Advances in Science and Technology Research Journal

Advances in Science and Technology Research Journal, 2027, 21(1), 302–318
https://doi.org/10.12913/22998624/235821
ISSN 2299-8624, License CC-BY 4.0

Received: 2026.06.15
Accepted: 2026.11.15
Published: 2026.12.01

Training machine learning agent behaviors in Unity: Proximal policy optimization vs. soft actor-critic algorithms

Krystian Czajkowski[1], Maria Skublewska-Paszkowska[1*], Pawel Powroznik[1], Marcin Barszcz[1], Krzysztof Dziedzic[1], Sylwester Korga[1], Kinga Chwaleba[1]

[1] Department of Computer Science, Lublin University of Technology, Lublin, Poland
* Corresponding author's e-mail: maria.paszkowska@pollub.pl

ABSTRACT

The gaming evolution is strictly associated with machine learning development, such as reinforcement learning. Thus, there is an ongoing need to study and apply machine learning algorithms to games to achieve human-like agent behavior. Among various algorithms, Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) are among the state-of-the-art reinforcement learning methods. They have been widely adopted due to their optimal balance of performance, stability, and ease of implementation. Factors such as the test environment, reward function, neural network architecture, and computational resource availability significantly influence the relative performance of both algorithms. Thus, the main aim of this study was to analyze PPO and SAC reinforcement learning methods across three Unity-created environments, utilizing parallel learning with discrete and continuous actions. Using cumulative reward as the primary metric, SAC performed slightly better in simpler scenarios up to 1.1%, whereas PPO achieved better results in more complex environments up to 7.4%. The ablation study assesses hyperparameter sensitivity. Batch size had a stronger impact on SAC performance, whereas buffer size had a stronger impact on PPO.

Keywords: unity, ML-agents, proximal policy optimization, soft actor-critic.

INTRODUCTION

Machine learning-assisted design highlights the potential of learning-based methods to efficiently explore and optimize complex design spaces [1, 2]. The evolution of gaming is closely linked to the advances in machine learning, including Reinforcement Learning (RL), which is widely applied for controlling robotic systems, computer simulations, video games, and autonomous vehicle control with a high degree of precision [2, 3]. RL has recently demonstrated its ability to achieve, and in some cases surpass, the performance of the best human players in increasingly complex computer games [4, 5]. These advances have shown that RL agents can autonomously learn effective strategies through repeated interaction with the environment, without requiring explicit instructions for solving a task [6]. The success of RL in complex games highlights its potential for addressing challenging decision-making problems characterized by large state spaces, dynamic environments, and long-term dependencies [7, 8]. Consequently, computer games have become an important benchmark for evaluating the capabilities and limitations of modern RL algorithms [9]. Thus, there is an ongoing need to study and apply machine learning algorithms in games to achieve human-like agent behavior [10, 11] as well as to enhance user experience and enjoyment [12]. The Unity engine [11, 13] enables the creation of complex, realistic simulation environments that authentically reflect real-world elements [14]. It is useful for creating virtual objects representing real objects within an artificial environment. In this way, preliminary training and testing of agents takes place in a safe environment, before being transferred to the real world [15].

The agent training process consists of a repetitive iterative cycle of taking actions, observing the environment, awarding rewards, and updating behavior policies [16, 17]. Among various algorithms, Proximal Policy Optimization (PPO) [18, 19] and Soft Actor-Critic (SAC) [20, 21] are among the state-of-the-art reinforcement learning methods. They have been widely adopted due to their optimal balance of performance, stability, and ease of implementation. The PPO algorithm optimizes the agent strategy while ensuring the stability of the learning process [22]. It is applied both for discrete and continuous action spaces. The SAC algorithm provides efficient learning using a given sample while maintaining the benefits of entropy maximization and stability [23]. While some studies indicate an advantage of PPO in environments that enable parallel simulation [19, 24], other research demonstrates better performance of SAC in tasks requiring efficient exploration [23, 25] or precise manipulation [26, 27]. Taha et al. [24] also reported interesting results: in the training phase, SAC learned the fastest and achieved the highest reward function values, whereas in the testing phase on new, random targets, PPO performed better. That may suggest SAC was overfitting to specific training conditions.

Comparing the PPO and SAC policies is of great interest [23, 28], especially in environments with varying levels of difficulty, such as car tracks of different complexities. It is stated that, in these tasks, the PPO algorithm is clearly superior to SAC in terms of the time required to complete the route. Factors such as the test environment, reward function, neural network architecture, and computational resource availability significantly influence the relative performance of both algorithms.

Although numerous studies compare PPO and SAC, their conclusions are often contradictory due to differences in environments, reward functions, neural network architectures, and evaluation protocols. Consequently, it remains unclear how increasing task complexity affects the relative performance of these algorithms. To bridge this research gap, the main objective of this study was to systematically compare the reinforcement learning algorithms proximal policy optimization and soft actor-critic under controlled experimental conditions with gradually increasing complexity while maintaining consistent training settings across experiments. By maintaining consistent reward design, neural network architecture, training protocol, and evaluation settings across all experiments, the proposed framework enables an objective assessment of the influence of task complexity on learning dynamics, policy performance, and hyperparameter sensitivity. This study provides a systematic and reproducible comparison of PPO and SAC under identical experimental conditions and offers practical guidelines for selecting an appropriate reinforcement learning algorithm according to task complexity.

The authors' contributions are as follows:

- Introduction of a controlled benchmark consisting of three independent reinforcement learning environments with progressively increasing task complexity, enabling a systematic comparison of PPO and SAC under identical experimental conditions. Unlike previous studies, which typically evaluate algorithms with respect to single tasks or heterogeneous benchmark datasets, the proposed setup isolates the impact of task complexity on algorithm performance. The results obtained help explain the conflicting conclusions presented in previous studies [18, 28].
- All projects enabled systematic training and evaluation of learned policies, including performance curves and post-training tests. SAC performed better in simpler scenarios, whereas PPO achieved better results in more complex environments.
- Hyperparameter sensitivity was assessed in the ablation study. Batch size had a stronger impact on SAC performance, while buffer size more strongly influenced PPO in the most complex environment. Based on the experimental analysis, practical recommendations for selecting PPO and SAC depending on the complexity and characteristics of reinforcement learning tasks were formulated, providing guidance for researchers and practitioners developing new machine learning applications.

MATERIALS AND METHODS

ML-Agents environment

Test environments were created using the Unity engine 6000.0.40f1. For training the ML-Agents Toolkit 4.0.0 with PyTorch 2.8.0+cu129 framework was utilized.

Three research environments were created, gradually increasing in complexity, to test PPO and SAC algorithms. In this study, task

complexity is considered as a multidimensional property determined by the size and structure of the observation as well as action spaces, environmental dynamics, and the number of objectives that the agent must satisfy. The detailed description is gathered in Table 1.

In each project, the agents' goal is to complete the task as quickly as possible while adhering to the rules established for the environment. The agents collect information about their surroundings using an observation vector, presented in Table 2–3. Each environment assigns penalties and rewards to the agent's actions, guiding it towards the correct solution to a problem.

All projects utilize a parallel learning process, which means training multiple environments simultaneously, significantly reducing the time needed to train behavioral policies.

Ball

The agent's task in the first environment is to move a player character and push the red ball to its destination (the yellow platform) as quickly as possible, as shown in Figure 1. The agent's starting position, the ball, and the destination point change their location with each episode. The agent controls the character by moving forward or backward and rotating it to the right or left. The episode, lasting 2000 steps, ends when the ball collides with the platform or the episode reaches the last step.

Forest

The second project presents a more complex environment, as shown in Figure 2. The agent's task is to gather all logs into the woodpile area within a predefined time, while maintaining a sufficient level of satiety by collecting food strewn throughout the map. Ignoring hunger results in penalties and health loss, leading to failure. The agent controls a character capable of forward and backward movement, as well as rotation around its own axis. The agent moves on a platform bounded by walls.

Upon collision with a log, an agent lifts it. Only one log can be carried at a time. The project contains a fixed number of trees, which makes the task more difficult by introducing observation noise. In addition, a health and satiety system has been implemented. The agent can pick up food by colliding with it. The satiety level decreases with each step of the simulation: a drop of the satiety level to zero causes a rapid loss of health, while maintaining it above 50% restores health. The episode ends after 6,000 steps when all the logs are collected, when health drops to zero, or when the last step is reached.

The primary challenge for the agent is decision-making between food consumption and log collection. Food availability is limited, but it replenishes every 2000 steps, so collecting everything at once can lead to starvation, health loss, and failure to complete the task. The agent should learn to balance between gathering logs to earn a higher reward faster, while keeping satiety high.

Car

The third environment presents the vehicle driven by the agent, the route of which is limited by barriers illustrated in Figure 3. Each subsequent episode generates a new route presented in Figure 4 a)-c) consisting of road surface, barriers, and checkpoints. The episode ends after reaching the last step (3000) or the end of the route.

The agent controls a vehicle that uses ready-made physics solutions from the Unity engine. The observations collected by the agent include the environmental variables listed in Table 3 and two ray sensors: the first to observe the surroundings, and the second to collect observations at a greater distance ahead. Their operating characteristics determine how both ray sensors are used: upon reaching a checkpoint, a ray sensor cannot

Table 1. Multidimensional characterization of task complexity across the three Unity environments

Dimension	Ball	Forest	Car
Observation complexity	Low	Medium	High
Action complexity	Low	Low	High
Environmental dynamics	Low	Medium	High
Objective complexity	Single	Multiple	Sequential
Decision dependency	Low	Medium	High
Overall complexity	Low	Medium	High

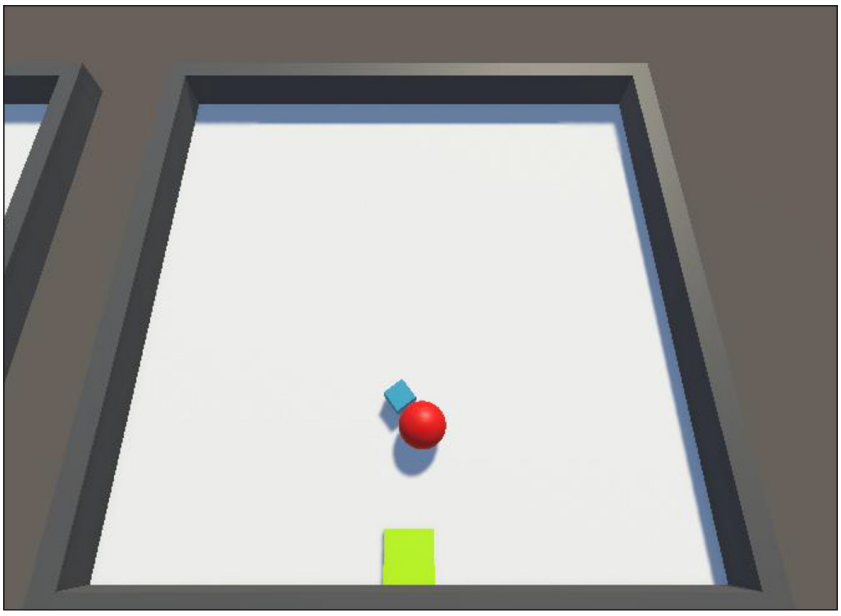


Figure 1. The Ball environment running – an agent pushing a ball toward a platform

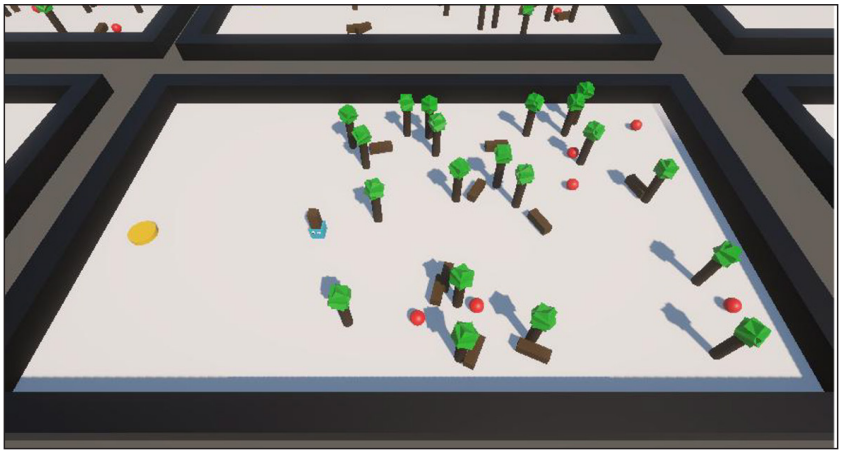


Figure 2. The Forest environment running – an agent transporting a log towards the woodpile

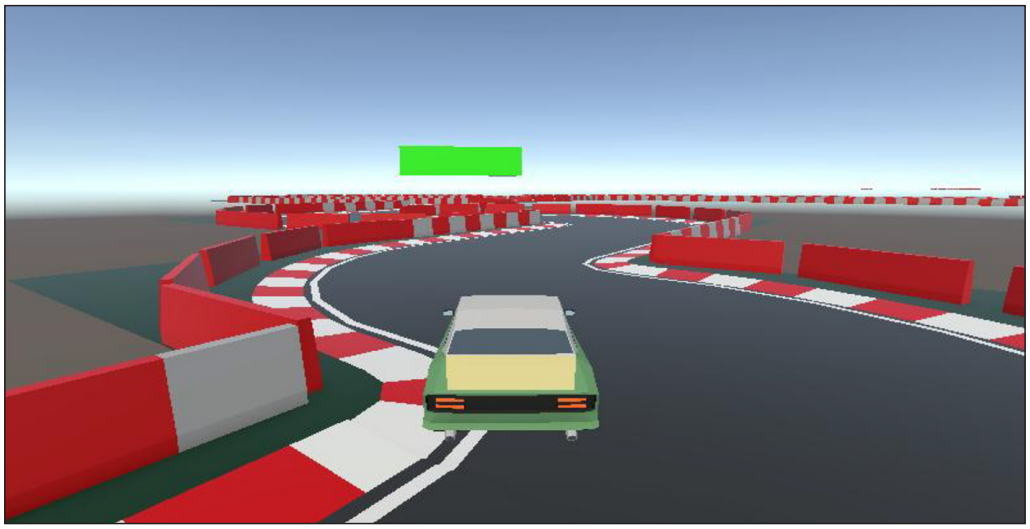


Figure 3. The Car environment running – an agent controlling vehicle

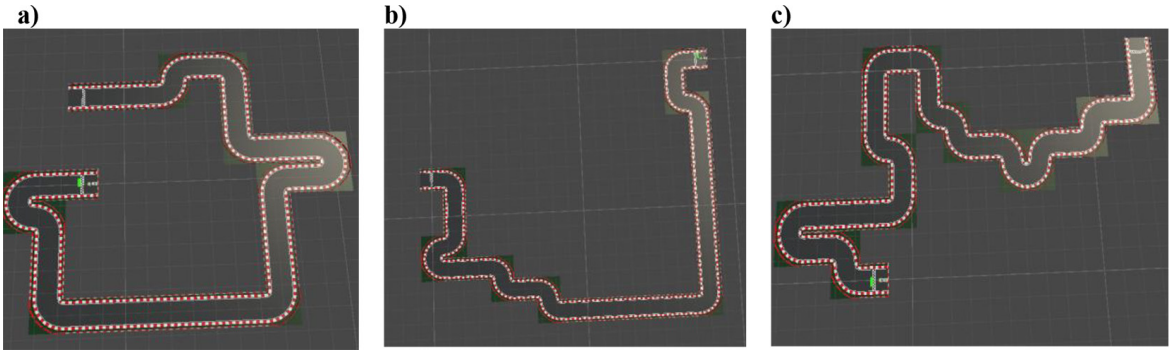


Figure 4. The Car environment – generated track examples

detect objects beyond it. Using another sensor at an angle allows the agent to observe the next checkpoint position, which, at higher speeds, helps it prepare to enter a turn.

The vehicle is controlled by continuous actions: throttle (-1,1), steering wheel angle (-1,1), and brake (True, False). The agent's task is to pass through all the checkpoints on the route as quickly as possible. Hitting barriers and low speed result in penalties, while high speed and a small number of steps between checkpoints result in a reward. The vehicle's characteristics make it risky to take corners at high speeds: the higher the speed, the more likely the car is to skid, further complicating the environment. The track consists of 3 types of elements: Start/Finish, straight, and turn. Each element has two anchors that determine the location of the next element, allowing it to be attached to the existing route.

The generator script fills a large 2-dimensional array with values corresponding to the route elements. The algorithm verifies whether the elements of the array in all possible directions to the last one can fit the entire length of the route using only straights, then randomly selects a route element with a 60% probability for a straight and 40% for a turn, excluding the direction in which the route may not fit. It guarantees the creation of a non-intersecting route with a fixed length of 34 elements.

Reinforcement learning algorithms

The selection of reinforcement learning algorithms considered in this study was determined by the capabilities of the framework. The ML-Agents framework provides built-in support for a limited number of reinforcement learning methods, which directly influenced the choice of PPO and SAC in this study.

Proximal policy optimization

Proximal policy optimization [19] is a policy-gradient-based reinforcement learning algorithm that aims to improve training stability by limiting policy updates. Instead of using unrestricted gradient ascent, PPO features a clipping mechanism. This mechanism limits how much the new policy can differ from the old one, reducing the risk of performance collapse and maintaining high sample efficiency. The PPO objective is defined as [19]:

$$L^{CLIP}(\theta) = E_t\left[min\left(r_t(\theta)\widehat{A_t}clip(r_t(\theta),1-\epsilon,1+\epsilon)\widehat{A_t}\right)\right] \quad (1)$$

where: θ stands for the parameters of the current policy, $E_t[\cdot]$ is the expected value over samples t collected during the agent's interaction with the environment, t denotes the time -step index, and $r_t(\cdot)$ is the ratio of the probability of taking an action under the new policy to the probability of taking the same action under the old policy.

The parameter ∈ specifies the maximum permissible range of strategy changes (0.2 in this study). The advantage function $\widehat{A_t}$ is estimated using generalized advantage estimation [19]:

$$\widehat{A_t} = \sum_{l=0}^{\infty}(\gamma\lambda)^l\left(r_{t+l} + \gamma V(s_{t+l+1}) - V(s_{t+l})\right) \quad (2)$$

where: γ is the discount factor, λ controls the bias-variance trade-off, l denotes the number of successive steps, r_{t+l} is the reward received after performing an action at time $t+1$, $V(s_{t+l})$ estimates the value of state s at time $t+l$, and s_{t+l} is the state of the environment at time $t+l$.

Soft actor-critic

The soft actor-critic [20] is an off-policy RL algorithm. It builds on the standard actor-critic framework by adding entropy maximization. The goal of SAC is to maximize both the expected return and the entropy of the policy. The maximum entropy objective is defined as [20]:

$$J(\pi) = E_{r\sim\pi}\left[\sum_{t=0}^{T} \gamma^t (r(s_t, a_t) + \alpha H(\pi(\cdot\,|s_t))\right] \quad (3)$$

where: π stands for the policy of the agent, defining the probability of selecting an action in a given state, $E_{r\sim\pi}[\cdot]$ is the expectation over trajectories generated by following policy π, t is the time step, T is the final time step, $r(s_t, a_t)$ denotes immediate reward obtained after taking action a_t in state s_t, α is entropy temperature coefficient controlling the trade-off between reward maximization and exploration, and H is entropy.

To mitigate overestimation bias, SAC employs two Q-functions. The policy optimization objective is given by [20]:

$$L_\pi = E_{st\sim D}\,[\alpha log\pi(a_t|s_t) - - min_{i=1,2} Q_{\theta i}(s_t, a_t)] \quad (4)$$

where: α is the temperature parameter, D denotes the replay buffer, $E_{st\sim D}[\cdot]$ is the expectation over states s_t sampled from the replay buffer D, a_t denotes action sampled from the current policy in state s_t, $Q_{\theta i}(s_t, a_t)$ is the action-value function estimated by the i-th critic network.

Environment specifications

The environmental specifications presented in Table 2 for Ball, Forest, and in Table 3 for Car projects contain all possible movement actions, observations, and rewards.

Metrics

The following general metrics are applied:

- Cumulative reward – total reward per episode,
- Total steps – total step number per episode,
- Success rate – number of successful episodes divided by the number of all episodes.

For the Forest project, the metrics are applied:

- Starvation rate – the number of episodes in which the agent ran out of health, divided by the number of all episodes,
- Time out rate – the number of episodes that reached the maximum step divided by the total number of all episodes.

In the case of the Car project, apart from the Time out rate, the additional metrics are utilized:

- Average speed – average speed reached during episode,
- Average track time – average travel time for the route,
- Average max speed reached – average maximum speed reached in the project.

The primary criterion used to evaluate the performance of the policy is the cumulative reward, which represents the overall objective optimized by the agent. The remaining metrics, such as success rate and collision rate, are components of the reward function and are therefore reflected in the cumulative reward. However, the other metrics are included is to provide a more transparent and interpretable view of the agent's behavior and to facilitate the analysis of individual aspects of its performance.

RESULTS

Training environments

Depending on the environment, different simulation time scales were applied. In the Ball and Forest environments, the training time scale was set to 20, the default value. Regarding the Car environment, the training time was set to 4 due to the fact that it required more precise control and applied more complicated physics. During training, the agent observes the state of the environment, selects actions, and receives rewards or penalties depending on the outcome of those actions. Using Unity ML-Agents, the agent gradually updates its policy to maximize the cumulative reward and improve its performance. The training process was repeated over many episodes, enabling the agent to discover effective strategies without explicitly programming the desired behavior. No early stopping was applied. Each configuration for all policies was trained once. All the best-performing configurations are gathered in Table 5.

The training processes for the three projects are gathered in Figures 5–7. These graphs show

Table 2. Ball and forest environment specification

Parameter		Ball environment	Forest environment
Movement	Forward movement	{-1, 0, 1}	{-1, 0, 1}
	Rotation	Counter-clockwise, Idle, Clockwise	Counter-clockwise, Idle, Clockwise
Observations	Rotation	1	1
	Goal position	2	-
	Ball position	2	-
	Health	-	1
	Satiety	-	1
	Woodpile status	-	1
	Have wood	-	1
Rewards	Ball approaching goal reward	ballGoalDistDiff · 0.25 (each step)	-
	Player approaching ball reward	playerBallDistDiff · 0.1 (each step)	-
	Ball collision reward	0.05	-
	Ball transported to base reward	10 · (maxStep − StepCount) / maxStep	-
	Wooden log gather reward	-	1
	Wooden log placement reward	-	1
	Woodpile completion reward	-	10 · (maxStep − StepCount) / maxStep
	Carrying log toward woodpile reward	-	0.005 (each step)
	Food consume reward	-	0.7
Penalties	Time step penalty	-0.001 (each step)	-0.0015 (each step)
	Wall collision penalty	-0.1	-
	Wall collision stay penalty	-0.01	-
	Hunger penalty	-	-0.004 (each step)
	Death penalty	-	-3 · ((maxParts − parts) / maxParts + (maxStep − StepCount) / maxStep)
	Tree collision penalty	-	-0.04
	Tree collision stay penalty	-	-0.005 (each step)
Ray Sensor	Detectable tags	Wall, Ball, Goal	WoodenLog, food, woodpile, enemy, tree
	Number of rays / angle	5 / 100°	8 / 80°

Note: maxStep – maximum number of steps for the environment; stepCount – current number of steps in the environment; maxParts – maximum number of wooden logs to collect; parts – current number of wooden logs collected; ballGoalDistDiff – distance difference between the ball and the goal; playerBallDistDiff – distance difference between the player and the ball.

the ratio of changes in selected metrics over time. Figures 5–7 a) and b) show the change in cumulative reward during training, with the difference being that the first one shows the average cumulative reward earned in a given training step, while the second shows it in real time. Figures 5–7 c) show the change in the average number of steps per episode in real time. Unity steps correspond to single iterations of the simulation, while training steps indicate the number of updates to the neural network weights. The difference stems from the fact that a single network update can use multiple Unity steps, as well as from agent settings such as Decision Period. When analyzing Figures 5–7 there is a clear difference in training length between the agent trained with the PPO algorithm and the one trained with SAC. The policy trained with PPO takes much less time than the one with SAC. The agent trained with SAC needs about half as many steps as the agent trained with PPO to achieve similar results. When training in the Car environment, a clear difference can be observed in the total reward and the number of steps per episode. Other training sessions in different environments do not yield such clear results.

Table 3. Car environment specification

Movement	Throttle	-1 to 1
	Steering (right/left)	-1 to 1
	Braking	0 or 1
Observations	Rotation	1
	Speed	1
	Rotation to next checkpoint	1
	Velocity	2
	Is max throttle input	1
Rewards	Checkpoint reward	1 · e^(-deltaStep · 0.004)
	Finish reward	20 · (stepsLeftToRestart)
	Drive toward checkpoint	0.001 (3 currentSpeed / maxSpeed) *(each step)*
	Speed reward	0.003 (currentSpeed / maxSpeed) *(each step)*
	Look at checkpoint	0.0025 *(each step)*
	Max throttle input	0.002 *(each step)*
Penalties	Time penalty	-0.003 *(each step)*
	Low speed penalty	-0.003 *(each step)*
	Drive away from checkpoint	-0.002 *(each step)*
	Collision penalty	-0.4
	Collision stay penalty	-0.003 *(each step)*
	Braking at low speed	-0.005 *(each step)*
Ray sensor (around player)	Detectable Tags	Checkpoint, TrackGround, CornerBorder, Wall
	Number of rays / angle	8 rays / 160°
Ray sensor (looking over checkpoint)	Detectable tags	Checkpoint, TrackGround, CornerBorder, Wall
	Number of rays / angle	4 rays / 30°

Note: currentSpeed – current vehicle speed; maxSpeed – maximum possible vehicle speed; e ≈ 2.72; stepsLeftToRestart – number of steps remaining to complete the current episode; deltaStep – difference in steps between checkpoints on the route.

Testing agents' performance

Tests were conducted for each trained agent. Three independent tests were performed for each configuration in all policies with the best-performing hyperparameters (Table 5) to reduce the influence of random initialization and environmental stochasticity. In total, 48 tests were conducted for three environments utilizing the same seed. During testing, the agent's policy is kept fixed, and its performance is assessed based on predefined metrics such as success rate, average steps, and cumulative reward. The obtained results can be compared between various algorithms, training configurations, and environmental complexities to determine the most effective behavior.

The tests covered all trained PPO and SAC policies, as configured in Tables 4–5. The average results obtained by the policies are compared in Tables 7–9. All test environments use the same seed to generate the environment, which means that all test environments were randomized but repeated for each policy test. Figures 5–7 show graphs generated during training for individual environments.

Ball environment

For the Ball environment, 1000 episodes were run for each trained policy. The highest reward was obtained by the agent trained with the SAC algorithm with the V0 configuration, achieving a total reward of 12.27, a success rate of 99.87%, and the lowest average number of steps of 295.40 (Table 6). The best policy trained with the PPO algorithm was the one with the initial V1 configuration, achieving 99.83% success rate, a total reward of 12.14, and 351.63 steps.

In the case of the SAC algorithm, all policies, regardless of configuration, achieved very high and similar results. In contrast, policies trained with the PPO algorithm achieved lower and divergent results. Therefore, the SAC algorithm outperforms in simple environments.

Table 4. Training steps and duration for PPO and SAC across configurations (V0-V2)

Version	Steps	V0 Time [h]	V1 Time [h]	V2 Time [h]
B1-PPO	5M	1.768	2.296	1.966
B2-PPO	5M	2.065	2.179	1.991
B3-PPO	5M	2.118	2.186	1.997
B1-SAC	3M	4.980	4.115	4.906
B2-SAC	3M	4.909	4.078	4.841
B3-SAC	3M	4.927	4.060	5.386
F1-PPO	6M	3.079	2.982	2.643
F2-PPO	6M	3.067	3.377	2.732
F3-PPO	6M	3.056	2.736	2.749
F1-SAC	4M	7.155	6.436	7.068
F2-SAC	4M	7.012	6.335	6.926
F3-SAC	4M	7.255	6.343	7.664
C1-PPO	10M	4.486	4.390	4.252
C2-PPO	10M	4.395	4.499	4.117
C3-PPO	10M	4.386	4.523	4.428
C1-SAC	6M	9.838	8.992	10.11
C2-SAC	6M	9.957	9.133	9.263
C3-SAC	6M	9.821	9.411	10.37

Note: B – Ball, F – Forest, C – Car. A detailed description of the file configuration can be found in Tables 7–9.

Table 5. Key varying hyperparameters and training limits of the selected best-performing variants (V0-V2) for each scenario and algorithm

Environment	Ball		Forest		Car	
Algorithm	PPO	SAC	PPO	SAC	PPO	SAC
Variant	V1	V0	V2	V2	V0	V1
Batch size	128	256	128	256	1024	256
Buffer size	40,960	100,000	81,920	200,000	20,480	400,000
Hidden units	128	128	128	128	256	256
Max steps	5,000,000	3,000,000	6,000,000	4,000,000	10,000,000	6,000,000
Time horizon	64	64	64	64	64	128
Memory (Enabled)	-	-	-	-	Yes	Yes

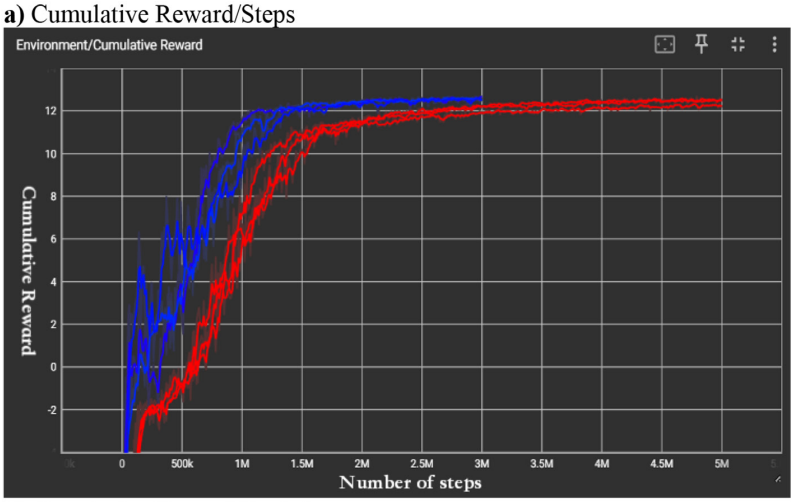
Forest environment

For the Forest environment, 500 episodes were run for each policy (Table 6). The agent trained with the SAC V2 algorithm achieved a success rate of 97.60%, several steps of 3390.99, and the highest average cumulative reward of 30.90. The agent trained with the PPO V2 algorithm achieved a cumulative reward of 30.55, a success rate of 97.73%, and a total of 3335.8 steps. Despite the lower success rate and the need for more steps, the first policy achieved a higher cumulative reward, demonstrating the effectiveness of the SAC algorithm in maximizing rewards and the importance of effective reward shaping. Therefore, the SAC algorithm is better suited to achieve the highest possible cumulative reward.

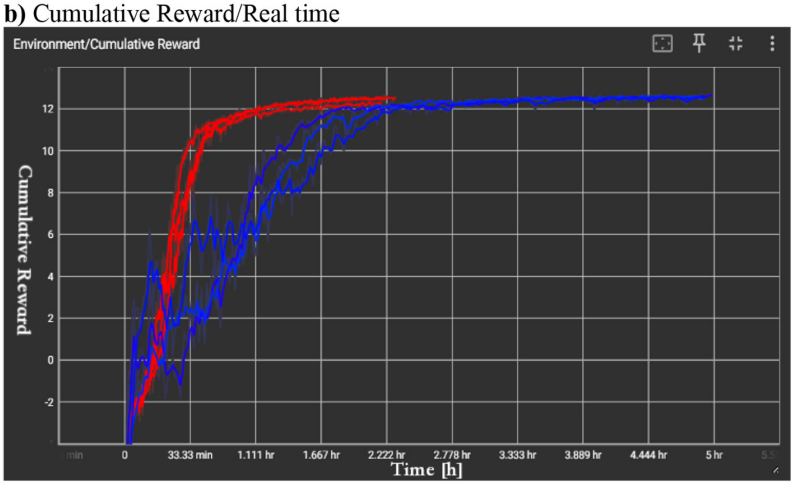
Car environment

For the Car environment, 500 episodes were run for each policy. Each agent trained with the PPO algorithm achieved better results than those trained with the SAC algorithm, regardless of the configuration (Table 6). At the same time, they achieved higher total rewards, significantly better times, average speeds, and maximum speeds. Despite the lower speed, which made driving easier, the policy trained with the SAC algorithm resulted in an average of 3 times as many collisions.

a) Cumulative Reward/Steps



b) Cumulative Reward/Real time



c) Episode length/Real time

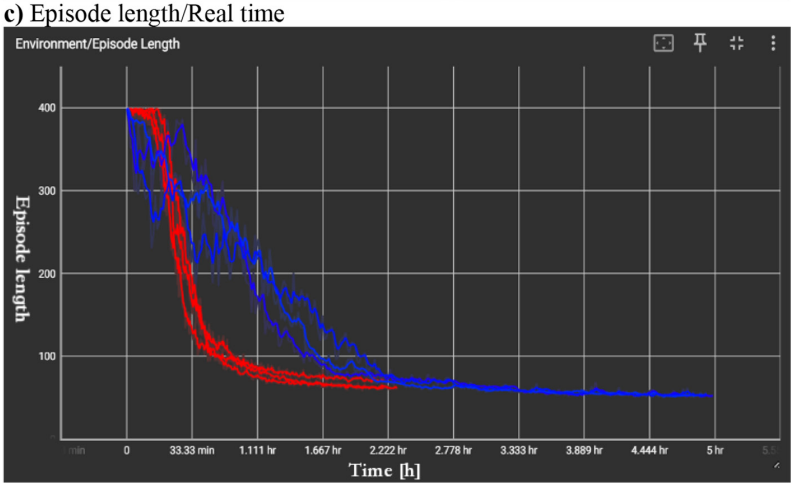


Figure 5. Ball environment training process
PPO – red, SAC – blue

Therefore, training with the PPO algorithm seems better suited to complex environments.

Ablation study

An ablation study was conducted to compare results across three developed environments with different hyperparameter settings during training (V0, V1, and V2).

Ball environment

This study verified that the best PPO algorithm configuration for the Ball environment was V1, with the smallest buffer and batch sizes

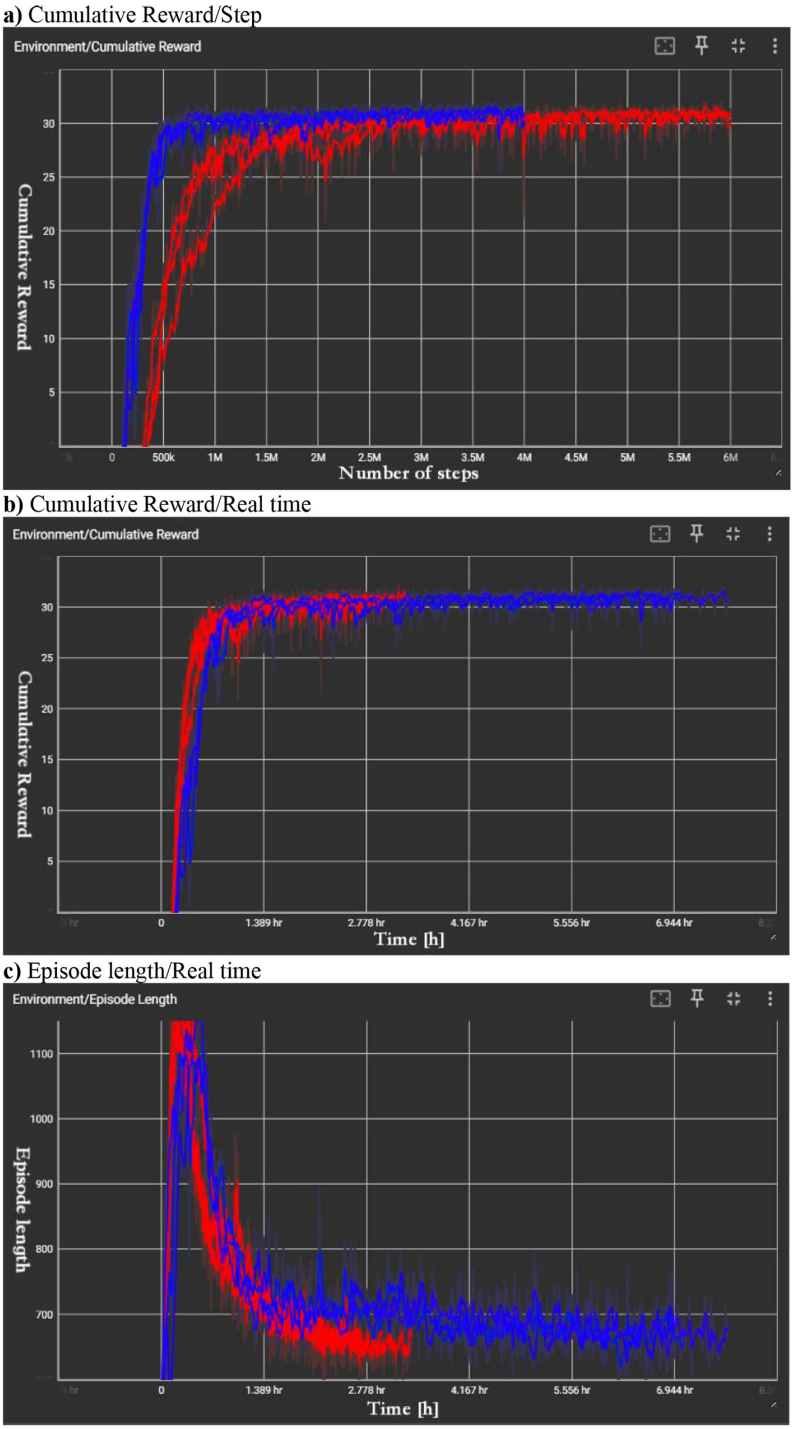


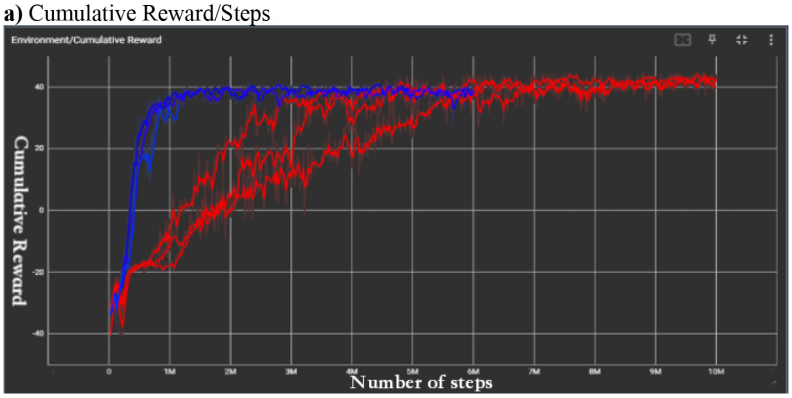
Figure 6. Forest environment training process
PPO – ted, SAC – blue

of 128 and 40960, respectively, as shown in Table 7. Increasing the batch size or buffer size negatively affected cumulative reward and success rate. For SAC, the V0 configuration with a batch size of 256 and a buffer size of 100000 achieved the highest cumulative reward, while other SAC variants performed slightly worse. These results indicate that PPO is more sensitive to batch size and buffer adjustments, whereas SAC is more robust but benefits from larger buffers.

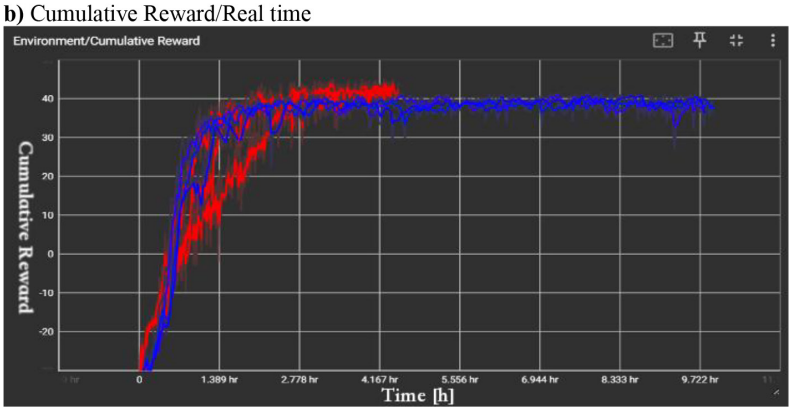
Forest environment

The study shows that for both PPO and SAC, the V2 setup with a batch size of 256 and a larger buffer size (81920 for PPO and 200000 for SAC),

a) Cumulative Reward/Steps



b) Cumulative Reward/Real time



c) Episode length/Real time

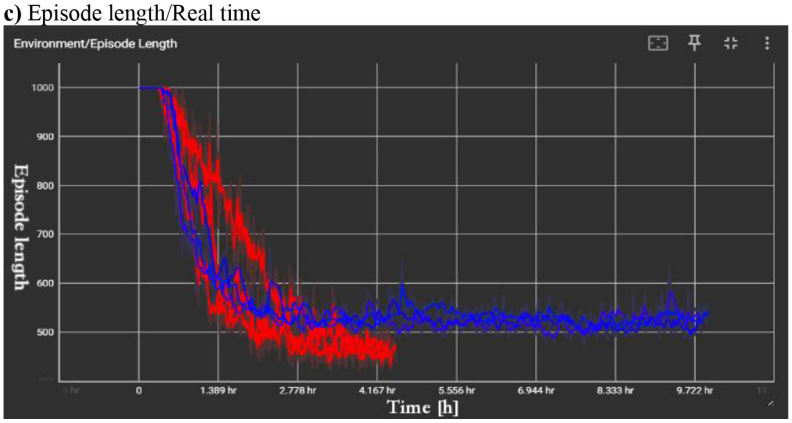


Figure 7. Car environment training
PPO – red, SAC – blue

Table 6. Best configurations for PPO and SAC across environments

Environment	Algorithm	Version	Success rate (%)	Average steps	Cumulative reward
Ball	PPO	V1	99.83 ± 0.06	351.63± 22.93	12.14 ± 0.13
Ball	SAC	V0	99.87 ± 0.12	295.40 ± 9.92	12.27 ± 0.06
Forest	PPO	V2	97.73 ± 0.5	3335.80±16.51	30.55 ± 0.05
Forest	SAC	V2	97.60 ± 0.72	3390.77± 77.40	30.90 ± 0.04
Car	PPO	V1	92.73 ± 0.5	1410.70 ± 3.80	41.15 ± 0.20
Car	SAC	V2	94.90 ± 0.78	1595.10± 16.53	38.09 ± 0.23

as presented in Table 8, consistently led to the highest success rates and cumulative rewards. It also resulted in the lowest rates of starvation and timeouts. Smaller batch sizes led to slightly reduced performance and less stable learning. The buffer size had the greatest effect on learning performance, while the batch size helped with stability and reward maximization.

Car environment

The ablation study for the PPO Car environment, presented in Table 9, shows that the V1 configuration achieved the highest total reward, fewest collisions, and shortest route time. Increasing the buffer and batch sizes decreased the success rate and total reward, suggesting that larger buffers may slow the agent's reactions in this environment. For SAC, a larger batch size results in reduced stability and total reward. This means that batch size had a strong impact on SAC stability, while buffer size played a more significant role for PPO.

Ablation study, presented in Table 7, shows that both PPO and SAC are relatively robust to changes in the investigated hyperparameters, as all variants achieved very high success rates (>98.8%). For PPO, reducing the batch size from 256 to 128 (PPO V1) substantially improved the average number of steps, from 483.30 to 351.63, while also increasing the cumulative reward. Increasing the buffer size to 81,920 (PPO V2) further improved the success rate to 99.90%, although the average number of steps increased compared with PPO V1. For SAC, increasing the buffer size from 100,000 to 200,000 (SAC V2) reduced the average number of steps from 330.40 to 306.77 and slightly increased the cumulative reward, while the success rate decreased marginally. Overall, the results suggest that batch size has a noticeable influence on PPO performance, whereas buffer size has a more pronounced effect on SAC, particularly in terms of episode efficiency and cumulative reward.

The ablation study in the Forest environment shows that PPO performance improves consistently with the tested parameter modifications (Table 8). Reducing the batch size from 256 to 128 (PPO V1) increased the success rate from 96.13% to 96.93% and reduced both the starvation rate and average number of steps. Increasing the buffer size to 81,920 (PPO V2) provided the best PPO results, achieving the highest success rate (97.73%), the lowest starvation rate (0.93%), and the highest cumulative reward (30.55). For SAC, the effect of batch size and buffer size was more pronounced. Reducing the batch size to 128 (SAC V1) substantially degraded performance, decreasing the success rate to 92.73% and increasing the starvation rate to 4.87%. In contrast, increasing the buffer size to 200,000 (SAC V2) considerably improved performance, resulting in a 97.60% success rate, the lowest starvation and termination rates (1.20%), and the highest cumulative reward (30.90). Overall, the results indicate that buffer size has a particularly important impact on SAC performance in the Forest environment, while PPO benefits more consistently from both the reduced batch size and increased buffer size.

The ablation study in the Car environment reveals different sensitivities to hyperparameter changes for PPO and SAC (Table 9). For PPO, reducing the batch size from 1024 to 512 (PPO V1) slightly increased the success rate from 92.40% to 92.70%, while also reducing the average number of steps, collision count, and route time, increasing the cumulative reward to 41.15. However, increasing the buffer size to 40,960 (PPO V2) negatively affected performance, resulting in a lower success rate (89.77%) and cumulative reward (38.77), as well as a higher timeout rate. For SAC, reducing the batch size from 512 to 256 (SAC V1) substantially improved the success rate from 91.33% to 95.10%, while reducing the average steps, collision count, and route time. Decreasing the buffer size to 200,000 (SAC V2) maintained a similarly high success rate (94.90%)

Table 7. Ablation study for Ball environment utilizing PPO and SAC algorithms

Parameter	PPO V0	PPO V1	PPO V2	SAC V0	SAC V1	SAC V2
BaS	256	128	256	256	128	256
BS	40960	40960	81920	100000	100000	200000
SR (%)	98.83± 1.85	99.83± 0.06	99.90± 0.10	99.87± 0.12	99.87± 0.15	99.43± 0.46
AS	483.30± 64.21	351.63± 22.93	442.07± 35.14	295.40± 9.92	330.40± 12.42	306.77± 9.31
CR	11.54± 0.63	12.14± 0.13	11.78± 0.04	12.27± 0.06	12.07± 0.09	12.18± 0.07

Notes: BaS – Batch size, BS – Buffer size, SR – Success rate, AS – Average steps, CR – Cumulative reward.

Table 8. Ablation study for the Forest environment utilizing PPO and SAC algorithms

Parameter	PPO V0	PPO V1	PPO V2	SAC V0	SAC V1	SAC V2
BaS	256	128	256	256	128	256
BS	40960	40960	81920	100000	100000	200000
SR (%)	96.13± 2.5	96.93± 0.7	97.73± 0.5	95.33± 1.22	92.73± 1.55	97.60± 0.72
StR (%)	1.53± 1.81	1.27± 0.41	0.93± 0.16	2.00± 0.35	4.87± 1.22	1.20± 1.11
TR (%)	2.33± 0.81	1.80± 0.53	1.33± 0.61	2.67± 0.99	2.40± 0.35	1.20± 0.40
AS	3478.50± 189.78	3278.20± 36.05	3335.80± 16.51	3474.47± 53.52	3576.90± 145.46	3390.77± 77.40
CR	29.95± 0.58	30.38± 0.08	30.55± 0.05	30.15± 0.03	29.98± 0.24	30.90± 0.04

Notes: BaS – Batch size, BS – Buffer size, SR – Success rate, StR – Starvation rate

and further reduced the average collision count to 1.50, while achieving the highest cumulative reward (38.09) among SAC variants. Overall, PPO V1 achieved the best overall PPO performance, whereas SAC benefited particularly from a smaller batch size and a reduced buffer size, with SAC achieving higher success rates than PPO in the Car environment.

DISCUSSION

In this study, the PPO and SAC algorithms were compared in three environments with increasing levels of complexity and different action spaces. In contrast to previous studies, which generally assessed reinforcement learning algorithms on individual benchmark setups, the proposed framework enables the isolated analysis of how task complexity affects algorithmic performance. Ball environment tests proved that the SAC algorithm outperformed PPO in terms of average cumulative reward. This can be attributed to better sampling efficiency enabled by SAC's off-policy nature and its entropy-maximizing objective, which favors exploration in environments with low dynamic complexity. The superior performance of SAC in simpler environments is consistent with its off-policy architecture, which allows more efficient reuse of collected experiences and promotes broader exploration through entropy

Table 9. Ablation study for the Car environment utilizing PPO and SAC algorithms

Parameter	PPO V0	PPO V1	PPO V2	SAC V0	SAC V1	SAC V2
BaS	1024	512	1024	512	256	256
BS	20480	20480	40960	400000	400000	200000
SR (%)	92.40± 1.40	92.73± 0.5	89.77± 2.05	91.33± 6.74	95.10± 1.41	94.90± 0.78
TR (%)	7.60± 1.40	7.27± 0.5	10.23± 2.05	8.67± 6.74	4.90± 1.41	5.10± 0.78
AS	1438.53± 33.37	1410.70± 3.80	1516.15± 17.42	1679.40± 101.04	1570.37± 23.35	1595.10± 16.53
AC	0.80± 0.10	0.57± 0.06	0.8± 0.10	2.60± 1.78	2.3± 0.66	1.57± 0.12
AMS	25.87± 0.23	25.60± 0.36	25.0± 0.70	20.47± 0.81	20.23± 1.14	21.13± 0.97
ASp	15.47± 0.29	15.53± 0.21	14.87± 0.12	13.43± 0.45	14.00± 0.10	13.93± 0.06
ART (s)	28.79± 0.67	28.22± 0.07	30.24± 0.44	33.61± 2.02	31.39± 0.47	31.81± 0.47
CR	40.41± 1.04	41.15± 0.20	38.77± 0.71	35.61± 3.80	37.62± 0.56	38.09± 0.23

Notes: BaS – Batch size, BS – Buffer size, SR – Success rate, TR – Timeout rate, AS – Average steps, AC – Average collision count, AMS – Average max speed, ASp – Average speed, ART – Average route time, CR – Cumulative reward

regularization. These characteristics are particularly beneficial when the environment is relatively easy to explore, and optimal policies can be identified with fewer stability constraints [20,21]. Conversely, PPO benefits from conservative policy updates, which reduce the risk of policy collapse and facilitate stable optimization in environments requiring long sequential decision-making [18,19].

In the Forest environment, both algorithms achieved similar results. SAC algorithm configuration achieved the highest cumulative reward, but policies trained with the PPO algorithm had higher average effectiveness in multiple runs. This suggests that SAC-trained policies have a higher potential maximum quality, but at the cost of greater variance in results. PPO, on the other hand, due to stable on-policy updates and limited policy changes, proved more resistant to environmental unpredictability and reward signal noise.

The obtained results are in agreement with recent studies [25], where SAC demonstrated superior maximum performance compared with PPO under similar environmental conditions, while PPO maintained lower variance and greater stability across repeated experiments. This consistency suggests that the relative advantage of SAC and PPO depends strongly on the balance between exploration requirements, task complexity, and environmental uncertainty.

In the Car environment, the policy trained with the PPO algorithm achieved a much higher reward, regardless of the algorithm configuration. The results are consistent with those from the literature review, confirming the clear advantage of the PPO algorithm in the time trial task [28]. One possible explanation for the lower SAC performance is its sensitivity to hyperparameter settings and the design of the reward function.

CONCLUSIONS

In this study, the performance of the SAC and PPO reinforcement learning algorithms was compared. The research showed that the SAC algorithm has greater potential in simple discrete action environments. In continuous-control driving environments, PPO significantly outperformed SAC, demonstrating its superiority in highly dynamic continuous tasks.

Beyond the comparative evaluation itself, this study introduced a controlled benchmarking framework composed of three Unity environments with progressively increasing task complexity. By maintaining identical training conditions across all experiments, the proposed framework enables the isolated assessment of the influence of task complexity on reinforcement learning performance. This contributes to explaining the inconsistent findings reported in previous studies and provides a reproducible methodology for future comparative analyses of reinforcement learning algorithms. Experiments demonstrated that the algorithm performance highly depends on the characteristics of the test environment. The obtained findings indicate that neither PPO nor SAC should be regarded as universally superior. Instead, their effectiveness depends on the interaction between task complexity, action space, and environmental dynamics. Consequently, the presented results provide practical guidance for selecting an appropriate reinforcement learning algorithm for specific application scenarios.

The experimental results suggest that algorithm selection should be guided by environment complexity and task characteristics, rather than by a universally superior RL algorithm. SAC showed strong performance in simpler tasks, whereas PPO demonstrated greater robustness in continuous-control environments. The ablation studies further indicate that PPO is particularly sensitive to batch size, while SAC is more strongly affected by buffer size. Therefore, algorithm-specific hyperparameter tuning should be performed for each environment.

REFERENCES

1. Dziuba-Koziel M, Koziel G, Markiewicz J, Kochanowicz M, Dorosz D, Miluski P, et al. Machine learning-assisted design of inverted gradient multiring optical fibres for flat-top beam generation. Opt Express. 2026 Feb;34(4):7159–7173. https://doi.org/10.1364/OE.587143
2. Asaad H, Askar S, Kakamin A, Faiq N. Exploring the impact of artificial intelligence on human-robot cooperation in the context of Industry 4.0. Appl Comput Sci. 2024;20(2):138–156. https://doi.org/10.35784/acs-2024-21
3. Dezhdar AH, Rahmati D, Mohtashami MA, Sharifi I. PPO and SAC reinforcement learning based

reference compensation. In: 2024 12th RSI International Conference on Robotics and Mechatronics (ICRoM). IEEE; 2024. p. 344–350. https://doi.org/10.1109/ICRoM64545.2024.10903564

4. Campbell M, Hoane AJ Jr, Hsu F-H. Deep Blue. Artif Intell. 2002;134(1–2):57–83. https://doi.org/10.1016/S0004-3702(01)00129-1
5. Kleczynski D, Drzał J, Pawłowicz B. Adaptive traffic signal control using proximal policy optimization transformer and RFID-based vehicle detection in the simulation of urban mobility environment. Adv Sci Technol Res J. 2026;20(5):372–385. https://doi.org/10.12913/22998624/216568
6. Skublewska-Paszkowska M, Powroźnik P, Barszcz M, Dziedzic K, Aristodou A. Identifying and animating movement of Zeibekiko sequences by spatial temporal graph convolutional network with multi attention modules. Adv Sci Technol Res J. 2024;18(8):217–227. https://doi.org/10.12913/22998624/193180
7. Berner C, Brockman G, Chan B, Cheung V, Dębiak P, Dennison C, et al. Dota 2 with large scale deep reinforcement learning. arXiv. 2019;arXiv:1912.06680. https://doi.org/10.48550/arXiv.1912.06680
8. Vinyals O, Babuschkin I, Czarnecki WM, Mathieu M, Dudzik A, Chung J, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature. 2019;575(7782):350–354. https://doi.org/10.1038/s41586-019-1724-z
9. Penelas G, Barbosa L, Reis A, Barroso J, Pinto T. Machine learning for decision support and automation in games: a study on vehicle optimal path. Algorithms. 2025;18(2):106. https://doi.org/10.3390/a18020106
10. Bin Ramlan AA, Ali AM, Abdul Hamid NH, Osman R. The implementation of reinforcement learning algorithm for AI bot in fighting video game. In: 2021 4th International Symposium on Agents, Multi-Agent Systems and Robotics (ISAMSR). IEEE; 2021. p. 96–100. https://doi.org/10.1109/ISAMSR53229.2021.9567749
11. Supli AA, Siqi X. Leveraging deep reinforcement learning for dynamic NPC behavior and enhanced player experience in Unity3D. Entertain Comput. 2025;55:101007. https://doi.org/10.1016/j.entcom.2025.101007
12. Raut U, Galchhaniya P, Nehete A, Shinde R, Bhoite A. Unity ML-Agents: revolutionizing gaming through reinforcement learning. In: 2024 2nd World Conference on Communication & Computing (WCONF). IEEE; 2024. p. 1–7. https://doi.org/10.1109/WCONF61366.2024.10692314
13. Xie J. Research on key technologies base Unity3D game engine. In: 2012 7th International Conference on Computer Science & Education (ICCSE). IEEE; 2012. p. 695–699. https://doi.org/10.1109/ICCSE.2012.6295169
14. Ouyang SG, Wang G, Yao JY, Zhu GHW, Liu ZY, Feng C. A Unity3D-based interactive three-dimensional virtual practice platform for chemical engineering. Comput Appl Eng Educ. 2018;26(1):91–100. https://doi.org/10.1002/cae.21863
15. Matulis M, Harvey C. A robot arm digital twin utilising reinforcement learning. Comput Graph. 2021;95:106–114. https://doi.org/10.1016/j.cag.2021.01.011
16. Lee H, Dahouda MK, Joe I. Character behavior automation using deep reinforcement learning. IEEE Access. 2023;11:101435–101442. https://doi.org/10.1109/ACCESS.2023.3313737
17. Giral F, Gomez I, Le Clainche S. Intercepting unauthorized aerial robots in controlled airspace using reinforcement learning. Aerospace Sci Technol. 2025;168:110985. https://doi.org/10.1016/j.ast.2025.110985
18. Radwan MO, Sedky AAH, Mahar KM. Obstacles avoidance of self-driving vehicle using deep reinforcement learning. In: 2021 31st International Conference on Computer Theory and Applications (ICCTA). IEEE; 2021. p.215-220. https://doi.org/10.1109/ICCTA54562.2021.9916640
19. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. arXiv [Preprint]. 2017;arXiv:1707.06347. https://doi.org/10.48550/arXiv.1707.06347
20. Haarnoja T, Zhou A, Abbeel P, Levine S. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: Proceedings of the 35th International Conference on Machine Learning. PMLR; 2018. p. 1861–1870.
21. Muzahid AJM, Kamarulzaman SF, Rahman MA. Comparison of PPO and SAC algorithms towards decision making strategies for collision avoidance among multiple autonomous vehicles. In: 2021 International Conference on Software Engineering & Computer Systems and 4th International Conference on Computational Science and Information Management (ICSECS-ICOCSIM). IEEE; 2021. p. 200–205. https://doi.org/10.1109/ICSECS52883.2021.00043
22. Evripidou E, Aristidou A, Charalambous P. Collaborative museum heist with reinforcement learning. Comput Anim Virtual Worlds. 2023;34(3–4):e2158. https://doi.org/10.1002/cav.2158
23. Kozlov D. Comparison of reinforcement learning algorithms in problems of acquiring locomotion skills in 3D space. In: 2022 VIII International Conference on Information Technology and Nanotechnology (ITNT). IEEE; 2022. p. 1–5. https://doi.org/10.1109/ITNT52450.2021.9649145

24. Taha M, Badawy A, Hegazy M. Developing hexapod locomotion through reinforcement learning. In: 2025 Intelligent Methods, Systems, and Applications (IMSA). IEEE; 2025. p. 61–66. https://doi.org/10.1109/IMSA65733.2025.11166981
25. Bayona Latorre AL. Comparative study of SAC and PPO in multi-agent reinforcement learning using Unity ML-Agents [master's thesis]. Bogotá: Universidad de los Andes; 2023.
26. Wang Y, Zheng S, Yang Z, Zhu Y, Zhang S, Leng J, et al. Digital twin-empowered robotic arm manipulation with reinforcement learning: a comprehensive survey. Robot Comput Integr Manuf. 2026;98:103151. https://doi.org/10.1016/j.rcim.2025.10315
27. Andrychowicz OM, Baker B, Chociej M, Józefowicz R, McGrew B, Pachocki J, et al. Learning dexterous in-hand manipulation. Int J Robot Res. 2019;39(1):3–20. https://doi.org/10.1177/0278364919887447
28. Xu C, Zhu R, Yang D. Karting racing: a revisit to PPO and SAC algorithm. In: 2021 International Conference on Computer Information Science and Artificial Intelligence (CISAI). IEEE; 2021. p. 310–316. https://doi.org/10.1109/CISAI54367.2021.00066