An I/O Characterizing Study of Offloading LLM Models and KV Caches to NVMe SSD

Zebin Ren[1], Krijn Doekemeijer[1], Tiziano De Matteis[1],
Christian Pinto[2], Radu Stoica[3], Animesh Trivedi[3]

[1]VU Amsterdam
[2]IBM Research, Dublin, Ireland
[3]IBM Research, Zurich, Switzerland

@Large Research
Massivizing Computer Systems
https://atlarge-research.com/

Background

LLM-based applications are becoming widely used.

ChatGPT
deepseek
Gemini
Claude
IBM
Llama 3
Granite

High memory cost

Model	Model Size (FP8)
Mistral-Large	123 GiB
GPT3-175B	175 GiB
OPT-175B	175 GiB
Llama3-405B	405 GiB

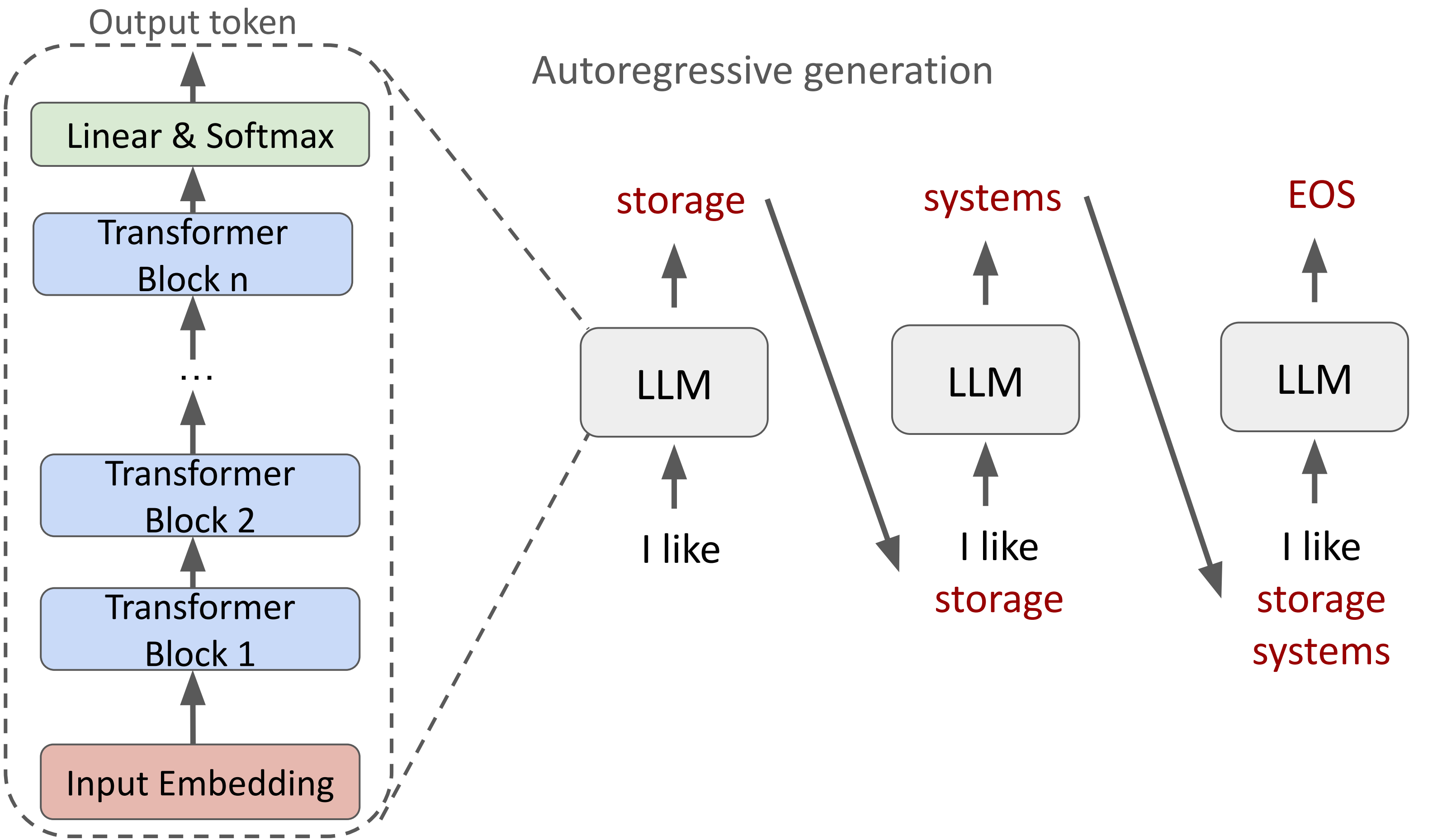
Modern LLMs fail to fit into a single GPU memory.

Solution:

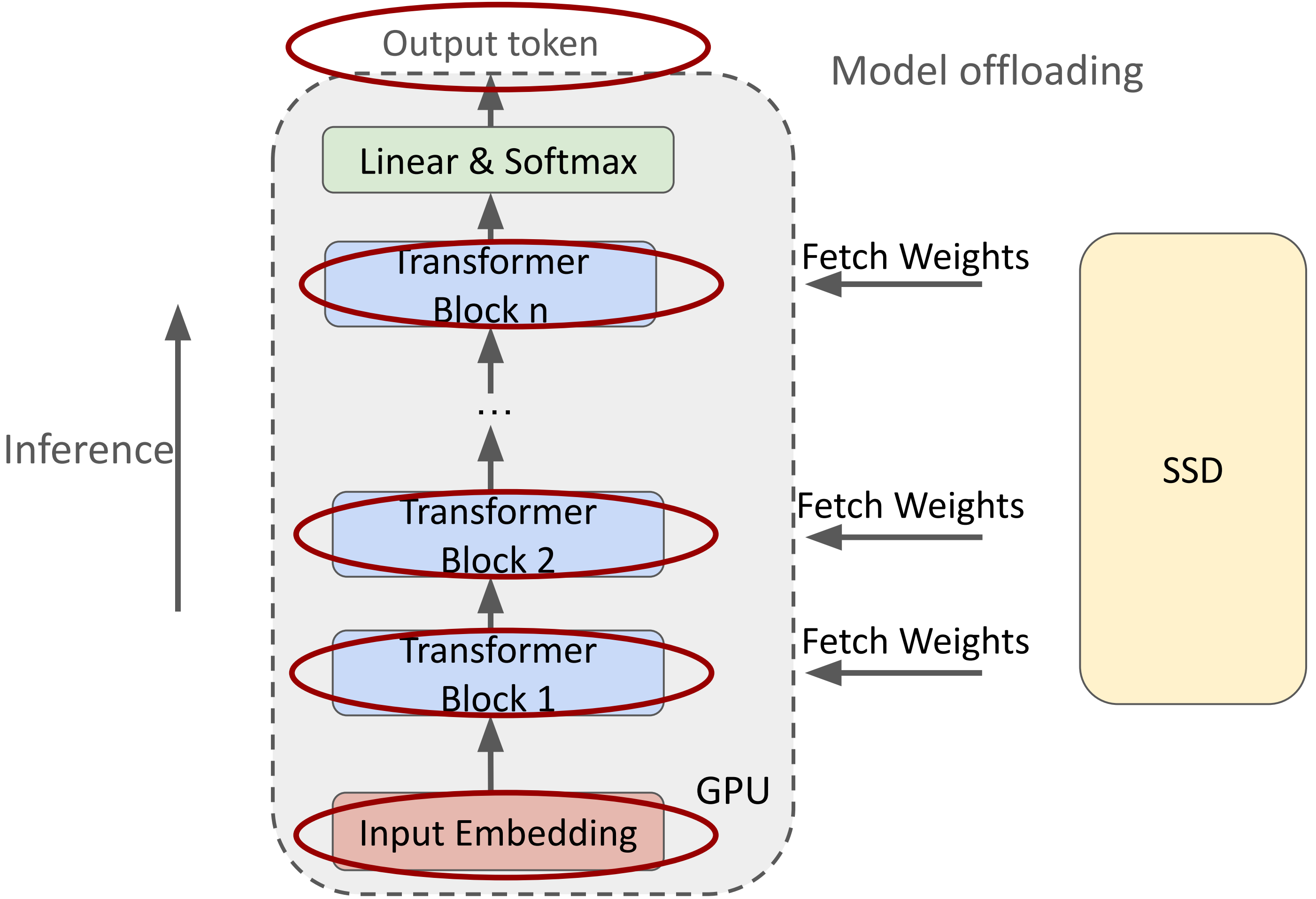
- Parallelism with multiple GPU.
- Quantization and sparsity.
- Compression.
- Model offloading.
- …

This work: Offloading the GPU usage to SSD during LLM inference.

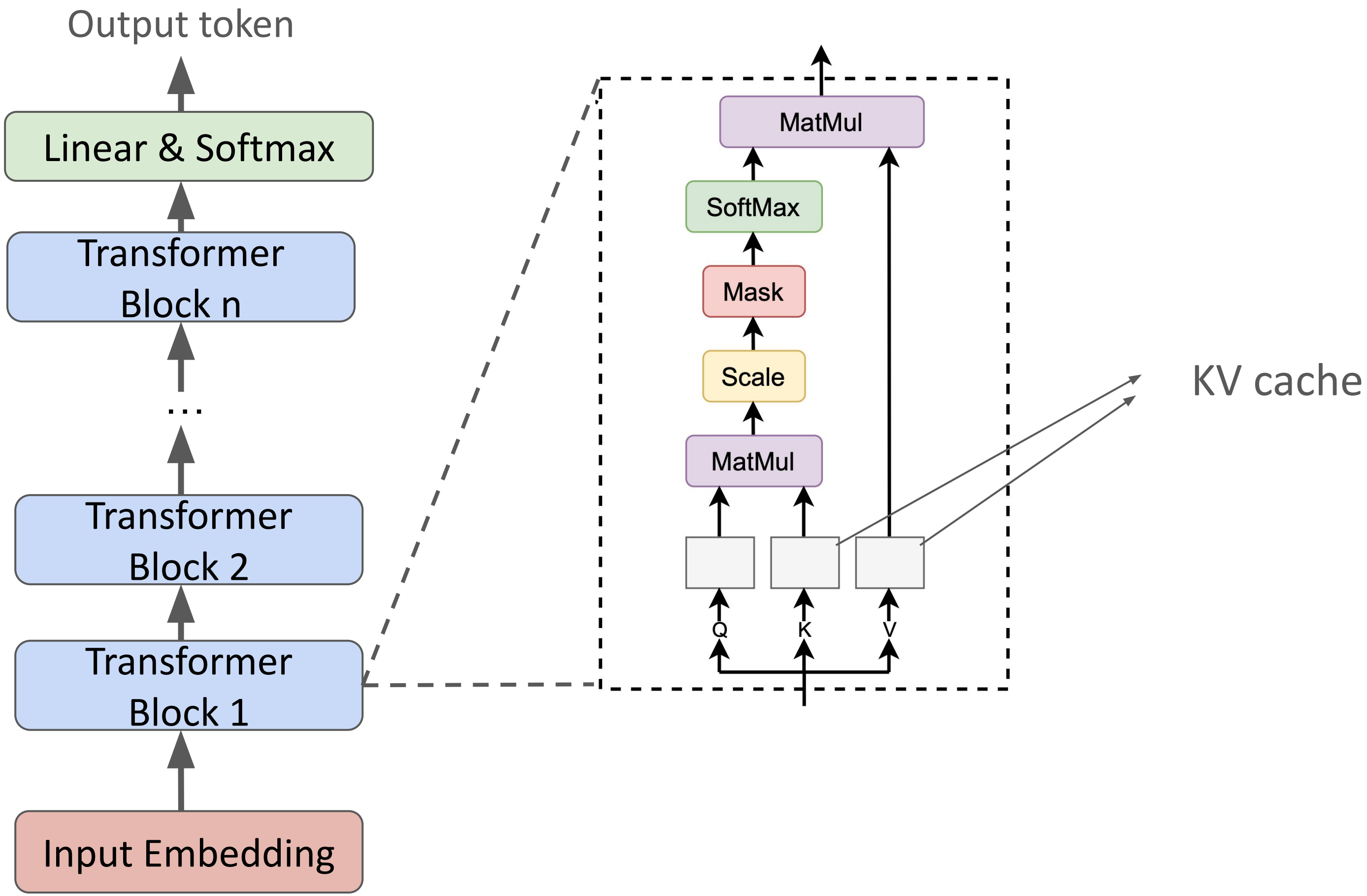
Offloading During LLM Inference



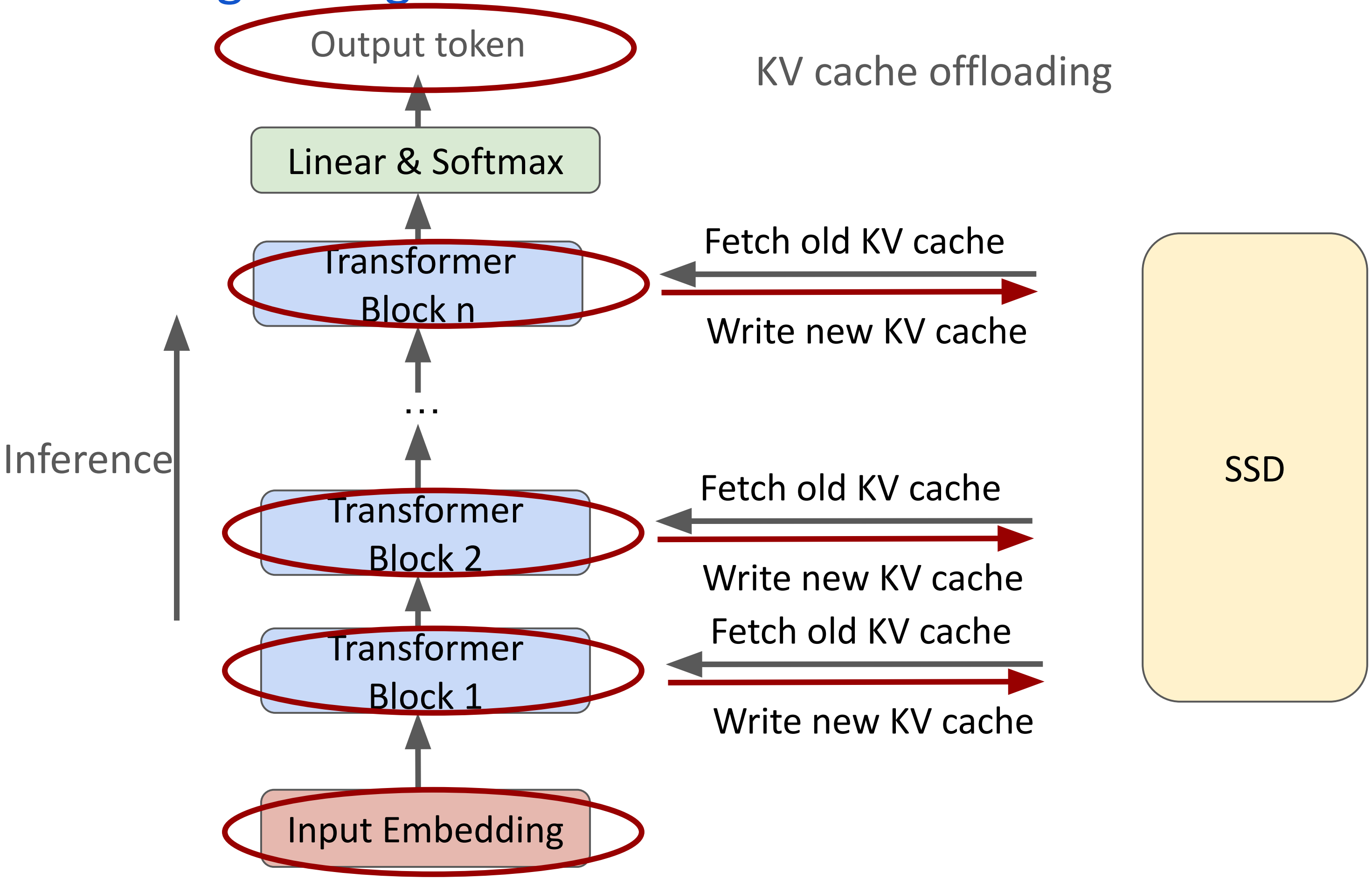
Offloading During LLM Inference



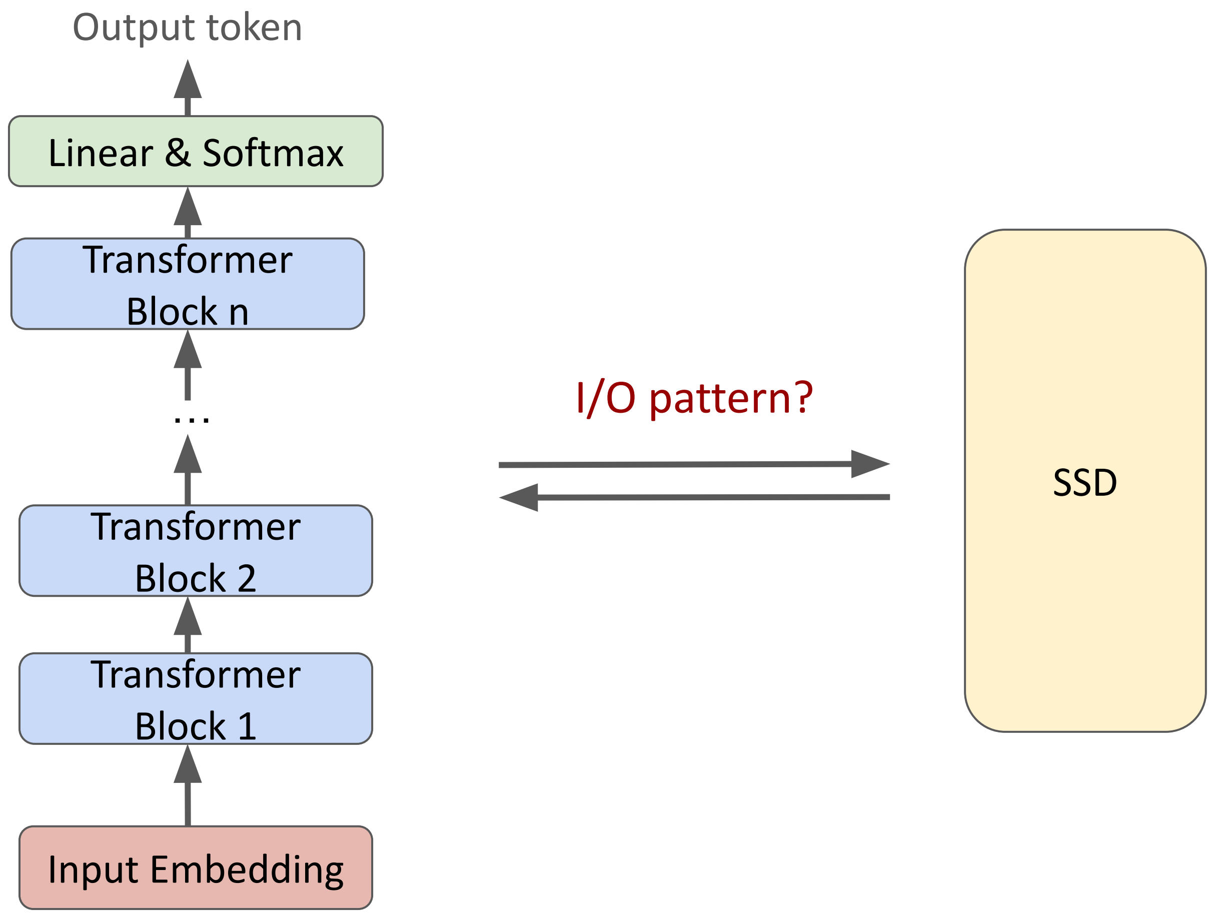
Offloading During LLM Inference



Offloading During LLM Inference



Offloading During LLM Inference



Research Questions

Q1: What is the tensor (model weights and KV cache) transferring bandwidth between the CPU/GPU memory and the SSDs?

- DeepNVMe — An I/O library for deep learning.
- Two interfaces: POSIX and libaio.

Q2: What are the I/O access patterns of model offloading?

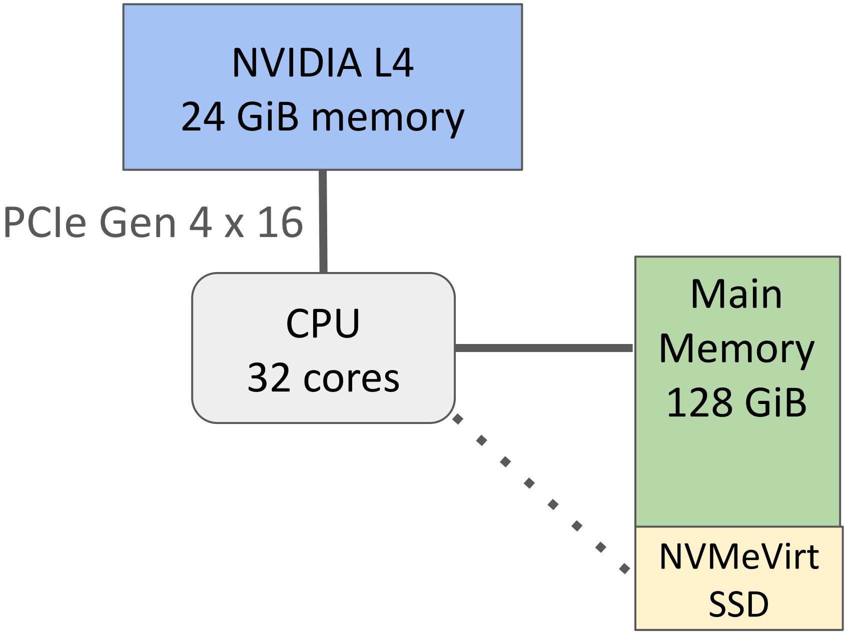
- DeepSpeed and FlexGen.
- Offload the model to NVMe disk and keep all other data in GPU.

Q3: What are the I/O access patterns of KV cache offloading?

- FlexGen.
- Offload the KV cache to NVMe disk and keep all other data in GPU.

Experiment Setup

Hardware



Software

NVMeVirt (SSD emulator)

- 4 CPU cores
- 5.3 GiB/s@512KiB with 1 thread
- 16.9 GiB/s@512KiB with 4 threads

LLM I/O library:

- DeepNVMe

LLM framework:

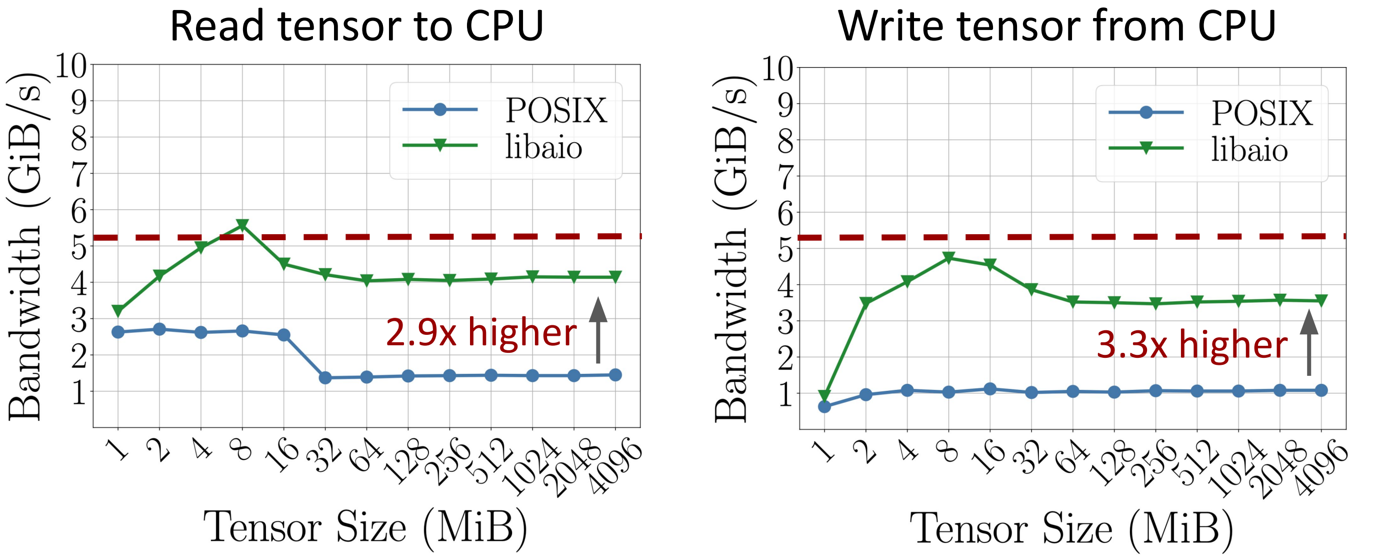
- DeepSpeed Inference
- FlexGen

LLM models:

- OPT 6.7B, 13B and 30B with FP16 quantization

Q1: Tensor transferring bandwidth between the CPU/GPU memory and the SSDs.

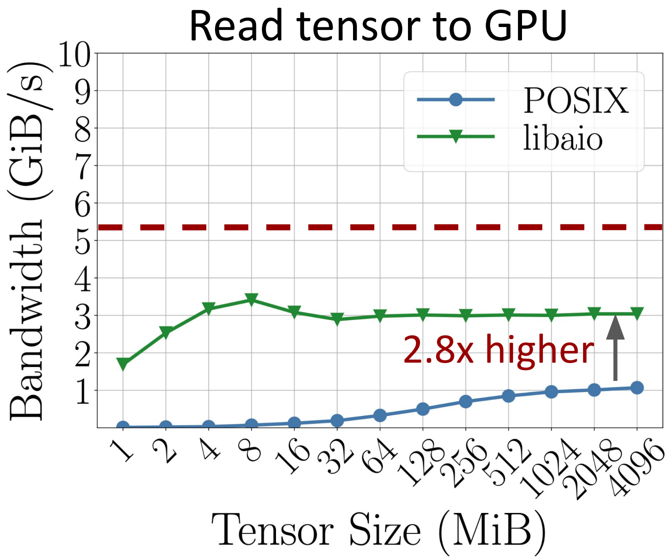
Transferring Tensors with DeepNVMe

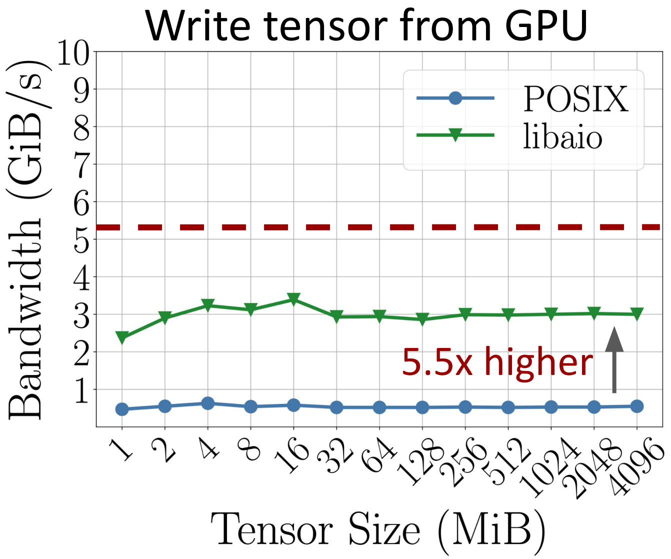


libaio delivers up to 2.9×, 3.3× higher bandwidth than POSIX for reading tensors to CPU, writing tensors from CPU.

Neither POSIX or libaio can reach the single-thread bandwidth of the NVMe SSD (5.3 GiB/s).

Transferring Tensors with DeepNVMe



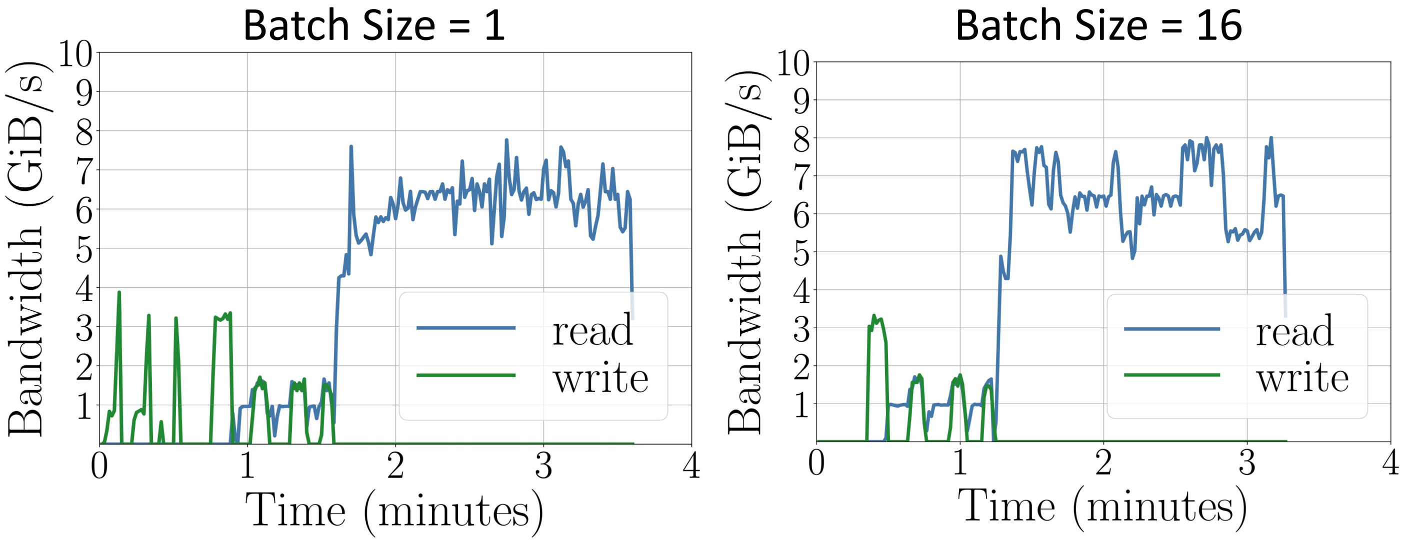


libaio delivers up to 2.8×, 5.5× higher bandwidth than POSIX for reading tensors to GPU, writing tensors from GPU.

Transferring tenser between GPU and SSD achieves lower bandwidth than transferring tenser between CPU and SSD.

Q2: I/O access patterns of model offloading.

Model offloading with DeepSpeed (OPT-13B)



The model is offloaded once and then read-only.

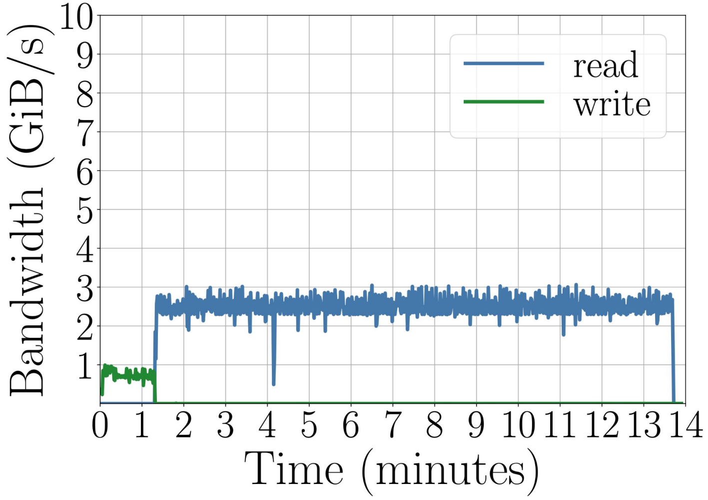
The batch size does not lead to increased I/O traffic.

The block-level I/O is dominated by 128 KiB reads and writes.

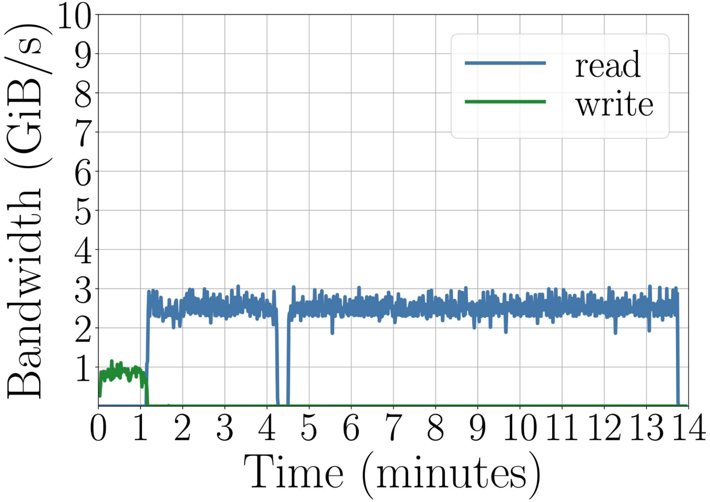
The SSD's sectors are uniformly accessed, indicating there is no hot spots.

Model offloading with FlexGen (OPT-30B)

Batch Size = 1



Batch Size = 16

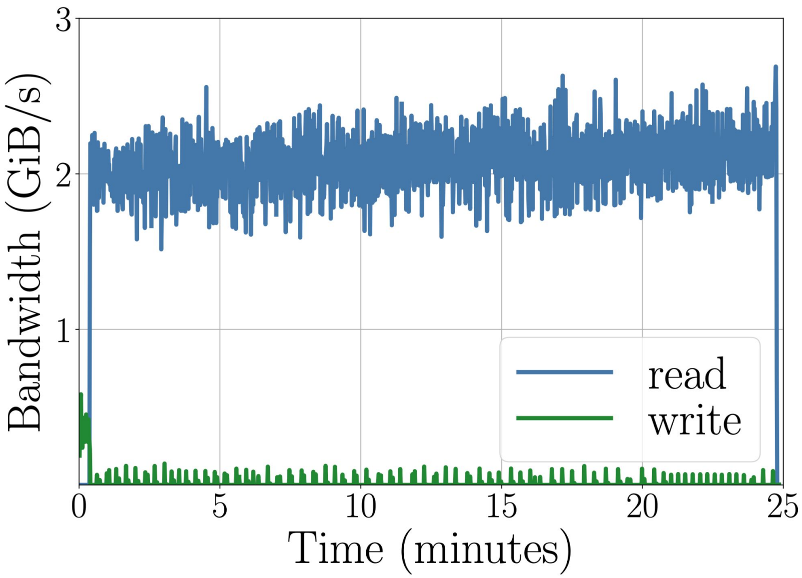


Similar I/O pattern between frameworks (DeepSpeed and FlexGen).

Neither DeepSpeed nor FlexGen can saturate the NVMeVirt SSD.

Q3: I/O access patterns of KV cache offloading.

KV Cache Offloading with FlexGen (OPT-6.7B, Batch Size 64)



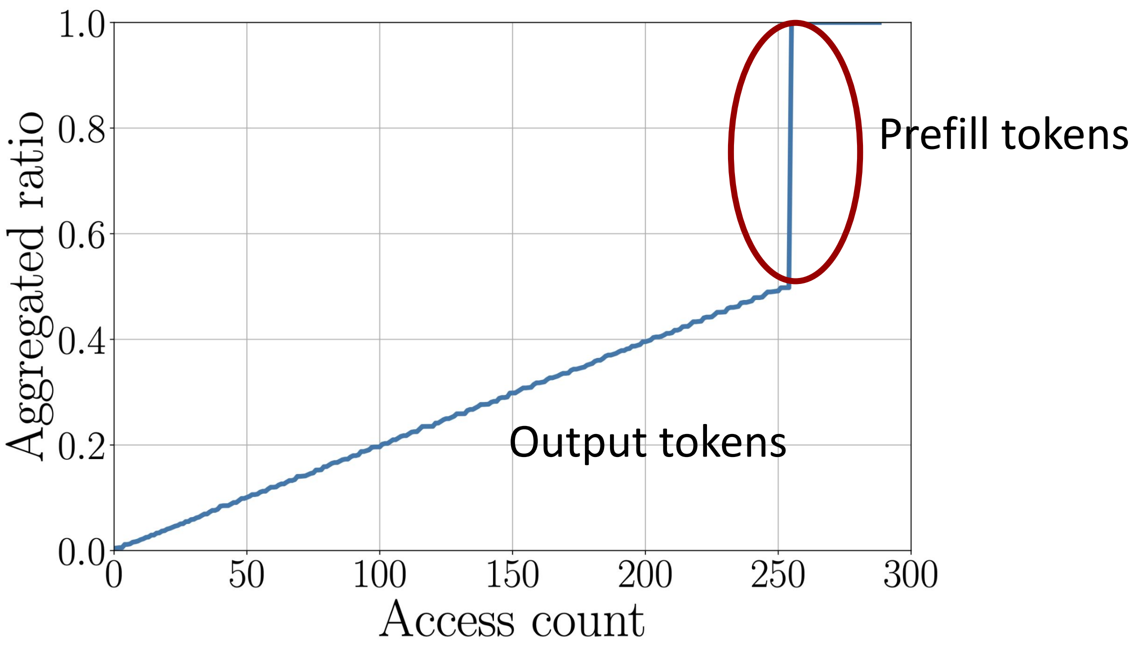
Average: 2.0 GiB/s.

Average: 11.0 MiB/s.

The read bandwidth of the KV cache is significantly higher than the write bandwidth.

The block-level I/O is dominated by 128 KiB reads and writes.

KV Cache Offloading with FlexGen (OPT-6.7B, Batch Size 64)



The sectors in KV cache offloading are accessed non-uniformly.

The earlier the KV cache is generated, the more times it is accessed.

Take-home Messages

1. Use libaio with tensor transferring for higher bandwidth.

- Use an I/O library optimized for deep learning for higher tensor transfer bandwidth.
- Use asynchronous I/O (such as libaio) than synchronous I/O interface.

2. LLM model offloading is dominated by large reads.

- Neither DeepSpeed nor FlexGen can reaches the maximum speed of the SSD.
- The I/O workload is dominated by 128 KiB reads.
- The sectors are accessed uniformly.

3. LLM KV cache offloading has higher read than write bandwidth.

- Read 2.0 GiB/s vs. write 11.0 MiB/s.
- Non-uniform I/O access.

GitHub

Paper: https://atlarge-research.com/pdfs/2025-cheops-llm.pdf
Source code:
https://github.com/stonet-research/cheops25-IO-characterization-of-LLM-model-kv-cache-offloading-nvme

PDF

Future Work

1. Effect of different hardware on the performance of SSD offloading during LLM inference.

- Different models and numbers of GPUs.
- Different models and numbers of SSDs.

2. LLM model-aware optimizations.

- Sparsity of activations.
- Different contributions of each tokens to the result.

3. Real-world LLM serving workloads.

- Multi-round conversations.

Resources

References

[1] DeepNVMe https://www.deepspeed.ai/tutorials/deepnvme/
[2] DeepSpeed https://github.com/deepspeedai/DeepSpeed
[3] FlexGen https://github.com/FMInference/FlexLLMGen
[4] HuggingFace: OPT model. https://huggingface.co/docs/transformers/en/model_doc/opt
[5] ZeRO-Inference: 20X Faster Inference Through Weight Quantization and KV Cache Offloading. https://github.com/microsoft/DeepSpeedExamples/blob/master/inference/huggingface/zero_inference/README.md
[6] Aminabadi et.al. DeepSpeed-Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. In SC22. https://doi.org/10.1109/SC41404.2022.00051
[7] Sheng et.al. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. In ICML'23 https://proceedings.mlr.press/v202/sheng23a.html
[8] Kim et.al. NVMeVirt: A Versatile Software-defined Virtual NVMe Device. In FAST'23.

Further Reading

[1] Chitty-Venkata et.al. A Survey of Techniques for Optimizing Transformer Inference. J. Syst. Archit. 144 (2023), 102990. https://doi.org/10.1016/J.SYSARC.2023.102990
[2] Gao et.al. Cost-efficient Large Language Model Serving for Multi-turn Conversations with CachedAttention. In ATC'24.
[3] Lee et.al. InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management. In OSDI'24

Thank you!
&
Questions?

Paper: https://atlarge-research.com/pdfs/2025-cheops-llm.pdf
Source code:
https://github.com/stonet-research/cheops25-IO-characterization-of-LLM-model-kv-cache-offloading-nvme

Backup Slides

Background

Model	Model Size (FP8)
Mistral-Large	123 GiB
GPT3-175B	175 GiB
OPT-175B	175 GiB
Llama3-405B	405 GiB

Modern LLMs fail to fit into a single GPU memory.

Solution:

- Parallelism with multiple GPU.
- Quantization and sparsity.
- Compression.
- Model offloading.
- …

This work: LLM offloading to SSD during inference.