

# 原生稀疏注意:硬件对齐和原生可训练稀疏注意

Jingyang Yuan<sup>\*1,2</sup>, 高华佐1, 代大麦1, 罗俊宇2, 赵亮1, 张正彦1, 谢振达1, 魏云霞1, 王lean 1, 肖志平3, 王玉青1, 阮冲1, 张明2, 梁文峰1,

曾旺定1

<sup>1</sup>DeepSeek-AI

2北京大学多媒体信息处理重点实验室, 北大-安科法学硕士实验室

3华盛顿大学

{yuanjy, mzhang\_cs}@ppku.edu.cn, {zengwangding, wenfeng.liang}@deepseek.com

## 摘要

长上下文建模对于下一代语言模型至关重要，然而标准注意机制的高计算成本带来了重大的计算挑战。稀疏注意为在保持模型能力的同时提高效率提供了一个有希望的方向。我们提出了NSA，一种原生可训练的稀疏注意机制，它将算法创新与硬件对齐优化相结合，以实现高效的长上下文建模。NSA采用动态分层稀疏策略，将粗粒度令牌压缩与细粒度令牌选择相结合，以保持全局上下文感知和局部精度。我们的方法通过两个关键创新推进了稀疏注意力设计:(1)我们通过算法强度平衡的算法设计实现了实质性的加速，并对现代硬件进行了实现优化。(2)我们实现了端到端训练，在不牺牲模型性能的情况下减少了预训练计算。如图1所示，实验表明，用NSA预训练的模型在一般基准测试、长上下文任务和基于指令的推理中保持或超过全注意力模型。同时，在解码、前向传播和后向传播的64k长度序列上，NSA在全注意力上实现了显著的加速，验证了其在整个模型生命周期中的效率。

## 1. 介绍

研究界越来越多地认识到，长上下文建模是下一代大型语言模型的关键能力，受到各种现实世界应用的驱动，包括深度推理(DeepSeek-AI, 2025;Zelikman等人, 2022)，库级代码生成(张, 2023a;张等人)和多回合自主代理系统(Park等人, 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(谷歌等人, 2024)，使模型能够处理整个代码库，冗长的文档，在数千个令牌上保持连贯的多回合对话，并在远程依赖关系中执行复杂的推理。然而，随着序列长度的增加，vanilla Attention (Vaswani et al., 2017)机制的高复杂性(Za-heer et al., 2020)会成为关键的延迟瓶颈。理论估计表明，注意力

---

\*Contribution during internship at DeepSeek-AI.

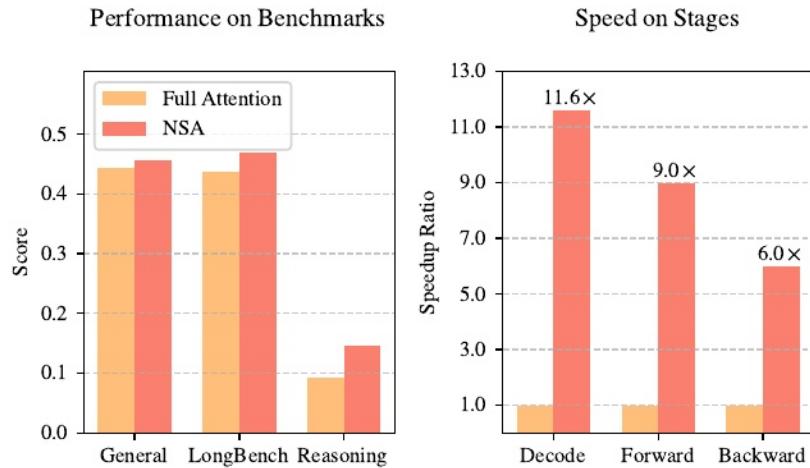


图1 |全注意力模型和我们的NSA模型的性能和效率比较。左图:尽管很稀疏，但NSA在一般基准、长上下文任务和推理评估中平均超过了完全注意力基线。右图:对于64k长度的序列处理，与全注意力相比，NSA在解码、前向传播和后向传播的所有阶段都实现了显著的计算速度提升。

在解码64k长度上下文时，使用softmax架构的计算占总延迟的70-80%，这强调了对更有效的注意力机制的迫切需要。

高效长上下文建模的一种自然方法是利用softmax注意力的固有稀疏性(Ge等人, 2023;Jiang et al., 2023)，其中选择性地计算关键查询键对可以在保持性能的同时显著降低计算开销。最近的进展通过不同的策略证明了这种潜力:KV-cache清除方法(Li et al., 2024;张等, 2023b;Zhou et al., 2024)，分段KV-cache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)。尽管有这些很有前景的策略，但现有的稀疏注意力方法在实际部署中往往存在不足。许多方法无法实现与其理论收益相当的加速;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来充分利用注意力的稀疏模式。

为了解决这些限制，部署有效的稀疏注意力必须解决两个关键挑战:(1)**硬件对齐的推理加速**:将理论计算的减少转化为实际速度的提高，需要在预填充和解码阶段进行硬件友好的算法设计，以缓解内存访问和硬件调度瓶颈;(2)**训练感知算法设计**:使用可训练算子进行端到端计算，在保持模型性能的同时降低训练成本。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，现有的方法仍然表现出明显的差距。

为了实现更有效和高效的稀疏注意，我们提出了NSA，一种集成了分层令牌建模的本地可训练稀疏注意架构。如图2所示，NSA通过将键和值组织到时间块中，并通过三个注意路径来处理它们，从而减少了每次查询的计算:压缩的粗粒度令牌、选择性保留的细粒度令牌和用于本地上下文信息的滑动窗口。然后我们实现了专门的内核，以最大限度地提高其实际效率。NSA介绍了

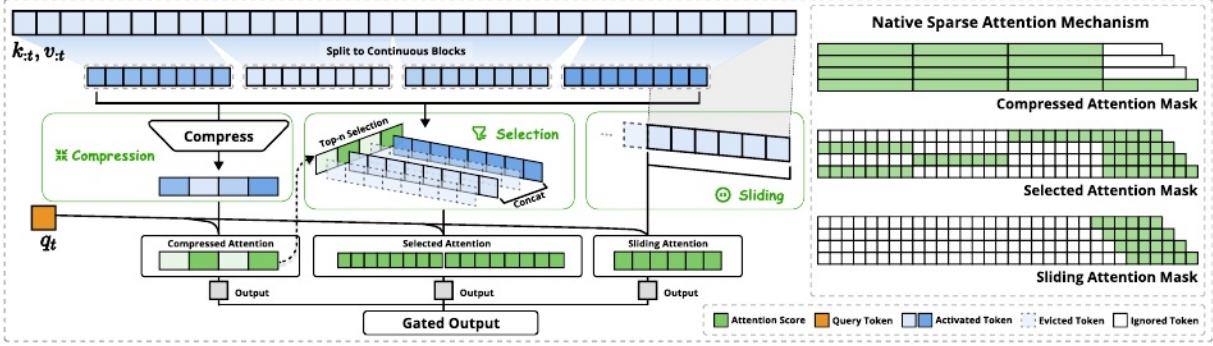


图2 | NSA架构概述。左图:框架通过三个并行的注意分支处理输入序列:对于给定的查询，前面的键和值被处理为粗粒度模式的压缩注意，重要令牌块的选择注意，以及局部上下文的滑动注意。右:每个分支产生的不同注意模式的可视化。绿色区域表示需要计算注意力得分的区域，白色区域表示可以跳过的区域。

核心创新对应上述关键要求:(1)硬件对齐系统:优化块稀疏关注张量核心利用率和内存访问，确保均衡的算术强度。(2)训练感知设计:通过高效算法和逆向运算实现稳定的端到端训练。这种优化使NSA能够同时支持高效部署和端到端培训。

我们通过对真实语言语料库的综合实验来评估NSA。在使用260B代币的27b参数变压器主干上进行预训练，我们评估了NSA在一般语言评估、长上下文评估和思维链推理评估方面的性能。我们进一步比较了A100 gpu上的内核速度与优化的Triton (Tillet et al., 2019)实现。实验结果表明，NSA方法的性能与全注意力基线相当甚至更好，同时优于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，对于更长的序列，加速比增加。这些结果验证了我们的分层稀疏注意设计有效地平衡了模型能力和计算效率。

## 2. 重新思考稀疏注意方法

现代稀疏注意方法在降低变压器模型的理论计算复杂度方面取得了重大进展。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主干，这可能会引入架构偏差，限制它们充分利用稀疏注意力优势的能力。在介绍我们的原生稀疏架构之前，我们通过两个关键镜头系统地分析了这些限制。

### 2.1. The Illusion of Efficient Inference

尽管在注意力计算中实现了稀疏性，但许多方法未能实现相应的推理延迟减少，主要是由于两个挑战:

相位限制稀疏性(**Phase-Restricted Sparsity**)。H2O (Zhang et al., 2023b)等方法应用稀疏性

在自回归解码期间，而在预填充期间需要计算密集的预处理(例如注意图计算，索引构建)。相比之下，MIInference (Jiang et al., 2024)等方法只关注预填充稀疏性。这些方法无法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全注意相当。阶段专门化降低了这些方法在预填充为主的工作负载(如书籍摘要和代码完成)或解码为主的工作负载(如长思维链(Wei et al., 2022)推理)中的加速能力。

**与高级注意力架构不兼容。**一些稀疏注意方法无法适应现代解码高效架构，如多查询注意(MQA) (Shazeer, 2019)和分组查询注意(GQA) (Ainslie等, 2023)，这些架构通过在多个查询头之间共享KV，显著降低了解码过程中的内存访问瓶颈。例如，在Quest (Tang et al., 2024)等方法中，每个注意头独立选择其KV-cache子集。尽管它在多头注意(MHA)模型中展示了一致的计算稀疏性和内存访问稀疏性，但它在基于GQA等架构的模型中呈现了不同的场景，其中KV-cache的内存访问量对应于同一GQA组中所有查询头的选择的联合。这种体系结构特征意味着，虽然这些方法可以减少计算操作，但所需的KV-cache内存访问仍然相对较高。这种限制迫使我们做出了一个关键的选择：虽然一些稀疏关注方法减少了计算，但它们的分散内存访问模式与来自高级架构的高效内存访问设计相冲突。

出现这些限制是因为许多现有的稀疏注意方法侧重于减少KV-cache或减少理论计算量，但难以在高级框架或后端实现显着的延迟减少。这促使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分析纯推理方法的两个关键见解：(1)**性能退化**：应用稀疏性事后强迫模型偏离其预训练的优化轨迹。正如Chen等人(2024)所证明的那样，前20%的注意力只能覆盖总注意力得分的70%，这使得预训练模型中的检索头等结构在推理过程中容易被修剪。(2)**训练效率需求**：长序列训练的高效处理对于现代LLM的发展至关重要。这既包括在更长的文档上进行预训练以增强模型能力，也包括随后的适应阶段，如长上下文微调和强化学习。然而，现有的稀疏注意方法主要针对推理，这使得训练中的计算挑战在很大程度上没有得到解决。这一限制阻碍了通过高效训练开发更有能力的长上下文模型。此外，将现有的稀疏注意力用于训练的努力也暴露了挑战：

**非可训练组件。**像ClusterKV (Liu et al., 2024)(包括k-means聚类)和MagicPIG (Chen等人, 2024)(包括基于simhash的选择)这样的方法中的离散操作在计算图中创建了不连续。这些不可训练的成分在令牌选择过程中阻止了梯度流，限制了模型学习最优稀疏模式的能力。

**低效的反向传播(backpropagation)。**一些理论上可训练的稀疏注意方法存在实际训练效率低下的问题。方法中使用的token - 颗粒选择策略

比如HashAttention (Desai et al., 2024)导致在注意力计算期间需要从KV缓存中加载大量单个令牌。这种非连续内存访问阻碍了FlashAttention等快速注意力技术的有效适应，FlashAttention依赖于连续内存访问和块计算来实现高吞吐量。因此，实现被迫回落到低硬件利用率，显著降低了训练效率。

### 2.3. 原生稀疏性(Native Sparsity)势在必行

这些在推理效率和训练可行性上的限制促使我们从根本上重新设计稀疏注意机制。我们提出了NSA，一个原生稀疏注意力框架，解决了计算效率和训练需求。在接下来的章节中，我们详细介绍了NSA的算法设计和运营商实现。

## 3. 方法论

我们的技术方法跨越了算法设计和内核优化。在下面的小节中，我们首先介绍我们的方法的背景。然后给出了NSA的总体框架和关键算法组成。最后，我们详细介绍了我们的硬件优化内核设计，以最大限度地提高实际效率。

### 3.1. 背景

注意机制广泛用于语言建模，其中每个查询令牌 $\mathbf{q}_t$  computes对所有前面的关键字的相关性得分 $\mathbf{k}_{:t}$ 生成值的加权和 $\mathbf{v}_{:t}$ 。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：

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

其中， $\alpha_{t,i}$  represents为 $\mathbf{q}_t$  and  $\mathbf{k}_i$ 之间的注意权重， $D_K$  is 为键的特征维度。随着序列长度的增加，注意力计算在整体计算成本中越来越占主导地位，这对长上下文处理提出了重大挑战。

**算术强度**是计算操作与内存访问的比率。它本质上决定了硬件上的算法优化。每个GPU都有一个关键的算法强度，由其峰值计算能力和内存带宽决定，以这两个硬件限制的比值计算。对于计算任务，高于该临界阈值的算术强度为计算约束(受GPU FLOPS限制)，低于该临界阈值的算术强度为内存约束(受内存带宽限制)。

具体来说，对于因果自注意机制，在训练和预填充阶段，批处理矩阵乘法和注意力计算表现出很高的算术强度，使得这些阶段在现代加速器上被计算绑定。相比之下，自回归解码会受到内存带宽的限制，因为它在每次前向传递时生成一个令牌，同时需要加载整个键值缓存，导致算术强度较低。这导致了不同的优化目标——减少训练和预填充期间的计算成本，同时减少解码期间的内存访问。

### 3.2. 总体框架

为了利用自然稀疏模式的注意力潜力，我们建议将原始的键值对 $\mathbf{k}_{:T}, \mathbf{v}_{:T}$ 等式(1)替换为更紧凑和信息密集的表示键值对集 $K \boxtimes_T, V \boxtimes_T$ given每个查询 $\mathbf{q}_T$ 。具体来说，我们将优化后的注意力输出正式定义如下：

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

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

其中 $K \boxtimes_T, V \boxtimes_T$ are基于当前查询动态构建 $\mathbf{q}_T$ and上下文记忆 $\mathbf{k}_{:T}, \mathbf{v}_{:T}$ 。我们可以设计各种映射策略来获得 $K \boxtimes_T^C, V \boxtimes_T^C$ 的不同类别，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 $C = \{\text{cmp}, \text{slc}, \text{win}\}$ ，分别表示键和值的压缩、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到。设 $N_T$ denote为重新映射的键/值的总数：

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

我们通过ensuring $N_T \ll T$ 保持高稀疏率。

### 3.3. 算法设计

在本小节中，我们将介绍我们的重新映射策略的设计 $F_K$ and $F_V$ :令牌压缩、令牌选择和滑动窗口。

#### 3.3.1. 令牌压缩

通过将键或值的顺序块聚合到块级表示中，我们获得了捕获整个块信息的压缩键和值。形式上，压缩的键表示被定义为：

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

其中 $L$ is块长度， $D$ is相邻块之间的滑动步幅， $\varphi$ is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。 $K \sim T \text{cmp} \in RDK \times TD - L \boxtimes$ 是由压缩键组成的张量。通常，我们采用 $D < L$ to缓解信息碎片化。一个类似的公式适用于压缩值表示 $V \boxtimes_T^{\text{cmp}}$ 。压缩表示捕获粗粒度的高级语义信息并减少注意力的计算负担。

### 3.3.2. Token选择

只使用压缩的键，值可能会丢失重要的细粒度信息，这促使我们有选择地保留单个键、值。下面我们将描述我们高效的令牌选择机制，该机制以较低的计算开销识别和保留最相关的令牌。

**区块选择(Blockwise Selection)**。我们的选择策略处理空间连续块中的键和值序列，其动机是两个关键因素：硬件效率考虑和注意力得分的固有分布模式。**块选择对于在现代gpu上实现高效计算至关重要**。这是因为与基于随机索引的读取相比，现代GPU架构在连续块访问方面表现出明显更高的吞吐量。此外，块计算可以实现张量核的最佳利用。这种架构特征已经建立了块内存访问和计算作为高性能注意力实现的基本原则，例如FlashAttention的基于块的设计。**分块选择遵循注意力得分固有的分布模式**。先前的研究(Jiang et al., 2024)表明，注意力得分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。

为了实现逐块选择，我们首先将键、值序列划分为选择块。为了确定最重要的块进行注意力计算，我们需要为每个块分配重要性分数。下面我们给出了计算这些块级重要性分数的方法。

**重要性分数计算**。计算块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了中间注意力分数，我们可以利用它来诱导选择块重要性分数，公式为：

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

其中  $\mathbf{p}_t^{\text{cmp}} \in \mathbb{R}^{L' \times H}$  是 QTand 压缩键之间的注意分数  $K \sim T$ 。让  $L'$  表示选择块大小。

当压缩块和选择块共享相同的阻塞方案，即  $L' = L = D$  时，我们可以直接通过  $\mathbf{p}_t^{\text{slc}} = \mathbf{p}_t^{\text{cmp}}$  直接获得选择块重要性分数  $\mathbf{p}_t^{\text{slc}}$ 。对于阻塞方案不同的情况，我们根据选择块的空间关系推导出选择块的重要性分数。给定  $D|L$  和  $D|L'$ ，我们有：

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

其中  $[ \cdot ]$  表示访问向量元素的索引操作符。对于使用 GQA 或 MQA 的模型，其中键值缓存在查询头之间共享，必须确保这些头之间一致的块选择，以最小化解码期间 KV 缓存加载。一组中跨查询头的共享重要性分数正式定义为：

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

其中上标中的  $(<e:1>)$  表示头索引， $H$  表示每组查询头的个数。这种聚合确保了同一组内的头之间一致的块选择。

**Top-NBlock选择。**在获得选择块重要性分数后，我们保留按块重要性分数排名的top-N sparse块内的令牌，公式为：

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

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

式中， $\text{rank}(\cdot)$ 为分值由高到低的排序位置，得分最高的为 $\text{rank} = 1$ ， $\mathcal{I}_t$ 是所选块的索引集， $\text{Cat}$ 为拼接操作。 $K \boxtimes_T^{\text{slc}} \in \mathbf{R}^{D_K \times N_L}$ 是由压缩键组成的张量。类似的公式适用于细粒度值 $V \boxtimes_T^{\text{slc}}$ 。然后，选定的键和值参与方程(5)中定义的 $q_{\text{Tas}}$ 的注意力计算。

### 3.3.3. 滑动窗口

在注意机制中，局部模式通常适应更快，并且可以主导学习过程，可能会阻止模型有效地从压缩和选择令牌中学习。为了解决这个问题，我们引入了一个专用的滑动窗口分支，它显式地处理本地上下文，允许其他分支(压缩和选择)专注于学习各自的特征，而不会被本地模式走捷径。具体来说，我们维护最近的令牌 $K \boxtimes_{\text{Twin}} = \mathbf{k}_{T-W:T}$ ， $V \boxtimes_{\text{Twin}} = \mathbf{v}_{T-W:T}$ 在一个窗口W，并将不同信息源(压缩令牌，和选择令牌，滑动窗口)的注意力计算隔离到单独的分支中。然后，这些分支输出通过学习的门控机制进行聚合。为了进一步防止具有边际计算开销的注意力分支之间的快捷学习，我们为三个分支提供了独立的键和值。这种架构设计通过防止局部和远程模式识别之间的梯度干扰来实现稳定的学习，同时引入最小的开销。

在获得所有三类键和值( $K \boxtimes_{\text{cmp}}$ ,  $V \boxtimes_{\text{cmp}}$ ;  $K \boxtimes_{\text{tslc}}$ ,  $V \boxtimes_{\text{tslc}}$ ; 和  $K \boxtimes_{\text{Twin}}$ ,  $V \boxtimes_{\text{Twin}}$ )，我们根据式(5)计算最终的注意力输出，再加上上面描述的压缩、选择和滑动窗口机制，这就构成了NSA的完整算法框架。

## 3.4. Kernel Design

为了在训练和预填充期间实现flashattention级的加速，我们在Triton上实现了硬件对齐的稀疏注意力内核。鉴于MHA是内存密集型且解码效率低下的，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很容易与现有的FlashAttention-2内核兼容，但我们引入了用于稀疏选择注意力的专门内核设计。如果我们遵循FlashAttention的策略，将暂时连续的查询块加载到SRAM中，这将导致低效的内存访问，因为块内的查询可能需要不连接的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：对于查询序列上的每个位置，我们将GQA组中的所有查询头(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递实现。提出的内核架构具有以下关键特征：

1. **以群体为中心的数据加载。**对于每个内部循环，在位置Tand加载组中所有头的查询 $Q \in \mathbf{R}^{[\boxtimes, D_K]}$ ，它们的共享稀疏键/值块索引 $I_T$ 。

2. 共享KV抓取。在内部循环中，顺序加载由 $L_{\text{intoSRAM}}$ 索引的连续键/值块为 $K \in \mathbb{R}^{[B_K, D_K]}$ 、 $V \in \mathbb{R}^{[B_K, D_V]}$ ，以最小化内存负载，其中 $B_K$ 是内核块大小满足 $B_K | L$ 。
3. 网格上的外环。由于不同查询块的内循环长度(与所选块计数N成正比)几乎保持相同，因此我们将查询/输出循环放在Triton的网格调度程序中，以简化和优化内核。

该设计通过(1)通过分组共享消除冗余KV传输，以及(2)在GPU流多处理器之间平衡计算工作负载，实现了接近最优的算法强度。

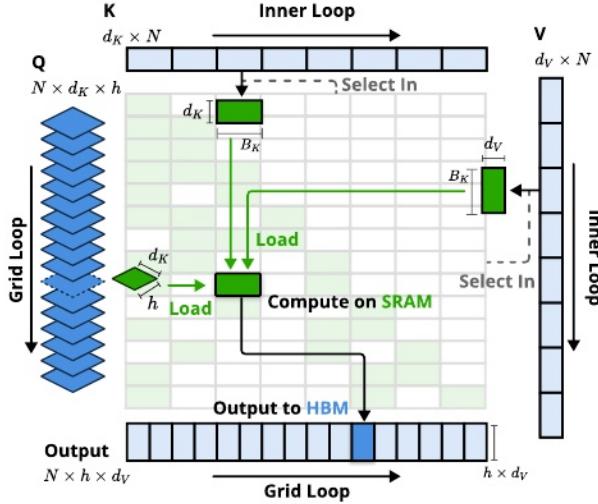


图3 | NSA的内核设计。内核通过GQA组(Grid Loop)加载查询，获取相应的稀疏KV块(Inner Loop)，并在SRAM上执行注意力计算。绿色块表示SRAM上的数据，蓝色块表示HBM上的数据。

## 4. 实验

我们通过三个镜头来评估NSA:(1)一般基准性能，(2)长上下文基准性能，以及(3)思维链推理性能，并与完全注意力基线和最先进的稀疏注意力方法进行比较。我们将我们的稀疏计算范式的效率分析推迟到第5节，在那里我们提供了关于训练和推理速度的详细讨论。

### 4.1. 预训练设置

按照最先进的llm的常见做法，我们的实验采用了一个结合分组查询注意(GQA)和混合专家(MoE)的主干，具有27B个总参数和3B个活动参数。该模型由30层组成，隐含维数为2560。对于GQA，我们将组数设置为4，共64个注意头。对于每个注意头，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，我们使用DeepSeekMoE (Dai et al., 2024; DeepSeek-AI, 2024)结构，72位路由专家和2位共享专家，并将top-k专家设置为6位。为了保证训练的稳定性，将第一层的MoE替换为SwiGLU形式的MLP。

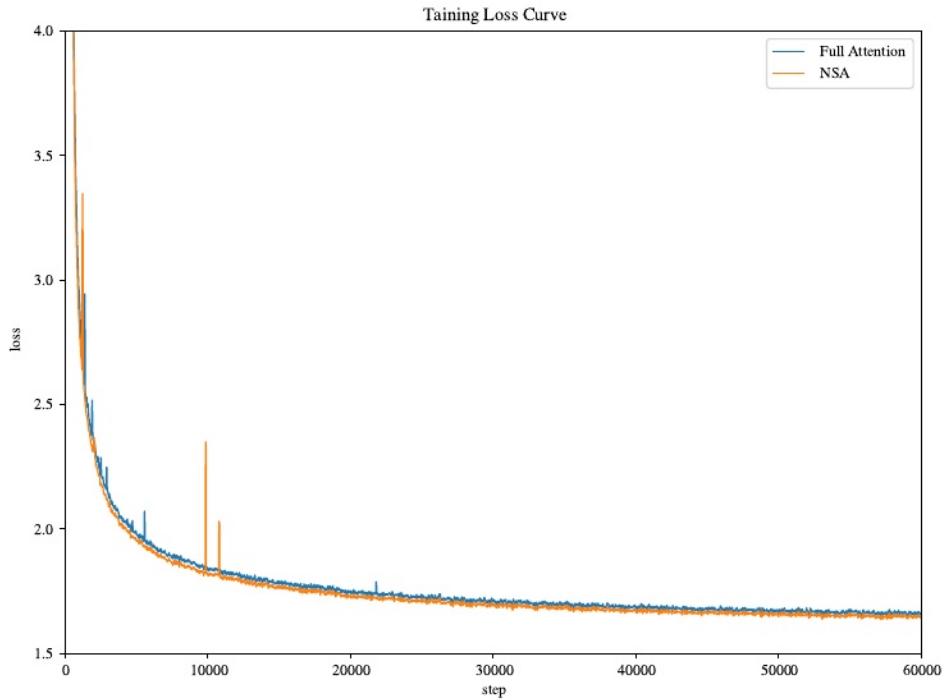


图4 |在27b参数模型上全注意力和NSA的预训练损失比较。两种模型均表现出稳定的收敛性，NSA的损失值较低。

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

表1 |在知识(MMLU, MMLU- pro, CMMLU), 推理(BBH, GSM8K, MATH, DROP)和编码(MBPP, HumanEval)任务上, 全注意力基线和NSA在一般基准上的预训练性能比较。尽管具有高稀疏性, 但NSA在大多数基准测试中实现了卓越的平均性能。

提出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 $D=16$ , 选择块大小 $L'=64$ , 选择块计数 $N=16$ (包括固定激活1个初始块和2个本地块), 滑动窗口大小 $W=512$ 。全注意力和稀疏注意模型都是在8k长度文本的270B个标记上进行预训练的, 然后使用YaRN对32k长度的文本进行持续训练和监督微调(Peng et al., 2024), 以实现长上下文适应。这两个模型都被训练到完全收敛, 以确保公平的比较。如图4所示, 我们的NSA和全注意力基线的预训练损失曲线显示出稳定平稳的下降, 其中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外, 我们还评估了几种最先进的推理阶段稀疏注意方法:H2O (Zhang et al., 2023b)、infLLM (Xiao et al., 2024)、Quest (Tang et al., 2024)和Exact-Top, 后者首先计算全注意力得分, 并选择每个查询对应的top-Nscores键, 然后计算这些位置上的注意力。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	<b>0.512</b>	0.623	0.409	0.350	0.305	<b>0.324</b>	0.294	0.830	<b>0.560</b>	0.163	0.437
NSA	<u>0.503</u>	<b>0.624</b>	<b>0.432</b>	<b>0.437</b>	<b>0.356</b>	0.307	<b>0.341</b>	<b>0.905</b>	0.550	<b>0.232</b>	<b>0.469</b>

表2 |我们的NSA和LongBench上基线的性能比较，包括单文档QA、多文档QA、合成和代码任务类别的子集。NSA的表现超过了包括全注意力在内的大多数基准。

跨越不同的稀疏注意范式，包括kv -缓存驱逐，查询感知选择，和精确的顶部Nsparse选择。

对于一般评估，大多数样本的长度在稀疏注意基线的局部上下文窗口内，这些方法有效地等效于全注意力。因此，在这种情况下，我们只提供NSA和全注意力基线之间的比较结果。在长上下文评估中，我们跨所有基线方法进行比较，将所有稀疏注意方法的稀疏度设置为相同，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，我们将比较限制为全注意力，因为稀疏的注意力基线不支持训练。

### 4.3. 性能对比

**一般评价。**我们在一套涵盖知识、推理和编码能力的综合基准上评估了预训练的NSA和全注意力基线，包括MMLU (hendricks等人， 2020)、MMLU- pro (Wang等人， 2024)、CMMLU (Li等人， 2023)、BBH (Suzgun等人， 2022)、GSM8K (Cobbe等人， 2021)、MATH (hendricks等人， 2020)、DROP (Dua等人， 2019)、MBPP (Austin等人， 2021)和HumanEval (Chen等人， 2021)。结果如表1所示。尽管它是稀疏的，但NSA实现了卓越的整体性能，在9个指标中的7个指标上优于所有基准，包括全注意力。这表明，尽管NSA在较短序列上可能无法充分利用其效率优势，但它表现出了较强的性能。值得注意的是，NSA在推理相关的基准测试中表现出了显著的进步(DROP: +0.042, GSM8K: +0.034)，这表明我们的预训练有助于模型开发专门的注意力机制。这种稀疏的注意力预训练机制迫使模型专注于最重要的信息，通过过滤掉不相关的注意力路径中的噪声，潜在地提高了性能。跨不同评估的一致性能也验证了NSA作为通用架构的健壮性。

**长上下文评估。**如图5所示，在64k上下文的大海捞针(Kamradt, 2023)测试中，NSA在所有位置都实现了完美的检索精度。这种性能源于我们的分层稀疏注意力设计，它结合了压缩令牌来进行高效的全局上下文扫描，以及选择令牌来进行精确的局部信息检索。粗粒度压缩以较低的计算成本识别相关的上下文块，而对所选令牌的令牌级关注确保了关键细粒度信息的保存。这种设计使NSA能够同时保持全球意识和局部精度。

我们还评估了LongBench上的NSA (Bai等人， 2023)对最先进的稀疏注意力的影响

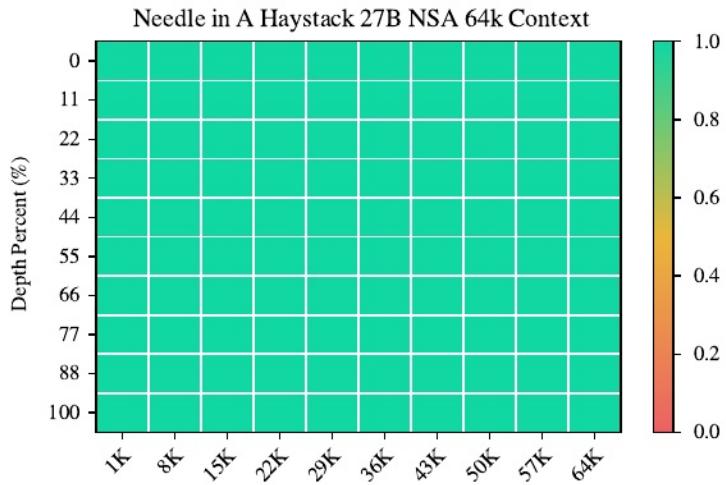


图5 |在64k上下文长度下，跨上下文位置的“大海捞针”检索精度。NSA通过分层稀疏注意力设计达到了完美的准确率。

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基线中每个查询激活的令牌设置为2560个令牌，这与处理32k序列长度时NSA中激活的令牌的平均数量相对应。在Stream-LLM (Xiao et al., 2023)之后，这个令牌预算包括前面的128个令牌和512个本地令牌。我们从LongBench中排除了某些子集，因为它们在所有模型中的得分都很低，这可能无法提供有意义的比较。如表2所示，NSA达到最高的平均得分0.469，优于所有基线(比全注意力 +0.032，比Exact-Top +0.046)。这种改进源于两个关键创新:(1)我们原生的稀疏注意力设计，能够在预训练期间对稀疏模式进行端到端优化，促进了稀疏注意力模块和其他模型组件之间的同步适应;以及(2)分层稀疏注意机制实现了局部和全局信息处理之间的平衡。

值得注意的是，NSA在需要长上下文复杂推理的任务上表现出了出色的性能，在多跳QA任务(HPQ和2Wiki)上比全注意力提高了+0.087和+0.051，超过了代码理解基线的性能(LCC: +0.069)，并且在通道检索方面优于其他方法(PassR-en:+0.075)。这些结果验证了NSA处理各种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优化模式方面提供了额外的好处。

**思维链推理评估(chain of thought Reasoning Evaluation)**。为了评估NSA与先进的下游训练模式的兼容性，我们考察了其通过训练后获得思维链数学推理能力的能力。鉴于强化学习在较小规模模型上的有效性有限，我们采用DeepSeek-R1的知识蒸馏，对10B个32k长度的数学推理痕迹进行监督微调(SFT)。这产生了两个可比较的模型:Full Attention- r(全注意基线)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)基准上评估了这两个模型。我们使用0.7的采样温度和0.95的top-Pvalue为每个问题生成16个答案，并获得平均分数。为了验证推理深度的影响，我们使用两个生成上下文限制(8k和16k token)进行实验，测量扩展推理链是否提高了准确性。附录A中提供了模型预测的示例比较。

Generation Token Limit	8192	16384
Full Attention-R	0.046	0.092
NSA-R	<b>0.121</b>	<b>0.146</b>

表3监督微调后基于AIME指令的评估。我们的NSA-R在8k和16k序列长度上都比Full Attention-R表现得更好

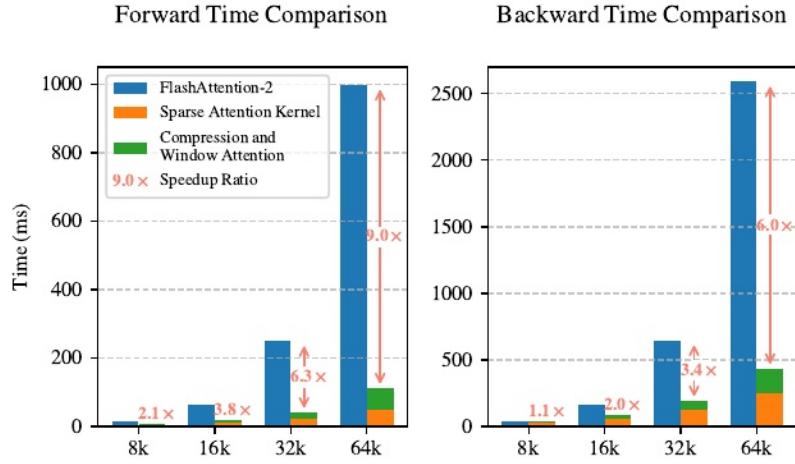


图6 | triton based NSA内核与triton based FlashAttention-2内核的比较。我们的实现显着降低了所有上下文长度的延迟，随着输入长度的增加，改进变得更加明显。

如表3所示，在8k上下文设置下，NSA-R的准确率明显高于Full Attention-R(+0.075)，在16k上下文设置下，这种优势持续存在(+0.054)。这些结果验证了原生稀疏注意的两个关键优势：(1)预训练的稀疏注意模式能够有效捕获对复杂数学推导至关重要的远程逻辑依赖；(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，而不会出现灾难性遗忘。跨上下文长度的一致内容的优异表现证实了稀疏注意力在原生集成到训练管道中时对高级推理任务的可行性。

## 5. 效率分析

我们在一个8 gpu的A100系统上评估了NSA在全注意力情况下的计算效率。在效率分析中，我们还将模型配置为GQA组G=4，每个组的头数<s:2> = 16，查询/键维D<sub>K</sub>=192，值维D<sub>V</sub>=128。按照第4节中相同的设置，我们设置NSA压缩块大小L=32，滑动步幅D=16，选择块大小L'=64，选择块计数N=16，滑动窗口大小W=512。

### 5.1. 训练速度

我们将基于triton的NSA注意力和Full attention的实现与基于triton based FlashAttention-2进行比较，以确保在同一后端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获得了越来越大的加速，在64k上下文长度时，向前加速达到9.0倍，向后加速达到6.0倍。值得注意的是，随着序列变长，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	<b>4×</b>	<b>6.4×</b>	<b>9.1×</b>	<b>11.6×</b>

表4 |在解码过程中每次注意操作的内存访问量(以等值令牌数计算)。由于解码的算术强度低和内存受限的性质，预期的加速与内存访问量近似线性。

硬件对齐的算法设计，以最大限度地提高稀疏注意力架构的效率:(1)块内存访问模式通过合并负载最大化张量核心利用率;(2)内核中微妙的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取决于内存访问瓶颈，这与KV缓存加载totheamount密切相关。在每个解码步骤中，我们的NSA最多只需要加载 $S^{\text{LD}}_{\text{压缩}}$ 令牌， $N_L^{\text{selected}}$ 令牌和 $W_{\text{neighbor}}$ 令牌，

其中 $S_{\text{is}}$ 为缓存的序列长度。如表4所示，随着解码长度的增加，我们的方法显示出延迟的显著降低，在64k上下文长度下实现了高达11.6倍的加速。这种在内存访问效率上的优势也会随着更长的序列而放大。

## 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，但理解替代策略遇到的挑战和分析注意力模式为未来的研究方向提供了有价值的背景。我们首先研究了替代令牌选择策略的挑战，这些策略激发了我们的设计选择，然后是可视化，提供了对注意力分布模式的见解。

### 6.1. 替代令牌选择策略的挑战

在设计NSA之前，我们探索了将现有的稀疏注意方法应用到训练阶段。然而，这些尝试遇到了各种各样的挑战，促使我们设计了一种不同的稀疏注意力架构：

**基于键集群的策略。**我们研究了基于聚类的策略，如Clus-terKV (Liu et al., 2024)。这些方法将来自同一集群的键和值存储在连续的内存区域中。虽然理论上对训练和推理是可行的，但它们面临着三个重大挑战:(1)动态聚类机制引入的非平凡计算开销;(2)集群间不平衡加剧了算子优化困难，特别是在混合专家(MoE)系统中，歪斜的专家并行(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现约束。这些综合因素造成了实质性的瓶颈，极大地限制了它们在实际部署中的有效性。

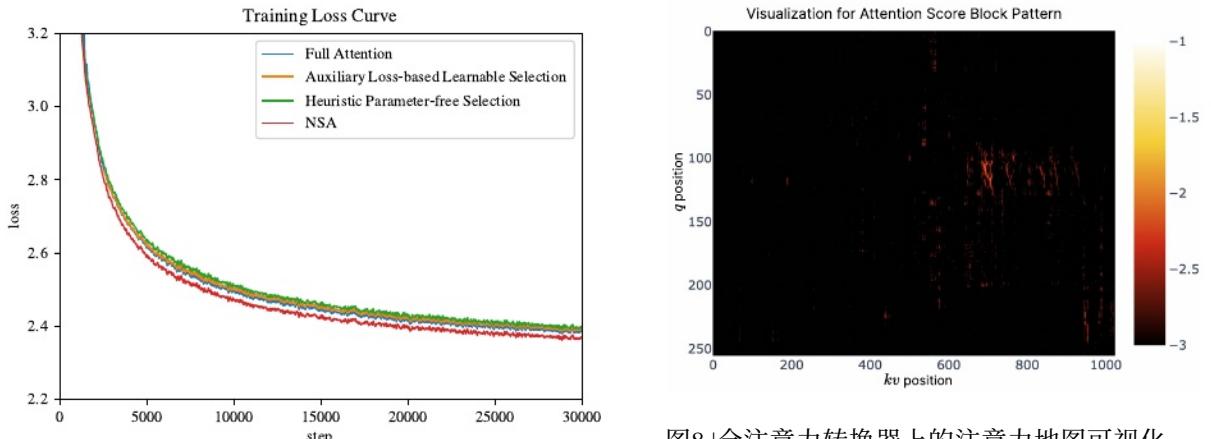


图7 | 比较具有全注意力和不同令牌选择策略的3b参数模型的训练损失。我们的NSA实现了更好的性能。

图8 | 全注意力转换器上的注意力地图可视化。

有颜色的区域表示更高的关注值。如图所示，注意力得分呈现块状聚类分布。

**其他分块选择策略。**我们还考虑了与NSA不同的块键值选择策略，如Quest (Tang等人, 2024)和InfLLM (Xiao等人, 2024)。这些方法依赖于计算每个块的重要性分数，并根据它们与 $Q_T$ 的相似性选择最上面的-N块。然而，现有方法面临两个关键问题：(1)由于选择操作不可微，基于神经网络的重要性分数计算依赖于辅助损失，这增加了算子开销，往往会降低模型性能；(2)启发式无参数重要分数计算策略召回率低，导致性能次优。我们在具有相似架构的3b参数模型上评估了这两种方法，并将其损失曲线与NSA和全注意力进行了比较。对于辅助的基于损失的选择方法，我们为每个块引入了额外的查询和代表性键来估计块的重要性分数。这些分数由每个块内原始查询和键之间的平均关注分数来监督。对于启发式无参数选择方法，遵循Quest的策略，我们使用查询和键块的坐标最小最大值之间的乘积实现直接选择，而不引入额外的参数。我们还探索了一种冷启动训练方法，在过渡到启发式块选择之前，对最初的1000步应用全注意力。如图7所示，两种方法都表现出较低的损失。

## 6.2. 可视化

为了探索变压器注意力分布的潜在模式，并为我们的设计寻找灵感，我们从图8中预训练的27B全注意力模型中可视化了注意力图。可视化揭示了有趣的模式，其中注意力得分倾向于表现出块状聚类特征，附近的键通常显示相似的注意力得分。这一观察启发了我们设计NSA的灵感，表明基于空间连续性选择关键块可能是一种很有前途的方法。块聚类现象表明，序列中相邻的令牌可能与查询令牌共享某些语义关系，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的注意力机制，该机制在连续的令牌块上运行，而不是在单个令牌上运行，旨在提高计算效率并保持高注意力模式。

## 7. 相关作品

我们回顾了现有的通过稀疏注意提高注意力计算效率的方法。这些方法根据其核心策略大致可以分为三组:(1)固定稀疏模式, (2)动态令牌修剪, (3)查询感知选择。我们分别介绍了每一类中的几个代表性作品。

### 7.1. Fixed Sparse Pattern

SlidingWindow是一种常用的方法，它允许查询只在固定的窗口内计算注意力。StreamingLLM (Xiao等人, 2023)通过维护上下文的两个关键部分来解决处理长文本流的挑战:注意力吸收(早期令牌)和本地上下文窗口。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刚性模式限制了它们在需要完全上下文理解的任务上的性能。

### 7.2. 动态令牌修剪

H2O (Zhang et al., 2023b)实现了一种自适应方法来减少解码过程中kv缓存内存的使用。该方法根据注意力得分,根据它们最近的效用,动态地驱逐对未来预测不太重要的令牌。SnapKV (Li et al., 2024)还引入了一种令牌修剪策略,通过选择性地仅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通过预填充期间的关注权重分析和投票来识别重要特征,然后通过将选择的压缩特征与最近的上下文相结合来更新KV缓存,以保持快速一致性。

### 7.3. Query-Aware Selection

Quest (Tang et al., 2024)采用了一种块选择策略,其中每个块的重要性是通过关键块的查询和坐标智能最小最大值之间的乘积来估计的。结果得分有助于选择top -Nimportant键值块进行关注。InfLLM (Xiao et al., 2024)通过维护注意力汇、局部上下文和可检索的块,将固定模式与检索相结合。该方法从每个块中选择具有代表性的关键字来估计块的重要性。HashAttention (Desai et al., 2024)通过使用学习函数将查询和键映射到Hamming空间,将关键令牌识别制定为推荐问题。ClusterKV (Liu et al., 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进行关注计算来实现稀疏性。

## 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。通过将分层令牌压缩与可训练架构内的块令牌选择集成在一起,我们的架构实现了加速训练和推理,同时保持了Full Attention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,并增强推理能力,从而提高了最先进的技术水平,所有这些都伴随着计算延迟的可测量减少,并实现了显著的加速。

## 参考文献

- 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.
- J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, et al. Program synthesis with large language models. arXiv preprint arXiv:2108.07732, 2021.
- Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang, Z. Du, X. Liu, A. Zeng, L. Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. arXiv preprint 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, et al. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021.
- Z. Chen, R. Sadhukhan, Z. Ye, Y. Zhou, J. Zhang, N. Nolte, Y. Tian, M. Douze, L. Bottou, Z. Jia, et al. Magicpig: Lsh sampling for efficient llm generation. arXiv preprint arXiv: 2410.16179, 2024.
- 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, 2021. URL <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, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. arXiv preprint arXiv:2401.06066, 2024.
- DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. 2024. URL <https://arxiv.org/abs/2405.04434>.
- DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL <https://arxiv.org/abs/2501.12948>.
- A. Desai, S. Yang, A. Cuadron, A. Klimovic, M. Zaharia, J. E. Gonzalez, and I. Stoica. Hashatten-tion: Semantic sparsity for faster inference. arXiv preprint arXiv:2412.14468, 2024.
- D. Dua, Y. Wang, P. Dasigi, G. Stanovsky, S. Singh, and M. Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. arXiv preprint arXiv:1903.00161, 2019.
- S. Ge, Y. Zhang, L. Liu, M. Zhang, J. Han, and J. Gao. Model tells you what to discard: Adaptive kv cache compression for llms. arXiv preprint 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, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. arXiv preprint arXiv:2403.05530, 2024.
- D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. arXiv preprint arXiv:2009.03300, 2020.
- H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, and L. Qiu. Llmlingua: Compressing prompts for accelerated inference of large language models. arXiv preprint 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, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. arXiv preprint arXiv:2407.02490, 2024.
- G. Kamradt. LLMTest NeedleInAHaystack. GitHub repository, 2023. URL [https://github.com/gkamradt/LLMTest\\_NeedleInAHaystack](https://github.com/gkamradt/LLMTest_NeedleInAHaystack). Accessed: [Insert Access Date Here].
- H. Li, Y. Zhang, F. Koto, Y. Yang, H. Zhao, Y. Gong, N. Duan, and T. Baldwin. Cmmlu: Measuring massive multitask language understanding in chinese. arXiv preprint arXiv: 2306.09212, 2023.
- Y. Li, Y. Huang, B. Yang, B. Venkitesh, A. Locatelli, H. Ye, T. Cai, P. Lewis, and D. Chen. Snapkv: Llm knows what you are looking for before generation. arXiv preprint arXiv: 2404.14469, 2024.
- G. Liu, C. Li, J. Zhao, C. Zhang, and M. Guo. Clusterkv: Manipulating llm kv cache in semantic space for recallable compression. arXiv preprint arXiv:2412.03213, 2024.
- J. S. Park, J. C. O' Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein. Generative agents: Interactive simulacra of human behavior. In S. Follmer, J. Han, J. Steimle, and N. H. Riehe, editors, Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, UIST 2023, San Francisco, CA, USA, 29 October– 1 November 2023, pages 2:1–2:22. ACM, 2023.
- B. Peng, J. Quesnelle, H. Fan, and E. Shippole. Yarn: Efficient context window extension of large language models. In ICLR. OpenReview.net, 2024.
- N. Shazeer. Fast transformer decoding: One write-head is all you need. 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, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. arXiv preprint arXiv:2210.09261, 2022.
- J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, and S. Han. Quest: Query-aware sparsity for efficient long-context llm inference. arXiv preprint arXiv:2406.10774, 2024.
- P. Tillet, H.-T. Kung, and D. Cox. Triton: an intermediate language and compiler for tiled neural network computations. In Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, pages 10–19, 2019.
- A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. Advances in Neural Information Processing Systems, 2017.
- Y. Wang, X. Ma, G. Zhang, Y. Ni, A. Chandra, S. Guo, W. Ren, A. Arulraj, X. He, Z. Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. arXiv preprint arXiv:2406.01574, 2024.
- J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems, 35:24824–24837, 2022.
- C. Xiao, P. Zhang, X. Han, G. Xiao, Y. Lin, Z. Zhang, Z. Liu, and M. Sun. Inflilm: Training-free long-context extrapolation for llms with an efficient context memory. In The Thirty-eighth Annual Conference on Neural Information Processing Systems, 2024.

- G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis. Efficient streaming language models with attention sinks. *arXiv preprint 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, et al. Big bird: Transformers for longer sequences. *Advances in neural information processing systems*, 33:17283–17297, 2020.
- 
- E. Zelikman, Y. Wu, J. Mu, and N. D. Goodman. Star: Bootstrapping reasoning with reasoning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 –December 9, 2022*, 2022.
- 
- F. Zhang, B. Chen, Y. Zhang, J. Keung, J. Liu, D. Zan, Y. Mao, J. Lou, and W. Chen. Repocoder: Repository-level code completion through iterative retrieval and generation. In H. Bouamor, J. Pino, and K. Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6–10, 2023, pages 2471– 2484. Association for Computational Linguistics, 2023a*.
- 
- K. Zhang, J. Li, G. Li, X. Shi, and Z. Jin. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. In L. Ku, A. Martins, and V. Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ACL 2024, Bangkok, Thailand, August 11–16, 2024, pages 13643–13658.
- 
- Z. Zhang, Y. Sheng, T. Zhou, T. Chen, L. Zheng, R. Cai, Z. Song, Y. Tian, C. Ré, C. Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023b.
- 
- Z. Zhou, C. Li, X. Chen, S. Wang, Y. Chao, Z. Li, H. Wang, R. An, Q. Shi, Z. Tan, et al. Llm × mapreduce: Simplified long-sequence processing using large language models. *arXiv preprint arXiv:2410.09342*, 2024.
-

## A. AIME结果示例

提示:

“|用户让 $\geq$ , YandZbe满足以下方程组的正实数

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

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

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

则 $\log_2(x^4y^3z^2)$ 的值为M where M and N are 相对素数正整数。找到M+N。请一步步推理，把你的最终答案放进去。|助理|”

□

我们的结果:

☒think☒⋯☒/think☒(2275个token)

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成它们的指数形式:

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

将每个变量表示为:

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

将这些表达式相互代入，我们得到:

-从 $x = \sqrt{2} \cdot yz$ ，我们得到 $x = 2 \cdot (2^{1/3} \cdot xz)$ ，导致。代入 $y = 2^{1/3} \cdot xz$ ，得到 $y = 2^{-2/3} \cdot xz$ 。代入 $z = 2^{1/4} \cdot xy$ ，得到 $z = 2^{-7/24} \cdot xy$ 。

接下来，我们计算 $x^4y^3z^2$ :

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

将这些相乘:

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

取以2为底的对数

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

绝对值为:

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

Thus, M= 25 and N= 8, giving M+ N= 33.

33

基线结果:

think ··· /think (9392个token)

给定对数方程的系统:

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

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

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

我们将这些方程转换成指数形式:

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

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

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

将它们相互代入, 我们发现:

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

通过逐步求解这些方程, 我们发现:

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

然而, 在验证第三个方程时, 我们意识到系统可能没有解。因此, 我们使用矩阵反演来解这个系统:

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

设 $A = \log_2(X)$ ,  $B = \log_2(Y)$ ,  $C = \log_2(Z)$ 。系统变成:

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

☒

我们使用Cramer规则和矩阵反演来求解这个系统。矩阵的行列式为-2, 解为:

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

由此, 我们有:

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

我们需要找到 $\log_2(x^4y^3z^2)$ :

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

转换成公分母(48):

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

将这些求和:

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

分数 $\frac{115}{16}$ 是最简单的形式, 所以是 $M=115$ 和 $N=16$ 。因此,  $M+N=115+16=131$ 。

131

### 提示:

“|用户|实数 $x$ and $y$ with分, 分>1满足 $\log_x(y^x)=\log_y(x^y)=10$ 。”的值是多少?

请一步步推理, 并把你的最终答案放进去。|助理|”  
□

### 我们的结果:

☒think☒⋯☒/think☒(15147个token)

给定方程:

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

我们首先对两个方程应用对数幂法则。

对于第一个方程:

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

设 $Z=\log_Y(X)$ 。那么,  $\log_X(Y)=Z$ 。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:

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

利用幂次法则, 我们得到:

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

由于 $\log_Y(X)=<e:1>$ , 这化简为:

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

我们现在有两个表达式:

1.  $x = 10z$
2.  $yz = \frac{5}{2}$

将 $X=10$  and  $Y=\frac{5}{2}$  代入乘积中, 我们得到:

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

由此, 得到的值为 $Y$  is 25。  $\square$

### 基线结果:

$\boxed{\text{think}} \cdots \boxed{\text{think}}$  (16223个token)

给定实数 $X$  and  $Y$  with  $\geq 2$ , 满足方程 $\log_X(Y^X) = 10$  和  $\log_Y(X^4Y) = 10$  的问题, 我们需要找到 $X$  and  $Y$  的值。

首先, 我们使用换底公式将对数方程转换成指数形式。

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

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

使用换底公式，得到 $\log$ 的溢(分)= $\ln \ln$ 溢(分)。由于对数取分式=94，我们得到对数取分式=49。因此，

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

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

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

同样，使用换底公式， $\log_x Y = \frac{9}{4}$ 。

接下来，我们互相表示XandYin项：

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

然后，我们计算出乘积的优劣：

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

然而，在重新评估并使用另一种方法(设置为：

$$\begin{aligned} \log_x(y^x) = 10 &\implies \frac{x \log_x y}{\log_x y} = 10 \implies x \cdot \frac{4}{9} = 10 \implies x = \frac{45}{2} \\ \log_y(x^4y) = 10 &\implies \frac{4 \log_y x + 1}{\log_y x} = 10 \implies 4 \cdot \frac{9}{4} + 1 = 10 \implies 9 + 1 = 10 \quad \text{consistent} \end{aligned}$$

最后，我们发现，乘积为：

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