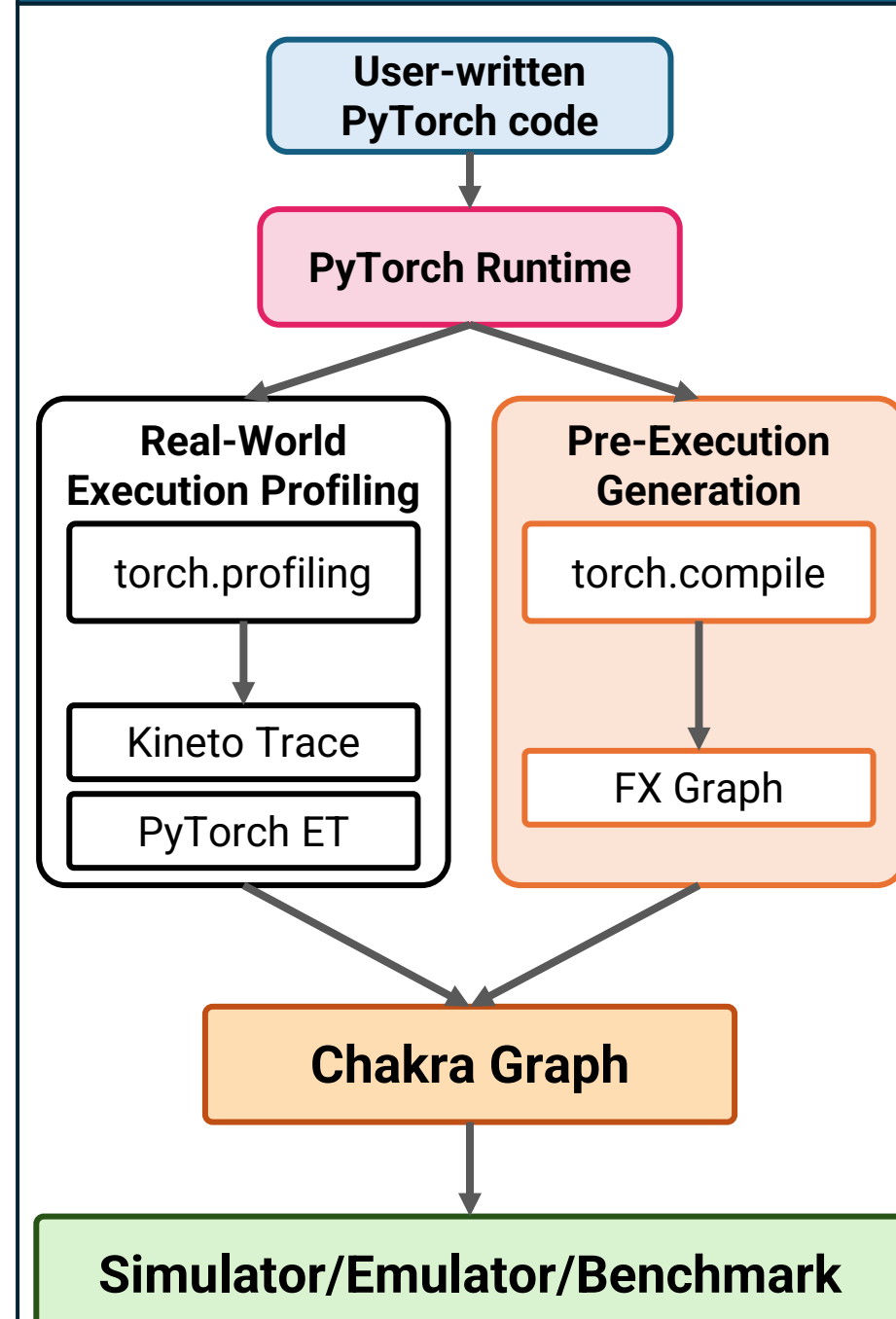


MOTIVATION

- Simulators and trace replay are essential in designing and optimizing HW/SW systems for distributed ML
- These tools need a **description of the workload**: They cannot just run the PyTorch code as-is
 - What operations are there?
 - What are the dependencies between the operations?
 - What does the input/output tensor look like?
- Even for the **same model**, the operation, scheduling, and ordering **change drastically** across parallelization strategies and workload optimizations
 - Hand-written workload descriptions don't scale well

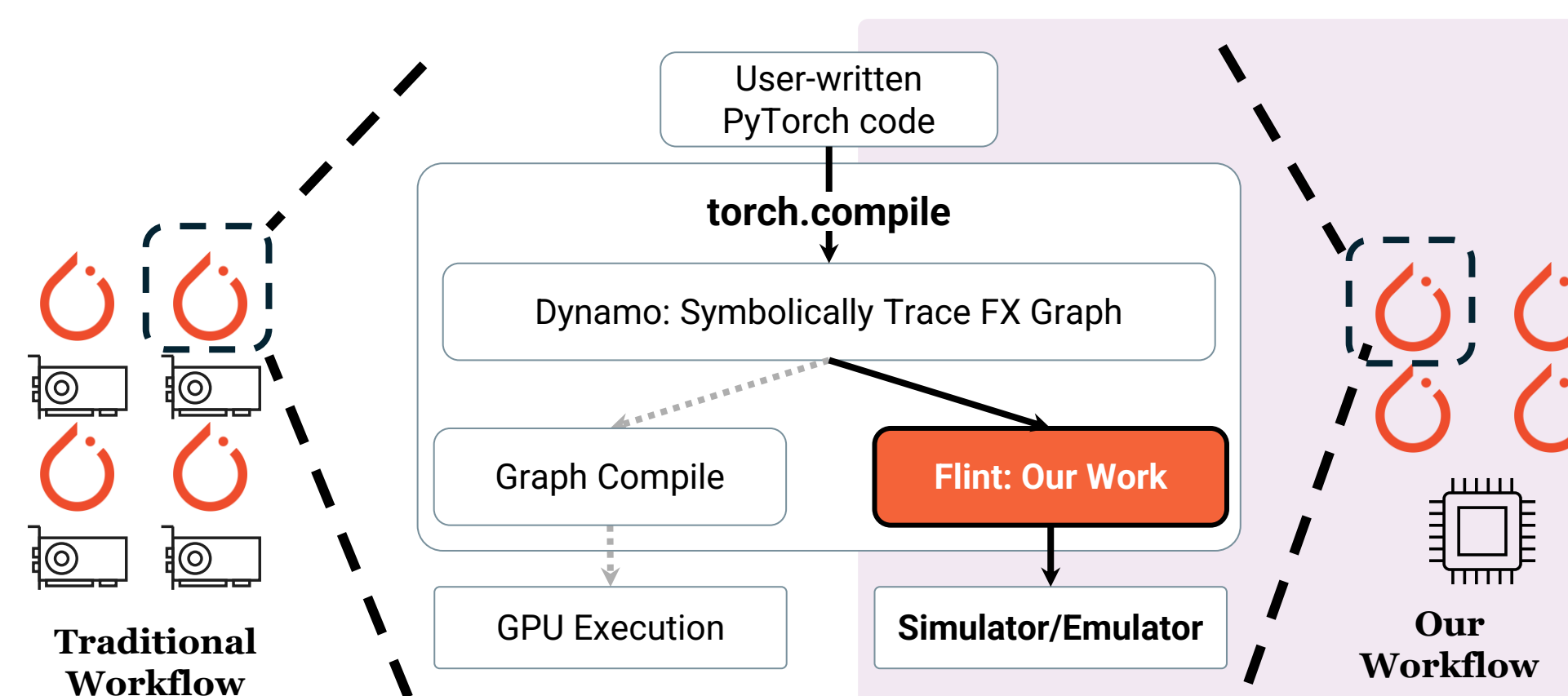
How can we easily obtain workload descriptions for various usecases?

BACKGROUND – CHAKRA ECOSYSTEM



- Chakra**: Graph based representation/trace format for workloads
- Integrated** into PyTorch profiling ecosystem
- MLCommons** Working Group: Several usecases across academia/industry

METHODOLOGY



[Key Idea]

- Leverage **torch.compile** to capture the **FX Graph**
 - FX Graph: Graph based IR. Symbolically traced JIT before running on the GPU
- Execution model: Launch unmodified PyTorch programs with torchrun on CPU only. Exit after graph capture
- Convert the graph to Chakra Traces and feed to simulator/emulators

[Challenges]

- High CPU memory use -> Symbolic trace with FakeTensors
- CPU kernels recorded(e.g., FlashAttention broken to CPU ops) -> Modify Dispatcher to dispatch GPU ops



Paper (arxiv)



GitHub

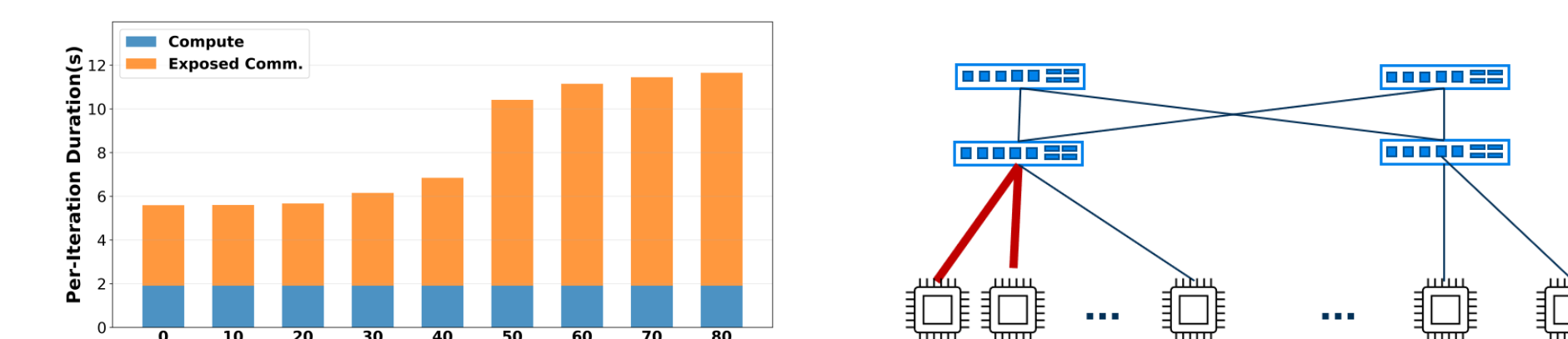
USECASES

I. Existing Systems

Compare compute-communication overlap strategies in SimpleFSDP

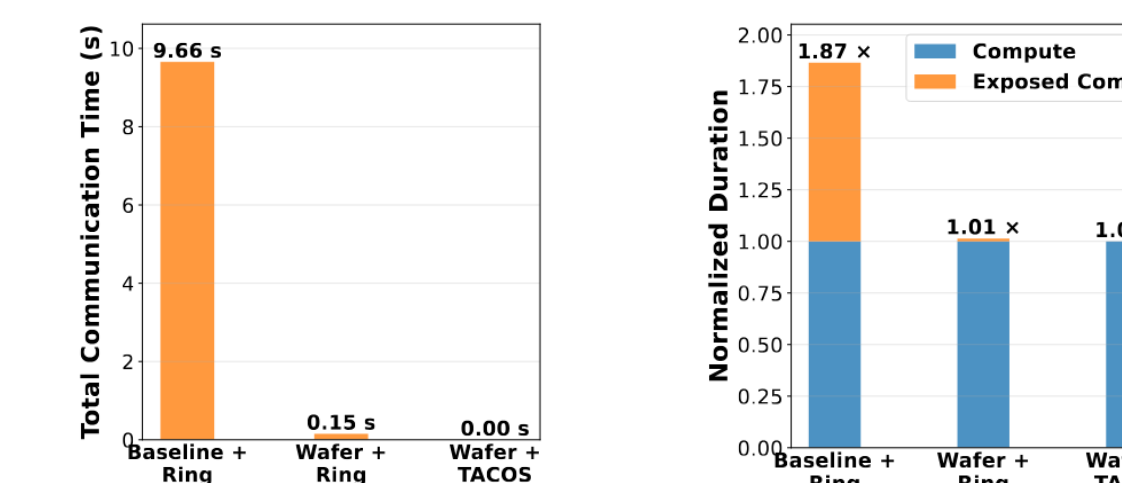


Emulate GPU endpoints to drive real traffic into real network fabric without GPUs



II. Future Systems

Glass-Packaging Systems: Collective algorithm study for future largescale glass-packaging systems



FUTURE WORK

- Inference**: End-to-end integration with inference engines (vLLM, etc.)
- Dynamic Workloads**: Capture data-dependent operations (MoE A2A, etc.)
- CleanerIntegration**: Direct dispatcher without modifying PyTorch

MOTIVATION

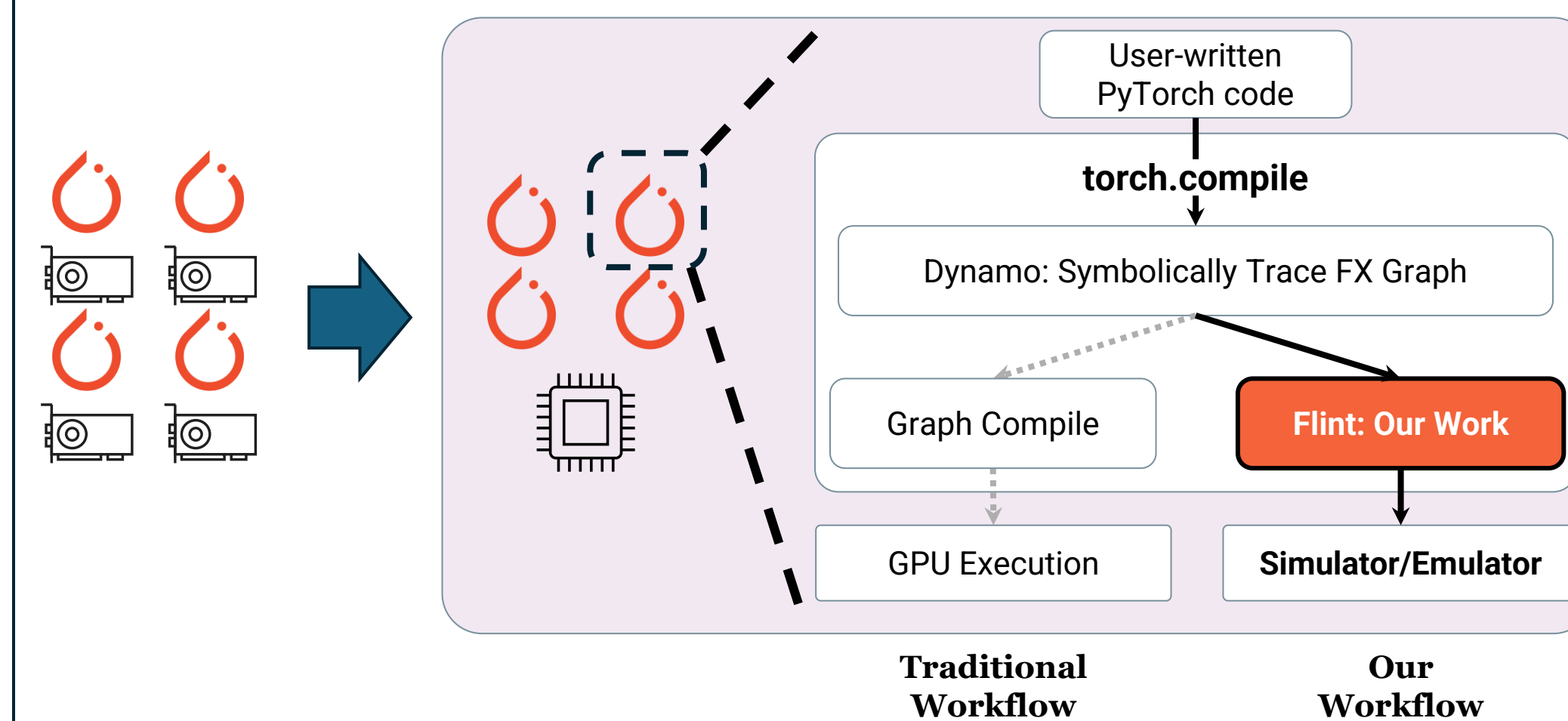
- Simulators/Emulators are essential in designing Hardware/Software systems for distributed Machine Learning
- These tools need a description of the workload: You cannot just run the PyTorch code as-is
 - What operations are there?
 - What are the dependencies between the operations?
 - What does the input/output tensor look like?
- Even for the same model, the operation, scheduling, and ordering may change drastically in a distributed environment across parallelization strategies or workload optimization
 - A hardcoded list does not scale well

How can you easily get a workload description for various usecases?

EXISTING APPROACHES

- **Symbolic Generation:** STAGE, AICB, etc. Need to manually extend tool at every new model
- **GPU API Intercept:** Phantora, Maya, etc.
 - Cannot capture true dependency of operations
- **Loop IR Capture:** PyTorchSim, etc.
 - More focused on compute

METHODOLOGY



[Key Idea]

- Leverage torch.compile to capture FX Graph
 - FX Graph: Graph based IR. Symbolically traced JIT before running on the GPU
- Execution model: torchrun the same PyTorch processes, but on CPU only. Exit after Graph Capture
- Convert graph to Chakra Traces. Feed to simulator/emulators
 - **Chakra Trace:** Graph based workload representation used by several toolchains.

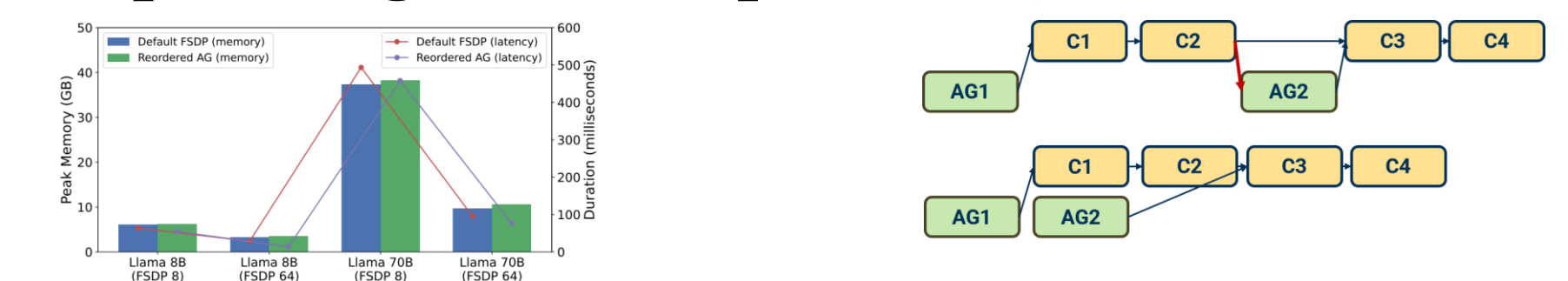
[Challenges]

- CPU Memory Consumption -> Use FakeTensors to symbolically trace the FX Graph
- CPU-specific operations recorded (e.g. FlashAttention broken to CPU ops) -> Modify Dispatcher to dispatch GPU ops

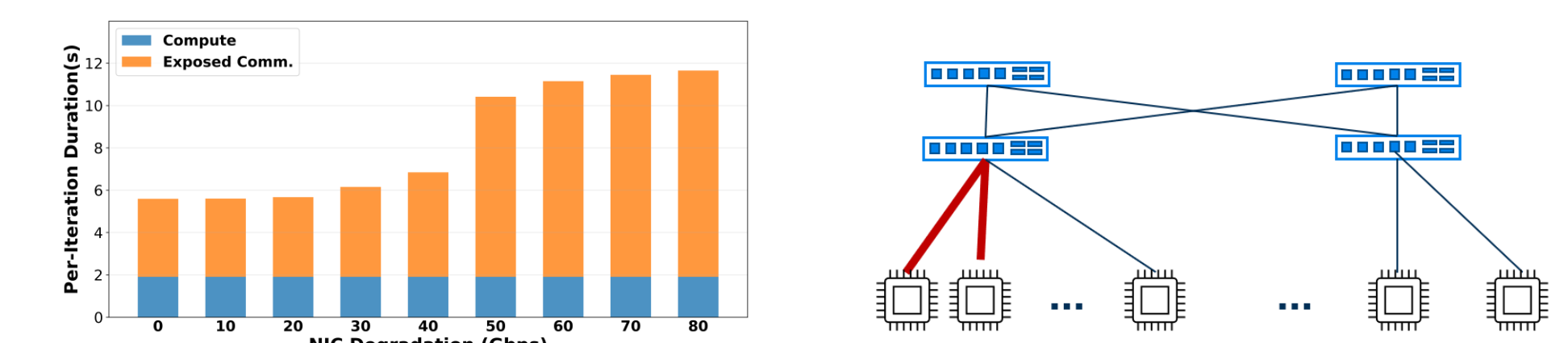
USECASES

I. Existing Systems

Compute-Communication Overlap: Explore different overlap strategies in SimpleFSDP

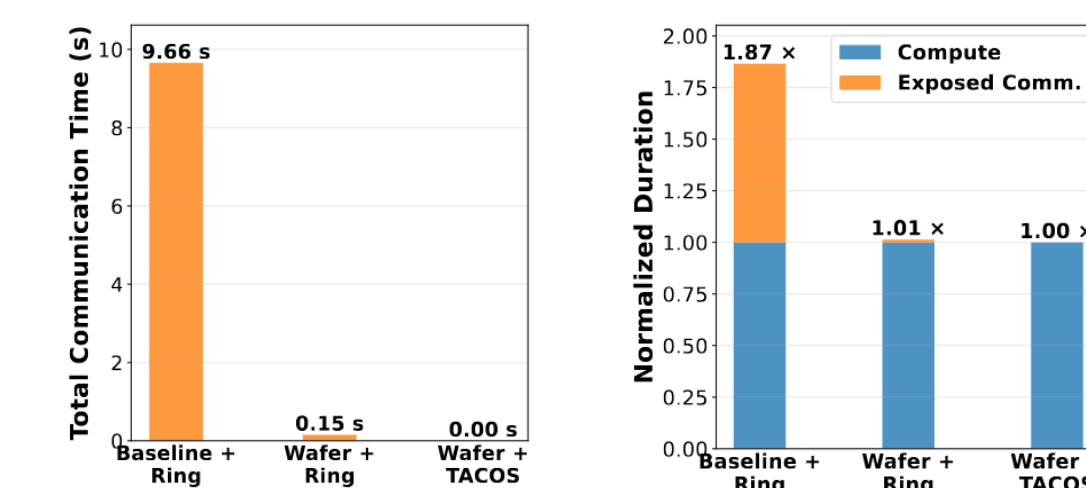


Real Network Study: Emulate GPUs and inject real traffic into real network fabric without GPUs



II. Future Systems

Glass-Packaging Systems: Algorithm study for future largescale glass-packaging systems



100,000-scale GPU Cluster: Workload behavior simulation for very large scale clusters



Paper (arxiv)



GitHub

FUTURE WORK

- **Inference:** End-to-end integration with Inference Systems (vLLM, etc.)
- **Dynamic Workloads:** Capturing data-dependent operations (MoE A2A, etc.)
- **Better Integration:** Directing dispatcher without modifying PyTorch