AI

MDPI

Article

Deep Reinforcement Learning: A Chronological Overview and Methods

Juan Terven

Centro de Investigación en Ciencia Aplicada y Tecnología Avanzada, Instituto Politécnico Nacional, Querétaro 76090, Mexico; jrtervens@ipn.mx

Abstract: Introduction: Deep reinforcement learning (deep RL) integrates the principles of reinforcement learning with deep neural networks, enabling agents to excel in diverse tasks ranging from playing board games such as Go and Chess to controlling robotic systems and autonomous vehicles. By leveraging foundational concepts of value functions, policy optimization, and temporal difference methods, deep RL has rapidly evolved and found applications in areas such as gaming, robotics, finance, and healthcare. Objective: This paper seeks to provide a comprehensive yet accessible overview of the evolution of deep RL and its leading algorithms. It aims to serve both as an introduction for newcomers to the field and as a practical guide for those seeking to select the most appropriate methods for specific problem domains. Methods: We begin by outlining fundamental reinforcement learning principles, followed by an exploration of early tabular Q-learning methods. We then trace the historical development of deep RL, highlighting key milestones such as the advent of deep Q-networks (DQN). The survey extends to policy gradient methods, actor–critic architectures, and state-of-the-art algorithms such as proximal policy optimization, soft actor–critic, and emerging model-based approaches. Throughout, we discuss the current challenges facing deep RL, including issues of sample efficiency, interpretability, and safety, as well as open research questions involving large-scale training, hierarchical architectures, and multi-task learning. Results: Our analysis demonstrates how critical breakthroughs have driven deep RL into increasingly complex application domains. We highlight existing limitations and ongoing bottlenecks, such as high data requirements and the need for more transparent, ethically aligned systems. Finally, we survey potential future directions, highlighting the importance of reliability and ethical considerations for real-world deployments.

Keywords: deep reinforcement learning; policy gradient methods; actor–critic architectures; model-based approaches; on-policy vs. off-policy

check for updates

Academic Editor: Rafał Dreżewski

Received: 27 January 2025
Revised: 17 February 2025
Accepted: 19 February 2025
Published: 24 February 2025

Citation: Terven, J. Deep Reinforcement Learning: A Chronological Overview and Methods. *AI* **2025**, *6*, 46. https://doi.org/10.3390/ai6030046

1. Introduction

Reinforcement learning (RL) is a paradigm of machine learning in which an agent learns an optimal behavior by interacting with an environment, receiving feedback in the form of rewards or penalties, and adapting its actions to maximize long-term returns [1]. Concretely, the agent aims to maximize the expected cumulative reward, which can be written in the infinite-horizon setting as follows:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right], \tag{1}$$

where r_t is the reward received at time t, and $0 \leq \gamma < 1$ is a discount factor that balances the importance of immediate versus future rewards. This stands in contrast to supervised learning, where labels are provided, and unsupervised learning, where the objective is to discover hidden patterns in unlabeled data. RL, instead, focuses on learning through sequential decision-making interactions.

Figure 1 illustrates the basic components of an RL setup, showing how the agent observes states from the environment, selects actions according to a policy, and obtains rewards. The power of RL lies in its potential to tackle problems involving delayed rewards, partial observability, and the need to generalize across broad state spaces. However, practical success often depends on factors such as well-designed reward functions, sufficient training data, and computational resources.

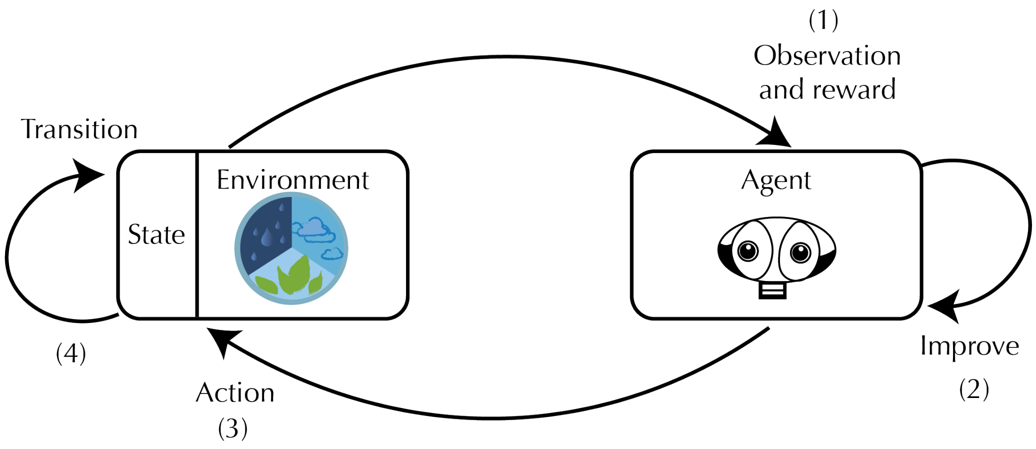

Figure 1. The reinforcement learning loop involves a series of steps: (1) Initially, the agent observes the environment. (2) This observation, along with the received reward, is utilized by the agent to enhance task performance. (3) Subsequently, the agent dispatches an action to the environment, aiming to exert positive control. (4) The environment transitions, altering its internal state based on the agent's action and its preceding state. This loop then initiates once more. The figure is inspired by [2].

While reinforcement learning has shown promise in domains such as healthcare, finance, and resource management, practical deployment remains nontrivial due to issues involving data scarcity, safety constraints, and ethical considerations. In healthcare, for example, patient data can be limited or protected by confidentiality regulations, restricting training opportunities and calling for careful validation before deploying RL-driven treatment policies. In finance, market dynamics are often nonstationary, and risky exploration can lead to large financial losses, necessitating robust risk management and rigorous backtesting. In high-stakes environments like robotics or autonomous driving, strict safety constraints must be met to avoid accidents or physical damage. This typically involves imposing limits on the agent's exploration policies or using methods that guarantee conservative performance under uncertainty. Moreover, ethical considerations arise when RL systems can influence critical outcomes or personal data, as in medical diagnoses or automated trading that affects large populations. Addressing these concerns calls for domain expertise, regulatory compliance, and multidisciplinary collaboration. Although existing work has introduced partial solutions, such as methods for offline learning, safe exploration, or interpretability, there is still no universal framework for seamlessly integrating reinforcement learning into real-world workflows where data, safety, and ethics converge in complex ways.

Reinforcement learning has proven effective across multiple domains with varying complexity levels and challenges, although these successes are often achieved under carefully controlled conditions and may not generalize to all real-world scenarios. For example, RL can require extensive exploration, which is difficult or risky in high-stakes environments, and it may suffer from sample inefficiency or reward design challenges. Despite these limitations, RL has made considerable progress in areas such as game playing, robotics, autonomous driving, finance, healthcare, alignment, and resource management, provided that the tasks and environments are amenable to iterative trial-and-error learning and that sufficient computational resources and domain knowledge are available.

In the realm of games, RL has shown success in both traditional board games and modern video games. Board games such as Go, Chess, and Shogi have long served as benchmarks for problem-solving and strategic reasoning. RL agents can learn to outmaneuver human players by iteratively refining their strategies through self-play, as demonstrated by AlphaGo [3]. Similarly, video games such as Atari, StarCraft II, and Dota 2 provide rich environments for RL research, where agents learn from raw sensory input (pixels) or partially observed states to make sequential decisions. OpenAI's Dota 2 agents [4] further highlighted how deep reinforcement learning can handle high-dimensional state spaces, complex team dynamics, and long planning horizons. Nonetheless, even in gaming, reward design and exploration can pose significant obstacles, illustrating the broader need for careful problem formulation and efficient search strategies.

In robotics and control systems, RL has been used to train autonomous agents for tasks such as grasping objects, navigation, and maintaining balance or locomotion [5–7]. By interacting with real or simulated environments, robot controllers refine their behavior based on reward signals indicating successful actions. This paradigm often eliminates the need for explicit modeling of the robot's dynamics [5,8]. However, deploying RL on physical robots highlights practical challenges, including data scarcity (due to the cost and time of real-world experimentation) and safety constraints (collisions or hardware damage must be avoided). Sim-to-real transfer methods partially mitigate these risks by pre-training in simulation, but discrepancies between simulated and real conditions remain a key area for research.

Autonomous vehicles also leverage RL to handle dynamic and uncertain driving environments [9,10]. Agents must learn lane-merging behaviors, obstacle avoidance, and real-time adaptation to rapidly changing traffic patterns. Simulators such as CARLA [11] enable exploration without risking real accidents. However, significant challenges arise when translating learned policies to actual roads, where errors could endanger human lives. Rigorous transfer learning and robust safety guarantees are therefore paramount. Some studies investigate safe multi-agent RL to better handle vehicle interactions [12], but formal certifications remain rare, and local regulatory requirements often demand interpretability and strict safety validation.

Finance and algorithmic trading benefit from RL's capacity to optimize sequential decision processes [13–15]. Here, an RL agent typically aims to maximize profit or limit risk by choosing actions such as buy, sell, or hold based on market indicators. Portfolio management tasks can be framed as Markov decision processes [16], although real-world data for training are often limited or expensive, and continuously changing market dynamics can invalidate learned policies. Thorough risk management protocols are essential to prevent large losses. Regulatory frameworks also impose constraints on automated trading, raising questions about the transparency and fairness of RL-driven strategies when real capital is at stake.

Healthcare offers another important yet challenging domain, where RL optimizes treatment policies, dosage regimens, and resource allocation [17–19]. The environment may be represented by patient states (vital signs, test results, demographic data), and actions map to medical interventions. However, clinical data are often scarce or proprietary, limiting broad experimentation. Moreover, interventions must be justified ethically and medically, since suboptimal exploration can harm patients. Existing studies emphasize careful offline analysis and validation using retrospective patient data [20], yet deploying RL in live clinical settings remains difficult due to both safety and ethical constraints.

Alignment-focused methods also illustrate how RL can address human values and intentions. Reinforcement learning from human feedback (RLHF) uses feedback from non-expert assessors to train a reward model [21], refining policies even when human preferences are not easily codified. This approach has successfully shaped large language models [22,23], improving their outputs according to user preferences. While RLHF is promising, human oversight may be expensive and limited in scale, and it introduces new vulnerabilities related to bias or malicious feedback. Additional research is needed to reliably detect misaligned behaviors and ensure that RL-driven systems continue to serve user interests.

In resource management scenarios, RL is used in distributed systems and cloud infrastructures [24–26]. Data centers rely on RL to allocate computational resources, balance server loads, and regulate energy consumption. Agents seek to minimize power usage without sacrificing service quality, leading to potentially large cost savings. Nevertheless, reward functions in such systems must balance conflicting objectives like performance, scalability, and energy cost, and designing these reward signals can be nontrivial. Scalable solutions further require extensive testing under real-world traffic conditions to ensure stable operation.

While reinforcement learning can achieve success in these diverse applications, it is not a universal solution. Factors such as data scarcity, safety requirements, ethical constraints, and regulatory compliance can significantly restrict naive implementations. In healthcare and finance, legally mandated oversight may slow experimentation, while in robotics and autonomous driving, exploring unsafe actions is unacceptable. Ongoing research on safe exploration, offline or constrained RL, and transparent decision-making reflects the need for more nuanced methods that address practical deployment hurdles. Identifying appropriate domain-specific reward functions and ensuring robust generalization to unseen states remain complex tasks. By acknowledging these constraints and refining RL techniques accordingly, researchers and practitioners can better align algorithmic innovation with the realities of real-world operations.

From Reinforcement Learning to Deep Reinforcement Learning

In its early years, reinforcement learning (RL) research was mainly based on tabular methods or relatively small-scale function approximators such as linear models with carefully hand-engineered features [27,28]. These methods excelled in low-dimensional tasks where state spaces were sufficiently small to be fully enumerated or captured by a compact feature set. However, as researchers pushed toward more complex problems, including high-dimensional or continuous state spaces, the limitations of these traditional techniques became increasingly apparent. In domains such as robotics, game playing, and autonomous systems, the number of possible states can grow exponentially, making it infeasible for tabular methods or simple function approximators to learn reliable policies at scale.

Deep neural networks, known for their representational capacity and the ability to learn hierarchical characteristics directly from raw data [29,30], offer a way to overcome these limitations. Unlike hand-engineered features, deep networks can automatically discover relevant features from high-dimensional inputs, be they images of game screens, sensor readings, or more abstract data streams. This synergy of RL with deep learning arose naturally from the need to handle large state spaces while avoiding labor-intensive feature engineering.

The transformative breakthrough occurred when deep Q-networks (DQNs) demonstrated human-level performance on dozens of Atari 2600 video games using only raw pixel inputs and game scores as the sole training signals [31,32]. This achievement highlighted the fact that a single deep convolutional network can learn to approximate action values effectively across a wide range of games.

To illustrate the contrast between classical Q-learning and its deep-learning counterpart, Figure 2 compares a tabular approach (top) with a neural network-based deep Q-learning architecture (bottom). In standard Q-learning, each state–action pair has a specific entry in a Q-table, which can quickly become infeasible for high-dimensional or continuous state spaces. Deep Q-learning avoids enumerating all possible states by using a neural network to approximate the Q-function, mapping raw state observations directly to estimated Q-values for each action.

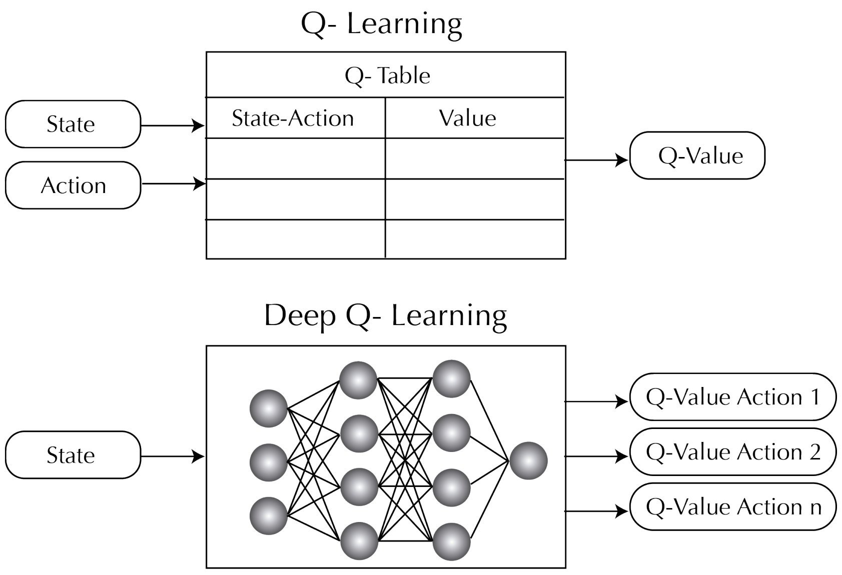

Figure 2. A conceptual comparison of tabular Q-learning (top) vs. deep Q-learning (bottom). In the tabular approach, each state–action pair is assigned a single Q-value entry, making it impractical for large or continuous state spaces. Deep Q-learning replaces the Q-table with a neural network that approximates action values from high-dimensional inputs such as images, enabling the algorithm to handle more complex tasks.

Following the success of DQN, researchers rapidly expanded the field of deep RL by introducing novel algorithms and architectures aimed at improving sample efficiency, stabilizing training, and addressing a spectrum of other challenges. Methods such as policy gradients [33], actor–critic frameworks [34,35], and distributional RL [36] quickly became mainstream, each focusing on different facets of the RL problem. In parallel, advancements in hardware—particularly GPUs and specialized accelerators—greatly enhanced the feasibility of large-scale experiments. Just as crucial, simulation frameworks provided standardized environments for benchmarking and training RL agents. OpenAI's Gym [37] and the more recent Gymnasium [38] offered a wide variety of task environments in both discrete and continuous domains, while other simulators such as MuJoCo [39,40], Unity ML-Agents [41], and diverse game engines [42,43] enabled richer, more complex scenarios.

In the tabular setting, Q-learning and other temporal difference methods are known to converge under mild conditions [1]. However, when function approximation with deep neural networks was introduced, convergence was no longer guaranteed in general. The interplay of function approximation, bootstrapping, and off-policy data (often called the "deadly triad") can lead to divergence in practice [1]. Although techniques like target networks and double DQN [44] mitigate some instabilities, the field still lacks a universally accepted convergence proof for large-scale, deep value-based, or policy-based methods.

Today, deep RL encompasses a broad spectrum of research topics, spanning from stability and convergence analyses [45], to advanced exploration strategies [46], to model-based planning [47,48]. The field continues to grow, reflecting an ever-increasing demand for intelligent agents capable of learning complex behaviors directly from rich sensory inputs. With breakthroughs arriving steadily, deep RL remains one of the fastest-evolving areas in machine learning, promising innovations for everything from autonomous vehicles to healthcare and beyond.

This paper provides a historical overview of early reinforcement learning (RL). It examines the evolution of RL with function approximation, neural networks, and policy gradients, highlighting deep Q-learning and its variants, policy gradient methods like REINFORCE, actor–critic algorithms, and advanced techniques such as trust region policy optimization (TRPO), proximal policy optimization (PPO), and soft actor–critic (SAC). We explore model-based methods and how RL combined with planning can enhance sample efficiency. The paper also considers evolutionary strategies and large-scale RL systems like AlphaStar, OpenAI Five, and AlphaFold. It concludes with a summary of methods, guidance on their application, and a discussion of challenges, open research questions, and potential future directions.

2. A Chronological Overview of Reinforcement Learning

In this section, we provide a chronological overview of the field of reinforcement learning, highlighting the key theoretical developments, landmark algorithms, and real-world breakthroughs that have shaped its growth over the decades. Figure 3 presents a high-level timeline of major RL milestones, starting from foundational concepts in the 1940s and 1950s, through the introduction of model-free methods in the 1980s and 1990s, to the emergence of deep RL and large-scale systems in recent years.

We begin by examining the foundations of RL, focusing on the Bellman equation, dynamic programming, and early temporal difference methods. Subsequently, we will trace how these ideas evolved to include function approximation, ultimately culminating in today's deep RL frameworks that leverage powerful neural networks and massive computational resources.

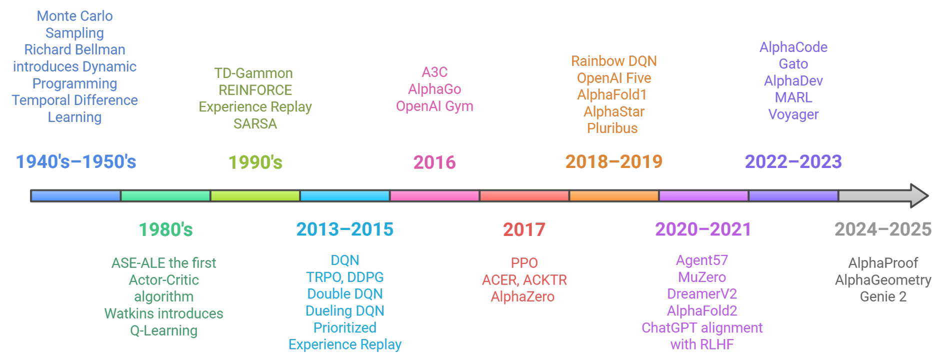

Figure 3. A timeline of major milestones and breakthroughs in reinforcement learning, illustrating its evolution from early theoretical foundations to modern deep RL successes.

2.1. The Bellman Equation and Dynamic Programming

Long before the advent of deep learning, the mathematics of RL were built on classical concepts from optimal control, dynamic programming, and Markov decision processes (MDPs) [1,49]. An MDP is defined by a tuple

$$(S, A, P, R, \gamma),$$

where S is a set of states, A is a set of actions, $P(s' \mid s, a)$ is a transition probability function describing the likelihood of moving from state s to s' by taking action a, $R(s, a)$ is a reward function, and $\gamma \in [0, 1]$ is a discount factor emphasizing the importance of future rewards.

The concept of value functions arises from the Bellman optimality equation [50]. For a state s, the optimal state-value function $V^*(s)$ is given by the following:

$$V^*(s) = \max_{a \in A} \Big[R(s, a) + \gamma \sum_{s'} P(s' \mid s, a)\, V^*(s') \Big]. \tag{2}$$

Similarly, the optimal action-value function $Q^*(s, a)$ satisfies:

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s'} P(s' \mid s, a) \max_{a'} Q^*(s', a'). \tag{3}$$

These equations can be solved via dynamic programming when the full model (P and R) is known. A classic practical example involves grid-world navigation: if one knows all possible moves and the probabilities of transitioning between grid cells, one can apply value iteration or policy iteration to identify the optimal path to a goal [1]. However, in many real-world tasks (e.g., robot manipulation or financial trading), the model is unknown or too complex to specify, motivating model-free methods.

Reinforcement learning algorithms can be broadly categorized as on-policy or off-policy. An on-policy algorithm evaluates or improves the policy that is currently used by the agent to make decisions. In contrast, an off-policy algorithm learns about a target policy different from the behavior policy that generates transitions.

2.2. Temporal Difference Learning and Q-Learning

Early RL research introduced temporal difference (TD) learning [51], which combines ideas from Monte Carlo estimation [1] and dynamic programming [52]. Q-learning [53] is a model-free off-policy TD method that incrementally updates $Q(s, a)$ using the following rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \Big[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \Big], \tag{4}$$

where α is the learning rate. This simple yet powerful algorithm can learn optimal policies even when the agent explores the environment via a non-optimal (e.g., ϵ-greedy) strategy. A well-known practical demonstration involves training an agent to play a simple maze game: the agent starts with no knowledge of the maze layout but incrementally learns which actions (left, right, up, down) lead to the highest reward when exiting the maze.

The hallmark of these early methods was a tabular approach, where each state–action pair had an explicit entry in a table. However, tabular Q-learning becomes infeasible for large or continuous state spaces. For instance, controlling an inverted pendulum with thousands of possible angle-velocity combinations would require a massive state–action table, making pure tabular methods computationally intractable [54].

2.3. Eligibility Traces and SARSA

Extensions to Q-learning came in the form of eligibility traces (e.g., TD(λ), SARSA(λ)), which provide a bridge between Monte Carlo and TD methods [1]. SARSA (state–action–reward–state–action) [55] is an on-policy variant of Q-learning, where updates incorporate the agent's actual behavior policy rather than the greedy policy used in Q-learning's update. A practical example can be found in adaptive traffic signal control [56], where an agent continuously learns the value of different light-timing actions based on real traffic conditions, balancing current throughput and future congestion.

While these methods marked major progress, they still faced a fundamental challenge, namely, scaling to high-dimensional or continuous domains. Continual improvements in training stability, state representations, and sample efficiency were necessary to move from small toy problems (e.g., simple mazes or single-joint control) toward the complex tasks we see in modern applications.

2.4. Early Attempts at Function Approximation in RL

2.4.1. Linear Function Approximation

To handle large or continuous state spaces, researchers began using function approximators for value functions. One of the simplest approximators is a linear model:

$$\hat{Q}(s, a; \boldsymbol{w}) = \boldsymbol{w}^\top \phi(s, a), \tag{5}$$

where $\phi(s, a)$ is a feature vector, and $\boldsymbol{w}$ is a weight vector. Learning $\boldsymbol{w}$ typically involves gradient descent updates that minimize a temporal difference error. Despite its simplicity, linear methods with carefully engineered features worked for some tasks—for instance, basic robotic control with precomputed distances or velocities, or simplified portfolio management [57].

However, performance strongly depended on the quality of the features, limiting widespread applicability. In high-dimensional or unstructured domains such as raw images, designing the right set of features was labor-intensive and prone to suboptimal solutions.

2.4.2. Neuro-Dynamic Programming and Early Neural Networks

Parallel to the above efforts, some researchers began exploring neural networks for function approximation as early as the 1980s and 1990s, a field sometimes referred to as neuro-dynamic programming [58]. One iconic example is Tesauro's TD-Gammon [59], which used a multi-layer perceptron to approximate the value function in the game of Backgammon, achieving near-expert-level play. However, these approaches often suffered from instability, mainly because neural networks were difficult to train in the presence of highly correlated data streams and non-stationary targets. The lack of stable optimization tricks (e.g., improved weight initialization [60,61], batch normalization [62], or better activation functions [63,64]) that surfaced decades later also contributed to the challenges.

Nevertheless, the conceptual seeds for merging RL and neural networks were planted. Early successes in controlled tasks hinted at the potential for deep networks to excel in RL if training could be stabilized. This idea presaged the breakthroughs that would come years later with the advent of more powerful computing hardware and refined deep learning methodologies.

3. The Birth of Deep Reinforcement Learning

A confluence of factors in the late 2000s and early 2010s—large datasets, powerful GPUs, improved network architectures, and refined optimization techniques—led to what is often referred to as the deep learning renaissance. Driven by advances in convolutional neural networks [65], recurrent architectures [66], and large-scale parallel computing, deep networks began achieving state-of-the-art performance in computer vision, speech recognition, and natural language processing [29,30]. These successes prompted researchers to revisit long-standing challenges in reinforcement learning (RL) with renewed optimism.

Among earlier attempts to combine RL and neural networks was Riedmiller's neural fitted Q-iteration (NFQ) [67], which showcased the potential of batch updates and function approximation for control tasks. However, factors such as unstable gradients, correlated data samples, and the non-stationary nature of the environment's interactions made it challenging to harness the full power of deep networks in RL. As a result, many early neuro-dynamic programming approaches [58] struggled to demonstrate robust performance at scale.

In the early 2010s, improved weight initialization schemes [60,61], regularization techniques such as dropout [68], and better optimizers (e.g., Adam [69]) had matured, enabling researchers to train deeper networks on increasingly diverse data. Furthermore, the rapid growth of specialized graphics processing units (GPUs) dramatically cut training time, facilitating iterative experimentation and systematic tuning of hyperparameters. This period set the stage for major breakthroughs in applying deep models to RL environments.

3.1. Atari Breakthrough: Deep Q-Network (DQN)

A pivotal moment in deep reinforcement learning occurred in 2013 and 2015 when researchers at DeepMind introduced the deep Q-network (DQN) [31,32]. By combining Q-learning with convolutional neural networks (CNNs) to process raw pixel inputs, DQN was able to learn directly from high-dimensional sensory data (e.g., frames of Atari games). However, training a deep neural network to estimate action-value functions $Q(s,a)$ introduces several challenges. Two key issues are:

1. Correlated data: Consecutive samples from an environment are highly correlated, meaning that training on them sequentially can cause stochastic gradient descent to perform poorly or converge slowly. Neural network updates typically assume that training samples are independent and identically distributed (i.i.d.).
2. Non-stationary targets: In Q-learning, target values depend on the current estimate of the Q-function itself, which changes during training. This leads to a moving target problem, where the updates chase a target that is simultaneously shifting with every training step.

DQN addressed these problems with two crucial stabilization techniques:

- Experience replay: Instead of updating from the most recent transition (s,a,r,s') only, the agent stores transitions in a replay buffer and samples mini-batches randomly for training. By doing so, the training data become more i.i.d., breaking the strong correlations present in sequential observations. This technique also improves sample efficiency by reusing past experiences, mitigating the variance introduced by consecutive, highly correlated samples [70].
- Target network: In standard Q-learning, the updated target $r + \gamma \max_{a'} Q(s',a')$ uses the same Q-network that is being trained, causing the target to shift each time the parameters update. DQN mitigates this non-stationary target issue by using a target network, a copy of the Q-network that is held fixed for a number of iterations and then periodically updated to the latest weights of the main network. This slows down changes in the target, reducing oscillations and stabilizing learning.

As illustrated in Figure 4, the DQN agent processes the last four frames of the game to capture motion (e.g., the velocity of the ball and paddles in Atari Pong) and then uses convolutional layers to learn spatial features. These extracted features are passed to fully connected layers, which output a separate Q-value estimate for each possible action. By training this deep architecture with large-scale experience replay and a target network, DQN achieved near or above human-level performance on many Atari 2600 games solely from raw pixel inputs and game scores [42].

The demonstration that an end-to-end deep neural network could learn effective control policies directly from raw high-dimensional data was both surprising and transformative. It showed that, with proper stabilization techniques, neural networks could be viable function approximators in reinforcement learning, paving the way for modern deep RL methods.

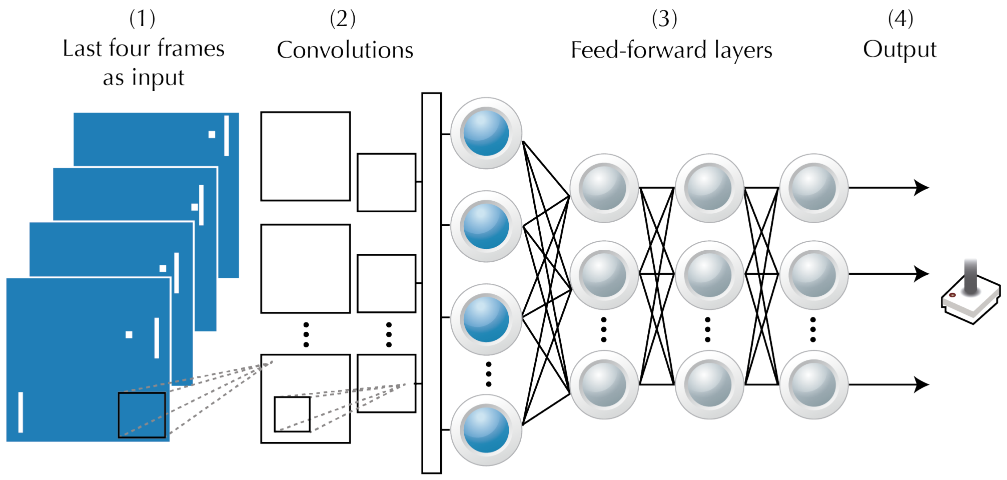

Figure 4. Atari DQN network architecture. (1) The last four frames are used to infer motion information (e.g., velocities of the ball and paddles). (2) A convolutional neural network extracts spatial features from these frames. (3) A feed-forward network further processes these features. (4) The final output estimates the expected Q-value for each possible action. Figure inspired by [2,32].

3.2. Extensions and Variants of Deep Q-Learning

Following DQN's success, a surge of improvements emerged:

- Double DQN: Addresses overestimation bias by decoupling the selection of the action that maximizes Q from the evaluation of that action [44].
- Dueling DQN: Introduces separate streams in the network to estimate the state-value function $V(s)$ and the advantage function $A(s,a)$. The final Q-value is then $Q(s,a) = V(s) + A(s,a)$. This architecture helps in states where the action choice is less critical [71].
- Prioritized experience replay: Assigns higher sampling probability to transitions with higher TD error, accelerating the correction of large errors, and improving sample efficiency [72].
- Rainbow DQN: Combines several improvements (double DQN, dueling networks, prioritized replay, multi-step learning, distributional RL, noisy nets) into a single agent that achieves strong performance across Atari games [73].
- Distributional DQN: Moves away from learning only the expected return to learning the distribution of returns, offering a richer training signal and often better policies [36].

Although Q-learning-based approaches remain popular, they may face challenges in continuous action spaces and can require large amounts of experience to converge. These limitations prompted the development of policy gradient methods, focusing on directly parameterizing and optimizing the policy.

4. Emergence of Policy Gradient Methods

Early research in reinforcement learning (RL) primarily focused on value-based methods like TD(λ), Q-learning, and SARSA, all of which derive a policy from estimates of state or action values. However, in the 1990s, a parallel line of work began to explore the idea of direct policy search, or policy gradient methods, where the policy itself was parameterized and optimized [33,74,75]. Over time, these efforts coalesced into a family of algorithms known for naturally handling continuous action spaces and facilitating smoother exploration strategies, contrasting with the sometimes brittle heuristics of value-based RL.

As shown in Figure 5, a generic policy gradient approach involves feeding the current state into a policy network that outputs action probabilities, which are then used to select actions in the environment. After the agent observes the resulting rewards and transitions, it adjusts the policy parameters by backpropagating a learning signal derived from the log probabilities of the chosen actions multiplied by their returns. This high-level flow applies to many policy gradient algorithms, though specific implementations differ in how they estimate returns or incorporate baselines.

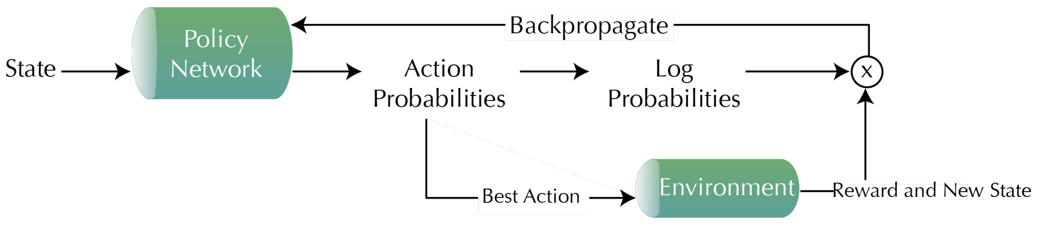

Figure 5. A simplified schematic of a generic policy gradient method. The policy network outputs probabilities for each action, and the agent samples an action to execute in the environment. Once a reward and a next state are observed, the negative log probabilities of the selected actions are multiplied by their corresponding returns and used to update the policy.

4.1. REINFORCE (1992)

One of the earliest explicit policy gradient algorithms is *REINFORCE* [76], which was introduced in the early 1990s. Often referred to as the Monte Carlo Policy Gradient, REINFORCE directly adjusts policy parameters using the log probabilities of actions weighted by their returns. Concretely, for a sampled trajectory, i.e.,

$$\tau = \{s_0, a_0, r_1, s_1, a_1, r_2, \ldots\},$$

generated by policy π_θ, the gradient update is as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\, G_t\right], \tag{6}$$

where $G_t = \sum_{k=t}^{T-1} \gamma^{k-t} r_{k+1}$ is the return from time step t. Although conceptually straightforward, REINFORCE suffers from high-variance gradient estimates, partly because each update depends on entire trajectories, and learning signals only arrive at the end of an episode. Nonetheless, it represented a major conceptual leap by advocating for direct optimization of the policy function.

As illustrated in Figure 6, the policy network in REINFORCE likewise produces action probabilities, but here, the negative log probabilities of the chosen actions are accumulated over a whole episode. Once the episode concludes, the product of these log probabilities and the respective returns becomes the learning signal for gradient ascent on the expected return.

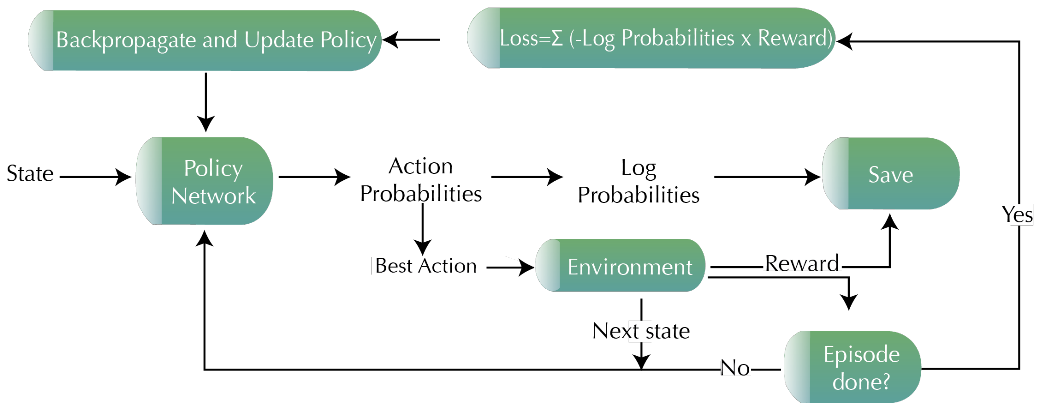

Figure 6. A schematic of the REINFORCE algorithm, inspired by [77]. The policy network outputs probabilities for each action. After an episode finishes, the update is computed by weighting the negative log probabilities of the taken actions by their cumulative rewards, effectively increasing the likelihood of rewarding actions.

Extension to Baselines and Variance Reduction (Late 1990s)

To address the high variance in REINFORCE, researchers in the late 1990s introduced techniques such as baseline subtraction [1,76]. In policy gradient methods, we define an objective function $J(\theta)$ that measures the expected return when following a parameterized policy π_θ. A trajectory τ is a sequence of states, actions, and rewards encountered by following π_θ in the environment, and we write $\tau \sim \pi_\theta$ to indicate that τ is drawn from the distribution over trajectories induced by π_θ.

The operator $\mathbb{E}_{\tau\sim\pi_\theta}[\cdot]$ denotes the expectation (or average) with respect to all possible trajectories τ generated by π_θ. Concretely, this expectation sums over all trajectories, weighting each by its probability under the policy π_θ.

By subtracting a learned baseline $b(s)$, commonly the state-value function $V^\pi(s)$, the policy gradient update can be written as follows:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t \mid s_t)\,\big(G_t - b(s_t)\big)\right], \tag{7}$$

where G_t is the return (the sum of discounted rewards) following time t. This subtraction does not change the expected gradient but can substantially reduce its variance by centering the update around the baseline.

Related ideas, such as advantage updating [78], introduced the advantage function $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$, which focuses on whether an action is better or worse than expected and thus provides more targeted updates.

4.2. Incorporating Policy Search into Practical Robotics (2000s)

By the early 2000s, policy gradients began seeing applications in robotics and control [5,6,79]. Robots learning to swing up pendulums or grasp objects demonstrated that policy gradients handle continuous torques naturally without discretization. For instance, in [6] it was shown how policy gradient formulations in robotics could outperform value-based counterparts by efficiently searching high-dimensional action spaces. Yet, these methods still struggled with sample inefficiency and high variance, motivating further innovations.

4.3. Generalized Advantage Estimation and Modern On-Policy Methods (Mid-2010s)

Building on these foundations, the generalized advantage estimator (GAE) [80] introduced a flexible way to trade off bias and variance when estimating advantages. Instead of using full returns (as in REINFORCE) or pure bootstrapping (as in temporal difference learning), GAE creates a more stable, low-variance advantage signal. This development improved many subsequent on-policy algorithms, such as trust region policy optimization (TRPO) [81] and proximal policy optimization (PPO) [82], both of which grew from policy gradient principles combined with additional constraints or clipping to control large updates.

With techniques like GAE, policy gradient methods became central to high-profile RL applications. For instance, controlling legged robots on complex terrain [83] or fine-tuning manipulator arms to handle delicate objects [84] no longer required extensive manual feature engineering. In these tasks, the policy directly outputs continuous torque commands, while the critic, which is a learned value function, provides stable feedback for gradient updates.

4.4. Deterministic Policies and Beyond

Most early policy gradient research assumed stochastic policies, but explorations into deterministic policy gradients emerged to tackle large action spaces with potentially lower-variance updates. Deep deterministic policy gradient (DDPG) [35], introduced around 2015, extended the policy gradient concept to a deterministic setting in high-dimensional continuous control tasks. This approach combined ideas from Q-learning (by learning an action-value critic) with a parameterized actor that outputs deterministic actions.

Deterministic policies can be easier to train in some continuous domains but rely heavily on exploratory noise processes. Moreover, whether a policy is stochastic or deterministic, policy gradient approaches must contend with sample efficiency and hyperparameter tuning challenges [85]. This has led to continuous evolution and refinement, culminating in advanced algorithms like soft actor–critic (SAC) [83] and distributed frameworks that unify policy gradient principles with large-scale parallelism.

From the introduction of REINFORCE in 1992 to modern actor–critic hybrids, policy gradient methods have matured into robust tools for tasks where continuous, high-dimensional, or stochastic actions dominate. However, despite these improvements, policy gradient methods can still suffer from high variance and sample inefficiency. To address these shortcomings, modern actor–critic techniques blend the strengths of policy gradients (actor) with the stability of a learned value function (critic), forming a class of algorithms that often outperform pure policy or value-based methods. The next section explores the evolution of these actor–critic methods and how they ultimately revolutionized continuous control, large-scale parallelization, and more.

5. Actor–Critic Methods

Actor–critic algorithms naturally combine policy gradient methods and value-based techniques, aiming to combine the strengths of both while mitigating their weaknesses. Early work on actor–critic can be traced back to the 1980s and 1990s [33,86], where researchers recognized that maintaining a separate critic (i.e., a value function or an action-value function) alongside a parameterized actor (i.e., a policy) could reduce variance in updates and accelerate convergence. Over time, actor–critic frameworks have evolved significantly, culminating in modern deep actor–critic approaches that dominate continuous control and other complex RL tasks.

As illustrated in Figure 7, the actor–critic setup involves an actor component that directly outputs the policy $\pi_\theta(a \mid s)$, and a critic component that approximates a value function (either $V^\pi(s)$ or $Q^\pi(s,a)$). The critic's estimate guides the actor's parameter updates by providing a training signal—often in the form of a temporal difference (TD) error—which helps reduce variance compared to purely policy-based methods. A key concept here is the advantage function $A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$ [1], reflecting how much better or worse an action is compared to the average action at that state.

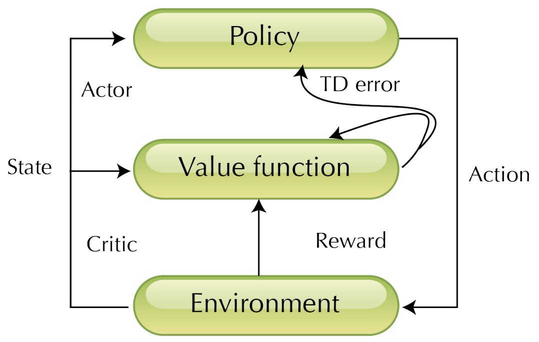

Figure 7. A schematic of the actor–critic architecture, inspired by [1]. The actor (policy) selects actions based on the current state, while the critic (value function) produces a temporal difference (TD) error signal to guide policy updates. This synergy combines the strengths of policy gradient and value-based methods, reducing variance and improving convergence in high-dimensional tasks.

In practice, a critic actor could learn to control a simple robot in a 2D simulator by updating its policy parameters based on the critic's assessment of how beneficial each action was relative to the baseline performance. This synergy between policy and value estimation proved essential for subsequent breakthroughs in deep reinforcement learning.

5.1. Deep Deterministic Policy Gradient (DDPG, 2015)

A major leap occurred when deep deterministic policy gradient (DDPG) [35] extended the actor–critic concept to high-dimensional, continuous action spaces in 2015. Before DDPG, continuous action tasks often required manual discretization or specialized methods. DDPG introduced the following:

- A deterministic policy $\mu_{\theta}(s)$, bypassing the need to sample from a distribution at every step.
- A replay buffer to store transitions and train the critic with uncorrelated mini-batches.
- Separate target networks for both the actor and the critic to stabilize training.
- Ornstein–Uhlenbeck noise [87] for temporally correlated exploration in continuous domains.

This architecture excelled in robotic manipulation tasks, such as controlling a simulated robotic arm in MuJoCo [8,39], where the agent learns to reach or grasp objects through trial-and-error interactions. By leveraging a deterministic actor, DDPG alleviated some of the variance issues in large action spaces. However, its performance can be sensitive to hyperparameter choices (e.g., learning rates, noise parameters, and network architecture), leading to brittle convergence behaviors if not tuned carefully [85].

5.2. Asynchronous Advantage Actor–Critic (A3C, 2016)

As deep RL advanced, researchers turned to methods that could more efficiently collect experiences and reduce correlation in training data. In 2016, asynchronous advantage actor–critic (A3C) [34] was proposed. Instead of a single agent interacting with the environment, A3C leverages multiple parallel environments (or actors) running asynchronously. Each worker conducts the following:

1. Interacts with its copy of the environment,
2. Computes gradients for both actor and critic locally,
3. Periodically updates a shared global model.

This parallelism decorrelates experience across workers, accelerating training and overcoming the replay buffer dependence found in methods like DQN [32] and DDPG [35].

In practice, A3C was shown to learn Atari games [42] faster than previous on-policy approaches and also proved effective for continuous control tasks in domains like robotic locomotion. A well-known example is training multiple agents to play Breakout or Pong concurrently, dramatically reducing overall wall-clock time to achieve strong performance. This asynchronous paradigm influenced subsequent large-scale RL frameworks, such as IMPALA [88], which further improved computational efficiency and scalability by building on similar multi-actor insights.

5.3. Trust Region Policy Optimization (TRPO, 2015) and Proximal Policy Optimization (PPO, 2017)

While asynchronous actor–critic methods boosted performance, policy gradient instability remained a concern. Trust region policy optimization (TRPO) [81] tackled this by enforcing a constraint on how much the policy can change between updates, using a form of local, second-order optimization. This prevented catastrophic policy shifts but came with considerable computational overhead.

To address the complexity of TRPO, proximal policy optimization (PPO) [82] introduced a simpler, first-order technique that clips the probability ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}, \tag{8}$$

where π_θ is the policy parameterized by θ. Intuitively, $r_t(\theta)$ measures how much more (or less) likely it is for the new policy π_θ to select action a_t at state s_t compared to the old policy $\pi_{\theta_{\text{old}}}$. If $r_t(\theta) > 1$, the new policy is more likely to choose a_t; if $r_t(\theta) < 1$, the new policy is less likely.

By capping $r_t(\theta)$ within $\left[1-\epsilon, 1+\epsilon\right]$, PPO emulates a trust-region update but is easier to implement and tune. This clipping ensures that policy updates do not shift the probability distribution over actions too drastically in a single step, promoting more stable training.

PPO rapidly gained popularity in both academia and industry due to its empirical performance, simplicity, and versatility. OpenAI famously employed PPO to train complex behaviors in its robotic hand manipulation tasks, such as solving a Rubik's Cube in hand, demonstrating robust learning under physical constraints in the real world [84,89]. Additionally, PPO has proven effective in multi-agent game environments (e.g., large-scale simulation and strategy games) where sample efficiency and stable updates are crucial [4].

Moreover, reinforcement learning from human feedback (RLHF) often leverages PPO to optimize language models and align them with human preferences [22,90]. In RLHF, a reward model is trained from human feedback on the model's output. Then, PPO is used to adjust the policy (e.g., the parameters of the language model) to produce responses that better match human annotations.

5.4. Soft Actor–Critic (SAC, 2018)

Despite these advances, exploration in continuous control tasks remained challenging. Soft actor–critic (SAC) [83] introduced the idea of maximum entropy RL into an actor–critic framework, aiming to encourage stochastic policies that remain diverse in their action selection. The key features of SAC include the following:

- Two Q-networks for more reliable target estimates,
- A temperature parameter α controlling the balance between maximizing reward and maximizing entropy.
- An off-policy setup that allows sample reuse from a replay buffer, significantly boosting sample efficiency.

By optimizing not just for reward but also for high action entropy, SAC avoids collapsing to deterministic or overly narrow policies, substantially improving exploration. In standard MuJoCo [39] benchmarks like HalfCheetah, Hopper, or Walker2D [37,91], SAC achieves state-of-the-art performance with stable convergence and lower sample complexity compared to earlier methods [83,92]. In practical robotic scenarios, for example, navigating uneven terrain or manipulating objects under uncertainty, SAC's stochastic exploration allows the agent to discover robust footholds and grasp strategies without extensive manual tuning [93].

From a theoretical standpoint, these model-free RL algorithms typically require a large number of samples to converge, even in simplified environments. In tabular or linear approximations, frameworks such as PAC-MDP [94] provide polynomial upper bounds on sample complexity. However, these results do not directly translate to deep function approximators, which can exhibit vastly different generalization behaviors. In practice, methods such as PPO [82] or SAC [83] often demand millions of interactions to learn robust policies. Developing theoretical frameworks that combine representation learning with RL remains a significant open challenge [45].

Deep RL involves optimizing high-dimensional, non-convex objective functions, making global convergence elusive. Policy gradient algorithms, including TRPO [81] and PPO, incorporate mechanisms (e.g., trust regions, clipping) that aim to stabilize training, but they do not offer strict global convergence guarantees. Off-policy algorithms (e.g., DQN, SAC) face additional complexities such as distribution shifts in replay buffers and overestimation biases in Q-values, further complicating theoretical analyses [95].

In chronological order, actor–critic methods have transformed from simple policy/value hybrids to sophisticated deep algorithms capable of tackling a broad spectrum of environments. The journey began with the insight that an actor–critic approach reduces the variance and improves learning stability compared to standalone policy gradients. Then, researchers introduced replay buffers, asynchronous training, trust-region constraints, and maximum-entropy objectives.

Today, actor–critic algorithms like PPO and SAC are often the first choice for continuous control tasks and have seen widespread deployment in both simulated benchmarks and real-world systems (e.g., robotic manipulation, drone flight, and industrial automation). Whether one prioritizes parallelism and speed (A3C), stable and large-step updates (PPO or TRPO), or robust exploration (SAC), modern actor–critic methods offer robust and versatile frameworks capable of learning complex, high-dimensional policies with relative ease.

6. Model-Based Reinforcement Learning

Another set of RL methods that evolved in parallel involves model-based approaches that explicitly or implicitly learn a representation of the environment's transition dynamics and reward function, enabling agents to plan and reason about future outcomes without relying solely on direct environment interaction. Compared to the model-free approaches discussed earlier, model-based methods have the potential to be more sample-efficient, making them particularly appealing for data-intensive tasks such as robotics, high-stakes decision-making, and domains where real-world interaction is costly. As shown in Figure 8, model-based methods introduce an additional modeling and planning stage, whereas model-free approaches typically learn value functions and policies directly from environmental experience.

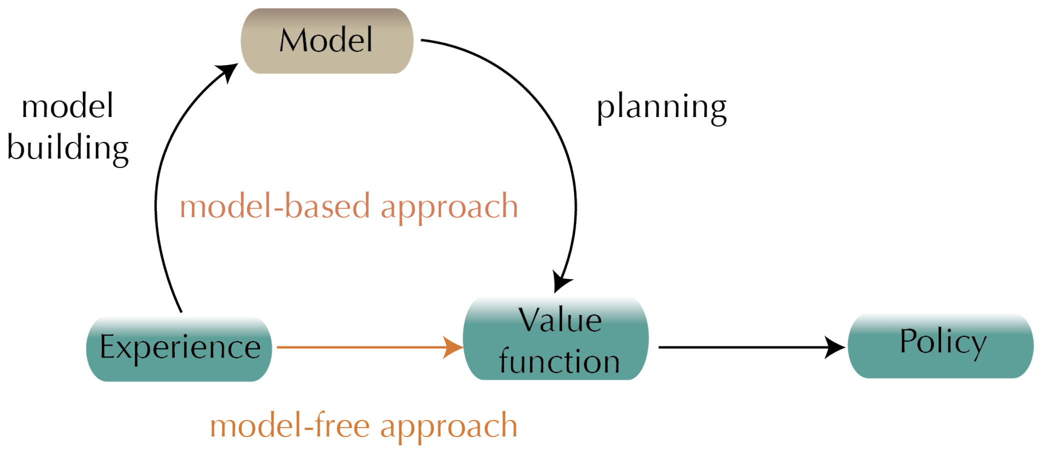

Figure 8. A schematic contrasting model-based and model-free RL [96]. The model-based approach uses observed experience to build an environment model, and then applies planning methods to improve value functions or policies. By comparison, model-free RL updates the value function or policy directly from experience without constructing an explicit model.

6.1. Early Formulations (1980s–1990s)

The conceptual foundations of model-based RL trace back to classical dynamic programming, and optimal control [52,97], where a known model of the environment (often specified via transition probabilities) allows an agent to plan and compute optimal policies. In RL, one of the earliest model-based ideas is Dyna [98], proposed in 1990. Dyna-Q, in particular, combined model-free Q-learning with a learned environment model: the agent stored observed transitions in a model and periodically performed simulated updates to the Q-values (a form of planning). This approach demonstrated how even a crude model could reduce the amount of real-world interaction needed.

Advances in Planning and Exploration (2000s)

Throughout the 2000s, researchers refined model-based algorithms, placing a strong emphasis on exploration strategies and theoretical performance guarantees. R-Max [99] exemplifies this era by assuming optimistic rewards for unvisited states, thereby incentivizing directed exploration to gather more accurate transition data. Similarly, model-based interval estimation (MBIE) [100] addressed sample complexity concerns by exploring states to reduce uncertainty in the learned model, offering rigorous bounds on the number of steps required to achieve near-optimal performance. In parallel, Bayesian approaches introduced priors over transition functions and rewards (e.g., [101]), enabling the agent to manage uncertainty more gracefully and focus data collection on high-value regions of the state space.

During this period, ideas from the control community also continued to influence model-based RL, particularly via optimization-based methods such as the iterative linear quadratic regulator (iLQR) and model predictive control (MPC). iLQR [102,103] extends the classical LQR approach to nonlinear dynamics by iteratively approximating the system around a nominal trajectory, and then solving a sequence of local LQR problems to refine the control law. This trajectory optimization philosophy can be combined with RL to guide policy search with locally optimal controls. MPC [104] builds on a similar principle but solves a finite-horizon optimal control problem at each timestep, leveraging a model to predict future states and adjust actions accordingly. Although iLQR and MPC originated in control theory, they share core ideas with model-based RL by using an explicit model of dynamics for planning or trajectory updates.

Model-based RL thus gained popularity for smaller-scale or well-structured tasks where accurate models could be derived from limited data or expert knowledge. This foundation laid the basis for subsequent techniques aiming to integrate robust exploration with model fidelity using deep network approaches.

6.2. Deep Learning in Model-Based Reinforcement Learning (2010s)

As deep neural networks gained traction, a natural question arose: *Can we learn high-capacity models for complex, high-dimensional environments?* Early successes in combining deep function approximators with model-based RL included learning approximate forward dynamics in robotic tasks such as predicting the next joint state given the current state and action [105,106]. These learned models then served as simulators for policy optimization, reducing real-world interactions.

A transformative moment came with DeepMind's work on board games and planning:

- AlphaGo [3]: Combined Monte Carlo Tree Search (MCTS) with deep neural networks for policy and value approximations. While the game of Go was fully known, that is, there are no unknown dynamics, AlphaGo's planning paradigm influenced subsequent model-based research.
- AlphaZero [107]: Generalized the AlphaGo approach to chess and shogi, emphasizing how powerful planning with deep policy and value networks can produce superhuman play from scratch.
- MuZero [48]: A landmark algorithm that learned a latent-space model of the environment without explicit prior knowledge of the rules. MuZero showed that an agent could discover its own compact representation of game states and dynamics, achieving strong performance in Go, chess, shogi, and Atari games.

MuZero's success emphasized the potential of learning internal models even when the ground-truth rules are not directly accessible.

6.3. Hybrid Approaches (Late 2010s–Present)

Recent work has explored more flexible or approximate models for real-world settings. For instance, ensemble methods [108,109] maintain multiple learned dynamics models to quantify uncertainty, enabling robust planning under imperfect dynamics. World models [47] train latent-space representations of the environment and roll out simulated futures (e.g., Dreamer [110]), thereby reducing the interactions needed in reality. Meanwhile, hybrid/residual methods [111] combine partial analytic models (e.g., known robot kinematics) with learned residual terms to capture unmodeled complexities. These advances tackle challenges such as compounding model errors, high-dimensional observations (e.g., images), and partial observability, pushing model-based RL toward more practical and scalable solutions.

Despite the algorithmic complexity, model-based RL can be significantly more sample-efficient than purely model-free approaches, especially when environmental interactions are expensive or time-consuming as in robotics, healthcare, or finance. By learning or having access to a model of transition dynamics and reward, the agent can plan, predict outcomes, and update its policy in simulation rather than interacting solely with the real environment at every step.

6.4. Approaches to Model Learning

A typical model-based RL pipeline begins with model learning, where the agent gathers environmental data to approximate the transition function $\hat{P}(s' \mid s, a)$ and the reward function $\hat{R}(s, a)$. Neural networks, Gaussian processes, or other regressors can be used to capture these dynamics [101,108]. Once a model is in place, the agent applies planning algorithms such as MCTS [3], Dyna [98], or trajectory optimization to iteratively refine its policy in simulation, rather than relying exclusively on real-environment rollouts.

Methods like iLQR and MPC can also be integrated here; the learned model is used to approximate future trajectories and identify locally optimal controls, reducing the need for exhaustive environment interactions.

Finally, the agent undergoes continuous execution and refinement, interacting with the real environment to collect fresh data and improve both the learned model and the policy itself. This cyclical process of learning, planning, and real-world testing offers a more sample-efficient alternative to purely model-free approaches in many scenarios.

6.5. Main Challenges

Although model-based RL confers several advantages, it also involves important trade-offs. First, model accuracy vs. planning horizon remains critical since even small discrepancies in the learned dynamics can compound dramatically over extended rollouts, necessitating robust model learning [112]. Second, the complexity and overhead of maintaining and updating a high-fidelity model in high-dimensional spaces can exceed that of purely model-free methods, leading to increased computational costs. Finally, generalization and transfer benefits theoretically allow a good model to be reused across similar tasks, but real-world deployments often face dynamic shifts that degrade performance if the model is not adequately adapted [113]. Despite these challenges, model-based strategies remain highly active in RL research, driven by their potential to reduce sample complexity and facilitate explicit planning. From superhuman board game performance (AlphaZero, MuZero) to efficient robotic policy learning (PETS, Dreamer), model-based RL continues to push the boundaries of reinforcement learning in complex real-world domains.

7. Other Branches: Evolutionary Strategies and Hierarchical RL

7.1. Evolutionary Strategies (ES)

Evolutionary strategies (ESs) arose from early research in evolutionary computation during the 1960s and 1970s, largely driven by work on biologically inspired optimization [114,115]. These early studies established the idea that a population of candidate solutions could be iteratively improved by applying variation operators such as mutation and recombination, then selecting and propagating those individuals that performed best under a given fitness metric. Unlike gradient-based RL methods, evolutionary strategies do not rely on differentiable objective functions. Instead, they treat the policy parameters as black-box variables to be sampled and evaluated directly in the environment.

By the early 2000s, the covariance matrix adaptation evolution strategy (CMA-ES) emerged as one of the most robust ES variants [116], adapting a multivariate Gaussian distribution to guide search in high-dimensional parameter spaces. More recently, large-scale ES implementations [117] demonstrated competitive performance with deep RL algorithms on domains like Atari by leveraging hundreds or thousands of parallel CPUs or cloud instances. Such scalability stems from the fact that each individual in the ES population can be evaluated independently, sidestepping the need to backpropagate errors through a network. This parallelism makes ES compelling for tasks in which gradients are sparse or non-differentiable, such as evolving neural network controllers for robot locomotion [118]. However, ES methods typically exhibit lower sample efficiency compared to gradient-based RL, often requiring many evaluations to converge in complex, high-dimensional domains. In practice, researchers and practitioners employ ES in scenarios where environment simulations can be easily parallelized, or where rigid assumptions about differentiability cannot be made.

7.2. Hierarchical Reinforcement Learning (HRL)

Hierarchical RL (HRL) has its roots in the 1990s, particularly in the FeUdal RL concept proposed by Dayan and Hinton [119], which introduced higher-level managers issuing subgoals to lower-level workers. This stratified approach recognized that complex tasks could be tackled more efficiently by decomposing long-horizon problems into smaller, more tractable pieces. Over time, researchers refined hierarchical notions into the options framework [120], allowing agents to learn and reuse temporally extended actions or sub-policies.

Contemporary HRL methods further this tradition by integrating hierarchical decomposition with modern deep RL. One influential example is the option-critic architecture [121], which learns both intra-option policies that define how an agent acts when a particular sub-policy is active and termination functions that dictate when to switch from one option to another. This design facilitates long-range planning and targeted exploration, as the agent can sequence learned behaviors like "navigate corridor", "open door", or "pick up object". In practice, HRL techniques have been effective in multi-stage decision-making tasks such as Minecraft resource collection and 3D navigation [122], as well as robot manipulation tasks where each sub-policy handles a distinct skill [123]. Another line of research, exemplified by FeUdal networks [124], explicitly implements hierarchical managers and workers, with the manager producing goals in an embedding space that the worker is trained to fulfill. Such structuring can significantly reduce the effective horizon of the task, leading to faster convergence and improved exploration.

Despite the substantial promise of HRL, the approach faces key challenges in option discovery, credit assignment across multiple policy levels, and increased computational overhead from managing multiple policy and termination functions. Nonetheless, the potential for more interpretable, reusable, and structured skill sets ensures that HRL continues to be a vibrant area of RL research, especially for tasks characterized by compositional sub-goals and extended planning horizons.

8. Large-Scale Specialized RL Systems

Although the preceding sections covered a variety of methods—from model-free algorithms to actor–critic hybrids, evolutionary strategies, hierarchical RL, and model-based approaches—a separate trajectory of progress has unfolded in large-scale specialized systems. These systems combine massive computational resources, significant engineering effort, and domain-specific insight to achieve superhuman or near-superhuman performance in complex environments. While some leverage model-based ideas (e.g., Monte Carlo Tree Search) and others rely more on self-play and policy gradient methods, all share a common emphasis on scaling and careful problem-specific tuning.

8.1. AlphaStar and OpenAI Five (2018–2019)

Building on the success of AlphaGo (2016) and AlphaZero (2017), DeepMind's AlphaStar [125] tackled the real-time strategy game StarCraft II, a domain characterized by hidden information, complex multi-agent dynamics, and a vast action space. AlphaStar employed a multi-agent training framework in which numerous policies were trained in parallel via self-play, each specializing in different strategies before being distilled into a final agent. This approach combined aspects of policy gradient methods, imitation learning, and evolutionary concepts (e.g., population-based training), ultimately achieving grandmaster-level play against professional human gamers.

Contemporaneously, OpenAI developed OpenAI Five [4] to tackle the multiplayer online battle arena game Dota 2. Similar to AlphaStar, OpenAI Five relied on large-scale self-play with massive computing resources, training multiple agents that learned complementary team strategies. A major engineering feat was adapting policy gradient algorithms (based primarily on proximal policy optimization, PPO) and multi-agent RL to the high-dimensional, partially observable environment. By mid-2019, OpenAI Five defeated champion-level human teams, demonstrating the potential of large-scale RL to master extremely complex tasks with long time horizons, rich state–action spaces, and significant cooperative elements.

8.2. AlphaFold (2020–2021)

While early large-scale systems focused on board games and esports, DeepMind's AlphaFold [126] pioneered a leap into structural biology. AlphaFold is not purely a reinforcement learning system; however, it incorporates inspiration from RL-like paradigms (notably self-play and iterative optimization) in conjunction with large-scale deep networks and generative modeling. First introduced in the CASP13 competition (2018), an improved version released in 2020–2021 demonstrated near-experimental precision in protein folding prediction. Although much of AlphaFold's inference process is based on deep neural architectures and specialized losses, the iterative refinement stage draws conceptually from RL research, where the model incrementally folds the protein, guided by feedback from a predicted structure quality metric. This success emphasized how RL-inspired techniques, when combined with domain-specific innovations and large computing power, could tackle long-standing scientific challenges outside of games.

8.3. Lessons and Ongoing Developments

Across AlphaStar, OpenAI Five, and AlphaFold, we see a consistent pattern: massive parallelization, domain-focused engineering, and hybrid algorithms that merge self-play, policy gradient, model-based rollouts, and large-scale data pipelines. These systems highlight the versatility of reinforcement learning when it is carefully adapted to specific domains and scaled beyond academic benchmarks. However, they also illustrate the resource-intensive nature of achieving top-tier performance, since each project required extensive computational infrastructure and expert tuning.

Current directions extend these insights to other fields, including robotics, healthcare, and multi-agent coordination, integrating specialized environments, domain knowledge, and large-scale distributed training. While not every RL application can afford the scale and specialization of AlphaStar or AlphaFold, these examples show how far the boundaries of reinforcement learning can be pushed with sufficient resources and targeted algorithmic design. They also set the stage for future breakthroughs, where RL might tackle open-ended domains like scientific discovery, automated design, or general multi-agent interactions, provided the algorithms, computing power, and collaboration frameworks continue to evolve.

9. When to Use Each Approach

After exploring a variety of deep reinforcement learning (RL) techniques, ranging from value-based methods (such as DQN and SARSA) to policy-based approaches (like REINFORCE), as well as actor–critic hybrids, model-based algorithms, evolutionary strategies, hierarchical frameworks, large-scale specialized systems, and alignment methods, we now summarize their usage scenarios, advantages, and limitations. Figure 9 categorizes these approaches into value-based, policy-based, model-based, and their combinations, and Table 1 provides a concise reference for when each approach is most appropriate, along with its key benefits and drawbacks.

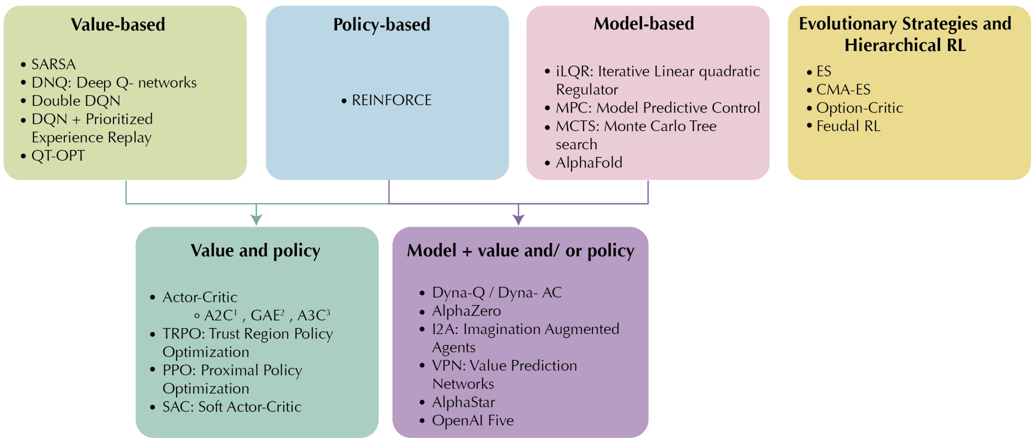
Value-based
• SARSA
• DNQ: Deep Q- networks
• Double DQN
• DQN + Prioritized Experience Replay
• QT-OPT
Policy-based
• REINFORCE
Model-based
• iLQR: Iterative Linear quadratic Regulator
• MPC: Model Predictive Control
• MCTS: Monte Carlo Tree search
• AlphaFold
Evolutionary Strategies and Hierarchical RL
• ES
• CMA-ES
• Option-Critic
• Feudal RL
Value and policy
• Actor-Critic
∘ A2C¹ , GAE² , A3C³
• TRPO: Trust Region Policy Optimization
• PPO: Proximal Policy Optimization
• SAC: Soft Actor-Critic
Model + value and/ or policy
• Dyna-Q / Dyna- AC
• AlphaZero
• I2A: Imagination Augmented Agents
• VPN: Value Prediction Networks
• AlphaStar
• OpenAI Five

Figure 9. A high-level taxonomy of RL methods, illustrating how approaches can be categorized into value-based, policy-based, or model-based, along with combined methods. Inspired by [127].

A high-level perspective can be broken down into the following main families:

- **Value-based (e.g., DQN and variants)**
  - Core Idea: Maintain explicit Q-values (or state-value functions).
  - Best For: Discrete action spaces (e.g., Atari, simple robotics with discrete actions), domains with moderate dimensionality, and large amounts of experience.
  - Strengths:
    - Typically straightforward to understand and implement (Q-learning or TD-learning backbone).
    - Proven success in discrete control tasks (e.g., classic Atari suite).
    - Robust theoretical grounding and extensive extensions (double DQN, Dueling DQN, Rainbow).
  - Weaknesses:
    - Struggles with continuous or very high-dimensional action spaces (requires discretization or specialized methods).
    - Off-policy nature can degrade performance if replay and exploration strategies are not carefully tuned.
  - Example use-cases: Video game playing (Atari), basic navigation tasks with limited action sets, or simple resource-allocation systems with discrete decisions.
- **Policy-based (e.g., REINFORCE)**
  - Core idea: Parameterize the policy directly and optimize it via gradient ascent on expected returns.
  - Best for: Continuous action settings, tasks requiring smooth exploration (e.g., robotics with direct torque control).
  - Strengths:
    - Naturally handles continuous or high-dimensional action spaces.
    - Offers a simpler pathway for stochastic exploration (sampling from a parameterized distribution).
  - Weaknesses:
    - High variance in gradient estimates, especially in pure Monte Carlo approaches.
    - Potentially sample-inefficient if updates rely on entire episodes (as in plain REINFORCE).
  - Example use-cases: Simple continuous-control benchmarks (e.g., mountain-car continuous), or low-DOF robotic arms.

- **Actor–critic (e.g., PPO, SAC, TRPO, DDPG)**
  - Core idea: Combine a learned value function (critic) with a parameterized policy (actor), reducing variance compared to pure policy-based methods.
  - Strengths:
    - Broad applicability in both discrete and continuous domains.
    - More sample-efficient than pure policy-based approaches.
    - Critic provides lower-variance gradient updates, while the actor optimizes the policy directly.
  - Key variants and comparisons:
    - **PPO (proximal policy optimization):**
      - On-policy algorithm using clipping to prevent catastrophic updates.
      - Achieves good performance in continuous control (e.g., MuJoCo) and multi-agent tasks.
      - Requires careful tuning of clipping hyperparameters and can be slower per iteration than simpler methods.
    - **SAC (soft actor–critic):**
      - Off-policy maximum-entropy approach encouraging broader exploration.
      - Often achieves state-of-the-art in continuous control tasks.
      - Slightly more complex (two Q-networks, temperature parameter) and memory-heavy replay buffer.
    - **DDPG (deep deterministic policy gradient):**
      - Off-policy algorithm for high-dimensional continuous action spaces.
      - Conceptually simpler but can be brittle in practice; sensitive to hyperparameters.
  - Weaknesses:
    - More complex to implement and tune than simpler value-based methods.
    - Large memory overhead in off-policy variants (replay buffers).
  - Example use-cases: Robotics manipulation, simulated locomotion, parallelizable tasks (e.g., A3C), or high-dimensional continuous control.
- **Model-based RL (e.g., iLQR, MPC, MCTS, MuZero)**
  - Core idea: Learn or use an environment model to plan or generate synthetic experiences.
  - Strengths:
    - Potentially much more sample-efficient, fewer real-world interactions required.
    - Can explicitly plan over future trajectories, leading to better long-horizon performance.
  - Weaknesses:
    - Model inaccuracies can compound over multiple prediction steps.
    - Potentially high computational overhead for model learning and planning at each time step.
  - Example use-cases: Robotics tasks with limited real-world data, board games with well-defined rules ([107]), and scheduling/resource management.
- **Evolutionary strategies (ES, CMA-ES)**
  - Core idea: Optimize policy parameters via population-based black-box optimization.
  - Strengths:
    - Easily parallelized; does not rely on gradient backpropagation.

        * Handles highly stochastic or rugged objective landscapes reasonably well.
    - Weaknesses:
        * Often requires many environment evaluations; can be sample-inefficient.
        * Scalability is heavily dependent on available parallel hardware.
    - Example use-cases: Non-differentiable domains or tasks with sparse/deceptive rewards, large-scale parallel CPU clusters.
- **Hierarchical RL (e.g., option-critic, FeUdal Networks)**
    - Core idea: Decompose tasks into subtasks or reusable skills for more efficient exploration and planning.
    - Strengths:
        * Addresses long-horizon tasks with sparse rewards.
        * Sub-policies (or "options") can be reused across tasks, potentially improving sample efficiency.
    - Weaknesses:
        * Requires discovering or defining useful subgoals, which is non-trivial.
        * Multi-level credit assignments can complicate training.
    - Example use-cases: Complex navigation, multi-stage robotic tasks (e.g., pick-and-place then door-opening), or open-world games such as Minecraft.
- **Large-scale specialized systems (e.g., AlphaStar, OpenAI Five, AlphaFold)**
    - Core idea: Combine massive computing resources with specialized domain knowledge (e.g., self-play, problem-specific architectures).
    - Strengths:
        * Can achieve superhuman or near-superhuman performance in complex domains.
        * Demonstrates upper-bound capabilities of RL when scale and engineering are not limiting factors.
    - Weaknesses:
        * Extremely resource-intensive and often heavily engineered for narrow tasks.
        * Limited direct generalization outside their specialized domain.
- **Alignment methods (e.g., RLHF)**
    - Core idea: Incorporate human-provided preference labels into the reward function, iteratively refining a policy to align with human values.
    - Strengths:
        * Directly targets user satisfaction, ethical considerations, or task-specific constraints.
        * Successful in refining large language models ([22,90]).
    - Weaknesses:
        * Requires substantial human supervision or feedback data.
        * Designing robust reward models is challenging; incomplete feedback can lead to unintended behaviors.

Table 1. Representative deep RL approaches with typical usage and trade-offs.

Type	Method	When to Use	Advantages	Limitations
Value-based (Off-policy)	SARSA, DQN [32], Double DQN [44], DQN + PER [72], QT-Opt [128]	Discrete actions, moderate state complexity, large data	Straightforward concept, proven success (e.g., Atari)	Not suited for continuous actions, tuning-sensitive
Policy-based (On-policy)	REINFORCE [76]	Continuous actions, smooth exploration, direct policy learning	Natural for continuous simpler exploration	High variance, sample-inefficient, delayed updates
Combined (Value + Policy)	Actor–critic (A2C [34], GAE [80], A3C [34]), TRPO [81], PPO [82], SAC [83]	Wide range of tasks, discrete/continuous, real-world complexity	Stable updates, reduced variance, broad applicability	Complex to implement, hyperparameter sensitivity, memory-intensive
Model-based	iLQR [102,103], MPC [104], MCTS [3], i.e., environment models + planning	Expensive interactions, limited data, robotics/control	Potentially fewer samples needed, can plan ahead	Requires accurate models, error compounding, added complexity
Combined (Model + Value/ Policy)	Dyna-Q/Dyna-AC [98], AlphaZero [107], I2A: Imagination Augmented Agents [129], VPN: Value Prediction Networks [130]	Tasks benefiting from partial models or partial planning	Mixes model-based planning and direct policy/value updates	Complexity in model learning + policy optimization
Evolutionary strategies	ES [117], CMA-ES [116]	Highly parallel tasks, non-differentiable, noisy envs	Easily parallelized, fewer assumptions	Lower sample efficiency, slower convergence
Hierarchical RL	Option-Critic [121], Feudal RL [119]	Complex tasks, subgoals, sparse rewards	Skill reuse, improved exploration, structured policies	Subgoal discovery, complex credit assignment
Large-scale specialized	AlphaStar [125], OpenAI Five [4], AlphaFold [126]	Games with multi-agent or domain-specific biology tasks	Superhuman results, domain expertise, powerful computing	Huge data & computing, domain constraints, specialized pipelines
Alignment	RLHF [21,22,90]	Settings where human feedback is crucial (LLM alignment)	Directly optimizes toward human preferences	Requires manual feedback data, reward modeling

10. Discussion: Current and Future Trends in Deep Reinforcement Learning

Deep reinforcement learning has undergone a remarkable evolution, yet several open issues persist. One major concern is sample efficiency, given that many popular algorithms—from value-based methods like DQN to actor–critic approaches such as PPO or SAC—often require millions of environmental interactions to converge [32,35]. Researchers have explored various remedies, including curiosity-driven exploration [131,132], count-based techniques [133], and intrinsic motivation signals that guide agents toward novel or uncertain states [134]. Additionally, off-policy learning with carefully curated replay buffers can improve data utilization and reduce the amount of real-world experience needed [35,70]. Transfer learning and multi-task paradigms enable agents to leverage knowledge from previously solved tasks, thereby accelerating adaptation to new domains [135,136].

Despite these approaches, sample efficiency remains a key bottleneck, particularly in safety-critical domains like healthcare or finance, where frequent exploration can be infeasible or risky. One promising direction is offline RL [137], which aims to learn high-performing policies exclusively from previously collected data. Although offline RL methods can circumvent excessive exploration, they face challenges related to dataset quality, distribution shift, and conservative policy updates. Model-based RL (e.g., DreamerV3 [138]) also offers better sample efficiency by predicting future states via learned dynamics models. However, model inaccuracies may compound over multi-step predictions, demanding robust uncertainty estimates or ensemble methods [108] to mitigate cascading errors. An open problem here is developing adaptive model-based strategies that seamlessly handle high-dimensional inputs (e.g., raw images) while maintaining tight uncertainty bounds to prevent catastrophic rollouts.

As RL transitions into safety-critical and high-stakes domains such as healthcare, robotics, and finance, concerns about safety, robustness, and interpretability become critical. Safe RL typically involves formulating additional constraints (e.g., maximum velocity or collision avoidance) to prevent catastrophic actions, ensuring that the agent avoids irreparable harm even during exploration [139,140]. While many safe RL frameworks exist, such as constrained policy optimization [140] or shielded RL [141], they can be conservative, causing slow convergence or under-exploration. Balancing adequate exploration with strict safety criteria thus remains an open challenge. Additionally, for real-world applications, robust RL seeks to maintain performance under environment uncertainties or adversarial perturbations [142,143], yet few methods offer formal guarantees for high-dimensional tasks. Future directions include integrating robust control theory with deep RL, as well as systematically quantifying uncertainty for out-of-distribution states.

Interpretability (or explainability) is another central concern, as black-box neural policies can be difficult to trust in mission-critical scenarios. Explainable RL research explores a wide range of methods, from policy distillation into decision trees [144] to saliency-based visual explanations [145]. Yet there is no consensus on which interpretability framework is best for different application domains. Moreover, interpretability methods can introduce trade-offs with performance or require special architectures (e.g., attention mechanisms) that might slow down training. A vital open problem is developing post hoc and inherent interpretability approaches that scale to complex policies (e.g., large language models in RL loops) without substantially hampering performance. Rigorous metrics to evaluate the quality of an explanation are also lacking.

Recent years have seen a surge in large-scale, multi-task RL approaches that blur boundaries between supervised, unsupervised, and reinforcement learning. Gato [146], a single transformer-based generalist agent, offered one of the first demonstrations that a single neural network can play Atari games, caption images, carry on dialogues, and even control real robot arms—all by switching context. Although Gato did not surpass specialized models on every task, it showed that multi-modal, multi-task RL is feasible, marking an early step toward more general-purpose agents capable of operating across diverse data domains. However, a critical open challenge is to ensure positive transfer: multi-task systems can sometimes underperform specialized single-task agents due to task interference or suboptimal data mixing. Methods such as multi-headed architectures [136] and task-specific adapters [147] help alleviate interference, yet a deeper understanding of how to automatically identify, separate, or merge skills across tasks is still needed.

In parallel, RL-driven algorithm discovery has captured attention. AlphaTensor [148] and AlphaDev [149] push beyond games to optimize core computational tasks. AlphaTensor discovered new, faster methods for matrix multiplication, solving a long-standing open math problem, while AlphaDev uncovered sorting algorithms that outperform decades of human fine-tuning. Notably, AlphaDev's solution was integrated into a major C++ standard library, highlighting how RL can drive innovation in software and computational science. Despite these successes, bridging algorithm discovery with mainstream applications requires making RL approaches interpretable and verifiable so that human experts can trust and adopt automatically discovered methods.

Multi-agent RL has also advanced. CICERO [150] from Meta AI achieved human-level performance in the negotiation-heavy board game Diplomacy by combining a language model for dialogue and an RL policy for strategic planning. CICERO could forge alliances, persuade players, and outmaneuver human opponents, all without revealing its AI identity. This result highlights the potential of combined language-and-strategy RL methods to tackle complex human-centric coordination problems, from business negotiations to multi-party resource allocation. Yet multi-agent systems complicate safety and interpretability concerns even further, as emergent behaviors might be difficult to anticipate or debug. Ensuring cooperative or safe outcomes may require novel mechanism design approaches that define protocols for agent interaction, an area that remains underexplored.

Model-based RL continues to gain traction, especially for long-horizon tasks. DreamerV3 [138] showed that a single world-model-based algorithm could achieve state-of-the-art performance on over 150 diverse tasks with minimal tuning, even solving the historically challenging diamond quest in Minecraft using sparse rewards. This milestone emphasizes how predictive models of the environment can vastly improve sample efficiency and generalization, moving RL closer to a truly off-the-shelf paradigm for complex real-world problems. However, these methods require robust mechanisms to handle model bias and compounding errors, and relatively few works provide formal guarantees on model accuracy or stability in high-dimensional, partially observable tasks.

10.1. Advances in Robotics and Embodied AI

Robotics is emerging as a key beneficiary of modern RL methods, particularly where large models, simulations, and hierarchical learning intersect. RoboCat [151] took a big step toward general-purpose robotic control by learning to operate multiple robot arm platforms via a self-improvement loop. Starting from a broad, multi-task dataset, RoboCat can learn new tasks with only a handful of demonstrations, generate synthetic experience in simulation, and integrate that data to strengthen its policy. This cycle led to rapid adaptation across different hardware and manipulations, highlighting how transferable, iterative RL can streamline training for new robotic tasks. Yet these paradigms can falter if the *sim-to-real gap* is large or if certain real-world conditions (e.g., lighting, friction) deviate significantly from the simulator. Achieving robust domain adaptation and safety in real robotic environments remains a key open problem.

Incorporating language-based reasoning has further expanded robotic capabilities. PaLM-SayCan [152] combined a large language model with an RL-derived affordance function to execute human commands in a kitchen setting. By grounding high-level requests (e.g., cleaning up a spill) in an action-value system, the robot could plan multi-step sequences with high success rates—the first instance of a real robot fluidly mixing language understanding with RL-based action feasibility. This development paves the way for more natural human-robot interaction in home and service environments, where instructions are given verbally or textually rather than by specialized code. Nonetheless, generating accurate symbolic reasoning or guaranteeing correct outcomes remains difficult; bridging linguistic abstractions with low-level policy controls is a rich area for future investigation.

Even harder challenges in robot manipulation, such as bi-manual coordination and fine motor skills, are becoming tractable. DeepMind's ALOHA Unleashed and DemoStart [153] enabled dexterous two-handed tasks (tying shoelaces, assembling gears) and multi-finger control with minimal simulation-to-reality gap. These methods use imitation learning and RL in carefully designed curricula, drastically reducing the need for huge real-world datasets while achieving near-human levels of manipulation skills. This points to a future where robots can learn a wide array of everyday dexterous tasks—from household assistance to intricate factory assembly—largely through simulation, domain randomization, and small amounts of real demonstration. However, guaranteeing safe exploration in physically interactive tasks remains nontrivial; many open questions revolve around sensor reliability, collision avoidance, and real-time adaptation when unexpected obstacles or failures occur.

10.2. Emerging Trends and Future Prospects

Beyond these individual breakthroughs, several major trends foreshadow how RL research may evolve. Systems like Genie 2 [154] promise an endless curriculum of procedurally generated worlds, removing the need for meticulously crafted training scenarios. This approach can dramatically improve the robustness of RL agents and facilitate open-ended exploration, yet ensuring meaningful or curriculum-aligned tasks automatically remains an active research direction.

From Voyager [155] in Minecraft to PaLM-SayCan in robotics, combining large language models with RL policies has proven powerful for complex planning, exploration, and communication. Future RL agents may rely on language reasoning even more deeply, bridging symbolic and continuous decision-making. Meanwhile, techniques for explainable policy grounding (e.g., generating textual rationales for actions) could help address interpretability demands in safety-critical deployments.

AlphaDev's discovery of novel sorting algorithms illustrates RL's potential in program optimization and mathematical problem-solving. We can expect further strides in "AI-discovered algorithms" for a wide range of applications, from compilers to advanced cryptography. However, the ability to formally verify or certify correctness and efficiency remains a pressing challenge, especially if the discovered techniques diverge substantially from conventional human-designed algorithms.

As RL finds real-world adoption, advanced forms of reward shaping and alignment will be crucial for guiding agents toward socially desirable or human-compatible outcomes. Techniques like RL from human feedback (RLHF) [90] or cooperative multi-agent objectives aim to incorporate user preferences and domain expertise more directly. Nevertheless, reward misspecification, reward hacking, and ambiguous human feedback persist as open research problems, underscoring the need for robust value alignment mechanisms.

Finally, the success of DreamerV3 and multi-task systems like Gato underscores a demand for "one-size-fits-all" algorithms capable of tackling diverse tasks with minimal fine-tuning. Advances in distributed, model-based, and hierarchical RL could bring us closer to developing generally capable agents. Yet questions regarding catastrophic forgetting, negative task transfer, and computational scalability remain unresolved.

Overall, these emerging research directions reflect the ongoing transformation of RL into a discipline that intersects with language modeling, robotics, software engineering, cognitive science, and more. Whether optimizing algorithms, negotiating with humans, or manipulating physical objects with dexterous skill, reinforcement learning is poised to tackle increasingly open-ended challenges, guided by new methods for efficient exploration, large-scale training, and principled safety. As these trends continue, we can anticipate RL methods becoming more pervasive in real-world applications, from everyday human-robot collaboration to fundamental scientific discovery.

10.3. Open Research Problems

While the aforementioned advances push RL forward, the following specific challenges remain open avenues for future investigation:

- Scalable safe exploration: Designing algorithms that respect strict safety constraints throughout training, while still exploring enough to discover optimal or near-optimal behaviors.
- Interpretable policy architectures: Developing intrinsic or post hoc interpretability methods that scale to high-dimensional policies (e.g., transformers), without significantly degrading performance.
- Robust domain adaptation: Minimizing the sim-to-real gap through adaptive models, ensemble methods, or uncertainty-aware policy updates that handle novel states gracefully.
- Multi-task and continual learning: Ensuring positive task transfer, reducing catastrophic forgetting, and automatically discovering shared representations for large sets of tasks or changing environments.
- Unified theoretical frameworks: Extending existing PAC-MDP or policy gradient theories to encompass large function approximators and partial observability, providing stronger convergence or sample complexity guarantees in deep settings.

11. Conclusions

Deep reinforcement learning has evolved dramatically from the early days of tabular Q-learning and linear function approximation to the cutting-edge methods that integrate deep networks, advanced optimization techniques, and, increasingly, model-based planning. The chronological perspective reveals how breakthroughs such as deep Q-networks paved the way for robust, generalizable methods in both discrete and continuous action spaces, spawning a rich ecosystem of algorithms including policy gradient, actor–critic, and distributional approaches. At the forefront, policy gradient methods like PPO, SAC, and TRPO have become mainstays for handling complex environments with continuous controls, while innovations in model-based RL, hierarchical decomposition, and maximum entropy strategies continue to push the envelope for tasks requiring efficient exploration, sample efficiency, and scalability.

For newcomers, understanding the fundamental building blocks—Markov decision processes, value functions, policy gradients, and the design principles of deep neural networks—paves the path to engaging deeply with cutting-edge research. Each class of algorithms has its own set of strengths, challenges, and natural domains of application, whether it be large-scale discrete control problems (DQN variants) or continuous robotics tasks (PPO, SAC). The field continues to wrestle with limitations in sample efficiency, interpretability, and safety, but ongoing research seeks to address these gaps through improved exploration strategies, robust algorithm designs, and cross-pollination with fields like generative modeling and neuroscience.

Looking forward, we can anticipate an increased emphasis on multi-task learning, meta-learning, and hierarchical architectures to tackle truly open-ended problems. As RL systems transition from games and simulations to real-world applications, questions of reliability, computational resources, and alignment with human values will take center stage. The future of deep RL will likely revolve around seamlessly integrating powerful function approximators with principled strategies for exploration, planning, and generalization, marking the next phase in the quest toward intelligent, autonomous systems that can learn and adapt to a complex, ever-changing world.

Funding: This research was funded by Instituto Politecnico Nacional grant number 20250165.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Acknowledgments: During the preparation of this manuscript/study, the author used grammar assistant tools to improve the grammar, clarity, and overall readability of the manuscript, and AI to assist with the wording and proofreading. The author has reviewed and edited the output and takes full responsibility for the content of this publication.

Conflicts of Interest: The author declares no conflicts of interest.

Glossary of Key Terms

Action	A decision or move the agent takes in a given state. Examples: moving a robot arm, pressing a button, or playing a particular card in a card game.
Actor	In actor–critic algorithms, the component that represents the policy, mapping states to actions.
Actor–critic	A class of algorithms combining policy-based (actor) and value-based (critic) methods. The critic estimates value functions to guide the actor's policy updates.
Advantage	A measure of how much better (or worse) a specific action is compared to the average action at a given state, typically defined as $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$.
AlphaGo/AlphaZero/ MuZero	Series of algorithms by DeepMind using deep networks and sophisticated tree search/planning to achieve superhuman performance in board games such as Go, chess, and shogi (MuZero also handles Atari games by learning a model).
Baseline	In policy gradient methods, a function (often the value function) subtracted from the return to reduce variance in gradient updates without introducing bias.
Critic	In actor–critic algorithms, the component that estimates value functions or Q-values, providing a training signal to the actor.
Deep deterministic policy gradient (DDPG)	An off-policy actor–critic method that handles continuous action spaces by learning a deterministic policy and using a separate Q-network as the critic.
Deep Q-network (DQN)	A pioneering deep RL method that approximates the Q-value function with a convolutional neural network, enabling learning directly from raw pixels (notably in Atari games). Introduced experience replay and target networks.
Discount factor (γ)	A scalar in $[0,1]$ determining how future rewards are weighted relative to immediate rewards. A value close to 1 places more emphasis on future returns.
Distributional RL	A family of algorithms (e.g., C51, QR-DQN) that learn the distribution of returns instead of just the mean, leading to richer training signals and sometimes more stable policies.

Eligibility traces	A mechanism (e.g., in TD(λ) or SARSA(λ)) to assign credit to states and actions that preceded a reward within an episode, helping bridge short- and long-term outcomes.
Entropy	A measure of randomness in a policy's action selection. Maximum entropy algorithms (e.g., SAC) seek to maximize both reward and policy entropy to encourage exploration.
Environment	The system or domain in which the agent operates. It defines states, transitions, and rewards.
Experience replay	A technique (first widely used in DQN) where past transitions (state, action, reward, next state) are stored in a buffer and sampled (often randomly) to break correlation and stabilize learning.
Exploration–exploitation trade-off	The fundamental dilemma in RL of choosing between trying actions that yield known rewards (exploitation) vs. trying actions to discover potentially better rewards (exploration).
Episode	A sequence of interactions from an initial state to a terminal or stopping condition in an RL task. After termination, the environment resets.
Hierarchical reinforcement learning (HRL)	Methods that break down tasks into subtasks or use "options" (temporally extended actions), enabling long-horizon credit assignment and skill reuse.
Markov decision process (MDP)	A formalism for RL tasks, defined by the tuple (S, A, P, R, γ). It assumes the Markov property: the next state and reward depend only on the current state and action.
Model-based RL	Approaches that learn or use an environment model (transition dynamics, rewards) to plan or simulate futures, often reducing the required real-world interactions.
Model-free RL	Approaches (e.g., Q-learning, DQN, policy gradients) that do not learn an explicit environment model. They directly learn a value function or policy from data.
Off-policy	Learning about a target policy π while following a different behavior policy μ during data collection. Q-learning is off-policy because it learns the greedy policy from data that may be ϵ-greedy or otherwise exploratory.
On-policy	Learning that evaluates or improves the same policy that is used for data collection. SARSA and policy gradient algorithms (e.g., REINFORCE) are on-policy methods.
Policy	A (deterministic or stochastic) mapping from states to actions. In RL, the goal is typically to find an optimal or high-performing policy.
Policy gradient methods	Algorithms like REINFORCE, PPO, TRPO, and SAC that optimize a parameterized policy directly via gradient ascent on expected returns.
Proximal policy optimization (PPO)	A popular on-policy actor–critic algorithm that uses a clipped probability ratio to prevent large destructive policy updates and achieve stable performance in continuous control and discrete tasks.
Q-learning	A foundational off-policy, model-free algorithm that iteratively updates Q-values using the Bellman optimality equation. The policy typically becomes greedy w.r.t. the learned Q-values.
Q-value	The expected return (cumulative discounted reward) for taking action a in state s and then following a particular policy thereafter.
Return	The cumulative discounted reward an agent receives, often starting from a particular time step t, denoted $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$.

Reward	A scalar signal from the environment indicating the immediate desirability of an action's outcome. The agent's objective is to maximize cumulative reward over time.
Sample efficiency	How effectively an algorithm uses collected transitions to learn. Algorithms like SAC and model-based methods often aim to be more sample-efficient, especially important in real-world domains like robotics.
SARSA	An on-policy temporal difference method for learning Q-values, named after the quintuple (S, A, R, S', A') used for updates.
Soft actor–critic (SAC)	An off-policy actor–critic algorithm that maximizes a trade-off between expected return and policy entropy. Especially effective for continuous control tasks.
Temporal difference (TD) error	The difference between a bootstrap estimate of the current value and the previously estimated value. Used to update value functions incrementally.
Trajectory/rollout	A sequence of states, actions, and rewards generated by following a policy in an environment for an episode or a fixed time horizon.
Value function	A function $V^{\pi}(s)$ predicting the expected return from state s when following policy π.
World models	A family of model-based RL approaches that learn compact latent representations of environment dynamics (e.g., using VAEs or recurrent networks) to imagine or plan future rollouts.

References

1. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
2. Morales, M. *Grokking Deep Reinforcement Learning*; Manning Publications: Shelter Island, NY, USA, 2020.
3. Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **2016**, *529*, 484–489. [CrossRef]
4. Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Dębiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; Hesse, C.; et al. Dota 2 with large scale deep reinforcement learning. *arXiv* **2019**, arXiv:1912.06680.
5. Kober, J.; Bagnell, J.A.; Peters, J. Reinforcement learning in robotics: A survey. *Int. J. Robot. Res.* **2013**, *32*, 1238–1274. [CrossRef]
6. Peters, J.; Schaal, S. Reinforcement learning of motor skills with policy gradients. *Neural Netw.* **2008**, *21*, 682–697. [CrossRef]
7. Levine, S.; Finn, C.; Darrell, T.; Abbeel, P. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.* **2016**, *17*, 1–40.
8. Gu, S.; Holly, E.; Lillicrap, T.; Levine, S. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 29 May–3 June 2017; pp. 3389–3396.
9. Kiran, B.R.; Sobh, I.; Talpaert, V.; Mannion, P.; Al Sallab, A.A.; Yogamani, S.; Pérez, P. Deep reinforcement learning for autonomous driving: A survey. *IEEE Trans. Intell. Transp. Syst.* **2021**, *23*, 4909–4926. [CrossRef]
10. Sallab, A.E.; Abdou, M.; Perot, E.; Yogamani, S. Deep reinforcement learning framework for autonomous driving. *arXiv* **2017**, arXiv:1704.02532. [CrossRef]
11. Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; Koltun, V. CARLA: An open urban driving simulator. In Proceedings of the Conference on Robot Learning, Mountain View, CA, USA, 13–15 November 2017; pp. 1–16.
12. Shalev-Shwartz, S.; Shammah, S.; Shashua, A. Safe, multi-agent, reinforcement learning for autonomous driving. *arXiv* **2016**, arXiv:1610.03295.
13. Moody, J.; Saffell, M. Reinforcement learning for trading. *Adv. Neural Inf. Process. Syst.* **1998**, *11*, 917–924.
14. Almahdi, S.; Yang, S.Y. An adaptive portfolio trading system: A risk-return portfolio optimization using recurrent reinforcement learning with expected maximum drawdown. *Expert Syst. Appl.* **2017**, *87*, 267–279. [CrossRef]
15. Jiang, Z.; Liang, J. Cryptocurrency portfolio management with deep reinforcement learning. In Proceedings of the 2017 Intelligent Systems Conference (IntelliSys), London, UK, 7–8 September 2017; pp. 905–913.
16. Beysolow, T., II. Market making via reinforcement learning. In *Applied Reinforcement Learning with Python: With OpenAI Gym, Tensorflow, and Keras*; Apress: New York, NY, USA, 2019; pp. 77–94.
17. Komorowski, M.; Celi, L.A.; Badawi, O.; Gordon, A.C.; Faisal, A.A. The artificial intelligence clinician learns optimal treatment strategies for sepsis in intensive care. *Nat. Med.* **2018**, *24*, 1716–1720. [CrossRef]

18. Raghu, A.; Komorowski, M.; Ahmed, I.; Celi, L.; Szolovits, P.; Ghassemi, M. Deep reinforcement learning for sepsis treatment. *arXiv* **2017**, arXiv:1711.09602.
19. Eckardt, J.N.; Wendt, K.; Bornhaeuser, M.; Middeke, J.M. Reinforcement learning for precision oncology. *Cancers* **2021**, *13*, 4624. [CrossRef]
20. Gottesman, O.; Johansson, F.; Meier, J.; Dent, J.; Lee, D.; Srinivasan, S.; Zhang, L.; Ding, Y.; Wihl, D.; Peng, X.; et al. Evaluating reinforcement learning algorithms in observational health settings. *arXiv* **2018**, arXiv:1805.12298.
21. Christiano, P.F.; Leike, J.; Brown, T.; Martic, M.; Legg, S.; Amodei, D. Deep reinforcement learning from human preferences. *Adv. Neural Inf. Process. Syst.* **2017**, *30*.
22. Ziegler, D.M.; Stiennon, N.; Wu, J.; Brown, T.B.; Radford, A.; Amodei, D.; Christiano, P.; Irving, G. Fine-tuning language models from human preferences. *arXiv* **2019**, arXiv:1909.08593.
23. Stiennon, N.; Ouyang, L.; Wu, J.; Ziegler, D.; Lowe, R.; Voss, C.; Radford, A.; Amodei, D.; Christiano, P.F. Learning to summarize with human feedback. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 3008–3021.
24. Mao, H.; Alizadeh, M.; Menache, I.; Kandula, S. Resource management with deep reinforcement learning. In Proceedings of the 15th ACM Workshop on Hot Topics in Networks, Atlanta, GA, USA, 9–10 November 2016; pp. 50–56.
25. Tesauro, G.; Jong, N.K.; Das, R.; Bennani, M.N. A hybrid reinforcement learning approach to autonomic resource allocation. In Proceedings of the 2006 IEEE International Conference on Autonomic Computing, Dublin, Ireland, 13–16 June 2006; pp. 65–73.
26. Haghshenas, K.; Pahlevan, A.; Zapater, M.; Mohammadi, S.; Atienza, D. Magnetic: Multi-agent machine learning-based approach for energy efficient dynamic consolidation in data centers. *IEEE Trans. Serv. Comput.* **2019**, *15*, 30–44. [CrossRef]
27. Barto, A.G.; Bradtke, S.J.; Singh, S.P. Learning to act using real-time dynamic programming. *Artif. Intell.* **1995**, *72*, 81–138. [CrossRef]
28. Kaelbling, L.P.; Littman, M.L.; Moore, A.W. Reinforcement learning: A survey. *J. Artif. Intell. Res.* **1996**, *4*, 237–285. [CrossRef]
29. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef] [PubMed]
30. Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Netw.* **2015**, *61*, 85–117. [CrossRef] [PubMed]
31. Mnih, V. Playing atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.
32. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529–533. [CrossRef] [PubMed]
33. Konda, V.R.; Tsitsiklis, J.N. Actor-critic algorithms. *Adv. Neural Inf. Process. Syst.* **1999**, *12*, 1008–1014.
34. Mnih, V. Asynchronous Methods for Deep Reinforcement Learning. *arXiv* **2016**, arXiv:1602.01783.
35. Lillicrap, T. Continuous control with deep reinforcement learning. *arXiv* **2015**, arXiv:1509.02971.
36. Bellemare, M.G.; Dabney, W.; Munos, R. A distributional perspective on reinforcement learning. In Proceedings of the 34th International Conference on Machine Learning (ICML 2017), Sydney, Australia, 6–11 August 2017; pp. 449–458.
37. Brockman, G. OpenAI Gym. *arXiv* **2016**, arXiv:1606.01540.
38. Towers, M.; Kwiatkowski, A.; Terry, J.; Balis, J.U.; De Cola, G.; Deleu, T.; Goulao, M.; Kallinteris, A.; Krimmel, M.; KG, A.; et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv* **2024**, arXiv:2407.17032.
39. Todorov, E.; Erez, T.; Tassa, Y. MuJoCo: A physics engine for model-based control. In Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vilamoura-Algarve, Portugal, 7–12 October 2012; pp. 5026–5033.
40. Zakka, K.; Tabanpour, B.; Liao, Q.; Haiderbhai, M.; Holt, S.; Luo, J.Y.; Allshire, A.; Frey, E.; Sreenath, K.; Kahrs, L.A.; et al. MuJoCo Playground. *arXiv* **2025**, arXiv:2502.08844.
41. Juliani, A. Unity: A general platform for intelligent agents. *arXiv* **2018**, arXiv:1809.02627.
42. Bellemare, M.G.; Naddaf, Y.; Veness, J.; Bowling, M. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.* **2013**, *47*, 253–279. [CrossRef]
43. Kempka, M.; Wydmuch, M.; Runc, G.; Toczek, J.; Jaśkowski, W. ViZDoom: A Doom-Based AI Research Platform for Visual Reinforcement Learning. In Proceedings of the 2016 IEEE Conference on Computational Intelligence and Games (CIG), Santorini, Greece, 20–23 September 2016; pp. 1–8.
44. Van Hasselt, H.; Guez, A.; Silver, D. Deep Reinforcement Learning with Double Q-Learning. In Proceedings of the 30th AAAI Conference on Artificial Intelligence (AAAI 2016), Phoenix, AZ, USA, 12–17 February 2016; Volume 30, pp. 2094–2100.
45. Du, S.S.; Luo, Y.; Wang, R.; Zhang, H. Provably Efficient Q-Learning with Function Approximation via Distribution Shift Error Checking Oracle. *Adv. Neural Inf. Process. Syst.* **2019**, **32**, 8058–8068.
46. Ostrovski, G.; Bellemare, M.G.; van den Oord, A.; Munos, R. Count-Based Exploration with Neural Density Models. In Proceedings of the 34th International Conference on Machine Learning (ICML 2017), Sydney, Australia, 6–11 August 2017; pp. 2721–2730.
47. Ha, D.; Schmidhuber, J. Recurrent world models facilitate policy evolution. *Adv. Neural Inf. Process. Syst.* **2018**, *31*, 2455–2467.
48. Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature* **2020**, *588*, 604–609. [CrossRef] [PubMed]
49. Puterman, M.L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*; John Wiley & Sons: Hoboken, NJ, USA, 1994.

50. Bellman, R. Dynamic programming. *Science* **1966**, *153*, 34–37. [CrossRef]
51. Sutton, R.S. Learning to predict by the methods of temporal differences. *Mach. Learn.* **1988**, *3*, 9–44. [CrossRef]
52. Bellman, R. *Dynamic Programming*; Princeton University Press: Princeton, NJ, USA, 1957.
53. Watkins, C.J.C.H. Learning from Delayed Rewards. Ph.D. Thesis, King's College, Cambridge, UK, 1989.
54. Doya, K. Reinforcement learning in continuous time and space. *Neural Comput.* **2000**, *12*, 219–245. [CrossRef]
55. Rummery, G.A.; Niranjan, M. *On-Line Q-Learning Using Connectionist Systems*; Department of Engineering, University of Cambridge: Cambridge, UK, 1994; Volume 37.
56. Prashanth, L.; Bhatnagar, S. Reinforcement learning with average cost for adaptive control of traffic lights at intersections. In Proceedings of the 2011 14th International IEEE Conference on Intelligent Transportation Systems (ITSC), Washington, DC, USA, 5–7 October 2011; pp. 1640–1645.
57. Moody, J.; Saffell, M. Learning to trade via direct reinforcement. *IEEE Trans. Neural Netw.* **2001**, *12*, 875–889. [CrossRef] [PubMed]
58. Bertsekas, D. *Neuro-Dynamic Programming*; Athena Scientific: Nashua, NH, USA, 1996.
59. Tesauro, G. Temporal difference learning and TD-Gammon. *Commun. ACM* **1995**, *38*, 58–68. [CrossRef]
60. Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings, Sardinia, Italy, 13–15 May 2010; pp. 249–256.
61. He, K.; Zhang, X.; Ren, S.; Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In Proceedings of the IEEE International Conference on Computer Vision, Santiago, Chile, 13–17 December 2015; pp. 1026–1034.
62. Ioffe, S. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv* **2015**, arXiv:1502.03167.
63. Nair, V.; Hinton, G.E. Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th International Conference on Machine Learning (ICML-10), Haifa, Israel, 21–24 June 2010; pp. 807–814.
64. Glorot, X.; Bordes, A.; Bengio, Y. Deep sparse rectifier neural networks. In Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings, Fort Lauderdale, FL, USA, 11–13 April 2011; pp. 315–323.
65. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1097–1105. [CrossRef]
66. Graves, A.; Mohamed, A.R.; Hinton, G. Speech recognition with deep recurrent neural networks. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 26–31 May 2013; pp. 6645–6649.
67. Riedmiller, M. Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method. In Proceedings of the Machine Learning: ECML 2005: 16th European Conference on Machine Learning, Porto, Portugal, 3–7 October 2005; Proceedings 16; Springer: Berlin/Heidelberg, Germany, 2005; pp. 317–328.
68. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
69. Kingma, D.P. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
70. Lin, L.J. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Mach. Learn.* **1992**, *8*, 293–321. [CrossRef]
71. Wang, Z.; Schaul, T.; Hessel, M.; Hasselt, H.; Lanctot, M.; Freitas, N. Dueling network architectures for deep reinforcement learning. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 1995–2003.
72. Schaul, T. Prioritized Experience Replay. *arXiv* **2015**, arXiv:1511.05952.
73. Hessel, M.; Modayil, J.; Van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; Silver, D. Rainbow: Combining improvements in deep reinforcement learning. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32.
74. Sutton, R.S.; McAllester, D.; Singh, S.; Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Adv. Neural Inf. Process. Syst.* **1999**, *12*.
75. Baxter, J.; Bartlett, P.L. Infinite-horizon policy-gradient estimation. *J. Artif. Intell. Res.* **2001**, *15*, 319–350. [CrossRef]
76. Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* **1992**, *8*, 229–256. [CrossRef]
77. Jayakody, D. REINFORCE—A Quick Introduction (with Code). 2023. Available online: https://dilithjay.com/blog/reinforce-a-quick-introduction-with-code (accessed on 20 January 2025).
78. Baird, L.C. Reinforcement learning in continuous time: Advantage updating. In Proceedings of the 1994 IEEE International Conference on Neural Networks (ICNN'94), Orlando, FL, USA, 28 June–2 July 1994; Volume 4, pp. 2448–2453.
79. Deisenroth, M.P.; Neumann, G.; Peters, J. A survey on policy search for robotics. *Found. Trends Robot.* **2013**, *2*, 1–142.
80. Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; Abbeel, P. High-dimensional continuous control using generalized advantage estimation. *arXiv* **2015**, arXiv:1506.02438.

81. Schulman, J. Trust Region Policy Optimization. *arXiv* **2015**, arXiv:1502.05477.
82. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
83. Haarnoja, T.; Zhou, A.; Abbeel, P.; Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 1861–1870.
84. Andrychowicz, O.M.; Baker, B.; Chociej, M.; Jozefowicz, R.; McGrew, B.; Pachocki, J.; Petron, A.; Plappert, M.; Powell, G.; Ray, A.; et al. Learning dexterous in-hand manipulation. *Int. J. Robot. Res.* **2020**, *39*, 3–20. [CrossRef]
85. Henderson, P.; Islam, R.; Bachman, P.; Pineau, J.; Precup, D.; Meger, D. Deep reinforcement learning that matters. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32.
86. Barto, A.G.; Sutton, R.S.; Anderson, C.W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Trans. Syst. Man Cybern.* **1983**, *13*, 834–846. [CrossRef]
87. Uhlenbeck, G.E.; Ornstein, L.S. On the theory of the Brownian motion. *Phys. Rev.* **1930**, *36*, 823. [CrossRef]
88. Espeholt, L.; Soyer, H.; Munos, R.; Simonyan, K.; Mnih, V.; Ward, T.; Doron, Y.; Firoiu, V.; Harley, T.; Dunning, I.; et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 1407–1416.
89. Akkaya, I.; Andrychowicz, M.; Chociej, M.; Litwin, M.; McGrew, B.; Petron, A.; Paino, A.; Plappert, M.; Powell, G.; Ribas, R.; et al. Solving rubik's cube with a robot hand. *arXiv* **2019**, arXiv:1910.07113.
90. Ouyang, L.; Wu, J.; Jiang, X.; Almeida, D.; Wainwright, C.; Mishkin, P.; Zhang, C.; Agarwal, S.; Slama, K.; Ray, A.; et al. Training language models to follow instructions with human feedback. *Adv. Neural Inf. Process. Syst.* **2022**, *35*, 27730–27744.
91. Duan, Y.; Chen, X.; Houthooft, R.; Schulman, J.; Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 1329–1338.
92. Haarnoja, T.; Ha, S.; Zhou, A.; Tan, J.; Tucker, G.; Levine, S. Learning to walk via deep reinforcement learning. *arXiv* **2018**, arXiv:1812.11103.
93. Peng, X.B.; Berseth, G.; Yin, K.; Van De Panne, M. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Trans. Graph.* **2017**, *36*, 1–13. [CrossRef]
94. Kakade, S.M. *On the Sample Complexity of Reinforcement Learning*; University of London, University College: London, UK, 2003.
95. Fujimoto, S.; Hoof, H.; Meger, D. Addressing function approximation error in actor-critic methods. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 1587–1596.
96. Baheti, P. The Beginner's Guide to Deep Reinforcement Learning. 2021. Available online: https://www.v7labs.com/blog/deep-reinforcement-learning-guide (accessed on 20 January 2025).
97. Bertsekas, D.P. *Dynamic Programming and Optimal Control: Volume I*, 4th ed.; Athena Scientific: Nashua, NH, USA, 2012.
98. Sutton, R.S. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *Machine Learning Proceedings 1990*; Elsevier: Amsterdam, The Netherlands, 1990; pp. 216–224.
99. Brafman, R.I.; Tennenholtz, M. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *J. Mach. Learn. Res.* **2002**, *3*, 213–231.
100. Kearns, M.; Singh, S. Near-optimal reinforcement learning in polynomial time. *Mach. Learn.* **2002**, *49*, 209–232. [CrossRef]
101. Deisenroth, M.; Rasmussen, C.E. PILCO: A model-based and data-efficient approach to policy search. In Proceedings of the 28th International Conference on Machine Learning (ICML-11), Bellevue, WA, USA, 28 June–2 July 2011; pp. 465–472.
102. Li, W.; Todorov, E. Iterative linear quadratic regulator design for nonlinear biological movement systems. In Proceedings of the First International Conference on Informatics in Control, Automation and Robotics, Setúbal, Portugal, 25–28 August 2004; SciTePress: Setúbal, Portugal, 2004; Volume 2, pp. 222–229.
103. Todorov, E.; Li, W. A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems. In Proceedings of the 2005, American Control Conference, Portland, OR, USA, 8–10 June 2005; IEEE: Piscataway, NJ, USA, 2005; pp. 300–306.
104. Mayne, D.Q.; Rawlings, J.B.; Rao, C.V.; Scokaert, P.O. Constrained model predictive control: Stability and optimality. *Automatica* **2000**, *36*, 789–814. [CrossRef]
105. Watter, M.; Springenberg, J.; Boedecker, J.; Riedmiller, M. Embed to control: A locally linear latent dynamics model for control from raw images. *Adv. Neural Inf. Process. Syst.* **2015**, *28*.
106. Oh, J.; Guo, X.; Lee, H.; Lewis, R.L.; Singh, S. Action-conditional video prediction using deep networks in atari games. *Adv. Neural Inf. Process. Syst.* **2015**, *28*.
107. Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* **2018**, *362*, 1140–1144. [CrossRef] [PubMed]

108. Chua, K.; Calandra, R.; McAllister, R.; Levine, S. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Adv. Neural Inf. Process. Syst.* **2018**, *31*.
109. Kurutach, T.; Clavera, I.; Duan, Y.; Tamar, A.; Abbeel, P. Model-ensemble trust-region policy optimization. *arXiv* **2018**, arXiv:1802.10592.
110. Hafner, D.; Lillicrap, T.; Fischer, I.; Villegas, R.; Ha, D.; Lee, H.; Davidson, J. Learning latent dynamics for planning from pixels. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; pp. 2555–2565.
111. Johannink, T.; Bahl, S.; Nair, A.; Luo, J.; Kumar, A.; Loskyll, M.; Ojea, J.A.; Solowjow, E.; Levine, S. Residual reinforcement learning for robot control. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 6023–6029.
112. Talvitie, E. Self-correcting models for model-based reinforcement learning. In Proceedings of the AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; Volume 31.
113. Nagabandi, A.; Konolige, K.; Levine, S.; Kumar, V. Deep dynamics models for learning dexterous manipulation. In Proceedings of the Conference on Robot Learning, Virtual Event, 13–18 July 2020; pp. 1101–1112.
114. Rechenberg, I. Evolutionsstrategien. In *Numerische Optimierung*; Schwefel, H.P., Ed.; Springer: Berlin/Heidelberg, Germany, 1978; pp. 83–114.
115. Schwefel, H.P. *Numerische Optimierung von Computer-Modellen Mittels der Evolutionsstrategie: Mit einer Vergleichenden Einführung in die Hill-Climbing-und Zufallsstrategie*; Springer: Berlin/Heidelberg, Germany, 1977; Volume 1.
116. Hansen, N.; Ostermeier, A. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.* **2001**, *9*, 159–195. [CrossRef] [PubMed]
117. Salimans, T.; Ho, J.; Chen, X.; Sidor, S.; Sutskever, I. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv* **2017**, arXiv:1703.03864.
118. Mouret, J.B.; Doncieux, S. Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evol. Comput.* **2012**, *20*, 91–133. [CrossRef]
119. Dayan, P.; Hinton, G.E. Feudal reinforcement learning. *Adv. Neural Inf. Process. Syst.* **1992**, *5*, 271–278.
120. Sutton, R.S.; Precup, D.; Singh, S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.* **1999**, *112*, 181–211. [CrossRef]
121. Bacon, P.L.; Harb, J.; Precup, D. The option-critic architecture. In Proceedings of the AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; Volume 31.
122. Oh, J.; Chockalingam, V.; Singh, S.; Lee, H. Control of memory, active perception, and action in minecraft. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 2790–2799.
123. Kulkarni, T.D.; Narasimhan, K.; Saeedi, A.; Tenenbaum, J. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Adv. Neural Inf. Process. Syst.* **2016**, *29*, 3675–3683.
124. Vezhnevets, A.S.; Osindero, S.; Schaul, T.; Heess, N.; Jaderberg, M.; Silver, D.; Kavukcuoglu, K. Feudal networks for hierarchical reinforcement learning. In Proceedings of the International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; pp. 3540–3549.
125. Vinyals, O.; Babuschkin, I.; Czarnecki, W.M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D.H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* **2019**, *575*, 350–354. [CrossRef] [PubMed]
126. Jumper, J.; Evans, R.; Pritzel, A.; Green, T.; Figurnov, M.; Ronneberger, O.; Tunyasuvunakool, K.; Bates, R.; Žídek, A.; Potapenko, A.; et al. Highly accurate protein structure prediction with AlphaFold. *Nature* **2021**, *596*, 583–589. [CrossRef]
127. Graesser, L.; Keng, W.L. *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*; Addison-Wesley Professional: Boston, MA, USA, 2019.
128. Kalashnikov, D.; Irpan, A.; Pastor, P.; Ibarz, J.; Herzog, A.; Jang, E.; Quillen, D.; Holly, E.; Kalakrishnan, M.; Vanhoucke, V.; et al. QT-Opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv* **2018**, arXiv:1806.10293.
129. Racanière, S.; Weber, T.; Reichert, D.; Buesing, L.; Guez, A.; Jimenez Rezende, D.; Puigdomènech Badia, A.; Vinyals, O.; Heess, N.; Li, Y.; et al. Imagination-augmented agents for deep reinforcement learning. *Adv. Neural Inf. Process. Syst.* **2017**, *30*.
130. Oh, J.; Singh, S.; Lee, H. Value prediction network. *Adv. Neural Inf. Process. Syst.* **2017**, *30*.
131. Burda, Y.; Edwards, H.; Storkey, A.; Klimov, O. Exploration by random network distillation. *arXiv* **2018**, arXiv:1810.12894.
132. Pathak, D.; Agrawal, P.; Efros, A.A.; Darrell, T. Curiosity-driven exploration by self-supervised prediction. In Proceedings of the International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; pp. 2778–2787.
133. Bellemare, M.; Srinivasan, S.; Ostrovski, G.; Schaul, T.; Saxton, D.; Munos, R. Unifying count-based exploration and intrinsic motivation. *Adv. Neural Inf. Process. Syst.* **2016**, *29*.
134. Oudeyer, P.Y.; Kaplan, F.; Hafner, V.V. Intrinsic motivation systems for autonomous mental development. *IEEE Trans. Evol. Comput.* **2007**, *11*, 265–286. [CrossRef]
135. Taylor, M.E.; Stone, P. Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.* **2009**, *10*, 1633–1685.

136. Rusu, A.A.; Rabinowitz, N.C.; Desjardins, G.; Soyer, H.; Kirkpatrick, J.; Kavukcuoglu, K.; Pascanu, R.; Hadsell, R. Progressive neural networks. *arXiv* **2016**, arXiv:1606.04671.
137. Levine, S.; Kumar, A.; Tucker, G.; Fu, J. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv* **2020**, arXiv:2005.01643.
138. Hafner, D.; Pasukonis, J.; Ba, J.; Lillicrap, T. Mastering diverse domains through world models. *arXiv* **2023**, arXiv:2301.04104.
139. Amodei, D.; Olah, C.; Steinhardt, J.; Christiano, P.; Schulman, J.; Mané, D. Concrete problems in AI safety. *arXiv* **2016**, arXiv:1606.06565.
140. Achiam, J.; Held, D.; Tamar, A.; Abbeel, P. Constrained policy optimization. In Proceedings of the International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; pp. 22–31.
141. Alshiekh, M.; Bloem, R.; Ehlers, R.; Könighofer, B.; Niekum, S.; Topcu, U. Safe reinforcement learning via shielding. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32.
142. Morimoto, J.; Doya, K. Robust reinforcement learning. *Neural Comput.* **2005**, *17*, 335–359. [CrossRef] [PubMed]
143. Pinto, L.; Davidson, J.; Sukthankar, R.; Gupta, A. Robust adversarial reinforcement learning. In Proceedings of the International Conference on Machine Learning, Sydney, Australia, 6–11 August 2017; pp. 2817–2826.
144. Coppens, Y.; Efthymiadis, K.; Lenaerts, T.; Nowé, A.; Miller, T.; Weber, R.; Magazzeni, D. Distilling deep reinforcement learning policies in soft decision trees. In Proceedings of the IJCAI 2019 Workshop on Explainable Artificial Intelligence, Macao, China, 10–16 August 2019; pp. 1–6.
145. Greydanus, S.; Koul, A.; Dodge, J.; Fern, A. Visualizing and understanding atari agents. In Proceedings of the International Conference on Machine Learning, Stockholm, Sweden, 10–15 July 2018; pp. 1792–1801.
146. Reed, S.; Zolna, K.; Parisotto, E.; Colmenarejo, S.G.; Novikov, A.; Barth-Maron, G.; Gimenez, M.; Sulsky, Y.; Kay, J.; Springenberg, J.T.; et al. A generalist agent. *arXiv* **2022**, arXiv:2205.06175.
147. Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; Morrone, B.; De Laroussilhe, Q.; Gesmundo, A.; Attariyan, M.; Gelly, S. Parameter-efficient transfer learning for NLP. In Proceedings of the International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019; pp. 2790–2799.
148. Fawzi, A.; Balog, M.; Romera-Paredes, B.; Hassabis, D.; Kohli, P. Discovering Novel Algorithms with AlphaTensor. 2022. Available online: https://deepmind.google/discover/blog/discovering-novel-algorithms-with-alphatensor/ (accessed on 8 February 2025).
149. Mankowitz, D.J.; Michi, A.; Zhernov, A.; Gelmi, M.; Selvi, M.; Paduraru, C.; Leurent, E.; Iqbal, S.; Lespiau, J.B.; Ahern, A.; et al. Faster sorting algorithms discovered using deep reinforcement learning. *Nature* **2023**, *618*, 257–263. [CrossRef] [PubMed]
150. FAIR; Bakhtin, A.; Brown, N.; Dinan, E.; Farina, G.; Flaherty, C.; Fried, D.; Goff, A.; Gray, J.; Hu, H.; et al. Human-level play in the game of Diplomacy by combining language models with strategic reasoning. *Science* **2022**, *378*, 1067–1074. [PubMed]
151. Bousmalis, K.; Vezzani, G.; Rao, D.; Devin, C.; Lee, A.X.; Bauza, M.; Davchev, T.; Zhou, Y.; Gupta, A.; Raju, A.; et al. Robocat: A self-improving foundation agent for robotic manipulation. *arXiv* **2023**, arXiv:2306.11706.
152. Ahn, M.; Brohan, A.; Brown, N.; Chebotar, Y.; Cortes, O.; David, B.; Finn, C.; Fu, C.; Gopalakrishnan, K.; Hausman, K.; et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv* **2022**, arXiv:2204.01691.
153. Bauza, M.; Chen, J.E.; Dalibard, V.; Gileadi, N.; Hafner, R.; Martins, M.F.; Moore, J.; Pevceviciute, R.; Laurens, A.; Rao, D.; et al. DemoStart: Demonstration-led auto-curriculum applied to sim-to-real with multi-fingered robots. *arXiv* **2024**, arXiv:2409.06613.
154. DeepMind. Genie 2: A Large-Scale Foundation World Model. 2024. Available online: https://deepmind.google/discover/blog/genie-2-a-large-scale-foundation-world-model/ (accessed on 8 February 2025).
155. Wang, G.; Xie, Y.; Jiang, Y.; Mandlekar, A.; Xiao, C.; Zhu, Y.; Fan, L.; Anandkumar, A. Voyager: An open-ended embodied agent with large language models. *arXiv* **2023**, arXiv:2305.16291.