强化学习自学教程

从表格方法到大模型对齐

每个概念附直觉解释 · 公式推导 · 伪代码 · 实例

整理时间：2026 年 4 月 | 覆盖范围：经典算法 + 2020–2026 前沿论文

- 绿色框 = 直觉解释（用类比和大白话讲清楚为什么）
- 蓝色框 = 公式推导（严格数学，但每步都有注释）
- 黄色框 = 具体例子（帮你建立感性认知）
- 红色框 = 常见陷阱和注意事项
- 灰色框 = 论文索引（附链接，方便进一步阅读）
- 算法框 = 伪代码（可以直接对照实现）

你不需要从头读到尾——每一章都是自包含的，但建议至少先读完第 1 章建立全局观。

Contents

第 1 章 RL 全局观：一张地图

想象你在一个陌生城市找路去餐厅。你没有地图（没有监督信号），也没有人告诉你每一步该怎么走（不是监督学习）。你只能一边走一边试，走对了会闻到食物香味（奖励），走错了会迷路（惩罚）。通过不断试错，你最终找到了最优路线。
这就是强化学习：一个智能体（Agent）在环境（Environment）中通过试错（Trial-and-Error）学习最优行为（Policy）。

1.1 MDP：所有 RL 的数学起点

定义 **1.1** (马尔可夫决策过程（MDP）)**.** $MDP = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$，其中：

- $\mathcal{S}$：状态空间——世界所有可能的状态
- $\mathcal{A}$：动作空间——*Agent* 可以采取的所有动作
- $P(s'|s,a)$：转移函数——在状态 s 执行动作 a 后到达 s' 的概率
- $R(s,a)$：奖励函数——执行动作后获得的即时反馈
- $\gamma \in [0,1)$：折扣因子——未来奖励的 ” 缩水率 ”

两个原因：(1) 今天的 1 块钱比明年的 1 块钱更值钱（时间偏好）；(2) 数学上需要让无限求和收敛。
$\gamma = 0.99$ 表示 100 步后的奖励只值当前的 $0.99^{100} \approx 0.37$，即未来很重要。
$\gamma = 0.9$ 表示 100 步后的奖励只值 $0.9^{100} \approx 0.00003$，即只关注近期。

■ 核心量的定义

策略（Policy）：$\pi(a|s)$ 表示在状态 s 下选择动作 a 的概率。
状态价值函数（Value Function）：从状态 s 开始，按策略 π 行动的期望累积回报：

$$V^\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \,\middle|\, s_0 = s \right]$$

动作价值函数（Q-Function）：在状态 s 执行动作 a，之后按 π 行动的期望回报：

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^\pi(s')$$

优势函数（Advantage）：动作 a 比平均水平好多少：

$$A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$$

直觉：$A > 0$ 说明这个动作比平均好，$A < 0$ 说明比平均差。

■ Bellman 方程：RL 的牛顿定律

Bellman 方程是所有 RL 算法的理论基石——它建立了当前状态价值和未来状态价值之间的递归关系：

Bellman 期望方程（评估策略 π 有多好）：

$$V^\pi(s) = \sum_a \pi(a|s) \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^\pi(s') \right]$$

Bellman 最优方程（找最好的策略）：

$$V^*(s) = \max_a \left[R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \right]$$

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

考虑一个 2×2 网格，Agent 在左上角 (1,1)，目标在右下角 (2,2)，到达目标奖励 +1，每步奖励 -0.04，$\gamma = 0.9$。
对目标右边的格子 (2,1)，向右走到达目标：

$$Q^*((2,1), \text{右}) = R + \gamma V^*(\text{目标}) = -0.04 + 0.9 \times 1 = 0.86$$

这就是 Bellman 方程的作用：把”长期价值”分解为”即时奖励 + 折扣的未来价值”。

1.2 演进路线图

阶段	代表	核心思想	瓶颈
1. 表格	Q-Learning	查表：$Q[s][a]$ 存每对 (状态, 动作) 的价值	维度灾难
2. 深度	DQN	神经网络替代表格，泛化到像素输入	连续动作
3. AC	PPO, SAC	Actor 输出动作 + Critic 打分	样本效率
4. 离线	CQL, DT	只用历史数据，不与环境交互	OOD
5. 世界模型	Dreamer	在”想象”中规划，减少真实交互	模型误差
6. 对齐	DPO, GRPO	训练 LLM 学人类偏好/推理能力	奖励建模
7. Agent	**Agentic RL**	RL 训练 LLM Agent 与环境多轮交互	信用分配

- 1→2：从”死记硬背”到”举一反三”（神经网络泛化）
- 2→3：从”只能选有限选项”到”可以精确转方向盘”（连续控制）
- 3→4：从”必须亲自试”到”看别人的数据就能学”（离线学习）
- 4→5：从”只看数据”到”在脑中想象后果再决策”（世界模型）
- 5→6：从”游戏/机器人”到”教 AI 说人话”（LLM 对齐）
- 6→7：从”一问一答”到”多步操作完成复杂任务”（Agent 训练）

第 2 章 DQN：神经网络遇上 Q-Learning

2.1 从表格到网络：为什么 Q-table 不够用

围棋的状态数约为 10^{170}——比宇宙中的原子数还多。即使是简单的 Atari 游戏，一帧 210×160 的 RGB 画面就有 $256^{210\times160\times3} \approx 10^{242,000}$ 种可能。
Q-table 需要为每个状态-动作对存一个值——这在连续或高维状态空间中完全不可能。
解决方案：用神经网络 $Q(s,a;\theta)$ 来近似 Q 函数。输入状态 s，输出所有动作的 Q 值。

2.2 DQN 的两个关键技巧

直接用神经网络替代 Q-table 会训练不稳定。DQN 引入两个技巧解决：

■ 技巧 1：经验回放 (Experience Replay)

将交互经验 (s,a,r,s') 存入缓冲区 $\mathcal{D}$，训练时随机采样一个 mini-batch。
为什么需要？相邻时间步的数据高度相关（你在游戏中连续几帧几乎一样）。直接用这些数据训练会导致过拟合。随机采样打破了时间相关性。

■ 技巧 2：目标网络 (Target Network)

维护两套参数：当前网络 θ 和目标网络 θ^-。θ^- 每隔 C 步才从 θ 复制一次。
为什么需要？如果用同一个网络既计算目标又更新参数，就像”追逐一个不断移动的目标”——永远追不上。目标网络提供一个相对稳定的”标靶”。

■ DQN 损失函数

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\underbrace{\left(\underbrace{r + \gamma \max_{a'} Q(s',a';\theta^-)}_{\text{TD 目标（用目标网络）}} - \underbrace{Q(s,a;\theta)}_{\text{当前估计}}\right)^2}_{\text{最小化二者之差}}\right]$$

Algorithm 1 DQN 算法

输入： 环境 env，回放缓冲区容量 N，目标网络更新频率 C

初始化 Q 网络 θ，目标网络 $\theta^- \leftarrow \theta$，回放缓冲区 $\mathcal{D}$
for 每个 episode **do**
  获取初始状态 s_0
  for 每一步 t **do**
    以 ϵ-贪心选择动作：$a_t = \begin{cases} \text{随机动作} & \text{概率}\epsilon \\ \arg\max_a Q(s_t,a;\theta) & \text{概率}1-\epsilon \end{cases}$
    执行 a_t，观测 (r_t, s_{t+1})
    存入缓冲区 $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t,a_t,r_t,s_{t+1})\}$
    从 $\mathcal{D}$ 采样 mini-batch $\{(s_j,a_j,r_j,s'_j)\}$
    计算目标 $y_j = r_j + \gamma \max_{a'} Q(s'_j,a';\theta^-)$
    梯度下降最小化 $(y_j - Q(s_j,a_j;\theta))^2$
    每 C 步更新：$\theta^- \leftarrow \theta$
  end for
end for

2.3 DQN 的改进：从 DQN 到 Rainbow

■ Double DQN：解决 Q 值过高估计

原始 DQN 用同一个网络选动作和评估，会系统性高估 Q 值。Double DQN 解耦这两步：

$$y = r + \gamma Q\left(s', \underbrace{\arg\max_{a'} Q(s', a'; \theta)}_{\text{当前网络选动作}}; \underbrace{\theta^-}_{\text{目标网络评估}}\right)$$

直觉：让”出主意的”和”打分的”是不同的人，避免自吹自擂。

■ Dueling DQN：分离状态价值和动作优势

将 Q 网络分成两支：

$$Q(s, a; \theta) = V(s; \theta_v) + A(s, a; \theta_a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a'; \theta_a)$$

直觉：有些状态本身就很好/很差（和你做什么动作无关），分开建模更高效。

○ Rainbow 及后续

Rainbow (2018)：整合 6 种改进——Double、Dueling、优先回放、多步回报、分布式 RL、NoisyNet。
Beyond The Rainbow (BTR) (ICML 2025)：在 Rainbow 上再叠 6 种新改进，单桌面 GPU 12 小时达 Atari-60 IQM=7.6。
CHAIN (NeurIPS 2024)：发现 Q 值估计和策略更新的”链式波动”，提出通用即插即用抑制模块。

DQN 输出所有动作的 Q 值，然后 $\arg\max$。当动作是连续的（如机器人关节角度 $\in \mathbb{R}$），这个 $\arg\max$ 运算无法进行——你不能对无穷多个实数取最大值。
这就是为什么需要下一章的 Actor-Critic 架构。

第 3 章 Actor-Critic：策略与价值的协奏

3.1 策略梯度：让网络直接输出动作

既然在连续空间中”选最优动作”很难，不如让网络直接输出动作。Actor 网络 $\pi_\theta(a|s)$ 就是一个参数化的策略——给定状态，直接输出动作（或动作的概率分布）。
但问题来了：怎么告诉 Actor ”你输出的动作好不好”？——需要一个 Critic 来打分。

■ 策略梯度定理（Policy Gradient Theorem）

目标：最大化期望回报 $J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$

梯度：

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \cdot A^{\pi_\theta}(s_t, a_t)\right]$$

逐项解读：

- $\nabla_\theta \log \pi_\theta(a_t|s_t)$：告诉参数 θ ”沿着什么方向调整会让动作 a_t 更可能被选中”
- $A(s_t, a_t)$：告诉你”这个动作比平均好多少”（优势函数）
- 两者相乘的含义：如果动作好（$A > 0$），就增加它的概率；如果动作差（$A < 0$），就降低它的概率

这就是 **REINFORCE** 算法的核心——用优势函数加权的策略梯度。

3.2 GAE：精确估计优势函数

■ GAE: Generalized Advantage Estimation

优势函数 $A(s_t, a_t)$ 怎么算？用 TD 误差的指数加权平均：
TD 误差（一步估计）：

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

直觉：$\delta_t > 0$ 意味着”实际比预期好”。
GAE（多步加权平均）：

$$A_t^{\text{GAE}} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} = \delta_t + \gamma\lambda\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \cdots$$

λ 的作用——偏差-方差平衡旋钮：

- $\lambda = 0$：$A_t = \delta_t$（纯 TD，低方差但高偏差——只看一步）
- $\lambda = 1$：$A_t = \sum_l \gamma^l \delta_{t+l}$（蒙特卡洛，低偏差但高方差——看完整轨迹）
- 实践中 $\lambda = 0.95$ 通常效果最好

3.3 PPO：Actor-Critic 的工业标准

策略梯度有一个致命弱点：更新步长很难控制。步子太小学得慢，步子太大直接崩溃（新策略完全偏离旧策略，采集的数据全部失效）。
TRPO 的做法是限制 KL 散度（用二阶优化，实现复杂）。PPO 的天才之处在于用一个简单的 clip 操作就近似达到了 TRPO 的效果。

■ PPO-Clip 目标函数（逐行解读）

$$\mathcal{L}^{\text{PPO}}(\theta) = \mathbb{E}_t\left[\min\left(\underbrace{r_t(\theta)A_t}_{\text{普通策略梯度}}, \underbrace{\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t}_{\text{裁剪版}}\right)\right]$$

其中 $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ 是新旧策略的概率比。
四种情况分析（$\epsilon = 0.2$）：

1. $A_t > 0, r_t > 1.2$：好动作，但更新太大 → clip 到 1.2，阻止过度更新
2. $A_t > 0, r_t < 1.2$：好动作，更新幅度合理 → 正常梯度上升
3. $A_t < 0, r_t < 0.8$：差动作，更新太大 → clip 到 0.8，阻止过度惩罚
4. $A_t < 0, r_t > 0.8$：差动作，更新幅度合理 → 正常梯度下降

一句话：clip 像安全带——让你学习，但不让你翻车。

Algorithm 2 PPO 算法

输入： 初始策略 π_{θ_0}，价值网络 V_ϕ，裁剪参数 $\epsilon = 0.2$，GAE 参数 $\lambda = 0.95$

for 每轮迭代 $k = 0, 1, 2, \ldots$ **do**
  用当前策略 π_{θ_k} 采集一批轨迹 $\{(s_t, a_t, r_t)\}$
  计算 GAE 优势估计 $\hat{A}_t$
  计算 return-to-go $\hat{R}_t = \hat{A}_t + V_\phi(s_t)$
  for 每个 epoch $e = 1, \ldots, K$（通常 $K = 3 \sim 10$） **do**
    for 每个 mini-batch **do**
      计算概率比 $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}$
      策略损失 $L_\pi = -\frac{1}{|B|}\sum_t \min(r_t A_t, \text{clip}(r_t, 1 \pm \epsilon)A_t)$
      价值损失 $L_V = \frac{1}{|B|}\sum_t (V_\phi(s_t) - \hat{R}_t)^2$
      总损失 $L = L_\pi + c_1 L_V - c_2 \mathcal{H}(\pi_\theta)$（熵正则鼓励探索）
      梯度下降更新 θ, ϕ
    end for
  end for
end for

3.4 TRPO 与 SAC

■ TRPO: Trust Region Policy Optimization

PPO 的理论前身。用 KL 散度约束代替 clip：

$$\max_\theta \ \mathbb{E}_t[r_t(\theta)A_t] \quad \text{s.t.} \quad \mathbb{E}_t[D_{\text{KL}}(\pi_{\theta_{\text{old}}} \| \pi_\theta)] \le \delta$$

需要二阶优化（Fisher 信息矩阵），实现复杂但理论保证更强。PPO 用一阶近似（clip）替代了这个约束。

■ SAC: Soft Actor-Critic（最大熵框架）

SAC 的核心思想：不仅要最大化奖励，还要最大化策略的熵（多样性）。

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_\pi \left[r(s_t, a_t) + \alpha \underbrace{\mathcal{H}(\pi(\cdot|s_t))}_{\text{策略熵}} \right]$$

为什么要最大化熵？

- 鼓励探索：高熵策略会尝试更多不同的动作
- 鲁棒性：不把所有鸡蛋放在一个篮子里
- 多模态：同一任务可能有多种好的解法，SAC 会保留所有好策略

Soft Bellman 方程：

$$Q(s,a) = r + \gamma \mathbb{E}_{s'}\left[V(s')\right], \quad V(s) = \mathbb{E}_{a\sim\pi}\left[Q(s,a) - \alpha \log \pi(a|s)\right]$$

α 是温度系数，SAC 可以自动调节：

$$\alpha^* = \arg\min_{\alpha} \mathbb{E}_{a\sim\pi^*}\left[-\alpha \log \pi^*(a|s) - \alpha\bar{\mathcal{H}}\right]$$

其中 $\bar{\mathcal{H}}$ 是目标熵（通常设为 $-\dim(\mathcal{A})$）。

	PPO	SAC	TD3	TRPO
On/Off-Policy	On	Off	Off	On
连续/离散	都行	连续	连续	都行
探索机制	熵正则	最大熵	高斯噪声	无特殊
稳定性				
最终性能				
实现复杂度	低	中	低	高

第 4 章 离线 RL：只看数据不试错

在医院里，你不能让 AI ”试错” 来学习用药方案——试错的代价是人命。自动驾驶也不能让 AI 随便撞车来学开车。
但医院有大量历史病历，路上有大量行车记录仪。能不能只用这些” 死数据”（从不与环境交互）来训练出好的决策策略？
这就是离线 **RL**（Offline RL，也叫 Batch RL）要回答的问题。

假设历史数据是由” 保守司机” 收集的（从不超车），那数据中根本没有” 超车” 这个动作。如果离线 RL 算法估计” 超车的 Q 值很高”——它怎么知道的？它根本没见过超车！
这就是 **OOD**（**Out-of-Distribution**）动作高估问题：对没见过的动作，Q 网络会随机输出一个值，而 max 操作会系统性地选中被高估的 OOD 动作。

$$\text{普通 Q-Learning：}\quad \pi(s) = \arg\max_a Q(s,a) \quad \leftarrow \text{如果} Q(s, a_{\text{OOD}}) \text{ 被高估，就会选错}$$

4.1 CQL：保守地估计 Q 值

■ CQL 核心思想：宁可低估，不可高估

在标准 Bellman 误差上加一项保守正则化——主动压低那些” 没在数据中见过” 的动作的 Q 值：

$$\mathcal{L}^{\text{CQL}}(\theta) = \underbrace{\alpha\left(\mathbb{E}_{s\sim\mathcal{D},\, a\sim\mu(a|s)}[Q_\theta(s,a)] - \mathbb{E}_{(s,a)\sim\mathcal{D}}[Q_\theta(s,a)]\right)}_{\text{保守项：压低 OOD 动作，抬高数据中的动作}} + \underbrace{\mathcal{L}^{\text{TD}}(\theta)}_{\text{标准 Bellman 误差}}$$

其中 $\mu(a|s)$ 是一个广泛分布（如均匀分布），$\mathcal{D}$ 是离线数据集。
直觉：第一项的含义是” 让 Q 值在所有可能动作上的平均值尽量低，但在数据中实际出现的动作上尽量高”——这样 OOD 动作的 Q 值自然就被压低了。
理论保证：CQL 学到的 Q 函数是 Q^π 的下界——你可以信任它，因为真实值只会更好。

○ CQL 系列演进

CQL (NeurIPS 2020)：奠基之作。保守 Q 函数 = Q 值下界。
Cal-QL (NeurIPS 2023)：CQL 太保守了！在线微调时 Q 值从下界开始回升太慢（"遗忘"现象）。Cal-QL 用校准的下界——一行代码改动。
SCQ (2024)：策略性保守——只在远离数据分布的 OOD 区域压低 Q 值，利用神经网络的插值能力让"接近数据"的区域保持正常估计。
CFCQL (NeurIPS 2023)：将 CQL 扩展到多智能体离线 RL——每个 Agent 用反事实正则化独立保守。

4.2 IQL：绝不看 OOD 动作一眼

■ IQL：比 CQL 更极端的保守策略

CQL 是"压低 OOD 的 Q 值"。IQL 更极端：根本不在 **OOD** 动作上求 **Q** 值。
关键洞察：如果我们只用数据集里出现过的 (s, a) 对来训练 Q 函数，并用期望分位数回归（expectile regression）来隐式地学习"数据里最好的动作有多好"，就完全避免了 OOD 问题。
Value Function 训练（上分位数逼近）：

$$\mathcal{L}_V(\psi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[L_2^\tau\big(Q_{\hat{\theta}}(s,a) - V_\psi(s)\big)\right]$$

其中 $L_2^\tau(u) = |\tau - \mathbb{1}(u < 0)| \cdot u^2$
当 $\tau > 0.5$ 时，$V(s)$ 会偏向 Q 值的上分位数——即"数据里最好的动作对应的价值"。
Q Function 训练（标准 TD，只用数据中的 (s, a)）：

$$\mathcal{L}_Q(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\big(r + \gamma V_\psi(s') - Q_\theta(s,a)\big)^2\right]$$

策略提取（Advantage-Weighted Regression）：

$$\pi(a|s) \propto \pi_\beta(a|s) \cdot \exp\big(\beta \cdot (Q(s,a) - V(s))\big)$$

其中 π_β 是行为策略（数据收集策略）。直觉：$Q(s,a) - V(s) > 0$ 的动作被放大，< 0 的被缩小。

4.3 Decision Transformer：RL 变成序列预测

传统 RL 通过 Bellman 方程做"动态规划"。Decision Transformer 说：何必这么复杂？我有大量轨迹数据，直接把它当作序列预测问题——就像 GPT 预测下一个 token 一样预测下一个动作！
关键 trick：把"你希望得到多少回报"也作为输入条件——想要高分就给高 return-to-go，模型自然会生成高分对应的动作序列。

■ Decision Transformer 的序列建模

将轨迹表示为 token 序列：

$$\tau = (\hat{R}_1, s_1, a_1, \hat{R}_2, s_2, a_2, \ldots, \hat{R}_T, s_T, a_T)$$

其中 $\hat{R}_t = \sum_{t'=t}^{T} r_{t'}$ 是 **return-to-go**（从第 t 步开始的累积剩余回报）。
训练：标准的因果 Transformer，用交叉熵损失预测动作：

$$\mathcal{L}(\theta) = -\sum_t \log \pi_\theta(a_t \,|\, \hat{R}_{\le t}, s_{\le t}, a_{<t})$$

推理：设定 $\hat{R}_1 = R_{\text{target}}$（你期望的总回报），然后自回归生成动作。

○ **Decision Transformer 系列**

Decision Transformer (NeurIPS 2021)：奠基之作。
Q-Transformer (ICML 2024, DeepMind)：将 Q-learning 融入 Transformer，支持大规模多任务策略。
Decision Mamba (NeurIPS 2024)：用 Mamba（状态空间模型）替代 Transformer，$O(n)$ 复杂度 vs Transformer 的 $O(n^2)$。
Hierarchical DT (ICML 2024)：分层 RL 框架解决 DT 无法拼接次优轨迹的问题。

4.4 扩散模型 + 离线 RL

扩散模型（Diffusion）擅长生成高质量样本。在离线 RL 中，我们可以把整条轨迹视为一个" 样本"——训练扩散模型学习" 好轨迹长什么样"，规划时就是从噪声去噪出一条好轨迹。

■ **Diffuser**：轨迹级扩散规划

前向过程：给轨迹 τ^0 逐步加噪声，变成纯噪声 $\tau^K \sim \mathcal{N}(0, I)$
反向过程（去噪 = 规划）：

$$\tau^{k-1} = \frac{1}{\sqrt{\alpha_k}}\left(\tau^k - \frac{1-\alpha_k}{\sqrt{1-\bar{\alpha}_k}}\epsilon_\theta(\tau^k, k)\right) + \sigma_k \mathbf{z}$$

其中 ϵ_θ 是去噪网络。
加入奖励引导（Classifier Guidance）：

$$\tilde{\nabla} \log p(\tau|\text{高奖励}) = \nabla \log p(\tau) + \nabla_\tau J(\tau)$$

直觉：在去噪过程中同时推向高奖励方向。

○ **扩散 + 离线 RL 系列**

Diffuser (ICML 2022)：奠基之作，950+ 引用。
Diffusion-QL (2023)：策略 = 以状态为条件的条件扩散模型。
DWM (2024)：扩散世界模型——单次前向预测多步未来（非递归），D4RL 提升 44%。
CoD (2024)：持续扩散模型——90 个序列任务的持续离线 RL。
DAWM (2025)：扩散动作世界模型——生成状态-奖励轨迹 + 逆动力学模型。

4.5 从离线到在线（Offline-to-Online）

离线 RL 用历史数据学到一个不错的策略后，如果有机会和环境少量交互，能不能继续提升？答案是能——但需要小心。直接做在线 RL 会导致离线阶段学到的保守 Q 值" 遗忘"，性能先暴跌再慢慢回升。

○ **Offline-to-Online 系列**

WSRL (ICLR 2025)：少量热身 rollout 启动在线微调，学习更快、渐近性能更高。
OPT (ICML 2025)：在线预训练阶段训练全新价值函数，D4RL 平均提升 30%。
三阶段理论 (2025)：揭示 O2O 存在三个不同阶段，每个阶段需要不同的稳定性-可塑性特性。
NeurIPS 2025 Oral：严格分类体系 + Unifloral 统一框架 + TD3-AWR/MoBRAC。

第 5 章 世界模型：在想象中学习

人类棋手不是每步都要真的下出来看结果——他们在脑中推演：”如果我走这里，对手可能走那里，然后我再走这里……”。
世界模型就是 AI 的”脑中推演器”：学习环境的动力学模型，在”想象”中模拟未来，在想象中训练策略——大幅减少与真实环境的交互次数。

■ Dreamer 世界模型（RSSM: Recurrent State Space Model）

Dreamer 在潜空间中建模世界：
编码器（观测 → 潜状态）：$z_t \sim q_\phi(z_t \mid h_t, o_t)$
转移模型（预测下一个潜状态）：$\hat{z}_{t+1} \sim p_\phi(z_{t+1} \mid h_t, a_t)$
奖励预测：$\hat{r}_t \sim p_\phi(r_t \mid h_t, z_t)$
循环模型（确定性状态传递）：$h_{t+1} = f_\phi(h_t, z_t, a_t)$
在想象中训练 **Actor-Critic**：从真实数据的潜状态出发，”想象”H 步轨迹：

$$\mathcal{L}_{\text{actor}} = -\mathbb{E}_{\text{imagined}}\left[\sum_{t=1}^{H} V_\psi(h_t, z_t)\right]$$

关键优势：Actor 的梯度通过世界模型反向传播，不需要真实环境交互。

○ 世界模型系列

DreamerV3 (Nature 2025)：通用算法，单一超参数配置跨越 150+ 任务。首次在 Minecraft 中从零收集钻石。
IRIS (ICLR 2023, top 5%)：离散自编码器 + 自回归 Transformer 世界模型，Atari 100k 超越人类。
Delta-IRIS (ICML 2024)：编码时间步间的随机差分而非全帧，Crafter SOTA，训练快 10 倍。
DyMoDreamer (2025)：同时超越 IRIS 系列和 DreamerV3（Crafter 10.3 vs 9.4）。
SafeDreamer (2023)：拉格朗日方法 + Dreamer = 安全 RL，几乎零成本满足安全约束。
SIMA 2 (DeepMind 2025)：基于 Gemini 的通用具身 Agent，3D 世界中自主学习新技能。

第 6 章 大模型对齐：从 RLHF 到 GRPO

GPT-4 经过海量文本预训练后，已经”知道”很多东西——但它不知道人类偏好什么。它可能会生成一个事实正确但有害的回答，或者一个安全但无用的回答。
RLHF 的核心思想：让人类对 LLM 的多个回答进行排序（”这个好，那个差”），训练一个奖励模型来模拟人类判断，再用 RL（PPO）让 LLM 学会生成人类偏好的回答。

6.1 RLHF 三阶段流水线

■ RLHF 完整流程

阶段 **1**（**SFT**）：在高质量人类演示数据上监督微调：π_{SFT}
阶段 **2**（奖励模型）：收集人类偏好数据 $(x, y_w \succ y_l)$，训练奖励模型 r_ϕ：

$$\mathcal{L}_{\text{RM}}(\phi) = -\mathbb{E}_{(x,y_w,y_l)}\left[\log \sigma\big(r_\phi(x,y_w) - r_\phi(x,y_l)\big)\right]$$

直觉：让好回答的奖励分数 > 差回答，用 sigmoid 二分类损失。
阶段 **3**（**PPO** 对齐）：以 r_ϕ 为奖励，用 PPO 优化 LLM 策略 π_θ：

$$\max_\theta \ \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)}\left[r_\phi(x,y) - \beta\, D_{\text{KL}}(\pi_\theta(y|x)\|\pi_{\text{ref}}(y|x))\right]$$

KL 约束防止 LLM 偏离太远变成”只讨好奖励模型的无意义文本”（reward hacking）。

1. 奖励模型可能不准，导致 reward hacking
2. PPO 需要同时维护 4 个模型（Actor, Critic, Reward Model, Reference），显存巨大
3. PPO 超参数敏感，训练不稳定

这些痛点催生了 DPO——能不能跳过奖励模型，直接用偏好数据优化 LLM?

6.2 DPO：绕过奖励模型

■ DPO 的推导（三步）

Step 1：RLHF 的最优策略有闭式解：

$$\pi^*(y|x) = \frac{1}{Z(x)}\pi_{\text{ref}}(y|x)\exp\left(\frac{r(x,y)}{\beta}\right)$$

Step 2：反解出奖励函数：

$$r(x,y) = \beta\log\frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} + \beta\log Z(x)$$

Step 3：代入奖励模型的 Bradley-Terry 损失，$Z(x)$ 对消：

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(x,y_w,y_l)}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)}\right)\right]$$

一句话：让 LLM 在好回答上比参考模型更自信，在差回答上比参考模型更不自信。

○ DPO 家族

DPO (NeurIPS 2023)：奠基之作。
IPO (2024)：更强理论保证，但实际常弱于 DPO。**KTO** (2024)：只需点赞/踩（不需配对偏好）。
ORPO (2024)：SFT + 对齐一步完成。
SimPO (NeurIPS 2024)：$r(x,y) = \frac{1}{|y|}\log\pi_\theta(y|x)$，无需参考模型，AlpacaEval2 超 DPO 6.4 分。
RainbowPO (ICLR 2025)：统一分析，证明 DPO 变体共享边际/主场优势项。
GRPO=DPO (2025)：证明 GRPO 本质上是通过对比对构造的 DPO。

6.3 GRPO：DeepSeek-R1 的核心算法

■ GRPO: Group Relative Policy Optimization

GRPO 是训练推理型 LLM（如 DeepSeek-R1）的主流算法。
核心思想：对每个 prompt x，采样 G 个回答，用组内统计量替代 Critic：
Step 1：采样一组回答 $\{y_1, \ldots, y_G\} \sim \pi_{\theta_{\text{old}}}(\cdot|x)$
Step 2：计算每个回答的奖励 $r_i = r(x, y_i)$
Step 3：组内归一化得优势：

$$A_i = \frac{r_i - \text{mean}(\{r_j\}_{j=1}^G)}{\text{std}(\{r_j\}_{j=1}^G)}$$

Step 4：PPO-style 更新：

$$\mathcal{L}_{\text{GRPO}}(\theta) = -\frac{1}{G}\sum_{i=1}^{G} \min\big(r_i(\theta)A_i,\ \text{clip}(r_i(\theta), 1 \pm \epsilon)A_i\big) + \beta D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}})$$

关键优势：无需 Critic 网络（组内统计量替代），显存减半！

■ DAPO：GRPO 的四项关键改进

DAPO（字节跳动/清华 AIR，2025）：

1. **Clip-Higher**：对好动作（$A > 0$）用更大的上裁剪界（$1 + 2\epsilon$ 而非 $1 + \epsilon$），防止熵崩塌——确保策略保持探索性
2. **Dynamic Sampling**：过滤全对/全错的 prompt（组内方差 =0，信息量为零）
3. **Token-level Loss**：Token 级梯度而非 Sequence 级——长短回答获得相同的梯度权重
4. **Overlong Reward Shaping**：对超长回答施加渐进惩罚

结果：AIME 2024 达 50 分，超 DeepSeek-R1-Zero (47)，训练步数减半。完全开源。

6.4 RLVR：可验证奖励的强化学习

RLHF 依赖人类标注偏好数据——昂贵且有噪声。RLVR 说：对于数学/代码这类有客观答案的任务，直接用程序验证器作为奖励！答对了 +1，答错了 0。
这就是 DeepSeek-R1 成功的关键——它不用人类打分，而是用数学验证器和代码测试作为奖励信号，用 GRPO 训练。结果涌现出了自我反思、验证、纠错等推理行为。

○ RLVR 系列

DeepSeek-R1 (Nature 2025)：纯 RL (GRPO) + 可验证奖励，MATH-500 达 97.3%。
RLVR 隐式激励推理 (2025)：证明 RLVR 扩展推理边界，跨域迁移（数学 RLVR 提升代码）。
RLVRR (ICLR 2026)：扩展到开放式生成——分解为内容奖励（可验证）+ 风格奖励（LLM 验证）。
RaR / K2V (2025–2026)：扩展到医疗、知识密集型域。

第 7 章 RL for LLM 推理

传统 LLM 通过预训练”记住”了大量知识。但面对需要多步推理的复杂问题（数学证明、代码调试），仅靠”回忆”是不够的——需要学会思考的过程。
OpenAI o1/o3 和 DeepSeek-R1 的核心突破就是用 RL 训练 LLM 学习推理过程，而非记忆答案。

7.1 推理型大模型

○ 推理型 LLM

OpenAI o1 (2024.09)：引入”思考 token”做隐式推理链。核心发现：测试时计算越多，性能越好。
OpenAI o3 (2024.12)：扩展 RL + 审慎对齐。Codeforces ELO 2727。
DeepSeek-R1 (Nature 2025)：纯 RL (GRPO) 训练涌现自我反思、验证、纠错。MATH-500 达 97.3%，AIME 2024 达 79.8%。

7.2 过程奖励模型（PRM）

传统方式只在最终答案给奖励（对/错）——相当于考试只看最终成绩。PRM 在推理的每一步都给反馈——相当于老师批改每一步演算过程。这对训练和搜索都有巨大帮助。

○ PRM 系列

PRM 开发经验 (2025)：发现 MC 估计的 PRM 不如 LLM-as-Judge。提出共识过滤。
R-PRM (EMNLP 2025)：生成式 PRM——不是给分，而是推理每步是否正确。
ThinkPRM (2025)：基于长链推理的生成式 PRM。超越判别式 PRM 和 LLM-as-Judge。
PRM for RL (ICLR 2025)：过程奖励 + beam search + 在线 RL = 信用分配。

7.3 Tricks or Traps?

○ **Tricks or Traps? (2025)**

系统性复现 RL for LLM 推理的各种技巧，关键发现：

- **Batch** 级归一化 > **Group** 级（和 GRPO 的假设矛盾!）
- **Clipping** 在 **LLM** 中导致熵崩塌（PPO 的经典 clip 不一定适合 LLM）
- Base 模型用 **Token** 级 **loss**，Instruction 模型用 **Sequence** 级
- 极简两技巧组合 + vanilla PPO loss 就能超越 **GRPO** 和 **DAPO**

启示：不要盲目追新算法。理解每个 trick 背后的原理，简单组合可能更好。

第 8 章 多智能体 RL（MARL）

前面所有章节都是单个 Agent 的故事。但现实世界很多问题需要多个 **Agent** 协作（多辆无人车协调行驶）或竞争（多人博弈游戏）。
MARL 的核心难题：每个 Agent 做决策时，环境的一部分——其他 Agent——也在同时改变策略。这就像你在下棋，但棋盘规则每回合都在变。

■ **MARL 的数学框架**

Dec-POMDP（分散式部分可观测 MDP）：
N 个 Agent，每个 Agent i 有局部观测 o_i（不能看到全局状态 s），独立选择动作 a_i，共享全局奖励 $R(s, a_1, \ldots, a_N)$。
CTDE 范式（Centralized Training, Decentralized Execution）：

- 训练时：允许使用全局信息（集中式 Critic）
- 执行时：每个 Agent 只用自己的局部观测决策

信用分配问题：团队获得了总奖励 R，但哪个 **Agent** 贡献了多少?

$$R_{\text{total}} = R(s, a_1, \ldots, a_N) \quad \rightarrow \quad R_i = ? \quad \text{(每个 Agent 的贡献)}$$

○ **MARL 前沿论文**

SMACv2 (NeurIPS 2023)：程序化生成星际争霸场景，原 SMAC 被” 解决” 后的更难版本。
SMPE^2 (ICML 2025)：信念表示推理 + 对抗探索，部分可观测协作 MARL SOTA。
MAGRPO (2025)：将 GRPO 扩展到多 Agent 多轮 LLM 设置（Dec-POMDP 建模）。
LLM-MCA (AAMAS 2025)：用 LLM 做集中式 Critic 进行多 Agent 信用分配。
” 谁导致了任务失败? ” (2025)：直接解决 LLM MAS 中的自动故障归因——和 AEGIS 高度相关。
CORY (NeurIPS 2024)：两个 LLM 副本（先锋/观察者）协作博弈共同进化，减少模式崩塌。

第 9 章 Agentic RL：用 RL 训练 LLM Agent

RLHF/DPO/GRPO 训练 LLM ” 说得好”（一问一答）。但真正有用的 AI 需要” 做得好”——在浏览器里查资料、写代码修 bug、操作电脑完成任务。
Agentic RL 是 2025 年最重要的新兴方向：用 RL 训练 LLM Agent 在真实环境中多轮交互完成复杂任务。

9.1 从单轮到多轮：范式跃迁

■ **RLHF vs Agentic RL 的根本区别**

RLHF——退化的单步 MDP：

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{x,y\sim\pi}[r(x, y) - \beta D_{\text{KL}}(\pi \| \pi_{\text{ref}})]$$

一个 prompt → 一个 response → 一个奖励。没有” 环境”，没有” 状态转移”。

Agentic RL——完整的 POMDP：

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} \gamma^t R(o_t, a_t)\right]$$

Agent 观测环境（o_t = 网页截图/终端输出/API 返回）→ 执行动作（a_t = 点击/输入命令/调用工具）→ 环境变化 → 新观测 → ... → 最终任务成功或失败。
核心新挑战：

1. 长程信用分配：50 步才知道成功/失败——哪步最关键?
2. 巨大动作空间：动作是自然语言（无限可能），不是有限选项
3. 稀疏奖励：只有最终成功/失败，中间步骤无反馈
4. 环境交互开销：每步需要真实执行（打开浏览器/运行代码），不像游戏可以快速模拟

9.2 Web/GUI Agent 训练

目标：训练 Llama-3.1-8B 在 WebArena（真实网站交互环境）中完成任务
挑战：

- 训练任务稀缺——不可能手动标注几万个网页操作序列
- 奖励极其稀疏——只有最终” 任务完成/失败”
- 在线训练时策略容易崩溃——旧数据失效

WebRL 的解法：

1. 自进化课程：自动生成越来越难的训练任务（不依赖人工标注）
2. **ORM**（Outcome-supervised Reward Model）：学习判断” 这个轨迹最终会成功吗”，提供中间奖励信号
3. 自适应 **RL**：根据 Agent 当前能力动态调整训练策略

结果：Llama-3.1-8B 从 4.8% → **42.4%**（WebArena-Lite），超过 GPT-4-Turbo (17.6%)!

○ Web/GUI Agent RL 论文

WebRL (2024)：自进化课程 RL，8B 模型从 4.8%→42.4% 超 GPT-4-Turbo。
DigiRL (NeurIPS 2024)：自主 RL 训练设备控制 Agent。使用 AWR + 步级/指令级两层价值函数解决长程信用分配。
ArCHer (2024)：分层多轮 RL——高层 Agent 设置子目标，低层 Agent 执行。
AgentQ (2024)：用 DPO 从交互数据更新策略。
WebAgent-R1 (2025)：端到端多轮 RL（M-GRPO = Multi-turn GRPO），动态上下文压缩 + 异步轨迹采集。

9.3 代码/软件工程 Agent 训练

代码世界有一个 RL 梦寐以求的东西——完美的验证器。写完代码运行测试，通过就是通过，不通过就是不通过。这就是天然的可验证奖励（RLVR）。
不需要人类标注、不需要奖励模型——只需要运行代码看结果。

■ RLEF: RL from Execution Feedback

RLEF（ICML 2025）的核心框架：
环境：代码编辑器 + 测试运行器
状态：当前代码 + 执行反馈（错误信息、测试结果）
动作：生成/修改代码
奖励：$r = \begin{cases} +1 & \text{所有测试通过} \\ 0 & \text{否则} \end{cases}$
关键洞察：LLM 单独生成代码时很强，但看到执行反馈后迭代修复的能力很弱。RLEF 用 RL 训练这个" 看反馈 → 改代码" 的迭代能力。
结果：采样效率提升一个数量级，8B/70B 模型大幅提升。

○ 代码 Agent RL 论文

RLEF (ICML 2025)：RL from Execution Feedback，多轮迭代代码修复。
DeepSWE (Together AI, 2025)：Qwen3-32B + 纯 RL，SWE-Bench-Verified 达 59%（Pass@16）。64×H100 训练 6 天。
SWE-RL (Meta, NeurIPS 2025)：基于开源软件演化数据的 RL 训练。
SWE Agent 72B (2025)：改进 DAPO 适配长上下文多轮 SWE 场景，Qwen2.5-72B 达 39%（基线翻倍）。

9.4 通用 Agent 训练框架

传统 RL 框架（如 stable-baselines、RLlib）是为游戏/机器人设计的——Agent 每步获得一个数字观测，执行一个数字动作，毫秒级返回。
但 LLM Agent 完全不同：

- 观测是一大段文本/截图（几千 token）
- 动作是自然语言（需要 LLM 推理）
- 每步执行可能需要秒级甚至分钟级（打开网页、运行代码）
- 需要 Agent 与真实环境异步通信

这催生了一批专门的 Agentic RL 训练框架。

○ 通用 Agent 训练框架

AgentGym (ACL 2025)：14 个环境的统一平台（Web/文本游戏/家务/编程/具身任务等）。提出 AgentEvol 自进化算法——Agent 从自身交互经验中持续进化。
AgentGym-RL (2025)：AgentGym 的 RL 版本。提出 **ScalingInter-RL**：

- 早期阶段：限制交互步数（利用 exploitation），确保稳定学习
- 后期阶段：逐步增大交互范围（转向 exploration），发现更多策略

在 27 个任务上匹配或超越商用模型。
AgentRL (2025)：异步多任务 Agentic RL 框架。跨策略采样——从多个历史策略中采样经验，增加探索多样性。Qwen2.5/GLM-4-9B 上达到 SOTA。
Agent Lightning (2025)：灵活可扩展框架。MDP 形式化 + 统一数据接口 + 信用分配模块。支持任意 Agent 任务。
Agent-R1 (2025)：端到端 RL 训练 LLM Agent 的模块化框架。基于 verl 扩展，支持多种环境。
L-Zero (L0) (2025)：可扩展端到端训练流水线。核心创新：低成本沙箱化并发 Worker 池——每个 Worker 独立运行一个 Agent 环境实例，并行采集轨迹。

9.5 OpenClaw-RL：通过对话训练个性化 Agent

现有的 Agentic RL 系统都假设集中式、批量训练模式——先收集大量数据，再离线训练。但如果你有一个自己部署的 LLM Agent，每天和它对话时产生的反馈（你的纠正、重新提问、工具执行结果）其实就是天然的训练信号。
能不能让 **Agent** 在你日常使用它的过程中，自动从对话反馈中学习改进?
这就是 OpenClaw-RL（★4.7k+，HuggingFace 日榜 #1）要做的事。

OpenClaw-RL（Gen-Verse, 2026.03）是一个完全异步的 RL 框架，将日常对话转化为 Agent 训练信号：

■ OpenClaw-RL 的核心洞察

每次 Agent 交互都会产生一个下一状态信号（next-state signal）：

- 用户回复（" 不对，我要的是..."）→ 纠正信号
- 工具执行结果（代码报错/测试通过）→ 验证信号
- GUI 状态变化（页面是否正确加载）→ 环境信号

这些信号是通用的——几乎所有 Agent 交互都有。OpenClaw-RL 的核心就是把这些信号恢复为在线学习源。

■ 两种互补的学习方法

方法 **1**：**Binary RL**（基于 **GRPO**）
用 PRM（过程奖励模型）将下一状态信号转化为每一轮的标量奖励，再用 GRPO 的优势估计 + PPO 裁剪损失更新策略：

$$r_t = \mathrm{PRM}(o_t, a_t, o_{t+1}), \quad A_t = \frac{r_t - \bar{r}}{\sigma_r}$$

$$\mathcal{L}_{\mathrm{BinaryRL}} = -\mathbb{E}_t \left[\min(r_t(\theta) A_t, \mathrm{clip}(r_t(\theta), 1 \pm \epsilon) A_t) \right]$$

方法 **2**：**OPD**（**Hindsight-Guided On-Policy Distillation**）
当下一状态揭示了有用的事后信息时，让 Judge 模型提取文本 hint，增强原始 prompt 构造 teacher：

$$\mathrm{hint}_t = \mathrm{Judge}(o_t, a_t, o_{t+1})$$

$$\text{teacher prompt}_t = \mathrm{concat}(o_t, \mathrm{hint}_t)$$

用 teacher 和 student 的 **token** 级对数概率差作为方向性优势信号：

$$A_t^{\mathrm{OPD}} = \log \pi_{\mathrm{teacher}}(a_t | o_t, \mathrm{hint}_t) - \log \pi_{\mathrm{student}}(a_t | o_t)$$

比标量奖励更丰富——告诉模型”每个 token 应该往哪个方向调整”。
方法 **3**：**Combination（Binary RL + OPD 融合）**
同时利用 Binary RL 的密集标量监督和 OPD 的 token 级方向信号，效果最强。

■ 完全异步 4 组件架构

OpenClaw-RL 将四个组件解耦为独立的异步循环，互不阻塞：

1. **Agent Serving**：模型持续提供服务，响应用户请求
2. **Rollout Collection**：从多轮交互中自动构建训练轨迹
3. **PRM/Judge Evaluation**：异步评估每一轮的奖励/hint
4. **Policy Training**：后台持续更新模型参数

关键优势：模型在训练的同时继续服务——用户感知不到训练在发生。

Track 1：个人 **Agent** 优化（小规模但个性化）
你部署了一个自己的 LLM Agent（基于 OpenClaw 平台）。每天使用过程中，你的纠正和反馈自动成为训练数据。几天后，Agent 就学会了你的个人偏好和工作习惯。
Track 2：通用 **Agent RL**（可扩展基础设施）
大规模并行化环境交互，支持四类 Agent 任务：

- **Terminal**：命令行操作 Agent
- **GUI**：图形界面操作 Agent（类似 OSWorld）
- **SWE**：软件工程 Agent（类似 SWE-Bench）
- **Tool-call**：工具调用 Agent

○ OpenClaw-RL 关键信息

论文：OpenClaw-RL: Train Any Agent Simply by Talking (2026.03)
代码：github.com/Gen-Verse/OpenClaw-RL（★4.7k+）
基于：Slime 训练框架 + OpenClaw 运行时
亮点：HuggingFace 日榜 #1，支持 LoRA 训练，支持云端部署（Tinker），社区已集成 SDFT/SDPO 等方法。

■ Agentic RL 的三大核心挑战与解决方案

挑战 **1**：长程信用分配
Agent 可能执行 50 步才知道最终成功/失败。如何把奖励信号回溯到关键步骤?

- 步级价值函数（DigiRL）：为每步训练一个价值估计器
- **PRM**（ICLR 2025）：每步都有过程奖励
- **M-GRPO**（WebAgent-R1）：多轮 GRPO，组内比较整条轨迹
- 轨迹级 **DPO**：比较成功和失败的完整轨迹

挑战 **2**：探索-利用平衡
动作空间是自然语言（无限大），如何高效探索而不崩溃?

- **ScalingInter-RL**（AgentGym-RL）：渐进式扩大探索范围

- 自进化课程（WebRL）：根据 Agent 能力动态调整任务难度
- 跨策略采样（AgentRL）：从历史策略库中多样化采样

挑战 **3**：训练基础设施
多轮交互需要 Agent 与真实环境实时通信，延迟高、资源消耗大。

- 异步轨迹采集（WebAgent-R1）：采集和训练并行
- 沙箱化并发 **Worker**（L0）：每个 Worker 独立环境实例
- 基于 **verl** 扩展（Agent-R1）：复用成熟的 RL 训练框架

第 10 章 总结与展望

方向	**2025–2026** 主导趋势
值基方法	BTR 桌面级 SOTA；CHAIN 通用插件减少波动
连续控制	SAC 冲性能，TD3 做真实机器人，扩散策略兴起
离线 RL	Diffuser 后继、Decision Mamba 替代 Transformer、IQL+ 扩散
离线 → 在线	热身启动 WSRL、在线预训练 OPT、三阶段理论
世界模型	DreamerV3 登 Nature；Delta-IRIS 提效；DyMoDreamer 双线超越
LLM 对齐	SFT→DPO/SimPO→GRPO/DAPO；GRPO 本质是 DPO
RLVR	最大范式转变：从数学/代码扩展到医疗、知识、开放生成
LLM 推理	DeepSeek-R1 登 Nature；生成式 PRM 替代判别式；极简技巧超 GRPO
MARL	SMACv2 重挑战；LLM 集中式信用分配；MAGRPO
Agentic RL	最大新方向：WebRL/RLEF/DeepSWE 证明 RL 大幅提升 Agent；通用框架涌现

回看这张演进路线图，每一代 RL 都在解决上一代的核心瓶颈。如果当前趋势延续：
短期（**2026–2027**）：Agentic RL 将成为训练 LLM Agent 的标准方法。WebRL/RLEF 模式（环境反馈 + 在线 RL）会替代纯 SFT。DeepSWE 在 SWE-Bench 上刷到 59% 只是开始。
中期（**2027–2028**）：世界模型 + Agentic RL 融合——Agent 不仅在真实环境交互学习，还在”想象”中预演。Dreamer 式的世界模型会进入 LLM Agent 训练。
长期（**2028+**）：多 Agent 系统的 RL 训练。不是一个 Agent，而是一群 Agent 协作/竞争学习。MAGRPO 是雏形。错误归因（AEGIS 方向）将变成刚需。