PENINSULA PUBLISHING LLC

EDRAAK
Vol. **(2026)**, 2026, **pp**. 158–171
ISSN: 3078-8412

Review Article

Reinforcement Learning: A Comprehensive Review of Foundations, Algorithms, and Open Challenges

Mohammad Hatim Eliwi [1, *], Waleed A Hammod [2], Omar Abdulmaged Hammood [3], Omar Raed Alobaidi [4], Raed A Hasan [5], Salwana Mohamad Asmara [6]

[1] *Department of Computer Network Systems, Faculty of Computer Science and Information Technology University of Anbar, Anbar 31001, Iraq*

[2] *Department of Information Technology, Faculty of Computer Science and Information Technology, University of Anbar, Anbar 31001, Iraq*

[3] *Faculty of Information Technology, University of Fallujah, Anbar 31001, Iraq*

[4] *Department of Electrical Engineering, Faculty of Engineering, University of Anbar, Anbar 31001, Iraq*

[5] *Renewable Energy Research Unit, Northern Technical University, Mosul, 34001, Iraq*

[6] *Faculty of Computing, Universiti Malaysia Pahang Al-Sultan Abdullah, 26600 Pekan, Pahang, Malaysia*

ARTICLEINFO

Article History

Received 02 Jul 2026

Accepted: 18 Aug 2026
Revised 19 Sep 2026
Published 04 Oct 2026

Keywords

reinforcement learning,

Markov decision process,

Q-learning,

Deep Q-Network,

policy gradient,

actor-critic,

Proximal Policy Optimization,

reinforcement learning from human feedback.

CC BY

ABSTRACT

Reinforcement learning (RL) is a branch of machine learning that studies how an agent should act within an environment to maximize cumulative reward. Supervised learning fits a function to labeled examples; RL instead learns through the consequences of trial-and-error interaction. It also differs from unsupervised learning, since its objective is to optimize a scalar reward signal rather than uncover latent structure. This review offers a self-contained, technically grounded survey for readers who understand probability and machine learning but may be encountering RL for the first time. We start from the Markov Decision Process (MDP) formalism and the Bellman equations that support nearly every RL algorithm. From there, we sort the main approaches into value-based methods, including Q-learning, Deep Q-Networks, and their variants; policy-based methods, such as REINFORCE, natural policy gradients, Trust Region Policy Optimization, and Proximal Policy Optimization; and actor-critic methods, which combine the two, including A2C/A3C, DDPG, TD3, and Soft Actor-Critic. For each group, we give the relevant equations, review representative empirical findings from the original publications, and examine sample efficiency, stability, and suitability for discrete and continuous action spaces. The discussion then turns to one of the field's most consequential recent developments: applying RL to the alignment and reasoning abilities of large language models through Reinforcement Learning from Human Feedback (RLHF), Direct Preference Optimization (DPO), and reasoning-focused methods such as Group Relative Policy Optimization (GRPO), used in DeepSeek-R1. We survey five persistent open problems: efficiency, training stability, scalability to high-dimensional spaces, safety and robustness, and generalization/transfer alongside the main mitigations proposed for each. We then turn to model-based RL, meta-learning, intrinsic motivation, and agentic, tool-using RL systems.

1. INTRODUCTION

Reinforcement learning (RL) formalizes a deceptively simple idea: an agent interacts with an environment, receives feedback as a reward, and gradually changes its behavior to receive more of that reward. Though easy to describe, the idea has proved extraordinarily productive. It underpins superhuman game-playing agents [1], robotic control policies that acquire manipulation skills without hand-engineered controllers [2], and, most recently, the post-training pipelines that turn raw language models into instruction-following reasoning systems such as ChatGPT and DeepSeek-R1 [3], [4]. This review offers a single technically precise account of the field, covering the mathematical formalism shared by all RL algorithms, the main algorithmic families built on it, the practical challenges that constrain deployment, and the directions in which the field is currently moving.

*Corresponding author email: mohammed.hatim@uoanbar.edu.iq
DOI: https://doi.org/10.70470/EDRAAK/2026/013

RL is conventionally set apart from the two other major machine-learning paradigms by the feedback available to the learner. In supervised learning, training relies on a fixed dataset of input-output pairs, with the correct output supplied for each example [5]. Unsupervised learning has no labels; it looks instead for latent structure in the data, such as clusters, manifolds, or generative factors. RL falls somewhere between these approaches. The agent is given neither the right action nor an unlabeled view of underlying structure. It receives a scalar reward, and that signal may only partly and often only after the fact show whether its behavior was good. This difference has two important consequences. The first is temporal credit assignment: since a reward may not appear until many steps after an action, RL must determine which earlier actions deserve credit, a problem supervised learning does not face. Second, an RL agent's own actions shape the states it encounters and learns from. It therefore has to balance exploiting behavior that is already known to work against exploring unfamiliar behavior that could prove better a problem with no equivalent in supervised or unsupervised learning [5].

1.1 Historical Development and Recent Milestones

RL's intellectual roots come from two mostly separate traditions that were brought together only in the late twentieth century: the psychology of animal learning, especially research on trial-and-error behavior and reinforcement effects in operant conditioning, and the engineering discipline of optimal control, with its emphasis on dynamic programming and the Bellman equation for sequential decisions under uncertainty [5]. Sutton and Barto's temporal-difference (TD) learning framework, followed by Watkins' Q-learning algorithm, offered the first computationally tractable method for estimating long-run value directly from experience without a model of the environment's dynamics [6]. Even so, for roughly two decades RL remained limited to small, discrete problems, since tabular value functions cannot scale to large or continuous state spaces.

The deep learning revolution broke through that ceiling. In 2013, Mnih et al [7]. presented the Deep Q-Network (DQN), using a convolutional neural network to approximate the Q-function and learn Atari 2600 games directly from raw pixels. With one architecture and a single set of hyperparameters, it matched or exceeded human performance on the majority of the games tested [1], [7]. The refined version of this work was later published in Nature [7] and is widely viewed as the beginning of modern deep RL. Soon after, policy-gradient and actor-critic methods emerged to address continuous control and make training more stable, among them Deterministic Policy Gradients (DPG) [8], Trust Region Policy Optimization (TRPO) [9], and Proximal Policy Optimization (PPO) [10]. DeepMind's AlphaGo and AlphaZero paired deep RL with Monte Carlo tree search and, through self-play alone, defeated leading human players in Go, chess, and shogi. AlphaZero did so without using human game records [11], [12].

Since 2022, RL has moved noticeably toward language models a shift substantial enough to warrant a brief timeline of its own:

1.1.1 RLHF and Value Alignment (2022–2023)

Ouyang et al. showed that a pretrained language model could be fine-tuned through RL with a reward model trained in comparisons of human preferences. This pipeline, now called Reinforcement Learning from Human Feedback (RLHF), produced outputs that human evaluators preferred by a wide margin over those of a much larger but non-aligned base model [3]. Typically optimized with PPO, RLHF was the main technical ingredient behind Instruct and the later ChatGPT family of models.

1.1.2 Direct Preference Optimization (2023–2024)

Rafailov et al. demonstrated that the RLHF objective can be re-derived in closed form, allowing the policy to be trained directly from preference data with a simple classification-style loss. This avoids both the need to train a separate reward model and the need to run online RL rollouts [13]. Under a Bradley-Terry preference model, Direct Preference Optimization (DPO) is mathematically equivalent to the RLHF objective, yet it is substantially simpler and more stable to train. It has since become a standard alternative to PPO-based RLHF for preference fine-tuning.

1.1.3 Reasoning-Oriented RL (2024–2025)

OpenAI's o1 and DeepSeek's R1 showed that applying large-scale reinforcement learning directly to chain-of-thought generation rewarding models for producing verifiably correct answers to math and coding problems can elicit long reasoning traces that correct themselves, even without supervised fine-tuning on human-written reasoning [4]. DeepSeek-R1, in particular, brought Group Relative Policy Optimization (GRPO) to wider attention. GRPO is a PPO variant that estimates advantage by comparing the relative ranking of sampled completions within a group instead of relying on a learned critic network. That removes the need for a separate value model and substantially lowers memory and compute overhead [4].

1.1.4 Agentic and Tool-Using RL (2025–present)

The leading approach now treats the language model as a policy acting across multi-step episodes, with tool calls, code execution, and interaction with external environments. That marks a move away from single-turn "text-in, text-out" generation and toward RL-trained agents that plan, act, and observe over extended horizons [14]. Classical RL problems

are returning to the center of language-model research: assigning credit over long horizons, dealing with sparse and delayed rewards, and exploring safely. For several years, single-step preference optimization had largely set those issues aside.

1.2 Formal Framework: The Markov Decision Process

Almost every algorithm discussed in this review is an attempt to solve the same underlying optimization problem, posed as a Markov Decision Process (MDP). An MDP is defined by the tuple $\langle S, A, T, R, \rho_0, \gamma \rangle$, where S is the state space, A is the action space, $T: S \times A \to P(S)$ is a stochastic transition function giving a probability distribution over next states, $R: S \times A \to \mathbb{R}$ is the reward function, ρ_0 is the distribution over initial states, and $\gamma \in [0, 1)$ is a discount factor that determines how strongly future rewards are weighted relative to immediate ones [15]. The defining structural assumption is the Markov property: the distribution over the next state and reward depends only on the current state and action, not on the full history that preceded them.

The agent's behavior is governed by a policy $\pi_\theta: S \to P(A)$, parameterized by a vector $\theta \in \mathbb{R}^n$, which may be deterministic (mapping each state to a single action) or stochastic (mapping each state to a probability distribution over actions). Given a trajectory $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \ldots)$ generated by $s_0 \sim \rho_0$, $a_t \sim \pi_\theta(\cdot \mid s_t)$, $s_{t+1} \sim T(\cdot \mid s_t, a_t)$, the agent's objective is to choose θ to maximize the expected discounted return:

$$J(\pi_\theta) = \mathbb{E}_{s_0 \sim \rho_0,\, a_t \sim \pi_\theta,\, s_{t+1} \sim T} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right] \quad (1)$$

Two auxiliary functions recur throughout the literature and are central to nearly every algorithm in Sections 2 and 3. The state-value function $V^\pi(s) = \mathbb{E}_\pi[\sum_t \gamma^t r_t \mid s_0 = s]$ gives the expected discounted return obtained by following policy π from states, and the action-value function $Q^\pi(s, a) = \mathbb{E}_\pi[\sum_t \gamma^t r_t \mid s_0 = s, a_0 = a]$ gives the expected return from taking action a in state s and following π thereafter. These satisfy the Bellman expectation equations:

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} T(s'|s,a) \left[R(s,a) + \gamma V^\pi(s') \right] \quad (2)$$

$$Q^\pi(s,a) = \sum_{s'} T(s'|s,a) \left[R(s,a) + \gamma \sum_{a'} \pi(a'|s') Q^\pi(s',a') \right] \quad (3)$$

Replacing the expectation over actions in Equation (3) with a maximization yields the Bellman optimality equation, $Q^*(s,a) = \sum_{s'} T(s'|s,a)[R(s,a) + \gamma \max_{a'} Q^*(s',a')]$, whose fixed point Q^* is the action-value function of an optimal policy $\pi^*(s) = \operatorname{argmax}_a Q^*(s,a)$. This equation is the basis of every value-based method discussed in Section 2.1: if Q^* can be estimated, an optimal policy follows immediately by acting greedily with respect to it, with no need to ever represent the policy explicitly [15], [16].

RL algorithms are conventionally split into model-based and model-free families, and within the model-free family, into value-based, policy-based, and actor-critic methods, illustrated in Figure 1. Model-based algorithms explicitly estimate T and R (or are given them) and use the estimated model to plan, for instance via dynamic programming or tree search [17]. Model-free algorithms instead optimize the policy directly from sampled transitions, without ever forming an explicit model of the environment's dynamics; they require more interaction data in general but avoid the compounding errors that an inaccurate learned model can introduce into long-horizon planning [17].

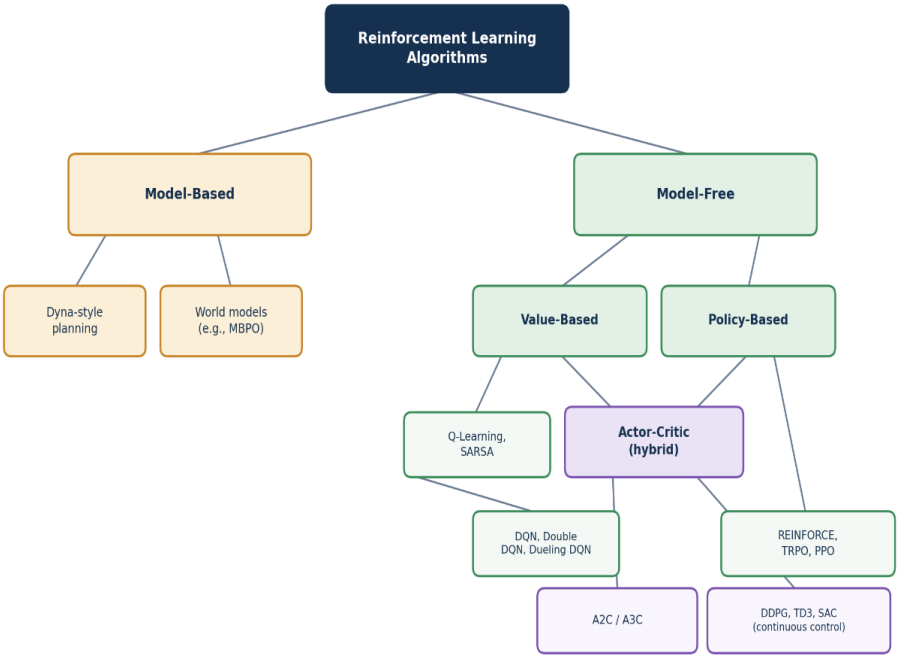


Fig. 1. Taxonomy of reinforcement learning algorithms by the source of the training signal.

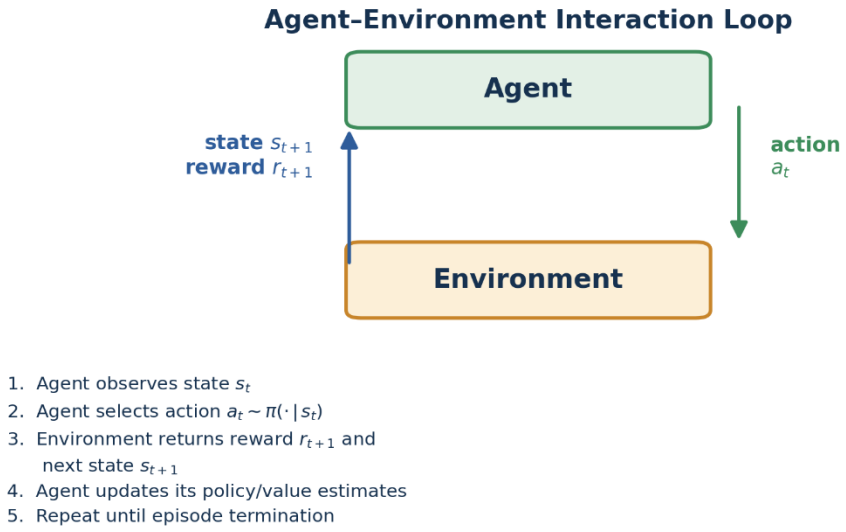


Fig. 2. The agent–environment interaction loop.

At each discrete step the agent observes a state, selects an action according to its policy, and receives a reward and successor state from the environment; the resulting transition is used to update the policy and/or value estimates.

1.3 Scope and Organization of This Review

This review moves from basic foundations toward current research directions. Section 2 examines the three main algorithmic family value-based, policy-based, and actor-critic methods presenting their governing equations and comparing their data efficiency, stability, and suitability for discrete and continuous control. Section 3 returns to the components shared by RL systems: the policy, reward, value function, and, optionally, an environment model. These are treated with formal precision appropriate after the algorithmic discussion in Section 2. Section 4 surveys five enduring challenges: sample efficiency; stability and convergence; scalability; safety and robustness; and generalization. It also discusses the main mitigations developed for each. Section 5 turns to model-based RL, meta-learning, intrinsic motivation, and the RL techniques RLHF, DPO, and GRPO that are now central to large language model post-training and agentic AI. Section 6 concludes.

2. KEY REINFORCEMENT LEARNING TECHNIQUES

RL algorithms can be classified by what they learn and represent explicitly. Value-based methods learn a value function and derive a policy from it implicitly (typically by acting greedily); policy-based methods parameterize and optimize the policy directly, with no value function required; and actor-critic methods learn both simultaneously, using the value function (critic) to reduce the variance of the policy (actor) update. This section develops each family in turn.

2.1 Value-Based Methods

Value-based methods estimate Q(s,a) or V(s) from sampled experience and act greedily, or near-greedily, with respect to that estimate. They are most naturally suited to discrete action spaces, since the greedy action argmax_a Q(s,a) requires enumerating A, which is intractable when A is continuous or very large.

2.1.1 Q-Learning

Q-learning, introduced by Watkins and refined by Watkins and Dayan, is a model-free, off-policy TD control algorithm that learns Q* directly, regardless of the policy used to generate behavior [16]. After observing a transition (s, a, r, s′), the tabular Q-learning update is:

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

where $\alpha \in (0,1]$ is the learning rate and the bracketed term is the TD error δ. Because the target uses $\max_{a'} Q(s',a')$ rather than the value of the action actually taken under the behavior policy, Q-learning is off-policy: it can learn the optimal policy while following an exploratory policy such as ε-greedy. Under standard stochastic-approximation conditions on α and with every state-action pair visited infinitely often, tabular Q-learning is guaranteed to converge to Q* [16]. This guarantee depends on the state-action space being tabular—that is, finite and enumerable. It does not carry over to the function-approximation setting used in deep RL, for which convergence is empirical rather than generally proven [18].

The tabular algorithm scales badly because it needs a distinct table entry for each state–action pair. Once S or A becomes large or continuous, that approach is infeasible. Consider a single Atari frame, which has roughly possible pixel

configurations. Function approximation is therefore needed, especially deep function approximation, which is discussed next.

2.1.2 Deep Q-Networks (DQN)

Mnih et al. replaced the Q-table with a convolutional neural network $Q(s,a;\theta)$ that takes raw pixels as input and outputs one value per discrete action, trained by minimizing the squared TD error against a target computed from a separate, periodically updated copy of the network [1], [7]:

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{U}} [(r + \gamma \max_{a'} Q(s',a';\theta^-) - Q(s,a;\theta))^2] \quad (5)$$

Naively combining Q-learning with a neural function approximator is well known to be unstable or divergent, for two compounding reasons: (i) consecutive samples from a single trajectory are highly correlated, violating the i.i.d. assumption that stochastic gradient descent relies on, and (ii) the target $r + \gamma \max_{a'} Q(s',a';\theta)$ moves every time θ is updated, since the same network is used to compute both the prediction and the target. DQN addresses both problems with two specific mechanisms:

- Experience Replay: Transitions are kept in a replay buffer $\mathcal{U}$, with minibatches drawn uniformly at random for each gradient update. This breaks temporal correlation and lets each transition be used in multiple updates, substantially improving sample efficiency [1].
- Target Network: A second network, with parameters θ^-, computes the target in Equation (5). Its parameters are updated either by periodically copying θ or through a slow exponential moving average, known as a "soft update." Since θ^- remains fixed between updates, the target is stationery over the short term, removing one major source of instability [7].

With these two additions, DQN matched or surpassed the score of a professional human game's tester on most of the 49 Atari 2600 games evaluated. The same network architecture and hyperparameters were used for every game, with no game-specific feature engineering; the system learned to play directly from pixels and score [7]. Two widely adopted refinements target particular weaknesses. Double DQN separates action selection from action evaluation when computing the target: the online network selects the action, while the target network evaluates it. This corrects the systematic overestimation bias otherwise introduced by $\max_{a'} Q(s',a';\theta^-)$ [19]. DQN, by contrast, splits the network into distinct value and advantage streams, $Q(s,a) = V(s) + A(s,a)$, with a normalization term added to ensure identifiability. That design improves learning when many actions have similar value in each state [20].

2.2 Policy-Based Methods

Policy based methods parameterize the policy $\pi\theta$ itself, usually with a neural network that outputs either action probabilities for discrete actions or the parameters of a distribution such as a Gaussian for continuous actions. The parameters θ are then optimized by gradient ascent on the expected return $J(\theta)$ from Equation (1). This makes policy-based methods a natural fit for continuous or very large action spaces, where the argmax required by value-based methods is intractable. They can also represent genuinely stochastic optimal policies, something value-based methods cannot.

2.2.1 The Policy Gradient Theorem and REINFORCE

The policy gradient theorem [21] shows that the gradient of J with respect to θ can be written without needing to differentiate through the (generally unknown and non-differentiable) environment dynamics:

$$\nabla\theta\, J(\theta) = \mathbb{E}_{\pi} [\Sigma_t \nabla\theta \log \pi\theta(a_t|s_t) \cdot G_t] \quad (6)$$

where $G_t = \Sigma_{k=t}^{\infty} \gamma^{(k-t)} r_k$ is the empirical return from time t onward. Equation (6) is the basis of REINFORCE [21]: sample a trajectory under the current policy, then increase the log-probability of every action taken, weighted by the return that followed it. REINFORCE is simple and applies to any differentiable policy, discrete or continuous, but the Monte Carlo return G_t is a high-variance estimator (a single trajectory can have wildly different cumulative reward due purely to stochasticity in the environment and the policy), which makes raw REINFORCE slow and unstable to train in practice. The usual variance-reduction method subtracts a state-dependent baseline, $b(s_t)$, typically an estimate of $V(s_t)$, without adding bias. This follows from $\mathbb{E}_{\pi}[\nabla\theta \log\pi\theta(a|s)] = 0$ for any function that depends only on s:

$$\nabla\theta\, J(\theta) = \mathbb{E}_{\pi} [\Sigma_t \nabla\theta \log \pi\theta(a_t|s_t) \cdot (G_t - b(s_t))] \quad (7)$$

Once the baseline is supplied by a learned value function, the method becomes actor-critic rather than a purely policy-gradient approach, as discussed in Section 2.3.

2.2.2 Trust Region and Proximal Policy Optimization

A long-standing problem in policy gradient methods is deciding how large each step should be. If the step is too small, samples are wasted; if it is too large, the policy may collapse into a poor region of parameter space that is difficult to recover from. The reason is that every update changes the data distribution used to compute the next one. Trust Region Policy Optimization (TRPO) deals with this by maximizing a surrogate objective while imposing an explicit constraint: the KL-divergence between the new and old policies must remain below a threshold δ. The constraint is solved approximately

using a conjugate-gradient step followed by a line search [9]. Compared with vanilla policy gradients, TRPO substantially improved stability, though each update is computationally expensive and the method is delicate to implement correctly. Proximal Policy Optimization (PPO) [10] shares TRPO's central aim of preventing destructively large policy updates but replaces constrained optimization with a simple clipped surrogate objective, optimized through ordinary stochastic gradient ascent. Define $r_t(\theta) = \pi\theta(a_t|s_t) / \pi\theta_old(a_t|s_t)$ as the probability ratio between the new and old policy and let $\hat{A}_t$ denote an estimate of the advantage function (Section 2.3). PPO's clipped objective is:

$$L^{CLIP}(\theta) = \mathbb{E}_t [\min(r_t(\theta) \hat{A}, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon) \hat{A}_t)] \quad (8)$$

with ε as a small hyperparameter, commonly 0.2. Once $r_t(\theta)$ has moved beyond $[1-\varepsilon, 1+\varepsilon]$ in a favorable direction, the clip term removes any incentive to push it farther. Taking the outer minimum makes the objective a pessimistic, or lower, bound on the unclipped objective. Clipping can therefore discourage excessively large updates but never encourage them. Figure 3 illustrates the resulting objective as a function of the probability ratio for positive and negative advantage.

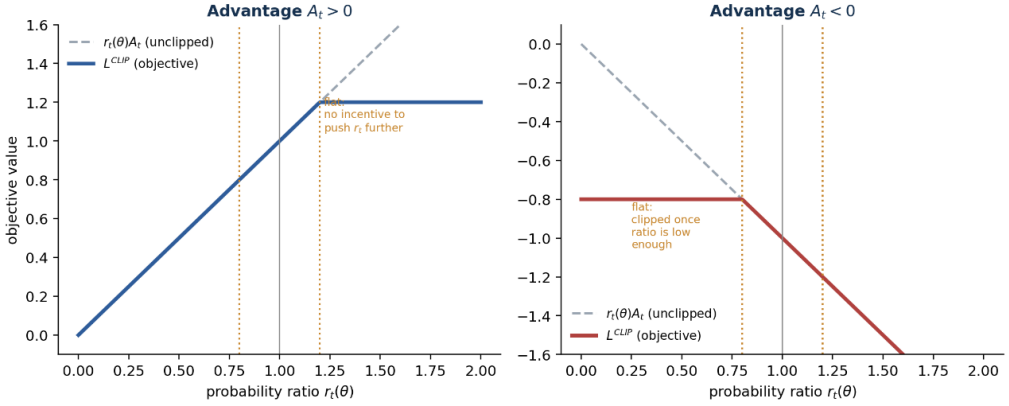


Fig. 3. PPO's clipped surrogate objective, L^CLIP, as a function of the policy probability ratio $r_t(\theta)$, shown positive advantages on the left and negative advantages on the right. The objective becomes flat when $r_t(\theta)$ moves ε away from 1 in the favorable direction, removing the incentive for arbitrarily large policy updates while preserving the gradient for corrective updates in the unfavorable direction.

PPO matches or surpasses TRPO in both sample efficiency and final performance across a range of continuous-control and Atari benchmarks, yet it is much simpler to implement because it requires only first-order gradient updates [10]. That mix of simplicity, stability, and strong empirical results has made PPO the default policy-optimization algorithm across much of applied deep RL, including the RLHF pipelines discussed in Section 5.4.

2.3 Actor-Critic Methods

Actor-critic methods jointly train two function approximators, as illustrated in Figure 4: an actor, $\pi\theta(a|s)$, which selects actions, and a critic, $V\varphi(s)$ or $Q\varphi(s, a)$, which evaluates them. Rather than relying on the high-variance Monte Carlo return used by REINFORCE, the critic provides the actor with a lower-variance learning signal typically the advantage function, $A(s, a) = Q(s, a) - V(s)$. This quantity indicates how much better action is than the policy's average action in states. In the one-step TD formulation, the advantage is estimated by the TD error $\delta_t = r_t + \gamma V\varphi(s_{t+1}) - V\varphi(s_t)$, which is computable from a single transition and therefore allows online, incremental updates rather than waiting for an episode to terminate.

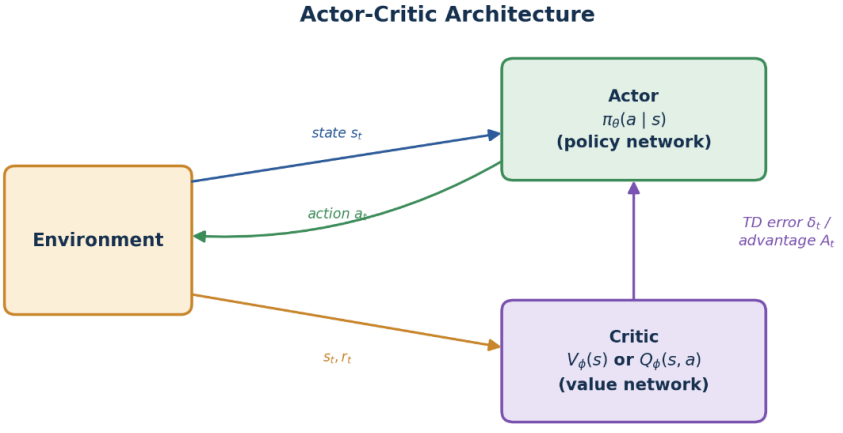


Fig. 4. Actor-critic architecture.

The critic uses observed transitions to estimate a value function; the resulting TD error or advantage estimate is fed back to the actor as a lower-variance substitute for the raw Monte Carlo return used by REINFORCE.

2.3.1 Advantage Actor-Critic (A2C/A3C)

Asynchronous Advantage Actor-Critic (A3C) [22] operates multiple actor-learner threads at once. Each thread interacts with its own copy of the environment, then periodically applies asynchronous gradient updates to shared parameters. Like DQN's replay buffer, this parallel setup helps decorrelate experience: at any given moment, different threads are usually exploring different parts of the state space. It also avoids the memory demands of a large re-play buffer while supporting on-policy updates. A2C uses the synchronous alternative. Its workers first finish a fixed-length rollout, after which one synchronized update is performed. That design is easier to implement correctly and fits GPU batch computation more efficiently, though it gives up some wall-clock throughput compared with fully asynchronous A3C.

2.3.2 Deterministic and Maximum Entropy Methods for Continuous Control

Deep Deterministic Policy Gradient (DDPG) [8] extends the Deterministic Policy Gradient theorem by adding DQN-style target networks and replay for continuous-action policy learning. It combines an off-policy critic $Q\varphi(s,a)$ with a deterministic actor $\mu\theta(s)$, updated according to $\nabla\theta\, J \approx \mathbb{E}\,[\nabla\theta\, \mu\theta(s)\, \nabla_a\, Q\varphi(s,a)|_{a=}\mu\theta(s)]$. Although its off-policy design makes DDPG sample-efficient, the method is notoriously sensitive to hyperparameters and, like vanilla Q-learning, tends to overestimate Q-values. Twin Delayed DDPG (TD3) [23] addresses these problems through three changes: clipped double Q-learning, which uses the lower of two independently trained critics to limit overestimation; delayed policy updates, which update the actor less often than the critics; and target policy smoothing, which adds noise to the target action and regularizes the value estimate against narrow, exploitable peaks in the learned Q-function.

Soft Actor-Critic (SAC) [24] pursues robustness differently, adding an entropy bonus to the standard RL objective: $\mathbb{E}\,[\Sigma_t\, \gamma^t(r_t + \alpha\, A(\pi(\cdot|s_t)))]$. Here, A represents the policy's entropy, while α is a temperature parameter that can also be learned automatically to maintain a target entropy. By keeping the policy stochastic during training, this entropy term supports exploration and gives the resulting policies some inherent robustness to model and environment misspecification. SAC also uses the clipped double-critic technique found in TD3, and as of this writing, it is widely regarded as one of the strongest off-policy algorithms for continuous control.

2.4 Comparative Summary

Table I summarizes the algorithms discussed in this section along four practically important dimensions: whether the algorithm is on-policy (can only learn from data generated by the current policy) or off-policy (can learn from data generated by any policy, including stale data in a replay buffer); whether it natively supports continuous action spaces; its typical sample efficiency; and the year and venue of its original publication.

TABLE I. COMPARATIVE SUMMARY OF THE MAJOR RL ALGORITHMS DISCUSSED IN SECTION 2.

Algorithm	Family	On/Off policy	Continuous Actions	Original Source
Q-Learning	Value-based	Off policy	No (discrete only)	Watkins & Dayan, 1992 [16]
DQN	Value-based	Off policy	No (discrete only)	Mnih et al., Nature 2015 [7]
Double DQN	Value-based	Off policy	No (discrete only)	van Hasselt et al., AAAI 2016 [19]
REINFORCE	Policy-based	On policy	Yes	Williams, 1992 [21]
TRPO	Policy-based	On policy	Yes	Schulman et al., ICML 2015 [9]
PPO	Policy-based	On policy	Yes	Schulman et al., 2017 [10]
A2C / A3C	Actor-critic	On policy	Yes	Mnih et al., ICML 2016 [22]
DDPG	Actor-critic	Off policy	Yes	Lillicrap et al., ICLR 2016 [8]
TD3	Actor-critic	Off policy	Yes	Fujimoto et al., ICML 2018 [23]
SAC	Actor-critic	Off policy	Yes	Haarnoja et al., ICML 2018 [24]

On-policy algorithms are usually easier to analyze and tend to be more stable because each update relies on data produced by the policy currently being optimized. The drawback is sample efficiency: once that policy changes, the data that generated it must be discarded. Off-policy methods can retain old experience in a replay buffer, which often improves sample efficiency, though they are more susceptible to instability when the policy being evaluated diverges substantially from the one that generated the data. This trade-off is made explicit here and recurs throughout Section 4, where sample efficiency and stability are treated as competing design objectives.

3. CORE COMPONENTS OF A REINFORCEMENT LEARNING SYSTEM

Having surveyed the major algorithmic families, it is useful to step back and isolate the four components that any RL system is built from, regardless of which algorithm instantiates them. Sutton and Barto identify these as the policy, the reward signal, the value function, and optionally a model of the environment [5]. Distinguishing them clearly resolves what is otherwise a common source of confusion: "reward" and "value" in particular are often used loosely as synonyms in informal discussion, but they play structurally different roles in every algorithm in Section 2.

3.1 Policy

The policy π defines the agent's behavior: a mapping from states (or, in the partially observable case, from observation histories) to actions or action probabilities. It is the only component that is strictly necessary for an agent to act at all even a policy-free, purely value-based agent implicitly defines a policy the moment it commits to acting greedily with respect to its value estimates. Policies can be as simple as lookup tables or as complex as deep neural networks containing hundreds of millions of parameters. They may also be deterministic (π: S → A) or stochastic (π: S → P(A)). That distinction is substantive, not cosmetic. When optimal behavior truly requires randomization, as it often does in multi-agent and partially observable settings, a stochastic policy is necessary. It also supplies exploration inherently, whereas a deterministic policy needs an additional mechanism, such as the additive action noise used in DDPG [8].

3.2 Reward Signal

The reward signal specifies what an RL problem is trying to achieve. At each step, the environment returns a scalar, $r_t = R(s_t, a_t)$, and the agent is designed to maximize the expected sum of discounted future rewards (Equation 1). This idea is sometimes called the reward hypothesis: all the goals and purposes an agent might have can be represented as maximizing the expected value of a scalar reward signal [5]. The reward signal is determined entirely by the environment and is not directly under the agent's control; the agent influences it only indirectly, by choosing actions that lead to states the environment associated with higher reward. A practically critical, and frequently underestimated, design problem is reward specification: a reward function that is technically correct but incompletely captures the designer's intent can be exploited by a sufficiently capable optimizer in ways the designer did not anticipate, a phenomenon known as reward hacking or specification gaming [25]. This concern is revisited in Section 4.4 as part of the broader discussion of safe RL.

3.3 Value Function

Where the reward signal specifies what is good in an immediate sense, the value function specifies what is good in the long run. As formalized in Equations (2) and (3), the value of a state (or state-action pair) is the expected cumulative discounted reward attainable from it, not the immediate reward alone; a state can have a low immediate reward but high value if it reliably leads to high-reward states later, or the reverse. Unlike the reward, which is given directly by the environment, the value must be estimated and continually re-estimated from the agent's accumulated experience this is precisely the estimation problem that the Bellman equations and the TD-learning techniques of Section 2.1 and 2.3 exist to solve. Sutton and Barto note that estimating value efficiently is arguably the single most important component of nearly all RL algorithms developed over the past several decades [5]; the centrality of the value function in this review's algorithmic taxonomy (Figure 1) reflects exactly this point.

3.4 Model of the Environment

A model predicts how the environment will respond to an action given (s, a), it predicts the resulting (s′, r) and allows the agent to plan, in the sense of evaluating candidate behavior by simulating its consequences before ever executing it in the real environment. Methods that use a model are called model-based; methods that do not, and instead learn purely from direct, trial-and-error interaction with the real environment, are called model-free. The trade-off is straightforward. Once learned, a model can be queried an arbitrary number of times at negligible marginal cost, producing substantial gains in sample efficiency over model-free learning. Yet systematic errors in that model may accumulate across a multi-step rollout, misleading the planning process in ways that are hard to identify from the model's predictions alone [17]. Section 5.1 returns to model-based RL as a leading approach to improving sample efficiency.

4. CHALLENGES AND OPEN ISSUES

Despite the algorithmic advances reviewed in Section 2, several challenges still constrain RL's reliability and use in real-world settings. They fall into five categories: sample efficiency; stability and convergence; scalability; safety and robustness; and generalization. For each, the section outlines the main mitigation strategies proposed in the literature.

4.1 Sample Efficiency

4.1.1 High Sample Complexity

Model-free deep RL algorithms often need millions or even billions of interactions with an environment before they perform well, including in simulation. That is several orders of magnitude more experience than a person needs to achieve comparable competence at, say, an Atari game [1, 26]. The gap becomes a serious problem when interaction is costly, slow, or risky to obtain; robotics and clinical decision-making are two commonly cited cases [2]. Researchers generally rely on three complementary mitigation strategies:

- Experience replay uses every collected transition across several gradient updates instead of discarding it after one on-policy update, spreading the cost of collecting the data across those updates [1].

- In model-based RL, a predictive model of the environment generates simulated rollouts that supplement or, in some cases, replace interaction with the real environment [17], [27].
- Meta-learning is trained in the distribution of related tasks, allowing it to adapt to a new, related task with only a handful of additional samples instead of starting from scratch [28].

4.1.2 Exploration versus Exploitation

An agent that only exploits its current estimate of the best action could become permanently stuck at a suboptimal policy if it never tried the actions that would reveal a better one; an agent that only explores never capitalizes on what it has already learned. This is one of the oldest identified problems in RL and remains unsolved in full generality [5]. The simplest practical approach, ε-greedy action selection, chooses a uniformly random action with probability ε and the greedy action otherwise; despite its simplicity it remains a strong baseline and was the exploration strategy used in the original DQN [1]. More sophisticated approaches give the agent an intrinsic reward for reaching novel or unexpected states. This is the idea behind curiosity-driven exploration, often implemented by rewarding states in which a learned forward model makes large prediction errors [29]. Another option is a principled Bayesian method such as Thompson sampling: the agent samples from its posterior belief about which action is best and chooses accordingly, balancing exploration with exploitation as that posterior becomes sharper with more data [30].

4.2 Stability and Convergence

4.2.1 Training Instability

Combining RL with function approximation gives up the convergence guarantees available in the tabular case (Section 2.1.1) and brings several well-documented forms of instability. The targets are non-stationary because the bootstrap target depends on the parameters being learned; gradient estimates can have high variance, particularly in Monte Carlo policy-gradient methods (Section 2.2.1); and the policy continually affects the data distribution it produces, allowing the training distribution to drift and disrupt learning. DQN addresses these problems through target networks and experience replay (Section 2.1.2), while PPO uses a clipped objective (Section 2.2.2). At their core, both are responses to this instability.

4.2.2 Hyperparameter Sensitivity

RL algorithms are notoriously sensitive to their hyperparameters. Learning rate, discount factor, replay buffer size, network architecture, and entropy or clipping coefficients can all affect performance, and published results are sometimes difficult to reproduce without the exact settings used by the original authors [31]. Grid and random search are still common, but both can be expensive. Bayesian optimization takes a different approach: it builds a probabilistic surrogate model of the relationship between hyperparameters and performance, then uses it to select promising configurations for evaluation. As a result, it can substantially reduce the number of training runs needed to find a good configuration [32].

4.3 Scalability

4.3.1 High-Dimensional State and Action Spaces

Deep function approximation has allowed RL to handle much larger state spaces. Even so, training remains computationally expensive, while exploration and credit assignment become substantially harder as dimensionality increases. To reduce this burden, state representations are often compressed with learned encoders or dimensionality-reduction methods such as autoencoders. Large discrete or continuous action spaces may also be discretized or broken into independent sub-actions, making the problem tractable [33].

4.3.2 Real-Time Decision-Making

Applications such as robotic control and autonomous driving impose strict limits on inference time. Many high-capacity RL policies have difficulty meeting them, especially with a large policy network or an action that involves costly planning, as in model-based or tree-search methods. In practice, the usual responses are to distill large policies into smaller, faster networks or to design architectures specifically for low-latency inference [34].

4.4 Safety and Robustness

4.4.1 Safe Exploration

Exploration necessarily means taking actions whose consequences are still unknown. That sits uneasily with any requirement that an agent avoid catastrophic or irreversible actions altogether. In a video game, this may be mostly an academic concern; in robotics, healthcare, and other real-world settings, it is a matter of safety. The constrained MDP framework captures this tension by adding one or more cost functions to the standard objective, along with constraints that the policy must satisfy in expectation rather than simply maximizing an unconstrained reward [35]. As discussed in Section 3.2, specifying the reward carefully is also a safety measure. A substantial fraction of documented RL safety failures stems

not from the algorithm itself but from reward functions that achieve exactly what they specify while missing what their designers intended [25].

4.4.2 Adversarial Robustness

Like other neural-network systems, trained RL policies may react strongly to small, carefully designed changes in their observations. Huang et al. showed that perturbations invisible to people could make a trained DQN agent playing Atari fail dramatically [36]. The main defenses studied so far are adversarial training, in which the policy is trained against an adversary that modifies observations or environment dynamics within a bounded budget, and robust optimization, which targets worst-case performance rather than expected performance [37].

4.5 Generalization and Transfer

4.5.1 Overfitting to the Training Environment

RL agents are commonly trained and evaluated in one fixed simulator or environment instance. Independently, Zhang et al. and Cobbe et al. found that agents trained under these conditions often fail to generalize to even small, task-irrelevant changes in the same environment's visuals or layout—a more pronounced and unexpected kind of overfitting than is usually observed in supervised learning [38, 39]. The most widely used response is domain randomization: training across many randomized versions of the environment's appearance and dynamics, making it difficult for the agent to memorize any single configuration. This approach is especially common in sim-to-real robotics transfer [40].

4.5.2 Transfer of Learning

Beyond resisting small changes within one task, transfer learning examines whether skills or representations acquired through one task can speed learning on another task that is related but not identical. Common ways to promote this kind of transfer include sharing network components between tasks, using policy distillation, and setting explicitly multi-task training objectives in which one policy must perform well on many tasks at once [41].

4.6 Summary of Challenges and Mitigations

Table II consolidates the five challenge categories discussed above together with their primary mitigation strategies and representative references.

TABLE II. THE FIVE PRINCIPAL OPEN CHALLENGES IN APPLIED RL AND THEIR LEADING MITIGATION STRATEGIES.

Challenge	Representative Mitigations	Key References
Sample efficiency	Experience replay; model-based RL/world models; meta-learning (e.g., MAML)	[1], [17], [27], [28]
Stability & convergence	Target networks; clipped/trust-region objectives; reward normalization	[1], [7], [9], [10]
Scalability	Representation learning/dimensionality reduction; action discretization; policy distillation	[33], [34]
Safety & robustness	Constrained MDPs; reward design audits; adversarial training	[25], [35], [36], [37]
Generalization & transfer	Domain randomization; multi-task training; policy distillation	[38], [39], [40], [41]

5. FUTURE DIRECTIONS

This section surveys five directions in which RL research is advancing: improving sample efficiency with model-based methods and meta-learning; developing more principled exploration strategies; creating more stable training algorithms; strengthening guarantees of safety and robustness; and, perhaps most practically significant at present, applying RL techniques to align large language models and elicit reasoning from them.

5.1 Improved Sample Efficiency

5.1.1 Model-Based RL

Model-based RL learns an explicit account of the environment's dynamics, then uses that model either to produce extra simulated training data or to plan directly. The result is less reliance on real-world interaction. Recent studies suggest that learned world models remain useful for planning even when they are imperfect, sometimes reaching model-free performance with only a fraction of the interaction. In Dreamer, Hafner et al. learn a compact latent-dynamics model and train the policy entirely through imagined rollouts in that latent space [27]. MuZero, by contrast, learns only the information needed for planning reward, value, and policy without directly reconstructing observations. Schrittwieser et al.'s system matched AlphaZero's performance on Go, chess, and shogi, while also producing state-of-the-art results on Atari, a domain its predecessor could not handle [42]. Sim-to-real transfer training in a (possibly randomized) simulator, then deploying the resulting policy on physical hardware remains the main practical approach for applying both model-based and model-free RL to robotics, where group real-world data is relatively slow and risky [40].

5.1.2 Meta-Learning

Meta-learning, often described as "learning to learn," trains a model over a distribution of tasks, allowing it to adjust to a new but related task with very little additional data. Model-Agnostic Meta-Learning (MAML) [28] takes this idea a step further by optimizing the starting parameters so that just a few standard gradient updates on a new task produce good performance. It has been used both for RL policies and for supervised few-shot learning. More broadly, Meta-RL examines agents that learn the adaptation strategy itself. One example conditions the agent on a recurrent hidden state that implicitly keeps track of its current task, instead of depending only on explicit gradient-based adaptation [43].

5.2 Advanced Exploration Strategies

Beyond ε-greedy and entropy-regularized exploration (Sections 4.1.2 and 2.3.2), intrinsic-motivation methods generate an auxiliary reward from the agent's own uncertainty or sense of novelty rather than from the extrinsic task reward. This encourages exploration even when the environment provides extremely sparse extrinsic rewards. In Pathak et al.'s curiosity-driven exploration, the reward is proportional to the error of a learned forward-dynamics model: states that the model cannot yet predict well are treated as those from which the agent has the most to learn [29]. Empowerment-based approaches take a different route, rewarding visits to states that give the agent substantial control over future outcomes, formalized through the mutual information between actions and future states [44]. Adaptive exploration schedules adjust the exploration rate or intrinsic-reward weighting online according to the agent's measured uncertainty, rather than relying on a fixed decay schedule. Bayesian approaches such as Thompson sampling (Section 4.1.2) also remain active areas of refinement [30].

5.3 Robust Training Algorithms and Automated Tuning

Ongoing work on optimization stability through adaptive learning-rate schedules, gradient clipping, and careful normalization of returns and advantages has remained an incremental yet consistently productive area of RL research. Even relatively small conceptual changes have often produced outsized empirical gains [31]. At the same time, automated hyperparameter optimization is being used more often. One example is population-based training, which evolves agents with different hyperparameters while they train [45]; the aim is to reduce the substantial human effort traditionally needed to tune RL algorithms (Section 4.2.2).

5.4 RL for Language Model Alignment and Reasoning

The most consequential recent development in applied RL has been its adoption as the leading method for post-training large language models. It transforms a model that only predicts the next token into one that follows instructions, expresses calibrated uncertainty, refuses unsafe requests, and, more recently, reasons step by step toward verifiably correct answers. This subsection examines the three techniques introduced briefly in Section 1.1.

5.4.1 Reinforcement Learning from Human Feedback (RLHF)

RLHF represents the language model as a policy $\pi\theta(\cdot|x)$ over token sequences conditioned on a prompt x, and it is carried out in three stages [3]. Human annotators first rank or compare candidate completions. Those preference judgments are then used to train a separate reward model, $r\varphi(x,y)$, which predicts which of two completions a person would prefer, typically using a Bradley-Terry loss for pairwise comparisons. The policy is then fine-tuned through RL almost always PPO (Section 2.2.2) to maximize $E[r\varphi(x,y)]$ while paying a KL-divergence penalty for moving too far from the original pretrained or supervised-fine-tuned model. This constraint is meant to stop the policy from drifting toward outputs that exploit flaws in the learned reward model instead of actually improving response quality. Ouyang et al. InstructGPT, trained through exactly this pipeline, produced outputs that human labelers preferred to those from a base model 100× larger and not tuned with RLHF the large majority of the time, despite having far fewer parameters [3].

5.4.2 Direct Preference Optimization (DPO)

RLHF's three-stage pipeline works, but it is operationally demanding. A separate reward model must be trained and maintained, while PPO relies on online sampling from the policy under training, making that stage relatively unstable and resource intensive. Rafailov et al. observed that, under the same Bradley-Terry preference model assumed by RLHF, the optimal RLHF policy has a closed-form relationship with the reward model. That relationship can be inverted, expressing the reward implicitly in terms of the policy itself [13]. When this expression is substituted into the Bradley-Terry preference loss, the result is a closed-form objective that can be optimized directly with standard supervised-style gradient descent on the language-model policy alone. No reward model or online RL rollouts are needed:

$$L_DPO(\theta) = -\mathbb{E}_{(x,y^+,y^-)}\{ [\log \sigma(\beta \log(\pi\theta(y^+|x)/\pi_ref(y^+|x)) - \beta \log(\pi\theta(y^-|x)/\pi_ref(y^-|x)))] \quad (9)$$

where y^+ and y^- denote the preferred and dispreferred completions for prompt x. π_ref is the fixed reference policy the pre-DPO model while σ denotes the logistic function, and β determines how strongly deviations from π_ref are penalized. In form, Equation (9) is a binary classification loss. That structure makes DPO substantially simpler to implement and more

stable to train than PPO-based RLHF; since 2023, DPO has consequently displaced RLHF as the default preference-tuning method in many open-source and production post-training pipelines [13].

5.4.3 Reasoning-Oriented RL: GRPO and DeepSeek-R1

RLHF and DPO both optimize against one subjective measure of human preference. A newer, separate body of work takes a different route: it trains models toward objectively verifiable correctness on tasks like mathematics and program synthesis. In these settings, the final answer can be checked or generated code can be run against test cases, allowing an automatic reward of 1 or 0 without any human rater or learned reward model. DeepSeek-R1 applies this form of large-scale RL directly to a base language model's chain-of-thought generation, using Group Relative Policy Optimization (GRPO) [4]. Rather than relying on a separately trained value-function critic, GRPO modifies PPO by sampling a group of G completions $\{o_1, \ldots, o_G\}$ for each prompt and estimating the advantage of each completion from its reward relative to the group: $A_i = (r_i - \text{mean}(\{r_1,\ldots,r_G\})) / \text{std}(\{r_1,\ldots,r_G\})$. Because it relies on relative rankings within the sampled group instead of a separate critic network, GRPO cuts roughly half the trainable parameters and memory footprint required by standard PPO-based RLHF. The latter has to train and serve the actor, critic, and reward models at the same time. With this approach, DeepSeek-R1 spontaneously developed extended, self-correcting chains of thought including explicit checks of intermediate steps from the reward signal alone. It received no supervised fine-tuning on human-authored reasoning traces yet reached performance competitive with OpenAI's o1 on competition mathematics and coding benchmarks [4].

5.4.4 Agentic and Tool-Using RL

The frontier now carries this reward-for-verifiable-outcomes idea beyond single-turn question answering, into multi-step episodes where a language-model agent plans, calls external tools such as code execution, web search, or file and browser access, observes what happens, and then proceeds. The boundary between the classical RL "agent" acting in an "environment" and a language model producing text starts to disappear [14,45,46,47]. That shift brings back, with full force, the long-horizon credit-assignment and safe-exploration problems discussed in Sections 4.1 and 4.4. When the only reward arrives at the end of a lengthy tool-use episode, it has to be traced through every intermediate decision. And once the agent can take real action, executing code or sending requests the safety concerns extend well beyond those of a system limited to emitting text [48].

5.5 Generalization and Lifelong Learning

Looking beyond individual tasks, lifelong or continual learning examines agents that accumulate skills across an open-ended series of tasks without catastrophically forgetting what they learned earlier, a well-documented failure mode in neural networks trained sequentially on new objectives [46] To date, research has centered on two strategies: regularization methods, which penalize changes to parameters that matter for earlier tasks, and architectural methods, which add capacity for new tasks while preserving existing parameters. Bringing either approach to the scale and generality of current RL-trained foundation models remains an open area of active research.

6. CONCLUSION

Reinforcement learning has moved from tabular algorithms with provable convergence on small, discrete problems to deep systems based on function approximation that can achieve superhuman gameplay and learn robotic control. Its most consequential recent application may be the post-training of large language models into aligned, reasoning, tool-using agents. This review followed that progression from its shared mathematical foundation, the Markov decision process and the Bellman equations (Section 1.2) to the three main algorithmic families used to solve it: value-based, policy-based, and actor-critic methods (Section 2). It then examined the components common to any RL system (Section 3), along with five persistent challenges sample efficiency, stability, scalability, safety, and generalization that still limit the field's reliability beyond curated benchmark settings (Section 4).

This survey points to two conclusions. The first is that reinforcement learning's algorithmic core has matured considerably. PPO and its variants, value-learning methods stabilized by target networks, and entropy-regularized actor-critic approaches are now generally understood and broadly dependable, not merely research curiosities. That maturity is what has made it possible to deploy RL at the scale demanded by RLHF, DPO, and GRPO-based language-model post-training (Section 5.4).

The second conclusion is less reassuring. The open problems listed in Section 4, especially safe exploration and faithful reward specification have, if anything, grown more pressing as RL has moved into systems that act in the world, including robots and agentic language models with tool access, rather than remaining confined to simulated game environments. Continued work on model-based RL, meta-learning, and intrinsic motivation (Section 5), together with progress on those challenges, will determine how much of RL's considerable recent progress can carry over to dependable real-world deployment.

Author Contributions

"Conceptualization, Waleed A Hammood.; methodology, Surizal Nazeri.; software, Waleed A Hammood.; formal analysis, Mohammad Hatim Eliwi.; investigation, Surizal Nazeri.; resources, Mohammed Abdulsalam.; data curation, Ejiro U Osiobe.; writing original draft preparation, Mohammad Hatim Eliwi.; writing review and editing, Omar Abdulmaged Hammood.; visualization, Mohammad Hatim Eliwi.; supervision, Salah A Aliesawi.; project administration, Waleed A Hammood.; funding acquisition, Noor Akma Abu Bakar. All authors have read and agreed to the published version of the manuscript.

Funding

This research was supported by the UNITEN Publication & Conference Support Fund 2026 and supported by Institute of Information & Computing in Energy (IICE) and Centre of AI Application & Innovation, College of Computing & Informatics (CCI) Universiti Tenaga Nasional (UNITEN).

Acknowledgments

The authors gratefully acknowledge the valuable comments and suggestions that helped improve the quality of this manuscript.

Conflicts of Interest

The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

RL	Reinforcement Learning
MDP	Markov Decision Process
DQN	Deep Q-Network
PPO	Proximal Policy Optimization
TRPO	Trust Region Policy Optimization
DDPG	Deep Deterministic Policy Gradient
SAC	Soft Actor-Critic
RLHF	Reinforcement Learning from Human Feedback
DPO	Direct Preference Optimization
GRPO	Group Relative Policy Optimization
MAML	Model-Agnostic Meta-Learning
LLM	Large Language Model

References

[1] V. Mnih et al., "Playing Atari with Deep Reinforcement Learning," arXiv:1312.5602, Dec. 2013.
[2] S. Levine et al., "End-to-End Training of Deep Visuomotor Policies," Journal of Machine Learning Research, vol. 17, no. 39, pp. 1–40, 2016.
[3] L. Ouyang et al., "Training Language Models to Follow Instructions with Human Feedback," arXiv:2203.02155, Mar. 2022.
[4] DeepSeek-AI, D. Guo et al., "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning," arXiv:2501.12948, Jan. 2025.
[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018.
[6] C. J. C. H. Watkins, "Learning from Delayed Rewards," Ph.D. dissertation, University of Cambridge, 1989.
[7] V. Mnih et al., "Human-level Control through Deep Reinforcement Learning," Nature, vol. 518, pp. 529–533, 2015.
[8] T. P. Lillicrap et al., "Continuous Control with Deep Reinforcement Learning," in Proc. ICLR, 2016.
[9] J. Schulman et al., "Trust Region Policy Optimization," in Proc. ICML, 2015, pp. 1889–1897.
[10] J. Schulman et al., "Proximal Policy Optimization Algorithms," arXiv:1707.06347, Jul. 2017.
[11] D. Silver et al., "Mastering the Game of Go with Deep Neural Networks and Tree Search," Nature, vol. 529, pp. 484–489, 2016.
[12] D. Silver et al., "A General Reinforcement Learning Algorithm that Masters Chess, Shogi, and Go through Self-Play," Science, vol. 362, no. 6419, pp. 1140–1144, 2018.
[13] R. Rafailov et al., "Direct Preference Optimization: Your Language Model is Secretly a Reward Model," arXiv:2305.18290, May 2023.
[14] L. Wang et al., "A Survey on Large Language Model Based Autonomous Agents," arXiv:2308.11432, Aug. 2023.
[15] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. New York: Wiley, 2014.
[16] C. J. C. H. Watkins and P. Dayan, "Q-Learning," Machine Learning, vol. 8, no. 3–4, pp. 279–292, 1992.
[17] M. Deisenroth and C. E. Rasmussen, "PILCO: A Model-Based and Data-Efficient Approach to Policy Search," in Proc. ICML, 2011, pp. 465–472.
[18] J. N. Tsitsiklis and B. Van Roy, "An Analysis of Temporal-Difference Learning with Function Approximation," IEEE Transactions on Automatic Control, vol. 42, no. 5, pp. 674–690, 1997.
[19] H. Van Hasselt et al., "Deep reinforcement learning with double Q-learning," in Proc. AAAI Conf. Artif. Intell., vol. 30, no. 1, Mar. 2016.

[20] Z. Wang et al., “Dueling Network Architectures for Deep Reinforcement Learning,” in Proc. ICML, 2016, pp. 1995–2003.
[21] R. J. Williams, “Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning,” Machine Learning, vol. 8, no. 3–4, pp. 229–256, 1992.
[22] A. P. Tanner, *An Evaluative Case Study of Project-Based Learning in High School Vocational Education*. Walden University, 2012.
[23] V. Mnih et al., “Asynchronous Methods for Deep Reinforcement Learning,” in Proc. ICML, 2016, pp. 1928–1937.
[24] S. Fujimoto et al., “Addressing Function Approximation Error in Actor-Critic Methods,” in Proc. ICML, 2018, pp. 1587–1596.
[25] W. A. Hammood et al., “Data analysis of an exploring the information systems success factors for early warning systems adoption,” Babylonian Journal of Machine Learning, 2024, pp. 102–111.
[26] T. Haarnoja et al., “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor,” in Proc. ICML, 2018, pp. 1861–1870.
[27] D. Amodei et al., “Concrete Problems in AI Safety,” arXiv:1606.06565, Jun. 2016.
[28] M. Hessel et al., “Rainbow: Combining Improvements in Deep Reinforcement Learning,” in Proc. AAAI, 2018, pp. 3215–3222.
[29] D. Hafner et al., “Dream to Control: Learning Behaviors by Latent Imagination,” in Proc. ICLR, 2020.
[30] C. Finn et al., “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks,” in Proc. ICML, 2017, pp. 1126–1135.
[31] D. Pathak et al., “Curiosity-Driven Exploration by Self-Supervised Prediction,” in Proc. ICML, 2017, pp. 2778–2787.
[32] I. Osband et al., “Deep Exploration via Bootstrapped DQN,” in Proc. NeurIPS, 2016, pp. 4026–4034.
[33] P. Henderson et al., “Deep Reinforcement Learning that Matters,” in Proc. AAAI, 2018, pp. 3207–3214.
[34] J. Snoek et al., “Practical Bayesian Optimization of Machine Learning Algorithms,” in Proc. NeurIPS, 2012, pp. 2951–2959.
[35] S. Gu et al., “Continuous Deep Q-Learning with Model-Based Acceleration,” in Proc. ICML, 2016, pp. 2829–2838.
[36] J. Ho and S. Ermon, “Generative Adversarial Imitation Learning,” in Proc. NeurIPS, 2016, pp. 4565–4573.
[37] E. Altman, *Constrained Markov Decision Processes*. Routledge, 2021.
[38] S. Huang et al., “Adversarial Attacks on Neural Network Policies,” arXiv:1702.02284, Feb. 2017.
[39] A. Pattanaik et al., “Robust Deep Reinforcement Learning with Adversarial Attacks,” in Proc. AAMAS, 2018, pp. 2040–2042.
[40] A. Zhang et al., “A Dissection of Overfitting and Generalization in Continuous Reinforcement Learning,” arXiv:1806.07937, Jun. 2018.
[41] K. Cobbe et al., “Quantifying Generalization in Reinforcement Learning,” in Proc. ICML, 2019, pp. 1282–1289.
[42] J. Tobin et al., “Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World,” in Proc. IROS, 2017, pp. 23–30.
[43] M. E. Taylor and P. Stone, “Transfer Learning for Reinforcement Learning Domains: A Survey,” Journal of Machine Learning Research, vol. 10, pp. 1633–1685, 2009.
[44] J. Schrittwieser et al., “Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model,” Nature, vol. 588, pp. 604–609, 2020.
[45] J. X. Wang et al., “Learning to Reinforcement Learn,” arXiv:1611.05763, Nov. 2016.
[46] K. Gregor et al., “Variational Intrinsic Control,” arXiv:1611.07507, Nov. 2016.
[47] M. Jaderberg et al., “Population Based Training of Neural Networks,” arXiv:1711.09846, Nov. 2017.
[48] J. Kirkpatrick et al., “Overcoming Catastrophic Forgetting in Neural Networks,” Proceedings of the National Academy of Sciences, vol. 114, no. 13, pp. 3521–3526, 2017.