DeepSeek-V4.1-Flash: Pushing the Limits of KV Cache Compression

DeepSeek-AI
research@deepseek.com

Abstract

The widespread adoption of long-horizon agents has made model workloads increasingly input-heavy. Although prior work has substantially reduced the cost of long-context computation, prefill remains computationally expensive, and large KV caches continue to strain HBM and SSD capacity and data-transfer bandwidth. Together, these compute, storage, and bandwidth demands constitute the primary bottleneck to further lowering deployment costs. To address this challenge, we introduce DeepSeek-V4.1-Flash, a multimodal Mixture-of-Experts (MoE) model with 552B backbone parameters and support for contexts of up to one million tokens. With its Causal Encoder-Decoder (CED) architecture, the model activates 16B parameters per token during decode but only 8B parameters during prefill, substantially improving cost efficiency for agentic workloads. To push the limits of KV cache compression, DeepSeek-V4.1-Flash combines cross-layer KV cache reuse in Compressed Sparse Attention 2 (CSA2) with FP4 KV caching. These designs reduce its global KV cache footprint (always in HBM) to 890 bytes per token, roughly 1/4 of the corresponding footprint of DeepSeek-V4-Flash. Further, through a dedicated deployment optimization known as SWA Bounded Replay, DeepSeek-V4.1-Flash reduces its persistent KV cache footprint (always on SSD or in host memory) to roughly 1/8 of that of DeepSeek-V4-Flash. Despite its much smaller KV cache footprint, the model delivers substantially better performance than the baseline. In addition, we streamline the DeepSeek-V4 architecture and introduce several efficient architectural extensions. We pretrain DeepSeek-V4.1-Flash on a multimodal corpus comprising 45T tokens and conduct comprehensive post-training, yielding strong performance across diverse text-based and multimodal agentic scenarios. Model checkpoints are available at https://huggingface.co/deepseek-ai/DeepSeek-V4.1-Flash.

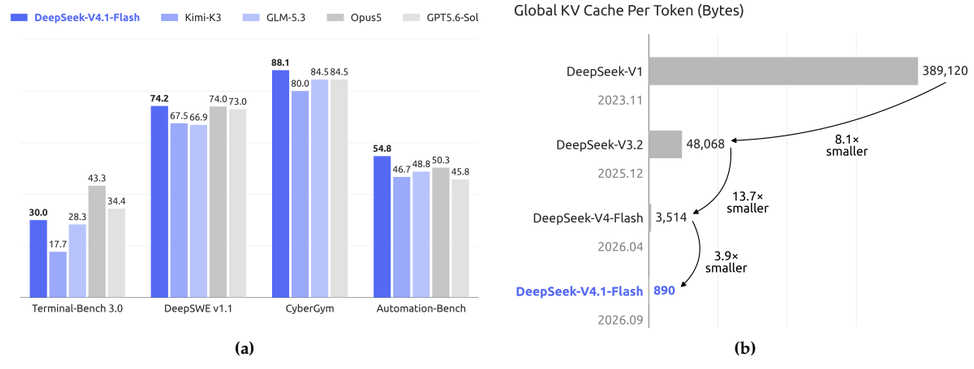


Figure 1 | **(a)** Performance of DeepSeek-V4.1-Flash and its counterparts on agentic benchmarks. **(b)** Global KV cache size per token (in bytes) across generations of DeepSeek models, highlighting DeepSeek's sustained efforts to reduce context memory requirements. DeepSeek-V4.1-Flash achieves approximately 4-fold and 437-fold reductions in per-token global KV cache size relative to DeepSeek-V4-Flash and DeepSeek-V1, respectively.

DeepSeek-V4.1-Flash：探索 KV 缓存压缩的极限

DeepSeek-AI
research@deepseek.com

摘要

长程智能体的广泛应用，使模型工作负载越来越偏重输入处理。尽管已有研究大幅降低了长上下文计算的成本，预填充阶段仍然需要大量计算，而庞大的 KV 缓存也持续对高带宽内存（HBM）和固态硬盘（SSD）的容量及数据传输带宽构成压力。这些计算、存储和带宽需求共同构成了进一步降低部署成本的主要瓶颈。为应对这一挑战，我们推出 DeepSeek-V4.1-Flash：一个主干参数量为 552B、支持最长一百万 token 上下文的多模态混合专家（MoE）模型。借助因果编码器—解码器（CED）架构，该模型在解码阶段每个 token 激活 16B 参数，而在预填充阶段仅激活 8B 参数，显著提高了智能体工作负载的成本效率。为了探索 KV 缓存压缩的极限，DeepSeek-V4.1-Flash 将压缩稀疏注意力 2（CSA2）中的跨层 KV 缓存复用与 FP4 KV 缓存结合起来。这些设计将其全局 KV 缓存占用（始终驻留于 HBM）降至每个 token 890 字节，约为 DeepSeek-V4-Flash 对应占用的四分之一。此外，通过一项名为 SWA 有界重放的专门部署优化，DeepSeek-V4.1-Flash 将持久化 KV 缓存占用（始终存放于 SSD 或主机内存）降至 DeepSeek-V4-Flash 的约八分之一。尽管 KV 缓存占用大幅缩小，该模型的性能仍显著优于基线。此外，我们简化了 DeepSeek-V4 架构，并引入了若干高效的架构扩展。我们在包含 45T token 的多模态语料上对 DeepSeek-V4.1-Flash 进行预训练，并开展全面的后训练，使其在多种基于文本和多模态的智能体场景中均表现出色。模型检查点可在 https://huggingface.co/deepseek-ai/DeepSeek-V4.1-Flash 获取。

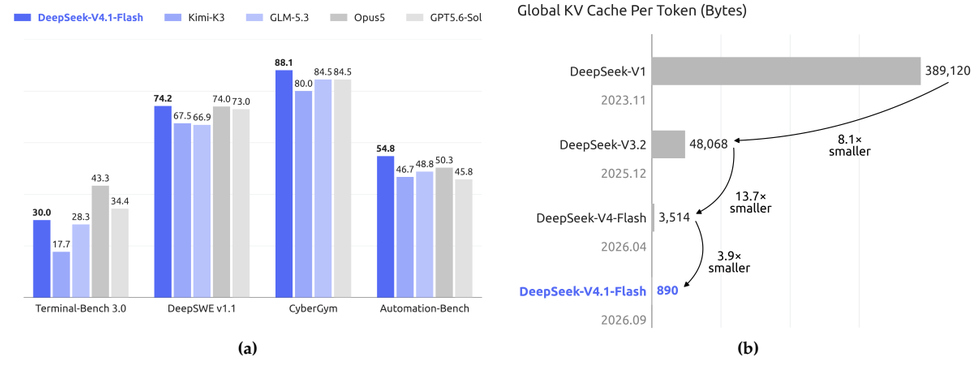


图 1 |（a）DeepSeek-V4.1-Flash 及其同类模型在智能体基准上的表现。（b）历代 DeepSeek 模型每个 token 的全局 KV 缓存大小（以字节计），展示了 DeepSeek 为降低上下文内存需求所做的持续努力。相较于 DeepSeek-V4-Flash 和 DeepSeek-V1，DeepSeek-V4.1-Flash 的每 token 全局 KV 缓存大小分别缩减至约四分之一和四百三十七分之一。

Contents

1 Introduction **4**

2 Architecture **7**

2.1 Overview . . . 7

2.1.1 Multimodal Architecture . . . 8

2.2 Causal Encoder-Decoder (CED) . . . 9

2.3 Compressed Sparse Attention 2 (CSA2) . . . 9

2.3.1 Cross-Layer KV and Index Reuse . . . 10

2.3.2 Hierarchical Sparse Indexer . . . 11

2.4 Efficient Architectural Extensions . . . 12

2.4.1 Single-Pass *m*HC . . . 12

2.4.2 Engram . . . 13

2.4.3 DSpark . . . 13

2.4.4 FP4 Main KV Cache . . . 14

2.5 Optimization . . . 14

3 General Infrastructures **16**

3.1 Training Infrastructure . . . 16

3.1.1 Multimodal Training Infrastructure . . . 16

3.1.2 Attention Sharing Training for CSA2 . . . 17

3.1.3 Engram . . . 18

3.2 Inference System . . . 18

3.2.1 Persistent KV Cache Management . . . 19

3.2.2 SWA Bounded Replay . . . 20

4 Pre-Training **20**

4.1 Data Construction . . . 20

4.2 Pre-Training Setups . . . 21

4.2.1 Model Setups . . . 21

4.2.2 Training Setups . . . 22

4.3 Evaluations . . . 23

4.3.1 Evaluation Benchmarks . . . 23

4.3.2 Evaluation Results . . . 23

5 Post-Training **25**

目录

1 引言 **4**

2 架构 **7**

2.1 概述 . . . 7

2.1.1 多模态架构 . . . 8

2.2 因果编码器—解码器（CED） . . . 9

2.3 压缩稀疏注意力 2（CSA2） . . . 9

2.3.1 跨层 KV 与索引复用 . . . 10

2.3.2 分层稀疏索引器 . . . 11

2.4 高效架构扩展 . . . 12

2.4.1 单次传递 mHC . . . 12

2.4.2 Engram . . . 13

2.4.3 DSpark . . . 13

2.4.4 FP4 主 KV 缓存 . . . 14

2.5 优化 . . . 14

3 通用基础设施 **16**

3.1 训练基础设施 . . . 16

3.1.1 多模态训练基础设施 . . . 16

3.1.2 CSA2 的注意力共享训练 . . . 17

3.1.3 Engram . . . 18

3.2 推理系统 . . . 18

3.2.1 持久化 KV 缓存管理 . . . 19

3.2.2 SWA 有界重放 . . . 20

4 预训练 **20**

4.1 数据构建 . . . 20

4.2 预训练配置 . . . 21

4.2.1 模型配置 . . . 21

4.2.2 训练配置 . . . 22

4.3 评测 . . . 23

4.3.1 评测基准 . . . 23

4.3.2 评测结果 . . . 23

5 后训练 **25**

5.1 Post-Training Pipeline . . . 25
5.1.1 Large-Scale Agent Task Synthesis . . . 25
5.1.2 RL in Synthesized Tasks . . . 26
5.1.3 Running Agents at Massive Scale: DSec . . . 27
5.1.4 Controllable Reasoning Effort in RL . . . 29
5.2 Asynchronous Post-training Infrastructure . . . 30
5.2.1 Overall Workflow . . . 30
5.2.2 Mitigating Length Bias and Off-Policy Effects . . . 31
5.2.3 Performance Optimization . . . 31
5.2.4 Large-Scale On-Policy Distillation . . . 31
5.3 Evaluation . . . 32
5.3.1 Evaluation Setup . . . 32
5.3.2 Evaluation Results . . . 33
5.3.3 Performance across reasoning efforts . . . 34
5.3.4 Performance across agent scaffolds . . . 34
5.3.5 Multi-Agent . . . 35

6 Conclusion, Limitations, and Future Directions **37**

A Author List **46**

B Evaluation Details **47**
B.1 Scaffold Configurations . . . 47
B.2 Reasoning Efforts across Scaffolds . . . 48
B.3 Detailed Results of Reasoning Benchmarks across Reasoning Efforts . . . 48

C Exponential Token Penalty in Reasoning Effort Control **49**

5.1 后训练流程 . . . 25
5.1.1 大规模智能体任务合成 . . . 25
5.1.2 合成任务中的强化学习 . . . 26
5.1.3 超大规模智能体运行：DSec . . . 27
5.1.4 强化学习中的可控推理强度 . . . 29
5.2 异步后训练基础设施 . . . 30
5.2.1 整体工作流程 . . . 30
5.2.2 缓解长度偏差与离策略效应 . . . 31
5.2.3 性能优化 . . . 31
5.2.4 大规模在线策略蒸馏 . . . 31
5.3 评测 . . . 32
5.3.1 评测设置 . . . 32
5.3.2 评测结果 . . . 33
5.3.3 不同推理强度下的表现 . . . 34
5.3.4 不同智能体运行框架下的表现 . . . 34
5.3.5 多智能体 . . . 35

6 结论、局限与未来方向 **37**

A 作者名单 **46**

B 评测细节 **47**
B.1 运行框架配置 . . . 47
B.2 不同运行框架下的推理强度 . . . 48
B.3 不同推理强度下推理基准的详细结果 . . . 48

C 推理强度控制中的指数 token 惩罚 **49**

1. Introduction

Applications of long-horizon agents have expanded rapidly in recent years, making ultra-long-context processing an increasingly important model workload. Supporting such workloads requires not only efficient long-sequence processing, but also the persistent storage, reuse, and transfer of large KV caches. KV cache management has therefore become a foundational capability for model deployment, while introducing substantial challenges across computation, storage, and communication. Prior advances in sparse attention (DeepSeek-AI, 2025, 2026b) have significantly reduced the computational cost of long-sequence processing, making persistent storage and data movement increasingly prominent bottlenecks.

Specifically, DeepSeek-V4 (DeepSeek-AI, 2026b) combines a global attention branch spanning the full context with local Sliding-Window Attention (SWA). The global branch maintains global KV, comprising main KV and indexer K, while SWA maintains local KV states. For a fixed window size, SWA KV storage is bounded independently of sequence length. For sufficiently long sequences, global KV therefore dominates the **runtime KV** footprint, which is constrained by HBM capacity. In addition, certain KV are persisted for prefix reuse, referred to as **persistent KV** caches, which are constrained by SSD and host memory capacity. I/O and interconnect bandwidth also limit cache migration and loading. Together, these constraints limit serving throughput, increase deployment costs, and ultimately hinder the deployment and adoption of agents over longer task horizons and across broader application scenarios.

Further reducing the KV cache footprint is therefore critical to alleviating storage and communication bottlenecks and lowering the cost of long-context serving. To this end, we develop DeepSeek-V4.1-Flash, a multimodal Mixture-of-Experts (MoE) model designed for more aggressive KV cache compression. DeepSeek-V4.1-Flash has 552B backbone parameters, natively supports multimodal inputs, and accommodates contexts of up to one million tokens. We adopt a Causal Encoder-Decoder (CED) architecture, in which decoder global KV is projected from the final encoder hidden states. This design enables the model to activate 8B parameters per token during prefill and 16B during decode, which is particularly cost-effective for input-heavy agentic scenarios. Despite being considerably larger than DeepSeek-V4-Flash, DeepSeek-V4.1-Flash requires only approximately 1/4 as much runtime KV cache storage and 1/8 as much persistent KV cache storage at the same sequence length. Moreover, DeepSeek-V4.1-Flash delivers better overall performance than DeepSeek-V4-Flash.

This level of KV cache compression is achieved through joint optimizations in model architecture, cache precision, and deployment strategy. Conceptually, DeepSeek-V4 can be viewed as an SWA-based local-processing backbone augmented with compressed global context. This perspective motivates us to focus on simplifying the global branch while largely preserving the local attention design. At the architectural level, we design Compressed Sparse Attention 2 (CSA2), which applies cross-layer reuse to global KV (including main KV and indexer K) and Top-K indices to substantially reduce KV cache storage. CSA2 has three statically assigned modes: Full, Reindex, and Reuse. Full Mode generates global KV and performs indexing. Reindex Mode reuses the global KV from a preceding layer, and uses its own indexer Q to rescore the shared indexer K and select fresh Top-K indices. Reuse Mode reuses both global KV and the Top-K indices in a preceding layer, and directly performs sparse attention. In all three modes, each layer retains its own global Q and SWA KV. Sharing global KV and indexer K reduces duplicated cache storage. In addition, different from DeepSeek-V4 that employs the Compressed Sparse Attention (CSA)–Heavily Compressed Attention (HCA) hybrid architecture, DeepSeek-V4.1-Flash uses pure CSA2. At the cache-precision level, we use FP4 global KV caches during training with only marginal performance degradation. Together, CSA2 and FP4 KV

1. 引言

近年来，长程智能体的应用迅速扩展，使超长上下文处理成为日益重要的模型工作负载。支持此类工作负载，不仅需要高效处理长序列，还需要持久存储、复用和传输庞大的 KV 缓存。因此，KV 缓存管理已成为模型部署的一项基础能力，同时也给计算、存储和通信带来重大挑战。此前稀疏注意力方面的进展 (DeepSeek-AI, 2025, 2026b) 已显著降低长序列处理的计算成本，使持久存储和数据搬移日益成为突出的瓶颈。

具体而言，DeepSeek-V4 (DeepSeek-AI, 2026b) 将覆盖完整上下文的全局注意力分支与局部滑动窗口注意力（SWA）结合起来。全局分支维护全局 KV，其中包括主 KV 和索引器 K；SWA 则维护局部 KV 状态。对于固定的窗口大小，SWA KV 的存储量具有与序列长度无关的上界。因此，当序列足够长时，全局 KV 会主导运行时 KV 占用，而这一占用受 HBM 容量限制。此外，部分 KV 会被持久保存以供前缀复用，这些缓存称为持久化 KV 缓存，其规模受 SSD 和主机内存容量限制。I/O 和互连带宽也会限制缓存迁移与加载。这些约束共同限制服务吞吐量、增加部署成本，并最终阻碍智能体在更长任务跨度和更广泛应用场景中的部署与采用。

因此，进一步缩小 KV 缓存占用，对于缓解存储和通信瓶颈、降低长上下文服务成本至关重要。为此，我们开发了 DeepSeek-V4.1-Flash：一个旨在实现更激进 KV 缓存压缩的多模态混合专家（MoE）模型。DeepSeek-V4.1-Flash 的主干参数量为 552B，原生支持多模态输入，并可容纳最长一百万 token 的上下文。我们采用因果编码器—解码器（CED）架构，其中解码器的全局 KV 由

编码器最终的隐藏状态投影得到。这一设计使模型在预填充阶段每个 token 激活 8B 参数，在解码阶段激活 16B 参数，对输入密集型智能体场景尤其具有成本效益。尽管 DeepSeek-V4.1-Flash 的规模显著大于 DeepSeek-V4-Flash，但在相同序列长度下，其运行时 KV 缓存存储需求仅约为后者的四分之一，持久化 KV 缓存存储需求仅约为后者的八分之一。此外，DeepSeek-V4.1-Flash 的总体性能也优于 DeepSeek-V4-Flash。

这种程度的 KV 缓存压缩，来自对模型架构、缓存精度和部署策略的联合优化。从概念上看，DeepSeek-V4 可以被理解为一个以 SWA 为基础、辅以压缩全局上下文的局部处理主干。这一视角促使我们将重点放在简化全局分支上，同时基本保留局部注意力设计。在架构层面，我们设计了压缩稀疏注意力 2（CSA2），对全局 KV（包括主 KV 和索引器 K）以及 Top-K 索引进行跨层复用，从而大幅减少 KV 缓存存储。CSA2 包含三种静态分配的模式：Full、Reindex 和 Reuse。Full 模式生成全局 KV 并执行索引。Reindex 模式复用前面某一层的全局 KV，并使用自身的索引器 Q 对共享的索引器 K 重新评分，选出新的 Top-K 索引。Reuse 模式同时复用前面某一层的全局 KV 和 Top-K 索引，直接执行稀疏注意力。在这三种模式中，每一层都保留自身的全局 Q 和 SWA KV。共享全局 KV 与索引器 K 可以减少重复的缓存存储。此外，DeepSeek-V4 采用压缩稀疏注意力（CSA）与高度压缩注意力（HCA）的混合架构，而 DeepSeek-V4.1-Flash 完全采用 CSA2。在缓存精度层面，我们在训练中使用 FP4 全局 KV 缓存，性能仅出现轻微下降。CSA2 与 FP4 KV

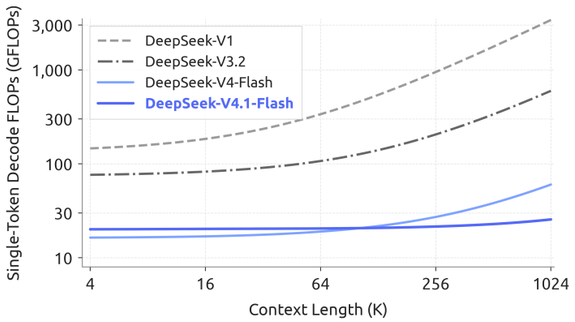


Figure 2 | Single-token Decode FLOPs versus context length across generations of DeepSeek models. We account for compute precision by weighting BF16, FP8, and FP4 operations by 1, 0.5, and 0.25, respectively. DeepSeek-V4.1-Flash maintains nearly constant Decode FLOPs as context length increases, substantially reducing the computational cost of long-context scenarios.

caching reduce global KV cache storage to approximately 1/4 of that of DeepSeek-V4-Flash, as shown in Figure 1(b). At the deployment level, DeepSeek-V4.1-Flash, like DeepSeek-V4, uses Sliding-Window Attention (SWA) in every layer. In DeepSeek-V4, we use a hybrid strategy to balance the storage cost of persisting SWA KV caches against the computation required for exact reconstruction. Exact reconstruction requires replaying the most recent $L \times n_{\text{win}}$ tokens, where L is the number of layers and n_{win} is the SWA window size. In DeepSeek-V4.1-Flash, we introduce SWA Bounded Replay, which approximately reconstructs the required SWA KV states by replaying only the most recent n_{win} tokens. Our experiments show that this incurs only negligible performance degradation. This finding establishes a new storage–computation trade-off, allowing us to avoid persisting SWA KV cache to SSD while incurring a small amount of prefill recomputation. With SWA Bounded Replay, the persistent KV cache footprint is further reduced to approximately 1/8 of that of DeepSeek-V4-Flash. Together, these optimizations greatly ease pressure on HBM and SSD capacity, reduce deployment costs, and pave the way for deployment at a larger scale.

Complementing CED and CSA2, we further streamline the original DeepSeek-V4 architecture. Additionally, we upgrade the original *m*HC (Xie et al., 2026) design to Single-Pass *m*HC, with an accompanying Mega-*m*HC deployment kernel that halves activation memory traffic relative to the original four-kernel implementation. Furthermore, we integrate the Engram (Cheng et al., 2026b) conditional memory module to strengthen model capabilities. We also introduce the DSpark (Cheng et al., 2026a) speculative decoding architecture to improve decoding efficiency through semi-autoregressive draft generation and confidence-scheduled verification. With all the architectural improvements combined, the single-token Decode FLOPs of DeepSeek-V4.1-Flash remain nearly constant across context lengths. Figure 2 shows that extending the context length 256-fold, from 4K to 1M, increases its Decode FLOPs by only 1/4, significantly less than the growth observed for DeepSeek-V4-Flash.

To fully realize the KV cache compression benefits of these architectural designs and further improve training and inference efficiency, we systematically co-optimize the training infrastructure and inference system for DeepSeek-V4.1-Flash, ensuring efficient and scalable large-scale multimodal training and long-context deployment. Training infrastructure supports disaggregated vision-encoder execution, balanced image sharding for long sequences, and cross-stage shared-state management for attention reuse. The inference system implements Encoder and De-

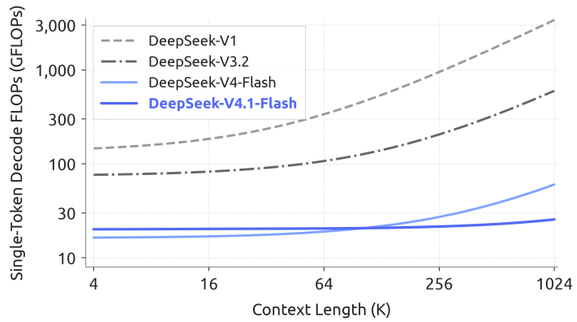


图 2 历代 DeepSeek 模型的单 token 解码 FLOPs 随上下文长度的变化。我们分别以 1、0.5 和 0.25 的权重计算 BF16、FP8 和 FP4 运算，以计入计算精度的影响。随着上下文长度增加，DeepSeek-V4.1-Flash 的解码 FLOPs 几乎保持不变，显著降低了长上下文场景的计算成本。

缓存共同将全局 KV 缓存存储量降至 DeepSeek-V4-Flash 的约四分之一，如图 1（b）所示。在部署层面，DeepSeek-V4.1-Flash 与 DeepSeek-V4 一样，在每一层都使用滑动窗口注意力（SWA）。在 DeepSeek-V4 中，我们采用混合策略，在持久保存 SWA KV 缓存的存储成本与精确重建所需的计算量之间取得平衡。精确重建需要重放最近的 $L \times n_{\text{win}}$ 个 token，其中 L 是层数，n_{win} 是 SWA 窗口大小。在 DeepSeek-V4.1-Flash 中，我们引入 SWA 有界重放，仅重放最近的 n_{win} 个 token，便可近似重建所需的 SWA KV 状态。实验表明，这只会带来可忽略的性能下降。这一发现建立了一种新的存储与计算权衡：只需少量预填充重计算，就可以避免将 SWA KV 缓存持久保存到 SSD。借助 SWA 有界重放，持久化 KV 缓存占用进一步降至 DeepSeek-V4-Flash 的约八分之一。这些优化共同大幅缓解了 HBM 和 SSD 的容量压力，降低部署成本，并为更大规模的部署铺平道路。

在 CED 与 CSA2 之外，我们进一步简化了原有 DeepSeek-V4 架构。此外，我们将原始 mHC (Xie et al., 2026) 设计升级为单次传递 mHC，并配套提供 Mega-mHC 部署内核，使激活内存流量相较原先的四内核实现减半。同时，我们集成了 Engram (Cheng et al., 2026b) 条件记忆模块，以增强模型能力。我们还引入 DSpark (Cheng et al., 2026a) 推测解码架构，通过半自回归草稿生成和按置信度调度的验证来提高解码效率。综合这些架构改进，DeepSeek-V4.1-Flash 的单 token 解码 FLOPs 在不同上下文长度下几乎保持恒定。图 2 显示，当上下文长度从 4K 扩展到 1M、增加 256 倍时，其解码 FLOPs 仅增加四分之一，增幅显著低于 DeepSeek-V4-Flash。

为了充分实现这些架构设计带来的 KV 缓存压缩收益，并进一步提高训练和推理效率，我们系统性地协同优化了 DeepSeek-V4.1-Flash 的训练基础设施与推理系统，确保大规模多模态训练和长上下文部署高效且可扩展。训练基础设施支持解耦的视觉编码器执行、面向长序列的均衡图像分片，以及用于注意力复用的跨阶段共享状态管理。推理系统实现了编码器和

coder SWA Bounded Replay paths. Further optimizations include communication–computation overlap, sharded Engram embedding tables, and inference kernel fusion. In particular, each CSA2 Reuse Mode layer executes with only 15 kernels during prefill and 11 during decode. We also separate long-lived global KV storage from short-lived encoder SWA KV in host memory, using bounded replay to approximately reconstruct missing encoder SWA states.

During pre-training, we train DeepSeek-V4.1-Flash on a large-scale multimodal corpus comprising 45T tokens. Sparse attention is trained from scratch at a sequence length of 64K, without any dense attention warmup stages. After pre-training, the model possesses native multimodal capabilities and supports contexts of up to one million tokens. In our evaluations, DeepSeek-V4.1-Flash-Base achieves world knowledge, reasoning and coding abilities comparable to DeepSeek-V4-Pro-Base, and delivers 5%–10% improvements on held-out evaluations, using only 1/3 total parameters and 1/4 activated parameters. Together, these results highlight its strong parameter efficiency and reflect improvements in training data quality for real-world deployment.

Building on this base model, we conduct post-training to elicit its reasoning and agentic capabilities. In contrast to the architectural innovations described above, our post-training introduces no algorithmic innovation: the recipe follows the standard paradigm of supervised fine-tuning (SFT) followed by reinforcement learning (RL) and on-policy distillation (OPD), without any modification beyond well-established practice used in DeepSeek-V4 development (DeepSeek-AI, 2026b). All substantive changes lie instead in the data pipeline. We develop large-scale automated pipelines for data synthesis and environment construction, and progressively scale the data, tasks, and rollouts employed during RL, thereby extending the model's capabilities across textual, multimodal, and agentic domains. Figure 1(a) summarizes DeepSeek-V4.1-Flash's performance on core agentic benchmarks. Our evaluation shows that, despite its compact size, DeepSeek-V4.1-Flash exhibits a distinctive capability profile:

- **Reasoning.** The model delivers strong reasoning ability, sustaining high accuracy on reasoning-intensive benchmarks such as mathematics and competitive programming, showing comparable performance with top open-source models, such as Kimi-K3(Team et al., 2026a) and DeepSeek-V4-Pro.
- **Agent.** DeepSeek-V4.1-Flash achieves performance on par with closed-source frontier models across standard agentic benchmarks like Terminal-Bench 2.1 (Merrill et al., 2026), DeepSWE v1.1 (DataCurve, 2026), and AutomationBench (Shepard and Salimans, 2026). It has proven fully capable of handling everyday coding tasks and white-collar workflows. However, a gap with giant models remains on science-oriented agentic tasks, such as Terminal-Bench 4.0 (Marten et al., 2026a), that require expert-level domain knowledge.
- **Multimodal.** Within the multimodal domain, the model surpasses top-tier open-source competitors like Kimi-K3 specifically on benchmarks evaluating visual reasoning and the interpretation of professional charts. Beyond formal metrics, it also exhibits practical utility in real-world visual agentic workflows, such as frontend development and office automation, where it can utilize rendered screen captures for visual inspection and self-correction. Nevertheless, we acknowledge that a distinct overall performance gap remains when compared to giant closed-source systems.

These results indicate that DeepSeek-V4.1-Flash can already match closed-source frontier models on the vast majority of benchmarks, and is capable of completing over 95% of real-world tasks. Meanwhile, its small activation footprint yields low inference latency and serving cost. We therefore believe that DeepSeek-V4.1-Flash offers a favorable trade-off between capability and efficiency, and can serve as a fast, affordable assistant supporting the daily work of a

解码器的 SWA 有界重放路径。进一步的优化包括通信与计算重叠、Engram 嵌入表分片，以及推理内核融合。特别是，每个 CSA2 Reuse 模式层在预填充时仅执行 15 个内核，在解码时仅执行 11 个内核。我们还在主机内存中将长期存放的全局 KV 与短期存放的编码器 SWA KV 分离，并利用有界重放近似重建缺失的编码器 SWA 状态。

在预训练阶段，我们使用包含 45T token 的大规模多模态语料训练 DeepSeek-V4.1-Flash。稀疏注意力从一开始就在 64K 序列长度下进行训练，不经过任何稠密注意力预热阶段。预训练完成后，模型具备原生多模态能力，并支持最长一百万 token 的上下文。在我们的评测中，DeepSeek-V4.1-Flash-Base 的世界知识、推理和编程能力与 DeepSeek-V4-Pro-Base 相当，在留出评测上还取得了 5% – 10% 的提升，而总参数量仅为后者的三分之一，激活参数量仅为后者的四分之一。这些结果共同凸显了模型出色的参数效率，并反映出面向真实部署场景的训练数据质量有所提升。

在这一基础模型之上，我们开展后训练以激发其推理与智能体能力。与前述架构创新不同，我们的后训练没有引入算法创新：训练方案遵循监督微调（SFT）、强化学习（RL）和在线策略蒸馏（OPD）的标准范式，没有超出 DeepSeek-V4 开发中成熟实践 (DeepSeek-AI, 2026b) 的任何修改。所有实质性变化都发生在数据流程中。我们开发了用于数据合成和环境构建的大规模自动化流程，并逐步扩大 RL 中使用的数据、任务和 rollout 的规模，从而将模型能力扩展到文本、多模态和智能体领域。图 1（a）概述了 DeepSeek-V4.1-Flash 在核心智能体基准上的表现。评测显示，尽管规模紧凑，DeepSeek-V4.1-Flash 仍展现出鲜明的能力特征：

- 推理。模型展现出强大的推理能力，在数学、竞争性编程等推理密集型基准上保持较高准确率，性能可与 Kimi-K3 (Team et al., 2026a) 和 DeepSeek-V4-Pro 等顶尖开源模型相媲美。
- 智能体。DeepSeek-V4.1-Flash 在 Terminal-Bench 2.1 (Merrill et al., 2026)、DeepSWE v1.1 (DataCurve, 2026) 和 AutomationBench (Shepard and Salimans, 2026) 等标准智能体基准上，达到与闭源前沿模型相当的性能。它已被证明完全能够处理日常编程任务和白领工作流程。然而，在 Terminal-Bench 4.0 (Marten et al., 2026a) 等需要专家级领域知识的科学类智能体任务上，与超大规模模型仍存在差距。
- 多模态。在多模态领域，模型在评估视觉推理和专业图表理解的基准上，超过了 Kimi-K3 等顶尖开源竞争模型。在正式指标之外，它还在前端开发、办公自动化等真实视觉智能体工作流程中展现出实用价值，能够利用渲染后的屏幕截图进行视觉检查和自我纠正。不过，我们承认，与超大规模闭源系统相比，其整体性能仍存在明显差距。

这些结果表明，DeepSeek-V4.1-Flash 已能在绝大多数基准上匹敌闭源前沿模型，并能够完成超过 95% 的真实世界任务。同时，较小的激活规模带来了较低的推理延迟和服务成本。因此，我们认为 DeepSeek-V4.1-Flash 在能力与效率之间实现了有利的权衡，可以作为一个快速、经济的助手，支持

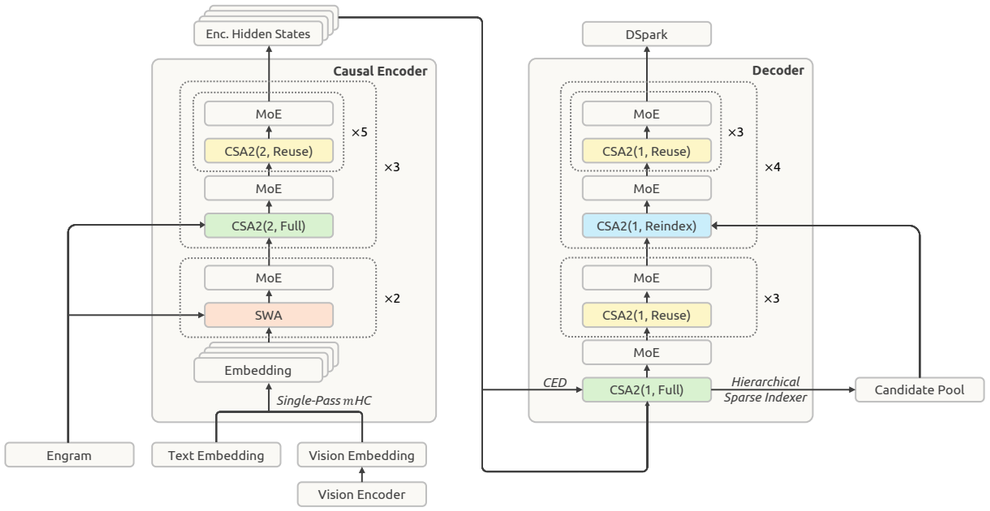


Figure 3 | **Overall architecture of DeepSeek-V4.1-Flash.** The 40-layer network is divided into a causal encoder and a decoder, each with 20 layers. All feed-forward layers use standard DeepSeekMoE. The first two encoder layers use sliding window attention (SWA); the rest use Compressed Sparse Attention 2 (CSA2), with CSA2(ratio, mode) specifying the compression ratio and mode. The model also uses Single-Pass *m*HC, Engram, DSpark, and a Hierarchical Sparse Indexer.

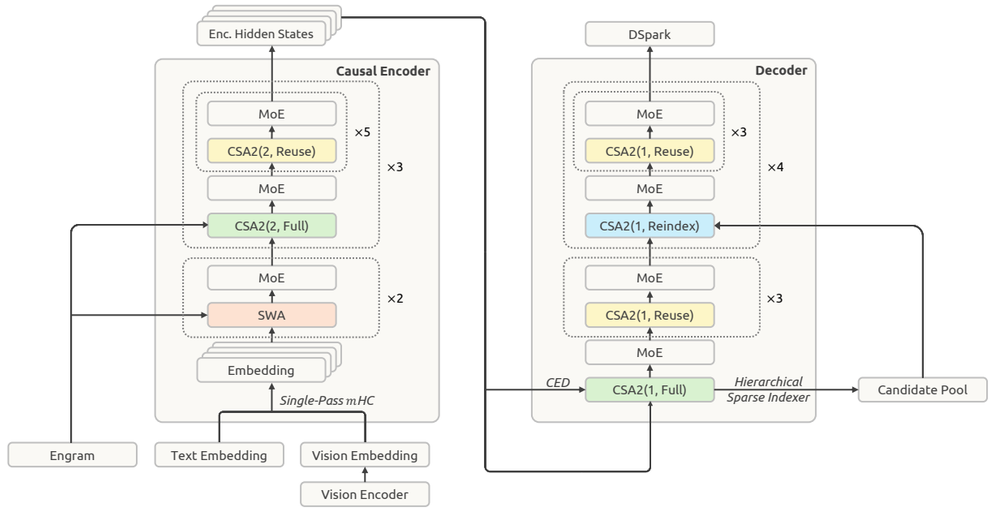


图 3 | DeepSeek-V4.1-Flash 的整体架构。这个 40 层网络分为因果编码器和解码器，两者各有 20 层。所有前馈层均采用标准 DeepSeekMoE。编码器的前两层使用滑动窗口注意力（SWA），其余层使用压缩稀疏注意力 2（CSA2），其中 CSA2(ratio, mode) 指定压缩比和模式。模型还采用单次传递 mHC、Engram、DSpark 和分层稀疏索引器。

broad population of users. In summary, DeepSeek-V4.1-Flash simultaneously improves model intelligence and inference efficiency while reducing deployment costs. It substantially lowers the cost barrier to deploying long-horizon agents at scale and creates new opportunities for their adoption across a broader range of scenarios. DeepSeek-V4.1-Flash also serves as a new starting point for our continued scaling efforts. Building on this foundation, we will pursue the joint scaling of model architecture, pre-training, and post-training to further explore the frontier of model intelligence.

广大用户的日常工作。总之，DeepSeek-V4.1-Flash 在降低部署成本的同时，提高了模型智能水平与推理效率。它显著降低了大规模部署长程智能体的成本门槛，并为在更广泛场景中采用智能体创造了新的机会。DeepSeek-V4.1-Flash 也为我们持续开展规模扩展提供了新的起点。在此基础上，我们将联合扩展模型架构、预训练和后训练，进一步探索模型智能的前沿。

2. Architecture

2. 架构

2.1. Overview

2.1. 概述

DeepSeek-V4.1-Flash is a multimodal mixture-of-experts (MoE) Transformer that takes images and text as input and generates text autoregressively. Its language backbone comprises 40 causal Transformer layers, organized into a 20-layer causal encoder followed by a 20-layer decoder. Each layer incorporates both global attention and sliding window attention (SWA), except for the first two layers, which use SWA only. A vision encoder and an MLP projector convert images into visual embeddings that are processed jointly with text embeddings, with multimodal data incorporated from the start of language-model pre-training. Overall, DeepSeek-V4.1-Flash has 552B backbone parameters and 196B Engram parameters, activating 8B parameters per token during prefill and 16B during decode. Figure 3 illustrates the overall architecture of DeepSeek-V4.1-Flash.

DeepSeek-V4.1-Flash 是一个多模态混合专家（MoE）Transformer，以图像和文本为输入，并以自回归方式生成文本。其语言主干包含 40 个因果 Transformer 层，由一个 20 层因果编码器和随后一个 20 层解码器构成。除仅使用 SWA 的前两层外，每层均结合全局注意力和滑动窗口注意力（SWA）。视觉编码器与 MLP 投影器将图像转换为视觉嵌入，并与文本嵌入一起处理；从语言模型预训练开始，多模态数据就已参与训练。总体而言，DeepSeek-V4.1-Flash 包含 552B 主干参数和 196B Engram 参数，在预填充时每个 token 激活 8B 参数，在解码时激活 16B 参数。图 3 展示了 DeepSeek-V4.1-Flash 的整体架构。

The Causal Encoder–Decoder (CED) architecture and Compressed Sparse Attention 2 (CSA2)

因果编码器—解码器（CED）架构与压缩稀疏注意力 2（CSA2）

address complementary costs of long-context inference. CED constructs the decoder's global key-value (KV) cache from encoder outputs, allowing most prompt tokens to bypass full decoder computation while retaining layer-local sliding-window attention. This nearly halves prefill computation, lowering the cost of processing new or uncached inputs in agentic workloads with growing contexts. CSA2 shares global KV across layers to reduce cache storage and reuses sparse selections to reduce indexing work. In the decoder, a Hierarchical Sparse Indexer restricts later indexers to a candidate pool selected by an earlier indexer, further reducing the number of entries scored per query.

We retain the shared and fine-grained routed experts of DeepSeekMoE (Dai et al., 2024), and introduce modality-specific load balancing (Wang et al., 2024a) for image and text tokens. Single-Pass *m*HC (Xie et al., 2026) revises residual-stream mixing to enable more efficient kernel fusion, and Engram (Cheng et al., 2026b) adds sparsely accessed conditional memory. We omit the MTP module during backbone pre-training and use DSpark (Cheng et al., 2026a) for speculative decoding. We train DSpark separately after the backbone pre-training stage. Additionally, we compress the main KV cache to FP4 to further reduce storage overhead. The following sections describe these components and the corresponding optimization changes.

2.1.1. Multimodal Architecture

The multimodal input pathway comprises a vision encoder and an MLP projector. For each input image, the vision encoder produces a spatial grid of visual features. A 3×3 pixel-unshuffle operation then rearranges each local neighborhood along the channel dimension, reducing the spatial resolution before the MLP projector maps the features to the hidden dimension of the language backbone. Finally, the resulting visual embeddings are inserted at the corresponding image-token positions in the input embedding sequence and processed jointly with text embeddings by the language backbone.

DeepSeek-ViT We train a vision encoder named DeepSeek-ViT from scratch to natively process images at varying resolutions. We build DeepSeek-ViT on the Vision Transformer (Doso- vitskiy et al., 2021) architecture with several modifications. To accommodate inputs of arbitrary resolutions, we replace standard absolute positional embeddings with 2D-RoPE. To align the ViT more closely with LLM design principles, we replace the patch embedding layer's convolution with a linear projection to ensure compatibility with the Muon optimizer. We also adopt RMSNorm (Zhang and Sennrich, 2019) for normalization and SwiGLU (Shazeer, 2020) as the activation function. Before feeding visual features into the LLM, we apply a pixel-unshuffle operation with 3×3 downsampling to reduce the visual token count by a factor of nine, effectively supporting input resolutions up to approximately 1344×1344 pixels.

Multimodal Auxiliary-loss-free Load Balancing for MoEs Image and text tokens exhibit distinct representation distributions and may induce different expert-routing preferences in MoEs. Balancing their aggregate load may therefore obscure modality-specific imbalance. To address this issue, we extend auxiliary-loss-free load balancing (Wang et al., 2024a) by maintaining separate expert-wise correction biases for text and image tokens. During routing, each token uses the correction biases associated with its modality for expert selection, while the original routing scores are retained for weighting the selected expert outputs. After each training step, the two sets of biases are updated independently according to their respective expert loads. This design balances expert utilization within each modality and contributes to stable and efficient multimodal training.

分别应对长上下文推理中相互补充的成本问题。CED 从编码器输出构造解码器的全局键值（KV）缓存，使大多数提示 token 可以跳过完整的解码器计算，同时保留各层局部的滑动窗口注意力。这使预填充计算量近乎减半，降低了上下文不断增长的智能体工作负载中处理新输入或未缓存输入的成本。CSA2 通过跨层共享全局 KV 来减少缓存存储，并复用稀疏选择结果来减少索引工作。在解码器中，分层稀疏索引器将后续索引器的搜索范围限制在较早索引器选出的候选池内，进一步减少每个查询需要评分的条目数。

我们保留了 DeepSeekMoE (Dai et al., 2024) 的共享专家和细粒度路由专家，并针对图像与文本 token 引入按模态区分的负载均衡 (Wang et al., 2024a)。单次传递 mHC (Xie et al., 2026) 修改了残差流的混合方式，以实现更高效的内核融合；Engram (Cheng et al., 2026b) 增加了稀疏访问的条件记忆。我们在主干预训练阶段省略 MTP 模块，并使用 DSpark (Cheng et al., 2026a) 进行推测解码。DSpark 在主干预训练阶段结束后单独训练。此外，我们将主 KV 缓存压缩为 FP4，以进一步降低存储开销。以下各节将介绍这些组件及相应的优化调整。

2.1.1. 多模态架构

多模态输入路径由视觉编码器和 MLP 投影器组成。对于每张输入图像，视觉编码器生成一个视觉特征的空间网格。随后，一个 3×3 的像素重排（pixel-unshuffle）操作沿通道维度重排每个局部邻域，降低空间分辨率，再由 MLP 投影器将这些特征映射到语言主干的隐藏维度。最后，得到的视觉嵌入被插入输入嵌入序列中对应的图像 token 位置，并由语言主干与文本嵌入共同处理。

DeepSeek-ViT。我们从零开始训练名为 DeepSeek-ViT 的视觉编码器，使其原生处理不同分辨率的图像。DeepSeek-ViT 基于 Vision Transformer (Doso- vitskiy et al., 2021) 架构，并作了若干修改。为适应任意分辨率的输入，我们用 2D-RoPE 替换标准的绝对位置嵌入。为了使 ViT 更贴近大语言模型的设计原则，我们将图像块嵌入层中的卷积替换为线性投影，以确保与 Muon 优化器兼容。我们还采用 RMSNorm (Zhang and Sennrich, 2019) 进行归一化，并以 SwiGLU (Shazeer, 2020) 作为激活函数。在将视觉特征送入大语言模型之前，我们应用具有 3×3 下采样的像素重排操作，将视觉 token 数量减少至九分之一，从而有效支持最高约 1344×1344 像素的输入分辨率。

面向 MoE 的多模态无辅助损失负载均衡。图像和文本 token 的表示分布不同，可能在 MoE 中引发不同的专家路由偏好。因此，平衡两者的总负载可能掩盖各模态内部的不均衡。为解决这一问题，我们扩展了无辅助损失负载均衡 (Wang et al., 2024a)，分别为文本和图像 token 维护逐专家的校正偏置。路由时，每个 token 使用其所属模态的校正偏置选择专家，同时保留原始路由分数，用于对所选专家的输出加权。每个训练步骤结束后，两组偏置根据各自的专家负载独立更新。这一设计平衡了每种模态内部的专家利用率，有助于实现稳定、高效的多模态训练。

2.2. Causal Encoder-Decoder (CED)

In agentic workflows, frequent tool calls generate extensive prefill requests, imposing severe computational overhead when KV caches miss. To alleviate this prefill bottleneck, we propose the Causal Encoder-Decoder (CED) architecture, inspired by YoCo (Sun et al., 2024). YoCo reduces prefill computation by allowing the upper half of the layers to directly share the KV cache generated by the lower half. Building upon this concept, CED introduces a series of structural improvements to enhance both the overall KV cache capacity and the computational depth of KV generation. Consequently, CED successfully reduces nearly half of the prefill computation while maintaining performance comparable to the baseline.

For global attention, CED treats the bottom $L/2$ layers of the Transformer as the causal encoder. For the upper half layers (i.e., the decoder, $l > L/2$), the KV entries are not derived from their respective hidden states H_l. Instead, they are projected directly from the hidden state of the $(L/2)$-th layer, $H_{L/2}$, using layer-dependent projection weights (W_l^{KV} and W_l^Z):

$$C_l = H_{L/2} W_l^{KV}, \quad Z_l = H_{L/2} W_l^Z, \quad l > \frac{L}{2}, \tag{1}$$

where C and Z represent the KV entries and their corresponding compression weights, respectively. This design allows CED to compute only the first half of the layers during the prefill phase, acquiring the upper-layer global KV cache with minimal computational cost.

For sliding window attention (SWA), CED maintains the conventional layer-wise computation across all layers. Specifically, for any layer l, the local keys and values are derived directly from the current layer's hidden state H_l. This design effectively increases the computational depth of local KV generation. However, maintaining this layer-wise computation necessitates an SWA replay process. During the prefill phase, computing the SWA KV cache for the decoder requires processing an additional $n_{\text{win}} \times L/2$ tokens (where n_{win} denotes the window size). For multi-turn interactions with short prompts per turn, this computational overhead in the decoder becomes non-negligible. Fortunately, prior work (Chen et al., 2025) has shown that the actual effective receptive field of SWA is much smaller than the theoretical $n_{\text{win}} \times L/2$. Motivated by this observation, we introduce Decoder SWA Bounded Replay, which only prefills the last n_{win} tokens of the prompt for the SWA computation, thereby significantly reducing the computational cost. Further details are provided in Section 3.2.2.

Overall, for a sequence length $N \gg n_{\text{win}}$, CED reduces the prefill complexity from $O(NL)$ to $O(NL/2 + n_{\text{win}} \times L/2) \approx O(NL/2)$, effectively halving the overall computation.

2.3. Compressed Sparse Attention 2 (CSA2)

Serving long contexts requires controlling both KV cache storage and attention computation. These costs can be reduced along three multiplicative dimensions: the entry size, where GQA (Ainslie et al., 2023) reduces the number of KV heads and MLA (DeepSeek-AI, 2024) shares a small latent across heads; the sequence dimension, where every m tokens are compressed into one entry, like CSA and HCA in DeepSeek-V4 (DeepSeek-AI, 2026b); and the layer dimension, where some layers reuse the caches (Brandon et al., 2024) and selections of other layers instead of keeping their own, or are replaced altogether by more efficient layers. Prior work has shown that compression along the layer dimension is effective: IndexCache (Bai et al., 2026) reuses Top-K indices across layers to cut indexer computation; YOIO (Sun et al., 2026b) computes the sparse routing once and shares it across all layers; and HySparse (Gao et al., 2026) lets sparse layers reuse the KV cache of dense layers. However, index reuse alone saves no main

2.2. 因果编码器—解码器（CED）

在智能体工作流程中，频繁的工具调用会产生大量预填充请求，在 KV 缓存未命中时造成严重的计算开销。为缓解这一预填充瓶颈，我们受 YoCo (Sun et al., 2024) 启发，提出因果编码器—解码器（CED）架构。YoCo 允许上半部分层直接共享下半部分层生成的 KV 缓存，从而减少预填充计算。在这一思路之上，CED 引入了一系列结构改进，增强了整体 KV 缓存容量以及 KV 生成的计算深度。因此，CED 在保持与基线相当性能的同时，成功减少了近一半的预填充计算量。

对于全局注意力，CED 将 Transformer 底部的 $L/2$ 层视为因果编码器。对于上半部分层（即解码器，$l > L/2$），KV 条目并非由各层自身的隐藏状态 H_l 推导而来，而是使用各层对应的投影权重，直接从第 $(L/2)$ 层的隐藏状态 $H_{L/2}$ 投影得到，投影权重为（W_l^{KV} 和 W_l^Z）：

$$C_l = H_{L/2} W_l^{KV}, \quad Z_l = H_{L/2} W_l^Z, \quad l > \frac{L}{2}, \tag{1}$$

其中，C 和 Z 分别表示 KV 条目及其对应的压缩权重。这一设计使 CED 在预填充阶段只需计算前半部分层，就能以极低的计算成本获得上层的全局 KV 缓存。

对于滑动窗口注意力（SWA），CED 在所有层上维持常规的逐层计算。具体而言，对于任意一层 l，局部键和值都直接由当前层的隐藏状态 H_l 生成。这一设计有效增加了局部 KV 生成的计算深度。然而，维持这种逐层计算需要执行 SWA 重放过程。在预填充阶段，为解码器计算 SWA KV 缓存，需要额外处理 $n_{\text{win}} \times L/2$ 个 token（其中 n_{win} 表示窗口大小）。对于每轮提示较短的多轮交互，解码器中的这一计算开销不可忽略。幸运的是，已有研究 (Chen et al., 2025) 表明，SWA 的实际有效感受野远小于理论值 $n_{\text{win}} \times L/2$。受这一观察启发，我们引入解码器 SWA 有界重放，在进行 SWA 计算时只对提示末尾的 n_{win} 个 token 做预填充，从而显著降低计算成本。更多细节见第 3.2.2. 节。

总体而言，对于序列长度 $N \gg n_{\text{win}}$，CED 将预填充复杂度从 $O(NL)$ 降至 $O(NL/2 + n_{\text{win}} \times L/2) \approx O(NL/2)$，有效地将总体计算量减半。

2.3. 压缩稀疏注意力 2（CSA2）

长上下文服务需要同时控制 KV 缓存存储与注意力计算成本。这些成本可以沿三个具有乘法效应的维度降低：在条目大小维度，GQA (Ainslie et al., 2023) 减少 KV 头的数量，MLA (DeepSeek-AI, 2024) 则在不同头之间共享一个较小的潜在表示；在序列维度，每 m 个 token 被压缩为一个条目，如 DeepSeek-V4 (DeepSeek-AI, 2026b) 中的 CSA 和 HCA；在层维度，部分层复用其他层的缓存 (Brandon et al., 2024) 和选择结果，而不再保留自身的缓存与选择结果，或者直接被更高效的层替代。已有研究表明，沿层维度压缩是有效的：IndexCache (Bai et al., 2026) 跨层复用 Top-K 索引，以削减索引器计算；YOIO (Sun et al., 2026b) 只计算一次稀疏路由，并在所有层之间共享；HySparse (Gao et al., 2026) 则让稀疏层复用稠密层的 KV 缓存。然而，仅复用索引并不能节省主

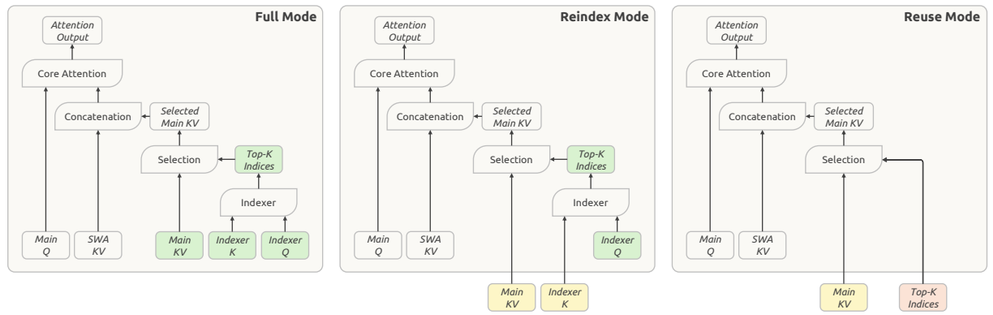


Figure 4 | **Three operating modes of CSA2.** The modes differ in how they obtain main KV, indexer K, and Top-K indices. Green blocks indicate quantities computed in the current layer; yellow blocks indicate main KV and indexer K reused from the most recent Full Mode layer; while red blocks indicate Top-K indices reused from the most recent index-producing (Full or Reindex Mode) layer. All three modes compute main Q and SWA KV in the current layer.

KV storage, network-wide routing sharing limits performance, and hybrid designs still retain full attention layers; more importantly, none of these methods covers all three multiplicative dimensions.

CSA2 exploits the three dimensions jointly: it shares main KV and indexer K across layers and allows layers to reuse Top-K indices, with cache sharing and index reuse decoupled. It combines these reuse strategies with a simplified compressor and a Hierarchical Sparse Indexer that narrows the search domain of subsequent indexing layers in the Decoder.

Similar to CSA, CSA2 includes a lightweight indexer that scores the main KV entries using indexer Q and indexer K and selects the Top-K entries for each query. Each Q attends to the selected entries together with the layer-local sliding-window KV (SWA KV). CSA2 also includes the uncompressed main KV setting as a special case with a compression ratio of 1. Meanwhile, CSA2 simplifies both the compressor and the indexer. In CSA, a compression ratio of m produces each main KV entry from $2m$ original KV cache entries, with overlapping source entries for adjacent compressed entries. It also includes absolute positional embedding to encode the positions of these $2m$ entries during compression. CSA2 removes this overlap and absolute positional embedding. In addition, CSA2 obtains indexer K by projecting main KV entries, replacing CSA's separate compression path from hidden states. Both designs simplify the implementation and increase the training efficiency.

Sections 2.3.1 and 2.3.2 describe the cross-layer reuse strategies and the Hierarchical Sparse Indexer, respectively.

2.3.1. Cross-Layer KV and Index Reuse

Each CSA2 layer is statically assigned one of three modes: Full, Reindex, or Reuse. In all three modes, the layer computes its own query and SWA KV and uses them together with the selected main KV entries to produce a new attention output. The modes differ in how they obtain main KV, indexer K, and Top-K indices. Figure 4 illustrates the three modes.

Full Mode. The layer computes its own main KV and indexer Q, projects indexer K from that main KV, and runs the indexer to produce fresh Top-K indices. It therefore executes the complete CSA2 computation path and has the same component responsibilities as a complete

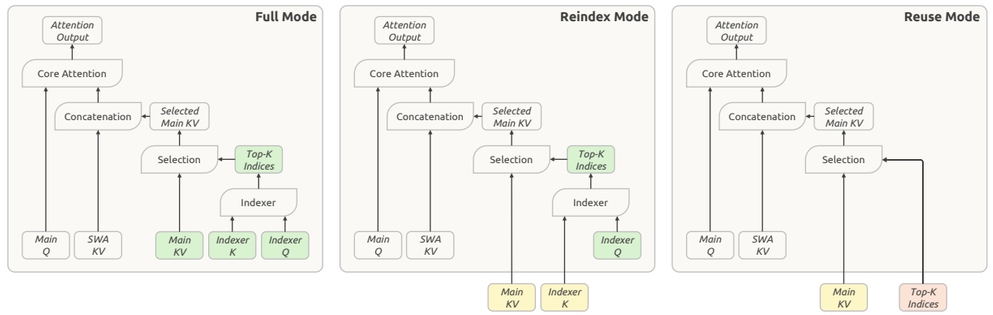


图 4 CSA2 的三种运行模式。各模式在获取主 KV、索引器 K 和 Top-K 索引的方式上有所不同。绿色模块表示当前层计算的量；黄色模块表示从最近的 Full 模式层复用的主 KV 与索引器 K；红色模块表示从最近产生索引的层（Full 或 Reindex 模式）复用的 Top-K 索引。三种模式都在当前层计算主 Q 和 SWA KV。

KV 存储，网络范围的路由共享会限制性能，而混合设计仍然保留完整注意力层；更重要的是，这些方法都没有覆盖全部三个具有乘法效应的维度。

CSA2 联合利用这三个维度：跨层共享主 KV 和索引器 K，并允许各层复用 Top-K 索引，同时将缓存共享与索引复用解耦。它将这些复用策略与简化的压缩器、分层稀疏索引器结合起来，后者能够缩小解码器中后续索引层的搜索范围。

与 CSA 类似，CSA2 包含一个轻量索引器，利用索引器 Q 和索引器 K 对主 KV 条目评分，并为每个查询选择 Top-K 条目。每个 Q 对所选条目以及该层局部的滑动窗口 KV（SWA KV）进行注意力计算。CSA2 也将未压缩的主 KV 配置纳入其中，作为压缩比为 1 的特例。同时，CSA2 简化了压缩器和索引器。在 CSA 中，当压缩比为 m 时，每个主 KV 条目由 $2m$ 个原始 KV 缓存条目生成，相邻压缩条目使用的源条目存在重叠。它还包含绝对位置嵌入，以在压缩过程中编码这 $2m$ 个条目的位置。CSA2 去除了这种重叠和绝对位置嵌入。此外，CSA2 通过投影主 KV 条目来获得索引器 K，替代了 CSA 从隐藏状态单独压缩生成索引器 K 的路径。这两项设计都简化了实现，并提高了训练效率。

第 2.3.1 节和第 2.3.2 节分别介绍跨层复用策略和分层稀疏索引器。

2.3.1. 跨层 KV 与索引复用

每个 CSA2 层都会被静态指定为三种模式之一：Full、Reindex 或 Reuse。在三种模式下，该层都会计算自身的查询和 SWA KV，并将它们与选出的主 KV 条目一起用于产生新的注意力输出。各模式的区别在于如何获得主 KV、索引器 K 和 Top-K 索引。图 4 展示了这三种模式。

Full 模式。该层计算自身的主 KV 和索引器 Q，从主 KV 投影得到索引器 K，并运行索引器生成新的 Top-K 索引。因此，它执行完整的 CSA2 计算路径，各组件承担的职责与 DeepSeek-V4 中一个完整的

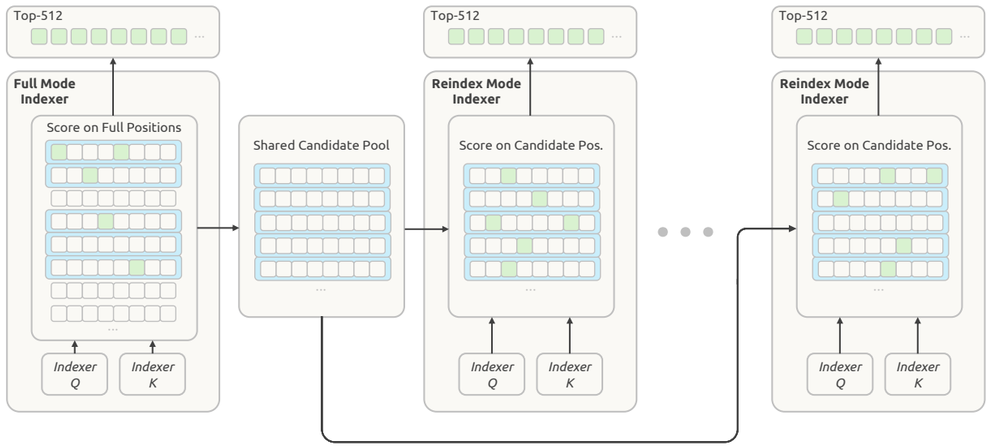


Figure 5 | **Hierarchical Sparse Indexer.** Each square represents a position; green squares mark selected indices, and blue rectangles mark blocks selected based on their maximum indexer scores. The decoder's first CSA2 layer in Full mode selects its own Top-512 indices and builds a shared candidate pool from the selected blocks for subsequent layers. CSA2 layers in Reindex mode then select their Top-512 indices from this pool.

CSA layer in DeepSeek-V4.

Reindex Mode. The layer reuses the most recent available main KV from a preceding layer together with its corresponding indexer K. The indexer computes its own query, rescores the reused keys, and produces fresh Top-K indices. This allows the sparse selection to change across layers while main KV and indexer K remain shared.

Reuse Mode. The layer reuses the most recent available main KV and the latest Top-K indices computed against that main KV by a preceding layer in Full or Reindex Mode. It performs attention using this selection without computing indexer Q or evaluating index scores.

Sharing main KV and indexer K reduces cache storage, while reusing Top-K indices avoids additional indexer computation. Reindex Mode preserves cache sharing while allowing the selected entries to change across layers. When CSA2 is combined with CED, the decoder layer assigned to Full Mode computes its own global KV from the hidden state of the $(L/2)$-th layer, i.e. the last layer of the causal encoder. The Reindex and Reuse Modes are unchanged.

2.3.2. Hierarchical Sparse Indexer

Cross-layer index reuse reduces the number of indexer evaluations, but the remaining indexers still score the full causally visible context. For extremely long contexts, this cost remains a major computational bottleneck. Prior work introduced indexer sparsity by scoring and pruning pooled block representations before token-level indexing (Xu et al., 2026b). We find that in the decoder, information from shallower indexers can naturally be used to restrict the candidates considered by deeper indexers without adding any extra state. We therefore introduce the Hierarchical Sparse Indexer, which is used only in the decoder of CED to reduce this repeated scoring during decode. For each query, the first layer assigned to Full Mode constructs a candidate pool that later re-indexing layers use as their search domain. For a fixed candidate-pool size, this changes the per-query cost of deeper indexers from linear in context length to

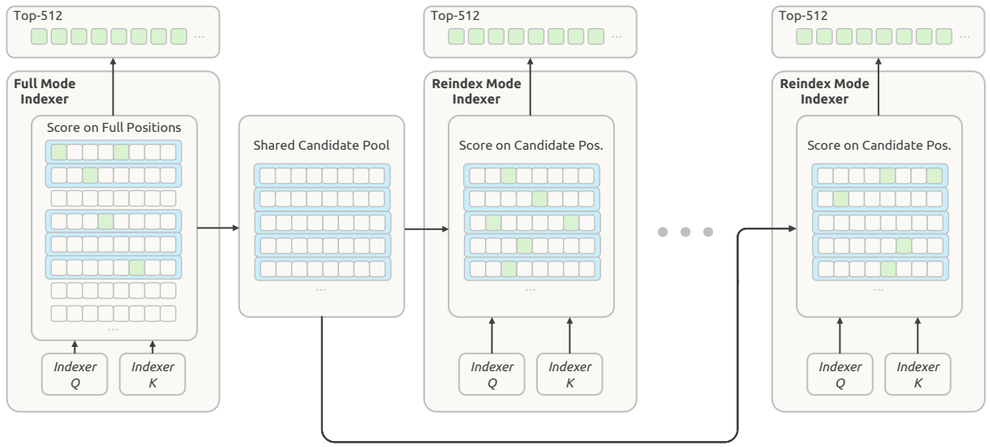


图 5 | 分层稀疏索引器。每个方格代表一个位置；绿色方格标出选中的索引，蓝色矩形标出根据块内最高索引器分数选出的块。解码器中第一个采用 Full 模式的 CSA2 层选择自身的 Top-512 索引，并由选中的块构建供后续层使用的共享候选池。随后，采用 Reindex 模式的 CSA2 层从该候选池中选择自身的 Top-512 索引。

CSA 层相同。

Reindex 模式。该层复用前面层中最近可用的主 KV 及其对应的索引器 K。索引器计算自身的查询，对复用的键重新评分，并生成新的 Top-K 索引。这样，在主 KV 和索引器 K 保持共享的同时，稀疏选择结果可以随层变化。

Reuse 模式。该层复用最近可用的主 KV，以及前面某个 Full 或 Reindex 模式层针对该主 KV 计算出的最新 Top-K 索引。它直接使用这一选择结果进行注意力计算，无需计算索引器 Q 或评估索引分数。

共享主 KV 和索引器 K 可以减少缓存存储，而复用 Top-K 索引则避免了额外的索引器计算。Reindex 模式在保留缓存共享的同时，允许各层选中的条目不同。当 CSA2 与 CED 结合时，被指定为 Full 模式的解码器层从第 $(L/2)$ 层、即因果编码器最后一层的隐藏状态计算自身的全局 KV。Reindex 与 Reuse 模式保持不变。

2.3.2. 分层稀疏索引器

跨层索引复用减少了索引器求值的次数，但其余索引器仍需要对完整的因果可见上下文评分。对于极长上下文，这项成本依然是主要计算瓶颈。已有研究在 token 级索引之前，对池化后的块表示进行评分和剪枝，从而引入索引器稀疏性 (Xu et al., 2026b)。我们发现，在解码器中，较浅层索引器的信息可以自然地用于限制较深层索引器考虑的候选项，而无需增加任何额外状态。因此，我们引入分层稀疏索引器，仅在 CED 的解码器中使用，以减少解码期间的重复评分。对于每个查询，首个被指定为 Full 模式的层会构建一个候选池，后续重新索引的层以此作为搜索范围。当候选池大小固定时，这使较深层索引器每个查询的成本从随上下文长度线性增长变为

constant. The mechanism is training-aware and introduced in post-training: the candidate restriction is applied identically during training and inference, so deeper indexers are optimized under the same search domain they use at inference. Figure 5 illustrates this process.

This first Full Mode layer scores all causally visible main KV positions and produces the Top-K indices for its own attention. It also performs blockwise candidate selection: each block is assigned the maximum index score among its positions, and the blocks with the highest scores are selected. It then collects the positions covered by the selected blocks into a candidate pool larger than the final Top-K set. For example, selecting 2,048 blocks with 8 positions each yields 16,384 candidate positions. This pool defines where later indexers search; the final Top-K selection determines which main KV entries each layer reads.

Subsequent layers in Reindex Mode score only the candidate positions for the corresponding query and select their own Top-K entries within that pool. Layers in Reuse Mode perform no new indexing and use the latest Top-K indices computed against the main KV they reuse. Thus, the candidate pool is shared across indexing layers, while their final selections can differ.

For a fixed candidate-pool size, the number of positions scored per query by each subsequent indexer is bounded independently of context length. The first Full Mode layer still scans the entire causally visible range. Hierarchical indexing therefore reduces the cost of later indexer evaluations while retaining the initial full-range pass.

2.4. Efficient Architectural Extensions

2.4.1. Single-Pass mHC

In DeepSeek-V4, we introduced mHC (Xie et al., 2026), which maintains n residual streams between adjacent Transformer blocks. For each token, we denote these streams by $X_l \in \mathbb{R}^{n\times d}$, where l is the block index and d is the hidden dimension. The streams are updated as follows:

$$X_{l+1} = B_l X_l + C_l\, \mathcal{F}_l(A_l X_l), \qquad (A_l, B_l, C_l) = \mathcal{H}(X_l), \tag{2}$$

where $A_l \in \mathbb{R}^{1\times n}$, $C_l \in \mathbb{R}^{n\times 1}$ and $B_l \in \mathbb{R}^{n\times n}$ are token-wise coefficients predicted from X_l. The coefficient predictor $\mathcal{H}$ includes normalization and projection.

Ideally, the residual transformation between two blocks is a single map from (X_{l-1}, Y_{l-1}) to $(X_l, \hat{X}_l)$, where $\hat{X}_l = A_l X_l$ is the current block input and $Y_{l-1} = \mathcal{F}_{l-1}(\hat{X}_{l-1})$ is the previous block output. Such a map requires $(n+1)d$ reads and $(n+1)d$ writes, giving a lower bound of $(2n+2)d$ on activation memory traffic. In practice, DeepSeek-V4 uses a multi-pass implementation of (2), with three kernels that execute sequentially due to data dependencies:

$$X_l = B_{l-1} X_{l-1} + C_{l-1} Y_{l-1} \qquad \text{Residual update, contraction over } n \tag{3}$$

$$(A_l, B_l, C_l) = \mathcal{H}(X_l) \qquad \text{Coefficients, contraction over } nd \tag{4}$$

$$\hat{X}_l = A_l X_l \qquad \text{Input mixing, contraction over } n \tag{5}$$

The three kernels read $(n+1)d$, nd and nd values respectively and write $(n+1)d$ in total. Including the pre-norm in $\mathcal{F}_l$, the total activation memory traffic is $(4n+4)d$, twice the lower bound.

In $\mathcal{H}$, the normalization weights are folded into the projection weights offline, and the RMS division is applied after the projection. Two of the three stages can therefore share one traversal of the residual: the residual update does not require a reduction across the hidden dimension, so each tile of X_l can be computed and immediately used to accumulate the projection outputs

常数。该机制考虑了训练过程，并在后训练中引入：训练与推理期间应用完全相同的候选限制，因此较深层索引器会在与推理时相同的搜索范围内得到优化。图 5 展示了这一过程。

第一个 Full 模式层对所有因果可见的主 KV 位置评分，并生成用于自身注意力计算的 Top-K 索引。它还执行按块进行的候选选择：每个块的分数取其内部各位置索引分数的最大值，再选出分数最高的块。随后，它将这些选中块覆盖的位置收集到一个候选池中，该池大于最终的 Top-K 集合。例如，选择 2,048 个块、每块包含 8 个位置，就会得到 16,384 个候选位置。该池定义了后续索引器的搜索范围；最终的 Top-K 选择则决定每层读取哪些主 KV 条目。

后续 Reindex 模式层只对相应查询的候选位置评分，并在该池内选出自身的 Top-K 条目。Reuse 模式层不进行新的索引计算，而是使用针对其复用的主 KV 所计算出的最新 Top-K 索引。因此，候选池在各索引层之间共享，而各层最终的选择可以不同。

当候选池大小固定时，后续每个索引器对每个查询所评分的位置数量都有一个与上下文长度无关的上界。第一个 Full 模式层仍然扫描整个因果可见范围。因此，分层索引在保留最初全范围扫描的同时，降低了后续索引器求值的成本。

2.4. 高效架构扩展

2.4.1. 单次传递 mHC

在 DeepSeek-V4 中，我们引入了 mHC (Xie et al., 2026)，它在相邻 Transformer 块之间维护 n 条残差流。对于每个 token，我们用 $X_l \in \mathbb{R}^{n\times d}$ 表示这些流，其中 l 是块索引，d 是隐藏维度。这些流按如下方式更新：

$$X_{l+1} = B_l X_l + C_l\, \mathcal{F}_l(A_l X_l), \qquad (A_l, B_l, C_l) = \mathcal{H}(X_l), \tag{2}$$

其中，$A_l \in \mathbb{R}^{1\times n}$、$C_l \in \mathbb{R}^{n\times 1}$ 和 $B_l \in \mathbb{R}^{n\times n}$ 是根据 X_l 预测的逐 token 系数。系数预测器 $\mathcal{H}$ 包含归一化与投影。

理想情况下，两个块之间的残差变换是一个从 (X_{l-1}, Y_{l-1}) 到 $(X_l, \hat{X}_l)$ 的单一映射，其中 $\hat{X}_l = A_l X_l$ 是当前块的输入，$Y_{l-1} = \mathcal{F}_{l-1}(\hat{X}_{l-1})$ 是前一个块的输出。这样一个映射需要 $(n+1)d$ 次读取和 $(n+1)d$ 次写入，因此激活内存流量的下界为 $(2n+2)d$。在实际实现中，DeepSeek-V4 对式（2）采用多次传递实现，由于数据依赖关系，三个内核需要依次执行：

$$X_l = B_{l-1} X_{l-1} + C_{l-1} Y_{l-1} \qquad \text{Residual update, contraction over } n \tag{3}$$

$$(A_l, B_l, C_l) = \mathcal{H}(X_l) \qquad \text{Coefficients, contraction over } nd \tag{4}$$

$$\hat{X}_l = A_l X_l \qquad \text{Input mixing, contraction over } n \tag{5}$$

这三个内核分别读取 $(n+1)d$、nd 和 nd 个值，总共写入 $(n+1)d$。计入 $\mathcal{F}_l$ 中的前置归一化后，总激活内存流量为 $(4n+4)d$，是理论下界的两倍。

在 $\mathcal{H}$ 中，归一化权重被离线折叠进投影权重，RMS 除法则在投影之后执行。因此，三个阶段中的两个可以共享对残差的一次遍历：残差更新不需要沿隐藏维度做归约，所以 X_l 的每个分块都可以在计算完成后，立即用于累加投影输出

and the sum of squares needed to compute the RMS. Input mixing cannot be fused into this pass because A_l is not available until the reduction over all hidden tiles is complete. It therefore requires a second read of X_l. This second pass can also incorporate input pre-norm. This two-pass implementation would require $(3n+2)d$ activation reads and writes in total, one additional read of X_l compared with the lower bound.

We therefore introduce **Single-Pass *m*HC**, which shifts the input-mixing coefficients by one block, i.e. every block consumes the mixing coefficients produced by the previous one, so that the above dependency disappears:

$$X_{l+1} = B_l X_l + C_l\, \mathcal{F}_l(A_{l-1} X_l), \qquad (A_l, B_l, C_l) = \mathcal{H}(X_l). \tag{6}$$

Input mixing now uses A_{l-1} instead of A_l, so it no longer depends on the coefficients computed from X_l. Each tile of X_l can therefore be used immediately for both input mixing and coefficient prediction, without waiting for the full reduction. Empirically, this shift incurs negligible performance degradation.

For pre-training, we keep the existing multi-kernel implementation, since the shift only changes which mixing coefficients each block applies. For deployment, we fuse residual update, input mixing, and coefficient prediction into a single kernel, **Mega-*m*HC**. The kernel implements *m*HC with $(3n+2)d$ activation reads and writes and Single-Pass *m*HC with $(2n+2)d$ activation reads and writes. Mega-*m*HC processes X_l in tiles along the hidden dimension. Each tile is used to compute the mixed input and to accumulate the quantities needed to predict (A_l, B_l, C_l) for the next block. The kernel also incorporates input pre-norm and FP8 conversion. The residual is thereby read once and written once, attaining the $(n+1)d$ reads and $(n+1)d$ writes of the ideal map and halving the activation memory traffic of our original implementation.

2.4.2. Engram

We augment DeepSeek-V4.1-Flash with Engram (Cheng et al., 2026c), the conditional memory module introduced in our previous work to decouple memorization from computation. We follow the original Engram design—tokenizer compression, multi-head hashing, context-aware gating, and multi-branch integration—with two modifications. First, we omit the short causal convolution because its performance gains do not justify the added complexity in our inference stack. Second, we optimize the Engram embedding with momentum-based update followed by Sinkhorn balancing, as detailed in Section 2.5.

We allocate 196B Engram parameters evenly across two modules. Each module uses N-gram orders $\{2,3,4\}$, with 8 hash heads and a total embedding dimension of 2048 per order. Each head indexes a table of approximately 16M entries, with table sizes chosen to be distinct primes. Both the embedding tables and the key/value projections use FP8 precision. The modules are placed at layers 1 and 14 (zero-indexed) to balance memory usage across training pipeline stages. During inference, deterministic addressing enables embeddings to be prefetched from host memory via background RDMA transfers, with prefetching for the first module overlapping computation in the first Transformer block. Further implementation details for Engram training and inference are discussed in Section 3.1.3.

2.4.3. DSpark

We equip DeepSeek-V4.1-Flash with DSpark (Cheng et al., 2026a), a speculative decoding module that combines semi-autoregressive drafting with confidence-scheduled verification.

以及计算 RMS 所需的平方和。输入混合无法融合进这次传递，因为只有在所有隐藏维度分块的归约完成后，才能得到 A_l。因此，它需要再次读取 X_l。这第二次传递也可以纳入输入的前置归一化。这种两次传递的实现总共需要 $(3n+2)d$ 次激活读取与写入，与理论下界相比多读取一次 X_l。

因此，我们引入单次传递 mHC，将输入混合系数向后移动一个块，也就是每个块使用前一个块产生的混合系数，从而消除上述依赖：

$$X_{l+1} = B_l X_l + C_l\, \mathcal{F}_l(A_{l-1} X_l), \qquad (A_l, B_l, C_l) = \mathcal{H}(X_l). \tag{6}$$

现在，输入混合使用 A_{l-1} 而非 A_l，因此不再依赖于根据 X_l 计算出的系数。这样，X_l 的每个分块都可以立即同时用于输入混合和系数预测，无需等待完整归约。实验表明，这种偏移带来的性能下降可以忽略。

在预训练中，我们保留现有的多内核实现，因为这种偏移只改变了各块所应用的混合系数。在部署中，我们将残差更新、输入混合和系数预测融合进单个内核 Mega-mHC。该内核实现 mHC 时需要 $(3n+2)d$ 次激活读取与写入，实现单次传递 mHC 时需要 $(2n+2)d$ 次激活读取与写入。Mega-mHC 沿隐藏维度对 X_l 分块处理。每个分块都用于计算混合输入，并累加为下一个块预测 (A_l, B_l, C_l) 所需的量。该内核还纳入了输入前置归一化和 FP8 转换。因此，残差只需读取一次、写入一次，达到了理想映射所需的 $(n+1)d$ 次读取与 $(n+1)d$ 次写入，使激活内存流量相较原始实现减半。

2.4.2. Engram

我们为 DeepSeek-V4.1-Flash 增加了 Engram (Cheng et al., 2026c) 条件记忆模块，这是我们在先前工作中引入的、用于将记忆与计算解耦的模块。我们遵循 Engram 的原始设计，包括分词器压缩、多头哈希、上下文感知门控和多分支集成，并作了两项修改。第一，我们省略了短因果卷积，因为它带来的性能收益不足以抵偿其在推理技术栈中增加的复杂性。第二，我们采用基于动量的更新，并在其后进行 Sinkhorn 平衡，以优化 Engram 嵌入，详见第 2.5. 节。

我们将 196B Engram 参数均匀分配给两个模块。每个模块使用 N-gram，其阶数为 $\{2,3,4\}$，每一阶有 8 个哈希头，总嵌入维度为 2048。每个头索引一张约有 16M 个条目的表，各表大小选为互不相同的素数。嵌入表和键值投影均使用 FP8 精度。这些模块被放置在第 1 层和第 14 层（从零开始计数），以平衡各训练流水线阶段的内存使用。推理时，确定性寻址使嵌入能够通过后台 RDMA 传输从主机内存预取，其中第一个模块的预取与第一个 Transformer 块的计算重叠进行。Engram 训练和推理的更多实现细节见第 3.1.3. 节。

2.4.3. DSpark

我们为 DeepSeek-V4.1-Flash 配备 DSpark (Cheng et al., 2026a)，这是一个推测解码模块，将半自回归草稿生成与按置信度调度的验证相结合。

The drafter comprises three Transformer blocks with a sliding attention window of 128 tokens. A single forward pass through these blocks computes base logits for five draft positions in parallel, while a lightweight Markov head models dependencies among the draft tokens. A confidence head predicts per-position conditional acceptance probabilities, which are used to estimate prefix survival probabilities. The scheduler combines these estimates with profiled engine throughput curves to dynamically select the verification length for each request, aiming to maximize expected system-wide token throughput under the current system load.

Unlike the MTP module in DeepSeek-V3 (DeepSeek-AI, 2024), which is trained jointly with the backbone throughout pre-training, DSpark is introduced in a dedicated stage after pre-training. In this stage, we train only DSpark while keeping the backbone frozen. During post-training, we continue to train DSpark alongside the backbone, without propagating gradients from the DSpark objective into the backbone. This keeps DSpark aligned with the evolving policy, enabling it to accelerate both online serving and rollout generation for RL and OPD.

2.4.4. FP4 Main KV Cache

Long-context agent workloads require large per-request KV caches, increasing serving costs. DeepSeek-V4 already uses quantization-aware training (QAT) (Jacob et al., 2018) for FP4 indexer queries and keys, accelerating index computation and reducing the indexer cache size. We adopt the OCP-standard MXFP4 format (Rouhani et al., 2023) to support as many hardware platforms as possible, despite the higher accuracy of alternative formats in our experiments. We now extend QAT to the main KV cache, where FP4 reduces storage rather than accelerates matrix multiplication. Dequantizing cached values before attention allows us to use a more accurate format without requiring native matrix-multiplication support for that format, preserving compatibility across hardware platforms.

Among the approximately four-bit formats evaluated, we select E2M1 with one E4M3 scale per 16 channels, following NVFP4 (Alvarez et al., 2025) but omitting its second-level global scale to balance accuracy and simplicity. Omitting this scale leaves ample dynamic range for the main KV cache: the format supports magnitudes up to $448 \times 6 = 2688$, far above the cache's magnitude bound. In DeepSeek-V4.1-Flash, the largest trained RMSNorm weight magnitude is approximately 1. After RMS normalization, the L2 norm of the 512-channel KV latent is at most approximately $\sqrt{512}$. RoPE preserves this norm, so the maximum absolute value across channels after rotation is also bounded by approximately $\sqrt{512} \approx 22.6$. Besides, the maximum magnitude observed during training is around 10. Therefore, omitting the global scale causes no measurable decrease in accuracy and simplifies the cache layout.

To enable FP4 main KV cache storage in DeepSeek-V4.1-Flash, we introduce QAT during post-training. The non-RoPE and RoPE components use the same quantization format. We quantize the cache after RoPE: quantizing before RoPE yields only a marginal accuracy improvement in our experiments and would introduce additional overhead during decoding. We retain FP8 for the SWA KV cache due to its sensitivity to quantization. Compared with the FP8 main KV cache in DeepSeek-V4, this format nearly halves the storage footprint, both in HBM and when offloaded to SSD.

2.5. Optimization

Building upon the optimization configuration used in DeepSeek-V4, we make some new modifications to better align with the architectural design.

First, we use head-wise Muon, where Query weights are split by head before applying the

草稿模型由三个 Transformer 块组成，滑动注意力窗口为 128 个 token。通过这些块的一次前向传递，即可并行计算五个草稿位置的基础 logits；一个轻量的 Markov 头负责建模草稿 token 之间的依赖关系。置信度头预测各位置的条件接受概率，并据此估计前缀存活概率。调度器将这些估计与通过性能剖析得到的引擎吞吐量曲线结合，为每个请求动态选择验证长度，目标是在当前系统负载下最大化整个系统的预期 token 吞吐量。

DeepSeek-V3（DeepSeek-AI, 2024）中的 MTP 模块在整个预训练过程中与主干联合训练，而 DSpark 是在预训练结束后的专门阶段引入的。在这一阶段，我们只训练 DSpark，并冻结主干。在后训练中，我们继续将 DSpark 与主干一起训练，但不将 DSpark 目标产生的梯度反向传播到主干中。这样可以让 DSpark 与不断演进的策略保持一致，使其既能加速在线服务，也能加速 RL 和 OPD 的 rollout 生成。

2.4.4. FP4 主 KV 缓存

长上下文智能体工作负载需要为每个请求保留庞大的 KV 缓存，从而推高服务成本。DeepSeek-V4 已对 FP4 索引器查询和键采用量化感知训练（QAT）(Jacob et al., 2018)，以加速索引计算并缩小索引器缓存。尽管实验中其他格式的准确性更高，我们仍采用 OCP 标准的 MXFP4 格式（Rouhani et al., 2023），以支持尽可能多的硬件平台。现在，我们将 QAT 扩展到主 KV 缓存；在这里，FP4 的作用是减少存储，而非加速矩阵乘法。在注意力计算之前对缓存值反量化，使我们能够采用更准确的格式，而无需硬件原生支持该格式的矩阵乘法，从而保持跨硬件平台的兼容性。

在评估的近似四位格式中，我们选择 E2M1，并为每 16 个通道配备一个 E4M3 缩放因子。这一设计遵循 NVFP4 (Alvarez et al., 2025)，但省略了其第二级全局缩放因子，以兼顾准确性与简洁性。省略该缩放因子后，主 KV 缓存仍拥有充足的动态范围：这一格式支持的幅值最高为 $448 \times 6 = 2688$，远高于缓存的幅值上界。在 DeepSeek-V4.1-Flash 中，训练得到的 RMSNorm 权重最大幅值约为 1。经过 RMS 归一化后，512 通道 KV 潜在表示的 L2 范数至多约为 $\sqrt{512}$。RoPE 保持这一范数，因此旋转后各通道的最大绝对值也不超过约 $\sqrt{512} \approx 22.6$ 此外，训练期间观察到的最大幅值约为 10。因此，省略全局缩放因子不会导致可测量的准确性下降，并能简化缓存布局。

为了在 DeepSeek-V4.1-Flash 中实现 FP4 主 KV 缓存存储，我们在后训练阶段引入 QAT。非 RoPE 和 RoPE 分量采用相同的量化格式。我们在 RoPE 之后对缓存进行量化：实验中，在 RoPE 之前量化仅带来轻微的准确性提升，却会在解码时引入额外开销。由于 SWA KV 缓存对量化较敏感，我们为其保留 FP8。与 DeepSeek-V4 的 FP8 主 KV 缓存相比，这一格式无论在 HBM 中，还是卸载到 SSD 后，都能将存储占用近乎减半。

2.5. 优化

在 DeepSeek-V4 所用优化配置的基础上，我们进行了一些新的调整，以更好地适配架构设计。

首先，我们使用逐头 Muon：先按注意力头拆分 Query 权重，再应用

Muon update. Here, we briefly discuss the motivation for such a design. By viewing Muon as a preconditioned gradient descent, vanilla Muon uses one preconditioner for all heads, whereas head-wise Muon provides different preconditioners for different heads. This design can better handle the heterogeneity across attention heads (Zhang et al., 2024; Zhang, 2026, Section 3). As a result, we observe that head-wise Muon outperforms vanilla Muon. The empirical advantage of head-wise Muon is also validated in GLM 5 (Zeng et al., 2026) and Kimi-K3 (Team et al., 2026a).

Second, applying Adam to the newly introduced Engram parameters substantially increases the optimizer-state memory footprint. To reduce memory usage during training, we instead optimize the Engram embedding tables, token embedding, and prediction head using a momentum-based update followed by Sinkhorn balancing. Sinkhorn balancing has previously been applied to linear-layer weight matrices in SinkGD (Scetbon et al., 2025); here, we extend it to these large parameter matrices. Like Muon, this approach requires only a momentum buffer while empirically outperforming Adam.

Basic Configurations. We retain AdamW (Loshchilov and Hutter, 2019) for normalization-layer weights and other non-matrix parameters, including biases and scaling factors. We use Muon (Jordan et al., 2024) for the weight matrices of linear transformations in the language-model backbone, the Engram projection layers, and the vision-language projector. We use head-wise Muon for Query and Key weights. We apply decoupled weight decay and Nesterov momentum to Muon (Nesterov, 1983; Liu et al., 2025); normalization-layer weights are also subject to weight decay, whereas biases and scaling factors are not. The Sinkhorn-balanced update also uses Nesterov momentum but does not apply weight decay. During pre-training, we keep the vision encoder frozen until the learning-rate decay stage, while its final normalization layer and the vision–language projector remain trainable. At the onset of learning-rate decay, we unfreeze the vision encoder and optimize it jointly with the LLM with a smaller learning rate.

Sinkhorn-Balanced Updates for Engram / Embedding / Prediction Head. The complete procedure is summarized in Algorithm 1. At a high level, it follows the same workflow as Muon, with Sinkhorn balancing taking the place of Newton–Schulz orthogonalization. We denote the larger matrix dimension by m, which corresponds to the vocabulary size for embedding tables and prediction heads, and denote the hidden dimension by n.

Given the Nesterov momentum update $\widehat{G}_t$, Sinkhorn balancing finds diagonal scaling matrices D_r and D_c such that

$$\Delta_t = \sqrt{n}\, U^{(K)} = \sqrt{n}\, D_r \widehat{G}_t D_c, \qquad \frac{1}{n}\sum_{j=1}^{n} (\Delta_t)_{ij}^2 \approx 1, \qquad \frac{1}{m}\sum_{i=1}^{m} (\Delta_t)_{ij}^2 \approx 1, \tag{7}$$

Thus, the procedure approximately equalizes the row-wise and column-wise RMS of the update matrix. Here, one row corresponds to one token index or n-gram identity; and one column encodes one hidden feature. Sinkhorn balancing exploits this token–feature structure by normalizing along both rows and columns. For numerical stability, rows satisfying $\rho_i \leqslant \tau\bar{\rho}$ are masked. The factor $\sqrt{n}$ converts unit row ℓ_2 norm into unit row-wise RMS. Separately, we adjust the effective learning rate as $\widetilde{\eta}_t = \gamma\eta_t$ to match the update magnitude of Adam. We set $\gamma = 0.18$, which is close to the factor 0.2 used in Moonlight (Liu et al., 2025).

More broadly, Sinkhorn balancing is closely related to optimizers that exploit matrix or tensor axis structure (Shazeer and Stern, 2018; Zhang et al., 2025a; Wen et al., 2025; Glentis et al., 2025; Deng et al., 2026; Yuan et al., 2026; Xu et al., 2026a). For example, Adafactor (Shazeer and

Muon 更新。下面简要说明这一设计的动机。将 Muon 视为一种预条件梯度下降方法时，标准 Muon 对所有注意力头使用同一个预条件器，而逐头 Muon 为不同的头提供不同的预条件器。这一设计能够更好地处理注意力头之间的异质性 (Zhang et al., 2024; Zhang, 2026, 第 3 节）。因此，我们观察到逐头 Muon 优于标准 Muon。逐头 Muon 的经验优势也在 GLM 5 (Zeng et al., 2026) 和 Kimi-K3 (Team et al., 2026a) 中得到验证。

第二，对新引入的 Engram 参数应用 Adam，会显著增加优化器状态的内存占用。为了降低训练时的内存使用，我们改为使用基于动量的更新，并在其后执行 Sinkhorn 平衡，来优化 Engram 嵌入表、token 嵌入和预测头。此前，SinkGD (Scetbon et al., 2025) 已将 Sinkhorn 平衡应用于线性层的权重矩阵；在这里，我们将它扩展到这些大规模参数矩阵。与 Muon 一样，这种方法只需要一个动量缓冲区，同时在实验中优于 Adam。

基本配置。对于归一化层权重，以及偏置、缩放因子等其他非矩阵参数，我们保留 AdamW (Loshchilov and Hutter, 2019)。对于语言模型主干、Engram 投影层和视觉—语言投影器中线性变换的权重矩阵，我们使用 Muon (Jordan et al., 2024)。对于 Query 和 Key 权重，我们使用逐头 Muon。我们为 Muon 应用解耦权重衰减和 Nesterov 动量 (Nesterov, 1983; Liu et al., 2025)；归一化层权重也施加权重衰减，而偏置与缩放因子不施加。经过 Sinkhorn 平衡的更新同样使用 Nesterov 动量，但不施加权重衰减。在预训练期间，视觉编码器保持冻结，直至学习率衰减阶段，而其最终归一化层与视觉—语言投影器始终保持可训练。在学习率开始衰减时，我们解冻视觉编码器，并以较小的学习率将其与大语言模型联合优化。

面向 Engram、嵌入与预测头的 Sinkhorn 平衡更新。完整流程见算法 1。总体而言，它遵循与 Muon 相同的工作流程，只是用 Sinkhorn 平衡取代 Newton – Schulz 正交化。我们用 m 表示矩阵较大的维度；对于嵌入表和预测头，这一维度对应词表大小；用 n 表示隐藏维度。

给定 Nesterov 动量更新 $\widehat{G}_t$，Sinkhorn 平衡寻找对角缩放矩阵 D_r 和 D_c，使得

$$\Delta_t = \sqrt{n}\, U^{(K)} = \sqrt{n}\, D_r \widehat{G}_t D_c, \qquad \frac{1}{n}\sum_{j=1}^{n} (\Delta_t)_{ij}^2 \approx 1, \qquad \frac{1}{m}\sum_{i=1}^{m} (\Delta_t)_{ij}^2 \approx 1, \tag{7}$$

因此，该过程近似均衡了更新矩阵逐行和逐列的 RMS。在这里，一行对应一个 token 索引或 n-gram 标识，一列编码一个隐藏特征。Sinkhorn 平衡通过同时沿行和列归一化，利用这种 token—特征结构。为保证数值稳定性，满足 $\rho_i \leqslant \tau\bar{\rho}$ 的行会被掩蔽。因子 $\sqrt{n}$ 将逐行单位 ℓ_2 范数转换为逐行单位 RMS。另外，我们将有效学习率调整为 $\widetilde{\eta}_t = \gamma\eta_t$，以匹配 Adam 的更新幅度。我们设定 $\gamma = 0.18$，这与 Moonlight (Liu et al., 2025) 中使用的因子 0.2 接近。

更广泛地说，Sinkhorn 平衡与利用矩阵或张量轴结构的优化器密切相关 (Shazeer and Stern, 2018; Zhang et al., 2025a; Wen et al., 2025; Glentis et al., 2025; Deng et al., 2026; Yuan et al., 2026; Xu et al., 2026a)。例如，Adafactor (Shazeer and

Algorithm 1 Momentum update with Sinkhorn balancing

Require: Weight $W_t \in \mathbb{R}^{m\times n}$, gradient G_t, momentum M_{t-1}, momentum coefficient β, base learning rate η_t, learning-rate correction γ, numerical constants ε and τ, and an odd number of normalization steps K

$M_t \leftarrow \beta M_{t-1} + (1-\beta)G_t$
$\widehat{G}_t \leftarrow \beta M_t + (1-\beta)G_t$ ▷ Nesterov momentum
$\rho_i \leftarrow \|\widehat{G}_{t,i,:}\|_2$ and $\bar{\rho} \leftarrow \frac{1}{m}\sum_{i=1}^{m}\rho_i$
$U^{(0)} \leftarrow \widehat{G}_t$
Set $U^{(0)}_{i,:} \leftarrow 0$ if $\rho_i \leqslant \tau\bar{\rho}$ ▷ Mask near-zero rows
for $k = 1, \ldots, K$ **do**
  if k is odd **then**
    for all $i = 1, \ldots, m$ **do**
      $U^{(k)}_{i,:} \leftarrow U^{(k-1)}_{i,:} / (\|U^{(k-1)}_{i,:}\|_2 + \varepsilon)$
    end for
  else
    for all $j = 1, \ldots, n$ **do**
      $U^{(k)}_{:,j} \leftarrow U^{(k-1)}_{:,j} / (\|U^{(k-1)}_{:,j}\|_2 + \varepsilon)$
    end for
  end if
end for
$\Delta_t \leftarrow \sqrt{n}\,U^{(K)}$ ▷ Convert unit row ℓ_2 norm to unit row RMS
$\widetilde{\eta}_t \leftarrow \gamma\eta_t$ ▷ Match the update magnitude of Adam
$W_{t+1} \leftarrow W_t - \widetilde{\eta}_t\Delta_t$

Stern, 2018) conducts row- and column-wise normalization in a different manner, and Adam-mini (Zhang et al., 2025a) uses an alternative row-wise normalization for embedding tables and prediction head. These normalization strategies may differ in optimization performance and communication overhead. We leave more detailed investigation as a future direction.

3. General Infrastructures

3.1. Training Infrastructure

3.1.1. Multimodal Training Infrastructure

Communication-Computation Overlap in Contrastive Learning. The vision encoder is first optimized with a contrastive objective before being fine-tuned with a generative next-token prediction loss. In the contrastive phase, the loss is computed over a full batch of text and vision pairs, so the features of both modalities must be all-gathered across data-parallel ranks, incurring substantial communication. Because the gradient of the text features depends only on the gathered visual features—and, symmetrically, the gradient of the visual features depends only on the gathered text features—each all-gather can be overlapped with the forward or backward pass instead of stalling the pipeline:

$$\begin{aligned}\text{Forward}(V) \;\to\; \big(\text{Forward}(T) \parallel \text{AllGather}(V)\big) \;\to\; \nabla_{\text{Text}} \\ \to\; \big(\text{Backward}(T) \parallel \text{AllGather}(T)\big) \;\to\; \nabla_{\text{Vision}} \;\to\; \text{Backward}(V),\end{aligned}$$

where V and T denote the visual and text features, $(A \parallel C)$ denotes the overlap of computation A with communication C, and ∇ denotes the gradient computation. In this schedule, the visual

Algorithm 1 Momentum update with Sinkhorn balancing

Require: Weight $W_t \in \mathbb{R}^{m\times n}$, gradient G_t, momentum M_{t-1}, momentum coefficient β, base learning rate η_t, learning-rate correction γ, numerical constants ε and τ, and an odd number of normalization steps K

$M_t \leftarrow \beta M_{t-1} + (1-\beta)G_t$
$\widehat{G}_t \leftarrow \beta M_t + (1-\beta)G_t$ ▷ Nesterov momentum
$\rho_i \leftarrow \|\widehat{G}_{t,i,:}\|_2$ and $\bar{\rho} \leftarrow \frac{1}{m}\sum_{i=1}^{m}\rho_i$
$U^{(0)} \leftarrow \widehat{G}_t$
Set $U^{(0)}_{i,:} \leftarrow 0$ if $\rho_i \leqslant \tau\bar{\rho}$ ▷ Mask near-zero rows
for $k = 1, \ldots, K$ **do**
  if k is odd **then**
    for all $i = 1, \ldots, m$ **do**
      $U^{(k)}_{i,:} \leftarrow U^{(k-1)}_{i,:} / (\|U^{(k-1)}_{i,:}\|_2 + \varepsilon)$
    end for
  else
    for all $j = 1, \ldots, n$ **do**
      $U^{(k)}_{:,j} \leftarrow U^{(k-1)}_{:,j} / (\|U^{(k-1)}_{:,j}\|_2 + \varepsilon)$
    end for
  end if
end for
$\Delta_t \leftarrow \sqrt{n}\,U^{(K)}$ ▷ Convert unit row ℓ_2 norm to unit row RMS
$\widetilde{\eta}_t \leftarrow \gamma\eta_t$ ▷ Match the update magnitude of Adam
$W_{t+1} \leftarrow W_t - \widetilde{\eta}_t\Delta_t$

Stern, 2018) 以不同方式进行逐行和逐列归一化，而 Adam-mini (Zhang et al., 2025a) 对嵌入表和预测头采用另一种逐行归一化。这些归一化策略在优化性能和通信开销方面可能有所不同。我们将更详细的研究留作未来方向。

3. 通用基础设施

3.1. 训练基础设施

3.1.1. 多模态训练基础设施

对比学习中的通信与计算重叠。视觉编码器首先使用对比目标进行优化，之后再通过生成式下一 token 预测损失进行微调。在对比阶段，损失基于整批文本与视觉配对计算，因此两种模态的特征都必须在数据并行进程之间执行 all-gather，带来大量通信。由于文本特征的梯度只依赖于收集到的视觉特征，而视觉特征的梯度同样只依赖于收集到的文本特征，所以每次 all-gather 都可以与前向或反向传递重叠，而无需让流水线停顿：

$$\begin{aligned}\text{Forward}(V) \;\to\; \big(\text{Forward}(T) \parallel \text{AllGather}(V)\big) \;\to\; \nabla_{\text{Text}} \\ \to\; \big(\text{Backward}(T) \parallel \text{AllGather}(T)\big) \;\to\; \nabla_{\text{Vision}} \;\to\; \text{Backward}(V),\end{aligned}$$

其中，V 和 T 表示视觉与文本特征，$(A \parallel C)$ 表示计算 A 与通信 C 的重叠，∇ 表示梯度计算。在这一调度中，视觉

features are gathered during the text forward pass and the text features during the text backward pass, so that both all-gathers are hidden entirely behind useful computation.

End-to-End Parallelism. To handle the model and data heterogeneity between the vision encoder and the LLM (Zhang et al., 2025b), we adopt the disaggregated encoder design used in recent training systems (Team et al., 2025, 2026b). The vision encoder is replicated outside the LLM parameter tree, and each training step is divided into three phases: vision encoder forward, LLM forward/backward, and vision encoder backward. This separation prevents interference between vision encoder and LLM computation. Load-balanced vision processing is confined to the first and last phases, while the LLM phase remains free of vision computation and preserves the parallel strategy of text-only training.

Long-Sequence Multimodal Training Optimization. DeepSeek-V4.1-Flash is trained on sequences of up to one million tokens, with a substantial share of training occurring at ultra-long sequence lengths in both the pre-training and post-training stages. At these sequence lengths, multimodal samples create heavy I/O, CPU, and memory bottlenecks.

- **Balanced image sharding.** During pre-training, a single ultra-long, image-dense sequence can exhaust one host's I/O, CPU, and memory during loading, so the images of each sequence are sharded across the CP ranks with load balancing, and each image is loaded exactly once. With images read once, loading stays hidden behind compute whenever

$$\frac{N \times \rho}{B_{\text{IO}}} < \frac{N \times C}{B_{\text{GPU}}} \quad \Longleftrightarrow \quad \rho < \frac{B_{\text{IO}}}{B_{\text{GPU}}} C,$$

  where N is the token count, ρ the raw bytes per token, C the per-token compute, and B_{IO}, B_{GPU} the file-system and GPU bandwidths. Since N cancels, the criterion involves only per-token quantities (ρ and C), independent of sequence length and cluster size; ρ is set by the vision-module configuration (e.g., the resolution cap or the spatial downsample). Storage throughput therefore becomes a bottleneck only for small models with low per-token compute, as in ablations, while production-scale models remain compute-bound.
- **Incremental image transfer.** Besides the balanced sharding above, the reinforcement-learning rollout transfers images to the inference engine only incrementally, and caches the engine's CPU-side decoding and preprocessing outputs on a distributed file system for reuse across rollouts and subsequent training.

3.1.2. Attention Sharing Training for CSA2

In Section 2.3, we introduce CSA2, an attention sharing method with three modes, some of which involve sharing one or more of the main KV, the indexer K, and the Top-K indices across multiple layers. Supporting CSA2 in large-scale distributed training requires additional coordination beyond the attention computation itself. In particular, layers that share attention components may be placed on different pipeline stages, making direct module reuse incompatible with conventional stage-local execution. Therefore, we adopt several designs to support CSA2 training.

Shadow indexers address this issue by placing a lightweight executable replica on each participating stage while retaining a single logical owner for the shared parameters. The owner remains responsible for optimization and checkpointing, whereas parameter synchronization

特征在文本前向传递期间收集，而文本特征在文本反向传递期间收集，从而使两次 all-gather 都被有效计算完全掩盖。

端到端并行。为了处理视觉编码器与大语言模型之间的模型和数据异质性 (Zhang et al., 2025b)，我们采用近期训练系统所使用的解耦编码器设计 (Team et al., 2025, 2026b)。视觉编码器的副本位于大语言模型参数树之外，每个训练步骤分为三个阶段：视觉编码器前向、大语言模型前向与反向、视觉编码器反向。这种分离避免了视觉编码器计算与大语言模型计算之间的相互干扰。需要负载均衡的视觉处理被限制在第一和最后阶段，而大语言模型阶段不包含视觉计算，保留了纯文本训练的并行策略。

长序列多模态训练优化。DeepSeek-V4.1-Flash 使用最长一百万 token 的序列进行训练，预训练和后训练阶段都有相当一部分训练在超长序列长度下进行。在这些序列长度下，多模态样本会造成严重的 I/O、CPU 和内存瓶颈。

• 均衡图像分片。在预训练中，单个包含大量图像的超长序列，可能在加载时耗尽一台主机的 I/O、CPU 和内存资源。因此，每个序列的图像会以负载均衡方式分片到各个 CP 进程，每张图像只加载一次。在图像只读取一次的情况下，只要满足以下条件，加载就能隐藏在计算之后：

$$\frac{N \times \rho}{B_{\text{IO}}} < \frac{N \times C}{B_{\text{GPU}}} \quad \Longleftrightarrow \quad \rho < \frac{B_{\text{IO}}}{B_{\text{GPU}}} C,$$

其中， N 是 token 数量， ρ 是每个 token 对应的原始字节数， C 是每个 token 的计算量， B_{IO} 与 B_{GPU} 分别是文件系统与 GPU 的带宽。由于 N 被约去，该判据只涉及逐 token 的量（ρ 和 C），与序列长度和集群规模无关； ρ 由视觉模块配置决定，例如分辨率上限或空间下采样。因此，存储吞吐量只有在消融实验等逐 token 计算量较低的小模型场景中才会成为瓶颈，生产规模的模型仍然受计算能力限制。• 增量图像传输。除上述均衡分片外，强化学习 rollout 仅以增量方式向推理引擎传输图像，并将引擎在 CPU 端进行解码和预处理得到的输出缓存在分布式文件系统中，以供不同 rollout 和后续训练复用。

3.1.2. CSA2 的注意力共享训练

在第 2.3, 节中，我们介绍了 CSA2：一种包含三种模式的注意力共享方法，其中一些模式涉及在多个层之间共享主 KV、索引器 K 和 Top-K 索引中的一项或多项。要在大规模分布式训练中支持 CSA2，除了注意力计算本身，还需要额外的协调。尤其是，共享注意力组件的层可能位于不同流水线阶段，使直接复用模块与传统的阶段内局部执行方式不兼容。因此，我们采用了若干设计来支持 CSA2 训练。

影子索引器通过在每个参与阶段放置一个轻量的可执行副本来解决这一问题，同时为共享参数保留唯一的逻辑持有者。该持有者仍负责优化和检查点保存，而参数同步

and gradient aggregation keep the shadow replicas consistent throughout training. This design preserves the original model semantics without requiring the pipeline scheduler to treat shared layers as a special execution unit.

Pipeline payload extensions provide the intermediate representations and sparse routing information required by downstream consumers when the source and consumer layers cross a pipeline boundary. These states are incorporated into the existing point-to-point communication path and partitioned consistently with context parallelism, avoiding unnecessary replication while maintaining the corresponding gradient flow.

Micro-batch-level shared-state management tracks the states associated with concurrently active pipeline micro-batches and coordinates their lifetimes across forward execution, activation recomputation, and backward propagation. Shared states are retained until their final consumer has completed and are then released promptly to limit additional memory overhead. The same runtime abstraction also handles stage placement and source–consumer relationships, allowing the attention implementation to access shared states without depending on the physical pipeline layout.

Together with lightweight adaptations to the optimizer, checkpointing, warm-up, and computation-graph tracing workflows, these mechanisms enable CSA2 to operate transparently under the existing distributed training interface and pipeline schedules.

3.1.3. Engram

Engram embedding tables are partitioned by row across dedicated process groups of engram parallel size. The group size controls the trade-off between per-device memory usage and the communication scope of embedding lookups. Optimizer states are further sharded across replicas of each table partition. Engram lookup indices depend solely on the input token sequence. Embedding prefetch is therefore initiated for the entire local batch before each pipeline stage begins processing microbatches for the current training step, minimizing interference with pipeline execution. Embedding gradients are buffered during backward and returned to their owning ranks after the backbone backward pass. For efficient integration with multimodal training, embedding prefetch and gradient transfers are scheduled to overlap with the vision encoder's forward and backward computation. Embeddings are stored and fetched in FP8, with the retrieved values and scaling factors passed directly to the following GEMM. For Engram table updates, Sinkhorn normalization maintains row and column scaling vectors across iterations to avoid repeated writes of the full normalized matrix. Row normalization and the accumulation of partial column statistics are fused into a single kernel to further reduce memory traffic. During RL rollouts, Engram embedding tables remain resident in GPU memory. This placement reduces host memory pressure and helps avoid out-of-memory failures caused by host memory fragmentation.

3.2. Inference System

DeepSeek-V4.1-Flash is designed with inference efficiency as a first-class concern. Although its architecture is conceptually complex, the resulting inference kernel flow is remarkably concise. Through reasonable kernel fusion, we encapsulate the intricate operations and keep hardware resources fully pipelined inside a small number of fused kernels—including the fused-RoPE-attention-RoPE-cast kernel in FlashMLA (Li and Liu, 2025), the Mega-Gate, Mega-*m*HC, and Mega-MoE kernels in DeepGEMM (Zhao et al., 2025), the kernels in TileKernels (Wang et al., 2026a), and the TopK kernel in DeepSelect (Qian et al., 2026). As a result, the vast majority of

与梯度聚合则在整个训练过程中保持影子副本的一致性。这一设计保留了原始模型语义，无需流水线调度器将共享层视为特殊的执行单元。

流水线载荷扩展在源层和使用层跨越流水线边界时，为下游使用方提供所需的中间表示与稀疏路由信息。这些状态被纳入现有的点对点通信路径，并以与上下文并行一致的方式分区，从而在保持相应梯度流的同时避免不必要的复制。

微批次级共享状态管理会跟踪并发活跃的流水线微批次所对应的状态，并在前向执行、激活重计算和反向传播之间协调其生命周期。共享状态会保留到最后一个使用方完成计算，随后及时释放，以限制额外内存开销。同一运行时抽象还负责处理阶段放置与源层—使用层关系，使注意力实现能够访问共享状态，而无需依赖流水线的物理布局。

这些机制与对优化器、检查点、预热和计算图追踪流程的轻量适配相结合，使 CSA2 能够在现有分布式训练接口和流水线调度下透明运行。

3.1.3. Engram

Engram 嵌入表按行划分到规模为 engram parallel size 的专用进程组中。进程组大小控制着每个设备的内存占用与嵌入查询通信范围之间的权衡。优化器状态进一步在各表分区的副本之间分片。Engram 查询索引仅取决于输入 token 序列。因此，在每个流水线阶段开始处理当前训练步骤的微批次之前，会针对整个本地批次启动嵌入预取，以尽量减少对流水线执行的干扰。嵌入梯度在反向阶段进行缓冲，并在主干反向传递结束后返回其所属进程。为高效整合多模态训练，嵌入预取与梯度传输被调度为与视觉编码器的前向和反向计算重叠。嵌入以 FP8 格式存储和获取，取回的值及缩放因子直接传递给后续 GEMM。对于 Engram 表更新，Sinkhorn 归一化在迭代之间保留行、列缩放向量，以避免反复写入完整的归一化矩阵。行归一化与部分列统计量的累加被融合进单个内核，以进一步减少内存流量。在 RL rollout 期间，Engram 嵌入表始终驻留 GPU 内存。这一放置方式降低了主机内存压力，并有助于避免主机内存碎片导致的内存不足故障。

3.2. 推理系统

DeepSeek-V4.1-Flash 在设计时将推理效率置于优先位置。尽管其架构在概念上较为复杂，最终的推理内核流程却十分精简。通过合理的内核融合，我们将复杂操作封装起来，使硬件资源在少量融合内核内部保持充分流水化；这些内核包括 FlashMLA (Li and Liu, 2025) 中的 fused-RoPE-attention-RoPE-cast 内核、DeepGEMM (Zhao et al., 2025) 中的 Mega-Gate、Mega-mHC 和 Mega-MoE 内核、TileKernels (Wang et al., 2026a) 中的内核，以及 DeepSelect (Qian et al., 2026) 中的 TopK 内核。因此，绝大多数

Transformer layers—those whose CSA2 operates in Reuse Mode—execute with only 15 kernels during prefill and 11 during decode, thereby achieving both high-throughput and low-latency inference.

At the deployment level, we adopt Encoder–Prefill–Decode (EPD) disaggregation, enabling vision encoding, prefill, and decoding to scale independently and overlap in execution.

3.2.1. Persistent KV Cache Management

Under identical workloads, V4.1 reduces the persistent KV cache footprint to 1/8 of that of V4. Two multiplicative factors account for this reduction: the persistent KV cache no longer stores SWA KV, which almost halves its size, and the global KV retained in it is further compressed to 1/4 of V4's footprint through architectural and precision optimizations.

In the V4 deployment, SWA KV accounts for nearly half of the persistent KV cache capacity. Within this cache, global KV and SWA KV are managed independently, governed by an LRU eviction policy. Global KV is stored in its entirety, and upon a hit, the complete prefix is reused. In contrast, SWA KV is cached at two specific points—the end of the prompt and the end of the output—to facilitate regeneration and multi-turn sessions; a hit allows computation to resume from that cached position. To maintain a high hit rate, we configured a sufficiently large persistent KV cache on SSD such that, under typical workloads, both types of KV remain resident for over 72 hours. Despite retaining only n_{win} KV entries at designated positions, the uncompressed SWA KV cache still incurs a substantial storage overhead, especially in multi-turn conversations with short turns.

Persistently storing SWA KV is both costly and ineffective, because its access pattern does not match the persistent KV cache's long retention policy. Unlike global KV, which exhibits long-tail reuse, SWA KV is reused only within a narrow, minute-scale window inside an active session and becomes dead once the session ends or the next turn begins. The V4 technical report proposed *Zero SWA Caching*, which avoids the storage overhead by recomputing missing SWA KV. Exact recovery, however, requires a full forward pass over $L \times n_{\text{win}}$ tokens, whose cost proved prohibitive in production deployments.

V4.1 therefore revises persistent KV cache management as follows:

1. SWA KV is no longer cached in the persistent KV cache and instead stored in a distributed memory pool provisioned from 10% of the host DRAM on each machine. Although this pool is far smaller in aggregate capacity, its short TTL (only minutes) allows expired entries to be recycled immediately for new sessions; under real-world workloads, this high turnover suffices to serve the vast majority of concurrent active sessions. Global KV remains in the persistent KV cache with a guaranteed lifetime of at least 72 hours.
2. Evicting SWA KV inevitably causes misses, which stay affordable thanks to a lightweight fallback, *Encoder SWA Bounded Replay* (detailed in Section 3.2.2). For the inevitable but infrequent requests that hit global KV but miss SWA KV, it recovers the missing state by recomputing only n_{win} tokens instead of a full $L \times n_{\text{win}}$-token forward pass. This bounded replay is the cornerstone of the design: it turns a catastrophic miss into a graceful, inexpensive degradation, thereby justifying the removal of SWA KV from the persistent KV cache.

Transformer 层，即 CSA2 运行于 Reuse 模式的那些层，在预填充时仅执行 15 个内核，在解码时仅执行 11 个内核，从而同时实现高吞吐量和低延迟推理。

在部署层面，我们采用编码器—预填充—解码（EPD）解耦，使视觉编码、预填充和解码能够独立扩展，并重叠执行。

3.2.1. 持久化 KV 缓存管理

在相同工作负载下，V4.1 将持久化 KV 缓存占用降至 V4 的八分之一。这种降低来自两个相乘的因素：持久化 KV 缓存不再存储 SWA KV，使其大小近乎减半；其中保留的全局 KV 又通过架构与精度优化，进一步压缩至 V4 对应占用的四分之一。

在 V4 的部署中，SWA KV 占持久化 KV 缓存容量的近一半。在该缓存中，全局 KV 与 SWA KV 分别管理，并采用 LRU
淘汰策略。全局 KV 会被完整存储，命中时复用整个前缀。相比之下，SWA KV 只在两个特定位置缓存，即提示末尾和输出末尾，以支持重新生成与多轮会话；命中后，计算可以从该缓存位置继续。为了保持较高的命中率，我们在 SSD 上配置了足够大的持久化 KV 缓存，使两类 KV 在典型工作负载下都能驻留超过 72 小时。尽管只在指定位置保留 n_{win} 个 KV 条目，未压缩的 SWA KV 缓存仍然带来可观的存储开销，尤其是在每轮较短的多轮对话中。

持久存储 SWA KV 成本高且效果有限，因为它的访问模式与持久化 KV 缓存的长期保留策略并不匹配。全局 KV 具有长尾复用特征，而 SWA KV 仅在活跃会话内一个狭窄的、以分钟计的时间窗口中被复用；一旦会话结束或下一轮开始，它便不再有用。V4 技术报告提出了零 SWA 缓存，通过重新计算缺失的 SWA KV 来避免存储开销。然而，精确恢复需要对 $L \times n_{\text{win}}$ 个 token 执行完整前向传递，实践证明这在生产部署中成本过高。

因此，V4.1 对持久化 KV 缓存管理作出如下调整：

1. SWA KV 不再存入持久化 KV 缓存，而是存放在由每台机器 10% 的主机 DRAM 构成的分布式内存池中。尽管该池的总容量小得多，但较短的 TTL（仅数分钟）使过期条目能够立即回收给新会话使用；在真实工作负载下，这种高周转率足以服务绝大多数并发活跃会话。全局 KV 仍保留在持久化 KV 缓存中，保证至少 72 小时的存活时间。2. 淘汰 SWA KV 不可避免地会造成未命中，但得益于轻量的回退机制——编码器 SWA 有界重放（详见第 3.2.2 节），这一代价仍可承受。对于不可避免但并不频繁的、命中全局 KV 却未命中 SWA KV 的请求，它仅重新计算 n_{win} 个 token 来恢复缺失状态，无需对完整的 $L \times n_{\text{win}}$ 个 token 执行前向传递。这种有界重放是整个设计的基石：它将灾难性的未命中转化为平稳且低成本的降级，从而为将 SWA KV 移出持久化 KV 缓存提供了依据。

3.2.2. SWA Bounded Replay

Since SWA dependencies accumulate across layers, exactly reconstructing the SWA KV of L layers would require replaying $L \times n_{\text{win}}$ tokens. SWA Bounded Replay instead replays only the most recent n_{win} tokens and truncates SWA to the replay segment, accepting approximate states: for a replay starting at position s, a query at position i attends to SWA keys in $\left[\max(s, i - W + 1), i\right]$.

Encoder SWA Bounded Replay. Encoder SWA Bounded Replay makes prefix caching depend only on global KV, allowing SWA KV to be removed from the persistent KV cache.

When the encoder SWA KV is missing, we replay the last n_{win} tokens of the cached prefix and process them together with the uncached suffix. The replayed tokens regenerate only SWA KV, reusing the cached global KV without recomputation or overwriting, while the uncached suffix generates both global KV and SWA KV.

By design, the replayed prefix state is approximate, so the global KV and SWA KV computed for the uncached suffix depend on the cache-hit position and are not mathematically identical across positions. Encouragingly, our experimental evidence confirms that this bounded replay strategy barely compromises response quality.

Decoder SWA Bounded Replay. Decoder SWA Bounded Replay bounds the decoder forward pass to n_{win} tokens, nearly halving total prefill computation.

Under CED, decoder global KV is projected from the final encoder hidden states. The only obstacle to ending prefill at the encoder is decoder SWA KV, which is generated from each decoder layer's own hidden states and is needed by the first decode steps. Since we never cache decoder SWA KV, exactly reconstructing it requires running the $\frac{L}{2}$ decoder layers over the last $\frac{L}{2} \times n_{\text{win}}$ prompt tokens, which is expensive when a short uncached suffix follows a long cached prefix. Therefore, we also apply the bounded replay strategy to this scenario: at every prefill, we replay the last n_{win} tokens of the prompt, feed their encoder outputs through the decoder layers under the same SWA truncation, and use the resulting decoder SWA KV only for decoding, not for prefix caching.

By design, the reconstructed decoder SWA KV is not mathematically equivalent to that from a full decoder forward pass. Also, we find that this strategy has only a negligible impact on response quality. For added safety, we additionally simulate the same replay during post-training for train-aware adaptation.

4. Pre-Training

4.1. Data Construction

Text Data Curation In pursuit of higher intelligence, we go beyond the general, sample-level quality reflected by small-scale data experiments and focus more on the holistic interactions among diverse corpora that offer unique information gains. We adopt a more systematic and standardized data construction pipeline to improve data quality and optimize the data mixture. Specifically, based on more comprehensive evaluations, a scaling ladder over model parameters and training data is carefully designed to guide large-scale training runs. We filter out model-generated content with limited information gain, including outputs from less capable models and low-quality machine-translated text. We regard such content as implicit duplication, as

3.2.2. SWA 有界重放

由于 SWA 的依赖会跨层累积，精确重建 L 层的 SWA KV，需要重放 $L \times n_{\text{win}}$ 个 token。SWA 有界重放则只重放最近的 n_{win} 个 token，并将 SWA 截断在重放片段内，接受近似状态：对于从位置 s 开始的重放，位于 i 的查询会关注以下范围内的 SWA 键：$\left[\max(s, i - W + 1), i\right]$

编码器 SWA 有界重放。编码器 SWA 有界重放使前缀缓存仅依赖全局 KV，从而可以将 SWA KV 移出持久化 KV 缓存。

当编码器 SWA KV 缺失时，我们重放已缓存前缀末尾的 n_{win} 个 token，并将它们与尚未缓存的后缀一起处理。重放的 token 只重新生成 SWA KV，复用已缓存的全局 KV，而不重新计算或覆盖它们；未缓存的后缀则同时生成全局 KV 和 SWA KV。

按照设计，重放得到的前缀状态是近似的，因此为未缓存后缀计算出的全局 KV 和 SWA KV 会依赖缓存命中的位置，在不同位置之间并不具有数学上的完全一致性。令人鼓舞的是，实验证据确认，这种有界重放策略几乎不影响响应质量。

解码器 SWA 有界重放。解码器 SWA 有界重放将解码器前向传递限制在 n_{win} 个 token 内，使总预填充计算量近乎减半。

在 CED 架构下，解码器的全局 KV 由编码器最终的隐藏状态投影得到。将预填充终止在编码器阶段的唯一障碍，是解码器 SWA KV：它由每个解码器层自身的隐藏状态生成，并且是最初几个解码步骤所必需的。由于我们从不缓存解码器 SWA KV，精确重建它需要让 $\frac{L}{2}$ 个解码器层处理提示末尾的 $\frac{L}{2} \times n_{\text{win}}$ 个 token；当较长的已缓存前缀后面只跟着较短的未缓存后缀时，这项计算十分昂贵。因此，我们也在这一场景中采用有界重放策略：每次预填充时，重放提示末尾的 n_{win} 个 token，在相同的 SWA 截断条件下将它们的编码器输出送入解码器各层，并仅将得到的解码器 SWA KV 用于解码，不用于前缀缓存。

按照设计，重建的解码器 SWA KV 与完整解码器前向传递得到的结果在数学上并不等价。同样，我们发现这种策略对响应质量的影响可以忽略。为进一步稳妥起见，我们还在后训练中模拟相同的重放过程，使模型在训练时适应这一机制。

4. 预训练

4.1. 数据构建

文本数据整理。为了追求更高的智能水平，我们不再仅关注小规模数据实验所反映的一般性、样本级质量，而是更加关注能够提供独特信息增益的多样语料之间的整体交互。我们采用更系统、更标准化的数据构建流程，以提升数据质量并优化数据混合比例。具体而言，我们基于更全面的评测，精心设计了覆盖模型参数量与训练数据规模的扩展阶梯，用来指导大规模训练。我们过滤掉信息增益有限的模型生成内容，包括能力较弱模型的输出和低质量机器翻译文本。我们将这些内容视为隐式重复，因为

it largely reformulates existing information and may become detrimental over long training horizons. We also explore model-in-the-loop data iteration approaches as a foundation for future large-scale synthetic data. In addition, we involve more domain experts to construct fine-grained data quality evaluation dimensions. Compared with the previous version, the new corpus incorporates more recent code from newly released open-source repositories, commits, libraries, and emerging frameworks to cover a broader range of programming languages and better reflects contemporary real-world software engineering scenarios.

Multimodal Data Curation Our multimodal pre-training dataset primarily comprises three types of data: image-text pairs, interleaved image-text data, and domain-specific data. Operating on the premise that raw web data naturally provides rich multimodal knowledge, we refrained from large-scale data synthesis; instead, we prioritized cleaning and utilizing the data in its native form to achieve the most direct and scalable visual knowledge compression during pre-training. During the initial data collection, we found that our crawling system was overly biased toward text-centric web content; we therefore re-bootstrapped it from Common Crawl to improve its coverage of multimodal sources. For image-text data, we extract images together with their associated alt text from webpages, filter them by applying an image-text relevance threshold, and deduplicate them based on image semantics. For interleaved data, we build this subset predominantly from webpages and PDFs. Processing large-scale multimodal corpora usually incurs higher CPU and disk storage costs than processing text-only corpora, which motivated us to organize the interleaved-data construction into progressively more expensive stages. Before image retrieval, we apply heuristic and statistical filtering, deduplication, and quality models to select high-value documents. The surviving documents are then assembled into interleaved image–text sequences, where filtering and deduplication are applied again in an image-aware manner. Finally, we employ SmolVLM (Marafioti et al., 2025) to conduct strict quality scoring on the image-text content, thereby extracting high-quality interleaved data. Documents filtered out during this process are partly recycled into additional image-text pairs via screening and recombination. To compensate for the inherent limitations of web-gathered data, we also incorporate domain-specific datasets to boost the model's capabilities in fine-grained visual perception (e.g., visual grounding and pointing), optical character recognition (OCR), and the acquisition of long-tail knowledge. We also collect extensive image-code pairs and computer-use trajectories to improve multimodal agentic understanding.

Data Integration and Deduplication As our text-only and multimodal data were processed through distinct pipelines, we constructed the final training corpus as the union of both data sources. For overlapping samples, we replace the text-only versions with their multimodal counterparts and use the larger epoch count of the two configurations. After this substitution, the resulting corpus uses a 7:1 token ratio of text-only to multimodal data. We minimize sample overlap during pre-training and context extension by jointly prefetching and assigning training samples. Ultra-long documents are deterministically pre-split before mixing to ensure a uniform distribution of training tokens across data shards and training steps. We further enhance our best-fit packing algorithm, achieving a padding rate of at most 10^{-4}.

4.2. Pre-Training Setups

4.2.1. Model Setups

We set the number of Transformer layers to 40 and the hidden dimension d to 5120. We adopt a Causal Encoder-Decoder architecture, with 20 layers in the encoder and 20 layers in the

它们主要是在重新表述已有信息，在长期训练中可能产生负面影响。我们也探索让模型参与循环的数据迭代方法，为未来大规模合成数据奠定基础。此外，我们邀请更多领域专家构建细粒度的数据质量评估维度。与上一版本相比，新语料纳入了更多来自新发布开源仓库、提交、库和新兴框架的近期代码，以覆盖更广泛的编程语言，并更好地反映当代真实世界的软件工程场景。

多模态数据整理。我们的多模态预训练数据集主要包含三类数据：图文对、图文交错数据和领域专用数据。我们认为原始网络数据天然蕴含丰富的多模态知识，因此没有进行大规模数据合成，而是优先清洗并以原生形式利用数据，以便在预训练中实现最直接、最具可扩展性的视觉知识压缩。在最初的数据收集过程中，我们发现爬取系统过度偏向以文本为中心的网页内容，因此从 Common Crawl 重新构建起点，改善对多模态来源的覆盖。对于图文数据，我们从网页中提取图像及其关联的替代文本，应用图文相关性阈值进行过滤，并根据图像语义去重。对于交错数据，我们主要从网页和 PDF 构建这一子集。处理大规模多模态语料通常比处理纯文本语料需要更多 CPU 和磁盘存储资源，因此我们将交错数据构建组织为成本逐步增加的多个阶段。在获取图像之前，我们通过启发式与统计过滤、去重和质量模型筛选高价值文档。随后，将保留下来的文档组装为图文交错序列，再以考虑图像的方式执行过滤和去重。最后，我们使用 SmolVLM（Marafioti et al., 2025）对图文内容进行严格的质量评分，从而提取高质量交错数据。过程中被筛掉的文档，有一部分会通过筛选和重组回收为额外的图文对。为弥补网络采集数据的固有限制，我们还加入领域专用数据集，以增强模型在细粒度视觉感知（例如视觉定位和指点）、光学字符识别（OCR）及长尾知识获取方面的能力。我们也收集了大量图像—代码对和计算机使用轨迹，以提高多模态智能体理解能力。

数据整合与去重。由于纯文本数据和多模态数据由不同的流程处理，我们将两类数据来源的并集构建为最终训练语料。对于重叠样本，我们用对应的多模态版本替换纯文本版本，并采用两种配置中较大的训练轮数。完成替换后，语料中纯文本与多模态数据的 token 比例为 7:1。我们通过联合预取与分配训练样本，尽量减少预训练和上下文扩展期间的样本重叠。超长文档在混合前会以确定性方式预先切分，确保训练 token 在数据分片和训练步骤之间均匀分布。我们进一步改进了最佳适配打包算法，将填充率控制在至多 10^{-4}。

4.2. 预训练配置

4.2.1. 模型配置

我们将 Transformer 层数设为 40，隐藏维度 d 设为 5120。我们采用因果编码器—解码器架构，其中编码器包含 20 层，解码器也包含 20 层，

decoder. For the first two layers, we use pure sliding window attention. The remaining 18 encoder layers use CSA2 with a compression rate of $m = 2$. These layers are divided into three identically configured groups of six layers. In each group, the first layer operates in Full Mode, and the remaining five layers operate in Reuse Mode. The 20 decoder layers use CSA2 with a compression rate of $m = 1$. These layers are divided into five groups of four layers. In the first group, the first layer operates in Full Mode, and the remaining three layers operate in Reuse Mode. The remaining four groups share the same configuration: the first layer operates in Reindex Mode, and the remaining three layers operate in Reuse Mode. For all CSA2 layers, we set the number of indexer query heads to 32, the indexer head dimension to 128, and the number of KV entries selected for sparse attention (i.e., attention top-k) to 512. We set the number of query heads to 64, the head dimension to 512, and the query compression dimension to 1280. For the Hierarchical Sparse Indexer, we select a maximum of 2,048 blocks with 8 positions, yielding up to 16,384 candidate positions in total. The number of output projection groups is set to 8, and the dimension of each intermediate attention output is set to 1024. For the additional branch of sliding window attention, the window size n_{win} is set to 128. We employ MoE layers in all Transformer blocks, using SwiGLU activation function with clamping (OpenAI, 2025) at a threshold of 10. Each MoE layer consists of 1 shared expert and 384 routed experts, where the intermediate hidden dimension of each expert is 2304. Among the routed experts, 6 experts will be activated for each token. As for *m*HC, the expansion factor is set to 4, and the number of Sinkhorn-Knopp iterations is set to 20. For the vision encoder, we set its number of layers to 32, the hidden dimension to 1024, the number of attention heads to 16, and the image patch size to 14. The vision MLP projector has 2 layers with a hidden dimension of 5120. Under this configuration, DeepSeek-V4.1-Flash comprises 552B backbone parameters, with 8B activated per token during prefill and 16B during decode.

4.2.2. Training Setups

We employ the Muon optimizer (Jordan et al., 2024; Liu et al., 2025) for the parameters of linear transformations, use AdamW optimizer (Loshchilov and Hutter, 2019) for the weights of all RMSNorm modules and other non-matrix parameters, and use Sinkhorn-balanced update for all embeddings and prediction head. For AdamW, we set its hyper-parameters to $\beta_1 = 0.9$, $\beta_2 = 0.95$, $\varepsilon = 10^{-20}$, and weight_decay = 0.1. For Muon, we set the momentum to 0.95 and the weight decay to 0.1, and rescale the RMS of each update matrix to 0.18 for reutilization of the AdamW learning rate. For Sinkhorn-balanced updating, we use the same momentum coefficient and learning-rate correction factor as for Muon and set $K = 11$, $\tau = 10^{-3}$, $\varepsilon = 10^{-20}$. Following (Cheng et al., 2026b), the learning rate of Engram is scaled by 5×. We train DeepSeek-V4.1-Flash on 45T tokens of multimodal data with no instability. We keep the batch size fixed at 100.6 million tokens throughout training. The learning rate is linearly warmed up over the first 2000 steps and then maintained at 2.6×10^{-4} until 28T tokens. Between 28T and 40T tokens, we decay the learning rate to 2.6×10^{-5} following a cosine schedule. We keep the learning rate at this value from 40T to 45T tokens. We train the model from scratch with sparse attention at a sequence length of 64K and extend the sequence length to 1M at 34T tokens. For auxiliary-loss-free load balancing, we set the bias update speed to 0.001 for both image and text tokens, while retaining a small sequence-level balance loss with a loss weight of 0.0001 to avoid extreme imbalance within single sequences. Similar to DeepSeek-V4, we employ sample-level attention masking during pre-training.

Vision Encoder Training. Our DeepSeek-ViT undergoes a separate training stage before being integrated with the language backbone. The training pipeline consists of two stages: *contrastive*

解码器。前两层仅使用滑动窗口注意力。其余 18 个编码器层使用压缩率为 $m = 2$ 的 CSA2。这些层划分为三组，每组六层，配置相同：每组第一层运行于 Full 模式，其余五层运行于 Reuse 模式。20 个解码器层使用压缩率为 $m = 1$ 的 CSA2，划分为五组，每组四层。第一组的第一层运行于 Full 模式，其余三层运行于 Reuse 模式。后面四组使用相同的配置：第一层运行于 Reindex 模式，其余三层运行于 Reuse 模式。对于所有 CSA2 层，我们将索引器查询头数设为 32，索引器头维度设为 128，用于稀疏注意力的 KV 条目选择数量（即 attention top-k）设为 512。查询头数设为 64，头维度设为 512，查询压缩维度设为 1280。对于分层稀疏索引器，我们最多选择 2,048 个块，每块包含 8 个位置，总计最多 16,384 个候选位置。输出投影组数设为 8，每个中间注意力输出的维度设为 1024。对于额外的滑动窗口注意力分支，窗口大小 n_{win} 设为 128。我们在所有 Transformer 块中使用 MoE 层，采用带截断的 SwiGLU 激活函数 (OpenAI, 2025)，截断阈值为 10。每个 MoE 层由 1 个共享专家和 384 个路由专家组成，每个专家的中间隐藏维度为 2304。对于每个 token，路由专家中有 6 个被激活。mHC 的扩展因子设为 4，Sinkhorn-Knopp 迭代次数设为 20。视觉编码器的层数设为 32，隐藏维度设为 1024，注意力头数设为 16，图像块大小设为 14。视觉 MLP 投影器包含 2 层，隐藏维度为 5120。在这一配置下，DeepSeek-V4.1-Flash 包含 552B 主干参数，在预填充阶段每个 token 激活 8B 参数，在解码阶段激活 16B 参数。

4.2.2. 训练配置

我们将 Muon 优化器 (Jordan et al., 2024; Liu et al., 2025) 用于线性
变换的参数，将 AdamW 优化器 (Loshchilov and Hutter, 2019) 用于所有 RMSNorm 模块的权重及其他非矩阵参数，并对所有嵌入和预测头采用 Sinkhorn 平衡更新。对于 AdamW，超参数设置为 $\beta_1 = 0.9$、$\beta_2 = 0.95$、$\varepsilon = 10^{-20}$，以及 weight_decay = 0.1 对于 Muon，我们将动量设为 0.95、权重衰减设为 0.1，并将每个更新矩阵的 RMS 重新缩放为 0.18，以复用 AdamW 的学习率。对于 Sinkhorn 平衡更新，我们采用与 Muon 相同的动量系数和学习率校正因子，并设置 $K = 11$、$\tau = 10^{-3}$、$\varepsilon = 10^{-20}$。遵循 (Cheng et al., 2026b) 的做法，Engram 的学习率按 5× 缩放。我们在 45T token 的多模态数据上训练 DeepSeek-V4.1-Flash，过程中没有出现不稳定。整个训练过程的批大小固定为 1.006 亿 token。学习率在前 2000 步线性预热，随后保持为 2.6×10^{-4}，直至训练到 28T token。在 28T 至 40T token 之间，学习率按余弦调度衰减至 2.6×10^{-5}；从 40T 到 45T token，学习率保持这一数值。模型从一开始就使用稀疏注意力，以 64K 序列长度训练，并在 34T token 时将序列长度扩展到 1M。对于无辅助损失负载均衡，我们为图像与文本 token 都设置 0.001 的偏置更新速度，同时保留一个损失权重为 0.0001 的小规模序列级均衡损失，以避免单个序列内部出现极端不均衡。与 DeepSeek-V4 类似，我们在预训练时采用样本级注意力掩码。

视觉编码器训练。DeepSeek-ViT 在接入语言主干之前，先经历独立训练阶段。训练流程包含两个阶段：对比

pre-training and *autoregressive fine-tuning*. During contrastive pretraining, we optimize the model using the sigmoid contrastive loss introduced by SigLIP (Zhai et al., 2023) on approximately 47B image-text pairs sourced from alt-text data. To efficiently learn visual representations from such massive datasets, we restrict the maximum input resolution to 224×224 pixels by downscaling larger images while preserving their aspect ratios. Although using higher resolutions in this phase yields notable gains, empirical results show that these benefits contribute little to the final model. Because the subsequent autoregressive stage specifically handles high-resolution extrapolation, scaling up resolutions during contrastive pretraining significantly increases computational overhead without much overall improvement. In the autoregressive fine-tuning stage, we connect the vision encoder to a 4B MoE LLM and train on 236B tokens across datasets including image captions, alt text, charts, and OCR, using a next-token prediction objective. This stage aims to enhance the encoder's ability to model fine-grained visual features. We therefore constrain the input resolution between 544×544 and 1344×1344 pixels by proportionally scaling out-of-bound images. After this stage, we discard the LLM and retain only the optimized vision encoder for the subsequent pre-training pipeline, where the same input-resolution policy is maintained.

4.3. Evaluations

4.3.1. Evaluation Benchmarks

We compare DeepSeek-V4.1-Flash-Base with its predecessor models DeepSeek-V4-Flash-Base and DeepSeek-V4-Pro-Base. We report benchmarks spanning five key dimensions: world knowledge, language understanding and reasoning, coding and mathematics, long context, and multimodal abilities.

World knowledge benchmarks include AGIEval (Zhong et al., 2023), MMLU-Pro (Wang et al., 2024b), C-Eval (Huang et al., 2023), MultiLoKo (Hupkes and Bogoychev, 2025), SimpleQA-Verified (Haas et al., 2025) and SuperGPQA (Du et al., 2025),

Language understanding and reasoning benchmarks include BigBench Hard (BBH) (Suzgun et al., 2022), BigBench Extra Hard (BBEH) (Kazemi et al., 2025), DROP (Dua et al., 2019) and HellaSwag (Zellers et al., 2019),

Coding and mathematical benchmarks include BigCodeBench (Zhuo et al., 2025), HumanEval (Chen et al., 2021), GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021) and MGSM (Shi et al., 2023),

Long context benchmark includes LongBench-V2 (Bai et al., 2025).

Multimodal benchmarks include MMMU-Pro (Yue et al., 2025), DocVQA (Mathew et al., 2021), CVBench (Tong et al., 2024) and RefCOCO/RefCOCO+/RefCOCO-g (Kazemzadeh et al., 2014; Nagaraja et al., 2016; Mao et al., 2016; Yu et al., 2016).

4.3.2. Evaluation Results

In Table 1, we provide a detailed comparison of the base models for DeepSeek-V4-Flash, DeepSeek-V4-Pro and DeepSeek-V4.1-Flash, all evaluated under our internal evaluation framework using strictly controlled and reproducible settings. Compared with DeepSeek-V4-Flash-Base and DeepSeek-V4-Pro-Base, our latest base model reveals a compelling efficiency gain. DeepSeek-V4.1-Flash activates a substantially smaller number of parameters than DeepSeek-V4-Pro-Base and occupies a heavily-reduced KV cache, yet its performance is fully on par

预训练与自回归微调。在对比预训练中，我们在约 47B 个来自替代文本数据的图文对上，使用 SigLIP (Zhai et al., 2023) 引入的 sigmoid 对比损失优化模型。为从如此庞大的数据集中高效学习视觉表示，我们在保留宽高比的同时缩小较大图像，将最大输入分辨率限制为 224×224 像素。虽然在这一阶段使用更高分辨率能带来明显收益，但实验结果表明，这些收益对最终模型的贡献很小。由于后续自回归阶段专门处理高分辨率外推，在对比预训练中提高分辨率会显著增加计算开销，却不会带来多少整体改善。在自回归微调阶段，我们将视觉编码器连接到一个 4B MoE 大语言模型，使用下一 token 预测目标，在包含图像描述、替代文本、图表和 OCR 的数据集上训练 236B token。此阶段旨在增强编码器对细粒度视觉特征的建模能力。因此，我们按比例缩放超出范围的图像，将输入分辨率限制在 544×544 到 1344×1344 像素之间。完成这一阶段后，我们丢弃大语言模型，仅保留优化后的视觉编码器，用于后续预训练流程，并继续采用相同的输入分辨率策略。

4.3. 评测

4.3.1. 评测基准

我们将 DeepSeek-V4.1-Flash-Base 与其前代模型 DeepSeek-V4-Flash-Base 和 DeepSeek-V4-Pro-Base 进行比较。我们报告涵盖五个关键维度的基准结果：世界知识、语言理解与推理、编程与数学、长上下文，以及多模态能力。

世界知识基准包括 AGIEval (Zhong et al., 2023)、MMLU-Pro (Wang et al., 2024b)、C-Eval (Huang et al., 2023)、MultiLoKo (Hupkes and Bogoychev, 2025)、SimpleQA-Verified (Haas et al., 2025) 和 SuperGPQA (Du et al., 2025)。

语言理解与推理基准包括 BigBench Hard（BBH）(Suzgun et al., 2022)、BigBench Extra Hard（BBEH）(Kazemi et al., 2025)、DROP (Dua et al., 2019) 和 HellaSwag (Zellers et al., 2019)。

编程与数学基准包括 BigCodeBench (Zhuo et al., 2025)、HumanEval (Chen et al., 2021)、GSM8K (Cobbe et al., 2021)、MATH (Hendrycks et al., 2021) 和 MGSM (Shi et al., 2023)。

长上下文基准包括 LongBench-V2 (Bai et al., 2025)。

多模态基准包括 MMMU-Pro (Yue et al., 2025)、DocVQA (Mathew et al., 2021)、CVBench (Tong et al., 2024)，以及 RefCOCO/RefCOCO+/RefCOCO-g (Kazemzadeh et al., 2014; Nagaraja et al., 2016; Mao et al., 2016; Yu et al., 2016)。

4.3.2. 评测结果

表 1 详细比较了 DeepSeek-V4-Flash、DeepSeek-V4-Pro 和 DeepSeek-V4.1-Flash 的基础模型。所有模型都在我们内部的评测框架下，使用严格控制且可复现的设置进行评估。相较于 DeepSeek-V4-Flash-Base 和 DeepSeek-V4-Pro-Base，最新的基础模型展现出显著的效率收益。DeepSeek-V4.1-Flash 激活的参数量远少于 DeepSeek-V4-Pro-Base，KV 缓存占用也大幅缩小，但其性能已达到

Table 1 | Comparison among DeepSeek-V4-Flash-Base, DeepSeek-V4-Pro-Base, and DeepSeek-V4.1-Flash-Base. All models are evaluated in our internal framework and share the same evaluation setting. Scores with a gap not exceeding 0.3 are considered to be at the same level. The highest score in each row is in **bold font**, and the second is underlined.

	Benchmark (Metric)	# Shots	DeepSeek-V4-Flash Base	DeepSeek-V4-Pro Base	DeepSeek-V4.1-Flash Base
	Architecture	-	MoE	MoE	MoE
	# Activated Params	-	13B	49B	8B/16B
	# Backbone Params	-	284B	1.6T	552B
World Knowl.	AGIEval (EM)	3-5-shot	83.9	**84.4**	83.4
	MMLU-Pro (EM)	5-shot	68.3	73.5	**74.1**
	C-Eval (EM)	5-shot	92.1	**93.1**	92.1
	MultiLoKo (LLM-Judge)	5-shot	42.6	**50.9**	45.5
	Simple-QA verified (EM)	25-shot	30.1	**55.2**	42.3
	SuperGPQA (EM)	5-shot	46.5	**53.9**	53.1
Lang. & Reas.	BBH (EM)	3-shot	86.9	**87.5**	86.1
	BBEH (EM)	1-shot	25.4	**29.8**	27.2
	DROP (F1)	1-shot	**88.6**	**88.7**	87.9
	HellaSwag (EM)	0-shot	85.7	**88.0**	87.2
Code & Math	BigCodeBench (Pass@1)	3-shot	56.8	59.2	**60.6**
	HumanEval (Pass@1)	0-shot	69.5	76.8	**79.4**
	GSM8K (EM)	8-shot	90.8	92.6	**93.0**
	MATH (EM)	4-shot	57.4	**64.5**	61.1
	MGSM (EM)	8-shot	**85.7**	84.4	80.2
Long Context	LongBench-V2 (EM)	1-shot	44.7	**51.5**	45.2
Multimodal	MMMU-Pro (EM)	4-shot	-	-	56.5
	CVBench (EM)	4-shot	-	-	77.9
	DocVQA (LLM-Judge)	4-shot	-	-	95.6
	RefCOCO-avg (Acc@0.5)	0-shot	-	-	86.0

with its predecessors. These results also reflect the substantial improvements we made to our pre-training data curation pipeline. In this version, we introduce native multimodal training and validate its effectiveness through corresponding multimodal evaluations. Trained on a more diverse and multimodal corpus, DeepSeek-V4.1-Flash achieves world knowledge and comprehension capabilities comparable to those of DeepSeek-V4-Pro. In reasoning and coding benchmarks, DeepSeek-V4.1-Flash shows consistent progress, reaching performance close to or better than DeepSeek-V4-Pro across multiple benchmarks.

To further assess the model's capabilities in real-world R&D scenarios, we additionally perform perplexity tests on dedicated internal corpora. Since perplexity tests are impossible through model APIs, we mainly focus on our own pretrained base models. For corpus selection, a separate evaluation set is collected based on our daily development, including internal documentation, proprietary code repositories and academic materials, which targets for reasoning, attribution and problem-solving on complex scientific problems and frontier research. The results are shown in Figure 6, where we report the bits-per-byte(BPB) of different models, with lower values indicating better performance.

Table 1 | Comparison among DeepSeek-V4-Flash-Base, DeepSeek-V4-Pro-Base, and DeepSeek-V4.1-Flash-Base. All models are evaluated in our internal framework and share the same evaluation setting. Scores with a gap not exceeding 0.3 are considered to be at the same level. The highest score in each row is in **bold font**, and the second is underlined.

	Benchmark (Metric)	# Shots	DeepSeek-V4-Flash Base	DeepSeek-V4-Pro Base	DeepSeek-V4.1-Flash Base
	Architecture	-	MoE	MoE	MoE
	# Activated Params	-	13B	49B	8B/16B
	# Backbone Params	-	284B	1.6T	552B
World Knowl.	AGIEval (EM)	3-5-shot	83.9	**84.4**	83.4
	MMLU-Pro (EM)	5-shot	68.3	73.5	**74.1**
	C-Eval (EM)	5-shot	92.1	**93.1**	92.1
	MultiLoKo (LLM-Judge)	5-shot	42.6	**50.9**	45.5
	Simple-QA verified (EM)	25-shot	30.1	**55.2**	42.3
	SuperGPQA (EM)	5-shot	46.5	**53.9**	53.1
Lang. & Reas.	BBH (EM)	3-shot	86.9	**87.5**	86.1
	BBEH (EM)	1-shot	25.4	**29.8**	27.2
	DROP (F1)	1-shot	**88.6**	**88.7**	87.9
	HellaSwag (EM)	0-shot	85.7	**88.0**	87.2
Code & Math	BigCodeBench (Pass@1)	3-shot	56.8	59.2	**60.6**
	HumanEval (Pass@1)	0-shot	69.5	76.8	**79.4**
	GSM8K (EM)	8-shot	90.8	92.6	**93.0**
	MATH (EM)	4-shot	57.4	**64.5**	61.1
	MGSM (EM)	8-shot	**85.7**	84.4	80.2
Long Context	LongBench-V2 (EM)	1-shot	44.7	**51.5**	45.2
Multimodal	MMMU-Pro (EM)	4-shot	-	-	56.5
	CVBench (EM)	4-shot	-	-	77.9
	DocVQA (LLM-Judge)	4-shot	-	-	95.6
	RefCOCO-avg (Acc@0.5)	0-shot	-	-	86.0

与前代模型相同的水平。这些结果也反映了我们对预训练数据整理流程所做的显著改进。在这一版本中，我们引入原生多模态训练，并通过相应的多模态评测验证其有效性。由于在更多样的多模态语料上训练，DeepSeek-V4.1-Flash 获得了与 DeepSeek-V4-Pro 相当的世界知识与理解能力。在推理和编程基准上，DeepSeek-V4.1-Flash 展现出持续进步，在多个基准上达到接近或优于 DeepSeek-V4-Pro 的性能。

为了进一步评估模型在真实研发场景中的能力，我们还在专门的内部语料上进行困惑度测试。由于无法通过模型 API 进行困惑度测试，我们主要关注自行预训练的基础模型。在语料选择上，我们基于日常研发工作收集了独立评测集，其中包括内部文档、专有代码仓库和学术材料，面向复杂科学问题及前沿研究中的推理、归因和问题求解。结果见图 6，我们报告了不同模型的每字节比特数（BPB），数值越低表示性能越好。

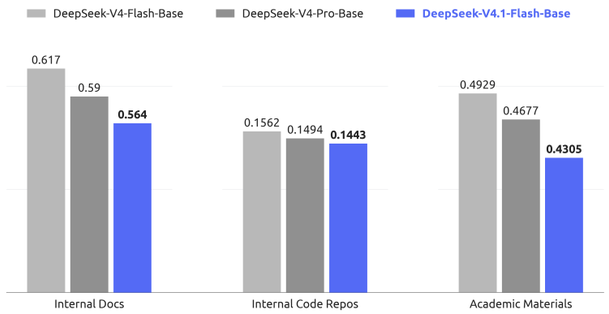


Figure 6 | Bits-per-bytes (BPB) comparison of DeepSeek-V4-Flash-Base, DeepSeek-V4-Pro-Base and DeepSeek-V4.1-Flash-Base on our held-out evaluation sets. DeepSeek-V4.1-Flash-Base achieves lowest BPB on all tasks and demonstrates greater potential to serve as a strong base-model.

5. Post-Training

5.1. Post-Training Pipeline

In this release, we refrain from introducing novel post-training algorithms. The overall recipe follows the standard paradigm of supervised fine-tuning (SFT) followed by reinforcement learning (RL) and on-policy distillation (OPD; Gu et al., 2024; Lu and Lab, 2025), without algorithmic modifications beyond well-established practices. Instead, our efforts are concentrated almost entirely on what the model is trained on rather than how it is optimized: we invest in large-scale, automated pipelines for data synthesis and environment construction. Concretely, the pipeline (i) synthesizes diverse, verifiable training tasks together with their reference solutions and reward signals, (ii) procedurally constructs and scales interactive agent environments in which trajectories can be collected and evaluated at low cost, and (iii) applies rigorous filtering, deduplication, and difficulty calibration to ensure data quality and curriculum balance. We find that, under a fixed and unremarkable optimization procedure, systematic improvements in the scale, diversity, and verifiability of synthesized data and environments account for essentially all of the observed gains. This observation echoes a broader lesson: at the current stage, the marginal return of engineering the data and environment pipeline substantially exceeds that of algorithmic novelty in post-training.

5.1.1. Large-Scale Agent Task Synthesis

Tasks serve as the fundamental fuel for agent learning. However, constructing high-quality training tasks has traditionally required substantial manual effort. We observe that the model is already beginning to exhibit the ability to construct its own training tasks, though this capability remains far from perfect. Recognizing this potential, we have invested considerable effort in strengthening the model's task-construction and quality-verification abilities.

We formalize each task as a triplet (problem, environment, verification system) and evaluate its quality along two dimensions: difficulty—ensuring the task is non-trivial—and correctness—guaranteeing that no critical flaws exist among the three components. Using difficulty and correctness as reward signals, we iteratively train the model to construct better tasks. We also monitor RL tasks across their full lifecycle. Whenever a task is used in a new RL run,

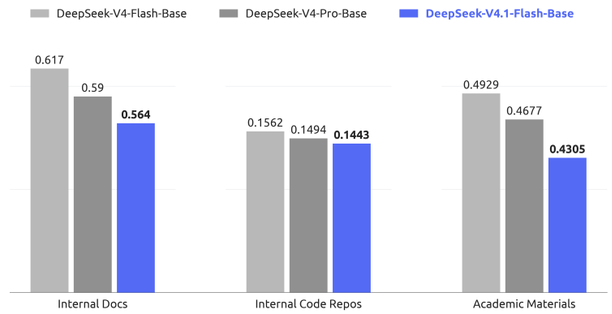


图 6 | DeepSeek-V4-Flash-Base、DeepSeek-V4-Pro-Base 和 DeepSeek-V4.1-Flash-Base 在留出评测集上的每字节比特数（BPB）比较。DeepSeek-V4.1-Flash-Base 在所有任务上都取得最低的 BPB，展现出作为强大基础模型的更大潜力。

5. 后训练

5.1. 后训练流程

在此次发布中，我们没有引入新的后训练算法。整体方案遵循监督微调（SFT）、强化学习（RL）和在线策略蒸馏（OPD；Gu et al., 2024; Lu and Lab, 2025）的标准范式，没有在成熟实践之外进行算法改动。我们的努力几乎完全集中于模型使用什么内容训练，而非如何优化模型：我们投入建设大规模、自动化的数据合成和环境构建流程。具体而言，该流程会：（i）合成多样且可验证的训练任务，以及对应的参考解答和奖励信号；（ii）通过程序构建并扩展交互式智能体环境，以低成本收集和评估轨迹；（iii）执行严格的过滤、去重和难度校准，保证数据质量与课程均衡。我们发现，在固定且并不特别的优化过程下，观察到的收益基本全部来自对合成数据和环境的规模、多样性及可验证性的系统性改进。这一观察呼应了一个更普遍的经验：在当前阶段，对数据与环境流程进行工程改进的边际回报，显著高于后训练中的算法创新。

5.1.1. 大规模智能体任务合成

任务是智能体学习的基本燃料。然而，构建高质量训练任务在传统上需要大量人工投入。我们观察到，模型已经开始展现出自行构建训练任务的能力，尽管这一能力还远不完善。认识到这一潜力后，我们投入了大量精力，增强模型的任务构建和质量验证能力。

我们将每个任务形式化为一个三元组（问题、环境、验证系统），并从两个维度评估其质量：难度，即确保任务并非轻而易举；正确性，即确保三个组成部分之间不存在严重缺陷。我们以难度和正确性作为奖励信号，迭代训练模型以构建更好的任务。我们也对 RL 任务的整个生命周期进行监测。每当某项任务用于一次新的 RL 训练时，

the resulting trajectories provide fresh evidence for quality re-auditing. Under this general framework, we have built dedicated training environment production pipelines for two core scenarios: general agents and coding agents.

General Agent. For general agents, we encourage internal employees and external partners to incorporate our latest model into their routine workflows and, on a voluntary basis, return interaction data and feedback. Based on the interfaces observed in the returned data, we construct a large set of mocked tools that reproduce the interfaces and behaviors of real-world tools and systems, including their input formats, output structures, API schemas, and behavioral constraints, covering both commonly used SaaS and enterprise applications as well as more specialized business back-end systems. In parallel, we collect negative feedback and model failure cases submitted by internal employees at scale, and incorporate them into the pipeline to generate both single-turn and multi-turn agent environments grounded in real workflows. By reconstructing the relevant tool context, user interaction patterns, and failure conditions, the pipeline enables systematic replay of failures and targeted reinforcement learning against observed model weaknesses.

Coding Agent. Coding agent training environments are built from two sources: 1) coding-agent sessions from internal employees and external partners, filtered to retain highly complex tasks or tasks on which model performance is poor, then deduplicated by trajectory; and 2) public GitHub repositories that meet a star-count threshold. Environment construction is carried out collaboratively by multiple specialized agents. First, an agent determines whether the project can be built and fully run inside a container and whether it can be automatically verified; if so, it selects a specific turn or commit as the task starting point, designs several sufficiently complex implementation directions, and produces concrete evaluation points, including both fail-to-pass and pass-to-pass points, along with a construction report, fetching external resources from the web as needed. Next, a separate agent sets up dependencies, the initial working directory, test code, and task descriptions in an isolated container, performs self-testing, removes any traces that could leak the task solution, and packages the environment as a new image layer. Then, multiple distinct agents attempt the task, and an independent quality-inspection agent reviews the environment together with the solving agents' trajectories, checking for environment issues, factual errors, mismatches between evaluation points and task descriptions, and hackability risks. If the inspection does not pass, a repair agent fixes all identified errors, adjusts evaluation points that are too easy or too difficult, and the task re-enters verification.

Through these pipelines, we can automatically and batch-produce RL training data that is correct, discriminative, and controllable in length and difficulty. Whether through the faithful reconstruction of real workflows in general agent environments or the precise construction of coding tasks in coding agent environments, both ultimately feed into a unified training system, driving iterative improvement of model capabilities under continuous quality monitoring.

5.1.2. RL in Synthesized Tasks

We use large-scale asynchronous RL in synthesized tasks to improve model performance and shape its behavior in complicated scenarios. We scale RL runs in two dimensions, i.e., training compute and the number of scaffolds. As shown in Figure 7 and Figure 8, performance continues to improve as we increase compute with cumulative RL steps, whether scaling within a single scaffold, jointly across variants of the same scaffold, or across heterogeneous scaffolds.

For RL training across diverse scaffolds, we decouple agent rollout execution into an agent sandbox and a worker container. The sandbox runs the scaffold and its tools, while the worker

产生的轨迹都会为重新审核质量提供新的证据。在这一通用框架下，我们针对两个核心场景建立了专门的训练环境生产流程：通用智能体与编程智能体。

通用智能体。对于通用智能体，我们鼓励内部员工与外部合作伙伴在日常工作流程中使用最新模型，并自愿反馈交互数据和意见。根据反馈数据中出现的接口，我们构建了大量模拟工具，复现现实工具和系统的接口与行为，包括输入格式、输出结构、API 模式和行为约束；覆盖范围既包括常用的 SaaS 与企业应用，也包括更专门的业务后端系统。同时，我们大规模收集内部员工提交的负面反馈和模型失败案例，并将其纳入流程，生成基于真实工作流程的单轮与多轮智能体环境。通过重建相关工具上下文、用户交互模式和失败条件，该流程能够系统性地重放失败，并针对观察到的模型弱点开展有针对性的强化学习。

编程智能体。编程智能体训练环境来自两个来源：1）内部员工与外部合作伙伴的编程智能体会话，经过筛选，仅保留高度复杂的任务或模型表现较差的任务，再按轨迹去重；2）达到星标数量门槛的公开 GitHub 仓库。环境构建由多个专门智能体协作完成。首先，一个智能体判断项目能否在容器内构建并完整运行，以及能否自动验证；如果可以，它会选择特定的交互轮次或提交作为任务起点，设计若干足够复杂的实现方向，并产生具体评估点，包括从失败变为通过和保持通过的评估点，同时输出构建报告，必要时从网络获取外部资源。随后，另一个智能体在隔离容器中设置依赖、初始工作目录、测试代码和任务描述，执行自测，移除可能泄露任务解答的痕迹，并将环境打包为新的镜像层。接着，多个不同的智能体尝试完成任务，一个独立质检智能体结合环境和求解智能体的轨迹进行审查，检查环境问题、事实错误、评估点与任务描述不匹配，以及可被投机利用的风险。如果质检未通过，修复智能体会纠正所有已识别错误，调整过于简单或过于困难的评估点，任务随后重新进入验证。

通过这些流程，我们可以自动、批量地生产正确、有区分度，且长度与难度可控的 RL 训练数据。无论是通用智能体环境中对真实工作流程的忠实重建，还是编程智能体环境中对编程任务的精确构建，最终都汇入统一的训练系统，在持续质量监测下推动模型能力迭代提升。

5.1.2. 合成任务中的强化学习

我们在合成任务中采用大规模异步 RL，以提高模型性能，并
塑造其在复杂场景中的行为。我们沿两个维度扩展 RL 训练，即训练计算量和运行框架数量。如图 7 和图 8 所示，无论是在单一运行框架中扩展、联合使用同一框架的不同变体，还是跨异构框架训练，随着累计 RL 步数增加、投入计算量扩大，性能都持续提升。

为了在多样的运行框架下进行 RL 训练，我们将智能体 rollout 的执行解耦为智能体沙箱与工作容器。沙箱运行框架及其工具，而工作容器

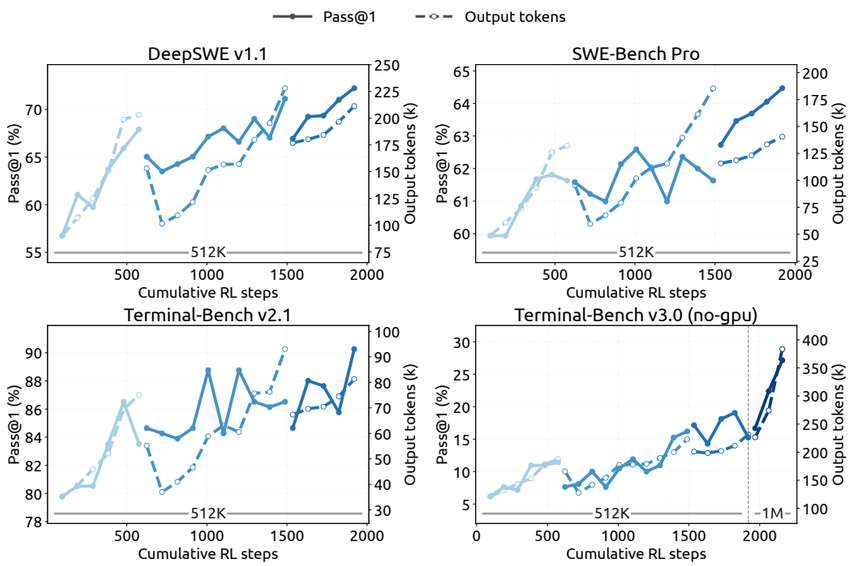


Figure 7 | Performance improves on various code agent benchmarks as RL training scales in the Minimal mode of DeepSeek Harness. Further extending maximal context length to 1M tokens continues to improve performance on extremely long-horizon tasks, e.g., Terminal-Bench v3.0.

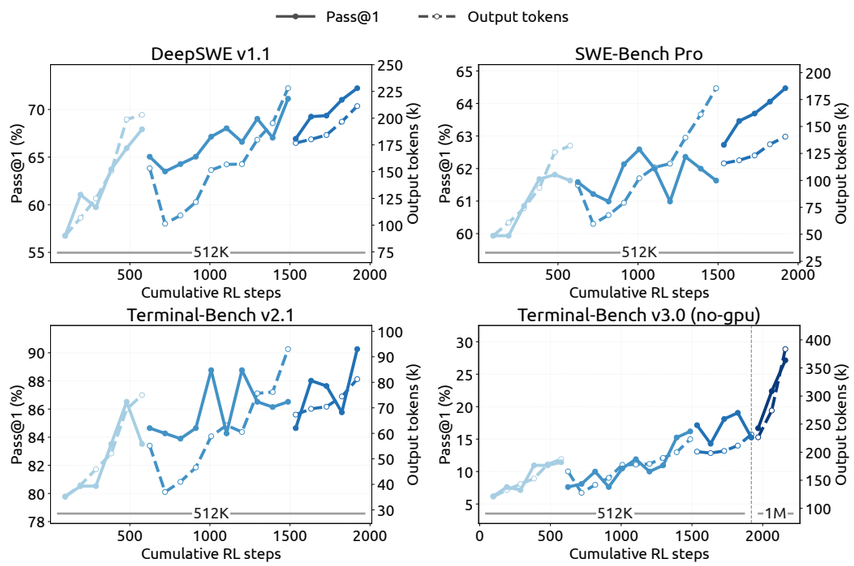


图 7 | 在 DeepSeek Harness 的 Minimal 模式下，随着 RL 训练规模扩大，各种编程智能体基准上的性能不断提升。进一步将最大上下文长度扩展到 1M token，还能继续提高 Terminal-Bench v3.0 等极长程任务上的性能。

provides a scaffold-agnostic control layer that orchestrates the rollout, normalizes heterogeneous interactions into a common trajectory schema, and communicates with the trainer. Both run on DSec (Section 5.1.3), outside the preemptible GPU training pool, separating long-lived rollouts from fine-grained training scheduling. During trainer preemption, rollout execution can be suspended and offloaded while preserving its full state for later resumption and releasing CPU and GPU resources. This design enables stable and efficient RL across heterogeneous scaffolds without modifying the underlying algorithms.

To extend effective RL compute beyond a single training run, we use model merging to reinitialize successive RL runs. Specifically, we merge checkpoints from runs across different scaffolds or configurations, combining improvements acquired along different optimization paths. In Figure 7 and Figure 8, disconnected curve segments reflect successive RL runs after model reinitialization. This yields further gains in both task performance and token efficiency, providing a simple and practical way to aggregate parallel RL compute and continue scaling across successive runs.

5.1.3. Running Agents at Massive Scale: DSec

As we transitioned from DeepSeek-V3 to V4, the rapidly growing number and diversity of agentic training environments motivated us to build *DeepSeek Elastic Compute (DSec)*, a production-grade sandbox platform for large-scale agentic training and evaluation. Its initial design addressed heterogeneous execution environments, scalable image distribution, multiple isolation backends, high-density resource management, command trajectory logging, and preemption-safe resumption.

V4.1 training further increased demand to millions of concurrent sandbox instances span-

提供与运行框架无关的控制层，协调 rollout，将异构交互规范化为统一的轨迹模式，并与训练器通信。两者都运行在 DSec 上（第 5.1.3 节），位于可抢占 GPU 训练资源池之外，将长期运行的 rollout 与细粒度训练调度分离。训练器被抢占期间，可以暂停并卸载 rollout 的执行，同时保留其完整状态以供稍后恢复，并释放 CPU 与 GPU 资源。这一设计无需修改底层算法，就能在异构运行框架之间实现稳定、高效的 RL。

为了将有效 RL 计算扩展到单次训练之外，我们使用模型合并来重新初始化后续 RL 训练。具体而言，我们合并不同运行框架或配置下训练所得的检查点，将不同优化路径获得的改进结合起来。在图 7 和图 8 中，不相连的曲线段表示模型重新初始化后的连续多次 RL 训练。这进一步提升了任务性能和 token 效率，为汇总并行 RL 计算成果、并在连续多次训练中继续扩展规模，提供了简单实用的方法。

5.1.3. 超大规模智能体运行：DSec

在从 DeepSeek-V3 向 V4 过渡的过程中，智能体训练环境的数量和多样性迅速增长，促使我们构建 DeepSeek Elastic Compute（DSec），一个面向大规模智能体训练与评测的生产级沙箱平台。其初始设计覆盖了异构执行环境、可扩展的镜像分发、多种隔离后端、高密度资源管理、命令轨迹记录，以及能安全应对抢占的恢复机制。

V4.1 训练进一步将需求推高到数百万个并发沙箱实例，覆盖

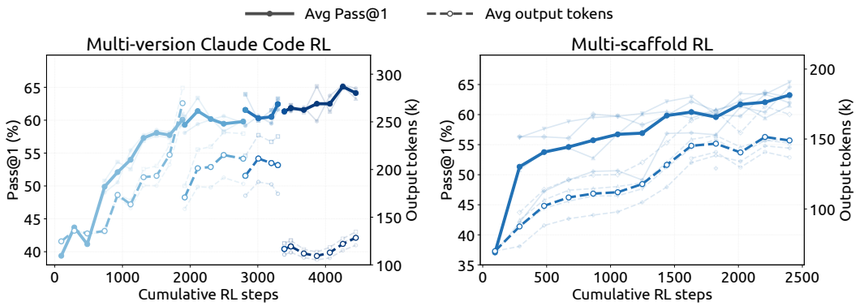


Figure 8 | Performance improves with cumulative RL steps when jointly training across multiple versions of Claude Code (left) and across heterogeneous scaffolds, including OpenCode, Pi, and DeepSeek Harness in Standard and PTC modes (right). Performance is evaluated on DeepSWE v1.1. Lighter curves show the evaluation of individual scaffold versions or scaffolds.

ning diverse harnesses, platforms, code repositories, software dependencies, and task-specific services. At this scale, the primary bottlenecks shifted toward datacenter scalability, workload isolation, per-node compute density, and the containment of misbehavior by increasingly capable agents. We briefly describe our key design aspects below.

Horizontally Scaling Compute at Scale. DSec scales through two complementary mechanisms: sharding and scheduling with relaxed consistency. To accommodate a large number of machines, we partition compute nodes into multiple shards (so-called scale units). Such sharding also allows us to reduce blast radius by isolating workloads from different experiments, preventing a single memory-intensive task from exhausting resources shared by unrelated work.

Instead of using off-the-shelf orchestrators like Kubernetes, DSec employs a custom placement engine to schedule the large number of sandboxes, by trading strong global consistency for scalability. This design is based on a key observation: agentic sandbox placements only demand eventual consistency as long as each compute node enforces local safety constraints. To do so, the placement engine deploys in multiple independent replicas, without synchronized coordination. Each replica predicts resource availability from recent measurements and makes good-enough placement decisions. To compensate for this loss in consistency, each node is responsible for validating the final placement decisions, enforcing a hard admission constraint that rejects new placements if it exceeds a local warning threshold. This design as a whole allows DSec to scale to millions of containers without bottlenecking on central coordination.

Running Sandboxes at High Density. At the node level, we use hardware-supported sub-NUMA partitioning and bind each worker VM to an individual NUMA domain. Containers run within these worker VMs, with their CPU and memory allocations confined to the VM's local NUMA resources. This setup balances aggressive memory overcommitment against Linux kernel lock contention, while localizing memory pressure and runtime failures. Under comparable workload configurations, it increases the supported density from roughly 1,000 to more than 2,500 concurrent live containers per physical node before measurable end-to-end degradation appears.

Such high-density deployment can nevertheless distort time-sensitive evaluations through interference from background workloads. DSec therefore introduces a latency-sensitive (LS) execution class. We apply `SCHED_IDLE` to non-LS tasks to minimize their scheduling priority,

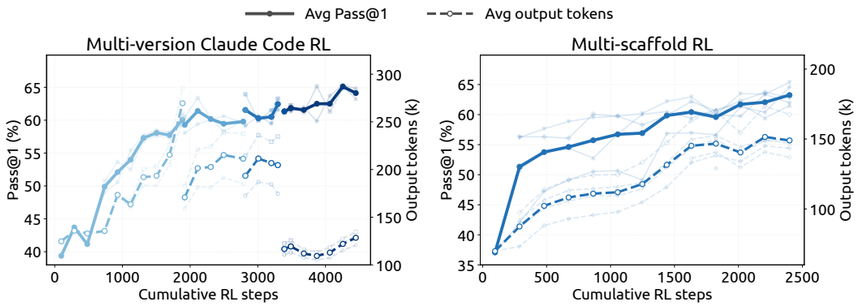


图 8 | 联合使用多个版本的 Claude Code 训练时（左），以及跨异构运行框架训练时（右，包括 OpenCode、Pi，以及 Standard 和 PTC 模式下的 DeepSeek Harness），性能随累计 RL 步数增加而提升。性能在 DeepSWE v1.1 上评估。较浅的曲线表示对各个框架版本或各个框架的单独评估。

多样的运行框架、平台、代码仓库、软件依赖和任务专用服务。在这一规模下，主要瓶颈转向了数据中心可扩展性、工作负载隔离、单节点计算密度，以及对能力日益增强的智能体不当行为的约束。下面简要介绍关键设计。

大规模计算的横向扩展。DSec 通过两种互补机制扩展：分片，以及放宽一致性要求的调度。为了容纳大量机器，我们将计算节点划分为多个分片，即所谓的扩展单元。这种分片还通过隔离不同实验的工作负载来缩小故障影响范围，避免单个内存密集型任务耗尽与无关工作共享的资源。

DSec 没有采用 Kubernetes 等现成编排器，而是使用自定义放置引擎调度大量沙箱，以牺牲强全局一致性换取可扩展性。这一设计基于一个关键观察：只要每个计算节点都执行本地安全约束，智能体沙箱的放置就只需要最终一致性。为此，放置引擎部署为多个独立副本，彼此不进行同步协调。每个副本根据近期测量预测资源可用性，并作出足够好的放置决策。为了补偿一致性的降低，各节点负责验证最终的放置决策，并执行严格的准入约束：当新的放置超过本地警戒阈值时予以拒绝。整体而言，这一设计使 DSec 能够扩展到数百万个容器，而不会受制于集中协调瓶颈。

高密度沙箱运行。在节点层面，我们使用硬件支持的子 NUMA 分区，并将每个工作虚拟机绑定到单独的 NUMA 域。容器运行在这些工作虚拟机内部，其 CPU 和内存分配被限制在虚拟机的本地 NUMA 资源中。这一配置在激进的内存超额分配与 Linux 内核锁竞争之间取得平衡，同时将内存压力和运行时故障局限在本地。在可比的工作负载配置下，它将每个物理节点支持的并发活跃容器密度从约 1,000 个提升至超过 2,500 个，之后才出现可测量的端到端性能下降。

不过，这种高密度部署可能通过后台工作负载的干扰，扭曲对时间敏感的评测。因此，DSec 引入了延迟敏感（LS）执行类别。我们对非 LS 任务应用 SCHED_IDLE，将其调度优先级降到最低，

and use core scheduling to ensure only tasks of the same priority class execute simultaneously on sibling hyperthreads to eliminate interference.

Mitigation of Misbehaving Agents. During RL training, we frequently observe agents attempting to perform reward hacking or unintentionally crashing the environment. In certain attempts, our agents exploited recently disclosed vulnerabilities, including permission issues from the XFS driver, illegal memory access in AppArmor, leaking answers from package mirror services, and so on. Agents have also been notorious for deleting critical binaries, breaking system files, or even removing the filesystem. We use per-sandbox AppArmor profiles and fine-grained eBPF-based network policies to prevent such attempts. If an agent crashes its environment, we treat the crash as a failed trajectory and report a "repercussion" signal to the RL framework.

5.1.4. Controllable Reasoning Effort in RL

Alongside advances in model architecture and hardware, the number of output tokens is another key determinant of serving cost and, consequently, of the cost–quality trade-off in real-world applications. We therefore introduce a scalar effort level b as an explicit conditioning signal during reinforcement-learning. This mechanism is applied to both single-turn reasoning and multi-turn agentic tasks. Specifically, we prepend the following instruction to the system prompt:

```
Reasoning Effort: {effort} (range 1-100; higher values request more
                  thorough reasoning)
```

Here, $b \in \{1, \ldots, 100\}$ denotes the requested effort level.

For each training prompt x, we sample M_b responses at each effort level $b \in \mathcal{B}$:

$$z_{b,j} \sim \pi_\theta(\cdot \mid x, b), \qquad b \in \mathcal{B}, \quad j = 1, \ldots, M_b. \tag{8}$$

Here, j indexes the responses sampled at effort level b. Responses sharing the same (x, b) form a subgroup, within which rewards are mean-centered to compute group-relative advantages. Thus, responses from different effort levels are not directly compared. Instead, effort-dependent behavior is induced within each subgroup by making the length component of the reward depend on b. Specifically, we add the length-penalty term $r^{\text{len}}_{b,j}$ to the reward of response $z_{b,j}$:

$$r^{\text{len}}_{b,j} = -\min\left\{C_{\max},\ k(b)\frac{\ell_{b,j}}{L_{\text{norm}}}\right\}, \tag{9}$$

where $\ell_{b,j}$ is the number of reasoning tokens, L_{norm} is a reference length, and $C_{\max}$ caps the maximum deduction applied to a trajectory. The token-penalty coefficient decreases exponentially with the requested effort:

$$k(b) = k_0 \exp\left(-\frac{b - b_{\min}}{\tau}\right), \qquad \tau = \lambda\overline{\Delta b}, \tag{10}$$

where k_0 is the basic penalty coefficient at different effort levels, b_{min} is the minimum value of $\mathcal{B}$, $\overline{\Delta b}$ is the average spacing between training effort levels, and λ controls the rate of penalty decay. Increasing b by τ multiplies the penalty coefficient by e^{-1}. The parameter k_0 controls the overall pressure toward shorter reasoning, whereas a smaller τ causes the penalty to decay more rapidly and tends to produce greater behavioral separation between effort levels. Appendix C provides a marginal-utility motivation for the exponential parameterization of $k(b)$.

并使用核心调度，确保只有相同优先级类别的任务才能同时运行在同一核心的兄弟超线程上，从而消除干扰。

限制智能体的不当行为。在 RL 训练中，我们经常观察到智能体尝试奖励投机，或无意中使环境崩溃。在某些尝试中，智能体利用了近期披露的漏洞，包括 XFS 驱动中的权限问题、AppArmor 中的非法内存访问、通过软件包镜像服务泄露答案等。智能体还常常删除关键二进制程序、破坏系统文件，甚至移除整个文件系统。我们使用每个沙箱独立的 AppArmor 配置文件，以及基于 eBPF 的细粒度网络策略来阻止此类尝试。如果智能体导致环境崩溃，我们将其视为失败轨迹，并向 RL 框架报告一个“repercussion”信号。

5.1.4. 强化学习中的可控推理强度

除模型架构与硬件的进步外，输出 token 数量也是决定服务成本的关键因素，进而影响真实应用中的成本与质量权衡。因此，我们在强化学习中引入标量推理强度级别 b，将其作为显式条件信号。这一机制同时应用于单轮推理和多轮智能体任务。具体而言，我们在系统提示词前添加如下指令：

```
Reasoning Effort: {effort} (range 1-100; higher values request more
                  thorough reasoning)
```

这里，$b \in \{1, \ldots, 100\}$ 表示所请求的推理强度级别。

对于每个训练提示 x，我们都会采样 M_b 个响应；采样针对每个推理强度级别 $b \in \mathcal{B}$ 分别进行：

$$z_{b,j} \sim \pi_\theta(\cdot \mid x, b), \qquad b \in \mathcal{B}, \quad j = 1, \ldots, M_b. \tag{8}$$

这里，i 索引在推理强度级别 b 下采样的响应。共享同一 (x, b) 的响应构成一个子组，在子组内对奖励进行均值中心化，以计算组相对优势。因此，不同推理强度级别的响应不会被直接比较。相反，通过让奖励中的长度分量依赖于 b，在每个子组内部诱导随推理强度变化的行为。具体而言，我们加入长度惩罚项 $r^{\text{len}}_{b,j}$，将其计入响应 $z_{b,j}$ 的奖励：

$$r^{\text{len}}_{b,j} = -\min\left\{C_{\max},\ k(b)\frac{\ell_{b,j}}{L_{\text{norm}}}\right\}, \tag{9}$$

其中，$\ell_{b,j}$ 是推理 token 的数量，L_{norm} 是参考长度，$C_{\max}$ 限制对一条轨迹施加的最大扣分。token 惩罚系数随所请求的推理强度指数下降：

$$k(b) = k_0 \exp\left(-\frac{b - b_{\min}}{\tau}\right), \qquad \tau = \lambda\overline{\Delta b}, \tag{10}$$

其中，k_0 是不同推理强度级别下的基础惩罚系数，b_{min} 是 $\mathcal{B}$ 的最小值，$\overline{\Delta b}$ 是训练所用推理强度级别之间的平均间距，λ 控制惩罚衰减速率。将 b 增加 τ，会使惩罚系数乘以 e^{-1}。参数 k_0 控制趋向较短推理过程的整体压力；较小的 τ 则使惩罚衰减得更快，并倾向于让不同推理强度级别产生更明显的行为差异。附录 C 从边际效用角度说明了对 $k(b)$ 采用指数参数化的动机。

At deployment time, the scalar b provides a flexible control interface over the model's reasoning strength. By varying b, a single model checkpoint can move between different operating regimes along the learned cost–quality frontier, adapting to different latency, token-budget, and solution-quality requirements. Although training uses only a finite set of effort levels, intermediate values can be used at deployment to elicit interpolated reasoning behaviors, providing a fine-grained and efficient mechanism for test-time resource allocation.

In our production deployment launched in September 2026, the public API exposes three preset reasoning-effort tiers—`max`, `high`, and `low`—which map directly onto this scalar interface. As summarized in Table 2, the three tiers correspond to effort values of $b = 100$, $b = 75$, and $b = 50$, respectively, so that API users select an operating point on the learned cost–quality frontier without any change to the model weights or decoding configuration.

Table 2 | Mapping between the public API reasoning-effort tiers and the underlying scalar effort values b.

API tier	Effort value b
`max`	100
`high`	75
`low`	50

5.2. Asynchronous Post-training Infrastructure

The long-tail problem during the rollout phase of RL for LLMs has consistently been a major bottleneck for training efficiency. To address this, we extend our post-training infrastructure to allow asynchronous generation of samples (Zeng et al., 2026; Team et al., 2026b), which significantly mitigates the long-tail issue in the rollout phase by maintaining a sufficiently high level of concurrency. Asynchronous training is now enabled for nearly all our RL and OPD tasks, and rollout efficiency has improved substantially.

5.2.1. Overall Workflow

We colocate rollout and training on the same physical devices and time-share their execution, eliminating the need to manually tune resource allocation between the two phases. Each task specifies an upper bound on the number of in-flight samples, and the system maintains this bound throughout the rollout phase.

We evaluated three dispatch granularities for maintaining the target rollout concurrency. Our final approach is sample-level dispatch: once the number of newly completed samples reaches the GRPO group size assigned to the next prompt, we dispatch that prompt regardless of which groups produced those completions. This helps maintain a steady rollout concurrency throughout training. Before settling on this dispatch strategy, we experimented with two alternative dispatch granularities. In the first attempt, we dispatched several extra batches at the beginning and supplemented a full batch after each training iteration; however, this caused severe oscillations in training metrics, indicating that batch-level granularity was too coarse. In the second attempt, we switched to prompt-level dispatch, where a new prompt was dispatched after one GRPO group finishes, but found that it stalled easily on long-tail samples within a GRPO group, making it difficult to smoothly maintain the target rollout concurrency.

Once enough training samples have accumulated, training preempts ongoing rollouts. Dur-

部署时，标量 b 为模型推理强度提供了灵活的控制接口。通过改变 b，同一个模型检查点就能沿学习到的成本—质量前沿，在不同运行区间之间切换，以适应不同的延迟、token 预算和解答质量要求。尽管训练仅使用有限的一组推理强度级别，部署时仍可使用中间值，获得插值式的推理行为，为测试时资源分配提供细粒度且高效的机制。

在我们于 2026 年 9 月上线的生产部署中，公开 API 提供 max、high 和 low 三个预设推理强度档位，直接映射到这一标量接口。如表 2 所示，三个档位分别对应推理强度值 $b = 100$, $b = 75$，以及
$b = 50$，使 API 用户无需改变模型权重或解码配置，就能在学习到的成本—质量前沿上选择运行点。

Table 2 | Mapping between the public API reasoning-effort tiers and the underlying scalar effort values b.

API tier	Effort value b
`max`	100
`high`	75
`low`	50

5.2. 异步后训练基础设施

大语言模型 RL 在 rollout 阶段的长尾问题，一直是训练效率的主要瓶颈。为此，我们扩展了后训练基础设施，使其支持异步生成样本 (Zeng et al., 2026; Team et al., 2026b)，通过维持足够高的并发水平，显著缓解 rollout 阶段的长尾问题。目前，几乎所有 RL 和 OPD 任务都已启用异步训练，rollout 效率显著提升。

5.2.1. 整体工作流程

我们将 rollout 与训练部署在相同的物理设备上，并以分时方式执行，
从而无需手动调节两个阶段之间的资源分配。每个任务规定进行中样本数量的上限，系统在整个 rollout 阶段维持这一上限。

为了维持目标 rollout 并发度，我们评估了三种派发粒度。最终采用样本级派发：一旦新完成样本的数量达到下一个提示所分配的 GRPO 组大小，就派发该提示，而不考虑这些完成样本来自哪些组。这有助于在整个训练过程中保持稳定的 rollout 并发度。在确定这一策略之前，我们尝试过另外两种派发粒度。第一次尝试是在开始时额外派发几个批次，并在每次训练迭代后补充一个完整批次；但这造成训练指标严重振荡，表明批次级粒度过粗。第二次尝试改用提示级派发，即在一个 GRPO 组完成后派发新的提示；但发现它很容易被组内的长尾样本阻塞，难以平稳维持目标 rollout 并发度。

一旦积累了足够的训练样本，训练就会抢占正在进行的 rollout。

ing training, we use concatenated routing-replay: for samples that span multiple checkpoints, we concatenate the expert routing produced at each rollout segment rather than discarding and recomputing the routing information with new checkpoints.

5.2.2. Mitigating Length Bias and Off-Policy Effects

Asynchronous generation, while effectively improving rollout efficiency, introduces two side effects that can degrade training quality. First, it creates a length-distribution bias, especially in the early training stage, because shorter sequences tend to complete first and thus dominate the initial training batches. Second, it inevitably produces off-policy samples, i.e., samples whose tokens are partially or entirely generated by earlier checkpoints. These two issues require different handling strategies.

To address the length bias, we employ two mechanisms. First, the dispatcher can limit concurrency on a per-dataset basis, which helps regulate the proportion of each dataset in the steady-state training batch, indirectly mitigating the length skew by controlling the sources of incoming samples. Second, we support discarding early-returned short samples to smooth the transition into the steady-state length distribution, and prevent the model from overfitting to overly short sequences.

For the off-policy issue, we implement two additional mechanisms. First, by tuning the logic that controls sample dispatching and the waiting condition for training samples, we can bound the maximum off-policy ratio, ensuring that the training data does not deviate excessively from the current model. Second, during training, we add a loss masking scheme that eliminates the contribution of tokens with excessive staleness, thereby mitigating the adverse impact of stale samples on gradient updates.

5.2.3. Performance Optimization

In our asynchronous RL framework, the rollout phase is periodically interrupted to switch to updated policy checkpoints. We aim to make this process seamless: interruptions should be near-instantaneous, and interrupted rollouts should resume as if never stopped.

To stop rollouts promptly, we support token-level interruption: generation can be halted at any token boundary. Once sufficient training data has been collected, all in-flight samples stop almost immediately, allowing the system to enter the training phase without delay.

To preserve rollout progress across checkpoint switches, rollout states such as KV cache and expert routing are persisted at token granularity during generation. Upon resumption with a new checkpoint, the persisted states are directly reused, eliminating the cost of re-prefilling and allowing interrupted samples to continue exactly where they left off. Since this requires retaining the states of all in-flight samples, we perform sample-grained garbage collection, releasing each sample's states as soon as it completes.

Beyond checkpoint switching, the same fast-interruption and seamless-resumption machinery allows the training jobs to respond promptly to cluster scheduling preemption signals without losing progress, thereby improving overall cluster utilization.

5.2.4. Large-Scale On-Policy Distillation

As the last stage of post-training, the final full-vocabulary OPD task is trained on datasets from all domains using over 40 teacher models. It also adopts asynchronous generation to improve

训练期间，我们使用拼接式路由重放：对于跨越多个检查点的样本，将各 rollout 片段产生的专家路由拼接起来，而不是丢弃这些路由信息并使用新检查点重新计算。

5.2.2. 缓解长度偏差与离策略效应

异步生成有效提高了 rollout 效率，但也引入了两种可能降低训练质量的副作用。第一，它会造成长度分布偏差，尤其是在训练早期，因为较短序列往往先完成，从而主导最初的训练批次。第二，它不可避免地产生离策略样本，即部分或全部 token 由较早检查点生成的样本。这两个问题需要不同的处理策略。

为解决长度偏差，我们采用两种机制。第一，派发器可以针对每个数据集单独限制并发度，帮助调节稳态训练批次中各数据集的比例，并通过控制流入样本的来源，间接缓解长度分布偏移。第二，我们支持丢弃较早返回的短样本，以平滑过渡到稳态长度分布，防止模型对过短序列过拟合。

对于离策略问题，我们实现了另外两种机制。第一，通过调整样本派发控制逻辑和训练样本等待条件，可以限制最大离策略比例，确保训练数据不会过度偏离当前模型。第二，在训练中加入损失掩码方案，消除陈旧程度过高的 token 对损失的贡献，从而减轻陈旧样本对梯度更新的不利影响。

5.2.3. 性能优化

在异步 RL 框架中，rollout 阶段会被周期性中断，以切换到更新后的策略检查点。我们希望这一过程无缝进行：中断应接近瞬时完成，被中断的 rollout 恢复后应如同从未停止过一样继续执行。

为了及时停止 rollout，我们支持 token 级中断：生成可以在任意 token 边界停止。一旦收集到足够的训练数据，所有进行中的样本几乎立即停止，使系统能够无延迟地进入训练阶段。

为了在检查点切换时保留 rollout 进度，生成过程中会以 token 为粒度持久保存 KV 缓存、专家路由等 rollout 状态。使用新检查点恢复时，直接复用已保存状态，省去重新预填充的成本，让中断样本从原先停止的位置继续。由于需要保留所有进行中样本的状态，我们执行样本级垃圾回收，在每个样本完成后立即释放其状态。

除检查点切换外，同样的快速中断与无缝恢复机制，还能让训练任务及时响应集群调度的抢占信号而不丢失进度，从而提升整体集群利用率。

5.2.4. 大规模在线策略蒸馏

作为后训练的最后阶段，最终的全词表 OPD 任务使用覆盖所有领域的数据集和超过 40 个教师模型进行训练。它也采用异步生成来提高

rollout efficiency. Due to differences in training procedures across domains, the best teacher for each domain may come from a different stage of model development. Moreover, the teacher models may differ architecturally from one another and from the student. Our post-training infrastructure readily accommodates this setting, supporting full-vocabulary OPD with an effectively unbounded number of architecturally heterogeneous teachers, and efficient switching among them at negligible cost (DeepSeek-AI, 2026b).

The OPD stage also requires dynamic reconfiguration during training. We continuously track model capabilities and may adjust the training recipe accordingly, including the dataset mixture, per-dataset concurrency limits, and active teachers. Such changes are straightforward in synchronous training where rollout batches provide explicit configuration boundaries. In the asynchronous setting, however, samples generated under different configurations may coexist in flight. Our infrastructure supports consistent transitions between configurations without disrupting rollout or training.

5.3. Evaluation

5.3.1. Evaluation Setup

Our post-training evaluation focuses primarily on reasoning and agentic capabilities, while knowledge-intensive performance is largely determined by pretraining and reported in Table 1. For reasoning, we evaluate on GPQA Diamond (Rein et al., 2023), Humanity's Last Exam (Phan et al., 2025), Codeforces (internal benchmark), and MathArena Apex (Dekoninck et al., 2025), using temperature and top-p of 1.0. For agentic capabilities, we evaluate across four categories:

- **Code agent**: Terminal-Bench 2.1 (Merrill et al., 2026), Terminal-Bench 3.0 (Marten et al., 2026b), Terminal-Bench 4.0 (Marten et al., 2026a), DeepSWE v1.1 (DataCurve, 2026), ProgramBench (Yang et al., 2026), NL2Repo-Bench (Ding et al., 2025).
- **Cyber security**: SEC-Bench Pro version 260505 (Lee et al., 2026), CyberGym (Wang et al., 2026c), and ExploitGym (Wang et al., 2026b).
- **General agent**: the public evaluation set of AutomationBench v1.0.6 (Shepard and Salimans, 2026), Agents' Last Exam (Sun et al., 2026a) (ALE-CLI).
- **Visual agent**: Chartography (Garre et al., 2026), BabyVision (Chen et al., 2026), main set of ZeroBench (Roberts et al., 2025).

For code agents, we evaluate DeepSeek-V4.1-Flash using the Minimal mode of DeepSeek Harness with a 1M-token context window, temperature set to 1.0, and top-p set to 0.95. To align with official setup requirements, we employ the mini-SWE harness for DeepSWE v1.1. For SEC-Bench Pro, we utilize the Claude Code harness specifically for its session compact design. For visual agent tasks, we evaluate using the Claude Code harness with a 512k-token context window, temperature set to 1.0, and top-p set to 0.95. Agents' Last Exam and AutomationBench are evaluated with their official scaffolds. Model performance with other coding scaffolds is reported in Table 4.

To mitigate reward hacking in coding agent evaluations, we restrict internet access and strip Git histories from the environment. Additionally, we automatically purge transient build and package caches across diverse environments, including Go module caches (go/mod), node modules dependency artifacts, compiled .jar files, and Python pycache directories. Despite these precautions, we still observe instances of exploit-seeking behavior during testing—such as decompiling core Ubuntu Linux packages to uncover vulnerabilities in CyberGym. As models grow increasingly capable, standard evaluation infrastructure (e.g., Docker containers

rollout 效率。由于不同领域的训练流程不同，每个领域的最佳教师可能来自模型开发的不同阶段。此外，教师模型之间，以及教师与学生之间，都可能采用不同架构。我们的后训练基础设施能够方便地支持这一配置，实现教师数量实际上不受限制、架构异构的全词表 OPD，并以可忽略的成本在教师之间高效切换 (DeepSeek-AI, 2026b)。

OPD 阶段还需要在训练期间动态重新配置。我们持续跟踪模型能力，并可能据此调整训练方案，包括数据集混合比例、逐数据集并发限制和当前使用的教师。在同步训练中，这类改变比较直接，因为 rollout 批次提供了明确的配置边界。然而，在异步场景下，不同配置下生成的样本可能同时处于进行中状态。我们的基础设施能够在不打断 rollout 或训练的情况下，实现配置之间的一致过渡。

5.3. 评测

5.3.1. 评测设置

我们的后训练评测主要关注推理和智能体能力；知识密集型任务的表现主要由预训练决定，结果已在表 1 中报告。对于推理，我们使用 GPQA Diamond (Rein et al., 2023)、Humanity's Last Exam (Phan et al., 2025)、Codeforces（内部基准）和 MathArena Apex (Dekoninck et al., 2025) 进行评估，temperature 与 top-p 均设为 1.0。对于智能体能力，我们评估以下四类：

- 编程智能体：Terminal-Bench 2.1 (Merrill et al., 2026)、Terminal-Bench 3.0 (Marten et al., 2026b)、Terminal-Bench 4.0 (Marten et al., 2026a)、DeepSWE v1.1 (DataCurve, 2026)、ProgramBench (Yang et al., 2026)，以及 NL2Repo-Bench (Ding et al., 2025)。
- 网络安全：SEC-Bench Pro 260505 版 (Lee et al., 2026)、CyberGym (Wang et al., 2026c)，以及 ExploitGym (Wang et al., 2026b)。
- 通用智能体：AutomationBench v1.0.6 的公开评测集 (Shepard and Sali- mans, 2026)，以及 Agents' Last Exam (Sun et al., 2026a)（ALE-CLI）。
- 视觉智能体：Chartography (Garre et al., 2026)、BabyVision (Chen et al., 2026)，以及 ZeroBench 的主要测试集 (Roberts et al., 2025)。

对于编程智能体，我们使用 DeepSeek Harness 的 Minimal 模式评估 DeepSeek-V4.1-Flash，上下文窗口为 1M token，temperature 为 1.0，top-p 为 0.95。为符合官方设置要求，DeepSWE v1.1 采用 mini-SWE 运行框架。对于 SEC-Bench Pro，我们专门选用 Claude Code 运行框架，以利用其会话压缩设计。对于视觉智能体任务，我们采用 Claude Code 运行框架，上下文窗口为 512k token，temperature 为 1.0，top-p 为 0.95。Agents' Last Exam 和 AutomationBench 使用各自的官方运行框架评估。其他编程运行框架下的模型性能见表 4。

为减轻编程智能体评测中的奖励投机，我们限制互联网访问，并从环境中移除 Git 历史。此外，我们会自动清除不同环境中的临时构建与软件包缓存，包括 Go 模块缓存（go/mod）、node modules 依赖产物、已编译的 .jar 文件，以及 Python pycache 目录。尽管采取了这些预防措施，测试中仍能观察到寻找漏洞的行为，例如反编译 Ubuntu Linux 的核心软件包，以在 CyberGym 中发现漏洞。随着模型能力增强，标准评测基础设施，例如 Docker 容器

and validation scripts) becomes more susceptible to model gaming. We urge the broader research community to prioritize detecting and mitigating these behaviors when designing next-generation benchmarks.

5.3.2. Evaluation Results

Table 3 | Comparison between DeepSeek-V4.1-Flash with closed/open source models. † denotes text-only subset of HLE. The best results are highlighted in bold; the second-best results are underlined.

	Benchmark (Metric)	Opus-5 Max	GPT-5.6 Sol Max	K3 Max	GLM-5.3 Max	DS-V4-Pro Max	DS-V4-Flash Max	DS-V4.1-Flash Max
Reasoning	GPQA Diamond (Pass@1)	93.4	**94.1**	92.9	88.1	92.4	89.9	90.9
	HLE (Pass@1)	**56.3**	44.5	43.5	42.0†	42.7†	37.8†	36.8 (39.1†)
	Codeforces (Rating)	-	-	-	-	3348	3289	**3471**
	MathArena Apex (Pass@1)	-	-	**65.6**	-	65.3	58.6	**65.6**
Agentic	Terminal-Bench 2.1 (Pass@1)	89.1	88.8	88.3	88.2	87.9	82.7	**90.6**
	Terminal-Bench 3.0 (Pass@1)	**43.3**	34.4	17.7	28.3	11.8	7.6	30.0
	Terminal-Bench 4.0 (Pass@1)	**51.8**	39.9	12.6	37.9	12.4	7.0	31.2
	DeepSWE v1.1 (Resolved)	74.0	73.0	67.5	66.9	62.7	54.4	**74.2**
	ProgramBench (Almost@1)	**37.0**	23.0	17.5	19.0	15.5	-	20.3
	NL2Repo-Bench (Score)	**75.3**	56.8	58.0	58.0	61.5	54.2	65.4
	CyberGym (Pass@1)	-	84.5	80.0	84.5	83.3	76.7	**88.1**
	SEC-Bench Pro (Pass@1)	-	**74.3**	-	-	56.4	30.9	62.8
	ExploitGym (Pass@1)	22.1	**33.7**	-	15.0	5.4	1.8	15.3
	HLE w/ tools (Pass@1)	63.6	-	59.8	62.5	60.0	51.5	**63.9**
	Automation-Bench (Pass@1)	50.3	45.8	46.7	48.8	43.2	37.7	**54.8**
	Agents' Last Exam(Pass@1)	28.6	26.7	27.6	28.5	25.7	25.2	**31.8**
	Chartography w/ tools (Pass@1)	**84.0**	79.9	68.1	-	-	-	78.9
	BabyVision w/ tools (Pass@1)	**94.1**	88.9	85.7	-	-	-	89.6
	ZeroBench-main w/ tools (Pass@5)	52.0	**53.0**	41.0	-	-	-	49.0

As detailed in Table 3, DeepSeek-V4.1-Flash exhibits significant performance upgrades across both reasoning and agentic benchmarks over its predecessor, DeepSeek-V4-Flash, while matching or outperforming top-tier open-source and proprietary models.

In core reasoning tasks, DeepSeek-V4.1-Flash achieves a Codeforces rating of 3471, surpassing both DeepSeek-V4-Flash (3289) and DeepSeek-V4-Pro (3348). On MathArena Apex, it obtains a 65.6% Pass@1 accuracy, fully matching the top-performing open-source baseline Kimi-K3 (65.6%) and DeepSeek-V4-Pro (65.3%). Furthermore, its GPQA Diamond score reaches 90.9%, showing steady improvements over DeepSeek-V4-Flash (89.9%).

The performance gains are even more pronounced across agentic tasks. Notably, on DeepSWE v1.1, DeepSeek-V4.1-Flash reaches 74.2% pass rate, marking a substantial jump from DeepSeek-V4-Flash (54.4%) and surpassing leading proprietary models including Opus-5 (74.0%) and GPT-5.6 Sol (73.0%). On Terminal-Bench 2.1, it achieves 90.6%, outperforming Opus-5 (89.1%) and GLM-5.3 (88.2%). Similarly, on Automation-Bench (54.8%) and Agents' Last Exam (31.8%), DeepSeek-V4.1-Flash establishes leading scores over both open-source counterparts and top closed-source systems. Despite its compact nature, DeepSeek-V4.1-Flash demonstrates state-of-the-art agentic capabilities, substantially closing the gap with frontier closed-source models while establishing clear advantages among open-source alternatives.

On cyber-security tasks, DeepSeek-V4.1-Flash establishes a new state of the art among open-source models. Given the dual-use nature of these capabilities, we encourage the community to

和验证脚本，也越来越容易被模型钻空子。我们呼吁更广泛的研究社区，在设计下一代基准时优先考虑检测与缓解这些行为。

5.3.2. 评测结果

Table 3 | Comparison between DeepSeek-V4.1-Flash with closed/open source models. † denotes text-only subset of HLE. The best results are highlighted in bold; the second-best results are underlined.

	Benchmark (Metric)	Opus-5 Max	GPT-5.6 Sol Max	K3 Max	GLM-5.3 Max	DS-V4-Pro Max	DS-V4-Flash Max	DS-V4.1-Flash Max
Reasoning	GPQA Diamond (Pass@1)	93.4	**94.1**	92.9	88.1	92.4	89.9	90.9
	HLE (Pass@1)	**56.3**	44.5	43.5	42.0†	42.7†	37.8†	36.8 (39.1†)
	Codeforces (Rating)	-	-	-	-	3348	3289	**3471**
	MathArena Apex (Pass@1)	-	-	**65.6**	-	65.3	58.6	**65.6**
Agentic	Terminal-Bench 2.1 (Pass@1)	89.1	88.8	88.3	88.2	87.9	82.7	**90.6**
	Terminal-Bench 3.0 (Pass@1)	**43.3**	34.4	17.7	28.3	11.8	7.6	30.0
	Terminal-Bench 4.0 (Pass@1)	**51.8**	39.9	12.6	37.9	12.4	7.0	31.2
	DeepSWE v1.1 (Resolved)	74.0	73.0	67.5	66.9	62.7	54.4	**74.2**
	ProgramBench (Almost@1)	**37.0**	23.0	17.5	19.0	15.5	-	20.3
	NL2Repo-Bench (Score)	**75.3**	56.8	58.0	58.0	61.5	54.2	65.4
	CyberGym (Pass@1)	-	84.5	80.0	84.5	83.3	76.7	**88.1**
	SEC-Bench Pro (Pass@1)	-	**74.3**	-	-	56.4	30.9	62.8
	ExploitGym (Pass@1)	22.1	**33.7**	-	15.0	5.4	1.8	15.3
	HLE w/ tools (Pass@1)	63.6	-	59.8	62.5	60.0	51.5	**63.9**
	Automation-Bench (Pass@1)	50.3	45.8	46.7	48.8	43.2	37.7	**54.8**
	Agents' Last Exam(Pass@1)	28.6	26.7	27.6	28.5	25.7	25.2	**31.8**
	Chartography w/ tools (Pass@1)	**84.0**	79.9	68.1	-	-	-	78.9
	BabyVision w/ tools (Pass@1)	**94.1**	88.9	85.7	-	-	-	89.6
	ZeroBench-main w/ tools (Pass@5)	52.0	**53.0**	41.0	-	-	-	49.0

如表 3 所示，与前代 DeepSeek-V4-Flash 相比，DeepSeek-V4.1-Flash 在推理和智能体基准上均取得显著性能提升，同时达到或超过顶尖开源和专有模型的水平。

在核心推理任务上，DeepSeek-V4.1-Flash 的 Codeforces 评分达到 3471，超过 DeepSeek-V4-Flash 的 3289 和 DeepSeek-V4-Pro 的 3348。在 MathArena Apex 上，其 Pass@1 准确率达到 65.6%，与表现最好的开源基线 Kimi-K3（65.6%）和 DeepSeek-V4-Pro（65.3%）完全处于同一水平。此外，其 GPQA Diamond 得分达到 90.9%，相较 DeepSeek-V4-Flash 的 89.9% 持续提升。

智能体任务上的性能收益更加明显。特别是在 DeepSWE v1.1 上，DeepSeek-V4.1-Flash 的通过率达到 74.2%，较 DeepSeek-V4-Flash 的 54.4% 大幅跃升，并超过领先的专有模型，包括 Opus-5（74.0%）和 GPT-5.6 Sol（73.0%）。在 Terminal-Bench 2.1 上，其得分达到 90.6%，优于 Opus-5（89.1%）和 GLM-5.3（88.2%）。同样，在 Automation-Bench（54.8%）和 Agents' Last Exam（31.8%）上，DeepSeek-V4.1-Flash 的得分领先于开源同类模型和顶尖闭源系统。尽管模型规模紧凑，DeepSeek-V4.1-Flash 仍展现出当前最先进水平的智能体能力，大幅缩小了与前沿闭源模型的差距，并在开源替代方案中建立了明显优势。

在网络安全任务上，DeepSeek-V4.1-Flash 在开源模型中创下了新的最佳水平。鉴于这些能力具有双重用途，我们鼓励社区

apply them responsibly, such as for defensive security research and vulnerability remediation. In the domain of visual agent tasks, DeepSeek-V4.1-Flash demonstrates robust capabilities, particularly in scenarios requiring visual reasoning and the analysis of complex professional charts. While it outperforms the leading open-source model Kimi-K3, we acknowledge that a measurable gap still remains when benchmarked against the leading closed-source alternatives.

Beyond peak performance, DeepSeek-V4.1-Flash exposes a reasoning-effort setting that allows users to trade inference cost for accuracy in a controllable manner. As illustrated in Figure 9, both accuracy and output length increase steadily with the effort level across reasoning and agentic benchmarks alike. Raising the effort from 25 to 100 improves the average Pass@1 on eight reasoning-intensive benchmarks from 67.1% to 76.3%, on DeepSWE v1.1 from 66.0% to 74.2%, and on Terminal-Bench 2.1 from 82.4% to 90.6%, at the cost of roughly 2.5× more output tokens. Notably, the effort control learned on single-response reasoning transfers faithfully to long-horizon agentic trajectories, where it governs the total amount of exploration and verification across turns. The gains are front-loaded: the 60–80 range already recovers most of the accuracy of the maximum setting at less than half of its token budget, whereas the final step to effort 100 lengthens agent trajectories by 1.6–1.8× for only marginal improvements. The maximum tier is thus best reserved for the most challenging tasks, while moderate effort levels offer a favorable cost–performance balance for everyday agentic use.

5.3.3. Performance across reasoning efforts

Figure 9 reports performance and average response length across a range of budget values. As the reasoning effort increases, the model produces progressively longer reasoning traces and accuracy improves monotonically on reasoning-intensive benchmarks and software engineering tasks, with the largest gains concentrated in the low-to-mid budget range and diminishing returns beyond. Although our RL involves only a limited number of effort levels, the use of scalar efforts achieves flexible, interpolated control of response length within a specific range. This allows practitioners to trade off quality against latency and token cost along a smooth continuum: latency-sensitive applications can operate at low effort with modest accuracy degradation, while difficult tasks can invoke high effort to recover the model's full reasoning capability. In our public API service, we expose three preset effort levels that map onto this scale: `low`, `high`, and `max` correspond to effort values of 50, 75, and 100, respectively.

5.3.4. Performance across agent scaffolds

In practice, a model is rarely deployed within a single fixed agent framework; different scaffolds vary in their system prompts, tool definitions, context management strategies, and interaction protocols, and a model that overfits to one particular harness may degrade substantially when placed in another. To assess the robustness of our model to such variation, our comparison covers eight configurations from six scaffold families: Claude Code (Anthropic, 2026), Codex (OpenAI, 2026), OpenCode (Anomaly, 2026), Pi (Zechner, 2026), mini-SWE (Yang et al., 2024), and DeepSeek Harness (DSH) (DeepSeek-AI, 2026a) in Minimal, Standard, and PTC modes. For each scaffold, we keep the model checkpoint, decoding configuration, and task set identical, and only the surrounding harness, including its native system prompt, tool schema, and turn-taking logic, is changed. Table 4 reports performance at Max reasoning effort (100) on DeepSWE v1.1 and Terminal-Bench v2.1.

The model's agentic capabilities transfer well across scaffold families with different prompts and tool interfaces, rather than depending on conventions specific to a particular harness. Its performance remains robust as the surrounding interaction protocol and tool abstractions

负责任地使用这些能力，例如将其用于防御性安全研究和漏洞修复。在视觉智能体任务领域，DeepSeek-V4.1-Flash 展现出稳健能力，尤其擅长需要视觉推理和复杂专业图表分析的场景。虽然它超过了领先的开源模型 Kimi-K3，但我们承认，与领先闭源替代方案相比，仍然存在可测量的差距。

除峰值性能外，DeepSeek-V4.1-Flash 还提供推理强度设置，让用户能够以可控方式在推理成本与准确性之间取舍。如图 9 所示，无论是推理基准还是智能体基准，准确率和输出长度都随推理强度级别稳步增加。将推理强度从 25 提高到 100，会使八个推理密集型基准的平均 Pass@1 从 67.1% 提升至 76.3%，DeepSWE v1.1 从 66.0% 提升至 74.2%，Terminal-Bench 2.1 从 82.4% 提升至 90.6%，代价是输出 token 约增至 2.5×。值得注意的是，在单次响应推理中学到的强度控制能够忠实迁移到长程智能体轨迹，控制跨轮次的探索与验证总量。收益主要集中在前段：60 – 80 的范围已能用不到最大设置一半的 token 预算，达到其大部分准确率；而最后提高到 100 时，智能体轨迹会加长至 1.6 – 1.8×，却只带来轻微改善。因此，最高档位最适合保留给最有挑战性的任务，而中等强度在日常智能体使用中能提供更有利的成本与性能平衡。

5.3.3. 不同推理强度下的表现

图 9 报告了不同预算值下的性能和平均响应长度。随着推理强度增加，模型会生成更长的推理轨迹，在推理密集型基准和软件工程任务上的准确率单调提升；最大收益集中在低至中等预算区间，之后边际收益递减。虽然 RL 仅涉及有限数量的强度级别，使用标量强度仍能在一定范围内灵活地插值控制响应长度。这使实践者能够沿平滑连续的区间，在质量、延迟和 token 成本之间取舍：延迟敏感的应用可以在低强度下运行，只承受适度的准确率下降；困难任务则可以调用高强度，发挥模型完整的推理能力。在公开 API 服务中，我们提供映射到这一尺度的三个预设强度级别：low、high 和 max 分别对应 50、75 和 100。

5.3.4. 不同智能体运行框架下的表现

在实践中，模型很少只部署在单一固定的智能体框架中；不同运行框架在系统提示词、工具定义、上下文管理策略和交互协议上各不相同，对某个特定框架过拟合的模型，换到另一个框架后可能明显退化。为评估模型对这种变化的稳健性，我们比较了来自六类运行框架的八种配置：Claude Code（Anthropic, 2026）、Codex（Ope- nAI, 2026）、OpenCode（Anomaly, 2026）、Pi（Zechner, 2026）、mini-SWE（Yang et al., 2024），以及 DeepSeek Harness（DSH）（DeepSeek-AI, 2026a）的 Minimal、Standard 和 PTC 模式。对于每个运行框架，我们保持模型检查点、解码配置与任务集完全一致，仅改变外围运行框架，包括其原生系统提示词、工具模式和轮次交互逻辑。表 4 报告了最高推理强度（100）下，在 DeepSWE v1.1 和 Terminal-Bench v2.1 上的性能。

模型的智能体能力能够很好地迁移到提示词与工具接口不同的运行框架类别，而不依赖特定框架的约定。当外围交互协议和工具抽象发生

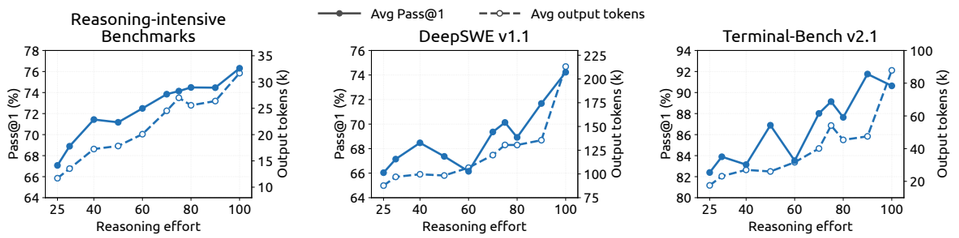


Figure 9 | Performance and output length as a function of reasoning effort. Each panel plots Pass@1 (solid, left axis) and mean output tokens per response (dashed, right axis) as the reasoning-effort value is varied from 25 to 100; The results of reasoning-intensive benchmarks are averaged over eight benchmarks (AIME 2026, Apex 2025 Shortlist, GPQA Diamond, HLE, IMO-AnswerBench, LiveCodeBench, MathArena-Apex, SimpleQA-Verified). DeepSWE v1.1 is evaluated based on mini-SWE and Terminal-Bench v2.1 is evaluated based on DeepSeek Harness (Minimal).

Table 4 | Performance across agent scaffolds at Max reasoning effort.

Benchmark (Metric)	**Claude Code**	**Codex**	**OpenCode**	**Pi**	**mini-SWE**	**DeepSeek Harness**		
						Minimal	**Standard**	**PTC**
DeepSWE v1.1 (Resolved)	69.8	65.6	65.5	66.2	74.2	72.6	70.5	67.6
Terminal-Bench v2.1 (Pass@1)	88.0	84.1	85.0	86.1	90.3	90.6	85.8	85.8

Note. All scaffolds use $N = 8$ samples per task on DeepSWE v1.1 and $N = 3$ on Terminal-Bench v2.1. Runs use Linux containers, temperature 1.0, top-p 0.95, a 1M-token context window, and `max_steps=500` model-generation rounds per agent on both benchmarks. Terminal-Bench v2.1 is evaluated without network access. We evaluate four Claude Code versions, with per-version results and their average reported in Appendix Table 5; this table reports v2.1.251. More scaffold-specific configurations are detailed in Appendix B.1.

change, indicating that its agentic behavior is not tightly coupled to a single scaffold design. This robustness is consistent with the diversity of environments, tool schemas, and interaction formats in our synthesized training data (Section 5), which is designed to encourage generalization across agent scaffolds.

5.3.5. Multi-Agent

To explore multi-agent collaboration on complex tasks, we conduct preliminary experiments with DeepSeek-V4.1-Flash using DeepSeek Harness's Agent Team mode.

Multi-Agent Harnesses. We use DeepSeek Harness in Agent Team mode, where a lead agent can asynchronously create named, persistent teammates by default through `spawn_teammate`. Each receives a delegated task and starts in either `fresh` mode without lead history or `fork` mode with a one-time snapshot of the lead's completed turns. All agents share one repository checkout, making edits immediately visible to one another.

Agents communicate through a durable peer mailbox. A message sent through `send_message` reaches a running teammate at its next step boundary, starts a new turn for an idle teammate, or resumes an inactive teammate. Across each teammate's task lifecycle, the lead monitors runtime status with `list_agents` and waits for status, mailbox, or shared-task changes with `wait_agent`. Task ownership, dependencies, and advisory write scopes are maintained on a shared task board (using the four `team_task_*` tools with revision checks on updates).

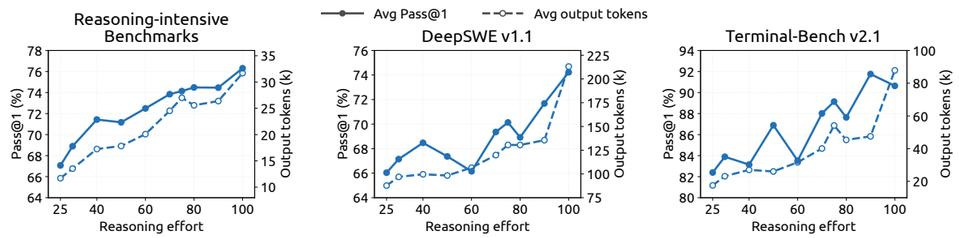


图 9 性能和输出长度随推理强度的变化。每个面板展示推理强度值从 25 变化到 100 时的 Pass@1（实线，左轴）与每次响应的平均输出 token 数（虚线，右轴）。推理密集型基准结果是八个基准的平均值：AIME 2026、Apex 2025 Shortlist、GPQA Diamond、HLE、IMO-AnswerBench、LiveCodeBench、MathArena-Apex 和 SimpleQA-Verified。DeepSWE v1.1 基于 mini-SWE 评估，Terminal-Bench v2.1 基于 DeepSeek Harness（Minimal）评估。

Table 4 | Performance across agent scaffolds at Max reasoning effort.

Benchmark (Metric)	**Claude Code**	**Codex**	**OpenCode**	**Pi**	**mini-SWE**	**DeepSeek Harness**		
						Minimal	**Standard**	**PTC**
DeepSWE v1.1 (Resolved)	69.8	65.6	65.5	66.2	74.2	72.6	70.5	67.6
Terminal-Bench v2.1 (Pass@1)	88.0	84.1	85.0	86.1	90.3	90.6	85.8	85.8

Note. All scaffolds use $N = 8$ samples per task on DeepSWE v1.1 and $N = 3$ on Terminal-Bench v2.1. Runs use Linux containers, temperature 1.0, top-p 0.95, a 1M-token context window, and `max_steps=500` model-generation rounds per agent on both benchmarks. Terminal-Bench v2.1 is evaluated without network access. We evaluate four Claude Code versions, with per-version results and their average reported in Appendix Table 5; this table reports v2.1.251. More scaffold-specific configurations are detailed in Appendix B.1.

变化时，模型性能仍保持稳健，表明其智能体行为并未与单一运行框架设计紧密耦合。这种稳健性与合成训练数据中环境、工具模式和交互格式的多样性一致（第 5 节）；这些数据正是为促进跨智能体运行框架的通用性而设计的。

5.3.5. 多智能体

为了探索复杂任务中的多智能体协作，我们使用 DeepSeek Harness 的 Agent Team 模式，对 DeepSeek-V4.1-Flash 开展初步实验。

多智能体运行框架。我们使用 DeepSeek Harness 的 Agent Team 模式，其中主智能体默认可以通过 spawn_teammate 异步创建具有名称、可持续存在的队友。每个队友接收一个委派任务，并以两种模式之一启动：fresh 模式不包含主智能体历史；fork 模式获得主智能体已完成轮次的一次性快照。所有智能体共享同一份仓库检出目录，彼此的修改立即可见。

智能体通过持久化的同伴邮箱通信。通过 send_message 发送的消息，会在运行中队友的下一个步骤边界送达，为空闲队友启动新一轮，或恢复非活跃队友。在每个队友的任务生命周期中，主智能体使用 list_agents 监控运行状态，并使用 wait_agent 等待状态、邮箱或共享任务的变化。任务归属、依赖关系和建议性的写入范围保存在共享任务板上，通过四个 team_task_* 工具维护，并在更新时检查修订版本。

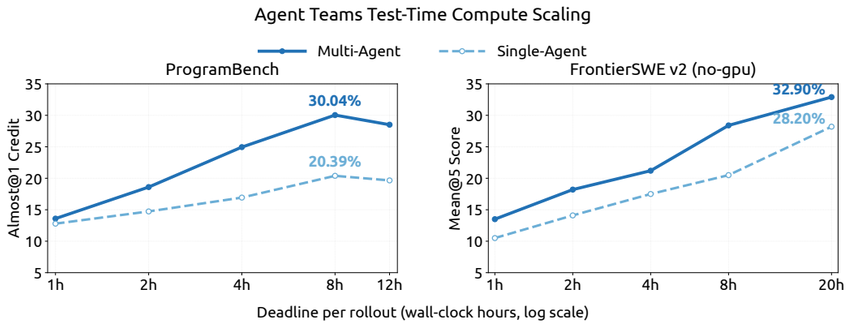


Figure 10 | Test-time compute scaling for single-agent and multi-agent configurations on ProgramBench (Almost@1) and FrontierSWE v2 (Mean@5) as functions of the per-rollout wall-clock deadline.

When intervention is needed, only the lead can interrupt a teammate's current turn through `interrupt_agent`. Once the required work is complete, the lead reviews and tests the combined changes and produces the final response.

Training. We train Agent Team mode with an RL reward combining task performance, a collaboration bonus that encourages delegation and inter-agent communication, and a derived-latency penalty that promotes efficient coordination. Derived latency is computed by representing execution events and their collaboration dependencies as a directed acyclic graph (DAG), assigning costs from token counts at fixed prefill/decode rates plus measured tool-execution time, and taking the length of the critical path. This encourages useful parallelism while penalizing unnecessary sequential work and synchronization, with reduced sensitivity to serving-side batching and queuing delays.

Performance. We construct a high-confidence subset of ProgramBench (Yang et al., 2026) by retaining only tasks for which the reference solution achieves a pass rate of at least 95% on the hidden test suite. This filtering procedure leaves 172 "golden" tasks. We also evaluate on FrontierSWE v2 (Kondra et al., 2026), a larger and more challenging successor to FrontierSWE that uses a substantially improved methodology. We construct a no-GPU subset from the currently public tasks by excluding tasks that require GPU access. We evaluate single-agent and multi-agent configurations on both benchmarks under explicit per-rollout wall-clock deadlines. The results reported here are preliminary: we compare the strongest observed multi-agent configurations with the strongest available single-agent baselines. On ProgramBench, we run up to three rollouts per task, corresponding to 516 planned rollouts for each configuration. We report Almost@1, which measures the fraction of individual rollouts achieving a score of at least 0.95. On FrontierSWE v2, we report Mean@5. ProgramBench deadlines range from 1 to 12 hours, while FrontierSWE v2 is evaluated at deadlines ranging from 1 to 20 hours. At each deadline, metrics are computed from the outputs available when the deadline is reached.

As shown in Figure 10, multi-agent configurations outperform their single-agent counterparts at every deadline on both benchmarks. On ProgramBench, Almost@1 increases from 13.59% at 1 hour to a peak of 30.04% at 8 hours for the multi-agent configuration, compared with 12.79% and 20.39% for the single-agent configuration. On FrontierSWE v2, Mean@5 increases from 13.50% at 1 hour to 32.90% at 20 hours for the multi-agent configuration, compared with an increase from 10.50% to 28.20% for the single-agent configuration.

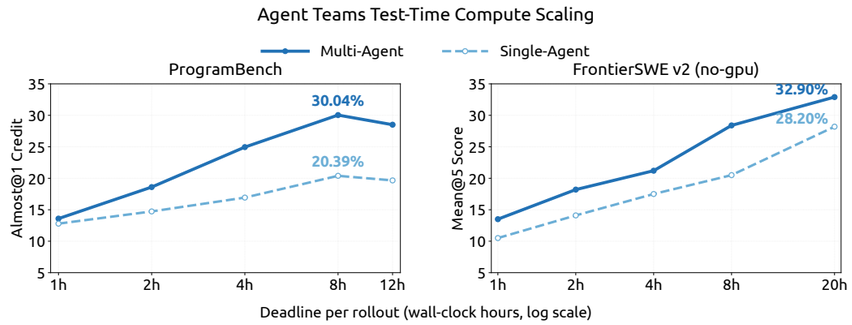


图 10 | 在 ProgramBench（Almost@1）与 FrontierSWE v2（Mean@5）上，单智能体和多智能体配置的测试时计算扩展结果，横轴为每次 rollout 的实际时间截止期限。

当需要干预时，只有主智能体可以通过
interrupt_agent 中断队友的当前轮次。所需工作全部完成后，主智能体审查并测试合并后的修改，然后给出最终响应。

训练。我们使用 RL 奖励训练 Agent Team 模式，奖励结合了任务表现、鼓励委派与智能体间通信的协作奖励，以及促进高效协调的推导延迟惩罚。推导延迟的计算方式是：将执行事件及其协作依赖表示为有向无环图（DAG），根据固定预填充与解码速率下的 token 数，加上实测工具执行时间，为事件赋予成本，再取关键路径长度。这会鼓励有用的并行，同时惩罚不必要的串行工作和同步，并降低对服务端批处理与排队延迟的敏感性。

性能。我们构建了 ProgramBench (Yang et al., 2026) 的高置信度子集，仅保留参考解答在隐藏测试套件上通过率至少为 95% 的任务。筛选后剩余 172 项“黄金”任务。我们还在 FrontierSWE v2 (Kondra et al., 2026) 上评估；它是 FrontierSWE 的后继版本，规模更大、更具挑战性，且方法有了显著改进。我们从当前公开任务中排除需要 GPU 访问的任务，构建无 GPU 子集。我们在两个基准上，使用明确的逐 rollout 实际时间截止期限，评估单智能体和多智能体配置。这里报告的是初步结果：比较观察到的最强多智能体配置与可获得的最强单智能体基线。在 ProgramBench 上，每项任务最多运行三次 rollout，每种配置计划运行 516 次。我们报告 Almost@1，衡量单次 rollout 中得分至少达到 0.95 的比例。在 FrontierSWE v2 上，我们报告 Mean@5。ProgramBench 的截止期限范围为 1 至 12 小时，FrontierSWE v2 为 1 至 20 小时。每个期限对应的指标，都根据到达截止时间时可用的输出计算。

如图 10 所示，在两个基准的所有截止期限下，多智能体配置都优于对应的单智能体配置。在 ProgramBench 上，多智能体配置的 Almost@1 从 1 小时的 13.59% 提升到 8 小时的峰值 30.04%，而单智能体配置对应为 12.79% 和 20.39%。在 FrontierSWE v2 上，多智能体配置的 Mean@5 从 1 小时的 13.50% 提升到 20 小时的 32.90%，单智能体配置则从 10.50% 提升到 28.20%。

6. Conclusion, Limitations, and Future Directions

In this work, we introduce DeepSeek-V4.1-Flash, a multimodal Mixture-of-Experts (MoE) model with support for contexts of up to one million tokens. Through joint optimization of model architecture, cache precision, and deployment strategy, DeepSeek-V4.1-Flash pushes the limits of KV cache compression. Its Causal Encoder-Decoder (CED) architecture enables the model to activate only 8B parameters per token during prefill, compared with 16B during decode, improving cost efficiency for input-heavy agentic workloads. At equal sequence lengths, cross-layer KV cache reuse in Compressed Sparse Attention 2 (CSA2) and FP4 KV caching reduce its global KV cache footprint (always in HBM) to 890 bytes per token, roughly 1/4 of the corresponding footprint of DeepSeek-V4-Flash. SWA Bounded Replay further reduces its persistent KV cache footprint (always on SSD or in host memory) to roughly 1/8 of that of DeepSeek-V4-Flash. These reductions alleviate HBM and SSD capacity pressure while the model delivers substantially better overall performance than DeepSeek-V4-Flash. Despite possessing a significantly smaller parameter footprint than contemporary open-source models such as GLM-5.3 and Kimi-K3, DeepSeek-V4.1 achieves comparable—and in several tasks, superior—performance across key benchmarks.

Although DeepSeek-V4.1-Flash substantially simplifies several architectural components relative to DeepSeek-V4-Flash, the newly introduced architectural changes also create robustness boundaries that have yet to be fully characterized. Our internal evaluations cover a diverse range of test cases and boundary conditions, and we have not observed any systematic degradation in model capabilities in the evaluated settings. Nevertheless, no finite test suite can cover every extreme input and deployment condition. Potential selection errors in CSA2 and approximate state reconstruction in SWA Bounded Replay may still cause capability degradation in untested boundary cases. Going forward, we will continue to expand our stress-testing and evaluation stack, with particular attention to sparse retrieval over long contexts and SWA state reconstruction at cache-resumption boundaries. We will also monitor real-world workloads, systematically characterize potential failure modes and robustness boundaries, and further improve model robustness under extreme conditions.

As AI models achieve remarkable performance capabilities, standard evaluation benchmarks have increasingly reached saturation. While DeepSeek-V4.1-Flash demonstrates performance that closely approaches top-tier models like Fable-5 and GPT-6 Astra—offering a highly comparable user experience in daily applications—a performance gap remains on the most challenging tasks. Although benchmark scores show a narrow margin, this parity does not imply that the model matches the frontier capabilities of leading closed-source systems on complex, high-difficulty reasoning and edge cases.

Consequently, we will continuously update our evaluation protocols to ensure rigorous assessment of state-of-the-art reasoning boundaries. Alongside continued efforts to reduce model costs, we believe that further advances in model intelligence will depend on the coordinated scaling of data, model capacity, and RL. With DeepSeek-V4.1-Flash as a new starting point, we will continue to explore the limits of model capabilities and systematically address key challenges in large-scale data synthesis and RL scaling. We will also actively integrate model–harness co-design, enabling the joint system to evolve and be optimized together. By advancing cost reduction and capability scaling in tandem, we hope to make highly capable agents more accessible and easier to deploy, further lowering the barriers to adopting AI technologies across a broader range of industries and scenarios.

6. 结论、局限与未来方向

在本研究中，我们推出 DeepSeek-V4.1-Flash，一个支持最长一百万 token 上下文的多模态混合专家（MoE）模型。通过联合优化模型架构、缓存精度和部署策略，DeepSeek-V4.1-Flash 探索了 KV 缓存压缩的极限。其因果编码器—解码器（CED）架构使模型在预填充时每个 token 仅激活 8B 参数，而解码时激活 16B 参数，提高了输入密集型智能体工作负载的成本效率。在相同序列长度下，压缩稀疏注意力 2（CSA2）的跨层 KV 缓存复用与 FP4 KV 缓存，将其全局 KV 缓存占用（始终驻留 HBM）降至每个 token 890 字节，约为 DeepSeek-V4-Flash 对应占用的四分之一。SWA 有界重放进一步将持久化 KV 缓存占用（始终存放于 SSD 或主机内存）降至 DeepSeek-V4-Flash 的约八分之一。这些缩减缓解了 HBM 与 SSD 的容量压力，而模型总体性能仍显著优于 DeepSeek-V4-Flash。尽管参数规模远小于 GLM-5.3、Kimi-K3 等同期开源模型，DeepSeek-V4.1 在关键基准上仍取得相当、并在若干任务上更优的性能。

虽然 DeepSeek-V4.1-Flash 相较 DeepSeek-V4-Flash 大幅简化了若干架构组件，新引入的架构改动也带来了尚未完全刻画的稳健性边界。内部评测覆盖了多样的测试案例与边界条件，在已评估的设置中，我们尚未观察到模型能力出现系统性下降。不过，任何有限测试套件都无法覆盖所有极端输入和部署条件。CSA2 中潜在的选择错误，以及 SWA 有界重放中的近似状态重建，仍可能在尚未测试的边界场景下导致能力下降。今后，我们将继续扩展压力测试和评测体系，尤其关注长上下文中的稀疏检索，以及缓存恢复边界上的 SWA 状态重建。我们也将监测真实工作负载，系统刻画潜在失效模式与稳健性边界，并进一步提高模型在极端条件下的稳健性。

随着 AI 模型展现出显著的性能能力，标准评测基准日益趋于饱和。虽然 DeepSeek-V4.1-Flash 的表现已十分接近 Fable-5、GPT-6 Astra 等顶尖模型，在日常应用中提供高度相近的用户体验，但在最具挑战性的任务上仍有性能差距。尽管基准分数差距较小，这种接近并不意味着模型在复杂、高难度推理和边界场景中，已经达到领先闭源系统的前沿能力。

因此，我们将持续更新评测协议，确保严格评估最先进的推理能力边界。在继续降低模型成本的同时，我们认为模型智能的进一步提升，将依赖于数据、模型容量和 RL 的协同扩展。以 DeepSeek-V4.1-Flash 为新的起点，我们将继续探索模型能力的极限，并系统解决大规模数据合成与 RL 扩展中的关键挑战。我们也会积极整合模型与运行框架的协同设计，使联合系统能够共同演进与优化。通过同时推进成本降低和能力扩展，我们希望让高能力智能体更易获得、更易部署，进一步降低各行业和场景采用 AI 技术的门槛。

References

J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. arXiv preprint arXiv:2305.13245, 2023.

E. Alvarez, O. Almog, E. Chung, S. Layton, D. Stosic, R. Krashinsky, and K. Aubrey. Introducing nvfp4 for efficient and accurate low-precision inference, 2025. URL https://developer.nvidia.com/blog/introducing-nvfp4-for-efficient-and-accurate-low-precision-inference/.

Anomaly. Opencode. https://github.com/anomalyco/opencode, 2026.

Anthropic. Claude code. https://code.claude.com/docs/en/overview, 2026.

Y. Bai, S. Tu, J. Zhang, H. Peng, X. Wang, X. Lv, S. Cao, J. Xu, L. Hou, Y. Dong, et al. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 3639–3664, 2025.

Y. Bai, Q. Dong, T. Jiang, X. Lv, Z. Du, A. Zeng, J. Tang, and J. Li. Indexcache: Accelerating sparse attention via cross-layer index reuse. arXiv preprint arXiv:2603.12201, 2026.

W. Brandon, M. Mishra, A. Nrusimha, R. Panda, and J. Ragan-Kelley. Reducing transformer key-value cache size with cross-layer attention. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, Advances in Neural Information Processing Systems, volume 37, pages 86927–86957. Curran Associates, Inc., 2024. doi: 10.52202/079017-2758. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/9e23d020c18e4c40d81c6a0fc7a46f68-Paper-Conference.pdf.

L. Chen, D. Xu, C. An, X. Wang, Y. Zhang, J. Chen, Z. Liang, F. Wei, J. Liang, Y. Xiao, et al. Powerattention: exponentially scaling of receptive fields for effective sparse attention. arXiv preprint arXiv:2503.03588, 2025.

L. Chen, W. Xie, Y. Liang, H. He, H. Zhao, Z. Yang, Z. Huang, H. Wu, H. Lu, Y. Bao, et al. Babyvision: Visual reasoning beyond language. arXiv preprint arXiv:2601.06521, 2026.

M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code. CoRR, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

X. Cheng, X. Yu, C. Shao, J. Li, Y. Xiong, Y. Qian, J. Zhu, S. Ma, X. Zhang, J. Ye, et al. Dspark: Confidence-scheduled speculative decoding with semi-autoregressive generation. arXiv preprint arXiv:2607.05147, 2026a.

X. Cheng, W. Zeng, D. Dai, Q. Chen, B. Wang, Z. Xie, K. Huang, X. Yu, Z. Hao, Y. Li, H. Zhang, H. Zhang, D. Zhao, and W. Liang. Conditional memory via scalable lookup: A new axis of

References

J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. arXiv preprint arXiv:2305.13245, 2023.

E. Alvarez, O. Almog, E. Chung, S. Layton, D. Stosic, R. Krashinsky, and K. Aubrey. Introducing nvfp4 for efficient and accurate low-precision inference, 2025. URL https://developer.nvidia.com/blog/introducing-nvfp4-for-efficient-and-accurate-low-precision-inference/.

Anomaly. Opencode. https://github.com/anomalyco/opencode, 2026.

Anthropic. Claude code. https://code.claude.com/docs/en/overview, 2026.

Y. Bai, S. Tu, J. Zhang, H. Peng, X. Wang, X. Lv, S. Cao, J. Xu, L. Hou, Y. Dong, et al. Longbench v2: Towards deeper understanding and reasoning on realistic long-context multitasks. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 3639–3664, 2025.

Y. Bai, Q. Dong, T. Jiang, X. Lv, Z. Du, A. Zeng, J. Tang, and J. Li. Indexcache: Accelerating sparse attention via cross-layer index reuse. arXiv preprint arXiv:2603.12201, 2026.

W. Brandon, M. Mishra, A. Nrusimha, R. Panda, and J. Ragan-Kelley. Reducing transformer key-value cache size with cross-layer attention. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, Advances in Neural Information Processing Systems, volume 37, pages 86927–86957. Curran Associates, Inc., 2024. doi: 10.52202/079017-2758. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/9e23d020c18e4c40d81c6a0fc7a46f68-Paper-Conference.pdf.

L. Chen, D. Xu, C. An, X. Wang, Y. Zhang, J. Chen, Z. Liang, F. Wei, J. Liang, Y. Xiao, et al. Powerattention: exponentially scaling of receptive fields for effective sparse attention. arXiv preprint arXiv:2503.03588, 2025.

L. Chen, W. Xie, Y. Liang, H. He, H. Zhao, Z. Yang, Z. Huang, H. Wu, H. Lu, Y. Bao, et al. Babyvision: Visual reasoning beyond language. arXiv preprint arXiv:2601.06521, 2026.

M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code. CoRR, abs/2107.03374, 2021. URL https://arxiv.org/abs/2107.03374.

X. Cheng, X. Yu, C. Shao, J. Li, Y. Xiong, Y. Qian, J. Zhu, S. Ma, X. Zhang, J. Ye, et al. Dspark: Confidence-scheduled speculative decoding with semi-autoregressive generation. arXiv preprint arXiv:2607.05147, 2026a.

X. Cheng, W. Zeng, D. Dai, Q. Chen, B. Wang, Z. Xie, K. Huang, X. Yu, Z. Hao, Y. Li, H. Zhang, H. Zhang, D. Zhao, and W. Liang. Conditional memory via scalable lookup: A new axis of

sparsity for large language models. CoRR, abs/2601.07372, 2026b. doi: 10.48550/ARXIV.2601.07372. URL https://doi.org/10.48550/arXiv.2601.07372.

X. Cheng, W. Zeng, D. Dai, Q. Chen, B. Wang, Z. Xie, K. Huang, X. Yu, Z. Hao, H. Zhang, et al. Conditional memory via scalable lookup: A new axis of sparsity for large language models. In Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 4968–4990, 2026c.

K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, et al. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168, 2021.

D. Dai, C. Deng, C. Zhao, R. X. Xu, H. Gao, D. Chen, J. Li, W. Zeng, X. Yu, Y. Wu, Z. Xie, Y. K. Li, P. Huang, F. Luo, C. Ruan, Z. Sui, and W. Liang. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. CoRR, abs/2401.06066, 2024. URL https://doi.org/10.48550/arXiv.2401.06066.

DataCurve. Deepswe v1.1, 2026. URL https://deepswe.datacurve.ai/.

DeepSeek-AI. Deepseek-v3 technical report. CoRR, abs/2412.19437, 2024. URL https://doi.org/10.48550/arXiv.2412.19437.

DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. CoRR, abs/2405.04434, 2024. URL https://doi.org/10.48550/arXiv.2405.04434.

DeepSeek-AI. Deepseek-v3.2: Pushing the frontier of open large language models, 2025. URL https://arxiv.org/abs/2512.02556.

DeepSeek-AI. Deepseek harness: Everything is a plugin. https://github.com/deepseek-ai/deepseek-harness, 2026a.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence. CoRR, abs/2606.19348, 2026b. URL https://doi.org/10.48550/arXiv.2606.19348.

J. Dekoninck, N. Jovanović, I. Petrov, and M. Vechev. Matharena apex: Unconquered final-answer problems, 2025. URL https://matharena.ai/apex/.

S. Deng, Z. Ouyang, T. Pang, Z. Liu, R. Jin, S. Yu, and Y. Yang. Rmnp: Row-momentum normalized preconditioning for scalable matrix-based optimization. arXiv preprint arXiv:2603.20527, 2026.

J. Ding, S. Long, C. Pu, H. Zhou, H. Gao, X. Gao, C. He, Y. Hou, F. Hu, Z. Li, et al. Nl2repo-bench: Towards long-horizon repository generation evaluation of coding agents. arXiv preprint arXiv:2512.12730, 2025.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In International Conference on Learning Representations, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

X. Du, Y. Yao, K. Ma, B. Wang, T. Zheng, K. Zhu, M. Liu, Y. Liang, X. Jin, Z. Wei, et al. Supergpqa: Scaling llm evaluation across 285 graduate disciplines. arXiv preprint arXiv:2502.14739, 2025.

sparsity for large language models. CoRR, abs/2601.07372, 2026b. doi: 10.48550/ARXIV.2601.07372. URL https://doi.org/10.48550/arXiv.2601.07372.

X. Cheng, W. Zeng, D. Dai, Q. Chen, B. Wang, Z. Xie, K. Huang, X. Yu, Z. Hao, H. Zhang, et al. Conditional memory via scalable lookup: A new axis of sparsity for large language models. In Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 4968–4990, 2026c.

K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, et al. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168, 2021.

D. Dai, C. Deng, C. Zhao, R. X. Xu, H. Gao, D. Chen, J. Li, W. Zeng, X. Yu, Y. Wu, Z. Xie, Y. K. Li, P. Huang, F. Luo, C. Ruan, Z. Sui, and W. Liang. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. CoRR, abs/2401.06066, 2024. URL https://doi.org/10.48550/arXiv.2401.06066.

DataCurve. Deepswe v1.1, 2026. URL https://deepswe.datacurve.ai/.

DeepSeek-AI. Deepseek-v3 technical report. CoRR, abs/2412.19437, 2024. URL https://doi.org/10.48550/arXiv.2412.19437.

DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. CoRR, abs/2405.04434, 2024. URL https://doi.org/10.48550/arXiv.2405.04434.

DeepSeek-AI. Deepseek-v3.2: Pushing the frontier of open large language models, 2025. URL https://arxiv.org/abs/2512.02556.

DeepSeek-AI. Deepseek harness: Everything is a plugin. https://github.com/deepseek-ai/deepseek-harness, 2026a.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence. CoRR, abs/2606.19348, 2026b. URL https://doi.org/10.48550/arXiv.2606.19348.

J. Dekoninck, N. Jovanović, I. Petrov, and M. Vechev. Matharena apex: Unconquered final-answer problems, 2025. URL https://matharena.ai/apex/.

S. Deng, Z. Ouyang, T. Pang, Z. Liu, R. Jin, S. Yu, and Y. Yang. Rmnp: Row-momentum normalized preconditioning for scalable matrix-based optimization. arXiv preprint arXiv:2603.20527, 2026.

J. Ding, S. Long, C. Pu, H. Zhou, H. Gao, X. Gao, C. He, Y. Hou, F. Hu, Z. Li, et al. Nl2repo-bench: Towards long-horizon repository generation evaluation of coding agents. arXiv preprint arXiv:2512.12730, 2025.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In International Conference on Learning Representations, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

X. Du, Y. Yao, K. Ma, B. Wang, T. Zheng, K. Zhu, M. Liu, Y. Liang, X. Jin, Z. Wei, et al. Supergpqa: Scaling llm evaluation across 285 graduate disciplines. arXiv preprint arXiv:2502.14739, 2025.

D. Dua, Y. Wang, P. Dasigi, G. Stanovsky, S. Singh, and M. Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In J. Burstein, C. Doran, and T. Solorio, editors, Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), pages 2368–2378. Association for Computational Linguistics, 2019. doi: 10.18653/V1/N19-1246. URL https://doi.org/10.18653/v1/n19-1246.

Y. Gao, J. Wei, Q. Zhang, Y. Cheng, S. Chen, Z. Tang, Z. Jiang, Y. Song, H. Zhang, L. Zhao, B. Yang, G. Wang, S. Cao, and F. Luo. Hysparse: A hybrid sparse attention architecture with oracle token selection and kv cache sharing. arXiv preprint arXiv:2602.03560, 2026.

S. Garre, C. Mutty, S. Mehta, and E. Chen. Chartography: A benchmark for professional chart understanding. arXiv preprint arXiv:2608.10677, 2026.

A. Glentis, J. Li, A. Han, and M. Hong. Memory-efficient llm pretraining via minimalist optimizer design. arXiv preprint arXiv:2506.16659, 2025.

Y. Gu, L. Dong, F. Wei, and M. Huang. Minillm: Knowledge distillation of large language models. In The Twelfth International Conference on Learning Representations, 2024.

L. Haas, G. Yona, G. D'Antonio, S. Goldshtein, and D. Das. Simpleqa verified: A reliable factuality benchmark to measure parametric knowledge. arXiv preprint arXiv:2509.07968, 2025.

D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the math dataset. arXiv preprint arXiv:2103.03874, 2021.

Y. Huang, Y. Bai, Z. Zhu, J. Zhang, J. Zhang, T. Su, J. Liu, C. Lv, Y. Zhang, J. Lei, et al. C-Eval: A multi-level multi-discipline chinese evaluation suite for foundation models. arXiv preprint arXiv:2305.08322, 2023.

D. Hupkes and N. Bogoychev. Multiloko: a multilingual local knowledge benchmark for llms spanning 31 languages. CoRR, abs/2504.10356, 2025. doi: 10.48550/ARXIV.2504.10356. URL https://doi.org/10.48550/arXiv.2504.10356.

B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2018.

K. Jordan, Y. Jin, V. Boza, J. You, F. Cesista, L. Newhouse, and J. Bernstein. Muon: An optimizer for hidden layers in neural networks. Cited on, page 10, 2024.

M. Kazemi, B. Fatemi, H. Bansal, J. Palowitch, C. Anastasiou, S. V. Mehta, L. K. Jain, V. Aglietti, D. Jindal, Y. P. Chen, et al. Big-bench extra hard. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 26473–26501, 2025.

S. Kazemzadeh, V. Ordonez, M. Matten, and T. Berg. Referitgame: Referring to objects in photographs of natural scenes. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 787–798, 2014.

D. Dua, Y. Wang, P. Dasigi, G. Stanovsky, S. Singh, and M. Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In J. Burstein, C. Doran, and T. Solorio, editors, Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers), pages 2368–2378. Association for Computational Linguistics, 2019. doi: 10.18653/V1/N19-1246. URL https://doi.org/10.18653/v1/n19-1246.

Y. Gao, J. Wei, Q. Zhang, Y. Cheng, S. Chen, Z. Tang, Z. Jiang, Y. Song, H. Zhang, L. Zhao, B. Yang, G. Wang, S. Cao, and F. Luo. Hysparse: A hybrid sparse attention architecture with oracle token selection and kv cache sharing. arXiv preprint arXiv:2602.03560, 2026.

S. Garre, C. Mutty, S. Mehta, and E. Chen. Chartography: A benchmark for professional chart understanding. arXiv preprint arXiv:2608.10677, 2026.

A. Glentis, J. Li, A. Han, and M. Hong. Memory-efficient llm pretraining via minimalist optimizer design. arXiv preprint arXiv:2506.16659, 2025.

Y. Gu, L. Dong, F. Wei, and M. Huang. Minillm: Knowledge distillation of large language models. In The Twelfth International Conference on Learning Representations, 2024.

L. Haas, G. Yona, G. D'Antonio, S. Goldshtein, and D. Das. Simpleqa verified: A reliable factuality benchmark to measure parametric knowledge. arXiv preprint arXiv:2509.07968, 2025.

D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the math dataset. arXiv preprint arXiv:2103.03874, 2021.

Y. Huang, Y. Bai, Z. Zhu, J. Zhang, J. Zhang, T. Su, J. Liu, C. Lv, Y. Zhang, J. Lei, et al. C-Eval: A multi-level multi-discipline chinese evaluation suite for foundation models. arXiv preprint arXiv:2305.08322, 2023.

D. Hupkes and N. Bogoychev. Multiloko: a multilingual local knowledge benchmark for llms spanning 31 languages. CoRR, abs/2504.10356, 2025. doi: 10.48550/ARXIV.2504.10356. URL https://doi.org/10.48550/arXiv.2504.10356.

B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2018.

K. Jordan, Y. Jin, V. Boza, J. You, F. Cesista, L. Newhouse, and J. Bernstein. Muon: An optimizer for hidden layers in neural networks. Cited on, page 10, 2024.

M. Kazemi, B. Fatemi, H. Bansal, J. Palowitch, C. Anastasiou, S. V. Mehta, L. K. Jain, V. Aglietti, D. Jindal, Y. P. Chen, et al. Big-bench extra hard. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 26473–26501, 2025.

S. Kazemzadeh, V. Ordonez, M. Matten, and T. Berg. Referitgame: Referring to objects in photographs of natural scenes. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), pages 787–798, 2014.

R. Kondra, S. Mhatre, A. Kumar, E. Chu, B. B. Ahmad, A. Nangia, R. Agarwal, A. Dasgupta, A. Sinha, B. Sridharan, K. Dave, B. Graham, G. Song, A. Rahul, W. H. Lim, A. Thangamuthu, R. Singh, D. Liu, N. Pour, C. Chen, and J. Mattern. Frontierswe v2. Proximal Blog, 2026. https://frontierswe.com/blog/v2.

H. Lee, J. Liu, D. Kim, W. Xia, Z. Zhang, C. S. Xia, and L. Zhang. Sec-bench pro: Can language models solve long-horizon software security tasks? arXiv preprint arXiv:2605.26548, 2026.

J. Li and S. Liu. Flashmla: Efficient multi-head latent attention kernels. `https://github.com/deepseek-ai/FlashMLA`, 2025.

J. Liu, J. Su, X. Yao, Z. Jiang, G. Lai, Y. Du, Y. Qin, W. Xu, E. Lu, J. Yan, Y. Chen, H. Zheng, Y. Liu, S. Liu, B. Yin, W. He, H. Zhu, Y. Wang, J. Wang, M. Dong, Z. Zhang, Y. Kang, H. Zhang, X. Xu, Y. Zhang, Y. Wu, X. Zhou, and Z. Yang. Muon is scalable for LLM training. CoRR, abs/2502.16982, 2025. URL `https://doi.org/10.48550/arXiv.2502.16982`.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In International Conference on Learning Representations, 2019. URL `https://openreview.net/forum?id=Bkg6RiCqY7`.

K. Lu and T. M. Lab. On-policy distillation. Thinking Machines Lab: Connectionism, 2025. doi: 10.64434/tml.20251026. https://thinkingmachines.ai/blog/on-policy-distillation.

J. Mao, J. Huang, A. Toshev, O. Camburu, A. L. Yuille, and K. Murphy. Generation and comprehension of unambiguous object descriptions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 11–20, 2016.

A. Marafioti, O. Zohar, M. Farré, M. Noyan, E. Bakouch, P. Cuenca, C. Zakka, L. B. Allal, A. Lozhkov, N. Tazi, V. Srivastav, J. Lochner, H. Larcher, M. Morlon, L. Tunstall, L. von Werra, and T. Wolf. Smolvlm: Redefining small and efficient multimodal models. arXiv preprint arXiv:2504.05299, 2025.

R. Marten, A. Shaw, I. Bercovich, B. Droste, T. Cerruti, S. Dillmann, R. Wang, D. Wahdany, A. Hart, K. Krauth, ScaleAI, Snorkel AI, Turing, gNucleus AI, Boolean AI, N. Carlini, S. Lyu, A. Wei, A. Khatua, B. Plüster, C. Dwivedi, C. Sutcliffe, Yuming, D. Tivris, D. Wang, H. W. Goh, H. Xing, H. Lin, I. Salia, J. Seol, J. Bao, J. Ouyang, J. Park, L. Walsh, L. Kong, M. Ivanov, M. Ubl, M. Liamets, O. Menis, P. Migdal, Q. Bao, R. Movva, R. Ben Chaim, N. Srinath, S. Bogdanik, S. Yadav, S. Benjamin, T. Kung, W. Hughes, X. Lan, H. Gupta, S. Mishra, C. Wang, H. He, J. Tu, K. Montgomery, Z. Tu, A. Naik, D. Mortensen, I. Zhang, Y. Mathur, E. Liu, K. Singh, M. Yu, S. Feng, V. Gangal, Z. Tao, S. Ruan, J. Mueller, J. Cabezas, J. Bauer, K. X. Li, R. Zhang, A. Feller, A. Madayan, L. Chen, B. Feuer, X. Li, B. Li, H. Raj, S. Galler, L. Shi, I. Segal, K. Buchanan, S. P., R. Desai, A. Schneider, C. Settles, X. Lin, M. Nezhurina, A. Wang, M. Kowalczyk, J.-X. Zhao, S. Satia, J. Hu, S. Atef, K. Chen, S. Vance, G. Segato, J. Jitsev, A. Dimakis, M. Merrill, A. Konwinski, and L. Schmidt. Terminal-Bench, Aug. 2026a. URL `https://github.com/harbor-framework/terminal-bench`.

R. Marten, A. Shaw, A. Konwinski, and Terminal-Bench Contributors. Terminal-bench 3.0: Harder tasks for better agents. `https://www.tbench.ai/news/terminal-bench-3-0`, 2026b. An Open Benchmark for Agent Work in Terminal Environments.

M. Mathew, D. Karatzas, and C. Jawahar. Docvqa: A dataset for vqa on document images. In 2021 IEEE Winter Conference on Applications of Computer Vision (WACV), pages 2199–2208. IEEE, 2021.

R. Kondra, S. Mhatre, A. Kumar, E. Chu, B. B. Ahmad, A. Nangia, R. Agarwal, A. Dasgupta, A. Sinha, B. Sridharan, K. Dave, B. Graham, G. Song, A. Rahul, W. H. Lim, A. Thangamuthu, R. Singh, D. Liu, N. Pour, C. Chen, and J. Mattern. Frontierswe v2. Proximal Blog, 2026. https://frontierswe.com/blog/v2.

H. Lee, J. Liu, D. Kim, W. Xia, Z. Zhang, C. S. Xia, and L. Zhang. Sec-bench pro: Can language models solve long-horizon software security tasks? arXiv preprint arXiv:2605.26548, 2026.

J. Li and S. Liu. Flashmla: Efficient multi-head latent attention kernels. `https://github.com/deepseek-ai/FlashMLA`, 2025.

J. Liu, J. Su, X. Yao, Z. Jiang, G. Lai, Y. Du, Y. Qin, W. Xu, E. Lu, J. Yan, Y. Chen, H. Zheng, Y. Liu, S. Liu, B. Yin, W. He, H. Zhu, Y. Wang, J. Wang, M. Dong, Z. Zhang, Y. Kang, H. Zhang, X. Xu, Y. Zhang, Y. Wu, X. Zhou, and Z. Yang. Muon is scalable for LLM training. CoRR, abs/2502.16982, 2025. URL `https://doi.org/10.48550/arXiv.2502.16982`.

I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In International Conference on Learning Representations, 2019. URL `https://openreview.net/forum?id=Bkg6RiCqY7`.

K. Lu and T. M. Lab. On-policy distillation. Thinking Machines Lab: Connectionism, 2025. doi: 10.64434/tml.20251026. https://thinkingmachines.ai/blog/on-policy-distillation.

J. Mao, J. Huang, A. Toshev, O. Camburu, A. L. Yuille, and K. Murphy. Generation and comprehension of unambiguous object descriptions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 11–20, 2016.

A. Marafioti, O. Zohar, M. Farré, M. Noyan, E. Bakouch, P. Cuenca, C. Zakka, L. B. Allal, A. Lozhkov, N. Tazi, V. Srivastav, J. Lochner, H. Larcher, M. Morlon, L. Tunstall, L. von Werra, and T. Wolf. Smolvlm: Redefining small and efficient multimodal models. arXiv preprint arXiv:2504.05299, 2025.

R. Marten, A. Shaw, I. Bercovich, B. Droste, T. Cerruti, S. Dillmann, R. Wang, D. Wahdany, A. Hart, K. Krauth, ScaleAI, Snorkel AI, Turing, gNucleus AI, Boolean AI, N. Carlini, S. Lyu, A. Wei, A. Khatua, B. Plüster, C. Dwivedi, C. Sutcliffe, Yuming, D. Tivris, D. Wang, H. W. Goh, H. Xing, H. Lin, I. Salia, J. Seol, J. Bao, J. Ouyang, J. Park, L. Walsh, L. Kong, M. Ivanov, M. Ubl, M. Liamets, O. Menis, P. Migdal, Q. Bao, R. Movva, R. Ben Chaim, N. Srinath, S. Bogdanik, S. Yadav, S. Benjamin, T. Kung, W. Hughes, X. Lan, H. Gupta, S. Mishra, C. Wang, H. He, J. Tu, K. Montgomery, Z. Tu, A. Naik, D. Mortensen, I. Zhang, Y. Mathur, E. Liu, K. Singh, M. Yu, S. Feng, V. Gangal, Z. Tao, S. Ruan, J. Mueller, J. Cabezas, J. Bauer, K. X. Li, R. Zhang, A. Feller, A. Madayan, L. Chen, B. Feuer, X. Li, B. Li, H. Raj, S. Galler, L. Shi, I. Segal, K. Buchanan, S. P., R. Desai, A. Schneider, C. Settles, X. Lin, M. Nezhurina, A. Wang, M. Kowalczyk, J.-X. Zhao, S. Satia, J. Hu, S. Atef, K. Chen, S. Vance, G. Segato, J. Jitsev, A. Dimakis, M. Merrill, A. Konwinski, and L. Schmidt. Terminal-Bench, Aug. 2026a. URL `https://github.com/harbor-framework/terminal-bench`.

R. Marten, A. Shaw, A. Konwinski, and Terminal-Bench Contributors. Terminal-bench 3.0: Harder tasks for better agents. `https://www.tbench.ai/news/terminal-bench-3-0`, 2026b. An Open Benchmark for Agent Work in Terminal Environments.

M. Mathew, D. Karatzas, and C. Jawahar. Docvqa: A dataset for vqa on document images. In 2021 IEEE Winter Conference on Applications of Computer Vision (WACV), pages 2199–2208. IEEE, 2021.

M. A. Merrill, A. G. Shaw, N. Carlini, B. Li, H. Raj, I. Bercovich, L. Shi, J. Y. Shin, T. Walshe, E. K. Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. arXiv preprint arXiv:2601.11868, 2026.

V. K. Nagaraja, V. I. Morariu, and L. S. Davis. Modeling context between objects for referring expression understanding. In European conference on computer vision, pages 792–807. Springer, 2016.

Y. Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. Soviet Mathematics Doklady, 27:372–376, 1983.

OpenAI. gpt-oss-120b & gpt-oss-20b model card. CoRR, abs/2508.10925, 2025. doi: 10.48550/ARXIV.2508.10925. URL https://doi.org/10.48550/arXiv.2508.10925.

OpenAI. Codex. https://github.com/openai/codex, 2026.

L. Phan, A. Gatti, Z. Han, N. Li, J. Hu, H. Zhang, C. B. C. Zhang, M. Shaaban, J. Ling, S. Shi, et al. Humanity's last exam. arXiv preprint arXiv:2501.14249, 2025.

Y. Qian, S. Liu, and Y. Li. Deepselect: High-performance topk kernels for deepseek sparse attention and sampling. https://github.com/deepseek-ai/DeepSelect, 2026.

D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. arXiv preprint arXiv:2311.12022, 2023.

J. Roberts, M. R. Taesiri, A. Sharma, A. Gupta, S. Roberts, I. Croitoru, S.-V. Bogolin, J. Tang, F. Langer, V. Raina, et al. Zerobench: An impossible visual benchmark for contemporary large multimodal models. arXiv preprint arXiv:2502.09696, 2025.

B. D. Rouhani, R. Zhao, A. More, M. Hall, A. Khodamoradi, S. Deng, D. Choudhary, M. Cornea, E. Dellinger, K. Denolf, S. Dusan, V. Elango, M. Golub, A. Heinecke, P. James-Roxby, D. Jani, G. Kolhe, M. Langhammer, A. Li, L. Melnick, M. Mesmakhosroshahi, A. Rodriguez, M. Schulte, R. Shafipour, L. Shao, M. Siu, P. Dubey, P. Micikevicius, M. Naumov, C. Verrilli, R. Wittig, D. Burger, and E. Chung. Microscaling data formats for deep learning, 2023.

M. Scetbon, C. Ma, W. Gong, and E. Meeds. Gradient multi-normalization for efficient LLM training. In The Thirty-ninth Annual Conference on Neural Information Processing Systems, 2025. URL https://openreview.net/forum?id=oanhUGY6un.

N. Shazeer. Glu variants improve transformer. arXiv preprint arXiv:2002.05202, 2020.

N. Shazeer and M. Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In International conference on machine learning, pages 4596–4604. PMLR, 2018.

D. Shepard and R. Salimans. Automationbench. arXiv preprint arXiv:2604.18934, 2026.

F. Shi, M. Suzgun, M. Freitag, X. Wang, S. Srivats, S. Vosoughi, H. W. Chung, Y. Tay, S. Ruder, D. Zhou, D. Das, and J. Wei. Language models are multilingual chain-of-thought reasoners. In The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net, 2023. URL https://openreview.net/forum?id=fR3wGCk-IXp.

Y. Sun, L. Dong, Y. Zhu, S. Huang, W. Wang, S. Ma, Q. Zhang, J. Wang, and F. Wei. You only cache once: Decoder-decoder architectures for language models. Advances in Neural Information Processing Systems, 37:7339–7361, 2024.

M. A. Merrill, A. G. Shaw, N. Carlini, B. Li, H. Raj, I. Bercovich, L. Shi, J. Y. Shin, T. Walshe, E. K. Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. arXiv preprint arXiv:2601.11868, 2026.

V. K. Nagaraja, V. I. Morariu, and L. S. Davis. Modeling context between objects for referring expression understanding. In European conference on computer vision, pages 792–807. Springer, 2016.

Y. Nesterov. A method of solving a convex programming problem with convergence rate $O(1/k^2)$. Soviet Mathematics Doklady, 27:372–376, 1983.

OpenAI. gpt-oss-120b & gpt-oss-20b model card. CoRR, abs/2508.10925, 2025. doi: 10.48550/ARXIV.2508.10925. URL https://doi.org/10.48550/arXiv.2508.10925.

OpenAI. Codex. https://github.com/openai/codex, 2026.

L. Phan, A. Gatti, Z. Han, N. Li, J. Hu, H. Zhang, C. B. C. Zhang, M. Shaaban, J. Ling, S. Shi, et al. Humanity's last exam. arXiv preprint arXiv:2501.14249, 2025.

Y. Qian, S. Liu, and Y. Li. Deepselect: High-performance topk kernels for deepseek sparse attention and sampling. https://github.com/deepseek-ai/DeepSelect, 2026.

D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman. GPQA: A graduate-level google-proof q&a benchmark. arXiv preprint arXiv:2311.12022, 2023.

J. Roberts, M. R. Taesiri, A. Sharma, A. Gupta, S. Roberts, I. Croitoru, S.-V. Bogolin, J. Tang, F. Langer, V. Raina, et al. Zerobench: An impossible visual benchmark for contemporary large multimodal models. arXiv preprint arXiv:2502.09696, 2025.

B. D. Rouhani, R. Zhao, A. More, M. Hall, A. Khodamoradi, S. Deng, D. Choudhary, M. Cornea, E. Dellinger, K. Denolf, S. Dusan, V. Elango, M. Golub, A. Heinecke, P. James-Roxby, D. Jani, G. Kolhe, M. Langhammer, A. Li, L. Melnick, M. Mesmakhosroshahi, A. Rodriguez, M. Schulte, R. Shafipour, L. Shao, M. Siu, P. Dubey, P. Micikevicius, M. Naumov, C. Verrilli, R. Wittig, D. Burger, and E. Chung. Microscaling data formats for deep learning, 2023.

M. Scetbon, C. Ma, W. Gong, and E. Meeds. Gradient multi-normalization for efficient LLM training. In The Thirty-ninth Annual Conference on Neural Information Processing Systems, 2025. URL https://openreview.net/forum?id=oanhUGY6un.

N. Shazeer. Glu variants improve transformer. arXiv preprint arXiv:2002.05202, 2020.

N. Shazeer and M. Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In International conference on machine learning, pages 4596–4604. PMLR, 2018.

D. Shepard and R. Salimans. Automationbench. arXiv preprint arXiv:2604.18934, 2026.

F. Shi, M. Suzgun, M. Freitag, X. Wang, S. Srivats, S. Vosoughi, H. W. Chung, Y. Tay, S. Ruder, D. Zhou, D. Das, and J. Wei. Language models are multilingual chain-of-thought reasoners. In The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net, 2023. URL https://openreview.net/forum?id=fR3wGCk-IXp.

Y. Sun, L. Dong, Y. Zhu, S. Huang, W. Wang, S. Ma, Q. Zhang, J. Wang, and F. Wei. You only cache once: Decoder-decoder architectures for language models. Advances in Neural Information Processing Systems, 37:7339–7361, 2024.

Y. Sun, X. Han, W. Zhang, Y. Pang, T. Wang, Y. Cao, Y. Huang, C. Duroiu, H. Zhang, J. Lin, et al. Agents' last exam. arXiv preprint arXiv:2606.05405, 2026a.

Y. Sun, Y. Zhang, L. Dong, J. Wang, and F. Wei. You only index once: Cross-layer sparse attention with shared routing. arXiv preprint arXiv:2606.06467, 2026b.

M. Suzgun, N. Scales, N. Schärli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. V. Le, E. H. Chi, D. Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. arXiv preprint arXiv:2210.09261, 2022.

K. Team, T. Bai, Y. Bai, Y. Bao, J. Cai, X. Cai, P. Cao, Y. Cao, Z. Chai, Y. Charles, et al. Kimi k3: Open frontier intelligence. arXiv preprint arXiv:2607.24653, 2026a.

K. Team, T. Bai, Y. Bai, Y. Bao, S. Cai, Y. Cao, Z. Chai, Y. Charles, H. Che, C. Chen, et al. Kimi k2.5: Visual agentic intelligence. arXiv preprint arXiv:2602.02276, 2026b.

M. L. Team, B. Wang, B. Xiao, B. Zhang, B. Rong, B. Chen, C. Wan, C. Zhang, C. Huang, C. Chen, et al. Longcat-flash-omni technical report. arXiv preprint arXiv:2511.00279, 2025.

S. Tong, E. Brown, P. Wu, S. Woo, M. Middepogu, S. C. Akula, J. Yang, S. Yang, A. Iyer, X. Pan, A. Wang, R. Fergus, Y. LeCun, and S. Xie. Cambrian-1: A fully open, vision-centric exploration of multimodal llms, 2024.

L. Wang, H. Gao, C. Zhao, X. Sun, and D. Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts. CoRR, abs/2408.15664, 2024a. URL https://doi.org/10.48550/arXiv.2408.15664.

X. Wang, C. Xu, H. Cao, R. Tian, W. Zhao, K. Yu, and C. Zhao. Tilekernels. https://github.com/deepseek-ai/TileKernels, 2026a.

Y. Wang, X. Ma, G. Zhang, Y. Ni, A. Chandra, S. Guo, W. Ren, A. Arulraj, X. He, Z. Jiang, T. Li, M. Ku, K. Wang, A. Zhuang, R. Fan, X. Yue, and W. Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. CoRR, abs/2406.01574, 2024b. URL https://doi.org/10.48550/arXiv.2406.01574.

Z. Wang, N. Schiller, H. Li, S. S. Narayana, M. Nasr, N. Carlini, X. Qi, E. Wallace, E. Bursztein, L. Invernizzi, et al. Exploitgym: Can ai agents turn security vulnerabilities into real attacks? arXiv preprint arXiv:2605.11086, 2026b.

Z. Wang, T. Shi, J. He, M. Cai, J. Zhang, and D. Song. Cybergym: Evaluating ai agents' real-world cybersecurity capabilities at scale. In International Conference on Learning Representations, volume 2026, pages 123341–123386, 2026c.

Z. Wen, Y. Shi, J. Wang, P. Luo, L. Qiao, D. Li, and T. Sun. Sron: State-free llm training via row-wise gradient normalization. 2025.

Z. Xie, Y. Wei, H. Cao, C. Zhao, C. Deng, J. Li, D. Dai, H. Gao, J. Chang, K. Yu, L. Zhao, S. Zhou, Z. Xu, Z. Zhang, W. Zeng, S. Hu, Y. Wang, J. Yuan, L. Wang, and W. Liang. mhc: Manifold-constrained hyper-connections, 2026. URL https://arxiv.org/abs/2512.24880.

R. Xu, J. Li, and Y. Lu. On the width scaling of neural optimizers under matrix operator norms i: Row/column normalization and hyperparameter transfer. arXiv preprint arXiv:2603.09952, 2026a.

Y. Sun, X. Han, W. Zhang, Y. Pang, T. Wang, Y. Cao, Y. Huang, C. Duroiu, H. Zhang, J. Lin, et al. Agents' last exam. arXiv preprint arXiv:2606.05405, 2026a.

Y. Sun, Y. Zhang, L. Dong, J. Wang, and F. Wei. You only index once: Cross-layer sparse attention with shared routing. arXiv preprint arXiv:2606.06467, 2026b.

M. Suzgun, N. Scales, N. Schärli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. V. Le, E. H. Chi, D. Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. arXiv preprint arXiv:2210.09261, 2022.

K. Team, T. Bai, Y. Bai, Y. Bao, J. Cai, X. Cai, P. Cao, Y. Cao, Z. Chai, Y. Charles, et al. Kimi k3: Open frontier intelligence. arXiv preprint arXiv:2607.24653, 2026a.

K. Team, T. Bai, Y. Bai, Y. Bao, S. Cai, Y. Cao, Z. Chai, Y. Charles, H. Che, C. Chen, et al. Kimi k2.5: Visual agentic intelligence. arXiv preprint arXiv:2602.02276, 2026b.

M. L. Team, B. Wang, B. Xiao, B. Zhang, B. Rong, B. Chen, C. Wan, C. Zhang, C. Huang, C. Chen, et al. Longcat-flash-omni technical report. arXiv preprint arXiv:2511.00279, 2025.

S. Tong, E. Brown, P. Wu, S. Woo, M. Middepogu, S. C. Akula, J. Yang, S. Yang, A. Iyer, X. Pan, A. Wang, R. Fergus, Y. LeCun, and S. Xie. Cambrian-1: A fully open, vision-centric exploration of multimodal llms, 2024.

L. Wang, H. Gao, C. Zhao, X. Sun, and D. Dai. Auxiliary-loss-free load balancing strategy for mixture-of-experts. CoRR, abs/2408.15664, 2024a. URL https://doi.org/10.48550/arXiv.2408.15664.

X. Wang, C. Xu, H. Cao, R. Tian, W. Zhao, K. Yu, and C. Zhao. Tilekernels. https://github.com/deepseek-ai/TileKernels, 2026a.

Y. Wang, X. Ma, G. Zhang, Y. Ni, A. Chandra, S. Guo, W. Ren, A. Arulraj, X. He, Z. Jiang, T. Li, M. Ku, K. Wang, A. Zhuang, R. Fan, X. Yue, and W. Chen. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. CoRR, abs/2406.01574, 2024b. URL https://doi.org/10.48550/arXiv.2406.01574.

Z. Wang, N. Schiller, H. Li, S. S. Narayana, M. Nasr, N. Carlini, X. Qi, E. Wallace, E. Bursztein, L. Invernizzi, et al. Exploitgym: Can ai agents turn security vulnerabilities into real attacks? arXiv preprint arXiv:2605.11086, 2026b.

Z. Wang, T. Shi, J. He, M. Cai, J. Zhang, and D. Song. Cybergym: Evaluating ai agents' real-world cybersecurity capabilities at scale. In International Conference on Learning Representations, volume 2026, pages 123341–123386, 2026c.

Z. Wen, Y. Shi, J. Wang, P. Luo, L. Qiao, D. Li, and T. Sun. Sron: State-free llm training via row-wise gradient normalization. 2025.

Z. Xie, Y. Wei, H. Cao, C. Zhao, C. Deng, J. Li, D. Dai, H. Gao, J. Chang, K. Yu, L. Zhao, S. Zhou, Z. Xu, Z. Zhang, W. Zeng, S. Hu, Y. Wang, J. Yuan, L. Wang, and W. Liang. mhc: Manifold-constrained hyper-connections, 2026. URL https://arxiv.org/abs/2512.24880.

R. Xu, J. Li, and Y. Lu. On the width scaling of neural optimizers under matrix operator norms i: Row/column normalization and hyperparameter transfer. arXiv preprint arXiv:2603.09952, 2026a.

Y. Xu, F. Meng, F. Jiang, Y. Wang, R. Zhou, Z. Wang, J. Wu, Z. Pan, X. Tang, W. Pei, et al. Hisa: Efficient hierarchical indexing for fine-grained sparse attention. arXiv preprint arXiv:2603.28458, 2026b.

J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. R. Narasimhan, and O. Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In The Thirty-eighth Annual Conference on Neural Information Processing Systems, 2024. URL https://arxiv.org/abs/2405.15793.

J. Yang, K. Lieret, J. Ma, P. Thakkar, D. Pedchenko, S. Sootla, E. McMilin, P. Yin, R. Hou, G. Synnaeve, D. Yang, and O. Press. Programbench: Can language models rebuild programs from scratch?, 2026. URL https://arxiv.org/abs/2605.03546.

L. Yu, P. Poirson, S. Yang, A. C. Berg, and T. L. Berg. Modeling context in referring expressions. In European conference on computer vision, pages 69–85. Springer, 2016.

J. Yuan, J. Zou, S. Wang, Y. Liu, and F. Nie. Nora: Normalized orthogonal row alignment for scalable matrix optimizer. arXiv preprint arXiv:2605.03769, 2026.

X. Yue, T. Zheng, Y. Ni, Y. Wang, K. Zhang, S. Tong, Y. Sun, B. Yu, G. Zhang, H. Sun, et al. Mmmu-pro: A more robust multi-discipline multimodal understanding benchmark. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 15134–15186, 2025.

M. Zechner. Pi: The coding-agent harness you can make your own. https://github.com/earendil-works/pi, 2026.

R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. HellaSwag: Can a machine really finish your sentence? In A. Korhonen, D. R. Traum, and L. Màrquez, editors, Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers, pages 4791–4800. Association for Computational Linguistics, 2019. doi: 10.18653/v1/p19-1472. URL https://doi.org/10.18653/v1/p19-1472.

A. Zeng, X. Lv, Z. Hou, Z. Du, Q. Zheng, B. Chen, D. Yin, C. Ge, C. Huang, C. Xie, et al. Glm-5: from vibe coding to agentic engineering. arXiv preprint arXiv:2602.15763, 2026.

X. Zhai, B. Mustafa, A. Kolesnikov, and L. Beyer. Sigmoid loss for language image pre-training. In 2023 IEEE/CVF International Conference on Computer Vision (ICCV), pages 11941–11952. IEEE, 2023.

B. Zhang and R. Sennrich. Root mean square layer normalization. Advances in neural information processing systems, 32, 2019.

Y. Zhang. On the principles behind neural network optimizers. arXiv preprint arXiv:2608.16760, 2026.

Y. Zhang, C. Chen, T. Ding, Z. Li, R. Sun, and Z.-Q. Luo. Why transformers need adam: A hessian perspective. Advances in neural information processing systems, 37:131786–131823, 2024.

Y. Zhang, C. Chen, Z. Li, T. Ding, C. Wu, D. D. Kingma, Y. Ye, Z.-Q. Luo, and R. Sun. Adam-mini: Use fewer learning rates to gain more. In International Conference on Learning Representations, volume 2025, pages 28033–28063, 2025a.

Y. Xu, F. Meng, F. Jiang, Y. Wang, R. Zhou, Z. Wang, J. Wu, Z. Pan, X. Tang, W. Pei, et al. Hisa: Efficient hierarchical indexing for fine-grained sparse attention. arXiv preprint arXiv:2603.28458, 2026b.

J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. R. Narasimhan, and O. Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In The Thirty-eighth Annual Conference on Neural Information Processing Systems, 2024. URL https://arxiv.org/abs/2405.15793.

J. Yang, K. Lieret, J. Ma, P. Thakkar, D. Pedchenko, S. Sootla, E. McMilin, P. Yin, R. Hou, G. Synnaeve, D. Yang, and O. Press. Programbench: Can language models rebuild programs from scratch?, 2026. URL https://arxiv.org/abs/2605.03546.

L. Yu, P. Poirson, S. Yang, A. C. Berg, and T. L. Berg. Modeling context in referring expressions. In European conference on computer vision, pages 69–85. Springer, 2016.

J. Yuan, J. Zou, S. Wang, Y. Liu, and F. Nie. Nora: Normalized orthogonal row alignment for scalable matrix optimizer. arXiv preprint arXiv:2605.03769, 2026.

X. Yue, T. Zheng, Y. Ni, Y. Wang, K. Zhang, S. Tong, Y. Sun, B. Yu, G. Zhang, H. Sun, et al. Mmmu-pro: A more robust multi-discipline multimodal understanding benchmark. In Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 15134–15186, 2025.

M. Zechner. Pi: The coding-agent harness you can make your own. https://github.com/earendil-works/pi, 2026.

R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. HellaSwag: Can a machine really finish your sentence? In A. Korhonen, D. R. Traum, and L. Màrquez, editors, Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers, pages 4791–4800. Association for Computational Linguistics, 2019. doi: 10.18653/v1/p19-1472. URL https://doi.org/10.18653/v1/p19-1472.

A. Zeng, X. Lv, Z. Hou, Z. Du, Q. Zheng, B. Chen, D. Yin, C. Ge, C. Huang, C. Xie, et al. Glm-5: from vibe coding to agentic engineering. arXiv preprint arXiv:2602.15763, 2026.

X. Zhai, B. Mustafa, A. Kolesnikov, and L. Beyer. Sigmoid loss for language image pre-training. In 2023 IEEE/CVF International Conference on Computer Vision (ICCV), pages 11941–11952. IEEE, 2023.

B. Zhang and R. Sennrich. Root mean square layer normalization. Advances in neural information processing systems, 32, 2019.

Y. Zhang. On the principles behind neural network optimizers. arXiv preprint arXiv:2608.16760, 2026.

Y. Zhang, C. Chen, T. Ding, Z. Li, R. Sun, and Z.-Q. Luo. Why transformers need adam: A hessian perspective. Advances in neural information processing systems, 37:131786–131823, 2024.

Y. Zhang, C. Chen, Z. Li, T. Ding, C. Wu, D. D. Kingma, Y. Ye, Z.-Q. Luo, and R. Sun. Adam-mini: Use fewer learning rates to gain more. In International Conference on Learning Representations, volume 2025, pages 28033–28063, 2025a.

44

Z. Zhang, Y. Zhong, Y. Jiang, H. Hu, J. Sun, Z. Ge, Y. Zhu, D. Jiang, and X. Jin. Disttrain: Addressing model and data heterogeneity with disaggregated training for multimodal large language models. In Proceedings of the ACM SIGCOMM 2025 Conference, pages 24–38, 2025b.

C. Zhao, L. Zhao, J. Li, Z. Xu, and C. Xu. Deepgemm: clean and efficient fp8 gemm kernels with fine-grained scaling. https://github.com/deepseek-ai/DeepGEMM, 2025.

W. Zhong, R. Cui, Y. Guo, Y. Liang, S. Lu, Y. Wang, A. Saied, W. Chen, and N. Duan. AGIEval: A human-centric benchmark for evaluating foundation models. CoRR, abs/2304.06364, 2023. doi: 10.48550/arXiv.2304.06364. URL https://doi.org/10.48550/arXiv.2304.06364.

T. Y. Zhuo, M. C. Vu, J. Chim, H. Hu, W. Yu, R. Widyasari, I. N. B. Yusuf, H. Zhan, J. He, I. Paul, S. Brunner, C. Gong, J. Hoang, A. R. Zebaze, X. Hong, W. Li, J. Kaddour, M. Xu, Z. Zhang, P. Yadav, and et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025. OpenReview.net, 2025. URL https://openreview.net/forum?id=YrycTjllL0.

Z. Zhang, Y. Zhong, Y. Jiang, H. Hu, J. Sun, Z. Ge, Y. Zhu, D. Jiang, and X. Jin. Disttrain: Addressing model and data heterogeneity with disaggregated training for multimodal large language models. In Proceedings of the ACM SIGCOMM 2025 Conference, pages 24–38, 2025b.

C. Zhao, L. Zhao, J. Li, Z. Xu, and C. Xu. Deepgemm: clean and efficient fp8 gemm kernels with fine-grained scaling. https://github.com/deepseek-ai/DeepGEMM, 2025.

W. Zhong, R. Cui, Y. Guo, Y. Liang, S. Lu, Y. Wang, A. Saied, W. Chen, and N. Duan. AGIEval: A human-centric benchmark for evaluating foundation models. CoRR, abs/2304.06364, 2023. doi: 10.48550/arXiv.2304.06364. URL https://doi.org/10.48550/arXiv.2304.06364.

T. Y. Zhuo, M. C. Vu, J. Chim, H. Hu, W. Yu, R. Widyasari, I. N. B. Yusuf, H. Zhan, J. He, I. Paul, S. Brunner, C. Gong, J. Hoang, A. R. Zebaze, X. Hong, W. Li, J. Kaddour, M. Xu, Z. Zhang, P. Yadav, and et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025. OpenReview.net, 2025. URL https://openreview.net/forum?id=YrycTjllL0.

45

Appendix

A. Author List

Authors are listed alphabetically by their first name. Names marked with * denote individuals who have departed from our team.

Research & Engineering: Anyi Xu, B. Li, Bangcai Lin, Bing Xue, BingCheng Xian, Bingzheng Xu, Bochao Wu, Bowei Zhang, Boyi Deng, C.C. Yu, Chao Jin, Chaofan Lin, Chen Dong, Chenbing Wang, Chenfan Feng, Chengda Lu*, Chenggang Zhao, Chengqi Deng, Chengyuan Zhang, Chenhao Xu, Chenqi Zhao, Chenze Shao, Chuhao Wang, Chuqi Zhang, Damai Dai, Dejian Yang, Deli Chen, Di Huang, Di Wu, Donghao Li, Erhang Li, Eric Fu, F. Zhou, Fangwei Zhou, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanglin Li, Guanting Chen*, Guoai Cao, Guofan Fan, Guolai Meng, Guowei Li, Haichuan Zhang, Haiyang Ma, Haiyang Shen, Han Li, Han Yu, Han Zhang, Hangyuan Deng, Hanwei Xu, Hanxiang Xu, Hanxun Zhong, Hao Guo, Hao Jiang, Hao Li, Hao Qin, Haodong Wen, Haofen Liang, Haofeng Huang, Haohua Liu, Haoling Zhang, Haoming Luo, Haoran Yang, Haotian Xu*, Haotian Yuan, Haoting Huang, Haowen Luo, Haoyang Cai, Haoyu Chen, Haozhe Ji, Hengran Zhang, Hengrui Wang, Hengxu Wu, Honghui Ding, Hongxuan Tang, Huadong Wang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J. Yang, J.H. Jin, J.H. Zhang, J.X. Zou, Jia Yu, Jiahui Zhou, Jiajun Chen, Jialiang Huang, Jialin Zhao, Jiamin Tang, Jian Zhou, Jianan Tong, Jianwen Li, Jiaqi Zhu, Jiarui Wang, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiaying Ding, Jibai Lu, Jiewen Hu, Jin Yan, Jincheng Zhai, Jingchang Chen, Jingcheng Hu, Jingli Zhou, Jingsheng Xu, Jingting Xiang, Jingyan Yun, Jingyang Yuan, Jingyuan Cheng, Jinhua Zhu, Jinpeng Wang, Jinyi Chen, Jinyi Hu, Jiping Yu, Jueliang Guo, Junbo Pei, Junbo Sun, Junguang Jiang, Junjie Qiu, Junkang Zhou, Junqi Liu, Junren Li, Junxian Li, Junxiao Song, Junyi Guo, Kai Dong, Kaifeng Chen, Kaige Gao, Kang Guan, Kangdong Yuan, Ke Hong, Ke Xu, Kefan Zhao, Kexin Ji, Kexin Zhang, Kexing Zhou, Kuai Yu, Lan Zhang, Lean Wang, Lecong Zhang, Lei Wang, Letian Gao, Liang Zhao, Liansheng Xu, Lihua Guo, Lingxiao Luo, Lingyue Fu, Litao Deng, Litong Wang, Liyue Zhang, Longhao Chen, Lu Chen, Luotian Huang, Luyao Ma, Luyao Wang, M.S. Di, Max Mei, Menghao Ye, Miao Cui, Mingchuan Zhang, Minghua Zhang*, Minghui Tang, Mingjing Zhang, Mingqi Wei, Mingshu Chen, Mingxing Liu, Mingxu Zhou, Mingyu Xu, Mingyu Yang, Mingze Wang, Muyang Chen, Ni Shentu, Ning Wang, Niufang Ning, Panpan Huang, Peixin Cong, Peiyi Wang, Peiyuan Xin, Pengfei Ren, Pengfei Yan, Pengle Zhang, Qi Kang, Qi Tang, Qiancheng Wang, Qiang Li, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qizhou Guo, Rongxian Xu, Rui Ding, Rui Hu, Rui Tian, Rui Yu, Ruidong Zhu, Ruifan Xu, Ruihan Yang, Ruihang Xia, Ruijie Lu, Ruilin Geng, Ruipeng Hong, Ruiqi Ge, Ruisong Zhang, Ruize Sun, Ruizhe Pan, Runji Wang, Runqian Chen, Runxin Xu, Ruohong Tian, Ruomeng Shen, Ruoyu Zhang, Ryan X., S.H. Liu, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoyuan Chen, Shengding Hu, Shengkai Lin, Shengwen Ran, Shengyu Liu, Shengyuan Jia, Shi Bai, Shi Feng, Shicheng Xu, Shichun Liu, Shiqiang Hu*, Shirong Ma, Shiyu Wang, Shiyuan Feng, Shufan Gong, Shuhan Lin, Shuiping Yu, Shunfeng Zhou, Shuo Yang, Shuomeng Wang, Shuting Guo, Shuting Pan, Shuying Yu, Sinuo Cao, Siyi Lin, Sizhe Chen, Songyang Chen, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji*, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tongrui Xiong, Wangding Zeng, Wei Liu, Wei Zhang, Weibin Xu, Weihao Zeng, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao*, Wenjun Gao, Wenkai Shao, Wenkai Yang, Wenli Zhang, Wenlu Wang, Wenlve Huang, Wenqian Yan, Wentao Zhang, Xi Gao, Xiang He, Xiang Li, Xiangli Li, Xiangwen Wang, Xiangying Zhang, Xiankui Wei, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojian Qu, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaoyao Zou, Xiaoyuan Li, Xicheng Guo, Xieting Chu,

附录

A. 作者名单

作者按名字的字母顺序排列。带 * 标记的姓名表示已经离开团队的成员。

研究与工程： Anyi Xu, B. Li, Bangcai Lin, Bing Xue, BingCheng Xian, Bingzheng Xu, Bochao Wu, Bowei Zhang, Boyi Deng, C.C. Yu, Chao Jin, Chaofan Lin, Chen Dong, Chenbing Wang, Chenfan Feng, Chengda Lu*, Chenggang Zhao, Chengqi Deng, Chengyuan Zhang, Chenhao Xu, Chenqi Zhao, Chenze Shao, Chuhao Wang, Chuqi Zhang, Damai Dai, Dejian Yang, Deli Chen, Di Huang, Di Wu, Donghao Li, Erhang Li, Eric Fu, F. Zhou, Fangwei Zhou, Fangyun Lin, Fangzhou Yuan, Feiyu Xia, Fucong Dai, Guangbo Hao, Guanglin Li, Guanting Chen*, Guoai Cao, Guofan Fan, Guolai Meng, Guowei Li, Haichuan Zhang, Haiyang Ma, Haiyang Shen, Han Li, Han Yu, Han Zhang, Hangyuan Deng, Hanwei Xu, Hanxiang Xu, Hanxun Zhong, Hao Guo, Hao Jiang, Hao Li, Hao Qin, Haodong Wen, Haofen Liang, Haofeng Huang, Haohua Liu, Haoling Zhang, Haoming Luo, Haoran Yang, Haotian Xu*, Haotian Yuan, Haoting Huang, Haowen Luo, Haoyang Cai, Haoyu Chen, Haozhe Ji, Hengran Zhang, Hengrui Wang, Hengxu Wu, Honghui Ding, Hongxuan Tang, Huadong Wang, Huanqi Cao, Huazuo Gao, Hui Qu, Hui Zeng, J. Yang, J.H. Jin, J.H. Zhang, J.X. Zou, Jia Yu, Jiahui Zhou, Jiajun Chen, Jialiang Huang, Jialin Zhao, Jiamin Tang, Jian Zhou, Jianan Tong, Jianwen Li, Jiaqi Zhu, Jiarui Wang, Jiasheng Ye, Jiashi Li, Jiaxin Xu, Jiaying Ding, Jibai Lu, Jiewen Hu, Jin Yan, Jincheng Zhai, Jingchang Chen, Jingcheng Hu, Jingli Zhou, Jingsheng Xu, Jingting Xiang, Jingyan Yun, Jingyang Yuan, Jingyuan Cheng, Jinhua Zhu, Jinpeng Wang, Jinyi Chen, Jinyi Hu, Jiping Yu, Jueliang Guo, Junbo Pei, Junbo Sun, Junguang Jiang, Junjie Qiu, Junkang Zhou, Junqi Liu, Junren Li, Junxian Li, Junxiao Song, Junyi Guo, Kai Dong, Kaifeng Chen, Kaige Gao, Kang Guan, Kangdong Yuan, Ke Hong, Ke Xu, Kefan Zhao, Kexin Ji, Kexin Zhang, Kexing Zhou, Kuai Yu, Lan Zhang, Lean Wang, Lecong Zhang, Lei Wang, Letian Gao, Liang Zhao, Liansheng Xu, Lihua Guo, Lingxiao Luo, Lingyue Fu, Litao Deng, Litong Wang, Liyue Zhang, Longhao Chen, Lu Chen, Luotian Huang, Luyao Ma, Luyao Wang, M.S. Di, Max Mei, Menghao Ye, Miao Cui, Mingchuan Zhang, Minghua Zhang*, Minghui Tang, Mingjing Zhang, Mingqi Wei, Mingshu Chen, Mingxing Liu, Mingxu Zhou, Mingyu Xu, Mingyu Yang, Mingze Wang, Muyang Chen, Ni Shentu, Ning Wang, Niufang Ning, Panpan Huang, Peixin Cong, Peiyi Wang, Peiyuan Xin, Pengfei Ren, Pengfei Yan, Pengle Zhang, Qi Kang, Qi Tang, Qiancheng Wang, Qiang Li, Qihao Zhu, Qingyang Li, Qinyu Chen, Qiushi Du, Qizhou Guo, Rongxian Xu, Rui Ding, Rui Hu, Rui Tian, Rui Yu, Ruidong Zhu, Ruifan Xu, Ruihan Yang, Ruihang Xia, Ruijie Lu, Ruilin Geng, Ruipeng Hong, Ruiqi Ge, Ruisong Zhang, Ruize Sun, Ruizhe Pan, Runji Wang, Runqian Chen, Runxin Xu, Ruohong Tian, Ruomeng Shen, Ruoyu Zhang, Ryan X., S.H. Liu, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shaofei Cai, Shaoheng Nie, Shaoyuan Chen, Shengding Hu, Shengkai Lin, Shengwen Ran, Shengyu Liu, Shengyuan Jia, Shi Bai, Shi Feng, Shicheng Xu, Shichun Liu, Shiqiang Hu*, Shirong Ma, Shiyu Wang, Shiyuan Feng, Shufan Gong, Shuhan Lin, Shuiping Yu, Shunfeng Zhou, Shuo Yang, Shuomeng Wang, Shuting Guo, Shuting Pan, Shuying Yu, Sinuo Cao, Siyi Lin, Sizhe Chen, Songyang Chen, Songyang Zhou, Tao Ni, Tao Yun, Tian Jin, Tian Pei, Tian Ye, Tianle Lin, Tianran Ji*, Tianyi Cui, Tianyuan Yue, Tingting Yu, Tongrui Xiong, Wangding Zeng, Wei Liu, Wei Zhang, Weibin Xu, Weihao Zeng, Weilin Zhao, Wen Liu, Wenfeng Liang, Wenjie Pang, Wenjing Luo, Wenjing Yao*, Wenjun Gao, Wenkai Shao, Wenkai Yang, Wenli Zhang, Wenlu Wang, Wenlve Huang, Wenqian Yan, Wentao Zhang, Xi Gao, Xiang He, Xiang Li, Xiangli Li, Xiangwen Wang, Xiangying Zhang, Xiankui Wei, Xiao Bi, Xiaodong Liu, Xiaohan Wang, Xiaojian Qu, Xiaokang Chen, Xiaokang Zhang, Xiaotao Nie, Xiaoyao Zou, Xiaoyuan Li, Xicheng Guo, Xieting Chu,

Xin Cheng, Xin Liu, Xin Xie, Xinbo Xu, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xintong Yao, Xinyang Chen, Xinyong Jiang, Xinyu Yang, Xinyu Yang, Xu Chen, Xuanyu Wang, Xubei Zhong, Xuecheng Su, Xuejie Liu, Xuheng Lin, Xujie Fan, Xuncheng Zhao, Xuwei Fu, Y.C. Yan, Y.H. Jiang, Y.T. Wu*, Y.W. M., Y.Z. Wang, Yafei Gao, Yang Yang, Yang Zhang, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Meng, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yaoyang Ye, Yehang Yin, Yexinrui Wu, Yi Qian, Yi Tao, Yi Yu, Yichao Zhang, Yichen Jiang, Yicheng Wang, Yifan Ding, Yifan Shi, Yifeng Peng, Yifeng Zhai, Yijia Wu, Yiliang Xiong, Yilun Wang, Ying He, Ying Zhou*, Yingjia Luo, Yinmin Zhong, Yiping Wang, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyao Yang, Yiyuan Liu, Yizai Cai, Yizhen Wei, Yizhi Wang, Yonglun Yang, Yongqi Zhuo, Yongqiang Guo, Yongtong Wu, Yu Wu, Yu Zhang, Yuan Bian, Yuan Cheng, Yuan Ou, Yuan Sun, Yuanfan Xu, Yuanhang Sun, Yuanhao Li, Yuchen Liu, Yuchen Yao, Yudong Han, Yuduan Wang, Yuhan Wu, Yuhao Meng, Yuheng Zou, YuKun Li, Yunchuan Wang, Yunfan Xiao, Yunfan Xiong, Yupeng Chen, Yuqian Cao, Yuqian Wang, Yuqing Chen, Yushun Zhang, Yutong Lin, Yuwei Xiao, Yuxian Gu, Yuxiang Chen, Yuxiang Huang, Yuxiang Luo, Yuxiang You, Yuxin Chen, Yuxin Xiang, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhe Guo, Yuzhen Huang, Yuzhuo Bai, Z.Y. Z., Zanlin Ni, Zehao Wang, Zehua Zhao, Zehui Ren, Zejun Zhao, Zhangli Sha, Zhanying Wang, Zhaochen Zhang, Zhaoshuai Du, Zhe Fu, Zhean Xu, Zhenda Xie, Zheng Liu, Zhengyan Zhang, Zhenhua Dong, Zhewen Hao, Zhibang Wang, Zhibin Gou, Zhicheng Ma, Zhihao Li, Zhihong Shao, Zhihuan Huang, Zhijie Li, Zhirui Lu, Zhixian Huang, Zhixuan Chen, Zhixuan Chen, Zhixuan Pan, Zhiyu Wu, Zhizhou Ren, Zhu He, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihui Gu, Zijia Zhu, Zili Zhang, Zilin Li, Zilong Hou, Zilong Lyu, Ziqiao Wang, Ziwei Xie, Ziya Zhang, Ziyi Gao, Zizheng Pan, Zonglin Li, Zongqing Yao, Zui Chen, Zuofan Wu

Business & Compliance: Chenchen Ling, Chengyu Hou, Chong Chen, D. Li, Di Qi, Dongjie Ji, Fang Wei, Fanyi Xia, Fei Xie, Feiyi Tan, Hailong Guo, Haiyan Zhai, Hui Zhou, Huihui Tan, Huijie Li, Jia Luo, Jia Song, Jialu Cai, Jian Liang, Jiangting Zhou, Jiaqi Gao, Jiayi Shao, Jie Chen, Jieyu Yang, Jin Chen, Jingde Zhang, Jingzi Zhou, Jinqian Wang, Jinyang Liu, JinZhao Sun, Junhua Ling, Junmin Zheng, Kaicheng Yang, Ke Xu, Le Su, Leyi Xia, Liangfeng Ding, Lin Zhuo, Linwang Ma, Linyan Zhu, Liyu Cai, Luqi Yao, M.K. Zhang, Meng Li, Miao Lin, Miaojun Wang, Min Zhang, Mingming Li, Mingming Wang, Mingze Yin, Minmin Han, Nan Cao, Ning Wang, Ningxin Ma, Panpan Wang, Peihan Lin, Peng Sun, Peng Zhang, Qian Ying, Qiang Xiang, Qiao Wang, Qingmiao Mao, Qiwei Jiang, Rongli Jin, Ruyi Chen, Sha Tao, Shangmian Sun, Shaoqing Wu, Shichao Zou, Si Lei, Tianyang Zhang, Tianyu Sun, Tingting Yin, W.L. Xiao, Wei An, Wei Li, Wei Wang, Weiwei Lin, Wenqing Hou, X. Lin, Xiangfei Meng, Xianzhu Huang, Xiao Peng, Xiaoqian Li, Xiaoting Zhang, Xiaowen Sun, Xiaoxiang Wang, Xiaoyu Ye, Xinrou Zhang, Xinyu Zhang, Xue Cao, Xueyin Chen, Yanan Zhou, Yanhong Xu, Yao Xia, Yao Xu, Yi Shao, Yihong Zhang, Yiling Ma, Ying Tang, Yining Lou, Yiru Chen, Yishi Piao, Yixuan Chen, Yong Xiong, Yuchen Xuan, Yuehan Yang, Yuer Xu, Yukun Zha, Yunxian Ma, Yuping Lin, Yuting Yan, Yutong Xie, Yuwen Sheng, Yuxuan Zhu, Zekai Zhang, Zhe Ju, Zhenzhen Lin, Zheren Gao, Zheyang Sun, Zhigang Yan, Zhongyu Wu, Zi Wang, Zihua Qu, Ziling Yan, Ziyi Wan

B. Evaluation Details

B.1. Scaffold Configurations

All scaffolds run in Linux task containers using the shared evaluation settings in Table 4. Each run starts from the benchmark task description and uses the prompts, task templates, and tool definitions supplied by the scaffold's runtime or evaluation integration. We add no experimental system prompt.

Xin Cheng, Xin Liu, Xin Xie, Xinbo Xu, Xingchao Liu, Xingchen Liu, Xingkai Yu, Xingyou Li, Xintong Yao, Xinyang Chen, Xinyong Jiang, Xinyu Yang, Xinyu Yang, Xu Chen, Xuanyu Wang, Xubei Zhong, Xuecheng Su, Xuejie Liu, Xuheng Lin, Xujie Fan, Xuncheng Zhao, Xuwei Fu, Y.C. Yan, Y.H. Jiang, Y.T. Wu*, Y.W. M., Y.Z. Wang, Yafei Gao, Yang Yang, Yang Zhang, Yanru Ma, Yanwen Huang, Yao Li, Yao Li, Yao Meng, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yaoyang Ye, Yehang Yin, Yexinrui Wu, Yi Qian, Yi Tao, Yi Yu, Yichao Zhang, Yichen Jiang, Yicheng Wang, Yifan Ding, Yifan Shi, Yifeng Peng, Yifeng Zhai, Yijia Wu, Yiliang Xiong, Yilun Wang, Ying He, Ying Zhou*, Yingjia Luo, Yinmin Zhong, Yiping Wang, Yisong Wang, Yixiang Zhang, Yixiao Chen, Yixuan Tan, Yixuan Wei, Yiyang Ma, Yiyao Yang, Yiyuan Liu, Yizai Cai, Yizhen Wei, Yizhi Wang, Yonglun Yang, Yongqi Zhuo, Yongqiang Guo, Yongtong Wu, Yu Wu, Yu Zhang, Yuan Bian, Yuan Cheng, Yuan Ou, Yuan Sun, Yuanfan Xu, Yuanhang Sun, Yuanhao Li, Yuchen Liu, Yuchen Yao, Yudong Han, Yuduan Wang, Yuhan Wu, Yuhao Meng, Yuheng Zou, YuKun Li, Yunchuan Wang, Yunfan Xiao, Yunfan Xiong, Yupeng Chen, Yuqian Cao, Yuqian Wang, Yuqing Chen, Yushun Zhang, Yutong Lin, Yuwei Xiao, Yuxian Gu, Yuxiang Chen, Yuxiang Huang, Yuxiang Luo, Yuxiang You, Yuxin Chen, Yuxin Xiang, Yuxuan Liu, Yuxuan Zhou, Yuyang Zhou, Yuzhe Guo, Yuzhen Huang, Yuzhuo Bai, Z.Y. Z., Zanlin Ni, Zehao Wang, Zehua Zhao, Zehui Ren, Zejun Zhao, Zhangli Sha, Zhanying Wang, Zhaochen Zhang, Zhaoshuai Du, Zhe Fu, Zhean Xu, Zhenda Xie, Zheng Liu, Zhengyan Zhang, Zhenhua Dong, Zhewen Hao, Zhibang Wang, Zhibin Gou, Zhicheng Ma, Zhihao Li, Zhihong Shao, Zhihuan Huang, Zhijie Li, Zhirui Lu, Zhixian Huang, Zhixuan Chen, Zhixuan Chen, Zhixuan Pan, Zhiyu Wu, Zhizhou Ren, Zhu He, Zhuoshu Li, Zhuping Zhang, Zian Xu, Zihao Wang, Zihui Gu, Zijia Zhu, Zili Zhang, Zilin Li, Zilong Hou, Zilong Lyu, Ziqiao Wang, Ziwei Xie, Ziya Zhang, Ziyi Gao, Zizheng Pan, Zonglin Li, Zongqing Yao, Zui Chen, Zuofan Wu

业务与合规： Chenchen Ling, Chengyu Hou, Chong Chen, D. Li, Di Qi, Dongjie Ji, Fang Wei, Fanyi Xia, Fei Xie, Feiyi Tan, Hailong Guo, Haiyan Zhai, Hui Zhou, Huihui Tan, Huijie Li, Jia Luo, Jia Song, Jialu Cai, Jian Liang, Jiangting Zhou, Jiaqi Gao, Jiayi Shao, Jie Chen, Jieyu Yang, Jin Chen, Jingde Zhang, Jingzi Zhou, Jinqian Wang, Jinyang Liu, JinZhao Sun, Junhua Ling, Junmin Zheng, Kaicheng Yang, Ke Xu, Le Su, Leyi Xia, Liangfeng Ding, Lin Zhuo, Linwang Ma, Linyan Zhu, Liyu Cai, Luqi Yao, M.K. Zhang, Meng Li, Miao Lin, Miaojun Wang, Min Zhang, Mingming Li, Mingming Wang, Mingze Yin, Minmin Han, Nan Cao, Ning Wang, Ningxin Ma, Panpan Wang, Peihan Lin, Peng Sun, Peng Zhang, Qian Ying, Qiang Xiang, Qiao Wang, Qingmiao Mao, Qiwei Jiang, Rongli Jin, Ruyi Chen, Sha Tao, Shangmian Sun, Shaoqing Wu, Shichao Zou, Si Lei, Tianyang Zhang, Tianyu Sun, Tingting Yin, W.L. Xiao, Wei An, Wei
Li, Wei Wang, Weiwei Lin, Wenqing Hou, X. Lin, Xiangfei Meng, Xianzhu Huang, Xiao Peng, Xiaoqian Li, Xiaoting Zhang, Xiaowen Sun, Xiaoxiang Wang, Xiaoyu Ye, Xinrou Zhang, Xinyu Zhang, Xue Cao, Xueyin Chen, Yanan Zhou, Yanhong Xu, Yao Xia, Yao Xu, Yi Shao, Yihong Zhang, Yiling Ma, Ying Tang, Yining Lou, Yiru Chen, Yishi Piao, Yixuan Chen, Yong Xiong, Yuchen Xuan, Yuehan Yang, Yuer Xu, Yukun Zha, Yunxian Ma, Yuping Lin, Yuting Yan, Yutong Xie, Yuwen Sheng, Yuxuan Zhu, Zekai Zhang, Zhe Ju, Zhenzhen Lin, Zheren Gao, Zheyang Sun, Zhigang Yan, Zhongyu Wu, Zi Wang, Zihua Qu, Ziling Yan, Ziyi Wan

B. 评测细节

B.1. 运行框架配置

所有运行框架都在 Linux 任务容器中运行，使用表 4 中的共享评测设置。每次运行都从基准任务描述开始，并使用由运行框架的运行时或评测集成提供的提示词、任务模板和工具定义。我们不添加实验性的系统提示词。

Table 5 | Performance across Claude Code versions at Max reasoning effort.

Benchmark (Metric)	v2.1.105	v2.1.238	v2.1.251	v2.1.259	Average
DeepSWE v1.1 (Resolved)	68.4	68.7	69.8	68.6	68.9
Terminal-Bench v2.1 (Pass@1)	87.3	88.4	88.0	87.6	87.8

Note. Average is computed from the unrounded Pass@1 scores of the four versions.

- **Claude Code (v2.1.105, v2.1.238, v2.1.251, v2.1.259).** We use the Claude Agent SDK with each version's native tool interface. Table 4 reports v2.1.251; Table 5 compares all four versions.
- **Codex (v0.147.0).** We use standard app-server mode with adapted tool schemas.
- **OpenCode (v1.18.15).** We use the `build` agent with shell and file tools, and native task delegation.
- **Pi (v0.84.2).** We use RPC mode with file and shell tools plus a search extension exposing `search`, `open_page`, and `find_in_page`.
- **mini-SWE.** We use the `mini_swe_v2` port[1] with a single `bash` tool, requiring a tool call each turn and a submission marker to finish.
- **DeepSeek Harness (DSH).** We use Minimal with a single `bash` tool; Standard with the full `sdk` profile and 26 initial function tools, including web search and fetch; and PTC with `run_code` for TypeScript programs using 24 underlying tools. Standard and PTC use v0.1.1+ custom.202609011522.

B.2. Reasoning Efforts across Scaffolds

Across all six panels in Figure 11, raising the reasoning-effort setting lengthens the trajectories: mean output tokens per trajectory grow monotonically with effort in every scaffold–benchmark pair. Pass@1 tracks this growth only loosely. It improves overall, but the response is not monotone, with plateaus and dips at intermediate settings in most panels. The three scaffolds are also calibrated differently: on DeepSWE v1.1, Claude Code is the flattest curve and spends comparatively few extra tokens, whereas DeepSeek Harness (Minimal) starts lowest and gains the most at the largest token cost, with mini-SWE in between. On Terminal-Bench v2.1 the three scaffolds are compressed into a narrow band, and the ordering at maximum effort favours DeepSeek Harness (Minimal), followed by mini-SWE and Claude Code: scaffold choice matters at least as much as the effort tier once the task is nearly saturated.

B.3. Detailed Results of Reasoning Benchmarks across Reasoning Efforts

Figure 12 shows that the reasoning-effort setting gives smooth, well-behaved control over both output length and accuracy, consistently across all eight benchmarks, which span competition mathematics, science QA, open-domain knowledge and code. On the length side, raising the effort from 25 to 100 scales the average response predictably on every benchmark – a uniform 2.0–3.1× increase, from 4.6k to 11.4k tokens per response on AIME 2026 and from 29.1k to 86.1k on MathArena Apex 2025 – with no runaway growth or anomalies, so the compute cost of any tier can be estimated in advance. Accuracy follows the same smooth trajectory and responds in the right direction on every benchmark, with no benchmark ever degrading as the effort increases: MathArena Apex 2025 gains +40.3 points (25.3%→65.6%) and Apex 2025 Shortlist +11.5, while even the already-saturated benchmarks remain stable (GPQA Diamond

[1] mini-swe-agent, commit `04d809ceab9d`.

Table 5 | Performance across Claude Code versions at Max reasoning effort.

Benchmark (Metric)	v2.1.105	v2.1.238	v2.1.251	v2.1.259	Average
DeepSWE v1.1 (Resolved)	68.4	68.7	69.8	68.6	68.9
Terminal-Bench v2.1 (Pass@1)	87.3	88.4	88.0	87.6	87.8

Note. Average is computed from the unrounded Pass@1 scores of the four versions.

- Claude Code（v2.1.105、v2.1.238、v2.1.251、v2.1.259）。我们使用 Claude Agent SDK，以及每个版本的原生工具接口。表 4 报告 v2.1.251，表 5 比较全部四个版本。
- Codex（v0.147.0）。我们使用标准 app-server 模式，并适配工具模式。
- OpenCode（v1.18.15）。我们使用 build 智能体，配备 shell 与文件工具，以及原生任务委派功能。
- Pi（v0.84.2）。我们使用 RPC 模式，配备文件与 shell 工具，并通过搜索扩展提供 search、open_page 和 find_in_page。
- mini-SWE。我们使用 mini_swe_v2 移植版[1]，仅配备一个 bash 工具，要求每轮进行一次工具调用，并用提交标记结束任务。
- DeepSeek Harness（DSH）。我们使用三种模式：Minimal 仅有一个 bash 工具；Standard 采用完整 sdk 配置，初始提供 26 个函数工具，包括网页搜索与抓取；PTC 则通过 run_code 执行使用 24 个底层工具的 TypeScript 程序。Standard 和 PTC 使用 v0.1.1+ custom.202609011522。

B.2. 不同运行框架下的推理强度

在图 11 的全部六个面板中，提高推理强度设置都会延长轨迹：在每一种运行框架与基准的组合中，每条轨迹的平均输出 token 数都随强度单调增长。Pass@1 与这种增长的对应关系则较弱。它总体有所改善，但响应并非单调，多数面板在中间设置上出现平台期或下降。三个运行框架的校准也不同：在 DeepSWE v1.1 上，Claude Code 的曲线最平坦，额外消耗的 token 相对较少；DeepSeek Harness（Minimal）的起点最低，以最大的 token 成本获得最大提升；mini-SWE 介于两者之间。在 Terminal-Bench v2.1 上，三个框架的表现压缩在狭窄区间内；在最高强度下，DeepSeek Harness（Minimal）领先，其次是 mini-SWE 和 Claude Code。当任务已接近饱和时，运行框架的选择至少与推理强度档位同样重要。

B.3. 不同推理强度下推理基准的详细结果

图 12 表明，推理强度设置能够平滑、稳定地控制输出长度和准确率，而且在覆盖竞赛数学、科学问答、开放域知识和编程的全部八个基准上都保持一致。在长度方面，将强度从 25 提高到 100，会使每个基准的平均响应长度以可预测的方式扩大，统一增长至 2.0 – 3.1×：AIME 2026 上从每次响应 4.6k token 增至 11.4k，MathArena Apex 2025 上从 29.1k 增至 86.1k，没有失控增长或异常，因此每个档位的计算成本都可以事先估计。准确率也沿相同的平滑轨迹变化，在每个基准上都朝正确方向响应，没有任何基准随着强度增加而退化：MathArena Apex 2025 提高 +40.3 个百分点（25.3%→65.6%），Apex 2025 Shortlist 提高 +11.5 个百分点；即使已趋于饱和的基准也保持稳定（GPQA Diamond

[1] mini-swe-agent，提交 04d809ceab9d。

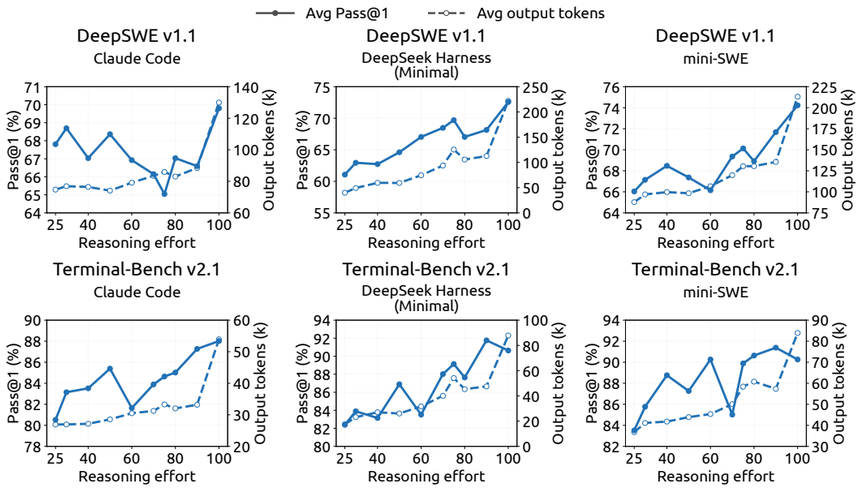


Figure 11 | Reasoning effort drives trajectory length consistently but correlates only weakly with accuracy across coding scaffolds. Each panel plots Pass@1 (%, solid, left axis) and mean output tokens per trajectory (k, dashed, right axis) against the reasoning-effort setting, for DeepSWE v1.1 (top row) and Terminal-Bench v2.1 (bottom row) under three agent scaffolds: Claude Code, DeepSeek Harness (Minimal) and mini-SWE. All panels come from the same checkpoint.

+1.3, LiveCodeBench +2.6), and AIME 2026 reaches a full 100%. The model thus exposes a single, reliable knob that moves the cost–accuracy operating point in a controlled and predictable way, allowing each deployment to select the effort tier that matches its latency and compute budget without sacrificing accuracy.

C. Exponential Token Penalty in Reasoning Effort Control

The scalar effort variable provides a deployment-time control over the cost–quality trade-off without imposing a hard token budget. During reinforcement learning, lower effort levels apply a stronger token penalty, whereas higher effort levels permit more computation. This section gives a simplified motivation for the exponential penalty schedule.

For a trajectory with ℓ reasoning tokens generated at effort level b, the length deduction is

$$r^{\text{len}}(\ell, b) = -\min\left\{C_{\max}, k(b)\frac{\ell}{L_{\text{norm}}}\right\}, \tag{11}$$

where L_{norm} is a reference length and $C_{\max}$ caps the deduction. The effort-dependent token-penalty coefficient is

$$k(b) = k_0 \exp\left(-\frac{b - b_{\min}}{\tau}\right), \tag{12}$$

where k_0 is the penalty coefficient at the lowest effort level $b_{\min}$ and τ controls the rate of penalty decay.

To motivate this choice, consider a fixed problem x. Let $p_x(\ell)$ denote its probability of being solved after ℓ reasoning tokens. In the uncapped region, define the preferred reasoning length

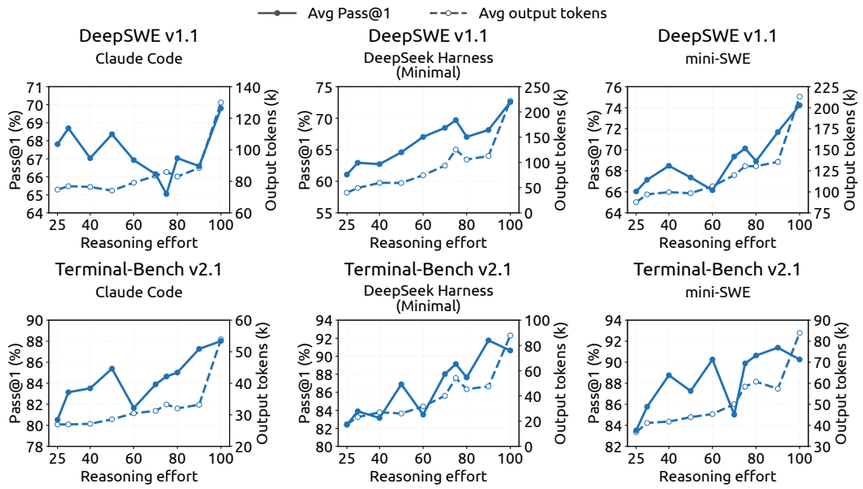


图 11 | 在不同编程运行框架下，推理强度持续推动轨迹长度增加，但与准确率的相关性较弱。每个面板展示推理强度设置变化时的 Pass@1（百分比，实线，左轴）和每条轨迹的平均输出 token 数（千，虚线，右轴）。上排为 DeepSWE v1.1，下排为 Terminal-Bench v2.1；使用的三种智能体运行框架分别是 Claude Code、DeepSeek Harness（Minimal）和 mini-SWE。所有面板都来自同一检查点。

提高 +1.3 个百分点，LiveCodeBench 提高 +2.6 个百分点），AIME 2026 则达到完整的 100%。因此，模型提供了单一且可靠的调节接口，以可控、可预测的方式移动成本—准确率运行点，让每次部署都能选择符合其延迟和计算预算的推理强度档位，而不牺牲准确性。

C. 推理强度控制中的指数 token 惩罚

标量推理强度变量无需施加硬性 token 预算，就能在部署时控制成本与质量的权衡。在强化学习中，较低的推理强度级别施加更强的 token 惩罚，较高的级别则允许更多计算。本节对指数惩罚调度的动机作简化说明。

对于一条包含 ℓ 个推理 token、在推理强度级别 b 下生成的轨迹，其长度扣分为

$$r^{\text{len}}(\ell, b) = -\min\left\{C_{\max}, k(b)\frac{\ell}{L_{\text{norm}}}\right\}, \tag{11}$$

其中，L_{norm} 是参考长度，$C_{\max}$ 限制扣分上限。依赖于推理强度的 token 惩罚系数为

$$k(b) = k_0 \exp\left(-\frac{b - b_{\min}}{\tau}\right), \tag{12}$$

其中，k_0 是最低推理强度级别 $b_{\min}$ 下的惩罚系数，τ 控制惩罚衰减速率。

为说明这一选择的动机，考虑一个固定问题 x。令 $p_x(\ell)$ 表示经过 ℓ 个推理 token 后该问题被解出的概率。在尚未触及上限的区域，定义偏好的推理长度

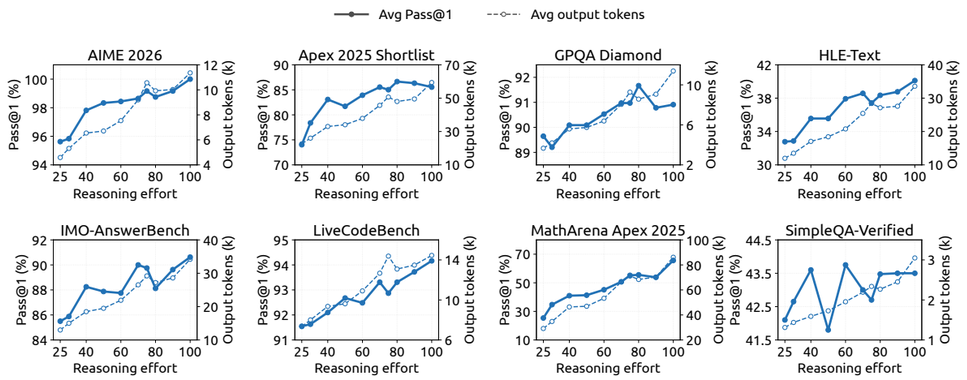


Figure 12 | Performance and output length as a function of reasoning effort on eight reasoning-intensive benchmarks. Each panel plots Pass@1 (solid, left axis) and mean output tokens per response (dashed, right axis) as the reasoning-effort value is varied from 25 to 100.

$\ell_x^*(b)$ by

$$\ell_x^*(b) \in \arg\max_{\ell \geqslant 0} \left[p_x(\ell) - k(b) \frac{\ell}{L_{\text{norm}}} \right]. \tag{13}$$

For an interior optimum, the first-order condition is

$$p_x'\left(\ell_x^*(b)\right) = \frac{k(b)}{L_{\text{norm}}}, \tag{14}$$

where $p_x'(\ell) = dp_x(\ell)/d\ell$ is the marginal improvement in solve probability from additional reasoning.

We assume that this marginal benefit decays approximately exponentially over the relevant operating range:

$$p_x'(\ell) \approx a_x \exp\left(-\frac{\ell}{s_x}\right), \tag{15}$$

where $a_x > 0$ is an instance-dependent scale and $s_x > 0$ determines the decay rate. Substituting Eqs. (12) and (15) into Eq. (14) gives

$$\ell_x^*(b) \approx C_x - s_x \log k_0 + \frac{s_x}{\tau}(b - b_{\min}), \tag{16}$$

where $C_x = s_x \log(a_x L_{\text{norm}})$ is independent of b. Thus, the exponential penalty schedule produces a simple first-order affine trend between the requested effort and the preferred reasoning length under this local model.

For two effort levels $b_2 > b_1$, the corresponding predicted length difference is

$$\ell_x^*(b_2) - \ell_x^*(b_1) \approx \frac{s_x}{\tau}(b_2 - b_1). \tag{17}$$

Consequently, k_0 mainly controls the overall pressure toward shorter reasoning, while τ controls the predicted sensitivity to effort.

This derivation is a local reward-level approximation, not a claim that measured average lengths must be linear or pointwise monotonic. Realized behavior may deviate because the

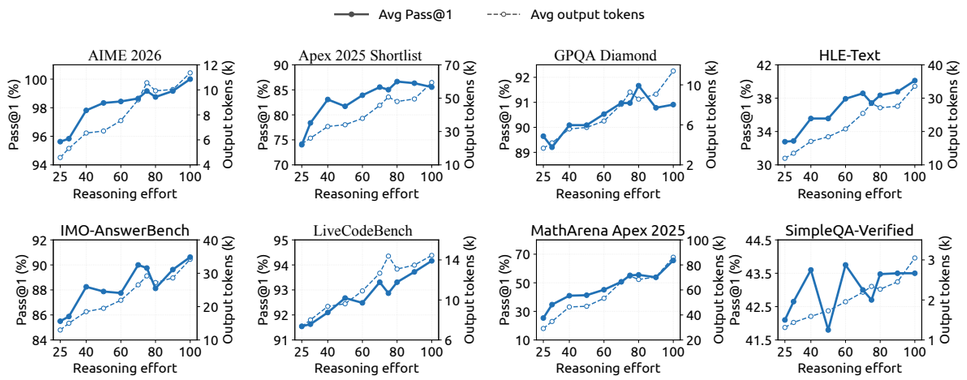


图 12 | 在八个推理密集型基准上，性能与输出长度随推理强度的变化。每个面板展示推理强度值从 25 变化到 100 时的 Pass@1（实线，左轴）和每次响应的平均输出 token 数（虚线，右轴）。

$\ell_x^*(b)$ by

$$\ell_x^*(b) \in \arg\max_{\ell \geqslant 0} \left[p_x(\ell) - k(b) \frac{\ell}{L_{\text{norm}}} \right]. \tag{13}$$

对于内部最优解，一阶条件为

$$p_x'\left(\ell_x^*(b)\right) = \frac{k(b)}{L_{\text{norm}}}, \tag{14}$$

其中， $p_x'(\ell) = dp_x(\ell)/d\ell$ 表示额外推理带来的求解成功概率的边际提升。

我们假设，在相关运行区间内，这一边际收益近似按指数衰减：

$$p_x'(\ell) \approx a_x \exp\left(-\frac{\ell}{s_x}\right), \tag{15}$$

其中， $a_x > 0$ 是依赖于实例的尺度， $s_x > 0$ 决定衰减速率。将式 (12) 和式 (15) 代入式 (14)，可得

$$\ell_x^*(b) \approx C_x - s_x \log k_0 + \frac{s_x}{\tau}(b - b_{\min}), \tag{16}$$

其中， $C_x = s_x \log(a_x L_{\text{norm}})$ 与 b 无关。因此，在这一局部模型下，指数惩罚调度会使所请求的推理强度与偏好的推理长度之间呈现简单的一阶仿射趋势。

对于两个推理强度级别 $b_2 > b_1$，相应的预测长度差为

$$\ell_x^*(b_2) - \ell_x^*(b_1) \approx \frac{s_x}{\tau}(b_2 - b_1). \tag{17}$$

因此， k_0 主要控制趋向较短推理过程的整体压力，而 τ 控制预测结果对推理强度的敏感程度。

这一推导是奖励层面的局部近似，并不声称实测平均长度必然是线性的，或在每一点上都单调。实际行为可能有所偏离，因为

effort instruction can directly change the reasoning strategy, generation is stochastic, agent trajectories contain different numbers of turns, and subgroup reward normalization changes optimization strength. The analysis also assumes an interior solution for which the penalty cap is inactive. Once the cap is reached, the marginal token penalty becomes zero and the capped region must be considered separately.

推理强度指令可能直接改变推理策略，生成过程具有随机性，智能体轨迹包含的轮次数不同，而子组奖励归一化也会改变优化强度。这一分析还假设存在内部解，且惩罚上限尚未生效。一旦达到上限，边际 token 惩罚就变为零，因此需要单独考虑达到上限后的区域。

51