ECAI 2025
I. Lynce et al. (Eds.)

doi:10.3233/FAIA251138

SHAC++: A Neural Network to Rule All Differentiable Simulators

Francesco Bertolotti[a], Gianluca Aguzzi[b], Walter Cazzola[c] and Mirko Viroli[b]

[a]Domyn, Milan, Italy
[b]Department of Computer Science and Engineering, University of Bologna, Bologna, Italy
[c]Department of Computer Science, University of Milano, Milano, Italy

Abstract. Reinforcement learning (RL) algorithms show promise in robotics and multi-agent systems but often suffer from low sample efficiency. While methods like SHAC leverage differentiable simulators to improve efficiency, they are limited to specific settings: they require fully differentiable environments, including transition and reward functions, and have primarily been demonstrated in single-agent scenarios. To overcome these limitations, we introduce SHAC++, a novel framework inspired by SHAC. SHAC++ removes the need for differentiable simulator components by using neural networks to approximate the required gradients, training these networks alongside the standard policy and value networks. This enables the core SHAC approach to be applied in both non-differentiable and multi-agent environments. We evaluate SHAC++ on challenging multi-agent tasks from the VMAS suite, comparing it against SHAC (where applicable) and PPO, a standard algorithm for non-differentiable settings. Our results demonstrate that SHAC++ significantly outperforms PPO in both single- and multi-agent scenarios. Furthermore, in differentiable environments where SHAC operates, SHAC++ achieves comparable performance despite lacking direct access to simulator gradients, thus successfully extending SHAC's benefits to a broader class of problems. The full implementation is openly available at https://github.com/f14-bertolotti/shacpp.

1 Introduction

Learning control policies is crucial for applications such as robotics [51], autonomous driving [20], and swarm intelligence [53]. These domains often present challenges including sparse rewards, complex dynamics, and multi-agent interactions. Many current RL approaches focus specifically on either single- or multi-agent settings, lacking a unified solution. Furthermore, RL algorithms are often *sample inefficient*, requiring extensive interaction with the environment to learn effective policies.

A promising direction to improve sample efficiency involves utilizing *differentiable simulators*, namely simulators that allow backpropagation through their components. SHAC [60] exemplifies this approach by using gradients obtained directly from the simulator to accelerate policy optimization. However, SHAC's applicability is constrained by two key limitations. First, it necessitates both a differentiable transition function *and* a differentiable reward function, which is often impractical, especially in the presence of sparse or complex reward structures. Second, while powerful, backpropagation through complex dynamics, such as object collisions [32] or intricate multi-agent interactions, can lead to unstable gradients [10, 43], potentially hindering learning. The original SHAC work also focused primarily on single-agent tasks.

In this work, we propose SHAC++ to address these limitations and extend gradient-based policy optimization beyond SHAC's constraints. Our goal is to develop a sample-efficient reinforcement learning framework that maintains the benefits of gradient-based optimization without requiring differentiable simulators. Drawing inspiration from model-based RL [22, 23], SHAC++ substitutes direct simulator differentiation with neural approximations by training networks that approximate the transition and reward functions concurrently with the policy function. This approach preserves the benefits of gradient information while eliminating strict differentiability requirements. It also enhances robustness to complex dynamics and enables application in both non-differentiable environments and multi-agent scenarios, albeit with additional computational overhead compared to SHAC.

We evaluate SHAC++ against two established baselines: PPO [48] (including its multi-agent variant MAPPO [21]), as the standard for non-differentiable environments, and the original SHAC [60], where applicable. Our experiments utilize the VMAS simulator [11], which provides physics-based multi-agent environments with varying levels of complexity. Through this comparative analysis on cooperative tasks with different agent populations, we investigate the following research questions:

- **RQ$_1$** Can we train a neural network to approximate the gradients of a differentiable simulator?
- **RQ$_2$** How does our algorithm compare to PPO/MAPPO and SHAC in both single-agent and multi-agent settings?
- **RQ$_3$** How does the performance of these algorithms change as the search space increases?

Our contributions include:

- a novel RL framework (SHAC++) that employs learned gradient approximations, eliminating the need for differentiable simulators;
- the first empirical evaluation of SHAC in multi-agent environments, establishing a baseline for differentiable MARL;
- a comprehensive comparison across environments with varying agent counts and differentiability properties;
- experimental evidence demonstrating that SHAC++ matches SHAC's performance in differentiable settings while substantially outperforming PPO in sample efficiency and final reward across both single-agent and complex multi-agent scenarios.

2 Background & Notation

2.1 Markov Decision Processes

To provide context for our work, we first introduce the reinforcement learning framework. Particularly, we focus on multi-agent settings, where multiple agents interact with each other and the environment. In doing so, each task is then modeled as a partially observable, decentralized, finite-horizon, Markov Decision Process (MDP) [49], $\mathcal{M}$: $(\mathcal{N}, \mathcal{S}, \mathcal{O}, \mathcal{A}, \pi, \mathcal{R}, F, H, \gamma, \mu)$. Where $\mathcal{N} = \{1, 2, \dots, n\} = [n]$ and $\mathcal{S}$ denote the set of agents and the state space, respectively. $\mathcal{O} = \{\mathcal{O}_i\}_{i\in\mathcal{N}}$ and $\mathcal{A} = \{\mathcal{A}_i\}_{i\in\mathcal{N}}$ are the observation space and action space for each agent. $\{P_i : \mathcal{S} \to \mathcal{O}_i\}_{i\in\mathcal{N}}$ are the projections from the state to the observation space and $\{R_i : \mathcal{S} \times \mathcal{A}_i \times \mathcal{S} \to \mathbb{R}\}_{i\in\mathcal{N}}$ are the reward functions for each agent. $F : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ denotes the transition function. γ and H denote the discount factor and the time horizon respectively. Finally, μ is the initial state distribution.

Further, let us denote the space of trajectories, $\mathcal{T} = (\mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathbb{R}^n)^*$, where each trajectory consists of a sequence of: starting state, action per agent, subsequent state, and reward per agent. Given a generic trajectory $\tau \in \mathcal{T}$, we denote its i-th starting state with s_i^τ, its i-th action for agent j with a_{ij}^τ, and its i-th reward for agent j with r_{ij}^τ.

A policy for agent i is a function $\pi_i : \mathcal{O}_i \to \mathcal{A}_i$ that maps its observation space to its action space. We will denote the collective policy, determined by the collection of agent-specific policies simply with $\pi = \{\pi_i\}_{i\in\mathcal{N}}$. We can use the policy to sample trajectories from the MDP by iteratively applying the transition, reward, projection and policy functions. In this case, for trajectory τ, we have that the i-th action for the j-th agent is $a_{ij}^\tau = \pi_j(P_j(s_i^\tau))$ and the next state is $s_{i+1}^\tau = F(s_i^\tau, a_{i1}, \dots, a_{in})$. The i-th reward for the j-th agent is $r_{ij}^\tau = R_j(s_i^\tau, a_{ij}^\tau, s_{i+1}^\tau)$. Finally, s_0 is sampled from the initial state distribution μ. For brevity, we will simply denote with $\tau \sim \pi$ a trajectory sampled following the policies π.

The goal of the MDP is to find a policy that maximizes the expected return

$$\mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{H-1}\sum_{i\in\mathcal{N}}\gamma^t r_{ti}^\tau\right]$$

2.2 Neural Networks

A neural network is a function $f_\theta : \mathbb{R}^n \to \mathbb{R}^m$ parameterized by its weights $\theta \in \mathbb{R}^w$, which maps an input vector $x \in \mathbb{R}^n$ to an output vector $y \in \mathbb{R}^m$. The network parameters θ are optimized by minimizing the loss function $\mathcal{L}_\theta$, which quantifies the discrepancy between the network's predictions and the target outputs based on the training data.

In this work, we will often approximate functions (such as transition F) with neural networks. Therefore, for each function g, we will denote the corresponding neural network with g_θ. For instance, the transition and its corresponding neural network are denoted as F and F_θ, respectively. The corresponding loss will be denoted as $\mathcal{L}_\theta^g$ and it will aim to minimize the differences between g and g_θ.

Throughout this notation, we use a single global set of parameters θ to describe all networks. Each network can either use its own distinct subset of θ or share parameters with other networks. In practice, parameter sharing occurs only among the policy networks.

To keep the notation simple, we formally consider only the case where policies have access to the full state, meaning that P_i is the identity function. Extending our framework to handle partial observability is straightforward. Nonetheless, our experiments do include scenarios where policies do not have access to the complete state information.

2.3 SHAC

Short Horizon Actor-Critic (SHAC) is a model-based RL algorithm designed to leverage differentiable simulators for efficient policy learning. It alternates between rolling out short trajectories (a fixed small horizon h) in parallel and updating an actor–critic pair using gradients that flow directly through the simulator's dynamics and reward model.

In its trivial multi-agent extension, SHAC trains policies $\pi_{1\theta}, \dots, \pi_{n\theta}$ by minimizing the loss function $\mathcal{L}_\theta^\pi : \mathcal{T}^N \to \mathbb{R}$ for a batch of N trajectories $B = \{\tau_1, \dots, \tau_N\}$ sampled using π. The loss $\mathcal{L}_\theta^\pi(B)$ is defined as:

$$\frac{1}{Nnh}\sum_{\tau\in B}\sum_{t=t_0}^{t_0+h-1}\sum_{i\in\mathcal{N}}\gamma^{t-t_0}R(s_t^\tau, a_{ti}^\tau) + \gamma^h V_\theta(s_{t_0+h}^\tau)$$

where $h \ll H$ is the rollout size (namely the number of steps used to estimate the return), t_0 is the initial time step of the rollout, $V_\theta : \mathcal{S} \to \mathbb{R}$ is the value function estimating the expected return. It is apparent that SHAC assumes that the policy, reward, value, projection, and transition functions are all differentiable, i.e., $\{\pi_i\}_{i\in\mathcal{N}}, \{R_i\}_{i\in\mathcal{N}}, V_\theta$, and $F \in \mathcal{C}^1$. While this is a common requirement for policy and value functions, differentiable transition and reward functions are rarely available.

The value function V_θ is trained to approximate a temporal difference-λ formulation [52] V, which in SHAC translates to minimize the loss $\mathcal{L}_\theta^V : \mathcal{T}^N \to \mathbb{R}$.

$$\mathcal{L}_\theta^V(B) = \frac{1}{Nh}\sum_{\tau\in B}\sum_{t=0}^{h}||V_\theta(s_t^\tau) - V(s_t^\tau)||^2$$

$$V(s_t) = (1-\lambda)\left(\sum_{k=1}^{h-t-1}\lambda^{k-1}G_t^k\right) + \lambda^{h-t-1}G_t^{h-t}$$

$$G_t^k = \left(\sum_{l=t}^{k-1+t}\sum_{i\in\mathcal{N}}\gamma^{l-t}r_{li}\right) + \gamma^k V(s_{t+k})$$

G_t^k represents the discounted sum of rewards up to step k, plus the bootstrapped value of the state at step $t+k$. The estimates V is computed by discounting longer (higher k) discounted sums, G_t^k.

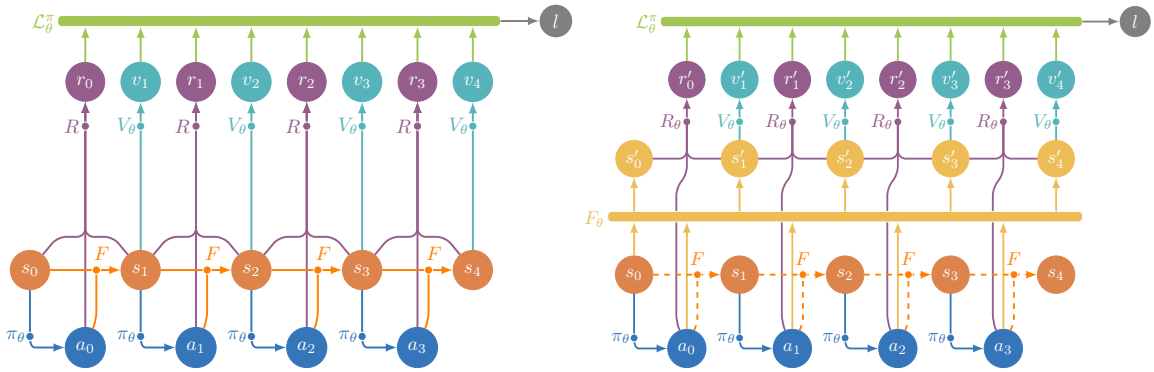
Finally, Fig. 1a presents an unrolled view of the SHAC algorithm for a single-agent scenario.

3 SHAC++

As mentioned previously, we aim to address the limitations of SHAC by approximating the gradients of a differentiable environment (R and F) with neural networks (R_θ and F_θ). In particular, we aim to train a neural network to approximate the reward function by minimizing:

$$\underset{\tau\sim\pi}{E}\left[\sum_{s_i, a_{ij}, s_{i+1}\in\tau}(R_{i\theta}(s_i, a_{ij}, s_{i+1}) - R_i(s_i, a_{ij}, s_{i+1}))^2\right]$$

In practice, this is achieved by leveraging a *replay buffer* to cache the state-action-reward triplets obtained from the simulator. The neural



(a) SHAC (b) SHAC++

Figure 1: Comparison between SHAC and SHAC++ during a rollout of 4 steps. Nodes with labels (such as r_i or r'_i) indicate values, while nodes without labels (such as π or R) represent functions. Dashed lines denote non-differentiability. a_i and r_i refer to the collection of actions and rewards for all agents. Note that the main advantage of SHAC++ is that the transition function F on the right can be non-differentiable, unlike the left F in SHAC, which must be differentiable.

Algorithm 1 SHAC++ minimal (no cache and no cool-down) pseudocode. $s_{:,0}$ denotes the first step of each trajectory in s. a and r denote the actions and rewards for all agents.

Initialize $\pi_\theta, V_\theta, R_\theta, F_\theta$ networks
Initialize learning rates $\eta_\pi, \eta_R, \eta_V, \eta_F$
Initialize environment $\mathcal{E}$
Initialize γ, α
for episode in $1, \dots, N_{\text{episodes}}$ **do**
  $s, a, r, v \sim \mathcal{E}(\pi)$
  $\hat{s} = F_\theta(s_{:,0}, a)$
  $\hat{r} = R_\theta(\hat{s})$
  $\hat{v} = V_\theta(\hat{s})$
  $\theta \leftarrow \theta - \eta_\pi \nabla \mathcal{L'}^{\pi}_{\theta}(\hat{r}, \hat{v}, \gamma, \alpha)$
  $\theta \leftarrow \theta - \eta_V \nabla \mathcal{L}^{V}_{\theta}(\hat{v}, s, v)$
  $\theta \leftarrow \theta - \eta_R \nabla \mathcal{L}^{R}_{\theta}(\hat{r}, s, a, r)$
  $\theta \leftarrow \theta - \eta_F \nabla \mathcal{L}^{F}_{\theta}(\hat{s}, s, a)$
end for

network is then trained to minimize the mean squared error (MSE) loss. Specifically, we minimize the loss $\mathcal{L}^{R_i}_{\theta} : (\mathcal{S} \times \mathcal{A}_i \times \mathcal{S})^N \to \mathbb{R}$:

$$\mathcal{L}^{R_i}_{\theta}(B) = \frac{1}{N} \sum_{s,a,s',r \in B} (R_{i\theta}(s,a) - r)^2$$

Where B represents a batch of state-action-state-reward tuples sampled uniformly from the cache.

Similarly, we aim to train a neural network to approximate the transition function. However, to avoid vanishing or exploding gradients, we employ a non-recursive approach similar to Action World Model [42]. This corresponds to train a neural network that given a starting observation from $\mathcal{S}$ and a sequence of h actions from $\mathcal{A}^h$, predicts the next h states from $\mathcal{S}^h$. We aim to minimize:

$$\underset{\tau \sim \pi}{E} \left[\sum_{\substack{s_t, \dots, s_{t+h+1}, \\ a_{t*}, \dots, a_{t+h*} \in \tau}} ||F_\theta(s_t, a_{t*}, \dots, a_{i+h*}) - [s_t, \dots, s_{t+h+1}]||^2 \right]$$

Where a_{t*} denotes the set of actions taken from all agents at time t, $\{a_{ti}\}_{i \in \mathcal{N}}$. In practice, this is achieved by caching trajectories and then training the neural network to minimize the MSE loss. That is, we minimize the loss $\mathcal{L}^{F}_{\theta} : \mathcal{T}^N \to \mathbb{R}$:

$$\frac{1}{Nh} \sum_{\tau \in B} ||F_\theta(s^\tau_0, a^\tau_{1*}, \dots, a_{h*}) - [s^\tau_1, \dots, s^\tau_h]||^2$$

Where B represents a batch of trajectories of length h sampled uniformly from the cache.

Further, we modify the SHAC policy loss, $\mathcal{L}^{\pi}_{\theta}$, to include a term that penalizes actions outside the action space. Suppose the action space is the open interval (l, r) for all agents, we minimize the loss $\mathcal{L'}^{\pi}_{\theta}$:

$$\mathcal{L'}^{\pi}_{\theta}(B) = \mathcal{L}^{\pi}_{\theta}(B) + \alpha \frac{1}{NH} \sum_{\tau \in B} \sum_{a \in \tau} d(a^\tau, (l, r))$$

$$d(x, (l, r)) = \begin{cases} (x - l)^2 & \text{if } x < l \\ (x - r)^2 & \text{if } x > r \\ 0 & \text{otherwise} \end{cases}$$

Here, α is a hyperparameter that controls the strength of the penalty, and in our experiments we set $\alpha = 1$. Typically, actions are either clipped to the allowed action space or constrained via activation functions such as hyperbolic tangent or sigmoid to ensure they remain within valid bounds. While these approaches are sufficient for algorithms like PPO and SHAC, in our case the penalty is crucial. This is because our policy is trained to optimize two neural networks that approximate the environment dynamics and reward. Since these networks are only trained on data within the valid action space, their outputs are unreliable for out-of-bound actions. As a result, the policy may exploit these approximations by generating actions far outside the allowed range. Action clipping may lead to zero gradients, while activation functions may results large pre-activation values.

To summarize, our algorithm, aims to minimize the 4 presented losses, $\mathcal{L'}^{\pi}_{\theta}$, $\mathcal{L}^{V}_{\theta}$, $\mathcal{L}^{R}_{\theta}$, and $\mathcal{L}^{F}_{\theta}$. A minimal pseudocode of our algorithm is presented in Algorithm 1. Fig. 1b presents an unrolled view of the SHAC++ algorithm.

However, consider that, to ensure fast convergence of the reward and transition networks, we employ a cache to store seen trajectories. To ensure that the cache is balanced, and not filled with 0-reward trajectories (which are common in case of sparse rewards), we divide the cache in K equally sized bins. Each bin is filled with trajectories that have a similar reward.

Additionally, to minimize the time spent between swapping among the different training procedures (policy, value, reward, transition), we apply a cool-down of $K^V/K^R/K^F$ episodes between running training procedure for the value/reward/transition networks.

4 Experiments

In this section, we evaluate SHAC++ against both PPO (for single agent settings), MAPPO (for multi-agent settings), and SHAC across various multi-agent scenarios and architectures. An ablation study replacing only the transition function (SHAC+, where the reward function remains differentiable) is provided in Section D in the appendix [1]. We select PPO as our baseline because, despite the emergence of more recent algorithms HAPPO [39], and SAC [36], it remains one of the few approaches that scales effectively to multi-agent settings without substantial modifications (e.g., MAPPO [21]). This makes PPO a representative benchmark for state-of-the-art performance in non-differentiable environments.

Scenarios. We evaluate our algorithms on several multi-agent scenarios from VMAS [11], which we selected over alternatives like PettingZoo [55], SMAC [47], MA-MuJoCo [35], and others due to its key advantages: differentiable physics (enabling gradient-based optimization), multi-agent design, vectorized implementation (for efficient parallel training), realistic physics-based simulation, and open-source accessibility. For more details about the choice of environments, see A in appendix [1]. We selected four distinct scenarios: Dispersion, Discovery, Transport, and Sampling, specifically for their cooperative nature and varying levels of required coordination (see A in appendix [1] for details). These scenarios can be categorized along two dimensions:

- **Reward differentiability:** Dispersion and Discovery feature non-differentiable reward functions (making SHAC inapplicable), while Transport and Sampling have differentiable rewards (allowing for SHAC implementation).
- **Observation completeness:** Transport and Dispersion provide agents with complete observation spaces relative to the environment state. In contrast, Discovery and Sampling have incomplete observation spaces, which presents an additional challenge for SHAC++ compared to SHAC, as SHAC++ lacks access to the complete environmental state.

This selection of scenarios enables a thorough evaluation of how each algorithm handles different types of multi-agent cooperation challenges.

Architectures. We test SHAC, PPO, and SHAC++ with both an MLP and a Transformer architecture. Specifically, we use a 1-layer MLP or a 1-layer single-head Transformer for policy, reward, and value network. The transition network used by SHAC++ is always a 3-layer single-head Transformer

While the MLP architecture is the simplest, the Transformer baseline benefits from positional invariance property of the agents observations. We apply the transformer architecture only if the number of agents is greater than 1, otherwise we use only the MLP architecture. For more details on the architectures, see B in appendix [1].

Hyperparameters. For all networks we use learning rate of $1e$-3 with Adam Optimizer [38]. For all scenarios, each training episode consists of 512 environments of 32 steps each. Each validation episode is composed of 512 environments of 512 steps each. We employ early stopping when the agents achieve 90% of the maximum reward in 90% of the episode's environments. If early stopping is not triggered, we stop training after $20,000$ episodes. The discount factor is set to 0.99 and lambda factor is set to 0.95. We report results for increasing number of agents $n \in \{1, 3, 5\}$. We repeat and report results for 3 runs with different seeds. See Table 2 and 3 in appendix [1] for a complete list of hyperparameters. Hyperparameter choices are the result of a small grid search on the transport scenario.

Overall, considering different training algorithms, architectures, scenarios, hyperparameters, we completed a total of 254 runs lasting between 1 to 8 hours each.

Hardware Setup. All experiments were conducted on a cluster with two nodes, each equipped with a V100 GPU (32GB of memory) and 200GB of RAM, and one node with four A100 GPUs (40GB of memory) and 512GB of RAM.

Figure 2: Snapshots of trained agents with SHAC++ in the Transport scenario. Colored circles represent agents pushing a red package (square) toward the green goal.

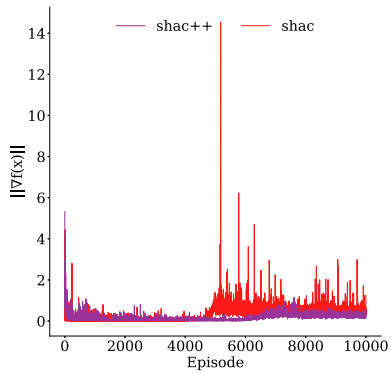


Figure 3: Policy gradient norms in the Transport environment over 5000 epochs. Spikes in gradient norm for SHAC (epoch 2700) and SHAC++ (epoch 3200) correspond to generalization events when agents begin frequent interactions with the package and each other.

4.1 Results

In this section, we compare the performance of SHAC++, PPO/MAPPO, and SHAC across different scenarios using the Transformer architecture for multi-agent settings and the MLP architecture for single-agent settings. The results show the mean and standard deviation of achieved rewards over 3 runs. An overview of the results is shown in Fig. 4 and Table 1. For results using the MLP architecture in all scenarios, refer to Figure 3 in appendix [1].

Dispersion. In Fig. 4, the first row displays results for the dispersion scenario (non-differentiable, thus SHAC is not applicable). In the first column (single agent, see also Table 1 and Figure 3 in appendix [1]), both PPO and SHAC++ perform well, though SHAC++ converges faster thanks to gradient approximation. In the second column (3 agents), PPO fails to converge, whereas SHAC++ converges in fewer than 10,000 episodes, suggesting emergent cooperative behavior. In the third column (5 agents), PPO fails to converge, while SHAC++ achieves high rewards (with no early stopping) but shows high variance due to increased environment complexity.

Discovery. The second row of Fig. 4 shows a scenario with partial observability, namely discovery. In the first column (single agent), SHAC++ outperforms PPO, though it does not reach the maximum reward, suggesting the world model struggles to capture

Env	N	PPO	SHAC	SHAC+ MLP	SHAC++	PPO	SHAC	SHAC+ Transformer	SHAC++
D	1	**1.00**	-	0.99	**1.00**	-	-	-	-
	3	0.22	-	0.64	0.99	0.15	-	1.00	**1.00**
	5	0.16	-	0.85	**1.00**	0.14	-	0.93	0.93
T	1	0.01	**1.00**	0.89	**1.00**	-	-	-	-
	3	0.01	0.84	0.95	0.41	0.00	0.95	**1.00**	0.95
	5	0.01	0.14	0.95	0.45	0.00	**1.00**	0.99	0.99
DI	1	0.33	-	**1.00**	0.35	-	-	-	-
	3	0.03	-	0.08	0.08	0.15	-	**1.00**	0.95
	5	0.15	-	0.14	0.14	0.39	-	0.61	**1.00**
S	1	0.07	0.58	0.59	**1.00**	-	-	-	-
	3	0.09	0.86	0.93	0.92	0.16	0.94	1.00	**1.00**
	5	0.12	0.83	0.92	0.86	0.43	**1.00**	0.89	0.92

Table 1: Normalized rewards (relative to the best performing model) for the different scenarios. Best results are in bold. D stands for Dispersion, T for Transport, DI for Discovery, and S for Sampling.

dynamics. In the second and third columns (3 and 5 agents), PPO again fails to converge, whereas SHAC++ maintains superior results, indicating cooperative behavior.

Transport. The third row of Fig. 4 presents a differentiable scenario, transport. In the first column (single agent), PPO fails to converge, while SHAC and SHAC++ achieve comparable performance. SHAC++ converges faster, but SHAC is more consistent. In the second and third columns (3 and 5 agents), PPO fails as in the single-agent case, while SHAC and SHAC++ both complete the task around 10,000 episodes. With more agents, convergence is faster because the task becomes easier (as more agents can push the package). The emergent of cooperation (see Fig. 2) can be also display but looking to the gradient norm in Fig. 3 we can see that the norm of the gradients for SHAC and SHAC++ increases when the agents start to collide with each other and the package. This is a sign of generalization, as the agents learn to avoid collisions and push the package together.

Sampling. The fourth row of Fig. 4 illustrates the sampling scenario, which is differentiable and thus applicable to SHAC. In the first column (single agent), PPO fails to converge, while SHAC++ outperforms SHAC. In the second and third columns (3 and 5 agents), PPO again fails to converge, while SHAC and SHAC++ both complete training in fewer than 5,000 episodes. Notably, SHAC does not exhibit the same stability advantage it displayed in other scenarios, possibly because gradients derived directly from the environment are noisier compared to those learned by the transition network.

5 Discussion

In the following, we discuss the results obtained in Section 4 and provide some insights into the performance of SHAC++ compared to SHAC and PPO, addressing the research questions posed in Section 1. Note that SHAC++ does not aim to outperform SHAC, but rather to perform comparably while being applicable even in non-differentiable environments.

5.1 Gradient Approximation with Neural Networks

RQ: *Can we train a neural network to approximate the gradients of a differentiable simulator?*

The proposed framework, SHAC++, appears to be successful in emulating the behavior of SHAC in differentiable environments such as transport and sampling. Therefore, we can conclude that it is definitely possible to approximate the gradients of a differentiable environment with a neural network. However, we note that this approach, while obtaining comparable results in most setups, tends to display higher variance. Both neural networks representing the reward and transition functions require sufficient training before becoming effective, which may entail multiple episodes. Interestingly, SHAC++ outperforms SHAC in the Sampling scenario. A possible explanation is that the complex dynamics of the environment may lead to unstable gradients.

5.2 Comparison to PPO and SHAC

RQ: *How does our algorithm compare to PPO and SHAC in both single-agent and multi-agent settings?*

The proposed algorithm appears capable of outperforming PPO across all scenarios, also showing better sample efficiency.

Single-agent:. In single-agent cases, while PPO manages to find near-good policies (e.g., in the Dispersion scenario), it often fails to complete scenarios within the same number of steps. Indeed, SHAC++ achieves better results in all environments (see Table 1). Compared to SHAC (where applicable), SHAC++ performs comparably.

Multi-agent:. This pattern becomes even more evident in multi-agent scenarios. In these cases, PPO often struggles to find even marginally good policies. SHAC++, on the other hand, eventually succeeds in understanding the environment and subsequently improves the policy to enable cooperation—see both Table 1 and Fig. 4. Its performance remains comparable to SHAC where SHAC can be applied. Overall, despite having to train multiple networks, SHAC++ converges much faster to optimal policies than PPO, as gradient guidance enables quicker and more accurate convergence to the desired outcome.

5.3 Scaling with Search Space

RQ: *How does the performance of these algorithms change as the search space increases?*

To analyze the impact of increasing search spaces, we primarily focused on scenarios with varying numbers of agents (up to 5 in the main experiments). The results demonstrate that performance patterns differ across scenarios. In the Dispersion scenario, SHAC++ shows increasing difficulty in finding optimal policies as the number of agents grows. Conversely, in Transport and Sampling scenarios, the system appears to converge more quickly with more agents. This counterintuitive behavior likely stems from the fact that, despite larger search spaces, the fundamental tasks in Sampling and Transport become relatively easier to solve with multiple agents, making policy discovery more straightforward. Even with approximated world dynamics (transition and reward functions), the guidance provided is sufficient to lead policies toward optimal behavior.

This scaling advantage is not observed in the Dispersion scenario, where complexity increases quadratically with agent count. Nevertheless, we observe consistent emergence of cooperative behavior patterns across all experimental scenarios, suggesting potential scalability to larger agent populations (see D for quantitative analysis with up to 20 agents in appendix [1]). The primary computational constraint lies in the transformer architecture employed for the action world model, where sequence length scales linearly with the number of agents (specifically, sequence length = $n \times s$, where n is the number of agents and s is the number of training steps). With agent populations exceeding approximately 50 agents, the self-attention mechanism's $O(n^2)$ complexity leads to prohibitive memory requirements.

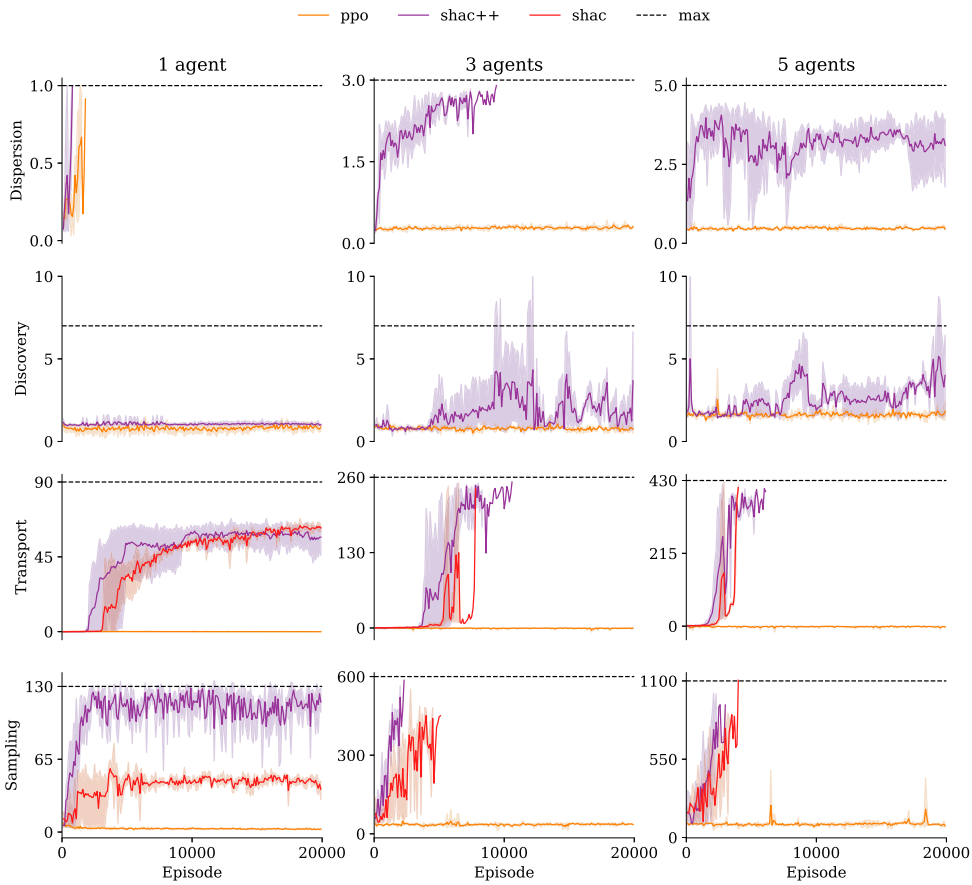


Figure 4: Performance comparison of SHAC++, PPO/MAPPO, and SHAC across different scenarios (Dispersion, Transport, Discovery, and Sampling) using the Transformer architecture for multi-agent settings and the MLP architecture for single-agent settings. The results show the mean and standard deviation of rewards over 3 runs. For results using the MLP architecture in all scenarios, refer to Figure 3 in appendix [1].

For such large-scale multi-agent systems, linear attention mechanisms (e.g., [9]) represent a viable architectural alternative.

6 Related Work

Differentiable Engines. Game engines [8, 7] and Physics engines [56, 18] have been widely used in the field of policy learning. Recently, there has been a trend towards differentiable engines, which allows for the use of gradient-based optimization methods, such as backpropagation. Notable examples are [31, 54, 5]. These frameworks may be built on top of auto differentiation libraries such [13, 4] (this is the case of VMAS) or directly with analytical gradients [14, 59]. Further, several works tackle the challenge of the non-smoothness of the contact dynamics [19, 46]

Deep Reinforcement Learning. Reinforcement learning has achieved significant advancements through the integration of deep neural networks. Foundational works such as DQN [44] and its variants [26] and AlphaZero [50] achieved human-level performance in complex environments. Subsequent algorithms improved efficiency, including A3C [29], DDPG and PPO for continuous control [27, 48], and SAC for off-policy learning [36]. Model-based approaches like Dreamer [23] and PlaNet [22] later enhanced sample efficiency.

Multi-Agent Reinforcement Learning. Multi-Agent Reinforcement Learning (MARL) is a subfield of reinforcement learning that focuses on learning policies for multiple agents that interact with each other [2]. This field has a wide range of applications, including robotics [17], traffic control [16], and game playing [6], and involves different types of agents such as cooperative, competitive, and mixed. In this paper, we primarily focus on cooperative multi-agent scenarios where agents collaborate to achieve common goals. Prominent works include extensions of single-agent algorithms like MADDPG [40] and MAPPO [21], as well as novel solutions such as COMA [24] for credit assignment, QMIX [28] for coordination, and Mean Field MARL [30] for large-scale scenarios.

Reinforcement Learning on Differentiable Engines. Precursors of SHAC with similar intuition are PODS [45] and CE-APG [33]. In [41], the authors introduce rendering to enhance model-based RL. In [61] authors employ a differentiable symbolic expression search engine. Trajectory optimization approach is taken in [58]. In [32] authors tackle unstable gradient due to stiff dynamics by adaptively truncating trajectories when these occur. With respect to this approach, we take a substantially different method, by emulating the differentiable environment with a trained neural network.

7 Limitations

While the proposed method shows promise, it is not a one-size-fits-all solution. Environments vary significantly in complexity and dynamics, and several limitations must be acknowledged.

World Model Learning and Cold-Start Problem. A central challenge lies in learning the world model from agent-environment interactions. Sparse or delayed rewards create a cold-start problem:

until sufficiently trained, agents struggle to explore effectively. This can be addressed by seeding with a cold-start dataset—manually designed or generated using algorithms like PPO or MPPI [3]—to bootstrap the world model and provide stable training foundation.

Baseline Performance and Implementation. Despite careful attention to implementation, including reference to best practices (e.g., the well-known "37 Implementation Details of Proximal Policy Optimization" blog [37]), we observed that PPO and MAPPO performed suboptimally in our experiments. To ensure the validity of these findings, we cross-verified results using a third-party implementation [12], which yielded consistent outcomes.

Policy Collapse and Robustness. Although early stopping is primarily used to conserve computational resources, it can give the false impression that the algorithm is inherently robust to policy collapse. In reality, we observed multiple instances of partial policy collapse—some of which recovered—and a few cases of complete collapse without recovery. One example of partial collapse appears in the five-agent variant of the dispersion scenario shown in Section C in appendix [1].

Partial Observability Challenges. The discovery scenario poses considerable difficulty for all algorithms due to inherent partial observability. Agents rely solely on simulated LiDAR for perception, providing only a narrow, local view. This makes it difficult for any method—including SHAC++—to construct accurate world models, often resulting in suboptimal performance and suggesting that partial observability remains challenging for the current approach.

Computational Overhead. SHAC++ addresses SHAC's limitations by adding transition and reward networks, increasing computational overhead. Training burden is especially pronounced with many agents, where the transition network must capture complex inter-agent dynamics. While often acceptable, this significantly slows training and demands substantial resources.

Interpretability. Regarding interpretability, SHAC++ does not offer significant advantages over its predecessors. As the policy is governed by a neural network, understanding or explaining its behavior remains a challenging task. This limitation is inherent to most deep reinforcement learning methods and highlights the ongoing need for techniques that enhance policy transparency and interpretability.

8 Conclusions & Future Work

In this paper, we introduced SHAC++, an extension of SHAC, designed to provide a reinforcement learning solution that is sample-efficient, applicable in both single-agent and multi-agent settings, and capable of handling non-differentiable environments. We demonstrated that SHAC++ preserves the core alignment with SHAC ($\mathbf{RQ_1}$), consistently outperforms the state-of-the-art PPO algorithm in both single-agent and multi-agent tasks ($\mathbf{RQ_2}$), and exhibits promising scalability in multi-agent environments, including the emergence of collaborative behaviors ($\mathbf{RQ_3}$).

Despite these encouraging results, several avenues remain for future exploration: scaling beyond tens of agents, handling extreme partial observability (e.g., discovery scenarios with limited sensory input), and expanding evaluation to broader environments—particularly from the cooperative AI community [25] and adversarial settings—to further validate SHAC++'s robustness and versatility.

Addressing the cold-start problem more effectively could also enhance performance, particularly in environments with sparse or delayed rewards. Several promising techniques for mitigating this issue include: *i*) leveraging datasets generated by alternative algorithms or human demonstrations to initialize the world model before policy training begins (world-model pre-training). *ii*) using Model Predictive Path Integral (MPPI) [3] to generate high-quality initial trajectories that can seed the world model and guide early exploration. The latter approach may also be integrated into SHAC++ during training to periodically introduce diverse trajectories, potentially improving robustness and convergence speed.

Finally, linear attention mechanisms like Performer [15] and Mamba [34] could improve SHAC++'s scalability in large multi-agent environments while avoiding the computational overhead of classical Transformers [57].

Acknowledgements

- Computational resources provided by INDACO Platform, which is a project of High Performance Computing at the University of MILAN http://www.unimi.it
- This work was partly supported by the MUR project "T-LADIES" (PRIN2020TL3X8X).
- This work was partly supported by the Italian PRIN project "CommonWears" (2020 HCWWLP).
- Computational cloud resources were provided by the Gemma 2 Google Research Award 2024.

References

[1] G. Aguzzi, F. Bertolotti, W. Cazzola, and M. Viroli. Shac++: A neural network to rule all differentiable simulators - appendix, 2025. URL https://zenodo.org/doi/10.5281/zenodo.16832958.

[2] S. V. Albrecht, F. Christianos, and L. Schäfer. *Multi-agent reinforcement learning: Foundations and modern approaches*. MIT Press, 2024.

[3] J. Alvarez-Padilla, J. Z. Zhang, S. Kwok, J. M. Dolan, and Z. Manchester. Real-time whole-body control of legged robots with model-predictive path integral control. *arXiv preprint arXiv:2409.10469*, 2024.

[4] J. Ansel and et al. PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24)*. ACM, Apr. 2024. doi: 10.1145/3620665.3640366.

[5] G. Authors. Genesis: A universal and generative physics engine for robotics and beyond, December 2024.

[6] B. Baker, I. Kanitscheider, T. M. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch. Emergent tool use from multi-agent autocurricula. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[7] M. Balla, G. E. Long, D. Jeurissen, J. Goodman, R. D. Gaina, and D. Perez-Liebana. Pytag: Challenges and opportunities for reinforcement learning in tabletop games. 2023.

[8] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[9] I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[10] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[11] M. Bettini, R. Kortvelesy, J. Blumenkamp, and A. Prorok. VMAS: A vectorized multi-agent simulator for collective robot learning. In *Distributed Autonomous Robotic Systems - 16th International Symposium, DARS 2022*, volume 28 of *Springer Proceedings in Advanced Robotics*, pages 42–56. Springer, 2022. doi: 10.1007/978-3-031-51497-5_4.

[12] M. Bettini, A. Prorok, and V. Moens. Benchmarl: Benchmarking multi-agent reinforcement learning. *Journal of Machine Learning Research*, 25(217):1–10, 2024. URL http://jmlr.org/papers/v25/23-1612.html.

[13] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[14] J. Carpentier and N. Mansard. Analytical derivatives of rigid body dynamics algorithms. In *Robotics: Science and systems (RSS 2018)*, 2018.

[15] K. Choromanski, V. Likhosherstov, D. Dohan, X. Song, A. Gane, T. Sarlos, P. Hawkins, J. Davis, A. Mohiuddin, L. Kaiser, et al. Rethinking attention with performers. *arXiv preprint arXiv:2009.14794*, 2020.

[16] T. Chu, J. Wang, L. Codecà, and Z. Li. Multi-agent deep reinforcement learning for large-scale traffic signal control. *IEEE Trans. Intell. Transp. Syst.*, 21(3):1086–1095, 2020. doi: 10.1109/TITS.2019.2901791.

[17] J. Chung, J. Fayyad, Y. A. Younes, and H. Najjaran. Learning team-based navigation: a review of deep reinforcement learning techniques for multi-agent pathfinding. *Artif. Intell. Rev.*, 57(2):41, 2024. doi: 10.1007/S10462-023-10670-6.

[18] E. Coumans and Y. Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning, 2016.

[19] J. Degrave, M. Hermans, J. Dambre, and F. Wyffels. A differentiable physics engine for deep learning in robotics. *Frontiers in neurorobotics*, 13:6, 2019.

[20] B. B. Elallid, N. Benamar, A. S. Hafid, T. Rachidi, and N. Mrani. A comprehensive survey on the application of deep and reinforcement learning approaches in autonomous driving. *Journal of King Saud University-Computer and Information Sciences*, 34(9):7366–7390, 2022.

[21] C. Y. et al. The surprising effectiveness of PPO in cooperative multi-agent games. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

[22] D. H. et al. Learning latent dynamics for planning from pixels. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, volume 97 of *Proceedings of Machine Learning Research*, pages 2555–2565. PMLR, 2019.

[23] D. H. et al. Dream to control: Learning behaviors by latent imagination. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[24] J. N. F. et al. Counterfactual multi-agent policy gradients. In *32nd AAAI Conference on Artificial Intelligence (AAAI)*, pages 2974–2982, 2018. doi: 10.1609/AAAI.V32I1.11794.

[25] J. P. A. et al. Melting pot 2.0. *CoRR*, abs/2211.13746, 2022. doi: 10.48550/ARXIV.2211.13746.

[26] M. H. et al. Rainbow: Combining improvements in deep reinforcement learning. In *Proc. 32nd AAAI Conf. Artificial Intelligence (AAAI)*, pages 3215–3222. AAAI Press, 2018. doi: 10.1609/AAAI.V32I1.11796.

[27] T. P. L. et al. Continuous control with deep reinforcement learning. In *4th International Conference on Learning Representations (ICLR)*, 2016.

[28] T. R. et al. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 4292–4301. PMLR, 2018.

[29] V. M. et al. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, volume 48, pages 1928–1937, 2016.

[30] Y. Y. et al. Mean field multi-agent reinforcement learning. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pages 5567–5576. PMLR, 2018.

[31] C. D. Freeman, E. Frey, A. Raichuk, S. Girgin, I. Mordatch, and O. Bachem. Brax - a differentiable physics engine for large scale rigid body simulation, 2021.

[32] I. Georgiev, K. Srinivasan, J. Xu, E. Heiden, and A. Garg. Adaptive horizon actor-critic for policy learning in contact-rich differentiable simulation. In *Forty-first International Conference on Machine Learning*, 2024.

[33] S. Gillen and K. Byl. Leveraging reward gradients for reinforcement learning in differentiable physics simulations. *arXiv preprint arXiv:2203.02857*, 2022.

[34] A. Gu and T. Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[35] A. Gupta, R. Kumar, M. Egorov, and S. Levine. Multi-agent reinforcement learning in mixed cooperative-competitive environments. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*, pages 11093–11100, 2021.

[36] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In J. G. Dy and A. Krause, editors, *International Conference on Machine Learning (ICML)*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR, 2018.

[37] S. Huang, R. F. J. Dossa, A. Raffin, A. Kanervisto, and W. Wang. The 37 implementation details of proximal policy optimization. In *ICLR Blog Track*, 2022. URL https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/.

[38] D. P. Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[39] J. Kuba, Z. Wang, K. Zhang, and T. Basar. Trust region policy optimization for multi-agent reinforcement learning. In *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021.

[40] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6379–6390, 2017.

[41] J. Lv, Y. Feng, C. Zhang, S. Zhao, L. Shao, and C. Lu. Sam-rl: Sensing-aware model-based reinforcement learning via differentiable physics-based simulation and rendering. *The International Journal of Robotics Research*, page 02783649241284653, 2023.

[42] M. Ma, T. Ni, C. Gehring, P. D'Oro, and P.-L. Bacon. Do transformer world models give better policy gradients? In *Forty-first International Conference on Machine Learning*, 2024.

[43] L. Metz, C. D. Freeman, S. S. Schoenholz, and T. Kachman. Gradients are not all you need. *arXiv preprint arXiv:2111.05803*, 2021.

[44] V. Mnih. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[45] M. A. Z. Mora, M. Peychev, S. Ha, M. Vechev, and S. Coros. Pods: Policy optimization via differentiable simulation. In *International Conference on Machine Learning*, pages 7805–7817. PMLR, 2021.

[46] G. Moritz, H. David, Z. Yonas, B. Moritz, T. Benrhnard, and C. Stelian. Analytically differentiable dynamics for multibody systems with friction contact. *ACM Transactions on Graphics*, 39(6), 2020.

[47] M. e. a. Samvelyan. The starcraft multi-agent challenge. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 2186–2188, 2019.

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

[49] L. S. Shapley. Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100, 1953.

[50] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[51] B. Singh, R. Kumar, and V. P. Singh. Reinforcement learning in robotic applications: a comprehensive survey. *Artificial Intelligence Review*, 55 (2):945–990, 2022.

[52] R. S. Sutton, A. G. Barto, et al. Introduction to reinforcement learning. vol. 135, 1998.

[53] J. Tang, G. Liu, and Q. Pan. A review on representative swarm intelligence algorithms for solving optimization problems: Applications and trends. *IEEE/CAA Journal of Automatica Sinica*, 8(10):1627–1643, 2021.

[54] A. H. Taylor, S. Le Cleac'h, Z. Kolter, M. Schwager, and Z. Manchester. Dojo: A differentiable simulator for robotics. *arXiv preprint arXiv:2203.00806*, 2022.

[55] J. K. e. a. Terry. Pettingzoo: Gym for multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 15032–15043, 2021.

[56] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[57] A. Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[58] W. e. a. Wan. Difftop: Differentiable trajectory optimization for deep reinforcement and imitation learning. *arXiv preprint arXiv:2402.05421*, 2024.

[59] K. Werling, D. Omens, J. Lee, I. Exarchos, and C. K. Liu. Fast and feature-complete differentiable physics engine for articulated rigid bodies with contact constraints. In *Robotics: Science and Systems*, 2021.

[60] J. Xu, M. Macklin, V. Makoviychuk, Y. Narang, A. Garg, F. Ramos, and W. Matusik. Accelerated policy learning with parallel differentiable simulation. In *International Conference on Learning Representations*, 2022.

[61] W. Zheng, S. Sharan, Z. Fan, K. Wang, Y. Xi, and Z. Wang. Symbolic visual reinforcement learning: A scalable framework with object-level abstraction and differentiable expression search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.