DeepSeek-V3.2: 推动开源大语言模型的前沿

DeepSeek-AI

research@deepseek.com

Abstract

我们推出了 DeepSeek-V3.2，该模型将高计算效率与卓越的推理及智能体性能完美融合。DeepSeek-V3.2 的关键技术突破如下：**(1) DeepSeek 稀疏注意力机制 (DSA)**：我们提出了 DSA，这是一种高效的注意力机制，能够在长上下文场景中保持模型性能的同时，大幅降低计算复杂度。**(2) 可扩展的强化学习框架**：通过实施稳健的强化学习流程并扩展后训练算力，DeepSeek-V3.2 的表现与 GPT-5 相当。值得注意的是，我们的高算力变体 DeepSeek-V3.2-Speciale 超越了 GPT-5，其推理能力与 Gemini-3.0-Pro 持平，并在 2025 年国际数学奥林匹克竞赛（IMO）和国际信息学奥林匹克竞赛（IOI）中均取得了 金牌 级别的成绩。**(3) 大规模智能体任务合成流水线**：为了将推理能力融入工具使用场景，我们开发了一种新颖的合成流水线，能够系统化地大规模生成训练数据。该方法促进了可扩展的智能体后训练，在复杂交互环境中显著提升了模型的泛化能力与指令遵循的鲁棒性。

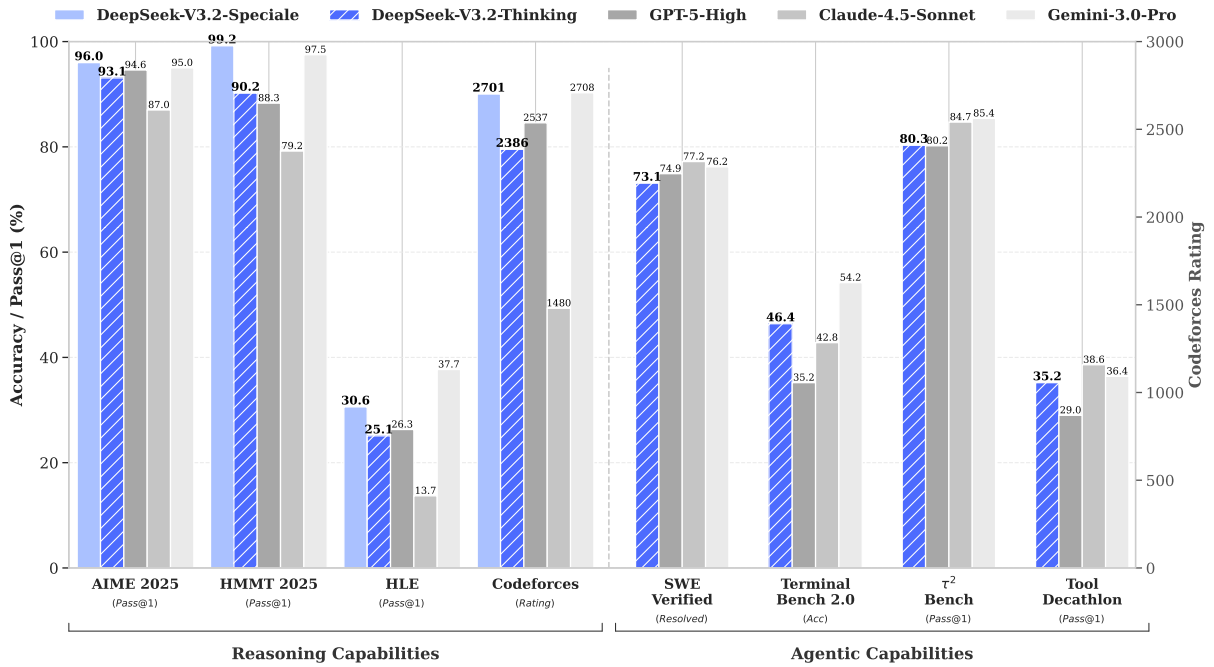


图 1 | DeepSeek-V3.2 及其对比模型的基准测试。对于 HMMT 2025，我们报告的是二月比赛的结果，与基线保持一致。对于 HLE，我们报告的是纯文本子集的结果。

1. Introduction

推理模型 (**??**) 的发布标志着大语言模型（LLM）演进过程中的一个关键节点，推动了可验证领域整体性能的显著飞跃。自这一里程碑以来，LLM 的能力得到了快速提升。然而，在过去几个月里，明显的分化趋势已经显现。尽管开源社区 (**????**) 仍在不断取得进展，但闭源专有模型 (**???**) 的性能提升轨迹却以显著更快的速度加速。因此，闭源与开源模型之间的性能差距非但没有缩小，反而似乎正在扩大，专有系统在复杂任务中展现出日益卓越的能力。

通过分析，我们识别出限制开源模型在复杂任务中表现的三个关键缺陷。首先，在架构层面，主要依赖原始注意力（vanilla attention）机制 (**?**) 严重制约了长序列的处理效率。这种低效性为可扩展部署和有效的后训练构成了重大障碍。其次，在资源分配方面，开源模型在后训练阶段的计算投入不足，限制了其在困难任务上的表现。最后，在智能体（AI agents）场景下，开源模型在泛化能力和指令遵循方面明显落后于其专有模型对应物 (**???**)，阻碍了其在实际部署中的有效性。

为了解决这些关键限制，我们首先引入了 DSA，这是一种旨在大幅降低计算复杂度的高效注意力机制。该架构有效突破了效率瓶颈，即使在长上下文场景中也能保持模型性能。其次，我们开发了一种稳定且可扩展的强化学习（RL）协议，允许在后训练阶段进行大幅度的计算扩展。值得注意的是，该框架分配了超过预训练成本 10% 的后训练计算预算，从而解锁了高级能力。第三，我们提出了一种新颖的流水线，以促进工具使用场景中的可泛化推理能力。首先，我们利用 DeepSeek-V3 (**?**) 的方法实施冷启动阶段，将推理与工具使用统一在单条轨迹中。随后，我们推进至大规模智能体任务合成阶段，生成了超过 1,800 个不同的环境和 85,000 个复杂提示。这些海量的合成数据驱动了 RL 过程，显著提升了模型在智能体场景下的泛化能力和指令遵循能力。

在多个推理基准测试中，DeepSeek-V3.2 取得了与 Kimi-k2-thinking 和 GPT-5 相当的性能。此外，DeepSeek-V3.2 显著提升了开源模型的智能体能力，在 **???** 引入的长尾智能体任务中表现出卓越的熟练度。DeepSeek-V3.2 成为智能体场景中极具成本效益的替代方案，在大幅降低成本的同时，显著缩小了开源模型与前沿专有模型之间的性能差距。值得注意的是，为了推动开源模型在推理领域的边界，我们放宽了长度限制以开发 DeepSeek-V3.2-Speciale。因此，DeepSeek-V3.2-Speciale 的性能达到了与领先的闭源系统 Gemini-3.0-Pro (**?**) 持平的水平。它在 2025 年国际信息学奥林匹克竞赛（IOI）、2025 年国际大学生程序设计竞赛（ICPC）世界总决赛、2025 年国际数学奥林匹克竞赛（IMO）以及 2025 年中国数学奥林匹克竞赛（CMO）中均展现了金牌级别的表现。

2. DeepSeek-V3.2 架构

2.1. DeepSeek 稀疏注意力

DeepSeek-V3.2 采用了与 DeepSeek-V3.2-Exp 完全相同的架构。与 DeepSeek-V3.1 的最后一个版本 DeepSeek-V3.1-Terminus 相比，DeepSeek-V3.2 唯一的架构改动是通过持续训练引入了

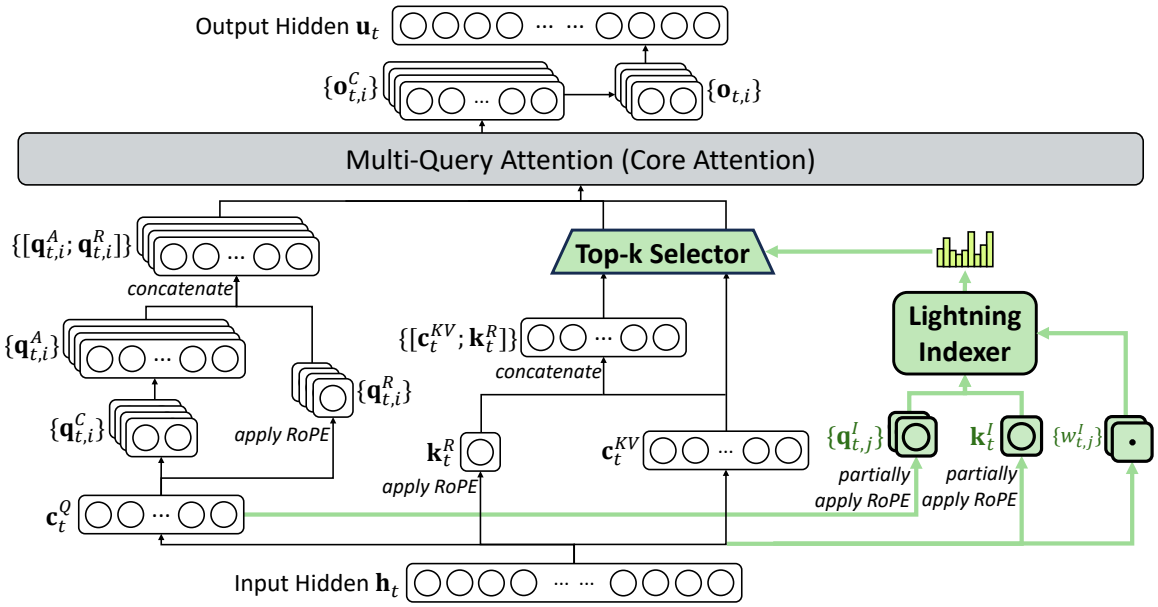


图 2 | DeepSeek-V3.2 的注意力架构，其中 DSA 在 MLA 下实例化。绿色部分展示了 DSA 如何根据索引器选择 Top-k 键值对条目。

DeepSeek Sparse Attention (DSA)。

DSA 原型。 DSA 的原型主要由两个组件构成：闪电索引器（lightning indexer）和细粒度 Token 选择机制。

闪电索引器计算查询 token $\mathbf{h}_t \in \mathbb{R}^d$ 与前序 token $\mathbf{h}_s \in \mathbb{R}^d$ 之间的索引分数 $I_{t,s}$，从而确定查询 token 应选择哪些 token：

$$I_{t,s} = \sum_{j=1}^{H^I} w^I_{t,j} \cdot \mathrm{ReLU}\left(\mathbf{q}^I_{t,j} \cdot \mathbf{k}^I_s\right), \tag{1}$$

其中 H^I 表示索引器头数；$\mathbf{q}^I_{t,j} \in \mathbb{R}^{d^I}$ 和 $w^I_{t,j} \in \mathbb{R}$ 由查询 token $\mathbf{h}_t$ 导出；$\mathbf{k}^I_s \in \mathbb{R}^{d^I}$ 由前序 token $\mathbf{h}_s$ 导出。出于吞吐量考虑，我们选择 ReLU 作为激活函数。鉴于闪电索引器头数较少且可基于 FP8 实现，其计算效率十分显著。

对于每个查询 token $\mathbf{h}_t$ 的索引分数 $\{I_{t,s}\}$，我们的 **细粒度 token 选择机制**仅检索与 Top-k 索引分数对应的键值对条目 $\{\mathbf{c}_s\}$。随后，通过在查询 token $\mathbf{h}_t$ 与稀疏选择的键值对条目 $\{\mathbf{c}_s\}$ 之间应用注意力机制，计算注意力输出 $\mathbf{u}_t$：

$$\mathbf{u}_t = \mathrm{Attn}\left(\mathbf{h}_t, \left\{\mathbf{c}_s \,\middle|\, I_{t,s} \in \text{Top-k}\left(I_{t,:}\right)\right\}\right). \tag{2}$$

在 MLA 下实例化 DSA。 出于从 DeepSeek-V3.1-Terminus 进行持续训练的考虑，我们为 DeepSeek-V3.2 基于 MLA (**?**) 实例化了 DSA。在算子层面，出于计算效率的考虑，每个键值对

条目必须在多个查询之间共享 (?)。因此，我们基于 MLA 的 MQA (?) 模式实现了 DSA[1]，在该模式下，每个潜在向量（即 MLA 的键值对条目）将在查询 token 的所有查询头之间共享。基于 MLA 的 DSA 架构如图 2 所示。我们还提供了 DeepSeek-V3.2 的开源实现[2]，以明确说明相关细节。

2.1.1. 持续预训练

我们从上下文长度已扩展至 128K 的 DeepSeek-V3.1-Terminus 基础检查点出发，执行持续预训练，随后进行后训练，从而构建出 DeepSeek-V3.2。

DeepSeek-V3.2 的持续预训练包含两个训练阶段。在这两个阶段中，训练数据的分布与用于 DeepSeek-V3.1-Terminus 的 128K 长上下文扩展数据完全一致。

密集预热阶段。 我们首先使用一个短暂的预热阶段来初始化闪电索引器。在此阶段，我们保留密集注意力，并冻结除闪电索引器外的所有模型参数。为了使索引器输出与主注意力分布对齐，对于第 t 个查询 token，我们首先通过沿所有注意力头求和来聚合主注意力分数。随后，该和沿序列维度进行 L1 归一化，以生成目标分布 $p_{t,:} \in \mathbb{R}^t$。基于 $p_{t,:}$，我们设定 KL 散度损失作为索引器的训练目标：

$$\mathcal{L}^I = \sum_t \mathbb{D}_{\mathrm{KL}}\big(p_{t,:} \,\big\|\, \mathrm{Softmax}(I_{t,:})\big). \tag{3}$$

在预热阶段，我们使用 10^{-3} 的学习率。我们仅训练索引器 1000 步，每步包含 16 条长度为 128K token 的序列，总计 21 亿个 token。

稀疏训练阶段。 在索引器预热之后，我们引入细粒度 token 选择机制，并优化所有模型参数，以使模型适应 DSA 的稀疏模式。在此阶段，我们同样保持将索引器输出与主注意力分布对齐，但仅考虑选定的 token 集合 $\mathcal{S}_t = \{s \,|\, I_{t,s} \in \text{Top-k}(I_{t,:})\}$：

$$\mathcal{L}^I = \sum_t \mathbb{D}_{\mathrm{KL}}\big(p_{t,\mathcal{S}_t} \,\big\|\, \mathrm{Softmax}(I_{t,\mathcal{S}_t})\big). \tag{4}$$

值得注意的是，我们将索引器输入从计算图中分离，以便进行独立优化。索引器的训练信号仅来自 $\mathcal{L}^I$，而主模型的优化仅依据语言建模损失。在此稀疏训练阶段，我们使用 7.3×10^{-6} 的学习率，并为每个查询 token 选择 2048 个键值 token。我们同时训练主模型和索引器共 15000 步，每步包含 480 条长度为 128K token 的序列，总计 9437 亿个 token。

2.2. 等效评估

标准基准测试 2025 年 9 月，我们在一系列侧重于多样化能力的基准测试上评估了 DeepSeek-V3.2-Exp，并将其与性能相近的 DeepSeek-V3.1-Terminus 进行了对比。尽管 DeepSeek V3.2 Exp

[1]我们在附录 A 中说明了 MLA 的 MQA 与 MHA 模式之间的差异。

[2]https://huggingface.co/deepseek-ai/DeepSeek-V3.2-Exp/tree/main/inference

在长序列上的计算效率显著提升，但在短上下文和长上下文任务中，我们均未观察到其性能相比 DeepSeek-V3.1-Terminus 有显著下降。

人类偏好 鉴于直接的人类偏好评估本质上容易受到偏差影响，我们采用 ChatbotArena 作为间接评估框架，以近似评估用户对新开发基座模型的偏好。DeepSeek-V3.1-Terminus 与 DeepSeek-V3.2-Exp 采用了相同的后训练策略，两者在 2025 年 11 月 10 日进行的评估中获得的 Elo 分数非常接近。这些结果表明，尽管引入了稀疏注意力机制，新基座模型的性能仍与上一代模型持平。

长上下文评估 在 DeepSeek-V3.2-Exp 发布后，多个独立机构使用此前未见的测试集进行了长上下文评估。其中一项代表性基准是 AA-LCR[3]，在该基准的推理模式下，DeepSeek-V3.2-Exp 的得分比 DeepSeek-V3.1-Terminus 高出四分。在 Fiction.liveBench 评估[4]中，DeepSeek-V3.2-Exp 在多项指标上均持续优于 DeepSeek-V3.1-Terminus 。这些证据表明，DeepSeek-V3.2-Exp 的基座检查点在长上下文任务上并未出现性能退化。

2.3. 推理成本

DSA 将主模型的核心注意力复杂度从 $O(L^2)$ 降低至 $O(Lk)$，其中 k ($\ll L$) 为选中的 token 数量。尽管 lightning 索引器的复杂度仍为 $O(L^2)$，但与 DeepSeek-V3.1-Terminus 中的 MLA 相比，其所需计算量大幅减少。结合我们的优化实现，DSA 在长上下文场景下实现了显著的端到端加速。图 3 展示了 DeepSeek-V3.1-Terminus 和 DeepSeek-V3.2 的 token 成本随序列中 token 位置的变化情况。这些成本是基于部署在 H800 GPU 上的实际服务进行基准测试估算得出的，GPU 租赁价格为每小时 2 美元。需要注意的是，针对短序列的预填充阶段，我们专门实现了一种掩码多头注意力（masked MHA）模式来模拟 DSA，该模式在短上下文条件下能够实现更高的效率。

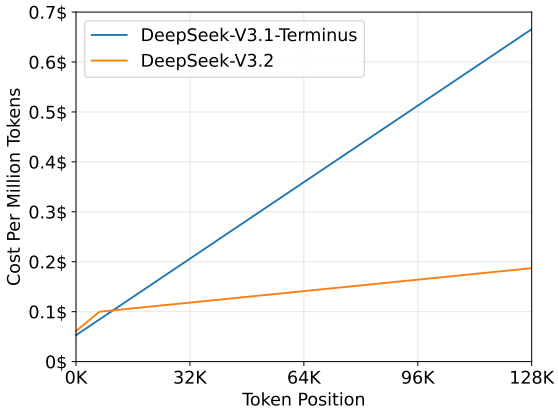
3. 后训练

在完成持续预训练后，我们进行后训练以构建最终的 DeepSeek-V3.2。DeepSeek-V3.2 的后训练同样采用了与稀疏持续预训练阶段相同的稀疏注意力机制。对于 DeepSeek-V3.2，我们沿用了与 DeepSeek-V3.2-Exp 相同的后训练流程，其中包括专家蒸馏和混合强化学习（RL）训练。

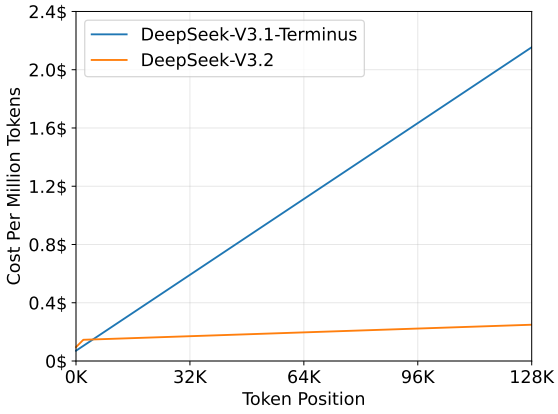
专家蒸馏 针对每项任务，我们最初会开发一个专门用于该特定领域的专家模型，所有专家模型均从相同的预训练 DeepSeek-V3.2 基础检查点微调而来。除写作任务和通用问答外，我们的框架还涵盖六个专业领域：数学、编程、通用逻辑推理、通用智能体任务、智能体编程和智能体搜索，且所有领域均支持思考模式与非思考模式。每个专家模型均通过大规模强化学习（RL）算力进行训练。此外，我们采用不同的模型分别生成长链式推理（思考模式）和直接响应生成（非思考模式）的训练数据。专家模型准备就绪后，将用于生成最终检查点所需的领域特定数据。实

[3] https://artificialanalysis.ai/evaluations/artificial-analysis-long-context-reasoning

[4] https://fiction.live/stories/Fiction-liveBench-April-6-2025/oQdzQvKHw8JyXbN87



(a) 预填充



(b) 解码

图 3 | DeepSeek-V3.1-Terminus 与 DeepSeek-V3.2 在 H800 集群上的推理成本。

验结果表明，在蒸馏数据上训练的模型性能仅略低于领域专家模型，且通过后续的 RL 训练可有效消除这一性能差距。

混合 RL 训练 对于 DeepSeek-V3.2，我们仍采用组相对策略优化（GRPO）（**??**）作为 RL 训练算法。与 DeepSeek-V3.2-Exp 类似，我们将推理、智能体和对齐训练合并为单一的 RL 阶段。该方法在有效平衡不同领域性能的同时，避免了多阶段训练范式中常见的灾难性遗忘问题。对于推理和智能体任务，我们采用基于规则的结局奖励、长度惩罚和语言一致性奖励。对于通用任务，我们采用生成式奖励模型，其中每个提示词（prompt）均有其专属的评估标准。

DeepSeek-V3.2 与 DeepSeek-V3.2-Speciale DeepSeek-V3.2 整合了从专家模型蒸馏而来的推理、智能体及人类对齐数据，并经过数千步的持续 RL 训练以达到最终检查点。为探索扩展思考的潜力，我们还开发了一个实验性变体 DeepSeek-V3.2-Speciale。该模型仅在推理数据上进行训练，并在 RL 阶段降低了长度惩罚。此外，我们引入了 DeepSeekMath-V2 (**?**) 的数据集与奖励方法，以增强数学证明能力。

我们将在第 3.1 节重点介绍如何构建稳定的方案以扩展 RL 算力，并在第 3.2 节探讨如何将思考能力整合到智能体任务中。

3.1. 扩展 GRPO

我们首先回顾 GRPO 的目标函数。GRPO 通过最大化以下目标函数来优化策略模型 π_θ，该目标函数基于针对每个问题 q 从旧策略 π_{old} 中采样的一组回复 $\{o_1, \cdots, o_G\}$ 计算得出：

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{q\sim P(Q),\{o_i\}_{i=1}^{G}\sim\pi_{\text{old}}(\cdot|q)}\Bigg[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|} \min\left(r_{i,t}(\theta)\hat{A}_{i,t}, \text{clip}\left(r_{i,t}(\theta), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_{i,t}\right) - \beta\mathbb{D}_{\text{KL}}\left(\pi_\theta(o_{i,t})\,\|\,\pi_{\text{ref}}(o_{i,t})\right)\Bigg], \tag{5}$$

其中

$$r_{i,t}(\theta) = \frac{\pi_\theta(o_{i,t}|q, o_{i,<t})}{\pi_{\text{old}}(o_{i,t}|q, o_{i,<t})} \tag{6}$$

是当前策略与旧策略之间的重要性采样比率。ε 和 β 分别为控制裁剪范围和 KL 惩罚强度的超参数。$\hat{A}_{i,t}$ 是 $o_{i,t}$ 的优势值，通过对每组内的结局奖励进行归一化来估计。具体而言，使用一组奖励模型为组内每个输出 o_i 打分，得到结局奖励 R_i，从而分别获得 G 个奖励 $\boldsymbol{R} = \{R_1, \cdots, R_G\}$。$o_{i,t}$ 的优势值通过从输出 o_i 的奖励中减去该组的平均奖励来计算，即 $\hat{A}_{i,t} = R_i - \text{mean}(\boldsymbol{R})$。

接下来，我们将概述直接基于 GRPO 算法构建的、用于稳定 RL 扩展的额外策略。

无偏 KL 估计 鉴于 $o_{i,t}$ 是从旧策略 $\pi_{\text{old}}(\cdot|q, o_{i,<t})$ 中采样的，我们修正了 K3 估计器 (**?**)，利用当前策略 π_θ 与旧策略 π_{old} 之间的重要性采样比率来获得无偏的 KL 估计：

$$\mathbb{D}_{\text{KL}}\left(\pi_\theta(o_{i,t})\,\|\,\pi_{\text{ref}}(o_{i,t})\right) = \frac{\pi_\theta(o_{i,t}|q, o_{i,<t})}{\pi_{\text{old}}(o_{i,t}|q, o_{i,<t})}\left(\frac{\pi_{\text{ref}}(o_{i,t}|q, o_{i,<t})}{\pi_\theta(o_{i,t}|q, o_{i,<t})} - \log\frac{\pi_{\text{ref}}(o_{i,t}|q, o_{i,<t})}{\pi_\theta(o_{i,t}|q, o_{i,<t})} - 1\right). \tag{7}$$

作为这一调整的直接结果，该 KL 估计器的梯度变得无偏，从而消除了系统性估计误差，有助于实现稳定收敛。这与原始 K3 估计器形成鲜明对比，尤其是在采样 token 在当前策略下的概率远低于参考策略时（即 $\pi_\theta \ll \pi_{\text{ref}}$）。在这种情况下，K3 估计器的梯度会赋予这些 token 不成比例且无界的巨大权重以最大化其似然，导致梯度更新充满噪声，这些噪声会累积并降低后续迭代中的样本质量，进而引发不稳定的训练动态。在实践中，我们发现不同领域从不同强度的 KL 正则化中受益。对于某些领域（如数学），应用相对较弱的 KL 惩罚甚至完全省略它都能带来性能提升。

离策略序列掩码 为提高 RL 系统的效率，我们通常生成大批量的 rollout 数据，随后将其划分为多个小批量（mini-batch）进行多次梯度更新。这种做法本质上会引入离策略（off-policy）行为。此外，用于高效数据生成的推理框架通常经过高度优化，其实现细节可能与训练框架存在差异。这种训练-推理不一致性进一步加剧了离策略程度。为稳定训练并提高对离策略更新的容忍

度，我们掩码那些导致显著策略分歧的负向序列，该分歧通过数据采样策略 π_{old} 与当前策略 π_θ 之间的 KL 散度来衡量。更具体地说，我们在 GRPO 损失中引入了一个二值掩码 M：

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{q\sim P(Q),\{o_i\}_{i=1}^{G}\sim\pi_{\text{old}}(\cdot|q)}\Bigg[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|} \min\left(r_{i,t}(\theta)\hat{A}_{i,t}, \text{clip}\left(r_{i,t}(\theta), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_{i,t}\right) M_{i,t} - \beta\mathbb{D}_{\text{KL}}\left(\pi_\theta(o_{i,t})\,\|\,\pi_{\text{ref}}(o_{i,t})\right)\Bigg], \quad (8)$$

其中

$$M_{i,t} = \begin{cases} 0 & \hat{A}_{i,t} < 0, \frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\log\frac{\pi_{\text{old}}(o_{i,t}|q,o_{i,<t})}{\pi_\theta(o_{i,t}|q,o_{i,<t})} > \delta \\ 1 & \text{otherwise,} \end{cases} \quad (9)$$

δ 是控制策略分歧阈值的超参数。需要注意的是，此处的 π_{old} 表示推理框架直接返回的采样概率，因此旧策略与当前策略之间的 KL 散度涵盖了上述两种离策略来源。同样值得注意的是，我们仅掩码具有负优势值的序列。

直观而言，模型从自身错误中学习获益最大，而高度离策略的负样本可能产生不利影响，甚至误导或破坏优化过程。经验观察表明，该离策略序列掩码操作在某些原本会表现出不稳定的训练场景中提升了稳定性。

保持路由 混合专家（MoE）模型通过在推理期间仅激活部分专家模块来提高计算效率。然而，推理与训练框架之间的差异，叠加策略更新的影响，可能导致即使对于相同输入，推理和训练过程中的专家路由也不一致。这种不一致性会引发活跃参数子空间的突变，从而破坏优化稳定性并加剧离策略问题。为缓解此问题，我们保留了推理框架采样期间使用的专家路由路径，并在训练期间强制执行相同的路由路径，确保优化相同的专家参数。我们发现，该保持路由（Keep Routing）操作对 MoE 模型的 RL 训练稳定性至关重要，自 DeepSeek-V3-0324 起已纳入我们的 RL 训练流程。

保持采样掩码 Top-p 和 top-k 采样是广泛用于提升大语言模型（LLM）生成响应质量的采样策略。在强化学习（RL）训练中采用这些策略同样具有优势，因为它能避免采样到概率极低的 token 并将其作为优化目标。尽管这种截断操作保留了样本质量，但它引入了 π_{old} 与 π_θ 动作空间之间的不匹配，这违反了重要性采样的原则并导致训练不稳定。为解决此问题，我们在从 π_{old} 采样时保留截断掩码，并在训练期间将其应用于 π_θ，确保两种策略共享相同的动作子空间。经验表明，将 top-p 采样与保持采样掩码策略相结合，能有效在 RL 训练期间保持语言一致性。

3.2. 工具使用中的思考

3.2.1. 思考上下文管理

DeepSeek-R1 已证明，引入思考过程可以显著提升模型解决复杂问题的能力。基于这一洞察，我们旨在将思考能力整合到工具调用场景中。

我们观察到，复制 DeepSeek-R1 的策略——即在收到第二轮消息时丢弃推理内容——会导致严重的 token 效率低下。这种方法迫使模型在每次后续工具调用时冗余地重新推理整个问题。为缓解此问题，我们开发了一种专为工具调用场景定制的上下文管理机制，如图 4 所示：

- 仅当对话中引入新的 **用户消息**时，才会丢弃历史推理内容。如果仅追加与工具相关的消息（例如工具输出），推理内容将在整个交互过程中 **保留**。
- 当移除推理轨迹时，**工具调用及其结果**的历史记录仍保留在上下文中。

值得注意的是，某些智能体框架（如 Roo Code 或 Terminus）通过用户消息模拟工具交互。由于上述上下文管理规则，这些框架可能无法充分受益于我们增强的推理持久性机制。因此，我们建议在使用此类架构时采用非思考模型以获得最佳性能。

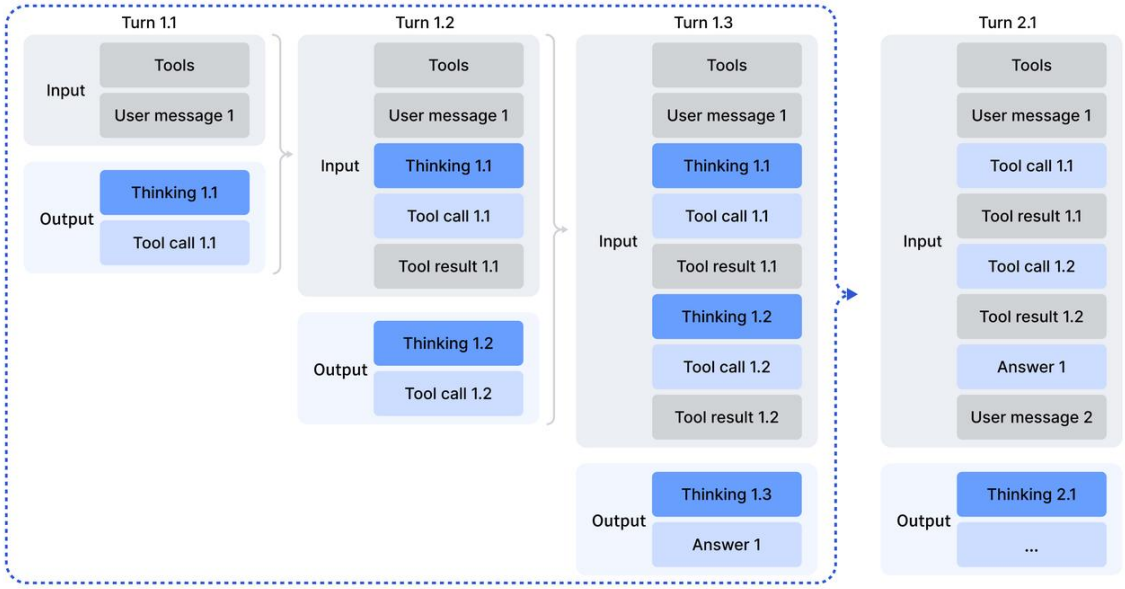


图 4 | 工具调用场景中的思考保留机制。

3.2.2. 冷启动

鉴于推理数据（非智能体）和非推理智能体数据的可用性，整合这两种能力的直接策略是通过精心设计的提示词。我们认为模型具备足够的能力准确遵循明确指令，从而能够在推理过程中无缝融入工具执行。

为展示冷启动机制的运行方式，我们选择性采样了训练数据，如附录表 5–7 所示。需要特

别注意的是，不同的任务提示词对应不同的系统提示词。表 5–7 展示了对应编程竞赛提示词的示例。表 5 展示了我们推理数据的一个示例，该系统提示词明确要求模型在给出最终答案前进行推理，并使用特殊标签 <think></think> 标记推理路径。表 6 展示了非推理智能体数据的提示词，其中系统提示词包含工具调用的指导。表 7 展示了我们设计的系统提示词，用于指示模型在其推理过程中融入多次工具调用。

通过这种方式，尽管工具使用模式中的推理可能缺乏鲁棒性，但模型偶尔仍能生成期望的轨迹，从而为后续的强化学习阶段奠定基础。

3.2.3. 大规模智能体任务

搜索智能体 我们采用基于 DeepSeek-V3.2 的多智能体流水线来生成多样化、高质量的训练数据。我们首先从大规模网络语料库中跨多个领域采样具有信息量的长尾实体。随后，一个问答构建智能体使用具有可配置深度和广度参数的搜索工具探索每个实体，并将发现的信息整合为问答对。多个具有异构配置（不同检查点、系统提示词等）的答案生成智能体为每个提出的问答对生成多样化的候选回复。一个具备搜索能力的验证智能体通过多轮验证所有答案，仅保留标准答案正确且所有候选答案均可验证为错误的样本。这些数据涵盖多种语言、领域和难度级别。为了补充这些可验证样本并更好地反映现实世界的使用情况，我们还从现有的有益性 RL 数据集中筛选实例进行数据增强，这些实例在使用搜索工具时能获得可衡量的收益。随后，我们制定了涵盖多个质量维度的详细评估标准，并采用生成式奖励模型根据这些标准对回复进行评分。这种混合方法使得模型能够同时优化事实可靠性和实际实用性。

代码智能体 我们通过从 GitHub 挖掘数百万个 Issue-Pull Request (PR) 对，构建了用于软件问题修复的大规模可执行环境。该数据集经过启发式规则和基于大语言模型（LLM）的判断进行严格过滤，以确保高质量，要求每条记录包含合理的问题描述、相关的黄金补丁（gold patch）以及用于验证的测试补丁。我们采用由 DeepSeek-V3.2 驱动的自动化环境配置智能体来为这些对构建可执行环境。该智能体负责包安装、依赖解析和测试执行。测试结果以标准的 JUnit 格式输出，确保跨编程语言和测试框架的一致性解析。仅当应用黄金补丁后，失败转通过（false-to-positive, F2P）测试用例数量非零（表明问题已修复）且通过转失败（pass-to-fail, P2F）测试用例数量为零（表明无回归）时，才认为环境构建成功。利用该流水线，我们成功构建了数万个可复现的问题修复环境，涵盖 Python、Java、JavaScript、TypeScript、C、C++、Go 和 PHP 等多种编程语言。

代码解释器智能体 我们利用 Jupyter Notebook 作为代码解释器来解决复杂的推理任务。为此，我们精心策划了一组多样化的问题，涵盖数学、逻辑和数据科学领域，每个问题都需要模型利用代码执行能力来得出解决方案。

通用智能体 为了在强化学习中扩展智能体环境和任务规模，我们采用了一个自动环境合成智能体，合成了 1,827 个面向任务的环境。这些任务难以解决但易于验证。合成工作流主要包括环

境和工具集构建、任务合成以及解决方案生成。具体而言，工作流如下进行：

1. 给定一个任务类别（例如规划旅行行程）和一个配备 bash 和搜索工具的沙盒，智能体首先使用这些工具从互联网生成或检索相关数据，并将其存储在沙盒数据库中。
2. 随后，智能体合成一组特定于任务的工具，每个工具均实现为一个函数。
3. 为了创建既具挑战性又可自动验证的任务，智能体最初基于当前数据库提出一个简单的任务，并附带用 Python 实现的解决方案函数和验证函数。解决方案函数仅限于调用工具函数或执行逻辑计算，不能调用其他函数或直接访问数据库，从而确保任务只能通过工具接口解决。此外，解决方案函数产生的结果必须经过验证函数的校验。如果解决方案未通过验证，智能体将修改解决方案或验证函数，直到解决方案的输出通过验证为止。随后，智能体会迭代地增加任务难度，并更新相应的解决方案和验证函数。在此迭代过程中，如果当前工具集不足以解决任务，智能体将扩充工具集。

遵循该工作流，我们获得了数千个 environment, tools, task, verifier 元组。随后，我们使用 DeepSeek-V3.2 在该数据集上进行强化学习，并仅保留 pass@100 非零的实例，最终得到 1,827 个环境及其对应的任务（共 4,417 个）。下面展示了一个合成的旅行规划示例。该示例表明，尽管在庞大的组合空间中搜索满足所有约束的旅行计划具有挑战性，但检查给定候选方案是否满足这些约束则相对简单直接。

合成任务示例：旅行规划

我计划从杭州出发进行为期三天的旅行，需要帮助制定 2025 年 10 月 1 日至 10 月 3 日的行程。有几个重要要求：在整个旅行期间，我不想重复任何城市、酒店、景点或餐厅。此外，请确保您推荐的每家酒店、餐厅和景点实际上都位于我当天停留的城市。关于第二天的另一件事——我想在预算上精打细算。如果我最终预订了每晚 800 元人民币或以上的豪华酒店，那么我需要更仔细地控制其他开支：我在两家餐厅（午餐和晚餐）的总花费应保持在 350 元人民币以下，两家餐厅的评分至少为 4.0 星，下午的景点门票需低于 120 元人民币。如果第二天的酒店属于中高档（500-800 元人民币），那么我的灵活性会稍大一些——我只需确保至少一家餐厅的评分在 4.0 星或以上，且景点门票低于 180 元人民币。对于更经济的酒店（200-500 元人民币区间），我只需确保至少一家餐厅的评分在 3.2 星或以上。您能帮我制定这份行程吗?

提交结果格式

[
{ "time": "2025-10-01", "city": "cite_name", "hotel": "hotel_name", "afternoon_restaurant": "restaurant_name", "afternoon_attraction": "attraction_name", "evening_restaurant": "restaurant_name" },
{ "time": "2025-10-02", "city": "cite_name", "hotel": "hotel_name", "afternoon_restaurant": "restaurant_name", "afternoon_attraction": "attraction_name", "evening_restaurant": "restaurant_name" },
{ "time": "2025-10-03", "city": "cite_name", "hotel": "hotel_name", "afternoon_restaurant": "restaurant_name", "afternoon_attraction": "attraction_name", "evening_restaurant": "restaurant_name" }]

旅行规划工具集

函数名称	描述
`get_all_attractions_by_city(city)`	获取指定城市的所有景点。
`get_all_cities()`	从数据库中获取所有城市。
`get_all_hotels_by_city(city)`	获取指定城市的所有酒店。
`get_all_restaurants_by_city(city)`	获取指定城市的所有餐厅。
`get_city_by_attraction(attraction)`	根据给定景点名称获取所在城市。
`get_city_by_hotel(hotel)`	根据给定酒店名称获取所在城市。
`get_city_by_restaurant(restaurant)`	根据给定餐厅名称获取所在城市。
`get_city_transport(city)`	获取指定城市的所有市内交通选项。
`get_infos_by_attraction(info_keywords, attraction)`	获取指定景点的特定信息。
`get_infos_by_city(info_keywords, city)`	获取指定城市的特定信息。
`get_infos_by_hotel(info_keywords, hotel)`	获取指定酒店的特定信息。
`get_infos_by_restaurant(info_keywords, restaurant)`	获取指定餐厅的特定信息。
`get_inter_city_transport(from_city, to_city)`	获取指定城市对之间的所有交通方式。
`get_weather_by_city_date(city, date)`	获取指定城市-日期组合的天气。
`submit_result(answer_text)`	提交最终答案内容。

4. 评估

4.1. 主要结果

我们在以下基准上对模型进行评估：MMLU-Pro (?)、GPQA Diamond (?)、Human Last Exam (HLE) 纯文本版 (?)、LiveCodeBench (2024.08-2025.04)、Codeforces、Aider-Polyglot、AIME 2025、HMMT 2025 年 2 月、HMMT 2025 年 11 月 (?)、IMOAnswerBench (?)、Terminal Bench 2.0、SWE-Verified (?)、SWE Multilingual (?)、BrowseComp (?)、BrowseCompZh (?)、τ^2-bench (?)、MCP-Universe (?)、MCP-Mark (?) 以及 Tool-Decathlon (?)。工具使用基准采用标准函数调用格式进行评估，其中模型被配置为思考模式（thinking mode）。对于 MCP-Universe (?) 和 MCP-Mark (?)，我们使用内部环境对所有模型进行评估，因为搜索和 Playwright 环境可能与官方设置略有不同。我们将温度（temperature）设置为 1.0，上下文窗口设置为 128K tokens。对于 AIME、HMMT、IMOAnswerBench 和 HLE 等数学相关任务，我们使用以下模板进行评估："`{question}\nPlease reason step by step, and put your final answer within \boxed{}.`" 对于 HLE，我们额外使用官方模板对 DeepSeek-V3.2-Thinking 进行了评估，得分为 23.9。

DeepSeek-V3.2 在推理任务上的表现与 GPT-5-high 相当，但略逊于 Gemini-3.0-Pro。与 K2-Thinking 相比，DeepSeek-V3.2 在输出 token 数量大幅减少的情况下取得了相当的成绩，如表 2 所示。这些性能提升可归因于分配给强化学习（RL）训练的计算资源增加。在过去几个月里，我们观察到性能提升与扩展的 RL 训练预算呈正相关，该预算已超过预训练成本的 10%。我们假设，通过分配更多的计算预算，推理能力有望得到进一步提升。值得注意的是，本文展示的

表 1 | DeepSeek-V3.2 与闭源/开源模型的对比。对于开源模型，我们仅与在工具使用（tooluse）中支持思考（thinking）的模型进行比较。粗体数字表示各类模型（开源和闭源）中的最佳得分。τ^2-Bench 的结果为各类别的平均分。关于 BrowseComp，带有 * 标记的性能表示使用了上下文管理技术。

	基准测试 (指标)	Claude-4.5-Sonnet	GPT-5 High	Gemini-3.0 Pro	Kimi-K2 Thinking	MiniMax M2	DeepSeek-V3.2 Thinking
英语	MMLU-Pro (EM)	88.2	87.5	**90.1**	84.6	82.0	**85.0**
	GPQA Diamond (Pass@1)	83.4	85.7	**91.9**	**84.5**	77.7	82.4
	HLE (Pass@1)	13.7	26.3	**37.7**	23.9	12.5	**25.1**
代码	LiveCodeBench (Pass@1-COT)	64.0	84.5	**90.7**	82.6	83.0	**83.3**
	Codeforces (Rating)	1480	2537	**2708**	-	-	2386
数学	AIME 2025 (Pass@1)	87.0	94.6	**95.0**	**94.5**	78.3	93.1
	HMMT Feb 2025 (Pass@1)	79.2	88.3	**97.5**	89.4	-	**92.5**
	HMMT Nov 2025 (Pass@1)	81.7	89.2	**93.3**	89.2	-	**90.2**
	IMOAnswerBench (Pass@1)	-	76.0	**83.3**	**78.6**	-	78.3
代码智能体	Terminal Bench 2.0 (Acc)	42.8	35.2	**54.2**	35.7	30.0	**46.4**
	SWE Verified (Resolved)	**77.2**	74.9	76.2	71.3	69.4	**73.1**
	SWE Multilingual (Resolved)	**68.0**	55.3	-	61.1	56.5	**70.2**
搜索智能体	BrowseComp (Pass@1)	24.1	**54.9**	-	-/60.2*	44.0	**51.4/67.6***
	BrowseCompZh (Pass@1)	42.4	63.0	-	62.3	48.5	**65.0**
	HLE (Pass@1)	32.0	35.2	**45.8**	**44.9**	31.8	40.8
工具使用	τ^2-Bench(Pass@1)	84.7	80.2	**85.4**	74.3	76.9	**80.3**
	MCP-Universe (Success Rate)	46.5	47.9	**50.7**	35.6	29.4	**45.9**
	MCP-Mark (Pass@1)	33.3	**50.9**	43.1	20.4	24.4	**38.0**
	Tool-Decathlon (Pass@1)	**38.6**	29.0	36.4	17.6	16.0	**35.2**

DeepSeek-V3.2 性能受到长度约束奖励模型的限制；在解除该限制后，我们观察到模型性能进一步提升，详见第 4.2 节。

在代码智能体（code agent）评估中，DeepSeek-V3.2 在 SWE-bench Verified 和 Terminal Bench 2.0 上均显著优于开源大语言模型，展现了其在真实编码工作流中的潜力。关于 Terminal Bench 2.0，如前所述，我们针对 “思考模式” 的上下文管理策略目前与 Terminus 不兼容；因此，报告的 46.4 分是使用 Claude Code 框架取得的。我们还在非思考模式下使用 Terminus 对 DeepSeek-V3.2 进行了评估，得分为 39.3。对于 SWE-bench Verified，主要分数是使用我们的内部框架获得的。在其他设置（包括 Claude Code 和 RooCode 框架，以及非思考模式）下的鲁棒性测试产生了一致的结果，分数在 72 到 74 之间。

在搜索智能体评估中，我们使用标准商业搜索 API 对我们的模型进行评估。由于 DeepSeek-V3.2 仅支持最大 128K 的上下文长度，约 20%+ 的测试用例超出了此限制。为解决这一问题，我们采用了一种上下文管理方法来计算最终得分。作为参考，在不使用上下文管理的情况下，得分为 51.4。更多细节请参见第 4.4 节。

在工具使用基准上，DeepSeek-V3.2 大幅缩小了开源与闭源大语言模型之间的性能差距，尽管仍低于前沿模型。对于 τ^2-bench，我们直接使用模型本身作为用户智能体，最终类别得分分别为：航空 63.8、零售 81.1、电信 96.2。对于 MCP 基准，我们采用函数调用格式，并将工具输出放置在标记为 “tool” 角色的消息中，而非 “user” 角色。在测试过程中，我们观察到 DeepSeek-

表 2 | 推理模型在各项基准测试上的性能与效率。对于每个基准测试，单元格显示准确率和输出 token 数量（单位：千）。每个基准测试中最高的准确率以粗体显示；第二高的以下划线显示。

基准测试	GPT-5 High	Gemini-3.0 Pro	Kimi-K2 Thinking	DeepSeek-V3.2 Thinking	DeepSeek-V3.2 Speciale
AIME 2025 (Pass@1)	94.6 (13k)	95.0 (15k)	94.5 (24k)	93.1 (16k)	**96.0** (23k)
HMMT Feb 2025 (Pass@1)	88.3 (16k)	97.5 (16k)	89.4 (31k)	92.5 (19k)	**99.2** (27k)
HMMT Nov 2025 (Pass@1)	89.2 (20k)	93.3 (15k)	89.2 (29k)	90.2 (18k)	**94.4** (25k)
IMOAnswerBench (Pass@1)	76.0 (31k)	83.3 (18k)	78.6 (37k)	78.3 (27k)	**84.5** (45k)
LiveCodeBench (Pass@1-COT)	84.5 (13k)	**90.7** (13k)	82.6 (29k)	83.3 (16k)	88.7 (27k)
CodeForces (Rating)	2537 (29k)	**2708** (22k)	-	2386 (42k)	2701 (77k)
GPQA Diamond (Pass@1)	85.7 (8k)	**91.9** (8k)	84.5 (12k)	82.4 (7k)	85.7 (16k)
HLE (Pass@1)	26.3 (15k)	**37.7** (15k)	23.9 (24k)	25.1 (21k)	30.6 (35k)

V3.2 经常进行冗余的自我验证，生成长度过长的轨迹。这种倾向经常导致上下文长度超过 128K 的限制，尤其是在 MCP-Mark GitHub 和 Playwright 评估等任务中。因此，这一现象阻碍了 DeepSeek-V3.2 的最终性能表现。然而，结合上下文管理策略可以进一步提升性能。我们将此确定为未来的研究方向，也是用户在实际应用中需要考虑的因素。即使 DeepSeek-V3.2 存在这一问题，它仍然显著优于现有的开源模型。值得注意的是，由于这些基准中使用的环境和工具集在 RL 训练期间并未出现过，观察到的性能提升证明了 DeepSeek-V3.2 能够将其推理策略泛化到域外智能体场景的能力。智能体场景下非思考模型的评估结果如附录表 8 所示。

4.2. DeepSeek-V3.2-Speciale 的结果

表 2 表明，DeepSeek-V3.2-Speciale 通过利用更多的推理 token 实现了卓越的性能，在多个基准测试上超越了最先进的 Gemini-3.0-Pro。值得注意的是，如表 3 所示，该通用模型在未进行针对性训练的情况下，在 2025 年国际信息学奥林匹克竞赛（IOI）和 ICPC 世界总决赛（ICPC WF）中达到了金牌水平。此外，通过引入 ? 中的技术，该模型在复杂证明任务中表现优异，在 2025 年国际数学奥林匹克竞赛（IMO）和中国数学奥林匹克竞赛（CMO）中达到了金牌标准[5]。详细的评估协议见附录 D。

然而，DeepSeek-V3.2-Speciale 的 token 效率仍显著低于 Gemini-3.0-Pro。为降低部署成本和延迟，我们在官方 DeepSeek-V3.2 的训练过程中施加了更严格的 token 限制，旨在优化性能与成本之间的权衡。我们认为，token 效率仍是未来研究的关键方向。

4.3. 合成智能体任务

在本节中，我们进行消融实验以研究合成智能体任务的影响。我们重点关注两个问题。首先，合成任务是否对强化学习具有足够的挑战性？其次，这些合成任务的泛化能力如何，即它们能否迁移到不同的下游任务或真实世界环境中？

为回答第一个问题，我们从通用合成智能体任务中随机采样 50 个实例，并对用于合成的模

[5]我们评估的是 CMO 2025 的英文版。IMO 2025 和 CMO 2025 的题目以及推理代码可在以下地址获取：`https://github.com/deepseek-ai/DeepSeek-Math-V2`。

表 3 | DeepSeek-V3.2-Speciale 在顶级数学与编程竞赛中的表现。对于 ICPC WF 2025，我们报告了每道成功解出题目的提交次数。DeepSeek-V3.2-Speciale 在 ICPC WF 2025 中排名第 2，在 IOI 2025 中排名第 10。

竞赛	P1	P2	P3	P4	P5	P6	总分	奖牌
IMO 2025	7	7	7	7	7	0	35/42	金牌
CMO 2025	18	18	9	21	18	18	102/126	金牌
IOI 2025	100	82	72	100	55	83	492/600	金牌

竞赛	A	B	C	D	E	F	G	H	I	J	K	L	总分	奖牌
ICPC WF 2025	3	-	1	1	2	2	-	1	1	1	1	1	10/12	金牌

型以及前沿闭源大语言模型进行评估。如表 4 所示，DeepSeek-V3.2-Exp 的准确率仅为 12%，而前沿闭源模型的准确率最高仅为 62%。这些结果表明，合成数据包含了对于 DeepSeek-V3.2-Exp 和前沿闭源模型均具有挑战性的智能体任务。

表 4 | 不同模型在通用合成任务上的准确率。

Pass@K	DeepSeek-v3.2-Exp	Sonnet-4.5	Gemini-3.0 Pro	GPT-5-Thinking
1	12%	34%	51%	62%
2	18%	47%	65%	75%
4	26%	62%	74%	82%

为探究在合成数据上进行强化学习（RL）能否泛化至不同任务或真实世界环境，我们将 RL 应用于 DeepSeek-V3.2 的 SFT 检查点（记为 DeepSeek-V3.2-SFT）。为排除长思维链（CoT）和其他 RL 数据的影响，我们仅在非思考模式下对合成智能体任务进行 RL 训练。随后，我们将该模型与 DeepSeek-V3.2-SFT 和 DeepSeek-V3.2-Exp 进行比较，其中 DeepSeek-V3.2-Exp 仅在搜索和代码环境中使用 RL 进行训练。如图 5 所示，在合成数据上进行大规模 RL 训练在 Tau2Bench、MCP-Mark 和 MCP-Universe 基准测试上相较于 DeepSeek-V3.2-SFT 取得了显著提升。相比之下，将 RL 限制在代码和搜索场景中并未在这些基准测试上提升性能，这进一步凸显了合成数据的潜力。

4.4. 搜索智能体的上下文管理

即使采用 128k 等扩展上下文窗口，智能体工作流（尤其是在基于搜索的场景中）仍经常遇到最大长度限制，导致推理过程被提前截断。这一瓶颈阻碍了测试时计算潜力的充分发挥。为解决这一问题，我们引入了上下文管理机制，采用简单策略在测试时扩展 Token 预算。当 Token 使用量超过上下文窗口长度的 80% 时，该机制将被触发。这些策略包括 (1) **Summary**，对溢出的轨迹进行摘要并重新启动生成过程；(2) **Discard-75%**，丢弃轨迹中前 75% 的工具调用历史以释放空间；(3) **Discard-all**，通过丢弃所有先前的工具调用历史来重置上下文（类似于新上下文工具 (**?**)）。作为对比，我们还实现了一个并行扩展基线 **Parallel-fewest-step**，该基线采样 N

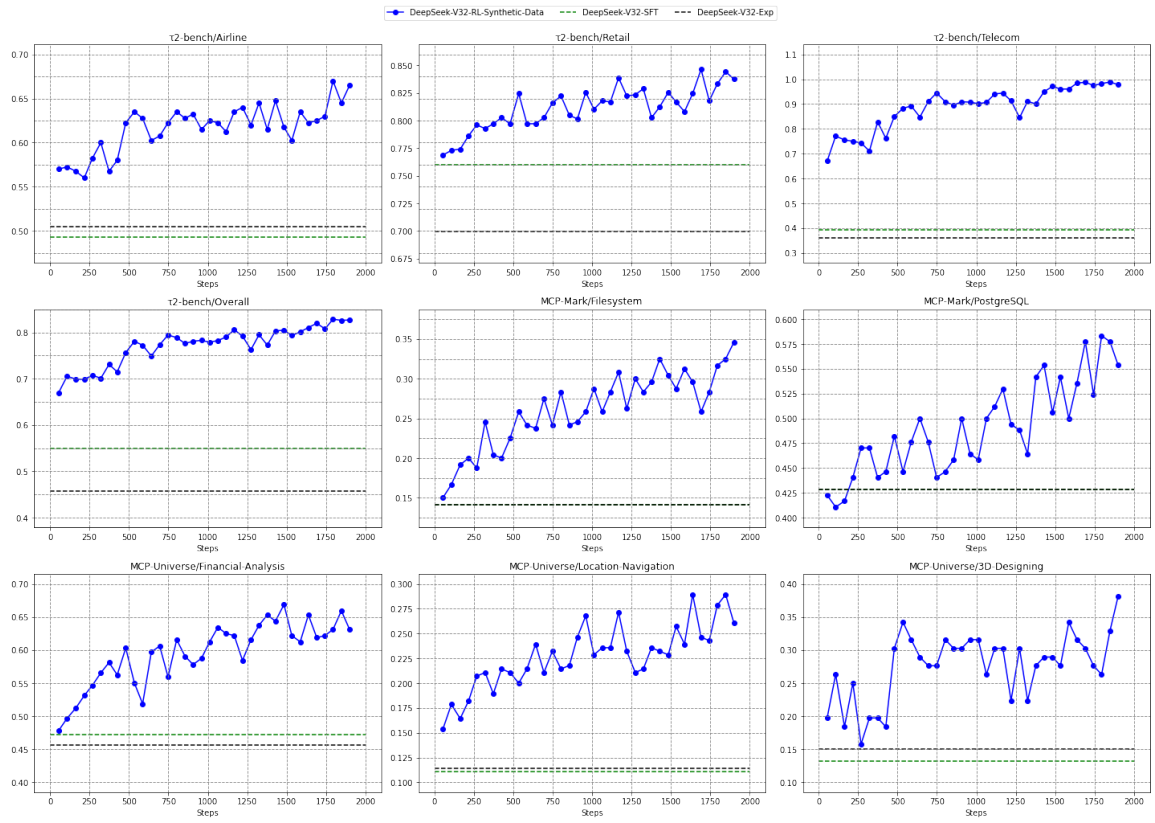


图 5 | 仅使用合成通用智能体数据对 DeepSeek-V3.2-SFT 进行强化学习训练。

条独立轨迹，并选择步数最少的轨迹。

我们在 BrowseComp 基准测试 (**?**) 上评估了这些策略。如图 6 所示，在不同的计算预算下，上下文管理通过允许模型扩展测试时计算，为执行额外的推理步骤提供了更多空间，从而带来了显著的性能提升。例如，Summary 策略将平均步数扩展至 364 步，性能提升高达 60.2。然而，其整体效率相对较低。尽管 Discard-all 策略较为简单，但其在效率和可扩展性方面均表现良好，取得了 67.6 的得分，在步数显著减少的情况下，性能与并行扩展相当。

综上所述，测试时计算既可以通过上下文管理进行串行扩展，也可以进行并行扩展，两者均能有效提升模型的解题能力。然而，不同策略在效率和可扩展性方面表现出显著差异。因此，在评估模型性能时，充分考虑实际计算成本至关重要。同时，寻找串行与并行扩展的最佳组合，以同时最大化效率与可扩展性，仍是未来工作的重要方向。

5. 结论、局限性与未来工作

在本工作中，我们提出了 DeepSeek-V3.2，这是一个有效弥合计算效率与高级推理能力之间差距的框架。借助 DSA，我们在不牺牲长上下文性能的前提下，解决了关键的计算复杂度问题。通过增加计算预算，DeepSeek-V3.2 在推理基准测试中达到了与 GPT-5 相当的性能。最后，我们大规模智能体任务合成流水线的集成显著提升了工具使用能力，为基于开源大语言模型构建稳健且可泛化的 AI 智能体开辟了新的可能性。此外，我们的高计算变体 DeepSeek-V3.2-Speciale

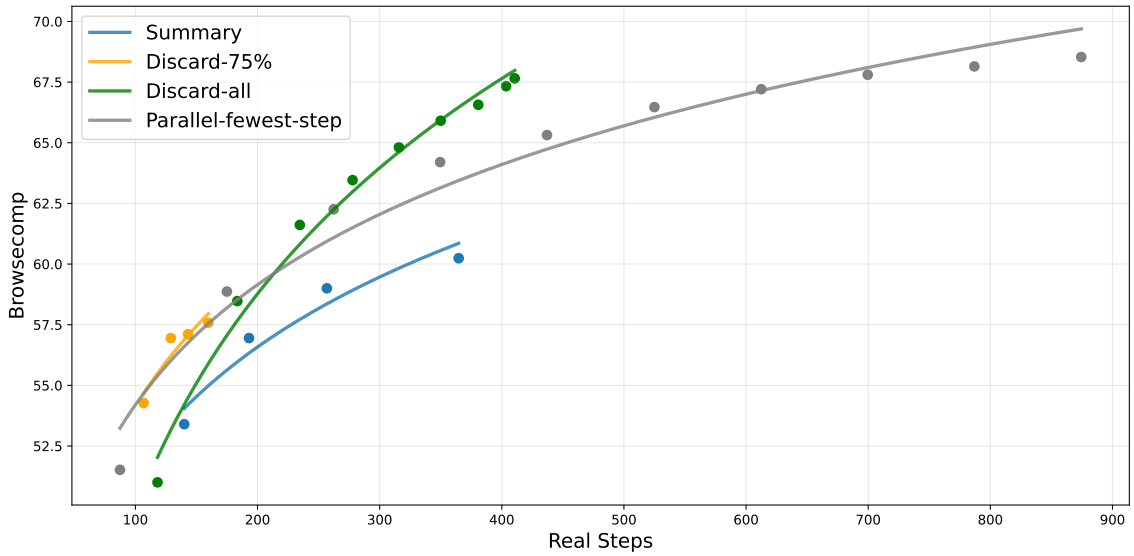


图 6 | 在不同测试时计算扩展策略下 Browsecomp 的准确率。

凭借在 IMO 和 IOI 中斩获金牌的优异成绩，为开源大语言模型树立了新的里程碑。

尽管取得了上述成果，但与 Gemini-3.0-Pro 等前沿闭源模型相比，我们仍承认存在一些局限性。首先，由于总训练 FLOPs 较少，DeepSeek-V3.2 的世界知识广度仍落后于领先的专有模型。我们计划在未来的迭代中通过扩大预训练计算规模来弥补这一知识差距。其次，Token 效率仍是一个挑战；DeepSeek-V3.2 通常需要更长的生成轨迹（即更多的 Token）才能达到与 Gemini-3.0-Pro 等模型相当的输出质量。未来的工作将专注于优化模型推理链的智能密度，以提升效率。第三，在解决复杂任务方面仍不及前沿模型，这将促使我们进一步优化基础模型与后训练方案。

附录

A. MLA 的 MHA 与 MQA 模式

图 7 展示了 MLA 的两个方面——MHA 与 MQA 模式——以及它们之间的转换。

B. 冷启动模板

表 5 | 推理数据系统提示词示例。该系统提示词要求模型在 <think></think> 标签内输出推理过程。

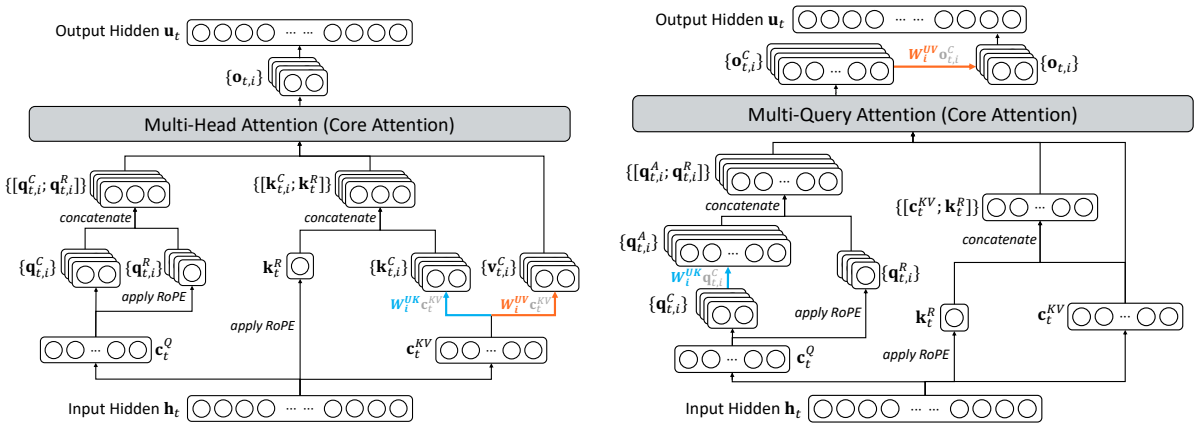
推理系统提示词	你是一位专业的 Python 程序员。你将收到一个问题（问题规范），并需要生成一个符合规范且能通过所有测试的正确 Python 程序。请在给出最终答案前先进行推理。推理过程需包含在 <think> </think> 标签内。最终答案在 </think> 标签后输出。
提示词	给定一个链表，两两交换其中相邻的节点，并返回交换后的链表头节点...
推理响应	<think> ... </think> [最终答案]

表 6 | {TOOL-DESCRIPTIONS} 和 {TOOLCALL-FORMAT} 将被替换为具体的工具及我们设计的工具调用格式。

Agent 系统提示词	使用 Python 解释器工具执行 Python 代码。该代码不会向用户显示。此工具应用于内部推理，而非用于打算向用户展示的代码（例如创建图表、表格或文件时）。当你向 python 发送包含 Python 代码的消息时，它将在有状态的 Jupyter notebook 环境中执行。python 将返回执行输出，或在 120.0 秒后超时。 ## 工具 你可以使用以下工具： {TOOL-DESCRIPTIONS} 重要提示：使用工具时请始终严格遵循以下格式： {TOOLCALL-FORMAT}
提示词	给定一个链表，两两交换其中相邻的节点，并返回交换后的链表头节点...
Agent 响应	[多轮工具调用] [最终答案]

表 7 | 模型在思考过程中执行工具调用。

需推理的 Agent 系统提示词	你是一个拥有 Python 解释器访问权限的有用助手。 - 在推理过程中（即在 <think></think> 内），你可以 ** 多次 ** 使用 Python 工具，最多执行 20 次代码。 - 在推理早期调用 Python 工具以辅助解决问题。继续推理并按需调用工具，直到得出最终答案。一旦得到答案，停止推理并使用 Markdown 和 LaTeX 展示你的解决方案。 - 在展示的最终解决方案步骤中，请勿调用任何工具。 - 为提高效率和准确性，只要可能，请优先使用代码执行而非基于语言的推理。保持推理简洁；让代码承担繁重的工作。 ## 工具 你可以使用以下工具： {TOOL-DESCRIPTIONS} 重要提示：使用工具时请始终严格遵循以下格式： {TOOLCALL-FORMAT}
提示词	给定一个链表，两两交换其中相邻的节点，并返回交换后的链表头节点...
带思考过程的 Agent 响应	<think> [多轮思考后工具调用] </think> [最终答案]



(a) MLA 的 MHA 模式。　　(b) MLA 的 MQA 模式。

图 7 | MLA 的 MHA 与 MQA 模式示意图。对于 DeepSeek-V3.1-Terminus，MHA 模式用于训练和预填充，而 MQA 模式用于解码。

C. 非思考型 DeepSeek-V3.2 Agent 评估

表 8 | DeepSeek-V3.2 非思考模式与思考模式的对比。表中的终端基准测试分数均基于 Claude Code 框架评估。基于 Terminus 框架的 Terminal Bench 2.0 非思考模式得分为 39.3。

	基准测试 (指标)	非思考模式	思考模式
代码智能体	Terminal Bench 2.0 (Acc)	37.1	46.4
	SWE Verified (Resolved)	72.1	73.1
	SWE Multilingual (Resolved)	68.9	70.2
工具使用	τ^2-bench (Pass@1)	77.2	80.3
	MCP-Universe (Success Rate)	38.6	45.9
	MCP-Mark (Pass@1)	26.5	38.0
	Tool-Decathlon (Pass@1)	25.6	35.2

非思考模式的表现略逊于思考模式，但仍具有竞争力。

D. IOI、ICPC 世界总决赛、IMO 和 CMO 的评估方法

在所有比赛中，模型的最大生成长度均设置为 128k。测试过程中不使用任何工具或互联网访问权限，并严格遵守比赛规定的时间限制和提交次数限制。

针对 IOI 评估，我们根据官方比赛规则设计了提交策略。该规则允许每道题最多提交 50 次，并根据所有子任务中获得的最高分进行评分。具体而言，我们首先为每道题采样 500 个候选解，随后应用多阶段过滤流程。在初始阶段，我们剔除了未能通过提供的样例测试用例或超出长度限制的无效提交。随后，我们使用 DeepSeek-V32-Exp 模型识别并移除那些模型明确表示无法或拒绝解题的样本。从剩余的候选解中，我们选择了思考轨迹最长的 50 个样本进行最终提交。

针对 ICPC 评估，我们采用了相同的过滤方法，但初始样本量较小。我们为每道题生成 32 个候选解，并应用相同的过滤标准来筛选提交。

在 IMO 和 CMO 任务中，我们采用“生成-验证-优化”循环。模型会迭代优化其解答，直到获得完美的自我评估或达到最大修改次数上限，该流程与 **?** 中的方法一致。

E. 作者名单

研究与工程: Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan*, Damai Dai, Daya Guo, Dejian Yang, Deli Chen, Erhang Li, Fangqi Zhou*, Fangyun Lin, Fucong Dai, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Hao Li, Haofen Liang, Haoran Wei, Haowei Zhang, Haowen Luo, Haozhe Ji, Honghui Ding, Hongxuan Tang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, Jialiang Huang, Jiashi Li, Jiaxin Xu, Jiewen Hu, Jingchang Chen, Jingting Xiang, Jingyang Yuan, Jingyuan Cheng, Jinhua Zhu, Jun Ran*, Junguang Jiang, Junjie Qiu, Junlong Li*, Junxiao Song, Kai Dong, Kaige Gao, Kang Guan, Kexin Huang*, Kexing Zhou, Kezhao Huang, Kuai Yu, Lean Wang, Lecong Zhang, Lei Wang, Liang Zhao, Liangsheng Yin*, Lihua Guo, Lingxiao Luo, Linwang Ma, Litong Wang, Liyue Zhang, M.S. Di, M.Y Xu, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Panpan Huang, Peixin Cong, Peiyi Wang, Qiancheng Wang, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Ruiling Xu, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, Runqiu Yin, Runxin Xu, Ruomeng Shen, Ruoyu Zhang, S.H. Liu, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoyuan Chen, Shengding Hu, Shengyu Liu, Shiqiang Hu, Shirong Ma, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, Songyang Zhou, Tao Ni, Tao Yun, Tian Pei, Tian Ye, Tianyuan Yue, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjun Gao, Wentao Zhang, Xi Gao, Xiangwen Wang, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xingkai Yu, Xingyou Li, Xinyu Yang, Xinyuan Li*, Xu Chen, Xuecheng Su, Xuehai Pan, Xuheng Lin, Xuwei Fu, Y.Q. Wang, Yang Zhang, Yanhong Xu, Yanru Ma, Yao Li, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yiliang Xiong, Ying He, Ying Zhou, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuduan Wang, Yue Gong*, Yuhan Wu, Yuheng Zou, Yukun Li, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Z.F. Wu, Z.Z. Ren, Zehua Zhao, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhibin Gou, Zhicheng Ma, Zhigang Yan, Zhihong Shao, Zhixian Huang, Zhiyu Wu, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihui Gu, Zijia Zhu, Zilin Li, Zipeng Zhang, Ziwei Xie, Ziyi Gao, Zizheng Pan, Zongqing Yao

数据标注: Bei Feng, Hui Li, J.L. Cai, Jiaqi Ni, Lei Xu, Meng Li, Ning Tian, R.J. Chen, R.L. Jin, S.S. Li, Shuang Zhou, Tianyu Sun, X.Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xinnan Song, Xinyi Zhou, Y.X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Zhen Huang, Zhipeng Xu, Zhongyu Zhang

商务与合规: Dongjie Ji, Jian Liang, Jianzhong Guo, Jin Chen, Leyi Xia, Miaojun Wang, Ming-

ming Li, Peng Zhang, Ruyi Chen, Shangmian Sun, Shaoqing Wu, Shengfeng Ye, T.Wang, W.L. Xiao, Wei An, Xianzu Wang, Xiaowen Sun, Xiaoxiang Wang, Ying Tang, Yukun Zha, Zekai Zhang, Zhe Ju, Zhen Zhang, Zihua Qu

作者名单按名字（First Name）字母顺序排列。带有 * 标记的名字表示已离开我们团队的成员。