

本文由 AINLP 公众号整理翻译，更多 LLM 资源请扫码关注!

AINLP

我爱自然语言处理

一个有趣有AI的自然语言处理社区



长按扫码关注我们

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-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

*Contribution during internship at DeepSeek-AI.

原生稀疏注意力：硬件对齐且可原生训练的稀疏注意力

元景阳^{*1,2}, 高华左¹, 戴大麦¹, 罗俊宇², 赵亮¹, 张正艳¹, 谢振达¹, 魏延鑫¹, 王磊¹, 肖志平³, 王玉清¹, 阮冲¹, 张明², 梁文峰¹, 曾望顶¹

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

摘要

长上下文建模对于下一代语言模型至关重要，但标准注意力机制的高计算成本带来了显著的计算挑战。稀疏注意力为提高效率同时保持模型能力提供了一个有希望的方向。我们提出了NSA，一种原生可训练的稀疏注意力机制，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，将粗粒度的标记压缩与细粒度的标记选择相结合，以保持全局上下文感知和局部精度。我们的方法通过两项关键创新推进了稀疏注意力设计：(1) 我们通过算术强度平衡的算法设计实现了显著的加速，同时对现代硬件进行了实现优化。(2) 我们实现了端到端的训练，减少了预训练计算，而不会牺牲模型性能。如图1所示，实验表明使用NSA预训练的模型在通用基准、长上下文任务和基于指令的推理方面保持或超过了全注意力模型。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方面，都实现了相对于全注意力的显著加速，验证了其在整个模型生命周期中的效率。

1. 引言

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) 等多样化的实际应用推动。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、长篇文档，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，纯注意力 (Vaswani et al., 2017) 机制的高复杂性 (Zaheer et al., 2020) 成为关键的延迟瓶颈。理论估计表明，注意力 $\{v^*\}$

*Contribution during internship at DeepSeek-AI.

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

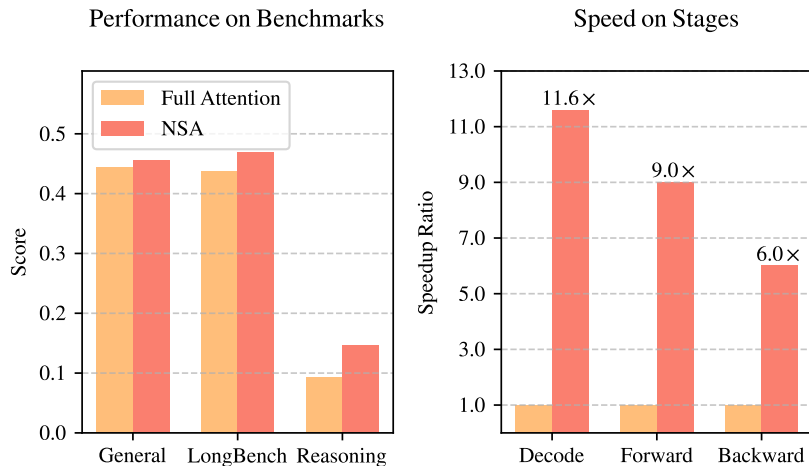


Figure 1 | Comparison of performance and efficiency between Full Attention model and our NSA. Left: Despite being sparse, NSA surpasses Full Attention baseline on average across general benchmarks, long-context tasks, and reasoning evaluation. Right: For 64k-length sequence processing, NSA achieves substantial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

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 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

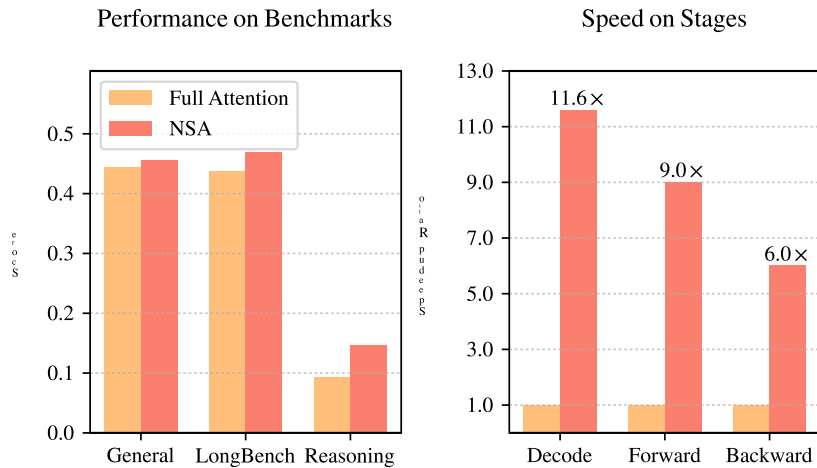


图1 | Full Attention模型与我们的NSA在性能和效率上的比较。左：尽管NSA是稀疏的，它在一般基准测试、长上下文任务和推理评估中平均超过了Full Attention基线。右：对于64k长度的序列处理，NSA在所有阶段：解码、前向传播和反向传播中，相比Full Attention实现了显著的计算加速。

在解码64k长度的上下文时，使用softmax架构的计算占总延迟的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注意力的固有稀疏性 (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缓存驱逐方法 (Li et al., 2024; Zhang et al., 2023b; Zhou et al., 2024)，块状KV缓存选择方法 (Tang et al., 2024; Xiao et al., 2024)，以及基于采样、聚类或哈希的选择方法 (Chen et al., 2024; Desai et al., 2024; Liu et al., 2024)。尽管这些策略很有前景，现有的稀疏注意力方法在实际部署中往往表现不足。许多方法未能实现与其理论增益相当的加速效果；此外，大多数方法主要集中在推理阶段，缺乏有效的训练时间支持以充分利用注意力的稀疏模式。

为了解决这些限制，有效的稀疏注意力部署必须应对两个关键挑战：(1)

Hardware-aligned inference speedup: 将理论上的计算减少转化为实际的速度提升，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)

Training-aware algorithm design: 通过可训练的操作符实现端到端的计算，以减少训练成本同时保持模型性能。这些要求对于实际应用实现快速的长上下文推理或训练至关重要。当考虑这两个方面时，现有方法仍然存在明显的差距。

为了实现更有效和高效的稀疏注意力，我们提出了NSA，一种集成了层次化令牌建模的原生可训练稀疏注意力架构。如图2所示，NSA通过将键和值组织成时间块，并通过三个注意力路径处理它们来减少每个查询的计算：压缩的粗粒度令牌、选择性保留的细粒度令牌以及用于局部上下文信息的滑动窗口。然后我们实现了专门的内核以最大化其实际效率。NSA引入了两种

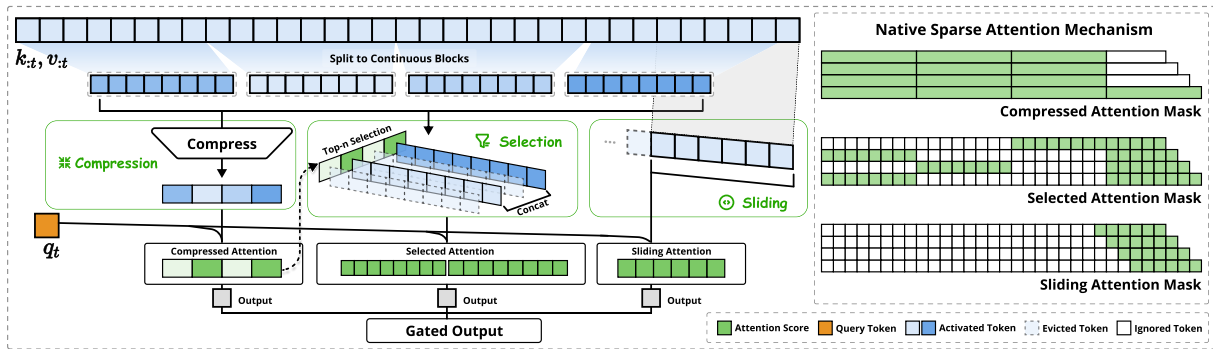


Figure 2 | Overview of NSA’s architecture. Left: The framework processes input sequences through three parallel attention branches: For a given query, preceding keys and values are processed into compressed attention for coarse-grained patterns, selected attention for important token blocks, and sliding 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.

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.

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

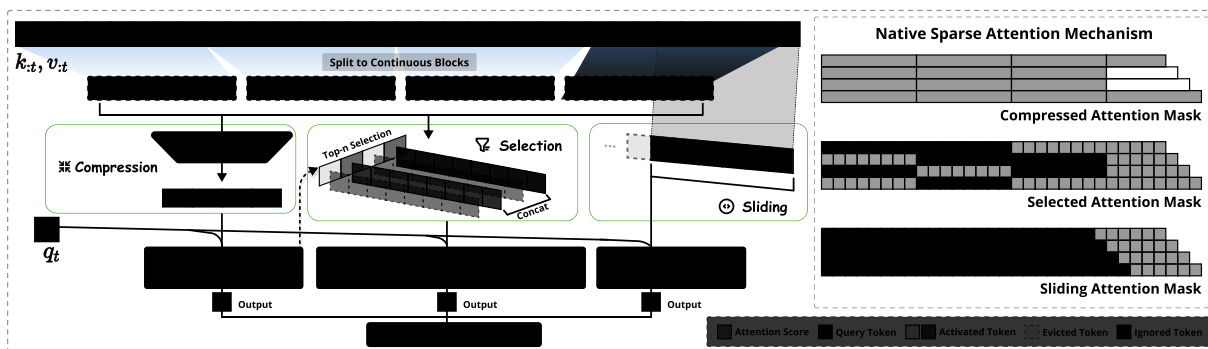


图2 | NSA架构概览。左：框架通过三个并行的注意力分支处理输入序列：对于给定的查询，前序的键和值被处理成压缩注意力以捕捉粗粒度模式，选择性注意力以关注重要令牌块，以及滑动注意力以处理局部上下文。右：每个分支产生的不同注意力模式的可视化。绿色区域表示需要计算注意力分数的区域，而白色区域表示可以跳过的区域。

对应上述关键要求的核心创新：(1) 硬件对齐系统：优化块稀疏注意力机制以充分利用 Tensor Core 和内存访问，确保算术强度平衡。(2) 训练感知设计：通过高效的算法和反向操作符实现稳定的端到端训练。这种优化使 NSA 能够支持高效的部署和端到端训练。

我们通过在真实世界语言语料库上的全面实验来评估NSA。在27B参数的Transformer主干上预训练，使用260B个token，我们评估了NSA在通用语言评估、长上下文评估和链式思维推理评估中的表现。我们进一步在A100 GPU上与优化的Triton (Tillet等, 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、前向和后向阶段都提供了显著的加速，对于更长的序列，加速比会进一步提高。这些结果验证了我们的分层稀疏注意力设计有效地平衡了模型能力和计算效率。

2. 重新思考稀疏注意力方法

现代稀疏注意力方法在降低变压器模型的理论计算复杂度方面取得了显著进展。然而，大多数方法主要在推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。在介绍我们的原生稀疏架构之前，我们通过两个关键视角系统地分析了这些限制。

2.1. 高效推理的幻觉

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

相位限制稀疏性。诸如H2O (Zhang等人, 2023b) 之类的方法应用稀疏性 $\{v^*\}$ 。

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 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

在自回归解码期间需要计算密集型的预处理（例如，注意力图计算、索引构建）。相比之下，像MIInference（Jiang等人，2024）这样的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等人，2022）推理）中的加速能力。

与高级注意力架构的不兼容。一些稀疏注意力方法无法适应现代高效的解码架构，如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-cache子集。虽然它在Multi-Head Attention (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-cache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-cache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高级架构的高效内存访问设计相冲突。

这些限制出现是因为许多现有的稀疏注意力方法专注于KV缓存减少或理论计算减少，但在先进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了先进架构和硬件高效实现的算法，以充分利用稀疏性来提高模型效率。

2.2. 可训练稀疏性的神话

我们对原生可训练稀疏注意力的追求源于对仅推理方法的分析中的两个关键见解：(1)

Performance Degradation: 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 等人 (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) **Training Efficiency Demands:** 有效处理长序列训练对于现代大语言模型 (LLM) 开发至关重要。这包括在更长的文档上进行预训练以增强模型容量，以及后续的适应阶段，如长上下文微调和强化学习。然而，现有的稀疏注意力方法主要针对推理，而训练中的计算挑战则 largely 未得到解决。这一限制阻碍了通过高效训练开发更强大的长上下文模型。此外，将现有稀疏注意力方法适应于训练的努力也暴露了挑战：

非训练组件。像 ClusterKV (Liu et al., 2024)（包括 k-means 聚类）和 MagicPIG (Chen et al., 2024)（包括基于 SimHash 的选择）中的离散操作在计算图中创建了不连续性。这些非训练组件阻止了通过 token 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。在方法中使用了基于 token 的选择策略。

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.

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 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.

像 HashAttention (Desai et al., 2024) 在注意力计算时需要从 KV 缓存中加载大量单独的 token。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分块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. 作为必要性的固有稀疏性

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，一种原生稀疏注意力框架，解决了计算效率和训练需求的问题。在接下来的章节中，我们将详细描述 NSA 的算法设计和操作符实现。

3. 方法论

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，我们首先介绍我们的方法论背景。然后我们展示 NSA 的整体框架，接着是其关键算法组件。最后，我们详细说明我们针对硬件优化的内核设计，该设计最大化了实际效率。

3.1. 背景

注意力机制在语言建模中被广泛使用，其中每个查询标记 q_t 都会计算与所有先前的键 k_i 的相关性分数，以生成值 v_i 的加权和。形式上，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 q_t 和 k_i 之间的注意力权重， 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的比重越来越大，给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例。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，计算方法是这两个硬件限制的比值。对于计算任务，当算术强度高于这个关键阈值时，任务变得计算受限（由 GPU FLOPS 限制），而低于这个阈值时，任务变得内存受限（由内存带宽限制）。

特别是在因果自注意力机制中，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，使得这些阶段在现代加速器上成为计算密集型。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只生成一个标记，同时需要加载整个键值缓存，导致算术强度低。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

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 compressed key representation is defined as:

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

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.

3.2. 总体框架

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{K}_t, \tilde{V}_t 替换公式 (1) 中的原始键值对 $k_{:t}, v_{:t}$ ，针对每个查询 q_t 。具体来说，我们正式定义优化后的注意力输出如下：

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

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

其中 \tilde{K}_t, \tilde{V}_t 是根据当前查询 q_t 和上下文记忆 $k_{:t}, v_{:t}$ 动态构建的。我们可以设计各种映射策略来获得不同类别的 $\tilde{K}_t^c, \tilde{V}_t^c$ ，并将其组合如下：

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

如图2所示，NSA有三种映射策略 $C = \{\text{cmp}, \text{slc}, \text{win}\}$ ，分别表示键值的压缩、选择和滑动窗口。 $g_t^c \in [0, 1]$ 是对应策略的门控分数 c ，通过MLP和sigmoid激活从输入特征中得出。设 N_t 表示重映射键值的总数：

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

我们通过确保 $N_t \ll t$ 来保持高稀疏率。

3.3. 算法设计

在本小节中，我们介绍了我们的重映射策略 f_K 和 f_V ：标记压缩、标记选择和滑动窗口。

3.3.1. Token Compression

通过将连续的键或值块聚合为块级表示，我们获得了捕捉整个块信息的压缩键和值。形式上，压缩键表示定义为： $\{v^*\}$

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

其中， l 是块的长度， d 是相邻块之间的滑动步幅， φ 是一个具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 $d < l$ 来缓解信息碎片化。类似的公式也适用于压缩值表示 \tilde{V}_t^{cmp} 。压缩表示捕获了更粗粒度的高层次语义信息，并减少了注意力机制的计算负担。

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:

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

where $\mathbf{p}_t^{\text{cmp}} \in \mathbb{R}^{\lfloor \frac{t-l}{d} \rfloor}$ is the attention scores between q_t and compression keys $\tilde{\mathbf{K}}_t^{\text{cmp}}$. 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:

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

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.

3.3.2. Token Selection

仅使用压缩键，值可能会丢失重要的细粒度信息，这促使我们有选择地保留个别键，值。下面我们将描述我们的高效标记选择机制，该机制能够识别并保留最相关的标记，同时计算开销较低。

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，这一策略受到两个关键因素的驱动：硬件效率考虑和注意力分数的内在分布模式。

Blockwise selection is crucial to achieve efficient computation on modern GPUs. 这是因为现代GPU架构在连续块访问方面表现出显著更高的吞吐量，而随机索引读取则不然。此外，块计算能够使Tensor Cores得到最优利用。这种架构特性已经将块状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。

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

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

重要性分数计算。计算块的重要性分数可能会引入显著开销。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，公式表示为：{v*}

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

其中 $\mathbf{p}_t^{\text{cmp}} \in \mathbb{R}^{\lfloor \frac{l}{d} \rfloor}$ 是 \mathbf{q}_t 和压缩键 $\tilde{\mathbf{K}}_t^{\text{cmp}}$ 之间的注意力分数。设 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，我们可以通过 $\mathbf{p}_t^{\text{slc}} = \mathbf{p}_t^{\text{cmp}}$ 直接获得选择块的重要性分数 $\mathbf{p}_t^{\text{slc}}$ 。对于分块方案不同的情况，我们根据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操作符。对于采用GQA或MQA的模型，如果键值缓存是在查询头之间共享的，那么必须确保在这些头之间进行一致的块选择，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被正式定义为：

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

其中上标中的 (h) 表示头索引， H 是每组查询头的数量。这种聚合确保了在同一组内的头之间选择的块具有一致性。

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:

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

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

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 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 .

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 稀疏块中的标记，表示为：

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

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

其中 $\text{rank}(\cdot)$ 表示按降序排列的排名位置， $\text{rank} = 1$ 对应最高分， \mathcal{I}_t 是选定块的索引集， Cat 表示连接操作。 $\tilde{\mathbf{K}}_t^{\text{slc}} \in \mathbb{R}^{d_k \times nl'}$ 是由压缩键组成的张量。类似的公式适用于细粒度值 $\tilde{\mathbf{V}}_t^{\text{slc}}$ 。选定的键和值随后与 \mathbf{q}_t 一起参与注意力计算，如公式 (5) 所定义。

3.3.3. Sliding Window

在注意力机制中，局部模式通常适应得更快，并可能主导学习过程，从而阻止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，该分支显式地处理局部上下文，使其他分支（压缩和选择）能够专注于学习它们各自的特征，而不会被局部模式所捷径。具体来说，我们在一个窗口 w 中保持最近的标记 $\tilde{\mathbf{K}}_t^{\text{win}} = \mathbf{k}_{t-w:t}$, $\tilde{\mathbf{V}}_t^{\text{win}} = \mathbf{v}_{t-w:t}$ ，并将不同信息源（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。这些分支的输出随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了独立的键和值。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 ($\tilde{\mathbf{K}}_t^{\text{cmp}}$, $\tilde{\mathbf{V}}_t^{\text{cmp}}$; $\tilde{\mathbf{K}}_t^{\text{slc}}$, $\tilde{\mathbf{V}}_t^{\text{slc}}$; 以及 $\tilde{\mathbf{K}}_t^{\text{win}}$, $\tilde{\mathbf{V}}_t^{\text{win}}$) 之后，我们根据公式 (5) 计算最终的注意力输出。结合上述描述的压缩、选择和滑动窗口机制，这构成了NSA的完整算法框架。

3.4. 核设计

为了在训练和预填充期间实现FlashAttention级别的加速，我们在Triton上实现了硬件对齐的稀疏注意力内核。鉴于MHA在解码过程中内存密集且效率低下，我们专注于与当前最先进的LLMs类似的具有共享KV缓存的架构，如GQA和MQA。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，但我们引入了专门的稀疏选择注意力内核设计。如果我们遵循FlashAttention将时间上连续的查询块加载到SRAM中的策略，这将导致内存访问效率低下，因为块内的查询可能需要不连续的KV块。为了解决这个问题，我们的关键优化在于不同的查询分组策略：对于查询序列中的每个位置，我们将GQA组内的所有查询头（它们共享相同的稀疏KV块）加载到SRAM中。图3展示了我们的前向传递实现。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。对于每个内循环，加载位于位置 t 的组中所有头的查询 $\mathbf{Q} \in \mathbb{R}^{[h, d_k]}$ 及其共享的稀疏键/值块索引 \mathcal{I}_t 。

2. **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.

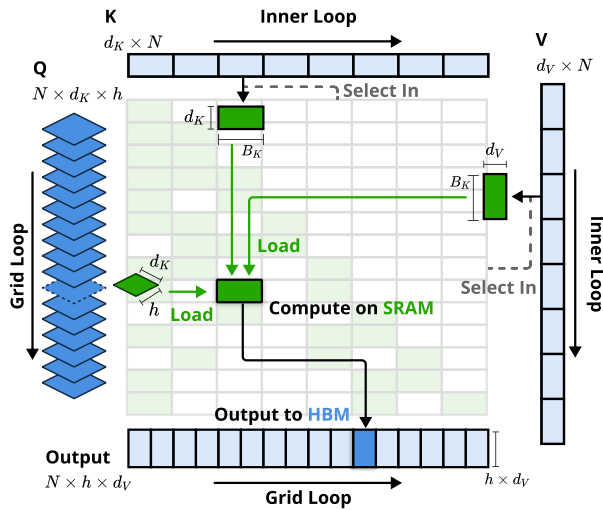


Figure 3 | Kernel design for NSA. The kernel loads queries by GQA groups (Grid Loop), fetches corresponding sparse KV blocks (Inner Loop), and performs attention computation on SRAM. Green blocks indicate data on SRAM, while blue indicates data on HBM.

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.

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

此设计通过（1）通过组间共享消除冗余的KV传输，以及（2）在GPU流式多处理器之间平衡计算 workload，实现了接近最优的算术强度 $\{v^*\}$ 。

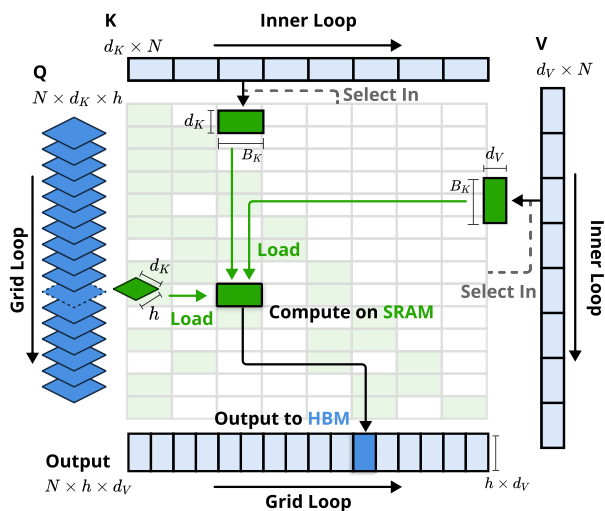


图3 | NSA的内核设计。内核通过GQA组（网格循环）加载查询，获取相应的稀疏KV块（内循环），并在SRAM上执行注意力计算。绿色块表示SRAM上的数据，而蓝色表示HBM上的数据。

4. 实验

我们通过三个角度评估NSA：(1) 一般基准测试性能，(2) 长上下文基准测试性能，以及(3) 连贯思考推理性能，与全注意力基线和最先进的稀疏注意力方法进行比较。我们把我们的稀疏计算范式的效率分析推迟到第5节，在那里我们提供了关于训练和推理速度的详细讨论。

4.1. 预训练设置

遵循最先进的LLM中的常见做法，我们的实验采用了一个结合了Grouped-Query Attention (GQA) 和Mixture-of-Experts (MoE)的主干网络，总参数量为27B，其中3B为活跃参数。该模型由30层组成，隐藏维度为2560。对于GQA，我们将组的数量设置为4，总共有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，我们使用DeepSeekMoE (Dai et al., 2024; DeepSeek-AI, 2024) 结构，具有72个路由专家和2个共享专家，并将顶级专家设置为6。为了确保训练的稳定性，第一层的MoE被替换为SwiGLU形式的MLP。

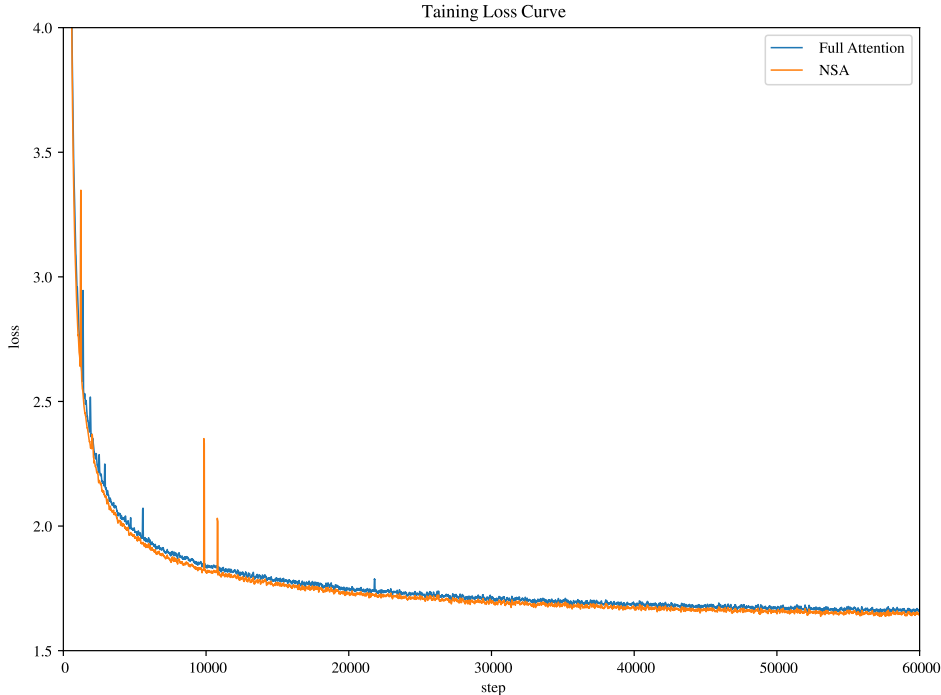


Figure 4 | Pretraining loss comparison between Full Attention and our NSA on 27B-parameter model. Both models exhibit stable convergence, with NSA achieving lower loss values.

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 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 | 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.

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 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

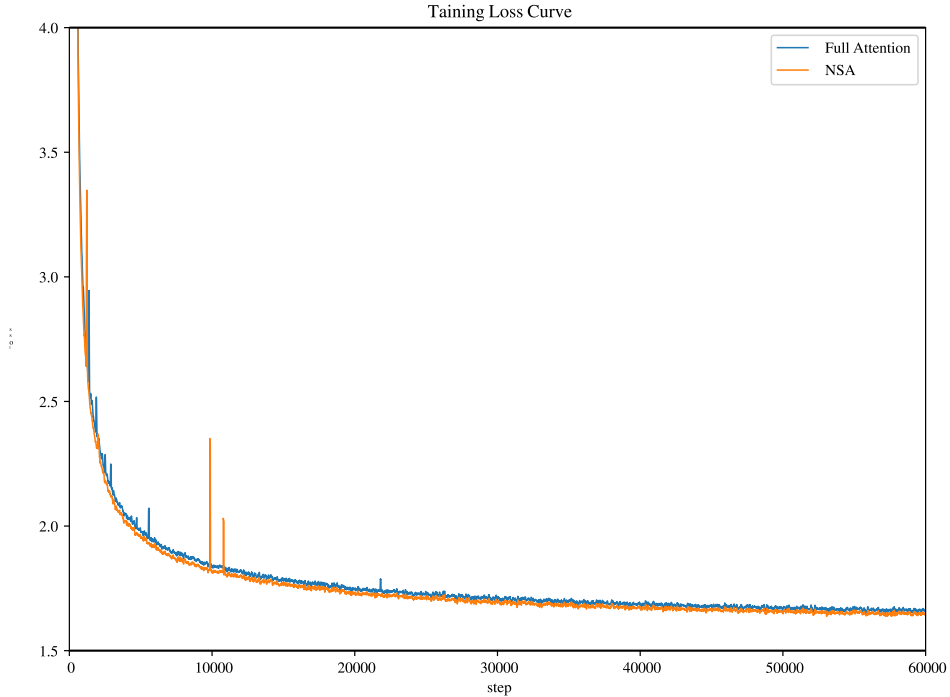


图4 | 27B参数模型的全注意力机制和我们的NSA的预训练损失比较。两种模型都表现出稳定的收敛性，NSA实现了更低的损失值。

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

表1 | 在通用基准上，全注意力基线和NSA的预训练性能比较，涵盖知识（MMLU, MMLU-PRO, CMMLU）、推理（BBH, GSM8K, MATH, DROP）和编程（MBPP, HumanEval）任务。尽管稀疏度很高，NSA在大多数基准上仍实现了 superior 平均性能。

所提出的架构在计算成本和模型性能之间实现了有效的权衡。对于NSA，我们设置压缩块大小 $l = 32$ ，滑动步长 $d = 16$ ，选择块大小 $l' = 64$ ，选择块数量 $n = 16$ （包括固定激活的1个初始块和2个局部块），滑动窗口大小 $w = 512$ 。全注意力模型和稀疏注意力模型都在2700亿个8k长度的文本token上进行了预训练，然后使用YaRN（Peng等，2024）在32k长度的文本上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。如图4所示，我们的NSA和全注意力基线的预训练损失曲线表现出稳定和平滑的下降，NSA始终优于全注意力模型。

4.2. 基线方法

除了与全注意力机制进行比较外，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), 以及 Exact-Top, 该方法首先计算全注意力得分，然后选择每个查询对应的前-n得分的键，最后在这些位置上计算注意力。这些方法

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

Table 2 | 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.

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 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 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

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

表2 | 在LongBench上我们的NSA与基线方法的性能比较，包括单文档问答、多文档问答、合成任务和代码任务类别的子集。NSA在大多数基线方法中表现最佳，包括全注意力机制。

涵盖了多种稀疏注意力机制，包括KV缓存驱逐、查询感知选择和精确的top-n稀疏选择。

对于一般评估，大多数样本的长度在稀疏注意力基线的局部上下文窗口内，这些方法实际上等效于全注意力。因此，我们仅在此设置下展示NSA与全注意力基线的比较结果。在长上下文评估中，我们对所有基线方法进行比较，所有稀疏注意力方法的稀疏度设置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，我们仅将比较限制在全注意力上，因为稀疏注意力基线不支持训练。

4.3. 性能比较

总体评估。我们对预训练的NSA和全注意力基线模型在涵盖知识、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等, 2020)、MMLU-PRO (Wang等, 2024)、CMMLU (Li等, 2023)、BBH (Suzgun等, 2022)、GSM8K (Cobbe等, 2021)、MATH (Hendrycks等, 2020)、DROP (Dua等, 2019)、MBPP (Austin等, 2021) 和HumanEval (Chen等, 2021)。结果如表1所示。尽管NSA的稀疏性，它仍实现了 superior overall performance, 超越了所有基线模型，包括在9个指标中的7个上超越了全注意力模型。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开发专门的注意力机制。这种稀疏注意力预训练机制迫使模型关注最重要的信息，通过过滤掉无关的注意力路径中的噪声，可能增强性能。在各种评估中的一致性能也验证了NSA作为通用架构的稳健性。

长上下文评估。如图5所示，在64k上下文的针 haystack (Kamradt, 2023) 测试中，NSA 在所有位置上实现了完美的检索准确性。这一性能源于我们的分层稀疏注意力设计，该设计结合了压缩 token 以高效地进行全局上下文扫描，以及选择 token 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，而对选定 token 的 token 级注意力则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度。

我们还评估了NSA o n LongBench (白等, 2023) 对比最新技术的基准测试性能。表1展示了在推理任务中，NSA在长上下文评估中实现了完美的检索准确性。

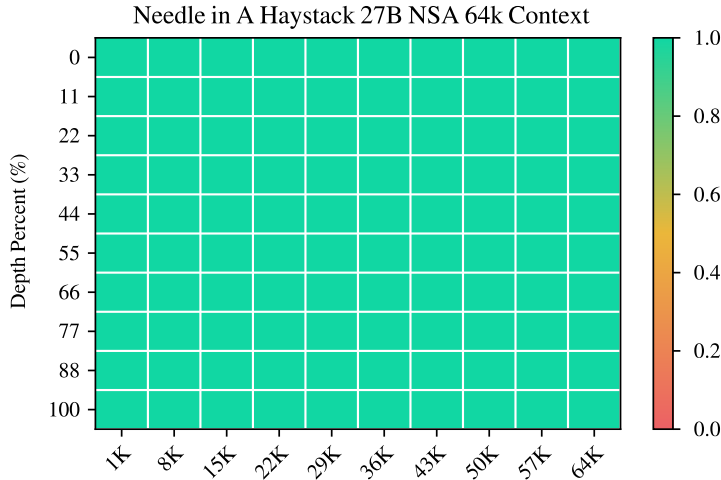


Figure 5 | Needle-in-a-Haystack retrieval accuracy across context positions with 64k context length. NSA achieves perfect accuracy through its hierarchical sparse attention design.

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 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.

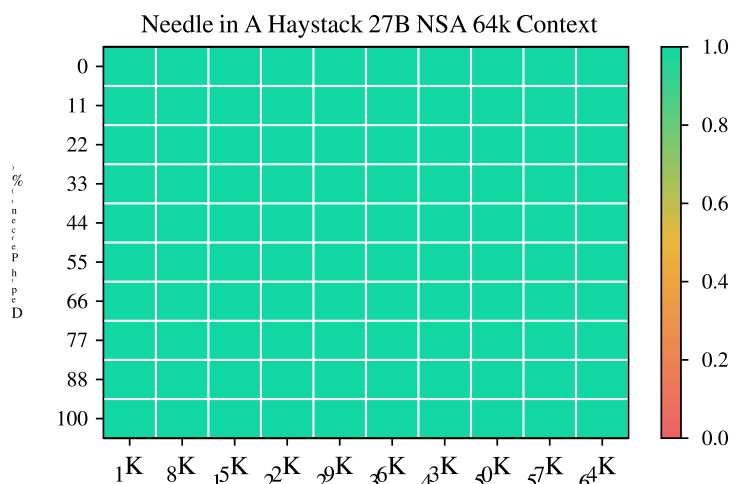


图5 | 在64k上下文长度下，不同上下文位置的针尖寻针检索准确性。NSA通过其分层稀疏注意力设计实现了完美的准确性。

方法和全注意力基线。为了确保一致的稀疏性，我们在所有稀疏注意力基线中将每个查询激活的令牌设置为2560个，这对应于NSA在处理32k序列长度时激活的令牌的平均数量。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，因为它们在所有模型中的得分较低，可能无法提供有意义的比较。如表2所示，NSA实现了最高的平均得分0.469，优于所有基线（比全注意力高+0.032，比Exact-Top高+0.046）。这种改进源于两个关键创新：(1) 我们的原生稀疏注意力设计，使在预训练期间对稀疏模式进行端到端优化成为可能，促进了稀疏注意力模块与其他模型组件之间的同步适应；(2) 层次稀疏注意力机制在本地和全局信息处理之间实现了平衡。

值得注意的是，NSA 在需要对长上下文进行复杂推理的任务上表现出色，多跳问答任务（HPQ 和 2Wiki）上比全注意力机制提高了 +0.087 和 +0.051，代码理解（LCC: +0.069）上超过了基线模型，并且在段落检索（PassR-en: +0.075）上优于其他方法。这些结果验证了 NSA 处理各种长上下文挑战的能力，其原生预训练的稀疏注意力机制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，我们通过后训练调查其获得链式思维数学推理能力的容量。鉴于强化学习在较小规模模型上的有限效果，我们采用从DeepSeek-R1的知识蒸馏，使用32k长度的数学推理痕迹的10B个token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）基准上评估这两个模型。我们使用0.7的采样温度和0.95的top-p值生成每个问题的16个响应，并获得平均分数。为了验证推理深度的影响，我们进行了两个生成上下文限制的实验：8k和16k个token，测量扩展的推理链是否提高准确性。模型预测的示例比较在附录A中提供。

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

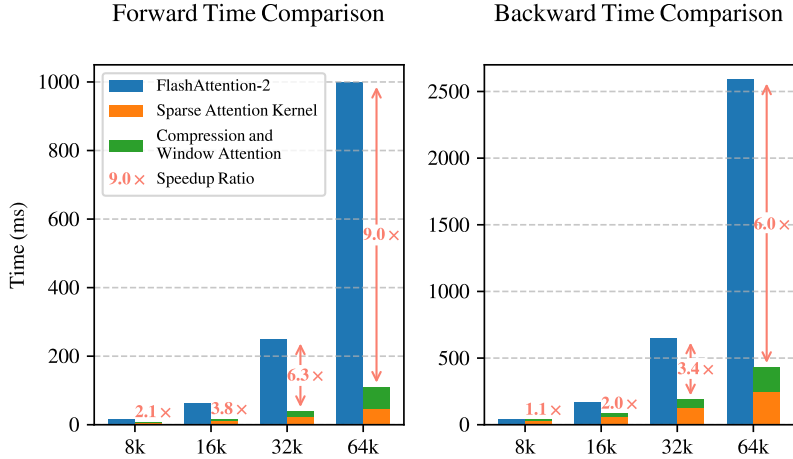


Figure 6 | Comparison of Triton-based NSA kernel with Triton-based FlashAttention-2 kernel. Our implementation significantly reduces latency across all context lengths, with the improvement becoming more pronounced as input length increases.

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, up to 9.0x forward and 6.0x backward speedup at 64k context-length. Notably, the speed advantage becomes more pronounced with longer sequences. This speedup stems from our

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

表3 | 监督微调后的基于AIME指令的评估。我们的NSA-R在8k和16k序列长度下都表现出比Full Attention-R更好的性能。

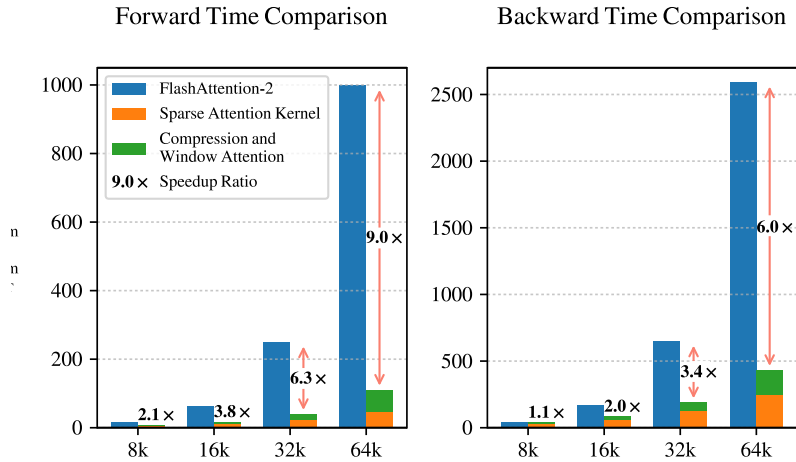


图6 | Triton-based NSA内核与Triton-based FlashAttention-2内核的比较。我们的实现显著减少了所有上下文长度的延迟，随着输入长度的增加，改进变得更加明显。

如表3所示，NSA-R在8k上下文设置下的准确率显著高于Full Attention-R (+0.075)，在16k上下文设置下这一优势仍然存在 (+0.054)。这些结果验证了原生稀疏注意力的两个关键优势：(1) 预训练的稀疏注意力模式能够高效地捕捉对复杂数学推导至关重要的长距离逻辑依赖；(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性表现优于其他方法，证实了当原生集成到训练流程中时，稀疏注意力对于高级推理任务的有效性。

5. 效率分析

我们在8-GPU A100系统上评估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 $g = 4$ ，每组头数 $h = 16$ ，查询/键维度 $d_k = 192$ ，值维度 $d_v = 128$ 。按照第4节中的相同设置，我们将NSA压缩块大小 $l = 32$ ，滑动步长 $d = 16$ ，选择块大小 $l' = 64$ ，选择块数量 $n = 16$ ，滑动窗口大小 $w = 512$ 。

5.1. 训练速度

我们比较了基于Triton实现的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是公平的。如图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，上下文长度为64k时，前向加速达到9.0x，反向加速达到6.0x。值得注意的是，随着序列长度的增加，速度优势变得更加明显。这种加速源自于我们的

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

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.

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.

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.

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

表4 | 解码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）。由于解码的低算术强度和内存绑定特性，预期的加速比大约与内存访问量成线性关系。

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

5.2. 解码速度

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，和 w 个邻近 token，其中 s 是缓存的序列长度。如表 4 所示，随着解码长度的增加，我们的方法表现出显著的延迟减少，在 64k 上下文长度时，最多可实现 11.6× 倍的加速。这种内存访问效率的优势在更长的序列中也会更加明显。

6. 讨论

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，但了解遇到的替代策略的挑战并分析注意力模式为未来的研究方向提供了宝贵的背景。我们首先检查了促使我们做出设计选择的替代令牌选择策略的挑战，然后通过可视化提供对注意力分布模式的见解。

6.1. 替代令牌选择策略的挑战

在设计NSA之前，我们探索了将现有的稀疏注意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。我们研究了基于聚类的策略，如 ClusterKV (Liu et al., 2024)。这些方法将来自同一聚类的键和价值存储在连续的内存区域中。虽然理论上适用于训练和推理，但它们面临三个显著的挑战：(1) 动态聚类机制引入的非 trivial 计算开销；(2) 由于聚类间不平衡导致的操作优化困难，特别是在 Mixture-of-Experts (MoE) 系统中，Expert Parallelism (EP) 组的执行时间不均衡导致持续的负载不平衡；(3) 实现约束，源于需要强制定期重新聚类 and 块序训练协议。这些因素的综合作用造成了显著的瓶颈，大大限制了它们在实际部署中的有效性。

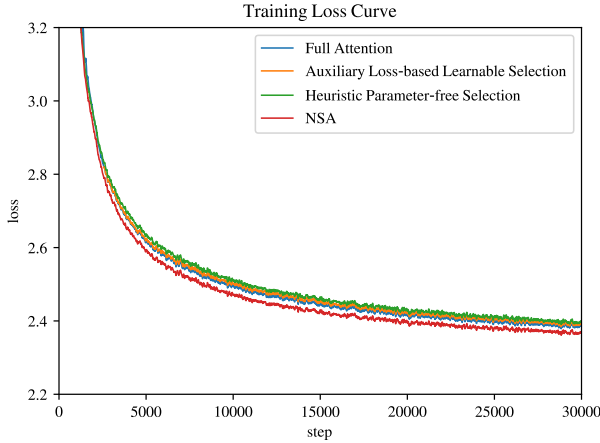


Figure 7 | Compare training loss on a 3B-parameter model with Full Attention and different token selection strategies and. Our NSA achieves better performance.

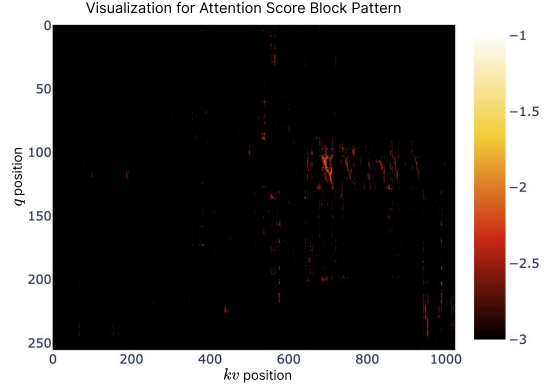


Figure 8 | Visualization of Attention Map on a Full Attention transformer. Light-colored regions indicate higher attention values. As shown in the figure, attention scores exhibit blockwise clustering distribution.

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 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.

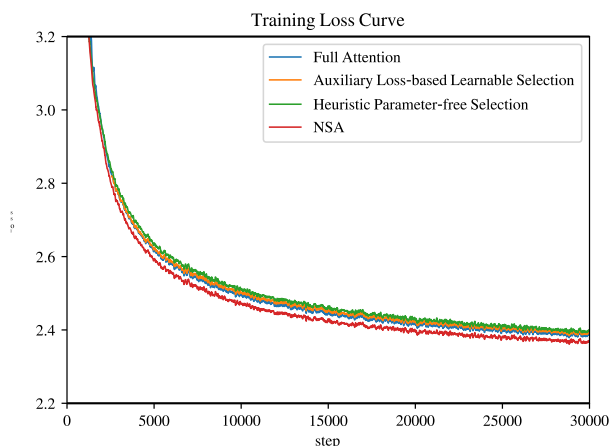


图7 | 比较在3B参数模型上使用全注意力机制和不同令牌选择策略的训练损失。我们的NSA实现了更好的性能。

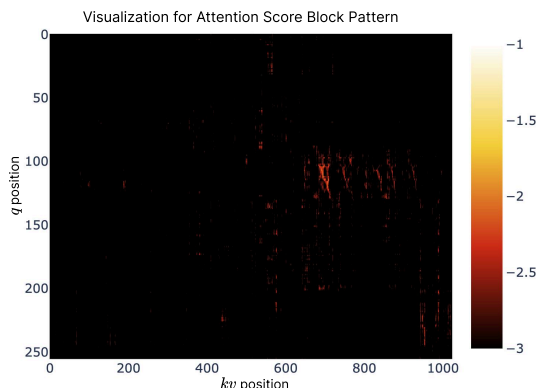


图8 | 全注意力变压器的注意力图可视化。浅色区域表示较高的注意力值。如图所示，注意力得分呈现出块状聚类分布。

其他分块选择策略。我们还考虑了与NSA不同的分块键、值选择策略，如Quest (Tang et al., 2024) 和 InfLLM (Xiao et al., 2024)。这些方法依赖于为每个分块计算重要性得分，并根据它们与 q_t 的相似性选择前 n 个分块。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得分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) 启发式无参数重要性得分计算策略的召回率较低，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和全注意力进行了比较。对于基于辅助损失的选择方法，我们为每个分块引入了额外的查询和代表性键，以估计分块的重要性得分。这些得分由每个分块内原始查询和键之间的平均注意力得分监督。对于启发式无参数选择方法，我们遵循Quest的策略，实现了使用查询和键块的坐标-wise min-max 乘积的直接选择，而无需引入额外参数。我们还探索了一种冷启动训练方法，即在最初的1000步中应用全注意力，然后再过渡到启发式分块选择。如图7所示，这两种方法的损失表现较差。

6.2. 可视化

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，我们在图8中可视化了我们预训练的27B全注意力模型的注意力图。可视化揭示了有趣的模式，注意力分数倾向于表现出块状聚类特征，相邻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个有前景的方法。块状聚类现象表明，序列中相邻的标记可能与查询标记共享某些语义关系，尽管这些关系的确切性质需要进一步研究。这一观察促使我们探索一种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

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.

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.

7. 相关工作

我们回顾了通过稀疏注意力提高注意力计算效率的现有方法。这些方法可以根据其核心策略大致分为三类：(1) 固定稀疏模式，(2) 动态令牌修剪，(3) 基于查询的选择。我们介绍了每个类别中的几个代表性工作。

7.1. 固定稀疏模式

SlidingWindow 是一种常用的方法，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 token）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全上下文理解的任务上的性能。

7.2. 动态令牌裁剪

H2O (Zhang et al., 2023b) 实现了一种自适应方法，以减少解码过程中的 KV 缓存内存使用。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 token。SnapKV (Li et al., 2024) 也引入了一种 token 剪枝策略，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. 查询感知选择

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 min-max 之间的乘积来估计的。结果分数有助于选择 top- n 重要的键值块进行注意力计算。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) 通过使用学习到的函数将查询和键映射到汉明空间，将关键令牌识别制定为一个推荐问题。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. 结论

我们介绍了NSA，一种硬件对齐的稀疏注意力架构，用于高效的长上下文建模。通过在可训练架构中集成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速度提升，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. Longbench: A bilingual, multitask benchmark for long context understanding. [arXiv preprint arXiv:2308.14508](#), 2023.
- M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. [arXiv preprint arXiv:2107.03374](#), 2021.
- Z. Chen, R. Sadhukhan, Z. Ye, Y. Zhou, J. Zhang, N. Nolte, Y. Tian, M. Douze, L. Bottou, Z. Jia, et al. Magicpig: Lsh sampling for efficient llm generation. [arXiv preprint arXiv:2410.16179](#), 2024.
- K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, et al. Training verifiers to solve math word problems, 2021. URL <https://arxiv.org/abs/2110.14168>, 2021.
- D. Dai, C. Deng, C. Zhao, R. Xu, H. Gao, D. Chen, J. Li, W. Zeng, X. Yu, Y. Wu, et al. Deepseekmoe: Towards ultimate expert specialization in mixture-of-experts language models. [arXiv preprint arXiv:2401.06066](#), 2024.
- DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. 2024. URL <https://arxiv.org/abs/2405.04434>.
- DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL <https://arxiv.org/abs/2501.12948>.
- A. Desai, S. Yang, A. Cuadron, A. Klimovic, M. Zaharia, J. E. Gonzalez, and I. Stoica. 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.

参考文献

- J. Ainslie, J. Lee-Thorp, M. de Jong, Y. Zemlyanskiy, F. Lebrón, 和 S. Sanghai. Gqa: 从多头检查点训练通用多查询变压器模型. arXiv 预印本 arXiv:2305.13245, 2023.
- J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le, 等. 使用大型语言模型进行程序合成. arXiv 预印本 arXiv:2108.07732, 2021.
- Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang, Z. Du, X. Liu, A. Zeng, L. Hou, 等. Longbench: 一个双语、多任务的长上下文理解基准. arXiv preprint arXiv:2308.14508, 2023.
- M. 陈, J. 特沃雷克, H. 俊, Q. 袁, H. P. D. O. 津托, J. 卡普兰, H. 爱德华兹, Y. 伯尔达, N. 约瑟夫, G. 布罗克曼, 等. 评估在代码上训练的大型语言模型. arXiv 预印本 arXiv:2107.03374, 2021.
- Z. 陈, R. Sadhukhan, Z. 叶, Y. 周, J. 张, N. Nolte, Y. 田, M. Douze, L. Bottou, Z. 贾, et al. Magicpig: Lsh 采样用于高效的 llm 生成. arXiv 预印本 arXiv:2410.16179, 2024.
- K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, 等. 训练验证器解决数学文字问题, 2021。URL <https://arxiv.org/abs/2110.14168>, 2021.
- D. 戴, C. 邓, C. 赵, R. 许, H. 高, D. 陈, J. 李, W. 曾, X. 余, Y. 吴, 等. Deepseekmoe: 朝向专家混合语言模型中的终极专家专业化. arXiv 预印本 arXiv:2401.06066, 2024.
- DeepSeek-AI. Deepseek-v2: 一个强大、经济且高效的专家混合语言模型。2024。URL <https://arxiv.org/abs/2405.04434>.
- DeepSeek-AI. Deepseek-r1: 通过强化学习激励大型语言模型的推理能力, 2025。URL <https://arxiv.org/abs/2501.12948>.
- A. 德赛, S. 杨, A. 库阿德龙, A. 克利莫维奇, M. 扎哈里亚, J. E. 岗萨雷斯, 和 I. 斯托伊卡. Hashatten-tion: 语义稀疏性以加快推理速度. arXiv 预印本 arXiv:2412.14468, 2024.
- D. Dua, Y. Wang, P. Dasigi, G. Stanovsky, S. Singh, 和 M. Gardner. Drop: 一个需要对段落进行离散推理的阅读理解基准. arXiv preprint arXiv:1903.00161, 2019.
- S. Ge, Y. Zhang, L. Liu, M. Zhang, J. Han, 和 J. Gao. 模型告诉你该丢弃什么: 适用于大语言模型的自适应 {v*} 缓存压缩. arXiv 预印本 arXiv:2310.01801, 2023.
- G. T. Google, P. Georgiev, V. I. Lei, R. Burnell, L. Bai, A. Gulati, G. Tanzer, D. Vincent, Z. Pan, S. Wang, 等. Gemini 1.5: 在数百万上下文标记中解锁多模态理解. arXiv 预印本 arXiv:2403.05530, 2024.
- D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song 和 J. Steinhardt. 测量大规模多任务语言理解. arXiv 预印本 arXiv:2009.03300, 2020.
- H. 姜, Q. 吴, C.-Y. 林, Y. 杨, 和 L. 邱. LlmLingua: 压缩提示以加速大语言模型的推理. arXiv preprint arXiv:2310.05736, 2023.

- H. Jiang, Y. Li, C. Zhang, Q. Wu, X. Luo, S. Ahn, Z. Han, A. H. Abdi, D. Li, C.-Y. Lin, et al. Minference 1.0: Accelerating pre-filling for long-context llms via dynamic sparse attention. arXiv preprint arXiv:2407.02490, 2024.
- G. Kamradt. LLMTest NeedleInAHaystack. GitHub repository, 2023. URL https://github.com/gkamradt/LLMTest_NeedleInAHaystack. Accessed: [Insert Access Date Here].
- H. Li, Y. Zhang, F. Koto, Y. Yang, H. Zhao, Y. Gong, N. Duan, and T. Baldwin. Cmmlu: Measuring massive multitask language understanding in chinese. arXiv preprint arXiv:2306.09212, 2023.
- Y. Li, Y. Huang, B. Yang, B. Venkitesh, A. Locatelli, H. Ye, T. Cai, P. Lewis, and D. Chen. Snapkv: Llm knows what you are looking for before generation. arXiv preprint arXiv:2404.14469, 2024.
- G. Liu, C. Li, J. Zhao, C. Zhang, and M. Guo. Clusterkv: Manipulating llm kv cache in semantic space for recallable compression. arXiv preprint arXiv:2412.03213, 2024.
- J. S. Park, J. C. O’Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein. Generative agents: Interactive simulacra of human behavior. In S. Follmer, J. Han, J. Steimle, and N. H. Riche, editors, Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology, UIST 2023, San Francisco, CA, USA, 29 October 2023– 1 November 2023, pages 2:1–2:22. ACM, 2023.
- B. Peng, J. Quesnelle, H. Fan, and E. Shippole. Yarn: Efficient context window extension of large language models. In ICLR. OpenReview.net, 2024.
- N. Shazeer. Fast transformer decoding: One write-head is all you need. CoRR, abs/1911.02150, 2019.
- M. Suzgun, N. Scales, N. Schärli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. V. Le, E. H. Chi, D. Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. arXiv preprint arXiv:2210.09261, 2022.
- J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, and S. Han. Quest: Query-aware sparsity for efficient long-context llm inference. arXiv preprint arXiv:2406.10774, 2024.
- P. Tillet, H.-T. Kung, and D. Cox. Triton: an intermediate language and compiler for tiled neural network computations. In Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, pages 10–19, 2019.
- A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. Advances in Neural Information Processing Systems, 2017.
- Y. Wang, X. Ma, G. Zhang, Y. Ni, A. Chandra, S. Guo, W. Ren, A. Arulraj, X. He, Z. Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. arXiv preprint arXiv:2406.01574, 2024.
- J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems, 35:24824–24837, 2022.
- C. Xiao, P. Zhang, X. Han, G. Xiao, Y. Lin, Z. Zhang, Z. Liu, and M. Sun. 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.

H. 江, Y. 李, C. 张, Q. 吴, X. 罗, S. Ahn, Z. 韩, A. H. Abdi, D. 李, C.-Y. 林, 等. Minference 1.0: 通过动态稀疏注意力加速长上下文 llm 的预填充. arXiv preprint arXiv:2407.02490, 2024. G. Kamradt. LLMTestNeedleInAHaystack_GitHub 仓库, 2023. URL https://github.com/gkamradt/LLMTest_NeedleInAHaystack. 访问日期: [在此插入访问日期]. H. 李, Y. 张, F. Koto, Y. 杨, H. 赵, Y. 龚, N. 段, 和 T. Baldwin. Cmmlu: 测量中文中的大规模多任务语言理解. arXiv preprint arXiv:2306.09212, 2023. Y. 李, Y. 黄, B. 杨, B. Venkitesh, A. Locatelli, H. 叶, T. 蔡, P. Lewis, 和 D. 陈. Snapkv: llm 在生成前就知道你在寻找什么. arXiv preprint arXiv:2404.14469, 2024. G. 刘, C. 李, J. 赵, C. 张, 和 M. 郭. Clusterkv: 在语义空间中操作 llm kv 缓存以实现可回忆的压缩. arXiv preprint arXiv:2412.03213, 2024.

J. S. Park, J. C. O’ Brien, C. J. Cai, M. R. Morris, P. Liang, 和 M. S. Bernstein. 生成代理: 人类行为的交互式模拟体. In S. Follmer, J. Han, J. Steimle, 和 N. H. Riche, 编辑, 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, 和 E. Shippole. Yarn: {v*} 大型语言模型的高效上下文窗口扩展. In ICLR. OpenReview.net, 2024.

N. Shazeer. 快速变压器解码: 仅需一个写头. CoRR, abs/1911.02150, 2019.

M. 苏兹古恩, N. 斯凯尔斯, N. 舍尔利, S. 盖尔曼, Y. 泰, H. W. 仲, A. 乔德赫里, Q. V. 莱, E. H. 奇, D. 周, 等. 挑战大基准任务及链式思维是否能解决它们. arXiv 预印本 arXiv:2210.09261, 2022.

J. Tang, Y. Zhao, K. Zhu, G. Xiao, B. Kasikci, 和 S. Han. Quest: Query-aware sparsity for efficient long-context llm inference. arXiv preprint arXiv:2406.10774, 2024.

P. Tillet, H.-T. Kung, 和 D. Cox. Triton: 一种用于分块神经网络计算的中间语言和编译器. In Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, 第 10–19 页, 2019.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, 和 I. Polosukhin. Attention is all you need. Advances in Neural Information Processing Systems, 2017.

Y. 王, X. 马, G. 张, Y. 倪, A. Chandra, S. 郭, W. 任, A. Arulraj, X. 何, Z. 江, 等. MMLU-Pro: 一个更强大和具有挑战性的多任务语言理解基准. arXiv 预印本 arXiv:2406.01574, 2024.

J. 魏, X. 王, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. 周, 等. Chain-of-thought prompting elicits reasoning in large language models. 神经信息处理系统进展, 35:24824–24837, 2022.

C. 肖, P. 张, X. 韩, G. 肖, Y. 林, Z. 张, Z. 刘, 和 M. 孙. Infillm: 利用高效上下文记忆实现无需训练的长上下文外推. In 第三十八屆年度神經信息處理系統會議, 2024.

- G. Xiao, Y. Tian, B. Chen, S. Han, and M. Lewis. Efficient streaming language models with attention sinks. [arXiv preprint arXiv:2309.17453](https://arxiv.org/abs/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](https://arxiv.org/abs/2008.00811), 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](https://arxiv.org/abs/2210.02352), 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](https://arxiv.org/abs/2310.07551), 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](https://arxiv.org/abs/2405.14208), 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](https://arxiv.org/abs/2310.12703), 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](https://arxiv.org/abs/2410.09342), 2024.

G. 肖, Y. 天, B. 陈, S. 韩, 和 M. Lewis. 带有注意力池的高效流式语言模型. arXiv preprint arXiv:2309.17453, 2023. M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. 王, L. 杨, 等. Big bird: 用于更长序列的变压器. *Advances in neural information processing systems*, 33:17283–17297, 2020. E. Zelikman, Y. Wu, J. Mu, 和 N. D. Goodman. Star: 用推理引导推理. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, 和 A. Oh, 编辑, *Advances in Neural Information Processing Systems 35: 2022年神经信息处理系统会议, NeurIPS 2022*, 美国新奥尔良, 2022年11月28日 – 12月9日, 2022.

F. 张, B. 陈, Y. 张, J. 郭, J. 刘, D. 詹, Y. 毛, J. 楼, 和 W. 陈. Repocoder: 通过迭代检索和生成实现仓库级别的代码补全. 在 H. 布阿莫, J. 皮诺, 和 K. 巴利, 编辑, 2023 年经验方法在自然语言处理会议论文集, EMNLP 2023, 新加坡, 2023 年 12 月 6 日–10 日, 第 2471–2484 页. 计算语言学协会, 2023a.

K. 张, J. 李, G. 李, X. 施, 和 Z. 金. Codeagent: 通过工具集成代理系统增强代码生成以应对现实世界中的仓库级编码挑战. 在 L. 库, A. 马丁斯, 和 V. 斯里库马尔, 编辑, 《计算语言学协会第 62 届年会论文集 (第 1 卷: 长论文)》, ACL 2024, 泰国曼谷, 2024 年 8 月 11–16 日, 第 13643–13658 页.

Z. 张, Y. 盛, T. 周, T. 陈, L. 郑, R. 蔡, Z. 宋, Y. 田, C. Ré, C. 巴雷特, 等. H2o: 用于大型语言模型高效生成推理的重击者神谕. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023b.

Z. 周, C. 李, X. 陈, S. 王, Y. 潮, Z. 李, H. 王, R. 安, Q. 石, Z. 谭, 等. Llm × mapreduce: 使用大型语言模型简化长序列处理. arXiv 预印本 arXiv:2410.09342, 2024.

A. 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 $|\log_2(x^4y^3z^2)|$ 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:

`<think>...</think>` (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 $x^4y^3z^2$:

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

Multiplying these together:

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

Taking the logarithm base 2:

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

The absolute value is:

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

A. AIME 结果示例

源文本: Prompt: 翻译文本: 提示:

“|用户| 设 x , y 和 z 是满足以下方程组的正实数:

翻译内容: - 保持公式符号 $\{v^*\}$ 不变。

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

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

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

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

我们的结果:

\langle think \rangle ... \langle /think \rangle (2275 个标记)

给定方程组:

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

我们将这些对数方程转换为它们的指数形式:

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

将每个变量表示为 x , y 和 z 的函数:

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

将这些表达式互相代入, 我们发现:

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

接下来, 我们计算 $x^4y^3z^2$:

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

将这些相乘:

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

取以2为底的对数:

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

绝对值是: $\{v^*\}$

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

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

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

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

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:

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

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

33

基线结果:

<think>...</think> (9392 个标记)

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解该系统:

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

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

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

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 $|\log_2(x^4y^3z^2)|$:

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

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, $m + n = 115 + 16 = 131$.

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:

<think>...</think> (15147 tokens)

Given the equations:

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：

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

因此，我们有：{v*}

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

我们需要找到 $|\log_2(x^4 y^3 z^2)|$ ：

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

转换为一个公分母 (48)：

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

将这些相加：

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

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

131

源文本: Prompt:翻译文本: 提示:

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

□

我们的结果:

<think>...</think> (15147 个标记)

给定方程:

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:

<think>...</think> (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.

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

我们首先将对数的幂规则应用于两个方程。

对于第一个方程：{v*}

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

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

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

对于第二个方程：

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

使用幂规则，我们得到：{v*}

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

自从 $\log_y(x) = z$ ，这简化为：

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

我们现在有两个表达式：

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

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

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

因此， xy 的值是 25。

基线结果：

<think>...</think> (16223 个标记)

已知问题中实数 x 和 y 满足 $x, y > 1$ 满足方程 $\log_x(y^x) = 10$ 和 $\log_y(x^{4y}) = 10$ ，我们需要找到 xy 的值。

首先，我们使用换底公式将对数方程转换为它们的指数形式。

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

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

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,

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

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

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

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:

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

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

Finally, we find that the product xy simplifies to:

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

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

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

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

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

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

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

再次使用换底公式, $\log_x y = \frac{4}{9}$ 。

接下来, 我们用彼此来表示 x 和 y :

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

然后我们计算乘积 xy :

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

然而, 在重新评估并使用另一种方法, 通过设置 $x = 2^a$ 和 $y = 2^b$, 我们发现:

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

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

最后, 我们发现产品 xy 简化为:

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