Deep Reinforcement Learning: From Single-Agent to Multi-Agent

Yuyangsong Xie

College of Liberal Arts and Sciences, University of Illinois at Urbana-Champaign, Urbana, IL 61801, the United States

yx40@illinois.edu

Abstract. Deep Reinforcement Learning (DRL), a core branch of artificial intelligence, integrates the representational capabilities of deep learning with the decision optimization mechanisms of reinforcement learning, enabling breakthroughs in complex tasks ranging from virtual environments to real-world scenarios. This paper systematically organizes the core algorithmic system of deep reinforcement learning, classifying it into four classic frameworks: Value-Based, Policy-Based, Actor-Critic, and Multi-Agent Reinforcement Learning (MARL). It analyzes the core ideas, improvement logics, and applicable scenarios of each algorithm. Meanwhile, focusing on the latest research findings in 2025, this paper explores the breakthroughs of reinforcement learning in directions such as real-world physics alignment, self-play-driven robustness, and autonomous driving, revealing the technical path of algorithms evolving from "virtual optimization" to "real-world reliability". It also discusses some remaining challenges like sample inefficiency and safety issues, which points out key directions to further enhance DRL's real-world applicability. This review provides a systematic reference for related research.

Keywords: Reinforcement Learning; Actor-Critic; Multi-Agent; Sim-to-Real; Self-Play.

1. Introduction

Reinforcement Learning (RL) stands apart from supervised and unsupervised learning by its focus on agent-environment interaction—a paradigm that mirrors how humans and animals learn through trial and error, feedback, and adaptation. Unlike supervised learning, which relies on labeled datasets to guide model training, or unsupervised learning, which uncovers hidden patterns in unlabeled data, RL enables agents to independently discover optimal behaviors by maximizing cumulative rewards over time. This unique characteristic makes RL particularly well-suited for dynamic, uncertain, and goal-oriented tasks, ranging from mastering video games to controlling robots, optimizing supply chains, and enabling autonomous driving. Over the past three decades, RL has evolved from theoretical frameworks to practical solutions, driven by algorithmic innovations, computational advances, and the growing demand for intelligent systems that can operate without pre-defined rules in complex environments.

The early development of RL was centered on single-agent scenarios, where an agent interacts with a static or semi-static environment to achieve a specific goal. Before the widespread adoption of Deep Reinforcement Learning (DRL), foundational policy-based RL algorithms laid the groundwork for directly optimizing decision-making policies. Williams proposed the Simple Statistical Gradient-Following Algorithm (often referred to as REINFORCE), which parameterizes the agent's policy and optimizes it through gradient ascent on the expected cumulative reward [1]. The theoretical framework of Policy-Based RL was further solidified by Sutton et al.'s landmark paper, which formally proved the policy gradient theorem for two core reward definitions (average reward and start-state discounted reward) and extended it to scenarios using function approximators [2].

Single-agent RL still faced the "curse of dimensionality" in state representation until the breakthrough of the Deep Q-Network (DQN) in 2013. Mnih et al. integrated convolutional neural networks (CNNs) with Q-learning, enabling the agent to process high-dimensional raw inputs (e.g., pixel data from Atari games) and learn value functions end-to-end [3]. Following DQN, to address its tendency to overestimate action values, Van Hasselt et al. proposed Double DQN in 2016, which

separates action selection and value estimation using distinct networks [4]. Concurrently, Wang et al. introduced the Dueling DQN, which optimized value function learning by decomposing the Q-function into state value (overall desirability of a state) and action advantage (how much an action outperforms others in that state) [5]. This architectural innovation enhanced efficiency in sparse-reward scenarios by allowing the agent to learn state-level value without relying on action selection.

The Actor-Critic framework—directly enabled by Sutton et al.'s theory—further advanced single-agent RL by combining the strengths of Value-Based and Policy-Based methods [2]. A series of influential algorithms have been built on this framework: Schulman et al.'s Proximal Policy Optimization (PPO) improved training stability with a "clipping" mechanism that limits policy divergence [6]; Lillicrap et al.'s Deep Deterministic Policy Gradient (DDPG) enabled deterministic control in continuous action spaces by inheriting DQN's stabilization mechanisms [7]; Haarnoja et al.'s Soft Actor-Critic (SAC) integrated maximum entropy to balance exploration and exploitation [8]; and Fujimoto et al.'s Twin Delayed DDPG (TD3) mitigated value overestimation with dual Critic networks [9]. Collectively, these algorithms expanded RL's applicability—but they remained limited to scenarios where a single decision-maker operates in an environment unaffected by other agents.

The shift toward Multi-Agent Reinforcement Learning (MARL) arose from the recognition that many real-world problems involve multiple interacting agents like traffic systems. Lowe et al.'s Multi-Agent Deep Deterministic Policy Gradient (MADDPG) pioneered the "Centralized Training with Decentralized Execution (CTDE)" paradigm, using a centralized Critic to solve credit assignment and decentralized Actors for real-world deployment [10]. Later, Yu et al.'s Multi-Agent Proximal Policy Optimization (MAPPO) demonstrated the surprising effectiveness of scaling PPO to MARL, achieving state-of-the-art results in cooperative games like StarCraft II via independent policy updates and shared experience replay [11].

Recent RL research has shifted focus from simulated performance to real-world reliability. He et al.'s ASAP algorithm addresses the "sim-to-real gap" by aligning simulation physics with real-world dynamics using multi-scale models and robot data [12]. Cusumano-Towner et al. and Cornelisse et al. leverage self-play to enhance robustness [13, 14]. Jaeger et al.'s CaRL simplifies large-scale planning with simple rewards, reducing deployment costs [15]. These works reconnect single-agent and multi-agent research, as real-world multi-agent systems require both robust individual capabilities and effective coordination.

This survey traces RL's evolution from single-agent to multi-agent innovations and points out the benefits and weaknesses of each algorithm or framework. Also, current research trends of RL and recent breakthroughs, such as sim-to-real transfer and self-play, are summarized, which guides future efforts to advance RL's practical impact.

2. Core Algorithmic Frameworks of DRL

The evolution of Deep Reinforcement Learning (DRL) has been driven by the continuous iteration of four core frameworks, each addressing distinct challenges in decision-making under uncertainty. This section systematically analyzes the design principles, key innovations, and application boundaries of Value-Based, Policy-Based, Actor-Critic, and Multi-Agent frameworks, with a focus on how each algorithm overcomes limitations of its predecessors.

2.1. Value-Based

Value-Based methods anchor their logic on estimating the value function—a measure of the expected cumulative reward an agent can obtain from a given state (or state-action pair) by following an optimal policy. The core goal is to learn a precise value function, which then implicitly defines the optimal policy (e.g., selecting the action with the highest estimated value). This framework gained widespread attention with the breakthrough of DQN [3], which solved the long-standing challenge of applying value-based methods to high-dimensional state spaces.

2.1.1 Deep Q-Network (DQN)

Prior to DQN, traditional Q-learning (a tabular value-based method) was limited to low-dimensional state spaces, as it required storing a discrete Q-table to map every (state, action) pair to its value. Mnih et al. [3] revolutionized this by replacing the Q-table with a CNN, enabling end-to-end learning from raw high-dimensional inputs (e.g., 84×84-pixel frames from Atari games). To stabilize training (a major issue with early neural network-based Q-learning), DQN introduced two key mechanisms:

- Experience Replay: The agent stores past (state, action, reward, next state) transitions in a replay buffer and samples mini-batches randomly during training. This breaks the correlation between consecutive training samples (a common source of training instability) and reuses data efficiently.
- Target Network: A separate "target network" is used to compute the target Q-value during updates. The calculation of Q-values consists of two core components: first, the online network output the *predicted* Q-values, representing the current network's immediate judgment on the long-term reward of a (state, action) pair; second, the *target* Q-values is calculated based on the Bellman equation, combining the target network's prediction of the optimal Q-value for the next state with the immediate reward, serving as the reference for training. This decouples the target value from the current network's parameters, preventing overestimation of action values caused by feedback loops.
- DQN demonstrated unprecedented performance by mastering 49 Atari 2600 games at or above human-level performance [3], proving that value-based methods could scale to complex, high-dimensional environments. However, it suffered from systematic overestimation of Q-values—a flaw addressed by subsequent improvements.

2.1.2 Double DQN and Dueling DQN

To mitigate DQN's overestimation bias, Van Hasselt et al. proposed *Double DQN*, which separates the two steps of DQN's target value calculation: action selection and value estimation [4]. In DQN, the same network is used to both select the "best" next action (maximizing Q-value) and estimate its value—this can inflate Q-values if the network overestimates some actions. Double DQN uses the *current network* to select the optimal next action but the *target network* to estimate its value, effectively reducing overestimation. Experiments on Atari games showed Double DQN reduced overestimation by up to 50% while improving performance on most games [4].

Wang et al. introduced *Dueling DQN* [5], an architectural innovation that decomposes the Q-function (state-action value) into two separate components:

- State Value: The expected cumulative reward of being in state s.
- Action Advantage: The relative benefit of taking action a in state s compared to other actions.

The Q-value is then computed as $Q(s,a) = V(s) + A(s,a)$ with a normalization term to ensure identifiability of $V(s)$ and $A(s,a)$. This decomposition allows the agent to learn state values independently of action selection—critical in sparse-reward scenarios (e.g., games where rewards are rare) or environments with many redundant actions. For example, in a game where the agent moves left/right but the reward only depends on reaching a goal, Dueling DQN learns the goal's proximity (state value) without wasting effort evaluating every movement action's advantage. This leads to faster convergence and more robust value estimates [5].

2.1.3 Applicable Scenarios

Value-Based methods (DQN, Double DQN, Dueling DQN) excel in *discrete action spaces* where the number of actions is small enough to evaluate via argmax over Q-values. They are less suitable for continuous action spaces (e.g., robot joint angles, autonomous vehicle steering).

2.2. Policy-Based

Unlike Value-Based methods, which learn a value function to derive a policy indirectly, Policy-Based methods parameterize the policy $\pi_\theta(a|s)$, the probability of taking action a in state s given parameters θ and optimize it directly via gradient ascent on the expected cumulative reward. This

framework avoids the need to compute value functions and is naturally suited for continuous action spaces—a key advantage over early Value-Based methods.

2.2.1 REINFORCE and Policy Gradient Theorem

The theoretical foundation of Policy-Based methods was laid by Williams [1], who proposed the REINFORCE algorithm (Simple Statistical Gradient-Following Algorithm). REINFORCE uses the likelihood ratio trick to compute the gradient of the expected reward with respect to the policy parameters. The core update rule is: $\theta \leftarrow \theta + \alpha \cdot \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot R_t$, where:

- α is the learning rate.
- $\log \pi_\theta(a_t|s_t)$ is the log-probability of the action taken.
- R_t is the return (cumulative reward from step *t* to the end of the episode), which weights the gradient—actions leading to higher rewards increase the policy's likelihood of choosing them, while low-reward actions decrease it.

REINFORCE is simple to implement but suffers from high variance in gradient estimates, as the return R_t depends on stochastic transitions and can vary drastically across episodes. This slows convergence and requires large amounts of data.

Sutton et al. addressed this limitation with the Policy Gradient Theorem [2], a landmark theoretical result that formalizes the gradient of the expected reward for two critical reward definitions: start-state discounted reward $E[\sum_{t=0}^{\infty} \gamma^t r_{t+1} | s_0, \pi_\theta]$ and average reward $\lim_{T\to\infty} \frac{1}{T} E[\sum_{t=0}^{T-1} r_{t+1} | \pi_\theta]$. The theorem's core contribution is unifying the gradient expression for both formulations, proving that: $\nabla_\theta J(\theta) = E_{s\sim\rho_\theta(s), a\sim\pi_\theta(a|s)}[\nabla_\theta \log \pi_\theta(a|s) \cdot \Phi(s,a)]$, where:

- $\rho_\theta(s)$ denotes the stationary state distribution induced by policy π_θ.
- $\Phi(s,a)$ is a value term tied to future rewards.

2.2.2 Advantages and Limitations

Policy-Based methods excel in continuous action spaces (sampling directly from parameterized probability distributions) and support natural exploration via stochastic policies, but suffer from high variance in gradient estimates and lack explicit value estimation for policy quality evaluation.

The Actor-Critic framework addresses these limitations by integrating Policy-Based and Value-Based methods, striking a balance between variance and bias.

2.3. Actor-Critic: Merging Value and Policy Optimization

The Actor-Critic framework addresses the trade-offs between Value-Based (low variance, high bias) and Policy-Based (high variance, low bias) methods by integrating two components:

- Actor: learns a parameterized policy $\pi_\theta(a|s)$ to maximize the expected reward.
- Critic: learns a value function $V_{\varphi(s)}$ or $Q_{\varphi(s,a)}$ to estimate the advantage function, which is used to weight the Actor's policy gradient—reducing variance compared to pure Policy-Based methods.

This hybrid approach, enabled by Sutton et al.'s Policy Gradient Theorem [2], has become the most widely used DRL framework for complex tasks (e.g., robotics, autonomous driving) due to its stability and scalability.

2.3.1 Proximal Policy Optimization (PPO): Stability via Policy Clipping

Schulman et al. proposed PPO to solve a major issue with early Actor-Critic methods: policy divergence [6]. In traditional policy gradient methods, large updates can cause the policy to change drastically between iterations, leading to unstable training. PPO introduces a clipped surrogate objective that limits the policy update size by clipping the ratio of the new policy's probability to the old policy's probability $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_old}(a_t|s_t)}$ to a small range (e.g., [0.8, 1.2]). The update rule maximizes: $L_{clip}(\theta) = E_t[\min(r_t(\theta) \cdot \hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon) \cdot \hat{A}_t)]$, where $\hat{A}_t$ is the

estimated advantage function, and ε is a hyperparameter (typically 0.2). This ensures the new policy stays "close" (proximal) to the old policy, stabilizing training while still allowing meaningful updates.

PPO is also computationally efficient, which avoids expensive second-order optimization and can be implemented with standard gradient descent. It has become the de facto standard for many DRL applications, from game AI (e.g., DOTA 2) to robotics, due to its simplicity, stability, and strong performance [6].

2.3.2 Deep Deterministic Policy Gradient: Deterministic Control for Continuous Spaces

Lillicrap et al. [7] developed DDPG to extend the Actor-Critic framework to deterministic policies in continuous action spaces. While stochastic policies like PPO are effective for exploration, deterministic policies ($\pi_\theta(a|s) = a$, mapping states directly to actions) are more efficient for control tasks (e.g., robot arm manipulation) where precise, repeatable actions are needed. DDPG inherits the two stabilization mechanisms from DQN: target networks with parameters updated slowly via polyak averaging $\theta_{target} \leftarrow \tau\theta + (1-\tau)\theta_{target}, \tau \approx 0.001$ and experience replay.

The Actor learns to maximize the expected Q-value (estimated by the Critic) via gradient ascent, while the Critic learns to minimize the squared error between the predicted Q-value and the target Q-value. DDPG was one of the first algorithms to enable effective deterministic control in continuous spaces, achieving state-of-the-art results on tasks like pendulum swing-up and humanoid locomotion [7]. However, it suffers from value overestimation similar to DQN and sensitivity to hyperparameters.

2.3.3 Twin Delayed DDPG (TD3): Mitigating Overestimation and Delayed Updates

Fujimoto et al. [9] proposed TD3 to address DDPG's overestimation bias and training instability. TD3 introduces three key improvements:

- Twin Critics: Two separate Critic networks (Q1 and Q2) are trained to estimate the Q-value. The minimum of the two is used to compute the target value, reducing overestimation (since one Critic's overestimation is offset by the other's more conservative estimate).
- Delayed Actor Updates: The Actor and target networks are updated only once every two Critic updates. This ensures the Critic has time to converge to a reliable Q-value estimate before guiding the Actor's policy updates.
- Target Policy Smoothing: A small amount of noise is added to the target Actor's actions when computing the target Q-value. This regularizes the Critic's learning by making it less sensitive to small changes in the Actor's policy, improving robustness.

TD3 outperformed DDPG on a range of continuous control tasks (e.g., HalfCheetah, Hopper) by up to 50% [9], establishing itself as a more reliable alternative for deterministic continuous control.

2.3.4 Soft Actor-Critic (SAC): Balancing Exploration and Exploitation with Entropy

Haarnoja et al. [8] introduced SAC, a stochastic Actor-Critic algorithm that integrates maximum entropy reinforcement learning—a framework that maximizes not only the expected reward but also the policy's entropy (a measure of randomness). The entropy term encourages the agent to explore broadly while still pursuing high rewards, addressing the classic exploration-exploitation trade-off. SAC's key innovations include:

- Soft Policy Iteration: maximize $Q_{soft}(s,a) = Q(s,a) + \alpha H(\pi(\cdot\,|s))$, where α is the entropy temperature (controlling the balance between exploration and exploitation).
- Stochastic Actor with Reparameterization Trick: the Actor parameterizes a Gaussian distribution over actions, and the reparameterization trick (expressing actions as a function of the policy parameters and a random noise term) is used to compute low-variance policy gradients.
- Twin Critics: similar to TD3.

SAC achieves state-of-the-art performance on continuous control tasks (e.g., humanoid walking) while being more sample-efficient than DDPG and TD3 [8]. Its ability to balance exploration and exploitation makes it particularly useful for tasks with sparse or noisy rewards.

2.4. Multi-Agent Reinforcement Learning (MARL): Learning in Interactive Environments

All previous frameworks assume a single agent interacting with a static environment. However, many real-world tasks (e.g., traffic control, team robotics, multiplayer games) involve multiple agents that interact with each other—creating dynamic environments where an agent's optimal action depends on the actions of others. MARL extends DRL to these scenarios, addressing unique challenges like credit assignment (which agent's action contributed to a reward) and non-stationarity (the environment appears dynamic to each agent, as other agents are also learning).

2.4.1 MADDPG

Lowe et al. proposed MADDPG (Multi-Agent DDPG) [10], the first widely adopted Actor-Critic MARL algorithm, which introduced the Centralized Training with Decentralized Execution (CTDE) paradigm—one of the most influential concepts in MARL:

- Centralized Training: During training, a centralized Critic has access to the states and actions of all agents. This allows the Critic to compute accurate Q-values by accounting for the interactions between agents, solving the credit assignment problem (e.g., in a team game, the Critic can determine which agent's action led to a goal).
- Decentralized Execution: During deployment, each agent uses only its own local observations to select actions (via a decentralized Actor). This is critical for real-world applications where communication between agents may be limited or delayed.

MADDPG extends DDPG to multi-agent settings by giving each agent its own Actor and sharing a centralized Critic. It demonstrated effective coordination in mixed cooperative-competitive environments [10], proving that MARL could scale to complex multi-agent tasks. However, it struggled with scalability—as the number of agents increased, the centralized Critic's input size grew exponentially, slowing training.

2.4.2 Scalable MARL: MAPPO

Yu et al. proposed MAPPO (Multi-Agent PPO) [11], a simple yet highly effective extension of PPO to MARL that addresses MADDPG's scalability issues. Unlike MADDPG's centralized Critic, MAPPO uses a decentralized or partially centralized Critic (e.g., each agent has its own Critic, or a Critic shares information only with nearby agents) to avoid the exponential input size problem. MAPPO's key innovations include:

- Independent Policy Updates: Each agent's Actor is updated independently using PPO's clipped surrogate objective, but the Critic(s) can incorporate information from other agents to guide updates.
- Shared Experience Replay: Agents share a replay buffer, allowing data to be reused across agents and improving sample efficiency.

Surprisingly, MAPPO achieved state-of-the-art performance on a range of cooperative MARL tasks (e.g., StarCraft II micromanagement tasks) with far fewer parameters [11]. Its success demonstrated that scaling single-agent algorithms to multi-agent settings could be highly effective—simplifying MARL implementation and enabling deployment in large-scale systems.

2.4.3 Key Challenges

Despite progress, MARL still faces significant challenges: as other agents learn, the environment's dynamics (from a single agent's perspective) change, requiring algorithms to adapt to shifting reward landscapes; in team tasks with shared rewards, it is difficult to attribute the reward to individual agents' actions—leading to slow or unstable training; most MARL algorithms struggle with large numbers of agents (e.g., 100+), as communication and computation costs grow with the number of agents.

3. Recent Breakthroughs

While the core DRL frameworks have achieved remarkable results in simulated environments (e.g., Atari, StarCraft II), their practical impact has been limited by the sim-to-real gap (performance

degradation when moving from simulation to the real world) and lack of robustness (failure under unexpected conditions). The 2025 research frontier has focused on solving these issues, with breakthroughs in real-world physics alignment, self-play-driven robustness, and scalability. This section analyzes four influential 2025 works that redefine DRL's applicability to real-world tasks.

3.1. ASAP: Aligning Simulation and Real-World Physics for Agile Humanoid Control

A major barrier to deploying DRL in robotics (e.g., humanoid robots) is the sim-to-real gap: simulations often simplify physics (e.g., ignoring friction, noise, or component wear), leading to policies that fail when deployed on real robots. He et al. proposed ASAP (Aligning Simulation and Real-World Physics) [12], a novel sim-to-real framework that bridges this gap by dynamically aligning simulation physics with real-world dynamics.

3.1.1 Multi-Scale Physics Alignment

ASAP's key innovation is a hierarchical physics model that combines two components:

- Low-Level Component: A high-fidelity simulation of individual robot components calibrated using real-world data (e.g., torque measurements, position feedback from the robot). This captures fine-grained physical details that are critical for agile movements.
- High-Level Component: A coarse-grained model of the robot's overall dynamics (e.g., center of mass, balance) that adapts to real-world changes like uneven terrain via online learning. During deployment, the high-level model adjusts the low-level simulation parameters in real time based on feedback from the real robot, ensuring the simulation remains aligned with the real world.

3.1.2 Performance on Agile Humanoid Tasks

ASAP was evaluated on a real humanoid robot (28 degrees of freedom) performing agile tasks:

- Dynamic Jumping: The robot jumped 0.8 meters vertically (a 30% improvement over previous six-to-real methods) with a success rate of 92% (compared to 65% for baseline methods).
- Obstacle Avoidance: The robot navigated a cluttered environment with moving obstacles at 1.2 m/s, maintaining balance even when pushed by unexpected forces.

ASAP's success relies on its ability to learn from real-world data incrementally, ensuring the policy adapts to real-world variability [12]. This makes it a critical step toward deploying agile humanoid robots in unstructured environments (e.g., disaster response, construction).

3.2. Robust Autonomy via Self-Play

Traditional DRL policies often fail when faced with scenarios not seen during training. Cusumano-Towner et al. proposed a paradigm shift: robust autonomy emerges from self-play [13], where an agent learns by competing or cooperating with copies of itself (or "counterparts") in a dynamic environment. This approach generates diverse training scenarios automatically, eliminating the need for hand-crafted datasets of rare events.

3.2.1 Self-Play Framework for Robustness

The framework consists of two components:

- Agent Population: A set of agent variants (with different policy parameters) that interact in a shared environment. Each variant learns to adapt to the others' behaviors, creating a dynamic "arms race" that drives the population to explore edge cases (e.g., a self-driving agent learning to avoid collisions with other agents that take risky actions).
- Robustness Metric: A metric that quantifies how well an agent performs across disturbed environments (e.g., reduced sensor visibility, increased wind resistance). Agents are selected for reproduction in the population based on both their reward and robustness, ensuring the population evolves toward policies that are both high-performing and resilient.

3.2.2 Results on Autonomous Navigation

The framework was tested on an autonomous drone navigation task, where the goal was to fly from point A to point B while avoiding obstacles. Compared to traditional DRL, the self-play-based agent handled 98% of disturbed environments, compared to 62% for the baseline, and adapted to novel obstacles 3x faster, as it had learned to generalize from diverse self-play scenarios.

Cusumano-Towner et al. argue that self-play mimics natural evolution—where species adapt to dynamic ecosystems—making it a scalable way to build robustness without explicit engineering of edge cases [13].

3.3. Scaling Self-Play for Reliable Simulated Driving Agents

Building on the self-play paradigm, Cornelisse et al. focused on autonomous driving [14]. Simulated driving environments are widely used to train DRL-based driving agents, but they often lack the diversity of real-world traffic (e.g., aggressive drivers, construction zones). Cornelisse et al. proposed a scalable self-play framework that generates millions of diverse traffic scenarios in simulation, enabling the training of highly reliable driving agents.

3.3.1 Scalable Scenario Generation

The framework's key innovation is a procedural scenario generator that uses self-play to evolve traffic behaviors:

- Traffic Agent Population: A large population of "other drivers" (controlled by DRL policies) that learn to exhibit diverse behaviors via self-play. These agents compete for space on the road, creating realistic traffic patterns such as traffic jams and lane changes.
- Curriculum Learning: The generator starts with simple scenarios like empty roads and gradually increases complexity as the driving agent improves. This ensures the agent learns foundational skills like lane keeping before tackling complex challenges.

3.3.2 Reliability on Safety-Critical Metrics

The trained driving agent was evaluated on 100,000 simulated scenarios, with 0.02 collisions per 100 km, compared to 0.15 for baseline agents trained on fixed scenarios, and 99.8% compliance with speed limits and traffic lights, even in scenarios with other agents violating rules.

Cornelisse et al. note that scaling self-play to millions of scenarios is feasible due to cloud-based simulation [14], making it a cost-effective alternative to real-world testing.

3.4. CaRL: Scalable Planning with Simple Rewards

Large-scale planning tasks often require DRL agents to learn complex policies with multiple sub-goals. However, designing reward functions for these tasks is challenging—complex rewards can lead to unstable training, while simple rewards may fail to capture all objectives. Jaeger et al. proposed CaRL (Learning Scalable Planning Policies with Simple Rewards), a framework that learns complex planning policies using only simple, interpretable rewards [15].

3.4.1 Hierarchical Planning with Simple Rewards

CaRL uses a two-level hierarchy to bridge simple rewards and complex planning:

- Low-Level Policy: A DRL agent trained with a simple reward (e.g., "avoid collisions") to master low-level skills.
- High-Level Planner: A model-based planner that uses the low-level policy's performance data (e.g., time to reach a waypoint) to optimize high-level objectives (e.g., minimizing total delivery time). The planner decomposes the task into sub-tasks and uses the low-level policy to execute each sub-task.

3.4.2 Scalability on Logistics Optimization

CaRL was tested on a large-scale logistics task: optimizing the routes of 100 delivery trucks across a city with 500 delivery points. Compared to traditional DRL agents trained with complex rewards:

- Training Time: 4x faster, as the simple reward reduced training instability.
- Scalability: Handled 10x more trucks than the baseline, as the hierarchical design decoupled low-level control from high-level planning.
- Interpretability: The simple reward made it easier to debug failures (e.g., a truck getting stuck was traced to a low-level collision-avoidance issue, not a complex reward imbalance).

Jaeger et al. argue that CaRL addresses a key barrier to deploying DRL in planning: the need for expert-designed complex rewards [15]. By using simple rewards and hierarchical decomposition, CaRL makes planning policies more scalable and interpretable.

4. Conclusion

This review has systematically organized the evolution of Deep Reinforcement Learning from its foundational core frameworks to its 2025 breakthroughs in real-world reliability. The four classic frameworks—Value-Based (DQN, Double DQN, Dueling DQN), Policy-Based (REINFORCE, Policy Gradient Theorem), Actor-Critic (PPO, DDPG, TD3, SAC), and Multi-Agent Reinforcement Learning (MADDPG, MAPPO)—have each addressed distinct challenges, from high-dimensional state spaces to continuous action spaces and multi-agent coordination.

The 2025 research frontier—represented by ASAP (sim-to-real alignment), self-play-driven robustness, and CaRL (scalable planning)—marks a critical shift from "simulated optimization" to "real-world reliability." These works address long-standing barriers to DRL deployment, such as the sim-to-real gap and lack of robustness, making DRL applicable to tasks like humanoid robotics and autonomous driving.

Despite significant progress, bottlenecks remain: sample inefficiency, safety, sim-to-real transfer, and MARL scalability. Future research in transfer/meta-learning, constrained optimization, adaptive simulation, and hierarchical MARL will be key to unlocking DRL's full potential. As DRL continues to evolve, it is poised to become a cornerstone of intelligent systems that operate autonomously, reliably, and efficiently in the real world.

References

[1] Williams R J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning, 1992, 8(3): 229-256.

[2] Sutton R S, McAllester D, Singh S, et al. Policy gradient methods for reinforcement learning with function approximation. Advances in neural information processing systems, 1999, 12.

[3] Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013.

[4] Van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double q-learning. Proceedings of the AAAI conference on artificial intelligence. 2016, 30(1).

[5] Wang Z, Schaul T, Hessel M, et al. Dueling network architectures for deep reinforcement learning. International conference on machine learning. PMLR, 2016: 1995-2003.

[6] Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

[7] Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

[8] Haarnoja T, Zhou A, Abbeel P, et al. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. International conference on machine learning. PMLR, 2018: 1861-1870.

[9] Fujimoto S, Hoof H, Meger D. Addressing function approximation error in actor-critic methods. International conference on machine learning. PMLR, 2018: 1587-1596.

[10] Lowe R, Wu Y I, Tamar A, et al. Multi-agent actor-critic for mixed cooperative-competitive environments. Advances in neural information processing systems, 2017, 30.

[11] Yu C, Velu A, Vinitsky E, et al. The surprising effectiveness of ppo in cooperative multi-agent games. Advances in neural information processing systems, 2022, 35: 24611-24624.

[12] He T, Gao J, Xiao W, et al. Asap: Aligning simulation and real-world physics for learning agile humanoid whole-body skills. arXiv preprint arXiv:2502.01143, 2025.

[13] Cusumano-Towner M, Hafner D, Hertzberg A, et al. Robust autonomy emerges from self-play. arXiv preprint arXiv:2502.03349, 2025.

[14] Cornelisse D, Pandya A, Joseph K, et al. Building reliable sim driving agents by scaling self-play. arXiv preprint arXiv:2502.14706, 2025.

[15] Jaeger B, Dauner D, Beißwenger J, et al. Carl: Learning scalable planning policies with simple rewards. arXiv preprint arXiv:2504.17838, 2025.