MS policy, and $\mathcal{L}$ represents MSE loss between the stored and the predicted actions.

After activating MS, the agent's behavior policy becomes as follows 4.2:

$$\pi^{\text{mixture}}_{\theta_{\text{old}},\theta'}(a_t \mid s_t) = \begin{cases} \pi_{\theta_{\text{old}}}(a_t \mid s_t), & \text{with probability } p \\ \pi_{\theta'}(a_t \mid s_t), & \text{with probability } 1-p \end{cases} \tag{4.2}$$

As we can notice from 4.2, when the MS policy is inactive, PPO$_{MS}$ behaves identically to the original PPO.

We adopt a switching strategy with high probability p to maintain the agent policy as primary while using the MS policy occasionally for targeted exploration. Further details are provided in the experimental setup section 5

This mixture of actions affects the learning process, as highlighted in study[17], where actions influence not only the immediate reward but also the subsequent state, thereby affecting all future rewards.

In line with this, our contribution emphasizes the nature of policies, where the behavior policy is designed to encourage diverse and exploratory actions, while the target policy is more deterministic to ensure stable and reliable performance[17].

An important aspect of RL agents is their learning approach, which can be either off-policy or on-policy[19]. The key difference is that in off-policy learning, the agent may update a policy different from the one used for action selection, meaning the behavior policy and target policy can differ. In contrast, on-policy learning focuses on evaluating and improving the same policy the agent uses for action selection, where the behavior and target policies are identical [18].

Similar to the probability ratio used in PPO, importance sampling is employed in PPO$_{MS}$ to address the mismatch between behavior and target policies, enabling the agent to learn from data generated by a different policy while still accurately estimating the value of the target policy.

$$r_t(\theta,\theta') = \frac{\pi_\theta(a_t \mid s_t)}{\pi^{\text{mixture}}_{\theta_{\text{old}},\theta'}(a_t \mid s_t)} \tag{4.3}$$

Hence, the surrogate objective is updated as shown in 4.4:

$$L^{CLIP}_t(\theta,\theta') = \hat{\mathbb{E}}_t\big[\min\Big(r_t(\theta,\theta')\hat{A}_t, \text{clip}\left(r_t(\theta,\theta'), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\Big)\big] \tag{4.4}$$

In order to check the convergence in PPO$_{MS}$, we calculate the penalty L^{E} which is the mean squared error between the behavior policy and the target policy. This is computed by measuring the discrepancy between individual elements in the action space. For mixture policy (i.e., behavior policy), actions can be directly retrieved from the replay buffer, while for the actor policy (i.e., target policy), representative actions are computed by passing the stored states in the replay buffer to the actor policy.

$$L^{\text{E}}_t(\theta,\theta') = \frac{1}{n}\cdot\sum_{n=0}^{space}\Big(\pi_\theta(a_t \mid s_t)_{[e_n]} - \pi^{\text{mixture}}_{\theta_{\text{old}},\theta'}(a_t \mid s_t)_{[e_n]}\Big)^2 \tag{4.5}$$

where e is an action element.

The penalty term L^{E} is then incorporated into the objective function as a regularization term to encourage alignment between the behavior and target policies.

$$L_t^{\mathrm{PPO}_{MS}}(\theta, \theta') = L_t^{\mathrm{CLIP}}(\theta, \theta') - c_1 L_t^{\mathrm{VF}}(\theta) + c_2 S[\pi_\theta](s_t) - c_3 L_t^{\mathrm{E}}(\theta, \theta') \tag{4.6}$$

where $c3 = 0.2$ to regulate the impact of the penalty within the objective.

The effect of L^{E} is analyzed in section 8 and illustrated in fig 6.

We consider actor-critic agents[20] as shown in fig 1. The interaction loop terminates when a stopping condition is met, such as reaching a maximum number of timesteps or an environmental termination. A complete interaction loop is called an *episode*, and the cumulative episode reward is called the *return*.

During training, experiences are stored in a fixed-size replay buffer, where rare or high-reward transitions may be overwritten, and redundant samples can reduce exploration and cause overfitting.

We address this by training MS in parallel with the agent, using a selective dataset of state-action pairs from high-return episodes.

A mutual optimization process emerges between the agent and MS. MS starts with no environmental knowledge and learns from the agent's interactions. MS then helps with focused, performance-oriented action sampling, which guides exploration and accelerates convergence as shown in result section 6.

For our experiments, we use PPO with neural network approximators for both the actor and critic[18]. We compare PPO and PPO_{MS} across different scenarios to evaluate the contribution of MS in improving behavior policy.

As shown in fig 1, after inserting MS, actions during training can be sampled either from the agent or MS. MS can only be activated if the behavior of the agent meets the assigned $\mathrm{threshold}_{MS}$ as described in the experimental setup section 5.

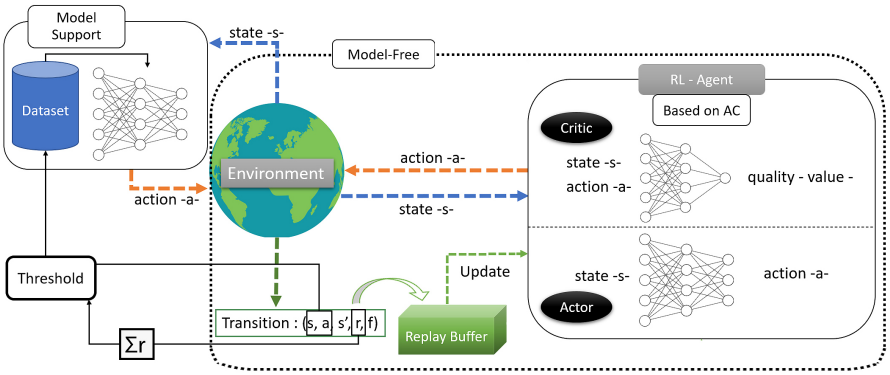

Figure 1. Here, (s, a, s', r, f) denote (state, action, state_{next},flag_{end}). The main replay buffer stores all episodes up to its fixed size, while $Dataset_{MS}$ keeps only high-reward episodes that exceed $\mathrm{threshold}_{MS}$, storing up to 60% of total transactions.

It is important to note the dependency between the agent and its MS. If the agent interacts poorly with the environment, then the transitions in the replay buffer will be inefficient, and highly selective MS i.e. with high threshold will reject most of these episodes, while a low threshold may accept them, risking inadequate learning. Thus, efficient agents are essential for properly training model-supports. See pseudocode 1 for details.

Data: transition records from environment
Result: training an agent & its model-support
set probability $p \in [0, 1]$;
set threshold$_{MS}$;
set epochs value;
initialize the RL environment;
while *not reaching a termination flag* **do**
  if *MS is not enabled* **then**
    check model-support conditions;
  end
  $\pi \leftarrow$ select(agent, MS, [p]);
  action $\leftarrow \pi(action \mid state)$;
  state$_{next}$, reward, flag $\leftarrow$ env(action);
  $sum_{rewards} \leftarrow sum_{rewards} + reward$;
  Replay $\leftarrow$ (state, action, state$_{next}$, reward, flag);
end
if $sum_{rewards} \geq threshold_{MS}$ **then**
  $Dataset_{MS} \leftarrow (states, actions)$;
end
if $(current_{step} \bmod update_{value}) = 0$ **then**
  for $K \leftarrow 1$ **to** *epochs* **do**
    update(agent, Replay$_{Buffer}$);
    update(MS,$Dataset_{MS}$);
  end
end

Algorithm 1: Pseudo code of a single episode of training stage with epoch updates.

At each timestep, an action is sampled from a policy chosen based on probability p, and the transition is stored in the replay buffer. If the episode return exceeds threshold$_{MS}$, its state-action pairs are added to $Dataset_{MS}$. The agent then is trained for certain epochs using $L_t^{\text{PPO}_{MS}}$ on replay buffer, while the model-support is updated by minimizing the mean squared error on $Dataset_{MS}$.

The Mean Squared Error (MSE) is used instead of Maximum Likelihood Estimation (MLE) because the model's output is the mean action, and the standard deviation is fixed during the update process. This makes MSE equivalent to MLE in this context, with the added benefit of simplification.

Our approach differs from ensemble learning, as we don't combine the sampling of two policies and then make a final prediction through methods like majority voting or aggregation. This allows us to maintain exploration, as we are not trying to influence the policy's opinion or bias it toward a specific behavior.

The PPO actor-critic has three Tanh-activated layers: the actor outputs mean actions, while the critic outputs a scalar value to evaluate action quality, and Tanh is used to match the action range $[-1, 1]$.

MS adopts the same architecture as the agent's actor to ensure structural neutrality, thus attributing any improvement to the use of the mixture policy. We also use a decay standard deviation technique for action sampling to shift from exploration to exploitation.

In continuous action space experiments, actions are sampled using the predicted mean actions and the assigned standard deviation, as shown in fig 2.

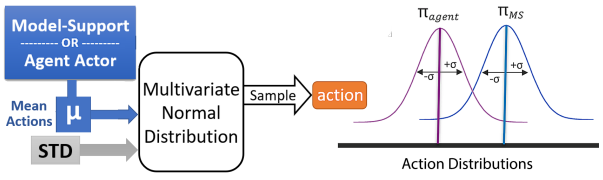

Figure 2. Steps for sampling an action from the agent or MS. The mean actions (μ) of these policies will differ, as they are trained on distinct datasets, though both distributions will use the same standard deviation (STD).

5. Experimental Setup

During training, we select a policy for sampling actions using a probability ranging from [0,1]. If p=0, actions are fully sampled from MS, and if p=1, actions come entirely from the gent. To prioritize the agent, we set p=0.95. Additionally, enabling MS depends on whether its Dataset$_{MS}$ is empty or not. A high threshold$_{MS}$ may delay MS activation, as the agent must first learn and obtain returns exceed that threshold.

For experiments, we used OpenAI Gym framework which is an open source Python library that has collections of reference environments for RL[21].

5.1. Experiments

For evaluation, we apply our methodology on different continuous experiments and compare it with standard PPO without adding model-support.

The experiments are:

- **Bipedal-Walker-v3**: A two-legged robot with two joints per leg trained to walk by applying torques.
- **Ant-v4**: A four-legged MuJoCo robot that learns to walk by controlling joint torques.
- **Hopper-v4**: A 2D robotic hopper that moves forward by applying torques to its hinges.
- **HalfCheetah-v4**: A 2D cheetah-like robot trained to run forward as fast as possible by applying torque to its joints.

The value of threshold$_{MS}$ can be varied based on the experiment. We therefore developed a systematic procedure based on the best return of standard PPO to determine the threshold value as shown in equation 5.1. This technique is inspired by study[22] to set the kernel parameters for vectorization with Deep Reinforcement Learning.

$$threshold_{MS} = argmax[factor \times PPO_{BestReturn}] \tag{5.1}$$

The range of the factor is [0..1[to make sure that the highest possible value of a threshold won't exceed PPO$_{BestReturn}$. The step in the range is 0.05, therefore, there are 20 different possible values for threshold$_{MS}$. We select the one that gives the best performance as showing in fig 3.

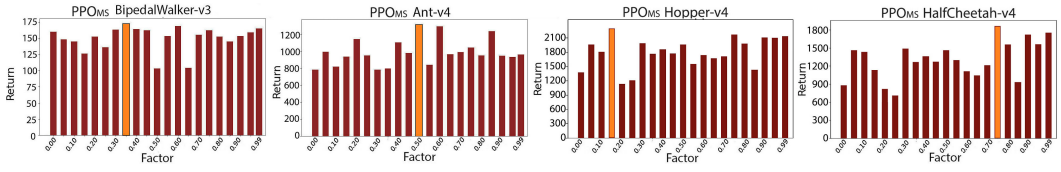

Figure 3. Determining Model-Support Thresholds.

We will also re-run the experiments with more greedy agents to check the effect of MS in critical cases.

6. Results - Training

The shaded high variance lines are the actual results while the bright lines are the results after smoothing. Max timesteps of training is 5×10^6 steps.

Table 1. Settings of Standard Deviation (STD).

Variables	Standard	Greedy
Initial STD	0.6	0.6
STD decay rate	0.05	0.08
STD decay freq	25 $\times 10^4 steps$	15 $\times 10^4 steps$
Minimum STD	0.1	0.01

6.1. **Case Study - Standard**

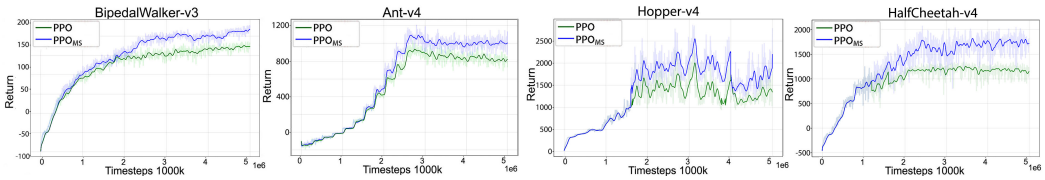

Figure 4. The complexity of the experiment plays an important role for determining the contribution of MS and the time of its activation.

As been shown in fig 4 and table 2, PPO$_{MS}$ outperforms PPO and provides better convergence during the training stage.

Table 2. Training - Final Average Return.

Experiments	PPO ↑	PPO$_{MS}$ ↑
BipedalWalker-v3	150.21	190.15
Ant-v4	800.32	1100.42
Hopper-v4	1450.77	2270.31
HalfCheetah-v4	1179.65	1730.74

We noticed that when the returns of the episodes do not reach the allocated threshold$_{MS}$, the Dataset$_{MS}$ will be empty and MS will not be enabled. Thus, sampling actions in PPO & PPO$_{MS}$ in that case will only be from agents and that explains why they have close behavior in these early timesteps.

7. **Results - Testing**

After training, model supports are removed, and the trained agents are tested over 500 episodes. For more challenging tests, noise is added to the environment states, consisting of two types:

- **Addition**: States are modified with noise in the range $[-0.1, 0.1]$, with a noiseless value of zero. adding noise will gradually removing the state's original properties.
- **Multiplication**: States are multiplied by noise within the range $[0.9, 1.1]$, with a noiseless value of one. This noise distorts the states and gradually alters their properties.

For further clarification and a detailed comparison of the two noise types, kindly check the abstract example in the Appendix.A.

At every timestep, one type of noise is added randomly to the environment conditions. The reason behind using the noise is to verify the performance and check the stability of the trained agents which is the capability to sustain high returns. We plotted different violin plots of the results as shown in Fig 5.

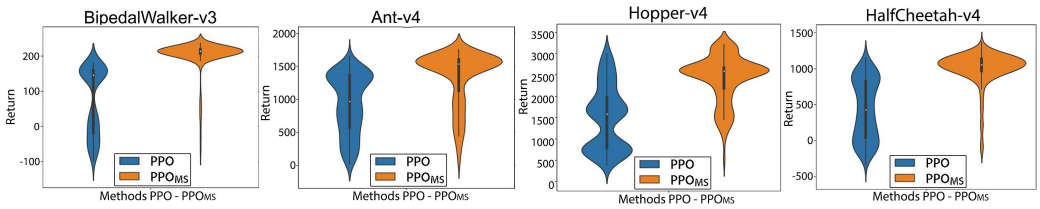

Figure 5. Testing trained agents on a large number of experiments reflects the positive effect of using MS.

We can conclude from these figs that the agents in PPO$_{MS}$ sustain higher performance during these episodes, and the number of episodes with high returns is greater compare to PPO. Table 3 presents a statistical summary of these results, where BipWalk and Hcheetah refer to BipedalWalker and HalfCheetah respectively.

Table 3. Statistical Summary of Testing.

Statistic	**BipWalk PPO**	**BipWalk PPO$_{MS}$**	**Ant PPO**	**Ant PPO$_{MS}$**	**Hopper PPO**	**Hopper PPO$_{MS}$**	**Hcheetah PPO**	**Hcheetah PPO$_{MS}$**
Mean	75.23	170.82	943.21	1243.21	1489.08	2420.57	432.1	1003.31
STD	94.52	46.05	446.44	410.44	734.89	550.64	398.04	243.98
Median	145.73	180.35	1163.53	1413.53	1581.2	2586.62	430.48	1056.09
Max	179.15	216.15	1531.31	1831.31	2995.92	3199.21	1152.86	1370.45
Min	-99.6	-78.34	-2.53	-2.03	381.64	445.08	-335.3	-165.6

8. **Analysis & Inferences**

The better performance and faster convergence of PPO$_{MS}$ can be attributed to several reasons:

- Creating a local memory for the agent by storing episodes based on high-yield performance promotes more effective reuse of past experiences.
- Enhancing learning process by combining reinforcement learning & supervised learning techniques to improve search space exploration.
- Having two different perspectives of sampling actions i.e. agent & MS can help to get out of local maxima.
- Balancing exploration and exploitation as the agent is trained using a replay buffer that captures recent and diverse interactions (i.e., fresh data), whereas the MS relies on a selectively stored dataset Dataset$_{MS}$, composed of high-performed trajectories.

If the behavior policy i.e. the mixture policy and the target policy i.e. the actor policy behave in the same way then the MSE between them will be close to zero and the ratio between them is close to one which is expected by the end of the training stage.

As shown in fig 6, there is a high error i.e., penalty in the early stages of training. However, over time, the use of importance sampling 4.3 and clipping technique 4.4 helps to progressively reduce these discrepancies.

We also compare the sampling and clipping ratios of the agent and MS policies to highlight the differences between them. As shown in table 4, there is a distinct contrast between the clipping ratios of the agent policy and the MS policy when compared to their respective sampling ratios. This reflects significant differences in their action distributions.

Despite MS's small sampling ratio, it has a higher clipping ratio (16–20%), suggesting that its policy updates involve larger changes within the behavior policy, even with the fewer actions it samples.

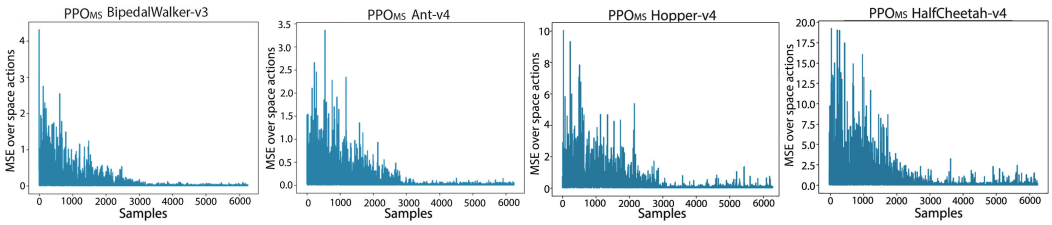

Figure 6. Analysing PPO_{MS} and computing the mean error between the B-Policy and T-Policy to measure the discrepancy.

Table 4. Sampling vs Clipping in PPO_{MS}.

Experiments	Sampling in PPO_{MS}		Clipping in PPO_{MS}	
	MS	Agent	MS	Agent
BipWalk	5%	95%	16.68%	83.32%
Ant	5%	95%	19.55%	80.45%
Hopper	5%	95%	15.84%	84.16%
Hcheetah	5%	95%	17.72%	82.28%

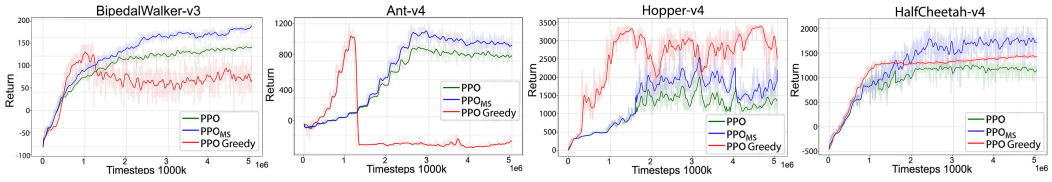

Figure 7. Combining PPO_{Greedy} with $\text{PPO}_{standard}$.

8.1. **Case Study - Greedy**

We set more challenging settings for PPO and made it more greedy by rapidly decreasing the value of the standard deviation as shown in table 1.

The performance of PPO_{Greedy} is plotted alongside the previous experiments in fig 4 to check the effects after changing STD.

As shown in fig 7, PPO_{Greedy} converges faster but suffers from instability, with significant performance drops. This results from excessive exploitation, which limits exploration and repeatedly samples early-learned actions, leading to underfitting. We attempt to incorporate model-support into PPO_{Greedy} after updating training parameters, using strategies like early stopping when the agent reaches peak performance.

8.2. **Model-Support on Greedy PPO**

The following fig 8 shows better performance after adding model-support to PPO_{Greedy} . This enhancement shows the potential of model-support to optimize the efficiency of model-free methods even under critical settings as here we only have limited number of timesteps for training with this bias greedy model-free method.

9. **Conclusion**

In this paper, we introduced a novel approach to enhance the performance of model-free reinforcement learning methods by integrating a supervised learning model, called model-support (MS), which autonomously learns from optimal agent-environment interactions and

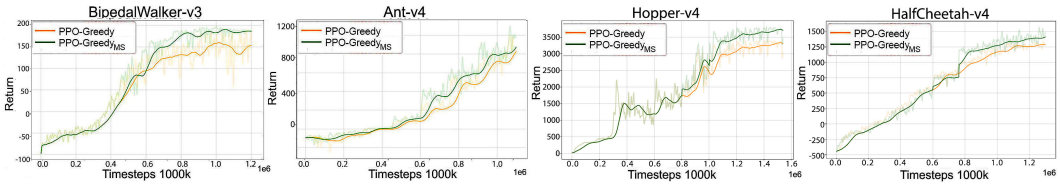

Figure 8. Positive contribution of model-support under critical greedy settings.

aims to build a form of local memory of the optimal trajectories performed by the agent, allowing it to remember best behaviors. Our findings suggest that this strategy has significant potential to advance the capabilities of model-free methods, which could open new avenues for addressing complex and dynamic environments and pave the way for more autonomous and scalable RL solutions.

10. **Future Work**

The next phase of our approach focuses on improving the adaptability of MS by incorporating dynamic settings like for example using dynamic thresholds that align with the agent's evolving behavior. This enhancement will make our method more generic and applicable across a wider range of RL tasks.

Appendix A. **Example of Noise Types**

Adding noise removes features, while multiplying by noise twists them. Assuming the state is a cosine function with linear noise, adding noise gradually turns the state into pure noise, while multiplying it by noise distorts the state, as illustrated in fig 9.

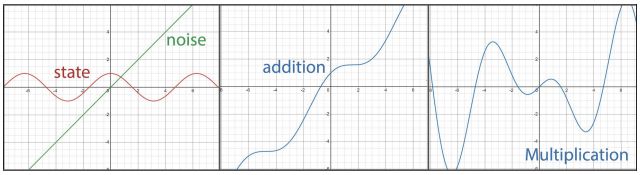

Figure 9. Abstract example of noise types.

References

[1] Z.-H. Zhou. *Machine learning*. Springer Nature, 2021.

[2] D. Morgan and R. Jacobs. "Opportunities and challenges for machine learning in materials science". In: *Annual Review of Materials Research* 50 (2020), pp. 71–103.

[3] A. Y. Majid, S. Saaybi, V. Francois-Lavet, R. V. Prasad, and C. Verhoeven. "Deep reinforcement learning versus evolution strategies: a comparative survey". In: *IEEE Transactions on Neural Networks and Learning Systems* (2023).

[4] L. Zhang, Q. Zhang, L. Shen, B. Yuan, X. Wang, and D. Tao. "Evaluating model-free reinforcement learning toward safety-critical tasks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 37. 12. 2023, pp. 15313–15321.

[5] T. M. Moerland, J. Broekens, A. Plaat, C. M. Jonker, et al. "Model-based reinforcement learning: A survey". In: *Foundations and Trends® in Machine Learning* 16.1 (2023), pp. 1–118.

[6] S. Çalışır and M. K. Pehlivanoğlu. "Model-free reinforcement learning algorithms: A survey". In: *2019 27th signal processing and communications applications conference (SIU)*. IEEE. 2019, pp. 1–4.

[7] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, et al. "Mastering atari, go, chess and shogi by planning with a learned model". In: *Nature* 588.7839 (2020), pp. 604–609.

[8] X.-Y. Liu and J.-X. Wang. "Physics-informed Dyna-style model-based deep reinforcement learning for dynamic control". In: *Proceedings of the Royal Society A* 477.2255 (2021), p. 20210618.

[9] G. Goecks, G. M. Gremillion, V. J. Lawhern, J. Valasek, and N. R. Waytowich. "Integrating Behavior Cloning and Reinforcement Learning for Improved Performance in Dense and Sparse Reward Environments". In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS '20)*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2020, pp. 465–473.

[10] Z. E. Lee and K. M. Zhang. "Generalized reinforcement learning for building control using Behavioral Cloning". In: *Applied Energy* 304 (2021), p. 117602.

[11] M. Laskey, C. Chuck, J. Lee, J. Mahler, S. Krishnan, K. Jamieson, A. Dragan, and K. Goldberg. "Comparing human-centric and robot-centric sampling for robot deep learning from demonstrations". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 358–365.

[12] M. Papini, D. Binaghi, G. Canonaco, M. Pirotta, and M. Restelli. "Stochastic Variance-Reduced Policy Gradient". In: *Proceedings of the 35th International Conference on Machine Learning*. Ed. by J. Dy and A. Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018, pp. 4026–4035. URL: https://proceedings.mlr.press/v80/papini18a.html.

[13] W. Meng, Q. Zheng, G. Pan, and Y. Yin. "Off-policy proximal policy optimization". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 37. 8. 2023, pp. 9162–9170.

[14] S. Emmons, B. Eysenbach, I. Kostrikov, and S. Levine. "RvS: What is Essential for Offline RL via Supervised Learning?" In: *International Conference on Learning Representations (ICLR)*. 2022. URL: https://openreview.net/forum?id=cfBkQZkQ8fM.

[15] A. Badrinath, Y. Flet-Berliac, A. Nie, and E. Brunskill. "Waypoint transformer: Reinforcement learning via supervised learning with intermediate targets". In: *Advances in Neural Information Processing Systems* 36 (2024).

[16] J. Lee, A. Xie, A. Pacchiano, Y. Chandak, C. Finn, O. Nachum, and E. Brunskill. "Supervised pretraining can learn in-context reinforcement learning". In: *Advances in Neural Information Processing Systems* 36 (2024).

[17] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[19] R. Fakoor, P. Chaudhari, and A. J. Smola. "P3o: Policy-on policy-off policy optimization". In: *Uncertainty in Artificial Intelligence*. PMLR. 2020, pp. 1017–1027.

[20] V. Konda and J. Tsitsiklis. "Actor-critic algorithms". In: *Advances in neural information processing systems* 12 (1999).

[21] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. "Openai gym". In: *arXiv preprint arXiv:1606.01540* (2016).

[22] A. Haj-Ali, N. K. Ahmed, T. Willke, Y. S. Shao, K. Asanovic, and I. Stoica. "Neurovectorizer: End-to-end vectorization with deep reinforcement learning". In: *Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization*. 2020, pp. 242–255.