MLCommons Chakra

Advancing performance benchmarking and SW/HW co-design using standardized execution traces

Presenter: Brian Coutinho, NVIDIA

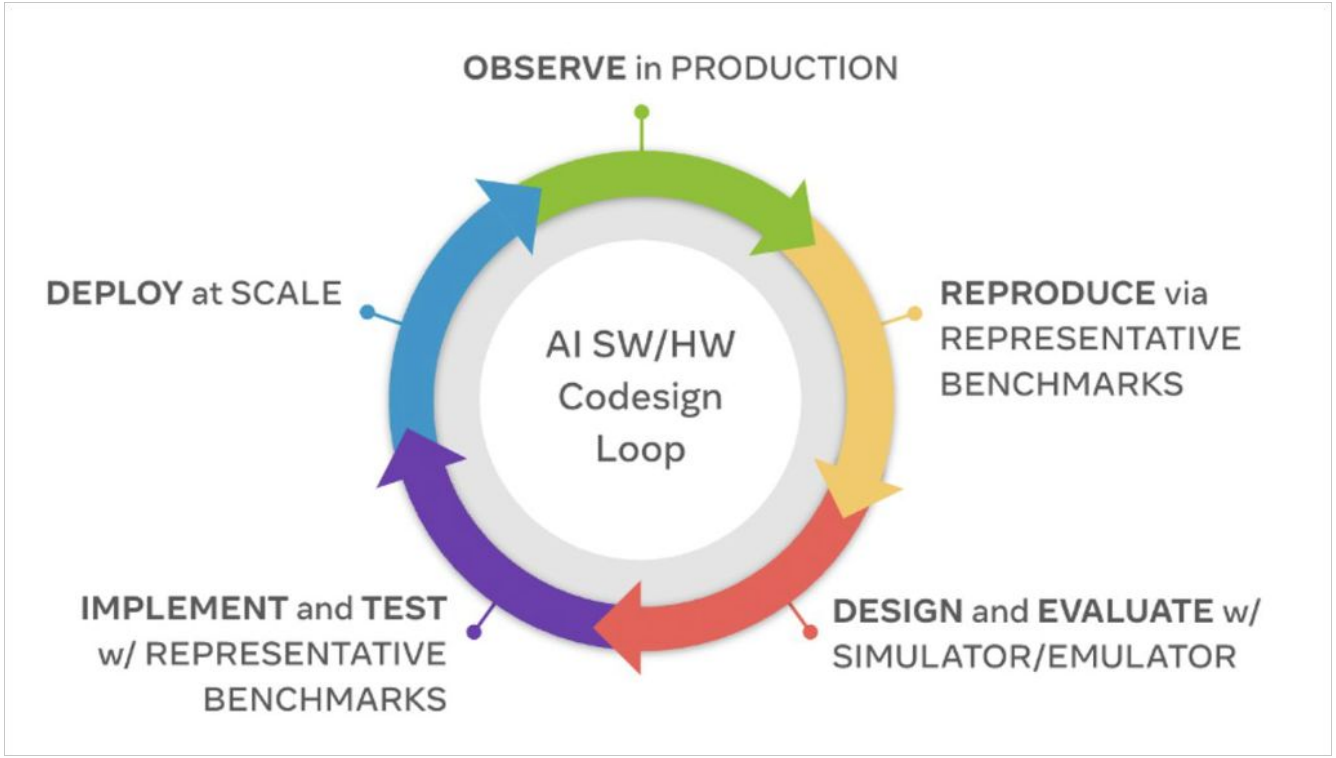
Srinivas Sridharan [1] Andy Balogh [2] Bradford M. Beckmann [3] Brian Coutinho [1] Louis Feng [4] Sheng Fu [1]
Sanshan Gao [1] Mehryar Garakani [5] Taekyung Heo [1] David Kanter [6] Josh Ladd [1] Ziwei Li [7] Winston Liu [2]
Changhai Man [7] Dan Mihailescu [2] Spandan More [3] Joongun Park [7] Ashwin Ramachandran [4]
Vinay Ramakrishnaiah [3] Saeed Rashidi [4] Vijay Janapa Reddi [8] Puneet Sharma [9] Phio Tian [1] William Won [3 7]
Hanjiang Wu [7] Huan Xu [7] Jinsun Yoo [7] Tushar Krishna [7]

1 NVIDIA 2 KEYSIGHT TECHNOLOGIES 3 AMD 4 Meta 5 SCALA COMPUTING 6 ML Commons 7 Georgia Tech 8 HARVARD UNIVERSITY 9 Hewlett Packard Enterprise

Outline

1. Motivation
2. Chakra Trace
   a. Trace Schema
   b. Collecting Chakra Traces
3. Downstream Use-cases
   a. Trace Analysis
   b. Trace Replay
   c. Simulation/Emulation
4. Conclusion

The HW/SW Co-design Cycle



Motivation

- **Fragmented ecosystem** for exchanging performance data between AI labs, hyperscalers and hardware vendors.

- **Open source AI benchmarks** like **MLPerf evolve slowly**
  - Challenges sharing proprietary models between frontier labs, hyperscalers and hardware vendors

- **Full stack benchmarking is expensive**
  - It requires scarce cluster resources, cross-domain expertise, and hard-to-isolate compute, memory, and network effects

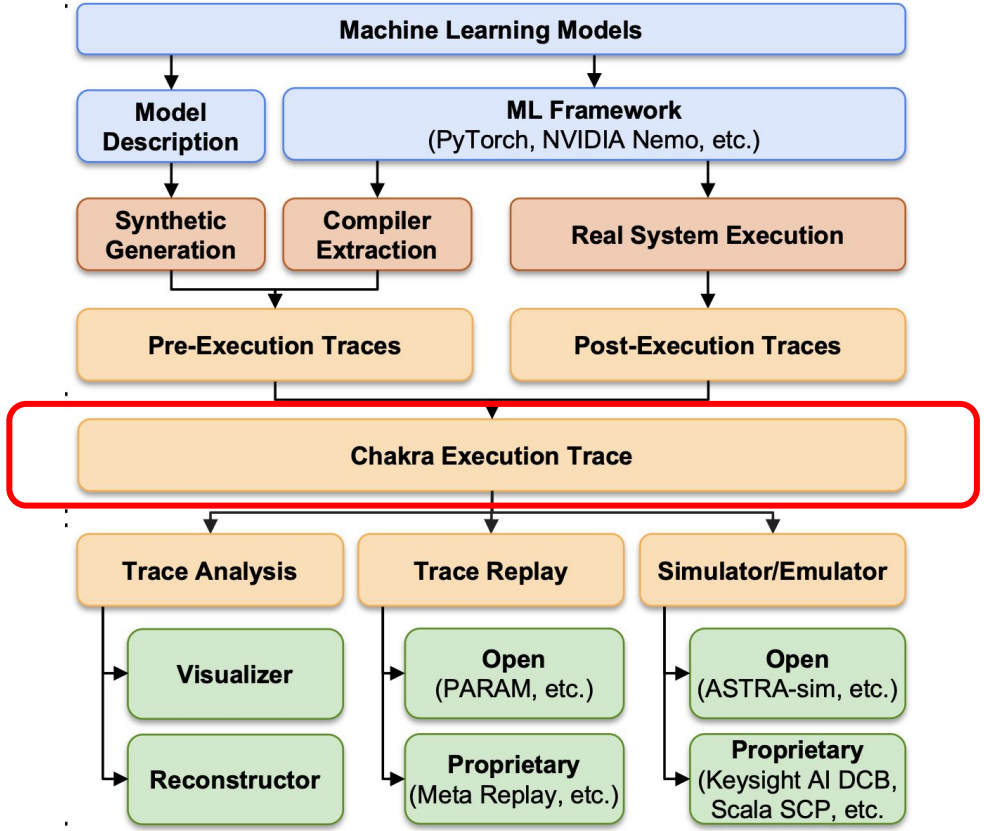
The Story

Goal: “A common graph format for co-design”

- **Chakra Execution Trace (ET)**: a mechanism to describe Distributed AI workload performance behavior in a AI system.
- Contains execution information without model params/secrets.
- Co-developed by Meta* and Georgia Tech in 2023, and effort started to standardize under MLCommons as a research working group.
- Today: 40+ member working group and ecosystem - hyperscalers, full stack/ compute vendors, research groups, emulation/simulation/testing vendors.

*https://engineering.fb.com/2023/09/07/networking-traffic/chakra-execution-traces-benchmarking-network-performance-optimization/

Chakra Ecosystem

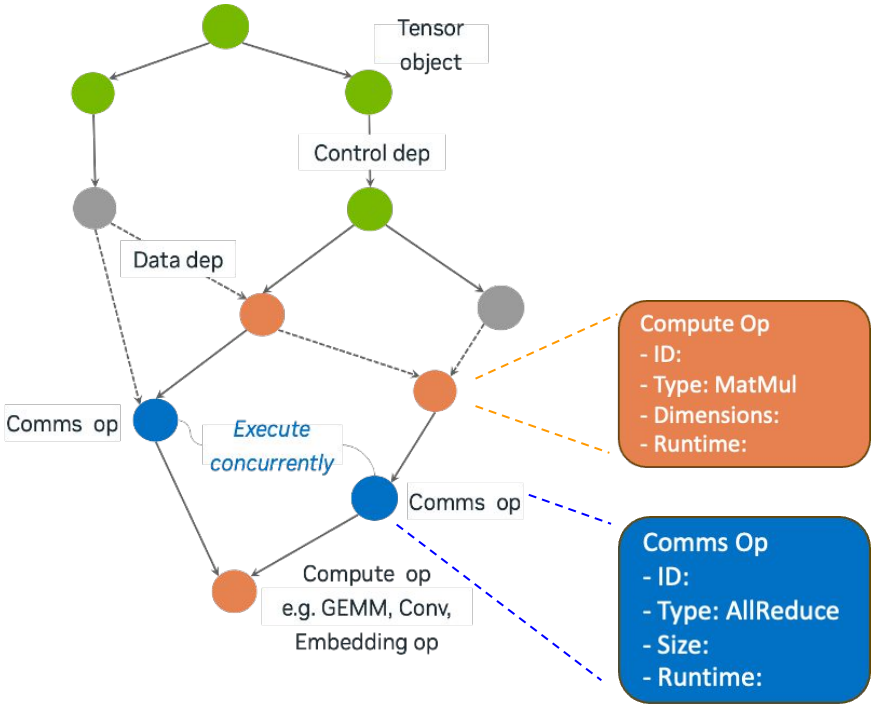


What is Chakra?

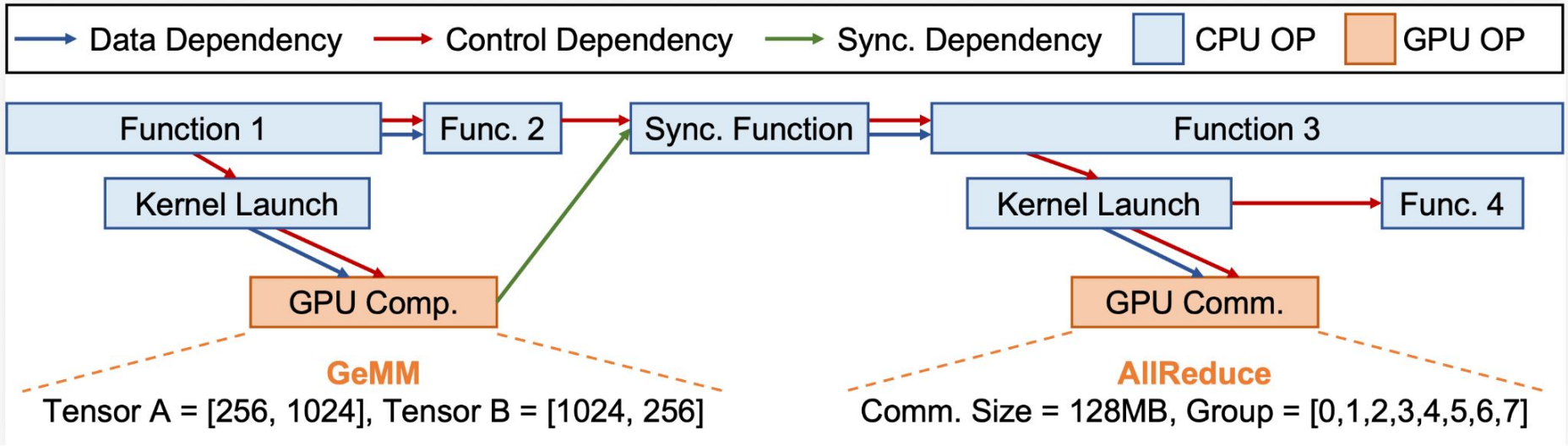
Essentially a DAG representation of an ML model Execution.

Fields	Description
name, id	Operator name, unique id
type	Compute/Comm
Ctrl deps, data deps	**Control (Caller-callee) / Data dependencies**
Input/Output tensors	Tensor in/outs.

Dependencies encode edges in the DAG

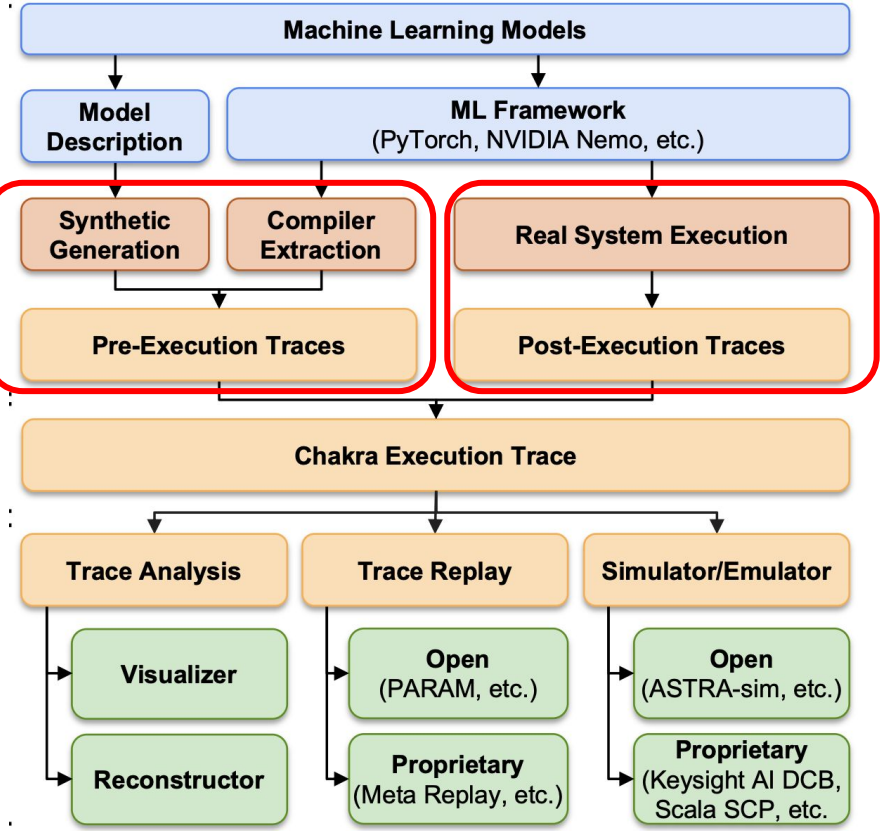


What is Chakra?



In a typical distributed AI job there is **one Chakra trace per rank or NPU**

Chakra Ecosystem



Collecting Chakra Traces

- **Pre-Execution Traces**: enables flexible simulation of the parallelization strategies without using actual systems.
  - Synthetic Generation (e.g., STAGE[1])
  - Compiler Extraction (e.g., Flint[2]: torch.fx to Chakra conversion).

- **Post-Execution Traces**: collected on real systems
  - but locked-in to that topology and parallelization strategy.

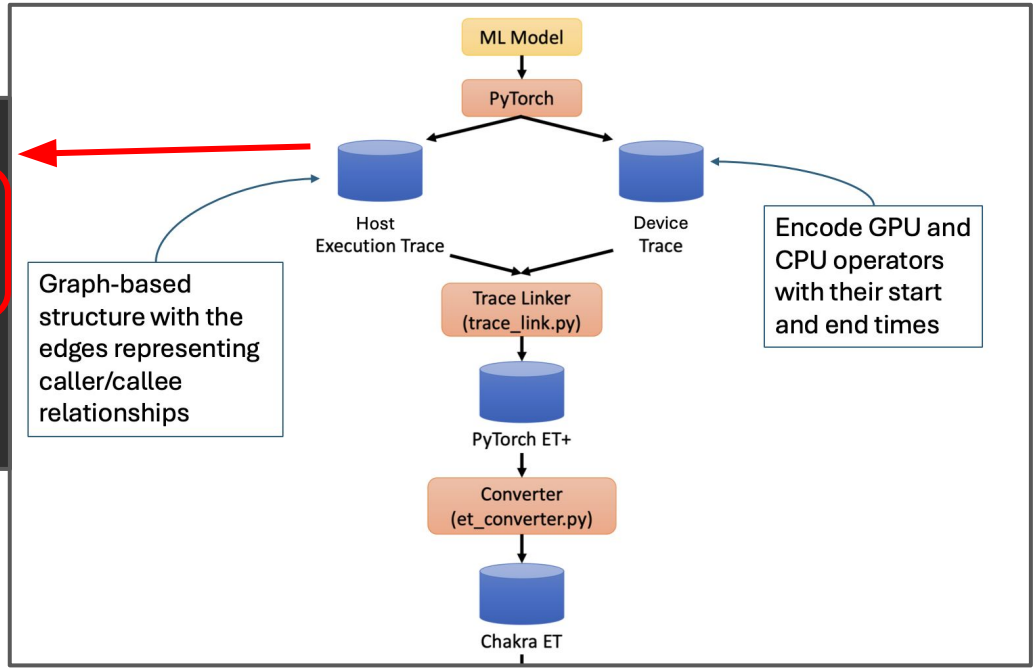
[1]Man et al., “STAGE: A Symbolic Tensor grAph GEnerator for distributed AI system co-design” , ISCA 2026. https://arxiv.org/abs/2511.10480

[2]Yoo et al., “Flint: Compiler Enabled Cluster-Free Design Space Exploration for Distributed ML”, https://arxiv.org/abs/2604.17550

Collecting Post-Execution Chakra Traces in PyTorch

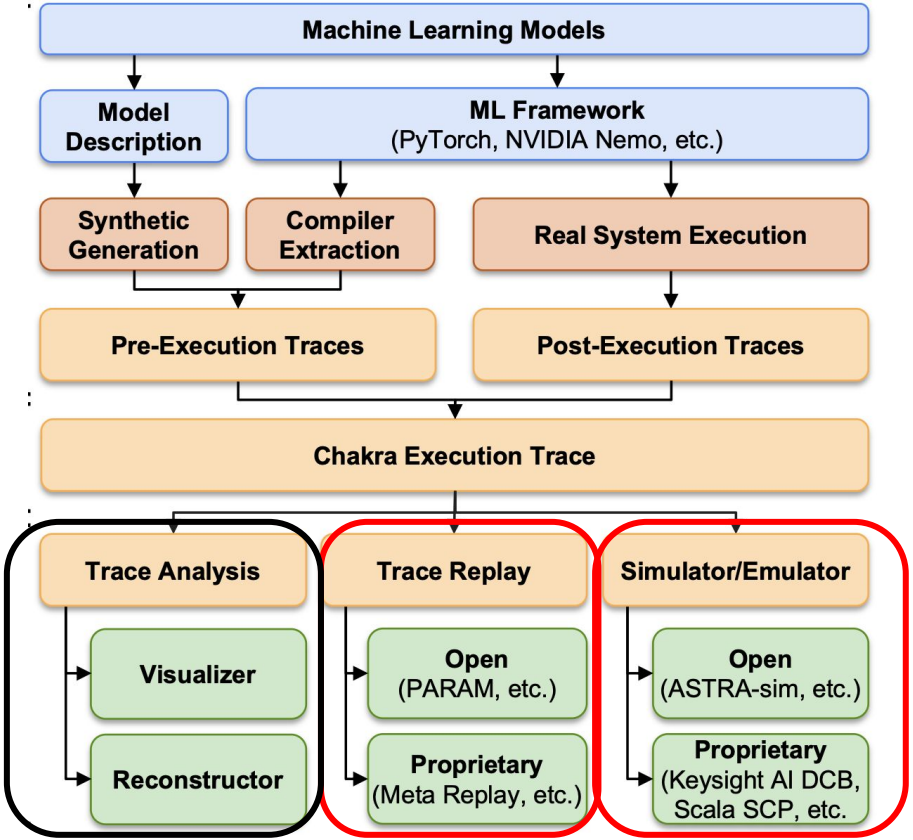
Easy and out-of-box support in PyTorch and frameworks (**NeMo**, **vLLM**)

```
with torch.profiler.profile(
    ...
    execution_trace_observer=(
        ExecutionTraceObserver().\
            register_callback("./execution_trace.json")
    ),
) as p:
    for iter in range(N):
        code_iteration_to_profile(iter)
        p.step()
```



*https://docs.pytorch.org/docs/2.12/profiler.html

Chakra Ecosystem



Use Case: Trace Replay

Trace Replay offers a scalable, model-free solution for re-executing Chakra traces.

Key Benefits:

- **Device Agnostic**
- **Data Privacy:** Uses randomly tensor data protecting proprietary models.
- **Fine grained control:** Isolate and execute specific operations
- **Rapid Setup**: Eliminates model initialization or job setup overheads.

Kernel	Size	Rks	Dur (ms)	BW Base (GB/s)	BW Ratio
ReduceScatter f32	9.0G	2	15.790	243.6	1.261
ReduceScatter f32	9.0G	2	57.378	157.7	1.106
AllGather	4.5G	2	7.848	289.2	1.065
AllGather	4.5G	2	7.491	312.8	1.034
ReduceScatter f32	404M	2	22.428	153.9	1.214
ReduceScatter f32	320.1M	2	0.778	174.6	1.236
ReduceScatter f32	256M	2	3.887	177.5	1.013
AllGather	202M	2	1.264	209.0	0.784
ReduceScatter f32	192.1M	2	0.494	170.3	1.197
AllGather	160.0M	2	0.395	164.6	1.291

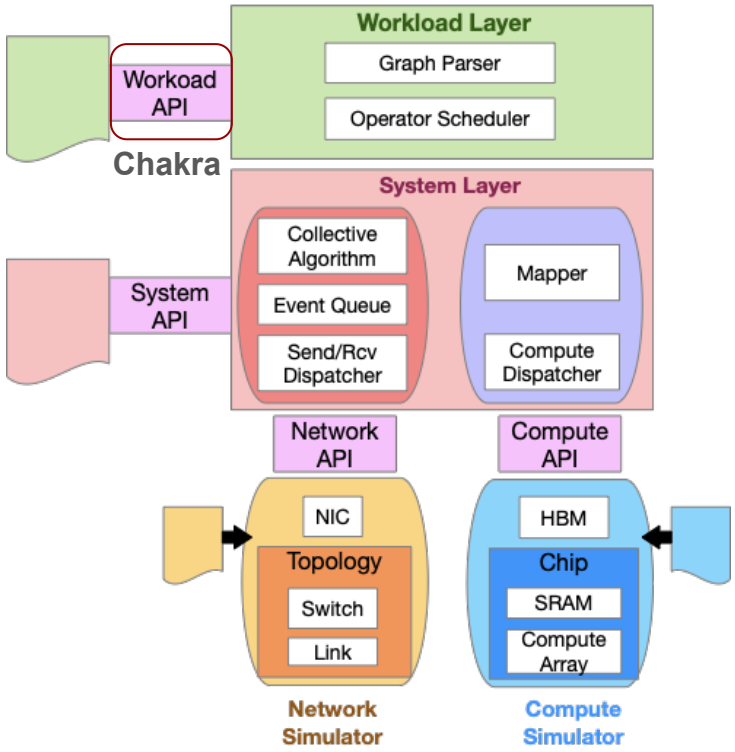
Communication replay comparison:
Megatron-LM 43B GPT model (48 layers) on four H100 nodes (32 ranks), PP=4, TP=4, DP=2

Use Case: Simulation

Simulation: “what-if-analysis” and co-design

- Open-source **Astra-sim** adopted Chakra trace as an input format
- **Enables co-design studies studying novel platforms**—spanning new fabric topologies, wafer-scale systems and topology-aware collective synthesis
- Enabled on proprietary simulators, e.g., **AMD Simulator, Scala Computing Platform*, ..**

https://astra-sim.github.io

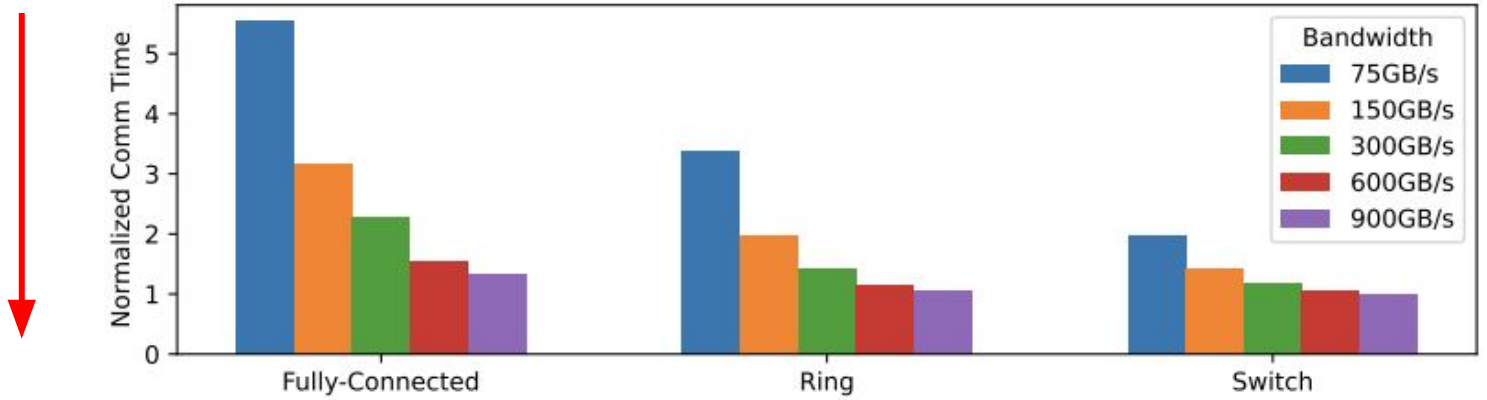


*https://www.scalacomputing.com/platform

Experimental Results: Simulation / Astra-sim

Normalized communication time by varying **network topology and bandwidth**

Lower is better



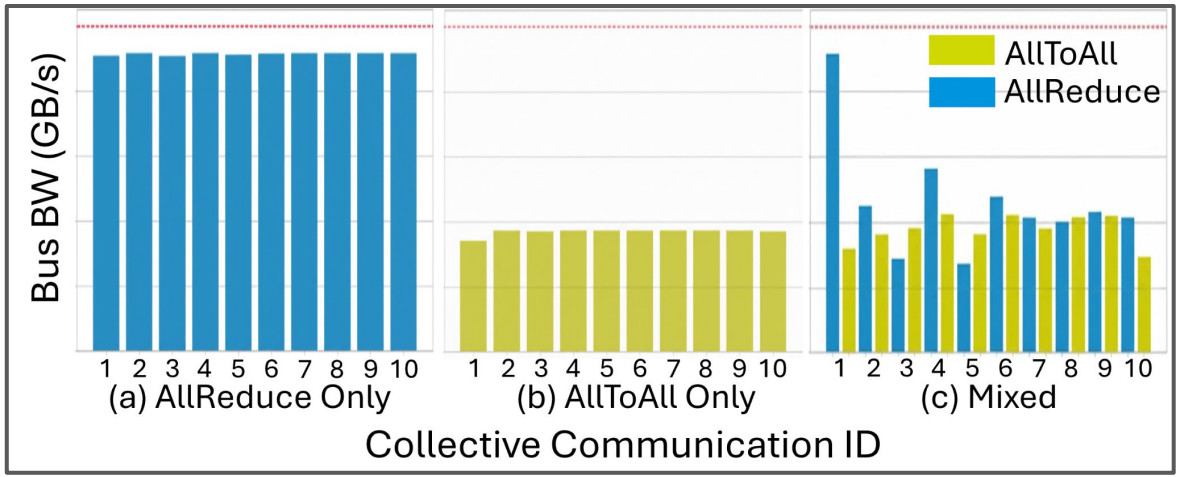
Simulating Mixtral8-7B with 8 H100s and varying network topologies using Astra-sim

Use Case: Hardware in Loop / Emulation

Emulation: *Shift left* validation from cluster level to component level

- **Chakra ET as a portable workload specification**
- An emulator replays the trace to generate at-scale network stimulus for a physical device under test (DUT) like a NIC or switch.Pioneering use in by **Keysight AI Datacenter Builder**

Higher is better



Future Work

- MLPerf Storage collaboration
  - Expanding Chakra to capture and represent Storage+Networking operations
  - Supporting **data loader, checkpointing** and **KVCache (Inference)**.
  - Replay and simulation of storage operations.
- Inference support
  - Supporting fused and graph mode kernels and KVCache operations.
- Core infrastructure
  - Handling large traces using compression.
  - Pairing Chakra with standardized infrastructure graphs (https://infragraph.dev/)

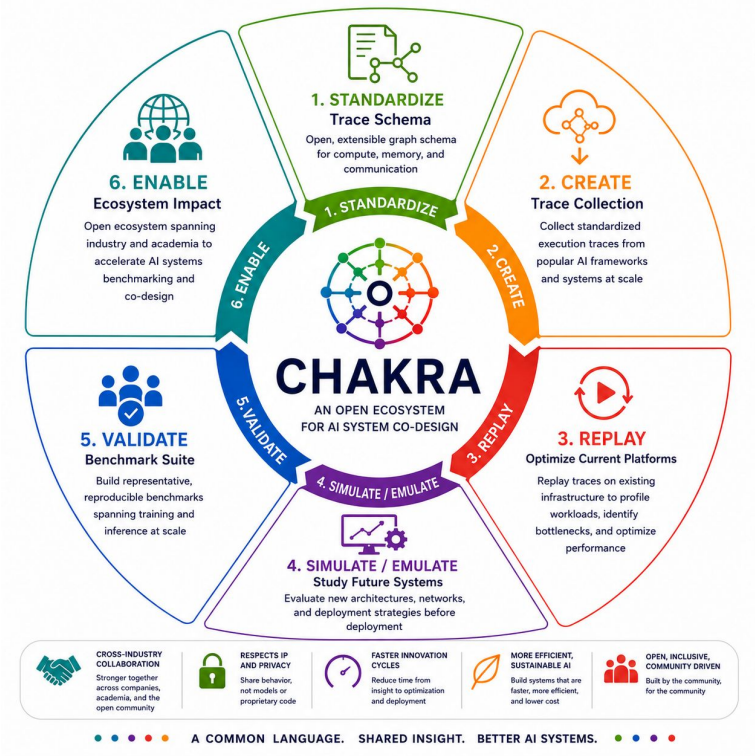
Conclusion

To recap

- Chakra positions itself as a common language for AI HW/SW co-design.
- It is an open standard with open source infrastructure and active community

Call to Action: Come join our working group and leverage and contribute to our tools to enable the next generation of seamless co-design for ML Systems

https://mlcommons.org/working-groups/research/chakra/
https://github.com/mlcommons/chakra



Resources for Community

Chakra Codebase: https://github.com/mlcommons/chakra

Chakra Wiki: https://github.com/mlcommons/chakra/wiki

Library of Chakra Traces:
https://github.com/mlcommons/chakra/wiki/Chakra-Trace-Library

Backup

What is Chakra?

Essentially a DAG representation of an ML model Execution.

Chakra Node

Fields	Description
name, id	Operator name, unique id
type	Compute/Comm
ctrl deps, data deps	**Control (Caller-callee) / Data dependencies**
Input/Output tensors	Operator in/output tensor

Compute Nodes

Communication Nodes

Comm Fields	Description
Comm type	Collective(AllReduce etc) or Point-point ops
Group	Process group (DP/PP/EP)
Tensor ids	Tensor in/outs, strides

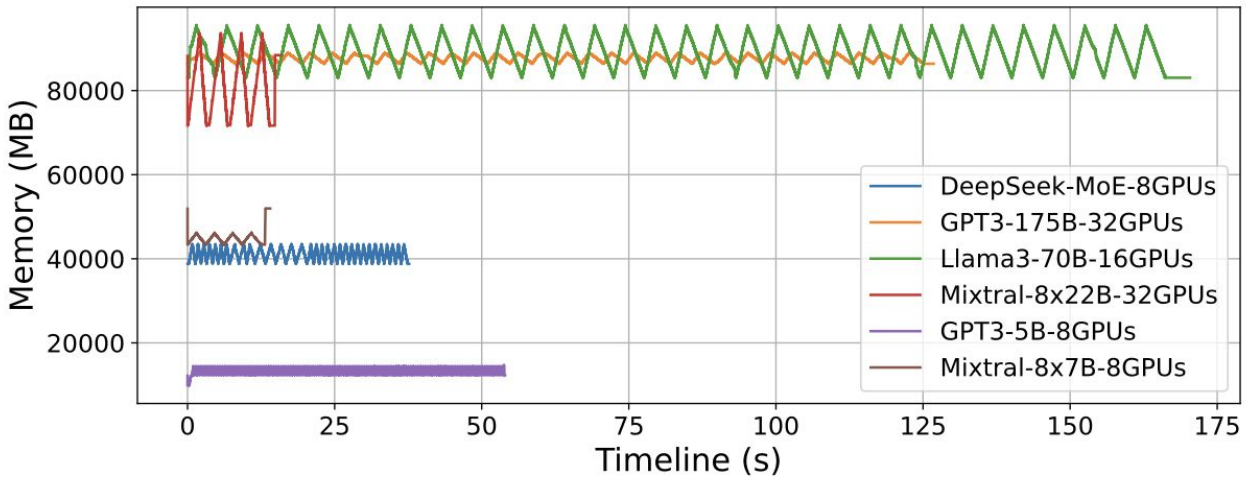
Dependencies encode edges in the DAG

Downstream Use-Cases

Consumer	What it does	Where it helps
Visualizer	Shows dependency graph and timeline structure	Debugging, teaching, trace validation
ET Feeder	Streams ready nodes to a simulator while preserving partial order	Large traces, bounded memory, pluggable scheduling
Replay	Re-executes compute and communication operations on real systems	Benchmarking, tuning, sub-trace isolation
Simulation	Projects workload behavior onto future systems and topologies	Pre-silicon what-if studies
Emulation	Generates realistic traffic against physical DUTs	Hardware-in-the-loop validation

Experimental Results: Trace Analysis

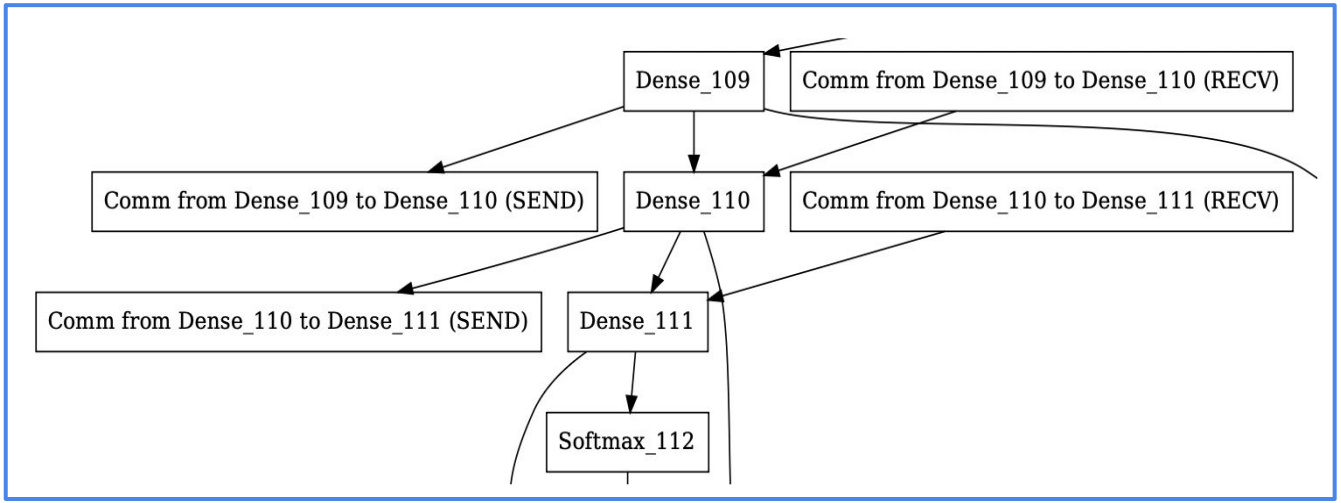
Memory utilization computed using Chakra Nodes



Use Case: Trace Analysis

Chakra trace can be leveraged as a standard profiler: bottlenecks, computing critical path, understanding overlap between compute/communication

ET Visualizer



ET Feeder

Provide ET nodes to a simulator/emulator honoring dependencies and specific policies.

Experimental Results: Trace Analysis

Evaluation System: 128 NVIDIA H100/200 GPUs, dual 32-core Intel Sapphire Rapids CPUs, and DDR5 DRAM. PyTorch 2.5, NVIDIA NeMo 24.07, and Megatron 0.10.0.