```
# chat.py
CUDA_VISIBLE_DEVICES=0 python chat.py --path model_path
```

如此，即可通过命令行进行交互式测试。

# 6. 强化学习

通过有监督微调，大语言模型已初步具备遵循人类指令并完成多类型任务的能力。然而该方法存在显著局限：首先需要构建海量指令-答案对数据集，高质量回复标注需耗费高昂人力成本；其次交叉熵损失函数要求模型输出与标准答案逐字匹配，既无法适应自然语言的表达多样性，也难以解决输出对输入微小变动的敏感性，这在需要深度推理的复杂任务中尤为突出。

当前大语言模型中的强化学习技术主要沿着两个方向演进：其一是基于人类反馈的强化学习（Reinforcement Learning from Human Feedback，RLHF），通过奖励模型对生成文本进行整体质量评估，使模型能自主探索更优的回复策略，并使得模型回复与人类偏好和价值观对齐。典型如 ChatGPT 等对话系统，通过人类偏好数据训练奖励模型，结合近端策略优化（Proximal Policy Optimization，PPO）算法实现对齐优化。其二是面向深度推理的强化学习框架，以 OpenAI 的 O 系列模型和 DeepSeek 的 R 系列为代表，通过答案校验引导模型进行多步推理。这类方法将复杂问题分解为长思维链（Chain-of-Thought）的决策序列，在数学证明、代码生成等场景中展现出超越监督学习的推理能力。

相较于传统监督学习，强化学习框架具有显著优势：在 RLHF 范式下，模型通过生成-反馈的闭环机制持续优化，摆脱对标准答案的绝对依赖；在深度推理场景中，强化学习能自主探索最优推理路径，通过价值函数估计引导模型突破局部最优解。两类方法都强调对生成文本的整体质量把控，前者侧重人类价值对齐，后者专注复杂问题求解，共同构成大语言模型能力进化的核心驱动力。

本章将系统阐述基于人类反馈的强化学习技术体系，解析奖励模型构建、策略优化算法等关键组件。同时深入探讨强化学习在深度推理任务中的创新应用，包括思维链强化、过程奖励设计等前沿方法。最后通过 verl 实践案例，展示强化学习技术在大语言模型训练中的工程实现与效果验证。

## 6.1 强化学习概述

**强化学习**（Reinforcement Learning，RL）研究的是**智能体**与**环境**交互的问题，其目标是使智能体在复杂且不确定的环境中最大化**奖励**。强化学习基本框架如图6.1 所示，主要由两部分组成：

智能体和环境。在强化学习过程中，智能体与环境不断交互。智能体在环境中获取某个状态后，会根据该状态输出一个**动作**，也称为**决策**。动作会在环境中执行，环境会根据智能体采取的动作，给出下一个状态及当前动作带来的奖励。智能体的目标就是尽可能多地从环境中获取奖励。本节将介绍强化学习的基本概念、强化学习与有监督学习的区别，以及在大语言模型中基于人类反馈的强化学习流程。

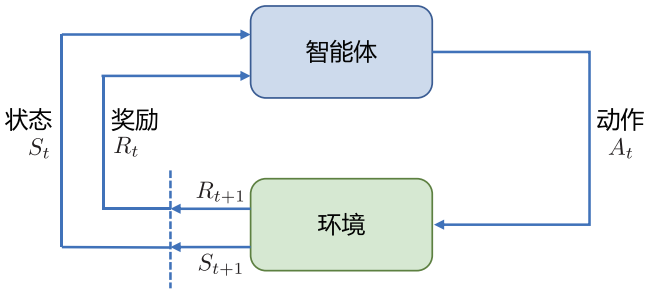


图 6.1　强化学习基本框架

在现实生活中，经常会遇到需要通过探索和试错来学习的情境。例如，孩子学会骑自行车的过程或是教宠物狗如何玩飞盘。宠物狗一开始对如何抓飞盘一无所知，但每当它成功抓住飞盘时，都可以给予它一定的奖励。这种通过与环境交互，根据反馈来学习最佳行为的过程正是强化学习的核心思想。通过宠物狗学习抓飞盘的例子，可以引出一些强化学习中的基本概念。

（1）**智能体与环境**：在宠物狗学习抓飞盘的场景中，宠物狗就是一个**智能体**（Agent），它做出**决策**（Decision）并执行动作。它所在的场景，包括飞盘的飞行轨迹和速度，以及其他可能的因素，构成了**环境**（Environment）。环境会根据智能体的行为给予反馈，通常以奖励的形式。

（2）**状态、行为与奖励**：每次宠物狗尝试抓飞盘，它都在评估当前的**状态**（State），这可能包括飞盘的位置、速度等。基于这些信息，它会采取某种**动作**（Action），如跳跃、奔跑或待在原地。根据宠物狗所执行的动作，环境随后会给出一个**奖励**（Reward），这可以是正面的（成功抓住飞盘）或负面的（错过了飞盘）。

（3）**策略与价值**：在尝试各种行为的过程中，宠物狗其实是在学习一个**策略**（Policy）。策略可以视为一套指导其在特定状态下如何行动的规则。与此同时，智能体还试图估计**价值**（Value）函数，也就是预测在未来采取某一行为所能带来的奖励。

总体来说，强化学习的目标就是让智能体通过与环境的互动，学习到一个策略，使其在将来能够获得的奖励最大化。这使得强化学习不总是关注短期奖励，而是在短期奖励与远期奖励之间找到平衡。

### 6.1.1 强化学习基础概念

智能体与环境的不断交互过程中，会获得很多观测 $o_i$。针对每一个观测，智能体会采取一个动作 $a_i$，也会得到一个奖励 $r_i$。可以定义历史 $H_t$ 是观测、动作、奖励的序列：

$$H_t = o_1, a_1, r_1, o_2, a_2, r_2, \cdots, o_t, a_t, r_t \tag{6.1}$$

由于智能体在采取当前动作时会依赖它之前得到的历史，因此可以把环境整体状态 $S_t$ 看作关于历史的函数：

$$S_t = f(H_t) \tag{6.2}$$

当智能体能够观察到环境的所有状态时，称环境是完全可观测的（Fully Observed），这时观测 $o_t$ 等于 $S_t$。当智能体只能看到部分观测时，称环境是部分可观测的（Partially Observed），这时观测是对状态的部分描述。整个状态空间使用 $S$ 表示。

在给定的环境中，有效动作的集合经常被称为**动作空间**（Action Space），使用 $A$ 表示。例如围棋（Go）这样的环境具有**离散动作空间**（Discrete Action Space），智能体的动作数量在这个空间中是有限的。智能体在围棋中的动作空间只有 361 个交叉点，而在物理世界中则通常是**连续动作空间**（Continuous Action Space）。在连续动作空间中，动作通常是实值的向量。例如，在平面中，机器人可以向任意角度进行移动，其动作空间为连续动作空间。

**策略**是智能体的动作模型，决定了智能体的动作。策略也可以用函数表示，该函数将输入的状态变成动作。策略可分为两种：随机性策略和确定性策略。**随机性策略**（Stochastic Policy）用 $\pi$ 函数表示，即 $\pi(a|s) = p(a_t = a|s_t = s)$，输入一个状态 $s$，输出一个概率，表示智能体所有动作的概率。利用这个概率分布进行采样，就可以得到智能体将采取的动作。**确定性策略**（Deterministic Policy）是智能体最有可能直接采取的动作，即 $a^* = \arg\max_a \pi(a|s)$。

**价值函数**的值是对未来奖励的预测，可以用它来评估状态的好坏。价值函数可以只根据当前的状态 $s$ 决定，使用 $V_\pi(s)$ 表示。也可以根据当前状态 $s$ 及动作 $a$，使用 $Q_\pi(s,a)$ 表示。$V_\pi(s)$ 和 $Q_\pi(s,a)$ 的具体定义如下：

$$V_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s\right], s \in S \tag{6.3}$$

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s, a_t = a\right] \tag{6.4}$$

其中，$\gamma$ 为**折扣因子**（Discount Factor），针对短期奖励和远期奖励进行折中；期望 $\mathbb{E}$ 的下标为 $\pi$ 函数，其值反映在使用策略 $\pi$ 时所能获得的奖励值。

根据智能体所学习机制的不同，可以把智能体归类为基于价值的智能体、基于策略的智能体和演员–评论员智能体。**基于价值的智能体**（Value-based Agent）显式地学习价值函数 $V_\pi(s)$ 或 $Q_\pi(s,a)$，隐式推导策略，典型算法如 Q-Learning。**基于策略的智能体**（Policy-based Agent）则直接学习策略函数 $\pi_\theta(a|s)$。策略函数的输入为一个状态，输出为对应动作的概率。基于策略的智能体并不学习价值函数，价值函数隐式地表达在策略函数中，典型算法如 REINFORCE。**演员–评论员智能体**（Actor-critic Agent）则是把基于价值的智能体和基于策略的智能体结合起来，既学习策略函数又学习价值函数，通过两者的交互得到最佳的动作，典型算法如 PPO。

### 6.1.2 强化学习与有监督学习的区别

在深度学习中，有监督学习和强化学习不同，可以用旅行方式对二者进行更直观的对比，有监督学习和强化学习可以看作两种不同的旅行方式，每种旅行都有自己独特的风景、规则和探索方式。

- **旅行前的准备：数据来源**

  **有监督学习**：这如同旅行者拿着一本旅行指南书，其中明确标注了各个景点、餐厅和交通方式。在这里，数据来源就好比这本书，提供了清晰的问题和答案。

  **强化学习**：旅行者进入了一个陌生的城市，手上没有地图，没有指南。他们只知道自己的目的，例如找到城市中的一家餐厅或博物馆。这座未知的城市，正是强化学习中的数据来源，充满了探索的机会。

- **路途中的指引：反馈机制**

  **有监督学习**：在这座城市里，每当旅行者迷路或犹豫时，都会有人告诉他们下一步应该如何去做。这就好比旅行者无需自己摸索，有监督学习会告诉他们如何行动。

  **强化学习**：在另一座城市，没有人会直接告诉旅行者如何走。只会告诉他们结果是好还是坏。例如，走进了一家餐厅，吃完饭后才知道这家餐厅是否合适。需要通过多次尝试，逐渐学习和调整策略。

- **旅行的终点：目的地**

  **有监督学习**：在这座城市旅行的目的非常明确，学习整个训练轨迹，就像参观完旅行指南上提及的所有景点。

  **强化学习**：在未知的城市，目标是学习如何在其中有效地行动，寻找最佳的路径，无论是寻找食物、住宿还是娱乐。

现代强化学习之父 Richard Sutton 在《苦涩的教训（The Bitter Lesson）》中指出，过去 70 年人工智能研究领域最重要的一堂课是，只有通用的、可规模化扩展的方法才是最终有效的，而且优势巨大。因此，结合 OpenAI 的研究实践，强化学习在大语言模型中的优势可重新归纳为以下三个维度：

（1）摆脱局部最优束缚的全局优化能力。监督学习依赖词元级精确标注，本质上将人类先验

知识固化为离散标签，导致模型陷入局部最优（如交叉熵损失对语义突变的迟钝性）。强化学习通过整体奖励信号替代人工拆解的局部规则，允许模型自主探索语义组合的可能性。这种粗粒度反馈 + 自主优化机制，既保留了自然语言的表达多样性（如“非常满意”与“无可挑剔”的等效性），又能捕捉关键否定词带来的语义反转（如“不推荐”与“强烈推荐”的极性差异），印证了《The Bitter Lesson》强调的“减少人工规则设计，让算法自主发现最优路径”原则。

（2）突破人类认知边界的知识演进机制。监督学习在求知型查询中的幻觉问题，本质源于其“知识天花板”——模型无法超越标注数据覆盖的认知范畴。RL 通过动态奖励函数构建知识可信度评估体系：对正确回答给予指数级奖励，对错误答案施加惩罚梯度，使模型自主发展出“知之为知之”的认知边界意识。这种不依赖静态知识库的持续进化模式，与 AlphaGo 通过自我对弈突破人类棋谱局限的路径异曲同工，实现了《The Bitter Lesson》倡导的算法应通过计算规模扩展而非人工知识注入来提升能力。

（3）面向复杂系统的长期价值建模范式。在多轮对话场景中，监督学习的即时反馈机制难以捕捉跨轮次的语义关联与长期目标。RL 通过价值函数网络建模状态-动作的长期收益，将对话连贯性、信息增量等抽象目标转化为可优化的数学指标。这种基于延迟奖励的序列决策框架，使模型能够自主平衡即时响应质量与对话终局目标的关系，正如 AlphaStar 在《星际争霸》中通过数千步决策实现战略布局，验证了放弃短期人工启发式设计，专注构建通用长期优化架构的前瞻性。

## 6.2 策略梯度方法

在强化学习领域，智能体通过与环境的交互试错来学习最优策略，其核心目标是通过最大化长期累积奖励，找到最佳决策路径。传统方法（如 Q-learning）通常基于“价值函数”间接优化策略——先评估动作的价值，再选择最优动作。然而，当面对高维或连续动作空间时（例如机器人控制、游戏角色复杂操作），这类方法可能面临计算瓶颈或难以收敛的问题。

策略梯度（Policy Gradient）方法提供了一种更直接的思路：它摒弃了“先估值再决策”的中间步骤，而是将策略本身参数化（例如用神经网络表示），直接通过梯度上升优化策略参数，让智能体更倾向于选择能带来高回报的动作。简单来说，策略梯度通过反复试验，统计哪些动作在特定状态下更容易获得奖励，并像“调整旋钮”一样微调策略，使得这些动作在未来被选中的概率逐渐增加。

这一方法的优势在于能天然处理连续动作、随机策略以及部分观测环境，但也面临梯度估计方差大、训练不稳定等挑战。本节将从策略梯度的基础概念出发，回顾经典算法如 REINFORCE，PPO 等，并讨论在大模型时代流行的 GRPO，RLOO 等方法。

### 6.2.1 策略梯度

策略梯度方法是强化学习中一类重要的算法，它直接优化策略函数 $\pi(a|s;\theta)$，以最大化预期的回报（累计奖励）$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$，其中 $\theta$ 是策略的参数。

假设环境初始状态分布为 $p_0(s)$，初始状态 $s_0 \sim p_0(s)$，智能体依据策略函数 $\pi(a|s;\theta)$ 给出动作 $a_0$，环境根据奖励函数 $r(s,a)$ 给出奖励，并依据转移概率 $P(s'|s,a)$ 转移到下一个状态 $s_1$。重复这一过程，可得到智能体与环境交互的轨迹（Trajectory）$\tau = (s_0, a_0, s_1, a_1, \cdots)$，其发生概率为：

$$P(\tau;\theta) = p_0(s_0)\prod_{t=0}^{\infty}\pi_\theta(a_t|s_t)P(s_{t+1}|s_t,a_t) \tag{6.5}$$

优化目标是最大化轨迹的期望回报 $J(\theta)$，即：

$$J(\theta) = \mathbb{E}_{\tau\sim P(\tau;\theta)}\left[R(\tau)\right] \tag{6.6}$$

使用梯度上升法优化参数 $\theta$，计算期望回报的梯度为：

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{\tau\sim P(\tau;\theta)}\left[R(\tau)\right] = \mathbb{E}_{\tau\sim P(\tau;\theta)}\left[\nabla_\theta \log P(\tau;\theta)R(\tau)\right] \tag{6.7}$$

这里运用了对数导数技巧 $\nabla_\theta P(\tau;\theta) = P(\tau;\theta)\nabla_\theta \log P(\tau;\theta)$。

进一步展开 $\nabla_\theta \log P(\tau;\theta)$，考虑到环境初始状态概率 $p_0(s_0)$ 和转移概率 $P(s_{t+1}|s_t,a_t)$ 通常与策略参数 $\theta$ 无关，导数为零，可得：

$$\nabla_\theta \log P(\tau;\theta) = \sum_{t=0}^{\infty}\nabla_\theta \log \pi_\theta(a_t|s_t) \tag{6.8}$$

代入后得到：

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim P(\tau;\theta)}\left[R(\tau)\sum_{t=0}^{\infty}\nabla_\theta \log \pi_\theta(a_t|s_t)\right] \tag{6.9}$$

理解该策略梯度公式的关键在于，$R(\tau)$ 可看作 $\pi_\theta(a_t|s_t)$ 的权重。但当前动作不影响历史奖励，用整条轨迹的累积回报衡量当前动作价值不合理。因此，使用从当前状态 $s_t$ 采取动作 $a_t$ 后的回报 $R_t = \sum_{t'=t}^{\infty}\gamma^{t'-t}r_{t'}$ 作为权重衡量动作价值，并将策略梯度按时刻累加：

$$\nabla_\theta J(\theta) = \sum_{t=0}^{\infty}\mathbb{E}_{\tau\sim P(\tau;\theta)}\left[R_t\nabla_\theta \log \pi_\theta(a_t|s_t)\right] \tag{6.10}$$

我们可以使用学习率为 $\eta$ 的梯度上升方法优化策略参数 $\theta$：

$$\theta \leftarrow \theta + \eta\nabla_\theta J(\theta) \tag{6.11}$$

在策略梯度算法中，累积回报 $R_t$ 包含轨迹的随机性，受初始状态、后续动作选择和环境状态

转移影响，不同轨迹间回报波动大，导致方差很大。直接用 $R_t$ 作为梯度更新权重，会使每一步梯度估计值不稳定，增加训练波动性，减缓策略收敛速度。

为降低策略梯度方法中回报 $R_t$ 的方差，计算 $\nabla_\theta J(\theta)$ 时通常引入基线（Baseline）。基线是仅依赖于状态 $s_t$ 的函数 $b(s_t)$，对期望回报中的梯度做如下变换，不改变其期望：

$$\nabla_\theta J(\theta) = \sum_{t=0}^{\infty} \mathbb{E}_{\tau \sim P(\tau;\theta)} \left[ (R_t - b(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t) \right] \tag{6.12}$$

使用 $R_t - b(s_t)$ 作为权重替代 $R_t$，因为 $b(s_t)$ 不依赖于动作 $a_t$，所以：

$$\begin{aligned} \mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)} \left[ b(s_t) \nabla_\theta \log \pi_\theta(a_t|s_t) \right] &= b(s_t) \mathbb{E}_{a_t \sim \pi_\theta(a_t|s_t)} \left[ \nabla_\theta \log \pi_\theta(a_t|s_t) \right] \\ &= b(s_t) \sum_{a_t} \left[ \pi_\theta(a_t|s_t) \nabla_\theta \log \pi_\theta(a_t|s_t) \right] \\ &= b(s_t) \sum_{a_t} \left[ \nabla_\theta \pi_\theta(a_t|s_t) \right] \\ &= b(s_t) \nabla_\theta \sum_{a_t} \left[ \pi_\theta(a_t|s_t) \right] = 0 \end{aligned} \tag{6.13}$$

常用的基线选择是状态价值函数 $V(s_t)$，即 $V(s_t) = \mathbb{E}_{\tau \sim P(\tau;\theta)}[R_t|s_t]$。此时，$R_t$ 可视为动作价值函数 $Q(s_t, a_t) = \mathbb{E}[R_t|s_t, a_t]$ 的蒙特卡洛估计，策略梯度更新公式进一步表示为：

$$\nabla_\theta J(\theta) = \sum_{t=0}^{\infty} \mathbb{E}_{\tau \sim P(\tau;\theta)} \left[ (Q(s_t, a_t) - V(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t) \right] \tag{6.14}$$

其中，$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$ 被称为优势函数，衡量动作 $a_t$ 相对于状态 $s_t$ 的预期回报提升。当 $R_t > V(s_t)$ 时，说明动作 $a_t$ 带来的实际回报高于状态 $s_t$ 的平均预期回报，应增加其选择概率；反之则降低概率。

### 6.2.2 REINFORCE 算法

REINFORCE 算法是最基础的策略梯度方法之一，由 Ronald J. Williams 于 1992 年提出。其核心思想是通过蒙特卡洛采样方法直接估计策略梯度，利用轨迹的完整回报（Complete Return）来更新策略参数 $\theta$，从而最大化期望累积奖励。

#### 1. 算法原理

考虑有限时间步的任务，轨迹 $\tau = (s_0, a_0, r_0, s_1, a_1, r_1, \cdots, s_T)$ 的累积回报为：

$$R_t = \sum_{k=t}^{T} \gamma^{k-t} r_k \tag{6.15}$$

其中 $\gamma \in [0,1]$ 为折扣因子。根据策略梯度，目标函数 $J(\theta)$ 的梯度可表示为：

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim P(\tau;\theta)} \left[ \sum_{t=0}^{T} R_t \nabla_\theta \log \pi_\theta(a_t|s_t) \right] \tag{6.16}$$

REINFORCE 算法通过蒙特卡洛采样近似该期望，使用 $N$ 条轨迹的样本均值估计梯度：

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} R_t^{(n)} \nabla_\theta \log \pi_\theta(a_t^{(n)}|s_t^{(n)}) \tag{6.17}$$

其中上标 $(n)$ 表示第 $n$ 条轨迹的采样结果。

### 2. 算法步骤

REINFORCE 算法的具体实现步骤如下：

(1) **初始化策略参数**：随机初始化策略网络参数 $\theta$。

(2) **采样轨迹**：使用当前策略 $\pi_\theta(a|s)$ 与环境交互，收集 $N$ 条轨迹 $\{\tau^{(1)}, \tau^{(2)}, \cdots, \tau^{(N)}\}$。

(3) **计算回报**：对每条轨迹 $\tau^{(n)}$，计算每个时刻 $t$ 的累积回报 $R_t^{(n)} = \sum_{k=t}^{T} \gamma^{k-t} r_k^{(n)}$。

(4) **估计梯度**：通过样本平均计算策略梯度估计值：

$$\hat{\nabla}_\theta J(\theta) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} G_t^{(n)} \nabla_\theta \log \pi_\theta(a_t^{(n)}|s_t^{(n)}) \tag{6.18}$$

(5) **更新参数**：沿梯度方向更新策略参数：

$$\theta \leftarrow \theta + \alpha \hat{\nabla}_\theta J(\theta) \tag{6.19}$$

其中 $\alpha$ 为学习率。

(6) **重复迭代**：重复步骤 2-5 直至策略收敛。

### 3. 引入基线降低方差

直接使用 $G_t$ 作为权重会导致梯度估计的高方差。为此，REINFORCE 算法常引入状态相关的基线函数 $b(s_t)$，将策略梯度修改为：

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim P(\tau;\theta)} \left[ \sum_{t=0}^{T} (R_t - b(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t) \right] \tag{6.20}$$

基线函数需满足与动作 $a_t$ 无关的条件。理论上，最优基线函数为状态价值函数 $V(s_t)$，此时 $R_t - V(s_t)$ 称为优势函数。实际中常使用状态价值函数的估计值 $\hat{V}(s_t)$ 作为基线，其参数可通过

监督学习更新：

$$\min_{\phi} \sum_{n=1}^{N} \sum_{t=0}^{T} \left( \hat{V}_{\phi}(s_t^{(n)}) - R_t^{(n)} \right)^2 \tag{6.21}$$

加入基线后，参数更新公式变为：

$$\theta \leftarrow \theta + \alpha \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T} \left( R_t^{(n)} - \hat{V}_{\phi}(s_t^{(n)}) \right) \nabla_{\theta} \log \pi_{\theta}(a_t^{(n)} | s_t^{(n)}) \tag{6.22}$$

基线函数不改变梯度的期望值，但能显著降低方差。数学上可证明：

$$\mathbb{E}_{a_t \sim \pi_{\theta}} \left[ b(s_t) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \right] = b(s_t) \nabla_{\theta} \sum_{a_t} \pi_{\theta}(a_t | s_t) = 0 \tag{6.23}$$

#### 4. 算法特性分析

基于蒙特卡洛采样的 REINFORCE 方法作为经典的策略梯度算法，存在以下显著缺陷：首先，其依赖完整轨迹采样的蒙特卡洛特性导致梯度估计方差过高，这不仅会显著延缓收敛速度，还容易引发策略更新方向的剧烈波动，造成训练过程的不稳定性；其次，算法必须等待整条轨迹结束后才能更新策略参数，在长周期任务或持续性环境中会大幅降低学习效率；此外，其在线学习机制要求每次策略更新后必须重新采样轨迹数据，导致样本利用率低下，难以适应大规模复杂任务的需求。虽然策略的随机性天然具备探索优势，但高方差问题可能削弱这一优势对学习效果的促进作用。最后，该方法主要适用于小规模离散动作空间场景，对函数近似误差敏感的特性也限制了其在连续动作空间或深度强化学习框架中的应用范围。

### 6.2.3 广义优势估计

为了克服蒙特卡洛方法的缺陷（高方差和完整轨迹依赖），研究者们提出了时序差分方法（Temporal Difference Methods，TD）。时序差分方法基于动态规划的思想，通过引入 Bootstrapping 机制，即利用当前的价值估计来更新自身，而不必等待完整的轨迹结束。这种方法允许在每个时间步进行更新，极大地提高了样本效率。

对于给定的状态 $s_t$ 和动作 $a_t$，时序差分方法的基本更新公式为：

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_t + \gamma V(s_{t+1}) Q(s_t, a_t)]$$

其中，$\alpha$ 是学习率，控制更新步长，$\gamma$ 是折扣因子，控制未来奖励的权重。由于只涉及单步奖励和下一个状态的估计，TD 方法的方差通常低于蒙特卡洛方法，可以在每个时间步进行更新，无须等待完整的轨迹结束，提高了样本效率。

因此，为了估计当前动作价值，不必采样未来的很多步，而只采样一步。对于一步之后的很多步结果，则使用状态价值函数进行估计，即

$$Q(s_t, a_t) = r_t + \gamma V(s_{t+1})$$

假设 $V(s_t)$ 是无偏的，那么动作价值也是无偏的，即：

$$\mathbb{E}\left[r_t + \gamma V(s_{t+1})\right] = \mathbb{E}\left[r_t + \gamma \mathbb{E}\left[\sum_{t'=t+1}^{T} \gamma^{t'-t-1} r_{t'}\right]\right]$$

通过展开，我们得到：

$$\mathbb{E}\left[r_t + \gamma \sum_{t'=t+1}^{T} \gamma^{t'-t-1} r_{t'}\right] = \mathbb{E}\left[r_t + \sum_{t'=t+1}^{T} \gamma^{t'-t} r_{t'}\right] = \mathbb{E}\left[\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}\right]$$

前面使用了 $V_\phi(s_t)$ 来近似 $V(s_t)$，这可能导致 $r_t + \gamma V_\phi(s_{t+1})$ 有较高的偏差，尽管其方差较低。

类似地，可以采样 $k$ 步奖励，即

$$Q^k(s_t, a_t) = r_t + \gamma r_{t+1} + \cdots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k})$$

随着 $k$ 的增大，这个结果逐渐趋向于蒙特卡洛方法。因此，从蒙特卡洛方法到时序差分方法，方差逐渐减小，偏差逐渐增大。$k$ 步优势可以定义为：

$$A_t^k = r_t + \gamma r_{t+1} + \cdots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k}) - V(s_t)$$

蒙特卡洛方法具有高方差、无偏差，而时序差分方法具有低方差、高偏差。为了权衡方差与偏差，广义优势估计（Generalized Advantage Estimation，GAE）方法将优势函数定义为 $k$ 步优势的指数平均：

$$A_t^{\mathrm{GAE}(\gamma,\lambda)} = (1-\lambda)\left(A_t^1 + \lambda A_t^2 + \lambda^2 A_t^3 + \cdots\right)$$

通过这种方式，广义优势估计能够同时利用蒙特卡洛方法和时序差分方法的优势，从而实现低方差、低偏差的效果。因此，GAE 广泛应用于策略梯度方法中。

然而，之前定义的广义优势估计的形式计算复杂度较高，需要求解多个 $k$ 步优势值。为了优化这一过程，有必要引入优化。可以通过引入 TD 误差（TD-error）$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$，将 $k$ 步优势 $A_t^k$ 转化为：

$$A_t^k = \sum_{l=1}^{k} \gamma^{l-1} \delta_{t+l-1}$$

通过这种方式，我们将 $k$ 步优势的计算转化为对每一步的 TD 误差的加权求和，从而降低了计算复杂度。

将上述结果代入广义优势估计的公式，可以得到：

$$A_t^{\mathrm{GAE}(\gamma,\lambda)} = (1-\lambda)\left(\delta_t + \lambda(\delta_t + \gamma\delta_{t+1}) + \lambda^2(\delta_t + \gamma\delta_{t+1} + \gamma^2\delta_{t+2}) + \cdots\right)$$

简化后，得到：

$$A_t^{\mathrm{GAE}(\gamma,\lambda)} = (1-\lambda)\left(\delta_t\left(\frac{1}{1-\lambda}\right) + \gamma\delta_{t+1}\left(\frac{\lambda}{1-\lambda}\right) + \gamma^2\delta_{t+2}\left(\frac{\lambda^2}{1-\lambda}\right) + \ldots\right)$$

最终，可以表示为：

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

GAE 的定义平滑地插值于高偏差（当 $\lambda = 0$ 时）和高方差（当 $\lambda = 1$ 时）之间，有效地管理了偏差与方差的权衡。

当 $\lambda = 0$ 时，GAE 退化为单步 TD 误差：

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

当 $\lambda = 1$ 时，GAE 退化为完整的蒙特卡洛方法：

$$A_t = \sum_{l=0}^{\infty}\gamma^l\delta_{t+l} = \sum_{l=0}^{\infty}\gamma^l r_{t+l} - V(s_t)$$

### 6.2.4 近端策略优化算法

获得广义优势函数后，我们可以低偏差和低方差地估计动作的相对优势，从而高效地引导策略梯度的更新。将优势函数 $A(s,a)$ 代入策略梯度公式，得到：

$$\begin{aligned}\nabla_\theta J(\theta) &= \sum_{t=0}^{\infty}\mathbb{E}_{(s_t,a_t)\sim\pi_\theta(a_t|s_t)}\left[A(s_t,a_t)\nabla_\theta\log\pi_\theta(a_t|s_t)\right] \\ &= \mathbb{E}_{(s,a)\sim\pi_\theta(a|s)}\left[A(s,a)\nabla_\theta\log\pi_\theta(a|s)\right]\end{aligned} \tag{6.24}$$

这个更新方式的问题在于，在实际更新策略参数 $\theta$ 的过程中，每次采样一批数据进行更新时，概率分布 $\pi_\theta(a|s)$ 会发生变化。由于分布改变，之前采集的数据便不能在下一轮更新中再利用。因

此，策略梯度方法需要不断地在环境交互中学习，训练效率较低。

注意，在策略梯度方法中，同一个智能体既负责与环境交互，也负责策略参数更新，这种训练方法被称为**同策略**（On-Policy）训练方法。相反，**异策略**（Off-Policy）训练方法将这两个职能分开，即固定一个智能体与环境交互而不更新，另一个智能体则只负责从采集的数据中学习更新参数。这种方式可以重复利用历史数据。然而，由于两个智能体的分布不同，直接更新会导致不稳定的训练。一种思路是调整这两个分布使它们保持一致，**重要性采样**（Importance Sampling）就是在这种思路下的重要技术。

### 1. 算法原理

假设我们希望计算期望 $\mathbb{E}_{x\sim P(x)}[f(x)]$，但采样数据来自另一个分布 $Q(x)$，可以通过设置采样数据的权重来修正结果：

$$\mathbb{E}_{x\sim P(x)}[f(x)] = \mathbb{E}_{x\sim Q(x)}\left[\frac{P(x)}{Q(x)}f(x)\right] \tag{6.25}$$

从 $P$ 中每次采样一个 $x^i$ 并计算 $f(x^i)$，都需要乘上一个重要性权重 $\frac{P(x^i)}{Q(x^i)}$ 来修正这两个分布的差异，这种方法被称为重要性采样。通过这种方式，我们可以从分布 $Q$ 中采样，并计算当 $x$ 服从分布 $P$ 时的期望。

不过，两个分布的差异不能过大，否则会导致以下问题：

(1) **高方差**：当分布差异较大时，权重 $\frac{P(x)}{Q(x)}$ 可能出现极端值，导致估计的期望值方差增大。

(2) **偏差**：为了解决高方差问题，通常需要对权重进行裁剪或限制，这可能引入偏差。

假设用于与环境交互的智能体策略为 $\theta'$，用于学习的智能体策略为 $\theta$，应用重要性采样后，可以将策略梯度公式改为异策略的形式，即：

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_{(s,a)\sim\pi_\theta(a|s)}\left[A(s,a)\nabla_\theta \log \pi_\theta(a|s)\right] \\ &= \mathbb{E}_{(s,a)\sim\pi_{\theta'}(a|s)}\left[\frac{p_\theta(s,a)}{p_{\theta'}(s,a)}A(s,a)\nabla_\theta \log \pi_\theta(a|s)\right]\end{aligned} \tag{6.26}$$

其中，$p_\theta(s,a) = \pi_\theta(a|s)p(s)$ 表示状态-动作对出现的概率，状态的概率被认为与策略无关，以便进行优化。因此，最终的策略梯度为：

$$\nabla_\theta J(\theta) = \mathbb{E}_{(s,a)\sim\pi_{\theta'}(a|s)}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)}A(s,a)\nabla_\theta \log \pi_\theta(a|s)\right] \tag{6.27}$$

从上述梯度形式反推 PPO 的目标函数为：

$$J(\theta) = \mathbb{E}_{(s,a)\sim\pi_{\theta'}(a|s)}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)}A(s,a)\right] \tag{6.28}$$

前面提到，重要性采样需要保证两个策略分布相似，否则高方差会导致优化不稳定。因此，PPO 算法引入了剪切机制，通过将权重限制在特定范围内来避免优化不稳定，即：

$$J_{\text{PPO}}(\theta) = \mathbb{E}_{(s,a)\sim\pi_{\theta'}(a|s)}\left[\text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta'}(a|s)}, 1-\varepsilon, 1+\varepsilon\right) A(s,a)\right] \tag{6.29}$$

其中，$\varepsilon$ 是超参数（例如可以设置为 0.1 或 0.2）。Clip 函数裁剪重要性权重的大小，限制权重在 $1-\varepsilon$ 和 $1+\varepsilon$ 之间。

#### 2. 算法流程

综合上面的推导过程，我们可以得到 PPO 算法的流程，如代码 6.1 所示。

**代码 6.1:** PPO 算法的流程

输入: 初始策略参数 $\theta_0$，初始价值函数参数 $\phi_0$
**for** $n = 0, 1, 2, \cdots$ **do**
  收集轨迹集合 $\mathcal{D}_n = \{\tau_i\}$，通过在环境中执行策略 $\pi_{\theta_n}$
  针对每条轨迹计算回报 $R_t$
  基于当前的价值函数 $V_{\phi_n}$，使用广义优势估计方法计算优势 $A_t$
  通过最小化策略梯度损失函数目标来更新策略：

$$\theta_{n+1} = \arg\max_\theta J_{\text{PPO}}(\theta_n)$$

  通过最小化均方误差来更新价值函数：

$$\phi_{n+1} = \arg\min_\phi \mathcal{L}(\phi_n)$$

**end for**

### 6.2.5 RLOO

REINFORCE Leave-One-Out（RLOO）算法是在 REINFORCE 算法基础上发展而来的一种改进算法，它主要针对 REINFORCE 算法梯度估计方差较高的问题，通过利用多个在线样本构建更有效的基线来降低方差，从而提升算法性能。

#### 1. 算法原理

RLOO 的核心在于改进基线的构建方式。在 REINFORCE 算法中，通常使用简单的移动平均基线，这种基线在处理复杂环境和多样本情况时存在一定局限性。RLOO 则利用每次采样得到的多个样本之间的关系，为每个样本单独构建基线。

假设在一次训练中，从策略 $\pi_\theta(a|s)$ 中采样得到 $k$ 个独立同分布的样本 $y_{(1)},\cdots,y_{(k)} \overset{i.i.d}{\sim} \pi_\theta(.|x)$，对于每个样本 $y_{(i)}$，其对应的奖励为 $R(y_{(i)},x)$。RLOO 构建的基线为除 $y_{(i)}$ 之外的其他 $k-1$ 个样本奖励的平均值，即 $\frac{1}{k-1}\sum_{j\neq i}R(y_{(j)},x)$。

基于此，RLOO 的策略梯度估计公式为：

$$\frac{1}{k}\sum_{i=1}^{k}\left[R\left(y_{(i)},x\right)-\frac{1}{k-1}\sum_{j\neq i}R\left(y_{(j)},x\right)\right]\nabla\log\pi\left(y_{(i)}|x\right) \tag{6.30}$$

这个公式的含义是，对每个样本的奖励减去用其他样本构建的基线，再乘以该样本动作概率的对数梯度，最后对所有样本的结果进行平均，以此来估计策略梯度。

### 2. 算法步骤

RLOO 算法的实现步骤在 REINFORCE 算法基础上有所扩展：

(1) **初始化策略参数**：同 REINFORCE 算法，随机初始化策略网络参数 $\theta$。

(2) **采样多组轨迹**：使用当前策略 $\pi_\theta(a|s)$ 与环境交互，每次收集 $k$ 条轨迹（即 $k$ 个样本），得到多组样本集 $\{(y_{(1)}^{(m)},\cdots,y_{(k)}^{(m)})\}_{m=1}^{M}$，其中 $m$ 表示组数，$M$ 为总的组数。

(3) **计算 RLOO 基线和梯度估计**：对于每组样本 $(y_{(1)}^{(m)},\cdots,y_{(k)}^{(m)})$，为每个 $y_{(i)}^{(m)}$ 计算 RLOO 基线 $\frac{1}{k-1}\sum_{j\neq i}R(y_{(j)}^{(m)},x)$，并计算相应的策略梯度估计值：

$$\hat{\nabla}_\theta J(\theta)_m=\frac{1}{k}\sum_{i=1}^{k}\left[R\left(y_{(i)}^{(m)},x\right)-\frac{1}{k-1}\sum_{j\neq i}R\left(y_{(j)}^{(m)},x\right)\right]\nabla\log\pi\left(y_{(i)}^{(m)}|x\right) \tag{6.31}$$

(4) **更新参数**：将多组样本的梯度估计值进行平均，得到最终的梯度估计值，然后沿梯度方向更新策略参数：

$$\hat{\nabla}_\theta J(\theta)=\frac{1}{M}\sum_{m=1}^{M}\hat{\nabla}_\theta J(\theta)_m \tag{6.32}$$

$$\theta\leftarrow\theta+\alpha\hat{\nabla}_\theta J(\theta) \tag{6.33}$$

其中 $\alpha$ 为学习率。

(5) **重复迭代**：重复步骤 2 - 4 直至策略收敛。

### 3. 与 REINFORCE 算法对比

与 REINFORCE 算法对比，RLOO 算法具备如下特点：

(1) **方差降低效果**：REINFORCE 算法使用简单基线（如移动平均基线），在降低方差方面效果有限。而 RLOO 通过利用多个样本构建动态基线，能更有效地降低梯度估计的方差。例如，在实验中，RLOO 在相同训练条件下，其奖励方差明显低于 REINFORCE 算法，这使得 RLOO 在优化过程中更加稳定，能够更快地收敛到较优的策略。

(2) **样本利用效率**：REINFORCE 算法在更新策略时，每个样本主要用于自身的梯度计算，样本之间的信息利用不足。RLOO 则充分利用了多个样本之间的关系，每个样本不仅用于自身的梯度计算，还参与构建其他样本的基线，大大提高了样本的利用效率。实验表明，在相同采样预算下，RLOO 能够实现更好的优化效果，如在多个数据集和模型上的实验显示，RLOO 在胜率和奖励优化方面均优于 REINFORCE 算法。

(3) **计算复杂度**：虽然 RLOO 在样本利用和方差降低上具有优势，但它的计算复杂度相对 REINFORCE 算法有所增加。在构建基线时，RLOO 需要对每个样本进行 $k-1$ 次奖励求和操作，随着样本数量 $k$ 的增加，计算量会相应增大。不过，在实际应用中，由于其在性能上的显著提升，这种计算复杂度的增加在可接受范围内。

#### 4. 算法特性分析

RLOO 算法在继承 REINFORCE 算法优点的同时，有效改进了其部分缺陷。它通过多样本构建基线的方式，降低了梯度估计的方差，提高了策略更新的稳定性和准确性，使得算法在复杂环境和大规模任务中表现更优。然而，RLOO 算法也并非完美无缺。在处理大规模样本时，其计算复杂度的增加可能会成为限制因素，需要消耗更多的计算资源和时间。此外，RLOO 算法对样本的独立性假设较为依赖，如果样本之间存在较强的相关性，可能会影响基线的有效性，进而影响算法性能。在实际应用中，需要根据具体问题的特点和资源情况，合理选择是否使用 RLOO 算法。

### 6.2.6 GRPO

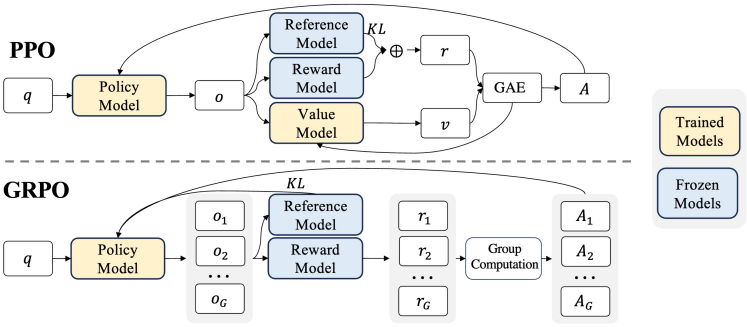


图 6.2 GRPO 算法流程图（需要重画）

Group Relative Policy Optimization（GRPO）是一种基于近端策略优化算法改进而来的优化算法，旨在解决传统 PPO 在计算资源和训练稳定性方面的问题。它通过创新的组奖励机制来估计基线，在不依赖独立价值模型的情况下实现高效训练，尤其适用于大型模型的优化。

### 1. 算法概述

传统的近端策略优化算法在训练过程中依赖独立的价值模型来估计奖励和减少方差。然而，这种方式在处理大型模型时会带来较高的计算成本和内存消耗。GRPO 则另辟蹊径，它不再使用独立的价值模型，而是通过组奖励来估计基线。具体来说，GRPO 从旧策略中抽取多个输出（形成组），利用组内奖励信息计算优势值，以此优化策略。这种方法避免了对每个样本都进行独立基线计算，大大减少了训练资源的消耗，在提升计算效率的同时，增强了训练过程的稳定性。

### 2. 算法原理

GRPO 的核心在于其优化目标函数的设计。目标函数 $J_{\text{GRPO}}(\theta)$ 旨在最大化策略的期望奖励，同时控制策略的变化幅度，确保训练的稳定性：

$$
\begin{aligned}
J_{\text{GRPO}}(\theta) =& \mathbb{E}_{q\sim P(Q),\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(O|q)}\Big[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\Big[\min\Big(\frac{\pi_\theta(o_{i,t}|q,o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|q,o_{i,<t})}\hat{A}_{i,t},\\
&\text{clip}\left(\frac{\pi_\theta(o_{i,t}|q,o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|q,o_{i,<t})},1-\epsilon,1+\epsilon\right)\hat{A}_{i,t}\Big)\Big]-\beta D_{KL}[\pi_\theta||\pi_{\text{ref}}]\Big]
\end{aligned}
\tag{6.34}
$$

在这个公式中 $\pi_\theta$ 代表当前正在优化的策略模型，其参数为 $\theta$，$\pi_{\theta_{\text{old}}}$ 是旧的策略模型，用于提供参考和对比。$G$ 表示组大小，即从旧策略 $\pi_{\theta_{\text{old}}}$ 中抽取的多个输出 $o_i$ 的数量。每个 $o_i$ 都是一个完整的输出序列，$|o_i|$ 表示序列 $o_i$ 的长度。$\hat{A}_{i,t}$ 是基于组内奖励计算得到的优势值，它衡量了在时间步 $t$ 采取动作 $o_{i,t}$ 相对于平均水平的优势程度，用于指导策略的更新。$\epsilon$ 和 $\beta$ 是超参数。$\epsilon$ 用于控制梯度剪切，防止策略更新幅度过大导致不稳定，$\beta$ 则控制 KL 散度 $D_{KL}$ 的权重，$D_{KL}[\pi_\theta||\pi_{\text{ref}}]$ 用于约束当前策略 $\pi_\theta$ 和参考策略 $\pi_{\text{ref}}$ 之间的差异，确保策略不会偏离参考策略太远。

通过对这个目标函数的优化，GRPO 能够在利用组内奖励信息的同时，平衡策略的探索与利用，实现高效稳定的训练。

### 3. 算法步骤

如图6.2所示，GRPO 算法实施的流程如下：

(1) **初始化策略参数**：随机初始化当前策略模型 $\pi_\theta$ 的参数 $\theta$ 以及旧策略模型 $\pi_{\theta_{\text{old}}}$ 的参数（通常初始值与 $\pi_\theta$ 相同）。

(2) **抽取组样本**：从分布 $P(Q)$ 中采样问题 $q$，然后根据旧策略 $\pi_{\theta_{\text{old}}}(O|q)$ 为每个问题 $q$ 抽取 $G$ 个输出 $\{o_i\}_{i=1}^{G}$。

(3) **计算优势值和目标函数**：对于每个输出 $o_i$ 的每个时间步 $t$，计算优势值 $\hat{A}_{i,t}$，并根据目标函

数 $J_{\text{GRPO}}(\theta)$ 的公式计算相应的项。在计算过程中，会用到当前策略 $\pi_\theta$ 和旧策略 $\pi_{\theta_{\text{old}}}$ 对动作的概率估计。

(4) **更新策略参数**：通过优化目标函数 $J_{\text{GRPO}}(\theta)$，计算梯度并更新当前策略模型 $\pi_\theta$ 的参数 $\theta$。通常使用随机梯度下降（SGD）或其变种算法来进行参数更新。

(5) **更新旧策略**：将更新后的当前策略 $\pi_\theta$ 的参数复制给旧策略模型 $\pi_{\theta_{\text{old}}}$，为下一轮迭代做准备。

(6) **重复迭代**重复步骤 2 - 5，直到达到预设的训练轮数、策略收敛或满足其他停止条件。

#### 4. 与 PPO 的对比

PPO 算法通过价值函数来估计奖励，并使用优势函数减少方差，其目标函数为：

$$J_{\text{PPO}}(\theta) = \mathbb{E}_{q\sim P(Q),o\sim\pi_{\theta_{\text{old}}}(O|q)}\left[\min\left(\frac{\pi_\theta(o|q)}{\pi_{\theta_{\text{old}}}(o|q)}A, \text{clip}\left(\frac{\pi_\theta(o|q)}{\pi_{\theta_{\text{old}}}(o|q)}, 1-\epsilon, 1+\epsilon\right)A\right)\right]$$

在这个公式中，依赖一个单独训练的价值函数来计算优势函数 $A$。而 GRPO 与之不同：

(1) **计算负担方面**：PPO 需要单独训练价值模型（critic），这增加了计算的复杂性和资源消耗。GRPO 则避免了这一过程，通过组内奖励估计直接计算优势值，减少了计算开销，在处理大型模型时优势明显。

(2) **基线估计效率**：PPO 对每个样本独立计算基线，在样本数量较大时效率较低。GRPO 通过分组计算奖励，避免了这种独立计算的问题，提高了基线估计的效率。

(3) **训练稳定性**：PPO 的优化依赖单个样本的奖励和基线计算，容易受到单一奖励样本的影响，导致方差较高。GRPO 通过优化组内奖励，减少了这种高方差的影响，使得训练更加稳定。

#### 5. 算法特性分析

GRPO 在计算效率、稳定性等方面具有显著优势：

(1) **计算资源友好**：减少了对独立价值模型的依赖，降低了计算复杂度和内存需求，使得在处理大型模型时能够更高效地利用计算资源，提升训练速度。

(2) **稳定性提升**：基于组奖励的优化方式降低了训练过程中的方差，使得策略更新更加稳定，有利于模型收敛到更优的策略。

(3) **应用效果良好**：在数学推理等任务中表现出色，如 DeepSeekMath 模型引入 GRPO 后，在 GSM8K 和 MATH 等数学基准测试中性能显著提升。

## 6.3 推理模型的强化学习

### 6.3.1 DeepSeek-R1

大语言模型发展过程中，提升推理能力是关键研究方向。OpenAI 的 o 系列模型率先通过增加思维链推理长度，在数学、编程和科学推理等任务中表现优异。然而，实现有效的测试时扩展，让

模型在不同场景高效运用推理能力，仍是学界和业界面临的挑战。此前研究尝试了多种方法，如基于过程的奖励模型、强化学习以及蒙特卡洛树搜索和波束搜索等搜索算法，但均未达到与 OpenAI o 系列模型相媲美的通用推理性能。在此背景下，DeepSeek 团队开展了基于纯强化学习提升模型推理能力的探索。

### 1. DeepSeek-R1-Zero：基于基座模型的强化学习

**1.1 强化学习算法** DeepSeek 的研究人员采用 GRPO 算法进行强化学习，该算法舍弃了传统 Actor-Critic 范式中与策略模型规模相当的 critic 模型，通过从一组得分估计基线来优化策略模型。通过这种方式，能够提高强化学习的效率，有利于大规模强化学习的开展。

**1.2 奖励建模** 采用基于规则的奖励系统，包含两种奖励类型：

- 准确性奖励：用于评估模型响应的正确性。对于有确定性答案的数学问题，要求模型按指定格式输出最终答案以便验证；对于 LeetCode 编程问题，利用编译器根据预定义测试用例生成反馈。
- 格式奖励：促使模型将思考过程置于‘<think>’和‘</think>’标签之间，确保推理过程清晰呈现。

不使用结果或过程神经奖励模型，因其在大规模强化学习中可能出现奖励黑客问题，且重新训练奖励模型会增加计算资源需求并使训练流程复杂化。

**1.3 训练模板** 设计简单训练模板，要求 DeepSeek-R1-Zero 先产生推理过程，再给出最终答案。模板为：用户提出问题，助手先在脑海中思考推理过程，然后提供答案，推理过程和答案分别包含在 <think> </think> 和 <answer> </answer> 标签内，训练时 prompt 会被具体推理问题替换。通过这种模板，在避免内容特定偏差的同时，引导模型遵循指定结构进行推理，便于观察模型在强化学习过程中的自然发展。

**1.4 性能、自我进化过程与顿悟时刻**

- 性能：在 AIME 2024 基准测试中，DeepSeek-R1-Zero 的平均 pass@1 分数从初始的 15.6% 显著提升至 71.0%，达到与 OpenAI-o1-0912 相当的性能水平。使用多数投票后，分数进一步提升至 86.7%，超过 OpenAI-o1-0912。在其他推理相关基准测试中，如 MATH-500、GPQA Diamond 等，也展现出强大的推理能力，证明了强化学习算法对模型性能优化的有效性。
- 自我进化过程：训练过程中，模型的思考时间和生成回答的长度不断增加，这并非外部调整所致，而是模型在强化学习环境中自我改进的结果。随着测试时计算量的增加，模型自发出现复杂行为，如反思先前步骤、探索多种解题方法等，显著提升了推理能力，使其能够处理更具挑战性的任务。
- 顿悟时刻：训练过程中出现“aha moment”，模型在中间版本学会重新评估初始解题方法，分配更多思考时间，这一行为不仅体现了模型推理能力的提升，也展示了强化学习可带来意想不到的复杂结果，凸显了强化学习在激发模型智能方面的潜力。

Question: If $a > 1$, then the sum of the real solutions of $\sqrt{a-\sqrt{a+x}} = x$ is equal to

Response: <think>
To solve the equation $\sqrt{a-\sqrt{a+x}} = x$, let's start by squaring both $\cdots$
$\left(\sqrt{a-\sqrt{a+x}}\right)^2 = x^2 \implies a-\sqrt{a+x} = x^2$.
Rearrange to isolate the inner square root term:
$(a-x^2)^2 = a+x \implies a^2-2ax^2+(x^2)^2 = a+x \implies x^4-2ax^2-x+(a^2-a) = 0$
$\cdots$
Wait, wait. Wait. That's an aha moment I can flag here.
Let's reevaluate this step-by-step to identify if the correct sum can be $\cdots$
We started with the equation:
$\sqrt{a-\sqrt{a+x}} = x$
First, let's square both sides:
$a-\sqrt{a+x} = x^2 \implies \sqrt{a+x} = a-x^2$
Next, I could square both sides again, treating the equation: $\cdots$
$\cdots$

图 6.3 Aha moment（需要重画）

**1.5 存在的问题** 尽管 DeepSeek-R1-Zero 展现出强大的推理能力，但存在可读性差和语言混合等问题。其生成内容可能包含多种语言，且缺乏便于用户阅读的格式，这限制了其在实际应用中的推广，促使研究人员进一步探索改进方案，从而引出 DeepSeek-R1 模型。

### 2. DeepSeek-R1：冷启动强化学习

**2.1 冷启动** 为解决 DeepSeek-R1-Zero 训练初期不稳定问题，DeepSeek-R1 构建并收集少量长思维链数据对 DeepSeek-V3-Base 模型进行微调，作为初始 RL 模型。数据收集方法多样，包括基于长 CoT 的少样本提示、直接引导模型生成带反思和验证的详细答案、收集 DeepSeek-R1-Zero 的可读输出并经人工标注后处理等。冷启动数据具有明显优势：一方面，改善了输出的可读性，通过设计特定输出格式，在每个回答末尾添加总结，并过滤不友好内容；另一方面，融入人类先验知识，提升了模型的性能潜力，为后续强化学习训练奠定更好基础。

**2.2 面向推理的强化学习** 在冷启动微调后，采用与 DeepSeek-R1-Zero 相同的大规模强化学习训练过程，聚焦于编码、数学、科学和逻辑推理等推理密集型任务。针对训练中发现的 CoT 语言混合问题，引入语言一致性奖励，根据 CoT 中目标语言单词的比例计算。虽然消融实验表明该奖励会使模型性能略有下降，但为提升生成内容的可读性，仍将其与推理任务准确性奖励直接相加作为最终奖励，持续训练模型直至推理任务收敛。

**2.3 拒绝采样和监督微调** 当面向推理的 RL 训练接近收敛时，利用此时的检查点收集用于后续轮次的监督微调数据。

- 推理数据：通过拒绝采样生成推理轨迹，扩展数据集。除基于规则奖励评估的数据外，纳入部分使用生成奖励模型评估的数据，将真实标注和模型预测输入 DeepSeek-V3 进行判断。同时，过滤掉语言混合、长段落和代码块等难以阅读的思维链，每个 prompt 采样多个回答并

仅保留正确答案，共收集约 600k 推理相关训练样本。

- 非推理数据：对于写作、事实性问答、自我认知和翻译等非推理任务，复用 DeepSeek-V3 的 pipeline 和部分 SFT 数据集。针对某些任务，先调用 DeepSeek-V3 生成潜在思维链再回答问题；对于简单查询，如“hello”，则不提供 CoT。最终收集约 200k 非推理训练样本。使用这约 800k 样本对 DeepSeek-V3-Base 进行两轮微调。

**2.4 全场景强化学习** 为使模型更好地符合人类偏好，进行二次强化学习训练，旨在提升模型的有用性、无害性并进一步优化推理能力。对于推理数据，沿用 DeepSeek-R1-Zero 的方法，基于规则奖励引导学习；对于通用数据，采用奖励模型捕捉复杂场景下的人类偏好，构建类似 DeepSeek-V3 的偏好对和训练提示分布。评估有用性时，专注于最终总结，确保回答对用户实用且相关；评估无害性时，审查模型的整个回答，识别并消除潜在风险、偏见和有害内容，使模型在推理能力突出的同时，更符合用户需求和安全标准。

#### 3. 蒸馏：赋予小模型推理能力

为使更小的模型具备类似 DeepSeek-R1 的推理能力，使用在 DeepSeek-R1 训练过程中收集的 800k 样本，对 Qwen 和 Llama 等开源模型进行直接微调。实验发现，这种简单的蒸馏方法能显著提升小模型的推理能力。在实验中，选择 Qwen2.5-Math-1.5B、Qwen2.5-Math-7B 等多种模型作为基础模型，仅对蒸馏模型进行 SFT，未引入 RL 阶段，以突出蒸馏技术的有效性，后续 RL 阶段的探索留给研究社区。结果显示，蒸馏后的小模型在多个推理基准测试中表现优异，如 DeepSeek-R1-Distill-Qwen-7B 在 AIME 2024 上的成绩超越了部分强大的基线模型。

#### 4. 总结

**4.1 强化学习训练创新** DeepSeek-R1-Zero 首次验证了大语言模型的推理能力可通过纯强化学习激发，无需监督微调作为前期步骤。这种创新训练方式使模型能够自主探索思维链以解决复杂问题，展现出自我验证、反思和生成长思维链等能力，为大语言模型推理能力提升开辟了新路径，推动了相关领域的研究发展。

**4.2 模型性能卓越** DeepSeek-R1 在多个推理任务中表现出色，在 AIME 2024 上 Pass@1 得分达到 79.8%，略超 OpenAI-o1-1217；在 MATH-500 上得分高达 97.3%，与 OpenAI-o1-1217 相当且远超其他模型。在编码相关任务中，于 Codeforces 竞赛中获得 2029 Elo 评级，超越 96.3% 的人类参与者；在知识类基准测试如 MMLU、GPQA Diamond 等任务中，也取得了优异成绩，展现出强大的知识掌握和推理应用能力。

### 6.3.2 Kimi k1.5

基于下一个 token 预测的语言模型预训练遵循缩放定律，即按比例增加模型参数和数据规模可提升模型智能程度。然而，这种方法严重依赖高质量训练数据的数量。在实际应用中，可用的高质量数据往往有限，这限制了模型性能的进一步提升。将强化学习与大语言模型相结合，有望

解决数据受限的问题。大语言模型可通过强化学习中的奖励机制，学习如何探索不同的推理路径，从而扩大训练数据的范围。但此前的相关研究成果未达到理想的竞争效果，Kimi k1.5 旨在探索一条更有效的技术路线。

### 1. 技术路线

**强化学习提示数据集构建：** RL 提示数据集的质量和多样性对强化学习的有效性至关重要。Kimi k1.5 定义了高质量 RL 提示集的三个关键属性：

多样覆盖：提示应涵盖 STEM、编程和一般推理等广泛学科，以增强模型的适应性和跨领域应用能力。为此，采用自动过滤器选择需要丰富推理且易于评估的问题，数据集来源广泛，包括不同领域的问题以及纯文本和图像 - 文本问答数据。

平衡难度：提示集应包含不同难度级别的问题，以促进模型的逐步学习并防止过度拟合。利用模型自身能力自适应评估提示难度，通过 SFT 模型多次生成答案计算通过率作为难度代理指标，并开发标签系统按领域和学科分类提示，实现难度平衡。

准确评估能力：提示应能被验证者客观可靠地评估，确保基于正确推理衡量模型性能。为避免奖励操纵，排除易出现验证错误的问题类型，并通过特定方法识别和移除易被操纵的提示。

此外，为提高模型的图像推理能力，数据还来源于现实世界数据、合成视觉推理数据和文本渲染数据这三个类别。

**预训练数据集的构建与处理：** Kimi k1.5 的预训练数据集涵盖英语、中文、代码、数学与推理以及知识数据五个领域，以确保数据多样性。为保证数据高质量，采用多种清洗方法：

针对英语和中文文本数据，建立多维质量过滤框架，包括基于规则的过滤、基于 FastText 的分类、基于嵌入的相似性分析和基于大模型的质量评估，最后通过动态采样率对不同质量的文档进行处理。

对于代码数据，对纯代码数据和文本-代码交错数据分别进行处理，前者遵循 BigCode 方法进行预处理和采样调整，后者采用基于句向量的方法召回高质量数据。数学与推理数据通过开发专门的数据清洗程序和 OCR 模型，以及两阶段数据清洗过程，提高数据质量。知识数据通过精心策划，利用内部语言模型添加多维标签，并实施复杂的过滤和采样管道，优化数据组成。

**微调数据集的构建：** Kimi k1.5 的 SFT 数据集包含约 100 万个文本示例，涵盖多种任务类型，如一般问答、编程、数学和科学等。此外，还构建了 100 万个文本 - 视觉示例，涵盖图表解读、OCR 等多种类别。

**多模态数据：**作为多模态模型，Kimi k1.5 的多模态数据包括字幕、图像-文本交错数据、OCR、知识和一般问题回答五类。对每类数据进行了针对性处理：

标题数据整合开源和内部数据，并严格限制合成数据比例，同时进行质量控制和图像分辨率调整。图像-文本交错数据则考虑开源数据集并构建自建数据，还通过数据重排序确保图像和文本顺序正确。OCR 数据来源多样，包括公开数据和自建数据集，并进行数据增强以提高模型的 OCR 能力。

### 2. 算法创新

**长上下文扩展：** Kimi k1.5 将 RL 的上下文窗口扩展到 128k，实验表明，上下文长度与模型解决问题的能力强相关，增加上下文长度可提升模型在困难推理基准测试中的性能。为解决长上下文带来的计算量增加问题，采用部分回放（partial rollouts）技术，通过重用之前轨迹的大部分来采样新轨迹，减少计算开销。具体操作时，部分展开系统将长响应分解为多个段，在多个迭代中逐步处理，加快训练速度。

**Long2short 的上下文压缩策略**

长上下文模型虽性能强大，但测试时 token 消耗较多。Kimi k1.5 提出多种方法将长上下文模型的思维先验转移到短上下文模型，以提高短上下文模型的性能：

1. 模型合并：通过简单平均长上下文模型和短上下文模型的权重，获得无需训练的新模型，有助于保持泛化能力。

2. 最短拒绝采样：基于模型对同一问题生成响应长度变化大的特点，对同一问题多次采样，选择最短的正确响应。

3. DPO：利用长上下文模型生成多个响应样本，将最短正确解决方案作为正样本，较长响应作为负样本（包括错误长响应和正确但超长响应），形成成对偏好数据用于 DPO 训练，DPO 细节参见下文 [245]。

4. 长到短强化学习：在标准 RL 训练后，选择性能和 token 效率平衡最佳的模型作为基础模型，进行单独的长到短 RL 训练，应用长度惩罚方案惩罚超长响应。

**改进的策略优化：** Kimi k1.5 推导出带有长推理链的强化学习公式，并采用在线镜像下降的变体进行策略优化。该算法通过以下方式进一步改进：

1. 采样策略：采用课程采样和优先级采样策略。课程采样从简单任务开始训练，逐渐过渡到困难任务，利用数据的难度标签提高训练效率；优先级采样跟踪每个问题的成功率，按比例采样问题，使模型专注于薄弱领域。

2. 长度惩罚：针对 RL 训练期间模型响应长度增加的问题，引入长度奖励限制 token 长度增长。在正确答案中提倡简短回答并惩罚较长回答，对错误答案的长回答明确惩罚。为缓解长度惩罚在训练初期对训练速度的影响，采用逐渐增加长度惩罚的方式。

### 3. 训练架构及工程框架

Kimi k1.5 模型的训练分为三个阶段：

1. 视觉语言预训练阶段：模型最初仅在语言数据上训练，建立语言基础，随后逐步引入交错式视觉 - 语言数据，获取多模态能力。视觉塔先独立训练，之后解除语言模型层的冻结并增加视觉 - 文本数据比例。

2. 视觉语言冷却阶段：模型继续使用高质量的语言和视觉语言数据集训练，加入合成数据可显著提高在数学推理、基于知识的任务和代码生成方面的性能。