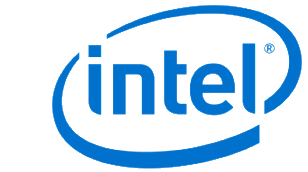


PIM Is All You Need: A CXL-Enabled GPU-Free System for Large Language Model Inference

Yufeng Gu*, Alireza Khadem*, Sumanth Umesh, Ning Liang,
Xavier Servot, Onur Mutlu, Ravi Iyer, and Reetuparna Das

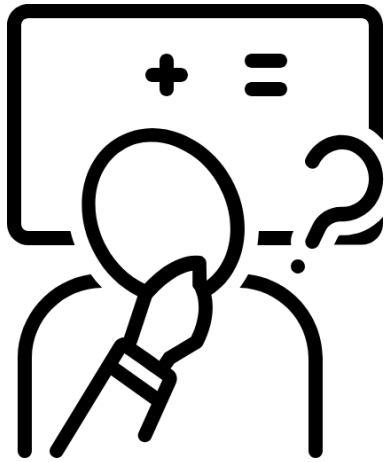
* Equal Contribution

April 2025

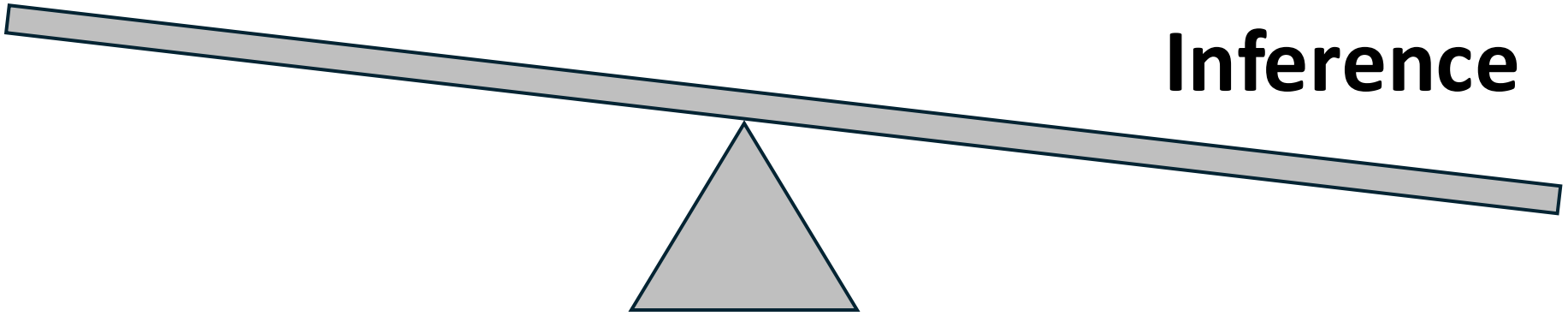
Open-source: https://github.com/Yufeng98/CENT


Training

2x
Carbon
Footprint

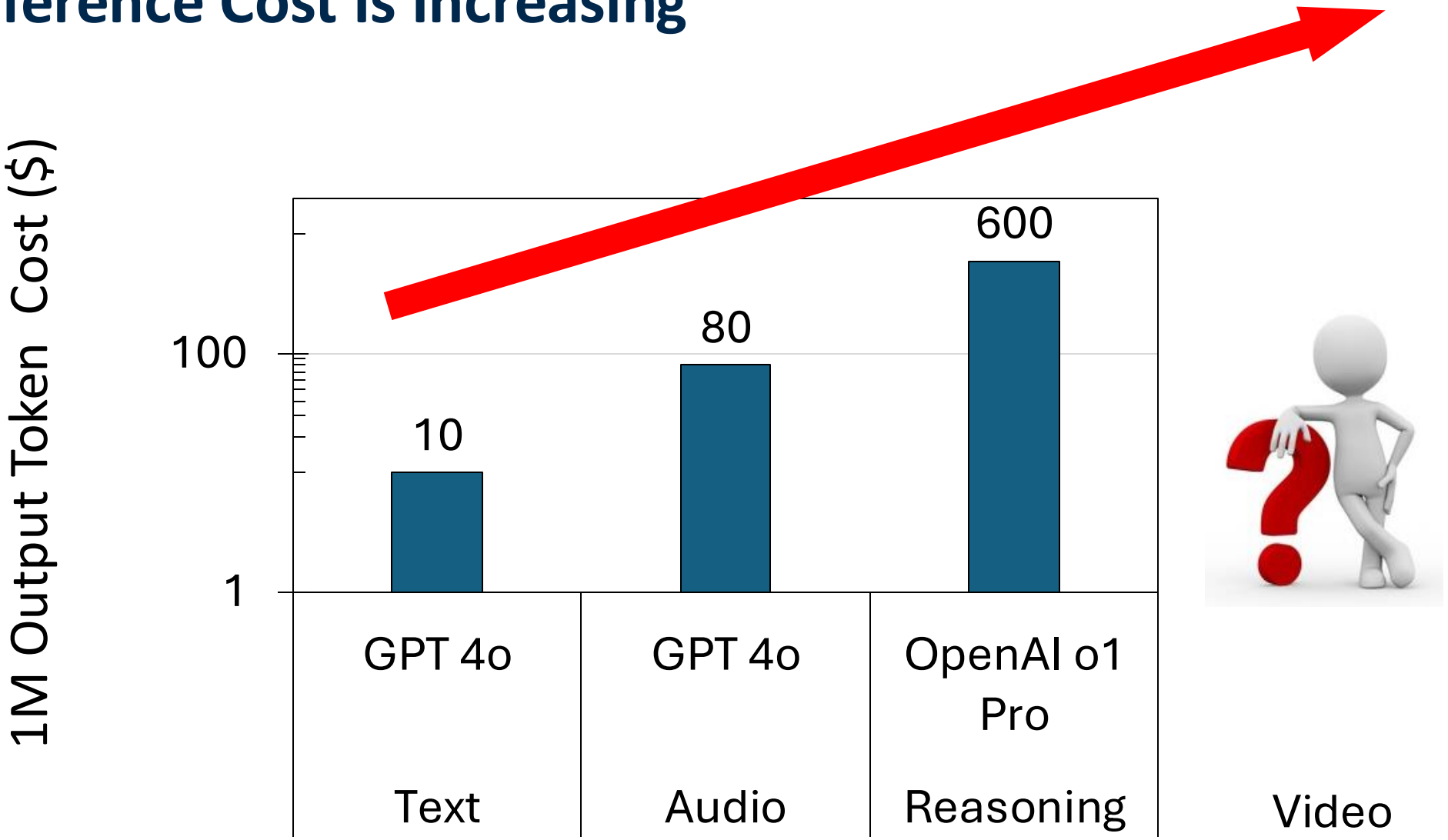


Inference



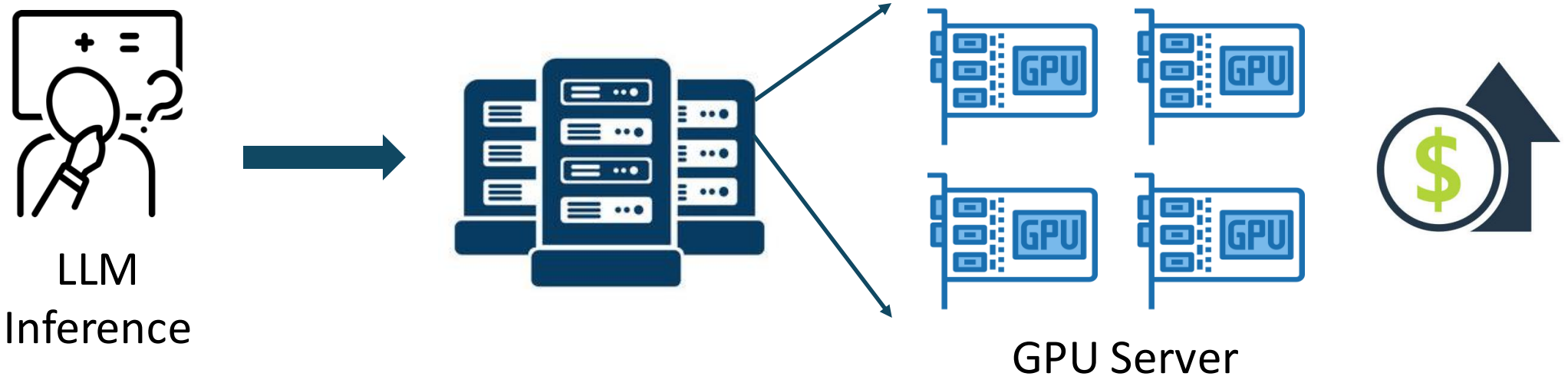
[1] Meta AI, Sustainable AI: Environmental Implications, Challenges and Opportunities, MLSys'22

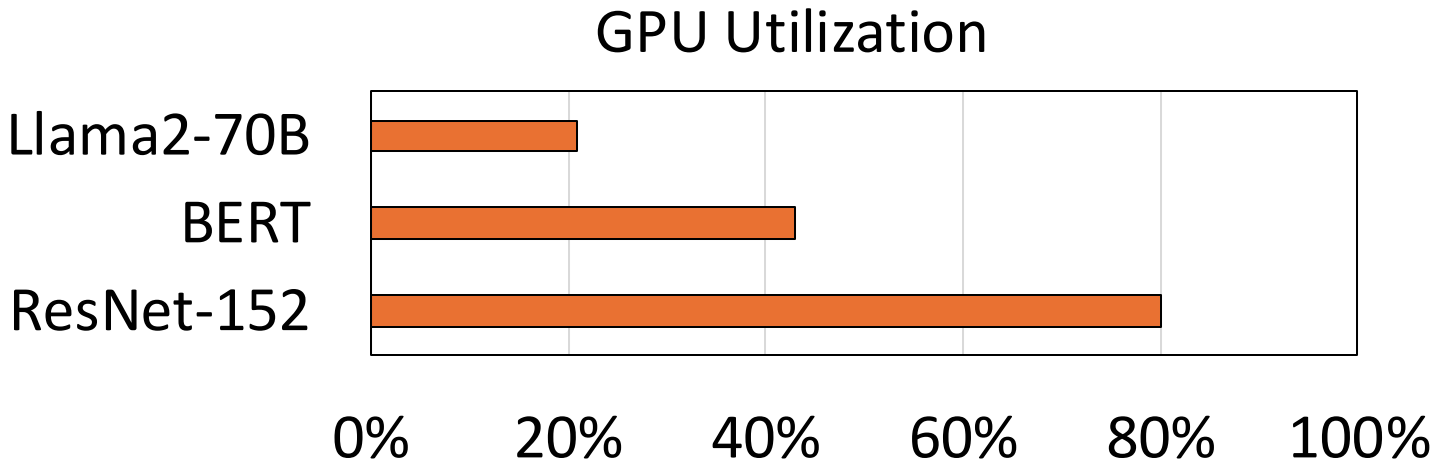
LLM Inference Cost Is Increasing



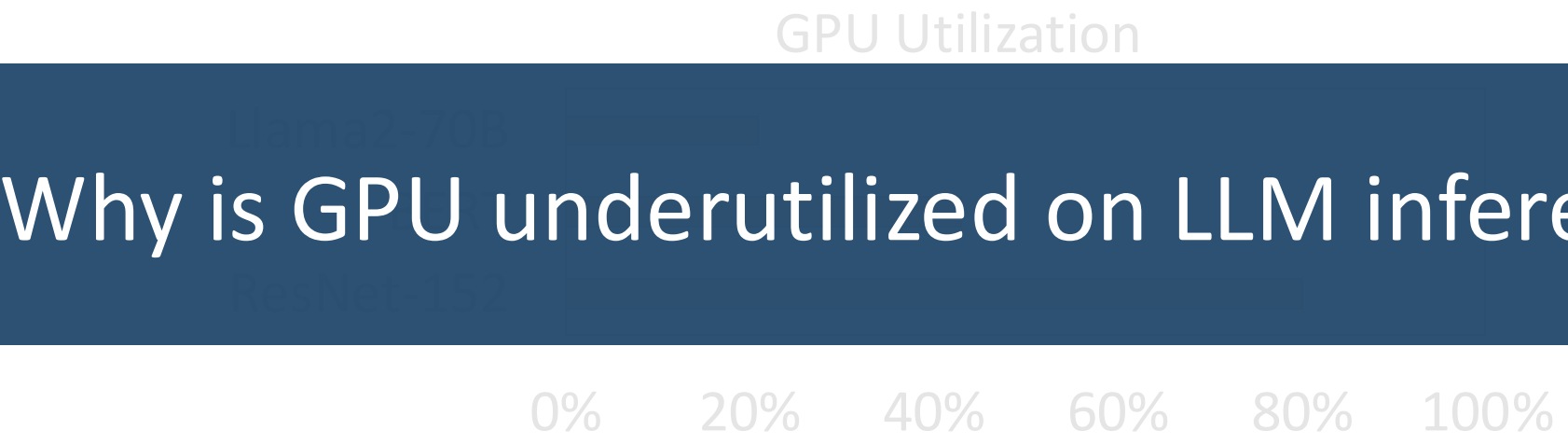
[1] https://openai.com/api/pricing/

Expensive GPU Server Is under Low Utilization





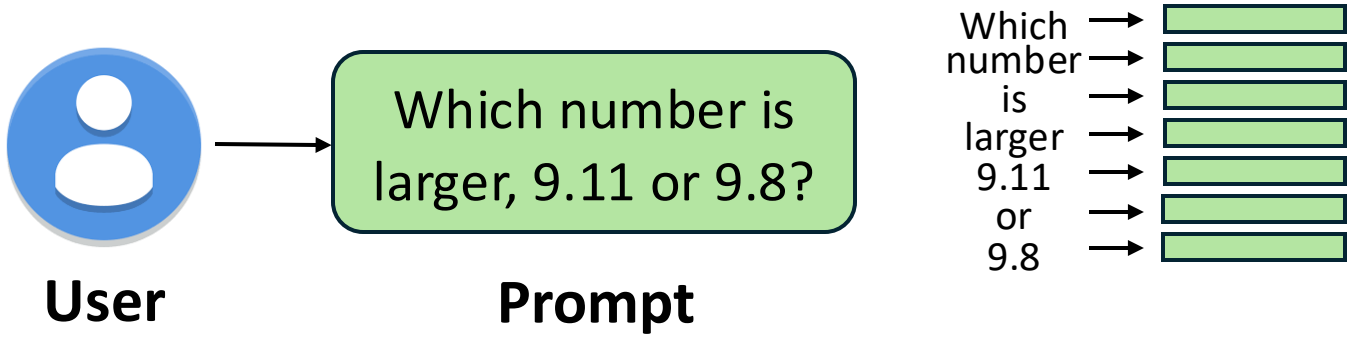
Expensive GPU Server Is under Low Utilization

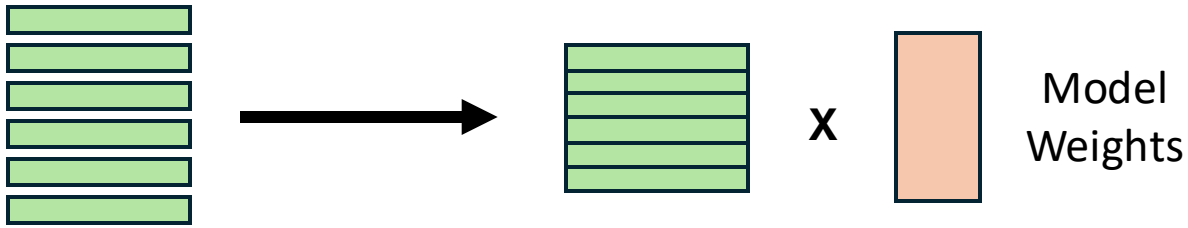


Why is GPU underutilized on LLM inference?

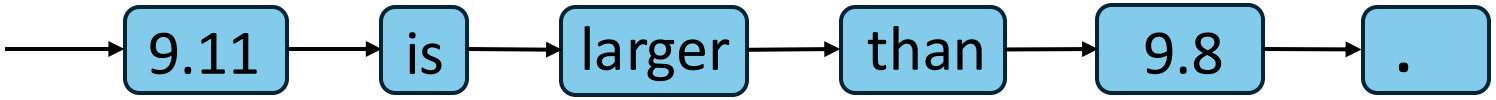
How does LLM Inference work?

Prefill Stage
(Prompt Summarization)

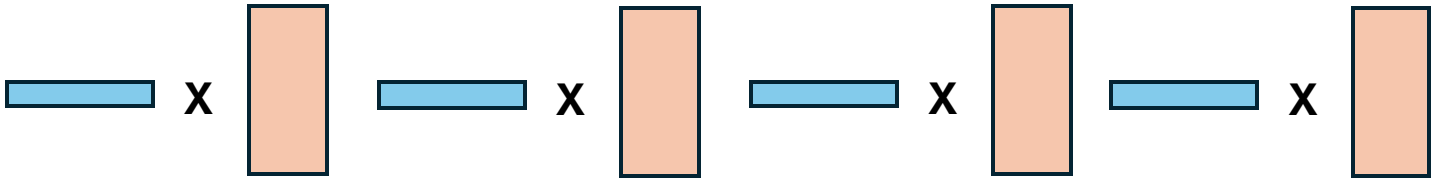




Decoding Stage
(Generate new tokens)



Generate output tokens

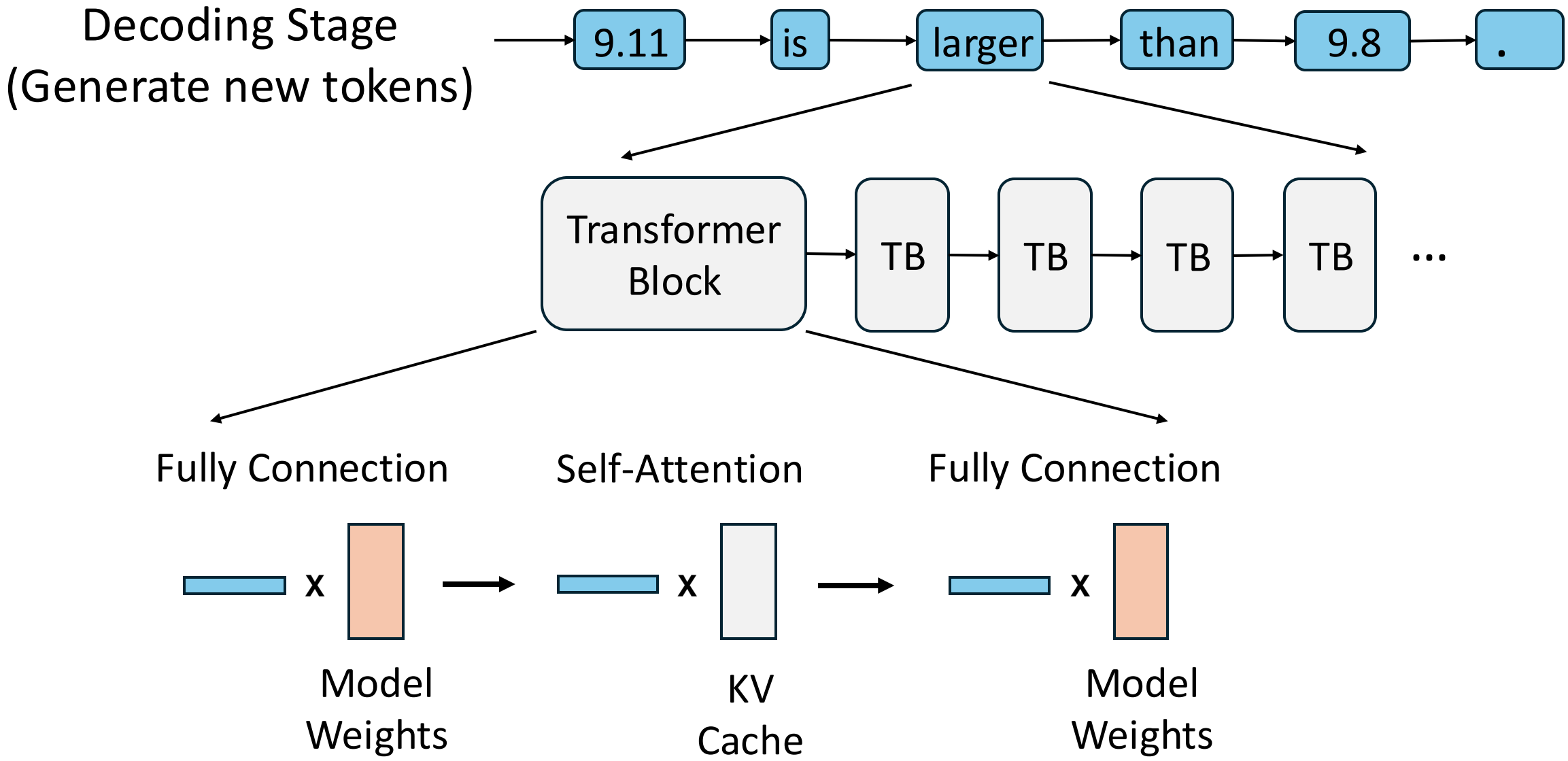


LLM Inference Latency Is Dominated by Decoding Phase

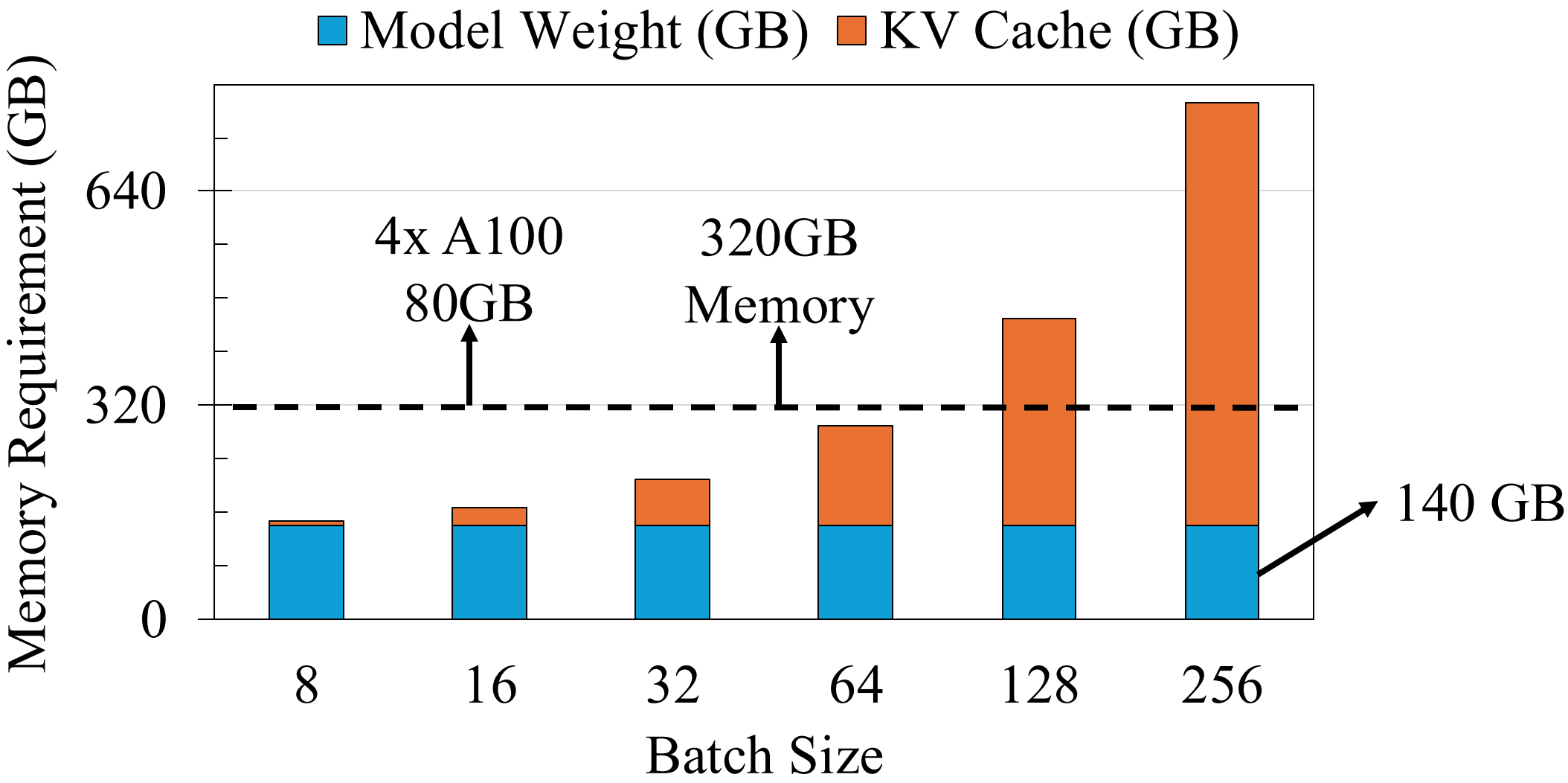
Decoding / E2E Latency Ratio		Prefill								
		128	256	512	1024	2048	4096	8192	16384	32768
Decoding	128	18.1%	48.3%	33.7%	29.0%	21.9%	18.8%	17.1%	16.5%	15.2%
	256	78.8%	38.6%	50.6%	45.2%	36.8%	31.4%	29.4%	28.2%	26.4%
	512	88.4%	80.0%	58.9%	62.6%	53.2%	47.9%	45.2%	45.3%	41.6%
	1024	94.5%	90.1%	83.3%	69.0%	65.2%	60.0%	62.6%	53.7%	50.1%
	2048	97.5%	95.6%	92.1%	87.6%	81.6%	76.2%	73.9%	76.1%	74.7%
	4096	99.2%	98.4%	97.2%	95.0%	92.7%	88.9%	85.3%	83.5%	85.6%
	8192	99.7%	99.5%	99.1%	98.4%	97.2%	96.0%	93.9%	91.3%	89.6%
	16384	99.9%	99.9%	99.8%	99.6%	99.2%	98.6%	97.7%	96.6%	94.7%
	32768	100%	100%	99.9%	99.9%	99.8%	99.6%	99.4%	98.7%	98.1%

Llama 70B serving with maximum batch size on GPUs using vLLM framework.

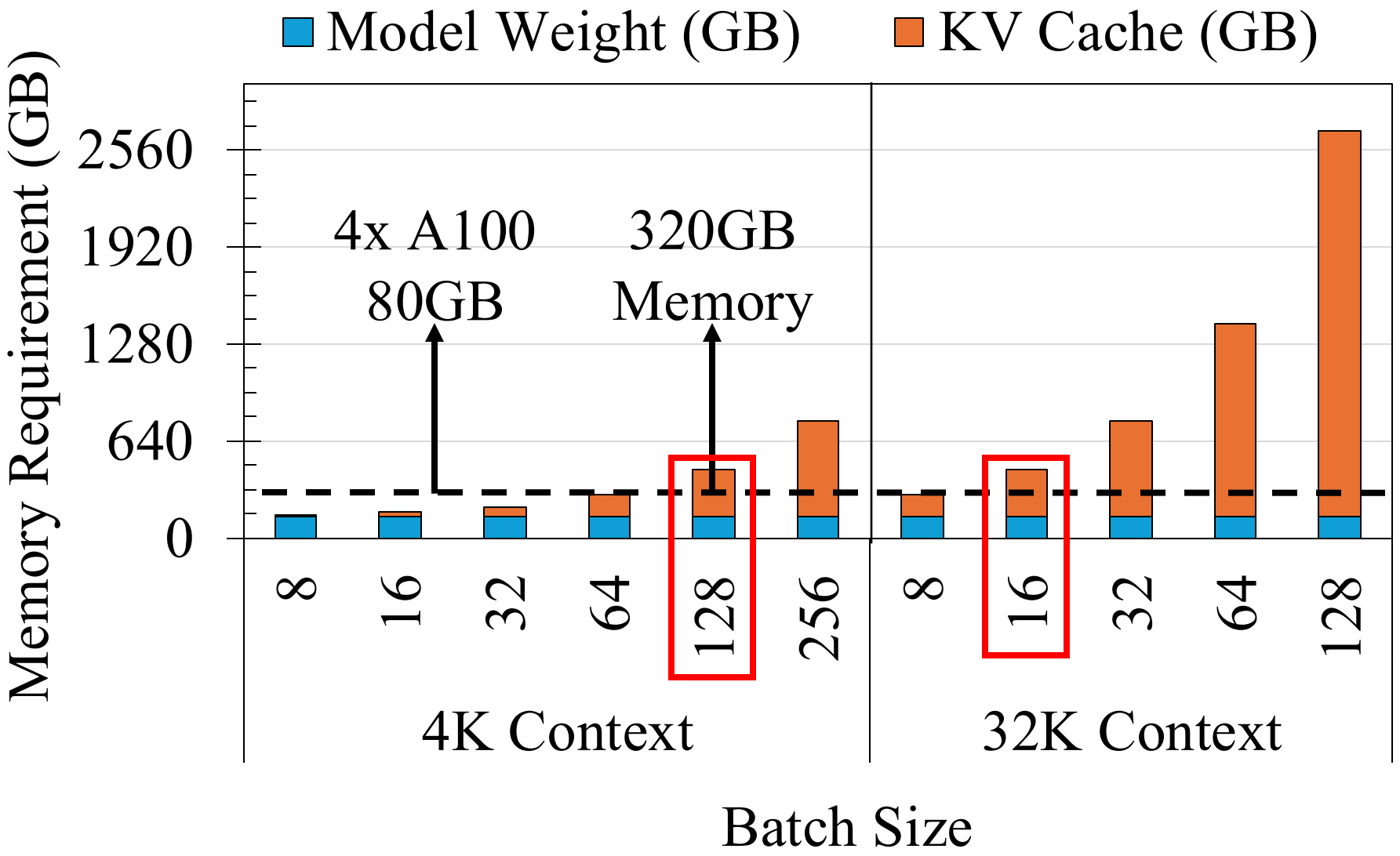
LLM Inference Decoding



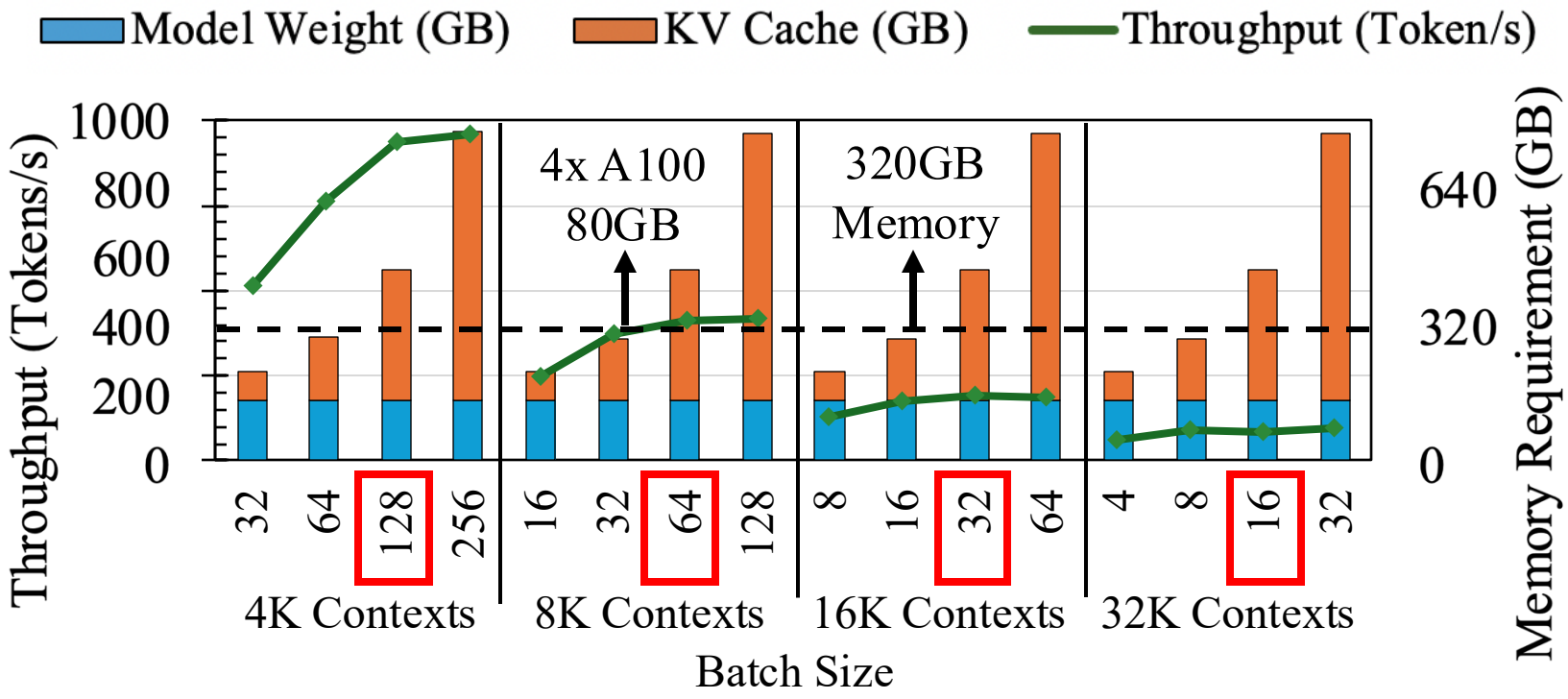
Llama 70B Inference Memory Requirement



GPU Memory Capacity Limits LLM Inference Batch Size

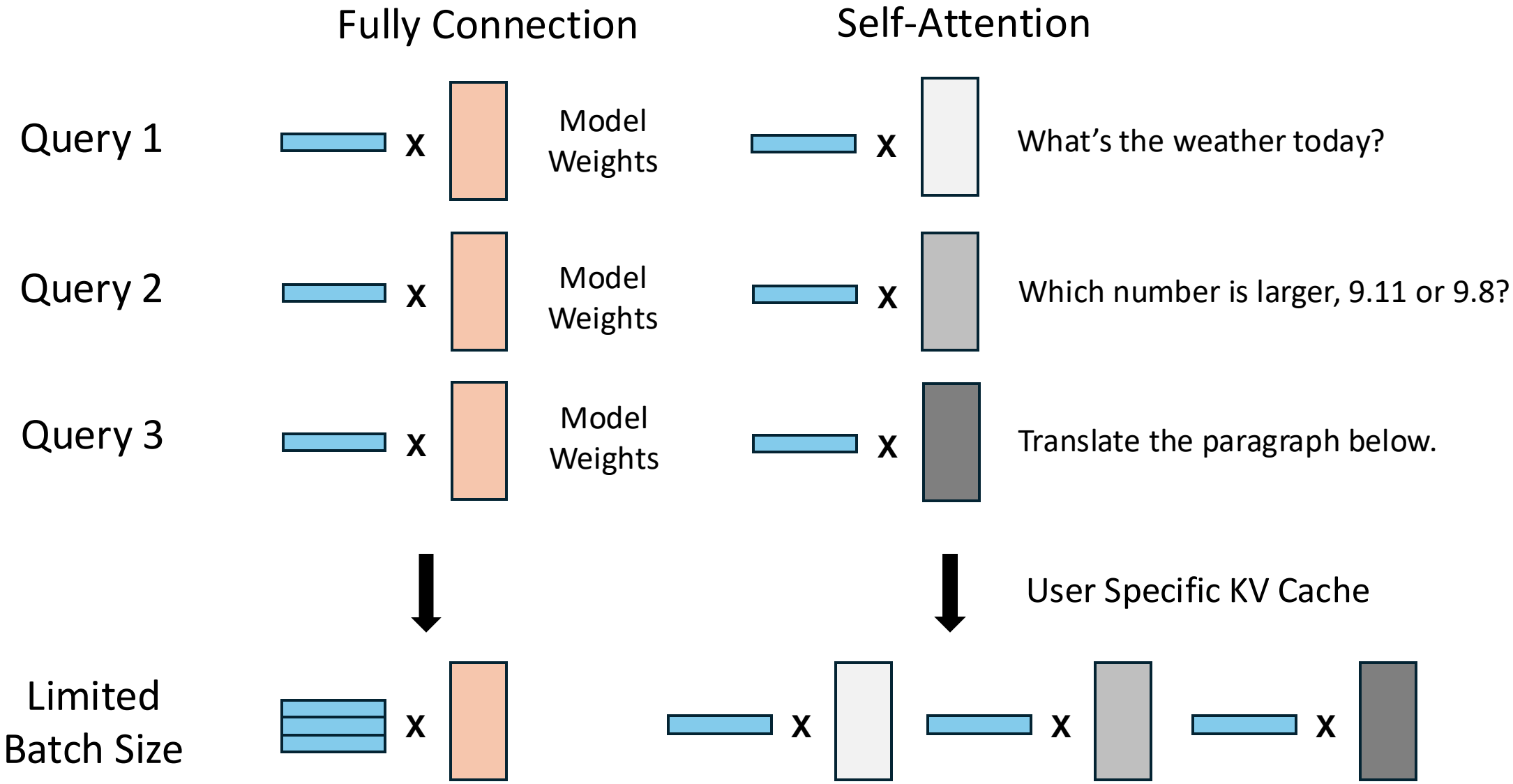


GPU Memory Capacity Limits LLM Inference Batch Size



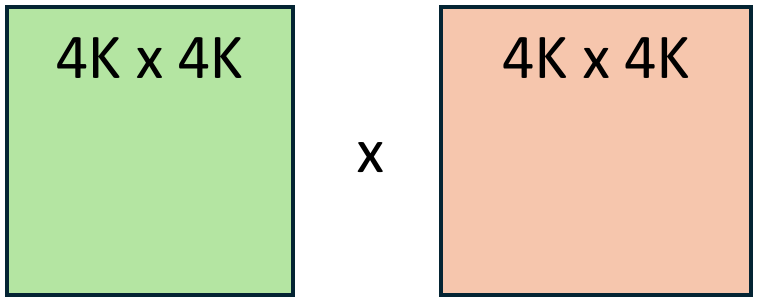
Context Length ↑ **Maximum Batch Size** ↓ **GPU Utilization** ↓

Batch Queries in Decoding



LLM Inference Has Low Operational Intensity

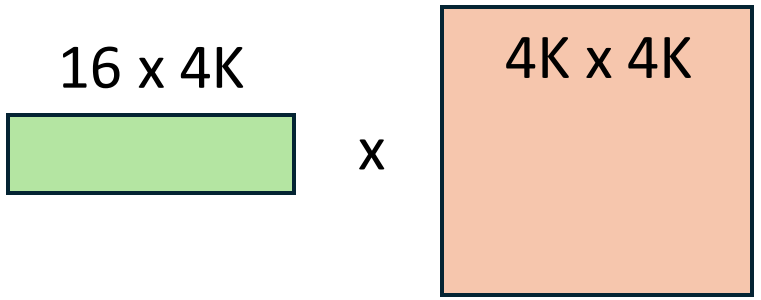
GEMM



Operational Intensity = **2000** Ops/Byte

$$\frac{4K * 4K * 4K * 2 \text{ Operations}}{(4K * 4K + 4K * 4K) * 2 \text{ Byte}}$$

LLM Inference



Operational Intensity = **16** Ops/Byte

$$\frac{16 * 4K * 4K * 2 \text{ Operations}}{(16 * 4K + 4K * 4K) * 2 \text{ Byte}}$$

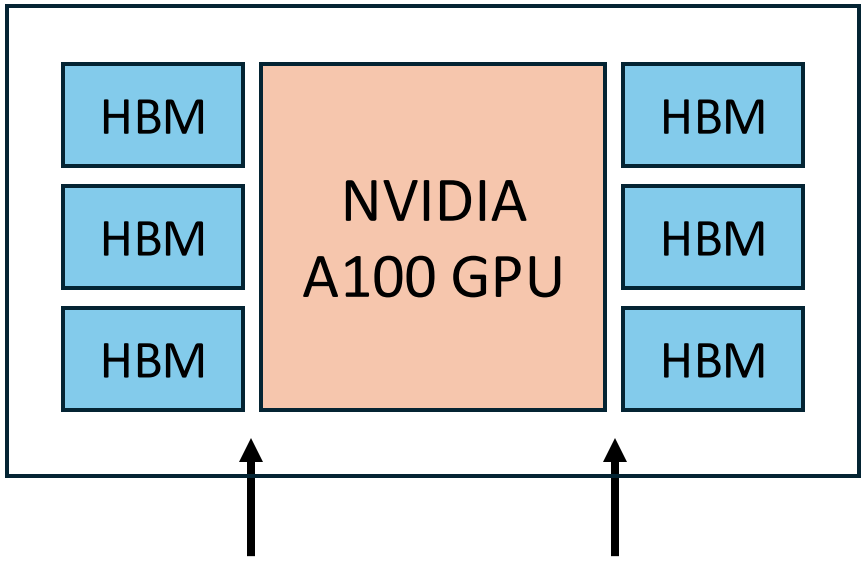
A100 GPU Operational Intensity = $\frac{312 \text{ TFLOPS (BF16)}}{2.0 \text{ TB/s Bandwidth}}$ = **156** Ops/Byte

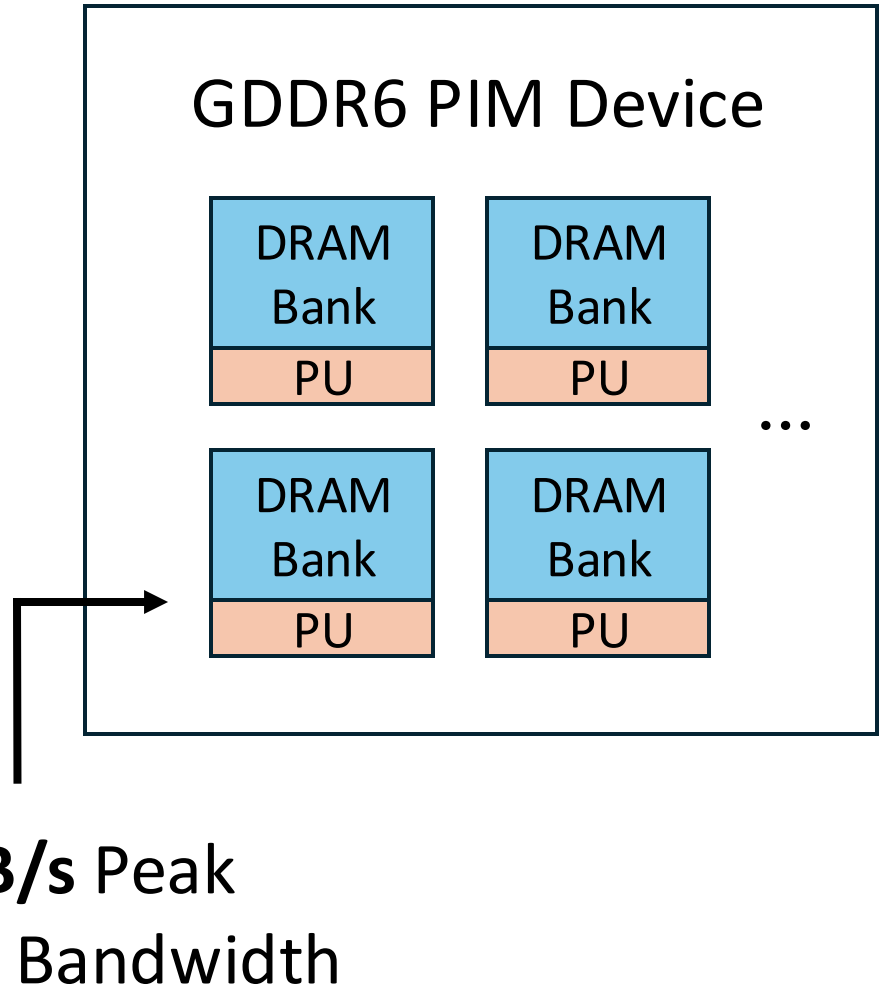
LLM Inference Has Low Operational Intensity

How can we efficiently serve LLM Inference with **Low Operational Intensity**

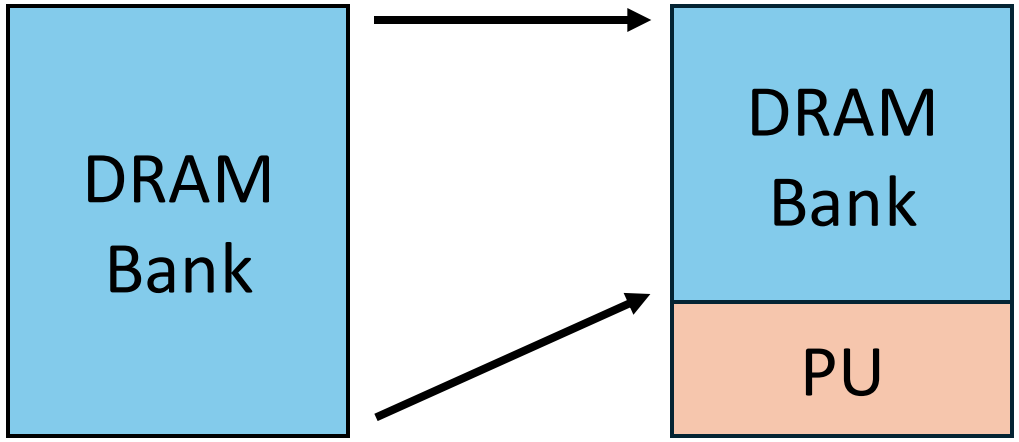
$$\text{Operational Intensity} = \frac{\text{Compute Throughput}}{\text{Memory Bandwidth}}$$

PIM Provides High Memory Bandwidth





Individual PIM Device Has Reduced Memory Density



	UPMEM [1]	AiM [2]	FIMDRAM [3]
Memory Density	25% - 50%	75%	75%

[1] The true processing in memory accelerator. In 2019 IEEE Hot Chips 31 Symposium (HCS), pages 1–24. IEEE Computer Society, 2019
[2] System architecture and software stack for GDDR6-AiM. In 2022 IEEE Hot Chips 34 Symposium (HCS), pages 1–25. IEEE, 2022.
[3] 25.4 a 20nm 6gb function-in-memory DRAM, based on HBM2 with a 1.2 tflops programmable computing unit using bank-level parallelism, for machine learning applications. In 2021 IEEE International Solid-State Circuits Conference (ISSCC), volume 64, pages 350–352. IEEE, 2021.

LLM Inference Requires Large Memory Capacity

Llama 70B
4K Context
Batch = 128

450 GB

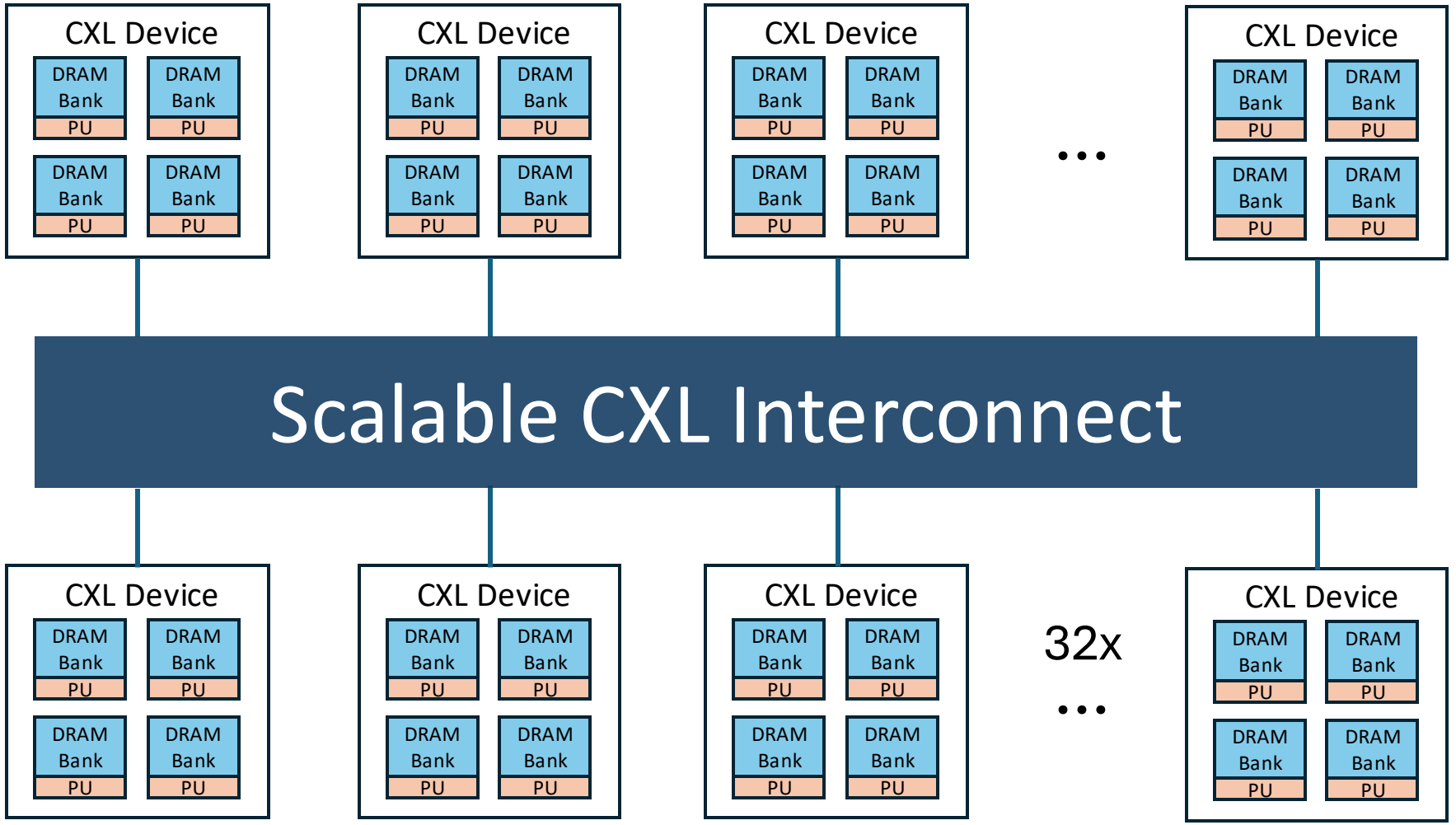
140 GB
Model Weights

310 GB
KV Cache

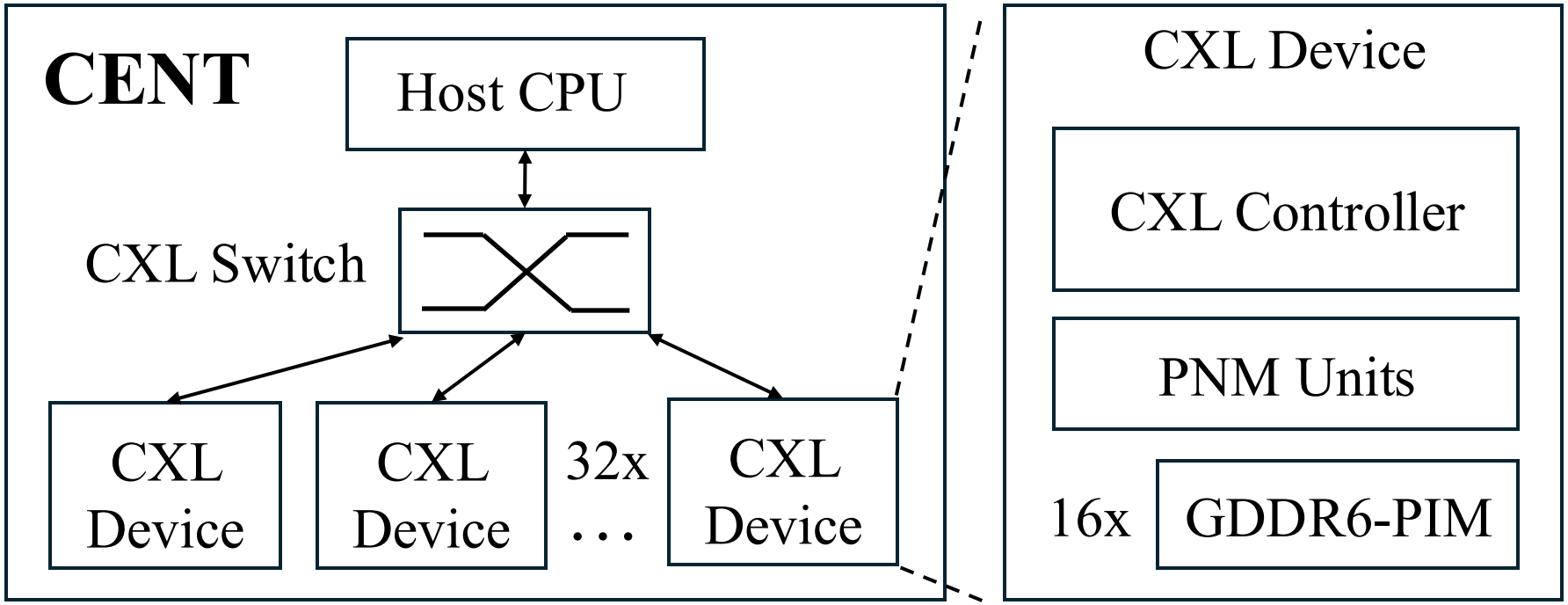
	Llama2	Llama3	DeepSeek R1	OpenAI o1	Gemini 1.5 Pro	Grok-3	Claude 3.5
Model	70 B	405 B	671 B	-	-	-	-
Contexts	4K	128 K	128 K	200 K	2 M	1 M	200 K

https://platform.openai.com/docs/models#o1, https://developers.googleblog.com/en/new-features-for-the-gemini-api-and-google-ai-studio/, https://x.ai/blog/grok-3, https://docs.anthropic.com/en/docs/about-claude/models, The Llama 3 Herd of Models, DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning

CXL Interconnect Provides Substantial Memory Capacity



CENT Architecture



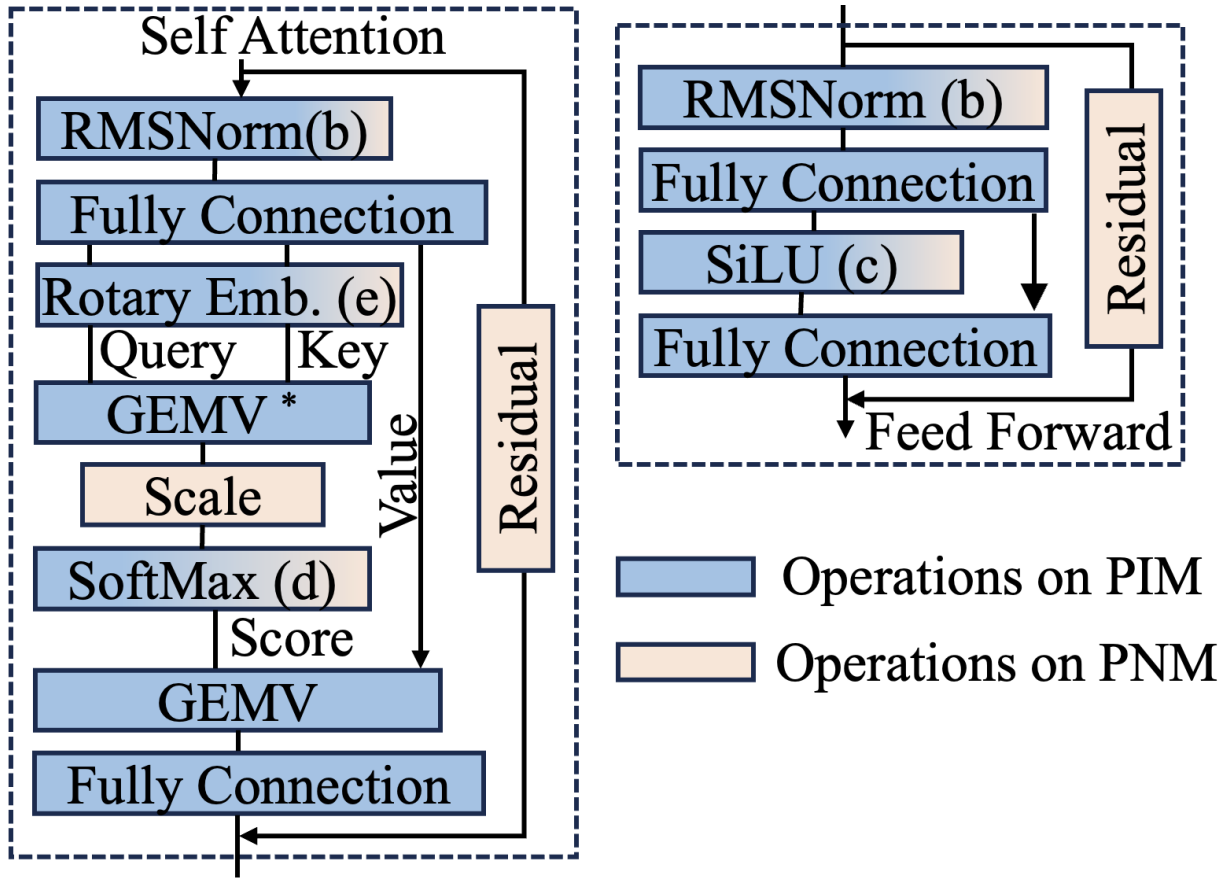
Full Transformer Block Execution on the CXL Device

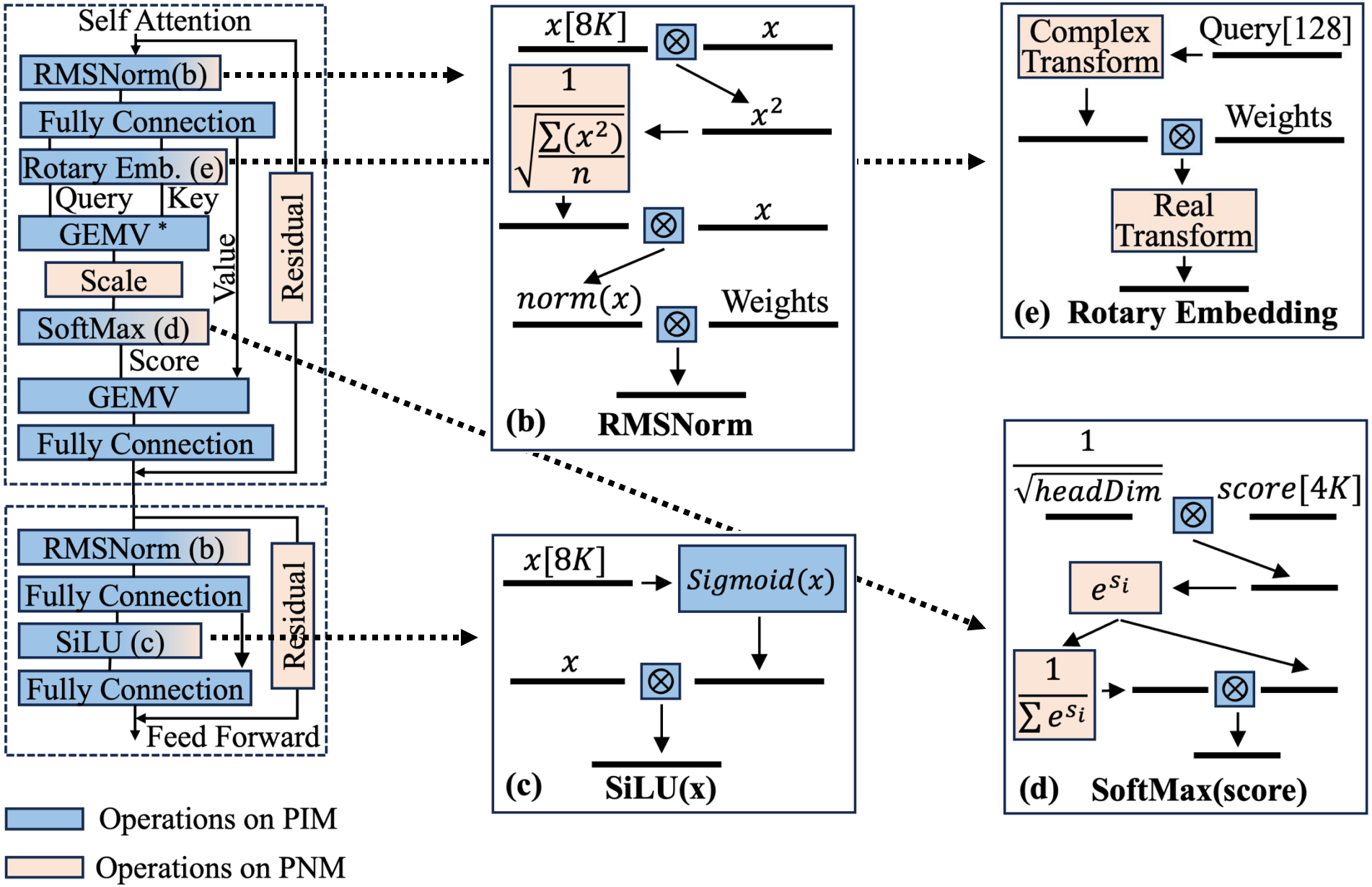
PIM Module **99%**

- Vector Multiplication
- Element-wise Multiplication

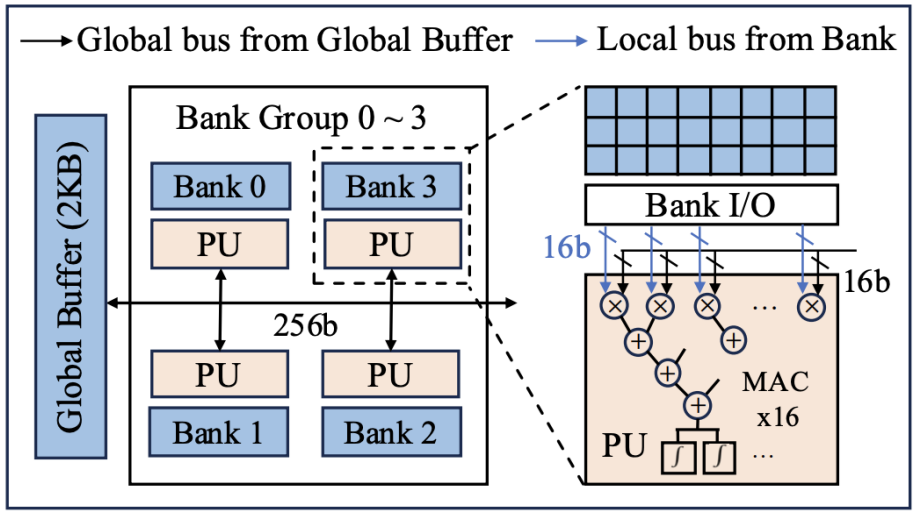
PNM Accelerator

- Vector Addition Units
- Reduction Trees
- Exponent Processors
- RISC-V cores for Special Functions

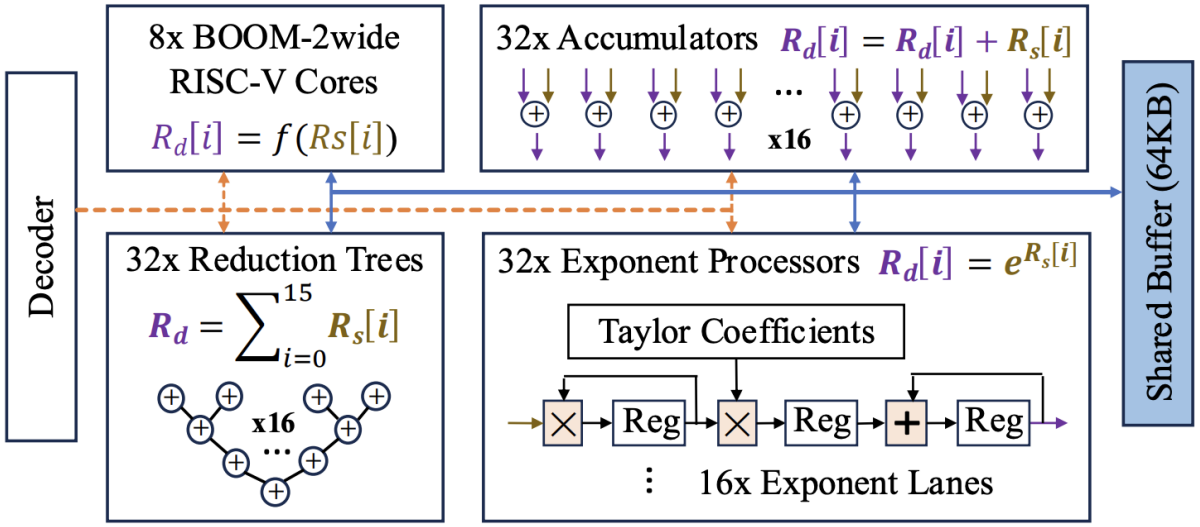



Self Attention
RMSNorm(b)
Fully Connection
Rotary Emb. (e)
Query
Key
GEMV *
Scale
SoftMax (d)
Score
GEMV
Fully Connection
Value
Residual
RMSNorm (b)
Fully Connection
SiLU (c)
Fully Connection
Residual
Feed Forward
x[8K]
x
1
√(∑(x²)/n)
x²
x
norm(x)
Weights
(b) RMSNorm
Complex Transform
Query[128]
Weights
Real Transform
(e) Rotary Embedding
x[8K]
Sigmoid(x)
x
(c) SiLU(x)
1
√headDim
score[4K]
e^si
1
∑e^si
(d) SoftMax(score)
Operations on PIM
Operations on PNM

Hierarchical PIM-PNM Design



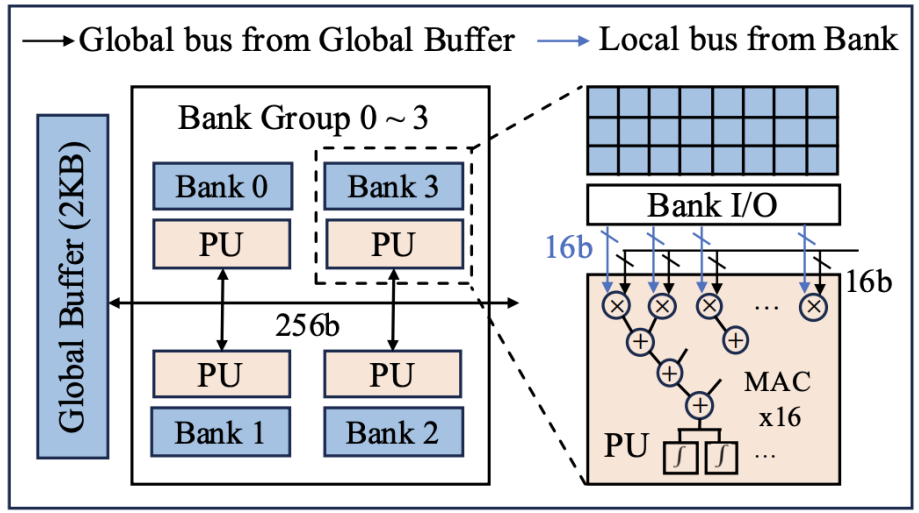
(a) GDDR6-PIM Channel



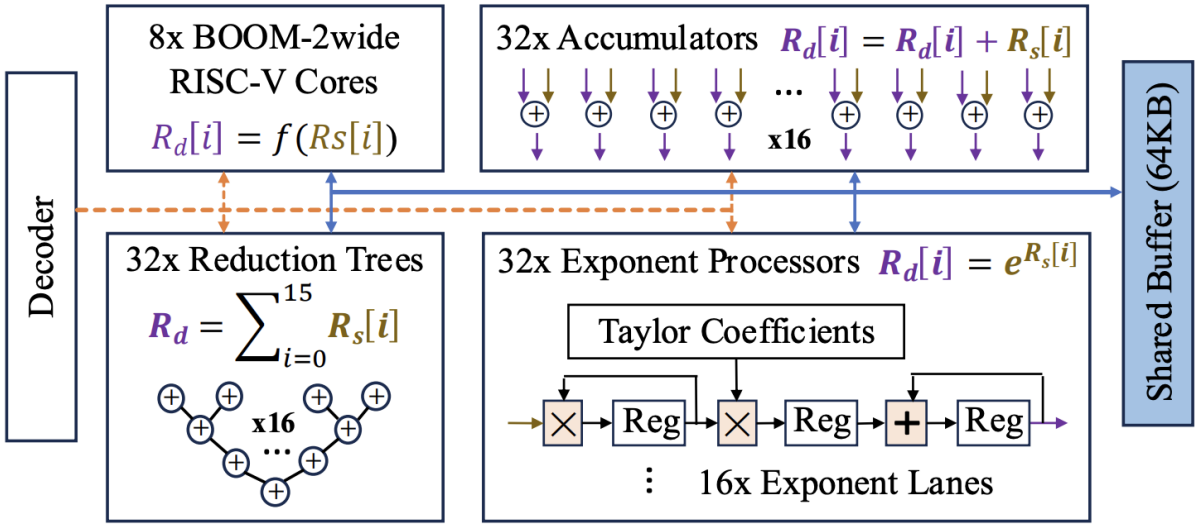
(b) PNM Units

- PNM Units use 16x parallelism, aggregating to 256-bit data access, matching the Shared Buffer data width.
- PNM Units have 32 lanes, matching with 32 channels, 2 channels per GDDR6-PIM module.

Hierarchical PIM-PNM Design



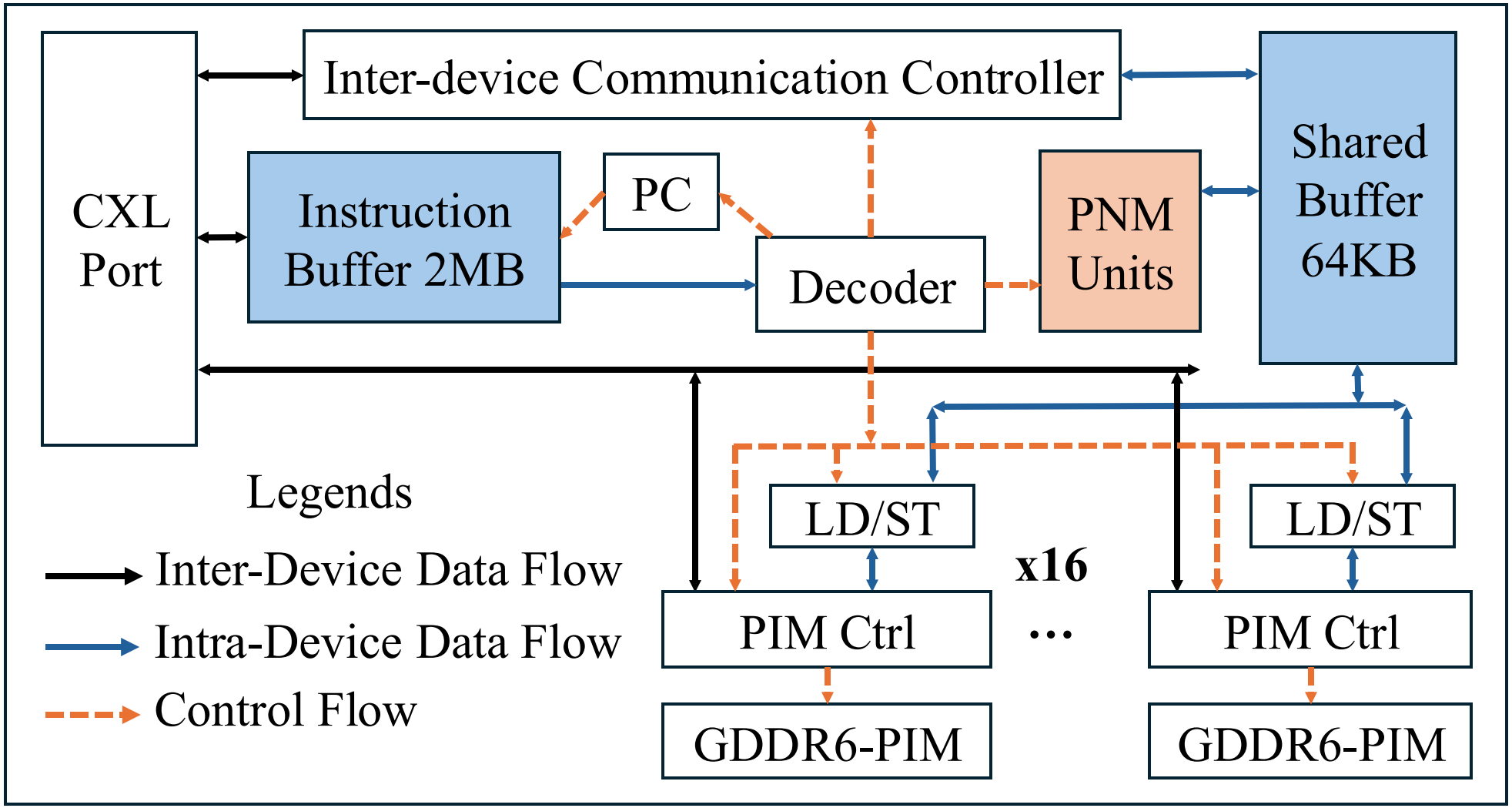
(a) GDDR6-PIM Channel



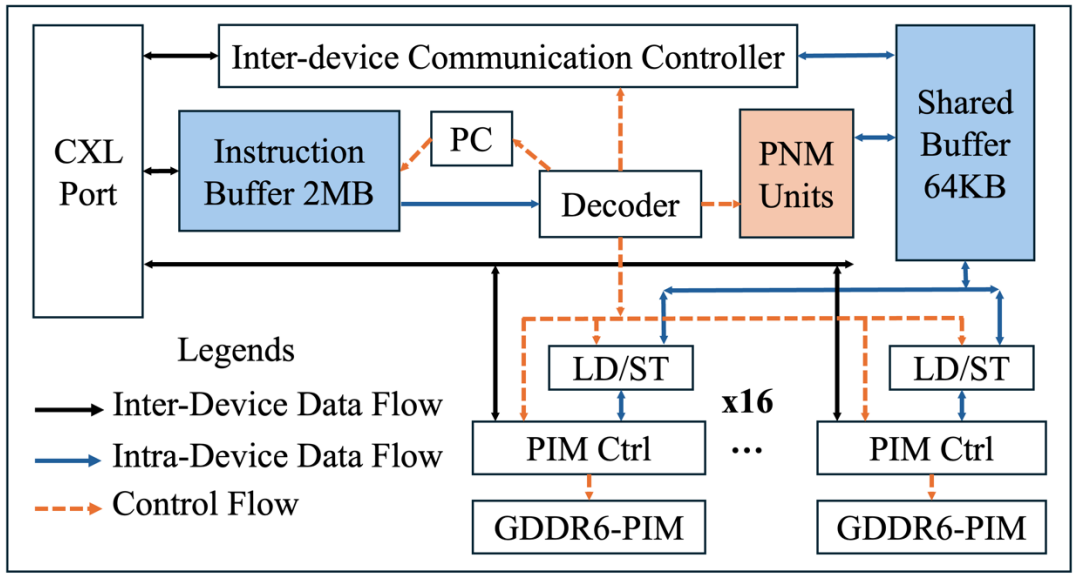
(b) PNM Units

Instruction	Assembly
Near-Bank PUs	
MAC All Bank	`MAC_ABK CHmask OPsize RO CO Regid`
Element-wise Mult.	`EW_MUL CHmask OPsize RO CO`
Activation Function	`AF CHmask AFid Regid`
PNM Units	
Exponent	`EXP OPsize Rd Rs`
Reduction	`RED OPsize Rd Rs`
Accumulation	`ACC OPsize Rd Rs`
RISCV operation	`RISCV OPsize PC Rd Rs`

CXL Device Architecture

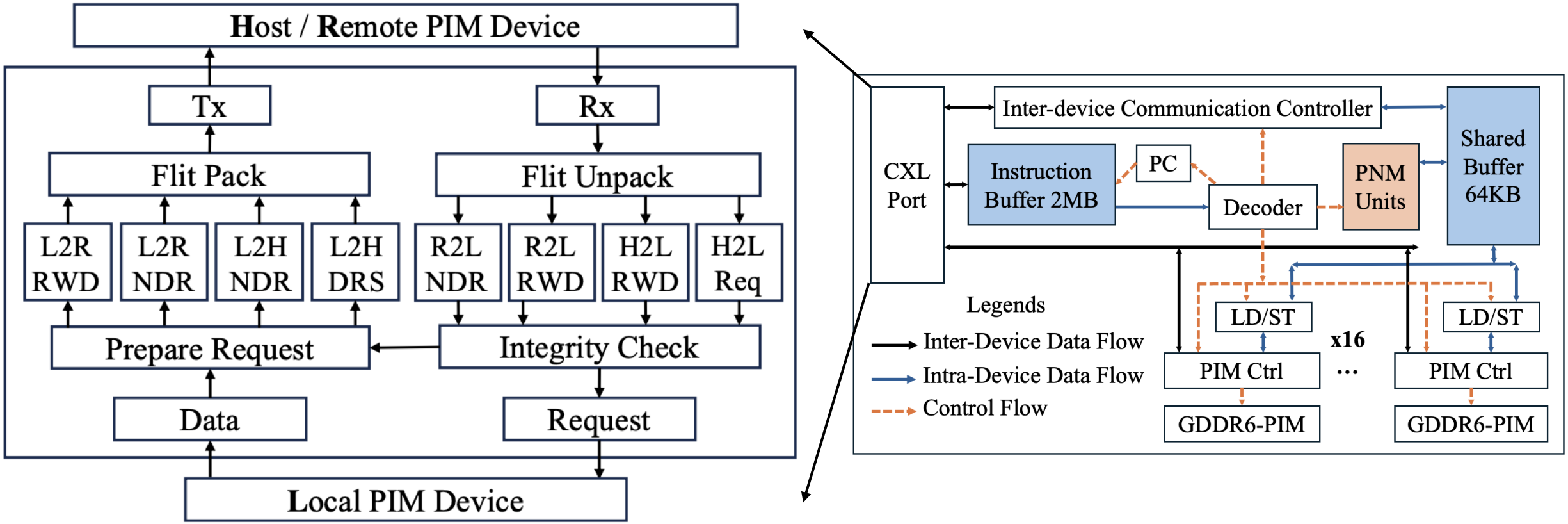


CENT Data Movement Instructions

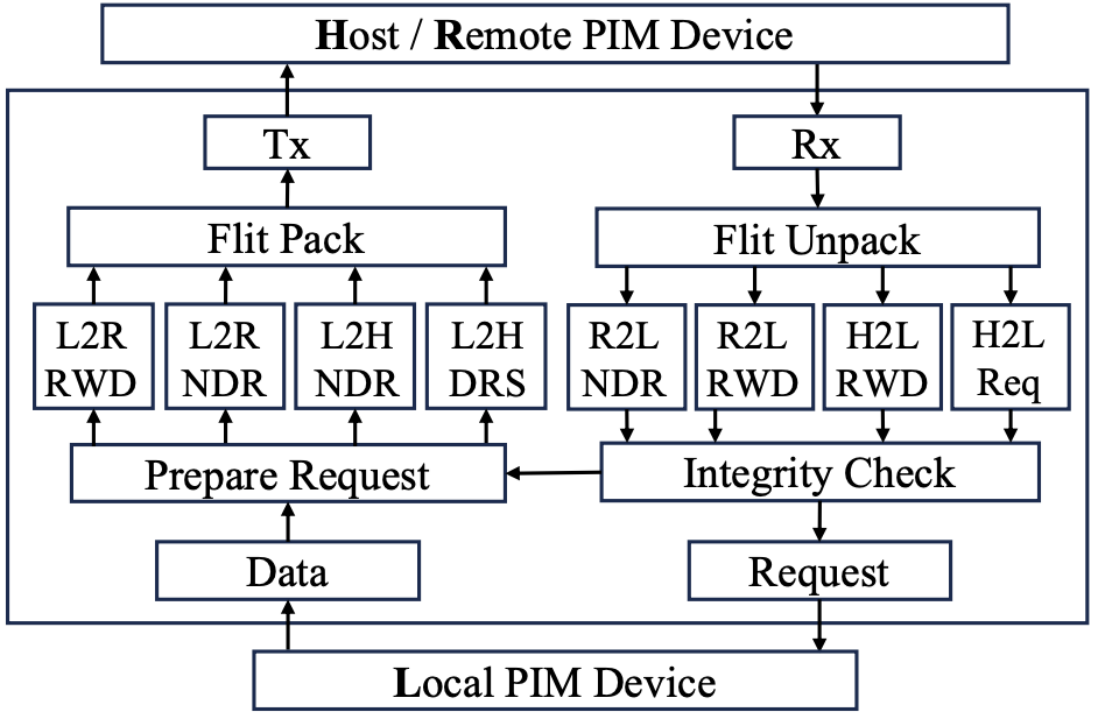


Instruction	Assembly
CXL Device ↔ CXL Device	
Send	`SEND_CXL DVid Rs Rd`
Receive	`RECV_CXL`
Broadcast	`BCAST_CXL DVcount Rs Rd`
Shared Buffer ↔ DRAM Banks	
Write Single Bank	`WR_SBK CHid OPsize BK RO CO Rs`
Read Single Bank	`RD_SBK CHid OPsize BK RO CO Rd`
Write All Banks	`WR_ABK CHid RO CO Rs Regid`
Global Buffer ↔ DRAM Banks	
Copy Bank → Global Buffer	`COPY_BKGB CHmask OPsize RO CO`
Copy Global Buffer → Bank	`COPY_GBBK CHmask OPsize RO CO`
Shared Buffer ↔ PUs	
Write bias	`WR_BIAS CHmask Rs`
Read MAC register	`RD_MAC CHmask Rd Regid`
Shared Buffer → Global Buffer	
Write Global Buffer	`WR_GB CHmask OPsize CO Rs`

CXL Port Architecture

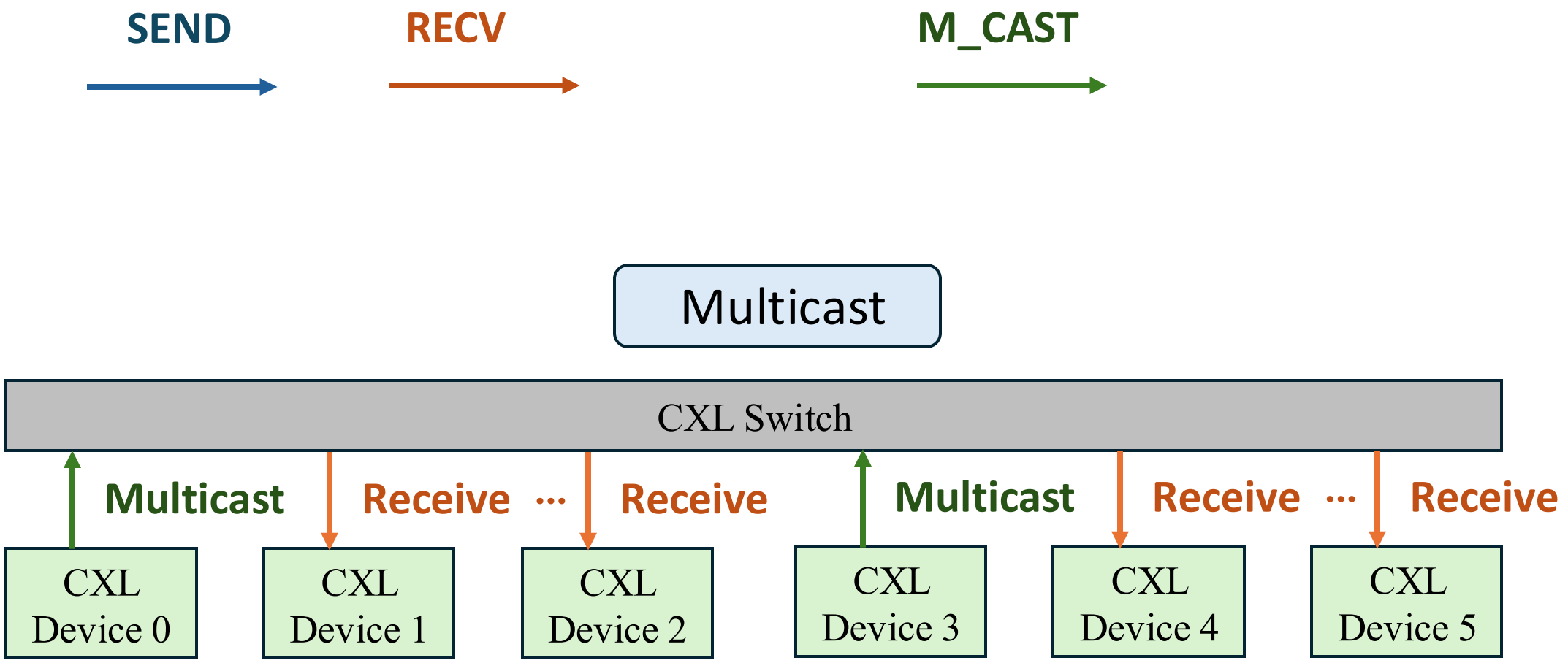


CXL Port Architecture

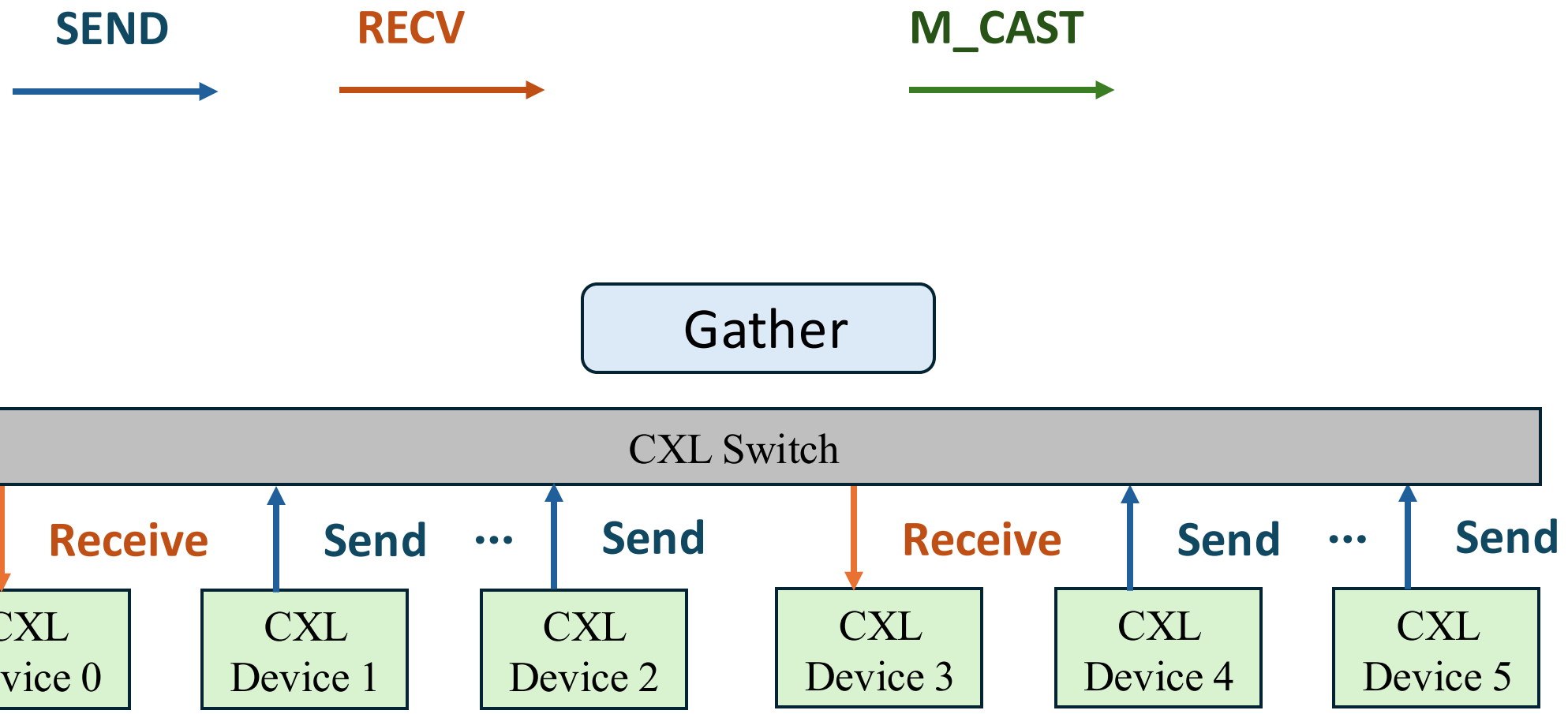


- **H**ost read data from **L**ocal PIM Device
  - H2L Request
  - L2H Data Response (DRS)
- **H**ost write data to **L**ocal PIM Device
  - H2L Request with Data (RWD)
  - L2H No Data Response (NDR)
- **L**ocal PIM Device send data to **R**emote PIM Device
  - L2R Request with Data (RWD)
  - R2L No Data Response (NDR)
- **R**emote PIM Device write data to **L**ocal PIM Device
  - R2L Request with Data (RWD)
  - L2R No Data Response (NDR)

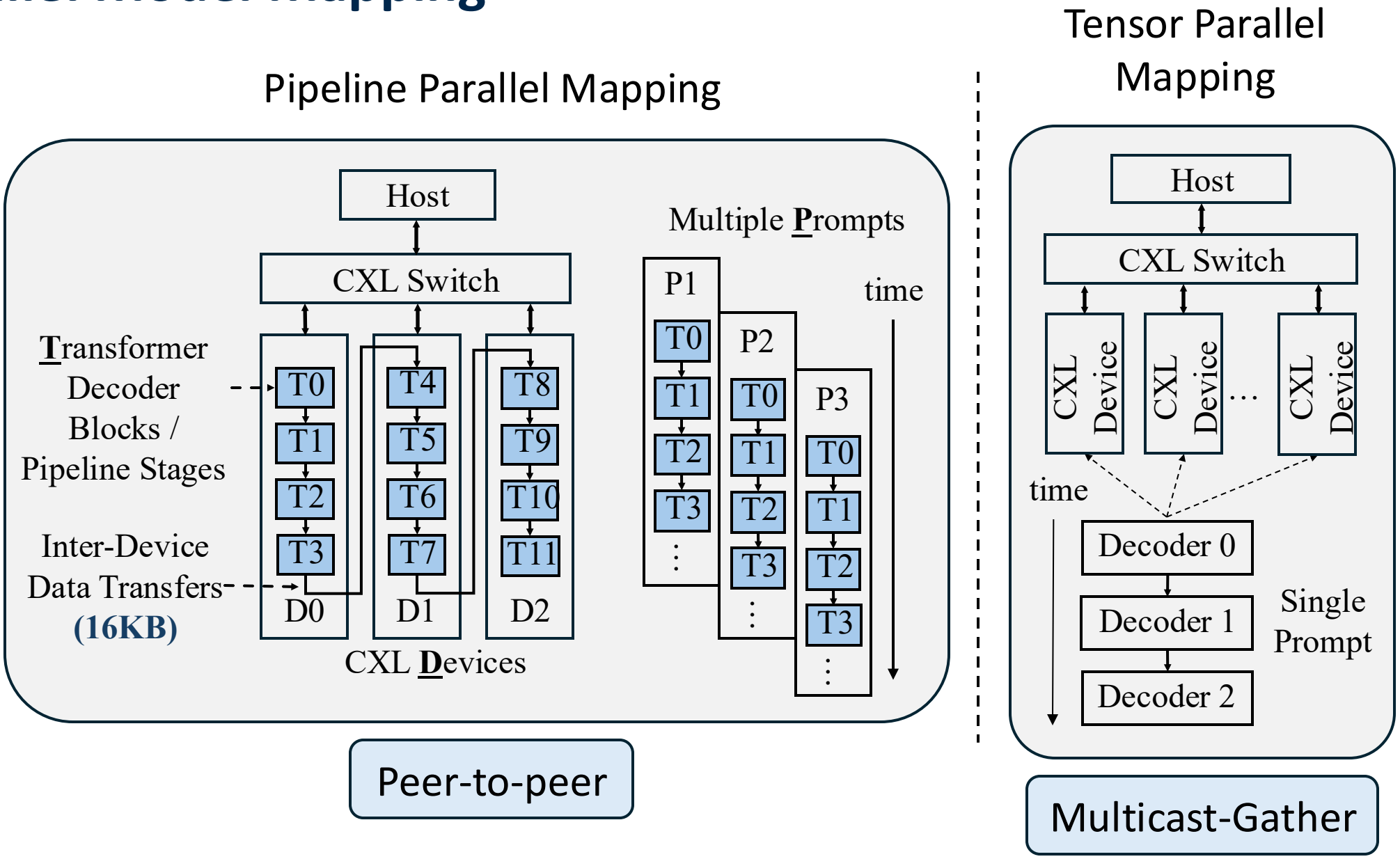
Inter-Device CXL Communication



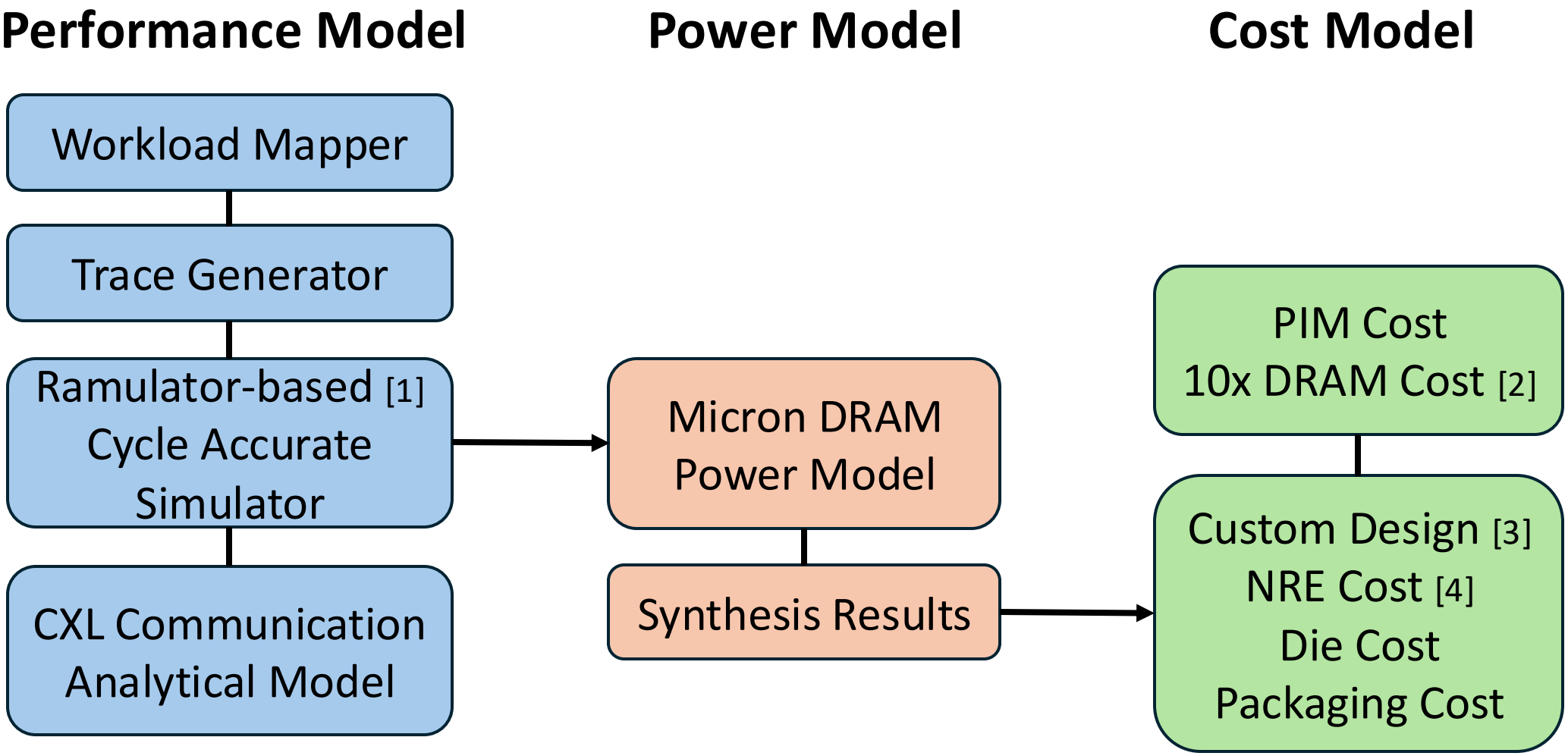
Inter-Device CXL Communication



Parallel Model Mapping

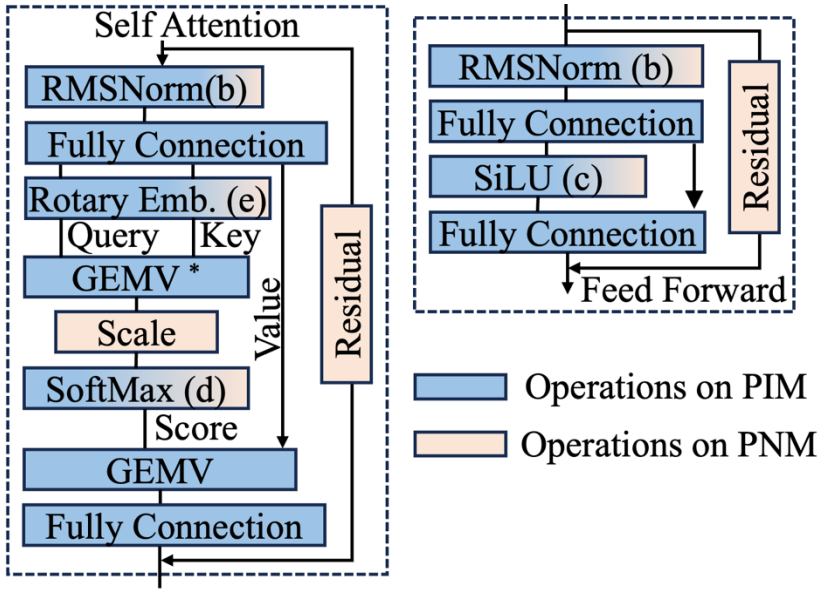


Evaluation Methodology



[1] Ramulator 2.0: A Modern, Modular, and Extensible DRAM Simulator.
[2] UPMEM. Accelerating compute by cramming it into dram memory
[3] Supply chain aware computer architecture. In Proceedings of the 50th Annual International Symposium on Computer Architecture, pages 1–15, 2023.
[4] Moonwalk: Nre optimization in ASIC clouds. ACM SIGARCH Computer Architecture News, 45(1):511–526, 2017

Performance Model



Workload Mapper

```
1.  Vector = {shared_buffer_addr, vector_dim}
2.  Matrix = {num_row, row_addr, matrix_dim}
3.  def GEMV(Vector, Matrix, Hardware_config) {
4.      for each channel_index in Hardware_config->num_channels:
5.          WR_GB (Vector->shared_buffer_addr)
6.          for each row_index in Matrix->num_rows:
7.              WR_BIAS (channel_index)
8.              MAC_ABK (channel_index, row_addr + row_index)
9.              RD_MAC (channel_index)
10. }
```

Trace Generator

Instruction	Assembly
Near-Bank PUs	
MAC All Bank	`MAC_ABK CHmask OPsize RO CO Regid`
Element-wise Mult.	`EW_MUL CHmask OPsize RO CO`
Activation Function	`AF CHmask AFid Regid`

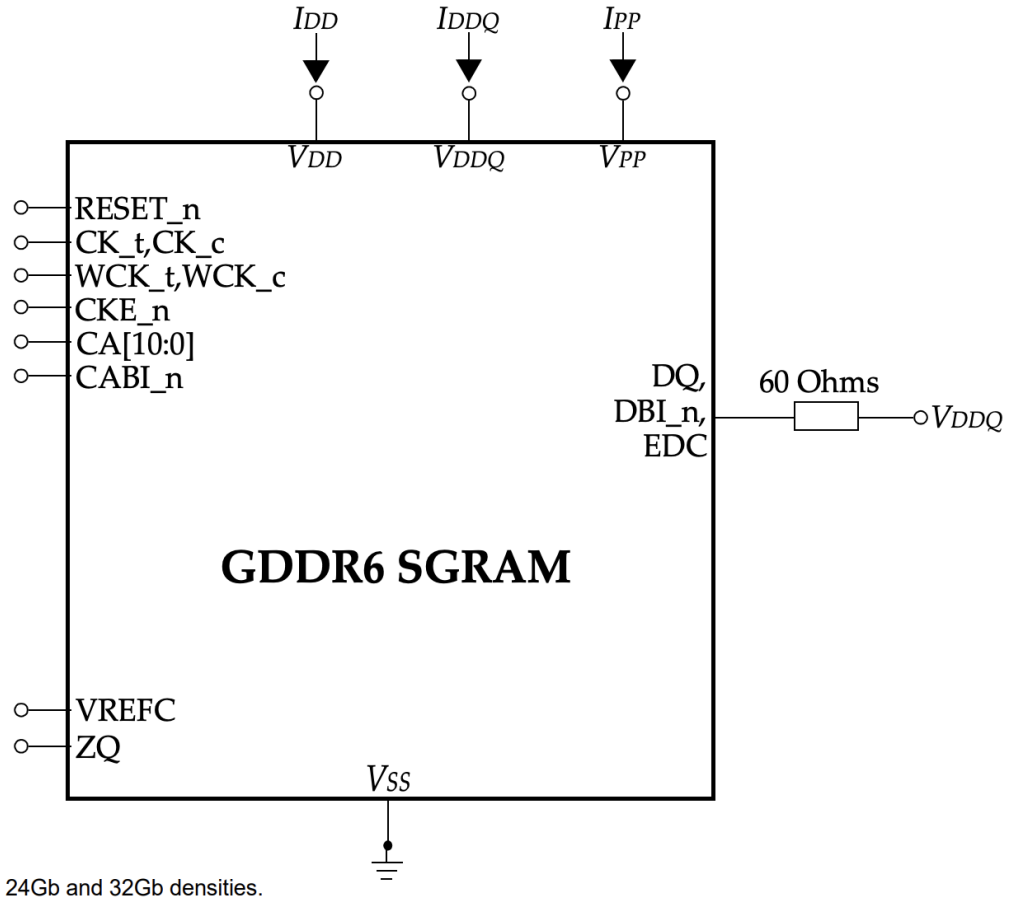
AiM Simulator

- Number of CXL Flits
- Round-Trip Latency [1]
- PCIe 6.0 Bandwidth

CXL Analytical Model

[1] Pond: CXL-based memory pooling systems for cloud platforms.

Micron DRAM Power Model



24Gb and 32Gb densities.

$$P(Act_{BG}) = \sum_{n=1}^{tRAS_{new}} I_{DD3N} \times V_{DD}/tRC_{new}$$

$$P(Pre_{BG}) = \sum_{n=1}^{tRP_{new}} I_{DD2N} \times V_{DD}/tRC_{new}$$

$$P(ACT) = \sum_{n=1}^{tRAS} (I_{DD0} - I_{DD3N}) \times V_{DD}/tRC_{new}$$

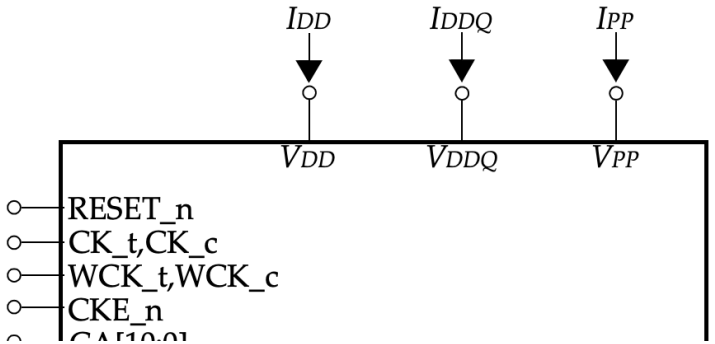
$$P(PRE) = \sum_{n=tRAS+1}^{tRC} (I_{DD0} - I_{DD2N}) \times V_{DD}/tRC_{new}$$

$$P(RD) = \sum_{n=1}^{tR} (I_{DD4R} - I_{DD3N}) \times V_{DD}/tRC_{new}$$

$$P(WR) = \sum_{n=1}^{tW} (I_{DD4W} - I_{DD3N}) \times V_{DD}/tRC_{new}$$

$$P(REF) = \sum_{n=1}^{tRP} \Big((I_{DD2N} \times V_{DD}) + (N(PRE) \times P(PRE)) \Big)/tREF + \sum_{n=1}^{tRFC} I_{DD5} \times V_{DD}/tREF \quad (7)$$

Micron DRAM Power Model



$$P(Act_{BG}) = \sum_{n=1}^{tRAS_{new}} I_{DD3N} \times V_{DD}/tRC_{new}$$

$$P(Pre_{BG}) = \sum_{n=1}^{tRP_{new}} I_{DD2N} \times V_{DD}/tRC_{new}$$

$$P(ACT) = \sum^{tRAS} (I_{DD0} - I_{DD3N}) \times V_{DD}/tRC_{new}$$

RESEARCH-ARTICLE | PUBLIC ACCESS

What Your DRAM Power Models Are Not Telling You: Lessons from a Detailed Experimental Study

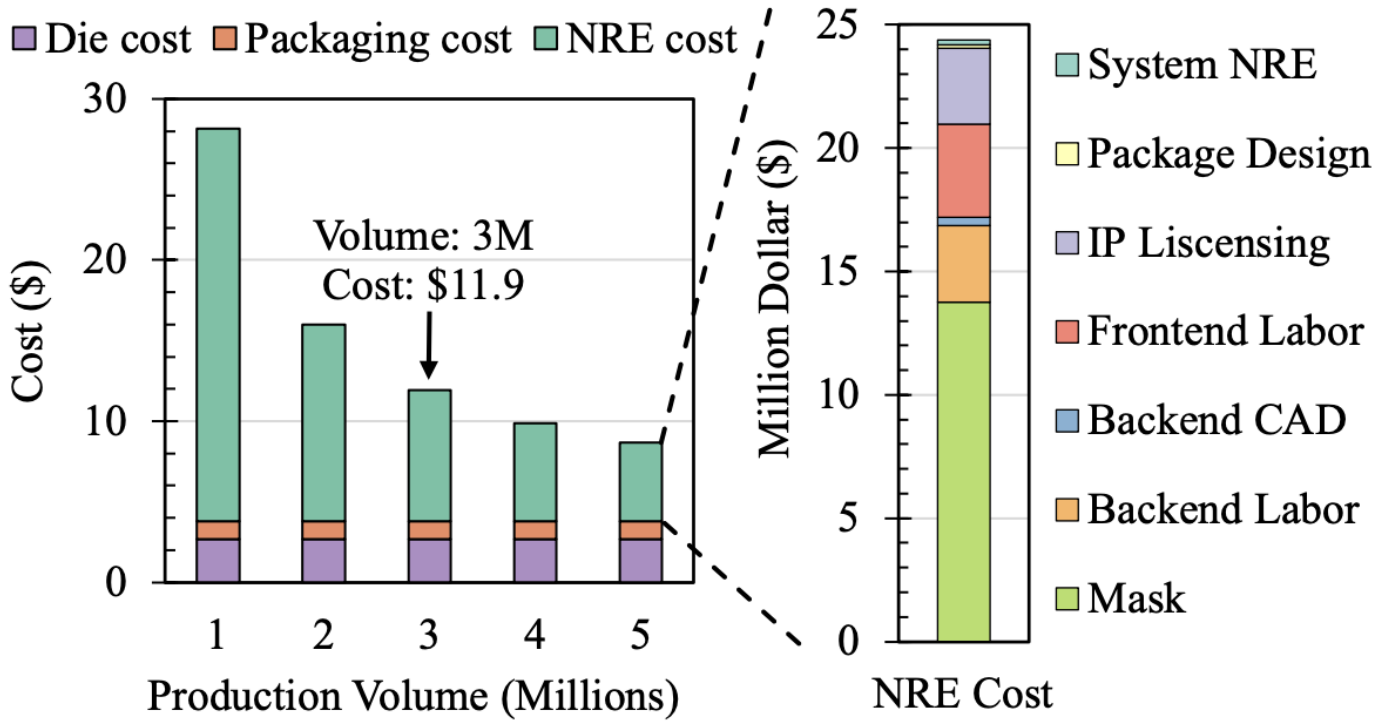
Authors: Saugata Ghose, Abdullah Giray Yaglikçi, Raghav Gupta, Donghyuk Lee, Kais Kudrolli, William X. Liu, Hasan Hassan, Kevin K. Chang, Niladrish Chatterjee, Aditya Agrawal, Mike O'Connor, Onur Mutlu | Authors Info & Claims

Proceedings of the ACM on Measurement and Analysis of Computing Systems, Volume 2, Issue 3 • Article No.: 38, Pages 1 - 41
https://doi.org/10.1145/3224419

$$+\sum_{n=1}^{tRFC} I_{DD5} \times V_{DD}/tREF \quad (7)$$

Cost Model

- GPU: $10,000 per A100, half of available prices
- PIM: 10x standard PIM modules
- Custom Design Cost: NRE, Die and Packaging costs, considering Production Volume



CENT versus GPU Baseline

System	CENT	GPU
Hardware	32 CXL Devices	4 NVIDIA A100
Memory	512 GB, GDDR6	320 GB, HBM2E

CENT versus GPU Baseline

System	CENT	GPU
Hardware	32 CXL Devices	4 NVIDIA A100
Memory	512 GB, GDDR6	320 GB, HBM2E
Compute Throughput	608 TFLOPS	1248 TFLOPS

CENT versus GPU Baseline

System	CENT	GPU
Hardware	32 CXL Devices	4 NVIDIA A100
Memory	512 GB, GDDR6	320 GB, HBM2E
Compute Throughput	608 TFLOPS	1248 TFLOPS
Peak Bandwidth	**512 TB/s**	**8 TB/s**

CENT versus GPU Baseline

System	CENT	GPU
Hardware	32 CXL Devices	4 NVIDIA A100
Memory	512 GB, GDDR6	320 GB, HBM2E
Compute Throughput	608 TFLOPS	1248 TFLOPS
Peak Bandwidth	**512 TB/s**	**8 TB/s**
3-Year *Owned* Total Cost of Ownership	0.73 $/hour	1.76 $/hour
3-Year *Rental* Total Cost of Ownership	1.05 $/hour	5.45 $/hour

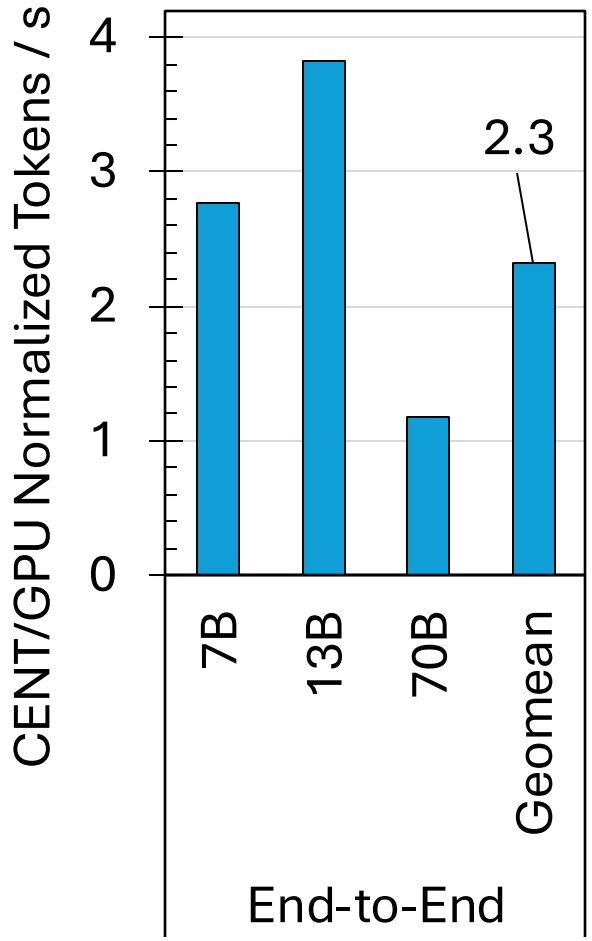
2.4x 

Cost Efficiency Comparison

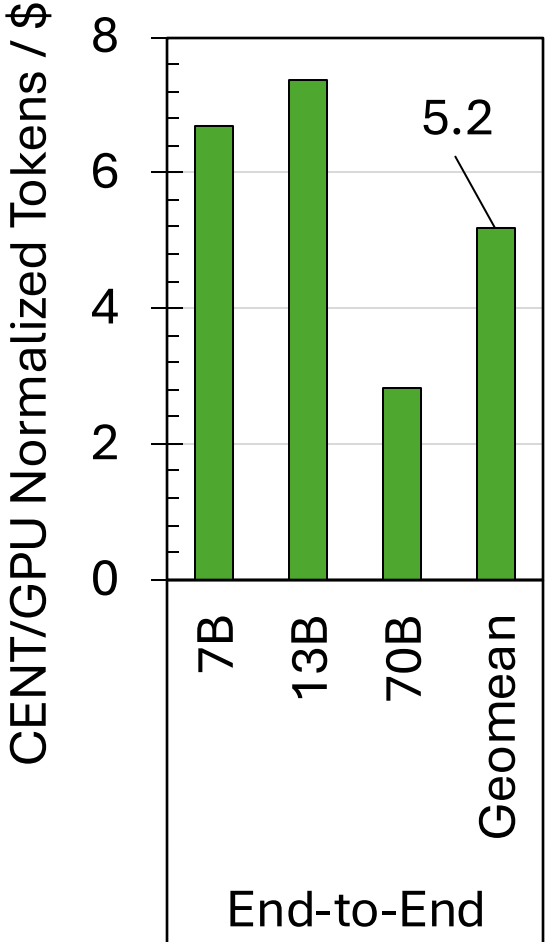
Prefill Tokens = 512
Decoding Tokens = 3.5K

	3-Year Owned TCO
GPU	1.76 $/hour
CENT	0.73 $/hour

2.4x
Hardware Cost

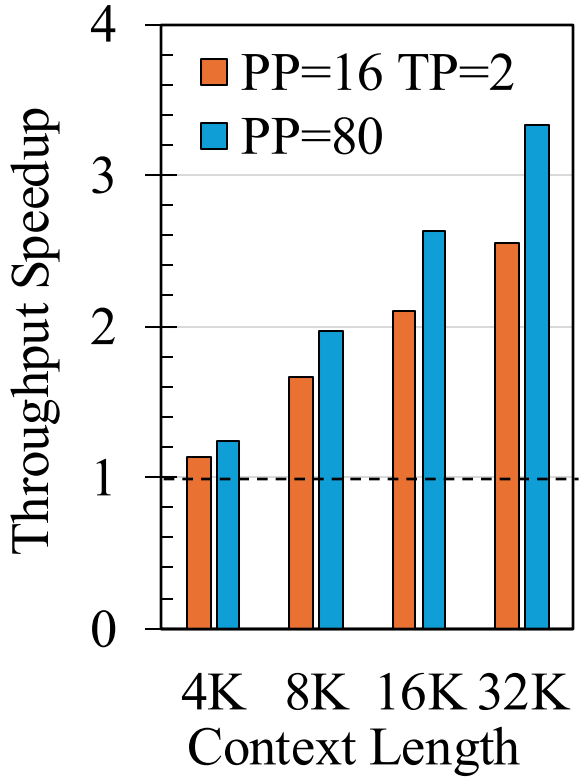


2.3x
Throughput

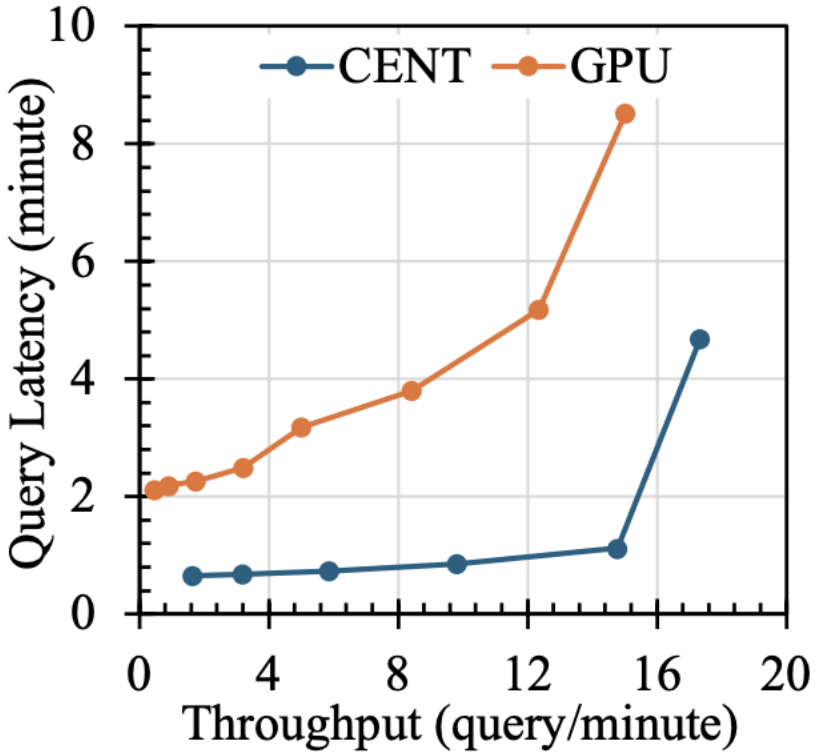


5.2x
Tokens per $

Long Context and QoS Analysis

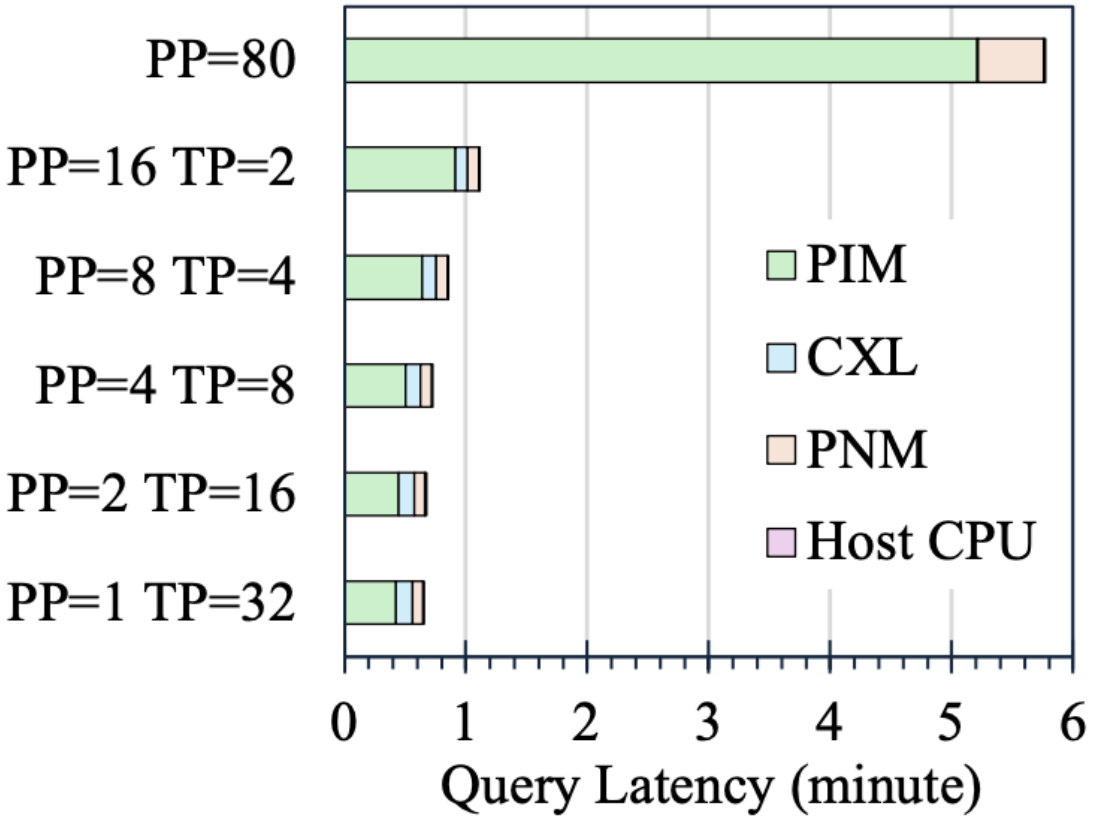


Decoding Throughput

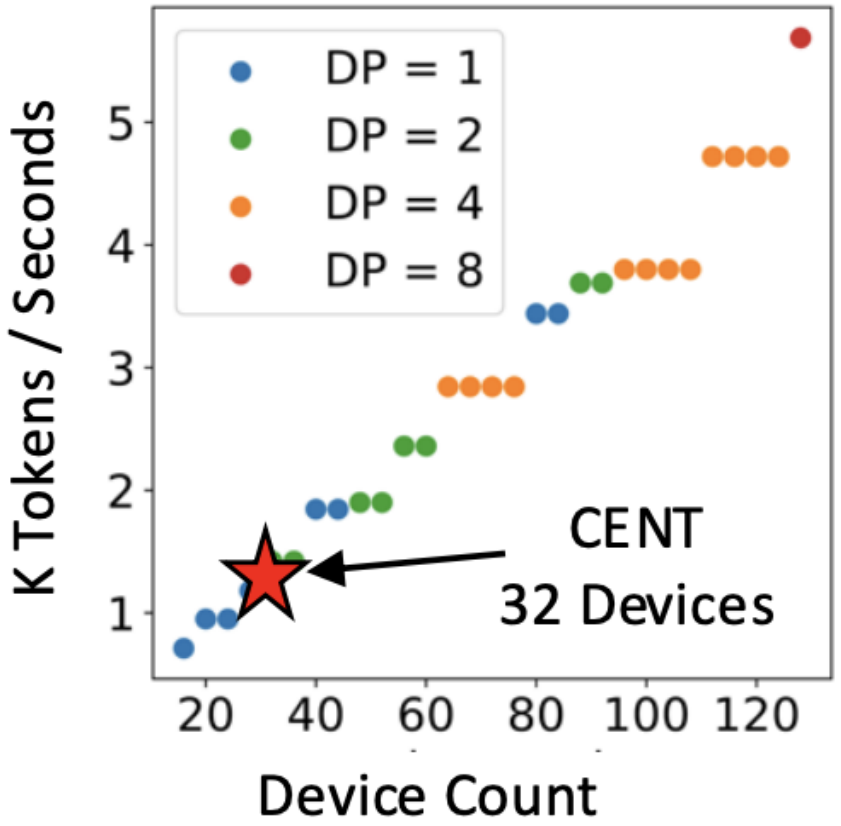


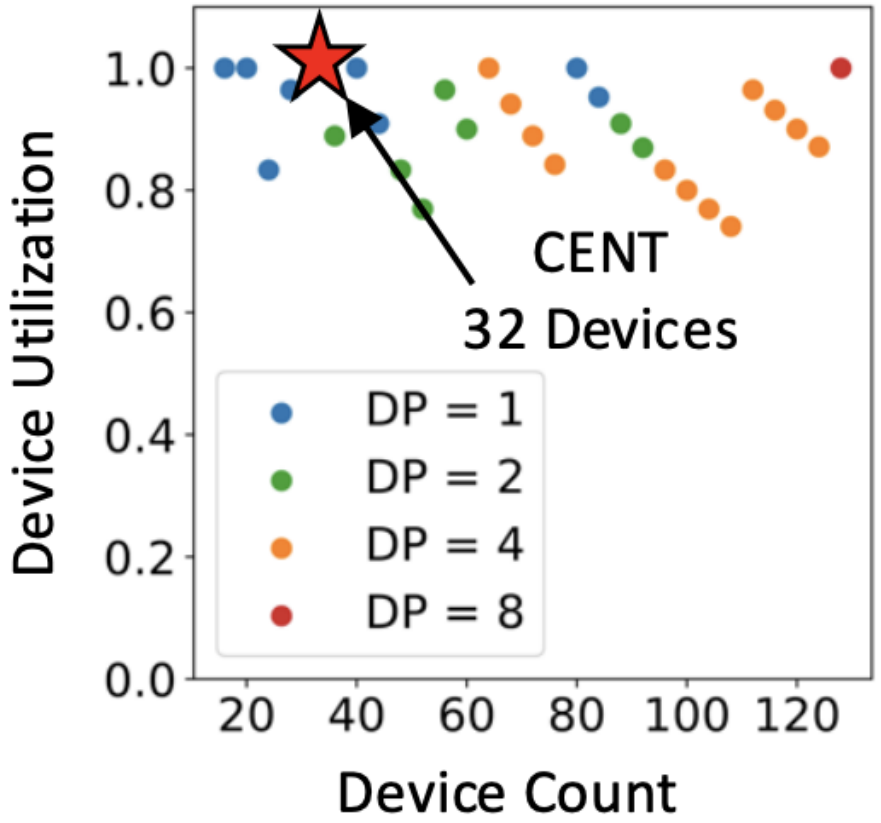
Comparison at 4K Context

CENT Latency Breakdown

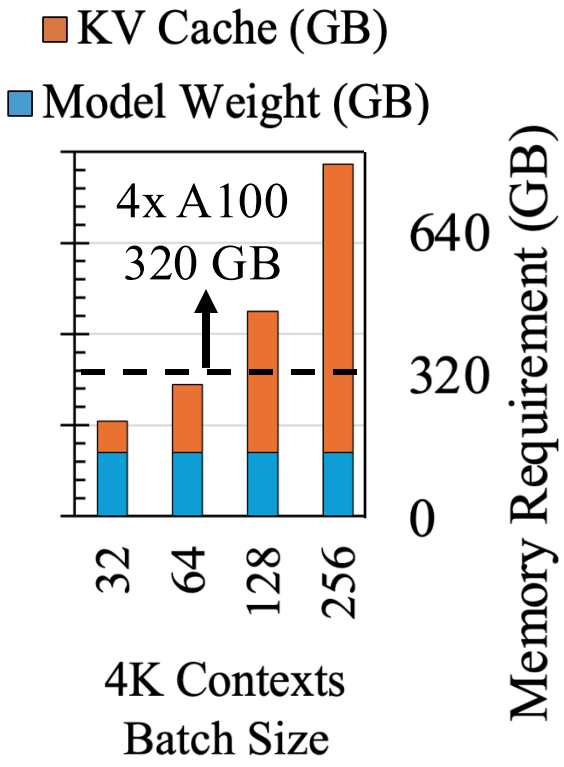


Scalability Study





CENT: A CXL-Enabled PIM System



Limited Batch

Sequential self-attention execution

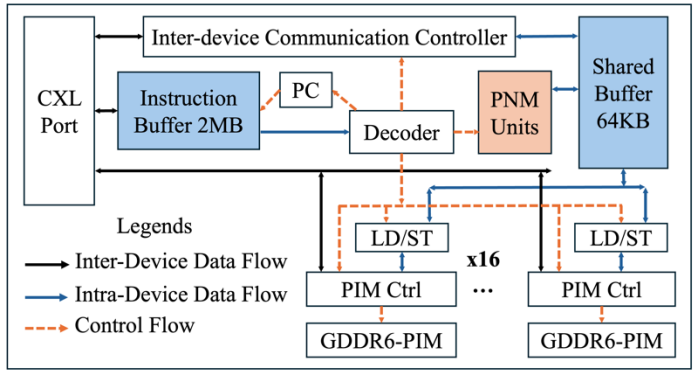
Low GPU Utilization

Low Operational Intensity

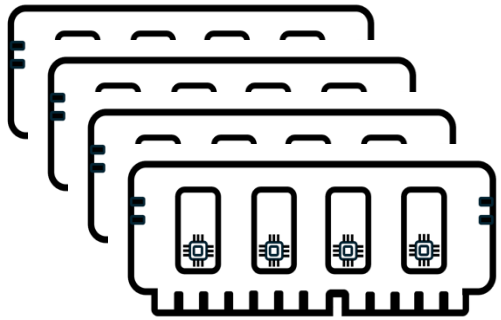
16 TB/s per GDDR6 channel

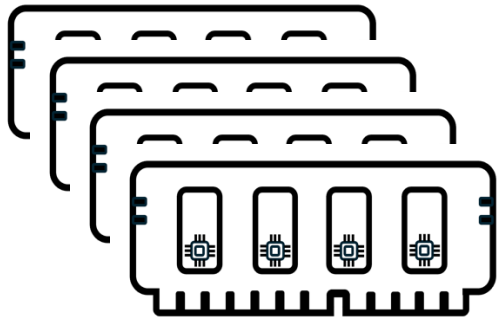
DRAM Bank

PU

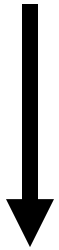


CENT: A CXL-Enabled PIM System





2.4x
Hardware Cost



2.3x
Throughput

5.2x
Tokens per $

PIM Is All You Need: A CXL-Enabled GPU-Free System for Large Language Model Inference

CENT Paper

CENT Artifact