

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

Long-context modeling is crucial for next-generation language models, yet the high computational cost of standard attention mechanisms poses significant computational challenges. Sparse attention offers a promising direction for improving efficiency while maintaining model capabilities. We present NSA, a Natively trainable Sparse Attention mechanism that integrates algorithmic innovations with hardware-aligned optimizations to achieve efficient long-context modeling. NSA employs a dynamic hierarchical sparse strategy, combining coarse-grained token compression with fine-grained token selection to preserve both global context awareness and local precision. Our approach advances sparse attention design with two key innovations: (1) We achieve substantial speedups through arithmetic intensity-balanced algorithm design, with implementation optimizations for modern hardware. (2) We enable end-to-end training, reducing pretraining computation without sacrificing model performance. As shown in Figure 1, experiments show the model pretrained with NSA maintains or exceeds Full Attention models across general benchmarks, long-context tasks, and instruction-based reasoning. Meanwhile, NSA achieves substantial speedups over Full Attention on 64k-length sequences across decoding, forward propagation, and backward propagation, validating its efficiency throughout the model lifecycle.

1. Introduction

The research community increasingly recognizes long-context modeling as a crucial capability for next-generation large language models, driven by diverse real-world applications ranging from in-depth reasoning (DeepSeek-AI, 2025; Zelikman et al., 2022), repository-level code generation (Zhang et al., 2023a; Zhang et al.) and multi-turn autonomous agent systems (Park et al., 2023). Recent breakthroughs, including OpenAI’s o-series models, DeepSeek-R1 (DeepSeek-

^{*}Contribution during internship at DeepSeek-AI.

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/。项目在线体验地址 <https://auth.gpt-academic.top/>。当前大语言模型: Qwen2.5-72B-Instruct，当前语言模型温度设定: 0.3。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。

长上下文建模对于下一代语言模型至关重要，然而，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，这是一种 Natively trainable Sparse Attention 机制，通过将算法创新与硬件对齐的优化相结合，实现高效的长上下文建模。NSA 采用动态分层稀疏策略，结合粗粒度的标记压缩与细粒度的标记选择，以保持全局上下文感知和局部精度。我们的方法通过两项关键创新推进了稀疏注意力设计：(1) 通过算术强度平衡的算法设计实现显著加速，并针对现代硬件进行实现优化。(2) 实现端到端训练，减少预训练计算而不牺牲模型性能。如 Figure 1 所示，实验表明使用 NSA 预训练的模型在通用基准、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，NSA 在 64k 长度的序列上，无论是在解码、前向传播还是反向传播中，都比全注意力模型实现了显著的加速，验证了其在整个模型生命周期中的效率。

1. Introduction

研究社区越来越认识到长上下文建模是下一代大型语言模型的关键能力，这由从深入推理 (DeepSeek-AI, 2025; Zelikman et al., 2022)、仓库级代码生成 (Zhang et al., 2023a; Zhang et al.) 到多轮自主代理系统 (Park et al., 2023) 的各种实际应用所驱动。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)，使模型能够处理整个代码库、长文档，维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，传统的注意力机制 (Vaswani et al., 2017) 的高复杂性 (Zaheer et al., 2020) 成为关键的延迟瓶颈。理论估计表明，当解码 64k 长度的上下文时，带有 softmax 架构的注意

^{*}Contribution during internship at DeepSeek-AI.

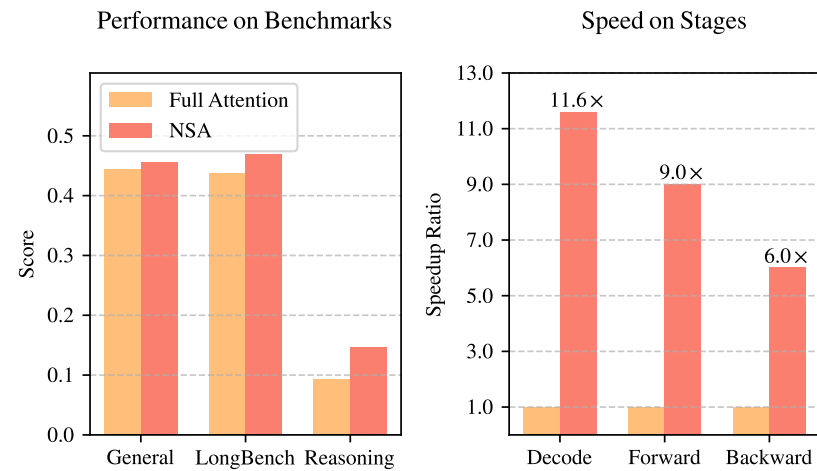


Figure 1 | Comparison of performance and efficiency between Full Attention model and our NSA. Left: Despite being sparse, NSA surpasses Full Attention baseline on average across general benchmarks, long-context tasks, and reasoning evaluation. Right: For 64k-length sequence processing, NSA achieves substantial computational speedup compared to Full Attention in all stages: decoding, forward propagation, and backward propagation.

AI, 2025), and Gemini 1.5 Pro (Google et al., 2024), enabling models to process entire codebases, lengthy documents, maintain coherent multi-turn conversations over thousands of tokens, and perform complex reasoning across long-range dependencies. However, the high complexity (Zaheer et al., 2020) of vanilla Attention (Vaswani et al., 2017) mechanisms emerges as a critical latency bottleneck as sequence length increases. Theoretical estimates indicate that attention computation with softmax architectures accounts for 70–80% of total latency when decoding 64k-length contexts, underscoring the urgent need for more efficient attention mechanisms.

A natural approach to efficient long-context modeling is to take advantage of the inherent sparsity of softmax attention (Ge et al., 2023; Jiang et al., 2023), where selectively computing critical query-key pairs can significantly reduce computational overhead while preserving performance. Recent advances demonstrate this potential through diverse strategies: KV-cache eviction methods (Li et al., 2024; Zhang et al., 2023b; Zhou et al., 2024), blockwise KV-cache selection methods (Tang et al., 2024; Xiao et al., 2024), and sampling, clustering or hashing-based selection methods (Chen et al., 2024; Desai et al., 2024; Liu et al., 2024). Despite these promising strategies, existing sparse attention methods often fall short in practical deployments. Many approaches fail to achieve speedups comparable to their theoretical gains; also, most methods mainly focus on inference stage, lacking effective training-time support to fully exploit the sparsity patterns of attention.

To address these limitations, the deployment of effective sparse attention must tackle two key challenges: (1) **Hardware-aligned inference speedup**: Converting theoretical computation reductions into actual speed improvements requires hardware-friendly algorithm design during

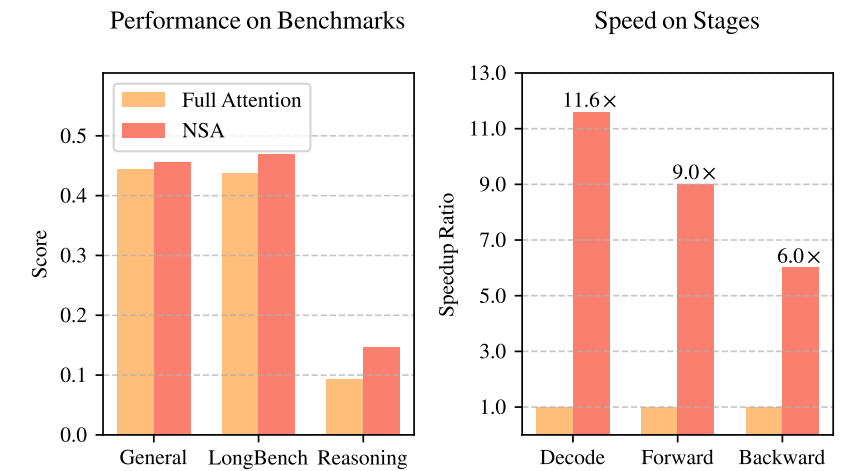


Figure 1 | Full Attention 模型与我们的 NSA 在性能和效率上的比较。左：尽管是稀疏的，NSA 在一般的基准测试、长上下文任务和推理评估中平均表现超过了 Full Attention 基线。右：对于 64k 长度的序列处理，NSA 在所有阶段（解码、前向传播和反向传播）中都实现了相对于 Full Attention 的显著计算加速。

力计算占总延迟的 70-80%，这突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 注意力的内在稀疏性 (Ge et al., 2023; Jiang et al., 2023)，通过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种潜力：KV 缓存淘汰方法 (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)。尽管这些策略很有前景，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，无法充分利用注意力的稀疏模式。

为了解决这些局限性，有效的稀疏注意力部署必须应对两个关键挑战：(1) 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) 训练感知的算法设计：通过可训练的操作器实现端到端计算，减少训练成本同时保持模型性能。这些要求对于实际应用中实现快速长上下文推理或训练至关重要。考虑这两个方面时，现有方法仍存在明显的差距。

为了实现更有效和高效的稀疏注意力，我们提出了 NSA，这是一种集成了层次化标记建模的本机可训练稀疏注意力架构。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进行处理来减少每个查询的计算：压缩的粗粒度标记、选择性保留的细粒度标记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化其实用效率。NSA 引入了两个核心创新，对应上述关键要求：(1) 硬件对齐系统：优化块状稀疏注意力以利用 Tensor Core 和内存访问，确保平衡的算术强度。(2) 训练感知设计：通过高效的算法和反向操作器实现稳定的端到端训练。这种优化使 NSA 能够支持高效的部署和端到端训练。

both prefilling and decoding stages to mitigate memory access and hardware scheduling bottlenecks; (2) *Training-aware algorithm design*: Enabling end-to-end computation with trainable operators to reduce training costs while maintaining model performance. These requirements are crucial for real-world applications to achieve fast long-context inference or training. When considering both aspects, existing methods still exhibit a noticeable gap.

To achieve more effective and efficient sparse attention, we present NSA, a Natively trainable Sparse Attention architecture that integrates hierarchical token modeling. As shown in Figure 2, NSA reduces per-query computation by organizing keys and values into temporal blocks and processing them through three attention paths: compressed coarse-grained tokens, selectively retained fine-grained tokens, and sliding windows for local contextual information. Then we implement specialized kernels to maximize its practical efficiency. NSA introduces two core innovations corresponding to the key requirements above: (1) Hardware-aligned system: Optimize blockwise sparse attention for Tensor Core utilization and memory access, ensuring balanced arithmetic intensity. (2) Training-aware design: Enable stable end-to-end training through efficient algorithms and backward operators. This optimization enables NSA to support both efficient deployment and end-to-end training.

We evaluate NSA through comprehensive experiments on real-world language corpora. Pretraining on a 27B-parameter transformer backbone with 260B tokens, we assess NSA’s performance across general language evaluations, long-context evaluations, and chain-of-thought reasoning evaluation. We further compare the kernel speed on A100 GPUs with optimized Triton (Tillet et al., 2019) implementations. Experimental results demonstrate that NSA achieves comparable or superior performance to full attention baseline, while outperforming existing sparse attention approaches. Additionally, NSA delivers substantial speedups across decoding, forward, and backward stages compared to Full Attention, with the speedup ratio increasing for longer sequences. These results validate that our hierarchical sparse attention design effectively balances model capability and computational efficiency.

2. Rethinking Sparse Attention Methods

Modern sparse attention methods have made significant strides in reducing the theoretical computational complexity of transformer models. However, most approaches predominantly apply sparsity during inference while retaining a pretrained Full Attention backbone, potentially introducing architectural bias that limits their ability to fully exploit sparse attention’s advantages. Before introducing our native sparse architecture, we systematically analyze these limitations through two critical lenses.

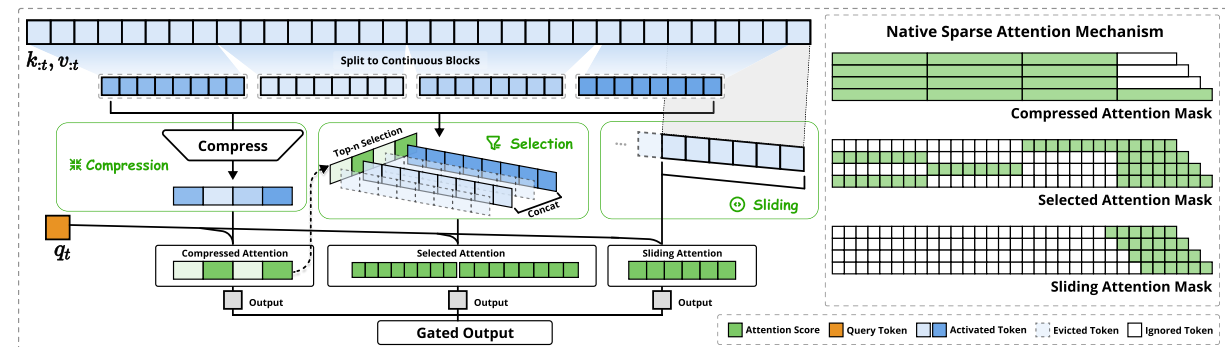


Figure 2 | NSA 架构概述。左：框架通过三个并行的注意力分支处理输入序列：对于给定的查询，前序的键和值被处理成压缩注意力以捕捉粗粒度模式，选择性注意力以关注重要令牌块，以及滑动注意力以处理局部上下文。右：每个分支产生的不同注意力模式的可视化。绿色区域表示需要计算注意力得分的区域，而白色区域表示可以跳过的区域。

我们通过在真实语言语料库上的全面实验评估了 NSA。在 270 亿参数的变压器骨干上预训练 2600 亿个标记后，我们在通用语言评估、长上下文评估和链式思维推理评估中评估了 NSA 的性能。我们进一步在 A100 GPU 上与优化的 Triton (Tillet et al., 2019) 实现进行了内核速度的比较。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、前向和反向阶段都实现了显著的加速，且加速比随着序列长度的增加而增加。这些结果验证了我们的层次化稀疏注意力设计在平衡模型能力和计算效率方面的有效性。

2. Rethinking Sparse Attention Methods

现代稀疏注意力方法在降低变压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，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。在介绍我们原生的稀疏架构之前，我们通过两个关键视角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，许多方法未能在推理延迟上实现相应的减少，主要由于两个挑战：

阶段限制的稀疏性。例如 H2O (Zhang et al., 2023b) 等方法在自回归解码期间应用稀疏性，但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，如 MInference (Jiang et al., 2024) 等方法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全注意力相当。阶段专业化减少了这些方法在预填充主导的工作负载（如书籍摘要和代码完成）或解码主导的工作负载（如长链思考 (Wei et al., 2022) 推理）中的加速能力。

与先进注意力架构的不兼容性。一些稀疏注意力方法无法适应现代解码高效的架构，如

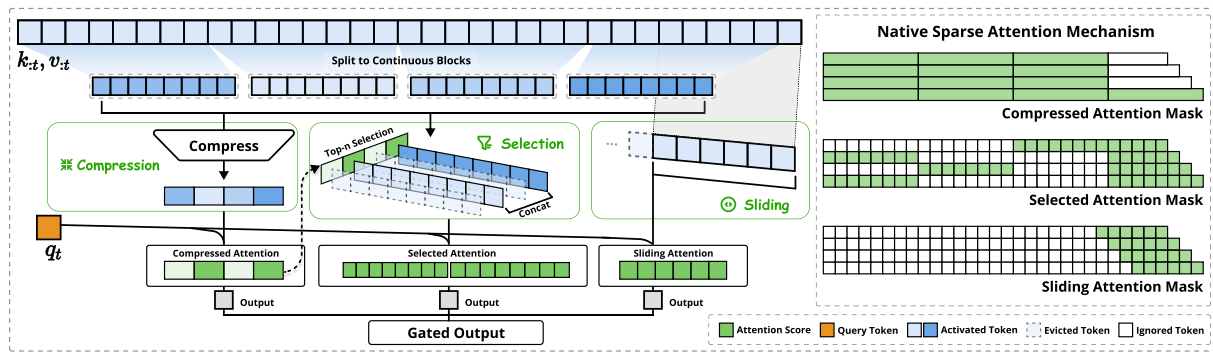


Figure 2 | Overview of NSA’s architecture. Left: The framework processes input sequences through three parallel attention branches: For a given query, preceding keys and values are processed into compressed attention for coarse-grained patterns, selected attention for important token blocks, and sliding attention for local context. Right: Visualization of different attention patterns produced by each branch. Green areas indicate regions where attention scores need to be computed, while white areas represent regions that can be skipped.

2.1. The Illusion of Efficient Inference

Despite achieving sparsity in attention computation, many methods fail to achieve corresponding reductions in inference latency, primarily due to two challenges:

Phase-Restricted Sparsity. Methods such as H2O (Zhang et al., 2023b) apply sparsity during autoregressive decoding while requiring computationally intensive pre-processing (e.g. attention map calculation, index building) during prefilling. In contrast, approaches like MInference (Jiang et al., 2024) focus solely on prefilling sparsity. These methods fail to achieve acceleration across all inference stages, as at least one phase remains computational costs comparable to Full Attention. The phase specialization reduces the speedup ability of these methods in prefilling-dominated workloads like book summarization and code completion, or decoding-dominated workloads like long chain-of-thought (Wei et al., 2022) reasoning.

Incompatibility with Advanced Attention Architecture. Some sparse attention methods fail to adapt to modern decoding efficient architectures like Multiple-Query Attention (MQA) (Shazeer, 2019) and Grouped-Query Attention (GQA) (Ainslie et al., 2023), which significantly reduced the memory access bottleneck during decoding by sharing KV across multiple query heads. For instance, in approaches like Quest (Tang et al., 2024), each attention head independently selects its KV-cache subset. Although it demonstrates consistent computation sparsity and memory access sparsity in Multi-Head Attention (MHA) models, it presents a different scenario in models based on architectures like GQA, where the memory access volume of KV-cache corresponds to the union of selections from all query heads within the same GQA group. This architectural characteristic means that while these methods can reduce computation operations, the required KV-cache memory access remains relatively high. This limitation forces a critical choice: while some sparse attention methods reduce computation, their scattered memory access

Multiple-Query Attention (MQA) (Shazeer, 2019) 和 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 显著减少了解码期间的内存访问瓶颈。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。虽然它在多头注意力 (MHA) 模型中展示了持续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, 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。这一限制迫使一个关键选择: 虽然一些稀疏注意力方法减少了计算, 但其分散的内存访问模式与先进架构的高效内存访问设计相冲突。

这些限制的出现是因为许多现有的稀疏注意力方法专注于 KV 缓存减少或理论上的计算减少, 但在先进的框架或后端中难以实现显著的延迟减少。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. The Myth of Trainable Sparsity

我们追求原生可训练的稀疏注意力是基于对仅推理方法的分析得出的两个关键见解: (1) 性能下降: 事后应用稀疏性迫使模型偏离其预训练的优化轨迹。正如 Chen et al. (2024) 所展示的, 前 20% 的注意力只能覆盖总注意力分数的 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受到剪枝的影响。(2) 训练效率需求: 高效处理长序列训练对于现代大语言模型 (LLM) 的开发至关重要。这包括在更长的文档上进行预训练以增强模型容量, 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, 对训练中的计算挑战关注较少。这一局限性阻碍了通过高效训练开发更强大的长上下文模型。此外, 将现有稀疏注意力适应训练的努力也暴露出挑战:

不可训练的组件。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产生了不连续性。这些不可训练的组件阻止了通过令牌选择过程的梯度流动, 限制了模型学习最优稀疏模式的能力。

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问题。像 HashAttention (Desai et al., 2024) 这样的方法在注意力计算过程中需要从 KV 缓存中加载大量单独的令牌, 这导致了非连续的内存访问。这种非连续的内存访问阻止了快速注意力技术 (如 FlashAttention) 的有效适应, 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, 实现被迫退回到低硬件利用率, 显著降低了训练效率。

2.3. Native Sparsity as an Imperative

这些推理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, 这是一种原生的稀疏注意力框架, 解决了计算效率和训练需求的问题。在接下来的章节中, 我们将详细说明 NSA 的算法设计和操作符实现。

pattern conflicts with efficient memory access design from advanced architectures.

These limitations arise because many existing sparse attention methods focus on KV-cache reduction or theoretical computation reduction, but struggle to achieve significant latency reduction in advanced frameworks or backends. This motivates us to develop algorithms that combine both advanced architectural and hardware-efficient implementation to fully leverage sparsity for improving model efficiency.

2.2. The Myth of Trainable Sparsity

Our pursuit of native trainable sparse attention is motivated by two key insights from analyzing inference-only approaches: (1) *Performance Degradation*: Applying sparsity post-hoc forces models to deviate from their pretrained optimization trajectory. As demonstrated by Chen et al. (2024), top 20% attention can only cover 70% of the total attention scores, rendering structures like retrieval heads in pretrained models vulnerable to pruning during inference. (2) *Training Efficiency Demands*: Efficient handling of long-sequence training is crucial for modern LLM development. This includes both pretraining on longer documents to enhance model capacity, and subsequent adaptation phases such as long-context fine-tuning and reinforcement learning. However, existing sparse attention methods primarily target inference, leaving the computational challenges in training largely unaddressed. This limitation hinders the development of more capable long-context models through efficient training. Additionally, efforts to adapt existing sparse attention for training also expose challenges:

Non-Trainable Components. Discrete operations in methods like ClusterKV (Liu et al., 2024) (includes k-means clustering) and MagicPIG (Chen et al., 2024) (includes SimHash-based selecting) create discontinuities in the computational graph. These non-trainable components prevent gradient flow through the token selection process, limiting the model’s ability to learn optimal sparse patterns.

Inefficient Back-propagation. Some theoretically trainable sparse attention methods suffer from practical training inefficiencies. Token-granular selection strategy used in approaches like HashAttention (Desai et al., 2024) leads to the need to load a large number of individual tokens from the KV cache during attention computation. This non-contiguous memory access prevents efficient adaptation of fast attention techniques like FlashAttention, which rely on contiguous memory access and blockwise computation to achieve high throughput. As a result, implementations are forced to fall back to low hardware utilization, significantly degrading training efficiency.

3. Methodology

我们的技术方法涵盖了算法设计和内核优化。在以下小节中，我们首先介绍方法论的背景。然后我们介绍 NSA 的整体框架，接着是其关键算法组件。最后，我们详细说明了我们的硬件优化内核设计，该设计最大化了实际效率。

3.1. Background

注意力机制 在语言模型中被广泛使用，其中每个查询标记 \mathbf{q}_t 计算与所有先前的键 $\mathbf{k}_{:t}$ 的相关性得分，以生成值 $\mathbf{v}_{:t}$ 的加权和。正式地，对于长度为 t 的输入序列，注意力操作定义为：

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

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，注意力计算在总体计算成本中变得越来越重要，为长上下文处理带来了显著的挑战。

算术强度 是指计算操作与内存访问的比率。它内在决定了算法在硬件上的优化。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，计算为这两个硬件限制的比率。对于计算任务，算术强度高于这个关键阈值时，任务受计算限制（由 GPU FLOPS 限制），而低于这个阈值时，任务受内存带宽限制（由内存带宽限制）。

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计算限制。相比之下，自回归解码变得受内存带宽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，导致算术强度较低。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. Overall Framework

为了利用具有自然稀疏模式的注意力的潜力，我们建议用更紧凑和信息密集的代表键值对 $\tilde{\mathbf{K}}_t, \tilde{\mathbf{V}}_t$ 替换 Equation (1) 中的原始键值对 $\mathbf{k}_{:t}, \mathbf{v}_{:t}$ ，给定每个查询 \mathbf{q}_t 。具体来说，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 $\tilde{\mathbf{K}}_t, \tilde{\mathbf{V}}_t$ 是根据当前查询 \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

2.3. Native Sparsity as an Imperative

These limitations in inference efficiency and training viability motivate our fundamental redesign of sparse attention mechanisms. We propose NSA, a natively sparse attention framework that addresses both computational efficiency and training requirements. In the following sections, we detail the algorithmic design and operator implementation of NSA.

3. Methodology

Our technical approach spans algorithm design and kernel optimization. In the following subsections, we first introduce the background of our methodology. Then we present the overall framework of NSA, followed by its key algorithmic components. Finally, we detail our hardware-optimized kernel design that maximizes practical efficiency.

3.1. Background

Attention Mechanism is widely used in language modeling where each query token \mathbf{q}_t computes relevance scores against all preceding keys $\mathbf{k}_{:t}$ to generate a weighted sum of values $\mathbf{v}_{:t}$. Formally, for an input sequence of length t , the attention operation is defined as:

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

where Attn denotes the attention function:

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

Here, $\alpha_{t,i}$ represents the attention weight between \mathbf{q}_t and \mathbf{k}_i , and d_k is the feature dimension of keys. As sequence length increases, attention computation becomes increasingly dominant in the overall computational cost, presenting significant challenges for long-context processing.

Arithmetic Intensity is the ratio of compute operations to memory accesses. It intrinsically shapes algorithm optimization on hardware. Each GPU has a critical arithmetic intensity determined by its peak compute capability and memory bandwidth, calculated as the ratio of these two hardware limits. For computation tasks, arithmetic intensity above this critical threshold becomes compute-bound (limited by GPU FLOPS), while below it becomes memory-bound (limited by memory bandwidth).

Specifically for causal self-attention mechanism, during training and prefilling phases, batched matrix multiplications and attention computations exhibit high arithmetic intensity, making these stages compute-bound on modern accelerators. In contrast, auto-regressive decoding becomes memory-bandwidth constrained because it generates one token per forward

略来获得不同类别的 $\tilde{K}_t^c, \tilde{V}_t^c$, 并将其组合如下:

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

如图Figure 2所示, NSA有三种映射策略 $\mathcal{C} = \{\text{cmp}, \text{slc}, \text{win}\}$, 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 的门控分数, 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:

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

我们通过确保 $N_t \ll t$ 来保持高稀疏度。

3.3. Algorithm Design

在本小节中, 我们介绍我们的重映射策略 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为块级表示, 我们获得了捕获整个块信息的压缩键和值。形式上, 压缩键表示定义为:

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

其中 l 是块长度, d 是相邻块之间的滑动步长, 而 φ 是一个带有块内位置编码的可学习的 MLP, 用于将块中的键映射到一个压缩键。 $\tilde{K}_t^{\text{cmp}} \in \mathbb{R}^{d_k \times \lfloor \frac{t-l}{d} \rfloor}$ 是由压缩键组成的张量。通常, 我们采用 $d < l$ 以减轻信息碎片化。压缩值表示 \tilde{V}_t^{cmp} 也有类似的公式。压缩表示捕获了更粗粒度的高层次语义信息, 并减少了注意力计算的负担。

3.3.2. Token Selection

仅使用压缩键和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个别键和值。下面我们将描述我们的高效令牌选择机制, 该机制能够以较低的计算开销识别并保留最相关的令牌。

块选择。我们的选择策略以空间连续块的形式处理键和值序列, 这主要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。块选择对于在现代GPU上实现高效计算至关重要。这是因为现代GPU架构在连续块访问方面表现出显著更高的吞吐量, 而基于随机索引的读取则不然。此外, 块计算能够充分利用张量核心。这种架构特性已将块内存访问和计算确立为高性能注意力实现的基本原则, 正如FlashAttention的块设计所展示的那样。块选择遵循注意力分数的固有分布模式。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, 这表明相邻的键往往具有相似的重要性水平。我们在 Section 6.2 中的可视化也显示

pass while requiring loading the entire key-value cache, resulting in low arithmetic intensity. This leads to different optimization goals — reducing computation cost during training and prefilling, while reducing memory access during decoding.

3.2. Overall Framework

To leverage the potential of attention with natural sparse pattern, we propose replacing the original key-value pairs $\mathbf{k}_{:t}, \mathbf{v}_{:t}$ in Equation (1) with a more compact and information-dense set of representation key-value pairs \tilde{K}_t, \tilde{V}_t given each query \mathbf{q}_t . Specifically, we formally define the optimized attention output as follows:

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

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

where \tilde{K}_t, \tilde{V}_t are dynamically constructed based on the current query \mathbf{q}_t and the contextual memory $\mathbf{k}_{:t}, \mathbf{v}_{:t}$. We can design various mapping strategies to get different categories of $\tilde{K}_t^c, \tilde{V}_t^c$, and combine them as follows:

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

As illustrated in Figure 2, NSA have three mapping strategies $C = \{\text{cmp}, \text{slc}, \text{win}\}$, representing compression, selection, and sliding window for keys and values. $g_t^c \in [0, 1]$ is the gate score for corresponding strategy c , derived from input features via an MLP and sigmoid activation. Let N_t denote the total number of remapped keys/values:

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

We maintain a high sparsity ratio by ensuring $N_t \ll t$.

3.3. Algorithm Design

In this subsection, we introduce the design of our remapping strategies f_K and f_V : token compression, token selection, and sliding window.

3.3.1. Token Compression

By aggregating sequential blocks of keys or values into block-level representations, we obtain compressed keys and values that capture the information of the entire block. Formally, the

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。为了识别注意力计算中最重要块，我们需要为每个块分配重要性分数。下面我们将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会产生中间注意力分数，我们可以利用这些分数来推导选择块的重要性分数，公式如下：

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

其中 $\mathbf{p}_t^{\text{cmp}} \in \mathbb{R}^{\lfloor \frac{l'}{d} \rfloor}$ 是 \mathbf{q}_t 和压缩键 \tilde{K}_t^{cmp} 之间的注意力分数。令 l' 表示选择块的大小。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 $\mathbf{p}_t^{\text{slc}} = \mathbf{p}_t^{\text{cmp}}$ 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。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运算符。对于采用GQA或MQA的模型，其中键值缓存跨查询头共享，必须确保这些头之间的块选择一致，以最小化解码期间的KV缓存加载。组内头之间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 (h) 表示头索引， H 是每组中的查询头数量。这种聚合确保了在同一组内的头之间选择一致的块。

Top- n 块选择。 在获得选择块的重要性分数后，我们保留按块重要性分数排名的前 n 个稀疏块中的标记，公式为：

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

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

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 是选定块的索引集， Cat 表示连接操作。 $\tilde{K}_t^{\text{slc}} \in \mathbb{R}^{d_k \times nl'}$ 是由压缩键组成的张量。类似的公式适用于细粒度值 \tilde{V}_t^{slc} 。选定的键和值然后与 \mathbf{q}_t 一起参与注意力计算，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，并可能主导学习过程，从而阻碍模型从压缩和选择标记中有效学习。为了解决这个问题，我们引入了一个专门的滑动窗口分支，显式地处理局部上下文，使其他分支（压缩和选择）能够专注于学习各自的特征，而不被局部模式所捷径。

compressed key representation is defined as:

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

where l is the block length, d is the sliding stride between adjacent blocks, and φ is a learnable MLP with intra-block position encoding to map keys in a block to a single compressed key. $\tilde{K}_t^{\text{cmp}} \in \mathbb{R}^{d_k \times \lfloor \frac{t-l}{d} \rfloor}$ is tensor composed by compression keys. Usually, we adopt $d < l$ to mitigate information fragmentation. An analogous formulation holds for the compressed value representation \tilde{V}_t^{cmp} . Compressed representations capture coarser-grained higher-level semantic information and reduce computational burden of attention.

3.3.2. Token Selection

Using only compressed keys, values might lose important fine-grained information, motivating us to selectively preserve individual keys, values. Below we describe our efficient token selection mechanism that identifies and preserves the most relevant tokens with low computational overhead.

Blockwise Selection. Our selection strategy processes key and value sequences in spacial continuous blocks, motivated by two key factors: hardware efficiency considerations and inherent distribution patterns of attention scores. *Blockwise selection is crucial to achieve efficient computation on modern GPUs.* That is because modern GPU architectures exhibit significantly higher throughput for continuous block accesses compared to random index-based reads. Also, blockwise computation enables optimal utilization of Tensor Cores. This architectural characteristic has established blockwise memory access and computation as a fundamental principle in high-performance attention implementations, as exemplified by FlashAttention’s block-based design. *Blockwise selection follows the inherent distribution patterns of attention scores.* Prior works (Jiang et al., 2024) have shown that attention scores often exhibit spatial continuity, suggesting that neighboring keys tend to share similar importance levels. Our visualization in Section 6.2 also shows this spatial continuous pattern.

To implement blockwise selection, we first divide key, value sequences into selection blocks. To identify the most important blocks for attention computation, we need to assign importance scores to each block. Below we present our method for computing these block-level importance scores.

Importance Score Computation. Computing block importance scores could introduce significant overhead. Fortunately, the attention computation of compression tokens produces intermediate attention scores that we can leverage to induce selection block importance scores, formulated as:

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

具体来说，我们在窗口 w 中维护最近的标记 $\tilde{K}_t^{\text{win}} = \mathbf{k}_{t-w:t}$, $\tilde{V}_t^{\text{win}} = \mathbf{v}_{t-w:t}$ ，并将不同信息源（压缩标记、选择标记、滑动窗口）的注意力计算隔离到不同的分支中。这些分支的输出然后通过一个学习的门控机制进行聚合。为了进一步防止跨注意力分支的捷径学习，并且具有较小的计算开销，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距离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（ $\tilde{K}_t^{\text{cmp}}, \tilde{V}_t^{\text{cmp}}$; $\tilde{K}_t^{\text{slc}}, \tilde{V}_t^{\text{slc}}$; 和 $\tilde{K}_t^{\text{win}}, \tilde{V}_t^{\text{win}}$ ）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. Kernel Design

为了在训练和预填充过程中实现与FlashAttention相当的加速，我们在Triton上实现了硬件对齐的稀疏注意力内核。鉴于MHA（多头注意力）在解码过程中占用大量内存且效率低下，我们专注于使用共享KV缓存的架构，如GQA和MQA，这些架构符合当前最先进的LLMs。虽然压缩和滑动窗口注意力计算可以与现有的FlashAttention-2内核兼容，但我们引入了专门设计的稀疏选择注意力内核。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，这将导致内存访问效率低下，因为块内的查询可能需要不同的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。如Figure 3所示，这是我们的前向传递实现。所提出的内核架构具有以下关键特性：

1. 以组为中心的数据加载。对于每个内部循环，加载组中位置 t 处所有头的查询 $Q \in \mathbb{R}^{[h, d_k]}$ 及其共享的稀疏键/值块索引 \mathcal{I}_t 。
2. 共享KV获取。在内循环中，按顺序将由 \mathcal{I}_t 索引的连续键/值块加载到SRAM中，表示为 $K \in \mathbb{R}^{[B_k, d_k]}, V \in \mathbb{R}^{[B_k, d_v]}$ ，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（与所选块数 n 成正比）对于不同的查询块几乎相同，我们将查询/输出循环放在 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，和 (2) 在 GPU 流式多处理器之间平衡计算工作负载，实现了接近最优的算术强度。

4. Experiments

我们通过三个角度评估 NSA：(1) 通用基准性能，(2) 长上下文基准性能，以及 (3) 链式思维推理性能，并与全注意力基线和最先进的稀疏注意力方法进行比较。我们将在 Section 5 中推迟对我们的稀疏计算范式的效率分析，在那里我们提供了关于训练和推理速度的详细讨论。

where $\mathbf{p}_t^{\text{cmp}} \in \mathbb{R}^{\lfloor \frac{l'}{d} \rfloor}$ is the attention scores between q_t and compression keys \tilde{K}_t^{cmp} . Let l' denote the selection block size. When compression blocks and selection blocks share the same blocking scheme, i.e., $l' = l = d$, we can directly obtain the selection block importance scores $\mathbf{p}_t^{\text{slc}}$ by $\mathbf{p}_t^{\text{slc}} = \mathbf{p}_t^{\text{cmp}}$ straightforwardly. For cases where the blocking schemes differ, we derive the importance scores for selection blocks according to their spatial relationship. Given $d \mid l$ and $d \mid l'$, we have:

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

where $[\cdot]$ denotes the indexing operator for accessing vector element. For models employing GQA or MQA where key-value caches are shared across query heads, consistent block selection across these heads has to be ensured to minimize KV cache loading during decoding. The shared importance scores across heads in a group are formally defined as:

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

where (h) in the superscript denotes the head index, and H is the number of query heads in each group. This aggregation ensures consistent block selection across heads within the same group.

Top- n Block Selection. After obtaining the selection block importance scores, We retain tokens within the top- n sparse blocks ranked by block importance scores, formulated as:

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

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

where $\text{rank}(\cdot)$ denotes the ranking position in descending order, with $\text{rank} = 1$ corresponding to the highest score, \mathcal{I}_t is the set of selected blocks' indices, Cat denotes the concatenation operation. $\tilde{K}_t^{\text{slc}} \in \mathbb{R}^{d_k \times nl'}$ is tensor composed by compression keys. An analogous formulation applies to the fine-grained value \tilde{V}_t^{slc} . The selected keys and values then participate in the attention computation with \mathbf{q}_t as defined in Equation (5).

3.3.3. Sliding Window

In attention mechanisms, local patterns typically adapt faster and can dominate the learning process, potentially preventing the model from effectively learning from compression and selection tokens. To address this issue, we introduce a dedicated sliding window branch that explicitly handles local context, allowing other branches (compression and selection) to focus on learning their respective features without being shortcut by local patterns. Specifically, we maintain recent tokens $\tilde{K}_t^{\text{win}} = \mathbf{k}_{t-w:t}$, $\tilde{V}_t^{\text{win}} = \mathbf{v}_{t-w:t}$ in a window w , and isolate attention

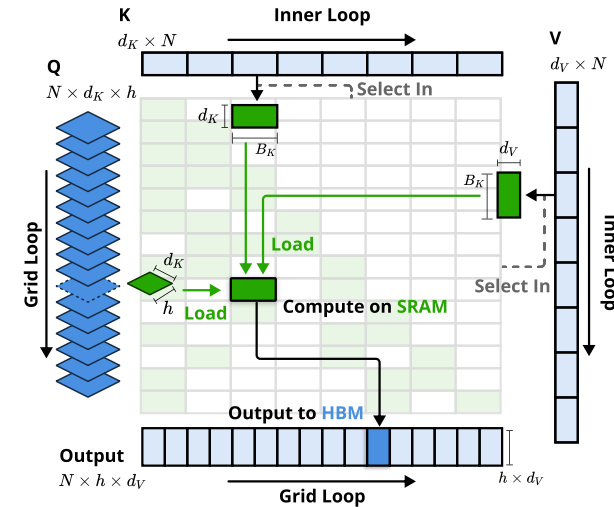


Figure 3 | NSA 的内核设计。内核通过 GQA 组（网格循环）加载查询，获取相应的稀疏 KV 块（内循环），并在 SRAM 上执行注意力计算。绿色块表示 SRAM 上的数据，而蓝色表示 HBM 上的数据。

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 | 预训练性能比较，全注意力基线和NSA在通用基准上的表现，涵盖知识（MMLU, MMLU-PRO, CMMLU）、推理（BBH, GSM8K, MATH, DROP）和编程（MBPP, HumanEval）任务。尽管稀疏性很高，NSA在大多数基准上仍实现了 superior 平均性能。

4.1. Pretraining Setup

遵循最先进的大型语言模型（LLM）中的常见做法，我们的实验采用了一种结合了分组查询注意力（GQA）和专家混合（MoE）的骨干架构，总参数量为27B，其中3B为活动参数。该模型由30层组成，隐藏维度为2560。对于GQA，我们将组数设置为4，总共有64个注意力头。每个头的查询、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，我们采用了DeepSeekMoE (Dai et al., 2024; DeepSeek-AI, 2024) 结构，包含72个路由专家和2个共享专家，并将顶级专家设置为6个。为了确保训练的稳定性，第一层的MoE被SwiGLU形式的MLP所替代。所提出的架构在计算成本和模型性能之间实现了有效的平衡。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1个初始块和2个局部块），滑动窗口大小 $w = 512$ 。全注意力模型和稀疏注意力模型都在270B个8k长度的文本令牌上进行了预训练，随后在32k长度的文本上使用YaRN (Peng et al., 2024) 进行持续训练和监督微调，以实现长上下文适应。两个模型都训练到完全收敛，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趋势，NSA始终优于全注意力模型。

computations of different information sources (compression tokens, and selected tokens, sliding window) into separate branches. These branch outputs are then aggregated through a learned gating mechanism. To further prevent shortcut learning across attention branches with marginal computational overhead, we provide independent keys and values for three branches. This architectural design enables stable learning by preventing gradient interference between local and long-range pattern recognition, while introducing minimal overhead.

After obtaining all three categories of keys and values ($\tilde{K}_t^{\text{cmp}}, \tilde{V}_t^{\text{cmp}}; \tilde{K}_t^{\text{slc}}, \tilde{V}_t^{\text{slc}}; \text{ and } \tilde{K}_t^{\text{win}}, \tilde{V}_t^{\text{win}}$), we compute the final attention output following Equation (5). Together with the compression, selection, and sliding window mechanisms described above, this forms the complete algorithmic framework of NSA.

3.4. Kernel Design

To achieve FlashAttention-level speedup during the training and prefilling, we implement hardware-aligned sparse attention kernels upon Triton. Given MHA is memory-intensive and inefficient for decoding, we focus on architectures with shared KV caches like GQA and MQA following the current state-of-the-art LLMs. While compression and sliding window attention computations are readily compatible with existing FlashAttention-2 kernels, we introduce the specialized kernel design for sparse selection attention. If we were to follow FlashAttention’s strategy of loading temporally continuous query blocks into SRAM, it would result in inefficient memory access since queries within a block may require disjoint KV blocks. To address this, our key optimization lies in a different query grouping strategy: for each position on the query sequence, we load all query heads within a GQA group (they share the same sparse KV blocks) into SRAM. Figure 3 illustrates our forward pass implementation. The proposed kernel architecture is characterized by the following key features:

1. **Group-Centric Data Loading.** For each inner loop, load all heads’ queries $Q \in \mathbb{R}^{[h, d_k]}$ in the group at position t and their shared sparse key/value block indices \mathcal{I}_t .
2. **Shared KV Fetching.** In the inner loop, Sequentially load continuous key/value blocks indexed by \mathcal{I}_t into SRAM as $K \in \mathbb{R}^{[B_k, d_k]}, V \in \mathbb{R}^{[B_k, d_v]}$ to minimize memory loading, where B_k is the kernel block size satisfying $B_k | l'$.
3. **Outer Loop on Grid.** Since the inner-loop length (proportional to the selected block count n) remains nearly identical for different query blocks, we put query/output loops in Triton’s grid scheduler to simplify and optimize the kernel.

This design achieves near-optimal arithmetic intensity by (1) eliminating redundant KV transfers through group-wise sharing, and (2) balancing compute workloads across GPU streaming multiprocessors.

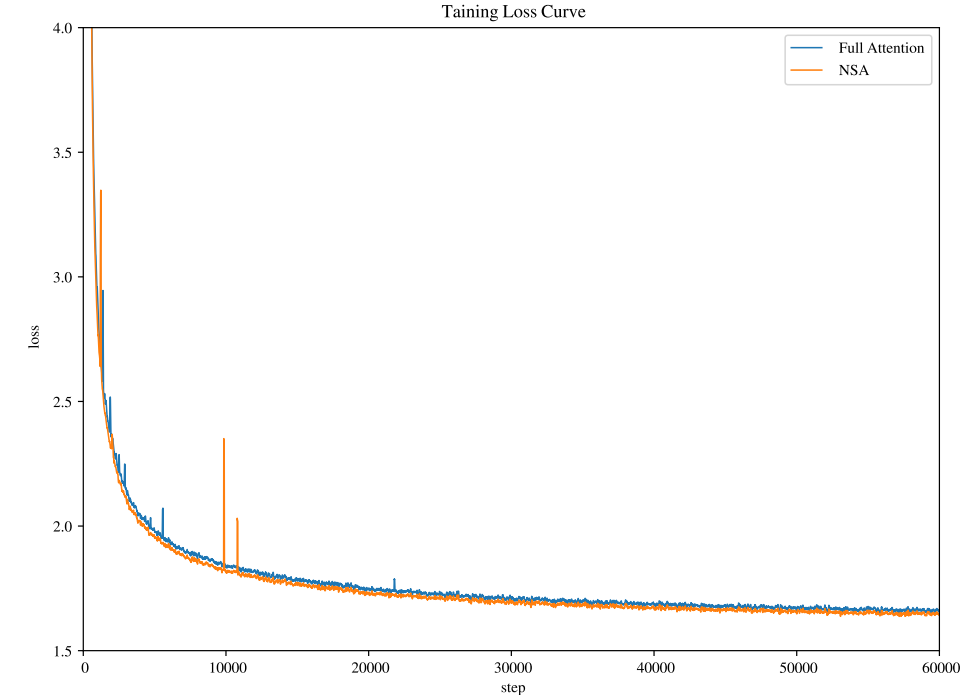


Figure 4 | 预训练损失比较：27B参数模型的全注意力机制和我们的NSA。两种模型都表现出稳定的收敛性，NSA达到了更低的损失值。

4.2. Baselines Methods

除了与全注意力机制进行比较外，我们还评估了几种最先进的推理阶段稀疏注意力方法：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 个得分最高的键，再在这些位置上计算注意力。这些方法涵盖了多种稀疏注意力范式，包括KV缓存驱逐、查询感知选择和精确的前 n 个稀疏选择。

在一般评估中，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，这些方法实际上等同于全注意力机制。因此，在这种设置下，我们仅展示NSA与全注意力基线的比较结果。在长上下文评估中，我们在所有基线方法之间进行比较，并将所有稀疏注意力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链式思维推理评估，我们将比较限制在全注意力机制上，因为稀疏注意力基线不支持训练。

4.3. Performance Comparison

总体评估。我们评估了预训练的 NSA 和全注意力基线，在涵盖知识、推理和编码能力的全面基准测试套件上进行了评估，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)。结果如 Table 1 所示。尽管其稀疏性，NSA 仍实现了整体性能的优越表现，

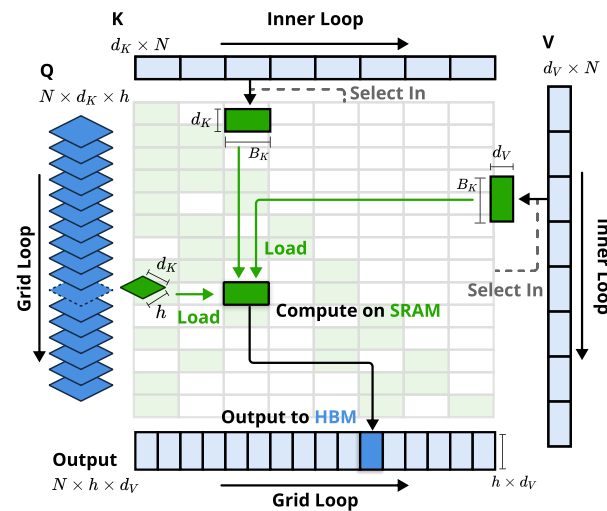


Figure 3 | Kernel design for NSA. The kernel loads queries by GQA groups (Grid Loop), fetches corresponding sparse KV blocks (Inner Loop), and performs attention computation on SRAM. Green blocks indicate data on SRAM, while blue indicates data on HBM.

4. Experiments

We evaluate NSA through three lenses: (1) general benchmarks performance, (2) long-context benchmarks performance, and (3) chain-of-thought reasoning performance, comparing against Full Attention baseline and state-of-the-art sparse attention methods. We defer the efficiency analysis of our sparse computation paradigm to Section 5, where we provide detailed discussions on training and inference speed.

4.1. Pretraining Setup

Following the common practice in state-of-the-art LLMs, our experiments adopt a backbone combining Grouped-Query Attention (GQA) and Mixture-of-Experts (MoE), featuring 27B total parameters with 3B active parameters. The model consists of 30 layers with a hidden dimension of 2560. For GQA, we set the number of groups to 4, with a total of 64 attention heads. For each head, the hidden dimensions of the query, key, and value are configured as $d_q = d_k = 192$ and $d_v = 128$, respectively. For MoE, we utilize the DeepSeekMoE (Dai et al., 2024; DeepSeek-AI, 2024) structure, with 72 routed experts and 2 shared experts, and set the top-k experts to 6. To ensure training stability, the MoE in the first layer is replaced by an MLP in the form of SwiGLU. The proposed architecture achieves an effective trade-off between computation cost and model performance. For NSA, we set compression block size $l = 32$, sliding stride $d = 16$, selected block size $l' = 64$, selected block count $n = 16$ (including fixed activating the 1 initial block and 2 local blocks), and sliding window size $w = 512$. Both Full Attention and sparse attention models are pretrained on 270B tokens of 8k-length texts, followed by continued training and supervised fine-tuning on 32k-length texts with YaRN (Peng et al., 2024) to achieve long-context

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	<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 | 在 LongBench 上，包括单文档 QA、多文档 QA、合成和代码任务类别的子集，我们的 NSA 与基线方法的性能比较。我们的 NSA 在大多数基线方法中表现最佳，包括 Full Attention。

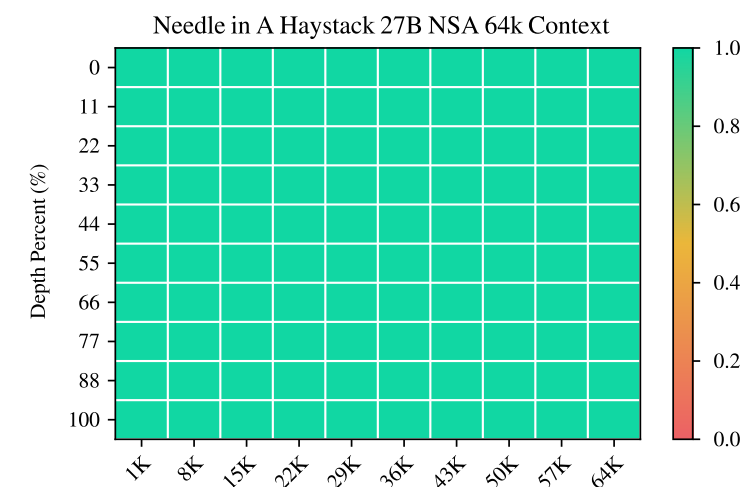


Figure 5 | 在64k上下文长度下，针尖在 haystack 中的检索准确性。NSA 通过其层次稀疏注意力设计实现了完美的准确性。

在9个指标中的7个指标上超过了所有基线，包括全注意力基线。这表明，虽然 NSA 在较短的序列上可能无法完全发挥其效率优势，但它表现出强大的性能。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出专门的注意力机制。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，通过过滤掉无关的注意力路径中的噪声，可能增强性能。在多样化的评估中的一致表现也验证了 NSA 作为通用架构的鲁棒性。

长上下文评估。如 Figure 5 所示，NSA 在64k上下文的“大海捞针” (Kamradt, 2023) 测试中实现了所有位置的完美检索准确率。这种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高效地扫描全局上下文，以及选择令牌以精确地检索局部信息。粗粒度的压缩以较低的计算成本识别相关的上下文块，而选定令牌的令牌级注意力确保了关键细粒度信息的保留。这种设计使 NSA 能够同时保持全局意识和局部精确度。

我们还在 LongBench (Bai et al., 2023) 上评估了 NSA，并与最先进的稀疏注意力方法和全注意力基线进行了对比。为了确保稀疏性的一致性，我们将所有稀疏注意力基线中每个查询

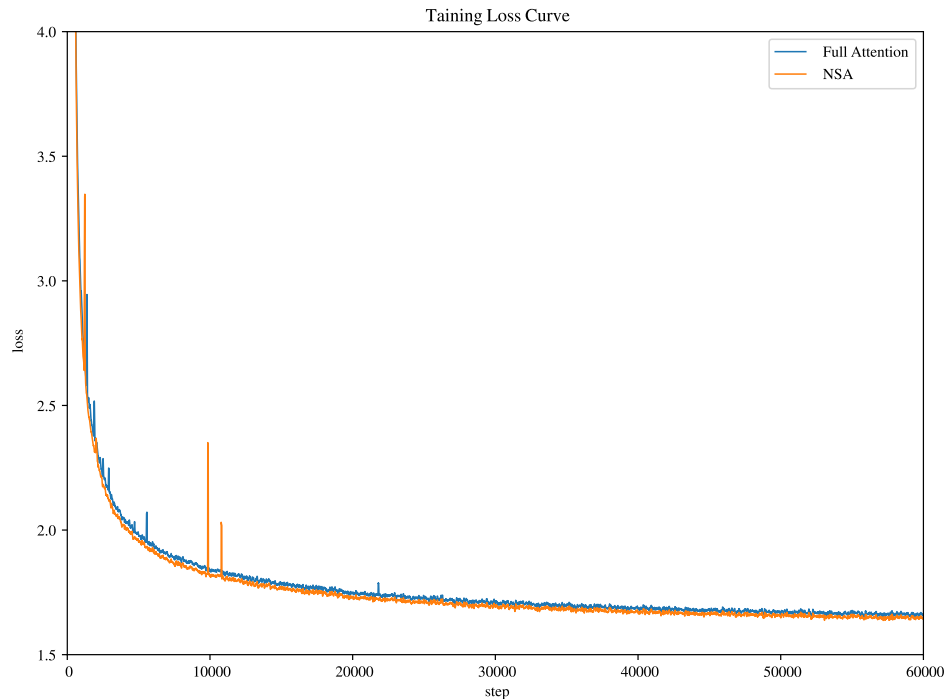


Figure 4 | Pretraining loss comparison between Full Attention and our NSA on 27B-parameter model. Both models exhibit stable convergence, with NSA achieving lower loss values.

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 10-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 | Pretraining performance comparison between the full attention baseline and NSA on general benchmarks, across knowledge (MMLU, MMLU-PRO, CMMLU), reasoning (BBH, GSM8K, MATH, DROP), and coding (MBPP, HumanEval) tasks. NSA achieves superior average performance on most benchmarks despite high sparsity.

adaptation. Both models are trained to full convergence to ensure fair comparison. As shown in Figure 4, the pretraining loss curve of our NSA and Full Attention baseline demonstrates stable and smooth decline, with NSA consistently outperforming the Full Attention model.

4.2. Baselines Methods

In addition to comparing with Full Attention, we evaluate several state-of-the-art inference-stage sparse attention methods: H2O (Zhang et al., 2023b), infLLM (Xiao et al., 2024), Quest (Tang et al., 2024), and Exact-Top, which first computes full attention score and select the top- n scores keys corresponding to each query and then calculates attention on these positions. These methods span diverse sparse attention paradigms, including KV-cache eviction, query-aware selection, and exact top- n sparse selection.

For general evaluation, where most samples have lengths within the local context window of

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

Table 3 | 基于AIME指令的评估在监督微调后进行。我们的NSA-R在8k和16k序列长度下均表现出优于Full Attention-R的性能。

激活的令牌数设置为2560个令牌，这对应于NSA在处理32k序列长度时激活的平均令牌数。根据StreamLLM (Xiao et al., 2023)，这个令牌预算包括前128个令牌和512个本地令牌。我们从LongBench中排除了某些子集，因为这些子集在所有模型中的得分较低，可能无法提供有意义的比较。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（比全注意力高+0.032，比Exact-Top高+0.046）。这一改进源于两个关键创新：(1) 我们的原生稀疏注意力设计，使得在预训练期间可以端到端优化稀疏模式，促进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中表现出色，在多跳问答任务（HPQ和2Wiki）中比全注意力高+0.087和+0.051，在代码理解（LCC: +0.069）中超过基线，在段落检索（PassR-en: +0.075）中优于其他方法。这些结果验证了NSA处理多样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与先进的下游训练范式的兼容性，我们调查了其通过后训练获取链式思维数学推理能力的能力。鉴于强化学习在较小规模模型上的效果有限，我们采用从DeepSeek-R1的知识蒸馏，使用32k长度的10B个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数学邀请赛（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- p 值为每个问题生成16个回答，并获得平均得分。为了验证推理深度的影响，我们进行了两个生成上下文限制的实验：8k和16k令牌，测量扩展的推理链是否提高准确性。模型预测的示例比较见Appendix A。

如表Table 3所示，NSA-R在8k上下文设置下的准确率显著高于全注意力-R（+0.075），这一优势在16k上下文设置下仍然存在（+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高效捕获对复杂数学推导至关重要的长程逻辑依赖；(2) 我们的架构通过硬件对齐设计保持了足够的上下文密度，以支持推理深度的增加而不会发生灾难性遗忘。在不同上下文长度下的一致优越性能证实了当原生集成到训练管道中时，稀疏注意力在高级推理任务中的可行性。

5. Efficiency Analysis

我们在8-GPU A100系统上评估了NSA相对于全注意力机制的计算效率。在效率分析中，我们还配置了模型的GQA组数 $g = 4$ ，每组头数 $h = 16$ ，查询/键维度 $d_k = 192$ ，以及值维度 $d_v = 128$ 。按照Section 4中的相同设置，我们将NSA压缩块大小 $l = 32$ ，滑动步长 $d = 16$ ，选择块大小 $l' = 64$ ，

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	<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 | Performance comparison between our NSA and baselines on LongBench, including subsets in single document QA, multi-document QA, synthetic and code task categories. NSA outperformed most of the baselines including Full Attention.

sparse attention baselines, these methods are effectively equivalent to Full Attention. Therefore, we present only the comparison results between NSA and Full Attention baseline in this setting. In the long-context evaluation, we conduct comparisons across all baseline methods, with the sparsity of all sparse attention methods set to the same to ensure a fair comparison. For chain-of-thought reasoning evaluation, which requires long-text supervised fine-tuning, we limit our comparison to Full Attention, as sparse attention baselines do not support training.

4.3. Performance Comparison

General Evaluation. We evaluated the pretrained NSA and Full Attention baseline, on a comprehensive suite of benchmarks spanning knowledge, reasoning, and coding capabilities, including 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), and HumanEval (Chen et al., 2021). The results are shown in Table 1. Despite its sparsity, NSA achieves superior overall performance, outperforming all baselines including Full Attention on 7 out of 9 metrics. This indicates that although NSA may not fully leverage its efficiency advantages on shorter sequences, it shows strong performance. Notably, NSA demonstrates significant gains in reasoning-related benchmarks (DROP: +0.042, GSM8K: +0.034), suggesting that our pretraining helps models to develop specialized attention mechanisms. This sparse attention pretraining mechanism forces model to focus on the most important information, potentially enhancing performance by filtering out noise from irrelevant attention pathways. The consistent performance across diverse evaluations also validates NSA’s robustness as a general-purpose architecture.

Long-Context Evaluation. As shown in Figure 5, NSA achieves perfect retrieval accuracy across all positions in 64k-context needle-in-a-haystack (Kamradt, 2023) test. This performance stems from our hierarchical sparse attention design, which combines compression tokens for efficient global context scanning, and selection tokens for precise local information retrieval. The coarse-grained compression identifies relevant context blocks at low computational cost, while

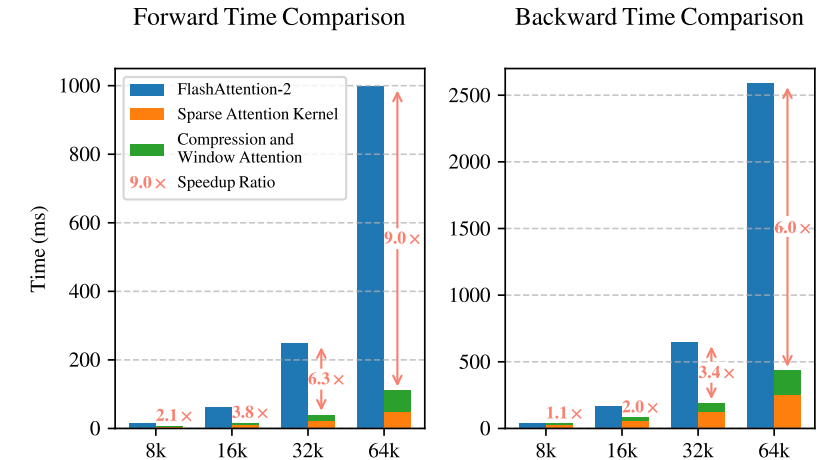


Figure 6 | 基于 Triton 的 NSA 内核与基于 Triton 的 FlashAttention-2 内核的比较。我们的实现显著减少了所有上下文长度的延迟，随着输入长度的增加，改进变得更加明显。

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 | 解码期间每次注意力操作的内存访问量（以等效的令牌数量表示）。由于解码的算术强度较低且受内存限制，预期的加速比大约与内存访问量成线性关系。

选择块数量 $n = 16$ ，以及滑动窗口大小 $w = 512$ 。

5.1. Training Speed

我们比较了基于Triton实现的我们的NSA注意力机制和全注意力机制与基于Triton的FlashAttention-2，以确保在相同后端上的速度比较公平。如Figure 6所示，随着上下文长度的增加，我们的NSA实现了逐渐增大的加速，最多在64k上下文长度时达到9.0×前向和6.0×反向加速。值得注意的是，随着序列变长，速度优势变得更加明显。这种加速源自我们针对硬件优化的算法设计，以最大化稀疏注意力架构的效率：(1) 块级内存访问模式通过合并加载最大化了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

Attention 的解码速度主要由内存访问瓶颈决定，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 只需要加载最多 $\lfloor \frac{s-l}{d} \rfloor$ 个压缩 token， nl' 个选定 token，以及 w 个邻近 token，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，我们的方法表现出显著的延迟减少，在 64k 上下文长度时，速度提高了高达 11.6×。这种内存访问效率的优势在更长的序列中也更加明显。

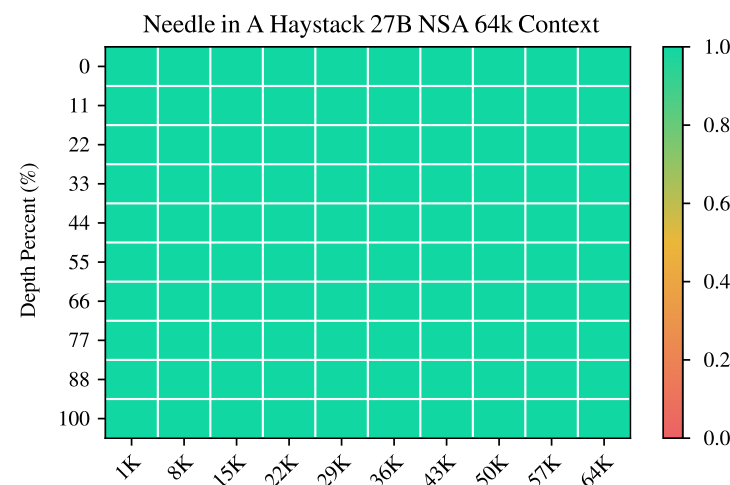


Figure 5 | Needle-in-a-Haystack retrieval accuracy across context positions with 64k context length. NSA achieves perfect accuracy through its hierarchical sparse attention design.

the token-level attention on selected tokens ensures the preservation of critical fine-grained information. This design enables NSA to maintain both global awareness and local precision.

We also evaluate NSA on LongBench (Bai et al., 2023) against state-of-the-art sparse attention methods and Full Attention baseline. To ensure consistent sparsity, we set the token activated by each query in all sparse attention baselines to 2560 tokens, which corresponds to the average number of tokens activated in NSA when handling 32k sequence lengths. Following StreamLLM (Xiao et al., 2023), this token budget includes the leading 128 tokens and 512 local tokens. We exclude certain subsets from LongBench due to their low scores across all models, which may not provide meaningful comparisons. As shown in Table 2, NSA achieves the highest average score 0.469, outperforming all baselines (+0.032 over Full Attention and +0.046 over Exact-Top). This improvement arises from two key innovations: (1) our native sparse attention design, which enables end-to-end optimization of sparse patterns during pretraining, facilitates synchronized adaptation between the sparse attention module and other model components; and (2) the hierarchical sparse attention mechanism achieves a balance between local and global information processing.

Notably, NSA demonstrates exceptional performance on tasks requiring complex reasoning over long contexts, achieving +0.087 and +0.051 improvements over Full Attention on multi-hop QA tasks (HPQ and 2Wiki), exceeding the performance of baselines on code understanding (LCC: +0.069), and outperforming other methods on passage retrieval (PassR-en: +0.075). These results validate NSA’s capability to handle diverse long-context challenges, with its natively pretrained sparse attention providing additional benefits in learning task-optimal patterns.

Chain-of-Thought Reasoning Evaluation. To evaluate NSA’s compatibility with advanced downstream training paradigms, we investigate its capacity to acquire chain-of-thought mathe-

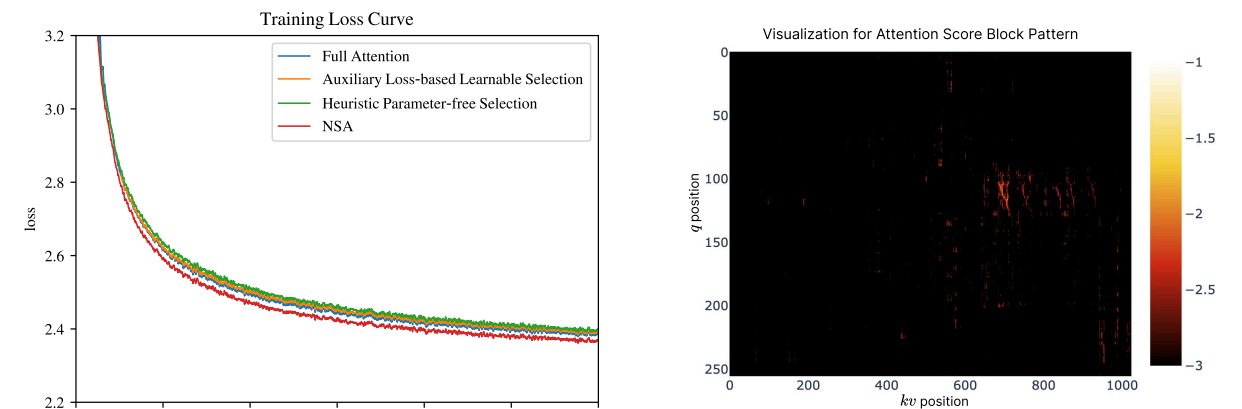


Figure 7 | 比较在3B参数模型上使用全注意力机制和不同令牌选择策略的训练损失。我们的NSA实现了更好的性能。

Figure 8 | 在完整注意力机制的Transformer上可视化注意力图。浅色区域表示较高的注意力值。如图所示，注意力得分呈现出块状聚类分布。

6. Discussion

在本节中，我们回顾了 NSA 的开发过程，并讨论了从探索不同的稀疏注意力策略中获得的关键见解。虽然我们的方法展示了有希望的结果，但了解在替代策略中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。我们首先考察了促使我们做出设计选择的替代标记选择策略的挑战，然后通过可视化提供对注意力分布模式的见解。

6.1. Challenges with Alternative Token Selection Strategies

在设计 NSA 之前，我们探索了将现有的稀疏注意力方法适应到训练阶段。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：

基于键聚类的策略。我们考察了基于聚类的策略，如 ClusterKV (Liu et al., 2024)。这些方法将来自同一聚类的键 (Keys) 和值 (Values) 存储在连续的内存区域中。虽然理论上在训练和推理中是可行的，但它们面临三个主要挑战：(1) 动态聚类机制引入的非 trivial 计算开销；(2) 由于聚类间的不平衡，特别是在混合专家 (Mixture-of-Experts, MoE) 系统中，倾斜的专家并行 (Expert Parallelism, EP) 组执行时间导致持续的负载不平衡，使得操作优化变得更加困难；(3) 实现约束来自于必须定期重新聚类 and 分块顺序训练协议的需求。这些因素的结合造成了显著的瓶颈，极大地限制了它们在实际部署中的有效性。

其他块选择策略。我们还考虑了与 NSA 不同的块选择策略，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一个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，这增加了操作开销并经常降低模型性能；(2) 启发式无参数的重要性分数计算策略召回率低，导致性能次优。我们在具有类似架构的 3B 参数模型上评估了这两种方法，并将它们的损失曲线与 NSA 和全注意力机制进行比较。对于基于辅助

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

Table 3 | AIME Instruction-based Evaluating after supervised fine-tuning. Our NSA-R demonstrates better performance than Full Attention-R at both 8k and 16k sequence lengths

mathematical reasoning abilities via post-training. Given the limited effectiveness of reinforcement learning on smaller-scale models, we employ knowledge distillation from DeepSeek-R1, conducting supervised fine-tuning (SFT) with 10B tokens of 32k-length mathematical reasoning traces. This produces two comparable models: Full Attention-R (Full Attention baseline) and NSA-R (our sparse variant). We assess both models on the challenging American Invitational Mathematics Examination (AIME 24) benchmark. We use a sampling temperature of 0.7 and a top- p value of 0.95 to generate 16 responses for each question and obtain the average score. To validate the impact of reasoning depth, we conduct experiments with two generation context limits: 8k and 16k tokens, measuring whether extended reasoning chains improve accuracy. Example comparisons of model predictions are provided in Appendix A.

As shown in Table 3, NSA-R achieves significantly higher accuracy than Full Attention-R under the 8k context setting (+0.075), with this advantage persisting at 16k contexts (+0.054). These results validate two key benefits of native sparse attention: (1) The pretrained sparse attention patterns enable efficient capture of long-range logical dependencies critical for complex mathematical derivations; (2) Our architecture’s hardware-aligned design maintains sufficient context density to support growing reasoning depth without catastrophic forgetting. The consistent outperformance across context lengths confirms sparse attention’s viability for advanced reasoning tasks when natively integrated into the training pipeline.

5. Efficiency Analysis

We evaluate the computational efficiency of NSA against Full Attention on an 8-GPU A100 system. In efficiency analysis, we also configure the model with GQA group $g = 4$, heads per group $h = 16$, query/key dimension $d_k = 192$, and value dimension $d_v = 128$. Following the same settings in Section 4, we set NSA compression block size $l = 32$, sliding stride $d = 16$, selected block size $l' = 64$, selected block count $n = 16$, and sliding window size $w = 512$.

5.1. Training Speed

We compare the Triton-based implementations of our NSA attention and Full Attention with Triton-based FlashAttention-2 to ensure fair speed comparison across the same backend. As shown in Figure 6, our NSA achieves progressively greater speedups as context length increases,

损失的选择方法，我们为每个块引入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注意力分数监督。对于启发式无参数的选择方法，我们遵循 Quest 的策略，实现直接选择，使用查询和键块的坐标最小最大值的乘积，而不引入额外参数。我们还探索了一种冷启动训练方法，在前 1000 步使用全注意力机制，然后过渡到启发式块选择。如 Figure 7 所示，这两种方法的损失表现较差。

6.2. Visualization

为了探索变压器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，其中注意力分数倾向于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。这一观察启发了我们设计NSA，表明基于空间连续性选择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。这一观察激励我们探索一种在连续标记块上操作的稀疏注意力机制，而不是在单个标记上操作，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类：(1) 固定稀疏模式，(2) 动态令牌剪枝，(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

SlidingWindow 是一种常用的方法，允许查询仅在固定窗口内计算注意力。StreamingLLM (Xiao et al., 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：注意力池（早期令牌）和本地上下文窗口。虽然这些方法有效地减少了内存和计算成本，但它们忽略上下文的刚性模式限制了在需要完整上下文理解的任务上的性能。

7.2. Dynamic Token Pruning

H2O (Zhang et al., 2023b) 实现了一种自适应方法，以减少解码过程中KV缓存的内存使用。该方法根据注意力分数动态地驱逐被认为对未来预测不太重要的令牌。SnapKV (Li et al., 2024) 也引入了一种令牌修剪策略，通过选择性地保留最关键的功能来减少KV缓存，从而实现高效的内存使用。SnapKV 通过在预填充期间进行注意力权重分析和投票来识别重要特征，然后通过将选定的压缩特征与最近的上下文结合起来更新KV缓存，以保持提示的一致性。

7.3. Query-Aware Selection

Quest (Tang et al., 2024) 采用了一种分块选择策略，其中每个块的重要性是通过查询和键块的坐标-wise 最小-最大值之间的乘积来估计的。结果得分有助于选择 top- n 重要的键-值块进行注

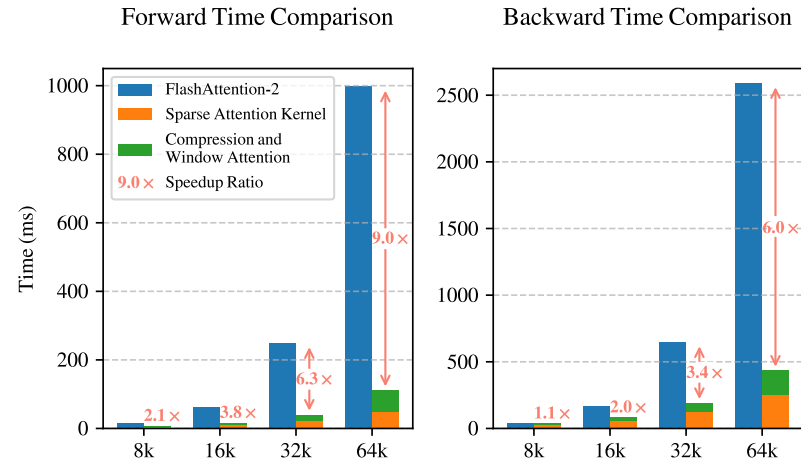


Figure 6 | Comparison of Triton-based NSA kernel with Triton-based FlashAttention-2 kernel. Our implementation significantly reduces latency across all context lengths, with the improvement becoming more pronounced as input length increases.

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 | Memory access volume (in equivalent number of tokens) per attention operation during decoding. Due to the low arithmetic intensity and memory-bound nature of decoding, the expected speedup is approximately linear with the volume of memory access.

up to 9.0× forward and 6.0× backward speedup at 64k context-length. Notably, the speed advantage becomes more pronounced with longer sequences. This speedup stems from our hardware-aligned algorithm design to maximize the efficiency of sparse attention architecture: (1) The Blockwise memory access pattern maximizes Tensor Core utilization through coalesced loads, (2) The delicate loop scheduling in the kernel eliminates redundant KV transfers.

5.2. Decoding Speed

The decoding speed of Attention is primarily determined by the memory access bottleneck, which is closely tied to the amount of KV cache loading. In each decoding step, Our NSA just needs to load at most $\lfloor \frac{s-l}{d} \rfloor$ compression tokens, nl' selected tokens, and w neighbor tokens, where s is the cached sequence length. As shown in Table 4, our method exhibits a significant reduction in latency as the decoding length increases, achieving up to 11.6× speedup at 64k context-length. This advantage in memory access efficiency also amplifies with longer sequences.

意力计算。InfLLM (Xiao et al., 2024) 通过维护注意力池、局部上下文和可检索块，将固定模式与检索结合。该方法从每个块中选择代表性键来估计块的重要性。HashAttention (Desai et al., 2024) 通过使用学习的函数将查询和键映射到汉明空间，将关键令牌识别表述为一个推荐问题。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来进行注意力计算，从而实现稀疏性。

8. Conclusion

我们提出了NSA，这是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层次化标记压缩与分块标记选择，我们的架构在保持全注意力性能的同时，实现了加速训练和推理。NSA通过展示通用基准性能与全注意力基线匹配、在长上下文评估中超出建模能力以及增强的推理能力，同时伴随着可测量的计算延迟减少和显著的加速，推进了最先进水平。

References

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

6. Discussion

In this section, we reflect on the development process of NSA and discuss key insights gained from our exploration of different sparse attention strategies. While our approach demonstrates promising results, understanding the challenges encountered with alternative strategies and analyzing attention patterns provides valuable context for future research directions. We first examine challenges with alternative token selection strategies that motivated our design choices, followed by visualizations that offer insights into attention distribution patterns.

6.1. Challenges with Alternative Token Selection Strategies

Before designing NSA, we explored adapting existing sparse attention methods to the training stage. However, these attempts encountered various challenges, prompting us to design a different sparse attention architecture:

Key-Clustering Based Strategies. We examined clustering-based strategies like ClusterKV (Liu et al., 2024). These methods store Keys and Values from the same cluster in contiguous memory regions. While theoretically feasible for training and inference, they face three significant challenges: (1) Non-trivial computational overhead introduced by dynamic clustering mechanisms; (2) Operator optimization difficulties exacerbated by inter-cluster imbalances, especially in Mixture-of-Experts (MoE) systems, where skewed Expert Parallelism (EP) group execution times lead to persistent load imbalances; (3) Implementation constraints arising from the need for mandatory periodic reclustering and chunk-sequential training protocols. These combined factors create substantial bottlenecks, significantly limiting their effectiveness for real-world deployment.

Other Blockwise Selection Strategies. We also considered blockwise key, value selection strategies different from NSA, such as Quest (Tang et al., 2024) and InfLLM (Xiao et al., 2024). These methods rely on computing an importance score for each block and selecting the top- n blocks based on their similarity with q_t . However, existing methods face two critical issues: (1) Since the selection operation is non-differentiable, importance score computation based on neural networks relies on auxiliary loss, which increases operator overhead and often degrades model performance; (2) Heuristic parameter-free importance score computation strategy suffer from low recall rates, leading to suboptimal performance. We evaluate both approaches on a 3B-parameter model with similar architecture and compare their loss curve with NSA and Full Attention. For the auxiliary loss-based selection method, we introduce additional queries and representative keys for each block to estimate the block importance scores. These scores are supervised by the mean attention scores between the original queries and keys within each block. For the heuristic parameter-free selection method, following the strategy of Quest, we implement direct selection using the product between queries and coordinate-wise min-max of

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. 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. Cmmu: 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.

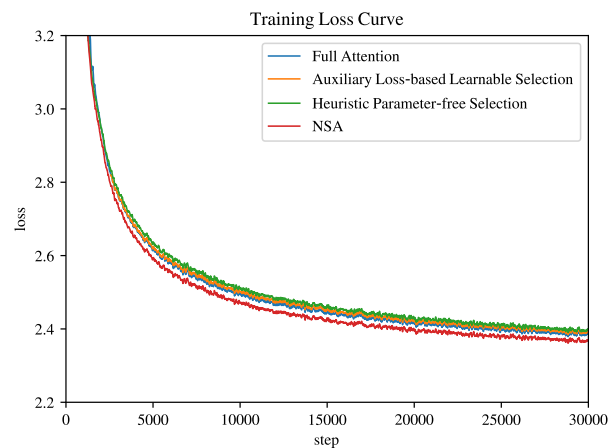


Figure 7 | Compare training loss on a 3B-parameter model with Full Attention and different token selection strategies and. Our NSA achieves better performance.

the key chunks, without introducing additional parameters. We also explore a cold-start training approach where Full Attention is applied for the initial 1000 steps before transitioning to the heuristic blockwise selection. As shown in Figure 7, both methods exhibited inferior loss.

6.2. Visualization

To explore potential patterns in transformer attention distributions and seek inspiration for our design, we visualize the attention map from our pretrained 27B Full Attention model in Figure 8. The visualization reveals interesting patterns where attention scores tend to exhibit blockwise clustering characteristics, with nearby keys often showing similar attention scores. This observation inspired our design of NSA, suggesting that selecting key blocks based on spatial continuity might be a promising approach. The blockwise clustering phenomenon indicates that tokens adjacent in the sequence may share certain semantic relationships with query tokens, though the exact nature of these relationships requires further investigation. This observation motivated us to explore a sparse attention mechanism that operates on continuous token blocks rather than individual tokens, aiming to enhance computational efficiency and preserve high-attention patterns.

7. Related Works

We review existing approaches that improve the efficiency of attention computation through sparse attention. These methods can be broadly categorized into three groups based on their core strategies: (1) fixed sparse pattern, (2) dynamic token pruning, and (3) query-aware selection. We introduce several representative works from each category.

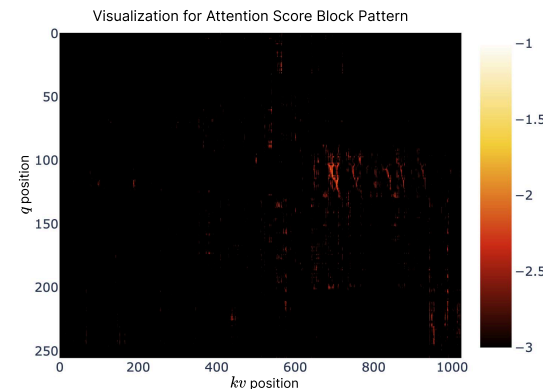


Figure 8 | Visualization of Attention Map on a Full Attention transformer. Light-colored regions indicate higher attention values. As shown in the figure, attention scores exhibit blockwise clustering distribution.

- 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. Infillm: 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.

7.1. Fixed Sparse Pattern

SlidingWindow is a commonly used approach that allows the query to compute attention only within a fixed window. StreamingLLM (Xiao et al., 2023) addresses the challenges of processing long text streams by maintaining two critical portions of the context: an attention sink (early tokens) and a local context window. While these approaches effectively reduce memory and computation costs, their rigid pattern of ignoring contexts limits their performance on tasks requiring full context understanding.

7.2. Dynamic Token Pruning

H2O (Zhang et al., 2023b) implements an adaptive approach to reduce KV-cache memory usage during decoding. This method dynamically evicts tokens deemed less important for future predictions based on their recent utility according to attention score. SnapKV (Li et al., 2024) also introduces a token pruning strategy that reduces the KV cache by selectively retaining only the most crucial features, enabling efficient memory usage. SnapKV identifies important features through attention weight analysis and voting during prefilling, then updates KV cache by combining selected compressed features with recent context to maintain prompt consistency.

7.3. Query-Aware Selection

Quest (Tang et al., 2024) employs a blockwise selection strategy where each chunk’s importance is estimated by product between query and coordinate-wise min-max of the key chunks. The results scores help to select top- n important key-value chunks for attention. InfLLM (Xiao et al., 2024) combines fixed patterns with retrieval by maintaining attention sinks, local context, and retrievable chunks. This method selects representative keys from each chunk to estimate chunk importance. HashAttention (Desai et al., 2024) formulates pivotal token identification as a recommendation problem by mapping queries and keys to Hamming space using learned functions. ClusterKV (Liu et al., 2024) achieves sparsity by firstly clustering keys and then selecting the most relevant clusters for attention computation based on query-cluster similarity.

8. Conclusion

We present NSA, a hardware-aligned sparse attention architecture for efficient long-context modeling. By integrating hierarchical token compression with blockwise token selection within a trainable architecture, our architecture achieves accelerated training and inference while maintaining Full Attention performance. NSA advances the state-of-the-art by demonstrating general benchmark performance matches full-attention baselines, exceeding modeling capability in long-context evaluations, and enhanced reasoning ability, all accompanied by measurable reductions in computational latency and achieving significant speedup.

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.

References

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

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. Long-bench: 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](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](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.

A. Example of AIME Results

Prompt:

“|User| 令 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 $|\log_2(x^4y^3z^2)|$ 的值为 $\frac{m}{n}$, 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 。|助手|”

Our Result:

给定方程组:

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

将每个变量表示为 x , y , 和 z 的函数:

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

将这些表达式互相代入, 我们得到:

- 从 $x = \sqrt{2} \cdot yz$ 和 $y = 2^{1/3} \cdot xz$, 我们得到 $x = \sqrt{2} \cdot (2^{1/3} \cdot xz) \cdot z$, 从而得出 $z = 2^{-5/12}$ 。
- 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 $x^4y^3z^2$:

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

将这些相乘:

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

Taking the logarithm base 2:

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. Cmmmlu: 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.

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

The absolute value is:

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

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

33

Baseline Result:

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：

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

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

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

将这些代入彼此，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：

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

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：

- 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

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

令 $A = \log_2(x)$, $B = \log_2(y)$, $C = \log_2(z)$ 。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:

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

Thus, we have:

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

我们需要找到 $|\log_2(x^4 y^3 z^2)|$:

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

转换为一个共同的分母 (48) :

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

Summing these:

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

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

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.

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:

给定方程:

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

我们首先将对数幂规则应用于两个方程。

对于第一个方程:

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

令 $z = \log_y(x)$ 。那么, $\log_x(y) = \frac{1}{z}$ 。将这个代入方程, 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:

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

A. Example of AIME Results

Prompt:

" | User | Let x, y and z be positive real numbers that satisfy the following system of equations:

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

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

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

Then the value of $|\log_2(x^4 y^3 z^2)|$ is $\frac{m}{n}$ where m and n are relatively prime positive integers. Find $m + n$. Please reason step by step, and put your final answer within \square . | Assistant | "

Our Result:

$\langle \text{think} \rangle \dots \langle / \text{think} \rangle$ (2275 tokens)

Given the system of equations:

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

We convert these logarithmic equations to their exponential forms:

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

Expressing each variable in terms of x, y , and z :

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

Substituting these expressions into each other, we find:

- From $x = \sqrt{2} \cdot yz$ and $y = 2^{1/3} \cdot xz$, we get $x = \sqrt{2} \cdot (2^{1/3} \cdot xz) \cdot z$, leading to $z = 2^{-5/12}$. - Substituting z into the equation for y , we get $y = 2^{-3/8}$. - Substituting z and y into the equation for x , we get $x = 2^{-7/24}$.

Next, we compute $x^4 y^3 z^2$:

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

Multiplying these together:

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

Taking the logarithm base 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 , 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 $\boxed{25}$ 。

Baseline Result:

给定问题, 实数 x 和 y 满足 $x, y > 1$, 并且满足方程 $\log_x(y^x) = 10$ 和 $\log_y(x^4 y) = 10$, 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, 我们用其中一个表示另一个:

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

然后我们计算乘积 xy :

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

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

The absolute value is:

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

Thus, $m = 25$ and $n = 8$, giving $m + n = 33$.

33

Baseline Result:

<think>...</think> (9392 tokens)

Given the system of logarithmic equations:

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

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

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

We convert these equations to exponential form:

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

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

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

Substituting these into each other, we find:

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

By solving these equations step-by-step, we find:

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

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

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

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

最后，我们发现乘积 xy 简化为：

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

However, upon verifying the third equation, we realize the system might not have a solution. Therefore, we use matrix inversion to solve the system:

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

Let $A = \log_2(x)$, $B = \log_2(y)$, $C = \log_2(z)$. The system becomes:

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

We solve this system using Cramer's rule and matrix inversion. The determinant of the matrix is -2, and the solutions are:

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

Thus, we have:

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

We need to find $|\log_2(x^4 y^3 z^2)|$:

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

Converting to a common denominator (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:

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

The fraction $\frac{115}{16}$ is in simplest form, so $m = 115$ and $n = 16$. Therefore, $m + n = 115 + 16 = 131$.

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 tokens)

Given the equations:

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

We start by applying the logarithm power rule to both equations.

For the first equation:

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

Let $z = \log_y(x)$. Then, $\log_x(y) = \frac{1}{z}$. Substituting this into the equation, we get:

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

For the second equation:

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

Using the power rule, we get:

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

Since $\log_y(x) = z$, this simplifies to:

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

We now have two expressions:

$$1. x = 10z$$

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

Substituting $x = 10z$ and $y = \frac{5/2}{z}$ into the product xy , we get:

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

Thus, the value of xy is 25.

Baseline Result:

$\langle \text{think} \rangle \dots \langle / \text{think} \rangle$ (16223 tokens)

Given the problem where real numbers x and y with $x, y > 1$ satisfy the equations $\log_x(y^x) = 10$ and $\log_y(x^4y) = 10$, we need to find the value of xy .

First, we convert the logarithmic equations to their exponential forms using the change of base formula.

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

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

Using the change of base formula, $\log_x(y) = \frac{\ln y}{\ln x}$. Since $\log_y x = \frac{9}{4}$, we have $\log_x y = \frac{4}{9}$. Thus,

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

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

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

Again, using the change of base formula, $\log_x y = \frac{4}{9}$.

Next, we express x and y in terms of each other:

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

We then compute the product xy :

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

However, upon re-evaluating and using another approach by setting $x = 2^a$ and $y = 2^b$, we find:

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

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

Finally, we find that the product xy simplifies to:

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