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

Zebin Ren
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands

Krijn Doekemeijer
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands

Tiziano De Matteis
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands

Christian Pinto
IBM Research Europe
Dublin, Ireland

Radu Stoica
IBM Research Europe
Zurich, Switzerland

Animesh Trivedi
IBM Research Europe
Zurich, Switzerland

Abstract

With the popularity of generative AI, LLM inference has become one of the most popular cloud workloads. Modern popular LLMs have hundreds of billions of parameters and support very large input/output prompt token sizes (100K–1M). As a result, their computational state during LLM inference can exceed the memory available on GPUs. One solution to this GPU memory problem is to offload the model weights and KV cache to the host memory. As the size of the models and prompts continue to increase, researchers have started to explore the use of secondary storage, such as SSDs, to store the model weights and KV cache. However, there is a lack of study on the I/O characteristics and performance requirements of these offloading operations. In order to have a better understanding of the performance characteristics of these offloading operations, in this work, we collect, study, and characterize the block layer I/O traces from two LLM inference frameworks, DeepSpeed and FlexGen, that support model and KV cache offloading to SSDs. Through our analysis of these I/O traces, we report that: (i) `libaio`-based tensor offloading delivers higher I/O bandwidth for both writing and reading tensors to/from the SSDs than `POSIX`; (ii) the I/O workload of model offloading is dominated by 128 KiB reads for both DeepSpeed and FlexGen in the block layer; (iii) model offloading does not saturate NVMe SSDs; and (iv) the I/O workload of KV cache offloading contains both read and write workloads dominated by 128 KiB requests, but the average bandwidth of read is much higher than write (2.0 GiB/s vs. 11.0 MiB/s). We open-source the scripts and the I/O traces of this work at https://github.com/stonet-research/cheops25-IO-characterization-of-LLM-model-kv-cache-offloading-nvme

CCS Concepts: • **General and reference** → *Empirical studies*; • **Software and its engineering** → **Secondary storage**.

Table 1. *Do popular LLMs (FP8) fit in GPU memory?*

GPU model (Nvidia)	A100 (40/80 GiB)	H100 (80 GiB)	L40S (48 GiB)	H200 (141 GiB)
Granite3.1-8B [6]	✓	✓	✓	✓
Mistral-Large [8]	✗	✗	✗	✓
GPT3-175B [20]	✗	✗	✗	✗
OPT-175B [5, 60]	✗	✗	✗	✗
Llama3-405B [7]	✗	✗	✗	✗

Keywords: Large language model, Model offloading, KV cache offloading, SSDs

ACM Reference Format:
Zebin Ren, Krijn Doekemeijer, Tiziano De Matteis, Christian Pinto, Radu Stoica, and Animesh Trivedi. 2025. An I/O Characterizing Study of Offloading LLM Models and KV Caches to NVMe SSD. In *5th Workshop on Challenges and Opportunities of Efficient and Performant Storage Systems (CHEOPS '25), March 30-April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3719330.3721230

1 Introduction

With the popularity of generative AI, Large Language Model (LLM) inference has become one of the most popular cloud workloads. Modern LLMs (e.g., IBM Granite, Meta Llama, OpenAI GPT-4, Google Gemini) are trained using trillions of tokens and contain hundreds of billions of parameters [6, 7, 15, 45]. This increase in model size leads to a higher quality of generated text and sampling efficiency [16, 34]. At the same time, there is an increased demand for long-context LLMs that support context windows up to 1M tokens for complex applications such as repository-level code understanding [18, 32] and long-document processing [42, 57]. As a result, the memory requirement of LLM inference has grown significantly over the past few years [22]. The speed at which the memory requirement increases is much faster than the speed at which GPU memory sizes increase, thus approaching and, in some cases, exceeding the memory sizes found in typical GPUs (see Tab. 1).

Multiple solutions have been proposed to address this memory pressure, such as quantization [23, 24, 27], leveraging model sparsity [13, 53], sharing of computational and model state [29, 59], compression [43], model parallelism [52],

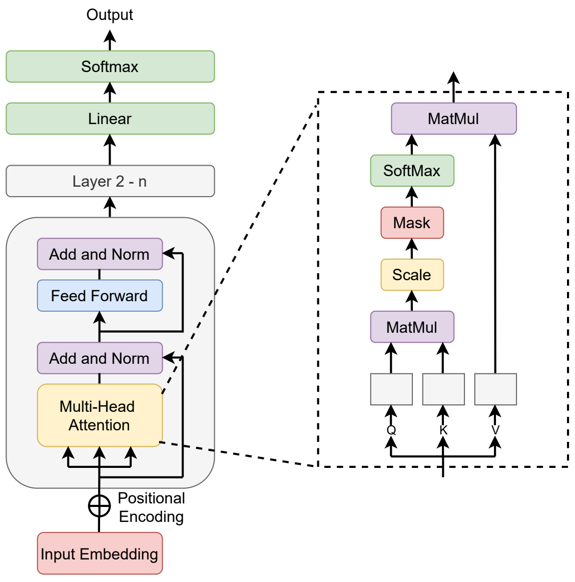

Figure 1. *State-of-the-practice LLM transformer architecture*

and memory offloading [11, 13, 51]. GPU memory offloading techniques are the focus of this work. In GPU memory offloading, an LLM inference serving framework can offload the model weights and KV cache to CPU memory or to storage devices during the LLM inference process[1]. As long-context LLMs are becoming common, the KV cache size has increased significantly [12, 57], and these KV caches may require long-term storing in a multi-turn conversations to restore the context so that the KV cache can be reused for future incoming requests [29].

The current focus of the community is largely on CPU offloading. However, recent interest has been in the integration of external storage to support emerging large models that can approach 100–1,000s GiB, (see Tab. 1) that can be stored more efficiently (i.e., cost, energy) on SSDs. Integrating external NVMe SSDs further enables various deployment and optimization opportunities, including sharing, optimizing, and managing LLM context states among multiple LLM inference requests [29, 48, 59]. Despite this, there is a lack of systematic study on the storage usage and the performance requirements of SSDs when offloading the GPU memory usage to SSDs during inference. This is the focus of this work.

The two main components for offloading during LLM inference are (i) the model weights (i.e., model offloading) and (ii) the key-value cache (i.e., KV cache offloading). The size of the memory consumed by the model depends on the size of the model and the quantization used. Due to the auto-regressive nature of the modern transformer architecture used in LLMs [56], during the iterative generation of output tokens, the LLM serving framework typically caches the keys and values of previously generated tokens (KV cache). The size of the KV cache depends on multiple factors, including the input and output token lengths, batch size, and the number of attention heads and layers. This work aims to characterize the SSD usage (access patterns and performance) of the model and KV cache offloading process during the LLM inference process. We use two state-of-the-art LLM inference frameworks, DeepSpeed Zero inference [14] and FlexGen [51], for this study with the OPT models [60] since both frameworks natively support the OPT models. We conduct our study on a GCP g2-standard-32 instance with the NVMeVirt framework [37] to emulate a fast NVMe device in memory. In §3, we give a more detailed description and motivation of our setup. During the serving process, when GPU memory is offloaded to the SSD, we collect block-level I/O traces, characterize these I/O traces, and report four main findings. Based on this analysis, we discuss the role the storage devices can play in supporting LLM inference during the offloading process.

Our key contributions in this work are:

- Performance benchmarking of DeepNVMe, an I/O library for tensor transfer between CPU/GPU memory and SSDs with two I/O interfaces, `POSIX` and `libaio.`
- Collect and characterize block-level SSD traces when the LLM inference frameworks offload the LLM models and KV caches to SSDs.
- To facilitate reproduction, we open-source the design and implementation of our code and traces at https://github.com/stonet-research/cheops25-IO-characterization-of-LLM-model-kv-cache-offloading-nvme.

2 Background on LLM Inference

In this section, we provide background on LLM inference, KV cache, and common LLM offloading techniques.

2.1 Large Language Model Inference

In this work, we focus on transformer-based LLMs. We visualize such an LLM design in Fig. 1. Transformers are composed of a series of transformer blocks. The input to each block first passes through the self-attention layer. In this self-attention layer, each input token is multiplied by a weight matrix to get the Q (query), K (key), and V (value) vectors. Then, the self-attention mechanism generates the embedding vectors, which captures the relationships between the input tokens. In order to extract multiple features, it is a common practice to use multi-head attention in a layer, where each head extracts different features. Then, the embedding vectors of different attention heads are concatenated and passed to the feed-forwarded layer (FFL).

[1]This work focuses on memory offloading during the LLM *inference* operation, not during model *training*. See §8 for related work discussion.

Table 2. *Details of the GCP benchmarking environment*

Component	***Configuration details***
GCP Instance	g2-standard-32
CPU	Intel(R) Xeon(R) @ 2.20GHz, 32 cores
Memory	128 GiB
GPU	NVIDIA L4, 24 GiB memory
Software	Ubuntu 22.04.1, kernel `6.8.0-1020-gcp`, NVMeVirt (commit 17f6f4d), FlexLLMGen 0.1.7, DeepSpeed 0.16.1

The LLM inference process contains two stages that always happen in sequence: the prefill stage and the decode stage. During the prefill stage, the input tokens are passed to the model. The model then generates the KV cache and the first output token. During the decode stage, the LLM generates the output tokens in an auto-regressive manner. In each pass of the model, the model takes the input tokens and all previously generated output tokens to generate the next output token (thus, the input length increases with each pass). The generation of new tokens stops when it reaches a predefined maximum generation length or the end-of-sequence (EOS) token. Multiple queries are usually batched to achieve high GPU utilization in multi-tenant environments, where multiple inferences happen concurrently.

2.2 Key-value Cache

In LLM inference, each output token depends on all the inputs and previously generated output tokens (§2.1). By default, when generating a new output token, all key-value embeddings of these inputs and previously generated output tokens are regenerated, leading to repeated computation. Therefore, a common optimization is to cache the previously generated key and value embeddings (i.e., input and output tokens) to prevent regeneration. This caching is referred to as KV cache and typically resides in the GPU memory and becomes a large part of the GPU memory consumption during the LLM inference process [57]. Offloading is essential to reduce memory requirements since all key and value embeddings of input and previously generated output tokens are cached, i.e., the total KV cache size increases linearly with the context window. Specifically, long-context LLMs, which currently support up to a 1M content window, require a large KV cache that can be too large for the GPU [57]. For example, a recent work reports that the LLaMa-65B model with 4× A100 GPUs can generate KV cache entries at 13.9 GiB/s, which exhausts the memory of any modern GPU in tens of seconds [29].

2.3 Memory and SSD Offloading

One solution to reduce the pressure on the GPU memory of LLM inference is to offload the model and KV cache to CPU memory and/or storage devices such as SSDs. Both the model and KV cache can be configured to be offloaded separately where the nature, granularity, and requirements of the offloading depend on the inference framework used, such as HuggingFace accelerate [4], DeepSpeed [14], and FlexGen [51]. The core idea behind offloading is that due to the iterative nature of computation in LLM serving, the required context (model or KV cache) can be fetched from host memory or storage devices as needed, thus lowering the memory requirements of GPUs as the GPUs do not have to hold *all* the model and KV cache data in GPU memory *all the time* during the LLM inference process.

3 Experimental Setup

Hardware and Software Environment: Tab. 2 shows our experimental setup. We use a g2-standard-32 GCP instance for this characterization study. Considering that the goal of our characterization is to study storage I/O patterns and identify framework capabilities without artificially being impacted by the slow(er) storage that comes with the GCP instance, we leverage the `NVMeVirt` framework to emulate a fast NVMe SSD in memory [37, 44]. We emulated a 96 GiB SSD (NVM model) (64 GiB for DeepSpeed) in DRAM and dedicated four isolated CPU cores to process I/O requests. In our microbenchmarks with fio [10], an NVMeVirt NVM SSD can support 9.3 μs of access latency (4 KiB request), 2.6 million IOPS (4 KiB request) with 16 threads, and a bandwidth of 5.3 GiB/s with single thread issuing 512 KiB requests which can be scaled up to 16.9 GiB/s with four threads. With this performance profile, we ensure that the speed of the NVMe SSD is not the bottleneck when gathering and characterizing I/O traces during the offloading process. We use this device for our characterization study in the remainder of the paper.
Selection of LLM Inference Frameworks: For our evaluation, we require the LLM inference frameworks to support SSD offloading. Model and KV cache offloading to CPU memory are widely supported by popular frameworks such as vLLM, DeepSpeed, and FlexGen [14, 38, 51]. However, SSD offloading has only recently gained attention [29] and is selectively available in open-source frameworks. In our investigation, we evaluate the DeepSpeed [3, 11] and FlexGen [2, 51] frameworks. DeepSpeed supports model offloading to SSDs for inference [14]. However, from the repositories examples [11], we infer that DeepSpeed only supports model offloading, not KV cache offloading to SSDs. FlexGen supports both model and KV cache offloading to SSDs. There are also other LLM inference frameworks but we do not use them for various reasons. For example, the recently published AttentionStore [29] also supports offloading KV caches to SSDs, but their code is not publicly available. Alternate to these frameworks that are recently proposed optimizations like the disaggregated decoding-prefill architecture [61] and LMCache [21, 43, 59] that externalize the KV cache from inference frameworks for distributed or GPU-parallel setups. In this work, we focus on a single machine, single GPU case,

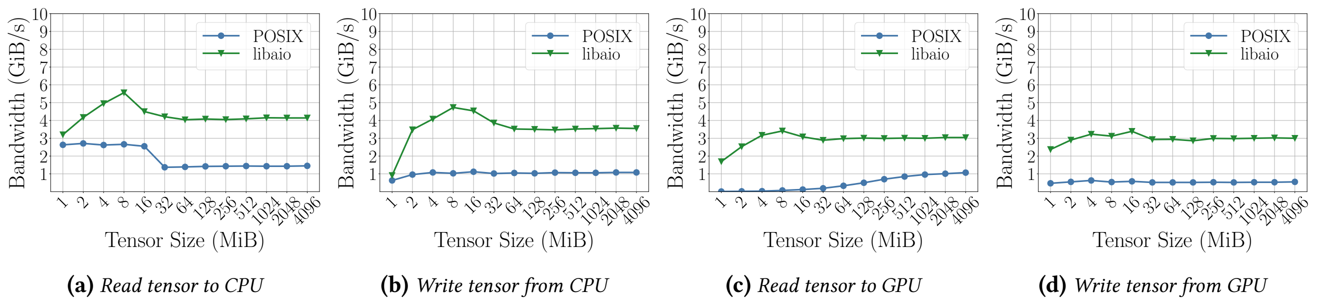

Figure 2. *Tensor offloading — Average bandwidth when transferring tensors between CPU and SSD (a–b); and between GPU and SSD (c–d)*

not a distributed setting. For these reasons, we use DeepSpeed for the model offloading and FlexGen for the model and KV cache offloading experiments.

Models and setup: We use OPT models for evaluation, as they are supported by both DeepSpeed and FlexGen [60] natively with FP16 quantization (a commonly used quantization). We use the largest model supported by the hardware for model offloading, which is OPT-13B for DeepSpeed inference and OPT-30B for FlexGen. We use dummy models since our experiments do not compare the accuracy of the models, where the dummy models are generated by the DeepSpeed examples and FlexGen framework [2, 11]. For model offloading, we also set the number of input tokens to 256 and the output tokens to 32. We have verified that neither the number of input nor output tokens affects the decode performance. For DeepSpeed, we use 64 GiB of the main memory for NVMeVirt SSD and 96 GiB for FlexGen. For KV cache offloading, we disable model offloading. Thus, we use OPT-6.7B since this is the largest model that fits completely into the GPU memory (24 GiB). We set the number of input and output tokens to 256 for KV cache offloading to show the I/O workload with a longer context than model offloading.

In Appendix A, we show all the commands that we run in our evaluations for this paper. A more complete set of scripts and our trace datasets are available in our artifact repository.

4 Transferring Tensors with `DeepNVMe`

Before we do a framework-level evaluation, we evaluate the performance of tensor transfer between the SSD and the CPU/GPU memory since this tensor transfer is used during the model and KV cache offloading. The DeepSpeed framework has an I/O library (`DeepNVMe`) for transferring tensors between the SSD storage devices and CPU or GPU memory [1]. We evaluate the bandwidth of transferring tensors between CPU–SSD and GPU–SSD with the `DeepNVMe` library. The I/O bandwidth for the tensor transferring allows us to understand how fast we can transfer model and KV cache tensors directly from and to the storage devices. Specifically, in this section, we investigate **Question #1: What is the tensor transferring bandwidth in `DeepNVMe` between the CPU/GPU memory and the SSDs, and what does this bandwidth depend on?**

To answer this question, we evaluate the impact of tensor size, I/O engines, and offloading methods on the bandwidth of transferring tensors with `DeepNVMe`. The tensor size is the total transfer size during offloading. The `DeepNVMe` library offers two different I/O engines: a synchronous `POSIX` API, and a high-performance asynchronous API based on `libaio`. Offloading tensors from the GPU is expected to lead to lower bandwidth than CPU offloading because the data is transferred via the CPU memory. DeepNVMe's experiments use single-threaded I/O, and we run each configuration five times and report both the average bandwidth and the standard deviation. We plot our results in Fig. 2, which shows the I/O bandwidth (y-axis, higher is better) with an increasing tensor size (x-axis). We have two observations:

1. `libaio` based asynchronous tensor offloading delivers higher tensor transfer bandwidth than `POSIX` for all evaluated scenarios. For example, offloading a 4 GiB tensor with `libaio` delivers up to 2.9×, 3.3×, 2.8× and 5.5× higher bandwidth than `POSIX` for reading tensors to CPU, writing tensors from CPU, reading tensors to GPU, and writing tensors from GPU respectively;
2. Neither `POSIX` nor `libaio` can achieve the single thread maximum bandwidth (maximum 4.1 GiB/s when reading tensors to CPU with `libaio`) shown in our baseline performance (5.3 GiB/s) of NVMeVirt SSD with fio.

Answer #1: `DeepNVMe` *provides higher tensor transferring bandwidth from/to SSDs with asynchronous* `libaio` *than with the synchronous* `POSIX` *interface, up to 5.5× bandwidth, but it still has a lower maximum bandwidth than the per-thread bandwidth achieved with fio.* `libaio` *achieves up to 5.5× bandwidth than* `POSIX`. *However, the maximum single-thread bandwidth achieved by* `DeepNVMe` *is still 22.6% lower than the baseline performance of the NVMeVirt SSD with fio.*

5 Model Offloading

In this section, we characterize the I/O access patterns and performance when offloading the models to the NVMeVirt

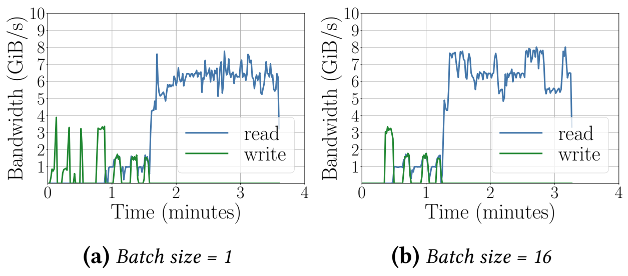

Figure 3. *Model offloading with DeepSpeed — SSD I/O bandwidth over time for* ***OPT-13B***

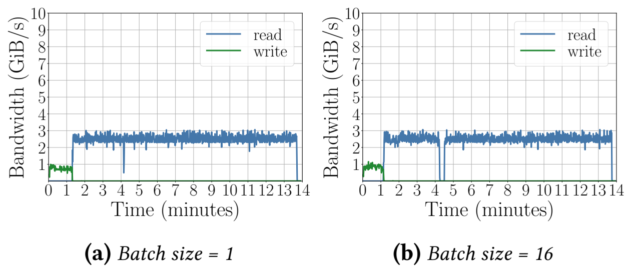

Figure 4. *Model offloading with FlexGen – SSD I/O bandwidth over time for* ***OPT-30B***

SSD with the DeepSpeed and FlexGen frameworks. We evaluate both frameworks with the largest OPT models they support out-of-the-box, which are OPT-13B for DeepSpeed and OPT-30B for FlexGen, respectively.

Question #2: What are the I/O access patterns of model offloading? To study the I/O access patterns that the frameworks request to the underlying SSD, we collect block-level I/O traces concurrently with our LLM inference workload using the `bpftrace` tool. With `bpftrace`, we probe using the `block_rq_issue` tracepoint and collect the I/O operation's type (i.e., read or write), request size, and accessed sectors. In Fig. 3 and Fig. 4, we show the read- and write bandwidth (y-axis) during inference over time (x-axis). In the plot, the bandwidth is the per-second aggregated bandwidth (to reduce plotting noise). We have the following observations:

1. When the SSD model offloading is enabled, we observe that all the writes occur at the start of the experiment. When the number of writes reduces and approximates zero, we observe an increase in reads from the SSD. We hypothesize that the number of writes reduces because the model data is not updated during the inference; hence, the model that is offloaded to the SSD does not need to be (re)written after the initial offloading operation.
2. The dominant request size for the block-level reads and writes is 128 KiB requests (not plotted). Neither DeepSpeed nor FlexGen provides any configuration parameter to tune the request size issued to the file system (and to the block layer).
3. The SSD's sectors are uniformly accessed by both frameworks, indicating no hot spot in the I/O access pattern (not plotted).
4. We report similar I/O access patterns (i.e., bandwidth, access ratio) across the `ext4` and `xfs` file systems (not plotted). Thus, in our experiments, the choice of file system has a minimal impact on the I/O access patterns.

Answer #2: *The I/O workload of both DeepSpeed and FlexGen inference with model offloading is dominated by 128 KiB reads that access a set of sectors uniformly. We only observe writes at the start of the offloading.*

Question #3: Can FlexGen or DeepSpeed saturate the performance of an NVMe SSD with model offloading? To answer this question, we compute the average read bandwidth of DeepSpeed and FlexGen during the experiments. The average read bandwidth achieved by DeepSpeed and FlexGen is 4.9 GiB/s and 2.6 GiB/s, respectively, which is significantly less than the SSD's saturation point.

Answer #3: *Although DeepSpeed achieves 63.3% higher bandwidth than the maximum read tensor bandwidth to CPU in the previous section (3.0 GiB/s), neither of them achieves the maximum per-thread bandwidth of the NVMeVirt SSD with fio (5.3 GiB/s).*

6 KV Cache Offloading with FlexGen

In this section, we characterize the I/O access patterns and performance when offloading the KV cache to the NVMeVirt SSD with FlexGen. We disable model offloading to ensure all SSD I/O is due to the KV cache offloading operations. In order to fit the model in the GPU memory while leaving space for the KV cache, we use a smaller model (OPT-6.7B) than the models used in §5. Further on, to simulate long-context scenarios (i.e., a typical use-case for KV-caching), we set the number of input tokens to 256 and the output tokens to 256 per request and use a batch size of 64.

Question #4: What are the I/O access patterns of KV cache offloading? In Fig. 5, we show the read and write bandwidth (y-axis) during the inference workload. Unlike model offloading, we observe that the KV cache offloading has a slightly lower read- (2.0 GiB/s) and write (11.0 MiB/s) bandwidth than the model offloading. Furthermore, the read bandwidth of the KV cache is significantly higher (186.2×) than the write bandwidth because the KV cache is written once but read multiple times during the inference process. Further on, we observe that the dominant request size in the block layer is 128 KiB for both reads and writes. Lastly, we observe that the KV cache I/O access pattern to the SSD is not uniform. In Fig. 6, we show the CDF of how many times each used sector (i.e., any sector read more than once) is read during the inference workload. In this CDF plot, the y-axis presents the percentage of requests that are accessed at least as many times as the value on the x-axis. We observe the

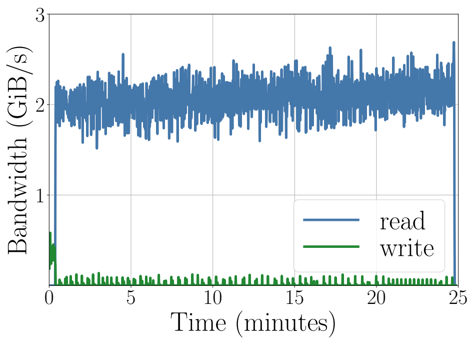

Figure 5. *KV Cache offloading with FlexGen – SSD I/O bandwidth over time for **OPT-6.7B** (batch size = 64). The y-axis is up to 3 GiB/s.*

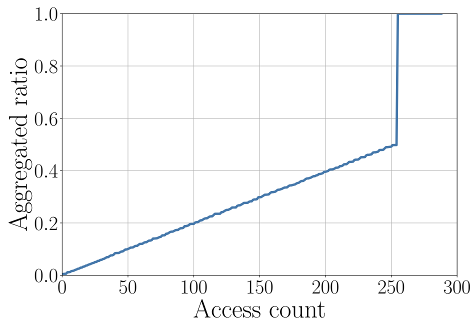

Figure 6. *KV Cache offloading with FlexGen – CDF of how many times each used sector is read **OPT-6.7B** (batch size = 64)*

ratio to increase linearly with the access count until a peak of around 256 times. This peak is expected because the KV cache for the input tokens is accessed once each time a new token is generated, and the total number of output tokens is 256. Each pass generates the KV cache for the new input token, and all the following token generations access this previously generated KV cache.

Answer #4: *In our evaluation, FlexGen KV cache offload operation has much higher read bandwidth requirements than write bandwidth (up to 186.2×). Additionally, both reads and writes are dominated by 128 KiB requests. The sectors in KV cache offloading are accessed non-uniformly.*

7 Discussion

Are Modern NVMe SSDs Ready for LLM Inference Workloads? We have taken the first step in this direction by characterizing SSD access patterns during the model and KV cache offloading operations. Modern high-speed SSDs such as a Samsung 990 Pro can deliver close to 6–7 GiB/s bandwidth (sequential, PCIe 4.0, m.2) [9], which matches closely to what we observed DeepSpeed can leverage. However, we are aware of the fact a more comprehensive investigation is required to explore the complete spectrum. For example, we are planning to explore the impact of more powerful GPUs than L4 (e.g., A100 and H100), which have significantly higher decoding speeds, thus requiring faster access to the offloaded model and KV caches. Additionally, in a multi-GPU and multi-SSD setup, the impact of NUMA, GPU-SSD affinities, I/O scheduling, CPU architecture (e.g., PCIe complex, lane sharing, and oversubscription), data transfer mechanism (GPUDirect vs. CPU coordinated), etc. can play a non-trivial role.

What role can SSD manufacturers play here? SSD internals are known to be complex [41] leading to various internal flash translation layer (FTL) implementations and designs. A big part of this complexity comes from offering the conventional read-anywhere, write-anywhere-and-anytime I/O interface on top of write-once, read-many, and erase flash NAND chips. As we explore in this work, model, and KV cache, offloading workloads (i) use large I/O sizes (i.e., 128 KiB) and (ii) are dominated by reads; hence, there is an opportunity to optimize the storage devices for this emerging important class of workload. In this context, NVMe interfaces such as ZNS and FDP, which give more control over data placement, QoS isolation, and parallelism management, are of special interest [19, 25, 26, 33].

KV Cache Management in LLM: KV cache management has emerged as a first-class problem for LLM inference workloads [40]. The capability to externalize and offload the KV cache opens up further opportunities regarding concepts such as compression/encoding [43], efficient scheduling via disaggregated prefill-decoding architectures [48], caching and blending [59], and streaming with fault tolerance [54]. We believe that access to a fast, efficient, and optimized storage (local and/or distributed) will play an important role regarding when and where LLM states can be (re)stored for efficient distributed request serving.

8 Related Work

Model Training and NVMe: While this paper focuses on model and KV cache offloading during inference, SSDs are also widely used in model training. For example, various works use SSDs in model training to overcome the memory wall [17, 36, 55]. BLAS-on-flash [55] is a library for flash-based matrix operations when the working set can not fit into the GPU memory. FlashNeuron [17] offloads the intermediate data to SSDs to increase the batch size, which enables higher GPU utilization. Behemoth [36] is a flash-centric DNN training system that uses NAND flash chips to replace the conventional HBM in conventional accelerators. DeepSpeed ZeRO-Offload [50] provides three stages that support offloading optimizer states, gradients, and parameters from the GPU memory. Our work focuses on SSD offloading with inference. The offloading mechanism in DeepSpeed can be used with inference. Thus, we select DeepSpeed as one of our evaluated frameworks.

Model and KV cache offloading for LLM inference: Model and KV cache offloading is widely used in LLM inference to reduce GPU memory usage. Below, we detail the state-of-the-practice and the state-of-the-art studies for such offloading. HuggingFace accelerate [4] allows initializing empty weights as placeholders and fetching the weights into GPU memory on demand. DeepSpeed [14, 49] supports offloading the model to the CPU or storage devices. CachedAttention [29] caches the KV cache in multi-turn conversations for reuse. These offloading frameworks offload the full model or KV cache without considering the sparsity of the model weights or neuron activations.

Many frameworks exploit the sparsity of models, activations, or the different importance of different tokens to reduce the traffic in offloading. LLM-in-a-flash [13] focuses on mobile devices and exploits the sparsity of ReLU in the FFN layer to reduce traffic between GPU and storage. PowerInfer [53] also exploits the sparsity of activations but only supports CPU offloading. FlexGen [51] is designed for latency-insensitive applications by batching and offloading to run LLM inference with limited memory. In our experiments, we take the simplest offloading strategy, which is to offload the whole model and KV cache instead of utilizing sparsity to reduce the I/O traffic. InfiniGen [39] offloads the KV cache for long-context generations and exploits the different importance of tokens to reduce the traffic. There are also studies that use in-storage processing in addition to offloading [35, 46, 58], but these devices are less readily available than traditional SSDs, which limits their usage.

LLM serving: LLM serving is designed to serve the LLMs in real workloads where various kinds of queries constantly come to the serving system, and the service providers usually need to achieve throughput and latency guarantees. vLLM [38] proposes PagedAttention to avoid wasting GPU memory for pre-allocating the KV cache for future generated tokens in GPU memory. ServerlessLLM [28] and DeepSpeed-FastGen [30] split the input into chunks. LoongServe [57] focus on long-context LLMs.

A recent trend is to separate the prefill and decode stages and distribute their computation across servers since they have different computational characteristics [31, 47, 48, 54, 61]. These methods require migrating the KV cache from the prefilled servers to the decode servers and are orthogonal to our single-server approach. Mooncake [48] proposes a KV cache-centric serving framework that balances throughput and latency. Our work focuses on the most straightforward case for inference and does not take this complexity into the evaluation.

9 Conclusion

In this paper, we investigate the I/O characteristics of offloading LLM models and KV caches to SSDs during LLM inference using two state-of-the-art LLM inference frameworks: DeepSpeed and FlexGen. Our results show that in a straightforward offloading setup (i.e., single CPU, single workload) (1) the I/O workload in LLM inference with model offloading using both DeepSpeed and FlexGen is dominated by 128 KiB reads, (2) in our setup, model offloading does not saturate NVMeVirt devices which can provide up to 16.9 GiB/s of bandwidth and (3) the I/O workload of KV cache offloading contains both read and write requests dominated by 128 KiB, but the read bandwidth is higher than the write bandwidth (2.0 GiB/s vs. 11.0 MiB/s). Though these findings are not surprising in itself, with this work we have taken the first step in analyzing the characteristics of I/O workload for SSD-based GPU memory offloading in LLM inference and made the I/O traces that we collected for the analysis publicly available. Based on these findings, we presented a discussion (along with opportunities) on SSD designs and KV cache management.

Acknowledgments

We thank the CHEOPS'25 reviewers for their invaluable and constructive feedback. This work is funded by The Dutch Research Council (NWO) grant number OCENW.KLEIN.561, Netherlands-funded projects NWO OffSense and GFP 6G FNS, and EU-funded projects MCSA-RISE Cloudstars and Horizon Graph-Massivizer. Krijn Doekemeijer is funded by the VU PhD innovation program. The authors would also like to thank the AtLarge group at Vrije Universiteit Amsterdam for their help with the paper.

References

[1] Accessed: 2025-02-26. DeepSpeed DeepNVMe. https://www.deepspeed.ai/tutorials/deepnvme/.
[2] Accessed: 2025-02-26. FMInference/FlexLLMGen. https://github.com/FMInference/FlexLLMGen.
[3] Accessed: 2025-02-26. GitHub DeepSpeed. https://github.com/microsoft/DeepSpeed.
[4] Accessed: 2025-02-26. HuggingFace Accelerate. https://huggingface.co/docs/accelerate/index.
[5] Accessed: 2025-02-26. HuggingFace: OPT model. https://huggingface.co/docs/transformers/en/model_doc/opt.
[6] Accessed: 2025-02-26. IBM Granite 3.1: Powerful Performance, Longer Context, New Embedding Models and More. https://www.ibm.com/new/announcements/ibm-granite-3-1-powerful-performance-long-context-and-more.
[7] Accessed: 2025-02-26. Introducing Meta Llama 3: The Most Capable Openly Available LLM to Date. https://ai.meta.com/blog/meta-llama-3/?utm_source=chatgpt.com.
[8] Accessed: 2025-02-26. Mistral: Model Weights. https://docs.mistral.ai/getting-started/models/weights/.
[9] Accessed: 2025-02-26. Samsung 990 pro. https://semiconductor.samsung.com/consumer-storage/internal-ssd/990-pro/.
[10] Accessed: 2025-02-26. Welcome to FIO's documentation! https://fio.readthedocs.io/en/latest/.
[11] Accessed: 2025-02-26. 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.

[12] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, Ada Gavrilovska and Douglas B. Terry (Eds.). USENIX Association, 117–134. https://www.usenix.org/conference/osdi24/presentation/agrawal

[13] Keivan Alizadeh, Seyed-Iman Mirzadeh, Dmitry Belenko, S. Khatamifard, Minsik Cho, Carlo C. del Mundo, Mohammad Rastegari, and Mehrdad Farajtabar. 2024. LLM in a Flash: Efficient Large Language Model Inference with Limited Memory. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 12562–12584. https://doi.org/10.18653/V1/2024.ACL-LONG.678

[14] Reza Yazdani Aminabadi, Samyam Rajbhandari, Ammar Ahmad Awan, Cheng Li, Du Li, Elton Zheng, Olatunji Ruwase, Shaden Smith, Minjia Zhang, Jeff Rasley, and Yuxiong He. 2022. DeepSpeed-Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis, Dallas, TX, USA, November 13-18, 2022*, Felix Wolf, Sameer Shende, Candace Culhane, Sadaf R. Alam, and Heike Jagode (Eds.). IEEE, 46:1–46:15. https://doi.org/10.1109/SC41404.2022.00051

[15] Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Slav Petrov, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy P. Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy, Michael Isard, Paul Ronald Barham, Tom Hennigan, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, Ryan Doherty, Eli Collins, Clemens Meyer, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, George Tucker, Enrique Piqueras, Maxim Krikun, Iain Barr, Nikolay Savinov, Ivo Danihelka, Becca Roelofs, Anaïs White, Anders Andreassen, Tamara von Glehn, Lakshman Yagati, Mehran Kazemi, Lucas Gonzalez, Misha Khalman, Jakub Sygnowski, and et al. 2023. Gemini: A Family of Highly Capable Multimodal Models. *CoRR* abs/2312.11805 (2023). https://doi.org/10.48550/ARXIV.2312.11805 arXiv:2312.11805

[16] Sher Badshah and Hassan Sajjad. 2024. Quantifying the Capabilities of LLMs Across Scale and Precision. *CoRR* abs/2405.03146 (2024). https://doi.org/10.48550/ARXIV.2405.03146 arXiv:2405.03146

[17] Jonghyun Bae, Jongsung Lee, Yunho Jin, Sam Son, Shine Kim, Hakbeom Jang, Tae Jun Ham, and Jae W. Lee. 2021. FlashNeuron: SSD-Enabled Large-Batch Training of Very Deep Neural Networks. In *19th USENIX Conference on File and Storage Technologies, FAST 2021, February 23-25, 2021*, Marcos K. Aguilera and Gala Yadgar (Eds.). USENIX Association, 387–401. https://www.usenix.org/conference/fast21/presentation/bae

[18] Ramakrishna Bairi, Atharv Sonwane, Aditya Kanade, Vageesh D. C., Arun Iyer, Suresh Parthasarathy, Sriram K. Rajamani, Balasubramanyan Ashok, and Shashank Shet. 2024. CodePlan: Repository-Level Coding using LLMs and Planning. *Proc. ACM Softw. Eng.* 1, FSE (2024), 675–698. https://doi.org/10.1145/3643757

[19] Matias Bjørling, Abutalib Aghayev, Hans Holmberg, Aravind Ramesh, Damien Le Moal, Gregory R. Ganger, and George Amvrosiadis. 2021. ZNS: Avoiding the Block Interface Tax for Flash-based SSDs. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, 689–703. https://www.usenix.org/conference/atc21/presentation/bjorling

[20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[21] Yihua Cheng, Kuntai Du, Jiayi Yao, and Junchen Jiang. 2024. Do Large Language Models Need a Content Delivery Network? *CoRR* abs/2409.13761 (2024). https://doi.org/10.48550/ARXIV.2409.13761 arXiv:2409.13761

[22] Krishna Teja Chitty-Venkata, Sparsh Mittal, Murali Emani, Venkatram Vishwanath, and Arun K. Somani. 2023. A Survey of Techniques for Optimizing Transformer Inference. *J. Syst. Archit.* 144 (2023), 102990. https://doi.org/10.1016/J.SYSARC.2023.102990

[23] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. GPT3.int8(): 8-bit Matrix Multiplication for Transformers at Scale. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (Eds.). http://papers.nips.cc/paper_files/paper/2022/hash/c3ba4962c05c49636d4c6206a97e9c8a-Abstract-Conference.html

[24] Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, Denis Kuznedelev, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, and Dan Alistarh. 2024. SpQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=Q1u25ahSuy

[25] Krijn Doekemeijer, Dennis Maisenbacher, Zebin Ren, Nick Tehrany, Matias Bjørling, and Animesh Trivedi. 2024. Exploring I/O Management Performance in ZNS with ConfZNS++. In *Proceedings of the 17th ACM International Systems and Storage Conference* (Virtual, Israel) *(SYSTOR '24)*. Association for Computing Machinery, New York, NY, USA, 162–177. https://doi.org/10.1145/3688351.3689160

[26] Krijn Doekemeijer, Nick Tehrany, Balakrishnan Chandrasekaran, Matias Bjørling, and Animesh Trivedi. 2023. Performance Characterization of NVMe Flash Devices with Zoned Namespaces (ZNS). In *2023 IEEE International Conference on Cluster Computing (CLUSTER)*. 118–131. https://doi.org/10.1109/CLUSTER52292.2023.00018

[27] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. OPTQ: Accurate Quantization for Generative Pre-trained Transformers. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net. https://openreview.net/forum?id=tcbBPnfwxS

[28] Yao Fu, Leyang Xue, Yeqi Huang, Andrei-Octavian Brabete, Dmitrii Ustiugov, Yuvraj Patel, and Luo Mai. 2024. ServerlessLLM: Low-Latency Serverless Inference for Large Language Models. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, Ada Gavrilovska and Douglas B. Terry (Eds.). USENIX Association, 135–153. https://www.usenix.org/conference/osdi24/presentation/fu

[29] Bin Gao, Zhuomin He, Puru Sharma, Qingxuan Kang, Djordje Jevdjic, Junbo Deng, Xingkun Yang, Zhou Yu, and Pengfei Zuo. 2025. Cost-efficient Large Language Model Serving for Multi-turn Conversations with CachedAttention. In *Proceedings of the 2024 USENIX Conference on Usenix Annual Technical Conference* (Santa Clara, CA, USA) *(USENIX ATC'24)*. USENIX Association, USA, Article 7, 16 pages.

[30] Connor Holmes, Masahiro Tanaka, Michael Wyatt, Ammar Ahmad Awan, Jeff Rasley, Samyam Rajbhandari, Reza Yazdani Aminabadi, Heyang Qin, Arash Bakhtiari, Lev Kurilenko, and Yuxiong He. 2024. DeepSpeed-FastGen: High-throughput Text Generation for LLMs via MII and DeepSpeed-Inference. *CoRR* abs/2401.08671 (2024). https://doi.org/10.48550/ARXIV.2401.08671 arXiv:2401.08671

[31] Cunchen Hu, Heyang Huang, Liangliang Xu, Xusheng Chen, Jiang Xu, Shuang Chen, Hao Feng, Chenxi Wang, Sa Wang, Yungang Bao, Ninghui Sun, and Yizhou Shan. 2024. Inference without Interference: Disaggregate LLM Inference for Mixed Downstream Workloads. *CoRR* abs/2401.11181 (2024). https://doi.org/10.48550/ARXIV.2401.11181 arXiv:2401.11181

[32] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-world Github Issues?. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=VTF8yNQM66

[33] Kanchan Joshi, Anuj Gupta, Javier Gonzalez, Ankit Kumar, Krishna Kanth Reddy, Arun George, Simon Lund, and Jens Axboe. 2024. I/O Passthru: Upstreaming a Flexible and Efficient I/O Path in Linux. In *22nd USENIX Conference on File and Storage Technologies (FAST 24)*. USENIX Association, Santa Clara, CA, 107–121. https://www.usenix.org/conference/fast24/presentation/joshi

[34] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling Laws for Neural Language Models. *CoRR* abs/2001.08361 (2020). arXiv:2001.08361 https://arxiv.org/abs/2001.08361

[35] Junkyum Kim, Myeonggu Kang, Yunki Han, Yanggon Kim, and Lee-Sup Kim. 2023. OptimStore: In-Storage Optimization of Large Scale DNNs with On-Die Processing. In *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2023, Montreal, QC, Canada, February 25 - March 1, 2023*. IEEE, 611–623. https://doi.org/10.1109/HPCA56546.2023.10071024

[36] Shine Kim, Yunho Jin, Gina Sohn, Jonghyun Bae, Tae Jun Ham, and Jae W. Lee. 2021. Behemoth: A Flash-centric Training Accelerator for Extreme-scale DNNs. In *19th USENIX Conference on File and Storage Technologies, FAST 2021, February 23-25, 2021*, Marcos K. Aguilera and Gala Yadgar (Eds.). USENIX Association, 371–385. https://www.usenix.org/conference/fast21/presentation/kim

[37] Sang-Hoon Kim, Jaehoon Shim, Euidong Lee, Seongyeop Jeong, Ilkueon Kang, and Jin-Soo Kim. 2023. NVMeVirt: A Versatile Software-defined Virtual NVMe Device. In *21st USENIX Conference on File and Storage Technologies (FAST 23)*. USENIX Association, Santa Clara, CA, 379–394. https://www.usenix.org/conference/fast23/presentation/kim-sang-hoon

[38] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, Jason Flinn, Margo I. Seltzer, Peter Druschel, Antoine Kaufmann, and Jonathan Mace (Eds.). ACM, 611–626. https://doi.org/10.1145/3600006.3613165

[39] Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. 2024. InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, Ada Gavrilovska and Douglas B. Terry (Eds.). USENIX Association, 155–172. https://www.usenix.org/conference/osdi24/presentation/lee

[40] Haoyang Li, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole Hu, Wei Dong, Qing Li, and Lei Chen. 2025. A Survey on Large Language Model Acceleration based on KV Cache Management. arXiv:2412.19442 [cs.AI] https://arxiv.org/abs/2412.19442

[41] Nanqinqin Li, Mingzhe Hao, Huaicheng Li, Xing Lin, Tim Emami, and Haryadi S. Gunawi. 2022. Fantastic SSD Internals and How to Learn and Use Them. In *Proceedings of the 15th ACM International Conference on Systems and Storage* (Haifa, Israel) *(SYSTOR '22)*. Association for Computing Machinery, New York, NY, USA, 72–84. https://doi.org/10.1145/3534056.3534940

[42] Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. 2024. Long-context LLMs Struggle with Long In-context Learning. *CoRR* abs/2404.02060 (2024). https://doi.org/10.48550/ARXIV.2404.02060 arXiv:2404.02060

[43] Yuhan Liu, Hanchen Li, Yihua Cheng, Siddhant Ray, Yuyang Huang, Qizheng Zhang, Kuntai Du, Jiayi Yao, Shan Lu, Ganesh Ananthanarayanan, et al. 2024. Cachegen: KV Cache Compression and Streaming for Fast Large Language Model Serving. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 38–56.

[44] Microsoft. 2025. NVMeVirt: A Versatile Software-defined Virtual NVMe Device. https://github.com/snu-csl/nvmevirt.

[45] OpenAI. 2023. GPT-4 Technical Report. *CoRR* abs/2303.08774 (2023). https://doi.org/10.48550/ARXIV.2303.08774 arXiv:2303.08774

[46] Xiurui Pan, Endian Li, Qiao Li, Shengwen Liang, Yizhou Shan, Ke Zhou, Yingwei Luo, Xiaolin Wang, and Jie Zhang. 2024. InstInfer: In-Storage Attention Offloading for Cost-Effective Long-Context LLM Inference. *CoRR* abs/2409.04992 (2024). https://doi.org/10.48550/ARXIV.2409.04992 arXiv:2409.04992

[47] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *51st ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2024, Buenos Aires, Argentina, June 29 - July 3, 2024*. IEEE, 118–132. https://doi.org/10.1109/ISCA59077.2024.00019

[48] Ruoyu Qin, Zheming Li, Weiran He, Mingxing Zhang, Yongwei Wu, Weimin Zheng, and Xinran Xu. 2024. Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving. *CoRR* abs/2407.00079 (2024). https://doi.org/10.48550/ARXIV.2407.00079 arXiv:2407.00079

[49] Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. 2021. ZeRO-infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2021, St. Louis, Missouri, USA, November 14-19, 2021*, Bronis R. de Supinski, Mary W. Hall, and Todd Gamblin (Eds.). ACM, 59. https://doi.org/10.1145/3458817.3476205

[50] Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. 2021. ZeRO-Offload: Democratizing Billion-Scale Model Training. In *Proceedings of the 2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*, Irina Calciu and Geoff Kuenning (Eds.). USENIX Association, 551–564. https://www.usenix.org/conference/atc21/presentation/ren-jie

[51] Ying Sheng, Lianmin Zheng, Binhang Yuan, Zhuohan Li, Max Ryabinin, Beidi Chen, Percy Liang, Christopher Ré, Ion Stoica, and Ce Zhang. 2023. FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA (Proceedings of Machine Learning Research, Vol. 202)*, Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (Eds.). PMLR, 31094–31116. https://proceedings.mlr.press/v202/sheng23a.html

[52] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2019. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. *CoRR* abs/1909.08053 (2019). arXiv:1909.08053 http://arxiv.org/abs/

1909.08053

[53] Yixin Song, Zeyu Mi, Haotong Xie, and Haibo Chen. 2024. PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles, SOSP 2024, Austin, TX, USA, November 4-6, 2024*, Emmett Witchel, Christopher J. Rossbach, Andrea C. Arpaci-Dusseau, and Kimberly Keeton (Eds.). ACM, 590–606. https://doi.org/10.1145/3694715.3695964

[54] Foteini Strati, Sara McAllister, Amar Phanishayee, Jakub Tarnawski, and Ana Klimovic. 2024. DéjàVu: KV-cache Streaming for Fast, Fault-tolerant Generative LLM Serving. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net. https://openreview.net/forum?id=AbGbGZFYOD

[55] Suhas Jayaram Subramanya, Harsha Vardhan Simhadri, Srajan Garg, Anil Kag, and Venkatesh Balasubramanian. 2019. BLAS-on-flash: An Efficient Alternative for Large Scale ML Training and Inference?. In *16th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2019, Boston, MA, February 26-28, 2019*, Jay R. Lorch and Minlan Yu (Eds.). USENIX Association, 469–484. https://www.usenix.org/conference/nsdi19/presentation/subramanya

[56] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[57] Bingyang Wu, Shengyu Liu, Yinmin Zhong, Peng Sun, Xuanzhe Liu, and Xin Jin. 2024. LoongServe: Efficiently Serving Long-Context Large Language Models with Elastic Sequence Parallelism. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles, SOSP 2024, Austin, TX, USA, November 4-6, 2024*, Emmett Witchel, Christopher J. Rossbach, Andrea C. Arpaci-Dusseau, and Kimberly Keeton (Eds.). ACM, 640–654. https://doi.org/10.1145/3694715.3695948

[58] Chunhua Xiao, Shi Qiu, and Dandan Xu. 2022. Cop-Flash: Utilizing Hybrid Storage to Construct a Large, Efficient, and Durable Computational Storage for DNN Training. In *IEEE 15th International Conference on Cloud Computing, CLOUD 2022, Barcelona, Spain, July 10-16, 2022*, Claudio Agostino Ardagna, Nimanthi L. Atukorala, Rajkumar Buyya, Carl K. Chang, Rong N. Chang, Ernesto Damiani, Gargi Banerjee Dasgupta, Fabrizio Gagliardi, Christoph Hagleitner, Dejan S. Milojicic, Tuan M. Hoang Trong, Robert Ward, Fatos Xhafa, and Jia Zhang (Eds.). IEEE, 209–218. https://doi.org/10.1109/CLOUD55607.2022.00041

[59] Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. 2024. CacheBlend: Fast Large Language Model Serving with Cached Knowledge Fusion. *arXiv preprint arXiv:2405.16444* (2024).

[60] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: Open Pre-trained Transformer Language Models. *CoRR* abs/2205.01068 (2022). https://doi.org/10.48550/ARXIV.2205.01068 arXiv:2205.01068

[61] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024*, Ada Gavrilovska and Douglas B. Terry (Eds.). USENIX Association, 193–210. https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin

A Commands Used in Benchmarking

In this section, we show the exact command used in our experiments. We replace the exact path in the commands by [PATH]. For all the commands, we assume the DeepSpeed and FlexGen are installed in the environment.
DeepNVMe test load and store:

```
python py_store_cpu_tensor.py --
    nvme_folder [PATH] --mb_size 1 --
    loop 5
python py_load_cpu_tensor.py --
    input_file [PATH] --loop 5
```

bpftrace script, the arguments in the third line are the major and minor values of the disk's (SSD's) device file:

```
// check major and minor:
// ls -l /dev/nvme0n1
tracepoint:block:block_rq_issue
/args->dev == ((259 << 20) | 4)
{
    printf("%llu,%s,%d,%llu,%d\n", nsecs
        , args->rwbs, args->bytes, args
        ->sector, args->nr_sector);
}
```

DeepSpeed inference with model offloading, we use a dummy model for benchmarking.

```
deepspeed --num_gpus 1 run_model.py --
    dummy --model facebook/opt-13b --cpu
    -offload --disk-offload --offload-
    dir [PATH] --prompt-len 256 --gen-
    len 32 --loops 1 --batch-size 1
```

FlexGen inference with model offloading, we use a dummy model for benchmarking. We limit the size of the memory that FlexGen can use.

```
sudo systemd-run --scope --property=
    MemoryLimit=10G python3 -m
    flexllmgen.flex_opt --model facebook
    /opt-30b --path _DUMMY_ --offload-
    dir [PATH] --prompt-len 256 --gen-
    len 32 --num-gpu-batches 1 --percent
     0 0 100 0 100 0 --gpu-batch-size 1
```

FlexGen inference with KV cache offloading. We limit the size of the memory that FlexGen can use.

```
sudo systemd-run --scope --property=
    MemoryLimit=10G python3 -m
    flexllmgen.flex_opt --model facebook
    /opt-6.7b --path _DUMMY_ --offload-
    dir [PATH] --prompt-len 256 --gen-
    len 256 --num-gpu-batches 1 --
    percent 100 0 0 0 100 0 --gpu-batch-
    size 1
```