玄界のGPUを最大限活用する方法

Solution Architecture and Engineering, NVIDIA | Oct 20th 2025

AGENDA

- Transformer Engine 概要
- NeMo Framework 概要
- その他のTips
-

Transformer Engine 概要

NVIDIA H100

Unprecedented Performance, Scalability, and Security for Every Data Center

Highest AI and HPC Performance

4PF FP8 (6X)| 2PF FP16 (3X)| 1PF TF32 (3X)| 60TF FP64 (3.4X)

3.35TB/s (1.5X), 80GB HBM3 memory*

Transformer Model Optimizations

6X faster on largest transformer models

Highest Utilization Efficiency and Security

7 Fully isolated & secured instances, guaranteed QoS

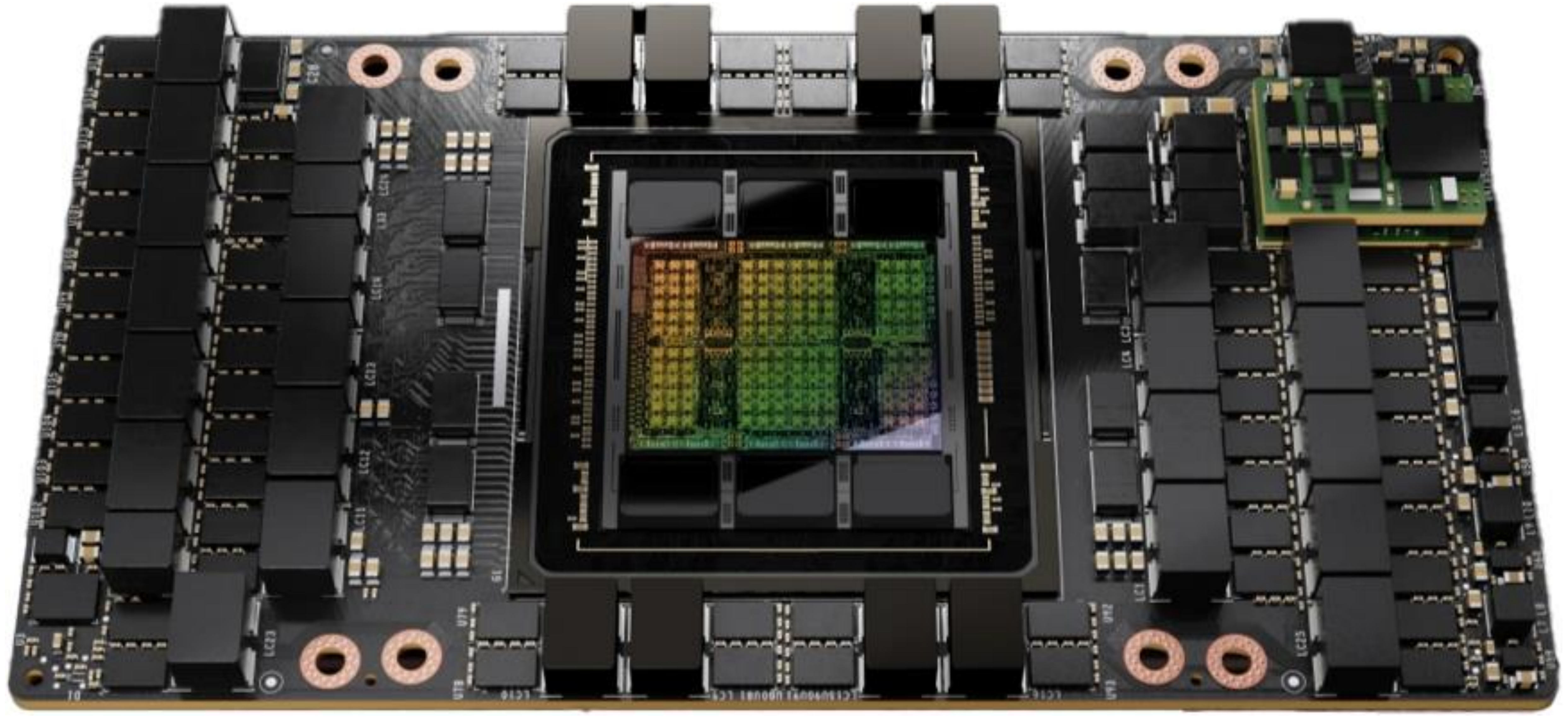
2nd Gen MIG | Confidential Computing

Fastest, Scalable Interconnect

900 GB/s GPU-2-GPU connectivity (1.5X)

128GB/s PCI Gen5

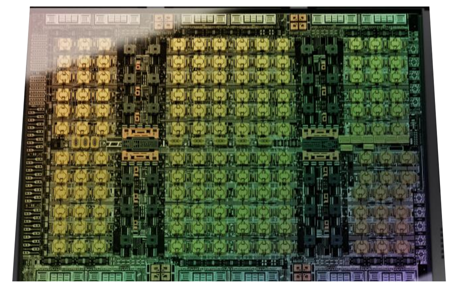
*NVIDIA H100 SXMは
94GB HBM2eモデルも存在します



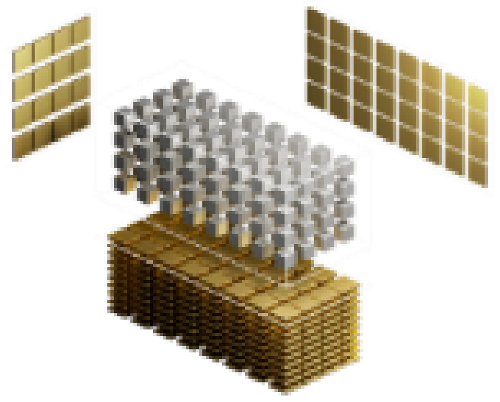
FP8, FP16, TF32 performance include sparsity. X-factor compared to A100

NVIDIA Hopper

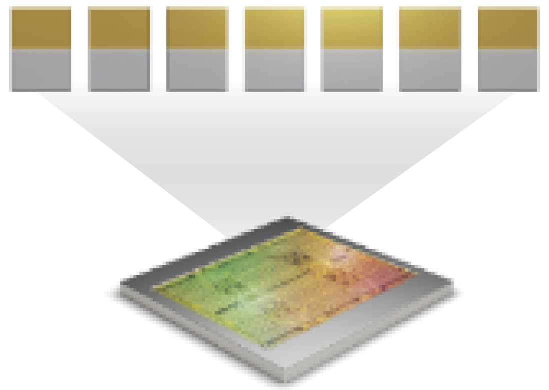
The Engine for the World's AI Infrastructure



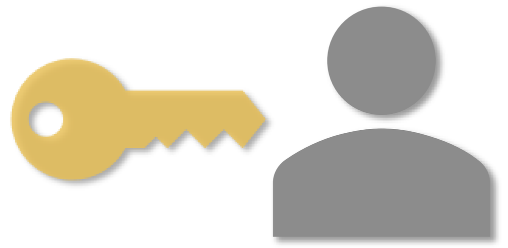
World's Most Advanced Chip



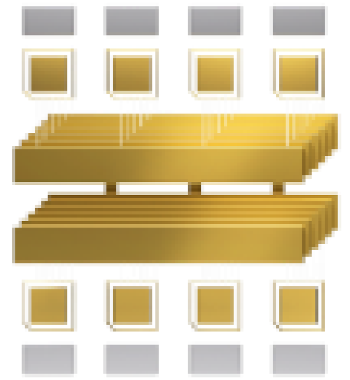
Transformer Engine



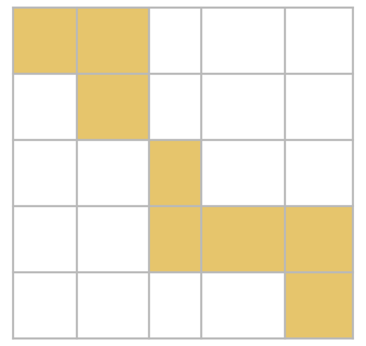
2^{nd} Gen MIG



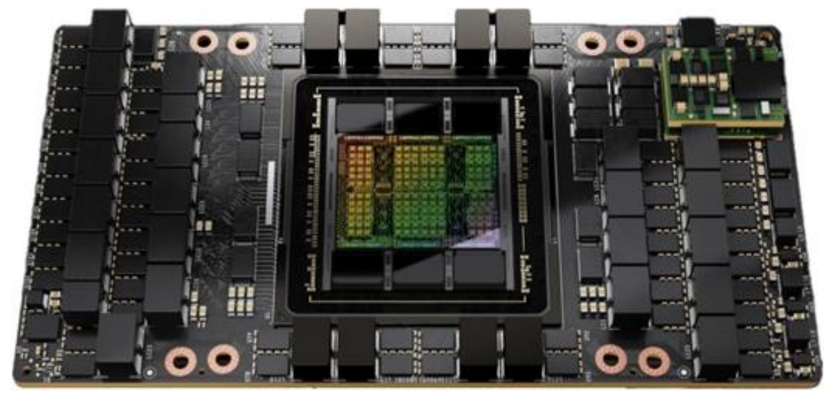
Confidential Computing



4^{th} Gen NVLink



DPX Instructions



H100 SXM

H100 NVL

NVIDIA AI Enterprise is a 5-Year Subscription

FP8

フォーマット

FP32 = 0.3952

sign | exponent | mantissa

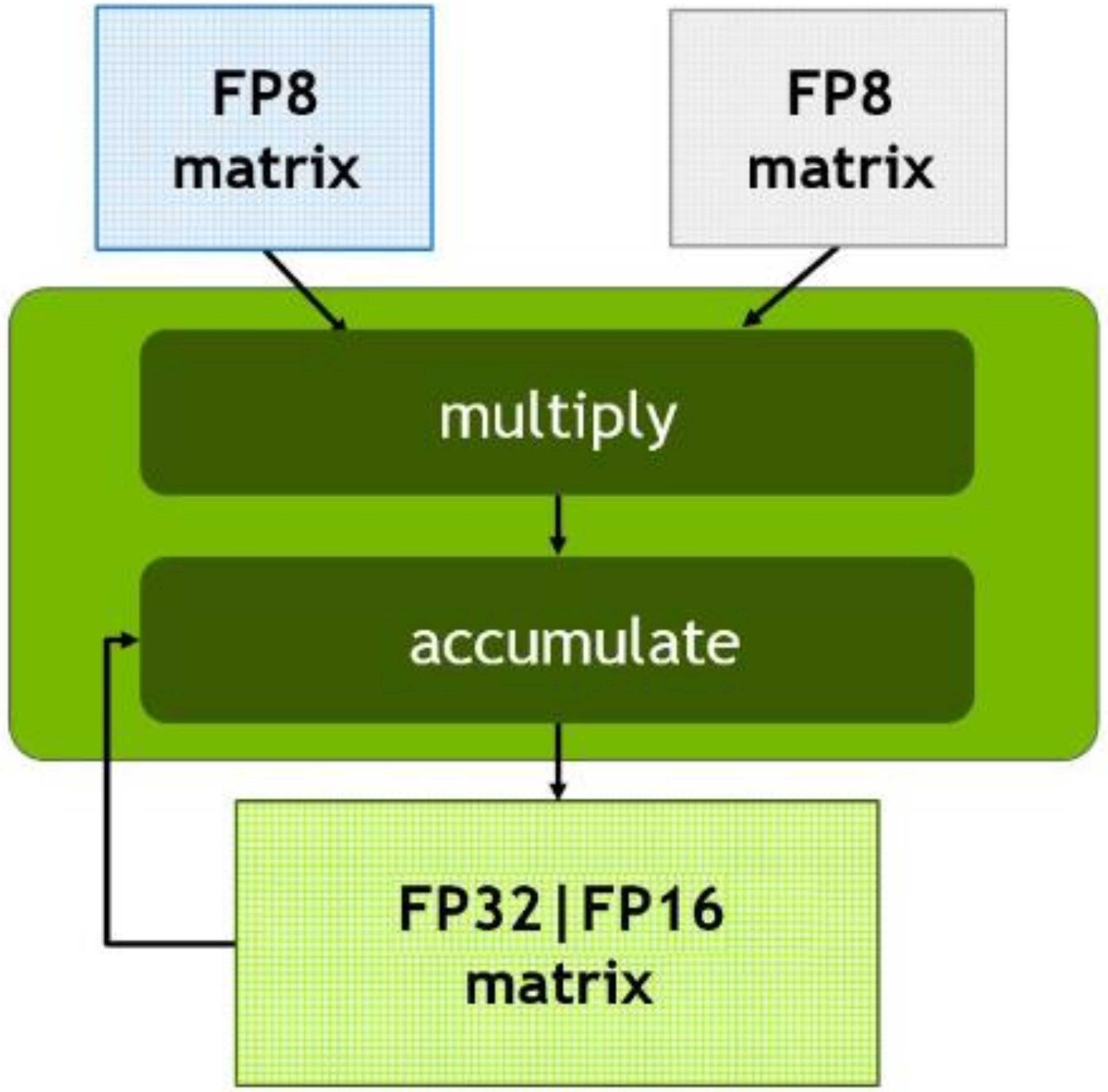
FP16	0	0	1	1	0	1	1	0	0	1	0	1	0	0	1	1	= 0.395264
BF16	0	0	1	1	1	1	1	0	1	1	0	0	1	0	1	0	= 0.394531
FP8 E4M3	0	0	1	0	1	1	0	1									= 0.40625
FP8 E5M2	0	0	1	1	0	1	1	0									= 0.375

ハードウェア命令

FP8 Tensor Core

- 8bit精度のGEMM
  - 16bitの2倍のFLOPS
    - 32x of FFMA (64x with 2:4 sparsity)
  - 4種のFP8フォーマットの組み合わせに対応
    - E4M3xE4M3, E5M2xE5M2, E4M3xE5M2, E5M2xE4M3
  - 出力はFP16 or FP32

- タイプ変換
  - 8-bit <-> 32-bit/16-bit
    - FP8 -> FP16
    - FP16/FP32 -> FP8



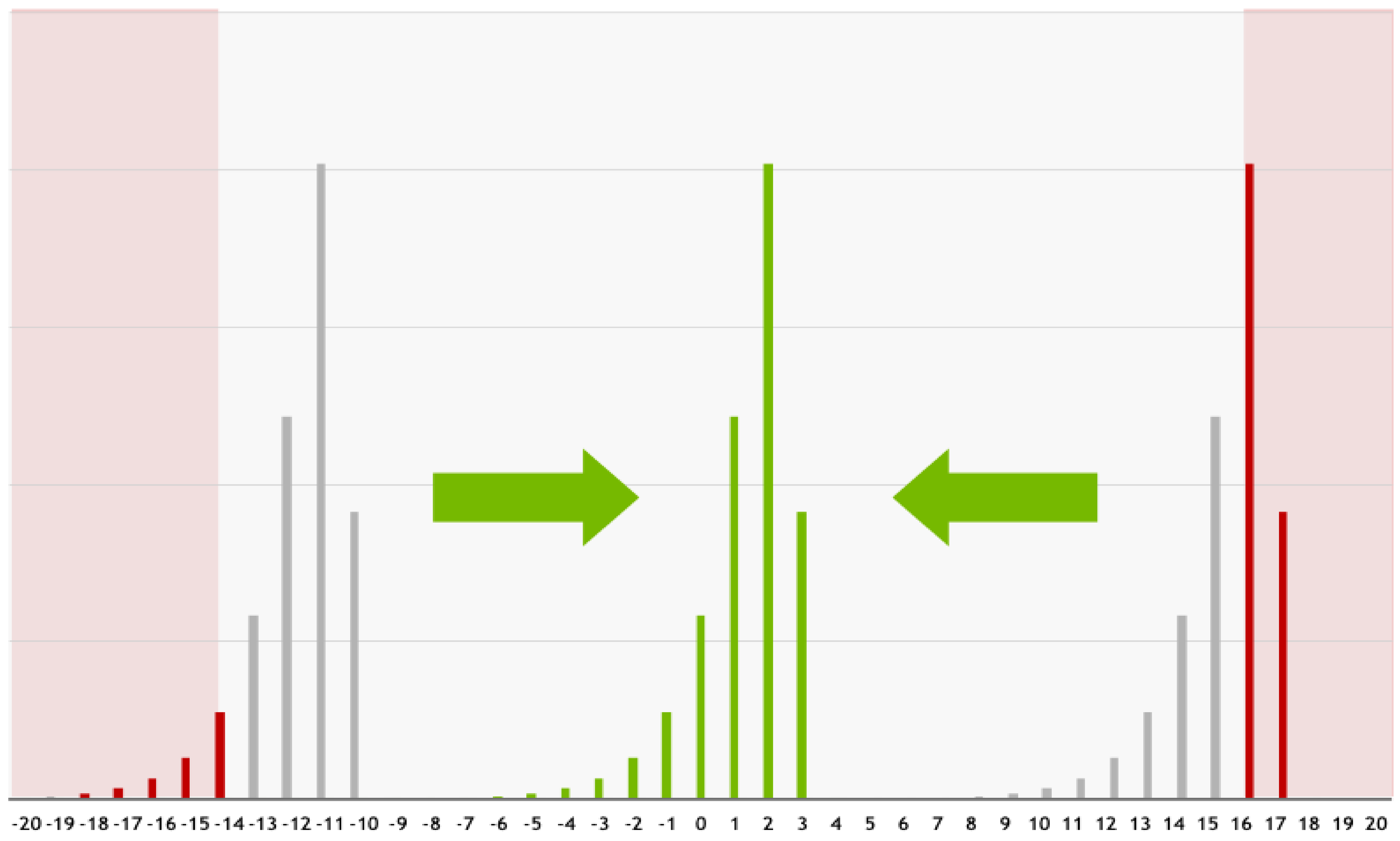
FP8の利点

- 計算の高速化
  - FP8 Tensor Coreは16bit Tensor Coreの2倍の演算速度
- メモリアクセスの高速化
  - 16bit -> 8bitによるメモリトラフィックの削減
- 推論環境へのデプロイが容易
  - FP8で学習済みのモデルは量子化による影響を受けにくい

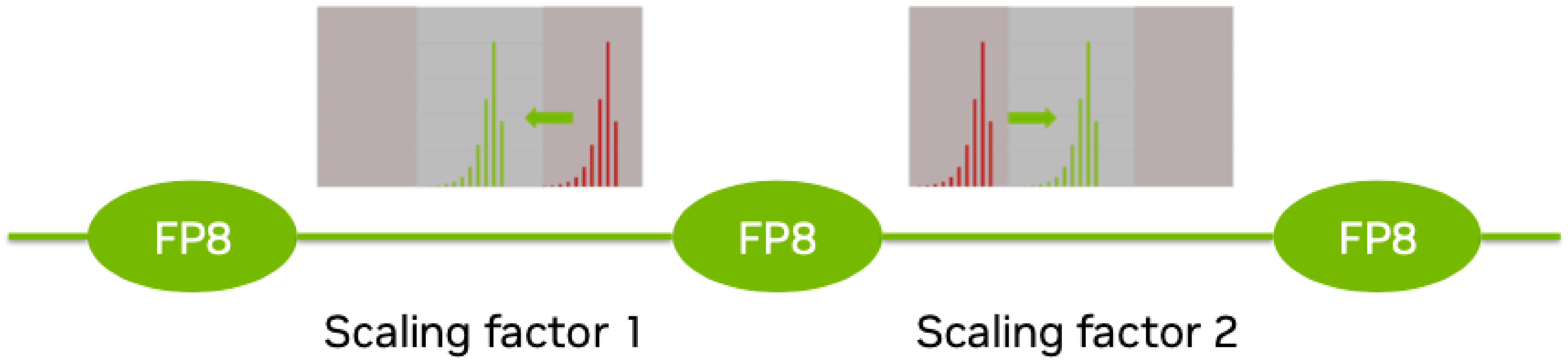
FP16とFP8におけるMixed Precisionの違い

Scaling Factor

- FP16: 単一のScaling factorをbackward passにのみ用いる
  - Scaling factorの変更はOverflow/Underflow発生時に行う



- FP8: Tensor毎にScaling factorを持つ
  - Scaling factorはforward/backward両方に対して必要
    - E4M3 for forward, E5M2 for backward
  - 単一のScaling factorでは精度を担保できない



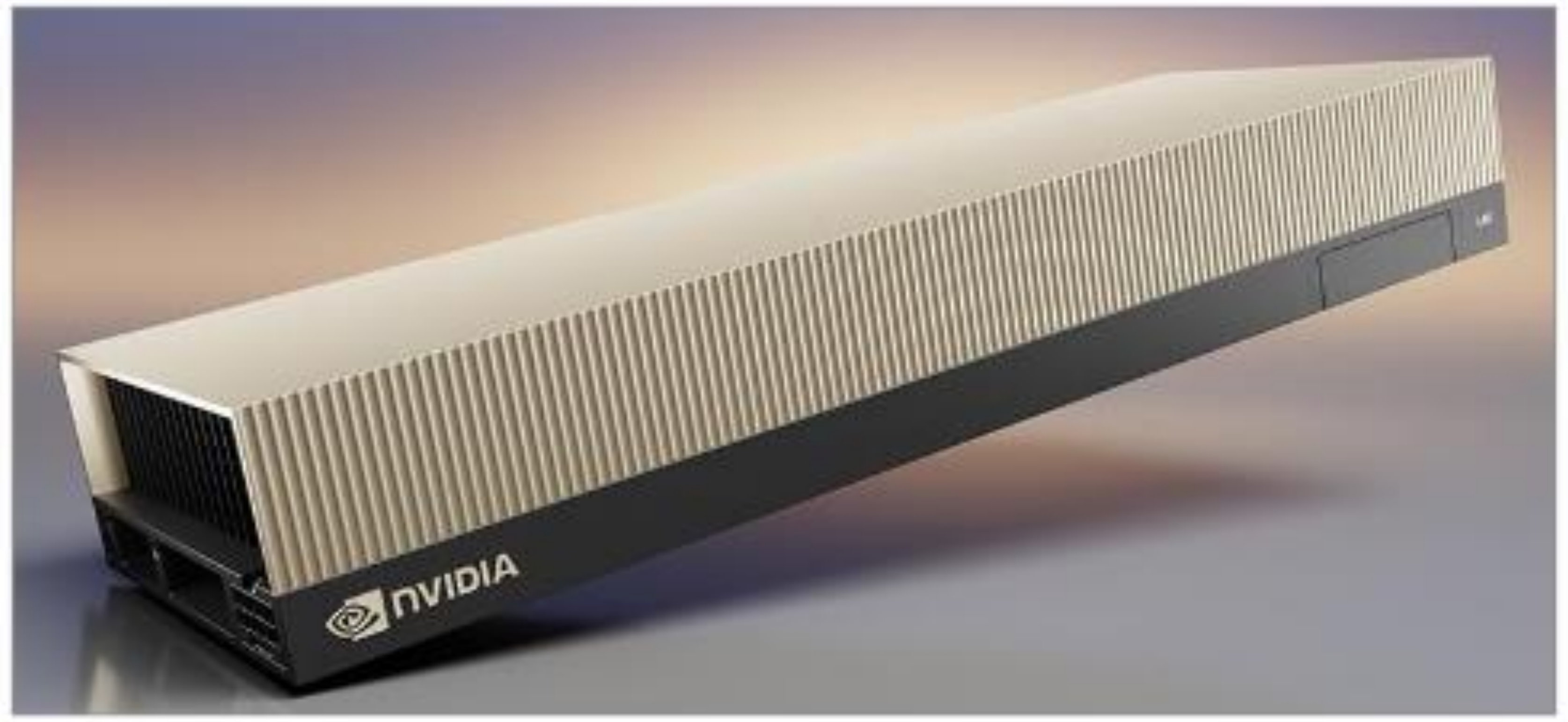
Transformer Engine

FP8のためのライブラリ

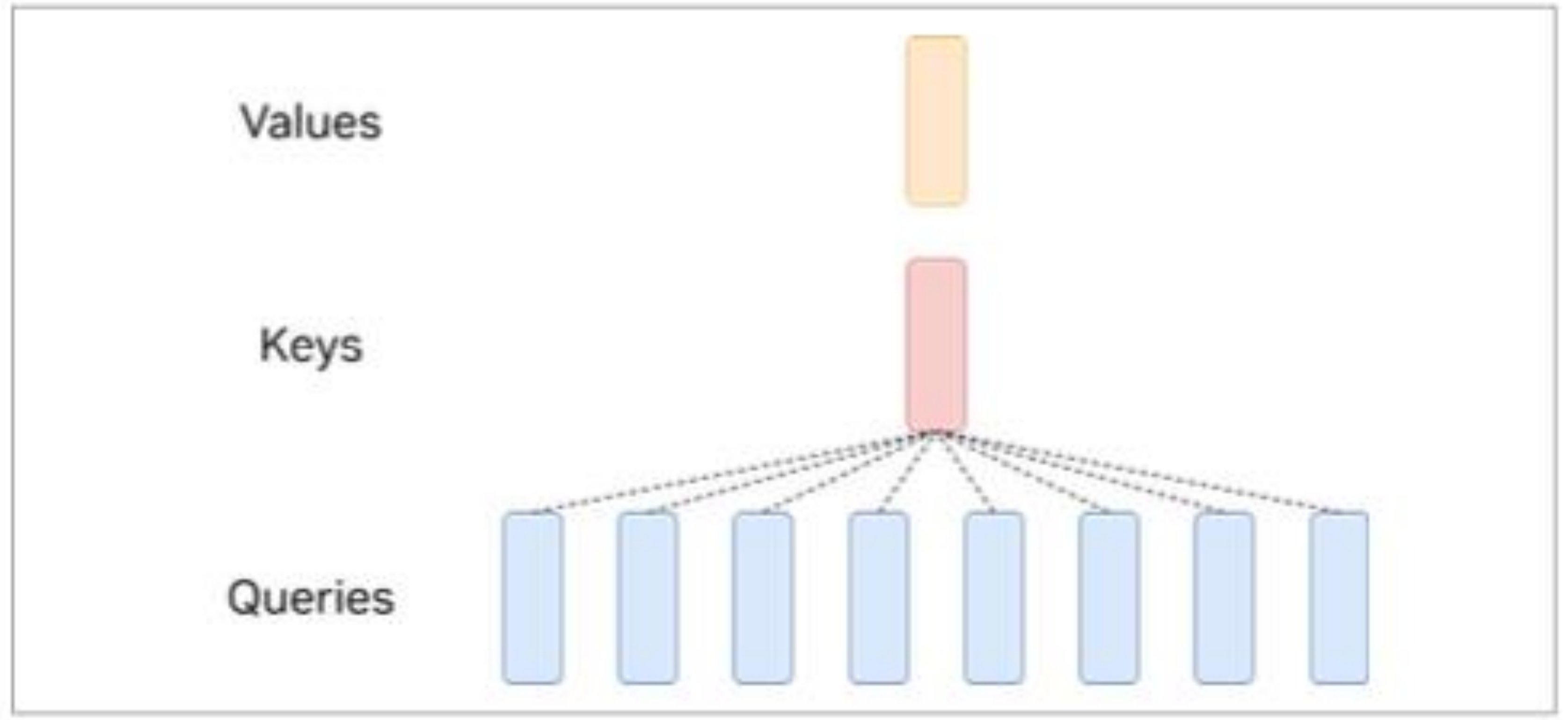
- Transformerブロックを高速に学習するためのオープンソースライブラリ
- FP8に最適化されたOperator
- PyTorch, JAX, PaddlePaddleをサポート
- 各フレームワークのNative Operatorと組み合わせ可能
- 分散学習のための各種Parallelismをサポート
- **https://github.com/NVIDIA/TransformerEngine**

最近の新機能

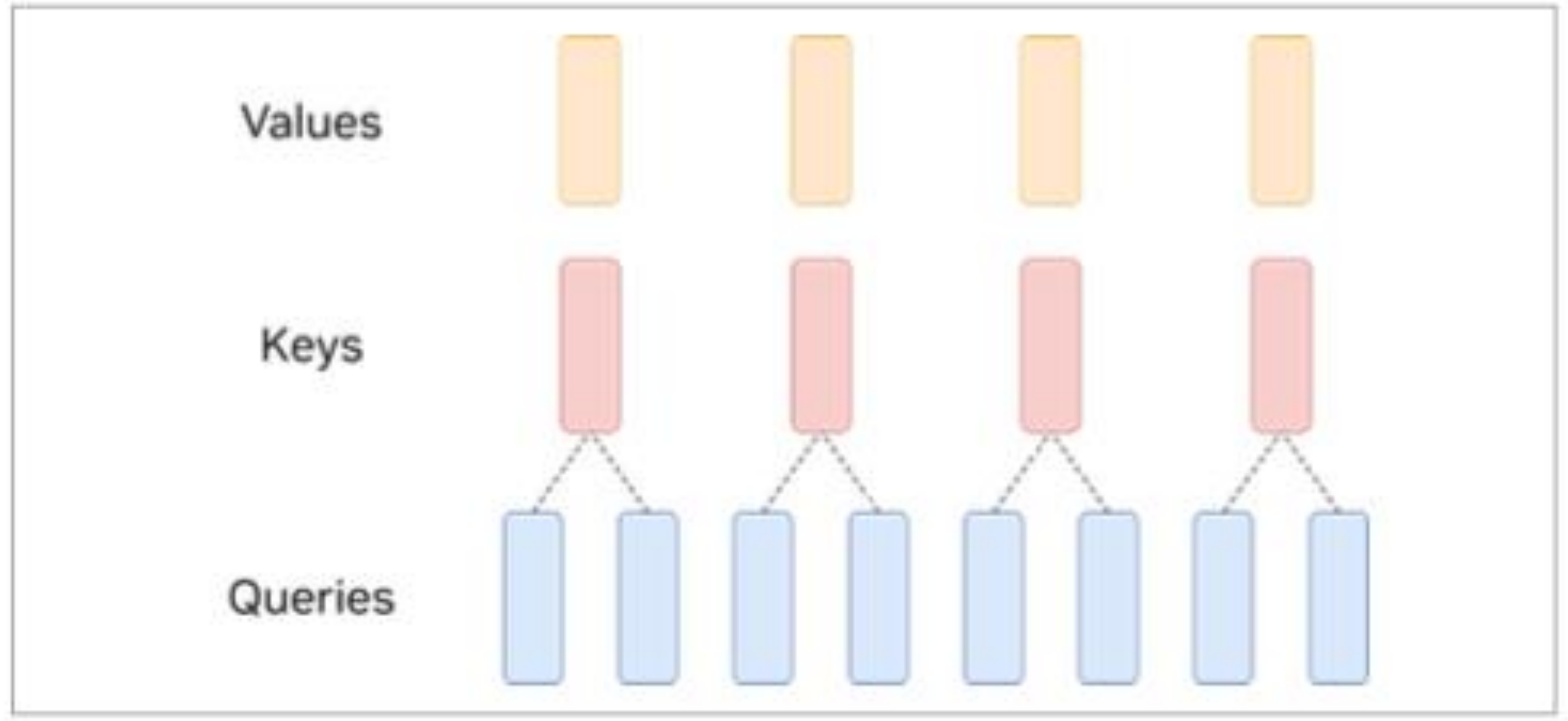
Since last GTC



Support FP8 on Ada GPUs



Multi-query attention



Grouped-query attention

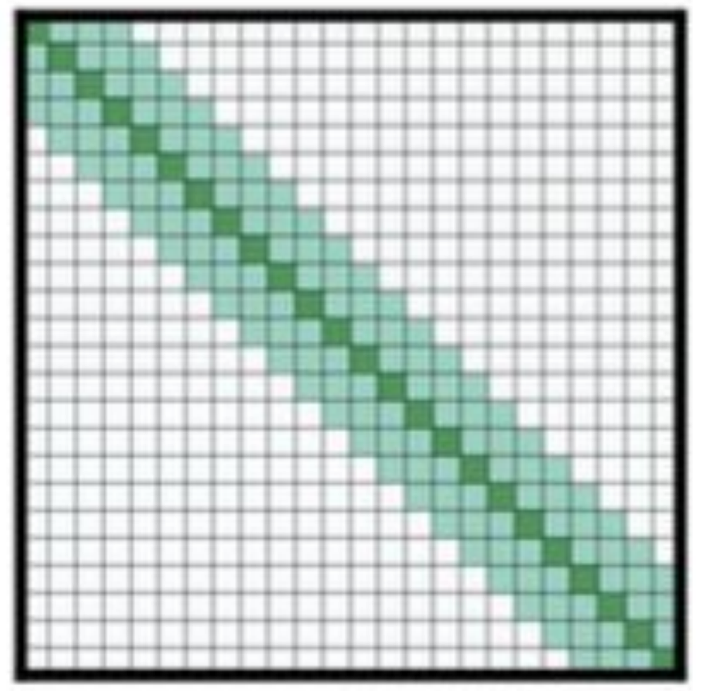
$$y = \frac{x}{RMS_{\varepsilon}(x)} * \gamma$$

where

$$RMS_{\varepsilon}(x) = \sqrt{\frac{1}{n}\sum_{i=0}^{n} x_i^2 + \varepsilon}$$

RMSNorm

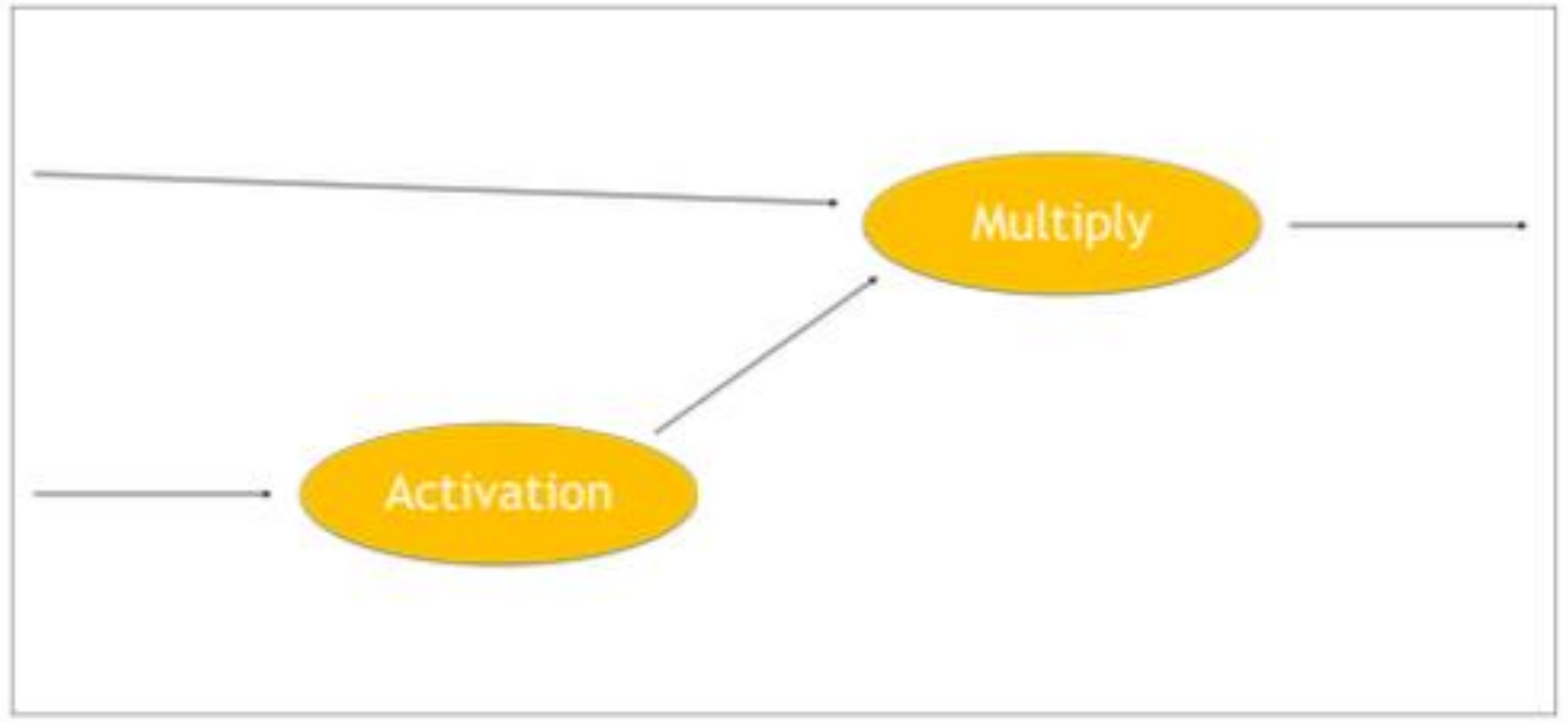
Including zero-centered gamma support



Sliding Window attention

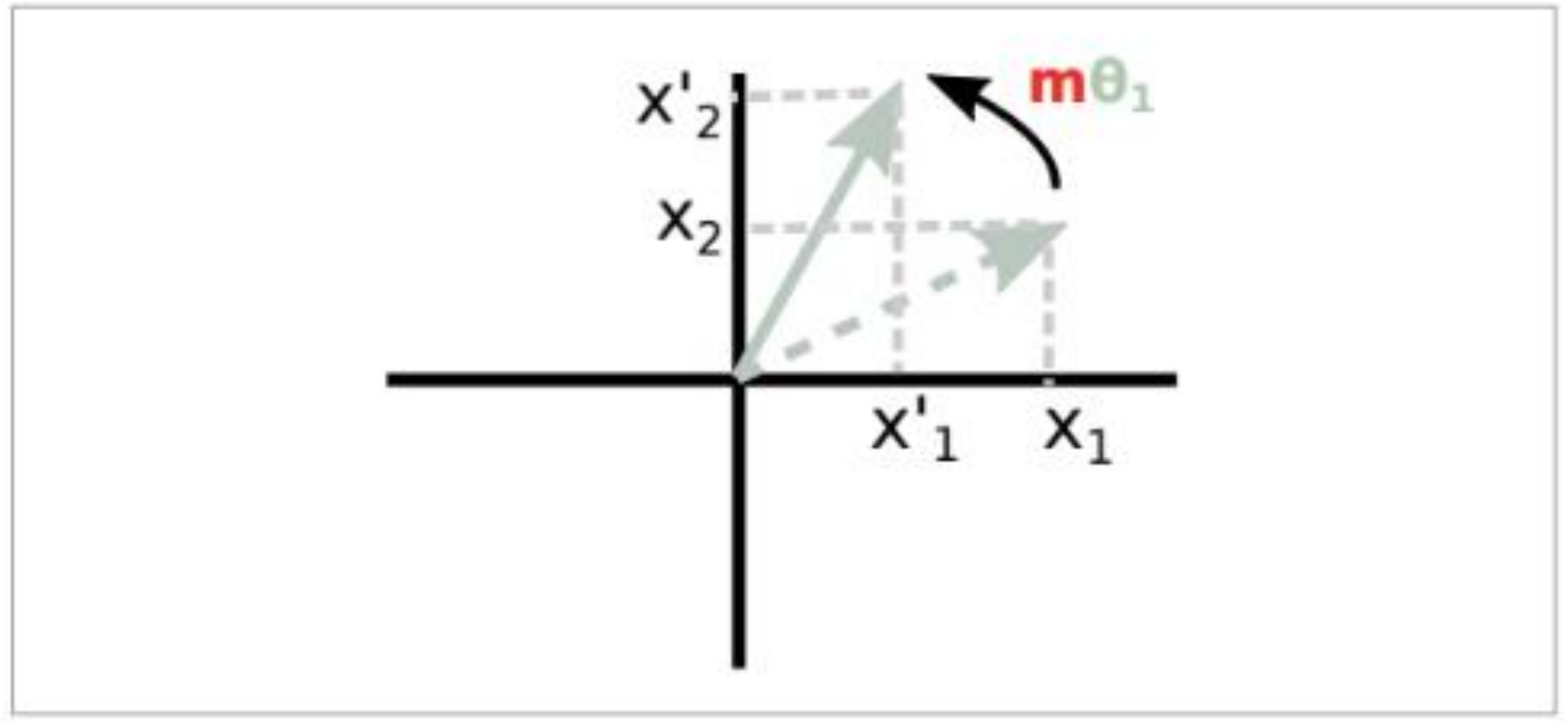
最近の新機能

Since last GTC



GLU activations

ReGLU, GeGLU, SwiGLU

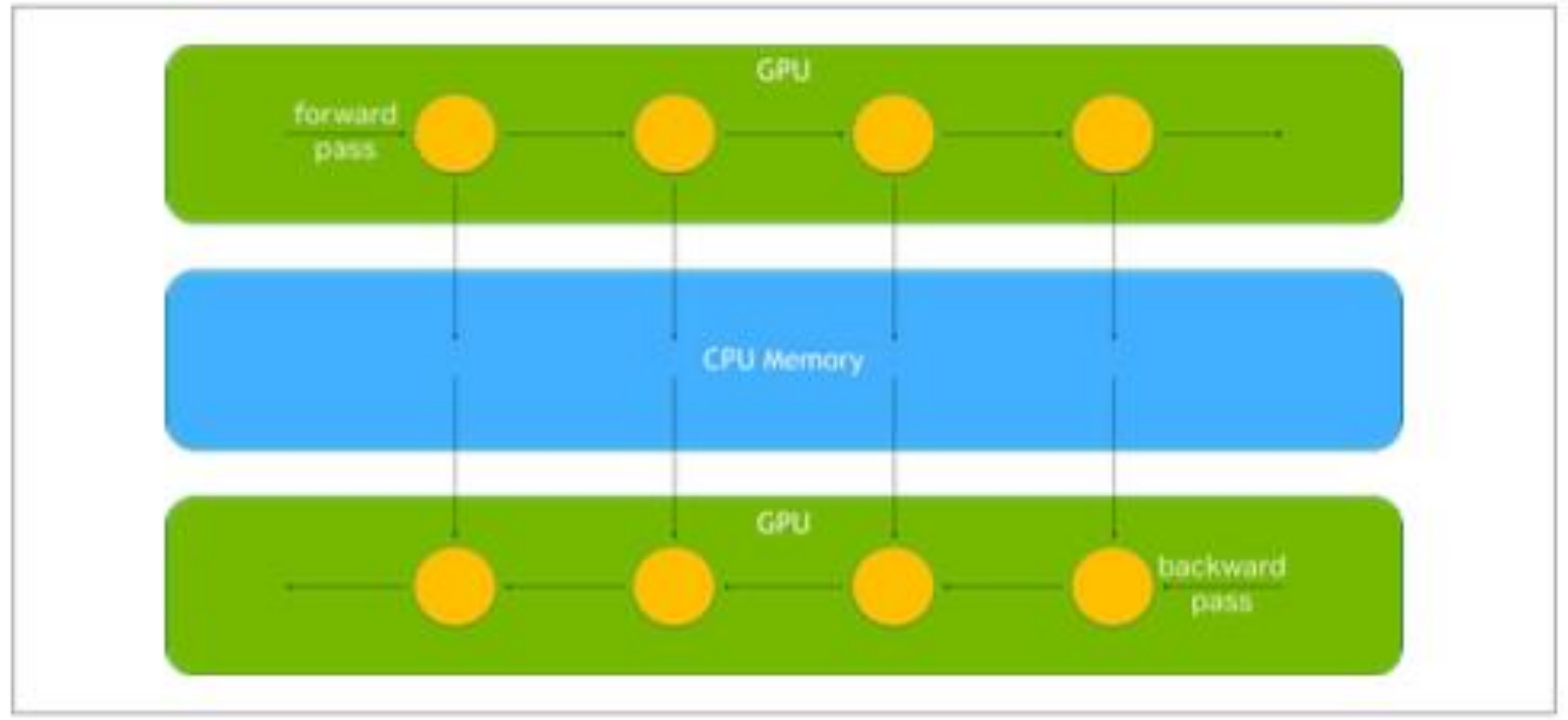


Rotary Position Embeddings

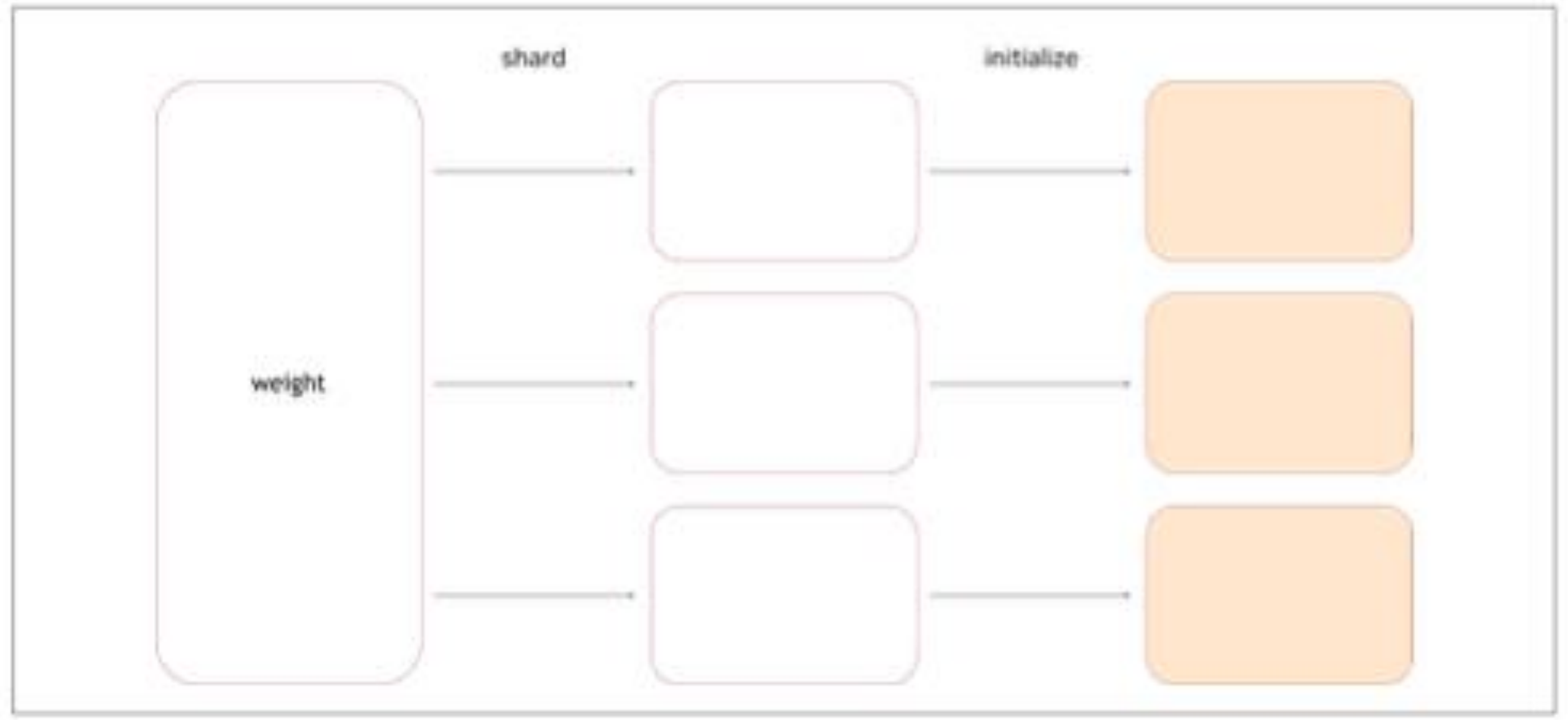
Fast fused implementation

Parallel Transformer

Falcon architecture



CPU offloading of activations



Lazy weight initialization in PyTorch

Using “meta” device

インストール方法

- pipでインストール
  - pip install git+https://github.com/NVIDIA/TransformerEngine.git@stable
- NGC containerを使用
  - docker run --gpus all -it --rm nvcr.io/nvidia/pytorch:24.04-py3

Transformer Engine API

Transformerを構成するモジュール

- FP8 supported modules
  - te.Linear (nn.Linear compatible)
  - te.LayerNorm (nn.LayerNorm compatible)
- Optimized core attention
  - te.DotProductAttention
- Fused/combined modules
  - te.LayerNormLinear
  - te.LayerNormMLP
  - te.MultiHeadAttention
  - te.TransformerLayer

Transformer Engine

FP8 autocast

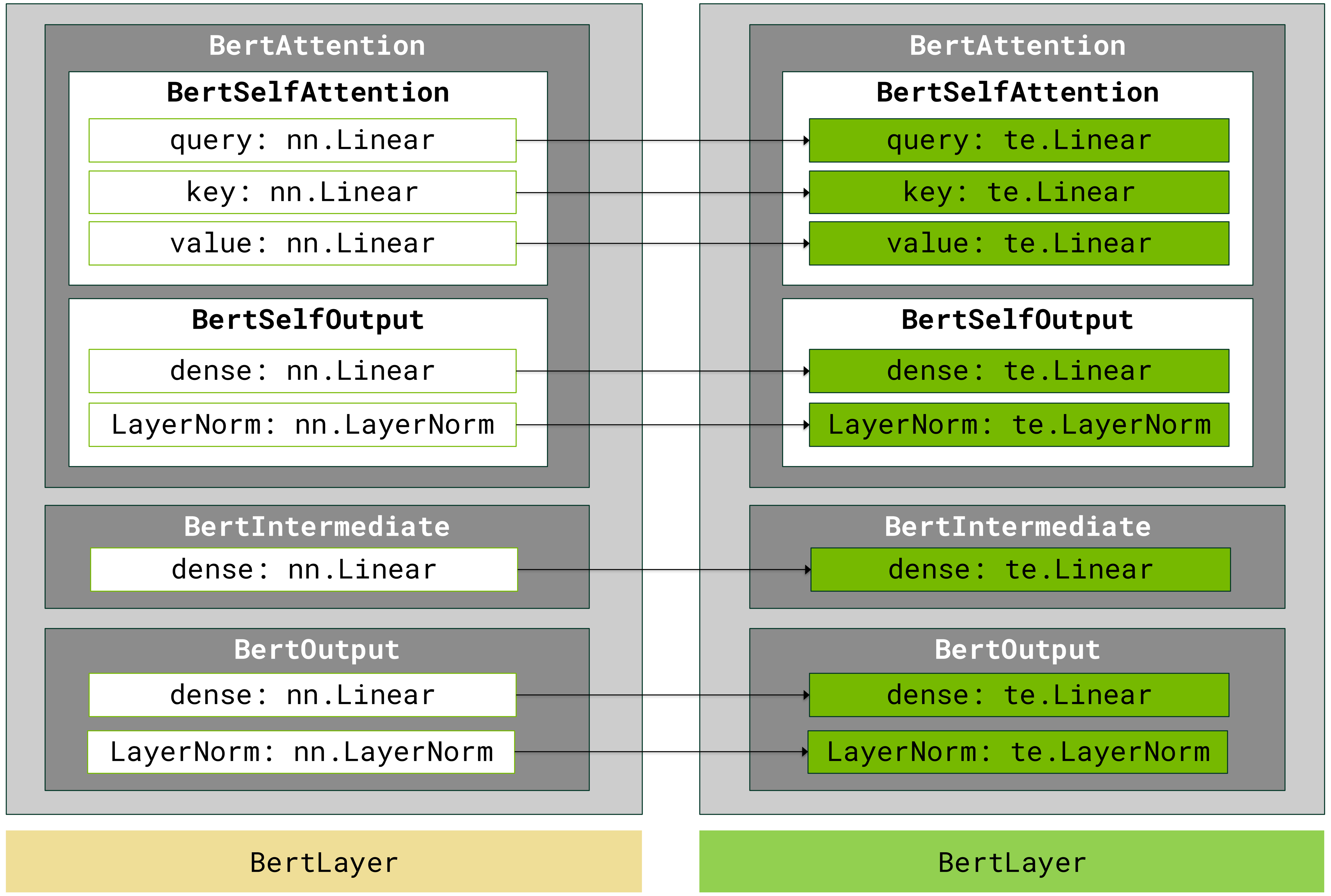
```
# Enable autocasting to FP8.
with te.fp8_autocast(enabled=True,
                     fp8_recipe=fp8_recipe):
    out = model(inp)
loss = out.sum()
loss.backward()
```

- **fp8_autocast** context managerの内部ではTransformer EngineのOperatorでFP8を使用するようになる
- FP8 recipeでhistory windowの長さ、Scaling factorの計算に用いるアルゴリズムなどを指定可能
- モデルのうちTransformer Engineのモジュールで構成された部分以外は変更されない
  - FP16/FP8のmixed precision trainingのためには、FrameworkのAMPと組み合わせる or モデル自体を16bitでロードする
- Backward passは **fp8_autocast** の外側で実施する必要あり（FP8のrecipeはforward passのものを引き継ぐ）

既存のモデルへの適用

Linear/LayerNorm の置き換え

- nn.Linear / nn.LayerNormを
  te.Linear / te.LayerNorm に1:1置き換え
- モジュールの階層構造、state_dictは完全に互換
- モデル作成後に簡単に置き換えることができ、
  変換コードがモデルアーキテクチャに依存しない
- Transformer以外でもLinearを含むモデルに適用可能
- te.TransformerLayer を使う場合と比較すると、
  若干パフォーマンスは落ちる
- HF accelerate / PyTorch Lightningなどの3rd-party libraryでFP8を使用する場合は自動的にこの方式でモデルが変換される

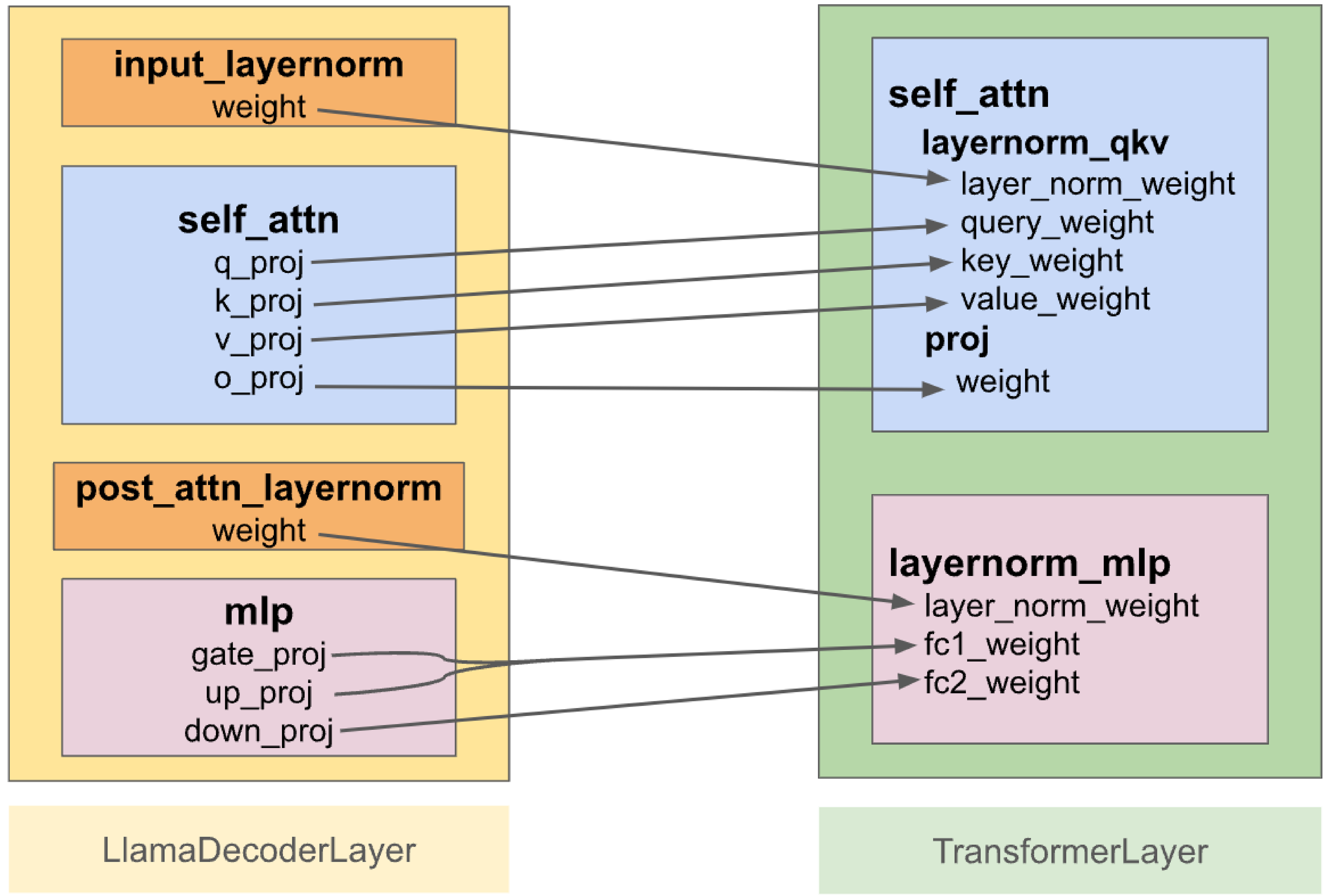



```
for name, module in model.named_children():
    if isinstance(module, nn.Linear):
        te_module = te.Linear(
            module.in_features, module.out_features)
        te_module.weight.copy_(module.weight)
        te_module.bias.copy_(module.bias)
        setattr(model, name, te_module)
    elif isinstance(module, nn.LayerNorm):
        te_module = te.LayerNorm(...
```

既存モデルへの適用

te.TransformerLayer を継承

- te.TransformerLayer を継承して引数/入出力/weightをオリジナルのLayer実装とマッピングする
- 最適化された実装による最大のパフォーマンス
  - FP8を使用しないケースにおいても多くのOSS実装より高速でメモリ効率が良い
- 高速・省メモリなAttention実装
  - Flash Attention, CuDNN Attention
- モジュール構造が変わるため、モデル毎の実装やパラメータのマッピングが必要




```
class TELlamaLayer(te.TransformerLayer):
    def __init__(self, config, layer_idx: int):
        super().__init__(
            config.hidden_size,
            config.intermediate_size,
            config.num_attention_heads,
            activation="swiglu",
            normalization="RMSNorm",
            ...)

    def forward(self, inputs, attention_mask, ...):
        return super().forward(
            inputs, attention_mask, ...)
```

Huggingface Llama2における公式チュートリアル
https://docs.nvidia.com/deeplearning/transformer-engine/user-guide/examples/te_llama/tutorial_accelerate_hf_llama_with_te.html

サードパーティライブラリ経由での利用

Huggingface Accelerate / PyTorch Lightning

- 自動でモデルの一部をTransformer Engineのモジュールに変換し、`fp8_autocast` ラップする
- モデル変換はLayerNorm / Linearを1:1で置き換え
- Transformer Engineのモジュールに置き換えられた部分以外のprecisionは各ライブラリの実装に依存

Huggingface Accelerate:

```
from accelerate import Accelerator
from accelerate.utils import FP8RecipeKwargs

kwargs = [FP8RecipeKwargs(backend="te", ...)]
accelerator = Accelerator(mixed_precision="fp8", kwarg_handlers=kwargs)

model, optimizer, train_dataloader, eval_dataloader, lr_scheduler = accelerator.prepare(
    model, optimizer, train_dataloader, eval_dataloader, lr_scheduler
)
```

PyTorch Lightning:

```
# Select 8bit mixed precision via TransformerEngine, with model weights in bfloat16
trainer = Trainer(precision="transformer-engine")

# Select 8bit mixed precision via TransformerEngine, with model weights in float16
trainer = Trainer(precision="transformer-engine-float16")

# Customize the fp8 recipe or set a different base precision:
from lightning.trainer.plugins import TransformerEnginePrecision

recipe = {"fp8_format": "HYBRID", "amax_history_len": 16, "amax_compute_algo": "max"}
precision = TransformerEnginePrecision(weights_dtype=torch.bfloat16, recipe=recipe)
trainer = Trainer(plugins=precision)
```

NeMo / Megatron-Coreでの利用

最も簡単なTransformer Engineの利用方法

- Megatron-CoreのGPT系モデルはTransformer Engineが組み込まれている
- YAMLまたはCLIのフラグで指定するだけで適用可能

by CLI:

```
python /opt/NeMo/examples/nlp/language_modeling/megatron_gpt_pretraining.py \
  --config-path /opt/NeMo-Megatron-Launcher/launcher_scripts/conf/training/gpt3 \
  --config-name 1b_improved \
  model.transformer_engine=True \
  model.fp8=True
```

by YAML:

```
model:
  ## Transformer Engine
  transformer_engine: True
  fp8: True                    # enables fp8 in TransformerLayer forward
  fp8_e4m3: False              # sets fp8_format = recipe.Format.E4M3
  fp8_hybrid: True             # sets fp8_format = recipe.Format.HYBRID
  fp8_margin: 0                # scaling margin
  fp8_interval: 1              # scaling update interval
  fp8_amax_history_len: 1024 # Number of steps for which amax history is recorded per tensor
  fp8_amax_compute_algo: max # 'most_recent' or 'max'. Algorithm for computing amax from history
  fp8_wgrad: True
  ub_tp_comm_overlap: False
  tp_comm_atomic_ag: False
  tp_comm_atomic_rs: False
```

パフォーマンス向上のためのTips

- Master weightがFP32の場合、FP16/BF16のmixed precisionをかならず併用する
  - `te.fp8_autocast` のみを使った場合、Linear/LayerNormはFP8で計算されるが他はFP32になる
  - モデルを16bitで読み込んでいる場合(full bf16/fp16 training)、特別な対応は不要
- Transformerの各LayerだけではなくOutput(token classification) Dense Layerも`te.Linear`にしておく
  - Language Modelingの場合、多くのケースで最も大きいDense Layer
  - FP8 Tensor Coreを使うためにはvocab_sizeを16の倍数にする必要あり
- Gradient accumulationの活用
  - FP8のscalingとweightの更新はgradient accumulationの最初のiterationのみで行われる
  - オーバーヘッドの削減

FP8での推論

3つの方法

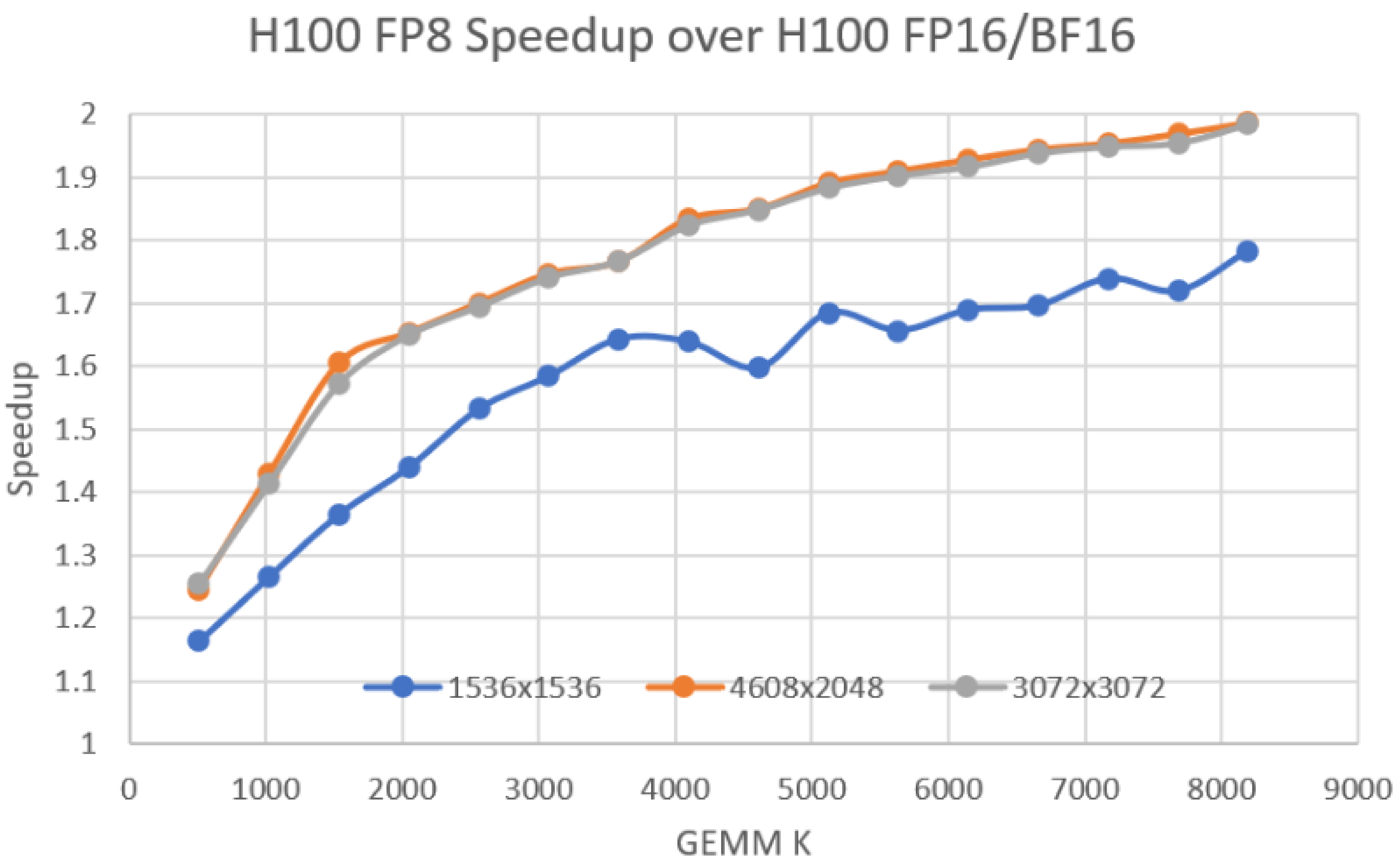
- PyTorch + TransformerEngine
  - パフォーマンス面では推奨されないが、モデル学習中のクイックな検証には有用
  - 学習中のEvaluationであれば、Training loopと同じように `fp8_autocast` で囲むだけで使用できる
  - FP8で学習していないモデルをFP8で推論する場合、Scaling factorの事前キャリブレーションが必要
  - キャリブレーション完了後であれば、FP8のweightのみロードすることも可能

- ONNX -> TensorRT export
  - Transformer EngineはONNXにエクスポートするためのhelper methodが実装されている
  - 主にLLM以外の推論向け

- TensorRT LLM / NIM LLM
  - 特定のLLMアーキテクチャ向けに最もパフォーマンスの良い方法
  - TP/PP/EPなど各種Parallelismに対応
  - TensorRT LLMは独自のPost-Training Quantization機能を持っているため、TEで学習したScalingは使わない
  - FP8 Post-Training Quantization on TRT-LLM doc:
    - https://github.com/NVIDIA/TensorRT-LLM/blob/v0.9.0/examples/llama/README.md#fp8-post-training-quantization

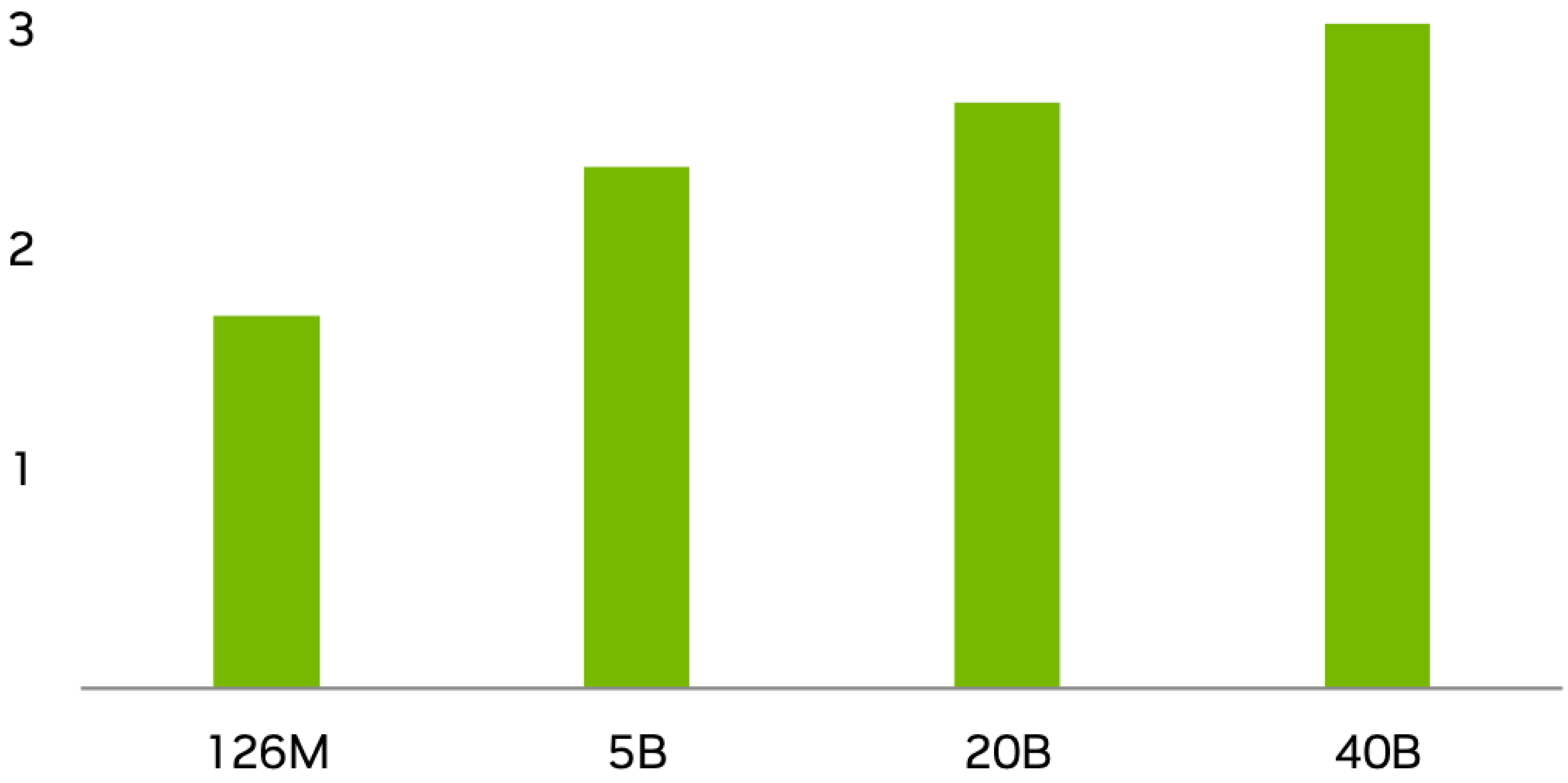
Performance Results

GEMM only



Performance Results

GPT-3 model using NeMo Megatron, H100 FP8 vs A100 FP16/BF16



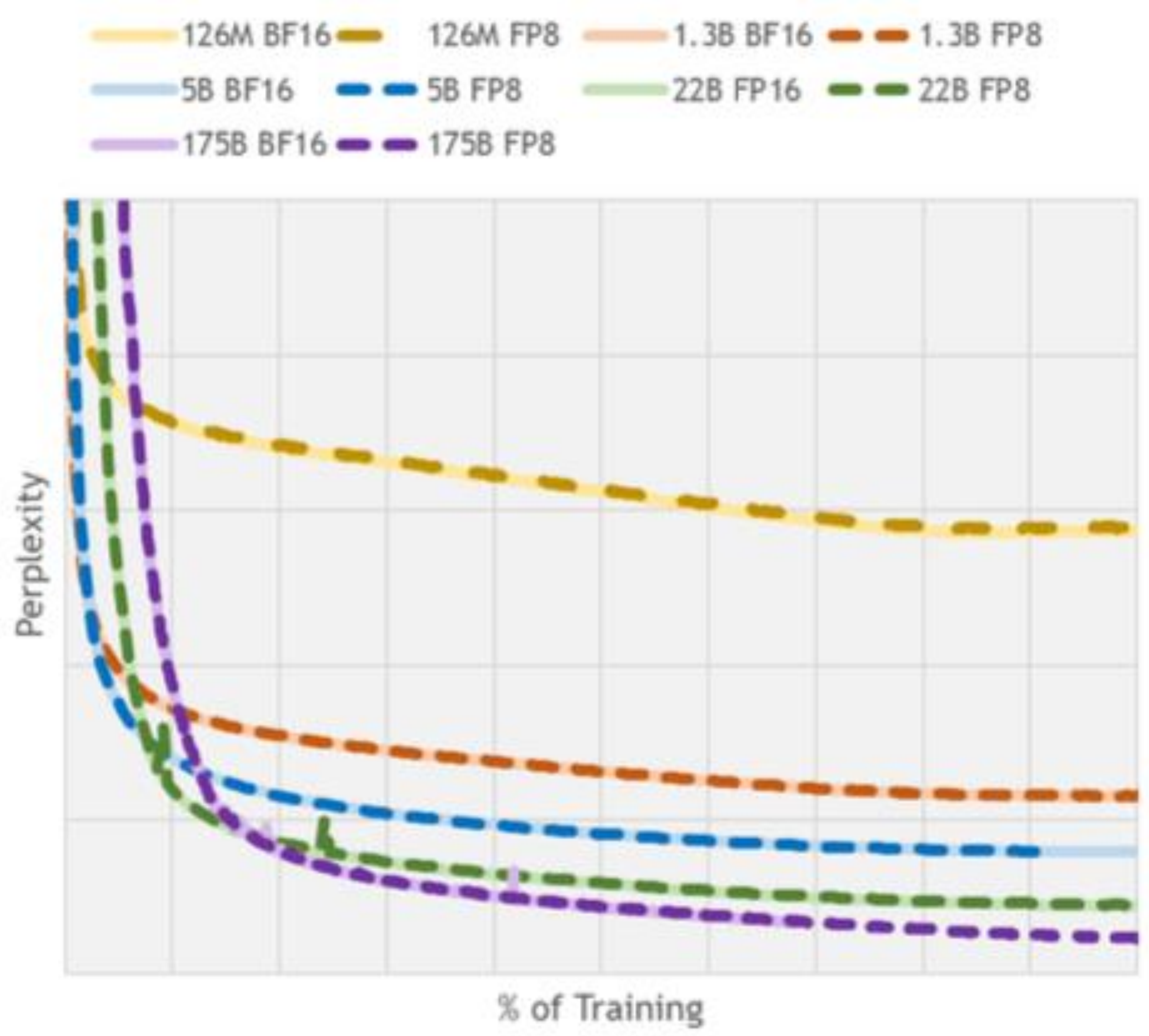
End -to-end training throughput comparison, results gathered using BigNLPcontainer version 23.02 on 8 nodes (64 GPUs).

精度への影響

Network	Metric	16-bits	FP8
GNMT	BLEU	24.83	24.65[1]
Vaswani Base	BLEU	26.87	26.83[1]
Vaswani Large	BLEU	28.43	28.35[1]
Transformer-XL Base	PPL*	22.71	22.76
Transformer-XL Large	PPL*	17.90	17.85
Megatron BERT Base	Loss*	1.352	1.357[1]
Megatron BERT Large	Loss*	1.163	1.167[1]
JoC BERT Large	F1	89.89	90.23
T5 Base	F1	91.68	91.88
T5 Large	F1	93.41	93.66
T5 Base	Rouge	42.88	42.88
T5 Large	Rouge	43.84	43.64
GPT 126M	PPL*	19.36	19.50
GPT 357M	PPL*	14.07	14.17
GPT 1.3B	PPL*	10.77	10.78
GPT 5B	PPL*	8.95	8.98
GPT 22B	PPL*	7.21	7.24
GPT 175B	PPL*	6.65	6.68

[1] Experiments on different FP8 recipe choices

* Lower means better



Results based on running on A100 (16-bit Tensor Cores) with FP8 IO and on H100 FP8 Tensor Cores

- 多くのモデルで16bit trainingと比較して大きな差分は観測されていない
- FP8 for Deep Learning(GTC2023): https://www.nvidia.com/en-us/on-demand/session/gtcspring23-s52166/

Useful Links

- Github: https://github.com/NVIDIA/TransformerEngine
- Documentation: https://docs.nvidia.com/deeplearning/transformer-engine/user-guide/
- GTC Sessions:
  - FP8 for Deep Learning(GTC23):
    https://www.nvidia.com/en-us/on-demand/session/gtcspring23-s52166/
  - FP8 Training with Transformer Engine(GTC23):
    https://www.nvidia.com/en-us/on-demand/session/gtcspring23-S51393/
  - What's New in Transformer Engine and FP8 Training(GTC24):
    https://www.nvidia.com/en-us/on-demand/session/gtc24-s62457/
- Japanese blog: https://developer.nvidia.com/ja-jp/blog/introduction-to-fp8-training-using-transformer-engine/

NeMo Framework 概要

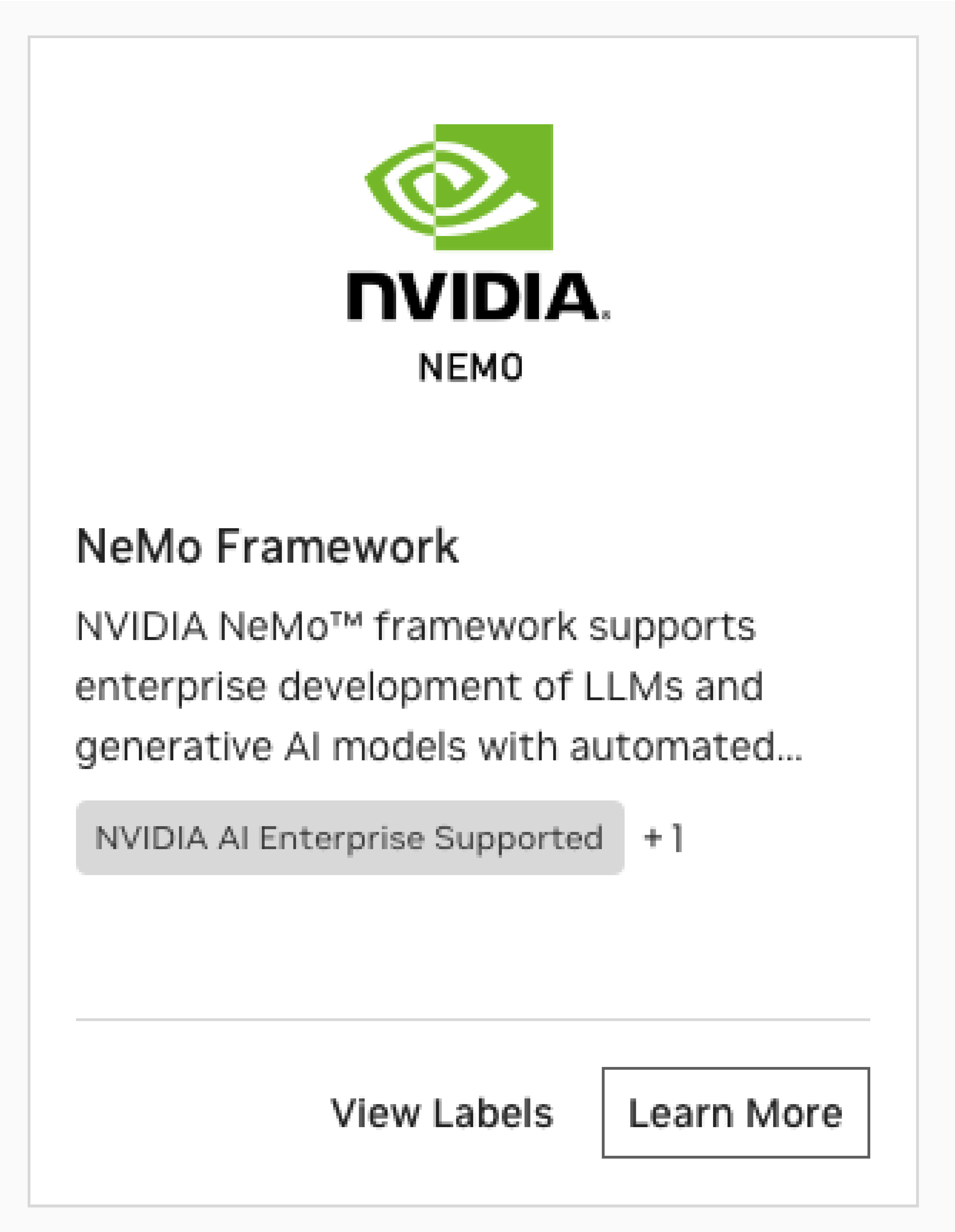
企業向け生成 AI アプリケーションの構築

NVIDIA NeMo による生成 AI モデルの構築、カスタマイズ、展開

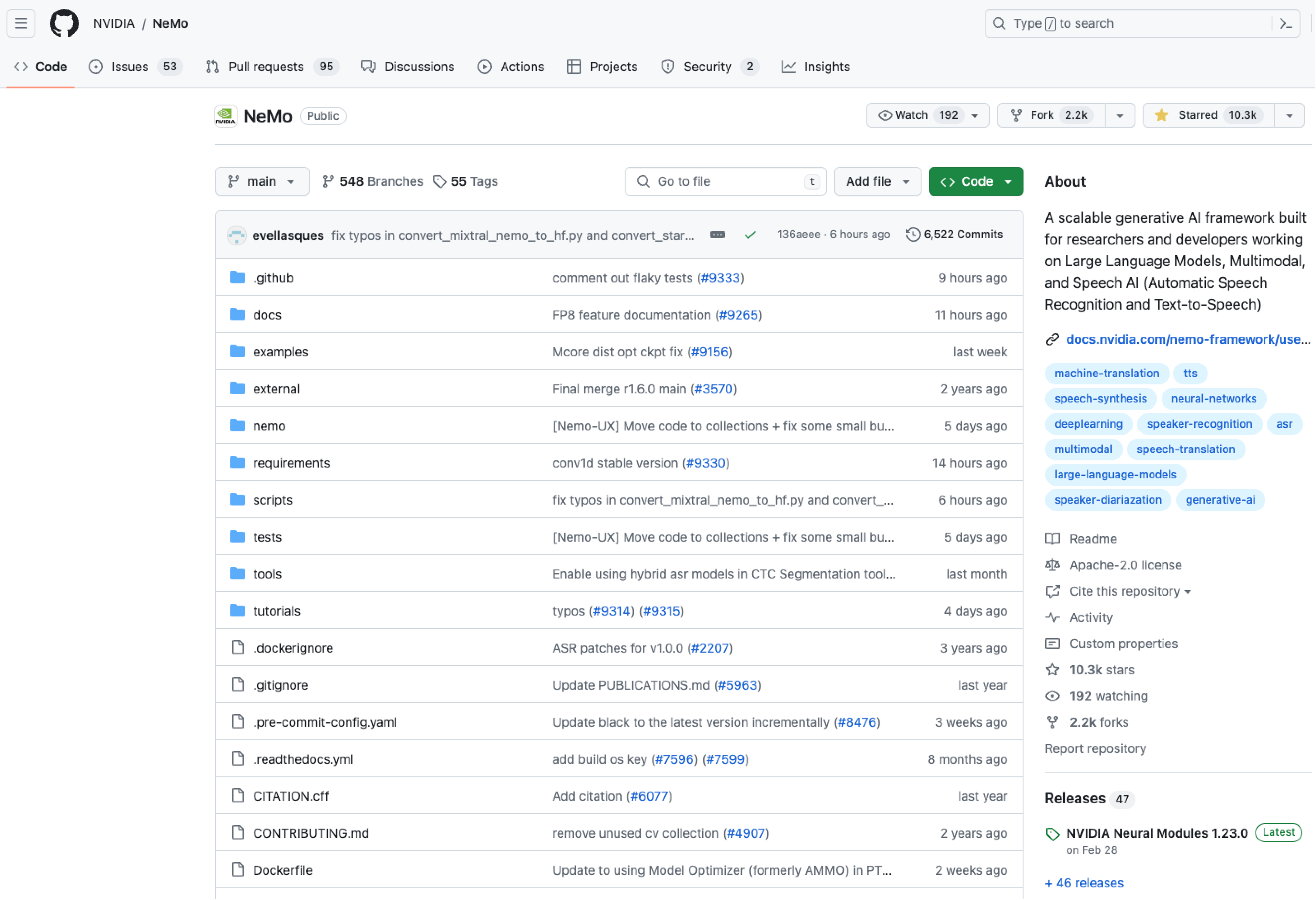
NVIDIA

NVIDIA NeMo Framework

生成 AI モデル構築、カスタマイズの為のエンドツーエンドのクラウドネイティブなフレームワーク



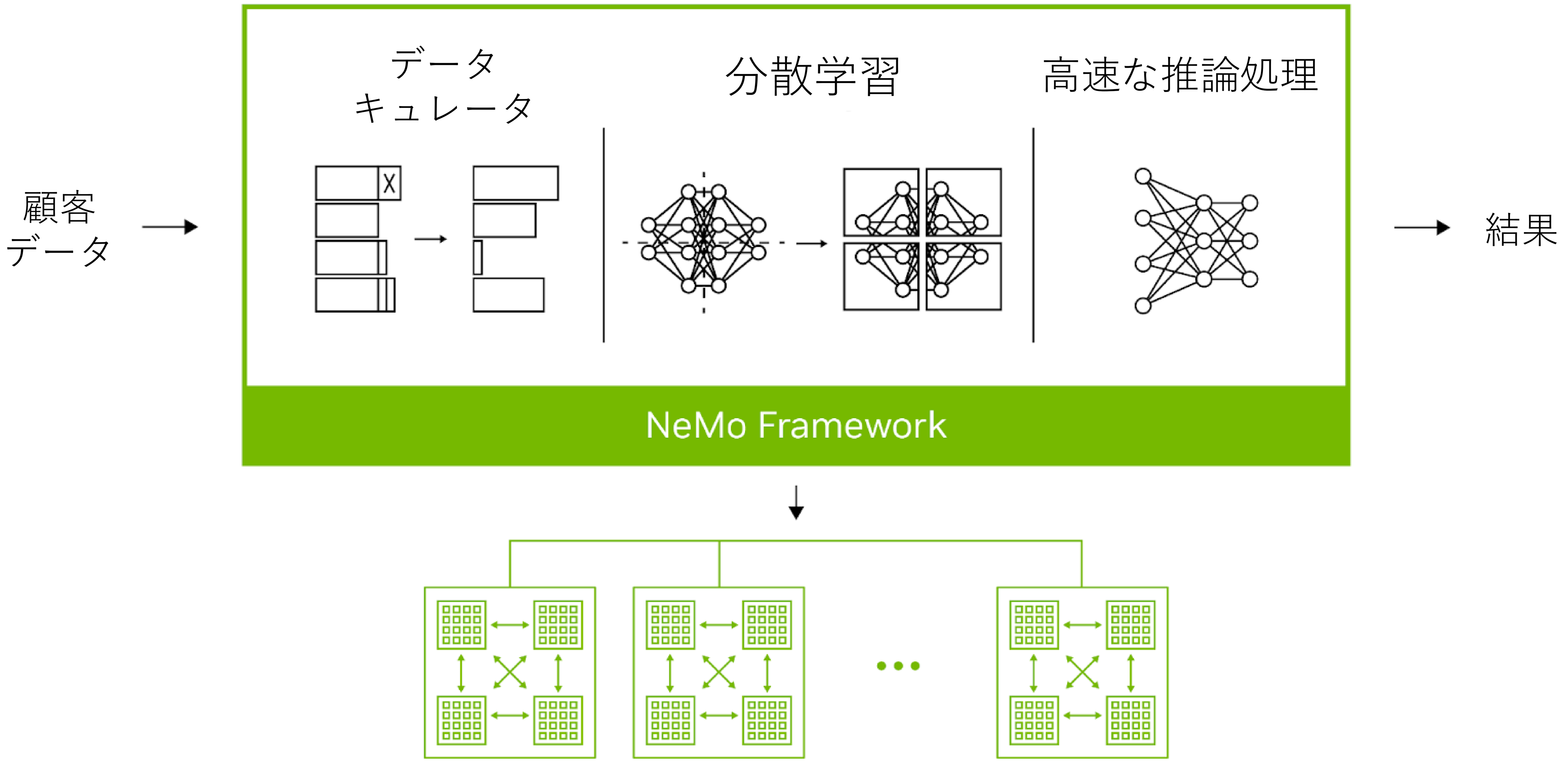
https://catalog.ngc.nvidia.com/orgs/nvidia/containers/nemo



https://github.com/NVIDIA/NeMo

NVIDIA NeMo Framework

生成 AI モデル構築、カスタマイズの為のエンドツーエンドのクラウドネイティブなフレームワーク



NVIDIA DGX SuperPODs

NVIDIA DGX Cloud

NVIDIA DGX Systems

https://developer.nvidia.com/nemo-framework

NeMo Framework コンテナ

✓データの収集、キュレーション

✓高度な分散学習技術

- **Data Parallelism**
- **Fully Sharded Data Parallelism (FSDP)**
- **Tensor Parallelism**
- **Pipeline Parallelism**
- **Sequence Parallelism**
- **Expert Parallelism**
- **Context Parallelism**

✓最適なハイパーパラメータを見つけるための自動コンフィギュレータツール

✓カスタマイズ手法のサポート

- **PEFT, SFT, RLHF, DPO, SteerLM etc.**

✓オーケストレータのサポート: **SLURM, Nephele, Kubernetes**

✓充実したサンプルコードや補助スクリプト

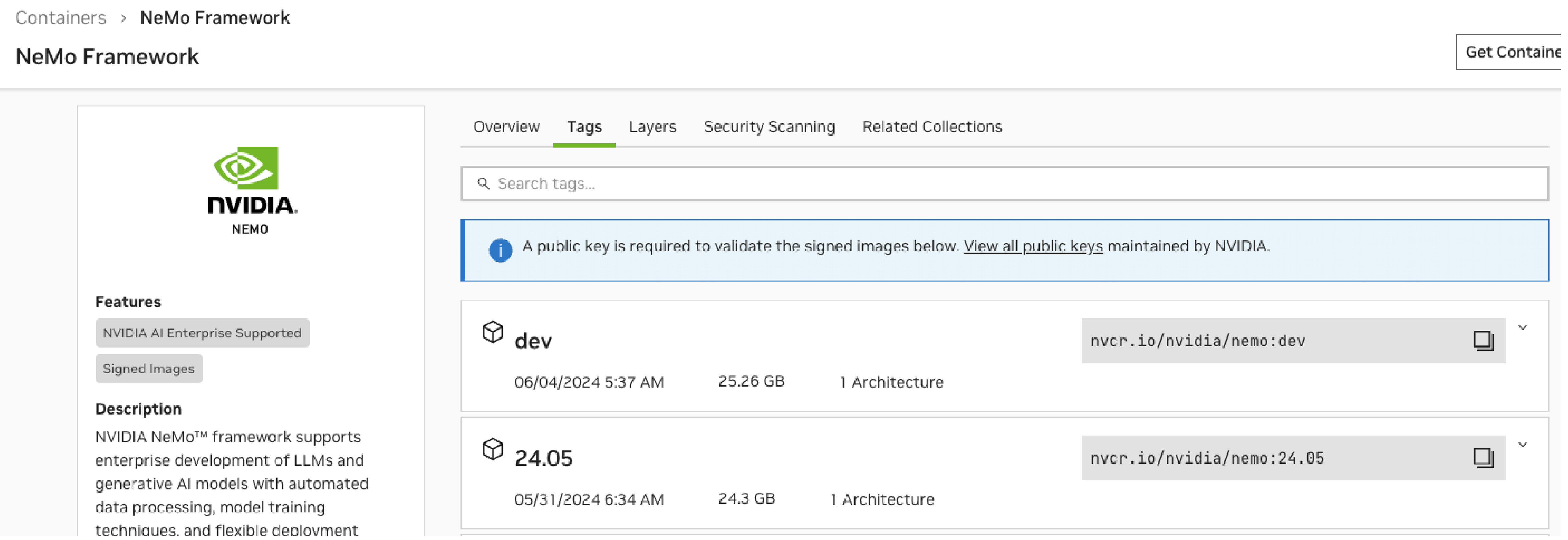
✓モダリティを超えたサポートの拡大

- **LLMs**
- **Multimodal (text2image, VLMなど)**
- **Speech**

NeMoのインストール手順

NGC

- example:
  - docker pull nvcr.io/nvidia/nemo:24.05



Github

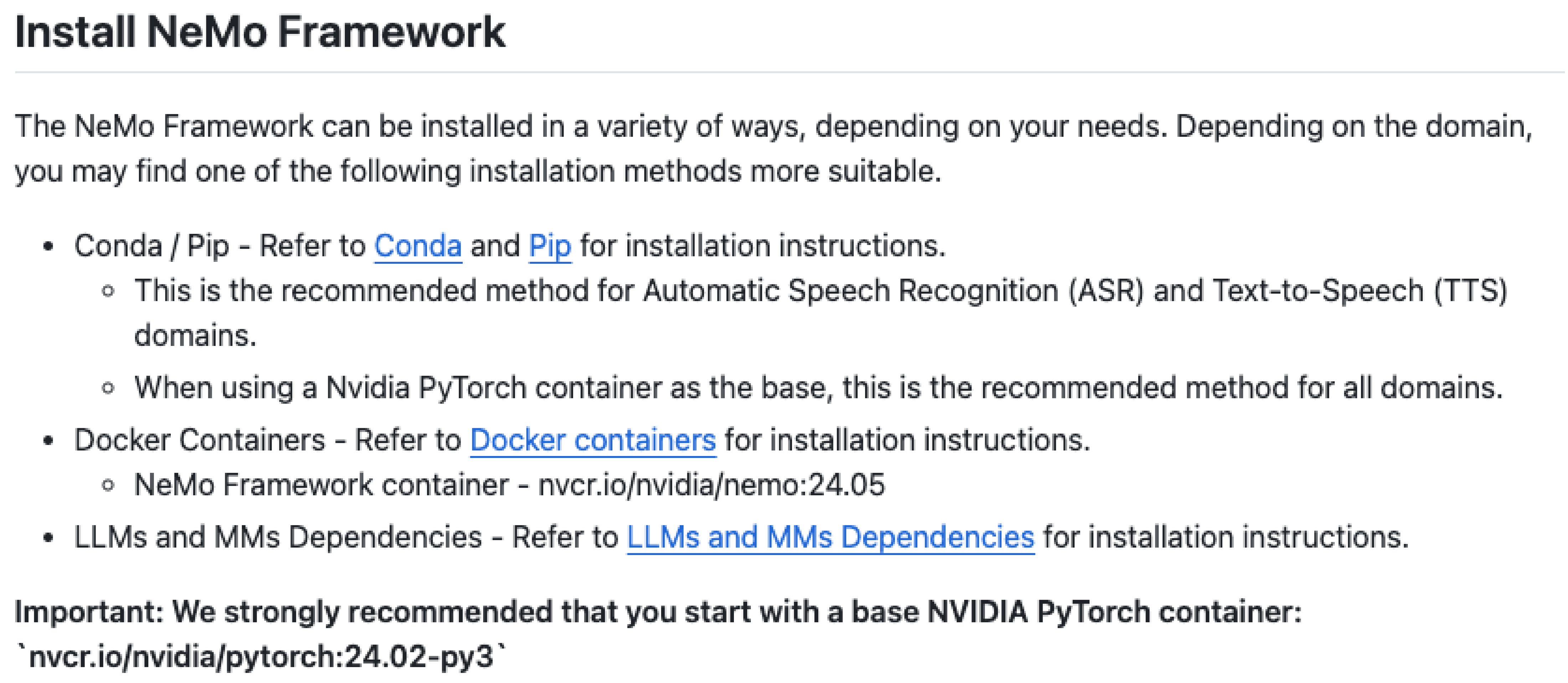
- example:

apt-get update && apt-get install -y libsndfile1 ffmpeg

pip install Cython

pip install nemo_toolkit['all']

- その他、Github上にpip(モダリティ指定)やソースビルド、dockerでの構築手順があります

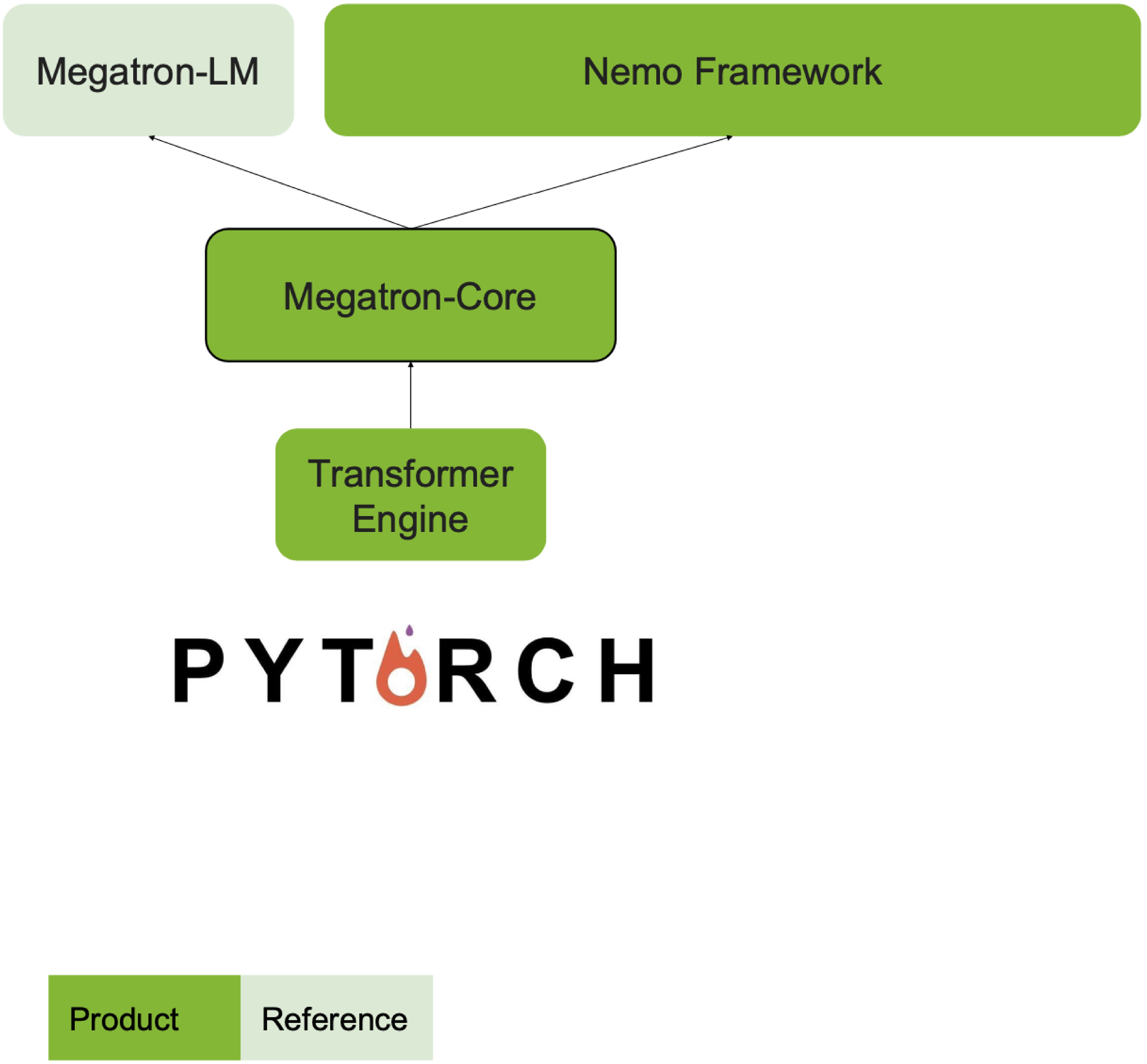


NVIDIA NeMo Frameworkのコンテナ構成

24.05 コンテナでの構成

https://catalog.ngc.nvidia.com/orgs/nvidia/containers/nemo

Megatron-LM と NeMo Framework



Core value Proposition

Nemo Framework: **Easy to use OOTB FW with large model collections** for Enterprise users to experiment, train, and deploy.

Megatron-Core: **Library for GPU optimized techniques for LLM training.** For customers to build custom LLM framework.

Megatron-LM: A **lightweight framework reference** for using Megatron-Core to build your own LLM framework.

Transformer Engine: **Hopper accelerated Transformer models.** Specific acceleration library, including FP8 training.

https://www.nvidia.com/ja-jp/on-demand/session/gtc24-s61626/

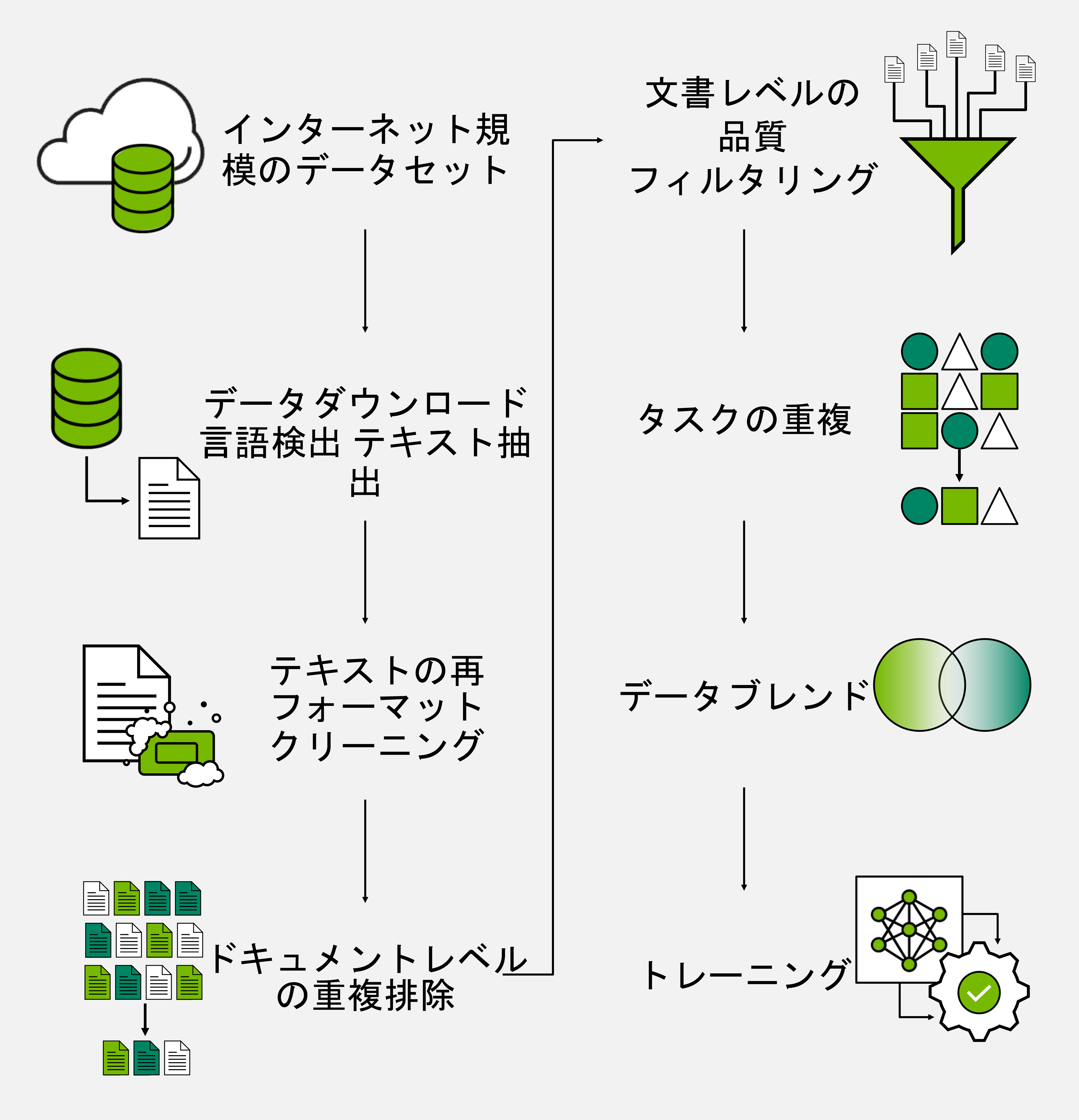
データキュレーションがモデルのパフォーマンスを向上

LLMのための大規模で高品質なデータセットを可能にするNeMo Curator

- 非構造化データソースを検索する負担を軽減
- データをダウンロードし、ドキュメントの抽出、クリーニング、重複排除、フィルタリングを大規模に実行

NeMo Curator のステップ

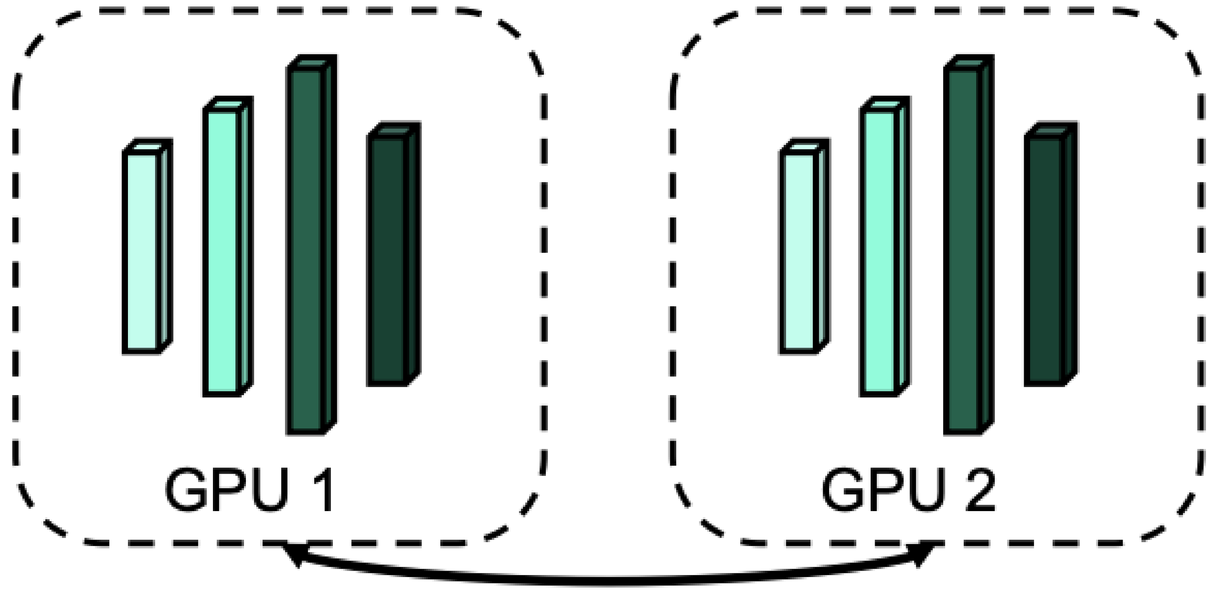
1. データのダウンロード、言語検出、テキスト抽出 - HTMLおよびLaTeXファイル
2. テキストの再フォーマットとクリーニング - 不正なユニコード、改行、繰り返し
3. GPUアクセラレーションによるドキュメントレベルの重複排除
   - ファジー重複排除
   - 正確な重複排除
4. 文書レベルの品質フィルタリング
   - クラシファイアベースのフィルタリング
   - ヒューリスティックに基づく多言語フィルタリング
5. タスク重複排除 - ドキュメント内の重複排除を実行します。



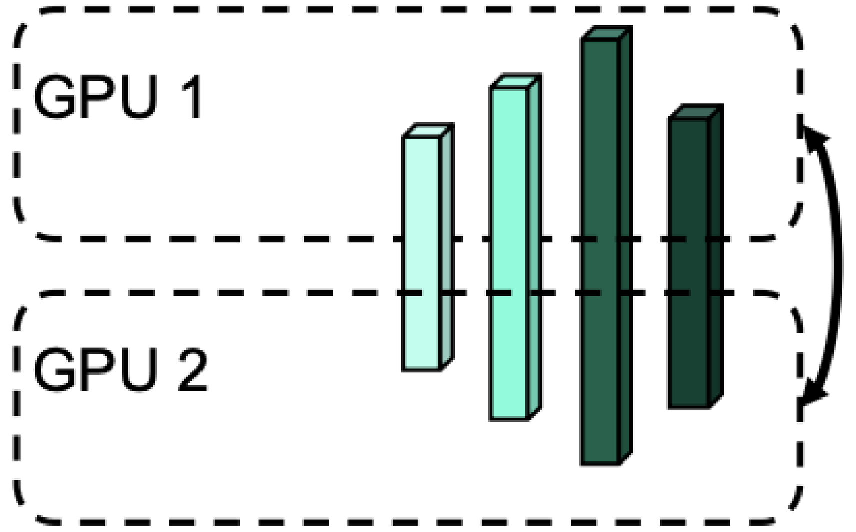
https://github.com/NVIDIA/NeMo-Curator

分散学習

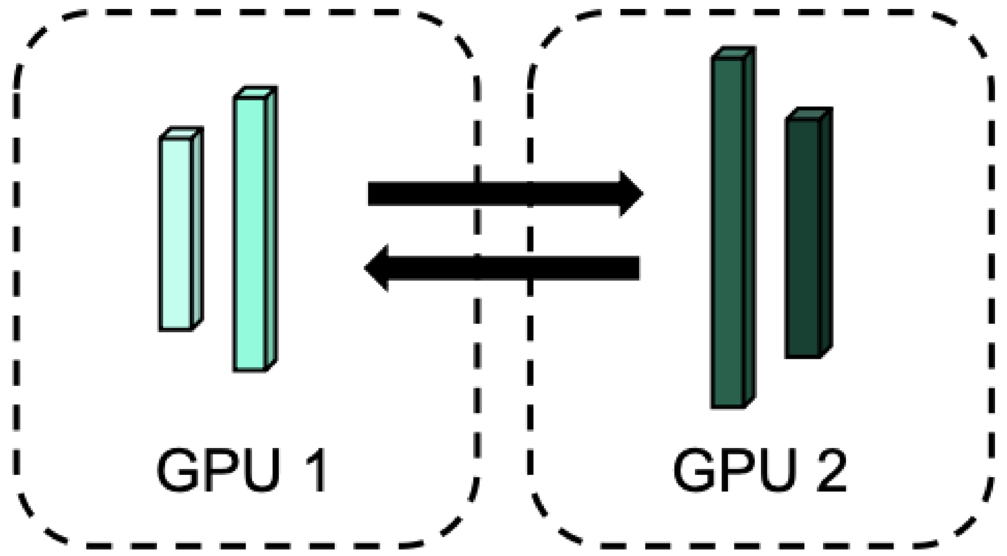
Data Parallelism



Model Parallelism



Tensor(intra-layer) parallelism



Pipeline(inter-layer) parallelism

- **Data Parallelism**
  - 各デバイスはモデルパラメータの複製を持つ
  - 入力データを複数のGPUに分割し、Iteration毎に重み勾配をall-reduceする
- **Model Parallelism**
  - モデルパラメータを複数のGPUに分割
  - **Tensor Parallelism:** 個々のレイヤーを複数のGPUに分割。各デバイスはレイヤー0,1,2,3の異なる部分を計算する。
  - **Pipeline Parallelism:** 複数のGPUにレイヤーを分割。レイヤー0,1とレイヤー2,3は異なるGPU上にある

https://www.nvidia.com/ja-jp/on-demand/session/gtc24-s61626/

Model Parallelism

- **Tensor Parallelism**
  - 各Transformerレイヤーのすべての MLPとself-attentionブロックをGPU間で分割する
  - 各レイヤの1回のフォワードパスとバックワードパスで4回のall-reduceが必要。
  - 通信負荷が高いため、ノード内通信（NVLink）が望ましい

- **Pipeline Parallelism**
  - 複数のGPUにレイヤーを分割。Pipeline ParallelismはPipelineバブルをもたらす
  - **Interleaved Pipeline**
    - 各GPUは、レイヤーの連続したセットではなく、レイヤーの複数のサブセット（モデルチャンクと呼ばれる）に対して計算を実行。レイヤ0,1,4,5とレイヤ2,3,6,7は異なるGPU上にある
    - バブルの割合をモデルチャンクの数だけ減らす
  - ステージ間のpoint-to-point通信が必要だが、これはall-reduceの通信コストに比べるとはるかに安いため、ノード間通信（IBなど）が使える

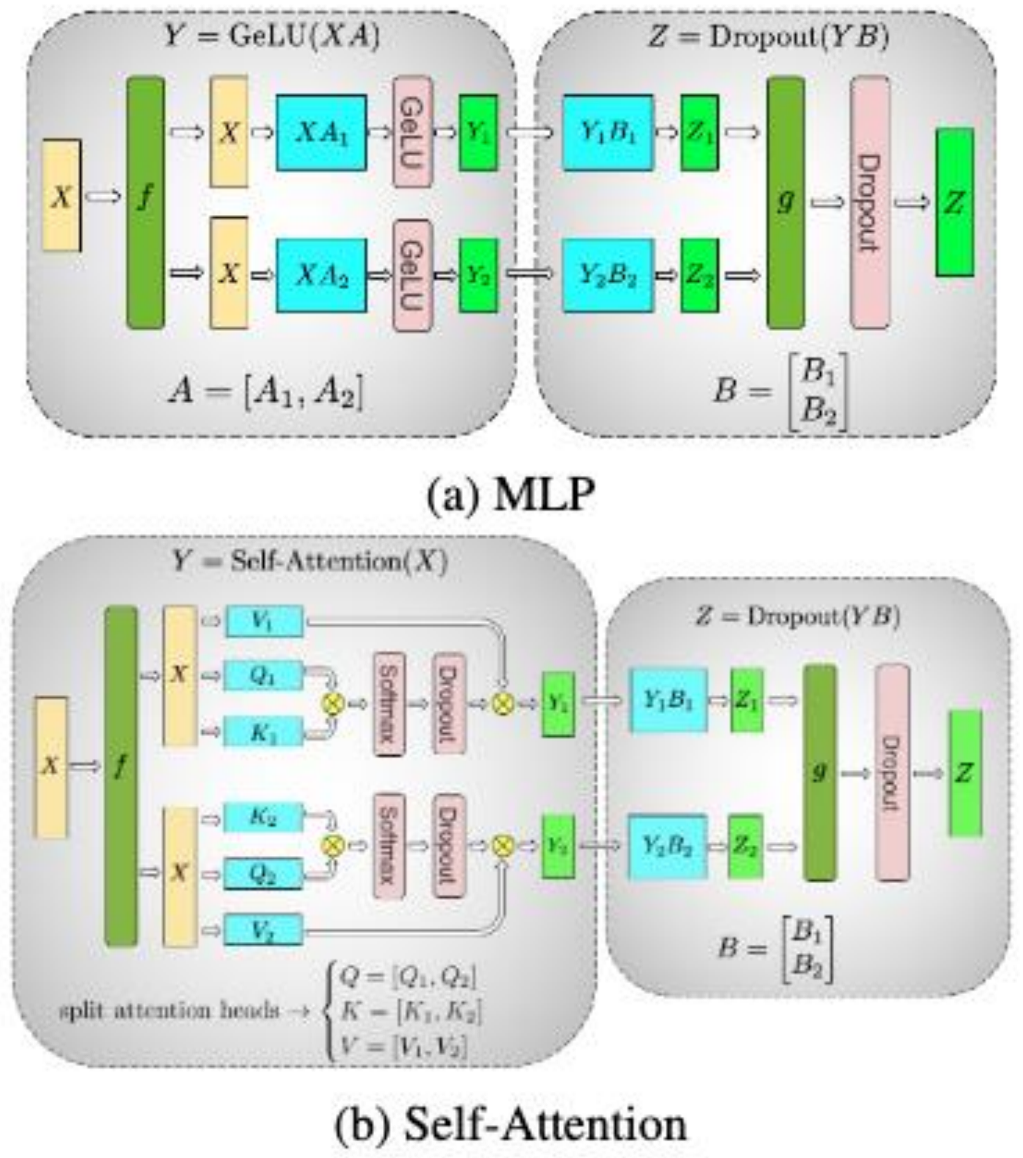


Figure 3. Blocks of Transformer with Model Parallelism. f and g are conjugate. f is an identity operator in the forward pass and all reduce in the backward pass while g is an all reduce in the forward pass and identity in the backward pass.

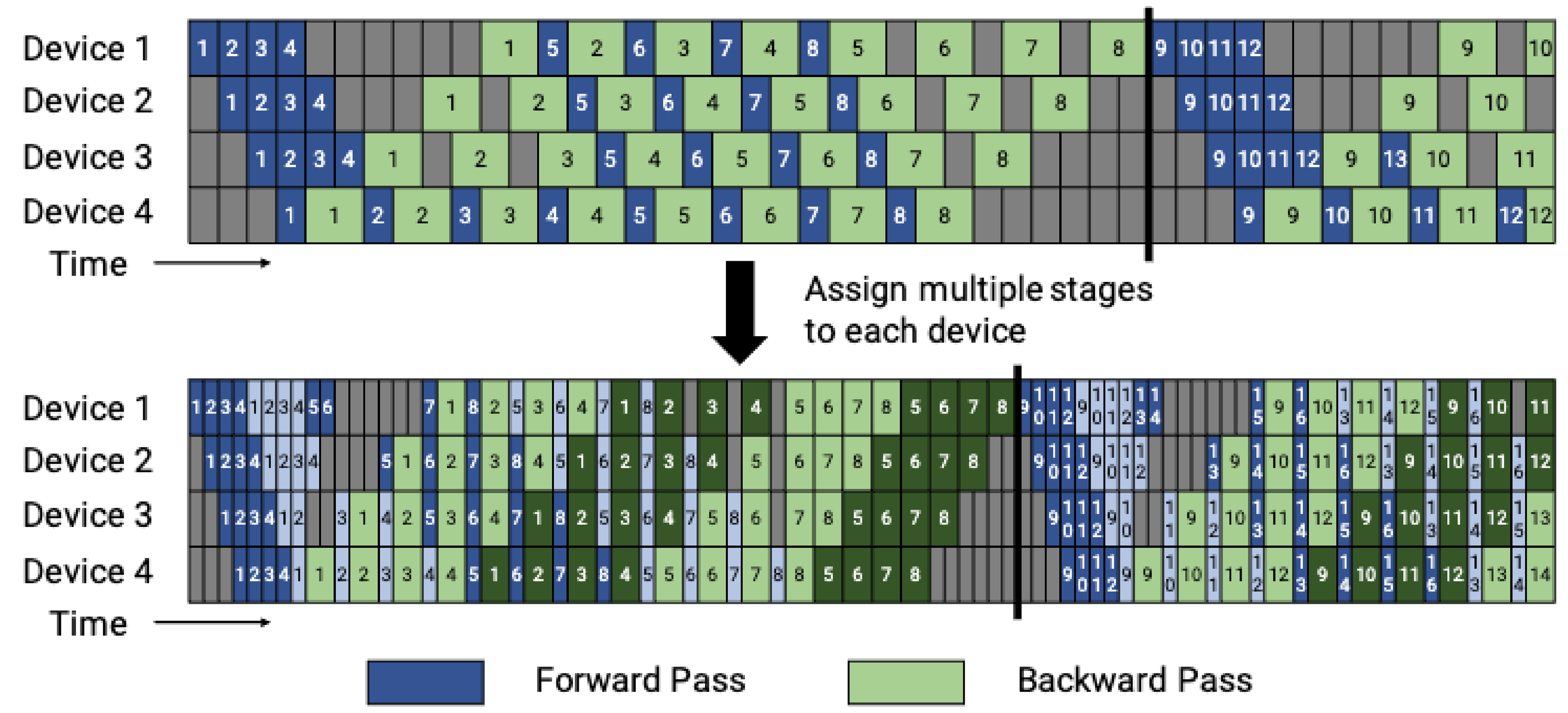


Figure 4: Default and interleaved 1F1B pipeline schedules. The top figure shows the default non-interleaved 1F1B schedule. The bottom figure shows the interleaved 1F1B schedule, where each device is assigned multiple chunks (in this case, 2). Dark colors show the first chunk and light colors show the second chunk. The size of the pipeline bubble is smaller (the pipeline flush happens sooner in the interleaved timeline).

https://arxiv.org/pdf/1909.08053
https://arxiv.org/pdf/2104.04473

Memory Saving

- **Sequence Parallelism**
  - LayerNormとDropoutをシーケンス次元に沿って分割し、activation memoryを削減
  - 4 reduce-scatter + 4 all-gather (~ 4 all-reduce)を (Tensor Parallelism) 1回のフォワードとバックワードで行う
- **Activation Recomputation**
  - メモリに保存する代わりに、バックワードパス中にactivationを再計算。
  - **Full activation recomputation**
    - Transformerレイヤーのすべての活性化を再計算
    - 必要なメモリを大幅に削減できるが、30～40%の計算オーバーヘッドが発生
  - **Selective activation recomputation**
    - 計算量とメモリのトレードオフに基づいて、再計算するactivationを選択
    - activationのメモリフットプリントを低減し、再計算のオーバーヘッドをわずかに低減
- **Distributed Optimizer/ZeRO**
  - Data Parallel GPU間でモデルの状態（オプティマイザの状態: 1、勾配: 2、モデルパラメータ: 3）を共有
  - ZeRO3では通信量が増える可能性あり
- **Offloading**
  - CPUのメモリを利用することでGPUのメモリ要件を削減するが、GPU-CPU-GPUの転送オーバヘッドを伴う
  - 勾配、パラメータ、オプティマイザの状態をCPUにオフロード
  - フォワードパスの後にGPUからCPUにactivationをオフロードし、バックワードパスの前にCPUからGPUにプリフェッチバックする

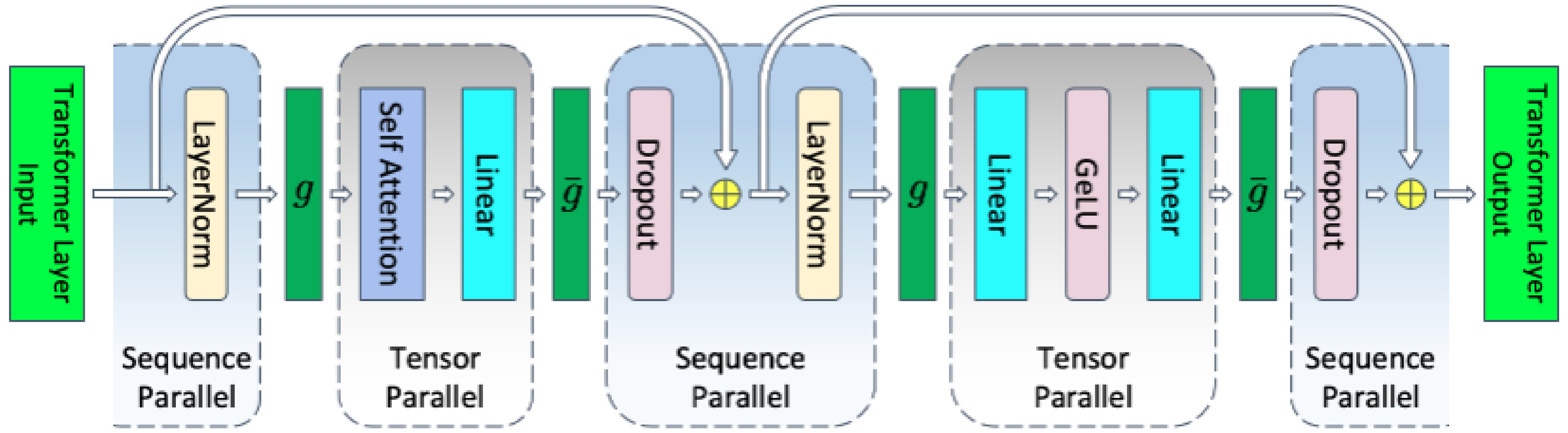


Figure 5: Transformer layer with tensor and sequence parallelism. g and $\bar{g}$ are conjugate. g is all-gather in the forward pass and reduce-scatter in the backward pass. $\bar{g}$ is reduce-scatter in forward pass and all-gather in backward pass.

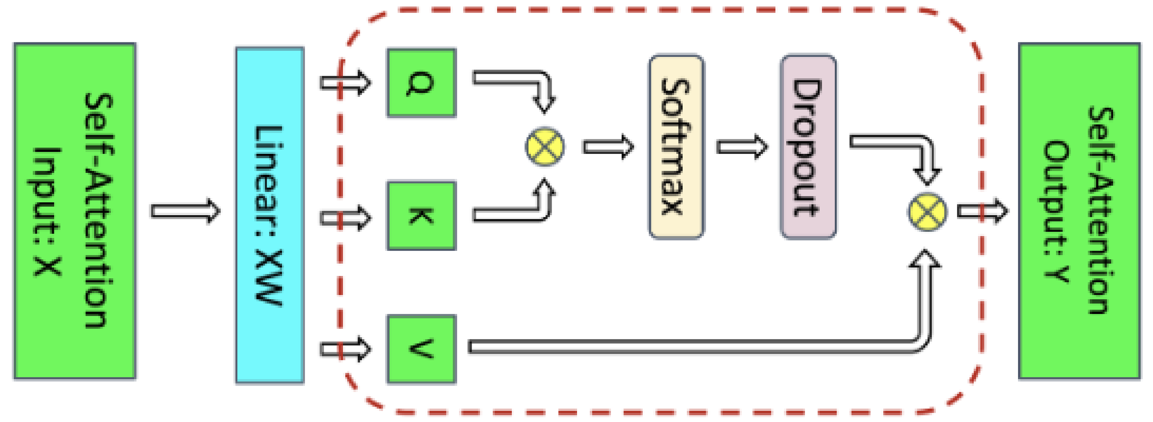


Figure 3: Self-attention block. The red dashed line shows the regions to which selective activation recomputation is applied (see Section 5 for more details on selective activation recomputation).

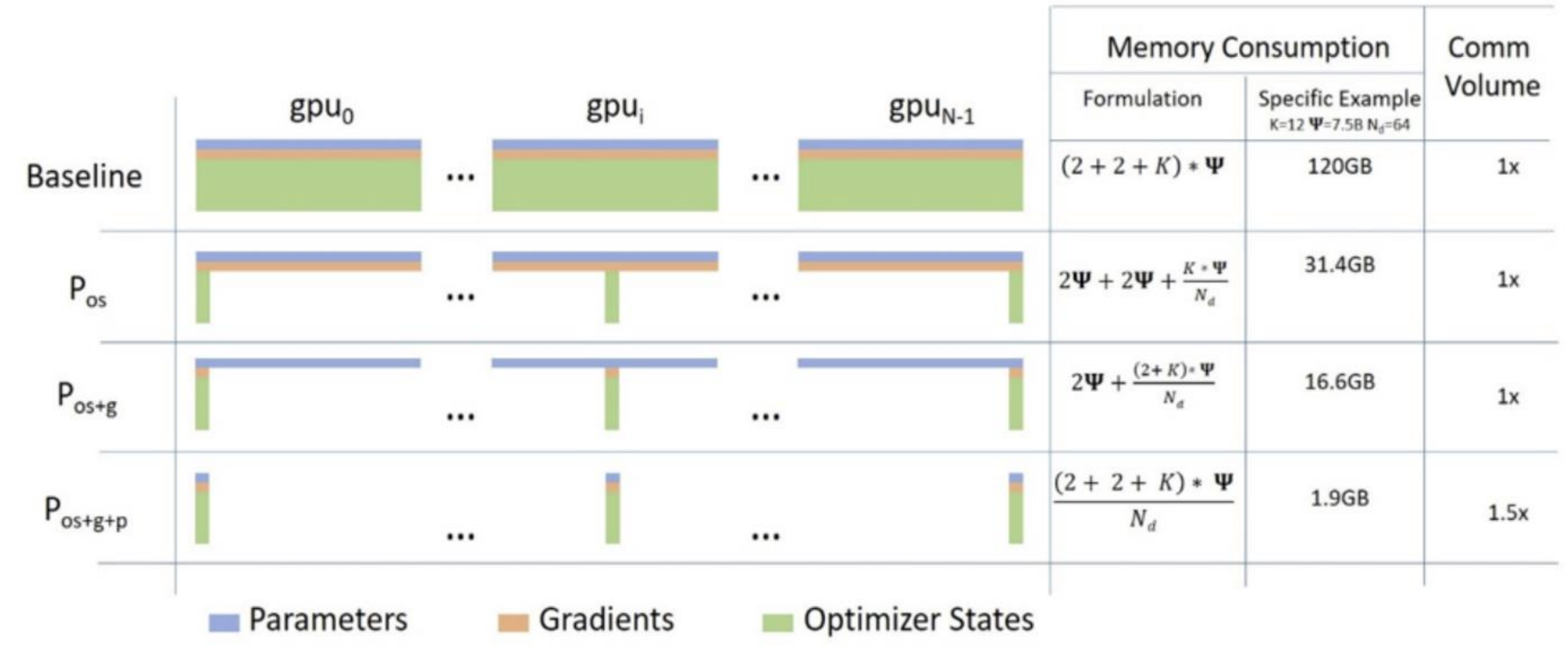


Figure 1: Memory savings and communication volume for the three stages of ZeRO compared with standard data parallel baseline. In the memory consumption formula, Ψ refers to the number of parameters in a model and K is the optimizer specific constant term. As a specific example, we show the memory consumption for a 7.5B parameter model using Adam optimizer where K=12 on 64 GPUs. We also show the communication volume of ZeRO relative to the baseline.

LLMのパラメータは増加傾向

新しい風を吹き込むTransformerモデル

- Llama2, Mistral, GPT等代表的なLLMのActive Parameter数とそのロードに必要なメモリサイズは以下の通り
  - 注: 以下はあくまでParameterのロードに必要なメモリのみの数値

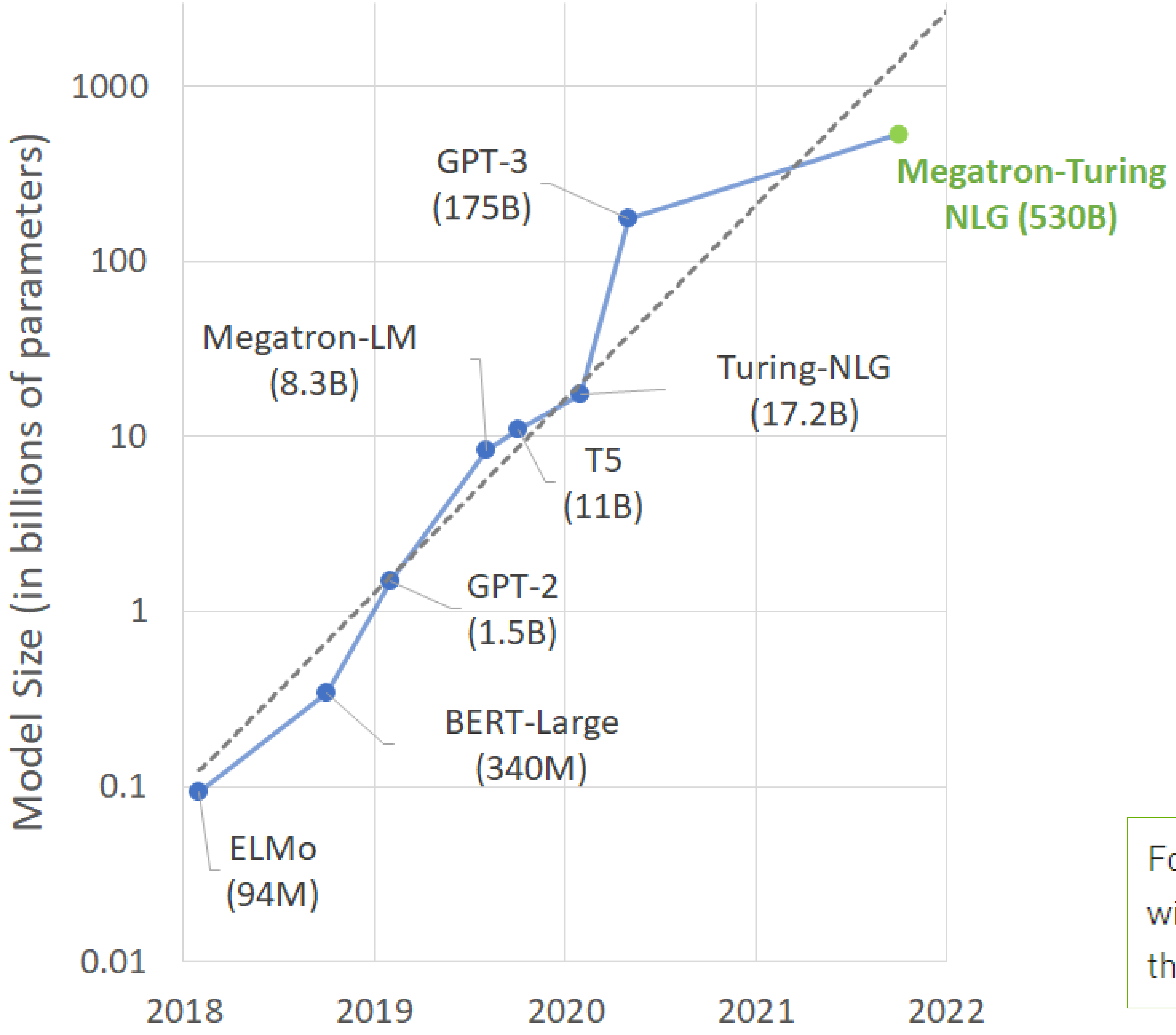
Model	Active Parameter	FP32	FP16/BF16
Llama 2 7B	7B	28GB	14GB
Llama 2 13B	13B	52GB	26GB
Llama 2 70B	70B	280GB	140GB
Mistral 7B	7B	28GB	14GB
Mixtral 8x7B	13B	188GB	94GB
GPT3 175B	175B	700GB	350GB

代表的なLLMの事前学習

モデル	発表時期	モデルサイズ(B)	事前学習 データサイズ	ハードウェア	学習時間
OPT (Meta)	May-2022	175	180B Token	992 80G A100	-
BLOOM (BigScience)	Nov-2022	176	366B Token	384 80G A100	105日
PLaMo (Preferred Networks)	Sep-2023	13	1.4T Token	480 40G A100	約30日
LLaMA (Meta)	Feb-2023	65	1.4T Token	2048 80G A100	21日
Llama 2 (Meta)	Jul-2023	70	2T Token	??? 80GB A100	1.72 M GPU hours (1×A100 GPU) 2048で35日ほど
Llama 3 (Meta)	Mar-2024	70	15T Token	24000 80GB H100	-
MT-NLG (MS/NVIDIA)	Jan-2022	530	270B Token	4480 80G A100	-

モデルがGPUに載らない？

分散学習の必要性



MT-NLGは530Bパラメータ

- モデルをすべてメモリにロードするだけで単純計算で 1,060 GB (in BF16)
- 8 x A100 (80GB) サーバが、2台（16 GPU）必要
- ワーキングメモリも当然必要

Q: 実際どうやって扱っているのか？

A: 4,480 x A100 GPUs (560 nodes)のクラスタ上で、一つのモデルを280 GPUs（8-way tensor parallel & 35-way pipeline parallel=35 nodes）に分割（Model Parallelism）して、それを16個複製（Data Parallelism）している

For example, for the 530 billion model, each model replica spans 280 NVIDIA A100 GPUs, with 8-way tensor-slicing within a node and 35-way pipeline parallelism across nodes. We then use data parallelism from DeepSpeed to scale out further to thousands of GPUs.

https://developer.nvidia.com/blog/using-deepspeed-and-megatron-to-train-megatron-turing-nlg-530b-the-worlds-largest-and-most-powerful-generative-language-model/

3D Parallelismによる分散学習 (NVIDIA Megatron)

[Efficient Large-Scale Language Model Training on GPU Clusters, Deepak Narayanan et al., 2021]

How to Run Pretraining on NeMo Framework?

```yaml
- _self_
- cluster: bcm  # Set to bcm for BCM and BCP clusters. Set to k8s for a k8s cluster.
- data_preparation: gpt3/download_gpt3_pile
- quality_filtering: heuristic/english
- training: gpt3/5b
- conversion: gpt3/convert_gpt3
- fine_tuning: null
- peft: null
- prompt_learning: null
- adapter_learning: null
- ia3_learning: null
- evaluation: gpt3/evaluate_all
- export: gpt3/export_gpt3
- rlhf_rm: gpt3/2b_rm
- rlhf_ppo: gpt3/2b_ppo
- override hydra/job_logging: stdout
```

```yaml
stages:
  #  - data_preparation
  - training
  #- conversion
  #- prompt_learning
  #- adapter_learning
  #- ia3_learning
  #- evaluation
  #- export
```

```yaml
model:
  micro_batch_size: 4
  global_batch_size: 256
  tensor_model_parallel_size: 1
  pipeline_model_parallel_size: 1
  virtual_pipeline_model_parallel_size: null # interleaved pipeline
  resume_from_checkpoint: null # manually set the checkpoint file to load from
  # model architecture
  encoder_seq_length: 2048
  max_position_embeddings: 2048
  num_layers: 12
  hidden_size: 768
  ffn_hidden_size: ${multiply:4, ${.hidden_size}}  # Transformer FFN hidden size. 4 * hidden_size.
  num_attention_heads: 12
  init_method_std: 0.023  # Standard deviation of the zero mean normal distribution used for weight initialization.')
  hidden_dropout: 0.1  # Dropout probability for hidden state transformer.
  kv_channels: null  # Projection weights dimension in multi-head attention. Set to hidden_size // num_attention_heads if null
  apply_query_key_layer_scaling: True # scale Q * K^T by 1 / layer-number.
  layernorm_epsilon: 1e-5
  make_vocab_size_divisible_by: 128 # Pad the vocab size to be divisible by this value for computation efficiency.
  pre_process: True # add embedding
  post_process: True # add pooler
  persist_layer_norm: True # Use of persistent fused layer norm kernel.
  gradient_as_bucket_view: True # Allocate gradients in a contiguous bucket to save memory (less fragmentation and buffer memory)
```

```
python3 main.py
```

https://docs.nvidia.com/nemo-framework/user-guide/latest/playbooks/pretraining.html

NeMo のモデルカスタマイズツール群

大規模言語モデルをユースケースに合わせてカスタマイズする方法

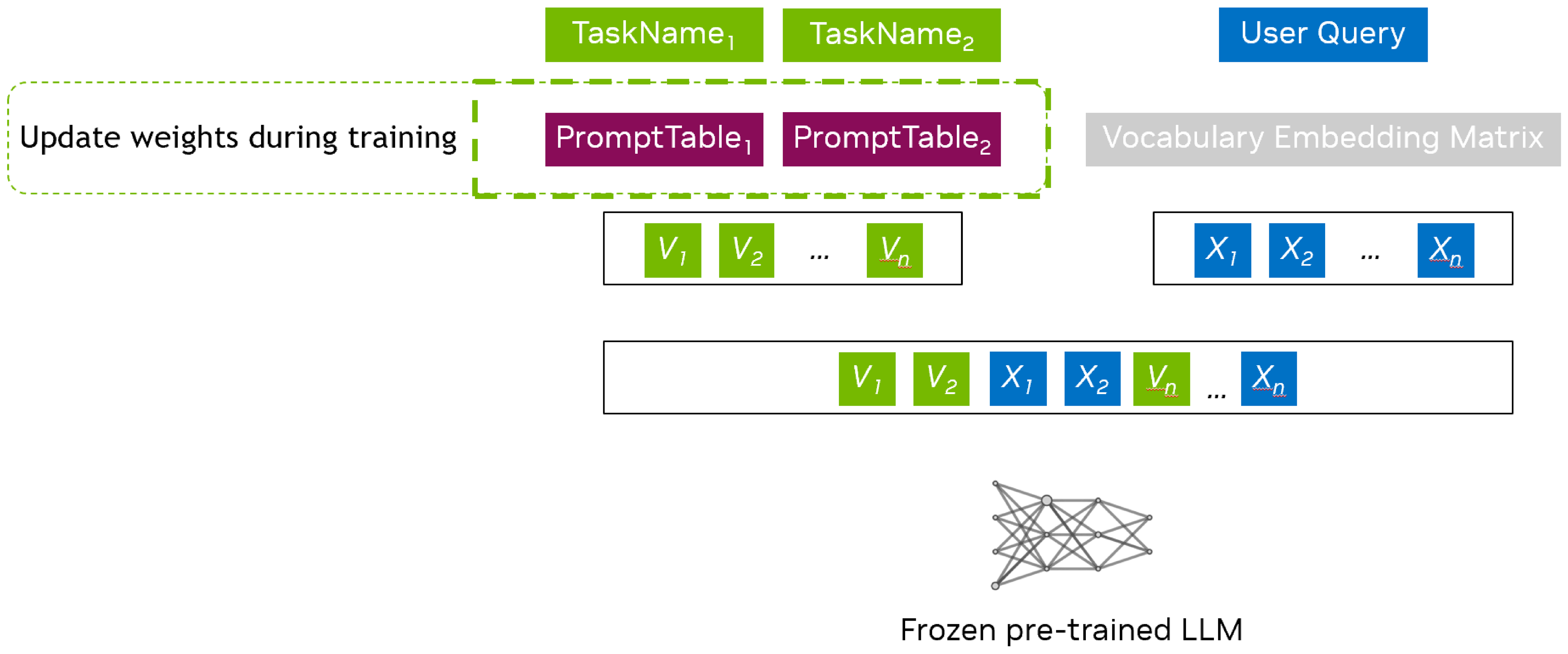
	PROMPT ENGINEERING	PROMPT LEARNING	PARAMETER EFFICIENT FINE-TUNING	FINE TUNING
技術	· Few-shot learning · Chain-of-thought reasoning · System prompting	· Prompt tuning · P-tuning	· Adapters · LoRA · IA3	· SFT · RLHF (PPO), DPO, SPIN · SteerLM
利点	· 事前に訓練されたLLMを活用した良好な結果 · 最も低い投資額 · 最も少ない専門知識	· 事前に訓練されたLLMを活用したより良い結果 · 低投資 · 古いスキルを忘れない	· 事前に訓練されたLLMを活用した最高の結果 · 古いスキルを忘れない	· 事前に訓練されたLLMを活用した最高の結果 · すべてのモデルパラメータを変更
課題	· 事前に訓練されたLLMに、多くのスキルやドメイン固有のデータを追加することはできない。	· すべてのモデルパラメータを変更する包括的な能力が少ない	· 中程度の投資 · トレーニングに時間がかかる · より多くの専門知識が必要	· 古いスキルを忘れる可能性 · 多額の投資 · 専門知識が最も必要

Prompt Learning

Prompt Tuning vs P-Tuning

Prompt Tuning

Embeddingのみ更新可能な特殊トークンのプロンプトを修正 。

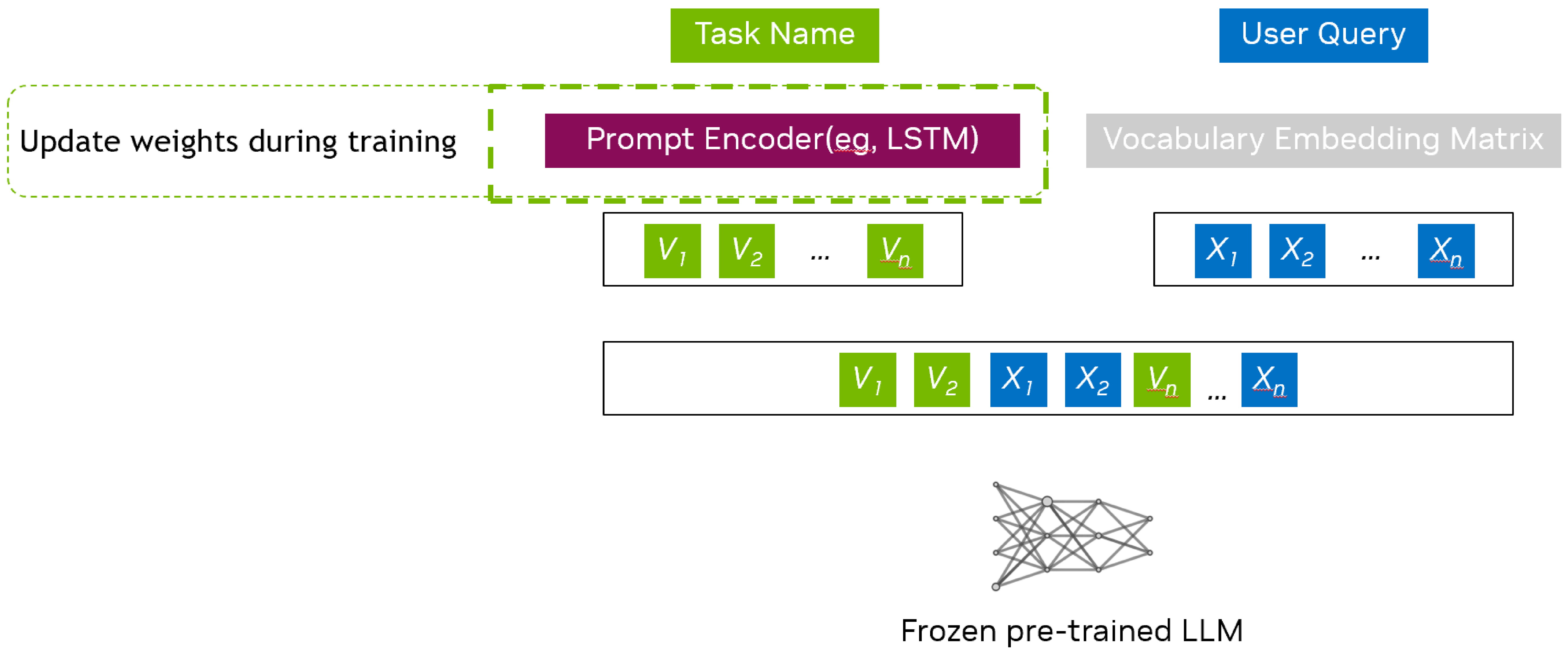


ファインチューニングするパラメータがほとんどなくなります。

ターゲットのタスクに適応する能力は限られていますが、ハードウェア リソースにかかるコストは低くなります。

P-Tuning

小規模な LSTM (Long Short-Term Memory) モデルを使用して、トークンの固定プロンプトの埋め込みを予測します。



さらに多くのパラメーターをチューニングする必要があります。

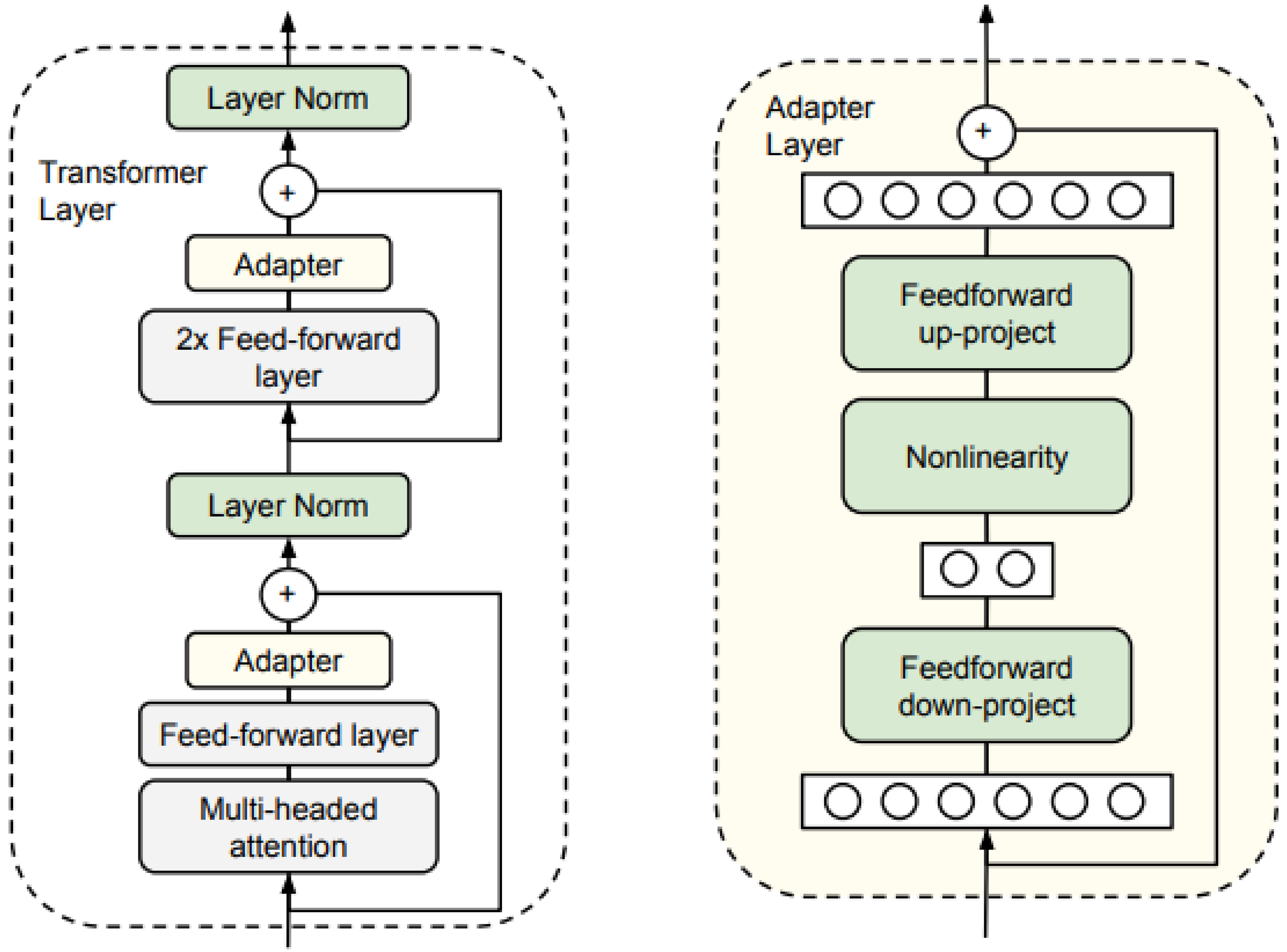
精度は向上しますが、ハードウェア リソースは増加します。

アダプターベースの手法

Adapter, LoRA, IA3

Adapter

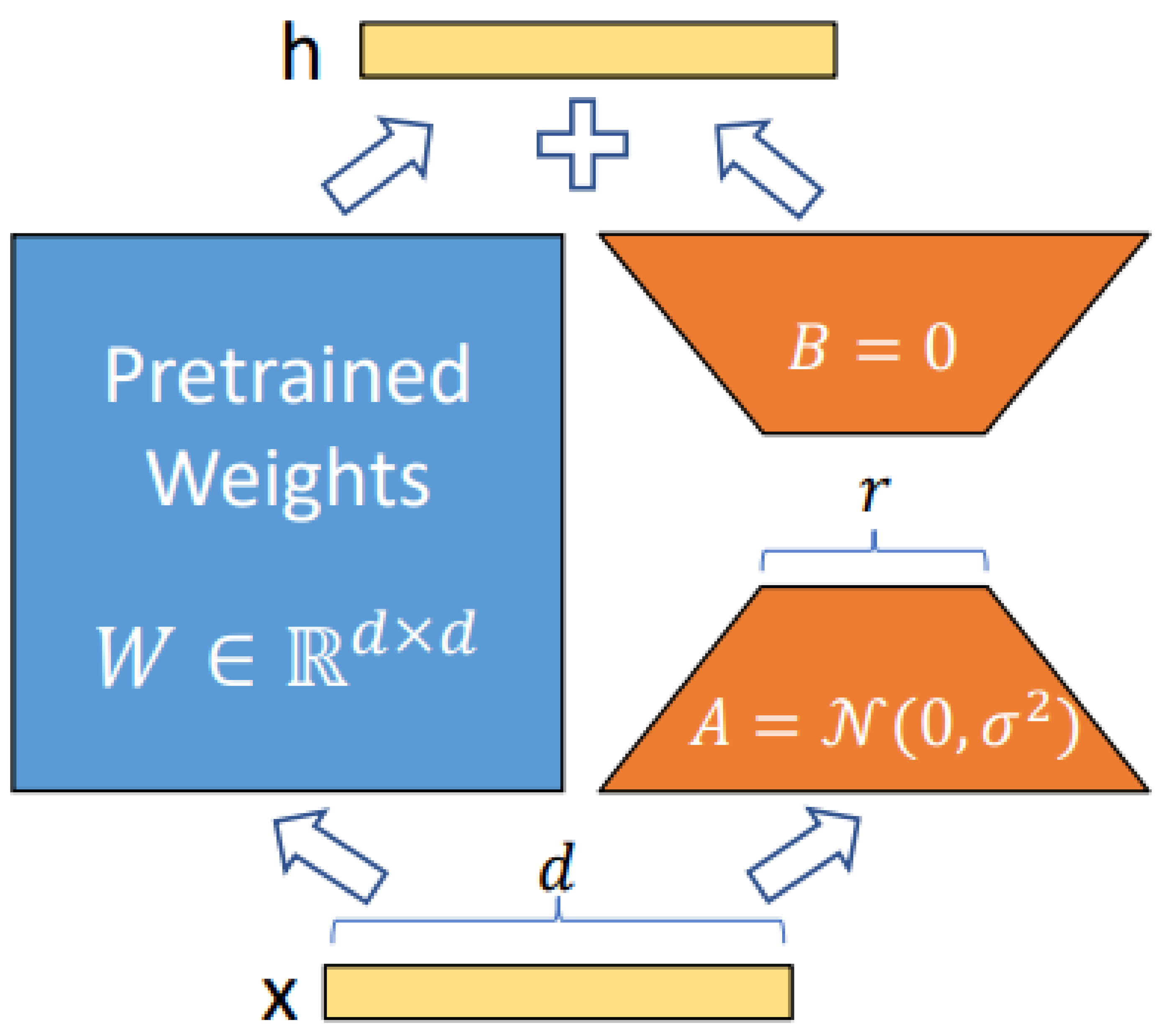
各Transformer層に挿入し、
Adapterの重みだけを更新



https://arxiv.org/abs/1902.00751

低ランク適応 (LoRA)

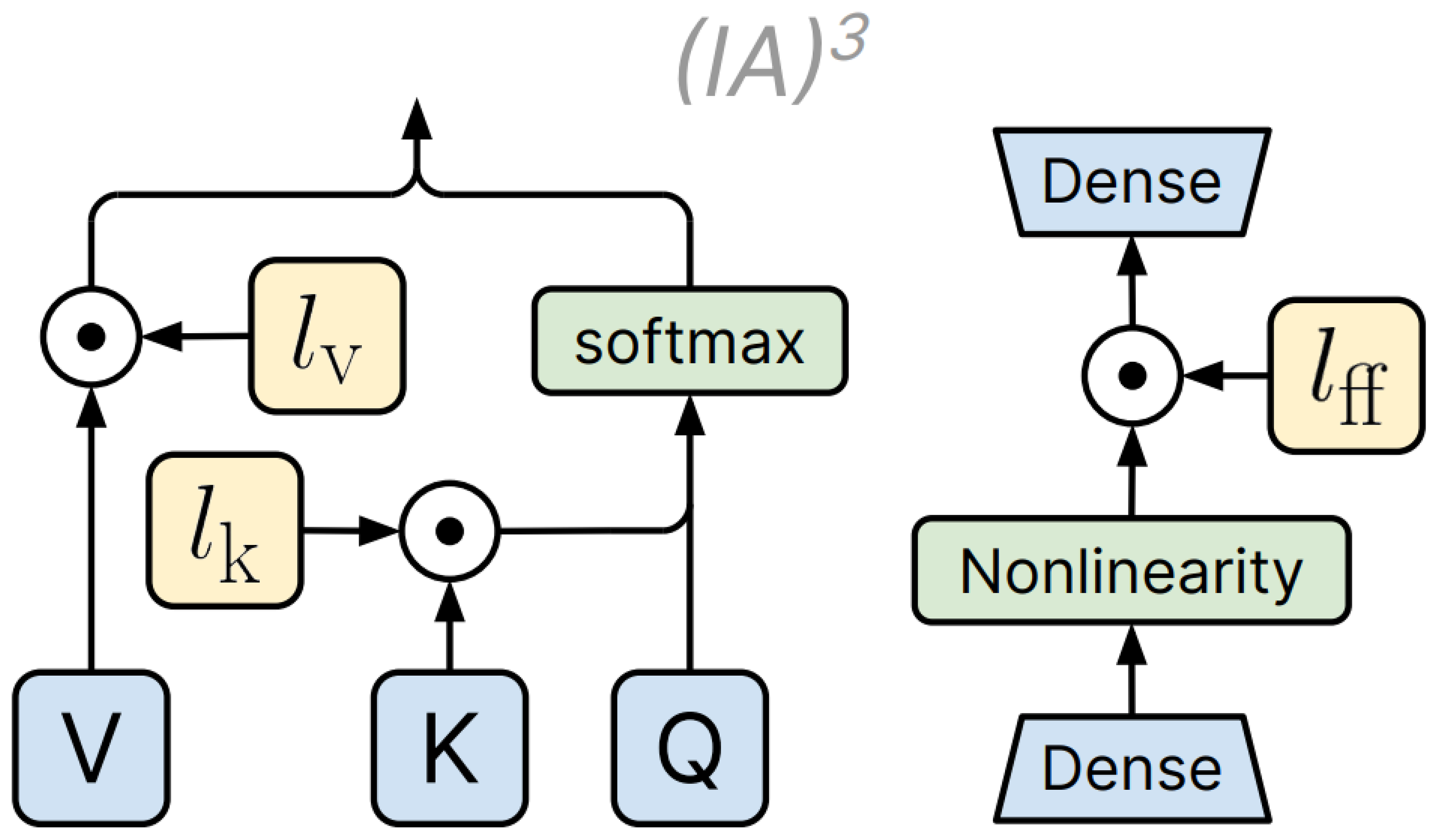
高密度層のランク分解行列を最適化



https://arxiv.org/abs/2106.09685

IA3

Adapterと似ているが、Key, Value, または FFN
内のレイヤ をスケーリングするベクトル



https://arxiv.org/abs/2205.05638

How to Run PEFT training on NeMo Framework?

```
MODEL="YOUR PATH TO llama2-7b.nemo"
TRAIN_DS="[YOUR PATH TO pubmedqa/pubmedqa_train.jsonl]"
VALID_DS="[YOUR PATH TO pubmedqa/pubmedqa_val.jsonl]"
TEST_DS="[YOUR PATH TO pubmedqa/pubmedqa_test.jsonl]"
TEST_NAMES="[pubmedqa]"
SCHEME="lora"
```

```
TRAIN_DS="[/path/to/dataset_1.jsonl,/path/to/dataset_2.jsonl]"
CONCAT_SAMPLING_PROBS="[0.3,0.7]"
```

```
torchrun --nproc_per_node=8 \
/opt/NeMo/examples/nlp/language_modeling/tuning/megatron_gpt_peft_tuning.py \
    trainer.devices=8 \
    trainer.num_nodes=1 \
    trainer.precision=bf16 \
    trainer.val_check_interval=20 \
    trainer.max_steps=50 \
    model.megatron_amp_O2=False \
    ++model.mcore_gpt=True \
    model.tensor_model_parallel_size=${TP_SIZE} \
    model.pipeline_model_parallel_size=${PP_SIZE} \
    model.micro_batch_size=1 \
    model.global_batch_size=8 \
    model.restore_from_path=${MODEL} \
    model.data.train_ds.num_workers=0 \
    model.data.validation_ds.num_workers=0 \
    model.data.train_ds.file_names=${TRAIN_DS} \
    model.data.train_ds.concat_sampling_probabilities=[1.0] \
    model.data.validation_ds.file_names=${VALID_DS} \
    model.peft.peft_scheme=${SCHEME}
```

https://docs.nvidia.com/nemo-framework/user-guide/latest/playbooks/llama2peft.html

教師ありファインチューニング(SFT, Instruction-Tuning)

LLMのZero-shotパフォーマンスを向上させる学習手法

FINETUNED LANGUAGE MODELS ARE ZERO-SHOT LEARNERS

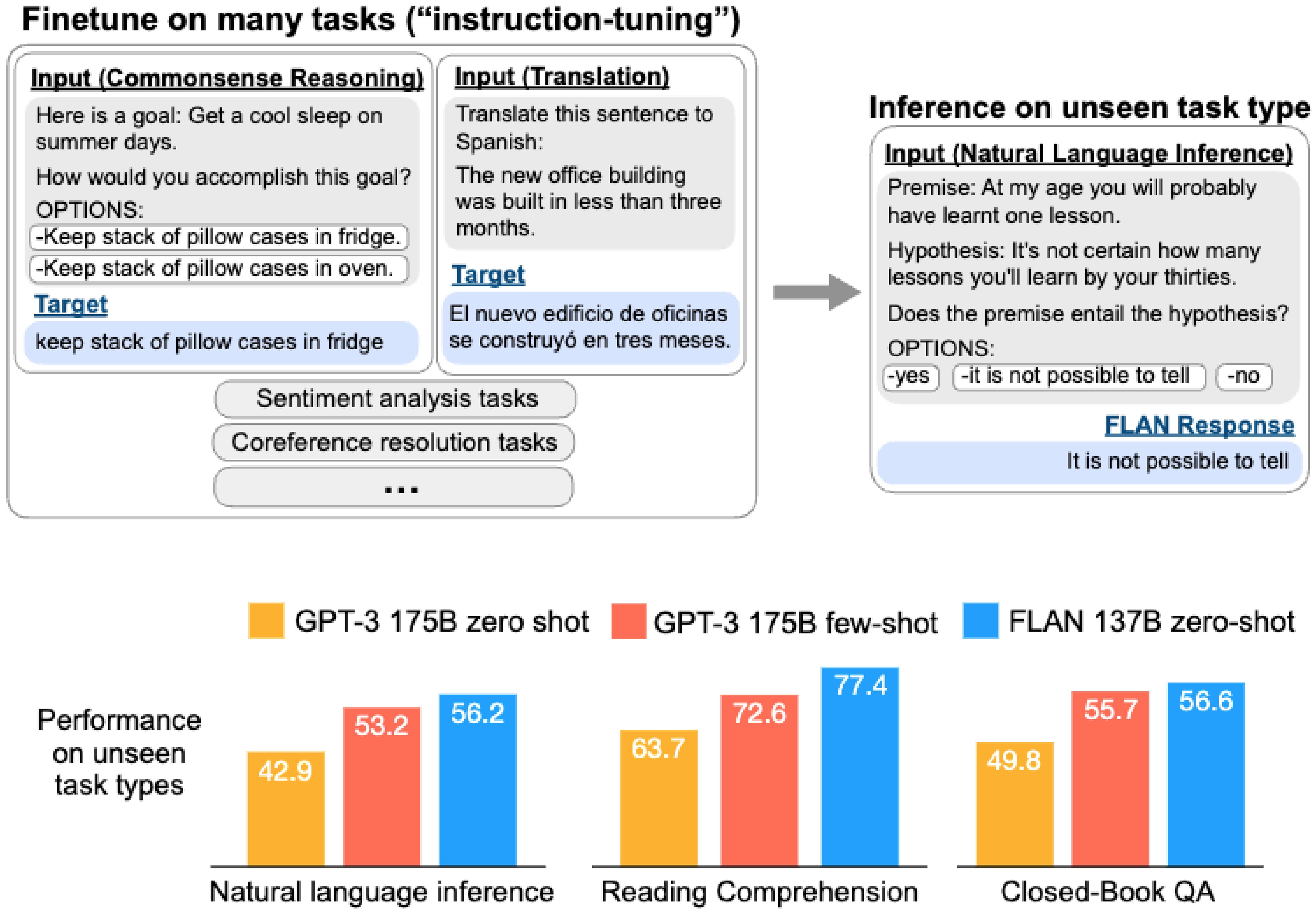
Jason Wei*, Maarten Bosma*, Vincent Y. Zhao*, Kelvin Guu*, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le

Google Research

ABSTRACT

This paper explores a simple method for improving the zero-shot learning abilities of language models. We show that *instruction tuning*—finetuning language models on a collection of datasets described via instructions—substantially improves zero-shot performance on unseen tasks.

We take a 137B parameter pretrained language model and instruction tune it on over 60 NLP datasets verbalized via natural language instruction templates. We evaluate this instruction-tuned model, which we call FLAN, on unseen task types. FLAN substantially improves the performance of its unmodified counterpart and surpasses zero-shot 175B GPT-3 on 20 of 25 datasets that we evaluate. FLAN even outperforms few-shot GPT-3 by a large margin on ANLI, RTE, BoolQ, AI2-ARC, OpenbookQA, and StoryCloze. Ablation studies reveal that number of finetuning datasets, model scale, and natural language instructions are key to the success of instruction tuning.



https://arxiv.org/pdf/2109.01652

How to Run SFT on NeMo Framework?

Step 2: SFT Training

Now that we have the data we will use NeMo-Aligner to do the supervised fine tuning.

Terminal Slurm

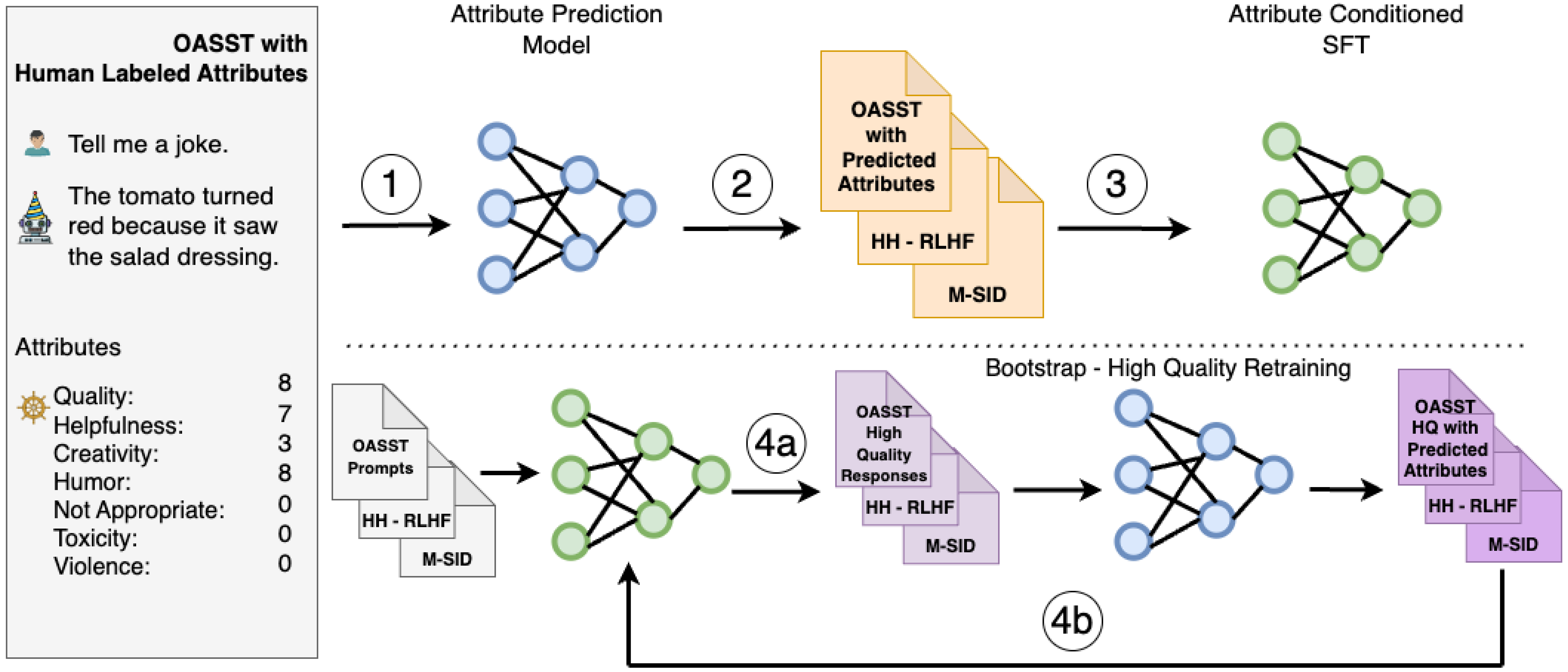
To run SFT on the terminal directly. Note that the working directory must be at the NeMo Aligner repo for this to work.

```
python examples/nlp/gpt/train_gpt_sft.py \
   trainer.precision=bf16 \
   trainer.num_nodes=1 \
   trainer.devices=8 \
   trainer.sft.max_steps=-1 \
   trainer.sft.limit_val_batches=40 \
   trainer.sft.val_check_interval=1000 \
   model.megatron_amp_O2=True \
   model.restore_from_path=/path/to/your/mcore_gpt.nemo \
   model.optim.lr=5e-6 \
   model.answer_only_loss=True \
   model.data.num_workers=0 \
   model.data.train_ds.micro_batch_size=1 \
   model.data.train_ds.global_batch_size=128 \
   model.data.train_ds.file_path=/path/to/databricks-dolly-15k-output.jsonl \
   model.data.validation_ds.micro_batch_size=1 \
   model.data.validation_ds.global_batch_size=128 \
   model.data.validation_ds.file_path=/path/to/databricks-dolly-15k-output.jsonl \
   exp_manager.create_wandb_logger=True \
   exp_manager.explicit_log_dir=/results \
   exp_manager.wandb_logger_kwargs.project=sft_run \
   exp_manager.wandb_logger_kwargs.name=dolly_sft_run \
   exp_manager.checkpoint_callback_params.save_nemo_on_train_end=True \
   exp_manager.resume_if_exists=True \
   exp_manager.resume_ignore_no_checkpoint=True \
   exp_manager.create_checkpoint_callback=True \
   exp_manager.checkpoint_callback_params.monitor=validation_loss
```

https://docs.nvidia.com/nemo-framework/user-guide/latest/modelalignment/sft.html#step-2-sft-training

属性スコアを利用したLLMの制御: SteerLM

推論中に LLM をカスタマイズする手法



1. 人間が注釈を付けたデータセットで予測モデルをトレーニングし、有用性、ユーモア、創造性などの任意の数の属性に関する応答品質を評価します。
2. 属性スコアを予測してさまざまなデータセットに注釈を付け、モデルで利用できるデータの多様性を豊かにします。
3. LLM をトレーニングする事でユーザーが認識する質や有用性などの属性の指定された組み合わせに基づいた応答を生成するようにファインチューニングします。
4. ブートストラップトレーニングは、最大の品質に基づいて多様な応答を生成することによるモデルサンプリングを通じて行われ、それらに対して微調整を行うことでさらなる適合性向上を図ります。

https://huggingface.co/nvidia/SteerLM-llama2-13B
https://arxiv.org/abs/2310.05344

How to Run SteerLM on NeMo Framework?

```
python /opt/NeMo-Aligner/examples/nlp/gpt/train_reward_model.py \
      trainer.num_nodes=32 \
      trainer.devices=8 \
      ++model.micro_batch_size=2 \
      ++model.global_batch_size=512 \
      ++model.data.data_impl=jsonl \
      pretrained_checkpoint.restore_from_path=/models/llama13b/llama13b.nemo \
      "model.data.data_prefix={train: ["data/merge_train_reg.jsonl"], validation: ["data/merge_val_reg.jsonl"], test: [
      exp_manager.explicit_log_dir=/results/reward_model_13b \
      trainer.rm.val_check_interval=10 \
      exp_manager.create_wandb_logger=True \
      exp_manager.wandb_logger_kwargs.project=steerlm \
      exp_manager.wandb_logger_kwargs.name=rm_training \
      trainer.rm.save_interval=10 \
      trainer.rm.max_steps=800 \
      ++model.tensor_model_parallel_size=4 \
      ++model.pipeline_model_parallel_size=1 \
      ++model.activations_checkpoint_granularity="selective" \
      ++model.activations_checkpoint_method="uniform" \
      model.global_batch_size=512 \
      model.optim.sched.constant_steps=0 \
      model.reward_model_type="regression" \
      model.regression.num_attributes=9
```

```
python examples/nlp/gpt/train_gpt_sft.py \
     trainer.num_nodes=32 \
     trainer.devices=8 \
     trainer.precision=bf16 \
     trainer.sft.limit_val_batches=40 \
     trainer.sft.max_epochs=1 \
     trainer.sft.max_steps=800 \
     trainer.sft.val_check_interval=800 \
     trainer.sft.save_interval=800 \
     model.megatron_amp_O2=True \
     model.restore_from_path=/models/llama70b \
     model.tensor_model_parallel_size=8 \
     model.pipeline_model_parallel_size=2 \
     model.optim.lr=6e-6 \
     model.optim.name=distributed_fused_adam \
     model.optim.weight_decay=0.01 \
     model.optim.sched.constant_steps=200 \
     model.optim.sched.warmup_steps=1 \
     model.optim.sched.min_lr=5e-6 \
     model.answer_only_loss=True \
     model.activations_checkpoint_granularity=selective \
     model.activations_checkpoint_method=uniform \
     model.data.chat=True \
     model.data.num_workers=0 \
     model.data.chat_prompt_tokens.system_turn_start=\'\<extra_id_0\>\' \
     model.data.chat_prompt_tokens.turn_start=\'\<extra_id_1\>\' \
     model.data.chat_prompt_tokens.label_start=\'\<extra_id_2\>\' \
     model.data.train_ds.max_seq_length=4096 \
     model.data.train_ds.micro_batch_size=1 \
     model.data.train_ds.global_batch_size=128 \
     model.data.train_ds.file_path=data/oasst/train_labeled_2ep.jsonl \
     model.data.train_ds.index_mapping_dir=/indexmap_dir \
     model.data.train_ds.add_eos=False \
     model.data.train_ds.hf_dataset=True \
     model.data.validation_ds.max_seq_length=4096 \
     model.data.validation_ds.file_path=data/oasst/val_labeled.jsonl \
     model.data.validation_ds.micro_batch_size=1 \
     model.data.validation_ds.global_batch_size=128 \
     model.data.validation_ds.index_mapping_dir=/indexmap_dir \
     model.data.validation_ds.add_eos=False \
     model.data.validation_ds.hf_dataset=True \
     exp_manager.create_wandb_logger=True \
     exp_manager.wandb_logger_kwargs.project=steerlm \
     exp_manager.wandb_logger_kwargs.name=acsft_training \
     exp_manager.explicit_log_dir=/results/acsft_70b \
     exp_manager.checkpoint_callback_params.save_nemo_on_train_end=True
```

https://docs.nvidia.com/nemo-framework/user-guide/latest/modelalignment/steerlm.html

Useful Link

- NVIDIA NeMo Framework
  - https://docs.nvidia.com/nemo-framework/user-guide/latest/overview.html
- NeMo
  - https://github.com/NVIDIA/NeMo
- NeMo-Aligner
  - https://github.com/NVIDIA/NeMo-Aligner/tree/main

その他の Tips

玄界上でGPUデバッグ

その他の Tips

- ここからは玄界上でGPUの稼働状況等々確認する方法等、いくつか有益(だと思われる)情報をいくつかご紹介しようと思います。

nvidia-smi

nvidia-smi を動かす

もっとも原始的なGPU稼働状況の確認方法

```
#!/bin/bash
#PJM -L rscgrp=b-batch
#PJM -L gpu=1
#PJM -j
#PJM -L elapse=0:03:00

source /etc/profile.d/modules.sh
source ./ddp-test/bin/activate

nvidia-smi   --query-gpu=timestamp,temperature.gpu,utilization.gpu,utilization.memory,memory.used,memory.free --format=csv -l 1 > nvidiasmi.log &\
   NVIDIA_SMI_PID=$!
torchrun --nnodes 1 --nproc_per_node 2 ./examples/distributed/ddp-tutorial-series/multigpu_torchrun.py 50 10

kill $NVIDIA_SMI_PID
```

Reference: /home/pj25001011/share/nvidia/genkai-example-nv/gpu-tips/01_nvidiasmi /run_ddp.sh

nvidia-smi を動かす

もっとも原始的なGPU稼働状況の確認方法

```
#!/bin/bash
#PJM -L rscgrp=b-batch
#PJM -L gpu=1
#PJM -j
#PJM -L elapse=0:03:00

source /etc/profile.d/modules.sh
source ./ddp-test/bin/activate

nvidia-smi   --query-gpu=timestamp,temperature.gpu,utilization.gpu,utilization.memory,memory.used,memory.free --format=csv -l 1 > nvidiasmi.log &\
   NVIDIA_SMI_PID=$!
torchrun --nnodes 1 --nproc_per_node 2 ./examples/distributed/ddp-tutorial-series/multigpu_torchrun.py 50 10

kill $NVIDIA_SMI_PID
```

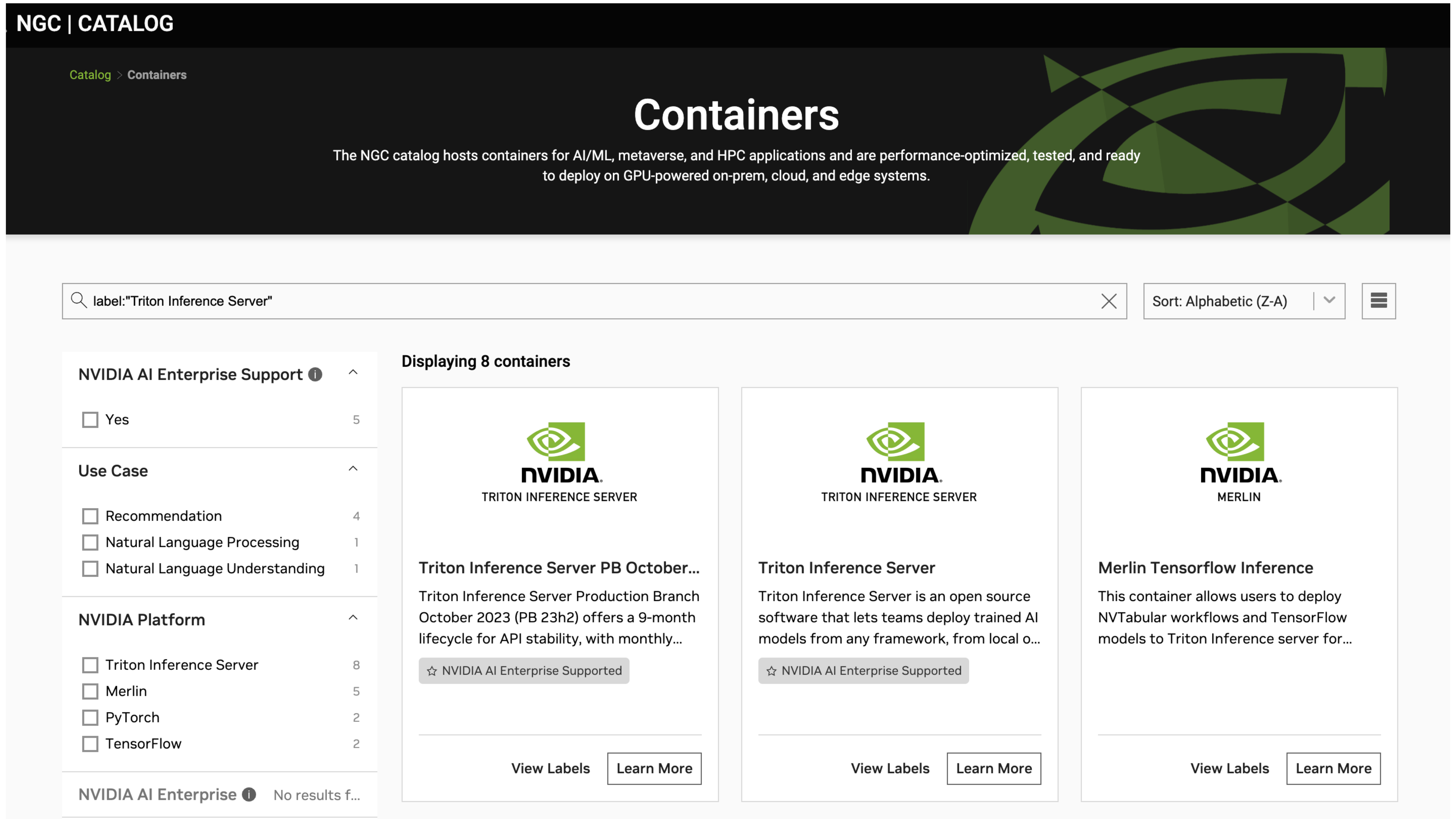
nvidia-smi --help-query-gpu
で取得可能なクエリが確認できます

Reference: /home/pj25001011/share/nvidia/genkai-example-nv/gpu-tips/01_nvidiasmi /run_ddp.sh

NGC container

NGC Catalog

NVIDIAコンテナでより高速にアプリケーションを構築



開発期間の短縮

Save OpEx and eliminate developers' time and effort on building containers with right dependencies

Download pre-built containers for a variety of use-cases

パフォーマンスの最適化

Scalable

Updated Monthly

Better performance on the same system

どこでもデプロイ

Docker | cri-o | containerd | Singularity

Bare metal, VMs, Kubernetes

Multi-cloud, on-prem, hybrid, edge

エンタープライズレディ

Production branches

Regular releases of security patches

Enterprise support with SLAs

NGC Catalog

NVIDIA GPUに最適化されたコンテナ (PyTorch24.04コンテナの例)

NVIDIA Docs Hub › NVIDIA Optimized Frameworks › NVIDIA Optimized Frameworks › PyTorch Release 24.04

PyTorch Release 24.04 (**PDF**)

The NVIDIA container image for PyTorch, release 24.04 is available on **NGC**.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and in
image. The container also includes the following:

- **Ubuntu 22.04** including **Python 3.10**
- **NVIDIA CUDA 12.4**
- **NVIDIA cuBLAS 12.4.5.8**
- **NVIDIA cuDNN 9.1.0.70**
- **NVIDIA NCCL 2.21.5**
- NVIDIA RAPIDS™ 24.02
- **rdma-core 39.0**
- NVIDIA HPC-X 2.18
- **OpenMPI 4.1.4+**
- GDRCopy 2.3
- TensorBoard 2.9.0
- **Nsight Compute 2024.1.0.13**
- **Nsight Systems 2024.2.1.38**
- **NVIDIA TensorRT™ 8.6.3**
- **Torch-TensorRT 2.3.0a0**
- **NVIDIA DALI® 1.36**
- **nvImageCodec 0.2.0.7**
- **MAGMA 2.6.2**
- **JupyterLab 2.3.2** including **Jupyter-TensorBoard**
- **TransformerEngine** 1.5
- PyTorch quantization wheel 2.1.2

Containers › PyTorch

PyTorch

Get Co

PyTorch | Accelerated with NVIDIA

Features

NVIDIA AI Enterprise Supported

Signed Images

Description

PyTorch is a GPU accelerated tensor computational framework. Functionality can be extended with common Python libraries such as NumPy and SciPy. Automatic differentiation is done with a tape-based system at the functional and neural network layer levels.

Publisher

Facebook

Latest Tag

24.05-py3

Modified

June 1, 2024

Compressed Size

8.57 GB

Overview | Tags | Layers | Security Scanning | Related Collections

Search tags...

A public key is required to validate the signed images below. View all public keys maintained by NVIDIA.

Tag	Date	Size	Architectures	Pull
24.05-py3	05/25/2024 3:45 AM	8.57 GB	2 Architectures	nvcr.io/nvidia/pytorch:24.05-py3
24.05-py3-igpu	05/25/2024 3:45 AM	5.36 GB	1 Architecture	nvcr.io/nvidia/pytorch:24.05-py3-igpu
24.04-py3	04/30/2024 8:42 AM	8.68 GB	2 Architectures	nvcr.io/nvidia/pytorch:24.04-py3
24.04-py3-igpu	04/30/2024 8:41 AM	5.42 GB	1 Architecture	nvcr.io/nvidia/pytorch:24.04-py3-igpu
24.03-py3 Signed	03/27/2024 8:59 AM	8.59 GB	2 Architectures	nvcr.io/nvidia/pytorch:24.03-py3

https://docs.nvidia.com/deeplearning/frameworks/pytorch-release-notes/rel-24-04.html#rel-24-04

https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pytorch/tags

NGC Catalog

NVIDIA GPUに最適化されたコンテナ (PyTorch24.04コンテナの例)

NVIDIA Docs Hub › NVIDIA Optimized Frameworks › NVIDIA Optimized Frameworks › PyTorch Release 24.04

PyTorch Release 24.04 (**PDF**)

The NVIDIA container image for PyTorch, release 24.04 is available on **NGC**.

Contents of the PyTorch container

This container image contains the complete source of the version of PyTorch in `/opt/pytorch`. It is prebuilt and in
image. The container also includes the following:

- **Ubuntu 22.04** including **Python 3.10**
- **NVIDIA CUDA 12.4**
- **NVIDIA cuBLAS 12.4.5.8**
- **NVIDIA cuDNN 9.1.0.70**
- **NVIDIA NCCL 2.21.5**
- NVIDIA RAPIDS™ 24.02
- **rdma-core 39.0**
- NVIDIA HPC-X 2.18
- **OpenMPI 4.1.4+**
- GDRCopy 2.3
- TensorBoard 2.9.0
- **Nsight Compute 2024.1.0.13**
- **Nsight Systems 2024.2.1.38**
- **NVIDIA TensorRT™ 8.6.3**
- **Torch-TensorRT 2.3.0a0**
- **NVIDIA DALI® 1.36**
- **nvImageCodec 0.2.0.7**
- **MAGMA 2.6.2**
- **JupyterLab 2.3.2** including **Jupyter-TensorBoard**
- **TransformerEngine** 1.5
- PyTorch quantization wheel 2.1.2

Containers › PyTorch

PyTorch

Get Co

PyTorch | Accelerated with NVIDIA

Features

NVIDIA AI Enterprise Supported

Signed Images

Description

PyTorch is a GPU accelerated tensor computational framework. Functionality can be extended with common Python libraries such as NumPy and SciPy. Automatic differentiation is done with a tape-based system at the functional and neural network layer levels.

Publisher

Facebook

Latest Tag

24.05-py3

Modified

June 1, 2024

Compressed Size

8.57 GB

Overview | Tags | Layers | Security Scanning | Related Collections

Search tags...

A public key is required to validate the signed images below. View all public keys maintained by NVIDIA.

Tag	Date	Size	Architectures	Pull
24.05-py3	05/25/2024 3:45 AM	8.57 GB	2 Architectures	nvcr.io/nvidia/pytorch:24.05-py3
24.05-py3-igpu	05/25/2024 3:45 AM	5.36 GB	1 Architecture	nvcr.io/nvidia/pytorch:24.05-py3-igpu
24.04-py3	04/30/2024 8:42 AM	8.68 GB	2 Architectures	nvcr.io/nvidia/pytorch:24.04-py3
24.04-py3-igpu	04/30/2024 8:41 AM	5.42 GB	1 Architecture	nvcr.io/nvidia/pytorch:24.04-py3-igpu
24.03-py3 Signed	03/27/2024 8:59 AM	8.59 GB	2 Architectures	nvcr.io/nvidia/pytorch:24.03-py3

https://docs.nvidia.com/deeplearning/frameworks/pytorch-release-notes/rel-24-04.html#rel-24-04

https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pytorch/tags

NVIDIA

玄界 はコンテナの実行が可能です

NGCコンテナを動かすと再現性等々便利な場合も…

```
#!/bin/bash
#PJM -L rscgrp=b-batch
#PJM -L gpu=2
#PJM -j
#PJM -L elapse=0:10:00
#PJM -L jobenv=singularity

source /etc/profile.d/modules.sh
module load singularity-ce/4.1.3

export WORKDIR=/home/${GROUP}/share/containers

TORCH_CONTAINER_IMAGE=${WORKDIR}/pytorch_24.04-py3.sif
echo $TORCH_CONTAINER_IMAGE
singularity run --nv $TORCH_CONTAINER_IMAGE bash -c "torchrun --nnodes 1 --nproc_per_node 2 ./examples/distributed/ddp-tutorial-series/multigpu_torchrun.py 50 10"
```

Reference: /home/pj25001011/share/nvidia/genkai-example-nv/gpu-tips/02_container /run_ddp.sh

Nsight Systems

Nsight Systems

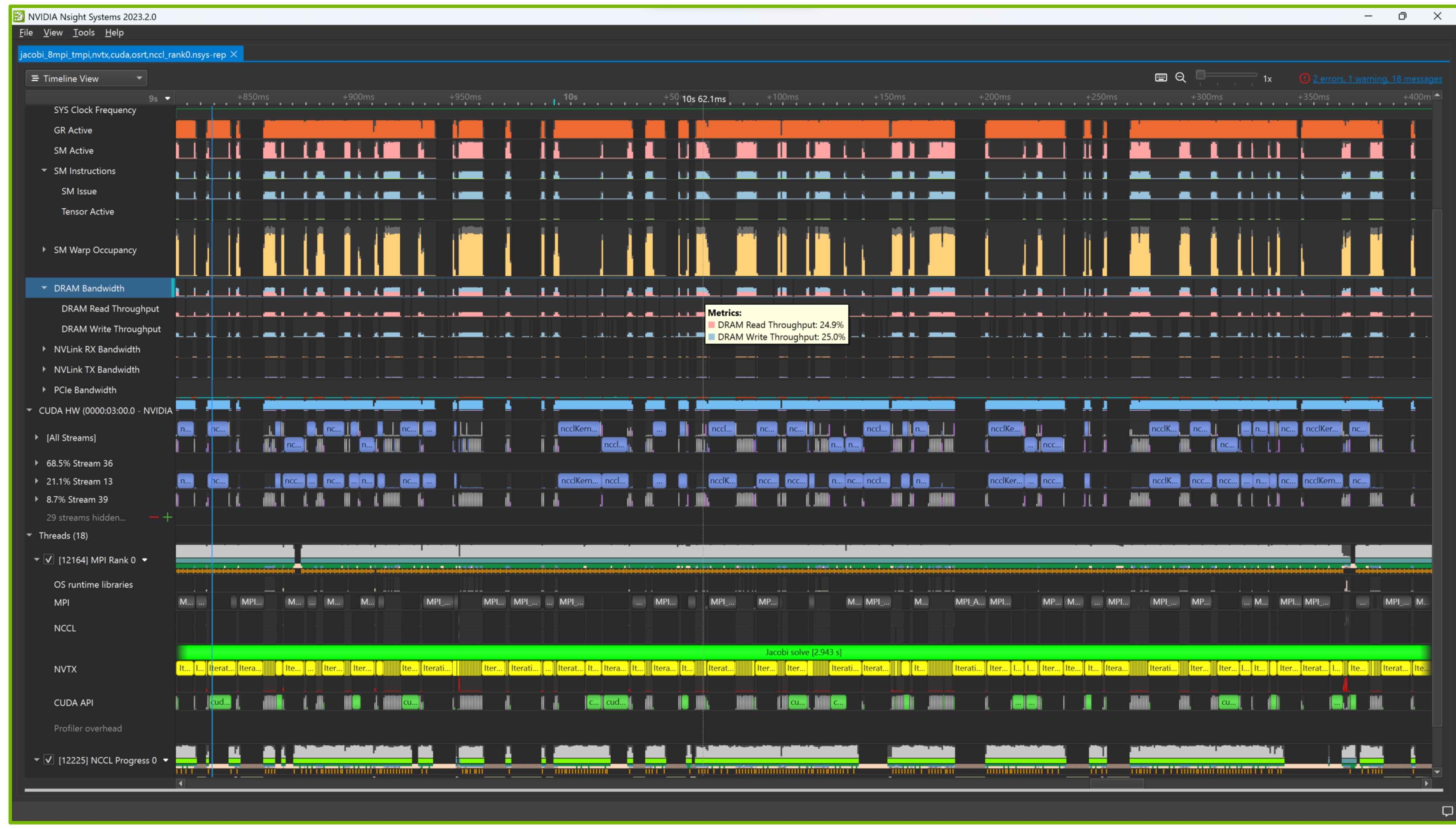
System Profiler

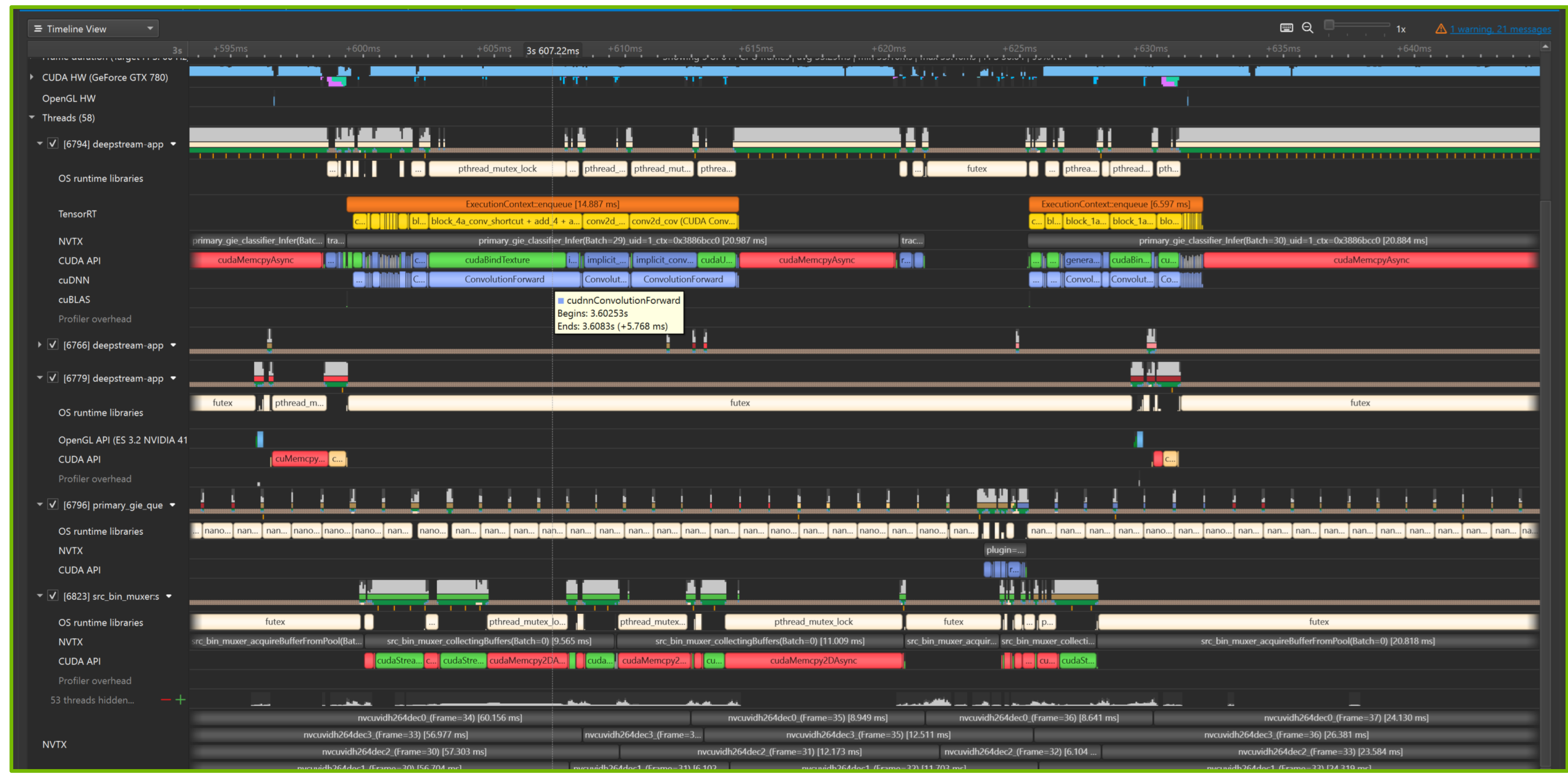
Key Features:

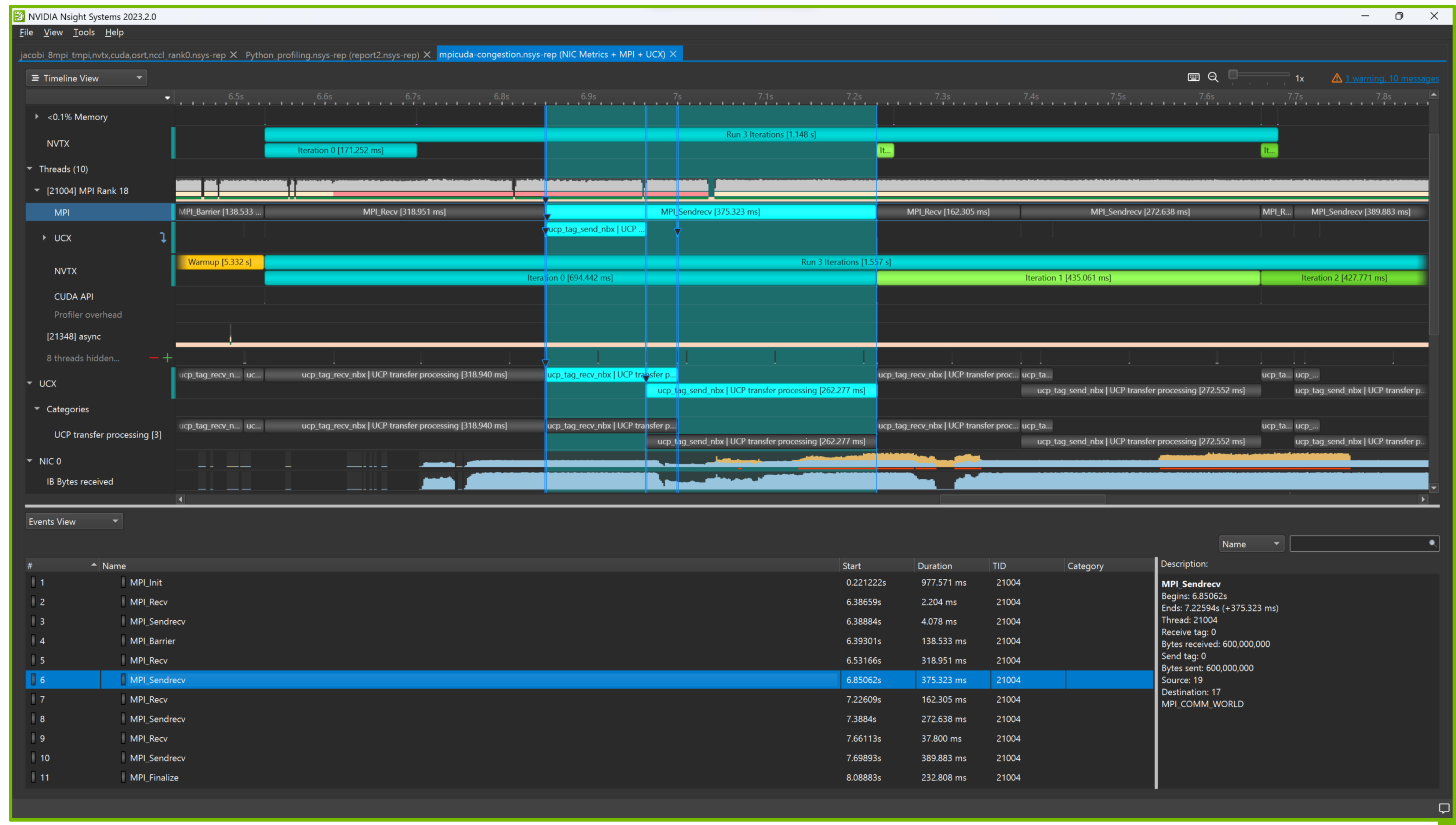
- System-wide application algorithm tuning
  - Multi-process tree support
- Locate optimization opportunities
  - Visualize millions of events on a very fast GUI timeline
  - Identify gaps of unused CPU and GPU time
- Balance your workload across multiple CPUs and GPUs
  - CPU algorithms, utilization and thread state
    GPU streams, kernels, memory transfers, etc
- Command Line, Standalone, IDE Integration

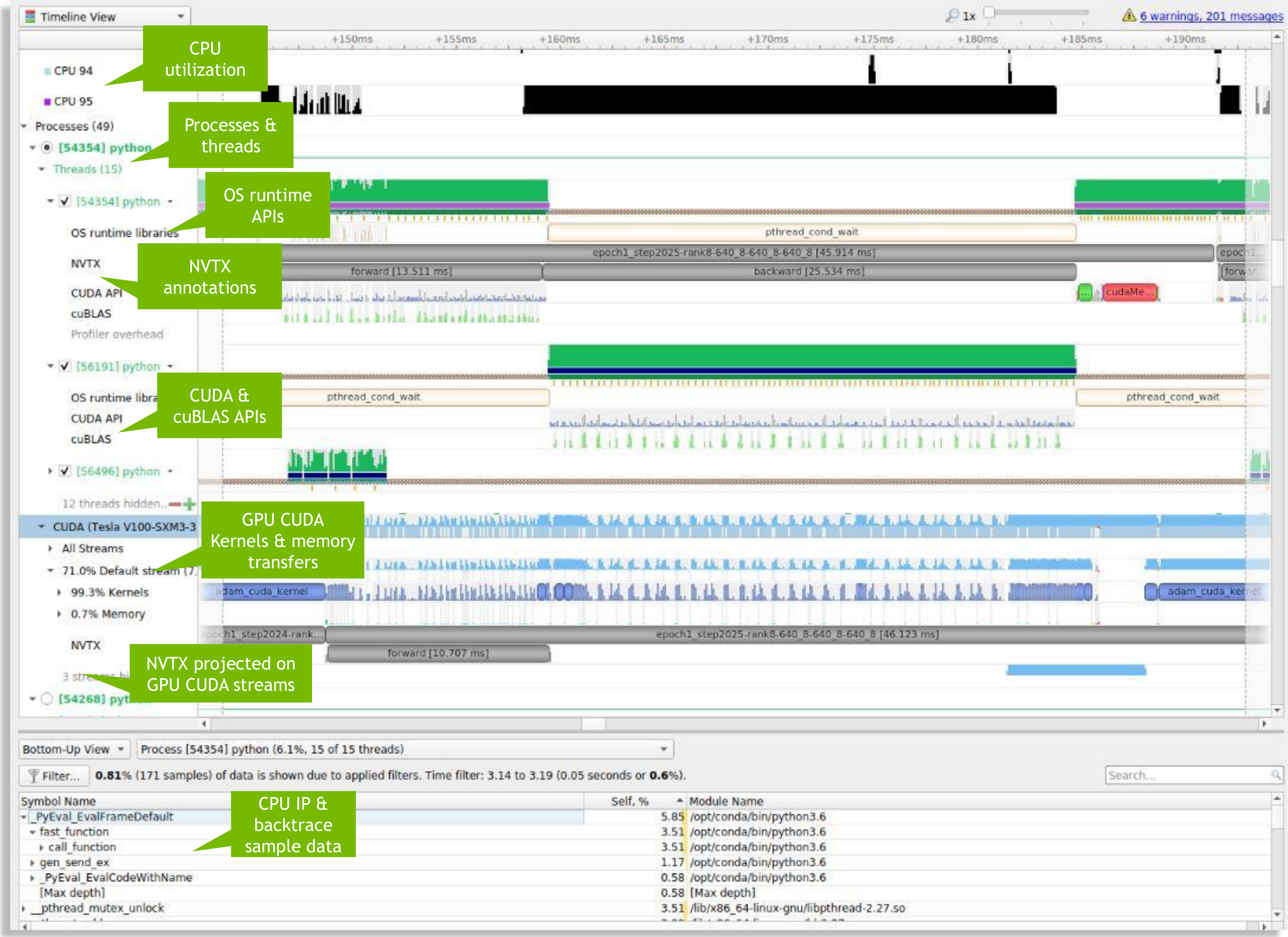
- OS: Linux (x86, Power, ARM Server, Tegra), Windows, macOS X (host)
- GPUs: Pascal+

- Docs/product: https://developer.nvidia.com/nsight-systems









Additionally

Trace:

- TensorRT
- Direct3D11,12,DXR
- Vulkan
- OpenGL
- OpenACC
- MPI
- OpenMP
- Ftrace
- ETW
- WDDM
- GPU Context Switch

Export:

- SQLite
- HDF5
- JSON

Architectures:

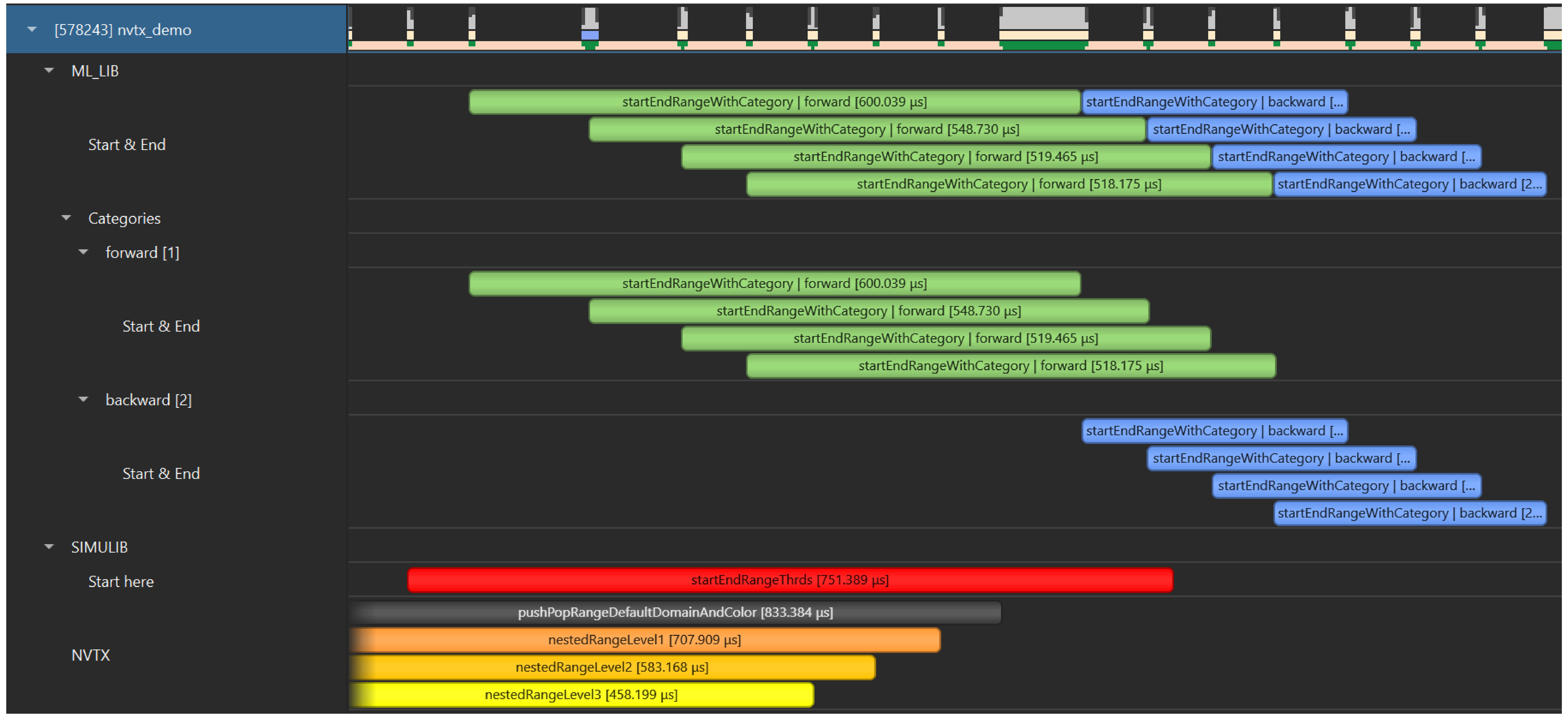
- X86_64
- Power
- Arm SBSA
- Tegra

NVIDIA Tools Extensions (NVTX)

C, C++, & Python

Decorate application source code with annotations (markers, ranges, nested ranges, …)
Help visualize execution with debugging, tracing and profiling tools
https://github.com/NVIDIA/NVTX/tree/release-v3/

```
#include <nvtx3/nvToolsExt.h>
```



Marker

```
nvtxMark("This is a marker");
```

Push-Pop range

```
nvtxRangePush("This is a push/pop range");
// Do something interesting in the range
nvtxRangePop();
```

Start-End range

```
// Somewhere in the code:
nvtxRangeHandle_t handle = nvtxRangeStart("This is a start/end range");
// Somewhere else in the code, not necessarily same thread as Start call:
nvtxRangeEnd(handle);
```

Advanced concepts like colors, domain, category…
API references https://nvidia.github.io/NVTX/doxygen/index.html and https://nvidia.github.io/NVTX/doxygen-cpp/index.html

NVIDIA Tools Extensions (NVTX)

https://github.com/NVIDIA/NVTX/blob/release-v3/python/docs/index.rst

アプリケーションのソースコードをアノテーション、プロファイリングツールによる実行の可視化。C, C++, & Pythonに対応

https://github.com/NVIDIA/NVTX/tree/release-v3/

pip install nvtx で簡単インストール(pythonの場合)

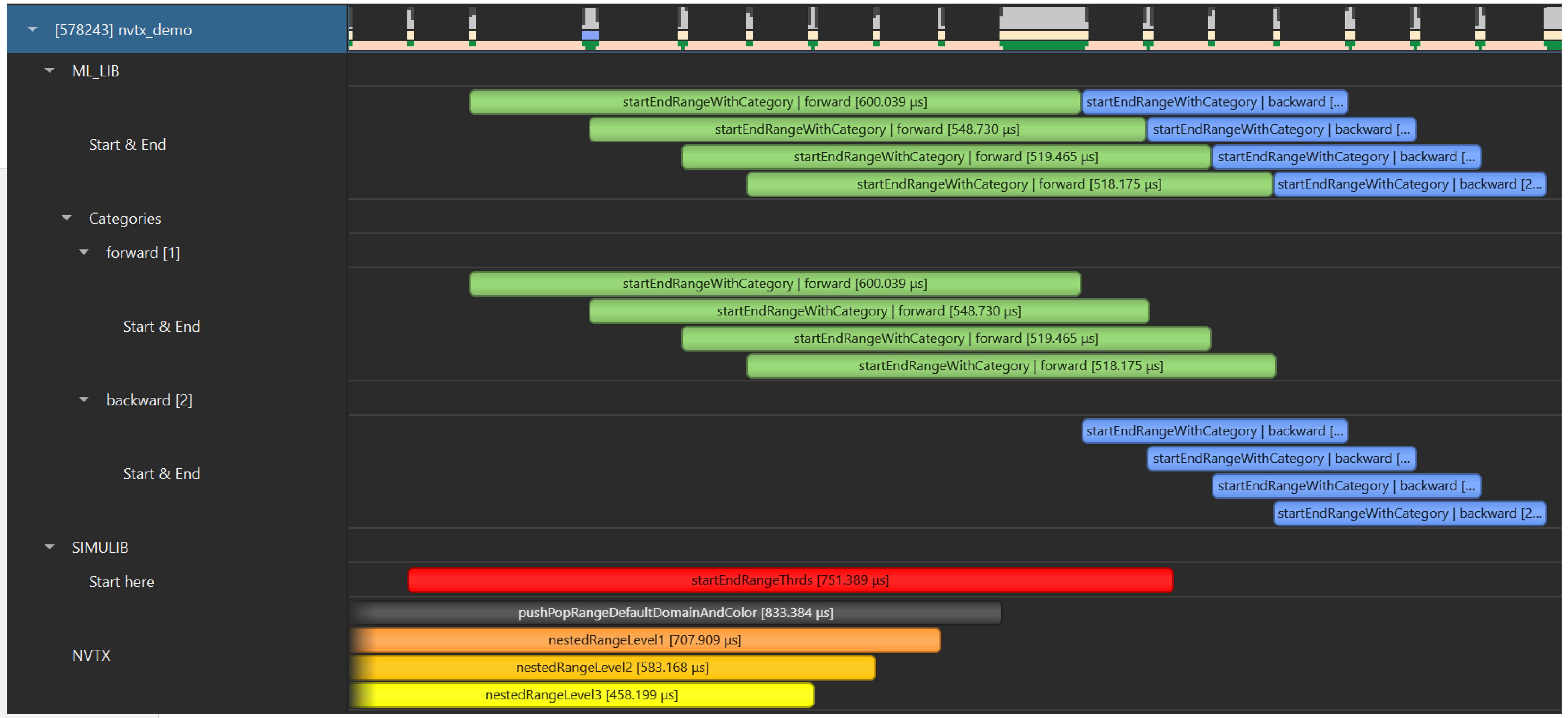
```
# example_lib.py

import time
import nvtx


def sleep_for(i):
    time.sleep(i)

@nvtx.annotate()
def my_func():
    time.sleep(1)

with nvtx.annotate("for_loop", color="green"):
    for i in range(5):
        sleep_for(i)
        my_func()
```



NVIDIA SDKs and NVTX

A Complete Ecosystem

nsys コマンドを利用

Nsight Systems でプロファイリングするには

プログラム全体のプロファイリング結果を result.qdrep に出力する場合

```
nsys profile –f true -o result -t cublas,cuda,cudnn,nvtx,openacc,osrt \
    python main.py ...
```

※ 長時間実行すると、非常に巨大なプロファイリング結果ファイルが生成されます！！

GPU Profiling Overview (基礎編)

https://docs.nvidia.com/nsight-systems/UserGuide/index.html#cli-profiling

Nsight Systems を使ってみる

NVIDIA 純正 GPU プロファイラ

```
#!/bin/bash
#PJM -L rscgrp=b-batch
#PJM -L gpu=2
#PJM -j
#PJM -L elapse=0:10:00
#PJM -L jobenv=singularity

source /etc/profile.d/modules.sh
module load singularity-ce/4.1.3

export WORKDIR=/home/${GROUP}/share/containers
TORCH_CONTAINER_IMAGE=${WORKDIR}/pytorch_24.04-py3.sif
echo $TORCH_CONTAINER_IMAGE
singularity run --nv $TORCH_CONTAINER_IMAGE bash -c "nsys status -e && nsys profile -f true -o nsys-test -t osrt,cuda,cudnn,nvtx torchrun --nnodes 1 --nproc_per_node 2 ./examples/distributed/ddp-tutorial-series/multigpu_torchrun.py 50 10"
```

Reference: /home/pj25001011/share/nvidia/genkai-example-nv/gpu-tips/03_nsys /run_ddp.sh

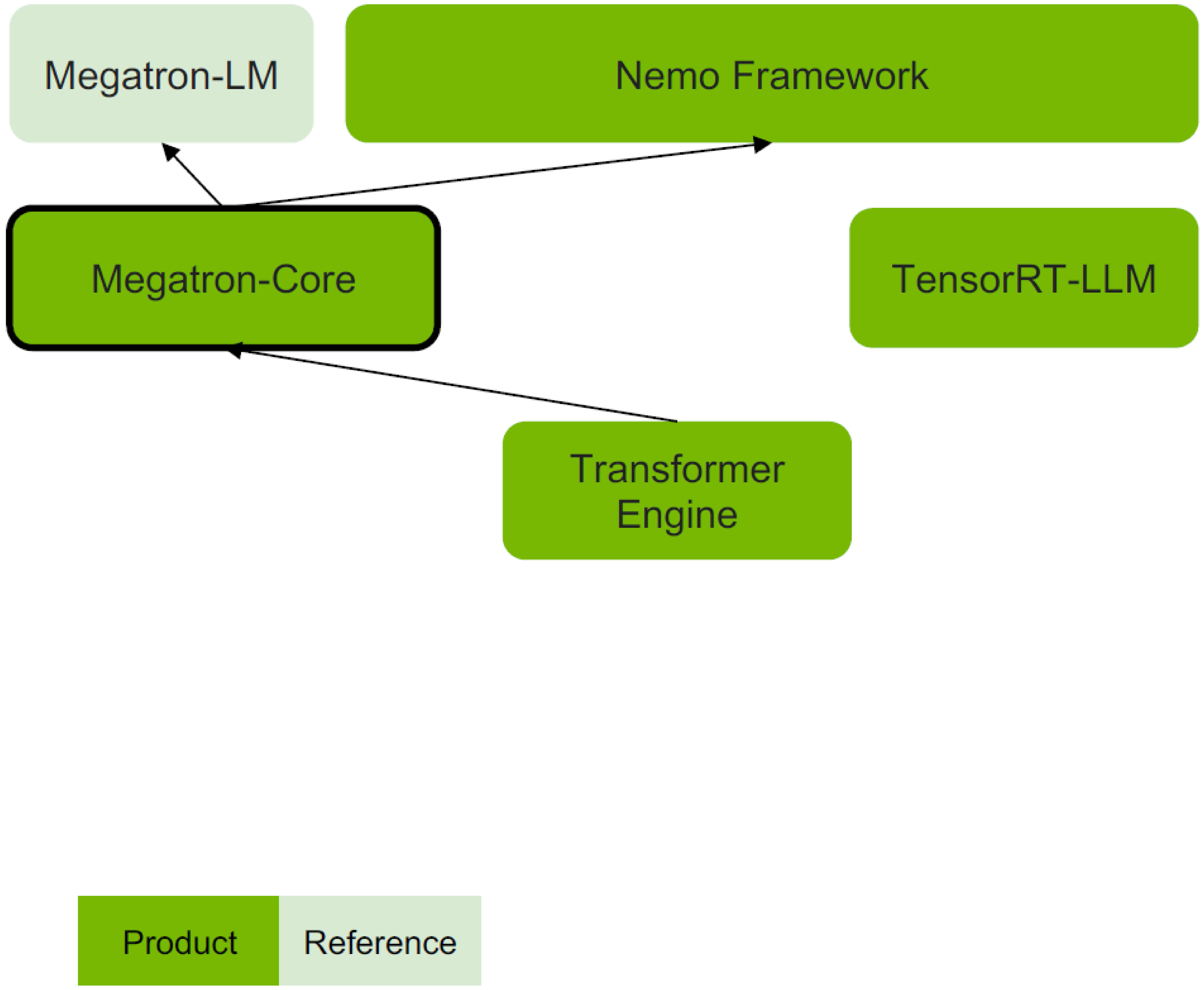
Nsight Systems を使ってみる

NVIDIA 純正 GPU プロファイラ

- nvtxによるラベルの可視化例

NeMo Framework/Megatron-LM

NeMo Framework/Megatron-LM/Megatron Core



Core value Proposition

Nemo Framework: Easy to use OOTB FW with a large model collections for Enterprise users to experiment, train, and deploy.

Megatron-LM: A **lightweight framework reference** for using Megatron-Core to build your own LLM framework.

Megatron-Core: Library for GPU optimized techniques for training transformer models at-scale.

Transformer Engine: Hopper accelerated Transformer models. Specific acceleration library, including FP8 on Hopper.

TensorRT-LLM: achieving optimal inference performance on the latest Large Language Models on NVIDIA GPUs

wandb インテグレーション

NeMo Framework/Megatron-LM/Megatron Core

- Wandbとは？
  - Weights & Bias 社の MLOpsプラットフォーム。実験結果の監視、GPUを含めたシステムメトリック情報など様々な機能を提供
  - `pip install wandb`でインストール後、`wandb login`して使用
  - 詳細はhttps://docs.wandb.ai/quickstartを参照ください

- NeMo Framework/Megatron-LMはwandbインテグレーション済であり、yamlまたは起動引数で指定するだけで簡単にwandb連携が可能です

NeMo Frameworkの場合以下のフラグを起動時に指定

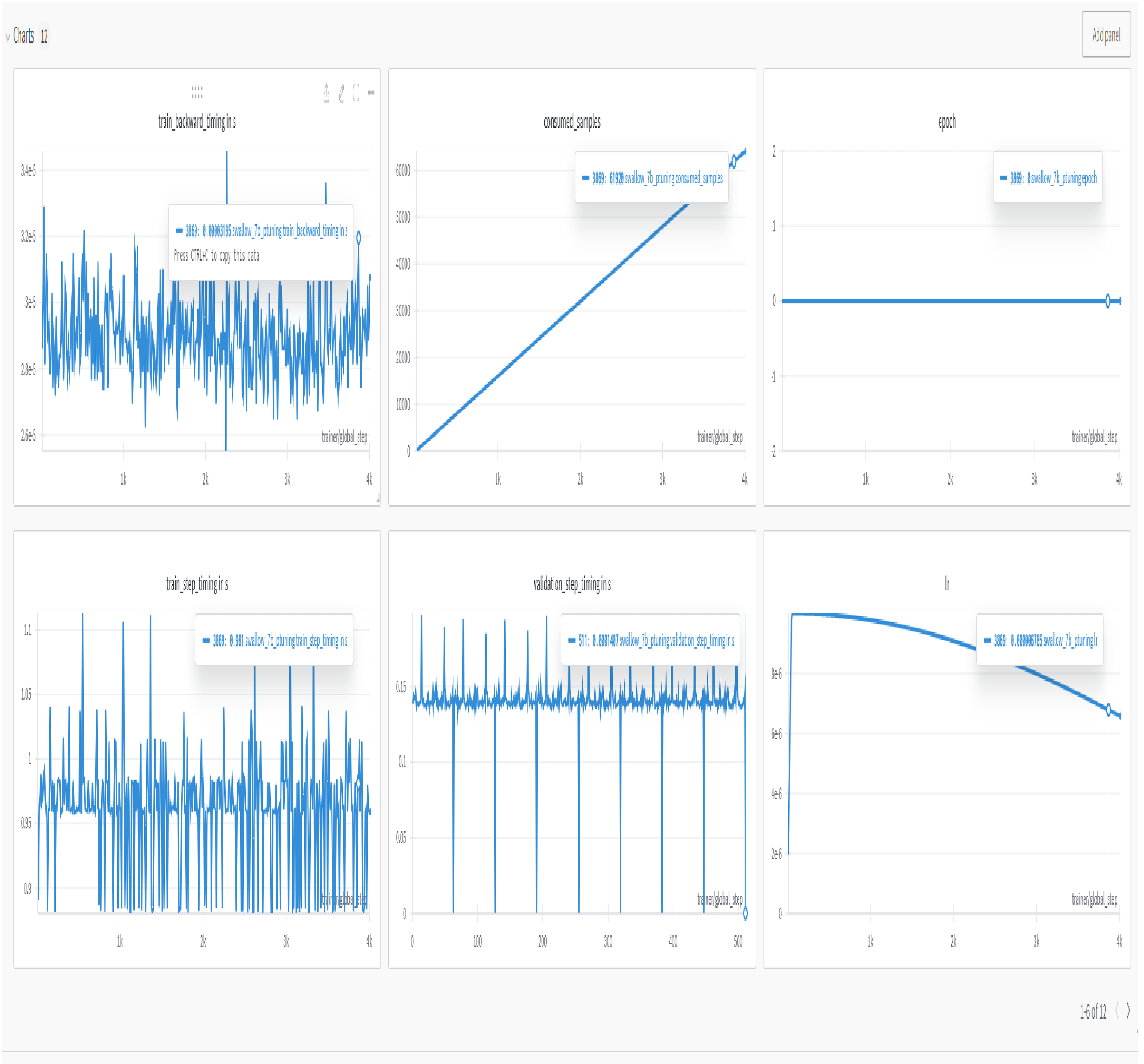
```
exp_manager.create_wandb_logger=True \
exp_manager.wandb_logger_kwargs.project=${PJ_NAME} \
exp_manager.wandb_logger_kwargs.name=${EXP_NAME} \
```

※Megatron-LM の情報は以下をご参照下さい
https://github.com/NVIDIA/Megatron-LM/blob/c4d12e26b2dc25a2eab7da92e2ac30338c0ed3de/examples/gpt3/gpt_config.yaml#L295

wandb インテグレーション

NeMo Framework/Megatron-LM/Megatron Core

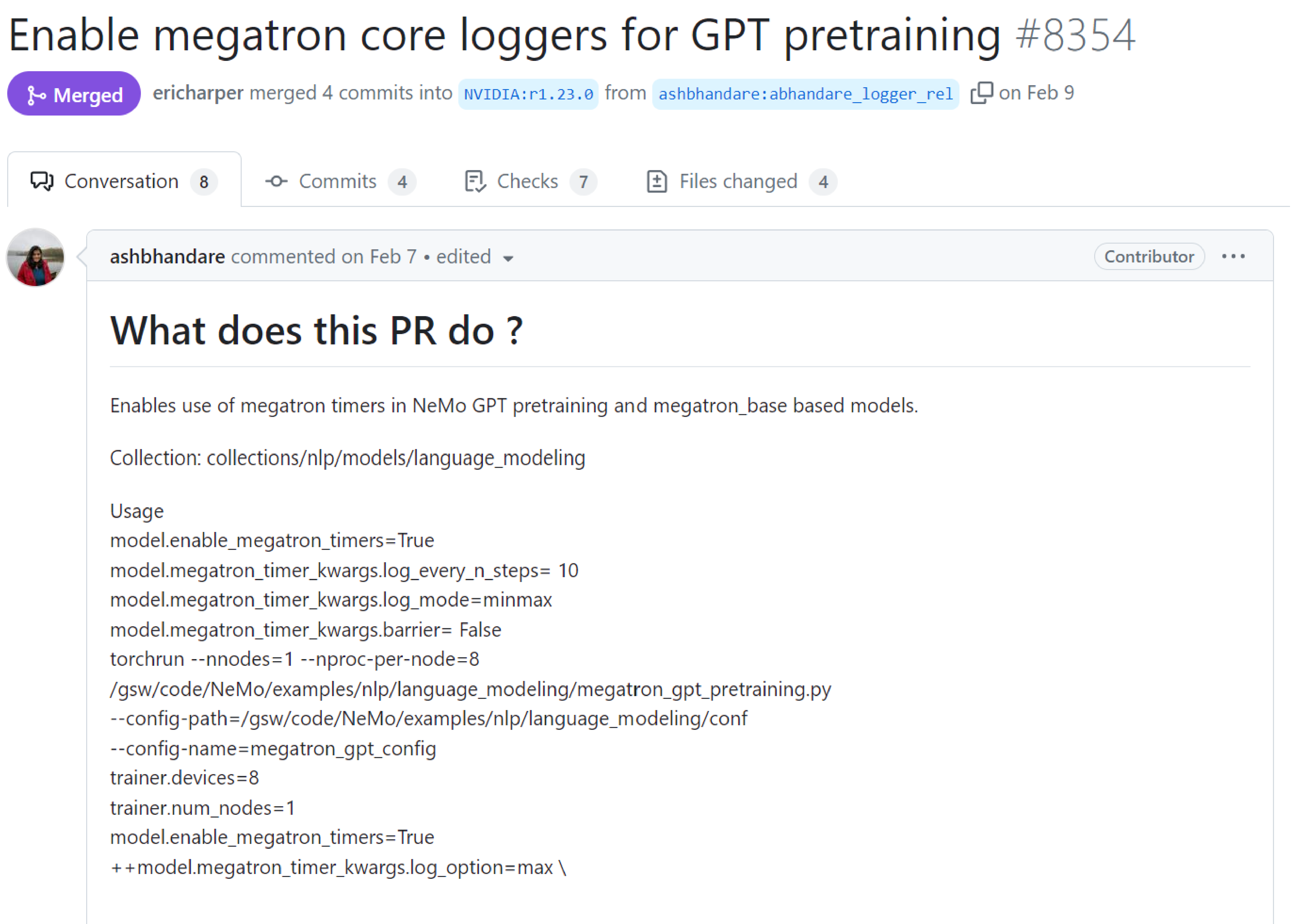
Wandb による可視化例



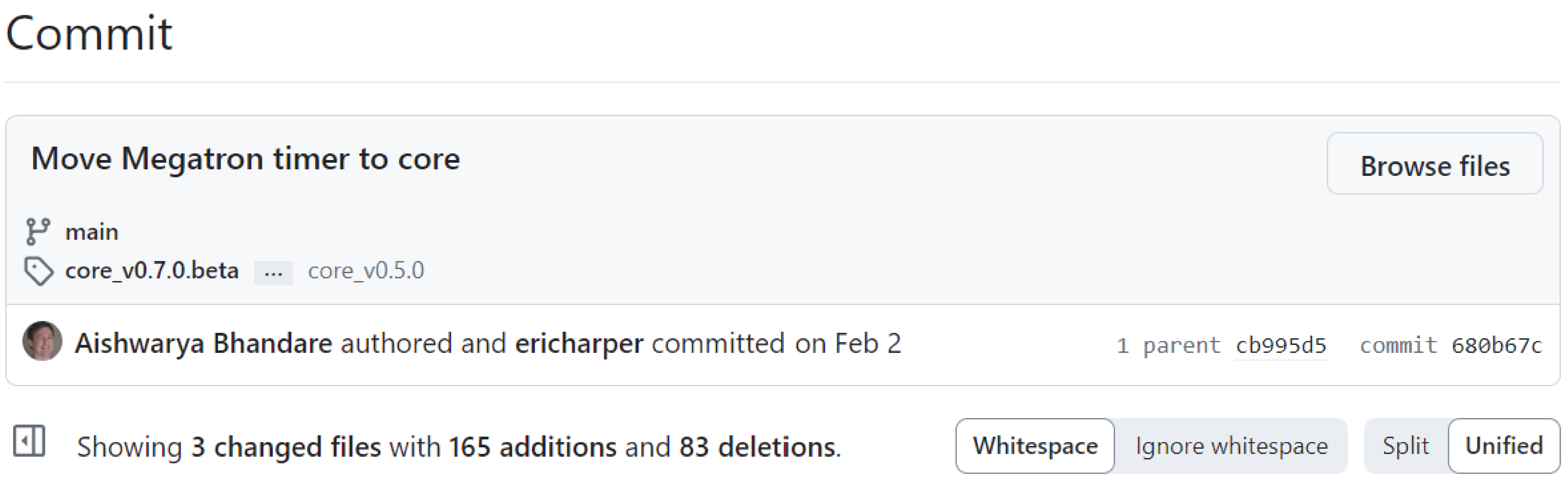
Megatron Timer

NeMo Framework/Megatron-LM/Megatron Core

- Megatron Core には学習の各処理時間をログに吐き出すTimer機能があり、YamlまたはTimer機能をONに出来る。学習のデバッグ時に便利



https://github.com/NVIDIA/NeMo/pull/8354



https://github.com/NVIDIA/Megatron-LM/commit/680b67c881b7b14a7bda32228f739fc27e88b429

Megatron Timer

NeMo Framework/Megatron-LM/Megatron Core

- NeMo Framework の場合以下のように指定することでTimerを有効化できる

```
export HYDRA_FULL_ERROR=1

singularity run --nv -B $USER_DIR $NEMO_CONTAINER_IMAGE \
    torchrun --nproc_per_node=${NGPU_PER_NODE} \
    /opt/NeMo/examples/nlp/language_modeling/tuning/megatron_gpt_finetuning.py \
    trainer.precision=bf16 \
    (..snip..)
    +model.enable_megatron_timers=True \
    ++model.megatron_timer_kwargs.log_option=max \
   (..snip..)
```

Timer ON 時の出力

```
ain_step_timing in s=0.439]^MEpoch 0: :  98%|
samples=784.0, train_step_timing in s=0.714]^MEp
=48.00, consumed_samples=784.0, train_step_timing in s=0.714]^MEpoch 0.
.170, global_step=49.00, consumed_samples=800.0, train_step_timing in s=0.687
    max time across ranks (ms):
        forward-backward ................................: 7432.24
        forward-compute .................................: 4061.91
        optimizer .......................................: 8770.77
        backward-compute ................................: 4037.24
        gradient_allreduce ..............................: 7.33

^MValidation: |          | 0/? [00:00<?, ?it/s]ESC[A
```

参考: https://github.com/NVIDIA/NeMo/blob/677203ab36d743c3398158f0f1d5fba552306993/examples/nlp/language_modeling/conf/megatron_gpt_config.yaml#L223

Appendix.

その他のTips 関連リンク

Nsight Systems

- GPU Profiling Overview (基礎編)
- Nsight Systems ユーザーガイド

NGC Catalog

- https://catalog.ngc.nvidia.com/
- NGC Catalog ユーザーガイド

Megatron-LM

- https://github.com/NVIDIA/Megatron-LM

NeMo Framework

- NeMo Frameworkコンテナ (NGC catalog)
- NeMo Framework ドキュメンテーション
- NeMo Framework Examples

Wandb

- https://docs.wandb.ai/quickstart

玄界用サンプルコード

- /home/pj25001011/share/nvidia/genkai-example-nv
- /home/pj25001011/share/nvidia/genkai-example-nv /README.md

NVIDIA®