

Using HugePages with PyTorch for Faster LLM Inference

Abstract:

LLMs scale, optimizing KV cache and storage is critical to boosting inference performance. This session explores recent storage optimizations in inference frameworks and how they affect memory allocation and registration in PyTorch. A key optimization is the use of HugePages (HPs) to streamline the scatter-gather process when writing to storage. During scatter-gather, the kernel gathers listed memory pages, translates virtual addresses to physical DRAM addresses, and sends them to the storage controller. HPs reduces the number of pages and scatter-gather entries, which improves efficiency and complies with storage systems that require shorter scatter-gather lists.

The shift to HP-based memory allocation, which enables the deployment of NVIDIA DOCA Memos (NVIDIA CMX) and NIXL storage-bound plugins, spurred activity in benchmarking and evaluating different approaches to memory allocation and their effects on storage performance.

The Theory:

During transfer, a list of memory locations are used to transfer kv caches to storage. That list goes down to a resolution of 4k pages. In a 128k context with the selected model, it brings to a count of over 2 million pages. Using HugePages brings that list down to 5k 2MB pages. Every time a transfer is requested, that list of 2M vs 5K comes into play

In a TLB miss, the effects are more pronounced with a faster pagewalk over DRAM, 512x less pages to search on linear access.

The Setup

vLLM was patched to use HugePages when allocating memory for PyTorch use in offload mode instead of shared memory (shm).

1. Background: the offload path

vLLM's OffloadingConnector (TieringOffloadingSpec) extends the GPU prefix cache with a CPU-primary tier plus optional secondary tiers. Completed KV blocks are copied off the GPU as they are produced; on a later request with the same prefix, blocks are looked up and promoted back. Critically for the storage question: only the CPU primary tier has direct GPU access — every GPU ↔ filesystem transfer is staged through the CPU tier.



This staging means that all data blocks are on 4k pages which is a potential explosion of page table entries per transfer, x 512 over a 2MB HugePage setup.

The Benchmarks

A 10.73 GB region was reserved for buffering between the fs offload tier and the GPU

The working set for the benchmarks were all greater than the 10.73 GB region, so the KV Caches were all loaded from the storage tier during prefill.

vLLM was used without prefix caching in the VRAM layer to force the storage scenario (--no-enable-prefix-caching parameter)

Component	Value
Hardware	8x NVIDIA B200 183 GB (only 2 or 6 used), 2 TB host DRAM, NVMe md0 for fs tier
Model	Qwen/Qwen3-30B-A3B-Instruct-2507-FP8, TP=1, DP=6, max_model_len=130,000, KV fp8
Connector	OffloadingConnector / TieringOffloadingSpec, cpu_bytes_to_use=10.73 GB per rank, block_size=256 tokens, LRU, fs secondary tier
shm config	default backend → /dev/shm/vllm_offload_*.mmap (container shm_size=32gb, ipc=host)
hugetlbfs config	cpu_memory_backend=hugetlbfs, cpu_memory_path=/hugepages, cpu_hugepage_block_size=2MB; host: vm.nr_hugepages=32,768 (DP6),
Workload	aiperf (KVCacheSpeedTest): per context length, a prefill phase warms the offload tiers with N unique prompts, then a 300 s measured phase re-issues them (sequential dataset, fixed seed), ctx/prompts: 16k/100, 32k/50, 64k/25, 128k/12; concurrency sweep 1, 3, 6, and 12. State (shm files, hugepage files, fs tier) scrubbed between runs

Results

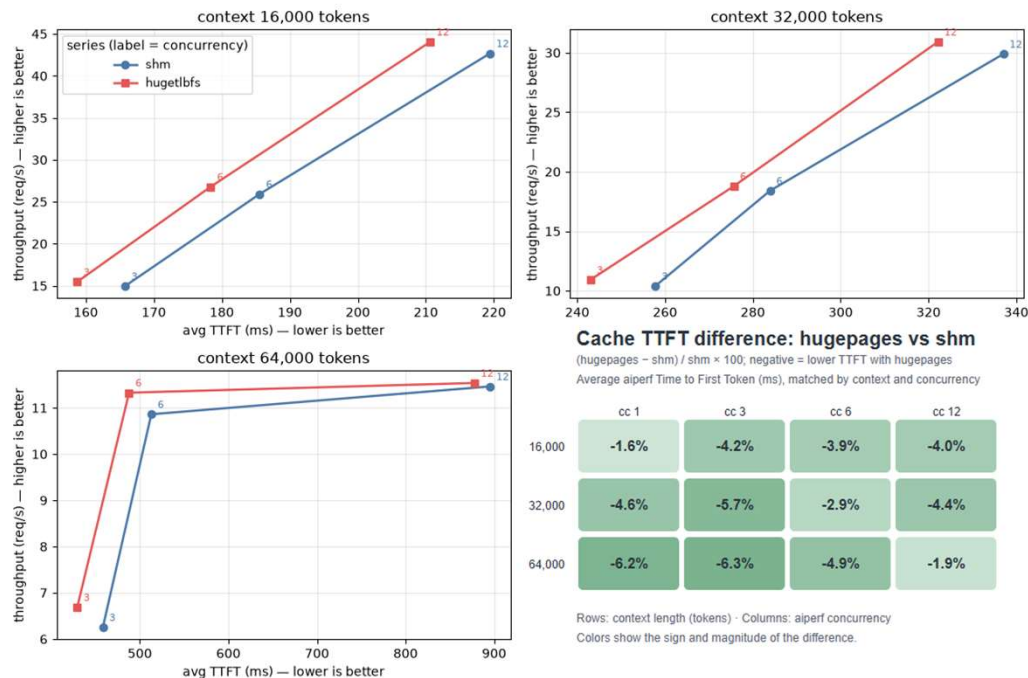
Concurrency sweep (DP=6): Pareto fronts

Avg TTFT vs request throughput per context length; point labels = aiperf concurrency (1, 3, 6, 12). The hugetlbfs front sits up-and-left of shm at every point.

Interpretation

Using HugePages instead of standard memory pages in this setup decreased TTFT 1-6% and increased throughput in the same range. HugePages with PyTorch yields benefit for inference with only a setup cost in this setup. The improvement specifically helped storage load results.

KV-cache-hit serving: Pareto fronts per context length (each point = one measured run)



Conclusions

We've explored how vLLM, one of the more popular datacenter inference engines handles memory allocation with PyTorch and showed how to make it register using HugePages streamlining the scatter-gather process by reducing the number of pages needed for transfers.

Using HugePages with PyTorch in vLLM with its distributed offload mode yields a decrease in TTFT when working alongside storage offload. The results are convincing and easy to implement with this example showing the steps forward in other similar domains.