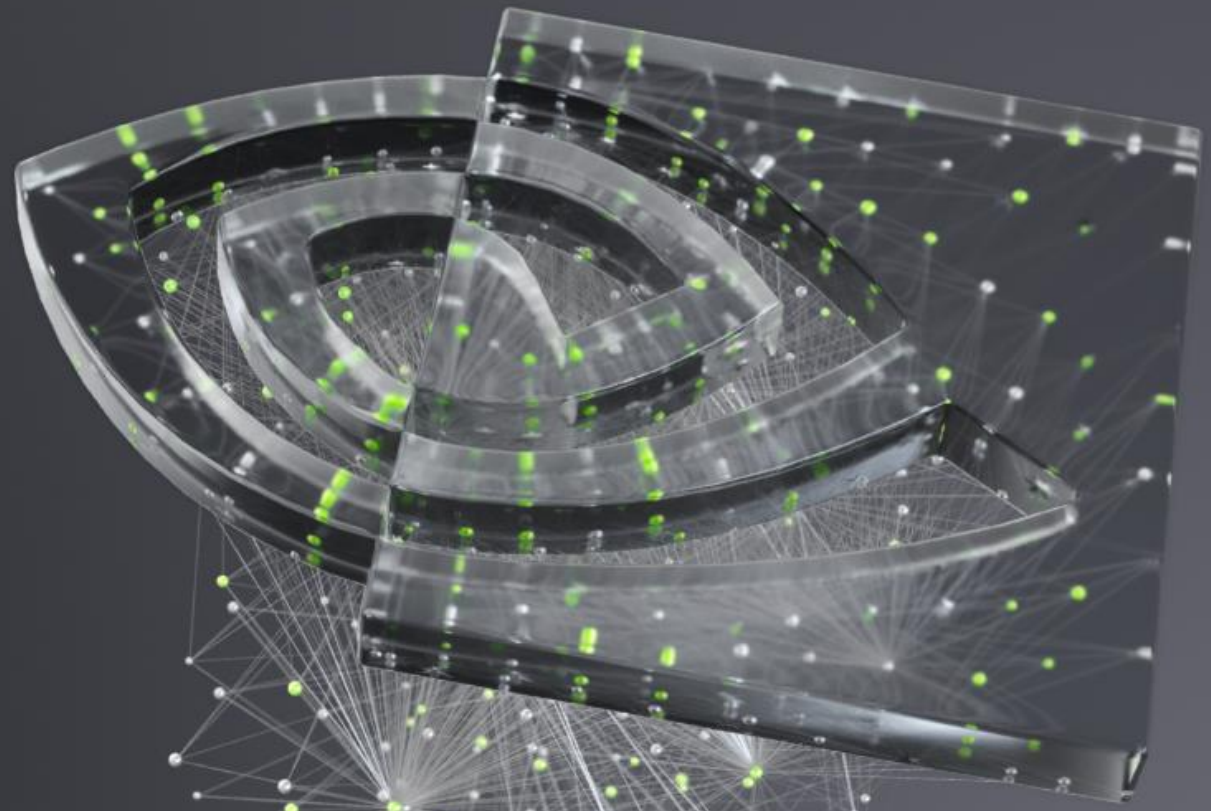


S51196 FASTERTRANSFORMER V5.4 NEW FEATURES AND BEYOND

BoYang Hsueh, 2023/03/24

AGENDA

What is FasterTransformer

FP8 on Bert and GPT

Development of ViT/SWIN

Optimization in FasterTransformer

FasterTransformer + Triton

WHAT IS FASTERTRANSFORMER

WHAT IS FASTERTRANSFORMER

What we provide

- FT provides highly optimized transformer layer for inference
- Based on C++, CUDA, cuBLAS and cuBLASlt
- Support FP32, FP16, BF16, INT8 (limited models) and FP8 (limited models)
- Support C++, TensorFlow, PyTorch and Triton backend
- Support MGMN serving (limited models)
- Open source (some kernels are binary files):
  - https://github.com/NVIDIA/FasterTransformer
  - https://github.com/triton-inference-server/fastertransformer_backend

WHAT IS FASTERTRANSFORMER

v5.1 updates

- Optimize the GPT workflow in specific cases
  - Padding removal for context phase (like what we do in BERT)
  - Shared context optimization
  - Interactive generation
  - Prompt/P tuning for nemo GPT
  - Streaming decoding for GPT
- Support new models
  - OPT/UL2/GPT-NEOX/mT5
- Support bfloat16 on most models
- Support MGMN BERT

WHAT IS FASTERTRANSFORMER

v5.2 updates

- New features on GPT
  - SmoothQuant
  - Extend int8 weight only quantization for more shapes
  - Support flash attention for long sequence on multi-head attention
- Prompt/P tuning for nemo T5
- Support new models
  - IA3, BLOOM, T5-MoE, SwinV2, BART, mBART

WHAT IS FASTERTRANSFORMER

v5.3 updates

- Support GPT-MoE
- Support FP8 for Bert and GPT (Experimental, need CUDA 11.8 and Hopper GPU)
- Support DeBERTa

WHAT IS FASTERTRANSFORMER

V5.4 updates

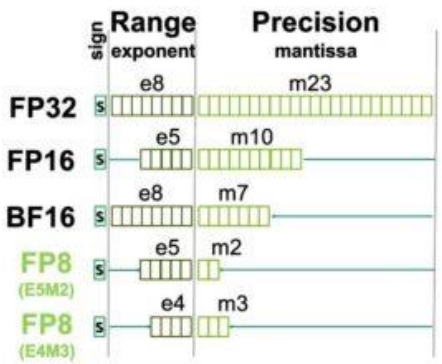
- Officially support FP8 on Bert and GPT.
- Support SmoothQuant on GPT

FP8 ON BERT AND GPT

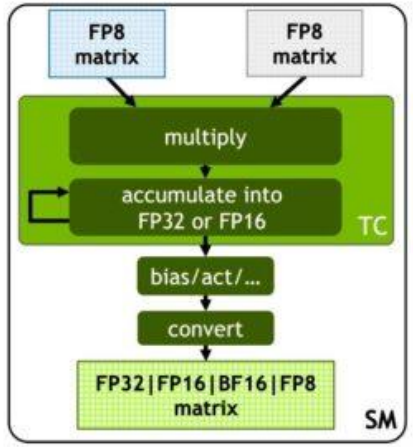
FP8 ON BERT AND GPT

What is FP8?

- FP8 has two formats: e4m3 and e5m2
- In inference, we only consider e4m3

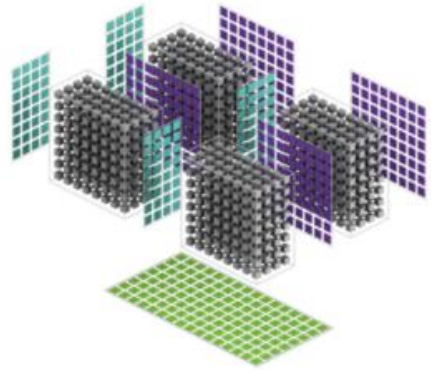


Allocate 1 bit to either range or precision

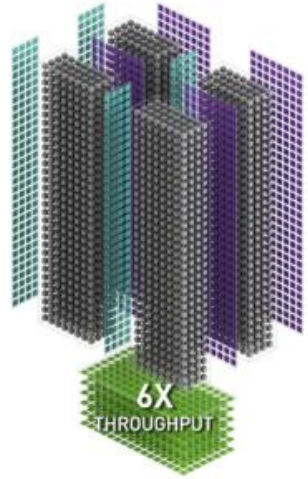


Support for multiple accumulator and output types

A100 FP16

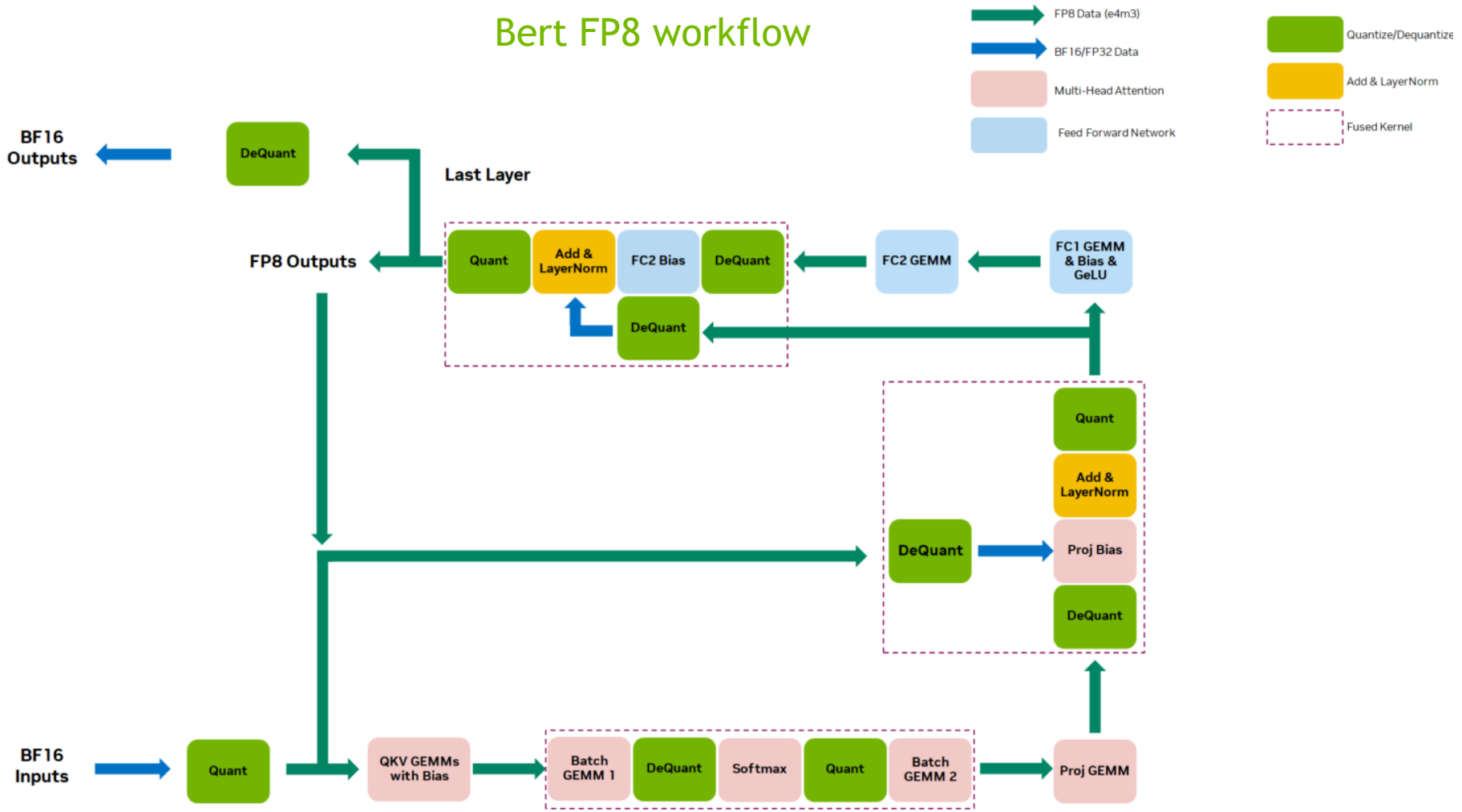


H100 FP8



FP8 ON BERT AND GPT

Bert FP8 workflow



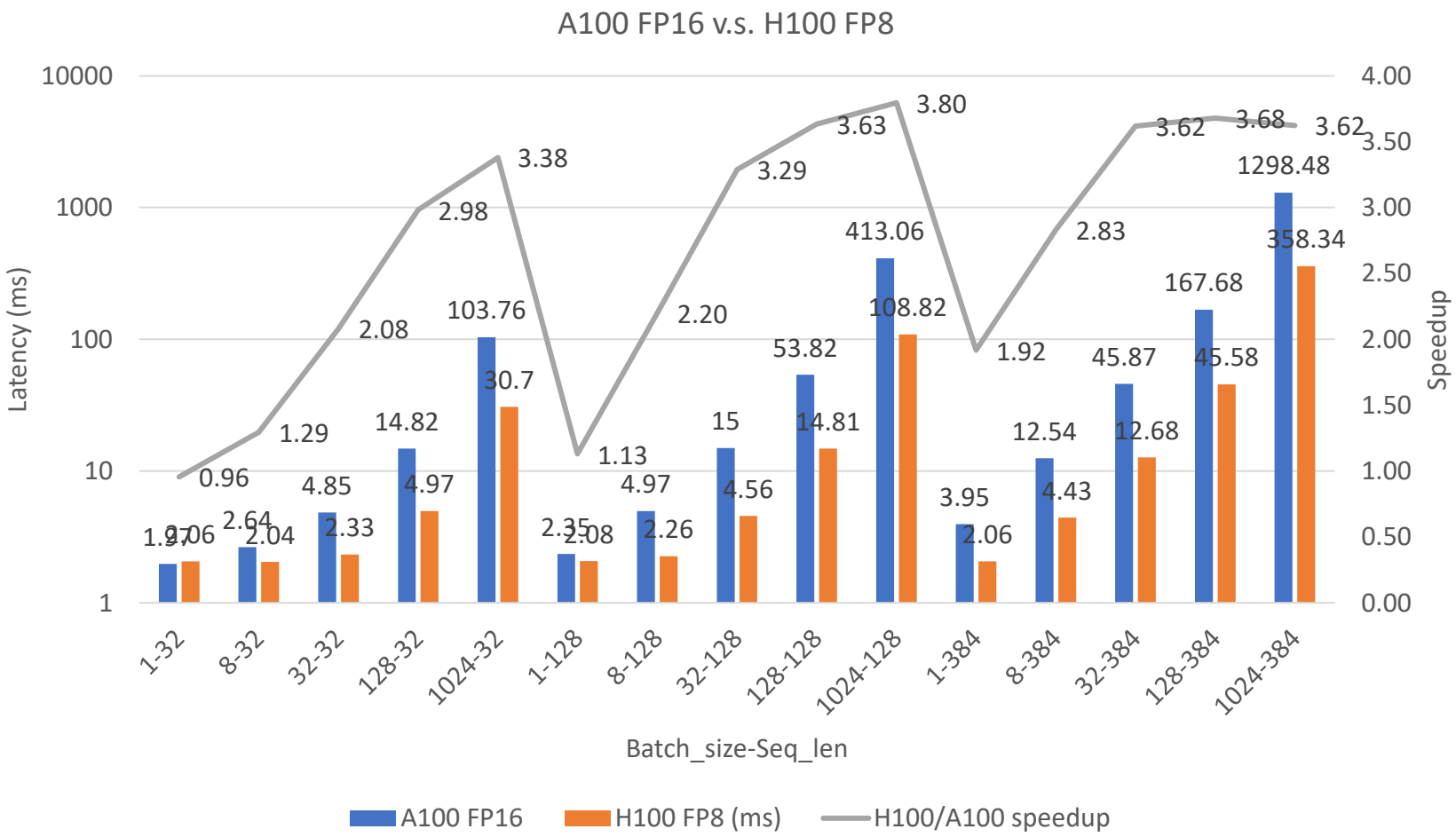
FP8 ON BERT AND GPT

BERT FP8 benchmark

- Environment
  - Nvcr.io/nvidia/pytorch:22.10-py3
  - A100: NVIDIA A100-SXM4-80GB, 400W, driver: 515.65.01, run with FP16
  - H100: H100 80GB HBM3, 700W, driver 525.85.12, run with FP8
  - Model: Bert-Large

FP8 ON BERT AND GPT

BERT FP8 benchmark



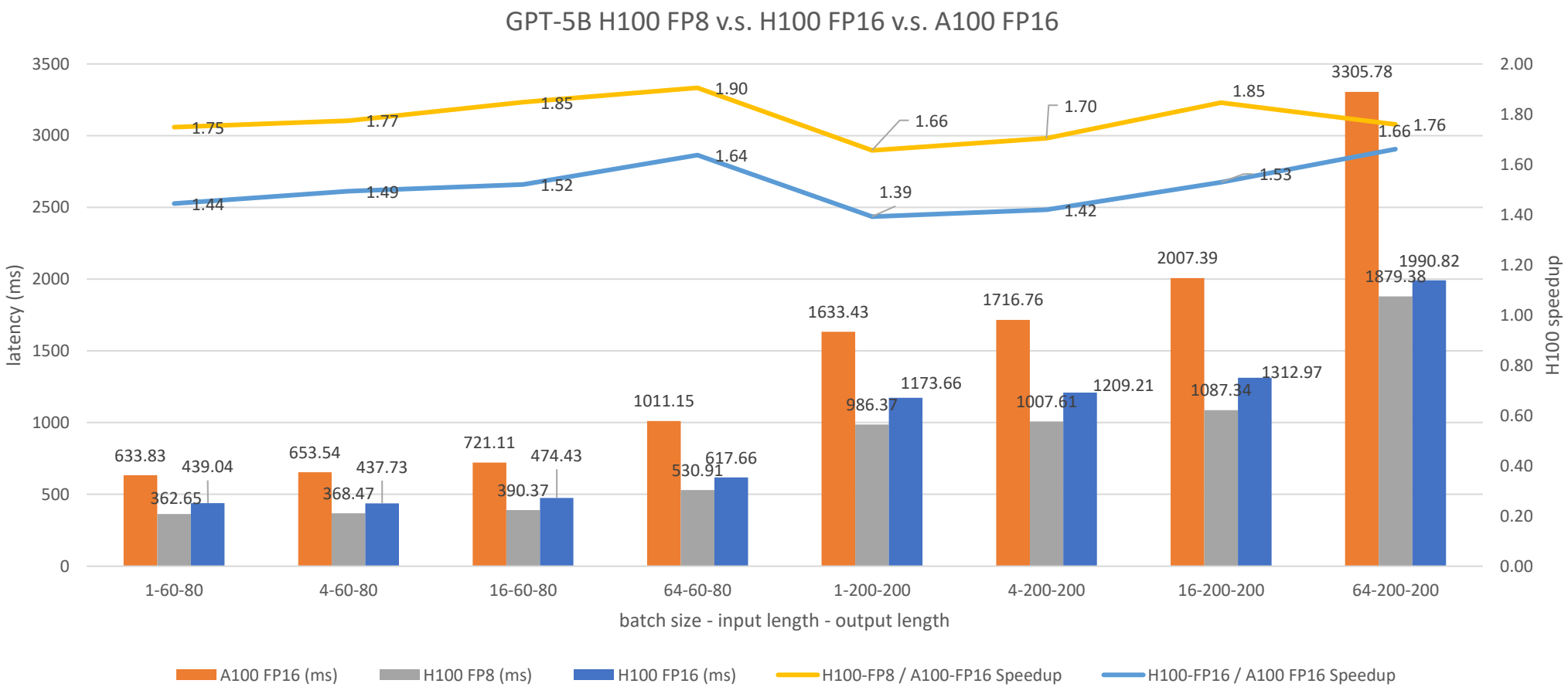
FP8 ON BERT AND GPT

Single GPU GPT FP8 benchmark

- Environment
  - Nvcr.io/nvidia/pytorch:22.10-py3
  - A100: NVIDIA A100-SXM4-80GB, 400W, driver: 515.65.01, run with FP16
  - H100: H100 80GB HBM3, 700W, driver 525.85.12, run with FP8
  - Model: GPT-5B

FP8 ON BERT AND GPT

Single GPU GPT FP8 benchmark



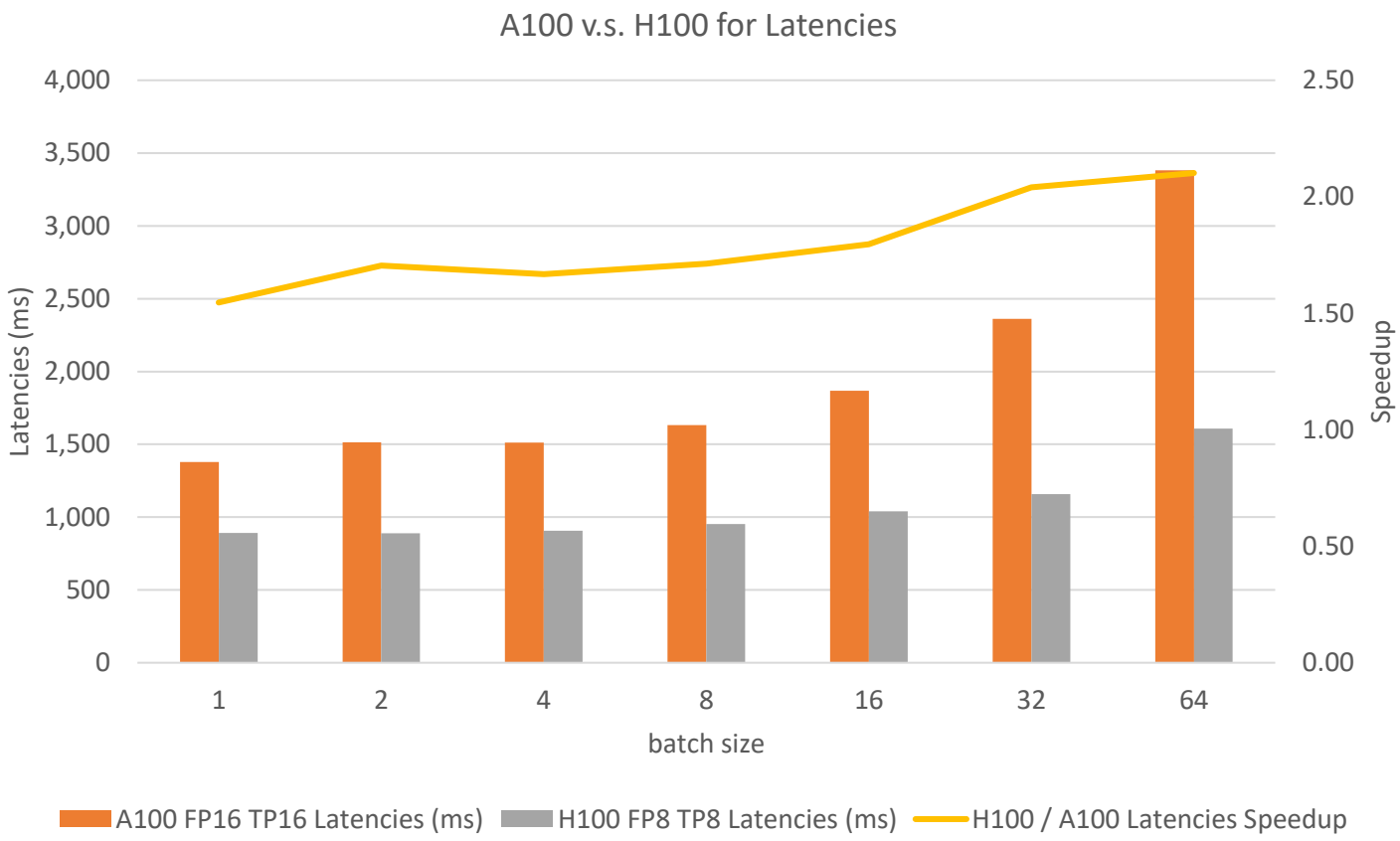
FP8 ON BERT AND GPT

Multi-GPU GPT FP8 benchmark Environment setting

- Nvcr.io/nvidia/pytorch:22.10-py3
- A100
  - Intra node: 8xA100-80GBs (with mclk 1593MHz, pclk 1410MHz) with AMD EPYC 7742 64-Core Processor, linked by NVSwitch
  - Inter node: Linked by Infiniband, 8x200Gb/s NICs
  - Inference data type: FP16
- H100
  - Intra node: 8xH100-80GBs (with mclk 2619MHz, pclk 1980MHz) with Intel(R) Xeon(R) Platinum 8480C, linked by NVSwitch
  - Inference data type: FP8
- Model: Megatron 530B

FP8 ON BERT AND GPT

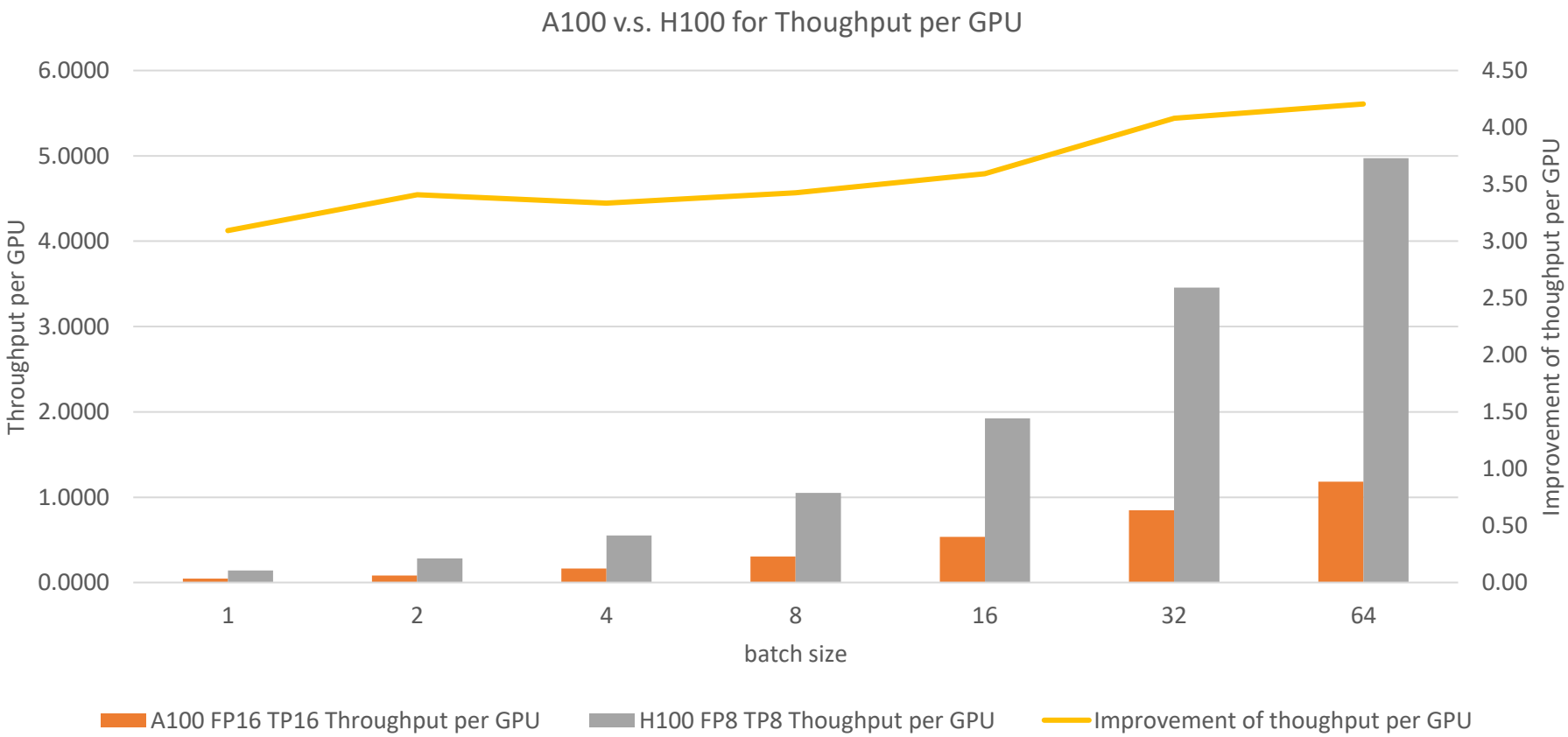
Multi-GPU GPT FP8 benchmark



Input length 60, output length 20

FP8 ON BERT AND GPT

Multi-GPU GPT FP8 benchmark

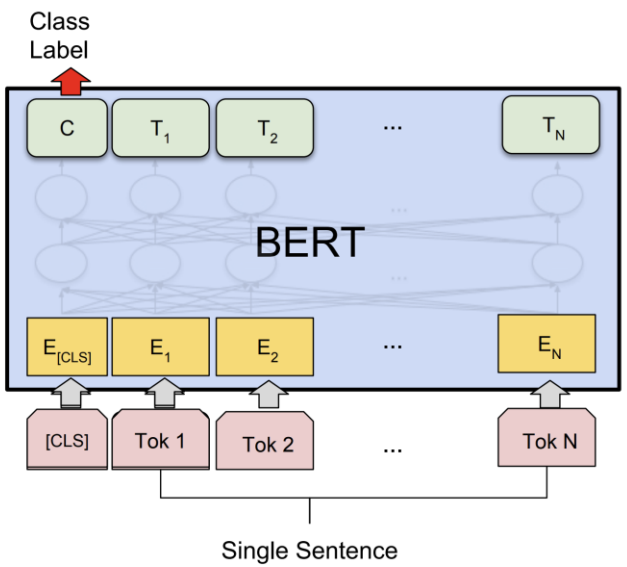


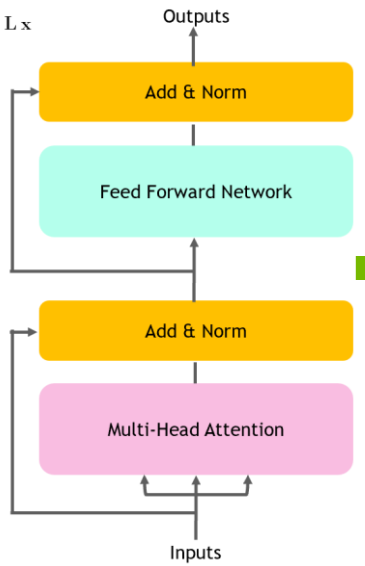
Input length 60, output length 20

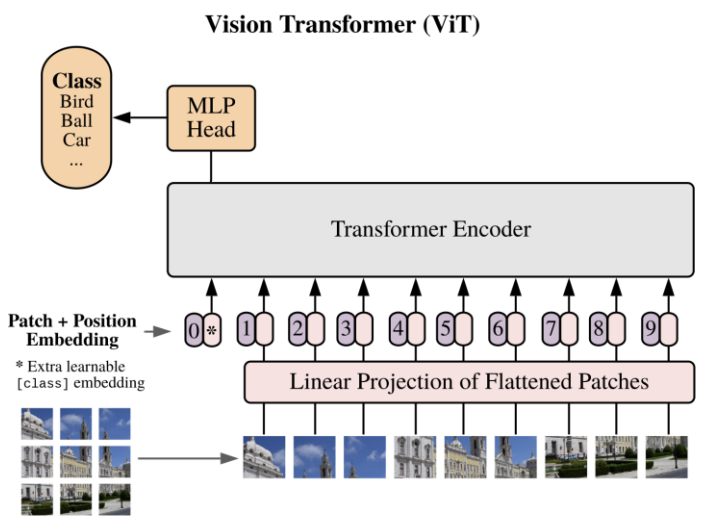
DEVELOPMENT OF VIT/SWIN

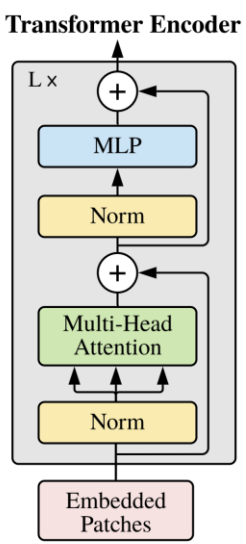
DEVELOPMENT OF VIT/SWIN

From Bert(NLP)To ViT (VISION)





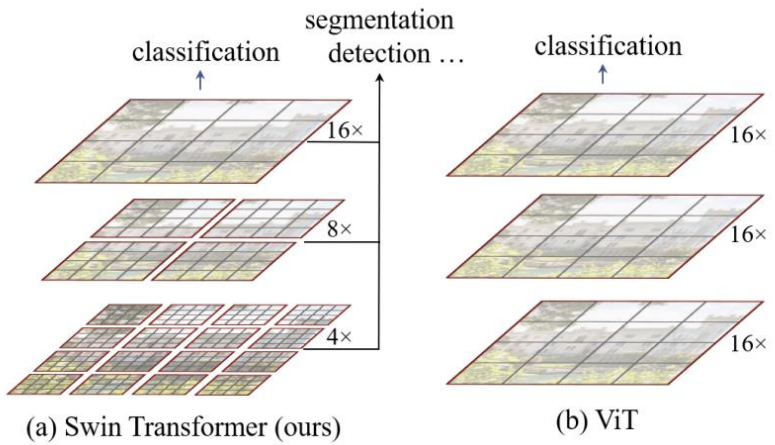




- ***BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding***
- **An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale**

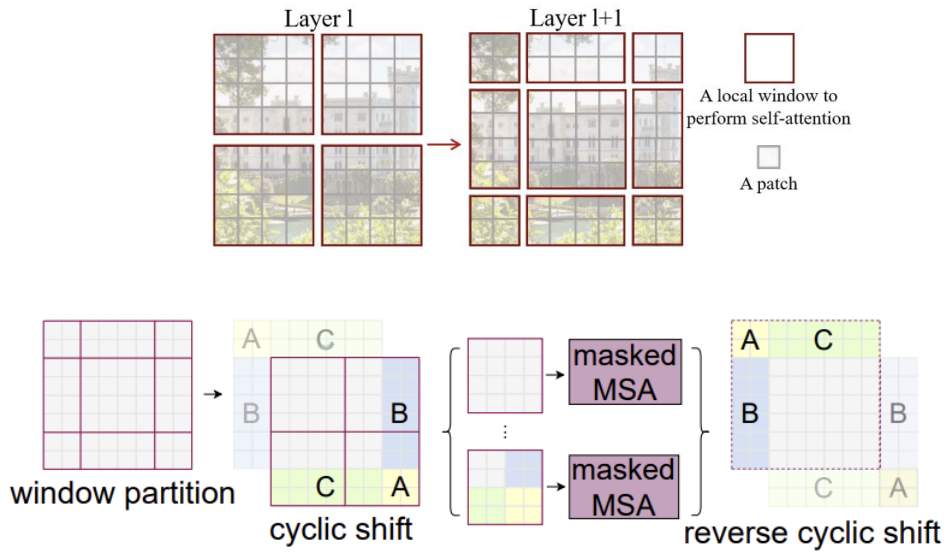
DEVELOPMENT OF VIT/SWIN

From ViT To Swin



linear computation complexity

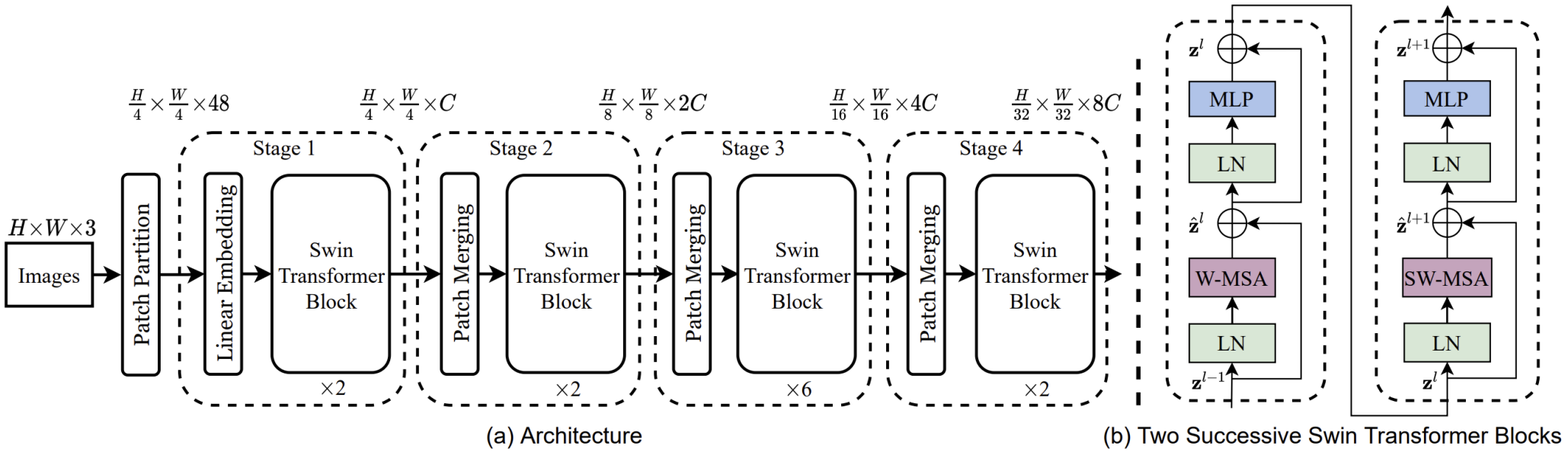
quadratic computation complexity



Shifted Windows Self-attention with relative position bias

- [Swin Transformer: Hierarchical Vision Transformer using Shifted Windows]

DEVELOPMENT OF VIT/SWIN



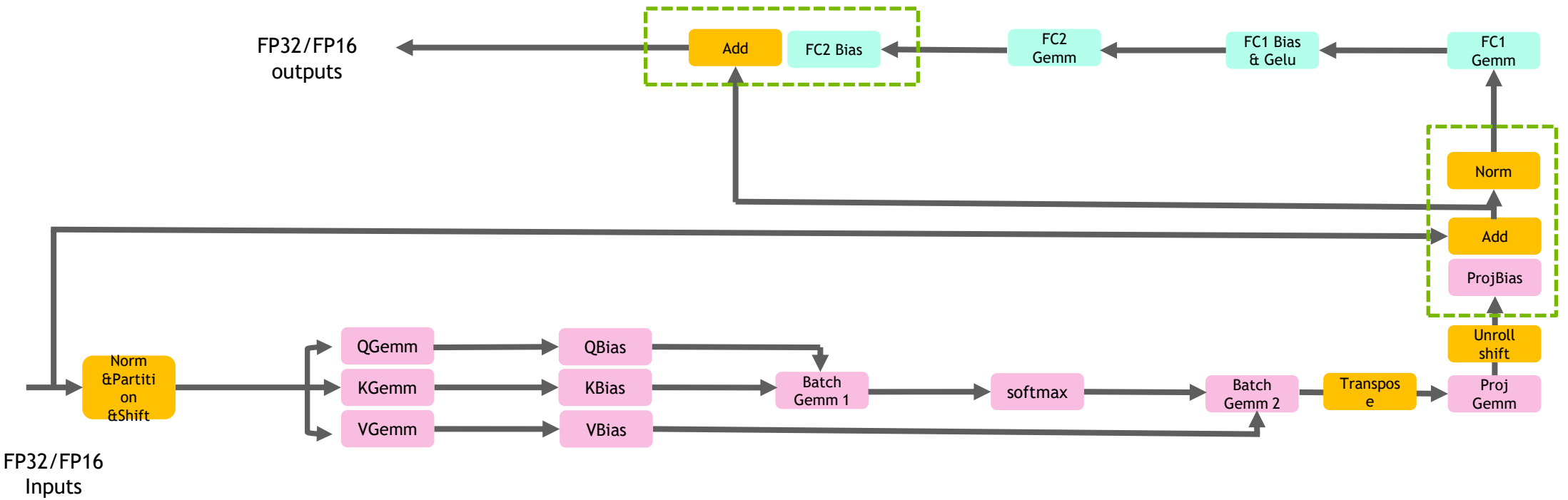
(a) Architecture (b) Two Successive Swin Transformer Blocks

Figure 3. (a) The architecture of a Swin Transformer (Swin-T); (b) two successive Swin Transformer Blocks (notation presented with Eq. (3)). W-MSA and SW-MSA are multi-head self attention modules with regular and shifted windowing configurations, respectively.

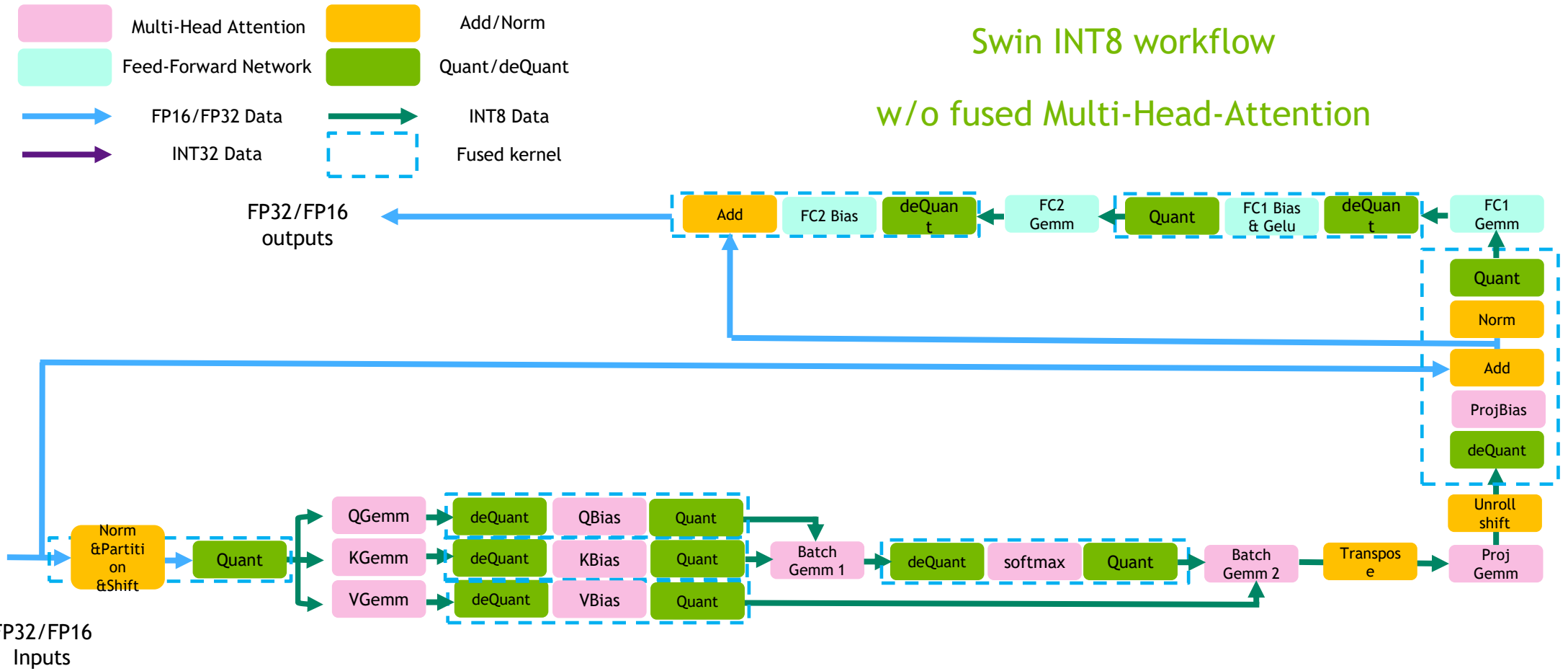
- Swin Transformer: Hierarchical Vision Transformer using Shifted Windows

DEVELOPMENT OF VIT/SWIN

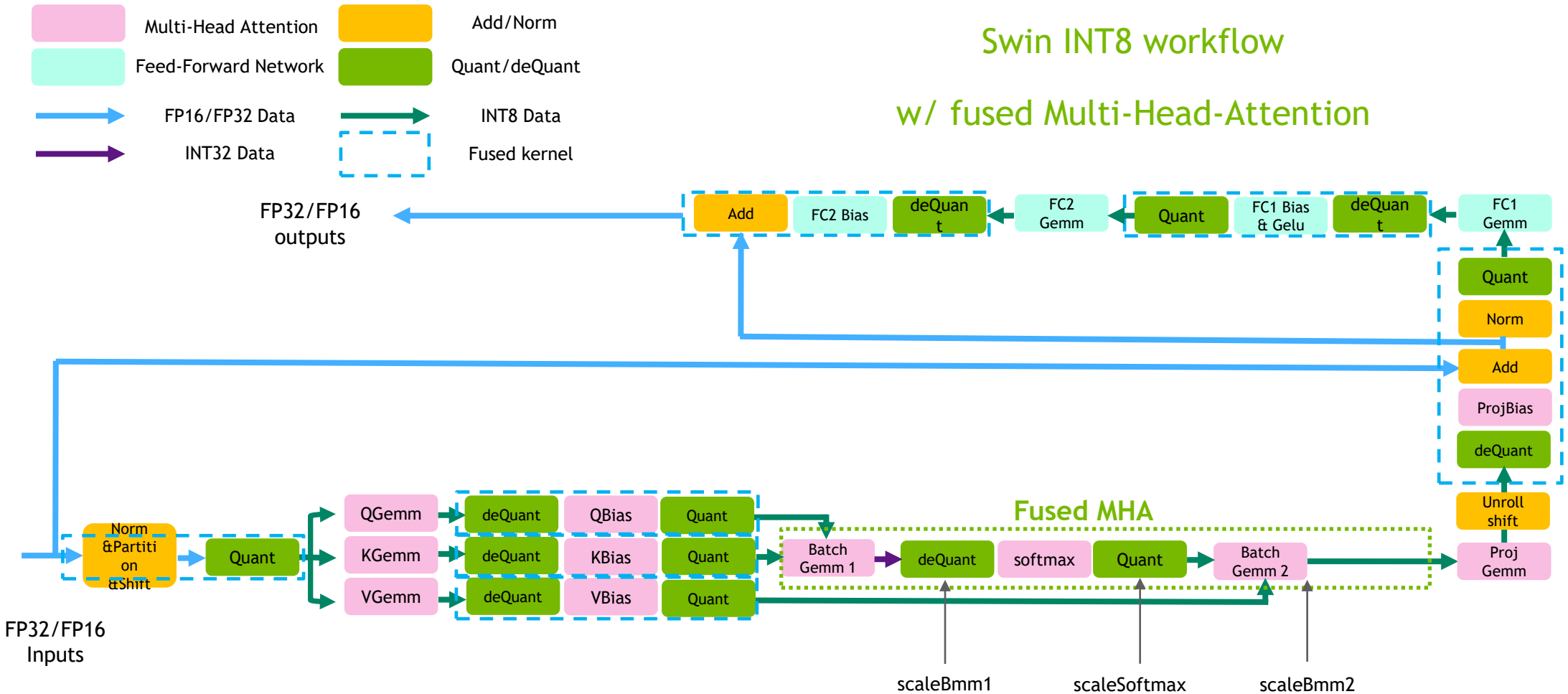
Swin FP16/FP32 workflow



INT8 QUANTIZATION OF SWIN LAYER



FUSED MHA FOR SWIN



IMPROVEMENT OF SWIN-LARGE PTQ ACCURACY

Accuracy on ImageNet

Window /Image size	Model name	FP16 Accuracy	Accuracy after PTQ(All GEMMs with INT8 O)	Accuracy after PTQ (FC2 & PatchMerge with INT32 O)
7/224	Swin-LARGE	86.25%	83.53%(-2.72%)	85.93%(-0.32%)
12/384	Swin-LARGE	87.25%	83.10%(-4.15%)	86.92%(-0.33%)
			INT8_MODE=1	INT8_MODE=2

How to improve the Swin-LARGE PTQ accuracy? (QAT of Swin-LARGE is very time-consuming)

Solution: Disable some quantization node.
Reason:
When 'k' gets large in GEMM(m*n*k), the output may have a very large scale, thus quantizing it may lead to significant accuracy loss.

Which GEMM has the largest 'k'?
For a swin transformer block with dim=96:

- QKV: k = 96
- Q*K: k = 32
- V*A: k = 32
- Proj: k = 96
- FC1: k = 96
- FC2: k = 96 * 4
- PatchMerge: k = 96 * 4

Relaxing FC2&PatchMerge to be INT32 output is enough for improving PTQ accuracy.

THE PERFORMANCE OF SWIN TRANSFORMER

Performance with SWIN-V1 model on one A10 GPU

Batch size	torch.jit.trace (ms)	FT cpp (ms)	FT cpp speedup	FT torch op (ms)	FT speedup
1	5.04	0.92	5.48	1.39	3.63
8	7.53	3.2	2.24	3.28	2.3
16	14.8	6.1	2.43	6.4	2.31
32	29.27	11.89	2.46	12.31	2.38

Image size: 224x224, window size 7x7, inference data type: FP16

THE PERFORMANCE OF SWIN TRANSFORMER

Performance with SWIN-V1 model on one A10 GPU

Batch_size	TINY with FP16 (ms)	TINY with INT8 (ms)	INT8 Speedup
1	0.91	1.01	0.91
8	3.2	2.15	1.49
16	6.1	3.96	1.54
32	11.89	7.4	1.61
64	23.2	15.28	1.52
Batch_size	**Large with FP16 (ms)**	**Large with INT8 (ms)**	**INT8 Speedup**
1	1.86	2.17	0.86
8	7.54	4.62	1.63
16	15.1	8.73	1.73
32	28.6	16.94	1.69
64	56.91	35.59	1.60

Image size: 224x224, window size 7x7, All ran by FT cpp

OPTIMIZATION IN FT

OPTIMIZATION IN FT

INT 8 weight only quantization

- Quantization is an optimization for both model size and inference speed
- There are two common ways to quantize the model
  - Post training quantization (PTQ): less cost, lower accuracy
  - Quantization aware training (QAT): higher cost, higher accuracy

	FP16	INT8
Weight size for 530B	1060 GBs	530 GBs
At least require # of node	2 nodes	1 node
peak flops	312 TFLOPs	624 TOPs

OPTIMIZATION IN FT

INT 8 weight only quantization

- For large language model, it is hard to use both PTQ (due to accuracy) and QAT (due to cost)
- We find a way to combine the accuracy of FP16 and weight size of INT8

	FP16	INT8
Weight size for 530B	1060 GBs	530 GBs
At least require # of node	2 nodes	1 node
peak flops	312 TFLOPs	624 TOPs

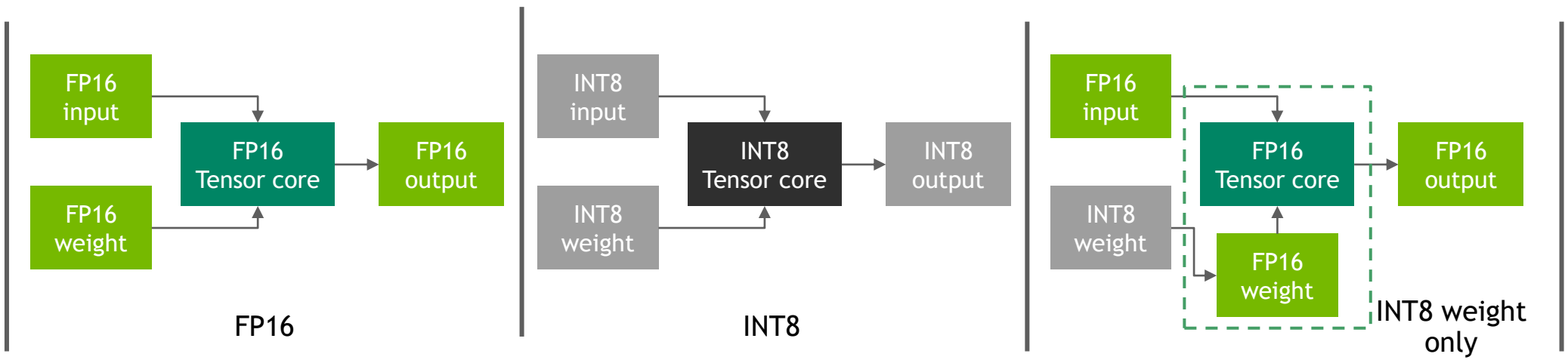
OPTIMIZATION IN FT

INT 8 weight only quantization

- For large language model, it is hard to use both PTQ (due to accuracy) and QAT (due to cost)
- We find a way to combine the accuracy of FP16 and weight size of INT8
- INT8 weight only save the weights by INT8, but activations are saved by FP16
- In GEMM, we load INT8 weight, cast to FP16, and use FP16 tensor core

	FP16	INT8	INT8 weight only
Weight size for 530B	1060 GBs	530 GBs	530 GBs
At least require # of node	2 nodes	1 node	1 node
peak flops	312 TFLOPs	624 TOPs	312 TFLOPs

OPTIMIZATION IN FT



	FP16	INT8	INT8 weight only
Weight size for 530B	1060 GBs	530 GBs	530 GBs
At least require # of node	2 nodes	1 node	1 node
peak flops	312 TFLOPs	624 TOPs	312 TFLOPs

OPTIMIZATION IN FT

INT 8 weight only quantization

Assume for GEMM $[M, K] \times [K, N] \rightarrow [M, N]$
Memory read/write in GEMM: $MK + KN + MN$
Compute in GEMM: $2MKN$

If M is small, like 1
Memory read/write in GEMM: $K + KN + N$
Compute in GEMM: $2KN$

- Pros of INT8 weight only
  - Small weight size (half of FP16)
  - Faster when batch size is small (bounded by reading weight when batch size is small)
  - Only need to quantize weight, simpler than PTQ and QAT
- Cons:
  - Little slower when batch size is large (need to cast INT8 to FP16 in GEMM)

	FP16	INT8	INT8 weight only
Weight size for 530B	1060 GBs	530 GBs	530 GBs
At least require # of node	2 nodes	1 node	1 node
peak flops	312 TFLOPs	624 TOPs	312 TFLOPs

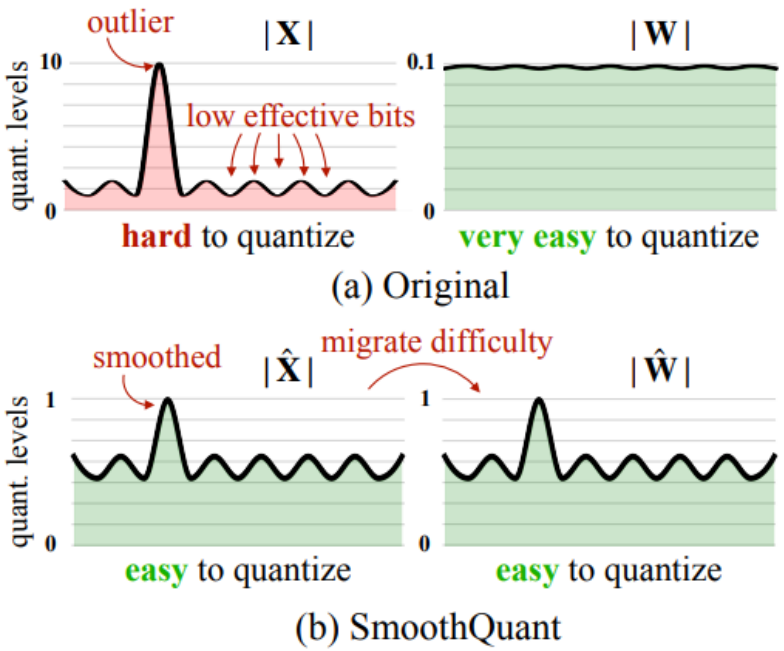
OPTIMIZATION IN FT

SmoothQuant

Table 1: SmoothQuant achieves high hardware efficiency while maintaining the accuracy of LLMs with 530 billion parameters in a training-free fashion.

	LLM (100B+) Accuracy	Hardware Efficiency
ZeroQuant	✗	✓
Outlier Suppression	✗	✓
`LLM.int8()`	✓	✗
SmoothQuant	✓	✓

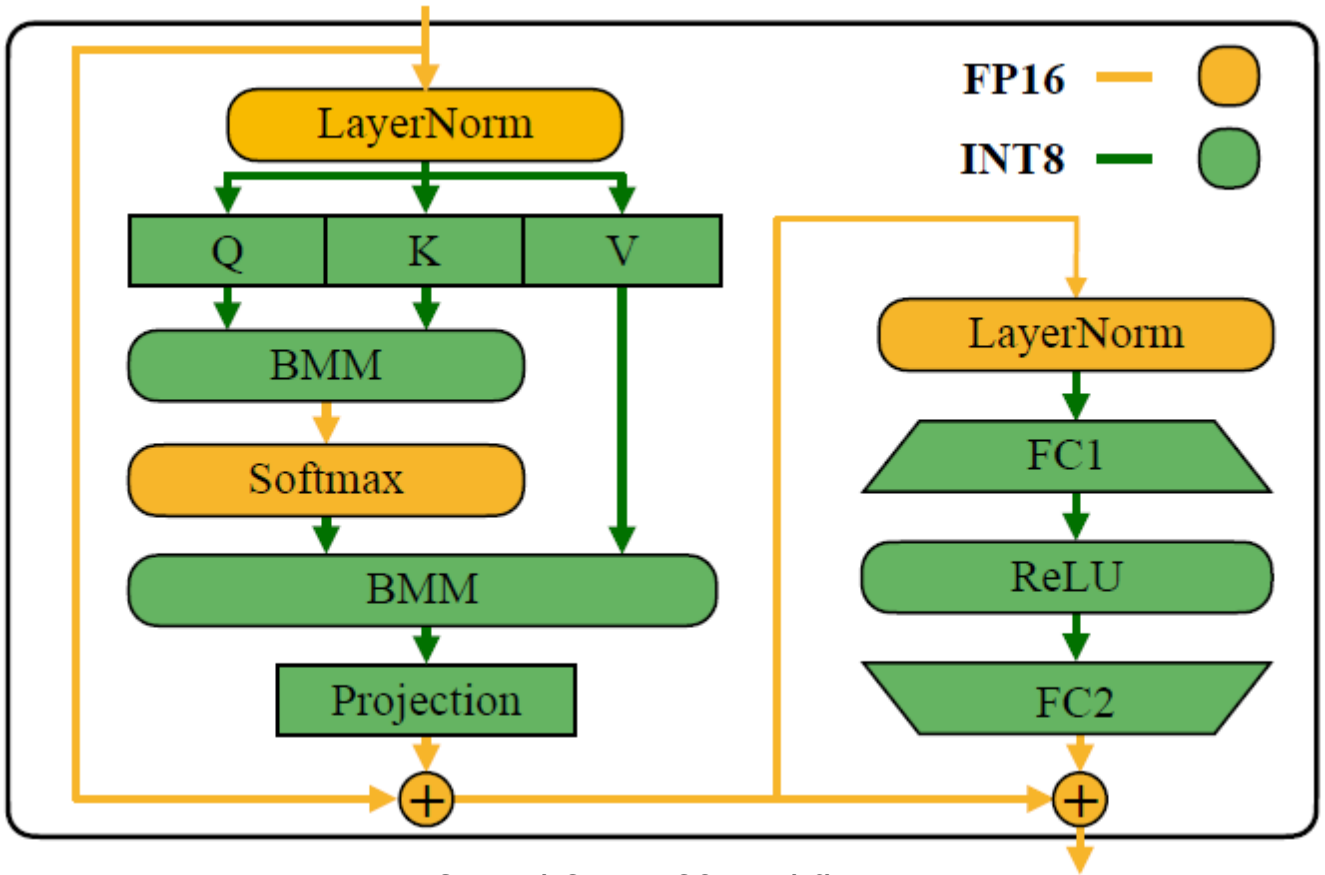
https://arxiv.org/pdf/2211.10438.pdf



https://arxiv.org/pdf/2211.10438.pdf

OPTIMIZATION IN FT

SmoothQuant



SmoothQuant O3 workflow

OPTIMIZATION IN FT

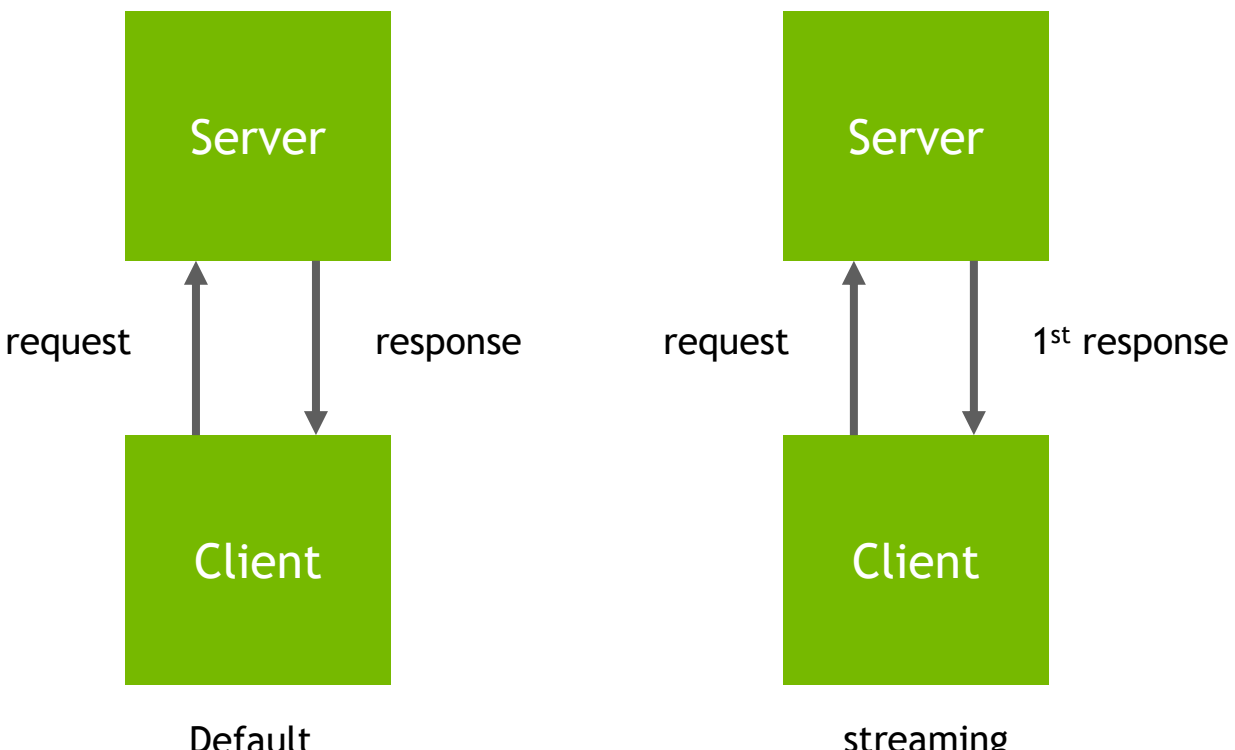
SmoothQuant

Input length	FP16-TP16 Latency (ms)	FP16-TP16 Memory (GBs per GPU)	INT8-TP8 Latency (ms)	INT8-TP8 Memory (GBs per GPU)
128	231.68	64.97	253.93	65.89
256	450.61	65.85	433.99	66.61
512	838.18	66.72	838.63	68.08
1024	1706.85	68.44	1688.69	71.20

Megatron 530B, Batch size 4, output length 1, linked by NVSwitch intra node, by Infiniband inter node.
Accuracy of FP16 on LAMBADA: 80.83
Accuracy of INT8 on LAMBADA: 80.56

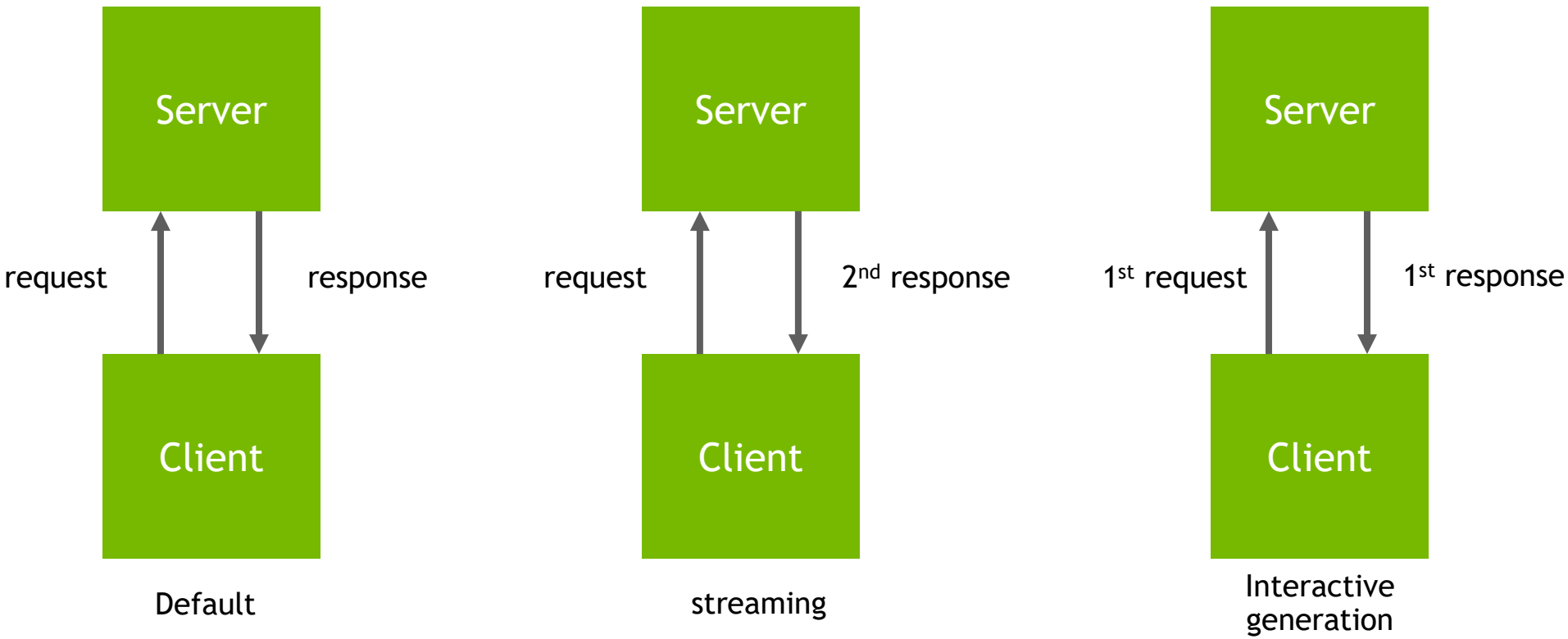
OPTIMIZATION IN FT

Triton generation feature



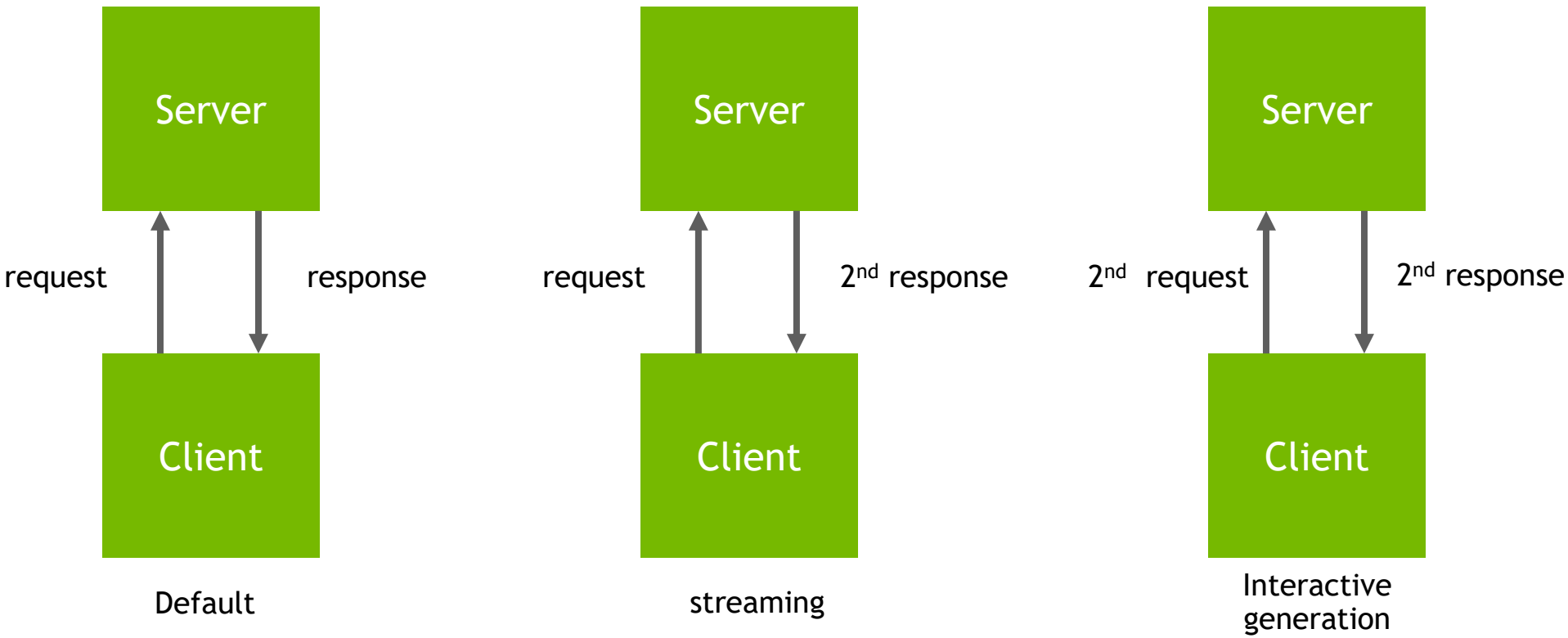
OPTIMIZATION IN FT

Streaming (Decoupled mode)



OPTIMIZATION IN FT

Interactive generation




FASTERTRANSFORMER + TRITON

DEVELOPERS CAN FOCUS ON MODELS AND APPLICATIONS

Triton Takes Care of Plumbing To Deploy Models for Inference

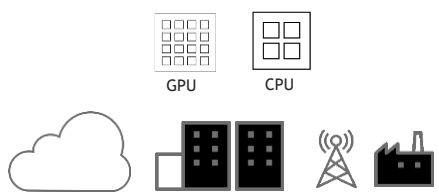
Standardized Inference Platform

TensorFlow, TensorRT, OpenVINO, PYTORCH, Custom Backends, ONNX RUNTIME, XGBoost, scikit-learn, LightGBM

Support for All Major Framework Backends for Flexibility & Consistency

Standard HTTP, gRPC, and C++ Communication

Inferencing on GPU and CPU

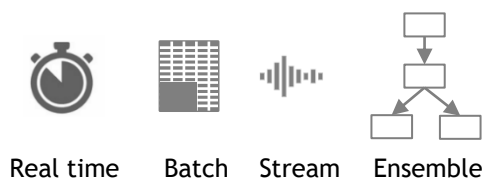


Inference Serving on GPU, CPU, and Mixed Workloads

Cloud | Data Center | Edge
Bare metal | Virtualization

Jetson | Windows | ARM

Different Types of Queries



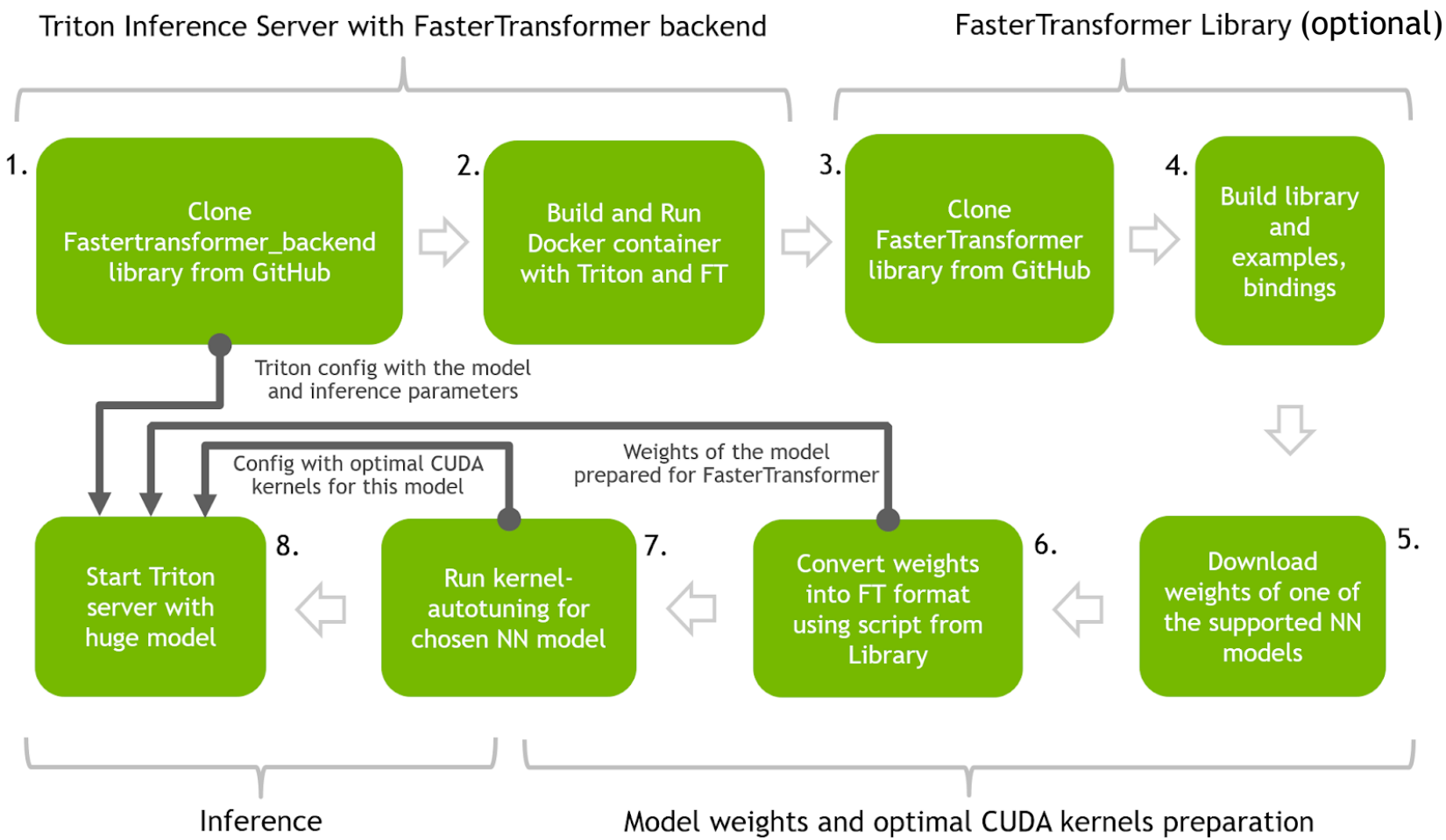
Support for Different Types of Inference Queries and Use Cases

Maximized Hardware Utilization

Concurrent Model Execution Increases Throughput & Utilization, lowering TCO

Dynamic Batching Maximizes Throughput Under Latency Constraint

FT + TRITON


Triton Inference Server with FasterTransformer backend
FasterTransformer Library (optional)
1.
Clone Fastertransformer_backend library from GitHub
2.
Build and Run Docker container with Triton and FT
3.
Clone FasterTransformer library from GitHub
4.
Build library and examples, bindings
Triton config with the model and inference parameters
Config with optimal CUDA kernels for this model
Weights of the model prepared for FasterTransformer
8.
Start Triton server with huge model
7.
Run kernel-autotuning for chosen NN model
6.
Convert weights into FT format using script from Library
5.
Download weights of one of the supported NN models
Inference
Model weights and optimal CUDA kernels preparation

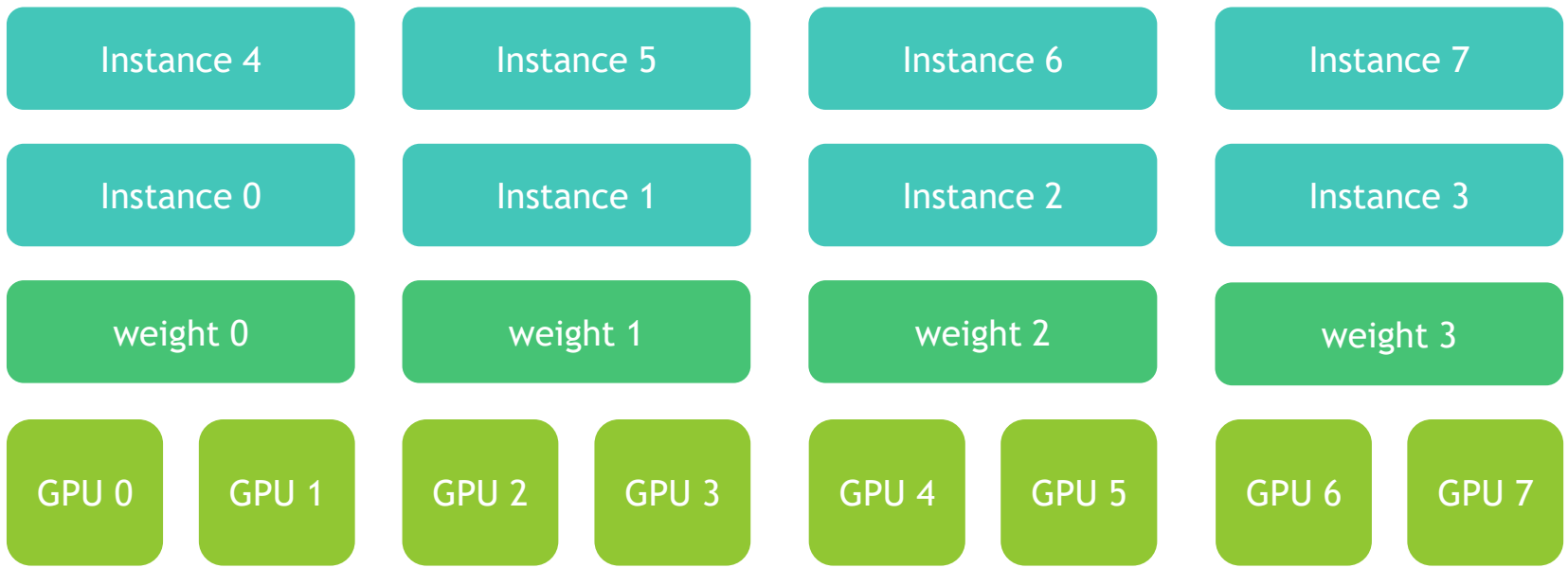
FT + TRITON

Configuration

- Inputs, output settings and runtime parameters (like TP size)
- instance_group:
  - **Must be KIND_CPU**: Control all GPUs by FT directly, necessary for MGMN serving.
  - assume 8 gpus, 8 model instances, tensor para size 2, then we will distribute model instances to [0, 1], [2, 3], [4, 5], [6, 7], [0, 1], [2, 3], [4, 5], [6, 7] GPUs;
  - Two instance instances on GPUs [0, 1] will share the same weights

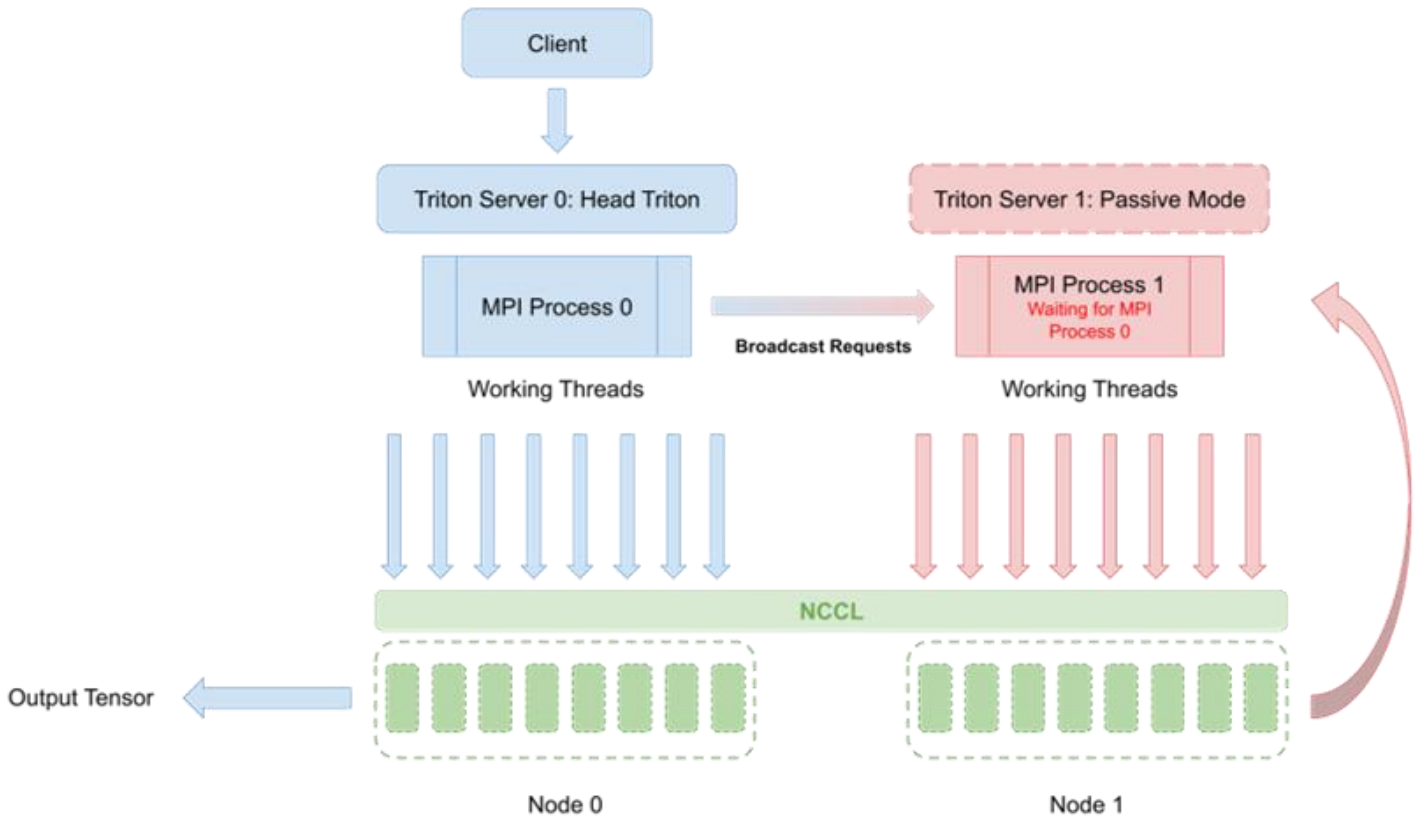
FT + TRITON

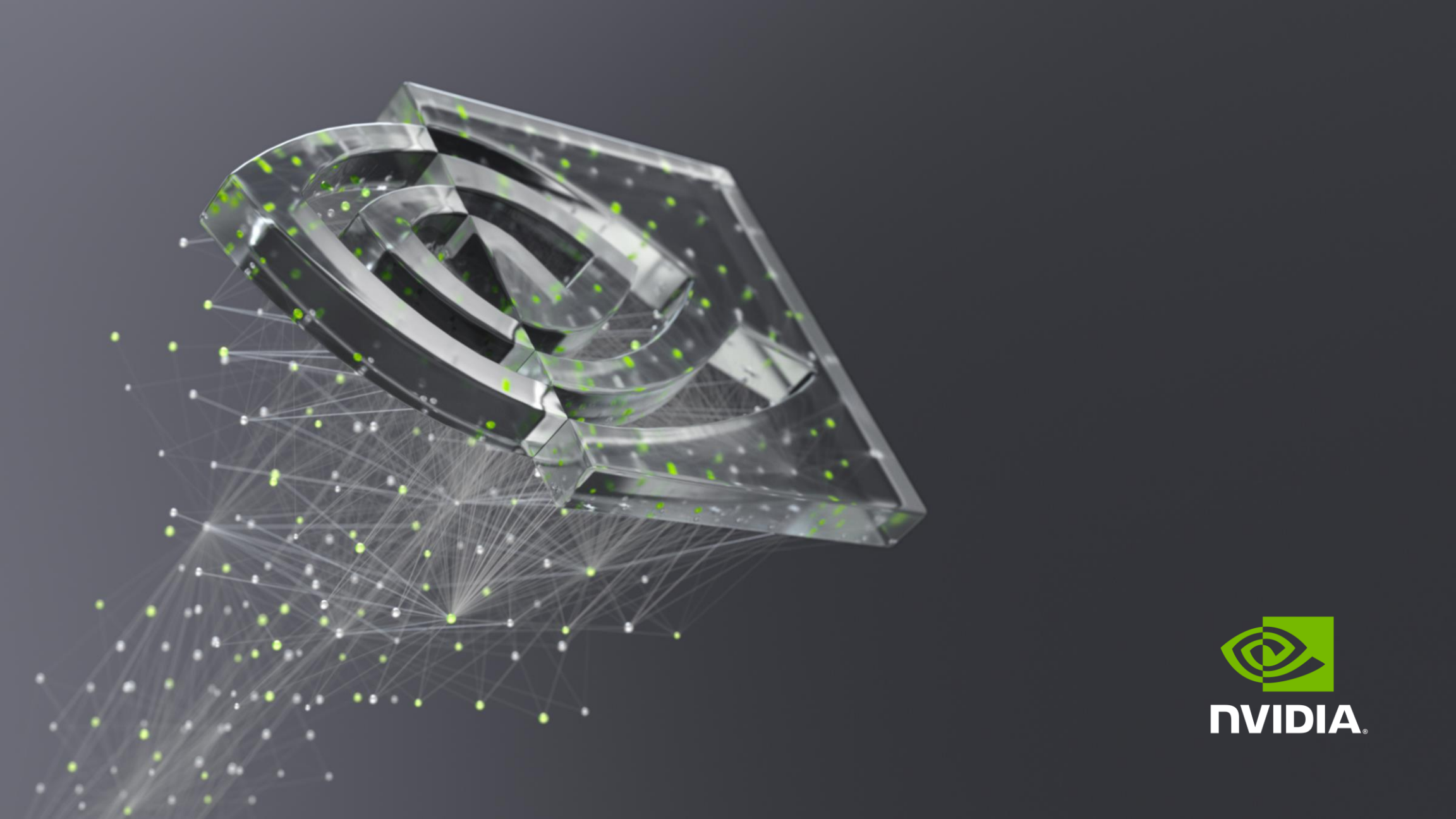
Instance group



FT + TRITON

Workflow on Triton




NVIDIA