MVAPICH
MPI, PGAS and Hybrid MPI+PGAS Library

Follow us on
https://twitter.com/mvapich

HiDL
High-Performance Deep Learning

Vendor-neutral and MPI-driven Solutions for Training and Inference with the HPC-AI Stack

Tutorial at MUG '26

Presented by

Nawras Alnaasan

The Ohio State University

alnaasan.1@osu.edu

https://www.linkedin.com/in/nawras-alnaasan

Jinghan Yao

The Ohio State University

yao.877@osu.edu

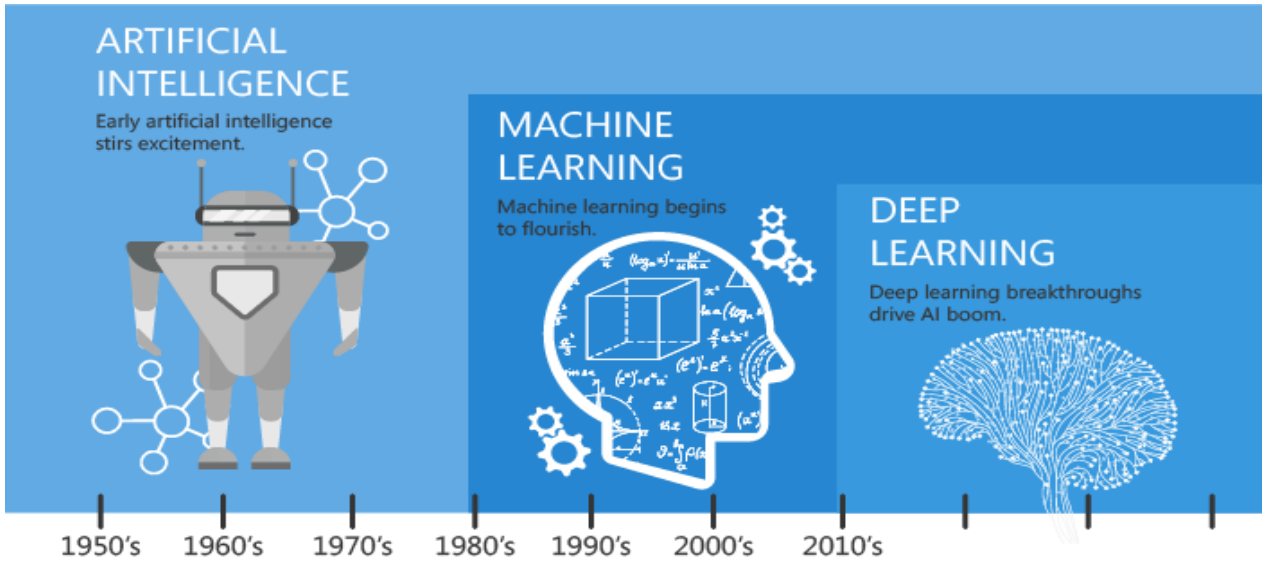
http://nowlab.cse.ohio-state.edu/member/yao.877/

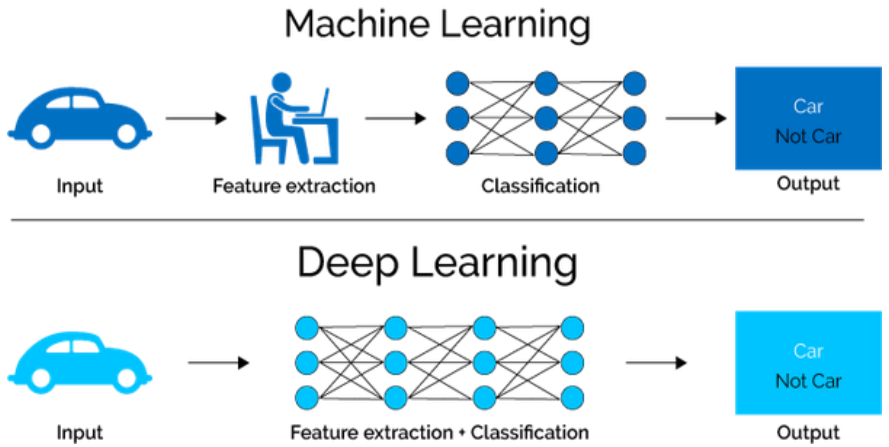
Outline

- **Introduction**
- Training
  - Deep Neural Network Training
  - Distributed Deep Learning
    - Data Parallelism
    - Sharded Data Parallelism
    - Communication in Parallelism Strategies
- Inference
  - Introduction to Inference
  - Advanced Inference Solutions
- The HPC-Accelerated AI (HPC-AI)
- Conclusion

What is Machine Learning and Deep Learning?

- Machine Learning (ML)
  - "the study of computer algorithms to improve automatically through experience and use of data"
- Deep Learning (DL) – a subset of ML
  - Uses Deep Neural Networks (DNNs)
  - **Perhaps, the most revolutionary subset!**
- Based on learning data representation
- DNN Examples: Convolutional Neural Networks, Recurrent Neural Networks, Hybrid Networks
- Data Scientist or Developer Perspective for using DNNs
  1. Identify DL as solution to a problem
  2. Determine Data Set
  3. Select Deep Learning Algorithm to Use
  4. Use a large data set to train an algorithm



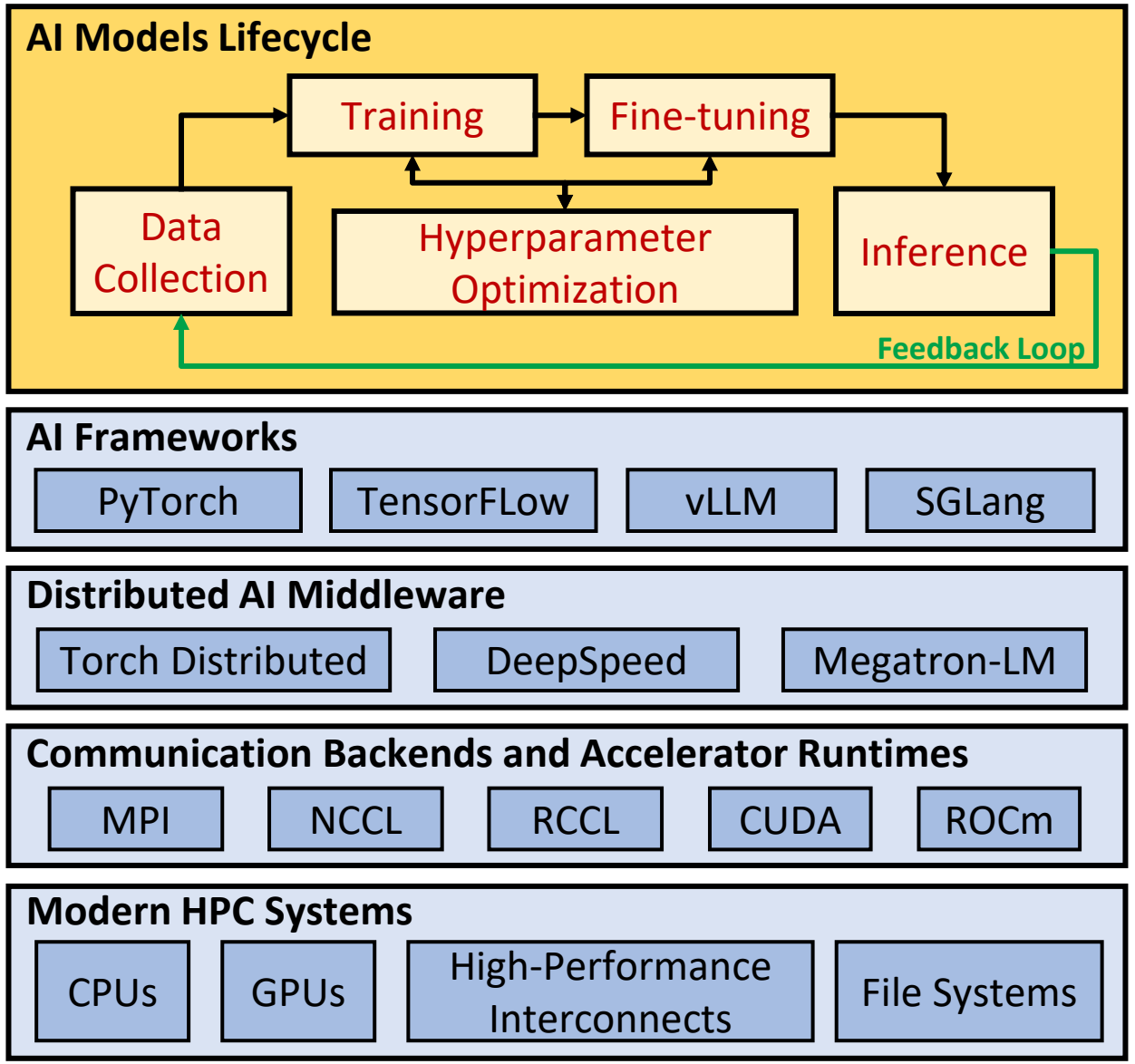


Courtesy: https://hackernoon.com/difference-between-artificial-intelligence-machine-learning-and-deep-learning-1pcv3zeg, https://blog.dataiku.com/ai-vs.-machine-learning-vs.-deep-learning, https://en.wikipedia.org/wiki/Machine_learning

Overview: The Lifecycle of AI Models

AI model lifecycle is an iterative process involving:

- **Training:** Learning model parameters using collected data to identify patterns and make predictions.
- **Fine-Tuning:** Adapting pretrained models to downstream tasks using a specialized datasets.
- **Inference:** Deploying trained models to serve predictions on real-world data.

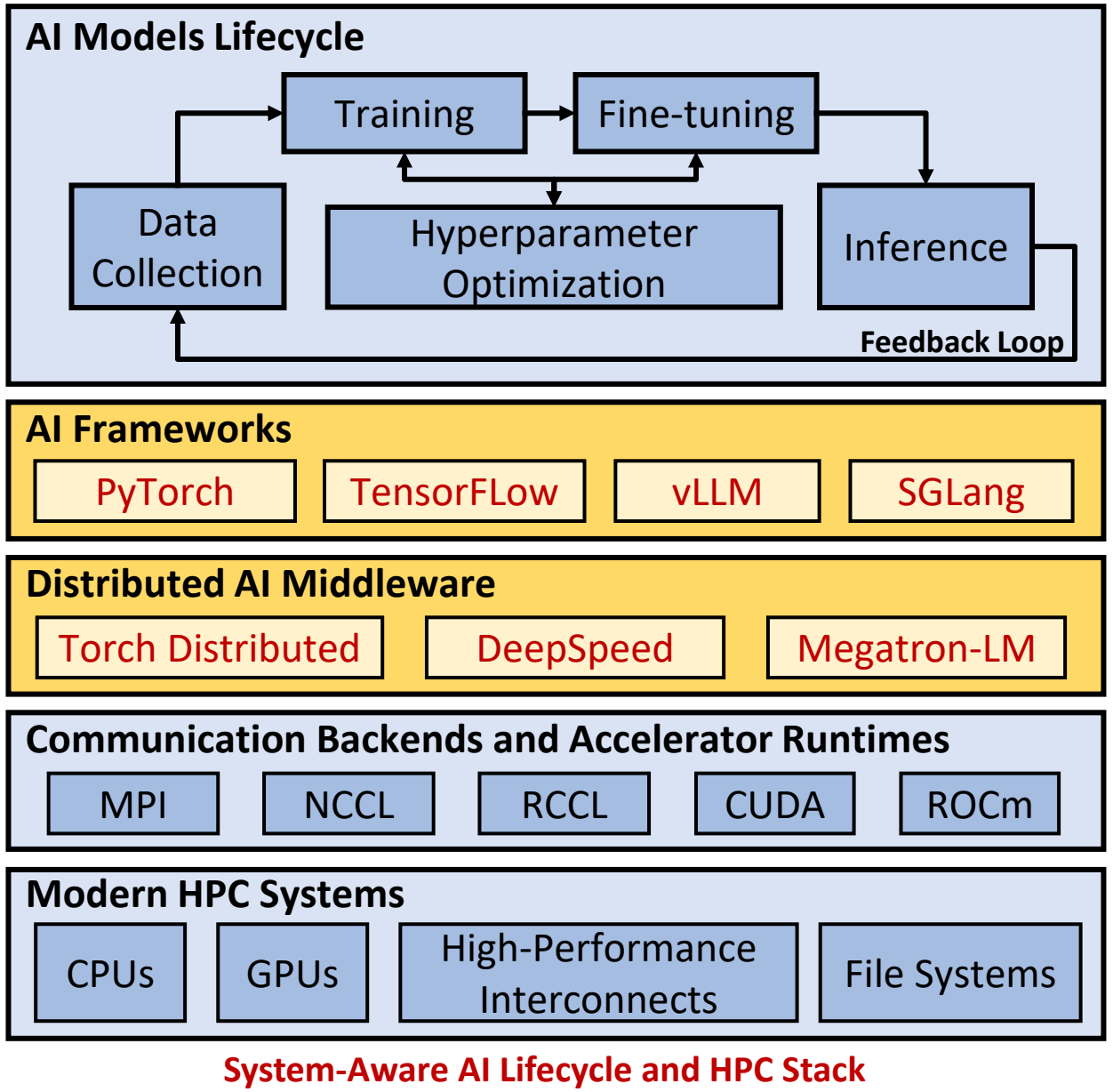


System-Aware AI Lifecycle and HPC Stack

Overview: AI Frameworks and Distributed Middleware

- **Main objectives of AI frameworks:**
  - Hide complex mathematics
  - Allow users to focus on Developing AI models
- **Support for Parallelism:**
  - We have saturated the peak potential of current-generation architectures
    - A single GPU or a many-core CPU is not enough!
  - Distributed AI Middleware implements various parallelism strategies to support large scale model training.

PYTORCH TensorFlow vLLM SGL



System-Aware AI Lifecycle and HPC Stack

Overview: Communication Backends and Accelerator Runtimes

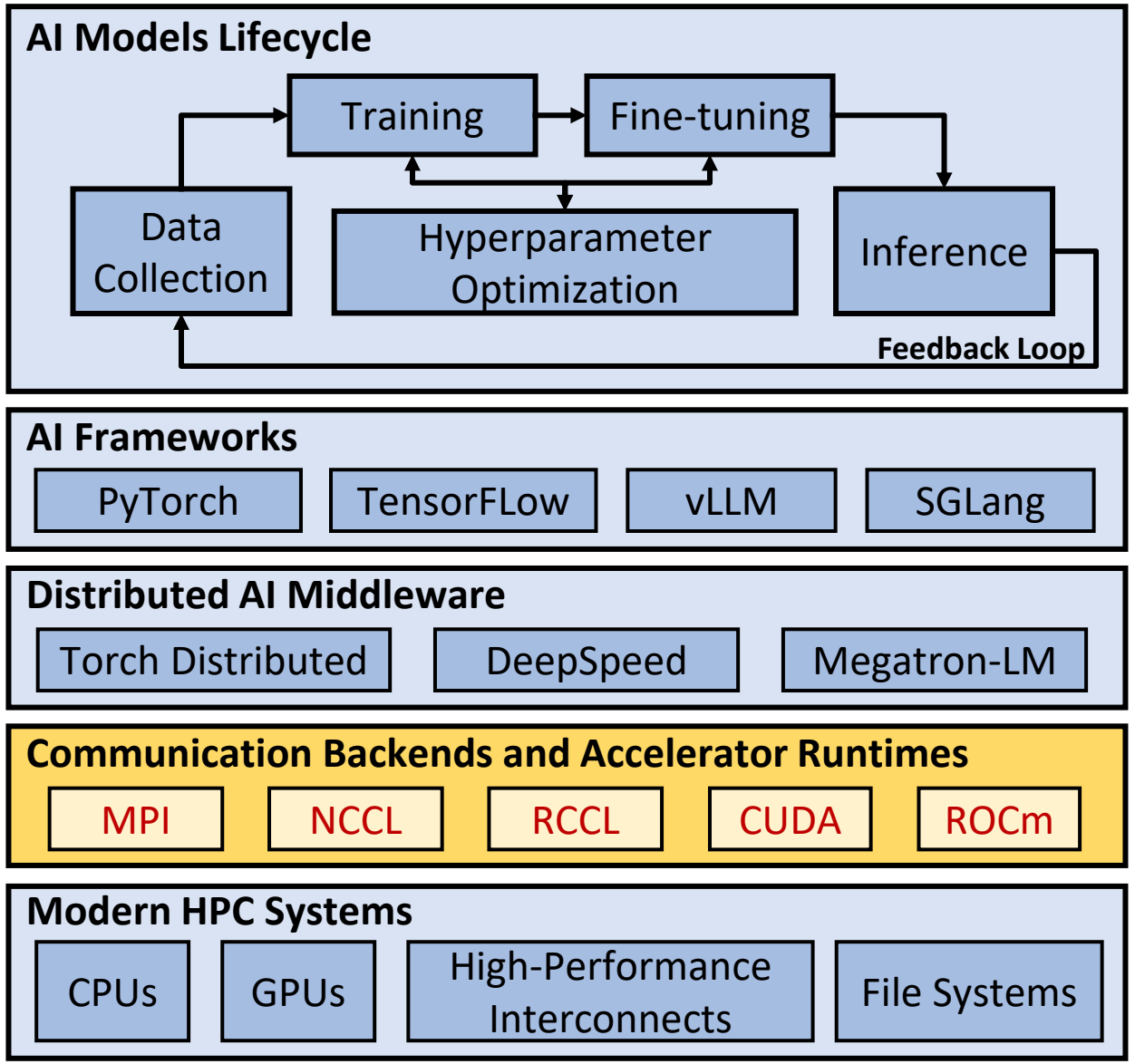
This layer provides the low-level communication primitives and control of accelerator hardware.

- **Communication Backends (MPI, NCCL, RCCL):**

  High-performance communication libraries implement collective operations (e.g., AllReduce, Broadcast) and point-to-point messaging, optimized for HPC interconnects and multi-GPU scaling.

- **Accelerator Runtimes (CUDA, ROCm):**

  Programming models and execution runtimes provide direct access to GPU hardware, including kernel execution, memory management, and device-level parallelism.

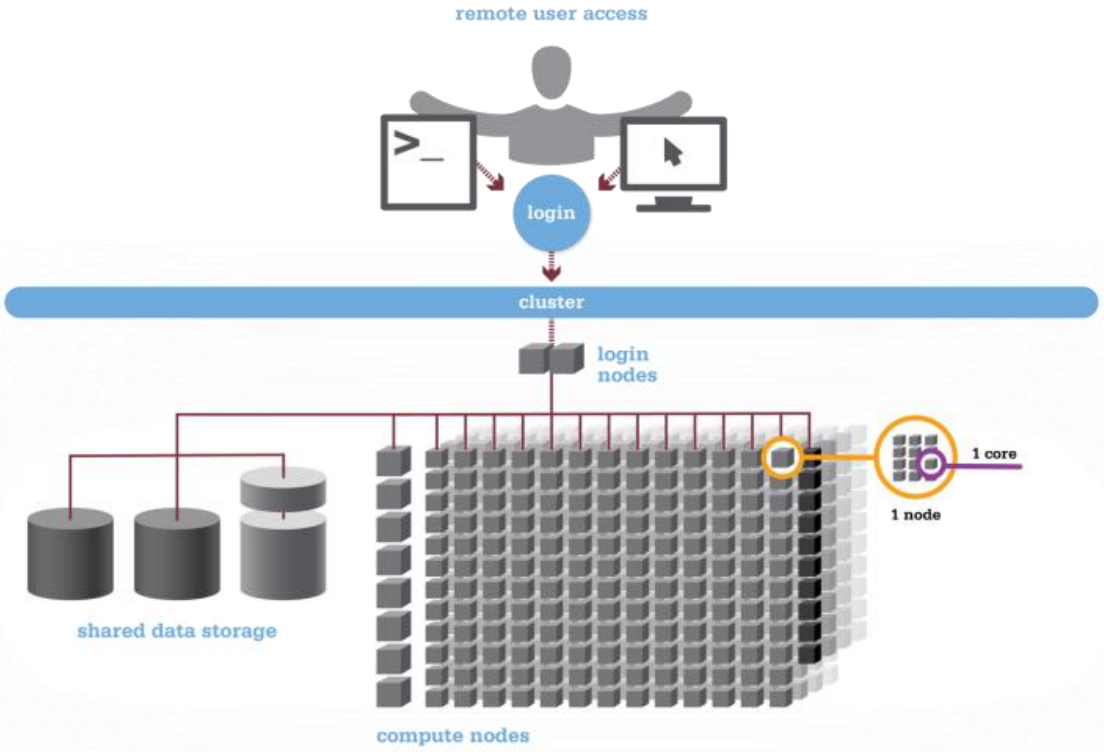


System-Aware AI Lifecycle and HPC Stack

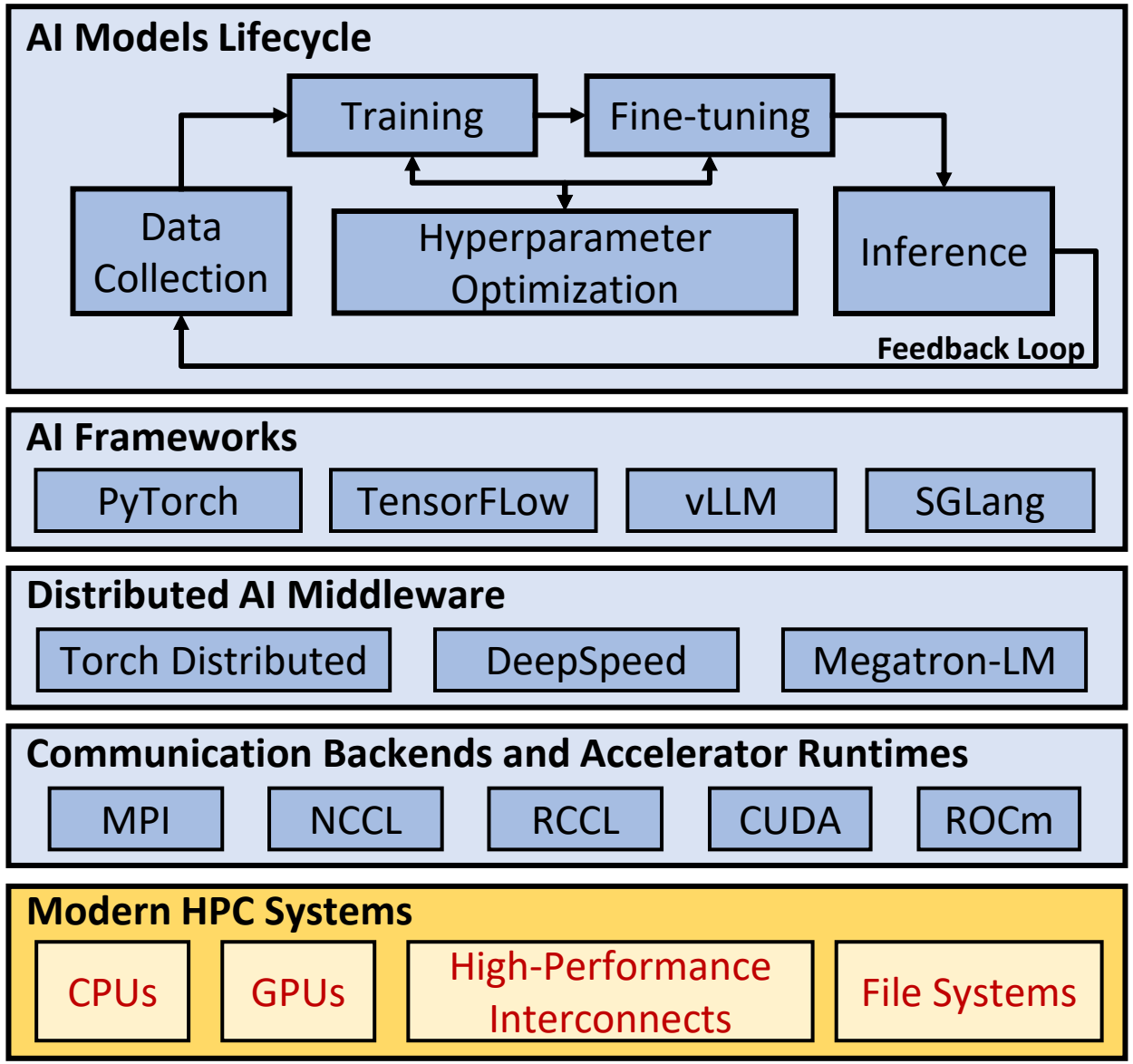
Overview: DL and High-Performance Architectures

HPC infrastructure consists of:

- Compute and Memory Architecture (CPUs, GPUs, HBM, DRAM)
- Storage and Interconnect (NVMe, Parallel File Systems, NVLink, InfiniBand)



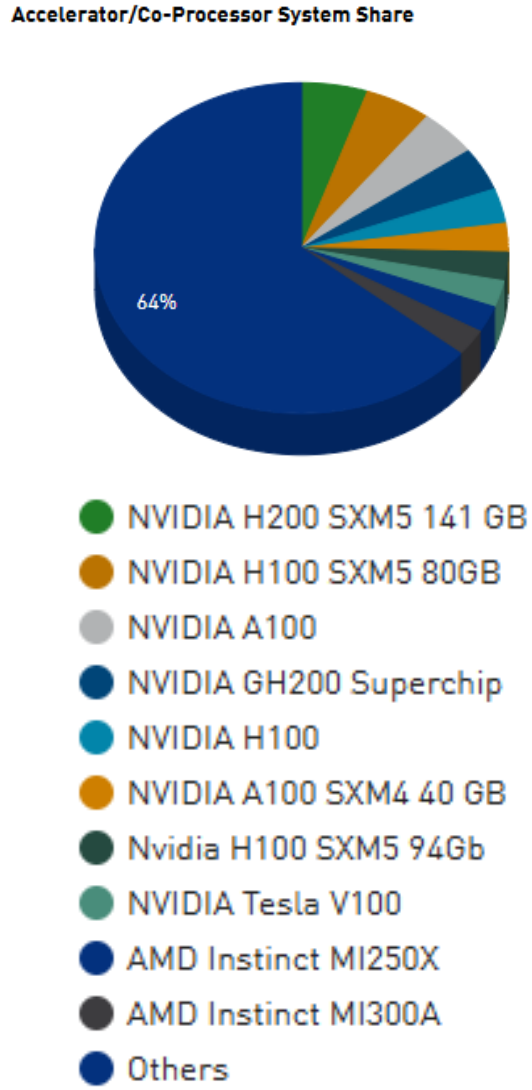
Courtesy: https://www.osc.edu/resources/getting_started/hpc_basics



System-Aware AI Lifecycle and HPC Stack

DL and High-Performance Architectures

- NVIDIA GPUs are the main driving force for faster training of DL models
- However, High Performance Architectures for DL and HPC are evolving
  - 272/500 Top HPC systems are using accelerator/co-processor (June '26)
  - DGX-1 (Pascal), DGX-2 (Volta), DGX A100, DGX H100, HGX A100, HGX H100
    - Dedicated DL supercomputers
    - NVIDIA Eos – An Exaflop AI Supercomputer using DGX H100 (Announced)
  - AMD Instinct MI300A GPUs power El Capitan – the #2 Top500.
  - AMD EPYC (Rome/Milan) CPUs have 64 cores/socket (Frontier – #3 on Top500)
  - Sapphire Rapids Xeon CPUs have 52 cores/socket (Aurora – #4 on Top500)
  - Domain Specific Accelerators for DNNs are also emerging



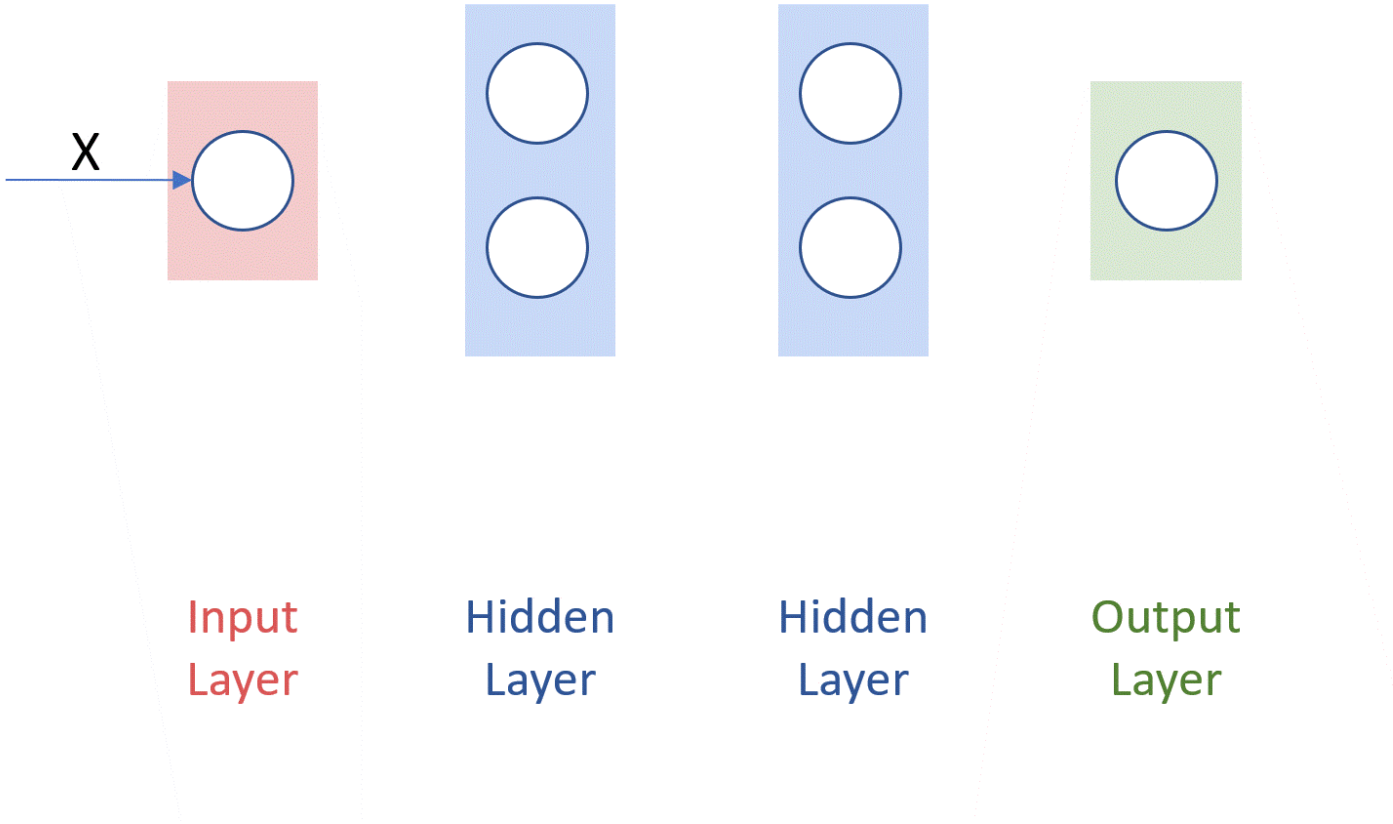
Accelerator/CP Performance Share www.top500.org

Outline

- Introduction
- **Training**
  - **Deep Neural Network Training**
  - Distributed Deep Learning
    - Data Parallelism
    - Sharded Data Parallelism
    - Communication in Parallelism Strategies
- Inference
  - Introduction to Inference
  - Advanced Inference Solutions
- The HPC-Accelerated AI (HPC-AI)
- Conclusion

Understanding the Deep Neural Network Concepts

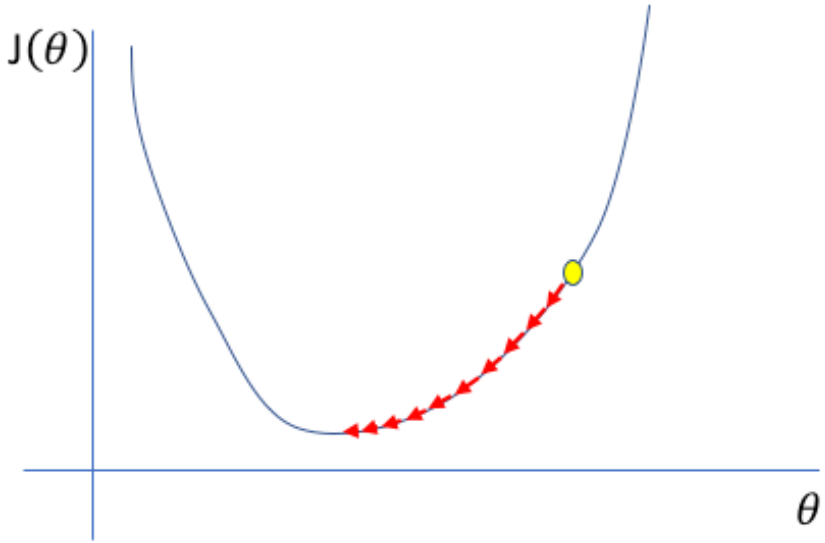
- Example of a 3-layer Deep Neural Network (DNN) – (input layer is not counted)



Courtesy: http://cs231n.github.io/neural-networks-1/

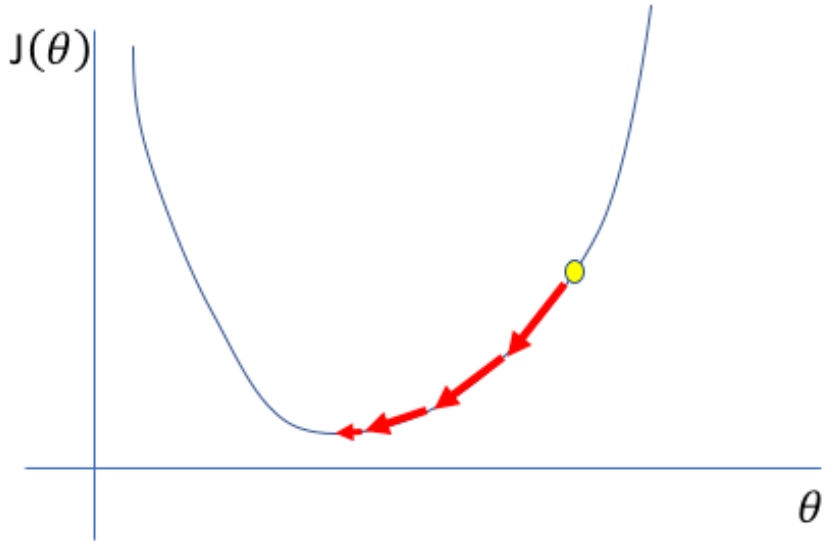
Essential Concepts: Learning Rate (α)

Too low



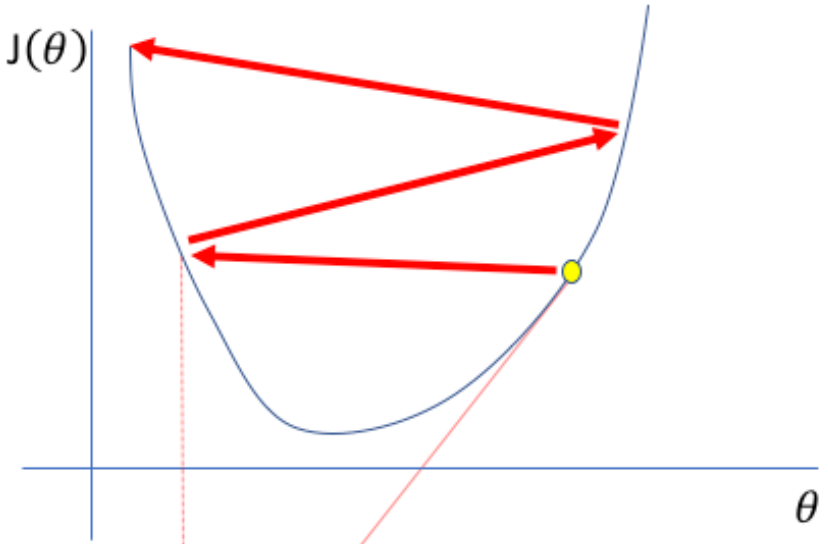
A small learning rate requires many updates before reaching the minimum point

Just right



The optimal learning rate swiftly reaches the minimum point

Too high

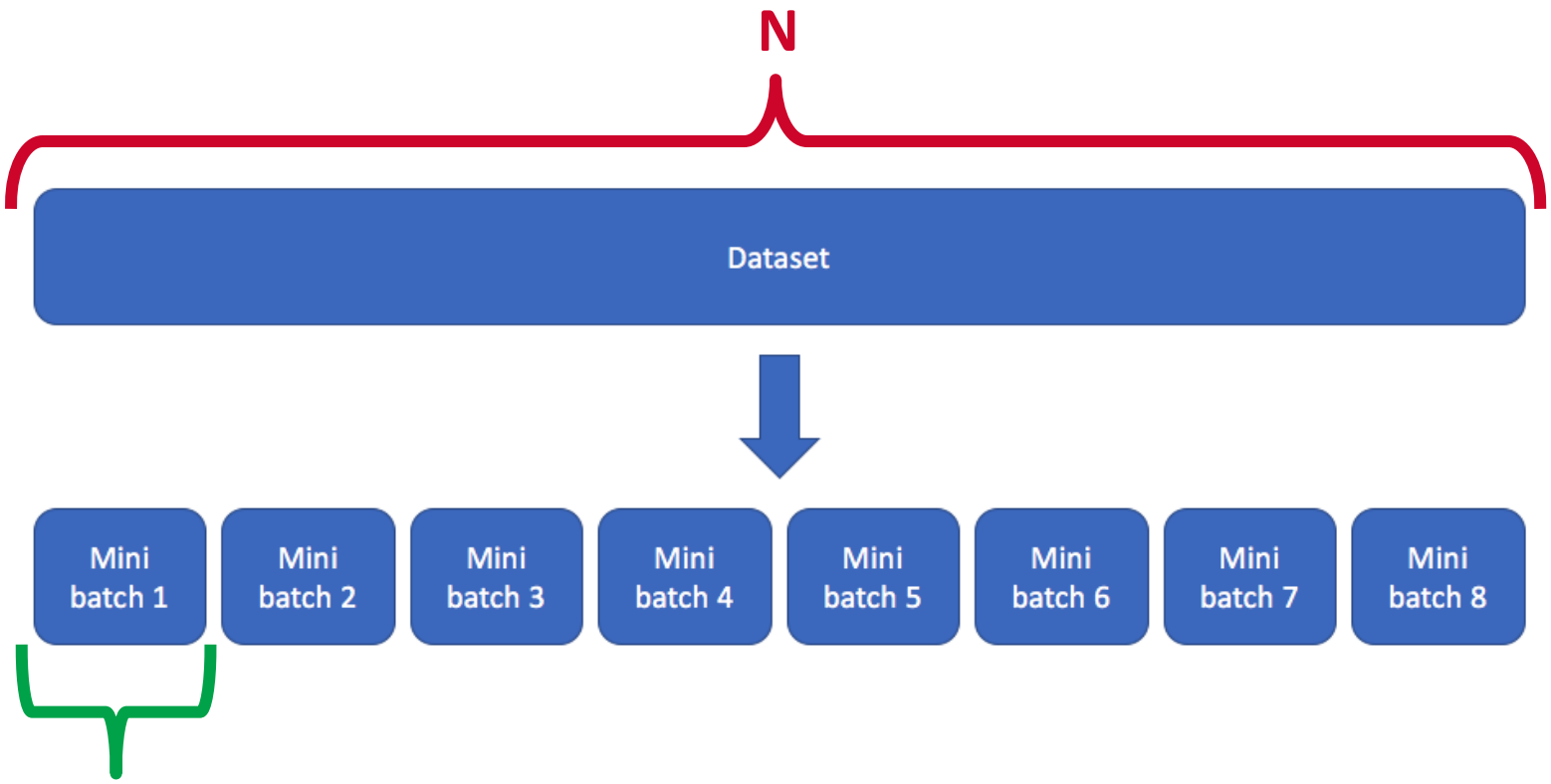


Too large of a learning rate causes drastic updates which lead to divergent behaviors

Courtesy: https://www.jeremyjordan.me/nn-learning-rate/

Essential Concepts: Batch Size

- Batched Gradient Descent
  - Batch Size = **N**
- Stochastic Gradient Descent
  - Batch Size = **1**
- Mini-batch Gradient Descent
  - Somewhere in the middle
  - Common:
    - **Batch Size** = 64, 128, 256, etc.
- Finding the optimal batch size will yield the fastest learning.

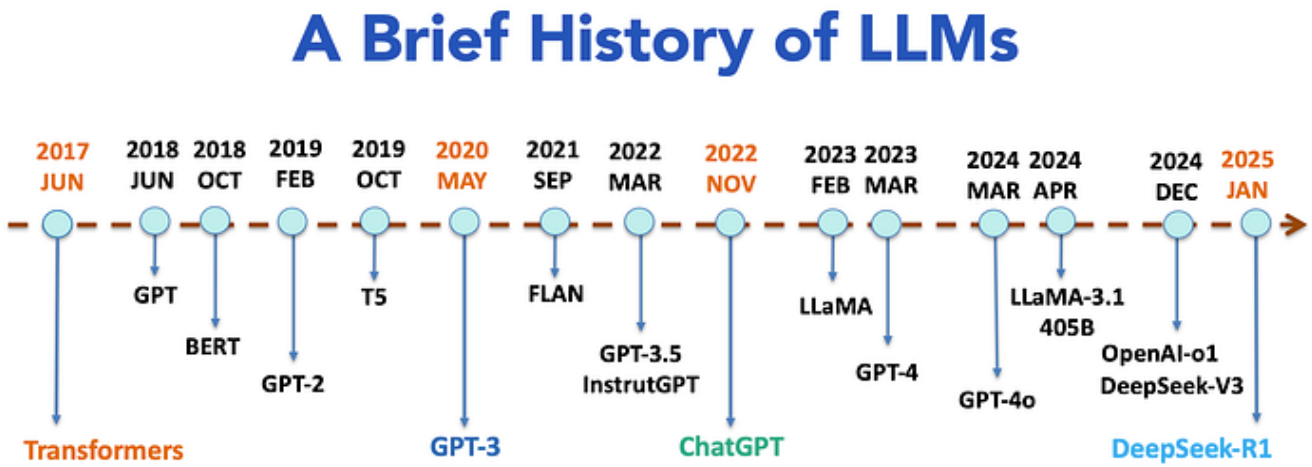


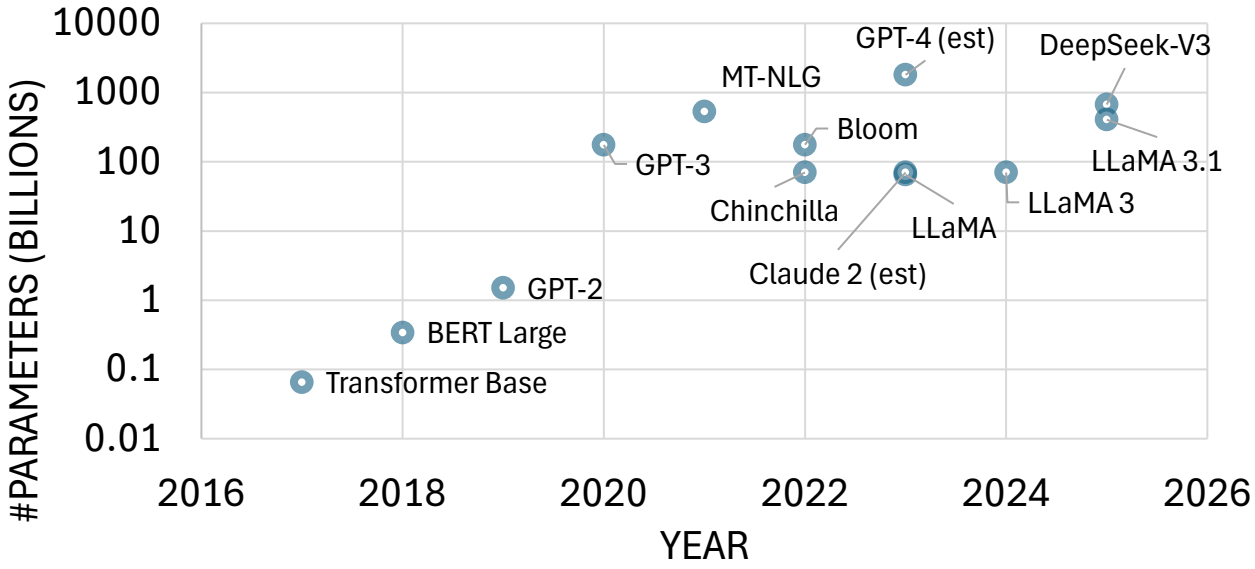
*One full pass over **N** is called an **epoch** of training*

Courtesy: https://www.jeremyjordan.me/gradient-descent/

Evolution of Language Models

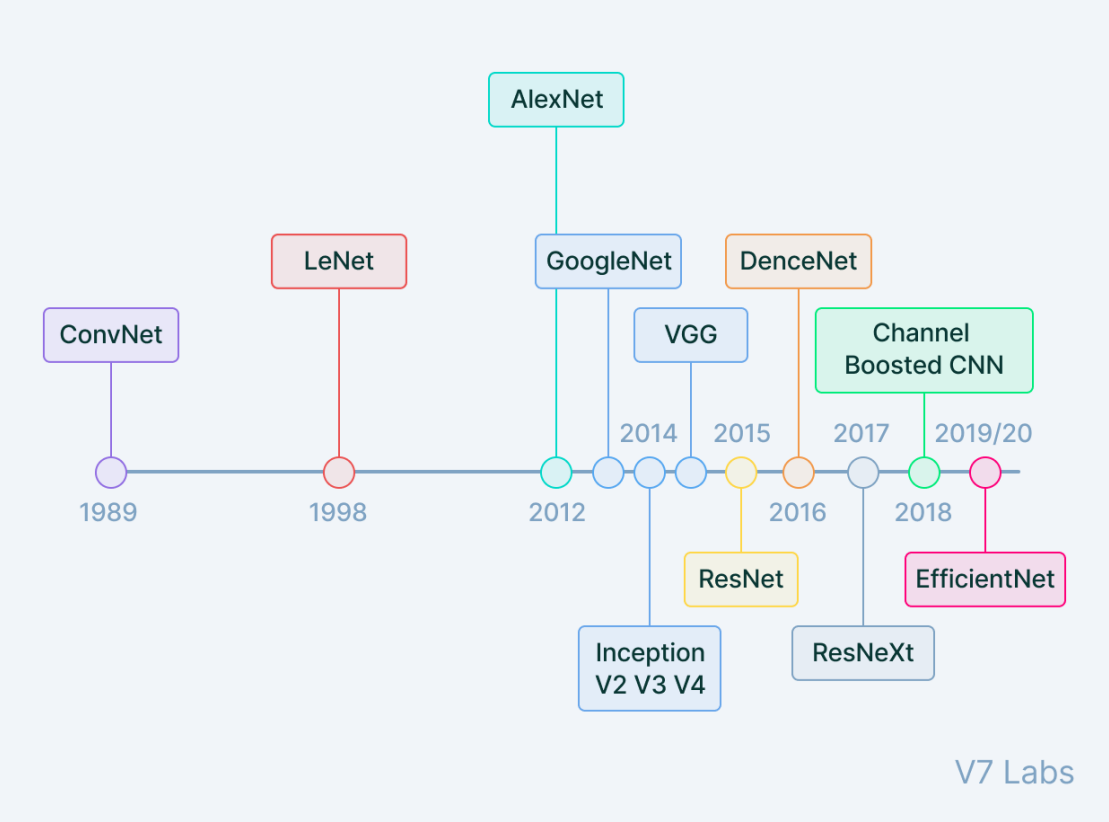
- LLM Performance improves predictably with scale (# of parameters and dataset size).
- Bigger ≠ always better: optimal performance requires balancing model size and data
- Diminishing returns at extreme scale → smaller gains per added parameter



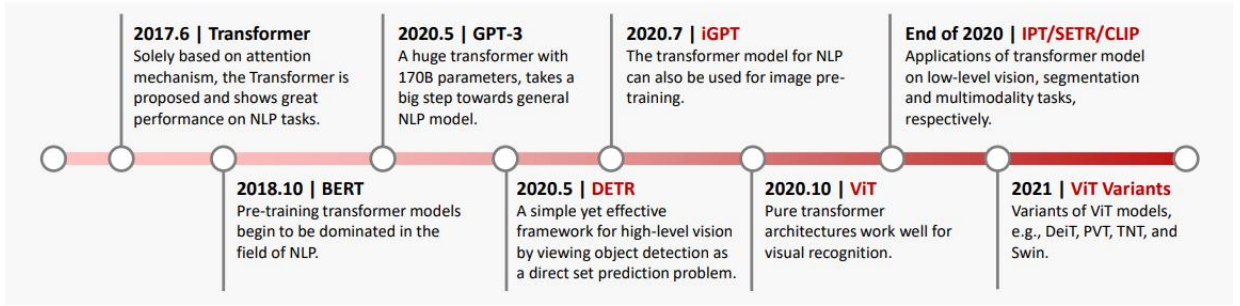


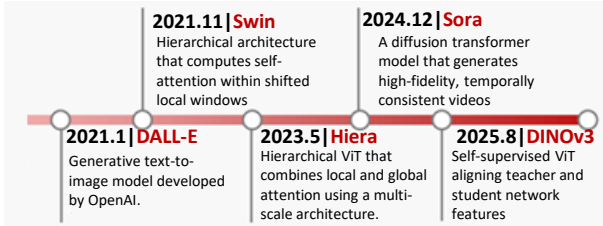
Evolution of Computer Vision Models

CNN Architectures



Vision Transformer Architectures





Courtesy: https://www.v7labs.com/blog/convolutional-neural-networks-guide

A Survey on Vision Transformer (Kai Han et. Al 2022) https://arxiv.org/abs/2012.12556

Outline

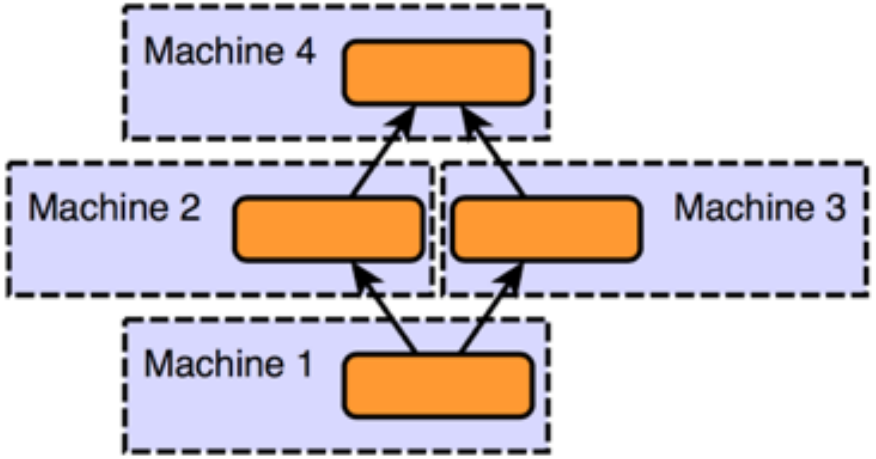
- Introduction
- **Training**
  - Deep Neural Network Training
  - **Distributed Deep Learning**
    - **Data Parallelism**
    - **Sharded Data Parallelism**
    - **Communication in Parallelism Strategies**
- Inference
  - Introduction to Inference
  - Advanced Inference Solutions
- The HPC-Accelerated AI (HPC-AI)
- Conclusion

The Need for Parallel and Distributed Training

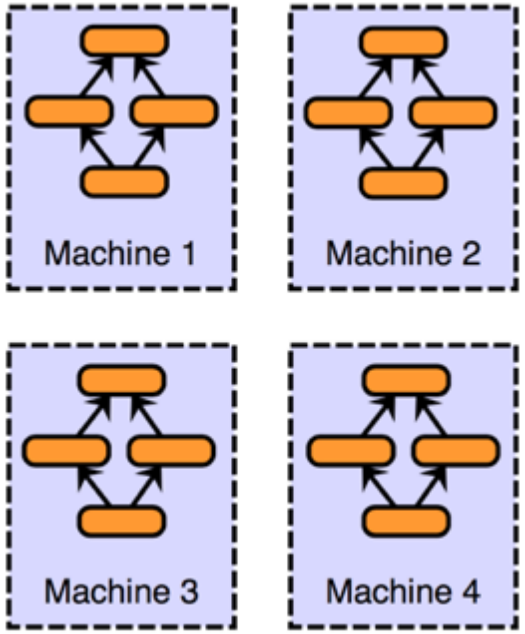
- Why do we need Parallel Training?
- Larger and Deeper models are being proposed
  - **Language Models: RNNs -> Transformers -> BERT – GPT – LLaMA**
  - **Vision Models: AlexNet** -> **ResNet** -> **NASNet – AmoebaNet → Vision Transformers**
  - DNNs require a lot of memory and a lot of computation
  - Larger models cannot fit a GPU's memory
- Single GPU training cannot keep up with ever-larger models
- Community has moved to multi-GPU training
- Multi-GPU in one node is good but there is a limit to Scale-up (8-16 GPUs)
- **Multi-node (Distributed or Parallel) Training is necessary!!**

Parallelization Strategies

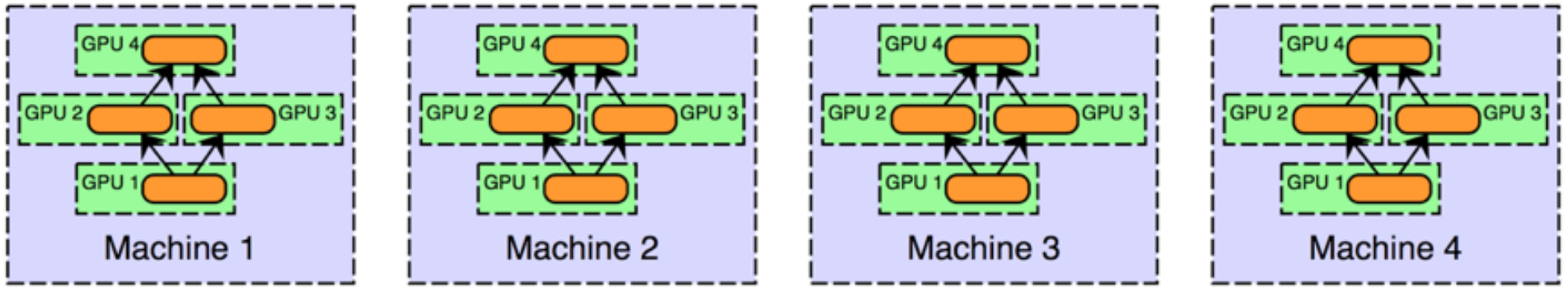
- Some parallelization strategies..
  - Data Parallelism or Model Parallelism
  - Hybrid Parallelism



Model Parallelism



Data Parallelism

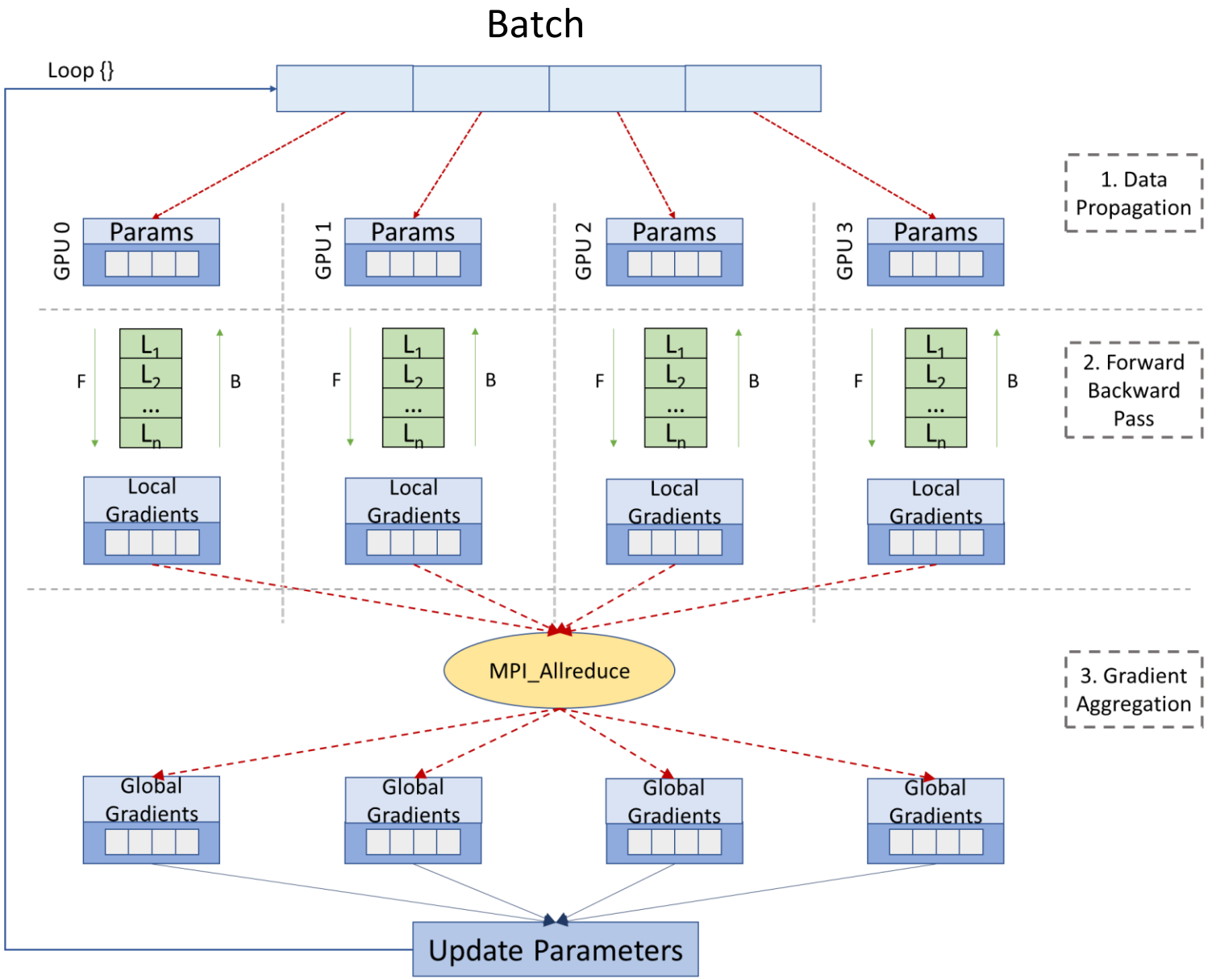


Hybrid (Model and Data) Parallelism

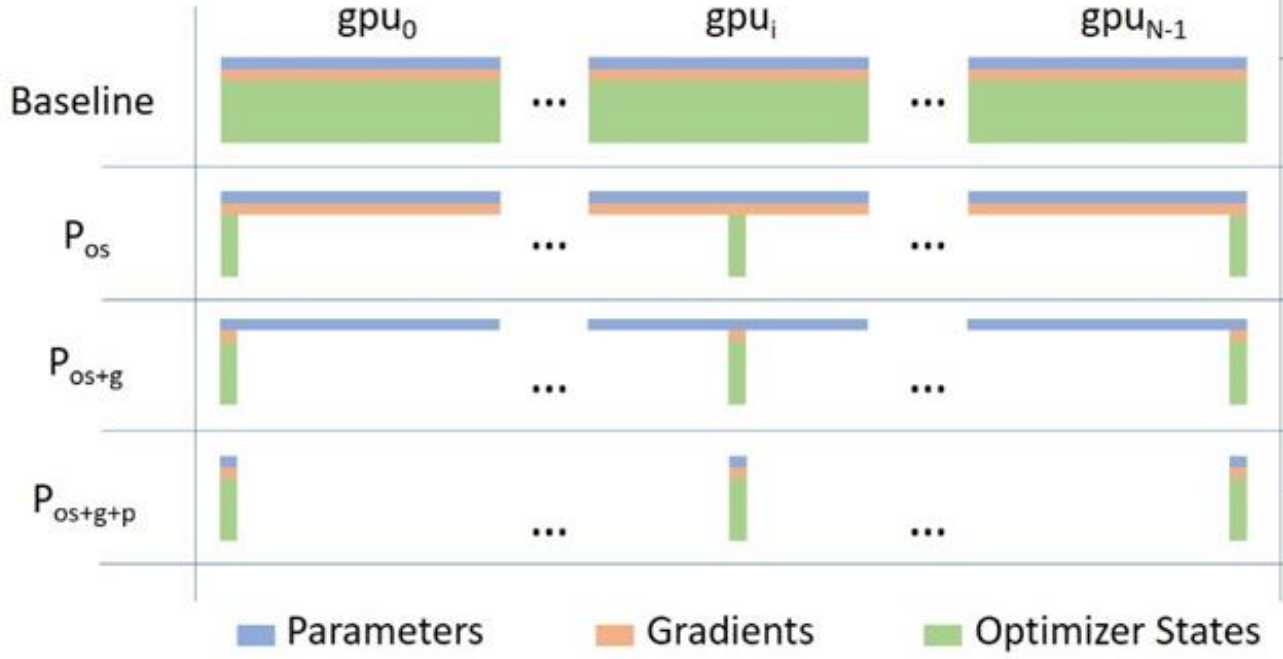
Courtesy: http://engineering.skymind.io/distributed-deep-learning-part-1-an-introduction-to-distributed-training-of-neural-networks

Data Parallelism and MPI Collectives

- Step1: Data Propagation
  - Distribute the Data among GPUs
- Step2: Forward Backward Pass
  - Perform forward pass and calculate the prediction
  - Calculate Error by comparing prediction with actual output
  - Perform backward pass and calculate gradients
- Step3: Gradient Aggregation
  - Call MPI_Allreduce to reduce the local gradients
  - Update parameters locally using global gradients



Sharded Data Parallelism (DeepSpeed ZeRO)



	Memory Consumption		Comm Volume
	Formulation	Specific Example K=12 Ψ=7.5B N_d=64	
Baseline	$(2+2+K)*\Psi$	120GB	1x
P_{os}	$2\Psi+2\Psi+\frac{K*\Psi}{N_d}$	31.4GB	1x
P_{os+g}	$2\Psi+\frac{(2+K)*\Psi}{N_d}$	16.6GB	1x
P_{os+g+p}	$\frac{(2+2+K)*\Psi}{N_d}$	1.9GB	1.5x

- Instead of being limited by the **device** memory, we are now limited by the **aggregate** memory
- ZeRO-Infinity introduces offload to CPU memory or NVMe disk for the truly desperate
- Since ZeRO removes the DP memory limit, do we still need MP?
  - There are still models and data samples that don't fit inside GPU memory ***even with ZeRO***
  - We can use pipeline + tensor parallelism along with ZeRO for these cases.

Communication in AI Parallelism Strategies

Parallelism Strategy	Description	Main Communication Operation(s)
Data Parallelism (DP)	Replicates the full model and splits the batch across devices.	AllReduce
FSDP / ZeRO	Shards parameters, gradients, and optimizer states.	AllGather + ReduceScatter
Tensor Parallelism (TP)	Splits individual layer/tensor computations across devices.	AllReduce
Pipeline Parallelism (PP)	Splits consecutive model layers into stages across devices.	Point-to-Point Send/Recv
3D Parallelism	Combines data, tensor, and pipeline parallelism simultaneously.	AllReduce/AllGather/ReduceScatter/P2P
Sequence Parallelism (SP)	Splits sequence-dependent activations/computation.	AllGather and ReduceScatter
Context Parallelism (CP)	Splits long-context tokens across devices.	P2P/Ring or AllGather
Expert Parallelism (EP)	Distributes MoE experts across devices and routes tokens.	All-to-All or All-to-Allv

Summary of major parallelism strategies and their associated communication operations

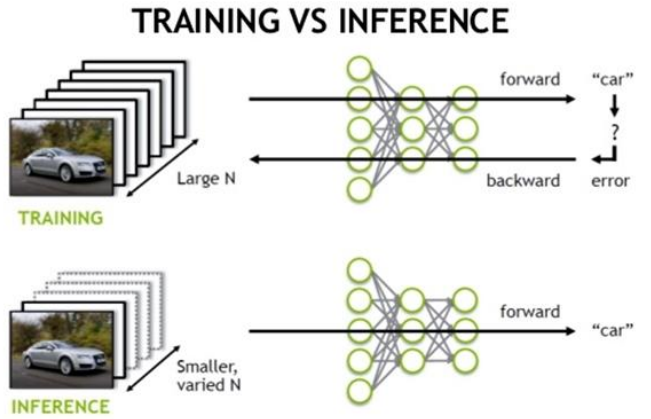
Outline

- Introduction
- Training
  - Deep Neural Network Training
  - Distributed Deep Learning
    - Data Parallelism
    - Sharded Data Parallelism
    - Communication in Parallelism Strategies
- **Inference**
  - **Introduction to Inference**
  - **Advanced Inference Solutions**
- The HPC-Accelerated AI (HPC-AI)
- Conclusion

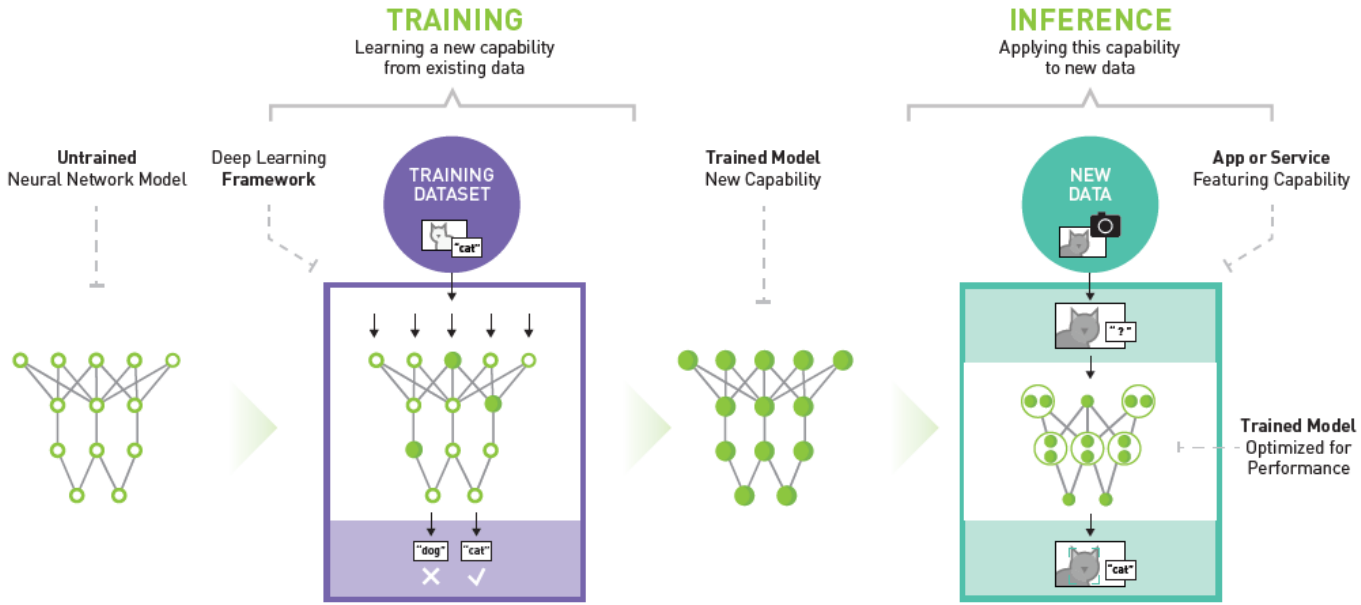
What is Deep Learning Inference?

- Deep learning Training & Inference

	Phase	Sensitivity
Training	Model-learning	Throughput
Inference	**User-facing**	**Latency**

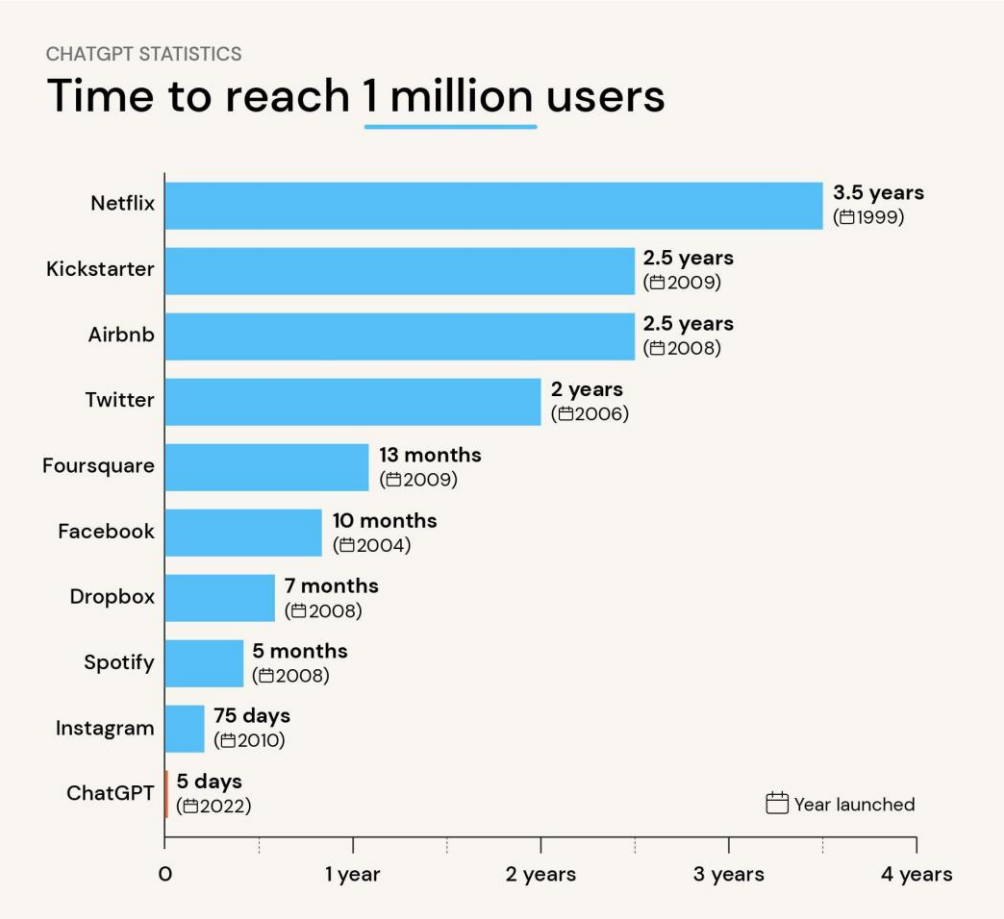


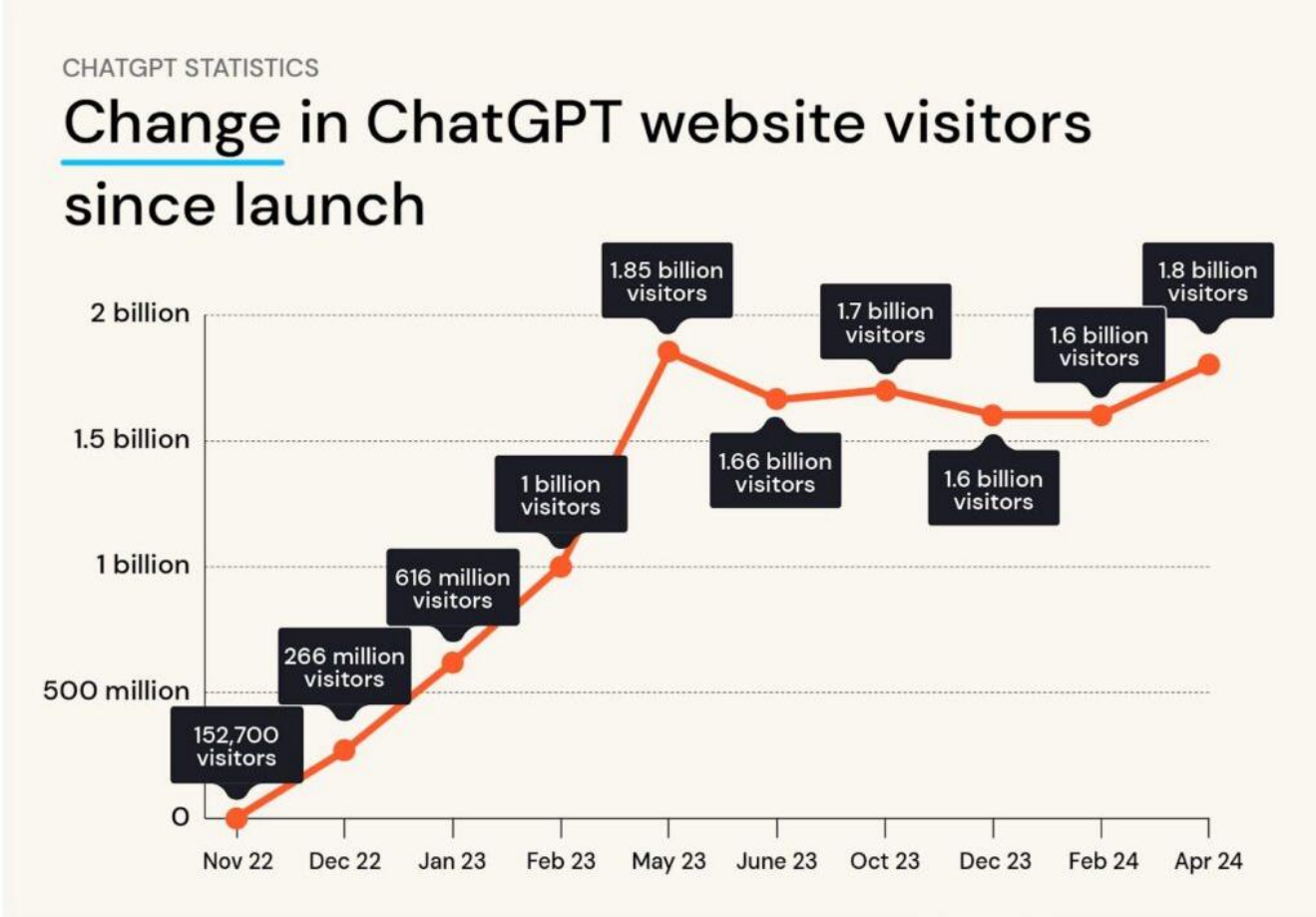
- Inference: Latency-sensitive
  - Final Phase of Deep Learning
  - **The closest end to users**
- Smaller batch size in the workflow
- User-end requests arrive randomly
- No need for model weights update
- Response time is the most crucial



Courtesy: https://developer.nvidia.com/blog/nvidia-deep-learning-inference-platform-performance-study/; https://www.exxactcorp.com/blog/HPC/discover-the-difference-between-deep-learning-training-and-inference

The mania of ChatGPT





Autoregressive property of GPT model

- **For a sequence with N tokens, training only takes 1 run on the model.**
- **However, inference takes N runs on the model. Totally sequential!!!**

Human: Hello.

AI: How

AI: How can

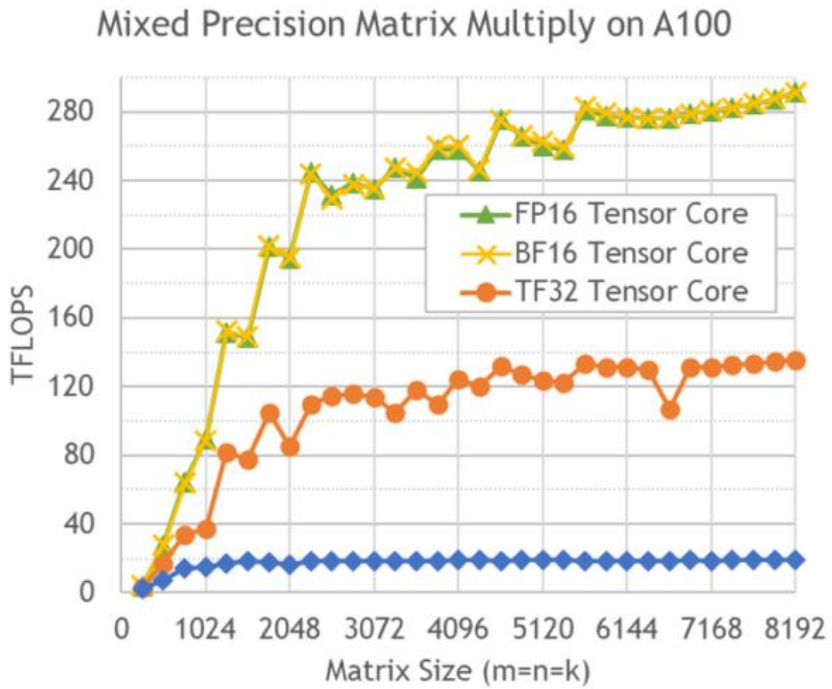
AI: How can I

AI: How can I help

AI: How can I help you

AI: How can I help you today?

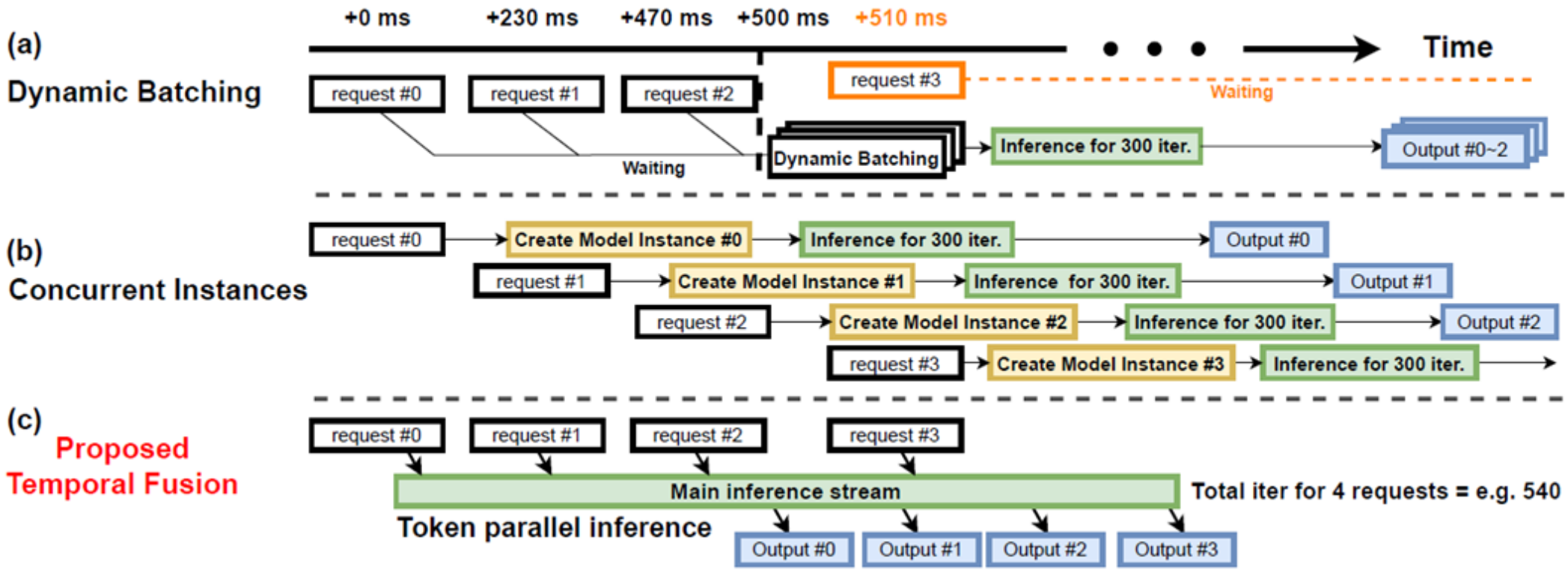
```
while True:
    new_word = None
    query = new_word
    key   = output_sentence
    value = output_sentence

    query = model.inference(query, key, value)

    if query != "$":
        output_sentence.append(query)
    else:
        print("Generation finishes.")
        break
```



- **It is wasting most GPU compute power!**

Flover: Efficient parallel inference on LLMs with In-flight batching[1]

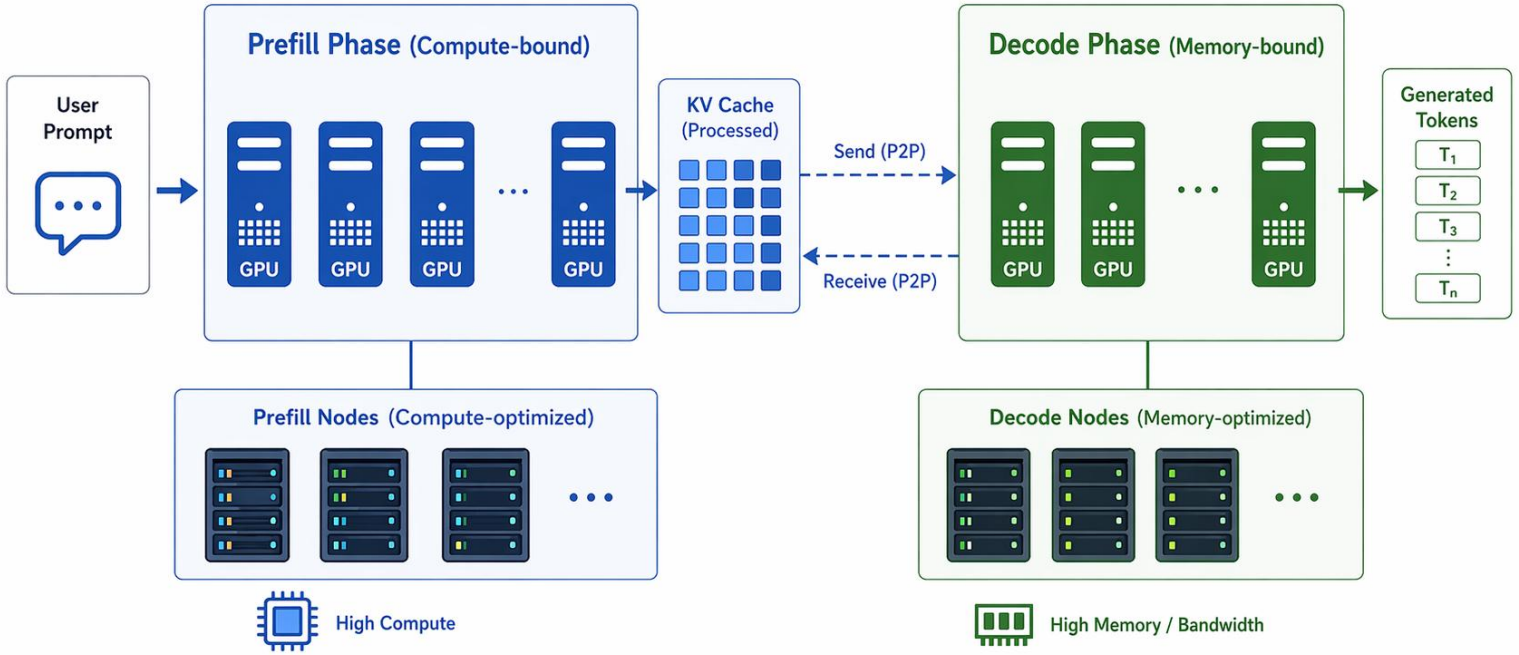
- When serving multiple requests, how to deliver both low-latency and high-throughput?
- For generative models such as GPT, LLaMA, the generation is sequential and regulated by 'for' loop.
  - For multiple requests that arrive at different time, how do we schedule the inference?



[1] Yao, Jinghan, Nawras Alnaasan, Tian Chen, Aamir Shafi, Hari Subramoni, D.K. Panda. "Flover: A Temporal Fusion Framework for Efficient Autoregressive Model Parallel Inference." *In Proceeding of HiPC 23*

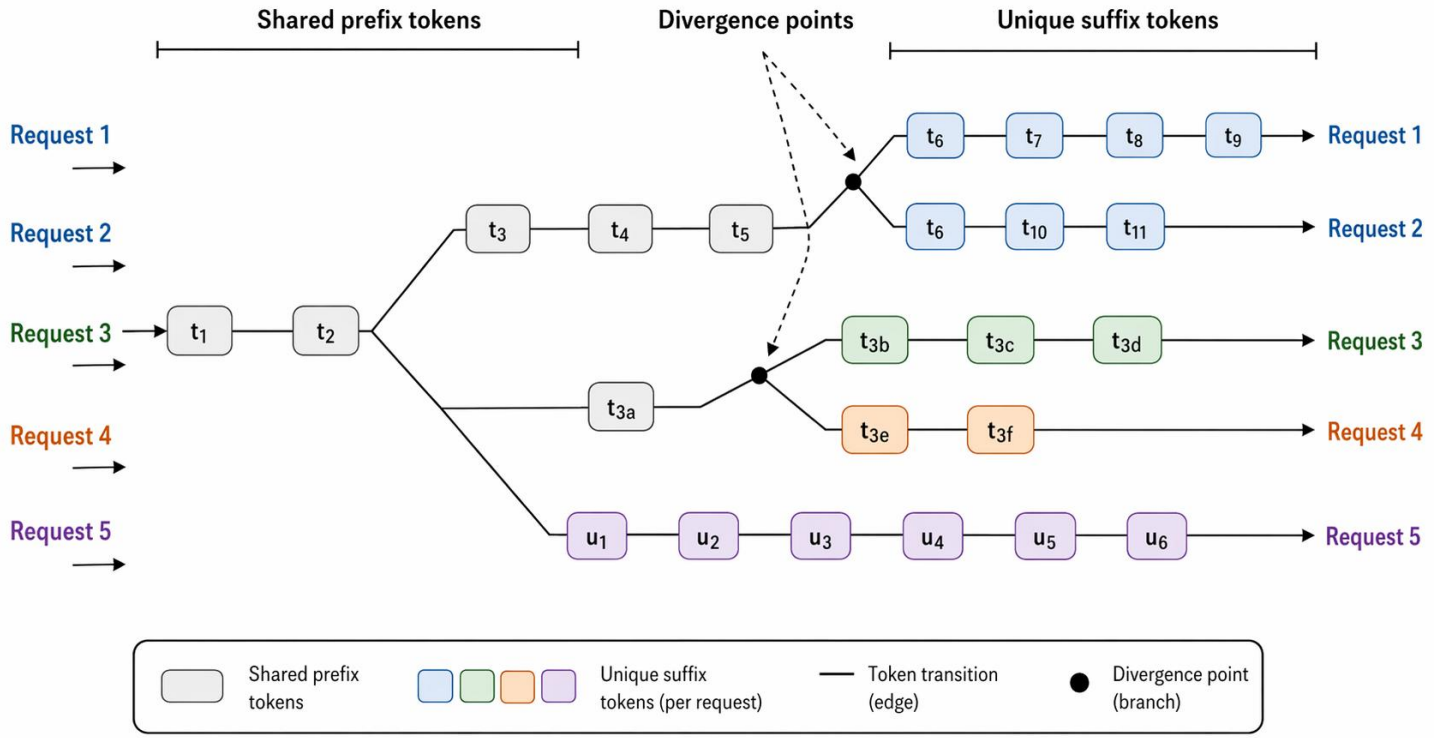
Multi-node Disaggregated Inference

- Prefilling is compute-bound, while decoding is memory bound. Deploying both phases on the same hardware is therefore not the first choice.
- Prefill-Decode disaggregation allows the split of prefill nodes and decode nodes.
- The processed KV cache is transferred from prefill nodes to decode nodes using point-to-point send and receive.



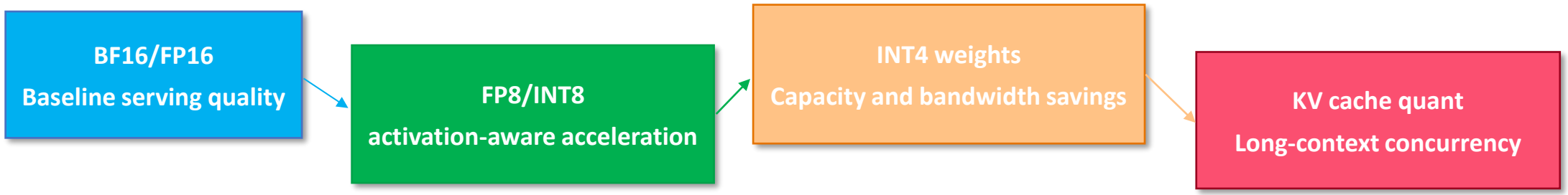
Memory Constraints: KV Cache

- Model weights mostly fixed; KV cache grows with active requests, context length, and output length.
- Instead of allocating a huge contiguous buffer for each request's KV cache, managing them as blocks (similar to page table) not only reduces fragmentation, enables larger dynamic batches, but also allows KV cache prefix sharing.



Quantization

- Quantization can reduce bytes, increase batch size.
- But it is also a systematic choice: quality, kernel support, calibration data, and target hardware all matter.



Activation + tensor-core formats

SmoothQuant, INT8, FP8, and emerging NVFP4/MXFP8 formats target fast hardware execution.

Weight-only quantization

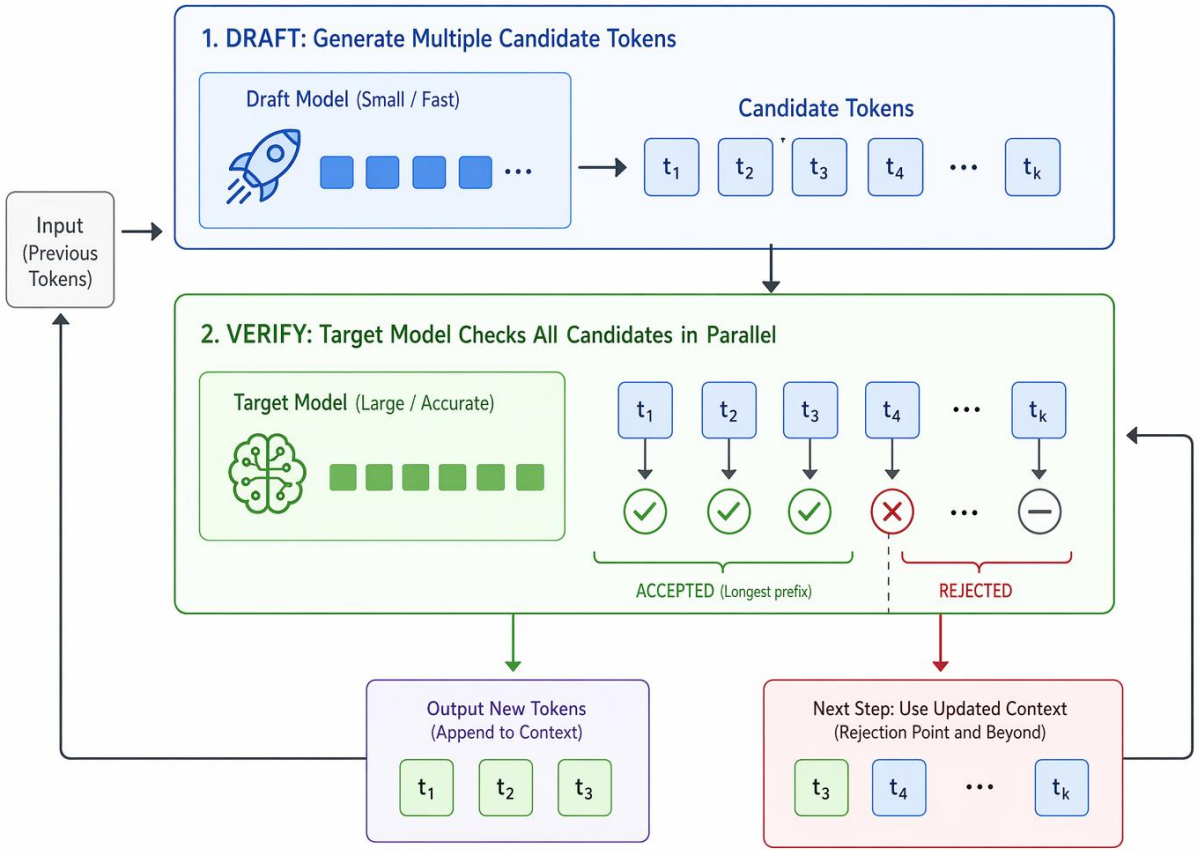
AWQ and GPTQ reduce model footprint and memory bandwidth, often with limited quality loss after calibration.

KV-cache quantization

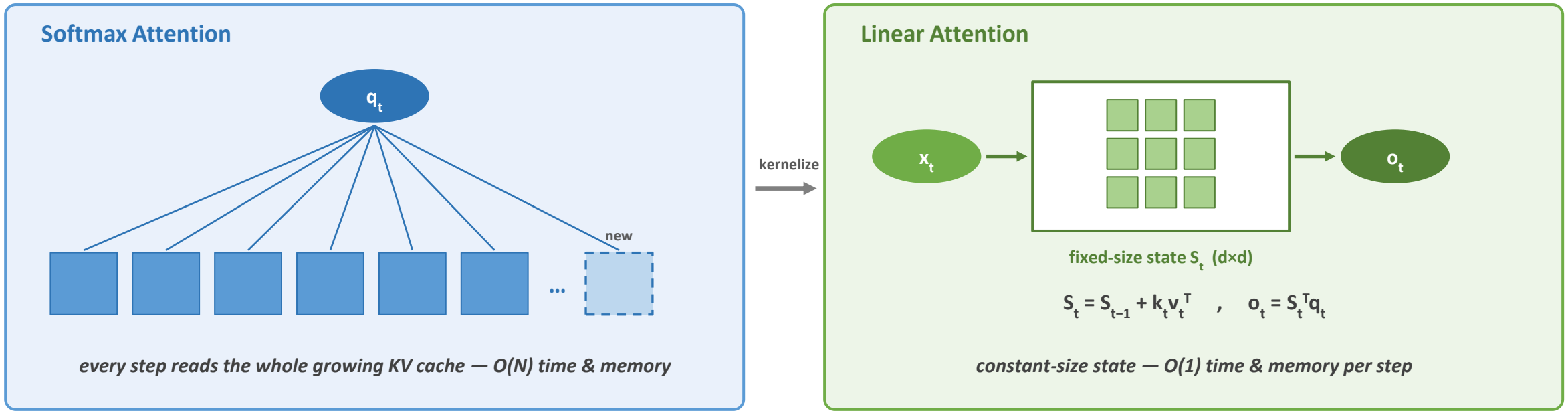
Compress the persistent cache, which matters most for long context, high concurrency, and streaming workloads.

Speculative Decoding

- Speculative decoding parallelizes the serial token generation loop.
- Draft fast: A small model, auxiliary heads, or feature predictor proposes several candidate tokens.
- Verify in parallel: The target model checks multiple candidates in one pass, preserving output distribution for accepted variants.
- Skip serial work: Accepted tokens reduce the number of target-model iterations; rejected tokens fall back safely.



Linear Attention



Linear Attention
softmax → kernel ϕ(q)·ϕ(k)

attention as an RNN (ICML '20)

SSMs: Mamba / Mamba-2
selective state spaces

hardware-aware scan; SSD ≈ linear attn

Gated DeltaNet (GDN)
delta rule + gating

erase-and-write memory (ICLR '25)

Hybrid LLMs
linear + full attention

Qwen3-Next (3 : 1), MiniMax-01, Kimi Linear

- **Fixed-size state bounds memory & latency, but limits exact recall → production models interleave a few full-attention layers.**

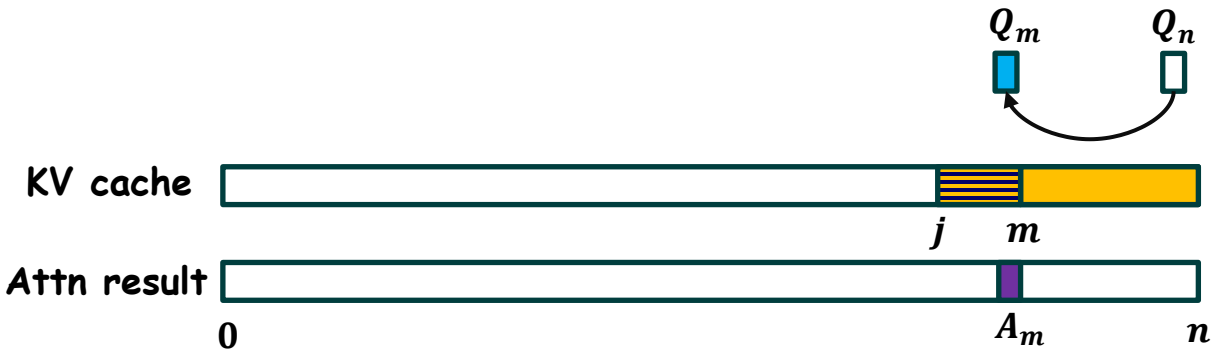
MAC-Attention: Match–Amend–Complete Decoding [1]

- Keep softmax attention — reuse its computation across semantically similar decode queries.

1. Match
reuse a similar recent query's prefix attention state

2. Amend
recompute a small band at the match boundary

3. Complete
attend over the new KV tail; stable LSE merge



KV %	Full Attn.	Quest	RocketKV	Multipole	MAC-Attn.
1	29.0	27.6	29.4	27.6	**30.2**
5	29.0	27.8	29.2	27.8	**30.4**
10	29.0	27.6	29.2	**30.2**	**30.2**
20	29.0	28.2	29.4	28.0	**29.6**

Overall accuracy on LongBench v2

KV %	Full Attn.	Quest	RocketKV	Multipole	MAC-Attn.
1	234.2	581.2	822.8	192.4	**62.9**
5	234.2	594.7	844.7	210.8	**64.0**
10	234.2	608.5	1042.5	265.4	**78.1**
20	234.2	640.5	1855.6	324.6	**103.8**

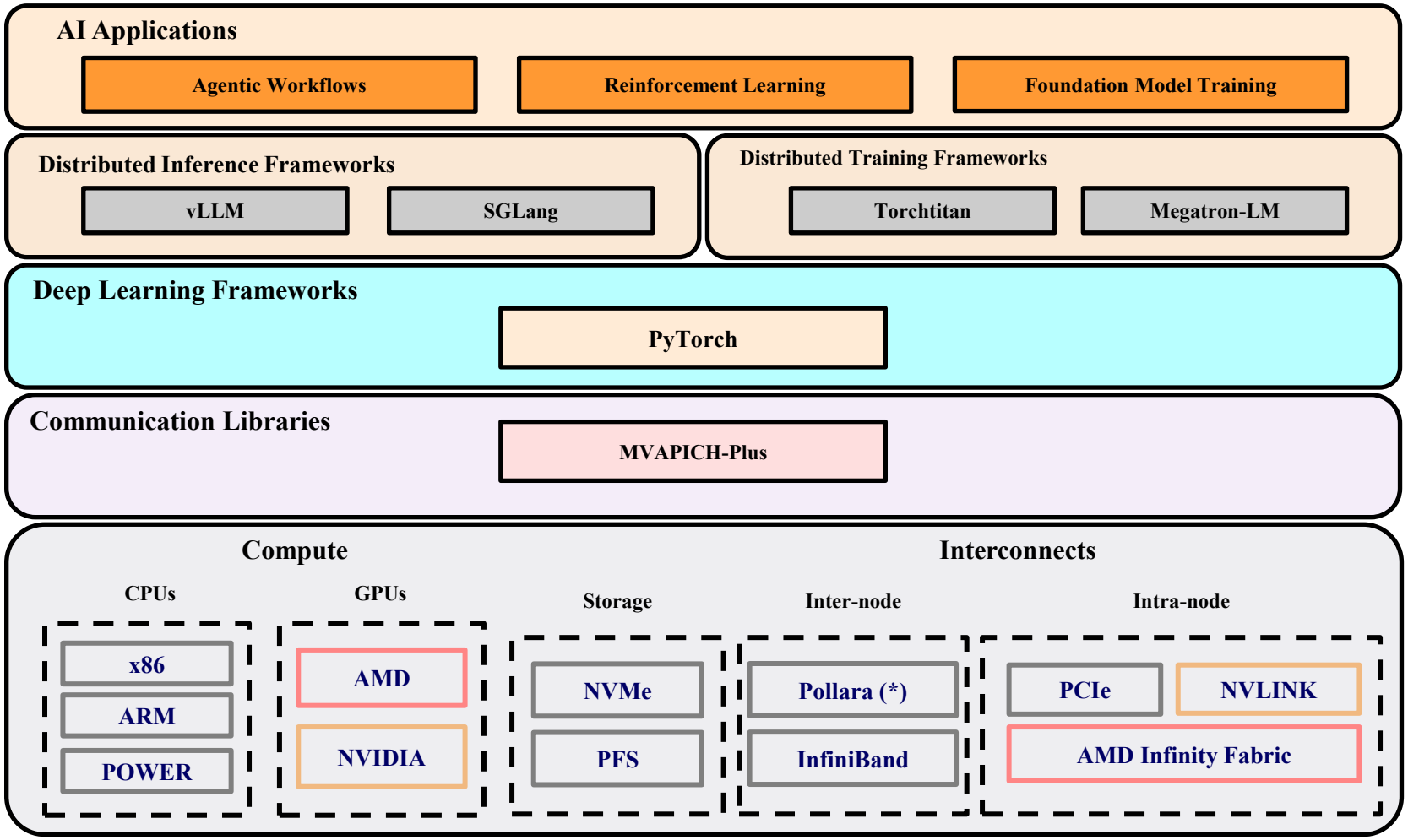
End-to-end Attention Latency (us) on 120K context

[1] Jinghan Yao, Sam Adé Jacobs, Walid Krichene, Masahiro Tanaka, Dhabaleswar K. Panda. "MAC-Attention: a Match-Amend-Complete Scheme for Fast and Accurate Attention Computation." In Proceedings of MLSys 2026 — github.com/YJHMITWEB/MAC-Attention

Outline

- Introduction
- Training
  - Deep Neural Network Training
  - Distributed Deep Learning
    - Data Parallelism
    - Sharded Data Parallelism
    - Communication in Parallelism Strategies
- Inference
  - Introduction to Inference
  - Advanced Inference Solutions
- **The HPC-Accelerated AI (HPC-AI)**
- Conclusion

HPC-Accelerated Vendor Neutral AI Stack with MPI Backend



* Upcoming

V1.0 is available from: https://github.com/OSU-Nowlab/osu-hidl

Getting Started with HPC-AI: Installation and Usage

Overview of the HPC-AI repository structure, installation and usage guides, and example workflows.

https://github.com/OSU-Nowlab/osu-hidl

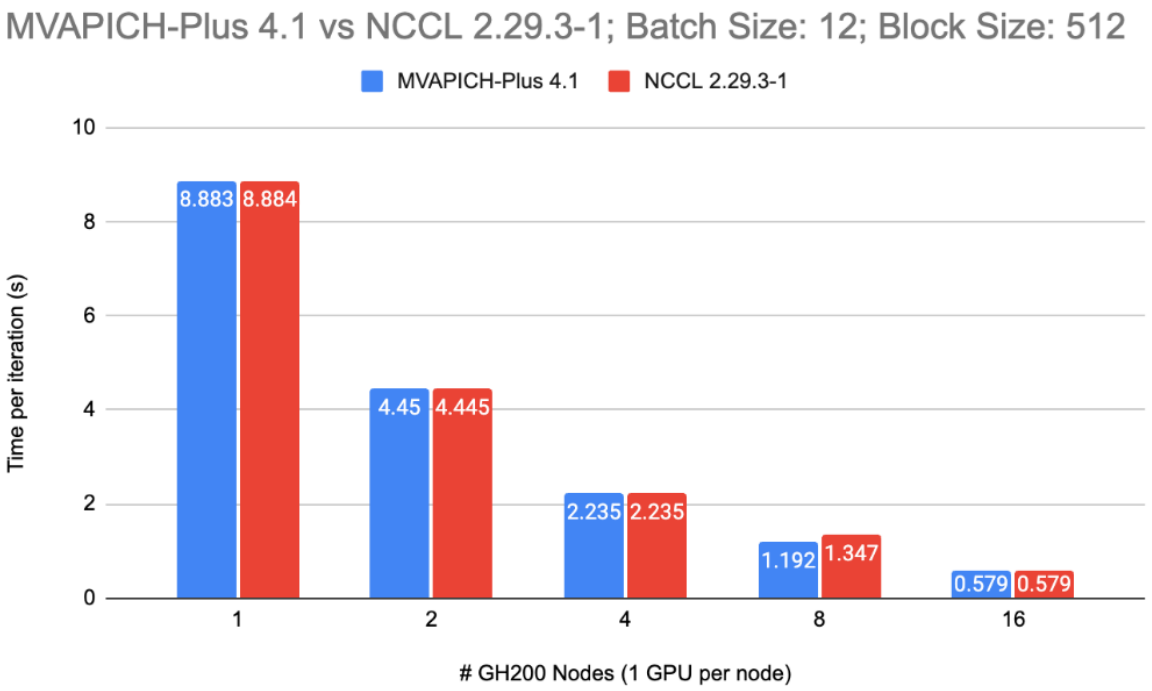
HPC-AI V1.0 Release

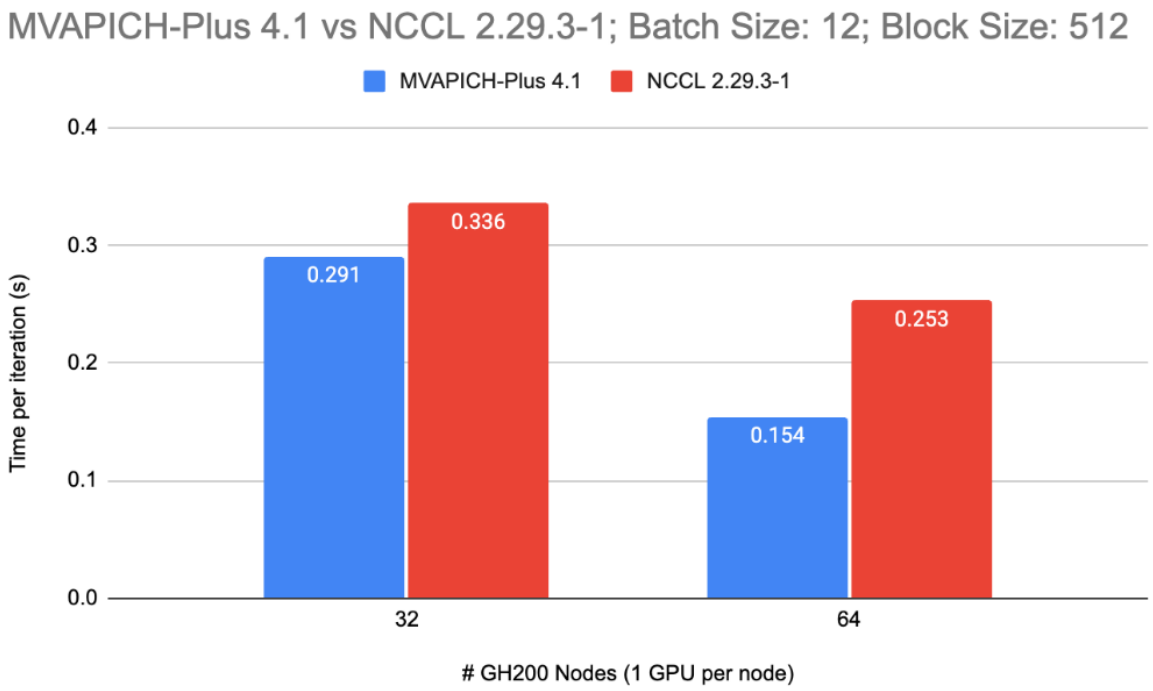
- Full-stack integration of training and inference frameworks, including **PyTorch, DeepSpeed, vLLM, and SGLang, with MVAPICH-Plus**
- Native PyTorch **Distributed Data Parallel (DDP)** training with MPI backend
- Advanced decoding method (**MAC-Attention**) and communication runtime (**MCR-DL**)
- Optimized **GPU-Direct Ring** and **two-level multi-leader** Allreduce algorithms
- **Low-precision communication support**, including **BF16** datatypes, for distributed AI workloads
- **Fork-safe execution** for distributed training and inference environments
- **Vendor-neutral communication stack** with performance competitive with GPU collective libraries such as NCCL and RCCL

- **Battle-tested on modern HPC systems**, including OLCF Frontier, TACC Vista, and SDSC Cosmos, across recent NVIDIA and AMD GPU architectures
- Compatible with:
  - InfiniBand Networks: Mellanox InfiniBand adapters (EDR, FDR, HDR, NDR)
  - Slingshot Networks: HPE Slingshot
  - GPU&CPU Support:
    - NVIDIA GPU A100, H100, GH200
    - AMD MI250X, MI300A GPUs
  - Software Stack:
    - CUDA [12.x] and Latest CuDNN
    - (NEW)ROCm [7.x]
    - (NEW)PyTorch [2.11.0]
    - (NEW)Training & Inference: DeepSpeed, vLLM, SGLang
    - (NEW)Advanced: MAC-Attention, MCR-DL
    - (NEW)Python [3.x]

For more details: https://hpc-ai.engineering.osu.edu/

Distributed Data Parallel Training on H200 (Vista)



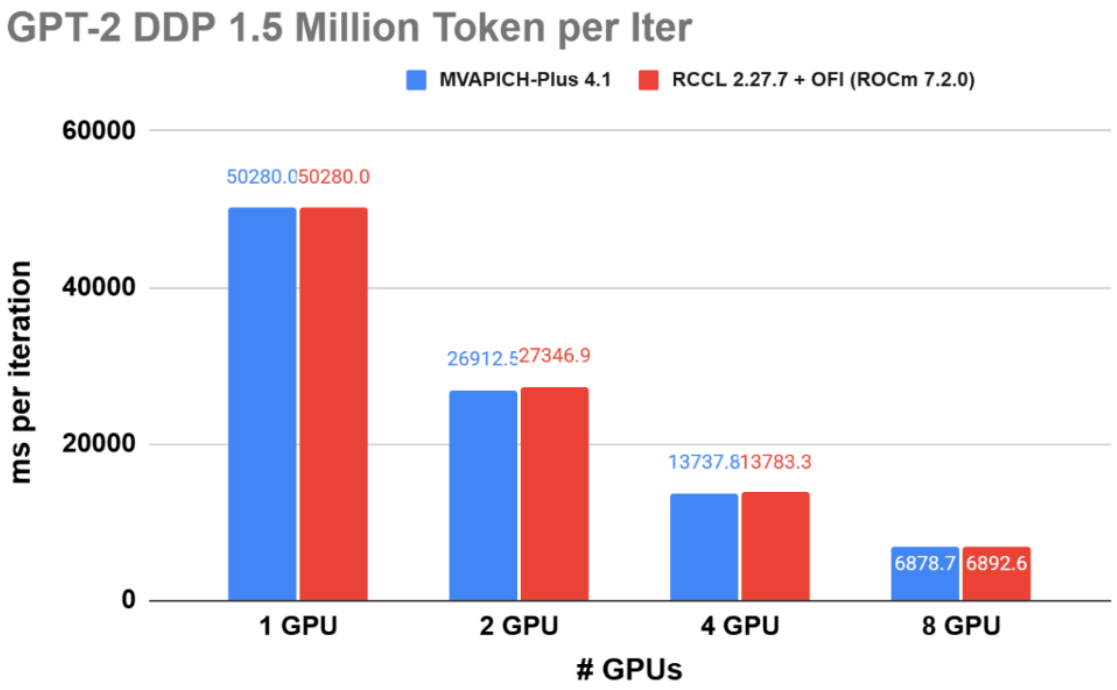


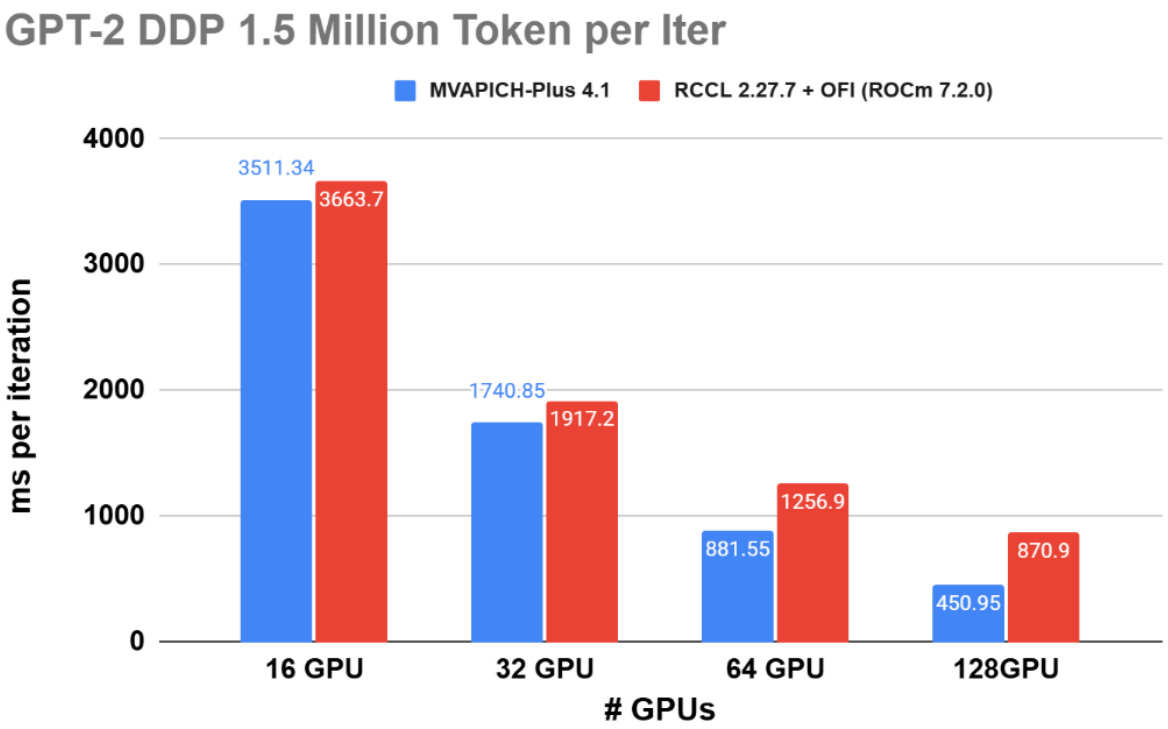
- **1.64x speedup**(compared to NCCL 2.29.3-1) on 64 GPUs

Machine Specifications: TACC Vista					
CPU Model	**CPU Core Info**	**Memory**	**GPU Model**	**GPU Memory**	**Interconnects**
NVIDIA Grace CPU	1x72@3.1 GHz	116 GB DDR5	NVIDIA H200 GPU (1/Node)	96 GB HBM 3	Mellanox NDR (400 Gb/s)

Model	**Batch Size**	**Block Size**	**Gradient Accumulation Steps**	**Benchmark**	**Dataset**	**DL Framework**
GPT-2	12	512	256	NanoGPT	OpenWebText	PyTorch 2.10.0

Distributed Data Parallel Training on MI250X (Frontier)





- **1.9x speedup**(compared to RCCL 2.27.7 + OFI) on 128 GPUs

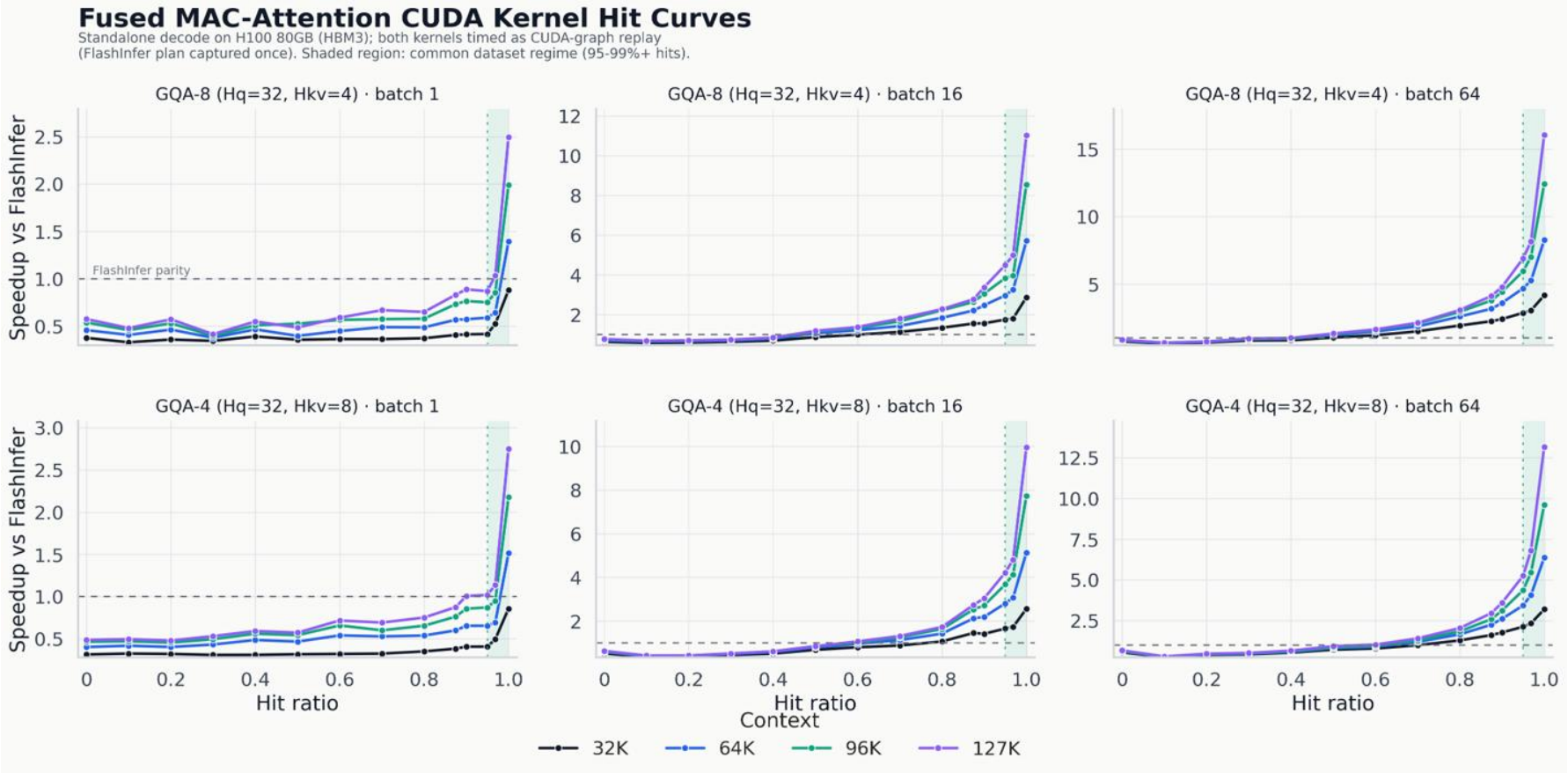
Machine Specifications: OLCF Frontier

CPU Model	CPU Core Info	Memory	GPU Model	GPU Memory	Interconnects
AMD EPYC 7A53 CPU	1x64@2GHz	512 GB DDR4	AMD MI250X (4/Node)	128 GB HBM 2e	HPE Slingshot (200 Gb/s)

Model	Batch Size	Block Size	Benchmark	Dataset	DL Framework
GPT-2	12	1024	NanoGPT	OpenWebText	PyTorch 2.10.0

Advanced Decoding: MAC-Attention on H100

- **Speedup grows with the query-match hit ratio — multi-× over FlashInfer in the common 95–99% hit regime, up to ~16× (127K context, batch 64).**



Both kernels timed as CUDA-graph replay on H100 80GB (HBM3); GQA-8 (Qwen3-30B shape) and GQA-4, batch 1–64, 32K–127K context.

Outline

- Introduction
- Training
  - Deep Neural Network Training
  - Distributed Deep Learning
    - Data Parallelism
    - Sharded Data Parallelism
    - Communication in Parallelism Strategies
- Inference
  - Introduction to Inference
  - Advanced Inference Solutions
- The HPC-Accelerated AI (HPC-AI)
- **Conclusion**

Conclusion

- Exponential growth in in Deep Learning frameworks and workloads.
- Provided an overview of issues, challenges, and opportunities for designing efficient communication runtimes for training and inference
  - Efficient, scalable, and hierarchical designs are crucial for DL frameworks
  - Co-design of communication runtimes and DL frameworks will be essential
- Presented a set of state-of-the-art solutions to demonstrate the complex interaction between DL middleware with the underling HPC technologies and middleware
- Demonstrated the scalability of efficient solutions on a diverse set of DL workloads.

Funding Acknowledgments

Funding Support by

Equipment Support by

Acknowledgments to all the Heroes (Past/Current Students and Staffs)

Current Students (Under/Graduate)

- N. Alnaasan (Ph.D.)
- Q. Anthony (Ph.D.)
- C.-C. Chen (Ph.D.)
- T. Chen (Ph.D.)
- N. Contini (Ph.D.)
- S. Gumaste (Ph.D.)
- J. Hatef (Ph.D.)
- G. Kuncham (Ph.D.)
- S. Lee (Ph.D.)
- B. Michalowicz (Ph.D.)
- J. Oswal (Ph.D.)
- T. Tran (Ph.D.)
- L. Xu (P.h.D.)
- S. Xu (Ph.D.)
- J. Yao (Ph.D.)
- S. Zhang (Ph.D.)
- S. Mohammad (M.S.)
- B. Lampe (B.S.)
- N. Klein (B.S.)

Current Research Specialist

- R. Motlagh

Current Software Engineers

- N. Shineman
- M. Lieber

Past Research Scientists

- K. Hamidouche
- S. Sur
- X. Lu
- M. Abduljabbar
- A. Shafi

Past Students

- A. Awan (Ph.D.)
- A. Augustine (M.S.)
- P. Balaji (Ph.D.)
- M. Bayatpour (Ph.D.)
- R. Biswas (M.S.)
- S. Bhagvat (M.S.)
- A. Bhat (M.S.)
- D. Buntinas (Ph.D.)
- L. Chai (Ph.D.)
- B. Chandrasekharan (M.S.)
- S. Chakraborthy (Ph.D.)
- N. Dandapanthula (M.S.)
- V. Dhanraj (M.S.)
- C.-H. Chu (Ph.D.)
- T. Gangadharappa (M.S.)
- K. Gopalakrishnan (M.S.)
- R. Gulhane (M.S.)
- J. Hashmi (Ph.D.)
- M. Han (M.S.)
- W. Huang (Ph.D.)
- A. Jain (Ph.D.)
- J. Jani (M.S.)
- W. Jiang (M.S.)
- J. Jose (Ph.D.)
- M. Kedia (M.S.)
- K. S. Khorassani (Ph.D.)
- S. Kini (M.S.)
- M. Koop (Ph.D.)
- P. Kousha (Ph.D.)
- K. Kulkarni (M.S.)
- R. Kumar (M.S.)
- S. Krishnamoorthy (M.S.)
- K. Kandalla (Ph.D.)
- M. Li (Ph.D.)
- P. Lai (M.S.)
- J. Liu (Ph.D.)
- M. Luo (Ph.D.)
- A. Mamidala (Ph.D.)
- G. Marsh (M.S.)
- V. Meshram (M.S.)
- A. Moody (M.S.)
- S. Naravula (Ph.D.)
- R. Noronha (Ph.D.)
- X. Ouyang (Ph.D.)
- S. Pai (M.S.)
- S. Potluri (Ph.D.)
- J. Queiser (M.S.)
- K. Raj (M.S.)
- R. Rajachandrasekar (Ph.D.)
- B. Ramesh (Ph.D.)
- D. Shankar (Ph.D.)
- G. Santhanaraman (Ph.D.)
- N. Sarkauskas (B.S. and M.S)
- V. Sathu (M.S.)
- N. Senthil Kumar (M.S.)
- A. Singh (Ph.D.)
- J. Sridhar (M.S.)
- S. Srivastava (M.S.)
- H. Subramoni (Ph.D.)
- S. Sur (Ph.D.)
- K. K. Suresh (Ph.D.)
- K. Vaidyanathan (Ph.D.)
- A. Vishnu (Ph.D.)
- J. Wu (Ph.D.)
- W. Yu (Ph.D.)
- J. Zhang (Ph.D.)
- Q. Zhou (Ph.D.)
- N. Chmura (B.S.)

Past Faculty

- H. Subramoni

Past Senior Research Associate

- J. Hashmi

Past Programmers

- A. Reifsteck
- D. Bureddy
- J. Perkins
- B. Seeds
- A. Guptha
- N. Pavuk

Past Post-Docs

- D. Banerjee
- X. Besseron
- M. S. Ghazimirsaeed
- H.-W. Jin
- J. Lin
- M. Luo
- E. Mancini
- K. Manian
- S. Marcarelli
- A. Ruhela
- J. Vienne
- H. Wang

Past Research Specialist

- M. Arnold
- J. Smith

Interested in learning more?

1. Join our half-day tutorials at IEEE Hot Interconnects 2026!

- **Tutorial: Principles and Practice of Scalable and Distributed AI Training and Inference**
  - Friday, August 21, 13:30 - 16:30 Pacific Time (PT)
- **Tutorial: High Performance and Smart Networking Technologies for HPC and AI**
  - Friday, August 21, 09:00 - 12:00 Pacific Time (PT)

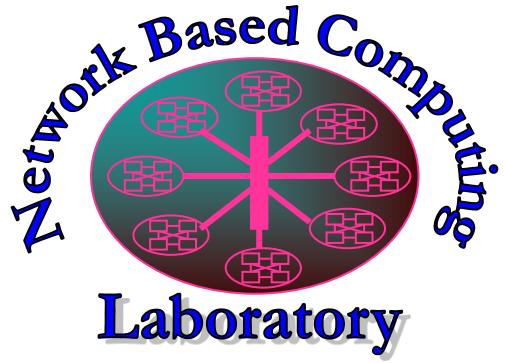
Hot Interconnects Registration (free) https://hoti.org/register.html

2. Join our full-day tutorials at Supercomputing (SC) 2026!

- **Tutorial: Principles and Practice of Scalable and Distributed AI Training and Inference**
  - Nov 15-20 (Exact date/time TBD)
- **Tutorial: High Performance and Smart Networking Technologies for HPC and AI**
  - Nov 15-20 (Exact date/time TBD)

Thank You!

{alnaasan.1, yao.877}@osu.edu



Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/

The High-Performance MPI/PGAS Project
http://mvapich.cse.ohio-state.edu/

The High-Performance Deep Learning Project
http://hidl.cse.ohio-state.edu/