



CS 498: Machine Learning System

Spring 2026

Minja Zhang

The Grainger College of Engineering

Tensor Slicing Model Parallelism

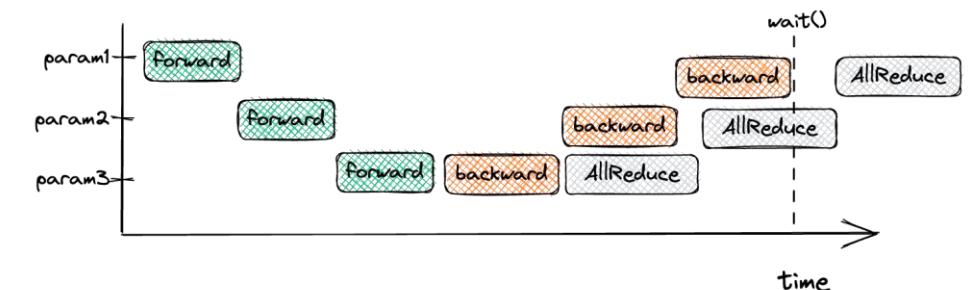
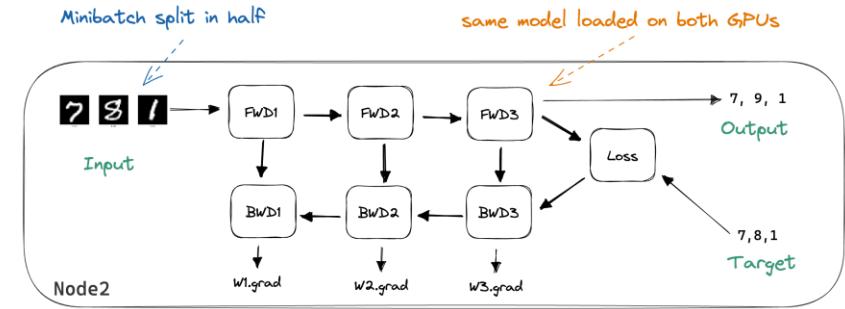
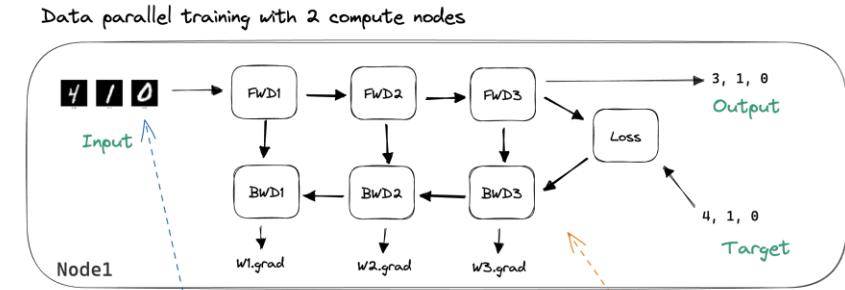
Learning Objectives

- Explain why data parallelism alone is insufficient for training LLMs
- Understand the difference between tensor vs. pipeline parallelism

Data Parallelism



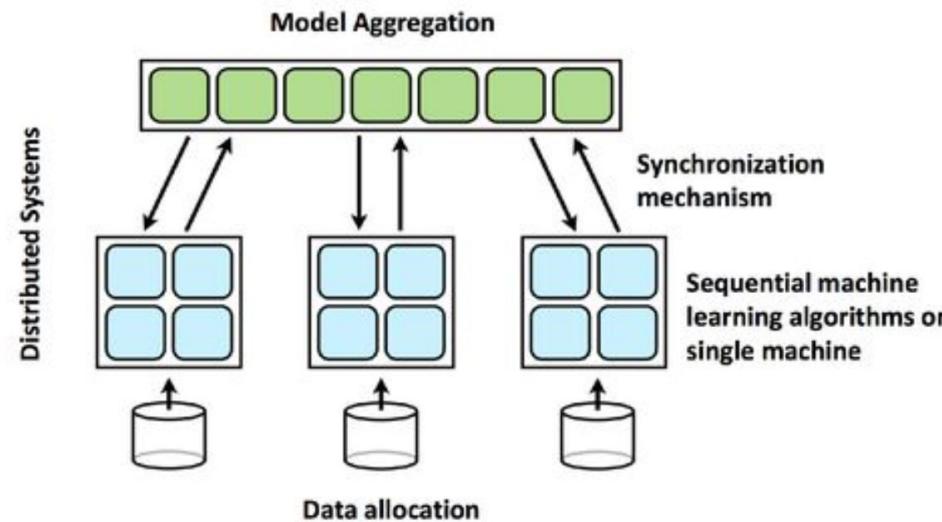
- Partitions a training minibatch across multiple devices
- Linearly scalable
- Slicing activation only
- Does not help with excessive model size



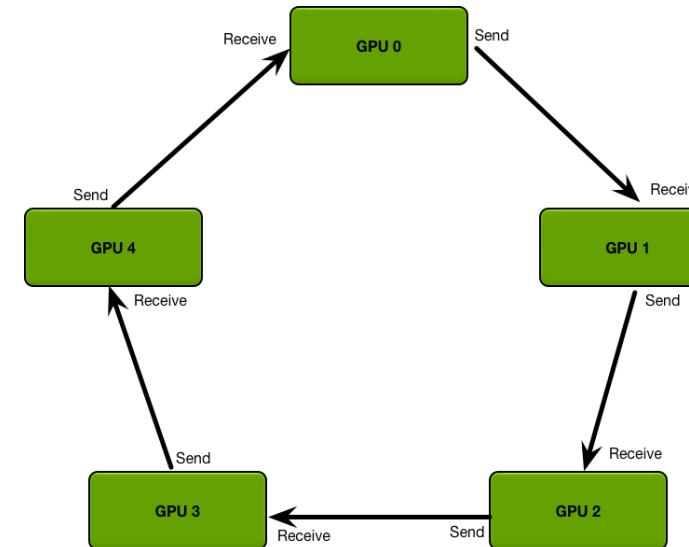
Recall: Data Parallelism



Parameter Server



AllReduce



Data parallelism does not help with excessive model size

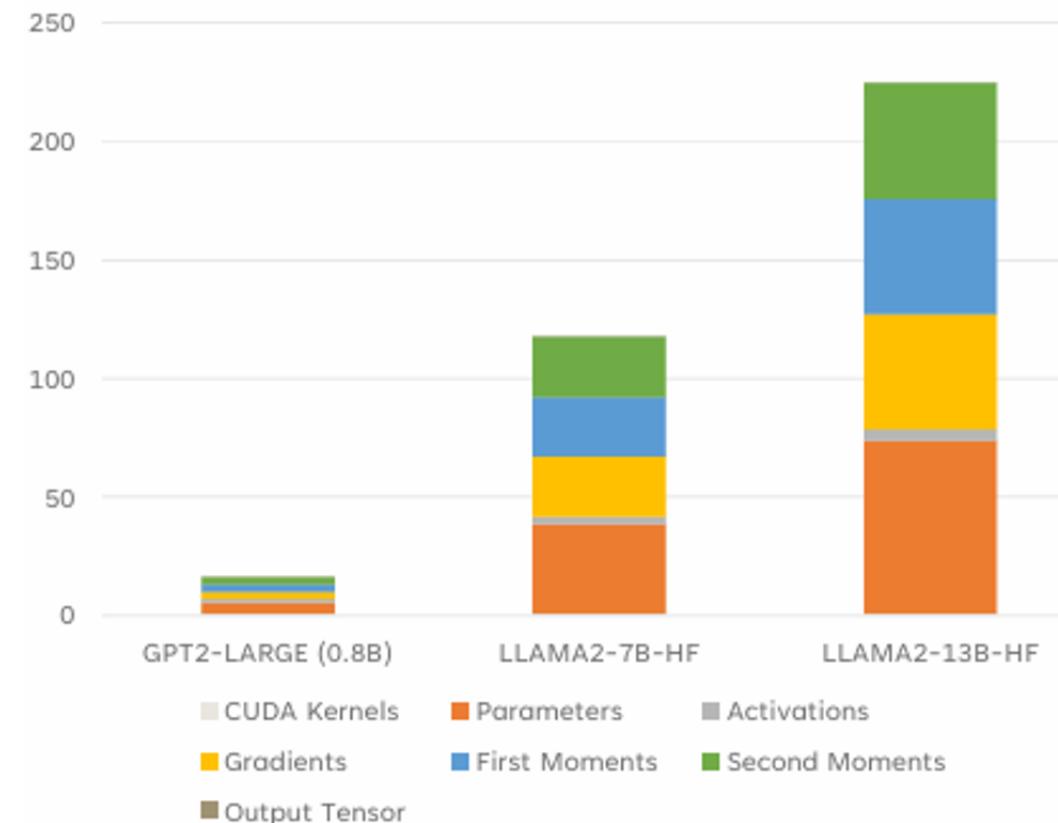
Large Models Require Large Memory



- Training >> Inference
- Adam >> SGD (*due to optimizer state*)
- Larger minibatch
- Scaling law => More parameters
- ...

Question: How do we speed up training or simply enable training?

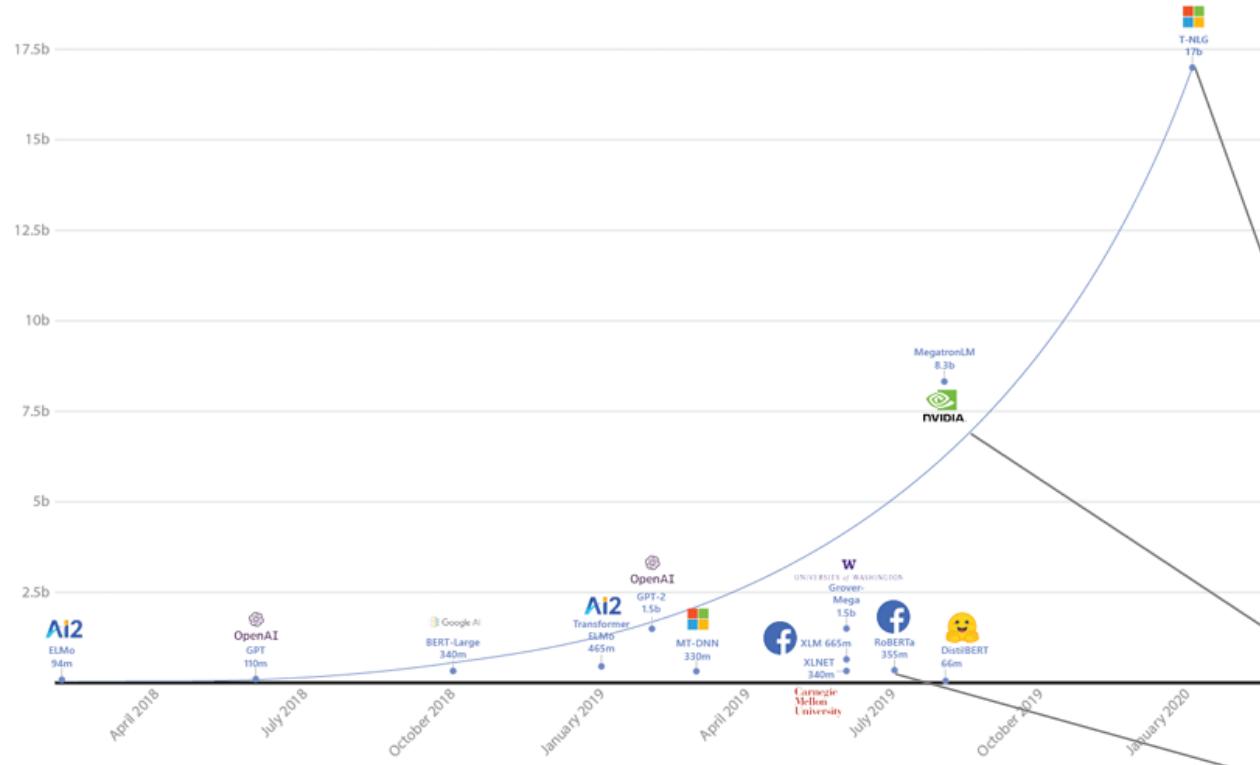
GPU VRAM Usage Estimation for training different models



- Paper: <https://arxiv.org/abs/1909.08053>
- Repo:
<https://github.com/NVIDIA/Megatron-LM>
- NVIDIA's framework, released in 2019, for efficiently training large-scale language models



Computational Needs of NLP



Model	# of Parameters	# of Iterations	# of GPUs	Training Time
T-NLG (MS + NV)	17.2 B	300 K	400	60 days
Megatron-GPT2 (NV)	8.3 B	300 K	512	10 days
Megatron-BERT (NV)	3.9 B	2 M	512	25 days
RoBERTa (FB)	345M	4 M	1024	1 day

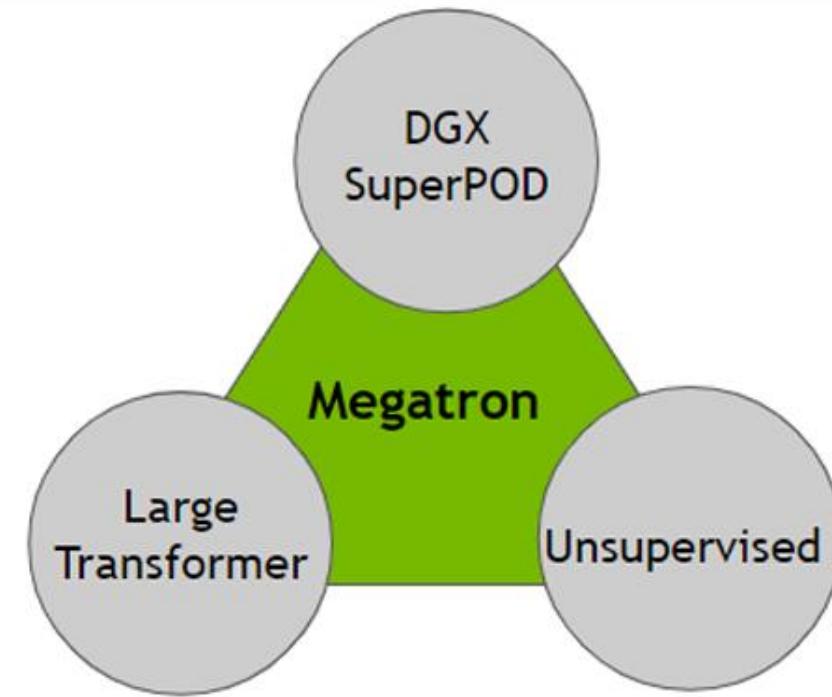
Motivation: Why Megatron?



Training **larger transformer-based** language models became an important way to advance SoTA NLP applications.

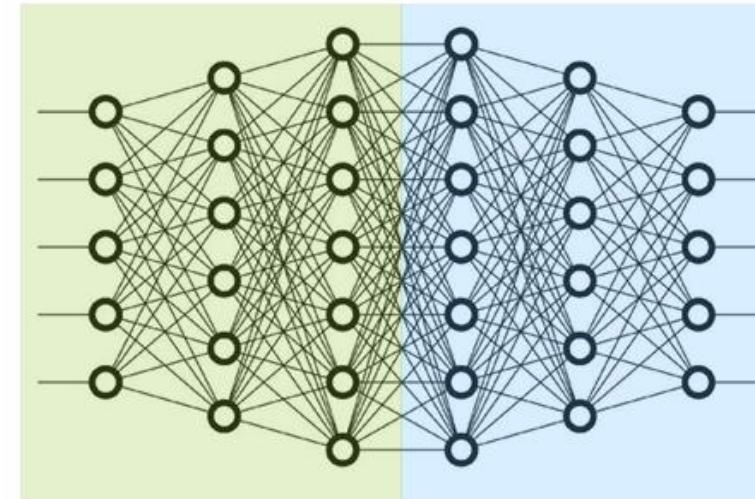
Unsupervised language models such as GPT-2, BERT, and XLNet demonstrate the power of scaling language models trained on a huge corpus.

Nvidia DGX boxes optimized for deep learning provides a unique opportunity for training very large models.



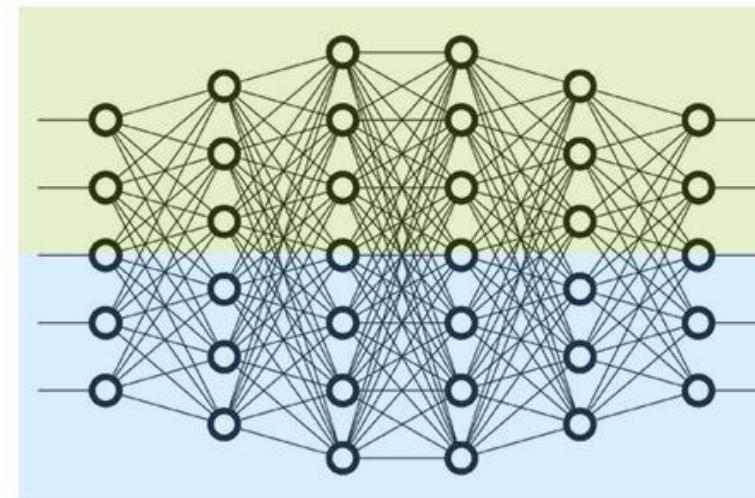
Inter-layer (Pipeline) parallelism

- Split sets of layers across multiple devices
- Layer 0, 1, 2 and layer 3, 4, 5 are on different devices



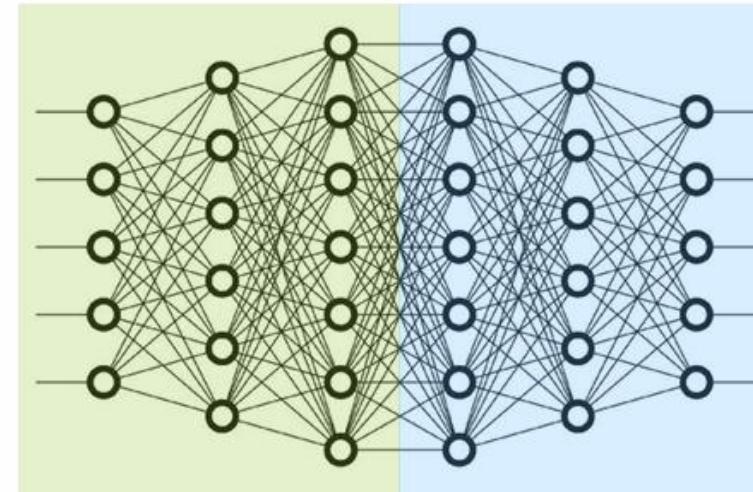
Intra-layer (Tensor) parallelism

- Split individual layers across multiple devices
- Both devices compute different parts of layer 0, 1, 2, 3, 4, 5



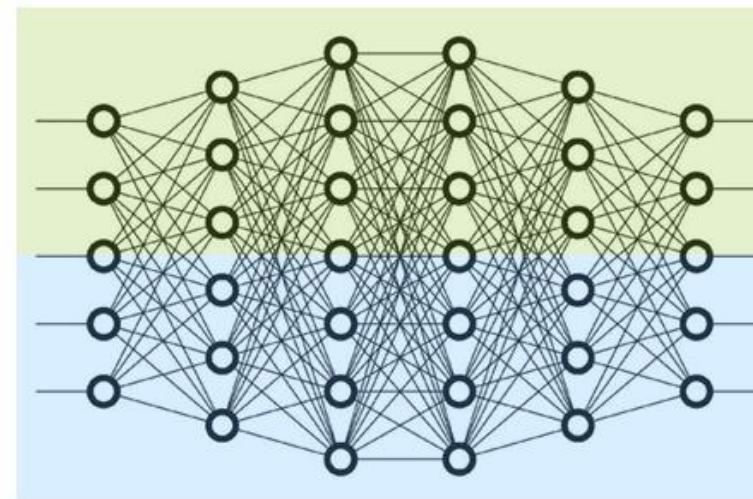
Inter-layer (Pipeline) parallelism

- Split sets of layers across multiple devices
- Layer 0, 1, 2 and layer 3, 4, 5 are on different devices

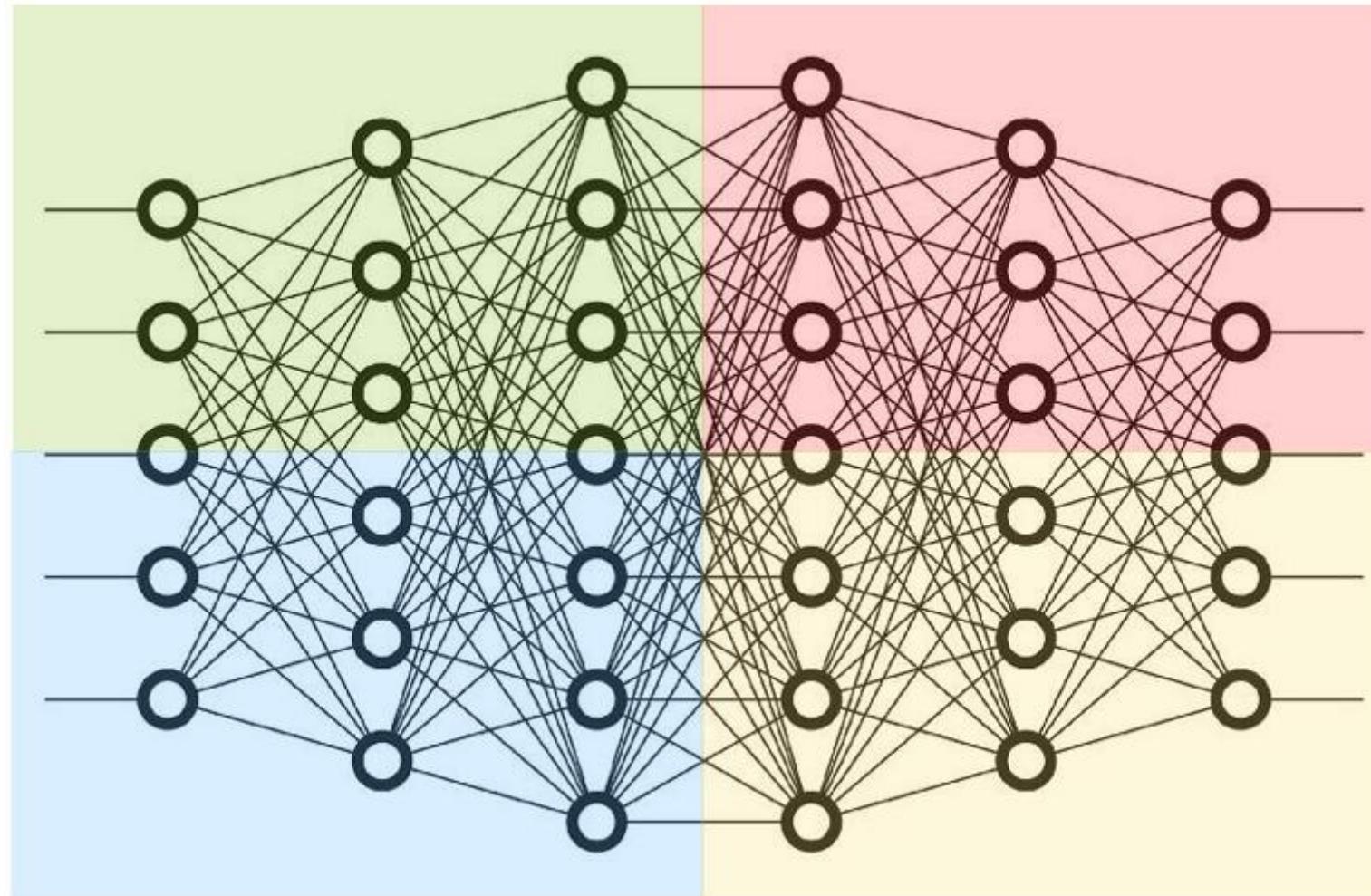


Intra-layer (Tensor) parallelism

- Split individual layers across multiple devices
- Both devices compute different parts of layer 0, 1, 2, 3, 4, 5

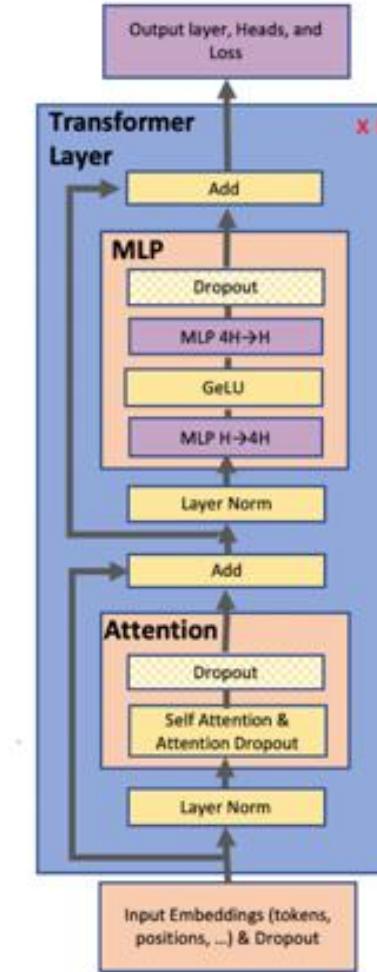


Complementary Types of Model Parallelism



Inter + Intra Parallelism

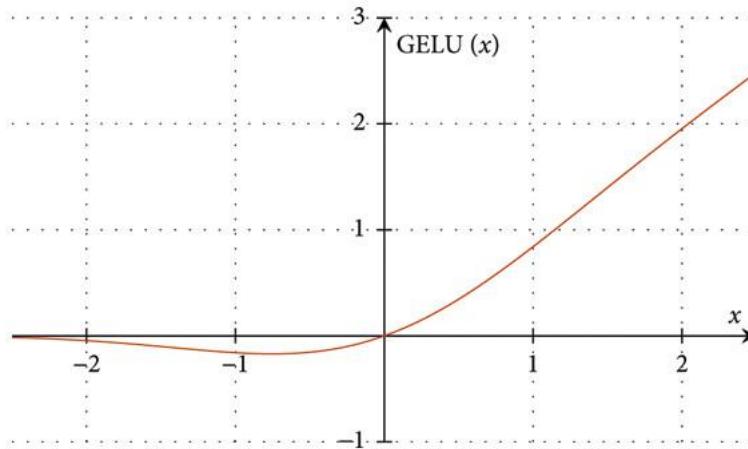
- Tailored for transformer networks
- Transformer layer
 - Self-attention block
 - Two-layer MLP
- Targets:
 - Reduce synchronization cost
 - Simple to implement with a few highly optimized collectives NCCL provides



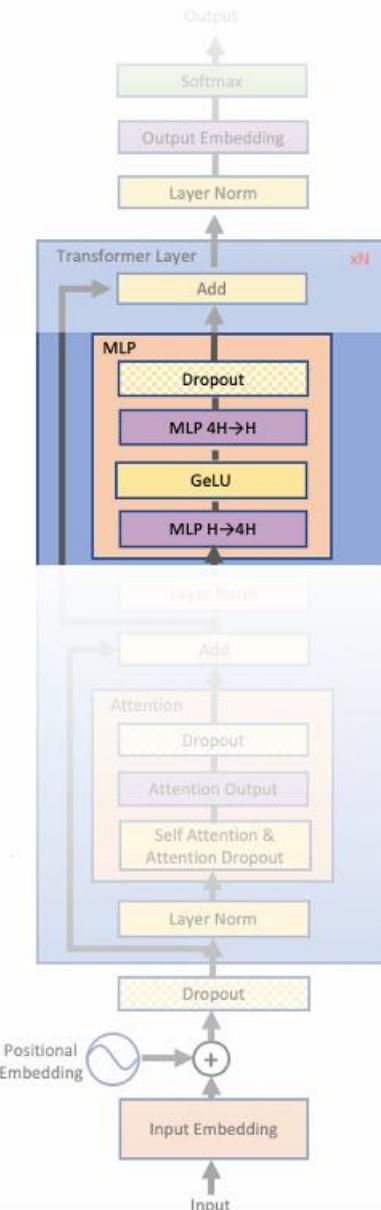
- ▶ MLP:

$$Y = \text{GeLU}(XA)$$

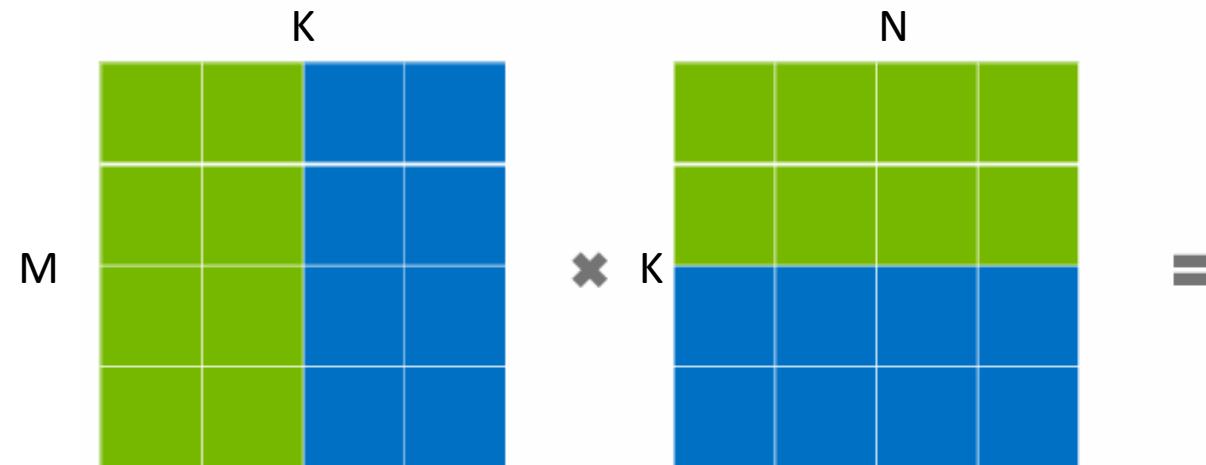
$$Z = \text{Dropout}(YB)$$



Dropout (YB):
Randomly drop a certain percentage of values to prevent overfitting



Tensor Parallel: Parallel GeMM (General Matrix Multiplications)

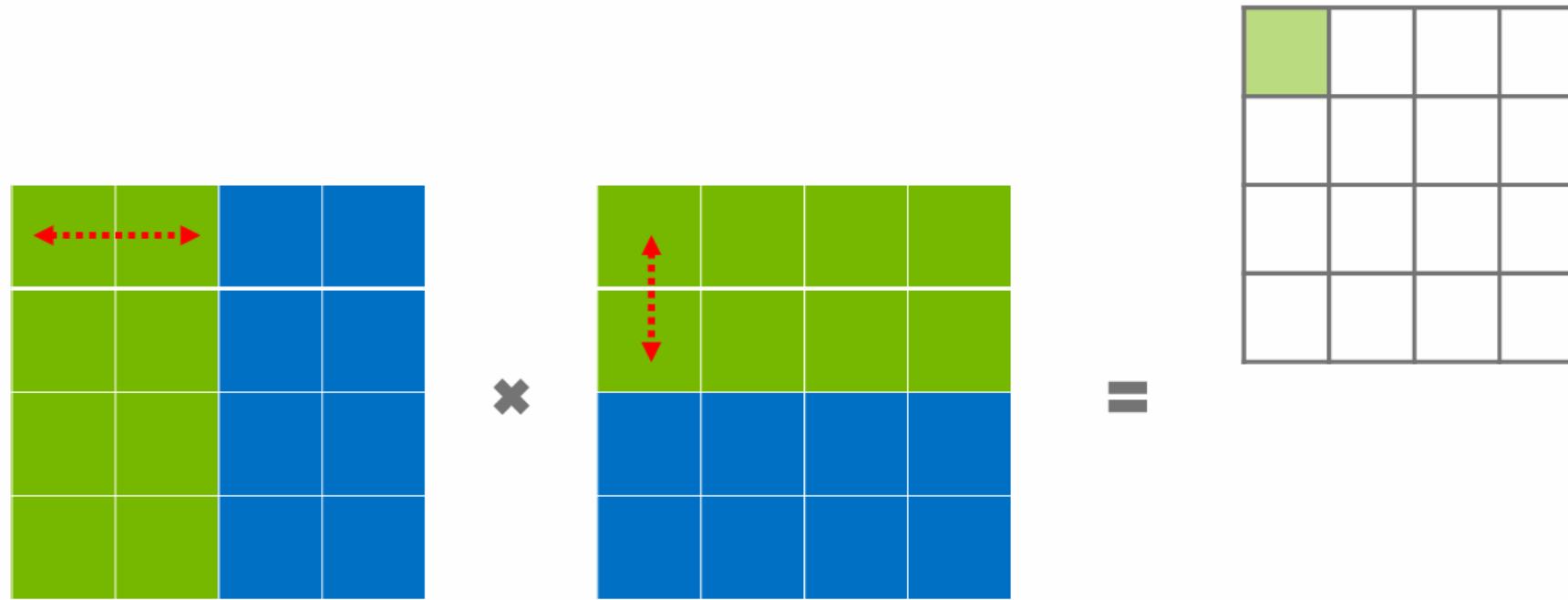


$$X = [X_1, X_2]$$

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$$

$$Y = Y_1 + Y_2$$

Tensor Parallel: Parallel GeMM (General Matrix Multiplications)

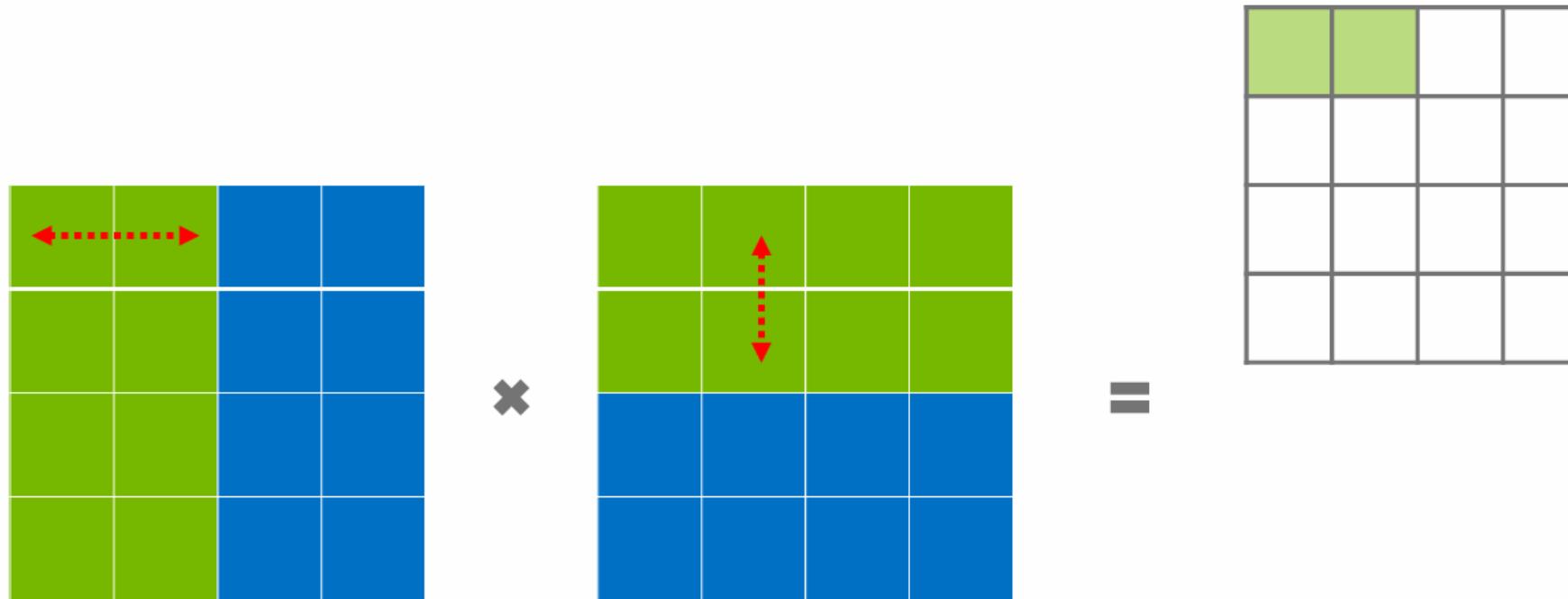


$$X = [X_1, X_2]$$

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$$

$$Y = Y_1 + Y_2$$

Tensor Parallel: Parallel GeMM (General Matrix Multiplications)

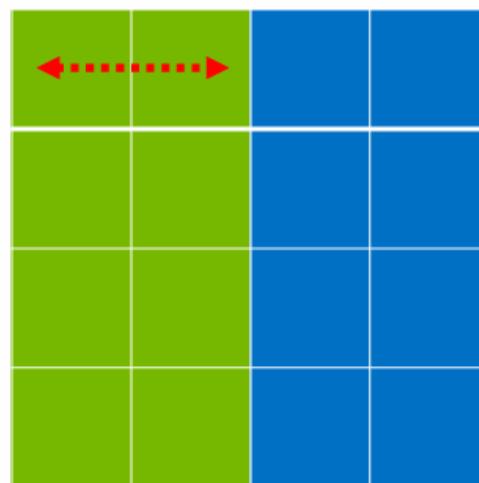


$$X = [X_1, X_2]$$

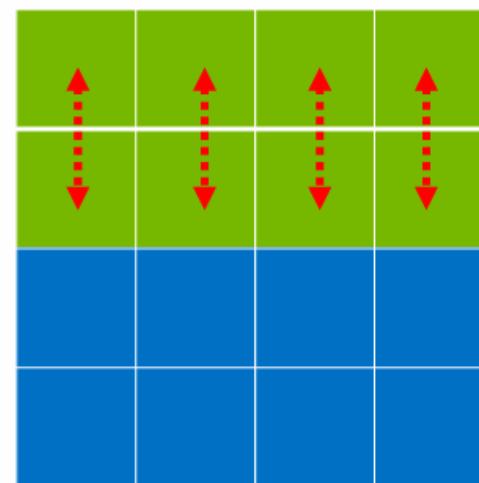
$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$$

$$Y = Y_1 + Y_2$$

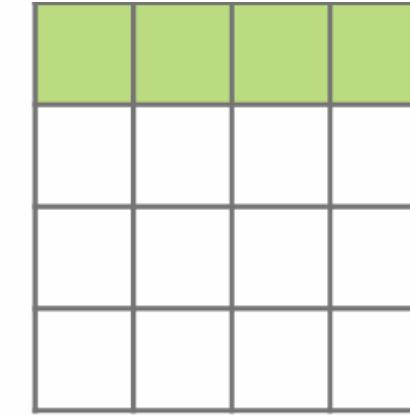
Tensor Parallel: Parallel GeMM



\times



$=$

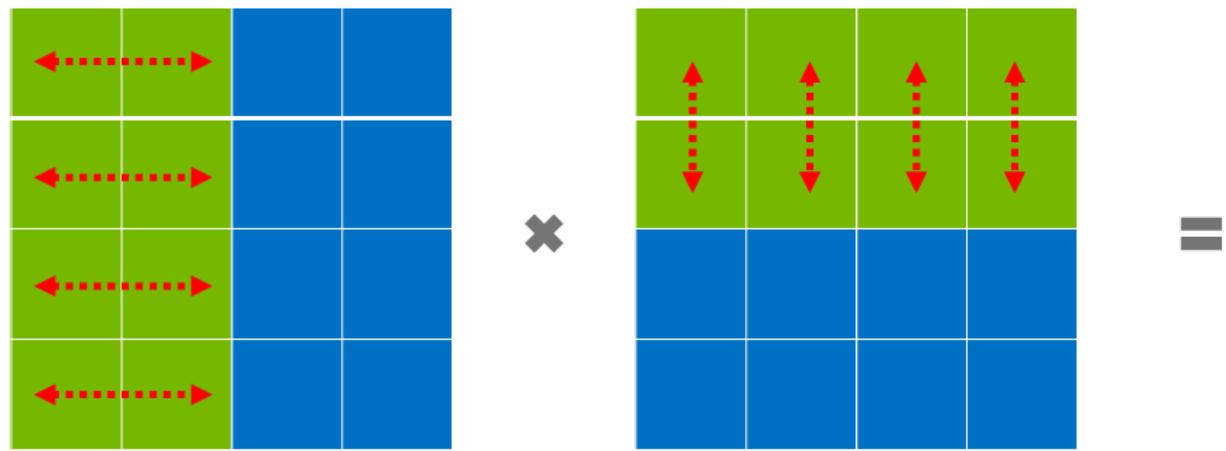


$$X = [X_1, X_2]$$

$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$$

$$Y = Y_1 + Y_2$$

Tensor Parallel: Parallel GeMM

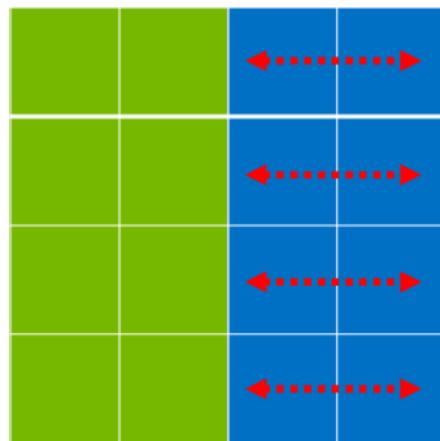


$$X = [X_1, X_2]$$

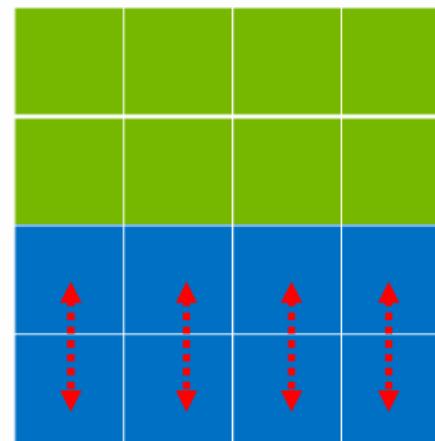
$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$$

$$Y = Y_1 + Y_2$$

Tensor Parallel: Parallel GeMM

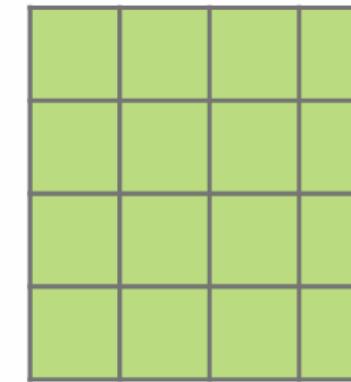


$$X = [X_1, X_2]$$



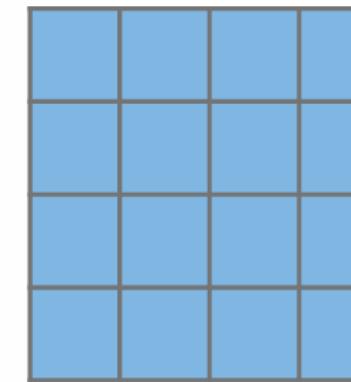
$$A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$$

=



Y_1

+

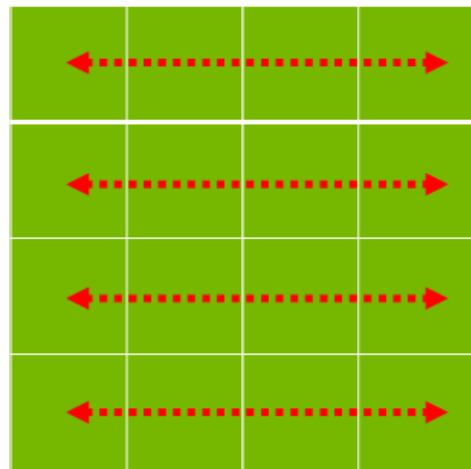
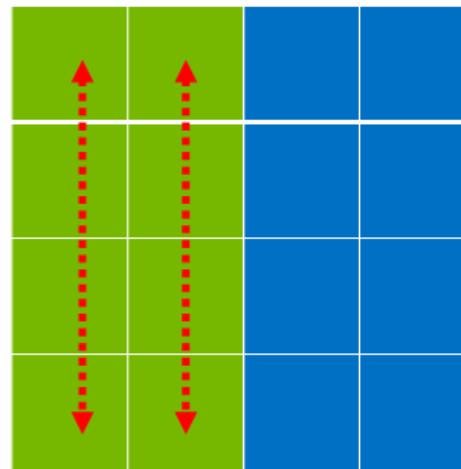
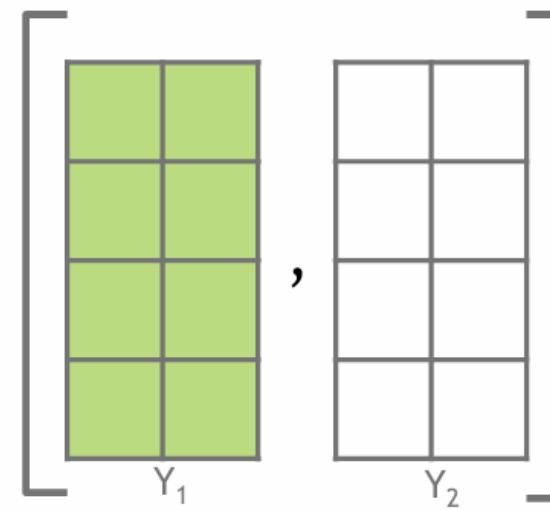


Y_2

= Y

$$Y = Y_1 + Y_2$$

Tensor Parallel: Parallel GeMM

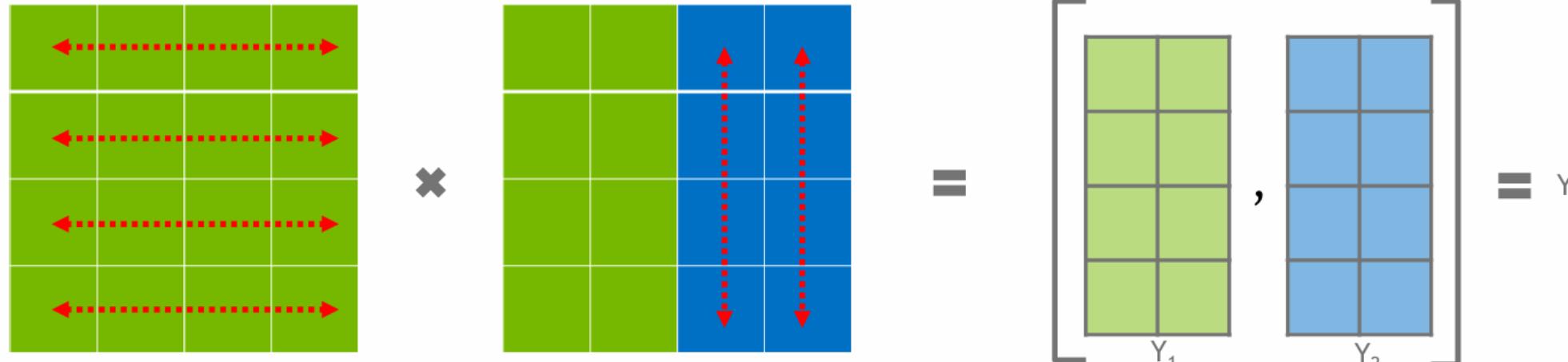
 X \times  $A = [A_1, A_2]$ $=$  $Y = [Y_1, Y_2]$ $= Y$

Tensor Parallel: Parallel GeMM

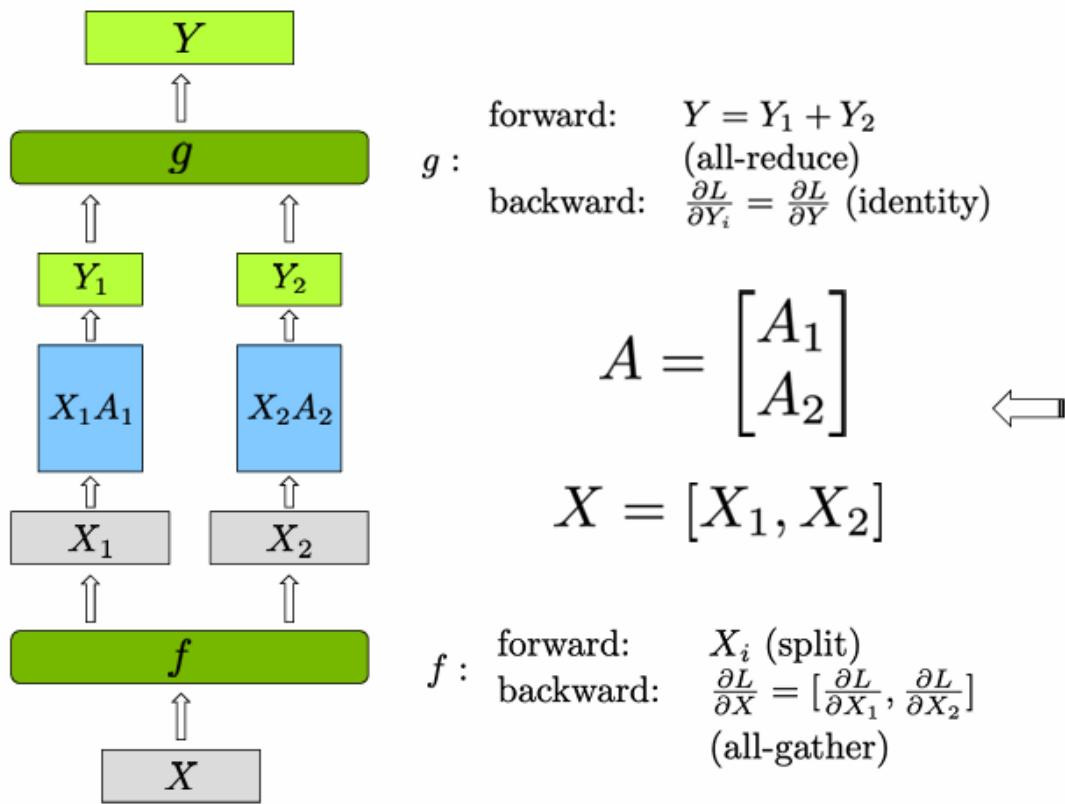


$$X \times A = [A_1, A_2] = [Y_1, Y_2]$$

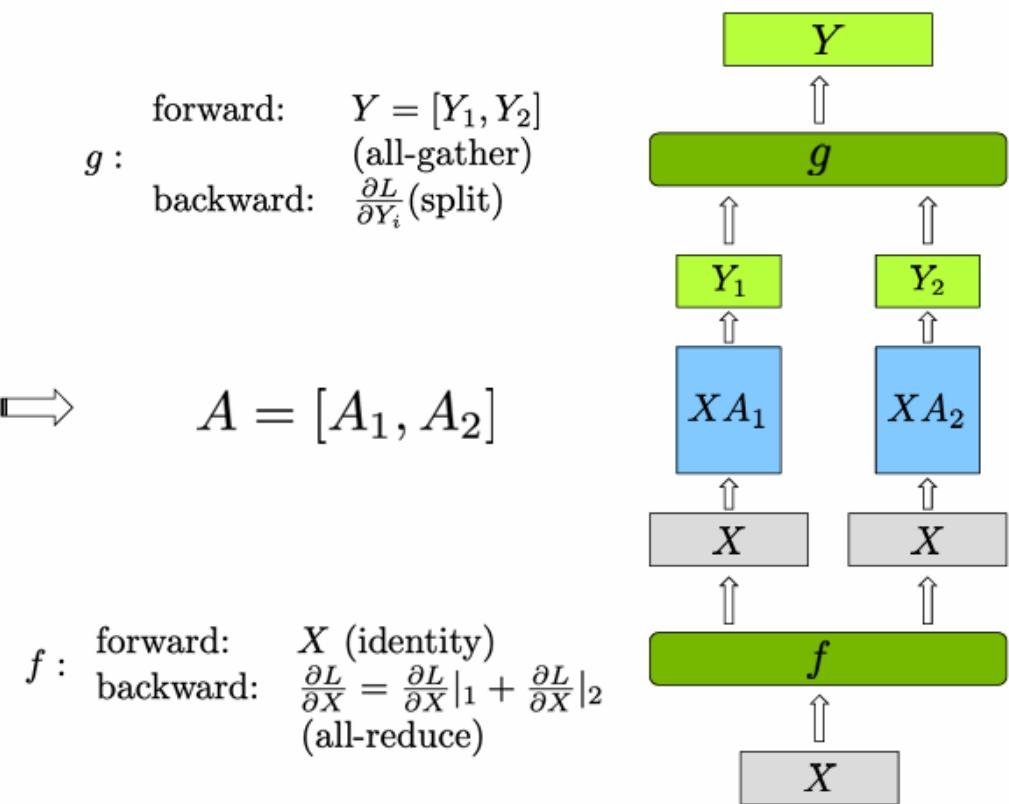
The diagram illustrates the parallel computation of a matrix multiplication. On the left, a green 4x4 matrix X is multiplied by a matrix A , which is partitioned into two parallel blocks A_1 and A_2 . The result is a matrix Y , partitioned into Y_1 and Y_2 . The diagram shows that the columns of X are processed in parallel by the two blocks of A , and the results are combined to form the final matrix Y .



Row Parallel Linear Layer



Column Parallel Linear Layer



Normal

Operation: $Y_{n \times n} = X_{n \times n} A_{n \times n}$

Flops: $2n^3$

Bandwidth: $6n^2$

Intensity: $\frac{1}{3}n$

Parallel

Operation: $Y_{n \times (n/p)} = X_{n \times n} A_{n \times (n/p)}$

Flops: $2n^3/p$

Bandwidth: $2n^2(1 + 2/p)$

Intensity: $\frac{1}{2 + p}n$

Ratio between serial and parallel

Flops: $1/p$ 

Bandwidth: $\frac{1 + 2/p}{3}$ 

Intensity: $\frac{3}{2 + p}$ 

Tensor Parallelism: MLP

► MLP:

$$Y = \text{GeLU}(XA)$$

$$Z = \text{Dropout}(YB)$$

- ▶ Approach 1: split X column-wise and A row-wise

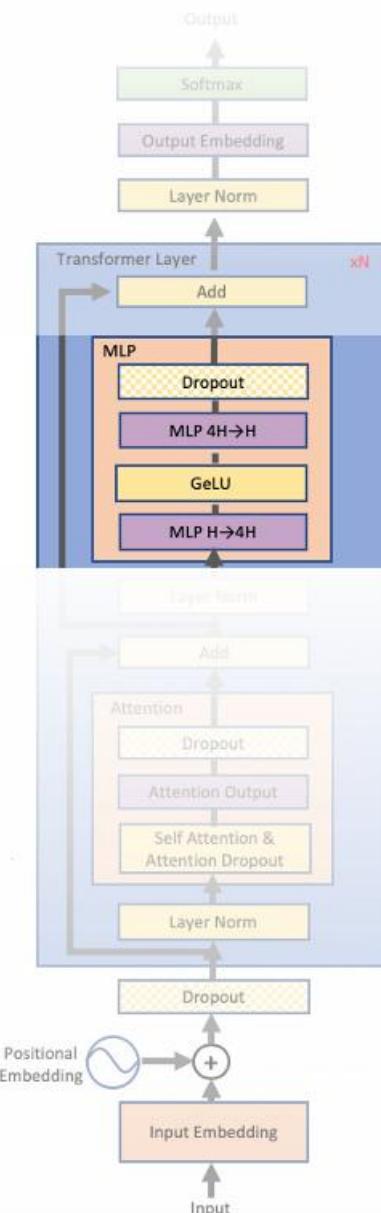
$$X = [X_1, X_2] \quad A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \quad \longrightarrow \quad Y = \text{GeLU}(X_1 A_1 + X_2 A_2)$$

GeLU of sums != sum of GeLUs

$$\begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix} \begin{pmatrix} A_0 & A_1 \\ A_2 & A_3 \end{pmatrix} = \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix} (A_0 \quad A_1) + \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix} (A_2 \quad A_3)$$

$= X_A A_A + X_B A_B$

Summation



Tensor Parallelism: MLP



- ▶ MLP:

$$Y = \text{GeLU}(XA)$$

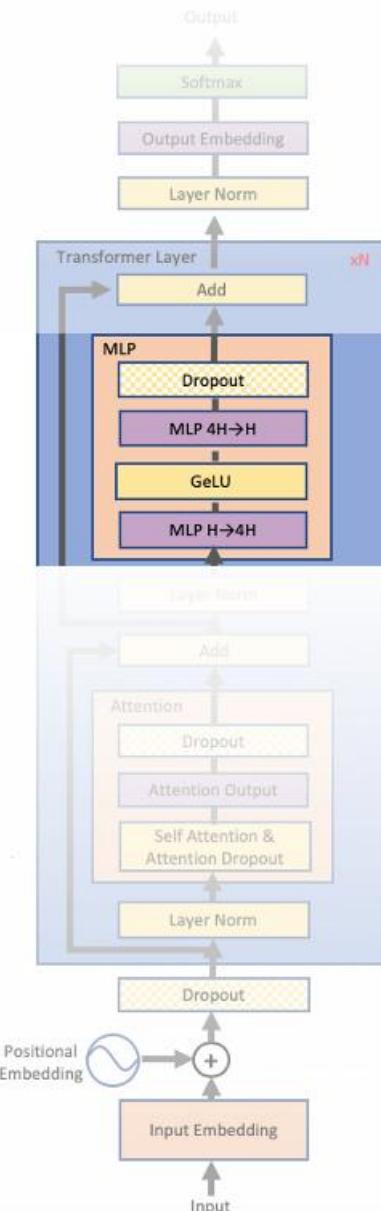
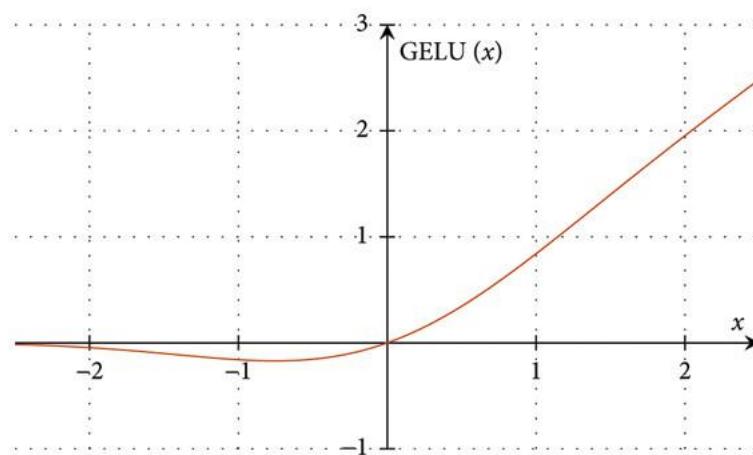
$$Z = \text{Dropout}(YB)$$

Requires AllReduce
before GeLU

- ▶ Approach 1: split X column-wise and A row-wise

$$X = [X_1, X_2] \quad A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \longrightarrow Y = \text{GeLU}(X_1A_1 + X_2A_2)$$

GeLU of sums \neq sum of GeLUs



Tensor Parallelism: MLP



- ▶ MLP:

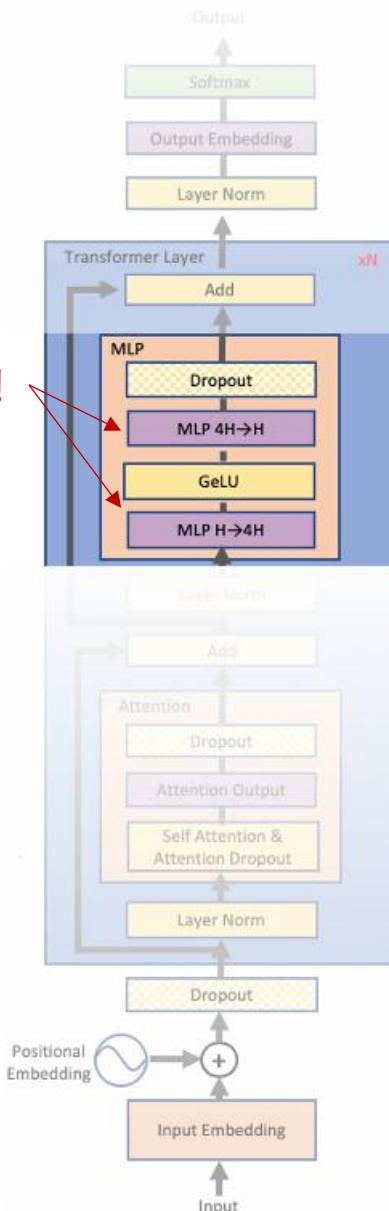
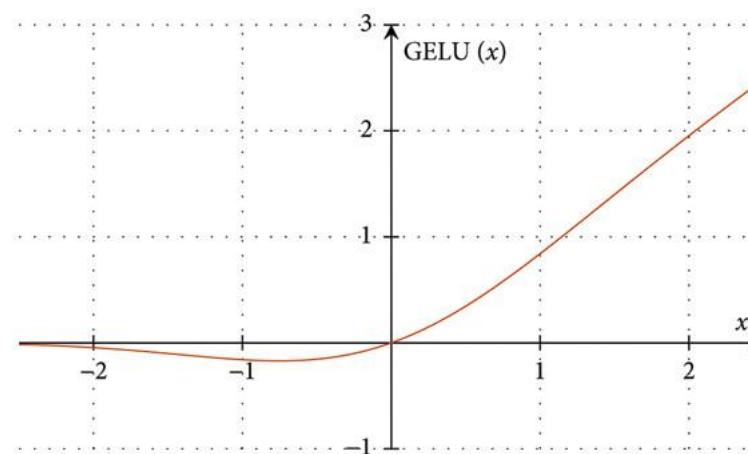
$$Y = \text{GeLU}(XA)$$
$$Z = \text{Dropout}(YB)$$

- ▶ Approach 1: split X column-wise and A row-wise

$$X = [X_1, X_2] \quad A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \longrightarrow Y = \text{GeLU}(X_1A_1 + X_2A_2)$$

GeLU of sums != sum of GeLUs

Shape(Y) = [B, S, 4H]!



Tensor Parallelism: MLP



- ▶ MLP:

$$Y = \text{GeLU}(XA)$$

$$Z = \text{Dropout}(YB)$$

- ▶ Approach 1: split X column-wise and A row-wise

$$X = [X_1, X_2] \quad A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \longrightarrow Y = \text{GeLU}(X_1A_1 + X_2A_2)$$

- ▶ Requires synchronization before GeLU

GeLU of sums \neq sum of GeLUs

- ▶ Approach 2: split A column-wise

$$A = [A_1, A_2] \longrightarrow [Y_1, Y_2] = [\text{GeLU}(XA_1), \text{GeLU}(XA_2)]$$

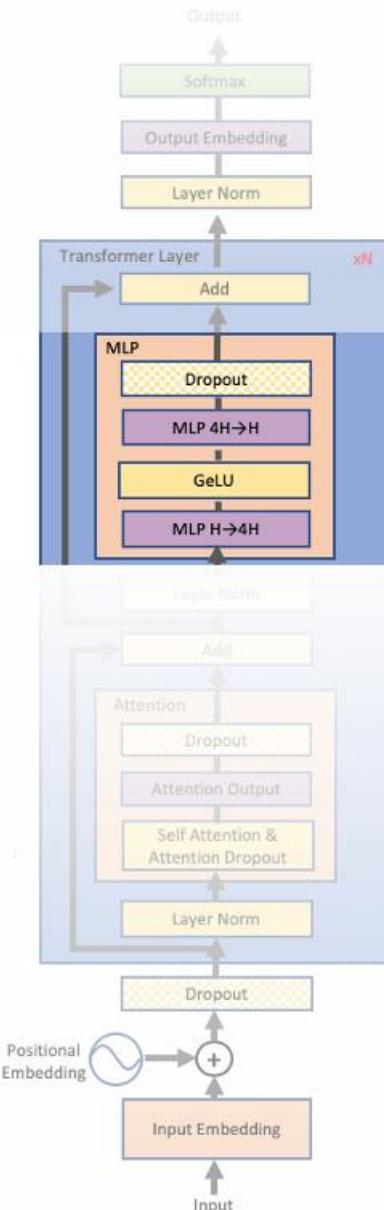
- ▶ No synchronization necessary

Can run in parallel now!

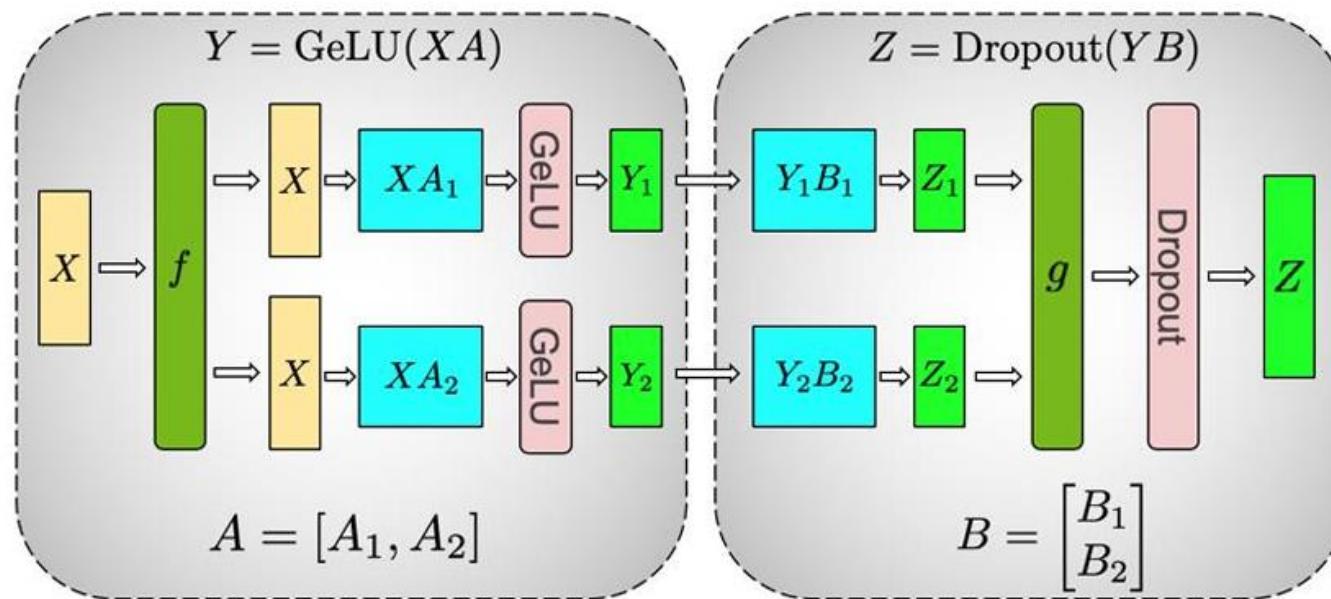
$$\begin{pmatrix} X_0 & X_1 \\ X_2 & X_3 \end{pmatrix} \begin{pmatrix} A_0 & A_1 \\ A_2 & A_3 \end{pmatrix} = \left(\begin{pmatrix} X_0 & X_1 \\ X_2 & X_3 \end{pmatrix} \begin{pmatrix} A_0 \\ A_2 \end{pmatrix}, \begin{pmatrix} X_0 & X_1 \\ X_2 & X_3 \end{pmatrix} \begin{pmatrix} A_1 \\ A_3 \end{pmatrix} \right)$$

$$= (XA_A \quad XA_B)$$

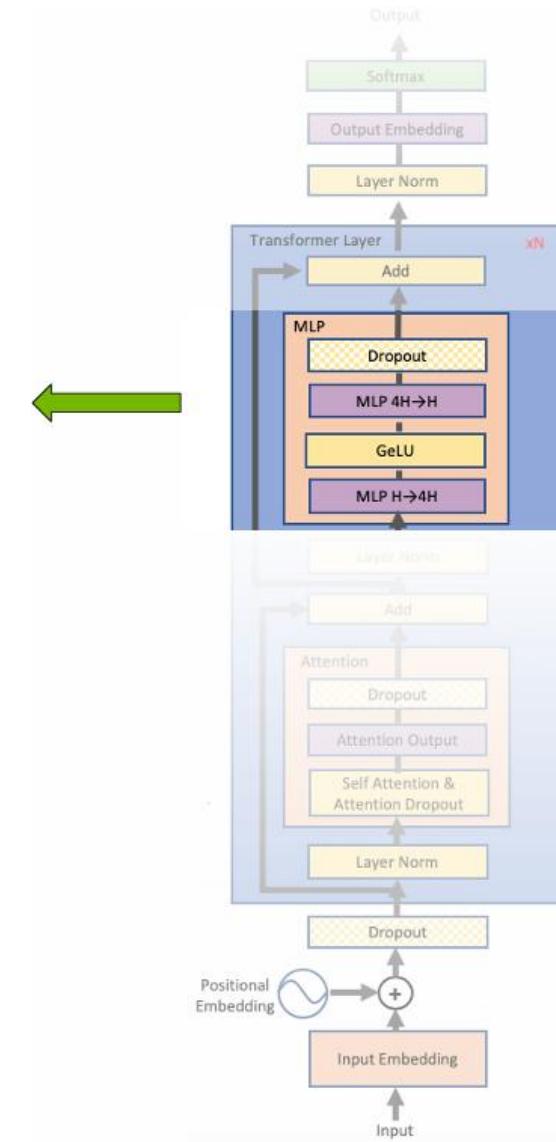
Concatenation



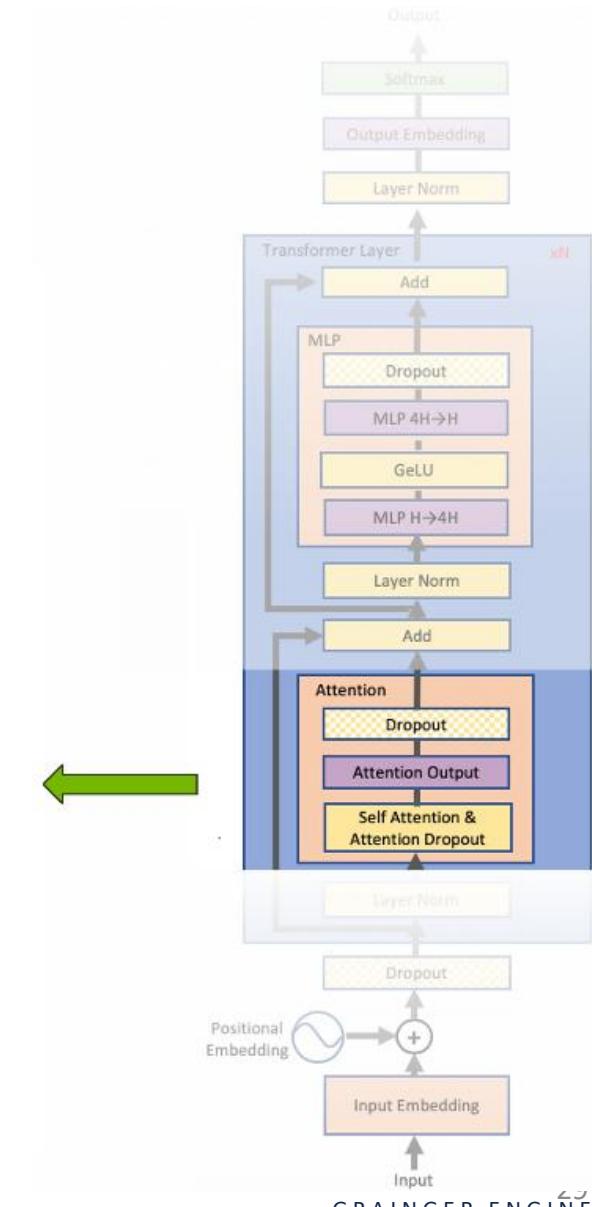
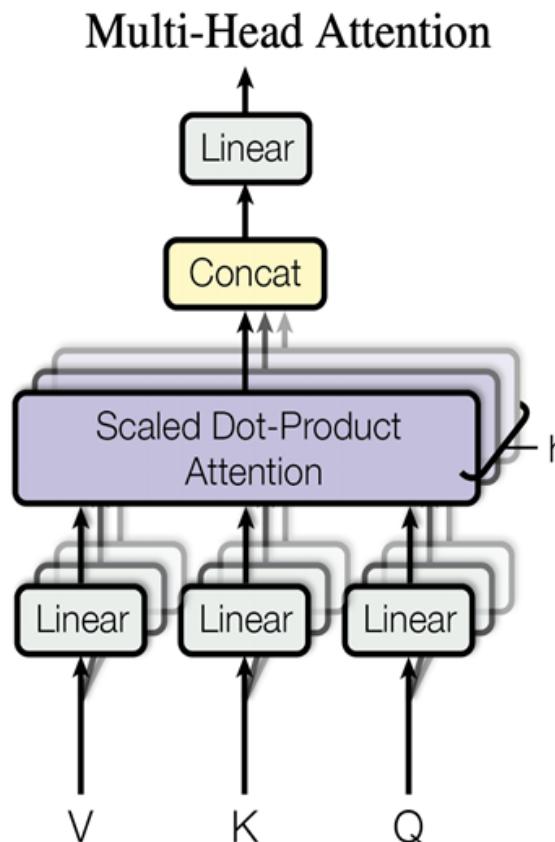
Tensor Parallelism: MLP



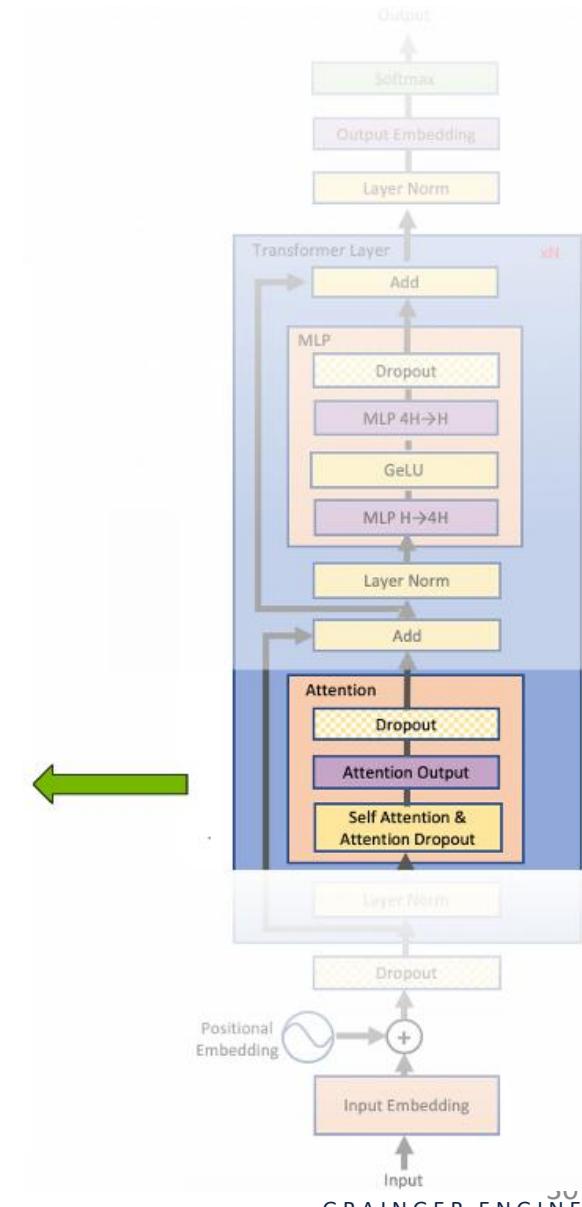
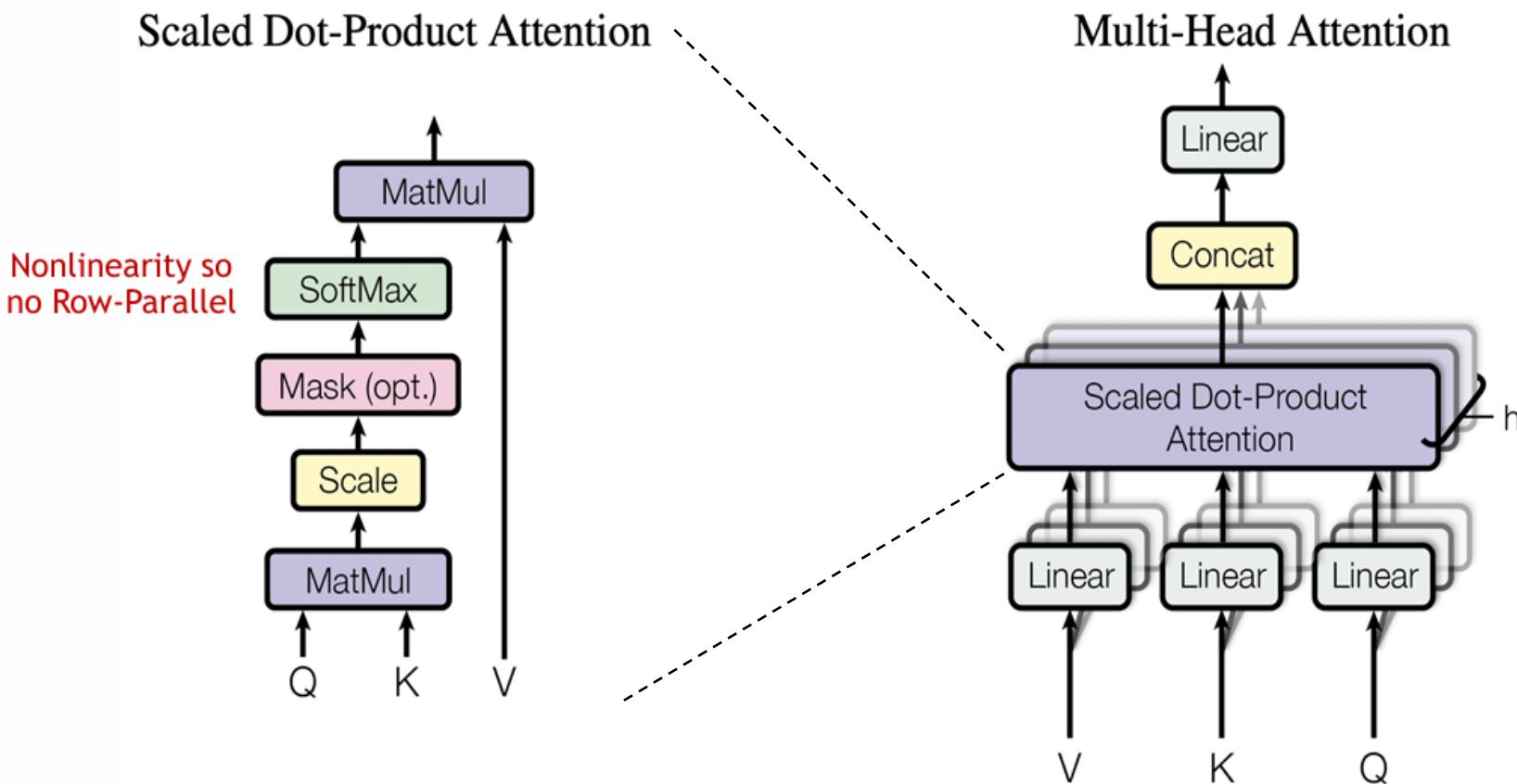
f and **g** are **conjugate**, **f** is **identity** operator in the forward pass and **all-reduce** in the backward pass while **g** is **all-reduce** in forward and **identity** in backward.



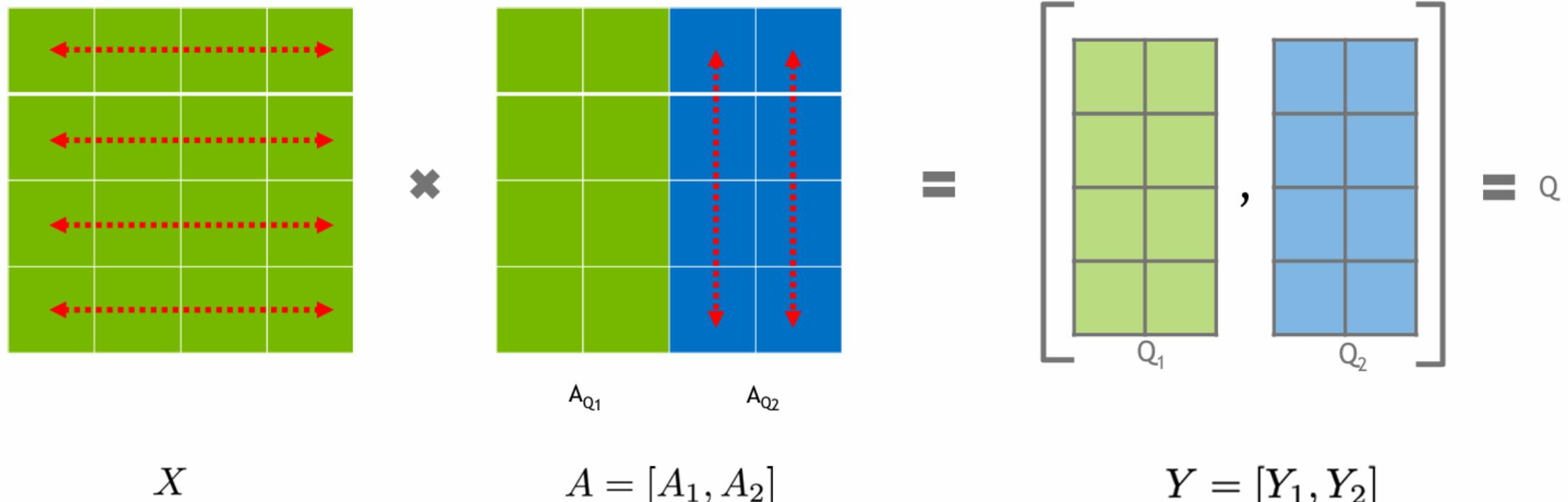
Tensor Parallelism: Self-Attention



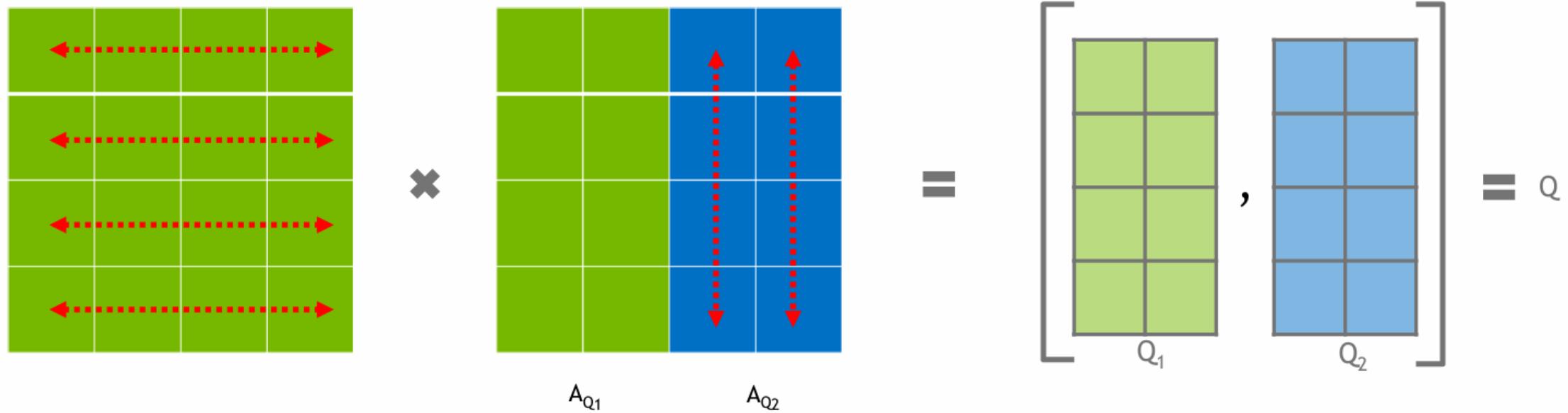
Tensor Parallelism: Self-Attention



Tensor Parallelism: Self-Attention



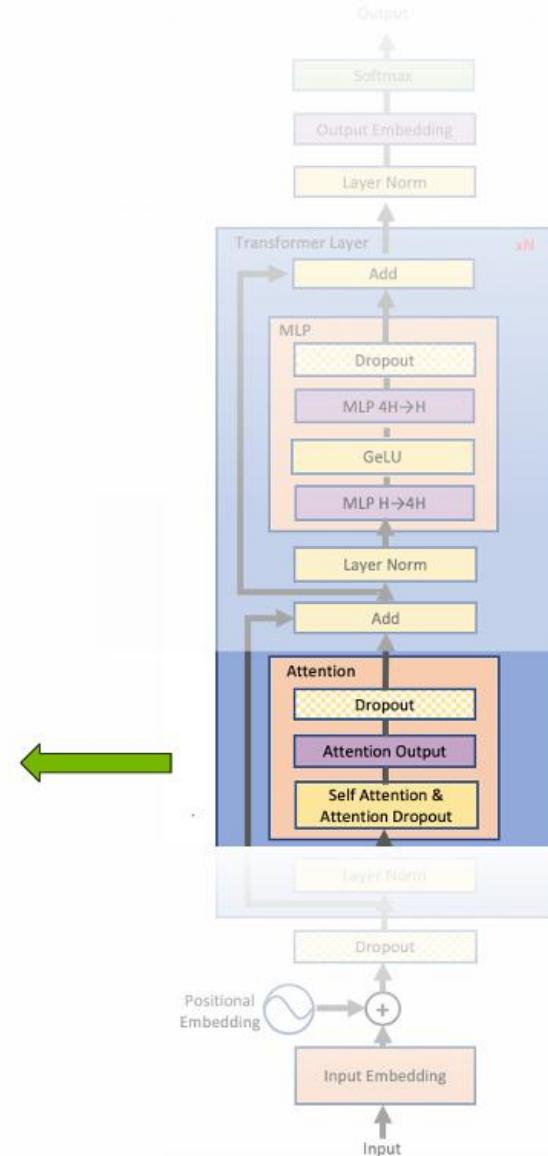
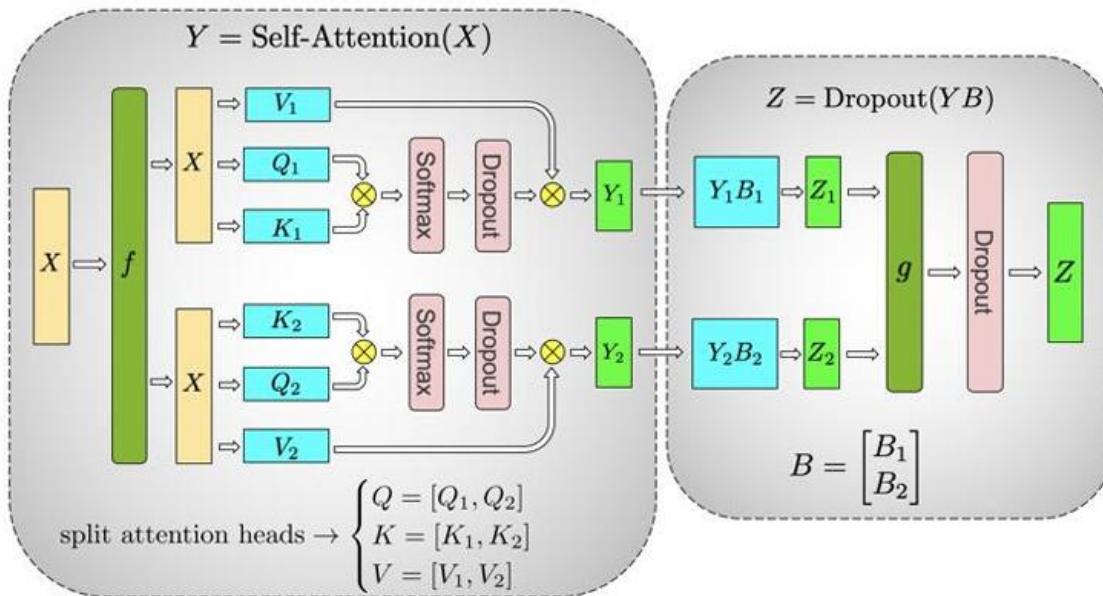
Tensor Parallelism: Self-Attention



Attention heads can be parallelized with Column Parallel GEMMs (ex. Query head 1 (Q_1) and Query head 2 (Q_2))

Question: What if we have more GPUs than the number of heads?

Tensor Parallelism: Self-Attention

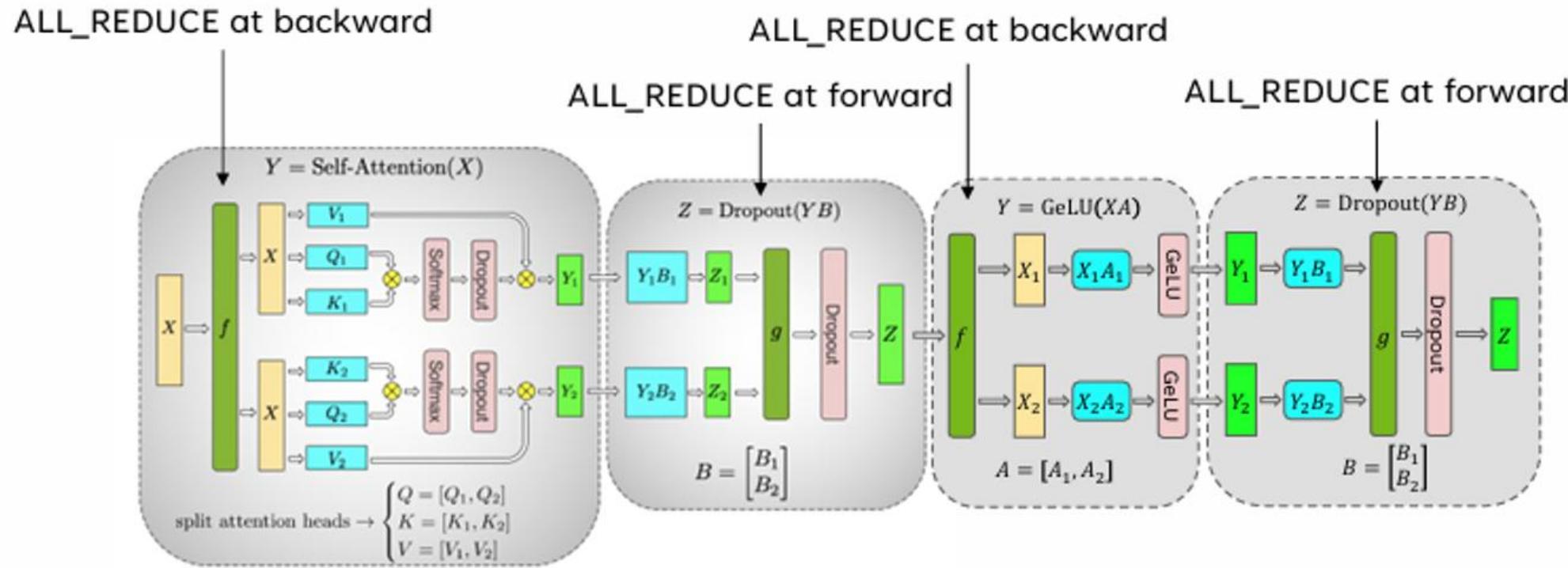


f and g are **conjugate**, f is **identity** operator in the forward pass and **all-reduce** in the backward pass while g is **all-reduce** in forward and **identity** in backward.

Tensor Parallelism: Communication Cost



- 4 total communication operations in 1 forward + backward pass



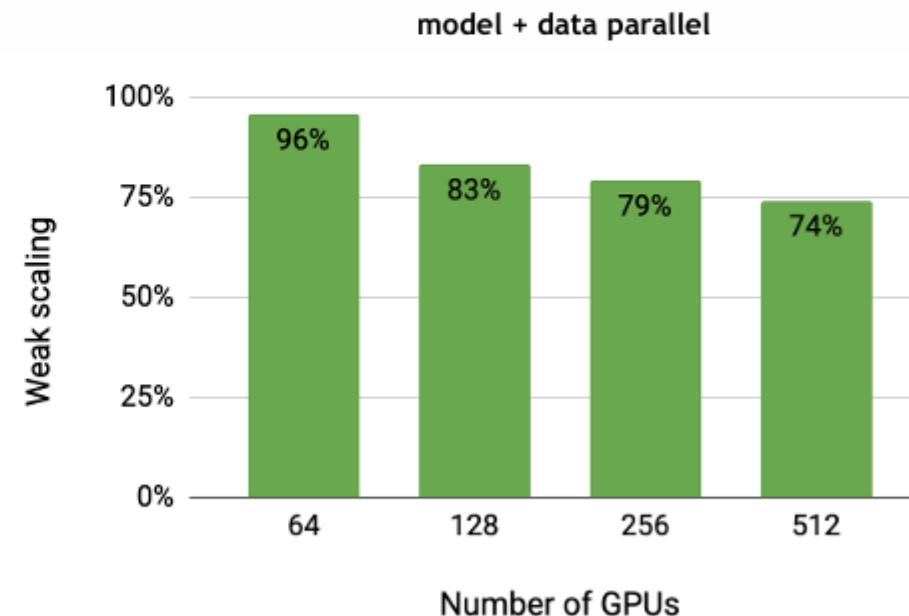
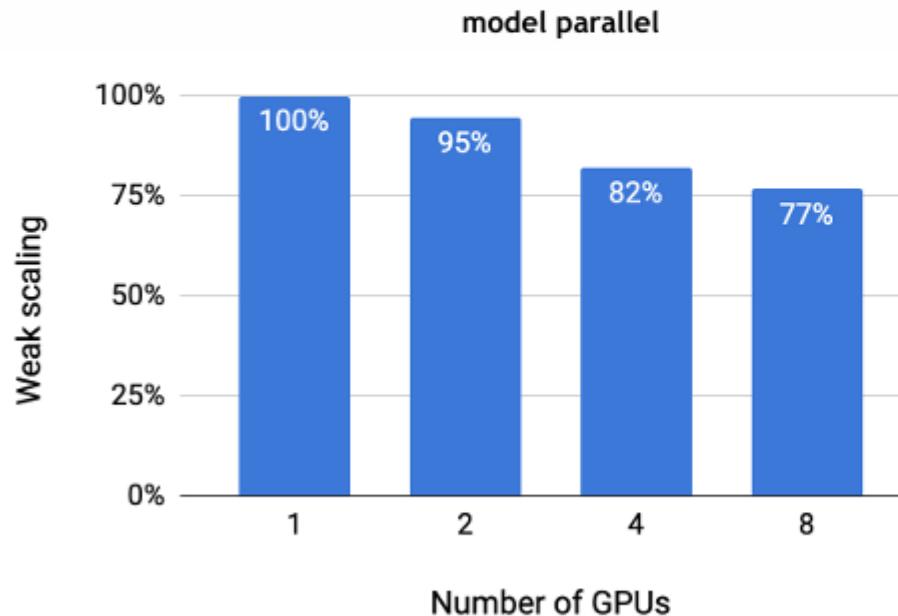
- 32 DGX-2H servers
 - 512 V100 SXM3 32GB GPUs
- Intra-server connection
 - 300 GB/s NVSwitch
- Inter-server connection
 - 100 GB/s InfiniBand (8 per server)
- GPT-2 & BERT Models
- TP (+ DP)
- Mixture of datasets

Experiments: Scalability



- GPT-2: 1B-8B
- Hidden size: 1536-3072
- Parameters/GPU: $\sim 1.B$
- Weak scaling@512 GPUs: 74%

Config	Hidden size	Attention heads	Number of layers	Number of parameters (billions)	Model parallel GPUs	Model+data parallel GPUs
1	1536	16	40	1.2	1	64
2	1920	20	54	2.5	2	128
3	2304	24	64	4.2	4	256
4	3072	32	72	8.3	8	512

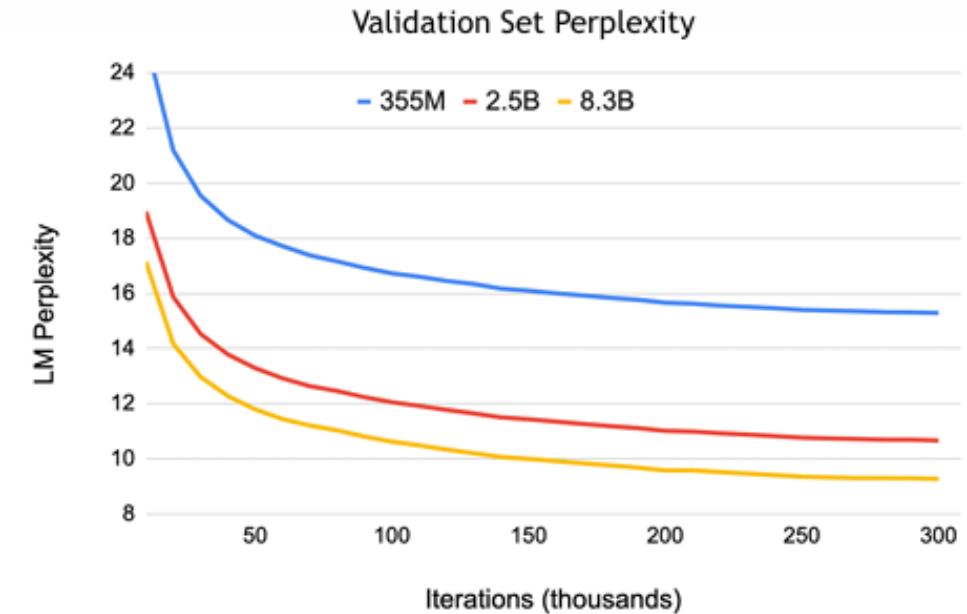


Baseline (1.2B parameters on a single GPU) achieves **39 TeraFLOPs per second**, i.e. **30% of the theoretical peak** during the entire training process

Experiments: GPT-2



- ▶ Training data: 174 GB WebText/CC-Stories/Wikipedia/RealNews
- ▶ 3 model sizes: 355 million, 2.5 billion, and 8.3 billion
- ▶ Zero-shot evaluation results for Wikitext-103 perplexity and Lambada cloze accuracy

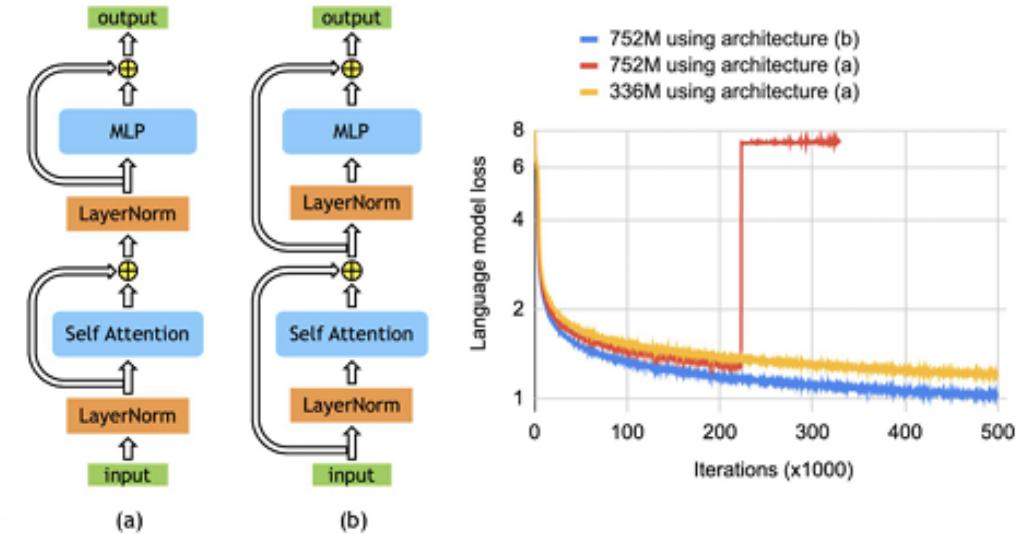


Model Size	Wikitext-103 (Perplexity ↓)	Lambada (Accuracy ↑)
355 M	19.22	46.26
2.5 B	12.68	61.52
8.3 B	10.81	66.51
Previous SOTA	16.43*	63.24**

Experiments: Megatron-BERT



- Scaling BERT to larger sizes is also possible
- Reordering residual connections to stabilize training
- Megatron-BERT 3.9B, 12x larger than BERT-Large, over 2 million iterations @ batch size 1024



Parameter Count	Parameter Multiplier	Hidden Size	Attention Heads	Layers	Model Parallel GPUs	Model + Data Parallel GPUs
334M	1x	1024	16	24	1	128
1.3B	4x	2048	32	24	1	256
3.9B	12x	2506	40	48	4	512

Experiments: SQuAD & RACE



Model	Trained tokens (ratio)	MNLI [†] m/mm accuracy	QQP [†] accuracy	SQuAD 1.1 [†] F1/EM	SQuAD 2.0 [†] F1/EM	RACE m/h [*] accuracy
RoBERTa	2	90.2 / 90.2	92.2	94.6 / 88.9	89.4 / 86.5	86.5 / 81.3
ALBERT	3	90.8	92.2	94.8 / 89.3	90.2 / 87.4	89.0 / 85.5
XLNet	2	90.8 / 90.8	92.3	95.1 / 89.7	90.6 / 87.9	88.6 / 84.0
Megatron-334M	1	89.7 / 90.0	92.3	94.2 / 88.0	88.1 / 84.8	86.9 / 81.5
Megatron-1.3B	1	90.9 / 91.0	92.6	94.9 / 89.1	90.2 / 87.1	90.4 / 86.1
Megatron-3.9B	1	91.4 / 91.4	92.7	95.5 / 90.0	91.2 / 88.5	91.8 / 88.6

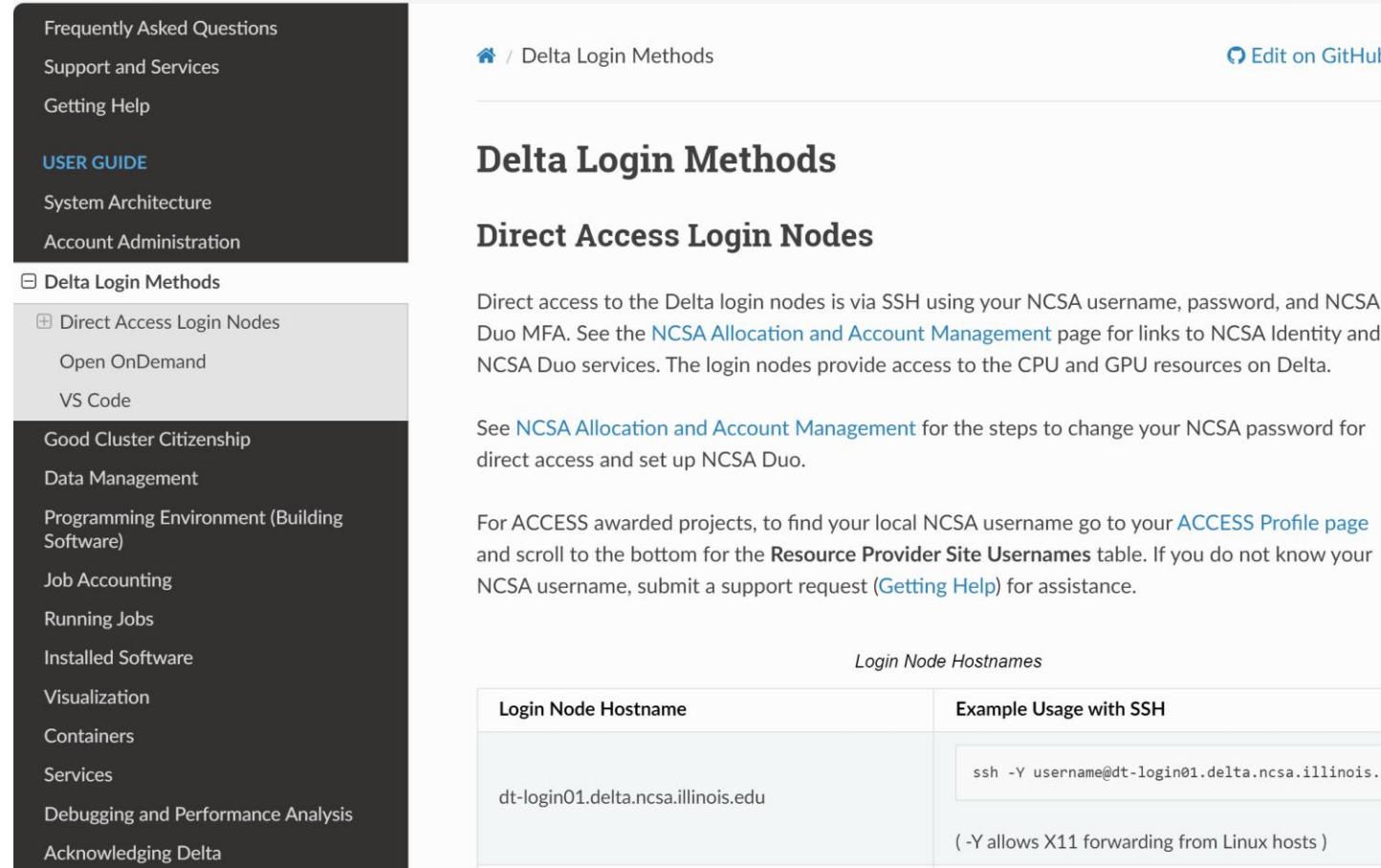
Median single model downstream results on Dev[†] and Test^{*} sets.
State of the art results are bolded.

Questions?

- 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



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



The screenshot shows a navigation sidebar on the left and a main content area on the right. The sidebar includes links for Frequently Asked Questions, Support and Services, Getting Help, and sections for USER GUIDE, System Architecture, and Account Administration. The main content area shows the title 'Delta Login Methods' and a sub-section 'Direct Access Login Nodes'. It explains that direct access is via SSH using NCSA credentials and NCSA Duo MFA, with a link to 'NCSA Allocation and Account Management'. It also provides instructions for ACCESS awarded projects to find their local NCSA username. A table titled 'Login Node Hostnames' is shown, with a single entry for 'dt-login01.delta.ncsa.illinois.edu' and an example SSH command: 'ssh -Y username@dt-login01.delta.ncsa.illinois.edu'. A note at the bottom of the table says '(-Y allows X11 forwarding from Linux hosts)'.

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

Home / 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	<pre>ssh -Y username@dt-login01.delta.ncsa.illinois.edu</pre> <p>(-Y allows X11 forwarding from Linux hosts)</p>

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.

The screenshot shows the homepage of the ACCESS Allocations website. The top navigation bar includes links for ALLOCATIONS, SUPPORT, OPERATIONS, and METRICS, along with icons for home, search, and login. The main header features the ACCESS logo with the tagline 'Allocations'. Below the header, a navigation bar offers links to Home, Get Started, Available Resources, ACCESS Impact, Policies & How-To, and About. The main content area asks 'Need access to computing, data analysis, or storage resources?' and encourages users to log in to get started. Three callout boxes provide information: 'What is an allocation?', 'Which resources?', and 'Ready to get started?'. The 'Ready to get started?' box includes a 'LOGIN' button.

ALLOCATIONS SUPPORT OPERATIONS METRICS

ACCESS Allocations

Home Get Started Available Resources ACCESS Impact Policies & How-To About

Need access to computing, data analysis, or storage resources?

You're in the right place! Read more below, or [login](#) to get started.

What is an allocation?

To get started, you need an ACCESS project and some resource units you can spend. **Your ACCESS project and resource units are what we refer to as an Allocation.** An allocation is your project to use a

Which resources?

We've got modeling and analysis systems, GPU-oriented systems, large-memory nodes, storage, and more. Resource providers have designed their systems to serve a wide range of research and education needs — including

Ready to get started?

It costs you nothing (really!), and you don't need an NSF award. To begin, you just need to

LOGIN

- 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 \(schedmd.com\)](#).

- 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	<pre>ssh -Y username@dt-login01.delta.ncsa.illinois.edu</pre> <p>(-Y allows X11 forwarding from Linux hosts)</p>
dt-login02.delta.ncsa.illinois.edu	<pre>ssh -l username dt-login02.delta.ncsa.illinois.edu</pre> <p>(-l username alt. syntax for <code>user@host</code>)</p>
login.delta.ncsa.illinois.edu	<pre>ssh username@login.delta.ncsa.illinois.edu</pre> <p>(round robin DNS name for the set of login nodes)</p>

- 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.

Course Project (Reproducibility Challenge)



4-credit undergraduate students, 3-credit graduate students

Some suggestions below, but it can be any machine learning system related papers that you are interested in

GPTCache: An Open-Source Semantic Cache for LLM Applications Enabling Faster Answers and Cost Savings	https://github.com/zilliztech/GPTCache
RouteLLM: Learning to Route LLMs with Preference Data	https://github.com/lm-sys/RouteLLM
LLM-QAT: Data-Free Quantization Aware Training for Large Language Models	https://github.com/facebookresearch/LLM-QAT
Speculative decoding in vLLM	https://docs.vllm.ai/en/v0.5.5/models/spec_decode.html
REST: Retrieval-Based Speculative Decoding	https://github.com/FasterDecoding/REST
MemGPT: Towards LLMs as Operating Systems	https://github.com/letta-ai/letta
...	...

4-credit graduate students

- Benchmark and analyze important DL workloads to understand their performance gap and identify important angles to optimize their performance.
- Apply and evaluate how existing solutions work in the context of emerging AI/DL workloads.
- Design and implement new algorithms that are both theoretically and practically efficient.
- Design and implement system optimizations, e.g., parallelism, cache-locality, IO-efficiency, to improve the compute/memory/communication efficiency of AI/DL workloads.
- Offer customized optimization for critical DL workloads where latency is extremely tight.
- Build library/tool/framework to improve the efficiency of a class of problems.
- Integrate important optimizations into existing frameworks (e.g., DeepSpeed), providing fast and agile inference.
- Combine system optimization with modeling optimizations.
- Combine and leverage hardware resources (e.g., GPU/CPU, on-device memory/DRAM/NVMe/SSD) in a principled way.