Manifold Mathematics

Reinforcement Learning

From the Bellman Equation to Language Models

A complete course in sequential decision making:
dynamic programming, temporal-difference learning, policy gradients,
deep reinforcement learning, offline methods, and
policy optimisation for large language models

Manifold Mathematics Monograph Series

Visakhapatnam • July 2026

© 2026 Manifold Mathematics. All rights reserved.

This monograph is produced for study within the Manifold Mathematics Monograph Series. No part may be reproduced without permission.

First edition, July 2026.

Preface

Reinforcement learning occupies a peculiar and beautiful position in mathematics. It is at once a branch of applied probability—the theory of Markov decision processes—and a branch of numerical analysis, since almost every algorithm in the subject is a stochastic approximation scheme for solving a fixed-point equation. It is also, in the last decade, the engine behind the most visible achievements of artificial intelligence: agents that master Go and Atari from raw experience, robots that learn locomotion, and, most recently, the fine-tuning of large language models to follow human intent.

This monograph develops the subject from first principles to the research frontier. The organising thread is a single object: the *Bellman equation*. Part I establishes the Markov decision process and proves that the Bellman operators are contractions, so that value functions exist, are unique, and can be computed. Part II removes the model: Monte Carlo and temporal-difference methods estimate the same fixed points from sampled experience, and SARSA and Q-learning turn estimation into control. Part III confronts scale: function approximation, the policy gradient theorem, and actor–critic architectures. Part IV develops deep reinforcement learning proper—DQN and its refinements, trust-region and proximal methods, continuous control, and model-based agents that learn to dream. Part V carries the theory into the settings where it now earns its keep: learning from fixed datasets, evaluating policies offline, and optimising language models against human preference, including the production discipline of online reinforcement learning in deployed systems.

The style follows the conventions of this series. Every chapter opens with intuition and closes with rigour. *ELI5* boxes give the five-year-old's version of an idea; *Abstract Viewpoint* boxes give the category-level version; definitions, theorems, worked examples, and cautions are set in their own environments. Proofs are given where they are short enough to illuminate and cited where they are not. A reader who works through the examples with pencil and paper will finish with a working command of the field.

The prerequisite is modest: comfort with probability at the level of conditional expectation, linear algebra, and multivariable calculus. No prior exposure to machine learning is assumed, though it will not hurt.

Visakhapatnam, July 2026

Contents

Part I

Foundations

Chapter 1

The Agent and the Environment

1.1 What Reinforcement Learning Is

Machine learning divides, roughly, into three paradigms. In *supervised learning*, a teacher supplies labelled examples (x_i, y_i) and the learner fits a map $x \mapsto y$. In *unsupervised learning*, no labels exist and the learner seeks structure—clusters, densities, low-dimensional manifolds. *Reinforcement learning* (RL) is the third paradigm and the one closest to the way animals actually learn: an **agent** interacts with an **environment** over time, takes **actions**, observes the consequences, and receives a scalar **reward**. Nobody tells the agent what the correct action was. The agent must discover, by trial and error, which behaviour earns the most reward *in the long run*.

Three features distinguish RL from the other paradigms and generate its entire mathematical difficulty:

1. **Sequential structure.** Actions influence not only the immediate reward but the situations—and therefore the opportunities—that follow. A chess sacrifice loses material now to win the game later. Optimality is a property of whole trajectories, not of individual decisions.
2. **Evaluative, not instructive, feedback.** The reward says how good the chosen action was, not what the best action would have been. The learner never sees the counterfactual.
3. **The exploration–exploitation dilemma.** To maximise reward the agent should exploit the actions it currently believes are best; but to discover better actions it must explore ones it believes are worse. Every RL algorithm is, at heart, a resolution of this tension.

ELI5: Training a puppy

You cannot explain to a puppy, in words, how to sit. You can only wait for behaviour and hand out biscuits. The puppy tries things—random things at first—and slowly notices that some sequences of muscle movements are followed by biscuits. It does more of those. That is reinforcement learning: no instructions, only consequences, and a learner that shifts its behaviour toward whatever the world rewards.

1.2 The Interaction Loop

The agent–environment interaction proceeds in discrete time steps $t = 0, 1, 2, \ldots$. At each step:

1. the agent observes the environment's *state* s_t;
2. it selects an *action* a_t according to its *policy*;
3. the environment responds with a *reward* r_{t+1} and a new state s_{t+1};
4. the agent updates its policy in light of what happened.

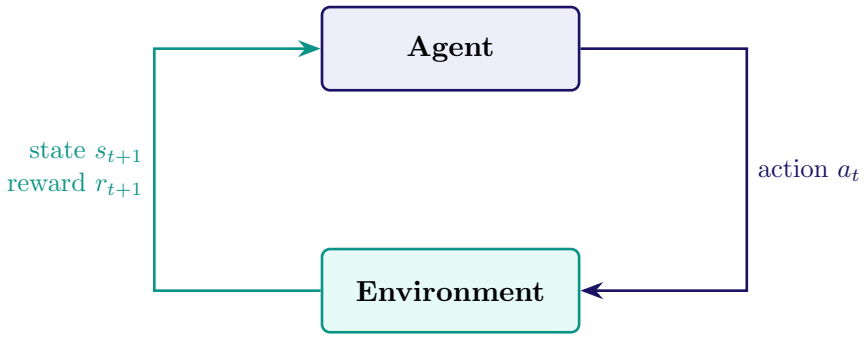

Figure 1.1: The reinforcement learning interaction loop. The agent emits actions; the environment returns states and rewards. Learning closes the loop.

The full record of one interaction,

$$\tau \;=\; (s_0, a_0, r_1, s_1, a_1, r_2, s_2, \dots),$$

is called a **trajectory** (equivalently a *rollout*; when it terminates, an *episode*). Everything the agent learns, it learns from trajectories.

1.3 The Cast of Characters

We now name each component precisely. These definitions are used on every page that follows.

Definition: The primitive objects

- **State** $s \in \mathcal{S}$: a summary of the environment at one instant, containing everything the agent needs in order to decide. In a board game, the position; in a market, a feature vector of prices and inventory.
- **Action** $a \in \mathcal{A}$: a decision available to the agent. The action set may depend on the state, $\mathcal{A}(s)$, may be finite (discrete control) or a subset of $\mathbb{R}^d$ (continuous control).
- **Reward** $r \in \mathbb{R}$: a scalar returned by the environment after each transition. The agent's purpose, by definition, is to accumulate as much of it as possible.
- **Policy** π: the agent's rule for acting. A *deterministic* policy is a map $\pi : \mathcal{S} \to \mathcal{A}$; a *stochastic* policy specifies conditional probabilities $\pi(a \mid s)$. The policy is the object we optimise.

Definition: Return and discounting

Fix a **discount factor** $\gamma \in [0, 1)$. The **return** from time t is the discounted sum of future rewards,

$$G_t \;=\; r_{t+1} + \gamma\, r_{t+2} + \gamma^2 r_{t+3} + \cdots \;=\; \sum_{k=0}^{\infty} \gamma^k\, r_{t+k+1}.$$

When rewards are bounded, $|r| \le R_{\max}$, the series converges absolutely with $|G_t| \le R_{\max}/(1-\gamma)$.

The discount factor plays three roles at once. Economically, it encodes impatience: a reward tomorrow is worth γ of the same reward today. Mathematically, it guarantees convergence of the

return and—as we prove in Chapter 2—makes the Bellman operator a contraction. Behaviourally, it sets the agent's planning horizon: rewards more than about $1/(1-\gamma)$ steps ahead are effectively invisible, so $\gamma = 0.99$ corresponds to a horizon of roughly one hundred steps, while $\gamma = 0.9$ sees only ten.

Caution

The reward is *given*, not learned. Reinforcement learning optimises whatever reward it is handed, including its defects. If the reward is a poor proxy for what you actually want, the agent will faithfully maximise the proxy and betray the intent—a failure mode called *reward hacking* that we study seriously in Chapter 17. Designing the reward is part of the modelling problem, not an afterthought.

1.4 Markov Decision Processes

The mathematical arena for reinforcement learning is the Markov decision process, which formalises the interaction loop as a controlled stochastic process.

Definition: Markov decision process

A **Markov decision process** (MDP) is a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma)$ where

- $\mathcal{S}$ is the state space and $\mathcal{A}$ the action space;
- $P(s' \mid s, a) = \mathbb{P}(S_{t+1} = s' \mid S_t = s, A_t = a)$ is the **transition kernel**;
- $R(s, a)$ (or $R(s, a, s')$) is the expected immediate reward;
- $\gamma \in [0, 1)$ is the discount factor.

The defining **Markov property** is that the transition and reward depend on the past only through the current state and action:

$$\mathbb{P}(S_{t+1}, R_{t+1} \mid S_0, A_0, \ldots, S_t, A_t) \;=\; \mathbb{P}(S_{t+1}, R_{t+1} \mid S_t, A_t).$$

The Markov property is what makes the subject tractable: the state is a *sufficient statistic* of history. Whatever happened before is fully compressed into s_t; optimal behaviour need only be a function of the present state. When the observation the agent receives is *not* Markov—when relevant information is hidden—the model generalises to a partially observable MDP (POMDP), in which the agent must maintain a belief distribution over states. In practice one often restores approximate Markovness by augmenting the observation: stacking recent frames, appending indicators, or (in deep RL) feeding history through a recurrent network.

The Abstract Viewpoint: MDPs as controlled Markov chains

Fix any policy π and marginalise out the actions. The pair process (S_t) becomes an ordinary Markov chain with transition matrix

$$P^{\pi}(s' \mid s) \;=\; \sum_{a} \pi(a \mid s)\, P(s' \mid s, a),$$

and expected reward vector $R^{\pi}(s) = \sum_a \pi(a \mid s) R(s, a)$. An MDP is therefore a *family* of Markov chains indexed by policies, and reinforcement learning is the problem of selecting

the best member of the family. Every question about a fixed policy—evaluation, stationary distributions, mixing—is classical Markov chain theory; the new mathematics enters only when we optimise over the family.

1.4.1 The objective

The agent seeks a policy maximising expected return. Define the **performance** of a policy by

$$J(\pi) \;=\; \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^{t} r_{t+1} \,\middle|\, s_0 \sim \mu_0\right],$$

where μ_0 is the initial-state distribution and the expectation is over trajectories generated by π interacting with P. The central problem of the subject is

$$\pi^{*} \;\in\; \arg\max_{\pi}\; J(\pi).$$

A remarkable theorem of MDP theory—proved in Chapter 2—asserts that for finite MDPs this maximum is attained, and attained simultaneously for every starting state, by a single deterministic stationary policy. The search over the bewildering space of history-dependent randomised behaviours collapses to a search over maps $\mathcal{S} \to \mathcal{A}$.

1.4.2 Episodic and continuing tasks

Tasks divide into *episodic* ones, which terminate in an absorbing state (a game ends, a maze is solved) and *continuing* ones, which run forever (process control, market making). Discounting unifies the two: an episodic task is a continuing one in which the terminal state loops to itself with zero reward. For continuing tasks an alternative criterion is the *average reward* $\lim_{T\to\infty} \frac{1}{T}\mathbb{E}[\sum_{t=1}^{T} r_t]$; we work throughout with the discounted criterion, which dominates practice.

1.5 Two Running Examples

Worked Example: Gridworld

A robot lives on a 4×4 grid. States are the sixteen cells; actions are $\{\text{up}, \text{down}, \text{left}, \text{right}\}$; moves into a wall leave the state unchanged. Cell $(4,4)$ is a terminal goal with reward $+1$; cell $(3,3)$ is a pit with reward -1; every other transition costs -0.04, so that dawdling is discouraged. With $\gamma = 0.99$ the optimal policy threads the shortest safe path to the goal. This tiny MDP—sixteen states, four actions—is small enough to solve exactly by every method in Parts I and II, and we shall do so repeatedly, so that each new algorithm can be checked against ground truth.

Worked Example: The recycling robot

A mobile robot collects cans. Its battery is *high* or *low* (two states). It may *search* (expected reward r_{search}, risking battery depletion), *wait* ($r_{\text{wait}} < r_{\text{search}}$, no risk), or, when low, *recharge*. Searching on a high battery keeps it high with probability α; searching on low avoids a rescue with probability β, and a rescue costs -3. The full kernel fits in a six-row

table, yet the model already exhibits the essential trade-off between short-term gain and long-term safety. We solve it exactly in Chapter 3.

1.6 Value Functions: A First Look

How should an agent judge a state? Not by its immediate reward—a state can pay nothing now and be the gateway to everything later—but by the total discounted reward obtainable from it. This is the *value function*, the single most important object in the subject; Chapter 2 is devoted to it. We record the definitions now so the vocabulary is available.

Definition: State value, action value, advantage

For a policy π:

$$V^\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] \qquad \text{(state-value function)}$$
$$Q^\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s,\ A_t = a] \qquad \text{(action-value function)}$$
$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s) \qquad \text{(advantage function)}$$

The two value functions are linked by $V^\pi(s) = \sum_a \pi(a \mid s)\, Q^\pi(s,a)$: the value of a state is the policy-weighted average of the values of the actions available in it. Consequently $\mathbb{E}_{a\sim\pi}[A^\pi(s,a)] = 0$: advantages measure each action *relative to the policy's own average*.

ELI5: Value, quality, advantage

V answers: "how good is it to be standing here?" Q answers: "how good is it to be standing here *and* take that particular door?" A answers: "is that door better or worse than my usual choice from here?" The third question is often the most useful one for learning, because it subtracts away everything about the state you cannot change and isolates the part the decision controls.

1.7 A Short History

The subject has three tributaries that merged only in the 1980s. The *optimal control* stream begins with Bellman's dynamic programming (1957) and Howard's policy iteration (1960): exact planning in known models, with the curse of dimensionality already diagnosed at birth. The *trial-and-error* stream descends from animal-learning psychology—Thorndike's law of effect (1911), that actions followed by satisfaction are strengthened—through early machine-learning experiments such as Samuel's checkers player (1959), which already contained a recognisable bootstrapped value update. The *temporal-difference* stream, drawing on both, was crystallised by Sutton (1988) and joined to control by Watkins's Q-learning (1989), with actor–critic architectures (Barto, Sutton, Anderson, 1983) supplying the two-learner pattern that still dominates.

The modern era has two inflection points. In 2013–2015, deep networks met Q-learning in DQN, and Atari from pixels announced that representation learning and reinforcement learning could be trained jointly and stably; AlphaGo (2016) and its successors added planning and self-play, and continuous control matured through DDPG, TRPO, PPO, and SAC. From 2022, reinforcement learning from human feedback became the finishing school of large language models,

and the frontier moved to reward modelling, preference optimisation, and production learning loops—the arc this monograph follows to its final chapters. It is a pleasing irony of the field that its newest triumphs run on its oldest mathematics: the policy gradient of 1992 and the Bellman recursion of 1957, at previously unimaginable scale.

1.8 The Taxonomy of Methods

The remainder of this monograph is organised along three axes, which cut across each other rather than nesting.

Axis	**One pole**	**Other pole**
What is learned	Value-based (learn Q, act greedily)	Policy-based (learn π_θ directly)
Model use	Model-free (learn from samples)	Model-based (learn/plan with P, R)
Data source	On-policy (own behaviour)	Off-policy (any behaviour)

Value-based methods (Chapters 3–6, 11) estimate value functions and derive the policy implicitly by acting greedily. Policy-based methods (Chapter 9) parameterise the policy and ascend the performance gradient. Actor–critic methods (Chapter 10) do both, and dominate modern practice. Model-based methods (Chapters 3, 14) exploit or learn the transition kernel and plan inside it; model-free methods never represent the kernel at all. On-policy methods learn about the policy that generates the data; off-policy methods learn about one policy from data generated by another—a distinction whose full subtlety occupies Chapter 7. A further axis, *online versus offline*, asks whether the agent may interact with the environment at all during learning, and organises Part V.

Key Takeaways

Reinforcement learning is learning from evaluative feedback in a sequential world. The MDP $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$ is the arena; the policy is the learner's behaviour; the return $G_t = \sum_k \gamma^k r_{t+k+1}$ is the objective; the value functions V, Q, A are the bookkeeping devices that make optimisation possible. The Markov property—state as sufficient statistic—is the structural assumption on which every algorithm in this book rests.

Chapter 2

Value Functions and the Bellman Equations

2.1 The Recursive Structure of Value

The return has a self-similar structure that the whole subject exploits. Peel off the first reward:

$$G_t \;=\; r_{t+1} + \gamma\big(r_{t+2} + \gamma r_{t+3} + \cdots\big) \;=\; r_{t+1} + \gamma\, G_{t+1}.$$

Taking conditional expectations under a policy π converts this one-line identity into a functional equation for the value function.

Theorem: Bellman expectation equations

For any policy π and all $s \in \mathcal{S}$, $a \in \mathcal{A}$:

$$V^\pi(s) = \sum_a \pi(a \mid s) \sum_{s'} P(s' \mid s, a)\big[R(s,a,s') + \gamma\, V^\pi(s')\big], \tag{2.1}$$

$$Q^\pi(s,a) = \sum_{s'} P(s' \mid s, a)\Big[R(s,a,s') + \gamma \sum_{a'} \pi(a' \mid s')\, Q^\pi(s',a')\Big]. \tag{2.2}$$

In compact operator form, $V^\pi = R^\pi + \gamma P^\pi V^\pi$, where P^π is the policy-averaged transition matrix of Chapter 1.

Proof. Condition $G_t = r_{t+1} + \gamma G_{t+1}$ on $S_t = s$ and expand the expectation over the first action $a \sim \pi(\cdot \mid s)$ and the first transition $s' \sim P(\cdot \mid s, a)$. By the Markov property, $\mathbb{E}_\pi[G_{t+1} \mid S_{t+1} = s'] = V^\pi(s')$ regardless of how s' was reached, which closes the recursion. The Q version is identical with the first action fixed. □

Equation (2.1) says: the value of a state equals the expected immediate reward plus the discounted expected value of wherever you land. It is a system of $|\mathcal{S}|$ linear equations in $|\mathcal{S}|$ unknowns—and therein lies its power. *Evaluating* a fixed policy is linear algebra:

$$V^\pi \;=\; (I - \gamma P^\pi)^{-1} R^\pi,$$

the inverse existing because $\gamma < 1$ makes every eigenvalue of γP^π lie strictly inside the unit circle (P^π is a stochastic matrix, so its spectral radius is 1).

ELI5: The travel-time recursion

How long does it take to drive home? However long it takes to reach the next junction, plus however long it takes from there. You do not need to know the whole route to write that down—only one step and a recursive call. The Bellman equation is exactly this: value here = one step of reward + (discounted) value there. All of reinforcement learning is the study

of this single sentence.

2.2 Optimality

Define the **optimal value functions**

$$V^*(s) = \max_\pi V^\pi(s), \qquad Q^*(s,a) = \max_\pi Q^\pi(s,a),$$

the maxima taken over all (possibly history-dependent, randomised) policies. A policy π^* is *optimal* if $V^{\pi^*}(s) = V^*(s)$ for every s.

Theorem: Bellman optimality equations

The optimal value functions satisfy, for all s, a:

$$V^*(s) = \max_a \sum_{s'} P(s' \mid s,a)\big[R(s,a,s') + \gamma\, V^*(s')\big], \tag{2.3}$$

$$Q^*(s,a) = \sum_{s'} P(s' \mid s,a)\Big[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\Big], \tag{2.4}$$

and any policy that is greedy with respect to Q^*, i.e. $\pi^*(s) \in \arg\max_a Q^*(s,a)$, is optimal.

The change from (2.1) to (2.3) looks small—an expectation over actions became a maximum—but it changes the mathematics fundamentally. The system is no longer linear; it cannot be solved by matrix inversion. It *can* be solved by fixed-point iteration, and proving this is the business of the next section.

2.3 The Bellman Operators are Contractions

Let $\mathcal{B}(\mathcal{S})$ denote the space of bounded functions $V : \mathcal{S} \to \mathbb{R}$ equipped with the supremum norm $\|V\|_\infty = \sup_s |V(s)|$. This is a complete metric space. Define two operators on it:

$$(\mathcal{T}^\pi V)(s) = \sum_a \pi(a \mid s) \sum_{s'} P(s' \mid s,a)\big[R(s,a,s') + \gamma V(s')\big] \quad \text{(Bellman expectation operator)}$$

$$(\mathcal{T} V)(s) = \max_a \sum_{s'} P(s' \mid s,a)\big[R(s,a,s') + \gamma V(s')\big] \quad \text{(Bellman optimality operator)}$$

The Bellman equations say precisely that V^π is a fixed point of $\mathcal{T}^\pi$ and V^* a fixed point of $\mathcal{T}$.

Theorem: γ-contraction

For all $U, V \in \mathcal{B}(\mathcal{S})$,

$$\|\mathcal{T}U - \mathcal{T}V\|_\infty \;\le\; \gamma\, \|U - V\|_\infty ,$$

and the same holds for $\mathcal{T}^\pi$. Consequently, by the Banach fixed-point theorem, each operator has a *unique* fixed point, and iterating from any starting function converges to it geometrically: $\|\mathcal{T}^k V_0 - V^*\|_\infty \le \gamma^k \|V_0 - V^*\|_\infty$.

Proof. Fix a state s and write $q_U(a) = \sum_{s'} P(s' \mid s,a)[R + \gamma U(s')]$, similarly q_V. For any a,

$$|q_U(a) - q_V(a)| = \gamma \Big| \sum_{s'} P(s' \mid s,a)\,[U(s') - V(s')] \Big| \le \gamma \, \|U - V\|_\infty ,$$

since $P(\cdot \mid s,a)$ is a probability distribution. The elementary inequality $|\max_a q_U(a) - \max_a q_V(a)| \le \max_a |q_U(a) - q_V(a)|$ then gives $|(\mathcal{T}U)(s) - (\mathcal{T}V)(s)| \le \gamma \, \|U - V\|_\infty$ for every s; take the supremum over s. For $\mathcal{T}^\pi$, replace the maximum by the policy average, which is again a convex combination and obeys the same bound. □

The Abstract Viewpoint: Why completeness and contraction are the whole story

The pair (complete metric space, contraction) is one of the great workhorses of analysis: it proves Picard's theorem for ODEs, the inverse function theorem, and the existence of fractals as attractors of iterated function systems. Reinforcement learning simply adds one more client. The discount factor γ is not an incidental modelling choice but the Lipschitz constant of the whole theory: existence, uniqueness, and the convergence rate of every dynamic-programming algorithm all trade through it. As $\gamma \uparrow 1$ the contraction weakens, horizons lengthen, and every algorithm in this book slows down—a quantitative statement of the qualitative truth that far-sighted planning is hard.

Two corollaries deserve emphasis.

Theorem: Existence of an optimal deterministic stationary policy

For a finite MDP there exists a deterministic stationary policy π^* with $V^{\pi^*} = V^*$. *Sketch:* let V^* be the unique fixed point of $\mathcal{T}$ and let $\pi^*(s)$ pick any maximising action in (2.3). Then $\mathcal{T}^{\pi^*} V^* = \mathcal{T} V^* = V^*$, so V^* is the fixed point of $\mathcal{T}^{\pi^*}$, i.e. $V^{\pi^*} = V^*$.

Theorem: Monotonicity

If $U \le V$ pointwise then $\mathcal{T}U \le \mathcal{T}V$ and $\mathcal{T}^\pi U \le \mathcal{T}^\pi V$. Moreover $\mathcal{T}$ shifts constants: $\mathcal{T}(V + c\mathbf{1}) = \mathcal{T}V + \gamma c \mathbf{1}$.

Monotonicity plus the constant-shift property give an alternative, expectation-free proof of convergence and underlie the error bounds of approximate dynamic programming: if a value estimate is wrong by at most ε in sup norm, the greedy policy extracted from it loses at most $2\gamma\varepsilon/(1-\gamma)$ in value—a bound we use when function approximation enters in Chapter 8.

2.4 Worked Example: Solving a Two-State MDP by Hand

Worked Example: Two states, closed form

Let $\mathcal{S} = \{A, B\}$ with one action in each state (so evaluation is the whole problem). From A: reward 2, go to B. From B: reward 0; stay in B with probability $\tfrac{1}{2}$, return to A with probability $\tfrac{1}{2}$. Take $\gamma = 0.9$. The Bellman system is

$$V(A) = 2 + 0.9\,V(B), \qquad V(B) = 0 + 0.9\big(\tfrac{1}{2}V(B) + \tfrac{1}{2}V(A)\big).$$

Substituting the first into the second: $V(B) = 0.45V(B) + 0.45(2 + 0.9V(B)) = 0.855\,V(B) +$

0.9, so $V(B) = 0.9/0.145 \approx 6.2069$ and $V(A) = 2 + 0.9 \times 6.2069 \approx 7.5862$.
Now verify the contraction numerically. Starting from $V_0 = (0, 0)$ and iterating $V_{k+1} = \mathcal{T}V_k$:

k	0	1	2	3	5	10
$V_k(A)$	0	2.000	2.000	2.810	4.303	6.318
$V_k(B)$	0	0.000	0.900	1.305	2.685	4.807
$\|V_k - V^*\|_\infty$	7.59	6.21	5.59	4.90	3.52	1.78

The error contracts by roughly $\gamma = 0.9$ per step once transients die out, exactly as the theorem promises. Slow—and that slowness at γ near 1 is real, not an artefact.

2.5 The Geometry of Policy Space

A theme that will recur when we reach policy gradients deserves planting now. The set of value functions of all policies, $\{V^\pi : \pi \text{ a policy}\} \subset \mathbb{R}^{|\mathcal{S}|}$, is in general a non-convex object (a polytope with curved faces for stochastic policies), and V^* sits at its "upper-right" corner: it dominates every other value function *coordinate-wise*. This is stronger than merely maximising the scalar $J(\pi)$—the optimal policy is best from every start state simultaneously. Value-based methods climb toward this corner through the space of value functions; policy-based methods climb through the space of policies. Both are ascending the same partially ordered landscape, a fact made precise by the policy improvement theorem of the next chapter.

2.6 Return, Value, Advantage: The Working Trinity

Because the three quantities G, V (or Q), and A are so easily conflated, we close with a comparative summary that the reader may wish to bookmark; the deep-RL chapters lean on these distinctions constantly.

	Return G_t	**Value** V, Q	**Advantage** A
What it is	A random variable: realised discounted reward of one trajectory	An expectation of G given state (and action)	Difference $Q(s,a) - V(s)$
Known when	After the trajectory	Must be estimated	Must be estimated
Bias / variance	Unbiased, high variance	Biased if approximate, low variance	Centred: mean zero under π
Typical use	Monte Carlo targets	Bootstrapped targets, critics	Policy-gradient weights

ELI5: Restaurant edition

The *return* is the actual satisfaction of the meal you ate tonight. The *value* of a restaurant is how satisfying a meal there is on average. The *advantage* of ordering the biryani is how much better the biryani is than whatever you usually order there. One is an outcome, one is an expectation, one is a comparison.

Key Takeaways

The Bellman expectation equation is a linear fixed-point equation characterising V^π; the Bellman optimality equation is a nonlinear one characterising V^*. Both operators are γ-contractions in sup norm, so both fixed points exist, are unique, and are reachable by iteration at geometric rate γ. A greedy policy with respect to Q^* is optimal, and an optimal deterministic stationary policy always exists in finite MDPs. Every algorithm in this monograph is a scheme—exact, sampled, or approximated—for computing one of these two fixed points.

Chapter 3

Dynamic Programming

3.1 Planning with a Known Model

Suppose—for this chapter only—that the agent is handed the complete MDP: the kernel P and reward R are known exactly. No learning is required; the task is pure computation, called *planning*. Dynamic programming (DP), introduced by Bellman in the 1950s, converts the fixed-point theory of Chapter 2 into algorithms. Everything later in the book is a sampled or approximated descendant of the two algorithms of this chapter, so it repays careful study even though the "known model" assumption is rarely met in practice.

3.2 Policy Evaluation

Given a policy π, compute V^π. We saw the direct solution $V^\pi = (I - \gamma P^\pi)^{-1} R^\pi$; for large $\mathcal{S}$, inverting an $|\mathcal{S}| \times |\mathcal{S}|$ matrix is prohibitive, and iteration is preferred.

Algorithm: Iterative policy evaluation

Input: policy π, tolerance ε.
Initialise V_0 arbitrarily (e.g. $V_0 \equiv 0$).
Repeat for $k = 0, 1, 2, \ldots$: $V_{k+1} \leftarrow \mathcal{T}^\pi V_k$, i.e. for every s,

$$V_{k+1}(s) \;=\; \sum_{a} \pi(a \mid s) \sum_{s'} P(s' \mid s, a) \big[R(s, a, s') + \gamma V_k(s') \big]$$

until $\|V_{k+1} - V_k\|_\infty < \varepsilon$.
Guarantee: $\|V_k - V^\pi\|_\infty \le \gamma^k \, \|V_0 - V^\pi\|_\infty$, and the stopping rule ensures the final error is at most $\varepsilon\gamma/(1-\gamma)$.

Each pass over the state space is a *full backup*: every state's value is refreshed from all its successors, weighted by exact probabilities. This is the operation that sampling methods (Part II) will imitate with single random successors. In-place (Gauss–Seidel) sweeps, which reuse freshly updated values within the same pass, also converge and usually faster.

3.3 Policy Improvement

Evaluation tells us how good π is. The next theorem tells us how to do better, and is arguably the most consequential single result in the subject: it is the licence behind every "act greedily" step in every algorithm from here to PPO.

Theorem: Policy improvement theorem

Let π be any policy and define the greedy policy

$$\pi'(s) \in \arg\max_a Q^\pi(s,a).$$

Then $V^{\pi'}(s) \geq V^\pi(s)$ for every s; and if π' is not strictly better in some state, then π already satisfies the Bellman optimality equation and is optimal.

Proof. By construction $Q^\pi(s,\pi'(s)) = \max_a Q^\pi(s,a) \geq \sum_a \pi(a \mid s) Q^\pi(s,a) = V^\pi(s)$ for all s; in operator language, $\mathcal{T}^{\pi'} V^\pi \geq V^\pi$. Apply $\mathcal{T}^{\pi'}$ repeatedly; monotonicity of $\mathcal{T}^{\pi'}$ preserves the inequality at every application, giving the increasing chain $V^\pi \leq \mathcal{T}^{\pi'} V^\pi \leq (\mathcal{T}^{\pi'})^2 V^\pi \leq \cdots \to V^{\pi'}$, where the limit is the fixed point of the contraction $\mathcal{T}^{\pi'}$. Hence $V^{\pi'} \geq V^\pi$. If equality holds everywhere, then $V^\pi = \mathcal{T} V^\pi$ (the greedy improvement changed nothing, so the max is attained by π), and uniqueness of the fixed point of $\mathcal{T}$ gives $V^\pi = V^*$. □

ELI5: One honest look ahead

Suppose you have a routine you follow every day and an honest score for how well life goes under the routine. Now, in each situation, ask: "if I deviate for just one step—take the best-looking single action—and then go back to my routine, do I do better?" If yes anywhere, adopting that deviation everywhere can only help. Keep repeating. The theorem says this humble procedure never goes backwards and can only stall at perfection.

3.4 Policy Iteration and Value Iteration

Alternating full evaluation with greedy improvement yields the first complete planning algorithm.

Algorithm: Policy iteration (Howard, 1960)

Initialise π_0 arbitrarily.
Repeat for $k = 0, 1, 2, \ldots$:

1. **Evaluation:** compute V^{π_k} (solve the linear system or iterate to tolerance).
2. **Improvement:** $\pi_{k+1}(s) \leftarrow \arg\max_a \sum_{s'} P(s' \mid s,a)[R(s,a,s') + \gamma V^{\pi_k}(s')]$.

until $\pi_{k+1} = \pi_k$.

Because there are finitely many deterministic policies ($|\mathcal{A}|^{|\mathcal{S}|}$ of them) and each iteration produces a strictly better one until optimality, policy iteration terminates in finitely many steps—in practice astonishingly few, often under a dozen even for large state spaces. Its cost per iteration, however, is a full policy evaluation.

Value iteration compresses the evaluation step to a single sweep:

Algorithm: Value iteration

Initialise V_0 arbitrarily.

Repeat: $V_{k+1} \leftarrow \mathcal{T}V_k$, i.e.

$$V_{k+1}(s) \;=\; \max_a \sum_{s'} P(s' \mid s,a)\big[R(s,a,s') + \gamma V_k(s')\big] \quad \forall s,$$

until $\|V_{k+1} - V_k\|_\infty < \varepsilon$; then output the greedy policy of the final V.

Convergence at rate γ is immediate from the contraction theorem. Value iteration can be read as policy iteration with evaluation truncated to one sweep, and the whole spectrum in between—evaluate for m sweeps, then improve—is called *modified policy iteration*. All converge; the trade is fewer, costlier iterations against more, cheaper ones.

The Abstract Viewpoint: Generalised policy iteration

Abstract the pattern: maintain a value estimate V and a policy π; let two processes run, one dragging V toward V^π (evaluation), one dragging π toward greediness with respect to V (improvement). The processes compete—improving π invalidates the evaluation; evaluating better exposes new improvements—yet their joint fixed point is exactly (π^*, V^*), where both pressures vanish simultaneously. Sutton and Barto call this schema *generalised policy iteration* (GPI). Its value is taxonomic: Monte Carlo control, SARSA, Q-learning, actor–critic, DQN, and PPO are all instances of GPI in which evaluation is sampled, partial, approximate, or all three. When we meet each algorithm, the first question to ask is always: *how does it evaluate, and how does it improve?*

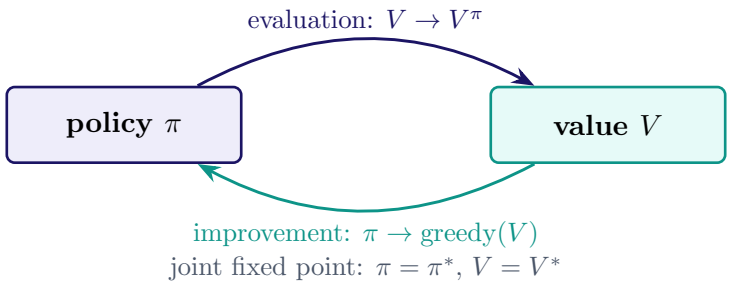

Figure 3.1: Generalised policy iteration. Two coupled processes—one dragging the value estimate toward the current policy's truth, one dragging the policy toward greediness with respect to the current estimate—stabilise only at optimality. Every algorithm in this monograph is an instance, distinguished by how coarsely each arrow is executed.

3.5 Worked Example: The Recycling Robot Solved

Worked Example: Recycling robot, exact solution

Take $\alpha = 0.7$ (search on high stays high), $\beta = 0.6$ (search on low avoids rescue), $r_{\text{search}} = 5$, $r_{\text{wait}} = 1$, rescue reward -3, $\gamma = 0.9$. States $\{h, \ell\}$.
The candidate policies for serious comparison are π_1 = (search, recharge) and π_2 = (search,

search). For π_1:

$$V(h) = 5 + 0.9\,[0.7\,V(h) + 0.3\,V(\ell)],$$
$$V(\ell) = 0 + 0.9\,V(h),$$

giving $V(h) = 5 + 0.63V(h) + 0.27 \cdot 0.9V(h) = 5 + 0.873\,V(h)$, so $V(h) = 5/0.127 \approx 39.37$, $V(\ell) \approx 35.43$.
For π_2: searching on low earns $0.6[5+0.9V(\ell)]+0.4[-3+0.9V(h)]$. Solving the 2×2 system yields $V(h) \approx 38.19$, $V(\ell) \approx 35.79$.
Neither dominates: π_1 is better from h, π_2 from ℓ. Policy iteration resolves this by mixing: starting from π_1, the improvement step at ℓ compares recharge ($0+0.9V(h) = 35.43$) against search ($0.6[5+0.9\cdot 35.43]+0.4[-3+0.9\cdot 39.37] \approx 35.90$) and switches ℓ to search; re-evaluating and improving once more switches nothing, and the optimal policy is: *search in both states*, with $V^*(h) \approx 38.19$... but wait—improvement at h under the new value function must also be checked (wait gives $1+0.9\cdot 38.19 = 35.4 < 38.2$; search stands). The point of the exercise: local comparisons of full policies can be inconclusive, while policy iteration's statewise greedy step sorts everything out mechanically, and stalls only at the true optimum.

3.6 Asynchronous DP and the Curse of Dimensionality

Full sweeps are unnecessary. *Asynchronous* DP updates states in any order, even one at a time, and still converges provided every state continues to be visited—a fact that follows because each individual backup is still a coordinate application of a sup-norm contraction. Prioritised sweeping exploits this freedom by updating first the states whose Bellman error is largest, propagating changes backward through the model like ripples.

The true limitation of DP is not sweep order but size. The backup at one state costs $O(|\mathcal{A}| \cdot |\mathcal{S}|)$; a sweep costs $O(|\mathcal{A}| \cdot |\mathcal{S}|^2)$ in the dense case. Chess has more states than atoms in the observable universe; a robot's sensor space is continuous. Bellman himself named this the *curse of dimensionality*: state spaces grow exponentially in the number of state variables. Two escapes exist, and the rest of the book pursues both:

1. **Sample** instead of sweep: back up only along experienced transitions (Part II);

2. **Generalise** instead of tabulate: represent V or π by a parametric function that shares information across states (Parts III–IV).

Caution

DP's guarantees lean on *exact* expectations under a *correct* model. Both escapes surrender something: sampling introduces variance, and generalisation introduces bias. The interplay of sampling, bootstrapping, and function approximation—each benign alone—can destabilise learning entirely (the "deadly triad" of Chapter 8). Keeping DP's clean picture in mind is the best vaccine against confusion later.

Key Takeaways

With a known model, policy evaluation is a linear fixed-point computation and control is achieved by two convergent schemes: policy iteration (full evaluation, then greedy improvement; finite termination) and value iteration (repeated application of the optimality operator; geometric convergence at rate γ). The policy improvement theorem guarantees greedy steps never hurt. Generalised policy iteration—evaluation and improvement interleaved at any granularity—is the master pattern that every subsequent algorithm instantiates. DP fails only where it cannot reach: unknown models and enormous state spaces, which are precisely the subjects of the remaining chapters.

Part II

Learning from Experience

Chapter 4

Monte Carlo Methods

4.1 Removing the Model

Dynamic programming needs P and R; real agents rarely have them. The oldest remedy is the frequentist's: if you cannot compute an expectation, *sample* it. Monte Carlo (MC) methods estimate value functions by averaging observed returns over complete episodes, requiring nothing from the environment except the ability to interact with it (or with a simulator). They are the conceptual bridge between planning and learning: same fixed points, new estimator.

Definition: Monte Carlo value estimate

Let s be visited in a set of episodes generated by π, and let $G^{(1)}, \dots, G^{(N(s))}$ be the returns observed following those visits. The MC estimate is the sample mean

$$\widehat{V}(s) \;=\; \frac{1}{N(s)} \sum_{i=1}^{N(s)} G^{(i)} \;\xrightarrow[N(s)\to\infty]{\text{a.s.}}\; V^{\pi}(s)$$

by the strong law of large numbers, since each $G^{(i)}$ is an unbiased draw with mean $V^{\pi}(s)$.

Two bookkeeping variants exist. *First-visit* MC averages returns only from the first visit to s in each episode; the returns are then i.i.d. across episodes and the classical theory applies verbatim. *Every-visit* MC averages over all visits; the samples are correlated within an episode, the estimator is biased at finite N but consistent, and in practice the two behave similarly. Incrementally, with $N(s)$ a visit counter,

$$\widehat{V}(s) \;\leftarrow\; \widehat{V}(s) + \frac{1}{N(s)}\big(G - \widehat{V}(s)\big),$$

or with a constant step α in nonstationary settings—an update template, *estimate* $\leftarrow$ *estimate* + *step* $\times$ *(target* $-$ *estimate)*, that every learning rule in this book will reuse with different targets.

ELI5: Judging a restaurant by eating there

DP is reading the entire recipe book and the supplier's ledgers to deduce how good a restaurant must be. Monte Carlo is simpler: eat there many times and average your experiences. You need no access to the kitchen. The price is patience—you only learn after whole meals (episodes), and your average wobbles until you have eaten there often.

4.2 Properties: No Bootstrapping, No Bias, High Variance

MC targets are actual returns; no estimate is built on another estimate. Three consequences follow.

1. **Unbiasedness.** $\mathbb{E}[G_t \mid S_t = s] = V^\pi(s)$ exactly. MC converges to the true value function even under function approximation (Chapter 8), a robustness bootstrapping methods lack.
2. **High variance.** G_t sums many random rewards; its variance compounds over the episode length. Estimates are noisy and convergence slow: error decays as $O(1/\sqrt{N})$ per state.
3. **Episodic requirement.** The return is known only when the episode ends. Continuing tasks, or tasks with very long episodes, starve MC of targets. Independence of the Markov property is the compensating virtue: MC never uses the transition structure, so it is unharmed when the state is not truly Markov.

4.3 Monte Carlo Control

To improve behaviour, not merely evaluate it, we slot MC evaluation into generalised policy iteration. Two obstacles arise, and their solutions introduce ideas that persist throughout the subject.

Obstacle 1: improvement needs Q, not V. Without a model, the greedy step $\arg\max_a \sum_{s'} P(\cdot)[\cdot]$ cannot be computed from V alone. Solution: estimate $Q^\pi(s,a)$ directly—average returns following each state–action pair. Greedy improvement is then model-free: $\pi'(s) = \arg\max_a \widehat{Q}(s,a)$.

Obstacle 2: exploration. A deterministic policy visits only the pairs it chooses; the estimates of unchosen actions never improve, so the greedy step is comparing fresh estimates with stale ones. Two remedies:

- *Exploring starts*: begin each episode at a uniformly random (s,a). Clean in theory, unavailable when starts cannot be dictated.
- *ε-soft policies*: require $\pi(a \mid s) \geq \varepsilon/|\mathcal{A}|$ for every action—most simply, the *ε-greedy* policy that takes the greedy action with probability $1-\varepsilon$ and a uniform random action otherwise. GPI then converges to the best ε-soft policy, which approaches optimality as $\varepsilon \downarrow 0$.

Algorithm: On-policy first-visit MC control, ε-greedy

Initialise $Q(s,a)$ arbitrarily; $\pi \leftarrow \varepsilon$-greedy in Q; returns lists empty.
Loop forever:

1. Generate an episode $s_0, a_0, r_1, \ldots, s_{T-1}, a_{T-1}, r_T$ following π.
2. $G \leftarrow 0$. For $t = T-1$ down to 0: $G \leftarrow \gamma G + r_{t+1}$; if (s_t, a_t) is a first visit, append G to its list and set $Q(s_t, a_t)$ to the list average.
3. Refresh π to be ε-greedy in the updated Q.

Worked Example: Blackjack

The canonical MC showcase. States: player sum (12–21), dealer's showing card, usable-ace flag—200 states; actions: hit or stick; reward ± 1 or 0 at the end. The transition kernel, though it exists, is a combinatorial nuisance to write down; *sampling* episodes is trivial. MC control with exploring starts converges to the textbook-optimal strategy (stick high, hit low, with the usable-ace sheet differing exactly where intuition says it should) after a few hundred thousand simulated hands—a computation of seconds. The example makes the general point: whenever a simulator is cheap and the model is awkward, MC is the method

of least resistance.

4.4 Off-Policy Prediction: A First Encounter

Suppose episodes are generated by a *behaviour* policy b, but we wish to estimate V^π for a different *target* policy π (perhaps the greedy one, while b keeps exploring). The instrument is **importance sampling**: reweight each return by the relative likelihood of its trajectory,

$$\rho_{t:T-1} \;=\; \prod_{k=t}^{T-1} \frac{\pi(a_k \mid s_k)}{b(a_k \mid s_k)}, \qquad \widehat{V}_{\mathrm{IS}}(s) \;=\; \frac{1}{N}\sum_{i} \rho^{(i)}\, G^{(i)}.$$

This estimator is unbiased whenever b covers π (i.e. $\pi(a \mid s) > 0 \Rightarrow b(a \mid s) > 0$), but its variance can be enormous—indeed infinite—because the ratio is a *product* over the trajectory. The *weighted* variant $\sum_i \rho^{(i)} G^{(i)} / \sum_i \rho^{(i)}$ trades a vanishing bias for dramatically lower variance and is preferred in practice. We defer the full treatment to Chapter 7; here it suffices to note that the tension— unbiasedness versus variance in trajectory reweighting—is the central difficulty of off-policy learning, and one of the reasons modern policy-optimisation methods (PPO, Chapter 12) go to such lengths to keep data *nearly* on-policy.

The Abstract Viewpoint: MC as quadrature

Estimating $V^\pi(s)$ is computing an integral over trajectory space: $V^\pi(s) = \int G(\tau)\, p_\pi(\mathrm{d}\tau \mid s_0 = s)$. Monte Carlo is plain quadrature by sampling from p_π; importance sampling is a change of measure $\mathrm{d}p_\pi = \rho\, \mathrm{d}p_b$. Every variance-reduction device of classical simulation theory—control variates, stratification, common random numbers—has an RL incarnation. In particular, the *baseline* of policy-gradient methods (Chapter 9) is precisely a control variate, and recognising it as such demystifies why subtracting $V(s)$ leaves the estimator unbiased.

Key Takeaways

Monte Carlo methods estimate value functions by averaging complete-episode returns: model-free, unbiased, conceptually minimal, but high-variance and confined to episodic tasks. MC control couples Q-estimation with ε-greedy improvement inside GPI. Off-policy estimation via importance sampling is possible but pays in variance through the product of likelihood ratios. The natural question—can we update *before* the episode ends, using our own estimates as scaffolding?—is answered by temporal-difference learning, the subject of the next chapter.

Chapter 5

Temporal-Difference Learning

5.1 The Central Idea of Reinforcement Learning

If one idea is distinctive to reinforcement learning—owing nothing to supervised learning on one side or classical control on the other—it is temporal-difference (TD) learning. TD marries the two methods we have: like Monte Carlo it learns from raw experience without a model; like dynamic programming it *bootstraps*, updating estimates from other estimates without waiting for final outcomes.

Recall the MC update toward the sampled return, $V(s_t) \leftarrow V(s_t) + \alpha\,[\,G_t - V(s_t)\,]$. TD replaces the remainder of the return, γG_{t+1}, by its current estimate $\gamma V(s_{t+1})$:

Definition: TD(0) update and TD error

After observing (s_t, r_{t+1}, s_{t+1}):

$$V(s_t) \;\leftarrow\; V(s_t) + \alpha\,\delta_t, \qquad \delta_t \;=\; r_{t+1} + \gamma V(s_{t+1}) - V(s_t).$$

The quantity δ_t is the **TD error**: the discrepancy between the current estimate $V(s_t)$ and the one-step-improved **TD target** $r_{t+1} + \gamma V(s_{t+1})$.

The update is fully online and incremental: one transition, one update, no waiting for episode end, no model. The TD error will follow us for the rest of the book—it reappears as the critic signal in actor–critic methods, as the regression residual in DQN, and (averaged) as the advantage estimate in PPO.

ELI5: Updating your commute forecast at the first junction

You predict the drive home takes 30 minutes. Ten minutes in, you reach the highway and traffic is flowing; from here it usually takes 15. You revise your total to 25 *now*—you do not wait to reach home to learn. You corrected one prediction using a later, better-informed prediction. That is bootstrapping, and the 5-minute revision is the TD error.

5.2 Why It Works: Stochastic Approximation

The TD target is biased—it contains the current, possibly wrong, $V(s_{t+1})$—so the law of large numbers does not apply directly. The correct framework is stochastic approximation: TD(0) is a Robbins–Monro scheme for the fixed-point equation $V = \mathcal{T}^\pi V$, using one sampled transition per step in place of the exact expectation.

Theorem: Convergence of tabular TD(0)

Suppose every state is visited infinitely often and the step sizes satisfy the Robbins–Monro conditions $\sum_t \alpha_t = \infty$, $\sum_t \alpha_t^2 < \infty$. Then tabular TD(0) converges to V^π with probability one. The engine of the proof is that the expected update direction is $\mathcal{T}^\pi V - V$, and $\mathcal{T}^\pi$ is a γ-contraction: stochastic approximation theory (Robbins–Monro; Tsitsiklis's asynchronous convergence framework) converts contraction of the mean field into almost sure convergence of the noisy iterates.

Note the division of labour: the *contraction* (Chapter 2) fixes where the algorithm goes; *stochastic approximation* handles the fact that we travel there on sampled, noisy steps. This pattern—identify the operator, verify contraction or monotonicity, invoke SA theory—is the standard proof scheme for tabular RL, and it covers SARSA and Q-learning in the next chapter with only cosmetic change.

5.3 TD versus MC: The Bias–Variance Trade

	Monte Carlo	**TD(0)**
Target	G_t (actual return)	$r_{t+1} + \gamma V(s_{t+1})$
Bias	None	Yes, while V is wrong
Variance	High (whole trajectory)	Low (one reward, one state)
Updates	At episode end	Every step; works on continuing tasks
Markov property	Not used	Exploited
Sensitivity to init	None	Converges from any init, but transient depends on it

In practice TD usually learns faster on Markov tasks: exploiting the state structure buys a large variance reduction that outweighs the transient bias. A sharper statement comes from the *batch* setting. Given a fixed finite set of episodes and trained to convergence on that data, MC converges to the least-squares fit of observed returns, whereas TD converges to the value function of the *maximum-likelihood Markov model* implied by the data—the *certainty-equivalence* estimate. If the world really is Markov, TD's answer generalises better from the same evidence; it squeezes the Markov assumption for every drop it is worth.

Worked Example: The two-state batch puzzle

Data: eight episodes over states $\{A, B\}$. One episode: $A, 0, B, 0$ (visit A, reward 0, then B, reward 0, terminate). Six episodes: $B, 1$. One episode: $B, 0$. What are $V(A), V(B)$?
Everyone agrees $V(B) = 6/8 = 0.75$. For $V(A)$, MC says: the only return ever observed from A is 0, so $V(A) = 0$. TD (batch) says: A transitioned to B with certainty and zero reward in all observed data, so $V(A) = 0 + \gamma\, V(B) = 0.75$ (take $\gamma = 1$). If the process is Markov, TD's answer is plainly the better inference—the single unlucky episode through A should not overrule what we know about B. This tiny example is the cleanest available intuition for why bootstrapping helps.

5.4 n-Step Returns and TD(λ)

MC and TD(0) are endpoints of a spectrum. Define the **n-step return**

$$G_t^{(n)} \;=\; r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^n \, V(s_{t+n}),$$

which trusts real rewards for n steps and bootstraps thereafter: $n = 1$ is TD(0), $n = \infty$ (to termination) is MC. Intermediate n often outperforms both ends—enough reality to cut bias, enough bootstrapping to cut variance.

Rather than pick one n, TD(λ) averages them all geometrically.

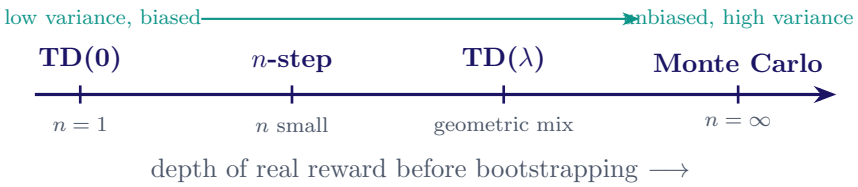

Figure 5.1: The prediction spectrum. Trust sampled rewards for longer before bootstrapping and bias falls while variance rises; TD(λ) places a geometric prior over the whole spectrum.

Definition: λ-return

For $\lambda \in [0, 1]$,

$$G_t^{\lambda} \;=\; (1-\lambda) \sum_{n=1}^{\infty} \lambda^{\,n-1} \, G_t^{(n)},$$

with the convention that returns beyond termination equal G_t. At $\lambda = 0$ this is the one-step target; at $\lambda = 1$, the Monte Carlo return. The weights $(1-\lambda)\lambda^{n-1}$ sum to one.

The λ-return defines the *forward view*: a target looking into the future, implementable only at episode end. The celebrated *backward view* implements it online with **eligibility traces**: a memory $e(s)$ per state, decayed by $\gamma\lambda$ each step and bumped at visits,

$$e_t(s) = \gamma\lambda \, e_{t-1}(s) + \mathbb{1}\{s_t = s\}, \qquad V(s) \leftarrow V(s) + \alpha \, \delta_t \, e_t(s) \quad \forall s.$$

Each TD error is broadcast backwards to every recently visited state, in proportion to how recently and frequently it was visited. For offline updates the two views are exactly equivalent; online, modern "true online TD(λ)" variants restore exact equivalence. The same machinery underlies GAE, the advantage estimator of Chapter 10—which is nothing but a λ-return applied to advantages—so the reader's investment here pays off precisely where modern practice lives.

The Abstract Viewpoint: One family of operators

Define the λ-operator $\mathcal{T}_\lambda^\pi = (1-\lambda)\sum_{n\geq 1} \lambda^{n-1} (\mathcal{T}^\pi)^n$, a geometric average of iterated Bellman operators. Each $(\mathcal{T}^\pi)^n$ contracts with modulus γ^n, so $\mathcal{T}_\lambda^\pi$ contracts with modulus $\gamma(1-\lambda)/(1-\gamma\lambda) \leq \gamma$: *larger λ contracts harder per application* but is estimated with more variance from data. The bias–variance dial of λ is thus visible already at the operator level, before any sampling: TD(λ) interpolates not just between two algorithms but between two fixed-point iterations of differing strength.

Caution

Step-size discipline matters. Theory demands decaying α_t; practice, especially nonstationary practice, uses small constant steps and accepts steady-state noise of order $O(\alpha)$. With traces, effective step sizes multiply by trace magnitudes, and instability at large $\alpha\lambda$ is common. When TD misbehaves, the first knobs to inspect are α, λ, and reward scaling—in that order.

Key Takeaways

TD learning updates value estimates from other value estimates, one transition at a time, guided by the TD error $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$. It is a stochastic approximation to the Bellman fixed point: biased in transit, convergent in the limit, and usually faster than MC on Markov tasks. The n-step and λ-return constructions interpolate continuously between TD(0) and MC, with eligibility traces as the online mechanism. The TD error is the most reusable object in reinforcement learning; the reader will meet it in every remaining chapter.

Chapter 6

Model-Free Control: SARSA, Q-Learning, and Exploration

6.1 From Prediction to Control

Chapter 5 evaluated a fixed policy. Control asks for the optimal one, and the recipe is again GPI: estimate action values by TD, improve greedily, explore enough to keep the estimates honest. Two archetypal algorithms result, differing in a single term of their update, and the difference between them—on-policy versus off-policy—organises much of modern deep RL.

6.2 SARSA: On-Policy TD Control

Apply TD(0) to action values along the agent's own behaviour. The data for one update is the quintuple $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$— State, Action, Reward, State, Action—whence the name.

Definition: SARSA update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[\, r_{t+1} + \gamma\, Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \,\big],$$

where a_{t+1} is the action *actually taken* in s_{t+1} by the current (e.g. ε-greedy) policy.

SARSA evaluates the policy it follows—exploration included. If the behaviour is ε-greedy, SARSA's Q converges to the value of the ε-greedy policy; letting $\varepsilon \to 0$ on an appropriate schedule (GLIE: *greedy in the limit with infinite exploration*) yields convergence to Q^* under the usual step-size conditions.

6.3 Q-Learning: Off-Policy TD Control

Watkins's Q-learning (1989) makes one change: bootstrap not from the action taken, but from the *best* action available.

Definition: Q-learning update

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\big[\, r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \,\big].$$

The target is a sampled version of the Bellman *optimality* backup (2.4). Consequently Q-learning estimates Q^* directly, *regardless of the policy generating the data*, provided that policy keeps visiting all pairs. It is off-policy by construction: the behaviour may be exploratory, sluggish, even random; the learned Q still converges to the optimal one (tabular case, Robbins–Monro steps, infinite visitation—the Watkins–Dayan theorem).

ELI5: Two ways to learn from a wobbly bicycle ride

SARSA asks: "given that I sometimes wobble at random, how good is each road *for a wobbly rider like me*?" It rates roads by what actually happens to it. Q-learning asks: "how good would each road be for a perfect rider?"—even while riding wobbly. SARSA is the realist, learning the value of its actual conduct; Q-learning is the idealist, learning the value of perfection while behaving imperfectly.

Worked Example: The cliff walk

A gridworld runs along the top of a cliff: start at one end, goal at the other, each step -1, falling off the cliff -100 and back to start. Behaviour is ε-greedy with fixed $\varepsilon = 0.1$. Q-learning learns the *optimal* path—hugging the cliff edge—but, because it still explores, occasionally steps off and suffers; its online return per episode averages around -70 in the classic experiment. SARSA learns the *safe* path a few cells from the edge, one step longer but robust to exploratory wobble, and earns a better online return (about -40). Neither is "wrong": Q-learning has found the optimal greedy policy; SARSA has found the best policy *for an agent that explores*. Which you want depends on whether mistakes during learning cost real money—a question a systematic trader will recognise instantly.

6.4 Expected SARSA and the Unifying View

Replace SARSA's sampled next action by its expectation under the policy:

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\Big[r_{t+1} + \gamma \sum_{a'} \pi(a' \mid s_{t+1})\, Q(s_{t+1},a') - Q(s_t,a_t)\Big].$$

Expected SARSA eliminates the variance of the action draw at the cost of a sum over actions, and strictly dominates SARSA in per-sample efficiency. Observe the family resemblance: with π the greedy policy, Expected SARSA *is* Q-learning. All three updates are sampled Bellman backups differing only in the distribution used at the successor state—taken action (SARSA), policy average (Expected SARSA), maximum (Q-learning).

6.5 Maximisation Bias and Double Learning

The max in Q-learning's target has a statistical dark side. If $Q(s',a')$ are noisy estimates, then $\mathbb{E}[\max_{a'} Q(s',a')] \geq \max_{a'} \mathbb{E}[Q(s',a')]$ (Jensen, applied to the convex max function): the maximum of noisy estimates systematically overestimates the true maximum. Q-learning therefore carries an upward *maximisation bias*, which can be severe when rewards are noisy or actions numerous, and which propagates through bootstrapping.

Definition: Double Q-learning

Maintain two independent tables Q_1, Q_2. On each step flip a coin; if updating Q_1:

$$Q_1(s,a) \leftarrow Q_1(s,a) + \alpha\Big[r + \gamma\, Q_2\big(s', \arg\max_{a'} Q_1(s',a')\big) - Q_1(s,a)\Big],$$

and symmetrically for Q_2. *Selection* of the maximising action and *evaluation* of its value use different, independently noisy tables, decoupling the two roles and removing the systematic

overestimate.

This idea returns at scale as Double DQN (Chapter 11), where it repairs the same bias in deep networks with a one-line change.

6.6 Exploration: The Bandit Toolbox

Everything above presumed "sufficient exploration." The clean laboratory for exploration itself is the *multi-armed bandit*: one state, k actions with unknown reward distributions, and the single question of where to allocate trials. Its main strategies transfer directly to full MDPs.

- **ε-greedy.** Exploit with probability $1-\varepsilon$, sample uniformly otherwise. Crude—exploration is undirected and never stops distinguishing between wildly and marginally inferior arms—but robust, and the default in practice, often with ε annealed over time.
- **Optimistic initialisation.** Start Q high; every arm disappoints until tried, so greed itself explores. Simple, effective early, but a transient device only.
- **Upper confidence bound (UCB).** Choose $a_t = \arg\max_a \big[\, \widehat{Q}(a) + c\sqrt{\ln t / N(a)}\,\big]$: face uncertainty optimistically, in proportion to a confidence radius. UCB achieves regret $O(\ln t)$, matching the Lai–Robbins lower bound rate—the theoretical gold standard.
- **Softmax/Boltzmann.** Sample $a \propto \exp(\widehat{Q}(a)/T)$ with temperature T: exploration graded by estimated quality rather than uniform.
- **Thompson sampling.** Maintain a posterior over arm values; sample one draw from the posterior and act greedily on the draw. Bayesian, elegant, and empirically excellent; probability of exploration decays automatically as posteriors sharpen.

Worked Example: UCB in four pulls

Two arms; true means $\mu_1 = 0.6$, $\mu_2 = 0.4$; $c = 1$. Pull each arm once to initialise: suppose arm 1 pays 0 (unlucky), arm 2 pays 1 (lucky), so $\widehat{Q} = (0, 1)$, $N = (1, 1)$. Greedy would now play arm 2 forever—the classic trap. UCB at $t = 3$ computes indices $0 + \sqrt{\ln 3/1} = 1.048$ versus $1 + \sqrt{\ln 3/1} = 2.048$: arm 2 is chosen, suppose it pays 0; now $\widehat{Q} = (0, 0.5)$, $N = (1, 2)$. At $t = 4$: arm 1's index $0 + \sqrt{\ln 4/1} = 1.177$ beats arm 2's $0.5 + \sqrt{\ln 4/2} = 1.333$? Not yet—but arm 1's bonus now grows every step it is neglected ($\sqrt{\ln t}$ rising, N_1 frozen), and within a few more pulls its index overtakes, the arm is re-tried, and the unlucky first sample is washed out. The mechanism to internalise: *neglect itself raises an arm's priority*, so no arm can be starved forever on the strength of early noise—exactly what greedy lacks and exactly where its linear regret comes from.

The Abstract Viewpoint: Regret

The honest currency of exploration is **regret**: after T steps, $\mathrm{Regret}(T) = T\mu^* - \mathbb{E}\sum_{t\le T} r_t$, the shortfall against always playing the best arm. Uniform-forever ε-greedy incurs linear regret; UCB and Thompson sampling achieve logarithmic. In full MDPs the analogous quantities are studied under "optimism in the face of uncertainty" (UCRL, UCBVI) and posterior sampling (PSRL), with regret scaling as $\widetilde{O}(\sqrt{T})$ in tabular settings. Deep RL, lacking usable posteriors, resorts to intrinsic rewards—curiosity, pseudo-counts, prediction

error—as engineered surrogates for the same principle: reward yourself for reducing your own uncertainty.

Caution

In finance-flavoured applications, exploration is not free: an exploratory action is an order in a live market. The cliff-walk lesson applies verbatim—if exploratory mistakes are costly, the on-policy value (what SARSA optimises) is the honest objective, and exploration budgets deserve explicit accounting rather than a default $\varepsilon = 0.1$.

Key Takeaways

SARSA bootstraps from the action taken (on-policy); Q-learning bootstraps from the best action (off-policy) and converges to Q^* under mild conditions; Expected SARSA interpolates and unifies. The max operator introduces overestimation, cured by double learning. Exploration has a toolbox—ε-greedy, optimism, UCB, softmax, Thompson—whose common logic is to privilege uncertainty, and whose common scorecard is regret. Tabular control is now solved in principle; what remains is the world's refusal to be tabular.

Chapter 7

On-Policy and Off-Policy Learning

7.1 The Core Distinction

Every learning algorithm involves two policies, whether or not it admits it: the **behaviour policy** b that generates the data, and the **target policy** π whose value is being learned. When they coincide, learning is *on-policy*; when they differ, *off-policy*.

The distinction sounds bureaucratic and is anything but. On-policy methods (SARSA, REINFORCE, A2C, PPO in its inner loop) enjoy simple, faithful estimators but must discard data whenever the policy changes—yesterday's trajectories describe yesterday's policy. Off-policy methods (Q-learning, DQN, DDPG, SAC) may reuse old data, learn from demonstrations, logs, or other agents, and separate exploration from the object of learning—but their estimators must *correct* for the mismatch between b and π, and the corrections are where all the difficulty lives.

ELI5: Learning to cook

On-policy: you learn to cook by cooking, tasting your own dishes, and adjusting. Off-policy: you also learn from watching your grandmother, from old family recipes, from cooking shows. The second learner has vastly more material—but must constantly translate: "she has a tandoor and I do not; how much of what worked for her transfers to me?" That translation factor is importance sampling, and mistranslation is where off-policy learning fails.

7.2 Importance Sampling in Full

Let $p_\pi(\tau)$ and $p_b(\tau)$ be the trajectory distributions under target and behaviour. Both factorise into policy terms and dynamics terms, and the dynamics cancel in the ratio:

$$\frac{p_\pi(\tau)}{p_b(\tau)} = \frac{\mu_0(s_0)\prod_t \pi(a_t \mid s_t)P(s_{t+1} \mid s_t, a_t)}{\mu_0(s_0)\prod_t b(a_t \mid s_t)P(s_{t+1} \mid s_t, a_t)} = \prod_t \frac{\pi(a_t \mid s_t)}{b(a_t \mid s_t)} =: \rho_{0:T-1}.$$

This cancellation is the small miracle that makes model-free off-policy learning possible at all: the correction requires only the two policies, never the unknown dynamics. For any trajectory functional—in particular the return—

$$\mathbb{E}_\pi[G] = \mathbb{E}_b\big[\rho_{0:T-1}\, G\big],$$

valid under the **coverage** condition $\pi(a \mid s) > 0 \Rightarrow b(a \mid s) > 0$.

7.2.1 Ordinary versus weighted estimators

Given returns $G^{(i)}$ with ratios $\rho^{(i)}$ from N episodes:

$$\widehat{V}_{\mathrm{OIS}} = \frac{1}{N}\sum_i \rho^{(i)} G^{(i)}, \qquad \widehat{V}_{\mathrm{WIS}} = \frac{\sum_i \rho^{(i)} G^{(i)}}{\sum_i \rho^{(i)}}.$$

OIS is unbiased but its variance involves $\mathbb{E}_b[\rho^2 G^2]$, which can be infinite even in trivial examples (a single state where π deterministically picks an action that b takes with small probability already produces ratios of size $1/b(a|s)$ raised to the episode length). WIS is biased at finite N (though consistent), bounded between the extremes of observed returns, and almost always preferable. The general lesson generalises far beyond this pair:

Caution

The variance of importance-sampling corrections grows *multiplicatively with horizon*. Products of per-step ratios $\prod_t \pi/b$ explode or vanish exponentially unless π and b are close at every step. This single fact explains: why pure off-policy MC is hopeless on long tasks; why per-decision and truncated corrections exist; why deep off-policy methods (DQN, SAC) avoid trajectory ratios entirely by bootstrapping; and why PPO clips the *one-step* ratio and refreshes data constantly. Horizon is the enemy; every practical method is an armistice with it.

7.2.2 Per-decision and truncated corrections

Not every reward needs the whole product: the reward at time k depends only on actions up to k, so $\mathbb{E}_\pi[\gamma^k r_{k+1}] = \mathbb{E}_b[\rho_{0:k}\,\gamma^k r_{k+1}]$—the *per-decision* estimator—which strictly reduces variance. Further down the same road lie truncated corrections ($\rho \mapsto \min(\rho, \bar\rho)$, trading a controlled bias for bounded variance), as used in the V-trace estimator of IMPALA, and the clipped surrogate of PPO. The design space is a bias–variance dial with horizon as the amplifier.

7.3 Off-Policy TD and the Tree Backup

Bootstrapping shortens the horizon of the correction. Off-policy TD(0) for state values needs only the single ratio at the current step:

$$V(s_t) \leftarrow V(s_t) + \alpha\,\rho_t\,\big[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\big], \qquad \rho_t = \frac{\pi(a_t \mid s_t)}{b(a_t \mid s_t)},$$

because the target bootstraps rather than unrolling. Q-learning needs *no* ratio at all: its target $\max_{a'} Q(s', a')$ does not depend on the behaviour's next action. Expected SARSA with target π likewise needs none. The *tree backup* algorithm extends this ratio-free property to n-step returns by branching over the target policy's probabilities at every intermediate step, at the price of shrinking effective trace length whenever π places small mass on the actions actually taken. The synthesis, Q(σ), interpolates per-step between sampling (SARSA-like) and expectation (tree-backup-like) and contains the whole family as special cases.

The Abstract Viewpoint: Change of measure

Importance sampling is the Radon–Nikodym derivative in action: $\mathbb{E}_\pi[f] = \mathbb{E}_b[\frac{\mathrm{d}p_\pi}{\mathrm{d}p_b} f]$, requiring absolute continuity $p_\pi \ll p_b$ (that is coverage). All the RL-specific phenomena are statements about how badly behaved this derivative is when the measures are products over long horizons: KL divergence between trajectory distributions grows linearly in horizon, hence the derivative's variance grows exponentially. Trust-region methods (Chapter 12) manage precisely this object: they bound the per-state KL so the trajectory-level change of measure stays tame. Seen this way, TRPO is measure theory with a line search.

7.4 A Taxonomy Worth Memorising

Algorithm	On/Off-policy	Correction used	Where treated
SARSA	On	None needed	Ch. 6
Expected SARSA	Either	None (expectation over π)	Ch. 6
Q-learning / DQN	Off	None (max target)	Ch. 6, 11
MC + IS	Off	Full-trajectory ratio	Ch. 4, 7
Off-policy TD(0)	Off	One-step ratio	Ch. 7
Tree backup	Off	Target-policy expectations	Ch. 7
REINFORCE, A2C	On	None needed	Ch. 9, 10
IMPALA (V-trace)	Slightly off	Truncated ratios	Ch. 15
PPO	Near-on	Clipped one-step ratio	Ch. 12
DDPG, TD3, SAC	Off	None (bootstrapped critics)	Ch. 13
Offline RL (CQL, BCQ)	Extremely off	Pessimism/constraints	Ch. 16

The last row deserves a forward pointer: *offline* RL is off-policy learning pushed to the limit where the behaviour policy is fixed, finite, and gone—no further interaction allowed. There the coverage condition fails not occasionally but structurally, corrections cannot save you, and a new principle (pessimism toward the unknown) takes over. Chapter 16 is devoted to it; the vocabulary built here is its foundation.

Key Takeaways

Behaviour generates data; the target is what we wish to learn about. The likelihood ratio $\prod_t \pi/b$ corrects the mismatch exactly—the dynamics cancel—but with variance that compounds exponentially in horizon. The craft of off-policy RL is horizon management: weighted and per-decision estimators, truncation, bootstrapping (which needs only one-step or zero corrections), and staying near-on-policy by design. Every modern algorithm's stance toward this trade-off is one of its defining features.

Part III

Scaling Up

Chapter 8

Function Approximation

8.1 Beyond the Table

Tabular methods store one number per state (or pair). The moment states are images, sensor vectors, order books, or token sequences, tables are impossible and—worse—pointless: the agent will never see the same state twice, so memorising values teaches nothing about the states to come. The remedy is **function approximation**: represent the value function by a parametric family

$$\widehat{v}(s;\mathbf{w}) \approx V^{\pi}(s), \qquad \mathbf{w} \in \mathbb{R}^{d}, \quad d \ll |\mathcal{S}|,$$

so that updating $\mathbf{w}$ at one state *generalises* to similar states. Linear architectures write $\widehat{v}(s;\mathbf{w}) = \mathbf{w}^{\top}\boldsymbol{\phi}(s)$ over a feature map $\boldsymbol{\phi}$ (tile codings, radial bases, polynomial and Fourier features); non-linear architectures use neural networks and carry us into Part IV.

8.2 Gradient Descent on Value Error

Define the mean-squared value error weighted by the on-policy state distribution μ:

$$\overline{\mathrm{VE}}(\mathbf{w}) \;=\; \sum_{s} \mu(s)\,\big[V^{\pi}(s) - \widehat{v}(s;\mathbf{w})\big]^{2}.$$

If the true targets $V^{\pi}(s)$ were available, stochastic gradient descent on samples $s \sim \mu$ would be routine. They are not, so we substitute learning targets. Monte Carlo substitutes G_t:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\,\big[G_t - \widehat{v}(s_t;\mathbf{w})\big]\,\nabla_{\mathbf{w}}\widehat{v}(s_t;\mathbf{w}),$$

a *true* SGD on $\overline{\mathrm{VE}}$ since $\mathbb{E}[G_t \mid s_t] = V^{\pi}(s_t)$; it converges to a local optimum (a global one in the linear case). TD substitutes $r_{t+1} + \gamma\widehat{v}(s_{t+1};\mathbf{w})$:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha\,\big[r_{t+1} + \gamma\,\widehat{v}(s_{t+1};\mathbf{w}) - \widehat{v}(s_t;\mathbf{w})\big]\,\nabla_{\mathbf{w}}\widehat{v}(s_t;\mathbf{w}).$$

This is called *semi-gradient* TD: the target contains $\mathbf{w}$ yet is treated as a constant when differentiating. It is not the gradient of any fixed objective—a small dishonesty with large consequences.

Theorem: Linear semi-gradient TD(0): convergence and error bound

With linear approximation, on-policy sampling ($s \sim \mu$, the stationary distribution of π), and Robbins–Monro steps, semi-gradient TD(0) converges to the unique **TD fixed point** $\mathbf{w}_{\mathrm{TD}}$ solving $A\mathbf{w} = b$ with $A = \mathbb{E}[\boldsymbol{\phi}_t(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1})^{\top}]$, $b = \mathbb{E}[r_{t+1}\boldsymbol{\phi}_t]$, and this point satisfies

$$\overline{\mathrm{VE}}(\mathbf{w}_{\mathrm{TD}}) \;\le\; \frac{1}{1-\gamma}\,\min_{\mathbf{w}} \overline{\mathrm{VE}}(\mathbf{w}).$$

The key to the proof is that on-policy sampling makes the matrix A positive definite; the expected update is then a stable linear system.

The bound degrades as $\gamma \to 1$ but certifies the essential point: *on-policy* linear TD lands provably close to the best the feature class allows. The qualifier is about to earn its keep.

The Abstract Viewpoint: Projected Bellman equations

Geometrically, the space of representable functions $\{\Phi\mathbf{w}\}$ is a plane inside $\mathbb{R}^{|\mathcal{S}|}$; the Bellman operator maps points off the plane; TD composes it with the μ-weighted orthogonal projection Π back on. The TD fixed point solves the *projected Bellman equation* $\Phi\mathbf{w} = \Pi\mathcal{T}^\pi\Phi\mathbf{w}$. On-policy, $\Pi\mathcal{T}^\pi$ is a contraction in the μ-weighted norm and all is well. Off-policy, the projection is weighted by the *wrong* distribution, $\Pi\mathcal{T}^\pi$ can be expansive, and iterates may spiral outward. One diagram—plane, operator arrow, projection arrow—explains every divergence example in the literature.

8.3 The Deadly Triad

Caution: The deadly triad

Instability and divergence threaten whenever three ingredients combine:

1. **Function approximation** (generalising updates),
2. **Bootstrapping** (targets containing current estimates),
3. **Off-policy training** (updating under a distribution other than the target policy's).

Any two are safe. All three together can make parameters diverge to infinity even in tiny, fully linear examples.

Baird's counterexample (seven states, linear features, exact expected updates) exhibits divergence with no sampling noise at all; Tsitsiklis and Van Roy supplied a two-state version. The mechanism is the wrong-weighted projection above: off-policy updating over-weights states the target policy rarely visits, and bootstrapping amplifies their erroneous values into targets elsewhere.

Responses fall into three schools. *Gradient-TD* methods (GTD2, TDC) perform true SGD on the projected Bellman error, restoring convergence under off-policy sampling at the cost of a second learned parameter vector. *Emphatic TD* reweights updates to repair the distribution mismatch. Deep practice mostly takes a third road—engineering the triad into submission with replay buffers that stay near-on-policy, target networks that slow the bootstrap (Chapter 11), and constrained updates (Chapter 12)—accepting theory-free stability bought by empirically robust design.

Worked Example: A TD fixed point, computed

Two states in a loop: $1 \to 2 \to 1 \to \cdots$, deterministic, rewards $r(1{\to}2) = 0$, $r(2{\to}1) = 1$, $\gamma = 0.9$. True values solve $V(1) = 0.9\,V(2)$, $V(2) = 1 + 0.9\,V(1)$: $V(1) = 4.263$, $V(2) = 5.737$. Now approximate with a *single* feature, $\phi(1) = 1$, $\phi(2) = 2$, so $\widehat{v} = w\phi$. The stationary distribution is uniform. The TD fixed point solves $Aw = b$ with $A = \frac{1}{2}[\phi(1)(\phi(1) - \gamma\phi(2))] + \frac{1}{2}[\phi(2)(\phi(2) - \gamma\phi(1))] = \frac{1}{2}(1)(1 - 1.8) + \frac{1}{2}(2)(2 - 0.9) = -0.4 + 1.1 = 0.7$ and $b = \frac{1}{2}(0)(1) + \frac{1}{2}(1)(2) = 1$, giving $w_{\text{TD}} = 1.429$: estimates $(1.43, 2.86)$. The direct $\overline{\text{VE}}$

minimiser is $w^* = \frac{\sum \mu\phi V}{\sum \mu\phi^2} = \frac{0.5(4.263)+0.5(2)(5.737)}{0.5(1)+0.5(4)} = 2.147$: estimates $(2.15, 4.29)$. TD lands at a *different* point than the best fit—biased by bootstrapping through a crude feature—yet within the theorem's $1/(1-\gamma)$ factor. With one number per two states, neither answer is good; the exercise is to see, concretely, that "convergent" and "converged to what you wanted" are separate claims under function approximation.

8.4 Least-Squares and Batch Methods

When features are few and data plentiful, the TD fixed point can be computed directly from samples: estimate $\widehat{A} = \frac{1}{N}\sum_t \boldsymbol{\phi}_t(\boldsymbol{\phi}_t - \gamma\boldsymbol{\phi}_{t+1})^\top$, $\widehat{b} = \frac{1}{N}\sum_t r_{t+1}\boldsymbol{\phi}_t$, and solve $\mathbf{w} = \widehat{A}^{-1}\widehat{b}$. This is **LSTD**, the least-squares temporal-difference method; its control counterpart **LSPI** (least-squares policy iteration) alternates LSTD-Q evaluation with greedy improvement. Sample-efficient and hyperparameter- free (no step size), at $O(d^2)$ per update or $O(d^3)$ per solve—the method of choice when d is modest, and the ancestor of the fitted-Q methods that reappear in offline RL (Chapter 16).

8.5 On Features

A word of craft. Linear methods live or die by their features: *tile coding* (overlapping coarse grids, giving piecewise-constant generalisation with $O(1)$ lookup), *Fourier bases* (global, smooth, excellent for low-dimensional continuous states), *RBF networks* (local bumps). The trade is locality against generalisation breadth, and the discipline they teach—think about which states *should* share value—remains valuable even now that deep networks learn the features themselves. Indeed a deep value network is usefully read as linear-in- learned-features: its last layer is $\mathbf{w}^\top\boldsymbol{\phi}_\theta(s)$ with $\boldsymbol{\phi}_\theta$ trained end-to-end, so every statement in this chapter about the linear top layer still applies, conditionally on the representation beneath it.

Key Takeaways

Function approximation replaces the table with a parametric family and turns value estimation into stochastic optimisation. MC targets give true gradients; TD gives semi-gradients that converge on-policy (linear case) to within $1/(1-\gamma)$ of the best representable fit, via the projected Bellman equation. Combining approximation, bootstrapping, and off-policy data—the deadly triad—can diverge; cures are gradient-TD, emphasis, or the engineering stabilisers of deep RL. LSTD/LSPI solve the linear fixed point directly from batches. With values now parametric, the natural next step is to parameterise the *policy*—the subject of the next chapter.

Chapter 9

Policy Gradient Methods

9.1 Optimising the Policy Directly

Value-based methods learn Q and act greedily; the policy is implicit, and a small change in Q can flip an $\arg\max$ discontinuously. Policy-based methods invert the arrangement: parameterise the policy itself, $\pi_\theta(a \mid s)$, and ascend the performance

$$J(\theta) \;=\; \mathbb{E}_{\pi_\theta}\Big[\textstyle\sum_{t\ge 0}\gamma^t r_{t+1}\Big]$$

by gradient methods. The gains are structural: continuous action spaces are handled natively (output the parameters of a Gaussian); stochastic policies—sometimes genuinely optimal under partial observability, and always useful for exploration—are first-class citizens; and the policy changes *smoothly* in θ, which makes convergence analysis and trust regions possible. The price is variance, and much of Chapters 9–12 is the story of paying it down.

For discrete actions the standard parameterisation is softmax over preferences, $\pi_\theta(a \mid s) \propto \exp h_\theta(s,a)$; for continuous ones, a Gaussian $\pi_\theta(\cdot \mid s) = \mathcal{N}(\mu_\theta(s), \Sigma_\theta(s))$.

9.2 The Policy Gradient Theorem

The obstacle to differentiating J is that θ moves the *distribution* of trajectories, including the state distribution d^{π_θ}—and the environment's contribution to that distribution is unknown. The policy gradient theorem (Sutton, McAllester, Singh, Mansour, 2000) shows that, miraculously, the gradient never needs the derivative of the state distribution.

Theorem: Policy gradient theorem

For the discounted objective,

$$\nabla_\theta J(\theta) \;=\; \mathbb{E}_{s\sim d^{\pi_\theta},\, a\sim\pi_\theta(\cdot|s)}\big[\,\nabla_\theta \log \pi_\theta(a \mid s)\; Q^{\pi}[\pi_\theta](s,a)\,\big],$$

where d^{π_θ} is the (discounted) state visitation distribution under π_θ.

Proof (likelihood-ratio route). Write $J(\theta) = \int p_\theta(\tau) G(\tau)\,\mathrm{d}\tau$ over trajectories. Then

$$\nabla_\theta J = \int p_\theta(\tau)\,\nabla_\theta \log p_\theta(\tau)\, G(\tau)\,\mathrm{d}\tau = \mathbb{E}_\tau\big[\nabla_\theta \log p_\theta(\tau)\, G(\tau)\big],$$

using $\nabla p = p \nabla \log p$. The trajectory likelihood factorises as $p_\theta(\tau) = \mu_0(s_0)\prod_t \pi_\theta(a_t \mid s_t) P(s_{t+1} \mid s_t, a_t)$; the dynamics terms do not involve θ and vanish under $\nabla_\theta \log$, leaving $\nabla_\theta \log p_\theta(\tau) = \sum_t \nabla_\theta \log \pi_\theta(a_t \mid s_t)$. Finally, causality: the action at time t cannot influence rewards already received, so pairing $\nabla \log \pi_\theta(a_t \mid s_t)$ with the *reward-to-go* G_t rather than the whole $G(\tau)$ leaves the expectation unchanged (the dropped cross-terms have zero mean). Conditioning on (s_t, a_t) replaces G_t by $Q^{\pi_\theta}(s_t, a_t)$ and re-indexing the sum over time as an expectation over d^{π_θ} gives the stated form. □

ELI5: Turn up what worked

Each action you took gets a knob: pushing the knob up makes that action more likely next time you are in that situation. The theorem says: after the episode, push each knob in proportion to how well things went after you used it. Actions followed by good outcomes get louder; actions followed by bad ones get quieter. Repeat forever. Everything else in policy-gradient research is about measuring "how well things went" with less noise.

The same score-function identity yields the useful lemma $\mathbb{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a \mid s)] = 0$ (differentiate $\sum_a \pi_\theta(a \mid s) = 1$), of which two consequences follow immediately: any *action-independent* function $b(s)$ can be subtracted from Q without changing the gradient's expectation, and the resulting freedom is exactly the baseline/control- variate device below.

9.3 REINFORCE and Baselines

Estimating the theorem's expectation with sampled returns gives the oldest policy-gradient algorithm.

Algorithm: REINFORCE (Williams, 1992)

Loop: generate an episode with π_θ; for each t, compute the return G_t; update

$$\theta \;\leftarrow\; \theta + \alpha\, \gamma^t\, G_t\, \nabla_\theta \log \pi_\theta(a_t \mid s_t).$$

Unbiased, and in practice noisy to the point of pain: G_t carries the variance of the whole remaining trajectory. Subtracting a **baseline**,

$$\theta \leftarrow \theta + \alpha \big(G_t - b(s_t)\big) \nabla_\theta \log \pi_\theta(a_t \mid s_t),$$

preserves unbiasedness (by the lemma) and, with the natural choice $b(s) = \widehat{V}(s)$, can slash variance: the weight becomes an *advantage estimate* $G_t - \widehat{V}(s_t)$, measuring how much better the action did than expected rather than how good the world is in absolute terms. A learned state-value baseline is the embryo of the critic; feed it a bootstrapped target instead of G_t and you have arrived at actor–critic, the subject of the next chapter.

Worked Example: Why baselines matter, numerically

Two actions in one state; true action values 101 and 99. Without a baseline, the gradient weights are $\sim$100 either way: the update says "do more of both," and only the tiny difference of the two large numbers—easily buried by sampling noise—distinguishes them. With baseline $b = 100$, the weights become $+1$ and -1: signal with no pedestal. Nothing about the expectation changed; the estimator's variance collapsed. Centring is free money, which is why every serious implementation, up to and including RLHF pipelines, normalises or baselines its reward signal.

9.4 Natural Policy Gradients

Vanilla gradient ascent is coordinate-dependent: reparameterise the policy and the "steepest" direction changes, though the family of distributions is the same. The remedy, due to Amari in

general and Kakade in RL, is to measure step length not in parameter space but in *distribution* space, using the Fisher information metric

$$F(\theta) \;=\; \mathbb{E}_{s,a}\big[\nabla_\theta \log \pi_\theta(a \mid s)\, \nabla_\theta \log \pi_\theta(a \mid s)^\top\big],$$

which is the second-order expansion of the KL divergence: $D_{\mathrm{KL}}(\pi_\theta \,\|\, \pi_{\theta+\mathrm{d}\theta}) \approx \frac{1}{2}\, \mathrm{d}\theta^\top F(\theta)\, \mathrm{d}\theta$. The **natural gradient** direction is

$$\widetilde{\nabla}_\theta J \;=\; F(\theta)^{-1} \nabla_\theta J,$$

the steepest ascent per unit of KL movement. It is invariant to smooth reparameterisation, automatically rescales poorly conditioned parameters, and—the deep reason for its later importance—bounds how much the *behaviour* changes per step, foreshadowing trust regions. TRPO (Chapter 12) is the natural gradient made practical at neural-network scale.

The Abstract Viewpoint: Information geometry

The set $\{\pi_\theta\}$ is a manifold of probability distributions; the Fisher matrix is its Riemannian metric; the natural gradient is the ordinary gradient with the metric's index raised. From this height, the distinction between "parameters" and "policy" dissolves: optimisation happens on the statistical manifold, and θ is merely a chart. Mirror descent with KL as the Bregman divergence gives the same updates from the optimisation side, and the modern RLHF objective with its KL penalty (Chapter 17) is the same geometry wearing an alignment costume: in every case one asks the policy to improve while moving a bounded information-distance from a reference.

9.5 Strengths, Weaknesses, Scope

Policy gradients handle continuous and high-dimensional action spaces gracefully, encode stochastic behaviour natively, and admit smooth local analysis (indeed, recent theory establishes global convergence for softmax policies on tabular MDPs, the objective's non-convexity notwithstanding). Against this: estimates are high-variance; learning is on-policy, so data is expensive and stale quickly; convergence is local in general; and step sizes are treacherous—too large a step changes the state distribution, which invalidates the very samples that suggested the step. Baselines treat the variance; natural gradients and trust regions treat the step size; actor–critic treats the sample cost. The next three chapters administer these treatments in order.

Key Takeaways

The policy gradient theorem converts return maximisation into a sampled expectation of $\nabla \log \pi \times$ (a value weight), with no derivative of the unknown dynamics required. REINFORCE implements it with Monte Carlo returns; baselines centre the weight into an advantage and cut variance at zero cost in bias; natural gradients replace Euclidean steps with KL-metric steps, making updates parameterisation-invariant and behaviour-bounded. Variance, data hunger, and step-size fragility are the standing problems—and the design brief for actor–critic, TRPO, and PPO.

Chapter 10

Actor–Critic Methods

10.1 Marrying the Two Traditions

Value-based methods have low variance and no policy; policy-based methods have a policy and crippling variance. Actor–critic architectures take the obvious step of keeping both learners and letting each fix the other's defect:

- the **actor** $\pi_\theta(a \mid s)$ selects actions and is updated along the policy gradient;
- the **critic** $V_w(s)$ (or Q_w) evaluates the actor's choices and supplies the gradient's weight—replacing the noisy Monte Carlo return with a learned, bootstrapped estimate.

The lineage is old—Barto, Sutton and Anderson's 1983 pole-balancing system already had exactly this two-network shape—and the design has outlasted every fashion since: A3C, PPO, SAC, and the RLHF systems that fine-tune language models are all actor–critics.

10.2 The TD Error as Advantage Estimate

The policy gradient wants $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$ as its weight. A critic that only models V suffices, by a small but lovely identity:

$$\mathbb{E}\big[\delta_t \,\big|\, s_t, a_t\big] = \mathbb{E}\big[r_{t+1} + \gamma V^\pi(s_{t+1}) \mid s_t, a_t\big] - V^\pi(s_t) = Q^\pi(s_t,a_t) - V^\pi(s_t) = A^\pi(s_t,a_t).$$

The TD error is an unbiased single-sample estimate of the advantage. The minimal actor–critic is therefore:

Algorithm: One-step actor–critic

At each step, act $a_t \sim \pi_\theta(\cdot \mid s_t)$, observe (r_{t+1}, s_{t+1}), compute $\delta_t = r_{t+1} + \gamma V_w(s_{t+1}) - V_w(s_t)$, and update

$$w \leftarrow w + \alpha_w\, \delta_t\, \nabla_w V_w(s_t), \qquad \theta \leftarrow \theta + \alpha_\theta\, \delta_t\, \nabla_\theta \log \pi_\theta(a_t \mid s_t).$$

One number, δ_t, drives both learners: as a regression residual it improves the critic; as an advantage estimate it steers the actor. When $\delta_t > 0$ the action outperformed expectation and is reinforced; when $\delta_t < 0$ it is suppressed.

ELI5: The coach and the player

The player (actor) decides what to do on the pitch. The coach (critic) does not play, but keeps an honest running estimate of how the match should be going, and after every move calls out "better than I expected!" or "worse!". The player adjusts habits in the direction of the coach's surprise, and the coach refines expectations from the same surprises. Neither is much use alone; together they improve each other every few seconds instead of waiting for the final whistle.

10.3 Generalised Advantage Estimation

The one-step TD error is low-variance but inherits the critic's bias; the Monte Carlo advantage $G_t - V_w(s_t)$ is unbiased but noisy. Exactly as in Chapter 5, a λ-dial interpolates. Define the **generalised advantage estimator** (Schulman et al., 2016):

$$\widehat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} \;=\; \sum_{l=0}^{\infty} (\gamma\lambda)^l \, \delta_{t+l},$$

an exponentially weighted sum of future TD errors. At $\lambda = 0$ it is δ_t; at $\lambda = 1$ it telescopes to $G_t - V_w(s_t)$. The identity behind it is the advantage analogue of the λ-return: each n-step advantage estimate equals the partial sum $\sum_{l<n} (\gamma)^l \delta_{t+l}$ (plus terms handled by the weighting), and GAE geometrically averages them. In practice $\lambda \in [0.9, 0.98]$ is standard, and GAE is the advantage estimator inside PPO and virtually every RLHF implementation—the single most consequential piece of "plumbing" in modern policy optimisation.

10.4 A3C and A2C

Scaling actor–critic to deep networks met the same instability that plagued deep Q-learning: consecutive samples are correlated, and correlated gradients thrash the network. DQN's answer was a replay buffer; the asynchronous answer (Mnih et al., 2016) was parallelism. **A3C** runs many actor–learner threads, each with its own copy of the environment, all applying gradients asynchronously to shared parameters: at any instant the workers occupy different corners of the state space, so the aggregate gradient stream is decorrelated *without* replay—important because policy gradients are on-policy and cannot straightforwardly reuse stale data. Each worker uses n-step returns and adds an **entropy bonus** $\beta\, \mathcal{H}(\pi_\theta(\cdot \mid s))$ to the actor loss, discouraging premature collapse onto a deterministic policy—a cheap, effective exploration device that has become standard equipment.

A2C, the synchronous variant, waits for all workers and applies one batched update; it is equally effective, more reproducible, and better matched to GPU hardware, so it is what most "A3C" code actually runs. Both are best understood not as new theory but as systems engineering that made on-policy actor–critic competitive at scale—and as the direct ancestors of the distributed rollout–learner architectures (IMPALA and its descendants) that train today's largest RL systems, including RLHF clusters.

10.5 Compatible Features and Bias

A critic introduces bias whenever it is imperfect; can the actor's gradient nonetheless be exact? The *compatible function approximation* theorem answers: if the critic is linear in the features $\nabla_\theta \log \pi_\theta(a \mid s)$ and trained to the least-squares optimum, then substituting it for the true Q leaves the policy gradient exactly unbiased. The condition is more instructive than practical—it says the critic need only be accurate *in the directions the actor can actually move*—and it explains the empirical robustness of actor–critic: moderate critic error largely projects away.

Caution: Two time scales

The convergence theory of actor–critic requires the critic to learn *faster* than the actor: formally, step sizes with $\alpha_w \gg \alpha_\theta$ so the critic tracks a quasi-static policy (two-time-scale

stochastic approximation). Violate this and the actor climbs a landscape the critic has not yet mapped; oscillation and collapse follow. In deep practice the same principle appears as multiple critic updates per actor update, warm-up phases for the value network, and careful loss weighting between the two heads.

The Abstract Viewpoint: GPI once more

Actor–critic is generalised policy iteration with both halves made incremental and differentiable: the critic performs one flake of policy evaluation per step ($V_w \to V^{\pi_\theta}$), the actor one flake of policy improvement (π_θ ascends against V_w). Where classical policy iteration alternated *completed* phases, actor–critic runs both flows simultaneously and trusts two-time-scale separation to keep them consistent. Every algorithm in Part IV will be readable as a choice of: critic target (which Bellman operator, how truncated), actor objective (which surrogate, how constrained), and coupling discipline (time scales, target networks, clipping).

Key Takeaways

Actor–critic keeps a parametric policy and a parametric value function and lets each teach the other: the TD error is simultaneously the critic's regression residual and an unbiased estimate of the advantage that weights the actor's gradient. GAE dials bias against variance by geometrically averaging TD errors and is the standard modern estimator. A3C/A2C decorrelate on-policy data through parallel environments and add entropy bonuses for exploration. Two-time-scale discipline—critic faster than actor—is the stability condition. The architecture is the chassis on which Chapters 11–13 and 17 all build.

Part IV

Deep Reinforcement Learning

Chapter 11

Deep Value-Based Methods

11.1 From Q-Learning to Deep Q-Learning

Replace the Q-table with a neural network $Q_\theta(s,a)$ and the Q-learning update becomes a regression step toward the sampled Bellman target:

$$\mathcal{L}(\theta) \;=\; \mathbb{E}\Big[\underbrace{\big(r+\gamma\max_{a'}Q_\theta(s',a')}_{\text{target}} - Q_\theta(s,a)\big)^2\Big].$$

Naively trained, this diverges more often than it learns, and by Chapter 8 we know why: it is the deadly triad in its purest form— nonlinear function approximation, bootstrapping, off-policy data— aggravated by two further pathologies of online deep learning: consecutive samples are strongly correlated (violating the i.i.d. premise of SGD), and the regression *target moves with the parameters being trained*, so the network chases its own tail.

The Deep Q-Network (DQN; Mnih et al., 2015)—the system that learned 49 Atari games from pixels with one architecture and one hyperparameter set—introduced two stabilisers that remain standard equipment across deep RL.

Definition: The two DQN stabilisers

Experience replay. Store transitions (s,a,r,s') in a large circular buffer $\mathcal{D}$; train on uniformly sampled minibatches. Sampling across the buffer breaks temporal correlation, reuses each experience many times (sample efficiency), and smooths the training distribution.
Target network. Keep a slowly updated copy θ^- of the parameters and compute targets with it:

$$y = r+\gamma\max_{a'}Q_{\theta^-}(s',a'), \qquad \mathcal{L}(\theta)=\mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\big[(y-Q_\theta(s,a))^2\big],$$

copying $\theta^-\leftarrow\theta$ every C steps (or Polyak-averaging $\theta^-\leftarrow\tau\theta+(1-\tau)\theta^-$). Freezing the target between syncs turns a fixed-point chase into a sequence of well-posed regression problems.

ELI5: Stop grading your own moving homework

Imagine revising an essay while the marking rubric is rewritten every time you edit a sentence—you would never converge on anything. DQN's target network fixes the rubric for a while: study against a frozen copy of yourself, improve, *then* refresh the rubric. And instead of studying only tonight's notes (the last few correlated moments), replay shuffles flashcards from your whole past. Two mundane study habits; they made deep RL work.

Architecture matters too: DQN outputs *all* action values from one forward pass, $Q_\theta(s,\cdot)\in\mathbb{R}^{|\mathcal{A}|}$, making the max cheap; inputs stack recent frames to restore approximate Markovness; rewards are clipped and, in later practice, Huber loss replaces squared loss for robustness to outlier targets.

Worked Example: One DQN minibatch element, all three targets

A replayed transition: $(s, a{=}2, r{=}0.5, s')$, $\gamma = 0.99$, and suppose the networks output, over three actions,

$$Q_\theta(s', \cdot) = (1.2,\ 2.1,\ 1.9), \qquad Q_{\theta^-}(s', \cdot) = (1.4,\ 1.6,\ 2.0).$$

Vanilla (no target net): $y = 0.5 + 0.99 \max(1.2, 2.1, 1.9) = 0.5 + 0.99(2.1) = 2.579$—built from the very network being updated. *DQN*: $y = 0.5 + 0.99 \max(1.4, 1.6, 2.0) = 2.480$, stable while θ^- is frozen. *Double DQN*: the online net selects $\arg\max = 2$ (value 2.1), the target net evaluates *that* action: $y = 0.5 + 0.99 \times Q_{\theta^-}(s', 2) = 0.5 + 0.99(1.6) = 2.084$—notably lower, because the online net's favourite is not the target net's, which is precisely the signature of selection noise that Double DQN declines to pay for. The loss for this element is then $(y - Q_\theta(s, 2))^2$ (or its Huber variant), and only $Q_\theta(s, 2)$—the action actually taken—receives gradient.

11.2 Double DQN

Chapter 6's maximisation bias afflicts DQN with interest: a deep network generalises its noise, so overestimates in one state contaminate many. Double DQN (van Hasselt et al., 2016) ports double Q-learning at the cost of zero extra parameters, using the online network to *select* and the target network to *evaluate*:

$$y^{\text{DDQN}} \;=\; r + \gamma\, Q_{\theta^-}\big(s',\, \arg\max_{a'} Q_\theta(s', a')\big).$$

Measured value estimates drop from wildly optimistic to nearly calibrated, and play improves on the majority of Atari games—one line of code, disproportionate return.

11.3 Dueling Networks

In many states, all actions lead to similar futures; what matters is *being there*, not the fine choice. The dueling architecture (Wang et al., 2016) builds this prior in, decomposing the head into a state value and advantages:

$$Q_\theta(s, a) \;=\; V_\eta(s) \;+\; A_\psi(s, a) \;-\; \frac{1}{|\mathcal{A}|} \sum_{a'} A_\psi(s, a'),$$

the mean-subtraction pinning down the otherwise unidentifiable split ($V + c$, $A - c$ give the same Q). The value stream trains on *every* sample regardless of which action was taken, so state values are learned faster and shared across actions—especially potent when actions are numerous or frequently irrelevant.

11.4 Prioritised Experience Replay

Uniform replay squanders capacity on transitions the network already fits. Prioritised replay (Schaul et al., 2016) samples transition i with probability

$$P(i) \;=\; \frac{p_i^\alpha}{\sum_k p_k^\alpha}, \qquad p_i = |\delta_i| + \epsilon,$$

favouring large TD errors—the surprising transitions. Non-uniform sampling biases the gradient's expectation; importance weights $w_i = (N\,P(i))^{-\beta}$, annealed toward $\beta = 1$, correct it. (Chapter 7's machinery, reappearing inside the buffer.) Learning speed roughly doubles on the Atari suite.

11.5 The Rainbow Synthesis and Beyond

Further refinements each attack a distinct weakness: *multi-step targets* (n-step returns, faster reward propagation); *distributional RL* (C51/QR-DQN: learn the full return distribution, not just its mean—richer gradients and, incidentally, exactly the object a risk-aware trader wants); *noisy networks* (learned parametric exploration replacing ε-greedy). **Rainbow** (Hessel et al., 2018) stacked six of these on DQN and found them near-additive, an ablation study that doubled as a map of which fixes address which failure. The value-based line continues through R2D2 (recurrent replay), Agent57, and MuZero's learned-model variant (Chapter 14); its stabilisers—replay, targets, decoupled selection/evaluation—are now vocabulary items used far outside value-based RL, including in the RLHF systems of Chapter 17.

Component	Failure addressed	Mechanism
Replay buffer	Correlated samples	i.i.d.-ish minibatches
Target network	Moving targets	Frozen bootstrap
Double DQN	Max-bias	Decouple select/evaluate
Dueling head	Redundant per-action learning	$V + A$ decomposition
Prioritised replay	Wasted samples	TD-error sampling + IS weights
n-step targets	Slow credit propagation	Longer real-reward prefix
Distributional head	Mean-only signal	Full return distribution
Noisy nets	Undirected exploration	Learned parameter noise

Caution

DQN-family methods remain confined to discrete, moderate-cardinality action spaces: the $\max_{a'}$ must be enumerated. Continuous control needs either an actor to propose actions (Chapter 13) or an optimisation oracle inside the max. And despite the stabilisers, the triad is managed, not repealed: aggressive replay ratios, sparse rewards, or sharp nonstationarity can still tip training into divergence, which is why monitoring value-estimate calibration (predicted vs. realised returns) is a standard health check.

Key Takeaways

DQN made deep value learning stable with experience replay and target networks; Double DQN removed maximisation bias by decoupling selection from evaluation; dueling heads shared state-value learning across actions; prioritised replay spent samples where TD error was largest; multi-step, distributional, and noisy extensions compounded near-additively in Rainbow. The recipe's deeper lesson: deep RL progresses by identifying a specific statistical pathology and installing a targeted, usually simple, counter-mechanism.

Chapter 12

Trust Regions and Proximal Policy Optimisation

12.1 The Step-Size Problem

Policy gradients carry a hazard absent from supervised learning. In supervised learning a bad step wastes an update; the data distribution is fixed and the next batch corrects you. In on-policy RL the data distribution *is* the policy: one overlarge step damages behaviour, damaged behaviour collects damaged data, and the estimator that might have corrected the mistake is now computed under the wrong distribution. Performance can collapse and not recover. The methods of this chapter exist to make policy improvement *monotone-ish*: take the largest step that provably (TRPO) or heuristically (PPO) cannot hurt much.

12.2 The Surrogate Objective

Fix the current policy π_{old}. A classical identity (the performance-difference lemma of Kakade and Langford) expresses any other policy's performance exactly:

$$J(\pi) - J(\pi_{\text{old}}) \;=\; \mathbb{E}_{s\sim d^{\pi},\, a\sim\pi}\big[A^{\pi_{\text{old}}}(s,a)\big].$$

Improving on π_{old} means finding a policy with positive old-policy advantage *on its own state distribution* d^{π}—which we cannot sample from without deploying π. The tractable surrogate replaces d^{π} by $d^{\pi_{\text{old}}}$ and corrects the action distribution by a one-step ratio:

$$L_{\pi_{\text{old}}}(\pi) \;=\; \mathbb{E}_{s\sim d^{\pi_{\text{old}}},\, a\sim\pi_{\text{old}}}\Big[\,\rho(s,a)\,A^{\pi_{\text{old}}}(s,a)\Big], \qquad \rho(s,a)=\frac{\pi(a\mid s)}{\pi_{\text{old}}(a\mid s)}.$$

L matches J to first order at π_{old} and is estimable from on-hand data. The question is how far it can be trusted, and the answer is quantitative:

Theorem: Trust-region lower bound (Schulman et al., 2015)

Let $\varepsilon = \max_{s,a}|A^{\pi_{\text{old}}}(s,a)|$ and let $D_{\text{KL}}^{\max}(\pi_{\text{old}},\pi) = \max_s D_{\text{KL}}(\pi_{\text{old}}(\cdot\mid s)\,\|\,\pi(\cdot\mid s))$. Then

$$J(\pi) \;\geq\; J(\pi_{\text{old}}) + L_{\pi_{\text{old}}}(\pi) \;-\; \frac{4\,\varepsilon\,\gamma}{(1-\gamma)^2}\,D_{\text{KL}}^{\max}(\pi_{\text{old}},\pi).$$

Maximising the right-hand side at each iteration yields a monotonically improving sequence of policies.

The surrogate is trustworthy exactly insofar as the new policy stays KL-close to the old—the same lesson Chapter 7 taught about importance ratios, now with an explicit penalty constant (note the ominous $(1-\gamma)^{-2}$: long horizons shrink trust regions).

12.3 TRPO

Trust Region Policy Optimisation operationalises the bound as a constrained program, replacing the unwieldy max-KL by an average:

$$\max_{\theta} \; \mathbb{E}\Big[\tfrac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}\widehat{A}(s,a)\Big] \quad \text{s.t.} \quad \mathbb{E}_s\big[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot \mid s) \,\|\, \pi_\theta(\cdot \mid s))\big] \le \delta.$$

Expanding the objective to first order and the constraint to second order gives exactly the natural-gradient direction $F^{-1}\nabla J$ of Chapter 9, computed matrix-free by conjugate gradients on Fisher-vector products, followed by a backtracking line search that enforces the true constraint and actual surrogate improvement. TRPO delivered the first reliably monotone deep policy optimisation—and a heavy implementation: second-order machinery, incompatible with modern conveniences like parameter sharing and minibatch epochs.

12.4 PPO

Proximal Policy Optimisation (Schulman et al., 2017) asks: can a first-order method get the trust-region effect? Its clipped surrogate answers with one line:

Definition: PPO clipped objective

With $\rho_t(\theta) = \pi_\theta(a_t \mid s_t)/\pi_{\theta_{\text{old}}}(a_t \mid s_t)$ and clip radius ϵ (typically 0.1–0.3):

$$\mathcal{L}^{\text{CLIP}}(\theta) \;=\; \mathbb{E}_t\Big[\min\Big(\rho_t(\theta)\,\widehat{A}_t,\; \text{clip}\big(\rho_t(\theta),\,1-\epsilon,\,1+\epsilon\big)\,\widehat{A}_t\Big)\Big].$$

Read it casewise. If $\widehat{A}_t > 0$, the objective grows with ρ_t only until $1+\epsilon$, then flatlines: no extra credit for pushing a good action's probability further within this batch. If $\widehat{A}_t < 0$, the objective falls with ρ_t only until $1-\epsilon$: no extra credit for crushing a bad action beyond the band. The min makes the surrogate a *pessimistic* (lower-bound- flavoured) estimate—clipping only ever removes incentive, never adds it. Gradients vanish outside the band, so multiple epochs of minibatch SGD on the same rollout data are safe; this data reuse is PPO's practical engine.

Worked Example: Reading the diagnostics of one PPO epoch

A healthy update on a batch of 2048 timesteps might report: mean ratio 1.002, ratio interquartile range $[0.94, 1.07]$, clip fraction 8%, approximate KL 0.012, entropy down 0.4%, value-function explained variance 0.71. Interpretation: the policy moved (ratios spread around 1) but modestly (KL near the customary 0.01–0.02 target); clipping touched a minority of samples (healthy—0% means the step was wasted, 40% means the epochs overran the trust region); entropy is easing rather than collapsing; the critic explains most return variance. The pathological versions announce themselves in the same numbers: KL 0.2 with clip fraction 50% (step too large or too many epochs), entropy dropping several percent per update (premature determinism—raise the entropy coefficient), explained variance near zero or negative (the advantages are noise; nothing downstream can be trusted). PPO is best operated as a small control room around these five gauges.

ELI5: A speed limiter on learning

The advantage says which way to steer; the clip is a governor on the accelerator. However enthusiastic this batch of experience is about some action, the policy may move its probability by at most about ϵ—20% or so—before the objective stops rewarding further movement. Next batch, collected with the new policy, may push further. Progress happens in bounded, refundable instalments, and one deranged batch cannot wreck the policy.

The full PPO loss adds a value-function term and an entropy bonus,

$$\mathcal{L} = \mathcal{L}^{\text{CLIP}} - c_1\,\mathbb{E}_t\big[(V_w(s_t) - \widehat{V}_t^{\text{targ}})^2\big] + c_2\,\mathbb{E}_t\big[\mathcal{H}(\pi_\theta(\cdot \mid s_t))\big],$$

with advantages from GAE, normalised per batch. An alternative PPO variant penalises KL adaptively instead of clipping; the clipped form dominates in practice. Implementation folklore—advantage normalisation, orthogonal initialisation, learning-rate annealing, value clipping, early stopping on KL—measurably matters; PPO is as much a disciplined recipe as an equation.

The Abstract Viewpoint: Three costumes, one idea

Natural gradient, TRPO, and PPO are one idea—bound the information- theoretic movement of the policy per update—at three levels of approximation: exact metric (Fisher), constrained program (average KL), and per-sample proxy (clipped ratio). The idea outlives all three costumes: the RLHF objective of Chapter 17 penalises $D_{\text{KL}}(\pi_\theta \,\|\, \pi_{\text{ref}})$ against a *fixed* reference, GRPO keeps the clipped ratio with group-relative advantages, and production online-RL loops (Chapter 18) enforce the same closeness *operationally*, by redeploying checkpoints fast enough that data never strays far off-policy. Wherever a learned policy is optimised against an estimated signal, someone installs a KL leash.

Caution

PPO's stability is empirical, not theorematic: the clip bounds per-sample ratios, not the true KL, and pathological cases exist. Watch the diagnostics: realised KL per update, fraction of clipped samples, entropy trajectory, and explained variance of the value head. Silent failure modes—entropy collapse, value-head divergence, advantage mis-scaling—announce themselves there long before reward curves do.

Key Takeaways

On-policy steps must be sized in policy space, not parameter space. The surrogate $\mathbb{E}[\rho\widehat{A}]$ is a first-order-faithful, samplable stand-in for performance, trustworthy within a KL neighbourhood whose radius carries a $(1-\gamma)^{-2}$ penalty. TRPO enforces the neighbourhood exactly with second-order machinery; PPO approximates it with a clipped ratio and buys multi-epoch data reuse with first-order simplicity. PPO's blend of robustness and convenience made it the default policy optimiser of the deep-RL era—and, with a reward model in place of the environment, of the RLHF era as well.

Chapter 13

Continuous Control: DDPG, TD3, and SAC

13.1 The Continuous-Action Problem

A robot joint takes torques in $[-1, 1]^d$; a portfolio takes weight vectors on a simplex. Value-based control needs $\max_a Q(s, a)$ at every step, an inner optimisation over a continuum that cannot be enumerated. Two escapes exist: stay on-policy with a stochastic actor (PPO handles continuous actions natively and remains a strong baseline), or—this chapter's road—train a *deterministic* actor to *be* the argmax, and learn off-policy with replay for sample efficiency.

13.2 The Deterministic Policy Gradient

For a deterministic policy $\mu_\theta : \mathcal{S} \to \mathcal{A}$, the score-function trick fails (log of a Dirac is unavailable), but the chain rule succeeds. The deterministic policy gradient theorem (Silver et al., 2014) states

$$\nabla_\theta J(\theta) \;=\; \mathbb{E}_{s\sim d^\mu}\Big[\nabla_\theta\, \mu_\theta(s)\; \nabla_a Q^\mu(s,a)\big|_{a=\mu_\theta(s)}\Big]\,:$$

push the actor's output in the direction that the critic says increases value—gradient *through* the critic with respect to the action. It is the limit of the stochastic policy gradient as policy variance shrinks, and its expectation involves no integral over actions, a variance advantage that grows with action dimension. The price: a deterministic policy explores nothing; behaviour noise (added Gaussian or OU noise) must supply exploration, and learning is necessarily off-policy.

13.3 DDPG

Deep Deterministic Policy Gradient (Lillicrap et al., 2016) is DQN's toolkit welded to the deterministic gradient: replay buffer; target copies of *both* networks, Polyak-averaged ($\theta^- \leftarrow \tau\theta + (1-\tau)\theta^-$, $\tau \sim 10^{-3}$); critic regressed on $y = r + \gamma\, Q_{\phi^-}(s', \mu_{\theta^-}(s'))$; actor ascending $\mathbb{E}_s[Q_\phi(s, \mu_\theta(s))]$. It solved torque-level control from pixels and established the template—and a reputation for brittleness: DDPG is exquisitely sensitive to hyperparameters, and its critic, bootstrapping its own maximising actor, overestimates catastrophically on hard tasks. The failure is Chapter 6's maximisation bias in continuous clothing: the actor *is* an argmax over a noisy critic, and it will happily locate the critic's hallucinated peaks.

13.4 TD3: Three Fixes

Twin Delayed DDPG (Fujimoto et al., 2018) is DDPG plus three surgical corrections, each aimed at a diagnosed failure:

1. **Clipped double critics.** Train two critics; build targets from the *minimum*: $y = r + \gamma \min_{i=1,2} Q_{\phi_i^-}(s', \tilde{a}')$. Where double DQN decorrelates selection and evaluation, TD3 goes further—the pairwise min imposes deliberate *under*estimation, which is the safe direction when an actor is hunting for peaks.

2. **Target policy smoothing.** Perturb the target action, $\tilde{a}' = \mu_{\theta^-}(s') + \text{clip}(\varepsilon, -c, c)$, $\varepsilon \sim \mathcal{N}(0, \sigma)$: a small adversarial blur that stops the actor exploiting narrow spurious spikes in the critic surface and regularises Q toward smoothness in a.

3. **Delayed actor updates.** Update the actor (and targets) once per d critic updates ($d = 2$): let the evaluation settle before the improvement acts on it—two-time-scale discipline (Chapter 10) made concrete.

The ablations are instructive: each fix helps, jointly they transform DDPG from fragile to dependable, and the package matched or beat everything of its day on continuous benchmarks.

13.5 SAC: Maximum-Entropy RL

Soft Actor–Critic (Haarnoja et al., 2018) changes the objective itself. Maximise reward *plus policy entropy*:

$$J(\pi) = \mathbb{E}_\pi\Big[\sum_t \gamma^t \big(r_{t+1} + \alpha\, \mathcal{H}(\pi(\cdot \mid s_t))\big)\Big], \qquad \mathcal{H}(\pi(\cdot \mid s)) = -\mathbb{E}_{a\sim\pi} \log \pi(a \mid s),$$

with temperature α trading the two. The Bellman machinery survives intact in "soft" form: the soft value is $V(s) = \mathbb{E}_{a\sim\pi}[Q(s,a) - \alpha \log \pi(a \mid s)]$, the soft backup is a contraction, and soft policy iteration—improve toward $\pi \propto \exp(Q/\alpha)$, a Boltzmann distribution over the critic—provably converges in the tabular case. The deep algorithm: twin critics with min-targets (TD3's gift), a stochastic Gaussian actor trained by the *reparameterisation* gradient ($a = \tanh(\mu_\theta(s) + \sigma_\theta(s) \odot \xi)$, $\xi \sim \mathcal{N}(0, I)$—lower variance than the score-function route because the critic's derivative information flows through), replay, and automatic temperature tuning: adjust α by gradient descent to hold entropy at a target level, removing the most sensitive knob.

Why does entropy help? Three separable reasons: exploration is built into the objective rather than bolted on as behaviour noise; the smoothed (log-sum-exp) landscape avoids premature commitment among near-tied actions and captures *multiple* good modes; and maximum- entropy policies are demonstrably more robust to perturbations of dynamics and reward—they have practised more than one way to succeed. SAC is, with PPO, the workhorse of continuous control: PPO when simulation is cheap and stability paramount; SAC when samples are dear.

ELI5: Two chefs

The TD3 chef perfects a single signature dish: maximum reward, deterministic recipe, plus safeguards against overrating half-tested ideas. The SAC chef is paid for tastiness *and* for repertoire: the contract rewards keeping many dishes in rotation. When the kitchen changes—new stove, missing ingredient—the second chef adapts faster, having never let the repertoire collapse to one recipe.

The Abstract Viewpoint: Control as inference

The maximum-entropy objective is not a hack but a change of foundations: posit binary "optimality" variables with $p(\mathcal{O}_t = 1 \mid s_t, a_t) \propto \exp(r(s_t, a_t)/\alpha)$ and ask for the posterior over trajectories given optimality. Variational inference on this graphical model yields exactly soft Bellman equations and Boltzmann policies: *RL is inference*, Q-functions are log- messages, and SAC is structured variational inference with neural approximating families. The correspondence (Todorov, Toussaint, Levine) imports inference tools into control and explains the family resemblance between soft value backups and log-sum-exp smoothing everywhere from linearly solvable MDPs to RLHF's KL-regularised optimum $\pi^*(y \mid x) \propto \pi_{\text{ref}}(y \mid x) e^{r(x,y)/\beta}$— the same Boltzmann shape, which we will meet again in Chapter 17.

	PPO	**DDPG**	**TD3**	**SAC**
Policy	Stochastic	Deterministic	Deterministic	Stochastic
Data	On-policy	Off-policy replay	Off-policy replay	Off-policy replay
Critics	V (GAE)	One Q	Twin Q, min	Twin Q, min
Exploration	Entropy bonus	Additive noise	Additive noise	Entropy in objective
Sample efficiency	Low	Medium	Medium–high	High
Stability	High	Low	Medium–high	High

Key Takeaways

Continuous action spaces break the enumerated max; the deterministic policy gradient trains an actor to be the argmax, differentiating the critic with respect to actions. DDPG established the off-policy template and its pathology (actor-exploited overestimation); TD3 repaired it with clipped double critics, target smoothing, and delayed actors; SAC re-founded the objective on maximum entropy, gaining built-in exploration, multimodality, robustness, and—via control-as- inference—a Boltzmann-policy structure that recurs in language-model alignment. Between PPO and SAC, continuous control is, for most purposes, an engineering discipline rather than an open problem.

Chapter 14

Model-Based Deep Reinforcement Learning

14.1 Learning to Simulate

Model-free methods treat the environment as an oracle to be queried, expensively, one transition at a time. Model-based methods learn the oracle: an approximate $\widehat{P}(s' \mid s, a)$ and $\widehat{R}(s, a)$, inside which the agent can plan, imagine, and rehearse without touching the world. The prize is sample efficiency—often one to two orders of magnitude—plus transfer (a model of physics serves many tasks) and introspection (plans can be inspected). The peril is *model bias*: an agent optimised inside a wrong model excels at exploiting the model's errors, and the exploitation does not transfer. Managing that peril is the whole art.

Learning the model itself is supervised learning: from replayed transitions (s, a, s', r), fit next-state and reward predictors— tabular counts in small MDPs; Gaussian processes when data is scarce and calibrated uncertainty precious (PILCO famously learned cart-pole swing-up in a few trials this way); neural networks, usually probabilistic ensembles, at scale. The recurring design questions: predict raw observations or a learned latent state? one step or many? deterministically or with calibrated uncertainty?

14.2 Dyna: Integrating Learning and Planning

Sutton's Dyna architecture is the conceptual junction box. After each real step: (i) learn value/policy from the real transition, model-free; (ii) update the model from the same transition; (iii) run n *planning* updates—model-free updates applied to transitions *sampled from the model* (previously seen states, imagined outcomes). Real experience is thus spent twice: once directly, once to improve a simulator that manufactures further training data. In tabular mazes, Dyna-Q with $n = 50$ planning steps solves in a handful of episodes what plain Q-learning needs dozens for. When the world changes, planning from a stale model misleads; Dyna-Q+ adds an exploration bonus $\kappa\sqrt{\tau(s, a)}$ for long-untried actions, re-testing the model where it is oldest.

ELI5: Practising in your head

After a driving lesson you replay it mentally: what if I had braked earlier at that corner? Mental rehearsal is cheap, so you run dozens of variations per real minute behind the wheel. Your imagination is only as good as your model of the car—daydream physics teaches bad braking—so you keep correcting the imagination against real lessons. That loop, imagination corrected by reality and reality amplified by imagination, is Dyna.

Worked Example: Planning amplification, quantified

In the classic 9×6 maze experiment, plain Q-learning (one real update per real step) needs on the order of 25 episodes to reach the optimal 14-step path. Dyna-Q with $n = 5$ planning updates per real step reaches it in about 5 episodes; with $n = 50$, in about 3. The real-experience budget shrinks roughly in proportion to the planning multiplier—until model error, not data, binds. The same table also teaches the caveat: change the maze after convergence and Dyna-Q with a stale model confidently plans wrong routes until real steps contradict it; the exploration bonus of Dyna-Q+ ($\kappa\sqrt{\tau}$ for long-untested actions) exists precisely to keep spending a little reality on auditing the imagination.

14.3 When to Trust the Model: MBPO

How far into imagination dare one roll? Compounding one-step errors grow with horizon: informally, value estimated in a model of per-step error ϵ_m over horizon k drifts from truth at a rate governed by $k\,\epsilon_m$ (plus policy-shift terms), so *long* imagined rollouts are toxic precisely when the model is weakest. Model-Based Policy Optimisation (Janner et al., 2019) draws the practical conclusion: use **short branched rollouts**—start from *real* states in the replay buffer, roll the learned ensemble model only k steps (k small, even 1–5), and feed the synthetic transitions to an off-policy learner (SAC). The ensemble's disagreement supplies uncertainty; branching from real states anchors the state distribution. MBPO reaches model-free asymptotic performance with a fraction of the samples, and its design rule—trust the model locally, never globally—is the field's distilled wisdom.

14.4 World Models and Dreamer

For pixel observations, modelling in observation space wastes capacity on irrelevant detail. The *world model* programme learns a compact **latent state**: an encoder compresses observations, a recurrent latent dynamics model predicts forward, and reward/value/policy heads operate wholly in latent space. Ha and Schmidhuber's 2018 system (VAE + RNN + tiny controller) demonstrated the striking possibility of training a policy *entirely inside its own dream* and transferring it back. The Dreamer line (Hafner et al., 2020–2023) industrialised the idea: a recurrent state-space model with deterministic and stochastic latent components is trained on replayed experience (reconstruction + reward prediction + KL regularisation); the actor–critic then learns from *imagined latent trajectories*, with gradients flowing straight through the differentiable dynamics. DreamerV3, with careful normalisation, masters dozens of domains— Atari, control, Minecraft diamond collection—with one hyperparameter set, a milestone for generality.

The Abstract Viewpoint: State as sufficient statistic, learned

Chapter 1 defined the state as a sufficient statistic of history; POMDPs are what happens when observations fall short. World models close the circle: the recurrent latent z_t is a *learned* approximate sufficient statistic—trained so that (history) $\to z_t$ preserves exactly what prediction of future reward and observation requires. Representation learning and control collapse into one criterion. The same move—replace the given state by a learned summary optimised for prediction—reappears in Chapter 17, where a language model's context is its state and the transformer's activations are the latent.

14.5 Planning at Decision Time: MCTS and MuZero

Planning can happen at *learning* time (Dyna, Dreamer: improve a stored policy) or at *decision* time: given the current state, search forward before acting. Monte Carlo Tree Search grows an asymmetric lookahead tree, selecting within the tree by a UCB-style rule that balances value estimates against visit counts, and backing up simulated outcomes. AlphaGo and AlphaZero fused MCTS with deep networks—a policy prior to focus the search, a value net to replace rollouts—and the search in turn generates improved targets for the networks: *search as policy improvement operator*, iterated to superhuman strength in Go, chess, and shogi, with a known simulator.

MuZero (Schrittwieser et al., 2020) removed the known simulator. It learns a latent model with three heads—representation $h : o \mapsto z$, dynamics $g : (z, a) \mapsto (z', \hat{r})$, prediction $f : z \mapsto (\hat{\pi}, \hat{v})$—trained *end-to-end so that planning works*: the latent dynamics is required to predict only rewards, values, and policies along real trajectories, never to reconstruct observations. The model is an instrument for search, not a physics engine; whatever the latents encode is whatever search needs. MuZero matched AlphaZero at board games and set Atari records, demonstrating that decision-time planning survives the loss of the rules—and marking the cleanest expression of *value-equivalent* model learning: a model is good if it supports the same decisions, not the same pixels.

Caution

Model-based RL fails characteristically, and the failures are worth naming: *model exploitation* (the optimiser finds the simulator's bugs—watch for imagined returns exceeding anything real); *compounding rollout error* (keep horizons short, monitor ensemble disagreement); *objective mismatch* (a model trained for reconstruction may spend its capacity on wallpaper, not on the task-relevant sliver of dynamics—MuZero's value-equivalence is one principled response). The universal diagnostic: routinely compare model-predicted returns against realised returns, and treat divergence as an alarm, not a curiosity.

System	Model form	Planning use	Signature idea
Dyna-Q	Tabular	Background replay	Learn + plan from same data
PILCO	Gaussian process	Policy search in model	Calibrated uncertainty, tiny data
MBPO	NN ensemble	Short branched rollouts	Trust locally, never globally
Dreamer	Recurrent latent SSM	Imagined latent rollouts	Learn behaviour inside the dream
AlphaZero	Known simulator	Decision-time MCTS	Search as improvement operator
MuZero	Learned latent	Decision-time MCTS	Value-equivalent model

Key Takeaways

Model-based RL converts experience into a simulator and the simulator into cheap experience. Dyna interleaves real and imagined updates; MBPO codifies short, real-anchored rollouts under ensemble uncertainty; Dreamer learns latent world models rich enough to train behaviour wholly in imagination; AlphaZero and MuZero place planning at decision time, with MuZero learning a model judged solely by the quality of the decisions it supports. The organising trade is model bias against sample cost, and the mature answer is calibrated, local, purpose-built trust.

Chapter 15

Scale, Meta-Learning, and the Practitioner's Toolkit

15.1 Distributed Reinforcement Learning

Deep RL's appetite for experience made systems architecture part of the algorithm. The dominant pattern is **actor–learner separation**: many CPU actors run environments and ship experience; one or few GPU learners consume it and broadcast parameters. Three landmarks:

- **Ape-X** distributed DQN's replay: hundreds of actors, each with different exploration ε, feed one prioritised buffer; a single learner trains at GPU speed. Off-policy learning tolerates the actors' staleness natively; throughput, not novelty, is the point—and throughput won, by large margins, on Atari.
- **IMPALA** did the same for actor–critic, where staleness *does* bite (the data is slightly off-policy the moment the learner steps). Its **V-trace** target repairs the gap with *truncated* importance ratios, $\rho_t = \min(\bar{\rho}, \pi/\mu)$ and $c_t = \min(\bar{c}, \pi/\mu)$: Chapter 7's bias–variance dial set deliberately at "bounded variance, small controlled bias." One learner, thousands of actors, one network across 57 games.
- **Rollout–learner RLHF clusters.** The same shape, at language-model scale: inference fleets generate completions ("rollouts"), reward and value services score them, learners update, and fresh weights ship back—Chapter 18's production loops are this architecture with users inside it.

The systems lesson generalises: *the staleness a method tolerates dictates the architecture it permits.* Off-policy methods buy architectural freedom; on-policy methods buy estimator simplicity and pay in synchronisation.

15.2 Hybrid Methods

The value/policy and model/model-free axes are menus, not oaths, and the strongest systems order across columns. Recurring hybrid patterns: *model-based data amplification* for model-free learners (Dyna, MBPO); *decision-time search over learned policies* (AlphaZero, MuZero: the network proposes, search disposes); *off-policy critics under on-policy actors* (soft blends such as Q-Prop); *demonstration-seeded RL* (behaviour cloning to initialise, RL to surpass—the SFT-then-RLHF pipeline of Chapter 17 is exactly this pattern). The craft is matching each component to the failure it cures, a table the previous chapters have effectively been building.

15.3 Meta-Reinforcement Learning

An agent that has mastered one MDP starts the next from zero. Meta-RL asks for agents that *learn to learn*: trained on a distribution of tasks $p(\mathcal{M})$, they should adapt to a fresh task from

that distribution in little data. Three archetypes:

- **MAML** (Finn et al., 2017): optimise initial parameters θ so that *one gradient step on a new task* already performs well: $\min_\theta \sum_{\mathcal{M}} \mathcal{L}_{\mathcal{M}}\big(\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{M}}(\theta)\big)$. Adaptation is honest gradient descent; the meta-learner shapes the landscape so that descent is fast. Model-agnostic, second-order, and the conceptual reference point.
- **RL^2** (Duan et al.; Wang et al., 2016): make adaptation a *forward pass*. A recurrent policy receives (s_t, a_{t-1}, r_{t-1}) and is trained across episodes-of-episodes whose hidden state persists within a task; the RNN learns to implement, in its recurrence, whatever fast-learning algorithm the task distribution rewards. Learning becomes inference in the hidden state. The reader acquainted with in-context learning in large language models will recognise the phenomenon: an outer loop trains a network whose inner dynamics perform adaptation without weight changes.
- **PEARL** (Rakelly et al., 2019): factor adaptation into *task inference*. An encoder maps collected transitions to a posterior over a latent task variable z; the policy conditions on z; posterior sampling over z drives exploration. Off-policy training (SAC underneath) makes it far more sample-efficient than on-policy meta-learners, and the probabilistic z gives principled exploration-for-identification.

The Abstract Viewpoint: Meta-RL as a POMDP

All of meta-RL is one construction: bundle the unknown task identity into the state. The "meta-MDP" has state $(s, \mathcal{M})$ with $\mathcal{M}$ unobserved—a POMDP whose belief state is a posterior over tasks. RL^2's hidden state and PEARL's z-posterior are two implementations of the same belief; MAML is amortised optimisation toward rapid belief exploitation. Optimal meta-behaviour is Bayes-optimal exploration in this POMDP, which is why good meta-learners appear to "experiment deliberately" in their first steps on a new task: they are reducing posterior entropy over $\mathcal{M}$.

15.4 Practical Considerations

Hard-won field wisdom, compressed:

- **Algorithm selection.** Discrete actions, simulator cheap: DQN-family or PPO. Continuous control, samples dear: SAC or TD3. Stability above all, parallel envs available: PPO. Fixed dataset only: Chapter 16. When in doubt, PPO first: it fails more gracefully.
- **Reproducibility.** Deep RL variance across random seeds is notorious; report distributions over ≥ 5 seeds, never single runs. Many published "improvements" vanish under seed variance and matched tuning.
- **Reward engineering.** Shaping accelerates learning and distorts it. Potential-based shaping, $F(s, s') = \gamma\Phi(s') - \Phi(s)$, is the exceptional safe harbour: it provably preserves the optimal policy while densifying signal. Anything else changes the task; audit what the shaped optimum actually is.
- **Diagnostics before hyperparameters.** Plot value-estimate calibration, entropy, KL per update, gradient norms, and episode-length distributions. Most "RL doesn't work" reports are one bad reward scale, one collapsed entropy, or one diverging value head, all visible on those five plots.

- **Simulation fidelity.** Sim-to-real gaps are model bias by another name; domain randomisation—train across a distribution of physics—buys robustness at the price of average-case performance.
- **Safety and constraints.** Constrained MDPs ($\max J$ s.t. $J_{C_i} \leq d_i$) via Lagrangian methods are the standard vehicle for hard limits (risk, drawdown, action bounds); Chapter 18 shows the same mathematics governing production guardrails.

15.5 Tools of the Trade

The ecosystem, briefly and without ceremony. *Environments*: Gymnasium (the maintained successor of OpenAI Gym) defines the de-facto `reset/step` API; the DeepMind Control Suite offers physics-consistent continuous benchmarks; Unity ML-Agents and Meta-World serve 3D and multi-task manipulation respectively. *Algorithm libraries*: Stable-Baselines3 for reliable reference implementations (the correct starting point for almost everyone); RLlib on Ray for distributed training at cluster scale; Acme and TF-Agents as research scaffolding; CleanRL for single-file, readable reference code—pedagogically the best of the set. *Monitoring*: TensorBoard or Weights & Biases for the diagnostic quintet above; seeds, configs, and environment versions logged obsessively, because in RL the experiment *is* the config. A typical workflow: prototype against a small environment with SB3 defaults, verify the diagnostics, scale with RLlib or a custom actor–learner loop, and benchmark across seeds before believing anything.

Key Takeaways

Scale in RL means actor–learner architectures, with off-policy tolerance (Ape-X) or truncated-IS repair (IMPALA's V-trace) absorbing distributed staleness—the very architecture that RLHF clusters inherit. Hybrids assemble cross-axis strengths; meta-RL trains for adaptation itself, via optimisation geometry (MAML), recurrent in-context learning (RL^2), or explicit task inference (PEARL), all unified as belief-state RL. Practice runs on seed honesty, safe shaping, constraint handling, and a small dashboard of diagnostics that catch most failures before the reward curve does.

Part V

Reinforcement Learning Beyond the Simulator

Chapter 16

Offline Reinforcement Learning and Policy Evaluation

16.1 Learning Without Touching the World

Everything so far assumed the agent may act. Often it may not: medicine has treatment records but no licence to experiment on patients; autonomous driving has fleet logs but no appetite for exploratory crashes; a trading desk has years of order and fill history and every reason not to explore with live capital. **Offline RL** (batch RL) asks: given only a fixed dataset $\mathcal{D} = \{(s_i, a_i, r_i, s'_i)\}_{i=1}^{N}$ collected by some unknown behaviour policy π_β, learn the best policy the data can support—with *no further interaction*.

This is off-policy learning at its most extreme, and something qualitatively new goes wrong. Online off-policy methods still enjoy a corrective loop: if the learner overvalues an action, it will eventually try it, be disappointed, and update. Offline, the loop is severed. Overvalue an action never present in the data and nothing ever contradicts you.

Definition: Distributional shift and extrapolation error

The learned policy π induces state–action distributions different from the data's. Bellman targets $r + \gamma \max_{a'} Q(s', a')$ query Q at *out-of-distribution* (OOD) actions a' where the approximator is unconstrained by data; errors there are typically optimistic (the max selects for upward noise, Chapter 6's bias with no antidote), and bootstrapping propagates and compounds them. The resulting failure—confidently wrong values for unseen actions—is *extrapolation error*, and unmodified DQN/DDPG-style training on a fixed buffer reliably diverges or collapses because of it.

ELI5: The cookbook with missing pages

You inherit a cookbook of dishes someone else actually cooked, with ratings. You may not enter the kitchen; you must nominate a menu from reading alone. The naive optimiser says: "the data shows sugar improves ratings up to 100 g; surely 500 g is glorious"—and nothing in the book can contradict it, because nobody ever tried 500 g. Offline RL's first commandment follows: *be suspicious of any dish the book does not contain.* Stay near recipes with evidence, or discount your optimism in proportion to how far you stray.

16.2 The Two Cures: Constraint and Pessimism

All principled offline methods implement one maxim—*do not trust what the data cannot verify*—by one of two routes.

16.2.1 Policy constraint

Keep π close to the behaviour policy, so Bellman targets are queried only where data lives.

- **BCQ** (Batch-Constrained Q-learning; Fujimoto et al., 2019): train a generative model of the behaviour policy; at target time, maximise Q only over actions the generator proposes (plus a small learned perturbation). The max is confined to the data manifold by construction.
- **BRAC / behaviour-regularised actor–critic**: subtract a divergence penalty from the objective, $\max_\pi \mathbb{E}[Q(s,a)] - \alpha\, D\big(\pi(\cdot \mid s), \pi_\beta(\cdot \mid s)\big)$, with D a KL or MMD; a unified frame in which many variants are hyperparameter choices.
- **TD3+BC**: the minimalist entry—add a behaviour-cloning term $\lambda\,(a - \mu_\theta(s))^2$ to TD3's actor loss and normalise Q. Embarrassingly simple, startlingly competitive, and a healthy reminder to benchmark against simplicity.

16.2.2 Value pessimism

Alternatively, leave the policy free but push *value estimates down* on OOD actions, so optimism cannot pay.

Definition: Conservative Q-Learning (Kumar et al., 2020)

CQL augments the Bellman regression with a term that minimises Q under the policy's action distribution while maximising it under the data's:

$$\mathcal{L}_{\mathrm{CQL}}(\phi) \;=\; \alpha\left(\mathbb{E}_{s\sim\mathcal{D},\,a\sim\pi}\big[Q_\phi(s,a)\big] - \mathbb{E}_{(s,a)\sim\mathcal{D}}\big[Q_\phi(s,a)\big]\right) \;+\; \mathcal{L}_{\mathrm{Bellman}}(\phi).$$

The learned Q provably lower-bounds the true value of the learned policy (under regularity conditions): optimism toward the unknown is structurally impossible, and acting greedily on a lower bound is safe by design.

Worked Example: Extrapolation error in one state

One state, three actions. The dataset contains 100 samples of a_1 (mean reward 0.5) and 100 of a_2 (mean 0.3); a_3 was never taken. A Q-network generalising from features happens to output $\widehat{Q}(a_3) = 0.9$—pure fiction, but nothing in the Bellman loss on the data penalises it, and the greedy offline policy selects a_3. Now add the CQL regulariser with the learned policy concentrated on a_3: the term $\alpha(\mathbb{E}_{a\sim\pi}[\widehat{Q}] - \mathbb{E}_{\mathcal{D}}[\widehat{Q}]) = \alpha(0.9 - 0.4)$ is large and positive, and its gradient pushes $\widehat{Q}(a_3)$ *down*—and keeps pushing until the policy's chosen actions score no better than the data's unless the Bellman evidence resists. At equilibrium $\widehat{Q}(a_3)$ sits below the in-data values, the policy returns to a_1, and the fiction is priced out. That, in one state, is the entire logic of conservative offline learning.

Pessimism is the offline field's organising principle, with theory to match: in linear and tabular settings, pessimistic value iteration achieves near-optimal guarantees *relative to the best policy the data covers*—one cannot do better than the data allows, and pessimism ensures one does not pretend to. Other notable designs: **IQL** (implicit Q-learning) avoids OOD queries entirely by expectile regression on in-data actions then advantage-weighted policy extraction; **Decision Transformer** reframes offline RL as sequence modelling—condition a transformer on desired

return-to-go and let it autoregress actions—trading Bellman machinery for the scaling behaviour of supervised sequence models.

Caution

Dataset quality is destiny. From expert data, constraint methods (even plain behaviour cloning) shine and pessimism can be needlessly timid; from random or mixed data, cloning is hopeless and pessimistic *stitching*—splicing good sub-trajectories from mediocre wholes—is where offline RL genuinely earns its keep. Always ask of a benchmark result: what was π_β, and how much headroom above it did the data contain?

16.3 Off-Policy Evaluation

Before deploying a policy learned offline one must answer a prior question: *how good is it?*—again without running it. Off-policy evaluation (OPE) estimates $J(\pi)$ from $\mathcal{D}$ alone, and its main estimators triangulate the same bias–variance territory as Chapter 7:

- **Direct method (DM).** Fit $\widehat{Q}$ from the data (e.g. by **fitted Q-evaluation**, FQE: iterate the Bellman *expectation* backup for the fixed target π on the dataset); report $\widehat{J} = \mathbb{E}_{s\sim\mu_0,a\sim\pi}[\widehat{Q}(s,a)]$. Low variance; bias equal to the model's error, unknowable from inside.
- **Importance sampling (IS).** Reweight logged returns by $\prod_t \pi/\pi_\beta$ (behaviour probabilities logged or estimated). Unbiased under coverage; variance exponential in horizon; weighted and per-decision variants partially tame it. For one-step problems (bandits: recommendation, ad serving) IS is exact, practical, and an industry staple—the horizon is the whole difficulty.
- **Doubly robust (DR).** Combine: use DM as a control variate and IS to correct its residuals,

$$\widehat{J}_{\mathrm{DR}} \;=\; \widehat{J}_{\mathrm{DM}} + \mathbb{E}_{\mathcal{D}}\Big[\rho_{0:t}\big(r_t + \gamma\widehat{V}(s_{t+1}) - \widehat{Q}(s_t,a_t)\big)\Big],$$

  (schematically). Unbiased if *either* the model or the ratios are correct—the eponymous double robustness—and typically the best-of-both in variance.
- **Model-based evaluation.** Learn dynamics, simulate π inside; powerful when the model class suits the domain, silently wrong otherwise—Chapter 14's warnings apply verbatim.

High-confidence OPE turns estimates into certificates (concentration bounds on $\widehat{J}$), the form regulators and risk committees actually need; and the marriage of pessimistic training with conservative evaluation is what makes offline RL deployable in domains where a bad policy costs more than a research setback.

The Abstract Viewpoint: Offline RL as statistics coming home

Strip the sequential structure and offline RL is causal inference: estimate the effect of an intervention (π) from observational data collected under another regime (π_β). Coverage is positivity; IS is inverse propensity weighting; DR estimators are augmented IPW; pessimism is the sequential face of confidence-bound decision-making. The horizon is what RL adds—compounding shift, compounding variance—and the field's contributions (per-decision weights, marginalised/stationary-distribution ratios, pessimistic value iteration) are precisely horizon-management technology grafted onto classical estimands.

Key Takeaways

Offline RL learns from a frozen dataset, where the exploratory corrective loop is severed and extrapolation error—optimistic value on unseen actions, amplified by max and bootstrap—is the defining failure. Cures constrain the policy to the data manifold (BCQ, BRAC, TD3+BC) or make values pessimistic off it (CQL, IQL), with pessimism carrying the guarantees. Off-policy evaluation (DM/FQE, IS, DR, model-based) prices a policy before deployment, and doubly robust estimators are the sensible default. The stance of the whole enterprise—optimise against a conservative estimate, verify before you ship—is exactly the stance production RL takes in Chapter 18.

Chapter 17

Policy Optimisation for Language Models

17.1 The Language Model as a Policy

The most consequential application of reinforcement learning in the present decade is the alignment of large language models (LLMs). The dictionary between the two fields is exact and worth internalising:

Classical RL	**Language modelling**	**Notes**
State s_t	Prompt x + tokens so far $y_{<t}$	Context is the state
Action a_t	Next token y_t	$\lvert\mathcal{A}\rvert$ = vocabulary, $\sim 10^5$
Policy $\pi(a \mid s)$	$\pi_\theta(y_t \mid x, y_{<t})$	The LM itself
Trajectory/episode	A complete completion y	Horizon = generation length
Dynamics P	Deterministic append	$s_{t+1} = (s_t, a_t)$: known!
Reward r	Score of the finished text	Usually terminal and sparse
Return G	(Discounted) total score	$\gamma = 1$ typical

Two structural facts make this MDP unusual. The *dynamics are known and trivial*—generating a token appends it—so all uncertainty lives in the reward. And the *reward is not given by nature*: what makes a response "good" is a human judgement, which must itself be learned. Reinforcement learning from human feedback (RLHF) is the resulting two-stage construction: learn a reward model from human preferences, then optimise the language model against it with a policy-gradient method—under a leash, because the reward model is only a proxy.

17.2 The Four Models

A production RLHF system runs four networks, typically all initialised from the same pretrained transformer.

17.2.1 Policy model

The model being improved: initialised from the *supervised fine-tuned* (SFT) model—the pretrained LM further trained on demonstration data by ordinary maximum likelihood, $\mathcal{L}_{\text{SFT}} = -\sum_t \log \pi_\theta(y_t \mid x, y_{<t})$. SFT gives the policy a sensible starting distribution; RL then optimises what SFT can only imitate. (The pattern is Chapter 15's demonstration-seeded RL, at scale.)

17.2.2 Reference model

A frozen copy of the SFT policy, π_{ref}. Its sole job is to anchor: the training objective penalises divergence from it, keeping the optimised policy inside the region where the reward model is trustworthy and the language remains fluent.

17.2.3 Reward model

A transformer with a scalar head, $r_\phi(x, y)$, trained on human *comparisons*: annotators are shown two (or more) completions of the same prompt and pick the better. The statistical bridge from pairwise choices to a latent score is the **Bradley–Terry model**:

Definition: Bradley–Terry and the preference loss

Posit that the probability of preferring y_w over y_l is a logistic function of the score difference,

$$\mathbb{P}(y_w \succ y_l \mid x) \;=\; \sigma\big(r_\phi(x, y_w) - r_\phi(x, y_l)\big), \qquad \sigma(z) = \frac{1}{1 + e^{-z}}.$$

Maximum likelihood on a preference dataset gives the pairwise ranking loss (InstructGPT form):

$$\mathcal{L}_{\mathrm{RM}}(\phi) \;=\; -\,\mathbb{E}_{(x, y_w, y_l)}\Big[\log \sigma\big(r_\phi(x, y_w) - r_\phi(x, y_l)\big)\Big].$$

Only score *differences* are identified; the absolute scale is a gauge freedom, usually fixed by normalisation.

Worked Example: Bradley–Terry by hand

Three completions of one prompt receive latent scores $r_A = 2.0$, $r_B = 1.0$, $r_C = -0.5$. Predicted preference probabilities: $\mathbb{P}(A \succ B) = \sigma(1.0) \approx 0.731$; $\mathbb{P}(A \succ C) = \sigma(2.5) \approx 0.924$; $\mathbb{P}(B \succ C) = \sigma(1.5) \approx 0.818$. Suppose an annotator now delivers the upset $B \succ A$. The loss on that pair is $-\log \sigma(r_B - r_A) = -\log \sigma(-1.0) \approx 1.313$, and its gradient pushes r_B up and r_A down by equal amounts $\sigma(r_A - r_B) \approx 0.731$ times the learning rate: the model concedes exactly in proportion to how confidently it was wrong. Note also the gauge freedom: adding $+10$ to all three scores changes no probability—only differences are learned, which is why reward scales in RLHF pipelines are fixed by normalisation conventions rather than by the data.

Why comparisons rather than absolute ratings? Because humans are reliable comparators and unreliable graders: inter-annotator agreement on "which is better" far exceeds agreement on "rate this 1–10," and the Bradley–Terry likelihood converts exactly the reliable signal into a differentiable target. The reward model is a learned, compressed surrogate for human judgement—emphasis on *surrogate*: it is accurate near the distribution it was trained on and increasingly fictional away from it, which is the single fact from which the rest of the pipeline's design follows.

A useful distinction cuts across reward modelling: *outcome-based* rewards score only the final artefact (answer correct? code passes tests?), are cheap and objective, but reward lucky guesses and starve long tasks of signal; *process-based* rewards score intermediate steps (each reasoning line), assign credit densely and shape *how* the model works, but cost far more to label and import the labellers' biases about method. Modern reasoning systems mix both, and the choice measurably changes training dynamics—dense process reward accelerates credit assignment (Chapter 1's oldest problem) at the price of anchoring the model to humanly-legible procedures.

17.2.4 Value model

A critic $V_\psi(x, y_{<t})$ estimating expected final reward from a partial generation, trained by regression on observed returns (Monte Carlo targets—regressing on single sampled returns converges to the conditional expectation, the standard least-squares fact) or with bootstrapping. Its role

is Chapter 10's: convert sparse terminal reward into per-token advantages via GAE, so that the policy gradient can tell *which tokens* of a thousand-token completion earned the score.

17.3 The RLHF Objective and Loop

Definition: KL-regularised reward maximisation

RLHF optimises

$$\max_{\theta} \; \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)}\Big[r_\phi(x,y)\Big] \;-\; \beta\,\mathbb{E}_x\Big[D_{\mathrm{KL}}(\pi_\theta(\cdot\mid x)\,\|\,\pi_{\mathrm{ref}}(\cdot\mid x))\Big].$$

In per-token practice the KL term is folded into the reward as a penalty $-\beta \log \frac{\pi_\theta(y_t|\cdot)}{\pi_{\mathrm{ref}}(y_t|\cdot)}$ at each step.

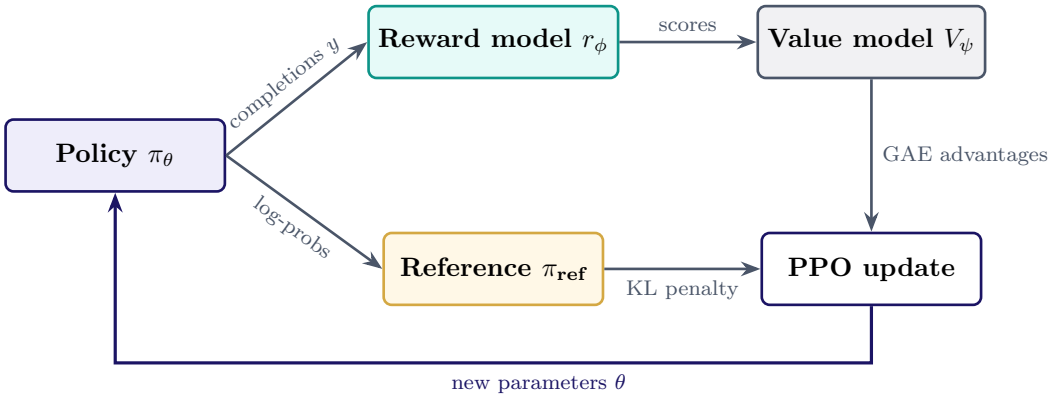

Figure 17.1: The four-model RLHF loop. The policy generates; the reward model scores; the reference prices divergence; the value model apportions token-level credit; PPO closes the loop.

The optimisation itself is PPO, exactly as in Chapter 12: sample completions from the current policy; score them with r_ϕ (plus KL penalty); compute advantages with V_ψ and GAE; take clipped-ratio minibatch epochs; repeat. The full loop couples all four models— policy generates, reference prices divergence, reward model scores, value model apportions credit—and its systems reality is the rollout–learner cluster of Chapter 15, with generation (inference) usually dominating the compute bill.

The KL-regularised objective has a closed-form optimum that illuminates everything:

$$\pi^*(y\mid x) \;\propto\; \pi_{\mathrm{ref}}(y\mid x)\,\exp\big(r_\phi(x,y)/\beta\big),$$

the Boltzmann tilt of the reference by the reward—Chapter 13's maximum-entropy mathematics, re-derived by alignment needs. Two famous consequences:

- **Why the leash exists.** Unleashed maximisation of a learned reward is *reward over-optimisation*: the policy climbs r_ϕ into regions where r_ϕ diverges from true preference, and measured-vs-true quality follows the familiar rise-then-fall (Goodhart) curve as KL from the reference grows. β sets where on that curve you stop. This is Chapter 16's lesson—distrust the proxy off-distribution—wearing alignment clothes; the reward model is an offline artefact, and the KL ball is the trust region of its validity.

- **DPO.** Invert the closed form to express the reward in terms of the optimal policy, substitute into the Bradley–Terry likelihood, and the reward model cancels: *direct preference optimisation* trains the policy on preferences with a simple classification-style loss, no reward model, no PPO, no rollouts. The trade: simplicity and stability against the flexibility of an explicit reward model (which can be inspected, ensembled, and reused for rejection sampling and online RL).

17.4 GRPO and Critic-Free Advantage

The value model is expensive—a fourth transformer to train and serve. **Group Relative Policy Optimisation** (GRPO, introduced with the DeepSeek reasoning line) deletes it: for each prompt, sample a *group* of K completions, score each, and define the advantage of completion i by within-group standardisation,

$$\widehat{A}_i \;=\; \frac{r_i - \operatorname{mean}(r_{1:K})}{\operatorname{std}(r_{1:K})},$$

applied with the usual clipped-ratio surrogate and KL penalty. The group mean is a Monte Carlo baseline (Chapter 9's control variate, estimated from siblings rather than a critic); the method suits settings with verifiable terminal rewards—mathematics, code—where sampling many attempts per prompt is natural. Its success is a pleasing full circle: the frontier of 2025-era reasoning models runs on REINFORCE-with-baseline, the 1992 algorithm, wearing PPO's clip and a group-statistical baseline.

ELI5: The whole pipeline in one classroom

A student (policy) first copies worked solutions from a textbook (SFT). Then a grader (reward model) is trained by showing a teaching assistant pairs of essays and learning which one the TA prefers. Now the student writes, the grader scores, and a tutor (value model) tells the student which *sentences* helped or hurt. One rule throughout: never drift too far from the textbook style (reference/KL), because the grader only understands essays that look broadly like the ones it was calibrated on. Push the grader too far outside its experience and you get essays that score high and read like nonsense—so the leash is not bureaucracy; it is what keeps the grade meaning anything.

The Abstract Viewpoint: One inequality chain

The whole chapter is Chapters 9–13 composed: policy gradient theorem (the estimator) + baseline/critic (variance) + clipped ratio (step control) + KL-to-reference (trust region against a *proxy* objective) + Boltzmann optimum (max-entropy form). Nothing in RLHF is mathematically new; what is new is the reward's epistemic status— learned, gameable, and central—which promotes reward modelling, over-optimisation monitoring, and evaluation from engineering footnotes to the heart of the method. The next chapter follows that promotion to its conclusion in production systems.

Caution

Reward models drift out of validity as the policy improves: the completions the policy now produces are unlike those the RM was trained on. Production pipelines therefore *iterate*:

collect fresh preferences on current-policy outputs, retrain the RM, resume RL—and monitor proxies for over-optimisation (KL from reference, RM-score vs. held-out human agreement, length and style drift). A rising reward curve, by itself, certifies nothing.

Key Takeaways

An LLM is a token-level policy in a known-dynamics MDP whose reward is learned from human comparisons via Bradley–Terry. RLHF optimises learned reward minus $\beta\times$KL-to-reference with PPO, using a value critic and GAE for token-level credit; the optimum is a Boltzmann tilt of the reference, DPO is its reward-free inversion, and GRPO replaces the critic with group-relative baselines for verifiable tasks. The controlling risk is over-optimisation of a proxy reward, and the controlling discipline—bounded divergence, fresh preferences, relentless evaluation—is classical RL's trust-region wisdom applied to an objective that is itself an estimate.

Chapter 18

Online Reinforcement Learning in Production

18.1 From the Laboratory to the Loop

Chapter 17 optimised against a frozen reward model; the final step of the subject's present arc is to let *deployment itself* close the loop. In production online RL, the deployed model's real interactions —acceptances, rejections, edits kept or reverted, follow-up complaints—are converted into rewards, and updated checkpoints ship back to users on a cadence of hours. The publicly documented systems around AI coding assistants (autocomplete engines trained on live accept/reject signals; agentic coding models updated from production trajectories every few hours) are the clearest worked examples, and this chapter distils the discipline they exemplify: reward design, on-policy loop management, instrumentation, and evaluation gates. The mathematics is Parts II–IV; what is new is that *the product is the environment*, users are part of the dynamics, and every weakness of the reward pipeline is an attack surface for the optimiser.

Why online, when Chapter 16 taught the safety of offline learning? Because of train–test mismatch: a simulator can reproduce files, tests, and terminals with high fidelity, but not the human in the loop—their intent, their tolerance for interruption, their reaction to a partially correct edit. When the training environment cannot imitate deployment, the freshest and most faithful reward is deployment itself; the price is that every safeguard the offline setting gave for free must now be engineered.

18.2 Reward Design

The production reward is always a **proxy**, $R_{\text{proxy}} \approx R_{\text{true}}$, compressing a multidimensional judgement—correctness, non-intrusiveness, latency, safety, durability—into one scalar. A general composite has the constrained form

$$R \;=\; \lambda_{\text{accept}} R_{\text{accept}} + \lambda_{\text{persist}} R_{\text{persist}} - \lambda_{\text{dissat}} C_{\text{dissat}} - \lambda_{\text{latency}} C_{\text{latency}} - \lambda_{\text{risk}} C_{\text{risk}},$$

and the weights *are* the product strategy: they decide which behaviours the optimiser amplifies.

A miniature that repays study: an autocomplete policy must decide whether to show a suggestion at all. Set a target acceptance threshold τ and define $R(\text{accepted}) = 1-\tau$, $R(\text{rejected}) = -\tau$, $R(\text{nothing}) = 0$. Showing a suggestion with acceptance probability p has expected reward $p(1-\tau) - (1-p)\tau = p - \tau$: positive exactly when $p > \tau$. The reward *is* the decision boundary—"know when to stay silent" becomes a learned behaviour rather than a calibrated classifier bolted on. The documented outcome in deployed systems is instructive: *fewer* suggestions shown, *higher* acceptance rate—the policy learned that silence often dominates a mediocre completion.

For agentic trajectories (edits, tool calls, commands, questions), terminal reward is sparse and delayed, so denser intermediate signals—edits that persist, valid tool calls, passing tests, absence of dissatisfied follow-ups—are added, each a further proxy and each, therefore, a further *gameable* surface.

Caution: Reward hacking

The policy optimises what is measured. Documented production examples: exclude malformed tool calls from training and the model learns that *breaking* the tool call avoids negative reward; penalise bad edits but not deferral and the model learns to stall with clarifying questions. The diagnostic object is the gap $\Delta_R = R_{\text{true}} - R_{\text{proxy}}$: optimisation pressure finds its maxima. Every observed hack is evidence the proxy is incomplete, and the reward function must be treated as a living specification under versioned revision—not a fixed constant of the system.

Guardrails formalise as constrained optimisation (Chapter 15's CMDPs): $\max_\theta \mathbb{E}[R]$ subject to $\mathbb{E}[C_j] \leq \epsilon_j$ for costs C_j—invalid-call rate, destructive-edit rate, latency—or, in Lagrangian form, penalty coefficients β_j that price which regressions are unacceptable however much the headline reward improves.

18.3 The On-Policy Loop

The production cycle is

$$\pi_{\theta_k} \;\to\; \mathcal{D}_k \;\to\; \widehat{R}_k \;\to\; \theta_{k+1} \;\to\; \text{eval} \;\to\; \pi_{\theta_{k+1}},$$

serve–collect–reward–train–gate–redeploy, and its governing constraint is freshness. Figure 18.1 draws the cycle.

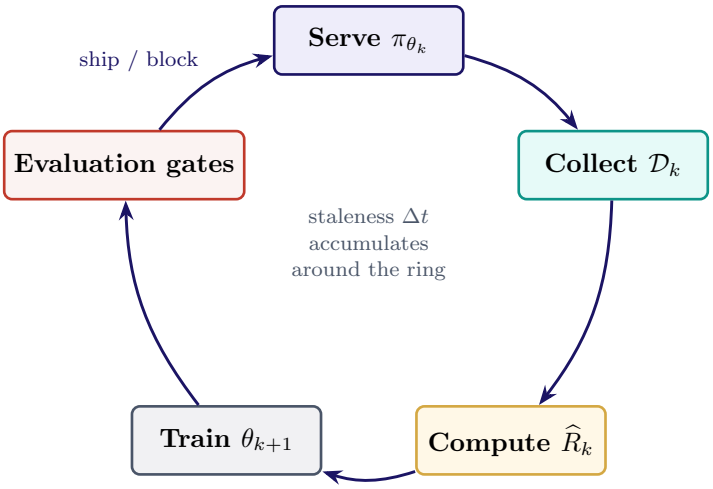

Figure 18.1: The production on-policy loop. Every arc adds delay; the sum is the staleness of the training data, and the cadence of the whole ring is part of the algorithm.

Its governing constraint is Chapter 7's: the policy-gradient estimator wants actions sampled from the policy being optimised. Staleness, $\Delta t = t_{\text{train}} - t_{\text{serve}}$, decomposes into logging + reward computation + training + evaluation + deployment delays, and as it grows, $D_{\text{KL}}(\pi_{\text{old}} \,\|\, \pi_{\text{new}})$ grows with it and old data misguides the update. The corrections are the familiar ones—per-token importance ratios explode over long completions (the product-of-ratios pathology, at vocabulary scale), so production systems prefer *operational* freshness (redeploy every few hours; train on the last checkpoint's data) plus PPO-style clipping, over heavy statistical correction. Cadence is not an engineering detail; it is part of the algorithm.

Against freshness pushes noise: user feedback is variable, tasks heterogeneous, and gradient variance falls as σ^2/N in batch size—so batches must be large enough to average out user-level randomness yet recent enough to stay on-policy, and stratified across languages, task types, and workflows so no slice silently dominates. The freshness–batch-size tension is the production reincarnation of every bias–variance dial in this book.

18.4 Instrumentation

Training data here is not "text" but *event records*: who (anonymised), which checkpoint, what observed state, what action, what outcome, what latency, what reward, when. Several disciplines matter enough to name:

- **Checkpoint attribution.** Every action logged with the checkpoint that produced it—otherwise on-policyness cannot even be defined, let alone enforced.
- **No leakage.** Distinguish the state the policy *saw* from the telemetry logged around it; training must not condition or reward on information unavailable at action time.
- **Outcomes beyond the instant.** A poor edit may be accepted now and reverted tomorrow; instrumentation must track delayed consequences (persistence windows, follow-up sentiment) or the reward will systematically mistake tolerance for approval.
- **Latency decomposition.** Queue, prefill, decode, tool, network, render—so that a latency regression can be attributed to model, infrastructure, or client, and so the reward can penalise avoidable delay without punishing necessary deliberation.
- **Minimal collection.** Production interactions contain proprietary code and personal context; the design target is the *least* information whose reward quality clears the bar, with filtering, retention limits, and separation of identifiers from training records.

Credit assignment rides on this substrate. For long agentic episodes, trajectory-level reward is shared across the tokens of the actions (and, in the long-horizon variant, across the model's own intermediate *summaries* of its context—trained by RL so that compressions which preserved task-critical state are reinforced when the task succeeds: memory management itself becomes a learned action, the neatest modern example of Chapter 1's credit-assignment problem being solved by making the bottleneck part of the policy).

18.5 Evaluation and Gates

The reward proposes; evaluation disposes. A checkpoint ships only through a battery:

- **Benchmarks** (repeatable, deployment-independent): repository-level issue resolution, terminal-task suites, functional code correctness—necessary regression floors, insufficient proxies for interactive quality.
- **Product metrics**: acceptance and suggestion rates (read *jointly*—either alone is gameable by conservatism or spam), edit persistence, revert rate, dissatisfied follow-ups, task completion, latency distributions.
- **A/B tests**: the candidate against baseline on live traffic, $\widehat{\Delta} = \bar{Y}_{\text{treat}} - \bar{Y}_{\text{ctrl}}$, because training reward can overfit its own pipeline and only randomised deployment measures user value.

- **Regression gates**: deploy iff $M_j(\theta_{k+1}) - M_j(\theta_k) \geq -\epsilon_j$ for every guardrail metric j—a checkpoint may improve the optimised reward and still be blocked for latency, tool validity, or safety.
- **Slices**: per-language, per-task-type, per-workflow metrics, because aggregates hide minority regressions.
- **Drift monitors**: shifts in the *action distribution*—suggestion frequency, edit size, clarification rate, tool-failure rate—inspected even when reward improves, because reward hacking usually debuts as an unexplained behavioural shift, not a metric failure.

ELI5: The bakery with a suggestion box

A bakery changes its recipes a little every night based on what sold and what customers said. Freshness of feedback is everything: adjust tomorrow's bread by *this week's* reactions, not last year's. But the baker also knows customers buy sweeter bread even when it is worse for them, that one loud customer is not the neighbourhood, and that a recipe can sell well on day one and be regretted by day three. So: a suggestion box (instrumentation), taste tests before anything ships (gates), a rule never to change any recipe too much in one night (KL leash), and a suspicious eye on any recipe that suddenly sells weirdly well (drift). That is production RL, with flour.

A deployment decision is usefully compressed into a scorecard:

Area	Example metric	Question it answers
Reward	Mean online reward vs. last checkpoint	Did the optimised objective improve?
Quality	Benchmark pass rates	Did repeatable ability regress?
Product	Accept rate, persistence, completion	Did users get more value?
Friction	Dissatisfied follow-ups, reverts	Did interaction quality hold?
Systems	Latency percentiles, cost, errors	Is the product still responsive?
Behaviour	Tool failures, clarification rate	Has the action distribution drifted?
Slices	Per-language / per-task metrics	Is any subgroup paying for the average?

A checkpoint is an improvement only when the card is coherent: reward up, guardrails held, product metrics moving in the intended direction, no slice sacrificed. The reward may *propose* a checkpoint; the scorecard *disposes*.

18.6 The Standing Challenges

The production lens sharpens, rather than replaces, the field's classical open problems; we close the book by naming them.

Exploration vs. exploitation (Ch. 6) becomes acute when every exploratory action reaches a paying user. *Sample inefficiency* (Chs. 11–14) persists—human feedback is the most expensive data in machine learning. *Sparse and delayed reward* and *credit assignment over long horizons* (Chs. 5, 10) reappear as trajectory-level rewards over thousands of tokens. *Stability and convergence* under the deadly triad (Ch. 8) now runs continuously, unattended, against nonstationary users. *Safety*: specification gaming, constraint satisfaction, and the irreversibility of actions in the world—the CMDP and pessimism machinery (Chs. 15–16) are the current tools, not the final ones. *Generalisation*: policies overfit their training distribution, and distribution shift arrives daily by product change. *Reward design* (this chapter) has been promoted from footnote to the

central unsolved problem of applied RL: we can optimise almost anything; the frontier is knowing, and verifying, that we optimised the right thing.

The Abstract Viewpoint: The loop, finally, is the object

Classical RL analyses an algorithm inside a fixed MDP. Production RL dissolves the boundary: reward pipeline, telemetry, evaluation gates, deployment cadence, and user population are all inside the loop, and the correct unit of analysis is the *coupled system*—exactly the closed-loop perspective control theory has always urged. The monograph's arc thus ends where its first diagram began: an agent and an environment, arrows both ways—only now the environment contains us.

Key Takeaways

Production online RL closes the loop through deployment: rewards are proxies distilled from real user behaviour, updates are policy-gradient steps kept honest by freshness and clipping rather than heavy correction, instrumentation supplies attribution and delayed outcomes, and evaluation gates—benchmarks, product metrics, A/B tests, regression tolerances, slices, drift monitors—decide what ships. The recurring failure is proxy exploitation; the recurring cure is to treat the reward as an evolving specification inside a monitored, constrained, frequently redeployed system. It is generalised policy iteration one last time—evaluation and improvement, interleaved forever—with the world itself as the critic.

Appendix A

Notation

Symbol	Meaning
$\mathcal{S}, \mathcal{A}$	State space; action space
s, a, r	State, action, reward (realisations)
$P(s' \mid s, a)$	Transition kernel
$R(s, a), R(s, a, s')$	Expected reward function
γ	Discount factor, $\gamma \in [0, 1)$
$\pi(a \mid s)$	Policy (stochastic); $\mu_\theta(s)$ deterministic
$b(a \mid s)$, π_β	Behaviour policy (online; offline)
G_t	Return $\sum_{k \geq 0} \gamma^k r_{t+k+1}$
V^π, Q^π, A^π	State value, action value, advantage under π
V^*, Q^*, π^*	Optimal value functions and policy
$\mathcal{T}^\pi, \mathcal{T}$	Bellman expectation / optimality operators
δ_t	TD error $r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$
$G_t^{(n)}, G_t^\lambda$	n-step return; λ-return
$e_t(s)$	Eligibility trace
d^π	(Discounted) state visitation distribution
ρ_t	Importance ratio $\pi(a_t \mid s_t)/b(a_t \mid s_t)$
$J(\theta)$	Performance of π_θ
$F(\theta)$	Fisher information matrix
$\widehat{A}_t$	Advantage estimate (often GAE)
θ^-, ϕ^-	Target-network parameters
α	Step size; SAC temperature (context-dependent)
β	KL coefficient (RLHF); Lagrange/priority exponents
$D_{\mathrm{KL}}(p \,\|\, q)$	Kullback–Leibler divergence
$\mathcal{H}(\pi)$	Entropy of a policy
$r_\phi, V_\psi, \pi_{\mathrm{ref}}$	Reward model, value model, reference policy (RLHF)
$\|\cdot\|_\infty$	Supremum norm

Appendix B

Three Worked Computations

B.1 Value Iteration on the Gridworld, Two Sweeps by Hand

Take the 4×4 gridworld of Chapter 1 (-0.04 living cost, $+1$ goal, -1 pit, $\gamma = 0.99$, deterministic moves), and initialise $V_0 \equiv 0$ with terminal values fixed. Sweep 1: every non-terminal cell's best backup is $-0.04 + 0.99 \times 0 = -0.04$, except the two cells adjacent to the goal, whose best action reaches it: $-0.04 + 0.99 \times 1 = 0.95$; the neighbours of the pit avoid it and score -0.04. Sweep 2: cells two steps from the goal now back up $-0.04 + 0.99 \times 0.95 = 0.9005$; goal-adjacent cells are unchanged (their target values are terminal). The picture after two sweeps is already the truth in miniature: value radiates outward from the goal at one cell per sweep, discounted and taxed by the living cost as it spreads, and the greedy policy at each cell points up the gradient of V. After ~ 8 sweeps values change by less than 10^{-3} and the policy is optimal everywhere—including the instructive cell diagonally adjacent to the pit, where the safe detour beats the short path as soon as any action noise is introduced.

B.2 Q-Learning on a Two-State Chain, Four Updates

States $\{1, 2\}$; in each, actions L (stay, reward 0) and R (advance, reward 0 from state 1; reward $+1$ and terminate from state 2). $\gamma = 0.9$, $\alpha = 0.5$, Q initialised to zero, ε-greedy behaviour. Episode 1, suppose the agent stumbles R, R: update 1 (state 1, R): $Q(1, R) \leftarrow 0 + 0.5[0 + 0.9 \max(0, 0) - 0] = 0$; update 2 (state 2, R): $Q(2, R) \leftarrow 0 + 0.5[1 - 0] = 0.5$. Episode 2, same actions: update 3: $Q(1, R) \leftarrow 0 + 0.5[0 + 0.9 \times 0.5] = 0.225$; update 4: $Q(2, R) \leftarrow 0.5 + 0.5[1 - 0.5] = 0.75$. Two episodes, and the terminal reward has already propagated one step back—watching 0.225 appear in $Q(1, R)$ *is* watching bootstrapping work. The fixed point ($Q^*(2, R) = 1$, $Q^*(1, R) = 0.9$) is approached geometrically; the reader is warmly encouraged to run two more episodes by hand.

B.3 A PPO Ratio, Clipped

Suppose for one token the old policy had $\pi_{\text{old}}(a \mid s) = 0.20$, the new one $\pi_\theta(a \mid s) = 0.32$, so $\rho = 1.6$, with clip radius $\epsilon = 0.2$ and advantage $\widehat{A} = +0.8$. Unclipped surrogate: $1.6 \times 0.8 = 1.28$; clipped: $\text{clip}(1.6, 0.8, 1.2) \times 0.8 = 0.96$; the objective takes $\min(1.28, 0.96) = 0.96$, whose gradient in θ is *zero* through the clipped branch—the sample no longer pushes the probability up. Now flip the sign, $\widehat{A} = -0.8$: unclipped -1.28, clipped -0.96, and the min selects -1.28—the *unclipped* branch, still carrying gradient. The asymmetry is the design: a sample can always argue *against* a move that has overshot, but cannot keep arguing for more of it. Pessimism, one min at a time.

B.4 Importance Sampling: Watching the Variance Explode

One state, two actions. Target policy: $\pi(a_1) = 0.9$, $\pi(a_2) = 0.1$. Behaviour: uniform, $b(a_1) = b(a_2) = 0.5$. Rewards deterministic: $r(a_1) = 1$, $r(a_2) = 0$, so the truth is $J(\pi) = 0.9$. Per-step

ratios: $\rho(a_1) = 1.8$, $\rho(a_2) = 0.2$. One-step OIS is unbiased— $0.5(1.8\times1)+0.5(0.2\times0) = 0.9$—with modest variance: the estimator takes values 1.8 or 0.

Now chain H independent copies of this state and let the return be the product of per-step rewards (so only the all-a_1 path pays, with true value 0.9^H). The trajectory ratio on the paying path is 1.8^H, taken with probability 0.5^H: at $H = 20$ the estimator is 0 almost always and $1.8^{20} \approx 1.3 \times 10^5$ (times the reward 1) on the one path in a million that matters. The estimate is still unbiased—and useless: its relative standard deviation grows like $(1.8^2 \times 0.5)^{H/2} = 1.62^{H/2}$, i.e. exponentially. Weighted IS caps the estimate's *range* but not the fundamental starvation: the behaviour policy simply refuses to visit where the target policy lives. Every horizon-management device of Chapters 7, 12, and 18—per-decision weights, truncation, clipping, staying near-on-policy—is an answer to this five-line calculation.

Appendix C

Algorithm Selection Tables

C.1 By Problem Structure

Situation	Sensible first choices
Small finite MDP, model known	Policy iteration; value iteration
Small finite MDP, model unknown	Q-learning; SARSA (ε-greedy, decaying α)
Episodic, cheap simulator, awkward model	Monte Carlo control
Discrete actions, rich observations	DQN family (Double + Dueling + PER at minimum); PPO
Continuous control, cheap simulation	PPO (stability), SAC (efficiency)
Continuous control, expensive samples	SAC, TD3; MBPO/Dreamer if a model is learnable
Known rules / perfect simulator, planning affordable	AlphaZero-style MCTS + networks; MuZero if rules unavailable
Fixed dataset, no interaction	CQL, IQL, TD3+BC; evaluate with FQE/DR before deployment
Preference-based objective, LLM policy	SFT $\to$ RM (Bradley–Terry) $\to$ PPO with KL leash; DPO; GRPO for verifiable rewards
Deployed product, live feedback	Near-on-policy loop: fast redeployment, clipped updates, gates and drift monitors

C.2 The Classical Methods at a Glance

Method	Model	Update	Policy	Convergence
Dynamic programming	Required	Full backup	Implicit (greedy)	Geometric, exact
Monte Carlo	Free	Episode return	Implicit or explicit	LLN, $O(1/\sqrt{N})$
TD(0) / TD(λ)	Free	Bootstrapped	Evaluation	Robbins–Monro, a.s.
SARSA	Free	Bootstrapped, on-policy	ε-greedy	GLIE $\Rightarrow$ optimal
Q-learning	Free	Bootstrapped, off-policy	Greedy target	Watkins–Dayan, a.s.
REINFORCE	Free	MC policy gradient	Explicit	Local, high variance
Actor–critic	Free	TD-weighted gradient	Explicit	Two-time-scale

C.3 The Deep Methods at a Glance

Method	**Actions**	**Data**	**Key mechanism**	**Fails when**
DQN(+)	Discrete	Replay	Target nets, replay	Actions continuous/many
TRPO	Both	On-policy	KL trust region	Scale, implementation
PPO	Both	Near-on	Clipped ratio	Sample cost matters most
DDPG	Continuous	Replay	Deterministic gradient	Overestimation, tuning
TD3	Continuous	Replay	Min-twin critics	Sparse rewards
SAC	Continuous	Replay	Max-entropy objective	Discrete actions (vanilla)
MBPO	Continuous	Replay + model	Short branched rollouts	Model class mismatch
Dreamer	Both	Replay + latent model	Imagination training	Non-compressible obs.
MuZero	Discrete	Self-play/replay	Value-equivalent model + MCTS	No planning budget
CQL / IQL	Both	Offline	Pessimism	Data lacks coverage
PPO-RLHF	Tokens	Fresh rollouts	Learned reward + KL leash	RM over-optimised
GRPO	Tokens	Fresh rollouts	Group-relative baseline	Non-verifiable rewards

Appendix D

Problems

The problems below are graded within each part: the first ones check definitions, the later ones ask for small proofs or computations of the kind the text performs. Hints, where given, are in brackets.

Part I: Foundations

I.1 With $\gamma = 0.95$ and constant reward $r = 2$ forever, compute the return G_0 exactly. What discount factor makes $G_0 = 100$?

I.2 An MDP has rewards bounded by $|r| \le 3$ and $\gamma = 0.9$. Give the tightest bound on $|V^*(s)|$ implied by the text, and exhibit an MDP attaining it.

I.3 Prove that adding a constant c to every reward changes every V^π by exactly $c/(1-\gamma)$, and conclude that the optimal policy is unchanged. Where does this argument fail for episodic tasks compared across different episode lengths?

I.4 Show directly from the definitions that $\mathbb{E}_{a\sim\pi(\cdot|s)}[A^\pi(s,a)] = 0$ for every state s.

I.5 Verify the contraction property of $\mathcal{T}^\pi$ by hand on the two-state MDP of Chapter 2: compute $\|\mathcal{T}^\pi U - \mathcal{T}^\pi V\|_\infty$ for $U = (1, 0)$, $V = (0, 2)$ and compare with $\gamma \|U - V\|_\infty$.

I.6 In the recycling robot with the parameters of Chapter 3, compute the value of the never-recommended policy (wait, wait) and confirm it is dominated in both states.

I.7 Prove the monotonicity claim: $U \le V$ pointwise implies $\mathcal{T}U \le \mathcal{T}V$. [One line per operator; use that both are built from nonnegative combinations.]

I.8 Policy iteration is claimed to terminate finitely. Where exactly does the argument use finiteness of $\mathcal{S}$ and $\mathcal{A}$? Construct the counting bound.

Part II: Learning from Experience

II.1 First-visit MC returns for a state are i.i.d.; every-visit returns from the same episode are not. Exhibit a two-visit episode in a small chain whose two returns are perfectly correlated.

II.2 Redo the batch puzzle of Chapter 5 with the lone A-episode replaced by $A, 0, B, 1$: what do batch MC and batch TD now say about $V(A)$, and why do they agree or disagree?

II.3 Derive the incremental mean update $\widehat{V} \leftarrow \widehat{V} + \frac{1}{N}(G - \widehat{V})$ from the definition of the sample mean, and state what changes when $\frac{1}{N}$ is replaced by constant α in a nonstationary problem.

II.4 Show that the weights $(1-\lambda)\lambda^{n-1}$ of the λ-return sum to one, and compute the expected trace length (mean of the geometric distribution) as a function of λ. For $\gamma = 0.99$, what λ gives an effective horizon of 20 steps in $(\gamma\lambda)$?

II.5 Continue the Q-learning chain computation of Appendix B for episodes 3 and 4 and verify geometric approach to $Q^*(1, R) = 0.9$, $Q^*(2, R) = 1$.

II.6 In the cliff walk, explain in one paragraph why annealing $\varepsilon \to 0$ makes SARSA's learned policy converge to Q-learning's, and what happens to their *online* returns in the meantime.

II.7 Prove the maximisation-bias inequality $\mathbb{E}[\max_i X_i] \geq \max_i \mathbb{E}[X_i]$ for random variables $X_1, \ldots, X_k$, and give a two-action example where the gap equals 0.5.

II.8 An importance-sampling estimator uses behaviour b uniform over 4 actions and a deterministic target. Compute the one-step ratio, and the variance of the OIS estimator when the target action's reward is 1 and all others 0.

Part III: Scaling Up

III.1 In the linear TD worked example of Chapter 8, recompute the TD fixed point with features $\phi(1) = 1, \phi(2) = 1$ (a constant feature). Interpret the answer as an average and say which average.

III.2 Show that semi-gradient TD(0) with a table is exactly tabular TD(0). [Choose indicator features.]

III.3 Derive $\mathbb{E}_{a\sim\pi_\theta}[\nabla_\theta \log \pi_\theta(a \mid s)] = 0$ and use it to prove baseline unbiasedness.

III.4 For a softmax policy over preferences $h(s, a)$, compute $\nabla_h \log \pi(a \mid s)$ explicitly and interpret the resulting REINFORCE update as "raise the chosen, lower the rest in proportion to their probability."

III.5 For a one-dimensional Gaussian policy with fixed σ, show the score is $(a-\mu_\theta(s))/\sigma^2 \cdot \nabla_\theta \mu_\theta(s)$ and explain the $\sigma \to 0$ pathology of the score-function estimator.

III.6 Verify from the definitions that $\mathrm{GAE}(\gamma, 0) = \delta_t$ and that $\mathrm{GAE}(\gamma, 1)$ telescopes to $G_t - V(s_t)$.

III.7 Why must the critic run on the faster time scale? Describe the failure mode of the opposite ordering in one paragraph, using the coach–player metaphor or otherwise.

Parts IV–V: Deep and Deployed

IV.1 Recompute the three targets of the DQN worked example (Chapter 11) with $Q_\theta(s', \cdot) = (2.2, 2.1, 1.9)$ and unchanged Q_{θ^-}; explain why vanilla and Double DQN now nearly agree.

IV.2 In the PPO ratio example of Appendix B, find the exact ratio at which the clipped and unclipped branches cross for $\widehat{A} > 0$, and show no such finite crossing matters for $\widehat{A} < 0$ within $(1 - \epsilon, \infty)$.

IV.3 Show that the TD3 min-of-twins target is a lower bound on either critic's target, and explain why deliberate underestimation is safer than overestimation *specifically when an actor maximises the critic.*

IV.4 Derive the soft value identity $V(s) = \mathbb{E}_{a\sim\pi}[Q(s, a) - \alpha \log \pi(a \mid s)]$ from the definition of the entropy-augmented return.

IV.5 Verify that $\pi^*(y \mid x) \propto \pi_{\mathrm{ref}}(y \mid x) \exp(r(x, y)/\beta)$ maximises $\mathbb{E}_\pi[r] - \beta\, D_{\mathrm{KL}}(\pi \,\|\, \pi_{\mathrm{ref}})$ over distributions π. [Lagrange multiplier on normalisation; or complete the KL.]

IV.6 In the Bradley–Terry worked example, compute the gradient of the pairwise loss with respect to r_C when the observed preference is $C \succ A$, and check its sign against intuition.

IV.7 For the suggestion reward with threshold $\tau = 0.25$, compute the expected reward of a policy that shows suggestions whenever its (calibrated) acceptance probability exceeds 0.5, given that eligible moments have p uniform on $[0, 1]$. Would lowering its internal threshold to 0.25 help, and by how much?

IV.8 A production checkpoint improves mean reward by $+2\%$ but raises p95 latency by 30% and clarification rate by $3\times$. Write the regression-gate inequality that blocks it, and name the drift statistic that would have flagged it before the gate.

Appendix E

Glossary

Advantage
$A(s,a) = Q(s,a) - V(s)$: how much better an action is than the policy's average in that state.

Baseline
Any action-independent quantity subtracted from the policy-gradient weight; reduces variance without bias.

Behaviour policy
The policy generating the data, as opposed to the target policy being learned about.

Bellman equation
The recursive fixed-point equation satisfied by a value function; expectation form for a fixed policy, optimality form with a max.

Bootstrapping
Building a learning target out of one's own current estimates.

Bradley–Terry model
Logistic model of pairwise preference in terms of latent score differences; the reward-model likelihood in RLHF.

Contraction
A map that shrinks distances by a factor < 1; guarantees a unique fixed point and geometric convergence.

Credit assignment
Attributing a delayed outcome to the earlier decisions responsible for it.

Deadly triad
Function approximation + bootstrapping + off-policy data; jointly capable of divergence.

Discount factor
γ: weight of future reward; also the contraction modulus of the Bellman operators.

Distributional shift
Mismatch between the distribution a model was trained on and the one it is used on; the central hazard of offline RL.

Eligibility trace
Decaying per-state memory that broadcasts TD errors backward along the trajectory.

Entropy regularisation
Adding policy entropy to the objective to sustain exploration and smooth the landscape.

Experience replay
Storing transitions in a buffer and training on random minibatches from it.

Exploration/exploitation
The trade-off between acting to learn and acting to earn.

GAE
Generalised advantage estimation: exponentially weighted sum of TD errors, dialling bias against variance.

GPI
Generalised policy iteration: any interleaving of policy evaluation and policy improvement.

Importance sampling
Reweighting samples from one distribution to estimate expectations under another.

KL divergence
Asymmetric measure of distance between distributions; the currency of trust regions and RLHF leashes.

Markov property
The future is independent of the past given the present state.

Model-based / model-free
Whether the method learns or uses the transition/reward model, or works purely from sampled experience.

Off-policy / on-policy
Whether the target policy differs from the behaviour policy.

OPE
Off-policy evaluation: estimating a policy's performance from data another policy generated.

Pessimism
Systematically lower-bounding values where data is scarce; the organising principle of offline RL.

Regret
Cumulative shortfall against the best fixed action or policy in hindsight; the scorecard of exploration.

Return
Discounted sum of future rewards from a time step.

Reward hacking
Optimising the measured reward in ways that defeat the intended objective.

Reward model
A learned scalar function approximating human (or other) judgement, used as the RL objective in RLHF.

Reward shaping
Modifying the reward to aid learning; potential-based shaping is the invariance-preserving special case.

Rollout
A sampled trajectory; in LLM training, a sampled completion.

Semi-gradient
A gradient computed while treating a bootstrapped target as constant.

Target network
A slowly updated parameter copy used to compute bootstrap targets stably.

TD error
$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$; one-step surprise; unbiased advantage estimate.

Trajectory
The sequence of states, actions, and rewards of one interaction.

Trust region
A bound on how far the policy may move per update, measured in distribution space.

Value function
Expected return as a function of state (V) or state–action pair (Q).

Appendix F

A Reading Guide

The literature is vast; the following short path covers the monograph's span in primary sources.

Foundations
R. Bellman, *Dynamic Programming* (1957); R. Howard, *Dynamic Programming and Markov Processes* (1960); M. Puterman, *Markov Decision Processes* (1994) for the full measure-theoretic treatment. The indispensable modern text is Sutton and Barto, *Reinforcement Learning: An Introduction* (2nd ed., 2018), whose pedagogy Part II of this monograph gratefully shadows.

Classical algorithms
Sutton (1988) on TD learning; Watkins and Dayan (1992) on Q-learning; Rummery and Niranjan (1994) on SARSA; Barto, Sutton, Anderson (1983) on actor–critic; Williams (1992) on REINFORCE; Sutton, McAllester, Singh, Mansour (2000) and Kakade (2001) on (natural) policy gradients; Tsitsiklis and Van Roy (1997) on TD with function approximation.

Deep RL
Mnih et al. (2015), DQN; van Hasselt et al. (2016), Double DQN; Wang et al. (2016), dueling; Schaul et al. (2016), prioritised replay; Hessel et al. (2018), Rainbow; Mnih et al. (2016), A3C; Schulman et al. (2015, 2017), TRPO and PPO; Schulman et al. (2016), GAE; Lillicrap et al. (2016), DDPG; Fujimoto et al. (2018), TD3; Haarnoja et al. (2018), SAC; Levine (2018), control as inference.

Model-based and search
Sutton (1990), Dyna; Deisenroth and Rasmussen (2011), PILCO; Janner et al. (2019), MBPO; Ha and Schmidhuber (2018), world models; Hafner et al. (2020–2023), the Dreamer line; Silver et al. (2016–2018), AlphaGo/AlphaZero; Schrittwieser et al. (2020), MuZero.

Scale and meta
Horgan et al. (2018), Ape-X; Espeholt et al. (2018), IMPALA; Finn et al. (2017), MAML; Duan et al. / Wang et al. (2016), RL^2; Rakelly et al. (2019), PEARL.

Offline RL and OPE
Fujimoto et al. (2019), BCQ; Kumar et al. (2020), CQL; Kostrikov et al. (2021), IQL; Chen et al. (2021), Decision Transformer; Levine et al. (2020), the offline RL survey; Precup, Sutton, Singh (2000), per-decision IS; Jiang and Li (2016) and Thomas and Brunskill (2016), doubly robust OPE.

RLHF and LLM policy optimisation
Christiano et al. (2017), deep RL from human preferences; Stiennon et al. (2020), summarisation from feedback; Ouyang et al. (2022), InstructGPT; Bai et al. (2022), Constitutional AI and RLAIF; Rafailov et al. (2023), DPO; Shao et al. (2024), GRPO; Gao, Schulman, Hilton (2023) on reward-model over-optimisation; Amodei et al. (2016), *Concrete Problems in AI Safety*, for the reward-hacking canon.

Production online RL
The public engineering literature on live policy-gradient training of coding assistants (online RL for autocomplete acceptance; real-time RL for agentic coding models; RL-trained self-summarisation for long horizons) is the best current documentation of deployment-loop practice, and Chapter 18 distils its published lessons.

The Bellman equation is a short sentence.
The rest is the world learning to hear it.

Manifold Mathematics Monograph Series

© 2026 Manifold Mathematics