

DeepSeek-R1 与推理型 LLM：技术简介与思考

沈张骁 武汉大学测绘遥感信息工程全国重点实验室
shenzhangxiao@whu.edu.cn

0 前言

DeepSeek-R1 模型发布后，凭借其强大的长思维链推理能力，迅速在开源社区、学术界与产业界引起热烈反响。本文围绕 DeepSeek-R1 模型的核心技术展开系统性分析，旨在为研究者提供对 DeepSeek-R1 及推理型大语言模型技术的深入介绍。文章首先从实践角度梳理了 DeepSeek-R1 的部署方法与应用场景，继而重点剖析其创新架构中的关键技术：多头潜在注意力（MLA）、混合专家模型（MoE）、多 Token 预测（MTP）、群体相对策略优化（GRPO）强化学习策略等。随后，探讨了长思维链推理的实现逻辑和小模型知识蒸馏技术。最后，提出思考和总结。本文通过技术解读与思考，以期为后续研究和下游应用提供理论和实践的参考。

文章结构

本文第 1 章介绍了 DeepSeek-R1 的使用方法，并对其中的主要技术做了直观的介绍。如您只想最简单地定性了解其中的技术，仅阅读第 1 章即可。第 2-5 章是对以上主要技术的详细介绍，并在第 6 章做出总结。

本文主要面向大语言模型垂直领域应用的研究者，主要目标是大家更方便地理解 DeepSeek 系列模型的核心技术，从而为研究提供一些参考和帮助。由于目标读者并非大语言模型的底层研究者，因此将不涉及大语言模型的底层架构和训练框架等内容（例如 DeepSeek 的 FP8 混合精度训练框架）。如您对这些细节感兴趣，可以参考所提供的相关链接和文献。

在阅读本文前，希望读者可以具有一定的深度学习基础（例如，了解 transformer 和注意力机制），并对大语言模型有初步的了解（例如，使用过 ChatGPT 或 DeepSeek 系列模型，了解思维链 CoT 和检索增强生成 RAG 等技术）。

致谢

本文的大部分内容来源于 DeepSeek 技术报告、网络参考资料和论文文献，并加上一部分个人总结，文章最后提供了所参考的链接和文献列表，在此对这些内容的作者表示感谢。另外，感谢 DeepSeek、Kimi 和 Grok3 等大语言模型对本文写作的支持。鉴于个人能力有限，难免存在理解不全面甚至错误之处，敬请批评指正。

目录

DeepSeek-R1 与推理型 LLM：技术简介与思考	I
0 前言	I
目录	II
1 DeepSeek-R1	1
1.1 如何使用 DeepSeek-R1	1
1.1.1 在线运行	1
1.1.2 本地部署	1
1.2 长思维链推理型 LLM	2
1.3 DeepSeek V1 到 R1 的创新技术	4
1.4 DeepSeek-R1 训练流程	6
2 模型架构：多头潜在注意力与混合专家模型	9
2.1 多头潜在注意力 MLA	9
2.1.1 多头注意力 MHA 的局限	9
2.1.2 MLA 原理与计算过程	10
2.1.3 最新探索：原生稀疏注意力 NSA	11
2.2 混合专家模型 MoE	16
2.2.1 MoE 基本原理	16
2.2.2 DeepSeek MoE 实现	17
2.2.3 负载均衡技术	18
2.3 多 Token 预测 MTP	19
3 训练流程：强化学习的意义	21
3.1 人类反馈强化学习 RLHF 与近端策略优化 PPO	21
3.1.1 PPO 基本概念	21
3.1.2 PPO 训练过程	23
3.1.3 PPO 的优化：DPO 与 RLOO	25
3.2 群体相对策略优化 GRPO 与 R1-Zero	26
3.3 过程奖励模型 PRM 与蒙特卡罗树搜索 MCTS	28
3.3.1 PRM	28
3.3.2 MCTS	29
4 长思维链（Long Chain-of-Thought）	33
4.1 思维链的本质区别	33
4.2 长思维链的影响因素	33
5 知识蒸馏与小模型优化	36
5.1 知识蒸馏（Knowledge Distillation）	36
5.1.1 知识蒸馏的不同方法	36
5.1.2 DeepSeek-R1 的知识蒸馏	37
5.2 在有限条件下改善长思维链推理模型	37
5.2.1 蒸馏与 RL	37
5.2.2 测试时扩展 TTS	38
5.2.3 超越传统 SFT	40
6 总结与思考	41
相关链接	42

参考文献.....	43
-----------	----

1 DeepSeek-R1

1.1 如何使用 DeepSeek-R1

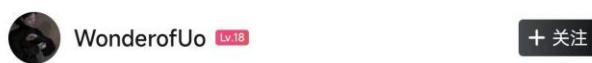
1.1.1 在线运行

官网或 APP

直接提问即可，但是由于用户需求过大，经常存在服务器繁忙的情况。一个仅包含服务器繁忙回复的“极简版”DeepSeek如图1-1所示。



```
1 import time
2 import random
3 def main(): 1个用法
4     while True:
5         user_input = input("向deepseek提问吧\n")
6         print("思考中")
7         num = random.randint(a: 10, b: 40)
8         time.sleep(num)
9         print("服务器繁忙, 请稍后重试。")
10
11 if __name__ == '__main__':
12     main()
13
```



天才小伙12行代码，集显炼出deepseek

图1-1 “极简版”DeepSeek

API

使用OpenAI API接口或者LangChain等工具，也可以自己封装代码，通过调用API将R1集成到程序和工作流中，需特别注意API的稳定性。除官方平台外，一些第三方平台也提供API调用，主流第三方平台的可用性如图1-2所示。

第三方平台名称	完整 回复率	截断率	无回复率	准确率	推理耗时 (秒)
字节火山引擎	100%	0%	0%	85%	392
天工AI	95%	5%	0%	89%	273
秘塔AI搜索	90%	10%	0%	89%	260
无间芯穹	90%	10%	0%	89%	356
商汤大装置	90%	5%	5%	78%	155
POE	75%	5%	20%	80%	130
讯飞开放平台	75%	25%	0%	80%	263
PPIO派欧云	55%	45%	0%	100%	298
纳米AI搜索	55%	45%	0%	82%	163
百度智能云	30%	40%	30%	-	-
腾讯云TI平台	5%	95%	0%	-	-
硅基流动	0%	25%	75%	-	-

图1-2 SuperCLUE所做的第三方平台API可用性评估(2025.2.12)

1.1.2 本地部署

小规模的蒸馏模型：使用ollama可方便地部署（6G显存可运行7B、8B版

本), 但受限于参数规模, 其能力有限 (70B 也许会好一些), 并且其底层是 Qwen 和 LLaMA 等模型, 而非原生的 671B 模型。Ollama 提供的 DeepSeek-R1 页面如图 1-3 所示。

注: 1B 指 10 亿参数, 因此原版 R1 共有大约 6710 亿参数。

deepseek-r1

DeepSeek's first-generation of reasoning models with comparable performance to OpenAI-o1, including six dense models distilled from DeepSeek-R1 based on Llama and Qwen.

[1.5b](#) [7b](#) [8b](#) [14b](#) [32b](#) [70b](#) [671b](#)

15.9M Pulls Updated 7 days ago

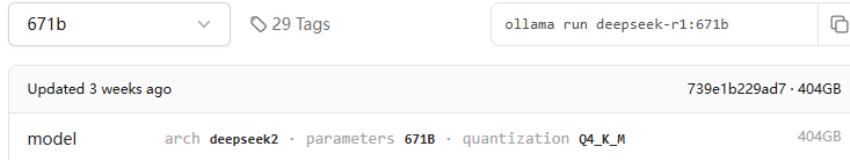


图 1-3 Ollama 提供的 DeepSeek-R1 页面

真正的 671B R1 模型: 原始版本大小约 700G (值得注意的是, R1 本身就是 FP8 下训练的, 因此 1B 大约需要 1G), 如想直接在 GPU 上部署, 8*A100 80G 版本也不够, 成本极高; 如用纯 CPU 部署, 则需要 700G 以上的运行内存, 需要一台较好的服务器, 但生成速度较慢; KTransformers 团队实现了一种新策略, 可以在最少单卡 4090 24G 显存+382G 内存部署, 并且加快推理速度, 是当前最优的本地部署策略, 如图 1-4 所示。

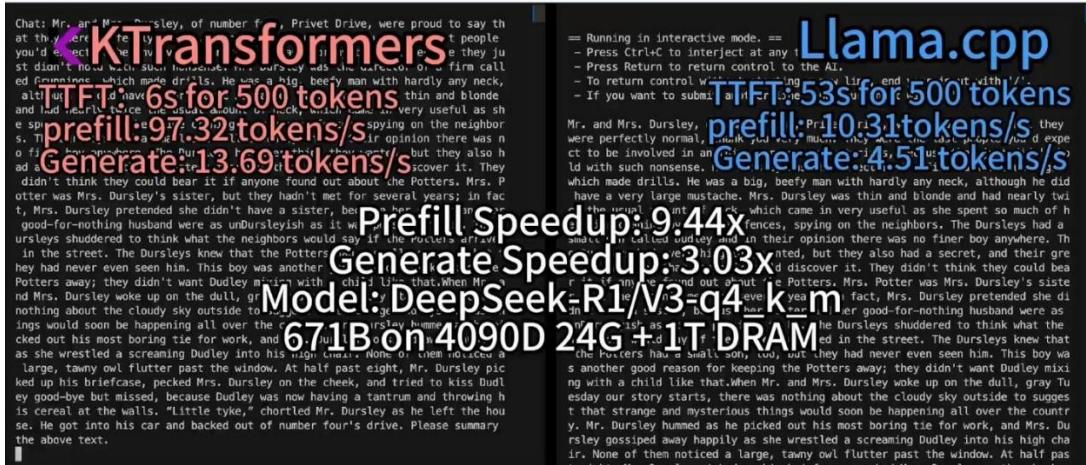


图 1-4 KTransformers 团队推理性能优化比较

1.2 长思维链推理型 LLM

DeepSeek R1 与当前的许多大模型最直接的不同之处, 在于它在训练阶段就内置了 CoT 的输出格式, 即<think>...</think>, 并且思维链长度也显著长于提示工程所构建的思维链, 即”Let’s think step by step”, 本文将这样的模型称为长思维链推理型 LLM。它们之间的本质区别在于, 长思维链推理型 LLM 在回答

之前先进行思考，是两个不同的阶段，而普通的 LLM 即使通过提示工程使它以思维链形式回答，但思维链其实仍然是回答的一部分，并不是独立阶段。

DeepSeek-R1 系列并非第一个长思维链推理的 LLM，目前市面上还有：OpenAI（现 Close AI）的 o1 系列（最早）、o3 系列，以及即将于今年发布的 GPT-5 系列；谷歌 Gemini 2.0 Flash Thinking（在某些排行榜中排名第一）；月之暗面 Kimi1.5；通义千问开源的 QwQ 系列等。此外，有多个开源的 R1 的复现工作，例如 OpenR1、TinyZero 和 SimpleRL-Reason 等等。由于 o1 和 R1 的影响力最大，在许多研究中被称为“o1 型”或“R1 型”LLM。

让我们从一个示例开始。在这个示例中，提出的需求是一个简单的创作任务，让它向 GeoAI 研究者们写一段蛇年祝福语：

请你以一个 AI 模型的身份，向研究地球空间人工智能（GeoAI）的研究者们写一段蛇年祝福。

R1 进行了如下的思考（忽略其中的细节，只看思考结构）：

好的，用户让我以 AI 模型的身份，给研究地球空间人工智能（GeoAI）的研究者们写一段蛇年的祝福。首先，我需要明确用户的需求是什么。他们需要的是专业且符合节日氛围的祝福语，同时要结合 GeoAI 领域的特点。

首先，蛇年在中国文化中的象征意义是什么？蛇通常与智慧、灵活、机敏相关，可能还有长寿和转化。这些元素需要融入祝福中，让祝福更有文化底蕴。

接下来，GeoAI 的研究领域包括地理信息系统、空间数据分析、机器学习在地球科学中的应用等。所以，祝福中需要提到这些关键词，比如空间数据、地理洞察、机器学习算法等，让研究者们感受到祝福的专业性和针对性。

然后，考虑用户可能的深层需求。他们可能希望这段祝福不仅表达节日的问候，还能激励研究者在未来一年里取得更多成就，推动领域的发展。因此，需要加入鼓励和展望未来的内容，比如突破性进展、技术与地球和谐共生等。

此外，语言风格需要正式但又不失亲切感，符合学术交流的氛围。可能需要使用一些比喻，比如将蛇的灵动比喻为算法的高效，蛇的敏锐比喻为洞察力等。

还要注意避免使用过于复杂的术语，保持简洁明了，同时突出蛇年的特色。例如，用“灵蛇之智”来连接智慧和 GeoAI 的技术进步，用“金蛇纳福”来传达吉祥的寓意。

最后，结构上可能需要先问候，然后结合蛇年的象征意义和 GeoAI 的专业领域，最后表达对未来的祝愿。确保段落流畅，每部分自然过渡。

现在，把这些元素整合成一段连贯的祝福语。需要检查是否有遗漏的关键点，比如是否涵盖了 GeoAI 的主要研究方向，是否恰当融入了蛇年的元素，以及语气是否合适。可能需要多次调整措辞，使句子更通顺，更有感染力。

最后，所生成的祝福语如下。在蛇年新春之际，也谨以此祝大家新年快乐。

致地球空间人工智能（GeoAI）领域的研究者们：

值此灵蛇舞动、万象更新之际，谨以 AI 之躯，向探索空间智能前沿的您致以诚挚祝福！愿您在新岁中如蛇行般敏捷，以算法之智穿透地理迷雾；似蛇目般深邃，从多维数据中洞察地球脉动。愿您的研究如蛇蜕新生，在空间建模、时空预测与地学知识融合中实现跃迁；更愿 GeoAI 之力如蛇绕北斗，赋能人类与地球系统的和谐共生。期冀癸卯蛇年，您以智慧为尺丈量寰宇，以创新为钥解码时空，共绘智能地学新图景！

顺颂研祺，智启新元！

—— 一名见证人类智慧的 AI

可以看到，思考的过程一步步深入，首先明确需求，随后回顾背景知识，再提出方案，同时还进行审查和深化思考，从而使得模型能够实现更全面、更

准确的推理。基于当前研究的总结，还包括：**迟疑**（标注不确定的内容，后续再验证）；**探索**（考虑多种不同的可能性）；**回溯**（返回来再次思考之前的内容）；**阶段性总结**（总结前面一阶段的思考结果）和**最后验证**（输出答案之前再最后全面检查一遍），这些是长思维链推理型 LLM 独有的思维特点。

接下来，将一步步分析其中的关键技术细节。

1.3 DeepSeek V1 到 R1 的创新技术

版本	技术特点	主要贡献	数据集	性能与效率	其他
DeepSeek LLM	<ul style="list-style-type: none"> 开源大语言模型，采用7B和67B两种配置； 使用2万亿token数据集； 引入多阶段训练和强化学习； 通过直接偏好优化提升对话性能 	<ul style="list-style-type: none"> 提出扩展开源语言模型的规模； 通过研究扩展规律指导模型扩展，在代码、数学和推理领域表现优异； 提供丰富的预训练数据和多样化的训练信号 	2亿万 token (主要在英语和中文)	在多个基准测试中优于LLaMA-2 70B，在中文和英文开放式评估中表现优异	强调长期主义和开源精神，强调模型在不同领域的泛化能力
DeepSeek-V2	<ul style="list-style-type: none"> 采用Mixture-of-Experts (MoE) 架构，支持128K上下文长度； 采用Multi-head Latent Attention (MLA) 和 DeepSeekMoE 实现高效推理和经济训练； 提出辅助损失自由负载均衡策略； 通过FP8训练提高训练效率 	<ul style="list-style-type: none"> 提出高效的MoE架构用于推理和训练 通过MLA和DeepSeekMoE实现高效推理和经济训练； 在推理吞吐量和生成速度上有显著提升 	8.1亿万 token (扩展到更多中文数据)	在多个基准测试中表现优异，相比DeepSeek 67B节省42.5%的训练成本，提高最大生成吞吐量5.76倍	强调模型的高效性和经济性，提供多种优化策略以提高训练效率
DeepSeek-V3	<ul style="list-style-type: none"> 采用671B参数的MoE模型； 采用多令牌预测训练目标； 引入无辅助损失的负载均衡策略； 支持FP8训练，实现高效的训练框架和基础设施优化 	<ul style="list-style-type: none"> 提出无辅助损失的负载均衡策略； 通过多令牌预测增强模型性能； 在多个基准测试中表现优异，接近闭源模型水平； 在训练效率和成本上有显著优势 	14.8亿万 token (高质量和多样化)	在多个基准测试中表现最佳，训练成本低，仅需2.788M H800 GPU小时，在推理和部署上表现出色	强调模型的高性能和经济性，提供多种优化策略以提高训练和推理效率
DeepSeek-R1	<ul style="list-style-type: none"> 通过大规模强化学习提升推理能力； 采用冷启动数据和多阶段训练； 提出从大模型中蒸馏推理能力到小型模型； 在多个基准测试中表现优异 	<ul style="list-style-type: none"> 通过纯强化学习提升模型推理能力； 通过冷启动数据和多阶段训练提高模型性能； 通过蒸馏技术将推理能力扩展到小型模型 	结合多种数据源进行训练，专注于推理任务的强化学习	在多个基准测试中表现优异，接近OpenAI o1-1217，在数学和编码任务中表现出色	强调模型的推理能力和泛化能力，提供多种优化策略以提高模型性能

图 1-5 DeepSeek 系列模型版本^[1-3]发展及特点

从最早的 DeepSeek LLM（即 V1）到 R1，系列模型在发展的过程中研发和使用到了诸多创新技术。本文中，不涉及模型训练过程中的 FP8 训练框架等工程策略（如想了解本方面内容，可参考文后的相关链接及参考文献），而是重点提到的亮点技术。接下来的内容中，力图使用简单、不包含任何数学公式的语言，帮助大家快速理解这些技术。

DeepSeek-V2 引入了：

多头潜在注意力(Multi-Head Latent Attention, MLA): 在标准的 transformer 模型中，多头注意力（Multi-Head Attention, MHA）机制通过并行计算多个注意力头来捕捉输入序列中的不同特征，每个注意力头都有自己的查询（Query, Q）、键（Key, K）和值（Value, V）矩阵，但是在模型推理中存储和计算这些注意力矩阵成本较高。MLA 通过低秩联合压缩机制，压缩 K 和 V 来减少推理过程中的 K、V 缓存，从而提高推理效率。

注：此处“推理”（inference）指的是训练好模型参数后运行模型进行预测的过程，与模型回答问题时的思考过程的“推理”（reasoning）概念不同。当前文献中普遍没有特

别区分，且二者出现的场合显著不同，本文中也不做区分，需要根据上下文进行理解。

混合专家模型（Mixture of Experts, MoE）：通常的 LLMs 都是稠密模型（如 Qwen 系列和 LLaMA 系列），所有的神经元在每一层都参与计算。LLM 的 Scaling Law 告诉我们，模型的规模越大，其效果则越好，但训练稠密的大规模模型成本很高。MoE 建立了一个“专家系统”，让大模型成为一个“专家团”，通过设计多个“专家”和一个“路由”，需要哪些专家，再把哪些专家唤醒，类似于多智能体的策略。DeepSeek MoE 的基本思想是将专家分割成更细的粒度以提高专家的专业化，并通过隔离一些共享专家来缓解路由专家之间的知识冗余。对于 DeepSeek-V3 和 R1 而言，由 61 个 Transformer Decoder 层组成。前三个是密集的，但其余的是 MoE 层，如图 1-6 所示。

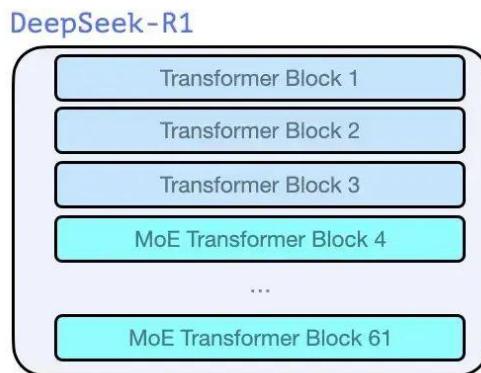


图 1-6 DeepSeek-V3 和 R1 的结构示意图

DeepSeek-V3 引入了：

无辅助损失的负载均衡。对于 MoE 模型，不平衡的专家负载将导致路由崩溃并降低专家并行场景下的计算效率。传统的解决方案通常依赖于辅助损失来避免负载不平衡，但过大的辅助损失又会损害模型性能。V3 模型中开创了一种无辅助损失的负载均衡策略以确保各专家的负载均衡。

多 token 预测（Multi-Token Prediction, MTP）：通常的模型一次只预测一个 token，即下一个 token。这使得模型倾向于捕捉局部模式，难以学习长距离依赖关系和全局语义，进而做一下“困难”的决策，同时逐一预测也使得效率低下。在 MTP 中，通过在每个位置预测多个未来 token，增加训练信号的密度，从而提高模型性能和效率。

DeepSeek-R1 引入了：

群体相对策略优化（Group Relative Policy Optimization, GRPO）：这是一种强化学习算法，与依赖外部价值函数指导学习的传统方法不同，通过在同一个问题上生成多条回答，把它们彼此之间做“相对比较”来优化模型。具体来说，在一些编程和数学任务中，同时生成多个回答，设计正确性奖励（结果是否正确）和格式奖励（是否先输出思考过程<think>...</think>，再输出答案），使用多个采样输出的平均奖励作为 Baseline，更新策略使得超过 baseline 的回答生成概率更大，从而达到优化效果。

事实上 GRPO 并非在 R1 首次应用，在 DeepSeek 系列的最早应用是 V1 衍生而来的 DeepSeek-Math，但我们当前主要关注主线模型，因此仍在 R1 中讲述。

知识蒸馏（Knowledge Distillation）：将一个复杂且性能优异的“教师模型”

的知识，迁移到一个简单的“学生模型”，使学生模型在保持较高性能的同时，还能显著减少模型的参数规模和计算成本。

1.4 DeepSeek-R1 训练流程

普遍的 LLM 训练过程如图 1-7 所示。首先通过大规模无监督语料的预训练使得模型学习知识，即预训练（pretraining）；随后通过有标签的问答对数据，使得模型学习如何遵循指令回答问题，即监督微调（Supervised Fine-tuning, SFT）；再通过人类偏好进一步微调，进一步完善其行为并使其符合人类偏好（通常是基于人类反馈的强化学习，即 RLHF），得到最终 LLM。后两步也被统称为后训练（post-training）。

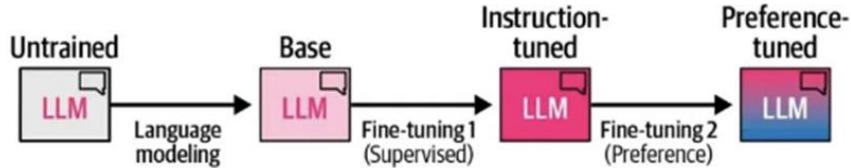


图 1-7 LLM 的一般预训练和后训练过程

R1 采用了更复杂的多阶段训练过程：

1. 纯强化学模型训练：基于预训练后但未经后训练的 DeepSeek-V3-base 模型，抛弃 SFT 而直接使用上面提到的 GRPO 强化学习方法训练，得到 DeepSeek-R1-Zero 模型。该模型已经可以进行推理，但并不符合人类的偏好，例如可能使用多种语言混合输出。本步骤的示意图如图 1-8 所示；

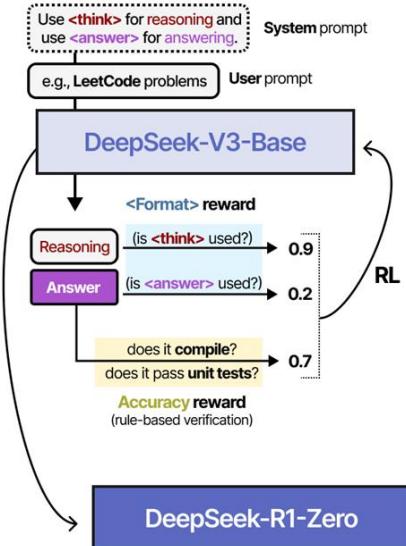


图 1-8 DeepSeek-R1-Zero 训练过程

2. 第一阶段 SFT：使用 DeepSeek-R1-Zero 模型，生成数千条“冷启动”数据（问题、思维链和回答）；再基于 DeepSeek-V3-base 模型，使用上面收集的冷启动数据进行 SFT；

为什么要进行这个冷启动阶段？R1-Zero 虽然已经具备推理能力，但刚才已经提到，其生成的结果并不符合人类喜好。冷启动旨在解决 AI 模型在强化学习初期可能遇到的不稳定和低效问题。通过一定量的高质量数据 SFT，让 RL 的训练更稳定，避免训练初期陷入“胡乱生成答案”的混乱状态；提升推理质量，让模型在强化学习前就具备一定的推理能

力，而不是完全从零开始；改善语言表达，减少语言混杂和重复内容，让推理过程更清晰、可读性更高。

3. 第一阶段 RL：应用与 DeepSeek-R1-Zero 相同的强化学习训练过程。除了正确性奖励和格式奖励外，还引入了语言一致性奖励（不能使用多种语言混合回答），对齐人类偏好。

4. 第二阶段 SFT：增强模型在写作、角色扮演和其他通用任务中的能力。利用第 3 步得到的模型生成 SFT 数据，再去微调其本身。监督数据的生成主要围绕以下两种：

- 推理数据（60 万条）：对第 3 步得到的模型使用拒绝采样生成推理过程，然后做拒绝采样，对于每个提示，采样多个响应并仅保留正确的响应；

- 非推理数据（20 万条）：对于非推理数据，如写作、事实问答、自我认知和翻译，重用 DeepSeek-V3 的部分监督微调数据集，调用 DeepSeek-V3 在回答问题之前生成潜在的 CoT（最简单的任务除外）。利用总共 80 万条样本对 DeepSeek-V3-Base 进行第二阶段 SFT。

注：拒绝采样（rejection sampling）首先使用训练好的模型生成大量的候选输出，对于 DeepSeek-R1 是多个候选推理路径。通过某种筛选机制（如人工评审或自动评分系统，对于 DeepSeek-R1 是用 DeepSeek-V3 评分）从这些候选输出中挑选出高质量的样本。

5. 第二阶段 RL：进一步使模型与人类偏好对齐，提高模型的有用性和无害性。基于 DeepSeek-V3 的流程，对于有用性，专注于最终回答（原文称为“总结”，summary）；对于无害性，评估模型的整个响应内容，包括推理过程和回答，以识别和减轻生成过程中可能出现的任何潜在风险、偏见或有害内容。

经过前 5 个步骤后，得到了 DeepSeek-R1 模型。

6. 知识蒸馏：使用第 3 步得到的模型生成的 80 万条样本对开源模型 Qwen2.5 和 LLaMA3 系列模型进行微调（SFT，而不包括强化学习）。研究结果表明，这种简单的蒸馏方法显著增强了小型模型的推理能力。

经过第 6 个步骤，得到 DeepSeek-R1-Distill 模型。总体步骤如图 1-9 所示：

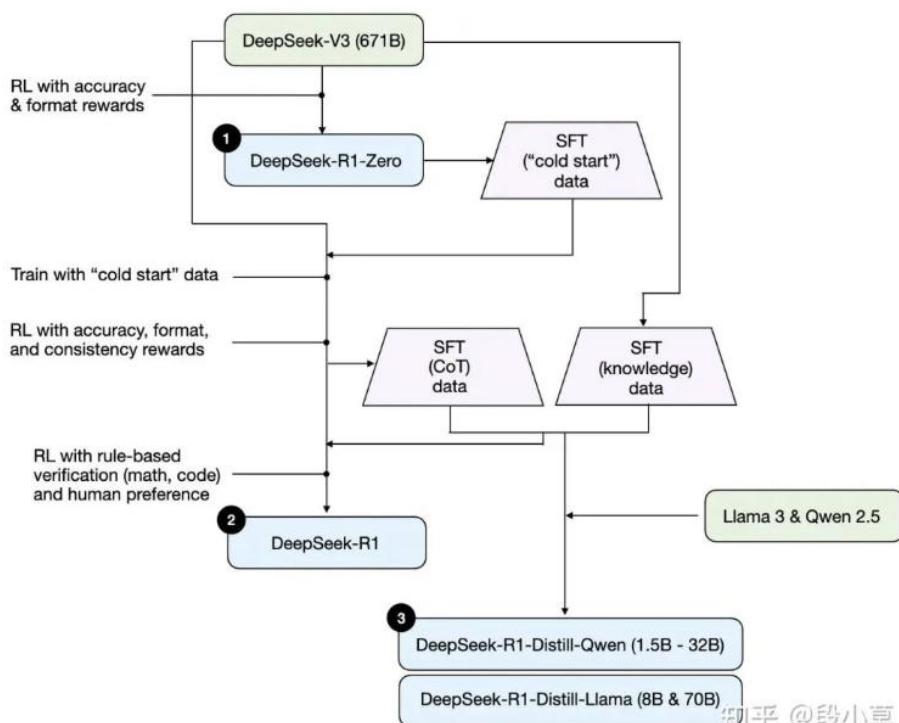


图 1-9 DeepSeek-R1 系列模型训练过程

那么，**长思维链（Long Chain-of-Thought）** 是如何被激发的？当前学者们的研究中认为，核心技能如分支和错误验证在基础模型中已经存在，但通过 RL 有效地激励这些技能以解决复杂任务需要仔细设计。此外，SFT 也能够让模型更好地学习如何进行推理。

2 模型架构：多头潜在注意力与混合专家模型

2.1 多头潜在注意力 MLA

2.1.1 多头注意力 MHA 的局限

图 2-1 (左) 是 Transformer 模型的典型结构示意图，包含编码器 Encoder 和解码器 Decoder 两部分。对于 DeepSeek 系列而言，属于 Decoder-Only 的模型，由许多层 transformer decoder 结构组成完整的模型。其中，多头注意力 (MHA) 是捕捉输入序列中的不同特征的关键机制，如图 2-1 (右) 所示。

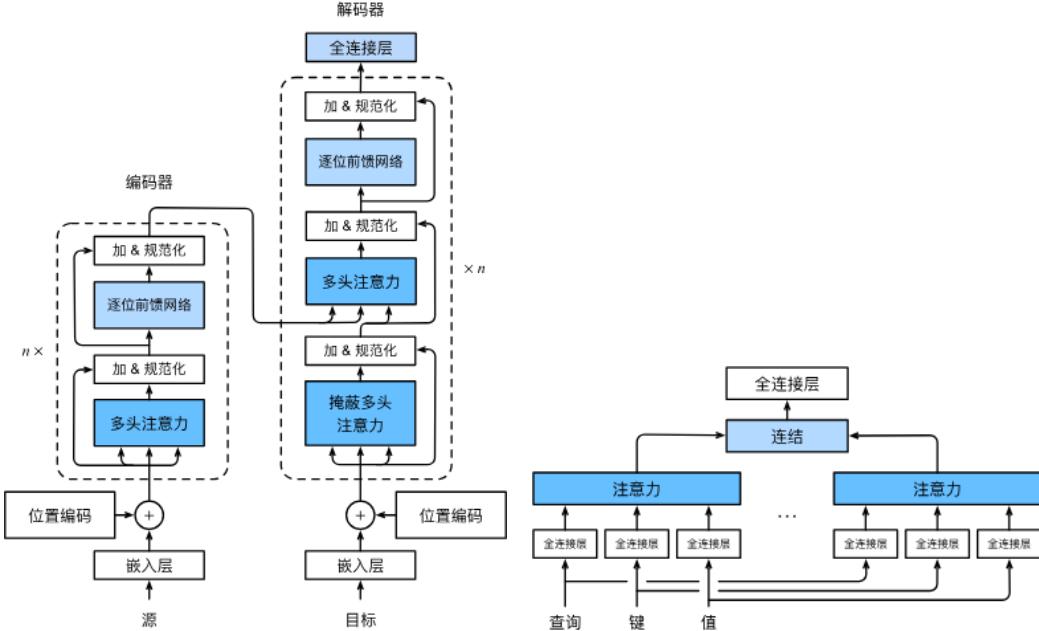


图 2-1 Transformer 模型架构 (左) 和多头注意力 (右)

标准 MHA 的计算过程：

(1) 通过三个矩阵计算查询 (Q)、键 (K) 和值 (V):

$$q_t = W^Q h_t, \quad k_t = W^K h_t, \quad v_t = W^V h_t$$

(2) 将 Q、K、V 切片成多个头进行多头注意力计算:

$$[q_{t,1}; q_{t,2}; \dots; q_{t,n_h}] = q_t, \quad [k_{t,1}; k_{t,2}; \dots; k_{t,n_h}] = k_t, \quad [v_{t,1}; v_{t,2}; \dots; v_{t,n_h}] = v_t$$

(3) 通过 softmax 函数计算权重并进行加权和:

$$o_{t,i} = \sum_{j=1}^t \text{Softmax}_j \left(\frac{q_{t,i}^T k_{j,i}}{\sqrt{d_h}} \right) v_{j,i}, \quad u_t = W^O [o_{t,1}; o_{t,2}; \dots; o_{t,n_h}]$$

然而，在计算时，每一个注意力头都需要存储一个不同的 Q、K、V 矩阵，从而导致存储空间和计算量需求大，需要压缩。

注：在模型推理中，GPU 的显存是有限的，一部分要用来存放 LLM 的参数和前向计算的激活值，选定模型后其占用是不变的；另外要用来存放模型的 KV Cache，这部分不仅依赖于模型的体量，还依赖于模型的输入长度，在推理过程中是动态增长的，当上下文长度足够长时，它的大小就会占主导地位，可能导致显存占用过大。个人实测，当使用 3090

24G 显卡部署 ollama Qwen2.5-32B 模型 4bit 量化版本时，上下文长度超出（大约）10240 将会爆显存。

为此，不同的 LLM 采用不同的方案对 K、V 进行压缩。而对于 Q，也可以单独进行类似的压缩。

例如，LLaMA2、3 等采用 GQA（将几个 Q 分为一组，共用 K 和 V）、PaLM 和 StarCoder 等采用 MQA（将所有 Q 都共用 K 和 V）。在 DeepSeek 中，引入了新的注意力机制 MLA，通过低秩键值联合压缩来显著减少推理时的键值缓存，从而提高推理效率。这几种压缩方式的对比如图 2-2 所示：

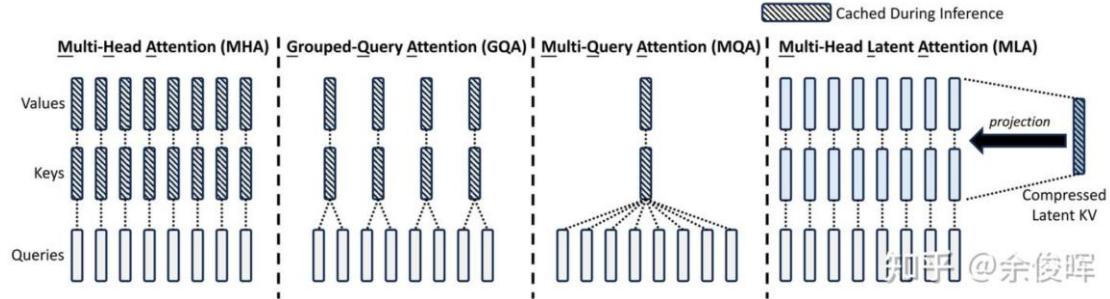


图 2-2 MHA、GQA、MQA、MLA 的对比

2.1.2 MLA 原理与计算过程

MLA 的计算过程如下：

(1) 压缩键和值：

设输入序列的第 t 个 token 的嵌入向量为 $\mathbf{h}_t \in \mathbb{R}^d$ ， d 是嵌入维度。通过一个下投影矩阵将 \mathbf{h}_t 压缩为一个低维的潜在向量 $\mathbf{c}_t^{KV} \in \mathbb{R}^{d_c}$ ，其中 $d_c \ll d_h n_h$ ， d_h 是每个注意力头的维度， n_h 是注意力头的数量：

$$c_t^K = W^{DK}(k) = W^{DK}(W^{K_proj} h_t) = (W^{DK} W^{K_proj}) h_t$$

(2) 重建键和值：

在推理阶段，通过上投影矩阵 $W^{UK} \in \mathbb{R}^{d_h n_h \times d_c}$ 和 $W^{UV} \in \mathbb{R}^{d_h n_h \times d_c}$ ，将压缩后的潜在向量 \mathbf{c}_t^{KV} 重建为键和值矩阵：

$$\mathbf{k}_t^C = W^{UK} \mathbf{c}_t^{KV}$$

$$\mathbf{v}_t^C = W^{UV} \mathbf{c}_t^{KV}$$

当前主流大模型的位置编码方式采用 RoPE（Rotary Positional Encoding，旋转位置编码），然而 RoPE 编码是和每一 token 位置有关的，而 MLA 得到的结果却与位置无关，导致 RoPE 并不兼容上面提到的 MLA。因此，DeepSeek 给 Q、K 增加一些专门用来做 RoPE 的维度（其中，K 增加的维度由所有头共享，在 deepseek 中为 64 维，相较于潜在向量的 512 维而言并不大）。这样一来，没有 RoPE 的维度就可以重复前面操作，在推理时 KV Cache 只需要存一次，而新增的带 RoPE 的维度就可以用来补充位置信息。

什么是旋转位置编码 RoPE？简单来说，RoPE 通过旋转矩阵编码绝对位置，同时将显式

的相对位置依赖性纳入自注意力公式中。例如，对于两个 Token，希望为位置为 m 的 Q 和位置为 n 的 K 内积找到一个函数 g ，其自变量包括它们处于序列中的绝对位置 x_1, x_2 ，以及相对位置 $m-n$ 。其优势在于方便外推性、非显式的长期衰减性并可以应用于线性自注意力模型。关于其具体计算，可以参考相关资料。

于是，完整的 MLA 计算过程如下：

$$\boxed{\mathbf{c}_t^{KV}} = W^{DKV} \mathbf{h}_t, \quad (1)$$

$$[\mathbf{k}_{t,1}^C; \mathbf{k}_{t,2}^C; \dots; \mathbf{k}_{t,n_h}^C] = \mathbf{k}_t^C = W^{UK} \mathbf{c}_t^{KV}, \quad (2)$$

$$\boxed{\mathbf{k}_t^R} = \text{RoPE}(W^{KR} \mathbf{h}_t), \quad (3)$$

$$\mathbf{k}_{t,i} = [\mathbf{k}_{t,i}^C; \mathbf{k}_t^R], \quad (4)$$

$$[\mathbf{v}_{t,1}^C; \mathbf{v}_{t,2}^C; \dots; \mathbf{v}_{t,n_h}^C] = \mathbf{v}_t^C = W^{UV} \mathbf{c}_t^{KV}, \quad (5)$$

其中 $\mathbf{c}_t^{KV} \in \mathbb{R}^{d_c}$ 是键和值的压缩潜在向量； $d_c (\ll d_h n_h)$ 表示 KV 压缩维度； $W^{DKV} \in \mathbb{R}^{d_c \times d}$ 表示降维投影矩阵； $W^{UK}, W^{UV} \in \mathbb{R}^{d_h n_h \times d_c}$ 分别是键和值的升维投影矩阵； $W^{KR} \in \mathbb{R}^{d_h^R \times d}$ 是用于生成携带旋转位置嵌入（Rotary Positional Embedding, RoPE）的解耦键的矩阵（[Su et al., 2024](#)）； $\text{RoPE}(\cdot)$ 表示应用 RoPE 矩阵的操作； $[\cdot; \cdot]$ 表示拼接。注意，对于 MLA，仅需在生成过程中缓存蓝色框中的向量（即 \mathbf{c}_t^{KV} 和 \mathbf{k}_t^R ），这显著减少了 KV 缓存，同时保持了与标准多头注意力（MHA）（[Vaswani et al., 2017](#)）相当的性能。

对于注意力查询，我们还执行低秩压缩，这可以在训练期间减少激活内存：

$$\mathbf{c}_t^Q = W^{DQ} \mathbf{h}_t, \quad (6)$$

$$[\mathbf{q}_{t,1}^C; \mathbf{q}_{t,2}^C; \dots; \mathbf{q}_{t,n_h}^C] = \mathbf{q}_t^C = W^{UQ} \mathbf{c}_t^Q, \quad (7)$$

$$[\mathbf{q}_{t,1}^R; \mathbf{q}_{t,2}^R; \dots; \mathbf{q}_{t,n_h}^R] = \mathbf{q}_t^R = \text{RoPE}(W^{QR} \mathbf{c}_t^Q), \quad (8)$$

$$\mathbf{q}_{t,i} = [\mathbf{q}_{t,i}^C; \mathbf{q}_{t,i}^R], \quad (9)$$

其中 $\mathbf{c}_t^Q \in \mathbb{R}^{d'_c}$ 是查询的压缩潜在向量； $d'_c (\ll d_h n_h)$ 表示查询压缩维度； $W^{DQ} \in \mathbb{R}^{d'_c \times d}$, $W^{UQ} \in \mathbb{R}^{d_h n_h \times d'_c}$ 分别是查询的降维和升维矩阵； $W^{QR} \in \mathbb{R}^{d_h^R n_h \times d'_c}$ 是生成携带 RoPE 的解耦查询的矩阵。

最终，注意力查询 ($\mathbf{q}_{t,i}$)、键 ($\mathbf{k}_{j,i}$) 和值 ($\mathbf{v}_{j,i}^C$) 被组合以生成最终的注意力输出 \mathbf{u}_t ：

$$\mathbf{o}_{t,i} = \sum_{j=1}^t \text{Softmax}_j \left(\frac{\mathbf{q}_{t,i}^T \mathbf{k}_{j,i}}{\sqrt{d_h + d_h^R}} \right) \mathbf{v}_{j,i}^C, \quad (10)$$

$$\mathbf{u}_t = W^O [\mathbf{o}_{t,1}; \mathbf{o}_{t,2}; \dots; \mathbf{o}_{t,n_h}], \quad (11)$$

其中 $W^O \in \mathbb{R}^{d \times d_h n_h}$ 表示输出投影矩阵。

2.1.3 最新探索：原生稀疏注意力 NSA

在 MLA 之后，DeepSeek 的最新论文中又公开了新的注意力压缩方式：原生稀疏注意力 NSA（Native Sparse Attention），代表了该方向的最新探索，并有可能在 DeepSeek 的下一代模型中采用。

NSA 实现了高效的长上下文建模，采用动态分层稀疏策略，结合粗粒度的 token 压缩与细粒度的 token 选择，以保持全局上下文感知和局部精度。NSA 的

论文^[12]中提到其关键创新在于：(1) 通过算术强度平衡的算法设计实现显著加速，并针对现代硬件进行实现优化。(2) 实现端到端训练，减少预训练计算而不牺牲模型性能。

为什么说它是“原生”稀疏注意力？这是指使用 **NSA** 的模型在预训练阶段其注意力就是稀疏的，而不是在预训练之后再做稀疏化处理（例如，可以在计算注意力分数之前，先识别出哪些键向量 k 和值向量 v 对当前查询向量 q 是重要的，忽略计算不重要的 k 、 v 的注意力得分。但是这仍然需要计算每个 k 对当前 q 的注意力分数并设定阈值，实质上并没有减少很多计算量，而且导致模型效果变差）。因此，**NSA** 重点在于原生地识别哪些 k 、 v 用来计算注意力。

首先，我们从模型推理中的两个阶段讲起。对于一个典型的推理型 LLM，分为处理 prompt，生成长 CoT 和生成最终回答三个步骤。这其中涉及到两个阶段：预填充（Pre-Filling）和解码（Decoding），其中输出 CoT 和回答的部分归为解码阶段，如图 2-3 所示。

注意，模型推理阶段（inference）和推理型 LLM（reasoning）的“推理”含义是不同的。

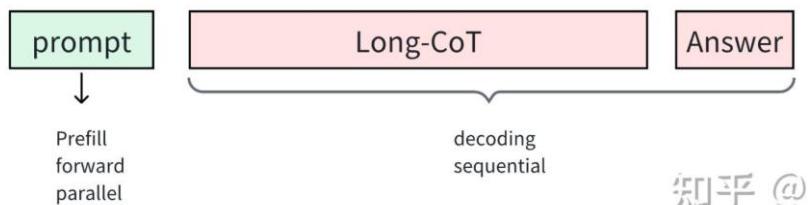


图 2-3 模型推理中的预填充和解码阶段

在预填充阶段，模型对输入的上下文信息进行处理，并准备好（用以生成文本的）初始状态的过程。具体而言，是填充每个注意力层的 KV-Cache，该计算过程是并行的，成本较低。

因此，使用 **MLA** 的好处在于，它减少了预填充阶段 KV-Cache 所占用的存储空间，从而使得 GPU 可以同时处理多个输入（比如，多个用户同时询问的问题），降低成本。

解码阶段则是模型根据预填充阶段准备好的上下文信息，逐步生成输出文本的过程。对于一般模型，就是逐个生成 token，然后更新上下文以生成下一个 token，直至生成完毕，对于 MTP，则一次生成多个 token。然而，该计算过程是串行的，成本较高。随着输出序列长度的增加，普通注意力机制的高复杂性（时间复杂度是长度的平方）成为关键的延迟瓶颈。在使用 softmax 进行 64k 长度上下文的解码时，注意力计算占总延迟的 70-80%。

解码阶段相比于预填充阶段的成本更高，这也是大模型 API 中输出 token 的价格高于输入 token 的原因。

NSA 是用来取代 **MLA** 的吗？从上面的分析来看，并不是，二者是可以结合的，**MLA** 用于预填充阶段，而 **NSA** 用于解码阶段。

如图 2-4 所示，**NSA** 架构将输入序列按时间维度划分为连续的 block，并通过三条并行的注意力分支进行处理：**压缩注意力**（Compressed Attention）通过可学习的 MLP 将一段序列压缩成单一向量，用于捕获粗粒度的全局信息；**选择性注意力**（Selected Attention）则保留最重要的细粒度 token 信息；**滑动窗口注意力**（Sliding Attention）用于处理近期的局部上下文。这种分层设计使得模型

能够在保持表达能力的同时大幅降低计算复杂度。

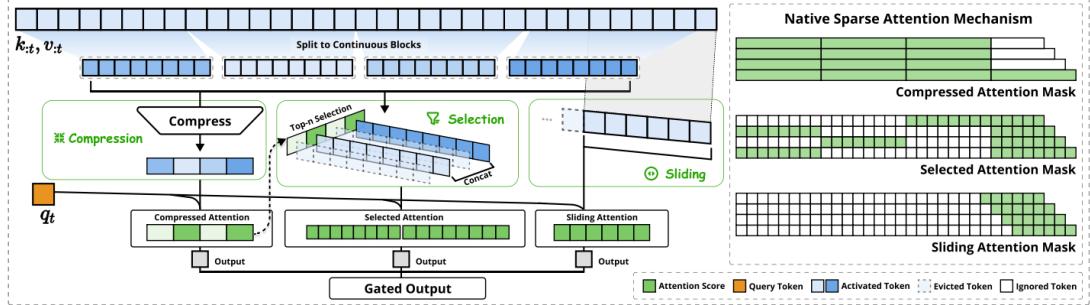


图 2-4 NSA 架构示意图

下面的网络资料对该过程进行了最直观的讲述：

脱离原论文公式，在 NSA 的框架图里我们用最简单的数值来描述，计算注意力前我们有 $\mathbf{q}_{:t}$ 和 $\mathbf{k}_{:t}, \mathbf{v}_{:t}$

1. 词元序列 t 为 32，即左上顶部蓝色方格长条每一格代表第 i 时刻的 $\mathbf{k}_i, \mathbf{v}_i$
2. 将 $\mathbf{k}_{:32}, \mathbf{v}_{:32}$ 按照长度为 8 划分为 4 块 $\mathbf{k}_{1:8}^{(1)}, \mathbf{v}_{1:8}^{(1)}, \mathbf{k}_{8:16}^{(2)}, \mathbf{v}_{8:16}^{(2)} \dots$
3. **压缩**：将每个 KV 块压成一个向量即 $\mathbf{k}_{1:8}^{(1)} \rightarrow \tilde{\mathbf{k}}^{(1)\text{cmp}} \in \mathcal{R}^{d \times c}$, $\mathbf{v}^{(1)}$ 同理，压缩的目的是把一段长度为 8 的序列压成 c 个序列，并且有关系 $c < 8$ ，为了便于理解我们将 $c = 1$ 既是 1 个 token 代表了一序列。至于怎么压缩我们后面探讨。这里的压缩注意力是 \mathbf{q}_{32} 和 4 个段落的 $\tilde{\mathbf{k}}^{(b)\text{cmp}}, b = 1, 2, 3, 4$ 进行计算得到 4 个注意力分数，所以这里做将 \mathbf{v} 进行计算后，就得到压缩的注意力输出了，代表了全局信息 $\mathbf{o}_{32}^{(\text{cmp})} \in \mathcal{R}^{d \times 1}$
4. **选择**：在压缩时得到的段落注意力分数，选择 top-2，即第 2 和第 4 的绿色块。那么我们就找到对应的段落块当成局部信息 $\mathbf{k}_{8:16}^{(2)}, \mathbf{v}_{8:16}^{(2)}, \mathbf{k}_{24:32}^{(4)}, \mathbf{v}_{24:32}^{(4)}$ ，获取到 KV 后我们也可以计算选择注意力，最终为 $\mathbf{o}_{32}^{(\text{sle})} \in \mathcal{R}^{d \times 1}$
5. **滑窗**：在原 $\mathbf{k}_{:32}, \mathbf{v}_{:32}$ 序列里取最近的 8 个键和值 $\mathbf{k}_{24:32}, \mathbf{v}_{24:32}$ ，同样可以得到滑窗注意力为 $\mathbf{o}_{32}^{(\text{win})} \in \mathcal{R}^{d \times 1}$
6. **门控**：汇聚三种注意力 $\mathbf{g}_{32}^{(\text{cmp})} \mathbf{o}_{32}^{(\text{cmp})} + \mathbf{g}_{32}^{(\text{sle})} \mathbf{o}_{32}^{(\text{sle})} + \mathbf{g}_{32}^{(\text{win})} \mathbf{o}_{32}^{(\text{win})}$ ，其中门控 \mathbf{g} 为输入 x_{32} 进行线性层变换并加入 sigmoid 激活得到。门控是可学习的。

至此我们再进一步分析，原来我们有 $t = 32$ 个上下文 KV，在压缩/选择/滑窗里我们分别有 $N_t = 4 + 8 * 2 + 8 = 28$ 个上下文 KV，实现了注意力的减少。

当我们的上下文为 64k 时，如果我们取 128 个全局压缩 KV，8 个 512 选择块 KV 和 近窗口 4096 个 KV，那么我们得到了压缩倍数 7.88：

$$65536 / (128 + 8 * 512 + 4096) = 65536 / 8320 \approx 7.88$$

接下来，看看 NSA 的具体实现：

压缩注意力将一段序列的 KV 压成一个 KV，代表片段的全局信息。在多头实现时，每个头关注不同的片段。对于位置 t 的 token，其计算公式如下：

$$\tilde{K}_t^{\text{cmp}} = f_K^{\text{cmp}}(\mathbf{k}_{\cdot:t}) = \left\{ \phi(\mathbf{k}_{id+1:id+l}) \mid 1 \leq i \leq \left\lfloor \frac{t-l}{d} \right\rfloor \right\}$$

总长度 块长度
重叠 stride, d=l 时无重叠

简要描述符号压缩算子 $\phi(\cdot) : \mathbb{R}^{l \times d} \rightarrow \mathbb{R}^{1 \times d}$, 具体的实现可以是一个MLP层和可以学习的位置编码(内部), MLP参数与数据块长度有关。

压缩注意力阶段, 我们得到了各个头各个片段的注意力得分。

$$\mathbf{p}_t^{\text{cmp}} = \text{Softmax}(\mathbf{q}_t^T \tilde{K}_t^{\text{cmp}}),$$

随后, 使用选择注意力判断哪些片段需要精细地提取特征, 各符号的含义与前面一致:

$$\mathbf{p}_t^{\text{slc}}[j] = \sum_{m=0}^{\frac{l'}{d}-1} \sum_{n=0}^{\frac{l}{d}-1} \mathbf{p}_t^{\text{cmp}} \left[\frac{l'}{d}j + m + n \right],$$

当压缩块和选择块共享相同的分块方案, 即 $l'=l=d$ 时, 可以获得选择块的注意力得分:

$$\mathbf{p}_t^{\text{slc}} = \mathbf{p}_t^{\text{cmp}}$$

对于具有 H 个头的多头注意力, 有:

$$\mathbf{p}_t^{\text{slc}'} = \sum_{h=1}^H \mathbf{p}_t^{\text{slc},(h)},$$

这里的实现可以参考图 2-5:

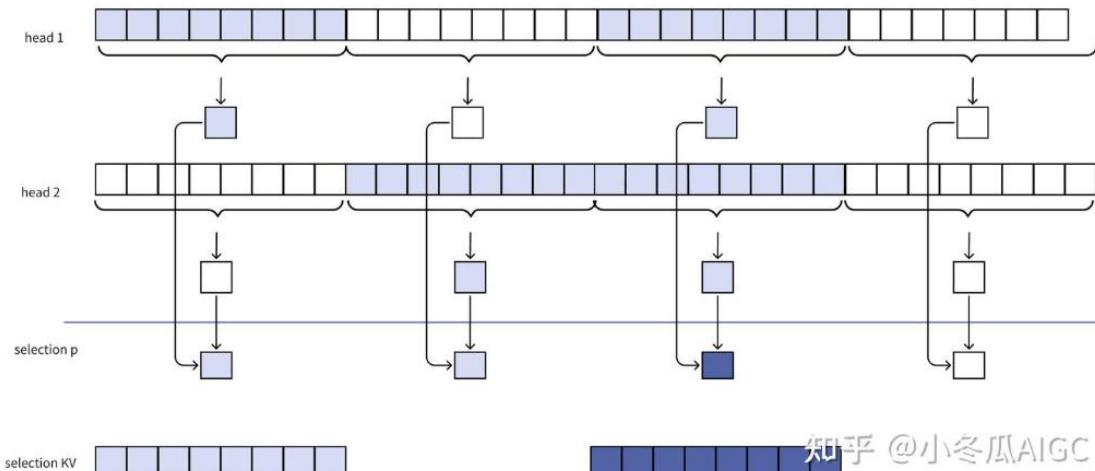


图 2-5 多头注意力下的压缩注意力与选择注意力实现

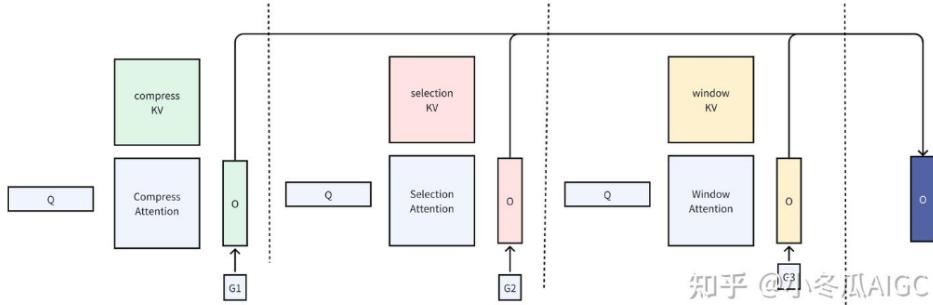
滑动窗口注意力只关注与当前 q 最近的 kv 片段, 实现较为简单:

$$\tilde{K}_t^{\text{win}} = \mathbf{k}_{[t-w:t]} \quad \tilde{V}_t^{\text{win}} = \mathbf{v}_{t-w:t}$$

窗口宽度为 w

最后, 通过包含一个 MLP 和 sigmoid 函数的门控机制, 将三种注意力的输出

出聚合为最终输出，如图 2-6 所示：



知乎 @小冬瓜AIGC

那么我们计算完整的 NSA， \mathcal{C} 代表三种注意力方法

$$\mathbf{o}_t^* = \sum_{c \in \mathcal{C}} g_t^c \cdot \text{Attn}(\mathbf{q}_t, \tilde{\mathbf{K}}_t^c, \tilde{\mathbf{V}}_t^c)$$

图 2-6 完整的 NSA 实现

在具体实现中，为了充分利用现代的 GPU 硬件，设计了下面的核函数：

Grid Loop: 先加载 Q，逐个元素计算 NSA

Inner Loop: 再加载 KV，当 t 增大时，由于 KV 块恒为 3 块，NSA 的加速越明显。

在图 2-7 中一个绿色块代表了一个 q 和一段 kv 计算。那么对于压缩注意力和窗口注意力，实际上是在一个 block 里面计算完。对于选择注意力，通过 MQA 和 GQA 等方法压缩 K、V，从而进一步减少 SRAM 与 HBM 之间的交换次数，提高效率。

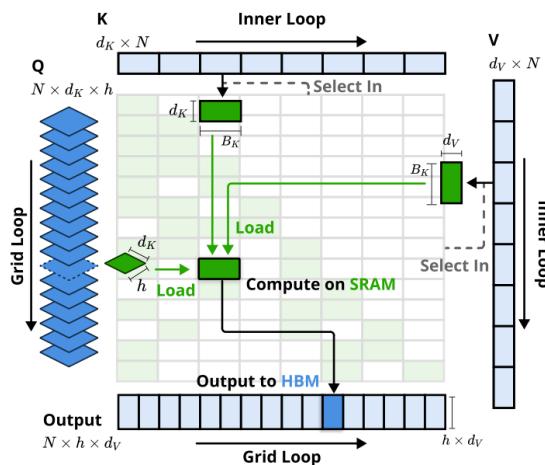


Figure 3 | NSA 的内核设计。内核通过 GQA 组（网格循环）加载查询，获取相应的稀疏 KV 块（内循环），并在 SRAM 上执行注意力计算。绿色块表示 SRAM 上的数据，而蓝色表示 HBM 上的数据。

图 2-7 NSA 核函数

跟随 DeepSeek 的脚步，Kimi 也发布了他们对稀疏注意力的探索：MOBA (MOBA: Mixture of Block Attention for Long-Context LLMS)。该技术通过将输入序列分成多个块，允许每个查询 token 动态选择最相关的块进行关注，显著提

升了处理长序列时的效率。在本文中，我们不对此进行详细介绍，有兴趣的读者可以关注其原论文。

2.2 混合专家模型 MoE

2.2.1 MoE 基本原理

在图 2-1（左）的 Transformer 结构中，具有前馈网络（FFN）层，通常由两个线性层和一层激活函数（ReLU）组成。在 MoE 中，它被多个 MoE 层和一个门控网络（路由器）替代，使得在推理时每次仅有部分“专家”网络被激活。如图 2-8 所示，混合专家模型主要由两个关键部分组成：

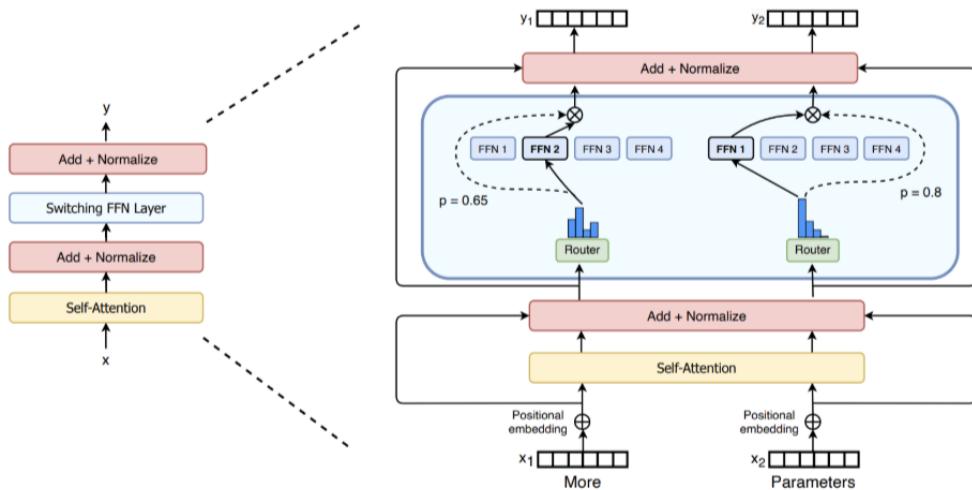


图 2-8 混合专家模型的基本结构

稀疏 MoE 层：这些层代替了传统 Transformer 模型中的 FFN 层。MoE 层包含若干“专家”，每个专家本身是一个独立的网络。在实际应用中，这些专家通常是 FFN，但也可以是更复杂的网络结构。在训练过程中，输入数据被门控模型分配到不同的专家模型中进行处理；在推理过程中，被门控选择的专家会针对输入的数据，产生相应的输出。这些输出最后会和每个专家模型处理该特征的能力分配的权重进行加权组合，形成最终的预测结果。

$$Importance(X) = \sum_{x \in X} G(x)$$

$$L_{importance}(X) = w_{importance} \cdot CV(Importance(X))^2$$

什么叫稀疏性？稀疏性允许我们仅针对整个系统的某些特定部分执行计算。这意味着并非所有参数都会在处理每个输入时被激活或使用，而是根据输入的特定特征或需求，只有部分参数集合被调用和运行。

门控网络 GateNet（也称为“路由器 router”）：由可学习的参数组成，用于决定哪些 Token 被发送到哪个专家。有时，一个 Token 也可以被发送到多个专家。它接收单个数据元素作为输入，然后输出一个权重，这些权重表示每个专家模型对处理输入数据的贡献。一般是通过 softmax 函数利用专家或 token 对概率分布进行建模，并选择前 K 个。

$$G(x) = \text{Softmax}(\text{KeepTopK}(H(x), k))$$

$$H(x)_i = (x \cdot W_g)_i + \text{StandardNormal}() \cdot \text{Softplus}((x \cdot W_{\text{noise}})_i)$$

$$\text{KeepTopK}(v, k)_i = \begin{cases} v_i & \text{if } v_i \text{ is in the top } k \text{ elements of } v. \\ -\infty & \text{otherwise.} \end{cases}$$

注：此“门控网络”与 GRU 或 LSTM 的“门”概念并不相同。MoE 的“门”概念主要是用于匹配数据和专家模型之间的连接，选择要通过的对象，而 GRU 或 LSTM 的“门”概念主要是一种控制信息流动的装置，它可以保留或通过一定比例的数据，控制流量。

2.2.2 DeepSeek MoE 实现

DeepSeek MoE 通过细粒度的专家分割和共享专家隔离来实现更高效的模型训练。其基本思想是将专家分割成更细的粒度以提高专家的专业化，并通过隔离一些共享专家来缓解路由专家之间的知识冗余，如图 2-9 所示。

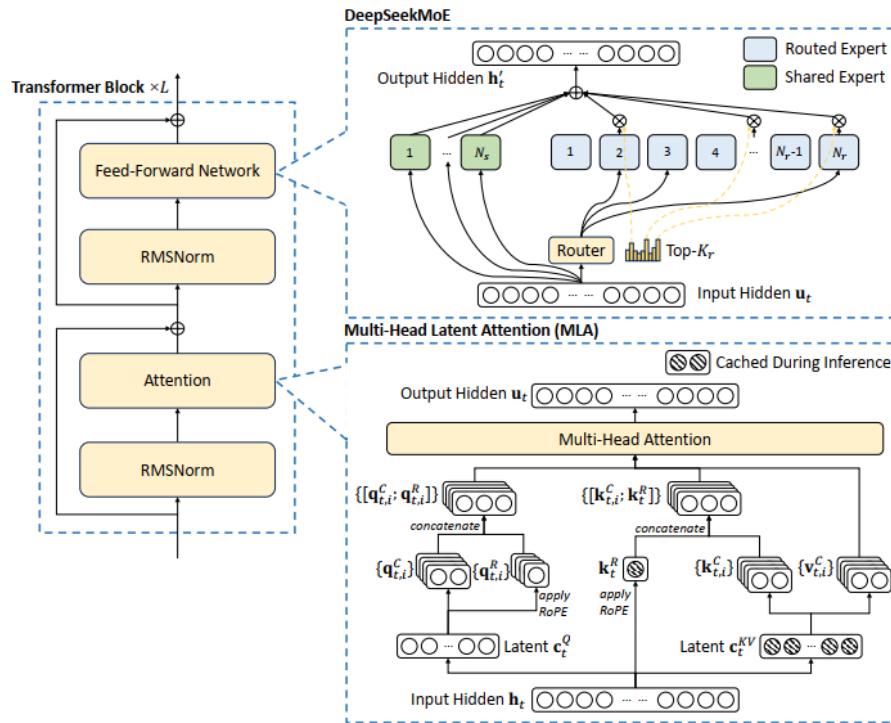


图 2-9 DeepSeek-V3 所使用的 MoE 架构（上）和 MLA（下）

细粒度的专家分割：在保持一致的专家参数数量和计算成本的同时，对专家进行了更细粒度的细分。具体来说，将每个专家 FFN 细分为 m 个较小的专家，方法是将 FFN 中间隐藏维度减小到其原始大小的 $1/m$ 倍。作为响应，将激活专家的数量增加到 m 倍以保持相同的计算成本。

共享专家隔离：在传统的策略中，分配给不同专家的 Token 需要一些共同的知识或信息。因此，多个专家可能会集中获取各自参数中的共同知识，从而导致专家参数的冗余。为此，隔离 K_s 个专家以作为共享专家。无论 router 模块如何，每个 token 都将确定性地分配给这些共享专家。为了保持恒定的计算成本，其他路由专家中激活的专家数量将减少 K_s 。这两个策略具体如图 2-10 所示。

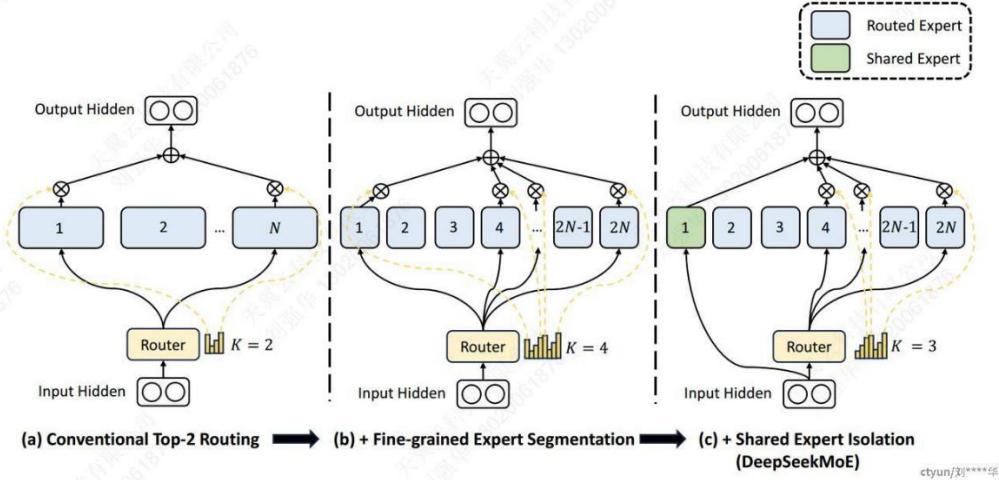


图 2-10 DeepSeek-V3 所使用的细粒度专家分割和共享专家隔离策略
完整的 DeepSeekMoE 架构中的 MoE 层计算公式如下：

令 \mathbf{u}_t 为第 t 个 token 的 FFN 输入，我们计算 FFN 输出 \mathbf{h}'_t 如下：

$$\mathbf{h}'_t = \mathbf{u}_t + \sum_{i=1}^{N_s} \text{FFN}_i^{(s)}(\mathbf{u}_t) + \sum_{i=1}^{N_r} g_{i,t} \text{FFN}_i^{(r)}(\mathbf{u}_t), \quad (20)$$

$$g_{i,t} = \begin{cases} s_{i,t}, & s_{i,t} \in \text{Topk}(\{s_{j,t} | 1 \leq j \leq N_r\}, K_r), \\ 0, & \text{otherwise,} \end{cases} \quad (21)$$

$$s_{i,t} = \text{Softmax}_i(\mathbf{u}_t^T \mathbf{e}_i), \quad (22)$$

其中， N_s 和 N_r 分别表示共享专家和路由专家的数量； $\text{FFN}_i^{(s)}(\cdot)$ 和 $\text{FFN}_i^{(r)}(\cdot)$ 分别表示第 i 个共享专家和第 i 个路由专家； K_r 表示被激活的路由专家的数量； $g_{i,t}$ 是第 i 个专家的门控值； $s_{i,t}$ 是 token 与专家的亲和度； \mathbf{e}_i 是本层中第 i 个路由专家的质心； $\text{Topk}(\cdot, K)$ 表示由第 t 个 token 与所有路由专家计算的亲和度得分中，取前 K 个最高分值组成的集合。

在 DeepSeek-V3 中，进一步将计算得分时所用的 softmax 函数改为了 sigmoid 函数，并对所有选定的亲和度得分进行归一化以生成门控值 $g_{i,t}$ 。

2.2.3 负载均衡技术

专家负载均衡：如果所有的 Token 都被发送到少数几个受欢迎的专家，那么训练效率将会降低。为了缓解这个问题，需要引入负载均衡策略来确保所有专家接收到大致相等数量的训练样本，从而平衡了专家之间的选择。通常，这是通过引入辅助损失来实现的。

在 DeepSeek-V2 中，通过设备限制路由机制来控制 MoE 相关的通信成本，先选择得分最高专家的 M 个设备，再在这些设备上执行 top-K 选择专家，以确保每个 token 的专家分布在最多 M 个设备上。此外，还设计了三种辅助损失来控制专家级负载平衡、设备级负载平衡和通信平衡。

在 DeepSeek-V3 中，进一步改进了平衡方式，引入了无辅助损失的负载均衡策略。具体而言，为每个专家引入一个偏置项 b_i ，并将其添加到相应的亲和度得分 $s_{i,t}$ 中以确定 top-K 路由：

$$g'_{i,t} = \begin{cases} s_{i,t}, & s_{i,t} + b_i \in \text{Topk}(\{s_{j,t} + b_j | 1 \leq j \leq N_r\}, K_r), \\ 0, & \text{otherwise.} \end{cases} \quad (16)$$

在每个步骤结束时，如果某个专家过载，则减少其对应的偏置项；如果某个专家负载不足，将增加其对应的偏置项。减少或增加的幅度 γ 是一个称为偏差更新速度的超参数。偏置项仅用于路由，而门控值仍从原始亲和力得分导出。

尽管 DeepSeek-V3 主要依赖于无辅助损失的策略来实现负载平衡，为了防止任何单个序列内的极端不平衡，还采用了一个序列级平衡损失作为辅助。通过以上策略，使得 V3 模型在训练和推理时不会因为专家平衡问题而丢弃任何 Token。

MoE 看起来也是一种多智能体协同结构，它与多智能体 **Multi-Agent** 有什么区别？

MoE 是一种模型架构优化技术，强调“模型内部的模块化分工”，所有专家共享同一任务目标（如生成连贯文本）。Multi-Agent 是一种系统级的分工协作框架，强调“外部独立实体间的交互与协调”，智能体有各自的目标和决策逻辑。具体的区别如表 2-1 所示：

表 2-1 混合专家模型与多智能体的对比

维度	混合专家模型 MoE	多智能体 Multi-Agent
架构层级	模型内部结构	系统级架构（多个独立模型）
参数共享	专家共享部分参数	参数不共享
协作机制	门控网络动态分配输入到专家	显式通信
通信开销	低（模型内部隐式分配）	高（需显式通信协议）
分工粒度	细（处理输入的不同部分）	粗（处理任务的不同阶段）
自主性	专家被动响应门控网络的分配	智能体可主动决策、发起协作
训练方式	端到端联合优化（专家+门控网络）	可独立训练或通过强化学习协作优化
计算效率	高（动态激活部分专家）	较低（需并行运行多个模型）

2.3 多 Token 预测 MTP

目前的大模型通过预测下一个 Token 作为训练目标，通过这样的训练，使模型具备知识和推理能力，但是存在以下三个问题：

效率低：相比于人类儿童学习语言，预测下一个 token 方法需要更大量的数据才能达到相同的流利度；

局部性：预测下一个 token 倾向于捕捉局部模式，难以学习长距离依赖关系和全局语义，进而做一下“困难”的决策；

推理速度慢：预测下一个 token 方法在推理时需要逐个生成 token，导致推理速度较慢。

为此，DeepSeek-V3 通过在每个位置预测多个未来 token，增加训练信号的密度，从而提高模型性能。其架构如图 2-11 所示：

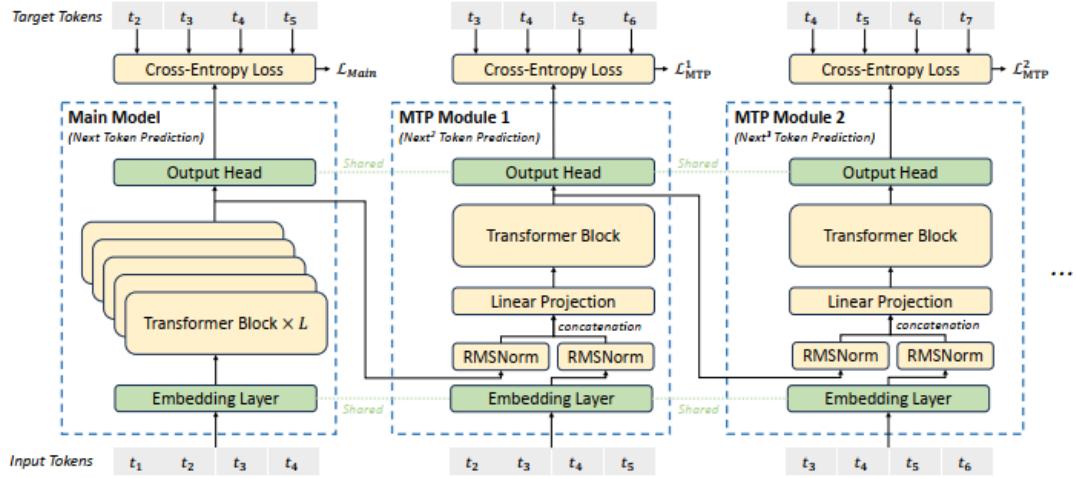


图 2-11 DeepSeek-V3 多 Token 预测模块

具体来说，使用 D 个顺序模块来预测 D 个额外的 token。第 k 个 MTP 模块由一个共享的嵌入层 Emb、一个共享的输出头 OutHead、一个 Transformer 块 TRM_k 和一个投影矩阵 $M_k \in \mathbb{R}^{d \times 2d}$ 组成。对于第 i 个输入 Token t_i 和第 k 个预测 Token (称为第 k 个预测深度)，首先将该 Token 在第 $(k-1)$ 个深度的表示 $\mathbf{h}_i^{k-1} \in \mathbb{R}^d$ 和第 $(i+k)$ 个 Token 的嵌入 $\text{Emb}(t_{i+k}) \in \mathbb{R}^d$ 通过线性投影进行组合：

$$\mathbf{h}_i^k = M_k[\text{RMSNorm}(\mathbf{h}_i^{k-1}); \text{RMSNorm}(\text{Emb}(t_{i+k}))], \quad (21)$$

组合后的向量作为第 k 层 Transformer 块的输入，生成当前层的输出表示。

$$\mathbf{h}_{1:T-k}^k = \text{TRM}_k(\mathbf{h}_{1:T-k}^k), \quad (22)$$

其中 T 表示输入序列的长度， $i:j$ 表示切片操作（包括左右边界）。最后，以当前层的输出表示作为输入，共享输出头计算第 k 个标预测深度的 Token 的概率分布 $\mathbf{p}_{i+1:k}^k \in \mathbb{R}^V$ ，其中 V 是输出词汇表的大小。

对于每个预测深度，使用其真实 Token 和预测得到的概率计算交叉熵损失：

$$\mathcal{L}_{\text{MTP}}^k = \text{CrossEntropy}(p_{2+k:T+1}^k, t_{2+k:T+1}) = -\frac{1}{T} \sum_{i=2+k}^{T+1} \log p_i^k[t_i], \quad (24)$$

其中 T 表示输入序列的长度， t_i 表示第 i 个位置的真实 Token， P 表示第 k 个 MTP 模块给出的 t_i 的预测概率。最后，计算所有深度的 MTP 损失的平均值，并乘以一个权重 λ ，以获得总体的 MTP 损失 L_{MTP} ，作为模型的训练目标。

$$\mathcal{L}_{\text{MTP}} = \frac{\lambda}{D} \sum_{k=1}^D \mathcal{L}_{\text{MTP}}^k. \quad (25)$$

3 训练流程：强化学习的意义

3.1 人类反馈强化学习 RLHF 与近端策略优化 PPO

3.1.1 PPO 基本概念

在图 1-7 中，预训练与 SFT 后的大模型，需要经过进一步的微调以对齐人类偏好，确保生成的回复符合人类阅读习惯，并不包含有害信息。在通常的大模型中，这一步是通过人类反馈强化学习（Reinforcement Learning with Human Feedback, RLHF）实现的。例如，GPT-3.5 的 RLHF 过程如图 3-1 所示。

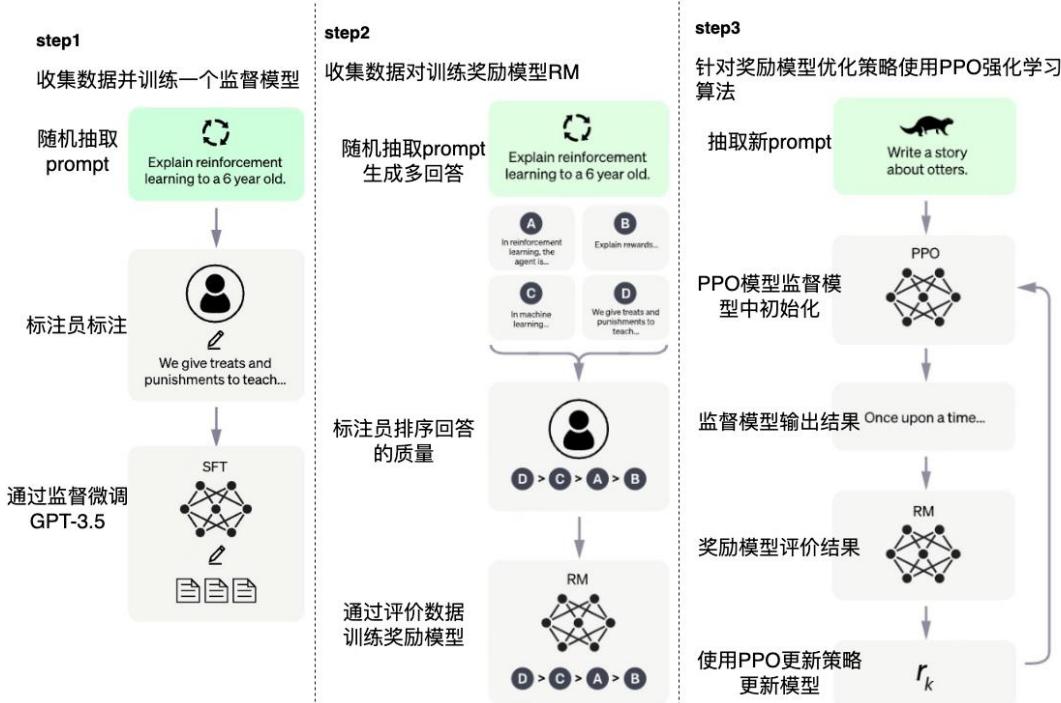


图 3-1 GPT-3.5 的 RLHF 流程示意图

在完成预训练和 SFT 之后，首先需要收集一些问答数据，通过人工根据一定的规则标注这些回答是否符合人类喜好，即奖励得分。随后，基于这些标注数据训练一个奖励模型 Reward Model（简称 RM 或 RW），在后面的 RLHF 过程中代替标注员得到奖励得分。其损失函数定义如下：

$$\mathcal{L}(\psi) = \log \sigma(r(x, y_w) - r(x, y_l)) \quad (3) \text{ 其中 } \sigma$$

是 sigmoid 函数， r 代表参数为 ψ 的奖励模型的值， $r(x, y)$ 表示针对输入提示 x 和输出 y 所预测出的单一标量奖励值。

在包括 GPT-3.5 在内的一系列主流大模型中，RLHF 所采用的策略叫做近端策略优化（Proximal Policy Optimization, PPO）。其中有四个主要模型，分别是：

Actor Model: 演员模型，即想要训练的目标语言模型，也称为策略模型 Policy Model

Critic Model: 评论家模型，预测总收益 V_t

Reward Model: 奖励模型，计算即时收益 R_t

Reference Model: 参考模型，给语言模型增加一些“约束”，防止模型朝不受控制的方向更新，导致效果越来越差。

其中，Actor/Critic Model 在 RLHF 阶段是需要训练的；而 Reward/Reference Model 是参数冻结的。RLHF 的一般流程是，基于 Critic/Reward/Reference Model，综合它们的结果计算奖励和损失函数，随后用于更新 Actor 和 Critic Model。

什么是收益？对于一个智能体（在 RLHF 中即为待训练的 LLM），收益就是它所产生的内容符合人类喜好的情况，也就是前面人工标注和奖励模型得到的奖励得分。在 t 时刻状态 S 的总收益 = 身处状态 S 带来的即时收益 + 从状态 S 出发后带来的未来收益。即时收益为真值，但由于未来收益的真值不可知，因此是一个预测值。用数学公式表达就是：

$$V_t = R_t + \gamma V_{t+1}$$

其中， V_t 是 t 时刻的总收益（即时+未来）， R_t 是 t 时刻的即时收益， V_{t+1} 是 $t+1$ 时刻的总收益（对于 t 时刻而言，就是未来收益）， γ 是一个权重系数，也称为折扣因子。当模型根据 R_t 和 V_t ，产生了一个新 Token（称为行动 A_t ）后，原来的上文+新的 Token 变成了新的上文，模型的状态变为 S_{t+1} ，随后再次计算即时收益和未来收益，再基于此做出下一步行动。

需要注意的是，在 RLHF 中行动 A_t 是由演员模型（语言模型）产生的，而 R_t 和 V_t 则由评论家模型产生。

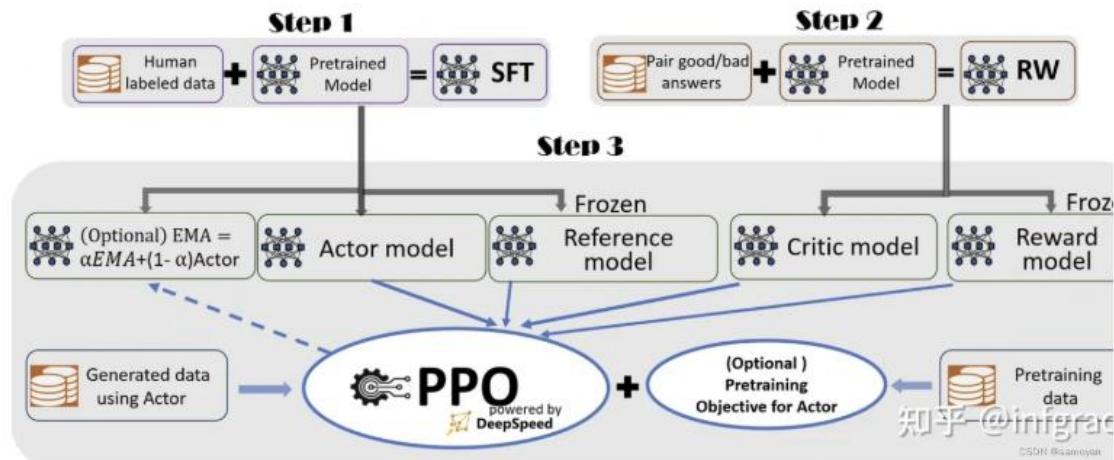


图 3-2 RLHF-PPO 过程中的四种模型及其关系

四种模型之间的关系如图 3-2 所示。

参考模型 Reference Model 通常也用 SFT 阶段得到的 SFT 模型做初始化，在训练过程中，它的参数是冻结的。为了使训练出来的 LLM 既能符合人类喜好，又尽量让它和 SFT 模型不要差异太大，则希望两个模型的输出分布尽量相似。由于 LLM 得到的是概率分布，因此采用 K-L 散度来衡量概率分布之间的接近性，计算方式为对数概率相减： $\text{ref_log_probs} - \log_probs$ ，如图 3-3 所示。

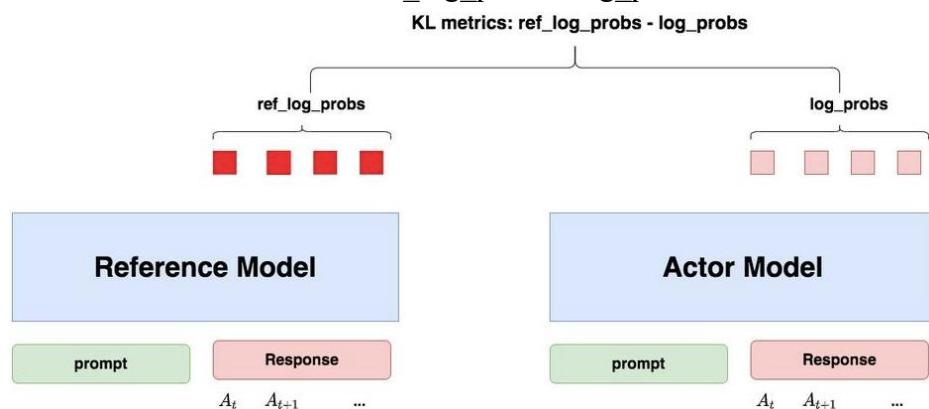


图 3-3 演员模型和参考模型的结构和 KL 散度计算

由于在 Actor Model 训练过程中，无法得到客观存在的总收益真值（这需要模型完成全部回答后才能知道），只能训练一个模型去预测它，也就是评论家模型 Critic Model。其设计和初始化方式也有很多种，例如和 Actor Model 共享部分参数，或从奖励模型 Reward Model 初始化而来等等。其结构与 Actor 和 Reference 模型相似，但多了一层 Value Head（通常为线性层），用于基于输出预测总收益，如图 3-4 左所示。

对于 Reward Model，它用于计算生成 token 的即时收益，已经在前一步训练完毕，可以得到及时收益的真值。在 RLHF 过程中，它的参数是冻结的。

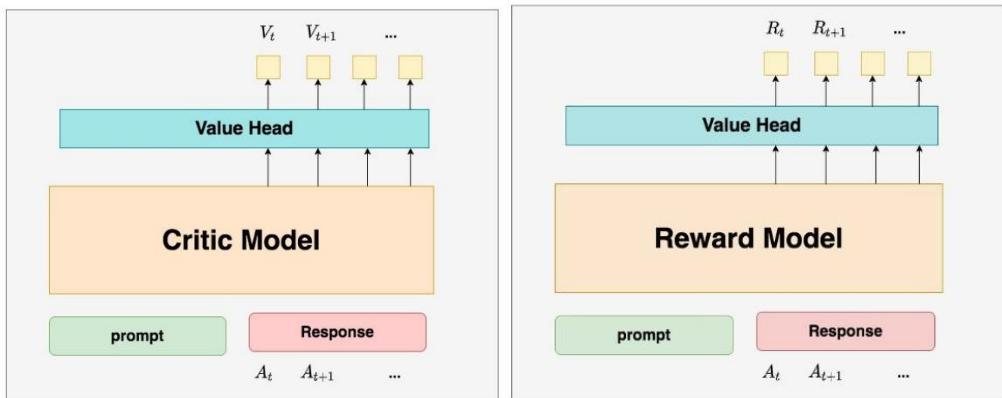


图 3-4 评论家模型和奖励模型的结构

在同一 batch 中，每一个输出所得到的收益 V 都是不同的，为了表征当前这个输出相对于其他输出的好坏程度，引入一个优势函数 $A(s,a)$ ，其定义为采取动作 a 在状态 s 下的收益与在状态 s 下的平均策略的收益之差。

在 PPO 中，采用 GAE (Generalized Advantage Estimation, 广义优势估计) 来计算优势，其表达式如下：

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

将每个时间步的TD残差加权求和，权重由 $\gamma\lambda$ 和 l 计算得到

其中， δ_{t+l} 是 TD 残差：

$$\delta_{t+l} = r_{t+l} + \gamma V(s_{t+l+1}) - V(s_{t+l})$$

状态 $t+l$ 的即时收益

γ 是折扣因子， λ 是介于 0 和 1 之间的参数，用于控制利用和探索之间的权衡。

3.1.2 PPO 训练过程

PPO 旨在通过策略梯度方法优化策略，同时限制每次策略更新的幅度，以保证训练过程的稳定性。

具体步骤如下：

1. 策略初始化：

初始化一个策略网络 π_θ ，其中 θ 是模型参数。

2. 数据收集：

使用当前策略 π_θ 生成一批文本输出，收集环境反馈，由 Reward Model 给出。

3. 计算优势估计：

计算每个生成动作（如生成的每句话）的优势值 A_t ，衡量其相对于平均水平的表现。

4. 策略更新：

使用裁剪的目标函数限制每次更新的步幅，最小化策略更新导致的性能波动，确保新策略 $\pi_{\theta_{\text{new}}}$ 不会偏离旧策略 $\pi_{\theta_{\text{old}}}$ 过多。

PPO 的损失函数形式为：

$$\mathcal{L}_{PPO}(\theta) = \mathbb{E}_{q \sim \rho(Q), o \sim \pi_{\theta_{\text{old}}}} \frac{1}{|o|} \min \left[\frac{\pi_{\theta}(o_t | q, o_{:t})}{\pi_{\theta_{\text{old}}}(o_t | q, o_{:t})}, \text{clip} \left[\frac{\pi_{\theta}(o_t | q, o_{:t})}{\pi_{\theta_{\text{old}}}(o_t | q, o_{:t})}, [1 - \epsilon, 1 + \epsilon] \right] \right] A_{\pi}(q, o_t)$$

A为状态q下采取动作
轨迹o_t的优势函数

限制更新幅度，防止参数变化过大

这个比值为新策略与旧策略在相同状态下
采取相同策略的概率之比

clip的范围，超出此范围的
会被强制缩放

其中

- 数据集: $\mathcal{D} = \{(q_i, o_i)\}_{i=1}^n$, 来自旧策略 $\pi_{\theta_{\text{old}}}$ 的采样
- 优势函数估计: $\hat{A}_t(k) = \delta_t + \gamma \lambda \hat{A}_{t+1}(k-1) \delta_t = r_t + \gamma v_w(s_{t+1}) - v_w(s_t)$
- $r_t = \pi_{\theta_{\text{rm}}}(q, o_{:t}) - \beta \text{KL}[\pi_{\theta} || \pi_{\theta_{\text{ref}}}]$ $\text{KL}[\pi_{\theta} || \pi_{\theta_{\text{ref}}}] = \log \frac{\pi_{\theta}(o_t | q, o_{:t})}{\pi_{\theta_{\text{ref}}}(o_t | q, o_{:t})}$
- 符号说明: 价值模型 v_w 、奖励模型 $\pi_{\theta_{\text{rm}}}$ 、策略模型 π_{θ} 、参考模型 $\pi_{\theta_{\text{ref}}}$ 两个策略之间的KL散度

导出这一损失函数的过程涉及大量的数学推导，此处不再展开，希望深入了解的可以查阅相关链接和参考文献。

不是有两个模型要训练吗，怎么只有一个损失函数，另一个模型怎么办？这里讨论的 PPO 损失函数是用来训练 Actor Model 的，该函数旨在优化策略，使其能够产生更高的预期回报；对于 Critic Model 的训练，则通过最小化价值函数预测值与实际回报之间的差异来实现，例如，常用的均方误差损失函数：

$$L_{VF} = \mathbb{E} [(V(s_t) - (r_t + \gamma V(s_{t+1})))^2]$$

其中， $V(s_t)$ 是模型对状态 s_t 的价值估计， r_t 是在状态 s_t 获得的即时奖励， γ 是折扣因子，

$V(s_{t+1})$ 是下一个状态 s_{t+1} 的价值估计。

在实践中，这个 γ 通常设置为 0.9 或 0.99，取决于预期生成的文本长度。

5. 迭代训练：

重复数据收集和策略更新的过程，多轮迭代后，策略逐步优化，更加符合预期的奖励信号。

以上过程的直观展示如图 3-5 所示。

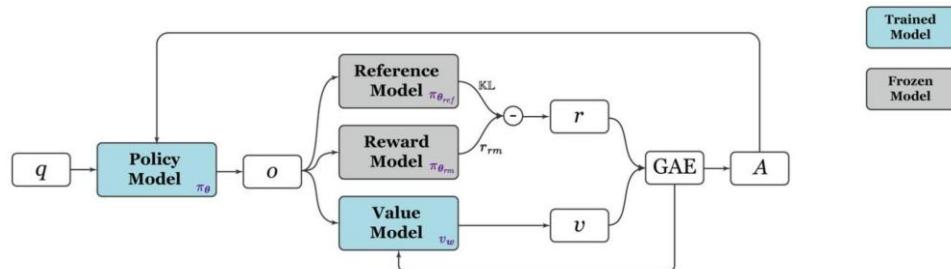


图 3-5 PPO 的优化过程

3.1.3 PPO 的优化: DPO 与 RLOO

PPO 需要同时运行四个类似的模型，对于大模型而言，需要同时训练两个大模型和推理另两个大模型，这无疑对算力提出了巨大的需求。因此，在一些模型中，采用了简化的方法 DPO (Direct Preference Optimization, 直接偏好优化)，只需要同时训练 Actor 和推理 Reference 这两个模型，其示意图如图 3-6 所示。

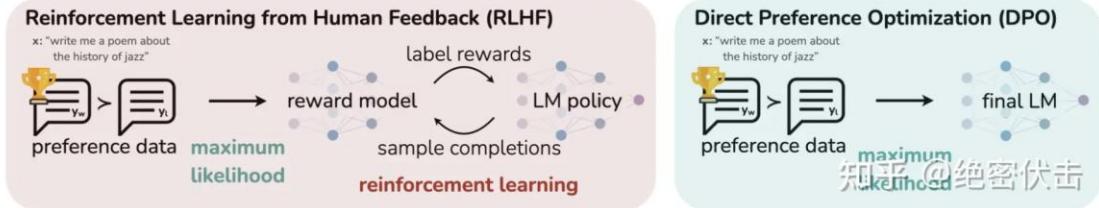


图 3-6 PPO 与 DPO 的对比

DPO 直接优化用户或专家的偏好，而非传统的累积奖励。在 DPO 中，通过对不同决策序列或策略，并根据用户或专家的偏好来优化模型，使得最终的策略能够更好地符合预期的行为。对于一个输入，分别收集较好的输出 (win) 和较差的输出 (lose)，DPO 的优化目标是最大化当前策略与参考策略在输出 win 与 lose 的概率比之差 (越大越好) 的期望，希望 LLM 输出的回答的评分能尽可能高，同时不要偏离参考太多。

同样忽略推导过程地给出其损失函数如下：

$$\max_{\pi_\theta} \left\{ \mathbb{E}_{(x, y_{\text{win}}, y_{\text{lose}}) \sim D} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_{\text{win}}|x)}{\pi_{\text{ref}}(y_{\text{win}}|x)} - \beta \log \frac{\pi_\theta(y_{\text{lose}}|x)}{\pi_{\text{ref}}(y_{\text{lose}}|x)} \right) \right] \right\}$$

数据集 D sigmoid 当前策略与参考策略生成win的输出的概率比
超参数

DPO 的优化过程是：

1. **数据收集：**从环境中收集数据，包括输入 x 和对应的 win 和 lose 的输出。
2. **策略更新：**使用上述损失函数直接更新策略 π_θ ，使策略更倾向于选择赢的输出，同时减少选择输的输出的概率。
3. **迭代优化：**重复数据收集和策略更新的过程，直到策略收敛或达到预设的迭代次数。

PPO 的另一种优化是 RLOO (REINFORCE Leave One-Out)。其主要思想是认为奖励只应该基于最终的完整答案，而对中间 token 的奖励建模是不必要的。因此，相比于 PPO，去除了 Critic Model，从而减少算力需求。

对于去除了 Critic Model 后的优势计算，其采用的思路是：对一个输入 prompt，采样多个回答，然后用留一法，即当前样本的奖励减去其他样本的奖励均值。这与后面将要提到的 GRPO 算法思路相似，只是 GRPO 是用当前样本的奖励减去所有样本的奖励均值。RLOO 的损失函数是：

$$\frac{1}{k} \sum_{i=1}^k [R(y_{(i)}, x) - \frac{1}{k-1} \sum_{j \neq i} R(y_{(j)}, x)] \nabla \log \pi(y_{(i)}|x) \quad \text{for } y_{(1)}, \dots, y_{(k)} \stackrel{i.i.d.}{\sim} \pi_\theta(\cdot|x)$$

采样数 第*i*个采样 其余样本的平均奖励 对策略参数求梯度
策略

图 3-7 展示了 PPO 与 RLOO 的直观对比：

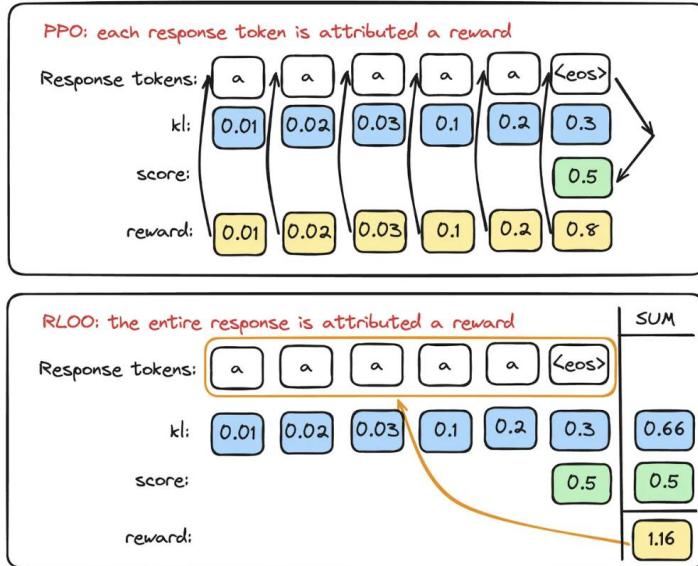


图 3-7 PPO 与 RLOO 的对比

3.2 群体相对策略优化 GRPO 与 R1-Zero

DeepSeek-R1-Zero 模型直接对基础模型进行强化学习训练，而不依赖任何 SFT 数据。其中，使用的强化学习策略是**群体相对策略优化 (Group Relative Policy Optimization, GRPO)**，同样放弃了 Critic Model，而是从一组输出的评分分数（也就是格式奖励和正确性奖励之和）中估计基线，降低了计算成本。

对于每个问题 q ，GRPO 从旧策略 $\pi_{\theta_{old}}$ 中采样一组输出 $\{o_1, o_2, \dots, o_G\}$ ，然后通过最大化以下目标来优化策略模型：

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^G \sim \pi_{\theta_{old}}(O|q)] \left[\frac{1}{G} \sum_{i=1}^G \left(\min \left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)} \right) A_i, \text{clip} \left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}, 1 - \varepsilon, 1 + \varepsilon \right) A_i \right) - \beta D_{KL}(\pi_\theta || \pi_{ref}) \right], \quad (1)$$

裁剪概率比，避免样本分布差异过大，保证训练稳定
优势
保证新旧策略差异不大，否则会惩罚

对一个问题，采样多个输出

$$D_{KL}(\pi_\theta || \pi_{ref}) = \frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - \log \frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - 1, \quad (2)$$

where ε and β are hyper-parameters, and A_i is the advantage, computed using a group of rewards $\{r_1, r_2, \dots, r_G\}$ corresponding to the outputs within each group:

$$\text{优势函数 } A_i = \frac{r_i - \text{mean}(\{r_1, r_2, \dots, r_G\})}{\text{std}(\{r_1, r_2, \dots, r_G\})}. \quad (3)$$

在训练时，使用同一问题的多个采样输出的平均奖励作为 Baseline，更新策略使得超过 baseline 的回答生成概率更大，但也通过 clip 和 KL 散度惩罚项作为限制，以免策略的更新幅度过大。将 PPO 与 GRPO 进行对比，如图 3-8 所示。

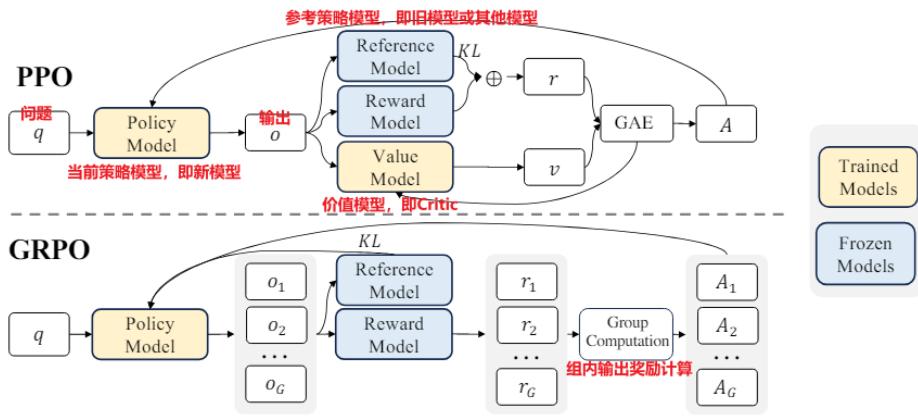


图 3-8 PPO 与 GRPO 的对比

图 3-9 为在社交媒体上找到的关于 GRPO 的详细介绍：

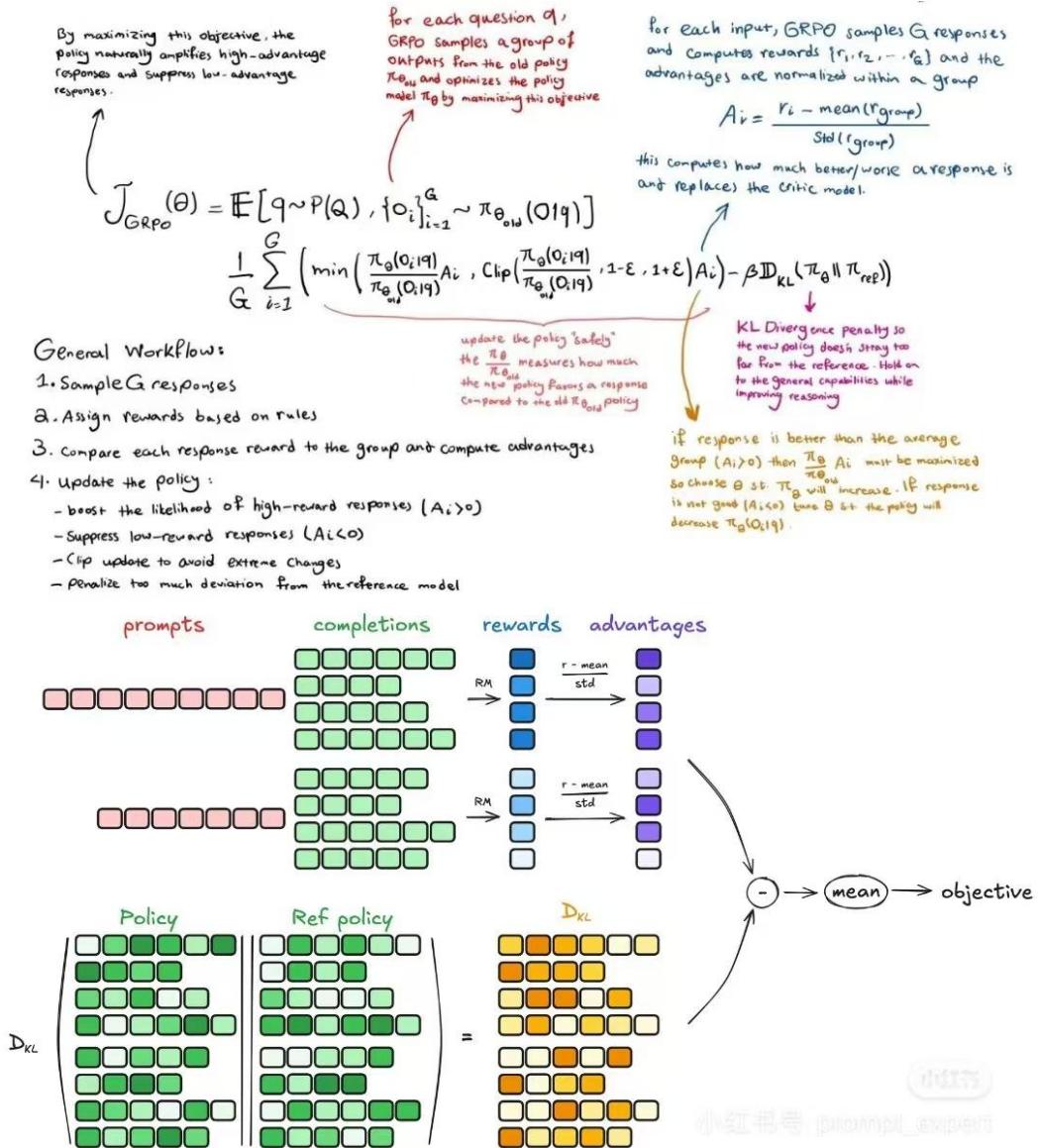


图 3-9 GRPO 的另一介绍示意图

在 DeepSeek-R1 的技术报告中，其采用了这样的提示词：

A conversation between User and Assistant. The user asks a question, and the Assistant solves it. The assistant first thinks about the reasoning process in the mind and then provides the user with the answer. The reasoning process and answer are enclosed within <think> </think> and <answer> </answer> tags, respectively, i.e., <think> reasoning process here </think> <answer> answer here </answer>. User: **prompt**. Assistant:

图 3-10 DeepSeek-R1 训练时的提示词

在训练过程中，还观察到了有趣的现象，即“顿悟”时刻（aha moment），模型学会了通过重新评估其初始方法来为问题分配更多的思考时间。这揭示了 RL 在解锁人工智能新水平方面的潜力，为未来更加自主和适应性强的模型铺平了道路。

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 ...

$$(\sqrt{a - \sqrt{a+x}})^2 = x^2 \implies a - \sqrt{a+x} = x^2.$$

Rearrange to isolate the inner square root term:

$$(a - x^2)^2 = a + x \implies a^2 - 2ax^2 + (x^2)^2 = a + x \implies x^4 - 2ax^2 - x + (a^2 - a) = 0$$

...

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 ...

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: ...

...

图 3-11 R1 训练中的“顿悟”时刻

R1-Zero 所使用的 RL 和 RLHF 有什么区别？

其核心区别在于仅在组内进行计算和自我迭代，而没有人类反馈的标签。模型自己从不断的反馈中（注意只有格式奖励和正确性奖励这两个规则）学习到了“我应该在回答前多思考一会，这样更有可能得到更高的分数”这个促成长思维链的核心策略。这一自我进化中最显著的方面之一是随着测试时间计算的增加，出现了复杂的行为，例如反思和探索。这些行为并不是由程序定义的，而是模型与强化学习环境互动的结果。

3.3 过程奖励模型 PRM 与蒙特卡罗树搜索 MCTS

过程奖励模型（Process Reward Model, PRM）与蒙特卡罗树搜索（Monte Carlo Tree Search, MCTS）是 DeepSeek-R1 技术报告中提到尝试过但失败的两种强化学习策略。然而在其他研究中，这两种策略经常被使用并可能有效。

3.3.1 PRM

过程奖励模型 PRM 是 OpenAI 在 2023 年 Let's Verify Step by Step 论文^[11]中提出的，旨在生成过程中，分步骤对每一步进行打分，是更细粒度的奖励模型，可以引导模型采用更好的方法来解决推理任务。然而，在实践中，遇到了三个难点：

1. 在一般推理中难以明确定义细粒度步骤；

2. 难以确定当前中间步骤是否正确。使用模型进行自动注释可能无法产生令人满意的结果，而手动注释不利于扩展。

3. 可能导致奖励攻击，并且重新训练奖励模型需要额外的训练资源，这会使整个训练流程复杂化。

什么叫奖励攻击？就是模型实际上没学到什么东西，但是通过技巧把奖励得分做得很髙。例如，在强化学习过程中，假如模型变成了不停地输出大量无用的思维链，再像非推理模型那样直接生成结果，就是一种奖励攻击。

虽然 PRM 展示了重新排序模型生成的前 N 个响应或辅助引导搜索的良好能力，如图 3-12 所示。然而，在大规模强化学习过程中，其优势与引入的额外计算开销相比来说十分有限。相比而言，在 5.2 节将会提到的测试时扩展（TTS）中，PRM 展示了它的应用潜力。此外，一些学者也认为，PRM 提供了比较稠密的监督信号，可以使训练更稳定或收敛得更快，或者使得一些直接得到正确结果较为困难的任务（例如很复杂的推理任务）拥有“过程分”从而一步步接近正确方向，不应被完全放弃。

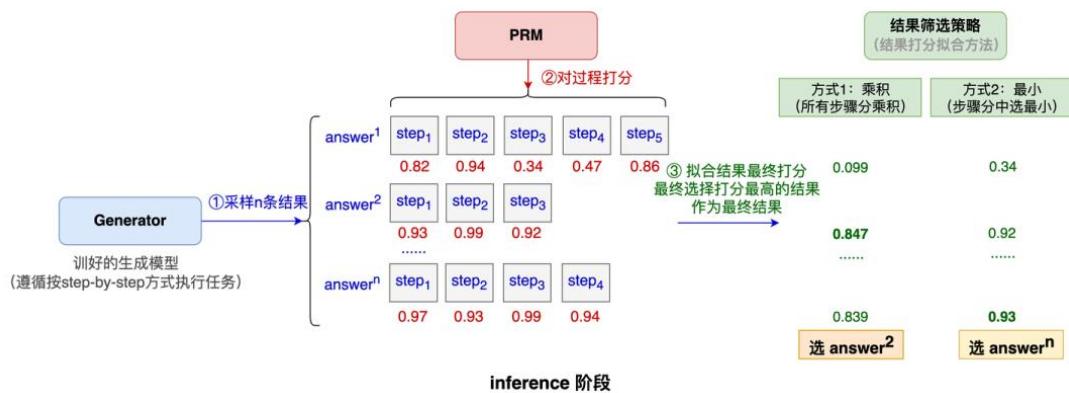


图 3-12 PRM 示意图

3.3.2 MCTS

蒙特卡罗树搜索 MCTS 是 1987 年 Bruce Abramso 提出的一种树搜索算法。如图 3-13 所示，MCTS 大概可以被分成四步：选择 (Selection)，拓展 (Expansion)，模拟 (Simulation)，反向传播 (Back Propagation)。在选择阶段，递归选择最优子节点，当到达一个叶子结点时，若叶子结点不是终止节点，就创建其他节点，选择其一进行拓展，从拓展节点开始，进行模拟输出。直到输出结束后，根据模拟结果，反向传播更新当前序列，该方法的好处在于：可以在不训练模型的情况下，通过增加模型的推理时间，增强模型的推理性能。事实上，该方法也是后面所介绍的 TTS 中的一部分思路来源。

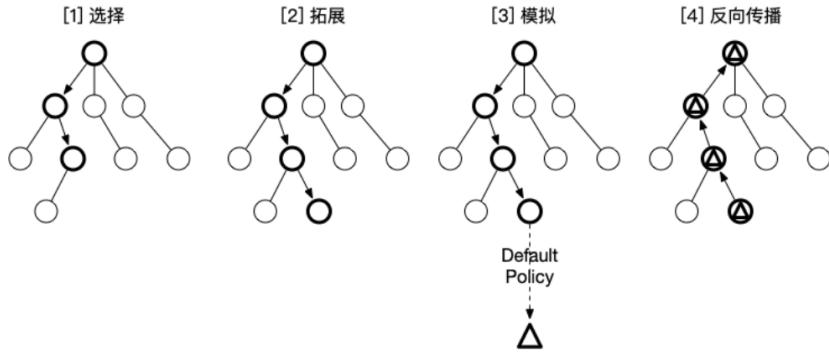


图 3-13 MCTS 示意图

受到 AlphaGo 和 AlphaZero 启发，如果对大模型使用该方法来增加推理时间，该方法会将答案分解为较小的部分，以允许模型系统地探索解决方案。为此，先提示模型去生成多个标签，这些标签对应于搜索中所需的具体推理步骤。对于训练，首先通过 MCTS，在预训练的价值模型引导下，使用收集的提示词去找到答案。随后，使用生成的问答对来训练演员模型和价值模型，不断迭代该过程。

然而，这种方法在扩展到训练时，遇到了 2 个挑战：

1.与国际象棋不同，国际象棋的搜索空间相对明确，而 token 的生成呈现出指数级的搜索空间。为了解决这个问题，为每个节点设置了最大拓展限制，但这可能导致模型陷入局部最优；

2.价值模型直接影响生成的质量，因为它指导搜索过程的每一步。但训练一个好的价值模型本身就很困难，这使得模型难以迭代改进。虽然 AlphaGo 的核心成功依赖于训练价值模型以逐步提高其性能，但由于 token 生成的复杂性，这一原则不适用于 DeepSeek 的设置。

总之，虽然 MCTS 在与预训练的价值模型配对时，可以在推理过程中提高性能，但通过自我搜索迭代提升模型性能仍然是一个重大挑战。

值得注意的是，在地学领域，已有研究尝试应用 MCTS 进行地学代码生成的优化，称为 GeoAgent^[7]，该研究由意大利和法国的一些学者完成，展示了强化学习在地学代码生成中的可能价值，其架构如图 3-14 所示。

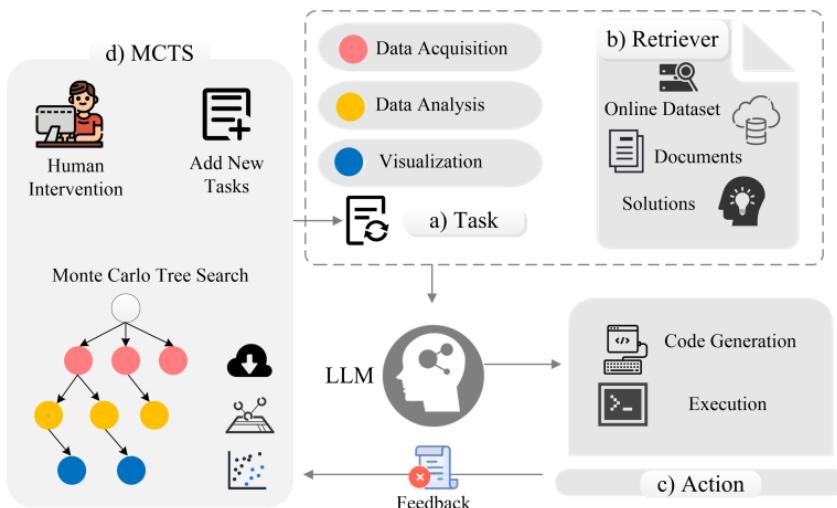


图 3-14 GeoAgent 的总体架构

GeoAgent 集成了代码解释器、静态分析和检索增强生成（RAG），服务于地学代码生成任务，并利用 MCTS 进行动态任务调整和优化。在代码生成过程中，RAG 将外部知识引入 LLM，特别是在使用不太常见的地理空间库时。虽然 RAG 通过整合相应的库文档增强了特定任务的编码能力，但由于次优检索器的偶尔无关增强，它可能会对较常用的 Python 模块的性能产生负面影响。为了解决这一问题，通过在 MCTS 框架内使用执行反馈来识别子任务之间的依赖关系并动态优化它们，确保每个代码段在逻辑上一致且与先前步骤紧密相关。

具体而言，GeoAgent 在 MCTS 框架内执行并收集执行反馈，MCTS 迭代地探索和评估多个代码候选方案，以优化选择最有希望的解决方案，而 LLM 则作为智能推理器，诊断和修复错误，其示例如图 3-15 所示。

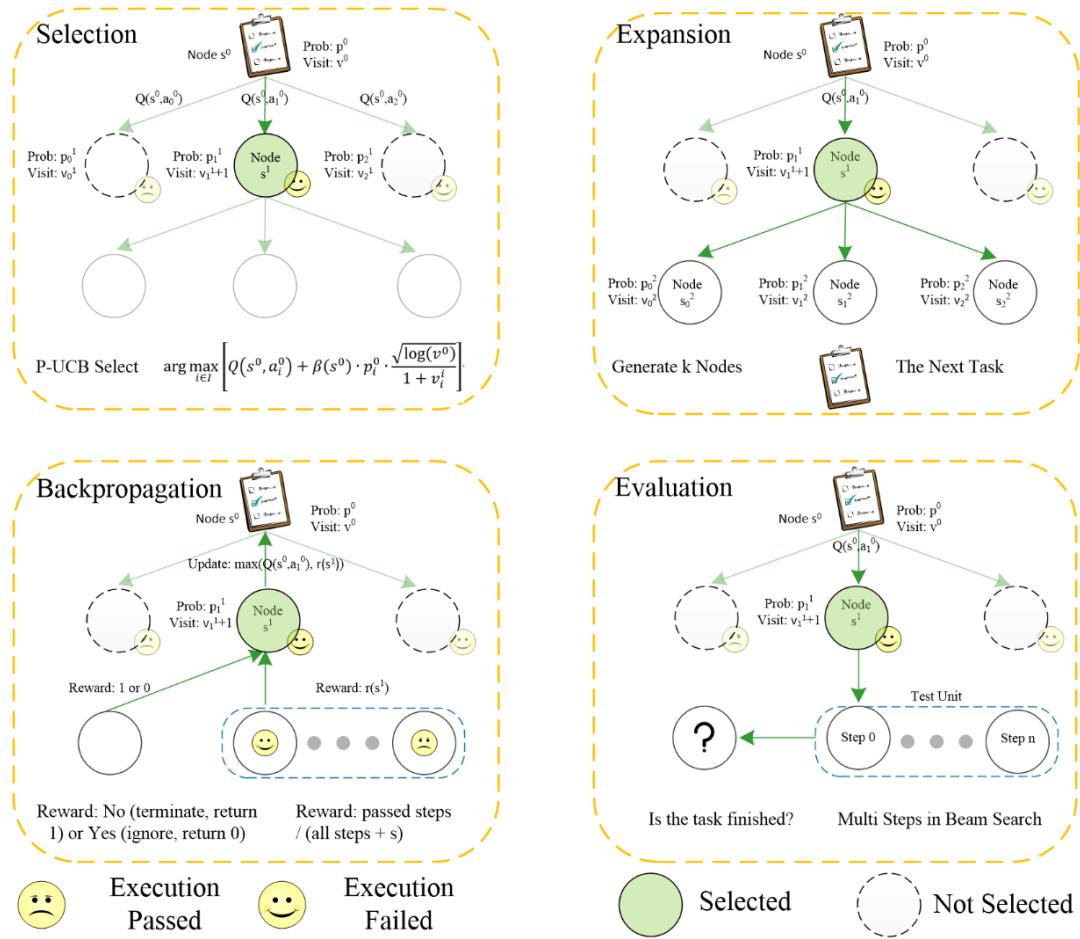


图 3-15 GeoAgent 在地学代码生成中应用 MCTS 的示例

在通用代码领域，o1 coder^[12]也结合 PRM 与 MCTS，做了类似的工作，其主要流程如图 3-16 所示。该框架包括测试用例生成器（TCG）用于标准化代码测试，使用 MCTS 生成包含推理过程的代码数据，并迭代微调策略模型，首先生成伪代码，然后生成完整的代码。其主要步骤包括

1. 训练 TCG，对 DeepSeek-1.3B-Instruct 模型进行 SFT 和 DPO
2. 在原始代码数据集上运行 MCTS 以获得推理数据集。
3. 使用推理数据 SFT 初始化策略模型

4. 使用推理数据初始化 PRM
5. PRM 基于推理过程计算过程奖励，TCG 提供结果奖励，使用 MCTS 进行策略模型更新
6. 使用更新后的策略模型生成新的推理数据，将其添加到现有数据集中，形成更新后的数据集，再次用于微调 PRM，重复 4-6 步，以此进行自我迭代更新。

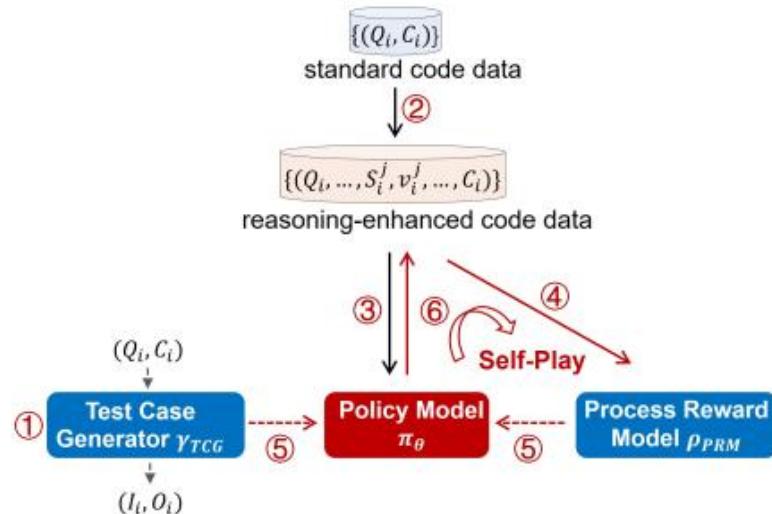


图 3-16 O1-coder 框架的主要流程

4 长思维链 (Long Chain-of-Thought)

4.1 思维链的本质区别

思维链 (Chain-of-Thought, CoT) 让模型在输出最后答案之前，以自然语言的形式生成思路或推理链，使得结果更可靠，对于人类更具可解释性。通常 CoT 是通过提示工程的形式实现的，一个最简单的示例是在所提的问题后加入一句话“Let’s think step by step”。

以 o1 系列和 R1 系列为为代表的推理模型更进一步地取得了重大突破，将思维链“内置”于模型中。这些模型的一个关键特点是能够通过扩展推理时间和使用更长的 CoT，从而实现更长、更结构化的推理过程。其思维特点包括：迟疑（标注不确定的内容，后续再验证）；探索（考虑多种不同的可能性）；回溯（返回来再次思考之前的内容）；阶段性总结（总结前面一阶段的思考结果）和最后验证（输出答案之前再最后全面检查一遍）。

但事实上，它们之间的本质区别在于，长思维链推理型 LLM 在回答之前先进行思考，是两个不同的阶段，而普通的 LLM 即使通过提示工程使它以思维链形式回答，但思维链其实仍然是回答的一部分，并不是一个独立阶段。前文的图 2-3 实际上也展示了这一本质不同。

4.2 长思维链的影响因素

在 Demystifying Long Chain-of-Thought Reasoning in LLMs 论文中^[4]，全面研究了 SFT、RL 等技术选择对长 CoT 的影响。此处，使用的 RL 策略为 PPO。

1. SFT 对长 CoT 的影响：带有长 CoT 的 SFT 具有更高的上限，并且使进一步 RL 变得更容易，如图 4-1 所示。SFT 的初始化很重要：高质量的长 CoT 显著提高了泛化能力和 RL 收益。

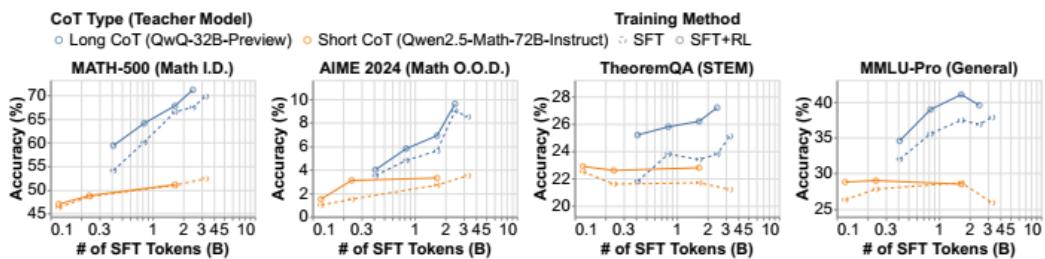


Figure 1. Scaling curves of SFT and RL on Llama-3.1-8B with long CoTs and short CoTs. SFT with long CoTs can scale up to a higher upper limit and has more potential to further improve with RL.

图 4-1 SFT 对长 CoT 的影响

2. RL 奖励函数对长 CoT 的影响：CoT 长度不总是稳定扩展。奖励塑造可稳定和控制 CoT 长度，提高准确性。余弦奖励（如图 4-2 所示）可通过调整超参数鼓励不同长度缩放行为。模型可能需更多训练样本以利用更大上下文窗口。长度奖励会被破解，但可被重复惩罚缓解。不同类型奖励和惩罚有不同最佳折扣因子。

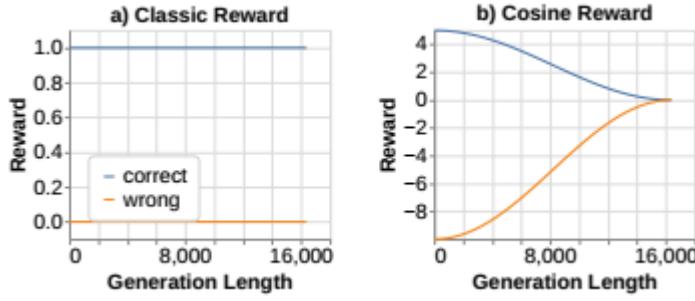


图 4-2 经典奖励函数和余弦奖励函数

3. 带有噪声但更多样的数据对长 CoT 的影响：可验证的奖励信号，如基于真实答案的信号，对于稳定长链 CoT 强化学习至关重要。添加噪声数据（有噪声但覆盖更全面的数据集）到 SFT 可平衡不同任务的性能。为了从带有噪声的可验证数据中获得奖励信号，基于规则的验证器在过滤短答案的提示集表现最佳，如图 4-3 所示。然而，在不同领域设计这些规则和整理提示集仍然是一项劳动密集型任务。

Table 3. 不同验证器和提示过滤方法下的 RL 性能。此处的所有模型都是从 Llama-3.1-8B 微调而来。“MATH Baseline”是在表 2 中仅使用 MATH 数据集通过 SFT 和 RL 训练的模型。其他模型则是通过从 QwQ-32B-Preview 蒸馏的 SFT 和不同设置的 RL 训练而来。

Prompt Set	Verifier Type	MATH 500	AIME 2024	Theo. QA	MMLU Pro-1k
MATH Baseline		59.4	4.0	25.2	34.6
SFT Initialization		46.6	1.0	23.0	28.3
Unfiltered	Rule-Based	45.4	3.3	25.9	35.1
	Model-Based	47.9	3.5	26.2	40.4
Filtered	Rule-Based	48.6	3.3	28.1	41.4
	Model-Based	47.9	3.8	26.9	41.4

图 4-3 不同验证器和提示过滤方法下的 RL 性能

4. Long CoT 能力的起源和 RL 挑战：长 CoT 的能力实际上在预训练阶段已经存在于基础模型中，但通过 RL 加强这些行为需要严格的条件，例如一个强大的基础模型。从小规模基础模型（文中为 Qwen2.5-Math-7B）进行类似 R1-Zero 的 RL 可以提高性能，但并不一定能得到模型对长 CoT 核心能力的“涌现”（在 R1 的技术报告中称为“Aha Moment”），如图 4-4 所示。

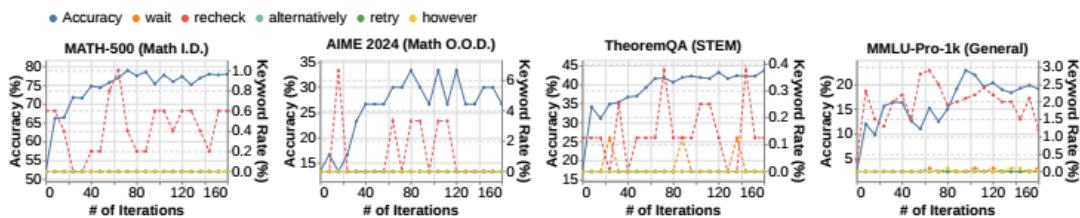


图 4-4 对 Qwen2.5-Math-7B 进行 RL 过程中的性能和长 CoT 核心关键词出现率

总结其核心发现：

1. SFT 可以简化训练流程，为 RL 提供更好的基础；
2. RL 不是总能顺利提高思维链的长度和复杂性，可以通过长度奖励函数鼓励复杂推理行为；
3. 可验证奖励信号对于稳定长链思维推理的 RL 至关重要，如果再结合使用包含噪声的、从网络提取的解决方案的数据，可以提高模型完成分布外任务（即训练集中没有见过的任务）的能力；
4. 长链推理的核心技能（如分支和错误验证）在基础模型中已经存在，但通过 RL 有效地激励这些技能以解决复杂任务需要仔细的设计，并可能需要足够强大的基础模型。

5 知识蒸馏与小模型优化

5.1 知识蒸馏（Knowledge Distillation）

5.1.1 知识蒸馏的不同方法

知识蒸馏的方式就是利用强大的“教师模型”的预测概率分布作为软标签对小规模的“学生模型”进行训练，从而在保持较高预测性能的同时，极大地降低了模型的复杂性和计算资源需求，实现模型的轻量化和高效化。模型蒸馏有三种不同的做法：

1. 数据蒸馏

数据蒸馏过程中，教师模型首先对问题做出回答，随后使用这些回答来训练学生模型。DeepSeekR1-Distill 系列模型就是使用 DeepSeek-r1 生成的 80 万条数据，在 Qwen2.5 和 LLaMA3 等模型基础上直接进行 SFT 而得到的。

2. 软标签蒸馏（Logits 蒸馏）

软标签指的是神经网络在应用 softmax 函数之前的原始输出分数。在这一过程中，学生模型被训练以模仿教师模型的软标签，而不仅仅是其最终预测结果，从而保留了教师模型更多的信息。其过程如图 5-1 所示。

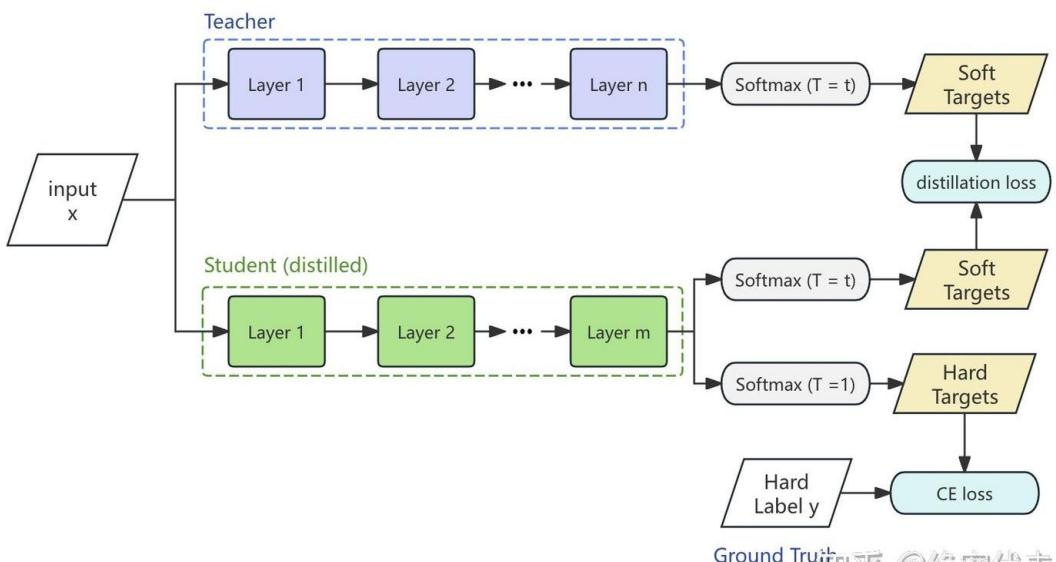


图 5-1 软标签蒸馏的过程示意图

损失函数：需要 2 个损失函数，一个是 KL 散度损失，用于衡量教师和学生模型两个概率分布之间的差异；另一个是交叉熵损失函数，用于衡量学生模型输出与真实标签之间的损失。而蒸馏的总损失为二者之和。

$$L_{\text{distill}} = T^2 \cdot \text{KL}(\mathbf{p}_t \| \mathbf{p}_s)$$

$$L_{\text{task}} = \text{CrossEntropy}(\mathbf{y}_{\text{true}}, \mathbf{p}_s^{\text{raw}})$$

$$L_{\text{total}} = \alpha L_{\text{distill}} + (1 - \alpha) L_{\text{task}}$$

3. 特征蒸馏

特征蒸馏是蒸馏教师模型的中间层，也就是学生模型直接学习教师模型的中间层信息，较软标签 Logits 蒸馏更为彻底，保留更多的信息，但同时也更为复杂，如图 5-2 所示。

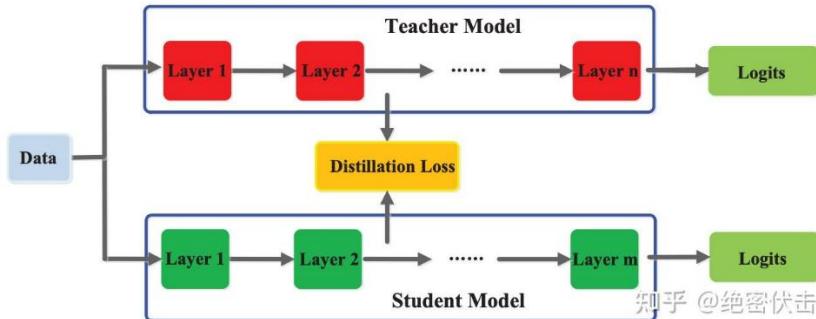


图 5-2 特征蒸馏的过程

5.1.2 DeepSeek-R1 的知识蒸馏

在 DeepSeek 中，就是采用 R1 生成的长 CoT 对 Qwen 和 LLaMA 等小模型进行微调，使它们也具备接近 R1 的能力。

那么这种蒸馏与一般的监督微调在方法上有什么区别？还真没有。

强化学习与知识蒸馏的效果比较：基于 Qwen-32B-Base 模型进行与 R1-Zero 一致的大规模强化学习和知识蒸馏，发现蒸馏模型的效果更好，如图 5-3 所示。

Model	AIME 2024		MATH-500	GPQA Diamond	LiveCodeBench
	pass@1	cons@64	pass@1	pass@1	pass@1
QwQ-32B-Preview	50.0	60.0	90.6	54.5	41.9
DeepSeek-R1-Zero-Qwen-32B	47.0	60.0	91.6	55.0	40.2
DeepSeek-R1-Distill-Qwen-32B	72.6	83.3	94.3	62.1	57.2

图 5-3 对 Qwen-32B 模型进行强化学习与蒸馏的效果比较

DeepSeek-R1 技术报告中的结论：

首先，将更强大的模型蒸馏到较小的模型中可以获得优异的结果，而较小的模型依赖于本文提到的大规模强化学习需要巨大的计算资源，并且可能无法达到蒸馏的效果。其次，虽然蒸馏策略既经济又有效，但要超越智能的边界仍然可能需要更强大的基础模型和更大规模的强化学习。

5.2 在有限条件下改善长思维链推理模型

即使 DeepSeek-R1 相较于 GPT 系列等传统模型已经显著减少了成本，但对于我们的科研团队而言，其算力需求和成本依然很高的（例如，R1 使用了 2048 块 H800 和数百万美元计算成本）。如何在算力有限的情况下更好地做出类似于 R1 这样的成果呢？当前的研究表明，可能有以下思路：

5.2.1 蒸馏与 RL

R1-distilled 蒸馏模型比 DeepSeek-R1 小得多，但仍能获得强大的推理性能。但它们的蒸馏过程使用了 80 万个 SFT 样本，同样需要相当大的计算资源。

不过，在 Sky-T1 模型的工作中，使用 17K SFT 样本训练了一个开源 32B 模型，总算力费用仅为 450 美元（8*H100），却同样达到了与 o1 相当的效果。在条件受限的情况下，较小、针对性的微调工作依然能够取得优异成果。该团队还在 7B 模型上尝试了类似 R1 的多步 RL 和 SFT 过程（但使用的是 PRIME 与 RLOO 算法）得到 Sky-T1-7B，在 R1 蒸馏模型上通过 RLOO 强化学习算法得到 Sky-T1-mini，揭示了蒸馏和 RL 的潜力，其评测结果如图 5-4 所示。

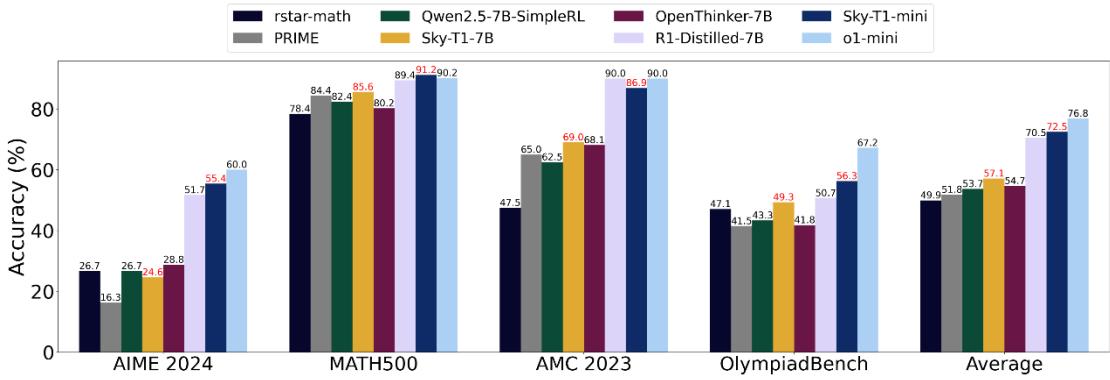


图 5-4 Sky-T1 系列与 baseline 模型在数学问题上的能力对比

尽管已有研究提到通过 RL 激励长思维链推理可能需要更强大的基础模型，但 TinyZero 在仅有 0.5B 至 3B 参数的模型上复制了 DeepSeek-R1-Zero 的方法，也展现了一些自我验证能力（图 5-5），展示出推理可以通过纯 RL 在小模型中得到。3B 模型的算力成本不到 30 美元。

User: Using the numbers [19, 36, 55, 7], create an equation that equals 65.
Assistant: Let me solve this step by step.

<think> We need to find an equation using the numbers 19, 36, 55, and 7 exactly once, with basic arithmetic operations, that equals 65. One possible combination is $55 + 36 - 19 + 7$. Let's check: $55 + 36 = 91$, $91 - 19 = 72$, and $72 + 7 = 79$. However, this doesn't equal 65. Another combination is $55 + 36 + 7 - 19$. Let's check: $55 + 36 = 91$, $91 + 7 = 98$, and $98 - 19 = 79$. This also doesn't equal 65. After trying different combinations, I found that $55 + 36 + 7 - 19 = 79$, which is close but not equal to 65. Finally, I found that $55 + 36 - 19 + 7 = 79$, which is still not equal to 65. However, if we try $55 + 36 - 7 + 19$, we get $55 + 36 = 91$, $91 - 7 = 84$, and $84 + 19 = 103$, which is not equal to 65. After further trial and error, I found that $55 + 36 + 7 - 19 = 79$, which is still not equal to 65. However, if we try $55 + 36 - 7 - 19$, we get $55 + 36 = 91$, $91 - 7 = 84$, and $84 - 19 = 65$. This equals 65. </think>
<answer> $55 + 36 - 7 - 19$ </answer>

知乎 @段小草

图 5-5 TinyZero 在 3B 模型上展现出的长思维链推理能力

5.2.2 测试时扩展 TTS

TTS（Test-time Scaling）最早由谷歌团队提出^[8]，其含义是增加在推理时的时间和 token 消耗，即使模型进行更多思考，来提高模型的性能。

李飞飞团队通过仅在 1k 条高质量数据上对 32B 模型进行 SFT 得到 s1 模型^[9]，并测试了在不同的推理时间下的效果表现，展示出了推理时的时间消耗与结果之间也存在一定的正相关关系，如图 5-6 所示。

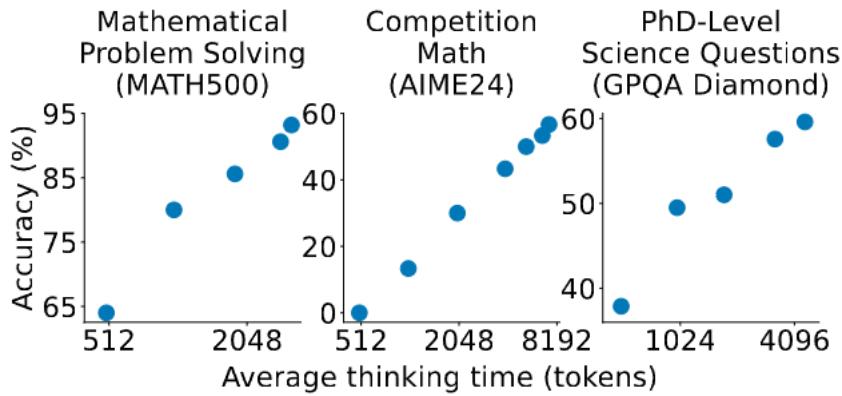


图 5-6 S1 模型性能与推理时间的关系

上海 AI 实验室的团队在 TTS 上进行了更深入的研究^[5]，比较了三种 TTS 策略：**BoN**（多个答案中选择最好的一个）、**束搜索**（在给定束宽度 N 和束大小 M 的情况下，首先生成 N 步，选择前 N/M 步进行后续搜索，在下一步中，策略模型为每个选定的前一步骤采样 M 步）和**多样化验证器树搜索 DVTS**（将搜索过程分为 N/M 个子树，每个子树独立使用束搜索进行探索），如图 5-7 所示。

选用的优化策略是在一定的问题、超参数和计算预算下，最大化模型输出与正确答案一致的期望概率。生成过程中，使用 PRM 引导模型一步步搜索。

$$\theta_{x,y^*(x)}^*(N) = \arg \max_{\theta} (\mathbb{E}_{y \sim \text{Target}(\theta, N, x)} [\mathbb{1}_{y=y^*(x)}]), \quad (2)$$

问题 x 计算预算 N θ 模型所产生的输出分布

其中 $y^*(x)$ 表示 x 的真实正确响应，而 $\theta_{x,y^*(x)}^*(N)$ 表示在计算预算为 N 的情况下，问题 x 的测试时计算最优缩放策略。

该研究的主要结论包括：

1.对于小模型，BoN 方法在简单问题上表现更好，而束搜索在困难问题上表现更好。对于中等规模模型，DVTS 在简单和中等问题上表现良好，而束搜索在困难问题上表现更好。对于较大的模型（72B），BoN 在所有难度级别上表现最佳；

2.通过最优的 TTS 策略，小模型可以超越大模型。例如，Llama-3.2-3B-Instruct 在 MATH-500 和 AIME24 上超越了 Llama-3.1-405B-Instruct。Qwen2.5-0.5B-Instruct 和 Llama-3.2-3B-Instruct 超越了 GPT-4o，DeepSeek-R1-Distill-Qwen-1.5B 超越了 o1-preview 和 o1-mini，DeepSeek-R1-Distill-Qwen-7B 超越了 o1 和 DeepSeek-R1；

3.TTS 与长 CoT 相比，在简单任务上比复杂任务更有效。TTS 在 MATH-500 和 AIME24 上优于 rStar-Math、Eurus-2、SimpleRL 和 Satori，但在 AIME24 上表现不如 DeepSeek-R1-Distill-Qwen-7B。

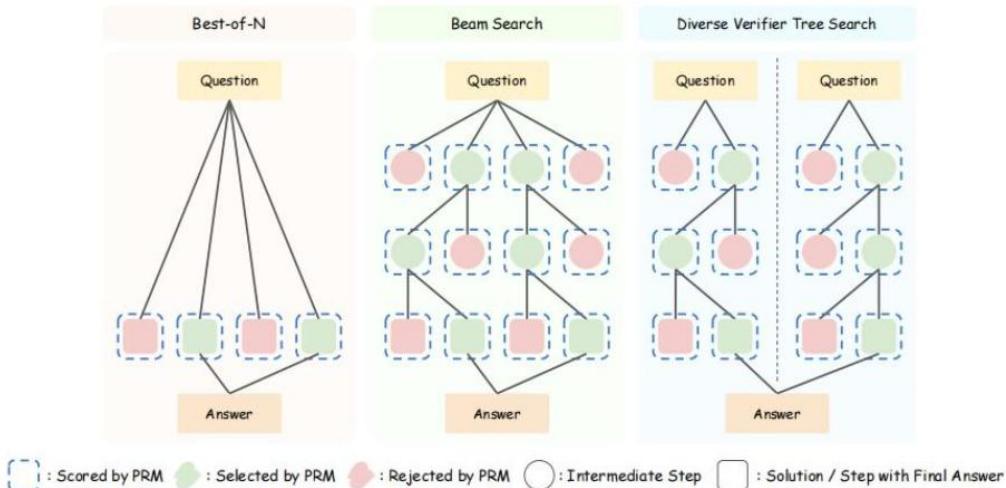


图 5-7 三种 TTS 策略的对比

5.2.3 超越传统 SFT

在论文 O1 Replication Journey: A Strategic Progress Report – Part 1 中^[6]，介绍了一种改进蒸馏（纯 SFT）过程的不同方法。

论文中的关键概念是“journey learning”，相比于传统的提供正确的思维路径的 SFT（文中称为“shortcut learning”），还包括了不正确的解决路径，让模型从错误中学习，如图 5-8 所示。通过这种方式，也观察到了模型加强自我修正能力，从而使推理模型变得更可靠。

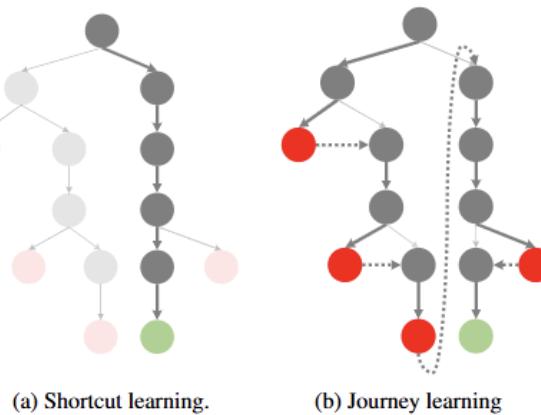


图 5-8 Journey learning 与传统 shortcut learning 的对比

此外，还有隐式推理（Latent Reasoning）等优化方案^[10]，学术界与工业界各类方法层出不穷，可参考相关文献并及时关注最新成果。

6 总结与思考

R1 一经发布便震惊世界，展现了长思维链推理模型的可能性，以及开源的价值（这才是真正的 OpenAI）。对其中的技术，个人提出以下总结：

1. 长思维链推理型模型很重要，学习和研究中不能掉队。GPT-5 也将与 o 系列合并，而不再是传统的短推理 LLM 了。

2. 当无限制堆大规模数据和模型的 Scaling Law 难以继续之后，后训练的过程显得尤为重要。特别是以 R1-Zero 为代表的强化学习技术，让模型能够自己学习如何回答问题，大大减少了对需要人工标注的 SFT 数据的需求。接下来学术界和产业界必将有大量的 R1 复现类工作和尝试 PRM、MCTS 等其他强化学习路线的工作。当然，其局限性在于 RL 收敛可能较为困难，需要设计合适的策略。

3. 强化学习优化模型需要一个足够强大的基础模型作为支撑，对于小模型而言，蒸馏则是一种更为可能的道路。蒸馏所用的数据更为重要，直接决定了蒸馏效果好坏。

4. 对于 SFT 和 RL，也有许多可以探索的降低成本、提高能力的路径，例如 5.2 中所举的一系列例子。

相关链接

使用与部署：

DeepSeek 官网：

<https://www.deepseek.com/>

Ollama 本地部署：

<https://ollama.com/library/deepseek-r1>

Ktransformers 本地部署优化：

<https://kvcache-ai.github.io/ktransformers/>

复现与类似尝试：

OpenR1 (Hugging Face 社区)：

<https://huggingface.co/open-r1>

TinyZero (UC Berkeley)：

<https://github.com/Jiayi-Pan/TinyZero>

simpleRL-reason (HKUST)：

<https://github.com/hkust-nlp/simpleRL-reason>

Sky-T1：

<https://novasky-ai.github.io/posts/sky-t1/>

S1：

<https://github.com/simplescaling/s1>

网络上的技术解读资料：

The Illustrated DeepSeek-R1 (原版及翻译版本)

<https://newsletter.languagemodels.co/p/the-illustrated-deepseek-r1>

<https://mp.weixin.qq.com/s/yfFxR01mCvPLgCRyxjlAGA>

DeepSeek-R1 技术笔记 (含图解和技术点介绍)

<https://mp.weixin.qq.com/s/7O0rHMwubamI5JUvfK9zw>

一文读懂 DeepSeek 的技术演进之路：DeepSeek-V3、DeepSeek-R1 和 Janus-Pro

<https://mp.weixin.qq.com/s/hwcH2Pnofdpr6pmAedMpKQ>

DeepSeek 是否有国运级创新？2 万字解读与硬核分析 V3/R1 的架构

<https://mp.weixin.qq.com/s/0n9lUH9WsDQTGnPcWeUkdA>

万字赏析 DeepSeek 创造之美：DeepSeek R1 是怎样炼成的？

<https://mp.weixin.qq.com/s/HB38prEMz275Rkj1g7mavA>

Understanding Reasoning LLMs (原版及翻译版本)

<https://magazine.sebastianraschka.com/p/understanding-reasoning-llms>

<https://zhuanlan.zhihu.com/p/22445775611>

【DeepSeek-R1 背后的技术】系列一：混合专家模型 (MoE)

https://blog.csdn.net/sinat_16020825/article/details/145429390

【DeepSeek-R1 背后的技术】系列二：大模型知识蒸馏 (Knowledge Distillation)

https://blog.csdn.net/sinat_16020825/article/details/145401293

【DeepSeek-R1 背后的技术】系列三：强化学习 (Reinforcement Learning, RL)

https://blog.csdn.net/sinat_16020825/article/details/145446510

几分钟了解 Deepseek 核心注意力机制 MLA

<https://zhuanlan.zhihu.com/p/23064167415>

从 PPO 到 Reinforce++, 再对比 GRPO

<https://zhuanlan.zhihu.com/p/22023807402>

RLHF 代码初见 01: 从 PPO、GRPO 到 REINFORCE+++

<https://zhuanlan.zhihu.com/p/23401911863>

图解大模型 RLHF 系列之：人人都能看懂的 PPO 原理与源码解读

<https://zhuanlan.zhihu.com/p/677607581>

大模型优化利器：RLHF 之 PPO、DPO

https://www.zhihu.com/tardis/bd/art/717010380?source_id=1001

RLHF 及其变体 Iterative DPO/RLOO/GRPO/REINFORCE 算法和工程分析

<https://zhuanlan.zhihu.com/p/714364995>

北大对齐团队深度硬核解读：OpenAI o1 开启「后训练」时代强化学习新范式

<https://blog.csdn.net/ys707663989/article/details/142732343>

一文读懂：混合专家模型 (MoE)-deepseek

<https://zhuanlan.zhihu.com/p/680190127>

多 Token 预测（MTP）

<https://zhuanlan.zhihu.com/p/18165397323>

[通俗易读]无痛理解旋转位置编码 RoPE

<https://zhuanlan.zhihu.com/p/8306958113>

深度解析 DeepSeek R1 蒸馏技术

<https://zhuanlan.zhihu.com/p/22833406186>

【手撕 NSA】DeepSeek 新作-原生稀疏注意力-超长文(附代码)

<https://zhuanlan.zhihu.com/p/24841366485>

下游应用：

DeepSeek 缝合 Claude，比单用 R1/o1 效果都好！GitHub 捞星 3k

<https://mp.weixin.qq.com/s/vhv4Eb5XoA2d4LKRqVRQag>

清华「DeepSeek 从入门到精通」正式发布！104 页超全解析

https://mp.weixin.qq.com/s/Igv5ZFoOX3BWC_sOiCcGkQ

深度学习与大模型基础知识教材：

Aston Zhang 等：动手学深度学习

<https://zh.d2l.ai/index.html>

毛玉仁、高云君等：大模型基础

<https://github.com/ZJU-LLMs/Foundations-of-LLMs>

Jay Alammar 等：Hands-On Large Language Models

<https://github.com/HandsOnLLM/Hands-On-Large-Language-Models>

参考文献

- [1] Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., ... & He, Y. (2025). Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. arXiv preprint arXiv:2501.12948.

- [2] Liu, A., Feng, B., Xue, B., Wang, B., Wu, B., Lu, C., ... & Piao, Y. (2024). Deepseek-v3 technical report. arXiv preprint arXiv:2412.19437.
- [3] Liu, A., Feng, B., Wang, B., Wang, B., Liu, B., Zhao, C., ... & Xu, Z. (2024). Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. arXiv preprint arXiv:2405.04434.
- [4] Yeo, E., Tong, Y., Niu, M., Neubig, G., & Yue, X. (2025). Demystifying Long Chain-of-Thought Reasoning in LLMs. arXiv preprint arXiv:2502.03373.
- [5] Liu, R., Gao, J., Zhao, J., Zhang, K., Li, X., Qi, B., ... & Zhou, B. (2025). Can 1B LLM Surpass 405B LLM? Rethinking Compute-Optimal Test-Time Scaling. arXiv preprint arXiv:2502.06703.
- [6] Qin, Y., Li, X., Zou, H., Liu, Y., Xia, S., Huang, Z., ... & Liu, P. (2024). O1 Replication Journey: A Strategic Progress Report--Part 1. arXiv preprint arXiv:2410.18982.
- [7] Chen, Y., Wang, W., Lobry, S., & Kurtz, C. (2024). An llm agent for automatic geospatial data analysis. arXiv preprint arXiv:2410.18792.
- [8] Snell, C., Lee, J., Xu, K., & Kumar, A. (2024). Scaling llm test-time compute optimally can be more effective than scaling model parameters. arXiv preprint arXiv:2408.03314.
- [9] Muennighoff, N., Yang, Z., Shi, W., Li, X. L., Fei-Fei, L., Hajishirzi, H., ... & Hashimoto, T. (2025). s1: Simple test-time scaling. arXiv preprint arXiv:2501.19393.
- [10] Geiping, J., McLeish, S., Jain, N., Kirchenbauer, J., Singh, S., Bartoldson, B. R., ... & Goldstein, T. (2025). Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach. arXiv preprint arXiv:2502.05171.
- [11] Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., ... & Cobbe, K. (2023). Let's verify step by step. arXiv preprint arXiv:2305.20050.
- [12] Jingyang, Y., Huazuo, G., Damai,, Junyu, L., ... & Wangding, Z. (2025). Native Sparse Attention: Hardware-Aligned and Natively Trainable Sparse Attention. arXiv preprint arXiv:2502.11089.
- [13] Zhang, Y., Wu, S., Yang, Y., Shu, J., Xiao, J., Kong, C., & Sang, J. (2024). o1-coder: an o1 replication for coding. arXiv preprint arXiv:2412.00154.