

Faster Neighborhood Attention

Reducing the $\mathcal{O}(n^2)$ Cost of Self Attention at the Threadblock Level

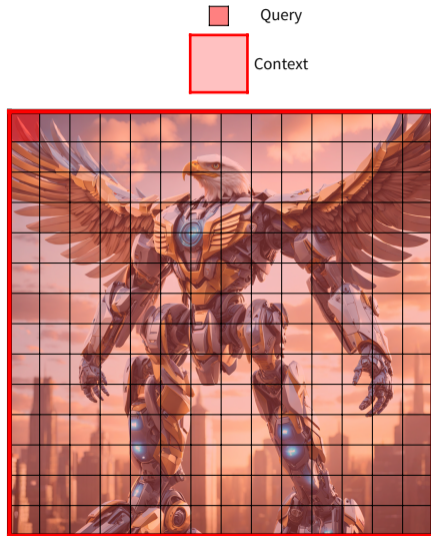
Ali Hassani ¹ Wen-mei Hwu ^{2,3} Humphrey Shi ^{1,3}

¹Georgia Tech, ²NVIDIA, ³UIUC .

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

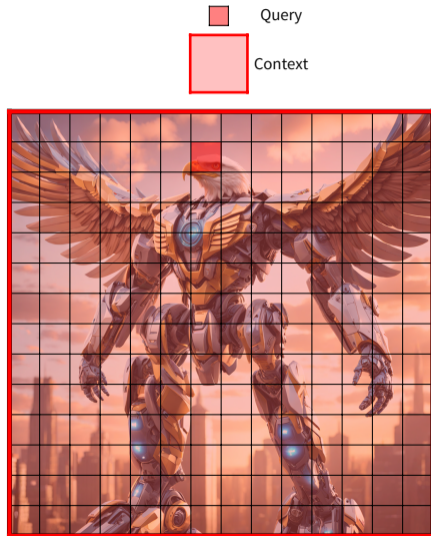
Self Attention

Any-to-any interaction between all n input tokens/pixels.



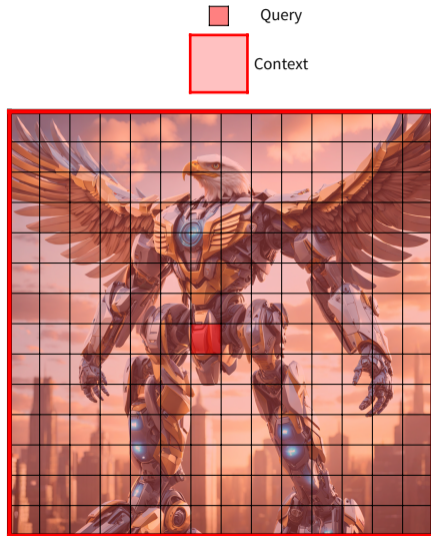
Self Attention

Any-to-any interaction between all n input tokens/pixels.



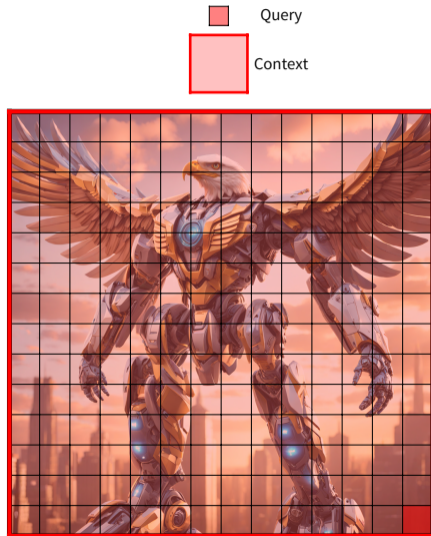
Self Attention

Any-to-any interaction between all n input tokens/pixels.



Self Attention

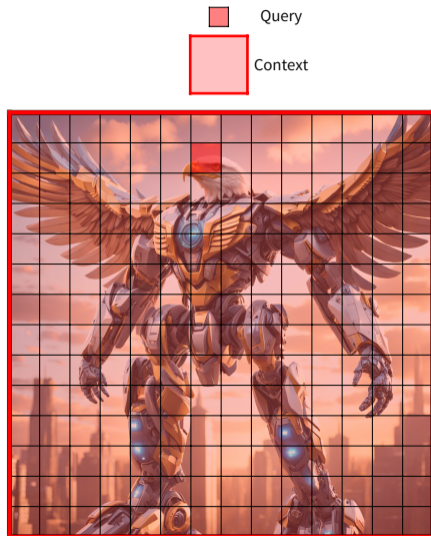
Any-to-any interaction between all n input tokens/pixels.



Self Attention

Any-to-any interaction between all n input tokens/pixels.

Attention weights, $\mathbf{P} \in \mathbb{R}^{n \times n}$, which gives us the notorious $\mathcal{O}(n^2)$ time (and space) complexity.

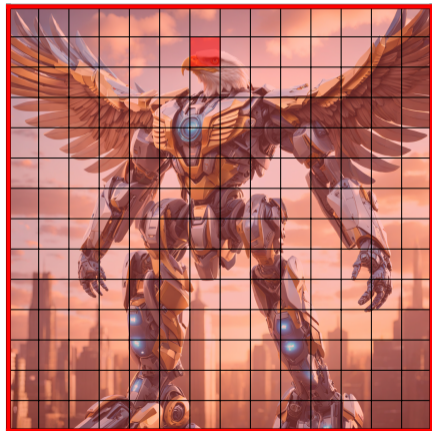


Self Attention

Any-to-any interaction between all n input tokens/pixels.

Attention weights, $\mathbf{P} \in \mathbb{R}^{n \times n}$, which gives us the notorious $\mathcal{O}(n^2)$ time (and space) complexity.

As n grows, This problem will be bounded by:



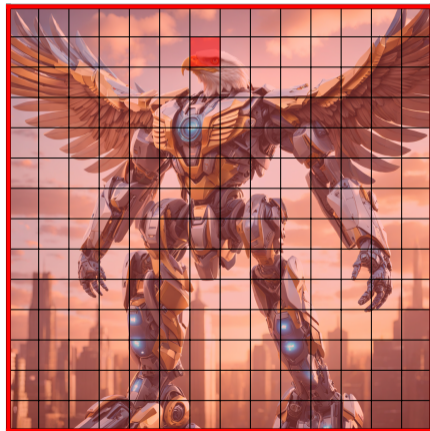
Self Attention

Any-to-any interaction between all n input tokens/pixels.

Attention weights, $\mathbf{P} \in \mathbb{R}^{n \times n}$, which gives us the notorious $\mathcal{O}(n^2)$ time (and space) complexity.

As n grows, This problem will be bounded by:

1. Memory capacity (assuming \mathbf{P} is stored in global memory)



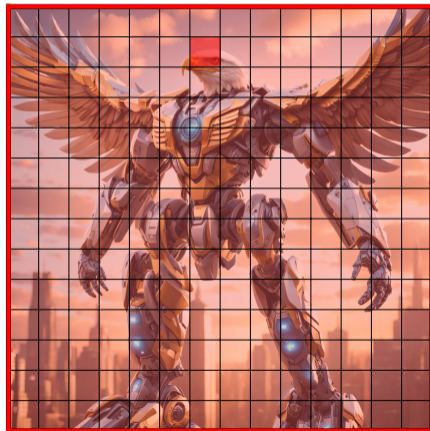
Self Attention

Any-to-any interaction between all n input tokens/pixels.

Attention weights, $\mathbf{P} \in \mathbb{R}^{n \times n}$, which gives us the notorious $\mathcal{O}(n^2)$ time (and space) complexity.

As n grows, This problem will be bounded by:

1. Memory capacity (assuming \mathbf{P} is stored in global memory)
2. Memory bandwidth (worst when \mathbf{P} is stored in global memory)



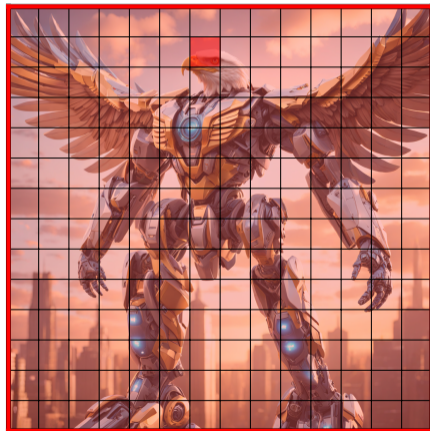
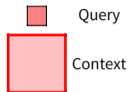
Self Attention

Any-to-any interaction between all n input tokens/pixels.

Attention weights, $\mathbf{P} \in \mathbb{R}^{n \times n}$, which gives us the notorious $\mathcal{O}(n^2)$ time (and space) complexity.

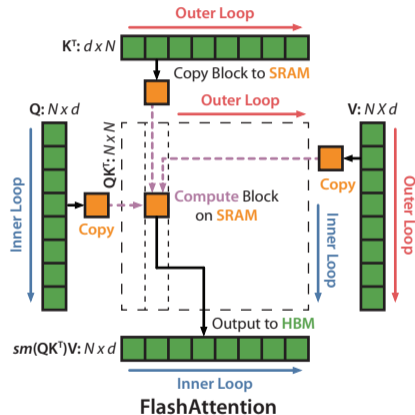
As n grows, This problem will be bounded by:

1. Memory capacity (assuming \mathbf{P} is stored in global memory)
2. Memory bandwidth (worst when \mathbf{P} is stored in global memory)
3. Computational power



Solution: Fused Attention

What if we don't store attention weights, P , in global memory in the first place?



Source: Flash Attention [1].

Background: fused multi-headed attention (FMHA)

In 2021, NVIDIA prototyped a fused attention kernel (FMHA) for GPT inference.

Background: fused multi-headed attention (FMHA)

In 2021, NVIDIA prototyped a fused attention kernel (FMHA) for GPT inference.

Key limitation: softmax reduction blocks second matrix multiply.

Background: fused multi-headed attention (FMHA)

In 2021, NVIDIA prototyped a fused attention kernel (FMHA) for GPT inference.

Key limitation: softmax reduction blocks second matrix multiply.

Online softmax [2] showed how softmax can be computed partially, and reduced into exact full softmax.

Background: fused multi-headed attention (FMHA)

In 2021, NVIDIA prototyped a fused attention kernel (FMHA) for GPT inference.

Key limitation: softmax reduction blocks second matrix multiply.

Online softmax [2] showed how softmax can be computed partially, and reduced into exact full softmax.

This was a major achievement, opening the door for distributed and highly parallel softmax implementations.

Background: fused multi-headed attention (FMHA)

In 2021, NVIDIA prototyped a fused attention kernel (FMHA) for GPT inference.

Key limitation: softmax reduction blocks second matrix multiply.

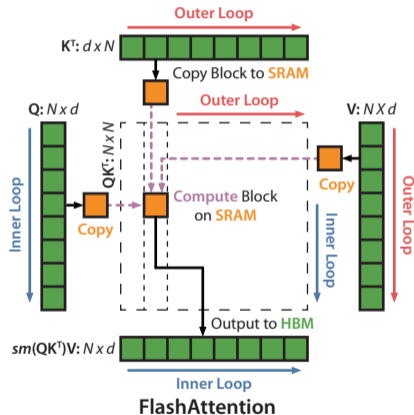
Online softmax [2] showed how softmax can be computed partially, and reduced into exact full softmax.

This was a major achievement, opening the door for distributed and highly parallel softmax implementations.

In 2022, and inspired by the Apex FMHA, Dao et al. [1] proposed Flash Attention; a fused attention kernel that accelerated existing BMM-style attention implementations, a key component of which was online softmax.

Background: fused multi-headed attention (FMHA)

Fused attention can fix two key bottlenecks at the same time: attention is no longer bound by memory bandwidth or memory capacity at scale.

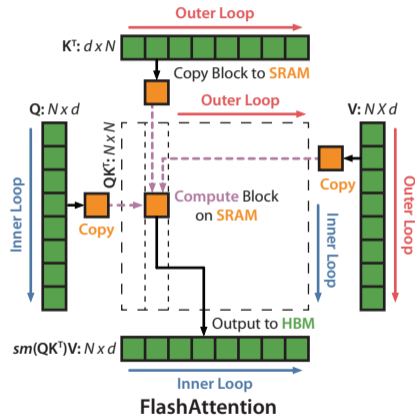


Source: Flash Attention [1].

Background: fused multi-headed attention (FMHA)

Fused attention can fix two key bottlenecks at the same time: attention is no longer bound by memory bandwidth or memory capacity at scale.

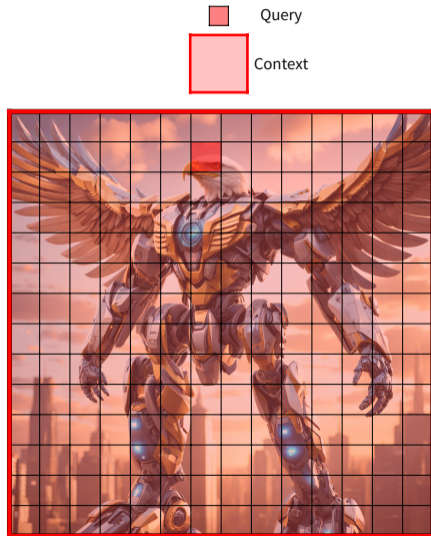
Fused attention kernels operating with FP8 precision have already exceeded the petaFLOP/s threshold on Hopper, and can achieve up to 60% of peak FLOP/s [3, 4].



Source: Flash Attention [1].

Background: Local attention

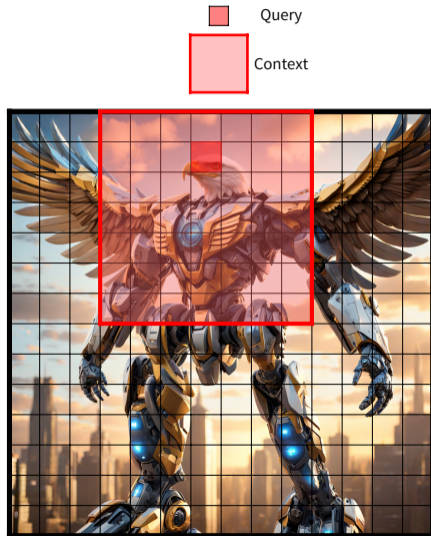
Parallel to fused attention, some worked on sparse/restricted self-attention patterns.



Background: Local attention

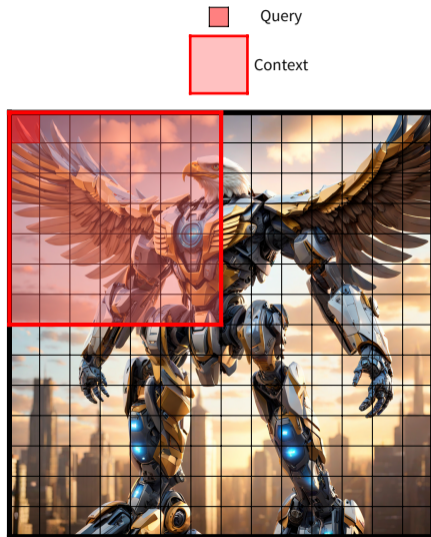
Parallel to fused attention, some worked on sparse/restricted self-attention patterns.

Among them was Neighborhood Attention [5, 6].



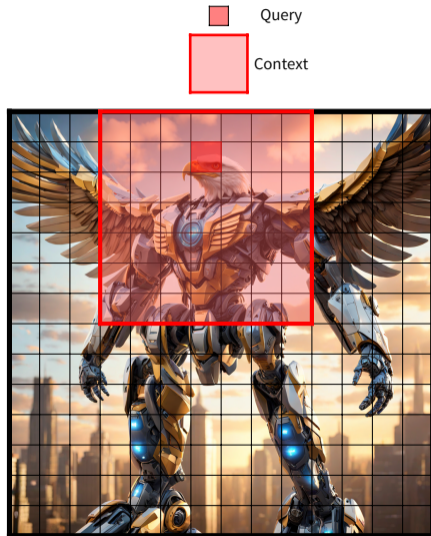
Background: Neighborhood attention

Every token/pixel attends to only those that are within a local neighborhood,



Background: Neighborhood attention

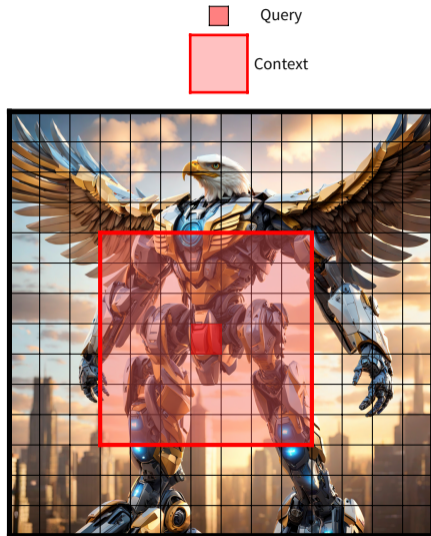
Every token/pixel attends to only those that are within a local neighborhood, which results in a linear complexity of $\mathcal{O}(nl)$, where ℓ is the neighborhood size.



Background: Neighborhood attention

Every token/pixel attends to only those that are within a local neighborhood, which results in a linear complexity of $\mathcal{O}(nl)$, where l is the neighborhood size.

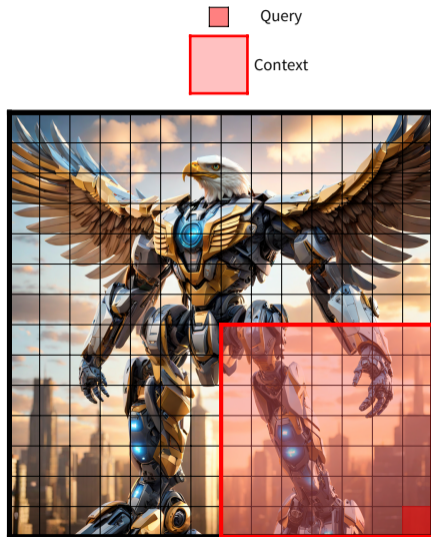
In many cases, this results in a sliding-window-like effect, similar to that in convolutional layers.



Background: Neighborhood attention

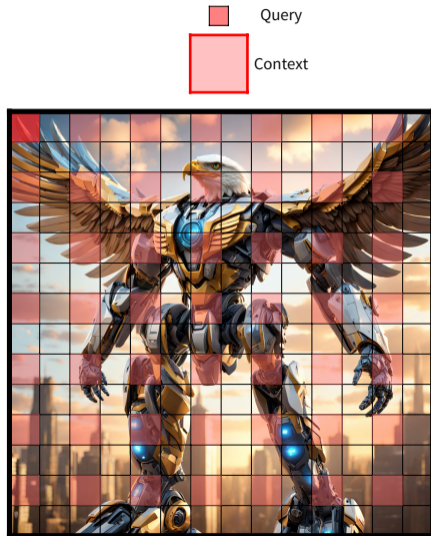
Every token/pixel attends to only those that are within a local neighborhood, which results in a linear complexity of $\mathcal{O}(nl)$, where l is the neighborhood size.

In many cases, this results in a sliding-window-like effect, similar to that in convolutional layers.



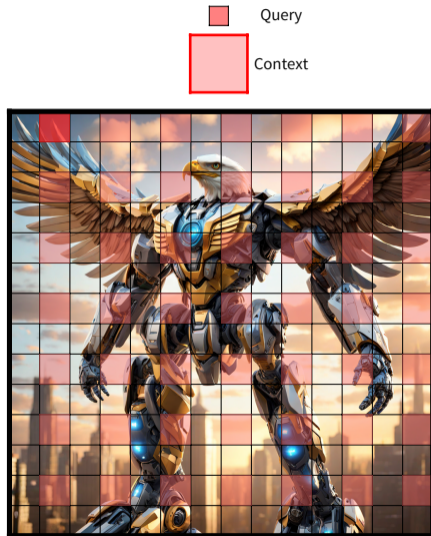
Background: Neighborhood attention

Neighborhood attention additionally can be dilated, similar to convolution.



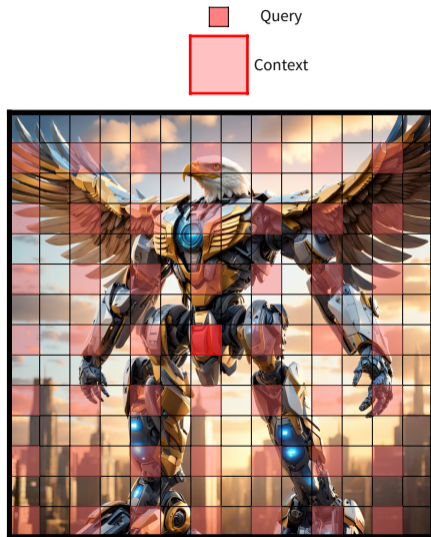
Background: Neighborhood attention

Neighborhood attention additionally can be dilated, similar to convolution.



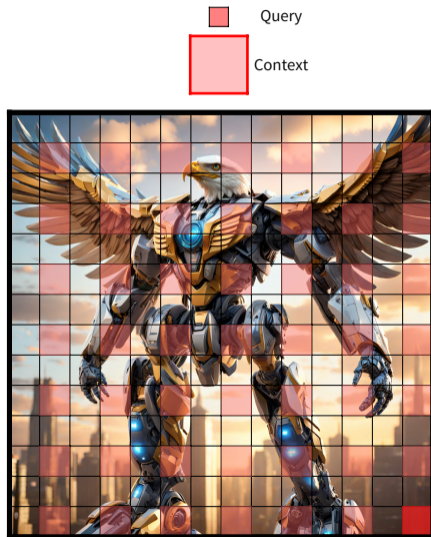
Background: Neighborhood attention

Neighborhood attention additionally can be dilated, similar to convolution.

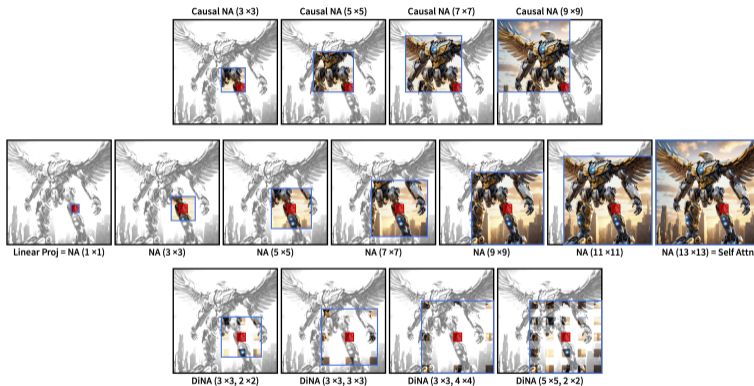


Background: Neighborhood attention

Neighborhood attention additionally can be dilated, similar to convolution.



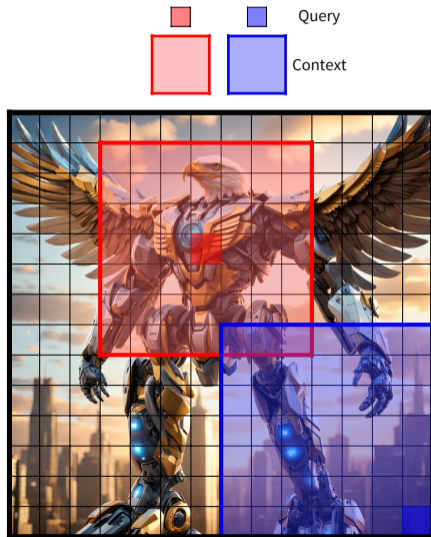
Background: Neighborhood attention



This creates a spectrum of possible attention patterns between linear projection and self attention.

Background: Neighborhood attention

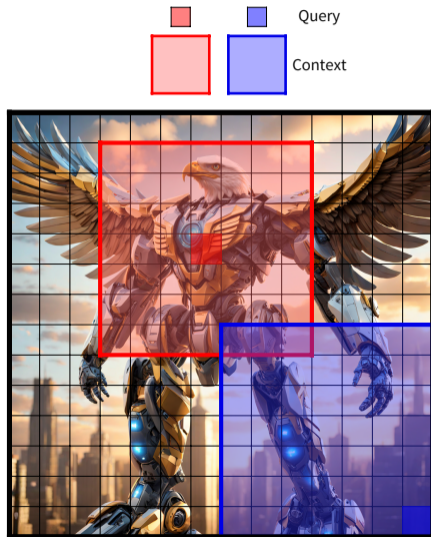
Neighborhood attention is a general matrix-vector multiplication (GEMV) problem.



Background: Neighborhood attention

Neighborhood attention is a general matrix-vector multiplication (GEMV) problem.

This is because context windows between different tokens are rarely identical.

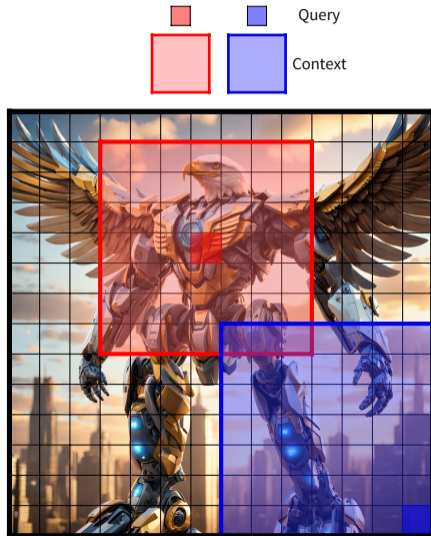


Background: Neighborhood attention

Neighborhood attention is a general matrix-vector multiplication (GEMV) problem.

This is because context windows between different tokens are rarely identical.

GEMVs are usually bound by memory bandwidth, cannot target Tensor Cores.



Motivation

$\mathcal{N}ATTEN$, neighborhood attention extension, only offers the naive kernels.

$\mathcal{N}ATTEN$, neighborhood attention extension, only offers the naive kernels.

1. Naive kernels are not performance-optimized, can't target Tensor Cores.

$\mathcal{N}ATTEN$, neighborhood attention extension, only offers the naive kernels.

1. Naive kernels are not performance-optimized, can't target Tensor Cores.
2. Fused attention much better than unfused kernels in most cases.

Attention is back-to-back GEMMs

GEMM: General Matrix-Matrix Multiplication

Consider single-batch single-head self-attention, $\mathbf{O} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, in which three operands $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{n \times d}$, and $\mathbf{V} \in \mathbb{R}^{n \times d'}$ are mapped to output $\mathbf{O} \in \mathbb{R}^{n \times d'}$ as follows:

Attention is back-to-back GEMMs

GEMM: General Matrix-Matrix Multiplication

Consider single-batch single-head self-attention, $\mathbf{O} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, in which three operands $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{n \times d}$, and $\mathbf{V} \in \mathbb{R}^{n \times d'}$ are mapped to output $\mathbf{O} \in \mathbb{R}^{n \times d'}$ as follows:

$$O_{ij} = \sum_{k=1}^n \text{softmax}_{k=1}^n \left(\sum_{l=1}^d Q_{il} K_{kl} \right) V_{kj} \quad (1)$$

Attention is back-to-back GEMMs

GEMM: General Matrix-Matrix Multiplication

Consider single-batch single-head self-attention, $\mathbf{O} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, in which three operands $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{n \times d}$, and $\mathbf{V} \in \mathbb{R}^{n \times d'}$ are mapped to output $\mathbf{O} \in \mathbb{R}^{n \times d'}$ as follows:

$$\mathbf{O}_{ij} = \sum_{k=1}^n \text{softmax}_{k=1}^n \left(\underbrace{\sum_{l=1}^d \mathbf{Q}_{il} \mathbf{K}_{kl}}_{\text{matmul}} \right) \mathbf{V}_{kj} \quad (1)$$

Attention is back-to-back GEMMs

GEMM: General Matrix-Matrix Multiplication

Consider single-batch single-head self-attention, $\mathbf{O} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, in which three operands $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{n \times d}$, and $\mathbf{V} \in \mathbb{R}^{n \times d'}$ are mapped to output $\mathbf{O} \in \mathbb{R}^{n \times d'}$ as follows:

$$\mathbf{O}_{ij} = \underbrace{\sum_{k=1}^n \underbrace{\text{softmax}_{k=1}^n \left(\underbrace{\sum_{l=1}^d \mathbf{Q}_{il} \mathbf{K}_{kl}}_{\text{matmul}} \right)}_{\text{matmul}} \mathbf{V}_{kj}}_{\text{matmul}} \quad (1)$$

Attention is back-to-back GEMMs

GEMM: General Matrix-Matrix Multiplication

Consider single-batch single-head self-attention, $\mathbf{O} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, in which three operands $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{n \times d}$, and $\mathbf{V} \in \mathbb{R}^{n \times d'}$ are mapped to output $\mathbf{O} \in \mathbb{R}^{n \times d'}$ as follows:

$$\mathbf{O}_{ij} = \underbrace{\sum_{k=1}^n \underbrace{\text{softmax}_{k=1}^n \left(\underbrace{\sum_{l=1}^d \mathbf{Q}_{il} \mathbf{K}_{kl}}_{\text{matmul}} \right)}_{\text{matmul}} \mathbf{V}_{kj}}_{\text{matmul}} \quad (1)$$

All 4 operands are matrices, and their modes are summarized below:

Attention is back-to-back GEMMs

GEMM: General Matrix-Matrix Multiplication

Consider single-batch single-head self-attention, $O = \text{Attention}(Q, K, V)$, in which three operands $Q, K \in \mathbb{R}^{n \times d}$, and $V \in \mathbb{R}^{n \times d'}$ are mapped to output $O \in \mathbb{R}^{n \times d'}$ as follows:

$$O_{ij} = \underbrace{\sum_{k=1}^n \underbrace{\text{softmax}_{k=1}^n \left(\underbrace{\sum_{l=1}^d Q_{il} K_{kl}}_{\text{matmul}} \right)}_{\text{matmul}} V_{kj}}_{\text{matmul}} \quad (1)$$

All 4 operands are matrices, and their modes are summarized below:

1. Attention mode (n): represents sequence,

Attention is back-to-back GEMMs

GEMM: General Matrix-Matrix Multiplication

Consider single-batch single-head self-attention, $\mathbf{O} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, in which three operands $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{n \times d}$, and $\mathbf{V} \in \mathbb{R}^{n \times d'}$ are mapped to output $\mathbf{O} \in \mathbb{R}^{n \times d'}$ as follows:

$$\mathbf{O}_{ij} = \underbrace{\sum_{k=1}^n \underbrace{\text{softmax}_{k=1}^n \left(\underbrace{\sum_{l=1}^d \mathbf{Q}_{il} \mathbf{K}_{kl}}_{\text{matmul}} \right)}_{\text{matmul}} \mathbf{V}_{kj}}_{\text{matmul}} \quad (1)$$

All 4 operands are matrices, and their modes are summarized below:

1. Attention mode (n): represents sequence,
2. Dimension mode (d and d'): head dimension.

Neighborhood Attention is a GEMV problem

Neighborhood attention on the other hand is a matrix-vector multiplication problem:

Neighborhood Attention is a GEMV problem

Neighborhood attention on the other hand is a matrix-vector multiplication problem:

$$\mathbf{O}_{ij} = \sum_{k \in \rho^\kappa(i)} \text{softmax}_{k \in \rho^\kappa(i)} \left(\sum_{l=1}^d \mathbf{Q}_{il} \mathbf{K}_{kl} \right) \mathbf{V}_{kj} \quad (2)$$

where $\rho^\kappa(i)$ is a set of all attention mode coordinates in $[1, n]$ that are neighbors of i , given window size κ .

Neighborhood Attention is a GEMV problem

Neighborhood attention on the other hand is a matrix-vector multiplication problem:

$$O_{ij} = \sum_{k \in \rho^\kappa(i)} \text{softmax}_{k \in \rho^\kappa(i)} \left(\sum_{l=1}^d Q_{il} K_{kl} \right) V_{kj} \quad (2)$$

where $\rho^\kappa(i)$ is a set of all attention mode coordinates in $[1, n]$ that are neighbors of i , given window size κ .

For any given $i, j \in [1, n], i \neq j$, we can't assume $\rho(i) = \rho(j)$. Therefore, the two matrix-matrix multiplications become matrix-vector multiplication problems.

Neighborhood attention as a GEMM

Neighborhood function (assuming no dilation, and window size κ), $\rho^\kappa(\cdot)$, has a special property:

Neighborhood attention as a GEMM

Neighborhood function (assuming no dilation, and window size κ), $\rho^\kappa(\cdot)$, has a special property:

$$\forall i \in [1, n], \quad |\rho^\kappa(i) \cap \rho^\kappa(i+1)| \in \{\kappa, \kappa - 1\}$$

Neighborhood attention as a GEMM

Neighborhood function (assuming no dilation, and window size κ), $\rho^\kappa(\cdot)$, has a special property:

$$\forall i \in [1, n], \quad |\rho^\kappa(i) \cap \rho^\kappa(i+1)| \in \{\kappa, \kappa - 1\}$$

In other words, query tokens close to each other overlap greatly in their neighborhoods.

Neighborhood attention as a GEMM

Neighborhood function (assuming no dilation, and window size κ), $\rho^\kappa(\cdot)$, has a special property:

$$\forall i \in [1, n], \quad |\rho^\kappa(i) \cap \rho^\kappa(i+1)| \in \{\kappa, \kappa - 1\}$$

In other words, query tokens close to each other overlap greatly in their neighborhoods.

This means we can simply define $\rho^\kappa(\cdot)$ as an implicit attention mask fused into the implementation.

Neighborhood attention as a GEMM

Neighborhood function (assuming no dilation, and window size κ), $\rho^\kappa(\cdot)$, has a special property:

$$\forall i \in [1, n], \quad |\rho^\kappa(i) \cap \rho^\kappa(i+1)| \in \{\kappa, \kappa - 1\}$$

In other words, query tokens close to each other overlap greatly in their neighborhoods.

This means we can simply define $\rho^\kappa(\cdot)$ as an implicit attention mask fused into the implementation.

GEMMs on massively parallel hardware like GPUs are computed in tiles. Tiles that only produce masked attention weights can be skipped entirely, saving compute!

Neighborhood attention as a GEMM

Neighborhood function (assuming no dilation, and window size κ), $\rho^\kappa(\cdot)$, has a special property:

$$\forall i \in [1, n], \quad |\rho^\kappa(i) \cap \rho^\kappa(i+1)| \in \{\kappa, \kappa - 1\}$$

In other words, query tokens close to each other overlap greatly in their neighborhoods.

This means we can simply define $\rho^\kappa(\cdot)$ as an implicit attention mask fused into the implementation.

GEMMs on massively parallel hardware like GPUs are computed in tiles. Tiles that only produce masked attention weights can be skipped entirely, saving compute!

(This is how Mistral's sliding window attention was implemented [7].)

But what about 2-D and 3-D?

Neighborhood attention was originally focused on vision and not language, and therefore has typically focused on 2-D and 3-D spaces.

But what about 2-D and 3-D?

Neighborhood attention was originally focused on vision and not language, and therefore has typically focused on 2-D and 3-D spaces.

In 2-D and 3-D spaces, masking based on coordinates is still possible, but the sparse computation pattern won't save nearly as much computation without redefining attention for 2-D and 3-D spaces.

2-D Attention is back-to-back GETTs!

GETT: General Tensor-Tensor Contraction

Consider 2-D self-attention, $\mathbf{O} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, in which three operands $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{h \times w \times d}$, and $\mathbf{V} \in \mathbb{R}^{h \times w \times d'}$ are mapped to output $\mathbf{O} \in \mathbb{R}^{h \times w \times d'}$.

2-D Attention is back-to-back GETTs!

GETT: General Tensor-Tensor Contraction

Consider 2-D self-attention, $\mathbf{O} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, in which three operands $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{h \times w \times d}$, and $\mathbf{V} \in \mathbb{R}^{h \times w \times d'}$ are mapped to output $\mathbf{O} \in \mathbb{R}^{h \times w \times d'}$.

All 4 operands are now rank-3 **tensors**, and their modes are summarized below:

2-D Attention is back-to-back GETTs!

GETT: General Tensor-Tensor Contraction

Consider 2-D self-attention, $\mathbf{O} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, in which three operands $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{h \times w \times d}$, and $\mathbf{V} \in \mathbb{R}^{h \times w \times d'}$ are mapped to output $\mathbf{O} \in \mathbb{R}^{h \times w \times d'}$.

All 4 operands are now rank-3 **tensors**, and their modes are summarized below:

1. Attention mode ($h \times w$): now assumes a 2-D layout,

2-D Attention is back-to-back GETTs!

GETT: General Tensor-Tensor Contraction

Consider 2-D self-attention, $\mathbf{O} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, in which three operands $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{h \times w \times d}$, and $\mathbf{V} \in \mathbb{R}^{h \times w \times d'}$ are mapped to output $\mathbf{O} \in \mathbb{R}^{h \times w \times d'}$.

All 4 operands are now rank-3 **tensors**, and their modes are summarized below:

1. Attention mode ($h \times w$): now assumes a 2-D layout,
2. Dimension mode (d and d'): unchanged.

2-D Attention is back-to-back GETTs!

GETT: General Tensor-Tensor Contraction

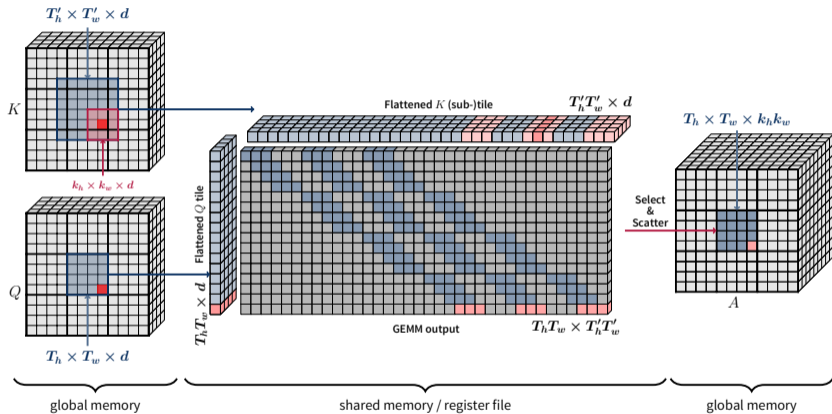
Consider 2-D self-attention, $\mathbf{O} = \text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})$, in which three operands $\mathbf{Q}, \mathbf{K} \in \mathbb{R}^{h \times w \times d}$, and $\mathbf{V} \in \mathbb{R}^{h \times w \times d'}$ are mapped to output $\mathbf{O} \in \mathbb{R}^{h \times w \times d'}$.

All 4 operands are now rank-3 **tensors**, and their modes are summarized below:

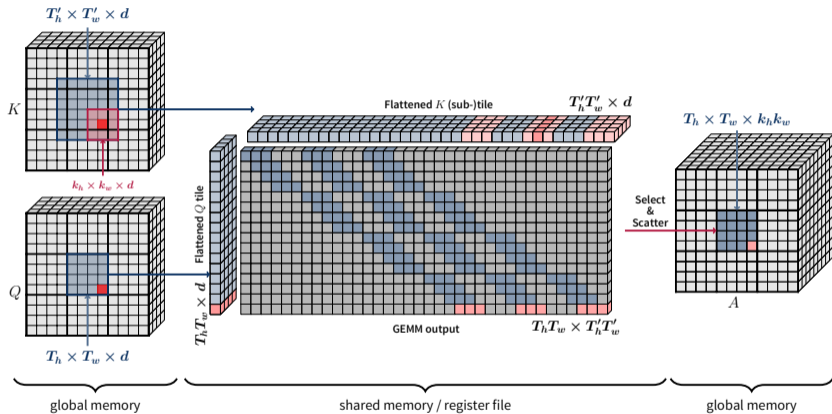
1. Attention mode ($h \times w$): now assumes a 2-D layout,
2. Dimension mode (d and d'): unchanged.

Note: this is unnecessary if you're only doing bi-directional self-attention. No masking, No compute to save.

Neighborhood attention as a GEMM/GETT

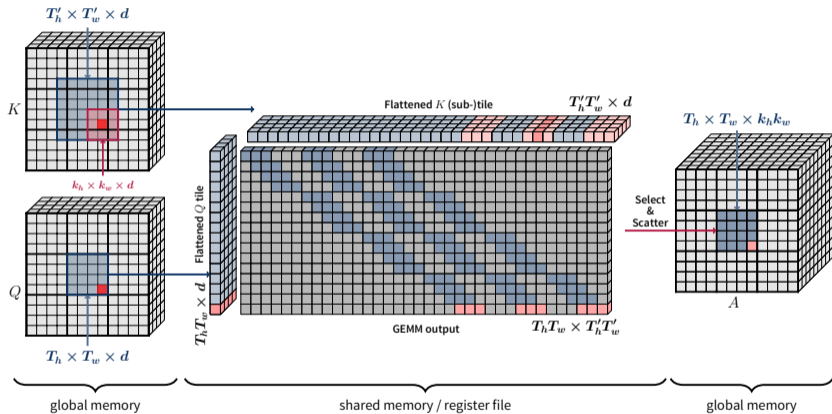


Neighborhood attention as a GEMM/GETT



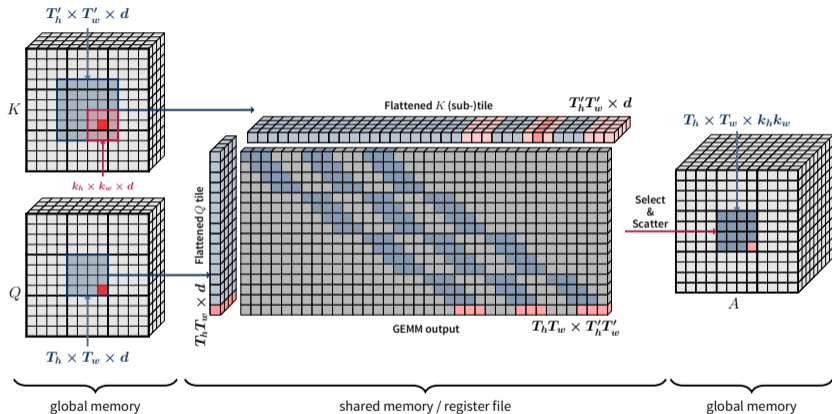
1. Modify tiling and predication to “convert” GEMM into GETT and maximize sparsity,

Neighborhood attention as a GEMM/GETT



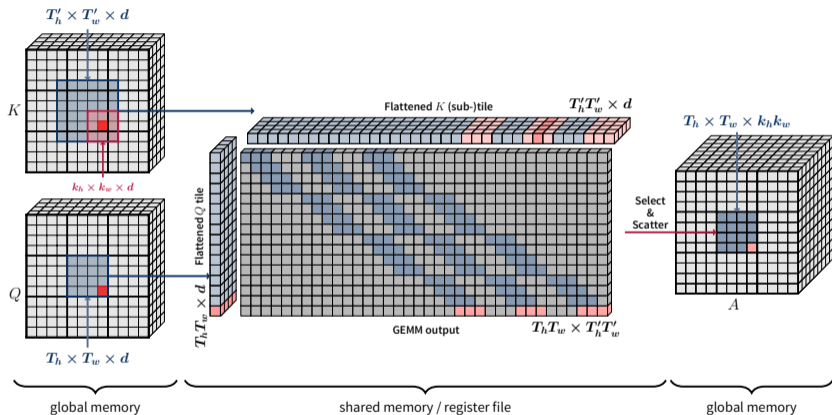
1. Modify tiling and predication to “convert” GEMM into GETT and maximize sparsity,
2. Fuse neighborhood attention masking as a scatter/gather operation.

Neighborhood attention as a GEMM/GETT



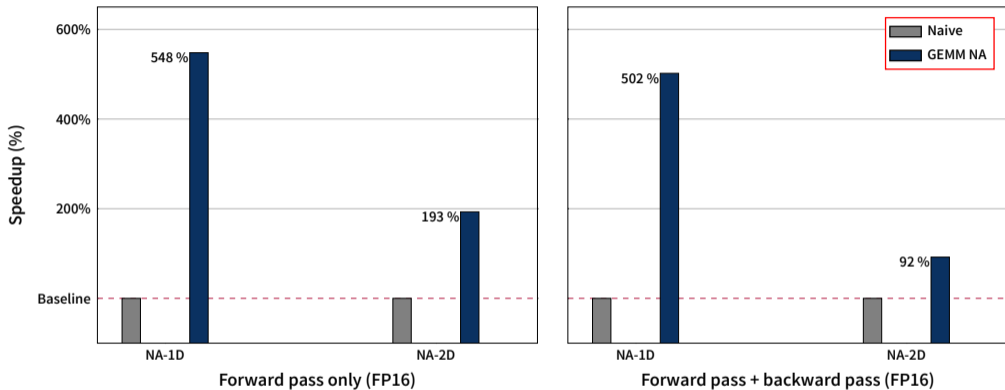
1. Modify tiling and predication to “convert” GEMM into GETT and maximize sparsity, *software predication is expensive!*

Neighborhood attention as a GEMM/GETT

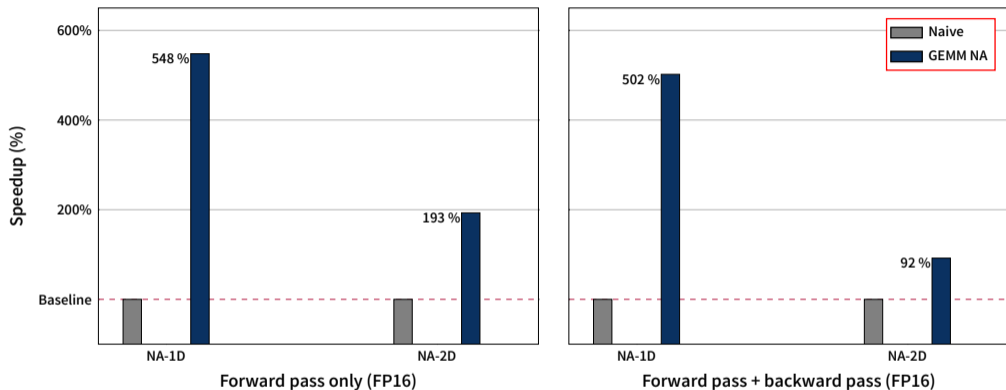


2. Fuse neighborhood attention masking as a scatter/gather operation.
breaks GEMM pipelining in lower precision

GEMM-based neighborhood attention performance

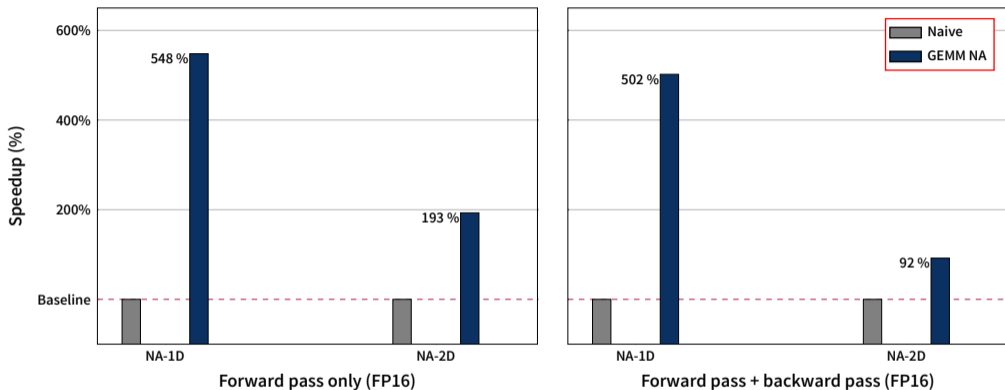


GEMM-based neighborhood attention performance



1-D requires no change to predication, has simpler scatter/gather logic.

GEMM-based neighborhood attention performance



1-D requires no change to predication, has simpler scatter/gather logic.

2-D suffers from predication and scatter/gather logic.

Fused neighborhood attention?

Other than being bound by memory bandwidth, all unfused neighborhood attention kernels are greatly limited by the scatter/gather operation.

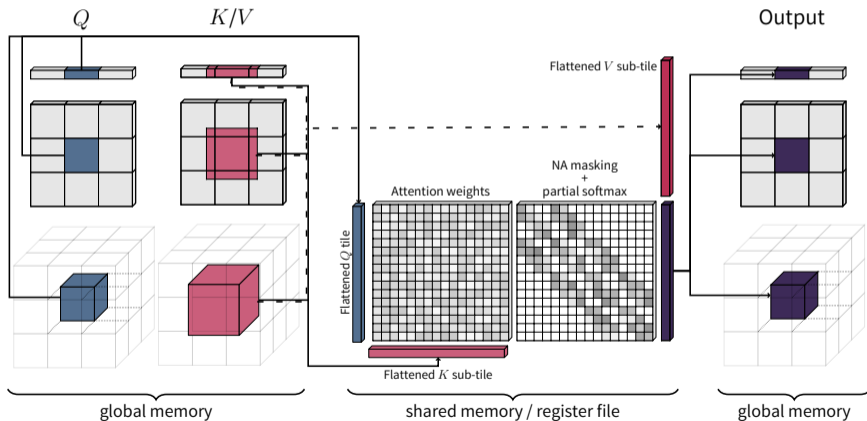
Fused neighborhood attention?

Other than being bound by memory bandwidth, all unfused neighborhood attention kernels are greatly limited by the scatter/gather operation.

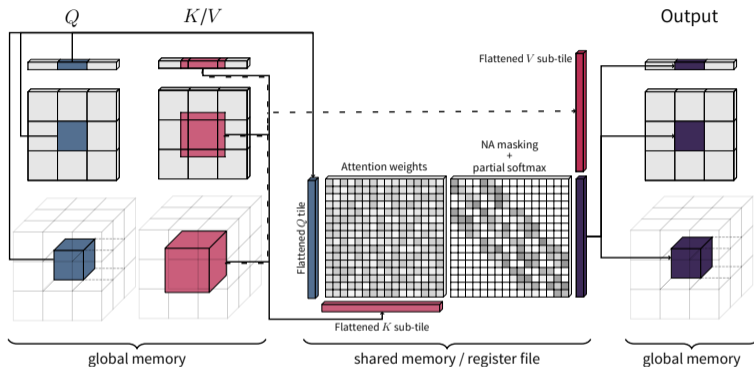
However, scatter/gather is not required if we keep attention weights in local memory!

Fused neighborhood attention

Fused back-to-back GETTs!

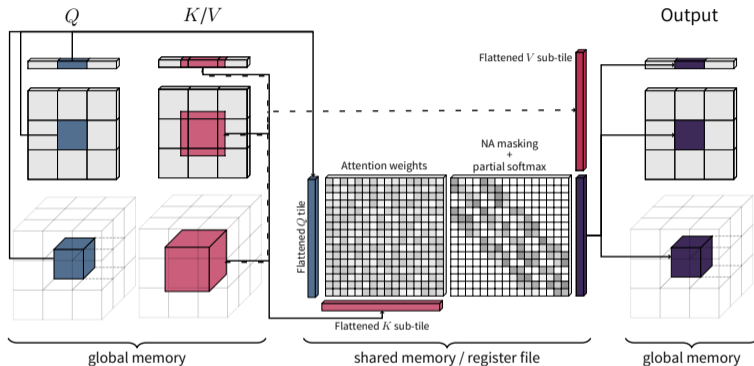


Fused neighborhood attention



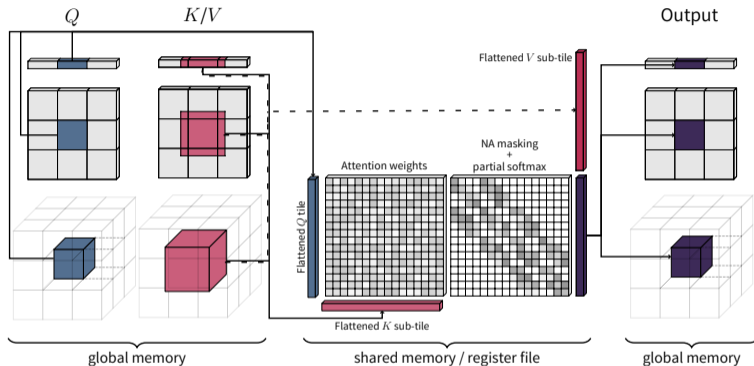
1. (Almost) constant global memory requirement (regardless of window size),

Fused neighborhood attention



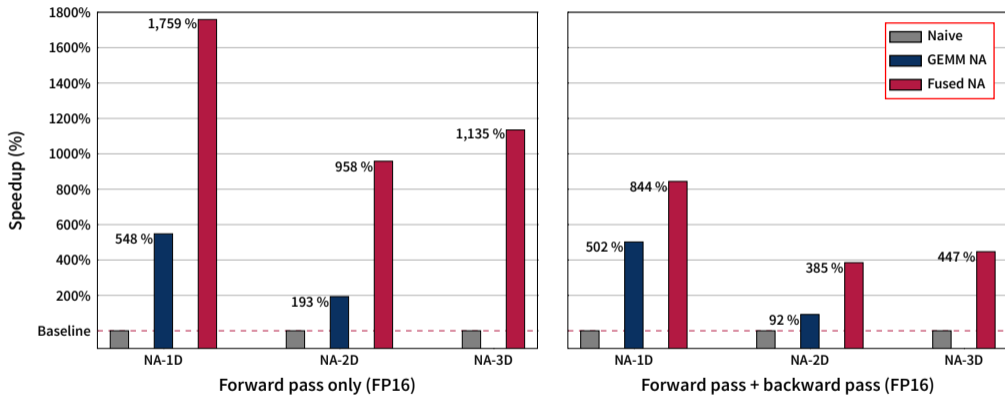
1. (Almost) constant global memory requirement (regardless of window size),
2. Scales up lower precision performance,

Fused neighborhood attention



1. (Almost) constant global memory requirement (regardless of window size),
2. Scales up lower precision performance,
3. Defines causal masking, and allows different parameters (window size, dilation, causality) *per dimension*.

Fused neighborhood attention: op-level performance



Relative performance improvement on Ampere (A100 PCIe).

Implementation

All of our implementations were done using NVIDIA's CUTLASS framework (2.X API).

Implementation

All of our implementations were done using NVIDIA's CUTLASS framework (2.X API).

Fused neighborhood attention was based on xFormers' FMHA kernel, and supports all NVIDIA architectures since Maxwell and up to and including Ampere.







Implementation

All of our implementations were done using NVIDIA's CUTLASS framework (2.X API).

Fused neighborhood attention was based on xFormers' FMHA kernel, and supports all NVIDIA architectures since Maxwell and up to and including Ampere.

All of them are already available through $\mathcal{N}ATTEN$, just `pip install natten` or refer to <https://shi-labs.com/natten>.

References I

-  T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, “Flashattention: Fast and memory-efficient exact attention with io-awareness,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.
-  M. Milakov and N. Gimelshein, “Online normalizer calculation for softmax,” *arXiv preprint arXiv:1805.02867*, 2018.
-  H. Blog, “Petaflops inference era: 1 pflops attention, and preliminary end-to-end results,” <https://medium.com/p/21f682cf2ed1>, 2024.
-  J. Shah, G. Bikshandi, Y. Zhang, V. Thakkar, P. Ramani, and T. Dao, “Flashattention-3: Fast and accurate attention with asynchrony and low-precision,” *arXiv preprint arXiv:2407.08608*, 2024.
-  A. Hassani, S. Walton, J. Li, S. Li, and H. Shi, “Neighborhood attention transformer,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.
-  A. Hassani and H. Shi, “Dilated neighborhood attention transformer,” *arXiv preprint arXiv:2209.15001*, 2022.

References II



A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. I. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier *et al.*, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.

Thank you for your *Attention*.