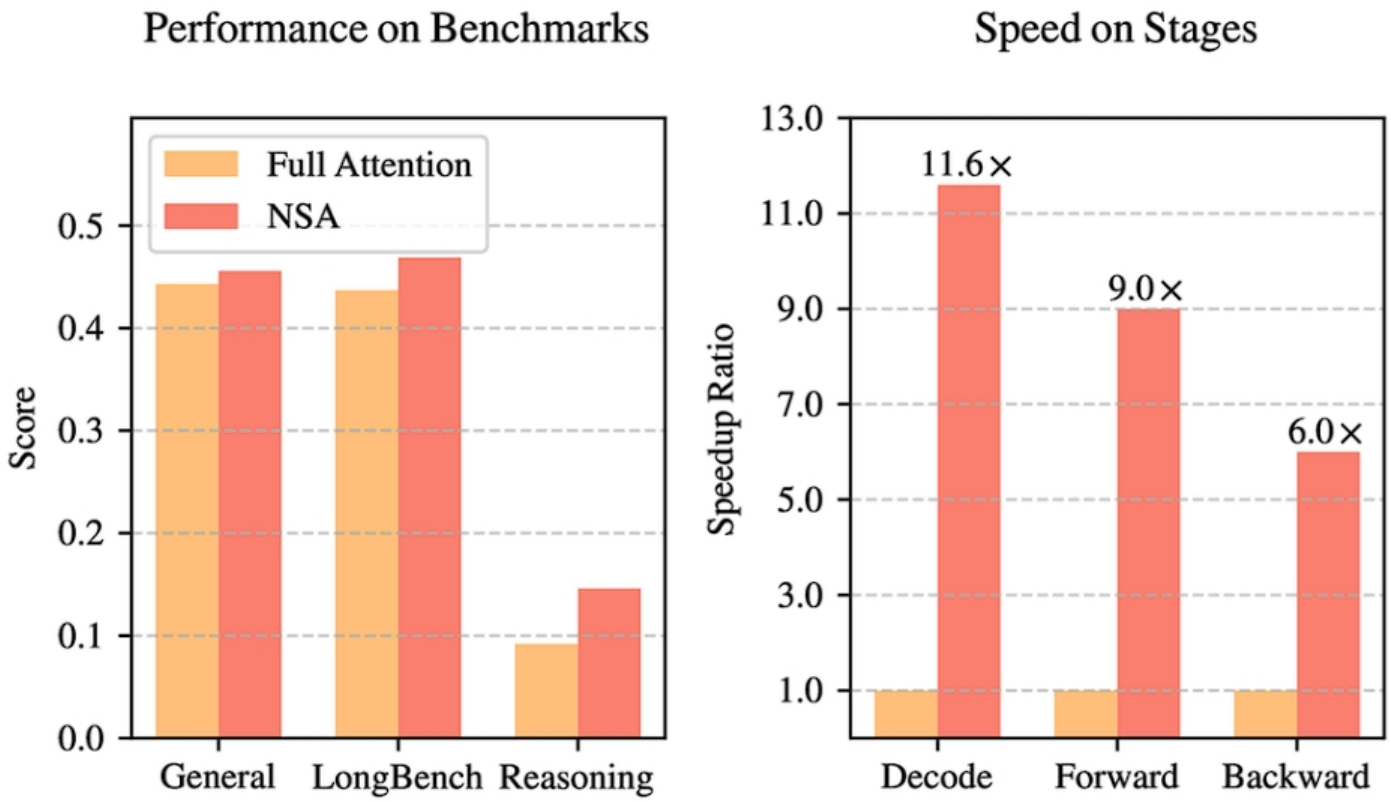
NSA

Native Sparse Attention（NSA）

注意力面临的严重的问题是计算复杂度过高（$O(N^2)$），这篇文章提出一种原生可训练的稀疏注意力机制（**Native Sparse Attention**），集成了分层字符建模，将算法创新与与硬件对齐的优化相结合，以实现高效的长文本建模。

如下图所示，NSA尽管是稀疏的，但在基准测试、长文本任务、推理任务上都取得了比全注意力更优的结果，而计算速度（解码、前向传播、反向传播）却远快于全注意力计算。



如下图所示，NSA 通过将键和值组织成时间块并通过三条注意力路径处理它们来减少每个查询的计算：压缩的粗粒度字符、选择性保留的细粒度字符和用于局部上下文信息的滑动窗口。此外为了使NSA能够同时支持高效部署和端到端训练，还引入了以下两个算法：

1. **硬件对齐系统**：优化块状稀疏注意力以利用 Tensor Core 和内存访问，确保均衡的算术强度。
2. **训练感知设计**：通过高效的算法和反向算子实现稳定的端到端训练。

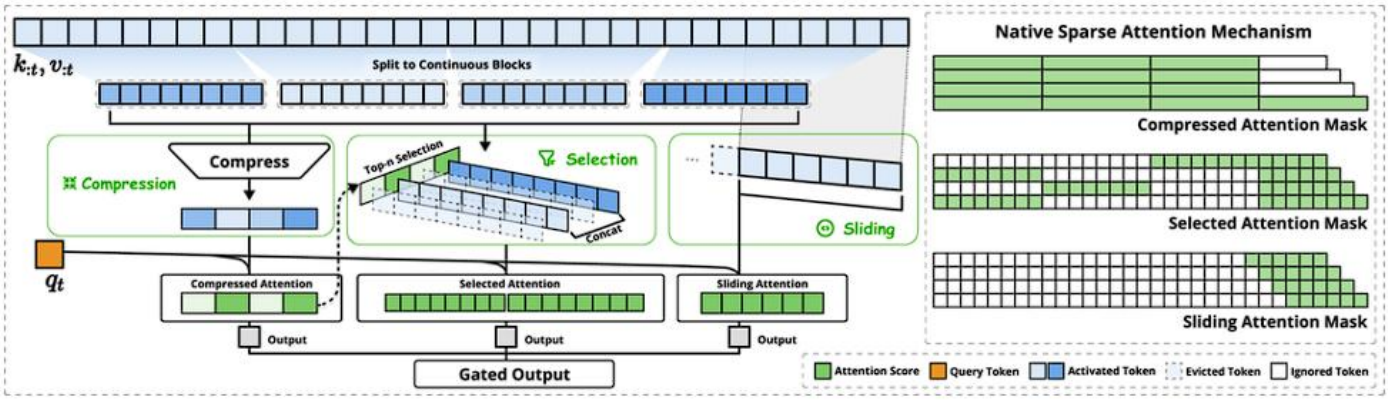


Figure 2: Overview of NSA's architecture. Left: The framework processes input sequences through three parallel attention branches: For a given query, preceding keys and values are processed into compressed attention for coarse-grained patterns, selected attention for important token blocks, and sliding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

稀疏注意力的缺点

现有的稀疏注意力方法主要在推理过程中应用稀疏性，同时保留预训练的全注意力骨干网络，这可能会引入架构偏差，限制其充分利用稀疏注意力优势的能力。具体来说面临以下的问题：

- **只能在某阶段实现稀疏性**：只在**自回归解码阶段**或者**预填充阶段**引入稀疏性，但另外几个阶段仍然需要全注意力计算。

❤️ **预填充Pre-filling**：

预填充指的是在大模型运行之前，将一部分已知或可以预先计算的内容输入到模型中，换句话说，就是通过提前处理某些不变或确定的输入来降低模型在推理阶段的计算负担。其主要运用在以下方面：

1. **输入优化**：在模型开始处理用户输入之前，Pre-filling可以将一些固定的上下文或不随用户输入改变的内容预先输入模型，计算出hidden states和KV cache。例如，在对话生成模型中，我们可以将与话题相关的背景知识或设定的固定信息提前输入到模型中，这样在实际对话过程中，模型无需重复处理这些背景信息。
2. **动态缓存**：许多模型在处理长文本或上下文时，会受到性能和时间消耗的限制。通过动态缓存（例如kv cache），模型可以在每次计算中保存一部分隐含层的计算结果，并在接下来的计算中复用这些缓存的数据。这种方式可以减少重复计算，加快响应速度。
3. **剪枝操作**：通过去除模型中不必要的层或节点，Pre-filling可以使模型结构更为简洁。这样在实际推理过程中，模型可以集中计算核心节点的权重，减少不必要的计算开销。

预填充能加快模型响应速度和节省计算资源。但pre-filling本身也需要大量的计算资源，此外如果后续用户的输入于预填充数据的数据不一致，可能导致模型生成错误。

- **不兼容MQA和GQA**：Multi-Query Attention 和 Group-Query Attention 通过跨多个查询头共享 KV 来显著减少解码过程中的内存访问瓶颈，所需的 KV 缓存内存访问量对应于同一 GQA 组内所有查询头选择的并集，仍然相对较高。这意味着即使共享了 Key 和 Value，访问的内存位置仍然是离散的，因为每个 Query 头或组内的 Query 头可能选择不同的 KV 缓存位置，这导致虽然稀疏注意力减少了计算量，但离散的内存访问减慢了计算速度。

离散的内存访问：

这是操作系统中的概念（算法工程师一定要学好数据结构和操作系统），指在计算机系统中，程序或硬件以非连续的方式访问内存中的数据。与连续的内存访问（如顺序读取或写入）相反，这种随机和间隔性的访问缓存命中率很低，无法利用缓存的局部性原理，因而需要频繁的从内存加载数据，这会导致模型推理速度变慢。

- **性能下降**：在推理时应用稀疏注意力会降性能。研究人员证明保留top-20%的注意力只能取得全注意力70%的性能。
- **忽略训练效率**：现有的稀疏注意力方法主要针对推理，在很大程度上忽略了训练中的计算挑战。
- **不可训练组件**：有些方法（ClusterKV、MagicPIG）中的离散运算会在计算图中产生不连续性。 这些不可训练的组件阻止了梯度通过字符选择过程的流动，限制了模型学习最佳稀疏模式的能力。
- **反向传播低效**：可训练的稀疏注意力（HashAttention等方法）使用的字符粒度选择策略，导致在注意力计算过程中，需要从KV缓存中加载大量单个字符。非连续的内存访问限制了Flash attention等加速注意力计算的方法的应用，减慢了训练速度。

基础知识

Attention的计算定义为：

$$\mathbf{o}_t = Attn(\mathbf{q}_t, \mathbf{k}_{:t}, \mathbf{v}_{:t}) = \sum_{i=1}^{t} \frac{a_{t,i} v_i}{\sum_{j=1}^{t} a_{t,j}} \quad (1)$$

其中 $a_{t,i}$ 表示 q_t 和 k_i 之间的注意力，注意力计算量随着序列长度增加而指数级增长。

算术强度：定义为计算操作与内存访问的比率，每个 GPU 都有一个算术强度，它由其峰值计算能力和内存带宽决定，计算公式为这两者硬件限制的比率。 对于计算任务，高于此阈值的算术强度变得计算受限（受GPU FLOPS限制），而低于此阈值则变得内存受限（受内存带宽限制）。

- 在训练和预填充阶段，批量矩阵乘法和注意力计算导致高算术强度。因此需要减少训练和预填充期间的计算成本。

- 相比之下，自回归解码受内存带宽限制，因为它每次前向传播生成一个 token，同时需要加载整个键值缓存，导致算术强度较低。因此需要减少解码期间的内存访问。

接下来为了使用稀疏注意力，使用更紧凑和信息密集的 $\tilde{K}_t, \tilde{V}_t$ 代替公式（1）中的原始 $\mathbf{k}_{:t}, \mathbf{v}_{:t}$ ，$\tilde{K}_t, \tilde{V}_t$ 是基于当前查询 q_t 和上下文记忆 $\mathbf{k}_{:t}, \mathbf{v}_{:t}$ 构建的：

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

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

可以设计各种映射策略来获得不同类别的 $\tilde{K}_t^c, \tilde{V}_t^c$ 组合起来：

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

NSA设计了三种映射策略 $\mathcal{C} = cmp, slc, win$ ，分别代表键和值的压缩、选择和滑动窗口。用 $g_t^c \in [0,1]$ 表示对应策略的门控分数，通过MLP和sigmoid激活函数从输入特征中导出，用 N_t 表示重新映射后键和值的总数，保证 $N_t << t$ 以实现高稀疏率。

字符压缩

通过将连续的键或值块聚合为压缩块，我们获得了能捕获这些块信息的压缩键和值：

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

其中 l 是块长度， d 是相邻块之间的滑动步长，φ是一个可学习的MLP模型，使用块内位置编码将块中的键映射到单个压缩键。要求 $d < l$ 以减轻信息碎片，对V采用相同的计算。

字符压缩捕获粗粒度的更高层次的语义信息，并减少了注意力机制的计算负担。但可能会丢失细粒度的信息。

字符选择

对于每个用户查询字符，只选择重要的键块和值块参与计算。

块选择

选择在空间连续的块中的键值序列，这是因为：

- GPU对于连续的块的访问吞吐量更高，此外，分块计算能够充分利用GPU中的 Tensor Cores（例如 flash attention）。
- 块选择遵循注意力分数的固有分布模式，现有工作证明注意力分数通常表现出空间连续性，相邻的键往往具有相似的重要性级别。

重要性分数

为了实现块选择，就要衡量键块和值块的重要性，利用字符压缩计算中产生中间注意力分数，可以利用这些分数来推导选择块的重要性分数：

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

令l'表示选择块的大小。当压缩块和选择块的块大小相同时，即l'=l时，我们可以直接通过 $\mathbf{p}_t^{slc} = \mathbf{p}_t^{cmp}$ 直接获得选择块的重要性分数 $\mathbf{p}_t^{slc}$ 。块大小不同时，则有：

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

其中 $[\cdot]$ 表示索引。对于采用 GQA 或 MQA 的模型，其中键值缓存跨查询头共享，必须确保这些头之间的一致块选择，以最大限度地减少解码期间的 KV 缓存加载。组内各个查询头采用相同的重要性分数，确保在同一组内的各查询头选择一致地块：

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

其中上标h是查询头索引，H是每组中查询头的数量。根据重要性分数，只保存前n的稀疏块，其中rank表示排名，rank=1对应最高分， $\mathcal{I}_t$ 表示所选快索引的集合，Cat表示拼接运算。对V采用相同的计算，只有重要性高的键和值块参与注意力计算。

$$\mathcal{I}_t = \{i | \text{rank}(\mathbf{p}_t^{\text{slc}'}[i]) \leqslant n\} \tag{11}$$

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

滑动窗口

为了防止局部模式（模型在处理输入数据时，过于关注局部区域或局部特征的模式，忽略了在整个输入序列上进行全局计算）主导学习过程，影响模型从压缩和选择tokens中学习，NSA引入了专门的滑动窗口分支来处理局部context。通过维护一个窗口w内的相近的tokens $\tilde{K}_t^{win} = k_{t-w:t}, \tilde{V}_t^{win} = v_{t-w:t}$ ，并将不同信息源（压缩符元、选择符元、滑动窗口）的注意力计算隔离到不同的分支中。最后通过学习的门控机制聚合分支输出，为了进一步防止注意力分支之间的捷径学习（指模型在训练过程中，没有学习到问题的核心特征，而是利用数据中的某些简单模式、偏差或噪声来快速优化损失函数，可以通过增大模型参数量解决），为三个分支提供独立的键值。

将以上三种键和值应用到公式（5）中，就得到了最终的NSA。

通俗解释以上三种映射策略

假设你需要跟房间中的一些人对话。

- **字符压缩：**可以把它想象成**过滤掉一部分不重要的声音**，只关注部分人的讲话。例如，我们可以设置规则，只听取每四个人中的一个人的声音，这样能减少计算负担，同时保留大致的信息。
- **字符选择：**这一策略类似于**只关注真正有价值的对话**。在 NSA 算法中，会挑选出最相关的信息，而不是平均对待所有内容。比如，在一场聚会中，你可能会特别留意讲故事最精彩的人，而忽略那些闲聊的人。
- **滑动窗口：**这个方法类似于**在房间里来回走动，逐个聆听不同的小团体对话**。相比于全局关注，它更加灵活，可以确保模型不会错过局部的重要信息。

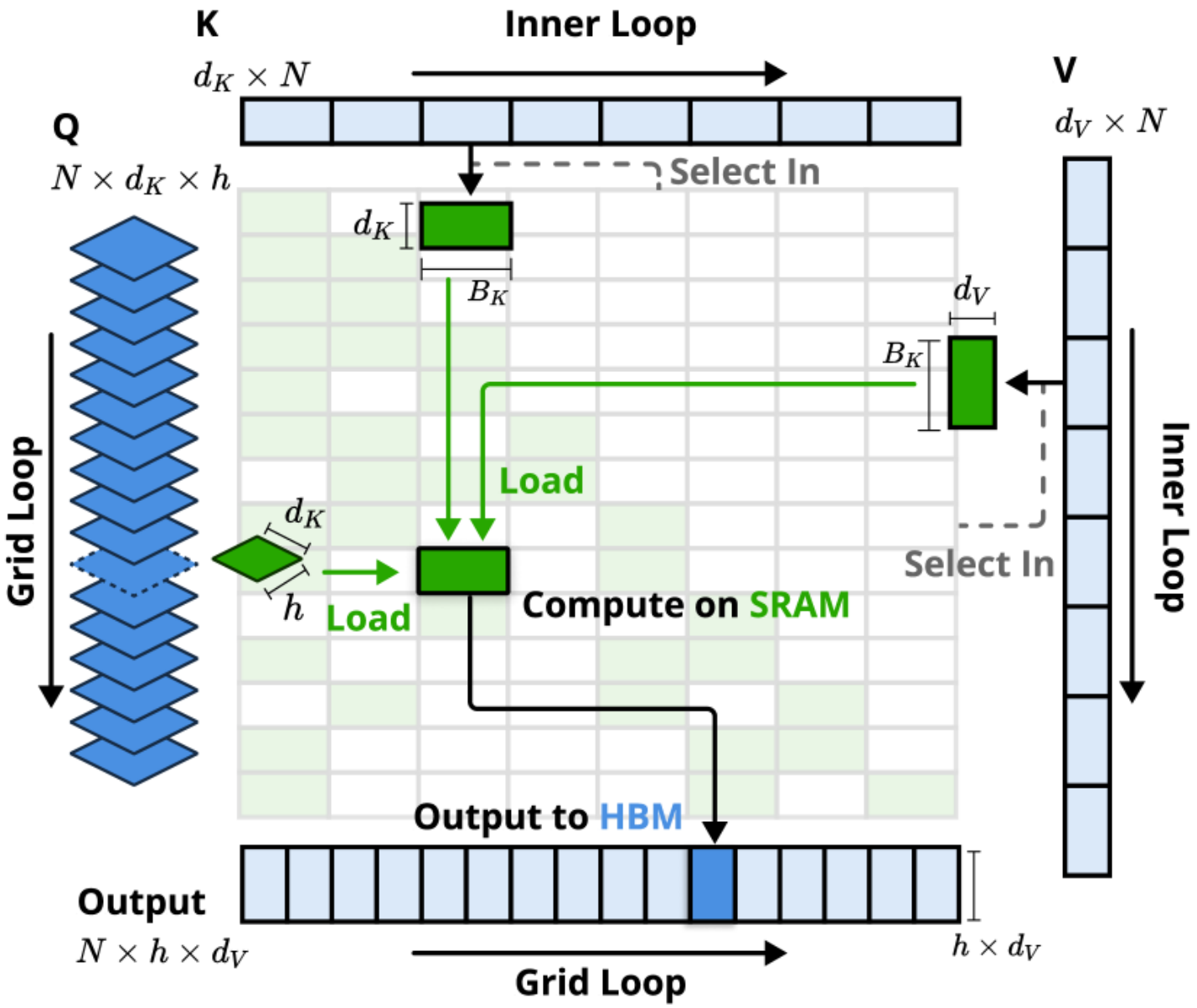
内核设计

为了在训练和预填充阶段实现类似FlashAttention的加速，NSA基于Triton（OpenAI开源的一个高效的编译器和编程框架，专门用于加速GPU上的深度学习模型）实现了硬件对齐的稀疏注意力内核。由于当前最先进的LLMs中都采用共享KV缓存的架构（如GQA和MQA），NSA对稀疏选择注意力进行了专门的内核设计。

对于查询序列上的每个位置，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。这种内核设计具有以下关键特征：

- **以组为中心的数据加载**：在每个内循环中，加载组中位置t处所有头的查询 $Q \in R^{[h,dk]}$ 及其共享的稀疏键/值块索引 $\mathcal{I}_t$ 。
- **共享KV获取**：在内循环中，按 $\mathcal{I}_t$ 顺序加载连续的键/值块到SRAM中，以最小化内存加载。
- **网格外循环**：由于不同查询块的内循环长度（与所选块数n成比例）几乎相同，将查询/输出循环放在Triton的网格调度器中，以简化和优化内核。

该设计通过组间共享消除冗余的KV传输，平衡GPU流多处理器上的计算工作负载。如下图所示，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，蓝色块表示HBM上的数据。



实验结果

在基准测试、长文本任务、推理任务上都取得了一定的性能提升。最显著的优点在于，在64k序列长度下，训练阶段，前向加速高达9.0×倍，反向加速高达6.0×倍；解码阶段，NSA的解码速度比Full Attention快11.6倍。

🎨 **NSA有以下创新点**：

- **显著的加速效果**: 通过算法设计和现代硬件实现的优化，在处理64k长序列时，NSA在解码、前向传播和后向传播阶段相比全注意力模型实现了显著的速度提升。
- **端到端训练支持**: NSA支持端到端训练，减少了预训练计算量而不牺牲模型性能，使得稀疏注意力模式可以在整个模型生命周期中有效利用。
- **动态分层稀疏策略**: NSA结合了粗粒度的token压缩、细粒度的token选择和滑动窗口策略，既保留了全局上下文感知，又保持了局部精度。

- **硬件对齐的系统优化**: 通过优化块状稀疏注意力以利用Tensor Core和内存访问，确保平衡的算术强度，最大化实际效率。
- **训练感知的算法设计**: 通过高效的算法和后向操作器实现稳定的端到端训练，支持高效部署和端到端训练。