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

Paper #123

Abstract. Reinforcement learning algorithms have demonstrated success in domains such as robotics and multi-agent systems, yet they often exhibit limited sample efficiency in complex environments, and a unified approach that effectively addresses both single- and multi-agent settings remains elusive. To challenge this gap, we introduce a novel framework, SHAC++, inspired by SHAC, which is a reinforcement learning algorithm designed for differentiable simulators. Unlike SHAC, SHAC++ eliminates the requirement for differentiable transition and reward functions by utilizing plain neural networks to estimate the gradients that would otherwise be derived directly from simulators. These neural networks are trained alongside the policy and value networks of SHAC. We evaluate SHAC++ on a set of challenging multi-agent scenarios from VMAS and compare its performance against SHAC and PPO—a current state-of-the-art algorithm for non-differentiable environments. Our results demonstrate that SHAC++ significantly outperforms PPO in both single- and multi-agent scenarios and achieves comparable performance to SHAC, despite not having access to simulator gradients.

1 Introduction

Learning control policies has important applications in robotics [40], autonomous driving [17], swarm intelligence [42] and many other fields. These scenarios present a variety of different challenges, such as sparse rewards, complex dynamics, and multi-agent interactions. Current approaches often focus on either single- or multi-agent scenarios, lacking a unified solution that works effectively in both settings. Moreover, reinforcement learning algorithms tend to be *sample inefficient*, namely requiring a large number of interactions with the environment to learn a good policy. A recently introduced method for the latter problem is SHAC [48] which exploits differentiable simulators to provide precise gradient information, thereby expediting policy optimization and improving learning efficiency.

Despite its effectiveness, SHAC has two main limitations. First, it requires both a differentiable simulator and a differentiable reward function to operate correctly. Obtaining these can be impractical, particularly in scenarios with inherently sparse rewards. Second, even when a differentiable simulator is available, SHAC has been found to be sensitive to complex dynamics, such as object collisions [20] or interactions in multi-agent systems. Backpropagation through these dynamics has been found, in general, to lead to unstable gradients [9, 31].

In this work, we aim to address both limitations by approximating the gradients of the differentiable simulator inspired by model-based reinforcement learning algorithms [23, 24]. Specifically, we propose replacing both the simulator and the reward function with two neural networks trained alongside the policy and value networks. This approach is designed to create an algorithm that is more robust to complex dynamics and sparse rewards, enabling it to function effectively across a broader range of environments, including both single-agent and multi-agent scenarios. However, this method also introduces additional complexity and computational overhead due to the need for training these additional networks.

We evaluate our framework, referred to as SHAC++, against two baseline algorithms: PPO [37], a state-of-the-art approach for non-differentiable environments (also widely used in multi-agent scenarios through the MAPPO variant [50]), and SHAC [48], which is tailored for differentiable simulators. Our goal is to outperform PPO and achieve performance comparable to SHAC in differentiable environments. The evaluation is conducted using VMAS, a state-of-the-art simulator designed for complex, physics-based multi-agent scenarios [10]. We compare these algorithms across cooperative tasks involving varying numbers of agents, providing a systematic analysis of how their performance scales with the increasing size of the search space.

With this work, we aim to answer the following research questions:

- **RQ₁** Can we train a neural network to approximate the gradients of a differentiable simulator?
- **RQ₂** How does our algorithm compare to PPO and SHAC in both single-agent and multi-agent settings?
- **RQ₃** How does the performance of these algorithms change as the search space increases?

We summarize our contributions as follows:

- We propose a new reinforcement learning framework that lifts the requirement for a differentiable simulator in SHAC.
- We assess the SHAC algorithm in multi-agent scenarios, where it has not been previously evaluated.
- We evaluate our algorithm against PPO and SHAC on a set of environments with increasing numbers of agents.
- We demonstrate that our algorithm matches SHAC's performance in differentiable environments while significantly outperforming PPO in terms of sample efficiency, convergence speed, 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 (RL) 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) [38], $\mathcal{M}$: $(\mathcal{N}, \mathcal{S}, \mathcal{O}, \mathcal{A}, \pi, \mathcal{R}, F, H, \gamma, \mu)$. Where $\mathcal{N} = \{1, 2, \ldots, n\} = [n]$ 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 obser-

vation 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 $s_{i+1}^\tau = F(s_i^\tau, a_{i1}, \ldots, a_{in})$. 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.

While we use a unique set of parameters θ for all the networks. Each network can either use its own subset of θ or it can share parameters with other networks. In practice, the parameters will be shared only for the policies.

For simplicity, we formally consider only the case where policies have access to the full state, meaning that P_i is the identity function. This allows us to focus on the fundamental aspects of our approach without the added complexity of partial observability. Extending our framework to handle partial observability is straightforward; however, we choose not to address it here to maintain clarity and simplicity in our presentation. Nonetheless, our experiments do include scenarios where policies do not have access to the complete state information.

2.3 SHAC

In its trivial multi-agent extension, SHAC trains policies $\pi_{1\theta}, \ldots, \pi_{n\theta}$ by minimizing the loss function $\mathcal{L}_\theta^\pi : \mathcal{T}^N \to \mathbb{R}$ for a batch of N trajectories $B = \{\tau_1, \ldots, \tau_N\}$ sampled usin 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 [41] 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_k^t 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, **??** presents an unrolled view of the SHAC algorithm for a single-agent scenario.

3 Experiments

In this section, we test SHAC++ with PPO and SHAC for a set of multi-agent scenarios and architectures. An ablation study when replacing only the transition function (meaning that the reward function is assumed differentiable) is also conducted in Appendix D.

Scenarios. As mentioned previously, we choose a few multi-agent scenarios from the VMAS package to test SHAC++, SHAC and PPO. In particular, we choose the scenarios Dispersion, Discovery, Transport, and Sampling. For details regarding each scenario, see Appendix A. Dispersion and Discovery have non-differentiable reward functions, thus SHAC is not applicable. Transport and Sampling have differentiable reward functions, thus SHAC is applicable. For Transport and Dispersion, the observation space is complete wrt. the state. For Discovery and Sampling, the observation space is incomplete wrt. the state. In these scenario, SHAC++ is slightly disadvantaged as it does not have access to the full state of the environment compared to SHAC.

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 Appendix B.

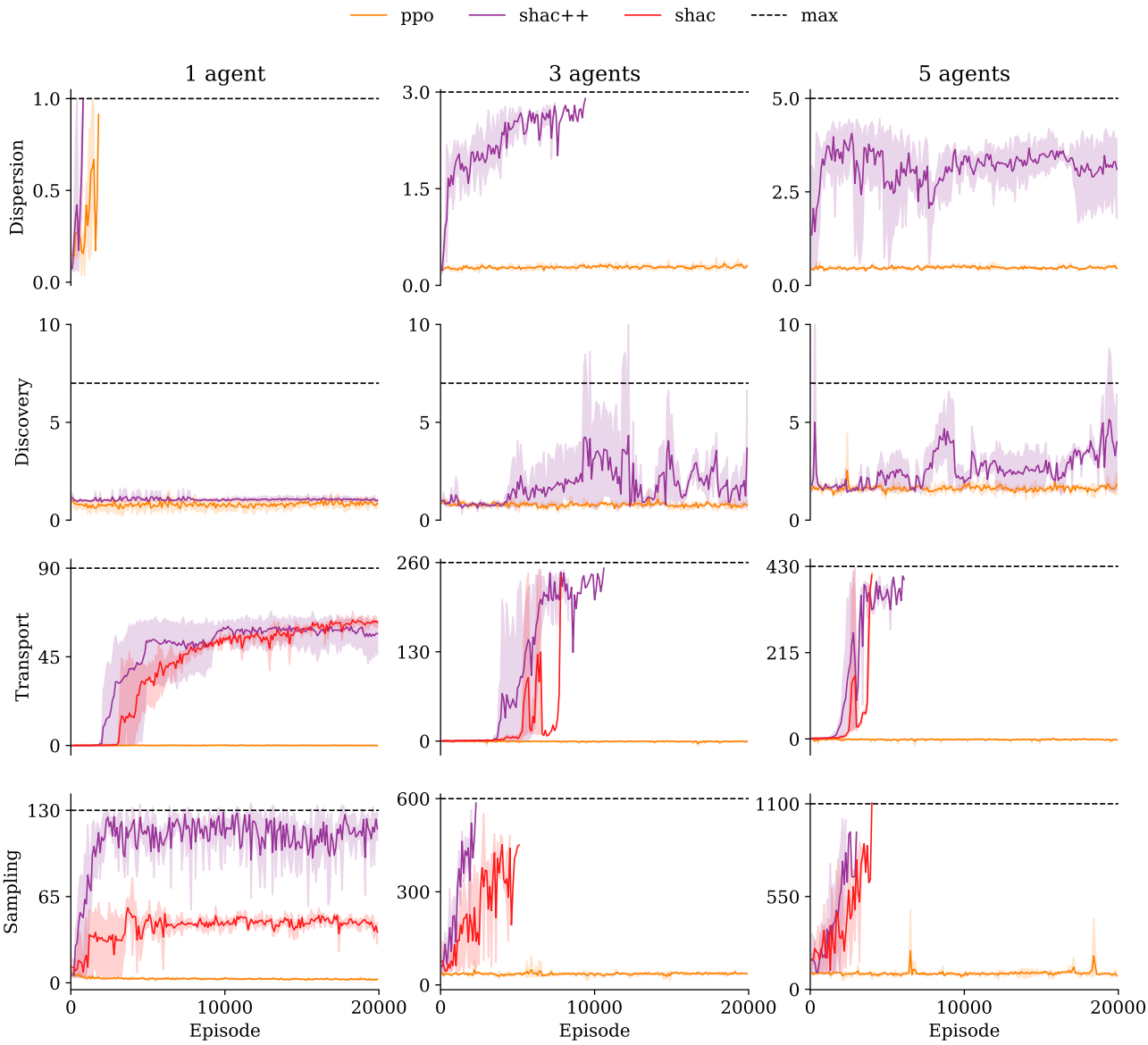


Figure 1: Performance comparison of SHAC++, PPO, and SHAC across different scenarios (Dispersion, Transport, Discovery, and Sampling) using the Transformer architecture for multi-agent settings and the MLP architecture for single-agent settings. The results show the mean and standard deviation of rewards over 3 runs. For results using the MLP architecture in all scenarios, refer to Fig. 6.

Environment	agents	PPO	SHAC	SHAC+ MLP	SHAC++	PPO	SHAC	SHAC+ Transformer	SHAC++
dispersion	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
transport	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
discovery	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**
sampling	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.

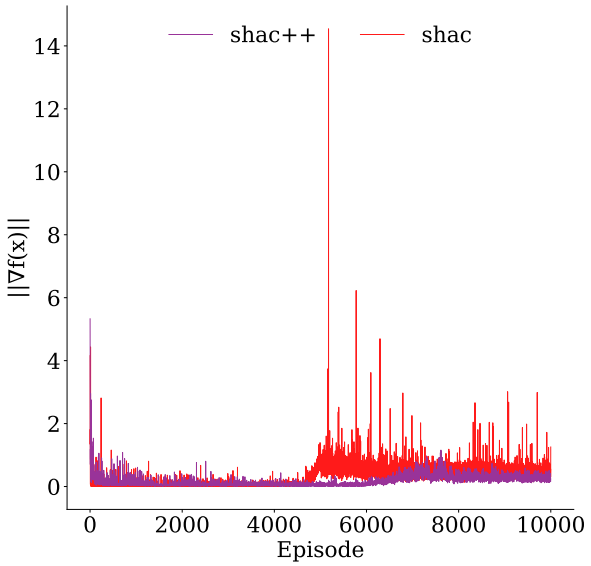


Figure 2: Norm of policy gradients on the Transport environment across 5000 epochs. The raise in norm for SHAC (epoch 2700) corresponds with occurrence of generalization, where collision between agents and the packages become frequent. Also SHAC++ experience a raise in norm, albeit more behaved, corresponds with the occurrence of generalization (epoch 3200).

Hyperparameters. For all networks we use learning rate of $1e$-3 with Adam Optimizer [27]. 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 3 and Table 2 for a complete list of hyperparameters.

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

Hardware Setup. Our experiments were conducted on a cluster of machines composed of 2 computational nodes, each with one NVIDIA Tesla V100 GPU (with 32GB of memory), 200GB of RAM, and two Intel Xeon Gold 6226R processors (32 cores each).

3.1 Results

In this section, we compare the performance of SHAC++, PPO, 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. 1 and Table 1. For results using the MLP architecture in all scenarios, refer to Fig. 6.

Dispersion. In Fig. 1, 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 Fig. 6), 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. 1 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 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. 1 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 can be also display but looking to the gradient norm in Fig. 2 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. 1 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.

4 Discussion

In the following, we discuss the results obtained in Section 3 and provide some insights into the performance of SHAC++ compared to SHAC and PPO trying to answer the research questions posed in Section 1. Note that SHAC++ does not aim to outperform SHAC, it merely aims to perform comparably with SHAC while being applicable even in non-differentiable environment.

RQ_1 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, it 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.

RQ_2 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 showing also a better sample efficiency. 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). Despite having to train multiple networks, SHAC++ converges much faster to optimal policies, as gradient guidance enables quicker and more accurate convergence to the desired outcome. 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. 1.

RQ_3 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. 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 counter intuitive 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 advantage does not hold in the Dispersion scenario, where complexity increases substantially with additional agents. Nevertheless, we observe the emergence of cooperation across all scenarios, suggesting potential scalability to larger agent populations. The main bottleneck lies in the transformer architecture for the action world model, where the sequence length scales with the number of agents (sequence length = num. agents times num. train steps). With significantly large populations ($\sim$50 agents), the attention mechanism becomes computationally expensive, leading to prohibitively large memory requirements. In these scenarios, a linear attention mechanism (such as [8]) may be employed.

The Discovery scenario, exhibiting high variance, shows no clear pattern compared to the other three scenarios. This might be attributed to partial observability, which significantly complicates convergence and inter-agent cooperation.

5 Related Work

Differentiable Engines. Game engines [7, 6] and Physics engines [44, 15] 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 [19, 43, 4]. These frameworks may be built on top of auto differentiation libraries such [11, 3] (this is the case of VMAS) or directly with analytical gradients [12, 47]. Further, several works tackle the challenge of the non-smoothness of the contact dynamics [16, 35]

Deep Reinforcement Learning. Deep reinforcement learning has revolutionized policy learning tasks by combining reinforcement learning with deep neural networks. Foundational works like DQN [32] (and its variants [25]) and AlphaZero [39] demonstrated human-level performance in complex environments such as Atari games and chess. Over the years, several algorithms have been proposed to create more stable and efficient learning methods. Notable examples include A3C for asynchronous learning [33], DDPG and PPO for continuous action spaces [28, 37] (also used in modern chatbot like ChatGPT), and SAC for continuous action spaces with off-policy learning [22]. Following the initial era of model-free algorithms, model-based algorithms have been introduced to enhance sample efficiency. Early works such as Dreamer [24] and PlaNet [23] have demonstrated promising results in terms of both sample efficiency and performance.

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 [14], traffic control [13], and game playing [5, 26], 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 need to collaborate to achieve a common goal. Prominent works in this area include extensions of single-agent algorithms to multi-agent settings, such as MADDPG [29] and MAPPO [50], as well as novel solutions to unique challenges in multi-agent settings. Examples of these solutions are COMA [18] for multi-agent credit assignment, QMIX [36] for coordination in cooperative multi-agent settings, and Mean Field MARL [49] for handling a large number of agents.

Reinforcement Learning on Differentiable Engines. Precursors of SHAC with similar intuition are PODS [34] and CE-APG [21]. In [30], the authors introduce rendering to enhance model-based RL. Zheng et al. [51] employ a differentiable symbolic expression search engine. Trajectory optimization approach is taken in [46]. Georgiev et al. [20] tackles 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.

6 Conclusions & Future Works

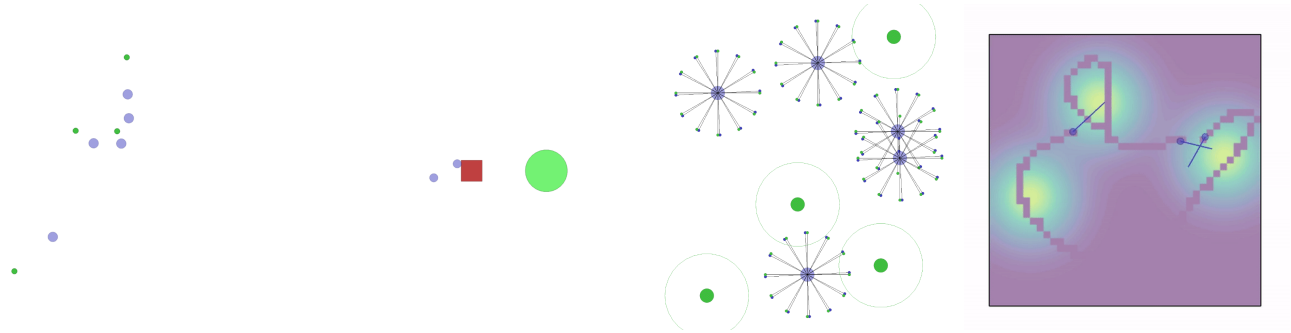
In this paper, we proposed SHAC++, an extension of SHAC, to provide a reinforcement learning solution that is both sample efficient, able to work in both single-agent and multi-agent scenarios, and capable of handling non-differentiable scenarios. We demonstrated that this solution maintains alignment with SHAC ($\mathbf{RQ_1}$), outperforms the current state-of-the-art PPO in both single-agent and multi-agent scenarios ($\mathbf{RQ_2}$), and exhibits promising scaling capabilities in multi-agent settings with emergent collaborative behaviors ($\mathbf{RQ_3}$).

Despite these encouraging results, several directions for future work remain. First, we aim to explore scalability beyond the current limitation of tens of agents. Additionally, as evidenced in the discovery scenario, the current approach struggles when agents have highly partial observations of the environment. Furthermore, we plan to evaluate our approach in other multi-agent environments, such as those presented from the cooperative AI community [1] and also in adversarial settings.

Acknowledgements

Redacted for anonimity

A Environments



(a) Dispersion (b) Transport (c) Discovery (d) Sampling

Figure 3: VMAS environments used in the experiments.

In this section, we provide a detailed discussion of the environment and scenarios used to test the SHAC++ algorithm. Specifically, we utilize four vectorized environments available as part of VMAS.

VMAS is a differentiable simulator designed for various multi-agent tasks. These tasks are built on top of a differentiable physics engine, resulting in differentiable transition functions for all tasks. However, the reward function is non-differentiable. To address this limitation, we created custom reward functions, where possible, to replicate the non-differentiable ones provided by VMAS. Consequently, some scenarios are compatible with all tested algorithms (PPO, SHAC, and SHAC++), while others are only applicable to PPO and SHAC++.

Furthermore, the observations produced by VMAS are often only partial representations of the full state. While in some cases these observations are sufficient to predict the next observations, in others they are not. In such cases, SHAC++ is expected to face challenges, as the world model is trained using only partial information. In contrast, SHAC computes gradients using access to the full world state. We chose to retain this limitation to better reflect the behavior of SHAC++ in more realistic environments.

A.1 Dispersion

In the Dispersion task, n agents must reach n randomly placed goals. In Fig. 3a, the agents are represented by blue dots, while the goals are depicted as green dots.

Each agent has a continuous 2-dimensional action space, bounded between -1 and 1, to represent acceleration along the x- and y-axes, respectively. Each agent observes its own position, velocity, and relative position to each goal. As a consequence, the world model has sufficient information to accurately predict the next states.

Each agent receives a reward of 1 upon reaching a goal, making the maximum possible reward n. While the transition function is differentiable, the reward function is not. Consequently, only PPO and SHAC++ are applicable to this scenario.

A.2 Transport

In the Transport problem, n agents work together to push a package to a randomly placed target location. In Fig. 3b, the agents are represented by blue dots, the package is shown as a red square, and the goal is depicted as a green circle. The more agents collaborate to push the package, the faster they can reach the goal and achieve the maximum reward.

Each agent has a continuous 2-dimensional action space, bounded between -1 and 1, to represent acceleration along the x- and y-axes, respectively. Each agent observes its own position, velocity, the relative position to the package, and the relative position between the package and the goal. As a consequence, the world model has sufficient information to accurately predict the next states.

Each agent receives a reward proportional to the distance between the initial position of the package and the goal. Consequently, the maximum reward corresponds to the distance between the package and the goal. Since the maximum reward varies across environments, we only plot a reference line representing good performance. Although this reward function is differentiable, the implementation provided by VMAS is not. Therefore, we created a custom reward function to replicate the original one used by all algorithms (SHAC++, SHAC, and PPO).

A.3 Discovery

In the Discovery task, n agents aim to collect as many randomly placed goals as possible. When they successfully collect k goals, another k goals are randomly placed. To collect a goal, at least s agents must be in proximity to it. In Fig. 3c, the agents are represented by blue dots, while the goals are depicted as green dots. The proximity area is indicated by a dashed circle. In our experiments, we set $s = 2$ and $k = 7$ when $n > 1$. When $n = 1$, we set $s = 1$ and $k = 7$.

Each agent has a continuous 2-dimensional action space, bounded between -1 and 1, to represent acceleration along the x- and y-axes, respectively. Each agent observes its own position, velocity, and lidar measurements of other agents and goals. As a consequence, the world model lacks sufficient information to accurately predict the next states. Specifically, the goal positions are not directly observed but are instead sensed through the lidar measurements. We believe this limitation significantly affects the performance of SHAC++.

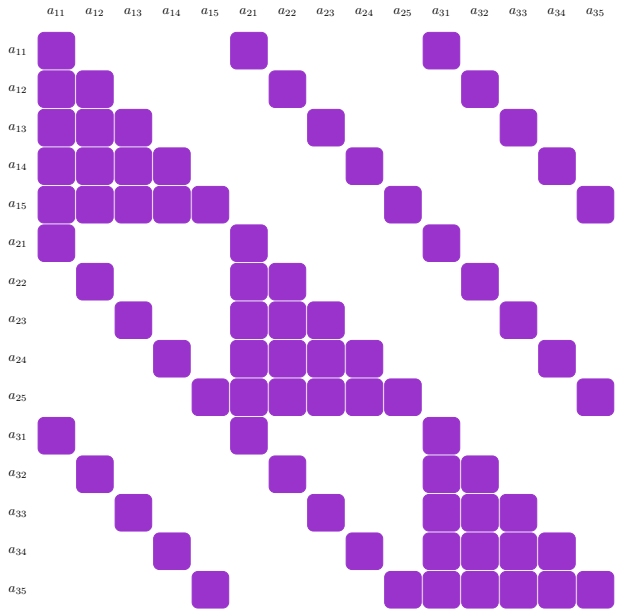


Figure 4: Attention matrix for the world model.

Each agent receives a reward of 1 when it collects a goal. Thus, the maximum reward depends on the amount of time the agents have to collect the goals. For early stopping purposes, we set the maximum reward to k. Additionally, the reference line is set to k. While the transition function is differentiable, the reward function is not. Therefore, only PPO and SHAC++ are applicable to this scenario.

A.4 Sampling

In the Sampling task, n agents must collect rewards from a grid. The rewards are sampled from k Gaussian distributions. In Fig. 3d, the agents are represented by blue dots, while the k reward-generating Gaussians are displayed as a scalar field, where the intensity of the color represents the respective reward value.

Each agent has a continuous 2-dimensional action space, bounded between -1 and 1, to represent acceleration along the x- and y-axes, respectively. Each agent observes its own position, velocity, lidar values from other agents, and rewards in a 3x3 grid around itself. As a consequence, the world model lacks sufficient information to accurately predict the next states. Specifically, it does not have access to the full reward distribution. We believe this limitation significantly affects the performance of SHAC++.

Each agent receives a reward equal to the value of the reward in the grid cell where it is located. Thus, the maximum reward depends on the placement of the Gaussians. Since the maximum reward varies across environments, we only plot a reference line representing good performance. While the reward function is differentiable, the one provided by VMAS is not. Therefore, we created a custom reward function to match the original one used by all algorithms (SHAC++, SHAC, and PPO).

B Architectures

As mentioned earlier, we test PPO, SHAC, SHAC++, and SHAC+ with two different architecture—Transformer and MLP. The former is a popular positional invariant architecture for sequence modeling [45]. While the latter is a simple feed-forward neural network.

The main advantage of the transformer architecture is its positional invariance property. In fact, the order in which the agent's observation are fed into the network should not affect the ultimate outcome, at least, for the tested environments. We believe that this leads to better sample efficiency that it is ultimately observed in policies achieving higher rewards (see Table 1).

In contrast, the MLP is fed with a concatenation of all agent observations. This results in a simple and lightweight implementation, leading to faster inference times. However, in our experiments, the MLP appears to underperform compared to the transformer architecture.

B.1 Policy/Value/Reward MLP Network

When implementing the policy, value, and reward functions with an MLP, we use a single hidden layer with sizes of 32, 96, and 160, depending on the number of agents. This choice was made to account for the increasing complexity that a higher number of agents should introduce. The activation function is always ReLU, and dropout is used only for the policy network. The learning rate is consistently set to 0.001. The cache for training these networks is set to $30,000$ samples. Gradient clipping is applied only to the policy network, with a threshold of 1. For a full list of hyperparameters, see Table 2 and Table 3.

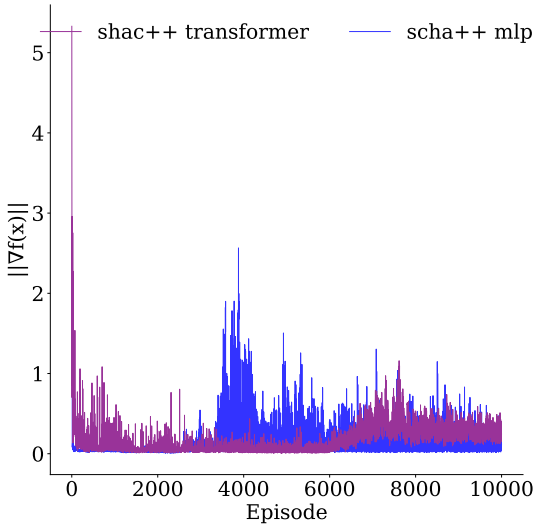


Figure 5: Gradient norm for a transformer architecture wrt. a MLP architecture.

B.2 Policy/Value/Reward Transformer Network

When implementing the policy, value, and reward functions with a transformer, we use a single hidden layer with a size of 64, regardless of the number of agents, and a feed-forward size of 128. We do not scale these sizes with the number of agents, as the transformer is expected to natively scale with the sequence length. The remaining hyperparameters are set consistently with those used in the MLP architecture. For a full list of hyperparameters, see Table 2 and Table 3.

B.3 World Model

When implementing the action world model, we consistently use a transformer architecture. Each action is represented as an embedding fed into the network. To maintain positional invariance between the agents, we employ a custom attention matrix that allows the action for step i of agent j, a_{ij}, to attend only to actions from the same step or previous steps of the same agent. Formally, a_{ij} can attend to all a_{kj} for $k < i$ and to all a_{ik} for $k \in \mathcal{N}$. This attention pattern is shown in Fig. 4. As a result, the world models process sequences of length $32 \cdot n$. We use a slightly deeper architecture to account for the increased complexity of the task, with 3 layers. The hidden size is set to 64 and the feed-forward size to 128. The remaining hyperparameters are set consistently with those used in the MLP architecture. For a full list of hyperparameters, see Table 2.

C Additional Experiments

Finally, we present the results of the experiments with the MLP architecture in Fig. 6. We observe that the MLP architecture underperforms compared to the Transformer architecture. Nonetheless, SHAC++ often outperforms PPO and generally performs on par with SHAC.

Interestingly, the Discovery scenario appears to be the most challenging, as none of the algorithms (PPO and SHAC++) are able to make any meaningful progress. This may be caused by the sample inefficiency of the MLP architecture, combined with the fact that the world model cannot access the full state.

The only scenario where SHAC outperforms SHAC++ is the Transport scenario. This may be due to the MLP's inability to overcome the gradient noise from the world model.

D Ablation Study

In the main paper, we focused on simultaneously replacing both the transition and reward functions with neural networks. This approach allows for greater generality and can be applied across a multitude of environments. However, when only the transition or reward function is differentiable and available, it is feasible to replace only one of them. This simplifies the algorithm, requiring the training of only a single additional network alongside the policy and value functions.

Specifically, we focus on replacing only the reward function. In fact, transition dynamics are often complex and challenging to model with a neural network, especially in multi-agent environments where observations depend not only on the actions of individual agents but also on those of other agents. This, in turn, may necessitate the use of larger architectures with greater computational demands, ultimately limiting the algorithm's practicality.

In contrast, the reward function is often simpler and, therefore, easier to model using lightweight neural networks. Moreover, a high degree of precision is typically not required in most scenarios. As a result, the reward function is an ideal candidate for replacement with a neural network.

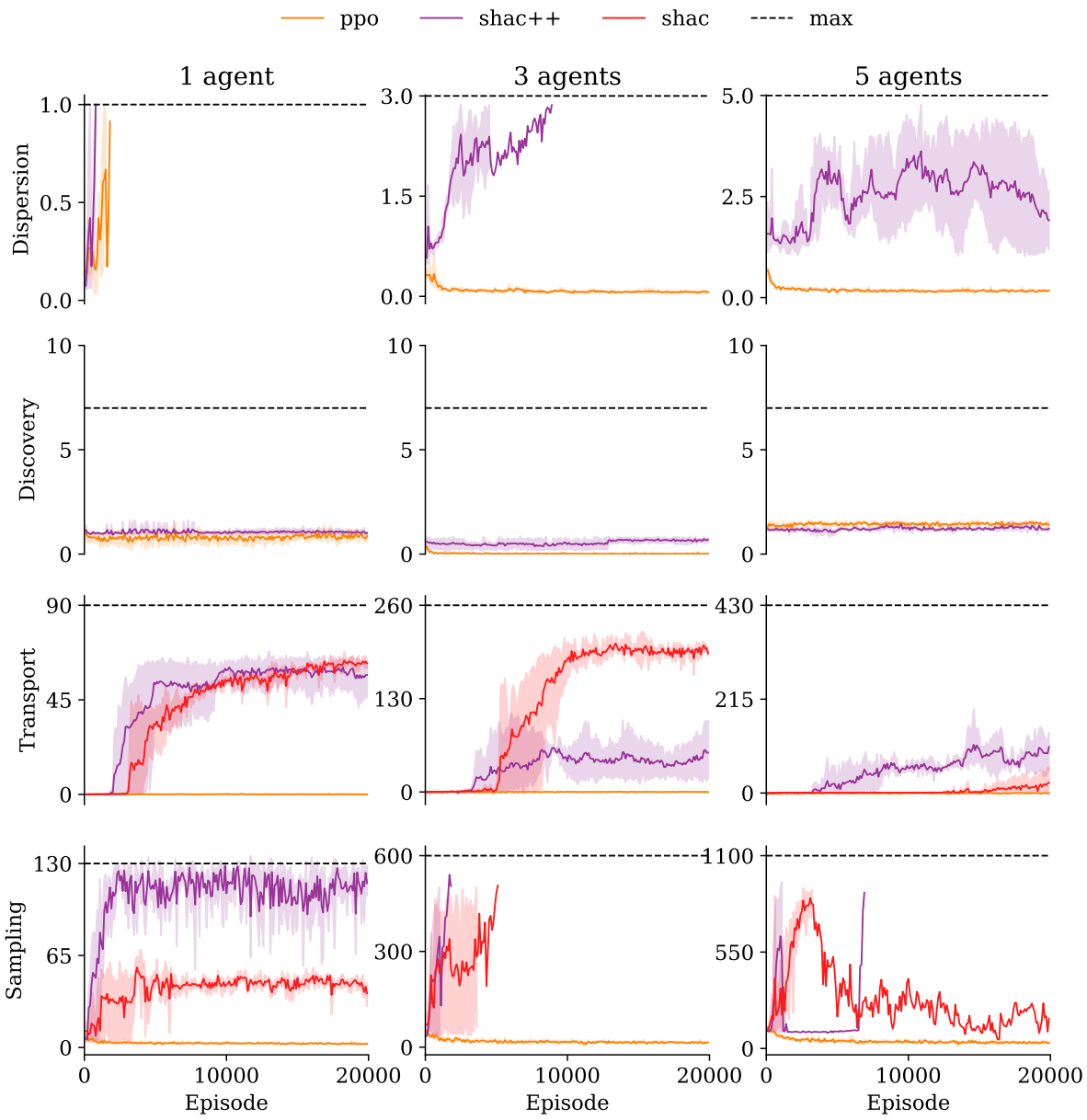


Figure 6: Comparison between SHAC++, PPO, and SHAC for increasing number of agents for Dispersion, Transport, Discovery, and Sampling scenarios with the MLP architecture. We show the mean and standard deviation of the normalized rewards across 3 runs. For the MLP architecture see Fig. 1.

This experiment also enables us to compare the final impact of gradients generated by a complex physics engine with those generated by a neural network. While the physics engine may produce more unstable gradients, the neural network may generate imprecise gradients, particularly during the early stages of training.

D.1 Results

We refer to the SHAC++ algorithm as SHAC+ when the transition function is differentiable and directly available. The results for the transformer architecture are shown in Fig. 7, while the results for the MLP architecture are presented in Fig. 8.

For the transformer architecture, SHAC++ generally appears to generate better policies, particularly in the Sampling and Transport environments. In the Dispersion environment, both algorithms perform comparably. Slightly better results are achieved with the SHAC+

Parameter	Transformer	MLP
number of training environments	512	512
training horizon	32	32
number of evaluation environments	512	512
evaluation horizon	512	512
policy layers	1	1
policy hidden size	64	160, 32, 96
policy feedforward size	128	-
policy heads	1	-
policy dropout	0.1	0.1
policy activation	ReLU	ReLU
policy variance	1	1
value layers	1	1
value hidden size	64	160, 32, 96
value feedforward size	128	-
value dropout	0.0	0.0
value activation	ReLU	ReLU
reward layers	1	1
reward hidden size	64	160, 32, 96
reward feedforward size	128	-
reward dropout	0.0	0.0
reward activation	ReLU	ReLU
world layers	3	3
world hidden size	64	64
world feedforward size	128	128
world dropout	0.0	0.0
world activation	ReLU	ReLU
policy learning rate	0.001	0.001
reward learning rate	0.001	0.001
value learning rate	0.001	0.001
value clip coefficient	None	None
reward clip coefficient	None	None
policy clip coefficient	1	1
world cache size	30000	30000
reward cache size	30000	30000
value cache size	30000	30000
world/value/reward batch size	1000	1000
world/value/reward cooldown epochs	10	10
world cache bins	25	25
reward cache bins	9	9
value cache bins	9	9
α	1.0	1.0
early stopping - max reward fraction	0.9	0.9
early stopping - max envs fraction	0.9	0.9
seed	42, 43, 44	42, 43, 44
max episodes	20000	20000
epochs between environment resets	10	10
epochs between evaluations	100	100
γ	0.99	0.99
λ	0.95	0.95

Table 2: SHAC/SHAC++/SHAC+ Hyperparameters

algorithm in the Discovery environment.

For the MLP architecture, the results are more mixed. In the Dispersion environment, SHAC++ appears to achieve better policies. Conversely, in the Transport environment, SHAC+ leads to substantially better policies. In the Discovery environment, both policies struggle to produce meaningful results. Finally, in the Sampling environment, SHAC++ performs on par with SHAC+, except in the single-agent case, where SHAC+ struggles.

D.2 Discussion

We can easily observe that the transformer architecture leads, generally speaking, to better policies. This is also evident by inspecting Table 1. This may be due to the positional invariance property of the transformers leading to more sample efficient networks.

Furthermore, with the transformer architecture, we observe a tendency for SHAC++ to produce better results. This may be attributed to the generated gradients being more stable, as shown in Fig. 5.

In contrast, for the MLP architecture, the results are more mixed. This could be attributed to the MLP architecture being less sample-efficient, which ultimately leads to less accurate gradients.

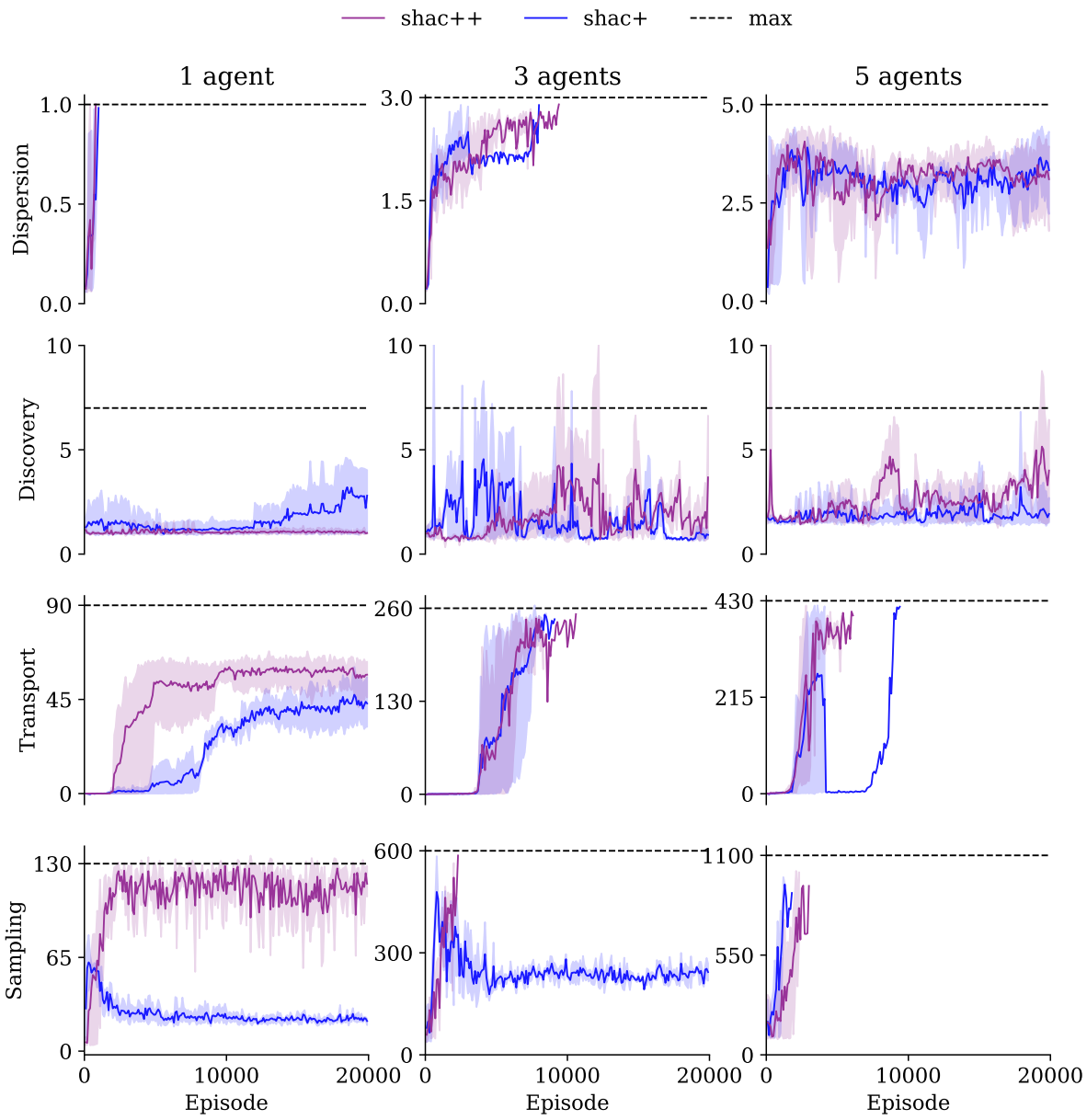


Figure 7: Comparison between SHAC++ with and without transition network, labelled SHAC+, for increasing number of agents for Dispersion, Transport, Discovery, and Sampling scenarios. Transformer architecture.

References

[1] J. P. Agapiou, A. S. Vezhnevets, E. A. Duéñez-Guzmán, J. Matyas, Y. Mao, P. Sunehag, R. Köster, U. Madhushani, K. Kopparapu, R. Comanescu, D. Strouse, M. B. Johanson, S. Singh, J. Haas, I. Mordatch, D. Mobbs, and J. Z. Leibo. Melting pot 2.0. *CoRR*, abs/2211.13746, 2022. doi: 10.48550/ARXIV.2211.13746. URL https://doi.org/10.48550/arXiv.2211.13746.

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

[3] J. Ansel, E. Yang, H. He, N. Gimelshein, A. Jain, M. Voznesensky, B. Bao, P. Bell, D. Berard, E. Burovski, G. Chauhan, A. Chourdia, W. Constable, A. Desmaison, Z. DeVito, E. Ellison, W. Feng, J. Gong, M. Gschwind, B. Hirsh, S. Huang, K. Kalambarkar, L. Kirsch, M. Lazos, M. Lezcano, Y. Liang, J. Liang, Y. Lu, C. Luk, B. Maher, Y. Pan, C. Puhrsch, M. Reso, M. Saroufim, M. Y. Siraichi, H. Suk, M. Suo, P. Tillet, E. Wang, X. Wang, W. Wen, S. Zhang, X. Zhao, K. Zhou, R. Zou, A. Mathews, G. Chanan, P. Wu, and S. Chintala. 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. URL https://pytorch.org/assets/pytorch2-2.pdf.

[4] G. Authors. Genesis: A universal and generative physics engine for robotics and beyond, December 2024. URL https://github.com/Genesis-Embodied-AI/Genesis.

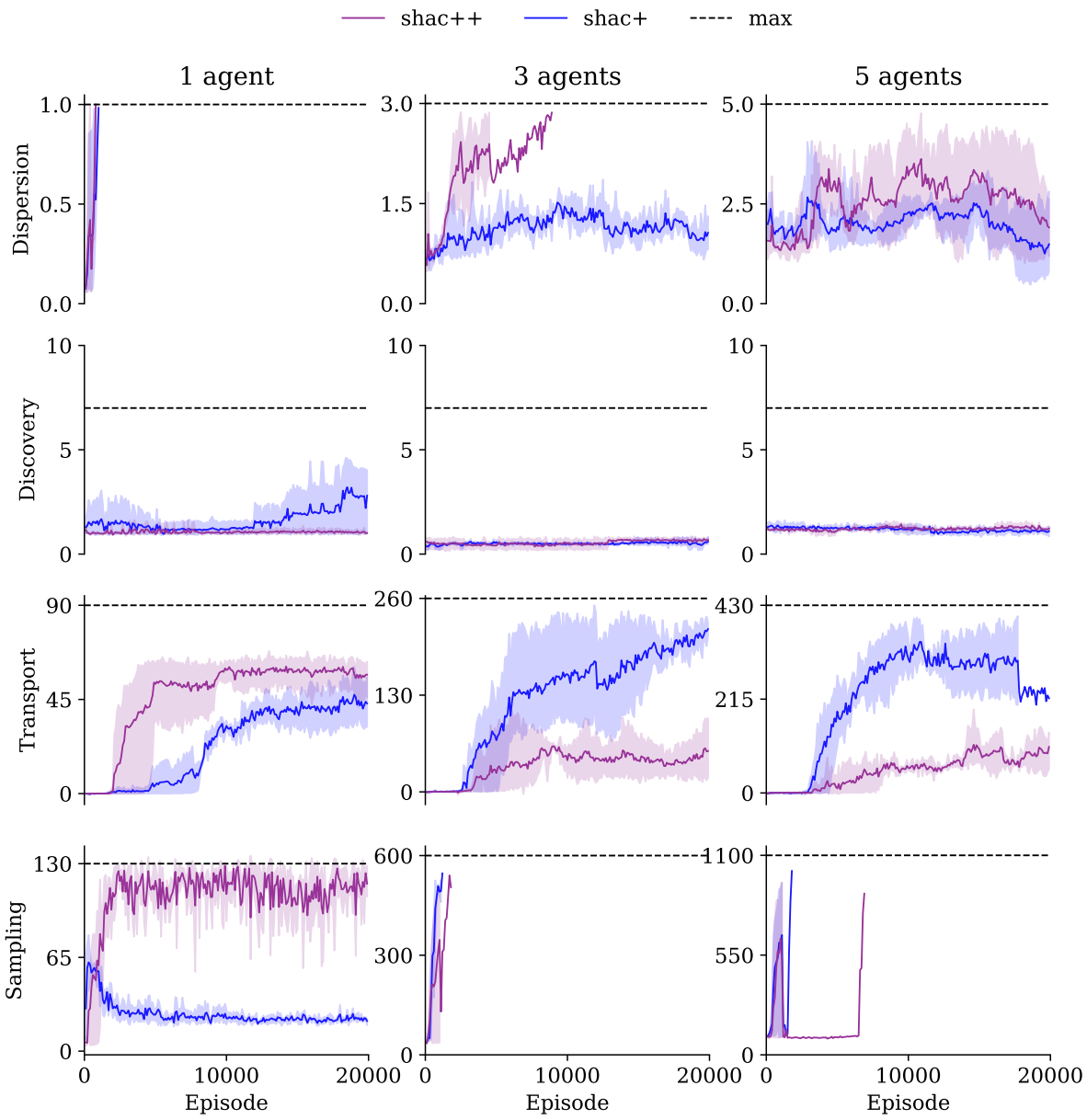


Figure 8: Comparison between SHAC++ with and without transition network, labelled SHAC+, for increasing number of agents for Dispersion, Transport, Discovery, and Sampling scenarios. MLP architecture.

[5] 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. URL https://openreview.net/forum?id=SkxpxJBKwS.

[6] 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.

[7] 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.

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

[9] 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.

[10] M. Bettini, R. Kortvelesy, J. Blumenkamp, and A. Prorok. VMAS: A vectorized multi-agent simulator for collective robot learning. In J. Bourgeois, J. Paik, B. Piranda, J. Werfel, S. Hauert, A. Pierson, H. Hamann, T. L. Lam, F. Matsuno, N. Mehr, and A. Makhoul, editors, *Distributed Autonomous Robotic Systems - 16th International Symposium, DARS 2022, Montbéliard, France, 28-30 November 2022*, volume 28 of *Springer Proceedings in Advanced Robotics*, pages 42–56. Springer, 2022. doi: 10.1007/978-3-031-51497-5_4. URL https://doi.org/10.1007/978-3-031-51497-5_4.

Parameter	Transformer	MLP
number of training environments	512	512
training horizon	32	32
number of evaluation environments	512	512
evaluation horizon	512	512
policy layers	1	1
policy hidden size	64	160, 32, 96
policy feedforward size	128	-
policy heads	1	-
policy dropout	0.1	0.1
policy activation	ReLU	ReLU
policy variance	1	1
value layers	1	1
value hidden size	64	160, 32, 96
value feedforward size	128	-
value dropout	0.0	0.0
value activation	ReLU	ReLU
policy learning rate	0.001	0.001
value learning rate	0.001	0.001
early stopping - max reward fraction	0.9	0.9
early stopping - max envs fraction	0.9	0.9
seed	42, 43, 44	42, 43, 44
max episodes	20000	20000
epochs between environment resets	10	10
epochs between evaluations	100	100
γ	0.99	0.99
λ	0.95	0.95
α	1.0	1.0

Table 3: PPO Hyperparmeters

[11] 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. URL http://github.com/jax-ml/jax.

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

[13] 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. URL https://doi.org/10.1109/TITS.2019.2901791.

[14] 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. URL https://doi.org/10.1007/s10462-023-10670-6.

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

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

[17] 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.

[18] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson. Counterfactual multi-agent policy gradients. In S. A. McIlraith and K. Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 2974–2982. AAAI Press, 2018. doi: 10.1609/AAAI.V32I1.11794. URL https://doi.org/10.1609/aaai.v32i1.11794.

[19] 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. URL http://github.com/google/brax.

[20] 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. URL https://openreview.net/forum?id=2FHWFG5ahw.

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

[22] 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, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR, 2018. URL http://proceedings.mlr.press/v80/haarnoja18b.html.

[23] D. Hafner, T. P. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2555–2565. PMLR, 2019. URL http://proceedings.mlr.press/v97/hafner19a.html.

[24] D. Hafner, T. P. Lillicrap, J. Ba, and M. Norouzi. 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. URL https://openreview.net/forum?id=S1lOTC4tDS.

[25] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. G. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In S. A. McIlraith and K. Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3215–3222. AAAI Press, 2018. doi: 10.1609/AAAI.V32I1.11796. URL https://doi.org/10.1609/aaai.v32i1.11796.

[26] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

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

[28] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. In Y. Bengio and Y. LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1509.02971.

[29] 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 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 6379–6390, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/68a9750337a418a86fe06c1991a1d64c-Abstract.html.

[30] 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.
[31] L. Metz, C. D. Freeman, S. S. Schoenholz, and T. Kachman. Gradients are not all you need. *arXiv preprint arXiv:2111.05803*, 2021.
[32] V. Mnih. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.
[33] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In M. Balcan and K. Q. Weinberger, editors, *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1928–1937. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/mniha16.html.
[34] 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.
[35] 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.
[36] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. N. Foerster, and S. Whiteson. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4292–4301. PMLR, 2018. URL http://proceedings.mlr.press/v80/rashid18a.html.
[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
[38] L. S. Shapley. Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100, 1953.
[39] 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.
[40] B. Singh, R. Kumar, and V. P. Singh. Reinforcement learning in robotic applications: a comprehensive survey. *Artificial Intelligence Review*, 55(2):945–990, 2022.
[41] R. S. Sutton, A. G. Barto, et al. Introduction to reinforcement learning. vol. 135, 1998.
[42] 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.
[43] 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.
[44] 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.
[45] A. Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.
[46] W. Wan, Y. Wang, Z. Erickson, and D. Held. Difftop: Differentiable trajectory optimization for deep reinforcement and imitation learning. *arXiv preprint arXiv:2402.05421*, 2024.
[47] 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.
[48] 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. URL https://openreview.net/forum?id=ZSKRQMvttc.
[49] Y. Yang, R. Luo, M. Li, M. Zhou, W. Zhang, and J. Wang. Mean field multi-agent reinforcement learning. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5567–5576. PMLR, 2018. URL http://proceedings.mlr.press/v80/yang18d.html.
[50] C. Yu, A. Velu, E. Vinitsky, J. Gao, Y. Wang, A. M. Bayen, and Y. Wu. The surprising effectiveness of PPO in cooperative multi-agent games. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.
[51] 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.