NVIDIA NeMo to train, customize and deploy LLMs

Giuseppe Fiameni - gfiameni@nvidia.com

Solutions Architects, NVAITC EMEA

Agenda

- Introduction to NeMo Framework
- NeMo curator
- Pre-training
- Model customization
- Deployment
- Megatron-LM
- Use cases

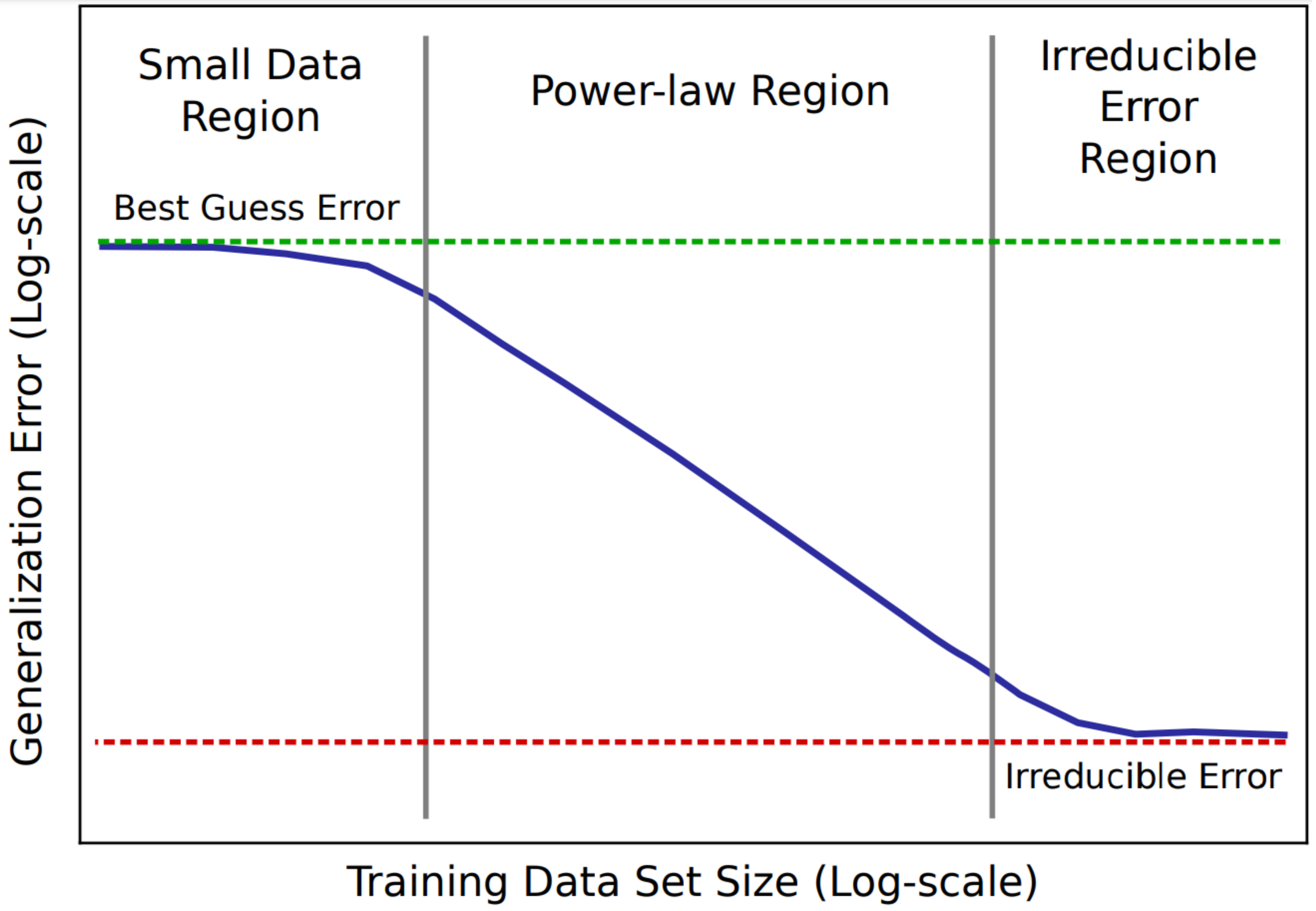
About Me

Giuseppe Fiameni – gfiameni@nvidia.com

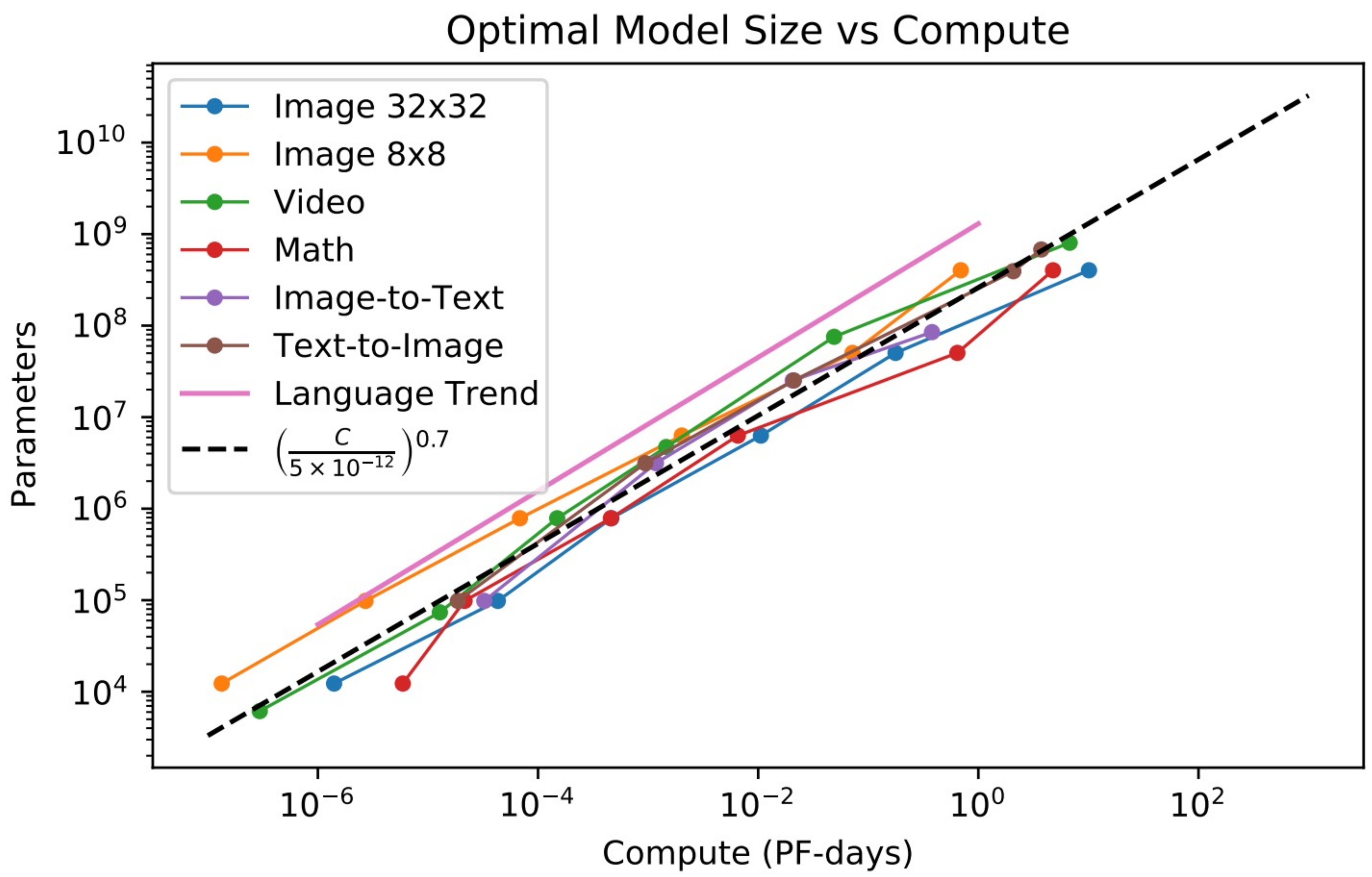
- Solutions Architect @ NVIDIA
  - Supporting Higher Education and Research through collaborations
- Lead of the **NVIDIA AI Technology Center (NVAITC)** program in EMEA
- HPC & AI

Scaling Laws of DL Training

Performance of neural networks increases with model/dataset size

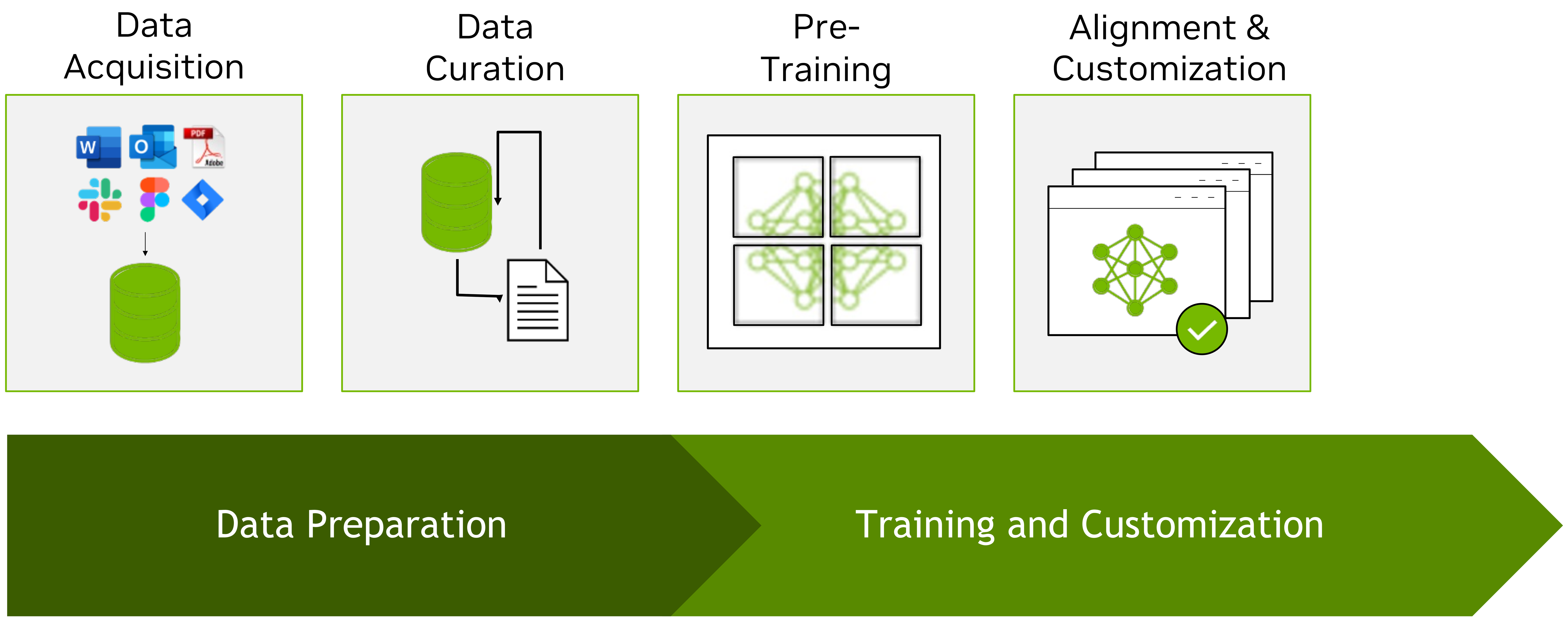


Joel Hestness et al, Baidu Research, 2017
Deep Learning Scaling is Predictable, Empirically
arXiv:1712.00409



Tom Henighan et al, OpenAI, 2020
Scaling laws for autoregressive generative modeling
arXiv:2010.14701

Building an LLM



LLMs in production

Overview of Nemo FW

https://github.com/NVIDIA/NeMo

Performance & Scalability

- More than 800 TFLOPs/sec/GPU
- Trained over 16k+ cluster size
- Supports 1M+ sequence length
- 4D parallelism
- GPU-accelerated data curation

Model Coverage

- Broad support for HF models
- 23 model families SOL accelerated - Incl LLM, SSMs, MOEs, SD, VLMs, VFMs, VLAs

SOTA Algorithms

- PEFT: LoRA, p-tuning, IA3, QLoRA, Adapters (Canonical)
- Reinforcement learning & Model alignment: GRPO, RLHF PPO, DPO, KTO, IPO, RLAIF, SteerLM, Rejection Sampling

Usability & Compatibility

- Hugging-face like pythonic APIs
- Fault tolerance and Resiliency to ensure smooth training via NVRx

Nemotron-4 340B Family of Models & Tools

Nemotron-4 340B Instruct

- Generate synthetic data at scale
- 9T tokens, 50 spoken + 40 programming languages
- Leading performance across benchmarks: Math, Instruction following, Human preferences

Nemotron-4 340B Reward

- Scores on: Helpfulness, Correctness, Coherence. Complexity, Verbosity
- Tops Reward Bench leaderboard covering chat, safety, and reasoning

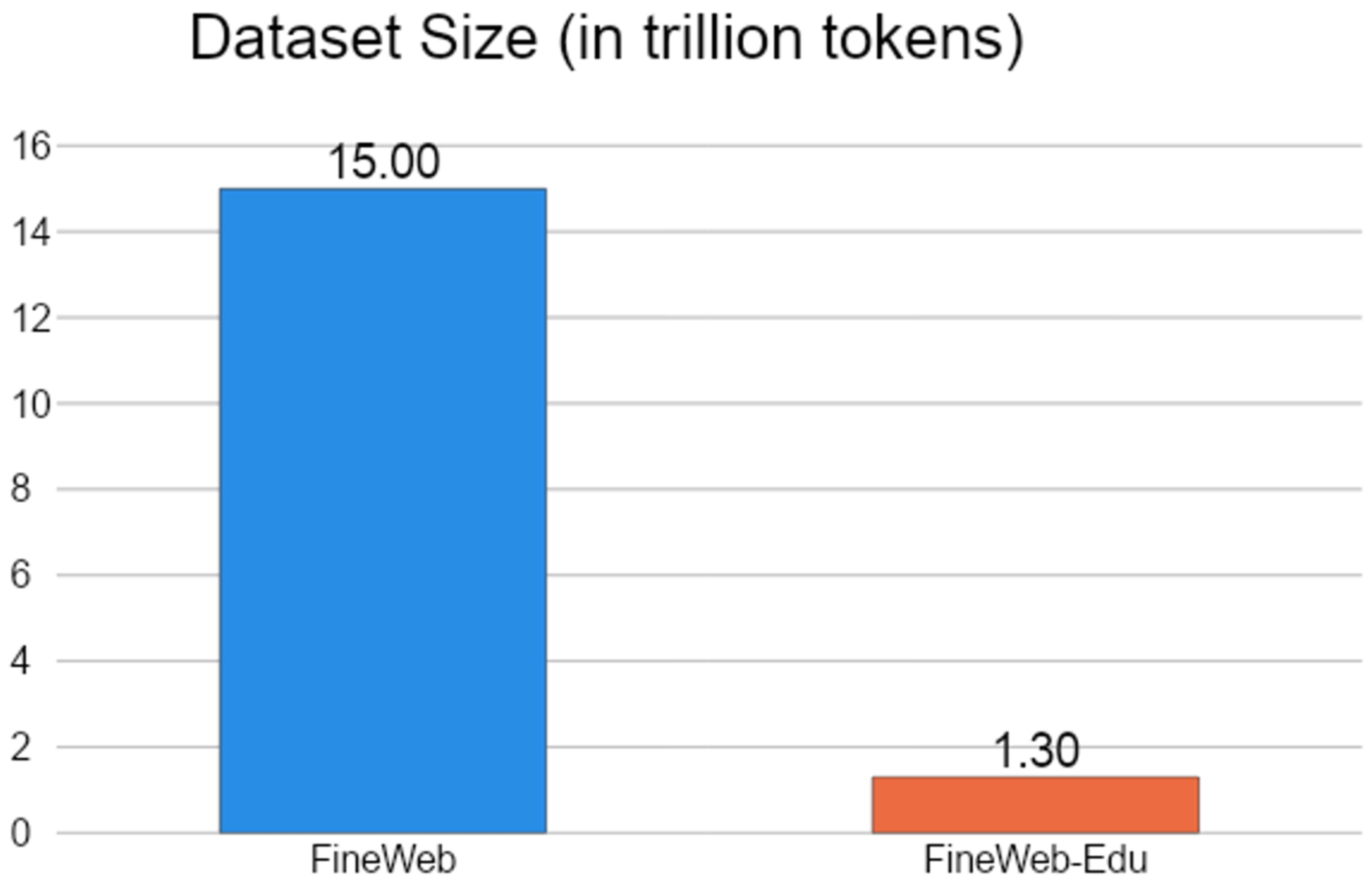
NeMo curator

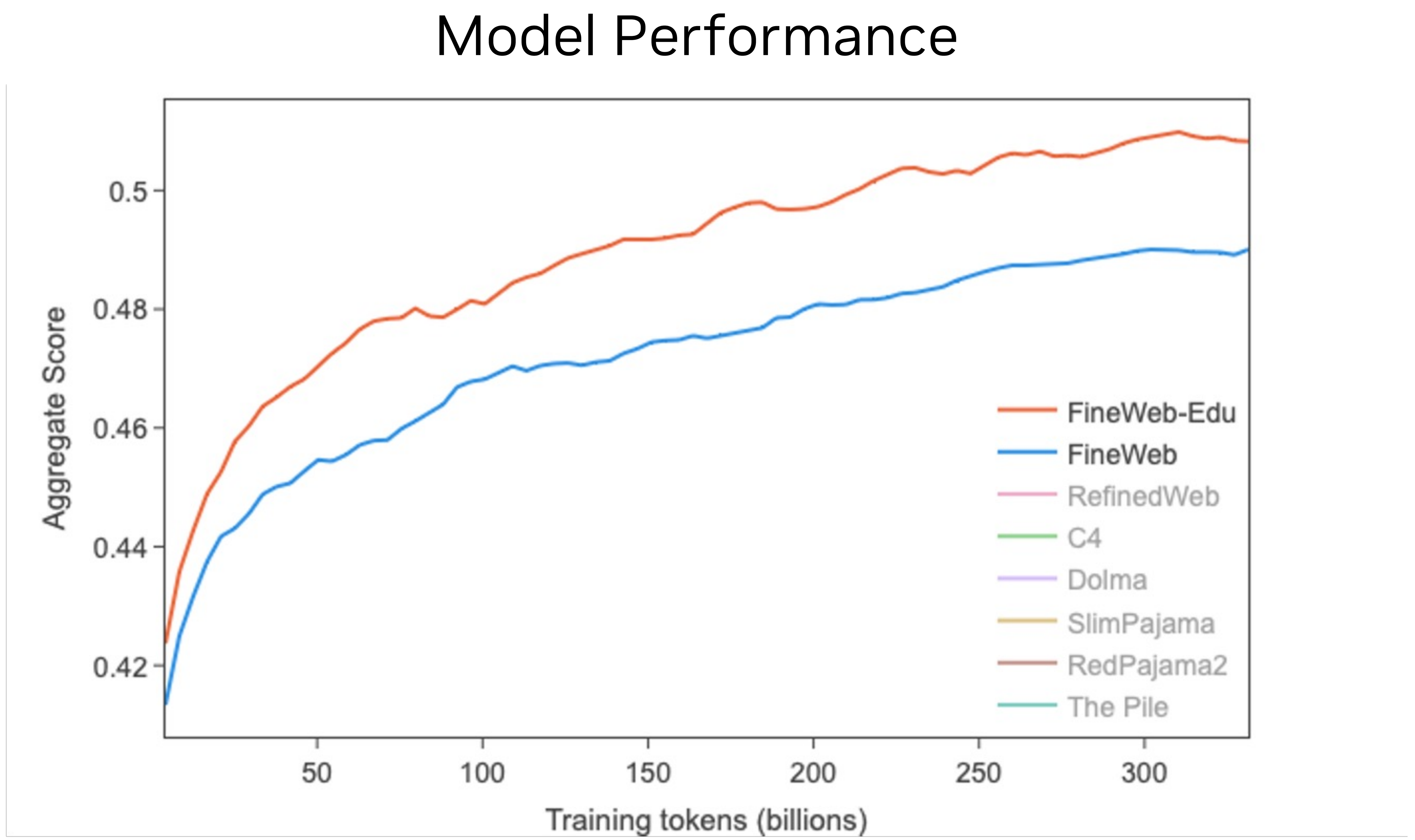
What is Clean Data?

✗	• The Earth is flat, and NASA is hiding the truth. • John Doe’s Social Security Number is 123-45-6789. • 9375028164730592846159273046851273 • test test test test test test • …
✓	• In machine learning, a neural network (also artificial neural network or neural net, abbreviated ANN or NN) is a model inspired by the structure and function of biological neural networks in animal brains.

Why Data Quality Matters

Less is more when pre-training LLMs





FineWeb: decanting the web for the finest text data at scale - https://hf.co/spaces/HuggingFaceFW/blogpost-fineweb-v1

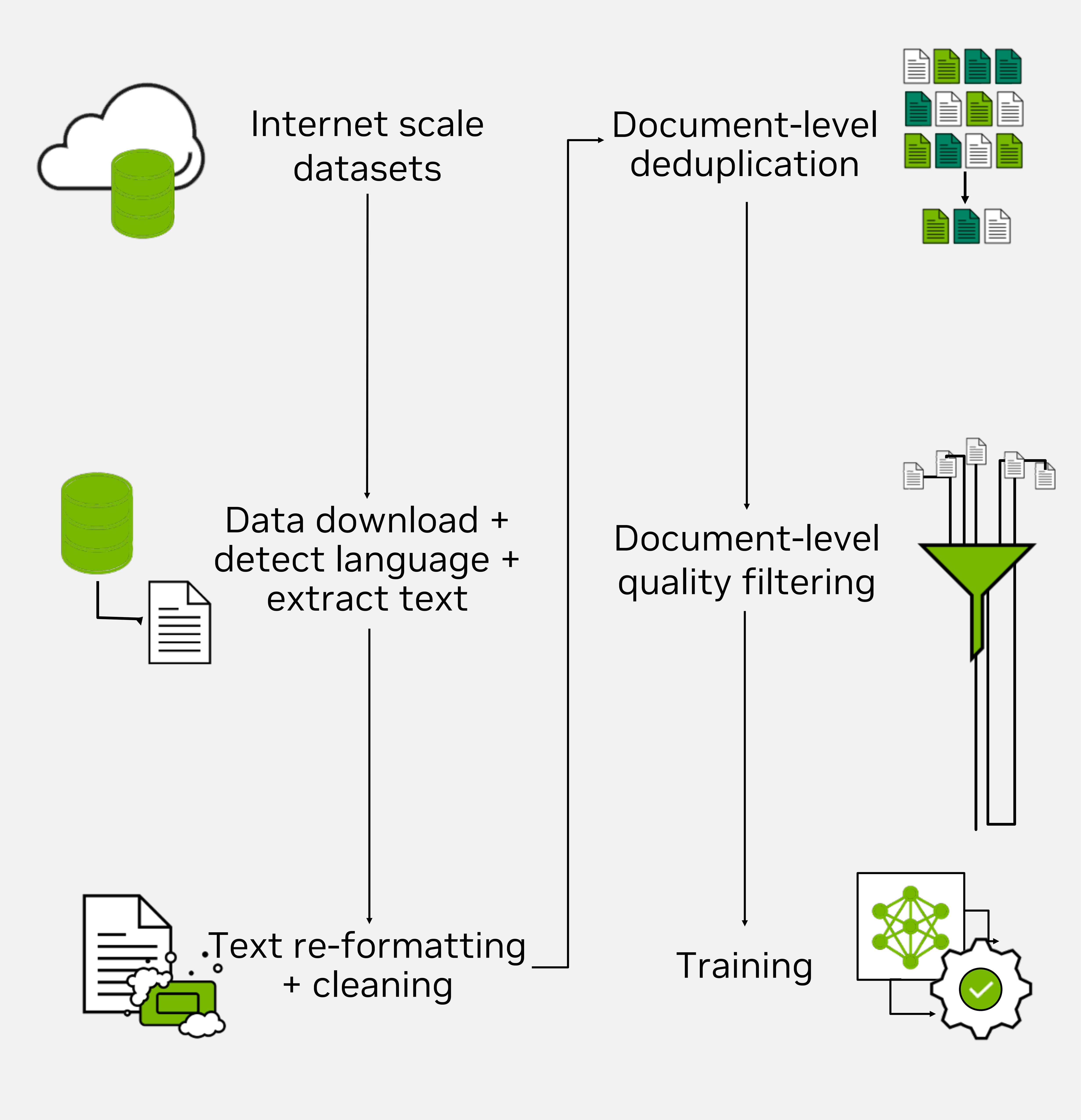
Data Curation Improves Model Perfomance

NeMo Data Curator enables large-scale high-quality datasets for LLMs

- Reduce the burden of combing through unstructured data sources
- Download data and extract, clean, deduplicate, and filter documents at scale

NeMo Data Curator steps:

1. Data download, language detection and text extraction - HTML and LaTeX files
2. Text re-formatting and cleaning - Bad Unicode, newline, repetition
3. GPU accelerated Document Level Deduplication
   - Fuzzy Deduplication
   - Exact Deduplication
4. Document-level quality Filtering
   - Classifier-based filtering
   - Multilingual Heuristic-based filtering

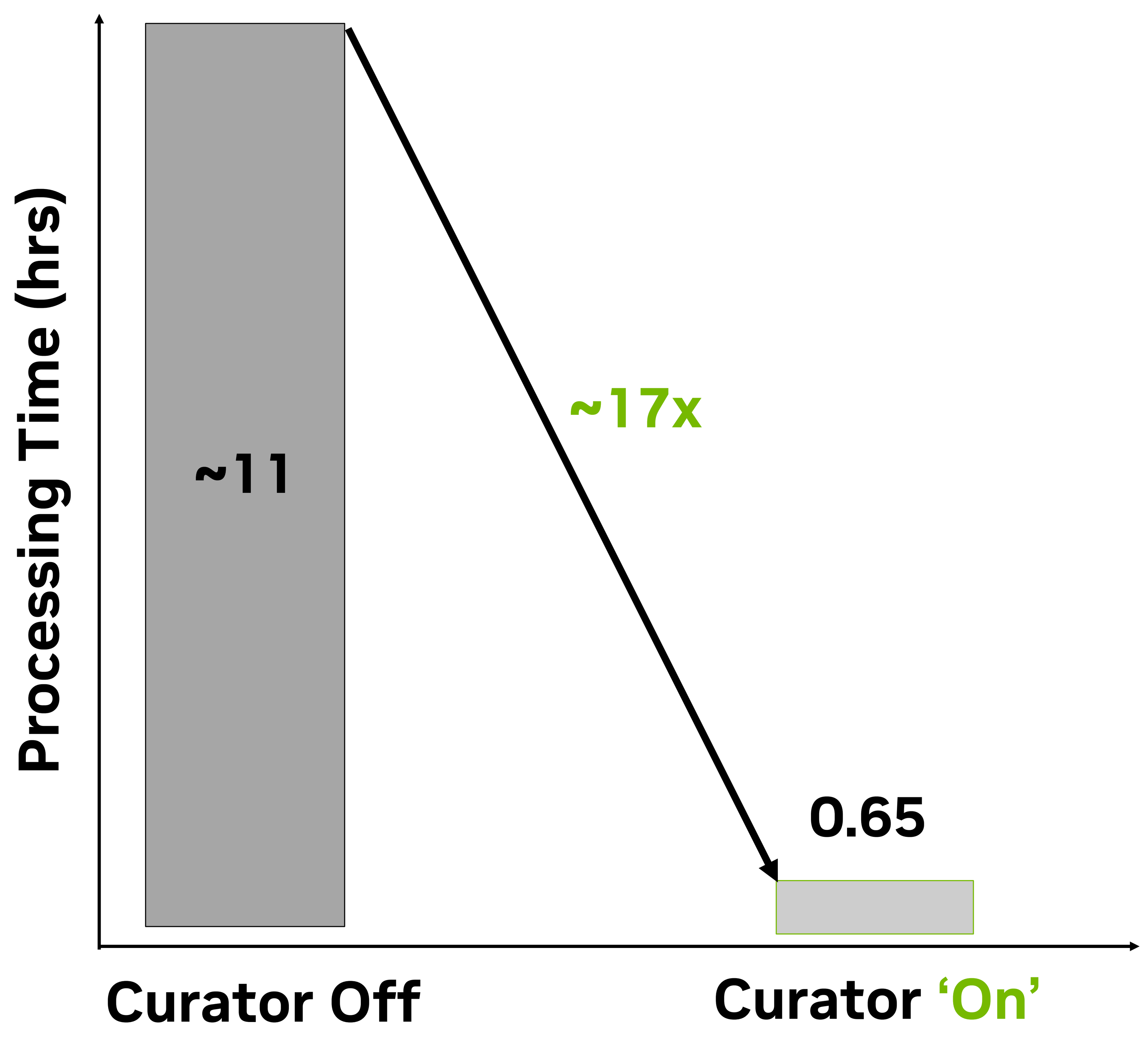


NeMo Curator : Text Processing Architecture

Easily integrate different features into your existing pipelines with Python APIs

NeMo Curator: World-Class Benchmarks

~17x Faster Processing

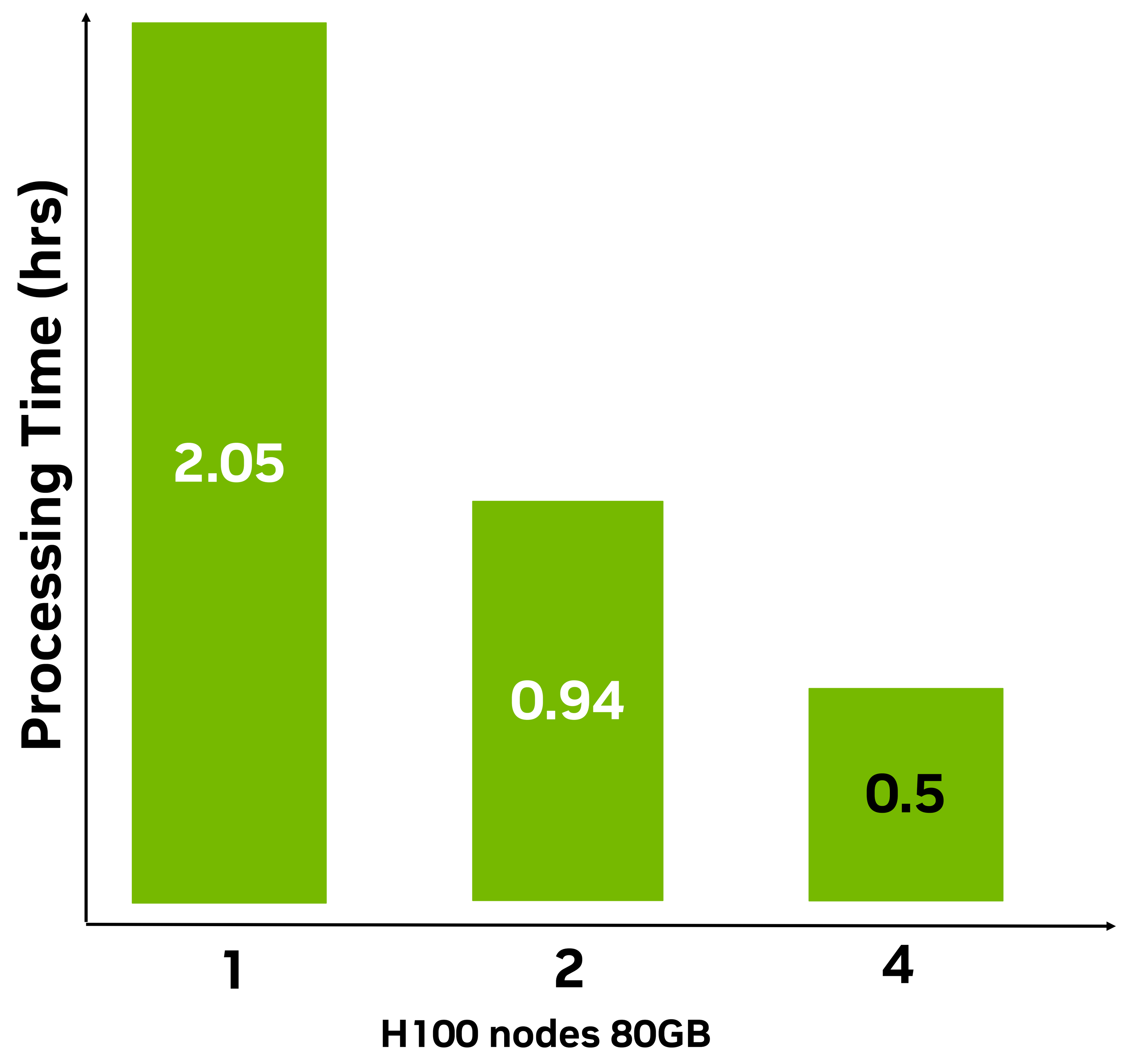


Processing time for fuzzy deduplication of RedPajama-v2 subset (8TB/1.78T tokens)

'On': Data processed with NeMo Curator

'Off' : Data processed with a leading alternative library on CPUs

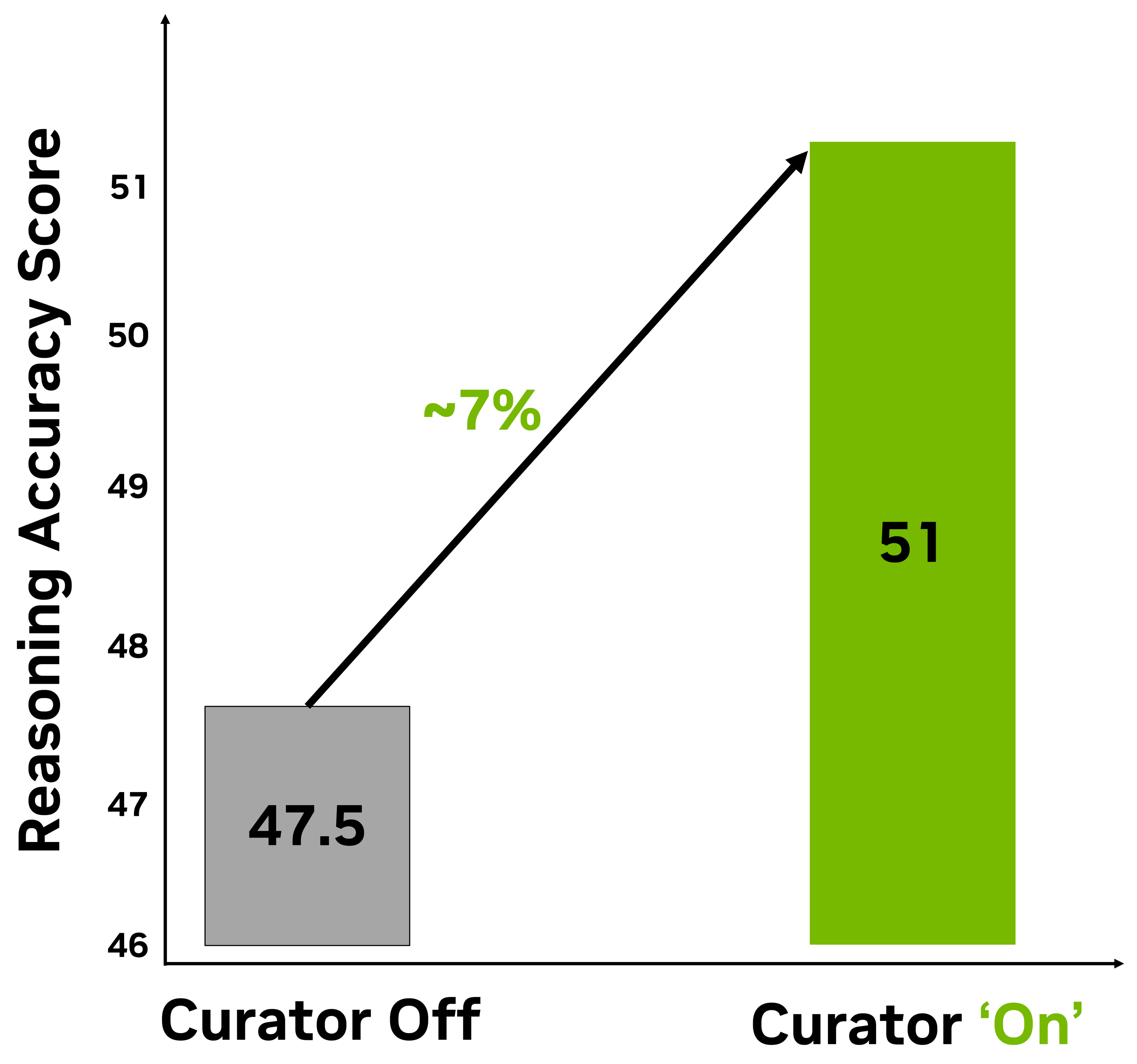
Near Linear Scaling



Processing time for fuzzy deduplication of RedPajama-v2 subset (8TB/1.78T tokens)

Scaling on 1, 2, 3, 4 H100 nodes 80GB

~7% Relative Improvement in Accuracy



Reasoning Accuracy: Average score on Race, PiQA, Winogrande, and HellaSwag for 357M GPT base model. (Scale: 0-100)

'On': Processed data with NeMo Curator

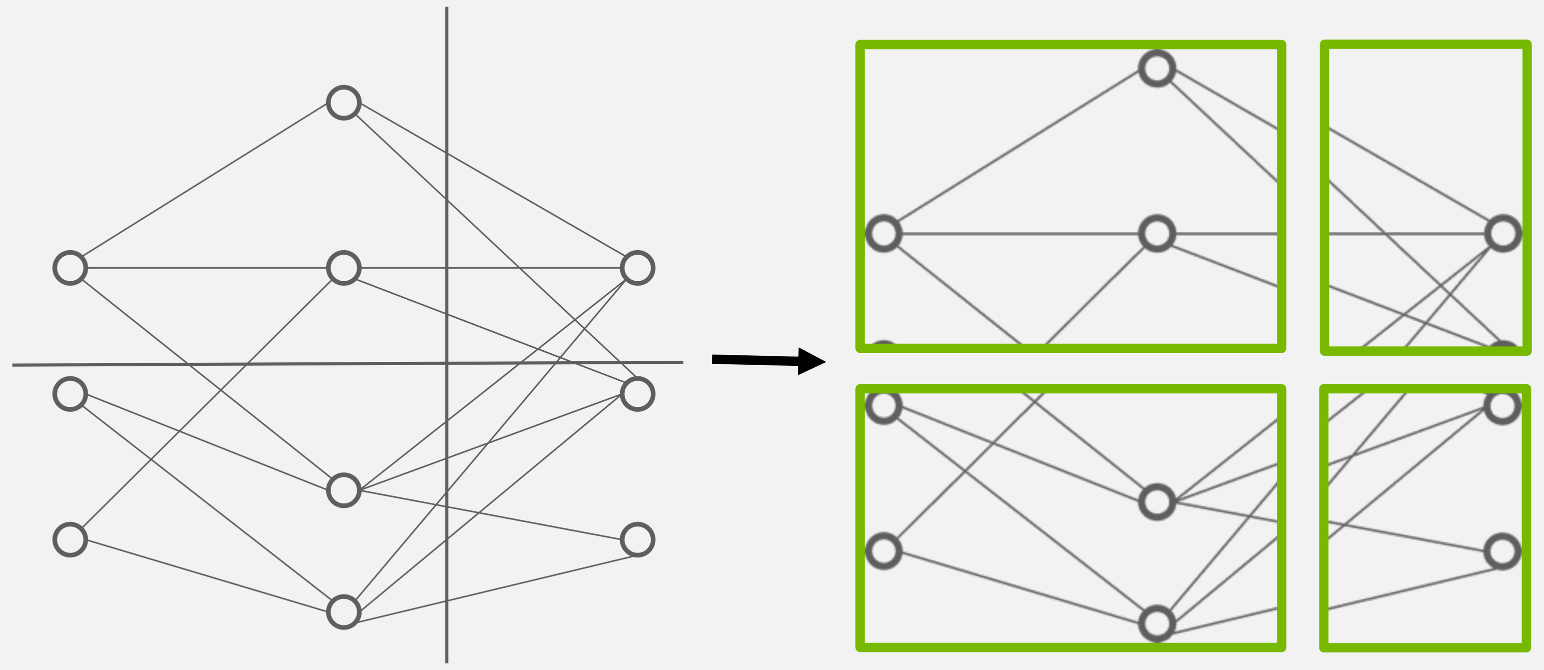
'Off' : Raw data without curation

Pre-training

Building Generative AI Foundation Models

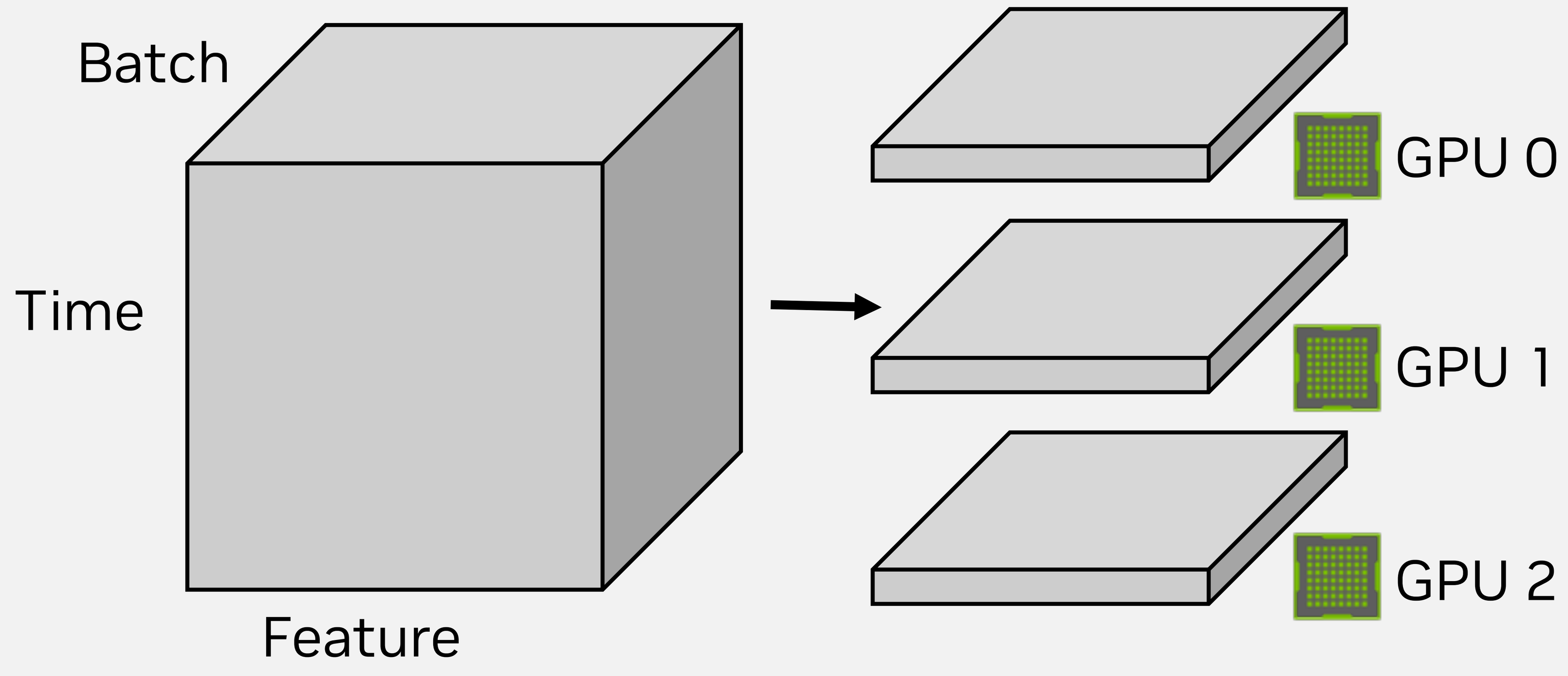
Efficiently and quickly training models using NVIDIA NeMo

Tensor & Pipeline Parallelism



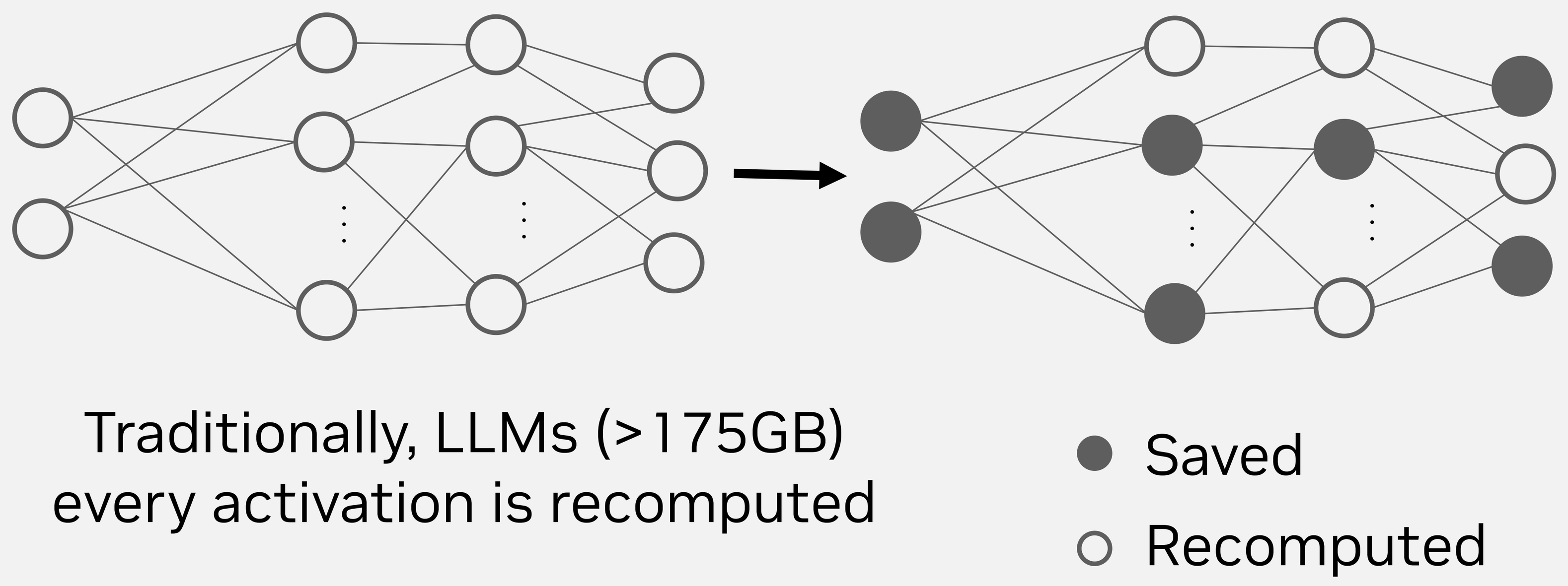
Reduced memory footprint and allows for large-scale training of LLMs across accelerated infrastructure

Sequence Parallelism



Working with tensor processing to increase the batch size that can be support for training

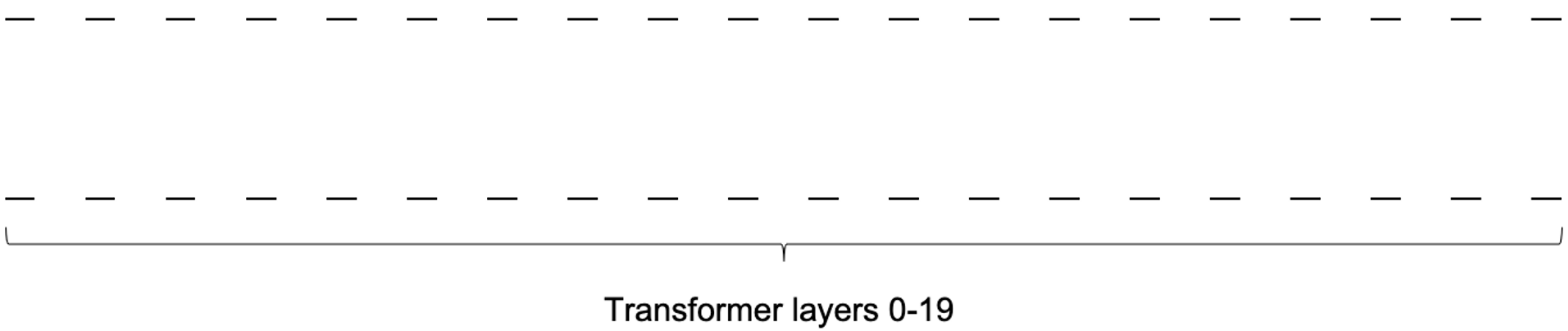
Selective Activation Recomputation



Smart activation checkpointing provides greatest trade-off between memory and recomputation

Different Parallelism - An example

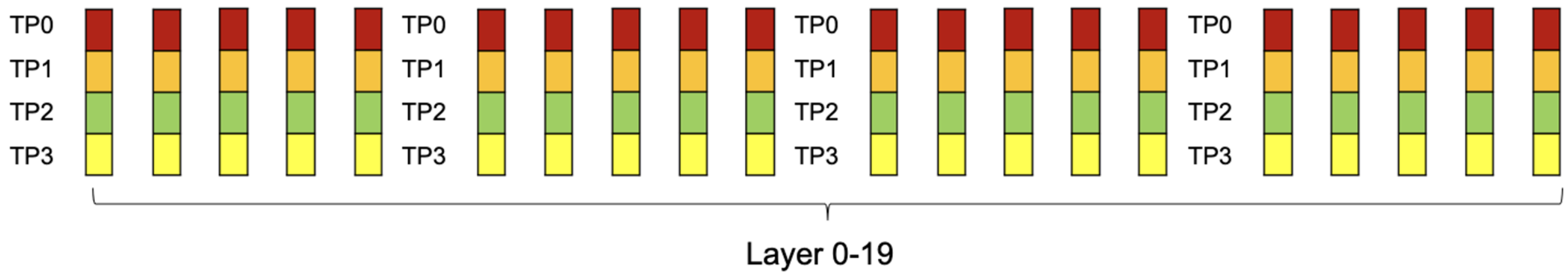
- Take a model with 20 layers as example
- Data parallel size: 2
- Tensor parallel size: 4
- Pipeline parallel size: 4



Different Parallelism - An example

Start with Tensor Parallelism

- Let's start with the tensor parallelism
- Tensor parallel size: 4
- Intra-layer splits
- All-Reduce across different Tensor Parallel ranks(TP0-TP3)



Different Parallelism - An example

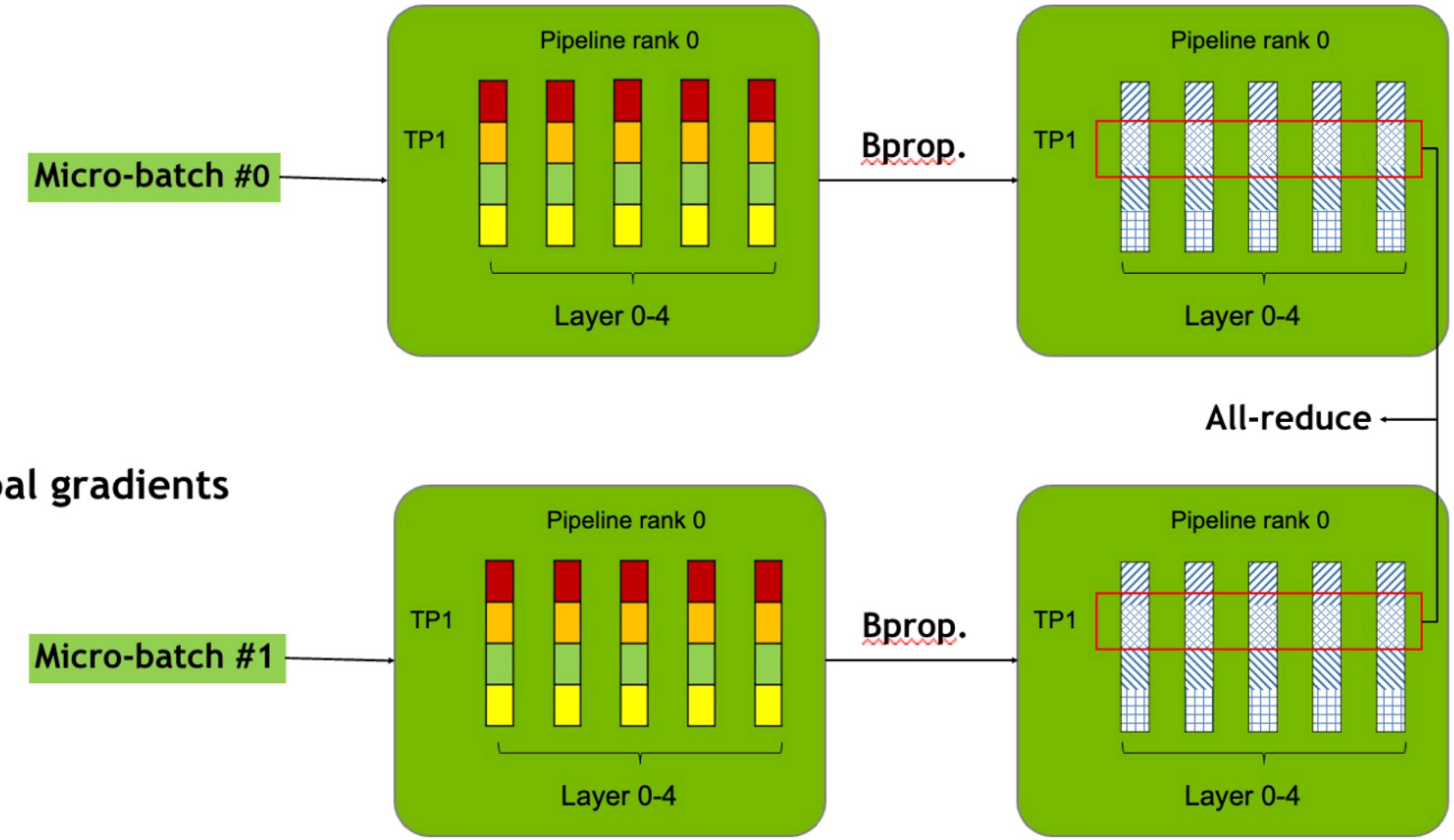
Add Pipeline Parallelism

- Then add pipeline parallelism
- Pipeline parallel size: 4
- Inter-layer splits
- Send-Receive between adjacent pipeline ranks
- Now 16 devices are engaged in the model parallelism

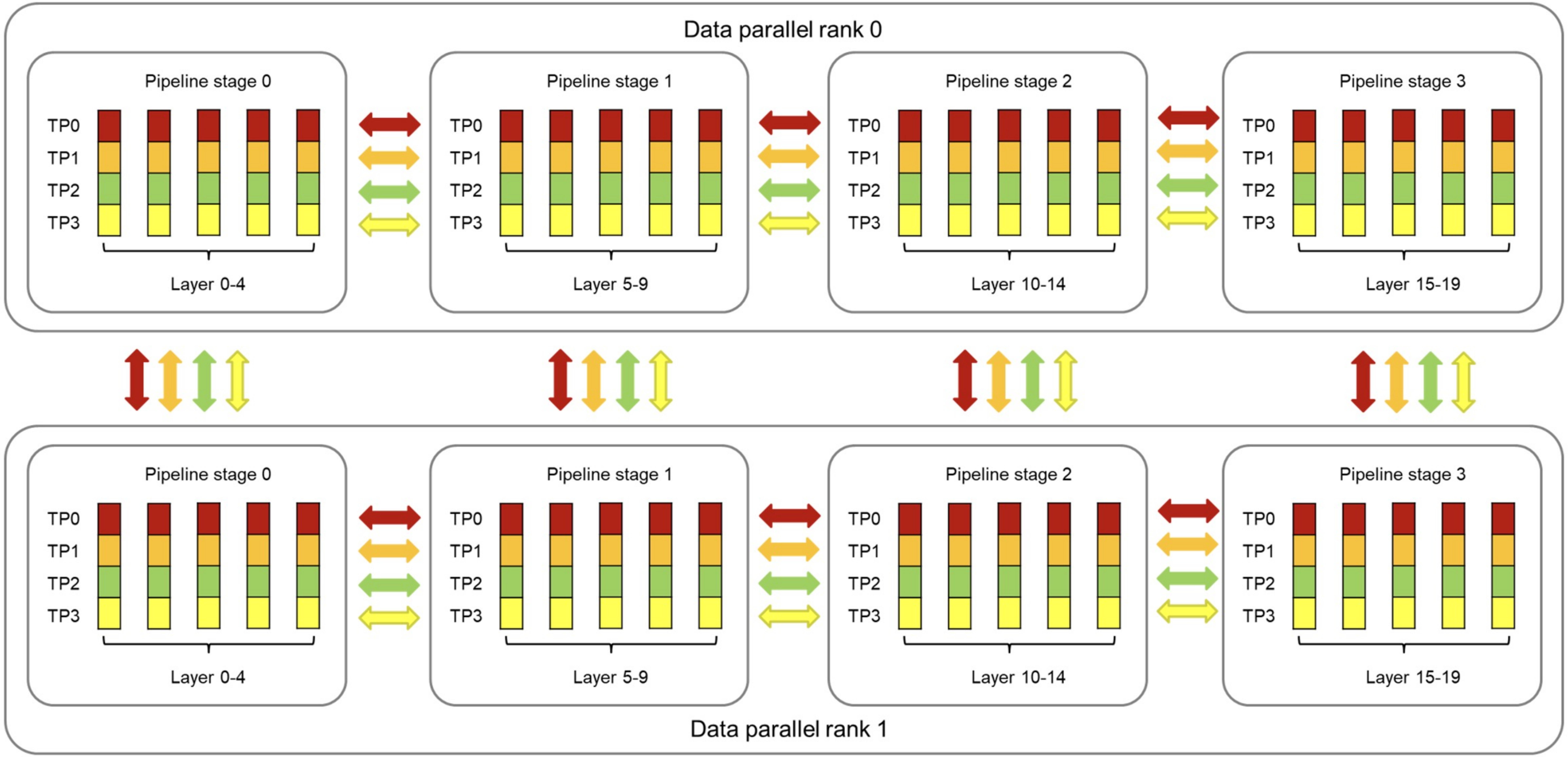
Different Parallelism - An example

Add Data Parallelism

- Finally add data parallelism
- Data parallel size: 2
- Each device has a paired device
- Paired devices:
  - have same parameters
  - consume different micro batches
  - perform all-reduce to get the global gradients



Different Parallelism - An Example

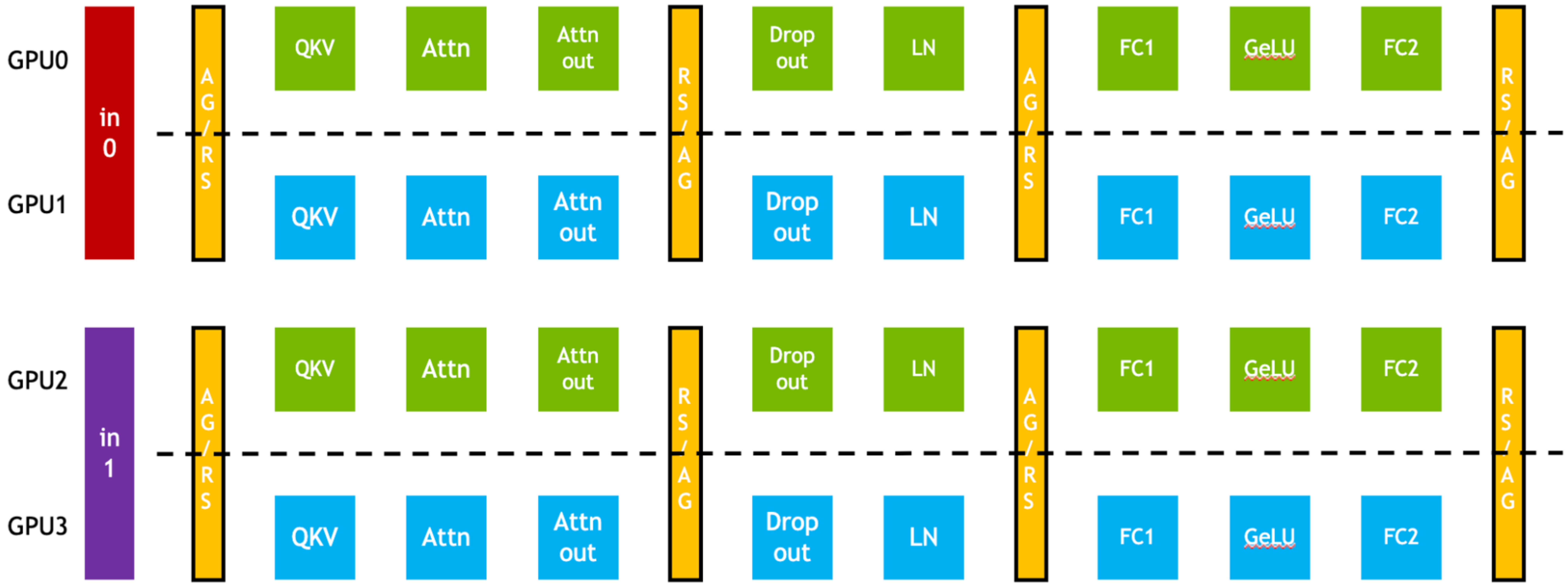


Expert Parallelism

Distributing experts in mixture-of-experts layers over GPUs

Sequence Parallelism (SP)

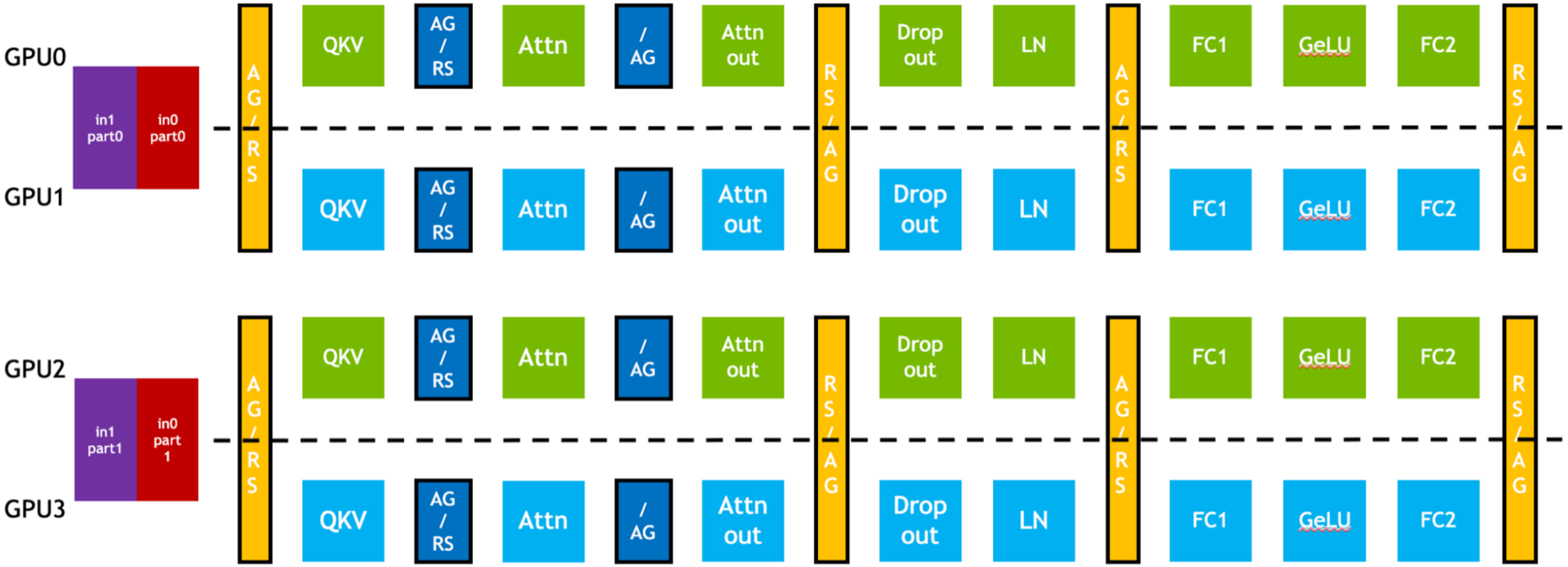
- SP only shards the outputs of specific transformer layers (typically Dropout and LayerNorm activations) along the sequence dimension across tensor parallel (TP) ranks.



Column-split: QKV and FC1, Row-split: Attn-output and FC2, SP only splits activations of dropout and LN along seq dim
AG/RS: AG in fwd and RS in bwd, RS/AG: RS in fwd and AG in bwd

Context Parallelism (CP)

- CP partitions the entire sequence (all activations, not just specific layers) across GPUs

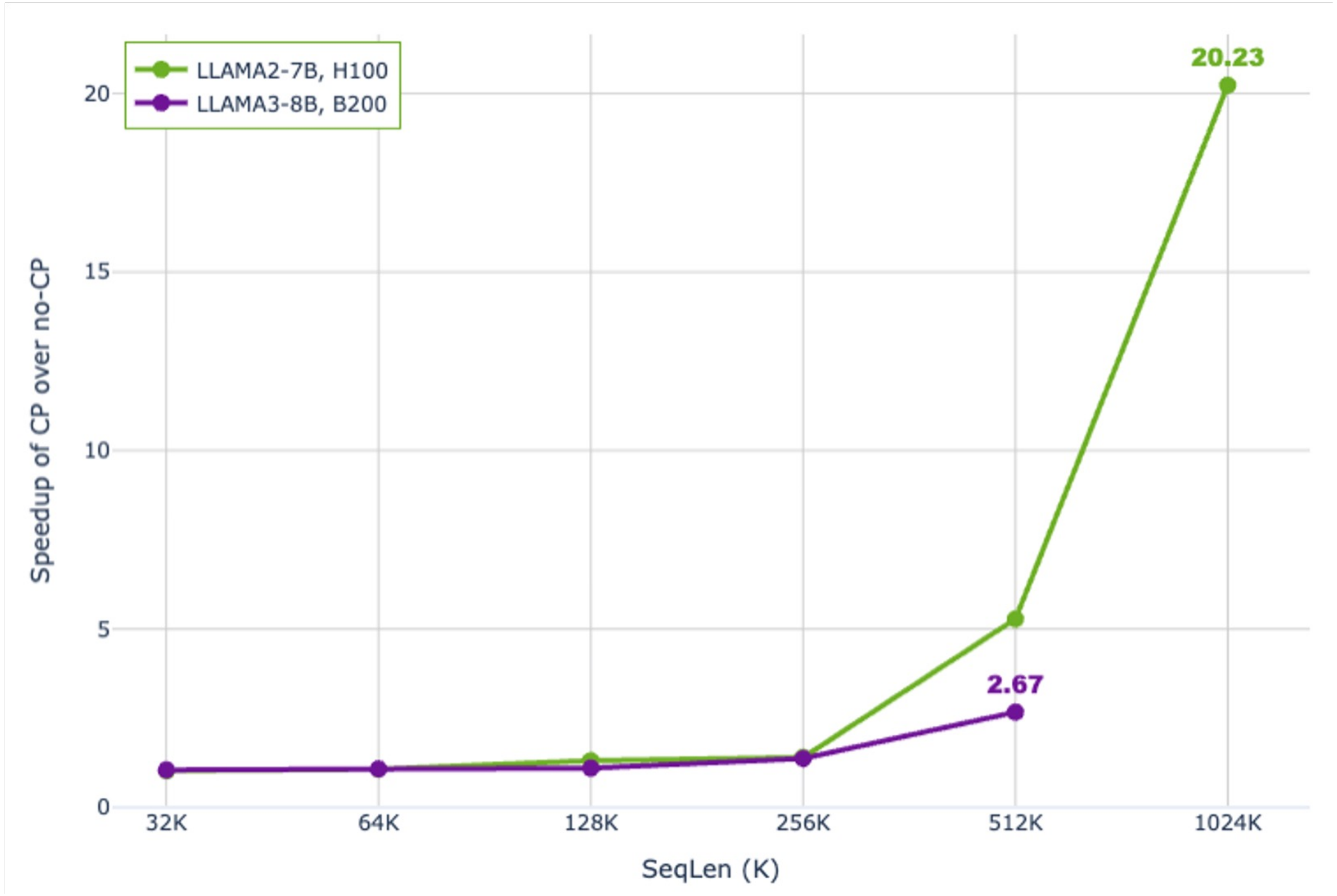


Column-split: QKV and FC1, Row-split: Attn-output and FC2, CP splits activations of whole transformer layer along seq dim
AG/RS: AG in fwd and RS in bwd, RS/AG: RS in fwd and AG in bwd, /AG: No-op in fwd and AG in bwd

SP and CP

- Sequence Parallelism
  - Splits the *processing of tokens* in a sequence across multiple GPUs.
  - Each GPU has a copy of the model parameters and works on a different chunk of the input tokens.
  - Mostly used for specific layers (like Dropout or LayerNorm) and is *often combined with tensor parallelism for other parts of the model*.
  - Only certain intermediate activations (not all) are partitioned, and communication needs are lower since each GPU can process its chunk mostly independently.
- Context Parallelism
  - Splits the *entire sequence* (all tokens and their activations) across multiple GPUs.
  - All network inputs and all intermediate activations are partitioned along the sequence dimension, not just some layers.
  - *Each GPU works on a subset of the sequence and must communicate with others to compute attention*, since attention requires information about all tokens (not just the local chunk).
  - Can be used standalone (without requiring tensor parallelism), and is *especially useful for very long sequences*, reducing the memory burden on each GPU.

Context Parallelism for Long-Context Training

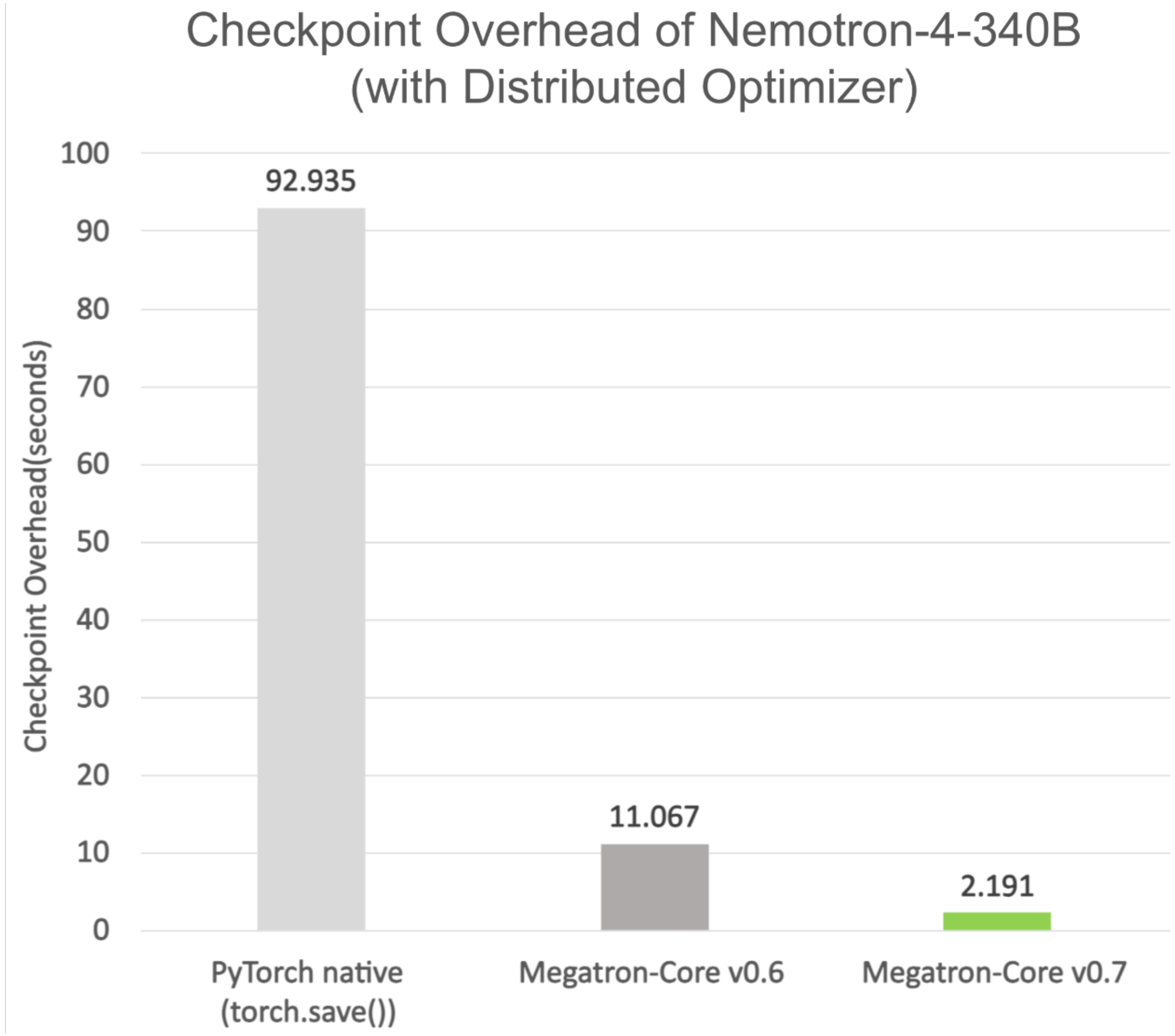


Llama2-7B, H100, Megatron-Core v0.6 with nemo.24.03.01 container

- **Context parallelism** (CP) partitions the network inputs and all activations along sequence dimension, whereas the previous **Sequence parallelism** (SP) only splits the sequence of Dropout and LayerNorm activations.
- With CP, Megatron-Core achieves **20x speedup** for a Llama2-7B model on 1024 H100 GPUs with 1024K seq length.
- Both CP and SP are also supported in the LLaVA pipeline in Megatron-Core for multimodality training.

[Context Parallelism overview](#) | [Benchmark details](#)

Fast Distributed Checkpointing for Large-scale Training



- Megatron-Core's fully parallel and async approach achieves a **42x reduction** in checkpoint overhead for a Nemotron-4-340B, compared to native torch.save().
- Users have the flexibility to use different training configurations when resuming from a checkpoint saved with Megatron-Core.
- "save" and "load" APIs almost transparently replace the equivalent APIs in Pytorch for ease-of-use.

[Train Generative AI Models More Efficiently with New NVIDIA Megatron-Core Functionalities - Blog]

Integration with NVIDIA Resiliency Extension

What is NVIDIA Resiliency Extension?

- Python package for extending PyTorch-based frameworks with resiliency features
  - pip install nvidia-resiliency-ext
- Can be used standalone or with Megatron-Core and NeMo
- Open source on GitHub: https://github.com/NVIDIA/nvidia-resiliency-ext

Functionality

- Straggler GPU detection
- Fault and hang detection
- In-job auto restart and graceful exit
- Coming soon:
  - In-process restart
  - Hierarchical Checkpointing (In-memory & Global storage)

NeMo & Megatron-LM integration

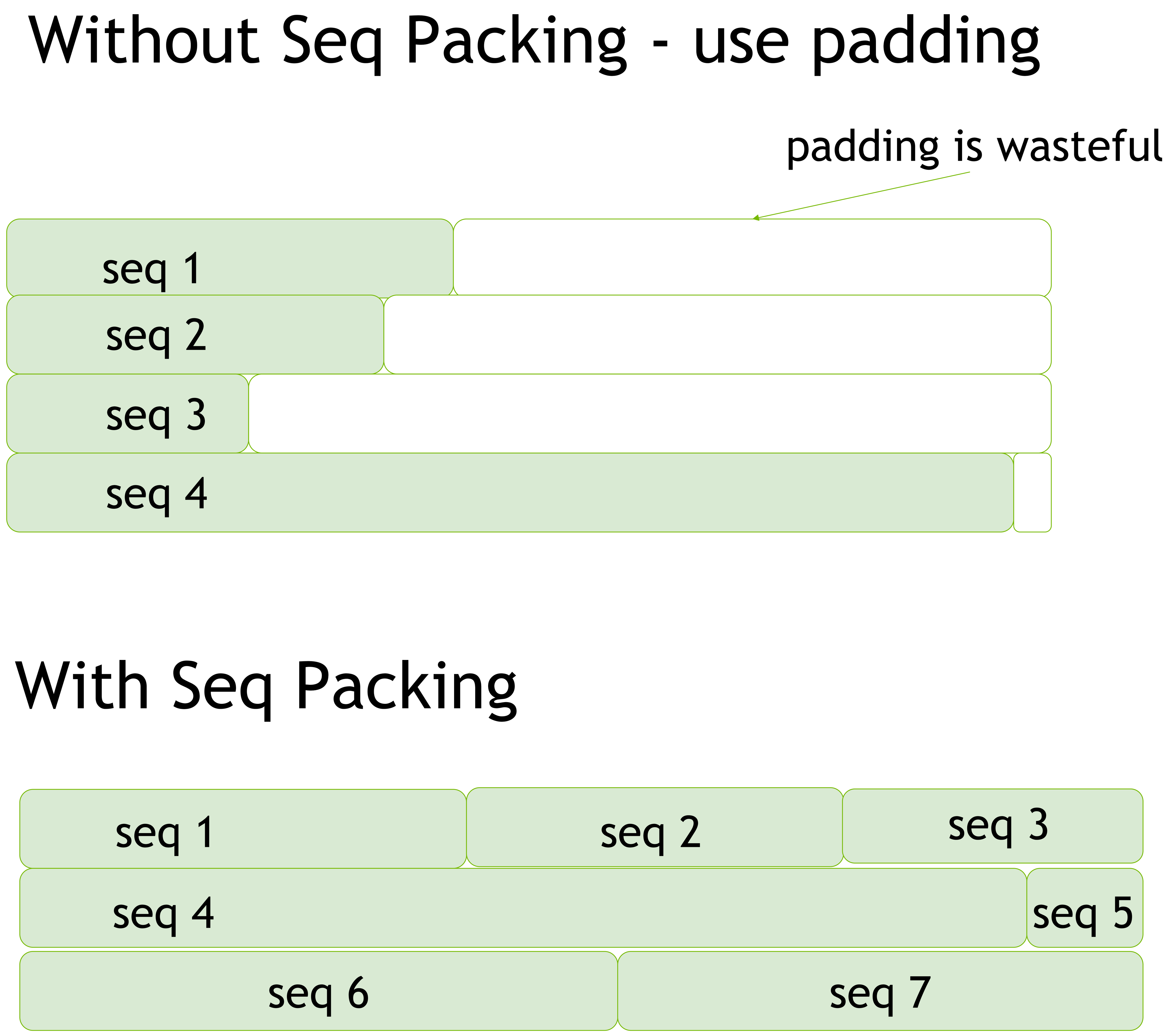
- Integration w/ NeMo and Megatron-LM (since MCorev0.9)
- Recommended as the best path to achieve the highest effective training time and performance for LLMs

	~10-20s	~1-2 min	Few minutes
	Automatic restart of training loop with healthy ranks without process restart.	Automatic restart of training processes without job restart.	Automatic restart of training job without user intervention.
	Recover from transient network link flap.	Recover from corrupted CUDA contexts (e.g., uncorrectable ECC).	Recover from permanent node or GPU failure.

NVIDIA Resiliency Extension

New Feature - Sequence Packing

- Sequence packing is a technique to improve training efficiency when handling datasets with variable-length sequences, which is common in language modelling
- Eliminates the need for padding
- Allows more tokens to be processed in each micro batch, maximizing both GPU compute and GPU memory.
- Makes use of variable-length attention kernels **(THD attention)** in FlashAttention and *TransformerEngine*, to avoid calculating attention values between sequences. This allows packing sequences to arbitrary lengths.



Performance Tuning Practice

Prerequisites

- Use the latest NVIDIA PyTorch or NeMo Image unless custom container is provided
- Enable Distributed Optimizer
  - With distributed optimizer, master weights and optimizer states will be sharded across all DP ranks. Try increasing the number of GPUs and reducing model parallel size to increase perf.
- Enable Communication Overlapping
  - *--tp-comm-overlap*
  - *--overlap-grad-reduce*
  - *--overlap-param-gather*
- Enable context parallel for long context training.
  - The efficiency of CP largely depends on whether its communication can be overlapped.
- Enable Grouped GEMM if num_local_experts > 1 for MoE
  - Recommended to use the TE version of Grouped GEMM (requires upgrading to MCore v0.8 and TE v1.7).
  - Supports Gradient Fusion and FP8.

Performance Tuning Practice

General rule: don't over-parallelize!

- Use just data parallelism + distributed optimizer if possible
- If this OOMs, use tensor model parallelism + sequence parallelism
    - In practice, we recommend setting TP_size to be less than or equal to **hidden_size / 2048** on H100.
- If this OOMs, then add pipeline parallelism into the mix

Check out our documentation

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

NVIDIA NeMo Framework User Guide

Training & Customization

Title with Link	Description
Quickstart with NeMo 2.0 API	The example showcases a running a simple training loop using NeMo 2.0. It uses the train API from the NeMo Framework LLM collection.
Pre-training & PEFT Quickstart with NeMo Run	An Introduction to running any of the supported NeMo 2.0 Recipes using NeMo-Run. This tutorial takes a pretraining and finetuning recipe and shows how to run it locally, as well as remotely, on a Slurm-based cluster.
Long-Context LLM Training with NeMo Run	Demonstrates using NeMo 2.0 Recipes with NeMo-Run for long-context model training, as well as extending the context length of an existing pretrained model.

```python
def configure_recipe(nodes: int = 1, gpus_per_node: int = 2):
    recipe = llm.nemotron3_4b.pretrain_recipe(
        dir="/checkpoints/nemotron", # Path to store checkpoints
        name="nemotron_pretraining",
        tensor_parallelism=2,
        num_nodes=nodes,
        num_gpus_per_node=gpus_per_node,
        max_steps=100, # Setting a small value for the quickstart
    )

    recipe.trainer.val_check_interval = 100
    return recipe
```

Continuous pre-training

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

```python
from nemo import lightning as nl
import nemo_run as run

recipe.resume = run.Config(
    nl.AutoResume,
    restore_config=run.Config(nl.RestoreConfig, path="nemo://meta-llama/Meta-Llama-3.1-8B"),
    resume_if_exists=True,
)
```

Llama 3.1 8B as base

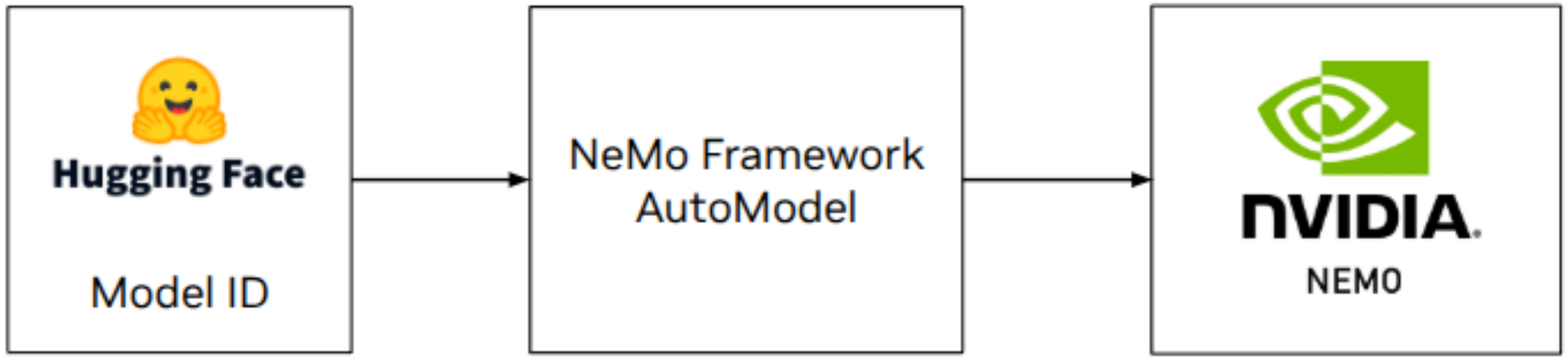
```python
# Use more parallelism parameters to fit the model as needed.
recipe.trainer.strategy.tensor_model_parallel_size = 1
recipe.trainer.strategy.pipeline_model_parallel_size = 1
recipe.trainer.strategy.context_parallel_size = 2
# Modify Data Blend if needed
new_paths = [.3, "path/to/data1", .7, "path/to/data2"]
recipe.data.paths = new_paths
# Or you can directly swap the data module if needed
new_data_module = run.Config(
  llm.PreTrainingDataModule,
  paths = new_paths,
  seq_length = seq_length,
  global_batch_size = gbs,
  micro_batch_size = mbs,
)
# Modify Learning Rate Scheduler if needed
recipe.optim.lr_scheduler.warmup_steps = warmup_steps
recipe.optim.lr_scheduler.min_lr = min_lr
recipe.optim.config.lr = max_lr
```

Define configuration

Introducing AutoModel in NVIDIA NeMo Framework

Enables users to seamlessly fine-tune any Hugging Face model for quick experimentation

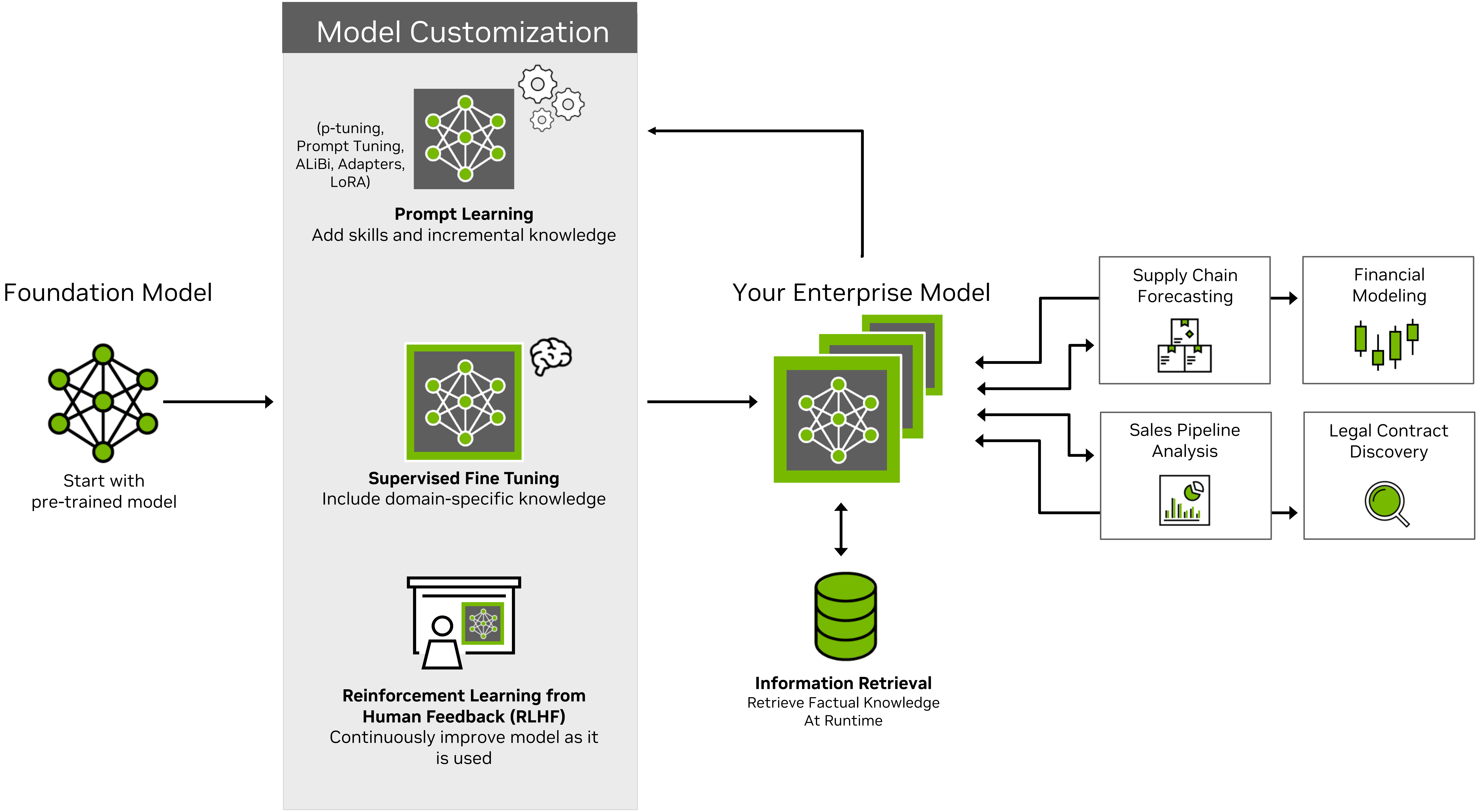
- Model parallelism to enable scaling—currently through Fully-Sharded Data Parallelism 2 (FSDP2) and Distributed Data Parallel (DDP), with Tensor Parallelism (TP) and Context Parallelism (CP) coming soon.
- Enhanced PyTorch performance with JIT compilation.
- Seamless transition to the latest optimal training and post-training recipes powered by Megatron-Core, as they become available.
- Export to vLLM for optimized inference, with NVIDIA TensorRT-LLM export coming soon.



Model customization

Model Customization for Enterprise Ready LLMs

Customization techniques to overcome the challenges of using foundation models



Suite of Model Customization Tools in NeMo

Ways To Customize Large Language Models For Your Use-Cases

	PROMPT ENGINEERING	PROMPT LEARNING	PARAMETER EFFICIENT FINE-TUNING	FINE TUNING
Techniques	• Few-shot learning • Chain-of-thought reasoning • System prompting	• Prompt tuning • P-tuning	• Adapters • LoRA • IA3	• SFT • RLHF
Benefits	• Good results leveraging pre-trained LLMs • Lowest investment • Least expertise	• Better results leveraging pre-trained LLMs • Lower investment • Will not forget old skills	• Best results leveraging pre-trained LLMs • Will not forget old skills	• Best results leveraging pre-trained LLMs • Change all model parameters
Challenges	• Cannot add as many skills or domain specific data to pre-trained LLM	• Less comprehensive ability to change all model parameters	• Medium investment • Takes longer to train • More expertise needed	• May forget old skills • Large investment • Most expertise needed

Deployment

TensorRT-LLM Optimizing LLM Inference

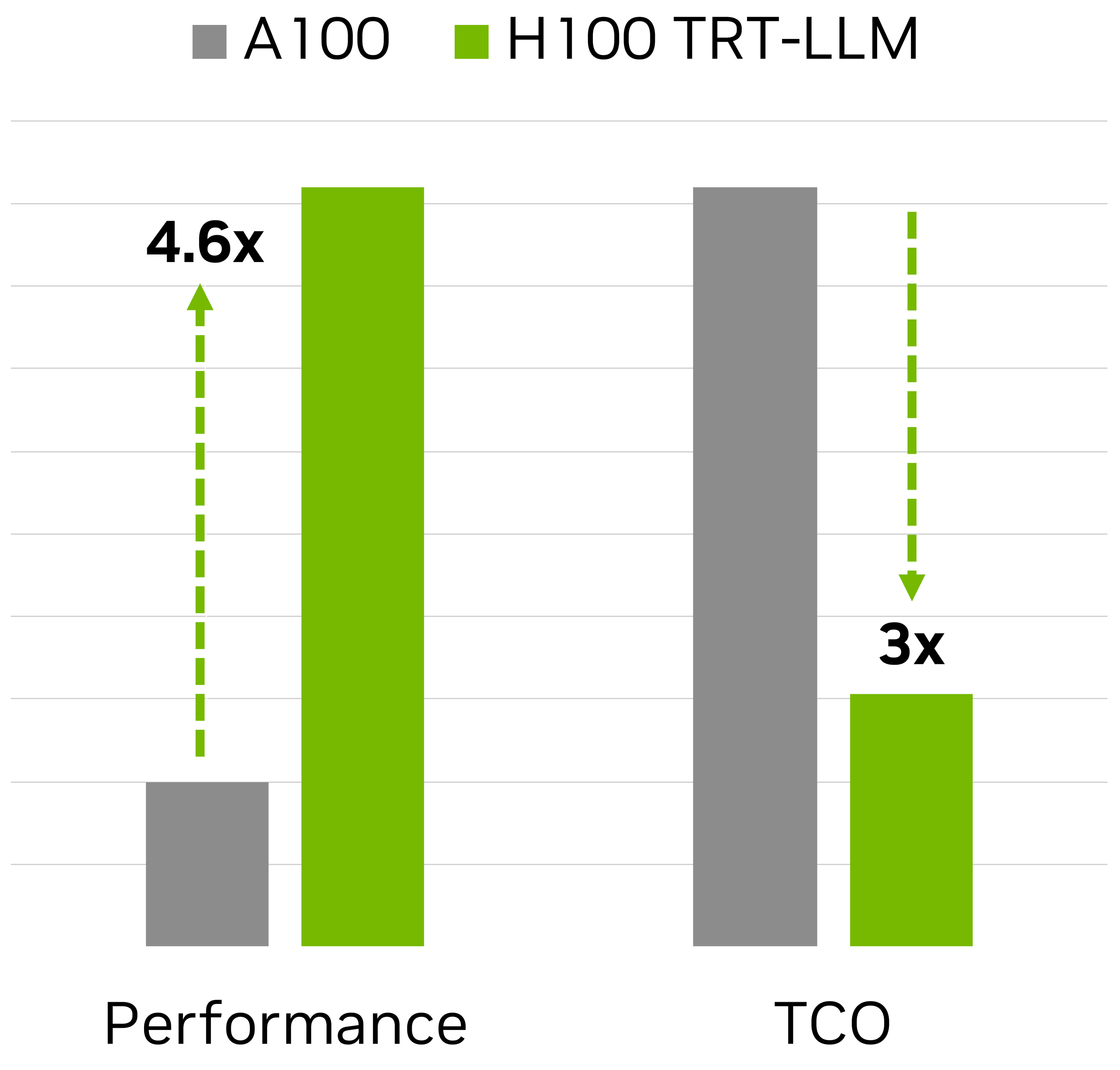
SoTA Performance for Large Language Models for Production Deployments

Challenges: LLM performance is crucial for real-time, cost-effective, production deployments. Rapid evolution in the LLM ecosystem, with new models & techniques released regularly, requires a performant, flexible solution to optimize models.

TensorRT-LLM is an **open-source** library to **optimize inference performance** on the latest **Large Language Models** for NVIDIA GPUs. It is built on FasterTransformer and TensorRT with a simple Python API for defining, optimizing, & executing LLMs for inference in production.

SoTA Performance

Leverage TensorRT compilation & kernels from FasterTransformers, CUTLASS, OAI Triton, ++



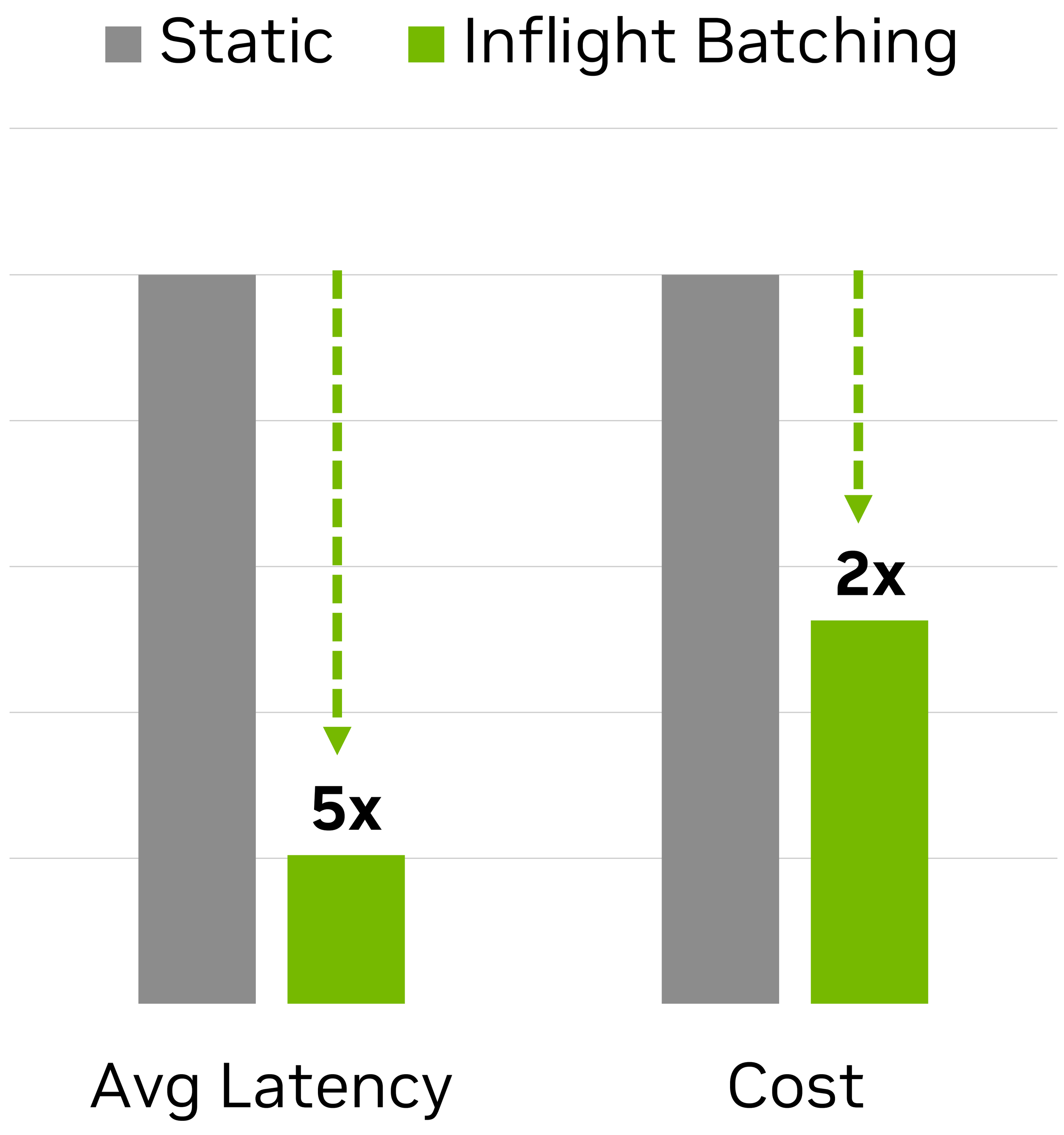
Ease Extension

Add new operators or models in Python to quickly support new LLMs with optimized performance

```
# define a new activation
def silu(input: Tensor) -> Tensor:
    return input * sigmoid(input)

#implement models like in DL FWs
class LlamaModel(Module)
  def __init__(…)
    self.layers = ModuleList([…])

  def forward (…)
    hidden = self.embedding(…)

    for layer in self.layers:
      hidden_states = layer(hidden)

    return hidden
```

LLM Batching with Triton

Maximize throughput and GPU utilization through new scheduling techniques for LLMs



Numbers are preliminary based on internal evaluation on Llama 7B on H100

TensorRT-LLM in the *DL Compiler* Ecosystem

TensorRT-LLM builds on TensorRT Compilation

TensorRT-LLM

LLM specific optimizations:

- FP8 quantization
- KV Caching
- Multi-GPU, Muti-Node
- Custom MHA optimizations
- Paged KV Cache (Attention)
- *etc…*

TensorRT

General Purpose Compiler

- Optimized GEMMs & general kernels
- Kernel Fusion
- Auto Tuning
- Memory Optimizations
- Multi-stream execution

NeMo Guardrails

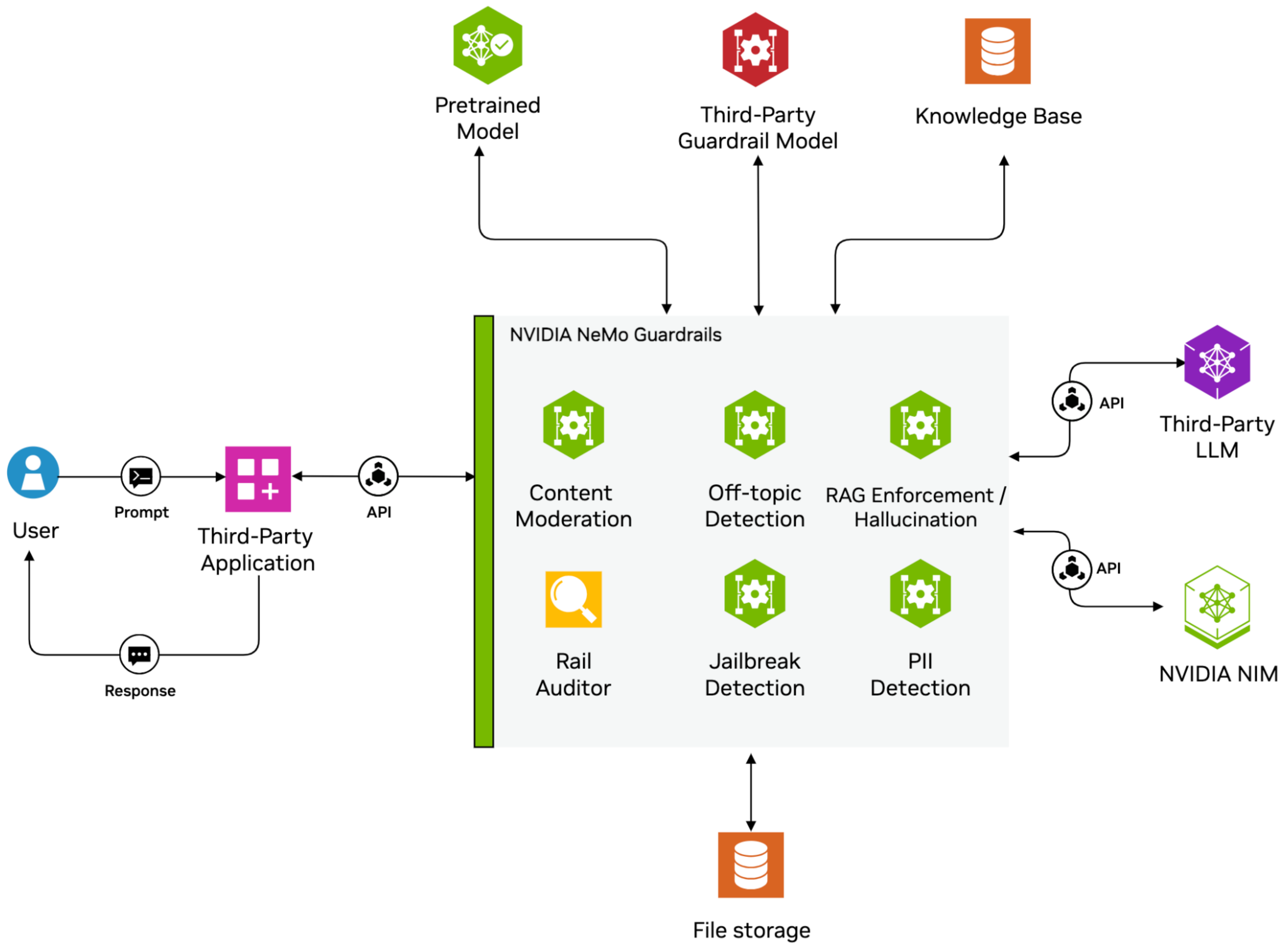
Scalable rail orchestration for safeguarding enterprise generative AI

Efficiently orchestrate multiple rails across applications with a modular framework

Use smart defaults or customize and extend rails leveraging a robust 3rd party ecosystem

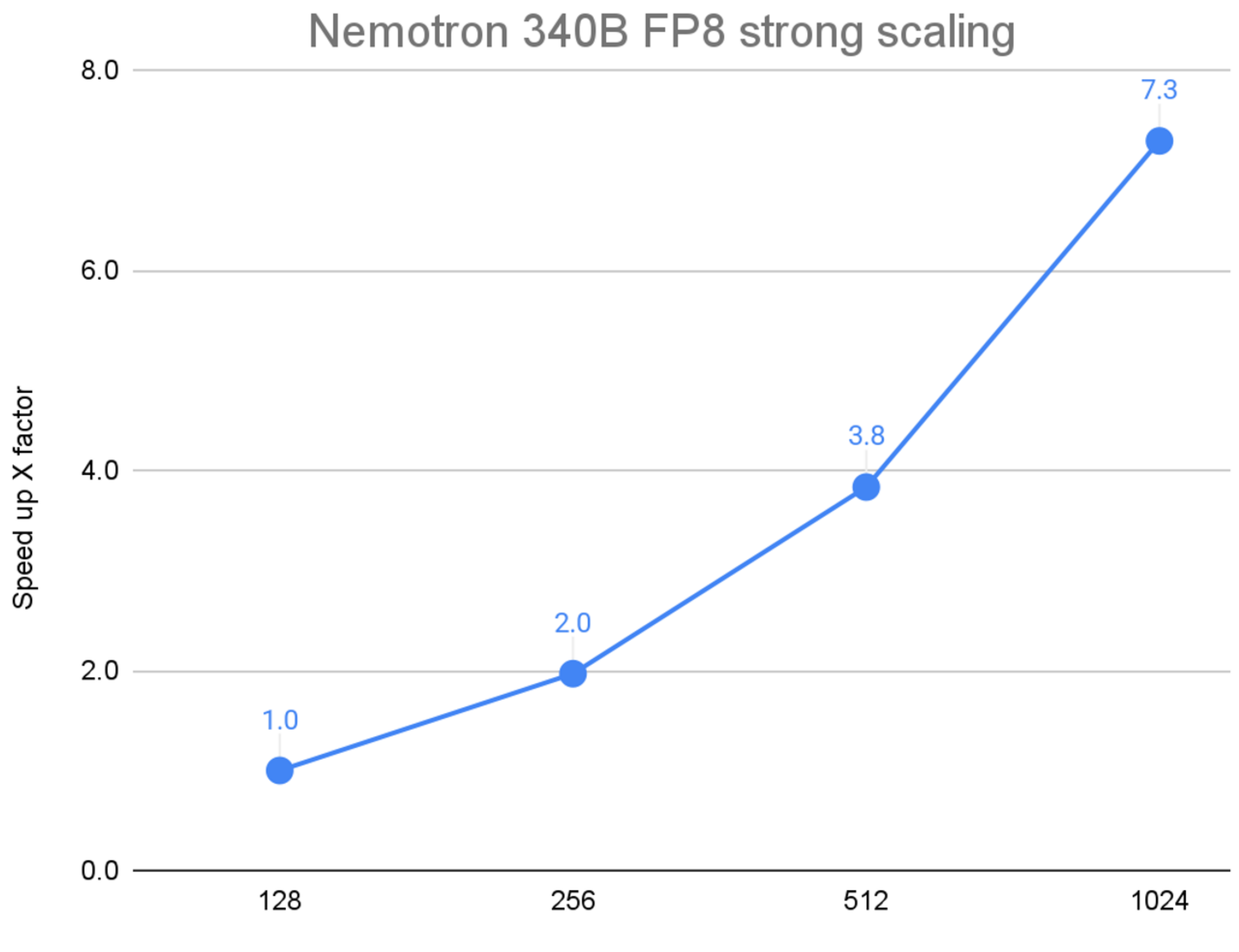
Continuously improve rail and application effectiveness with built-in auditing and analytics

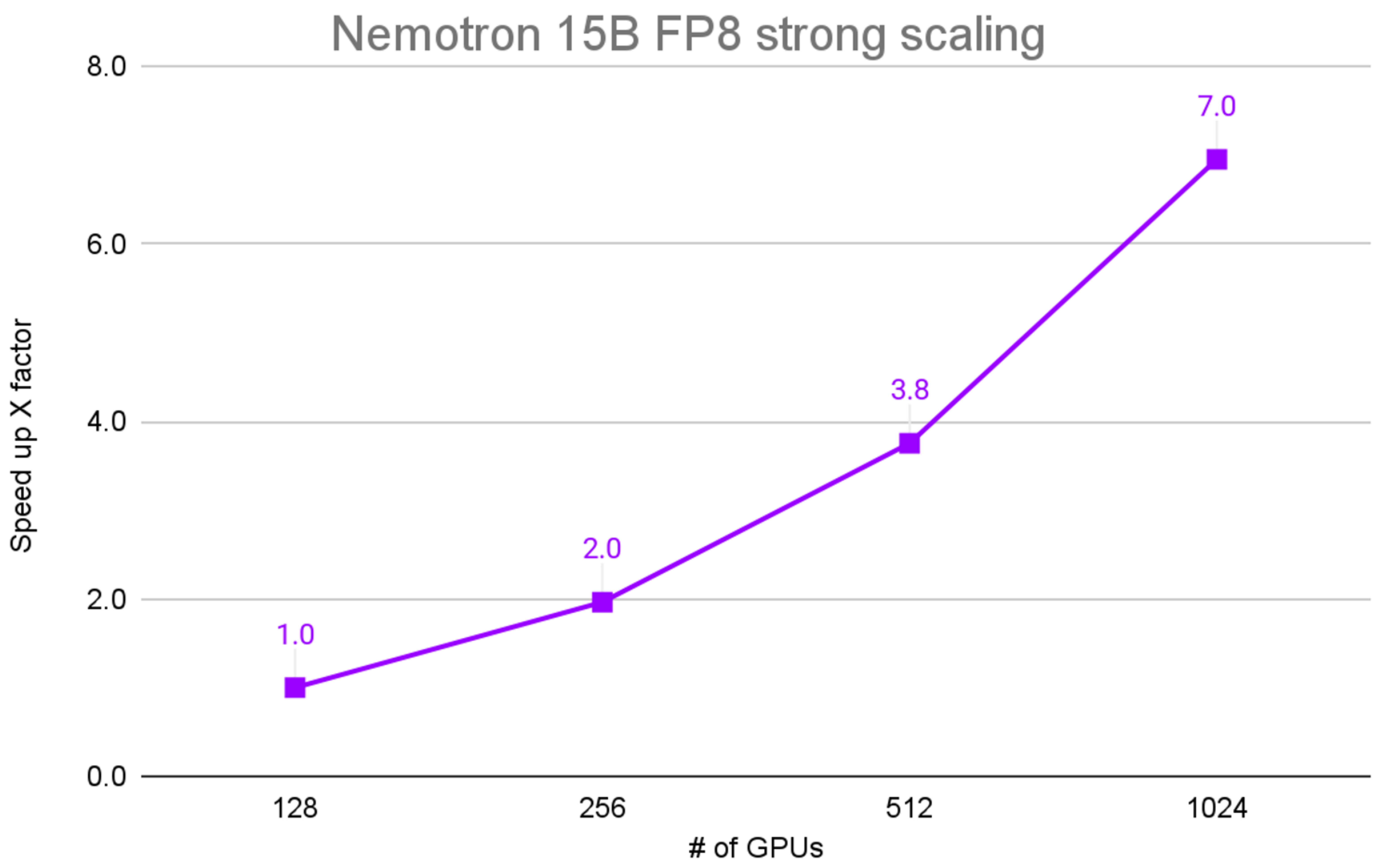
Leverage open-source and portable, enterprise grade microservices ecosystem



Nemo FW Strong Scaling

More than 90% scaling efficiency





Micro-scaling Data Formats for Deep Learning

GPT training loss curve, using MXFP6_E3M2 for weights, activations, and gradients

Model	FP32	MXFP6	
		E2M3	E3M2
GPT-20M	3.98	4.02	4.01
GPT-150M	3.30	3.33	3.32
GPT-300M	3.11	3.13	3.12
GPT-1.5B	2.74	2.75	2.75

Table 7: Language model loss for training from scratch using MXFP6_E3M2 for weights, activations, and gradients.

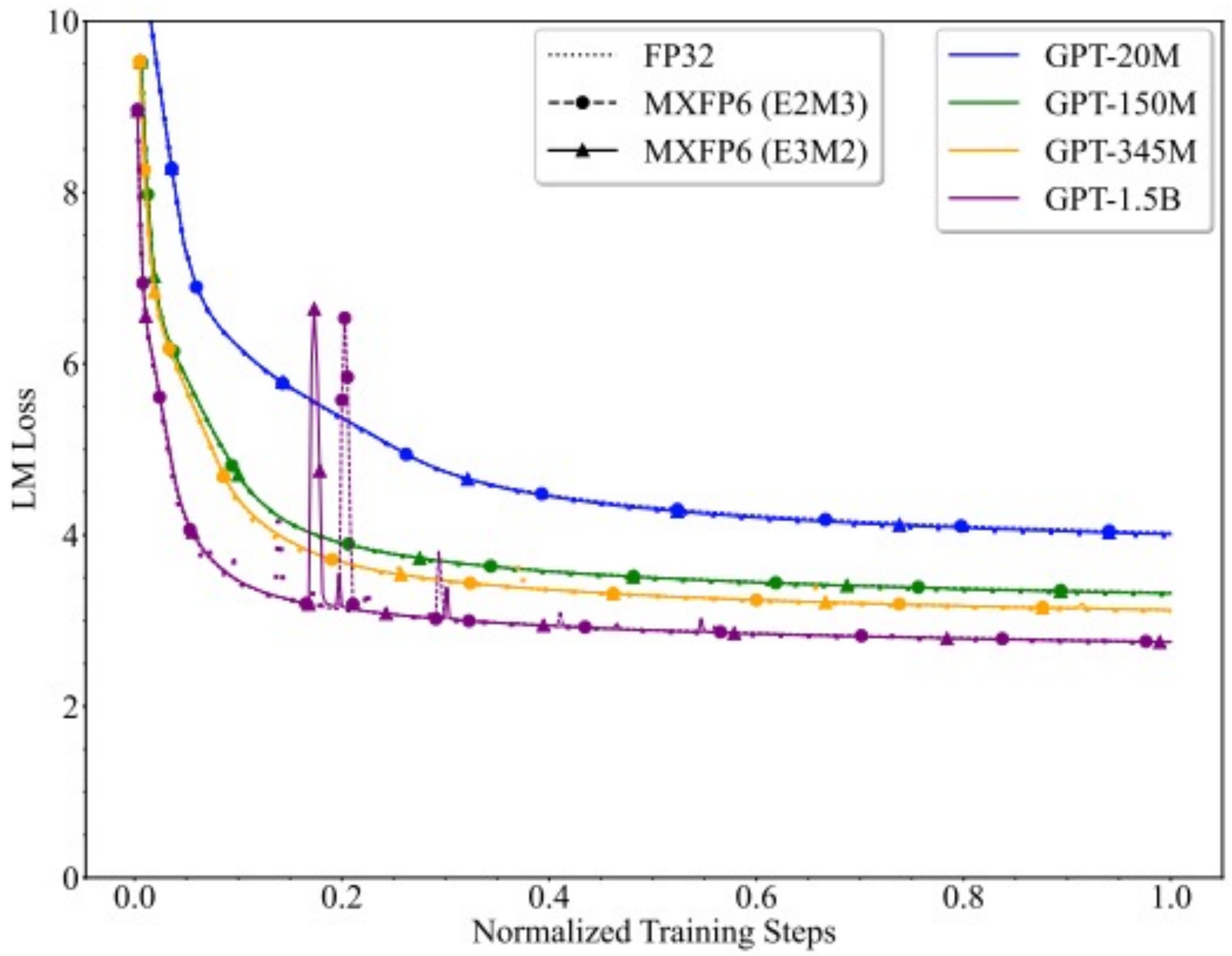
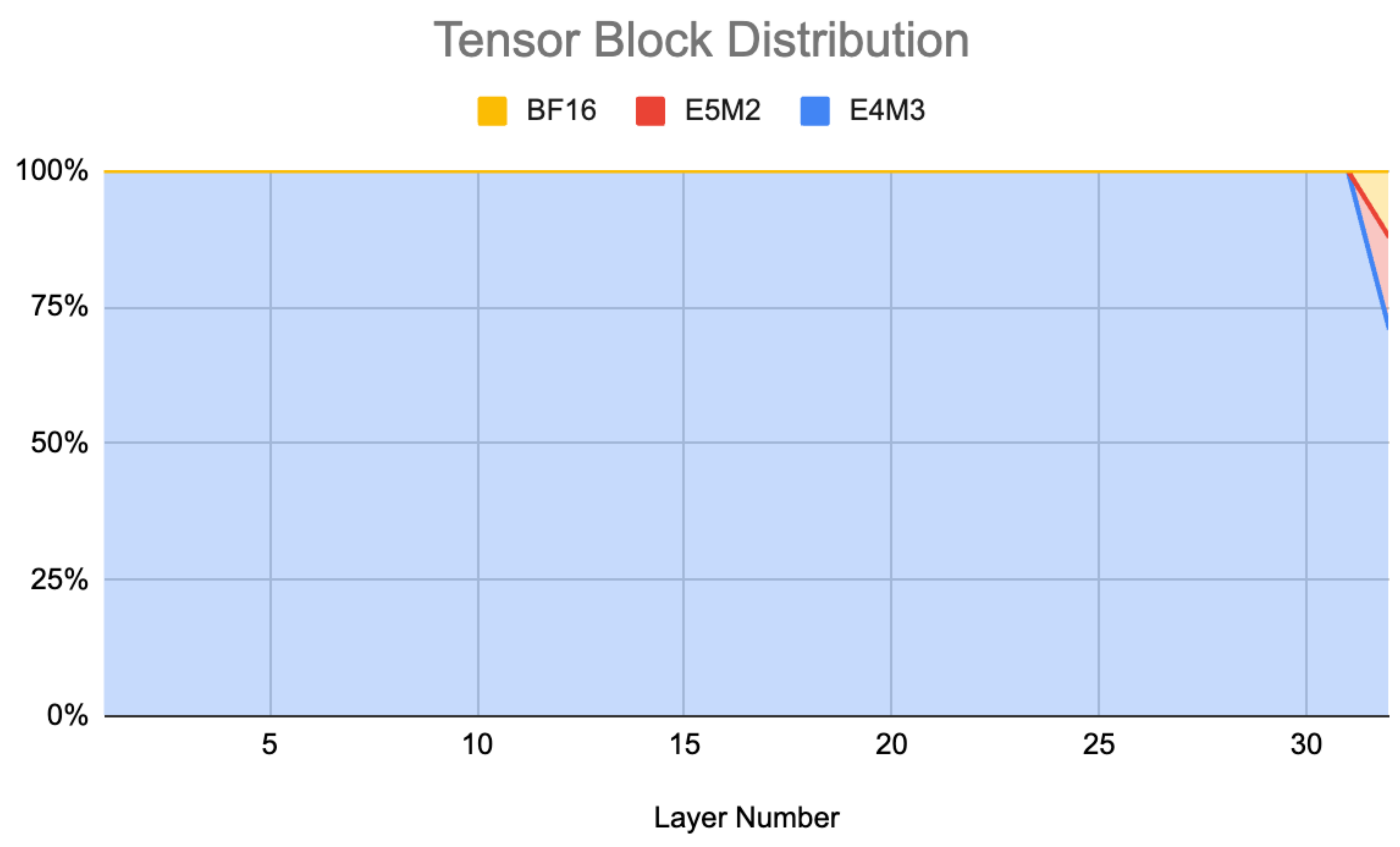
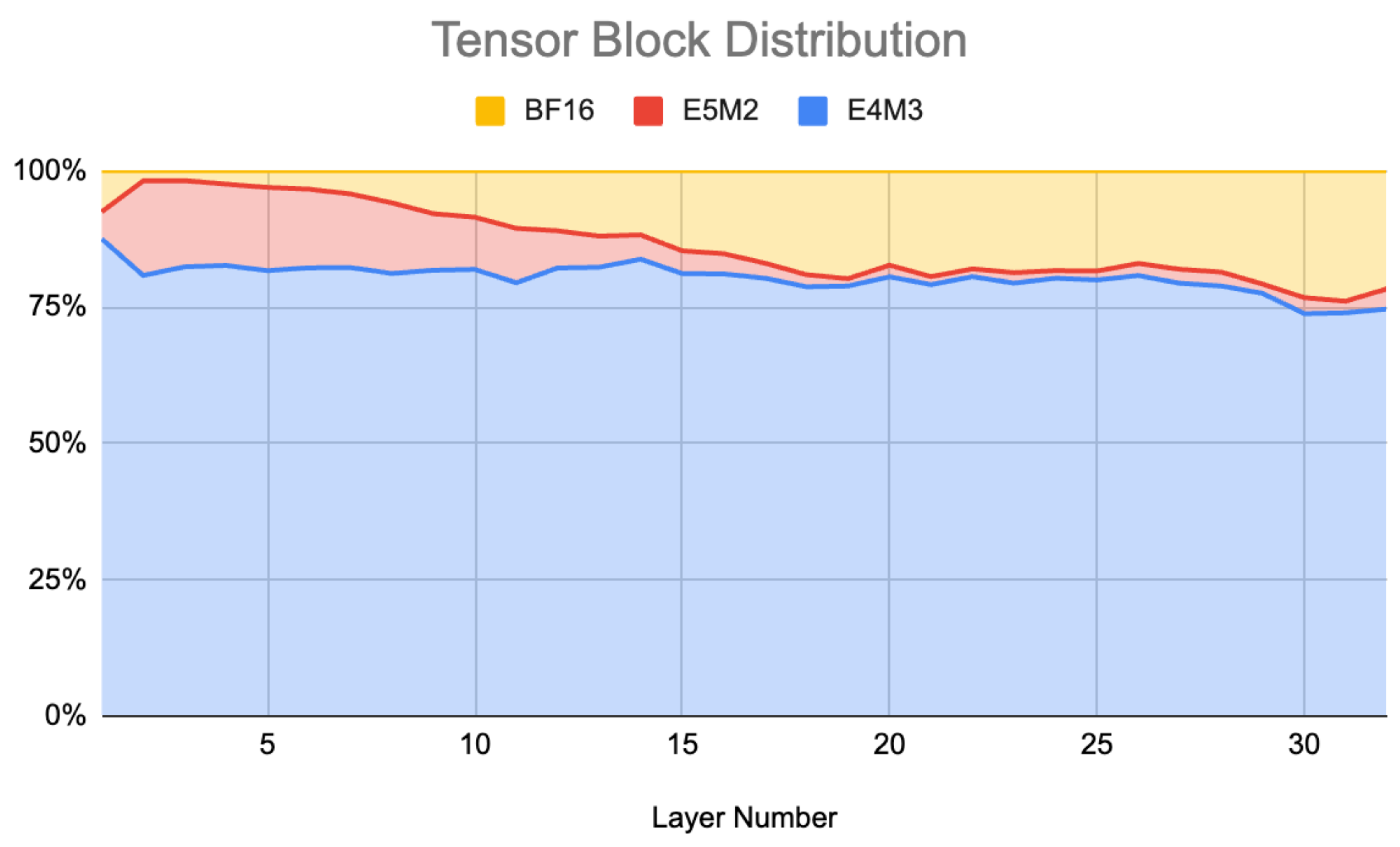


Figure 3: GPT training loss curve, using MXFP6_E3M2 for weights, activations, and gradients.

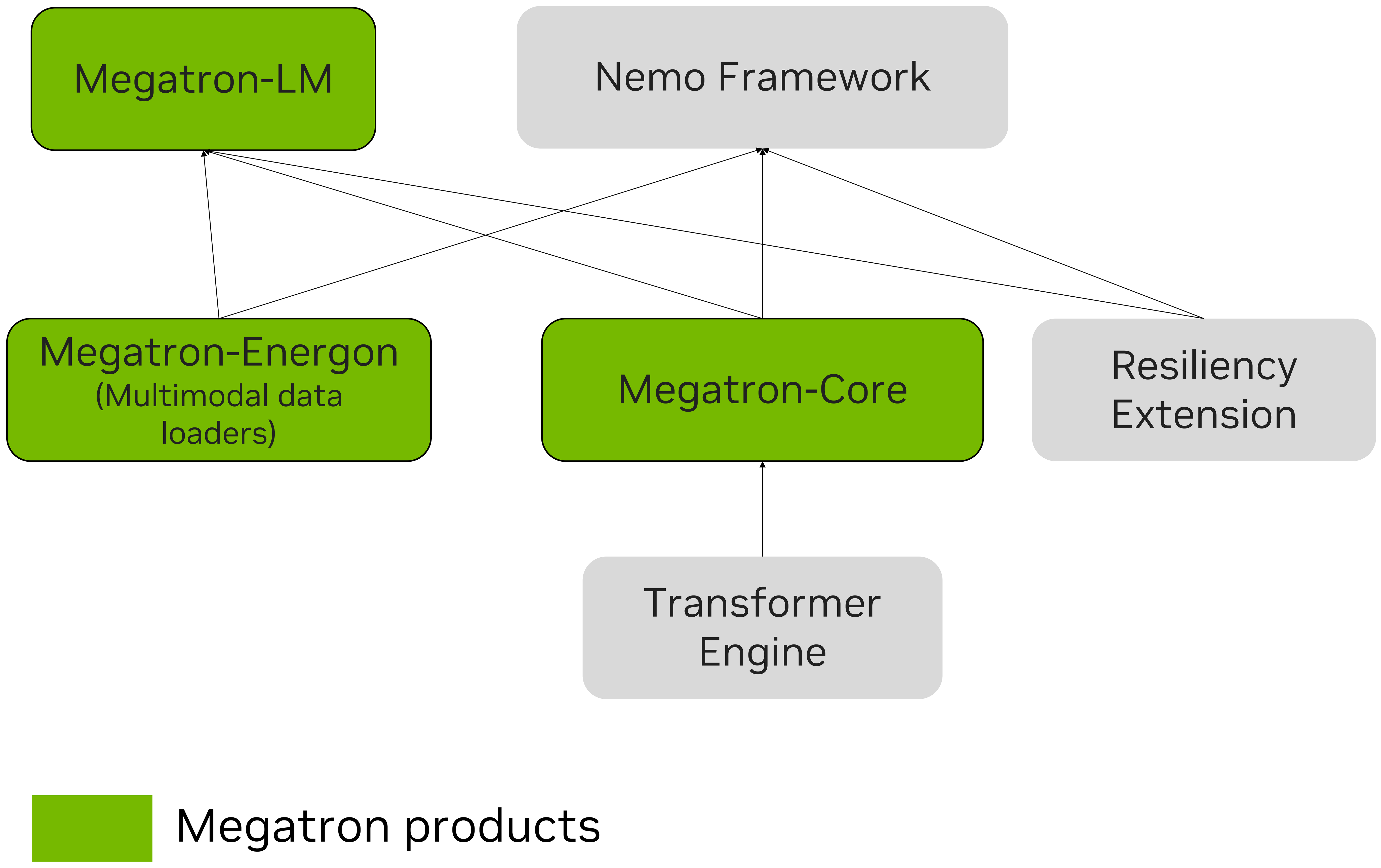
"Microscaling Data Formats for Deep Learning" Darvish Rouhani B., Zhao R., More A., Hall M., Khodamoradi A., Deng S., Choudhary D., et al., 2023, arXiv, arXiv:2310.10537. doi:10.48550/arXiv.2310.10537

Tensor block distribution





Recap of NVIDIA's GenAI Training offerings



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 reference training framework for using Megatron-Core to build your own LLM framework.

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

Megatron-Energon: Multimodal data loaders for Megatron-Core.

Resiliency Extension: A library for resiliency features for PyTorch-based training

Transformer Engine: **Accelerated kernels and FP8 mixed precision.** Specific acceleration library, including FP8 on Hopper and Blackwell.

NOT COMPREHENSIVE

Key benefits of Megatron-Core

Performance at Scale

Parallelism Techniques
- Data/Tensor/Pipeline/Sequence/Context Parallel
- Virtual PP for improved performance
- Hierarchical Context Parallelism
- EP for MoE models
- Enc-Dec Parallelism (e.g. T5)

Memory Saving Techniques
- Selective Activation Recompute
- Offloading (Activation, weight, optimizer state)
- Attention: FAv2, FA-cuDNN, GQA, MQA, SWA
- SSM, MLA

Distributed Optimizers
- Zero-1 (fully implemented), Zero-2/3 (WIP)
- Precision aware optimizers (BF16, FP8)
- Custom FSDP - 15% speedup
- CPU offloading with hybrid device optimizers
- Overlapping CPU optimizer with data transfers

Additional Performance Features
- FP8 via Transformer Engine
- MLPerf Optimizations
- TRT-LLM based Inference
- Gradient accumulation fusion
- Pipeline comm optimizations
- Microbatch grouping for virtual pipeline stages

Mixture-of-Expert (MoE)

- Token drop and dropless approaches
- FP8 support
- Expert MP with configurable expert TP/MP
- Expert DP
- Grouped GEMM for MoE layers
- All-to-all token dispatche

Customizable Building Blocks

Optimized transformer blocks with modular and composable APIs

Canonical architectures:
- Decoder (GPT), Encoder (BERT), Enc-Dec (T5)
- RAG (RETRO), MoE, ViT/DiT
- Hybrid (Mamba SSM)

Releases with SOTA features

Multimodal Training
- Sequence and context parallelism for LLaVA
- Variable sequence lengths across microbatches

Blackwell support
- QAT for FP4 inference
- MXFP8 recipe support

Leverage Cutting-edge Research
- Stay at the forefront of distributed training
- SSM-based hybrid models

Scalability & Training Resiliency

- Fast distributed checkpointing
- Integration with NVIDIA Resiliency Extension (WIP)
  - Hang, Straggler and SDC detection
  - In-job, In-process restart
  - Hierarchical Checkpointing
- Configurable distributed timeout
- NCCL comm configuration
- Gloo process groups for CPU operations

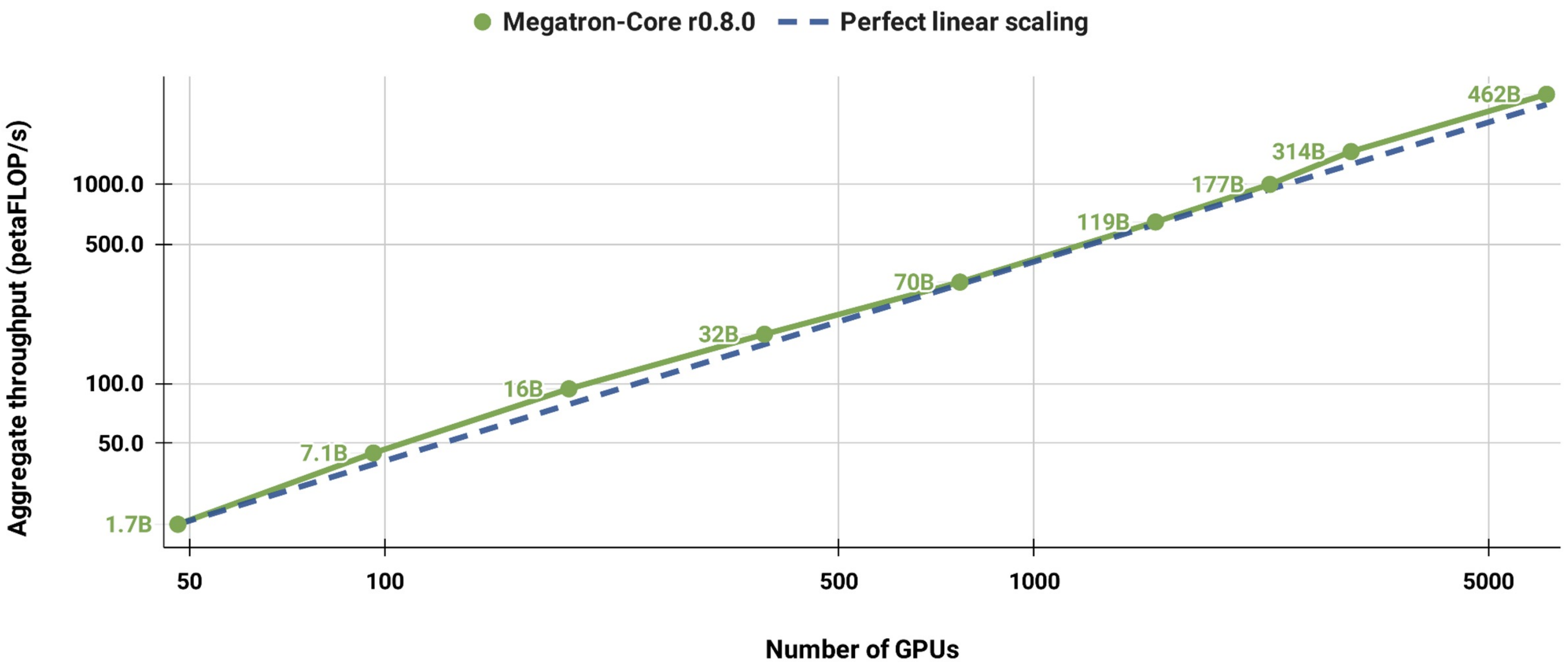
Software Choices for LLM Developers

	Persona 1: Research in LLM Framework & Models	**Persona 2: Develop your own LLM and ConvAI models from scratch**	**Persona 3: Deploy and operationalize SOTA models for production**
	Research Framework Next Gen LLM Model M-LM (reference)	Model Development	Integration
			Nemo Service (cloud or on-prem)
		Nemo FW	Nemo FW
	Megatron-Core	Megatron-Core	Megatron-Core
	PyTorch	PyTorch+Lightning	PyTorch+Lightning
Value	Core optimizations/kernels for LLM training at scale with latest updates from NV.	End-to-end training open source framework. Train from scratch w/guaranteed convergence on a specific set of SotA model architectures and data types Fine-tuning customization techniques Optimized conversion to TRT	Deploy and tune LLM to production: Fine tune, containerize, microservices, enterprise integration, RAG Pre-built containers and services to operate infrastructure and MLOps integration
Challenges	Requires developer to have their own framework implementation. Only for experts in the field of distributed AI training software.	Expects AI practitioner skills (training scripts, job). User provides infra and operates infra. Traditional framework only, no automation/services/MLops	Supports only pre-trained community and NV specific models. No train from scratch or novel model architectures Microservice interfaces, not a framework

NVIDIA

World-Leading Training Speed and Scalability

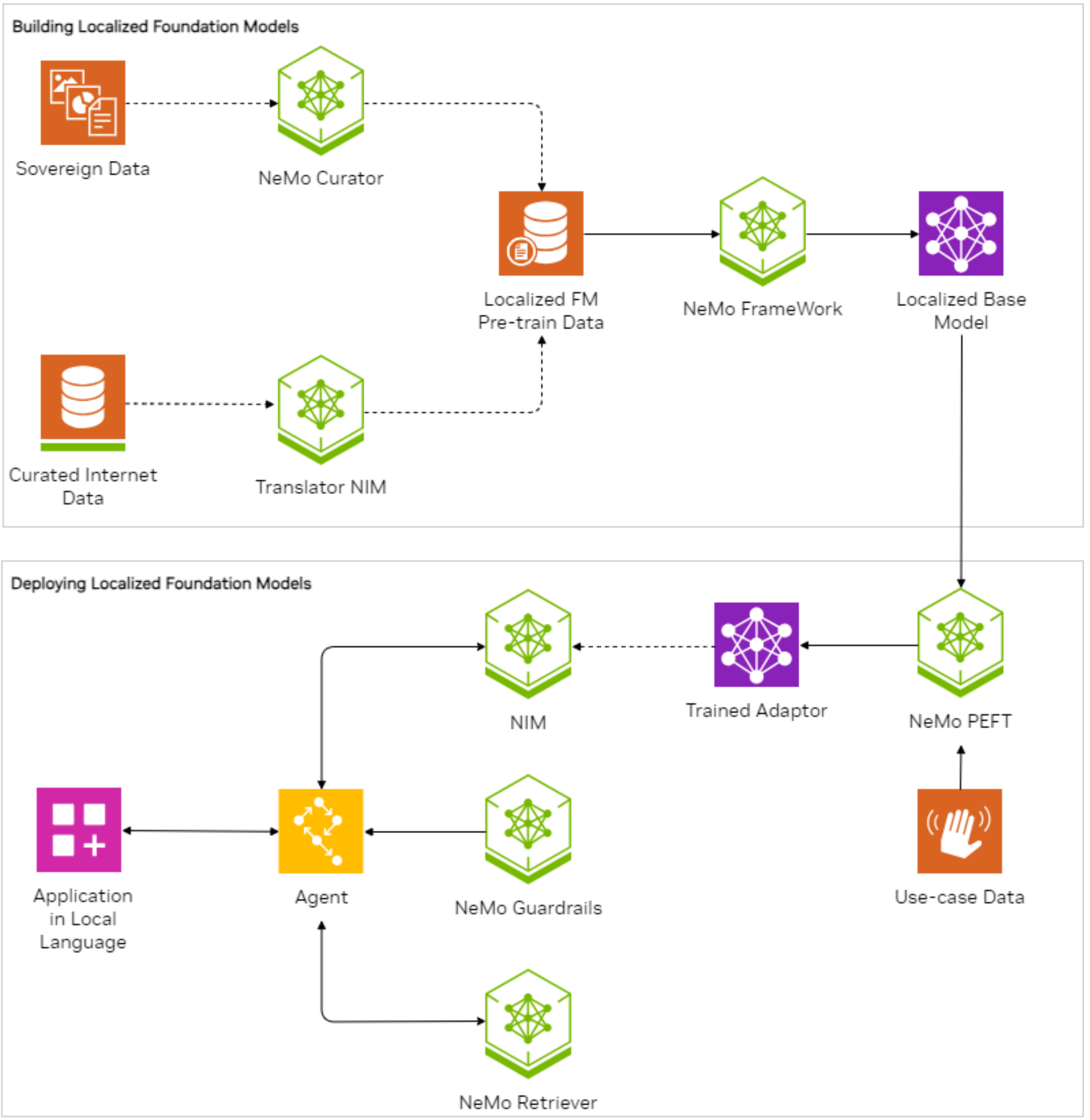
- Weak scaling experiments with GPT models ranging from 2B to 462B parameters
- Megatron-Core demonstrates **superlinear scaling up to 6144 H100 GPUs**



Benchmark details

Canonical workflow for building and deploying localized FMs

Train FM from scratch on local language data and fine-tune for individual use-cases



- **EuroLLM**
  - Collection of Sovereign multi-lingual LLMs with a focus on EU languages
  - Model: 1.7B, 9B (available on HF), 22B (WIP) and 9B NIMification (WIP)
  - NV usage: Megatron LM, FW, NIM
  - Platform: 400xH100s (MareNostrum5).
- **ETH Zurich**
  - Building a foundation model for Swiss German
  - In development: 70B pre-trained with FP8 precision
  - NV Usage: Megatron LM
  - Platform: CSCS ALPS (Grace Hopper)
- **BritLLM**
  - Goal to produce training and evaluation data, freely available models aligned with UK interests
  - Model: 3B released, developing larger and multi-lingual
  - NV Usage: Megatron LM
  - Platform: Isambard AI (Grace Hopper)
- **Salamandra / ALIA**
  - Collection of Sovereign multi-lingual LLMs with a focus on Spain’s official languages
  - Model: 2B, 7B, 40B all available on HuggingFace, 7B NIMification (WIP)
  - NV Usage : Nemo Framework (1.x), NIM
  - Platform : 1000+ H100s (MareNostrum5)

NVIDIA