DeepSeek-V4：迈向高效百万Token上下文智能

DeepSeek-V4: Towards Highly Efficient Million-Token Context Intelligence

DeepSeek Team

摘要

DeepSeek-V4 支持百万级 token 上下文窗口，采用 Hybrid Attention 混合注意力架构，具备世界顶级推理性能。相比前代模型，Agent 能力大幅提高，支持更长上下文理解和更复杂的任务规划。模型已在网页端、APP 和 API 全面上线。V4 在推理效率上实现突破性优化，通过创新的注意力机制和上下文管理技术，在保持高性能的同时大幅降低计算成本。

Introduction

引言 4 2 架构 6

专家并行中的细粒度通信-计算重叠 15

高性能批次无关与确定性内核库 18

mHC 的高性价比与内存高效实现 21

面向长上下文注意力的上下文并行 21

用于灵活激活检查点的扩展自动微分 21

面向全词汇表 OPD 的高效教师调度 34

Conclusion, Limitations, and Future Directions

44

A 作者名单与致谢

54

A.1 作者名单 . . . . . .

54

A.2 致谢 . . . . . .

55

B 评估细节

55

3

1. Introduction

推理模型（DeepSeek-AI, 2025; OpenAI, 2024c）的兴起确立了测试时扩展（test-time scaling）的新范式，推动了大型语言模型（LLMs）性能的显著提升。然而，该扩展范式从根本上受限于标准注意力机制（vanilla attention mechanism）的二次方计算复杂度（Vaswani et al., 2017），这为超长上下文和推理过程带来了难以逾越的瓶颈。与此同时，长视野场景与任务的涌现——从复杂的智能体工作流到大规模跨文档分析——也使得对超长上下文的高效支持成为未来发展的关键。尽管近期的开源工作（Bai et al., 2025a; DeepSeek-AI, 2024; MiniMax, 2025; Qwen, 2025）提升了通用能力，但在处理超长序列时存在的核心架构低效问题仍是主要障碍，限制了测试时扩展的进一步收益，并阻碍了对长视野场景与任务的深入探索。为突破超长上下文下的效率瓶颈，我们研发了 DeepSeek-V4 系列模型，包括预览版的 DeepSeek-V4-Pro（1.6T 参数，激活 49B）和 DeepSeek-V4-Flash（284B 参数，激活 13B）。通过架构创新，DeepSeek-V4 系列在处理超长序列的计算效率上实现了显著飞跃。这一突破实现了对百万级 token 上下文长度的高效支持，为下一代大语言模型开启了百万级上下文的新纪元。我们相信，高效处理超长序列的能力将开启测试时扩展的下一前沿，为深入探索长视野任务铺平道路，并为探索在线学习等未来范式奠定必要基础。与 DeepSeek-V3 架构（DeepSeek-AI, 2024）相比，DeepSeek-V4 系列保留了 DeepSeekMoE 框架（Dai et al., 2024）与多 Token 预测（MTP）策略，同时在架构与优化方面引入了多项关键创新。为提升长上下文效率，我们设计了一种结合压缩稀疏注意力（Compressed Sparse Attention, CSA）与重度压缩注意力（Heavily Compressed Attention, HCA）的混合注意力机制。CSA 沿序列维度对 KV 缓存进行压缩，随后执行 DeepSeek 稀疏注意力（DSA）（DeepSeek-AI, 2025）；而 HCA 则对 KV 缓存施加更激进的压缩，但保留密集注意力计算。

为增强建模能力，我们引入了流形约束超连接（Manifold-Constrained Hyper-Connections, mHC）（Xie et al., 2026），对传统残差连接进行了升级。此外，我们将 Muon 优化器（Jordan et al., 2024; Liu et al., 2025）引入 DeepSeek-V4 系列的训练中，从而实现了更快的收敛速度与更高的训练稳定性。为实现 DeepSeek-V4 系列的高效训练与推理，并提升开发效率，我们引入了多项基础设施优化。首先，我们为 MoE 模块设计并实现了一个单一融合算子，实现了计算、通信与内存访问的完全重叠。其次，我们采用领域特定语言（DSL）TileLang（Wang et al., 2026），以平衡开发生产力与运行时效率。第三，我们提供了高效的批次不变且确定性的算子库，

以确保训练与推理过程中的逐位可复现性。第四，我们对 MoE 专家权重及索引器 QK 路径引入 FP4 量化感知训练，以降低内存占用与计算开销。第五，在训练框架方面，我们通过张量级检查点扩展自动微分框架，以实现细粒度的重计算控制；同时，我们结合针对 Muon 优化器的混合 ZeRO 策略、通过重计算与融合算子实现的低成本 mHC 方案，以及用于管理压缩注意力机制的两阶段上下文并行策略，进一步提升了训练效率。最后，在推理框架方面，我们设计了结合磁盘存储策略的异构 KV 缓存结构，以实现共享前缀的高效复用。

通过采用混合 CSA 与 HCA，并结合计算与存储的精度优化，与 DeepSeek-V3.2 相比，DeepSeek-V4 系列实现了显著更低的推理 FLOPs 和大幅缩减的 KV 缓存大小，尤其在长上下文场景下表现更为突出。图 1 右侧展示了 DeepSeek-V3.2 与 DeepSeek-V4 系列的单 token 推理 FLOPs 估算值及累积 KV 缓存大小。在 1M token 上下文场景中，即使激活参数更多的 DeepSeek-V4-Pro，其单 token FLOPs（以等效 FP8 FLOPs 计）和 KV 缓存大小也仅为 DeepSeek-V3.2 的 27% 和 10%。

此外，DeepSeek-V4-Flash凭借更少的激活参数，将效率进一步提升：在1M token上下文设置下，相较于DeepSeek-V3.2，其单token浮点运算次数（FLOPs）仅为10%，KV缓存占用仅为7%。此外，DeepSeek-V4系列的路由专家参数采用FP4精度。尽管在现有硬件上，FP4×FP8运算的峰值FLOPs与FP8×FP8相同，但在未来硬件上，理论上可实现1/3的效率提升，这将进一步增强DeepSeek-V4系列的计算效率。在预训练阶段，我们分别在32T token和33T token的数据集上训练了DeepSeek-V4-Flash与DeepSeek-V4-Pro。预训练完成后，这两个模型均能原生且高效地支持1M长度的上下文。在内部评估中，凭借更高效的参数设计，DeepSeek-V4-Flash-Base已在大多数基准测试中超越了DeepSeek-V3.2-Base。DeepSeek-V4-Pro-Base则进一步放大了这一优势，在DeepSeek基础模型中树立了新的性能标杆，在推理、代码、长上下文及世界知识任务上实现了全面领先。DeepSeek-V4系列的后训练流程采用两阶段范式：首先独立训练领域特定专家，随后通过同策略蒸馏（on-policy distillation）实现统一模型的整合（Lu and Lab，2025）。初始阶段，针对数学、代码、智能体（agent）及指令遵循等各个目标领域，分别独立训练专属的专家模型。基座模型首先在高质量的领域特定数据上进行监督微调（SFT），以奠定基础能力。随后，采用基于组相对策略优化（GRPO）的强化学习（RL）（DeepSeek-AI，2025），在针对特定成功标准定制的奖励模型引导下，进一步优化模型以生成符合领域规范的行为。该阶段最终产出一系列各具专长的领域专家模型，均在各自领域表现卓越。最后，为整合上述各项专长，通过同策略蒸馏训练出一个统一的单一模型。在此过程中，统一模型作为学生模型，通过优化与教师模型之间的反向KL散度（reverse KL loss）进行学习。

核心评估结果摘要 知识能力：在广泛的世界知识评估中，DeepSeek-V4-Pro 的最大推理努力模式（DeepSeek-V4-Pro-Max）在 SimpleQA (OpenAI, 2024d) 和 Chinese-SimpleQA (He et al., 2024) 基准测试上显著优于领先的开源模型。在教育知识方面（通过 MMLU-Pro (Wang et al., 2024b)、HLE (Phan et al., 2025) 和 GPQA (Rein et al., 2023) 进行评估），DeepSeek-V4-Pro-Max 相比其开源同类模型展现出微小的领先优势。尽管在这些基于知识的评估中仍略逊于领先的闭源模型 Gemini-3.1-Pro，但 DeepSeek-V4-Pro-Max 已显著缩小了与它的差距。 推理能力：通过扩展推理 token 数量，DeepSeek-V4-Pro-Max 在标准推理基准测试上展现出优于 GPT-5.2 和 Gemini-3.0-Pro 的性能。然而，其性能仍略逊于 GPT-5.4 和 Gemini-3.1-Pro，这表明其发展轨迹大约落后于当前最先进的前沿模型 3 到 6 个月。此外，DeepSeek-V4-Flash-Max 实现了与 GPT-5.2 和 Gemini-3.0-Pro 相当的性能，确立了其作为复杂推理任务高性价比架构的地位。

在智能体评估中，尽管DeepSeek-V4-Flash-Max在多项基准测试上的性能与DeepSeek-V4-Pro-Max相当，但在更为复杂、高难度的任务上，其表现仍落后于规模更大的对应版本。

2. Architecture

总体而言，DeepSeek-V4 系列保留了 Transformer (Vaswani et al., 2017) 架构与多Token预测（MTP）模块 (DeepSeek-AI, 2024; Gloeckle et al., 2024)，同时在 DeepSeek-V3 的基础上引入了多项关键升级：(1) 首先，我们引入流形约束超连接（mHC）(Xie et al., 2026) 以增强传统残差连接；

(2) 其次，我们设计了一种混合注意力架构，通过压缩稀疏注意力与重度压缩注意力大幅提升了长上下文效率。(3) 第三，我们采用 Muon (Jordan et al., 2024; …

2.1. Designs Inherited from DeepSeek-V3

混合专家（MoE）。 与以往的 DeepSeek 系列模型（DeepSeek-AI, 2024; DeepSeek-AI, 2024）一致，DeepSeek-V4 系列在前馈神经网络（FFN）中同样采用了 DeepSeekMoE 范式（Dai et al., 2024），该范式配置了细粒度路由专家与共享专家。与 DeepSeek-V3 不同，我们将计算亲和度分数的激活函数从 Sigmoid(·)

更改为 Sqrt(Softplus(·))。在负载均衡方面，我们采用了无辅助损失策略（DeepSeek-AI, 2024; Wang et al., 2024a），并辅以轻微的序列级平衡损失，以防止单个序列内部出现极端的不平衡现象。针对 DeepSeek-V4，我们移除了对路由目标节点数量的限制，并重新设计了并行策略以维持训练效率。此外，与 DeepSeek-V3 相比，我们将初始几个 Transformer 块中的稠密 FFN 层替换为采用哈希路由（Hash routing）（Roller et al., 2021）的 MoE 层。哈希路由策略根据针对输入 token ID 预定义的哈希函数，来确定每个 token 的目标专家。

多 Token 预测（MTP）。 与 DeepSeek-V3 类似，DeepSeek-V4 系列也设置了 MTP 模块与训练目标。鉴于 MTP 策略已在 DeepSeek-V3 中得到验证，我们在 DeepSeek-V4 系列中直接沿用该策略，未作任何修改。

2.2. Manifold-Constrained Hyper-Connections

如图2所示，DeepSeek-V4系列引入了流形约束超连接（mHC）（Xie et al., 2026），以增强相邻Transformer块之间的传统残差连接。与朴素超连接（HC）（Zhu et al., 2025）相比，mHC的核心思想是将残差映射约束至特定的流形上，从而在保持模型表达能力的同时，提升跨层信号传播的稳定性。本节简要介绍标准HC，并阐述我们如何设计mHC以实现稳定训练。

标准超连接。标准HC将残差流的宽度扩展 n_{hc} 倍。具体而言，残差流的形状从 \mathbb{R}^d 扩展至 \mathbb{R}^{n_{hc} \times d}，其中 d 为实际层输入的隐藏维度。令 X_l = [x_{l,1}; \dots; x_{l,n_{hc}}]^T \in \mathbb{R}^{n_{hc} \times d} 表示第 l 层之前的残差状态。HC引入了三个线性映射：输入映射 A_l \in \mathbb{R}^{1 \times n_{hc}}、残差变换 B_l \in \mathbb{R}^{n_{hc} \times n_{hc}} 以及输出映射 C_l \in \mathbb{R}^{n_{hc} \times 1}。残差状态的更新公式如下： X_{l+1} = B_l X_l + C_l F_l(A_l X_l)，(1) 其中 F_l 表示第 l 层（例如MoE层），其输入与输出形状均为 \mathbb{R}^d。需要注意的是，实际层输入 A_l X_l \in \mathbb{R}^d 同样为 d 维，因此扩展后的残差宽度不会影响内部层的设计。HC将残差宽度与实际隐藏维度解耦，在计算开销极小的情况下提供了一个互补的扩展维度，因为 n_{hc} 通常远小于隐藏维度 d。然而，尽管HC在提升模型性能方面展现出潜力，但我们发现，在堆叠多层时，训练过程经常会出现数值不稳定性，这阻碍了HC的进一步扩展。

流形约束残差映射。mHC的核心创新在于将残差映射矩阵 B_l 约束到双随机矩阵（Birkhoff多面体）的流形 \mathcal{M} 上，从而增强跨层信号传播的稳定性：B_l \in \mathcal{M} \coloneqq \{ M \in \mathbb{R}^{n \times n} \mid M 1_n = 1_n, 1_n^T M = 1_n^T, M \geq 0 \}。(2) 该约束确保了映射矩阵的谱范数 \|B_l\|_2 被限制在1以内，使得残差变换具有非扩张性，从而提高了前向传播与反向传播过程中的数值稳定性。此外，集合 \mathcal{M} 对乘法封闭，这保证了在mHC深层堆叠场景下的稳定性。

此外，输入变换 和输出变换 也通过 Sigmoid 函数约束为非负且有界，以避免信号相消的风险。 动态参数化。三个线性映射的参数是动态生成的，它们被分解为动态（输入相关）分量和静态（输入无关）分量。给定输入 ∈Rhc×，首先对其进行展平和归一化：= RMSNorm(vec()) ∈R1×hc。随后，我们遵循传统的超连接（HC）方法生成无约束的原始参数 ∈R1×hc、∈Rhc×hc 和 ∈Rhc×1： = pre · (pre) + pre, (3) = res · Mat(res) + res, (4) = post · (post)+ post, (5) 其中，pre, post∈Rhc×hc 和 res∈Rhc×2hc 是用于生成动态分量的可学习参数；Mat(·) 将大小为 1 × 2hc 的向量重塑为大小为 hc × hc 的矩阵；pre∈R1×hc、post∈Rhc×1 和 res∈Rhc×hc 是可学习的静态偏置；pre、res、post∈R 是初始化为较小值的可学习门控因子。

应用参数约束。在获得无约束原始参数 、、 后，我们对其施加前述约束以增强数值稳定性。具体而言，对于输入和输出映射，我们采用 Sigmoid 函数 (·) 以确保其非负性与有界性： = ( ), (6) = 2( ). (7) 对于残差映射 ，我们将其投影到双随机矩阵流形 M 上。这通过 Sinkhorn-Knopp 算法实现，该算法首先对 应用指数函数以确保正性，得到 (0) = exp( )，随后迭代执行列归一化和行归一化： () = T(T((1))), (8) 其中 T 和 T 分别表示行归一化和列归一化。该迭代过程收敛至受约束的双随机矩阵 = (max)。我们选取 max = 20 作为实际经验值。

… KV Token 的隐藏状态 … 压缩索引键 Query Token 的隐藏状态 多查询注意力 压缩 KV 条目 … Top-k 选择器 选中的压缩 KV 条目 … 共享键值多查询注意力 索引查询 查询 滑动窗口 KV 条目 拼接 Token 级压缩器 … Token 级压缩器 闪电索引器 索引分数

该方法将 KV 条目数量压缩至原来的 1/m，随后应用 DeepSeek 稀疏注意力机制以进一步加速。此外，将少量滑动窗口 KV 条目与选定的压缩 KV 条目相结合，以增强局部细粒度依赖关系。

2.3. Hybrid Attention with CSA and HCA

当上下文长度达到极端规模时，注意力机制逐渐成为模型中的主要计算瓶颈。针对DeepSeek-V4，我们设计了两种高效的注意力架构——压缩稀疏注意力（Compressed Sparse Attention, CSA）与强压缩注意力（Heavily Compressed Attention, HCA）——并采用其交错混合配置，从而大幅降低了长文本场景下的注意力计算开销。CSA融合了压缩与稀疏注意力策略：首先将每个token的键值（Key-Value, KV）缓存压缩为单个条目，随后应用DeepSeek稀疏注意力（DeepSeek Sparse Attention, DSA）（DeepSeek-AI, 2025），使得每个查询token仅关注个压缩后的KV条目。HCA旨在实现极致压缩，通过将每´（≫）个token的KV缓存整合为单个条目来实现。CSA与HCA的混合架构显著提升了DeepSeek-V4系列的长上下文效率，使得在实际应用中支持百万级token的上下文成为可能。本节将阐述该混合注意力架构的核心技术，同时我们提供了开源实现[1]，以明确无误地说明更多细节。

2.3.1. Compressed Sparse Attention

2.3.1. 压缩稀疏注意力

CSA的核心架构如图3所示，该架构首先将每个token的KV缓存压缩为一个条目，随后应用DeepSeek稀疏注意力以进一步加速。压缩键值条目。设∈R×为输入隐藏状态序列，其中为序列长度，为隐藏层维度。CSA首先计算两组KV条目，
∈R×及其对应的压缩权重，∈R×，其中为头

1https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro/tree/main/inference

维度：

= · ,

= · ,

(9)

= · ,

= · ,

(1...

2.3.1. 压缩稀疏注意力

接下来，查询词元 与前置压缩块 之间的索引得分 ,∈R（

最后，我们将中间输出 $[o'_{G,t,1}; o'_{G,t,2}; \ldots; o'_{G,t,g}] \in \mathbb{R}^{d_g g}$ 投影至最终的注意力输出 $\hat{o}_t \in \mathbb{R}^d$。

2.3.2. Heavily Compressed Attention

2.3.2. 重度压缩注意力

HCA的核心架构如图4所示，该架构以更高的压缩程度对KV缓存进行压缩，但不采用稀疏注意力机制。

压缩键值条目。

总体而言，HCA的压缩策略与CSA相似，但采用了更大的压缩率´（≫），且不执行重叠 11 压缩。设∈R×为输入隐藏状态序列，HCA首先计算原始KV条目∈R×及其对应的压缩权重∈R×：

= · ,

(20)

= · ,

(21)

其中,∈R×为可训练参数。接下来，中的每´个KV条目将...

2.3.3. Other Details

除了上述描述的CSA和HCA核心架构外，我们的混合注意力机制还结合了其他几项技术。为保持行文清晰，我们在前述介绍中省略了这些附加技术，并将在本小节中对其进行简要说明。此外，本小节仅聚焦于这些技术的核心思想，为简化起见可能会省略部分细微细节。我们建议读者参考我们的开源实现以获取明确无误的细节。 查询与键值条目归一化。对于CSA和HCA，我们在核心注意力操作之前，对查询的每个头以及压缩KV条目的唯一头额外执行一次RMSNorm操作。该归一化操作可避免注意力logits爆炸，并可能提升训练稳定性。

部分旋转位置编码。对于CSA和HCA，我们部分采用旋转位置编码（RoPE）(Su et al., 2024) 应用于注意力查询、KV条目以及核心注意力输出。具体而言，对于CSA和HCA中使用的每个查询向量和KV条目向量，我们对其最后64个维度应用RoPE。由于KV条目同时充当注意力键和值，朴素的核心注意力输出 {o_{t,i}} 将携带绝对位置编码，该编码源自KV条目的加权和。作为应对措施，我们还在每个 o_{t,i} 的最后64个维度上应用位置为 -i 的RoPE。通过这种方式，核心注意力的输出也将携带相对位置编码——每个KV条目对核心注意力输出的贡献也将与查询和KV条目之间的距离相关。 滑动窗口注意力的附加分支。为了在CSA和HCA中严格保持因果性，每个查询仅关注其前面的压缩KV块。因此，查询无法访问其自身压缩块内其他token的信息。同时，在语言建模中，近期的token通常与查询token具有更高的相关性。基于这些原因，我们以滑动窗口的方式为CSA和HCA引入了一个补充注意力分支，以更好地建模局部依赖关系。具体而言，对于每个查询token，我们额外生成对应于最近 n_win 个token的 n_win 个未压缩KV条目。在CSA和HCA的核心注意力中，滑动窗口内的这些KV条目将与压缩KV条目一同使用。

在 CSA 和 HCA 的核心注意力机制中，我们采用了注意力汇聚（attention sink）技巧（OpenAI, 2025; Xiao et al., 2024）。具体而言，我们设置了一系列可学习的汇聚对数几率 {′ 1, ′ 2, ..., ′ }。对于第 个注意力头，Exp(′) 将被添加到注意力得分的分母中：

,,=

Exp(,,)

Σ

Exp(,,) + Exp(′

) ,

(27)

其中 ,,, ,,∈R 分别表示第 个注意力头中第 个查询 token 与第 个前置 token 或压缩块之间的注意力得分与注意力对数几率。该技术允许每个查询头调整其总注意力得分，使其...

2.3.4. Efficiency Discussion

得益于混合CSA与HCA的采用，以及低精度计算与存储的结合，DeepSeek-V4系列的注意力模块在注意力FLOPs和KV缓存大小方面均实现了显著的效率提升，尤其在长上下文场景中表现突出。首先，我们对KV条目采用混合存储格式：旋转位置编码（RoPE）维度使用BF16精度，其余维度则采用FP8精度。与纯BF16存储相比，这种混合表示方式将近KV缓存大小减半。其次，闪电索引器（lightning indexer）内的注意力计算采用FP4精度执行，从而在极长上下文下加速注意力操作。第三，相较于DeepSeek-V3.2，DeepSeek-V4系列采用了更小的注意力top-k值，从而提升了模型在短文本和中长文本上的效率。最后，也是最重要的一点，压缩注意力与混合注意力技术大幅降低了KV缓存大小与计算FLOPs。 以头维度为128的BF16 GQA8（Ainslie等人，2023）作为基线（这是大语言模型注意力的一种常见配置），在1M上下文设置下，DeepSeek-V4系列的KV缓存大小可大幅缩减至该基线的约2%。

此外，即使与DeepSeek-V3.2（DeepSeek-AI，2025）这一本身已具备较高效率的基线相比，DeepSeek-V4系列在效率方面仍展现出显著优势。图1右侧展示了两者在推理FLOPs与KV缓存大小方面的对比。

2.4. Muon Optimizer

由于 Muon（Jordan 等，2024；Liu 等，2025）优化器具有更快的收敛速度和更高的训练稳定性，我们在 DeepSeek-V4 系列模型的大多数模块中采用了该优化器。我们 Muon 优化的完整算法总结于算法 1。

基本配置。我们对嵌入模块、预测头模块、mHC 模块的静态偏置和门控因子，以及所有 RMSNorm 模块的权重保留使用 AdamW（Loshchilov 和 Hutter，2017）优化器。其余所有模块均使用 Muon 进行更新。遵循 Liu 等（2025）的方法，我们也对 Muon 参数应用权重衰减，采用 Nesterov（Jordan 等，2024；Nesterov，1983）技巧，并对更新矩阵的均方根（RMS）进行重缩放，以便复用我们的 AdamW 超参数。与他们不同的是，我们采用混合牛顿-舒尔茨（Newton-Schulz）迭代进行正交化。

混合牛顿-舒尔茨迭代。对于给定矩阵 M，设其奇异值分解（SVD）为 M = UΣV^T。牛顿-舒尔茨迭代旨在将 M 近似正交化为 UV^T。通常，M 会首先被归一化为 M_0 = M / ||M||_F，以确保其最大奇异值不超过 1。随后，每次牛顿-舒尔茨迭代执行以下操作： M_k = a M_{k-1} + b (M_{k-1} M_{k-1}^T) M_{k-1} + c (M_{k-1} M_{k-1}^T)^2 M_{k-1}。 (28)

我们的混合牛顿-舒尔茨方法在两个不同阶段共执行 10 次迭代。在前 8 步中，我们使用系数 (a, b, c) = (3.4445, -4.7750, 2.0315) 以驱动快速收敛，使奇异值接近 1。在最后 2 步中，我们切换至系数 (a, b, c) = (2, -1.5, 0.5)，从而将奇异值精确稳定在 1。

避免注意力 Logits 爆炸。DeepSeek-V4 系列的注意力架构允许我们直接在注意力查询（queries）和键值（KV）条目上应用 RMSNorm，这有效防止了注意力 logits 的爆炸。因此，我们在 Muon 优化器中未采用 QK-Clip 技术（Liu 等，2025）。

3.1. Fine-Grained Communication-Computation Overlap in Expert Parallelism

混合专家（MoE）模型可通过专家并行（EP）进行加速。然而，EP需要复杂的节点间通信，并对互联带宽和延迟提出了较高要求。为了缓解EP中的通信瓶颈，并在较低的互联带宽需求下实现更高的端到端性能，我们提出了一种细粒度的EP方案，该方案将通信与计算融合至单个流水线内核中，以实现通信-计算重叠。通信延迟可被隐藏。我们EP方案的核心洞察在于，MoE层中的通信延迟可被有效地隐藏在计算之下。如图5所示，在DeepSeek-V4系列模型中，每个MoE层主要可分解为四个阶段：两个通信主导阶段（分发Dispatch与合并Combine）和两个计算主导阶段（线性层1 Linear-1与线性层2 Linear-2）。我们的性能剖析表明，在单个MoE层内，通信的总耗时少于计算的总耗时。因此，在将通信与计算融合为统一流水线后，计算仍是主要的性能瓶颈，这意味着系统可以在不降低端到端性能的前提下容忍较低的互联带宽。

L1 Act L2 (a) 朴素方案 通信 计算 L1 Act L2 理论加速比：1.42× (b) Comet 分发 计算 激活与合并 L1 L2 L1 L2 L1 L2 Act Act Act 专家波次1 专家波次2 专家波次3 理论加速比：1.92× (c) 本文方案 分发 All-to-All 线性层1 GEMM SwiGLU + FP8格式转换

线性层2 GEMM 合并 All-to-All 图5 | 本文EP方案及相关工作的示意图。Comet（Zhang et al., 2025b）分别将分发（Dispatch）与线性层1（Linear-1）重叠，以及将线性层2（Linear-2）与合并（Combine）重叠。我们的EP方案通过将专家划分并调度为多个波次，实现了更细粒度的重叠。理论加速比基于DeepSeek-V4-Flash架构的配置进行评估。 细粒度EP方案。为了进一步降低互联带宽需求并放大重叠带来的收益，我们引入了一种更细粒度的专家划分方案。受多项相关工作的启发（Aimuyo et al., 2025; Zhang et al., 2025b），我们将专家划分并调度为多个波次（waves）。每个波次仅包含少量专家。一旦波次内的所有专家完成通信，计算即可立即开始，而无需等待其他专家。

在稳态下，当前波次的计算、下一波次的Token传输以及已完成专家的结果发送均并发进行，如图5所示。这在专家之间形成了一个细粒度流水线，使整个波次过程中的计算与通信保持连续。基于波次的调度提升了极端场景下的性能，例如强化学习（RL）rollout过程，该过程通常会遇到长尾小批次。
性能与开源超级内核。我们在NVIDIA GPU和华为昇腾NPU平台上验证了该细粒度专家并行（EP）方案。与强大的非融合基线相比，该方案在通用推理工作负载上实现了1.50~1.73倍的加速，在强化学习rollout和高速智能体服务等延迟敏感场景下最高可达1.96倍。我们已将基于CUDA的超级内核实现MegaMoE2作为DeepGEMM的组件开源。

观察与建议。我们分享了内核开发过程中的观察与经验教训，并向硬件厂商提出若干建议，以期助力高效的硬件设计并实现更好的软硬件协同设计： 计算-通信比。完全的计算-通信重叠取决于计算-通信比，而非仅仅取决于带宽。设峰值计算吞吐量为 C，互联带宽为 B，当 C/B \le V_{comp}/V_{comm} 时，通信可被完全隐藏，其中 V_{comp} 表示计算量，V_{comm} 表示通信量。对于DeepSeek-V4-Pro，每个token-专家对需要 6hd FLOPs（SwiGLU门控、上投影和下投影）但仅需 3h 字节的通信（FP8分发与BF16合并），该条件可简化为：C/B \le 2d = 6144 FLOPs/Byte。也就是说，每GBps的互联带宽足以隐藏6.1 TFLOP/s计算所产生的通信开销。一旦带宽达到此阈值，它便不再是瓶颈，此时投入额外的硅片面积来进一步提升带宽将带来边际收益递减。我们鼓励未来的硬件设计瞄准此类平衡点，而非无条件地扩展带宽。 功耗预算。极致的内核融合会同时使计算、内存和网络处于高负载状态，使得功耗降频成为限制性能的关键因素。我们建议未来的硬件设计为此类完全并发的工作负载提供充足的功耗余量。
通信原语。

我们采用基于拉取的机制，每个GPU主动从远程GPU读取数据，从而避免了细粒度推送所带来的高通知延迟。未来具备更低延迟跨GPU信号传输能力的硬件将使推送机制变得可行，并支持更自然的通信模式。 激活函数。我们提议用一种低成本的逐元素激活函数替代SwiGLU，该激活函数不涉及指数或除法运算。这直接减轻了GEMM后的处理负担；且在相同参数预算下，移除门控投影可扩大中间维度，从而进一步放宽对带宽的要求。

3.2. Flexible and Efficient Kernel Development with TileLang

在实际应用中，我们复杂的模型架构本会产生数百个细粒度的 PyTorch ATen 算子。我们采用 TileLang (Wang et al., 2026) 开发了一组融合内核以替代其中绝大多数算子，从而以极小的开发代价实现最优性能。该工具还使我们在验证阶段能够快速原型化注意力变体等算子。这些内核在模型架构开发、大规模训练以及最终的推理服务生产部署中发挥着关键作用。作为一种领域特定语言（DSL），TileLang 在开发生产力与运行时效率之间取得了平衡，既支持快速开发，又能在同一代码库内实现深度、迭代的优化。此外，我们与 TileLang 社区紧密合作，致力于构建更加敏捷、高效且稳定的内核开发工作流。 通过主机端代码生成降低调用开销。随着加速器性能不断提升，CPU 端的调度开销日益凸显。对于小型且高度优化的内核，此类固定的主机端开销极易成为利用率与吞吐量的瓶颈。该开销的一个常见来源是，主机端逻辑（如运行时契约检查）通常出于灵活性考虑使用 Python 编写，从而导致每次调用产生固定的额外成本。我们通过主机端代码生成（Host Codegen）来缓解这一开销，将大部分主机端逻辑迁移至生成的主机代码中。具体而言，我们首先在中间表示（IR）层级联合生成设备内核与轻量级主机启动器（launcher），并将从语言前端解析出的必要元数据（如数据类型、秩/形状约束以及步长/布局假设）嵌入其中。随后，该启动器被转换（lowered）为基于 TVM-FFI (Chen et al., 2018) 框架构建的主机源代码，其紧凑的调用约定与零拷贝张量互操作机制共同将主机端开销降至最低。在运行时，该生成的主机代码负责执行验证与参数编组，将所有每次调用的检查操作移出 Python 执行路径。我们的测量结果表明，CPU 端的验证开销已从每次调用的数十至数百微秒降至不足一微秒。

SMT 求解器辅助的形式化整数分析。TileLang 内核涉及复杂的张量索引算术运算，需要强大的形式化整数分析支持。

在布局推断、内存冲突检测和边界分析等编译阶段，编译器必须验证整数表达式是否满足特定属性，以启用相应的优化。因此，更强的形式化分析能力能够解锁更高级、更复杂的优化机会。为此，我们将 Z3 SMT 求解器（De Moura 和 Bjørner, 2008）集成到 TileLang 的代数系统中，为张量程序中的大多数整数表达式提供形式化分析能力。通过将 TileLang 的整数表达式转换为 Z3 的无量词非线性整数算术（QF

_NIA），我们在计算开销与形式化表达能力之间取得了平衡。基于整数线性规划（ILP）求解器，QF_NIA 能够无缝解析内核中常见的标准线性整数表达式。此外，其固有的非线性推理能力有效应对了诸如针对可变张量形状的向量化等高级挑战。在合理的资源限制下，Z3 提升了整体优化性能，同时将编译时间开销控制在仅数秒内。该优化对多个编译阶段均产生了显著影响，涵盖向量化、屏障插入与代码简化等。 数值精度与逐位可复现性。在生产环境中，数值正确性与可复现性与原始吞吐量同等重要。因此，我们默认优先保证精度：编译器层面禁用了快速数学（fast-math）优化，而影响精度的近似计算仅作为显式的、需手动启用的前端算子提供（例如 T.__exp、T.__log 和 T.__sin）。相反，当需要严格的 IEEE-754 语义时，TileLang 提供了符合 IEEE 标准的内建函数，并支持显式舍入模式（例如 T.ieee_fsqrt、T.ieee_fdiv 和 T.ieee_add），使开发者能够精确指定数值行为。我们还致力于实现逐位可复现性，以便将生成的内核与手写 CUDA 基线进行验证对比。我们将 TileLang 的代数简化与降级（lowering）规则与主流 CUDA 工具链（如 NVCC）保持一致，以避免引入非预期位级差异的转换。布局注解（例如 T.annotate_layout）进一步允许用户固定依赖于布局的降级决策，保持求值与累加顺序与参考 CUDA 实现一致，从而在需要时实现逐位相同的输出。

我们的评估表明，这些面向精度与可复现性的设计决策并未牺牲性能：在保守的默认配置下，TileLang内核的性能依然保持竞争力；同时，框架提供了可调参数，允许用户有选择地放宽数值约束以提升运行速度。

3.3. High-Performance Batch-Invariant and Deterministic Kernel Libraries

为实现高效的训练与推理，我们开发了一套全面的高性能计算内核。除了提供基础功能并最大化硬件利用率外，另一个关键的设计目标是确保预训练、后训练和推理流水线之间的训练可复现性及逐位对齐。因此，我们实现了端到端、逐位批次不变且确定性的内核，同时将性能开销降至最低。这些内核有助于调试、稳定性分析以及保持后训练行为的一致性。 批次不变性（Batch Invariance）。批次不变性确保任意给定 token 的输出保持逐位一致，无论其在批次中的位置如何。为实现批次不变性，主要挑战如下： 注意力机制（Attention）。为实现批次不变性，我们无法采用 split-KV 方法（Dao 等，2023），该方法会将单个序列的注意力计算分布在多个流多处理器（SM）上以平衡 SM 负载。然而，放弃该技术会导致严重的线程束量化（wave-quantization）问题3，从而对 GPU 利用率产生不利影响。为此，我们开发了一种用于批次不变解码的双内核策略。第一个内核在单个 SM 内计算整个序列的注意力输出，确保满载线程束（wave）的高吞吐量。第二个内核为了最小化最后一个部分填充线程束的延迟，从而缓解线程束量化问题，采用多个 SM 处理单个序列。为保证这两个内核的逐位一致性，我们精心设计了第二个内核的计算路径，确保其累加顺序与第一个内核完全相同。此外，第二个内核利用线程块集群内的分布式共享内存4，实现了跨 SM 的高速数据交换。这种双内核方法有效地将批次不变解码的开销控制在可忽略不计的水平。 矩阵乘法（Matrix Multiplication）。传统的 cuBLAS 库（NVIDIA Corporation, 2024）无法实现批次不变性。因此，我们使用 DeepGEMM（Zhao 等，2025）对其进行端到端替换。此外，对于极小的批次大小，传统实现通常采用 split-k（Osama 等，2023）技术以提升性能。

遗憾的是，split-k 技术无法保证批次不变性，而这是 DeepSeek-V4 的一项关键特性[3][4][18]。因此，我们在大多数场景中放弃了 split-k 技术，但这可能会导致性能下降。为此，我们引入了一系列优化措施，使我们的矩阵乘法实现能够在大多数主要场景中达到甚至超越标准 split-k 的性能。

确定性。 确定性训练对于调试硬件或软件问题极具价值。此外，当训练出现损失突增等异常时，确定性使研究人员能够更轻松地定位数值原因，并进一步优化模型设计。训练中的非确定性通常源于非确定性的累加顺序，这往往是由于使用了原子加法指令所致。该问题主要发生在反向传播阶段，具体体现在以下几个部分： **注意力反向传播（Attention Backward）。** 在稀疏注意力机制的传统反向传播实现中，我们使用 `atomicAdd` 来累加 KV token 的梯度。由于浮点加法不满足结合律，这会引入非确定性。为解决该问题，我们为每个流多处理器（SM）分配独立的累加缓冲区，随后在所有缓冲区之间执行全局确定性求和。 **MoE 反向传播（MoE Backward）。** 当来自不同 rank 的多个 SM 并发地向接收端 rank 的同一缓冲区写入数据时，协商写入位置也会引入非确定性。为解决此问题，我们在每个单 rank 内部设计了 token 顺序预处理机制，并结合跨多 rank 的缓冲区隔离策略。该策略确保了专家并行发送结果以及 MoE 反向传播中累加顺序的确定性。 **mHC 中的矩阵乘法。** mHC 涉及一个输出维度仅为 24 的矩阵乘法。在极小批次大小下，我们不得不使用 split-k 算法（Osama 等，2023），其朴素实现会导致非确定性。为克服这一问题，我们分别输出每个 split 部分，并在后续的内核中执行确定性归约，从而在保持性能的同时确保确定性。

3.4. FP4 Quantization-Aware Training

为实现部署时的推理加速与显存节省，我们在训练后阶段引入了量化感知训练（QAT）(Jacob et al.,

2018)，使模型能够适应量化带来的精度下降。我们将 FP4（MXFP4）量化 (Rouhani et al., 2023) 应用于两个组件：(1) MoE 专家权重，这是 GPU 显存占用的主要来源 (OpenAI, 2025)；(2) CSA 索引器中的查询-键（QK）路径，其中 QK 激活值的缓存、加载与乘法运算完全在 FP4 精度下进行，从而加速长上下文场景下的注意力分数计算。此外，在此 QAT 过程中，我们进一步将索引分数 :,: 从 FP32 量化至 BF16。该优化使 top-k 选择器实现了 2 倍加速，同时保持了 99.7% 的 KV 条目召回率。 对于 MoE 专家权重，遵循 QAT 的常规做法，优化器维护的 FP32 主权重首先被量化为 FP4，随后反量化回 FP8 用于计算。值得注意的是，我们的 FP4 到 FP8 反量化过程是无损的。这是因为与 FP4（E2M1）相比，FP8（E4M3）多出 2 个指数位，提供了更大的动态范围。因此，只要每个 FP8 量化块（128 × 128 块）内 FP4 子块（1 × 32 块）的最大与最小缩放因子之比不超过某一阈值，细粒度的缩放信息即可被 FP8 扩展的动态范围完全吸收。我们通过实验验证了当前权重满足该条件。这使得整个 QAT 流程无需任何修改即可完全复用现有的 FP8 训练框架。在反向传播中，梯度针对前向传播中相同的 FP8 权重进行计算，并直接回传至 FP32 主权重，这等价于在量化操作上应用直通估计器（STE）。这也避免了对转置权重重新量化的需求。

在强化学习训练的推理与 rollout 阶段（不涉及反向传播），我们直接使用真实的 FP4 量化权重，而非模拟量化。这确保了采样阶段的模型行为与在线部署完全一致，同时减少了算子显存加载以实现实际加速，并显著降低了显存消耗。我们对 CSA 索引器中的 QK 路径采用相同的处理方式。

3.5. Training Framework

本训练框架基于为DeepSeek-V3（DeepSeek-AI，2024）所开发的可扩展且高效的基础设施构建。在训练DeepSeek-V4时，我们继承了这一稳健基础，并引入了多项关键创新以适配其新颖的架构组件（具体包括Muon优化器、mHC及混合注意力机制），同时保持了高训练效率与稳定性。

3.5.1. Efficient Implementation of Muon

Muon 优化器需要完整的梯度矩阵来计算参数更新，这在与零冗余优化器（ZeRO）结合时带来了挑战。传统的 ZeRO 专为 AdamW 等逐元素优化器设计，在此类优化器中，单个参数矩阵可在多个 rank 间进行划分与更新。为解决这一冲突，我们为 Muon 设计了一种混合的 ZeRO 分桶分配策略。针对稠密参数，我们限制了 ZeRO 并行度的最大规模，并采用背包算法将参数矩阵分配至各 rank，以确保每个 rank 的负载大致均衡。每个 rank 上的分桶会进行填充，以匹配跨所有 rank 的最大分桶尺寸，从而提升 reduce-scatter 操作的效率。在我们的配置中，每个 rank 最多管理五个参数矩阵，此类填充通常带来的内存开销不足 10%。当数据并行的整体规模超出 ZeRO 限制时，我们会在额外的数据并行组中冗余计算 Muon 更新，以计算开销换取总分桶内存的降低。

对于 MoE 参数，我们独立优化每个专家。我们首先将所有层中所有专家的 SwiGLU 下投影矩阵展平，随后依次展平上投影矩阵与门控矩阵。接着，对展平后的向量进行填充，以确保能将其均匀分配至所有 rank，且不会拆分任何逻辑上独立的矩阵。鉴于专家数量庞大，我们不对 MoE 参数的 ZeRO 并行度设置上限，且填充开销亦可忽略不计。
此外，在每个 rank 上，形状相同的连续参数将自动合并，从而支持批量执行牛顿-舒尔茨（Newton-Schulz）迭代，以提升硬件利用率。进一步地，我们观察到，当采用 BF16 矩阵乘法计算时，Muon 中的牛顿-舒尔茨迭代仍能保持稳定。基于此，我们进一步采用随机舍入方式，将需在数据并行 rank 间同步的 MoE 梯度量化至 BF16 精度，使通信量减半。为避免低精度加法器引入的累积误差，我们采用两阶段方法替代了传统的基于树或环的 reduce-scatter 集合通信。首先，通过 all-to-all 操作在各 rank 间交换局部梯度；随后，每个 rank 在 FP32 精度下执行局部求和。该设计有效保障了数值鲁棒性。20

3.5.2. Cost-Effective and Memory-Efficient Implementation of mHC

与传统残差连接相比，引入mHC会增加激活内存消耗以及流水线阶段间的通信量。为缓解这些开销，我们实施了多项优化策略。首先，我们为训练和推理阶段精心设计并实现了mHC的融合内核。其次，我们引入了一种重计算策略，用于选择性保存中间张量的检查点。具体而言，我们重新计算层间的大部分隐藏状态以及所有归一化层输入，同时避免对计算密集型操作进行重计算，从而在内存节省与计算开销之间取得平衡。第三，我们调整了DualPipe的1F1B重叠调度方案，以适应增加的流水线通信量，并实现mHC中部分操作的并发执行。综合来看，这些优化将mHC的墙钟时间开销控制在仅为重叠1F1B流水线阶段的6.7%。关于工程优化的更多细节，请参阅专门介绍mHC的论文（Xie et al., 2026）。

3.5.3. Contextual Parallelism for Long-Context Attention

传统的上下文并行（CP）对序列维度进行划分，每个 rank 维护连续的 个 token。这为我们的压缩注意力机制（即 CSA 和 HCA）带来了两项挑战。一方面，训练样本由多个序列拼接打包而成，每个序列独立以 （或

´）倍率进行压缩，末尾不足 个的 token 将被丢弃。因此，压缩后的 KV 长度通常小于 /，且在不同 rank 间各不相同。另一方面，压缩过程需要 个连续的 KV 条目，这些条目可能会跨越两个相邻 CP rank 之间的边界。为应对这些挑战，我们设计了一种两阶段通信方案。在第一阶段，每个 rank 将其最后 个未压缩的 KV 条目发送至 rank +1。随后，rank +1 将接收到的部分条目与其本地的 个未压缩 KV 条目一同压缩，生成长度固定为 / + 1 的压缩条目，其中包含部分填充（padding）条目。在第二阶段，通过所有 CP rank 上的 All-Gather 操作收集本地压缩的 KV 条目。接着，一个融合的选择与填充（select-and-pad）算子将它们重新组织为完整的压缩 KV 条目集合，总长度为 cp_size · /。所有填充条目均置于末尾。对于 HCA 和 CSA 中的索引器（indexer），每个查询 token 对应的压缩 KV 条目的可见范围可通过规则预先计算。对于 CSA 中的稀疏注意力机制，top- 选择器会为每个查询显式指定可见压缩 KV 条目的索引。

3.5.4. Extended Automatic Differentiation for Flexible Activation Checkpointing

传统的激活检查点实现通常以整个模块为粒度，在反向传播阶段决定保留或重新计算其输出激活。这种粗粒度往往导致重计算开销与激活内存占用之间的权衡难以达到最优。另一种替代方案是手动实现整个网络层的前向与反向逻辑，并显式管理张量的检查点状态。尽管该方法支持细粒度控制，但牺牲了自动微分框架的便捷性，大幅增加了开发复杂度。 为在不牺牲编程效率的前提下实现细粒度控制，我们实现了一种支持自动微分的张量级激活检查点机制。借助该机制，开发者仅需实现前向传播过程，并选择性地对特定张量进行标注，即可实现自动检查点保存与重计算。本框架利用 TorchFX (Reed et al., 2022) 追踪完整的计算图。针对每个被标注的张量，框架会执行反向遍历，以定位重计算该张量所需的最小计算子图。我们将此类最小子图定义为"重计算图"，并将其插入至反向传播逻辑中，置于对应梯度计算步骤之前。

与手动实现方案相比，该设计在训练阶段不会引入任何额外开销。本框架中的重计算通过直接释放被标注张量的 GPU 内存，并重用重计算所得张量的底层存储指针来实现，全程无需进行 GPU 内存拷贝。此外，由于计算图追踪过程会实际执行模型，我们能够精确追踪每个张量的底层存储指针，从而实现对共享存储张量（如 reshape 操作的输入与输出）重计算过程的自动去重。这一机制使开发者在进行重计算标注时，无需再深入考量底层内存管理的细节。

3.6. Inference Framework

我们的推理框架主要继承自DeepSeek-V3，仅在KV缓存管理方面存在部分差异。

3.6.1. KV Cache Structure and Management

为高效管理DeepSeek-V4中混合注意力机制产生的异构KV缓存，我们设计了一种定制化的KV缓存布局。该布局如图6所示，下文将对其进行详细阐述。 DeepSeek-V4中的异构KV条目。DeepSeek-V4系列的混合注意力机制引入了多种类型的KV条目，这些条目具有不同的键值（KV）缓存大小和更新规则。用于稀疏选择的闪电索引器向KV缓存中引入了额外的维度，其嵌入大小与主注意力机制中的不同。CSA和HCA中采用的压缩技术分别将序列长度缩减为原来的1/和1/´，从而降低了整体KV缓存的大小。因此，不同层之间的KV缓存大小各不相同。此外，滑动窗口注意力（SWA）层也使用不同的KV缓存大小，并采用独立的缓存命中与驱逐策略。在压缩分支中，每个token生成一个KV条目。当剩余token数量不足以进行压缩时，所有待处理的token及其关联的隐藏状态必须保留在缓冲区中，直到能够执行压缩操作。这些缓冲的token代表了由位置上下文决定的序列状态，同样在KV缓存框架内进行统一管理。

混合注意力KV缓存管理的挑战。混合注意力机制违背了PagedAttention及其变体背后的基本假设。

尽管近期的混合KV缓存管理算法（例如Jenga (Zhang et al., 2025a)、Hymba (Dong et al., 2025)）针对通用混合注意力模型或特定结构进行了优化，但在PagedAttention框架下将所有层的KV缓存进行统一整合仍面临两大主要障碍：
多样化的缓存策略，例如滑动窗口注意力（Sliding Window Attention）中使用的策略。
高性能注意力内核施加的限制，包括对齐要求。

为了实现DeepSeek-V4的高效KV缓存管理，我们设计了相应的策略以克服这两项挑战。 用于SWA和未压缩尾部Token的状态缓存。为了解决第一个障碍，我们采用了一种替代的缓存管理机制。由于SWA旨在有限的KV缓存容量下提升性能，将其与压缩分支中的未压缩尾部token一同视为状态空间模型是合理的。因此，相应的KV缓存可被视为仅依赖于当前位置的序列特定状态。据此，我们预先分配一个固定且有限大小的状态缓存池，并动态地将其分配给每个序列。
稀疏注意力内核协同设计。

针对第二个障碍，传统的高性能注意力算子通常假设每个块包含固定数量的个token以优化性能，这对应于CSA中的·个原始token以及HCA中的·´个原始token。通过采用高性能的稀疏注意力算子，不同网络层能够支持每个块包含可变数量的token，且不会导致性能下降。实现这一目标需要对KV缓存布局与稀疏注意力算子进行协同设计。例如，对块进行填充以使其与缓存行对齐，可有效提升性能。因此，对于压缩比为的CSA

与压缩比为′的HCA，每个块所包含的原始token数量可以是这两个压缩比最小公倍数lcm(, ′)的任意整数倍。

3.6.2. On-Disk KV Cache Storage

在提供 DeepSeek-V4 推理服务时，我们采用了一种基于磁盘的 KV 缓存存储机制，以消除共享前缀请求的重复预填充（prefilling）开销。针对 CSA/HCA 中的压缩 KV 条目与滑动窗口注意力（SWA）中的未压缩 KV 条目，我们分别设计了独立的存储管理方案。 对于 CSA 和 HCA，我们直接将所有压缩后的 KV 条目持久化至磁盘。当请求命中已缓存的前缀时，系统会读取并重用该前缀对应的压缩 KV 条目，直至最后一个完整的压缩块。特别地，对于位于尾部不完整块中的前缀 token，由于 CSA 和 HCA 并未存储未压缩的 KV 条目，我们仍需对其进行重新计算以恢复未压缩的 KV 状态。

对于 SWA KV 条目，由于其未进行压缩且存在于模型的每一层，其数据量约为压缩后 CSA 和 HCA KV 条目的 8 倍。为高效管理这些庞大的 SWA KV 条目，我们提出并实现了三种不同的磁盘 SWA KV 缓存管理策略，每种策略均在存储开销与计算冗余之间提供了不同的权衡： 全量 SWA 缓存（Full SWA Caching）。该策略存储所有 token 的完整 SWA KV 条目，从而实现计算零冗余。在此策略下，命中前缀的 SWA KV 条目仅需读取该前缀内最后 win 个 token 的磁盘缓存即可重建。尽管实现了计算零冗余，但该策略在现代基于 SSD 的存储系统中效率较低——每次命中请求仅会访问已存储 SWA KV 缓存的一小部分，从而导致访问模式不均衡且呈现写入密集型特征。
周期性检查点（Periodic Checkpointing）。该策略每隔 个 token 对最后 win 个 token 的 SWA KV 条目保存一次检查点，其中 为可调参数。对于命中前缀的请求，系统加载最近的检查点状态，随后重新计算剩余的尾部 token。通过调节参数 ，该策略可实现存储开销与计算成本之间的按需权衡。 零 SWA 缓存（Zero SWA Caching）。该策略不存储任何 SWA KV 条目。对于命中前缀的请求，需执行更多的重新计算以恢复 SWA KV 条目。具体而言，在每一注意力层中，每个 token 的 SWA KV 条目仅依赖于上一层最近 win 个 token 的 SWA KV 条目。因此，借助已缓存的 CSA 和 HCA KV 条目，仅需重新计算最后 win · 个 token，即可恢复 层模型的最后 win 个 SWA KV 条目。

根据具体的部署场景，我们选择最合适的策略，以实现存储与计算之间期望的权衡。

4.1. Data Construction

在DeepSeek-V3预训练数据的基础上，我们致力于构建一个更多样化、更高质量且具备更长有效上下文的训练语料库。我们持续优化数据构建流程。针对网络来源数据，我们实施了过滤策略以剔除批量自动生成及模板化内容，从而降低模型崩溃的风险（Zhu et al., 2024）。数学与编程语料依然是我们训练数据的核心组成部分；此外，我们在中期训练阶段引入了智能体数据，以进一步提升DeepSeek-V4系列的代码能力。针对多语言数据，我们为DeepSeek-V4构建了规模更大的语料库，从而提升其对跨文化长尾知识的捕捉能力。在DeepSeek-V4的数据构建中，我们特别注重长文档数据的筛选与整理，优先纳入学术论文、技术报告等具有独特学术价值的材料。综合上述各类数据，我们的预训练语料库规模超过32万亿（32T）个词元（tokens），涵盖数学内容、代码、网页、长文档及其他高质量类别。 在预训练数据方面，我们主要沿用了DeepSeek-V3的预处理策略。在分词方面，基于DeepSeek-V3的分词器，我们引入了少量用于上下文构建的特殊词元，并将词表大小保持为128K。同时，我们继承了DeepSeek-V3的词元分割（token-splitting）与中间填充（Fill-in-the-Middle, FIM）策略（DeepSeek-AI, 2024）。受Ding等人（2024）工作的启发，我们将不同来源的文档打包组合为合适的序列，以最大程度减少样本截断。与DeepSeek-V3不同，我们在预训练阶段采用了样本级注意力掩码。

4.2.1. Model Setups

DeepSeek-V4-Flash。 我们将Transformer层数设置为43，隐藏层维度d设置为4096。对于前两层，我们采用纯滑动窗口注意力。对于后续层，CSA与HCA交替使用。对于CSA，我们将压缩率m设置为4，索引器查询头数量n_h^I设置为64，索引器头维度c_I设置为128，用于稀疏注意力的KV条目数量（即注意力top-k）设置为512。对于HCA，我们将压缩率m'设置为128。对于CSA和HCA，我们将查询头数量n_h设置为64，头维度c设置为512，查询压缩维度d_c设置为1024。输出投影组数量g设置为8，每个中间注意力输出的维度d_g设置为1024。对于滑动窗口注意力的附加分支，窗口大小n_{win}设置为128。我们在所有Transformer块中均引入MoE层，但对前3个MoE层采用Hash路由策略。每个MoE层包含1个共享专家和256个路由专家，其中每个专家的中间隐藏层维度为2048。在路由专家中，每个Token将激活6个专家。多Token预测深度设置为1。对于mHC，扩展因子n_{hc}设置为4，Sinkhorn-Knopp迭代次数t_{max}设置为20。在此配置下，DeepSeek-V4-Flash的总参数量为284B，其中每个Token激活的参数量为13B。

DeepSeek-V4-Pro。 我们将Transformer层数设置为61，隐藏层维度d设置为7168。对于前两层，我们采用HCA。对于后续层，CSA与HCA交替使用。对于CSA，我们将压缩率m设置为4，索引器查询头数量n_h^I设置为64，索引器头维度c_I设置为128，用于稀疏注意力的KV条目数量（即注意力top-k）设置为1024。对

于HCA，我们将压缩率m'设置为128。对于CSA和HCA，我们将查询头数量n_h设置为128，头维度c设置为512，查询压缩维度d_c设置为1536。输出投影组数量g设置为16，每个中间注意力输出的维度d_g设置为1024。对于滑动窗口注意力的附加分支，窗口大小n_{win}设置为

128. We employ MoE layers in all Transformer blocks, but use the Hash routing strategy for the

我们在所有 Transformer 模块中均采用 MoE 层，但仅对前 3 个 MoE 层使用 Hash 路由策略。每个 MoE 层包含 1 个共享专家和 384 个路由专家，其中每个专家的中间隐藏维度为 3072。在路由专家中，每个 token 将激活 6 个专家。多 token 预测深度设置为 1。对于 mHC，扩展因子 hc 设置为 4，Sinkhorn-Knopp 迭代次数 max 设置为 20。在此配置下，DeepSeek-V4-Pro 的总参数量为 1.6T，其中每个 token 激活的参数量为 49B。

4.2.2. Training Setups

DeepSeek-V4-Flash。我们对大部分参数采用 Muon 优化器（Jordan 等，2024；Liu 等，2025），但对嵌入模块、预测头模块以及所有 RMSNorm 模块的权重使用 AdamW 优化器（Loshchilov 和 Hutter，2017）。对于 AdamW，我们将其超参数设置为 1 = 0.9，2 = 0.95， = 10^2，以及 weight_decay = 0.1。对于 Muon，我们将动量设置为 0.95，权重衰减设置为 0.1，并将每个更新矩阵的均方根（RMS）重新缩放至 0.18，以便复用 AdamW 的学习率。我们在 32T 个 token 上训练 DeepSeek-V4-Flash。与 DeepSeek-V3 类似，我们也采用了一种批次大小调度策略，将批次大小（以 token 计）从较小值逐步增加至 75.5M，并在训练的大部分阶段保持在该值。学习率在前 2000 步进行线性预热，并在训练的大部分阶段保持在 2.7×10^4。在训练接近尾声时，我们最终按照余弦调度策略将学习率衰减至 2.7×10。训练初始序列长度为 4K，我们逐步将训练序列长度扩展至 16K、64K 和 1M。关于稀疏注意力的设置，我们首先在前 1T 个 token 上使用密集注意力对模型进行预热，然后在序列长度达到 64K 时引入稀疏注意力，并在后续训练中保持使用该机制。在引入注意力稀疏性时，我们首先设置一个较短的阶段来预热 CSA 中的 lightning indexer，随后在训练的大部分阶段使用稀疏注意力训练模型。对于无辅助损失的负载均衡，我们将偏置更新速度设置为 0.001。对于平衡损失，我们将其损失权重设置为 0.0001，以避免单序列内出现极端不平衡。MTP 损失权重在训练的大部分阶段设置为 0.3，在学习率开始衰减时调整为 0.1。 DeepSeek-V4-Pro。除部分超参数的具体取值外，DeepSeek-V4-Pro 的训练设置与 DeepSeek-V4-Flash 基本一致。我们对大部分参数采用 Muon 优化器，但对嵌入模块、预测头模块以及所有 RMSNorm 模块的权重使用 AdamW 优化器。AdamW 和 Muon 的超参数与 DeepSeek-V4-Flash 相同。我们在 33T 个 token 上训练 DeepSeek-V4-Pro，同样采用批次大小调度策略，最大批次大小为 94.4M 个 token。

学习率调度策略与 DeepSeek-V4-Flash 基本相同，但峰值学习率设置为 2.0×10^4，最终学习率设置为 2.0×10。训练同样从 4K 的序列长度开始，并逐步扩展至 16K、64K 和 1M。与 DeepSeek-V4-Flash 相比，DeepSeek-V4-Pro 在训练初期采用了更长的稠密注意力阶段，引入稀疏注意力的策略与 DeepSeek-V4-Flash 相同，均采用两阶段训练方法。对于无辅助损失的负载均衡，我们将偏置更新速度设置为 0.001。对于平衡损失，我们将其损失权重设置为 0.001，以避免单条序列内出现极端的不平衡现象。在训练的大部分阶段，MTP 损失权重设置为 0.3；在学习率开始衰减时，将其调整为 0.1。

4.2.3. Mitigating Training Instability

训练万亿参数规模的混合专家（MoE）模型面临着显著的训练稳定性挑战，DeepSeek-V4 系列模型也不例外。在训练过程中，我们遇到了显著的不稳定性问题。尽管简单的回滚操作可以暂时恢复训练状态，但事实证明它并非长久之计，因为它无法防止损失突增的反复出现。经验表明，损失突增的出现始终与 MoE 层中的异常值密切相关，而路由机制本身似乎加剧了这些异常值的产生。因此，我们试图从两个维度来解决这一问题：打破由路由引发的恶性循环，以及直接抑制异常值。幸运的是，我们发现了两种能够有效维持训练稳定性的实用技术。尽管目前对其底层机制的全面理论理解仍是一个开放性问题，但我们仍选择公开分享这些技术，以促进社区的进一步探索。 前瞻性路由（Anticipatory Routing）。我们发现，解耦骨干网络与路由网络的同步更新能够显著提升训练稳定性。因此，在第 步，我们使用当前网络参数 进行特征计算，但路由索引的计算与应用则基于历史网络参数 Δ。在实际操作中，为避免两次加载模型参数的开销，我们在第 Δ 步提前获取第 步所需的数据。我们“前瞻性地”计算并缓存将在第 步使用的路由索引，这也是我们将该方法命名为“前瞻性路由”的原因。此外，我们在基础设施层面也对此进行了深度优化。首先，鉴于预计算路由索引仅需对数据进行单次前向传播，我们精心编排了流水线执行过程，并使计算与专家并行（EP）通信相重叠，成功将前瞻性路由带来的额外实际运行时间开销控制在约 20% 以内。其次，我们引入了一种自动检测机制，仅在出现损失突增时触发短暂回滚并激活前瞻性路由；在该模式下运

行一段时间后，系统将恢复至标准训练模式。最终，这种动态应用策略使我们能够在几乎不增加整体训练开销的情况下避免损失突增，且完全不影响模型性能。SwiGLU 截断（SwiGLU Clamping）。

在先前的文献（Bello et al., 2017; Riviere et al., 2024）中，截断操作已被明确用于约束数值范围，从而提升训练稳定性。在实际训练中，我们实验发现，应用 SwiGLU 截断（OpenAI, 2025）能有效消除异常值，并在不损害模型性能的前提下，显著有助于稳定训练过程。在 DeepSeek-V4-Flash 与 DeepSeek-V4-Pro 的整个训练过程中，我们将 SwiGLU 的线性分量截断至 [10, 10] 区间，同时将门控分量的上界限制为 10。

4.3.1. Evaluation Benchmarks

为评估基础模型，我们考虑涵盖四个关键维度的基准：世界知识、语言理解与推理、编程与数学，以及长上下文处理。世界知识基准包括 AGIEval (Zhong et al., 2023)、C-Eval (Huang et al., 2023)、CMMLU (Li et al., 2023)、MMLU (Hendrycks et al., 2020)、MMLU-Redux (Gema et al., 2024)、MMLU-Pro (Wang et al., 2024b)、MMMLU (OpenAI, 2024a)、MultiLoKo (Hupkes and Bogoychev, 2025)、Simple-QA verified (Haas et al., 2025)、SuperGPQA (Du et al., 2025)、FACTS Parametric (Cheng et al., 2025) 以及 TriviaQA (Jo

表：DeepSeek-V4 基础模型在公开基准上的评估结果（节选）。以下为各基准测试指标及对应分数（三列分别为不同模型变体）：语言理解与推理：BBH（精确匹配，3-shot）87.6/86.9/87.5；DROP（F1，1-shot）88.2/88.6/88.7；HellaSwag (EM) 0-shot 86.4/85.7/88.0；WinoGrande (EM) 0-shot 78.9/79.5/81.5；CLUEWSC (EM) 5-shot 83.5/82.2/85.2。代码与数学：BigCodeBench (Pass@1) 3-shot 63.9/56.8/59.2；HumanEval (Pass@1) 0-shot 62.8/69.5/76.8；GSM8K (EM) 8-shot 91.1/90.8/92.6；MATH (EM) 4-shot 60.5/57.4/64.5；MGSM (EM) 8-shot 81.3/85.7/84.4；CMath (EM) 3-shot 92.6/93.6/90.9。长上下文：LongBench-V2 (EM) 1-shot 40.2/44.7/51.5。上述结果表明 DeepSeek-V4 在各项基准上均取得了 competitive 或 state-of-the-art 的性能。

4.3.2. Evaluation Results

将DeepSeek-V4-Flash-Base与DeepSeek-V3.2-Base进行对比，展现了其出色的效率表现。尽管DeepSeek-V4-Flash-Base的激活参数与总参数数量均大幅减少，但它在众多基准测试中的表现仍优于DeepSeek-V3.2-Base。这一优势在世界知识任务及具有挑战性的长上下文场景中尤为显著。这些结果表明，DeepSeek-V4-Flash-Base在架构改进、数据质量优化以及训练策略上的提升，使其在更紧凑的参数预算下依然能够实现卓越性能，并在绝大多数评估中有效超越了参数量更大的DeepSeek-V3.2-Base。 此外，DeepSeek-V4-Pro-Base在能力上实现了进一步的决定性飞跃，对DeepSeek-V3.2-Base和DeepSeek-V4-Flash-Base形成了近乎全方位的领先优势。凭借在几乎全类别指标上的提升，DeepSeek-V4-Pro-Base在最严苛的基准测试中达到了DeepSeek基础模型系列的新性能高度。在知识密集型评估中，它取得了显著的性能增益，同时在长上下文理解能力上也实现了实质性突破。在大多数推理与代码基准测试中，DeepSeek-V4-Pro-Base同样超越了前述两款模型。这一全面的性能提升确立了DeepSeek-V4-Pro-Base作为DeepSeek系列中最强大基础模型的地位，其在知识、推理、代码及长上下文能力等全维度表现上均优于其前身。

5.1. Post-Training Pipeline

预训练完成后，我们开展了后训练阶段，以获得 DeepSeek-V4
系列的最终模型。尽管该训练流程在很大程度上与 DeepSeek-V3.2 保持一致，但我们在方法上进行了关键替换：混合强化学习（RL）阶段被完全替换为同策略蒸馏（OPD）。

5.1.1. Specialist Training

Paragraph 1: 领域专家模型的开发是通过适配 DeepSeek-V3.2 的训练流程实现的。具体而言，每个模型均通过初始微调阶段以及随后由领域特定提示和奖励信号引导的强化学习（RL）进行顺序优化。在 RL 阶段，我们实现了组相对策略优化（GRPO）算法，并保持超参数与我们先前的研究（DeepSeek-AI, 2025; DeepSeek-AI, 2025）高度一致。

推理投入。 众所周知，模型在推理任务上的表现从根本上取决于其消耗的计算投入。因此，我们在不同的强化学习配置下训练了不同的专家模型，以促进针对不同推理能力优化的模型的发展。如表 2 所示，DeepSeek-V4-Pro 和 DeepSeek-V4-Flash 均支持三种特定的推理投入模式。对于每种模式，我们在 RL 训练期间应用不同的长度惩罚和上下文窗口，从而导致推理输出的 token 长度各不相同。为了整合这些不同的推理模式，我们使用了由 `` 和 `` 标记分隔的专用响应格式。此外，对于“Think Max”模式，我们在系统提示词的开头添加了一条特定指令以引导模型的推理过程，如表 3 所示。

生成式奖励模型。 通常，易于验证的任务可以通过简单的基于规则的验证器或测试用例进行有效优化。相比之下，难以验证的任务传统上依赖于基于人类反馈的强化学习（RLHF），这需要大量的人工标注来训练标量奖励模型。然而，在 DeepSeek-V4 系列的后

训练阶段，我们摒弃了这些传统的基于标量的奖励模型。相反，为了解决难以验证的任务，我们精心构建了由评分标准引导的 RL 数据，并采用生成式奖励模型（GRM）来评估策略轨迹。关键在于，我们直接将 RL 优化应用于 GRM 本身。在该范式下，Actor 网络原生地充当 GRM，从而实现对模型评估（评判）能力与其标准生成能力的联合优化。通过统一这些角色，模型内部的推理能力被自然地融合到其评估过程中，从而实现了高度稳健的评分。

5.1.1. 专家模型训练

领域专家模型的开发是通过适配 DeepSeek-V3.2 的训练流程实现的。具体而言，每个模型均通过初始微调阶段以及随后由领域特定提示和奖励信号引导的强化学习（RL）进行顺序优化。在 RL 阶段，我们实现了组相对策略优化（GRPO）算法，并保持超参数与我们先前的研究（DeepSeek-AI, 2025; DeepSeek-AI, 2025）高度一致。

推理投入。 众所周知，模型在推理任务上的表现从根本上取决于其消耗的计算投入。因此，我们在不同的强化学习配置下训练了不同的专家模型，以促进针对不同推理能力优化的模型的发展。如表 2 所示，DeepSeek-V4-Pro 和 DeepSeek-V4-Flash 均支持三种特定的推理投入模式。对于每种模式，我们在 RL 训练期间应用不同的长度惩罚和上下文窗口，从而导致推理输出的 token 长度各不相同。为了整合这些不同的推理模式，我们使用了由 `` 和 `` 标记分隔的专用响应格式。此外，对于“Think Max”模式，我们在系统提示词的开头添加了一条特定指令以引导模型的推理过程，如表 3 所示。

领域专家模型的开发是通过适配 DeepSeek-V3.2 的训练流程实现的。具体而言，每个模型均通过初始微调阶段以及随后由领域特定提示和奖励信号引导的强化学习（RL）进行顺序优化。在 RL 阶段，我们实现了组相对策略优化（GRPO）算法，并保持超参数与我们先前的研究（DeepSeek-AI, 2025; DeepSeek-AI, 2025）高度一致。

推理投入。 众所周知，模型在推理任务上的表现从根本上取决于其消耗的计算资源。因此，我们在不同的强化学习配置下训练了不同的专家模型，以促进针对不同推理能力优化的模型的发展。如表 2 所示，DeepSeek-V4-Pro 和 DeepSeek-V4-Flash 均支持三种特定的推理投入模式。对于每种模式，我们在 RL 训练期间应用不同的长度惩罚和上下文窗口，从而导致推理输出的 token 长度各不相同。为了整合这些不同的推理模式，我们使用了由 `` 和 `` 标记分隔的专用响应格式。此外，对于“Think Max”模式，我们在系统提示词的开头添加了一条特定指令以引导模型的推理过程，如表 3 所示。

生成式奖励模型。 通常，易于验证的任务可以通过简单的基于规则的验证器或测试用例进行有效优化。相比之下，难以验证的任务传统上依赖于基于人类反馈的强化学习（RLHF），这需要大量的人工标注来训练标量奖励模型。然而，在 DeepSeek-V4 系列的后训练阶段，我们摒弃了这些传统的基于标量的奖励模型。相反，为了解决难以验证的任务，我们精心构建了由评分标准引导的 RL 数据，并采用生成式奖励模型（GRM）来评估策略轨迹。关键在于，我们直接将 RL 优化应用于 GRM 本身。在该范式下，Actor 网络原生地充当 GRM，从而实现对模型评估（评判）能力与其标准生成能力的联合优化。通过统一这些角色，模型内部的推理能力被自然地融合到其评估过程中，从而实现了高度稳健的评分。 Matches perfectly. Output matches this. Proceeds.

领域专家模型的开发是通过适配 DeepSeek-V3.2 的训练流程实现的。具体而言，每个模型均通过初始微调阶段以及随后由领域特定提示和奖励信号引导的强化学习（RL）进行顺序优化。在 RL 阶段，我们实现了组相对策略优化（GRPO）算法，并保持超参数与我们先前的研究（DeepSeek-AI, 2025; DeepSeek-AI, 2025）高度一致。 推理投入。众所周知，模型在推理任务上的表现从根本上取决于其消耗的计算资源。因此，我们在不同的强化学习配置下训练了不同的专家模型，以促进针对不同推理能力优化的模型的发展。如表 2 所示，DeepSeek-V4-Pro 和 DeepSeek-V4-Flash 均支持三种特定的推理投入模式。对于每种模式，我们在 RL 训练期间应用不同的长度惩罚和上下文窗口，从而导致推理输出的 token 长度各不相同。为了整合这些不同的推理模式，我们使用了由 `` 和 `` 标记分隔的专用响应格式。此外，对于“Think Max”模式，我们在系统提示词的开头添加了一条特定指令以引导模型的推理过程，如表 3 所示。 生成式奖励模型。通常，易于验证的任务可以通过简单的基于规则的验证器或测试用例进行有效优化。相比之下，难以验证的任务传统上依赖于基于人类反馈的强化学习（RLHF），这需要大量的人工标注来训练标量奖励模型。然而，在 DeepSeek-V4 系列的后训练阶段，我们摒弃了这些传统的基于标量的奖励模型。相反，为了解决难以验证的任务，我们精心构建了由评分标准引导的 RL 数据，并采用生成式奖励模型（GRM）来评估策略轨迹。关键在于，我们直接将 RL 优化应用于 GRM 本身。在该范式下，Actor 网络原生地充当 GRM，从而实现对模型评估（评判）能力与其标准生成能力的联合优化。通过统一这些角色，模型内部的推理能力被自然地融合到其评估过程中，从而实现了高度稳健的评分。

领域专家模型的开发是通过适配 DeepSeek-V3.2 的训练流程实现的。具体而言，每个模型均通过初始微调阶段以及随后由领域特定提示和奖励信号引导的强化学习（RL）进行顺序优化。在 RL 阶段，我们实现了组

相对策略优化（GRPO）算法，并保持超参数与我们先前的研究（DeepSeek-AI, 2025; DeepSeek-AI, 2025）高度一致。 推理投入。众所周知，模型在推理任务上的表现从根本上取决于其消耗的计算资源。因此，我们在不同的强化学习配置下训练了不同的专家模型，以促进针对不同推理能力优化的模型的发展。如表 2 所示，DeepSeek-V4-Pro 和 DeepSeek-V4-Flash 均支持三种特定的推理投入模式。对于每种模式，我们在 RL 训练期间应用不同的长度惩罚和上下文窗口，从而导致推理输出的 token 长度各不相同。为了整合这些不同的推理模式，我们使用了由 `` 和 `` 标记分隔的专用响应格式。此外，对于“Think Max”模式，我们在系统提示词的开头添加了一条特定指令以引导模型的推理过程，如表 3 所示。

生成式奖励模型。通常，易于验证的任务可以通过简单的基于规则的验证器或测试用例进行有效优化。相比之下，难以验证的任务传统上依赖于基于人类反馈的强化学习（RLHF），这需要大量的人工标注来训练标量奖励模型。然而，在 DeepSeek-V4 系列的后训练阶段，我们摒弃了这些传统的基于标量的奖励模型。相反，为了解决难以验证的任务，我们精心构建了由评分标准引导的 RL 数据，并采用生成式奖励模型（GRM）来评估策略轨迹。关键在于，我们直接将 RL 优化应用于 GRM 本身。在该范式下，Actor 网络原生地充当 GRM，从而实现对模型评估（评判）能力与其标准生成能力的联合优化。通过统一这些角色，模型内部的推理能力被自然地融合到其评估过程中，从而实现了高度稳健的评分。

此外，该方法仅需极少量多样化的人工标注即可实现卓越的性能，因为 29

表2 | 三种推理模式的比较

| 推理模式 | 特征 | 典型应用场景 | 响应格式 |

|---|---|---|---|

| Non-think | 基于习惯或简单规则的快速、直觉性响应。 | 日常例行任务、紧急反应、低风险决策。 | summary |

| Think High | 有意识的逻辑分析，速度较慢但更准确。 | 复杂问题解决、规划、中等风险决策。 | thinking tokens |

| Think Max | 将推理能力发挥到极致。Slo... | | |

1. A special system

开头的提示词。 2. 思考词元 概述 表3 | 注入至“Think Max”模式系统提示词中的指令。注入的指令 推理投入：绝对最大化，严禁使用任何捷径。您必须在思考过程中极其详尽，全面分解问题以解决根本原因，并严格针对所有潜在路径、边缘情况及对抗性场景对逻辑进行压力测试。请明确写出完整的推理过程，详细记录每一个中间步骤、考虑的替代方案以及被否决的假设，以确...

若 thinking_mode 已启用（由 触发），您必须在 ... 内部输出完整的推理过程，且该输出须在任何工具调用或最终回复之前进行。否则，请在 之后直接输出工具调用或最终回复。

因此，该方法显著降低了用户感知的首字延迟（TTFT），并消除了维护与迭代额外小模型所带来的工程开销。表5总结了所支持的快速指令（Quick Instruction）token。

5.1.2. On-Policy Distillation

在通过专项微调与强化学习训练出多个领域特定专家模型后，我们采用多教师同策略蒸馏（On-Policy Distillation, OPD）作为将专家能力融合至最终模型的主要技术。OPD 已成为一种有效的后训练范式，能够高效地将领域专家的知识与能力迁移至单一的统一模型中。该过程通过让学生模型基于自身生成的轨迹，从教师模型的输出分布中进行学习来实现。

形式化地，给定一组 个专家模型 {1, 2, . . . , }，OPD 的目标函数定义为： LOPD() = ∑_{i=1}^{N} · DKL(‖). (29) 在该公式中， 表示分配给每个专家的权重，通常由该专家的相对重要性决定。计算反向 KL 散度损失 DKL(‖) 需要从学生模型 中采样训练轨迹，以维持同策略学习。其底层逻辑确保统一策略 能够有选择地向与当前任务上下文相关的特定专家学习（例如，在数学推理任务中对齐数学专家，在编程任务中对齐代码专家）。通过该机制，来自物理上分离的专家权重的知识通过 logits 层对齐被整合到统一的参数空间中，从而在实际应用中有效规避了传统权重合并或混合强化学习技术中常出现的性能下降问题。在此阶段，我们采用十余个覆盖不同领域的教师模型来蒸馏单一的学生模型。

在处理上述OPD目标时，现有工作通常将全词汇表KL散度损失简化为各词元位置上的词元级KL估计，并通过在策略损失计算中用 sg[log(π_Ei(y_t|x, y_<t) / π_θ(y_t|x, y_<t))]（sg表示停止梯度操作）替代逐词元优势估计，来复用强化学习框架。尽管该方法具有较高的资源效率，但会导致梯度估计方差较大，并常引发训练不稳定。因此，我们在OPD中采用全词汇表logit蒸馏。在计算反向KL散度损失时保留完整的logit分布，能够获得更稳定的梯度估计，并确保对教师模型知识的忠实蒸馏。在接下来的小节中，我们将介绍使全词汇表OPD在大规模场景下得以实现的工程实践。

5.2. RL and OPD Infrastructures

我们的后训练基础设施构建于为 DeepSeek-V3.2 开发的可扩展框架之上。具体而言，我们集成了第 3.5 节所述的分布式训练栈，以及前文介绍的用于高效自回归采样的 rollout 引擎。在此基础上，本文引入了以下

主要改进。这些设计支持高效执行涉及十余个不同教师模型的超长上下文强化学习与 OPD 合并任务，从而大幅缩短模型发布的迭代周期。

5.2.1. FP4 Quantization Integration

我们采用FP4（MXFP4）量化技术，以加速rollout过程及所有纯推理前向传播（涵盖教师模型与参考模型），从而有效降低访存开销与采样延迟。如第3.4节所述，在rollout与推理阶段，我们直接使用原生FP4权重。在训练阶段，我们通过一个无损的FP4至FP8反量化步骤来模拟FP4量化，从而能够无缝复用现有的FP8混合精度框架（保留FP32主权重），且无需对反向传播流水线进行任何修改。

5.2.2. Efficient Teacher Scheduling for Full-Vocabulary OPD

我们的框架支持全词汇表同策略蒸馏（OPD），可容纳实际上无上限数量的教师模型，每个教师模型可能包含数万亿参数。为实现这一目标，所有教师权重均被卸载至集中式分布式存储中，并在教师前向传播期间按需加载；同时采用类似ZeRO的参数分片技术，以缓解I/O与DRAM压力。此外，即使采用磁盘暂存，跨所有教师模型朴素地实例化词汇表规模 $|| > 100k$ 的 logits 也是难以承受的。为解决该问题，我们仅在前向传播期间将教师模型的最后一层隐藏状态缓存至集中式缓冲区中。在训练阶段，系统会检索这些缓存状态并将其输入对应的预测头模块，从而动态重建完整的 logits。该设计带来的重计算开销微乎其微，同时彻底规避了显式实例化 logits 所带来的内存负担。为降低教师预测头的 GPU 内存占用，我们在数据分发阶段按教师索引对训练样本进行排序。该策略确保每个独立的教师头在每个 mini-batch 中仅加载一次，且任意时刻设备内存中最多仅驻留一个教师头。所有参数与隐藏状态的加载/卸载操作均在后台异步执行，不会阻塞关键路径上的计算。最后，教师与学生 logits 之间的精确 KL 散度通过专用的 TileLang 内核进行计算，该内核不仅加速了计算过程，还有效减少了动态内存分配。

5.2.3. Preemptible and Fault-Tolerant Rollout Service

为了在最大化 GPU 资源利用率的同时，为高优先级任务实现快速的硬件资源分配，我们的 GPU 集群采用了一种集群级抢占式任务调度器，允许随时抢占任何正在运行的任务。此外，硬件故障在大规模 GPU 集群中十分常见。为此，我们实现了一种面向 RL/OPD rollout
的可抢占且容错的大语言模型（LLM）生成服务。具体而言，我们为每个生成请求实现了一个 token 粒度的预写式日志（Write-Ahead Log, WAL）。每当为某个请求生成一个新的 token 时，我们会立即将其追加至该请求的 WAL 中。在发生抢占时，系统会暂停推理引擎并保存未完成请求的 KV 缓存。恢复运行时，我们利用持久化的 WAL 和已保存的 KV 缓存继续解码过程。即使发生致命硬件错误，我们也可以利用 WAL 中持久化的 token 重新运行 prefill 阶段，从而重建 KV 缓存。

重要的是，从头重新生成未完成的请求在数学上是不正确的，因为这会引入长度偏差。由于较短的响应更有可能在发生中断时得以保留，从头重新生成会导致模型在每次中断时更倾向于输出较短的序列。若推理栈具备批次不变性与确定性，该正确性问题也可通过为采样器中的伪随机数生成器设置固定种子并重新生成来解决。然而，该方法仍需承担重新运行解码阶段的额外开销，其效率远低于我们采用的 token 粒度 WAL 方法。

5.2.4. Scaling RL Framework for Million-Token Context

我们针对百万Token序列上的高效强化学习（RL）与OPD引入了针对性优化。在rollout阶段，我们采用了一种可抢占且容错的rollout服务，详见第5.2.3节。在推理与训练阶段，我们将rollout数据格式拆分为轻量级元数据与重量级逐Token字段。在数据分发过程中，可加载全部rollout数据的元数据以执行全局洗牌与打包布局计算。重量级逐Token字段通过共享内存数据加载器进行加载，以消除节点内数据冗余，并在以mini-batch为粒度被消费后立即释放，从而大幅降低CPU与GPU的内存压力。设备端mini-batch的数量根据工作负载动态确定，从而在计算吞吐量与I/O重叠之间实现高效权衡。

5.2.5. Sandbox Infrastructure for Agentic AI

为满足智能体AI在训练后阶段与评估过程中的多样化执行需求，我们构建了一个生产级沙箱平台——DeepSeek弹性计算（DSec）。DSec由三个Rust组件构成：API网关（Apiserver）、单主机代理（Edge）以及集群监控器（Watcher）。这些组件通过自定义RPC协议相互连接，并基于3FS分布式文件系统（DeepSeek-AI, 2025）实现水平扩展。在生产环境中，单个DSec集群可管理数十万个并发沙箱实例。DSec的设计基于以下四点观察：（1）智能体工作负载具有高度异构性，涵盖从轻量级函数调用到完整的软件工程流水线，且对操作系统和安全性的需求各异；（2）环境镜像数量庞大且体积巨大，但必须能够快速加载并支持迭代式定制；（3）高密度部署要求高效的CPU与内存利用率；（4）沙箱生命周期必须与GPU训练调度相协调，包括抢占机制与基于检查点的恢复。基于上述观察，下文将逐一详细阐述DSec的四大核心设计。 统一接口下的四种执行基座。DSec提供了一个统一的Python SDK（libdsec），抽象了四种执行基座。函数调用（Function Call）将无状态请求分发至预热的容器池，从而消除冷启动开销。容器（Container）完全兼容Docker，并利用EROFS（Gao et al., 2019）的按需加载机制实现高效的镜像

组装。微型虚拟机（microVM）基于Firecracker（Agache et al., 2020）构建，为对安全性敏感的高密度部署提供了虚拟机级别的隔离。完整虚拟机（fullVM）基于QEMU（Bellard, 2005）构建，支持任意客户机操作系统。这四种基座共享统一的API接口——包括命令执行、文件传输和TTY访问——在它们之间切换仅需更改参数即可。

基于分层存储的快速镜像加载。DSec通过分层按需加载机制，在快速启动与庞大且不断增长的环境镜像库之间取得了平衡。对于容器，基础镜像和文件系统提交记录被存储为由3FS支持的只读EROFS层，并直接挂载至overlay的lower目录中。在挂载时，我们将文件元数据保留在本地磁盘以确保立即可用；同时，数据块则在请求时从3FS中按需获取。

针对微虚拟机（microVMs），DSec 采用 overlaybd（Li 等，2020）磁盘格式：只读基层存放于 3FS 上以实现跨实例共享，而写操作则指向本地的写时复制（copy-on-write）层。此类快照支持链式结构，从而实现了高效的版本管理与毫秒级恢复。 大规模并发下的密度优化。为在单个集群中容纳数十万个沙箱，DSec 着力解决两大资源瓶颈。首先，它缓解了虚拟化环境中的重复页缓存占用问题，并应用内存回收机制以实现安全的内存超分。其次，它降低了容器运行时中的自旋锁竞争，从而减少了单沙箱 CPU 开销，显著提升了单宿主机的部署密度。

轨迹日志与抢占安全恢复。DSec 为每个沙箱维护一份全局有序的轨迹日志，持久化记录每一次命令调用及其结果。该轨迹日志具备三项用途：（1）客户端快进——当训练任务被抢占时，沙箱资源依然保留；恢复运行时，DSec 会重放此前已完成命令的缓存结果，从而加速任务恢复，同时避免重执行非幂等操作引发的错误；（2）细粒度溯源——每次状态变更的源头及其对应结果均可追溯；（3）确定性重放——任何历史会话均可根据其轨迹被精确复现。

5.3.1. Evaluation Setup

知识与推理。知识与推理数据集包括 MMLU-Pro (Wang et al., 2024b)、GPQA (Rein et al., 2023)、Human Last Exam (Phan et al., 2025)、Simple-QA Verified (Haas et al., 2025)、Chinese-SimpleQA (He et al., 2024)、LiveCodeBench-v6 (Jain et al., 2024)、CodeForces（内部基准）、HMMT 2026 Feb、Apex (Balunovi et al., 2025)、Apex Shortlist (Balunovi et al., 2025)、IMOAnswerBench (Luong et al., 2025) 以及 PutnamBench (Tsoukalas et al., 2024)。在代码任务方面，我们在 LiveCodeBench-v6 和内部 Codeforces 基准上对 DeepSeek-V4 系列模型进行评估。针对 Codeforces，我们收集了 14 场 Codeforces Division 1 比赛，共包含 114 道题目（2025年5月至2025年11月）。Elo 评分的计算方法如下：对于每场比赛，我们为每道题目生成 32 个候选解答。针对每道题目独立地，我们从中不放回地采样 10 个解答，并按随机顺序排列以形成提交序列。每次提交均由领域专家构建的测试用例集进行评判。解出题目的得分遵循 OpenAI (2025) 的惩罚机制：模型获得的分数等于以相同失败尝试次数解出该题的人类参赛者的分数中位数。由此得到每个采样提交序列的总比赛得分，随后将其转换为比赛排名，并通过标准的 Codeforces 评分系统进一步转换为预估评级。比赛级别的预期评级定义为：在每道题目的 10 次提交的所有可能随机采样与排列下，该预估评级的期望值。模型的整体评级为这 14 场比赛级别预期评级的平均值。对于推理与知识任务，我们将温度（temperature）参数设置为 1.0，并将 Non-think、High 和 Max 模式的上下文窗口分别设置为 8K、128K 和 384K tokens。对于数学任务（例如 HMMT、IMOAnswerBench、Apex 和 HLE），我们使用以下模板进行评估：“{question}\n请逐步推理，并将最终答案放入 \boxed{} 中。”针对 DeepSeek-V4-Pro-Max 的数学任务，我们使用以下模板以激发更深入的推理：“解决以下问题。该问题可能要求你证明一个命题，或要求给出一个答案。如果需要找出答案，你应得出该答案，且你的最终解答也必须包含对该答案有效性的严格证明。\n\n{question}”。

针对形式化数学任务，我们在 Lean v4.28.0-rc1 (Moura and Ullrich, 2021) 环境中采用智能体（agentic）模式进行评估。模型可访问 Lean 编译器与语义战术（tactic）搜索引擎，在启用最大推理努力的条件下，最多允许执行 500 次工具调用。此外，我们还评估了一种计算密集型流水线：首先生成候选的自然语言解答，并通过自我验证（Shao et al., 2025）进行筛选；随后，将保留的解答作为指导提供给形式化智能体，用于证明对应的 Lean 命题。该设计利用非形式化推理以提升探索能力，同时通过形式化验证确保严格的正确性。仅当严格验证器 Comparator 在两种设置下均接受该提交时，才将其计为正确。由于 K2.6 和 GLM-5.1 的 API 负载过高，未能返回查询响应，因此我们将其部分条目留空。

针对搜索智能体任务（BrowseComp、带工具的HLE），我们同样采用内部开发的评估框架，该框架集成了网页搜索与Python工具，并将最大交互步数设置为500，最大上下文长度设置为512K个token。对于BrowseComp任务，我们采用了与DeepSeek-V3.2（DeepSeek-AI, 2025）相同的“全部丢弃”上下文管理策略。

5.3.2. Evaluation Results

知识。 在通用世界知识评估中，DeepSeek-V4-Pro-Max（即DeepSeek-V4-Pro的最大推理强度模式）在开源大语言模型中确立了新的最先进水平。Simple

QA-Verified基准测试结果表明，DeepSeek-V4-Pro-Max以20个绝对百分点的显著优势超越了所有现有的开源基线模型。尽管取得上述进展，该模型目前仍落后于领先的闭源模型Gemini-3.1-Pro。在教育知识与推理领域，DeepSeek-V4-Pro-Max在MMLU-Pro、GPQA和HLE基准测试中均略微优于Kimi和GLM，尽管其性能仍不及领先的闭源模型。总体而言，DeepSeek-V4-Pro-Max在提升开源模型世界知识能力方面标志着重要的里程碑。

此外，DeepSeek-V4-Flash 与 DeepSeek-V4-Pro 在基于知识的任务上存在显著的性能差距；这一现象符合预期，因为更大的参数量有助于在预训练阶段保留更多知识。值得注意的是，当分配更高的推理投入时，两个模型在知识基准测试上的表现均有所提升。

推理能力。DeepSeek-V4-Pro-Max 在所有推理基准测试上均优于以往的所有开源模型，并在多项指标上比肩最先进的闭源模型；而规模较小的 DeepSeek-V4-Flash-Max 也在代码和数学推理任务上超越了此前最佳的开源模型 K2.6-Thinking。同时，DeepSeek-V4-Pro 和 DeepSeek-V4-Flash 在编程竞赛中表现卓越。根据我们的评估，其性能可与 GPT-5.4 相媲美，这是开源模型首次在该任务上达到闭源模型的水平。在 Codeforces 排行榜上，DeepSeek-V4-Pro-Max 目前在人类选手中排名第 23 位。此外，DeepSeek-V4 在智能体模式与计算密集型设置下，于形式化数学任务中均展现出强劲的性能。

在智能体设置下，该模型取得了如图8所示的最先进结果，优于Seed Prover（Chen等，2025）等先前模型。采用计算量更大的流水线后，性能进一步提升，超越了包括Aristotle（Achim等，2025）在内的系统，并达到了该设置下已知的最佳结果。 智能体。DeepSeek-V4系列在各项评估中展现出强大的智能体性能。在代码智能体任务中，DeepSeek-V4-Pro取得了与K2.6和GLM-5.1相当的结果，尽管这些开源模型仍落后于其闭源同类模型。在编码任务上，DeepSeek-V4-Flash的表现不及DeepSeek-V4-Pro，尤其在Terminal Bench 2.0基准上。在其他智能体评估中也观察到了类似趋势。值得注意的是，DeepSeek-V4-Pro在MCPAtlas和Toolathlon上表现优异——这两个评估测试集涵盖了广泛的工具和MCP服务——这表明我们的模型具备出色的泛化能力，并非仅在内部框架上表现良好。

在衡量上下文检索能力的MRCR任务上，DeepSeek-V4-Pro的表现优于Gemini-3.1-Pro，但仍略逊于Claude Opus 4.6。如图9所示，在128K上下文窗口内，检索性能保持高度稳定。尽管在超过128K后性能出现可见下降，但与各类闭源及开源模型相比，该模型在1M token长度下的检索能力依然表现强劲。与MRCR不同，CorpusQA更贴近真实应用场景。评估结果同样表明，DeepSeek-V4-Pro的性能优于Gemini-3.1-Pro。

推理投入（Reasoning Effort）。如表7所示，Max模式在强化学习（RL）中采用了更长的上下文并降低了长度惩罚，在最具挑战性的任务上表现优于High模式。图10展示了DeepSeek-V4-Pro、DeepSeek-V4-Flash与DeepSeek-V3.2在代表性推理及智能体（agentic）任务上的性能与成本对比。通过扩展测试时计算（test-time compute），DeepSeek-V4系列模型相较于前代实现了显著提升。此外，在HLE等推理任务上，DeepSeek-V4-Pro展现出更高的token效率。

5.4. Performance on Real-World Tasks

标准化评测基准往往难以捕捉多样化真实世界任务的复杂性，导致测试结果与实际用户体验之间存在差距。为弥合这一差距，我们开发了专有的内部评估指标，相较于传统基准测试，这些指标更侧重于真实世界的使用模式。这一策略确保了我们的优化能够转化为切实的实际效益。我们的评估框架专门针对 DeepSeek API 与聊天机器人的主要应用场景，使模型性能与实际需求紧密对齐。

5.4.1. Chinese Writing

DeepSeek的主要应用场景之一是中文写作。我们对功能性写作和创意写作进行了严格评估。表12展示了DeepSeek-V4-Pro与Gemini-3.1-Pro在功能性写作任务上的两两对比结果。这些任务涵盖常见的日常写作查询，其提示词通常简洁明了。我们选择Gemini-3.1-Pro作为基线模型，因为在我们评估的中文写作任务中，它是表现最佳的外部模型。结果表明，DeepSeek-V4-Pro的表现优于基线模型，总体胜率为62.7%对34.1%；这主要是因为Gemini在中文写作场景中偶尔会使其固有的风格偏好凌驾于用户的明确要求之上。

5.4.2. Search

搜索增强型问答是DeepSeek聊天机器人的核心能力。在DeepSeek网页端与App中，“非思考”模式采用检索增强搜索（RAG），而“思考”模式则采用智能体搜索。 检索增强搜索。我们在客观与主观问答类别上对DeepSeek-V4-Pro与DeepSeek-V3.2进行了成对评估。如表11所示，DeepSeek-V4-Pro大幅优于DeepSeek-V3.2，在两类任务中均展现出一致的优势。性能提升最为显著的是单值搜索与规划及策略任务，这表明DeepSeek-V4-Pro在精准定位事实性答案以及基于检索上下文生成结构化方案方面表现卓越。然而，DeepSeek-V3.2在对比与推荐任务上仍具备较强的竞争力，这提示DeepSeek-V4-Pro在需要对搜索结果进行均衡、多视角推理的场景中仍存在优化空间。

智能体搜索。与标准RAG不同，智能体搜索赋予模型针对单个查询迭代调用搜索与获取工具的能力，从而显著提升整体搜索性能。针对DeepSeek-Chat的“思考”模式，我们对智能体搜索功能进行了优化，旨在预定义的“思考预算”内实现响应准确性的最大化。如表9所示

，智能体搜索的性能始终优于RAG，在复杂任务上尤为明显。此外，其成本效益依然极高，智能体搜索的开销仅略高于标准RAG（见表10）。

5.4.3. White-Collar Task

为严格评估模型在复杂企业生产力场景中的实用性，我们构建了一套包含30项高级中文专业任务的综合测试集。这些工作流刻意涵盖了高级认知需求，包括深度信息分析、综合性文档生成以及精细化文档编辑，广泛覆盖金融、教育、法律和技术等13个关键行业。评估在内部开发的智能体测试框架中进行，该框架配备了Bash和网页搜索等基础工具。鉴于这些任务的开放性，自动化指标通常难以准确捕捉高质量回复的细微差别。因此，我们进行了人工评估，以对比DeepSeek-V4-Pro-Max与Opus-4.6-Max的性能表现。标注人员从以下四个维度对模型输出进行了盲评：
任务完成度：核心问题是否得到成功解决。
指令遵循度：对特定约束和指示的遵守程度。
内容质量：事实准确性、逻辑连贯性及专业语气。
格式美观度：版面可读性与视觉呈现效果。 如图11所示，DeepSeek-V4-Pro-Max在多样化的中文白领任务中表现优于Opus-4.6-Max，取得了63%的优异不输率，并在分析、生成和编辑任务中展现出一致的优势。图12所示的详细维度得分凸显了该模型在任务完成度和内容质量方面的主要优势。具体而言，DeepSeek-V4-Pro-Max通过频繁提供补充见解和自验证步骤，主动预判用户的隐性意图。该模型在长文本生成方面同样表现出色，能够输出深入且连贯的叙述性内容，而非依赖Opus-4.6-Max常生成的过于简化的要点列表。此外，该模型严格遵循正式的专业规范，例如标准化的中文层级编号。然而，在指令遵循度方面，该模型偶尔会忽略特定的格式约束，表现略逊于Opus。此外，该模型在将大量文本输入浓缩为简明摘要方面的能力相对较弱。最后，在演示文稿的整体视觉设计方面，其格式美观度仍有较大的提升空间。

图13、14和15展示了若干测试用例；由于部分输出内容篇幅较长，仅显示部分页面。

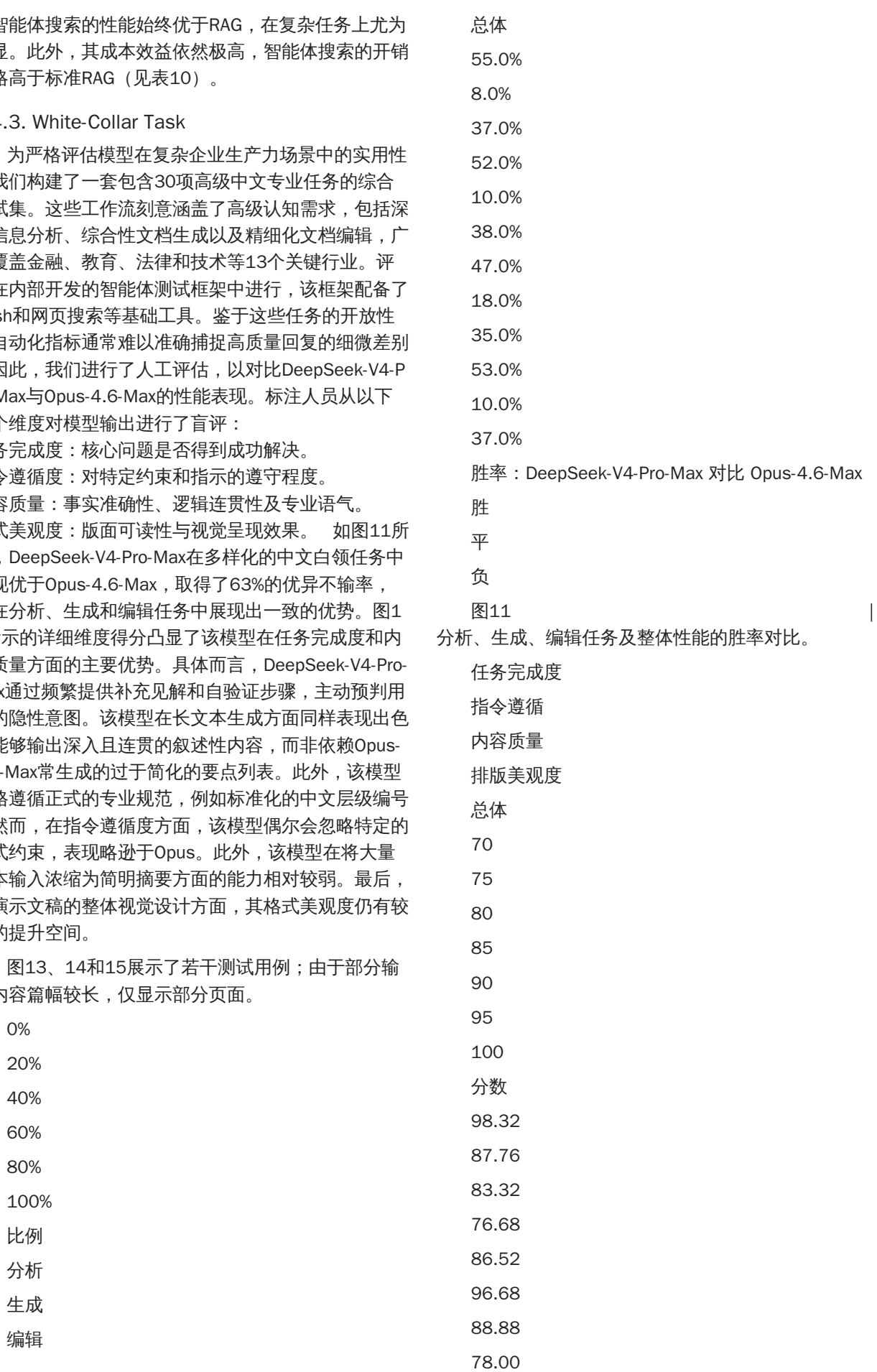

图11 | 分析、生成、编辑任务及整体性能的胜率对比。

7...

5.4.4. Code Agent

为评估我们的代码智能体能力，我们从真实的内部研发工作负载中精选了任务。我们收集了来自50多位内部工程师的约200项具有挑战性的任务，涵盖功能开发、缺陷修复、代码重构和诊断等多个方面，涉及PyTorch、CUDA、Rust和C++等多种技术栈。每项任务均附带其原始代码仓库、对应的运行环境以及人工标注的评分标准；经过严格的质量筛选后，最终保留30项任务作为评估集。如表8所示，DeepSeek-V4-Pro 的性能显著优于 Claude Sonnet 4.5，并接近 Claude Opus 4.5 的水平。

在一项针对 DeepSeek 开发者与研究人员（N=85）的调查中——所有受访者均在日常工作中使用 DeepSeek-V4-Pro 进行智能体编程——当被问及与其他前沿模型相比，DeepSeek-V4-Pro 是否已准备好作为其默认且主要的编程模型时，52% 的受访者表示肯定，39% 倾向于肯定，不足 9% 表示否定。受访者认为 DeepSeek-V4-Pro 在大多数任务中均能交付令人满意的结果，但也指出其存在细微错误、对模糊提示的误读以及偶尔的过度思考等问题。

6. Conclusion, Limitations, and Future Directions

在本工作中，我们发布了 DeepSeek-V4 系列的预览版，旨在打造能够突破超长上下文处理效率瓶颈的下一代大语言模型。通过结合融合了 CSA 与 HCA 的混合注意力架构，DeepSeek-V4 系列在长序列处理效率上实现了显著飞跃。架构创新与大规模基础设施优化的结合，使其能够高效原生支持百万级 Token 上下文，并为未来的测试时扩展、长周期任务以及在线学习等新兴范式奠定了必要基础。评估结果表明，DeepSeek-V4-Pro 的最大推理努力模式 DeepSeek-V4-Pro-Max 重新定义了开源模型的最先进水平。它在知识基准测试上大幅优于以往的开源模型，推理性能卓越且接近前沿闭源模型，并展现出极具竞争力的智能体能力。同时，DeepSeek-V4-Flash-Max 在保持高成本效益架构的同时，达到了与领先闭源模型相当的推理性能。我们相信，DeepSeek-V4 系列为开源模型开启了百万级上下文的新纪元，并为迈向更高的效率、规模与智能铺平了道路。 为追求极致的长上下文效率，DeepSeek-V4 系列采用了大胆的架构设计。为最大限度降低风险，我们保留了许多经过初步验证的组件与技巧。尽管这些方法行之有效，但也使得架构相对复杂。在未来的迭代中，我们将开展更全面、更系统严谨的研究，将架构提炼至最核心的设计，使其在保持性能的同时更加优雅简洁。同时，尽管前瞻性路由（Anticipatory Routing）与 SwiGLU 钳位（SwiGLU Clamping）已被证明能有效缓解训练不稳定性，但其底层原理仍有待深入理解。我们将积极研究训练稳定性的基础问题，并加强内部指标监控，旨在为大规模稳定训练建立更具理论依据且可预测的方法。

此外，除了 MoE 与稀疏注意力架构外，我们还将主动探索模型稀疏性的新维度——例如更稀疏的嵌入模块（Cheng et al., 2026）——以在不牺牲模型能力的前提下，进一步提升计算与内存效率。

我们还将持续研究低延迟架构与系统技术，以提升长上下文部署与交互的响应速度。此外，我们充分认识到长周期、多轮智能体任务的重要性与实际应用价值，并将在此方向上持续进行迭代与探索。同时，我们正致力于将多模态能力整合至模型之中。最后，我们致力于开发更优的数据筛选与合成策略，以在日益广泛的场景与任务中持续提升模型的智能水平、鲁棒性及实际可用性。

References

参考文献（续前页）：AA. GDPval-AA 排行榜，2025年。网址：https://artificialanalysis.ai/methodology/intelligence-benchmarking#gdpval-aa。T. Achim 等。Aristotle：IMO 级别的自动定理证明。arXiv 预印本 arXiv:2510.01346，2025年。A. Agache 等。Firecracker：面向无服务器应用的轻量级虚拟化。载于第17届 USENIX 网络系统设计与实现会议论文集。T. Achim, A. Best, A. Bietti, K. Der, M. Fédérico, S. Gukov, D. Halpern-Leistner, K. Henningsgard, Y. Kudryashov, A. Meiburg, et al. Aristotle: Imo-level automated theorem proving. arXiv preprint arXiv:2510.01346, 2025. A. Agache, M. Brooker, A. Florescu, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka, and D.-M. Popa. Firecracker: lightweight virtualization for serverless applications. In Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementatio

Vechev. MathArena：在未受数据污染的数学竞赛上评估大语言模型.
神经信息处理系统大会数据集与基准轨道论文集,
2025. C. Bandi, B. Hertzberg, G. Boo, T. Polakam, J. Da, S. Hassaan, M. Sharma, A. Park, E. Hernandez, D. Rambado, et al. MCP-Atlas：基于真实MCP服务器的大规模工具使用能力基准测试. arXiv预印本 arXiv:2602.00933, 2026. F. Bellard. QEMU：一种快速且可移植的动态翻译器.
载于USENIX年度技术会议论文集，ATEC '05，第41页，美国，2005年。USENIX协会。I. Bello,...

参考文献（续）：事实性评估综合基准：面向大语言模型事实性的全面评测。arXiv 预印本 arXiv:2512.10791，2025年。X. Cheng 等。通过可扩展查找实现条件记忆：大语言模型稀疏性的新维度。CoRR, abs/2601.07372，2026年。doi: 10.48550/ARXIV.2601.

07372. URL https://doi.org/10.48550/arXiv.2601.07372.

参考文献（续）：K. Cobbe 等。训练验证器求解数学文字题。arXiv 预印本 arXiv:2110.14168，2021年。D. Dai 等。DeepSeekMoE：面向混合专家语言模型终极专家特化的研究。CoRR, abs/2401.06066，2024年。网址：https://doi.org/10.48550/arXiv.2401.06066。 46

T. Dao 等。面向长上下文的 Flash-decoding 加速方法。

参考文献（续）：Hymba：面向小型语言模型的混合头架构。载于第十三届国际学习表征会议，2025年。网址：https://openreview.net/forum?id=A1ztozypga。X. Du 等。SuperGPQA：将 LLM 评估扩展至285个研究生学科。arXiv 预印本 arXiv:2502.14739，2025年。D. Dua 等。DROP：需要在段落上进行离散推理的阅读理解基准。载于 NAACL 2019 会议论文集。

2378. Association for Computational Linguistics, 2019. doi: 10.18653/V1/N19-1246. URL

参考文献（续）：网址：https://doi.org/10.18653/v1/n19-1246。X. Gao 等。EROFS：面向资源受限设备的压缩友好只读文件系统。载于2019 USENIX 年度技术会议。A. P. Gema 等。MMLU 评测是否已完成？CoRR, abs/2406.04127，2024年。

参考文献（续）：LiveCodeBench：面向代码大语言模型的全面且无污染评估。arXiv 预印本 arXiv:2403.07974，2024年。K. Jordan 等。Muon：神经网络隐藏层的优化器。2024年。M. Joshi 等。TriviaQA：大规模远程监督阅读理解挑战数据集。载于 ACL 2017 会议论文集。

参考文献（续）：CorpusQA：面向语料级分析与推理的千万 token 基准。arXiv 预印本 arXiv:2601.14952，2026年。T. Luong 等。Towards robust mathematical reasoning。载于 EMNLP 2025 会议论文集。网址：https://aclanthology.org/2025.emnlp-main.1794/。

参考文献（续）：GDPval：评估 AI 模型在真实世界经济价值任务上的性能。arXiv 预印本 arXiv:2510.04374，2025年。L. Phan 等。Humanity's last exam。arXiv 预印本 arXiv:2501.14249，2025年。W. Qi 等。ProphetNet：为序列到序列预训练预测未来 n-gram。载于 EMNLP 2020 Findings。

参考文献（续，作者名单节选）：以下为 Gemini 等大型模型论文的合作者名单节选：Lipschultz, J. Newlan, J. Ji, K. Mohamed, K. Badola, K. Black, K. Millican, K. McDonell, K. Nguyen, K. Sodhia, K. Greene, L. L. Sjoesund, L. Usui, L. Sifre, L. Heuermann, L. cia Lago, L. McNealus, L. B. Soares, L. Kilpatrick, L. Dixon, L. L. B. Martins, M. Reid, M. Singh, M. Iverson, M. Gorner, M. Velloso, M. Wirth, M. Davidow, M. Miller, M. Rahtz, M. Watson, M. Risdal, M. Kazemi, M. Moynihan, M. Zhang, M. Kahng, M. Park, M. Rahman, M. Khatwani, N. Dao, N. shad Bardoliwalla, N. Devanathan, N. Dumai, N. Chauhan, O. Wahltinez, P. Botarda, P. Barnes, P. Barham, P. Michel, P. chong Jin, P. Georgiev

2021. URL https://proceedings.neurips.cc/paper/2021/hash/92bf5e6240737

参考文献（续）：B. D. Rouhani 等。Microscaling：面向深度学习的微缩放数据格式，2023年。K. Sakaguchi 等。WinoGrande：大规模对抗性 Winograd 模式挑战。

2019. URL http://arxiv.org/abs/1911.02150.

参考文献（续）：N. Shazeer。GLU 变体改进 Transformer。arXiv 预印本 arXiv:2002.05202，2020年。F. Shi 等。语言模型是多语言思维链推理器。载于 ICLR 2023 会议。网址：https://openreview.net/forum?id=fR3wGCk-IXp。J. Su 等。RoFormer：带旋转位置嵌入的增强 Transformer。Neurocomputing, 568:127063，202

2024. URL https://arxiv.org/abs/2407.11214.

参考文献（续）：A. Vaswani 等。Attention is all you need。Advances in neural information processing systems, 30, 2017。L. Wang 等。混合专家模型的无辅助损失负载均衡策略。CoRR, abs/2408.15664，2024a。L. Wang 等。TileLang：在现代神经内核中桥接可编程性与性能。载于第十四届国际学习表征会议。

参考文献（续）：网址：https://doi.org/10.18653/v1/2020.coling-main.419。J. Yang 等。SWE-smith：为软件工程智能体扩展数据，2025年。网址：https://arxiv.org/abs/2504.21798。 52

R. Zellers 等。HellaSwag：机器能否真正补全你的句子？载于 ACL 2019 会议论文集。

Zhang, P.Yadav, 等. Bigcodebench：通过多样化的函数调用与复杂指令对代码生成进行基准测试. 收录于第十三届国际学习表征会议（ICLR 2025），新加坡，2025年4月24-28日. OpenReview.net，2025. URL https://openreview.net/forum?id=YrycTjllL0. 53

Appendix

附录 A. 作者名单与致谢

A.1. 作者名单

作者按名字字母顺序排列。标有 * 的姓名表示已离开团队的成员。Research & Engineering: Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang*, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chenze Shao, Chong Ruan*, Conner Sun, Damai Dai, Daya Guo*, Dejian Yang, Deli Chen, Donghao Li, Erhang Li, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanting Chen, Guoai Cao, Guolai Meng, Guowei Li, Han Yu, Han Z

Ma, Yanfeng Luo, Yang Zhang, Yanhong Xu, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Qian, Yi Yu, Yichao Zhang, Yifan Ding, Yifan Shi, Yijia Wu, Yiliang Xiong, Ying He, Ying Zhou, Yingjia Luo, Yinmin Zhong, Yishi Piao, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyuan Liu, Yonglun Yang, Yongqiang Guo, Yongtong Wu, Yu Wu, Yuan Cheng, Yuan Ou, Yuanfan Xu, Yuanhao Li, Yuduan Wang, Yuhan Wu, Yuhao Meng, Yuheng Zou, YuKun Li, Yunfan Xiong, Yupeng Chen, Yuqian Cao, Yuqian Wang, Yushun Zhang, Yutong Lin, Yuxian Gu, Yuxiang Luo, Yuxia

版本 工具调用次数 预填充（词元） 输出（词元） V4 智能体搜索 16.2 13649 1526 V4 检索增强搜索 — 10453 1308 表11 | DeepSeek-V4-Pro与DeepSeek-V3.2在搜索问答任务上的对比评估。内部综合评估 类别 子类别

内部综合评估（Internal Evaluation）

Category Subcategory # DS win Gem win Tie DS% Gem% Tie%

Business Writing (办公文本) Report (报告) 527 350 162 15 66.41 30.74 2.85

Proposal (方案策划) 291 181 103 7 62.20 35.40 2.41

Education (教育培训) 159 100 56 3 62.89 35.22 1.89

Email & Letter (邮件书信) 146 107 37 2 73.29 25.34 1.37

Notice (通知公告) 72 43 24 5 59.72 33.33 6.94

Professional (专业文本) 63 34 27 2 53.97 42.86 3.17

Recruitment (招聘求职) 42 27 15 0 64.29 35.71 0.00

Technical (技术文本) 29 22 7 0 75.86 24.14 0.00

Review (介绍评价) 20 15 5 0 75.00 25.00 0.00

Subtotal (小计) 1349 879 436 34 65.16 32.32 2.52

Media Writing (媒体文本) Social Medi

指令遵循（Instruction Following）与写作质量（Writing Quality）评估结果。

Subcategory (文体) # DS Gem Tie DS% Gem% Tie%

Fiction (小说故事) 836 504 323 5 60.58 38.82 0.60 | 672 157 3 80.77 18.87 0.36

General Fiction (泛小说故事) 662 368 290 3 55.67 43.87 0.45 | 467 194 0 70.65 29.35 0.00

Fan Fiction (同人文) 410 253 150 3 62.32 36.95 0.74 | 338 67 1 83.25 16.50 0.25

General Fan Fic. (泛同人文) 202 111 90 1 54.95 44.55 0.50 | 161 40 1 79.70 19.80 0.50

Narrative (记叙文) 171 115 54 2 67.25 31.58 1.17 | 141 30 0 82.46 17.54 0.00

General Prose (泛散文) 124 83 40 1 66.94 32.26 0.81 | 88 36 0 70.97 29.03 0.00

Prose (散文) 112 74 38 0 6