DeepSeek-V3 技术报告

左元翻译

日期：February 7, 2025

摘　要

我们推出了 DeepSeek-V3，这是一个强大的混合专家（Mixture-of-Experts, MoE）语言模型，总参数量为 6710 亿，每个 Token 激活的参数量为 370 亿。为了实现高效的推理和低成本的训练，DeepSeek-V3 采用了多头潜在注意力（Multi-head Latent Attention, MLA）和 DeepSeekMoE 架构，这两种架构在 DeepSeek-V2 中得到了充分验证。此外，DeepSeek-V3 首创了一种无辅助损失（auxiliary-loss-free）的负载均衡策略，并将多 Token 预测设定为训练目标，以实现更强的性能。我们在 14.8 万亿个多样且高质量的 Token 上进行预训练，得到了 DeepSeek-V3 模型，随后通过监督微调（Supervised Fine-Tuning）和强化学习（Reinforcement Learning）充分挖掘 DeepSeek-V3 的潜力。综合评估表明，DeepSeek-V3 的表现优于其它开源模型，并与领先的闭源模型性能相当。尽管性能卓越，DeepSeek-V3 的完整训练仅需 278.8 万 H800 GPU 小时。此外，DeepSeek-V3 的训练过程非常稳定。在整个训练过程中，我们没有遇到任何不可恢复的损失突增问题，也没有执行任何回滚操作。模型检查点可在https://github.com/deepseek-ai/DeepSeek-V3获取。

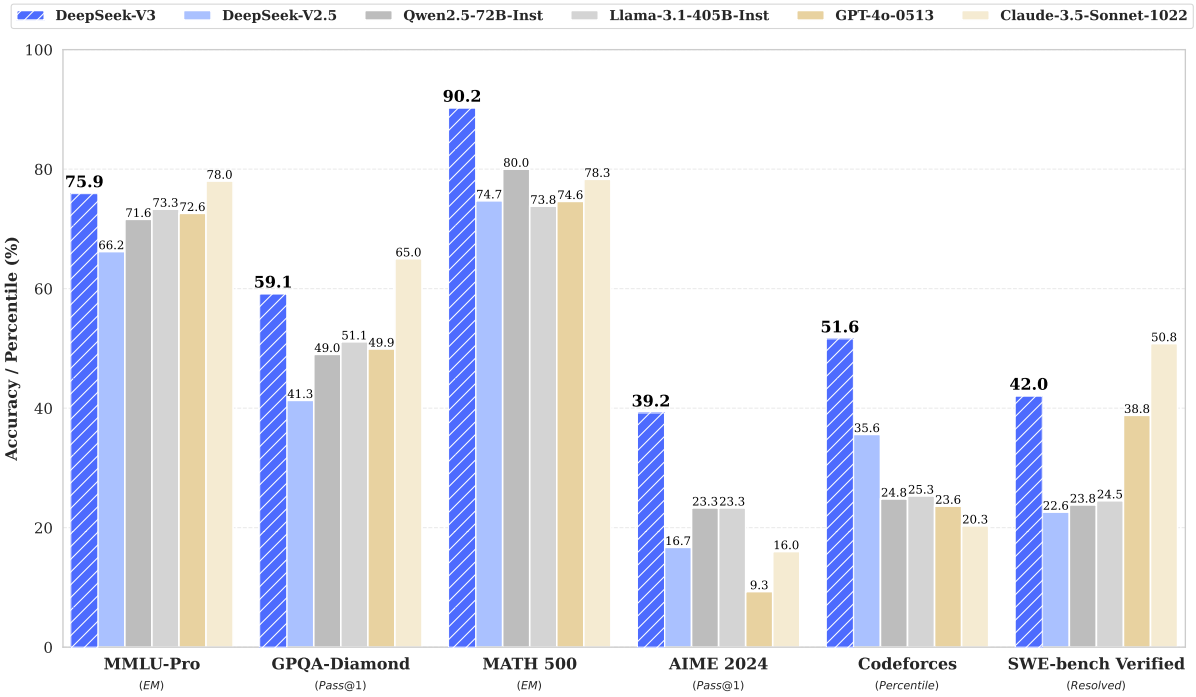


图 1: DeepSeek-V3 和竞品在性能基准测试中的比较。

目录

Training Costs	Pre-Training	Context Extension	Post-Training	Total
in H800 GPU Hours	2664K	119K	5K	2788K
in USD	$5.328M	$0.238M	$0.01M	$5.576M

表 1: DeepSeek-V3 的训练花费，假设 H800 每个 GPU 小时的租赁费用是 $2。

1 简介

近年来，大语言模型（Large Language Models, LLMs）正在经历快速的迭代和进化，逐步缩小与通用人工智能（Artificial General Intelligence, AGI）的差距。除了闭源模型外，开源模型，包括 DeepSeek 系列、LLaMA 系列、Qwen 系列和 Mistral 系列，也在取得显著进展，努力缩小与闭源模型的差距。为了进一步突破开源模型的能力边界，我们继续扩大模型规模，并推出了 DeepSeek-V3，这是一个拥有 6710 亿参数的大型混合专家（Mixture-of-Experts, MoE）模型，其中每个 Token 激活的参数量为 370 亿。

本着以终为始的精神，我们始终致力于实现强大的模型性能同时尽可能的节省训练和推理的成本。因此，在架构方面，DeepSeek-V3 仍然采用多头潜在注意力（MLA）以实现高效推理，以及采用 DeepSeekMoE 以实现低成本的训练。这两种架构在 DeepSeek-V2 中得到了验证，证明了它们在保持模型强大性能的同时仍能实现高效的训练和推理。除了基础架构外，我们还实现了两种额外策略以进一步增强模型能力。首先，DeepSeek-V3 首创了一种无辅助损失的负载均衡策略，旨在最小化负载均衡策略[1]带来的模型性能下降。其次，DeepSeek-V3 将多 Token 预测设定为训练目标。我们观察到，在评估基准上，整体性能提升了。

为了实现高效的训练，我们实现了 FP8 混合精度训练，并对训练框架进行了全面优化。低精度训练作为高效训练模型的一种解决方案，很有前景。其发展与硬件能力的进步密切相关。在本文所描述的工作中，我们引入了 FP8 混合精度训练框架，并首次在超大规模模型上验证了低精度训练的有效性。通过支持 FP8 的计算和存储，我们实现了训练的加速并减少了 GPU 内存的使用量。在训练框架方面，我们设计了 DualPipe 算法以实现高效的流水线并行，DualPipe 算法减少了流水线气泡，并通过计算-通信重叠（computation-communication overlap）隐藏了训练过程中的大部分通信。这种重叠确保了随着模型的进一步扩展，只要我们保持恒定的计算-通信比，就仍然可以在节点间使用细粒度的专家，同时实现近乎零的 All-to-All 通信开销。此外，我们还开发了高效的跨节点 All-to-All 通信内核，以充分利用 InfiniBand 和 NVLink 的带宽。此外，我们精心优化了内存占用，使得在不使用昂贵的张量并行的情况下，也能训练出 DeepSeek-V3。通过这些努力，我们实现了高效的训练。

在预训练阶段，我们在 14.8 万亿个高质量且多样化的 Token 上训练出了 DeepSeek-V3。预训练过程非常稳定。在整个训练过程中，我们没有遇到任何不可恢复的损失突增，也没有执行任何回滚操作。接下来，我们对 DeepSeek-V3 进行了两阶段的上下文长度扩展。在第一阶段，最大上下文长度扩展到 32K，在第二阶段进一步扩展到 128K。随后，我们对 DeepSeek-V3 的基础模型进行了后训练，包括监督微调（Supervised Fine-Tuning, SFT）和强化学习（Reinforcement Learning, RL），以使模型与人类偏好对齐并进一步释放模型的潜力。在后训练阶段，我们从 DeepSeek-R1 系列模型中蒸馏了推理能力，同时保持了模型准确性和生成长度之间的平衡。

我们在广泛的基准测试上对 DeepSeek-V3 进行了评估。尽管训练成本很低，综合评估表明，DeepSeek-V3-Base 已成为当前最强的开源基础模型，尤其是在编程和数学领域。其聊天版本也在标准基准和开放式基准测试中优于其它开源模型，并与领先的闭源模型（包括 GPT-4o 和 Claude-3.5-Sonnet）性能相当。

最后，我们再次强调 DeepSeek-V3 的训练成本非常低，具体见表 1。这是通过我们对算法、框架和硬件的协同优化实现的。在预训练阶段，每万亿 Token 的训练仅需 18 万 H800 GPU 小时，也就是在我们自己的一个由 2048 个 H800 GPU 组成的集群上训练时间仅需 3.7 天。因此，我们花了不到两个月的时间就完

[1]因为负载均衡策略的使用不可避免

成了模型的预训练，并只消耗了 266.4 万个 GPU 小时。结合上下文长度扩展所需的 11.9 万 GPU 小时和后训练所需的 0.5 万 GPU 小时，DeepSeek-V3 的完整训练仅需 278.8 万 GPU 小时。假设 H800 GPU 的租赁价格为每小时 2 美元，我们的总训练成本仅为 557.6 万美元。需要注意的是，上述成本仅包括 DeepSeek-V3 的官方训练成本，不包括在架构、算法或数据上的前期研究和消融实验的相关成本。

我们的主要贡献包括：

架构：创新的负载均衡策略和训练目标

- 基于 DeepSeek-V2 使用的高效架构，我们首创了一种无辅助损失的负载均衡策略，最大限度地减少了因采用负载均衡而导致的性能下降。
- 我们将多 Token 预测（Multi-Token Prediction, MTP）设置为训练目标，并证明其对模型性能有益。它还可以用于推测解码（speculative decoding）以实现推理加速。

预训练：向着终极的训练效率进发！

- 我们设计了一个 FP8 混合精度训练框架，并首次在超大规模模型上验证了 FP8 低精度训练的可行性和有效性。
- 通过算法、框架和硬件的协同设计，我们克服了跨节点 MoE 训练中的通信瓶颈，实现了近乎完全的计算-通信重叠。这显著提高了我们的训练效率并降低了训练成本，使我们能够在不增加额外开销的情况下进一步扩大模型规模。
- 仅以 266.4 万个 H800 GPU 小时的成本，我们在 14.8 万亿 Token 上预训练出了 DeepSeek-V3，它可以说是当前最强的开源基础模型。预训练后的后续训练阶段仅需 10 万 GPU 小时。

后训练：从 DeepSeek-R1 进行知识蒸馏

- 我们引入了一种创新的方法，可以将长思维链（Chain-of-Thought, CoT）模型（特别是 DeepSeek R1 系列模型之一）的推理能力蒸馏到标准大语言模型（尤其是 DeepSeek-V3）中。我们的工作巧妙地将 R1 的验证和反思能力融入了 DeepSeek-V3 模型中，显著提升了模型的推理性能。同时，我们还控制了 DeepSeek-V3 的输出风格和输出长度。

评估结果总结

- **知识**：(1) 在教育类基准测试（如 MMLU、MMLU-Pro 和 GPQA）上，DeepSeek-V3 优于所有其它开源模型，在 MMLU 上得分为 88.5，在 MMLU-Pro 上得分为 75.9，在 GPQA 上得分为 59.1。其性能与领先的闭源模型（如 GPT-4o 和 Claude-Sonnet-3.5）相当，缩小了开源模型与闭源模型在该领域的差距。(2) 在事实性基准测试中，DeepSeek-V3 在 SimpleQA 和中文 SimpleQA 上均表现出优于其他开源模型的性能。尽管在英文事实知识（SimpleQA）上略逊于 GPT-4o 和 Claude-Sonnet-3.5，但在中文事实知识（中文 SimpleQA）上超越了这些模型，凸显了其在中文事实知识方面的优势。
- **编程、数学和推理**: (1) DeepSeek-V3 在所有没有长思维链（CoT）的开源模型和闭源模型中，在数学相关基准测试上实现了最先进的性能。值得注意的是，它在特定基准测试（如 MATH-500）上甚至优于 o1-preview，展示了其强大的数学推理能力。(2) 在编程相关任务中，DeepSeek-V3 成为编程竞赛基准测试（如 LiveCodeBench）中表现最佳的模型，巩固了其在该领域的领先地位。在工程相关任务中，尽管 DeepSeek-V3 的表现略低于 Claude-Sonnet-3.5，但它仍然显著领先于所有其它模型，展示了其在多样化技术基准测试中的竞争力。

在本文的剩余部分，我们首先详细阐述了 DeepSeek-V3 的模型架构（第2节）。随后，介绍了基础设施，包括计算集群、训练框架、对 FP8 训练的支持、推理部署策略以及对未来 AI 硬件的一些设计建议。接下来，我们描述了预训练过程，包括训练数据的构建、超参数设置、长上下文扩展技术、相关评估以及一些讨论（第4节）。之后，我们讨论了后训练过程，包括监督微调（SFT）、强化学习（RL）、相关评估和讨论（第5节）。最后，总结了这项工作，讨论了 DeepSeek-V3 的现有局限性，并提出了未来研究的潜在方向（第6节）。

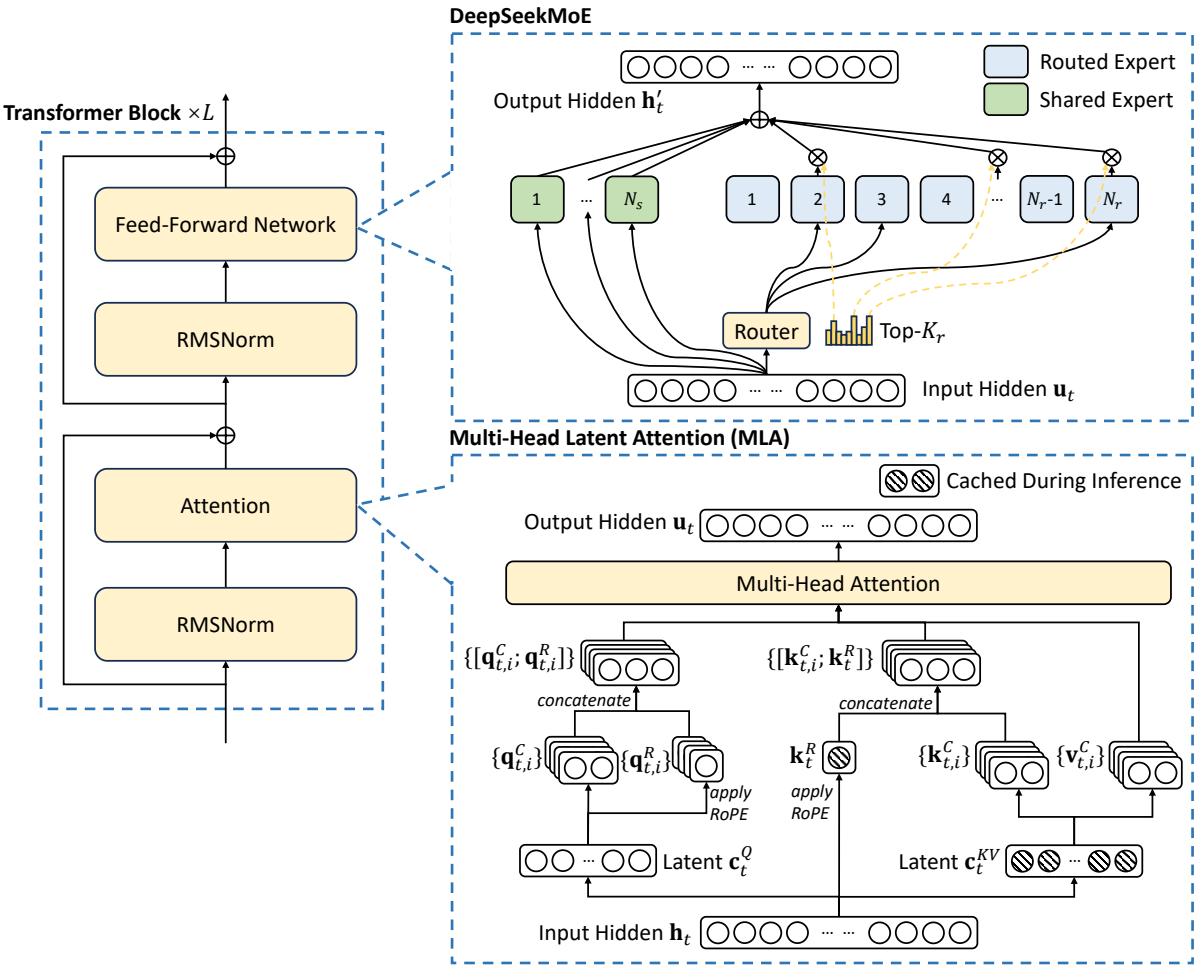


图 2: DeepSeek-V3 的基本架构图，DeepSeek-V3 延续了 DeepSeek-V2 的思想，采用了 MLA 和 DeepSeek-MoE 来实现高效的推理和低成本的训练。

2 架构

首先来介绍 DeepSeek-V3 的基本架构。DeepSeek-V3 的特点是采用多头潜在注意力机制（Multi-head Latent Attention，MLA）以实现高效推理，采用 DeepSeekMoE 以实现模型的低成本训练。随后，我们提出了将多 Token 预测（Multi-Token Prediction，MTP）设定为训练目标的方案，并观察到它能够提升评估基准上的整体性能。对于其它没有明确提及的小细节，DeepSeek-V3 沿用了 DeepSeek-V2 的设置。

2.1 基本架构

DeepSeek-V3 的基本架构仍然没有脱离 Transformer 。为了实现高效的推理和低成本的训练，DeepSeek-V3 还采用了 MLA 和 DeepSeekMoE，这两者已在 DeepSeek-V2 中得到了充分验证。与 DeepSeek-V2 相比，DeepSeek-V3 有一点不同，那就是我们额外引入了一种无辅助损失的负载均衡策略，从而减轻了采用负载平衡策略所带来的性能下降。图 2 展示了 DeepSeek-V3 的基本架构，我们将在本节中简要回顾 MLA 和 DeepSeekMoE 的细节。

2.1.1 多头潜在注意力机制（Multi-Head Latent Attention）

在注意力机制方面，DeepSeek-V3 采用了 MLA 架构。设 d 表示嵌入的维度，n_h 表示注意力头的数量，d_h 表示每个头的维度，$\mathrm{h}_t \in \mathbb{R}^d$ 表示在给定的注意力层中第 t 个 Token 的注意力输入。MLA 的核心是对

注意力的键（keys）和值（values）进行低秩（low-rank）联合压缩，以减少推理过程中键-值（Key-Value，KV）缓存的使用：

$$\boxed{\mathrm{c}_t^{KV}} = W^{DKV}\mathrm{h}_t, \tag{1}$$

$$[\mathrm{k}_{t,1}^C; \mathrm{k}_{t,2}^C; ...; \mathrm{k}_{t,n_h}^C] = \mathrm{k}_t^C = W^{UK}\mathrm{c}_t^{KV}, \tag{2}$$

$$\boxed{\mathrm{k}_t^R} = \mathrm{RoPE}(W^{KR}\mathrm{h}_t), \tag{3}$$

$$\mathrm{k}_{t,i} = [\mathrm{k}_{t,i}^C; \mathrm{k}_t^R], \tag{4}$$

$$[\mathrm{v}_{t,1}^C; \mathrm{v}_{t,2}^C; ...; \mathrm{v}_{t,n_h}^C] = \mathrm{v}_t^C = W^{UV}\mathrm{c}_t^{KV}, \tag{5}$$

其中 $\mathrm{c}_t^{KV} \in \mathbb{R}^{d_c}$ 是键和值的压缩潜在向量；$d_c (\ll d_h n_h)$ 表示 KV 的压缩维度；$W^{DKV} \in \mathbb{R}^{d_c \times d}$ 表示下投影矩阵（down-projection matrix）；$W^{UK}, W^{UV} \in \mathbb{R}^{d_h n_h \times d_c}$ 表示键和值的上投影矩阵（up-projection matrix）；$W^{KR} \in \mathbb{R}^{d_h^R \times d}$ 是用于生成解耦键（decoupled key）的矩阵，解耦键携带了旋转位置嵌入（Rotary Positional Embedding，RoPE）；$\mathrm{RoPE}(\cdot)$ 表示应用 RoPE 矩阵的运算；$[\cdot;\cdot]$ 表示拼接。请注意，对于 MLA，只有蓝框中的向量（即 c_t^{KV} 和 k_t^R）需要在生成过程中缓存，而这可以显著减少 KV 缓存，同时保持与标准多头注意力（Multi-Head Attention，MHA）相当的性能。

对于注意力查询（queries），我们也进行了低秩压缩，这可以减少训练过程中的激活内存：

$$\mathrm{c}_t^Q = W^{DQ}\mathrm{h}_t, \tag{6}$$

$$[\mathrm{q}_{t,1}^C; \mathrm{q}_{t,2}^C; ...; \mathrm{q}_{t,n_h}^C] = \mathrm{q}_t^C = W^{UQ}\mathrm{c}_t^Q, \tag{7}$$

$$[\mathrm{q}_{t,1}^R; \mathrm{q}_{t,2}^R; ...; \mathrm{q}_{t,n_h}^R] = \mathrm{q}_t^R = \mathrm{RoPE}(W^{QR}\mathrm{c}_t^Q), \tag{8}$$

$$\mathrm{q}_{t,i} = [\mathrm{q}_{t,i}^C; \mathrm{q}_{t,i}^R], \tag{9}$$

其中 $\mathrm{c}_t^Q \in \mathbb{R}^{d_c'}$ 是查询（queries）的压缩潜在向量；$d_c' (\ll d_h n_h)$ 表示查询压缩维度；$W^{DQ} \in \mathbb{R}^{d_c' \times d}, W^{UQ} \in \mathbb{R}^{d_h n_h \times d_c'}$ 分别是查询的下投影矩阵和上投影矩阵；$W^{QR} \in \mathbb{R}^{d_h^R n_h \times d_c'}$ 是一个矩阵，用来生成携带 RoPE 的解耦查询。

最终，通过组合注意力查询 ($\mathrm{q}_{t,i}$)、键 ($\mathrm{k}_{j,i}$) 和值 ($\mathrm{v}_{j,i}^C$) 来产生最终的注意力输出 u_t：

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

$$\mathrm{u}_t = W^O[\mathrm{o}_{t,1}; \mathrm{o}_{t,2}; ...; \mathrm{o}_{t,n_h}], \tag{11}$$

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

2.1.2 带有无辅助损失负载均衡的 DeepSeekMoE

DeepSeekMoE 的基本架构。 对于前馈网络（Feed-Forward-Networks），DeepSeek-V3 采用了 DeepSeekMoE 架构。与传统的 MoE 架构（如 GShard）相比，DeepSeekMoE 使用了更细粒度的专家，并将一些专家隔离作为共享专家。设 u_t 表示第 t 个 Token 的前馈网络输入，并计算前馈网络的输出 h_t' 如下：

$$\mathrm{h}_t' = \mathrm{u}_t + \sum_{i=1}^{N_s} \mathrm{FFN}_i^{(s)}(\mathrm{u}_t) + \sum_{i=1}^{N_r} g_{i,t}\,\mathrm{FFN}_i^{(r)}(\mathrm{u}_t), \tag{12}$$

$$g_{i,t} = \frac{g'_{i,t}}{\sum_{j=1}^{N_r} g'_{j,t}}, \tag{13}$$

$$g'_{i,t} = \begin{cases} s_{i,t}, & s_{i,t} \in \mathrm{Topk}(\{s_{j,t} | 1 \le j \le N_r\}, K_r), \\ 0, & \text{otherwise}, \end{cases} \tag{14}$$

$$s_{i,t} = \mathrm{Sigmoid}\left(\mathrm{u}_t{}^T \mathrm{e}_i\right), \tag{15}$$

其中 N_s 和 N_r 分别表示共享专家的数量和路由专家的数量；$\mathrm{FFN}_i^{(s)}(\cdot)$ 和 $\mathrm{FFN}_i^{(r)}(\cdot)$ 分别表示第 i 个共享专家和第 i 个路由专家；K_r 表示激活的路由专家数量；$g_{i,t}$ 是第 i 个专家的门控值；$s_{i,t}$ 是 Token 与专家的亲和度；e_i 是第 i 个路由专家的中心向量；$\mathrm{Topk}(\cdot, K)$ 表示包含 K 个最高分数的集合，这些分数是为第 t 个 Token 和所有路由专家计算的亲和度分数。与 DeepSeek-V2 略有不同，DeepSeek-V3 使用 Sigmoid 函数来计算亲和度分数，并对所有选定的亲和度分数进行归一化，从而生成门控值。

无辅助损失的负载均衡。 对于 MoE 模型，不均衡的专家负载会导致路由崩溃，并在专家并行的场景中降低计算效率。传统的解决方案通常依赖于辅助损失来避免负载不均衡。然而，过大的辅助损失会损害模型性能。为了在负载均衡和模型性能之间实现更好的权衡，我们首创了一种无辅助损失的负载均衡策略，以确保负载均衡。

具体来说，我们为每个专家引入一个偏置项 b_i，并将其添加到相应的亲和度分数 $s_{i,t}$ 中，以确定 top-K 路由：

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

请注意，偏置项仅用于路由。门控值将与前馈网络的输出相乘，且门控值仍然来源于原始的亲和度分数 $s_{i,t}$。在训练过程中，我们会持续监控每个训练步骤中整个批次（batch）的专家负载。在每个步骤结束时，如果有某个专家负载过高，我们会将这个专家对应的偏置项减少 γ；如果某个专家负载不足，则增加相应的 γ，其中 γ 是一个称为偏置更新速度的超参数。通过动态调整，DeepSeek-V3 在训练过程中维持了专家的负载均衡，并且实现了比那些采用了纯辅助损失负载均衡策略的模型更好的性能。

互补的序列级辅助损失。 虽然 DeepSeek-V3 主要依赖于无辅助损失的负载均衡策略，但为了防止单个序列内部的负载极端不均衡，我们还采用了互补的序列级均衡损失：

$$\mathcal{L}_{\mathrm{Bal}} = \alpha \sum_{i=1}^{N_r} f_i P_i, \tag{17}$$

$$f_i = \frac{N_r}{K_r T} \sum_{t=1}^{T} \mathbb{1}\left(s_{i,t} \in \mathrm{Topk}(\{s_{j,t} | 1 \le j \le N_r\}, K_r)\right), \tag{18}$$

$$s'_{i,t} = \frac{s_{i,t}}{\sum_{j=1}^{N_r} s_{j,t}}, \tag{19}$$

$$P_i = \frac{1}{T} \sum_{t=1}^{T} s'_{i,t}, \tag{20}$$

其中均衡因子 α 是一个超参数，针对 DeepSeek-V3 会被赋予一个极小的值；$\mathbb{1}(\cdot)$ 表示指示函数；T 表示序列中 Token 的数量。序列级均衡损失会尽量使得每个序列的专家负载保持均衡。

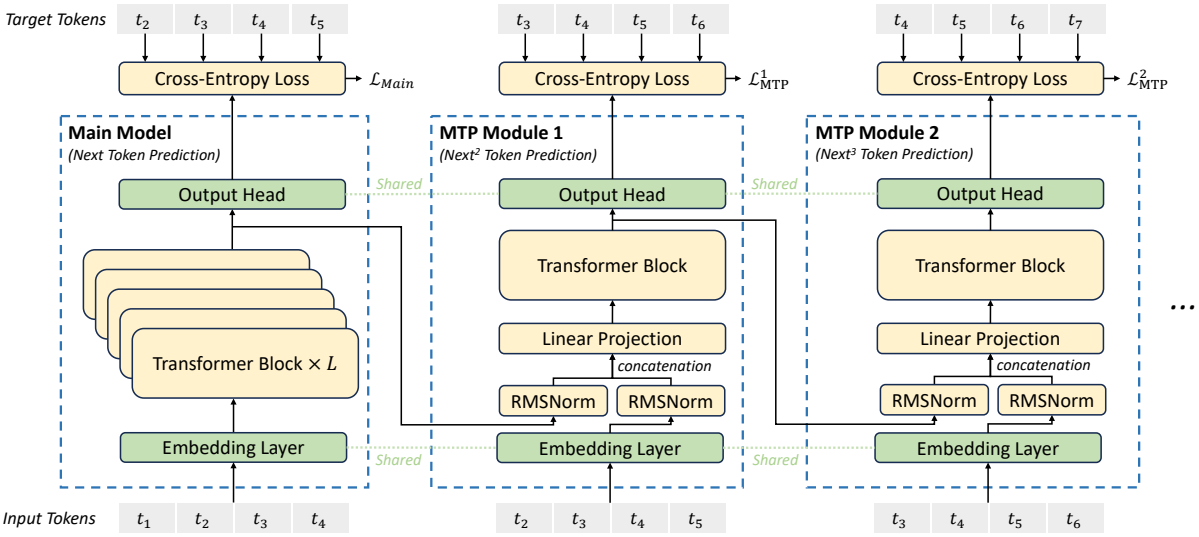


图 3: 多 Token 预测的实现架构图。针对每个深度的每个 Token 的预测，我们都会维护完整的因果链。

限制节点数量的路由。 与 DeepSeek-V2 使用的设备限制路由类似，DeepSeek-V3 也采用了一种受限路由机制，以限制训练过程中的通信成本。简而言之，我们确保每个 Token 最多只会被发送到 M 个节点，这些节点是根据分布在每个节点上的专家的最高 $\frac{K_r}{M}$ 个亲和度分数之和进行选择的。在这个约束下，我们的 MoE 训练框架几乎可以实现完全的计算-通信重叠。

不丢弃任何 Token 由于采用了有效的负载均衡策略，DeepSeek-V3 在整个训练过程中都保持了良好的负载均衡。因此，DeepSeek-V3 在训练过程中不会丢弃任何 Token。此外，我们还实现了特定的部署策略，以确保推理的负载均衡，因此 DeepSeek-V3 在推理过程中也不会丢弃任何 Token。

2.2 多 Token 预测（Multi-Token Prediction）

受到文章《Better & faster large language models via multi-token prediction》的启发，我们为 DeepSeek-V3 研究并设定了多 Token 预测（MTP）训练目标。这样训练目标就将预测范围扩展到每个位置的多个未来 Token。一方面，MTP 训练目标密集化了训练信号，有可能提高数据效率。另一方面，MTP 可能使模型能够提前规划其表示，以更好地预测未来的 Token。图 3 展示了我们对 MTP 的实现。与《Better & faster large language models via multi-token prediction》使用独立输出头来并行预测 D 个额外 Token 不同，我们是顺序的预测了额外的 Token，并在每个预测深度都维护了完整的因果链。在本节中，我们将介绍 MTP 实现的详细信息。

MTP 模块。 具体来说，我们的 MTP 实现使用 D 个顺序模块来预测 D 个额外的 Token。第 k 个 MTP 模块由一个共享的嵌入层 $\mathrm{Emb}(\cdot)$、一个共享的输出头 $\mathrm{OutHead}(\cdot)$、一个 Transformer 块 $\mathrm{TRM}_k(\cdot)$ 和一个投影矩阵 $M_k \in \mathbb{R}^{d \times 2d}$ 组成。对于第 i 个输入的 Token t_i，在第 k 个预测深度，我们首先将第 $(k-1)$ 深度的第 i 个 Token 的表示 $\mathrm{h}_i^{k-1} \in \mathbb{R}^d$ 与第 $(i+k)$ 个 Token 的嵌入 $Emb(t_{i+k}) \in \mathbb{R}^d$ 通过线性投影结合起来：

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

其中 $[\cdot;\cdot]$ 表示拼接。特别地，当 $k = 1$ 时，h_i^{k-1} 指的是主模型给出的表示。注意，对于每个 MTP 模块，它的嵌入层会和主模型共享。结合后的 $\mathrm{h}_i'^k$ 作为第 k 个深度的 Transformer 块的输入，以生成当前深度的输出表示 h_i^k：

$$\mathrm{h}_{1:T-k}^{k} = \mathrm{TRM}_k(\mathrm{h}_{1:T-k}^{\prime k}), \tag{22}$$

其中 T 表示输入序列的长度，$_{i:j}$ 表示切片运算（包括左右边界）。最后，以 h_i^k 作为输入，共享的输出头将计算第 k 个额外预测的 Token 的概率分布 $P_{i+1+k}^k \in \mathbb{R}^V$，其中 V 是词汇表的大小：

$$P_{i+k+1}^k = \mathrm{OutHead}(\mathrm{h}_i^k). \tag{23}$$

输出头 $\mathrm{OutHead}(\cdot)$ 会将表示 线性映射成 logits，随后使用 $\mathrm{Softmax}(\cdot)$ 函数来计算第 k 个额外 Token 的预测概率。此外，对于每个 MTP 模块，其输出头也与主模型共享。我们维护预测的因果链的原则和 EAGLE 维护预测的因果链的原则是一样的，但 EAGLE 的主要目标是推测解码，而我们是利用 MTP 来改善训练。

MTP 训练目标。 对于每个预测深度，我们计算交叉熵损失 $\mathcal{L}_{\mathrm{MTP}}^k$：

$$\mathcal{L}_{\mathrm{MTP}}^k = \mathrm{CrossEntropy}(P_{2+k:T+1}^k, t_{2+k:T+1}) = -\frac{1}{T}\sum_{i=2+k}^{T+1} \log P_i^k[t_i], \tag{24}$$

其中 T 表示输入序列的长度，t_i 表示第 i 个位置的真实 Token，$P_i^k[t_i]$ 表示由第 k 个 MTP 模块给出的 t_i 的相应预测概率。最后，我们计算所有深度的 MTP 损失的平均值，并乘以一个权重因子 λ，以获得整体 MTP 损失 $\mathcal{L}_{\mathrm{MTP}}$，作为 DeepSeek-V3 的额外训练目标：

$$\mathcal{L}_{\mathrm{MTP}} = \frac{\lambda}{D}\sum_{k=1}^{D} \mathcal{L}_{\mathrm{MTP}}^k. \tag{25}$$

推理中的 MTP。 MTP 策略旨在提高主模型的性能，因此在推理过程中，我们可以直接丢弃 MTP 模块，主模型也可以独立且正常地运行。此外，我们还可以将这些 MTP 模块重新用于推测解码，以进一步改进生成的延迟。

3 基础设施

3.1 计算集群

DeepSeek-V3 在一个配备 2048 个 NVIDIA H800 GPU 的集群上进行训练。每个 H800 集群节点包含 8 个通过 NVLink 和 NVSwitch 连接的 GPU。不同节点之间使用 InfiniBand 互连来促进通信。

3.2 训练框架

DeepSeek-V3 的训练得益于 HAI-LLM 框架，这是我们的工程师从零开始打造的高效轻量级训练框架。总体而言，DeepSeek-V3 使用了 16 路流水线并行（Pipeline Parallelism）、跨越 8 个节点的 64 路专家并行（Expert Parallelism）和 ZeRO-1 数据并行（Data Parallelism）。

为了促进 DeepSeek-V3 的高效训练，我们进行了细致的工程优化。首先，我们设计了 DualPipe 算法以实现高效的流水线并行。与现有的流水线并行方法相比，DualPipe 具有更少的流水线气泡。更重要的是，它在前向和反向过程中重叠计算和通信阶段，从而解决了跨节点专家并行带来的高通信开销问题。其次，我们开发了高效的跨节点 All-to-All 通信内核，以充分利用 InfiniBand 和 NVLink 带宽，从而将专用于通信的流式多处理器（SM）节省了一部分出来。最后，我们在训练过程中仔细优化了内存占用，从而使我们能够在不使用昂贵的张量并行（Tensor Parallelism）的情况下训练 DeepSeek-V3。

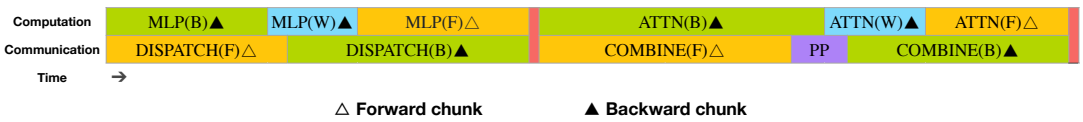


图 4: 针对一对儿独立的前向和反向计算块的重叠策略（Transformer 块的边界未对齐）。橙色表示前向，绿色表示“输入的反向”，蓝色表示“权重的反向”，紫色表示流水线并行通信，红色表示屏障。All-to-All 通信和流水线并行通信都可以完全隐藏。

图 5: 8 个 PP 排名和 20 个微批次在两个方向上的 DualPipe 调度示例。反向方向上的微批次与前向方向上的微批次对称，因此为了简化说明，我们省略了它们的批次 ID。两个由共享黑色边框围住的单元具有相互重叠的计算和通信。

3.2.1 DualPipe 和计算-通信叠加

对于 DeepSeek-V3，跨节点专家并行引入的通信开销导致计算-通信比约为 1:1，这使得效率低下。为了解决这一挑战，我们设计了一种创新的流水线并行算法，称为 DualPipe 算法，它不仅通过有效重叠前向和反向计算-通信阶段来加速模型训练，还减少了流水线气泡的产生。

DualPipe 算法的关键思想是在一对前向和反向块内重叠计算和通信。具体来说，我们将每个块分为四个组件：`attention`、`all-to-all dispatch`、`MLP` 和 `all-to-all combine`。特别地，对于反向块，`attention` 和 `MLP` 进一步拆分为两个部分：输入的 `backward` 和权重的 `backward`，类似于 ZeroBubble。此外，我们还有一个流水线并行通信组件。如图 4 所示，对于一对前向和反向块，我们重新排列这些组件，并手动调整专用于通信与计算的 GPU 流式多处理器的比例。在这种重叠策略中，我们可以确保在执行期间，所有 All-to-All 通信和流水线并行通信可以完全隐藏。由于采用了高效的重叠策略，完整的 DualPipe 算法调度如图 5 所示。它采用双向管道调度，也就是同时从管道的两端输入微批次（micro batch），这样大量通信就可以完全重叠了。这种重叠还确保了，即使模型会进一步扩展，只要我们保持恒定的计算-通信比，就可以在节点之间使用细粒度的专家，同时将 All-to-All 通信的开销降低到接近于零。

此外，即使在通信开销不高的那些更一般的场景中，DualPipe 算法仍然表现出效率优势。在表 2 中，我们总结了不同的流水线并行方法的流水线气泡和内存使用情况。如表中所示，与 ZB1P 和 1F1B 相比，DualPipe 算法显著减少了流水线气泡，同时仅将峰值激活内存增加了 $\frac{1}{PP}$ 倍。尽管 DualPipe 算法需要维护两份模型参数的副本，但由于我们在训练中使用了较大的专家并行规模，所以这并不会显著增加内存消耗。与 Chimera 相比，DualPipe 算法仅要求管道的阶段数量和微批次数量可被 2 整除就行，而不要求微批次数量可被管道的阶段数量整除。此外，对于 DualPipe 算法，随着微批次数量的增加，流水线气泡和激活内存都不会增加。

3.2.2 跨节点 All-to-All 通信的高效实现

为了确保 DualPipe 算法的计算性能，我们定制了高效的跨节点 All-to-All 通信内核（包括调度和合并），以节省用于通信的流式多处理器（SM）的数量。这些内核的实现与我们的 MoE 门控算法和集群的网络拓扑共同设计。具体来说，在我们的集群中，跨节点的 GPU 通过 InfiniBand 完全互连，而节点内的通信则通过 NVLink 处理。NVLink 的带宽为 160 GB/s，约为 InfiniBand（50 GB/s）的 3.2 倍。为了有效利用 InfiniBand 和 NVLink 的不同带宽，我们限制每个 Token 最多调度到 4 个节点，从而减少 InfiniBand

Method	Bubble	Parameter	Activation
1F1B	$(PP-1)(F+B)$	1×	PP
ZB1P	$(PP-1)(F+B-2W)$	1×	PP
DualPipe (Ours)	$(\frac{PP}{2}-1)(F\&B+B-3W)$	2×	$PP+1$

表 2: 不同流水线并行方法下的流水线气泡和内存使用情况比较。F 表示前向块的执行时间，B 表示完整反向块的执行时间，W 表示“反向权重”块的执行时间，$F\&B$ 表示两个相互重叠的前向和反向块的执行时间。

流量。对于每个 Token，当其路由决策做出后，它会首先通过 InfiniBand 传输到目标节点上具有相同节点索引的 GPU。一旦到达目标节点，我们将努力确保它能够即时通过 NVLink 转发到承载其目标专家的特定 GPU，而不被后续到达的 Token 阻塞。通过这种方式，InfiniBand 和 NVLink 的通信得以完全重叠，每个 Token 可以有效地选择每个节点中的平均 3.2 个专家，而不会产生 NVLink 的额外开销。这意味着，尽管 DeepSeek-V3 实际上只选择 8 个路由专家，但它可以将这个数字扩展到最多 13 个专家（4 个节点 × 3.2 个专家/节点），同时维持相同的通信成本。总体而言，在这种通信策略下，仅需 20 个流式多处理器就足以充分利用 InfiniBand 和 NVLink 的带宽。

具体来说，我们采用了 Warp 专门化技术，将 20 个流式多处理器（SM）划分为 10 个通信通道。在调度过程中，(1) InfiniBand 发送、(2) InfiniBand 到 NVLink 的转发和 (3) NVLink 接收分别由相应的 Warp 处理。分配给每个通信任务的 Warp 数量根据所有流式多处理器的实际工作负载动态调整。

类似地，在合并过程中，(1) NVLink 发送、(2) NVLink 到 InfiniBand 的转发和累加，以及 (3) InfiniBand 接收和累加也由动态调整的 Warp 处理。此外，调度和合并内核与计算流重叠，因此我们还考虑它们对其他流式多处理器计算内核的影响。具体而言，我们采用定制的 PTX（并行线程执行）指令，并自动调优通信块的大小，这显著减少了 L2 缓存的使用和对其他流式多处理器的干扰。

3.2.3 在最小化开销的情况下，极致的节省内存

为了减少训练过程中的内存占用，我们采用了以下技术。

重新计算 RMSNorm 和 MLA 的上投影（up-projection）。 我们在反向传播过程中重新计算了所有 RMSNorm 运算和 MLA 上投影，从而消除持续存储其输出激活的需要。通过这种策略，虽然会有少量的开销，但显著减少了存储激活所需的内存。

CPU 中的指数移动平均（Exponential Moving Average）。 在训练过程中，我们保留模型参数的指数移动平均（EMA），以便在学习率衰减后对模型性能进行早期估计。EMA 参数存储在 CPU 内存中，并在每个训练步骤后异步更新。这种方法使我们能够维护 EMA 参数的同时，不会增加额外的内存或时间开销。

在多 token 预测中，共享嵌入和输出头。 通过 DualPipe 算法策略，我们将模型的最浅层（包括嵌入层）和最深层（包括输出头）部署在相同排位的流水线并行上。这种安排使得 MTP 模块和主模型之间共享的嵌入和输出头可以在物理上共享参数和梯度。这种物理共享机制进一步提高了内存效率。

3.3 FP8 低精度训练

受到最近低精度训练的研究进展的启发，我们提出了一种细粒度的混合精度框架，也就是利用 FP8 数据格式来训练 DeepSeek-V3 模型。尽管低精度训练具有巨大潜力，但在激活值、权重和梯度中常常存在异常值，这也限制了低精度训练的应用。尽管低精度技术在推理量化方面已取得显著进展，但关于低

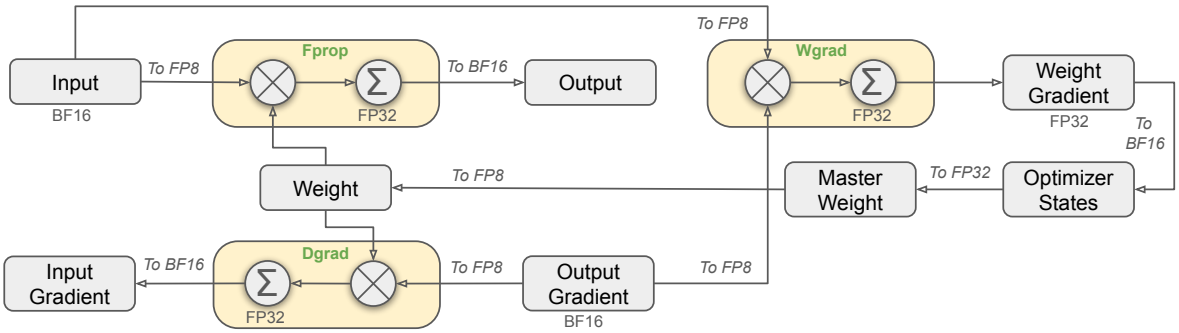


图 6: FP8 数据格式的混合精度框架概览。为了清晰易懂，我们只画出了线性运算符。

精度技术在大模型预训练中的成功应用的研究仍然相对较少。为了解决这个挑战并有效扩展 FP8 格式的动态范围，我们提出了一种细粒度的量化策略：基于 $1 \times N_c$ 元素的切片分组或基于 $N_c \times N_c$ 元素的块分组。通过提高精度并累加的方式，我们有效缓解了与去量化（dequantization）相关的开销，这对实现准确的 FP8 通用矩阵乘法（GEMM）至关重要。此外，为了进一步减少 MoE 训练中的内存和通信开销，我们以 FP8 格式缓存和调度激活值，同时以 BF16 格式存储低精度优化器状态。我们在与 DeepSeek-V2-Lite 和 DeepSeek-V2 类似的两个模型规模上验证了所提出的 FP8 混合精度框架，并训练了大约 1 万亿个 tokens（详细信息请参见附录 A.1）。值得注意的是，与 BF16 基准相比，我们的 FP8 训练模型的相对损失误差始终保持在 0.25% 以下，处于训练随机性的可接受范围内。

3.3.1 混合精度框架

基于广泛采用的低精度训练技术，我们提出了一种用于 FP8 训练的混合精度框架。在该框架中，大多数计算密集型运算使用 FP8 进行处理，而一些关键运算则通过策略性地保持原始数据格式，来平衡训练效率和数值稳定性。整体框架如图 6所示。

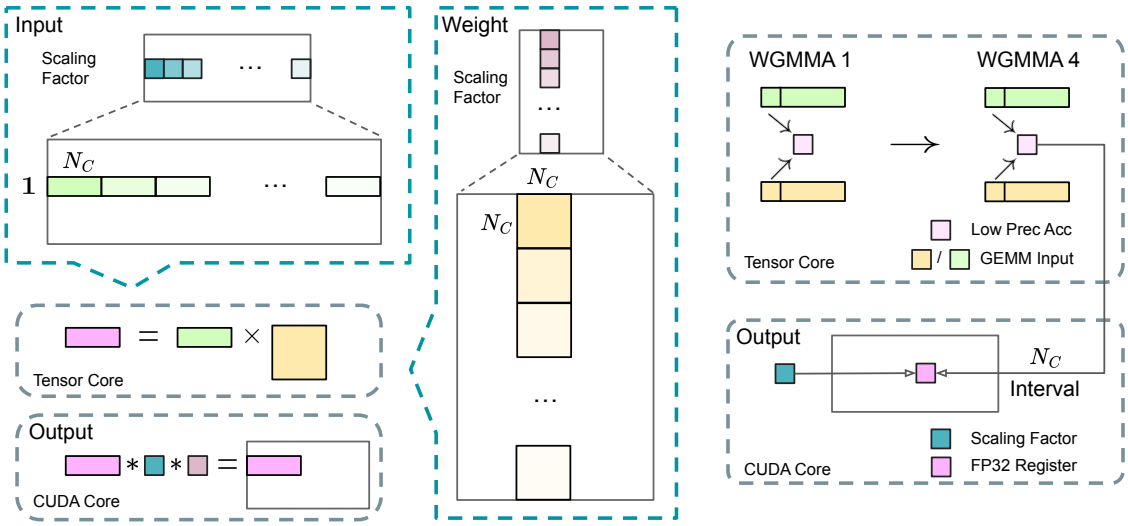
首先，为了加速模型训练，大多数核心计算核（即 GEMM 操作）采用 FP8 精度实现。这些 GEMM 运算接受 FP8 张量作为输入，并生成 BF16 或 FP32 格式的输出。如图 6所示，与线性运算相关的三个 GEMM 运算——`Fprop`（前向传播）、`Dgrad`（激活反向传播）和 `Wgrad`（权重反向传播）——都在 FP8 中执行。与原始的 BF16 方法相比，这种设计理论上使得计算速度翻倍。此外，FP8 的 `Wgrad` GEMM 允许激活值以 FP8 格式存储，并在反向传播中使用。这大大减少了内存消耗。

尽管 FP8 格式在效率上具有优势，但某些运算对低精度计算很敏感，所以要求更高的精度。此外，一些低成本运算也可以使用更高的精度，而对整体训练成本的影响几乎可以忽略不计。因此，在经过仔细调查后，我们决定对于以下组件保持原始精度（例如 BF16 或 FP32）：嵌入模块、输出头、MoE 门控模块、归一化运算符和注意力运算符。针对性地保留高精度可以确保 DeepSeek-V3 训练过程中的稳定性。为了进一步确保数值稳定性，我们使用更高的精度存储主权重、权重梯度和优化器状态。尽管这些高精度组件会引入一些内存开销，但通过在我们的分布式训练系统中实现的高效分片技术，可以最小化它们的影响。

3.3.2 量化和乘法带来的精度提升

基于我们的 FP8 混合精度框架，我们引入了几种策略来提高低精度训练的准确性，重点优化量化方法和乘法过程。

细粒度的量化。 在低精度训练框架中，由于 FP8 格式的动态范围有限，且其指数位数较少，溢出和下溢是常见的挑战。标准的做法是，输入分布通过将输入张量的最大绝对值缩放到 FP8 可表示的最大值，来



(a) Fine-grained quantization (b) Increasing accumulation precision

图 7: (a) 我们提出了一种细粒度量化方法，以减轻由异常特征值引起的量化误差；为了简单说明，仅展示了 `Fprop`。(b) 配合我们的量化策略，当矩阵乘法累加运算（MMA）达到间隔 $N_C = 128$ 时，就将中间结果提升至 CUDA 核进行运算，从而提升了 FP8 GEMM 的精度，实现高精度的累加。

对齐到 FP8 格式的表示范围。这种方法使得低精度训练对激活值的异常值非常敏感，而异常值可能会严重降低量化精度。为了解决这个问题，我们提出了一种细粒度的量化方法，在更细的层面上进行缩放。如图 7（a）所示，(1) 对于激活值，我们按 `1x128` 的切片方式对元素进行分组和缩放（即每个 Token 每 128 个通道）；(2) 对于权重，我们按 `128x128` 一块的方式对元素进行分组和缩放（即每 128 个输入通道对 128 个输出通道）。这种方法确保量化过程能够根据较小的元素组调整缩放，来更好地适应异常值。在附录 A.2中，我们进一步讨论了当我们按与权重量化相同的方式对激活值进行块级分组和缩放时，可能出现的训练不稳定性。

我们方法中的一个关键修改是，顺着 GEMM 运算内部维度，引入了每组缩放因子。这一功能在标准的 FP8 GEMM 中并不直接支持，但结合我们的精确 FP32 累加策略，可以高效地实现。

值得注意的是，我们的细粒度量化策略与微缩放格式的理念高度一致，而 NVIDIA 下一代 GPU（Blackwell 系列）的 Tensor Cores 已宣布支持具有更小量化粒度的微缩放格式。这里希望我们的设计能为未来的工作提供参考，以跟上最新 GPU 架构的发展。

提高累加精度。 低精度 GEMM 运算常常遭遇下溢问题，它们的精度在很大程度上依赖于高精度的累加，而这通常是在 FP32 精度下进行的。然而，我们观察到，在 NVIDIA H800 GPU 上，FP8 GEMM 的累加精度仅能保留大约 14 位，这明显低于 FP32 的累加精度。当内部维度 `K` 较大时，这个问题会变得更加明显，因为在大模型训练的典型场景中，批大小和模型宽度都会增大。以 `K` = 4096 的两个随机矩阵的 GEMM 运算为例，在初步测试中，Tensor Cores 中有限的累加精度导致最大相对误差接近 2%。尽管存在这些问题，有限的累加精度仍然是一些 FP8 框架中的默认选项，这严重制约了训练的准确性。

为了解决这个问题，我们采用了提升到 CUDA 核进行运算的方法以获得更高的精度。这一过程如图 7（b）所示。具体来说，在张量核（Tensor Cores）上执行 MMA（矩阵乘法累加）时，使用有限的位宽累加中间结果。一旦达到 N_C 的间隔，将这些中间结果复制到 CUDA 核上的 FP32 寄存器中，并在那里执行全精度的 FP32 累加。如前所述，我们的细粒度量化在内部维度 `K` 上应用每组缩放因子。这些缩放因子可以在 CUDA 核上高效地与去量化过程一起进行乘法运算，且几乎没有额外的计算开销。

值得注意的是，这一修改减少了单个 warpgroup 的 WGMMA（warp 组级别矩阵乘法累加）指令发射

率。然而，在 H800 架构上，通常会有两个 WGMMA 并行存在：一个 warpgroup 执行提升操作，另一个则执行 MMA 运算。这种设计使得两个操作能够重叠执行，从而保持了张量核的高利用率。根据我们的实验，将 N_C = 128 元素（相当于 4 个 WGMMAs）设置为最小的累加间隔，可以显著提高精度，同时不会引入过多的开销。

尾数优于指数。 先前 NVIDIA 某项工作采用了混合 FP8 格式，也就是在 `Fprop` 中使用 `E4M3`（4 位指数和 3 位尾数），在 `Dgrad` 和 `Wgrad` 中使用 `E5M2`（5 位指数和 2 位尾数）。而我们选择了不同的策略，我们在所有张量中都采用 `E4M3` 格式，以获得更高的精度。我们认为这种方法可行的原因在于我们的细粒度量化策略，即切片级别和块级别的缩放。通过对较小的元素组进行运算，我们的方法有效地在这些分组元素之间共享指数位，从而减轻了有限动态范围的影响。

在线量化。 延迟量化被应用于张量级量化框架中，该框架维护了先前迭代中最大绝对值的历史记录，以推测当前值。为了确保准确的缩放因子并简化框架，我们对每个 `1x128` 激活切片或 `128x128` 权重块在线计算最大绝对值。基于该值，我们推导出缩放因子，然后在线将激活值或权重量化为 FP8 格式。

3.3.3 低精度数据的存储和通信

结合我们的 FP8 训练框架，通过将缓存的激活值和优化器状态压缩为低精度格式，进一步减少了内存消耗和通信开销。

低精度优化器状态。 我们采用 BF16 数据格式而不是 FP32 来跟踪 AdamW 优化器中的一阶和二阶矩，且没有引起明显的性能下降。然而，主权重（由优化器存储）和梯度（用于批量大小累加）仍然使用 FP32，以确保整个训练过程中的数值稳定性。

低精度激活值。 如图 6所示，`Wgrad` 运算使用 FP8 执行。为了减少内存消耗，在线性运算的反向传播中，将激活值缓存为 FP8 格式是一个自然的选择。然而，针对一些运算，我们在低成本高精度训练中进行了特别考虑：

(1) 注意力运算符后面的线性**输入**。这些激活值也用于注意力运算符的反向传播，因此对精度比较敏感。我们为这些激活值采用了定制的 `E5M6` 数据格式。此外，这些激活值将在反向传播中从 `1x128` 量化切片转换为 `128x1` 切片。为了避免引入额外的量化误差，所有的缩放因子都采用 2 的整数次幂进行四舍五入缩放。

(2) MoE 中的 SwiGLU 运算符的输入。为了进一步降低内存开销，我们缓存 SwiGLU 运算符的输入，并在反向传播中重新计算其输出。这些激活值也使用我们的细粒度量化方法以 FP8 格式存储，在内存效率和计算精度之间取得平衡。

低精度数据的通信。 通信带宽是 MoE 模型训练中的一个关键瓶颈。为了缓解这一挑战，我们将 MoE 上投影前的激活值量化为 FP8，然后应用 `dispatch` 组件，这与 MoE 上投影中的 FP8 `Fprop` 兼容。与注意力运算符后面的线性输入类似，这些激活值的缩放因子是 2 的整数次幂。类似的策略也应用于 MoE 下投影前的激活梯度。对于前向和反向的 `combine` 组件，我们将它们保留为 BF16，以确保训练过程中关键部分的训练精度。

3.4 推理和部署

我们在 H800 集群上部署了 DeepSeek-V3，其中每个节点内的 GPU 通过 NVLink 互连，集群内所有 GPU 通过 InfiniBand 实现全互连。为了同时确保在线服务的服务级目标（SLO）和高吞吐量，我们采用了以下部署策略，将预填充和解码阶段分开。

3.4.1 预填充（Prefilling）

预填充阶段的最小部署单元由 4 个节点和 32 个 GPU 组成。attention 部分采用 4 路张量并行（TP4）与序列并行（SP）相结合，再加上 8 路数据并行（DP8）。其较小的张量并行大小为 4，限制了张量并行的通信开销。对于 MoE 部分，我们使用 32 路专家并行（EP32），确保每个专家处理足够大的批次大小，从而提高计算效率。对于 MoE 的 All-to-All 通信，我们采用与训练相同的方法：首先通过 InfiniBand 在节点间传输 Token，然后通过 NVLink 在节点内的 GPU 之间转发。特别地，对于浅层中的稠密 MLP，我们使用 1 路张量并行，以节省张量并行的通信开销。

为了在 MoE 部分实现不同专家之间的负载均衡，我们需要确保每个 GPU 处理大致相同数量的 Token。为此，我们引入了一种冗余专家的部署策略，复制高负载专家并进行冗余部署。高负载专家是根据在线部署过程中收集的统计信息检测到的，并定期进行调整（例如，每 10 分钟调整一次）。在确定冗余专家集合后，我们根据观察到的负载，仔细地在一个节点内重新安排专家，尽量在不增加跨节点 All-to-All 通信开销的情况下均衡各 GPU 的负载。对于 DeepSeek-V3 的部署，我们在预填充阶段设置了 32 个冗余专家。对于每个 GPU，除了原有的 8 个专家外，还将托管一个额外的冗余专家。

此外，在预填充阶段，为了提高吞吐量并隐藏 All-to-All 和张量并行通信的开销，我们同时处理两个具有相似计算负载的微批次，将一个微批次的 attention 和 MoE 与另一个微批次的 dispatch 和 combine 重叠执行。

最后，我们正在探索一种动态冗余策略，针对专家，每个 GPU 托管更多的专家（例如，16 个专家），但每次推理步骤中仅激活 9 个专家。在每层的 All-to-All 操作开始之前，我们即时计算全局最优的路由方案。鉴于预填充阶段涉及的大量计算，计算这个路由方案的开销几乎可以忽略不计。

3.4.2 解码

在解码过程中，我们将共享专家视为路由专家。从这个角度来看，每个 Token 在路由过程中将选择 9 个专家，其中共享专家被视为高负载专家，将始终被选择。解码阶段的最小部署单元由 40 个节点和 320 个 GPU 组成。attention 部分采用 4 路张量并行（TP4）与序列并行（SP）相结合，再配合 80 路数据并行，而 MoE 部分使用 320 路专家并行。对于 MoE 部分，每个 GPU 只托管一个专家，64 个 GPU 负责托管冗余专家和共享专家。dispatch 和 combine 部分的 All-to-All 通信通过 InfiniBand 进行直接点对点传输，以实现低延迟。此外，我们利用 IBGDA 技术进一步最小化延迟并提高通信效率。

与预填充阶段类似，我们根据在线服务中的专家负载统计，定期确定冗余专家集合。然而，由于每个 GPU 只托管一个专家，我们不需要重新安排专家。我们还在探索解码阶段的动态冗余策略，但这需要更仔细地优化计算全局最优路由方案的算法，并与 dispatch 核进行融合以减少开销。

此外，为了提高吞吐量并隐藏 All-to-All 通信的开销，我们还在探索在解码阶段同时处理两个具有相似计算负载的微批次。与预填充阶段不同，attention 在解码阶段消耗了较大部分的时间。因此，我们将一个微批次的 attention 与另一个微批次的 dispatch+MoE+combine 重叠执行。在解码阶段，每个专家的批次大小相对较小（通常在 256 个 Token 以内），瓶颈是内存访问而非计算。由于 MoE 部分只需要加载一个专家的参数，因此内存访问开销最小，因此使用较少的流式多处理器不会显著影响整体性能。因此，为了避免影响 attention 部分的计算速度，我们可以将少部分流式多处理器分配给 dispatch+MoE+combine 部分。

3.5 对硬件设计的建议

基于我们在实现 All-to-All 通信和 FP8 训练策略时的经验，我们想对 AI 硬件提供商提出以下设计芯片的建议：

3.5.1 通信硬件

在 DeepSeek-V3 中，我们通过在计算过程中隐藏通信延迟来实现计算与通信的重叠。这显著减少了与串行计算和通信相比对通信带宽的依赖。然而，目前的通信实现依赖于昂贵的流式多处理器（例如，我们为此在 H800 GPU 中分配了 132 个流式多处理器中的 20 个），这将限制计算吞吐量。此外，使用流式多处理器进行通信会导致显著的低效，因为张量核完全没有得到充分利用。

目前，在 All-to-All 通信中，流式多处理器主要执行以下任务：

- **在 InfiniBand 和 NVLink 域之间转发数据**，同时汇聚从单个 GPU 发往同一节点内多个 GPU 的 InfiniBand 流量。
- **在 RDMA 缓冲区（已注册的 GPU 内存区域）和输入/输出缓冲区之间传输数据**。
- **执行 `reduce` 运算**，用于 `all-to-all combine`。
- **在通过 InfiniBand 和 NVLink 域向多个专家传输分块数据时，管理细粒度的内存布局**。

我们期望未来的硬件供应商能够开发出将这些通信任务从宝贵的计算单元流式多处理器中卸载的硬件，作为 GPU 协处理器或类似 NVIDIA SHARP 的网络协处理器。此外，为了降低应用程序编程的复杂性，我们希望该硬件能够从计算单元的角度统一 InfiniBand（扩展）和 NVLink（扩展）网络。通过这个统一接口，计算单元可以轻松地通过提交基于简单原语的通信请求，完成诸如 `read`、`write`、`multicast` 和 `reduce` 等操作，跨越整个 InfiniBand-NVLink-统一域。

3.5.2 计算硬件

张量核中更高的 FP8 GEMM 累加精度。 在 NVIDIA Hopper 架构的张量核实现中，FP8 GEMM（通用矩阵乘法）采用定点累加，在加法之前通过基于最大指数的右移对尾数积进行对齐。我们的实验表明，在符号填充右移后，它仅使用每个尾数积的最高 14 位，并截断超过此范围的位。然而，例如，为了从 32 次 FP8×FP8 相乘的累加中获得精确的 FP32 结果，至少需要 34 位精度。因此，我们建议未来的芯片设计在张量核中提高累加精度，以支持全精度累加，或根据训练和推理算法的精度需求选择适当的累加位宽。这种方法可以确保误差保持在可接受的范围内，同时保持计算效率。

支持 Tile 级和块级量化。 目前的 GPU 仅支持张量级别的量化，缺乏对像我们这种 Tile 级别和块级别量化的原生支持。在当前的实现中，当达到 N_C 间隔时，部分结果会从张量核复制到 CUDA 核，乘以缩放因子后，添加到 CUDA 核上的 FP32 寄存器中。尽管结合我们精确的 FP32 累加策略，去量化开销已经得到了显著缓解，但频繁的张量核和 CUDA 核之间的数据移动仍然限制了计算效率。因此，我们建议未来的芯片支持细粒度量化，通过使张量核能够接收缩放因子并实现带组缩放的 MMA（矩阵乘法累加）。这样，整个部分和累加去量化的过程可以直接在张量核内部完成，直到最终结果产生，从而避免频繁的数据移动。

支持在线量化。 尽管我们的研究已证明在线量化的有效性，当前的实现仍难以有效支持在线量化。在现有过程中，我们需要从 HBM（高带宽内存）中读取 128 个 BF16 激活值（上一步计算的输出）进行量化，然后将量化后的 FP8 值写回到 HBM，再次读取用于 MMA。为了解决这一低效问题，我们建议未来的芯片将 FP8 类型转换和 TMA（张量内存加速器）访问集成为一个融合操作，这样量化过程可以在激活值从全局内存传输到共享内存的过程中完成，从而避免频繁的内存读写操作。我们还建议支持一个 Warp

级的类型转换指令，以加速运算，并进一步促进层归一化和 FP8 类型转换的更好融合。另外，也可以采用近内存计算的方法，将计算逻辑放置在靠近 HBM 的位置。在这种情况下，BF16 元素可以在从 HBM 读取到 GPU 时直接转换为 FP8，减少大约 50% 的芯片外内存访问。

支持转置 GEMM 运算。 当前的架构使得将矩阵转置与 GEMM 运算融合变得繁琐。在我们的工作流程中，前向传播过程中，激活值被量化为 `1x128` 的 FP8 Tile 并存储。在反向传播过程中，矩阵需要被读取、去量化、转置、重新量化为 `128x1` Tile，并存储在 HBM 中。为了减少内存操作，我们建议未来的芯片在执行 MMA 运算之前，允许从共享内存中直接读取转置后的矩阵，适用于训练和推理所需的精度。结合 FP8 格式转换与 TMA 访问的融合，这一增强将显著简化量化工作流程。

4 预训练（Pre-Training）

4.1 训练数据的构成

与 DeepSeek-V2 相比，我们通过增强数学和编程样本的比例来优化预训练语料库，同时扩展了多语言覆盖范围，超越了英语和中文。此外，我们的数据处理流水线经过改进，以最小化冗余，同时保持语料库的多样性。受 Ding2024FewerTI 的启发，我们实现了文档打包方法以确保数据完整性，但在训练过程中未采用跨样本注意力掩码。最后，DeepSeek-V3 的训练语料库由 14.8T 高质量和多样化的 Token 组成。

在 DeepSeekCoder-V2 的训练过程中，我们观察到填充中间（Fill-in-Middle，FIM）策略并未损害下一个 Token 预测能力，同时使模型能够基于上下文线索准确预测中间文本。与 DeepSeekCoder-V2 一致，我们在 DeepSeek-V3 的预训练中也采用了 FIM 策略。具体来说，我们使用前缀-后缀-中间（Prefix-Suffix-Middle，PSM）框架来构建数据，如下所示：

$$\texttt{<|fim\_begin|>}f_{\text{pre}}\texttt{<|fim\_hole|>}f_{\text{suf}}\texttt{<|fim\_end|>}f_{\text{middle}}\texttt{<|eos\_token|>}.$$

该结构在文档级别应用，作为预打包过程的一部分。FIM 策略的应用比例为 0.1，符合 PSM 框架。

DeepSeek-V3 的分词器采用字节级 BPE 分词器，并将词汇表扩展为 128K 个 Token。分词器的预分词器和训练数据经过修改，以优化多语言压缩效率。此外，与 DeepSeek-V2 相比，新的预分词器引入了将标点符号和换行符结合的 Token。然而，当模型处理没有终止换行符的多行提示时，这种技巧可能会引入 Token 边界偏差，特别是在少样本评估提示中。为了解决这个问题，我们在训练过程中随机拆分一定比例的这种组合 Token，从而使模型接触到更广泛的特殊情况，并减轻这种偏差。

4.2 超参数

模型超参数。 我们将 Transformer 层数设置为 61，隐藏层维度设置为 7168。所有可学习参数的初始化标准差为 0.006。在 MLA 中，我们将注意力头数 n_h 设置为 128，每个头的维度 d_h 设置为 128。KV 压缩维度 d_c 设置为 512，查询压缩维度 d'_c 设置为 1536。对于解耦查询和解耦键，我们将每个头的维度 d_h^R 设置为 64。我们将除了前三层之外的所有前馈网络替换为 MoE 层。每个 MoE 层包含 1 个共享专家和 256 个路由专家，每个专家的中间隐藏层维度为 2048。在路由专家中，每个 Token 将激活 8 个专家，每个 Token 将确保最多发送到 4 个节点。多 Token 预测深度 D 设置为 1，即除了精确的下一个 Token 外，每个 Token 还会预测一个额外的 Token。与 DeepSeek-V2 类似，DeepSeek-V3 也在压缩后的潜在向量后使用额外的 RMSNorm 层，并在宽度瓶颈处乘以额外的缩放因子。在此配置下，DeepSeek-V3 包含 671B 总参数，其中每个 Token 激活的参数为 37B。

训练超参数。 我们采用 AdamW 优化器，超参数设置为 $\beta_1 = 0.9$，$\beta_2 = 0.95$，以及 weight_decay = 0.1。在预训练期间，我们将最大序列长度设置为 4K，并在 14.8T 个 Token 上进行预训练 DeepSeek-V3。关于学习率调度，我们首先在前 2K 步内将学习率从 0 线性增加到 2.2×10^{-4}。然后，我们保持学习率为 2.2×10^{-4}，直到模型处理了 10T 个训练 Token。接下来，我们将学习率逐渐衰减至 2.2×10^{-5}，在 4.3T 个 Token 内按余弦衰减曲线进行衰减。在最后 500B Token 的训练过程中，我们在前 333B Token 内保持学习率为 2.2×10^{-5} 不变，并在剩余的 167B Token 内切换为 7.3×10^{-6} 的常数学习率。梯度裁剪范数设置为 1.0。我们采用批次大小调度策略，在训练的前 469B Token 中，批次大小逐渐从 3072 增加到 15360，之后在剩余训练中保持 15360。我们利用流水线并行将模型的不同层部署到不同的 GPU 上，对于每一层，路由专家将均匀地部署在 64 个 GPU 上，这些 GPU 属于 8 个节点。关于限制了节点数量的路由，每个 Token 最多将被发送到 4 个节点（即 $M = 4$）。为了实现无辅助损失的负载均衡，我们将偏置更新速度 γ 设置为 0.001，适用于前 14.3T 个 Token，在剩余的 500B Token 中设置为 0.0。对于平衡损失，我们将 α 设置为 0.0001，以避免单个序列内出现极端不均衡。MTP 损失权重 λ 在前 10T Token 中设置为 0.3，在剩余的 4.8T Token 中设置为 0.1。

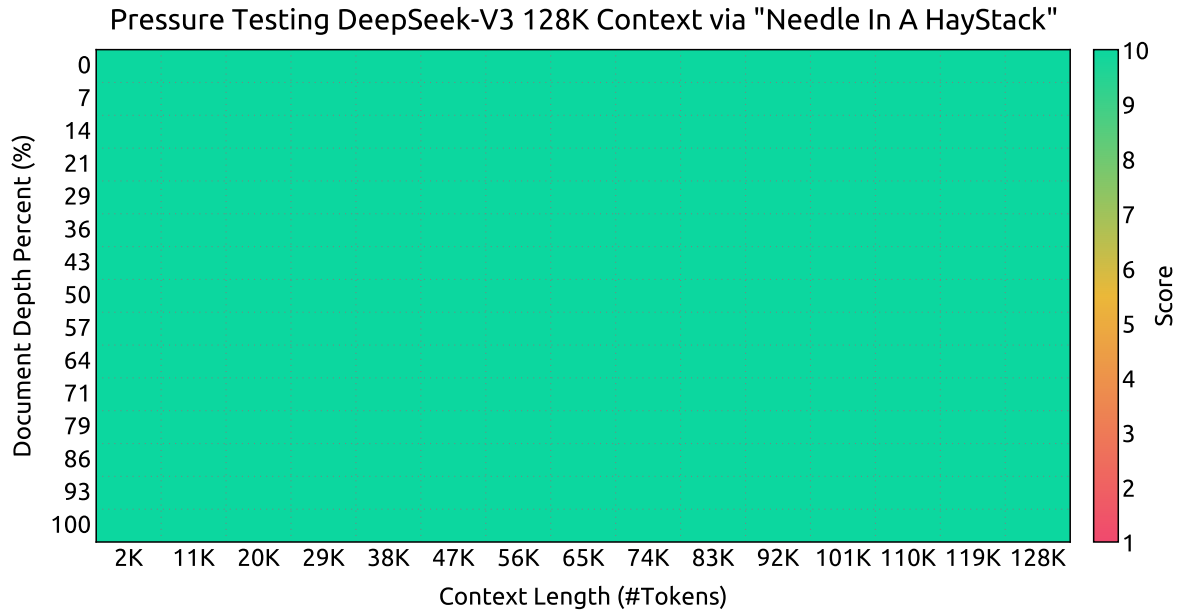


图 8: 在"Needle In A Haystack" (NIAH) 测试集上的评估结果。DeepSeek-V3 在 128K 以内的上下文窗口长度下，都表现很好。

4.3 长上下文扩展

我们采用与 DeepSeek-V2 类似的方法，在 DeepSeek-V3 中实现长上下文能力。在预训练阶段之后，我们使用 YaRN 进行上下文扩展，并执行两个额外的训练阶段，每个阶段包括 1000 步，逐步将上下文窗口从 4K 扩展到 32K，再扩展到 128K。YaRN 的配置与在 DeepSeek-V2 中使用的配置一致，仅应用于解耦共享键 k_t^R。两个阶段的超参数保持相同，分别为：比例因子 $s = 40$，$\alpha = 1$，$\beta = 32$，以及缩放因子 $\sqrt{t} = 0.1 \ln s + 1$。在第一个阶段，序列长度设置为 32K，批次大小为 1920。在第二个阶段，序列长度增加到 128K，批次大小减少到 480。两个阶段的学习率都设置为 7.3×10^{-6}，与预训练阶段的最终学习率相匹配。

通过这两阶段的扩展训练，DeepSeek-V3 能够处理最大长度为 128K 的输入，并保持强大的性能。图 8展示了经过监督微调后的 DeepSeek-V3 在"Needle In A Haystack"（NIAH）测试中取得了显著的表现，且在上下文窗口长度达到 128K 时，展现了持续的健壮性。

4.4 评估

4.4.1 评估基准

DeepSeek-V3 的基础模型是在一个多语言语料库上进行预训练的，其中英语和中文占据了大多数。因此，我们对其在一系列基准测试中的表现进行了评估，这些基准测试主要是英语和中文，以及一个多语言基准。我们的评估基于集成在我们的 HAI-LLM 框架中的内部评估框架。基准被分类并列出如下，其中：带下划线 的基准为中文基准，带(双下划线) 的基准为多语言基准：

多学科多项选择数据集包括 MMLU、MMLU-Redux、MMLU-Pro、MMMLU、C-Eval 和 CMMLU。

语言理解与推理数据集包括 HellaSwag、PIQA、ARC 和 BigBench Hard (BBH)。

闭卷问答数据集包括 TriviaQA 和 NaturalQuestions。

阅读理解数据集包括 RACE、DROP、C3 和 CMRC。

参考消歧数据集包括 CLUEWSC 和 WinoGrande 。

语言建模数据集包括 Pile 。

中文理解与文化数据集包括 CCPM 。

数学数据集包括 GSM8K、MATH、MGSM 和 CMath 。

代码数据集包括 HumanEval、LiveCodeBench-Base (0801-1101)、MBPP 和 CRUXEval。

标准化考试包括 AGIEval 。请注意，AGIEval 包括英语和中文两个子集。

根据我们之前的工作，我们对以下数据集采用基于困惑度的评估：HellaSwag、PIQA、WinoGrande、RACE-Middle、RACE-High、MMLU、MMLU-Redux、MMLU-Pro、MMMLU、ARC-Easy、ARC-Challenge、C-Eval、CMMLU、C3 和 CCPM；对于 TriviaQA、NaturalQuestions、DROP、MATH、GSM8K、MGSM、HumanEval、MBPP、LiveCodeBench-Base、CRUXEval、BBH、AGIEval、CLUEWSC、CMRC 和 CMath，我们采用基于生成的评估。此外，我们对 Pile-test 进行基于语言建模的评估，并使用每字节比特数（Bits-Per-Byte, BPB）作为指标，以确保在使用不同分词器的模型之间进行公平比较。

4.4.2 评估结果

在表 3中，我们将 DeepSeek-V3 的基础模型与最先进的开源基础模型进行比较，包括 DeepSeek-V2-Base（我们之前发布的版本）、Qwen2.5 72B Base 和 LLaMA-3.1 405B Base。我们使用内部评估框架对所有这些模型进行评估，并确保它们共享相同的评估设置。请注意，由于我们评估框架在过去几个月中的变化，DeepSeek-V2-Base 的性能与我们之前报告的结果略有不同。总体而言，DeepSeek-V3-Base 在各项基准测试中全面超越了 DeepSeek-V2-Base 和 Qwen2.5 72B Base，并在大多数基准测试中超过了 LLaMA-3.1 405B Base，基本上成为了最强的开源模型。

从更详细的角度来看，我们分别将 DeepSeek-V3-Base 与其他开源基础模型进行比较。

1. 相较于 DeepSeek-V2-Base，由于我们模型架构的改进、模型规模和训练 Token 数量的扩大，以及数据质量的提升，DeepSeek-V3-Base 的性能显著提高，这也是意料之中的结果。
2. 与最先进的中文开源模型 Qwen2.5 72B Base 相比，尽管激活参数只有一半，DeepSeek-V3-Base 在英语、多语言、代码和数学基准测试上也展现出显著优势。至于中文基准，除了 CMMLU（一个中文多学科多项选择任务），DeepSeek-V3-Base 的表现也优于 Qwen2.5 72B。
3. 与激活参数多达 11 倍的最大开源模型 LLaMA-3.1 405B Base 相比，DeepSeek-V3-Base 在多语言、代码和数学基准测试上也表现得更好。在英语和中文语言基准上，DeepSeek-V3-Base 的表现具有竞争力或更好，尤其在 BBH、MMLU 系列、DROP、C-Eval、CMMLU 和 CCPM 上表现突出。

由于我们的高效架构和全面的工程优化，DeepSeek-V3 实现了极高的训练效率。在我们的训练框架和基础设施下，训练 DeepSeek-V3 每万亿个 Token 仅需 180K H800 GPU 小时，这比训练 72B 或 405B 这样的稠密模型便宜得多。

	Benchmark (Metric)	# Shots	DeepSeek-V2 Base	Qwen2.5 72B Base	LLaMA-3.1 405B Base	DeepSeek-V3 Base
	Architecture	-	MoE	Dense	Dense	MoE
	# Activated Params	-	21B	72B	405B	37B
	# Total Params	-	236B	72B	405B	671B
English	Pile-test (BPB)	-	0.606	0.638	**0.542**	0.548
	BBH (EM)	3-shot	78.8	79.8	82.9	**87.5**
	MMLU (EM)	5-shot	78.4	85.0	84.4	**87.1**
	MMLU-Redux (EM)	5-shot	75.6	83.2	81.3	**86.2**
	MMLU-Pro (EM)	5-shot	51.4	58.3	52.8	**64.4**
	DROP (F1)	3-shot	80.4	80.6	86.0	**89.0**
	ARC-Easy (EM)	25-shot	97.6	98.4	98.4	**98.9**
	ARC-Challenge (EM)	25-shot	92.2	94.5	**95.3**	**95.3**
	HellaSwag (EM)	10-shot	87.1	84.8	**89.2**	**88.9**
	PIQA (EM)	0-shot	83.9	82.6	**85.9**	84.7
	WinoGrande (EM)	5-shot	**86.3**	82.3	85.2	84.9
	RACE-Middle (EM)	5-shot	73.1	68.1	**74.2**	67.1
	RACE-High (EM)	5-shot	52.6	50.3	**56.8**	51.3
	TriviaQA (EM)	5-shot	80.0	71.9	**82.7**	**82.9**
	NaturalQuestions (EM)	5-shot	38.6	33.2	**41.5**	40.0
	AGIEval (EM)	0-shot	57.5	75.8	60.6	**79.6**
Code	HumanEval (Pass@1)	0-shot	43.3	53.0	54.9	**65.2**
	MBPP (Pass@1)	3-shot	65.0	72.6	68.4	**75.4**
	LiveCodeBench-Base (Pass@1)	3-shot	11.6	12.9	15.5	**19.4**
	CRUXEval-I (EM)	2-shot	52.5	59.1	58.5	**67.3**
	CRUXEval-O (EM)	2-shot	49.8	59.9	59.9	**69.8**
Math	GSM8K (EM)	8-shot	81.6	88.3	83.5	**89.3**
	MATH (EM)	4-shot	43.4	54.4	49.0	**61.6**
	MGSM (EM)	8-shot	63.6	76.2	69.9	**79.8**
	CMath (EM)	3-shot	78.7	84.5	77.3	**90.7**
Chinese	CLUEWSC (EM)	5-shot	82.0	82.5	**83.0**	**82.7**
	C-Eval (EM)	5-shot	81.4	89.2	72.5	**90.1**
	CMMLU (EM)	5-shot	84.0	**89.5**	73.7	88.8
	CMRC (EM)	1-shot	**77.4**	75.8	76.0	76.3
	C3 (EM)	0-shot	77.4	76.7	**79.7**	78.6
	CCPM (EM)	0-shot	**93.0**	88.5	78.6	92.0
Multilingual	MMMLU-non-English (EM)	5-shot	64.0	74.8	73.8	**79.4**

表 3: DeepSeek-V3-Base 与其他代表性的开源基础模型的比较。所有模型在我们的内部框架中进行评估，并共享相同的评估设置。得分差距不超过 0.3 的模型被认为处于同一水平。DeepSeek-V3-Base 在大多数基准测试中表现最佳，特别是在数学和代码任务上。

Benchmark (Metric)	# Shots	Small MoE Baseline	Small MoE w/ MTP	Large MoE Baseline	Large MoE w/ MTP
# Activated Params (Inference)	-	2.4B	2.4B	20.9B	20.9B
# Total Params (Inference)	-	15.7B	15.7B	228.7B	228.7B
# Training Tokens	-	1.33T	1.33T	540B	540B
Pile-test (BPB)	-	**0.729**	**0.729**	0.658	**0.657**
BBH (EM)	3-shot	39.0	**41.4**	70.0	**70.7**
MMLU (EM)	5-shot	50.0	**53.3**	**67.5**	66.6
DROP (F1)	1-shot	39.2	**41.3**	68.5	**70.6**
TriviaQA (EM)	5-shot	56.9	**57.7**	**67.0**	**67.3**
NaturalQuestions (EM)	5-shot	**22.7**	22.3	27.2	**28.5**
HumanEval (Pass@1)	0-shot	20.7	**26.8**	44.5	**53.7**
MBPP (Pass@1)	3-shot	35.8	**36.8**	61.6	**62.2**
GSM8K (EM)	8-shot	25.4	**31.4**	72.3	**74.0**
MATH (EM)	4-shot	10.7	**12.6**	38.6	**39.8**

表 4: MTP 策略的消融结果。MTP 策略持续的增强了模型在大多数评估基准上的性能。

4.5 讨论

4.5.1 多 Token 预测的消融研究

在表 4中，我们展示了 MTP 策略的消融结果。具体来说，我们在两个不同规模的基线模型上验证了 MTP 策略。在小规模上，我们训练了一个包含 15.7B 总参数的基线 MoE 模型，使用了 1.33T 的 Token 数据。在大规模上，我们训练了一个包含 228.7B 总参数的基线 MoE 模型，使用了 540B 的 Token 数据。在这两个模型的基础上，保持训练数据和其他架构不变，我们在其上附加了一个深度为 1 的 MTP 模块，并训练了两个使用 MTP 策略的模型进行比较。请注意，在推理过程中，我们直接丢弃 MTP 模块，因此比较模型的推理成本完全相同。从表中可以观察到，MTP 策略在大多数评估基准上始终增强了模型性能。

4.5.2 无辅助损失均衡策略的消融研究

在表 5中，我们展示了无辅助损失平衡策略的消融结果。我们在两个不同规模的基线模型上验证了这一策略。在小规模上，我们训练了一个包含 15.7B 总参数的基线 MoE 模型，使用了 1.33T 的 Token 数据。在大规模上，我们训练了一个包含 228.7B 总参数的基线 MoE 模型，使用了 578B 的 Token 数据。这两个基线模型纯粹使用辅助损失的负载均衡策略，并使用带有 top-K 亲和度归一化的 Sigmoid 门控函数。它们控制辅助损失强度的超参数与 DeepSeek-V2-Lite 和 DeepSeek-V2 相同。在这两个基线模型的基础上，保持训练数据和其他架构不变，我们去除了所有辅助损失，并引入无辅助损失负载均衡策略进行比较。从表中可以观察到，无辅助损失策略在大多数评估基准上始终实现了更好的模型性能。

4.5.3 批次级负载均衡 vs. 序列级负载均衡

无辅助损失均衡策略与序列级辅助损失之间的关键区别在于它们的均衡范围：批次级与序列级。与序列级辅助损失相比，批次级均衡施加了更灵活的约束，因为它不要求每个序列在领域内保持均衡。这种灵活性使得专家能够更好地专注于不同领域。为了验证这一点，我们记录并分析了在 Pile 测试集上，16B 辅助损失基线模型和 16B 无辅助损失模型在不同领域的专家负载。如图 9所示，我们观察到无辅助损失模型表现出更明显的专家专业化模式，正如预期的那样。

Benchmark (Metric)	# Shots	Small MoE Aux-Loss-Based	Small MoE Aux-Loss-Free	Large MoE Aux-Loss-Based	Large MoE Aux-Loss-Free
# Activated Params	-	2.4B	2.4B	20.9B	20.9B
# Total Params	-	15.7B	15.7B	228.7B	228.7B
# Training Tokens	-	1.33T	1.33T	578B	578B
Pile-test (BPB)	-	0.727	**0.724**	0.656	**0.652**
BBH (EM)	3-shot	37.3	**39.3**	66.7	**67.9**
MMLU (EM)	5-shot	51.0	**51.8**	**68.3**	67.2
DROP (F1)	1-shot	38.1	**39.0**	**67.1**	**67.1**
TriviaQA (EM)	5-shot	**58.3**	**58.5**	66.7	**67.7**
NaturalQuestions (EM)	5-shot	**23.2**	**23.4**	27.1	**28.1**
HumanEval (Pass@1)	0-shot	22.0	**22.6**	40.2	**46.3**
MBPP (Pass@1)	3-shot	**36.6**	35.8	59.2	**61.2**
GSM8K (EM)	8-shot	27.1	**29.6**	70.7	**74.5**
MATH (EM)	4-shot	**10.9**	**11.1**	37.2	**39.6**

表 5: 无辅助损失平衡策略的消融结果。与纯粹基于辅助损失的方法相比，无辅助损失策略在大多数评估基准上始终实现了更好的模型性能。

为了进一步探讨这种灵活性与模型性能优势之间的关联，我们还设计并验证了一种批次级辅助损失，该损失鼓励每个训练批次进行负载均衡，而不是每个序列。实验结果表明，当达到类似的批次级负载均衡时，批次级辅助损失也能实现与无辅助损失方法相似的模型性能。具体来说，在我们的 1B MoE 模型实验中，验证损失为：2.258（使用序列级辅助损失）、2.253（使用无辅助损失方法）和 2.253（使用批次级辅助损失）。我们在 3B MoE 模型上也观察到了类似的结果：使用序列级辅助损失的模型验证损失为 2.085，而使用无辅助损失方法或批次级辅助损失的模型验证损失均为 2.080。

此外，尽管批次级负载均衡方法表现出一致的性能优势，但它们在效率上也面临两个潜在挑战：(1) 某些序列或小批次内的负载不均衡，以及 (2) 推理过程中因领域转移引起的负载不均衡。第一个挑战自然由我们使用大规模专家并行和数据并行的训练框架解决，这保证了每个微批次的较大规模。对于第二个挑战，我们还设计并实现了一个高效的推理框架，采用冗余专家部署，如第 3.4 节所述，以克服该问题。

5 后训练（Post-Training）

5.1 监督微调 (Supervised Fine-Tuning)

我们精心策划了指令调优数据集，包括 150 万个实例，涵盖多个领域，每个领域采用不同的数据创建方法，以满足其特定需求。

推理数据。 对于与推理相关的数据集，包括数学、代码竞赛问题和逻辑谜题，我们通过利用内部的 DeepSeek-R1 模型生成数据。具体来说，虽然 R1 生成的数据表现出较强的准确性，但也存在过度思考、格式不佳和长度过长等问题。我们的目标是平衡 R1 生成的推理数据的高准确性与常规格式化推理数据的清晰性和简洁性。

为了建立我们的方法论，我们首先开发一个专门针对特定领域（如代码、数学或一般推理）的专家模型，使用结合的监督微调（SFT）和强化学习（RL）训练流程。该专家模型作为最终模型的数据生成器。训练过程涉及为每个实例生成两种不同类型的监督微调样本：第一种将问题与其原始响应配对，格式为

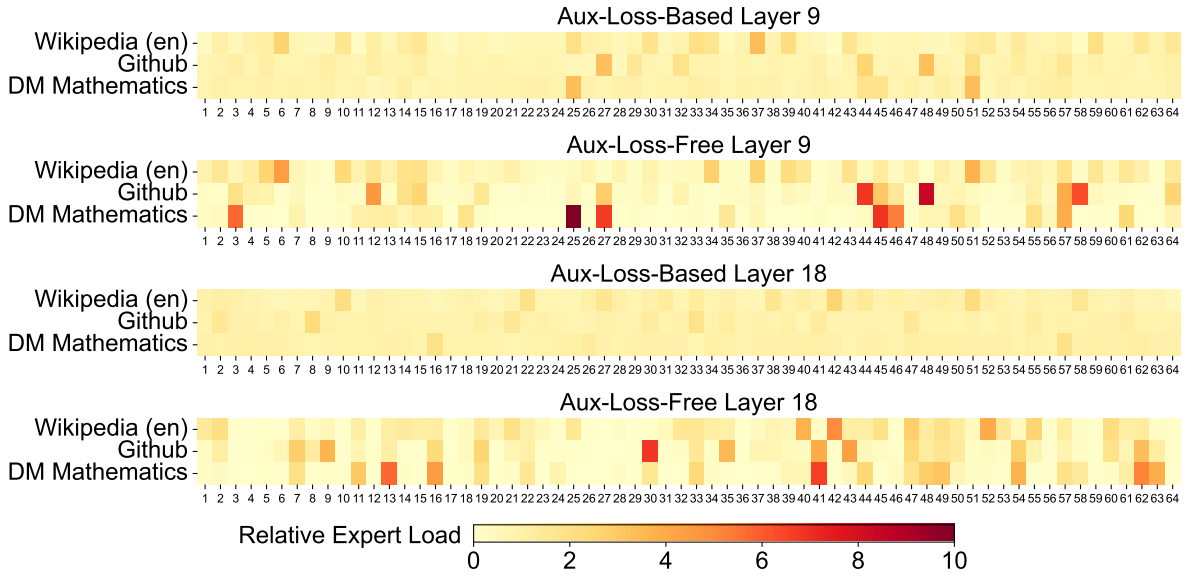


图 9: 无辅助损失模型和基于辅助损失模型在 Pile 测试集三个领域的专家负载。无辅助损失模型表现出比基于辅助损失模型更明显的专家专业化模式。相对专家负载表示实际专家负载与理论平衡专家负载之间的比率。由于篇幅限制，我们仅展示两个层的结果作为示例，所有层的结果详见附录 B。

< 问题, 原始响应 >；第二种则在问题和 R1 响应旁边添加系统提示，格式为 < 系统提示, 问题, R1 响应 >。

系统提示经过精心设计，包含指导模型生成丰富反思和验证机制的响应的指令。在强化学习阶段，模型利用高温采样生成响应，整合来自 R1 生成数据和原始数据的模式，即使在没有明确的系统提示的情况下。在经过数百步的强化学习训练后，中间强化学习模型学会了整合 R1 模式，从而战略性地提升整体性能。

完成强化学习训练阶段后，我们实现了拒绝采样，以策划高质量的监督微调数据供最终模型使用，专家模型被用作数据生成源。这种方法确保最终训练数据保留 DeepSeek-R1 的优点，同时生成简洁有效的响应。

非推理数据。 对于非推理数据，如创意写作、角色扮演和简单问答，我们利用 DeepSeek-V2.5 生成响应，并招募人工注释员验证数据的准确性和正确性。

监督微调设置。 我们对 DeepSeek-V3-Base 进行两轮的微调，使用监督微调数据集，采用余弦衰减学习率调度，从 5×10^{-6} 开始，逐渐降低到 1×10^{-6}。在训练过程中，每个单一序列由多个样本打包而成。然而，我们采用样本掩码策略，以确保这些示例保持隔离且相互不可见。

5.2 强化学习

5.2.1 奖励模型

我们在强化学习过程中采用基于规则的奖励模型（Rule-Based Reward Model，RM）和基于模型的奖励模型。

基于规则的奖励模型。 对于可以通过特定规则验证的问题，我们采用基于规则的奖励系统来确定反馈。例如，某些数学问题具有确定的结果，我们要求模型以指定格式（例如，框内）提供最终答案，从而应用规则来验证正确性。同样，对于 LeetCode 问题，我们可以利用编译器根据测试用例生成反馈。通过尽可能利用基于规则的验证，我们确保了更高的可靠性，因为这种方法对操控或利用具有抵抗力。

基于模型的奖励模型。 对于具有自由形式真实答案的问题，我们依赖奖励模型来判断响应是否与预期的真实答案匹配。相反，对于没有明确真实答案的问题，例如涉及创意写作的问题，奖励模型的任务是根据问题和相应答案作为输入提供反馈。奖励模型从 DeepSeek-V3 监督微调检查点进行训练。为了增强其可靠性，我们构建了偏好数据，不仅提供最终奖励，还包括导致该奖励的思维链。这种方法有助于降低特定任务中奖励操控的风险。

5.2.2 组相对策略优化 (Group Relative Policy Optimization)

与 DeepSeek-V2 类似，我们采用了组相对策略优化（GRPO），该方法放弃了通常与策略模型大小相同的评估模型，而是从组得分中估计基线。具体而言，对于每个问题 q，GRPO 从旧的策略模型 $\pi_{\theta_{old}}$ 中抽样一组输出 $\{o_1, o_2, \cdots, o_G\}$，然后通过最大化以下目标来优化策略模型 π_θ：

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^{G} \sim \pi_{\theta_{old}}(O|q)]$$
$$\frac{1}{G}\sum_{i=1}^{G}\left(\min\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}A_i, \text{clip}\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}, 1-\epsilon, 1+\epsilon\right)A_i\right) - \beta\mathbb{D}_{KL}\left(\pi_\theta||\pi_{ref}\right)\right), \tag{26}$$

$$\mathbb{D}_{KL}\left(\pi_\theta||\pi_{ref}\right) = \frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - \log\frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - 1, \tag{27}$$

其中 ϵ 和 β 是超参数；

π_{ref} 是参考模型；

A_i 是优势，来源于与每组内输出对应的奖励 $r_1, r_2, \ldots, r_G$：

$$A_i = \frac{r_i - \text{mean}(\{r_1, r_2, \cdots, r_G\})}{\text{std}(\{r_1, r_2, \cdots, r_G\})}. \tag{28}$$

我们在强化学习过程中融入了来自不同领域的提示，例如编程、数学、写作、角色扮演和问答。这种方法不仅使模型与人类偏好更加一致，还提高了基准测试中的表现，特别是在可用的监督微调数据有限的情况下。

5.3 评估

5.3.1 评估设置

评估基准。 除了我们用于基础模型测试的基准外，我们还在 IFEval、FRAMES、LongBench v2、GPQA、SimpleQA、C-SimpleQA、SWE-Bench Verified、Aider [2]、LiveCodeBench（2024 年 8 月至 2024 年 11 月的问题）、Codeforces [3]、中国国家高中数学奥林匹克（CNMO 2024）[4]以及美国邀请数学考试 2024（AIME 2024）上进一步评估指令模型。

比较基线。 我们对我们的聊天模型进行了全面评估，与多个强基线进行比较，包括 DeepSeek-V2-0506、DeepSeek-V2.5-0905、Qwen2.5 72B Instruct、LLaMA-3.1 405B Instruct、Claude-Sonnet-3.5-1022 和 GPT-4o-0513。对于 DeepSeek-V2 模型系列，我们选择最具代表性的变体进行比较。对于闭源模型，通过其各自的 API 进行评估。

[2] https://aider.chat

[3] https://codeforces.com

[4] https://www.cms.org.cn/Home/comp/comp/cid/12.html

详细评估配置。 对于包括 MMLU、DROP、GPQA 和 SimpleQA 在内的标准基准，我们采用来自 simple-evals 框架的评估提示[5]。我们在零-shot 设置中使用 Zero-Eval 提示格式进行 MMLU-Redux 的评估。对于其他数据集，我们遵循数据集创建者提供的原始评估协议和默认提示。对于代码和数学基准，HumanEval-Mul 数据集包括 8 种主流编程语言（Python、Java、C++、C#、JavaScript、TypeScript、PHP 和 Bash）。我们使用 CoT 和非 CoT 方法评估模型在 LiveCodeBench 上的表现，数据收集时间为 2024 年 8 月至 2024 年 11 月。Codeforces 数据集的评估使用竞争者的百分比进行衡量。SWE-Bench Verified 使用无代理框架进行评估。我们使用 "diff" 格式评估与 Aider 相关的基准。对于数学评估，AIME 和 CNMO 2024 的评估温度为 0.7，结果在 16 次运行中取平均，而 MATH-500 则采用贪婪解码。我们允许所有模型在每个基准上输出最多 8192 个 Token。

	Benchmark (Metric)	DeepSeek V2-0506	DeepSeek V2.5-0905	Qwen2.5 72B-Inst.	LLaMA-3.1 405B-Inst.	Claude-3.5-Sonnet-1022	GPT-4o 0513	DeepSeek V3
	Architecture	MoE	MoE	Dense	Dense	-	-	MoE
	# Activated Params	21B	21B	72B	405B	-	-	37B
	# Total Params	236B	236B	72B	405B	-	-	671B
English	MMLU (EM)	78.2	80.6	85.3	**88.6**	**88.3**	87.2	**88.5**
	MMLU-Redux (EM)	77.9	80.3	85.6	86.2	**88.9**	88.0	**89.1**
	MMLU-Pro (EM)	58.5	66.2	71.6	73.3	**78.0**	72.6	75.9
	DROP (3-shot F1)	83.0	87.8	76.7	88.7	88.3	83.7	**91.**
	IF-Eval (Prompt Strict)	57.7	80.6	84.1	86.0	**86.5**	84.3	86.1
	GPQA-Diamond (Pass@1)	35.3	41.3	49.0	51.1	**65.0**	49.9	59.1
	SimpleQA (Correct)	9.0	10.2	9.1	17.1	28.4	**38.2**	24.9
	FRAMES (Acc.)	66.9	65.4	69.8	70.0	72.5	**80.5**	73.3
	LongBench v2 (Acc.)	31.6	35.4	39.4	36.1	41.0	48.1	**48.7**
Code	HumanEval-Mul (Pass@1)	69.3	77.4	77.3	77.2	81.7	80.5	**82.6**
	LiveCodeBench (Pass@1-COT)	18.8	29.2	31.1	28.4	36.3	33.4	**40.5**
	LiveCodeBench (Pass@1)	20.3	28.4	28.7	30.1	32.8	34.2	**37.6**
	Codeforces (Percentile)	17.5	35.6	24.8	25.3	20.3	23.6	**51.6**
	SWE Verified (Resolved)	-	22.6	23.8	24.5	**50.8**	38.8	42.0
	Aider-Edit (Acc.)	60.3	71.6	65.4	63.9	**84.2**	72.9	79.7
	Aider-Polyglot (Acc.)	-	18.2	7.6	5.8	45.3	16.0	**49.6**
Math	AIME 2024 (Pass@1)	4.6	16.7	23.3	23.3	16.0	9.3	**39.2**
	MATH-500 (EM)	56.3	74.7	80.0	73.8	78.3	74.6	**90.2**
	CNMO 2024 (Pass@1)	2.8	10.8	15.9	6.8	13.1	10.8	**43.2**
Chinese	CLUEWSC (EM)	89.9	90.4	**91.4**	84.7	85.4	87.9	90.9
	C-Eval (EM)	78.6	79.5	86.1	61.5	76.7	76.0	**86.5**
	C-SimpleQA (Correct)	48.5	54.1	48.4	50.4	51.3	59.3	**64.8**

表 6: DeepSeek-V3 与其他代表性聊天模型的比较。所有模型在限制输出长度为 8K 的配置下进行评估。对于样本少于 1000 的基准测试，使用不同的温度设置进行多次测试，以得出稳健的最终结果。DeepSeek-V3 是表现最佳的开源模型，并且在与前沿的闭源模型相比时也展现了竞争力。

[5] https://github.com/openai/simple-evals

5.3.2 标准评估

表 6 展示了评估结果，表明 DeepSeek-V3 是表现最好的开源模型。此外，它在与前沿闭源模型如 GPT-4o 和 Claude-3.5-Sonnet 的竞争中也表现出色。

英文基准测试 MMLU 是一个广泛认可的基准，旨在评估大型语言模型在多种知识领域和任务中的表现。DeepSeek-V3 展现出具有竞争力的性能，与顶尖模型如 LLaMA-3.1-405B、GPT-4o 和 Claude-Sonnet 3.5 不相上下，同时显著超越 Qwen2.5 72B。此外，DeepSeek-V3 在 MMLU-Pro 这一更具挑战性的教育知识基准测试中表现优异，接近 Claude-Sonnet 3.5。在 MMLU-Redux（一个修正标签的 MMLU 精简版）中，DeepSeek-V3 超越了其他同类模型。此外，在 GPQA-Diamond 这一博士级评估测试平台上，DeepSeek-V3 取得了卓越的成绩，仅次于 Claude 3.5 Sonnet，且显著超越了所有其他竞争对手。

在长文本理解基准测试中，如 DROP、LongBench v2 和 FRAMES，DeepSeek-V3 继续展示其作为顶尖模型的地位。在 DROP 的 3-shot 设置中，它取得了令人印象深刻的 91.6 F1 分数，超越了该类别中的所有其他模型。在 FRAMES 这一需要在超过 10 万个 Token 上下文中进行问答的基准测试中，DeepSeek-V3 紧随 GPT-4o，其表现显著优于其他所有模型。这展示了 DeepSeek-V3 在处理极长文本任务方面的强大能力。

DeepSeek-V3 的长文本处理能力在 LongBench v2 上得到了进一步验证，该数据集在 DeepSeek V3 发布前几周发布。在事实知识基准测试 SimpleQA 中，DeepSeek-V3 落后于 GPT-4o 和 Claude-Sonnet，这主要是由于其设计重点和资源分配。DeepSeek-V3 将更多的训练 Token 分配给学习中文知识，从而在 C-SimpleQA 上表现出色。在遵循指令的基准测试中，DeepSeek-V3 显著超越了其前身 DeepSeek-V2 系列，突显了其在理解和遵循用户定义格式约束方面的改进能力。

代码与数学基准测试 编程是大语言模型面临的一项具有挑战性且实用的任务，涵盖了工程相关的任务，如 SWE-Bench-Verified 和 Aider，以及算法任务，如 HumanEval 和 LiveCodeBench。在工程任务中，DeepSeek-V3 落后于 Claude-Sonnet-3.5-1022，但显著优于开源模型。开源的 DeepSeek-V3 有望促进与编码相关的工程任务的进展。通过提供其强大的能力，DeepSeek-V3 可以推动软件工程和算法开发等领域的创新与改进，使开发者和研究人员能够拓展开源模型在编程任务中的边界。

在算法任务中，DeepSeek-V3 展现出卓越的表现，超越了 HumanEval-Mul 和 LiveCodeBench 等基准测试中的所有基线。这一成功归因于其先进的知识蒸馏技术，有效提升了其在算法任务中的代码生成和问题解决能力。

在数学基准测试中，DeepSeek-V3 展现出卓越的表现，显著超越了基线，并为非 o1-like 模型设定了新的最先进水平。具体而言，在 AIME、MATH-500 和 CNMO 2024 上，DeepSeek-V3 的绝对得分比第二名模型 Qwen2.5 72B 高出约 10%，这一差距在如此具有挑战性的基准测试中是相当显著的。这一卓越能力突显了 DeepSeek-R1 中蒸馏技术的有效性，该技术已被证明对非 o1-like 模型极为有益。

中文基准测试 Qwen 和 DeepSeek 是两个在中文和英文方面具有强大支持的代表性模型系列。在事实基准测试中文 SimpleQA 中，DeepSeek-V3 超越了 Qwen2.5-72B 16.4 分，尽管 Qwen2.5 是在一个更大的语料库上训练的，包含 18T Token，比 DeepSeek-V3 预训练的 14.8T Token 多出 20%。

在 C-Eval（一个代表性的中文教育知识评估基准）和 CLUEWSC（中文 Winograd 语料库挑战）中，DeepSeek-V3 和 Qwen2.5-72B 表现出相似的性能水平，表明这两个模型在应对具有挑战性的中文推理和教育任务方面都经过了良好的优化。

Model	Arena-Hard	AlpacaEval 2.0
DeepSeek-V2.5-0905	76.2	50.5
Qwen2.5-72B-Instruct	81.2	49.1
LLaMA-3.1 405B	69.3	40.5
GPT-4o-0513	80.4	51.1
Claude-Sonnet-3.5-1022	85.2	52.0
DeepSeek-V3	**85.5**	**70.0**

表 7: 英文开放性对话评估。对于 AlpacaEval 2.0，我们采用长度控制胜率作为评估标准。

Model	Chat	Chat-Hard	Safety	Reasoning	Average
GPT-4o-0513	96.6	70.4	86.7	84.9	84.7
GPT-4o-0806	96.1	76.1	88.1	86.6	86.7
GPT-4o-1120	95.8	71.3	86.2	85.2	84.6
Claude-3.5-sonnet-0620	96.4	74.0	81.6	84.7	84.2
Claude-3.5-sonnet-1022	96.4	79.7	91.1	87.6	88.7
DeepSeek-V3	96.9	79.8	87.0	84.3	87.0
DeepSeek-V3 (maj@6)	96.9	82.6	89.5	89.2	89.6

表 8: GPT-4o，Claude-3.5-sonnet 和 DeepSeek-V3 在 RewardBench 上的性能比较。

5.3.3 开放式评估

除了标准基准测试，我们还使用大型语言模型（LLMs）作为评判者，对开放式生成任务进行了评估，结果如表 7 所示。具体而言，我们遵循了 AlpacaEval 2.0 和 Arena-Hard 的原始配置，利用 GPT-4-Turbo-1106 进行成对比较的评判。

在 Arena-Hard 上，DeepSeek-V3 以超过 86% 的胜率击败基线模型 GPT-4-0314，其表现与顶尖模型如 Claude-Sonnet-3.5-1022 不相上下。这突显了 DeepSeek-V3 的强大能力，尤其是在处理复杂提示时，包括编码和调试任务。此外，DeepSeek-V3 成为第一个在 Arena-Hard 基准测试中超过 85% 的开源模型，这一成就显著缩小了开源模型与闭源模型之间的性能差距，为开源模型在挑战性领域的成就设定了新标准。

同样，DeepSeek-V3 在 AlpacaEval 2.0 上表现出色，超越了闭源和开源模型。这证明了它在写作任务和处理简单问答场景方面的卓越能力。值得注意的是，它以 20% 的显著优势超过了 DeepSeek-V2.5-0905，突显了在处理简单任务方面的重大改进，并展示了其进步的有效性。

5.3.4 DeepSeek-V3 作为通用的奖励模型

我们将 DeepSeek-V3 的判断能力与最先进的模型进行比较，即 GPT-4o 和 Claude-3.5。表 8 展示了这些模型在 RewardBench 中的表现。

DeepSeek-V3 的表现与最佳版本的 GPT-4o-0806 和 Claude-3.5-Sonnet-1022 相当，同时超越了其他版本。此外，DeepSeek-V3 的判断能力还可以通过投票技术得到增强。因此，我们使用 DeepSeek-V3 结合投票技术，对开放式问题提供自我反馈，从而提高对齐过程的有效性和鲁棒性。

Model	LiveCodeBench-CoT		MATH-500	
	Pass@1	Length	Pass@1	Length
DeepSeek-V2.5 Baseline	31.1	718	74.6	769
DeepSeek-V2.5 +R1 Distill	37.4	783	83.2	1510

表 9: 从 DeepSeek-R1 蒸馏的贡献。LiveCodeBench 和 MATH-500 的评估设定和表 6中的一致。

5.4 讨论

5.4.1 从 DeepSeek-R1 蒸馏

我们基于 DeepSeek-V2.5 对来自 DeepSeek-R1 的蒸馏贡献进行了消融实验。基线模型是在短思维链（CoT）数据上训练的，而其竞争模型则使用了上述专家检查点生成的数据。

表 9 显示了蒸馏数据的有效性，在 LiveCodeBench 和 MATH-500 基准测试中均显著提高了性能。我们的实验揭示了一个有趣的权衡：蒸馏导致了更好的性能，但也显著增加了平均响应长度。为了在模型准确性和计算效率之间保持平衡，我们为 DeepSeek-V3 在蒸馏中仔细选择了最佳设置。

我们的研究表明，从推理模型中进行知识蒸馏是后训练优化的一个有前景的方向。虽然我们目前的工作集中在数学和编程领域的数据蒸馏上，但这种方法在各种任务领域中显示出更广泛应用的潜力。在这些特定领域中展示的有效性表明，长思维链蒸馏可能对增强其他需要复杂推理的认知任务中的模型性能具有重要价值。进一步探索这种方法在不同领域的应用仍然是未来研究的重要方向。

5.4.2 自我奖励（Self-Rewarding）

奖励在强化学习中发挥着关键作用，引导优化过程。在一些通过外部工具验证相对简单的领域，例如某些编程或数学场景，强化学习表现出卓越的效果。然而，在更一般的场景中，通过硬编码构建反馈机制是不可行的。在 DeepSeek-V3 的开发过程中，对于这些更广泛的背景，我们采用了宪法 AI（Constitutional AI）方法，利用 DeepSeek-V3 自身的投票评估结果作为反馈来源。这种方法产生了显著的对齐效果，显著提升了 DeepSeek-V3 在主观评估中的表现。通过整合额外的宪法输入，DeepSeek-V3 可以朝着宪法方向进行优化。

我们相信，这种将补充信息与大语言模型结合作为反馈来源的范式至关重要。大语言模型作为一个多功能处理器，能够将来自不同场景的非结构化信息转化为奖励，最终促进大语言模型的自我改进。除了自我奖励外，我们还致力于发现其他通用且可扩展的奖励方法，以持续提升模型在一般场景中的能力。

5.4.3 多 Token 预测评估

DeepSeek-V3 不仅仅预测下一个单个 Token，而是通过 MTP 技术预测下两个 Token。结合推测解码的框架，可以显著加快模型的解码速度。

一个自然的问题是关于额外预测 Token 的接受率。根据我们的评估，第二个 Token 预测的接受率在不同生成主题中范围在 85% 到 90% 之间，显示出一致的可靠性。这个高接受率使得 DeepSeek-V3 能够实现显著提高的解码速度，达到 1.8 倍的 TPS（每秒 Token 数）。

6 结论，局限性和未来的方向

在本文中，我们介绍了 DeepSeek-V3，这是一个大型 MoE 语言模型，具有 671B 的总参数和 37B 的激活参数，训练数据量达到 14.8T Token。除了 MLA 和 DeepSeekMoE 架构外，它还开创了一种无辅助损失的负载均衡策略，并设定了多 Token 预测训练目标，以实现更强的性能。

由于支持 FP8 训练和精细的工程优化，DeepSeek-V3 的训练具有成本效益。后训练阶段成功地从 DeepSeek-R1 系列模型中提炼了推理能力。全面评估表明，DeepSeek-V3 已成为当前最强的开源模型，并且在性能上与领先的闭源模型如 GPT-4o 和 Claude-3.5-Sonnet 相媲美。

尽管性能强大，但它仍保持了经济的训练成本。完整训练（包括预训练、上下文长度扩展和后训练）仅需 2.788M H800 GPU 小时。

尽管我们认可 DeepSeek-V3 的强大性能和成本效益，但我们也认识到它存在一些局限性，尤其是在部署方面。首先，为了确保高效的推理，DeepSeek-V3 的推荐部署单元相对较大，这可能会对小型团队造成负担。其次，尽管我们对 DeepSeek-V3 的部署策略已经实现了端到端生成速度比 DeepSeek-V2 快两倍以上，但仍存在进一步优化的潜力。幸运的是，随着更先进硬件的发展，这些局限性有望自然得到解决。

DeepSeek 始终坚持开放源代码模型的长期发展路线，旨在稳步接近 AGI（人工通用智能）的最终目标。未来，我们计划在以下方向进行战略性投资研究。

- 我们将持续研究和优化我们的模型架构，旨在进一步提高训练和推理效率，努力实现对无限上下文长度的高效支持。此外，我们还将尝试突破 Transformer 的架构限制，从而推动其建模能力的边界。
- 我们将不断迭代训练数据的数量和质量，并探索纳入额外训练信号来源，旨在推动数据在更广泛维度上的扩展。
- 我们将持续探索和迭代模型的深度思考能力，旨在通过扩展推理的长度和深度来增强模型的智能和解决问题的能力。
- 我们将探索更全面和多维度的模型评估方法，以防止在研究中倾向于优化固定的一组基准，这可能会对模型能力产生误导性的印象，并影响我们的基础评估。

Appendix

A 低精度训练的融合研究

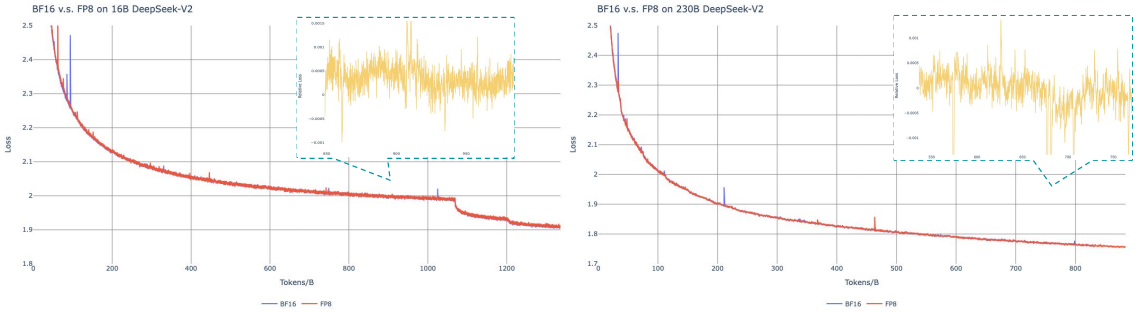


图 10: BF16 和 FP8 训练的损失曲线比较。结果通过指数移动平均（EMA）平滑，系数为 0.9。

A.1 FP8 训练 v.s. BF16 训练

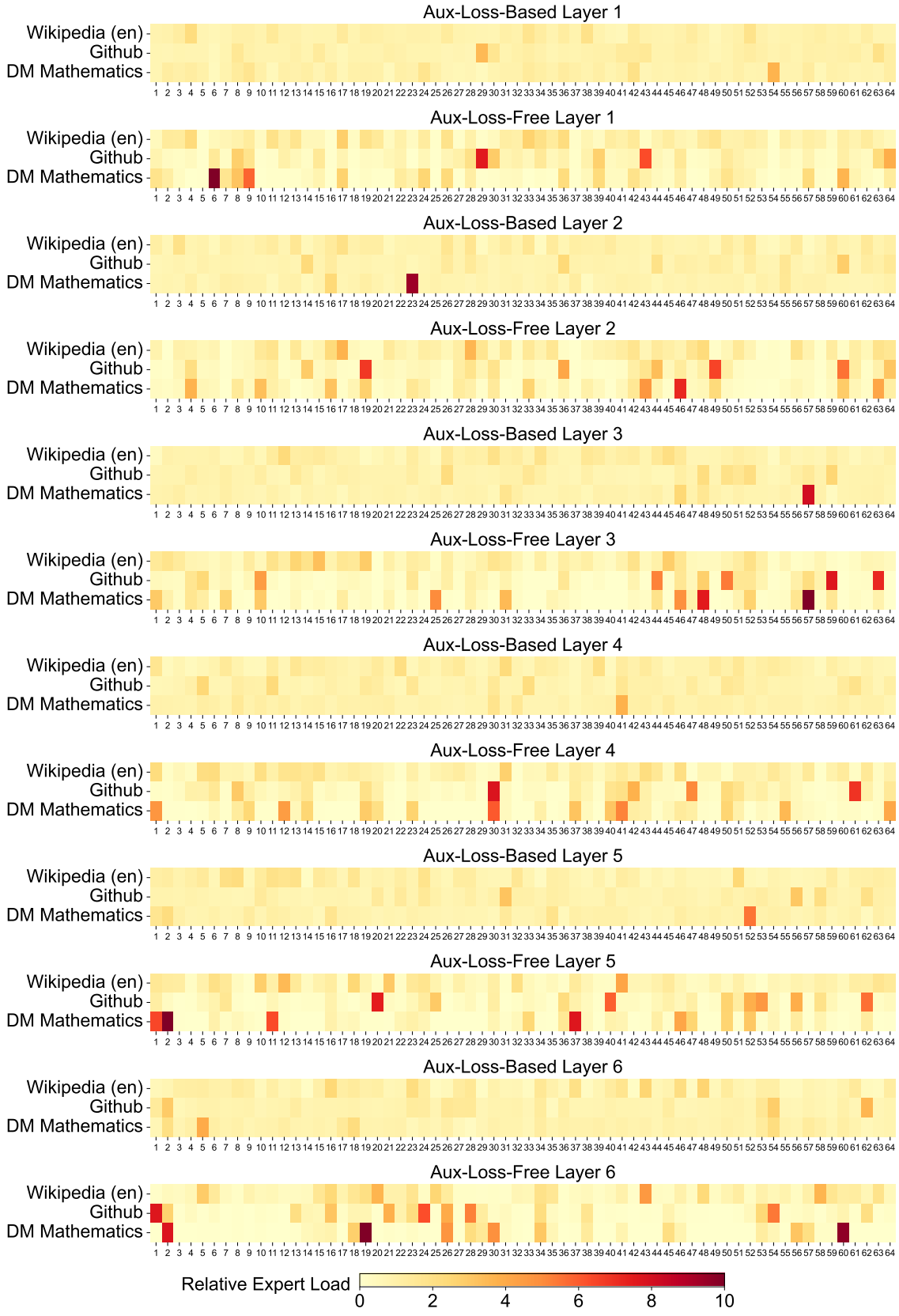
我们通过与两种基线模型在不同规模上进行比较，验证了我们的 FP8 混合精度框架与 BF16 训练的效果。在小规模上，我们训练了一个基线 MoE 模型，总参数约为 16B，使用了 1.33T 的 Token。在大规模上，我们训练了一个基线 MoE 模型，总参数约为 230B，使用了大约 0.9T 的 Token。我们在图 10 中展示了训练曲线，并证明在我们的高精度累积和细粒度量化策略下，相对误差保持在 0.25% 以下。

A.2 有关块级量化的讨论

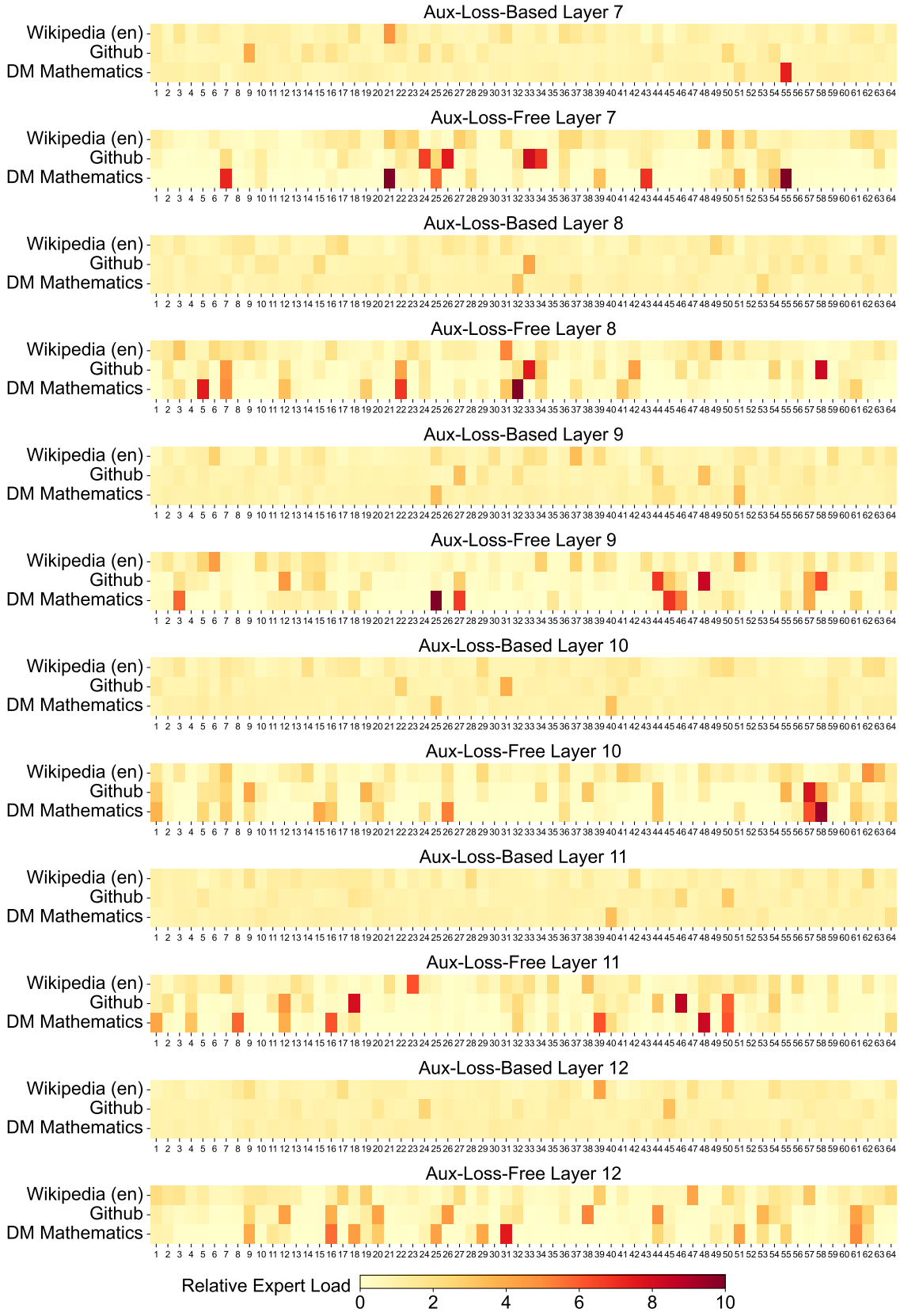
尽管我们的块状细粒度量化有效减轻了特征异常值引入的误差，但它需要对激活量化进行不同的分组，即在前向传播中使用 `1x128`，而在反向传播中使用 `128x1`。激活梯度也需要类似的过程。一种简单的策略是对每 `128x128` 元素进行块状量化，就像我们对模型权重进行量化的方式一样。这样，反向传播时只需要进行转置。因此，我们进行了一项实验，其中与 `Dgrad` 相关的所有张量都在块状基础上进行量化。结果显示，计算激活梯度并以链式方式向浅层反向传播的 `Dgrad` 操作对精度非常敏感。具体而言，对激活梯度进行块状量化会导致一个约 16B 总参数的 MoE 模型在训练约 300B Token 时发生模型发散。我们假设这种敏感性产生的原因是激活梯度在 Token 之间高度不平衡，导致与 Token 相关的异常值。这些异常值无法通过块状量化方法有效管理。

B 16B 辅助损失模型和无辅助损失模型的专家专长模式

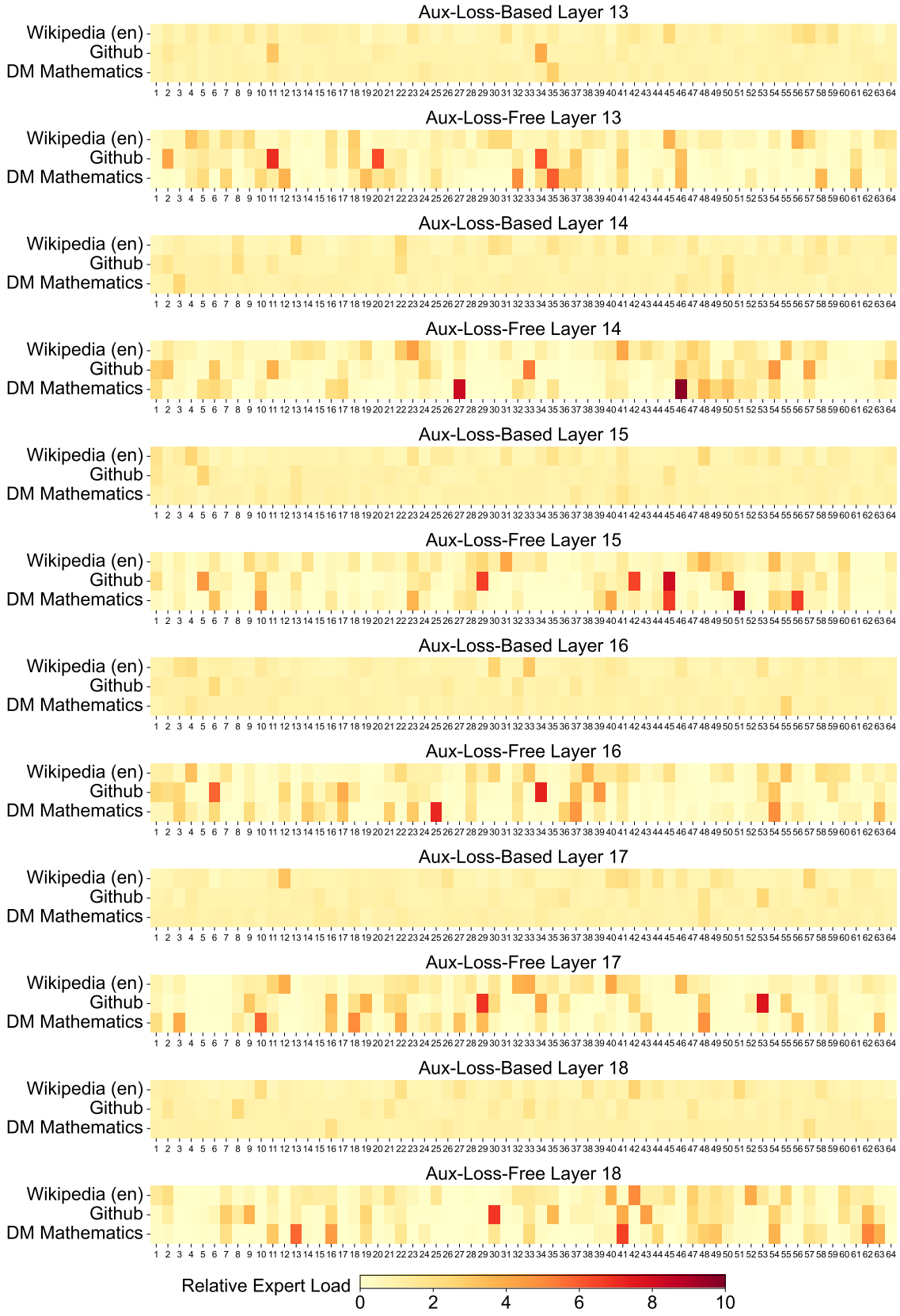
我们记录了 16B 辅助损失基线模型和无辅助损失模型在 Pile 测试集上的专家负载。无辅助损失模型在所有层次上通常具有更强的专家专长，如图 10 所示。


Aux-Loss-Based Layer 1
Aux-Loss-Free Layer 1
Aux-Loss-Based Layer 2
Aux-Loss-Free Layer 2
Aux-Loss-Based Layer 3
Aux-Loss-Free Layer 3
Aux-Loss-Based Layer 4
Aux-Loss-Free Layer 4
Aux-Loss-Based Layer 5
Aux-Loss-Free Layer 5
Aux-Loss-Based Layer 6
Aux-Loss-Free Layer 6
Wikipedia (en)
Github
DM Mathematics
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64
Relative Expert Load
0
2
4
6
8
10

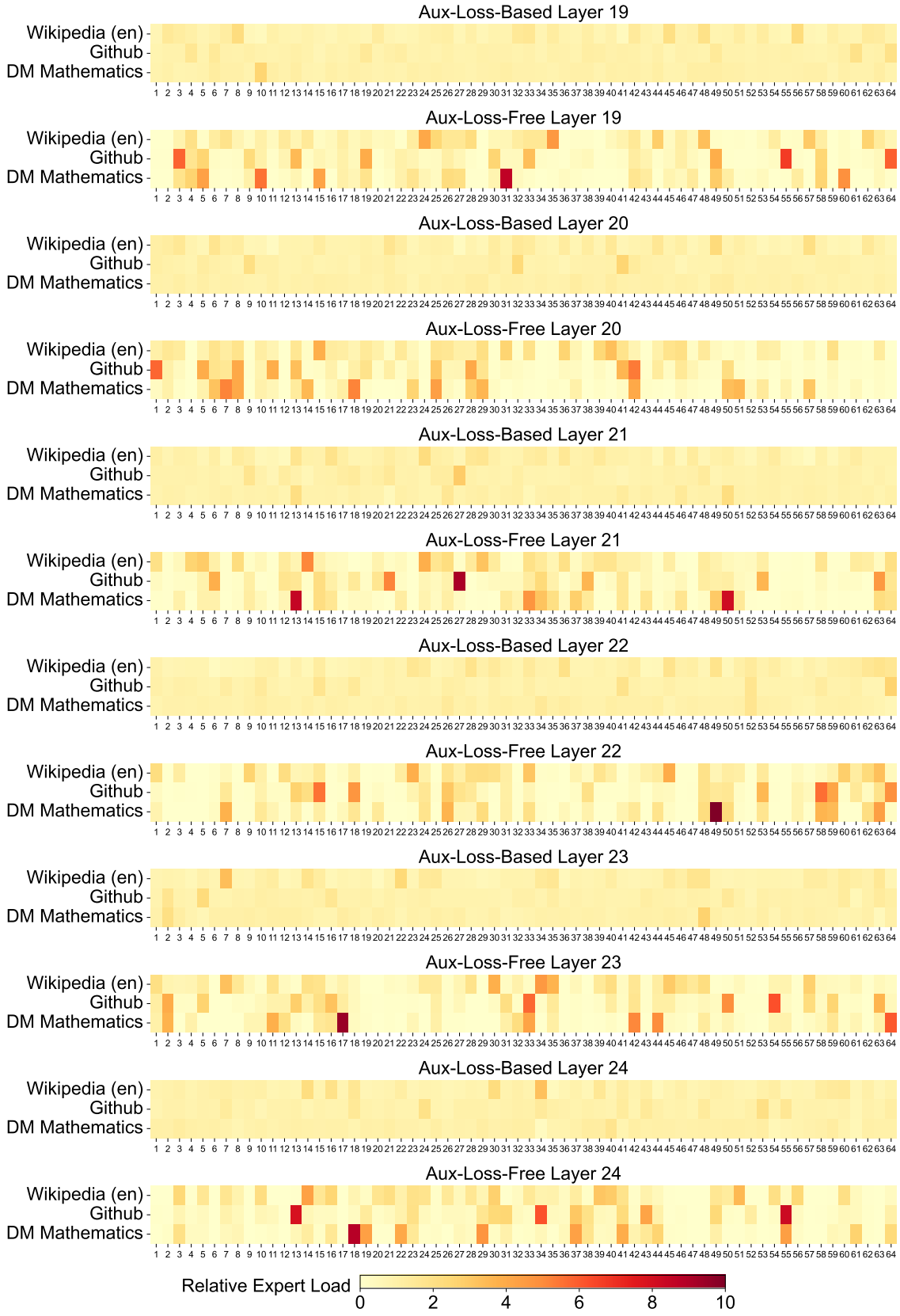
(a) Layers 1-7


Aux-Loss-Based Layer 7
Aux-Loss-Free Layer 7
Aux-Loss-Based Layer 8
Aux-Loss-Free Layer 8
Aux-Loss-Based Layer 9
Aux-Loss-Free Layer 9
Aux-Loss-Based Layer 10
Aux-Loss-Free Layer 10
Aux-Loss-Based Layer 11
Aux-Loss-Free Layer 11
Aux-Loss-Based Layer 12
Aux-Loss-Free Layer 12
Wikipedia (en)
Github
DM Mathematics
Relative Expert Load
0
2
4
6
8
10

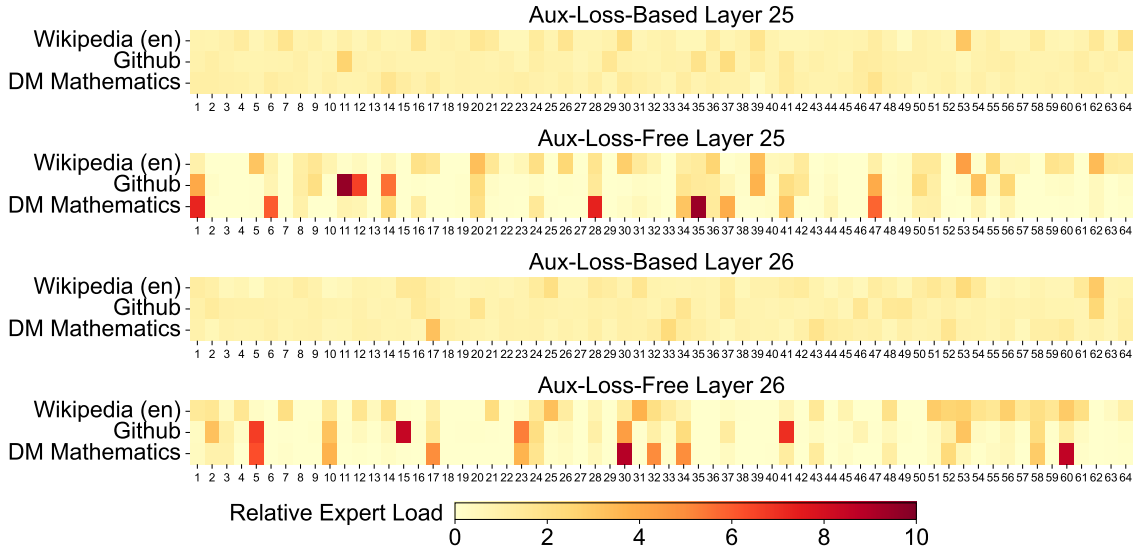
(b) Layers 7-13



(c) Layers 13-19



(d) Layers 19-25



(e) Layers 25-27

图 10: 无辅助损失模型和有辅助损失模型在 Pile 测试集三个领域的专家负载。无辅助损失模型显示出比有辅助损失模型更强的专家专长模式。相对专家负载表示实际专家负载与理论平衡专家负载之间的比率。