Improving the Performance of Out-of-Core LLM Inference Using Heterogeneous Host Memory

Sudhanshu Gupta
Department of Computer Science
University of Rochester
Rochester, NY, USA
sgupta45@cs.rochester.edu

Sandhya Dwarkadas
Department of Computer Science
University of Virginia
Charlottesville, VA, USA
sandhya@virginia.edu

***Abstract*—The memory footprint of modern applications like large language models (LLMs) far exceeds the memory capacity of accelerators they run on and often spills over to host memory. As model sizes continue to grow, DRAM-based memory is no longer sufficient to contain these models, resulting in further spill-over to storage and necessitating the use of technologies like Intel Optane and CXL-enabled memory expansion. While such technologies provide more capacity, their higher latency and lower bandwidth has given rise to heterogeneous memory configurations that attempt to strike a balance between capacity and performance. This paper evaluates the impact of such memory configurations on a GPU running out-of-core LLMs. Starting with basic host/device bandwidth measurements using an Optane and Nvidia A100 equipped NUMA system, we present a comprehensive performance analysis of serving OPT-30B and OPT-175B models using FlexGen, a state-of-the-art serving framework.**

Our characterization shows that FlexGen's weight placement algorithm is a key bottleneck limiting performance. Based on this observation, we evaluate two alternate weight placement strategies, one each optimizing for inference latency and throughput. When combined with model quantization, our strategies improve latency and throughput by 27% and 5x, respectively. These figures are within 9% and 6% of an all-DRAM system, demonstrating how careful data placement can effectively enable the substitution of DRAM with high-capacity but slower memory, improving overall system energy efficiency.

***Index Terms*—Compute express link, heterogeneous memory, GPU inference, large language models, non-volatile memory.**

I. Introduction

The memory footprint of modern applications like large language models (LLMs) continues to grow at a staggering rate, with model parameter size increasing by 410x every two years [1]. This growth in size and complexity has enabled the wide proliferation of AI-enabled applications, from chat bots [2] and coding assistants [3] to impacting "literature and medicine" [4].

While massively parallel processors like GPUs continue to underpin the development of LLMs, their compute capacity remains underutilized [1], [5], [6] with memory capacity emerging as the key bottleneck. The challenge of accommodating these models in memory brings a long known problem to the forefront, that of DRAM capacity scaling [7]. Emerging memory technologies like phase change memory (PCM) [8], resistive RAM (ReRAM) [9], and spin-transfer torque RAM (STT-RAM) [10]–[13] improve density compared to traditional DRAM while achieving varying degrees of performance parity. Concurrently, interconnect technologies like compute express link (CXL) [14] allow for technology-agnostic expansion of main memory capacity and direct access from accelerators like GPUs [15], [16]. The performance impact of these technologies on accelerator performance is of considerable importance but heavily understudied. This work aims to fill that gap, contributing to a broad category of solutions that attempt to transparently expand GPU memory capacity while hiding the associated performance costs.

In this paper, we characterize the performance impact of heterogeneous host memory on accelerator performance using an Intel Optane and Nvidia A100 equipped system. We first present basic bandwidth measurements when moving data between host and GPU under different host memory configurations, showing how host to GPU and GPU to host bandwidth drop by up to 41% and 92%, respectively, compared to traditional DRAM memory. Prior work has shown that CXL-expanded memory only achieves up to 47% and 30% of the theoretical maximum bandwidth of the underlying DRAM memory [17], while highlighting significant performance variations across CXL controller architectures and the underlying memory technology.

To understand the real world impact of this performance deficit, we evaluate the inference performance of the open pre-trained transformer (OPT) family of LLMs [18] under various memory configurations using FlexGen [19], a state-of-the-art LLM serving framework that supports distribution of model weights across GPU memory, host memory, and permanent storage. Our results show an average 33% increase in per-layer processing time for OPT-175B with Optane as main memory compared to DRAM main memory, a direct result of the lower Optane bandwidth and the memory bound nature of LLM inference. While compression helps reduce this memory bottleneck, a deeper analysis of FlexGen's compute schedule reveals an imbalance in the compute/communication pipeline as the root cause. This imbalance is a byproduct of its weight placement scheme.

We address the imbalance with two alternate weight placement schemes, one each optimizing for latency and throughput. The first scheme, called HeLM, allocates weights for

each layer in a compute time-aware fashion to improve the overlap of compute time of layer *i* with the weight transfer time of layer *i+1*. This allows HeLM to achieve a more balanced compute/communication pipeline, improving average time between tokens (TBT) by 27%. The second scheme, called All-CPU, offloads all weights to host memory and leaves GPU memory for key/value caches and hidden state. This boosts the maximum possible batch size from 8 to 44 and brings about a 5x improvement in throughput. HeLM's TBT and All-CPU's throughput come within 9% and 6% of an all-DRAM system, highlighting how careful data placement can help hide the performance deficiencies of emerging memory technologies.

In summary, this paper makes the following contributions:

1) Quantification of host/device data movement performance with Intel Optane, a high capacity but low performance byte-addressable memory, as host memory, showing significantly lower bandwidth compared to traditional DRAM.
2) Characterization of LLM performance on a real system when using such memory, pointing at inefficient weight placement as a performance bottleneck.
3) Evaluation of two model weight placement schemes that optimize for latency and throughput, performing within 9% and 6% of an all-DRAM system, respectively.
4) Performance projections on to CXL-enabled memory, highlighting efficacy of the proposed policies across a range of memory performance characteristics.

II. Background

A. Large Language Models

The advent of modern LLMs is largely attributable to the Transformer [20], a machine learning network architecture that is able to extract word meanings and contextualize them as part of the broader prompt. Figure 1 shows the architecture of a decoder-only transformer model and the auto-regressive nature of LLM inference. At the heart of a decoder block is the multi-head attention (MHA) layer. The symbols Q, K, and V represent the query, key, and value vector representation of each token, obtained by multiplying the embedding vector of each token with the respective weight matrices. Each self attention block, called an attention *head*, extracts different semantic meaning from each token based on its weights. The decoder concatenates the output of each head with the original vector representation of each token to update its meaning and then passes it through a feed forward network (FFN) layer, often implemented as a multi-layer perceptron. FFN layer further refines the meaning of each token based on learned weights, producing another delta vector that is used to update each token.

Figure 1 also highlights two distinct inference phases: prefill and decode. During prefill, the entire input sequence is parsed by the model in a series of GEMM computations to produce the first token and store the key/value (KV) representations in a KV cache. In successive iterations, the decode stages utilize

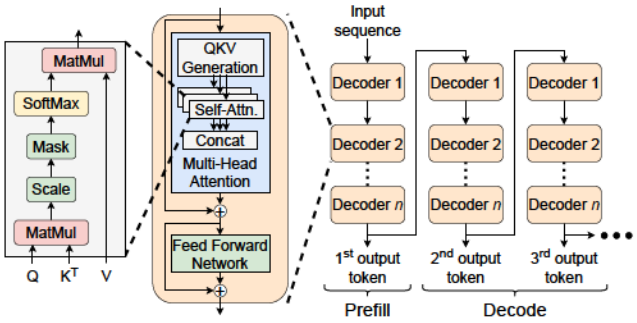


Fig. 1: Architecture of a decoder-only LLM. Prefill and decode perform GEMM and GEMV operations, rendering them compute and memory bound, respectively.

the KV cache and parse only the previously generated token in a series of GEMV computations to predict the next token.

Because of differences in the operational intensity of GEMM and GEMV, prefill is usually compute-bound while decode is memory-bound. Since most generative AI applications produce thousands of output tokens (e.g., LLaMa 4 has a context window size of 10 million tokens [21]), LLM inference is largely memory bound. A common technique to improve data reuse is batching of multiple requests, effectively turning the GEMV computations in decode phase to GEMM computations. The size of each batch is, however, limited by the storage requirements of the KV cache for each prompt.

Memory performance is not the only limiting factor, however. Model sizes since the introduction of transformers have exploded in size, from 175 billion in GPT-3 in 2020 [22] to 2 trillion in LLaMa 4 in 2025 [21]. Larger models incorporate more attention heads and larger weight matrices, allowing for semantically richer understanding of text and improved model performance. This exponential growth, however, has far outpaced the growth in AI hardware memory capacity as well [1], demanding solutions that offer both high bandwidth *and* capacity.

B. LLM Servers

Given the exponential growth in LLM model sizes, storing all model weights on accelerator memory has been increasingly challenging [1]. In response, several LLM frameworks have been proposed that enable offloading parts of the model to host memory or a backing store like disks. FlexGen [19] is one such framework that distributes model weights, KV cache, and hidden state between GPU memory, host main memory, and storage, and employs a "zig-zag" compute schedule on the GPU that optimizes for throughput and weight reuse. The compute schedule overlaps the computation of a batch of requests on layer *j* (e.g., MHA) with the loading of layer *j+1* (e.g., FFN) and its associated KV cache onto the GPU. Listing 1 shows FlexGen's computation schedule. Our characterization focuses on the overlap of compute with weight transfer given that the model weights are often at least an order of magnitude larger than both the KV cache and the hidden state.

Some LLM serving frameworks like llama.cpp [23], PowerInfer [24], and PowerInfer-2 [25] support concurrent CPU/GPU computation to avoid weight transfer bottlenecks. Among these servers, only PowerInfer-2 supports offloading to GPU memory, host memory, *and* storage. However, PowerInfer-2 is not open source. We therefore use FlexGen running on GPUs for our evaluation.

Listing 1: FlexGen computation schedule

```
for i in range(execute_gen_len):
    for j in range(num_layers):
        load_weight(i, j+1)
        compute_layer(i, j)
        sync()
```

C. Intel Optane

Intel introduced Optane Data Center Persistent Memory Module (DCPMM, simply referred to as Optane in this paper) in 2019, a PCM-based 3D Xpoint memory technology that is significantly more dense than traditional DRAM technology [26], [27]. Packaged in a DIMM form factor, Optane is byte-addressable and fits into regular DDR4 slots, communicating using a custom DDR4-based protocol called DDR-T [28], [29]. While extremely high capacity and energy efficient, Optane suffers from worse performance and limited write endurance, all properties of the underlying material. Prior work has shown that Optane achieves nearly 2.5x lower sequential read bandwidth compared to DRAM and about 6x lower write bandwidth [30]–[32]. These evaluations show a non-linear relationship between increasing concurrency and write bandwidth. Being PCM-based also limits the life of each memory module in terms of its write endurance [26].

Optane can be configured in two modes: App Direct and Memory. App Direct mode exposes Optane as a fast storage device with an optimized file system that bypasses the Linux page cache, enabled by its byte-addressable capabilities [33], [34]. Memory mode, meanwhile, exposes it as large main memory with DRAM serving as a direct-mapped cache to hide its performance deficiencies. Within App Direct mode, it is also possible to interface with the memory directly without any file system. This allows libraries like Memkind to expose Optane as a memory-only NUMA node, creating a flat memory hierarchy between DRAM and Optane and enabling direct use of the large memory pool by applications [35]. We use this support to compare DRAM and Optane performance.

In its Q2 2022 financial report, Intel announced that it is closing its SSD business, including Optane [36]. While no longer in production, Optane continues to serve as an effective evaluation platform for high capacity byte addressable memory technologies [37], [38]. Furthermore, CXL-expanded memory (see Section II-D) can be backed by Optane [39], ensuring broader applicability of findings related to Optane.

D. Compute Express Link

Compute express link (CXL) [14] was announced in 2019 [40] as an industry-standard interconnect technology to connect processors, devices, and memory expanders over the PCI Express bus interface. The CXL standard defines three protocols: CXL.io for I/O devices, CXL.cache for cache-coherent devices (e.g., caching accelerators), and CXL.memory for memory-enabled devices (e.g., memory expanders). The CXL 1.1 specification [41] defines three types of devices and what combination of the three protocols each would use.

Of the three types of CXL devices, Type-3 is of particular note. Defined for memory expanders, Type-3 devices use CXL.io and CXL.memory protocols to allow for coherent expansion of main memory capacity over PCIe. By providing load/store semantics similar to traditional main memory, Type-3 devices (simply referred to as CXL memory in this paper) provide transparent expansion of main memory without the limitations of traditional DDR interfaces. Compared to DDR, PCIe offers higher per-pin bandwidth and lower per-bit transfer energy [17]. Moreover, the memory technology across the interconnect is not bound to be DRAM, allowing for use of high density media like SSDs [39], [42]–[45] and Optane [39], as shown in Figure 2.

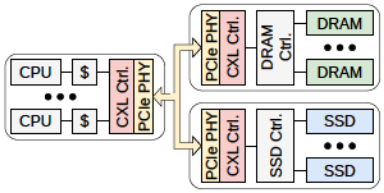


Fig. 2: CXL Type 3 devices enable transparent, coherent, and technology agnostic expansion of main memory.

Like Optane, CXL memory capacity comes at a performance cost. While performance varies with the design of the CXL controller itself [17], CXL adds at least 70 nanoseconds to round-trip memory access latency [46], not accounting for contention or memory technology variations at the expander. The achievable bandwidth is also limited by the underlying PCIe technology standard, which is 64 GB/s for the latest PCIe 5.0 x16 link [47]. In comparison, our DDR4-based evaluation system (Section III-A) achieves 157 GB/s across 8 memory channels. While PCIe 6.0 nearly doubles this bandwidth to 121 GB/s, it has not yet entered production at the time of writing.

III. Evaluation Methodology

A. Platform

We perform our evaluation on an Intel Optane-equipped dual-socket machine. The configuration is listed in Table I. Each socket has 4 memory controllers with 32 GB DDR4-2933 DRAM and 128 GB Optane DCPMM per channel/controller, providing a total of 256 GB DRAM and 1 TB Optane across the system. The system is paired with a Nvidia Ampere-based A100 GPU using 16 PCIe Gen 4 links that provide a maximum theoretical bandwidth of 32.0 GB/s. The GPU has 40 GB of HBM2 memory, organized as 5 stacks with 8 memory dies per stack [48].

TABLE I: System configuration

CPU	
Model	Dual socket Intel Xeon Gold 6330 (Ice Lake)
Frequency	Base: 2.0 GHz, turbo: 3.1 GHz
Cores (per socket)	28 (56 threads)
Memory (per socket)	4 memory controllers 16 GB DDR4-2933 DRAM x2 (per controller) 128 GB Optane (200 series) x1 (per controller)
GPU	
Model	Nvidia A100
Memory	40GB HBM2 (1215 MHz, 1555 GB/s)
Interface	PCIe Gen 4 x16 (32.0 GB/s)

Our evaluation considers all available configurations of Optane/DRAM. This includes Optane as storage with ext4-DAX file system [33], Optane Memory Mode (Optane main memory with DRAM cache), and Optane + DRAM main memory which is enabled by the Memkind library [35]. The last configuration exposes Optane memory as memory-only NUMA nodes.

B. Benchmarks

We use NVIDIA nvbandwidth [49] for basic bandwidth measurements between host and GPU. Our characterization presents results for both the NUMA nodes and with all combinations of Optane/DRAM host memory. To quantify real world performance impact of Optane on GPU performance, we use FlexGen [19] to run two variants of OPT models [18], OPT-30B and OPT-175B. OPT-30B models the scenario where model size surpasses GPU memory but fits in host DRAM. OPT-175B pushes further and surpasses host DRAM memory but fits in Optane memory, allowing us to compare the performance of heterogeneous main memory to traditional disk offloading. OPT-30B and OPT-175B contain 48 and 96 decoder blocks, resulting in 96 and 192 hidden layers (MHA + FFN), respectively. Along with one input embedding layer and one output embedding layer, the models have a total of 98 and 194 layers. Table II lists the various memory configurations we evaluate for each model. The input and output sequence lengths are limited to 128 and 21 tokens, respectively. We use prompts from the C4 dataset [50] and repeat each prompt 10 times.

TABLE II: LLM model/memory configuration

Model (# of Decoders)	Memory Configuration			Label
	SSD	Optane	DRAM	
OPT-30B (48)	N/A	N/A	Memory	DRAM
		Memory	N/A	NVDRAM
			Cache	MemoryMode
OPT-175B (96)	Storage	N/A	Memory	SSD
	N/A	Storage	Memory	FSDAX
		Memory	N/A	NVDRAM
			Cache	MemoryMode

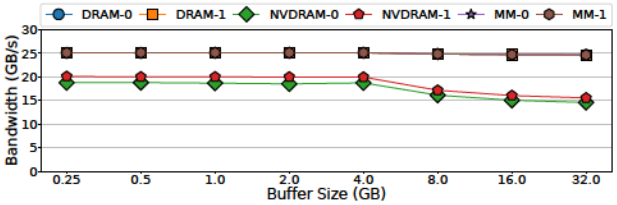
C. Metrics

We evaluate LLM performance using three key metrics: time to first token (TTFT), time between tokens (TBT), and throughput in terms of tokens per second. TTFT measures prefill latency, the inference stage that processes the entire input prompt. TBT measures decode latency, the successive stages of inference that utilize the KV cache from the prefill stage alongside the previously generated token to generate the next token. Finally, throughput measures the overall token generation rate across the entire process. For each metric, we present the arithmetic mean across all its values except the first, which we discard to account for cold start effects.

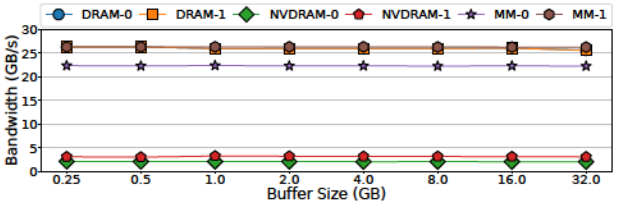
IV. GPU/Host Data Movement Characterization

A. Basic Bandwidth Measurements

Figure 3 presents host to GPU (3a) and GPU to host (3b) bandwidth for buffer sizes between 256 MB and 32 GB. The figure presents the bandwidth when copying to/from DRAM, Optane DRAM (NVDRAM), and Optane Memory Mode (MM) for both the NUMA nodes.



(a) Host to GPU. DRAM-0, DRAM-1, MM-0, and MM-1 overlap perfectly.



(b) GPU to host. DRAM-0, DRAM-1, and MM-1 overlap perfectly.

Fig. 3: Host/GPU memory copy bandwidth. The numbers 0 and 1 represent the two NUMA nodes.

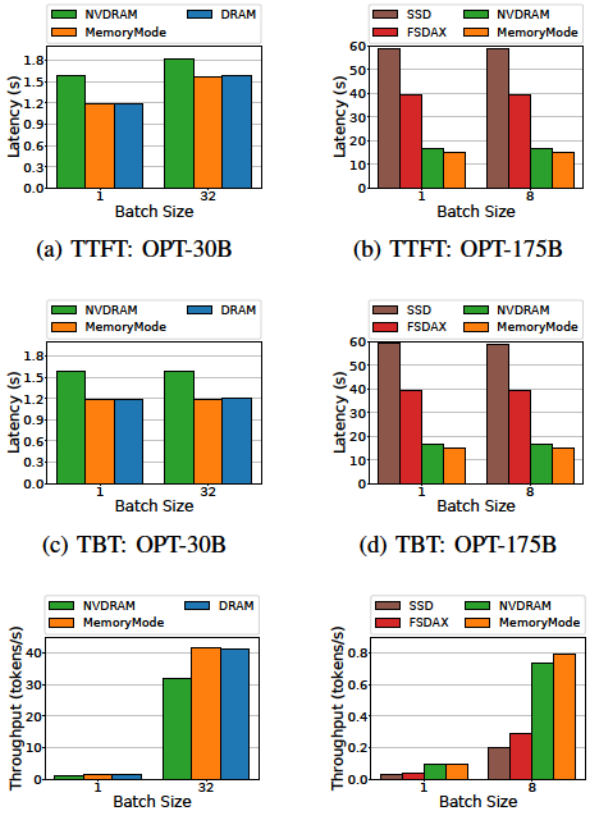
Figure 3a shows how host to GPU bandwidth suffers a near constant loss of 20% with NVDRAM compared to DRAM up to a buffer size of 4 GB, with NVDRAM bandwidth dropping from 19.91 GB/s at 4 GB to 15.52 GB/s at 32 GB and increasing the performance deficit to 37%. We attribute this drop in performance to potentially non-consecutive data placement on NVM media due to wear-leveling and to misses in the Address Indirection Table (AIT) buffer that translates physical addresses to NVM media addresses [29], [32], [51]. MM is able to completely hide this performance gap, however, because the buffer size fits within the DRAM cache capacity (note that the MM and DRAM lines in Figure 3a overlap each other).

The performance gap between DRAM and NVDRAM is even wider when it comes to Optane's write performance. GPU to host bandwidth (Figure 3b) is 88% lower with NVDRAM compared to DRAM across all buffer sizes, maxing out at 3.26 GB/s with a buffer size of 1 GB. This bandwidth is consistent with prior observations [30]. We also notice how bandwidth for Optane is higher on NUMA node 1 compared to NUMA node

0. This is because the GPU is connected to PCIe ports local to node 1, meaning that accesses to node 0 need to go over the on-chip interconnect. Prior work has shown how Optane write performance worsens when accessed remotely [31]. Our own results using Intel Memory Latency Checker [52] also confirm this, including remote MM's inability to reach remote DRAM bandwidth.

B. LLM Performance

Figure 4 shows the performance of OPT-30B and OPT-175B models in terms of average TTFT, TBT, and throughput. The three metrics are presented for a batch size of 1 along with the maximum permissible size based on available GPU memory to avoid the KV cache impacting communication overheads (32 for OPT-30B and 8 for OPT-175B).



(a) TTFT: OPT-30B (b) TTFT: OPT-175B

(c) TBT: OPT-30B (d) TBT: OPT-175B

(e) Throughput: OPT-30B (f) Throughput: OPT-175B

Fig. 4: Time to first token (TTFT), time between tokens (TBT), and throughput (tokens/s).

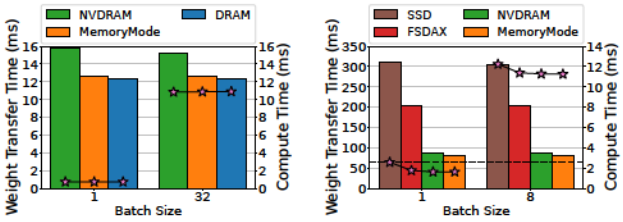
SSD and FSDAX configurations are, unsurprisingly, the slowest performing configurations. While FSDAX improves TTFT/TBT/throughput by 33.46%/33.48%/35.31% and 33.44%/33.58%/46.68% for OPT-175B with batch sizes 1 and 8, respectively, it falls short of reaching NVDRAM's performance. This is largely a result of Optane being exposed through the file system interface in FSDAX, requiring the use of a bounce buffer in DRAM when copying weights from Optane to GPU.

NVDRAM's lower bandwidth compared to DRAM impacts both TTFT and TBT significantly, hurting overall throughput. OPT-30B's TTFT increases by 33.03% and 15.05% with batch sizes 1 and 32, respectively, under NVDRAM compared to DRAM. Similarly, TBT goes up by 33.03% and 30.55% under the two batch sizes. This leads to a reduction in throughput by 18.96% and 22.68%, respectively. MemoryMode matches DRAM performance because the host-side model weights fit within DRAM cache. While there is no DRAM optima to compare against for OPT-175B, MemoryMode improves TTFT/TBT/throughput compared to NVDRAM by 7.67%/7.69%/0.60% and 8.90%/8.92%/7.98% for batch sizes 1 and 8, respectively. Keeping in mind that the model size outgrows the DRAM cache size here, an all-DRAM system likely performs even better than this.

Increasing batch sizes improves throughput almost linearly, as seen in Figures 4e and 4f. Ordinarily, prefill is compute bound because of its high operational intensity. As a consequence, TTFT tends to increase with an increase in batch size. We see this with OPT-30B where TTFT increases by 32.41%, 14.51%, and 31.50% under DRAM, NVDRAM, and MemoryMode configurations, respectively, when going from a batch size of 1 to 32 (Figure 4a). OPT-175B does not experience an increase in TTFT (Figure 4b) with increasing batch size because its large weight size makes its prefill stage memory bound. Decode, on the other hand, is memory bound because it consists of a series of GEMV computations which have low operational intensity. While increasing the batch size helps convert the GEMV computation in the feed forward network (FFN) stage (Section II-A) into GEMM, each prompt must still perform a series of GEMV operations in the multi-head attention (MHA) stage (Section II-A) with its own local KV cache. This limits the scaling of TBT with increasing batch sizes, as seen in Figures 4c and 4d.

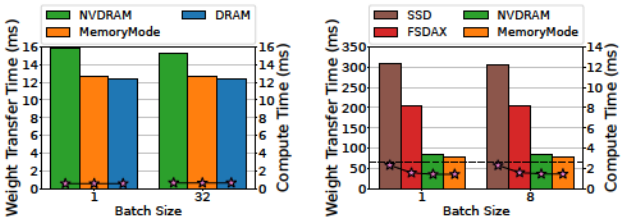
In order to better understand prefill and decode performance, we use FlexGen's built-in timers to get a breakdown of the time spent on compute and communication in each phase. Recall that FlexGen overlaps compute in layer j with the loading of weights for layer $j+1$ (Section II-B). Figure 5 presents this overlap on a per-layer basis for each model with different batch sizes, separate for both prefill and decode stages. Since both OPT-30B and OPT-175B consist of several decoder blocks (Table II), the longer running operation within this pipeline affects the overall inference latency. For OPT-175B, we also measure and show the ideal average weight transfer time in an all-DRAM system by running the model with 8 decoder blocks instead of the default 96.

Figure 5 shows how the average weight transfer time under each memory configuration affects its TTFT and TBT performance (Figure 4), highlighting the memory-bound nature of LLM inference. Looking at OPT-30B's prefill stage (Figure 5a), we observe that the average compute time increases by about 15x for all three configurations when the batch size is increased from 1 to 32. This is the reason behind the increase in TTFT we observed earlier (Figure 4a) since many layers become compute-bound (even though the average compute



(a) Prefill: OPT-30B

(b) Prefill: OPT-175B

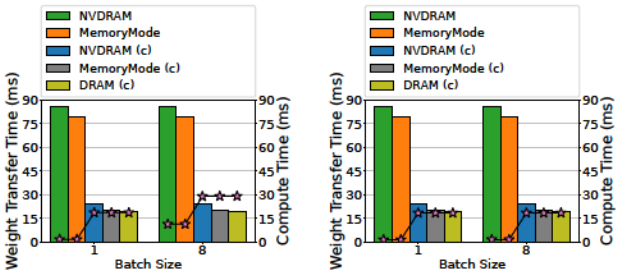


(c) Decode: OPT-30B

(d) Decode: OPT-175B

Fig. 5: Compute/communication overlap during prefill and decode stages. The bars represent average weight transfer time while the line represents average compute time. The horizontal dashed line represents the ideal weight transfer time on an all-DRAM system. Note the different scales for the two y-axes in (b) and (d).

time is below the average weight transfer time). The decode stage stays largely memory-bound, however, even with the large batch size (Figure 5c). Batching has been known to have minimal impact on the arithmetic intensity of the decode stage [6]. OPT-175B, meanwhile, is memory-bound in both prefill and decode stages because of its significantly larger weight sizes (hidden layer size of 12,288 versus OPT-30B's 7,168). While an all-DRAM system would improve the average weight transfer time in both stages by 32.78% and 22.41% compared to NVDRAM and MemoryMode, respectively, it will still be orders-of-magnitude higher than the compute time.



(a) OPT-175B: Prefill

(b) OPT-175B: Decode

Fig. 6: Compute/communication overlap during prefill and decode stages with compression. The bars represent average weight transfer time while the line represents average compute time. The symbol *(c)* represents compressed configurations.

Compression/quantization is a well-known strategy to decrease model size, reducing both the model footprint and weight transfer time at the cost of increased computation, due to on-the-fly decompression, and potential accuracy loss. FlexGen supports compressing model weights down from FP16 to a 4-bit representation using group-wise quantization [53], reducing the model size to nearly a quarter with a negligible loss in accuracy [19]. This allows the model to fit entirely on host memory, even with traditional DRAM, obviating the need to offload to storage. Figure 6 highlights the compute/communication tradeoff of compression for OPT-175B for NVDRAM, MemoryMode, and DRAM configurations. Compared to the baseline, compression reduces weight transfer time by 72% and 74% for NVDRAM and MemoryMode, respectively, bringing it within 25% and 6% of DRAM ideal. The compute time, meanwhile, increases by anywhere between 2.5x-13x for both NVDRAM and MemoryMode configurations. Compression allows OPT-30B to fit fully into GPU memory, which we do not present.

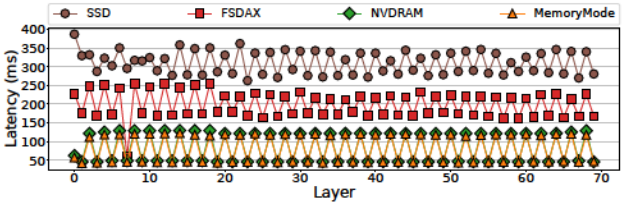
V. Impact of Weight Placement

Given the memory-bound nature of LLM inference, we take a closer look at the cost of transferring each weight to the GPU. We focus on model weight placement since the weight size dominates the total memory footprint. For instance, for a single OPT-175B self-attention block, the model weights occupy 3.38 GB of memory while the KV cache, the second highest contributor to the total memory footprint, occupies 47.98 MB for a batch size of 1 at the maximum context length of 2048 (72x smaller than weights). The total memory footprint of the model weights is 324.48 GB while that of the KV cache is 4.5 GB. For context, the GPU we use for our evaluation has 40 GB of onboard memory, which can hold the entire KV cache (4.5 GB), but not model weights (324.48 GB). Given this, we first evaluate the cost of transferring weights under FlexGen's existing weight strategy, highlighting an imbalance in compute and communication overlap. Based on our analysis, we propose two alternate weight placement schemes, both of which utilize compression to minimize data movement. The goal of the first scheme is to optimize for latency by mitigating the imbalance in compute and communication. The second scheme, meanwhile, optimizes for throughput by maximizing the batch size. We present these optimizations for NVDRAM and MemoryMode configurations only using OPT-175B, making a case for such memory technologies as an effective DRAM replacement for LLM inference.

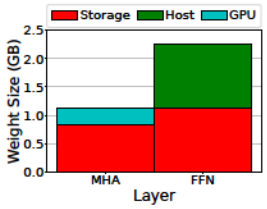
A. FlexGen's Weight Transfer Costs

Figure 7a plots the latency of loading each layer of OPT-175B up to layer 70 of 194 for all memory configurations with compression. The plot has a striking sawtooth pattern that continues all the way until layer 194 (not shown). This is a result of FlexGen's weight placement scheme, presented in Listing 2. Given a list of weights (`weight_specs`) and a user-specified percentage distribution across storage, host, and GPU (described in `policy`), the allocator, `init_weight_list`, distributes each layer's weights across the hierarchy to meet this goal. To achieve this, the function iterates over all the

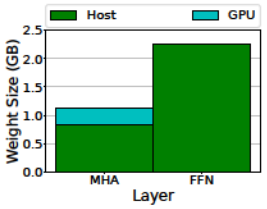
weights of the layer (line 17) and calculates the percentage contribution of the weights preceding weight *i* to the total layer size (lines 18-20). Based on this percentage and the input percentage distribution, `get_choice()` returns the device to allocate weight *i* on.



(a)



(b) SSD/FSDAX. Storage: SSD/Optane, host: DRAM.

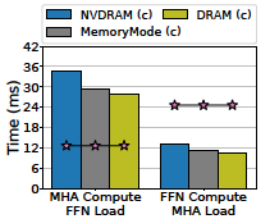


(c) NVDRAM/MemoryMode. Host: Optane.

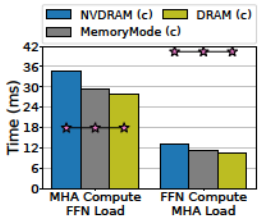
Fig. 7: Per-layer weight load latency for a subset of OPT-175B layers (70/194) (a) and weight distribution of multi-head attention (MHA) and feed forward network (FFN) layers in SSD/FSDAX (b) and NVDRAM/MemoryMode (c) configurations.

Our experiments show that this allocation scheme is imperfect and struggles to achieve the desired percentage distribution because differences in weight sizes do not lend themselves well to such a fine-grained distribution. For instance, for *(storage, host, GPU)* ratios of *(65, 15, 20)* under SSD/FSDAX configurations, the achieved overall weight distribution is *(58.6, 33.1, 8.3)*. Similarly, the input and achieved distribution for NVDRAM/MemoryMode is *(0, 80, 20)* and *(0, 91.7, 8.3)*, respectively. Furthermore, the weight distribution scheme is unaware of the relative size of each layer, which leads to the imbalanced weight transfer times across layers we see in Figure 7a. In particular, the dips and ridges in the figure correspond to multi-head attention (MHA) and feed forward network (FFN) layers, respectively. Figures 7b and 7c show the weight distribution of these two layers under SSD/FSDAX and NVDRAM/MemoryMode configurations, respectively. In both cases, we see how the larger FFN layer gets no allocation on the GPU while the smaller MHA layer does.

A direct consequence of this asymmetric weight distribution is that weigh transfer cannot be hidden effectively behind computation. Figure 8 shows the time spent in loading weights for FFN/MHA layers and how it overlaps with computing MHA/FFN for the prefill stage. MHA has a lower computation time than FFN, yet it is overlapped with the transfer of a larger set of weights because of FlexGen's weight distribution.



(a) Batch size 1



(b) Batch size 8

Fig. 8: Overlap of MHA/FFN compute with the transfer of FFN/MHA weights in the prefill stage of OPT-175B with compression enabled. The bars represent average weight transfer time while the line represents average compute time. The overlap in decode stage with both batch sizes is nearly identical to prefill with batch size 1.

Listing 2: FlexGen weight allocation algorithm

```
1  def get_device(cur_percent, percents, choices):
2      percents = numpy.cumsum(percents)
3      for i in range(len(percents)):
4          if cur_percent < percents[i]:
5              return choices[i]
6      return choices[-1]
7
8  def init_weight_list(weight_specs, policy, env):
9      dev_percents = [policy.disk_percent,
10                     policy.cpu_percent,
11                     policy.gpu_percent]
12     dev_choices = [env.disk, env.cpu, env.gpu]
13
14     sizes = [spec.size for spec in weight_specs]
15     sizes_cumsum = numpy.cumsum(sizes)
16
17     for i in range(len(weight_specs)):
18         mid_percent = (sizes_cumsum[i] - \
19                        sizes[i] / 2) / \
20                       sizes_cumsum[-1]
21         dev = get_choice(mid_percent * 100,
22                          dev_percents,
23                          dev_choices)
24         dev.allocate(weight_specs[i])
```

B. HeLM: Latency Optimizing Weight Placement

In order to balance the compute/communication pipeline, we introduce **He***terogeneous* **L***ayerwise* **M***apping* (HeLM), a modified weight placement algorithm that attempts to equalize computation of layer *i* with weight transfer time of layer *i+1*. The key idea behind HeLM is to allocate more GPU space for layers whose transfer time will be overlapped with shorter computing layers. HeLM accomplishes this by allocating the weights of the first fully connected (FC) layer of FFN on the GPU, along with the weights of all the bias and normalization layers for both MHA and FFN. The rest of the MHA and FFN weights are offloaded on to the host memory. The algorithm is presented in Listing 3 and illustrated in Figure 9.

Listing 3 shows how HeLM uses a custom weight distribution for MHA (lines 2-3) and FFN (lines 4-5) layers, along with sorting the weights in increasing order by size (line 13). Note that HeLM specifies device percentages in the order *(GPU, host, storage)*, instead of the default *(storage, host, GPU*.

Listing 3: HeLM weight allocation algorithm. The algorithm follows the default allocation algorithm line 16 onwards (Listing 2, line 14).

```
def init_weight_list(weight_specs, policy, env):
    if is_mha(weight_specs):
        dev_percents = [10, 90, 0]
    elif is_ffn(weight_specs):
        dev_percents = [30, 70, 0]
    else:
        dev_percents = [policy.gpu_percent,
                        policy.cpu_percent,
                        policy.disk_percent]

    dev_choices = [env.gpu, env.cpu, env.disk]

    weight_specs = list(sorted(weight_specs,
                        key=lambda x: x.size)

    sizes = [spec.size for spec in weight_specs]
    ...
```

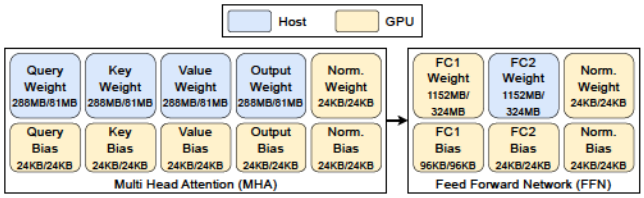


Fig. 9: Breakdown of HeLM's weight distribution across host and GPU. The number under each weight is the uncompressed/compressed size of the weight.

Figure 10 shows the weight distribution of MHA and FFN layers achieved by HeLM. This distribution reduces the time to transfer FFN weights by 49.33% while increasing it by 32.55% for MHA layers, as seen in Figure 11a. However, the increase in MHA load time is easily overlapped with FFN computation, leading to an overall reduction in layer processing time.

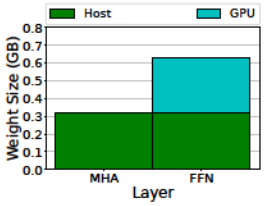
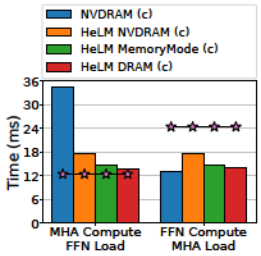


Fig. 10: HeLM's weight distribution

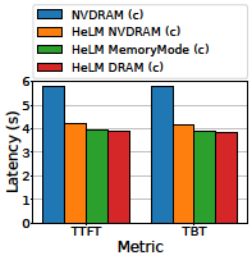
The balanced compute/communication pipeline directly results in improvements to inference latency. Figure 11b shows how HeLM improves TTFT and TBT on NVDRAM by 27.20% and 27.44% compared to the baseline scheme (Section V-A). These numbers are within 8.75% and 8.91% of DRAM. MemoryMode, meanwhile, experiences an improvement of 31.90% and 32.28%, both of which are within 1.73% and 1.64% of DRAM. These results highlight how careful data placement can enable the use of Optane-like emerging memory technologies, and the heterogeneous configurations they enable, in latency-sensitive LLM serving scenarios.

C. All-CPU: Throughput Optimizing Weight Placement

We study a second optimization called All-CPU where all weights are placed on host memory, leaving GPU memory for



(a) Compute/comm. overlap



(b) TTFT and TBT

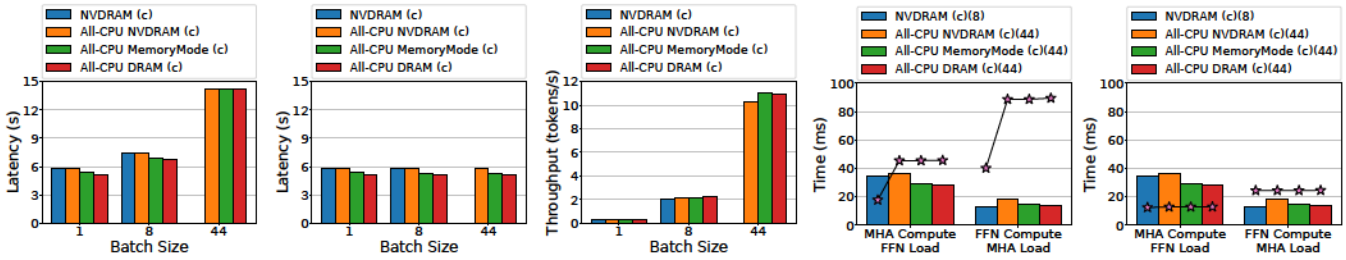
Fig. 11: Impact of HeLM on (a) compute/communication overlap during decode and (b) time to first token (TTFT) and time between tokens (TBT). This is evaluated using OPT-175B with a batch size of 1. In Figure (a), the bars represent average weight transfer time while the line represents average compute time.

KV cache and hidden state. This makes sense because even with HeLM, only 33% of the total weights are held in the GPU memory. By pushing all of them out to host memory, we can trade weight transfer time for improved weight reuse enabled by a higher batch size. While this optimization has been explored before [19], we present it in the context of heterogeneous memory and evaluate how it fares compared to traditional DRAM.

Figures 12a, 12b, and 12c compare the TTFT, TBT, and throughput of All-CPU to the baseline scheme (Section V-A) for batch sizes 1, 8, and 44, the latter of which is only possible with All-CPU. With all three batch sizes, the KV cache continues to fit inside GPU memory. All-CPU does not have a significant impact on either TTFT or TBT (1% degradation) or throughput (5% gain) with NVDRAM compared to the baseline at batch sizes 1 and 8. This highlights the minimal performance advantage of keeping model weights on GPU at all when optimizing for throughput. All-CPU makes better use of that space by allocating it to the KV cache instead and expanding the batch size. A key result here is the 5x increase in throughput when going from baseline NVDRAM at batch size 8 to All-CPU NVDRAM at batch size 44 (Figure 12c). In fact, the throughput at batch size 44 with All-CPU NVDRAM is within 6% of All-CPU DRAM.

Figures 12d and 12e show how the compute/communication overlap varies between the baseline scheme with a batch size of 8 and All-CPU with a batch size of 44. While MHA weight transfer time increases significantly with All-CPU relative to FlexGen's baseline weight allocation (Figure 7c, which allocates some MHA weights on the GPU), it is completely hidden behind computation in both prefill and decode stages. Interestingly, compute time in the decode stage does not increase when the batch size is increased from 8 to 44 (Figure 12e), indicating potential compute under-utilization at a lower batch size. By maximizing the batch size, All-CPU improves compute utilization, which leads to an overall increase in throughput (Figure 12c).

All-CPU MemoryMode reduces TTFT/TBT compared to



(a) Time to first token (TTFT) (b) Time between tokens (TBT) (c) Throughput (tokens/sec) (d) Compute/comm. overlap (prefill) (e) Compute/comm. overlap (decode)

Fig. 12: Performance impact of All-CPU weight allocation on OPT-175B. Figures (d) and (e) compare compute/communication overlap with baseline weight allocation and batch size 8 to All-CPU and batch size 44. In both the figures, the bars represent average weight transfer time while the line represents average compute time.

All-CPU NVDRAM by 5.83%/5.77% with batch size 1, by 6.86%/9.46% with batch size 8, and by 0.24%/8.39% with batch size 44. It impacts the throughput the most at batch size 44, however, improving it by 7.57% and performing at-par with DRAM (1.15% better). This is evidence that optimized data placement on heterogeneous memory can not only achieve latency close to an all-DRAM system, but also throughput.

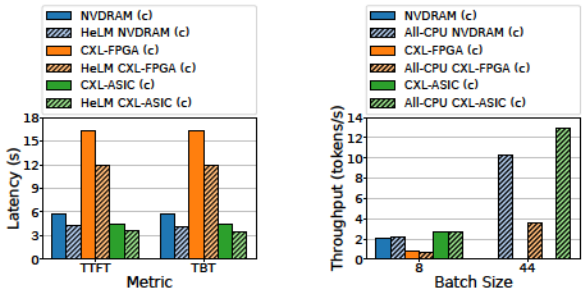
D. CXL Performance Projections

Like Intel Optane, CXL memory provides high capacity at the cost of performance. This cost varies based on both the CXL controller architecture as well as the underlying memory technology [17]. In order to evaluate the impact of our proposed optimizations on CXL memory, we borrow the bandwidth of two different CXL configurations from prior work and project the performance of each. These configurations are presented in Table III. CXL-FPGA is based on evaluation presented by Sun et al. [17] (called CXL-C in their paper) and uses an FPGA-based CXL controller backed by single channel DDR4-3200 memory. CXL-ASIC, meanwhile, is borrowed from Wang et al. [54] (called System A in their paper) and is based on an undisclosed commercial ASIC implementation backed by single channel DDR5-4800 memory. We utilize the bandwidth numbers for each configuration from the respective paper to project weight transfer times for each layer and calculate the achievable compute/communication overlap (Table IV), TTFT/TBT (Figure 13a), and throughput (Figure 13b), comparing it to NVDRAM.

TABLE III: CXL configurations

Name	Memory Technology	Bandwidth (GB/s)
CXL-FPGA [17]	DDR4-3200 x1	5.12
CXL-ASIC [54]	DDR5-4800 x1	28

Table IV shows the compute/communication overlap for each CXL configuration under all three weight allocation policies: baseline (Section V-A), HeLM (Section V-B), and All-CPU (Section V-C). CXL-FPGA and CXL-ASIC cover a wide performance spectrum owing to differences in their CXL controller design. CXL-FPGA achieves considerably lower memory bandwidth than both NVDRAM and CXL-ASIC. The lower bandwidth means that CXL-FPGA stays largely memory bound across all weight allocation policies and inference stages, except All-CPU prefill with a batch size of 44. CXL-ASIC significantly outperforms both NVDRAM and CXL-FPGA, being the only configuration that achieves FFN load latency lower than MHA compute latency with HeLM. These results highlight how HeLM and All-CPU are able to improve the compute/communication overlap across a wide variety of CXL memory implementations.



(a) (b)

Fig. 13: Projected performance improvements offered by HeLM (batch size=1) (a) and All-CPU (b) on CXL-based systems using OPT-175B.

The improved compute/communication overlap with HeLM directly translates to lower inference latency, as shown in Figure 13a. HeLM improves TTFT/TBT by 27% and 21% for CXL-FPGA and CXL-ASIC, respectively. The improvements for CXL-ASIC come from reducing FFN weight transfer time and increasing MHA weight transfer time, thereby balancing compute with communication. CXL-FPGA, on the other hand, performs better simply because HeLM is able to store more weights on the GPU compared to the baseline scheme.

The gains from All-CPU are more varied. While NVDRAM and CXL-ASIC experience nearly the same performance with both baseline and All-CPU at batch size 8, CXL-FPGA suffers an 8.35% drop in throughput due to its poor memory performance. Nonetheless, both CXL-ASIC and CXL-FPGA

TABLE IV: Overlap of compute and communication with different weight allocation policies under NVDRAM configuration and the three different CXL configurations. A ratio of 1 indicates perfect overlap, while lower and higher values indicate memory-boundedness and compute-boundedness, respectively.

Allocation Policy	Batch Size	Stage	MHA compute/FFN Load (ratio)			FFN Compute/MHA Load (ratio)		
			NVDRAM (c)	CXL-FPGA (c)	CXL-ASIC (c)	NVDRAM (c)	CXL-FPGA (c)	CXL-ASIC (c)
Baseline	1	Prefill	0.36	0.1	0.56	1.86	0.53	2.9
		Decode	0.36	0.1	0.55	1.85	0.53	2.88
	8	Prefill	0.52	0.14	0.79	3.07	0.87	4.77
		Decode	0.36	0.1	0.55	1.85	0.53	2.88
HeLM	1	Prefill	0.72	0.2	1.12	1.4	0.4	2.18
		Decode	0.71	0.2	1.1	1.4	0.4	2.16
All-CPU	1	Prefill	0.37	0.1	0.56	1.41	0.4	2.18
		Decode	0.36	0.1	0.55	1.39	0.39	2.16
	8	Prefill	0.51	0.14	0.79	2.3	0.65	3.57
		Decode	0.36	0.1	0.55	1.39	0.39	2.16
	44	Prefill	1.25	0.37	2.01	4.82	1.43	7.84
		Decode	0.35	0.1	0.57	1.33	0.4	2.16

achieve 4.74x and 5.04x higher throughput when going from the baseline scheme at batch size 8 to All-CPU at batch size 44, highlighting the efficacy of the placement scheme regardless of memory performance.

VI. Related Work

LLM memory optimizations: Two of the most well-known solutions to reduce model size include pruning/sparsification [55]–[58] and quantization [59]–[62]. FlexGen uses group-wise quantization (GWQ) [53] to compress weights down to four bits. This has shown to preserve accuracy for both encoder-only models like BERT [53] and decoder-only models like OPT [19]. KV cache is the second highest contributor to the memory footprint during inference, accounting for as much as 30% of the total size [63]. Optimizations to minimize KV cache size include quantization [64], [65], virtual memory-like block granular management [63], prediction of per-prompt cache size [66], and temporal locality prediction-based dynamic offloading to storage [67]. These approaches can be combined with our work to further increase batch sizes.

Tiered memory: Use of technologies like Intel Optane and CXL memory is of particular interest to the HPC and ML communities given the large memory footprint of their applications. Prior work has extensively characterized the performance impact of Optane memory on HPC applications [68], [69], with some exploring application-specific optimizations for applications like materials simulation [70], weather forecasting [70], plasma simulation [71], and ML training [72]. When it comes to CXL memory, recent work has looked at both application-agnostic transparent page management across local memory and CXL memory [73], as well as application latency tolerance-aware allocation of virtual machines in datacenters [74]. Our work demonstrates how careful application-aware placement of data between heterogeneous host memory and GPU memory can compensate for the former's performance deficit compared to traditional DRAM.

GPU memory expansion: Our work very closely matches that of Choi et al. [75], where the authors evaluate the LLM serving performance of Nvidia Grace Hopper Superchip (GHS) [76], using vLLM framework [63] to serve LLaMa 3.1 8B, 70B, and 405B models [77]. There are two key differences between our work and theirs. First, GHS pairs only traditional DRAM with the CPU, while we evaluate the impact of emerging memory technologies and the heterogeneous configurations they enable. Second, unlike FlexGen, vLLM considers GPU memory as an inclusive cache. While this is similar to the All-CPU layout we evaluate, our work also evaluates a flat memory hierarchy where weights can be placed across GPU and host memories. Keeping these differences in mind, we consider the two works to be complementary to each other.

Other ways of exposing host memory as expanded GPU memory include Nvidia's Unified Virtual Addressing [78] that allows for shared pointers between the host and GPU. Nvidia's Unified Memory (UM) [79] supports on-demand transparent movement of pages from host to GPU memory. Such movement comes with significant overheads, spawning a large body of work optimizing the prefetch/eviction policies [80]–[84], software hint-driven placement and prefetching [85], GPU throttling and compression [84], and hardware-assisted memory management [86]. Beyond host memory, several software frameworks allow direct access to storage from the GPU, both for direct file access [87]–[90] as well as to provide crash consistency [91], [92]. Gouk et al. [16] developed and synthesized a CXL controller for GPUs to directly access CXL memory expanders. Prior work has also looked at expanding on-chip GPU memory itself with storage class memory for both capacity [93], [94] and persistence [95]–[97].

VII. Conclusion

As large language models continue to evolve, the growth in model sizes will continue to stress the memory subsystem for performance and capacity. This paper shows how replacing DRAM with emerging technologies like Intel Optane can enable larger model sizes that fit in main memory, but not without a performance penalty. Diving deeper into the performance characteristics of running inference on OPT-30B and OPT-175B models with FlexGen, a LLM serving framework, we show how this performance degradation is largely a function of data placement and balancing computation with commu-

nication. We evaluate two alternate data placement schemes, one each optimizing for latency and throughput. The latency optimizing scheme, called HeLM, performs compute-time aware data placement that attempts to equalize the compute time of layer *i* and the weight transfer time of layer *i+1*. HeLM improves compute/communication pipeline balance and achieves token generation latency on Optane main memory within 9% of an all-DRAM system. The throughput optimizing scheme, called All-CPU, offloads all weights to the host memory, bumping the maximum possible batch size up from 8 to 44. The increased batch size helps All-CPU Optane net a throughput increase of 5x compared to baseline DRAM at a batch size of 8, while maintaining the same time between tokens. All-CPU Optane is within 6% of All-CPU DRAM, paving the way for models that exceed the capacity of DRAM. Our projections on CXL-enabled memory indicate that these findings remain valid for a broad spectrum of CXL devices. The presented techniques may be generalized to other models and frameworks by adapting to their compute schedule and data movement costs. We hope that the insights presented in this paper inform the design of improved weight placement algorithms that can automatically make latency/throughput tradeoffs based on desired quality of service requirements.

ACKNOWLEDGEMENTS

This work was supported in part by U.S. National Science Foundation grant CNS-1900803. We would like to thank the anonymous program committee and artifact evaluation committee reviewers for their feedback.

REFERENCES

[1] A. Gholami, Z. Yao, S. Kim, C. Hooper, M. W. Mahoney, and K. Keutzer, "AI and Memory Wall," *IEEE Micro*, vol. 44, no. 3, pp. 33–39, 2024.

[2] K. I. Roumeliotis and N. D. Tselikas, "ChatGPT and Open-AI Models: A Preliminary Review," *Future Internet*, vol. 15, no. 6, 2023. [Online]. Available: https://www.mdpi.com/1999-5903/15/6/192

[3] N. Friedman, "Introducing GitHub Copilot: your AI pair programmer," Jun. 2021. [Online]. Available: https://github.blog/news-insights/product-news/introducing-github-copilot-ai-pair-programmer/

[4] S. Bubeck, V. Chandrasekaran, R. Eldan, J. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y. Li, S. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang, "Sparks of Artificial General Intelligence: Early experiments with GPT-4," 2023, arXiv: 2303.12712. [Online]. Available: https://arxiv.org/abs/2303.12712

[5] A. K. Kamath, R. Prabhu, J. Mohan, S. Peter, R. Ramjee, and A. Panwar, "POD-Attention: Unlocking Full Prefill-Decode Overlap for Faster LLM Inference," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS '25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 897–912, event-place: Rotterdam, Netherlands. [Online]. Available: https://doi.org/10.1145/3676641.3715996

[6] P. Patel, E. Choukse, C. Zhang, A. Shah, I. Goiri, S. Maleki, and R. Bianchini, "Splitwise: Efficient Generative LLM Inference Using Phase Splitting," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, 2024, pp. 118–132.

[7] O. Mutlu, "Main Memory Scaling: Challenges and Solution Directions," in *More than Moore Technologies for Next Generation Computer Design*, R. O. Topaloglu, Ed. New York, NY: Springer New York, 2015, pp. 127–153. [Online]. Available: https://doi.org/10.1007/978-1-4939-2163-8_6

[8] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable High Performance Main Memory System Using Phase-Change Memory Technology," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA '09. New York, NY, USA: Association for Computing Machinery, 2009, pp. 24–33, event-place: Austin, TX, USA. [Online]. Available: https://doi.org/10.1145/1555754.1555760

[9] Y. Chen, "ReRAM: History, Status, and Future," *IEEE Transactions on Electron Devices*, vol. 67, no. 4, pp. 1420–1433, 2020.

[10] D. Apalkov, A. Khvalkovskiy, S. Watts, V. Nikitin, X. Tang, D. Lottis, K. Moon, X. Luo, E. Chen, A. Ong, A. Driskill-Smith, and M. Krounbi, "Spin-transfer torque magnetic random access memory (STT-MRAM)," *J. Emerg. Technol. Comput. Syst.*, vol. 9, no. 2, May 2013, place: New York, NY, USA Publisher: Association for Computing Machinery. [Online]. Available: https://doi.org/10.1145/2463585.2463589

[11] E. Kültürsay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013, pp. 256–267.

[12] D. Ralph and M. Stiles, "Spin transfer torques," *Journal of Magnetism and Magnetic Materials*, vol. 320, no. 7, pp. 1190–1216, Apr. 2008, publisher: Elsevier BV. [Online]. Available: http://dx.doi.org/10.1016/j.jmmm.2007.12.019

[13] M. Shihab, J. Zhang, S. Gao, J. Sloan, and M. Jung, "Couture: Tailoring STT-MRAM for Persistent Main Memory," in *4th Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (INFLOW 16)*. Savannah, GA: USENIX Association, Nov. 2016. [Online]. Available: https://www.usenix.org/conference/inflow16/workshop-program/presentation/shihab

[14] CXL Consortium, "Compute Express Link." [Online]. Available: https://computeexpresslink.org/

[15] M. Arif, A. Maurya, and M. M. Rafique, "Accelerating Performance of GPU-based Workloads Using CXL," in *Proceedings of the 13th Workshop on AI and Scientific Computing at Scale Using Flexible Computing*, ser. FlexScience '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 27–31, event-place: Orlando, FL, USA. [Online]. Available: https://doi.org/10.1145/3589013.3596678

[16] D. Gouk, S. Kang, H. Bae, E. Ryu, S. Lee, D. Kim, J. Jang, and M. Jung, "Breaking Barriers: Expanding GPU Memory with Sub-Two Digit Nanosecond Latency CXL Controller," in *Proceedings of the 16th ACM Workshop on Hot Topics in Storage and File Systems*, ser. HotStorage '24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 108–115, event-place: Santa Clara, CA, USA. [Online]. Available: https://doi.org/10.1145/3655038.3665953

[17] Y. Sun, Y. Yuan, Z. Yu, R. Kuper, C. Song, J. Huang, H. Ji, S. Agarwal, J. Lou, I. Jeong, R. Wang, J. H. Ahn, T. Xu, and N. S. Kim, "Demystifying CXL Memory with Genuine CXL-Ready Systems and Devices," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 105–121, event-place: Toronto, ON, Canada. [Online]. Available: https://doi.org/10.1145/3613424.3614256

[18] S. Zhang, S. Roller, N. Goyal, M. Artetxe, M. Chen, S. Chen, C. Dewan, M. Diab, X. Li, X. V. Lin, T. Mihaylov, M. Ott, S. Shleifer, K. Shuster, D. Simig, P. S. Koura, A. Sridhar, T. Wang, and L. Zettlemoyer, "OPT: Open Pre-trained Transformer Language Models," 2022, arXiv: 2205.01068. [Online]. Available: https://arxiv.org/abs/2205.01068

[19] Y. Sheng, L. Zheng, B. Yuan, Z. Li, M. Ryabinin, D. Y. Fu, Z. Xie, B. Chen, C. Barrett, J. E. Gonzalez, P. Liang, C. Ré, I. Stoica, and C. Zhang, "FlexGen: High-Throughput Generative Inference of Large Language Models with a Single GPU," 2023, arXiv: 2303.06865.

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[21] Meta, "The Llama 4 herd: The beginning of a new era of natively multimodal AI innovation," Apr. 2025. [Online]. Available: https://ai.meta.com/blog/llama-4-multimodal-intelligence/

[22] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh,

D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33. Curran Associates, Inc., 2020, pp. 1877–1901. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[23] G. Gerganov and D. Devesa, "llama.cpp," Mar. 2023. [Online]. Available: https://github.com/ggml-org/llama.cpp

[24] Y. Song, Z. Mi, H. Xie, and H. Chen, "PowerInfer: Fast Large Language Model Serving with a Consumer-grade GPU," in *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, ser. SOSP '24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 590–606, event-place: Austin, TX, USA. [Online]. Available: https://doi.org/10.1145/3694715.3695964

[25] Z. Xue, Y. Song, Z. Mi, X. Zheng, Y. Xia, and H. Chen, "PowerInfer-2: Fast Large Language Model Inference on a Smartphone," 2024, arXiv: 2406.06282. [Online]. Available: https://arxiv.org/abs/2406.06282

[26] Intel, "Intel Optane DC Persistent Memory Product Brief," 2019. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/optane-dc-persistent-memory-brief.pdf

[27] J. Choe, "Intel's 2nd Generation XPoint Memory - Will it be worth the long wait ahead?" Apr. 2021. [Online]. Available: https://www.techinsights.com/blog/memory/intels-2nd-generation-xpoint-memory

[28] Intel, "DDR4/DDR-T DIMM Memory Interface," Jun. 2020. [Online]. Available: https://www.intel.com/content/www/us/en/docs/programmable/683867/current/ddr4-ddr-t-dimm-memory-interface.html

[29] J. Zhang, N. Beckwith, and J. J. Li, "GORDON: Benchmarking Optane DC Persistent Memory Modules on FPGAs," in *2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2021, pp. 97–105.

[30] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R. Dulloor, J. Zhao, and S. Swanson, "Basic Performance Measurements of the Intel Optane DC Persistent Memory Module," *arXiv:1903.05714 [cs]*, Aug. 2019, arXiv: 1903.05714. [Online]. Available: http://arxiv.org/abs/1903.05714

[31] I. B. Peng, M. B. Gokhale, and E. W. Green, "System evaluation of the Intel optane byte-addressable NVM," in *Proceedings of the International Symposium on Memory Systems - MEMSYS '19*. Washington, District of Columbia: ACM Press, 2019, pp. 304–315. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3357526.3357568

[32] J. Yang, J. Kim, M. Hoseinzadeh, J. Izraelevitz, and S. Swanson, "An Empirical Guide to the Behavior and Use of Scalable Persistent Memory," in *18th USENIX Conference on File and Storage Technologies (FAST 20)*. Santa Clara, CA: USENIX Association, Feb. 2020, pp. 169–182. [Online]. Available: https://www.usenix.org/conference/fast20/presentation/yang

[33] L. K. Archives, "Direct Access for files." [Online]. Available: https://www.kernel.org/doc/Documentation/filesystems/dax.txt

[34] J. Xu and S. Swanson, "NOVA: a log-structured file system for hybrid volatile/non-volatile main memories," in *Proceedings of the 14th Usenix Conference on File and Storage Technologies*, ser. FAST'16. USA: USENIX Association, 2016, pp. 323–338, event-place: Santa Clara, CA.

[35] M. Biesek, "Memkind Support For KMEM DAX Option," Jan. 2020. [Online]. Available: https://pmem.io/blog/2020/01/memkind-support-for-kmem-dax-option/

[36] Intel, "Intel Reports Second-Quarter 2022 Financial Results," Jul. 2022. [Online]. Available: https://www.intc.com/news-events/press-releases/detail/1563/intel-reports-second-quarter-2022-financial-results

[37] M. Du and M. L. Scott, "Buffered Persistence in B+ Trees," *Proc. ACM Manag. Data*, vol. 2, no. 6, Dec. 2024, place: New York, NY, USA Publisher: Association for Computing Machinery. [Online]. Available: https://doi.org/10.1145/3698801

[38] S. Karim, J. Wünsche, M. Kuhn, G. Saake, and D. Broneske, "NVM in Data Storage: A Post-Optane Future," *ACM Trans. Storage*, Apr. 2025, place: New York, NY, USA Publisher: Association for Computing Machinery. [Online]. Available: https://doi.org/10.1145/3731454

[39] Intel, "Migration from Direct-Attached Intel Optane Persistent Memory to CXL-Attached Memory," Nov. 2022. [Online]. Available: https://www.intel.com/content/dam/www/central-libraries/us/en/documents/2022-11/optane-pmem-to-cxl-tech-brief.pdf

[40] W. Calvert, "Intel, Google and others join forces for CXL interconnect," Mar. 2019. [Online]. Available: https://www.datacenterdynamics.com/en/news/intel-google-and-others-join-forces-cxl-interconnect/

[41] CXL Consortium, "Compute Express Link 1.1 Specification," Jun. 2019. [Online]. Available: https://computeexpresslink.org/

[42] M. Jung, "Hello bytes, bye blocks: PCIe storage meets compute express link for memory expansion (CXL-SSD)," in *Proceedings of the 14th ACM Workshop on Hot Topics in Storage and File Systems*, ser. HotStorage '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 45–51, event-place: Virtual Event. [Online]. Available: https://doi.org/10.1145/3538643.3539745

[43] S. Pei and R. Pitchumani, "CMM-H (CXL Memory Module – Hybrid): Rethinking Storage for the Memory-Centric Computing Era," Dec. 2023. [Online]. Available: https://semiconductor.samsung.com/us/news-events/tech-blog/rethinking-storage-for-the-memory-centric-computing-era/

[44] Samsung, "Samsung CXL Solutions – CMM-H," Mar. 2024. [Online]. Available: https://semiconductor.samsung.com/us/news-events/tech-blog/samsung-cxl-solutions-cmm-h/

[45] S.-P. Yang, M. Kim, S. Nam, J. Park, J.-y. Choi, E. H. Nam, E. Lee, S. Lee, and B. S. Kim, "Overcoming the Memory Wall with CXL-Enabled SSDs," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. Boston, MA: USENIX Association, Jul. 2023, pp. 601–617. [Online]. Available: https://www.usenix.org/conference/atc23/presentation/yang-shao-peng

[46] D. D. Sharma, "Compute Express Link®: An open industry-standard interconnect enabling heterogeneous data-centric computing," in *2022 IEEE Symposium on High-Performance Interconnects (HOTI)*, 2022, pp. 5–12.

[47] Wikipedia, "PCI Express link performance." [Online]. Available: https://en.wikipedia.org/wiki/PCI_Express#Comparison_table

[48] NVIDIA, "NVIDIA A100 Tensor Core GPU Architecture," 2020.

[49] NVIDIA, "nvbandwidth," Apr. 2022. [Online]. Available: https://github.com/NVIDIA/nvbandwidth

[50] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020. [Online]. Available: http://jmlr.org/papers/v21/20-074.html

[51] Z. Wang, X. Liu, J. Yang, T. Michailidis, S. Swanson, and J. Zhao, "Characterizing and Modeling Non-Volatile Memory Systems," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 496–508.

[52] V. Viswanathan, K. Kumar, T. Willhalm, S. Sakthivelu, and S. Srikanthan, "Intel Memory Latency Checker," May 2024. [Online]. Available: https://www.intel.com/content/www/us/en/developer/articles/tool/intelr-memory-latency-checker.html

[53] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Q-BERT: Hessian Based Ultra Low Precision Quantization of BERT," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, pp. 8815–8821, Apr. 2020. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/6409

[54] X. Wang, J. Liu, J. Wu, S. Yang, J. Ren, B. Shankar, and D. Li, "Performance Characterization of CXL Memory and Its Use Cases," in *2025 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2025.

[55] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, pp. 1135–1143, event-place: Montreal, Canada.

[56] Y. LeCun, J. Denker, and S. Solla, "Optimal Brain Damage," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2. Morgan-Kaufmann, 1989. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf

[57] X. Ma, G. Fang, and X. Wang, "LLM-pruner: on the structural pruning of large language models," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, ser. NIPS '23. Red Hook, NY, USA: Curran Associates Inc., 2023, event-place: New Orleans, LA, USA.

[58] M. Xia, T. Gao, Z. Zeng, and D. Chen, "Sheared LLaMA: Accelerating Language Model Pre-training via Structured Pruning," 2024, arXiv: 2310.06694. [Online]. Available: https://arxiv.org/abs/2310.06694

[59] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: training deep neural networks with binary weights during propagations," in *Proceedings of the 29th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, pp. 3123–3131, event-place: Montreal, Canada.

[60] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6869–6898, Jan. 2017, publisher: JMLR.org.

[61] C. Guo, J. Tang, W. Hu, J. Leng, C. Zhang, F. Yang, Y. Liu, M. Guo, and Y. Zhu, "OliVe: Accelerating Large Language Models via Hardware-Friendly Outlier-Victim Pair Quantization," in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ser. ISCA '23. New York, NY, USA: Association for Computing Machinery, 2023, event-place: Orlando, FL, USA. [Online]. Available: https://doi.org/10.1145/3579371.3589038

[62] A. Zadeh, I. Edo, O. Awad, and A. Moshovos, "GOBO: Quantizing Attention-Based NLP Models for Low Latency and Energy Efficient Inference," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2020, pp. 811–824. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/MICRO50266.2020.00071

[63] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica, "Efficient Memory Management for Large Language Model Serving with PagedAttention," in *Proceedings of the 29th Symposium on Operating Systems Principles*, ser. SOSP '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 611–626, event-place: Koblenz, Germany. [Online]. Available: https://doi.org/10.1145/3600006.3613165

[64] C. Hooper, S. Kim, H. Mohammadzadeh, M. W. Mahoney, Y. S. Shao, K. Keutzer, and A. Gholami, "KVQuant: Towards 10 Million Context Length LLM Inference with KV Cache Quantization," 2024, arXiv: 2401.18079. [Online]. Available: https://arxiv.org/abs/2401.18079

[65] Z. Liu, J. Yuan, H. Jin, S. H. Zhong, Z. Xu, V. Braverman, B. Chen, and X. Hu, "KIVI: a tuning-free asymmetric 2bit quantization for KV cache," in *Proceedings of the 41st International Conference on Machine Learning*, ser. ICML'24. JMLR.org, 2024, place: Vienna, Austria.

[66] Y. Jin, C.-F. Wu, D. Brooks, and G.-Y. Wei, "S^3: Increasing GPU Utilization during Generative Inference for Higher Throughput," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 18 015–18 027. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/3a13be0c5dae69e0f08065f113fb10b8-Paper-Conference.pdf

[67] B. Gao, Z. He, P. Sharma, Q. Kang, D. Jevdjic, J. Deng, X. Yang, Z. Yu, and P. Zuo, "Cost-Efficient Large Language Model Serving for Multi-turn Conversations with CachedAttention," in *2024 USENIX Annual Technical Conference (USENIX ATC 24)*. Santa Clara, CA: USENIX Association, Jul. 2024, pp. 111–126. [Online]. Available: https://www.usenix.org/conference/atc24/presentation/gao-bin-cost

[68] O. Patil, L. Ionkov, J. Lee, F. Mueller, and M. Lang, "Performance characterization of a DRAM-NVM hybrid memory architecture for HPC applications using intel optane DC persistent memory modules," in *Proceedings of the International Symposium on Memory Systems*, ser. MEMSYS '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 288–303, event-place: Washington, District of Columbia, USA. [Online]. Available: https://doi.org/10.1145/3357526.3357541

[69] R. S. Venkatesh, T. Mason, P. Fernando, G. Eisenhauer, and A. Gavrilovska, "Scheduling HPC Workflows with Intel Optane Persistent Memory," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2021, pp. 56–65.

[70] M. Weiland, H. Brunst, T. Quintino, N. Johnson, O. Iffrig, S. Smart, C. Herold, A. Bonanni, A. Jackson, and M. Parsons, "An Early Evaluation of Intel's Optane DC Persistent Memory Module and Its Impact on High-Performance Scientific Applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019, event-place: Denver, Colorado. [Online]. Available: https://doi.org/10.1145/3295500.3356159

[71] J. Ren, J. Luo, I. Peng, K. Wu, and D. Li, "Optimizing Large-Scale Plasma Simulations on Persistent Memory-Based Heterogeneous Memory with Effective Data Placement across Memory Hierarchy," in *Proceedings of the ACM International Conference on Supercomputing*, ser. ICS '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 203–214, event-place: Virtual Event, USA. [Online]. Available: https://doi.org/10.1145/3447818.3460356

[72] M. Hildebrand, J. Khan, S. Trika, J. Lowe-Power, and V. Akella, "AutoTM: Automatic Tensor Movement in Heterogeneous Memory Systems Using Integer Linear Programming," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 875–890. [Online]. Available: https://doi.org/10.1145/3373376.3378465

[73] H. A. Maruf, H. Wang, A. Dhanotia, J. Weiner, N. Agarwal, P. Bhattacharya, C. Petersen, M. Chowdhury, S. Kanaujia, and P. Chauhan, "TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, pp. 742–755, event-place: Vancouver, BC, Canada. [Online]. Available: https://doi.org/10.1145/3582016.3582063

[74] H. Li, D. S. Berger, L. Hsu, D. Ernst, P. Zardoshti, S. Novakovic, M. Shah, S. Rajadnya, S. Lee, I. Agarwal, M. D. Hill, M. Fontoura, and R. Bianchini, "Pond: CXL-Based Memory Pooling Systems for Cloud Platforms," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, pp. 574–587, event-place: Vancouver, BC, Canada. [Online]. Available: https://doi.org/10.1145/3575693.3578835

[75] W. Choi, J. Jeong, H. Jang, and J. Ahn, "GPU-Centric Memory Tiering for LLM Serving With NVIDIA Grace Hopper Superchip," *IEEE Computer Architecture Letters*, vol. 24, no. 1, pp. 33–36, 2025.

[76] NVIDIA, "NVIDIA GH200 Grace Hopper Superchip Architecture," 2023. [Online]. Available: https://resources.nvidia.com/en-us-grace-cpu/nvidia-grace-hopper

[77] Meta, "Introducing Llama 3.1: Our most capable models to date," Jul. 2024. [Online]. Available: https://ai.meta.com/blog/meta-llama-3-1/

[78] T. C. Schroeder, "Peer-to-Peer & Unified Virtual Addressing," Nvidia, 2011. [Online]. Available: https://developer.download.nvidia.com/CUDA/training/cuda_webinars_GPUDirect_uva.pdf

[79] M. Harris, "Unified Memory for CUDA Beginners," Jun. 2017. [Online]. Available: https://developer.nvidia.com/blog/unified-memory-cuda-beginners/

[80] D. Ganguly, Z. Zhang, J. Yang, and R. Melhem, "Interplay between hardware prefetcher and page eviction policy in CPU-GPU unified virtual memory," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 224–235, event-place: Phoenix, Arizona. [Online]. Available: https://doi.org/10.1145/3307650.3322224

[81] ——, "Adaptive Page Migration for Irregular Data-intensive Applications under GPU Memory Oversubscription," in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2020, pp. 451–461.

[82] D. Ganguly, R. Melhem, and J. Yang, "An Adaptive Framework for Oversubscription Management in CPU-GPU Unified Memory," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2021, pp. 1212–1217.

[83] S. Go, H. Lee, J. Kim, J. Lee, M. K. Yoon, and W. W. Ro, "Early-Adaptor: An Adaptive Framework for Proactive UVM Memory Management," in *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2023, pp. 248–258.

[84] C. Li, R. Ausavarungnirun, C. J. Rossbach, Y. Zhang, O. Mutlu, Y. Guo, and J. Yang, "A Framework for Memory Oversubscription Management in Graphics Processing Units," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 49–63, event-place: Providence, RI, USA. [Online]. Available: https://doi.org/10.1145/3297858.3304044

[85] S. Chien, I. Peng, and S. Markidis, "Performance Evaluation of Advanced Features in CUDA Unified Memory," in *2019 IEEE/ACM Workshop on Memory Centric High Performance Computing (MCHPC)*, 2019, pp. 50–57.

[86] J. Park, D. Jeong, and J. Kim, "UVMMU: Hardware-Offloaded Page Migration for Heterogeneous Computing," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–6.
[87] P. Markthub, M. E. Belviranli, S. Lee, J. S. Vetter, and S. Matsuoka, "DRAGON: Breaking GPU Memory Capacity Limits with Direct NVM Access," in *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2018, pp. 414–426.
[88] A. Thompson and C. Newburn, "GPUDirect Storage: A Direct Path Between Storage and GPU Memory," Aug. 2019. [Online]. Available: https://developer.nvidia.com/blog/gpudirect-storage/
[89] J. Zhang, D. Donofrio, J. Shalf, M. T. Kandemir, and M. Jung, "NVMMU: A Non-volatile Memory Management Unit for Heterogeneous GPU-SSD Architectures," in *2015 International Conference on Parallel Architecture and Compilation (PACT)*, 2015, pp. 13–24.
[90] H. Zhang, Y. Zhou, Y. Xue, Y. Liu, and J. Huang, "G10: Enabling An Efficient Unified GPU Memory and Storage Architecture with Smart Tensor Migrations," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 395–410, event-place: Toronto, ON, Canada. [Online]. Available: https://doi.org/10.1145/3613424.3614309
[91] S. Pandey, A. K. Kamath, and A. Basu, "GPM: Leveraging Persistent Memory from a GPU," in *Proceedings of the Twenty-Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '22. Lausanne, Switzerland: Association for Computing Machinery, 2022. [Online]. Available: https://doi.org/10.1145/3503222.3507758
[92] ——, "Scoped Buffered Persistency Model for GPUs," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ser. ASPLOS 2023. New York, NY, USA: Association for Computing Machinery, 2023, pp. 688–701, event-place: Vancouver, BC, Canada. [Online]. Available: https://doi.org/10.1145/3575693.3575749
[93] J. Hong, S. Cho, G. Park, W. Yang, Y.-H. Gong, and G. Kim, "Bandwidth-Effective DRAM Cache for GPU s with Storage-Class Memory," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2024, pp. 139–155.
[94] B. Wang, B. Wu, D. Li, X. Shen, W. Yu, Y. Jiao, and J. S. Vetter, "Exploring hybrid memory for GPU energy efficiency through software-hardware co-design," in *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, 2013, pp. 93–102.
[95] S. Chen, F. Zhang, L. Liu, and L. Peng, "Efficient GPU NVRAM Persistence with Helper Warps," in *Proceedings of the 56th Annual Design Automation Conference 2019*, ser. DAC '19. New York, NY, USA: Association for Computing Machinery, 2019, event-place: Las Vegas, NV, USA. [Online]. Available: https://doi.org/10.1145/3316781.3317810
[96] S. Chen, L. Liu, W. Zhang, and L. Peng, "Architectural Support for NVRAM Persistence in GPUs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 5, pp. 1107–1120, 2020.
[97] Z. Lin, M. Alshboul, Y. Solihin, and H. Zhou, "Exploring Memory Persistency Models for GPUs," in *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2019, pp. 311–323.

APPENDIX

A. Abstract

This artifact appendix describes how to reproduce key results from the paper. The artifact includes a modified version of FlexGen that implements the two proposed weight allocation schemes, along with the author collected data and helper scripts to plot the figures described in the paper.

B. Artifact check-list (meta-information)

- **Algorithm:** HeLM and All-CPU.
- **Program:** FlexGen.
- **Model:** OPT-30B and OPT-175B.
- **Data set:** c4/realnewslike.
- **Hardware:** Modern x86-64 based CPU with at least 100 GB DRAM+Optane memory and a CUDA-compatible GPU with at least 4 GB memory.
- **Metrics:** Time to first token (TTFT), time between tokens (TBT), throughput (tokens/second), compute/communication latency overlap.
- **How much disk space required (approximately)?:** 450 GB.
- **How much time is needed to prepare workflow (approximately)?:** 2-4 hours.
- **How much time is needed to complete experiments (approximately)?:** 4 days.
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** Apache-2.0.
- **Data licenses (if publicly available)?:** OPT-175B license (model) and Apache-2.0 (data set).
- **Archived (provide DOI)?:** https://doi.org/10.5281/zenodo.16905746.

1) How to access: The artifact is available on both GitHub and Zenodo at the following links..

- Zenodo: https://zenodo.org/records/16905746
- GitHub: https://github.com/Sacusa/FlexLLMGen

2) Hardware dependencies: Recent x86-64 based CPU with at least 100 GB of heterogeneous memory. Our system, for instance, combines DRAM and Intel Optane. A CUDA-compatible GPU with at least 4 GB of onboard memory.

3) Software dependencies: PyTorch >= 1.12.

4) Data sets: c4/realnewslike: https://huggingface.co/datasets/allenai/c4.

5) Models: FlexGen automatically downloads most model weights. The weights for OPT-175B can be obtained at https://huggingface.co/Neko-Institute-of-Science/OPT-175B-NumPy

C. Installation

1) Download the input data set and models from the links above using HuggingFace CLI.
2) Follow the instructions in `README.md` to set up FlexGen.

D. Evaluation and expected results

The raw data used for the figures in this paper can be found in `output/` directory. The scripts in `output/scripts` can be used the generate the figures in PDF format.

E. Methodology

Submission, reviewing and badging methodology:

- https://www.acm.org/publications/policies/artifact-review-and-badging-current
- https://cTuning.org/ae