Carnegie Mellon University

Splitwise: Efficient Generative LLM Inference Using Phase Splitting

Patel2024

Chenxi Wan, Pradyut Ganesh, Rutwik Pandit

Splitwise authors

Ricardo Bianchini: PhD (1995), Technical Fellow and Corporate Vice President at Microsoft Azure, IEEE Fellow 2015, ACM Fellow 2016
Saeed Maleki: PhD (2015), Prev. Principal Research Software Development Engineer at Microsoft, Now MTS @ XAi
Íñigo Goiri: PhD (2011), Principal Research SDE at Microsoft
Aashaka Shah: PhD (2023), Senior Researcher at Microsoft Research
Chaojie Zhang: PhD (2022), Senior Research SDE at Microsoft
Esha Choukse: PhD (2019), Principal Researcher at Microsoft Azure
Pratyush Patel: PhD Student at the University of Washington

Introduction

- **Transformer model sizes have grown steadily**
- **LLM clusters are very expensive and power hungry**
- **Inference demand far outweigh that of the training**

Introduction

Generative LLM inference

- prompt computation phase -> computationally intensive and requires the high FLOPs
- token generation phase -> memory bound

GPU

- the HBM capacity and bandwidth on these GPUs has not scaled at the same rate as compute and power

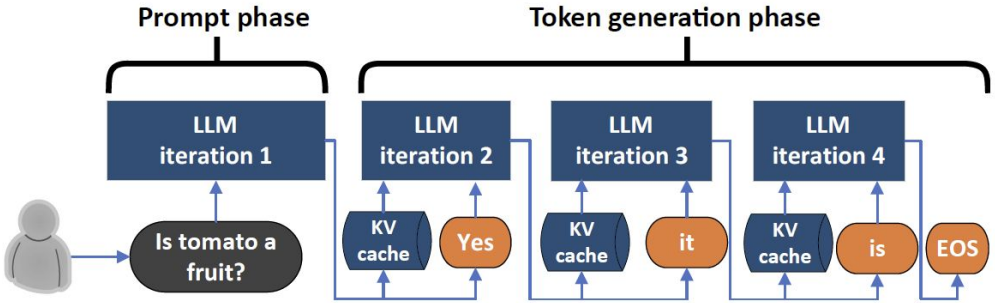


Fig. 1: An LLM inference example.

	A100	H100	Ratio
TFLOPs	19.5	66.9	3.43×
HBM capacity	80GB	80GB	1.00×
HBM bandwidth	2039GBps	3352GBps	1.64×
Power	400W	700W	1.75×
NVLink	50Gbps	100Gbps	2.00×
Infiniband	200GBps	400GBps	2.00×
Cost per machine [5]	$17.6/hr	$38/hr	2.16×

Background–LLM metrics

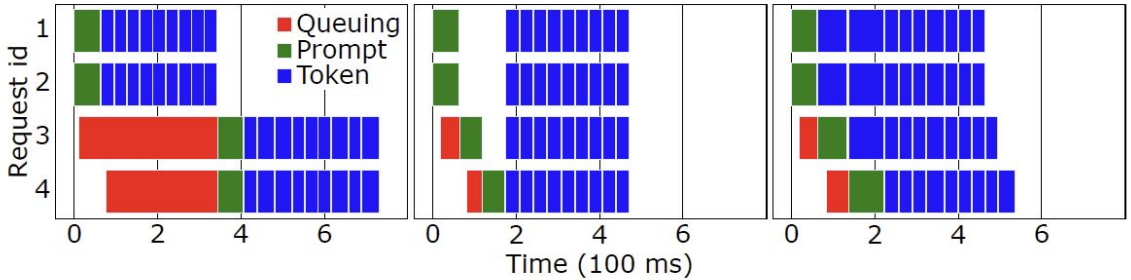
- 4 key inference metrics: E2E latency, TTFT, TBT, and Throughput
- Batch tasks (e.g. summarization) → throughput matters most
- Latency-sensitive tasks (e.g. conversation) → TTFT & TBT matter most

Metric	Importance to user
End-to-end (E2E) latency	Total query time that the user sees
Time to first token (TTFT)	How quickly user sees initial response
Time between tokens (TBT)	Average token streaming latency
Throughput	Requests per second

TABLE II: Performance metrics for LLMs.

Background–Batching of requests

- Batching requests together increases throughput
- 3 batching mechanisms:
  - Request-level batching → simple, but long wait times → high TTFT & E2E latency
  - Continuous batching → shorter TTFT, but hurts TBT tail latency
  - Mixed batching → prompt + token phases run together → best balance ✓



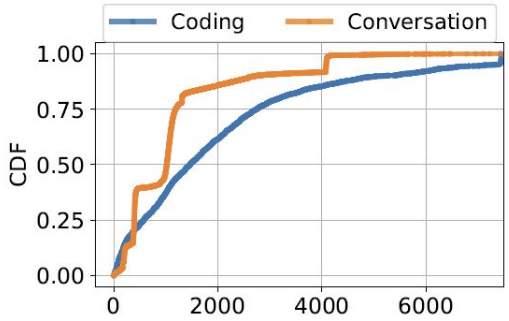
(a) Request-level. (b) Continuous. (c) Mixed.

Characterization – Setup

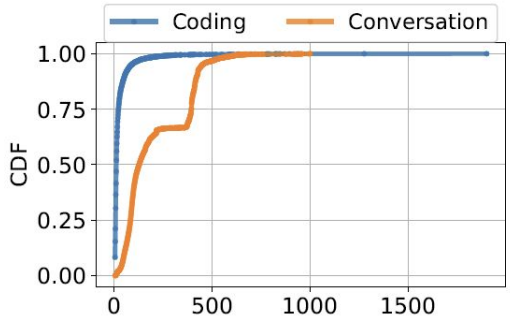
- Goal: characterize prompt vs. token phase to guide Splitwise design
- Production traces from 2 Azure LLM services: coding and conversation
- Models: BLOOM-176B and Llama2-70B on 8× H100

Characterization – Token Distributions

- Coding: large prompts (median 1500 tokens), very few output tokens (median 13)
- Conversation: moderate prompts (median 1020 tokens), more outputs (median 129)
- → Different services have very different prompt/token distributions



(a) Prompt input tokens.



(b) Generated output tokens.

Characterization – Batching

- 60–70% of conversation time: running ≤20 active tokens
- Coding: runs with a single token >20% of the time
- → Mixed continuous batching still leaves most of the batch nearly empty

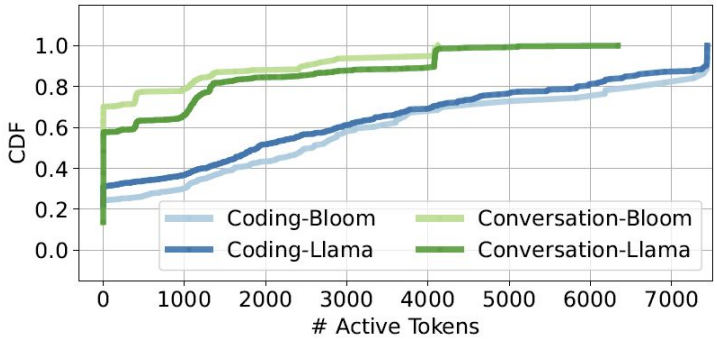
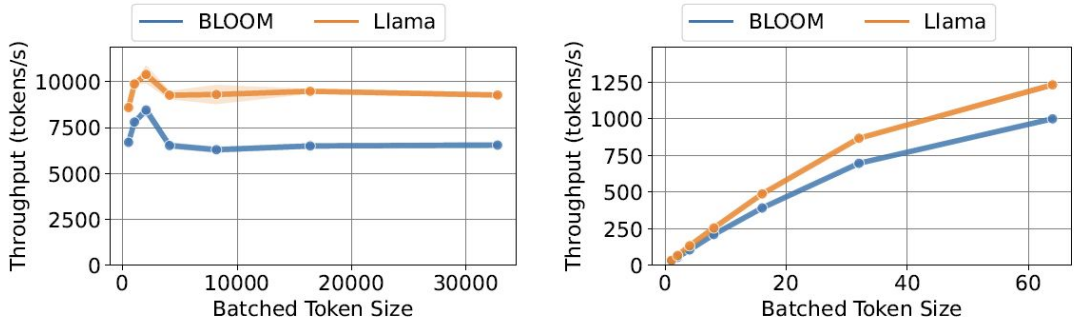


Fig. 4: Cumulative distribution of time spent with various active batched tokens.

Characterization – Throughput

- Prompt phase: throughput degrades after 2048 tokens → batch size must be limited
- Token phase: throughput keeps scaling with batch size

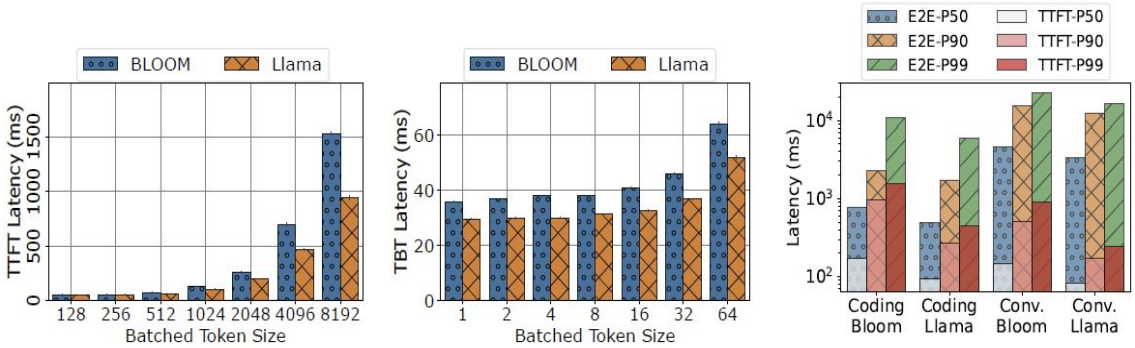


(a) Prompt phase. (b) Token generation phase.

Fig. 6: Impact of batching on the throughput for the 2 LLMs.

Characterization – Latency

- TTFT grows linearly with prompt size → compute-bound
- TBT barely changes as batch size grows
- Most of E2E time is spent in the token phase — even for coding



(a) TTFT by prompt size.

(b) TBT by batch size.

(c) Latencies on prod traces (no batching).

Characterization –Memory

- Batching during the prompt phase is compute-bound, whereas the token phase is limited by memory capacity

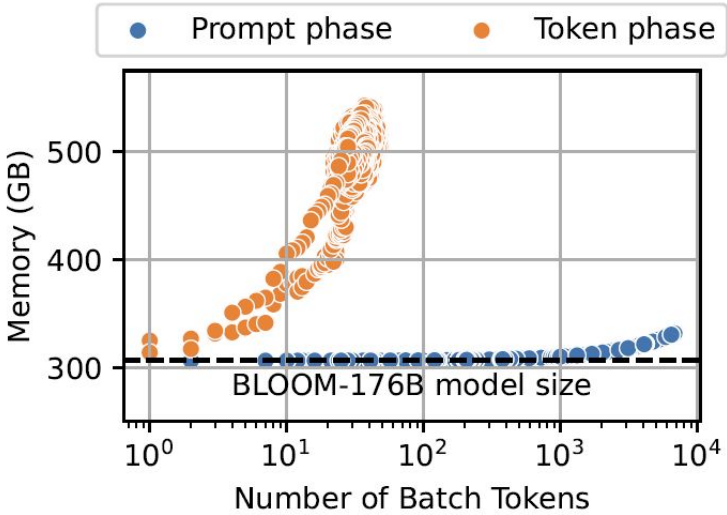
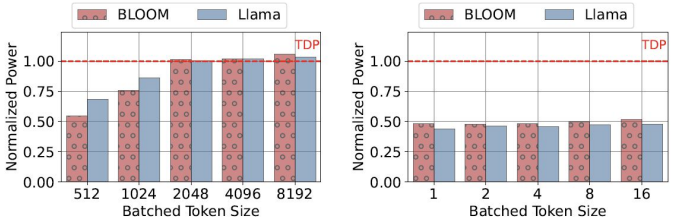


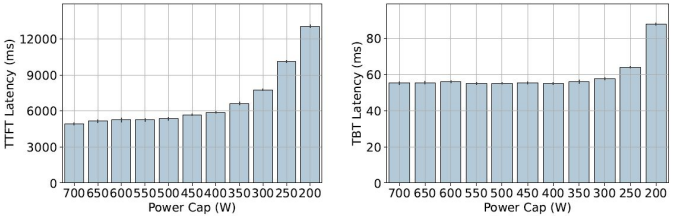
Fig. 7: Required memory with batching in prompt/token phases.

Characterization – Power and GPU



(a) Prompt phase. (b) Token generation phase.

Fig. 8: Maximum and mean power utilization varying the batching size.



(a) Prompt phase. (b) Token generation phase.

Fig. 9: Impact of power cap on the prompt and token generation latency with the maximum batch size possible.

	Coding			Conversation		
	A100	**H100**	**Ratio**	**A100**	**H100**	**Ratio**
TTFT	185 ms	95 ms	0.51×	155 ms	84 ms	0.54×
TBT	52 ms	31 ms	0.70×	40 ms	28 ms	0.70×
E2E	856 ms	493 ms	0.58×	4957 ms	3387 ms	0.68×
Cost [5]	$0.42	$0.52	1.24×	$2.4	$3.6	1.5×
Energy	1.37 Whr	1.37 Whr	1×	7.9 Whr	9.4 Whr	1.2×

TABLE IV: P50 request metrics on A100 vs. H100 without batching on Llama-70B.

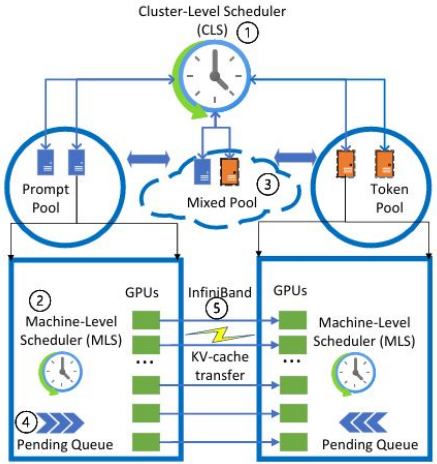
Introduction to the Splitwise Architecture

Core Concept: Physically decouple the prompt computation and token generation phases onto separate machine pools.

Three Machine Pools:

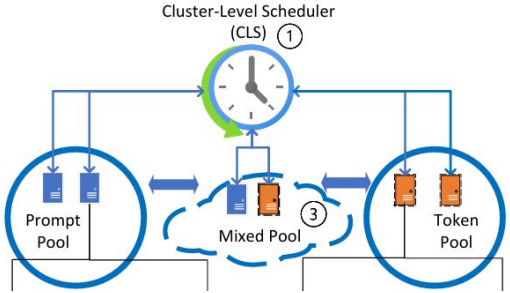
- Prompt Pool: Computes initial prompt, generates KV-cache.
- Token Pool: Executes continuous token generation.
- Mixed Pool: Dynamically scales to handle workload fluctuations.

Two-Level Scheduling: Cluster-Level Scheduler (CLS) manages routing; Machine-Level Scheduler (MLS) manages local batching.



Cluster-Level Scheduling (CLS) - Pool Management

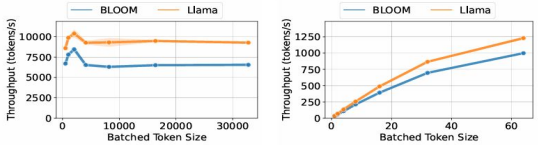
- Dynamic Pool Allocation: Initial assignment is based on expected load and token distributions.
- The Mixed Pool Strategy: Prompt/token machines temporarily enter the mixed pool to resolve queue backlogs and prevent fragmentation.
- Routing Algorithm: Uses Join the Shortest Queue (JSQ) based on pending tokens, not pending requests.
- Simultaneous Assignment: CLS assigns both a prompt and token machine to a request at the exact same time.



Machine-Level Scheduling (MLS) - Local Execution

Prompt Machines (Compute-Bound):

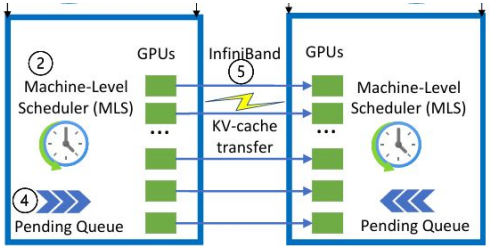
- Uses First-Come-First-Serve (FCFS).
- Constraint: Hard limit of 2048 prompt tokens per batch to prevent throughput degradation.



(a) Prompt phase. (b) Token generation phase.

Token Machines (Memory-Bound):

- Uses FCFS.
- Constraint: Batches as many requests as possible until GPU memory capacity is nearly exhausted.



Mixed Machines:

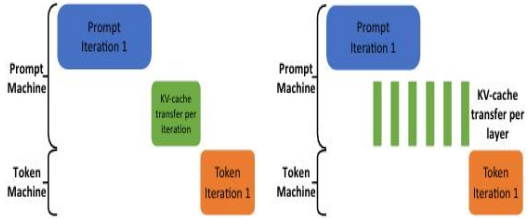
- Strictly prioritizes prompt processing to meet Time to First Token (TTFT) SLOs.
- Preemption: Temporarily pauses active token generation for new prompts, using aging priority to prevent starvation.

Hiding Latency: Optimized KV-Cache Transfer

The Bottleneck: Transferring massive KV-cache state over InfiniBand between machines.
Serialized Transfer (Naive): Waits for full prompt completion before sending; heavily penalizes Time Between Tokens (TBT).
Asynchronous Layer-wise Transfer:

- KV-cache for layer N is transferred over InfiniBand while the GPU computes layer N+1.
- Virtually eliminates network transfer overhead for large prompts.

Adaptive Selection: Uses serialized transfers for small prompts (avoids synchronization overhead) and layer-wise for large prompts.



(a) Serialized KV-cache transfer. (b) Optimized KV-cache transfer per-layer during prompt phase.

Cluster Provisioning & Hardware Variants

- Splitwise-AA: DGX-A100s for both pools (Homogeneous).
- Splitwise-HH: DGX-H100s for both pools (Homogeneous).
- Splitwise-HA (Heterogeneous): DGX-H100 for Prompt Pool + DGX-A100 for Token Pool. Capitalizes on older GPUs' cost-efficiency for memory-bound tasks.
- Splitwise-HHcap (Power-Capped): Token machines power-capped to 70% total (50% per GPU). Token phase suffers no performance impact from the power drop.

	A100	H100	Ratio
TFLOPs	19.5	66.9	3.43×
HBM capacity	80GB	80GB	1.00×
HBM bandwidth	2039GBps	3352GBps	1.64×
Power	400W	700W	1.75×
NVLink	50Gbps	100Gbps	2.00×
Infiniband	200GBps	400GBps	2.00×
Cost per machine [5]	$17.6/hr	$38/hr	2.16×

TABLE I: NVIDIA A100 vs. H100 specifications.

Provisioning Optimization & Search Space

Sizing Methodology: Event-driven simulator determines optimal machine counts.

Simulator Inputs: Target hardware, LLM performance model, request traces, SLO limits, and optimization goal.

Optimization Goals:

- Throughput: Maximize requests per second.
- Cost: Minimize deployment dollars (critical for end-users).
- Power: Minimize total provisioned watts (critical for data centers).

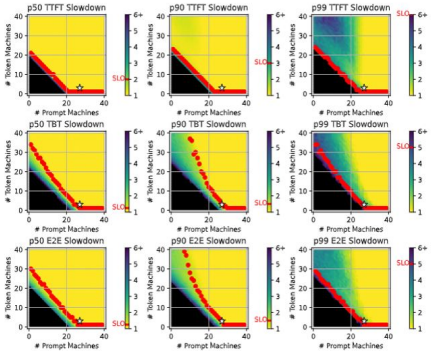


Fig. 12: Design space for provisioning a Splitwise-HH cluster. Cluster configurations targets a peak throughput of 70 RPS. The cost-optimal Splitwise-HH configuration is marked with ⋆ (27 prompt and 3 token machines).

Methodology Overview

Real Hardware Experiments

- 2× DGX-A100 + 2× DGX-H100 on Azure
- Modified vLLM with MSCCL++ for KV-cache transfer
- Zero-copy one-sided put over InfiniBand
- Validates transfer mechanism & measures overhead

Event-Driven Cluster Simulator

- Models pools, schedulers, memory, queues, KV-cache transfer
- Performance model: < 3% MAPE, validated with 50K+ iterations
- Enables large-scale cluster design exploration

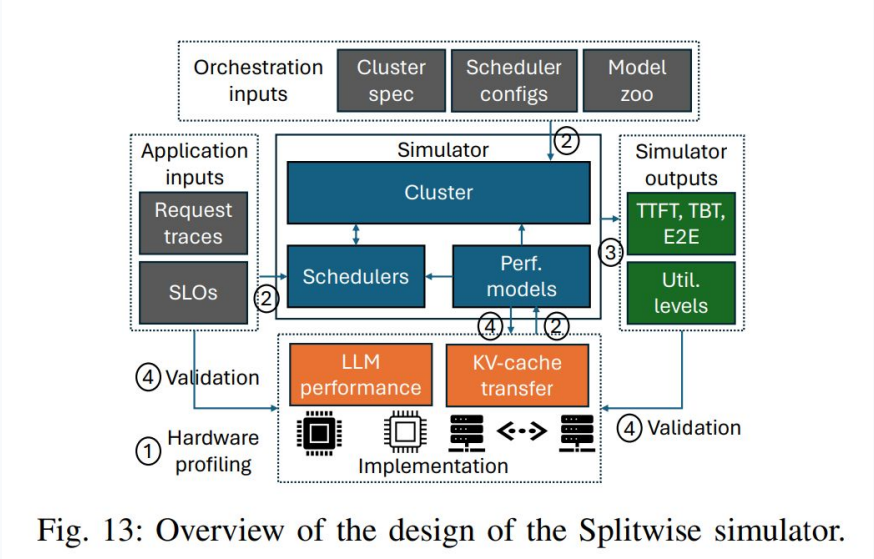


Fig. 13: Overview of the design of the Splitwise simulator.

Figure 13: Simulator design

Carnegie Mellon University

SLOs & Simulation Inputs

	P50	P90	P99
TTFT	2×	3×	6×
TBT	1.25×	1.5×	5×
E2E	1.25×	1.5×	5×

Slowdown relative to uncontended DGX-A100. All 9 SLOs must be met.

Simulation Inputs

- Production traces from Azure (coding + conversation)
- Piecewise linear performance model (< 3% MAPE)
- Communication model benchmarked over InfiniBand
- Poisson arrival rate tuned for different load levels

Baselines

- Baseline-A100: cluster of DGX-A100 machines, mixed batching, no phase splitting
- Baseline-H100: cluster of DGX-H100 machines, mixed batching, no phase splitting
- Both use state-of-the-art mixed continuous batching for fair comparison

Evaluated Splitwise Designs

Design	Prompt Machine	Token Machine	Key Idea
Splitwise-AA	DGX-A100	DGX-A100	Homogeneous, cheaper GPUs
Splitwise-HH	DGX-H100	DGX-H100	Homogeneous, latest GPUs
Splitwise-HHcap	DGX-H100	DGX-H100 (50% pwr cap)	Save power on token machines
Splitwise-HA	DGX-H100	DGX-A100	Best compute for prompts, cheaper tokens

- Power capping works because token generation is memory-bound — 50% power cap has almost no latency impact
- Splitwise-HA uses best compute (H100) for prompts and cheaper hardware (A100) for tokens
- Provisioning: grid search over (# prompt, # token) machine combinations via simulator
- Workload shapes pool sizing: coding → more prompt machines; conversation → more token machines

KV-Cache Transfer: Minimal Overhead

- Serialized transfer latency grows linearly with prompt size
- Per-layer optimized transfer: ~constant overhead (~5ms H100, ~8ms A100)
- Transfer overhead vs. prompt computation: < 7%
- End-to-end latency impact: only 0.8%
- 2nd token delay: 16.5% (optimized) vs. 64% (naive serialized)
- Automatically picks serialized for small prompts, per-layer for large

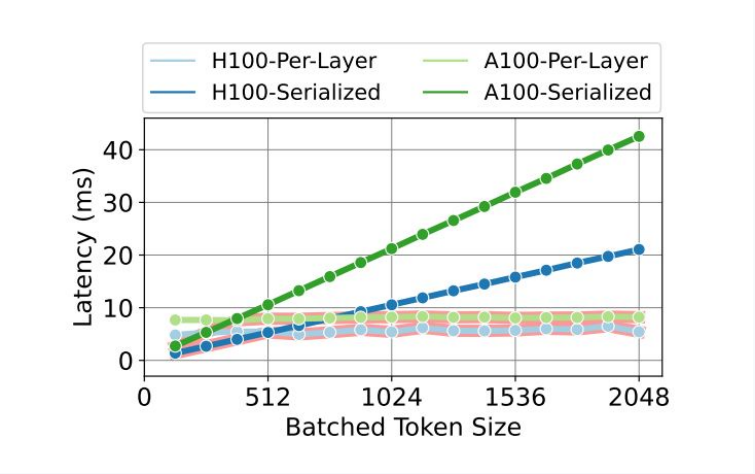


Figure 14: KV-cache transfer latency

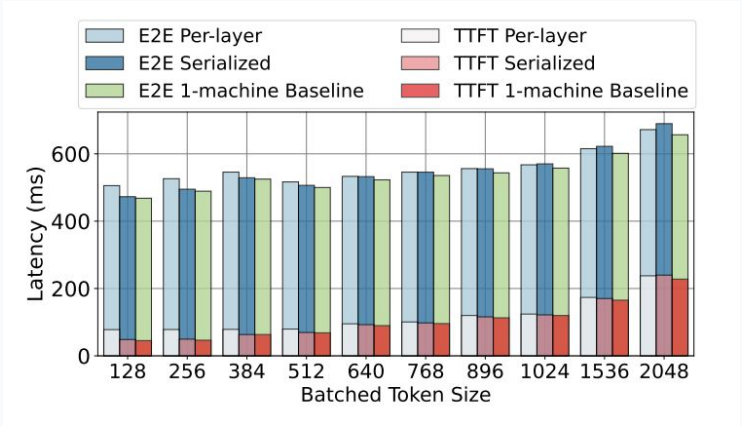


Figure 15: E2E & TTFT overhead

Carnegie Mellon University

Iso-Power Throughput: Coding Trace

- Fixed power budget = 40× DGX-H100 (fits 70 DGX-A100)
- Baseline-H100 suffers high TBT at load (mixed batching + large prompts)
- Splitwise-HH, HHcap, AA all outperform Baseline-H100
- Splitwise-HA bridges gap: low TTFT + high throughput
- Mixed pool activates at high load (visible after 90 RPS)
- Coding: median 1500 prompt tokens, 13 output tokens

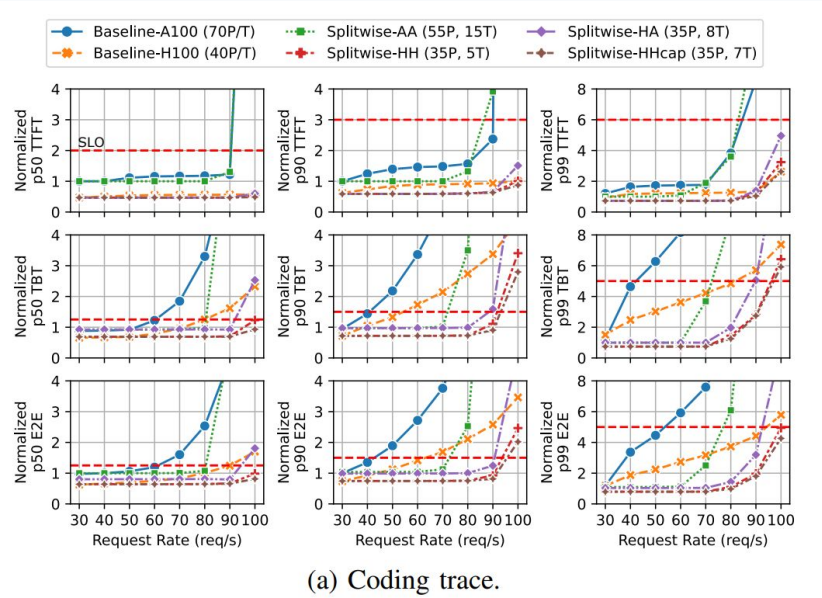


Figure 16a: Coding trace latency metrics

Carnegie Mellon University

Iso-Power Throughput: Conversation Trace

- Splitwise-HHcap dominates on all metrics
- Longer token generation → token machines stay busy and well-utilized
- Power capping token machines gives free power savings
- More token machines provisioned: (25P, 15T) for HH vs (35P, 5T) in coding
- Conversation: median 1020 prompt tokens, 129 output tokens

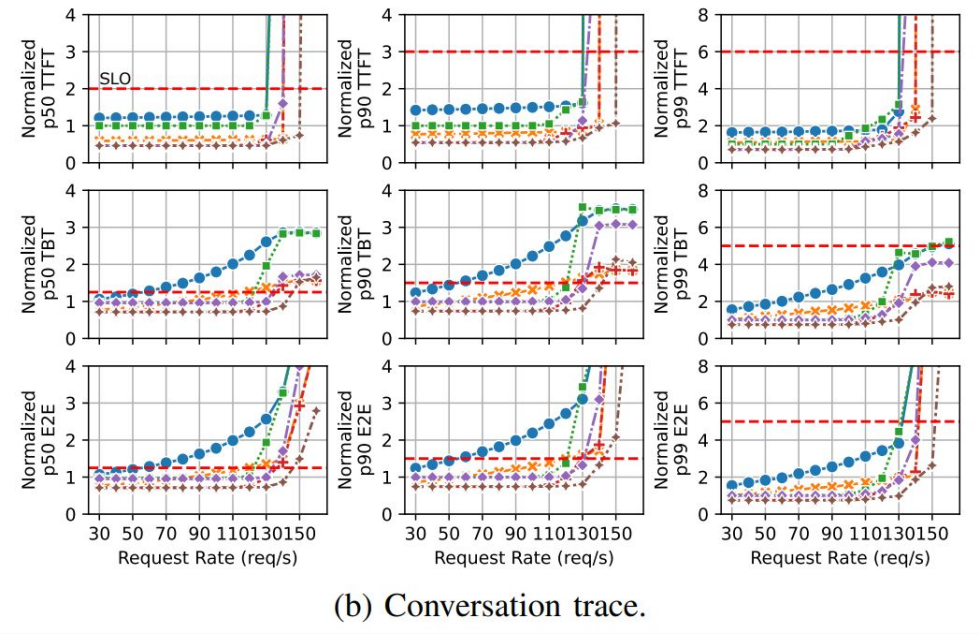
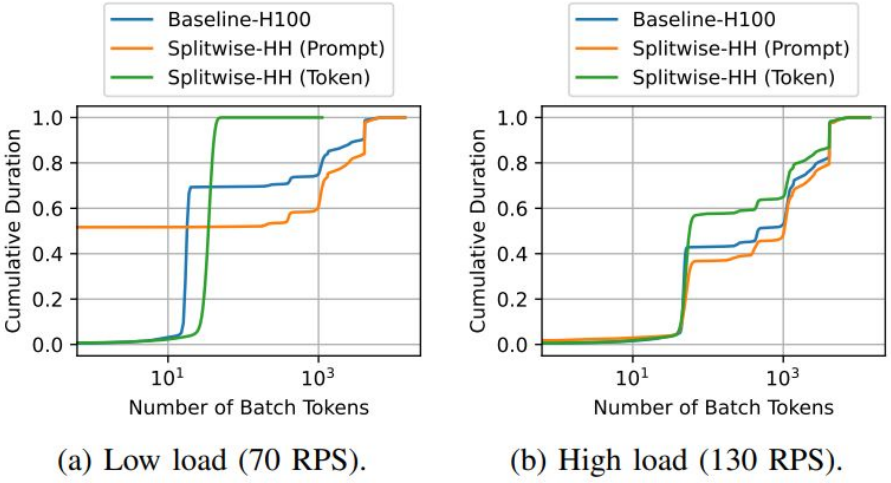


Figure 16b: Conversation trace latency metrics

Carnegie Mellon University

Why Splitwise Batches Better

- Low load (70 RPS): Baseline-H100 spends 70% of time running ≤ 15 tokens
- Splitwise prompt machines: idle or running large prompt batches efficiently
- Splitwise token machines: batch tokens much more effectively
- High load (130 RPS): mixed pool usage increases, batch sizes converge
- Splitwise still maintains latency advantage at high load



(a) Low load (70 RPS). (b) High load (130 RPS).

Fig. 17: Cumulative distribution of time spent at various batched token sizes for iso-power throughput-optimized design.

Figure 17: Batch token size distributions

Carnegie Mellon University

Throughput-Optimized Cluster Summary

Iso-Power (same power budget)

- Splitwise-AA: 2.15× more throughput than Baseline-A100, same power and cost
- Splitwise-HA: 1.18× more throughput, 10% lower cost, same power

Iso-Cost (same cost budget)

- Splitwise-AA: 1.4× more throughput than Baseline-H100
- Tradeoff: 25% more power, 2× physical space
- Uses older, cheaper, more available A100 GPUs

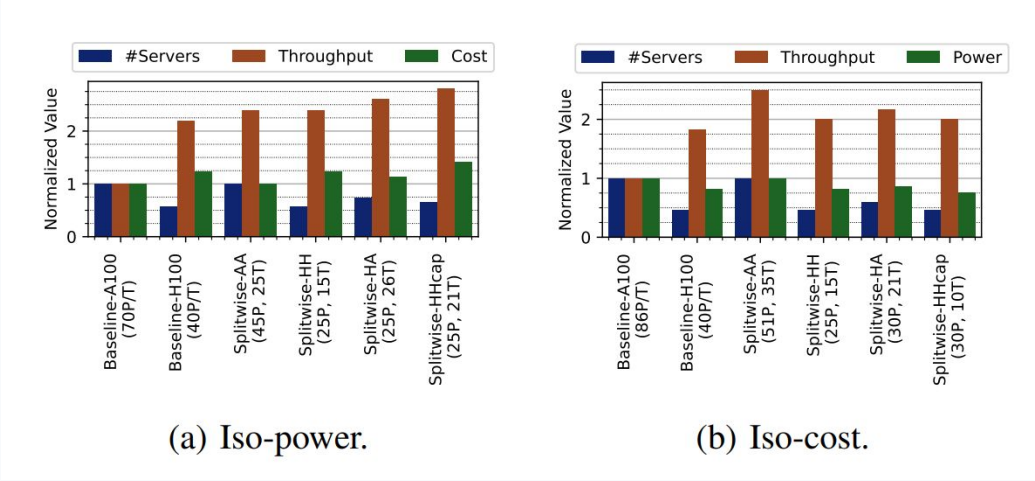


Figure 18: Throughput-optimized summary (iso-power & iso-cost)

Carnegie Mellon University

Iso-Throughput: Power & Cost Optimized

Power-Optimized (match throughput, minimize power)

- Splitwise-HHcap: same throughput as Baseline-H100 at 25% lower power
- Same cost and space — clear win for power-constrained CSPs

Cost-Optimized (match throughput, minimize cost)

- Splitwise-AA: same throughput as Baseline-H100 at 25% lower cost
- Homogeneous designs unchanged between power/cost optimizations

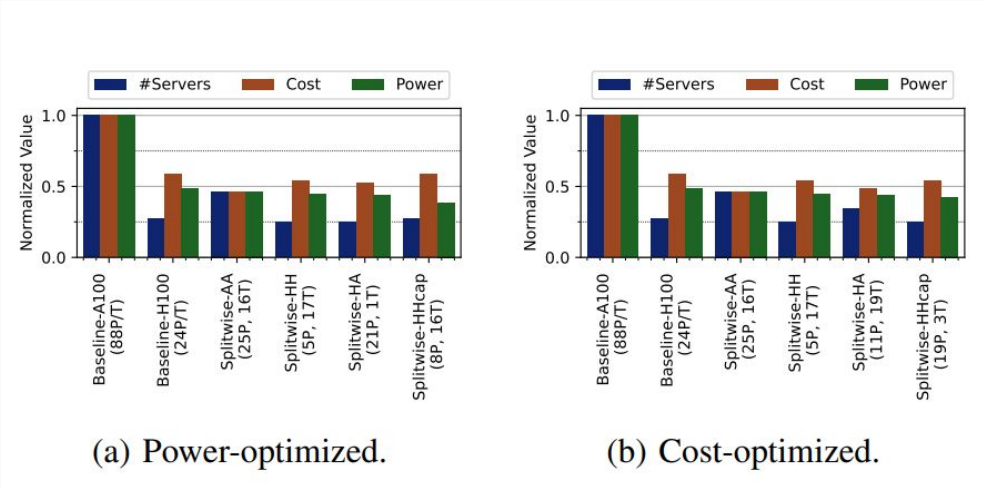


Figure 19: Iso-throughput summary (power & cost optimized)

Carnegie Mellon University

Robustness to Workload Changes

Test 1: Conversation trace on coding-optimized cluster

- Splitwise-AA, HH: adapt via mixed pool, no throughput loss
- Splitwise-HA, HHcap: ~7% throughput reduction
- All Splitwise designs still beat both baselines

Test 2: Llama-70B on BLOOM-176B cluster

- All Splitwise designs outperform baselines at higher load
- HH and HHcap achieve best latency consistently

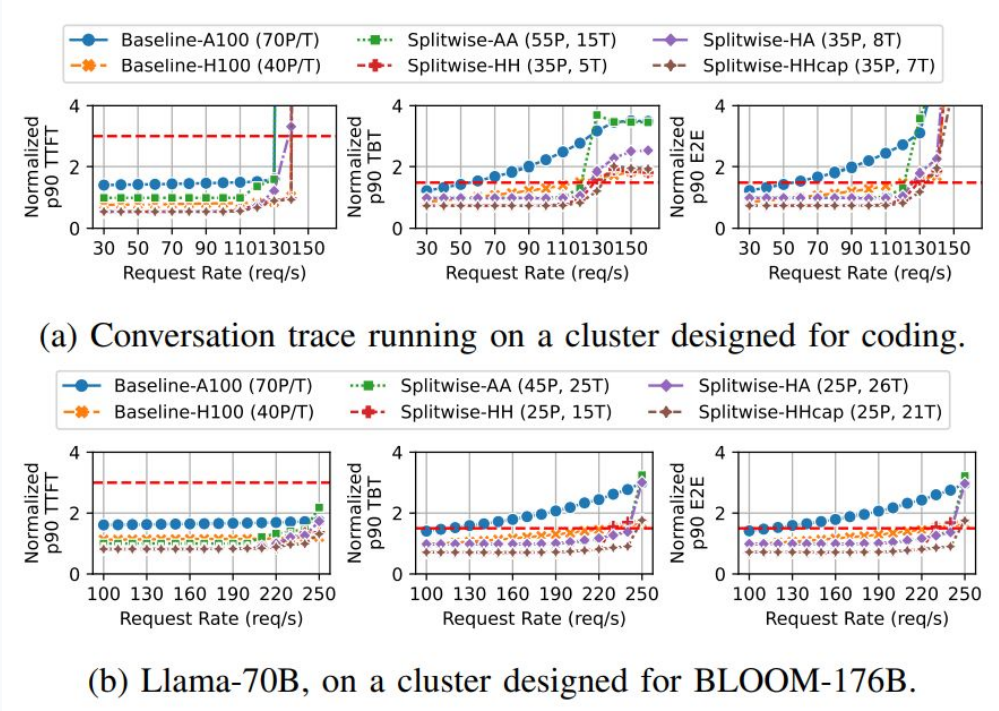


Figure 20: Latency under mismatched workloads

Carnegie Mellon University

Batch Jobs & Discussion

Batch Jobs (no latency SLOs)

- Under max stress, all machines enter mixed pool → Splitwise devolves to baseline
- Splitwise-AA = Baseline-A100 at 0.89 RPS/$;
- Splitwise-HH = Baseline-H100 at 0.75 RPS/$
- Splitwise is designed for latency-sensitive workloads; gracefully degrades for batch jobs

Discussion Highlights

- Extends to all autoregressive transformer LLMs including MoEs
- Applicable to alternative hardware (AMD MI-250, Intel Sapphire Rapids with HBM)
- Interconnect: even 10× lower bandwidth may suffice; KV-cache compression could help
- No accuracy impact — lossless KV-cache transfer, same inference parameters
- Fault tolerance: restart from scratch on failure; optional KV-cache checkpointing

Key Takeaways

- 1.4× higher throughput at 20% lower cost (iso-cost)
- 2.35× more throughput under same cost and power (iso-power)
- 25% power savings at same throughput (iso-throughput)
- KV-cache transfer overhead < 1% of E2E latency
- Robust to workload and model changes via mixed pool
- Graceful degradation — never worse than baseline
- Open-source implementation and simulator

Discussion: Summary Question #1

What Did the Paper Get Right?

- Layer-wise pipelined KV-cache transfer is the key engineering insight that makes the idea actually work
- Mixed pool design elegantly handles load imbalance without wasting machines
- Built on vLLM with real Azure traces — measurements are trustworthy and conclusions are operationally meaningful

Discussion: Summary Question #2

What Did the Paper Get Wrong?

- Paper implicitly says A100 more cost optimized for same throughput if decode heavy workload.
- Why do companies move to next generation GPUs with even higher FLOP/BW ?
- Hint: Is it really Robust to Workload Changes ?