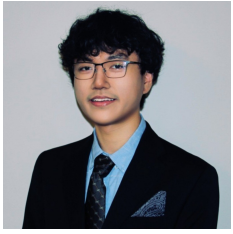
CXL-SpecKV

A Disaggregated FPGA Speculative KV-Cache for Datacenter LLM Serving

Authors



Dong Liu
Yale University

dong.liu@aya.yale.edu

Yanxuan Yu
Columbia University

yy3523@columbia.edu

Motivation: KV-cache is the GPU memory wall

Why KV-cache becomes the wall

- Autoregressive decoding must keep **all past** KV tensors.
- KV footprint scales with **context length** $\times$ **batch size**.

Example (LLaMA-2 70B)

$\sim$ **640 GB** KV-cache
2K tokens, batch 32

Memory pressure caps batch size $\rightarrow$ lower throughput and poor cost-efficiency.

Motivation: what we want instead

Scale KV capacity without slowing decode

- **Capacity**: expand KV space beyond HBM limits
- **Latency**: keep per-token decode close to GPU-local
- **Practicality**: preserve model accuracy and integration simplicity

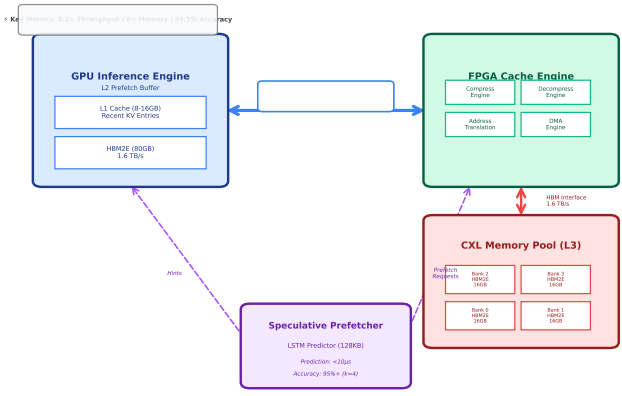
What this enables

- Larger batch $\rightarrow$ higher throughput
- Better cost-efficiency for datacenter serving

System overview: disaggregate + speculate + accelerate

CXL-SpecKV in one slide

- **Disaggregate**: CXL pooling ⇒ 4–8× KV capacity.
- **Accelerate**: FPGA (de)compression + translation ⇒ ∼ 3.2× less bandwidth.
- **Speculate**: prefetch next-k KV pages ⇒ ∼ 95% hit rate.



CXL-SpecKV system architecture.

Outcome

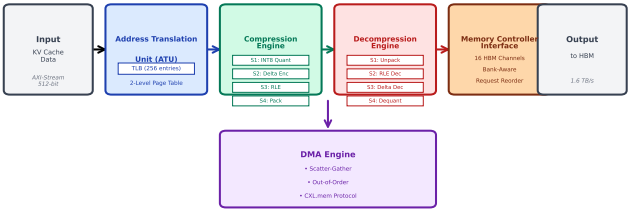
Near GPU-local decode latency with much larger KV capacity, enabling up to 3.2× throughput and 99.5% accuracy.

Method: components & workflow

Core components

- **CXL manager**: L1/L2/L3 + migration
- **Prefetcher**: next-k KV prefetch
- **FPGA engine**: translate + (de)compress
- **Integration**: vLLM / TensorRT-LLM

Latency hiding: overlap prediction → DMA → compute.

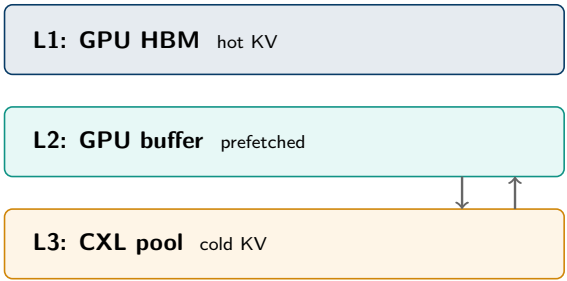


FPGA cache engine datapath (trimmed for readability).

CXL memory manager: hot/cold KV pages

3-tier KV hierarchy

Keep hot KV near the GPU; push cold KV to the CXL pool.



Batch-aware layout: interleave pages for burst DMA.

Policy

Promote hot pages to **L1**; keep cold pages in **L3**.

Why it matters

- **Throughput**: no KV-capacity batch bottleneck
- **Latency**: stable tail latency (GPU-resident working set)

Takeaway

Hot/cold KV management enables capacity scaling *without* tail-latency penalties.

Coherence protocol: device-mediated

Architecture & MESI

GPU (PCIe) → FPGA (home agent) → CXL memory

Device-mediated

GPU does *not* run CXL.cache. FPGA translates; maintains dir; issues invalidations.

I/S/E/M: Invalid, Shared, Exclusive, Modified.

L1∼100 ns; L2∼100 ns; L3∼300 ns.

Paths

Read: dir check; MISS→CXL fetch.
Write: invalidation→MODIFIED.
Prefetch: read-only, no dir update.

Ours vs. native

	Ours	Native	
GPU	PCIe	CXL	*Pragmatic*
Latency	∼50 ns	∼30 ns	
Deploy	Today	Future	

for real-world deployment.

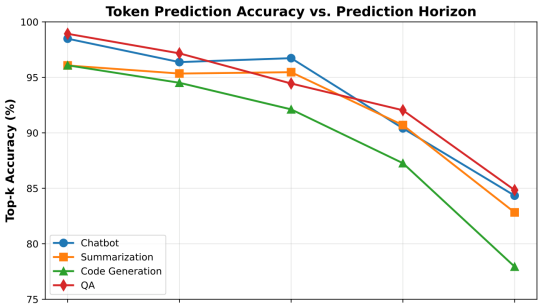
Components

FPGA: `coherence_directory.v` (4096 entries).
SW: `CoherenceManager`.

References

AMD Infinity Fabric, Intel CXL switches, NVIDIA Grace-Hopper, ARM CMN-700.

Speculative prefetching: accuracy → latency hiding



Top-4 accuracy vs. horizon k.

Mechanism

next-k token IDs → KV addresses → non-blocking DMA (L2).

Key numbers (70B)

- **Hit**: **94.7%**
- **Coverage**: **96.3%**
- **Latency**: **383 ns**

Cost

128KB LSTM, $< 10\,\mu s$ FPGA.

Accuracy ⇒ hit rate ⇒ latency hidden.

What the engine does

- **Streaming KV pipeline**: address translation + compression/decompression + DMA
- **Deep pipelining**: $D_{pipe} = 20$ stages for compression and 20 for decompression
- **Per-engine throughput**: **51.2 GB/s** (512b @ 800MHz, II=1)

Implementation (Agilex-7, post-P&R)

812 MHz

max frequency

ALM	30.5%
DSP	25.9%
M20K	15.8%
REG	14.0%

Compression

3.2×

avg ratio (2.5–4.0 across layers)

Decompression

25 cycles

critical path; $\bar{L} \approx 19$ cycles w/ bypass

Accuracy (WikiText-103)

3.21× compression with ∼1.2% perplexity change (vs. FP16).

Latency: hide CXL overhead

Measured overhead vs. GPU-only

LLaMA-2 70B chatbot, batch 32 (see paper latency breakdown table).

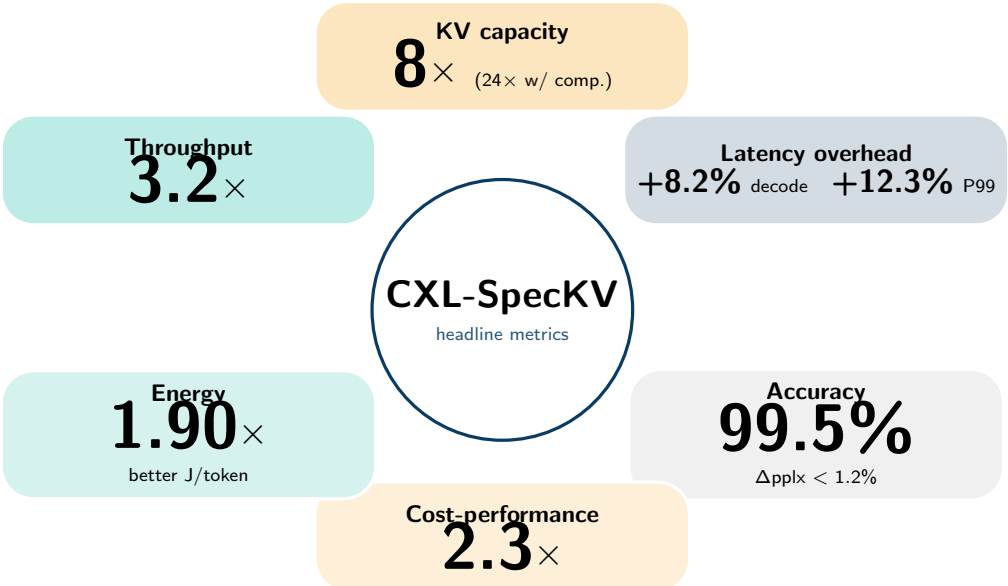
TTFT	Decode	Tail (P99)
+4.2%	+8.2%	+12.3%
prefill (sequential access)	per-token latency	occasional prefetch misses

Why P99 rises: rare misses require synchronous CXL fetch; otherwise access is overlapped by prediction + DMA.

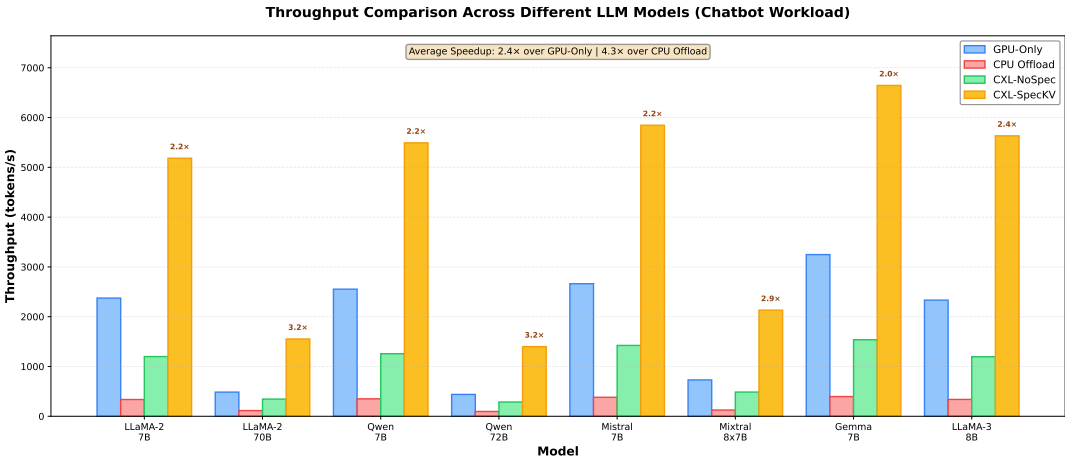
Takeaway
High hit rate $\Rightarrow$ CXL latency is mostly hidden; decode stays close to GPU-local.

Practical tip
If you need strict tail SLOs, avoid GPU queuing (e.g., target P99 $<$ 50ms with moderate batch sizes).

Results at a glance

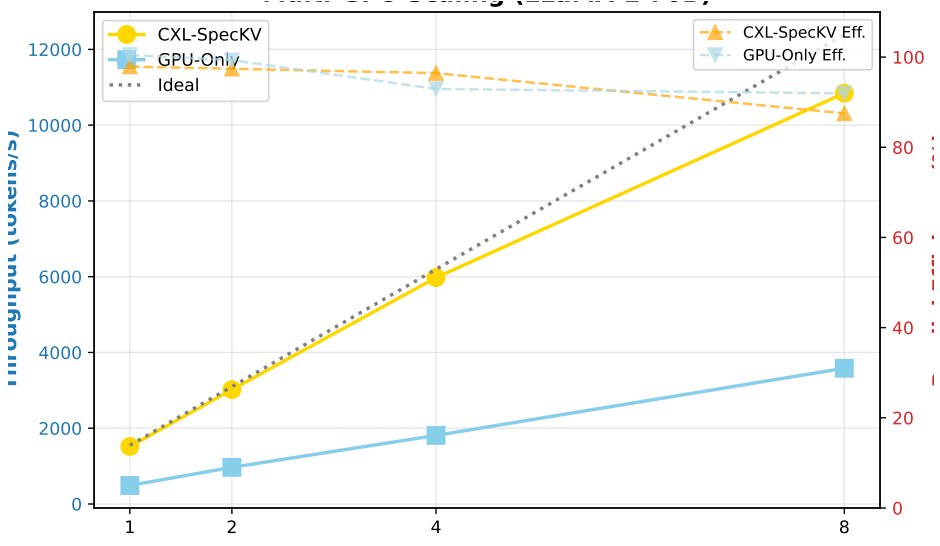


Results: throughput



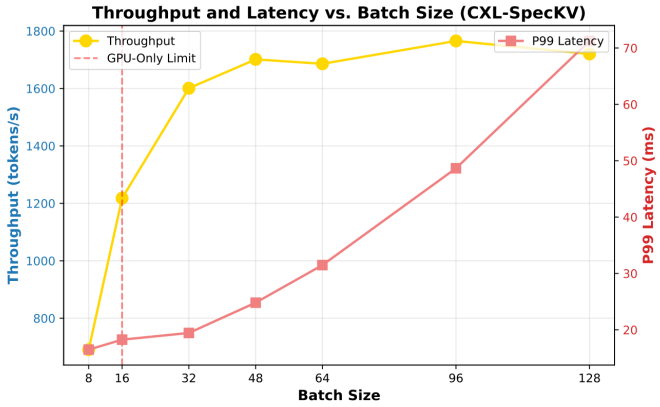
Highlights: 2.1–3.2× vs. GPU-only; 4.3× avg vs. CPU offload.

Scalability: multi-GPU



Takeaway: 95% @ 4 GPUs; 87% @ 8 GPUs; 1 FPGA → 2 GPUs.

Batch size sensitivity

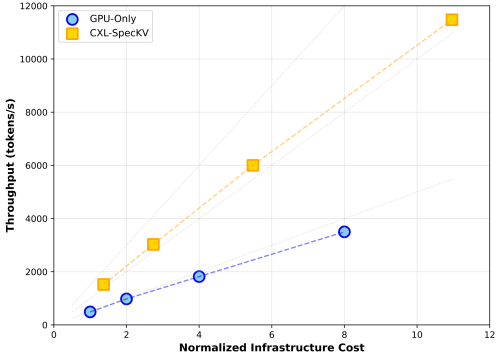


Rule of thumb: batch 32–48 for P99 $<$ 50ms; batch 64–96 for peak throughput.

Cost-performance & energy

Summary

- 2.3 × better cost-performance
- 1.90 × energy efficiency
- 8 × KV capacity (24 × with compression)



Takeaways

Three takeaways

- **Scale KV capacity** via CXL-attached FPGA memory pooling.
- **Hide CXL latency** with speculative *KV-page* prefetching (predict addresses).
- **Reduce bandwidth** using an FPGA cache engine for translation + (de)compression.

In one sentence

Disaggregate KV + speculate prefetch + accelerate (de)compression $\Rightarrow$ scalable LLM serving without sacrificing responsiveness.

Code (open source): `https://github.com/FastLM/CXL-SpecKV`

Next: CXL 3.0 bandwidth scaling · multi-tenant QoS · 32K–128K contexts.

Contact: dong.liu@aya.yale.edu yy3523@columbia.edu

CXL-SpecKV FPGA Hardware (1/2)

Directory structure

`hardware/`

- `rtl/` – RTL sources
- `scripts/` – Quartus build

Key RTL modules

`atu.v`, `kv_compress.v`, `kv_decompress.v`, `dma_engine.v`, `cxl_mem_if.v`, `coherence_directory.v`, `prefetch_core.v`, `cxl_speckv_top.v`

Components overview

1. **ATU** (`atu.v`): TLB (256 entries), 4KB pages; 4 cycles hit, 15 cycles page walk
2. **KV compress** (`kv_compress.v`): 20-stage pipeline, Algorithm 2; 1 entry/cycle throughput
3. **KV decompress** (`kv_decompress.v`): Inverse pipeline; RLE $\rightarrow$ delta $\rightarrow$ dequantize
4. **DMA engine** (`dma_engine.v`): Scatter-gather, 16 descriptors, AXI4 master

CXL memory interface

`cxl_mem_if.v`: CXL.mem + CXL.cache; burst transfers; cache invalidation on writes

CXL-SpecKV FPGA Hardware (2/2)

Core components (cont'd)

6. Coherence Directory (`coherence_directory.v`)

- MESI protocol; 4096-entry dir; up to 4 sharers
- CXL home agent: GPU $\leftrightarrow$ FPGA $\leftrightarrow$ CXL coherent

7. Prefetch core (`prefetch_core.v`): Algorithm 1; LSTM prediction, ATU, L1/L2 check, DMA gen.
8. Top-level (`cxl_speckv_top.v`): MMIO, CXL.mem/cache, component interconnect

Build

`quartus_sh -t build_quartus.tcl`; top: `cxl_speckv_top`; device: Intel Agilex-7

Resource utilization (Agilex-7)

ALMs	30%
DSP	26%
M20K	16%
f_{max}	812 MHz

Integration

- LSTM: Nios II / HLS / custom RTL; $< 10\,\mu$s @ 800 MHz
- CXL IP: CXL.mem, CXL.cache
- HBM: 16 ch, AXI4, 1.6 TB/s agg.

See: `COHERENCE_IMPLEMENTATION.md`, `ARCHITECTURE.md`

Q&A

Q: Does the GPU run CXL.cache?

A: No. GPU uses PCIe for communication; FPGA acts as CXL home agent and translates.

Q: Where is coherence managed?

A: On the FPGA. Directory is authoritative; FPGA issues all invalidations.

Q: Is this "real" coherent memory?

A: Yes. Device-mediated coherence is standard in heterogeneous systems (e.g., AMD Infinity Fabric, Intel CXL switches).

References


M. K. Aguilera, N. Amit, W. J. Bolosky, A. Chaugule, C. Coppens, J. Duchesne, C. Guo, et al.
Remote regions: a simple abstraction for remote memory.
arXiv preprint arXiv:1908.06189, 2019.


M. AI.
The llama 3 herd of models.
arXiv preprint arXiv:2407.21783, 2024.


K. Alizadeh, I. Mirzadeh, D. Belenko, K. Khatamifard, M. Cho, C. C. Del Mundo, M. Rastegari, and M. Farajtabar.
Llm in a flash: Efficient large language model inference with limited memory.
arXiv preprint arXiv:2312.11514, 2024.

Kivi: A tuning-free asymmetric 2bit quantization for kv cache.
arXiv preprint arXiv:2402.02750, 2024.


X. Miao, G. Oliaro, Z. Zhang, X. Cheng, Z. Wang, Z. Wong, Y. Zhu, and Z. Jia.
Specinfer: Accelerating generative llm serving with speculative inference and token tree verification.
arXiv preprint arXiv:2305.09781, 2024.


R. Nallapati, B. Zhou, C. Gulcehre, B. Xiang, et al.
Abstractive text summarization using sequence-to-sequence rnns and beyond.
In *Proceedings of The 20th SIGNLL Conference on Computational Natural*