THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY | DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

RELAXED SYSTEM LAB

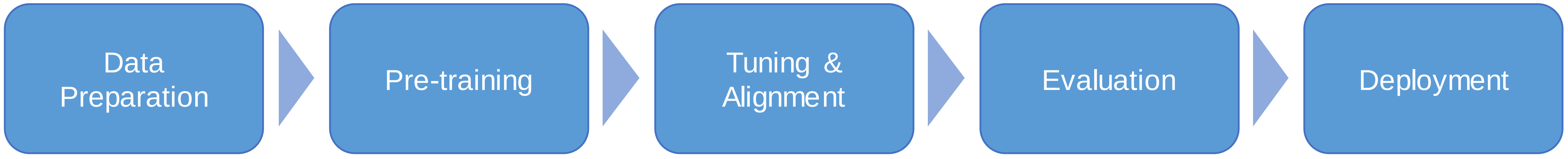
Final Review

COMP4901Y

Binhang Yuan

The Path Towards a Foundation Model

RELAXED SYSTEM LAB



Data Preparation

Preparing a massive corpus for pretraining

A multi-stage pipeline:
- Acquisition (e.g. OCR)
- Cleaning;
- Filtering;
- De-duplication;
- Sampling;
- Shuffling.

Pre-training

Training a foundation model by fitting the large corpus by auto-regression

Core infra:
a large-scale parallel training system running over thousands of GPUs

Tuning & Alignment

Tune the model so it can follow user instructions effectively and safely.

Various methods:

- SFT
- RLHF
- RLAIF

Evaluation

Evaluate the model from different aspects:

- Comprehension
- Instruction following
- Expression
- Reasoning
- Coding
- Interaction
- Safety

Deployment

Deploy the model to serve users under different settings:
- Cloud
- Local devices

Require system efforts to optimize the performance, cost, and latency.

<u>**Covered by this course!**</u>

What We Have Explored.

Date	Topic
W1 - 01/31	Introduction and Logistics
W2 - 02/05, 02/07	Machine Learning Preliminary & PyTorch Tensors
W3 - ~~02/12~~ (Lunar New Year), 02/14	Stochastic Gradient Descent
W4 - 02/19, 02/21	Auto-Differentiation & PyTorch Autograd
W5 - 02/26, 02/28	Nvidia GPU Performance & Collective Communication Library
W6 - 03/04, 03/06	Transformer Architecture & Large Scale Pretrain Overview
W7 - 03/11, 03/13	Data Parallel Training & Pipeline Parallel Training
W8 - 03/18, 03/20	Tensor Model Parallel Training & Optimizer Parallel Training

What We Have Explored.

RELAXED
SYSTEM LAB

Date	Topic
W10 - 04/08, 04/10	Generative Inference Workflow & Hugging Face Library
W11 - 04/15, 04/17	Generative Inference Optimization & Speculative Decoding
W12 - 04/22, 04/24	Prompt Engineering Overview & Practices
W13 - 04/29	Parameter Efficient Fine-tuning (LoRA)
W14 - 05/06, 05/08	Final Eaxme Review

RELAXED
SYSTEM LAB

ML System Basics

ML System Basics

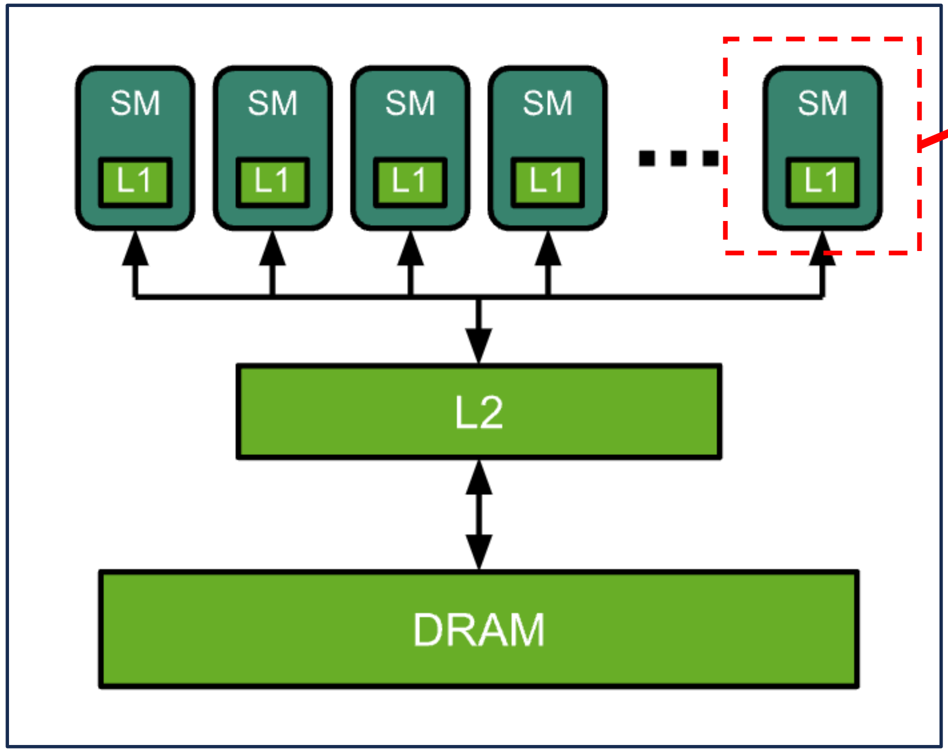
- Machine learning preliminary & PyTorch tensors:
    - Einstein notation in PyTorch.
- Stochastic gradient descent:
    - Define the empirical risk;
    - Optimizing the empirical loss.
- Automatic differentiation:
    - Forward mode AD;
    - Reverse Mode AD;
    - Compute the gradient of a Linear Layer.

Summary of a Linear Layer Computation

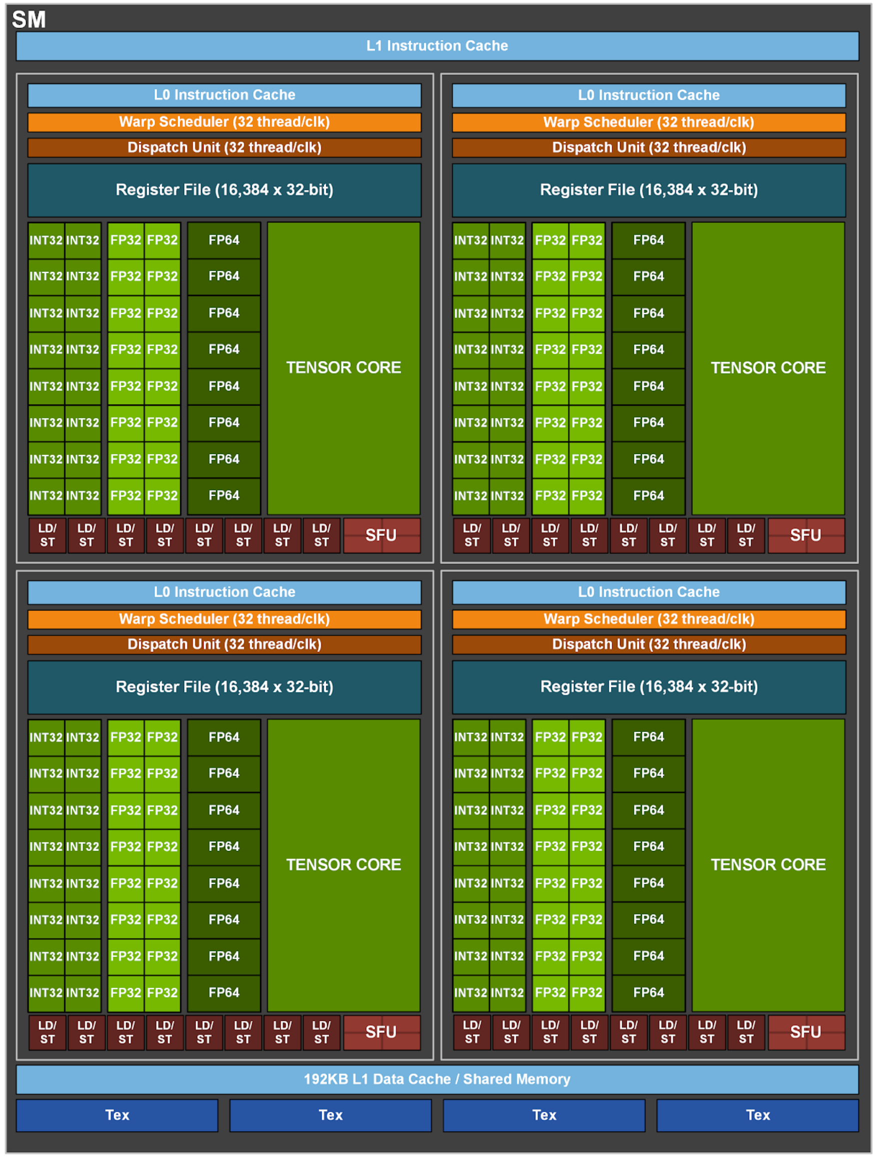
- Forward computation of a linear layer: $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{W}$
  - Given input: $\boldsymbol{X} \in \mathbb{R}^{B \times D_1}$
  - Given weight matrix: $\boldsymbol{W} \in \mathbb{R}^{D_1 \times D_2}$
  - Compute output: $\boldsymbol{Y} \in \mathbb{R}^{B \times D_2}$
- Backward computation of a linear layer:
  - Given gradients w.r.t output: $\frac{\partial L}{\partial \boldsymbol{Y}} \in \mathbb{R}^{B \times H_2}$
  - Compute gradients w.r.t weight matrix: $\frac{\partial L}{\partial \boldsymbol{W}} = \boldsymbol{X}^{\boldsymbol{T}} \frac{\partial L}{\partial \boldsymbol{Y}} \in \mathbb{R}^{B \times H_2}$
  - Compute gradients w.r.t input: $\frac{\partial L}{\partial \boldsymbol{X}} = \frac{\partial L}{\partial \boldsymbol{Y}} \boldsymbol{W}^{T} \in \mathbb{R}^{B \times H_2}$

GPU Computation and Communication

Ampere GPU Architecture

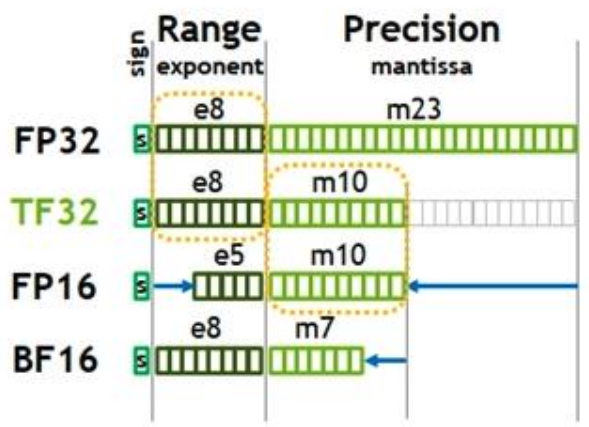


108 SM in a A100 GPU


SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
INT32
FP32
FP64
TENSOR CORE
LD/ST
SFU
192KB L1 Data Cache / Shared Memory
Tex

A100 GPU Tensor Core Computation

- Multiply-add is the most frequent operation in modern neural networks. This is known as the fused multiply-add (FMA) operation.
- Includes one multiply operation and one add operation, counted as two float operations.
- A100 GPU has 1.41 GHz clock rate.
- The Ampere A100 GPU Tensor Cores multiply-add operations per clock:



Ampere A100 GPU FMA per clock on a SM					
FP64	TF32	FP16	INT8	INT4	INT1
64	512	1024	2048	4096	16384

Arithmetic Intensity

- Thus, on a given processor a given algorithm is math limited if:
  - $T_{math} > T_{mem}$
  - $\frac{\#op}{BW_{math}} > \frac{\#bytes}{BW_{mem}}$

- By simple algebra, the inequality can be rearranged to:
  - $\frac{\#op}{\#bytes} > \frac{BW_{math}}{BW_{mem}}$

- The left-hand side: the algorithm's *arithmetic intensity*.
- The right-hand side: *ops:byte ratio*.

NCCL Operators

- Optimized collective communication library from Nvidia to enable high-performance communication between CUDA devices.
- NCCL Implements:
  - **AllReduce**;
  - **Broadcast;**
  - **Reduce;**
  - **AllGather;**
  - **Scatter;**
  - **Gather;**
  - **ReduceScatter;**
  - **AlltoAll.**
- Easy to integrate into DL framework (e.g., PyTorch).

Ring based AllReduce

- In ring based AllReduce, we assume:
  - N: bytes to aggregate
  - B: bandwidth of each link
  - k: number of GPUs
  - The original tensor is equally split into k chunks.
- Ring based AllReduce implementation has two phases:
  - Reduction phrase (Aggregation phrase);
  - AllGather Phrase.
  - Total time: $\frac{2(k-1)N}{kB}$.

Language Model

Autoregressive Language Models

- A common way to write the joint distribution $p(x_{1:L})$ of a sequence to $x_{1:L}$ is using the *chain rule of probablity*:

$$p(x_{1:L}) = p(x_1)p(x_2|x_1)p(x_3|x_1,x_2)\dots p(x_L|x_{1:L-1}) = \prod_{i=1}^{L} p(x_i|x_{1:i-1})$$

- In particular, $p(x_i|x_{1:i-1})$ is a conditional probability distribution of the next token x_i given the previous tokens $x_{1:i-1}$.
- An autoregressive language model is one where each conditional distribution $p(x_i|x_{1:i-1})$ can be computed efficiently (e.g., using a feedforward neural network).

TransformerBlocks$(x_{1:L} \in R^{L\times D}) \rightarrow \mathbb{R}^{L\times D}$

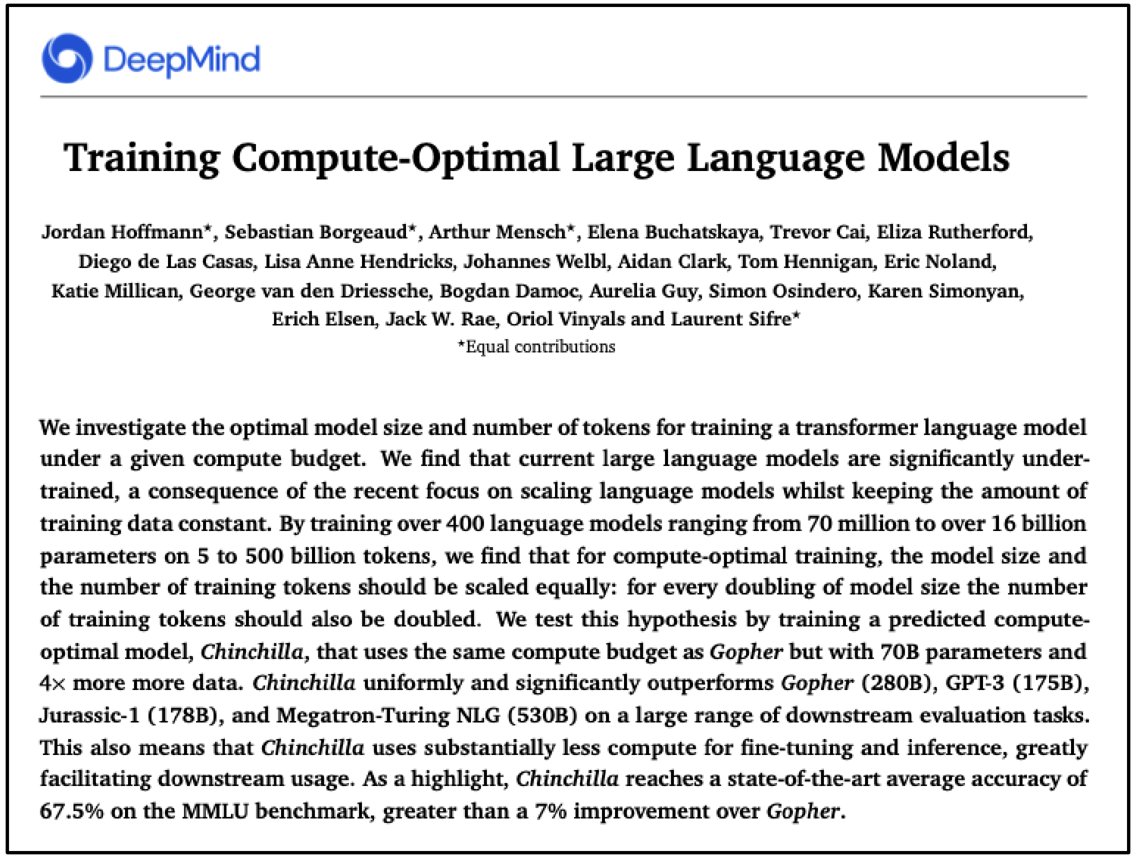
RELAXED SYSTEM LAB

- B is the batch size;
- L is the sequence length;
- D is the model dimension;
- Multi-head attention:
  $D = n_H \times H$
- H is the head dimension;
- n_h is the number of heads.

Computation	Input	Output
$Q = xW^Q$	$x \in \mathbb{R}^{L\times D}$, $W^Q \in \mathbb{R}^{D\times D}$	$Q \in \mathbb{R}^{L\times D}$
$K = xW^K$	$x \in \mathbb{R}^{L\times D}$, $W^K \in \mathbb{R}^{D\times D}$	$K \in \mathbb{R}^{L\times D}$
$V = xW^V$	$x \in \mathbb{R}^{L\times D}$, $W^V \in \mathbb{R}^{D\times D}$	$V \in \mathbb{R}^{L\times D}$
$[Q_1, Q_2 \dots, Q_{n_h}] = \text{Partition}_{-1}(Q)$	$Q \in \mathbb{R}^{L\times D}$	$Q_i \in \mathbb{R}^{L\times H}, i = 1, \dots n_h$
$[K_1, K_2 \dots, K_{n_h}] = \text{Partition}_{-1}(K)$	$K \in \mathbb{R}^{L\times D}$	$K_i \in \mathbb{R}^{L\times H}, i = 1, \dots n_h$
$[V_1, V_2 \dots, V_{n_h}] = \text{Partition}_{-1}(V)$	$V \in \mathbb{R}^{L\times D}$	$V_i \in \mathbb{R}^{L\times H}, i = 1, \dots n_h$
$\text{Score}_\text{i} = \text{softmax}(\frac{Q_i K_i^T}{\sqrt{D}}), i = 1, \dots n_h$	$Q_i, K_i \in \mathbb{R}^{L\times H}$	$\text{score}_\text{i} \in \mathbb{R}^{L\times L}$
$Z_i = \text{score}_i V_i, i = 1, \dots n_h$	$\text{score}_\text{i} \in \mathbb{R}^{L\times L}$, $V_i \in \mathbb{R}^{L\times H}$	$Z_i \in \mathbb{R}^{L\times H}$
$Z = \text{Merge}_{-1}([Z_1, Z_2 \dots, Z_{n_h}])$	$Z_i \in \mathbb{R}^{L\times H}, i = 1, \dots n_h$	$Z \in \mathbb{R}^{L\times D}$
$\text{Out} = ZW^O$	$Z \in \mathbb{R}^{L\times D}$, $W^O \in \mathbb{R}^{D\times D}$	$\text{Out} \in \mathbb{R}^{L\times D}$
$A = \text{Out}\, W^1$	$\text{Out} \in \mathbb{R}^{L\times D}$, $W^1 \in \mathbb{R}^{D\times 4D}$	$A \in \mathbb{R}^{L\times 4D}$
$A' = \text{relu}(A)$	$A \in \mathbb{R}^{L\times 4D}$	$A' \in \mathbb{R}^{L\times 4D}$
$x' = A'W^2$	$A' \in \mathbb{R}^{L\times 4D}$, $W^2 \in \mathbb{R}^{4D\times D}$	$x' \in \mathbb{R}^{L\times D}$

Scaling Law

- Intuitively:
  - Increase parameter # N $\rightarrow$ better performance
  - Increase dataset scale D $\rightarrow$ better performance
- But we have a fixed computational budget on $C \approx 6ND$
- ***To maximize model performance, how should we allocate C to N and D?***



DeepMind

Training Compute-Optimal Large Language Models

Jordan Hoffmann*, Sebastian Borgeaud*, Arthur Mensch*, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals and Laurent Sifre*
*Equal contributions

We investigate the optimal model size and number of tokens for training a transformer language model under a given compute budget. We find that current large language models are significantly undertrained, a consequence of the recent focus on scaling language models whilst keeping the amount of training data constant. By training over 400 language models ranging from 70 million to over 16 billion parameters on 5 to 500 billion tokens, we find that for compute-optimal training, the model size and the number of training tokens should be scaled equally: for every doubling of model size the number of training tokens should also be doubled. We test this hypothesis by training a predicted compute-optimal model, *Chinchilla*, that uses the same compute budget as *Gopher* but with 70B parameters and 4× more more data. *Chinchilla* uniformly and significantly outperforms *Gopher* (280B), GPT-3 (175B), Jurassic-1 (178B), and Megatron-Turing NLG (530B) on a large range of downstream evaluation tasks. This also means that *Chinchilla* uses substantially less compute for fine-tuning and inference, greatly facilitating downstream usage. As a highlight, *Chinchilla* reaches a state-of-the-art average accuracy of 67.5% on the MMLU benchmark, greater than a 7% improvement over *Gopher*.

Evaluating Distributed Computation

- Scaling law tells us given a fixed computation budget, how should we decide the model scale and data corpus.
- The computation budget is formulated by the total FLOPs demanded during the computation.
- But the GPU cannot usually work at its peak FLOPs.
- *How can we evaluate the performance of a distributed training workflow?*
  - Training throughput (token per second);
  - Scalability;
  - Model FLOPs Utilization.

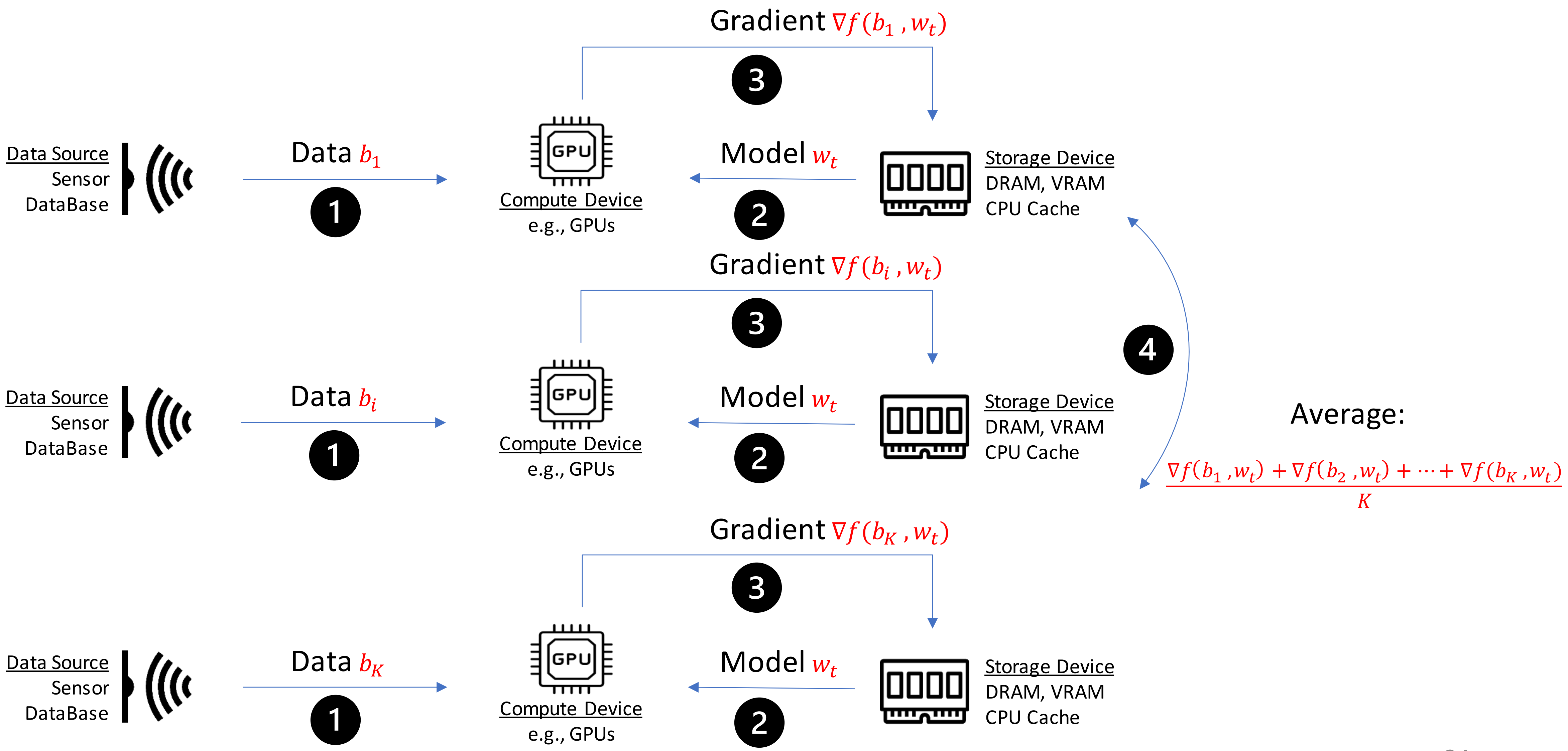
RELAXED
SYSTEM LAB

Parallel Training

Parallel Training

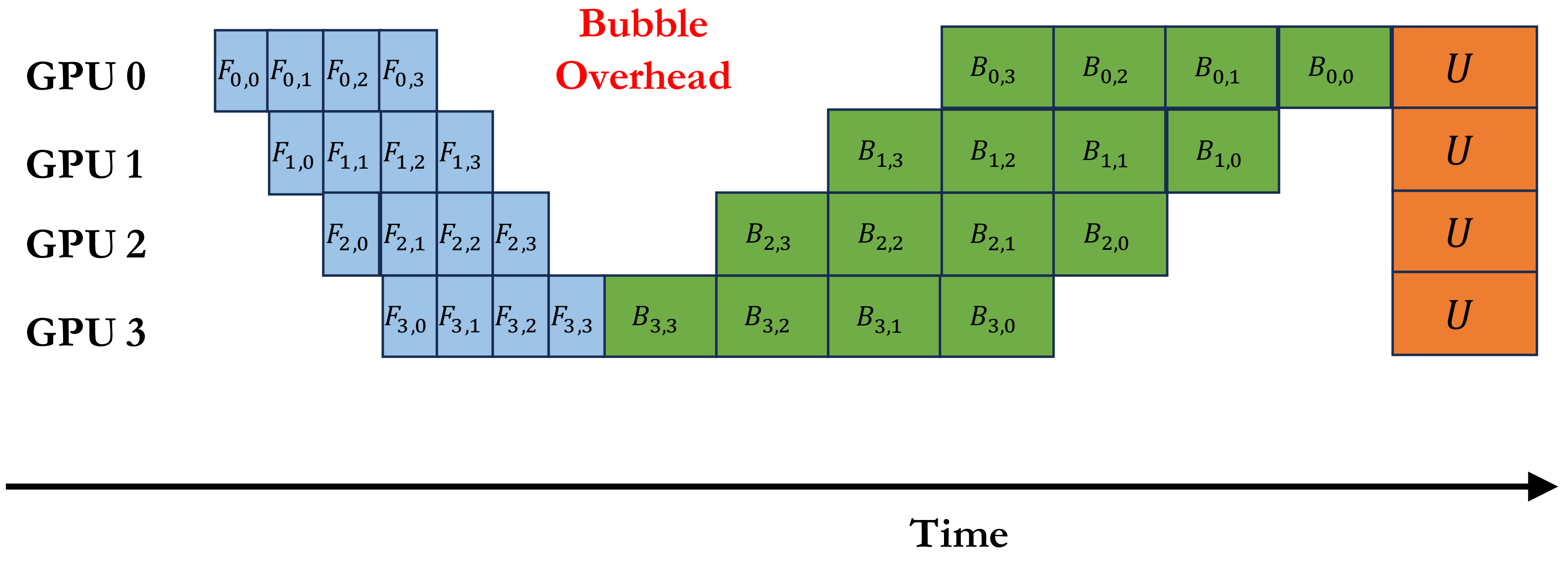
- Categories:
    - Data parallelism;
    - Pipeline parallelism;
    - Tensor model parallelism;
    - Optimizer parallelism.
- What are their communication paradigms?
    - Communication targets;
    - Communication volume.
- What are their advantages and disadvantages?

Data Parallel Training



Pipeline Parallel Training - GPipe

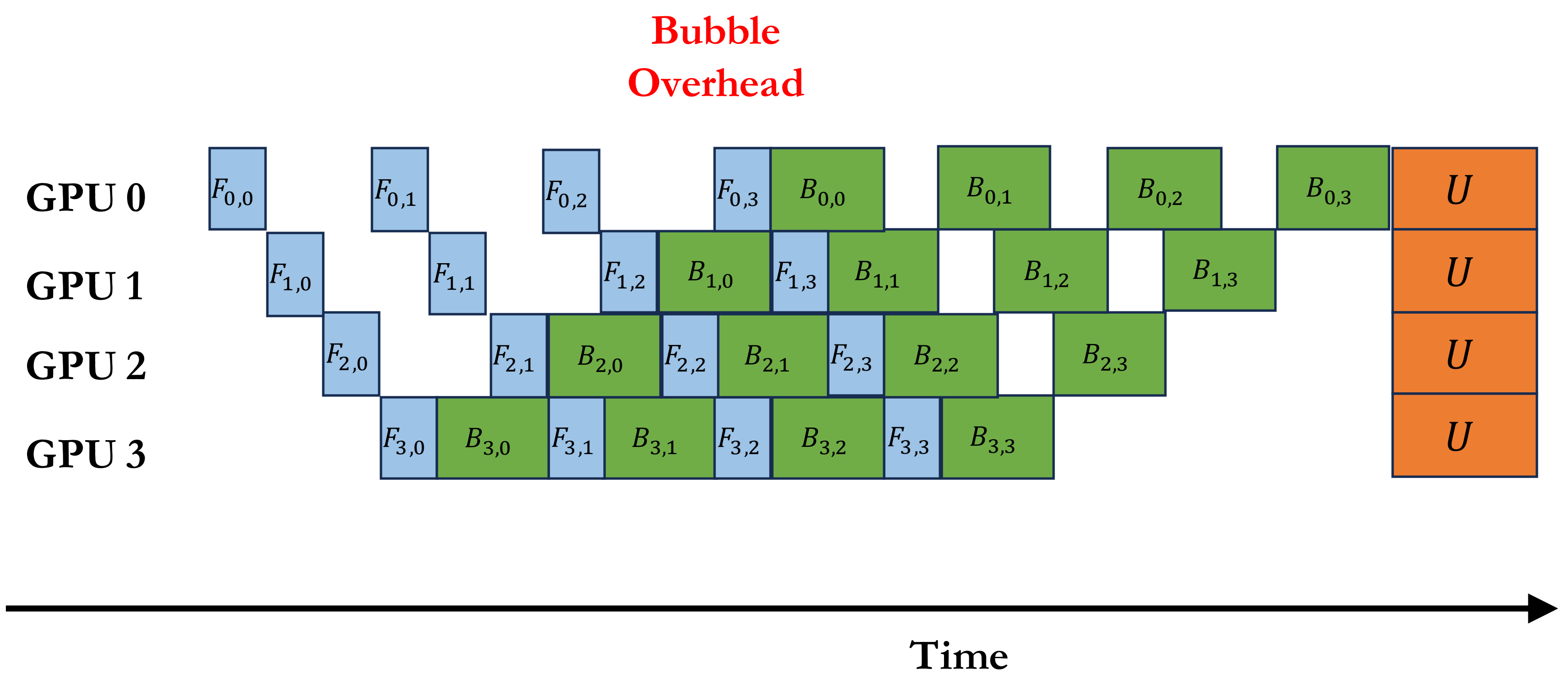
RELAXED SYSTEM LAB


Bubble Overhead
GPU 0
GPU 1
GPU 2
GPU 3
$F_{0,0}$ $F_{0,1}$ $F_{0,2}$ $F_{0,3}$
$F_{1,0}$ $F_{1,1}$ $F_{1,2}$ $F_{1,3}$
$F_{2,0}$ $F_{2,1}$ $F_{2,2}$ $F_{2,3}$
$F_{3,0}$ $F_{3,1}$ $F_{3,2}$ $F_{3,3}$
$B_{3,3}$ $B_{3,2}$ $B_{3,1}$ $B_{3,0}$
$B_{2,3}$ $B_{2,2}$ $B_{2,1}$ $B_{2,0}$
$B_{1,3}$ $B_{1,2}$ $B_{1,1}$ $B_{1,0}$
$B_{0,3}$ $B_{0,2}$ $B_{0,1}$ $B_{0,0}$
U
U
U
U
Time

The number on each block represents the stage index and the micro-batch index.

- If we ignore the computation time of optimizer updates.
- Suppose:
  - K is the number of GPUs;
  - M is the number of micro-batches;
- What is the percentage of bubble overhead? $\frac{K-1}{M+K-1}$

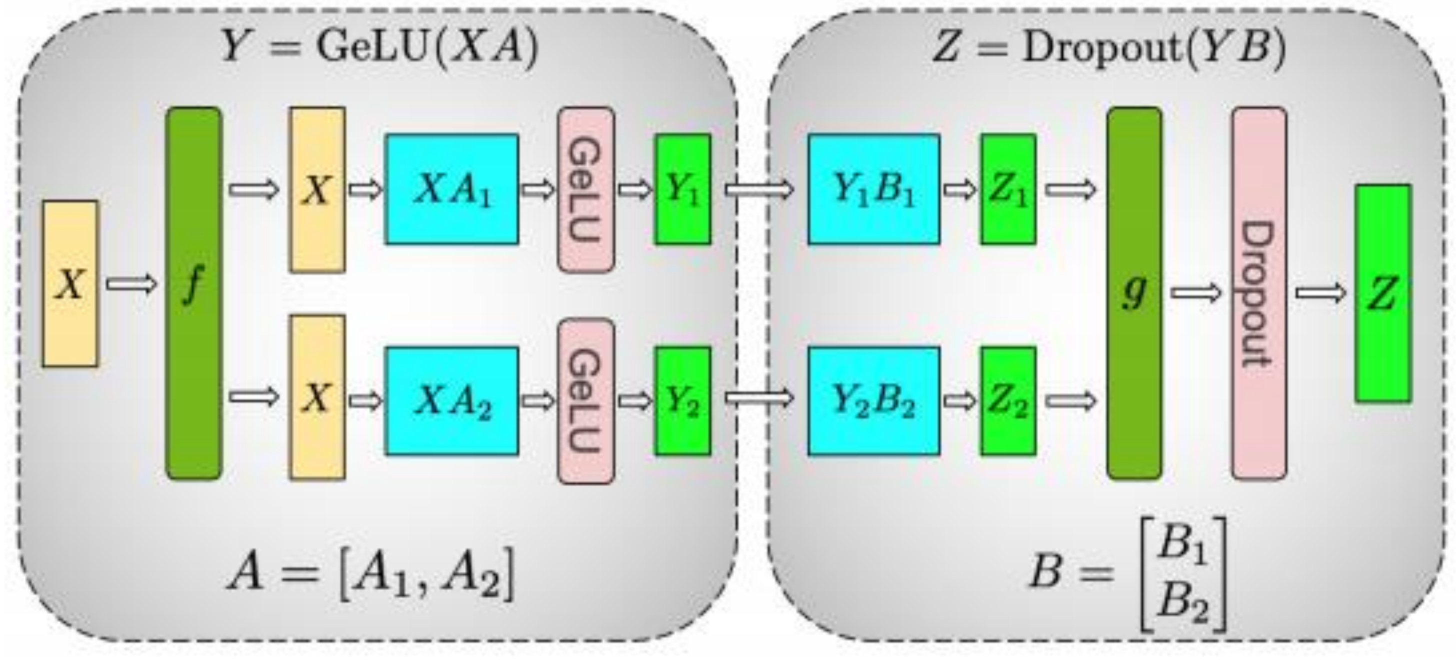
Pipeline Parallel Training - 1F1B


Bubble Overhead
GPU 0
GPU 1
GPU 2
GPU 3
$F_{0,0}$ $F_{0,1}$ $F_{0,2}$ $F_{0,3}$ $B_{0,0}$ $B_{0,1}$ $B_{0,2}$ $B_{0,3}$ U
$F_{1,0}$ $F_{1,1}$ $F_{1,2}$ $B_{1,0}$ $F_{1,3}$ $B_{1,1}$ $B_{1,2}$ $B_{1,3}$ U
$F_{2,0}$ $F_{2,1}$ $B_{2,0}$ $F_{2,2}$ $B_{2,1}$ $F_{2,3}$ $B_{2,2}$ $B_{2,3}$ U
$F_{3,0}$ $B_{3,0}$ $F_{3,1}$ $B_{3,1}$ $F_{3,2}$ $B_{3,2}$ $F_{3,3}$ $B_{3,3}$ U
Time

The number on each block represents the stage index and the micro-batch index.

- If we ignore the computation time of optimizer updates.
- Suppose:
  - K is the number of GPUs;
  - M is the number of micro-batches;
- What is the percentage of bubble overhead? $\frac{K-1}{M+K-1}$

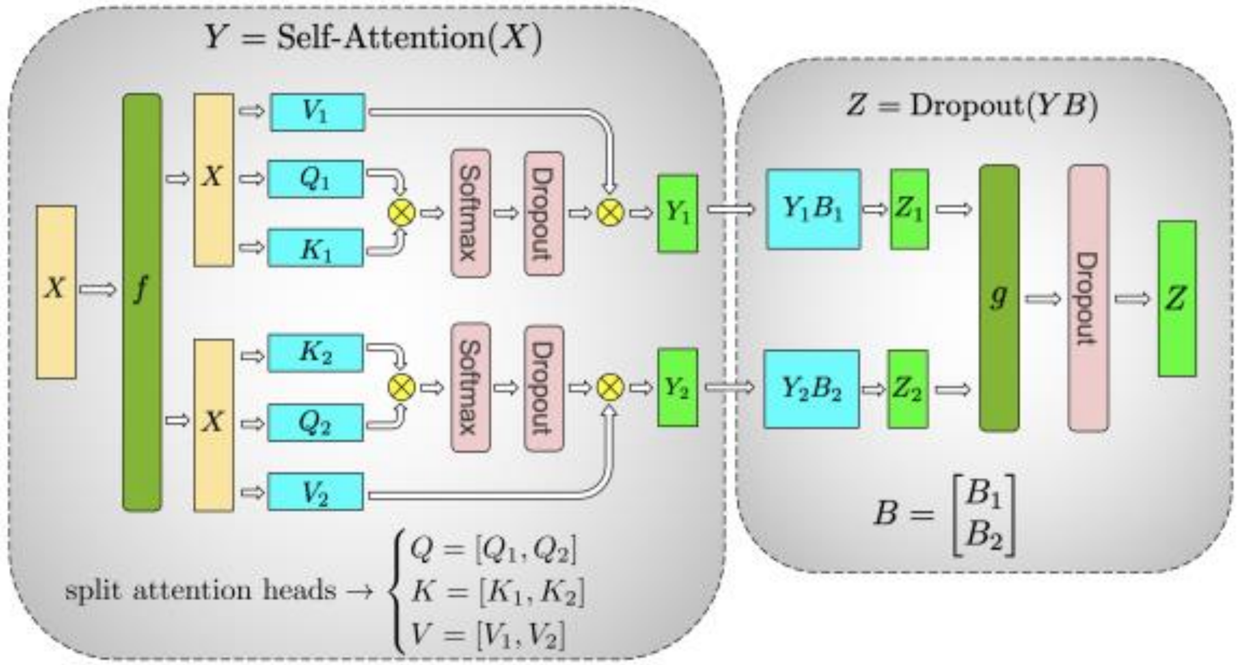
Tensor Model Parallelism - MLP



(a) MLP

- f is the identity operator in the forward pass and the AllReduce operator in the backward pass.
- g is the AllReduce operator in the forward pass and the identity operator in the backward pass.

Tensor Model Parallelism - Multi-Head Attention

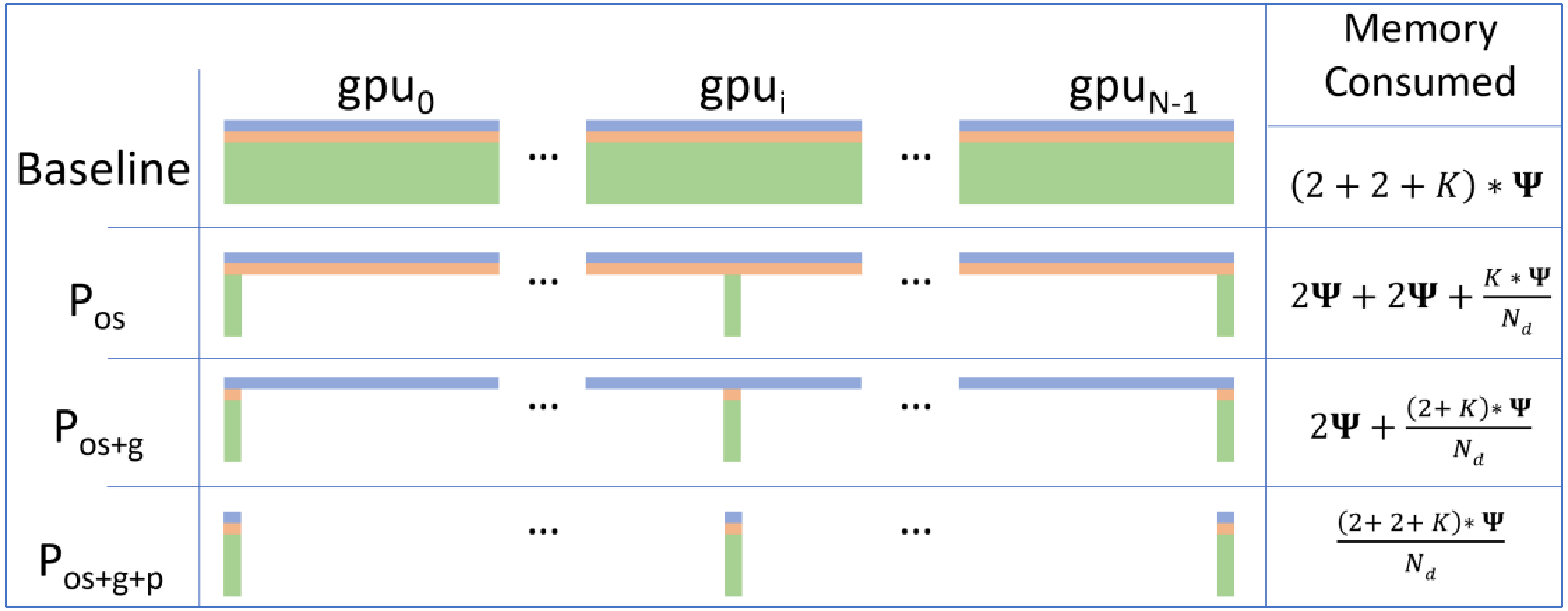


$Y = \text{Self-Attention}(X)$
X
f
X
V_1
Q_1
K_1
Softmax
Dropout
Y_1
X
K_2
Q_2
V_2
Softmax
Dropout
Y_2
split attention heads $\rightarrow$ $Q = [Q_1, Q_2]$, $K = [K_1, K_2]$, $V = [V_1, V_2]$
$Z = \text{Dropout}(YB)$
Y_1B_1
Z_1
Y_2B_2
Z_2
g
Dropout
Z
$B = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$

(b) Self-Attention

- f is the identity operator in the forward pass and the AllReduce operator in the backward pass.
- g is the AllReduce operator in the forward pass and the identity operator in the backward pass.

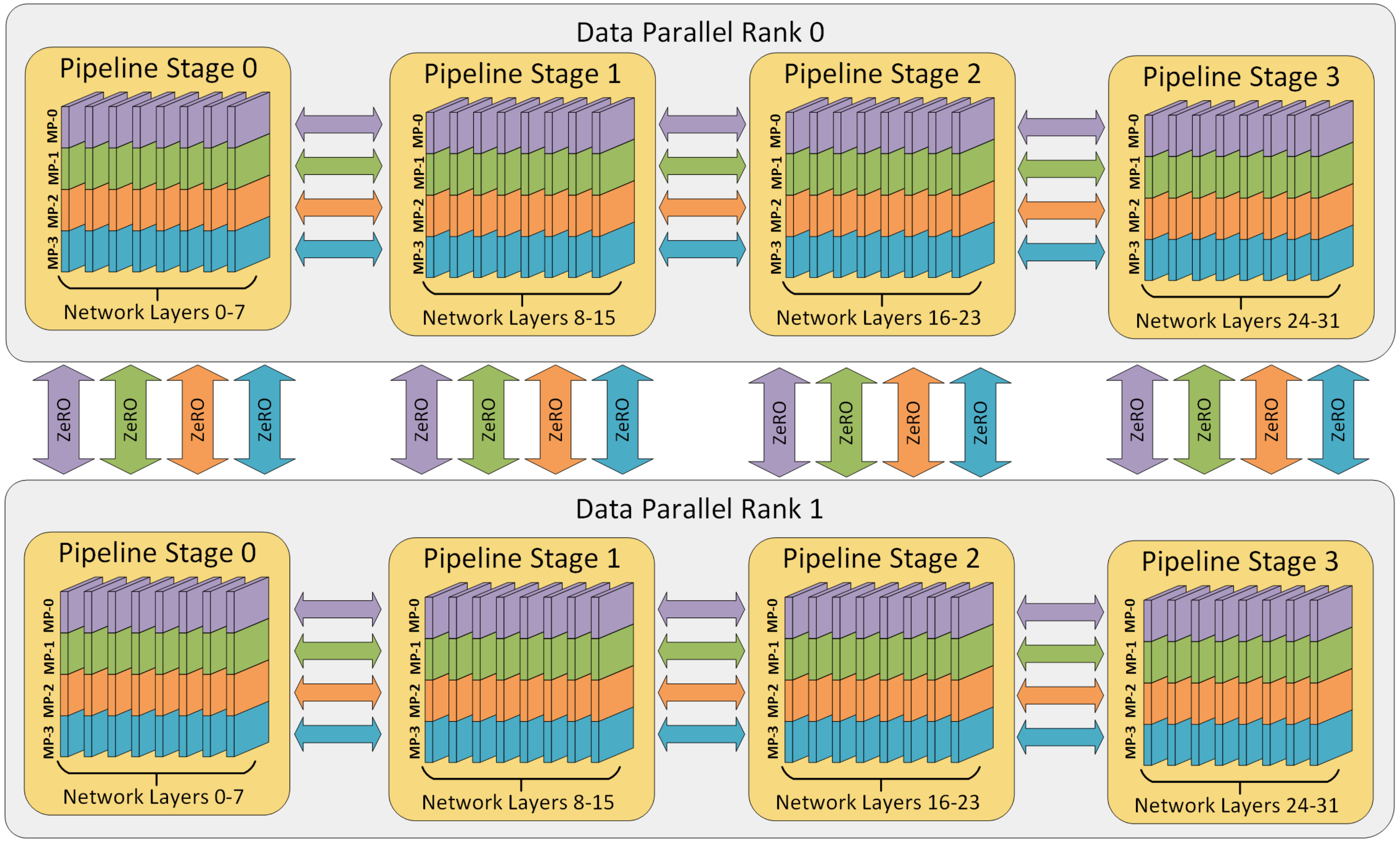
Zero Redundancy Optimizer (ZeRO)



	gpu_0		gpu_i		gpu_{N-1}	Memory Consumed
Baseline		…		…		$(2+2+K)*\Psi$
P_{os}		…		…		$2\Psi + 2\Psi + \frac{K*\Psi}{N_d}$
P_{os+g}		…		…		$2\Psi + \frac{(2+K)*\Psi}{N_d}$
P_{os+g+p}		…		…		$\frac{(2+2+K)*\Psi}{N_d}$

- ψ is the total number of parameters;
- K denotes the memory multiplier of optimizer states;
- N_d denotes the parallel degree.

Data-, Pipeline-, Tensor Model-, Optimizer- Parallelisms



https://www.deepspeed.ai/getting-started/

Generative Inference

Autoregressive Generation

- Autoregressive generation with two phrases computation:
  - **<u>Prefill phrase</u>**:
    - The model takes a prompt sequence as input and engages in the generation of a key-value cache (KV cache) for each Transformer layer.
    - Computation bounded.
  - **<u>Decode phrase</u>**:
    - For each decode step, the model updates the KV cache and reuses the KV to compute the output.
    - IO bounded.
- Parallel Generative Inference:
  - Pipeline parallelism;
  - Tensor model parallelism.

Prefill: TransformerBlocks$(x \in R^{L\times D}) \rightarrow x' \in \mathbb{R}^{L\times D}$

RELAXED SYSTEM LAB

For each inference request:

- ~~$B = 1$;~~
- L is the input sequence length;
- D is the model dimension;
- Multi-head attention: $D = n_H \times H$
- H is the head dimension;
- n_h is the number of heads.

Computation	Input	Output
$Q = xW^Q$	$x \in \mathbb{R}^{L\times D}$, $W^Q \in \mathbb{R}^{D\times D}$	$Q \in \mathbb{R}^{L\times D}$
$K = xW^K$	$x \in \mathbb{R}^{L\times D}$, $W^K \in \mathbb{R}^{D\times D}$	$K \in \mathbb{R}^{L\times D}$
$V = xW^V$	$x \in \mathbb{R}^{L\times D}$, $W^V \in \mathbb{R}^{D\times D}$	$V \in \mathbb{R}^{L\times D}$
$[Q_1, Q_2 \ldots, Q_{n_h}] = \text{Partition}_{-1}(Q)$	$Q \in \mathbb{R}^{L\times D}$	$Q_i \in \mathbb{R}^{L\times H}, i = 1, \ldots n_h$
$[K_1, K_2 \ldots, K_{n_h}] = \text{Partition}_{-1}(K)$	$K \in \mathbb{R}^{L\times D}$	$K_i \in \mathbb{R}^{L\times H}, i = 1, \ldots n_h$
$[V_1, V_2 \ldots, V_{n_h}] = \text{Partition}_{-1}(V)$	$V \in \mathbb{R}^{L\times D}$	$V_i \in \mathbb{R}^{L\times H}, i = 1, \ldots n_h$
$\text{Score}_i = \text{softmax}(\frac{Q_i K_i^T}{\sqrt{D}}), i = 1, \ldots n_h$	$Q_i, K_i \in \mathbb{R}^{L\times H}$	$\text{score}_i \in \mathbb{R}^{L\times L}$
$Z_i = \text{score}_i V_i, i = 1, \ldots n_h$	$\text{score}_i \in \mathbb{R}^{L\times L}$, $V_i \in \mathbb{R}^{L\times H}$	$Z_i \in \mathbb{R}^{L\times H}$
$Z = \text{Merge}_{-1}([Z_1, Z_2 \ldots, Z_{n_h}])$	$Z_i \in \mathbb{R}^{L\times H}, i = 1, \ldots n_h$	$Z \in \mathbb{R}^{L\times D}$
$\text{Out} = ZW^O$	$Z \in \mathbb{R}^{L\times D}$, $W^O \in \mathbb{R}^{D\times D}$	$\text{Out} \in \mathbb{R}^{L\times D}$
$A = \text{Out}\, W^1$	$\text{Out} \in \mathbb{R}^{L\times D}$, $W^1 \in \mathbb{R}^{D\times 4D}$	$A \in \mathbb{R}^{L\times 4D}$
$A' = \text{relu}(A)$	$A \in \mathbb{R}^{L\times 4D}$	$A' \in \mathbb{R}^{L\times 4D}$
$x' = A'W^2$	$A' \in \mathbb{R}^{L\times 4D}$, $W^2 \in \mathbb{R}^{4D\times D}$	$x' \in \mathbb{R}^{L\times D}$

Generate the first token.

$$p(x_{L+1}|x_{1:L}) = \text{softmax}(x_L W_{lm})$$

Decode: TransformerBlocks$(t \in R^{1\times D}) \rightarrow t' \in \mathbb{R}^{1\times D}$

For each inference request:

- ~~$B = 1$;~~
- L is the current cached sequence length; it increases by 1 after each step.
- D is the model dimension;
- Multi-head attention:
  $D = n_H \times H$
- H is the head dimension;
- n_h is the number of heads.

Computationt	Input	Output
$Q = Q_d = tW^Q$	$t \in \mathbb{R}^{1\times D}$, $W^Q \in \mathbb{R}^{D\times D}$	$Q, Q_d \in \mathbb{R}^{1\times D}$
$K_d = tW^K$	$t \in \mathbb{R}^{1\times D}$, $W^K \in \mathbb{R}^{D\times D}$	$K_d \in \mathbb{R}^{1\times D}$
$K = \text{concat}(K_{\text{cache}}, K_d)$	$K_{\text{cache}} \in \mathbb{R}^{L\times D}, K_d \in \mathbb{R}^{1\times D}$	$K \in \mathbb{R}^{(L+1)\times D}$
$V_d = tW^V$	$t \in \mathbb{R}^{1\times D}$, $W^V \in \mathbb{R}^{D\times D}$	$V_d \in \mathbb{R}^{1\times D}$
$V = \text{concat}(V_{\text{cache}}, V_d)$	$V_{\text{cache}} \in \mathbb{R}^{L\times D}, V_d \in \mathbb{R}^{1\times D}$	$V \in \mathbb{R}^{(L+1)\times D}$
$[Q_1, Q_2 \ldots, Q_{n_h}] = \text{Partition}_{-1}(Q)$	$Q \in \mathbb{R}^{1\times D}$	$Q_i \in \mathbb{R}^{1\times H}, i = 1, \ldots n_h$
$[K_1, K_2 \ldots, K_{n_h}] = \text{Partition}_{-1}(K)$	$K \in \mathbb{R}^{(L+1)\times D}$	$K_i \in \mathbb{R}^{(L+1)\times H}, i = 1, \ldots n_h$
$[V_1, V_2 \ldots, V_{n_h}] = \text{Partition}_{-1}(V)$	$V \in \mathbb{R}^{(L+1)\times D}$	$V_i \in \mathbb{R}^{(L+1)\times H}, i = 1, \ldots n_h$
$\text{Score}_\text{i} = \text{softmax}(\frac{Q_i K_i^T}{\sqrt{D}}), i = 1, \ldots n_h$	$Q_i \in \mathbb{R}^{1\times H}, K_i \in \mathbb{R}^{(L+1)\times H}$	$\text{score}_\text{i} \in \mathbb{R}^{1\times(L+1)}$
$Z_i = \text{score}_i V_i, i = 1, \ldots n_h$	$\text{score}_\text{i} \in \mathbb{R}^{1\times(L+1)}$, $V_i \in \mathbb{R}^{(L+1)\times H}$	$Z_i \in \mathbb{R}^{1\times H}$
$Z = \text{Merge}_{-1}([Z_1, Z_2 \ldots, Z_{n_h}])$	$Z_i \in \mathbb{R}^{1\times H}, i = 1, \ldots n_h$	$Z \in \mathbb{R}^{1\times D}$
$\text{Out} = ZW^O$	$Z \in \mathbb{R}^{1\times D}$, $W^O \in \mathbb{R}^{D\times D}$	$\text{Out} \in \mathbb{R}^{1\times D}$
$A = \text{Out}\, W^1$	$\text{Out} \in \mathbb{R}^{1\times D}$, $W^1 \in \mathbb{R}^{D\times 4D}$	$A \in \mathbb{R}^{1\times 4D}$
$A' = \text{relu}(A)$	$A \in \mathbb{R}^{1\times 4D}$	$A' \in \mathbb{R}^{1\times 4D}$
$t' = A'W^2$	$A' \in \mathbb{R}^{1\times 4D}$, $W^2 \in \mathbb{R}^{4D\times D}$	$t' \in \mathbb{R}^{1\times D}$

Update the KV cache:

$$K = \text{concat}(K_{\text{cache}}, K_d)$$

$$V = \text{concat}(V_{\text{cache}}, V_d)$$

Generate the second token:

$$p(x_{L+2} | x_{1:L+1}) = \text{softmax}(\underset{\downarrow}{x_{L+1}} W_{lm})$$

Output of last transformer block's t'.

Customize Text Generation

- **Greedy search**: in every generation step, keep the token with the highest probability.
- **Beam search**: always keeps k candidates and picks the best of the candidates at the end.
- **Sampling**: instead of determinstic selecting the largest tokens, we use a random number generator to sample tokens following the distribution computed by the LM.
  - **Top-k sampling**: only the k (e.g., k = 6) most likely next words are filtered to be sampled.
  - **Top-p sampling**: chooses from the smallest possible set of words whose cumulative probability exceeds the probability p (e.g., p = 0.92).

Generative Inference Optimization

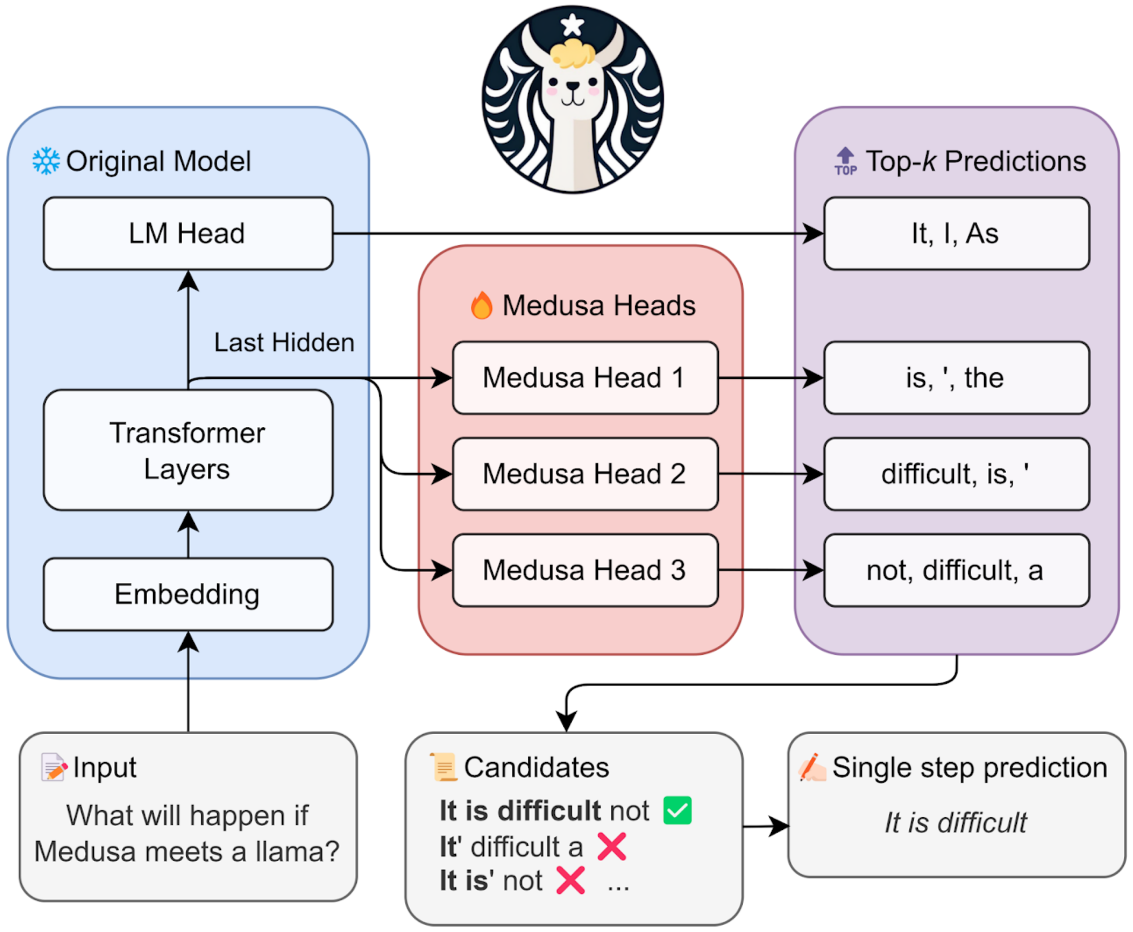
- Decrease the I/O volume - model compression:
  - Model quantization;
  - Knowledge distillation.
- Increase the computation load for each generation step:
  - Speculative decoding;
  - Parallel decoding.

Speculative Decoding

- **Observation**:
  - A small *assistant/ speculative model* very often generates the same tokens as the large original LLM (sometime s referred to as the *main/ target model*).
- Speculative decoding overview:
  - The assistant model auto-regressively generates a sequence of N candidate tokens;
  - The candidate tokens are passed to the original LLM to be verified. The original model takes the candidate tokens as input and performs ***a single forward pass***:
    - All candidate tokens up to the first mismatch are correct;
    - After the first mismatch, replace the first incorrect candidate token with the correct token from the main model (fox), and discard all predicted tokens that come after this mismatched token.
  - Repeat this process until the end condition is reached.

Parallel Decoding

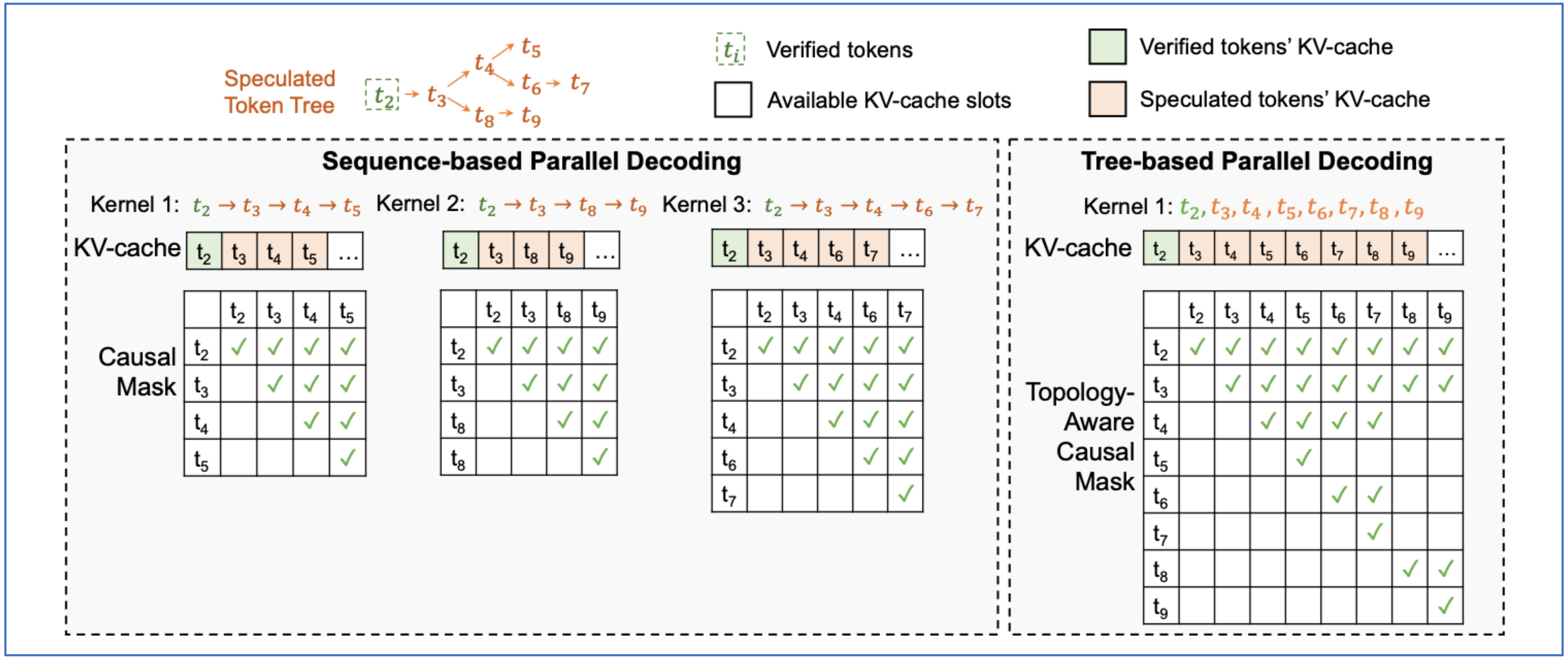
- Rather than pulling in an entirely new assistant model to predict subsequent tokens, Medusa simply extends the original LLM itself.
- Medusa heads are the additional decoding heads built on top of the last hidden states of the LLM, enabling the prediction of several subsequent tokens in parallel.
- Each Medusa head is a single layer of feed-forward network, augmented with a residual connection.
- Training these heads is straightforward: for a relatively small dataset, the original model remains static; only the Medusa heads are updated.



https://www.together.ai/blog/medusa

Tree Attention

- Homework 4 solutions will be released after the DDL + 5 days.



Speculated Token Tree
$t_2 \rightarrow t_3$; $t_3 \rightarrow t_4$; $t_3 \rightarrow t_8$; $t_4 \rightarrow t_5$; $t_4 \rightarrow t_6 \rightarrow t_7$; $t_8 \rightarrow t_9$
t_i Verified tokens
☐ Available KV-cache slots
🟩 Verified tokens' KV-cache
🟧 Speculated tokens' KV-cache
Sequence-based Parallel Decoding
Kernel 1: $t_2 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5$
Kernel 2: $t_2 \rightarrow t_3 \rightarrow t_8 \rightarrow t_9$
Kernel 3: $t_2 \rightarrow t_3 \rightarrow t_4 \rightarrow t_6 \rightarrow t_7$
KV-cache
Causal Mask
Tree-based Parallel Decoding
Kernel 1: $t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9$
KV-cache
Topology-Aware Causal Mask

Sequence-based Parallel Decoding

Kernel 1 KV-cache: | t_2 | t_3 | t_4 | t_5 | … |

	t_2	t_3	t_4	t_5
t_2	✓	✓	✓	✓
t_3		✓	✓	✓
t_4			✓	✓
t_5				✓

Kernel 2 KV-cache: | t_2 | t_3 | t_8 | t_9 | … |

	t_2	t_3	t_8	t_9
t_2	✓	✓	✓	✓
t_3		✓	✓	✓
t_8			✓	✓
t_8				✓

Kernel 3 KV-cache: | t_2 | t_3 | t_4 | t_6 | t_7 | … |

	t_2	t_3	t_4	t_6	t_7
t_2	✓	✓	✓	✓	✓
t_3		✓	✓	✓	✓
t_4			✓	✓	✓
t_6				✓	✓
t_7					✓

Tree-based Parallel Decoding

KV-cache: | t_2 | t_3 | t_4 | t_5 | t_6 | t_7 | t_8 | t_9 | … |

	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9
t_2	✓	✓	✓	✓	✓	✓	✓	✓
t_3		✓	✓	✓	✓	✓	✓	✓
t_4			✓	✓	✓	✓		
t_5				✓				
t_6					✓	✓		
t_7						✓		
t_8							✓	✓
t_9								✓

RELAXED
SYSTEM LAB

Improve the Generative Inference Quality

Prompt Enginering

- ***Prompt engineering*** is the practice of developing and optimizing prompts to efficiently use large language models (LLMs) for a variety of applications.
- Rules of Thumb from OpenAI.
- Advanced prompt engineering:
  - Baseline: zero-shot prompting;
  - Few-shot prompting;
  - Chain-of-thought prompting;
  - Self-consistency;
  - Tree of thoughts (ToT);
  - Retrieval augmented generation.
- Model safety issues:
  - Prompt injection;
  - Prompt leaking;
  - Jailbreaking.

Soft Prompts

- ***Soft prompts*** are *learnable* tensors combined with the original input embeddings that can be optimized for a dataset. The downside is that they aren't human-readable because you aren't matching these "virtual tokens" to the embeddings of a real word.
  - Prompt tuning;
  - Prefix tuning;
  - P-tuning.

Parameter Efficient Fine-Tuning

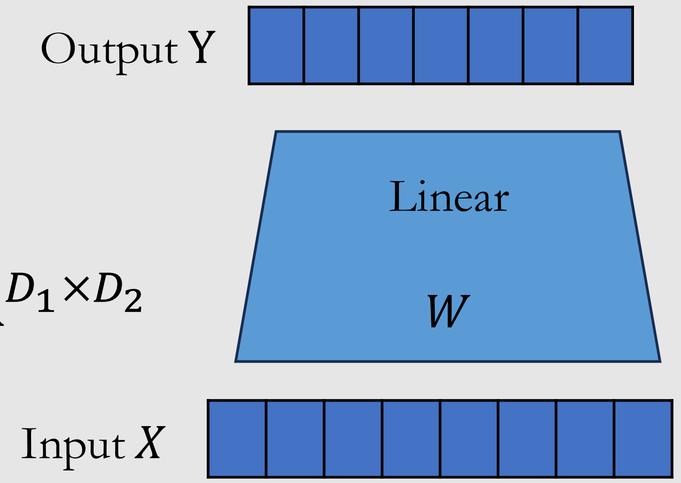
- *Parameter efficient fine-tuning (PEFT):* Rather than finetuning the entire model, we finetune only small amounts of weights.
    - Frozen layer/Subset fine-tuning: pick a subset of the parameters, fine-tune only those layers, and freeze the rest of the layers.
    - Adapters: add additional layers that have few parameters and tune only the parameters of those layers, keeping all others fixed.
    - Low-rank adaption (LoRA): learn a low-rank approximation of the weight matrices.

LoRA

- Keep the original pre-trained parameters W fixed during fine-tuning;
- Learn an additive modification to those parameters ΔW;
- Define ΔW as a low-rank decomposition: $\Delta W = AB$.

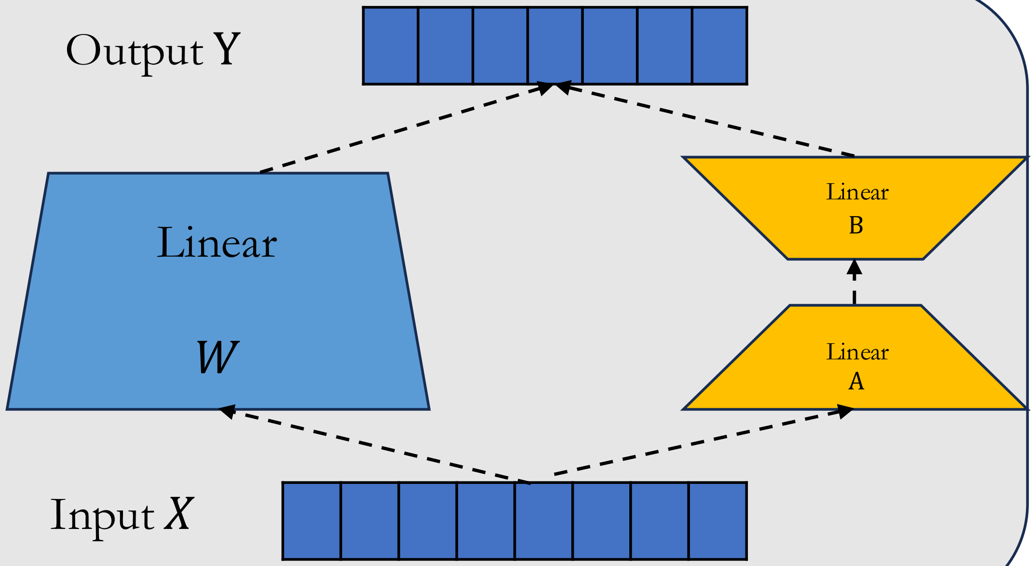
Standard Linear Layer

- $Y = XW$
- $X \in \mathbb{R}^{D_1}, Y \in \mathbb{R}^{D_2}, W \in \mathbb{R}^{D_1 \times D_2}$



LoRA Linear Layer

- $Y = XW + XAB = X(W + AB)$
- $X \in \mathbb{R}^{D_1}, Y \in \mathbb{R}^{D_2}, W \in \mathbb{R}^{D_1 \times D_2}$
- $A \in \mathbb{R}^{D_1 \times R}, B \in \mathbb{R}^{R \times D_2}$



Final Exam Arrangement

- Date: **May 27, 2024 (Monday)**
- Time:
  - **180 minutes, 12:30-15:30** (You can leave earlier if you want.)
- Location:
  - **G010, CYT Bldg**
- Bring:
  - Your student ID;
  - One page of your cheating sheet in A4 size, printed or hand-written.
  - No electronic devices are allowed.
- Do NOT use a pencil in the exam. Otherwise, you are not allowed to appeal for any grading disagreements!
- The **HKUST Academic Honor Code** applies! I am very serious about this!
- Absence:
  - I must be informed, and your appeal must be confirmed by email before the exam starts.

RELAXED
SYSTEM LAB

Good Luck!