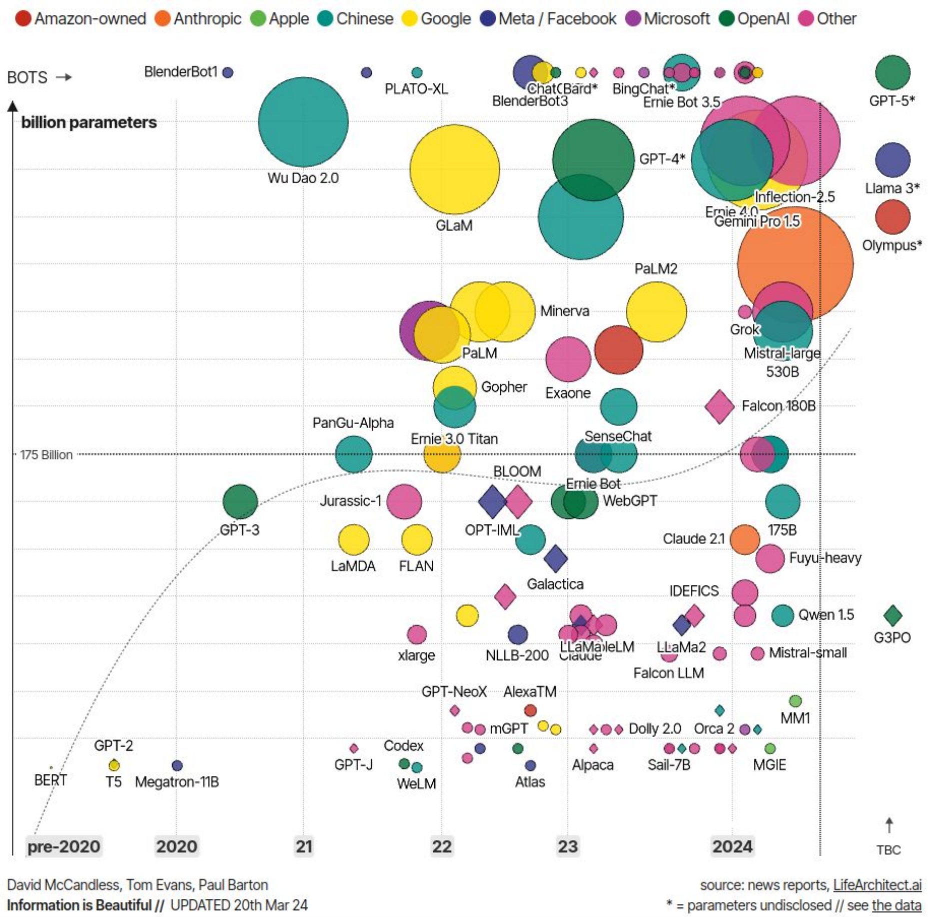
DataStates-LLM: Lazy Asynchronous Checkpointing for Large Language Models

HPDC'24. Authors: Avinash Maurya, Robert Underwood, M. Mustafa Rafique, Franck Cappello, Bogdan Nicolae

Avinash Maurya
Postdoctoral Researcher
Mathematics and Computer Science Division, Argonne National Laboratory

University of Maryland, SysML Seminar, 22nd April 2025

Motivation: LLM Pre-Training is Expensive



Model	Number of GPUs	Duration
GPT-3 (175B)	10,000	34 days
GPT-4	25,000	Several months
PaLM (540B)	6,144	2 months
Turing NLG	560	Several months
Bloom (176B)	384	3 months
Chinchilla (70B)	4,096	1 month
T5 (11B)	1,024	1 month

LLM pre-training: How much does it cost?

Model Size (B)	Tokens (Trillion)	Aurora Time (h)	Polaris Time (h)	Aurora Time (Days)	Polaris Time (Days)	Cloud Cost ($3 GPU/hr)
7	2	2.29	333	0.10	14	$437K
7	3	3.34	500	0.14	21	$656K
70	2	22.88	3,333	0.95	139	$4,374K
70	3	34.31	5,000	1.43	208	$6,561K
200	6	196.08	28,571	8.17	1,190	$37,496K
200	10	326.80	47,619	13.62	1,984	$62,494K
1000	10	1633.99	238,095	68.08	9,921	$312,470K
1000	20	3267.97	476,190	136.17	19,841	$624,941K

LLM Pretraining is Resource-intensive & Time-consuming

Datacenter Traces Reveal Urgent need for Efficient Resilience

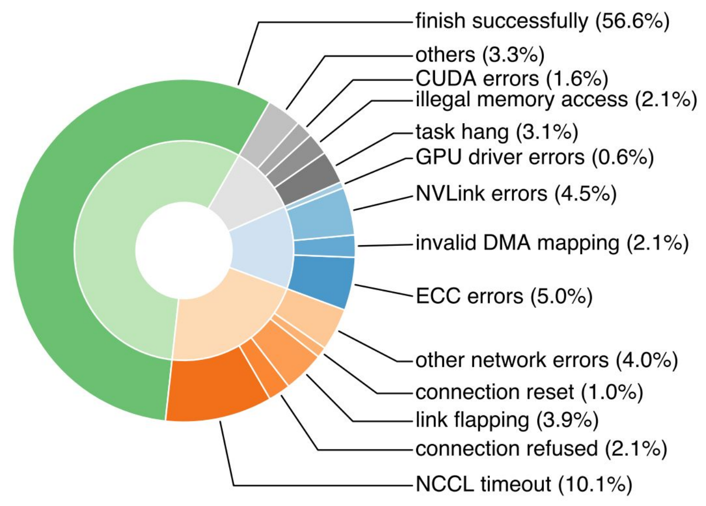


Fig: Failures on Alibaba Cloud consisting of 256 NVIDIA H800 GPUs running LLM training*

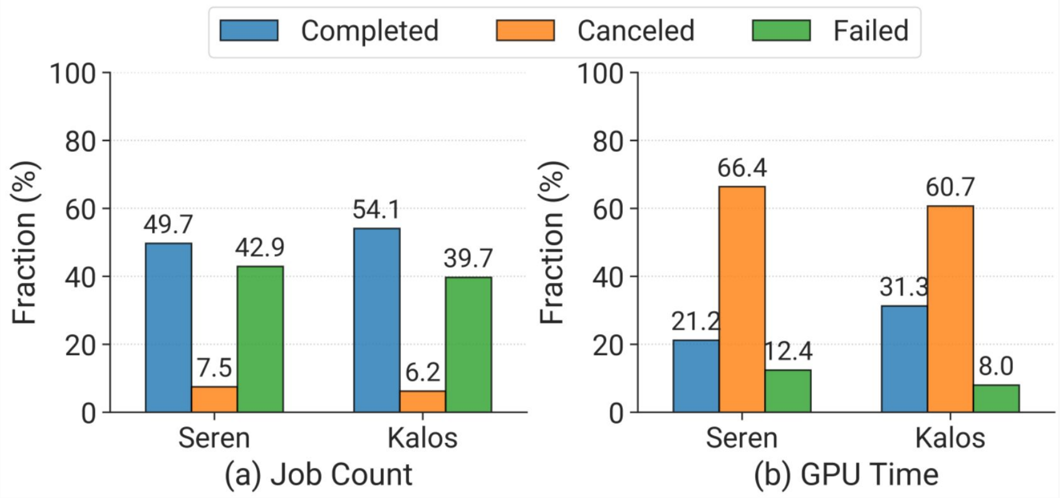
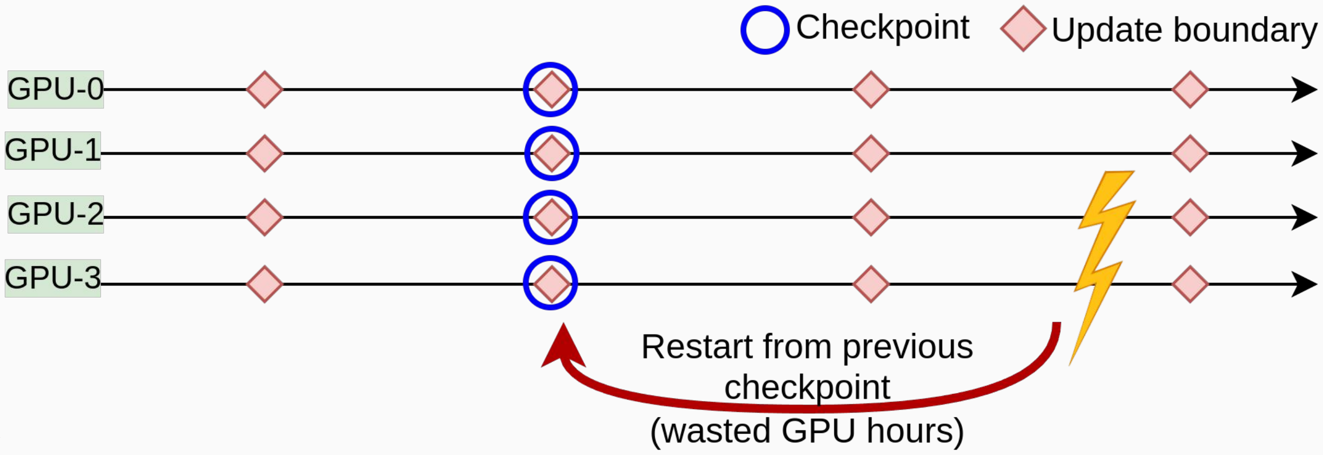


Fig: Failures on Shanghai AI Laboratory's LLM Clusters: Seren and Kalos, housing a total of 4704 A100 GPUs in total^

*Unicron: Economizing Self-Healing LLM Training at Scale; He, Tao, et. al. 2023, https://arxiv.org/pdf/2401.00134
^Characterization of Large Language Model Development in the Datacenter, Hu, Qinghao, et. al., 2024, https://arxiv.org/pdf/2403.07648v2

Checkpointing as a Fundamental Primitive for LLMs



Scenarios:

Failures

- NCCL timeout
- NVLink error
- Invalid DMA mapping
- Task hung up
- Link flapping

Impacts one or more processes

Undesirable training trajectories

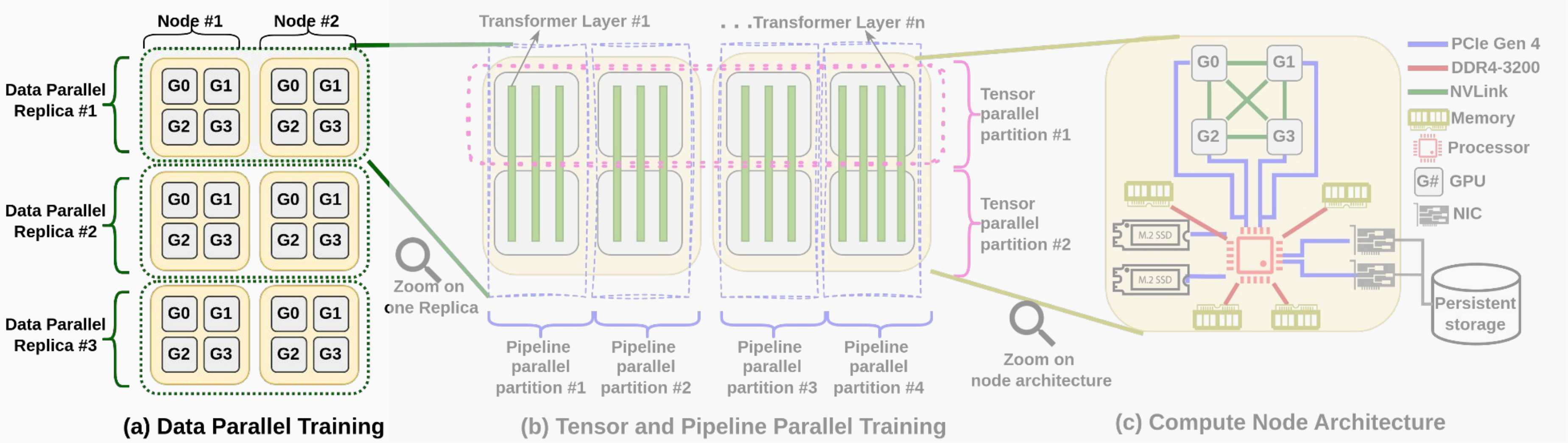
- Google PaLM reported model spikes at arbitrary training points
- Restart from checkpoints taken 100s of timesteps ago
- Costly fine-grained checkpointing due to lack of efficient checkpoint engine

Productive and Administrative

- Understanding Model Evolution
- Forensics, Biases & Ethics: periodic evaluation in the background
- Suspend-resume (e.g. every 6 hours)
- Elastic training: Vary number of GPUs

3D Parallelism: How to Scale LLM Pre-Training (1)

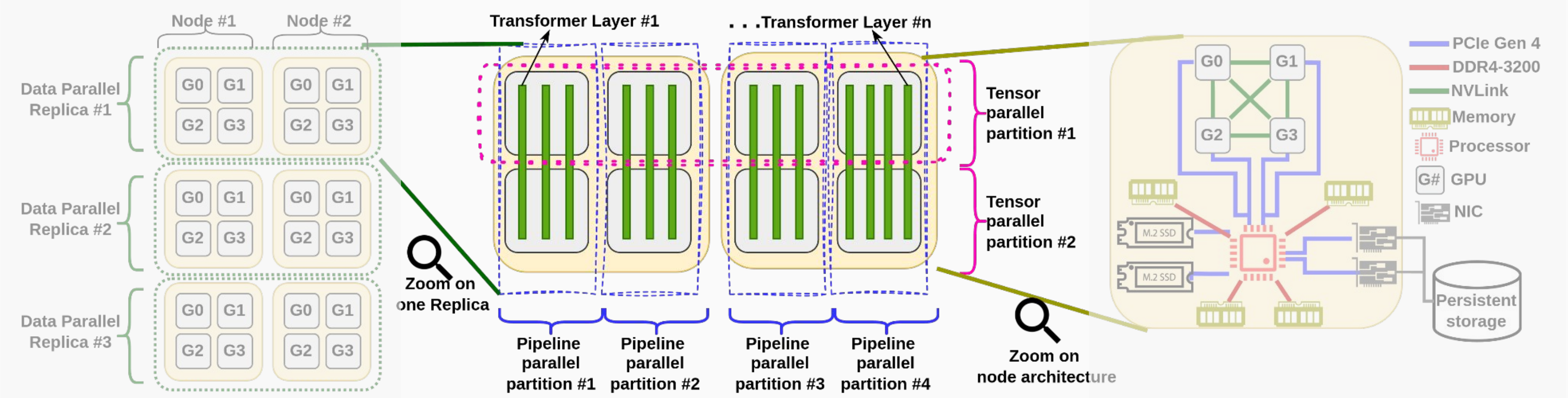
HPDC'24



- Input data is split across data-parallel instances to improve training throughput
- Gradients are averaged using all-reduce to keep the replicas in sync and learn the same pattern

3D Parallelism: How to Scale LLM Pre-Training (2)

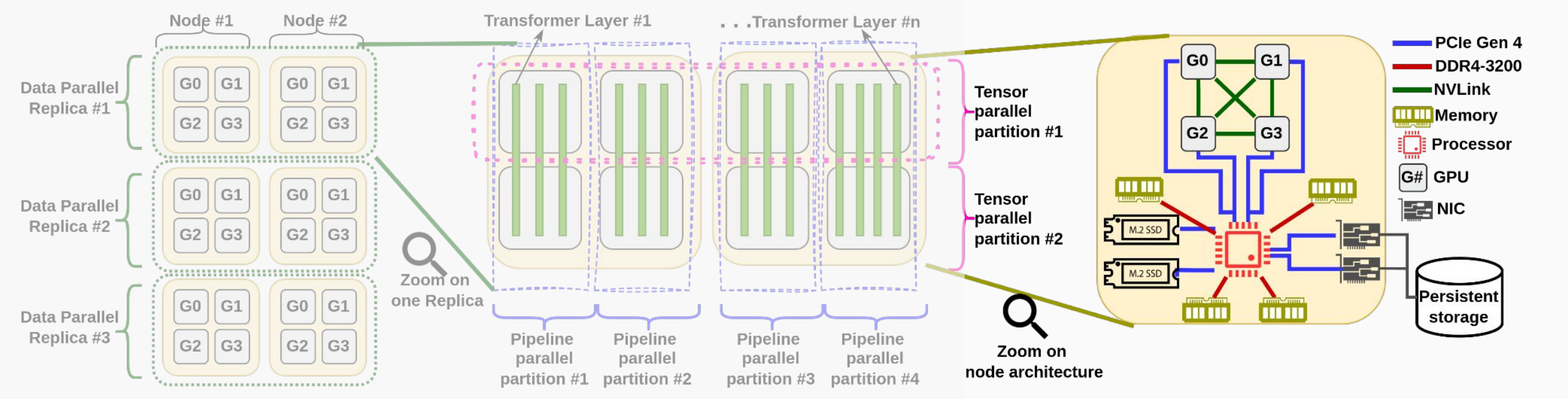
HPDC'24



(a) Data Parallel Training (b) Tensor and Pipeline Parallel Training (c) Compute Node Architecture

- Tensor parallelism splits individual layers horizontally across multiple GPUs
- Pipeline parallelism groups multiple layer together into successive stages

Checkpointing under 3D Parallelism: Use Heterogeneous Storage



(a) Data Parallel Training

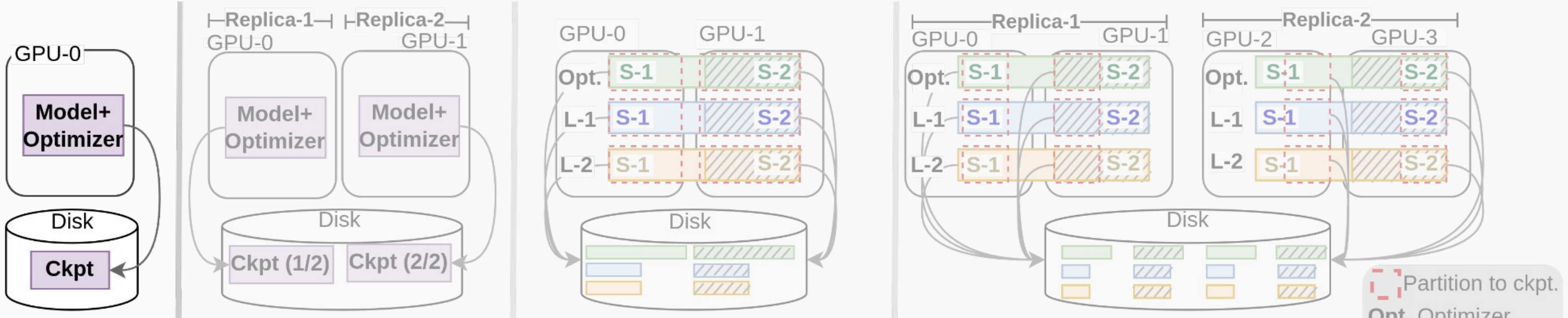
(b) Tensor and Pipeline Parallel Training

(c) Compute Node Architecture

- PCIe Interconnects (25GB/s+) are used to capture checkpoints to host memory
- From there, multi-level storage hierarchy: node-local NVMe, remote storage (PFS)

What Do We Need to Checkpoint?

HPDC'24



(a) Conventional DNN (PP=1, TP=1, DP=1)

(b) Conventional DNN (PP=1, TP=1, DP=2)

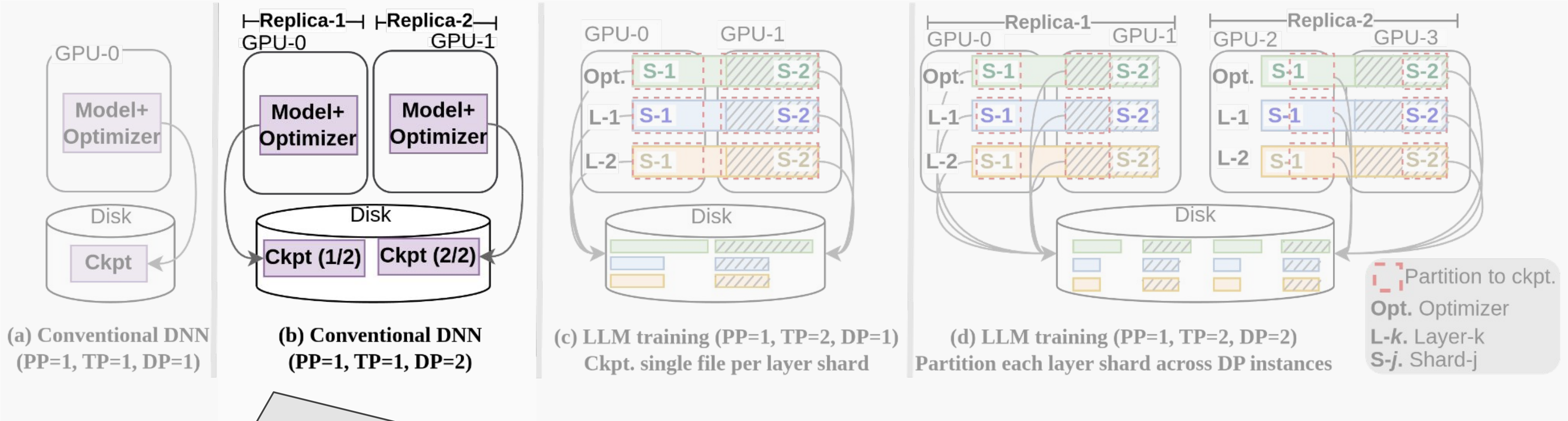
(c) LLM training (PP=1, TP=2, DP=1) Ckpt. single file per layer shard

(d) LLM training (PP=1, TP=2, DP=2) Partition each layer shard across DP instances

- Produces a single checkpoint file
- What do we need to checkpoint: Metadata (e.g. PRNG state), model parameters, optimizer state

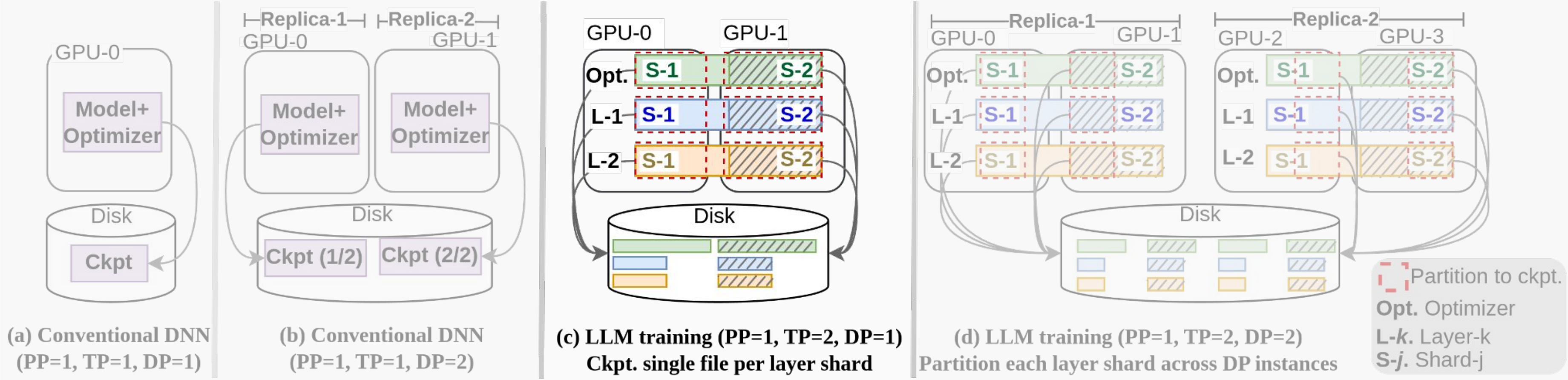
Data Parallelism: Parallel Checkpointing

HPDC'24



- Each data-parallel replica owns a complete copy of the model
- Checkpointing in parallel exploits the I/O bandwidth of all GPUs/nodes
- Examples: DeepFreeze, TorchSnapshot, etc.

Model and Optimizer State Fine-Grain Sharding (1)



(a) Conventional DNN (PP=1, TP=1, DP=1)

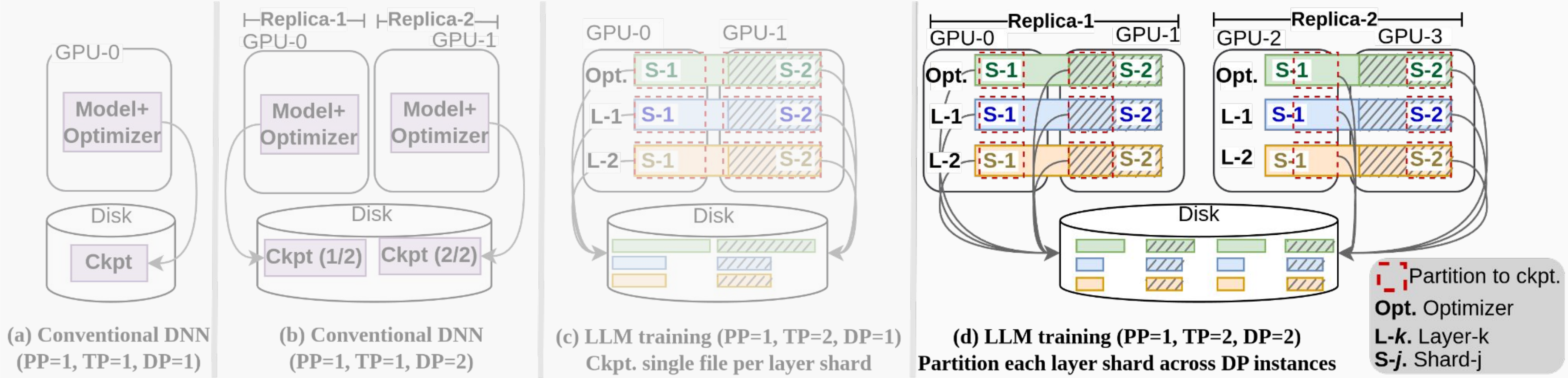
(b) Conventional DNN (PP=1, TP=1, DP=2)

(c) LLM training (PP=1, TP=2, DP=1) Ckpt. single file per layer shard

(d) LLM training (PP=1, TP=2, DP=2) Partition each layer shard across DP instances

- Each model layer and optimizer shard produces a different checkpoint file for each GPU (e.g. DeepSpeed)
- Helpful for elastic/universal checkpoint-restart (use different data, tensor, pipeline-parallelism on restart)
- All shards need to be consistently captured for a successful checkpoint

Model and Optimizer State Fine-Grain Sharding (2)



(a) Conventional DNN (PP=1, TP=1, DP=1)

(b) Conventional DNN (PP=1, TP=1, DP=2)

(c) LLM training (PP=1, TP=2, DP=1) Ckpt. single file per layer shard

(d) LLM training (PP=1, TP=2, DP=2) Partition each layer shard across DP instances

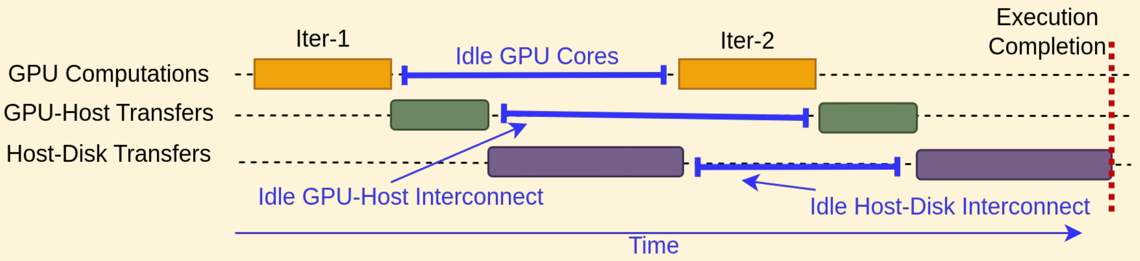
- Independent files per shard enables highly parallel I/O, but too many files may introduce I/O bottlenecks on shared storage (PFS)

Goal: High-Performance, Scalable Checkpointing that Masks I/O Overheads

Synchronous and Asynchronous Data Movement Techniques

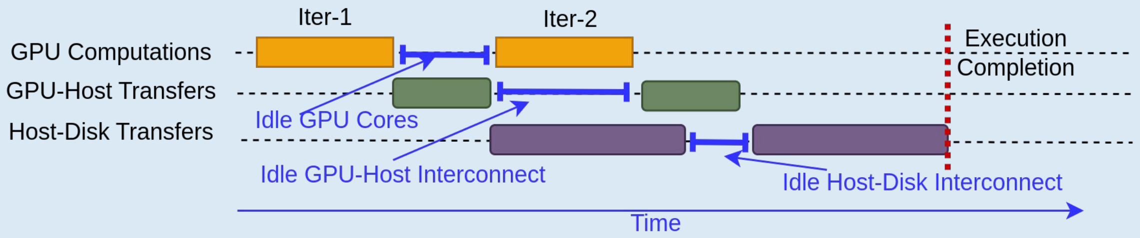
Synchronous Data Movement

- ✔ Easy to design and debug
- ✔ Avoids CPU oversubscription
- ✖ Underutilized computational resources, memory tiers and interconnects

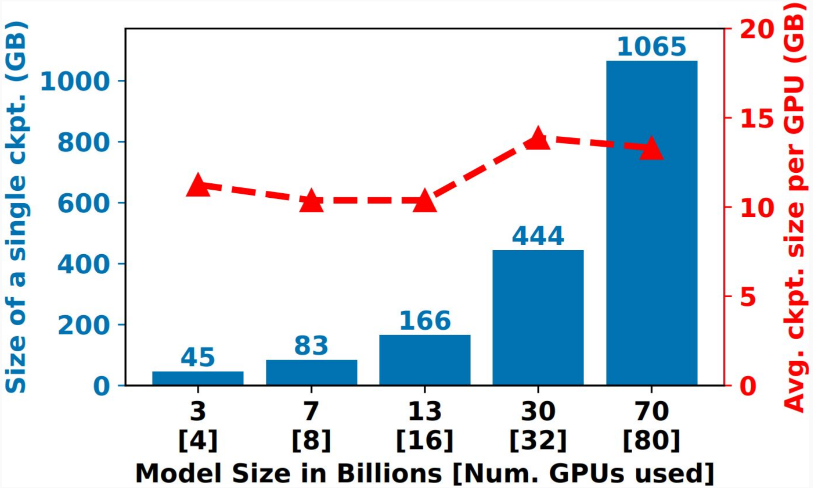


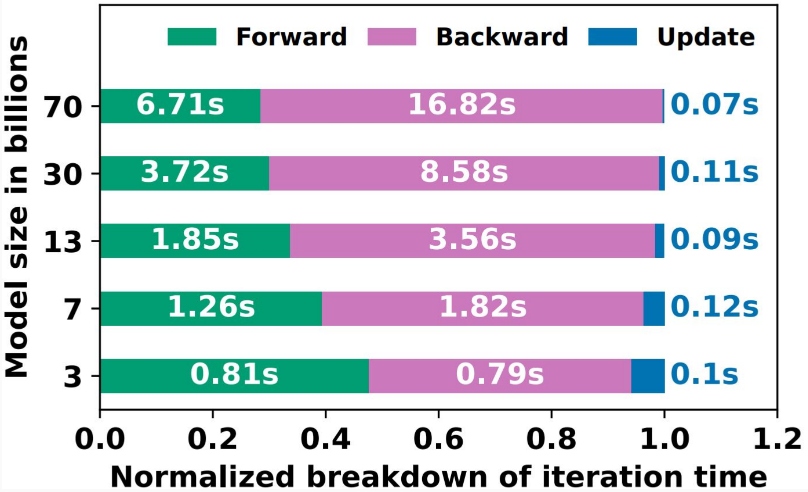
Partially Asynchronous Data Movement

- ✔ Easy extension of existing engines
- ✔ Mitigates slow I/O bottlenecks beyond the host tier
- ✖ Underutilization of spare GPU memory and GPU-host interconnect



Observations Driving Our Design





- Checkpoint size increases linearly with the number of model parameters
- Checkpoint **shard per GPU is load balanced** and remains the similar for different model sizes
- Forward and backward pass consume majority of the iteration duration (>95%) during training
- **Model and optimizer states are immutable** during forward and backward passes

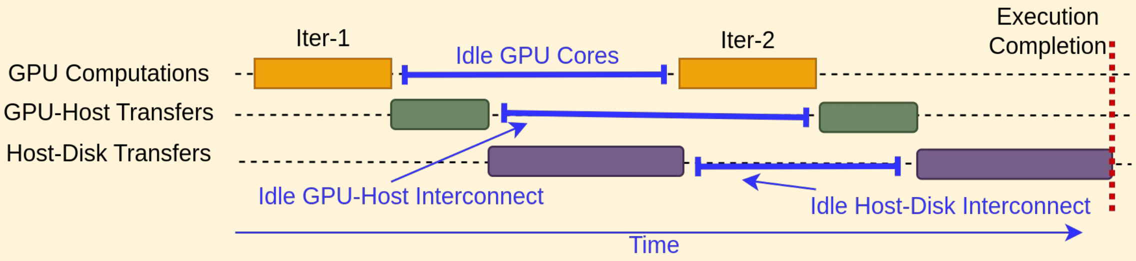
DataStates-LLM: Key Ideas and Design Principles

- Leverage Immutability: Lazy Non-Blocking Copies Overlapping with Forward and Backward Pass
  - Model and optimizer states do not change during forward and backward passes
  - Keep copying until the start of update phase; block updates if previous copies are pending
- Coalescing of GPU Model/Optimizer Shards to Host Memory
  - Prepare the host memory for efficient GPU-host data transfers (pre-pinning)
  - Optimize host memory layout for bulk transfer of shards from multiple GPUs
- Streamlined Multi-level Flushing to Persistent Storage
  - Start streaming to disk as soon as partial checkpointing data is copied from GPU to host memory
  - Parallel use of two physical links: GPU-to-host and host-to-disk
- Asynchronous Distributed Consolidation of Model and Optimizer Shards
  - Asynchronous multi-level flushing necessitate consensus to commit a valid and consistent checkpoint version

Synchronous and Asynchronous Data Movement Techniques

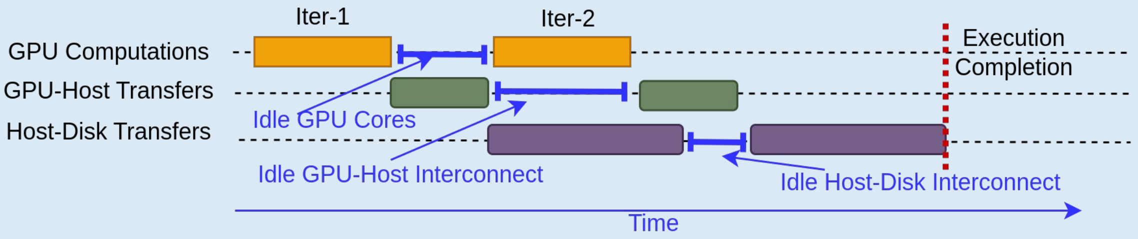
Synchronous Data Movement

- ✔ Easy to design and debug
- ✔ Avoids CPU oversubscription
- ✖ Underutilized computational resources, memory tiers and interconnects



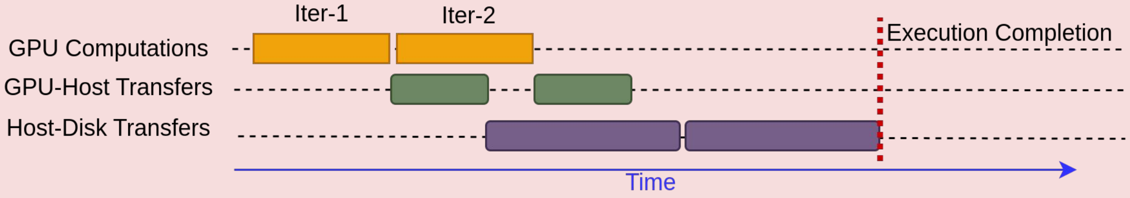
Partially Asynchronous Data Movement

- ✔ Easy extension of existing engines
- ✔ Mitigates slow I/O bottlenecks beyond the host tier
- ✖ Underutilization of spare GPU memory and GPU-host interconnect

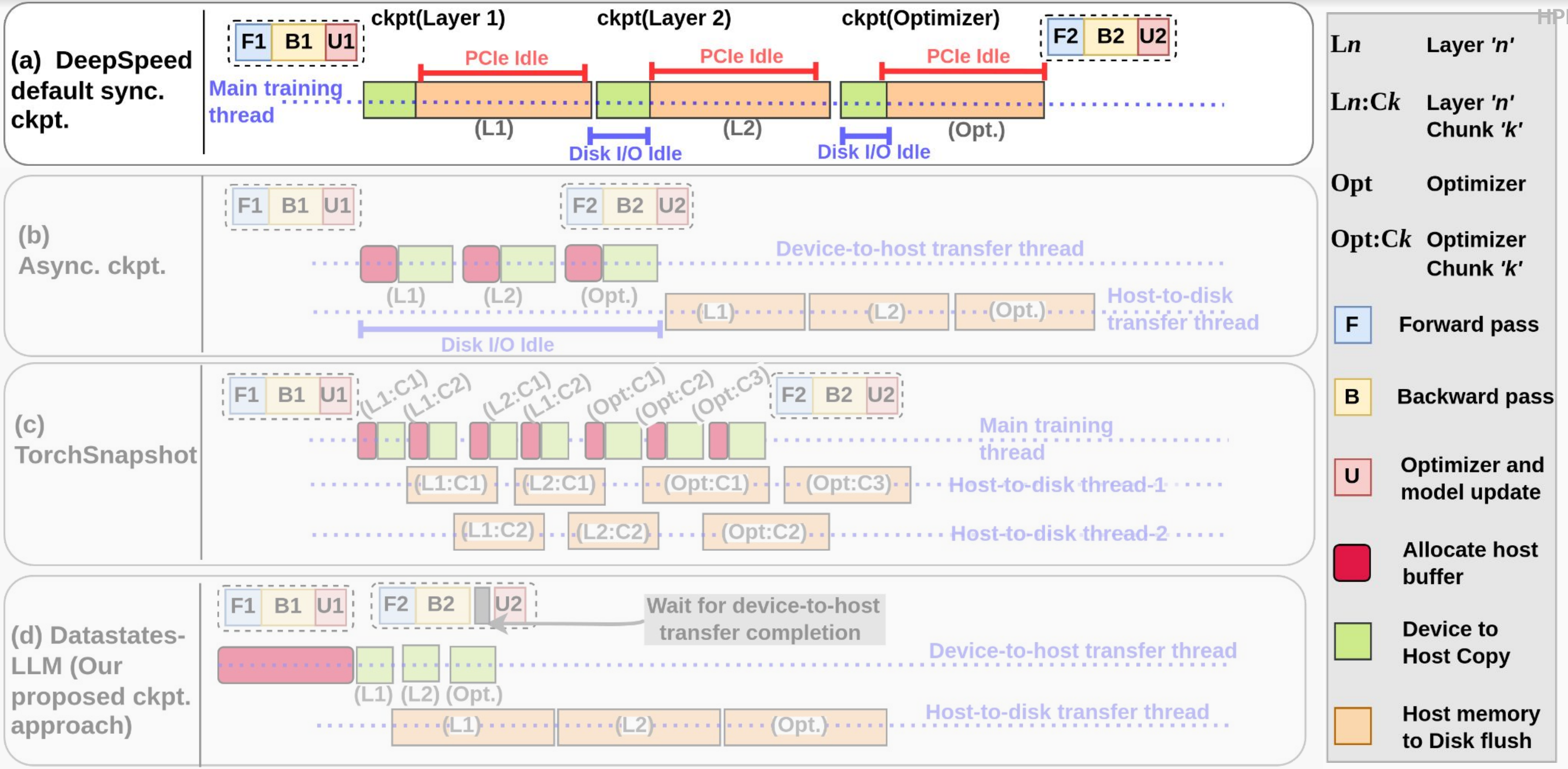


Asynchronous Data Movement

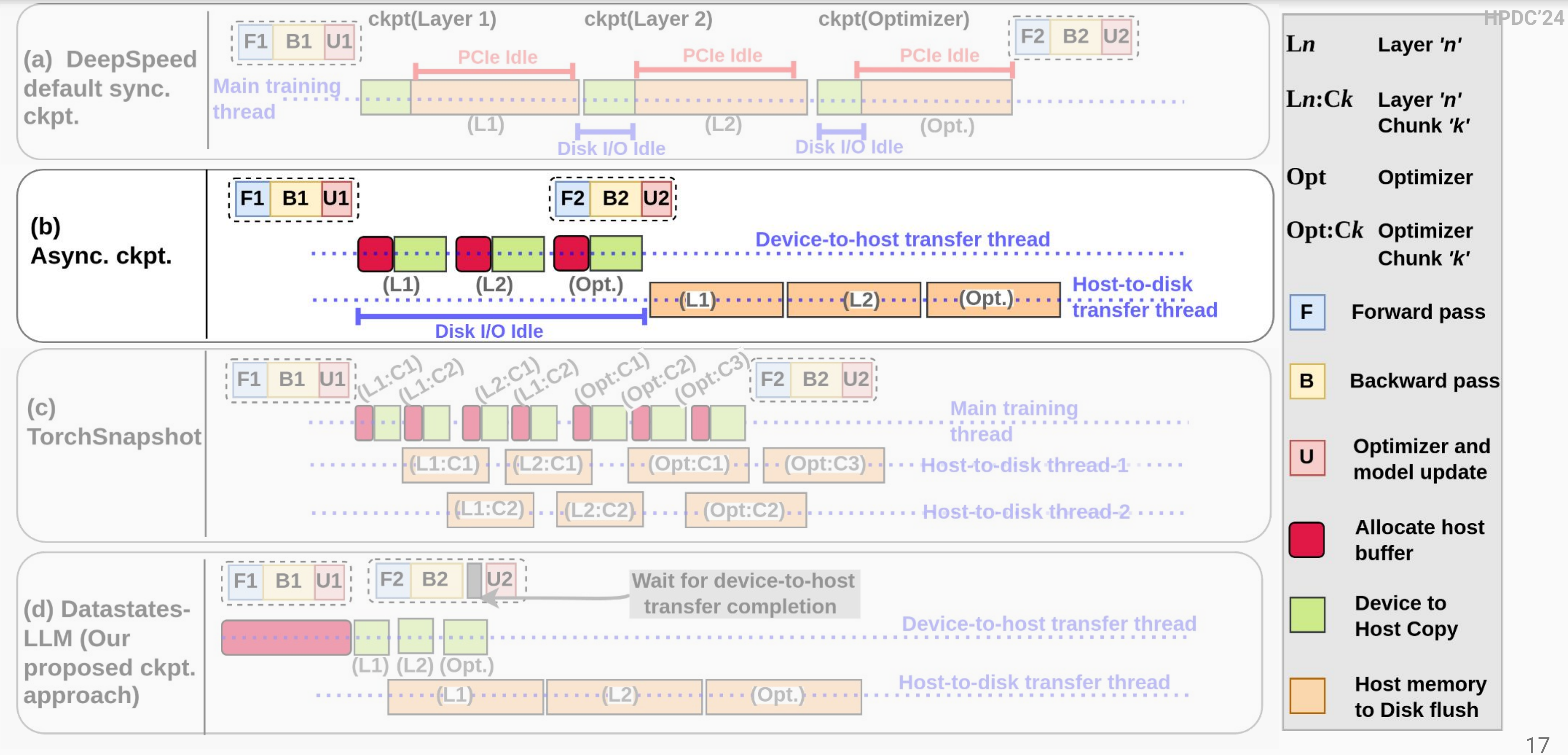
- ✔ Mitigates slow I/O bottlenecks and memory utilization for all tiers
- ✖ Complex overlap centric design makes it challenging to design & debug



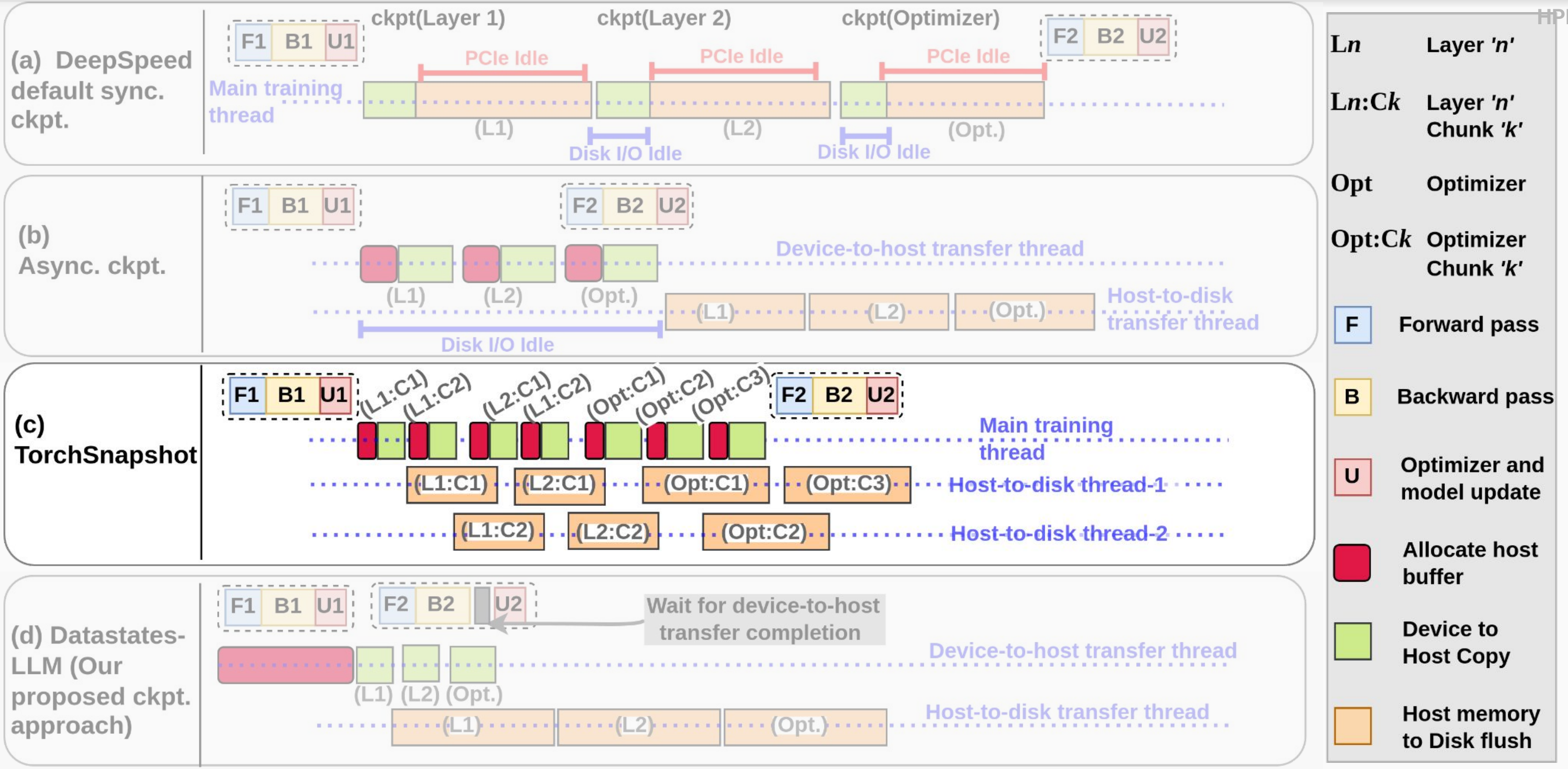
Comparison with State of Art Checkpointing Approaches



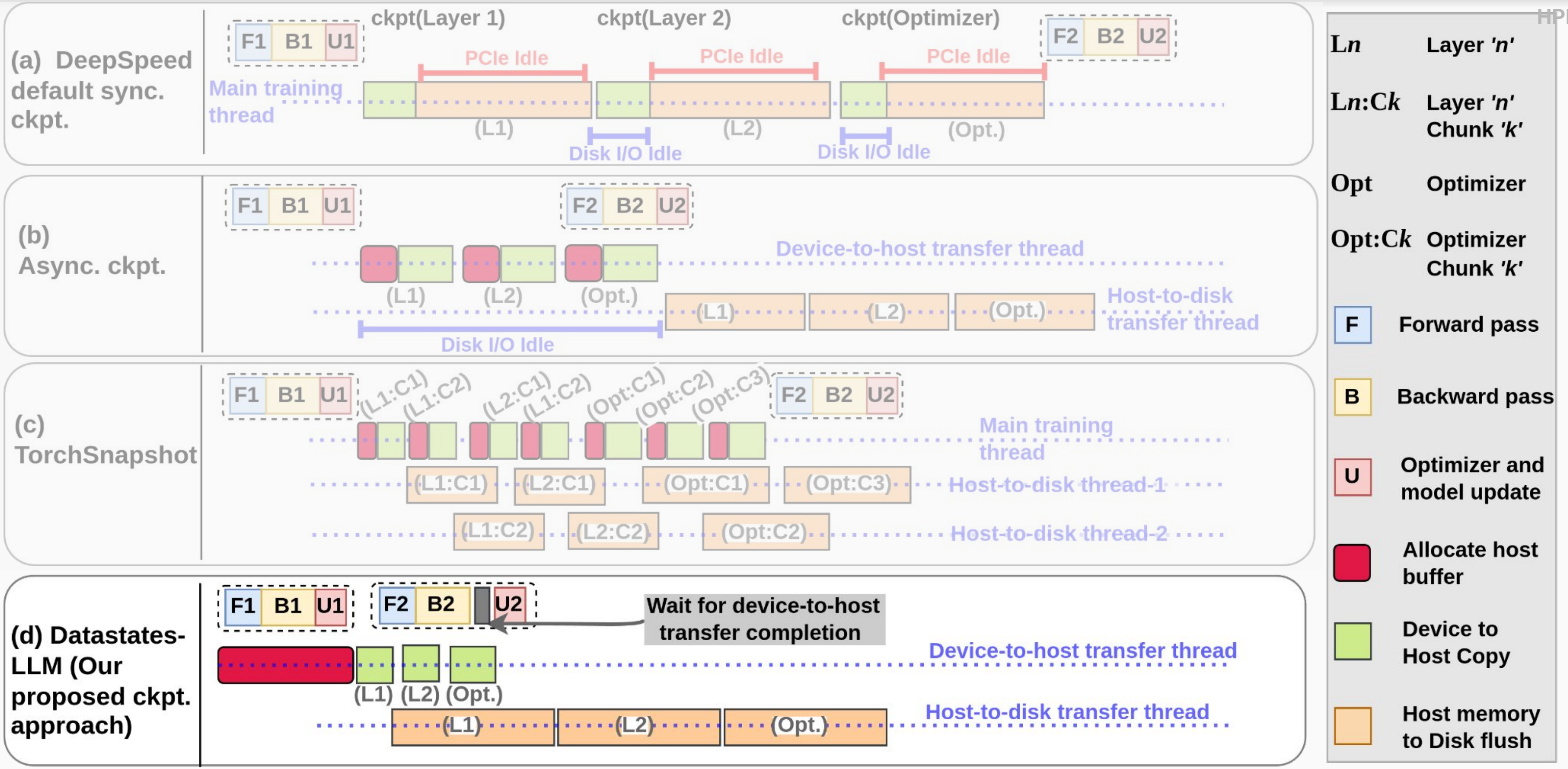
Comparison with State of Art Checkpointing Approaches



Comparison with State of Art Checkpointing Approaches



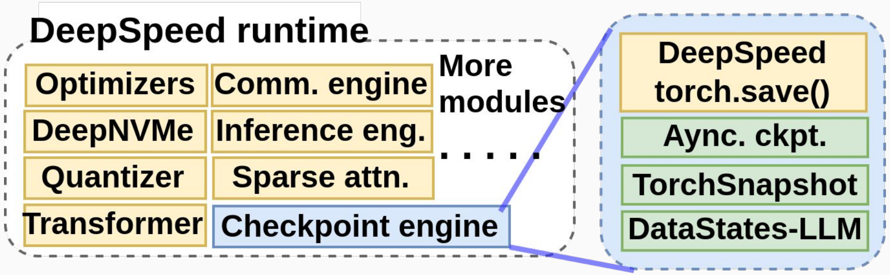
Comparison with State of Art Checkpointing Approaches



Implementation and Integration with DeepSpeed

- Module extension to DeepSpeed, state-of-art LLM training runtime
- Written in C++/CUDA and exposed through Python and C++ APIs
  - Eliminates inefficiencies arising from Python Global Interpreter Lock (GIL)
  - Uses dedicated CUDA-streams overlapping D2H and H2D transfers using hardware copy engines
  - Leverages PyBind11
- Openly available and extensible to other accelerators and runtimes (e.g., Pytorch Lightning)

DataStates-LLM

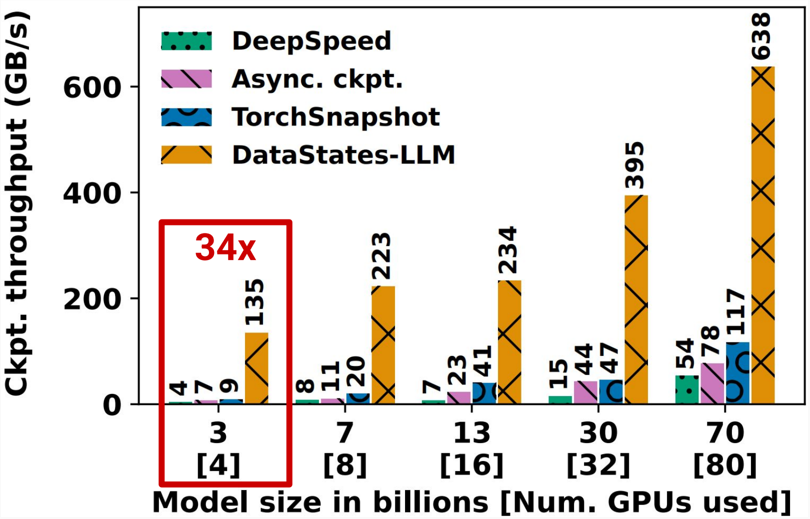


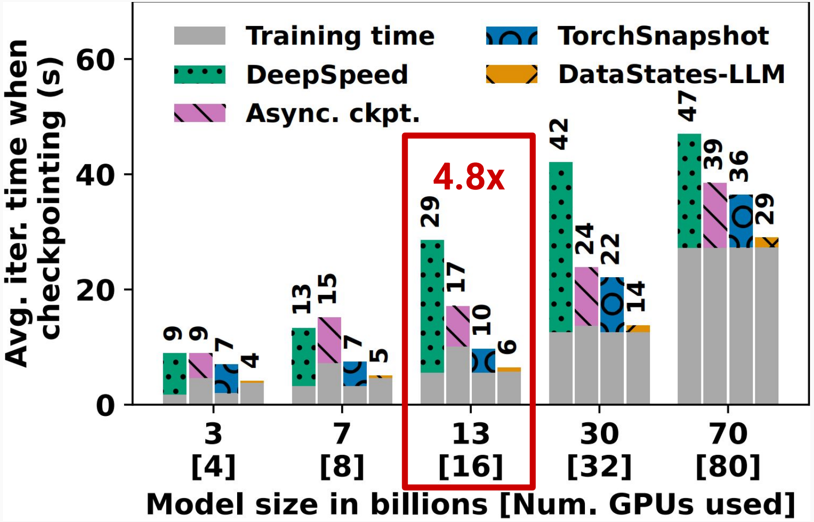
Experimental Evaluation

- Experimental Setup: ALCF Polaris testbed
  - Every node: 4xA100 40GB GPUs and 512 GB host memory
  - We use up to 512 GPUs
  - Each GPU mapped to a different NUMA domain with PCIe Gen 4 device-host throughput: 25 GB/s
  - Luster file system for persistence with 160 OST and 40 metadata servers with aggregated bandwidth of 650 GB/s

- Model and runtime configuration
  - 5 models from real-world setups: 3B, 7B, 13B, 30B, 70B
  - Tensor-parallelism: 4 (max GPUs per node), pipeline parallelism: number of nodes, ZeRO stage-1
- Compared approaches
  - DeepSpeed, Asynchronous checkpointing, TorchSnapshot, DataStates-LLM (Ours)
- Performance metrics
  - Checkpointing throughput — Measures checkpoint_size/total_blocking_time
  - Iteration slowdown — Measures impact of I/O overheads incurred by checkpointing
  - End-to-end training time — Measures impact of slow asynchronous flushes to disk

Checkpointing Performance for Different Model Sizes

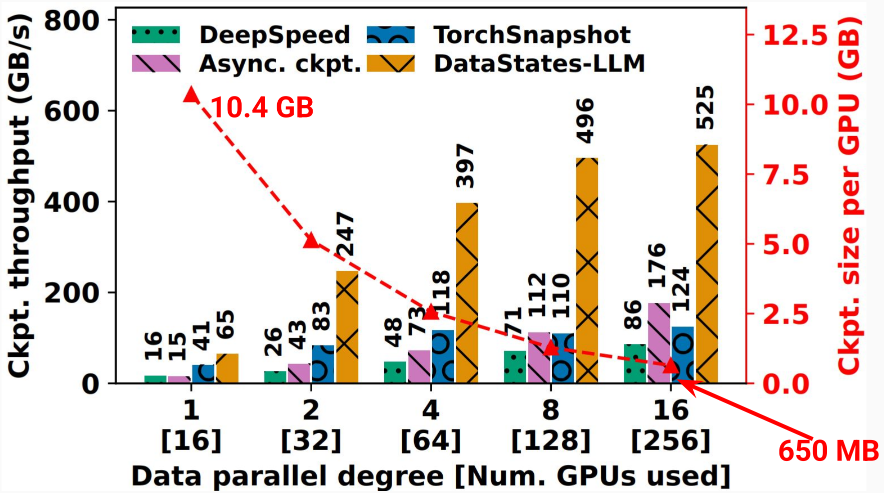




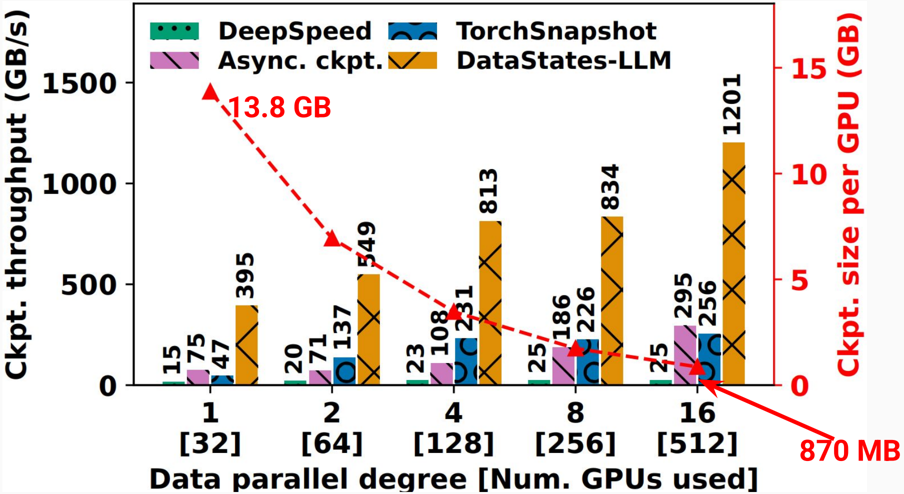
- DataStates-LLM achieves **4x – 34x faster checkpointing** throughput compared to state-of-art
- Checkpointing throughput increases for larger models because all GPUs flush in parallel

- DataStates-LLM achieves **1.3x – 4.8x faster iterations** compared to state-of-art
- DataStates-LLM shows negligible overheads on the training iteration when checkpointing

Increasing Data-Parallelism: Strong Scalability of Checkpoint Performance



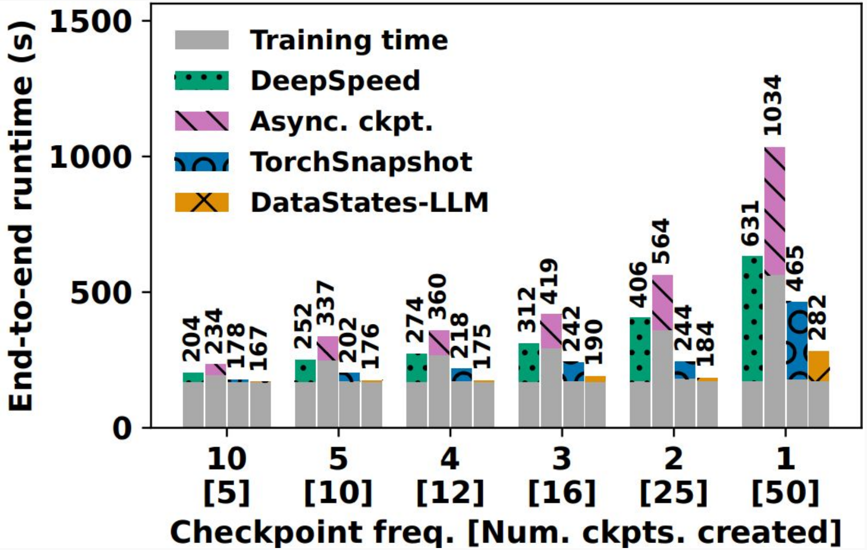
13B model



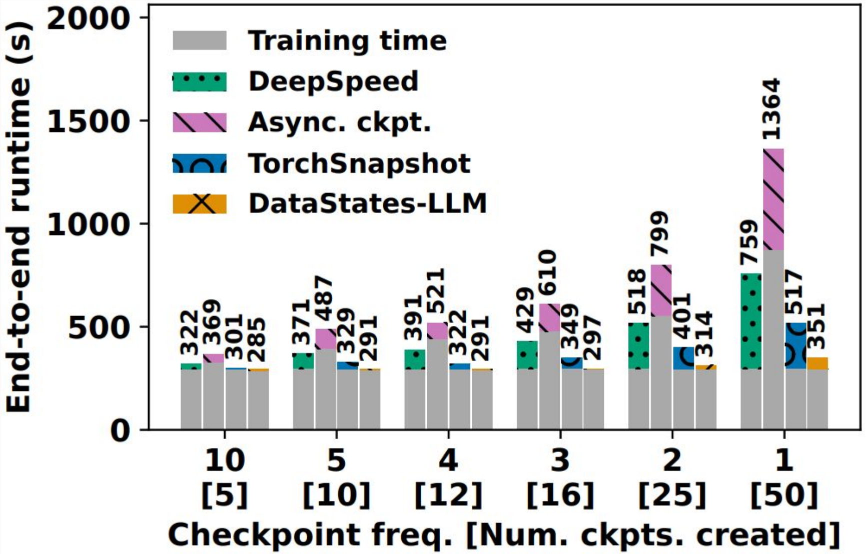
30B model

- All approaches scale well to increasing data parallel replicas due to more parallel channels for flushes
- Our approach achieves **1.75x – 48x faster checkpointing** compared to state-of-art

End-to-end Runtime for Increasing Checkpointing Frequency



7B model



13B model

Our overlap centric design achieves **3x – 4.2x faster** end-to-end training compared to state-of-art checkpointing engines, irrespective of the I/O pressure due to increasing checkpointing frequency

Key takeaways: Asynchronous Checkpointing for LLMs

- Large-scale distributed LLM training running with advanced hybrid parallelism strategies are prone to failures and undesirable trajectories, necessitating checkpointing
- State-of-the-art checkpointing engines are inefficient because
  - They do not exploit immutable training phases to overlap checkpoint I/O
  - They underutilize available interconnect and memory resources
- DataStates-LLM efficiently and transparently captures globally consistent checkpoints
  - Uses preallocated pinned buffers for fast DMA
  - Coalescing of model/optimizer shards
  - Lazy non-blocking checkpoint snapshotting overlapping with immutable phases
  - Streaming multi-level flushing to persistent storage
  - Asynchronous distributed consensus of checkpoint
- DataStates-LLM achieves **4x – 48x faster checkpointing** and **1.3x – 4.8x faster iterations** compared to state-of-the-art approaches

DataStates
DataStates EvoStore
DataStates LLM
DataStates AI

DataStates LLM

Our Current and Future Research Directions

Pre-training

- Optimized hybrid GPU-CPU computations with optimizer offloading (*Middleware'24*)
  - Accelerate updates by 2.5x using overlapping I/O and combined computations of CPU and GPU
- Accelerated training for memory-constrained scenarios requiring disk-offloaded optimizers
  - Leveraging idle remote storage bandwidth for 3x faster backward and update phases
- Utilizing low rank linear layer representations for accelerated and memory efficient pre-training (ArXiv'25)

Inferencing

- Characterizing KV cache access patterns under concurrency (*IPDPS'25*)
  - Report key findings for optimizing KV cache movement and request scheduling
- Unified, dynamic, asynchronous model and KV cache offloading

More…

- EAIRA: A Methodology for Evaluating AI Models as Scientific Research Assistants (*IJHPC'25*)
- DataStates-LLM Elastic Checkpoint, Evaluations, and Recovery