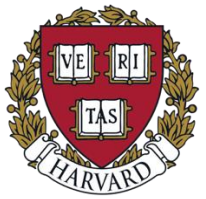
NEO: Saving GPU Memory Crisis with CPU Offloading for Online LLM Inference

Xuanlin Jiang[1], **Yang Zhou**[2,3], Shiyi Cao[2], Ion Stoica[2], Minlan Yu[4]

[1]Peking University, [2]UC Berkeley, [3]UC Davis, [4]Harvard University

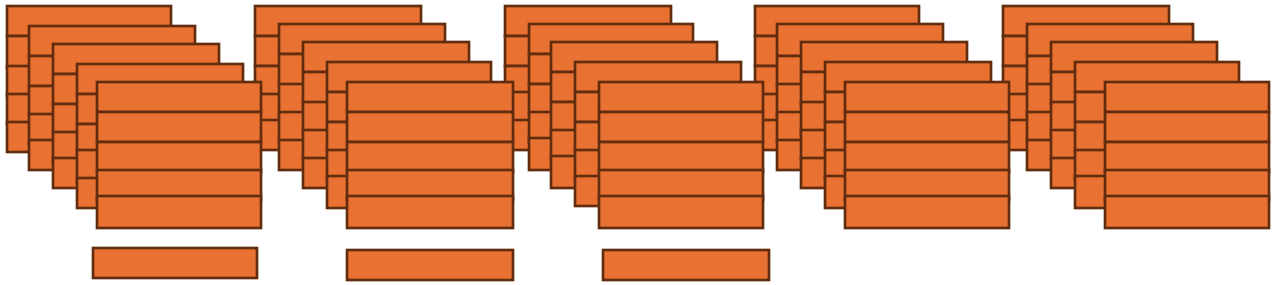


The GPU Memory Crisis

- The GPU memory crisis in LLM inference

5 requests
(5000 tokens each)

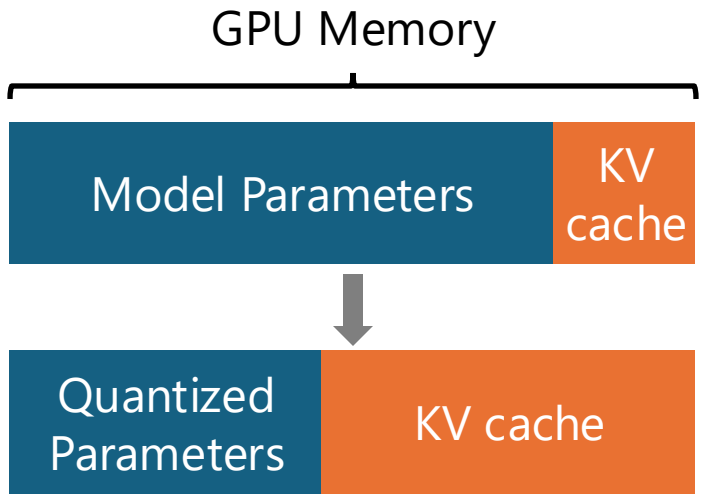
Batch size needed to saturate GPU: ~128



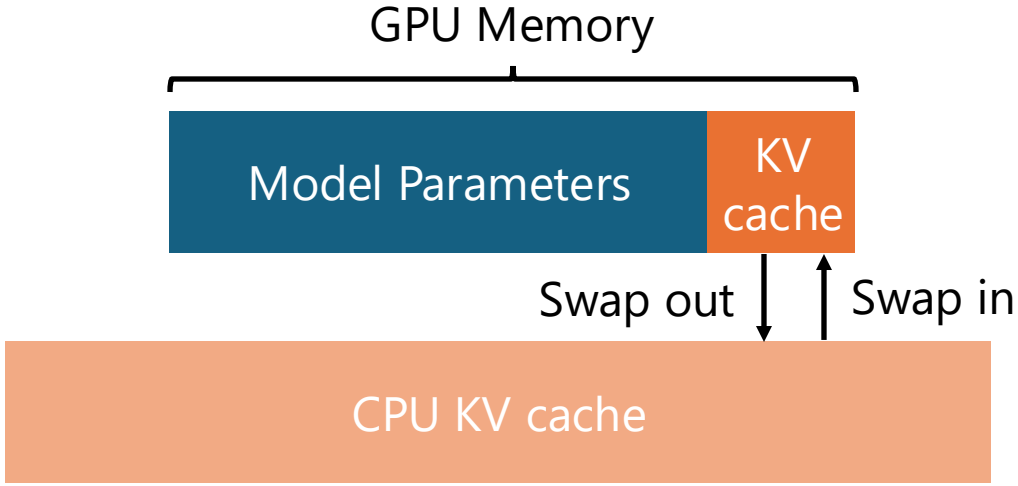
Small GPU memory size → Small KV-cache size → Small batch size → Low GPU utilization → Low inference throughput

Existing Solutions

- Solution #1: quantization & sparsification
  - Could hurt output quality

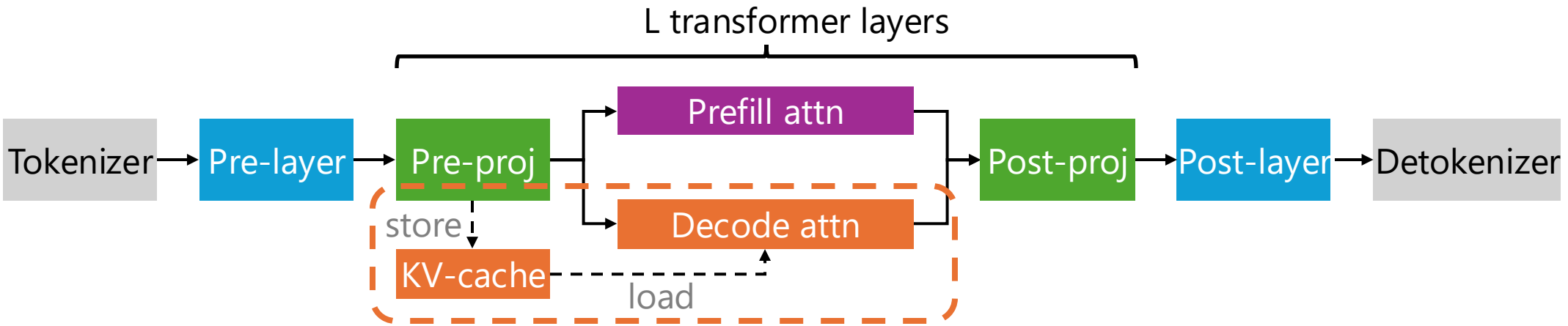


- Solution #2: memory offloading + on-demand swapping
  - PCIe bandwidth is too low



NEO Key Insights

- Observation #1: Only decoding attention requires KV-cache



KV-cache is only used here!

NEO Key Insights

- Observation #2: CPUs are much closer to GPUs in terms of memory bandwidth than computation, while decoding attention is memory-bound

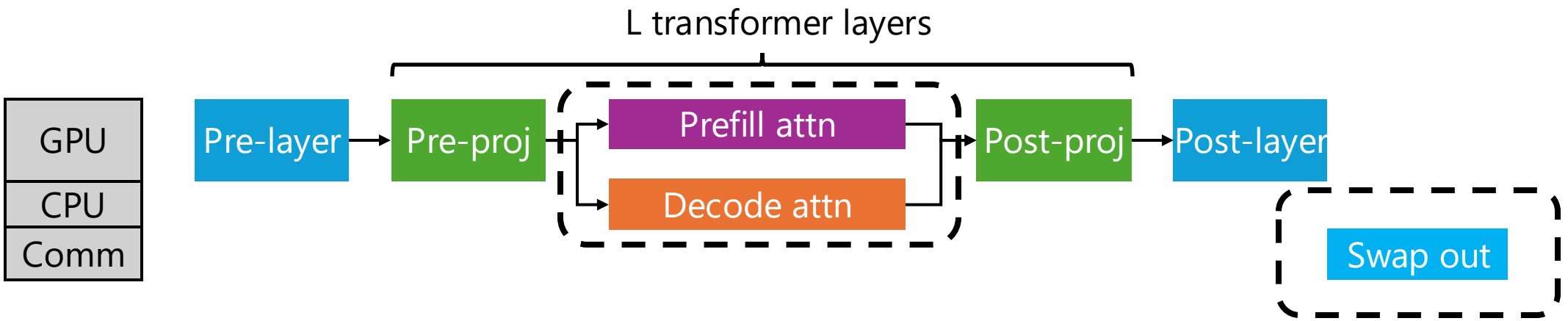
- FastDecode[1]: modern x86 server CPU vs A10G GPU
  - Computation: 1 vs. 125 TFLOPS
  - Memory bandwidth: 200 vs. 600 GB/s
  - Memory bandwidth is **1~2 orders of magnitudes** closer!

[1] He, J. and Zhai, J. FastDecode: High-Throughput GPU Efficient LLM Serving using Heterogeneous Pipelines

Challenges

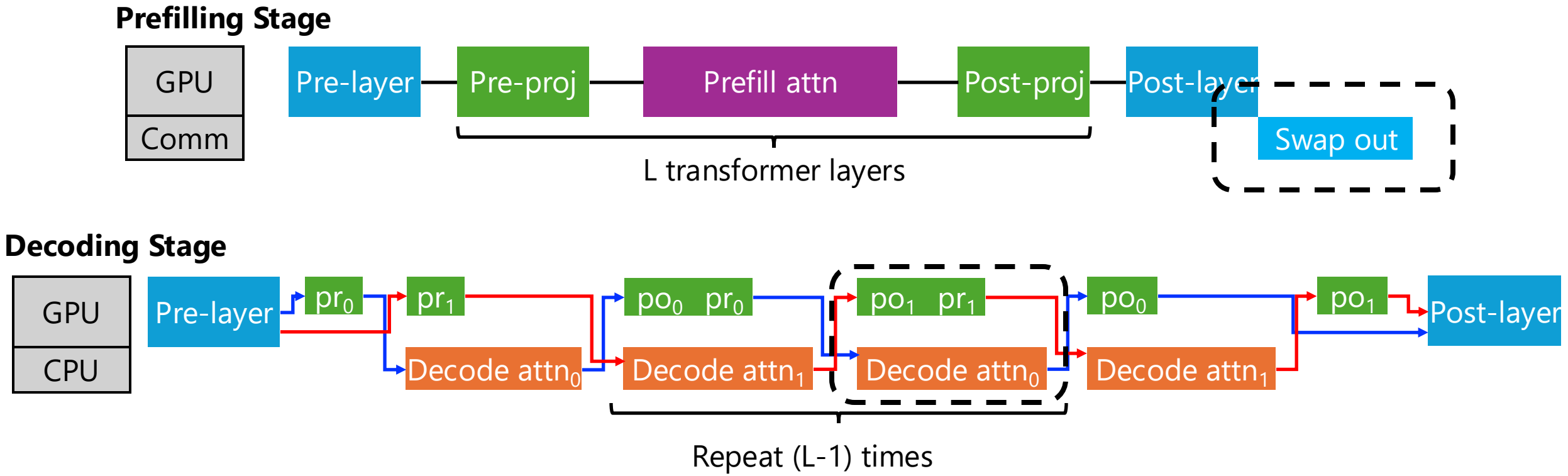
- Key insight: offload KV-cache & decoding attention computation to CPU
- How to efficiently overlap CPU and GPU within each iteration?
  - Should reconstruct the inference pipeline
- How to efficiently schedule requests across inference iterations?
  - Should use a dynamic adaptive scheduling policy

Strawman #1: Simple Offloading



- Problems:
  - No CPU-GPU overlapping
  - No computation-communication overlapping

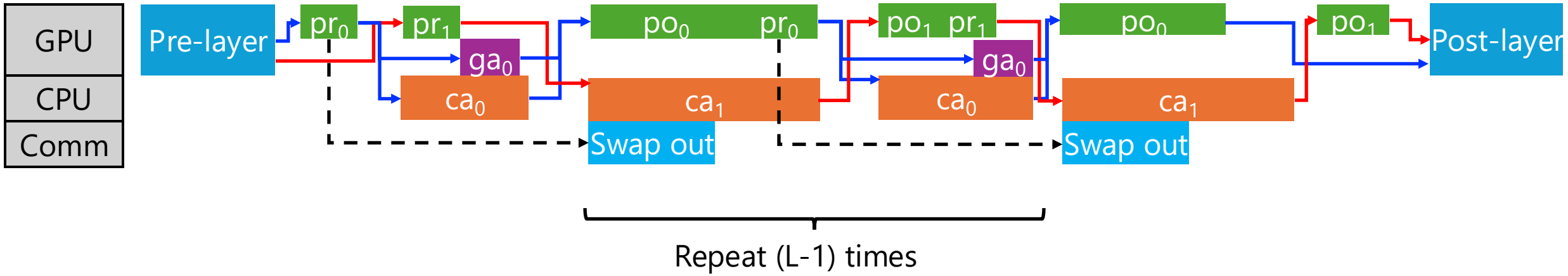
Strawman #2: Symmetric Pipelining (FastDecode)



- Problems:
  - Insufficient overlapping
  - Unused GPU memory

NEO Asymmetric Pipelining

ga = GPU attn (prefill + decode mixed)
ca = CPU attn (decode only)



- Benefits:
  - Mixed batching + layer-wise swapping → high overlapping
  - Partial offloading → high GPU memory utilization

NEO Load-Aware Scheduling: Principles

- **Greedy**
  - Choose between GPU-only schedule and asymmetric pipelining schedule
  - Pick the one with a higher estimated throughput
- **Balancing**
  - ”Bubbles” should be minimized
- **Hiding CPU**
  - GPU must be busy when CPU is busy
- **Maximizing GPU**
  - Sufficient batch size should be achieved to utilize GPU fully

Check our paper for the detailed scheduling algorithm!

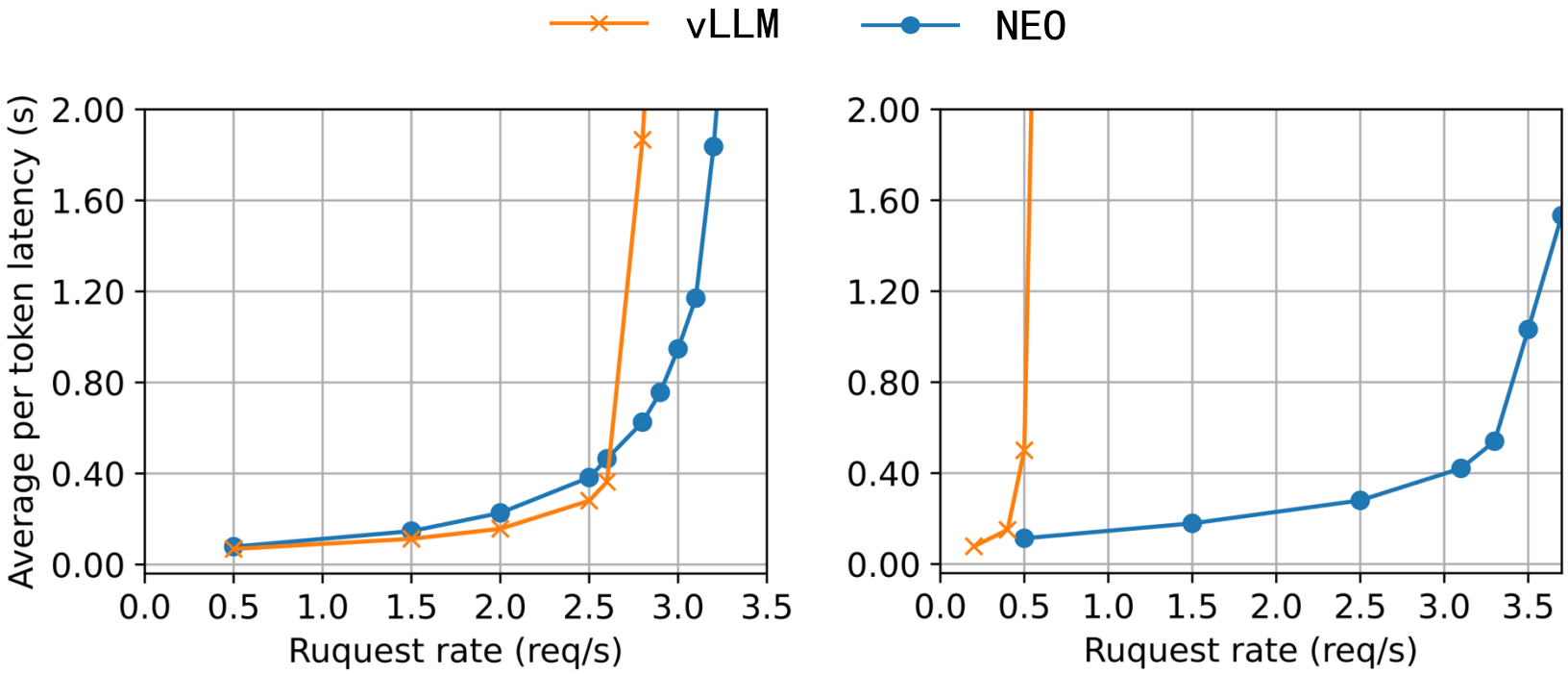
Evaluation: Setup

- Datasets:

Dataset	Avg. Input Length	Avg. Output Length
Azure Code	2047.85	27.88
OpenAI Summary Comparison	372.03	45.25

- Models: Llama-3.1-70B, Llama-3.1-8B, Llama-2-7B
- GPUs: 2×H100, A10G, T4

Evaluation: Latency vs Load

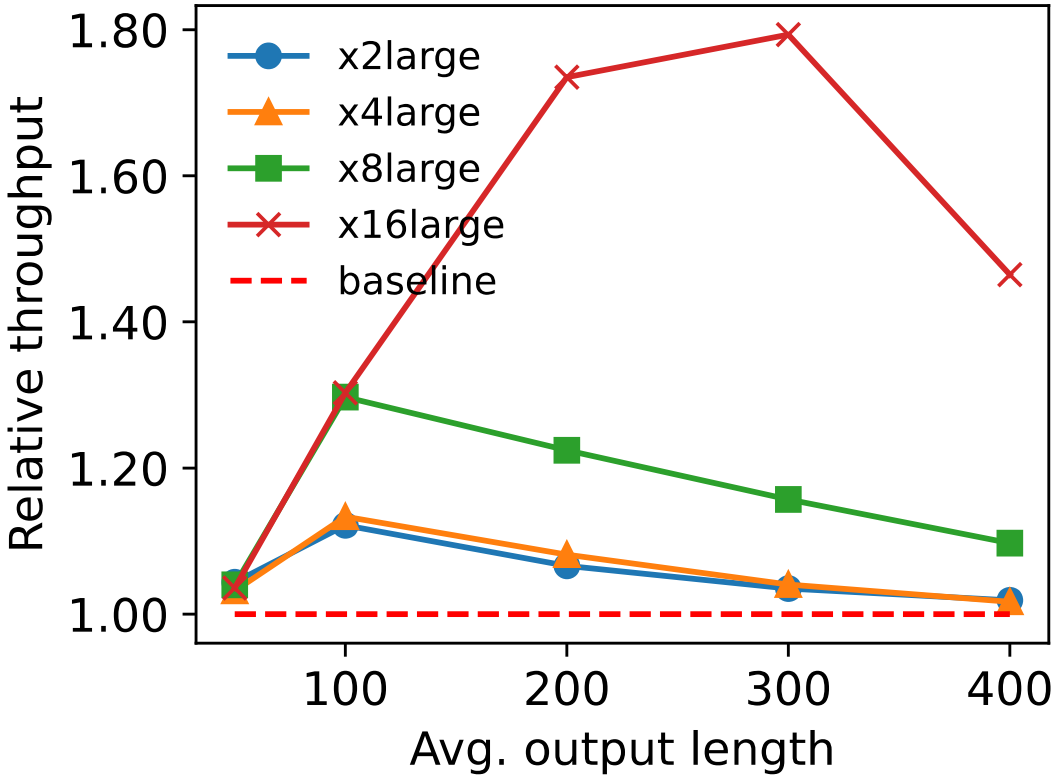


2xH100 + Llama-3.1-70B
Azure Code

T4 + Llama-2-7B
OpenAI Summary Comparison

- 14% higher throughput on H100 (at 2 sec latency)
- 6.6× higher on T4 (at 1 sec latency)

Evaluation: Throughput



A10G + Llama-3.1-8B

- Up to 79% higher throughput on A10G (increasing with more CPU cores)

Conclusion

- NEO: the first CPU offloading system for online LLM inference that achieves performance gains over GPU-only systems
  - … with the same hardware cost (i.e., local host CPU) and inference accuracy
  - Key techniques: asymmetric pipelining & load-aware scheduling
  - Up to 14%-6.6× throughput gains on different GPUs

GitHub repo: NEO-MLSys25/NEO

Thank You!