Charon: A Unified and Fine-Grained Simulator for Large-Scale LLM Training and Inference

Mengtian Yang, Zhekun Zhang, Mingheng Wu, Jianwen Yan, Hanshi Sun, Li-wen Chang

Outline

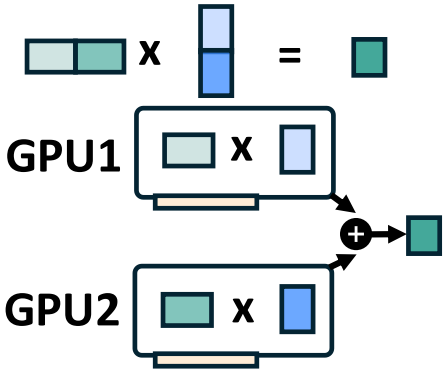
- Background and Design Motivation for LLM Simulator
- Charon Simulator Design
  - System Design Overview
  - Graph-based Frontend
  - Multi-engine Driven Backend
  - Operator Overlap
- Simulation Experiments and Case Studies
- Conclusion

Background

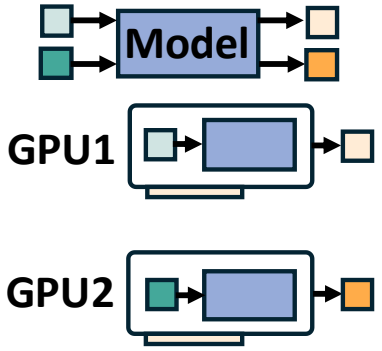
- **Parallelism strategies** are critical in large-scale LLM training and inference
  - TP/DP/EP/PP/SP ...

- Modern LLM training/inference typically **mix several of these parallelism simultaneously**

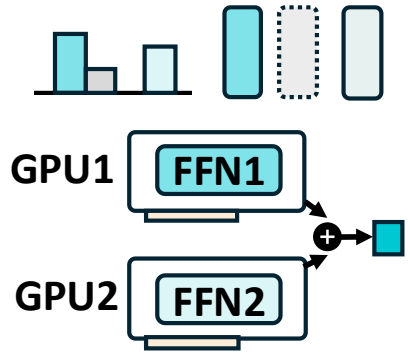
Tensor Parallel (TP)



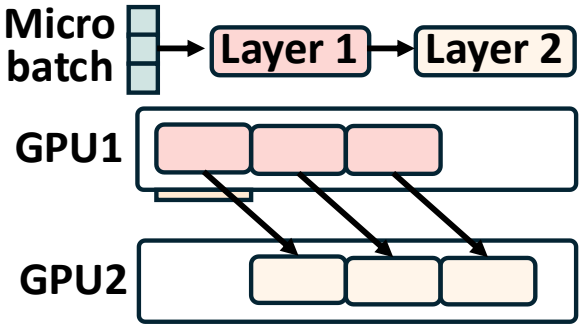
Data Parallel (DP)



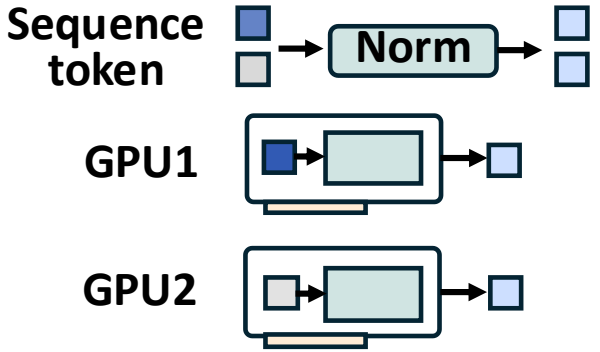
Expert Parallel (EP)



Pipeline Parallel (PP)

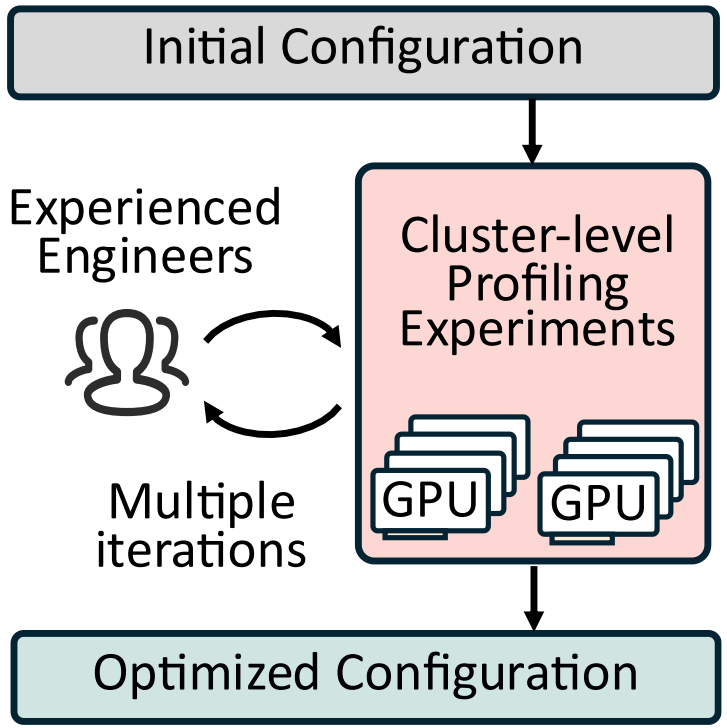


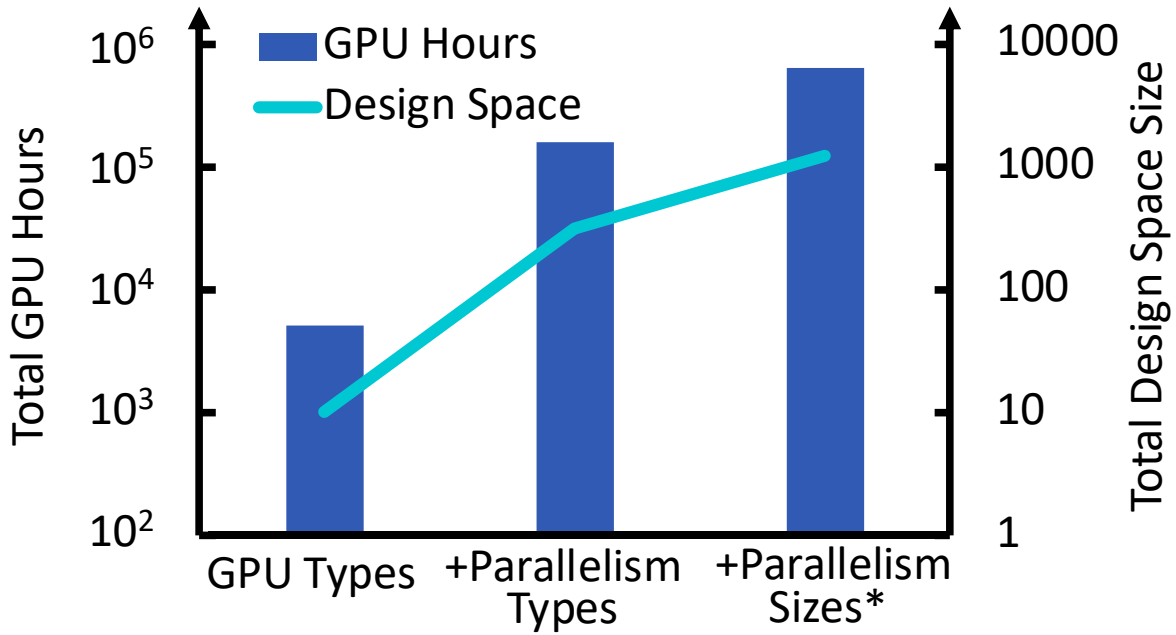
Sequence Parallel (SP)



Design Motivation

- Profiling-based Tuning relies on **excessive cluster-level profiling experiments**
- **Requires up to 10^5 GPU hours** to tune considering the enormous design space



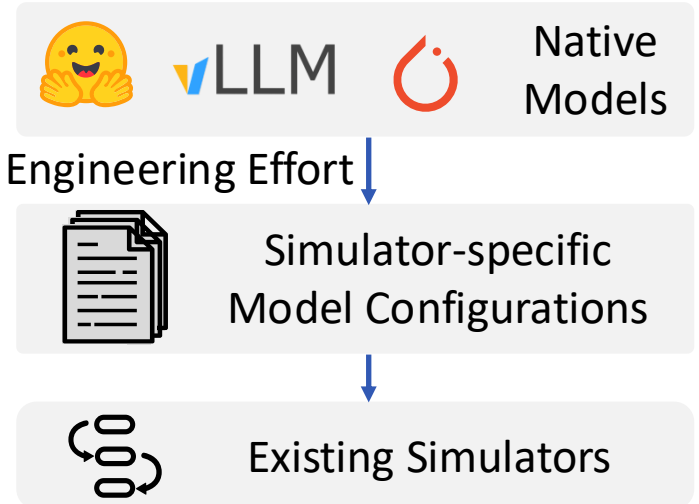


*Suppose for each parallelism type, the parallel size has 4 choices

Design Motivation

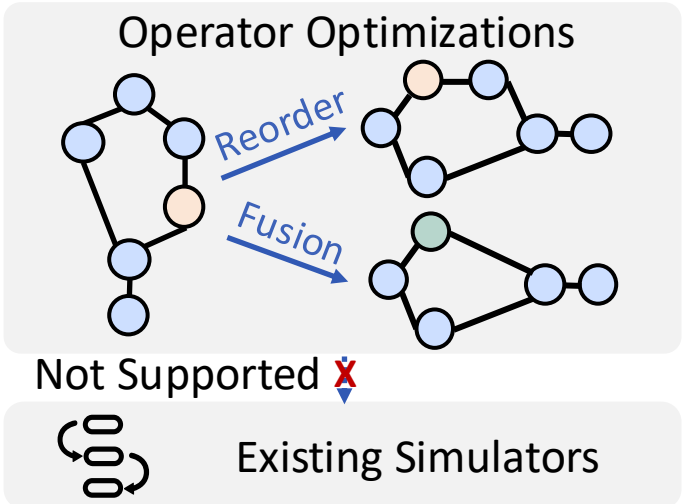
❑ **Challenges and limitations** for existing LLM simulation frameworks

Challenge 1



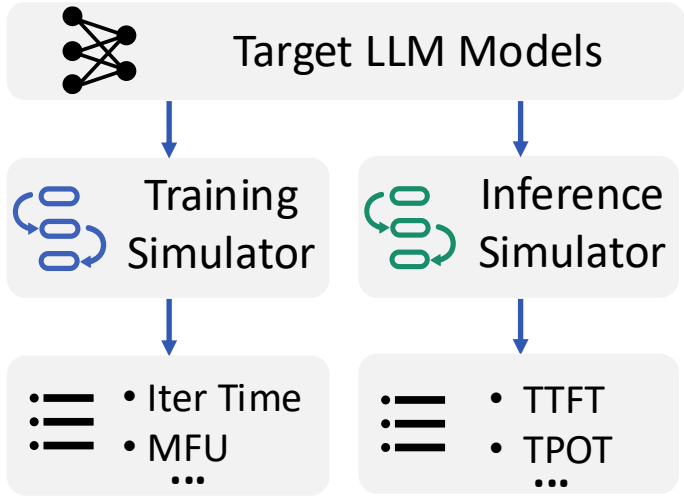
High engineering effort required for model conversion ☹

Challenge 2



Missing native support for operator optimizations ☹

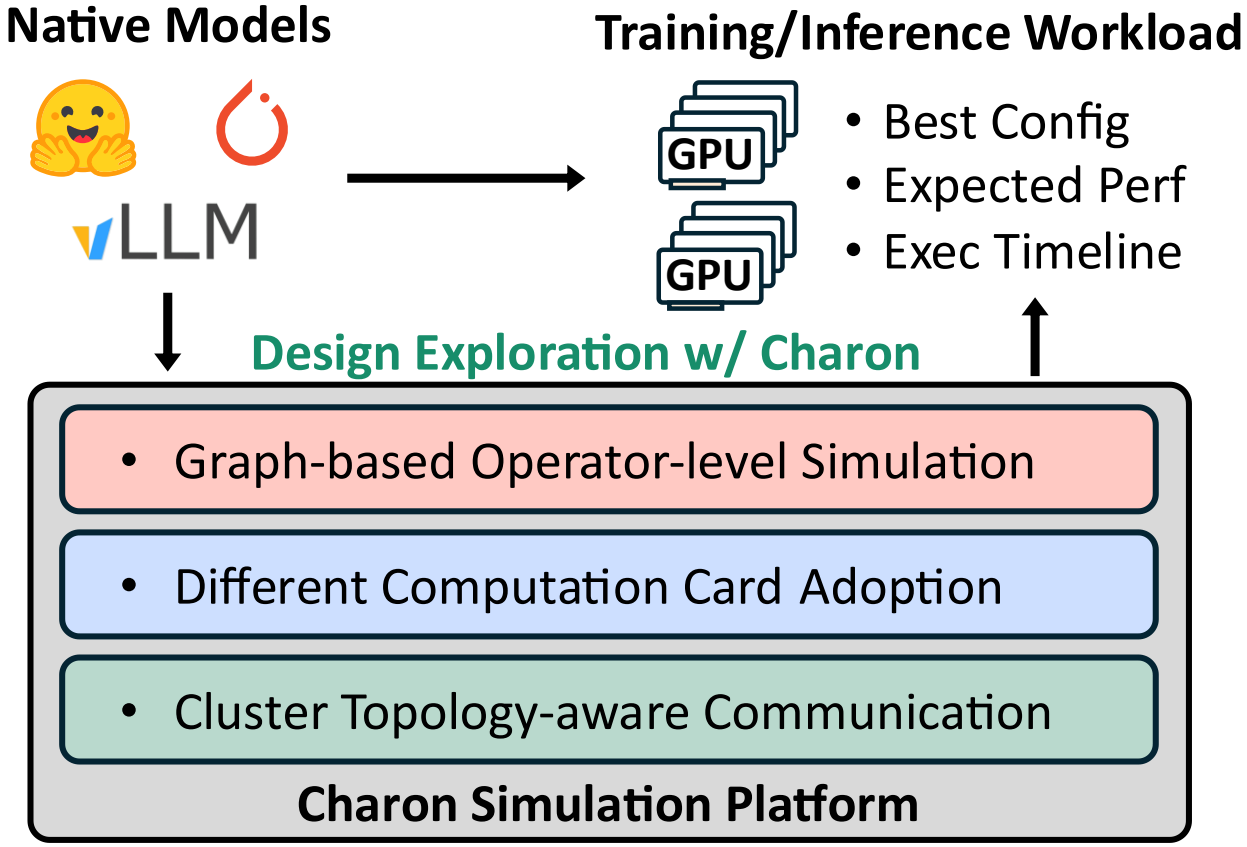
Challenge 3

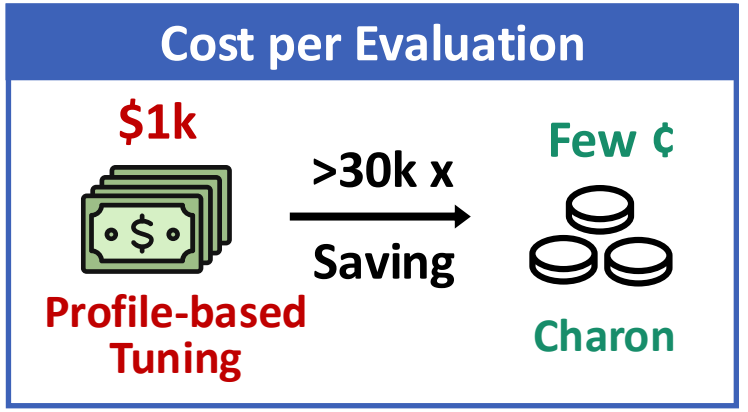


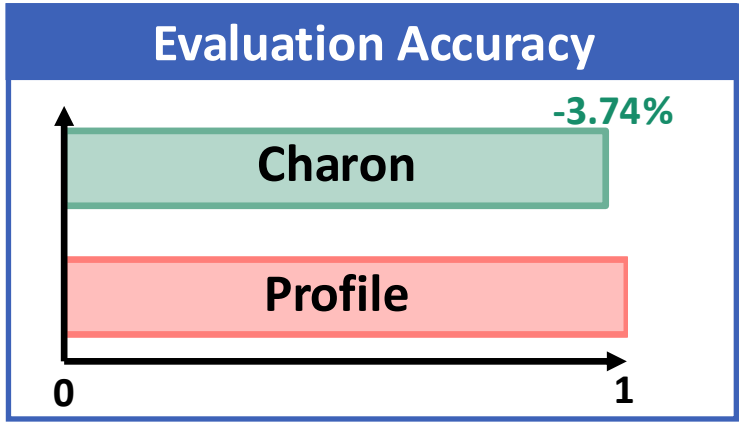
Fragmented simulators yield only coarse end-to-end metrics ☹

Proposed Solution: Charon

- **Unified and Fine-grained simulation** for training and inference from native models
- **Accelerated design exploration** with massive cost reduction



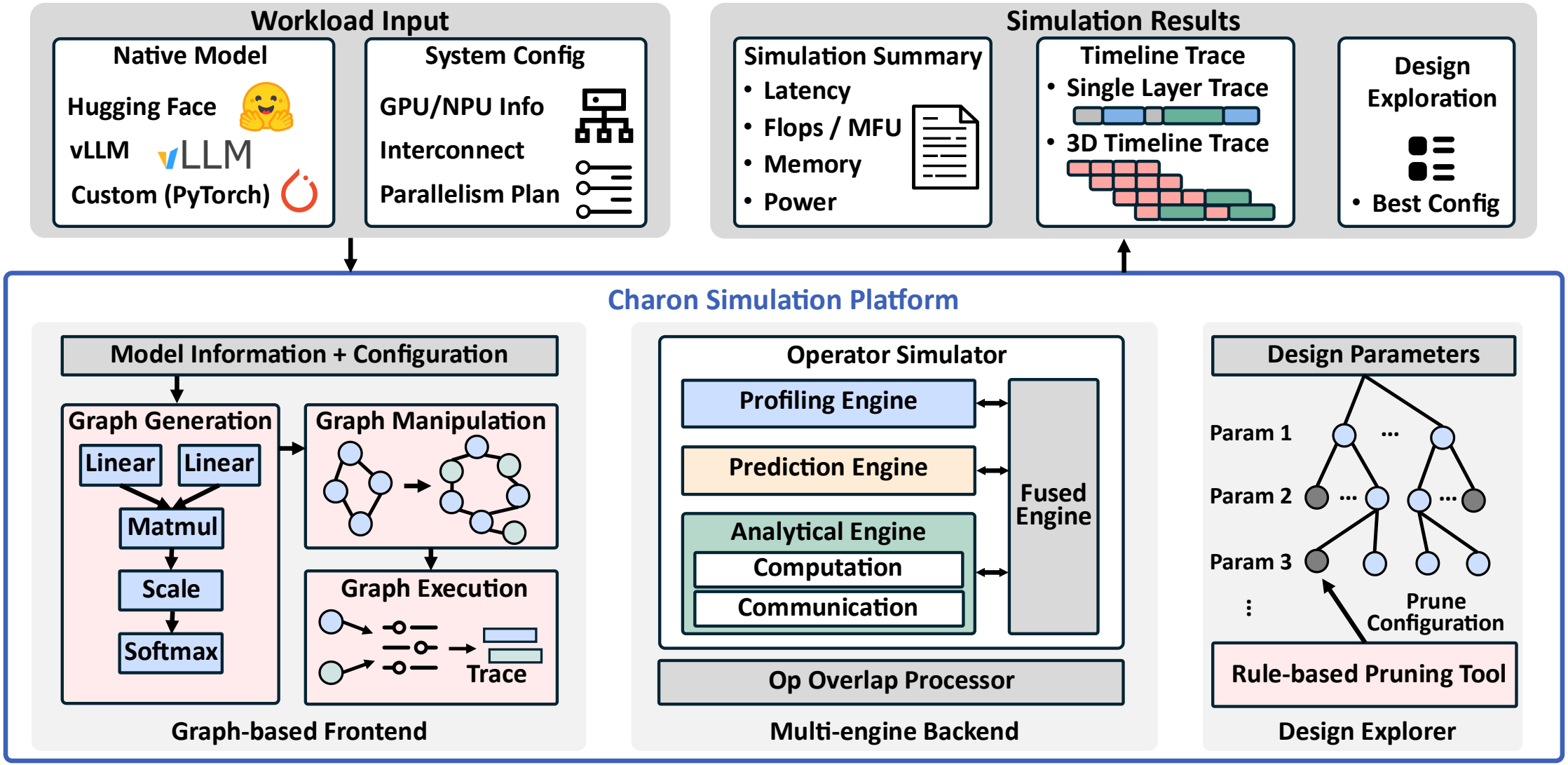




Outline

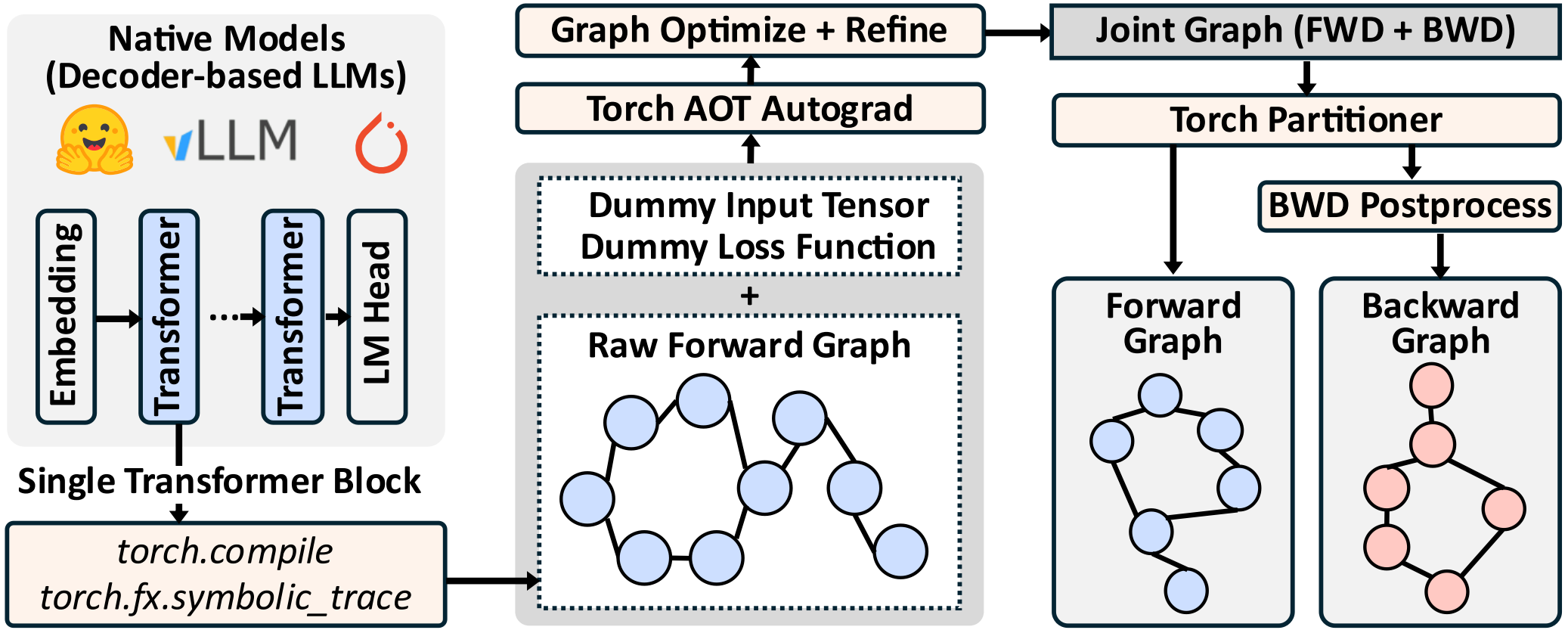
- Background and Design Motivation for LLM Simulator
- **Charon Simulator Design**
  - **System Design Overview**
  - **Graph-based Frontend**
  - **Multi-engine Driven Backend**
  - **Operator Overlap**
- Simulation Experiments and Case Studies
- Conclusion

System Overview



Graph-based Frontend

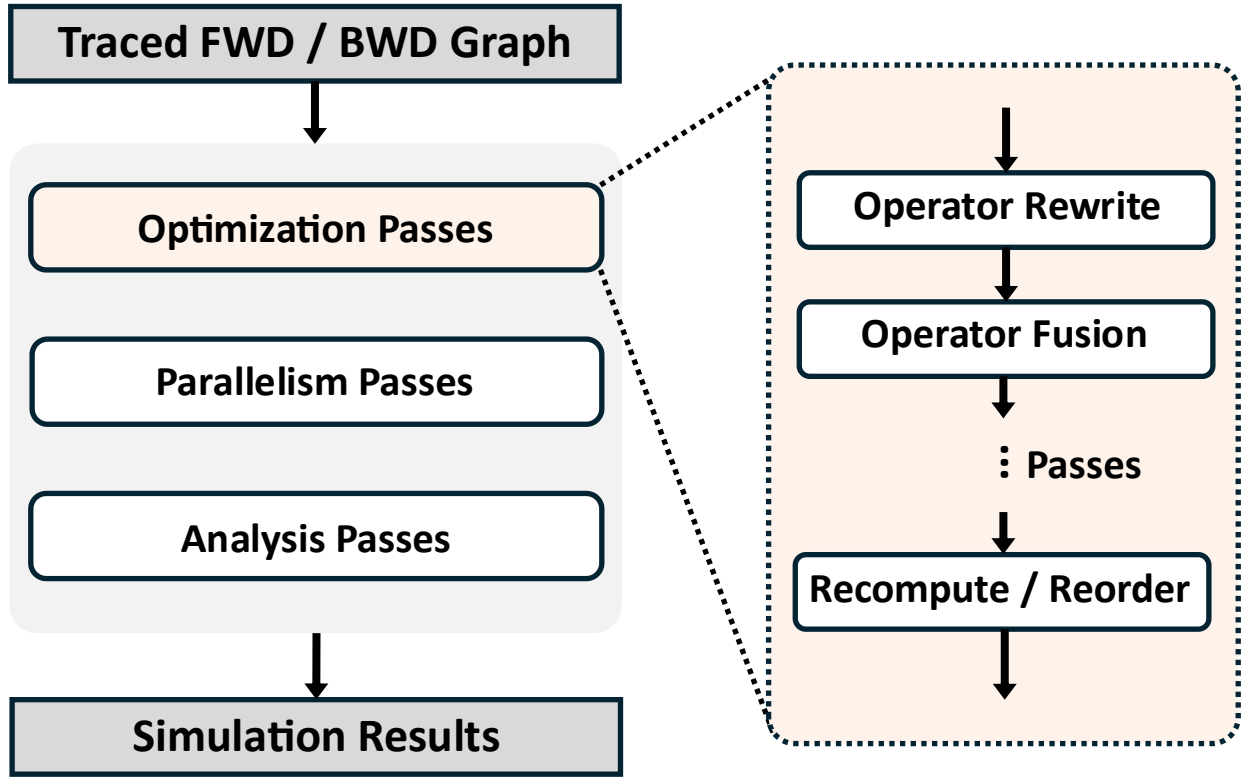
- Graph generation from **native torch-based models** using *symbolic_trace*
- **Automated backward graph generation** using *aot_autograd* with mocked loss function



Graph-based Frontend

❑ **Compiler-style graph manipulation** abstracts optimizations into modular passes

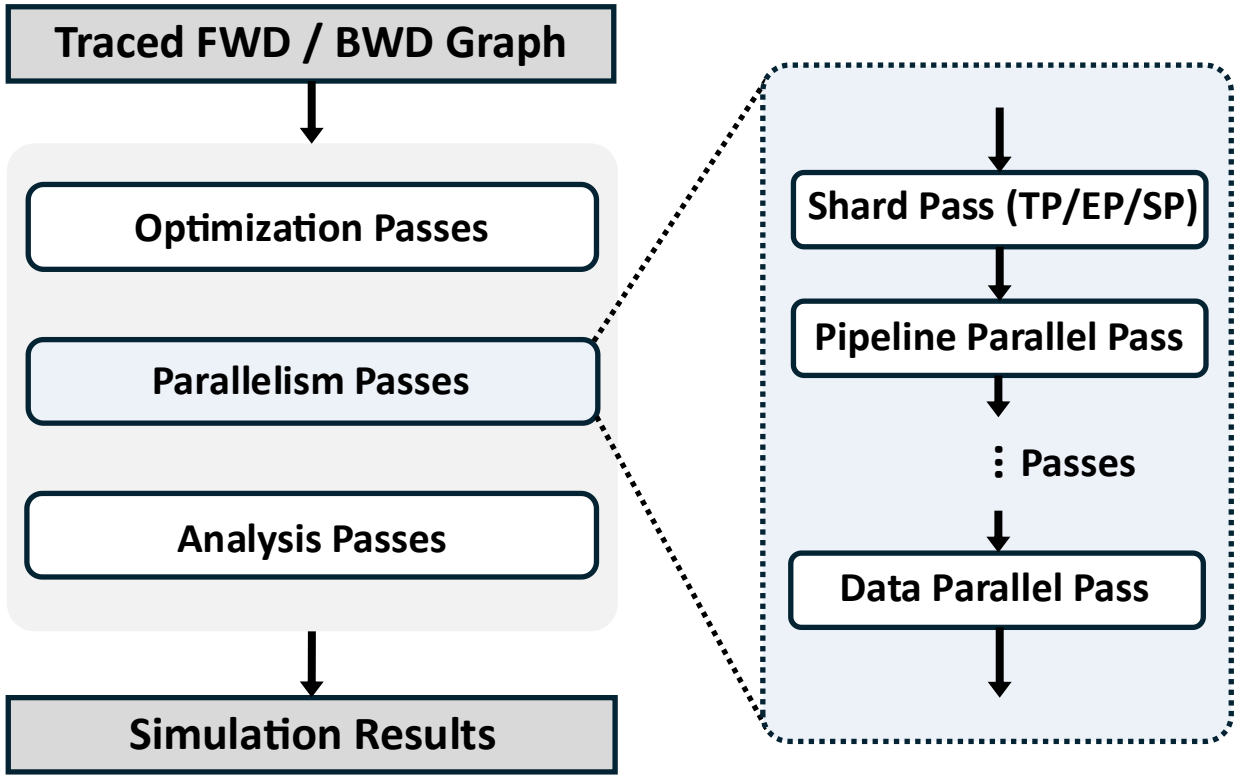
- Traverses the computation graph for specific patterns
- Modifies attributes and connections to enable fusion and reordering
- Evaluates recomputations using interleaved analysis passes



Graph-based Frontend

❑ Modular passes introduce collective communication for **hybrid parallelism strategies**

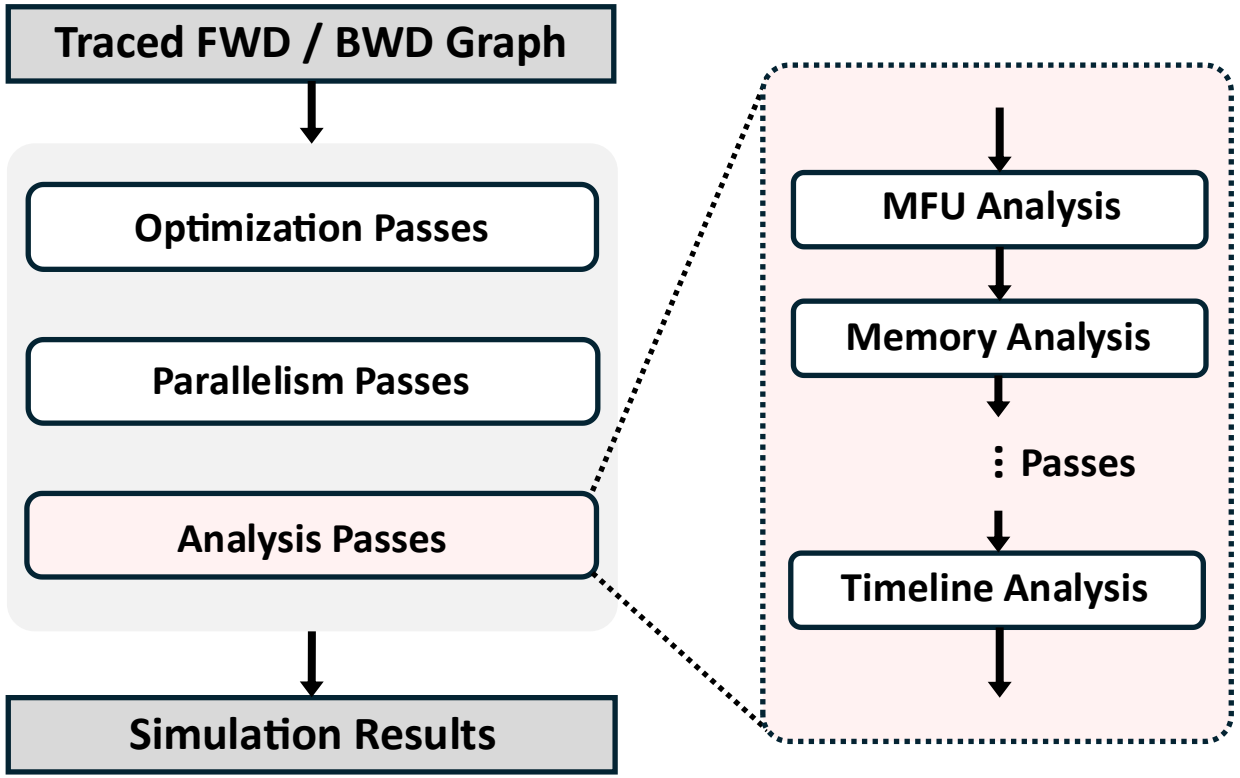
- **Shard Pass**: Adjusts shape and inserts communications for TP/EP/SP
- **PP Pass**: Constructs inter-stage dependencies and inserts send/recv
- **DP Pass**: Handles gradient synchronization and state sharding



Graph-based Frontend

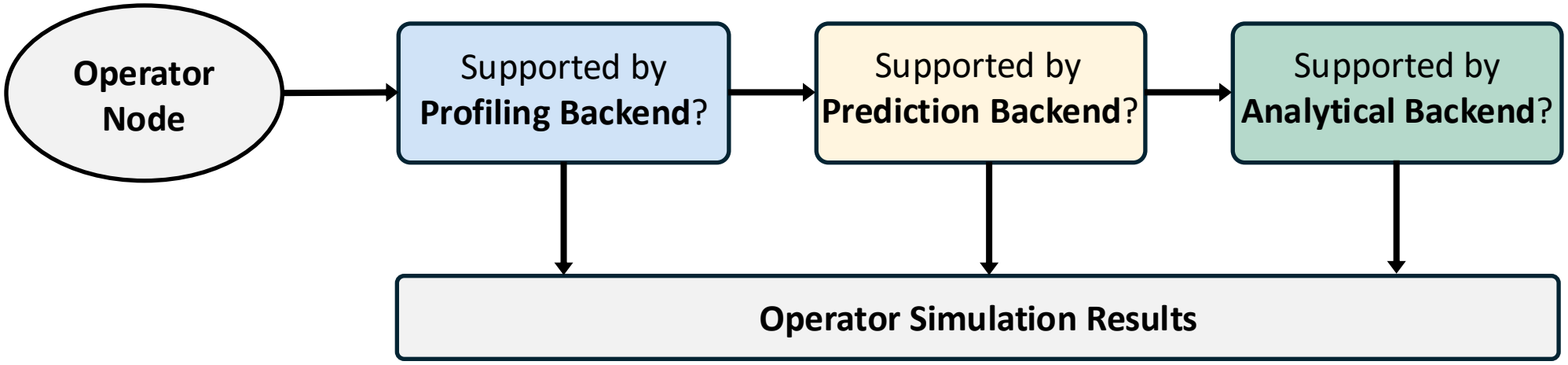
❑ Customizable pass-based analyzers generate **multi-granularity simulation insight**

- Calculates dependency-unaware metrics directly from the graph
- Models memory via liveness-based traversal for dynamic allocation/reuse
- Generates execution timeline traces with operator scheduler



Multi-engine Backend

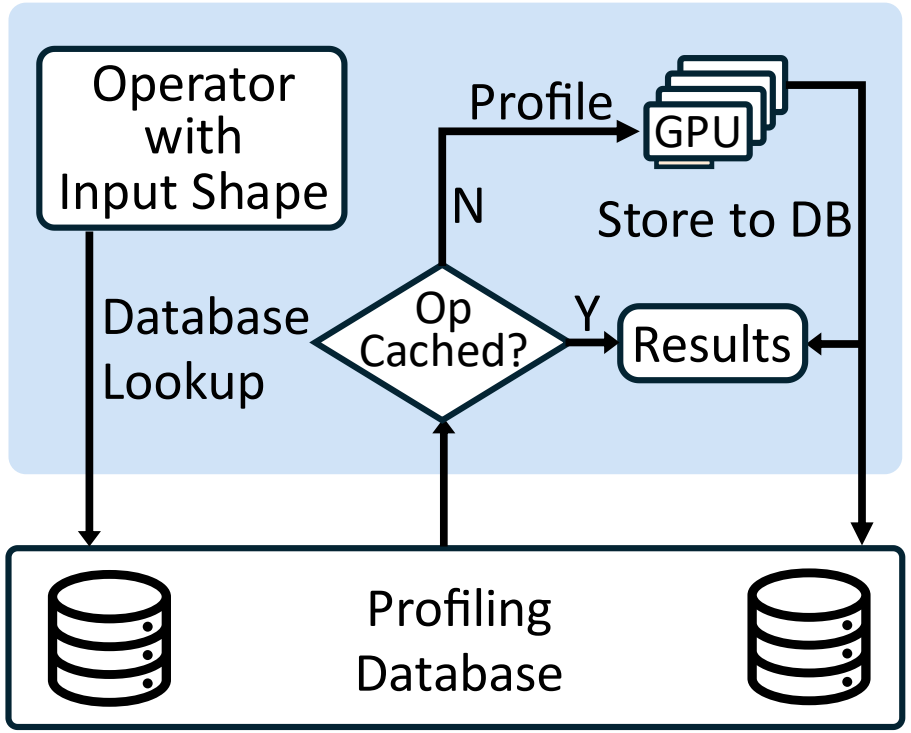
- Supports three different simulation engines through a **seamlessly fused execution flow**
  - Routes operators through multiple engines with fallback mechanisms
  - Ensures compatibility for heterogeneous workloads and emerging models



Multi-engine Backend

❑ **Profiling Backend**: Achieves maximum accuracy via operator profiling

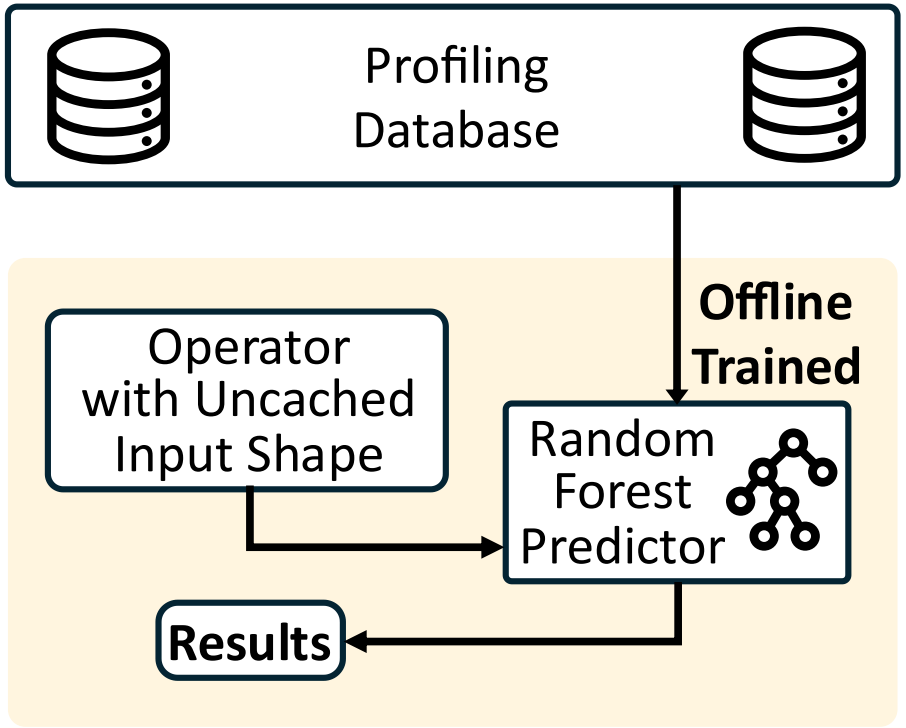
- Automatically generates and dispatches profiling tasks to in-house cluster
- Maintains a dedicated profiling database to store measured results
- Prioritizes database lookups to instantly retrieve and reuse cached latencies



Multi-engine Backend

❑ **Prediction Backend**: Leverages lightweight models to predict latency

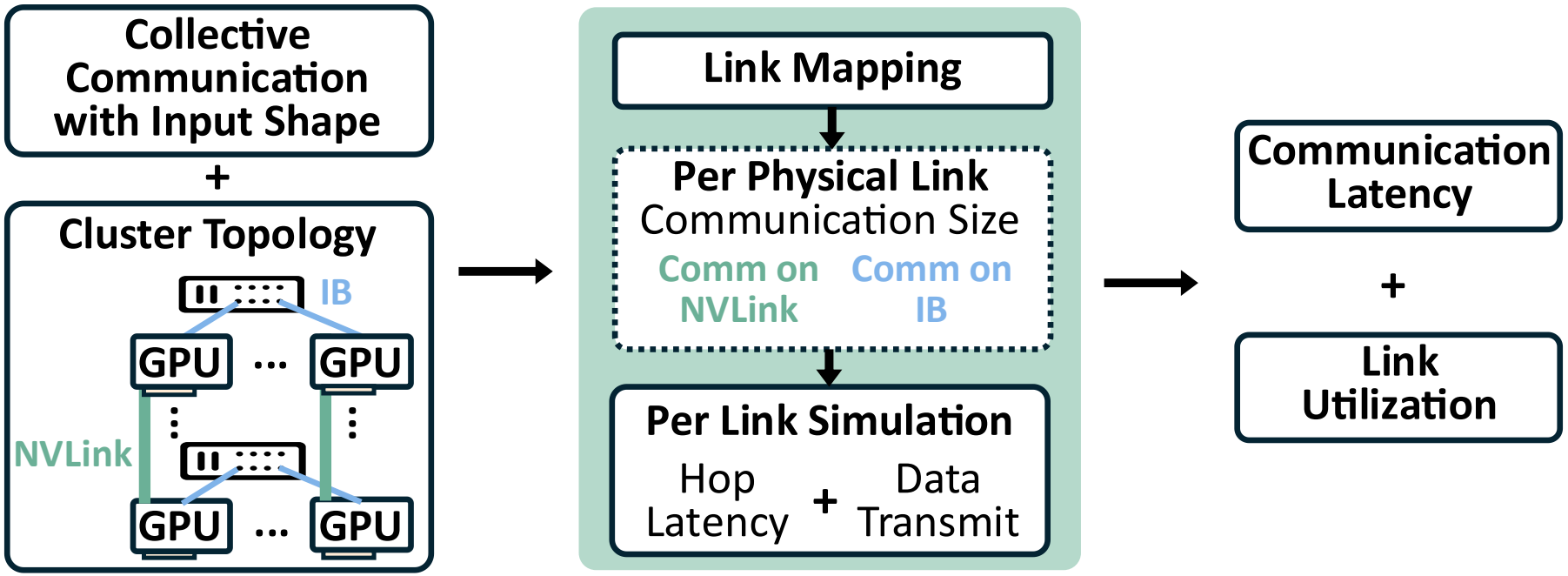
- Predicts performance via operator type and shape on cache misses
- Eliminates real-time hardware execution and speeds up large-scale workloads safely



Multi-engine Backend

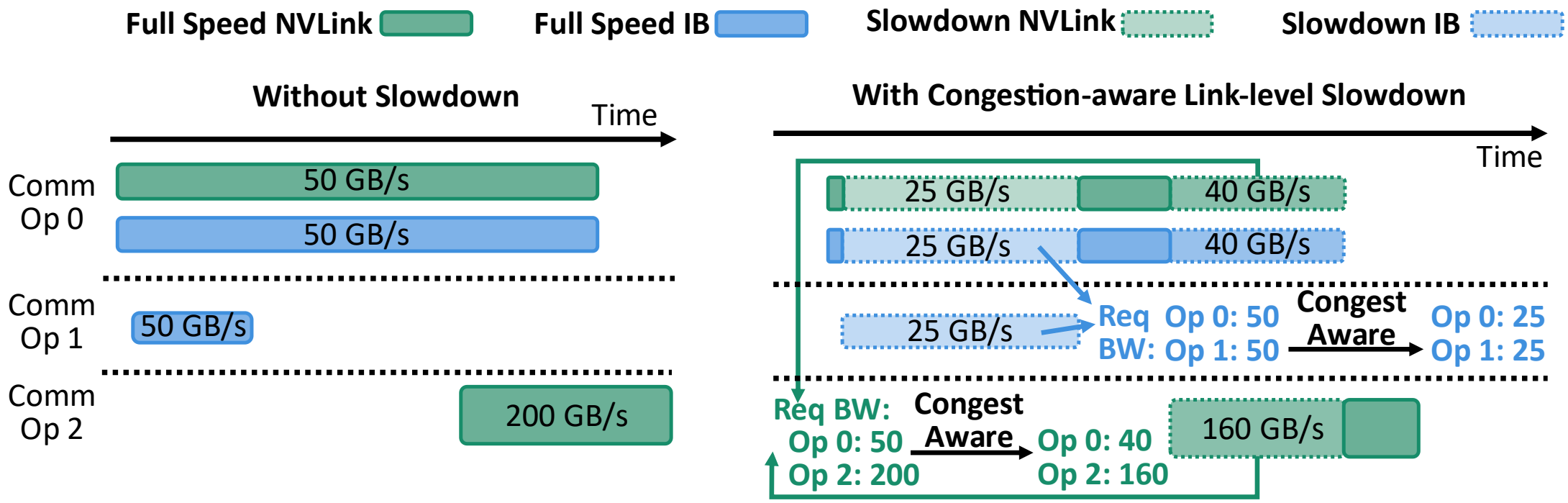
❑ **Analytical Backend**: Serves as a fallback for new operators without profiling data

- Computation Operator: Roofline model analysis
- Communication Operator: Cluster-aware link-wise simulation



Operator Overlap

- **Two-level slowdown modeling** for operator overlaps
  - Ratio-based operator-level slowdown for computation-communication overlap
  - Congestion-aware link-level slowdown for communication-communication overlap

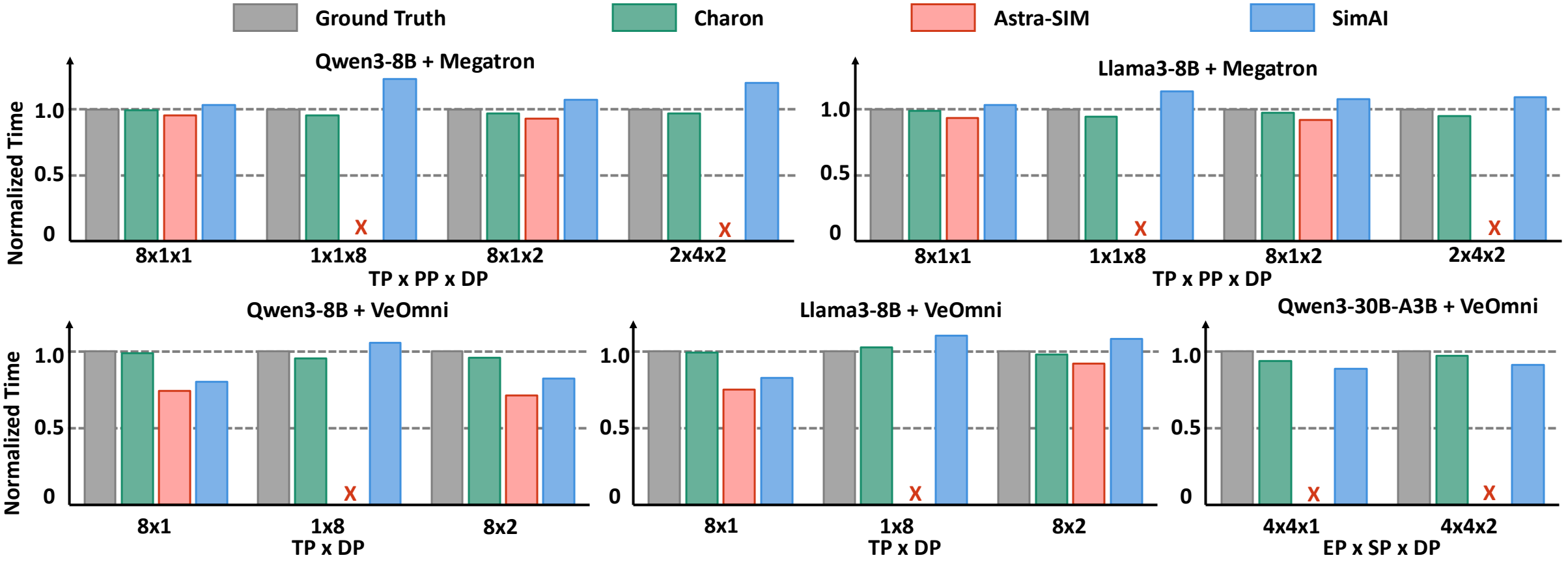


Outline

- Background and Design Motivation for LLM Simulator
- Charon Simulator Design
  - System Design Overview
  - Graph-based Frontend
  - Multi-engine Driven Backend
  - Operator Overlap
- **Simulation Experiments and Case Studies**
- Conclusion

Experiments

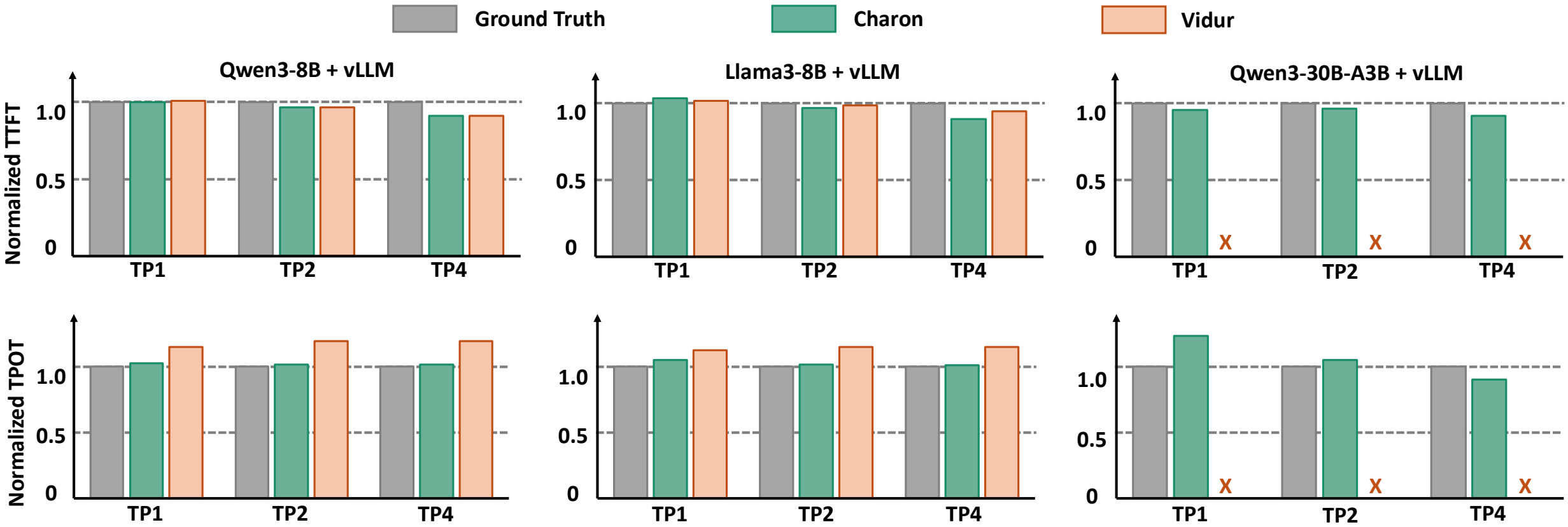
❑ End-to-end Simulation: Training time per iteration



X: Simulation cannot generate valid results / Not supported

Experiments

❑ End-to-end Simulation: Inference TTFT / TPOT

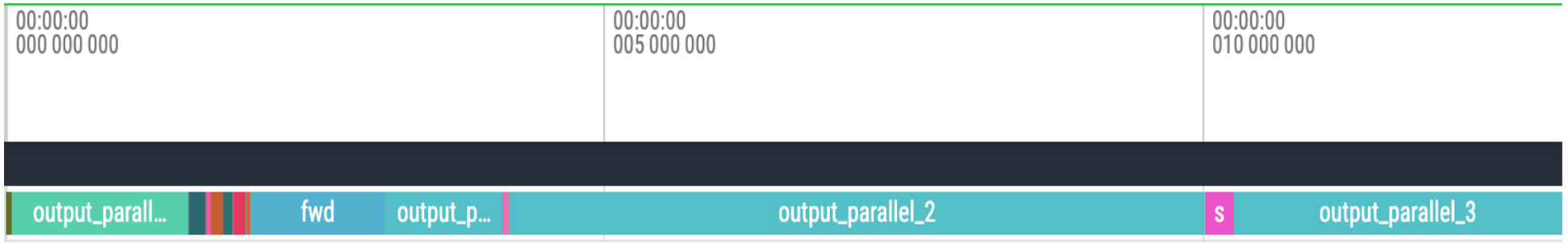


X: Simulation cannot generate valid results / Not supported

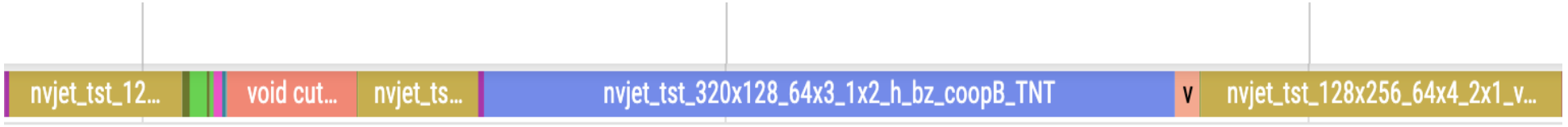
Experiments

❑ Timeline generation

- Generated Trace with Charon Simulator



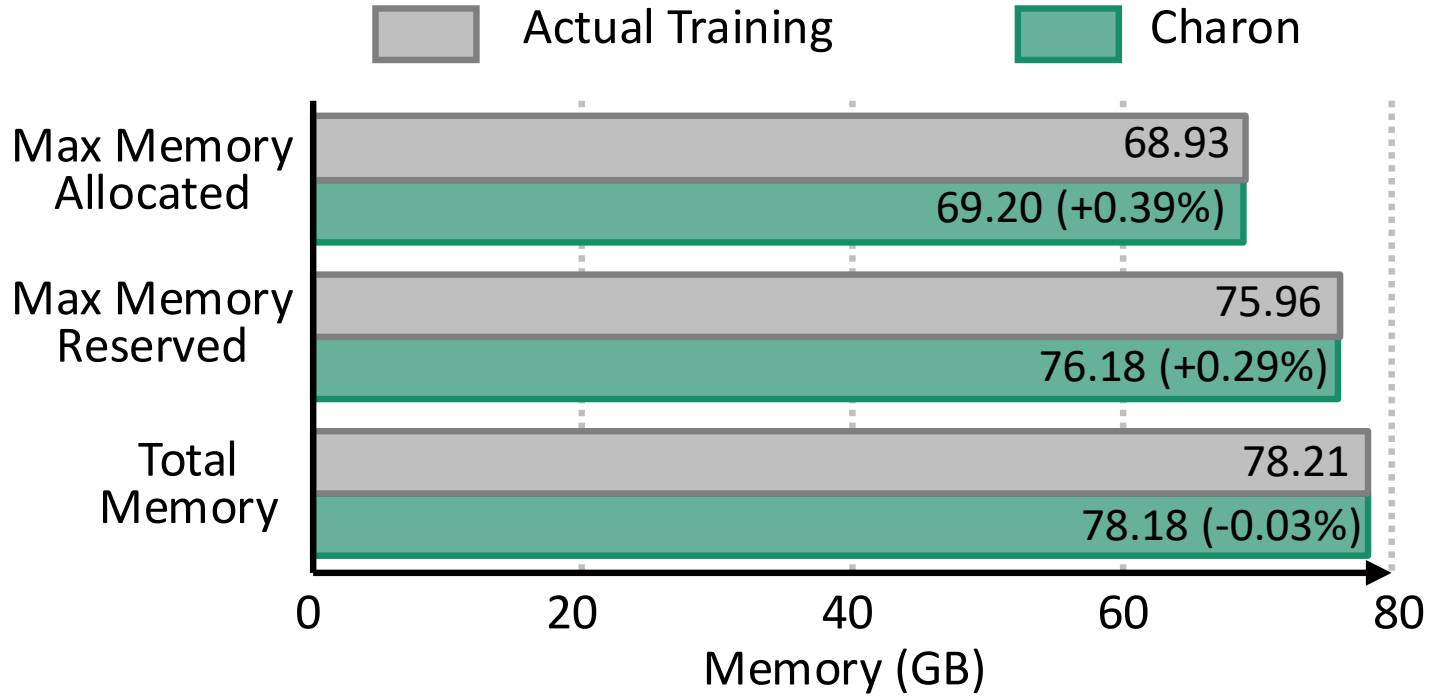
- Profiled Trace with vLLM



Experiments

❑ Memory simulation

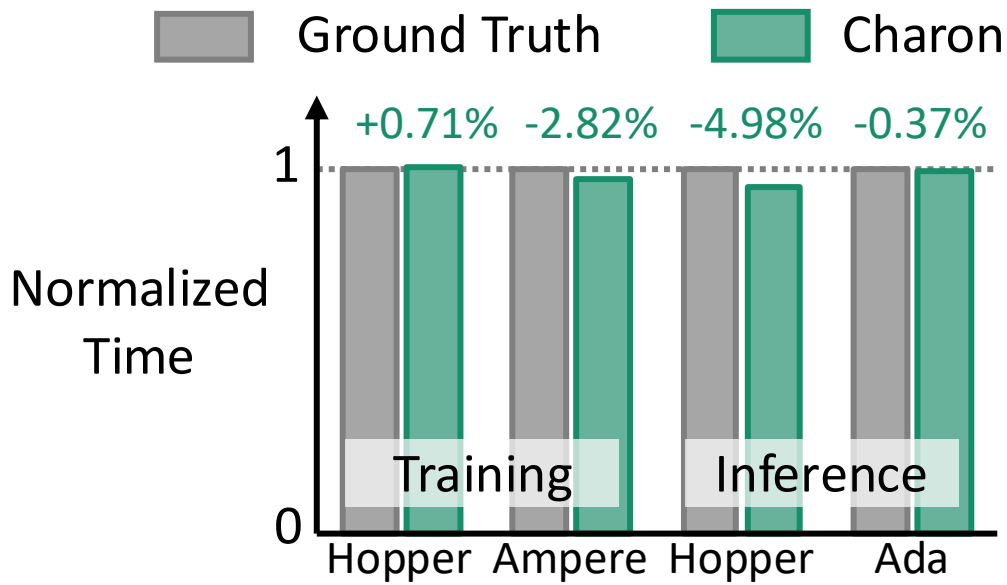
- Qwen3-30B-A3B with FSDP8, Batch Size = 2, Seqlen=8192



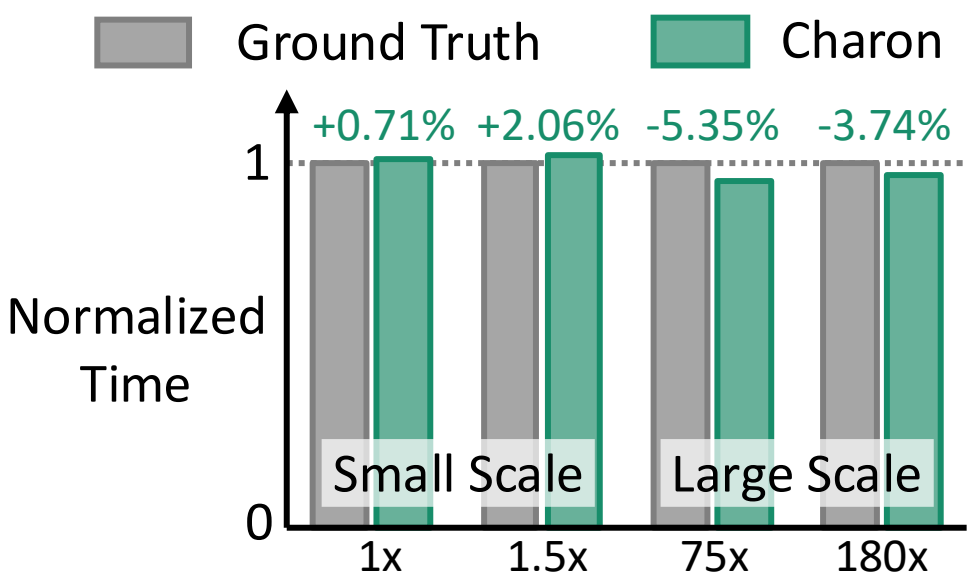
Experiments

❑ Across different GPU and cluster scale

- Different GPU Hardware

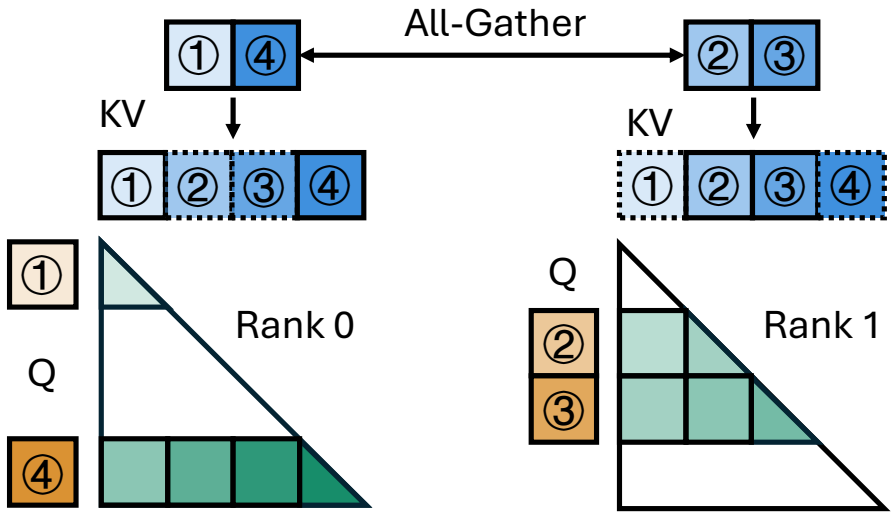


- Different Cluster Scale

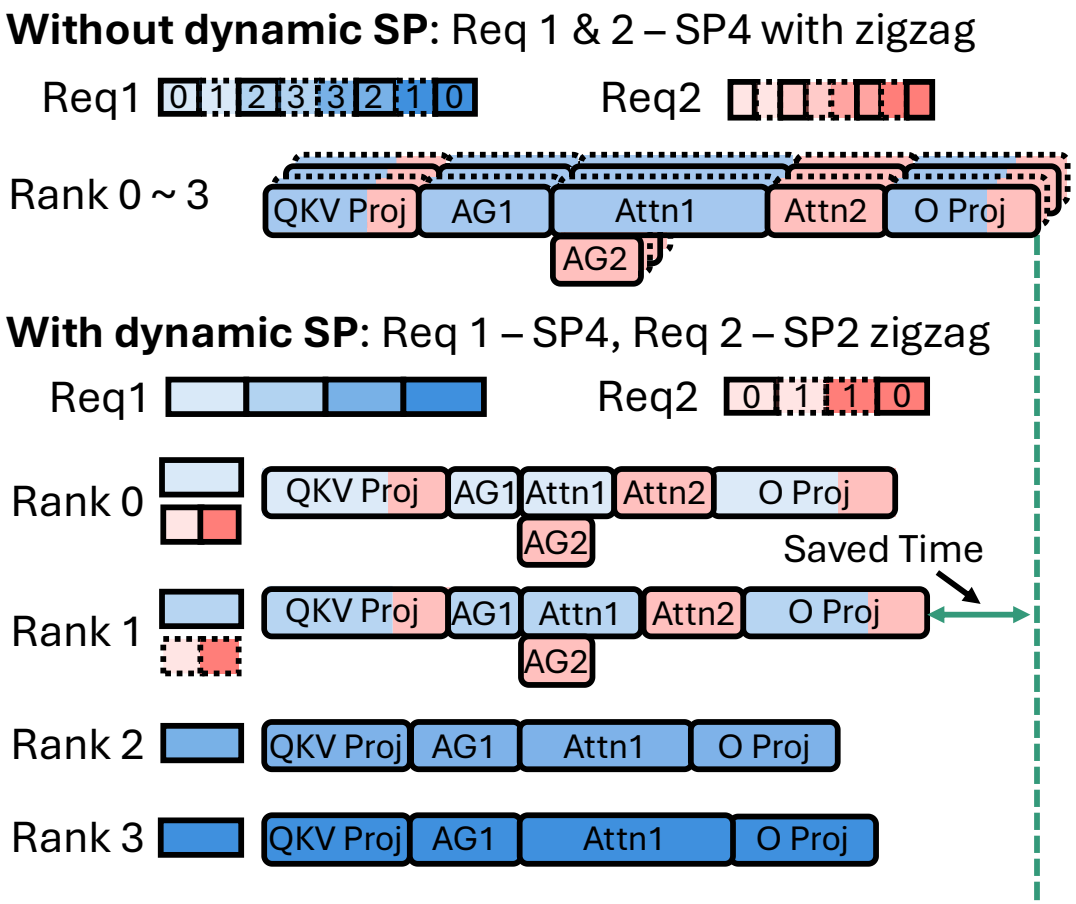


Case Study

❑ Dynamic sequence parallel strategy tuning



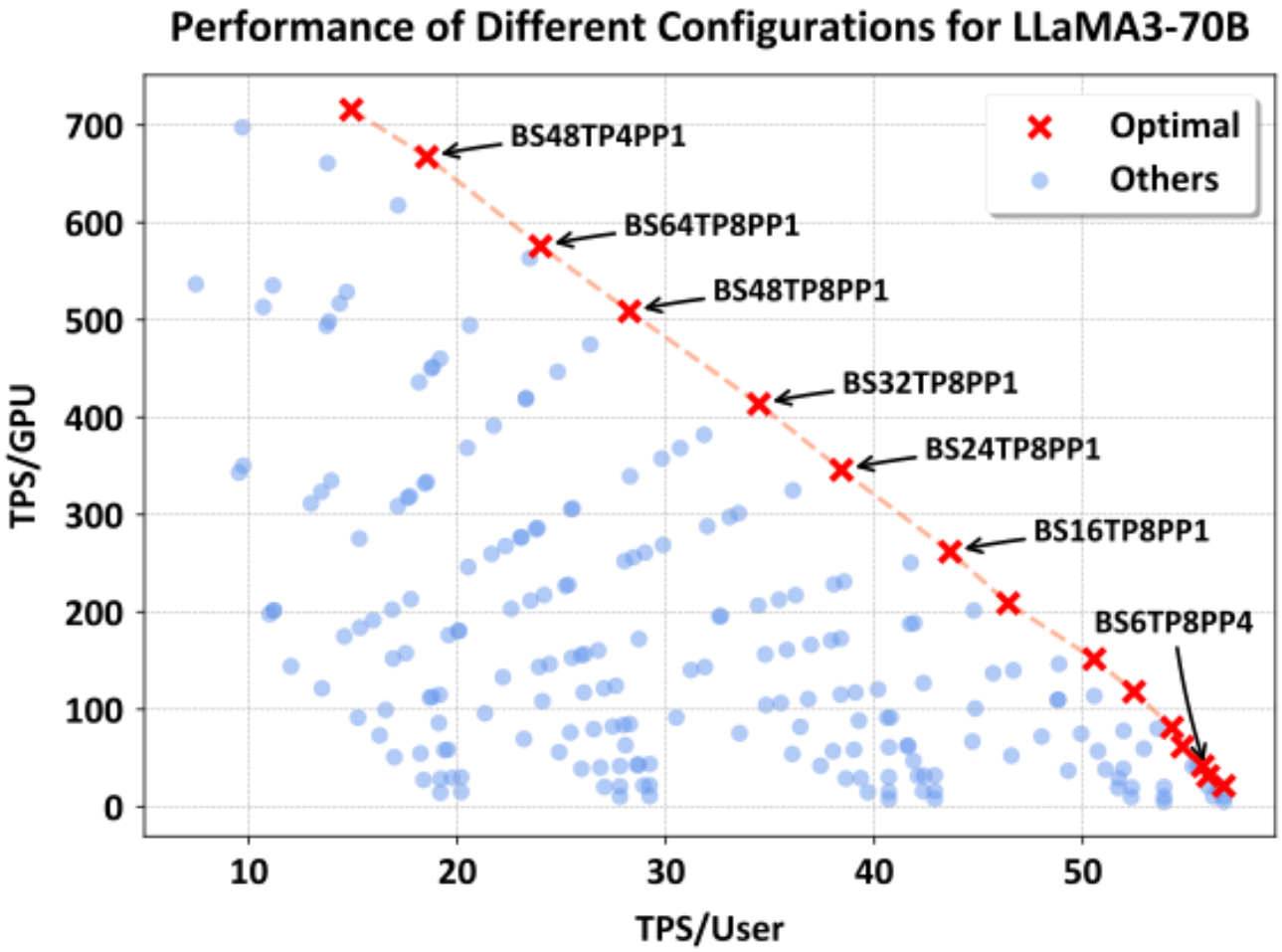
- Assigns different SP configurations per request within a batch with Charon
- Achieves 15% reduction in Attention block latency on Llama-3 70B



Case Study

❑ Optimal inference performance via simulation

- Systematically explores the vast inference design space in two minutes to construct the Pareto frontier
- Validated in production environment and outperformed manually tuned configuration



Outline

- Background and Design Motivation for LLM Simulator
- Charon Simulator Design
  - System Design Overview
  - Graph-based Frontend
  - Multi-engine Driven Backend
  - Operator Overlap
- Simulation Experiments and Case Studies
- **Conclusion**

Conclusion

- We propose **Charon, a unified and fine-grained simulator** supporting end-to-end LLM training and inference simulation
- Natively ingests PyTorch/vLLM/HuggingFace models, **supports flexible optimization and parallelism strategies** via a pass-based graph design
- Delivers **high simulation fidelity through operator-level simulation** using a fused multi-engine backend and operator overlap modeling
- Provides **multi-granularity insights** ranging from high-level system metrics to detailed execution timelines
- Validated across **diverse models, parallelism strategies, hardware architectures, cluster scales**, proven under production environments

Thanks

Q&A