

# **A Personal Review of the Incredible Journey towards Deepseek-R1**

**Reporter: Bin Hong**

# Outline

1

概览

2

技术详解

3

分析

4

总结

# 概览

## ➤ 背景

- Deepseek-R1 爆火出圈
- 以小得多的成本训出了比肩 OpenAI o1 的模型

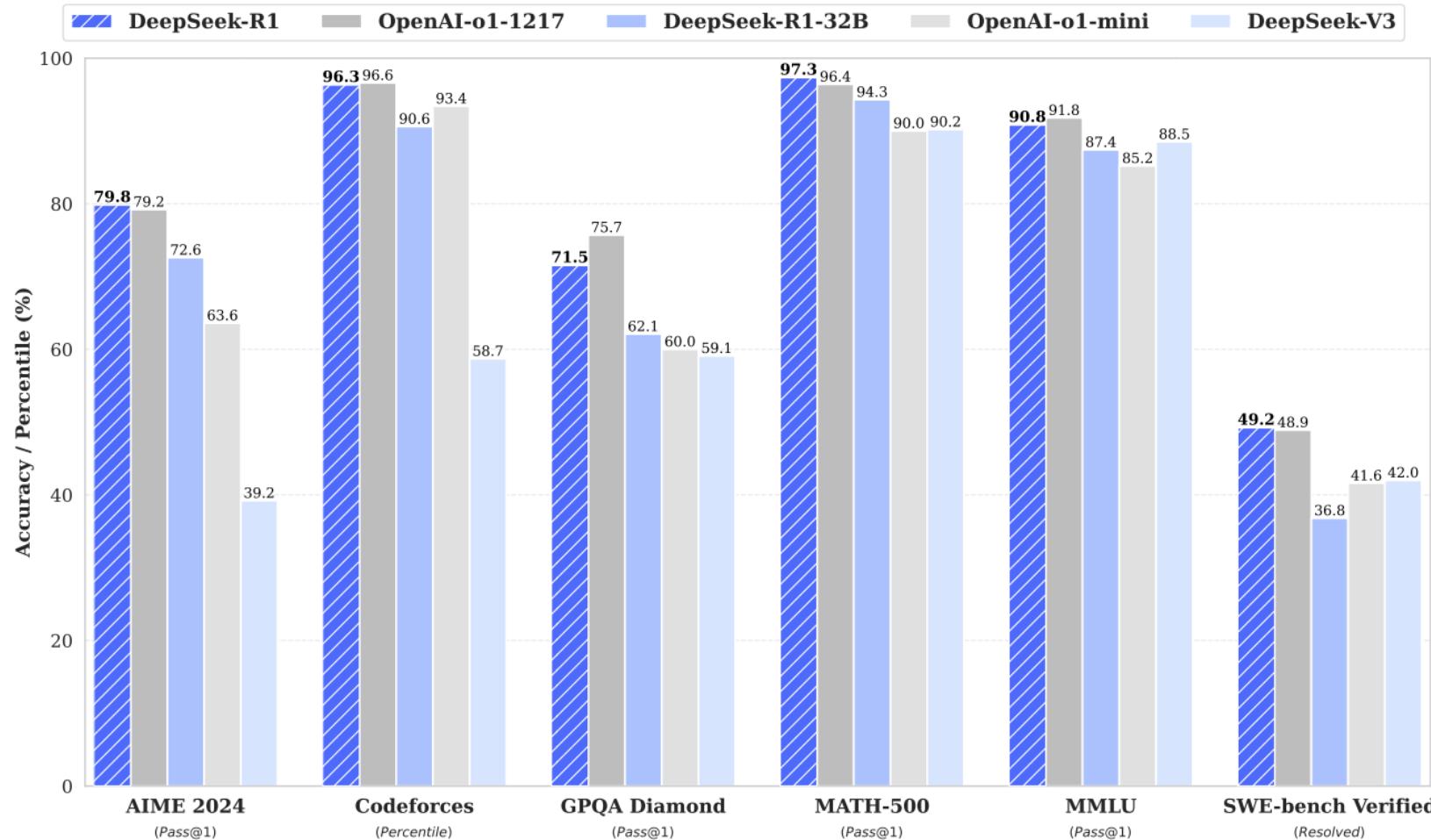


Figure 1 | Benchmark performance of DeepSeek-R1.

# 概览

## ➤ Deepseek 系列模型发展史 (R1 相关)

模型	时间	总参数量	激活参数量	上下文长度	训练token量	架构	关键技术	与 R1 相关的关键结论
llm	2024-01-05	67B	67B	4K	2T+1.5M+?	dense	Follow llama, 数据获取、清洗、合成、配比。Pre-training+二阶段SFT+DPO, 架构上主要是RoPE和GQA, 以及一个修正的scaling law	初步实验发现强化学习可以大幅度提升模型推理能力 确立长期和开源两个目标
moe	2024-01-11	142B (145B ongoing)	71B	-	245B	dpsk_moe	dpsk_moe: 细粒度专家+共享专家+负载均衡, 成功scaling到142B	提出了 R1 使用的基本架构
				-	245B	dpsk_moe	dpsk_moe: 细粒度专家+共享专家+负载均衡, 成功scaling到145B	提出了 R1 使用的基本架构
coder	2024-01-26	33B	33B	16K	2T	dense	数据获取、清洗、合成、配比。 引入FIM (增强文本填充) RoPE, Pre-training + SFT	dpsk_math 使用的 base model 的初级版本
coder v1.5	2024-01-26	7B	7B	4K	2T+2T	dense	基于DeepSeek-LLM-7B使用额外2T数据重新配比 continue Pre-training	dpsk_math 使用的 base model
math	2024-04-27	7B	7B	4K	120B+77.6w+?	dense	基于DeepSeek-Coder-Base-v1.5 7B进行SFT: fastText数据选择, 数据中集成CoT, PoT, tool等 <b>GRPO</b> : 提升top_k答案一致性	提出并应用 GRPO
coder v2	2024-06-17	236B	21B (NIAH?)	128K	1.17T	dpsk_moe	基于DeepSeek-V2 自然语言数据从 deepseek v2 中采样。 GRPO, FIM, YaRN	-
v2	2024-06-19	236B	21B	128K	8.1T+1.5M	dpsk_moe	MLA, GRPO, YaRN	提出并应用 MLA, 成功应用 YaRN。强化学习中 code 和 数学数据有特殊作用: 二阶段RL, 并解耦不同任务的 reward
v3	2024-12-27	671B	37B	128K	14.8T+1.5M	dpsk_moe +	MLA, deepseek MTP DualPipe, FP8, FIM 数据合成: 基于 DeepSeek-V2.5-based R1补充长CoT数据 新的 Moe 大规模 GRPO	优化了dpsk_moe架构, 并调整了训练逻辑。R1训练的 base model, 目前deepseek的正式版本
r1	2025-01-22	671B	37B	128K	-	dpsk_moe +	冷启动 SFT 多领域GRPO RFT 蒸馏到小模型	-

# 概览

## ➤ Deepseek 系列模型发展史 (R1 相关)

模型	时间	总参数量	激活参数数量	上下文长度	训练token量	架构	关键技术	与 R1 相关的关键结论
llm	2024-01-05	67B	67B	4K	2T+1.5M+?	dense	Follow llama, 数据获取、清洗、合成、配比。Pre-training+二阶段SFT+DPO, 架构上主要是RoPE和GQA, 以及一个修正的scaling law	初步实验发现强化学习可以大幅度提升模型推理能力 确立长期和开源两个目标
moe	2024-01-11	142B (145B ongoing)	71B	-	240B	dpsk_moe	145B	提出 J_KI 使用的基本架构
coder	2024-01-26	33B	33B	16K	2T	dense	数据获取、清洗、合成、配比。 引入FIM (增强文本填充) RoPE, Pre-training + SFT	dpsk_math 使用的 base model 的初级版本
coder v1.5	2024-01-26	7B	7B	4K	2T+2T	dense	基于DeepSeek-LLM-7B使用额外2T数据重新配比 continue Pre-training	dpsk_math 使用的 base model
math	2024-04-27	7B	7B	4K	120B+77.6w+?	dense	基于DeepSeek-Coder-Base-v1.5 7B进行SFT: fastText数据选择, 数据中集成CoT, PoT, tool等 <b>GRPO</b> : 提升top_k答案一致性	提出并应用 GRPO
coder v2	2024-06-17	236B (NIAH?)	21B	128K	1.17T	dpsk_moe	基于DeepSeek-V2 自然语言数据从 deepseek v2 中采样。 GRPO, FIM, YaRN	-
v2	2024-06-19	236B	21B	128K	8.1T+1.5M	dpsk_moe	MLA, GRPO, YaRN	提出并应用 MLA, 成功应用 YaRN。强化学习中 code 和 数学数据有特殊作用: 二阶段RL, 并解耦不同任务的 reward
v3	2024-12-27	671B	37B	128K	14.8T+1.5M	dpsk_moe +	MLA, deepseek MTP DualPipe, FP8, FIM 数据合成: 基于 DeepSeek-V2.5-based R1 补充长CoT数据 新的 Moe 大规模 GRPO	优化了dpsk_moe架构, 并调整了训练逻辑。R1训练的 base model, 目前deepseek的正式版本
r1	2025-01-22	671B	37B	128K	-	dpsk_moe +	冷启动 SFT 多领域GRPO RFT 蒸馏到小模型	-

没有盲目堆叠网络一口气提高参数量，而是逐步提升

# 概览

## ➤ Deepseek 系列模型发展史 (R1 相关)

模型	时间	总参数量	激活参数量	上下文长度	训练token量	架构	关键技术	与 R1 相关的关键结论
llm	2024-01-05	67 B	67 B	4K	2T+1.5M+?	dense	Follow llama, 数据获取、清洗、合成、配比。Pre-training+二阶段SFT+DPO, 架构上主要是RoPE和GQA, 以及一个修正的scaling law	初步实验发现强化学习可以大幅度提升模型推理能力 确立长期和开源两个目标
moe	2024-01-11	142 B (145 B ongoing)	71 B	-	245 B	dpsk_moe	很早就开始尝试稀疏激活 145B	提出了 R1 使用的基本架构
coder	2024-01-26	33B	33B	16K	2T	dense	数据获取、清洗、合成、配比。 引入FIM (增强文本填充) RoPE, Pre-training + SFT	dpsk_math 使用的 base model 的初级版本
coder v1.5	2024-01-26	7B	7B	4K	2T+2T	dense	基于DeepSeek-LLM-7B使用额外2T数据重新配比 continue Pre-training	dpsk_math 使用的 base model
math	2024-04-27	7B	7B	4K	120B+77.6w+?	dense	基于DeepSeek-Coder-Base-v1.5 7B进行SFT: fastText数据选择, 数据中集成CoT, PoT, tool等 <b>GRPO</b> : 提升top_k答案一致性	提出并应用 GRPO
coder v2	2024-06-17	236 B	21 B (NIAH?)	128K	1.17T	dpsk_moe	基于DeepSeek-V2 自然语言数据从 deepseek v2 中采样。 GRPO, FIM, YaRN	-
v2	2024-06-19	236 B	21 B	128K	8.1T+1.5M	dpsk_moe	MLA, GRPO, YaRN	提出并应用 MLA, 成功应用 YaRN。强化学习中 code 和 数学数据有特殊作用: 二阶段RL, 并解耦不同任务的 reward
v3	2024-12-27	671B	37 B	128 K	14.8 T+1.5M	dpsk_moe +	MLA, deepseek MTP DualPipe, FP8, FIM 数据合成: 基于 DeepSeek-V2.5-based R1补充长CoT数据 新的 Moe 大规模 GRPO 冷启动 SFT 多领域GRPO RFT 蒸馏到小模型	优化了dpsk_moe架构, 并调整了训练逻辑。R1训练的 base model, 目前deepseek的正式版本
r1	2025-01-22	671B	37 B	128 K	-	dpsk_moe +	-	-

# 概览

## ➤ Deepseek 系列模型发展史 (R1 相关)

模型	时间	总参数量	激活参数量	上下文长度	训练token量	架构	关键技术	与 R1 相关的关键结论
llm	2024-01-01			-			Follow llama, 数据获取、清洗、合成、配比。Pre-training+二阶段SFT+DPO, 架构上主要是 RoPE和GQA, 以及一个修正的 scaling law	初步实验发现强化学习可以大幅度提升模型推理能力 确立长期和开源两个目标
moe	2024-01-11		(145B ongoing)	-	245B	dpsk_moe	dpsk_moe: 细粒度专家+共享专家+负载均衡, 成功scaling到142B dpsk_moe: 细粒度专家+共享专家+负载均衡, 成功scaling到145B	提出了 R1 使用的基本架构 提出了 R1 使用的基本架构
coder	2024-01-26	33B	33B	16K	2T	dense	数据获取、清洗、合成、配比。 引入FIM (增强文本填充) RoPE, Pre-training + SFT	dpsk_math 使用的 base model 的初级版本
coder v1.5	2024-01-26	7B	7B	4K	2T+2T	dense	基于DeepSeek-LLM-7B使用额外2T数据重新配比 continue Pre-training	dpsk_math 使用的 base model
math	2024-04-27	7B	7B	4K	120B+77.6w+?	dense	基于DeepSeek-Coder-Base-v1.5 7B进行SFT: fastText数据选择, 数据中集成CoT, PoT, tool等 <b>GRPO</b> : 提升top_k答案一致性	提出并应用 GRPO
coder v2	2024-06-17	236B	21B	128K (NIAH?)	1.17T	dpsk_moe	基于DeepSeek-V2 自然语言数据从 deepseek v2 中采样。 GRPO, FIM, YaRN	-
v2	2024-06-19	236B	21B	128K	8.1T+1.5M	dpsk_moe	MLA, GRPO, YaRN	提出并应用 MLA, 成功应用 YaRN。强化学习中 code 和 数学数据有特殊作用: 二阶段RL, 并解耦不同任务的 reward
v3	2024-12-27	671B	37B	128K	14.8T+1.5M	dpsk_moe +	MLA, deepseek MTP DualPipe, FP8, FIM 数据合成: 基于 DeepSeek-V2.5-based R1补充长CoT数据 新的 Moe 大规模 GRPO 冷启动 SFT 多领域GRPO RFT 蒸馏到小模型	优化了dpsk_moe架构, 并调整了训练逻辑。R1训练的 base model, 目前deepseek的正式版本
r1	2025-01-22	671B	37B	128K	-	dpsk_moe +	-	-

# 概览

## ➤ Deepseek 系列模型发展史 (R1 相关)

模型	时间	总参数量	激活参数量	上下文长度	训练token量	架构	关键技术
llm	2024-01-05	67B	67B	4K	2T+1.5M+?	dense	Follow llama, 数据获取、清洗、合成、配比。Pre-training+二阶段RL, DPSK_MOE
moe	2024-01-11	142B (145B ongoing)	71B	-	245B	dpsk_moe	UPDATES, 增强文本填充, DPSK_MOE, 针对145B
coder	2024-01-26	33B	33B	16K	2T	dense	数据获取、清洗、合成、配比。 引入FIM (增强文本填充) RoPE, Pre-training + SFT
coder v1.5	2024-01-26	7B	7B	4K	2T+2T	dense	基于DeepSeek-LLM-7B使用额外2T数据重新配比 continue Pre-training
math	2024-04-27	7B	7B	4K	120B+77.6w+?	dense	基于DeepSeek-Coder-Base-v1.5 7B进行SFT: fastText数据选择, 数据中集成CoT, PoT, tool等 <b>GRPO</b> : 提升top_k答案一致性
coder v2	2024-06-17	236B (NIAH?)	21B	128K	1.17T	dpsk_moe	基于DeepSeek-V2 自然语言数据从 deepseek v2 中采样。 GRPO, FIM, YaRN
v2	2024-06-19	236B	21B	128K	8.1T+1.5M	dpsk_moe	MLA, GRPO, YaRN
v3	2024-12-27	671B	37B	128K	14.8T+1.5M	dpsk_moe +	MLA, deepseek MTP DualPipe, FP8, FIM 数据合成: 基于 DeepSeek-V2.5-based R1补充长CoT数据 新的Moe 大规模GRPO 冷启动SFT 多领域GRPO RFT 蒸馏到小模型
r1	2025-01-22	671B	37B	128K	-	dpsk_moe +	-

### 与 R1 相关的关键结论

初步实验发现强化学习可以大幅度提升模型推理能力  
确立长期和开源两个目标

提出了 R1 使用的基本架构

提出了 R1 使用的基本架构

dpsk\_math 使用的 base model 的初级版本

dpsk\_math 使用的 base model

提出并应用 GRPO

提出并应用 MLA, 成功应用 YaRN。强化学习中 code 和  
数学数据有特殊作用: 二阶段RL, 并解耦不同任务的  
reward

优化了dpsk\_moe架构, 并调整了训练逻辑。R1训练的  
base model, 目前deepseek的正式版本

# 概览

## ➤ Deepseek 系列模型发展史

- 两个主线：Scaling、降本增效
- 两个支线：数据质量、强化学习
- 一个风格：脚踏实地，稳扎稳打，实事求是，从巨量实验和尝试中总结经验
- 整体感：各个研究方向通力合作，成果汇聚成最终的 R1，浑然天成。



# Outline

1

概览

2

技术详解

3

分析

4

总结

# 技术详解

- Deepseek LLM：起点
  - 主要是 Follow 开源工作 LLaMA 2
  - 对 dense 架构做小调整
  - “小心翼翼” 地验证开源工作，尤其是 Scaling Law，并对其进行了修正
  - 经典开发流程：数据处理 - 预训练 - SFT – 强化学习

# 技术详解

## ➤ RMSNorm & SwiGLU

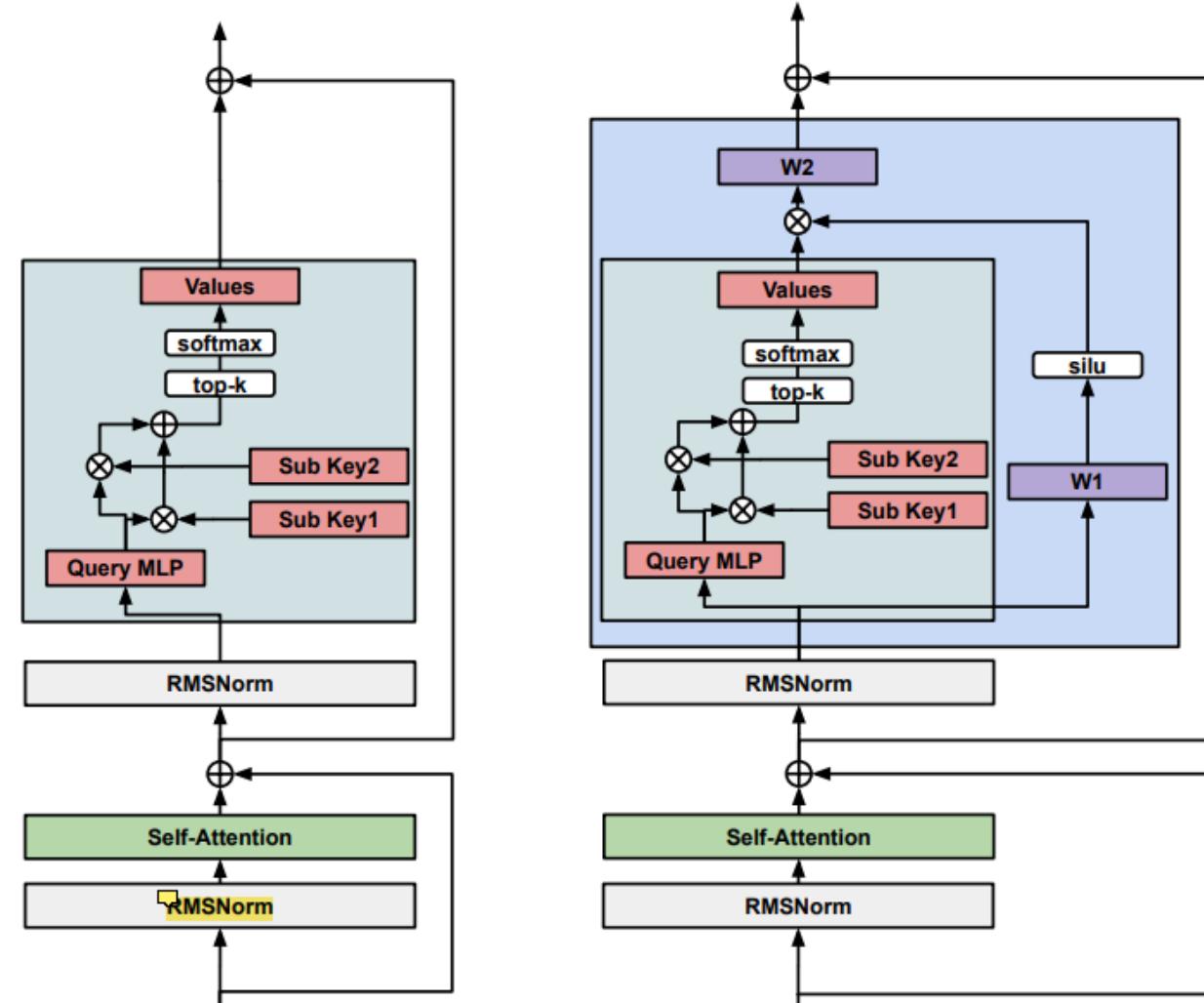
### ➤ RMSNorm:

$$\text{LayerNorm}(x) = \gamma \odot \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

$$\text{RMSNorm}(x) = \gamma \odot \frac{x}{\sqrt{\text{Mean}(x^2) + \epsilon}}$$

### ➤ SwiGLU:

$$\text{FFN}_{\text{SwiGLU}}(x, W, V, W_2) = (\text{SiLU}(xW) \otimes xV)W_2$$



# 技术详解

- RoPE (旋转位置编码)

- 绝对位置编码：

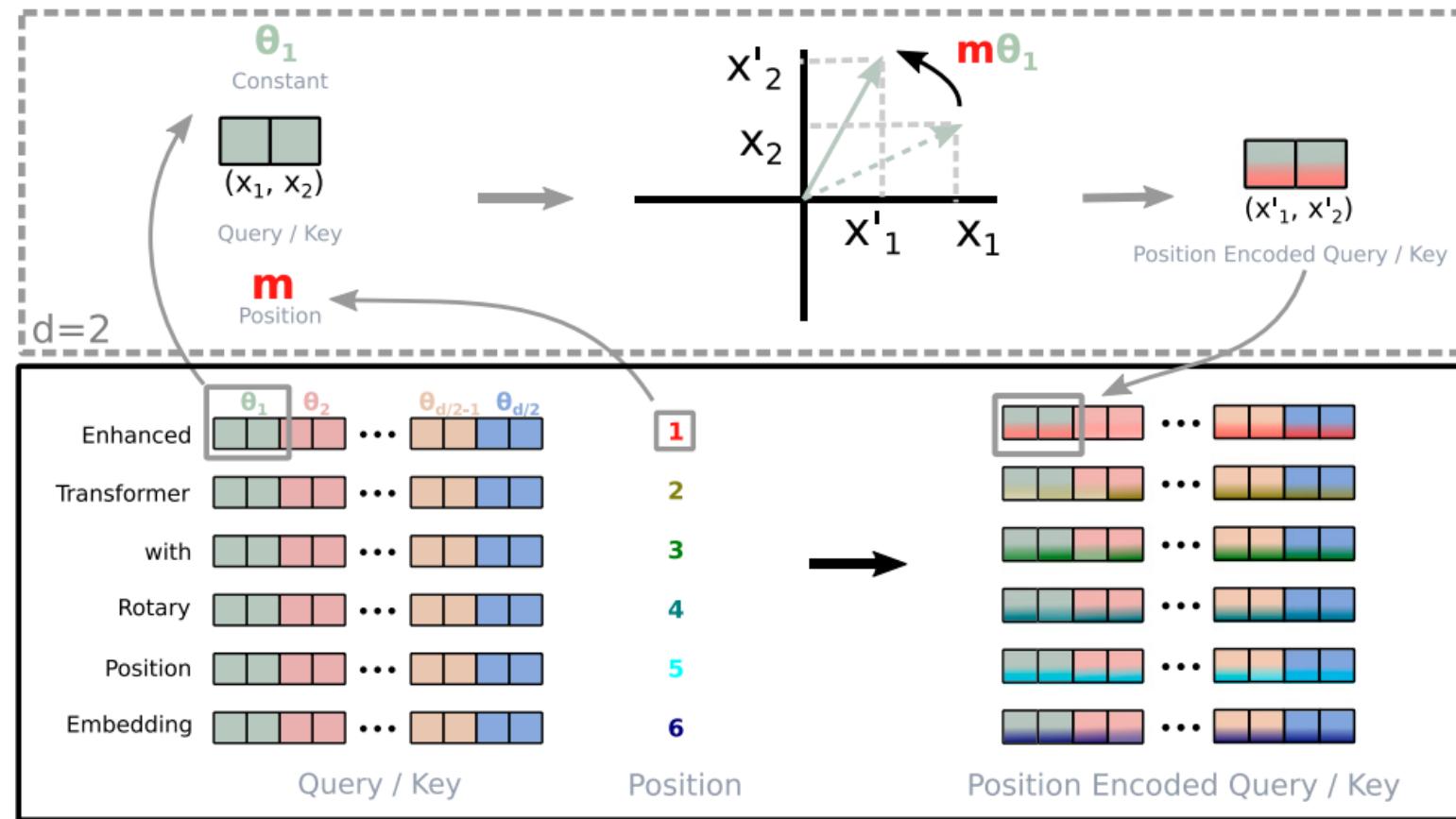
$$f_{\{q,k,v\}}(x_i, i) = W_{\{q,k,v\}}(x_i + p_i)$$

$$p_{i,2t} = \sin\left(\frac{i}{10000^{\frac{2t}{d}}}\right)$$

$$p_{i,2t+1} = \cos\left(\frac{i}{10000^{\frac{2t}{d}}}\right)$$

# 技术详解

## ➤ RoPE (旋转位置编码)



# 技术详解

- RoPE (旋转位置编码)

- 位置信息只需要注入注意力分数，因此只涉及 K 和 Q：

$$\langle f_q(x_m, m), f_k(x_n, n) \rangle = g(x_m, x_n, m - n)$$

- 一种二维的实现：

$$f_q(x_m, m) = (W_q x_m) e^{im\theta}$$

$$f_k(x_n, n) = (W_k x_n) e^{in\theta}$$

$$g(x_m, x_n, m - n) = \text{Re}[(W_q x_m)(W_k x_n)^* e^{i(m-n)\theta}]$$

# 技术详解

- RoPE (旋转位置编码) 二维实现方式的正确性：

$$f_q(x_m, m) = (W_q x_m) e^{im\theta}$$

$$f_k(x_n, n) = (W_k x_n) e^{in\theta}$$

$$g(x_m, x_n, m - n) = \text{Re}[(W_q x_m)(W_k x_n)^* e^{i(m-n)\theta}]$$

- 利用欧拉公式： $e^{ix} = \cos x + i \sin x$ ，得到：

$$e^{im\theta} = \cos(m\theta) + i \sin(m\theta)$$

$$e^{in\theta} = \cos(n\theta) + i \sin(n\theta)$$

$$e^{i(m-n)\theta} = \cos((m-n)\theta) + i \sin((m-n)\theta)$$

# 技术详解

- RoPE (旋转位置编码) 二维实现方式的正确性：
  - 回看公式： $f_q(x_m, m) = (W_q x_m) e^{im\theta}$ ，括号内两个维度分别视为实部和虚部：

$$f_q(x_m, m) = (W_q x_m) e^{im\theta} = q_m e^{im\theta} \quad q_m = \begin{pmatrix} q_m^{(1)} \\ q_m^{(2)} \end{pmatrix} = W_q x_m = \begin{pmatrix} W_q^{(11)} & W_q^{(12)} \\ W_q^{(21)} & W_q^{(22)} \end{pmatrix} \begin{pmatrix} x_m^{(1)} \\ x_m^{(2)} \end{pmatrix}$$

$$q_m e^{im\theta} = (q_m^{(1)} + i q_m^{(2)}) * (\cos(m\theta) + i \sin(m\theta)) \quad q_m = [q_m^{(1)}, q_m^{(2)}] = [q_m^{(1)} + i q_m^{(2)}]$$

- 于是可以拆分出一个旋转矩阵：

$$\begin{aligned} f_q(x_m, m) &= (W_q x_m) e^{im\theta} = q_m e^{im\theta} \\ &= [q_m^{(1)} \cos(m\theta) - q_m^{(2)} \sin(m\theta), q_m^{(2)} \cos(m\theta) + q_m^{(1)} \sin(m\theta)] \\ &= \begin{pmatrix} \cos(m\theta) & -\sin(m\theta) \\ \sin(m\theta) & \cos(m\theta) \end{pmatrix} \begin{pmatrix} q_m^{(1)} \\ q_m^{(2)} \end{pmatrix} \end{aligned}$$

# 技术详解

➤ RoPE (旋转位置编码) 二维实现方式的正确性：

➤ 同理：

$$\begin{aligned}f_k(x_n, n) &= (W_k x_n) e^{in\theta} = k_n e^{in\theta} \\&= [k_n^{(1)} \cos(n\theta) - k_n^{(2)} \sin(n\theta), k_n^{(2)} \cos(n\theta) + k_n^{(1)} \sin(n\theta)] \\&= \begin{pmatrix} \cos(n\theta) & -\sin(n\theta) \\ \sin(n\theta) & \cos(n\theta) \end{pmatrix} \begin{pmatrix} k_n^{(1)} \\ k_n^{(2)} \end{pmatrix}\end{aligned}$$

➤ 接下来是代入内积公式，利用三角函数和复数的性质进行一大串化简。  
(此处省略)

# 技术详解

- RoPE (旋转位置编码) 二维实现方式的正确性:

- 最后得到内积的形式:  $S = \tilde{Q}^T \tilde{K}$

$$= Q^T R_m^T R_n K$$

或者利用三角函数性质更具体地得到:

$$\begin{pmatrix} q_m^{(1)} & q_m^{(2)} \end{pmatrix} \begin{pmatrix} \cos((m-n)\theta) & -\sin((m-n)\theta) \\ \sin((m-n)\theta) & \cos((m-n)\theta) \end{pmatrix} \begin{pmatrix} k_n^{(1)} \\ k_n^{(2)} \end{pmatrix}$$

- 其中  $\theta_j = 10000^{-2(j-1)/d}, j \in [1, 2, \dots, d/2]$

# 技术详解

## ➤ RoPE (旋转位置编码)

➤ 隐层维度一般都是偶数。利用矩阵分块，对于大于 2 的维度两两一组处理：

$$f_{\{q,k\}}(\mathbf{x}_m, m) = \mathbf{R}_{\Theta,m}^d \mathbf{W}_{\{q,k\}} \mathbf{x}_m \quad \mathbf{q}_m^T \mathbf{k}_n = \left( \mathbf{R}_{\Theta,m}^d \mathbf{W}_q \mathbf{x}_m \right)^T \left( \mathbf{R}_{\Theta,n}^d \mathbf{W}_k \mathbf{x}_n \right) = \mathbf{x}_m^T \mathbf{W}_q \mathbf{R}_{\Theta,n-m}^d \mathbf{W}_k \mathbf{x}_n$$

$$\mathbf{R}_{\Theta,m}^d = \underbrace{\begin{pmatrix} \cos m\theta_0 & -\sin m\theta_0 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_0 & \cos m\theta_0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_1 & -\sin m\theta_1 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_1 & \cos m\theta_1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2-1} & -\sin m\theta_{d/2-1} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2-1} & \cos m\theta_{d/2-1} \end{pmatrix}}_{\mathbf{W}_m} \quad \mathbf{R}_{\Theta,n-m}^d = \left( \mathbf{R}_{\Theta,m}^d \right)^T \mathbf{R}_{\Theta,n}^d$$

$$\Theta = \left\{ \theta_i = 10000^{-2(i-1)/d}, i \in [1, 2, \dots, d/2] \right\}$$

# 技术详解

- RoPE (旋转位置编码) : 计算高效性
  - 显然旋转矩阵是个计算效率很差的稀疏矩阵, 于是可以:

$$\mathbf{R}_{\Theta,m}^d \mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{d-2} \\ x_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_0 \\ \cos m\theta_0 \\ \cos m\theta_1 \\ \cos m\theta_1 \\ \vdots \\ \cos m\theta_{d/2-1} \\ \cos m\theta_{d/2-1} \end{pmatrix} + \begin{pmatrix} -x_1 \\ x_0 \\ -x_3 \\ x_2 \\ \vdots \\ -x_{d-1} \\ x_{d-2} \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_0 \\ \sin m\theta_0 \\ \sin m\theta_1 \\ \sin m\theta_1 \\ \vdots \\ \sin m\theta_{d/2-1} \\ \sin m\theta_{d/2-1} \end{pmatrix}$$

# 技术详解

➤ RoPE (旋转位置编码) : 远程衰减 (注意力分配)

➤ 内积:

$$\left( \mathbf{R}_{\Theta,m}^d \mathbf{W}_q \mathbf{x}_m \right)^T \left( \mathbf{R}_{\Theta,n}^d \mathbf{W}_k \mathbf{x}_n \right) = \operatorname{Re} \left[ \sum_{i=0}^{d/2-1} \mathbf{q}_{[2i:2i+1]} \mathbf{k}_{[2i:2i+1]}^* e^{i(m-n)\theta_i} \right]$$

记  $h_i = \mathbf{q}_{[2i:2i+1]} \mathbf{k}_{[2i:2i+1]}^*$ ,  $S_j = \sum_{i=0}^{j-1} e^{i(m-n)\theta_i}$ , 并约定  $h_{d/2} = 0, S_0 = 0$

➤ 根据 Abel 变换 (分部求和法), 得到:

$$\sum_{i=0}^{d/2-1} \mathbf{q}_{[2i:2i+1]} \mathbf{k}_{[2i:2i+1]}^* e^{i(m-n)\theta_i} = \sum_{i=0}^{d/2-1} h_i (S_{i+1} - S_i) = \sum_{i=0}^{d/2-1} S_{i+1} (h_{i+1} - h_i)$$

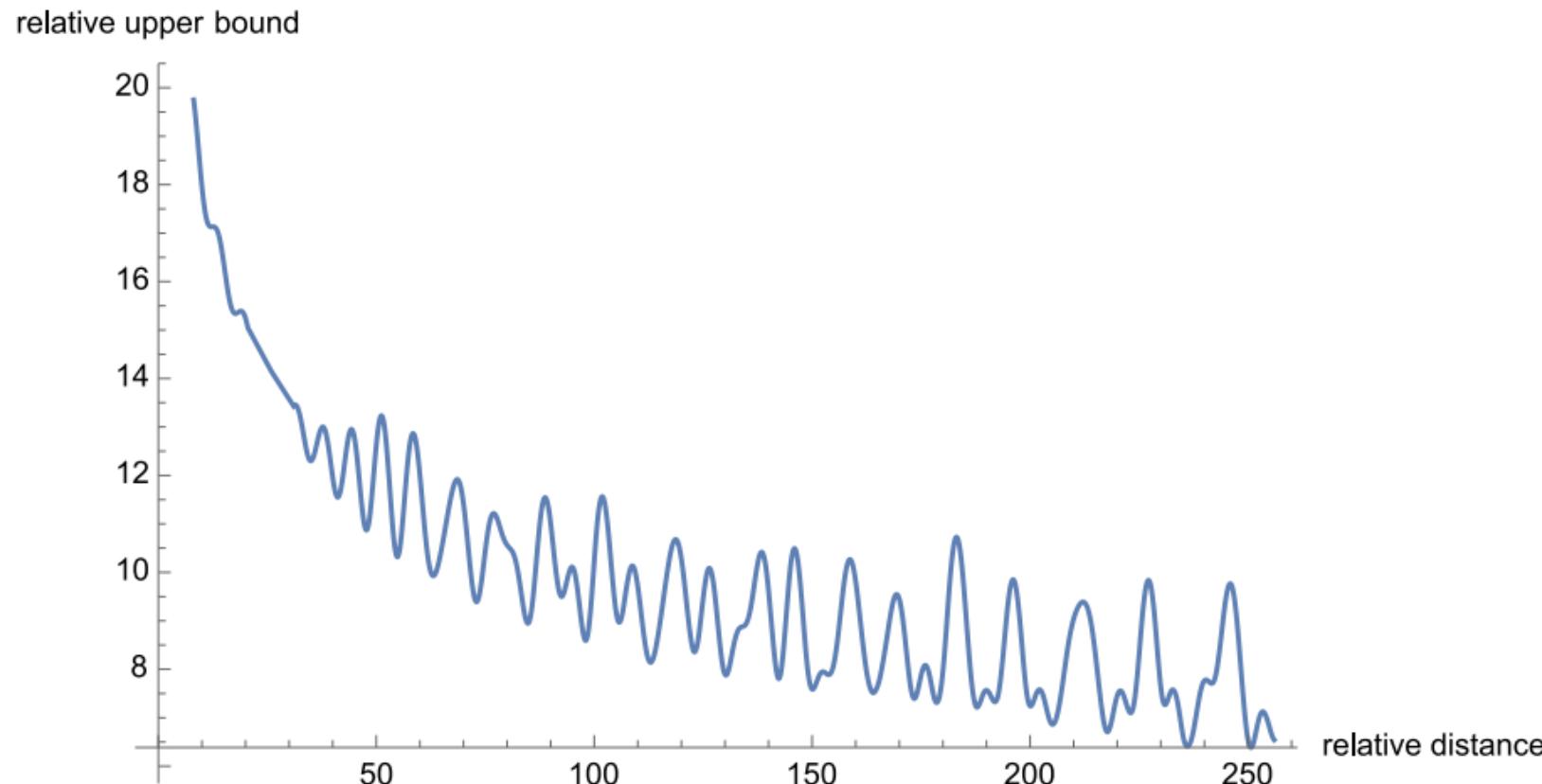
# 技术详解

- RoPE (旋转位置编码) : 远程衰减 (注意力分配)
  - 从而得到一个上界:

$$\begin{aligned} \left| \sum_{i=0}^{d/2-1} \mathbf{q}_{[2i:2i+1]} \mathbf{k}_{[2i:2i+1]}^* e^{i(m-n)\theta_i} \right| &= \left| \sum_{i=0}^{d/2-1} S_{i+1} (h_{i+1} - h_i) \right| \\ &\leq \sum_{i=0}^{d/2-1} |S_{i+1}| |h_{i+1} - h_i| \\ &\leq \left( \max_i |h_{i+1} - h_i| \right) \sum_{i=0}^{d/2-1} |S_{i+1}| \end{aligned}$$

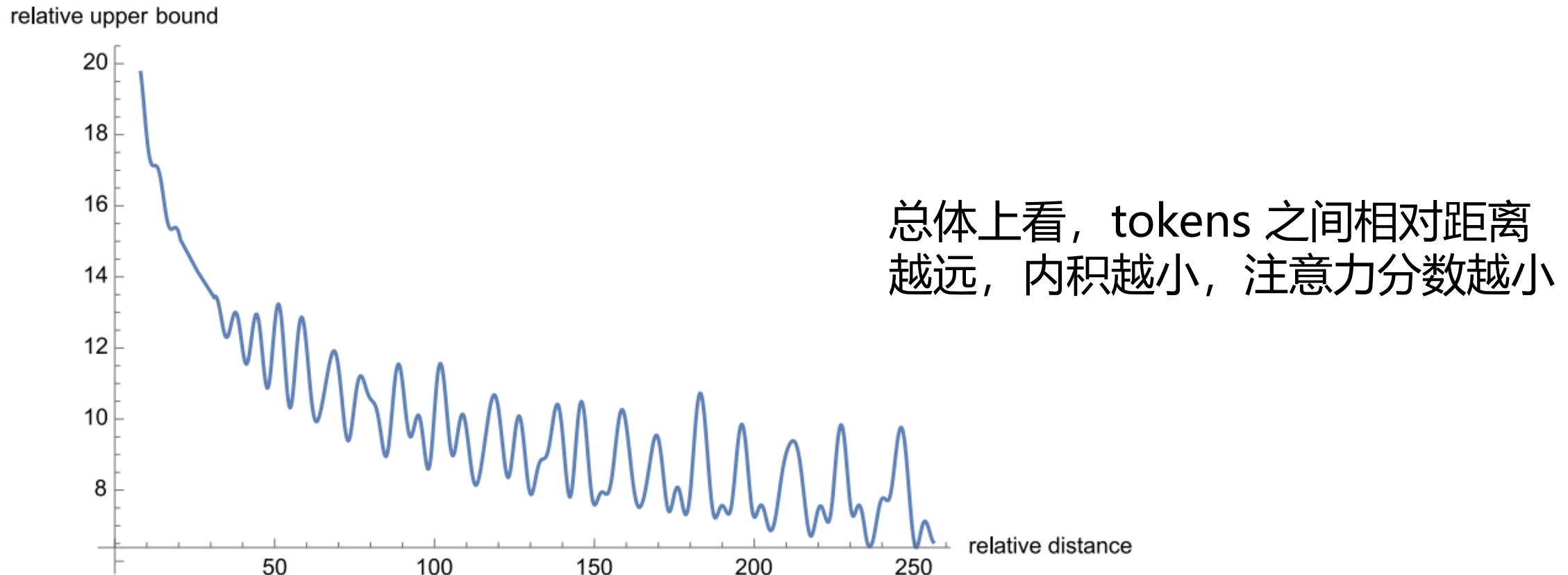
# 技术详解

- RoPE (旋转位置编码) : 远程衰减 (注意力分配)
  - 可以通过观察上界随 tokens 相对距离的变化来看内积变化的趋势:



# 技术详解

- RoPE (旋转位置编码) : 远程衰减 (注意力分配)
  - 可以通过观察上界随 tokens 相对距离的变化来看内积变化的趋势:



# 技术详解

- GQA (分组 Query 注意力)
  - 主要是为了推理加速

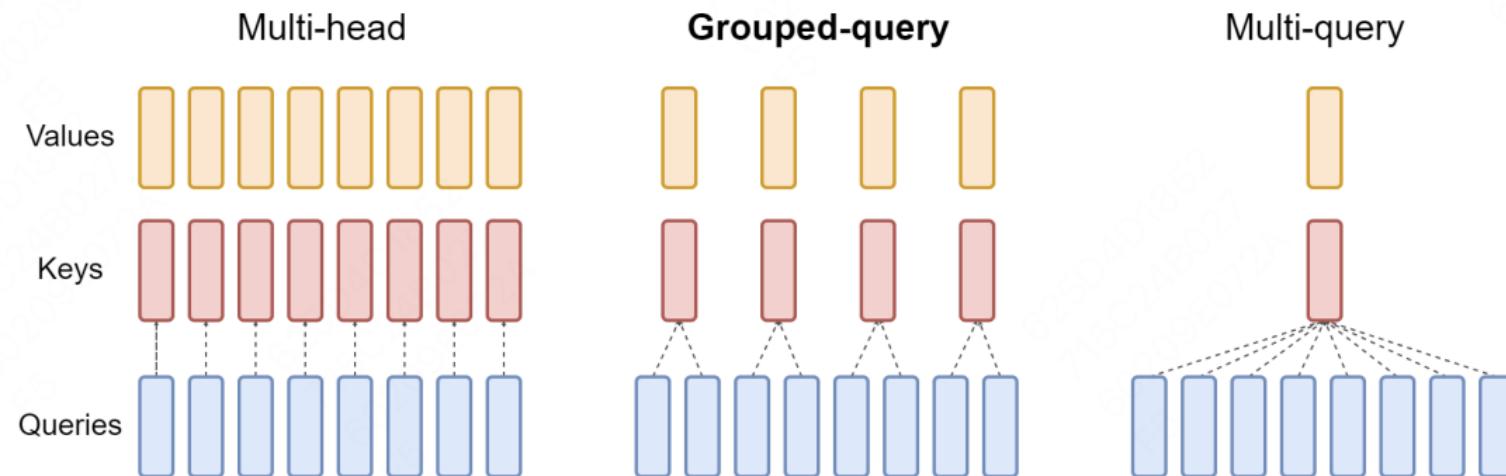


Figure 2: Overview of grouped-query method. Multi-head attention has  $H$  query, key, and value heads. Multi-query attention shares single key and value heads across all query heads. Grouped-query attention instead shares single key and value heads for each *group* of query heads, interpolating between multi-head and multi-query attention.

# 技术详解

- DPO (直接偏好优化) : 强化学习发展简史-概述
  - 强化学习的基本范式: 动作、状态、奖励
  - 目标: : 最大化累计奖励/最大化优势函数 (相对于平均情况的累计奖励)
  - 强化学习是与环境交互, 类似状态机 (但完全不同! )。执行某个动作, 使得 agent 发生状态变化, 并获得对应的奖励。
  - 策略: 状态和动作组成的序列/状态决定动作的条件分布
  - 早期为值学习, 直接建模奖励函数。比如与环境交互不断获得动作和状态对应的奖励存入表格, 或者用神经网络拟合。
  - 值学习灵活性差, 策略学习直接学习以状态为条件的动作的条件分布

# 技术详解

- DPO (直接偏好优化) : 强化学习发展简史-值学习
- 值学习的典型: Q-learning

---

## Algorithm 1 确定性环境的 Q-learning

---

算法参数 设定探索系数  $\epsilon$  和更新步长  $\alpha \in (0, 1]$

初始化 任意  $s \in S$  和  $a \in A$ , 任意初始化  $\hat{Q}(s, a)$ , 比如说初始化为 0

for 每一幕 do

    初始化  $s$

    repeat

        使用一种行为策略, 在  $s$  选择  $a$ , 比如基于当前 Q 表的  $\epsilon$  贪心策略

        执行动作  $a$ , 观察得到奖励  $r$  和下一个状态  $s'$

        更新  $\hat{Q}(s, a)$ :

$$\hat{Q}(s, a) \leftarrow \hat{Q}(s, a) + \alpha \left( r + \gamma \max_a \hat{Q}(s', a) - \hat{Q}(s, a) \right)$$

$s \leftarrow s'$

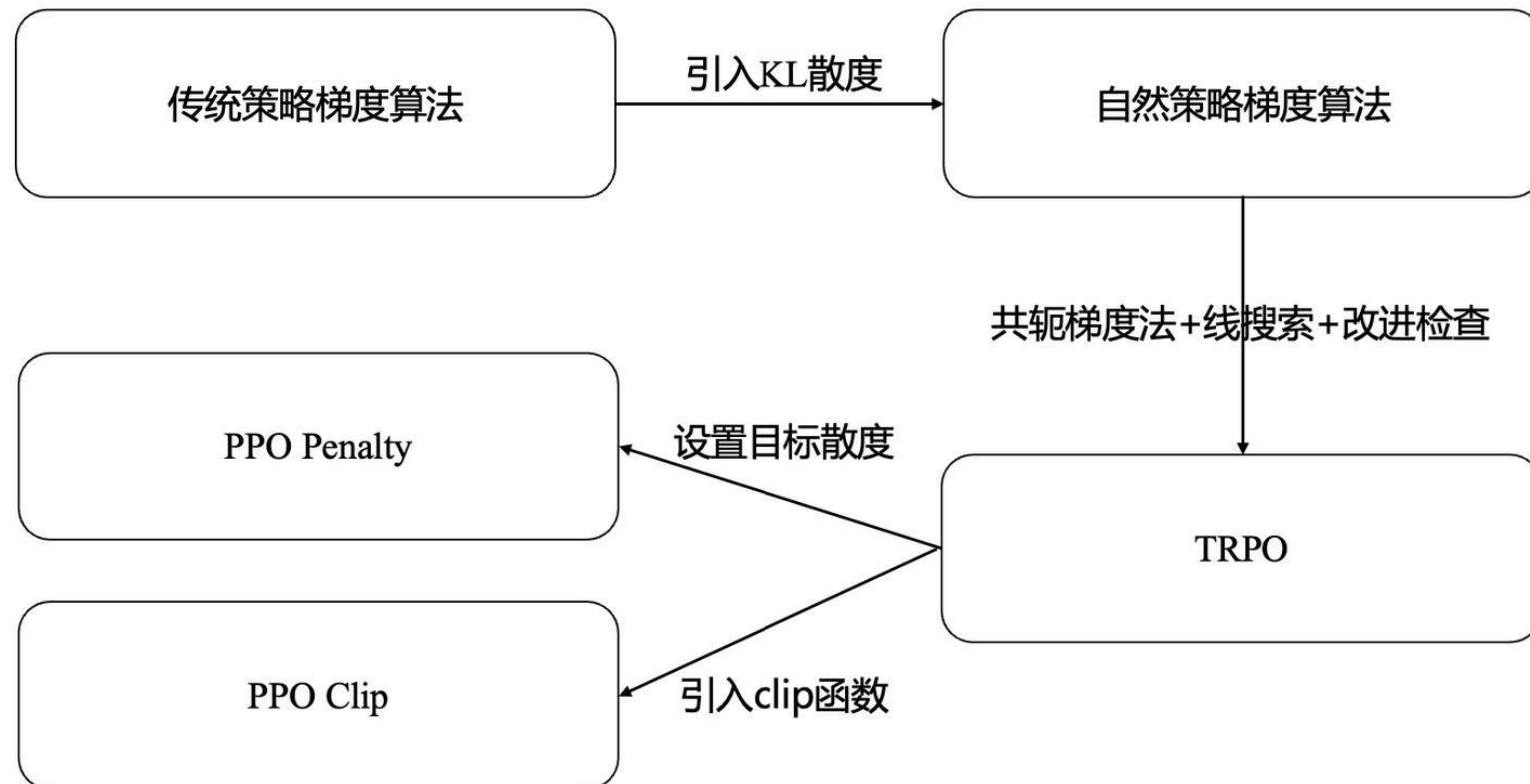
    until  $s$  是终止状态

end for

---

# 技术详解

- DPO (直接偏好优化) : 强化学习发展简史-策略学习
- 从简单梯度下降到 PPO (OpenAI 2017)



# 技术详解

➤ DPO (直接偏好优化) : 强化学习发展简史-传统策略梯度下降

➤ 目标函数:

$$\max_{\theta} J(\theta) = \max_{\theta} E_{\tau \sim \pi_{\theta}} R(\tau) = \max_{\theta} \sum_{\tau} P(\tau; \theta) R(\tau)$$

$$P(\tau; \theta) = \prod_{t=0}^T P(s_{t+1} | s_t, a_t) \cdot \pi_{\theta}(a_t | s_t)$$

➤ 梯度下降:

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \sum_{\tau} \nabla_{\theta} P(\tau; \theta) R(\tau) \\ &= \sum_{\tau} P(\tau; \theta) \frac{\nabla_{\theta} P(\tau; \theta)}{P(\tau; \theta)} R(\tau) \\ &= \sum_{\tau} P(\tau; \theta) \nabla_{\theta} \log P(\tau; \theta) R(\tau) \\ &= \mathbb{E}_{\tau \sim \pi_{\theta}} \nabla_{\theta} \log P(\tau; \theta) R(\tau)\end{aligned}$$

其中

$$\nabla_{\theta} \log P(\tau; \theta) = \nabla_{\theta} \left[ \sum_{t=0}^T \log P(s_{t+1} | s_t, a_t) + \sum_{t=0}^T \log \pi_{\theta}(a_t | s_t) \right]$$

# 技术详解

## ➤ DPO (直接偏好优化) : 强化学习发展简史-传统策略梯度下降

注意到  $\sum_{t=0}^T \log P(s_{t+1}|s_t, a_t)$  只和环境相关 (因为决定转移的要素都是给定的) , 于是上式可以进一步写成

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

## ➤ 应用时不可能真的遍历求解, 于是进行采样 (分布的形式可以预定义) :

$$P(\tau^{(i)}; \theta) = \frac{1}{m}, \forall i \in [1, m], i \in \mathbb{N}^+$$

$$\begin{aligned}\nabla_{\theta} J(\theta) &\approx \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} \log P(\tau^{(i)}; \theta) R(\tau^{(i)}) \\ &= \frac{1}{m} \sum_{i=1}^m \left( \sum_{t^{(i)}=0}^{T^{(i)}} \nabla_{\theta} \log \pi_{\theta}(a_{t^{(i)}}|s_{t^{(i)}}) \right) R(\tau^{(i)})\end{aligned}$$

$$\nabla_{\theta} J(\theta) \approx \frac{1}{n} \sum_{i=1}^n (\nabla_{\theta} \log \pi_{\theta}(a_{t^{(i)}}|s_{t^{(i)}})) R(t^{(i)})$$

# 技术详解

- DPO (直接偏好优化) : 强化学习发展简史-自然策略梯度下降
  - 防止策略更新幅度过大导致性能骤降
  - 用 KL 散度阈值限制更新前后的策略 (条件分布) 的差别
  - 实际上是一个带约束的优化问题:  $\Delta\theta^* = \underset{\mathcal{D}_{\text{KL}}(\pi_\theta \| \pi_{\theta+\Delta\theta}) \leq \epsilon}{\text{argmax}} J(\theta + \Delta\theta)$
  - KL 散度计算公式:  
$$\mathcal{D}_{\text{KL}}(\pi_\theta \| \pi_{\theta+\Delta\theta}) = \sum_{x \in \mathcal{X}} \pi_\theta(x) \log\left(\frac{\pi_\theta(x)}{\pi_{\theta+\Delta\theta}(x)}\right)$$
  - 求解该约束问题需要使用类似 SVM 的拉格朗日松弛方法以及非常复杂的数学推导, 这里省略并直接给出结论:

$$\Delta\theta = \sqrt{\frac{2\epsilon}{\nabla J(\theta)^\top F(\theta)^{-1} \nabla J(\theta)}} \tilde{\nabla} J(\theta) \quad \tilde{\nabla} J(\theta) = F(\theta)^{-1} \nabla J(\theta)$$

# 技术详解

➤ DPO (直接偏好优化) : 强化学习发展简史-信任域策略优化 (TRPO)

➤ 未来的 PPO 的优化对象

➤ 引入优势函数重定义优化目标:  $\hat{g} = \hat{\mathbb{E}}_t \left[ \nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \hat{A}_t \right]$

$$J(\pi_{\theta+\Delta\theta}) = J(\pi_{\theta}) + \mathbb{E}_{\tau \sim \pi_{\theta+\Delta\theta}} \sum_{t=0}^{\infty} \gamma^t A^{\pi_{\theta}}(s_t, a_t) \quad A^{\pi_{\theta}}(s, a) = \mathbb{E}(Q^{\pi_{\theta}}(s, a) - V^{\pi_{\theta}}(s))$$

➤ 用一个折扣分布模拟状态的分布 (广义优势估计, 可以视为一种采样方法) :

$$\rho_{\pi}(s) = (P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \dots)$$

$$J(\pi_{\theta+\Delta\theta}) = J(\pi_{\theta}) + \sum_{s \in \mathcal{S}} \rho_{\pi_{\theta+\Delta\theta}}(s) \sum_{a \in \mathcal{A}} \pi_{\theta+\Delta\theta}(a | s) A^{\pi_{\theta}}(s, a)$$

➤ 用当前策略近似更新后的策略

$$J(\pi_{\theta+\Delta\theta}) \approx J(\pi_{\theta}) + \sum_{s \in \mathcal{S}} \rho_{\pi_{\theta}}(s) \sum_{a \in \mathcal{A}} \pi_{\theta+\Delta\theta}(a | s) A^{\pi_{\theta}}(s, a)$$

# 技术详解

➤ DPO (直接偏好优化) : 强化学习发展简史-信任域策略优化 (TRPO)

➤ 引入重要性采样 (用旧策略作为替代分布) :  $J(\pi_{\theta+\Delta\theta}) \approx J(\pi_\theta) + \mathbb{E}_{s \sim \rho_{\pi_\theta}} \frac{\pi_{\theta+\Delta\theta}(a | s)}{\pi_\theta(a | s)} A^{\pi_\theta}(s, a)$

➤ 最后得到优化目标 ( “替代优势” ) :  $J(\pi_{\theta+\Delta\theta}) - J(\pi_\theta) \approx \mathbb{E}_{s \sim \rho_{\pi_\theta}} \frac{\pi_{\theta+\Delta\theta}(a | s)}{\pi_\theta(a | s)} A^{\pi_\theta}(s, a) = \mathcal{L}_{\pi_\theta}(\pi_{\theta+\Delta\theta})$

➤ 优势函数用广义优势估计计算:

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1}$$

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

# 技术详解

- DPO (直接偏好优化) : 强化学习发展简史-近端策略优化 (PPO)
  - TRPO 的缺陷: 过于复杂, 与包含噪声 (如深度学习中常见的 Dropout) 或参数共享的架构不兼容。另外, 在 Scalability、数据利用效率上, 以及鲁棒性 (主要是指性能不依赖调参) 上有缺陷
  - PPO 是 OpenAI 为了 LLM 技术的一个铺垫: 大模型上的强化学习
  - 其目标函数为:

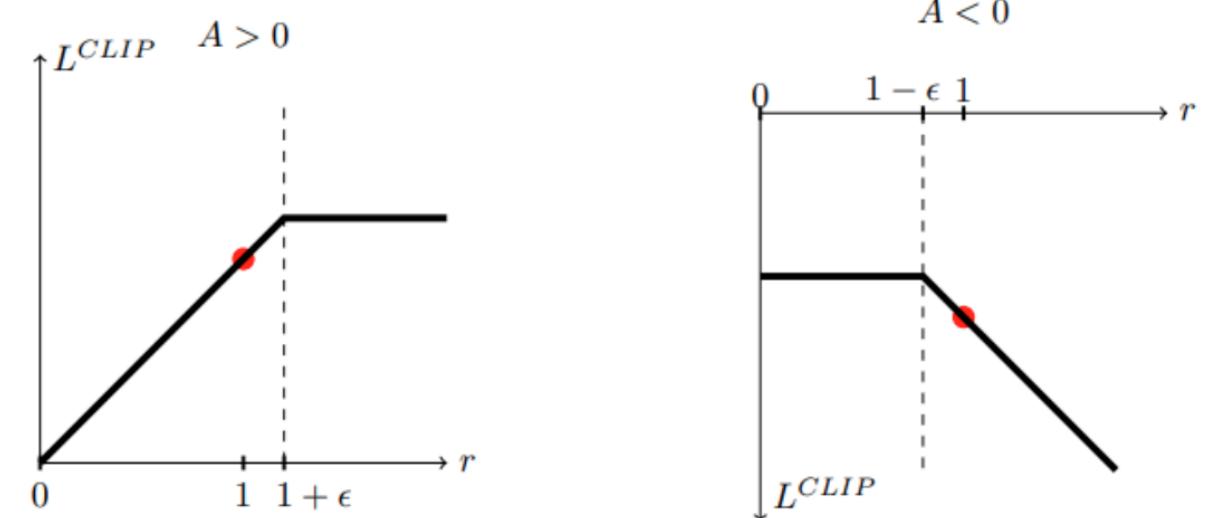
$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[ \min(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t) \right]$$

- 其中  $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ , 这个比值从 TRPO 开始反复出现。实际上是重要性采样的修正权重。

# 技术详解

- DPO (直接偏好优化) : 强化学习发展简史-近端策略优化 (PPO)
- Clip 操作将  $r_t(\theta)$  限制在区间  $[1 - \epsilon, 1 + \epsilon]$  内:

- 如果  $r_t(\theta) > 1 + \epsilon$ , 则将其裁剪为  $1 + \epsilon$ 。
- 如果  $r_t(\theta) < 1 - \epsilon$ , 则将其裁剪为  $1 - \epsilon$ 。
- 如果  $r_t(\theta)$  在  $[1 - \epsilon, 1 + \epsilon]$  内, 则保持不变。



# 技术详解

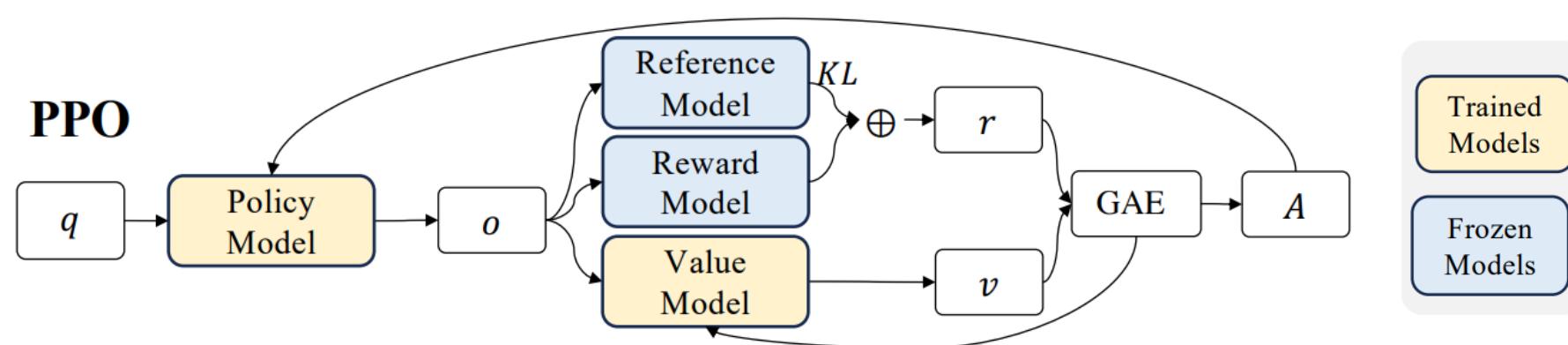
- DPO (直接偏好优化) : 强化学习发展简史-近端策略优化 (PPO)
  - 将对 KL 散度的硬约束 (强制要求小于阈值) , 更改为自适应的惩罚项:

$$L^{KL PEN}(\theta) = \hat{\mathbb{E}}_t \left[ \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} \hat{A}_t - \beta \cdot \text{KL}[\pi_{\theta_{\text{old}}}(\cdot | s_t), \pi_\theta(\cdot | s_t)] \right]$$

- 迭代时, 某次更新若更新幅度过大, 则增大  $\beta$ , 否则减小
- PPO 的两项优化总体来看都是为了 Scale: 提升训练效率和稳定性

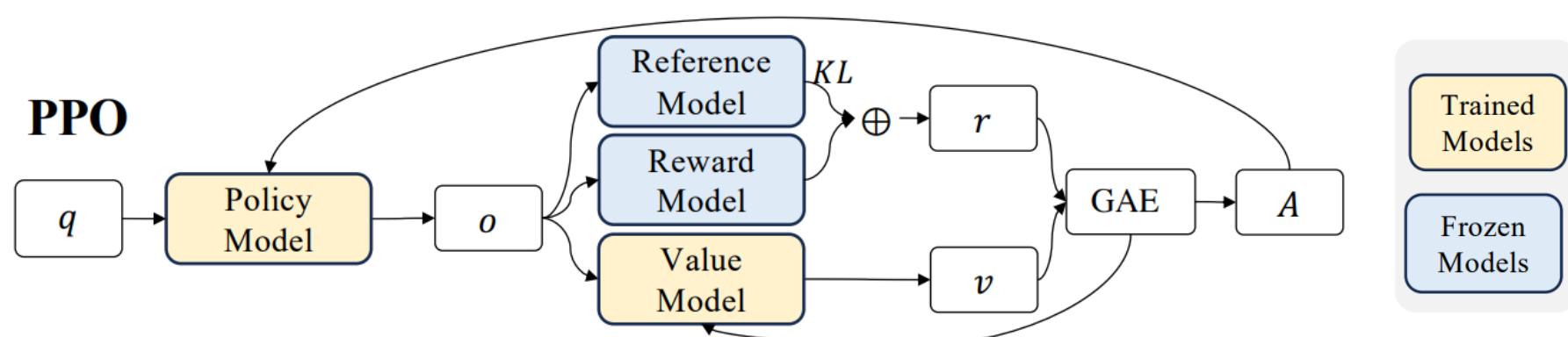
# 技术详解

- DPO (直接偏好优化) : 强化学习发展简史-PPO 如何用于大模型?
  - Reference Model: 经过 SFT 的 LLM
  - Reward Model: 打分模型, 可以通过 LLM + 线性层来得到。数据由 SFT 后的模型输出 (可以加上其他数据, 比如人类标注数据), 由人类打分得到。
  - Value Model: 价值估计模型
  - Policy Model: 正在训练的 LLM



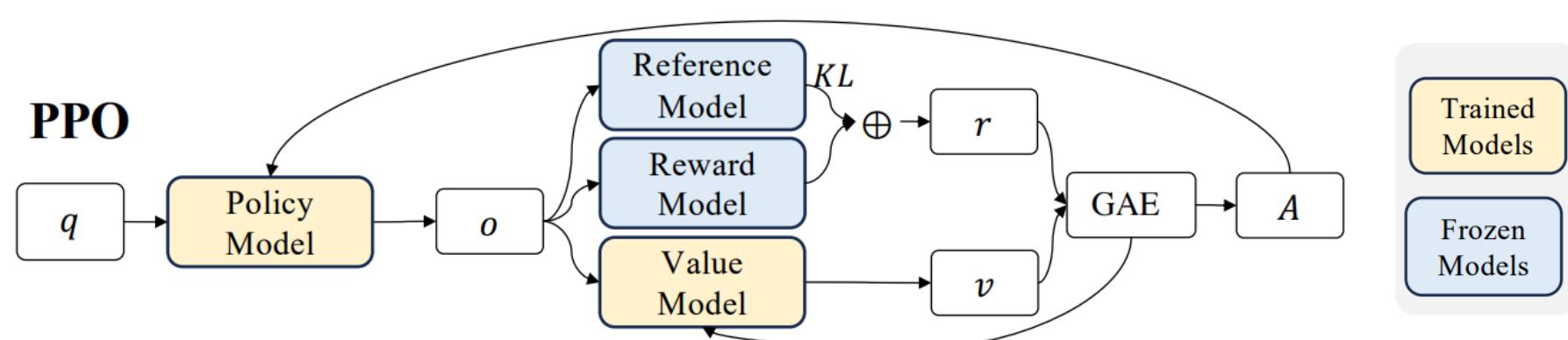
# 技术详解

- DPO (直接偏好优化) : 强化学习发展简史-PPO 如何用于大模型?
  - 回顾 PPO (忽略 clip) :  $L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t \left[ \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)} \hat{A}_t - \beta \cdot KL[\pi_{\theta_{old}}(\cdot | s_t), \pi_\theta(\cdot | s_t)] \right]$
  - 旧策略  $\pi_{\theta_{old}}(a_t | s_t)$  对应 Reference Model
  - 新策略  $\pi_\theta(a_t | s_t)$  对应 Policy Model
  - Reward Model 是提前训练好的比如  $loss(\theta) = -\frac{1}{\binom{K}{2}} E_{(x, y_w, y_l) \sim D} [\log (\sigma(r_\theta(x, y_w) - r_\theta(x, y_l)))]$
  - Value Model 则随着 LLM 训练更新



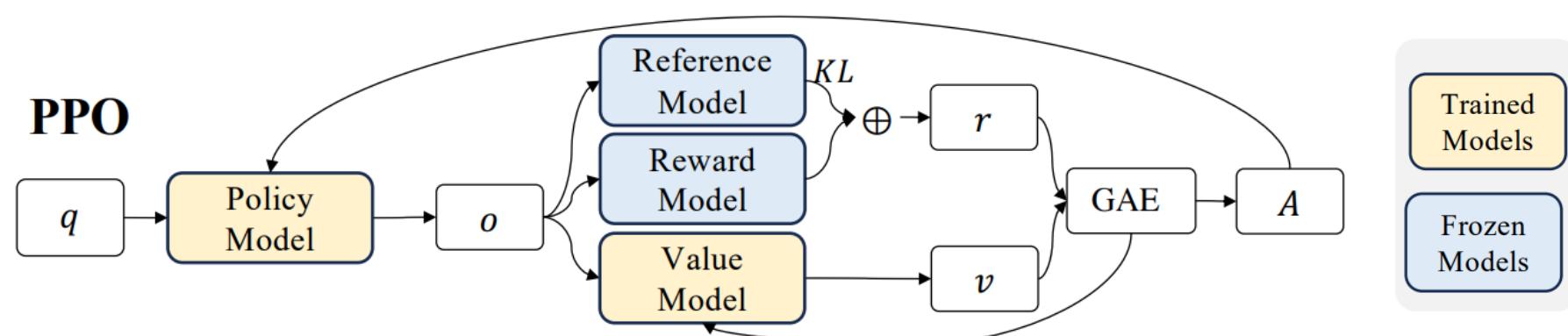
# 技术详解

- DPO (直接偏好优化) : 强化学习发展简史-PPO 如何用于大模型?
  - 回顾 PPO (忽略 clip) :  $L^{KL PEN}(\theta) = \hat{\mathbb{E}}_t \left[ \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)} \hat{A}_t - \beta \cdot KL[\pi_{\theta_{old}}(\cdot | s_t), \pi_\theta(\cdot | s_t)] \right]$
  - 在 Next Tokens Prediction 的任务中, PPO 公式中的 t 代表了时间步
  - 于是对每个 token 来计算每一步的 Loss
  - Reward Model 用于计算已经得到的奖励 (生成完成的部分)
  - Value Model 则是估算未来预期的总奖励 (未生成完成的部分), 在训练时使用已生成的 Reward 来进行更新



# 技术详解

- DPO (直接偏好优化) : 强化学习发展简史-PPO 如何用于大模型?
  - 回顾 GAE:  $\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \dots + (\gamma\lambda)^{T-t+1}\delta_{T-1}$        $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$
  - Reward Model 用于计算  $r_t$
  - Value Model 用于动态地拟合  $V$  函数
  - 在 2022 年的一篇中, KL penalty 变为对 reward 来应用      
$$r_t = r_\phi(q, o_{\leq t}) - \beta \log \frac{\pi_\theta(o_t|q, o_{<t})}{\pi_{ref}(o_t|q, o_{<t})}$$



# 技术详解

- DPO (直接偏好优化) : 强化学习发展简史-DPO (直接偏好优化)
  - 回顾传统策略梯度方法 (这里引入 KL penalty) :

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

- 经过一系列变形 (这里省略) , 得到:

$$\min_{\pi} \mathbb{E}_{x \sim D} \mathbb{E}_{y \sim \pi(y|x)} [\log \frac{\pi(y|x)}{\frac{1}{Z(x)} \pi_{\text{ref}}(y|x) e^{\frac{1}{\beta} r(x,y)}} - \log Z(x)]$$

- 其中  $Z(x) = \sum_y \pi_{\text{ref}}(y|x) e^{\frac{1}{\beta} r(x,y)}$
- 假设:

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

- 再经过一系列变形得到:  $\min_{\pi} \mathbb{E}_{x \sim D} [\mathbb{D}_{\text{KL}}(\pi(y|x) || \pi^*(y|x)) - \log Z(x)]$
- KL 散度的最小值条件:  $\pi(y|x) = \pi^*(y|x) = \frac{1}{Z(x)} \pi_{\text{ref}}(y|x) e^{\frac{1}{\beta} r(x,y)}$

# 技术详解

- DPO (直接偏好优化) : 强化学习发展简史-DPO (直接偏好优化)
- 回顾传统策略梯度方法 (这里引入 KL penalty) :

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

➤ 令  $\pi_r(y|x) = \frac{1}{Z(x)} \pi_{\text{ref}}(y|x) e^{\frac{1}{\beta} r(x, y)}$ , 则  $r(x, y) = \beta \log \frac{\pi_r(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z(x)$

➤ 利用 Bradley-Terry 模型:

$$P(i > j) = \frac{\alpha_i}{\alpha_i + \alpha_j}$$

$\alpha_i$  表示第*i*个元素的实力。

$P(i > j)$  表示第*i*个元素战胜第*j*个元素的概率。

一般的Loss函数:

$$\text{Loss} = -\mathbb{E}_{(\alpha_x, \alpha_y) \sim D} [\ln \frac{\alpha_x}{\alpha_x + \alpha_y}]$$

强化学习里:

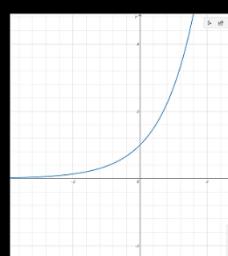
大模型输入的prompt是x, 回答是y。

回答y的好坏 (实力得分) 是靠Reward模型来评估。

$$P(y_1 > y_2) = \frac{r(x, y_1)}{r(x, y_1) + r(x, y_2)}$$

$r(x, y)$  可能返回负数, 所以加上指数函数

$$P(y_1 > y_2) = \frac{\exp(r(x, y_1))}{\exp(r(x, y_1)) + \exp(r(x, y_2))}$$



$$P(y_w > y_l) = \frac{\exp(r(x, y_w))}{\exp(r(x, y_w)) + \exp(r(x, y_l))}$$

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$

$$\text{Loss} = -\mathbb{E}_{(x, y_w, y_l) \sim D} [\ln \frac{\exp(r(x, y_w))}{\exp(r(x, y_w)) + \exp(r(x, y_l))}]$$

$$= -\mathbb{E}_{(x, y_w, y_l) \sim D} [\ln \frac{1}{1 + \exp(r(x, y_l) - r(x, y_w))}]$$

$$= -\mathbb{E}_{(x, y_w, y_l) \sim D} [\ln \sigma(r(x, y_w) - r(x, y_l))] \quad -\ln \sigma(r(x, y_w) - r(x, y_l))$$

# 技术详解

- DPO (直接偏好优化) : 强化学习发展简史-DPO (直接偏好优化)
  - 回顾传统策略梯度方法 (这里引入 KL penalty) :

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

➤ 令  $\pi_r(y|x) = \frac{1}{Z(x)} \pi_{\text{ref}}(y|x) e^{\frac{1}{\beta} r(x, y)}$ , 则  $r(x, y) = \beta \log \frac{\pi_r(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z(x)$

➤ BT 模型只涉及对两个回答的偏好选择, 恰好可以通过相减消去  $Z(x)$

$$P(y_w \succ y_l | x) = \frac{1}{1 + e^{r(x, y_l) - r(x, y_w)}} = \frac{1}{1 + \exp(\beta \log \frac{\pi_r(y_l|x)}{\pi_{\text{ref}}(y_l|x)} - \beta \log \frac{\pi_r(y_w|x)}{\pi_{\text{ref}}(y_w|x)})}$$

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

- DPO 通过技巧跳过了奖励模型, 本质上是一种离线强化学习, 需要提前构造数据集 (或者说, 没有跳过造偏好数据这一步)

# 技术详解

- Deepseek MoE：降低成本的核心研究
  - Follow MoE 经典工作 GShard，做了创新优化并成功 scale 到 142 B
  - 全新的稀疏激活架构：大量工程难题
    - 新的架构意味着需要自研训练方法、自研分布式策略等等
    - 新的架构如何 scale？
    - 舍弃开源社区的一大部分支持
  - 引入了辅助 Loss (V3 版本被优化)

# 技术详解

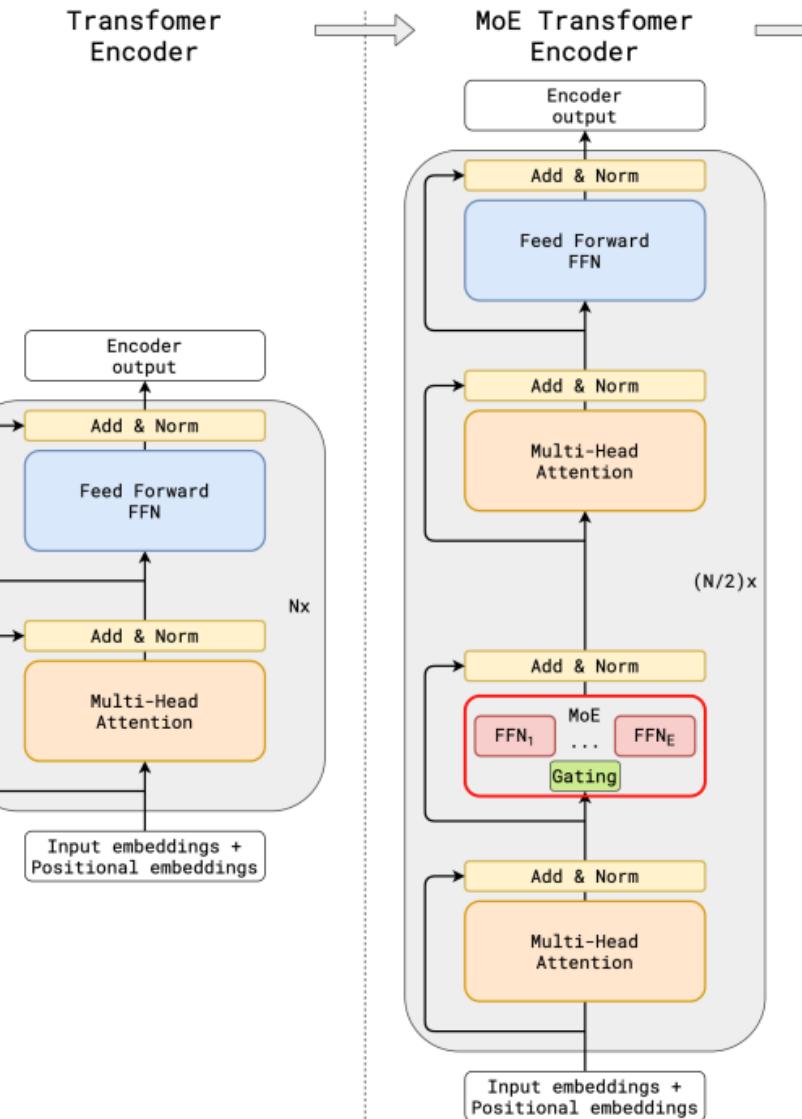
- Background: GShard
  - 在 MHA 后加入由多个 FFN 组成的 MoE layer
  - MoE layer:

$$\mathcal{G}_{s,E} = \text{GATE}(x_s)$$

$$\text{FFN}_e(x_s) = w_{oe} \cdot \text{ReLU}(w_{ie} \cdot x_s)$$

$$y_s = \sum_{e=1}^E \mathcal{G}_{s,e} \cdot \text{FFN}_e(x_s)$$

- 其中，路由单元  $\text{GATE}(\cdot)$  由一个线性层和 softmax 激活函数组成，用于计算路由权重
- 每个 token 会被不同的专家（们）处理

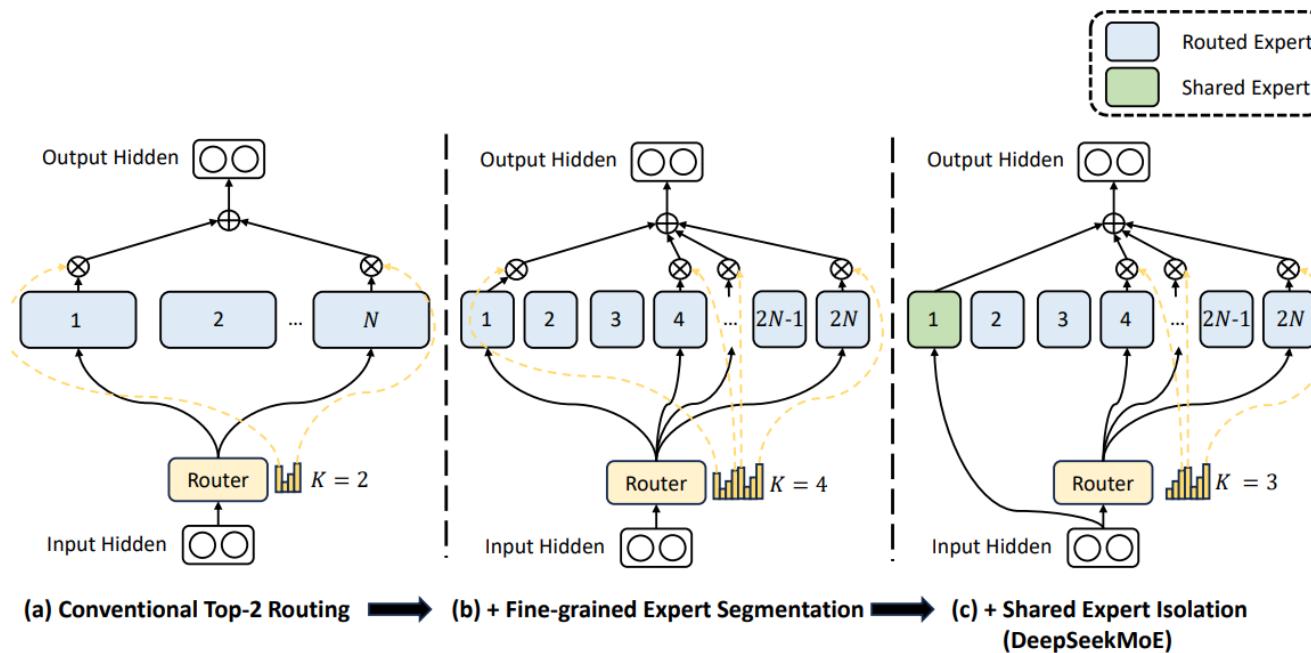


# 技术详解

## ➤ Deepseek MoE：四个公式

### ➤ 优化目标：

- 减少知识叠加：缓解一个专家要处理多种“类型”的 token 的情况——拆分专家的隐层维度
- 减少知识冗余：缓解不同专家“记忆”着相同“类型”的 token 的情况——设定始终激活的专家



# 技术详解

## ➤ Deepseek MoE：四个公式

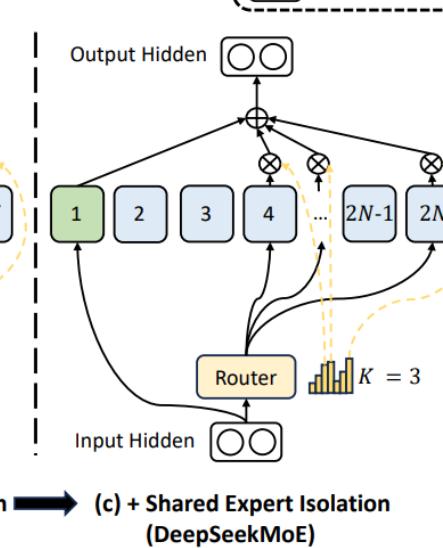
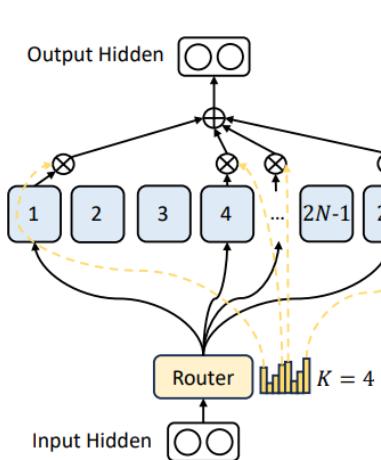
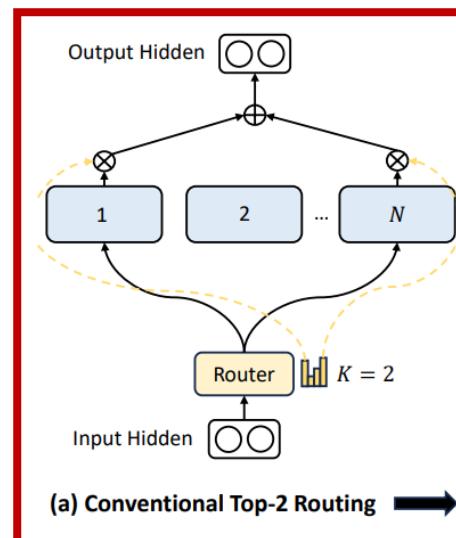
### ➤ 公式一：“传统” MoE

$$\mathbf{h}_t^l = \sum_{i=1}^N \left( g_{i,t} \text{FFN}_i \left( \mathbf{u}_t^l \right) \right) + \mathbf{u}_t^l,$$

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

$$s_{i,t} = \text{Softmax}_i \left( \mathbf{u}_t^{lT} \mathbf{e}_i^t \right),$$

➤ 其中  $\mathbf{e}_i^l$  是某个专家处理的 tokens 的质心，训练时可以通过滑动均值更新



$$C_i^t = \alpha C_i^{t-1} + (1 - \alpha) \cdot \frac{1}{|\mathcal{X}_i^t|} \sum_{x \in \mathcal{X}_i^t} x$$

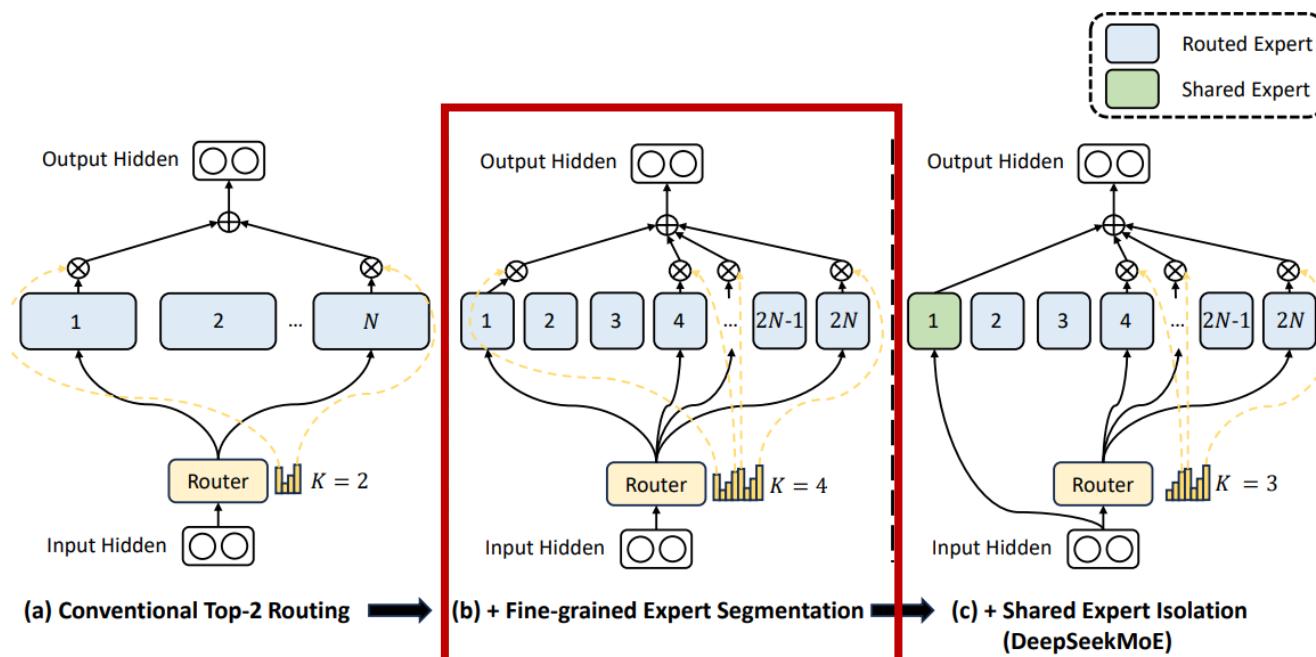
# 技术详解

- Deepseek MoE：四个公式
- 公式二：减少专家隐层维度

$$\mathbf{h}_t^l = \sum_{i=1}^{mN} \left( g_{i,t} \text{FFN}_i \left( \mathbf{u}_t^l \right) \right) + \mathbf{u}_t^l,$$

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

$$s_{i,t} = \text{Softmax}_i \left( \mathbf{u}_t^{lT} \mathbf{e}_i^l \right),$$



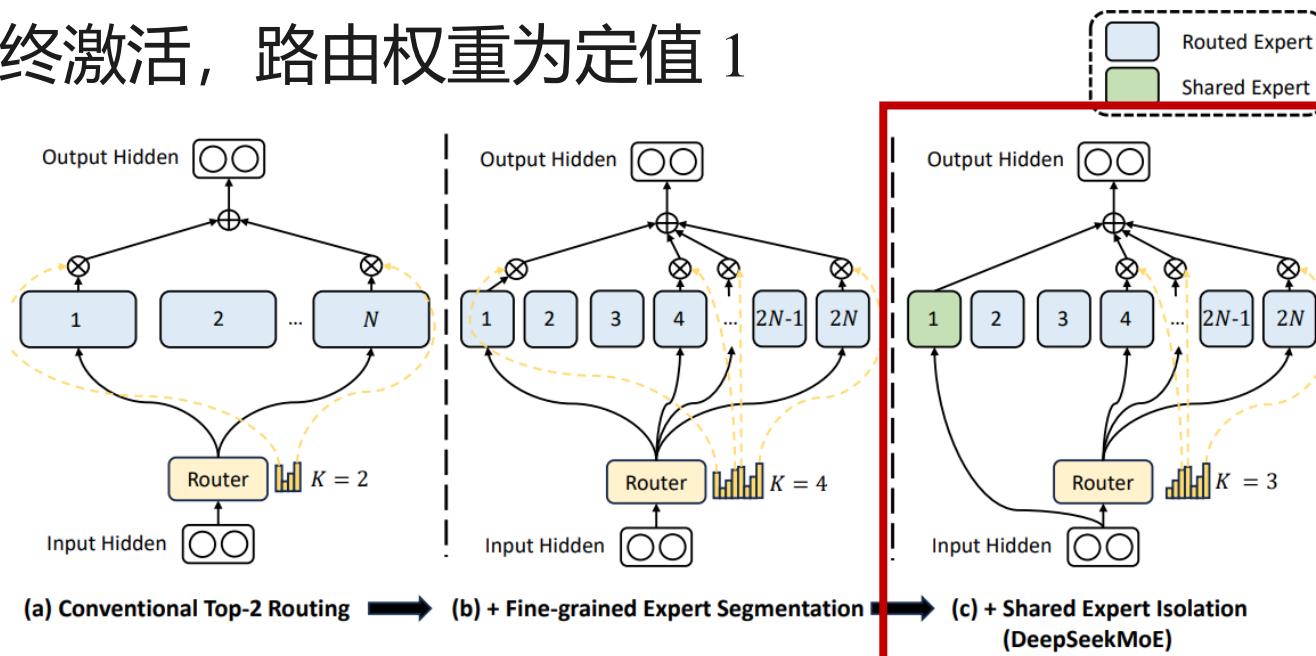
# 技术详解

## ➤ Deepseek MoE：四个公式

### ➤ 公式三：引入共享专家

$$\mathbf{h}_t^l = \sum_{i=1}^{K_s} \text{FFN}_i(\mathbf{u}_t^l) + \sum_{i=K_s+1}^{mN} \left( g_{i,t} \text{FFN}_i(\mathbf{u}_t^l) \right) + \mathbf{u}_t^l,$$
$$g_{i,t} = \begin{cases} s_{i,t}, & s_{i,t} \in \text{Topk}(\{s_{j,t} | K_s + 1 \leq j \leq mN\}, mK - K_s), \\ 0, & \text{otherwise,} \end{cases}$$
$$s_{i,t} = \text{Softmax}_i(\mathbf{u}_t^{lT} \mathbf{e}_i^l).$$

### ➤ 共享专家始终激活，路由权重为定值 1



# 技术详解

## ➤ Deepseek MoE：四个公式

### ➤ 公式四：辅助损失

### ➤ 专家层面：

$$\mathcal{L}_{\text{DevBal}} = \alpha_2 \sum_{i=1}^D f'_i P'_i, \quad (15)$$

$$f'_i = \frac{1}{|\mathcal{E}_i|} \sum_{j \in \mathcal{E}_i} f_j, \quad (16)$$

$$P'_i = \sum_{j \in \mathcal{E}_i} P_j, \quad (17)$$

where  $\alpha_2$  is a hyper-parameter called device-level balance factor. In practice, we set a small expert-level balance factor to mitigate the risk of routing collapse, and meanwhile set a larger device-level balance factor to promote balanced computation across the devices.

$$\mathcal{L}_{\text{DevBal}} = \alpha_2 \sum_{i=1}^D f'_i P'_i, \quad (15)$$

$$f'_i = \frac{1}{|\mathcal{E}_i|} \sum_{j \in \mathcal{E}_i} f_j, \quad (16)$$

$$P'_i = \sum_{j \in \mathcal{E}_i} P_j, \quad (17)$$

where  $\alpha_2$  is a hyper-parameter called device-level balance factor. In practice, we set a small expert-level balance factor to mitigate the risk of routing collapse, and meanwhile set a larger device-level balance factor to promote balanced computation across the devices.

### ➤ 设备层面：

### ➤ V2 中进一步引入了设备间通信的损失

# 技术详解

## ➤ Deepseek MoE：效果

➤ 大幅度减少计算量；计算量和模型表现之间的性价比更好

➤ 与 LLM 中提到的新 scale law 似乎有联系：

➤ 回顾：Deepseek LLM 提出用 non-embedding FLOPs per token 代替原本的参数量 N

Metric	# Shot	Dense	Hash Layer	Switch	GShard	DeepSeekMoE
# Total Params	N/A	0.2B	2.0B	2.0B	2.0B	2.0B
# Activated Params	N/A	0.2B	0.2B	0.2B	0.3B	0.3B
FLOPs per 2K Tokens	N/A	2.9T	2.9T	2.9T	4.3T	4.3T
# Training Tokens	N/A	100B	100B	100B	100B	100B
Pile (Loss)	N/A	2.060	1.932	1.881	1.867	1.808
HellaSwag (Acc.)	0-shot	38.8	46.2	49.1	50.5	54.8
PIQA (Acc.)	0-shot	66.8	68.4	70.5	70.6	72.3
ARC-easy (Acc.)	0-shot	41.0	45.3	45.9	43.9	49.4
ARC-challenge (Acc.)	0-shot	26.0	28.2	30.2	31.6	34.3
RACE-middle (Acc.)	5-shot	38.8	38.8	43.6	42.1	44.0
RACE-high (Acc.)	5-shot	29.0	30.0	30.9	30.4	31.7
HumanEval (Pass@1)	0-shot	0.0	1.2	2.4	3.7	4.9
MBPP (Pass@1)	3-shot	0.2	0.6	0.4	0.2	2.2
TriviaQA (EM)	5-shot	4.9	6.5	8.9	10.2	16.6
NaturalQuestions (EM)	5-shot	1.4	1.4	2.5	3.2	5.7

Metric	# Shot	GShard×1.5	Dense×16	DeepSeekMoE
Relative Expert Size	N/A	1.5	1	0.25
# Experts	N/A	0 + 16	16 + 0	1 + 63
# Activated Experts	N/A	0 + 2	16 + 0	1 + 7
# Total Expert Params	N/A	2.83B	1.89B	1.89B
# Activated Expert Params	N/A	0.35B	1.89B	0.24B
FLOPs per 2K Tokens	N/A	5.8T	24.6T	4.3T
# Training Tokens	N/A	100B	100B	100B
Pile (Loss)	N/A	1.808	1.806	1.808
HellaSwag (Acc.)	0-shot	54.4	55.1	54.8
PIQA (Acc.)	0-shot	71.1	71.9	72.3
ARC-easy (Acc.)	0-shot	47.3	51.9	49.4
ARC-challenge (Acc.)	0-shot	34.1	33.8	34.3
RACE-middle (Acc.)	5-shot	46.4	46.3	44.0
RACE-high (Acc.)	5-shot	32.4	33.0	31.7
HumanEval (Pass@1)	0-shot	3.0	4.3	4.9
MBPP (Pass@1)	3-shot	2.6	2.2	2.2
TriviaQA (EM)	5-shot	15.7	16.5	16.6
NaturalQuestions (EM)	5-shot	4.7	6.3	5.7

# 技术详解

- Deepseek MoE：效果
- 与 LLaMA2-7B 的对比：超大的计算优势

Metric	# Shot	LLaMA2 SFT 7B	DeepSeek Chat 7B	DeepSeekMoE Chat 16B
# Total Params	N/A	6.7B	6.9B	16.4B
# Activated Params	N/A	6.7B	6.9B	2.8B
FLOPs per 4K Tokens	N/A	187.9T	183.5T	74.4T
HellaSwag (Acc.)	0-shot	67.9	71.0	72.2
PIQA (Acc.)	0-shot	76.9	78.4	79.7
ARC-easy (Acc.)	0-shot	69.7	70.2	69.9
ARC-challenge (Acc.)	0-shot	50.8	50.2	50.0
BBH (EM)	3-shot	39.3	43.1	42.2
RACE-middle (Acc.)	5-shot	63.9	66.1	64.8
RACE-high (Acc.)	5-shot	49.6	50.8	50.6
DROP (EM)	1-shot	40.0	41.7	33.8
GSM8K (EM)	0-shot	63.4	62.6	62.2
MATH (EM)	4-shot	13.5	14.7	15.2
HumanEval (Pass@1)	0-shot	35.4	45.1	45.7
MBPP (Pass@1)	3-shot	27.8	39.0	46.2
TriviaQA (EM)	5-shot	60.1	59.5	63.3
NaturalQuestions (EM)	0-shot	35.2	32.7	35.1
MMLU (Acc.)	0-shot	50.0	49.7	47.2
WinoGrande (Acc.)	0-shot	65.1	68.4	69.0
CLUEWSC (EM)	5-shot	48.4	66.2	68.2
CEval (Acc.)	0-shot	35.1	44.7	40.0
CMMLU (Acc.)	0-shot	36.9	51.2	49.3

# 技术详解

- Deepseek MoE：效果
- “初见端倪”：一年后震撼硅谷的技术起点

Metric	# Shot	DeepSeek 67B (Dense)	GShard 137B	DeepSeekMoE 145B	DeepSeekMoE 142B (Half Activated)
# Total Params	N/A	67.4B	136.5B	144.6B	142.3B
# Activated Params	N/A	67.4B	21.6B	22.2B	12.2B
Relative Expert Size	N/A	N/A	1	0.125	0.125
# Experts	N/A	N/A	0 + 16	4 + 128	2 + 128
# Activated Experts	N/A	N/A	0 + 2	4 + 12	2 + 6
FLOPs per 4K Tokens	N/A	2057.5T	572.7T	585.6T	374.6T
# Training Tokens	N/A	245B	245B	245B	245B
Pile (Loss.)	N/A	1.905	1.961	<b>1.876</b>	1.888
HellaSwag (Acc.)	0-shot	74.8	72.0	<b>75.8</b>	74.9
PIQA (Acc.)	0-shot	79.8	77.6	<b>80.7</b>	80.2
ARC-easy (Acc.)	0-shot	69.0	64.0	<b>69.7</b>	67.9
ARC-challenge (Acc.)	0-shot	<b>50.4</b>	45.8	48.8	49.0
RACE-middle (Acc.)	5-shot	<b>63.2</b>	59.2	62.1	59.5
RACE-high (Acc.)	5-shot	<b>46.9</b>	43.5	45.5	42.6
DROP (EM)	1-shot	<b>27.5</b>	21.6	<b>27.8</b>	28.9
GSM8K (EM)	8-shot	<b>11.8</b>	6.4	<b>12.2</b>	13.8
MATH (EM)	4-shot	2.1	1.6	<b>3.1</b>	2.8
HumanEval (Pass@1)	0-shot	<b>23.8</b>	17.7	19.5	23.2
MBPP (Pass@1)	3-shot	<b>33.6</b>	27.6	<b>33.2</b>	32.0
TriviaQA (EM)	5-shot	57.2	52.5	<b>61.1</b>	59.8
NaturalQuestions (EM)	5-shot	22.6	19.0	<b>25.0</b>	23.5
MMLU (Acc.)	5-shot	<b>45.1</b>	26.3	39.4	37.5
WinoGrande (Acc.)	0-shot	70.7	67.6	<b>71.9</b>	70.8
CLUEWSC (EM)	5-shot	69.1	65.7	<b>71.9</b>	72.6
CEval (Acc.)	5-shot	<b>40.3</b>	26.2	37.1	32.8
CMMLU (Acc.)	5-shot	<b>40.6</b>	25.4	35.9	31.9
CHID (Acc.)	0-shot	88.5	86.9	<b>90.3</b>	88.3

# 技术详解

## ➤ Deepseek Coder：填空方法

➤ 中间填空 (Fill-in-the-Middle, FIM)：FIM方法将代码分成前缀、中间和后缀，通过重新排列来训练模型填补中间部分

- 中间填空 (Fill-in-the-Middle, FIM)：

- PSM模式：将代码分割为前缀 (Prefix)、后缀 (Suffix)、中间 (Middle)，重组为 `<f_pre><f_suf><f_middle>`，通过特殊标记 (如 `<|fim_start|>`) 分隔。
- 数学实现：对代码片段  $f$ ，随机选择分割点  $s$ ，构造训练样本：

Input:  $f_{\text{pre}} \oplus f_{\text{suf}}$ , Target:  $f_{\text{middle}}$

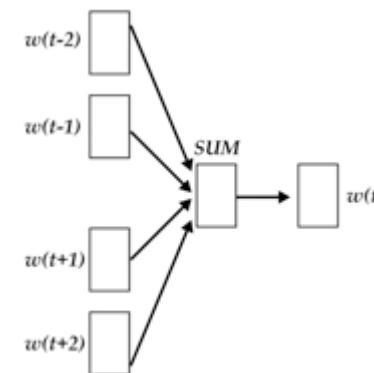
- FIM率：实验中50%的样本使用FIM，平衡代码生成与填充能力。

`<|fim_begin|> $f_{\text{pre}}$ <|fim_hole|> $f_{\text{suf}}$ <|fim_end|> $f_{\text{middle}}$ <|eos_token|>`.

➤ 训练时需要调整 FIM 和 Next Token Prediction 两种数据的配比

# 技术详解

- Deepseek Math：“R1 之父”
  - 基于 DeepSeek-Coder-Base-v1.5 7B 进行 SFT，数据中集成CoT、PoT、tool等
  - 自动化数据选择：fastText 分类器（类似词嵌入中的 CBOW 的模型架构）



- 致胜一手：GRPO（组关联策略优化）

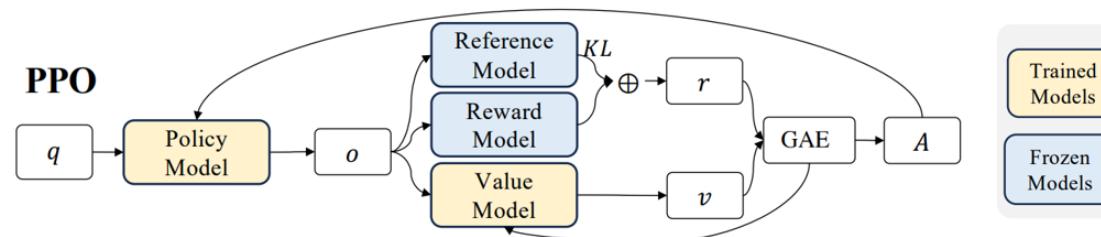
# 技术详解

## ➤ GRPO (组关联策略优化) : 强化学习发展新章

### ➤ 回顾: PPO in LLM

$$\mathcal{J}_{PPO}(\theta) = \mathbb{E}[q \sim P(Q), o \sim \pi_{\theta_{old}}(O|q)] \frac{1}{|o|} \sum_{t=1}^{|o|} \min \left[ \frac{\pi_\theta(o_t|q, o_{<t})}{\pi_{\theta_{old}}(o_t|q, o_{<t})} A_t, \text{clip} \left( \frac{\pi_\theta(o_t|q, o_{<t})}{\pi_{\theta_{old}}(o_t|q, o_{<t})}, 1 - \varepsilon, 1 + \varepsilon \right) A_t \right]$$

- Reference Model: 经过 SFT 的 LLM
- Reward Model: 打分模型, 可以通过 LLM + 线性层来得到。数据由 SFT 后的模型输出 (可以加上其他数据, 比如人类标注数据), 由人类打分得到。
- Value Model: 价值估计模型
- Policy Model: 正在训练的 LLM



- Reward Model 只给最后一个 token 打分, 但却要 Value Model 准确估计每个 token 上的预期 Reward。不划算的计算量!

# 技术详解

## ➤ GRPO (组关联策略优化) : 强化学习发展新章

➤ GRPO 的目标函数:  $\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^G \sim \pi_{\theta_{old}}(O|q)]$

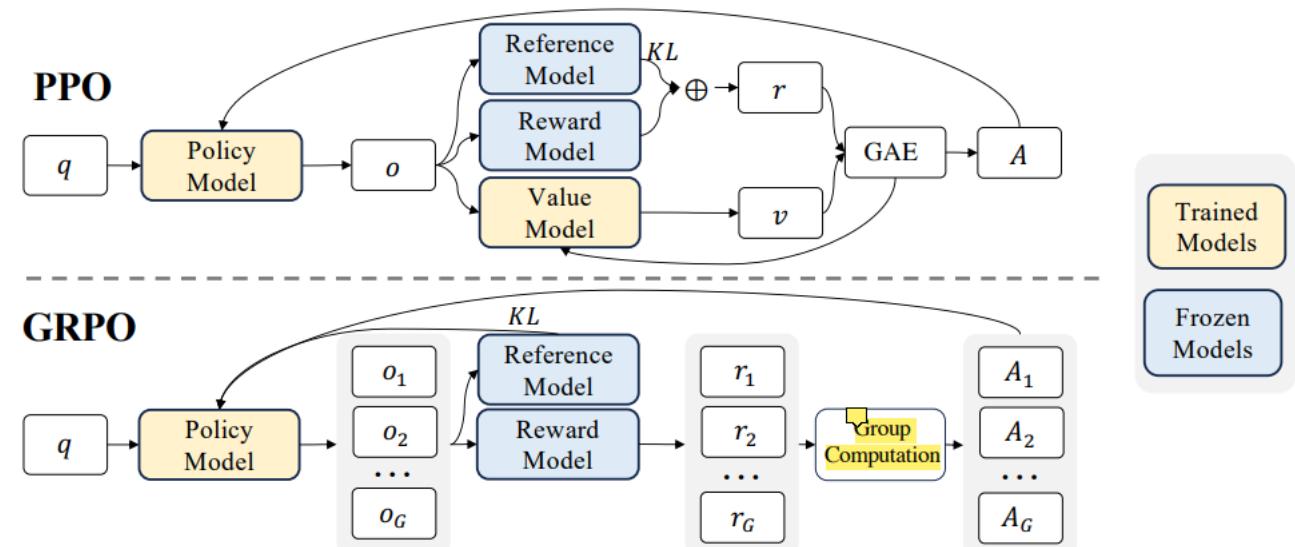
$$\frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \left\{ \min \left[ \frac{\pi_\theta(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q, o_{i,<t})} \hat{A}_{i,t}, \text{clip} \left( \frac{\pi_\theta(o_{i,t}|q, o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q, o_{i,<t})}, 1 - \varepsilon, 1 + \varepsilon \right) \hat{A}_{i,t} \right] - \beta \mathbb{D}_{KL} [\pi_\theta || \pi_{ref}] \right\}$$

- 相对于 PPO 的改动: 分组生成输出, 计算 reward 和优势函数, 每一组的 Loss 都是 PPO Loss, 然后求平均
- 用正则化 Reward 计算优势函数:

$$\hat{A}_{i,t} = \tilde{r}_i = \frac{r_i - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r})}$$

➤ 两种策略:

1. 每组一个 sequence reward, 对组做正则化
2. 每个 token 一个 reward, 组内正则化后对时间步 t 之后的 token reward 求和



# 技术详解

## ➤ GRPO (组关联策略优化) : 强化学习发展新章

### ➤ 算法:

---

#### Algorithm 1 Iterative Group Relative Policy Optimization

---

**Input** initial policy model  $\pi_{\theta_{\text{init}}}$ ; reward models  $r_\varphi$ ; task prompts  $\mathcal{D}$ ; hyperparameters  $\varepsilon, \beta, \mu$

- 1: policy model  $\pi_\theta \leftarrow \pi_{\theta_{\text{init}}}$
- 2: **for** iteration = 1, ..., I **do**
- 3:   reference model  $\pi_{\text{ref}} \leftarrow \pi_\theta$
- 4:   **for** step = 1, ..., M **do**
- 5:     Sample a batch  $\mathcal{D}_b$  from  $\mathcal{D}$
- 6:     Update the old policy model  $\pi_{\theta_{\text{old}}} \leftarrow \pi_\theta$
- 7:     Sample G outputs  $\{o_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot | q)$  for each question  $q \in \mathcal{D}_b$
- 8:     Compute rewards  $\{r_i\}_{i=1}^G$  for each sampled output  $o_i$  by running  $r_\varphi$
- 9:     Compute  $\hat{A}_{i,t}$  for the t-th token of  $o_i$  through group relative advantage estimation.
- 10:    **for** GRPO iteration = 1, ...,  $\mu$  **do**
- 11:      Update the policy model  $\pi_\theta$  by maximizing the GRPO objective (Equation 21)
- 12:    Update  $r_\varphi$  through continuous training using a replay mechanism.

---

**Output**  $\pi_\theta$

---

### ➤ Row 12: 迭代式强化学习

- 采用经验回放策略，每次保留 10% 旧数据，引入新数据重训 Reward Model
- 防止过时的 Reward Model 限制策略更新

# 技术详解

➤ GRPO (组关联策略优化) : 强化学习发展新章

➤ 另外的细节无偏 KL 散度 (token level) :

$$D_{KL} [\pi_\theta || \pi_{ref}] = \frac{\pi_{ref}(o_{i,t}|q, o_{i,<t})}{\pi_\theta(o_{i,t}|q, o_{i,<t})} - \log \frac{\pi_{ref}(o_{i,t}|q, o_{i,<t})}{\pi_\theta(o_{i,t}|q, o_{i,<t})} - 1,$$

➤ 原本的 KL 散度:

$$D_{KL} [\pi_\theta || \pi_{ref}] = \sum_{o_{i,t}} \pi_\theta(o_{i,t}|q, o_{i,<t}) \log \frac{\pi_\theta(o_{i,t}|q, o_{i,<t})}{\pi_{ref}(o_{i,t}|q, o_{i,<t})}$$

➤ GRPO 不再依赖 Value Model, 计算效率大幅度提升

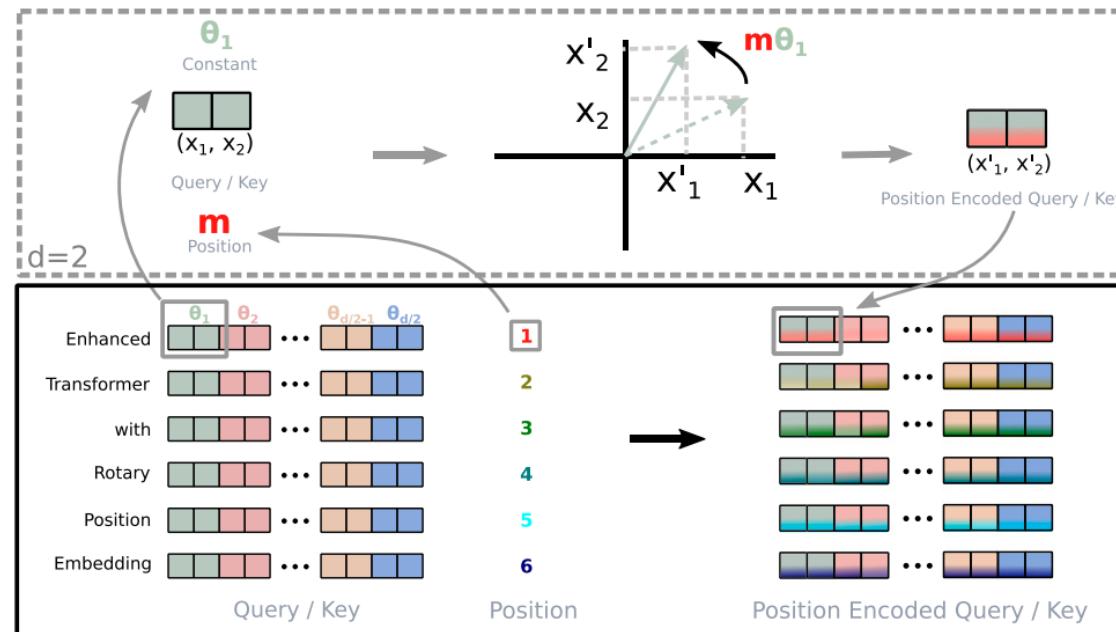
# 技术详解

- Deepseek V2：技术大爆发——震撼硅谷，近在眼前
  - 进一步 Scale dpsk\_moe 架构：总参数量236B（激活21B/Token），相比DeepSeek 67B（dense）节省42.5%训练成本（172.8K vs 300.6K GPU小时/万亿Token）
  - 长上下文扩展：使用 YaRN 方法将上下文窗口从4K扩展至128K
  - 新的注意力机制 MLA
    - 推理效率提升：KV缓存量化：将KV缓存压缩至6比特/元素，结合MLA实现93.3%缓存减少。
    - 生成吞吐量：单节点（ $8 \times H800$ ）最大生成吞吐量达50K Token/秒，为DeepSeek 67B的5.76倍
  - 二阶段强化学习：解耦 Reward
  - SFT 经验：SFT 需要足量数据 (>10K)

# 技术详解

## ➤ YaRN：高效位置编码与更好的上下文扩展

➤ 回顾 RoPE:  $q_m^T k_n = \left( R_{\Theta, m}^d W_q x_m \right)^T \left( R_{\Theta, n}^d W_k x_n \right) = x_m^T W_q R_{\Theta, n-m}^d W_k x_n \quad R_{\Theta, n-m}^d = \left( R_{\Theta, m}^d \right)^T R_{\Theta, n}^d$



➤ 同一的位置编码范式:  $f'_W(x_m, m, \theta_d) = f_W(x_m, g(m), h(\theta_d))$

$$\text{RoPE: } \begin{aligned} g(m) &= m \\ h(\theta_d) &= \theta_d \end{aligned} \quad f_W(x_m, m, \theta_d) = \begin{bmatrix} \cos(m\theta_d) & -\sin(m\theta_d) \\ \sin(m\theta_d) & \cos(m\theta_d) \end{bmatrix} W x_m$$

# 技术详解

## ➤ YaRN: Background

➤ 频率参数:  $\theta_d = b^{-2d/|D|}$

- 其中:
  - $b$  是一个固定的基数 (通常取  $b = 10000$ )
  - $|D|$  是隐藏层的维度数。
  - $d$  表示隐藏层的某一具体维度。

## ➤ RoPE 的局限性:

➤ 频率不变性: 频率参数在预训练时被固定, 无法适应更长的上下文长度。

➤ 频率分布的刚性: 所有维度的频率分布固定, 不支持动态调整, 导致当序列长度超出预训练范围时, 旋转编码出现混乱。

➤ 线性内插:  $\cdot g(m) = m/s$ , 其中  $s = L'/L$  是上下文扩展比例。  $f'_W(x_m, m, \theta_d) = f_W(x_m, \frac{m}{s}, \theta_d)$

➤ 通过线性缩放将窗口拉回预训练时的长度

➤ 神经正切核理论 (NTK) : 非常复杂, 这里省略

# 技术详解

## ➤ YaRN: Background (关系不大的一点前置)

### ➤ NTK-Aware Interpolation:

$$g(m) = m$$

$$h(\theta_d) = b^{-2d/|D|} \cdot s^{-2d/|D|}$$

这里  $d = 0$  时,  $h(\theta_d) = 1$ , 有没有  $s$  都不影响, 因此也叫直接外推。

这里  $d = |D|/2$  时,  $h(\theta_d) = b^{-1} \cdot s^{-1}$ , 变成了线性内插。

### ➤ Dynamic NTK:

$$g(m) = m$$

$$h(\theta_d) = b^{-2d/|D|} \cdot (as - a + 1)^{-2d/|D|}$$

或者采用指数形式调整:

$$h(\theta_d) = b^{-2i/d} \cdot \exp(-a \cdot (2i + 1)^b)$$

其中  $a(d/2)^b = \log k$ ,  $k$  是和 NTK 理论相关的参数

# 技术详解

## ➤ YaRN: Method

### ➤ NTK-by-parts Interpolation:

➤ 波长:  $\lambda_d = \frac{2\pi}{\theta_d} = 2\pi b^{\frac{2d}{|D|}}$  (其中  $\theta_d = b^{-2d/|D|}$ ,  $b = 10000$ )

➤ 原始上下文长度与波长的比值:  $r(d) = \frac{L}{\lambda_d}$

➤ 沿用原始位置:  $g(m) = m$

➤ 分段变换频率参数:  $h(\theta_d) = (1 - \gamma(r(d)))\frac{\theta_d}{s} + \gamma(r(d))\theta_d$

$$\gamma(r) = \begin{cases} 0, & \text{if } r < \alpha \\ 1, & \text{if } r > \beta \\ \frac{r-\alpha}{\beta-\alpha}, & \text{otherwise} \end{cases}$$

➤ 高频线性插值; 低频 RoPE; 中频加权平均

### ➤ YaRN: 一点小小的改动

➤ 在注意力机制中引入温度参数  $t$ :  $\text{softmax}\left(\frac{q_m^T k_n}{t\sqrt{|D|}}\right)$ ,  $\sqrt{\frac{1}{t}} = 0.1 \ln(s) + 1$

➤ 超低训练成本 (0.1% 预训练数据配比即可)。

➤ 几乎兼容目前所有主流的Transformer实现。

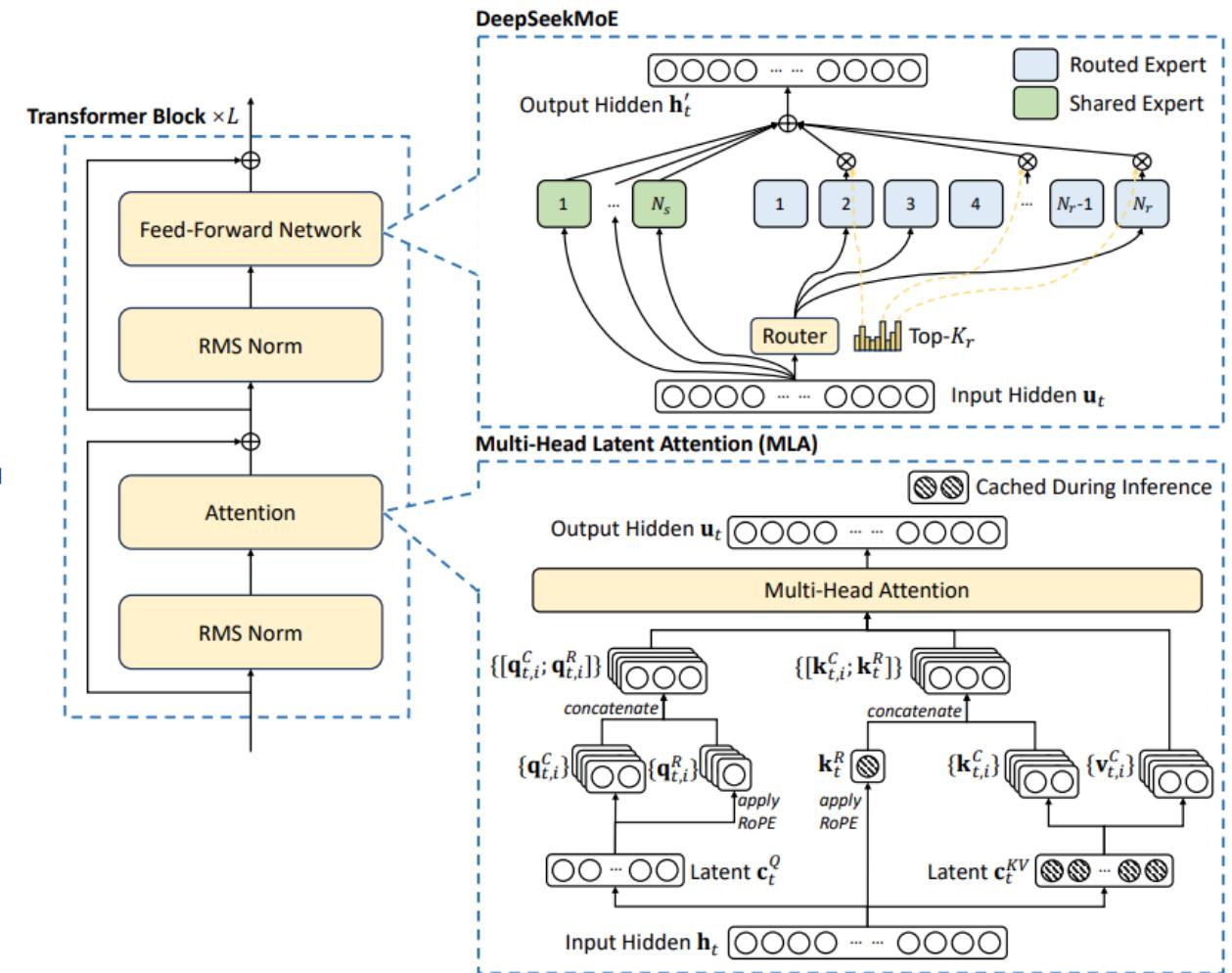
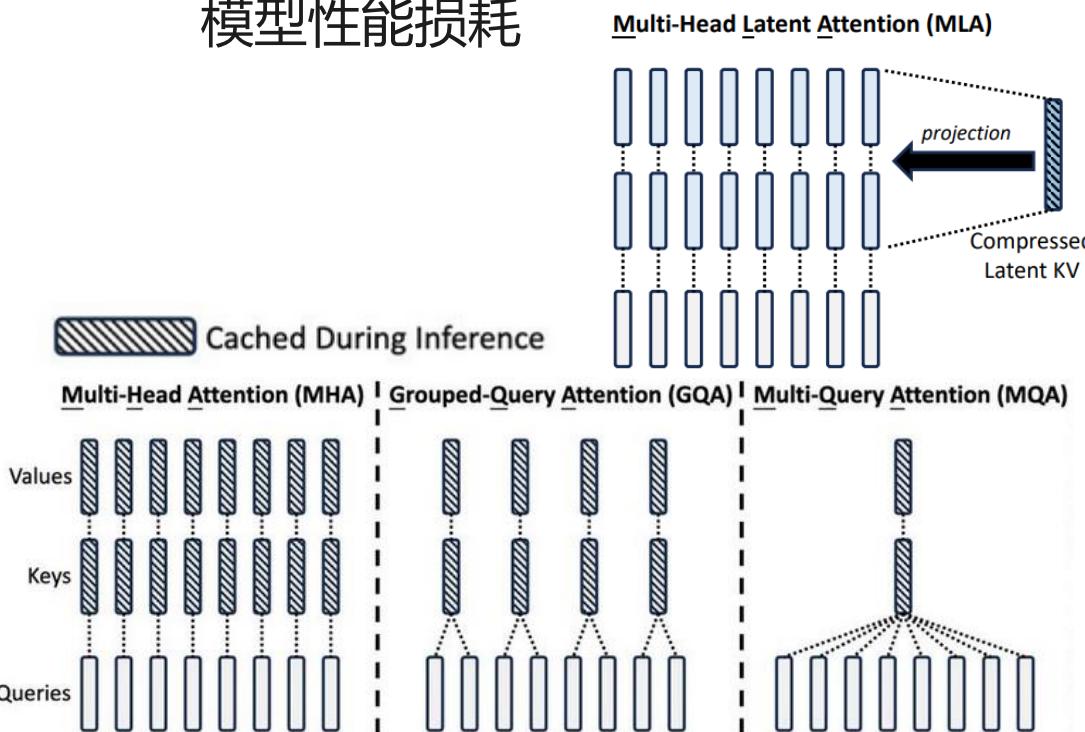
➤ 性能优越, 无论是否微调均能在 128k 上下文中表现出色 (大海捞针实验证)

# 技术详解

## ➤ MLA (Multi-head Latent Attention) : Overview

### ➤ Motivation:

- 降低推理时 KV Cache 的存储开销
- 缓解 GQA 和 MQA 等方法导致的模型性能损耗



# 技术详解

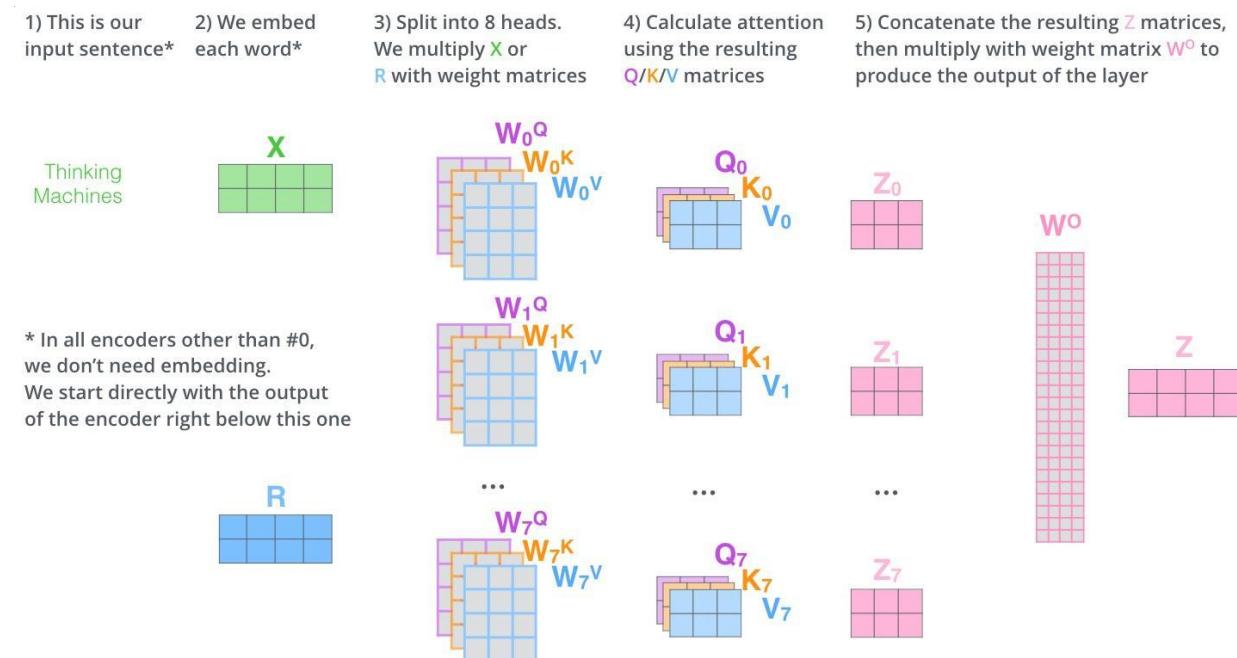
## ➤ MLA (Multi-head Latent Attention)

### ➤ Multi-head Attention:

$$\mathbf{q}_t = W^Q \mathbf{h}_t, \quad \mathbf{k}_t = W^K \mathbf{h}_t, \quad \mathbf{v}_t = W^V \mathbf{h}_t,$$

$$[\mathbf{q}_{t,1}; \mathbf{q}_{t,2}; \dots; \mathbf{q}_{t,n_h}] = \mathbf{q}_t, \\ [\mathbf{k}_{t,1}; \mathbf{k}_{t,2}; \dots; \mathbf{k}_{t,n_h}] = \mathbf{k}_t, \\ [\mathbf{v}_{t,1}; \mathbf{v}_{t,2}; \dots; \mathbf{v}_{t,n_h}] = \mathbf{v}_t, \\ \mathbf{o}_{t,i} = \sum_{j=1}^t \text{Softmax}_j \left( \frac{\mathbf{q}_{t,i}^T \mathbf{k}_{j,i}}{\sqrt{d_h}} \right) \mathbf{v}_{j,i}, \\ \mathbf{u}_t = W^O [\mathbf{o}_{t,1}; \mathbf{o}_{t,2}; \dots; \mathbf{o}_{t,n_h}],$$

这里  $\mathbf{q}_{t,i}, \mathbf{k}_{t,i}, \mathbf{v}_{t,i} \in \mathbb{R}^{d_h}$  分别表示query、key和value的第  $i$  个attention head,  
 $W^O \in \mathbb{R}^{d \times d_h n_h}$  表示输出映射矩阵。那么如果模型结构是MHA，那么在推理时，**KV Cache**对  
于每个token需要缓存的参数有  $2n_h d_h l$ 。（这里  $l$  表示网络层数）



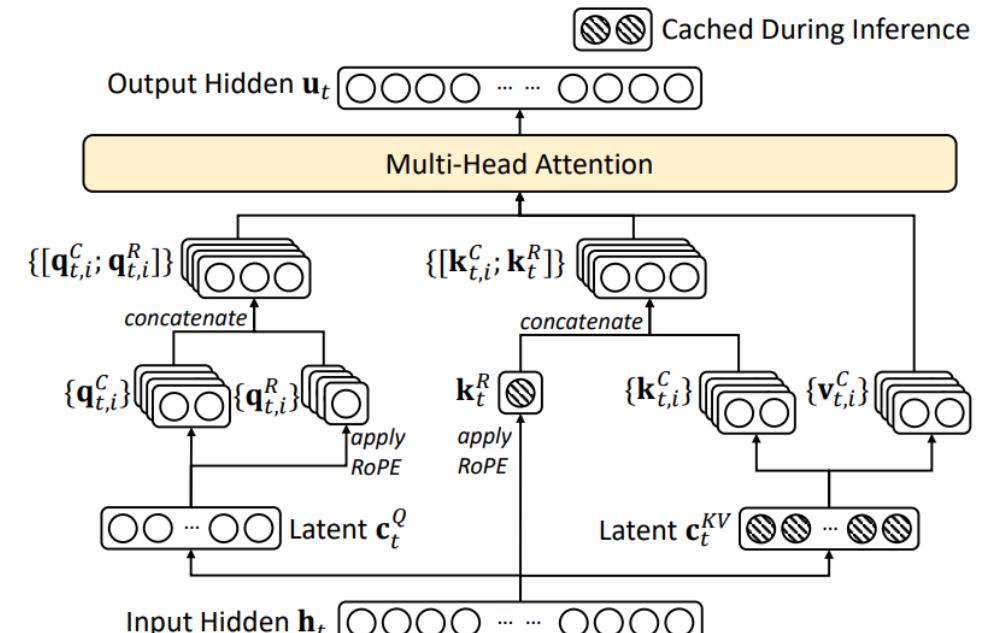
# 技术详解

- MLA (Multi-head Latent Attention) : 站在 LoRA 的肩膀上
  - 对 K 和 V 进行低秩联合压缩:

$$\begin{aligned}\mathbf{c}_t^{KV} &= W^{DKV} \mathbf{h}_t, & \mathbf{c}_t^Q &= W^{DQ} \mathbf{h}_t, \\ \mathbf{k}_t^C &= W^{UK} \mathbf{c}_t^{KV}, & \mathbf{q}_t^C &= W^{UQ} \mathbf{c}_t^Q, \\ \mathbf{v}_t^C &= W^{UV} \mathbf{c}_t^{KV},\end{aligned}$$

- 其中，上标以 U 开头的是升维投影矩阵；以 D 开头的则是降维投影矩阵
- 压缩 K 和 V：降低 KV cache
- 压缩 Q：降低激活内存

这样在推理时，只需要缓存隐向量  $\mathbf{c}_t^{KV}$  即可，这样MLA对应的每一个token的KV Cache参数只有  $d_c l$  个。其中， $d_c (\ll d_h n_h)$



# 技术详解

## ➤ MLA (Multi-head Latent Attention) : 站在 LoRA 的肩膀上

### ➤ 解耦 RoPE: 为什么?

$$\begin{aligned}\tilde{Q} &= R_m Q & S &= \tilde{Q}^T \tilde{K} \\ \tilde{K} &= R_n K & &= Q^T R_m^T R_n K\end{aligned}$$

### ➤ 如果不解耦.....推理时

#### ➤ 对压缩态的 $C_{KV}$ 应用 RoPE:

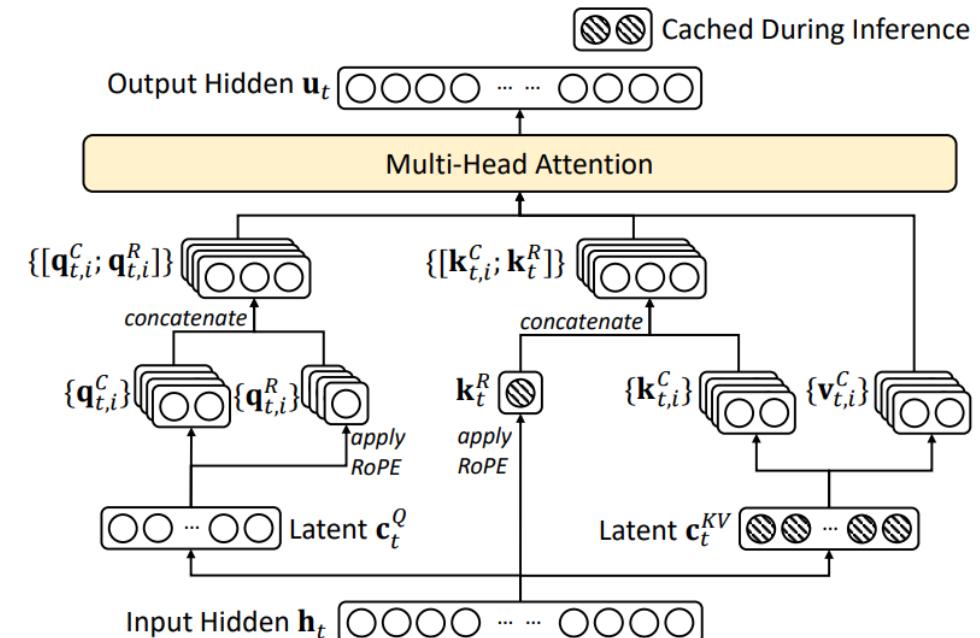
$$S = C_Q^T (W^{UQ})^T R_m^T W^{UK} R_n C_{KV}$$

此时两个投影矩阵被一个与当前生成的 token 位置有关的 旋转矩阵 分隔，  
于是升维过程变得与 token 位置相关

#### ➤ 对恢复到高维后的向量应用 RoPE:

$$S = C_Q^T (W^{UQ})^T R_m^T R_n W^{UK} C_{KV}$$

由于只有  $C_{KV}$  被缓存，实际上每次仍然需要重新计算之前的所有 token 对应 key。



# 技术详解

➤ MLA (Multi-head Latent Attention) : 站在 LoRA 的肩膀上

➤ 解耦 RoPE: Method

- 用额外的 q 和 k 记录位置信息
- 包含位置信息的 q/k 与恢复维度的 q/k 拼接
- 最后计算多头注意力

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

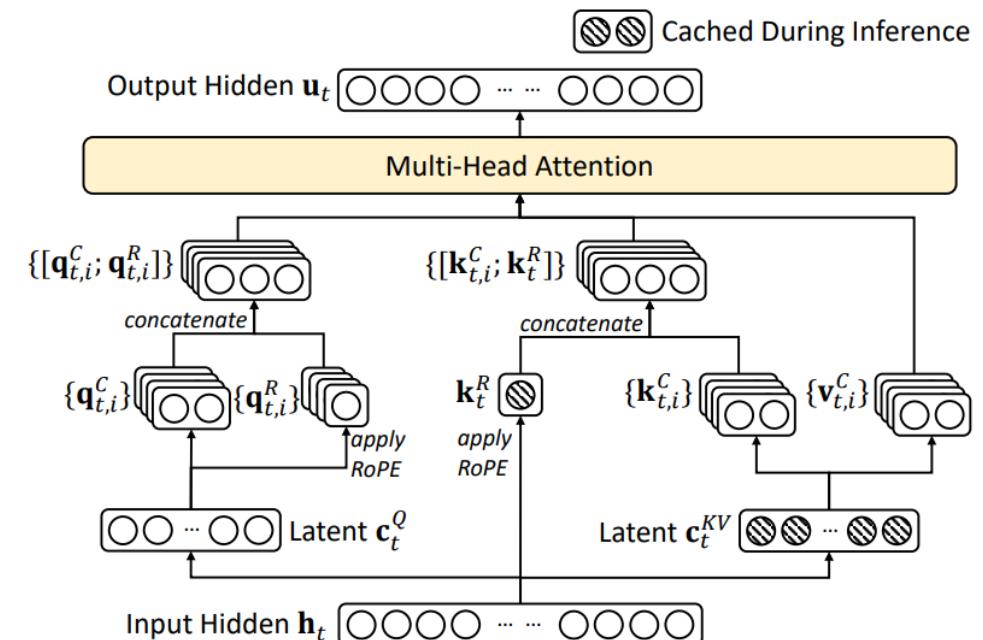
$$\mathbf{k}_t^R = \text{RoPE}(W^{KR} \mathbf{h}_t),$$

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

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

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

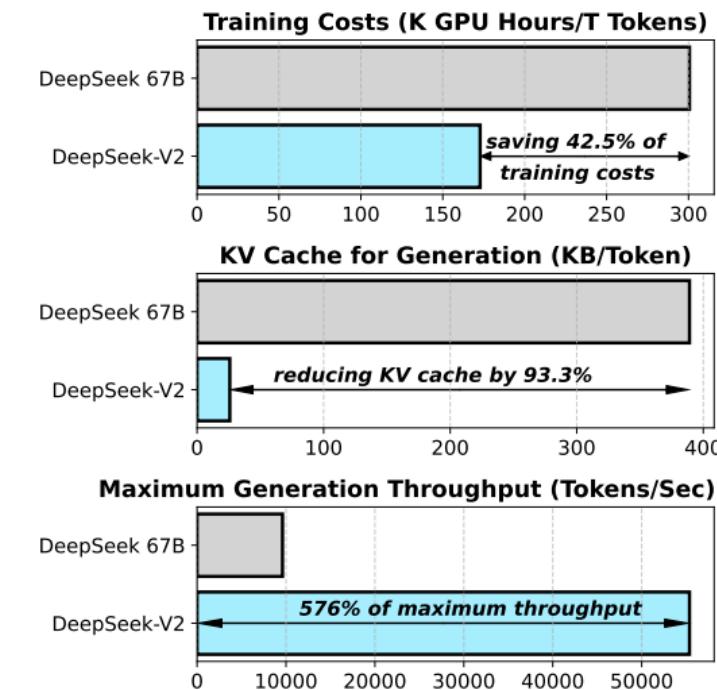
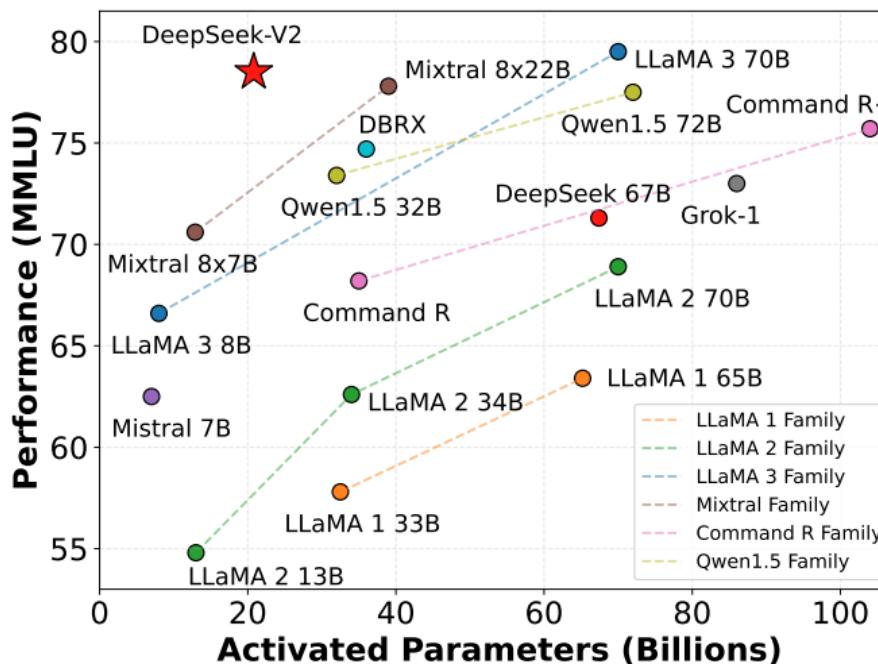
$$\mathbf{u}_t = W^O [\mathbf{o}_{t,1}; \mathbf{o}_{t,2}; \dots; \mathbf{o}_{t,n_h}],$$



# 技术详解

- MLA (Multi-head Latent Attention) : 站在 LoRA 的肩膀上
- 效果

Attention Mechanism	KV Cache per Token (# Element)	Capability
Multi-Head Attention (MHA)	$2n_h d_{hl}$	Strong
Grouped-Query Attention (GQA)	$2n_g d_{hl}$	Moderate
Multi-Query Attention (MQA)	$2d_{hl}$	Weak
MLA (Ours)	$(d_c + d_h^R)l \approx \frac{9}{2}d_{hl}$	Stronger



# 技术详解

- 二阶段 RL: Reward 的力量
  - 不同数据在强化学习下有不同特征
  - Code 和 Math 这样的推理密集型数据，在强化学习期间可以持续提升
  - 解耦 Reward 函数，对不同类型数据应用不同的 Reward 模型：

$$r_i = RM_{reasoning}(o_i).$$

$$r_i = c_1 \cdot RM_{helpful}(o_i) + c_2 \cdot RM_{safety}(o_i) + c_3 \cdot RM_{rule}(o_i),$$

- 修正和调整 Reward 的思路，在 Deepseek 之后的工作中也有体现

# 技术详解

- Deepseek V3：半年时间重塑底层设施——震撼硅谷，就在今天
  - 成功 Scale 到 671B，激活参数 37B（作为对比，同时期 Meta 的 sparse model 还在小模型阶段）
  - 原生 FP8，部署门槛大大降低
  - MoE + MLA + MTP +  
底层设施优化 (DualPipe + FP8 混合精度 + 适配分布式策略)
    - 极致降低成本、高效计算
  - 训练成本：总成本 2.788M H800 GPU 小时（约 557.6 万美元），预训练效率为 180K GPU 小时 / 万亿 token
  - 高质量数据：提高推理数据占比；从 R1 蒸馏
  - Self-Rewarding
  - 主要是 mlsys 方向以及更底层方向的优化，超低成本训出超强模型

# 技术详解

- 新的 Deepseek MoE：四个优化
  - 优化一：激活函数由 softmax 更换为 sigmoid + 路由权重归一化

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

$$g_{i,t} = \frac{g'_{i,t}}{\sum_{j=1}^{N_r} g'_{j,t}},$$

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

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

- 优化原因：一个猜想，提升数值稳定性，提升训练时的稳定性和收敛效果

# 技术详解

- 新的 Deepseek MoE：四个优化
  - 优化二：动态路由权重偏置

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

- 比较并选取 Topk 专家时，每个专家有自己的偏置项。训练时的每一步，若专家过载则减小  $\gamma$ ，否则增大  $\gamma$ 。 $\gamma$  称为偏置更新速度。
- 优化原因：负载均衡，防止引发路由崩溃（专家过载），以便专家并行提升计算效率

# 技术详解

- 新的 Deepseek MoE：四个优化
  - 优化三：删去其他辅助损失，增加一项互补的序列内辅助损失

$$\begin{aligned}\mathcal{L}_{\text{Bal}} &= \alpha \sum_{i=1}^{N_r} f_i P_i, \\ f_i &= \frac{N_r}{K_r T} \sum_{t=1}^T \mathbb{1} (s_{i,t} \in \text{Topk}(\{s_{j,t} | 1 \leq j \leq N_r\}, K_r)), \\ s'_{i,t} &= \frac{s_{i,t}}{\sum_{j=1}^{N_r} s_{j,t}}, \\ P_i &= \frac{1}{T} \sum_{t=1}^T s'_{i,t},\end{aligned}$$

- 优化原因：辅助损失过大影响模型性能，于是去除其他损失；为了提升计算效率引入新的损失（trade-off），用于防止单个 sequence 专家负载的过度不均衡

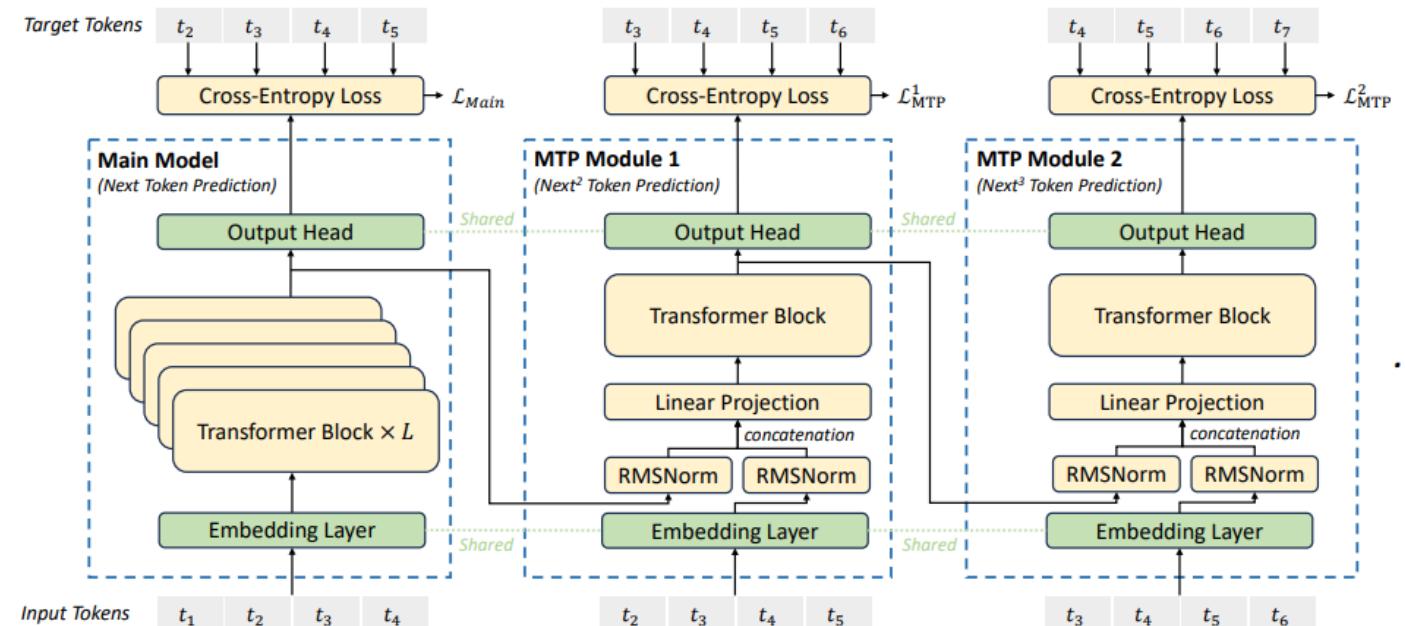
# 技术详解

- 新的 Deepseek MoE：四个优化
  - 优化四：设备间均衡改用固定的 rule（之前是用辅助损失），并且取消 token-dropping（丢弃超出负载上限的 tokens）

# 技术详解

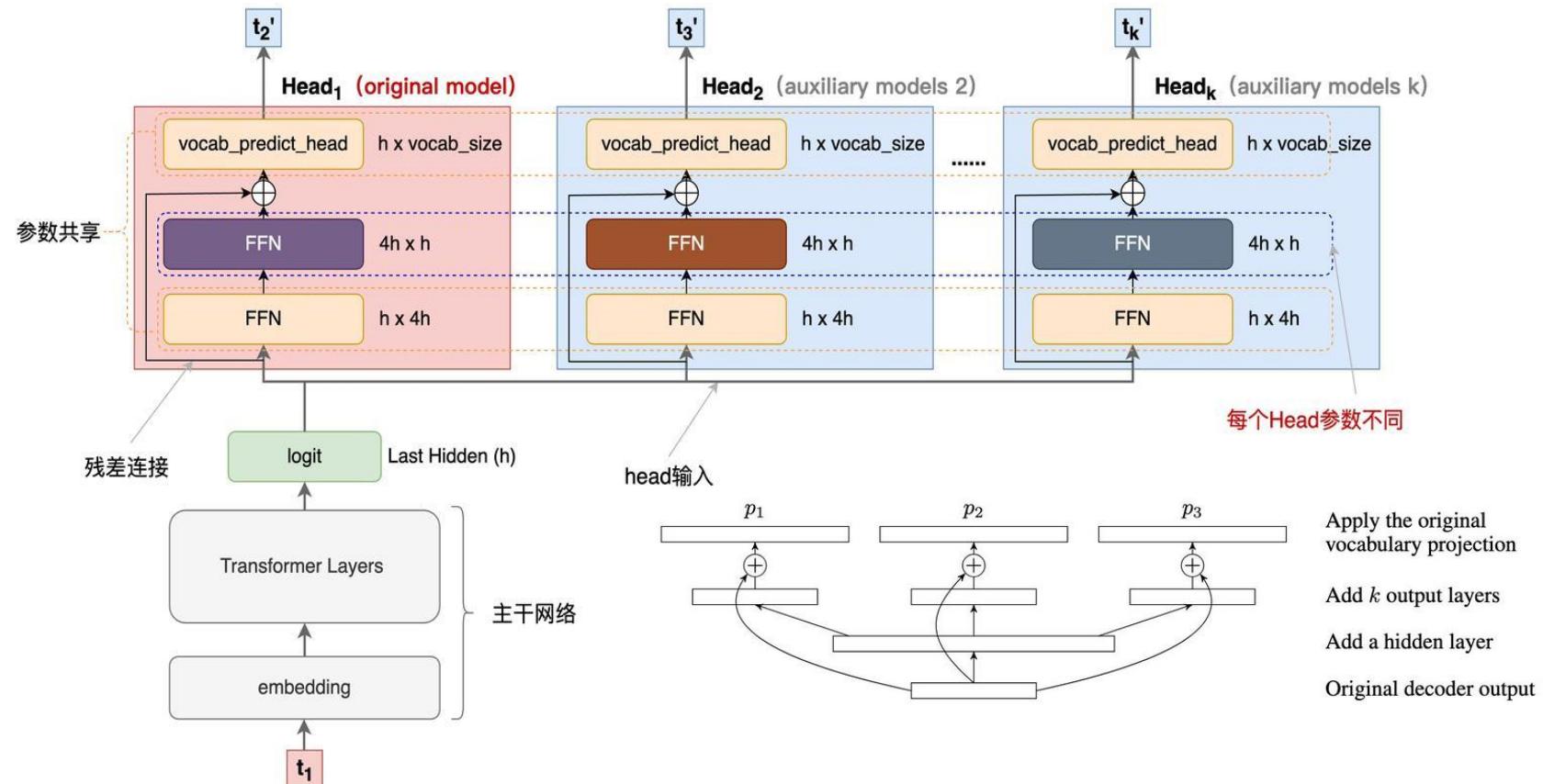
## ➤ MTP (Multi-Tokens Prediction) : Background

- 在训练阶段，一次生成多个后续token，可以一次学习多个位置的label，进而有效提升样本的利用效率，提升训练速度；
- 在推理阶段通过一次生成多个token，实现成倍的推理加速来提升推理性能。
- 提高数据样本利用率



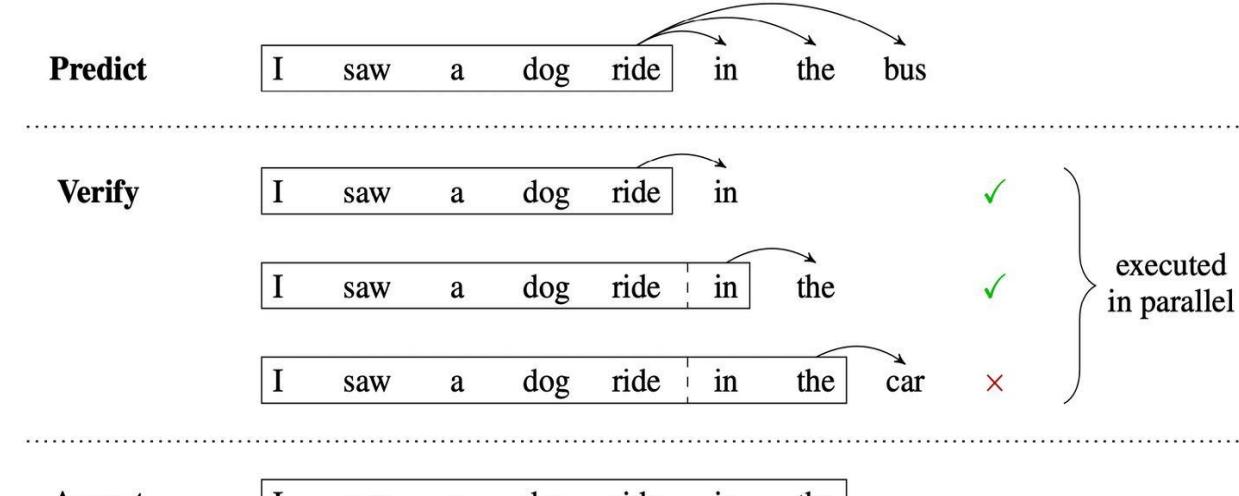
# 技术详解

- MTP (Multi-Tokens Prediction) : Block-wise Parallel Decoding
- 多头 FFN + output layer, “并行”生成多个位置的 tokens



# 技术详解

- MTP (Multi-Tokens Prediction) : Block-wise Parallel Decoding
- Google 2018 NIPS

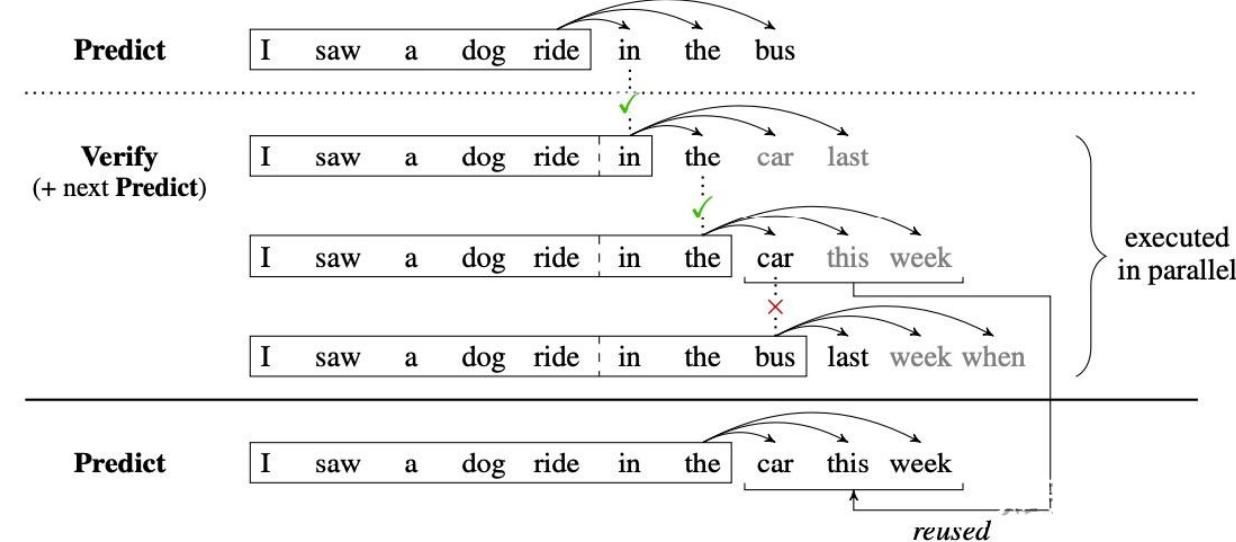


## ➤ 分阶段推理：

- 阶段一：预测，生成 k 个 tokens (单步完成)
- 阶段二：验证，每次将前  $i-1$  个生成的 tokens 和前缀序列拼接，由第一个 head (也就是预测 Next Token 的 head) 预测第  $i$  个生成的 token (封装成一个 batch，可以单步完成)
- 阶段三：接受，将生成的 tokens 序列验证通过的最长的前缀作为结果接受。
- 并行处理大幅度提高计算效率

# 技术详解

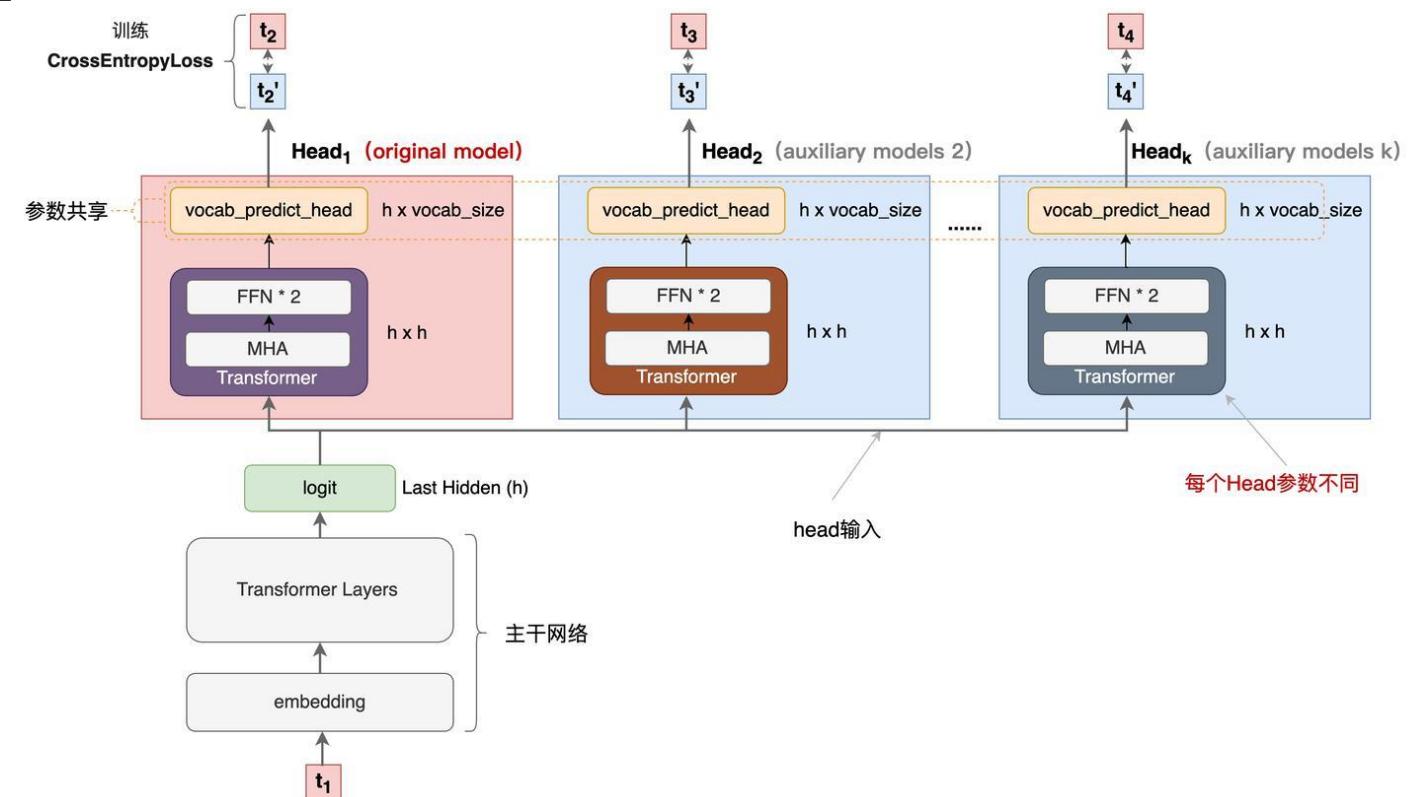
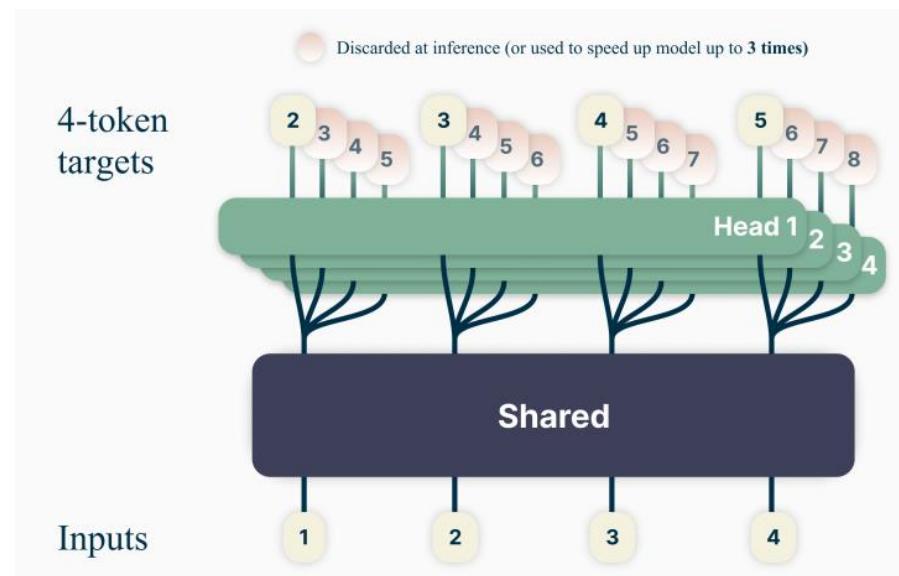
- MTP (Multi-Tokens Prediction) : Block-wise Parallel Decoding
  - Google 2018 NIPS



- 流水线推理：
  - 阶段一：预测，生成 k 个 tokens (单步完成)
  - 阶段二：验证，每次将前  $i-1$  个生成的 tokens 和前缀序列拼接，发给 k 个 head，和下一周期的预测阶段流水线并行。
  - 阶段三：接受，将生成的 tokens 序列验证通过的最长的前缀作为结果接受。
- 进一步提高计算效率

# 技术详解

- MTP (Multi-Tokens Prediction) : Meta's MTP
  - Meta 2024.04 ICML
  - 改进: FFN 更换为 Transformer  
多个头并行计算 Loss

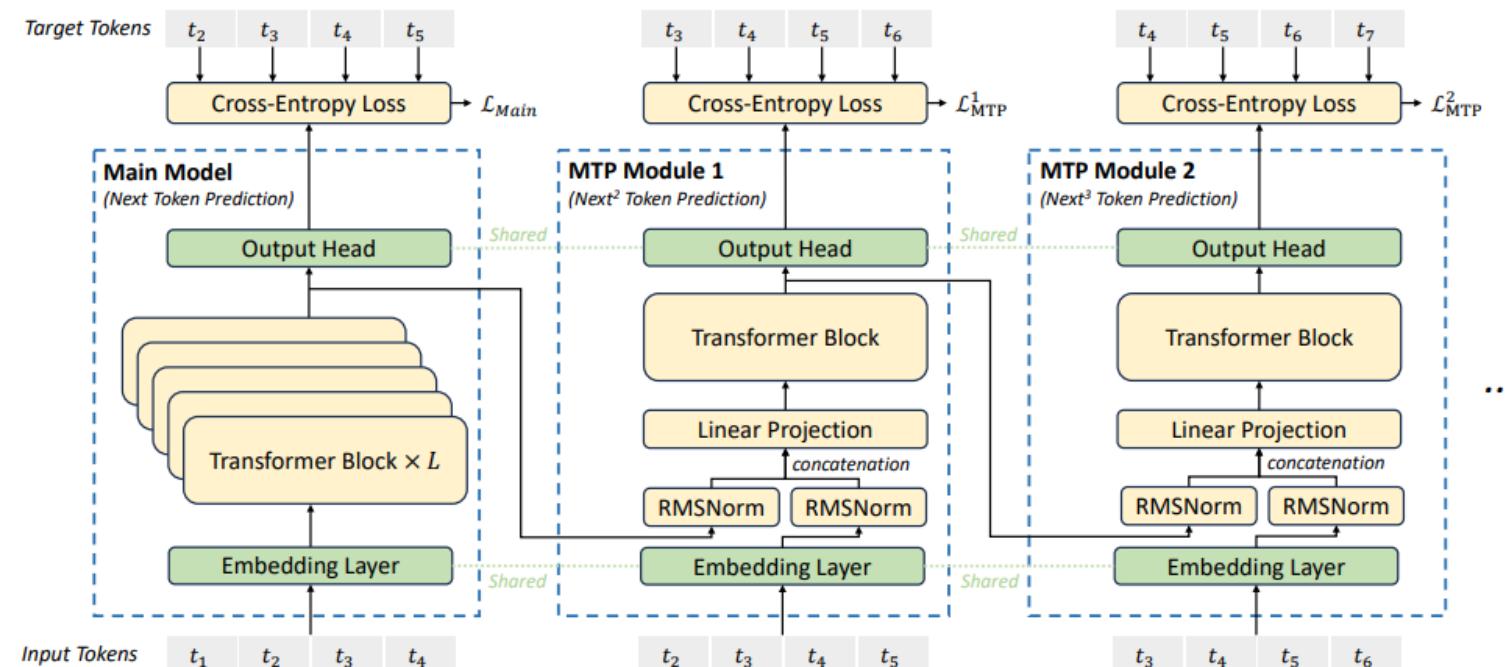


# 技术详解

## ➤ MTP (Multi-Tokens Prediction) : Deepseek MTP

### ➤ 主要改进：

- 级联模块：前一个 MTP 模块的输出会接入下一个 MTP 模块的输入
- 流水线化更完全：空间换时间



# 技术详解

## ➤ MTP (Multi-Tokens Prediction) : Deepseek MTP

### ➤ 级联模块:

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

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

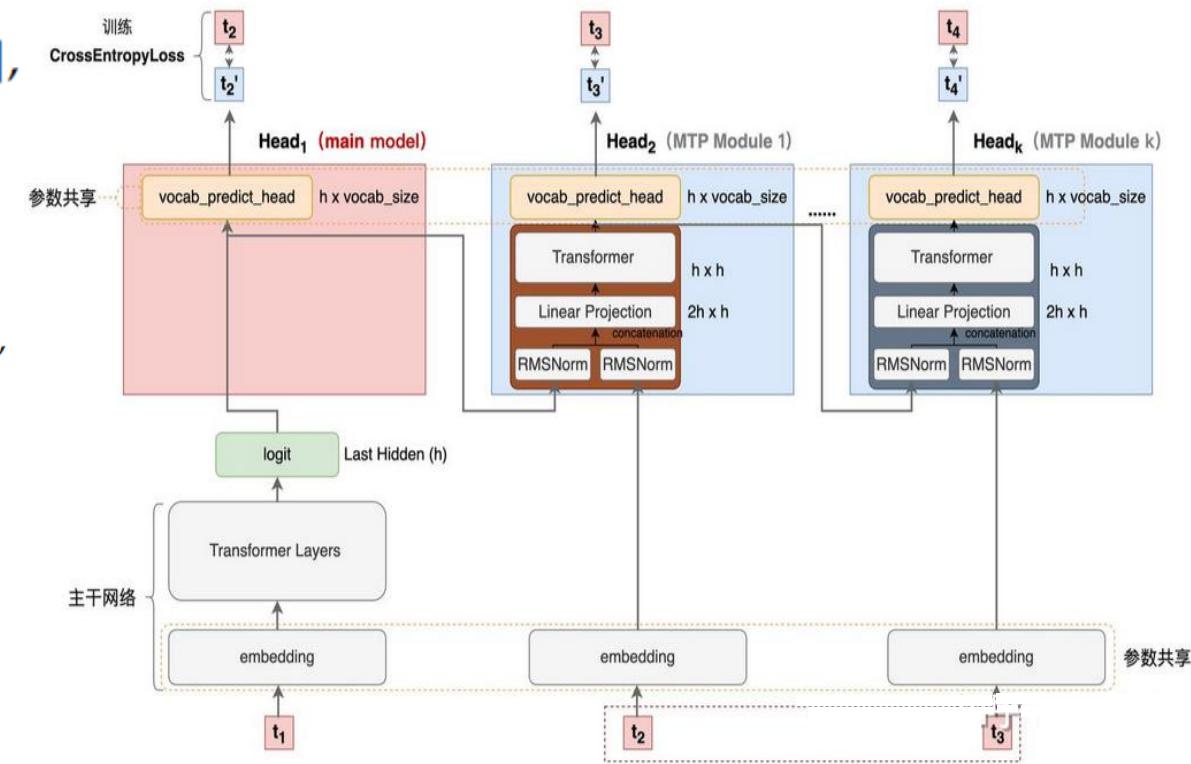
$$P_{i+k+1}^k = \text{OutHead}(\mathbf{h}_i^k).$$

➤ 目标函数:  $\mathcal{L}_{\text{MTP}}^k = \text{CrossEntropy}(P_{2+k:T+1}^k, t_{2+k:T+1}) = -\frac{1}{T} \sum_{i=2+k}^{T+1} \log P_i^k[t_i],$

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

### ➤ 推理:

- 只保留 Next Token Prediction 的 head (实验验证效果正常)
- 或 按照 background 中的三阶段投机解码

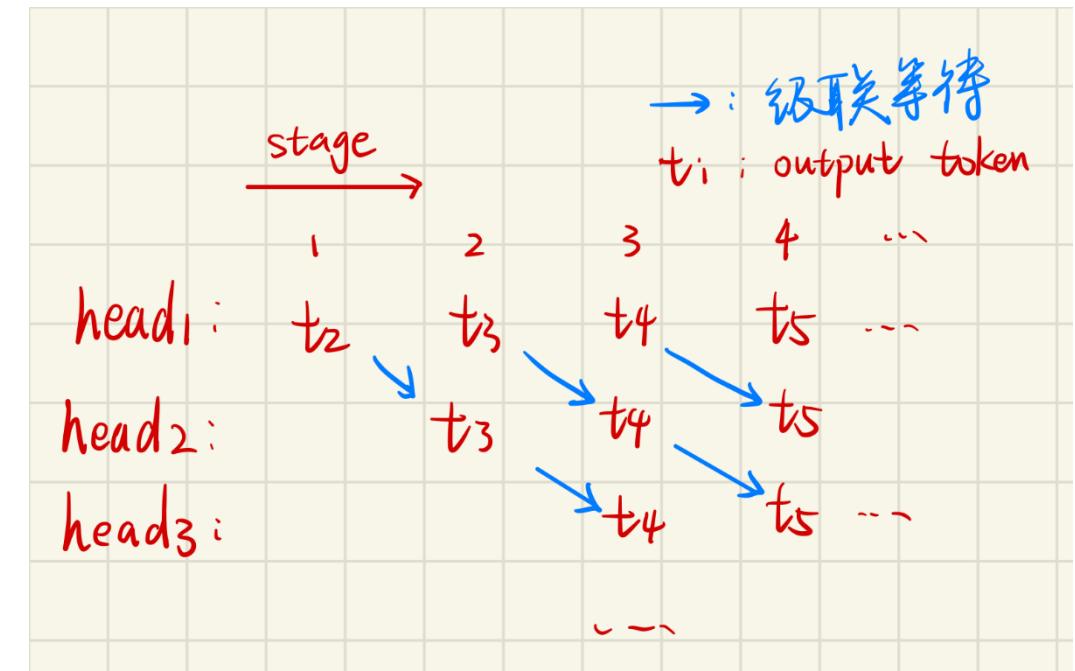


# 技术详解

## ➤ MTP (Multi-Tokens Prediction) : Deepseek MTP

### ➤ 流水线化:

- 假设每个 MTP 计算耗时相等，则计算流程如图
- 理想状态下是无冲突、无延迟的流水线
- 并且注意到每个时刻会产生多组输出，恰好与 GRPO 的组关联相对应
- 计算效率和数据利用率大幅度提高！



# 技术详解

## ➤ 强化学习 和 Self-Rewarding, 其他

- 强化学习：完全依赖 Model-based Reward 容易遇到 Reward Hacking  
(偷懒获取虚高的 reward, 比如训练跑步, 机器人发现原地抖动也可以获取 Reward, 于是策略变成原地抖动)
  - 对于有客观、确定性的好坏判断标准的数据, 用 Rule-based Reward Model
  - 对于主观性强、开放域数据, 则使用 Model-based Reward Model
- Self-Rewarding: constitutional AI, 通过自我投票自己评估自己的回答
- 其他: 从 R1 蒸馏推理数据 (尤其是长 CoT) 用于训练。

# 技术详解

- Deepseek R1：AI 研究者的浪漫——独属于探索者的 Aha 时刻
  - 训练成本低却能匹敌 OpenAI o1 最新版本的国产最强推理模型
  - 除了常见的推理任务，尤其震撼的是具有极强的创造力
  - 自从 2022 年末 DALL·E 和 ChatGPT 以来，第二次爆火出圈的大模型
    - 在国外，低成本高效率引发硅谷震动
    - 在国内，霸占热搜，成为几乎人尽皆知的现象级产品
    - 推出后立刻获得各界广泛关注，甚至访问和下载 Deepseek 在美国某些州立刻被立法确定为犯罪
    - 拥有幻方财力之支持的 Deepseek 在巨大的访问压力和主要来自美国的 DDoS 攻击下，R1 web service 出于半瘫痪状态，API 则被 block。Github issue 存在大量疑似机器人/水军用户恶意攻击
    - 具有严格的审查机制（包含本地运行时审查）
    - 多轮 alignment 下仍然被发现存在数据污染（OpenAI 审查规则）
  - 完全开源、允许商用，产品协议明确可“模型蒸馏”
    - 回顾 Deepseek LLM：source configurations, 7B and 67B. Guided by the scaling laws, we introduce DeepSeek LLM, a project dedicated to advancing open-source language models with a long-term perspective.  
(伟大，无需多言)
    - Github 已有开源复现项目，当前约 1k star (USTC 大四学生 MSRA intern, Deepseek V3 contributer) , star 者中不乏业界人士、著名学者 <https://github.com/Unakar/Logic-RL>

# 技术详解

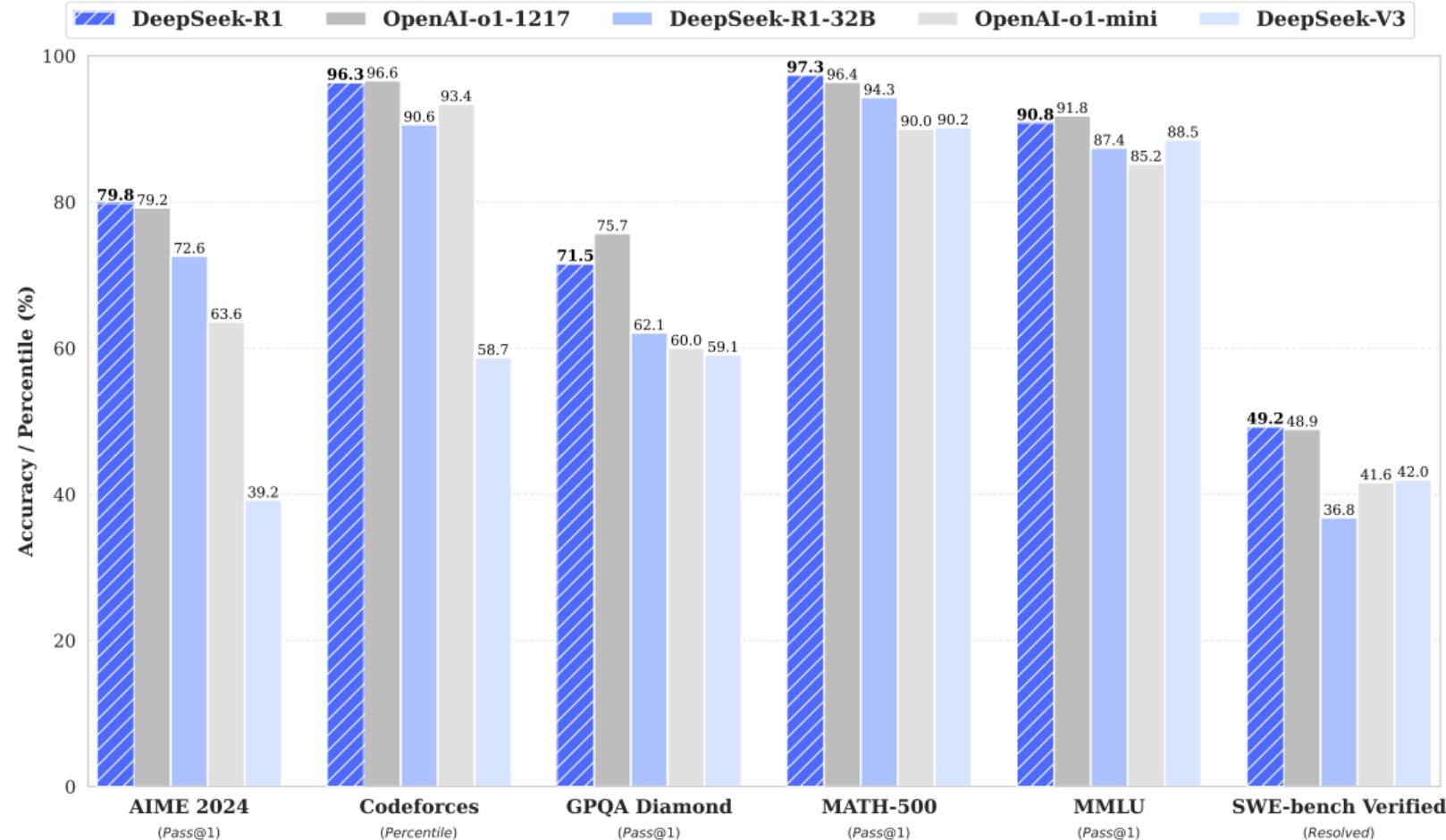
- Deepseek R1：AI 研究者的浪漫——独属于探索者的 Aha 时刻
  - R1-Zero：基于 DeepSeek-V3-Base，无 SFT，纯粹的大规模 GRPO
    - Accuracy rewards 和 Format rewards：利用 Reward 来修正模型行为
    - 不使用神经网络建模 Reward：大规模 RL 中容易发生 Reward Hacking
    - 大规模 RL 时自然发生进一步“涌现”
    - 短板：可读性（语言混合和输出格式）

# 技术详解

- Deepseek R1：AI 研究者的浪漫——独属于探索者的 Aha 时刻
  - R1：基于 DeepSeek-V3-Base，四阶段微调
    - cold start SFT：小规模长 CoT 推理数据 SFT，缓和可读性问题，提升后续 RL 效果
    - 推理导向的强化学习：引入语言一致性 Reward，在推理密集型任务上强化学习
    - 拒绝采样和SFT：优化开放域和创造性任务表现
      - 推理数据：DeepSeek-V3 as judge，清洗数据，从多个回复中选择正确的那些（拒绝采样）
      - 非推理数据：复用 DeepSeek-V3 流程，部分任务使用 DeepSeek-V3 生成 CoT。
    - 全场景强化学习：提高模型帮助性和安全性，不同数据不同方法
      - 推理数据：复用 R1-Zero 流程，Rule-based Reward Model
      - 通用数据：复用 Deepseek V3 流程，Model-based Reward Model
      - 帮助性：关注模型最后的总结，评估有效性和相关性
      - 安全性：关注推理过程和最后的总结，判断是否有潜在风险、偏见、有害信息等
  - 从 R1 蒸馏到小模型：只做 SFT 不做 RL，蒸馏有效提升小模型推理能力
  - 失败的探索：
    - PRM，中间步骤标注困难、奖励劫持 Reward Hacking；MCTS，搜索空间爆炸、值模型训练困难

# 技术详解

- Deepseek R1：AI 研究者的浪漫——独属于探索者的 Aha 时刻
- 效果



# 技术详解

- Deepseek R1：AI 研究者的浪漫——独属于探索者的 Aha 时刻
- 效果

Benchmark (Metric)		Claude-3.5-Sonnet-1022	GPT-4o 0513	DeepSeek V3	OpenAI o1-mini	OpenAI o1-1217	DeepSeek R1
English	Architecture	-	-	MoE	-	-	MoE
	# Activated Params	-	-	37B	-	-	37B
	# Total Params	-	-	671B	-	-	671B
	MMLU (Pass@1)	88.3	87.2	88.5	85.2	<b>91.8</b>	90.8
	MMLU-Redux (EM)	88.9	88.0	89.1	86.7	-	<b>92.9</b>
	MMLU-Pro (EM)	78.0	72.6	75.9	80.3	-	<b>84.0</b>
	DROP (3-shot F1)	88.3	83.7	91.6	83.9	90.2	<b>92.2</b>
	IF-Eval (Prompt Strict)	<b>86.5</b>	84.3	86.1	84.8	-	83.3
	GPQA Diamond (Pass@1)	65.0	49.9	59.1	60.0	<b>75.7</b>	71.5
	SimpleQA (Correct)	28.4	38.2	24.9	7.0	<b>47.0</b>	30.1
Code	FRAMES (Acc.)	72.5	80.5	73.3	76.9	-	<b>82.5</b>
	AlpacaEval2.0 (LC-winrate)	52.0	51.1	70.0	57.8	-	<b>87.6</b>
	ArenaHard (GPT-4-1106)	85.2	80.4	85.5	92.0	-	<b>92.3</b>
	LiveCodeBench (Pass@1-COT)	38.9	32.9	36.2	53.8	63.4	<b>65.9</b>
	Codeforces (Percentile)	20.3	23.6	58.7	93.4	<b>96.6</b>	96.3
Math	Codeforces (Rating)	717	759	1134	1820	<b>2061</b>	2029
	SWE Verified (Resolved)	<b>50.8</b>	38.8	42.0	41.6	48.9	49.2
	Aider-Polyglot (Acc.)	45.3	16.0	49.6	32.9	<b>61.7</b>	53.3
	AIME 2024 (Pass@1)	16.0	9.3	39.2	63.6	79.2	<b>79.8</b>
Chinese	MATH-500 (Pass@1)	78.3	74.6	90.2	90.0	96.4	<b>97.3</b>
	CNMO 2024 (Pass@1)	13.1	10.8	43.2	67.6	-	<b>78.8</b>
	CLUEWSC (EM)	85.4	87.9	90.9	89.9	-	<b>92.8</b>
	C-Eval (EM)	76.7	76.0	86.5	68.9	-	<b>91.8</b>
	C-SimpleQA (Correct)	55.4	58.7	<b>68.0</b>	40.3	-	63.7

# Outline

1

概览

2

技术详解

3

分析

4

总结

# Outline

1

概览

2

技术详解

3

分析

4

总结