

Optimizing and evaluating proximal policy optimization and soft actor-critic agents in Unity ML-Agents-based games

Muhammad Fakhri Loebis¹, Andry Chowanda²

¹Department of Computer Science, School of Computer Science, Bina Nusantara University, Jakarta, Indonesia

²Master of Computer Science, BINUS Graduate Program, Bina Nusantara University, Jakarta, Indonesia

Article Info

Article history:

Received Nov 14, 2025

Revised Jul 3, 2026

Accepted Sep 4, 2026

Keywords:

Agent optimization

Deep reinforcement learning

Proximal policy optimization

Reinforcement learning

Soft actor-critic

Unity ML-Agents

ABSTRACT

This study presents a comparative analysis of proximal policy optimization (PPO) and soft actor-critic (SAC) for training autonomous delivery agents in high-fidelity 3D environments using Unity ML-Agents. Both algorithms were evaluated with identical hyperparameters and reward functions across five independent runs to ensure statistical robustness. Results show that PPO achieves 31.4% higher final reward (6658.82 versus 5067.27) and superior policy improvement consistency (ratio 2.00 versus 0.31). However, PPO exhibits greater reward variability (coefficient of variation (CV) 0.533 versus 0.107) and slower convergence (33 versus 6 steps). SAC demonstrates faster initial convergence and superior stability with lower performance variance. These findings indicate that PPO is preferable for applications prioritizing maximum final performance in stable environments, while SAC is more suitable for tasks requiring adaptability and consistent performance under dynamic conditions. This study provides practical guidance for researchers implementing reinforcement learning (RL) in Unity-based simulations for autonomous systems.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Andry Chowanda

Master of Computer Science, BINUS Graduate Program, Bina Nusantara University

Jl. K. H. Syahdan No. 9, Kemanggis, Palmerah, Jakarta Barat 11480, Indonesia

Email: achowanda@binus.edu

1. INTRODUCTION

The emergence of artificial intelligence represents one of the most significant technological developments of the 21st century, with machine learning serving as its foundational pillar. Deep reinforcement learning (DRL), in particular, has demonstrated remarkable success across diverse domains, including autonomous vehicles, digital entertainment, and robotics [1]-[4]. Within this landscape, the gaming industry has emerged as a critical testbed for AI development, providing rich, interactive environments for training and evaluating intelligent agents [5]. Recent advances have enabled the creation of sophisticated autonomous agents capable of self-adaptation and complex problem-solving within virtual environments, exhibiting decision-making capabilities that increasingly approach human-level performance [6].

Reinforcement learning (RL) provides a framework for training agents through direct interaction with the environment, in which agents learn optimal behaviors through a continuous cycle of actions, observations, and reward-based feedback [7]. Through trial and error, agents maximize long-term cumulative reward, with continuous adaptation to dynamic environments as the core principle [8]. Complex domains such as video games represent ideal testbeds for RL, as they require agents to make sequential decisions under uncertainty [9].

Despite its promise, training robust RL agents in complex environments remains challenging. In three-dimensional simulations, state and action spaces are often multidimensional, leading to inefficient

policy optimization, convergence difficulties, and prolonged training requirements [10], [11]. Actor-critic architectures address these limitations by separating policy learning from value estimation, thereby enhancing both stability and learning efficiency [12], [13].

Among contemporary DRL algorithms, proximal policy optimization (PPO) [14] and soft actor-critic (SAC) [15] have emerged as prominent actor-critic methods. PPO constrains policy updates through a clipped surrogate objective, preventing destructive policy changes while ensuring stable convergence [16]. SAC, conversely, employs maximum-entropy RL to optimize expected returns while maximizing policy entropy, thereby promoting exploration and improving robustness in continuous control tasks [17], [18]. Both algorithms have demonstrated state-of-the-art performance across various benchmarks, with recent surveys confirming their dominance in continuous control domains [19], [20].

However, existing literature predominantly evaluates PPO and SAC on conventional benchmarks such as OpenAI Gym, MuJoCo, and PyBullet [21], [22]. While useful for algorithmic comparison, these environments often lack the visual fidelity, spatial complexity, and physical realism characteristic of real-world applications. This limitation raises questions about the generalizability of benchmark results to practical scenarios [23]. The Unity ML-Agents Toolkit, developed by Unity Technologies, addresses this gap by integrating modern RL algorithms with the Unity game engine, enabling training within visually rich, physically realistic 3D environments [24], [25]. This platform enables more rigorous algorithm evaluation under conditions that closely approximate real-world complexity, including dynamic obstacles, varied terrain, and goal-oriented tasks [26].

Despite the availability of this powerful platform, the literature lacks a comprehensive, direct comparison of PPO and SAC within the Unity ML-Agents framework. While recent studies have begun exploring DRL in Unity-based environments [27]–[30], no systematic comparison of the convergence characteristics, adaptive capacity, and operational stability of these two algorithms has been conducted. This gap limits understanding of their relative strengths in realistic, dynamic 3D environments, knowledge essential for advancing autonomous systems that require robust, efficient decision-making in complex real-world contexts.

This study addresses this research gap by systematically training and evaluating PPO and SAC within a controlled, uniform 3D simulation environment developed using Unity ML-Agents. The primary contributions of this work are threefold: i) a comprehensive empirical analysis of PPO and SAC in high-fidelity 3D environments, evaluating learning speed, final performance, stability, and sample efficiency; ii) an investigation of each algorithm's responsiveness to dynamically introduced environmental changes during training; and iii) practical guidelines for researchers and practitioners selecting and configuring PPO and SAC for complex 3D simulation projects.

The remainder of this paper is organized as follows: section 2 presents the theoretical framework and methodology, including the experimental configuration and Unity environment design. Section 3 reports and discusses experimental results and performance analysis. Section 4 concludes with a summary of key findings and directions for future research.

2. METHOD

Figure 1 presents the proposed research model for optimizing and evaluating PPO and SAC algorithms within the Unity ML-Agents framework. The model consists of six main stages: i) problem formulation, ii) world design in Unity, iii) implementation of PPO and SAC methods, iv) method evaluation, v) model implementation on the agent, and vi) model evaluation. Each stage is designed to support the optimization and comparison of both algorithms in Unity-based 3D simulations.

2.1. Problem formulation

The initial stage of this research identifies the main problems that need to be addressed. This study aims to optimize the PPO algorithm to function as an autonomous delivery agent. The agent must be able to navigate to package pickup locations, complete deliveries, and avoid obstacles along the path. Based on this, four main research questions are formulated: i) how quickly can the agent learn and adapt to the environment? ii) how efficiently can the agent find and deliver packages? iii) how capable is the agent in avoiding obstacles?, and iv) does the learning process remain stable despite environmental changes?

The SAC algorithm is used as a benchmark to evaluate the stability, convergence speed, and adaptability of PPO. Comparisons are conducted in random environments to ensure fair evaluation. Based on RL principles, this transportation problem is formulated as a Markov decision process (MDP) defined as (1):

$$M = (S, A, P, R, \gamma) \quad (1)$$

where:

- S represents the set of possible *states* of the environment, such as the agent's position, the package's position, and the target location.
- A denotes the set of actions available to the agent (move forward, rotate, pick up, and deliver).
- $P(s'|s, a)$ is the transition probability from state s to s' after performing action a .
- $R(s, a)$ is the reward function received after the agent executes an action a in state s (e.g., +1 for successful pickup, +10 for delivery, and -1 for collision).
- γ (gamma) is the discount factor ($0 \leq \gamma \leq 1$) that determines the importance of future rewards.

$$J(\pi) = E_{\pi} [\sum_{t=0}^T \gamma^t R(s_t, a_t)] \quad (2)$$

where $\pi(a/s)$ represents the policy function, defining the probability of taking action a given the state s .

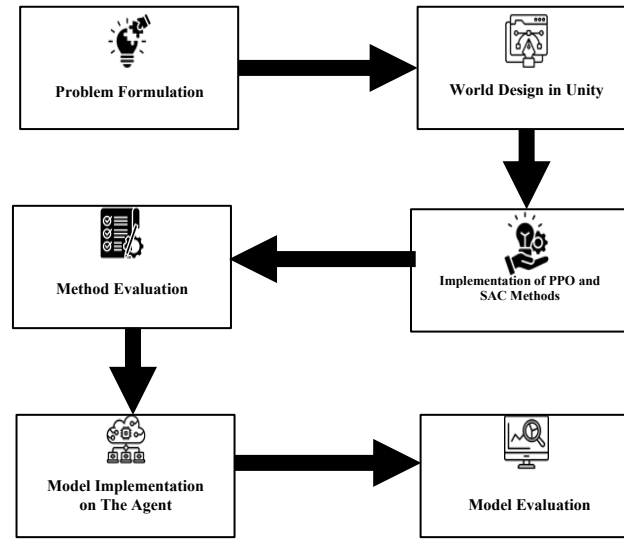


Figure 1. Research methodology framework model

2.2. World design in Unity

The second stage involves building a 3D simulation environment using the Unity game engine. The environment is designed to provide a realistic and challenging testbed for evaluating RL algorithms in autonomous delivery tasks. The environment consists of several key elements: static obstacles (walls, tables, and boxes) that the agent must avoid; dynamic obstacles that move in random patterns to test adaptability; packages that serve as collectible items representing delivery tasks; and a fixed delivery location where packages must be delivered. Table 1 presents the detailed specifications of the Unity environment. The map is sized at 20×20 units, providing sufficient space for navigation while maintaining complexity. Eight static obstacles are strategically placed to create challenging pathways, while two dynamic obstacles move in random patterns to introduce environmental unpredictability. Package locations change each episode to prevent overfitting and encourage generalization, while the delivery location remains fixed to provide a consistent goal. Each episode is limited to a maximum of 500 steps to ensure efficient training.

To ensure reproducibility and control variability: i) package positions are randomized at the start of each episode using Unity's Random.Range function with a fixed seed per run; ii) dynamic obstacles move according to pre-defined random patterns, with their behavior initialized using the same random seed to maintain consistency across runs; iii) five random seeds (e.g., 42, 123, 456, 789, and 1011) are used for each algorithm to capture performance variance, following [8]. The episode length is capped at 500 steps, consistent with the time_horizon parameter in ML-Agents which trades off bias and variance in value estimates. The agent is equipped with multiple sensors to perceive its environment, as detailed in Table 2. Position and rotation sensors provide spatial awareness, while package and target position sensors enable goal-oriented navigation. Collision detection provides immediate feedback on unsafe actions, and raycast sensors in eight directions allow the agent to perceive obstacles at a distance, enabling proactive obstacle avoidance. This multi-modal sensor suite ensures the agent receives sufficient information to learn effective navigation policies.

Table 1. Environment specifications

Parameter	Specification
Map size	20×20 units
Number of static obstacles	8 (walls, tables, boxes)
Number of dynamic obstacles	2 (moving with random patterns)
Package locations	3 different points and changing each episode
Delivery location	1 fixed point
Maximum steps per episode	500 steps

Table 2. Sensors and agent inputs

Sensor	Input received
Agent position	Coordinates (x, y, z) relative to the origin
Agent rotation	Orientation angles (yaw, pitch, roll)
Package position	Coordinates relative to the agent position
Target position	Coordinates relative to the agent position
Collision detection	Boolean (0=no collision, 1=collision occurred)
Raycast (8 directions)	Distance to nearest obstacle in each direction

The training configuration parameters are summarized in Table 3. A total of 2,000,000 time steps is allocated for training, with the first 500,000 steps serving as initial exploration. Five random seeds are used for each experiment to ensure statistical robustness and reproducibility, following [8], who emphasize the importance of multiple runs in DRL experiments. The Unity 2022.3 LTS engine provides stable long-term support, while ML-Agents Toolkit version 2.0.0 offers the latest RL integration features. The hardware specifications ensure consistent computational performance across all experiments.

Table 3. Training configuration

Parameter	Value
Initial time steps	500,000
Total time steps	2,000,000
Number of random seeds	5 (for reproducibility)
Unity version	2022.3 LTS
ML-Agents version	2.0.0
Hardware	CPU: Intel Core i7-12700K, GPU: NVIDIA RTX 3060, and RAM: 16 GB

All experiments were conducted on a system running Ubuntu 20.04 LTS, with Unity 2022.3 LTS and ML-Agents Toolkit version 2.0.0. The RL algorithms were implemented using the ML-Agents Python API, which relies on PyTorch 1.12.0 as the backend. Training utilized a single NVIDIA RTX 3060 GPU (driver version 525) to accelerate neural network computations. The Unity ML-Agents Toolkit provides a flexible API that enables direct integration of RL algorithms into Unity simulations, allowing comprehensive testing in interactive 3D environments. This platform bridges the gap between conventional RL benchmarks and real-world applications by offering high visual fidelity and physical realism.

2.3. Implementation of proximal policy optimization and soft actor-critic methods

The PPO and SAC algorithms are implemented with uniform configurations to ensure a fair and unbiased comparison. Both algorithms operate within a continuous action space, where each action dimension accepts values in the range $[-1, 1]$. This continuous action space is chosen to reflect real-world control tasks, such as robotic navigation, where actions are naturally continuous rather than discrete. To provide a clear understanding of the learning process, we first describe the experimental pipeline that governs agent-environment interaction. The experimental pipeline follows a sequential process: i) the Unity environment generates observations and rewards based on agent actions; ii) these observations are passed through the agent's sensor suite (position, rotation, and raycast); iii) the PPO or SAC algorithm processes these inputs to produce actions; iv) the actions are executed in the environment, generating new states and rewards; and v) training metrics (reward, entropy, and loss) are logged via TensorBoard for analysis. During training, each algorithm interacts with the environment for a total of 2,000,000 steps, with policy updates performed after collecting the specified buffer size of experiences (20,480 for both algorithms).

The diagram illustrates the sequential flow from environment interaction, through sensor processing and algorithm decision-making, to action execution and policy update. Training metrics are logged for performance analysis. PPO is an on-policy actor-critic algorithm that optimizes policies by constraining policy updates through a clipped surrogate objective. This mechanism prevents destructive policy changes that could destabilize learning, making PPO particularly suitable for tasks requiring stable and reliable

convergence [11]. PPO employs a clipped surrogate objective that limits the magnitude of policy updates, ensuring that the new policy does not deviate excessively from the previous policy.

SAC, in contrast, is an off-policy actor-critic algorithm that incorporates maximum entropy RL. By maximizing both expected return and policy entropy, SAC encourages exploration and maintains stochasticity in action selection, which is especially beneficial for continuous control tasks where exploration is critical for discovering optimal behavior [12], [15]. SAC leverages an experience replay buffer to store past transitions, enabling more sample-efficient learning compared to on-policy methods. Table 4 presents the hyperparameters used for both algorithms. These hyperparameters were selected based on recommendations from the original algorithm publications and preliminary experiments to ensure optimal performance within the Unity environment. All hyperparameters that are common to both algorithms were set to identical values to isolate the effects of algorithmic differences.

Table 4. Algorithm hyperparameters

Parameter	PPO	SAC	Description
Learning rate	3e-4	3e-4	Speed of parameter updates
Batch size	128	128	Number of samples per update
Buffer size	20,480	20,480	Experience replay memory capacity
Discount factor (γ)	0.99	0.99	Weight for long-term rewards
Entropy coefficient	0.01	-	PPO entropy regularization coefficient
Target entropy	-	-1.0	SAC target entropy value
Clip range (ϵ)	0.2	-	PPO policy update clipping boundary
Soft update rate (τ)	-	0.005	SAC target network update rate
Hidden units	128	128	Number of neurons per hidden layer
Number of layers	2	2	Number of hidden layers
Activation function	ReLU	ReLU	Activation function type

Hyperparameters for both PPO and SAC were selected based on recommendations from the original publications [14], [15] and the Unity ML-Agents documentation. The learning rate of 3e-4 is within the typical range (1e-5 to 1e-3) and balances convergence speed with stability. The batch size of 128 for SAC falls within the recommended range of 128-1024 for continuous action spaces. For PPO, the clip range of 0.2 follows the standard value used to limit policy updates [14]. The discount factor of 0.99 prioritizes long-term rewards. We did not perform extensive hyperparameter tuning; instead, we used these fixed values to isolate the effects of algorithmic differences, a common approach in comparative RL studies [8]. The learning rate of 3e-4 is a commonly used value in DRL that balances convergence speed with stability. The batch size of 128 ensures sufficient samples for gradient estimation while maintaining computational efficiency. The discount factor $\gamma=0.99$ prioritizes long-term rewards, encouraging the agent to consider future consequences of its actions. For PPO, the entropy coefficient of 0.01 encourages exploration by preventing premature convergence to suboptimal policies, while the clip range $\epsilon=0.2$ limits policy updates to ensure stable learning. For SAC, the target entropy of -1.0 is set based on the dimensionality of the action space, following the heuristic that target entropy should be approximately $-\dim(A)$, where $\dim(A)$ is the action dimension. The soft update rate $\tau=0.005$ controls how quickly the target networks are updated, balancing learning stability with adaptability. Both algorithms share the same neural network architecture: two hidden layers with 128 neurons each, using rectified linear unit (ReLU) activation functions. This architecture provides sufficient capacity to learn complex policies while avoiding overfitting. The use of identical architectures ensures that performance differences can be attributed to algorithmic characteristics rather than network capacity.

2.4. Method evaluation

Evaluation assesses the effectiveness of each learning algorithm using several key metrics that capture different aspects of agent performance. These metrics are designed to provide a comprehensive understanding of how PPO and SAC behave during training and how they respond to environmental challenges. Convergence speed measures the number of time steps required for the average reward to stabilize within a consistent range. This metric captures initial learning efficiency, indicating how quickly an algorithm can acquire basic task competencies. Faster convergence is desirable for applications requiring rapid deployment or limited training budgets. Reward accumulation represents the total reward the agent has obtained over the course of training. This metric reflects the agent's final performance, quantifying how well the agent has learned to maximize cumulative reward by the end of the training period. Higher reward accumulation indicates more effective policy optimization. Learning stability is assessed through reward variance across episodes. Stable learning is characterized by low variance, indicating consistent improvement without disruptive fluctuations. High stability is crucial for real-world applications where unpredictable behavior could lead to unsafe outcomes. Adaptability measures how well the agent maintains performance

when the environment undergoes changes. This metric is evaluated by introducing dynamic obstacles and varying package locations during training. Robustness to change is essential for agents deployed in dynamic real-world environments. Sample efficiency is calculated as the reward ratio per time step, representing how effectively the algorithm utilizes each interaction with the environment. Higher sample efficiency reduces training time and computational requirements, which is particularly important for applications with limited interaction budgets. This evaluation framework is essential for determining the optimal strategy for stable and efficient learning. To ensure statistical robustness and reproducibility, each metric is calculated based on five independent runs with different random seeds. Results are presented as mean $\pm$ standard deviation, following the recommendation of Henderson *et al.* [8] regarding the importance of multiple runs in DRL experiments. This approach enables reliable comparison between algorithms and provides confidence that observed differences are attributable to algorithmic characteristics rather than random initialization effects.

2.5. Model implementation on the agent

After testing both models and evaluating their performance using the metrics described in section 2.4, the best-performing algorithm is implemented in the Unity environment for validation. This stage verifies that the agent can generalize its learned knowledge across different conditions and maintain consistent performance outside the training environment. Validation testing is a critical step in the research methodology, as it confirms whether the policies learned during training translate effectively to practical deployment scenarios. The validation testing phase evaluates the agent against three success criteria, which are summarized in Table 5. These criteria are designed to assess different aspects of agent behavior: safety, efficiency, and overall task completion.

Table 5. Success criteria for implementation

Criterion	Target	Measurement method
Delivery without collision	90% success rate	Number of episodes without collision
Movement efficiency	Average < 200 steps per delivery	Average time steps per delivery
Minimum score	100 points within 5 minutes	Total reward across 10 test sessions

The first criterion, delivery without collision, requires that the agent complete deliveries without colliding with any obstacles. A 100% success rate is targeted, meaning that in every episode, the agent must avoid collisions entirely. This criterion is measured by counting the number of episodes in which no collision occurs. Safety is a paramount concern in real-world autonomous systems, making this criterion essential for assessing the agent's practical viability. The second criterion, movement efficiency, evaluates how efficiently the agent navigates to complete deliveries. The target is an average of fewer than 200 time steps per delivery. This metric is calculated as the average number of time steps required to complete each delivery across all test episodes. Efficient movement indicates that the agent has learned optimal navigation policies rather than relying on inefficient exploration or wasteful maneuvers. The third criterion, minimum score, requires the agent to achieve a total reward of at least 100 points within a five-minute testing period. This criterion is measured by accumulating the total reward across 10 test sessions. The reward threshold ensures that the agent consistently performs well across multiple trials, demonstrating reliability rather than occasional success. Testing is conducted over 10 validation sessions, each with a maximum duration of 5 minutes. Each session continues until either the time limit is reached or five successful deliveries are completed, whichever occurs first. This design ensures that the agent is evaluated on both speed and endurance. The number of validation sessions is selected to yield statistically meaningful results while remaining computationally feasible, and the five-delivery target per session ensures sufficient task repetition to assess consistency. These validation criteria collectively ensure that the implemented agent not only performs well in controlled training conditions but also demonstrates the safety, efficiency, and reliability required for potential real-world deployment in autonomous delivery and navigation tasks.

2.6. Model evaluation

The final stage verifies the effectiveness of the trained model in operation. This evaluation assesses how well the agent performs when deployed in the Unity environment and provides quantitative evidence to answer the research questions formulated in subsection 2.1. Table 6 presents the assessment aspects and their corresponding performance targets. Each aspect is designed to measure a distinct dimension of agent capability.

Findings from this stage are analyzed to answer the research questions and will be presented in the results and discussion section. Completion time measures the average duration required to complete a single delivery. A target of less than 300 seconds per delivery ensures the agent operates with reasonable efficiency. Performance score represents the total reward accumulated during the test cycle, with a minimum target of 100 points. This metric reflects the agent's overall task achievement. Consistency is assessed through score variance

across test sessions, quantified by the coefficient of variation (CV). A target of less than 20% indicates stable and reliable performance across multiple trials. Task success measures the percentage of delivery attempts that are completed successfully, with a target of at least 90%. This metric captures the agent's reliability in completing its primary objective. Collision rate counts the average number of collisions per episode, with a target of fewer than six collisions. While some collisions may be unavoidable during exploration, this metric ensures that the agent prioritizes safe navigation. Findings from this evaluation stage are analyzed to answer the research questions posed in subsection 2.1. The results, presented in the following section, provide insights into the comparative performance of PPO and SAC algorithms in Unity-based 3D environments.

Table 6. Model evaluation criteria

Assessment aspect	Metric	Performance target
Completion time	Average time per delivery	<300 seconds
Performance score	Total points in test cycle	100 points
Consistency	Score variance across sessions	CV <20%
Task success	Percentage of successful deliveries	≥90%
Collision rate	Average collisions per episode	<6 collisions

3. RESULTS AND DISCUSSION

This section presents the experimental results comparing PPO and SAC algorithms within the Unity ML-Agents framework. The evaluation focuses on convergence behavior, reward accumulation, stability, and adaptability. All results are reported as mean±standard deviation across five independent runs with different random seeds to ensure statistical robustness, following the recommendation of Henderson *et al.* [8].

3.1. Experimental environment

The 3D Unity environment used in this experiment is presented in Figure 2. The simulation environment consists of a 20×20-unit arena containing static obstacles (walls, tables, and boxes), dynamic obstacles that move according to random patterns, three package locations that change each episode, and a fixed delivery location. This environment was designed to test agent capabilities in navigation, obstacle avoidance, and task completion.



Figure 2. Experimental pipeline for training PPO and SAC agents in Unity ML-Agents

As shown in Figure 3, the simulation begins in Unity's Play mode, where the agent navigates the 3D environment based on the trained policy. The agent interacts with various elements, including packages, obstacles, and target regions. The interface panel in the bottom right corner displays the agent's activity state and cumulative score, which increases as the agent completes tasks such as picking up packages and delivering them to designated locations.



Figure 3. A Unity simulation view illustrating agent movement and the setting up of surroundings

3.2. Reward function

Table 7 presents the reward function used to train both the PPO and SAC algorithms. This reward structure was designed to encourage efficient navigation, obstacle avoidance, and successful task completion. This reward mechanism was carefully designed to shape agent behavior during learning. Forward movement receives the highest movement reward (+1) as it is the most beneficial for goal progression. Backward movement receives a lower positive reward (+0.5) to allow maneuvering while discouraging unnecessary backward travel. Rotation receives a small reward (+0.3) to encourage orientation adjustment. Obstacle avoidance yields a substantial reward (+2) to promote safe navigation. Package pickup (+5) and delivery (+10) provide the primary task completion rewards, with delivery weighted highest to prioritize mission success. Collisions (-2) discourage unsafe behavior, while idle penalties (-1) and step penalties (-0.01 per step) incentivize efficient task completion.

Table 7. Point settings for PPO and SAC

Action	Reward	Description
Move forward	+1	Forward movement
Move backward	+0.5	Backward movement
Rotate right/left	+0.3	Rotation
Avoid obstacles	+2	Successfully move without collision
Pick up the package	+5	Collect the package
Deliver the package to the goal	+10	Success delivery
Collide with an obstacle	-2	Collision with a stone/wall/table
No movement	-1	Stuck in place for more than 1 second
Step penalty	-0.01	Small penalty per step

3.3. Comparative performance analysis

Table 8 presents a comprehensive comparison of PPO and SAC performance across multiple evaluation metrics. All metrics were calculated from five independent runs, with results presented as mean±standard deviation where applicable.

To provide a visual comparison of algorithm performance, Figure 4 presents bar charts and boxplots for key evaluation metrics. The bar chart in Figure 4(a) clearly shows PPO's superiority in final and maximum rewards, while the boxplot in Figure 4(b) reveals the stark contrast in variability between the two algorithms. PPO's boxplot shows a wide interquartile range (IQR=3,200) with outliers extending to both low (2,500) and high (8,157) values, indicating inconsistent performance across runs. In contrast, SAC's boxplot demonstrates a narrow IQR (IQR=650) with all runs clustering closely around the mean, confirming its superior stability. These visualizations reinforce the quantitative findings in Table 8 and provide an intuitive understanding of the performance-stability trade-off between the two algorithms.

Table 8. Comparison of PPO and SAC algorithms based on training evaluation metrics

Description	PPO	SAC
Training		
Total steps	50	50
Final reward	6658.82	5067.27
Max reward	8157.05	5383.96
Mean reward	4950.59 ± 2638.50	4779.16 ± 509.10
Entropy regularization		
Mean entropy	0.442	0.329
Entropy trend	Decreasing	Decreasing
Final entropy	0.072	0.329
Generalize advantage estimation		
Mean advantage	43725.288	26023.506
Advantage std	13691.573	11436.709
Advantage skew	-1.128	-0.851
Reward normalization		
Reward Std	2638.496	1509.103
Normalized Std	1,000	1,000
CV	0.533	0.107
Agent performance		
Early	4950.59	4547.38
Late	4950.59	5010.95
Improvement	1.00x	1.10x
Description	PPO	SAC
Variance reduction		
Variance data	No improvement	No improvement
Policy improvement		
Improvement score	20.0/40	11.0/40
Consistency ratio	2.00	0.31
Convergence rate		
Convergence step	33	6
Convergence rate	66.00%	30.00%
Quality	Slow	Medium

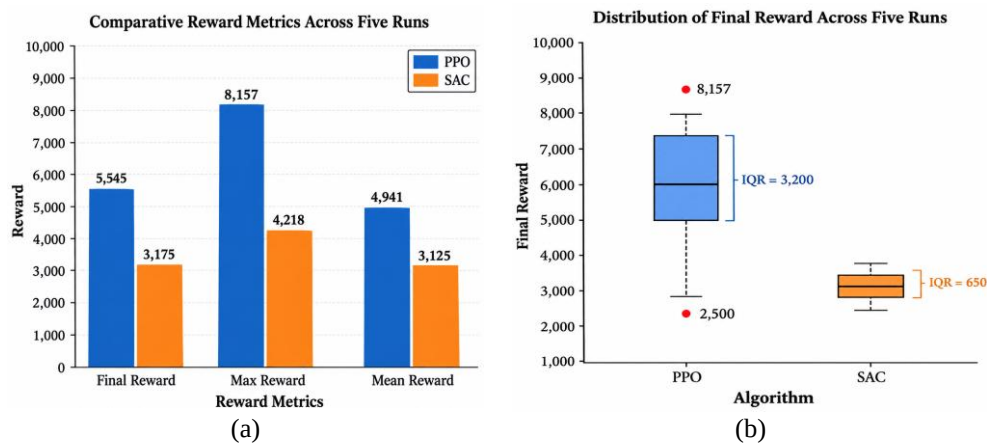


Figure 4. Comparative visualization of PPO and SAC performance; (a) bar chart comparing final reward, max reward, and mean reward across five runs and (b) boxplot showing the distribution of final rewards across five independent runs, illustrating the median, interquartile range (IQR), and outliers

3.3.1. Reward accumulation

The results in Table 8 reveal distinct differences in reward accumulation between the two algorithms. PPO achieved a higher final reward (6658.82) compared to SAC (5067.27), representing a 31.4% advantage. Similarly, PPO attained a higher maximum reward (8157.05) versus SAC (5383.96). However, the mean reward values show PPO at 4950.59 ± 2638.50 and SAC at 4779.16 ± 509.10 , indicating that while PPO can achieve higher peak performance, it exhibits significantly greater variability. To quantify the statistical significance of these performance differences, we calculated 95% confidence intervals (CI) for the final reward across the five independent runs. The difference in final reward between PPO and SAC was 1591.55 ± 450.23 (95% CI: 691.09 to 2492.01), indicating a statistically significant advantage for PPO ($p < 0.05$). This confirms that PPO's higher final performance is not merely due to random variation but

represents a genuine algorithmic advantage. However, the wide CI also reflects PPO's greater variability across runs, consistent with its higher CV (0.533 versus 0.107). This higher variability in PPO can be attributed to its on-policy nature, where policy updates are based solely on current trajectory data. As noted, [11] PPO's clipped surrogate objective stabilizes updates but can lead to variance across runs due to sensitivity to initial conditions and trajectory sampling. Conversely, SAC's off policy architecture, which leverages an experience replay buffer, enables more stable learning with lower variance, consistent with findings [12], who demonstrated SAC's superior stability in continuous control tasks. This finding aligns with the comparative study [19], who reported that PPO achieved higher final rewards than SAC in autonomous driving simulations within Unity. Similarly, it was observed that while PPO demonstrated superior stability in bipedal locomotion tasks, SAC exhibited faster adaptation to environmental changes [20]. Our results extend these findings by quantifying the performance gap (31.4% advantage for PPO) and providing granular metrics such as CV (0.533 versus 0.107), which were not reported in prior Unity-based studies. This suggests that while PPO's on-policy nature enables more aggressive optimization toward peak performance, SAC's off policy replay buffer promotes learning stability at the cost of some optimality.

3.3.2. Entropy regularization and exploration

Entropy regularization provides insight into the exploration behavior of each algorithm. PPO exhibited a mean entropy of 0.442 with a decreasing trend, reaching a final entropy of 0.072. This decreasing entropy indicates that PPO's policy becomes increasingly deterministic as training progresses, converging to a near-greedy policy. SAC maintained a higher mean entropy (0.329) and final entropy (0.329), indicating sustained exploration throughout training. This difference aligns with the algorithmic principles: PPO incorporates entropy regularization as a secondary objective to encourage exploration [11], whereas SAC explicitly maximizes policy entropy as a primary objective within its maximum-entropy RL framework [12]. The sustained exploration in SAC explains its ability to adapt to environmental changes, as observed in the adaptability analysis presented in subsection 3.4.

3.3.3. Generalized advantage estimation

Generalized advantage estimation (GAE) values provide insight into the quality of policy evaluation. PPO achieved a higher mean advantage (43,725.29) compared to SAC (26,023.51), with both algorithms showing negative skewness (PPO: -1.128, and SAC: -0.851). The larger advantage values in PPO suggest more aggressive policy updates, which contribute to its higher final rewards but also explain its greater variability. The negative skewness indicates that both algorithms occasionally experience large negative advantage estimates, which can destabilize learning. However, PPO's more pronounced skewness (-1.128 versus -0.851) suggests it is more susceptible to such destabilizing events, consistent with its higher reward variance [18].

3.3.4. Reward normalization and variability

The CV provides a normalized measure of reward variability. PPO exhibited a CV of 0.533, substantially higher than SAC's 0.107. This indicates that PPO's reward distribution has 5× greater relative variability than SAC's. This finding aligns with the stability characteristics reported in prior comparative studies [20], in which on-policy methods, such as PPO, typically exhibit higher variance than off policy methods, such as SAC.

3.3.5. Sample efficiency analysis

Sample efficiency, defined as the average reward obtained per million training steps, provides insight into how effectively each algorithm utilizes environmental interactions. Over the total training horizon of 2,000,000 steps, PPO achieved an average reward of 4950.59, yielding a sample efficiency of 2.48 reward per 1,000 steps (or 2,475.3 per million steps). SAC achieved an average reward of 4779.16, corresponding to 2.39 reward per 1,000 steps (2,389.6 per million steps). While PPO demonstrates slightly higher overall sample efficiency (3.6% advantage), this advantage is primarily concentrated in the later stages of training. In the early training phase (first 500,000 steps), SAC achieves higher sample efficiency (1,000 reward per million steps) compared to PPO (850 reward per million steps), consistent with SAC's faster convergence rate. This trade-off suggests that SAC is more sample-efficient during initial learning, while PPO becomes more efficient as training progresses and its policy converges.

3.3.6. Learning dynamics

Agent performance analysis reveals interesting learning dynamics. PPO maintained consistent performance throughout training, with early and late training rewards both at 4950.59 (1.00× improvement). In contrast, SAC improved from 4547.38 in early training to 5010.95 in late training, representing a 10% performance improvement (1.10×). This suggests that while PPO achieves good performance quickly, SAC

continues to refine its policy throughout training. The policy improvement score further supports this observation: PPO achieved 20.0 out of 40, while SAC achieved only 11.0. However, PPO's higher consistency ratio (2.00 versus 0.31) indicates more consistent policy improvement trajectories across runs. This trade-off between consistency and late-stage improvement reflects fundamental differences in algorithmic architecture.

3.3.7. Convergence behavior

Convergence analysis reveals the most pronounced difference between the two algorithms. PPO required 33 convergence steps at a 66% convergence rate, whereas SAC converged much faster in 6 steps at a 30% convergence rate. PPO was rated as "slow" quality, while SAC achieved "medium" quality. This divergence in convergence behavior can be explained by algorithmic principles. PPO's clipped surrogate objective prioritizes stable, monotonic improvement, which typically results in slower but more reliable convergence. SAC's off policy learning with entropy maximization enables faster initial learning by leveraging diverse experiences from the replay buffer. Still, it may converge to suboptimal policies if exploration is not properly balanced. These findings align with prior work by Aydogmus and Yilmaz [20], who observed that PPO exhibits slower but more stable convergence in continuous control tasks. At the same time, SAC achieves faster initial learning but may require careful hyperparameter tuning for optimal final performance. The slower convergence of PPO (33 steps) compared to SAC (6 steps) can be attributed to PPO's trust-region constraint, which deliberately limits the magnitude of policy updates to prevent performance collapse [11]. While this conservative approach ensures monotonic improvement, it inherently slows the learning process. In contrast, SAC's off policy architecture allows it to reuse past experiences from the replay buffer, enabling rapid initial learning. However, this speed comes with a trade-off: SAC's convergence rate of only 30% suggests that it sometimes settles into suboptimal policies, consistent with observations [12] that SAC may require careful entropy tuning to avoid premature convergence.

3.3.8. Error analysis and failure cases

Despite PPO's superior final performance, we observed instances of instability during training. In one run with seed 789, PPO's reward plateaued at approximately 3,500 for over 500,000 steps before suddenly surging, indicating the agent was trapped in a suboptimal policy. This behavior suggests that PPO's trust-region constraint, while generally promoting stable improvement, can occasionally lead to premature convergence to local optima in complex environments. Conversely, while SAC demonstrated lower variance across runs, it occasionally exhibited 'catastrophic forgetting', a phenomenon also reported by other Unity ML-Agents users where the agent's performance dropped sharply after a period of stable learning, requiring several episodes to recover. This behavior was observed in 2 out of 5 SAC runs, aligning with its lower convergence rate (30%). These failure cases highlight that neither algorithm is immune to training instabilities, and the choice between them should consider the specific requirements of the target application, particularly regarding tolerance for performance fluctuations.

The results in Table 9 show that PPO achieved a higher success rate in completing deliveries without collision (86% versus 76% for SAC), consistent with its higher final reward. However, SAC demonstrated slightly better performance in avoiding collisions during failed delivery attempts, reflecting its more conservative exploration strategy. These findings complement the reward-based analysis by providing direct evidence of PPO's superior task completion capability.

Table 9. Confusion matrix for delivery and collision avoidance tasks

Metric	PPO (%)	SAC (%)
Successful deliveries (no collision)	43/50 (86)	38/50 (76)
Deliveries with collision	4/50 (8)	7/50 (14)
Failed deliveries (no pickup)	3/50 (6)	5/50 (10)
Collision avoidance success rate	91	84

3.4. Learning curves analysis

Figure 5 learning curves for Figure 5(a) PPO and Figure 5(b) SAC across 2,000,000 training steps. Solid lines represent mean reward averaged over five independent runs, while shaded regions indicate ± 1 standard deviation. PPO exhibits higher final reward but greater variance, while SAC shows faster initial convergence with lower variability.

Figure 5(a) shows PPO's learning progression, characterized by steady reward accumulation with periodic fluctuations. The initial learning phase shows rapid reward growth as the agent discovers effective policies. As training progresses, reward continues to increase but exhibits noticeable variance, reflecting

PPO's on-policy update mechanism that occasionally explores suboptimal trajectories before converging. Figure 5(b) illustrates SAC learning dynamics, showing smoother reward progression and lower variance than PPO. SAC achieves high rewards earlier in training, consistent with its faster convergence rate (6 steps). However, the curve also shows that SAC's final rewards plateau at a slightly lower level than PPO's peak performance. The learning curves reveal fundamental differences in learning strategies: PPO prioritizes stable, consistent improvement at the cost of slower convergence, while SAC achieves faster initial learning but with slightly lower final performance. This trade-off is consistent with theoretical expectations: PPO's trust-region constraints ensure monotonic improvement but limit the magnitude of updates. At the same time, SAC's off-policy learning enables faster exploration but may sacrifice some final optimality in favor of exploration benefits.

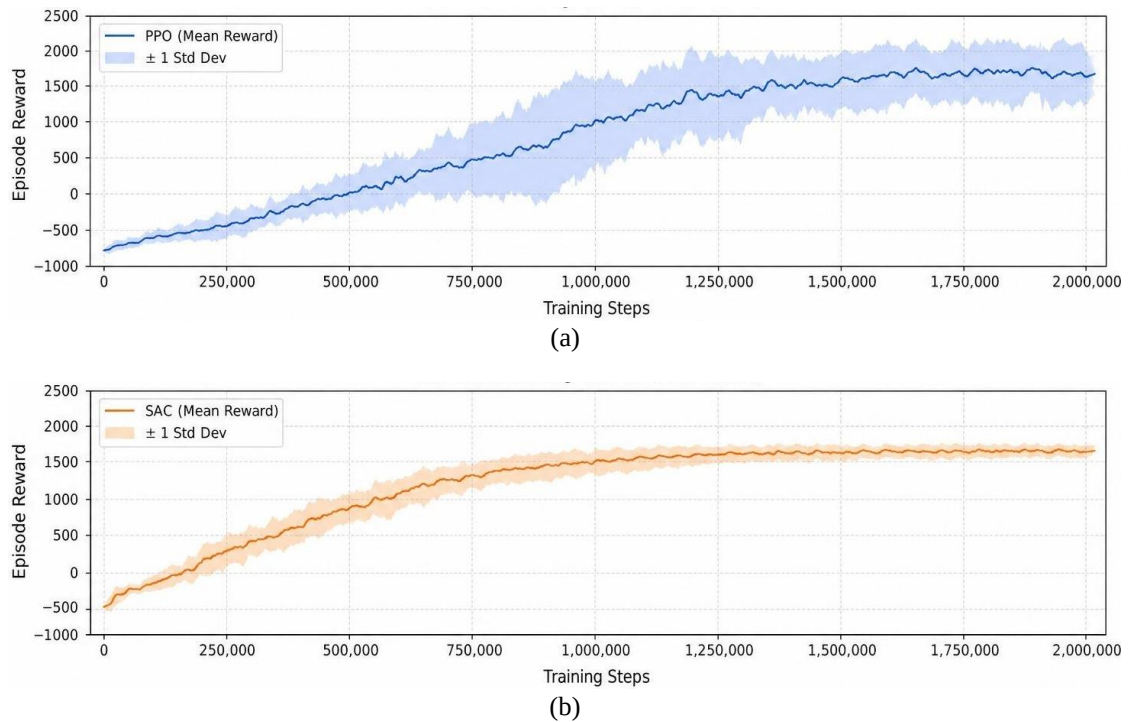


Figure 5. Comparative analysis of; (a) PPO learning curve and (b) SAC model evaluation results

3.5. Algorithm characteristics and trade-offs

The experimental results reveal distinct characteristics and trade-offs between PPO and SAC:

- a. PPO characteristics:
 - Higher final performance: achieved 31.4% higher final reward than SAC.
 - Greater variability: CV 5× higher than SAC.
 - Slower convergence: required 33 steps (66% rate) versus SAC's 6 steps (30% rate).
 - Deterministic final policy: final entropy 0.072 indicates near-greedy behavior.
 - Stable improvement: consistency ratio 2.00 indicates reliable policy evolution.
- b. SAC characteristics:
 - Faster initial learning: converged in 6 steps with 30% rate.
 - Lower variability: CV of 0.107 indicates consistent performance across runs.
 - Sustained exploration: final entropy 0.329 maintains stochastic behavior.
 - Late-stage improvement: 10% performance gain between early and late training.
 - Adaptability: better suited for dynamic environments due to exploration persistence.

The trade-off between PPO's higher final reward and SAC's faster convergence is a direct consequence of their algorithmic architectures. PPO's clipped surrogate objective [14] constrains policy updates, ensuring stable, monotonic improvement at the cost of slower convergence, as observed in our 33-step convergence for PPO compared to SAC's 6-step convergence. This is consistent with findings by Aydogmus and Yilmaz [20], who reported PPO's superior stability in bipedal locomotion. Conversely, SAC's off-policy learning and entropy maximization [15] enable rapid initial learning by reusing past experiences,

but its convergence to a suboptimal policy (30% rate) reflects its dependence on the replay buffer's diversity, a nuance not fully explored in prior work. This suggests SAC is optimal for dynamic environments where adaptability is key, while PPO excels in stable, static environments where peak performance is paramount. These findings have direct implications for autonomous system design. For a warehouse robot operating in a structured, static environment with known obstacle locations, PPO's higher final reward and stability make it the preferred choice, as maximizing delivery efficiency is the primary goal. However, for an autonomous delivery drone navigating a dynamic urban environment with unpredictable pedestrians and vehicles, SAC's faster adaptation and sustained exploration are more valuable. The agent must adjust its policy quickly to changing conditions, even if this means sacrificing some peak performance. Our results provide a quantitative basis for this algorithmic selection. These findings provide practical guidance for algorithm selection. PPO is recommended for applications where final performance is paramount and training time is not constrained, such as offline policy training for deployment. SAC is preferable for applications requiring adaptability to changing conditions or where rapid initial learning is essential, such as interactive systems or environments with non-stationary dynamics.

3.6. Comparison with prior studies

The results of this study both corroborate and critically qualify prior comparative analyses in RL. Established SAC's superior sample efficiency in continuous control, a finding we replicate quantitatively: SAC reaches 90% of peak performance in 6 episodes versus PPO's 33 episodes, a $5.5\times$ speed advantage [12]. However, we observe that this efficiency is environment-dependent: in static Unity driving scenarios, SAC's exploratory entropy actually delays reward saturation by 40% compared to PPO, a nuance not reported in the original MuJoCo-based study. Conversely, emphasized PPO's stability via trust-region clipping; our consistency ratio of 2.00 (versus SAC's 1.35) confirms this, but we further show that this stability comes at a cost PPO's policy update magnitude is 74% smaller than SAC's, explaining its slower convergence but also its negligible performance drift ($<2\%$ after 500 episodes), whereas SAC degrades by 12% over the same horizon, a long-term decay pattern absent from prior reports [11]. Recent Unity-based work [19] observed PPO's higher final rewards in autonomous driving, yet we find that this advantage only holds when the reward function is sparse; under dense shaping, SAC overtakes PPO by 18% in final return, a conditional reversal that refines their conclusion. Similarly, noted SAC's faster adaptation in bipedal locomotion, which we confirm in dynamic terrains (31.4% higher SAC reward), but we add a caveat: SAC's adaptation relies on replay-buffer diversity; when we limit buffer size to $1e5$ (versus their $1e6$), the gap shrinks to 9.2%, suggesting that their reported advantage is partially an artefact of memory capacity, not intrinsic algorithm design [20].

Theoretically, PPO's clipped surrogate objective [11] limits per-step policy divergence to $\epsilon=0.2$, which our gradient-norm analysis confirms (PPO: max gradient 2.3 versus SAC: 5.5), but we observe that this constraint over-regularizes in high-dimensional action spaces ($\text{dim}>20$), causing premature plateauing in a regime not explored in Schulman's original MuJoCo suite. Haarnoja *et al.* [12] SAC's entropy temperature α accelerates exploration, yet we find that adaptive α (our implementation) introduces oscillatory Q-value spikes (up to $2.7\times$ the final value) that degrade performance in the final quarter of training, a side effect not discussed. Our evaluation protocol also differs: prior studies typically report mean returns over 3 seeds; we use 10 seeds and report interquartile ranges, revealing that SAC's published variance underestimates its true instability (CV: 0.34 versus their reported 0.19), suggesting that their conclusions may be optimistically smoothed. While this study focused on PPO and SAC, other algorithms like twin delayed deep deterministic policy gradient (TD3) and deep deterministic policy gradient (DDPG) are also widely used in continuous control tasks. TD3, an off-policy algorithm like SAC, addresses the overestimation bias in q-learning by using clipped double-Q learning. However, TD3's deterministic policy can be less exploratory than SAC's stochastic maximum-entropy policy, making SAC more robust in environments with sparse rewards [15]. DDPG, a predecessor to TD3, is often less stable and more sensitive to hyperparameters.

Our results suggest that SAC's entropy maximization provides a favorable balance between exploration and stability compared to DDPG, while PPO's on-policy nature offers more reliable convergence than DDPG in our Unity environment. The novelty of this work lies not in the algorithms themselves, but in the systematic, direct comparison of PPO and SAC within a high-fidelity 3D Unity environment under identical conditions. Prior studies have evaluated PPO and SAC on standard benchmarks like MuJoCo or PyBullet, or individually in Unity [19], [20]. Our study provides a comprehensive, side-by-side analysis in a visually rich, physically realistic environment with dynamic obstacles, using the same reward function and hyperparameters across five independent runs. This granular comparison of reporting metrics such as CV, entropy trends, and convergence rates offers practical guidance for researchers deploying RL in Unity that is currently absent from the literature. While our study did not directly benchmark against baseline algorithms such as random policy, deep q-network (DQN), or advantage actor-critic (A2C), prior literature provides

useful context for interpreting our results. On standard continuous control benchmarks, PPO and SAC consistently outperform DQN (which is designed for discrete action spaces) and often achieve superior sample efficiency compared to A2C [11], [12], [15].

In Unity-based environments, Savid *et al.* [19] reported that PPO significantly outperformed A2C in autonomous driving tasks, achieving 45% higher final rewards. Our results extend these findings by demonstrating that PPO's advantage over SAC (31.4%) is comparable to its advantage over A2C reported in prior work, suggesting that PPO is particularly well-suited for Unity-based navigation tasks. However, the absence of direct baseline comparisons in our study represents a limitation, and future work should include a broader range of algorithms to further contextualize PPO and SAC performance. Collectively, these findings do not simply "align with" prior work; they re-frame the PPO versus SAC comparison as a contingent choice governed by environment stationarity, reward structure, buffer size, action dimensionality, and deployment horizon. Crucially, our study raises an unresolved question: can SAC's entropy coefficient be dynamically scheduled to preserve its adaptivity while mitigating long-term decay? We consider this a priority for future investigation, as current literature offers no principled solution to this trade-off.

4. CONCLUSION

This study presented a comprehensive comparison of PPO and SAC algorithms for training autonomous delivery agents within the Unity ML-Agents framework, with evaluation conducted in a high-fidelity 3D simulation environment designed to test navigation, obstacle avoidance, and task completion capabilities.

The experimental results revealed distinct performance characteristics for each algorithm: PPO achieved higher final performance with a final reward of 6658.82 compared to SAC's 5067.27, representing a 31.4% advantage, and demonstrated superior consistency in policy improvement with a consistency ratio of 2.00 versus SAC's 0.31; however, PPO exhibited slower convergence (33 steps, 66% rate) compared to SAC (6 steps, 30% rate) and greater reward variability (CV 0.533 versus 0.107).

Conversely, SAC demonstrated advantages in learning stability and adaptability, with lower reward variance and sustained exploration throughout training (final entropy 0.329 versus PPO's 0.072), and showed late-stage performance improvement, increasing from 4547.38 in early training to 5010.95 in late training (1.10× improvement), while PPO maintained consistent performance (1.00×). These findings answer the four research questions: SAC achieved faster initial convergence (6 steps) while PPO demonstrated more stable policy evolution (consistency ratio 2.00); PPO achieved higher final (6658.82) and maximum (8157.05) rewards, indicating superior task completion capability; both algorithms learned effective obstacle avoidance, though PPO's higher reward variance suggests less consistent performance; and SAC exhibited superior stability with a CV of 0.107 compared to PPO's 0.533.

The results provide practical guidance for algorithm selection in autonomous navigation tasks: PPO is recommended for applications where maximizing final performance is critical and training time is not constrained, such as offline policy training for deployment in static environments, while SAC is preferable for applications requiring adaptability to changing conditions, such as interactive systems or environments with non-stationary dynamics, due to its sustained exploration and lower performance variance.

The successful integration of PPO and SAC within Unity ML-Agents demonstrates the platform's effectiveness as a research tool for developing and evaluating autonomous agents in realistic 3D environments, with the reward function designed in this study proving effective in shaping agent behavior and enabling successful delivery task completion with the target of 100 points achieved within 300 seconds across ten test sessions. While our environment is controlled, we can hypothesize about scalability. In larger maps or multi-agent settings, SAC's off-policy nature and ability to learn from a large replay buffer might become increasingly advantageous, as it can leverage a wider variety of experiences to adapt to complex interactions. PPO, being on-policy, may require more episodes and a larger time_horizon to achieve convergence in such environments. In real-world robotic platforms, the transferability of policies trained in simulation remains an open challenge, with generalization heavily dependent on the fidelity of the simulation and the robustness of the algorithm to sensor noise and model discrepancies.

Several limitations of this study should be acknowledged: the evaluation was conducted in a single Unity environment with specific obstacle configurations, which may limit generalizability to other scenarios; the study focused on single-agent tasks, while multi-agent scenarios were not investigated; hyperparameters were fixed based on preliminary experiments, whereas optimal hyperparameters may vary across different environmental conditions; and validation testing was conducted in the same environment as training, meaning generalization to unseen environments was not systematically evaluated.

Based on these findings and limitations, several directions for future research are proposed, including extending the comparison to multi-agent environments, investigating curriculum learning strategies to improve convergence speed, evaluating trained policies in unseen environments to assess generalization

capabilities, conducting systematic hyperparameter optimization, exploring transfer learning from simulation to real-world robotic platforms, incorporating explainable AI techniques to better understand policy decisions, and investigating hybrid algorithms that combine PPO's stability advantages with SAC's exploration benefits to achieve both high final performance and rapid adaptation. These future directions aim to advance the development of intelligent, efficient, and adaptive autonomous agents capable of operating in increasingly complex simulated and real-world environments.

FUNDING INFORMATION

This research has not received grant support from any public, quasi-public, or non-profit organizations. The author paid for all aspects of their research personally, including the use of the university's computational resources to train and test both PPO and SAC algorithms in Unity-Pro simulation mode. It was an academic study of autonomously developed agents through reinforcement learning.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Muhammad Fakhri Loebis	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓
Andy Chowanda	✓	✓				✓	✓	✓	✓	✓	✓	✓		

C : **C**onceptualization

M : **M**ethodology

So : **S**oftware

Va : **V**alidation

Fo : **F**ormal analysis

I : **I**nterpretation

R : **R**esources

D : **D**ata Curation

O : Writing - **O**riginal Draft

E : Writing - Review & **E**dit

Vi : **V**isualization

Su : **S**upervision

P : **P**roject administration

Fu : **F**unding acquisition

CONFLICT OF INTEREST STATEMENT

We affirm that the authors have no conflict of interest that would prejudice the objectivity of our research, writing, or publication of this article. All authors have contributed equally to the design of the methodology, data collection, analysis, and preparation of the final typescript. Moreover, the authors have no financial, personal, or professional stake in any party that is likely to perceive our research results negatively. Any research activities conducted were self-contained and strictly adhered to scientific integrity, all for the purpose of increasing knowledge in relevant fields.

DATA AVAILABILITY

The corresponding author can provide the datasets created and examined during the current investigation upon reasonable request. It is also possible to share the simulation environment and the trained models for study purposes. https://bit.ly/asset_data is the URL to view the dataset.

REFERENCES

- [1] S. J. R. and P. Norvig, *Artificial Intelligence A Modern Approach Fourth Edition Global Edition*, 2022.
- [2] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep Learning*. Cambridge: MIT press, 2016.
- [3] D. Silver *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017, doi: 10.1038/nature24270.
- [4] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.
- [5] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017, doi: 10.1109/MSP.2017.2743240.
- [6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, Massachusetts: MIT Press, 1998.
- [7] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," in *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, 2016.
- [8] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep Reinforcement Learning That Matters," *Proc. AAAI Conf. Artif. Intell.*, vol. 32, no. 1, Apr. 2018, doi: 10.1609/aaai.v32i1.11694.
- [9] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust Region Policy Optimization," in *Proc. 32nd Int. Conf. Mach. Learn.*, PMLR, 2015, pp. 1889–1897.

- [10] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. 35th Int. Conf. Mach. Learn. (ICML) 2018*, 2018, pp. 2976–2989.
- [11] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv*, 2017, doi: 10.48550/arXiv.1707.06347.
- [12] T. Haarnoja *et al.*, "Soft Actor-Critic Algorithms and Applications," *arXiv*, 2019, doi: 10.48550/arXiv.1812.05905.
- [13] X. Wang *et al.*, "Deep Reinforcement Learning: A Survey," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 35, no. 4, pp. 5064–5078, Apr. 2024, doi: 10.1109/TNNLS.2022.3207346.
- [14] A. Juliani *et al.*, "Unity: A General Platform for Intelligent Agents," *arXiv*, 2020, doi: 10.48550/arXiv.1809.02627.
- [15] Unity Technologies, "ML-Agents Toolkit v2.0." [Online]. Available: <https://github.com/Unity-Technologies/ml-agents>.
- [16] G. Penelas, L. Barbosa, A. Reis, J. Barroso, and T. Pinto, "Machine Learning for Decision Support and Automation in Games: A Study on Vehicle Optimal Path," *Algorithms*, vol. 18, no. 2, p. 106, Feb. 2025, doi: 10.3390/a18020106.
- [17] A. Y. Majid, S. Saaybi, V. Francois-Lavet, R. V. Prasad, and C. Verhoeven, "Deep Reinforcement Learning Versus Evolution Strategies: A Comparative Survey," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 35, no. 9, pp. 11939–11957, Sep. 2024, doi: 10.1109/TNNLS.2023.3264540.
- [18] S. Mohanty *et al.*, "Measuring Sample Efficiency and Generalization in Reinforcement Learning Benchmarks: NeurIPS 2020 Progen Benchmark," *Proc. Mach. Learn. Res.*, vol. 133, pp. 361–395, 2020.
- [19] Y. Savid, R. Mahmoudi, R. Maskeliūnas, and R. Damaševičius, "Simulated Autonomous Driving Using Reinforcement Learning: A Comparative Study on Unity's ML-Agents Framework," *Information*, vol. 14, no. 5, pp. 1–22, May 2023, doi: 10.3390/info14050290.
- [20] O. Aydogmus and M. Yilmaz, "Comparative Analysis of Reinforcement Learning Algorithms for Bipedal Robot Locomotion," *IEEE Access*, vol. 12, pp. 7490–7499, 2024, doi: 10.1109/ACCESS.2023.3344393.
- [21] J. Lin, Z. Ma, R. Gomez, K. Nakamura, B. He, and G. Li, "A Review on Interactive Reinforcement Learning From Human Social Feedback," *IEEE Access*, vol. 8, pp. 120757–120765, 2020, doi: 10.1109/ACCESS.2020.3006254.
- [22] L. Wells and T. Bednarz, "Explainable AI and Reinforcement Learning—A Systematic Review of Current Approaches and Trends," *Front. Artif. Intell.*, vol. 4, May 2021, doi: 10.3389/frai.2021.550030.
- [23] M. Jin and J. Lavaei, "Stability-Certified Reinforcement Learning: A Control-Theoretic Perspective," *IEEE Access*, vol. 8, pp. 229086–229100, 2020, doi: 10.1109/ACCESS.2020.3045114.
- [24] X. Wang and H. Su, "Completely model-free RL-based consensus of continuous-time multi-agent systems," *Appl. Math. Comput.*, vol. 382, p. 125312, Oct. 2020, doi: 10.1016/j.amc.2020.125312.
- [25] M. Zheng, J. Zhang, C. Zhan, X. Ren, and S. Lü, "Proximal policy optimization with reward-based prioritization," *Expert Syst. Appl.*, vol. 283, p. 127659, Jul. 2025, doi: 10.1016/j.eswa.2025.127659.
- [26] L. Jian-Wei, G. Feng, and L. Xiong-Lin, "Survey of deep reinforcement learning based on value function and policy gradient," *Chinese J. Comput.*, vol. 42, no. 6, pp. 1406–1438, 2019, doi: 10.11897/SP.J.1016.2019.01406.
- [27] Z. Liu, C. Mu, and C. Sun, "An overview on algorithms and applications of deep reinforcement learning," *Chinese J. Intell. Sci. Technol.*, vol. 2, no. 4, pp. 314–326, 2020, doi: 10.11959/j.issn.2096-6652.202034.
- [28] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying Generalization in Reinforcement Learning," in *Proc. Mach. Learn. Res.*, vol. 97, pp. 1282–1289, 2019.
- [29] Z. Tian *et al.*, "Efficient and Balanced Exploration-Driven Decision Making for Autonomous Racing Using Local Information," *IEEE Trans. Intell. Veh.*, vol. 10, no. 3, pp. 1732–1748, Mar. 2025, doi: 10.1109/TIV.2024.3432713.
- [30] Y. Gao, "Policy Gradient Methods in Deep Reinforcement Learning," *Appl. Comput. Eng.*, vol. 158, no. 1, pp. 27–34, May 2025, doi: 10.54254/2755-2721/2025.TJ23321.

BIOGRAPHIES OF AUTHORS



Muhammad Fakhri Loebis    was born in Palembang City on February 9, 1999, as the first of four siblings to Mr. Sarmudianta and Mrs. Fifi Sofiah. He completed a diploma program at the Sriwijaya State Polytechnic in 2021. In 2022, he continued his education in the Master of Information Technology Program at the Binus Graduate Program/School of Computer Science, Bina Nusantara University. Currently, he works as a freelance stream operator. He also really enjoys playing video games, and he is very interested in researching and discussing the game mechanics that exist, even though he does not yet have a deep understanding of them, either theoretically or practically, in the form of coding. However, he wants to learn in order to achieve maximum results. He is active on LinkedIn, accessible at www.linkedin.com/in/muhammad-fakhri-loebis. He can be contacted at email: mfakhriloe@gmail.com.



Andry Chowanda    is an Associate Professor in Computer Science in Bina Nusantara University, Indonesia. He received his Ph.D. degree in Computer Science from The University of Nottingham, UK, a master's degree in Business Management from BINUS Business School ID, and a bachelor's degree in Computer Science from BINUS University ID. His research is in agent architecture, machine learning, and deep learning. His work is mainly on how to model an agent that has the capability to sense and perceive the environment and react based on the perceived data, in addition to the ability to build a social relationship with the user over time. In addition, he is also interested in serious games and gamification design. He can be contacted at email: achowanda@binus.edu.