IMPERIAL

NVIDIA

MLSynth: Towards Synthetic ML Traces

Adel Sefiane

Imperial College London

NVIDIA

Alireza Farshin

NVIDIA

Marios Kogias

Imperial College London

08/09/2025

Networking for AI Computing (NAIC)

The ML/Networking research gap

AI Factories continue to scale and networking is becoming more of a bottleneck.

Networking solutions optimised for AI training has gained steam

- Congestion control
- Scheduling
- QoS

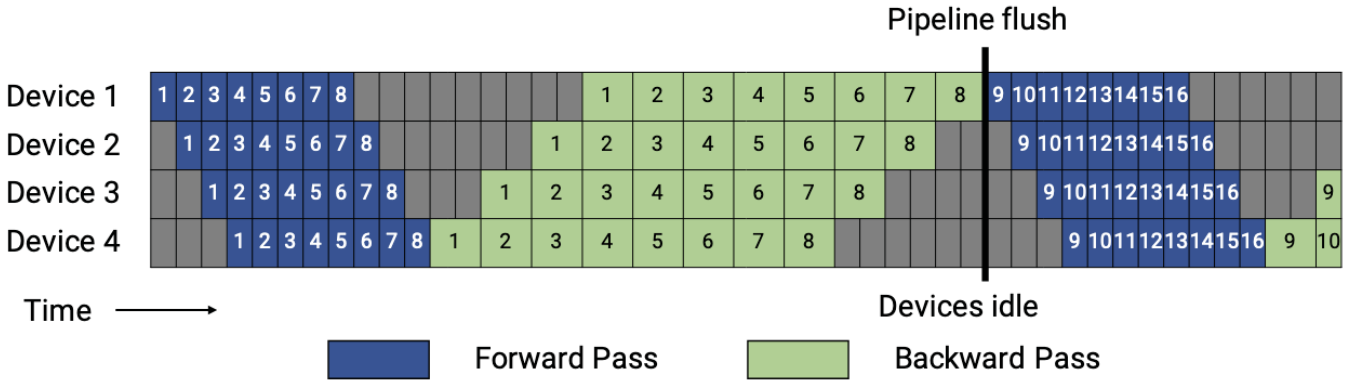
Networking papers do not leverage AI workloads.

ML papers do not leverage networking simulators.

There is a disconnect between networking research and ML research

What is so special about ML workloads?

ML Training workloads are deterministic, with sequential dependencies and large flows.



GPipe communication pattern[1]

ML Training workloads are highly predictable and are defined by their dependencies

[1] Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM, Narayanan et al.

Option 1: Real test beds

Run your solution on a real test-bed.

Used in many ML-centric papers.

- Cassini (NSDI ‘24)
- MLTCP (HotNets ‘24)

✅ Gold standard accuracy

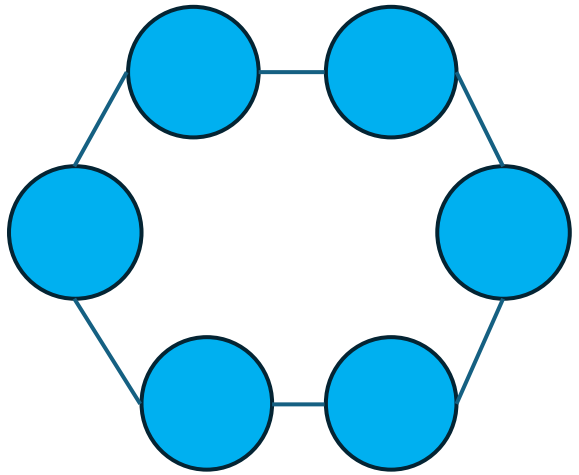
❌ Costly, unsuitable for iterative development

Not practical

Option 2: Analytical workloads

Simulate traffic patterns at flow level or analytically.

Captures high level traffic patterns (all-reduce, all-to-all, etc.)



Ring topology

✅ Quick & transparent

❌ Doesn’t model packet-level dynamics

Inaccurate

Option 3: Workload traces

Use a workload representing ML training in simulation

Use isolated traffic patterns (i.e. incast, allreduce)

- ParaLet (NAIC ‘24)
- ACC (NSDI ’24)

✅ Quick and easy

❌ Inaccurate

Inaccurate

Collect a trace of a workload from a test-bed.

- ASTRA-sim & Chakra
- SimAI

✅ Accurate

❌ Each trace is specific to the test bed setup - not tunable!

Lack of control

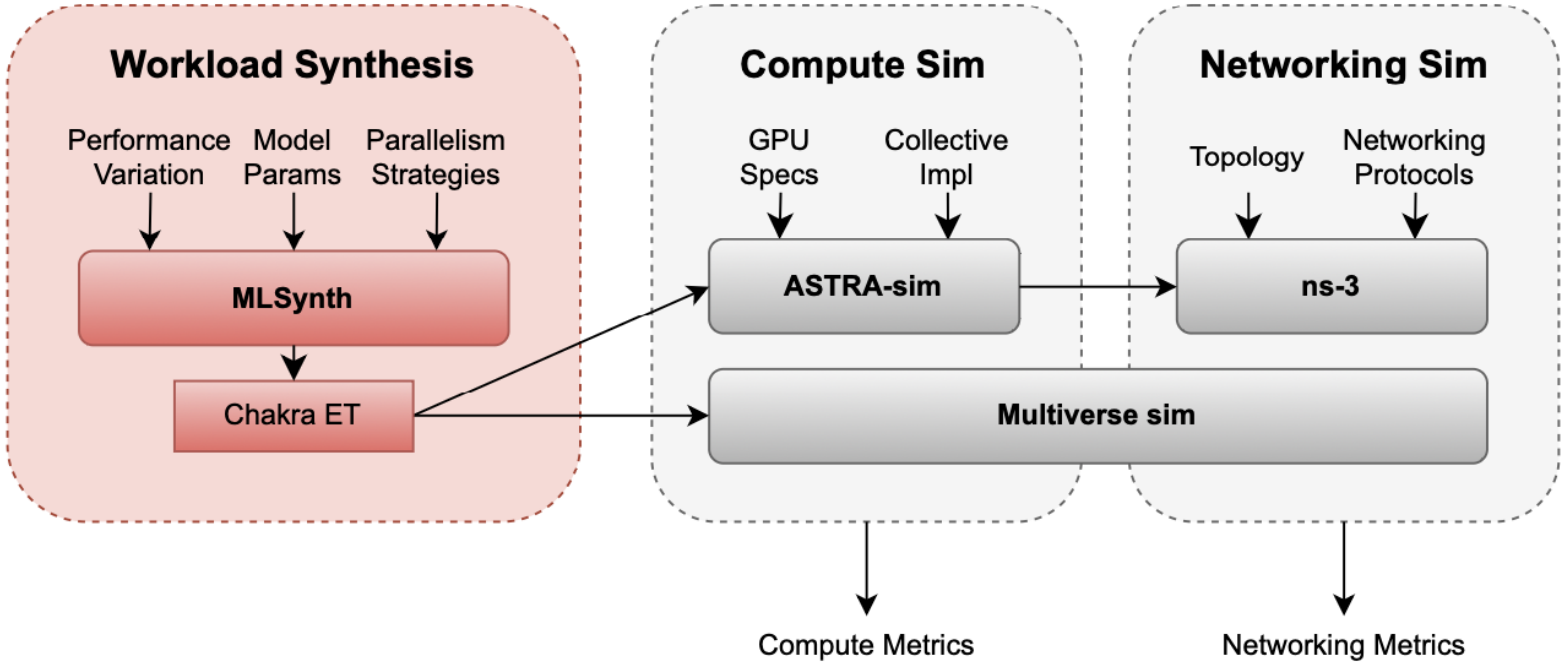
MLSynth: Synthesise ML workloads

✅ **Accurate**: captures workload nuances
- Models FLOPs and dependencies between operations

✅ **Reproducible**: workload is clearly defined by model config
- Workload definition is agnostic to underlying hardware

✅ **Tunable**: full control over the workload
- Simulate GPU & NIC slowdown with performance variation functionality
- Scale the workload to shorten runtime

Where does MLSynth fit in?



MLSynth-generated workloads are compatible with most new simulators!

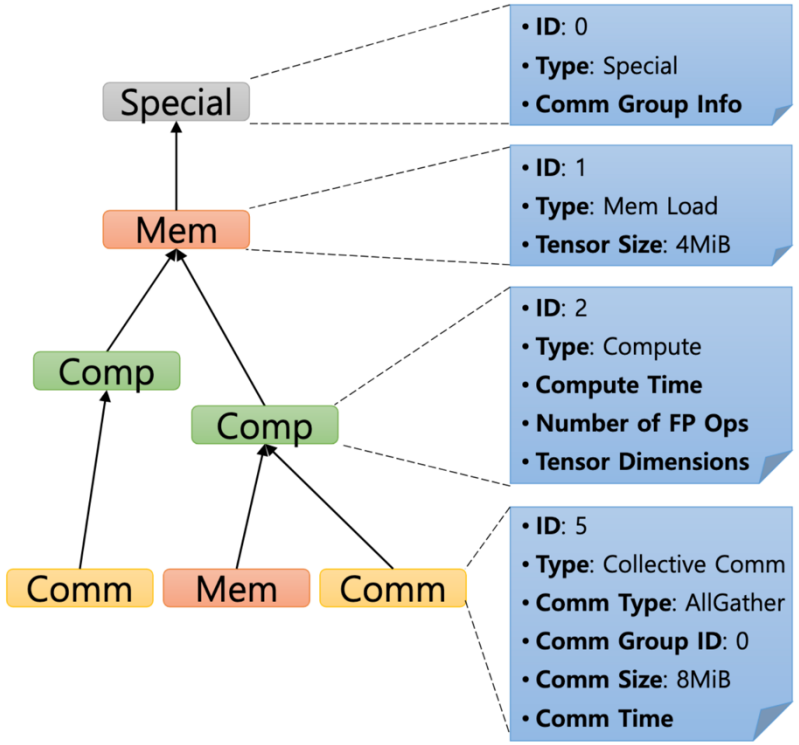
Accuracy: How are ML workloads represented?

Chakra Execution Traces

- Directed Acyclic Graph of compute & communication operations
- Part of MLCommons

Adopted as the standard for new simulators:

- ASTRA sim
- Multiverse sim



Chakra execution trace [1]

[1] Chakra: Advancing Performance Benchmarking and Co-design using Standardized Execution Traces, 2023

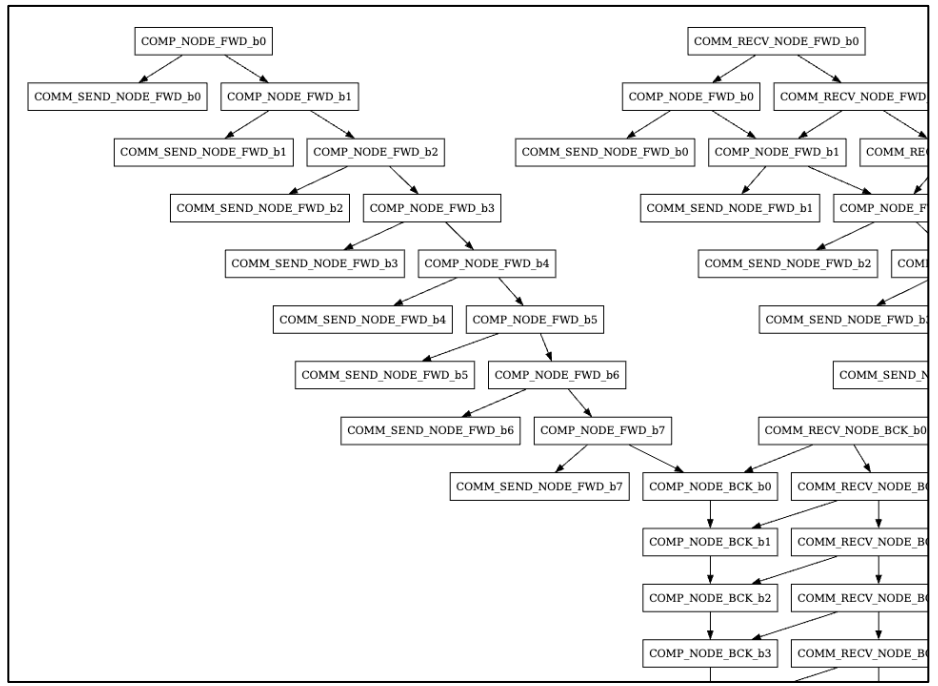
How do I use MLSynth?

GPT 175-B

Vocabulary size: 51,200
Sequence length: 2,048
Hidden dimension: 20,480
Batch size: 8

Parallelisation Strategy

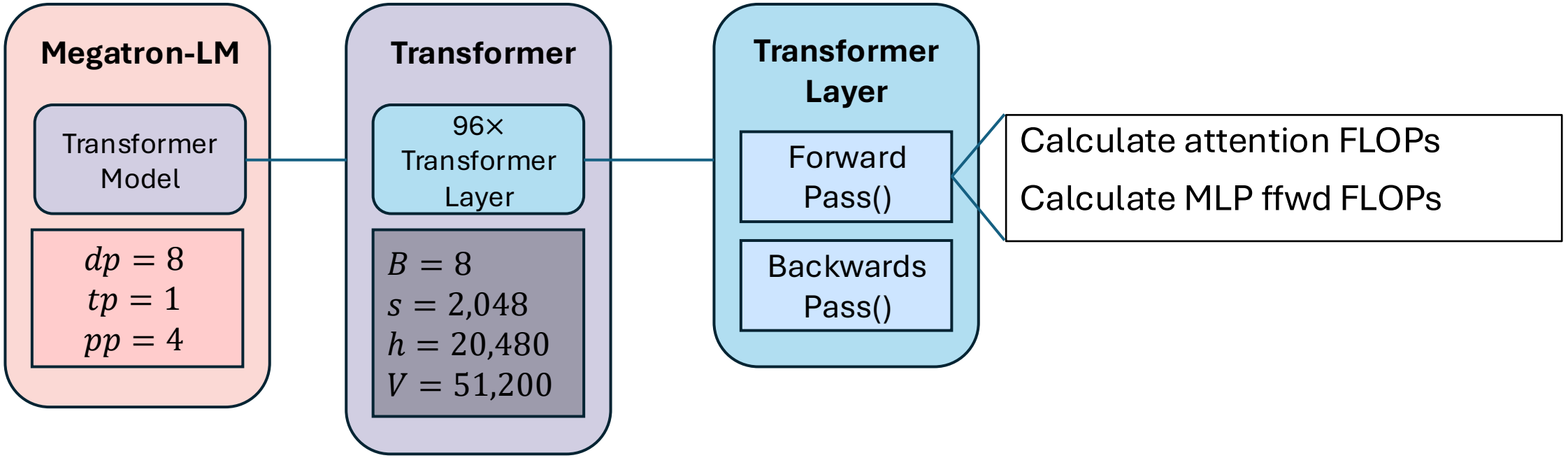
Pipeline parallel: 4
Tensor parallel: 1
Data parallel: 8



Your Input
Model config

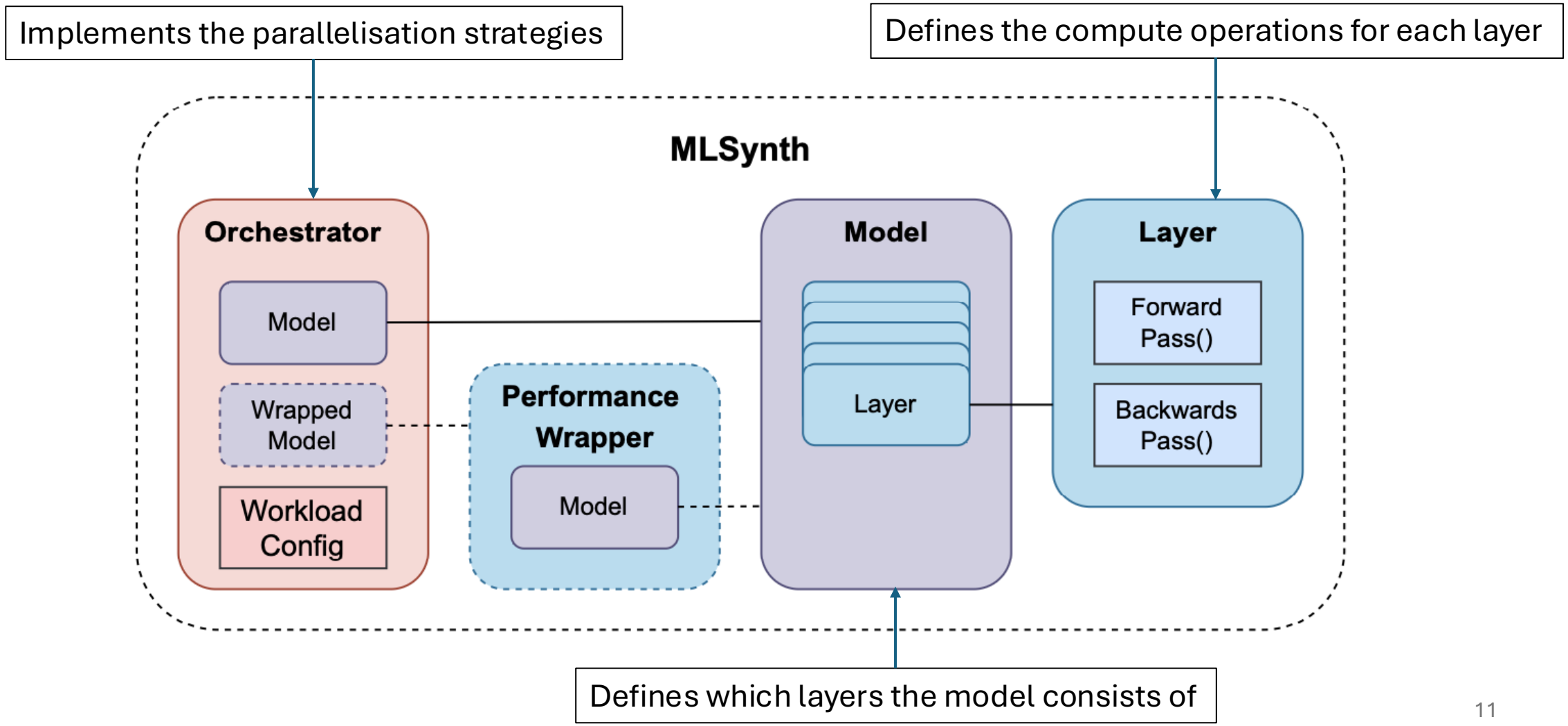
MLSynth's Output
Chakra Execution Trace

Reproducibility: How the Transformer model is implemented in MLSynth



B = batch size, s = sequence length, h = hidden dim, v = vocabulary size

What if I want to implement my own model?



Tunability: Performance Variation

ByteDance reported 10.4% of GPU hours wasted due to stragglers and slowdowns[1]

How can we simulate:

- A specific point of failure
- Stochastic performance variation

Before: Modify simulators to induce desired behaviour

Now: Inject performance variation directly into workload, agnostic of simulator

GPU slowdown

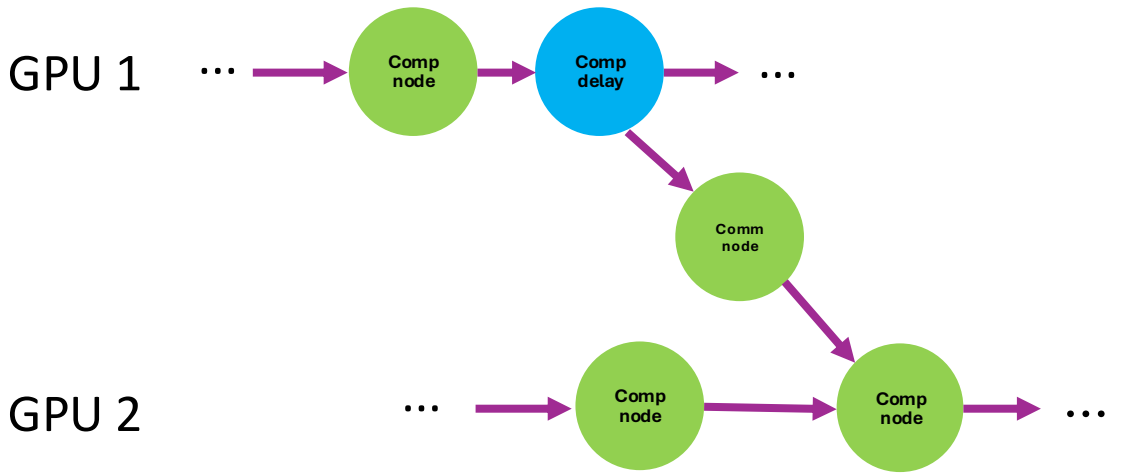
NIC delay

[1] Understanding Stragglers in Large Model Training Using What-if Analysis, Lin et al. 2025

How is performance variation implemented?

Example: let's simulate a GPU experiencing slowing (e.g. thermal throttling).

- Simulate reduced performance by increasing the number of FLOPs
- Performance wrapper modifies the generated compute nodes to enact delays

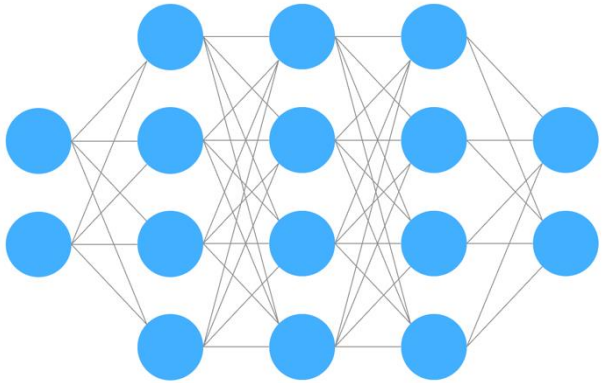


Added compute delay to simulate reduced GPU performance

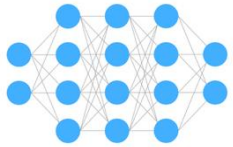
Tunability: Scaling your workloads

Q: My experiment is taking days to run… Can I run a smaller workload to iterate faster on my work?

A: Yes! MLSynth includes a 'scale' parameter that allows you to linearly scale the size of the model being trained (GPT 175B → GPT 175M)



GPT 175-B

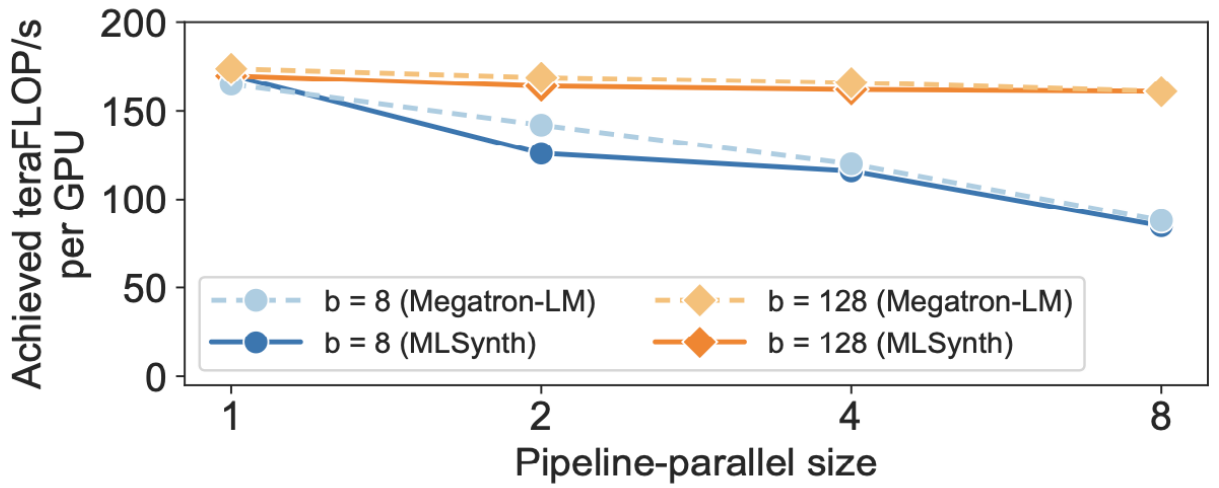


GPT 175-M

An experiment that took 24 hours to run at full scale was accurately replicated in 23 minutes at 1% scale with MLSynth's workload scaling!

Can you trust synthetic workloads?

Q: Do MLSynth-generated workloads exhibit the same characteristics as real-world workloads?
A: Yes!



Replicate a scaling experiment from the Megatron-LM paper:

Scale the number of pipeline stages and observe GPU use efficiency.

- One pipeline stage: 1 DGX A100 node (8 GPUs)
- Nodes connected in a three-level fat tree

Megatron-LM setup: Selene supercomputer

MLSynth setup: One (1) generic computer

MLSynth: sythnesise your future now!

Synthesise an accurate ML workload into a Chakra execution trace from generic model parameters.

1. Synthesise a workload (e.g. GPT 175B) with any parallelism (e.g. pipeline + data)
2. Feed the workload to a simulator (e.g. ASTRA-sim)
3. Happy researching!

MLSynth Currently Includes:

- Transfomer model (standard & mixture-of-experts)
- Megatron-LM parallelisation strategies
- Performance variation

NetMLSim/MLSynth