

本地稀疏注意力：硬件对齐和本地可训练的稀疏注意力

袁京阳^{*1,2}, 高华作¹, 戴大麦¹, 罗俊宇², 赵亮¹, 张正言¹, 谢振达¹, 魏宇轩¹, 王怜¹, 肖志平³, 王玉清¹, 阮冲¹, 张铭², 梁文峰¹, 曾望鼎¹

¹DeepSeek-AI ²多媒体信息处理重点实验室, 北京大学, P
KU-Anker LLM实验室 ³华盛顿大学 {yuanjy, mzhang_cs}@pku.edu.cn,
{zengwangding, wenfeng.liang}@deepseek.com

摘要

长上下文建模对下一代语言模型至关重要, 但标准注意机制的高计算成本带来了显著的计算挑战。稀疏注意提供了一个有前景的方向, 以提高效率, 同时保持模型能力。我们提出了NSA, 一种本地可训练的稀疏注意机制, 它将算法创新与硬件对齐的优化相结合, 以实现高效的长上下文建模。NSA采用动态分层稀疏策略, 结合粗粒度的令牌压缩和细粒度的令牌选择, 以保持全球上下文意识和局部精度。我们的方法通过两个关键创新推动了稀疏注意设计: (1) 我们通过算术强度平衡的算法设计实现了显著的加速, 并针对现代硬件进行了实现优化。(2) 我们实现了端到端训练, 减少了预训练计算而不牺牲模型性能。如图1所示, 实验表明, 使用NSA预训练的模型在一般基准、长上下文任务和基于指令的推理中保持或超过了全注意模型。同时, NSA在64k长度序列的解码、前向传播和反向传播中相较于全注意实现了显著的加速, 验证了其在模型生命周期中的效率。

1. 引言

研究界越来越认识到长上下文建模是下一代大型语言模型的一项关键能力, 这一能力受到多种现实世界应用的推动, 这些应用包括深入推理 (DeepSeek-AI, 2025; Zelikman等, 2022)、库级代码生成 (Zhang等, 2023a; Zhang等) 和多轮自主代理系统 (Park等, 2023)。最近的突破, 包括OpenAI的o系列模型、DeepSeek-R1 (DeepSeek-AI, 2025) 和Gemini 1.5 Pro (Google等, 2024), 使模型能够处理整个代码库、冗长的文档, 保持数千个标记的连贯多轮对话, 并在长距离依赖中执行复杂推理。然而, 随着序列长度的增加, 普通注意力机制 (Vaswani等, 2017) 的高复杂性 (Zaheer等, 2020) 成为一个关键的延迟瓶颈。理论估计表明, 注意力 $\{v^*\}$

*Contribution during internship at DeepSeek-AI.

5
2
0
2
b
e
F
6
1
L
C
s
c
1
v
9
8
0
1
1
2
0
5
2
v
i
X
r
a

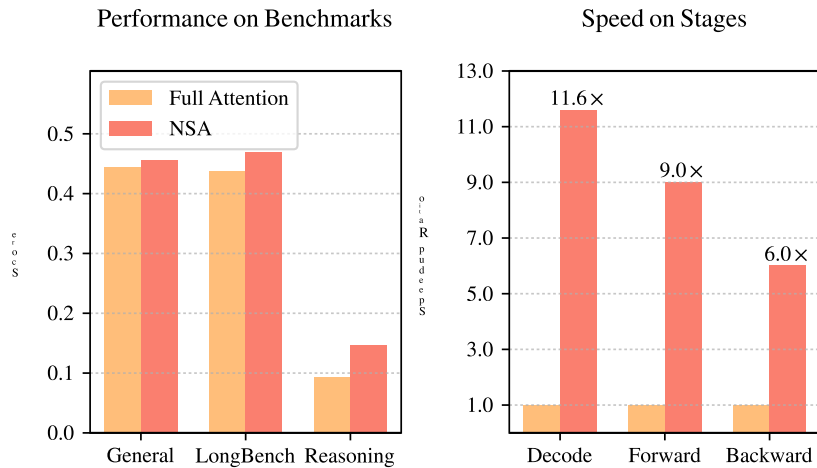


图1 | 全注意力模型与我们的NSA在性能和效率上的比较。左侧：尽管稀疏，NSA在一般基准、长上下文任务和推理评估中平均超越全注意力基线。右侧：在处理64k长度序列时，NSA在解码、前向传播和反向传播的所有阶段相比全注意力实现了显著的计算加速。

使用softmax架构的计算在解码64k长度上下文时占总延迟的70-80%，凸显了对更高效注意力机制的迫切需求。

一种自然的高效长上下文建模方法是利用softmax注意力的固有稀疏性 (Ge et al., 2023; Jiang et al., 2023)，在这种方法中，选择性地计算关键的查询-键对可以显著减少计算开销，同时保持性能。最近的进展通过多种策略展示了这一潜力：KV缓存驱逐方法 (Li et al., 2024; Zhang et al., 2023b; Zhou et al., 2024)、块状KV缓存选择方法 (Tang et al., 2024; Xiao et al., 2024) 以及基于采样、聚类或哈希的选择方法 (Chen et al., 2024; Desai et al., 2024; Liu et al., 2024)。尽管这些有前景的策略，现有的稀疏注意力方法在实际部署中往往表现不佳。许多方法未能实现与其理论增益相当的加速；此外，大多数方法主要集中在推理阶段，缺乏有效的训练时间支持，以充分利用注意力的稀疏性模式。

为了应对这些限制，有效稀疏注意力的部署必须解决两个关键挑战：(1)

Hardware-aligned inference speedup: 将理论计算减少转化为实际速度提升需要在预填充和解码阶段进行硬件友好的算法设计，以减轻内存访问和硬件调度瓶颈；(2)

Training-aware algorithm design: 启用具有可训练算子的端到端计算，以降低训练成本，同时保持模型性能。这些要求对于实现快速长上下文推理或训练的现实应用至关重要。在考虑这两个方面时，现有方法仍然存在明显差距。

为了实现更有效和高效的稀疏注意力，我们提出了NSA，一种原生可训练的稀疏注意力架构，集成了分层令牌建模。如图2所示，NSA通过将键和值组织成时间块并通过三条注意力路径处理它们，从而减少每个查询的计算：压缩的粗粒度令牌、选择性保留的细粒度令牌和用于局部上下文信息的滑动窗口。然后，我们实现了专门的内核以最大化其实际效率。NSA引入了两个

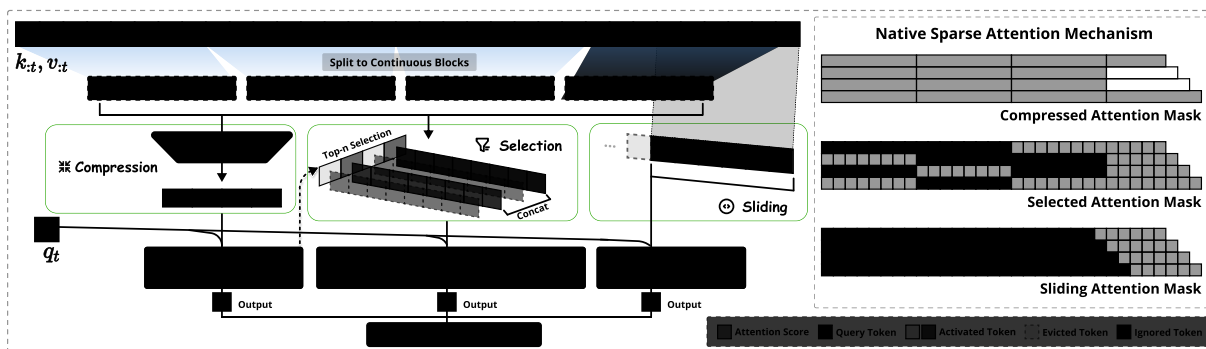


图2 | NSA架构概述。左侧：该框架通过三个并行注意力分支处理输入序列：对于给定的查询，前面的键和值被处理为粗粒度模式的压缩注意力、重要令牌块的选择注意力和局部上下文的滑动注意力。右侧：每个分支产生的不同注意力模式的可视化。绿色区域表示需要计算注意力分数的区域，而白色区域表示可以跳过的区域。

与上述关键需求相对应的核心创新：(1) 硬件对齐系统：优化块状稀疏注意力以提高Tensor Core的利用率和内存访问，确保算术强度平衡。(2) 训练感知设计：通过高效算法和反向算子实现稳定的端到端训练。此优化使NSA能够支持高效部署和端到端训练。

我们通过在真实世界语言语料库上进行全面实验来评估NSA。在一个具有27B参数的变换器骨干上进行260B标记的预训练，我们评估NSA在一般语言评估、长上下文评估和思维链推理评估中的表现。我们进一步比较了在A100 GPU上使用优化的Triton (Tillet et al., 2019) 实现的内核速度。实验结果表明，NSA在性能上与全注意力基线相当或更优，同时超越了现有的稀疏注意力方法。此外，与全注意力相比，NSA在解码、前向和后向阶段提供了显著的加速，且对于更长的序列，加速比率不断增加。这些结果验证了我们的层次稀疏注意力设计有效地平衡了模型能力和计算效率。

2. 重新思考稀疏注意力方法

现代稀疏注意力方法在降低变换器模型的理论计算复杂性方面取得了显著进展。然而，大多数方法主要在推理过程中应用稀疏性，同时保留预训练的全注意力骨干网络，这可能引入架构偏差，限制了它们充分利用稀疏注意力优势的能力。在介绍我们本土的稀疏架构之前，我们通过两个关键视角系统地分析这些局限性。

2.1. 高效推理的幻觉

尽管在注意力计算中实现了稀疏性，但许多方法未能相应地减少推理延迟，主要由于两个挑战：

相位限制稀疏性。方法如 H2O (Zhang et al., 2023b) 应用稀疏性

在自回归解码期间，需要在预填充过程中进行计算密集型的预处理（例如，注意力图计算、索引构建）。相比之下，像 MInference (Jiang et al., 2024) 这样的方案仅专注于预填充稀疏性。这些方法未能在所有推理阶段实现加速，因为至少有一个阶段的计算成本与完全注意力相当。阶段专业化降低了这些方法在以预填充为主的工作负载（如书籍摘要和代码补全）或以解码为主的工作负载（如长链思维 (Wei et al., 2022) 推理）中的加速能力。

与高级注意力架构的不兼容性。一些稀疏注意力方法未能适应现代解码高效架构，如多查询注意力 (MQA) (Shazeer, 2019) 和分组查询注意力 (GQA) (Ainslie等, 2023)，这些架构通过在多个查询头之间共享KV显著减少了解码过程中的内存访问瓶颈。例如，在Quest (Tang等, 2024) 等方法中，每个注意力头独立选择其KV缓存子集。尽管它在多头注意力 (MHA) 模型中表现出一致的稀疏性和内存访问稀疏性，但在基于GQA等架构的模型中则呈现出不同的场景，其中KV缓存的内存访问量对应于同一GQA组内所有查询头选择的并集。这种架构特性意味着，尽管这些方法可以减少计算操作，但所需的KV缓存内存访问仍然相对较高。这一限制迫使我们做出一个关键选择：虽然一些稀疏注意力方法减少了计算，但它们分散的内存访问模式与高级架构的高效内存访问设计相冲突。

这些限制出现是因为许多现有的稀疏注意力方法专注于KV缓存减少或理论计算减少，但在先进框架或后端中难以实现显著的延迟减少。这激励我们开发结合先进架构和硬件高效实现的算法，以充分利用稀疏性来提高模型效率。

2.2. 可训练稀疏性的神话

我们对原生可训练稀疏注意力的追求源于对仅推理方法分析的两个关键见解：(1)

Performance Degradation: 事后施加稀疏性迫使模型偏离其预训练的优化轨迹。正如陈等人 (2024) 所示，前20%的注意力只能覆盖70%的总注意力分数，这使得预训练模型中的检索头等结构在推理过程中容易受到剪枝的影响。(2) **Training Efficiency Demands:** 高效处理长序列训练对现代大型语言模型 (LLM) 开发至关重要。这包括在更长的文档上进行预训练以增强模型能力，以及后续的适应阶段，如长上下文微调和强化学习。然而，现有的稀疏注意力方法主要针对推理，导致训练中的计算挑战基本上未得到解决。这一限制阻碍了通过高效训练开发更强大的长上下文模型。此外，努力将现有稀疏注意力适应于训练也暴露出挑战：

不可训练组件。像ClusterKV (Liu et al., 2024) (包括k-means聚类) 和MagicPIG (Chen et al., 2024) (包括基于SimHash的选择) 等方法中的离散操作在计算图中创建了不连续性。这些不可训练组件阻止了通过令牌选择过程的梯度流动，限制了模型学习最佳稀疏模式的能力。

低效的反向传播。一些理论上可训练的稀疏注意力方法在实际训练中存在效率低下的问题。在这些方法中使用的基于令牌粒度的选择策略

像 HashAttention (Desai 等, 2024) 导致在注意力计算过程中需要从 KV 缓存中加载大量单独的标记。这种不连续的内存访问阻碍了快速注意力技术 (如 FlashAttention) 的有效适应, 而这些技术依赖于连续的内存访问和块状计算以实现高吞吐量。因此, 实施被迫退回到低硬件利用率, 显著降低了训练效率。

2.3. 原生稀疏性作为一种必要性

这些推理效率和训练可行性方面的限制促使我们对稀疏注意机制进行根本性的重新设计。我们提出了 NSA, 一种本质上稀疏的注意框架, 旨在解决计算效率和训练需求。在接下来的部分中, 我们详细介绍了 NSA 的算法设计和操作实现。

3. 方法论

我们的技术方法涵盖算法设计和内核优化。在接下来的子章节中, 我们首先介绍我们方法论的背景。然后, 我们展示 NSA 的整体框架, 接着是其关键算法组件。最后, 我们详细介绍我们优化硬件的内核设计, 以最大化实际效率。

3.1. 背景

注意机制广泛应用于语言建模, 其中每个查询令牌 \mathbf{q}_t 计算与所有前面的键 $\mathbf{k}_{:t}$ 的相关性得分, 以生成值 \mathbf{v}_t 的加权。形式上, 对于长度为 t 的输入序列, 注意操作定义为:

$$\mathbf{o}_t = \text{Attn}(\mathbf{q}_t, \mathbf{k}_{:t}, \mathbf{v}_{:t}) \quad (1)$$

其中 Attn 表示注意力函数:

$$\text{Attn}(\mathbf{q}_t, \mathbf{k}_{:t}, \mathbf{v}_{:t}) = \sum_{i=1}^t \frac{\alpha_{t,i} \mathbf{v}_i}{\sum_{j=1}^t \alpha_{t,j}}, \quad \alpha_{t,i} = e^{\frac{\mathbf{q}_t^\top \mathbf{k}_i}{\sqrt{d_k}}}. \quad (2)$$

在这里, $\alpha_{t,i}$ 表示 \mathbf{q}_t 和 \mathbf{k}_i 之间的注意力权重, 而 d_k 是键的特征维度。随着序列长度的增加, 注意力计算在整体计算成本中变得越来越占主导地位, 这给长上下文处理带来了重大挑战。

算术强度是计算操作与内存访问的比率。它本质上影响硬件上的算法优化。每个 GPU 都有一个由其峰值计算能力和内存带宽决定的关键算术强度, 计算方法是这两个硬件限制的比率。对于计算任务, 超过这个关键阈值的算术强度变为计算限制 (受 GPU FLOPS 限制), 而低于它则变为内存限制 (受内存带宽限制)。

具体来说, 对于因果自注意力机制, 在训练和预填充阶段, 批量矩阵乘法和注意力计算表现出高算术强度, 使这些阶段在现代加速器上受限于计算。相比之下, 自回归解码受到内存带宽的限制, 因为它在每次前向传递中生成一个标记, 同时需要加载整个键值缓存, 导致低算术强度。这导致了不同的优化目标——在训练和预填充期间减少计算成本, 而在解码期间减少内存访问。

3.2. 整体框架

为了利用自然稀疏模式的注意力潜力，我们建议在公式 (1) 中用更紧凑和信息密集的代表键值对 \tilde{k}_t 、 \tilde{v}_t 替换原始的键值对 $k_{:t}$ 、 $v_{:t}$ ，以便于每个查询 q_{t_0} 。具体而言，我们正式定义优化后的注意力输出如下：

$$\tilde{K}_t = f_K(\mathbf{q}_t, \mathbf{k}_{:t}, \mathbf{v}_{:t}), \quad \tilde{V}_t = f_V(\mathbf{q}_t, \mathbf{k}_{:t}, \mathbf{v}_{:t}) \quad (3)$$

$$\mathbf{o}_t^* = \text{Attn}(\mathbf{q}_t, \tilde{K}_t, \tilde{V}_t) \quad (4)$$

其中 \tilde{K}_t 、 \tilde{V}_t 是基于当前查询 q_t 和上下文记忆 $k_{:t}$ 、 $v_{:t}$ 动态构建的。我们可以设计各种映射策略以获取不同类别的 \tilde{K}_t^c 、 \tilde{V}_t^c ，并将它们组合如下：

$$\mathbf{o}_t^* = \sum_{c \in C} g_t^c \cdot \text{Attn}(\mathbf{q}_t, \tilde{K}_t^c, \tilde{V}_t^c). \quad (5)$$

如图2所示，NSA有三种映射策略 $C = \{\text{cmp}, \text{slc}, \text{win}\}$ ，分别代表键和值的压缩、选择和滑动窗口。 $g_t^c \in [0, 1]$ 是对应策略 c 的门得分，通过MLP和sigmoid激活从输入特征中得出。令 N_t 表示重新映射的键/值的总数：

$$N_t = \sum_{c \in C} \text{size}[\tilde{K}_t^c]. \quad (6)$$

我们通过确保 $N_t \ll t$ 来维持高稀疏比。

3.3. 算法设计

在本小节中，我们介绍了我们的重映射策略 f_K 和 f_V 的设计：令牌压缩、令牌选择和滑动窗口。

3.3.1. Token Compression

通过将连续的键或值块聚合成块级表示，我们获得了压缩的键和值，这些键和值捕捉了整个块的信息。正式地，压缩的键表示定义为：

$$\tilde{K}_t^{\text{cmp}} = f_K^{\text{cmp}}(\mathbf{k}_{:t}) = \left\{ \varphi(\mathbf{k}_{id+1:id+l}) \mid 1 \leq i \leq \left\lfloor \frac{t-l}{d} \right\rfloor \right\} \quad (7)$$

其中 l 是块长度， d 是相邻块之间的滑动步幅， φ 是一个可学习的 MLP，具有块内位置编码，用于将块中的键映射到单个压缩键。 $\tilde{K}_t^{\text{cmp}} \in \mathbb{R}^{d_k \times \lfloor \frac{t-l}{d} \rfloor}$ 是由压缩键组成的张量。通常，我们采用 $d < l$ 来减轻信息碎片化。压缩值表示 \tilde{V}_t^{cmp} 的类似公式也适用。压缩表示捕获了更粗粒度的高级语义信息，并减少了注意力的计算负担。

3.3.2. Token Selection

仅使用压缩键，值可能会丢失重要的细粒度信息，这促使我们选择性地保留单个键和值。下面我们描述了我们高效的令牌选择机制，该机制识别并保留最相关的令牌，同时具有低计算开销。

块状选择。我们的选择策略在空间连续块中处理键和值序列，这受到两个关键因素的驱动：硬件效率考虑和注意力分数的固有分布模式。**Blockwise selection is crucial to achieve efficient computation on modern GPUs.** 这是因为现代GPU架构在连续块访问方面的吞吐量显著高于随机索引读取。此外，块状计算能够实现Tensor Cores的最佳利用。这一架构特性使得块状内存访问和计算成为高性能注意力实现的基本原则，正如FlashAttention的基于块的设计所示。

Blockwise selection follows the inherent distribution patterns of attention scores. 先前的研究（Jiang等，2024）表明，注意力分数通常表现出空间连续性，这表明相邻的键往往具有相似的重要性水平。我们在第6.2节中的可视化也显示了这种空间连续模式。



为了实现块选择，我们首先将键值序列划分为选择块。为了识别对注意力计算最重要的块，我们需要为每个块分配重要性分数。下面我们介绍计算这些块级重要性分数的方法。

重要性分数计算。计算块的重要性分数可能会引入显著的开销。幸运的是，压缩令牌的注意力计算产生的中间注意力分数可以被用来引导选择块的重要性分数，公式化为： $\{v^*\}$

$$\mathbf{p}_t^{\text{cmp}} = \text{Softmax}\left(\mathbf{q}_t^T \tilde{\mathbf{K}}_t^{\text{cmp}}\right), \quad (8)$$

其中 $\mathbf{p}_t^{\text{cmp}} \in \mathbb{R}^{\lfloor \frac{l}{d} \rfloor}$ 是 \mathbf{q}_t 和压缩键 $\tilde{\mathbf{K}}_t^{\text{cmp}}$ 之间的注意力分数。令 l' 表示选择块大小。当压缩块和选择块共享相同的分块方案，即 $l' = l = d$ 时，我们可以直接通过 $\mathbf{p}_t^{\text{slc}} = \mathbf{p}_t^{\text{cmp}}$ 直接获得选择块的重要性分数 $\mathbf{p}_t^{\text{slc}}$ 。对于分块方案不同的情况，我们根据选择块的空间关系推导其重要性分数。给定 $d \mid l$ 和 $d \mid l'$ ，我们有：

$$\mathbf{p}_t^{\text{slc}}[j] = \sum_{m=0}^{\frac{l'}{d}-1} \sum_{n=0}^{\frac{l}{d}-1} \mathbf{p}_t^{\text{cmp}} \left[\frac{l'}{d}j + m + n \right], \quad (9)$$

其中 $[\cdot]$ 表示用于访问向量元素的索引操作符。对于采用GQA或MQA的模型，其中键值缓存在查询头之间共享，必须确保在这些头之间的一致块选择，以最小化解码期间的KV缓存加载。组内头之间共享的重要性分数正式定义为：

$$\mathbf{p}_t^{\text{slc}'} = \sum_{h=1}^H \mathbf{p}_t^{\text{slc},(h)}, \quad (10)$$

其中 (h) 在上标中表示头索引，而 H 是每组中的查询头数量。这种聚合确保在同一组内各个头之间的一致块选择。

顶级- n 块选择。在获得选择块重要性分数后，我们保留在按块重要性分数排名的前- n 稀疏块中的标记，公式如下：

$$\mathcal{I}_t = \{i \mid \text{rank}(\mathbf{p}_t^{\text{slc}'}[i]) \leq n\} \quad (11)$$

$$\tilde{\mathbf{K}}_t^{\text{slc}} = \text{Cat} [\{\mathbf{k}_{i'l'+1:(i+1)l'} \mid i \in \mathcal{I}_t\}], \quad (12)$$

其中 $\text{rank}(\cdot)$ 表示按降序排列的排名位置， $\text{rank} = 1$ 对应于最高分， \mathcal{I}_t 是所选块的索引集合， Cat 表示连接操作。 $\tilde{\mathbf{K}}_t^{\text{slc}} \in \mathbb{R}^{d_k \times nl'}$ 是由压缩键组成的张量。类似的公式适用于细粒度值 $\tilde{\mathbf{V}}_t^{\text{slc}}$ 。然后，所选的键和值参与与 \mathbf{q}_t 的注意力计算，如公式 (5) 所定义。

□

3.3.3. Sliding Window

在注意力机制中，局部模式通常适应得更快，并且可能主导学习过程，从而可能阻止模型有效地从压缩和选择标记中学习。为了解决这个问题，我们引入了一个专门的滑动窗口分支，明确处理局部上下文，使其他分支（压缩和选择）能够专注于学习各自的特征，而不被局部模式所干扰。具体来说，我们在窗口 w 中维护最近的标记 $\tilde{\mathbf{K}}_t^{\text{win}} = \mathbf{k}_{t-w:t}$ 、 $\tilde{\mathbf{V}}_t^{\text{win}} = \mathbf{v}_{t-w:t}$ ，并将不同信息来源（压缩标记和选择标记、滑动窗口）的注意力计算隔离到不同的分支中。这些分支输出通过学习的大门控机制进行聚合。为了进一步防止在注意力分支之间的快捷学习，同时保持边际计算开销，我们为三个分支提供独立的键和值。这种架构设计通过防止局部和长程模式识别之间的梯度干扰，实现了稳定的学习，同时引入了最小的开销。

在获得所有三类键和值 ($\tilde{\mathbf{K}}_t^{\text{cmp}}$, $\tilde{\mathbf{V}}_t^{\text{cmp}}$; $\tilde{\mathbf{K}}_t^{\text{slc}}$, $\tilde{\mathbf{V}}_t^{\text{slc}}$; 以及 $\tilde{\mathbf{K}}_t^{\text{win}}$, $\tilde{\mathbf{V}}_t^{\text{win}}$) 后，我们根据公式 (5) 计算最终的注意力输出。结合上述的压缩、选择和滑动窗口机制，这构成了NSA的完整算法框架。

3.4. 核心设计

为了在训练和预填充期间实现FlashAttention级别的加速，我们在Triton上实现了硬件对齐的稀疏注意力内核。考虑到多头注意力（MHA）在解码时内存密集且效率低下，我们专注于具有共享KV缓存的架构，如GQA和MQA，遵循当前最先进的LLM。虽然压缩和滑动窗口注意力计算与现有的FlashAttention-2内核兼容，但我们引入了专门的稀疏选择注意力内核设计。如果我们遵循FlashAttention的策略，将时间上连续的查询块加载到SRAM中，这将导致低效的内存访问，因为一个块内的查询可能需要不相交的KV块。为了解决这个问题，我们的关键优化在于不同的查询分组策略：对于查询序列中的每个位置，我们将GQA组内的所有查询头（它们共享相同的稀疏KV块）加载到SRAM中。图3展示了我们的前向传播实现。所提出的内核架构具有以下关键特征：

□

1. 以组为中心的数据加载。对于每个内循环，在位置 t 加载组中所有头的查询 $\mathbf{Q} \in \mathbb{R}^{[h, d_k]}$ 及其共享的稀疏键/值块索引 \mathcal{I}_t 。

2. 共享 KV 获取。在内循环中，顺序加载由 I_t 索引的连续键/值块到 SRAM 中作为 $K \in \mathbb{R}^{[B_k, d_k]}$ 、 $V \in \mathbb{R}^{[B_k, d_v]}$ ，以最小化内存加载，其中 B_k 是满足 $B_k | l'$ 的内核块大小。3. 网格上的外循环。由于内循环长度（与选定的块数 n 成正比）对于不同的查询块几乎保持不变，我们将查询/输出循环放入 Triton 的网格调度器中，以简化和优化内核。

该设计通过（1）通过组内共享消除冗余的KV传输，以及（2）在GPU流处理器之间平衡计算负载，实现了接近最佳的算术强度。

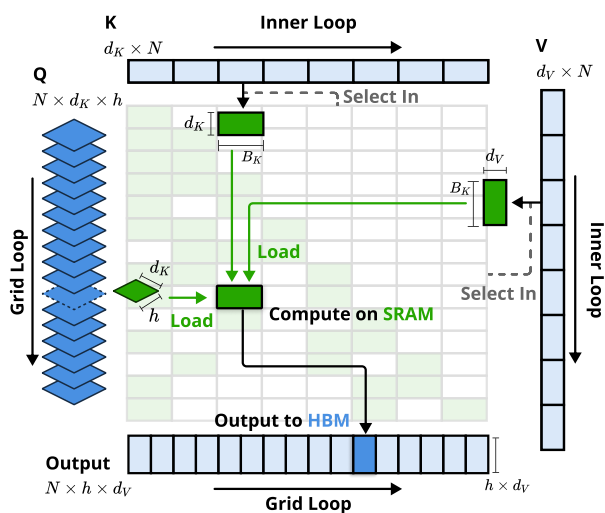


图3 | NSA的内核设计。内核通过GQA组（网格循环）加载查询，获取相应的稀疏KV块（内部循环），并在SRAM上执行注意力计算。绿色块表示SRAM上的数据，而蓝色块表示HBM上的数据。

4. 实验

我们通过三个方面评估NSA：（1）一般基准性能，（2）长上下文基准性能，以及（3）链式推理性能，比较全注意力基线和最先进的稀疏注意力方法。我们将稀疏计算范式的效率分析推迟到第5节，在那里我们提供关于训练和推理速度的详细讨论。

4.1. 预训练设置

根据最先进的LLM的常见做法，我们的实验采用了结合了分组查询注意力（GQA）和专家混合（MoE）的骨干网络，具有27B的总参数和3B的活跃参数。该模型由30层组成，隐藏维度为2560。对于GQA，我们将组的数量设置为4，总共64个注意力头。对于每个头，查询、键和值的隐藏维度分别配置为 $d_q = d_k = 192$ 和 $d_v = 128$ 。对于MoE，我们利用DeepSeekMoE（Dai等，2024；DeepSeek-AI，2024）结构，具有72个路由专家和2个共享专家，并将前k个专家设置为6。为了确保训练的稳定性，第一层的MoE被替换为SwiGLU形式的MLP。

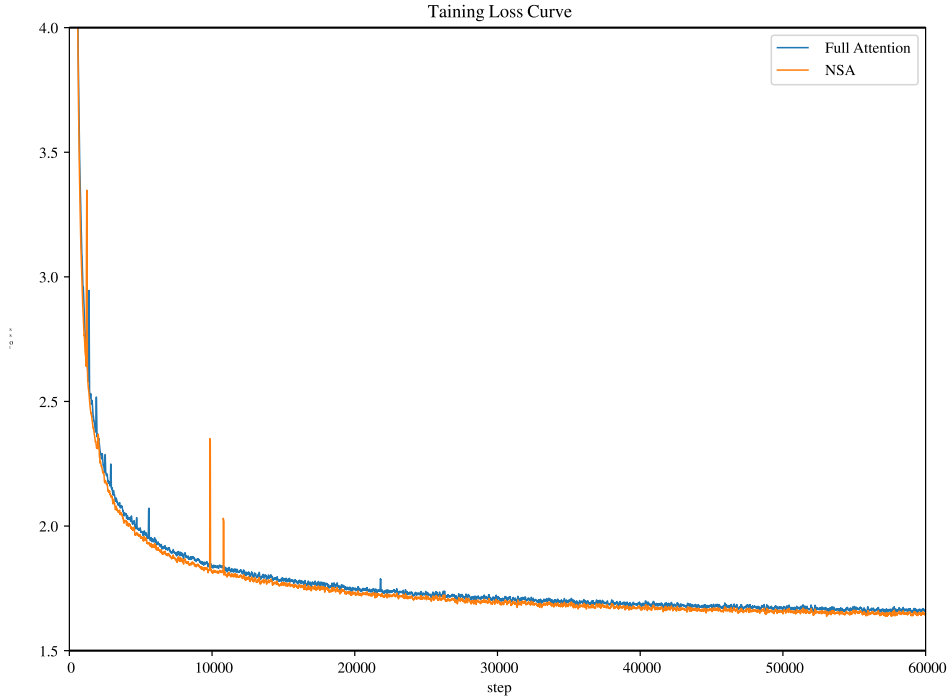


图4 | 全注意力与我们的NSA在27B参数模型上的预训练损失比较。两个模型都表现出稳定的收敛，NSA实现了更低的损失值。

Model	MMLU Acc. 5-shot	MMLU-PRO Acc. 5-shot	CMMLU Acc. 5-shot	BBH Acc. 3-shot	GSM8K Acc. 8-shot	MATH Acc. 4-shot	DROP F1 1-shot	MBPP Pass@1 3-shot	HumanEval Pass@1 0-shot	Avg.
Full Attn	0.567	0.279	0.576	0.497	0.486	0.263	0.503	0.482	0.335	0.443
NSA	0.565	0.286	0.587	0.521	0.520	0.264	0.545	0.466	0.348	0.456

表1 | 全注意力基线与NSA在一般基准上的预训练性能比较，涵盖知识（MMLU, MMLU-PRO, CMMLU）、推理（BBH, GSM8K, MATH, DROP）和编码（MBPP, HumanEval）任务。尽管稀疏性较高，NSA在大多数基准上实现了更优的平均性能。

所提议的架构在计算成本和模型性能之间实现了有效的权衡。对于NSA，我们设置压缩块大小 $l = 32$ ，滑动步幅 $d = 16$ ，选择的块大小 $l' = 64$ ，选择的块数量 $n = 16$ （包括固定激活的1个初始块和2个局部块），以及滑动窗口大小 $w = 512$ 。全注意力和稀疏注意力模型均在270B个8k长度文本上进行预训练，随后在32k长度文本上进行继续训练和监督微调，使用YaRN（Peng et al., 2024）以实现长上下文适应。两个模型均训练到完全收敛，以确保公平比较。如图4所示，我们的NSA和全注意力基线的预训练损失曲线表现出稳定和平滑的下降，NSA始终优于全注意力模型。 □

4.2. 基线方法

除了与全注意力进行比较，我们评估了几种最先进的推理阶段稀疏注意力方法：H2O（Zhang et al., 2023b）、infLLM（Xiao et al., 2024）、Quest（Tang et al., 2024）和Exact-Top，它首先计算全注意力得分并选择与每个查询对应的前- n 个得分键，然后在这些位置上计算注意力。这些方法

Model	SQA			MQA				Synthetic		Code	Avg.
	MFQA-en	MFQA-zh	Qasper	HPQ	2Wiki	GovRpt	Dur	PassR-en	PassR-zh	LCC	
H2O	0.428	0.429	0.308	0.112	0.101	0.231	0.208	0.704	0.421	0.092	0.303
InfLLM	0.474	0.517	0.356	0.306	0.250	0.277	0.257	0.766	0.486	0.143	0.383
Quest	0.495	0.561	0.365	0.295	0.245	0.293	0.257	0.792	0.478	0.135	0.392
Exact-Top	0.502	0.605	0.397	0.321	0.288	0.316	0.291	0.810	0.548	0.156	0.423
Full Attn	0.512	0.623	0.409	0.350	0.305	0.324	0.294	0.830	0.560	0.163	0.437
NSA	0.503	0.624	0.432	0.437	0.356	0.307	0.341	0.905	0.550	0.232	0.469

表2 | 我们的NSA与基线在LongBench上的性能比较，包括单文档QA、多文档QA、合成和代码任务类别的子集。NSA的表现超过了大多数基线，包括全注意力。

跨越多样的稀疏注意力范式，包括KV缓存驱逐、查询感知选择和精确的top-n稀疏选择。

对于一般评估，大多数样本的长度在稀疏注意力基线的局部上下文窗口内，这些方法实际上等同于全注意力。因此，我们仅在此设置中呈现NSA与全注意力基线之间的比较结果。在长上下文评估中，我们对所有基线方法进行比较，所有稀疏注意力方法的稀疏性设置相同，以确保公平比较。对于需要长文本监督微调的思维链推理评估，我们将比较限制在全注意力，因为稀疏注意力基线不支持训练。

4.3. 性能比较

一般评估。我们在一套全面的基准测试中评估了预训练的NSA和全注意力基线，这些基准测试涵盖了知识、推理和编码能力，包括MMLU (Hendrycks等, 2020)、MMLU-PRO (Wang等, 2024)、CMMLU (Li等, 2023)、BBH (Suzgun等, 2022)、GSM8K (Cobbe等, 2021)、MATH (Hendrycks等, 2020)、DROP (Dua等, 2019)、MBPP (Austin等, 2021) 和HumanEval (Chen等, 2021)。结果如表1所示。尽管存在稀疏性，NSA在整体性能上表现优越，在9个指标中有7个超过了所有基线，包括全注意力。这表明，尽管NSA可能未能充分利用其在较短序列上的效率优势，但其表现依然强劲。值得注意的是，NSA在与推理相关的基准测试中表现出显著的提升 (DROP: +0.042, GSM8K: +0.034)，这表明我们的预训练有助于模型发展专门的注意力机制。这种稀疏注意力预训练机制迫使模型专注于最重要的信息，可能通过过滤掉无关注意力路径中的噪声来增强性能。不同评估中的一致表现也验证了NSA作为通用架构的稳健性。

长上下文评估。如图5所示，NSA在64k上下文的针在干草堆中 (Kamradt, 2023) 测试中实现了完美的检索准确率。这一性能源于我们的分层稀疏注意力设计，该设计结合了压缩令牌以实现高效的全局上下文扫描，以及选择令牌以实现精确的局部信息检索。粗粒度压缩以低计算成本识别相关上下文块，而对选定令牌的令牌级注意力确保了关键细粒度信息的保留。这一设计使NSA能够同时保持全局意识和局部精度。

我们还评估 NSA o n LongBench (Bai 等, 2023) 对抗最先进的 t 稀疏注意力

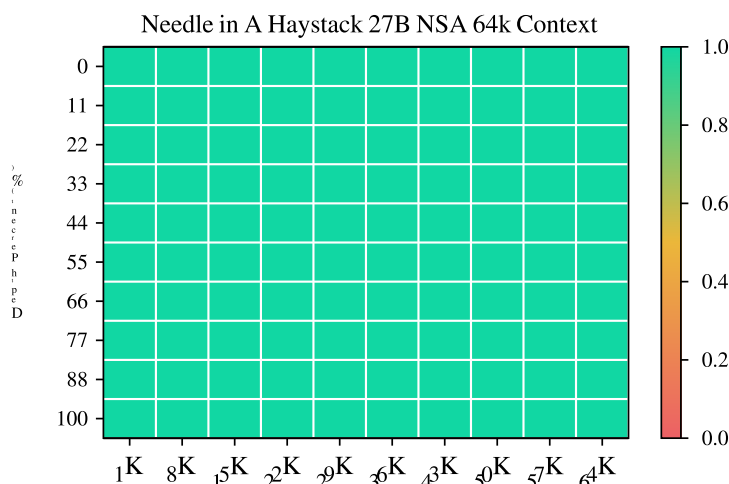


图5 | 在64k上下文长度下，各上下文位置的针在干草堆中检索准确性。NSA通过其分层稀疏注意力设计实现了完美的准确性。

方法和全注意力基线。为了确保一致的稀疏性，我们将所有稀疏注意力基线中每个查询激活的令牌设置为2560个令牌，这对应于在处理32k序列长度时NSA激活的平均令牌数。根据Stream-LLM (Xiao等, 2023)，这个令牌预算包括前128个令牌和512个局部令牌。由于某些子集在所有模型中的得分较低，我们从LongBench中排除了它们，这可能无法提供有意义的比较。如表2所示，NSA实现了最高的平均得分0.469，超越了所有基线（+0.032高于全注意力和+0.046高于精确顶级）。这一改进源于两个关键创新：（1）我们本地的稀疏注意力设计，使得在预训练期间能够对稀疏模式进行端到端优化，促进了稀疏注意力模块与其他模型组件之间的同步适应；（2）层次稀疏注意力机制在局部和全局信息处理之间实现了平衡。

值得注意的是，NSA在需要对长上下文进行复杂推理的任务上表现出色，在多跳问答任务（HPQ和2Wiki）上实现了相较于全注意力的+0.087和+0.051的提升，超越了代码理解基线的表现（LCC: +0.069），并在段落检索（PassR-en: +0.075）上优于其他方法。这些结果验证了NSA处理多样化长上下文挑战的能力，其原生预训练的稀疏注意力在学习任务最优模式方面提供了额外的好处。

链式思维推理评估。为了评估NSA与先进下游训练范式的兼容性，我们研究其通过后训练获取链式思维数学推理能力的的能力。鉴于强化学习在小规模模型上的有限效果，我们采用来自DeepSeek-R1的知识蒸馏，使用10B个32k长度的数学推理轨迹进行监督微调（SFT）。这产生了两个可比较的模型：全注意力-R（全注意力基线）和NSA-R（我们的稀疏变体）。我们在具有挑战性的美国邀请数学考试（AIME 24）基准上评估这两个模型。我们使用0.7的采样温度和0.95的top-p值为每个问题生成16个响应，并获得平均分数。为了验证推理深度的影响，我们进行了两种生成上下文限制的实验：8k和16k个token，测量扩展推理链是否提高准确性。模型预测的示例比较见附录A。



Generation Token Limit	8192	16384
Full Attention-R	0.046	0.092
NSA-R	0.121	0.146

表3 | AIME 基于指令的评估在监督微调后进行。我们的 NSA-R 在 8k 和 16k 序列长度下表现优于 Full Attention-R。

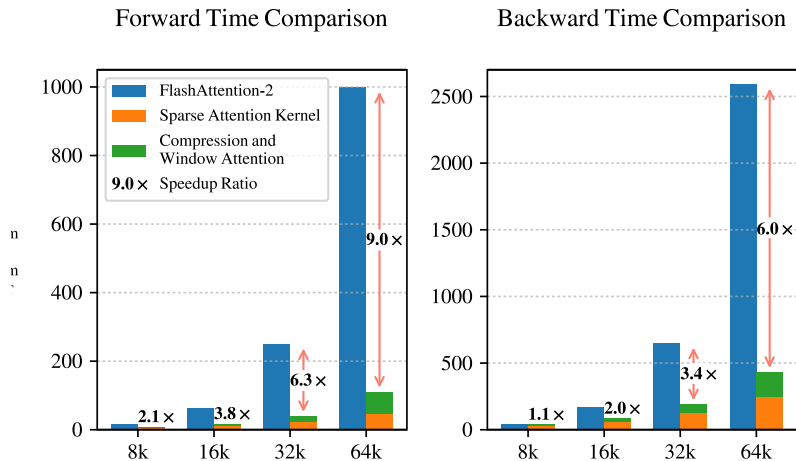


图6 | Triton基础的NSA内核与Triton基础的FlashAttention-2内核的比较。我们的实现显著降低了所有上下文长度的延迟，随着输入长度的增加，改进变得更加明显。

如表3所示，在8k上下文设置下，NSA-R的准确性显著高于Full Attention-R (+0.075)，这种优势在16k上下文中仍然存在 (+0.054)。这些结果验证了原生稀疏注意力的两个关键好处：（1）预训练的稀疏注意力模式能够有效捕捉对复杂数学推导至关重要的长程逻辑依赖；（2）我们架构的硬件对齐设计保持了足够的上下文密度，以支持不断增长的推理深度而不会导致灾难性遗忘。在不同上下文长度下的一致性超越确认了稀疏注意力在原生集成到训练管道时对高级推理任务的可行性。

5. 效率分析

我们在一个8-GPU A100系统上评估NSA相对于全注意力的计算效率。在效率分析中，我们还将模型配置为GQA组 $g = 4$ ，每组头数 $h = 16$ ，查询/键维度 $d_k = 192$ ，以及值维度 $d_v = 128$ 。按照第4节中的相同设置，我们将NSA压缩块大小 $l = 32$ ，滑动步幅 $d = 16$ ，选择的块大小 $l' = 64$ ，选择的块数量 $n = 16$ ，以及滑动窗口大小 $w = 512$ 。

5.1. 训练速度

我们比较了基于Triton的NSA注意力和全注意力的实现与基于Triton的FlashAttention-2，以确保在相同后端下进行公平的速度比较。如图6所示，随着上下文长度的增加，我们的NSA实现了逐渐更大的加速，在64k上下文长度下，前向加速达到9.0x，后向加速达到6.0x。值得注意的是，随着序列的延长，速度优势变得更加明显。这种加速源于我们的

Context Length	8192	16384	32768	65536
Full Attention	8192	16384	32768	65536
NSA	2048	2560	3584	5632
Expected Speedup	4×	6.4×	9.1×	11.6×

表4 | 解码过程中每次注意力操作的内存访问量（以等效的标记数量计）。由于解码的算术强度低且受内存限制，预期的加速大约与内存访问量成线性关系。

硬件对齐的算法设计以最大化稀疏注意力架构的效率：(1) 块状内存访问模式通过合并加载最大化张量核心的利用率，(2) 内核中的精细循环调度消除了冗余的KV传输。

5.2. 解码速度

注意力的解码速度主要受内存访问瓶颈的影响，这与KV缓存加载的数量密切相关。在每个解码步骤中，我们的NSA最多只需要加载 $\lfloor \frac{s-1}{d} \rfloor$ 个压缩标记、 nl' 个选定标记和 w 个邻近标记，其中 s 是缓存的序列长度。如表4所示，我们的方法在解码长度增加时显著减少了延迟，在64k上下文长度时实现了高达11.6×的加速。这种内存访问效率的优势在更长的序列中也会得到增强。

6. 讨论

在本节中，我们反思了NSA的发展过程，并讨论了从我们对不同稀疏注意力策略的探索中获得的关键见解。虽然我们的方法展示了有希望的结果，但理解与替代策略所遇到的挑战并分析注意力模式为未来的研究方向提供了宝贵的背景。我们首先检查了激励我们设计选择的替代令牌选择策略的挑战，随后是提供注意力分布模式见解的可视化。

6.1. 替代代币选择策略的挑战

在设计NSA之前，我们探索了将现有的稀疏注意力方法适应于训练阶段。然而，这些尝试遇到了各种挑战，促使我们设计了一种不同的稀疏注意力架构：

基于关键聚类的策略。我们研究了基于聚类的策略，如 ClusterKV (Liu et al., 2024)。这些方法将来自同一聚类的键和值存储在连续的内存区域中。虽然在理论上可行用于训练和推理，但它们面临三个重大挑战：(1) 动态聚类机制引入的非平凡计算开销；(2) 由于跨聚类不平衡而加剧的操作优化困难，特别是在专家混合 (MoE) 系统中，偏斜的专家并行 (EP) 组执行时间导致持续的负载不平衡；(3) 由于需要强制定期重新聚类和块顺序训练协议而产生的实施限制。这些综合因素造成了显著的瓶颈，极大地限制了它们在实际部署中的有效性。

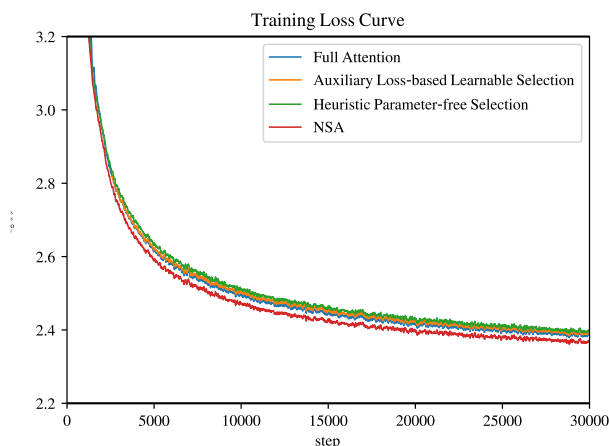


图7 | 比较在一个3B参数模型上使用全注意力和不同的令牌选择策略的训练损失。我们的NSA表现更好。

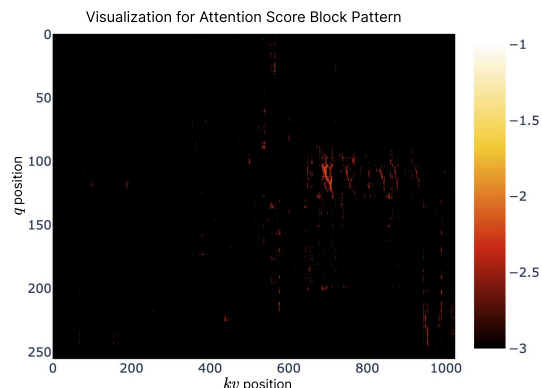


图8 | 全注意力变换器上注意力图的可视化。浅色区域表示较高的注意力值。如图所示，注意力分数呈现块状聚类分布。

其他块状选择策略。我们还考虑了与NSA不同的块状关键值选择策略，例如Quest (Tang等, 2024) 和InfLLM (Xiao等, 2024)。这些方法依赖于为每个块计算重要性分数，并根据它们与 q_t 的相似性选择前- n 个块。然而，现有方法面临两个关键问题：（1）由于选择操作是不可微分的，基于神经网络的重要性分数计算依赖于辅助损失，这增加了操作员的开销，并且往往会降低模型性能；（2）无启发式参数的重要性分数计算策略的召回率较低，导致性能不佳。我们在一个具有相似架构的3B参数模型上评估这两种方法，并将它们的损失曲线与NSA和全注意力进行比较。对于基于辅助损失的选择方法，我们为每个块引入额外的查询和代表性关键，以估计块的重要性分数。这些分数由原始查询和每个块内的关键之间的平均注意力分数进行监督。对于无启发式参数的选择方法，遵循Quest的策略，我们使用查询与关键块的坐标最小-最大值之间的乘积进行直接选择，而不引入额外的参数。我们还探索了一种冷启动训练方法，在过渡到启发式块状选择之前，前1000步应用全注意力。如图7所示，这两种方法的损失均表现不佳。



6.2. 可视化

为了探索变换器注意力分布中的潜在模式并寻求设计灵感，我们在图8中可视化了我们预训练的27B全注意力模型的注意力图。可视化揭示了有趣的模式，其中注意力分数往往表现出块状聚类特征，附近的键通常显示出相似的注意力分数。这一观察激发了我们对NSA的设计，表明基于空间连续性选择关键块可能是一种有前景的方法。块状聚类现象表明，序列中相邻的标记可能与查询标记共享某些语义关系，尽管这些关系的确切性质需要进一步研究。这一观察促使我们探索一种稀疏注意力机制，该机制在连续的标记块上操作，而不是单个标记，旨在提高计算效率并保持高注意力模式。

7. 相关工作

我们回顾了通过稀疏注意力提高注意力计算效率的现有方法。这些方法可以根据其核心策略大致分为三组：（1）固定稀疏模式，（2）动态令牌修剪，以及（3）查询感知选择。我们介绍了每个类别中的几个代表性工作。

7.1. 固定稀疏模式

SlidingWindow 是一种常用的方法，它允许查询仅在固定窗口内计算注意力。StreamingLLM (Xiao 等, 2023) 通过维护上下文的关键部分来解决处理长文本流的挑战：一个注意力接收器（早期标记）和一个局部上下文窗口。虽然这些方法有效地降低了内存和计算成本，但它们忽略上下文的严格模式限制了它们在需要全面理解上下文的任务上的表现。

7.2. 动态令牌修剪

H2O (Zhang et al., 2023b) 实现了一种自适应方法，以减少解码过程中 KV-cache 内存的使用。该方法根据注意力分数动态驱逐被认为对未来预测不太重要的标记，基于它们最近的效用。SnapKV (Li et al., 2024) 还引入了一种标记修剪策略，通过选择性保留最关键的特征来减少 KV 缓存，从而实现高效的内存使用。SnapKV 通过注意力权重分析和预填充期间的投票来识别重要特征，然后将选定的压缩特征与最近的上下文结合来更新 KV 缓存，以保持提示的一致性。

7.3. 查询感知选择

Quest (唐等, 2024) 采用块状选择策略，其中每个块的重要性通过查询与关键块的坐标-wise 最小-最大值的乘积来估计。结果分数有助于选择前- n 个重要的键值块以供注意。InfLLM (肖等, 2024) 通过维护注意力接收器、局部上下文和可检索块，将固定模式与检索相结合。该方法从每个块中选择代表性键以估计块的重要性。HashAttention (德赛等, 2024) 将关键令牌识别公式化为推荐问题，通过使用学习的函数将查询和键映射到汉明空间。ClusterKV (刘等, 2024) 通过首先对键进行聚类，然后根据查询-聚类相似性选择最相关的聚类来实现稀疏性。

8. 结论

我们提出了NSA，一种硬件对齐的稀疏注意力架构，用于高效的长上下文建模。通过在可训练架构中集成分层令牌压缩和块状令牌选择，我们的架构实现了加速训练和推理，同时保持了全注意力性能。NSA通过展示一般基准性能与全注意力基线相匹配、在长上下文评估中超越建模能力以及增强推理能力，推动了最先进技术的发展，所有这些都伴随着可测量的计算延迟减少和显著的加速。

参考文献

- J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón 和 S. Sanghai. Gqa: 从多头检查点训练广义多查询变换器模型。arXiv 预印本 arXiv:2305.13245, 2023.
- J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le 等人。使用大型语言模型进行程序合成。arXiv 预印本 arXiv:2108.07732, 2021。
- Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang, Z. Du, X. Liu, A. Zeng, L. Hou, 等. Longbench: 一种用于长上下文理解的双语多任务基准。arXiv 预印本 arXiv:2308.14508, 2023.
- M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman 等人。评估在代码上训练的大型语言模型。arXiv 预印本 arXiv:2107.03374, 2021.
- Z. Chen, R. Sadhukhan, Z. Ye, Y. Zhou, J. Zhang, N. Nolte, Y. Tian, M. Douze, L. Bottou, Z. Jia 等. M agicpig: 用于高效 LLM 生成的 LSH 采样。arXiv 预印本 arXiv:2410.16179, 2024.
- K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano 等人。训练验证者解决数学文字问题, 2021。网址 <https://arxiv.org/abs/2110.14168>, 2021。
- D. Dai, C. Deng, C. Zhao, R. Xu, H. Gao, D. Chen, J. Li, W. Zeng, X. Yu, Y. Wu 等. Deepseekmoe: 朝着混合专家语言模型的终极专家专业化迈进。arXiv 预印本 arXiv:2401.06066, 2024.
- DeepSeek-AI。Deepseek-v2: 一种强大、经济且高效的专家混合语言模型。2024。网址 <https://arxiv.org/abs/2405.04434>。
- DeepSeek-AI。Deepseek-r1: 通过强化学习激励大型语言模型的推理能力, 2025年。网址 <https://arxiv.org/abs/2501.12948>。
- A. Desai, S. Yang, A. Cuadron, A. Klimovic, M. Zaharia, J. E. Gonzalez, 和 I. Stoica. Hashatten- tion: 语义稀疏性以加快推理。arXiv 预印本 arXiv:2412.14468, 2024.
- D. Dua, Y. Wang, P. Dasigi, G. Stanovsky, S. Singh 和 M. Gardner. Drop: 一项需要对段落进行离散推理的阅读理解基准。arXiv 预印本 arXiv:1903.00161, 2019.
- S. Ge, Y. Zhang, L. Liu, M. Zhang, J. Han, 和 J. Gao. 模型告诉你该丢弃什么: 针对 llms 的自适应 kv 缓存压缩。arXiv 预印本 arXiv:2310.01801, 2023.
- G. T. Google, P. Georgiev, V. I. Lei, R. Burnell, L. Bai, A. Gulati, G. Tanzer, D. Vincent, Z. Pan, S. Wang 等人。Gemini 1.5: 解锁跨越数百万个上下文标记的多模态理解。arXiv 预印本 arXiv:2403.05530, 2024.
- D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, 和 J. Steinhardt. 测量大规模多任务语言理解。arXiv 预印本 arXiv:2009.03300, 2020.
- H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, 和 L. Qiu. LlmLingua: 压缩提示以加速大型语言模型的推理。arXiv 预印本 arXiv:2310.05736, 2023.

H. Jiang, Y. Li, C. Zhang, Q. Wu, X. Luo, S. Ahn, Z. Han, A. H. Abdi, D. Li, C.-Y. Lin 等人。Minference 1.0: 通过动态稀疏注意力加速长上下文 llms 的预填充。arXiv 预印本 arXiv:2407.02490, 2024。G. Kamradt。LLMTest NeedleInAHaystack。GitHub 仓库, 2023。网址 https://github.com/gkamradt/LLMTest_NeedleInAHaystack。访问时间: [在此插入访问日期]。H. Li, Y. Zhang, F. Koto, Y. Yang, H. Zhao, Y. Gong, N. Duan 和 T. Baldwin。Cmmlu: 衡量中文的大规模多任务语言理解。arXiv 预印本 arXiv:2306.09212, 2023。Y. Li, Y. Huang, B. Yang, B. Venkitesh, A. Locatelli, H. Ye, T. Cai, P. Lewis 和 D. Chen。Snapkv: llm 在生成之前知道你在寻找什么。arXiv 预印本 arXiv:2404.14469, 2024。G. Liu, C. Li, J. Zhao, C. Zhang 和 M. Guo。Clusterkv: 在语义空间中操控 llm kv 缓存以实现可回忆的压缩。arXiv 预印本 arXiv:2412.03213, 2024。

J. S. Park, J. C. O' Brien, C. J. Cai, M. R. Morris, P. Liang, 和 M. S. Bernstein。生成代理: 人类行为的互动仿真体。在 S. Follmer, J. Han, J. Steimle, 和 N. H. Riche 编辑的《第36届年度 ACM 用户界面软件与技术研讨会论文集》, UIST 2023, 旧金山, 加利福尼亚州, 美国, 2023年10月29日-2023年11月1日, 第2:1-2:22页。ACM, 2023。

B. Peng, J. Quesnelle, H. Fan, 和 E. Shippole。Yarn: 大型语言模型的高效上下文窗口扩展。在 ICLR。OpenReview.net, 2024。

N. Shazeer。快速变换器解码: 你只需要一个写头。CoRR, abs/1911.02150, 2019。

M. Suzgun, N. Scales, N. Schärli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. V. Le, E. H. Chi, D. Zhou 等人。挑战大基准任务以及思维链是否能解决这些任务。arXiv 预印本 arXiv:2210.09261, 2022。

J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, 和 S. Han。Quest: 查询感知稀疏性用于高效的长上下文 LLM 推理。arXiv 预印本 arXiv:2406.10774, 2024。

P. Tillet, H.-T. Kung 和 D. Cox。Triton: 一种用于平铺神经网络计算的中间语言和编译器。在第三届 ACM SIGPLAN 国际机器学习与编程语言研讨会论文集中, 页码 10-19, 2019。

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, 和 I. Polosukhin。注意力是你所需要的一切。神经信息处理系统进展, 2017。

Y. Wang, X. Ma, G. Zhang, Y. Ni, A. Chandra, S. Guo, W. Ren, A. Arulraj, X. He, Z. Jiang 等。Mmlu-pro: 一个更强大且具有挑战性的多任务语言理解基准。arXiv 预印本 arXiv:2406.01574, 2024。

J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou 等。连锁思维提示在大型语言模型中引发推理。神经信息处理系统进展, 35:24824-24837, 2022。

C. Xiao, P. Zhang, X. Han, G. Xiao, Y. Lin, Z. Zhang, Z. Liu, 和 M. Sun。Inflm: 无需训练的长上下文外推, 针对具有高效上下文记忆的 llms。在第三十八届神经信息处理系统年会, 2024。

G. Xiao, Y. Tian, B. Chen, S. Han, 和 M. Lewis. 高效的流式语言模型与注意力汇. arXiv 预印本 arXiv:2309.17453, 2023. M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, 等. 大鸟: 用于更长序列的变换器. 神经信息处理系统进展, 33:17283–17297, 2020. E. Zelikman, Y. Wu, J. Mu, 和 N. D. Goodman. 星: 用推理引导推理. 在 S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, 和 A. Oh, 编辑, 神经信息处理系统进展 35: 2022 年神经信息处理系统年会, NeurIPS 2022, 美国路易斯安那州新奥尔良, 2022 年 11 月 28 日 – 12 月 9 日, 2022.

F. Zhang, B. Chen, Y. Zhang, J. Keung, J. Liu, D. Zan, Y. Mao, J. Lou, 和 W. Chen. Repocoder: 通过迭代检索和生成实现的库级代码补全. 在 H. Bouamor, J. Pino, 和 K. Bali, 编辑, 2023 年自然语言处理实证方法会议论文集, EMNLP 2023, 新加坡, 2023 年 12 月 6 日至 10 日, 页码 2471–2484. 计算语言学协会, 2023a.

K. Zhang, J. Li, G. Li, X. Shi, 和 Z. Jin. Codeagent: 通过工具集成代理系统增强代码生成, 以应对真实世界的仓库级编码挑战. 在 L. Ku, A. Martins, 和 V. Srikumar, 编辑, 计算语言学协会第 62 届年会论文集 (第 1 卷: 长篇论文), ACL 2024, 泰国曼谷, 2024 年 8 月 11 日至 16 日, 页码 13643–13658.

Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett 等. H2o: 用于大型语言模型高效生成推理的重击手预言机. 神经信息处理系统进展, 36:34661–34710, 2023b.

Z. Zhou, C. Li, X. Chen, S. Wang, Y. Chao, Z. Li, H. Wang, R. An, Q. Shi, Z. Tan 等. Llm × mapreduce: 使用大型语言模型简化长序列处理. arXiv 预印本 arXiv:2410.09342, 2024.

A. AIME结果示例

提示:

“|用户|设 x 、 y 和 z 为满足以下方程组的正实数:

翻译文本:

$$\log_2 \left(\frac{x}{yz} \right) = \frac{1}{2}$$

$$\log_2 \left(\frac{y}{xz} \right) = \frac{1}{3}$$

$$\log_2 \left(\frac{z}{xy} \right) = \frac{1}{4}$$

然后 $|\log_2(x^4y^3z^2)|$ 的值是 $\frac{m}{n}$, 其中 m 和 n 是互质的正整数。求 $m+n$ 。请逐步推理, 并将最终答案放在“|助手|”中。 \square

我们的结果:

<思考>...</思考> (2275个标记)

给定方程组:

$$1. \log_2 \left(\frac{x}{yz} \right) = \frac{1}{2} \quad 2. \log_2 \left(\frac{y}{xz} \right) = \frac{1}{3} \quad 3. \log_2 \left(\frac{z}{xy} \right) = \frac{1}{4}$$

我们将这些对数方程转换为它们的指数形式:

$$1. \frac{x}{yz} = \sqrt{2} \quad 2. \frac{y}{xz} = 2^{1/3} \quad 3. \frac{z}{xy} = 2^{1/4}$$

将每个变量表示为 x 、 y 和 z :

$$1. x = \sqrt{2} \cdot yz \quad 2. y = 2^{1/3} \cdot xz \quad 3. z = 2^{1/4} \cdot xy$$

将这些表达式相互代入, 我们发现:

- 从 $x = \sqrt{2} \cdot yz$ 和 $y = 2^{1/3} \cdot xz$, 我们得到 $x = \sqrt{2} \cdot (2^{1/3} \cdot xz) \cdot z$, 导致 $z = 2^{-5/12}$ 。
- 将 z 代入 y 的方程, 我们得到 $y = 2^{-3/8}$ 。
- 将 z 和 y 代入 x 的方程, 我们得到 $x = 2^{-7/24}$ 。

接下来, 我们计算 $x^4y^3z^2$:

$$- x^4 = (2^{-7/24})^4 = 2^{-7/6} \quad - y^3 = (2^{-3/8})^3 = 2^{-9/8} \quad - z^2 = (2^{-5/12})^2 = 2^{-5/6}$$

将这些相乘:

$$x^4y^3z^2 = 2^{-7/6} \cdot 2^{-9/8} \cdot 2^{-5/6} = 2^{-75/24} = 2^{-25/8}$$

取以2为底的对数:

$$\log_2(x^4y^3z^2) = -25/8$$

绝对值是:

$$|\log_2(x^4y^3z^2)| = 25/8$$

因此, $m = 25$ 和 $n = 8$, 得出 $m + n = 33$ 。

33

基线结果:

〈思考〉.../〈思考〉 (9392 代币)

给定对数方程组:

$$\log_2 \left(\frac{x}{yz} \right) = \frac{1}{2}$$

$$\log_2 \left(\frac{y}{xz} \right) = \frac{1}{3}$$

$$\log_2 \left(\frac{z}{xy} \right) = \frac{1}{4}$$

我们将这些方程转换为指数形式:

$$\sqrt{2} = \frac{x}{yz} \implies x = \sqrt{2} \cdot yz$$

$$2^{\frac{1}{3}} = \frac{y}{xz} \implies y = 2^{\frac{1}{3}} \cdot xz$$

$$2^{\frac{1}{4}} = \frac{z}{xy} \implies z = 2^{\frac{1}{4}} \cdot xy$$

将这些相互代入, 我们发现:

$$1. x = \sqrt{2} \cdot yz \quad 2. y = 2^{\frac{1}{3}} \cdot xz \quad 3. z = 2^{\frac{1}{4}} \cdot xy$$

通过逐步解决这些方程, 我们发现:

$$-z = 2^{-5/12} \cdot xy = 2^{-2/3} \cdot y = 2^{-1/24} \cdot x = 2^{1/24}$$

然而, 在验证第三个方程时, 我们意识到该系统可能没有解。因此, 我们使用矩阵求逆来求解该系统:

$$\begin{cases} \log_2(x) - \log_2(y) - \log_2(z) = \frac{1}{2} \\ \log_2(y) - \log_2(x) - \log_2(z) = \frac{1}{3} \\ \log_2(z) - \log_2(x) - \log_2(y) = \frac{1}{4} \end{cases}$$

让 $A = \log_2(x)$, $B = \log_2(y)$, $C = \log_2(z)$ 。系统变为:

$$\begin{cases} A - B - C = \frac{1}{2} \\ B - A - C = \frac{1}{3} \\ C - A - B = \frac{1}{4} \end{cases}$$

我们使用克拉默法则和矩阵求逆来解决这个系统。矩阵的行列式为 -2，解为：

$$A = \frac{61}{48}, \quad B = \frac{13}{16}, \quad C = -\frac{1}{6}$$

因此，我们有：

$$x = 2^{61/48}, \quad y = 2^{13/16}, \quad z = 2^{-1/6}$$

我们需要找到 $|\log_2(x^4y^3z^2)|$ ：

$$4 \log_2 x + 3 \log_2 y + 2 \log_2 z = 4 \cdot \frac{61}{48} + 3 \cdot \frac{13}{16} + 2 \cdot \left(-\frac{1}{6}\right)$$

转换为一个共同的分母（48）：

$$4 \cdot \frac{61}{48} = \frac{244}{48}, \quad 3 \cdot \frac{13}{16} = \frac{117}{48}, \quad 2 \cdot \left(-\frac{1}{6}\right) = -\frac{16}{48}$$

将这些相加：

$$\frac{244}{48} + \frac{117}{48} - \frac{16}{48} = \frac{345}{48} = \frac{115}{16}$$

分数 $\frac{115}{16}$ 是最简形式，因此 $m = 115$ 和 $n = 16$ 。因此， $m + n = 115 + 16 = 131$ 。

131

提示：

“|用户| 实数 x 和 y 与 $x, y > 1$ 满足 $\log_x(y^x) = \log_y(x^{4y}) = 10$ 。 xy 的值是多少？请逐步推理，并将你的最终答案放在 . |助手|”

□

我们的结果：

<思考>...</思考> (15147个标记)

给定方程：

$$\begin{aligned} 1. \log_x(y^x) &= 10 \\ 2. \log_y(x^{4y}) &= 10 \end{aligned}$$

我们首先对两个方程应用对数幂法则。

对于第一个方程：

$$\log_x(y^x) = x \cdot \log_x(y) = 10$$

让 $z = \log_y(x)$ 。然后， $\log_x(y) = \frac{1}{z}$ 。将其代入方程，我们得到：

$$x \cdot \frac{1}{z} = 10 \implies x = 10z$$

对于第二个方程：

$$\log_y(x^{4y}) = y \cdot \log_y(x^4) = 10$$

使用幂法则，我们得到：

$$y \cdot 4 \log_y(x) = 10 \implies 4y \log_y(x) = 10$$

由于 $\log_y(x) = z$ ，这简化为：

$$4yz = 10 \implies yz = \frac{5}{2}$$

我们现在有两个表达式：

$$\begin{aligned} 1. x &= 10z \\ 2. y &= \frac{5/2}{z} \end{aligned}$$

将 $x = 10z$ 和 $y = \frac{5/2}{z}$ 代入乘积 xy ，我们得到：

$$xy = (10z) \cdot \left(\frac{5/2}{z}\right) = 10z \cdot \frac{5}{2z} = 10 \cdot \frac{5}{2} = 25$$

因此， xy 的值是 25。

基线结果：

<思考>...</思考> (16223个标记)

给定实数 x 和 y 满足方程 $\log_x(y^x) = 10$ 和 $\log_y(x^{4y}) = 10$ ，我们需要找到 xy 的值。

首先，我们使用换底公式将对数方程转换为指数形式。

$$1. \log_x(y^x) = 10$$

$$x \log_x(y) = 10$$

使用换底公式, $\log_x(y) = \frac{\ln y}{\ln x}$ 。由于 $\log_y x = \frac{9}{4}$, 我们有 $\log_x y = \frac{4}{9}$ 。因此,

$$x \cdot \frac{4}{9} = 10 \implies x = \frac{10 \cdot 9}{4} = \frac{90}{4} = \frac{45}{2}$$

$$2. \log_y(x^4 y) = 10$$

$$4 \log_y x + 1 = 10 \implies 4 \log_y x = 9 \implies \log_y x = \frac{9}{4}$$

再次使用换底公式, $\log_x y = \frac{4}{9}$ 。

接下来, 我们用彼此的关系表达 x 和 y :

$$x = \frac{45}{2} \quad \text{and} \quad y = \left(\frac{45}{2}\right)^{4/9}$$

我们接着计算乘积 xy :

$$xy = \left(\frac{45}{2}\right) \cdot \left(\frac{45}{2}\right)^{4/9} = \left(\frac{45}{2}\right)^{1+4/9} = \left(\frac{45}{2}\right)^{13/9}$$

然而, 在重新评估并采用另一种方法, 将 $x = 2^a$ 和 $y = 2^b$ 设置后, 我们发现:

$$\log_x(y^x) = 10 \implies \frac{x \log_x y}{\log_x y} = 10 \implies x \cdot \frac{4}{9} = 10 \implies x = \frac{45}{2}$$

$$\log_y(x^4 y) = 10 \implies \frac{4 \log_y x + 1}{\log_y x} = 10 \implies 4 \cdot \frac{9}{4} + 1 = 10 \implies 9 + 1 = 10 \quad \text{consistent}$$

最后, 我们发现乘积 xy 简化为:

$$xy = \left(\frac{45}{2}\right)$$

本文由 AINLP 公众号整理翻译，更多 LLM 资源请扫码关注!

AINLP

我爱自然语言处理

一个有趣有AI的自然语言处理社区



长按扫码关注我们