

Native Sparse Attention: Hardware-Aligned and Natively Trainable Sparse Attention

Jingyang Yuan^{*1,2}, Huazuo Gao¹, Damai Dai¹, Junyu Luo², Liang Zhao¹, Zhengyan Zhang¹, Zhenda Xie¹,
Y. X. Wei¹, Lean Wang¹, Zhiping Xiao³, Yuqing Wang¹, Chong Ruan¹, Ming Zhang², Wenfeng Liang¹,
Wangding Zeng¹

¹DeepSeek-AI

²Key Laboratory for Multimedia Information Processing, Peking University, PKU-Anker LLM Lab

³University of Washington

{yuanjy, mzhang_cs}@pku.edu.cn, {zengwangding, wenfeng.liang}@deepseek.com

Abstract

*警告：该PDF由GPT-Academic开源项目调用大语言模型+Latex翻译插件一键生成，版权归原文作者所有。翻译内容可靠性无保障，请仔细鉴别并以原文为准。项目Github地址 https://github.com/binary-husky/gpt_academic/。当前大语言模型: gpt-4o-mini，当前语言模型温度设定: 0。为了防止大语言模型的意外谬误产生扩散影响，禁止移除或修改此警告。

长上下文建模对下一代语言模型至关重要，但标准注意力机制的高计算成本带来了显著的计算挑战。稀疏注意力为提高效率同时保持模型能力提供了一个有前景的方向。我们提出了 NSA，一种Natively可训练的Sparse Attention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采用动态层次稀疏策略，结合粗粒度的标记（Token）压缩和细粒度的标记选择，以保留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动了稀疏注意力设计：（1）通过算术强度平衡的算法设计，实现了显著的加速，并对现代硬件进行了实现优化。（2）我们实现了端到端的训练，减少了预训练计算而不牺牲模型性能。如Section 1所示，实验结果表明，使用NSA进行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与全注意力模型保持一致或超越。同时，NSA在解码、前向传播和反向传播过程中，对于64k长度序列相较于全注意力实现了显著加速，验证了其在模型生命周期中的效率。

1. Introduction

近年来，研究界越来越认识到长上下文建模作为下一代大型语言模型的关键能力，这一能力受到各种现实世界应用的驱动，这些应用包括深入推理 (DeepSeek-AI, 2025; Zelikman et al., 2022)、仓库级代码生成 (Zhang et al., 2023a; Zhang et al.) 和多轮自主智能体系统 (Park et al., 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 et al., 2024)，使得模型能够处理整个代码库、冗长的文档，维护数千个标记（Token）上的连贯多轮对话，并在长范围依赖中进行复杂推理。然而，普通注意力（Attention）机制的高复杂性 (Zaheer et al., 2020) 在序列长度增加时成为一个关键的延迟瓶颈。理论估计表明，在解

*Contribution during internship at DeepSeek-AI.

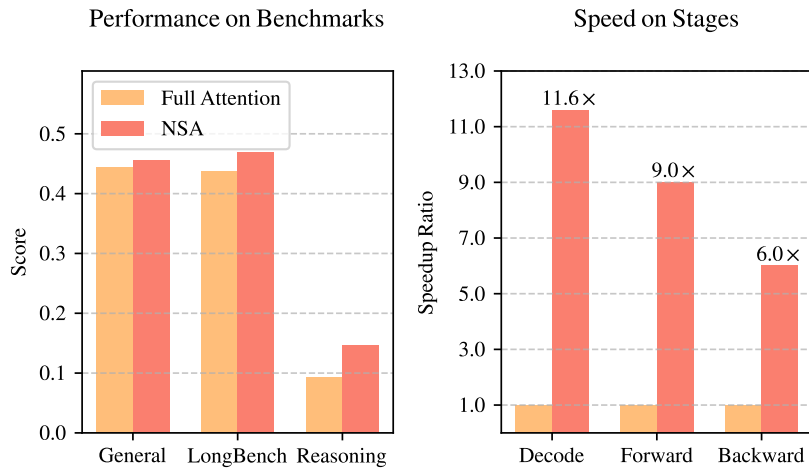


Figure 1 | 对全注意力模型（Full Attention model）和我们的NSA之间的性能和效率进行比较。左：尽管稀疏，NSA在一般基准、长上下文任务和推理评估中平均超越全注意力基线。右：对于64k长序列处理，NSA在所有阶段（解码、正向传播和反向传播）相比全注意力模型实现了显著的计算加速。

码64k长度的上下文时，使用softmax架构的注意力计算占总延迟的70–80%，这凸显了对更高效的注意力机制的迫切需求。

实现高效长上下文建模的自然方法是利用softmax注意力的内在稀疏性 (Ge et al., 2023; Jiang et al., 2023)，在选择性计算关键的查询-键对时，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种潜力：KV-cache驱逐方法 (Li et al., 2024; Zhang et al., 2023b; Zhou et al., 2024)、块状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)。尽管这些有前途的策略，现有的稀疏注意力方法在实际部署中往往难以达到效果。许多方法未能实现与其理论增益相媲美的加速；此外，大多数方法主要集中在推理阶段，缺乏有效的训练时支持，无法充分利用注意力的稀疏模式。

为了解决这些局限性，部署有效的稀疏注意力必须面对两个关键挑战：（1）与硬件对齐的推理加速：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设计硬件友好的算法，以减轻内存访问和硬件调度的瓶颈；（2）关注训练的算法设计：通过可训练操作符启用端到端计算以降低训练成本，同时保持模型性能。这些要求对于现实应用实现快速长上下文推理或训练至关重要。考虑到这两个方面，现有方法仍然表现出显著差距。

为了实现更有效和高效的稀疏注意力，我们提出了NSA，一种本地可训练的稀疏注意力架构，集成了分层标记（Token）建模。如Figure 2所示，NSA通过将键和值组织成时间块并通过三个注意力路径处理来减少每个查询的计算：压缩的粗糙标记（Token）、选择性保留的细粒度标记（Token）和用于局部上下文信息的滑动窗口。然后，我们实现专门的内核以最大限度地提高其实际效率。NSA引入了两个与上述关键需求对应的核心创新：（1）硬件对齐系统：优化块状稀疏注意力以便充分利用Tensor核心并进行内存访问，确保算术强度均衡。（2）关注训练的

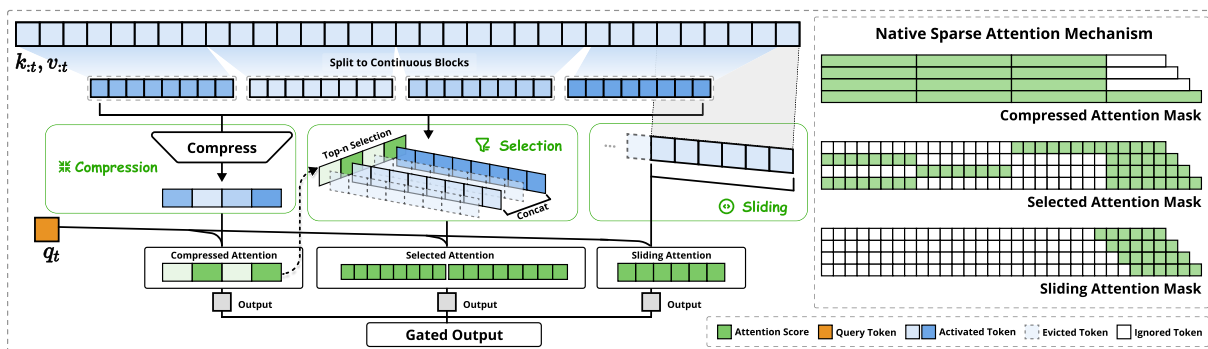


Figure 2 | NSA架构的概述。左侧：该框架通过三个并行的注意力分支处理输入序列：对于给定的查询，前面的键（keys）和值（value）被处理为粗粒度模式的压缩注意力，重要标记（Token）块的选定注意力，以及局部上下文的滑动注意力。右侧：不同分支产生的注意力模式的可视化。绿色区域表示需要计算注意力得分的区域，而白色区域代表可以跳过的区域。

设计：通过高效的算法和反向操作符实现稳定的端到端训练。这一优化使得NSA能够支持高效部署和端到端训练。

我们通过在现实语言语料库上进行全面实验来评估NSA。在具有260B标记（Token）的27B参数transformer骨干网络上进行预训练，我们评估NSA在通用语言评估、长上下文评估和思维链推理评估中的性能。我们进一步比较在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在性能上与完全注意力基线相当或优越，同时优于现有的稀疏注意力方法。此外，与完整注意力相比，NSA在解码、前向和反向阶段提供了显著的加速，随着序列的延长，速度提升比率增加。这些结果验证了我们的层级稀疏注意力设计在模型能力和计算效率之间有效平衡。

2. Rethinking Sparse Attention Methods

现代稀疏注意力方法在减少transformer（transformer）模型的理论计算复杂度方面取得了显著进展。然而，大多数方法主要在推断过程中应用稀疏性，同时保留预训练的全注意力骨架，这可能引入架构偏差，从而限制其充分利用稀疏注意力优势的能力。在介绍我们本土稀疏架构之前，我们通过两个关键视角系统地分析这些局限性。

2.1. The Illusion of Efficient Inference

尽管在注意力计算中实现了稀疏性，但许多方法在推断延迟上未能相应减少，主要面临两个挑战：

相位限制稀疏性（**Phase-Restricted Sparsity**）。像 H2O (Zhang et al., 2023b) 这样的方法在自回归解码期间应用稀疏性，同时在预填充期间需要计算密集的预处理（例如，注意力图计算、索引构建）。相比之下，像 MInference (Jiang et al., 2024) 这样的方法仅关注于预填充稀疏性。这些方法未能在所有推断阶段实现加速，因为至少一个阶段的计算成本与全注意力（**Full Attention**）相当。相位专业化降低了这些方法在以预填充为主的工作负载（如书籍总结和代码

补全)或以解码为主的工作负载(如长链推理)中的加速能力。

与高级注意力架构的不兼容性 (**Incompatibility with Advanced Attention Architecture**)。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 (Multiple-Query Attention, MQA) (Shazeer, 2019) 和分组查询注意力 (Grouped-Query Attention, GQA) (Ainslie et al., 2023), 这些架构通过在多个查询头之间共享 KV 有效减少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,每个注意力头独立选择其 KV 缓存子集。尽管在多头注意力 (Multi-Head Attention, MHA) 模型中展示了一致的计算稀疏性和内存访问稀疏性,但在基于 GQA 等架构的模型中则出现了不同的情况,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。这一架构特性意味着,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。这一限制迫使进行一个关键选择: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来自于高级架构的高效内存访问设计相冲突。

这些局限性产生的原因在于,许多现有的稀疏注意力方法专注于 KV 缓存的减少或理论计算的减少,但在高级框架或后端实现中却难以实现显著的延迟减少。这促使我们开发结合高级架构与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. The Myth of Trainable Sparsity

我们的目标是实现本地可训练的稀疏注意力,这源于对仅推理方法分析的两个关键洞见:(1) 性能退化:在后期施加稀疏性迫使模型偏离其预训练优化轨迹。如Chen et al. (2024)所示,前20%的注意力仅能覆盖总注意力分数的70%,使得在推理过程中,预训练模型中的检索头等结构容易受到修剪的影响。(2) 训练效率需求:高效处理长序列训练对现代大型语言模型 (LLM) 的发展至关重要。这包括在较长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 and 强化学习。然而,现有的稀疏注意力方法主要针对推理,导致在训练中计算挑战得到的解决相对较少。这一局限性妨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了一些挑战:

不可训练的组件:像ClusterKV (Liu et al., 2024) (包括k均值聚类) 和MagicPIG (Chen et al., 2024) (包括基于SimHash的选择) 的方法中的离散操作在计算图中造成不连续性。这些不可训练的组件阻止了通过标记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低效的问题。在像HashAttention (Desai et al., 2024)的方法中使用的标记粒度选择策略导致在注意力计算过程中需要从KV缓存中加载大量独立的标记。这种非连续的内存访问妨碍了快速注意力技术(如FlashAttention)的高效适应,而后者依赖于连续内存访问和块状计算以实现高吞吐量。因此,实施结果被迫回落到低硬件利用率,从而显著降低训练效率。

2.3. Native Sparsity as an Imperative

这些推断效率和训练可行性的限制促使我们对稀疏注意力机制进行根本性的重新设计。我们提出了 NSA，一个本质上稀疏的注意力框架，旨在解决计算效率和训练需求。在接下来的章节中，我们详细介绍 NSA 的算法设计和操作符实现。

3. Methodology

我们的技术方法涉及算法设计和内核优化。在以下小节中，我们首先介绍我们方法论的背景。然后，我们呈现 NSA 的整体框架，随后是其关键算法组件。最后，我们详细介绍了我们优化硬件的内核设计，以最大化实际效率。

3.1. Background

注意力机制 广泛应用于语言建模，其中每个查询标记（Token） \mathbf{q}_t 计算与所有前面的键 $\mathbf{k}_{:t}$ 的相关性得分，以生成一个加权的价值 (value) 总和 $\mathbf{v}_{:t}$ 。形式上，对于长度为 t 的输入序列，注意力操作被定义为：

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

其中 Attn 表示注意力函数 (attention function)：

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

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。当序列长度增加时，注意力计算在整体计算成本中变得越来越占主导地位，这对于长上下文处理提出了显著挑战。

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临界算术强度，该算术强度计算为这两个硬件限制的比率。对于计算任务，算术强度高于这一临界阈值时变为计算绑定 (受 GPU 的浮点运算能力限制)，而低于该阈值时变为内存绑定 (受内存带宽限制)。

具体而言，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变为计算绑定。相比之下，自回归解码变为内存带宽受限，因为它每次前向传递生成一个标记 (Token)，同时需要加载整个键值缓存，从而导致低算术强度。这导致不同的优化目标——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. Overall Framework

为了利用具有自然稀疏模式的注意力潜力，我们建议将Equation (1)中的原始键-值对 $\mathbf{k}_{:t}, \mathbf{v}_{:t}$ 替换为给定每个查询 \mathbf{q}_t 的更紧凑且信息密集的特征键-值对 $\tilde{\mathbf{K}}_t, \tilde{\mathbf{V}}_t$ 。具体而言，我们正式定义优化后的注意力输出如下：

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

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

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

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

如图 Figure 2 所示，NSA 具有三种映射策略 $C = \{\text{cmp}, \text{slc}, \text{win}\}$ ，分别表示键和值的压缩（compression）、选择（selection）和滑动窗口（sliding window）。 $g_t^c \in [0, 1]$ 是对应策略 c 的门得分（gate score），通过 MLP 和 sigmoid 激活从输入特征中得出。设 N_t 表示重映射的键/值的总数量：

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

通过确保 $N_t \ll t$ ，我们保持较高的稀疏性比率（sparsity ratio）。

3.3. Algorithm Design

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：标记（Token）压缩、标记（Token）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征，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。形式上，压缩的键表征（representation）定义为：

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

在这里， l 是块长度， d 是相邻块之间的滑动步幅，并且 φ 是一个可学习的多层感知机（MLP），具有块内位置编码，用于将块中的键映射到单个压缩键。 $\tilde{\mathbf{K}}_t^{\text{cmp}} \in \mathbb{R}^{d_k \times \lfloor \frac{t-l}{d} \rfloor}$ 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。对于压缩值表征（representation） \tilde{v}_t^{cmp} 也有类似的公式。压缩表征捕捉到更粗粒度的高级语义信息，并减少注意力（attention）的计算负担。

3.3.2. Token Selection

使用仅压缩的标记（Token），价值（value）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标记（Token）和价值（value）。下面我们描述了一种高效的标记（Token）选择机制，该机制能够以低计算开销识别和保留最相关的标记（Token）。

分块选择（Blockwise Selection）。我们的选择策略以空间连续块的方式处理关键（key）和价值（value）序列，这源于两个关键因素：硬件效率考虑和注意力分数的内在分布模式。分块选择对在现代GPU上实现高效计算至关重要。这是因为现代GPU架构在连续块访问上的吞吐量显著高于基于随机索引的读取。此外，分块计算能够实现Tensor Cores的最佳利用。这一架构特性将分块内存访问和计算确立为高性能注意力实现中的基本原则，FlashAttention的基于块的设计就是一个例子。分块选择遵循注意力分数的内在分布模式。之前的研究 (Jiang et al., 2024) 表明，注意力分数通常表现出空间连续性，这表明相邻的关键（key）往往共享相似的重要性水平。我们在Section 6.2中的可视化也展示了这种空间连续模式。

为了实现分块选择，我们首先将关键（key）和价值（value）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（Importance Score Computation）。计算块的重要性分数可能会引入显著的开销。幸运的是，压缩标记（Token）的注意力计算生成了中间的注意力分数，我们可以利用这些分数来引导选择块的重要性分数，其公式为：

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

其中 $\mathbf{p}_t^{\text{cmp}} \in \mathbb{R}^{\lfloor \frac{l}{d} \rfloor}$ 是 \mathbf{q}_t 与压缩键 $\tilde{\mathbf{K}}_t^{\text{cmp}}$ 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 $\mathbf{p}_t^{\text{slc}} = \mathbf{p}_t^{\text{cmp}}$ 轻松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，我们根据它们的空间关系推导选择块的重要性分数。由于 $d \mid l$ 和 $d \mid l'$ ，我们得出：

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

在这里 $[\cdot]$ 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（model），当关键-价值（key-value）缓存在查询头（query heads）之间共享时，必须确保这些头之间的一致块选择，以最小化解码期间KV缓存的加载。组内头部之间共享的重要性评分被正式定义为：

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

其中, (h) 在上标中表示头索引 (head index), H 是每组中的查询头数量。此聚合确保在同一组内的各个头之间保持一致的块选择。

Top- n 块选择 (Block Selection). 在获得选择块的重要性评分后, 我们保留按块重要性评分排名的前- n 个稀疏块中的标记 (token), 其公式为:

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

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

其中 $\text{rank}(\cdot)$ 表示按降序排列的排名位置, $\text{rank} = 1$ 对应最高分, \mathcal{I}_t 是所选区块的索引集合, Cat 表示连接操作。 $\tilde{\mathbf{K}}_t^{\text{slc}} \in \mathbb{R}^{d_k \times nl'}$ 是由压缩键构成的张量 (tensor)。类似的公式适用于细粒度价值 $\tilde{\mathbf{V}}_t^{\text{slc}}$ 。然后, 所选的键和价值参与与 \mathbf{q}_t 的注意力计算, 如Equation (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, 并且可能占主导地位于学习过程中, 这可能会阻止模型有效地从压缩 (compression) 和选择 (selection) 标记 (Token) 中学习。为了解决这个问题, 我们引入一个专用的滑动窗口分支, 明确处理局部上下文, 从而使其他分支 (压缩和选择) 能够专注于学习各自的特征, 而不被局部模式所影响。具体来说, 我们在窗口 w 中维护近期的标记 (Token) $\tilde{\mathbf{K}}_t^{\text{win}} = \mathbf{k}_{t-w:t}$, $\tilde{\mathbf{V}}_t^{\text{win}} = \mathbf{v}_{t-w:t}$, 并将不同信息源 (压缩标记 (Token)、被选择标记 (Token)、滑动窗口) 的注意力计算隔离到不同的分支中。然后通过学习的门控机制将这些分支输出聚合在一起。

为了进一步防止注意力分支之间的捷径学习, 同时保持计算开销很小, 我们为三个分支提供独立的键 (keys) 和值 (values)。这种架构设计通过防止局部和长范围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 (keys) 和值 (values) ($\tilde{\mathbf{K}}_t^{\text{cmp}}, \tilde{\mathbf{V}}_t^{\text{cmp}}; \tilde{\mathbf{K}}_t^{\text{slc}}, \tilde{\mathbf{V}}_t^{\text{slc}}; \text{和 } \tilde{\mathbf{K}}_t^{\text{win}}, \tilde{\mathbf{V}}_t^{\text{win}}$) 之后, 我们按照 Equation (5) 计算最终的注意力输出。结合上述压缩 (compression)、选择 (selection) 和滑动窗口机制, 这形成了 NSA 的完整算法框架。

3.4. Kernel Design

为了实现与FlashAttention相当的训练和预填充速度提升, 我们在Triton上实现了硬件对齐的稀疏注意力内核。鉴于多头注意力 (MHA) 在解码时内存密集且效率低下, 我们专注于具有共享键值缓存 (KV caches) 的架构, 如当前最先进的大型语言模型 (LLMs) 中的GQA和MQA。尽管压缩和滑动窗口注意力计算与现有的FlashAttention-2内核完全兼容, 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将时间上连续的查询块加载到SRAM的策略，将导致低效的内存访问，因为一个块中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一种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。Figure 3 说明了我们的前向传播实现。

所提出的内核架构的关键特征如下：

1. 以组为中心的数据加载. 对于每个内部循环，在位置 t 加载组中所有头的查询 $Q \in \mathbb{R}^{[h, d_k]}$ 及其共享的稀疏键/值块索引 \mathcal{I}_t .
2. 共享 KV 获取. 在内部循环中，顺序地将由 \mathcal{I}_t 索引的连续键/值块加载到 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 $B_k \mid l'$ 的核块大小。
3. 网格上的外循环. 由于内循环长度（与所选块数 n 成正比）在不同的查询块中几乎相同，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）通过组间共享消除冗余的KV传输，以及（2）在GPU流式多处理器之间平衡计算负载，从而实现接近最优的算术强度（arithmetic intensity）。

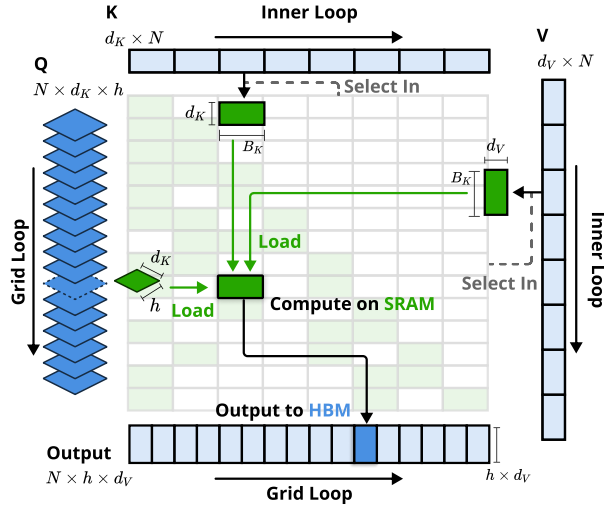


Figure 3 | 内核设计用于 NSA。内核通过 GQA 组（网格循环）加载查询，获取相应的稀疏 KV 块（内部循环），并在 SRAM 上执行注意力计算。绿色块表示 SRAM 上的数据，而蓝色表示 HBM 上的数据。

4. Experiments

我们通过三个方面评估 NSA：(1) 通用基准性能，(2) 长上下文基准性能，和 (3) 连锁推理性能，与完整注意力基线和最先进的稀疏注意力方法进行比较。我们将稀疏计算范式的效率分析推迟到 Section 5，在那里我们提供有关训练和推理速度的详细讨论。

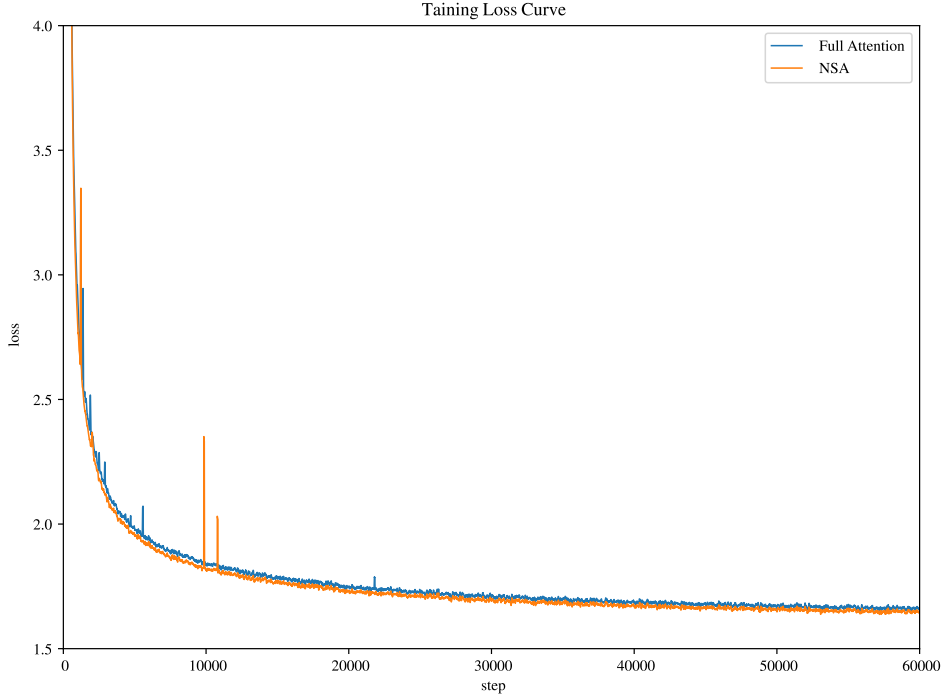


Figure 4 | 在27B参数模型上，完全注意力与我们的 NSA 的预训练损失比较。两个模型都表现出稳定的收敛性，而 NSA 实现了更低的损失值。

4.1. Pretraining Setup

根据最新的LLM（大语言模型）的常用做法，我们的实验采用了结合了分组查询注意力（Grouped-Query Attention, GQA）和专家混合（Mixture-of-Experts, MoE）的基础架构，具有27B的总参数和3B的激活参数。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，总共有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 et al., 2024; DeepSeek-AI, 2024)结构，拥有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由SwiGLU形式的MLP替换。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 $d = 16$ ，选择块大小 $l' = 64$ ，选择块数量 $n = 16$ （包括固定激活的1个初始块和2个局部块），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长度文本的标记上进行了预训练，然后在32k长度文本上使用YaRN (Peng et al., 2024)进行了继续训练和监督微调，以实现长上下文适应。两个模型均训练至完全收敛，以确保公平比较。如 Figure 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，其中NSA始终优于全注意力模型。

4.2. Baselines Methods

除了与全注意力（Full Attention）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。这些方法首先计算完整的注意力得分，然后选择与每个查询对应的前 n 个得分钥匙，然后

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

Table 1 | 全注意力基线和 NSA 在知识 (MMLU, MMLU-PRO, CMMLU)、推理 (BBH, GSM8K, MATH, DROP) 和编码 (MBPP, HumanEval) 任务的普遍基准上的预训练性能比较。尽管稀疏性较高, NSA 在大多数基准上实现了优越的平均性能。

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	<u>0.316</u>	0.291	0.810	0.548	0.156	0.423
Full Attn	0.512	<u>0.623</u>	<u>0.409</u>	<u>0.350</u>	<u>0.305</u>	0.324	<u>0.294</u>	<u>0.830</u>	0.560	<u>0.163</u>	<u>0.437</u>
NSA	<u>0.503</u>	0.624	0.432	0.437	0.356	0.307	0.341	0.905	<u>0.550</u>	0.232	0.469

Table 2 | 我们的方法 (NSA) 与 LongBench 上的基准模型在单文档问答、多个文档问答、合成任务和代码任务类别的子集中的性能比较。我们的 NSA 超越了大多数基准模型, 包括全注意力 (Full Attention)。

在这些位置计算注意力。这些方法涵盖了多种稀疏注意力的范式, 包括 KV-cache 驱逐、查询感知选择和精确的前- n 稀疏选择。

对于一般评估, 其中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, 这些方法在效果上等同于全注意力。因此, 在这种设置下, 我们仅展示 NSA 和全注意力基线之间的比较结果。在长上下文评估中, 我们对所有基线方法进行了比较, 所有稀疏注意力方法的稀疏性均设置为相同, 以确保公平比较。对于需要长文本监督微调的推理评估 (chain-of-thought reasoning), 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. Performance Comparison

总体评估. 我们在涵盖知识、推理和编程能力的全面基准测试套件上评估了预训练的 NSA 和完全注意力 (Full Attention) 基线, 包括 MMLU (Hendrycks et al., 2020)、MMLU-PRO (Wang et al., 2024)、CMMLU (Li et al., 2023)、BBH (Suzgun et al., 2022)、GSM8K (Cobbe et al., 2021)、MATH (Hendrycks et al., 2020)、DROP (Dua et al., 2019)、MBPP (Austin et al., 2021) 和 HumanEval (Chen et al., 2021)。结果显示在 Table 1 中。尽管存在稀疏性, NSA 在总体性能上表现优越, 在 9 个指标中的 7 个上超越了所有基线, 包括完全注意力 (Full Attention)。这表明, 虽然 NSA 可能未能充分利用其在较短序列上的效率优势, 但其表现仍然强劲。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增益 (DROP: +0.042, GSM8K: +0.034), 这表明我们的预训练帮助模型开发了专业的注意力机制。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, 可能通过过滤来自无关注意力通路的噪声来提高性能。在多种评估中的一致表现也验证了 NSA 作为通用架构的鲁棒性。

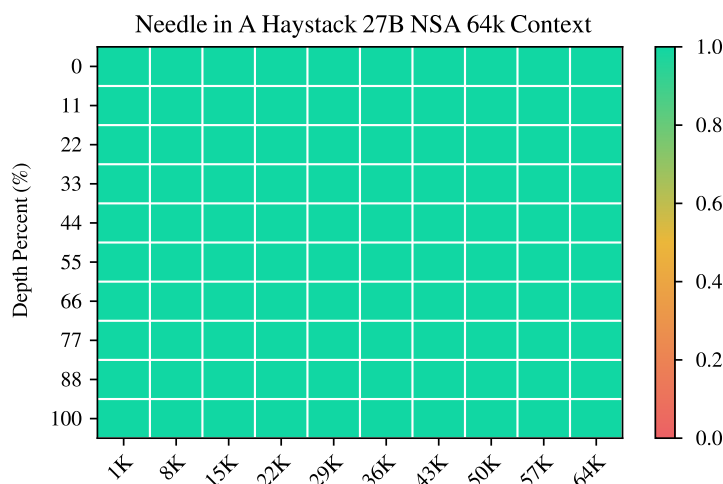


Figure 5 | 在64k上下文长度下，各个上下文位置的针在干草堆中检索精度。NSA 通过其分层稀疏注意力设计实现了完美的精度。

长上下文评估。如 Figure 5 所示，NSA 在 64k 上下文的找针（needle-in-a-haystack）(Kamradt, 2023) 测试中，在所有位置上均实现了完美的检索准确性。该性能源于我们的分层稀疏注意力设计，它结合了用于高效全局上下文扫描的压缩标记（compression tokens）和用于精确本地信息检索的选择标记（selection tokens）。粗粒度压缩以低计算成本识别相关上下文块，而对所选标记的标记级注意力确保了关键细粒度信息的保留。该设计使 NSA 能够保持全局意识和本地精度。

我们还在 LongBench (Bai et al., 2023) 上评估了 NSA，与最先进的稀疏注意方法和完整注意基线进行对比。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标记（Token）设置为 2560 个，这对应于 NSA 在处理 32k 序列长度时激活的标记（Token）的平均数量。按照 StreamLLM (Xiao et al., 2023) 的方法，这个标记（Token）预算包含前 128 个标记（Token）和 512 个局部标记（Token）。由于某些子集在所有模型中的得分很低，可能无法提供有意义的比较，我们从 LongBench 中排除了这些子集。如 Table 2 所示，NSA 实现了最高的平均得分 0.469，超过了所有基线（相比完整注意增加了 +0.032，相比 Exact-Top 增加了 +0.046）。这一改善源于两个关键创新：（1）我们的本地稀疏注意设计，这使得在预训练过程中能够对稀疏模式进行端到端优化，促进了稀疏注意模块与其他模型组件之间的同步适应；（2）分层稀疏注意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）中相比完整注意取得了 +0.087 和 +0.051 的提升，在代码理解（LCC: +0.069）上超越了基线，并且在段落检索（PassR-en: +0.075）上表现优于其他方法。这些结果验证了 NSA 处理多样化长上下文挑战的能力，其本地预训练的稀疏注意为学习任务最优模式提供了额外的好处。

思维链推理评估。为了评估 NSA 与高级下游训练范式的兼容性，我们调查了其通过后期训练获取思维链数学推理能力的的能力。考虑到强化学习在小规模模型上的有限有效性，我们采用

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

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

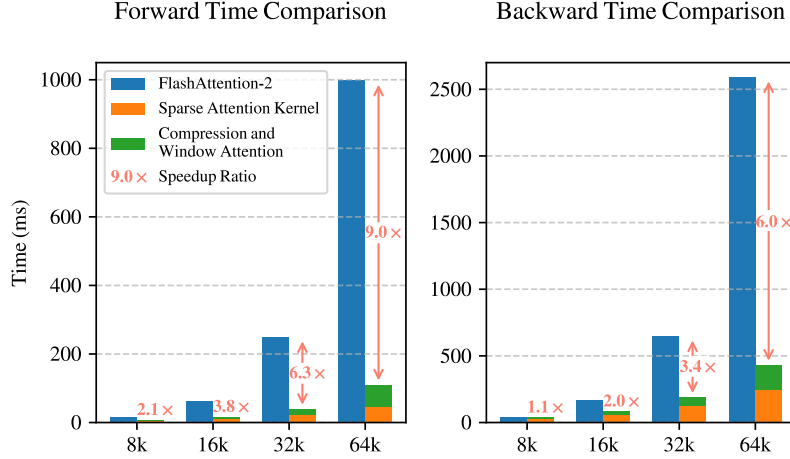


Figure 6 | 比较基于Triton的 NSA 内核与基于Triton的FlashAttention-2内核。我们的实现显著减少了所有上下文长度的延迟，随着输入长度的增加，改进变得更加明显。

了来自 DeepSeek-R1 的知识蒸馏，使用 10B 个长为 32k 的数学推理轨迹进行监督微调（SFT）。这产生了两个可比模型：Full Attention-R（完整注意基线）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 个响应并获得平均得分。为验证推理深度的影响，我们进行了一些实验，设置了两个生成上下文限制：8k 和 16k 个标记（Token），以测量延长的推理链是否提高准确性。模型预测的示例比较见 Appendix A。

如Table 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)，而且在16k上下文下这一优势仍然存在 (+0.054)。这些结果验证了原生稀疏注意力的两个关键优势：（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远程逻辑依赖关系；（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优越性确认了稀疏注意力在训练流程中原生集成时对复杂推理任务的可行性。

5. Efficiency Analysis

我们在一个8-GPU A100系统上评估 NSA 相对于全注意力（Full Attention）的计算效率。在效率分析中，我们还配置了模型，GQA组 $g = 4$ ，每组头数 $h = 16$ ，查询/键维度 $d_k = 192$ ，和值的维度 $d_v = 128$ 。根据 Section 4 中的相同设置，我们设置 NSA 的压缩块大小 $l = 32$ ，滑动步幅 $d = 16$ ，选择的块大小 $l' = 64$ ，选择的块计数 $n = 16$ ，滑动窗口大小 $w = 512$ 。

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×

Table 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（Token）数量计）。由于解码的算术强度低且受内存限制，预期的加速大致与内存访问量呈线性关系。

5.1. Training Speed

我们比较了基于Triton的NSA注意力和全注意力（Full Attention）与基于Triton的FlashAttention-2的实现，以确保在相同后端下进行公平的速度比较。如Figure 6所示，随着上下文长度的增加，我们的NSA实现了逐步更大的加速，64k上下文长度时的前向加速为9.0×，反向加速为6.0×。值得注意的是，速度优势在更长的序列中变得更加明显。这种加速源于我们为稀疏注意力架构最大化效率而设计的硬件对齐算法：（1）块状内存访问模式通过合并加载最大化了张量核心（Tensor Core）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. Decoding Speed

解码速度的主要决定因素是内存访问瓶颈，这与KV缓存加载量密切相关。在每个解码步骤中，我们的NSA最大只需要加载 $\lfloor \frac{s-l}{d} \rfloor$ 个压缩标记（Token）、 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 Table 4 所示，随着解码长度的增加，我们的方法在延迟方面表现出显著减少，在64k上下文长度时实现高达11.6× 的加速。内存访问效率的这一优势在更长的序列中也得到了增强。

6. Discussion

在本节中，我们回顾了NSA的开发过程，并讨论了我们在探索（exploration）不同稀疏注意力策略时获得的关键见解。尽管我们的方法展示了令人鼓舞的结果，但理解与替代策略相关的挑战以及分析注意力模式为未来的研究方向提供了有价值的背景。我们首先检查了激励我们设计选择的替代标记（Token）选择策略的挑战，随后提供了可视化，揭示了注意力分布模式的见解。

6.1. Challenges with Alternative Token Selection Strategies

在设计 NSA 之前，我们探索了将现有稀疏注意方法适应训练阶段。然而，这些尝试遇到了各种挑战，这促使我们设计了不同的稀疏注意架构：

基于关键簇的策略。我们研究了基于簇的策略，如 ClusterKV (Liu et al., 2024)。这些方法将来自同一簇的键（Keys）和价值（Values）存储在连续的内存区域中。尽管理论上在训练和推理中是可行的，但它们面临三个显著的挑战：（1）动态聚类机制引入的非平凡计算开销；（2）

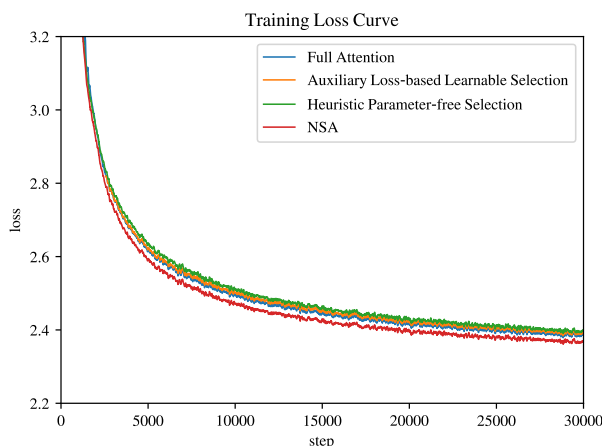


Figure 7 | 比较在一个具有3B参数的模型（model）上使用全注意力（Full Attention）和不同标记（Token）选择策略的训练损失。我们的NSA实现了更好的性能。

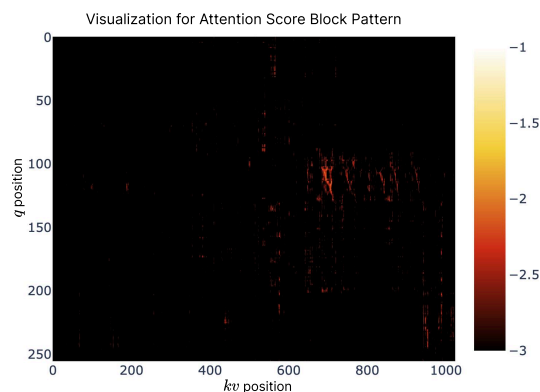


Figure 8 | 在全注意力transformer上可视化注意力图。浅色区域表示更高的注意力价值（value）。如图所示，注意力分数呈现块状簇（cluster）分布。

由于簇间不平衡导致的操作优化困难，尤其是在专家混合（Mixture-of-Experts, MoE）系统中，歪斜的专家并行（Expert Parallelism, EP）组执行时间导致持续的负载不平衡；（3）由于需要强制周期性重新聚类 and 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。

其他块状选择策略. 我们还考虑了与 NSA 不同的块状关键字、价值（value）选择策略，例如 Quest (Tang et al., 2024) 和 InfLLM (Xiao et al., 2024)。这些方法依赖于为每个块计算重要性分数，并根据与 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的，基于神经网络的重要性分数计算依赖于辅助损失，增加了操作开销，且通常降低了模型性能；（2）无参数的启发式重要性分数计算策略的召回率低，导致次优性能。我们 在一个具有类似架构的 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（Full Attention）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字，以估计块的重要性分数。这些分数通过原始查询和每个块内的关键字之间的平均注意力分数进行监督。对于无参数的启发式选择方法，我们遵循 Quest 的策略，使用查询与关键块的坐标最小-最大之间的乘积来实现直接选择，而不引入额外的参数。我们还探索了一种冷启动训练方法，即在转移到启发式块状选择之前，首先对前 1000 步应用全注意力。正如在 Figure 7 中所示，这两种方法表现出较差的损失。

6.2. Visualization

为了探索 transformer（transformer）注意力分布中的潜在模式并为我们的设计寻求灵感，我们在 Figure 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簇（cluster）特征，附近的标记（Token）通常显示相似的注意力分数。这个观察促使我们设计了 NSA，这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。块状簇现象表明，序列中相邻的标记（Token）可能与查询标记（Token）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。这个观察激励我们探索一种稀疏注意力机制，该机制在连续标记（Token）块上操作，而不是单个标记（Token），旨在提高计算效率并保留高注意力模式。

7. Related Works

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）动态标记（Token）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. Fixed Sparse Pattern

滑动窗口（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（attention）。StreamingLLM (Xiao et al., 2023) 通过维护上下文的关键部分：一个注意力接收器（attention sink）（早期标记（early tokens））和一个局部上下文窗口，来解决处理长文本流的挑战。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在需要全面理解上下文的任务上的性能。

7.2. Dynamic Token Pruning

H2O (Zhang et al., 2023b) 实现了一种自适应方法，以在解码过程中减少 KV 缓存内存使用。该方法根据注意力得分动态驱逐被视为对未来预测不那么重要的标记（Token），基于其最近的有效性。SnapKV (Li et al., 2024) 还引入了一种标记（Token）剪枝策略，通过有选择性地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过程中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相结合来更新 KV 缓存，以保持提示的一致性。

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。结果得分有助于选择前 n 个重要的关键-价值（key-value）块进行注意力（attention）计算。InfLLM (Xiao et al., 2024) 通过维护注意力汇聚点、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关键（key）来评估块的重要性。HashAttention (Desai et al., 2024) 将关键标记（token）识别表述为推荐问题，通过使用学习函数将查询与关键映射到汉明空间（Hamming space）。ClusterKV (Liu et al., 2024) 首先通过聚类关键（key）来实现稀疏性，然后基于查询-簇（cluster）相似性选择最相关的簇进行注意力计算。

8. Conclusion

我们提出了 NSA，一种与硬件对齐的稀疏注意力架构，用于高效的长上下文建模。通过将层次化标记（Token）压缩与可训练架构中的块状标记（Token）选择相结合，我们的架构在保持完全注意力（Full Attention）性能的同时，实现了加速的训练和推理。NSA 通过展示一般基准性能与完全注意力基线（full-attention baselines）匹配，在长上下文评估中超越建模能力，并增强推理能力，所有这些都伴随着可测量的计算延迟降低和显著的加速。

References

- J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, and S. Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. [arXiv preprint arXiv:2305.13245](https://arxiv.org/abs/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](https://arxiv.org/abs/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. Long-bench: A bilingual, multitask benchmark for long context understanding. [arXiv preprint arXiv:2308.14508](https://arxiv.org/abs/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](https://arxiv.org/abs/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](https://arxiv.org/abs/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](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](https://arxiv.org/abs/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](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](https://arxiv.org/abs/2501.12948).

- A. Desai, S. Yang, A. Cuadron, A. Klimovic, M. Zaharia, J. E. Gonzalez, and I. Stoica. Hashattention: 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. 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. Riche, editors, [Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, UIST 2023, San Francisco, CA, USA, 29 October 2023– 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. Inllm: 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 \times mapreduce: Simplified long-sequence processing using large language models. arXiv preprint arXiv:2410.09342, 2024.

A. Example of AIME Results

Prompt:

“|用户| 设 x, y 和 z 为满足以下方程组的正实数:

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

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

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

然后 $|\log_2(x^4y^3z^2)|$ 的价值 (value) 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 \square 内。|Assistant|

Our Result:

<think>...</think> (2275 标记 (Token))

给定以下方程组:

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

我们将这些对数方程转换为指数形式:

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

用 x 、 y 和 z 表示每个变量:

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

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

接下来, 我们计算 $x^4y^3z^2$:

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

将它们相乘:

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

Taking the logarithm base 2:

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

The absolute value is:

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

因此, $m = 25$ 和 $n = 8$, 从而得到 $m + n = 33$ 。

33

Baseline Result:

<think>...</think> (9392 标记 (Token))

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:

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

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

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

将这些互相代入, 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，我们意识到这个系统可能没有解。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 (Cramer's rule) 和矩阵求逆来解这个系统。矩阵的行列式 (determinant) 是 -2，解为：

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

Thus, we have:

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

我们需要计算 $|\log_2(x^4 y^3 z^2)|$ ：

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

转换为公分母 (48)：

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

Summing these:

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

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

Prompt:

“|User | Real numbers x and y with $x, y > 1$ satisfy $\log_x(y^x) = \log_y(x^{4y}) = 10$.
What is the value of xy ?
Please reason step by step, and put your final answer within \square . |Assistant| ”

Our Result:

$\langle \text{think} \rangle \dots \langle / \text{think} \rangle$ (15147 标记 (Token))

给定以下方程:

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:

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

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

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

For the second equation:

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

使用幂规则，我们得到：

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

由于 $\log_y(x) = z$ ，这可以简化为：

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

我们现在有两个表达式：

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

将 $x = 10z$ 和 $y = \frac{5/2}{z}$ 代入乘积 xy 中，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 的价值 (value) 是 25。

Baseline Result:

<think>...</think> (16223 标记 (Token))

给定问题：满足方程 $\log_x(y^x) = 10$ 和 $\log_y(x^4y) = 10$ 的实数 x 和 y （其中 $x, y > 1$ ），我们需要找到 xy 的价值 (value)。

首先，我们使用换底公式将对数方程转换为其指数形式。

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, 我们用相互的形式表示 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 $y = 2^b$ 进行设定后, 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:

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