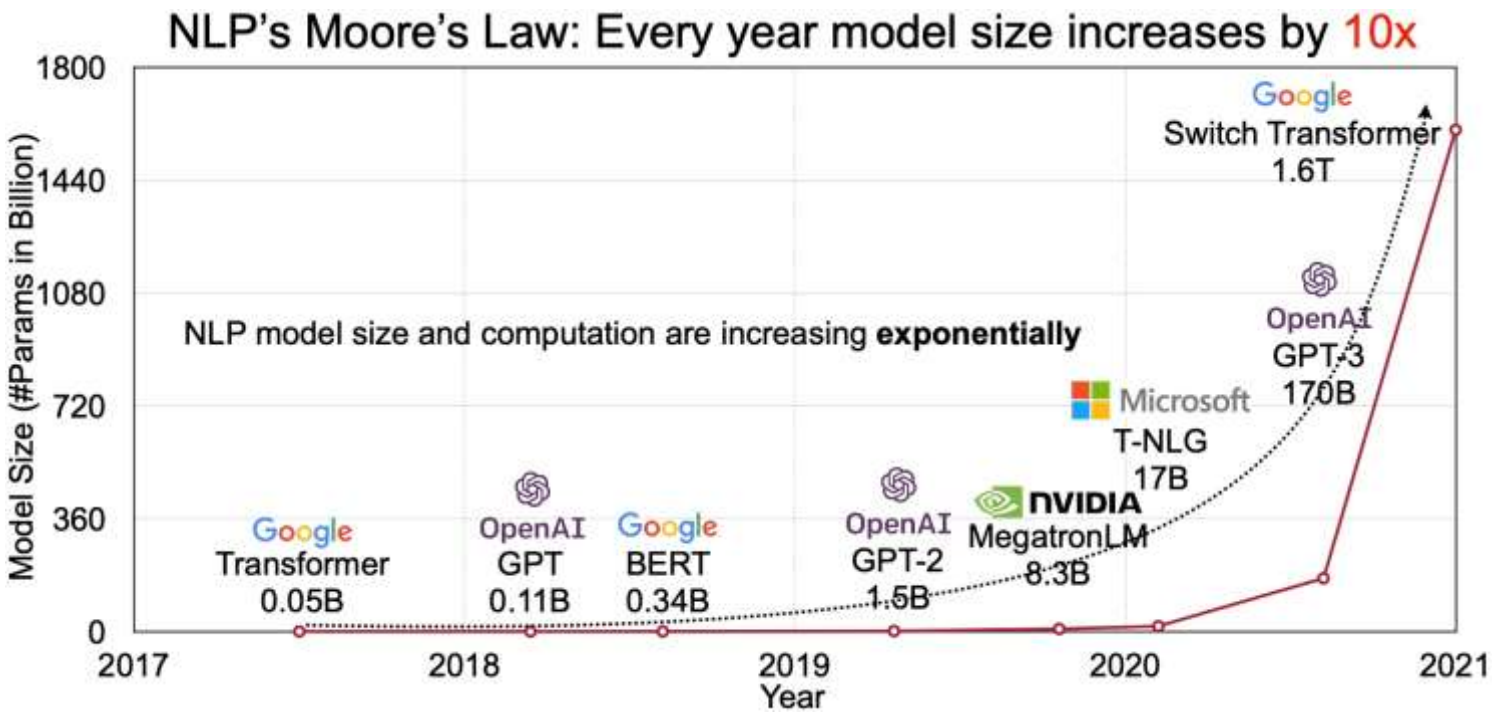
Training Very Large Language Models

Jie Tang
Department of Computer Science & Technology
Tsinghua University

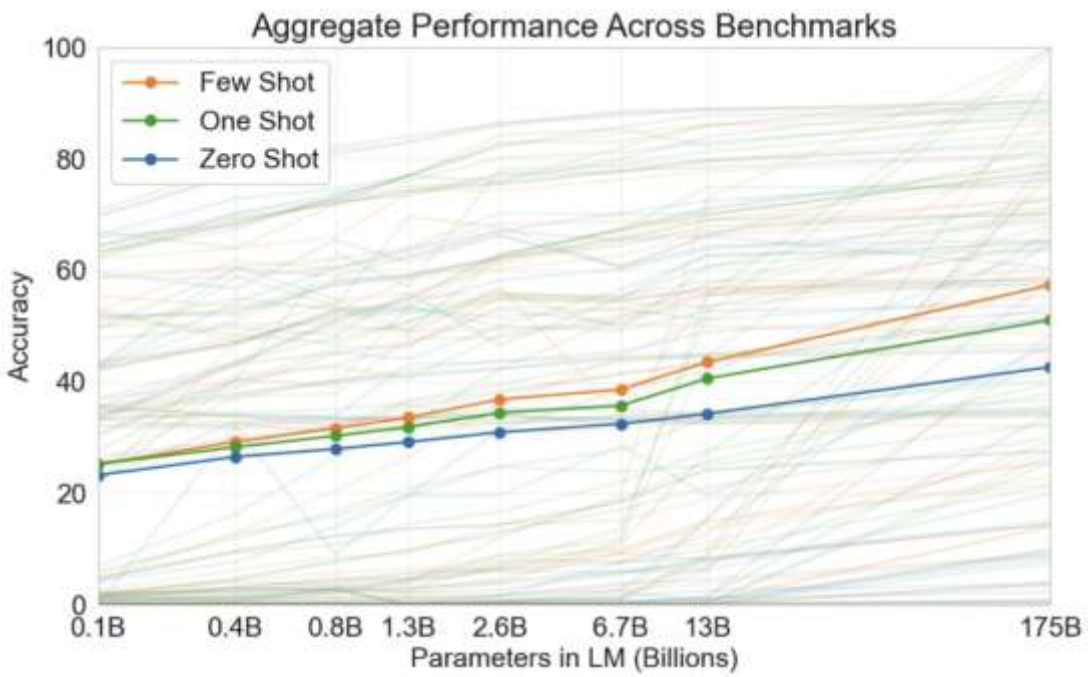
Training Big Models

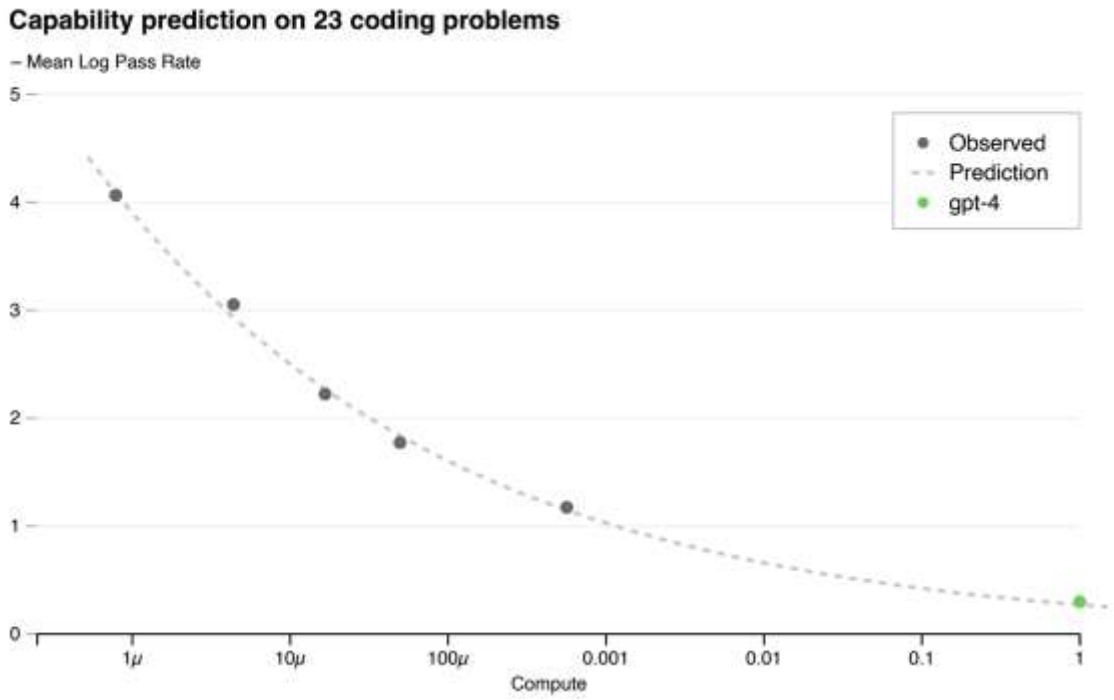
- **Moore's Law for LLM**: the size grows 10 times a year
- Eager to train larger and larger models in this era



Bigger Model, Better Accuracy

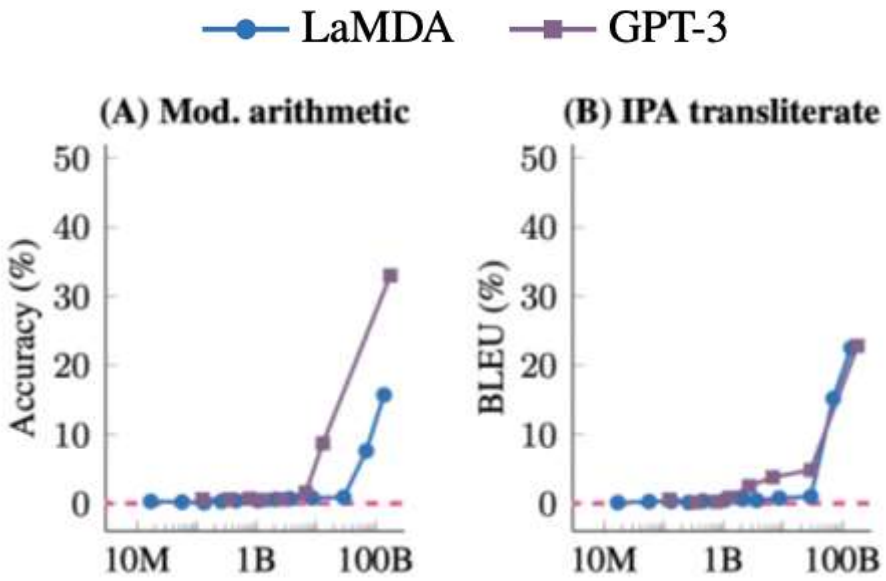
- GPT-3 & GPT-4 show that the performance contiune to improve as the model and computation scale up

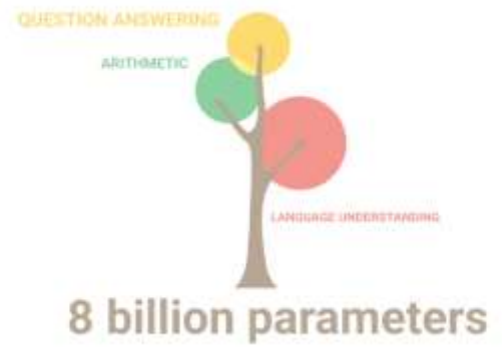




Emerging Ability of Foundation Models

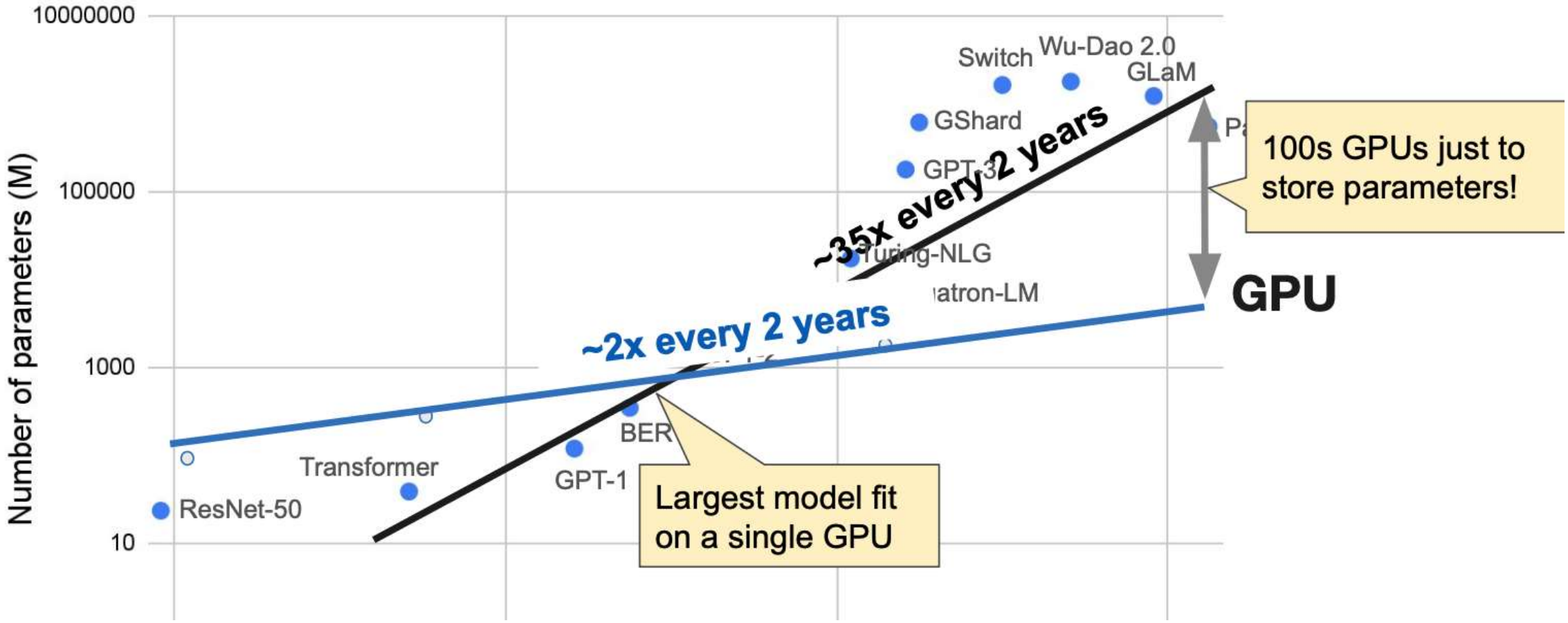
- Emergent Abilities of Big Models (~100B, or 10^{23} FLOPs)
  - *An ability is emergent if it is not present in smaller models but is present in larger models*
  - The ability to perform a task is emergent when a language model achieves random performance until a certain scale, after which performance significantly increases to well-above random
- One foundation model for all tasks





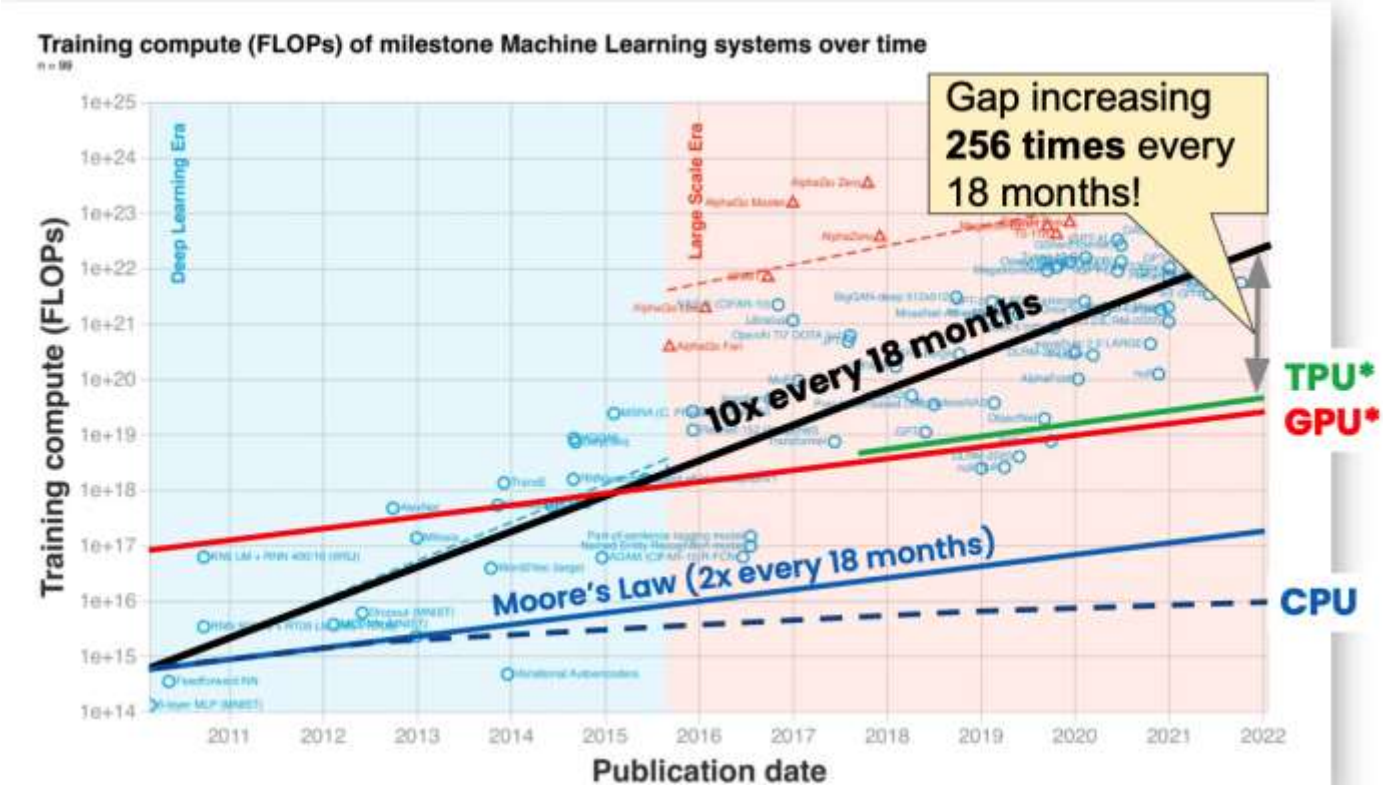
Training Big Models

- Growing gap between memory of a single GPU and model
  - We need parallel to train super-large models



Training Big Models

- High training costs: more FLOPS => more computational resources
  - GPT-3 used 10,000 V100s at a cost of $4.6 million
  - PaLM 540B used 6144 TPUv4s: $2.56*10^{24}$ FLOPs (260 years for one A100)



Challenges in Training Big Models

Memory Challenge

GPT-3 has 175B parameters and requires 1.8TB of gpu memory to initiate training (weights + gradient + optimizer state), which is far more than a single card's gpu memory.

Performance Challenge

Large model training is computationally intensive, requiring hardware and software and joint tuning to squeeze the performance of the cluster as much as possible to save computational cost and time

Challenges in Training Big Models

Communication Challenge

Parallel training of large models involves multi-machine communication, which requires reasonable design of parallel strategies for different model sizes, data sample sizes, and cluster topology features to improve communication efficiency

Maintenance Challenge

Operation and maintenance of hardware and software of thousands of node clusters to ensure the correctness of computation, high availability of model training, and timely recovery in case of training failure

Outline

- **How to train big models**
  - Intro-model optimization:
    - Mixed Precision & Rematerialization
  - Data parallelism and ZeRO optimizer
  - Model parallelism
    - Inter-op parallelism: tensor parallelism
    - Intra-op parallelism: pipeline parallelism
- Train 100B-scale models in action
  - Take GLM-130B as the example
- Data behind large language models
  - Common Crawl, WebText, etc.
  - Documentation for datasets

Intro-model optimization

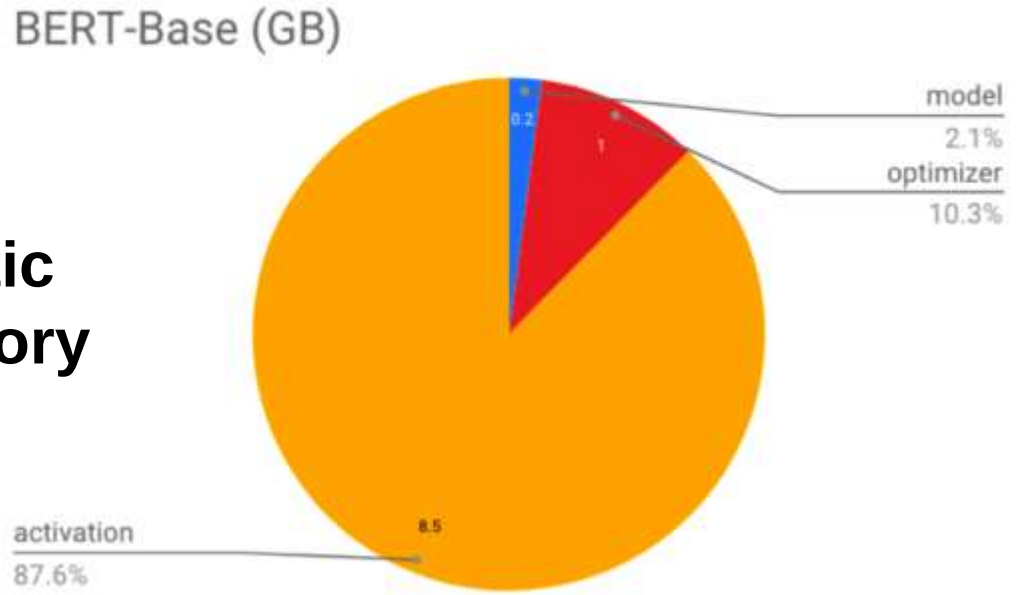
Assuming the model can fit in a single GPU, how can we train a model as large / fast as possible using a single GPU ?

- Use a better GPU (3090Ti → A100 / GH200) *not our topic*
- Reduce memory consumption
- Reduce computional cost
- Balance memory & computation

Where the Memory Goes?

- Model
  - Parameters (fp32) 4 bytes
  - Gradients (fp32) 4 bytes
- Optimizer states
  - Adam m (fp32) 4 bytes
  - Adam v (fp32) 4 bytes

Static Memory



- Activations: saved in forward function for backward
- Other: intermediate results in operators

Dynamic Memory

Mixed-Precision Training: FP format

- 16-bit format: FP16 / BF16
- 2-8x speedup compared to FP32
- BF16: 3080, A100, TPU only

bfloat16: Brain Floating Point Format

Range: ~$1e^{-38}$ to ~$3e^{38}$

fp32: Single-precision IEEE Floating Point Format

Range: ~$1e^{-38}$ to ~$3e^{38}$

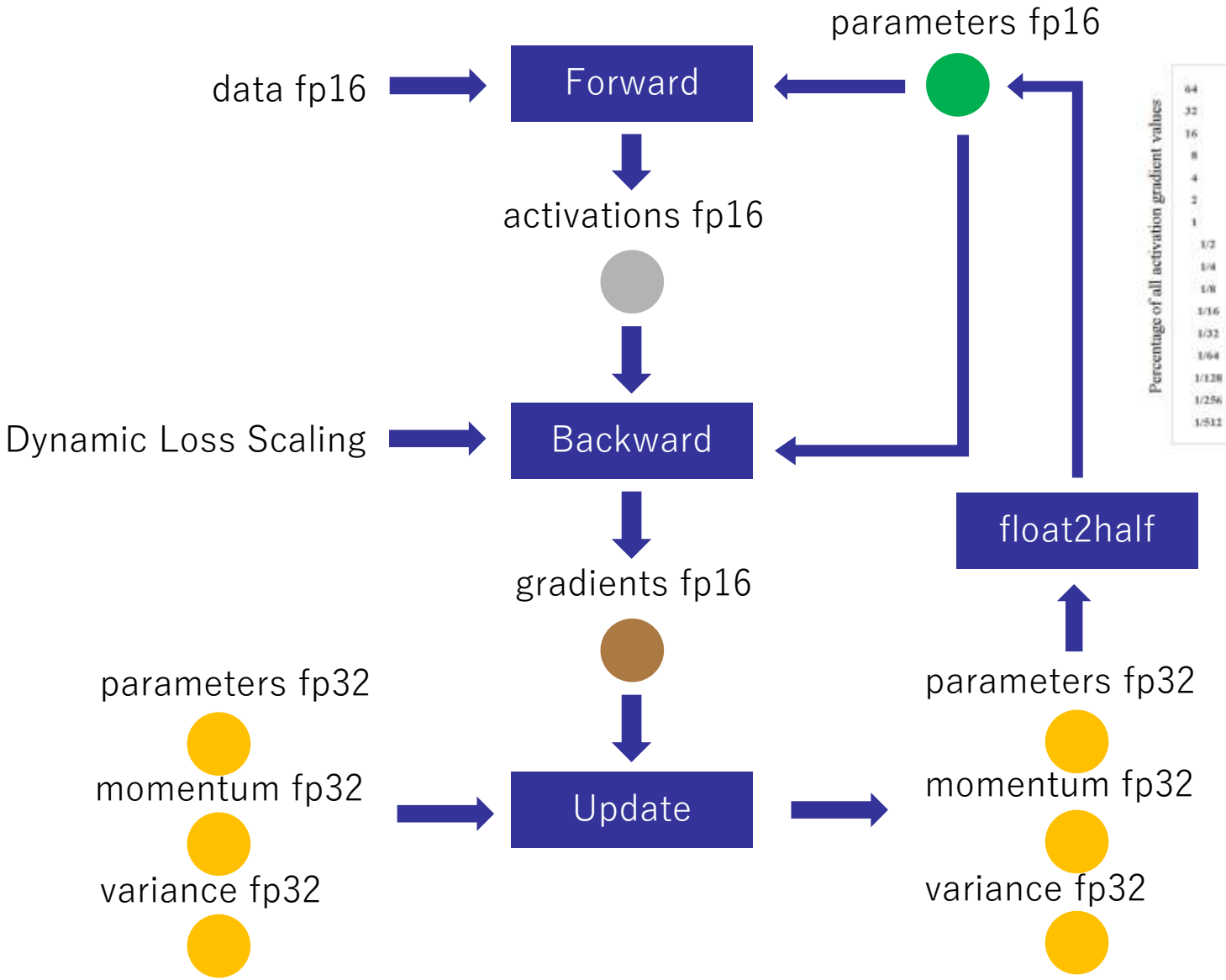
fp16: Half-precision IEEE Floating Point Format

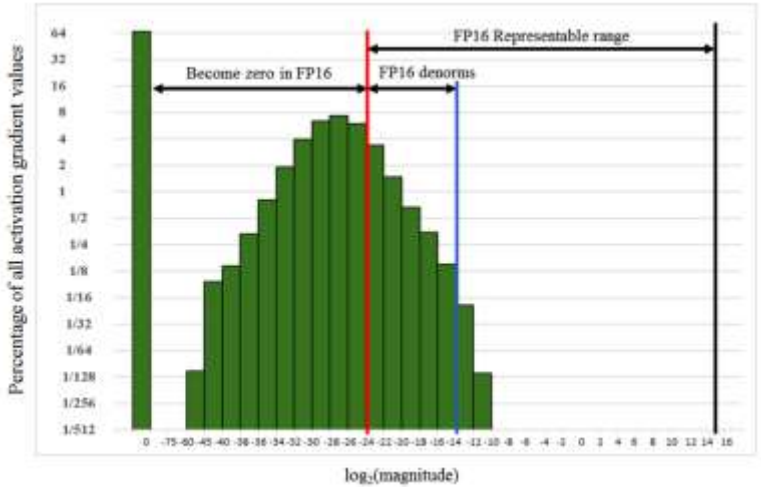
Range: ~$5.96e^{-8}$ to 65504

NVIDIA A100 TENSOR CORE GPU SPECIFICATIONS (SXM4 AND PCIE FORM FACTORS)

	A100 40GB PCIe	A100 80GB PCIe	A100 40GB SXM	A100 80GB SXM
FP64	9.7 TFLOPS			
FP64 Tensor Core	19.5 TFLOPS			
FP32	19.5 TFLOPS			
Tensor Float 32 (TF32)	156 TFLOPS | 312 TFLOPS*			
BFLOAT16 Tensor Core	312 TFLOPS | 624 TFLOPS*			
FP16 Tensor Core	312 TFLOPS | 624 TFLOPS*			
INT8 Tensor Core	624 TOPS | 1248 TOPS*			
GPU Memory	40GB HBM2	80GB HBM2e	40GB HBM2	80GB HBM2e
GPU Memory Bandwidth	1,555GB/s	1,935GB/s	1,555GB/s	2,039GB/s
Max Thermal Design Power (TDP)	250W	300W	400W	400W
Multi-Instance	Up to 7	Up to 7	Up to 7	Up to 7

Mixed-Precision Training



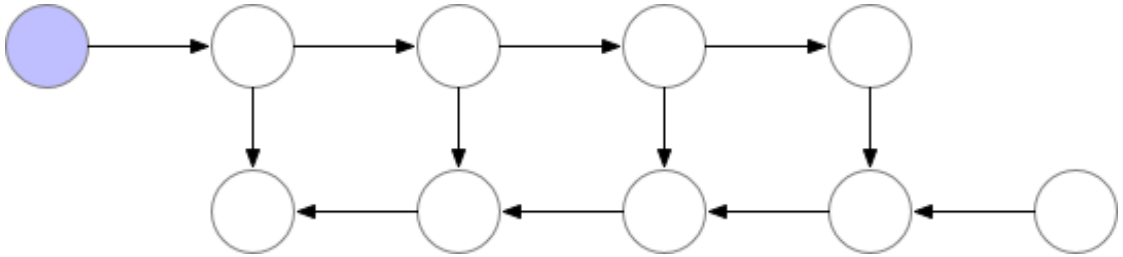


Mixed-Precision Training

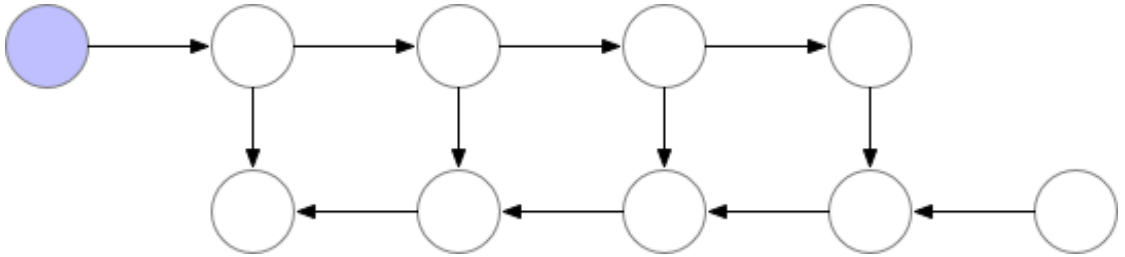
- Advantages:
  - Reduce the memory overhead for model weight
  - Speed up training due to fewer bit operation
- Disadvantage
  - More techniques are required to stabilize the training due to more limited numeric range (parameters & activations with FP16)
  - Optimizer states with FP16 can cause instability and harm the performance
- One step further ?
  - Float-8 is now supported in H100 / H800
  - How to stabilize the training with more limited range still deserves further exploration.

Rematerialization

- Rematerialization:
  - Dynamic memory is saved the computational graph for backpropagation and takes larger proportion than static memory
  - Save dynamic memory by recomputing activations when used.
- Drop activations in forward, recompute them in backward
  - Forward: 1 unit, backward: 2 unit, remat: 1 unit => 25% overhead
  - Also known as checkpoint activations, activation recomputing



Vanilla forward-backward

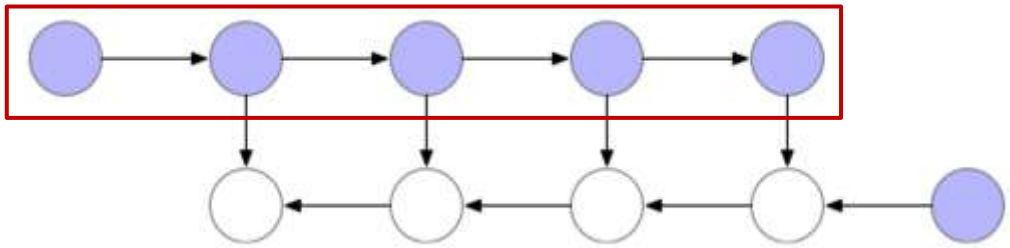


Checkpointing

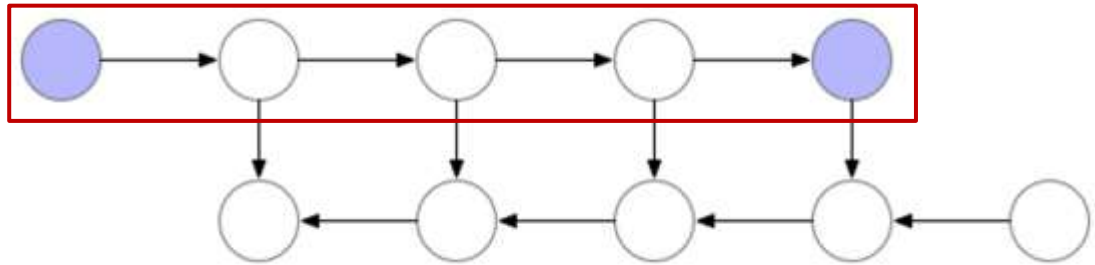
PC from https://github.com/cybertronai/gradient-checkpointing

Gradient Checkpointing

- Vanilla forward-backward
  - save ***all*** activation states for backward
- Gradient checkpointing
  - Almost save no activation states
  - Less memory usage but extra computation
  - The memory is less sensitive the number of layers



Vanilla forward-backward



Checkpointing

PC from https://github.com/cybertronai/gradient-checkpointing

Where the memory goes?

- Model
  - Parameters (half) 4-> **2** bytes
  - Gradients (half) 4-> **2** bytes

Mixed Precision

⇒ **Training can be faster**

- Optimizer states
  - Adam m (fp32) 4 bytes
  - Adam v (fp32) 4 bytes
  - Adam maintains an *additional* Master Weight (fp32, 4 bytes) in mixed precision mode

⇒
- Is memory saved?
- In parallel training, model weights and gradients will be replicated for multiple times but optimizer states are only saved once

Where the memory goes?

- Model
  - Parameters (half) 4-> **2** bytes
  - Gradients (half) 4-> **2** bytes

Mixed Precision

- Optimizer states
  - Adam m (fp32) 4 bytes
  - Adam v (fp32) 4 bytes
  - Master Weight（fp32, 4 bytes）

- ⇓ **Activations**：checkpointing reduces 50%~60% memory
- Other：intermediate results in operators

Parallelism in Training

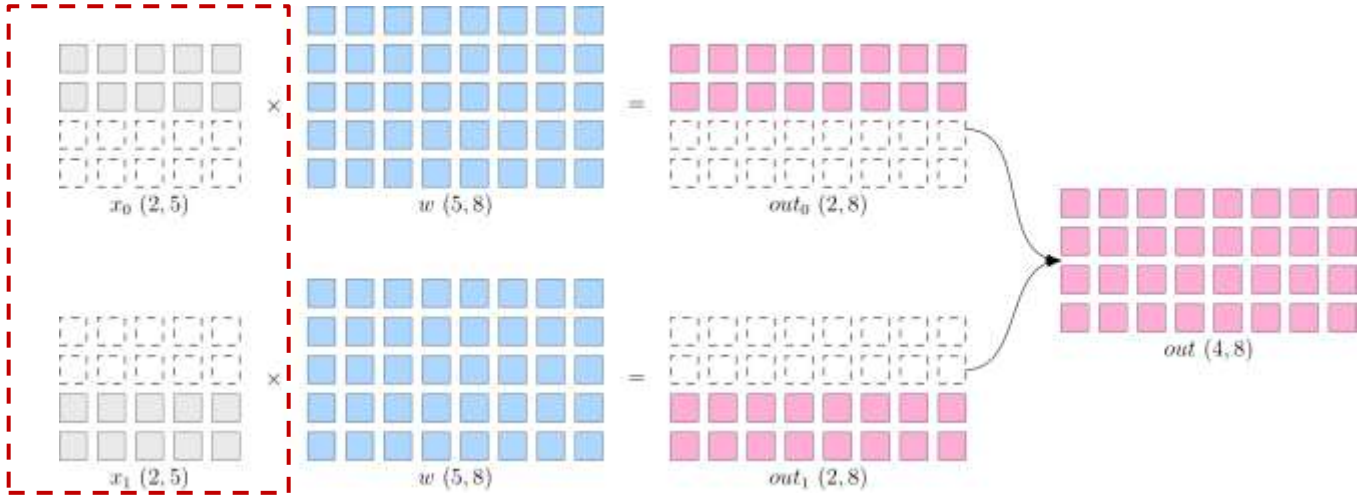
Necessity of Parallelism

- The increasing amount of data-set significantly increases the training time. Therefore it is important to come up with parallel and distributed algorithms which can run much faster and which can drastically reduce training times.
- Current Model are becoming too large to fit into one single GPU. It requires multiple GPUs to load the model weights and states

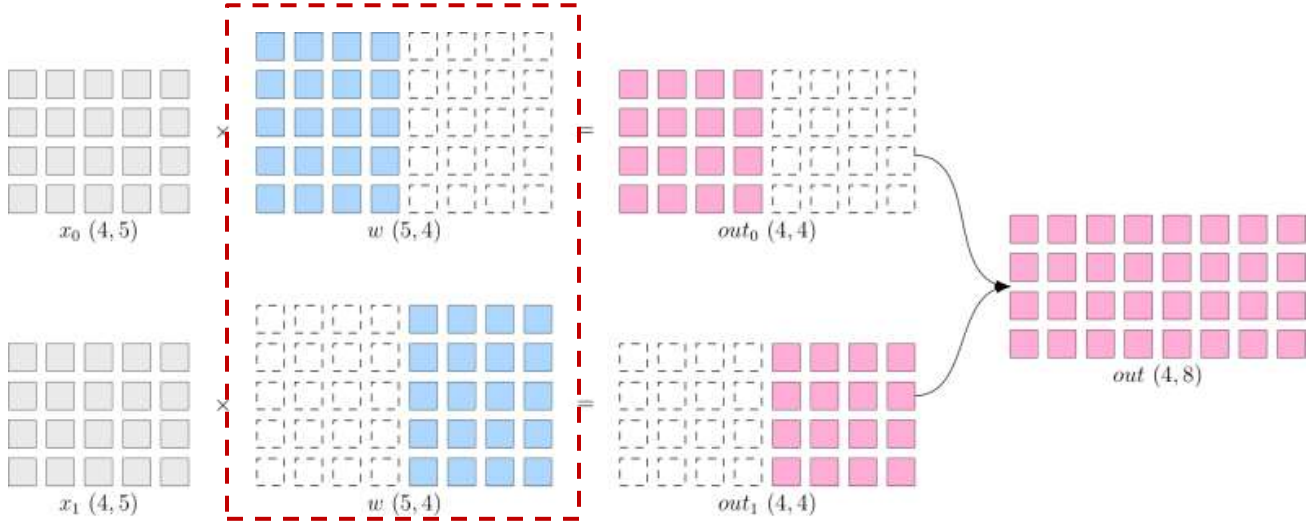
- Model Parallel: distribute **model weights** into different GPUs
- Data Parallel: shard **data** into model replicas with multiple GPUs

Data & Model Parallelism: Simple Illustration

Data Parallel:
split input data



Model Parallel:
Split model weights



Model can fit in a single GPU

DATA PARALLELISM: MORE THROUGHTPUT

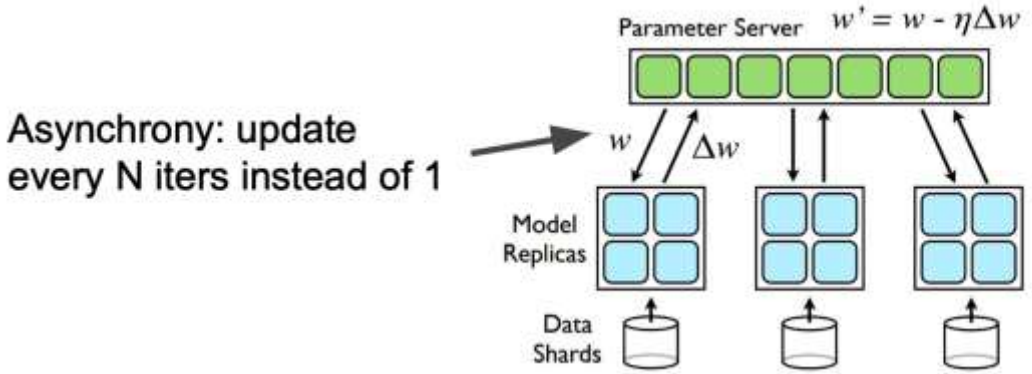
Data Parallelism

- Training requires a larger batch size to make use of more and more data.
- It is hard to increase the batch size infinitely within one GPU
  - Especially, when the model grows larger and larger

- Data Parallelism:
  - Duplicate model into multiple GPUs
  - And shard data to model replicas
  - The global batch size increases linearly to the number of GPUs

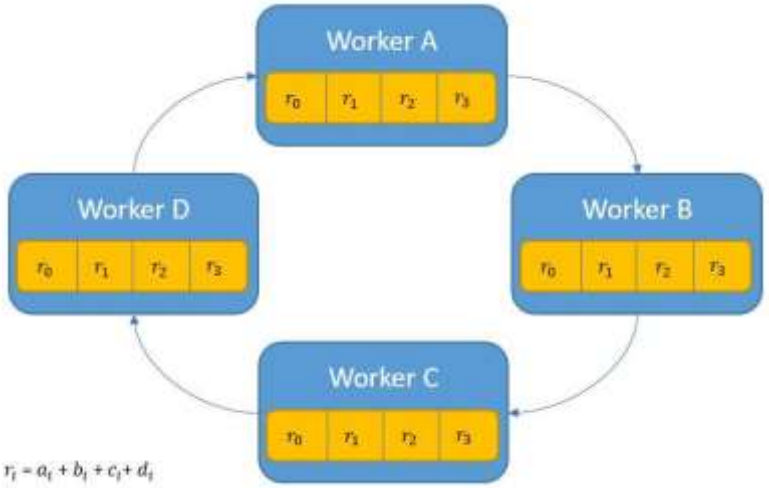
Data Parallelism: Parameter Server

- Data Parallelism: shard data to model replicas
- Distributed Subgradient Descent: The Parameter Server approach
  - Load data from distributed dataloader
  - Perform forward + backward => gradient
  - Push gradient to the Server
  - Pull weight from the Server
- Asynchrony : usually update every N iters



Data Parallelism: All-Reduce

- Cons for Parameter Server:
  - Hard for load balancing
- Distributed Subgradient Descent: The All-Reduce approach
  - Load data from distributed dataloader
  - Perform forward + backward => gradient
  - ~~Push gradient to PS~~ **Perform All-Reduce**
  - ~~Pull weight from PS~~ **Update weight locally**
- Pros:
  - Balanced communication for all workers
  - Easy to implement (PyTorch DDP)
- Cons:
  - Redundant parameter update for all workers




```
import torch.nn.parallel as dist
from torch.nn.parallel import DistributedDataParallel as DDP

dist.init_process_group("nccl", rank=rank, world_size=world_size)
ddp_model = DDP(Model(), device_ids=[rank])

for batch in data_loader:
   loss = train_step(ddp_model, batch)
```

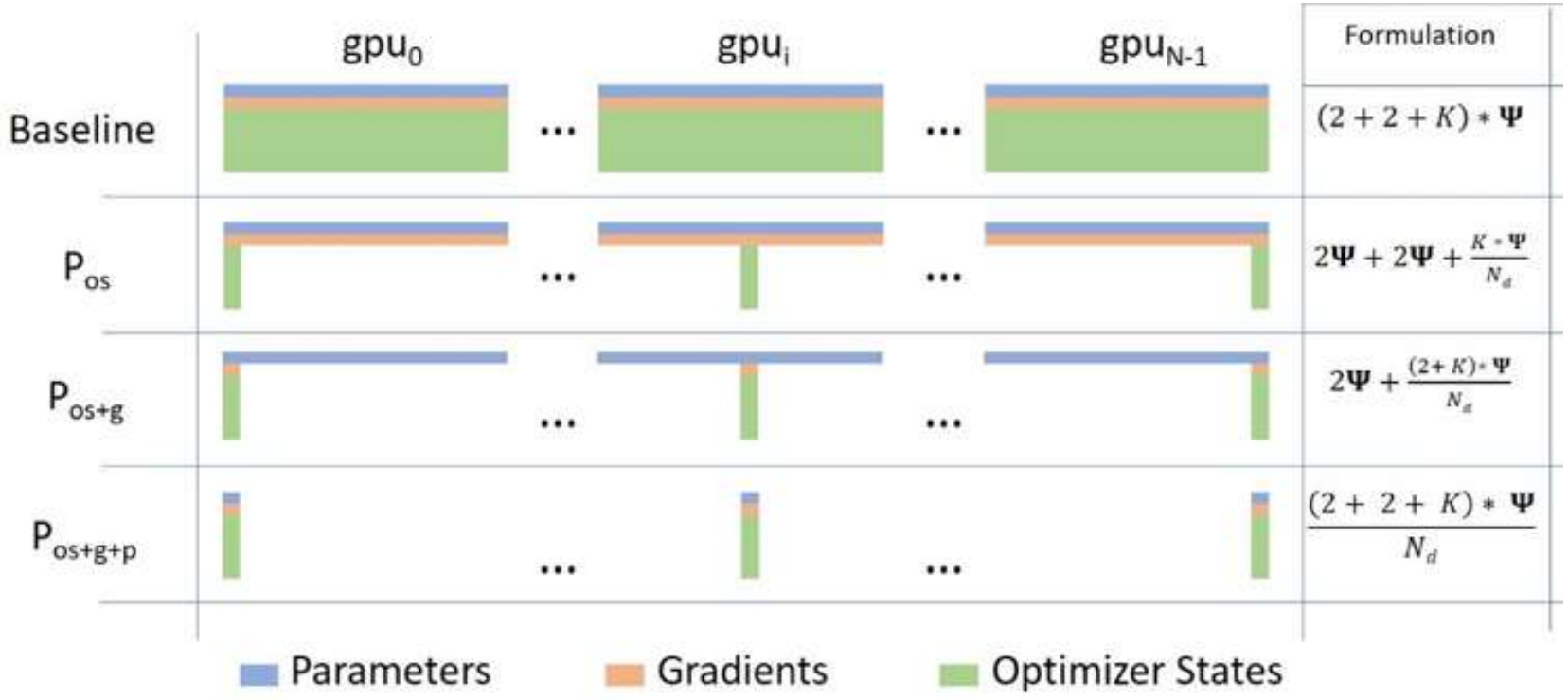
Data Parallelism: ZeRO Optimizer

- Problem for pure data parallelism
  - Each GPU maintains a replica of gradients, optimizer states and model weights while they are the same across GPUs **Redundant**
  - Model: Parameters（half）2 bytes + Gradients （half）2 bytes
  - Optimizer states: Master Weight（fp32） 4 bytes + Adam 8 bytes

- Solution: ***ZeRO**:* Zero Redundancy Optimizer
  - Partition all parameters to different GPUs
    - Optimizer states P_{os} ----- *(Zero-1)*
    - Gradients P_g ----- *(Zero-2)*
    - Model weight P_p ----- *(Zero-3)*

Data Parallelism: ZeRO Optimizer

- ZeRO: Zero Redundancy Optimizer
- Solution: Partition optimizer states, gradients, and model weight



Data Parallelism: ZeRO Optimizer

- ZeRO: Zero Redundancy Optimizer
- Solution: Partition optimizer states, gradients, and model weight

M is the number of parameters, N is the number of devices.

	Optimizer States (12M)	Gradients (2M)	Model Weights (2M)	Memory Cost	Communication Cost
Data Parallelism	Replicated	Replicated	Replicated	$16M$	all-reduce(2M)
ZeRO Stage 1	Partitioned	Replicated	Replicated	$4M + \frac{12M}{N}$	all-reduce(2M)
ZeRO Stage 2	Partitioned	Partitioned	Replicated	$2M + \frac{14M}{N}$	all-reduce(2M)
ZeRO Stage 3	Partitioned	Partitioned	Partitioned	$\frac{16M}{N}$	1.5 all-reduce(2M)

Data Parallelism ?

- As the model becomes larger, only *zero-3* can be applied to train super-large models
  - 100B parameters takes around 200G for model weights
- Zero-3 brings extra communication cost and the cost is linear to the model size, which could be a heavy burden when the computation is far less than communication.

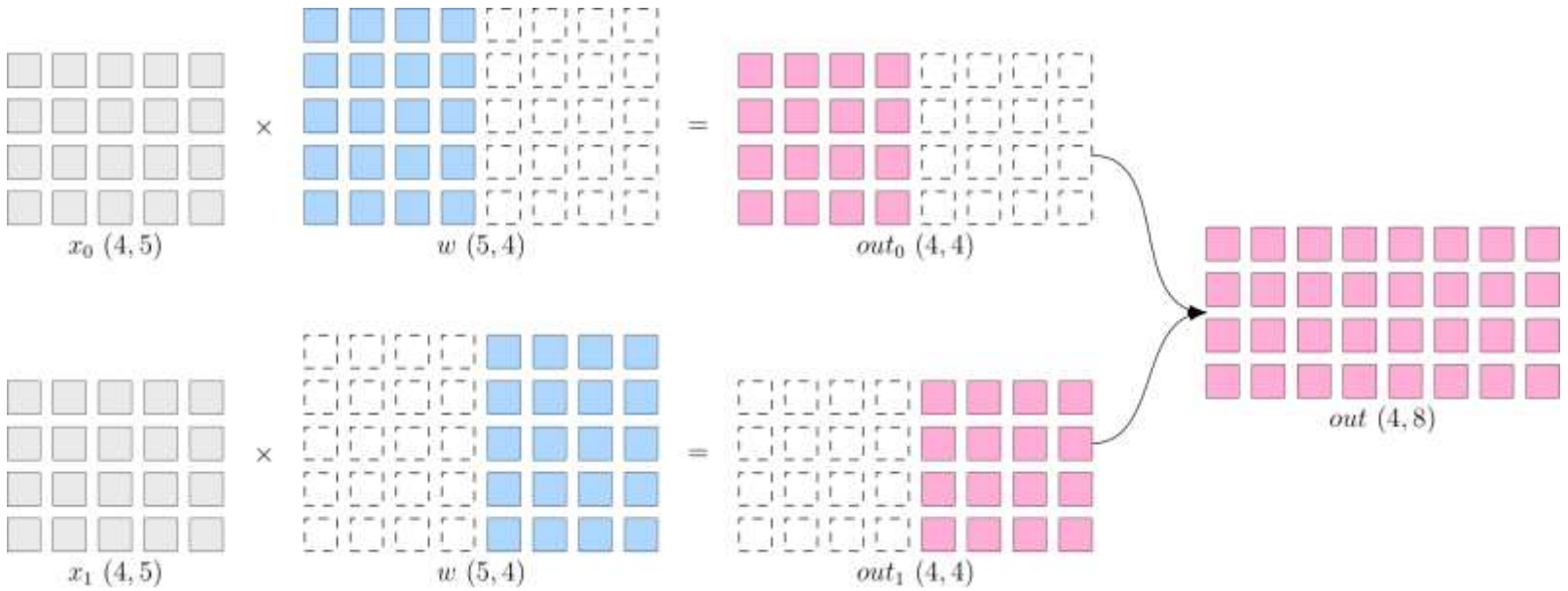
- How can we reduce the communication cost or make the cost not that sensitive to model size ?

Model cannot fit in a single GPU

TENSOR PARALLELISM

Tensor Parallel

- Tensor parallelism splits a tensor (i.e., model weights) into N chunks along a specific dimension
- Each device only holds 1/N of the whole tensor while not affecting the training process

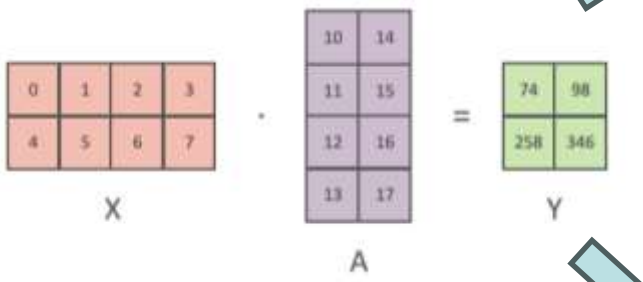


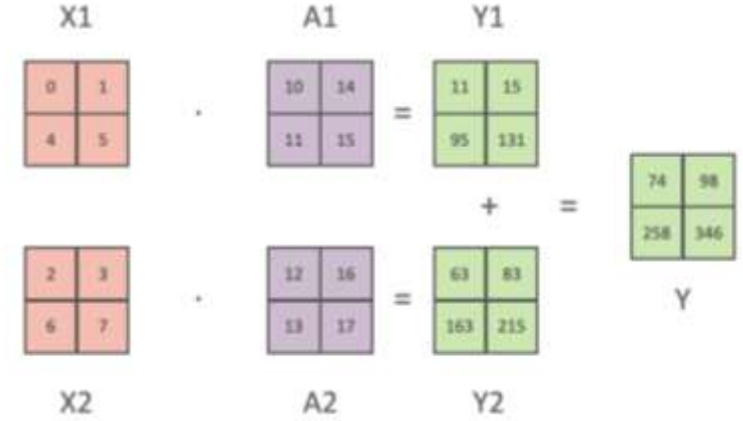
Tensor Parallel

- Input data $X \in \mathbb{R}^{n \times d_1}$
- model weights A $\in \mathbb{R}^{d_1 \times d_2}$

$$W = [W_1, \dots, W_3]^T, W_i \in \mathbb{R}^{d_2 \times \frac{d_1}{m}}$$

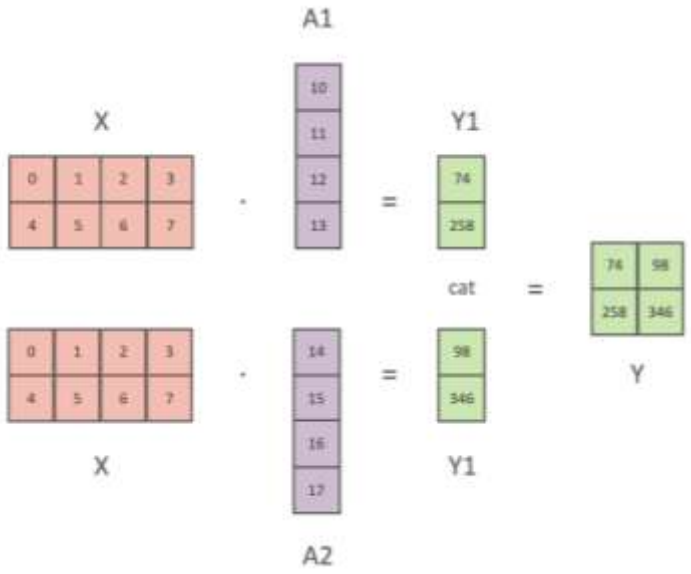
Row-parallel with all-reduce





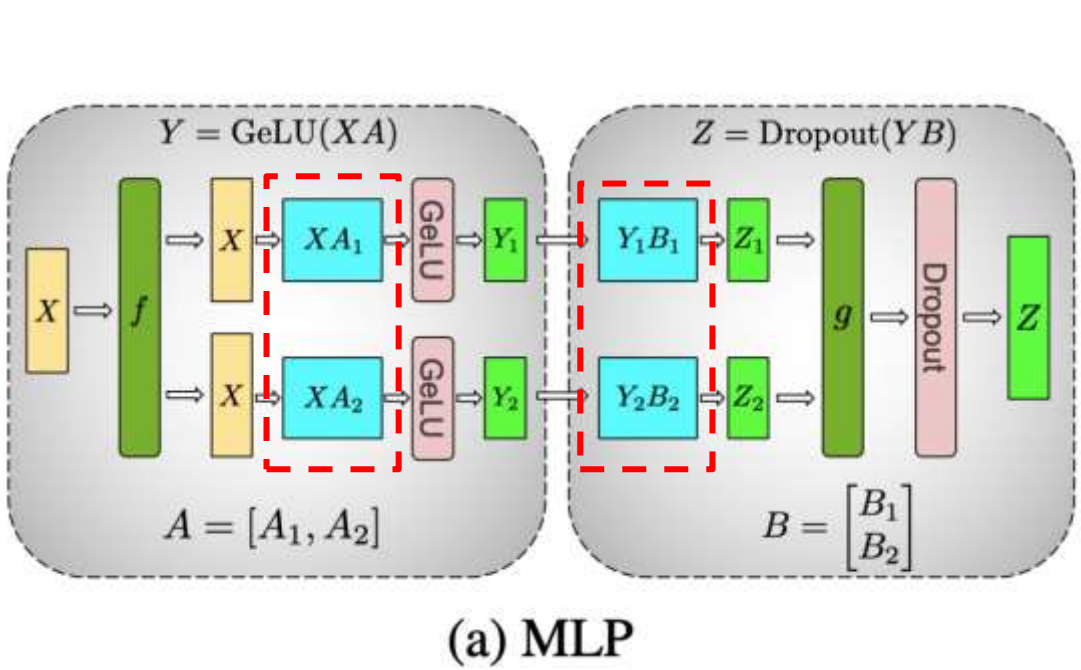
$$W = [W_1, \dots, W_3], W_i \in \mathbb{R}^{d_1 \times \frac{d_2}{m}}$$

Column-parallel with all-gather

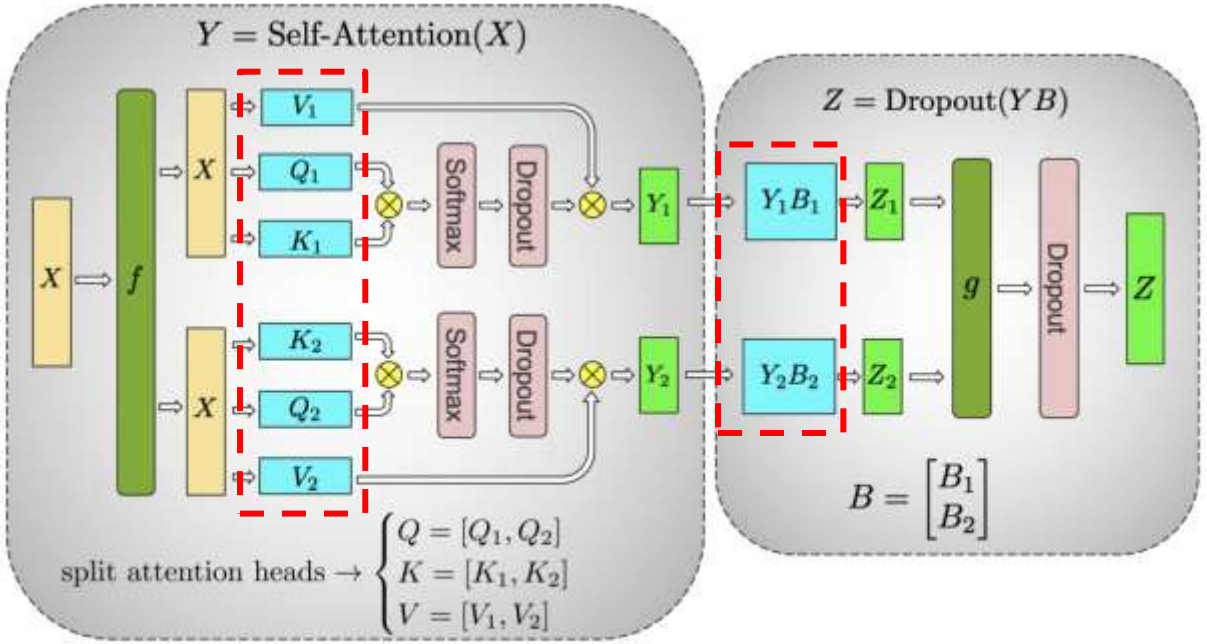


Tensor Parallel

- Tensor Parallel for Transformer
  - Two linear layer in MLP
  - 4 linear layers in Attention (query / key / value / output）



(a) MLP



(b) Self-Attention

Model cannot fit in a single machine

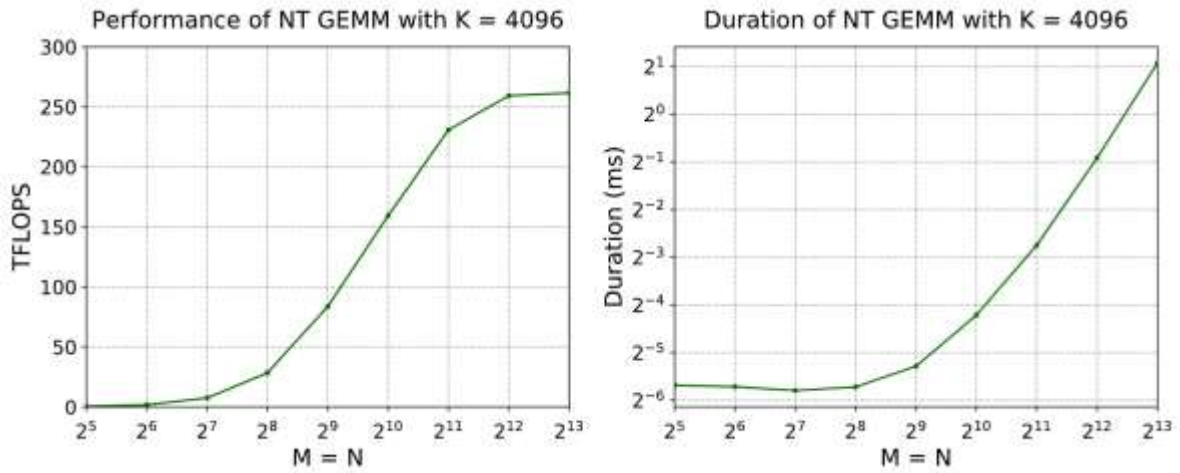
PIPELINE PARALLELISM

Pipeline Parallelism: Motivation

- Cons: Tensor Parallel is often limited intra node

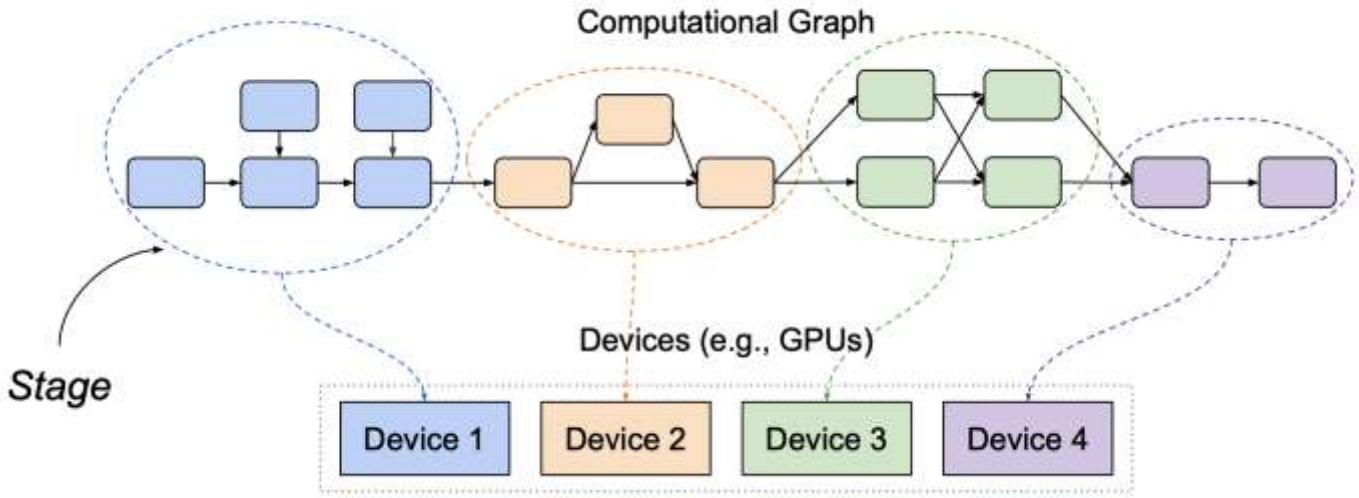
  1. Ring All-Reduce: bounded by the minimum communication bandwidth
     - DGX-A100: 600GB/s bandwidth for any two GPU intra nodes
     - One DGX-A100 with 1 or 2 ConnectX-6 IB cards, each with 200Gb bandwidth
     - **1/12 bandwidth cut for doing tensor parallel across nodes**

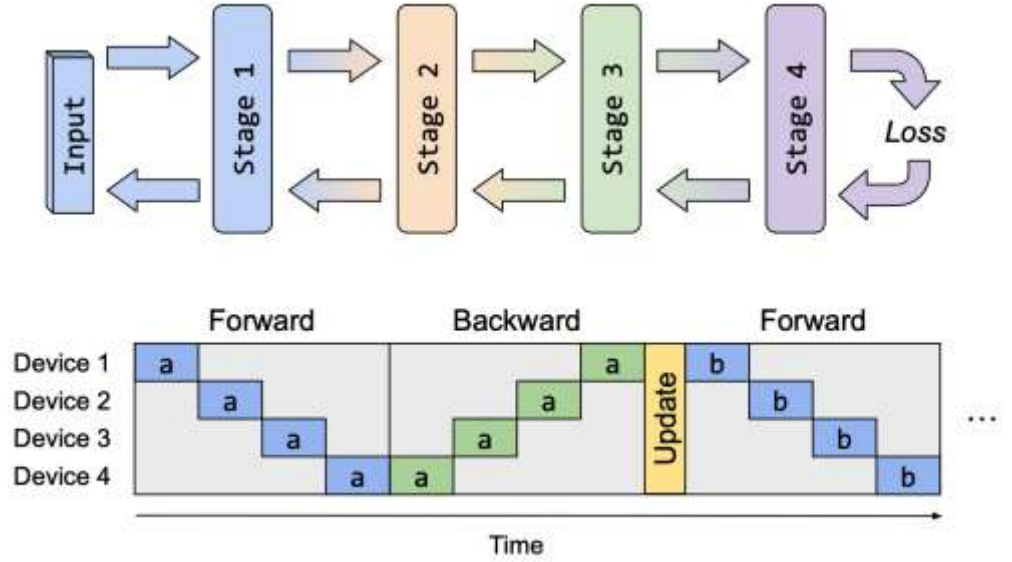
  2. Reduce granularity: Too small input to the neural network will reduce the hardware efficiency.



Pipeline Parallelism

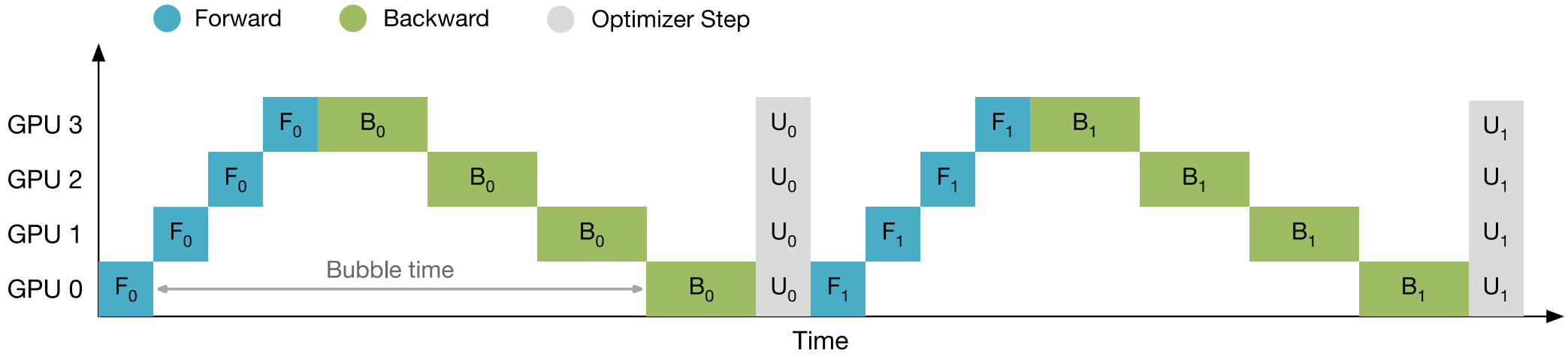
- Pipeline parallelism
  - Split model into multiple stages
  - Transfer output between stages
- Pros: Small data transfer cost, since only communicates stage outputs at stage boundaries
- Cons: Pipeline bubbles





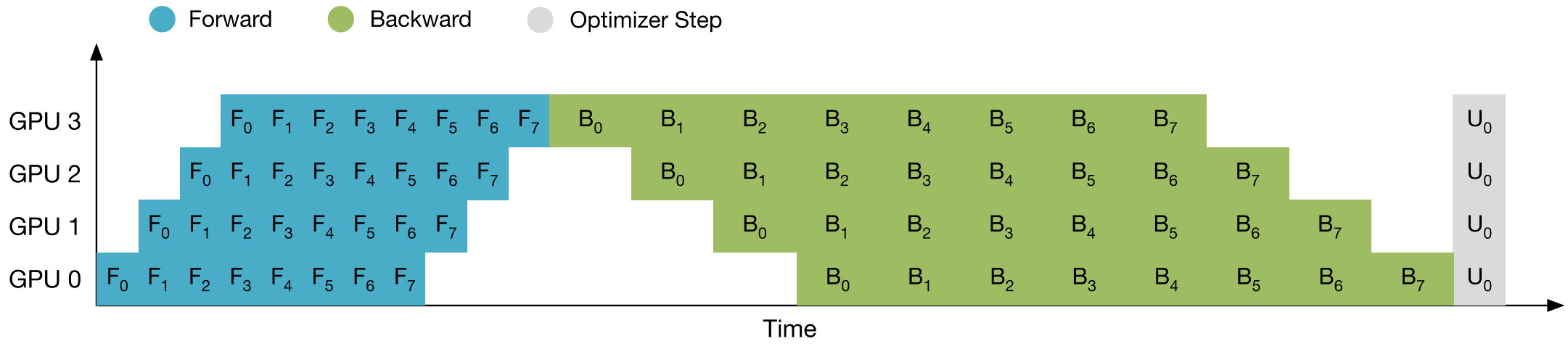
Pipeline Parallelism: Bubble Analysis

- Only 1 device activated at a time.
- Pipeline bubble percentage = bubble_area / total_area = (P - 1) / P, assuming P stages.



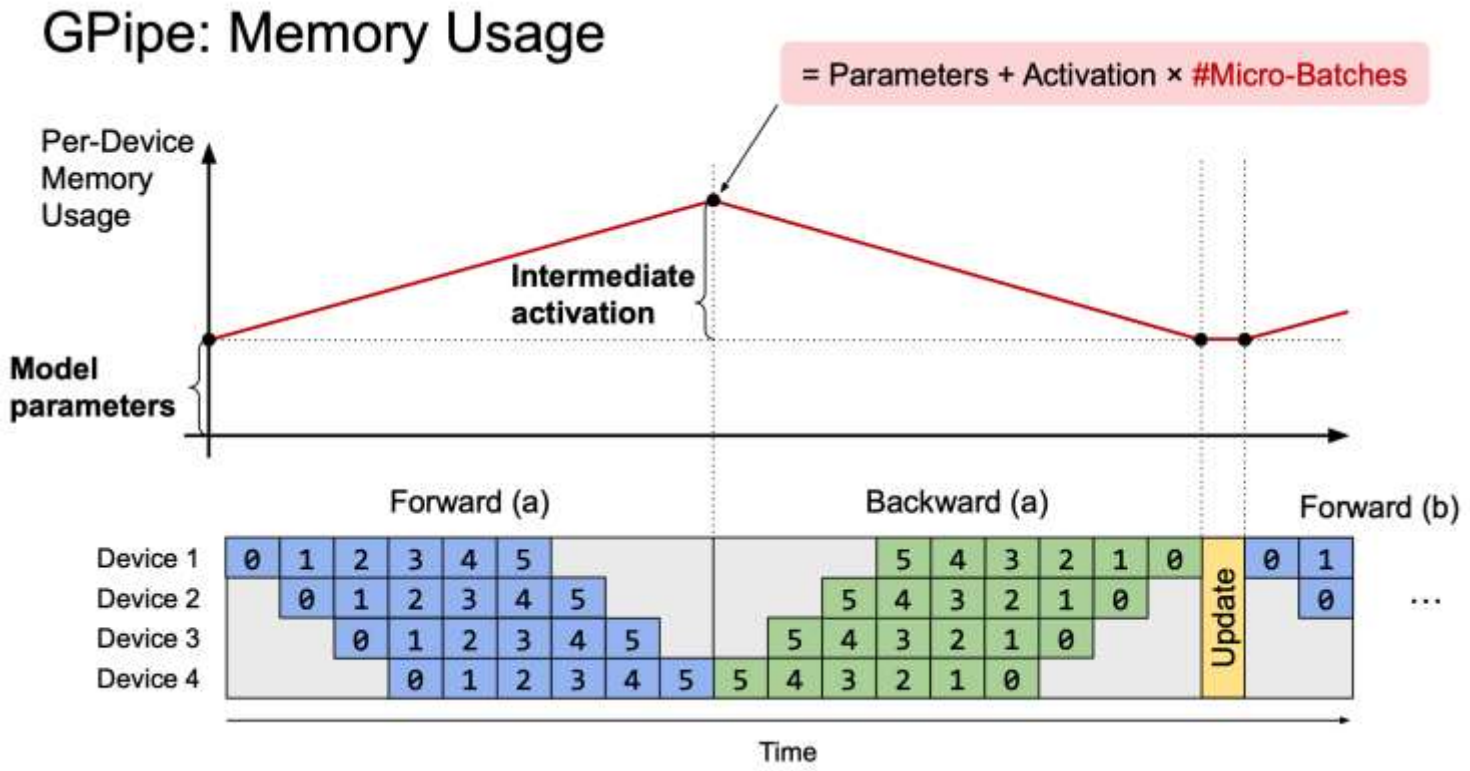
Pipeline Parallelism: GPipe

- GPipe: Reduce Pipeline Bubbles via Pipelining Inputs
- Pipeline bubbles percentage = (P - 1) / (P - 1 + M) with P stages and M inputs.
- **Problem: Memory usage**



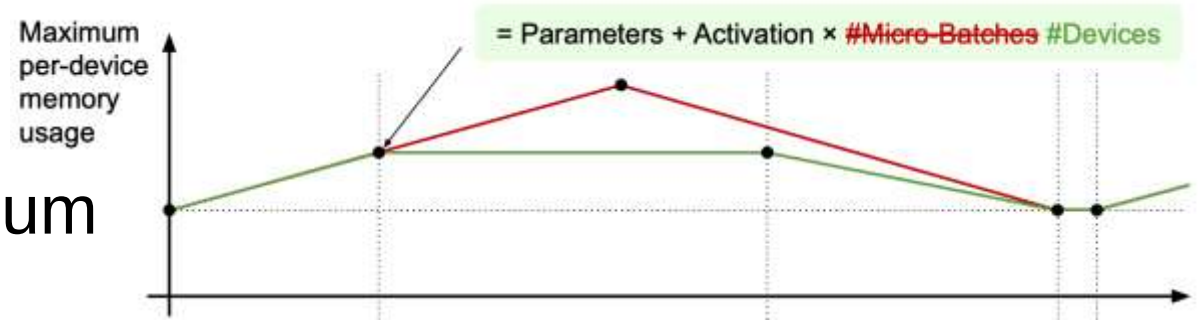
Pipeline Parallelism: GPipe

- Intermediate activation of all M micro batches need to be stored
- bubbles percentage = (P - 1) / (P - 1 + M)
- More micro batches => less bubbles percentages, more memory usage

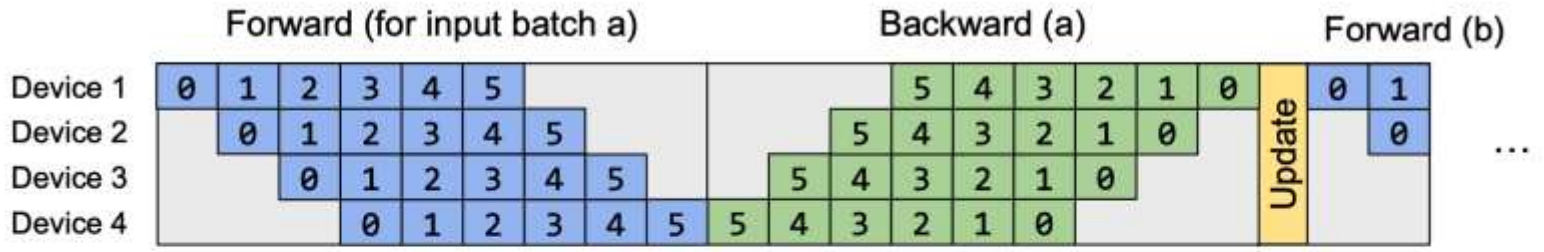


Pipeline Parallelism: 1F1B

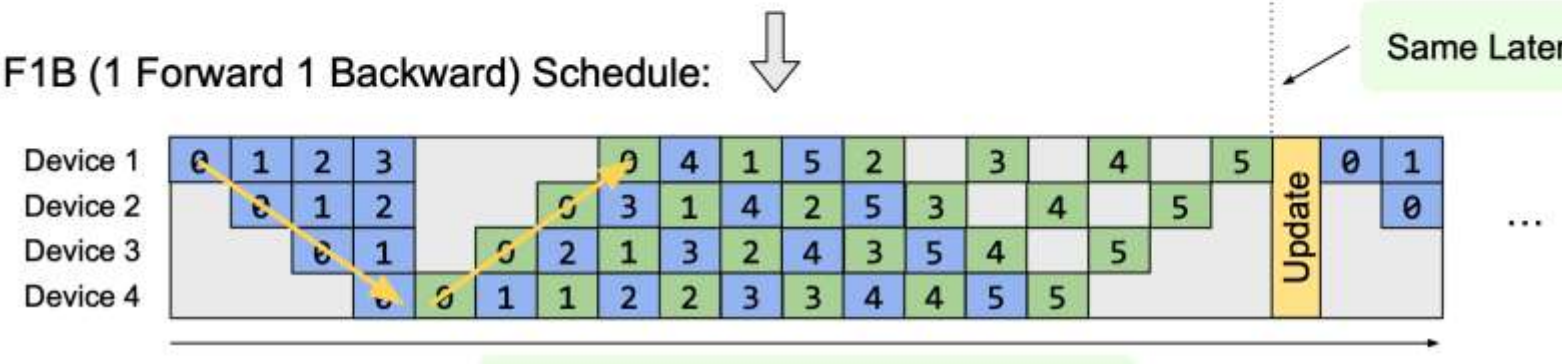
- 1F1B schedule (PipeDream-Flush): Perform backward as early as possible
- Same lantency
- Memory is only relevant to stage num



GPipe Schedule:



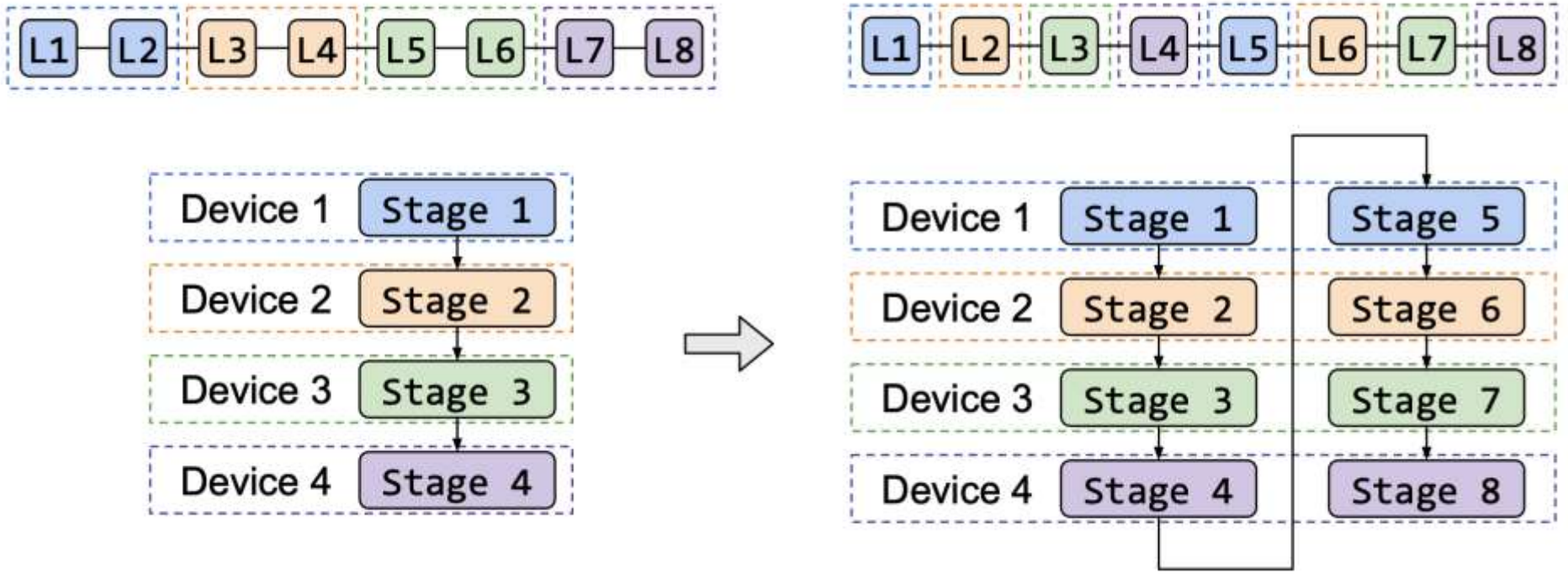
1F1B (1 Forward 1 Backward) Schedule:



Same Latency

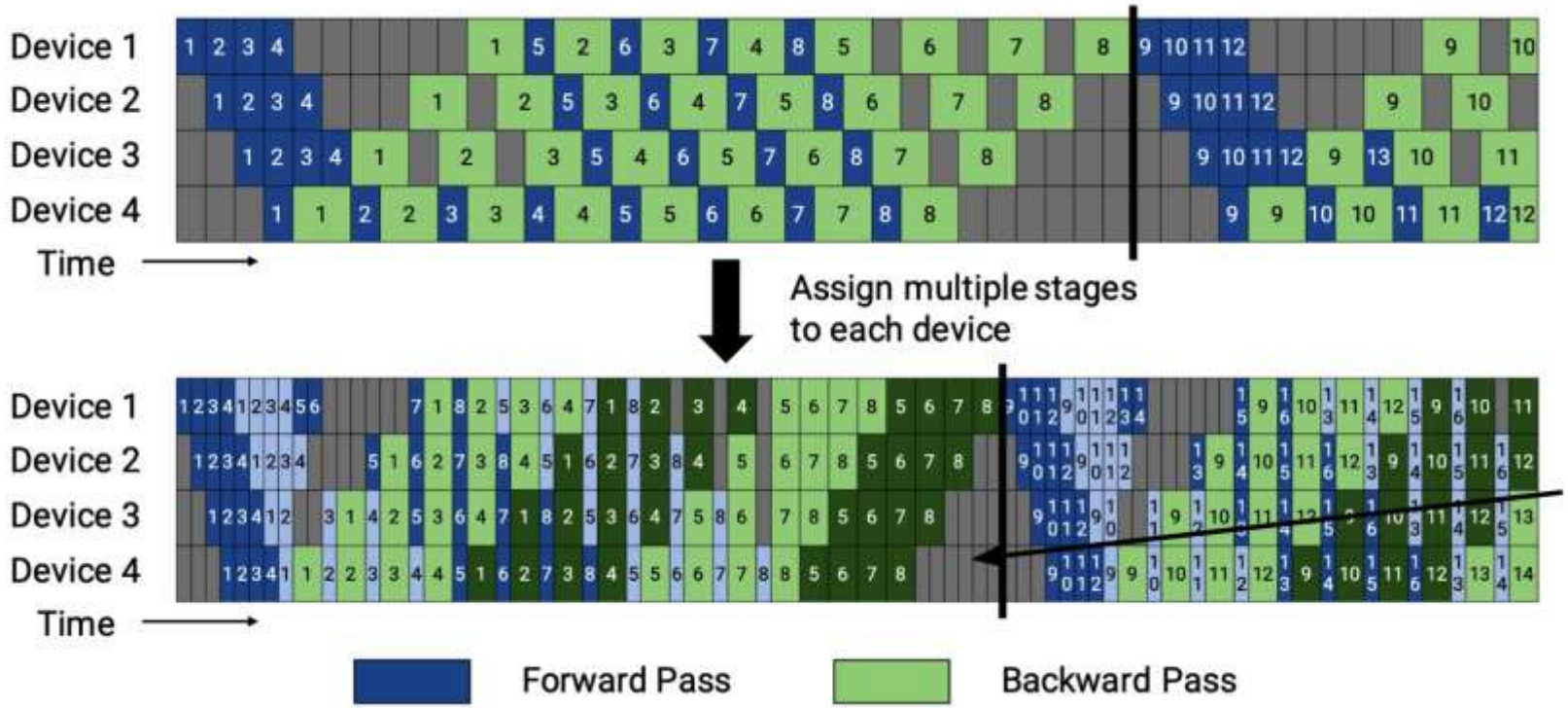
Pipeline Parallelism: Interleaved 1F1B

- **Idea:** Slice the neural network into more fine-grained stages and assign multiple stages to reduce pipeline bubble.



Pipeline Parallelism: Interleaved 1F1B

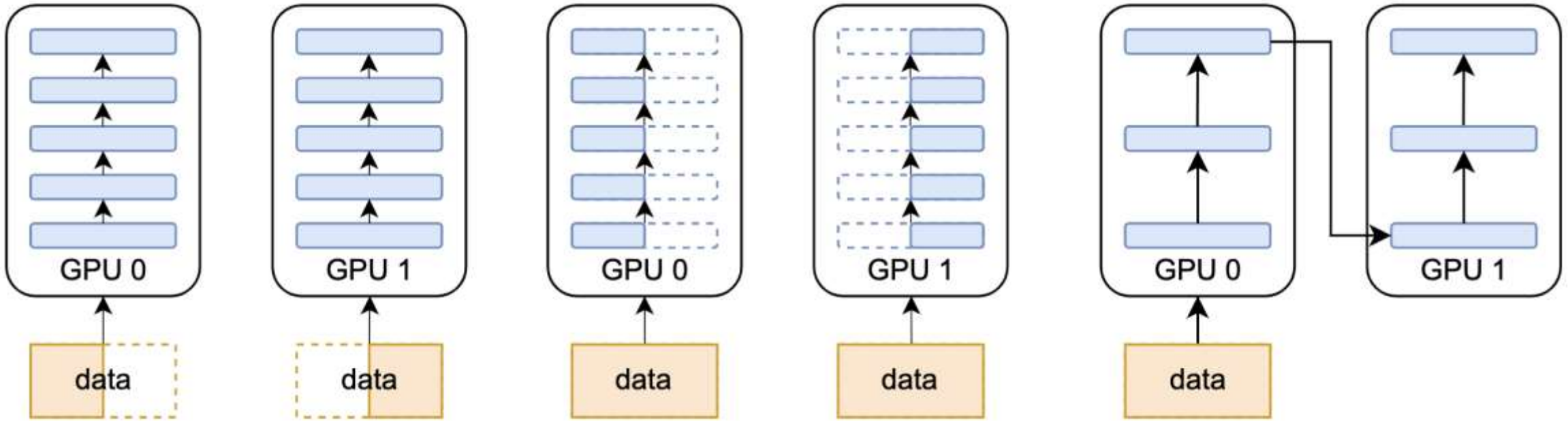
- Pro: Higher pipeline efficiency with fewer pipeline bubbles.
- Con: More communication overhead between stages.



Bubble: (P - 1) / (P - 1 + M) / V

Data & Model Parallelism: Summary

- Data parallel: same model, different data
- Tensor parallel: same data, part of model
- Pipeline parallel: split different model layers into GPUs
  - Tensor / pipeline parallel are usually combined with data parallel and *zero-1*



(a) data parallel (b) tensor parallel (c) pipeline parallel

Training Very Large Language Models

Jie Tang, KEG, Tsinghua University

http://keg.cs.tsinghua.edu.cn/jietang

https://github.com/THUDM

Appendix

Outline

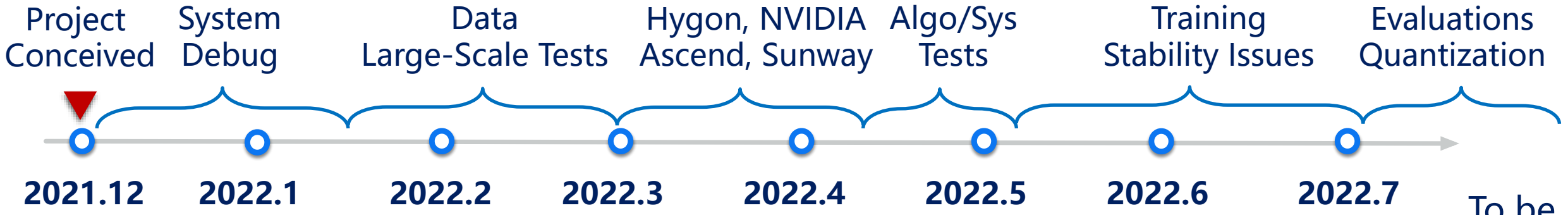
- How to train big models
  - Rematerialization
  - Data parallelism and ZeRO optimizer
  - Model parallelism
    - Inter-op parallelism: tensor parallelism
    - Intra-op parallelism: pipeline parallelism
- **Train 100B-scale models in action**
  - GLM-130B & LLAMA 3.1
- Data behind large language models
  - Common Crawl, WebText, etc.
  - Documentation for datasets

LLM: Challenge

- It is not easy to train and open-source an 100B……
  - **Outrageous Training Costs**
    - GPT-3: **10k V100s**, **$4.6M** for machine cost, **$12M** (estimated) total cost
  - **Mystery Algorithm**
    - Dramatic instability in training observed（BLOOM、OPT）
    - Very limited training details are disclosed from GPT-3/PaLM/Gopher……
  - **High Inference Threshold**
    - GPT-3 requires at least a DGX-A100 (8*80G) for offline inference

GLM-130B

- To date, 11 months have witnessed numerous challenges
  - **Engineering:** How to train 100B-scale models from scratch?
    - Hygon DCU, NVIDIA A100, Ascend 910, Sunway
    - Frequent & random hardware failures, Megatron-DeepSpeed 3D pipeline, CUDA kernel efficiency, GPU memory overflow, 10K+ threads TCP init & comms…
  - **Algorithm:** How to stabilize the training of 100B-scale models?
    - The gradient norms of embeddings, Native Post-LN, Pre-LN3, Sandwich-LN4, dataloader state seeds, computation precision choices in Softmax / Attention



http://keg.cs.tsinghua.edu.cn/glm-130b/

GLM-130B: Advantages

- **Bilingual:** supports both English and Chinese.
- **Performance (EN):** better than GPT-3 175B (+4.0%), OPT-175B (+5.5%), and BLOOM-176B (+13.0%) on LAMBADA and slightly better than GPT-3 175B (+0.9%) on MMLU.
- **Performance (CN):** significantly better than ERNIE TITAN 3.0 260B on 7 zero-shot CLUE datasets (+24.26%) and 5 zero-shot FewCLUE datasets (+12.75%).
- **Fast Inference:** supports fast inference on both SAT and FasterTransformer (up to 2.5X faster) with a single A100 server.
- **High Usability**：INT4 Weight Quantization, allowing inference on 4 x 3090 or 8 x 1080 Ti
- **Reproducibility:** all results (30+ tasks) can be easily reproduced with open-sourced code and model checkpoints.
- **Cross-Platform:** training and inference on NVIDIA, Hygon DCU, Ascend 910, and Sunway.

GLM-130B: Experiments for Architecture

- DeepNorm: a method to train 1000 layers of Post-LN stably

$$\mathrm{DeepNorm}(x) = \mathrm{LayerNorm}(\alpha \cdot x + g(x)), \quad \alpha > 1$$

- Rotary Position Encoding(RoPE): relative position encoding

$$(\boldsymbol{R}_m q)^\top (\boldsymbol{R}_n k) = q^\top \boldsymbol{R}_m^\top \boldsymbol{R}_n k = q^\top \boldsymbol{R}_{n-m} k$$

- Gated Linear Unit (GLU): replacement of FFN layer to steadily improve model performance

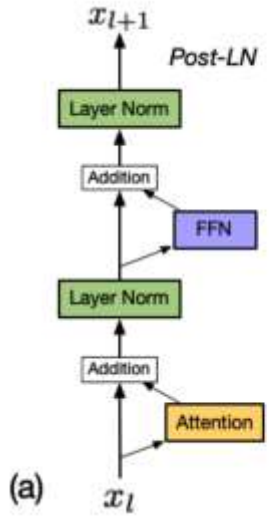
$$\mathrm{FFN}_{\mathrm{GLU}}(x, \boldsymbol{W}, \boldsymbol{V}, \boldsymbol{W}_2) = (\sigma(xW_1) \otimes x\boldsymbol{V})\boldsymbol{W}_2$$

	RTE	COPA	BoolQ	WSC	Avg
glm-base (original paper)	71.2	71.0	77.0	72.1	72.825
glm-base-geglu-postln	**72.80±1.04**	74.00±0.82	76.34±0.15	74.36±0.91	74.375
glm-base-geglu-deepnorm	70.16±0.85	**78.33±0.47**	76.93±0.14	71.79±0.91	74.3025
glm-base-geglu-preln	71.24±1.04	75.33±2.49	76.75±0.17	**80.77±2.72**	**76.0225**
glm-base-geglu-sandwich	71.00±0.61	77.00±1.63	**77.24±0.43**	78.21±1.81	75.8625
glm-base-gau-postln	69.43±1.70	71.33±1.25	76.33±0.24	75.00±1.57	73.0225
glm-base-gau-preln	diverged				
glm-base-gau-preln-0.1-shrink	71.84±1.06	75.33±1.89	75.80±0.34	76.28±3.27	74.8125
glm-base-gau-sandwich	69.92±0.61	75.67±0.94	77.00±0.15	72.44±1.81	73.7575
glm-base-sandwich	71.00±0.74	72.33±1.70	76.75±0.05	73.72±2.40	73.45

small-scale experiments for optimal architecture

Model Structure：Pre-LN/Post-LN？

- Original BERT[1] use Post-LN
  - Post-LN is easy to diverge, In recent years, models generally adopt Pre-LN[2]
- Pre-LN is also unstable in 10B+ billions / multi-modality training
- Sandwich-LN[3] can alleviate the phenomenon



1. Xiong, Ruibin, et al. "On layer normalization in the transformer architecture." International Conference on Machine Learning. PMLR, 2020
2. Ding, Ming, et al. "Cogview: Mastering text-to-image generation via transformers." Advances in Neural Information Processing Systems 34 (2021).
3. Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." NAACL, 2019

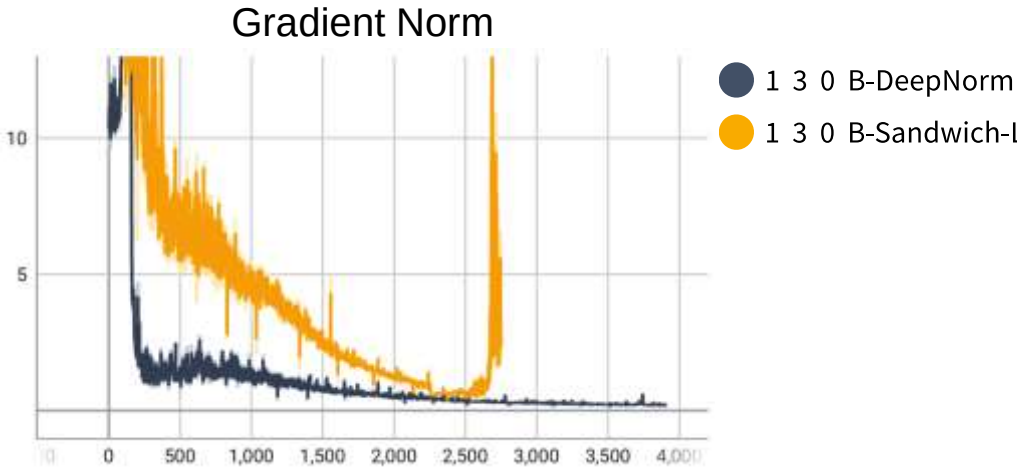
Model Architecture: Pre-LN/Post-LN ?

- Pre-LN is easy to train, but its performance is not as good as Post-LN after stable training[1]
  - Pre-LN constant path is good for training, but the front layer gradient is larger, which is not good for finetuning
- DeepNet[2]： Adjust the residual error, change the initialization → stabilize 1000 layer Post-LN

$$\mathrm{DeepNorm}(x) = \mathrm{LayerNorm}(\alpha \cdot x + g(x)), \quad \alpha > 1$$

- 100B models: DeepNorm seems to be more stable than Sandwich LN, so

SQuAD	Public	Post-LN	Pre-LN	RealFormer
v1.1 (F1)	90.9	91.68±0.12	91.06±0.09	**91.93**±0.12
v1.1 (EM)	84.1	85.15±0.13	83.98±0.24	**85.58**±0.15
v2.0 (F1)	81.9	82.51±0.12	80.30±0.12	**82.93**±0.05
v2.0 (EM)	78.7	79.57±0.12	77.35±0.16	**79.95**±0.08

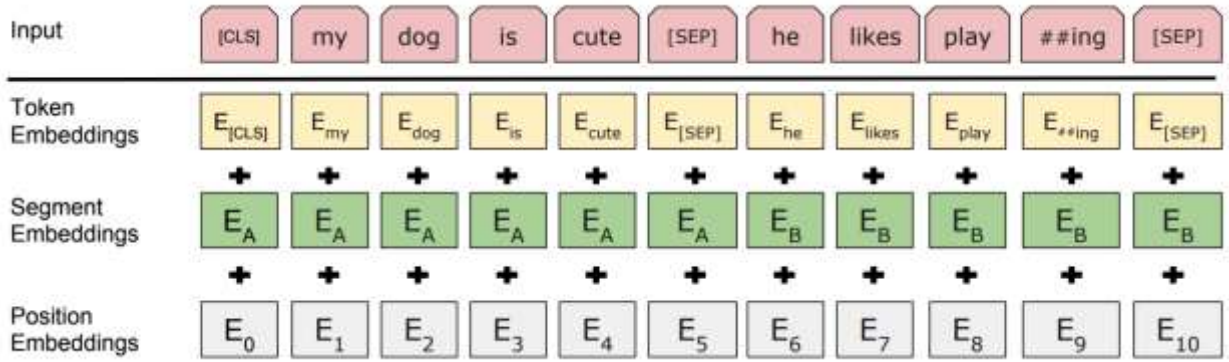


1. He, Ruining, et al. "Realformer: Transformer likes residual attention." arXiv preprint arXiv:2012.11747 (2020).
2. Wang, Hongyu, et al. "Deepnet: Scaling transformers to 1,000 layers." arXiv preprint arXiv:2203.00555 (2022)

Model Architecture: Absolute/Relative PE

- Absolute position encoding: sinusoidal[1], learnable[2]

$$\begin{cases} \boldsymbol{p}_{k,2i} = \sin\left(k/10000^{2i/d}\right) \\ \boldsymbol{p}_{k,2i+1} = \cos\left(k/10000^{2i/d}\right) \end{cases}$$



- Relative position encoding: Relative distance between pairs of modeling words in Attention

$$\boldsymbol{q}_i\boldsymbol{k}_j^\top = (\boldsymbol{x}_i + \boldsymbol{p}_i)\boldsymbol{W}_Q\boldsymbol{W}_K^\top(\boldsymbol{x}_j + \boldsymbol{p}_j)^\top = (\boldsymbol{x}_i\boldsymbol{W}_Q + \boldsymbol{p}_i\boldsymbol{W}_Q)\left(\boldsymbol{W}_K^\top\boldsymbol{x}_j^\top + \boldsymbol{W}_K^\top\boldsymbol{p}_j^\top\right)$$

$$a_{i,j} = \mathrm{softmax}\left(\boldsymbol{x}_i\boldsymbol{W}_Q\left(\boldsymbol{x}_j\boldsymbol{W}_K + \boldsymbol{R}_{i,j}^K\right)^\top\right)$$

Google[3]

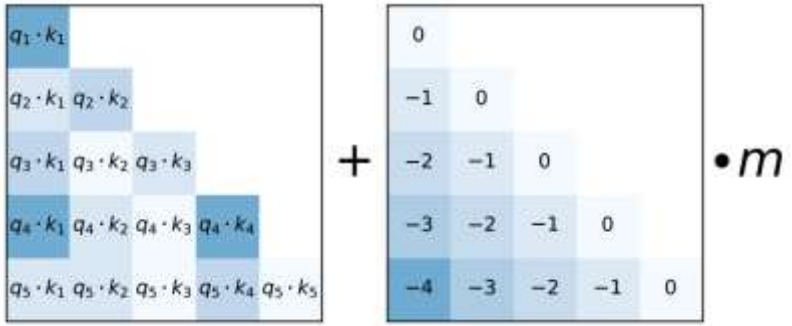
$$\boldsymbol{x}_i\boldsymbol{W}_Q\boldsymbol{W}_K^\top\boldsymbol{x}_j^\top + \boldsymbol{x}_i\boldsymbol{W}_Q\boldsymbol{W}_K^\top\boldsymbol{R}_{i-j}^\top + \boldsymbol{u}\boldsymbol{W}_Q\boldsymbol{W}_K^\top\boldsymbol{x}_j^\top + \boldsymbol{v}\boldsymbol{W}_Q\boldsymbol{W}_K^\top\boldsymbol{R}_{i-j}^\top$$

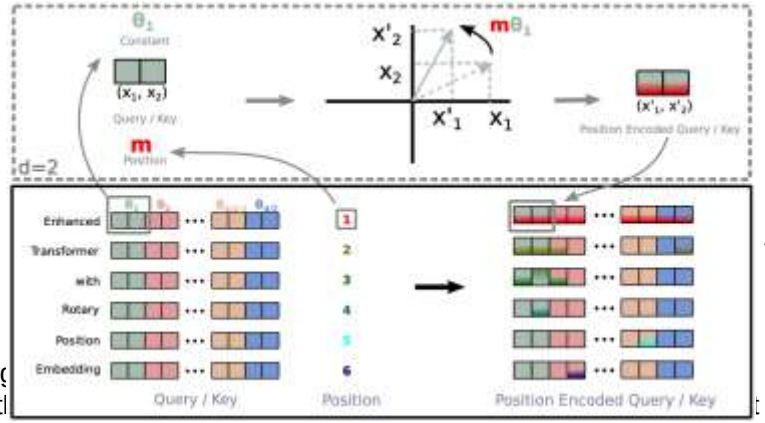
Transformer-XL[4]

1. Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).
2. Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." NAACL, 2019
3. Shaw, Peter, Jakob Uszkoreit, and Ashish Vaswani. "Self-attention with relative position representations." arXiv preprint arXiv:1803.02155 (2018).
4. Dai, Zihang, et al. "Transformer-XL: Attentive Language Models beyond a Fixed-Length Context." Proceedings of the 57th Annual Meeting of the Association for Computational

Model Architecture: ALiBi/RoPE

- ALiBi: Add a bias matrix to attention score matrix
  - BigScience found that ALiBi significantly improved the performance of Zero shot, and 176B model adopted
- RoPE: Rotary encoding, realize relative encoding using absolute encoding
  - Used in EleutherAI and Google PaLM 530B to improve stable performance
- Experiments show that RoPE is more effective for GLM and easier to achieve bi-directional relative attention





$$(\boldsymbol{R}_m q)^\top (\boldsymbol{R}_n k) = q^\top \boldsymbol{R}_m^\top \boldsymbol{R}_n k = q^\top \boldsymbol{R}_{n-m} k$$

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix}$$

1. Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for lang
2. Press, Ofir, Noah A. Smith, and Mike Lewis. "Train short, test long: Attention with : arXiv:2108.12409 (2021).
3. Su, Jianlin, et al. "Roformer: Enhanced transformer with rotary position embedding." arXiv preprint arXiv:2104.09864 (2021).

GLM-130B: Core Technology

- Architecture of GLM-130B
  - Architecture：GLM（General Language Model）
  - Component improvements：RoPE， DeepNorm， GeGLU
- **Engineering implementation of GLM-130B**
  - Parallel strategy：Data, tensor, pipeline 3D parallelism
  - Efficient multi-platform adaptation
- **Training strategy of GLM-130B**
  - Training stability**:** The biggest challenge of large models
  - Gradient explosion: embedding layer gradient reduction strategy
  - Attention overflow: FP32 softmax computing strategy

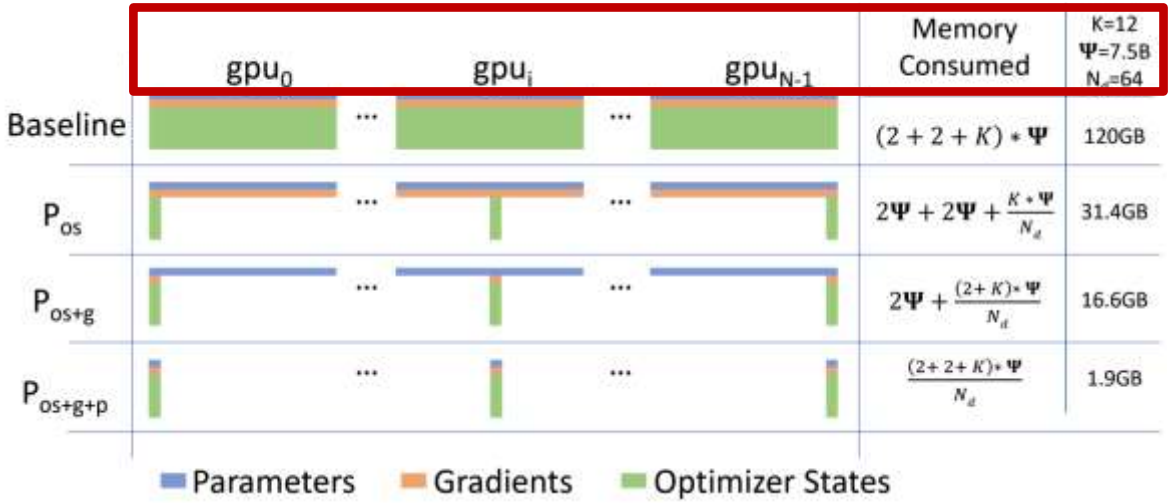
Parallel Strategy: Efficiently Training

- GPT-3 model requires 2.8T of gpu memory to store training state + intermediate activation function values

$$\underbrace{175\text{B}}_{\text{GPT-3 param.}} \times \left(\underbrace{2}_{\text{weights}} + \underbrace{2}_{\text{gradients}} + \underbrace{4}_{\text{Master weights}} + \underbrace{4+4}_{\text{Adam states}} \right) = 2.8 \text{ TB}$$

- Challenge: What parallel approach to efficient training with far more than a single card's gpu memory (40GB)?

1. Use ZeRO optimizer to share the optimizer state within the data parallel group → ~25%

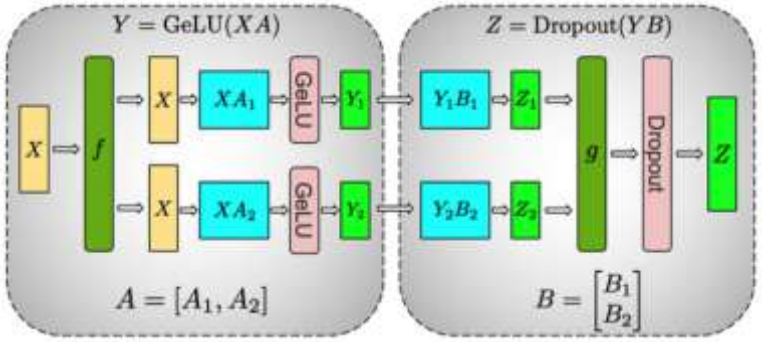


Parallel Strategy: Efficiently Training

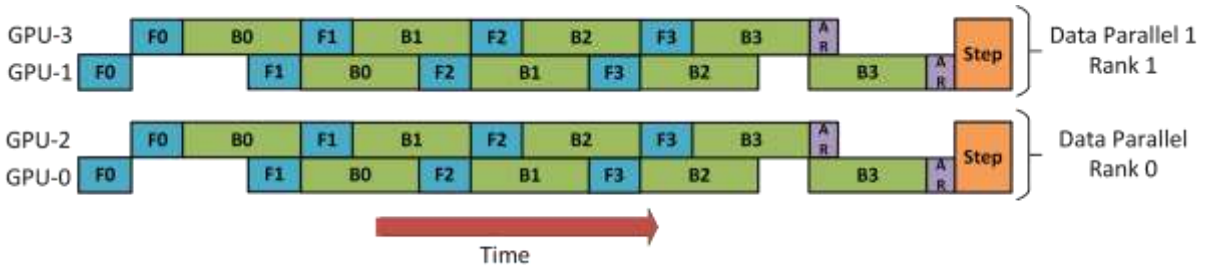
- Far exceeds the gpu memory of a single card, how to train efficiently?

  2. Model parallel: distributing model parameters to multiple GPUs

- Tensor parallel: slice and dice the parameter matrix and compute one part per GPU → additional communication to reduce computational granularity
- Pipeline parallel: split the network into multiple parallel segments → introduce pipeline bubbles



Tensor parallel



Pipeline parallel

Parallel Strategy: Efficiently Training

- Far exceeds the gpu memory of a single card, how to train efficiently? 2.

  Model parallel: distributing model parameters to multiple GPUs

  - Tensor parallel: slice and dice the parameter matrix and compute one part per GPU → additional communication to reduce computational granularity
  - Pipeline parallel: split the network into multiple parallel segments → introduce pipeline bubbles
  - ZeRO-3： Distribute parameters into data parallel groups, fetch parameters before computation → additional communication time

- Analysis： bubble share of pipeline： $\frac{n/t-1}{m+n/t-1}$， ignore when n / t << 4m

- **Parallel strategy: tensor parallelism scales slowly as the model size increases, but does not exceed the single machine size (<= 8), and the rest all use pipeline parallel to reduce the bubble share by adjusting the micro-batch size**

Parallel Strategy: Efficiently Training

- Other optimizations
  - Operator fusion: fuse multiple element-wise algorithms → **~10% computational speedup**
  - Pipeline balancing: place one less layer at the beginning and end of the pipeline to balance consumption → **~10% memory savings**

Parallel Strategy: Efficiently Training

- Test cluster configuration：
  - A100 Cluster: 96 DGX-A100 machines with 2 x 200GB IB NICs
  - Hygon DCU: 3000 machines with 4 DCU cards and 4 x 50G IB per machine
  - Sunway processor: 8192 nodes, one SW26010-PRO processor per node

Great variability in hardware

- Estimated time based on training a GPT-3 175B scale model with 300B words.

	Model	Nodes			Layer	BSZ	HFU	Time
集群名称	模型规模	机器数	TP[a]	PP[b]	层数	批处理	利用率	预计时间[c]
A100	176B	96	4	14	96	3584	41.06%	50 天
Hygon	183B	3000	4	25	100	6000	31.29%	64 天
Sunway	173B	8192	4	6	94	4096	18.39%	59 天
BMTrain-A100	176B	96	-	-	-	-	33.00%	62 天

[a] 张量模型并行规模　[b] 流水线模型并行规模

[c] 训练 300B = 300×10^9 单词量所需要的时间估计

High Util.

Reasonable

+25% Faster

Parallel Strategy: Efficiently Training

- Scalability experiments on Hygon DCU: training models of different sizes and recording their cumulative performance and hardware utilization according to the parallel strategy

Model Scale (Billion)	Node Number	Tensor Parallel	Pipeline Parallel	Hidden Size	Layer Num	Batch Size	Perf (PFLOPS)	Hardware Utilization
1.7	16	1	1	2304	24	512	0.44	31.83%
3.6	32	2	1	3072	30	512	0.83	30.06%
7.5	32	4	1	4096	36	512	0.72	26.34%
18.4	128	4	2	6144	40	1024	2.99	27.07%
39.1	256	4	4	8192	48	1536	7.18	32.46%
76.1	512	4	8	10240	60	2048	14.3	32.41%
145.6	1024	4	16	12288	80	3072	30.3	34.29%
301.1	2048	4	32	16384	96	4608	94.3	53.33%

Utilization Rate Gradually Improve

Parallel Strategy: Efficiently Training

- Overall shows **super linear scalability**: the larger the scale, the more efficient the computation
  - Model computational characteristics: model size increases ⇒ matrix multiplication share increases ⇒ utilization rate increases
  - Reasonable parallelism strategy: appropriate model parallel division and batch size control the bubble share of pipeline parallelism

Model Scale (Billion)	Node Number	Tensor Parallel	Pipeline Parallel	Hidden Size	Layer Num	Batch Size	Perf (PFLOPS)	Hardware Utilization
1.7	16	1	1	2304	24	512	0.44	31.83%
3.6	32	2	1	3072	30	512	0.83	30.06%
7.5	32	4	1	4096	36	512	0.72	26.34%
18.4	128	4	2	6144	40	1024	2.99	27.07%
39.1	256	4	4	8192	48	1536	7.18	32.46%
76.1	512	4	8	10240	60	2048	14.3	32.41%
145.6	1024	4	16	12288	80	3072	30.3	34.29%
301.1	2048	4	32	16384	96	4608	94.3	53.33%

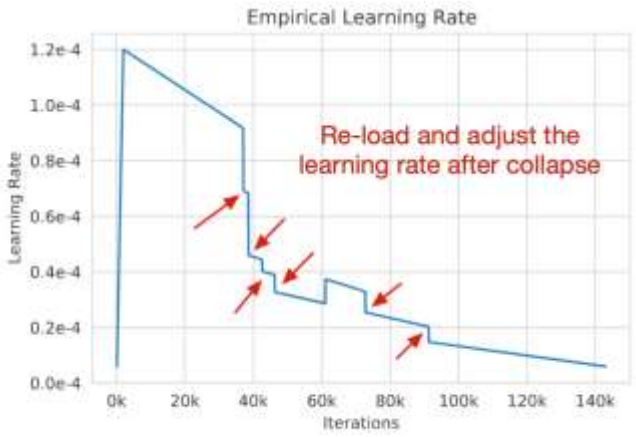
Utilization Rate Gradually Improve

GLM-130B: Core Technology

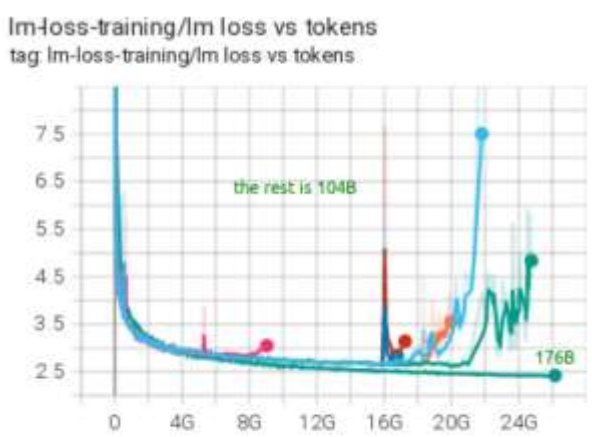
- **Architecture of GLM-130B**
  - Architecture：GLM（General Language Model）
  - Component improvements：RoPE, DeepNorm, GeGLU
- **Engineering implementation of GLM-130B**
  - Parallel strategy：Data, tensor, pipeline 3D parallelism
  - Efficient multi-platform adaptation
- **Training strategy of GLM-130B**
  - Training stability**:** The biggest challenge of large models
  - Gradient explosion: embedding layer gradient reduction strategy
  - Attention overflow: FP32 softmax computing strategy

Most fatal challenge: Stability

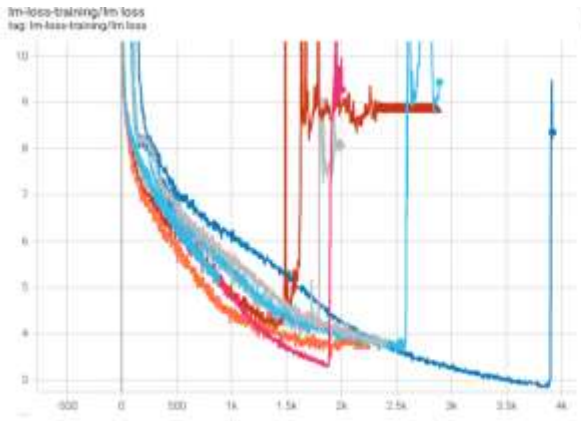
- Tradeoff: Stability (Slow) or Efficiency (Instable)
- Existing Solutions
  - **OPT-175B:** manually adjust LR & skip data when collapses (performance drop)
  - **BLOOM 176B**: embedding norm & BF16（performance drop, few platform）



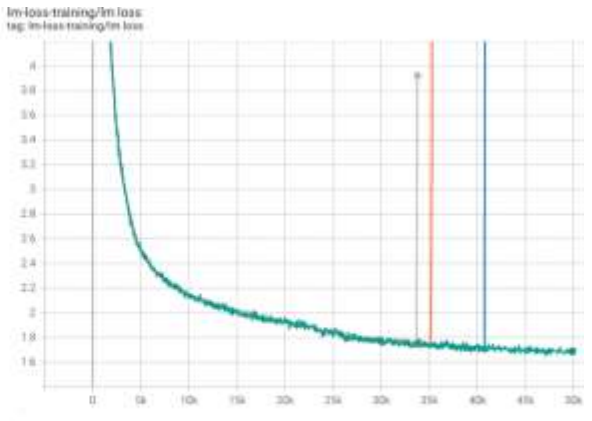
(a) OPT 175B's experiments



(b) BLOOM 176B's experiments

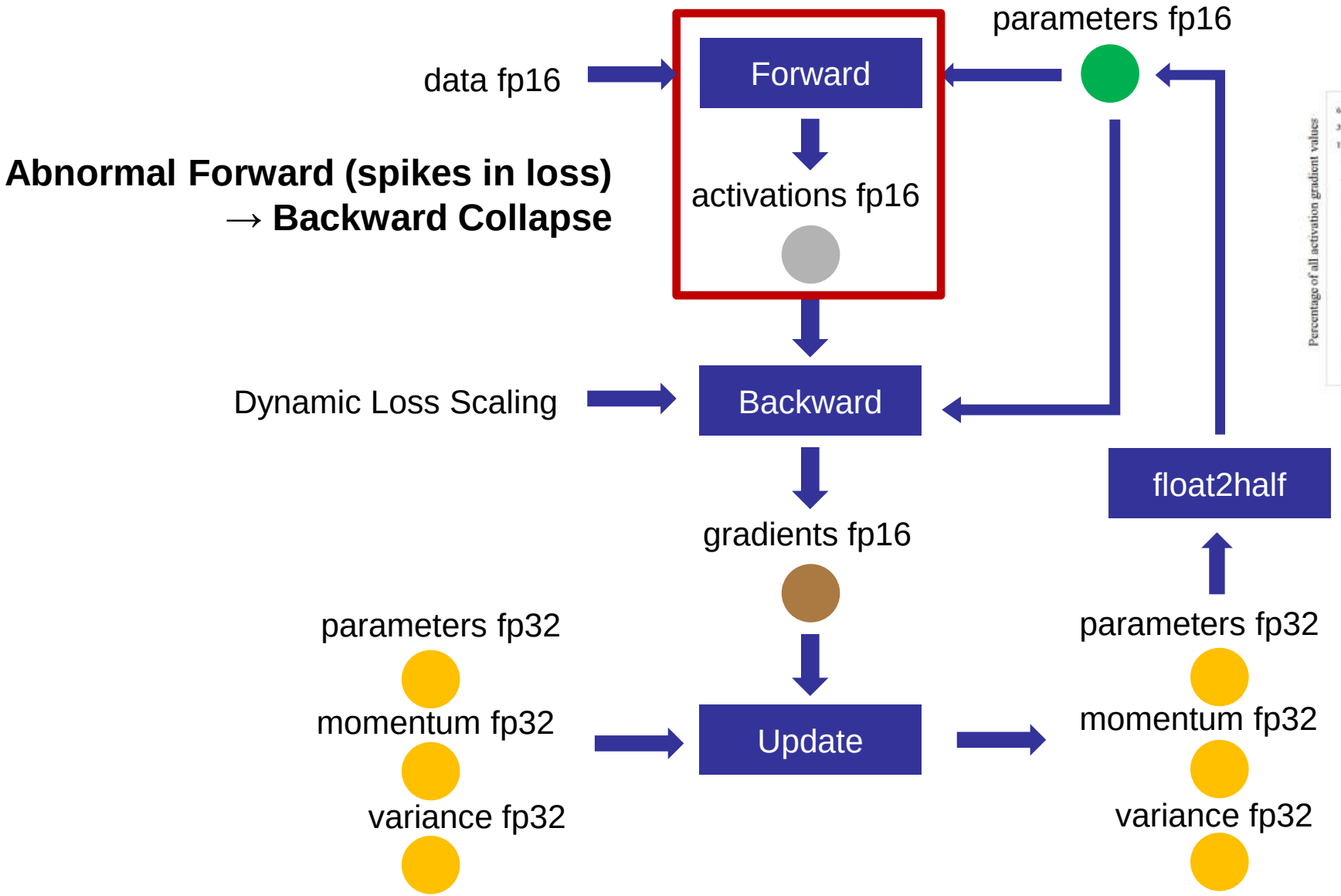


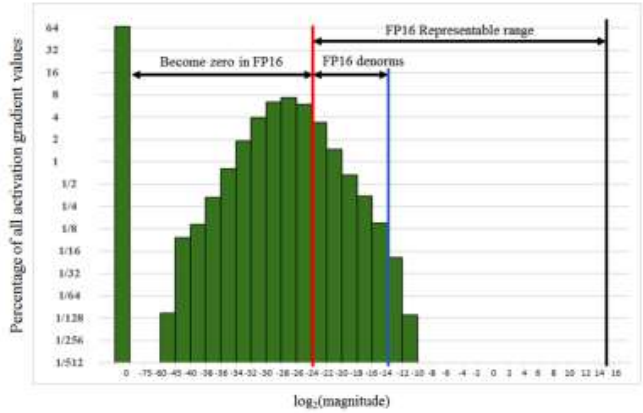
(c) GLM 130B's experiments



(c) GLM 130B's real training

Mixed-precision Training





1. Micikevicius, Paulius, et al. "Mixed Precision Training." International Conference on Learning Representations. 2018.

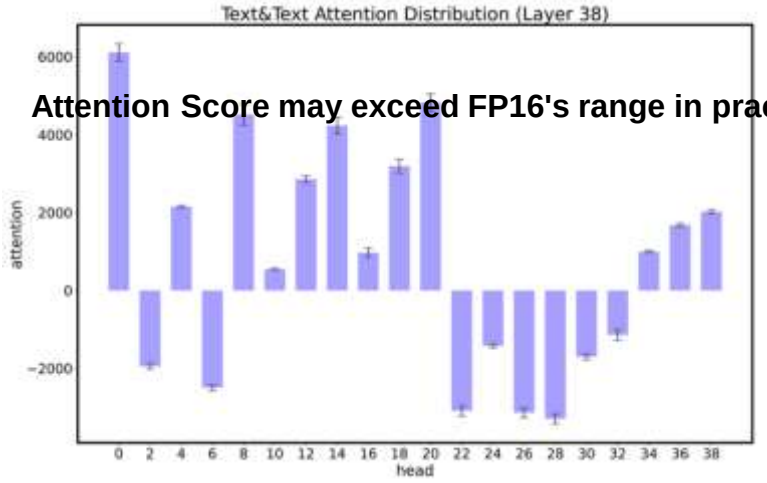
GLM-130B: Stabilizing Strategy

- Attention score: Softmax in 32 to avoid overflow

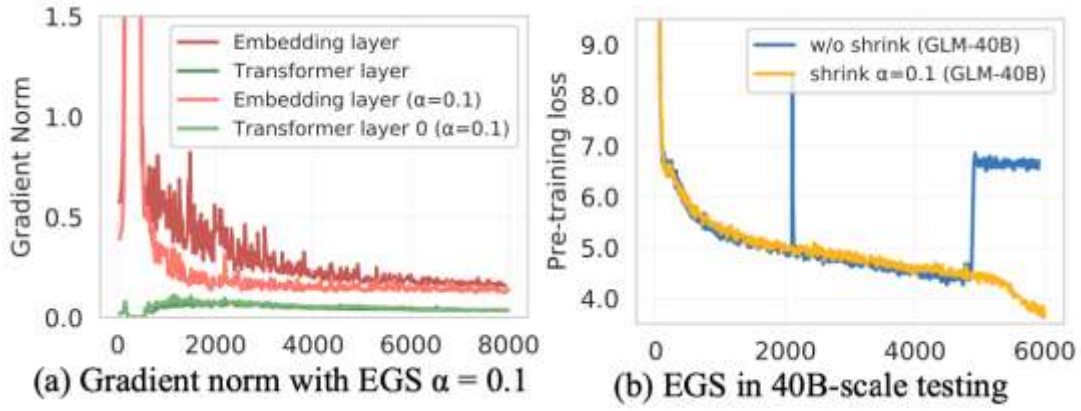
$$\text{softmax}\left(\frac{\boldsymbol{Q}_i\boldsymbol{K}_i^\top}{\sqrt{d}}\right) = \text{softmax}\left(\left(\frac{\boldsymbol{Q}_i\boldsymbol{K}_i^\top}{\alpha\sqrt{d}} - \max\left(\frac{\boldsymbol{Q}_i\boldsymbol{K}_i^\top}{\alpha\sqrt{d}}\right)\right)\times\alpha\right) = \text{FP16}\left(\text{softmax}\left(\text{FP32}\left(\frac{\boldsymbol{Q}_i\boldsymbol{K}_i^\top}{\alpha\sqrt{d}}\right)\times\alpha\right)\right)$$

- Embedding Layer Gradient Shrink (EGS):

```
word_embedding = word_embedding * alpha + \
                 word_embedding .detach() * (1 – alpha)
```



Attention Score may exceed FP16's range in practice



Embedding Layer gradients can be magnitudes larger than others

The 100B-Scale Models

Model	Open-sourced	Architecture					
		Major Lang.	Back-bone	Training Objective	Layer-Norm	PE	FFN
GPT-3 175B	×	English	GPT	SSL only	Pre-LN	APE	FFN
OPT-175B	✓					APE	FFN
PaLM-540B	×					RoPE	SwiGLU
BLOOM-176B	✓	Multi-lingual	GPT	SSL only	Pre-LN	ALiBi	FFN
GLM-130B	✓	Bilingual (English & Chinese)	GLM	SSL & MIP	Deep-Norm	RoPE	GeGLU
Effect	An open LLM for everyone	Less Bias & Toxicity: StereoSet: +12.7% CSP(↓): -1.4% RTP(↓): -3.0%	Perf. Improvements: BIG-bench-lite: +5.2% LAMBADA: +2.3% CLUE: +24.3% FewCLUE: +12.8%		Deep-Norm improves stability in training	RoPE works better with GLM	GeGLU performs better than FNN

Aohan Zeng, Xiao Liu, et al., GLM-130B: An open bilingual pre-trained models. arXiv: 2010.02414

The 100B-Scale Models

Model	Training		Inference & Evaluation			
	Floating-point	Stabilization	Quantization	Acceleration	Reproducibility	Cross-Platform
GPT-3 175B	FP16	*undisclosed*	*undisclosed*	*undisclosed*	×	NVIDIA
OPT-175B	FP16	Hand-tuning	INT8	Megatron	×	NVIDIA
PaLM-540B	BF16	Hand-tuning	*undisclosed*	*undisclosed*	×	*undisclosed*
BLOOM-176B	BF16	Embedding Norm	INT8	Megatron	×	NVIDIA
GLM-130B	FP16	Embedding Gradient Shrink (EGS)	INT4	Faster-Transformer	✓	• NVIDIA • Hygon DCU • Ascend 910 • Sunway
Effect	FP16 supports more computing platforms	EGS improves numerical stability with little accuracy loss	It saves 75% mem in inference. 4 × 3090 or 8 × 2080 Ti.	×7-8.4 faster than Pytorch, ×2.5 faster than Megatron	All evaluation data & scripts are open	It supports more diverse adoption of LLMs

Aohan Zeng, Xiao Liu, et al., GLM-130B: An open bilingual pre-trained models. arXiv: 2010.02414

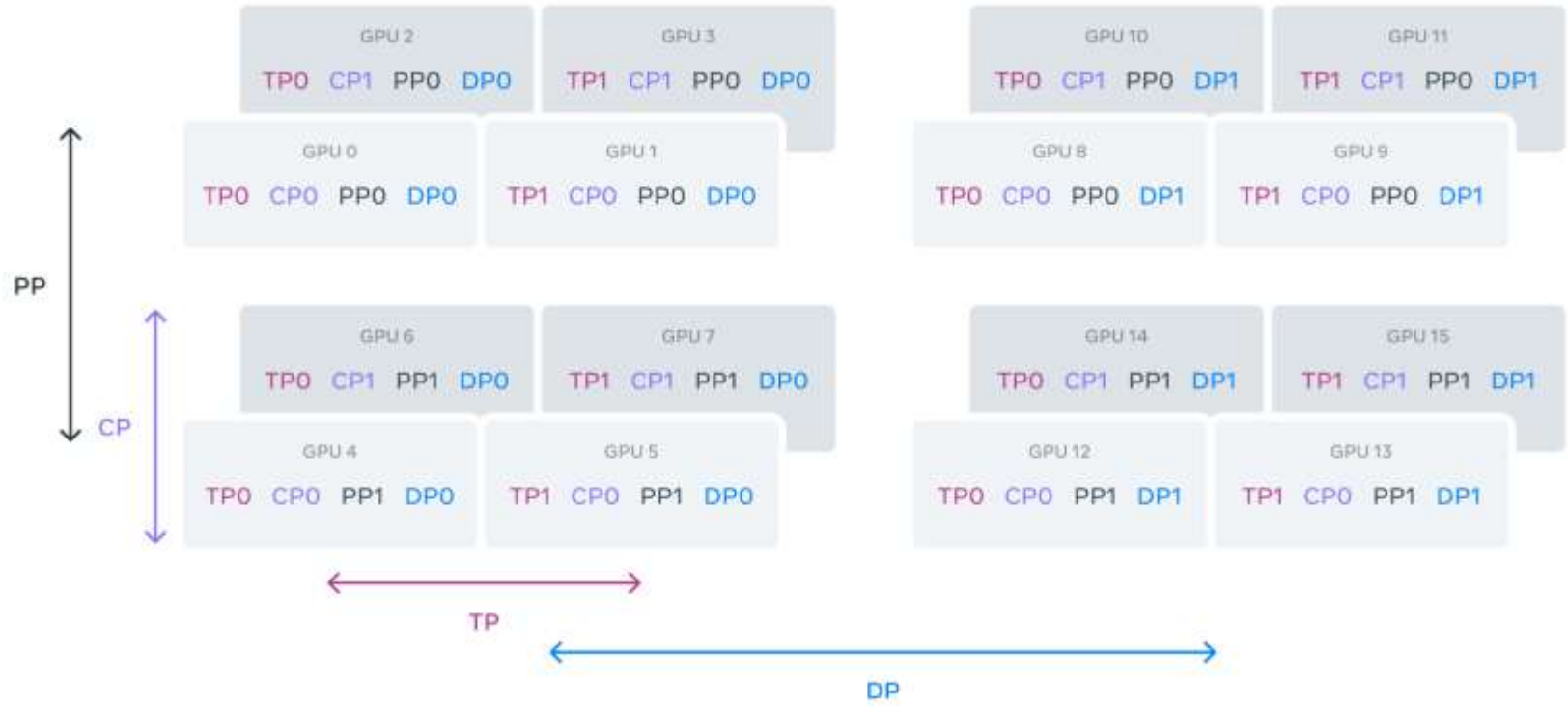
LLAMA 3.1 405B

- **Training infrastructure**
  - Compute, storage and Network
- **Parallelism Strategy**
  - Parallel strategy：4D parallelism
  - Improvements on pipeline parallelism challenges
  - Context parallelism for long sequences
- **Training strategy**
  - Training stability: numerical fix
  - Cluster stability: automated intervention
  - Environmental factors on training performance

LLAMA 3.1: Training Infrastructure

- Compute
  - Trained on **up to 16K H100 GPUs**, 700W TDP with 80GB HBM3
  - Each server: 8 GPUs, connected with NVLink
- Storage
  - Tectonic distributed file system, **240PB** from 7500 servers with SSDs
  - Sustainable throughput: **2TB/s**, peak throughput: **7TB/s**
  - Major challege: **ckpts saving**, high and burst writing of storage
- Network
  - 405B: RoCE-based, 24K GPUs connected by a three-layer Clos network
  - **Load balancing**: reducing the traffic per flow, Enhanced-ECMP protocol
  - **Congestion control**: deep-buffer switches in the spine

LLAMA 3.1: Parallelism Strategy



GPUs are divided into groups in the order of **[TP, CP, PP, DP]**

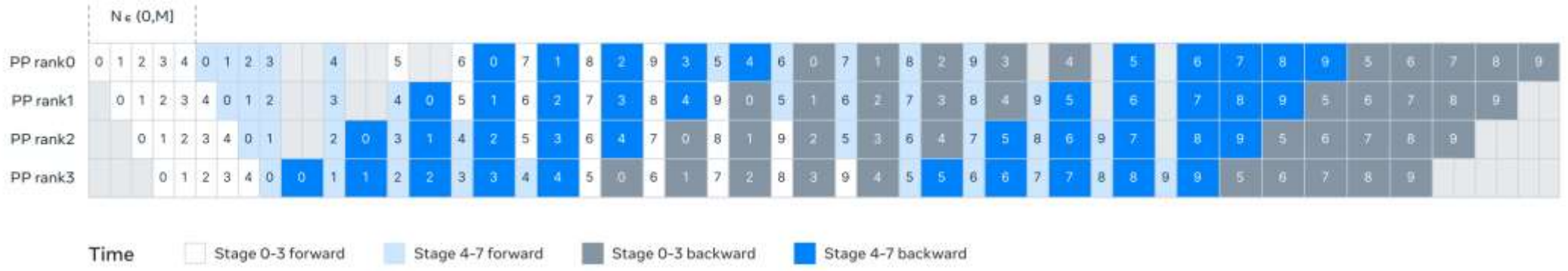
- representing GPU with para index: [D1, D2, D3, D4]
- TP group[GPU0, GPU1], CP group[GPU0, GPU2],
  PP group[GPU0, GPU4], DP group[GPU0, GPU8]

LLAMA 3.1: Parallelism Strategy

Improvements on pipeline parallelism

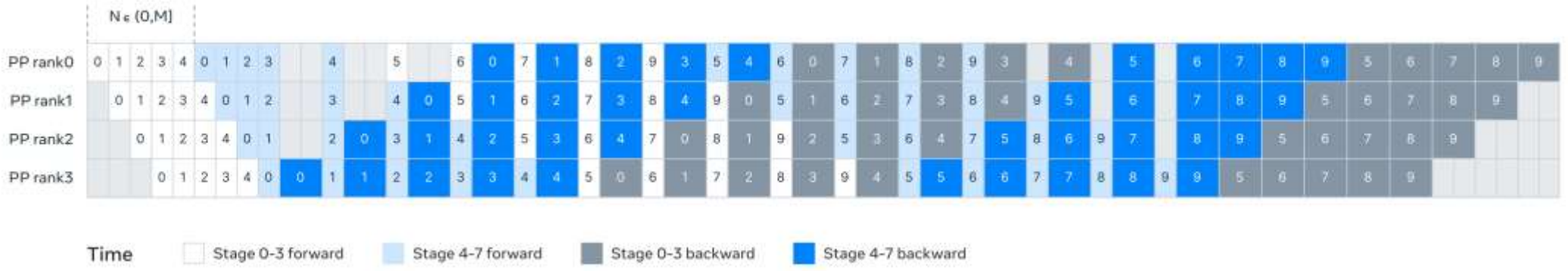
- Previous challenges
  - **Batch size constraints**: not able to flexibly adjust batch size
  - **Memory imbalance**: the first stage is with embedding
  - **Computation imbalance**: the last stage is with output and loss calculation

Pipeline parallelism in LLAMA 3:



LLAMA 3.1: Parallelism Strategy

Pipeline parallelism in LLAMA 3:



- Allowing setting N flexibility
  - fewer micro-batches when having batch size limit at large scale
  - when we have batch size limit at large scale
- Reduce one Transformer layer each from the first and the last stages (to balance)
- An interleaved schedule with V pipeline stages on one pipeline rank
- Asynchronous point-to-point communication in PP

LLAMA 3.1: Parallelism Strategy

- **Context Parallelism**
- improve memory efficiency and **enable training on extremely long sequences up to 128K**.
- partition the input sequence into 2 × CP chunks, each CP rank receives 2 for better load balancing.
- Differences: first all-gather K and V, then compute attention output for the local Q
  - Previous: overlap communication and computation in a ring-like structure
  - **Advantage 1**: easier and flexible to support various attention masks
  - **Advantage 2**: the exposed all-gather latency is small due to GQA

LLAMA 3.1: Training Strategy

- Training stability
  - Use FP32 gradient accumulation and reduce-scatter gradients
  - FP32 gradients for multiple times forward modules (vision inputs, etc.)
- Cluster stability: automated intervention
  - 466 job interruptions during 54 days, 47 planned, 419 unexpected
  - 78% confirmed hardware issues, 58.7% GPU issues from unexpect issues
  - reducing startup times, traciing communication events, auto timeout and restart
- Environmental factors on training performance
  - a diurnal 1-2% throughput variation based on time-of-day, resulting from higher mid-day temperatures impacting GPU dynamic voltage and frequency scaling.

Outline

- How to train big models
  - Intro-model optimization:
    - Mixed Precision & Rematerialization
  - Data parallelism and ZeRO optimizer
  - Model parallelism
    - Inter-op parallelism: tensor parallelism
    - Intra-op parallelism: pipeline parallelism
- Train 100B-scale models in action
  - GLM-130B & LLAMA 3.1
- **Data behind large language models**
  - Common Crawl, WebText, etc.
  - Documentation for datasets

Data behind large language models

- Large language models are trained on “raw text”. To be highly capable (e.g., have linguistic and world knowledge), this text should span a **broad** range of domains, genres, languages, etc.
- **Web**: a major source.
  - Huge. the Google search index is 100 petabytes (reference).
- **Private datasets** that reside in big companies are even larger than what’s available publicly.
  - WalMart generates 2.5 petabytes of data each hour!

Data behind large language models

- Common Crawl
- WebText and OpenWebText
- Colossal Clean Crawled Corpus (C4)
- GPT-3 dataset
- The Pile

Common Crawl

- **Common Crawl** is a nonprofit organization that crawls the web and provides snapshots that are free to the public.
- Scale: The April 2021 snapshot of Common Crawl has 320 terabytes of data
- Applications: A standard source of data to train many models such as T5, GPT-3, and Gopher.

Common Crawl

Representation harms. Despite the richness of web data, it has been noted in Bender et al, 2021 that:

- Despite the size, large-scale data still has **uneven representation** over the population.
- Internet data overrepresents younger users from developed countries.
- GPT-2's training data is based on Reddit, which according to Pew Internet Research's 2016 survey, 67% of Reddit users in the US are men, 64% between ages 18 and 29.
- 8.8-15% of Wikipedians are female.
- Filtering "bad words" could further marginalize certain populations (e.g., LGBT+).

Takeaway: it is crucial to understand and document the composition of the datasets used to train large language models.

WebText

- WebText: **dataset used in training GPT-2**
- Goal: obtain **diverse but high-quality** dataset.
- Previous work:
  - Datasets were trained on news, Wikipedia, or fiction.
  - Common Crawl contains a lot of junk (gibberish, boilerplate text).
  - Trinh & Le, 2018 selected a tiny subset of Common Crawl based on n-gram overlap with the target task.
- Process for creating WebText:
  - Scraped all outbound links that received at least 3 karma (upvotes).
  - Filtered out Wikipedia to be able to evaluate on Wikipedia-based benchmarks.
  - End result is 40 GB of text.
- WebText was not released by OpenAI

WebText and OpenWebText

- OpenWebText: WebText was replicated (in spirit) by the OpenWebText dataset.
- Process for creating OpenWebText:
  - Extracted all the URLs from the Reddit submissions dataset.
  - Used Facebook's fastText to filter out non-English.
  - Removed near duplicates.
  - End result is 38 GB of text.

Dataset: WebText and OpenWebText

- **Toxicity analysis**.
  - RealToxicityPrompts (Gehman et al. 2020) : a dataset of 100K naturally occurring, sentence-level prompts derived from a large corpus of English web text, paired with toxicity scores from a widelyused toxicity classifier.
  - Analyzed these two datasets and found:
    - 2.1% of OpenWebText has toxicity score >= 50%
    - 4.3% of WebText (from OpenAI) has toxicity score >= 50%
    - News reliability correlates negatively with toxicity (Spearman $\rho = -0.35$)
    - 3% of OpenWebText comes from banned or quarantined subreddits, e.g., /r/The_Donald and /r/WhiteRights
- Takeaway: We can construct datasets to evaluate toxicity score of large datasets / models.

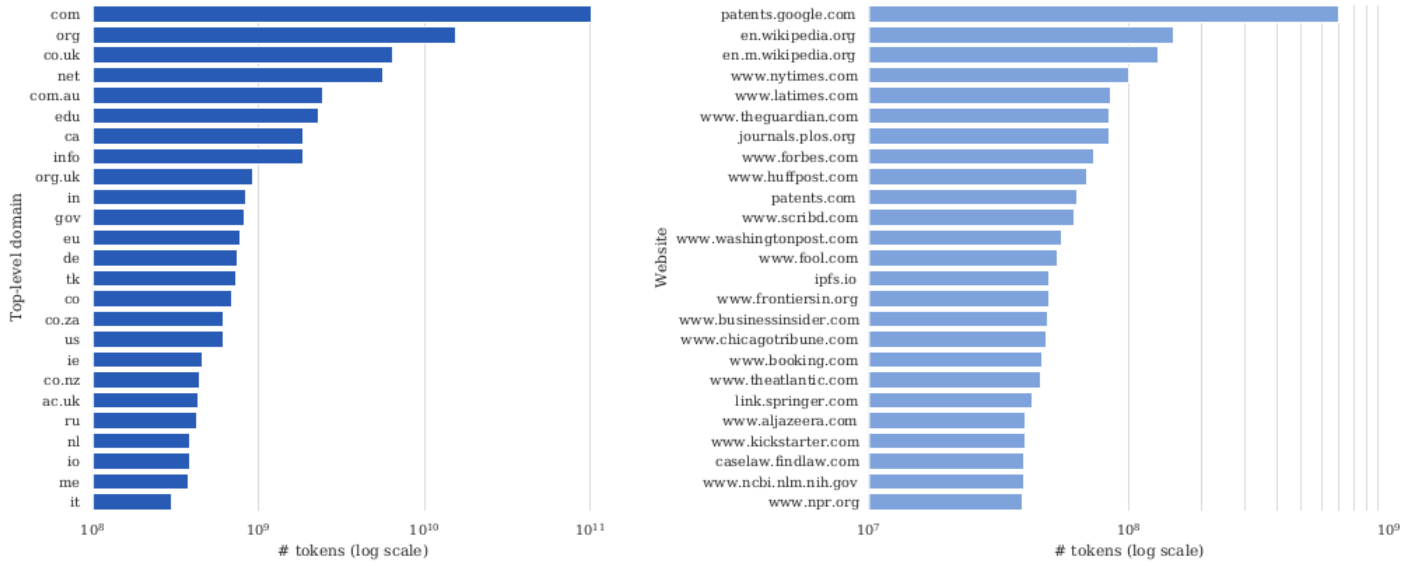
Colossal Clean Crawled Corpus (C4)

- The Colossal Clean Crawled Corpus (C4) is a larger was created to train the T5 model.
- Processes for constructing C4
  - Started with April 2019 snapshot of Common Crawl (1.4 trillion tokens)
  - Removed “bad words”
  - Removed code (“{“)
  - langdetect to filter out non-English text
  - Resulted in 806 GB of text (156 billion tokens)

Colossal Clean Crawled Corpus

Analysis. Dodge et al. 2021 performed a thorough analysis of the C4 dataset.

- A surprising amount of data from patents.google.com
- 65% pages in the Internet Archive; out of those, 92% pages written in the last decade
- 51.3% pages are hosted in the United States; fewer from India even though lots of English speakers there
- Some text from patents.google.com are automatically created, and thus have systematic errors:
  - Filed in a foreign country's official language (e.g., Japanese) is automatically translated into English
  - Automatically generated from optical character recognition (OCR)



Benchmark data contamination

- When we are evaluating the capabilities of large language models using benchmark data (e.g., question-answer pairs), it makes a difference whether the benchmark data appears in the training data of the language model. If so, then the benchmark performance will be **biased** up.
- Normally, in machine learning, data hygiene (keeping the training data separate from the test) is relatively easy, but **in the case of large language models**, both the training data and benchmark data are derived from the Internet, **it can be difficult to a priori guarantee their separation.**

Benchmark data contamination in C4

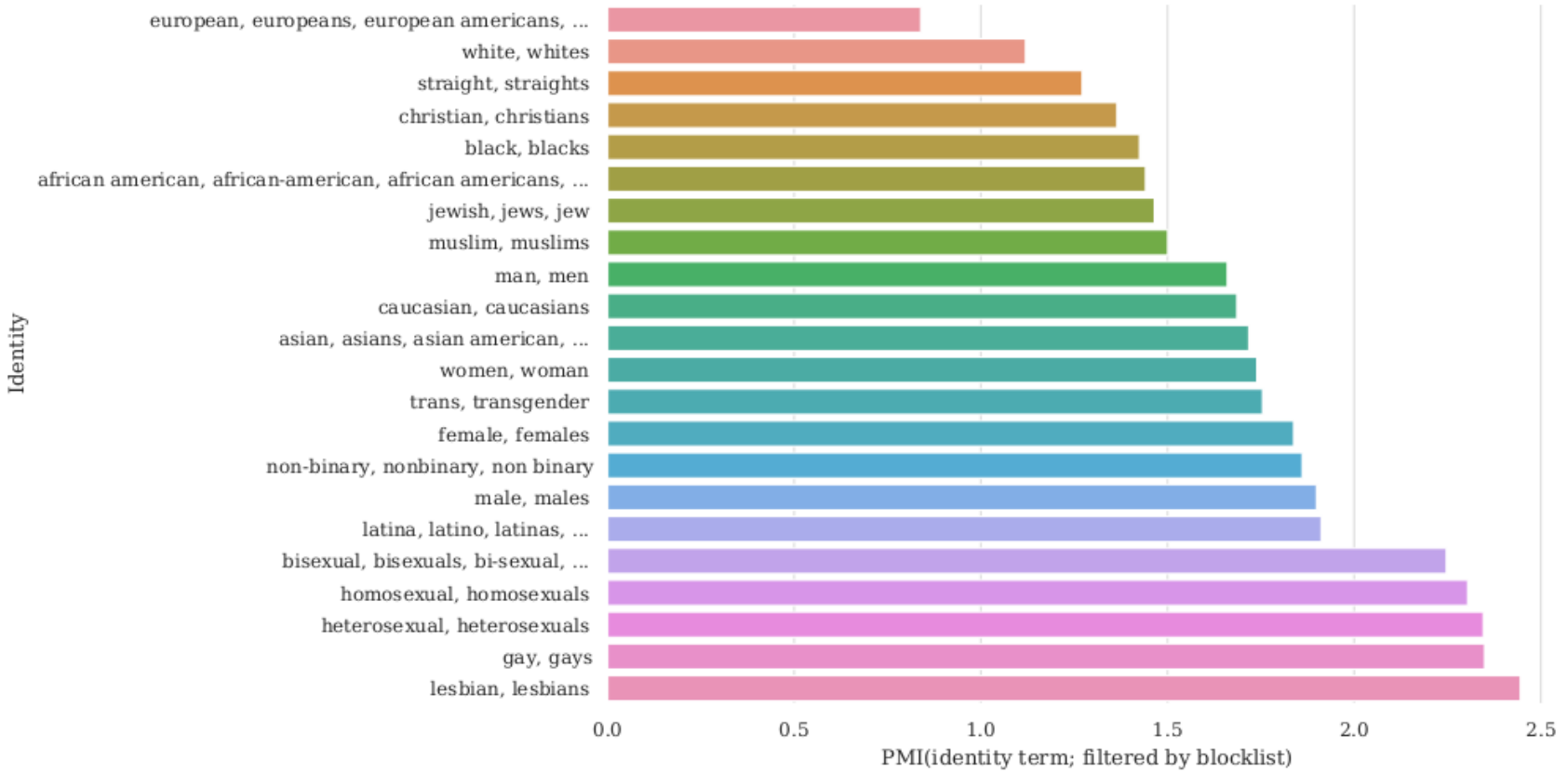
- There are two types of contamination:
  - **Input-and-output contamination**: both the input and output appear in the training data. Varies from 1.87% to 24.88% (XSum is 15.49%).
  - **Input contamination**: the input appears in the training data. Varies from 1.8% to 53.6% (QNLI, which is derived from Wikipedia).
- Note that contamination is not due to hosting datasets (as they are usually stored in a JSON file, not as a webpage).

Harms in C4

- **Representational harms**
  - They look at co-occurrence with ethnicity terms (e.g., *Jewish*) and sentiment-bearing words (e.g., *successful*).
  - *Jewish* has 73.2% positive sentiment, *Arab* has 65.7% positive (7.5% difference).
  - Variation across sites (New York Times had a 4.5% difference, Al Jazeera had 0% difference).
- **Allocational harms**
  - Recall C4 is a filtered version of Common Crawl (only about 10%).
  - Mentions of sexual orientations (e.g., *lesbian*, *gay*) more likely to be filtered out; of those filtered out, non-trivial fraction are non-offensive (e.g., 22% and 36%).
  - Certain dialects are more likely to be filtered (AAE: 42%, Hispanic-aligned English: 32%) than others (White American English: 6.2%)

Allocational harms in C4

- **Pointwise Mutual Information (PMI)** between identity mentions and documents being filtered out by the blocklist. Identities with higher PMI (e.g., lesbian, gay) have higher likelihood of being filtered out.



GPT-3 dataset

1. Selected subset of Common Crawl that's **similar to a reference dataset** (WebText).
   1. Downloaded 41 shards of Common Crawl (2016-2019).
   2. Trained a binary classifier to predict WebText versus Common Crawl.
   3. Sampled (kept) a document with higher probability if classifier deems it more similar to WebText.
2. Performed **fuzzy deduplication** (detect 13-gram overlap, remove window or documents if occurred in <10 training documents), removing data from benchmark datasets.
3. Expanded the diversity of the **data sources** (WebText2, Books1, Books2, Wikipedia).
4. During training, Common Crawl is **downsampled** (Common Crawl is 82% of the dataset, but contributes only 60%).

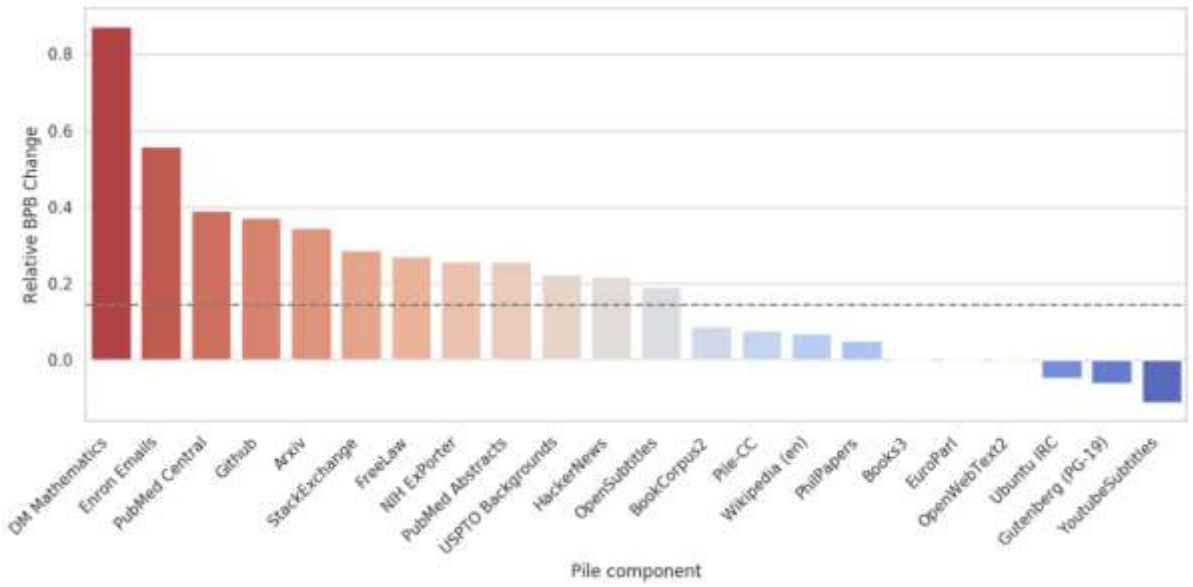
Dataset	Quantity (tokens)	Weight in training mix	Epochs elapsed when training for 300B tokens
Common Crawl (filtered)	410 billion	60%	0.44
WebText2	19 billion	22%	2.9
Books1	12 billion	8%	1.9
Books2	55 billion	8%	0.43
Wikipedia	3 billion	3%	3.4

The Pile

- While a web crawl is a natural place to look for broad data, it's not the only strategy, and GPT-3 already hinted that it might be productive to look at other sources of higher quality.
- EleutherAI (a nonprofit organization committed to building open language models), pushed this idea even farther. They released The Pile, a dataset for language modeling, where the key idea is to source it from smaller high-quality sources (academic + professional sources).
- **Data composition**.
  - 825 GB English text
  - 22 high-quality datasets

The Pile

- The key idea is to source it from smaller high-quality sources (academic + professional sources).
- contains a lot of information that's not well covered by GPT-3's



Component	Raw Size	Weight	Epochs	Effective Size	Mean Document Size
Pile-CC	227.12 GiB	18.11%	1.0	227.12 GiB	4.33 KiB
PubMed Central	90.27 GiB	14.40%	2.0	180.55 GiB	30.55 KiB
Books3†	100.96 GiB	12.07%	1.5	151.44 GiB	538.36 KiB
OpenWebText2	62.77 GiB	10.01%	2.0	125.54 GiB	3.85 KiB
ArXiv	56.21 GiB	8.96%	2.0	112.42 GiB	46.61 KiB
Github	95.16 GiB	7.59%	1.0	95.16 GiB	5.25 KiB
FreeLaw	51.15 GiB	6.12%	1.5	76.73 GiB	15.06 KiB
Stack Exchange	32.20 GiB	5.13%	2.0	64.39 GiB	2.16 KiB
USPTO Backgrounds	22.90 GiB	3.65%	2.0	45.81 GiB	4.08 KiB
PubMed Abstracts	19.26 GiB	3.07%	2.0	38.53 GiB	1.30 KiB
Gutenberg (PG-19)†	10.88 GiB	2.17%	2.5	27.19 GiB	398.73 KiB
OpenSubtitles†	12.98 GiB	1.55%	1.5	19.47 GiB	30.48 KiB
Wikipedia (en)†	6.38 GiB	1.53%	3.0	19.13 GiB	1.11 KiB
DM Mathematics†	7.75 GiB	1.24%	2.0	15.49 GiB	8.00 KiB
Ubuntu IRC	5.52 GiB	0.88%	2.0	11.03 GiB	545.48 KiB
BookCorpus2	6.30 GiB	0.75%	1.5	9.45 GiB	369.87 KiB
EuroParl†	4.59 GiB	0.73%	2.0	9.17 GiB	68.87 KiB
HackerNews	3.90 GiB	0.62%	2.0	7.80 GiB	4.92 KiB
YoutubeSubtitles	3.73 GiB	0.60%	2.0	7.47 GiB	22.55 KiB
PhilPapers	2.38 GiB	0.38%	2.0	4.76 GiB	73.37 KiB
NIH ExPorter	1.89 GiB	0.30%	2.0	3.79 GiB	2.11 KiB
Enron Emails†	0.88 GiB	0.14%	2.0	1.76 GiB	1.78 KiB
The Pile	**825.18 GiB**			**1254.20 GiB**	**5.91 KiB**

They also performed analysis of pejorative content, gender/religion biases. The findings are qualitatively similar to previous work.

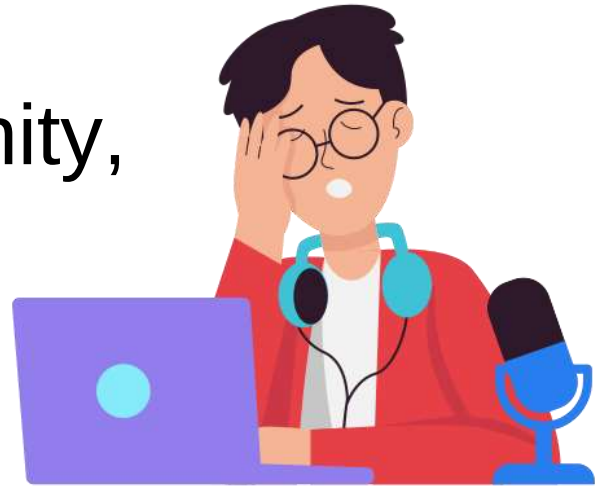
Takeaway

- The total amount of data out there (web, private data) is massive.
- Training on “all of it” (even Common Crawl) doesn’t work well (not effective use of compute).
- Filtering / curation (OpenWebText, C4, GPT-3 dataset) is needed, but can result in biases.
- Curating non-web high-quality datasets is promising (The Pile).
- Important to carefully document and inspect these datasets.

Documentation for datasets

- Documentation is important
  - Examples from other fields:
    - **Electronics industry** has a well-established protocol where every component has a datasheet with operating characteristics, test results, recommended and usage.
    - **Nutrition labels**: The FDA mandates that food be labeled with their nutrition content.
- But within the machine learning community,

it has been a fairly ad-hoc process…



Documentation for datasets

- **Datasheets for datasets** (Gebru et al., 2018) is an influential paper that provides community norms around documentation.
- **Data statements** (Bender & Friedman, 2018) is related framework that is more tailored to language datasets.
- The emphasis is on **transparency**.
- Two purposes:
  1. Dataset creators: reflect on decisions, potential harms (e.g., social biases) when creating the dataset.
  2. Dataset consumers: know when the dataset can and can't be used.

Documentation for datasets

A sample of the questions from each category are provided below:

- **Motivation**
  - For what purpose was the dataset created?
- **Composition**
  - What do the instances that comprise the dataset represent (e.g., documents, photos, people, countries)?
  - Is any information missing from individual instances?
- **Collection process**
  - How was the data associated with each instance acquired?
  - Who was involved in the data collection process?

Documentation for datasets

- **Preprocessing/cleaning/labeling**
  - Was any preprocessing/cleaning/labeling of the data done?
  - Is the software that was used to preprocess/clean/label the data available?
- **Uses**
  - Has the dataset been used for any tasks already?
  - Are there tasks for which the dataset should not be used?
- **Distribution**
  - How will the dataset will be distributed?
- **Maintenance**
  - Who will be supporting/hosting/maintaining the dataset?
  - Will the dataset be updated (e.g., to correct labeling errors, add new instances, delete instances)?

As an example, let's look at the datasheet for The Pile.

Summary

- Rematerialization and parallelism are critical to scale up model and data size
  - Data parallelism and ZeRO optimizer
  - Model parallelism
    - Tensor parallelism and pipeline parallelism
- Training 100B-scale models in action
  - Architecture design, training corpus, stability all make sense
- Data behind large language models
  - Common Crawl, WebText, etc.
  - Documentation for datasets
-

Training Very Large Language Models

Jie Tang, KEG, Tsinghua University

http://keg.cs.tsinghua.edu.cn/jietang

https://github.com/THUDM