CS 498: Machine Learning System
Spring 2026

Minjia Zhang

The Grainger College of Engineering

Arithmetic Intensity

AI = #ops / #bytes

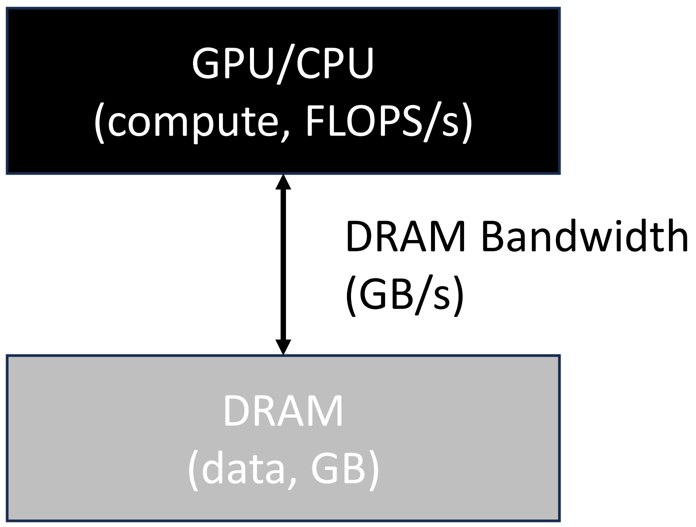
Arithmetic Intensity

AI= #ops / #bytes

Used to evaluate the efficiency of computational algorithms

System performance is bound by

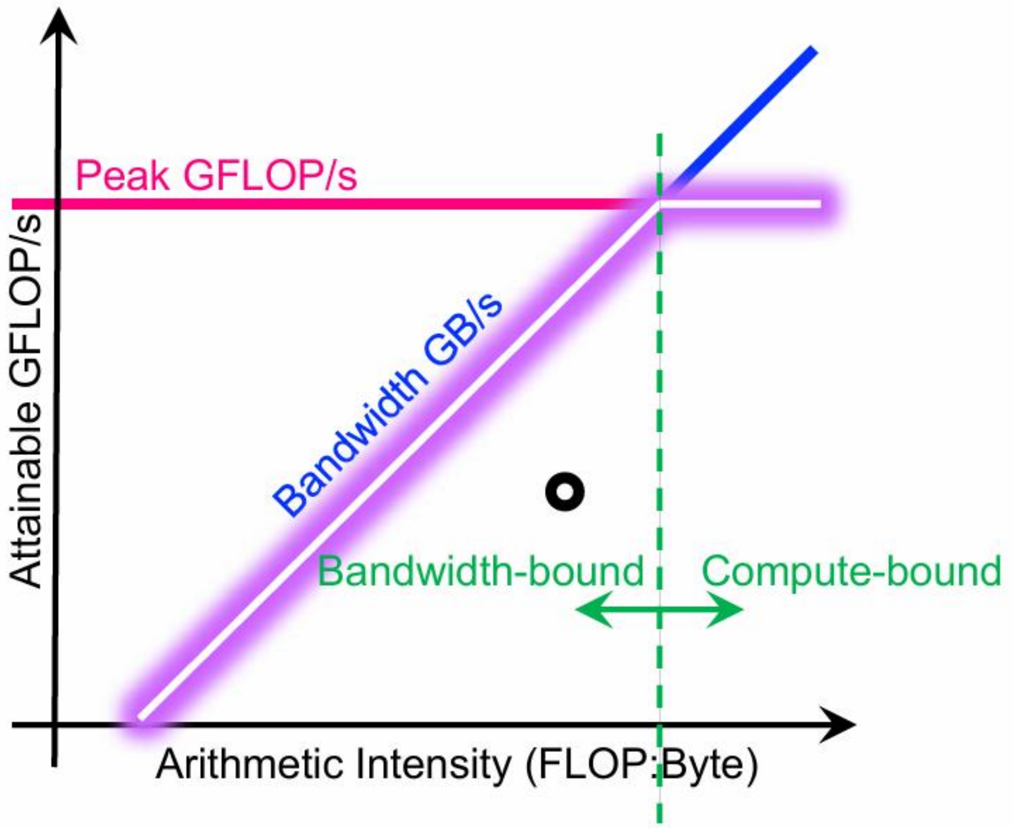
1) The peak compute TFLOPS
2) The memory bandwidth



Roofline Model

The Roofline model provides a relatively simple way for performance estimates based on the computation of workload and hardware characteristics

- High AI: Compute-bound
- Low AI: Memory bandwidth bound



Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures, 2008

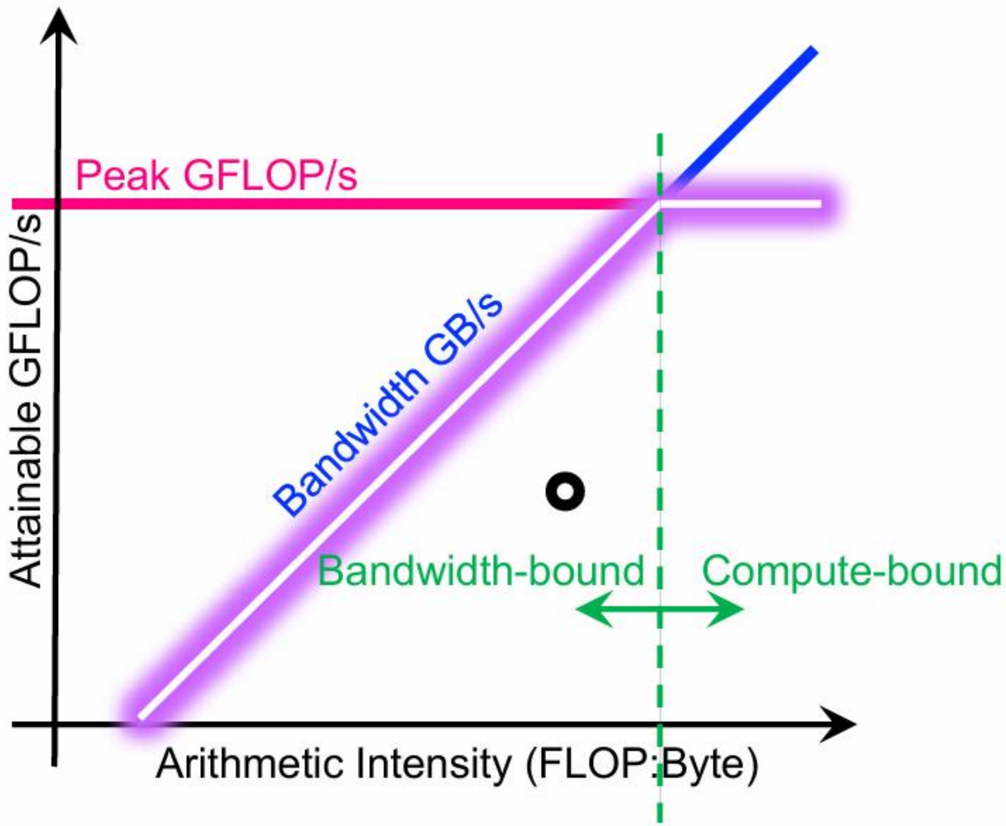
Why Roofline Model?

Helps identify the bottlenecks

Program performance depends on how well it fits the hardware architecture

Create optimizations to exhaust both compute and bandwidth at the same time (many times it is impossible)

The mode also tells you when to stop



Roofline: An Insightful Visual Performance Model for Floating-Point Programs and Multicore Architectures, 2008

Arithmetic Intensity: A Toy Example

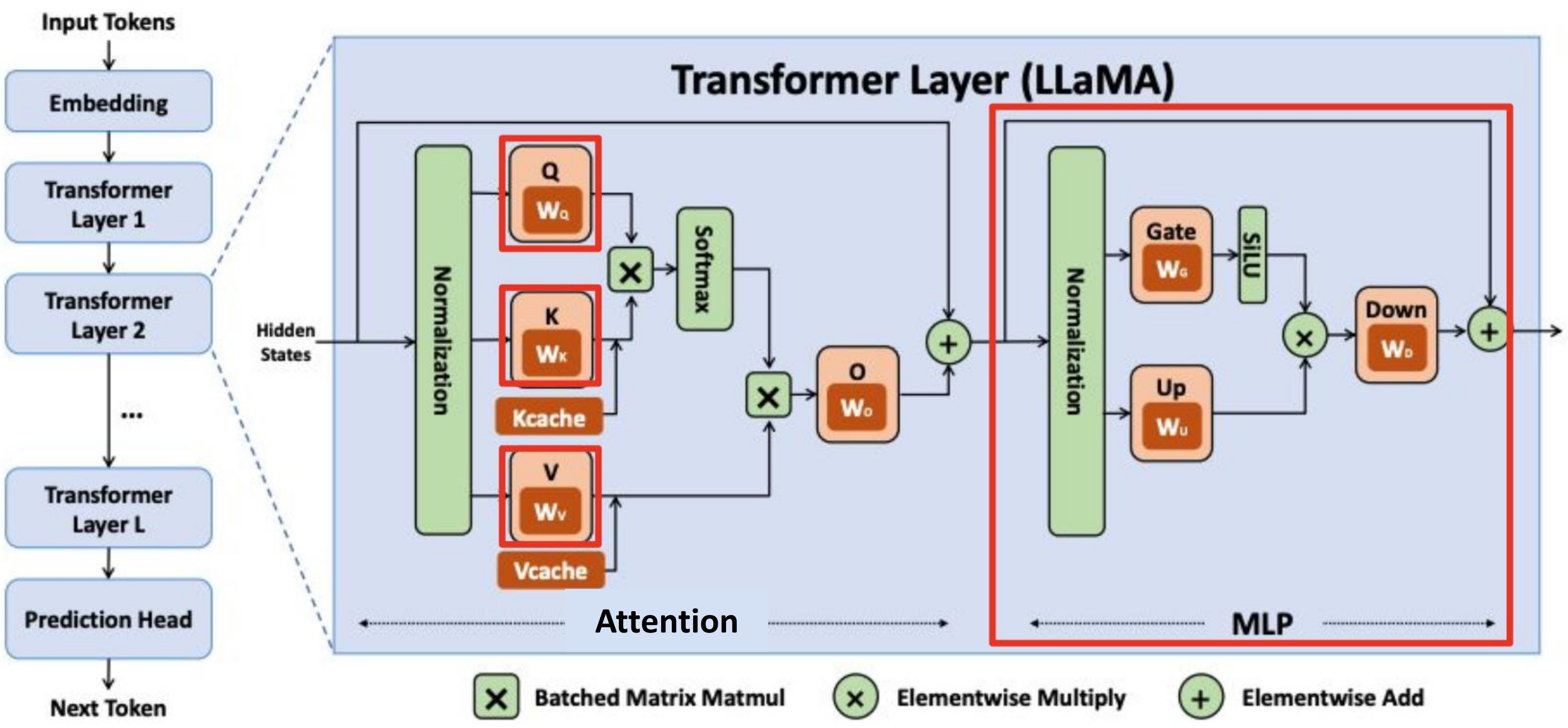
```
void add(int n, float* A, float* B, float* C){
   for (int i=0; i<n; i++)
      C[i] = A[i] + B[i];
}
```

Two loads, one store per math op

1. Read A[i]
2. Read B[i]
3. Add A[i] + B[i]
4. Store C[i]

Arithmetic intensity = 1/3

Arithmetic Intensity: Transformers



Arithmetic Intensity: Transformers (Attention)

Let us start with the attention block: $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.

2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \text{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.

3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.

4: Return $\mathbf{O}$.

Arithmetic Intensity: Transformers (Attention)

Let us start with the attention block: $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \text{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

Notations:

- h: Hidden dimension of QKV (often 4096)
- N: Input length (e.g., 4096 tokens)

Compute Flops:

- Step 1: Q matrix $_{(N * h)}$ x K^T matrix $_{(h * N)}$

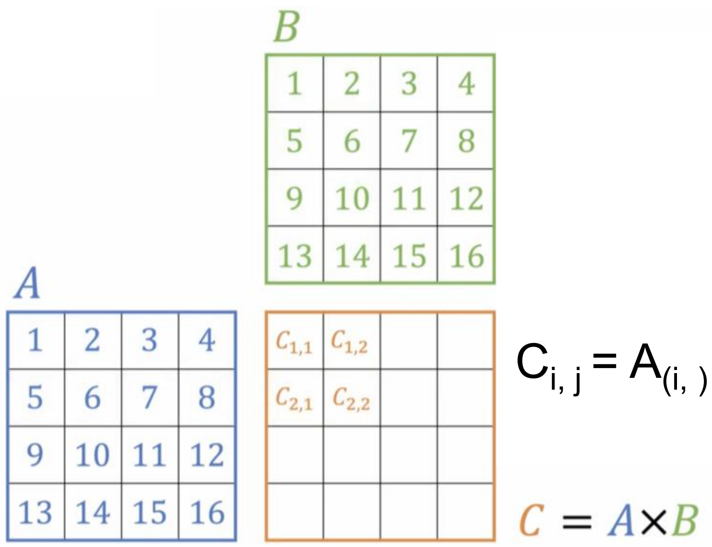
Arithmetic Intensity: Transformers (Attention)

Let us start with the attention block: $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \text{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.



Compute Flops:

- Step 1: Q matrix $_{(N * h)}$ x K^T matrix $_{(h * N)}$

$C_{i,j} = A_{(i,\ )} \times B_{(,\ j)}$: $1 \times 1 + 2 \times 5 + 3 \times 9 + 4 \times 13$

h-element mul & add -> 2 * h Flops

Arithmetic Intensity: Transformers (Attention)

Let us start with the attention block: $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \text{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

Notations:

- h: Hidden dimension of QKV (often 4096)
- N: Input length (e.g., 4096 tokens)

Compute Flops:

- Step 1: Q matrix $_{(N * h)}$ x K^T matrix $_{(h * N)}$

Flops: 2 * h * N * N

Arithmetic Intensity: Transformers (Attention)

Let us start with the attention block: $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{QK}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \text{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{PV}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

Notations:

- h: Hidden dimension of QKV (often 4096)
- N: Input length (e.g., 4096 tokens)

Compute Flops:

- $\underbrace{2 * h * N * N}_{\text{Step 1 (matrix mul)}} + \underbrace{3 * N * N}_{\text{Step 2 (softmax)}} + \underbrace{2 * N * N * h}_{\text{Step 3}}$

Arithmetic Intensity: Transformers (Attention)

Let us start with the attention block: $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.

2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \text{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.

3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.

4: Return $\mathbf{O}$.

Notations:

- h: Hidden dimension of QKV (often 4096)
- N: Input length (e.g., 4096 tokens)

Compute Flops:

- 2 * h * N * N + 3 * N * N + 2 * N * N *h

Step 1 (matrix mul) Step 2 (softmax) Step 3

Loads/stores: (2*h*N + N*N) * 2 (2 bytes per element due to half precision)

Step 1

Arithmetic Intensity: Transformers (Attention)

Let us start with the attention block: $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \text{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

Notations:

- h: Hidden dimension of QKV (often 4096)
- N: Input length (e.g., 4096 tokens)

Compute Flops:

- 2 * h * N * N + 3 * N * N + 2 * N * N *h

Step 1 (matrix mul) Step 2 (softmax) Step 3

Loads/stores: 2*2*h*N + 2*N*N + 2*N*N + 2*N*N + 2(N*N + N*h) + 2*N*h

Step 1

Arithmetic Intensity: Transformers (Attention)

Let us start with the attention block: $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d}$ in HBM.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from HBM, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top$, write $\mathbf{S}$ to HBM.
2: Read $\mathbf{S}$ from HBM, compute $\mathbf{P} = \text{softmax}(\mathbf{S})$, write $\mathbf{P}$ to HBM.
3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from HBM, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to HBM.
4: Return $\mathbf{O}$.

Notations:

- h: Hidden dimension of QKV (often 4096)
- N: Input length (e.g., 4096 tokens)

Compute Flops:

- 2 * h * N * N + 3 * N * N + 2 * N * N *h

Step 1 (matrix mul) Step 2 (softmax) Step 3

Attention Compute: (4*h + 3) * N^2; Memory IO: 8 * N^2 + 8 * N * h

Arithmetic Intensity: Transformers (Attention)

Attention Compute: (4*h + 3) * N^2; Memory IO: 8 * N^2 + 8 * N * h

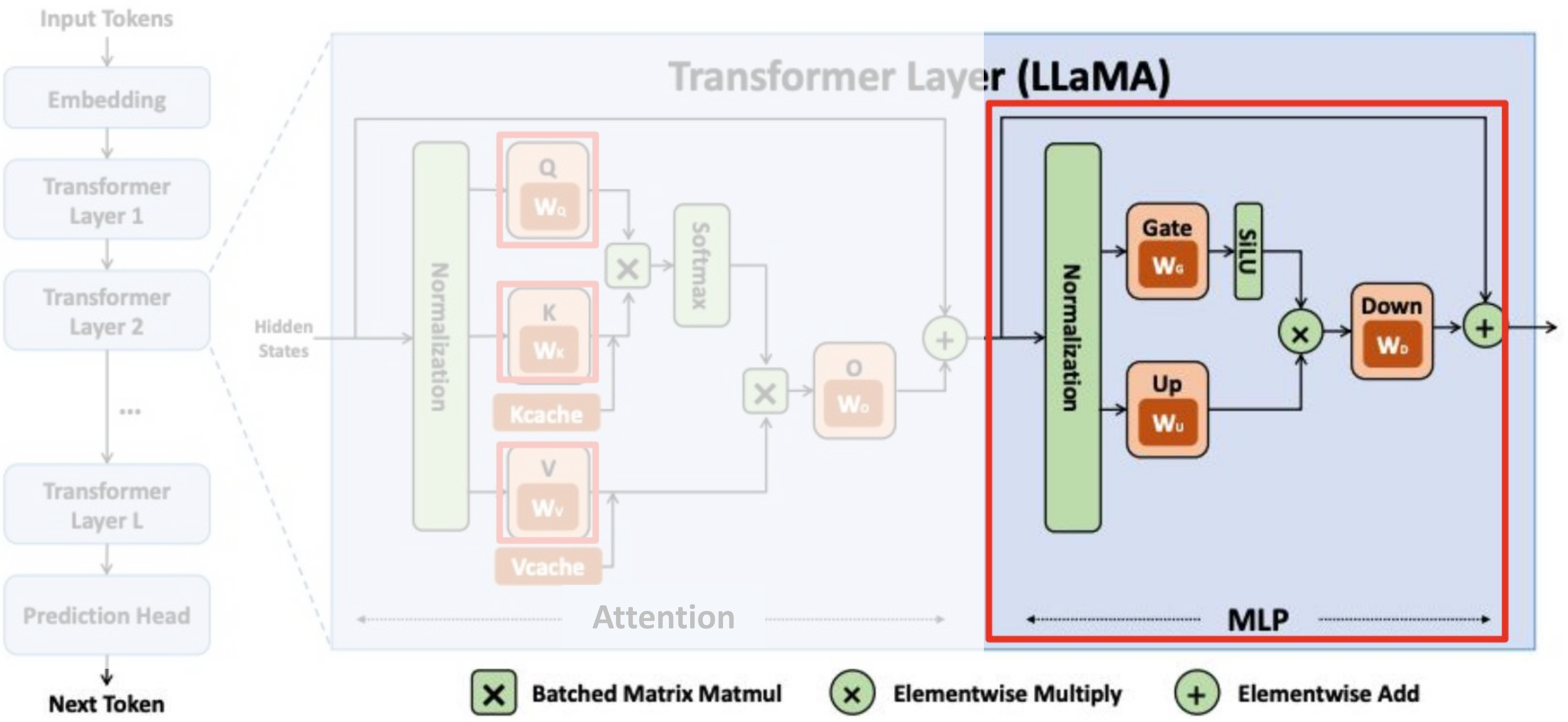
↓

Arithmetic Intensity = {(4*h + 3) * N^2}/{8 * N^2 + 8 * N * h}

↓

Arithmetic intensity increases as the input length N grows.

Arithmetic Intensity: Transformers (MLP)



Linear layer, essentially a GEMM: X * W

- Shape: X (N, K), W(K, M)

Compute: 2*N*K*M

Memory: 2*N*K + 2*K*M + 2*N*M

Arithmetic Intensity: Transformers

MLP FLOPs/Memory ops = 1365 ops/byte

- N = K = M = 4096

A10 GPU Analysis:

- Compute capability: 125TF, Memory bandwidth: 600GB/s
- Ops/byte = 125TF / 600GB/s = 208.3 ops/byte

Arithmetic Intensity: Transformers

MLP FLOPs/Memory ops = 1365 ops/byte

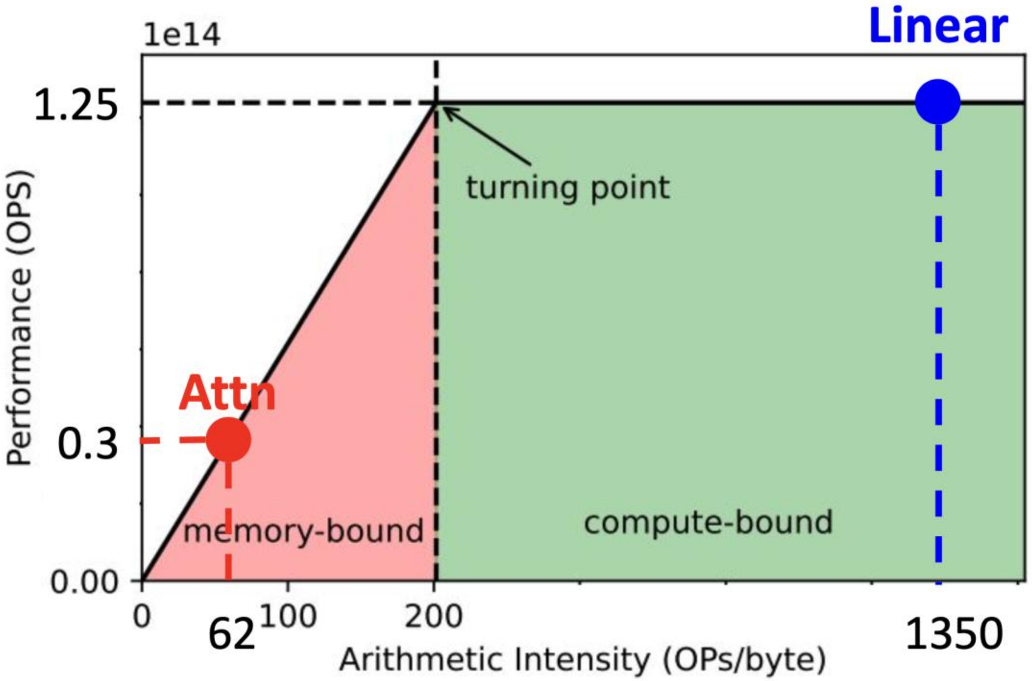
- N = K = M = 4096

A10 GPU Analysis:

- Compute capability: 125TF, Memory bandwic 600GB/s
- Ops/byte = 125TF / 600GB/s = 208.3 ops/byt

MLP's AI is much higher than ~200 ops/byt

- Compute bound

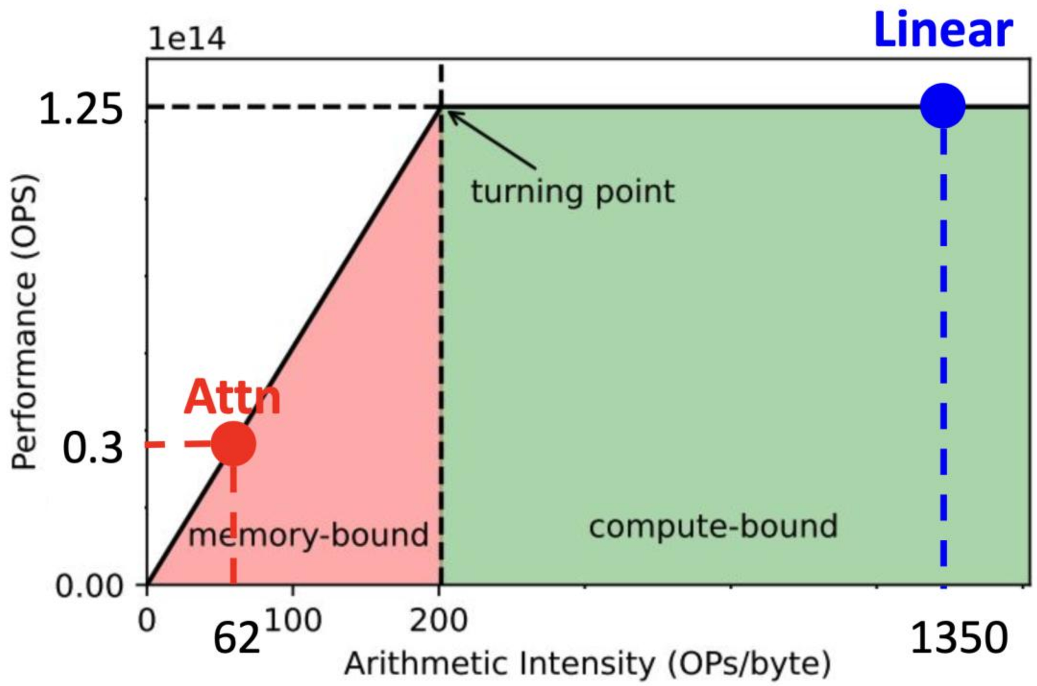


Short input length (N)

How to handle the discrepancy of these two transformer components to maximize efficiency?

Arithmetic Intensity: Transformers

Attention Compute: (4*h + 3) * N^2; Memory IO: 8 * N^2 + 8 * N * h

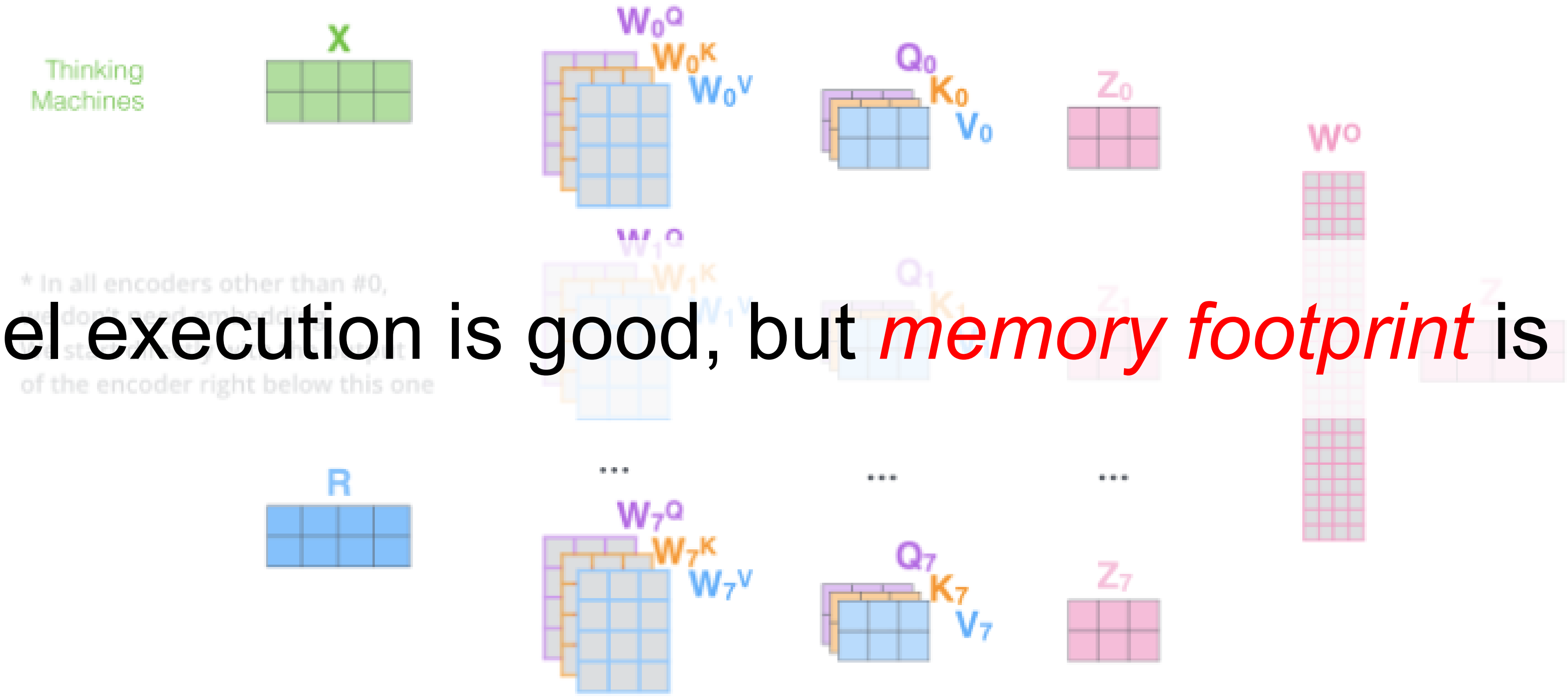


Attention can become compute-bound as N grows

How to speed up?

Multi-head Attention

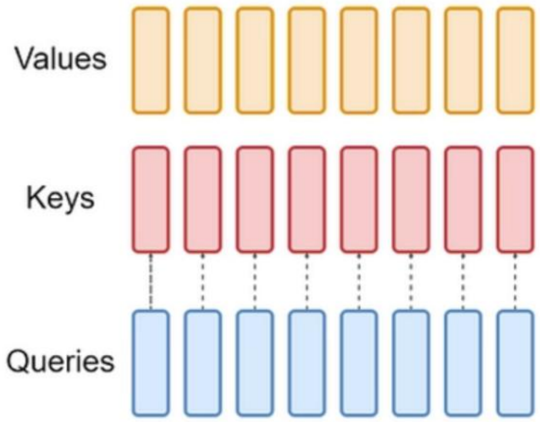
- Do many attention head calculation *in parallel, and combine*
- Each head has its *own* set of parameters
- Different heads can learn different “interactions” between inputs



Parallel execution is good, but *memory footprint* is still large

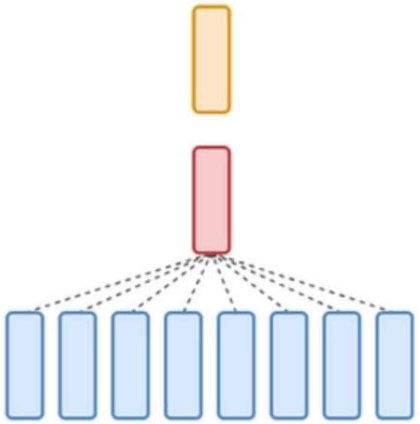
Multi-Query Attention

Multi-head Attention



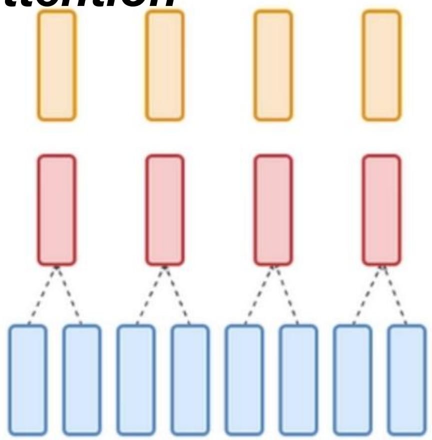
Each head digests QKV separately

Multi-Query Attention



Share single key and value heads across query heads

Grouped-Query Attention

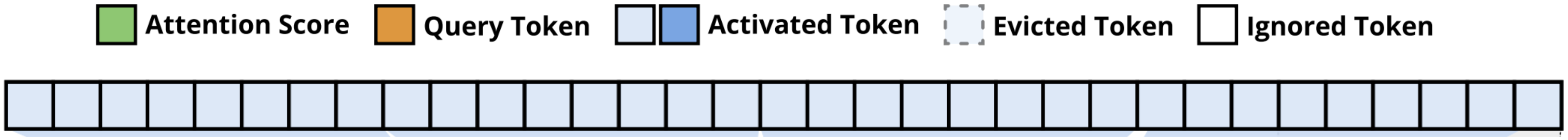


Share single key and value heads for each group of query heads

Better compute, smaller memory footprint, but quality may drop

Native Sparse Attention (ACL’25 from DeepSeek)

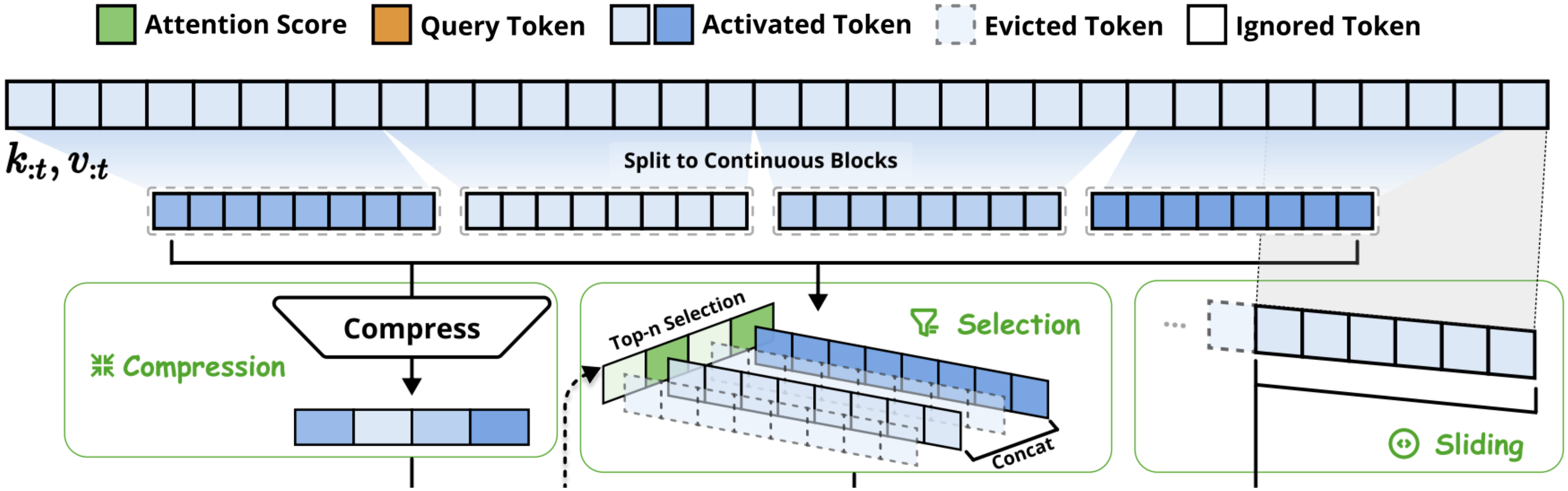
- Key insight: we can summarize long-context input, identify and focus on key words



[1] Native Sparse Attention: Hardware-Aligned and Natively Trainable Sparse Attention, ACL, 2025

Native Sparse Attention (ACL'25 from DeepSeek)

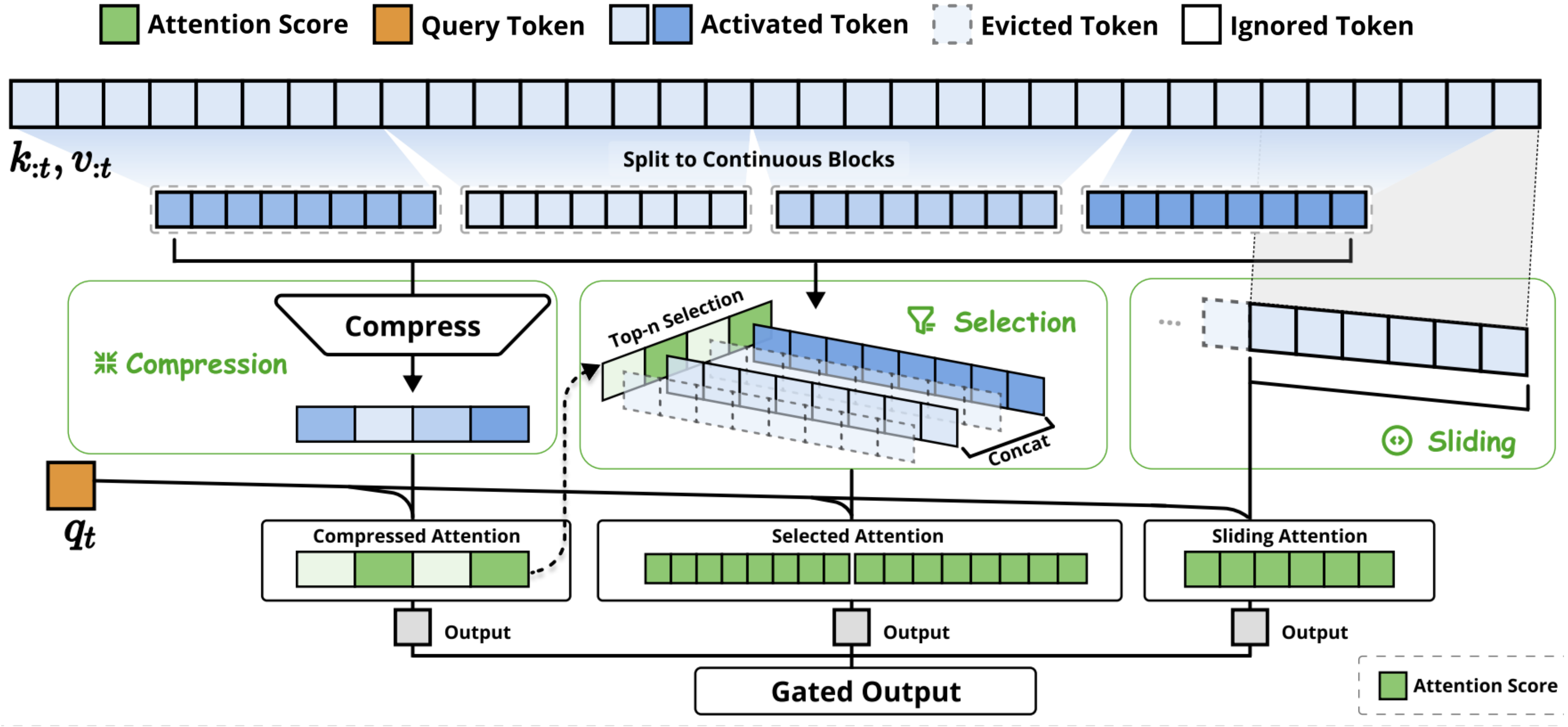
- Key insight: we can summarize long-context input, identify and focus on key words



[1] Native Sparse Attention: Hardware-Aligned and Natively Trainable Sparse Attention, ACL, 2025

Native Sparse Attention (ACL'25 from DeepSeek)

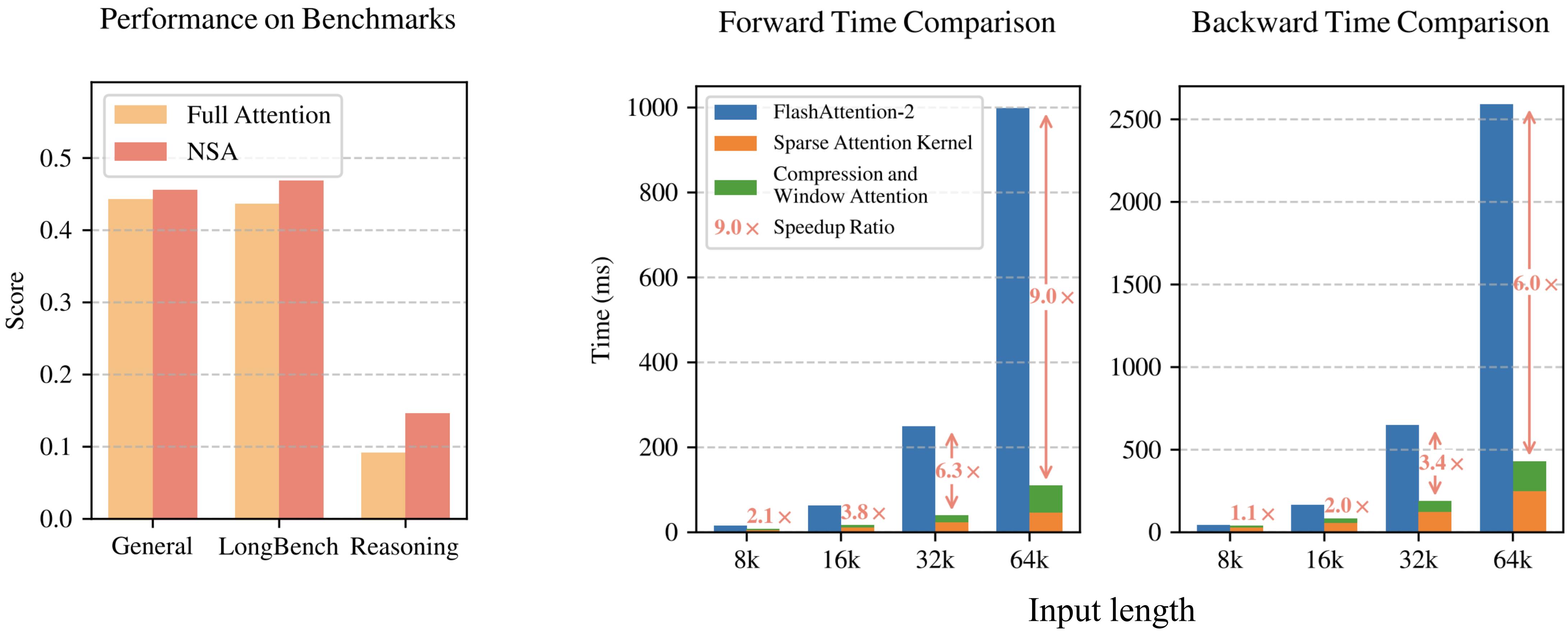
- Key insight: we can summarize long-context input, identify and focus on key words



[1] Native Sparse Attention: Hardware-Aligned and Natively Trainable Sparse Attention, ACL, 2025

Native Sparse Attention Performance

- Better quality, compute, and memory footprint



[1] Native Sparse Attention: Hardware-Aligned and Natively Trainable Sparse Attention, ACL, 2025

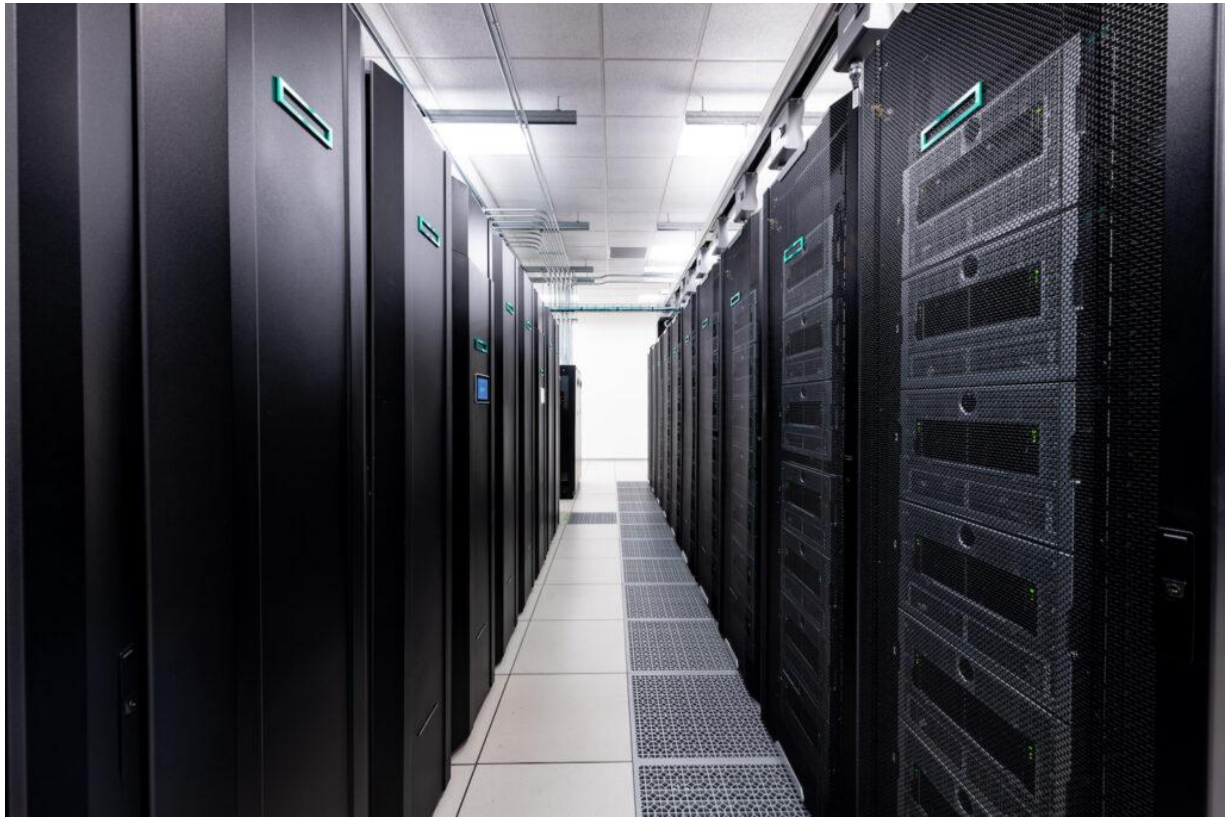
Summary

- **Transformers Deep Dive**
  - Transformer architecture
    - Tokenization, position embedding, MHSA mechanism
  - Multi-head & Multi-query Attention
    - Parallel execution of heads
  - Native Sparse Attention (ACL’25 Best Paper Award)
    - Summarize, focus on key tokens and neighboring tokens

Hardware: Delta

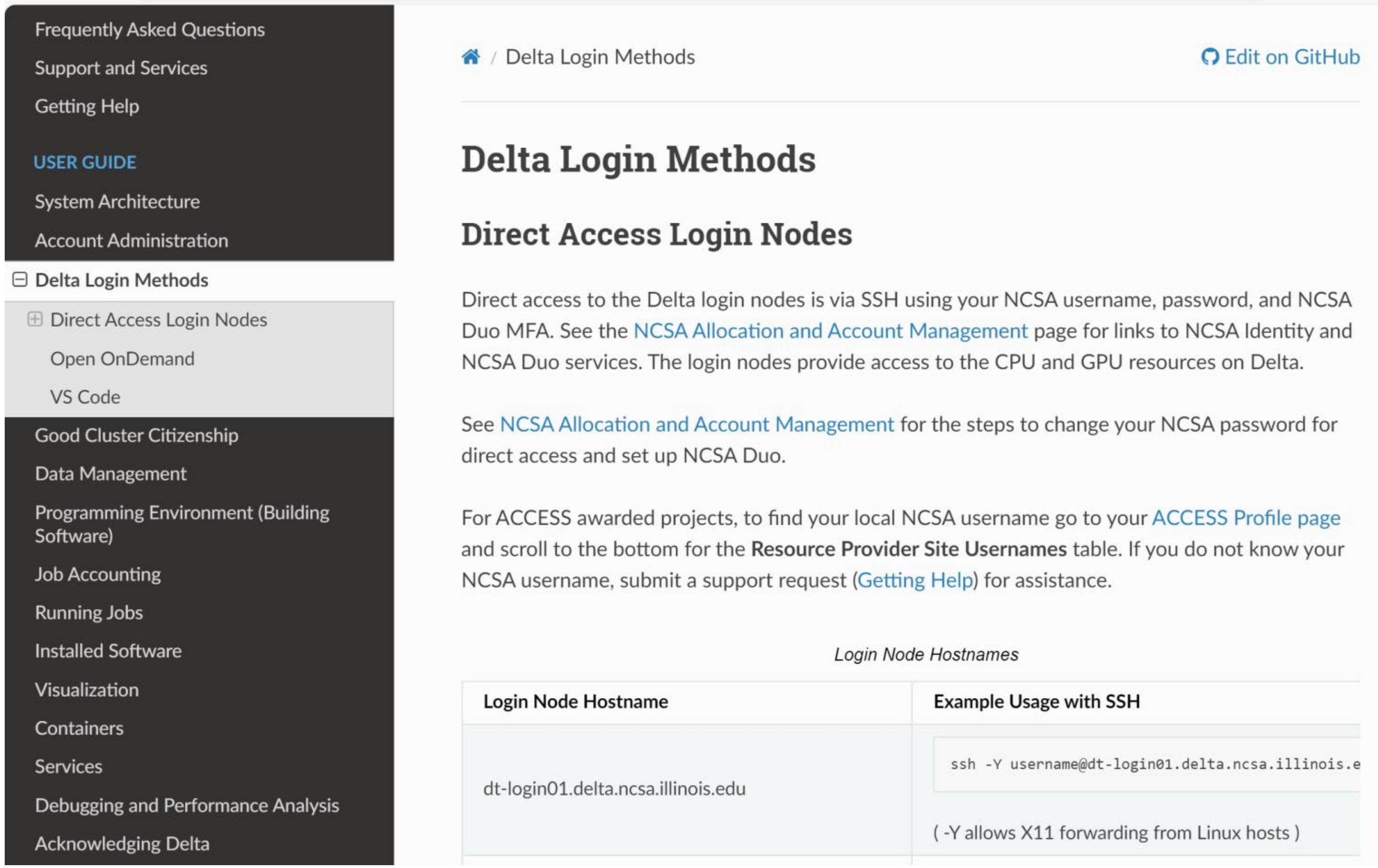
- Home page: https://www.ncsa.illinois.edu/research/project-highlights/delta/

- 100 quad A100 GPU node, each with 4 A100
- 100 quad A40 GPU node, each with 4 A40
- 5 8-way A100 GPU, each with 8 A100
- 1 MI100 node, 8 MI100



Delta Onboarding

- https://docs.ncsa.illinois.edu/systems/delta/en/latest/user_guide/accessing.html



Frequently Asked Questions
Support and Services
Getting Help

USER GUIDE
System Architecture
Account Administration
Delta Login Methods
Direct Access Login Nodes
Open OnDemand
VS Code
Good Cluster Citizenship
Data Management
Programming Environment (Building Software)
Job Accounting
Running Jobs
Installed Software
Visualization
Containers
Services
Debugging and Performance Analysis
Acknowledging Delta

/ Delta Login Methods   Edit on GitHub

Delta Login Methods

Direct Access Login Nodes

Direct access to the Delta login nodes is via SSH using your NCSA username, password, and NCSA Duo MFA. See the NCSA Allocation and Account Management page for links to NCSA Identity and NCSA Duo services. The login nodes provide access to the CPU and GPU resources on Delta.

See NCSA Allocation and Account Management for the steps to change your NCSA password for direct access and set up NCSA Duo.

For ACCESS awarded projects, to find your local NCSA username go to your ACCESS Profile page and scroll to the bottom for the **Resource Provider Site Usernames** table. If you do not know your NCSA username, submit a support request (Getting Help) for assistance.

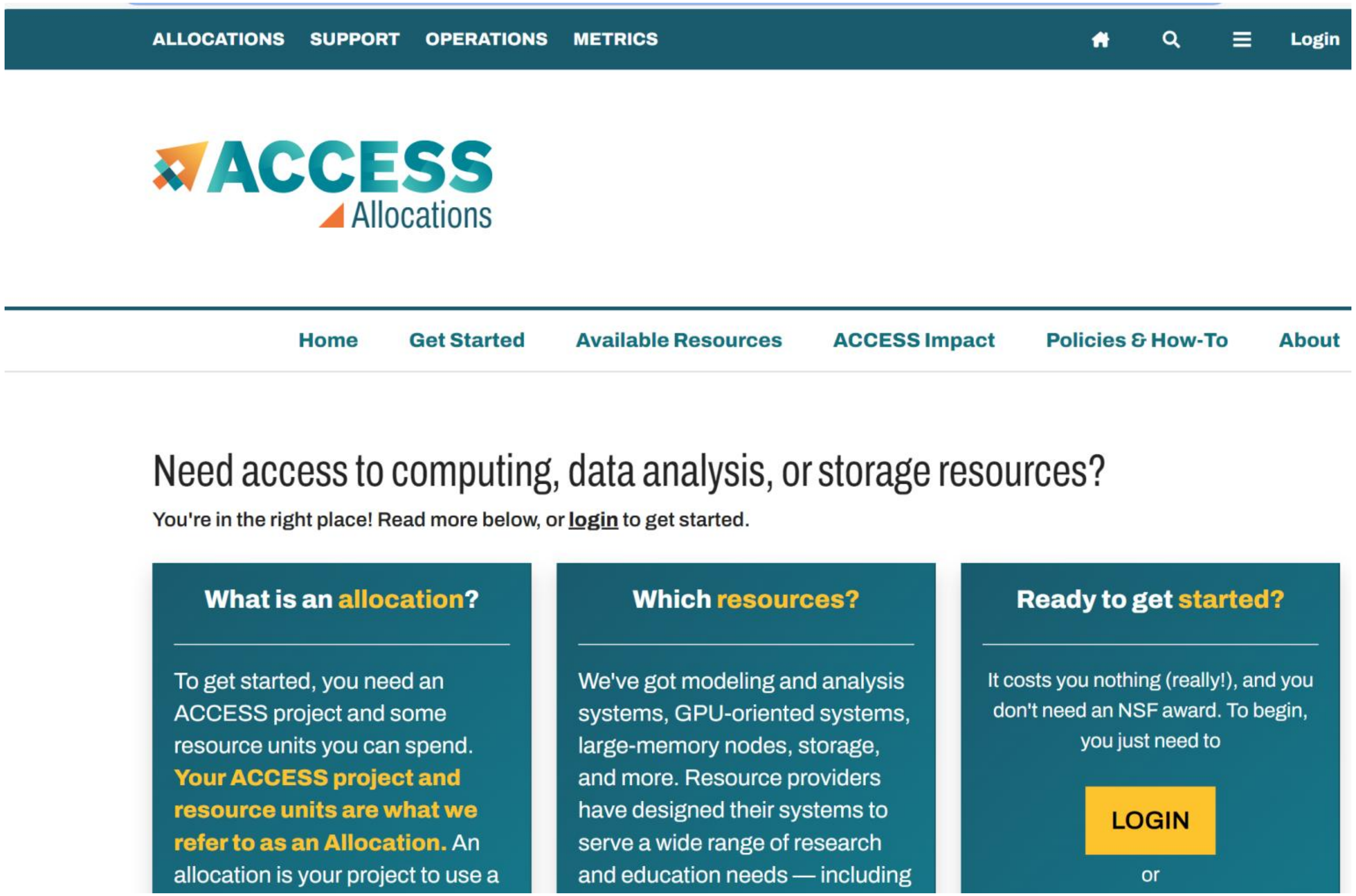
Login Node Hostnames

Login Node Hostname	Example Usage with SSH
dt-login01.delta.ncsa.illinois.edu	`ssh -Y username@dt-login01.delta.ncsa.illinois.e` (-Y allows X11 forwarding from Linux hosts)

Step 1: Create ACCESS ID

- Register an ACCESS id at: https://access-ci.org/ (top right-hand corner)

- After you register, send the instructor your ACCESS id. The instructor will add you to access to his GPU allocation.



Step 2: Get Started

- Delta uses Slurm to manage jobs/GPUs
- Please watch this tutorial video: Getting Started on NCSA's Delta Supercomputer.
- After that, you may want to check Delta User Documentation — UIUC NCSA Delta User Guide (illinois.edu).
- Please learn how to use slurm to get GPUs: Slurm Workload Manager - Quick Start User Guide (schedmd.com).

Step 3: SSH Login

- You shall use ssh to login to the node: [Delta Login Methods — UIUC NCSA Delta User Guide (illinois.edu)]().
- For instance, you can use commands such as "srun -A bcjw-delta-gpu --time=00:30:00 --nodes=1 --ntasks-per-node=16 --partition=gpuA100x4,gpuA40x4 --gpus=1 --mem=32g --pty /bin/bash"
- Maintaining Persistent Sessions: tmux

Delta Login Methods

Direct Access Login Nodes

Direct access to the Delta login nodes is via SSH using your NCSA username, password, and NCSA Duo MFA. See the NCSA Allocation and Account Management page for links to NCSA Identity and NCSA Duo services. The login nodes provide access to the CPU and GPU resources on Delta.

See NCSA Allocation and Account Management for the steps to change your NCSA password for direct access and set up NCSA Duo.

For ACCESS awarded projects, to find your local NCSA username go to your ACCESS Profile page and scroll to the bottom for the **Resource Provider Site Usernames** table. If you do not know your NCSA username, submit a support request (Getting Help) for assistance.

Login Node Hostnames

Login Node Hostname	Example Usage with SSH
dt-login01.delta.ncsa.illinois.edu	`ssh -Y username@dt-login01.delta.ncsa.illinois.e` (-Y allows X11 forwarding from Linux hosts)
dt-login02.delta.ncsa.illinois.edu	`ssh -l username dt-login02.delta.ncsa.illinois.e` (-l username alt. syntax for `user@host`)
login.delta.ncsa.illinois.edu (round robin DNS name for the set of login nodes)	`ssh username@login.delta.ncsa.illinois.edu`

Additional Info

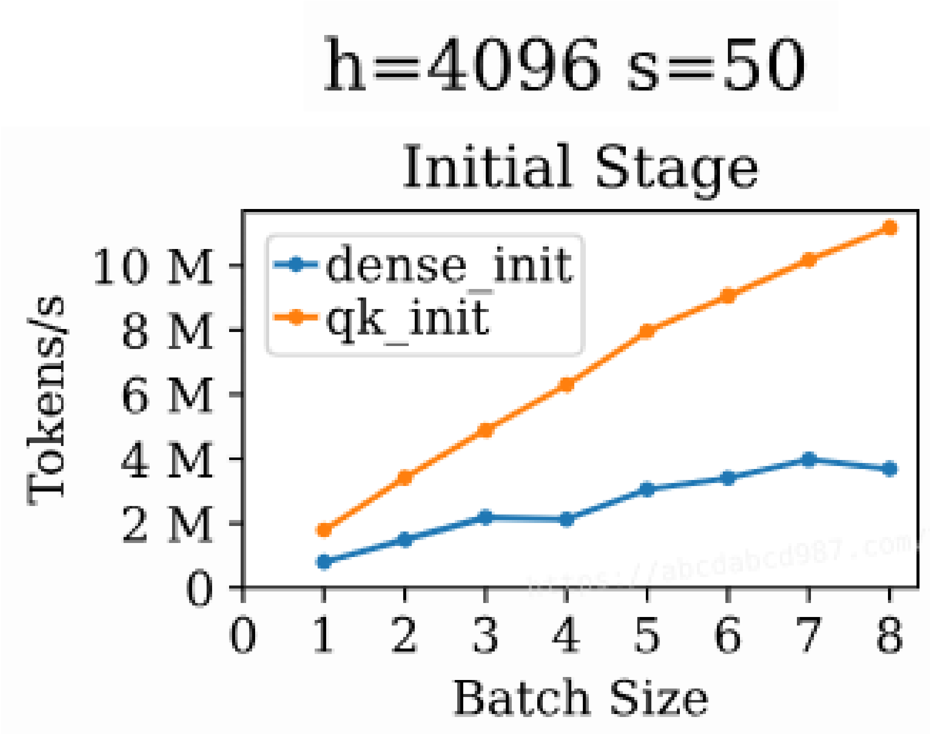
- It is the instructor’s own research allocation, and it has a limit. So please be mindful when using GPU resources.
  - Avoid allocating too many GPUs at once
  - Turn off the job when you are not using the GPUs

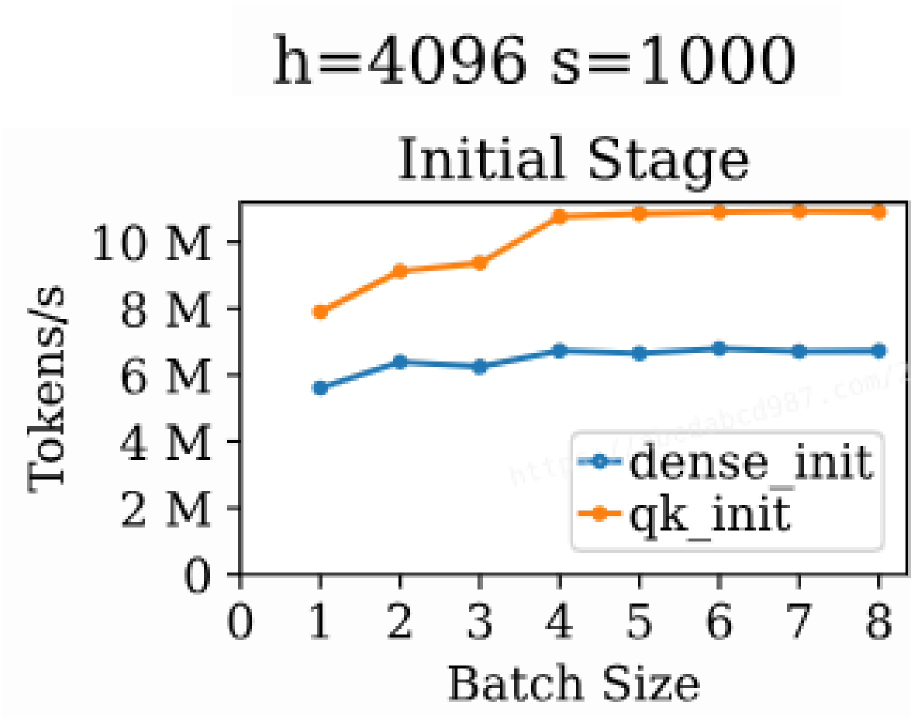
- The allocation has 500 GB of storage in total (shared by the class and other students in the instructor’s lab)
  - Please avoid downloading large data files and super large model checkpoints, e.g., one llama7b checkpoint consumes roughly 14GB.

- After class:
  - Walk through the AI calculation of Transformers
  - How many floating-point operations in total for a 7B decoder-only model?

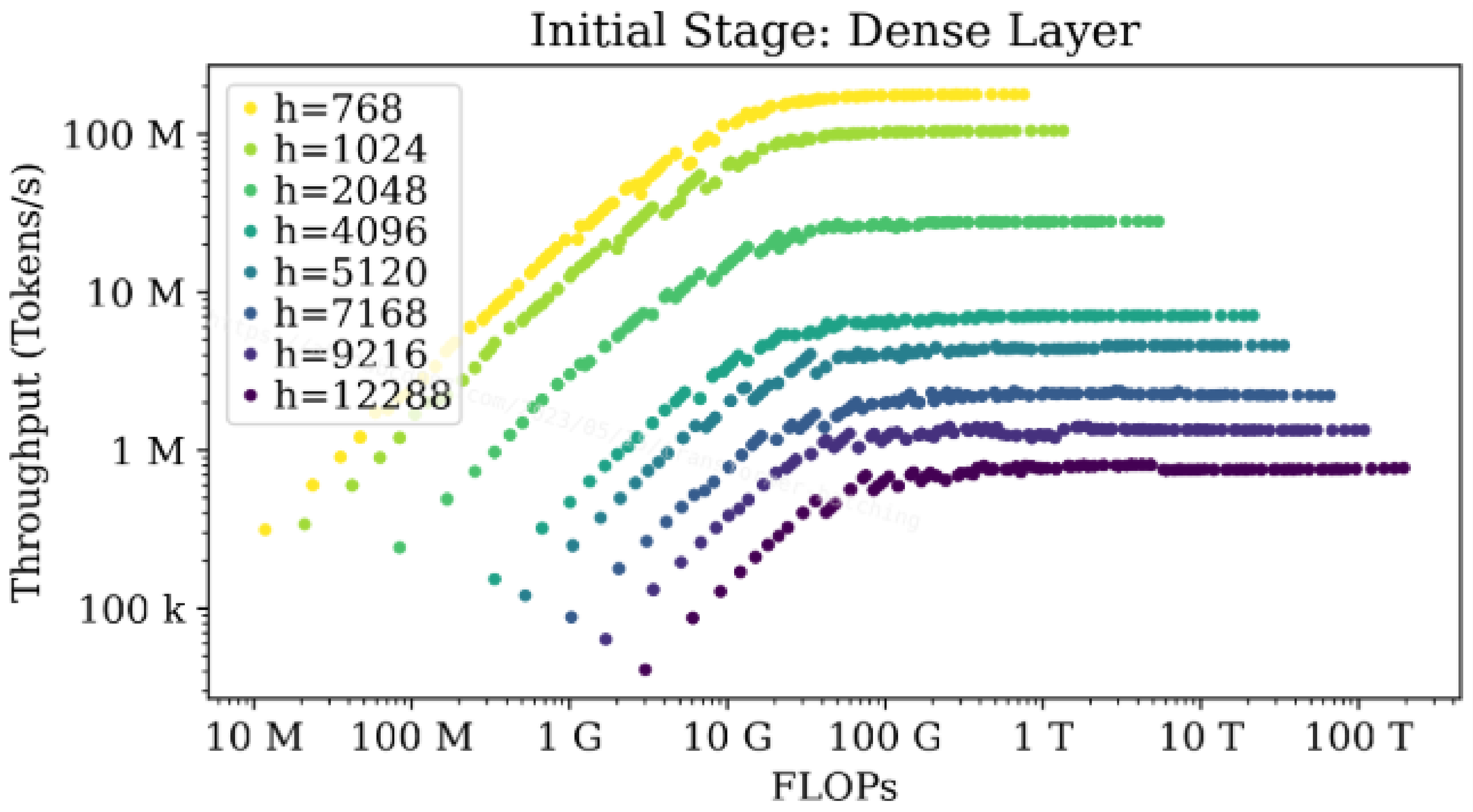
Questions?

Transformer Performance: Varying Batch Sizes

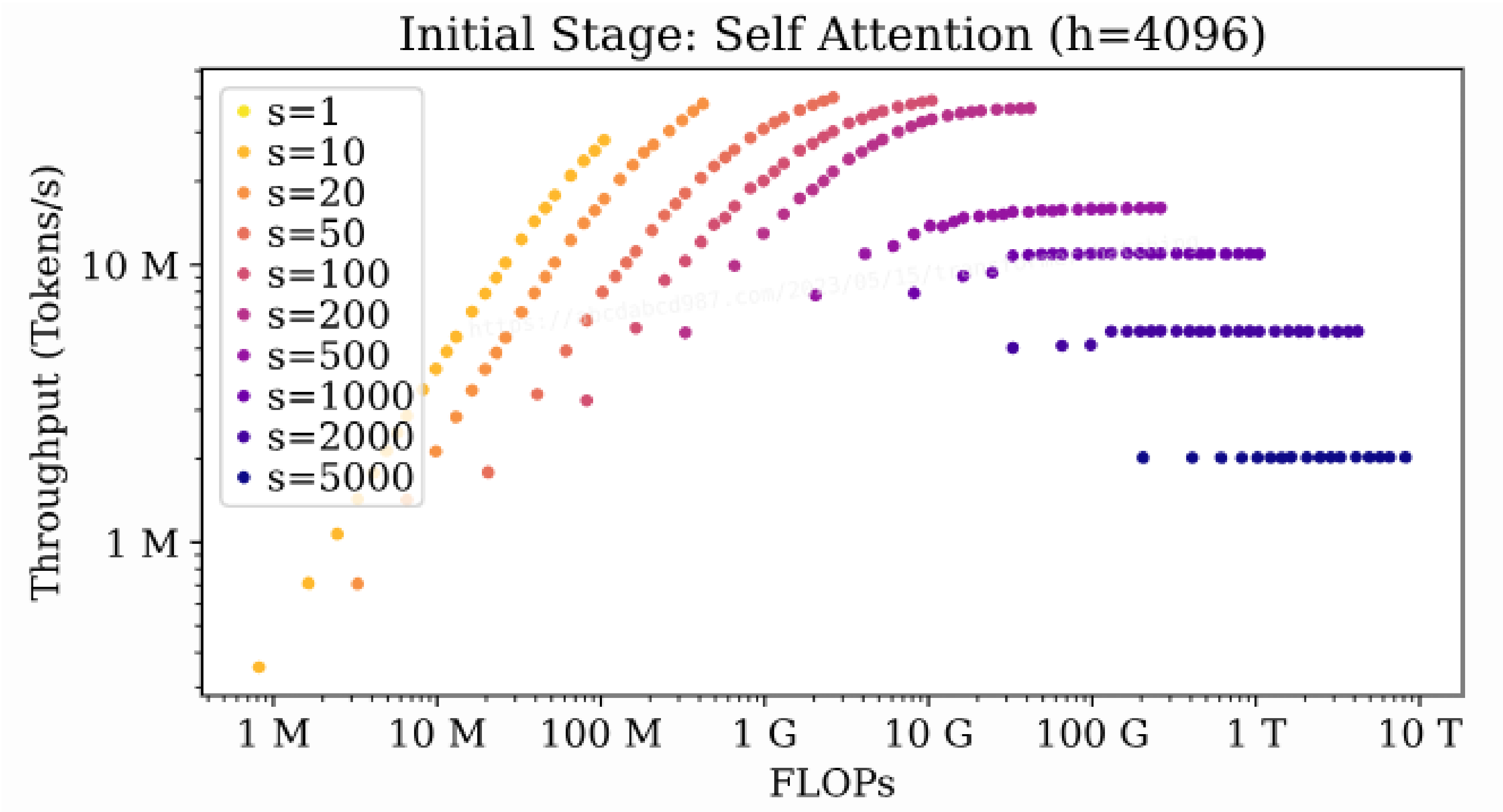




Transformer Performance: Batching of Dense Layer



Transformer Performance: Batching of Self-Attention



Transformer Performance: Roofline