

Supplementary README for HiFC: High-efficiency Flash-based KV Cache Swapping for Scaling LLM Inference

This document describes the supplementary materials provided with our NeurIPS 2025 submission. It includes implementation details, configuration files, and procedures to reproduce the results presented in the main paper.

1 Directory Structure

Table 1: Important Modified Files for HiFC

```
supplement/  
  HiFC_README.pdf  
  HiFC_Appendix_F_G.pdf  
  requirements-hifc.txt  
  conda-environment.yml  
  hifc_vllm/  
    vllm/  
      config.py  
      attention/ops/paged_attn.py  
      core/block_manager.py  
      core/scheduler.py  
      core/block/cpu_gpu_block_allocator.py  
      core/block/naive_block.py  
      core/block/prefix_caching_block.py  
      engine/arg_utils.py  
      worker/cache_engine.py  
      csrc/cache_kernels.cu  
      benchmarks/HiFC_Experiments/  
        benchmark_scripts  
        benchmark_logs
```

- **config.py / class FlashCacheConfig:** Defines HiFC-specific parameters such as swap file paths, thread count, and I/O granularity. Provides a centralized interface to initialize and validate Flash Cache (FC) settings.

- **cache_engine.py / class CacheEngine:** Manages dynamic KV cache movement between GPU HBM and Flash Cache using GDS and cuFile. Issues asynchronous multi-threaded swap-in/out requests and synchronizes I/O completion events. Byte offsets are computed per layer and KV type, and passed to the swap function. To maintain SSD LBA continuity, a continuous range allocation algorithm is applied, preserving SSD performance and lifespan.
- **block_manager.py / class SelfAttnBlockSpaceManager:** Handles logical block allocation for self-attention layers and interfaces with the FC block allocator. The allocator mimics GPU-like block tables, while physical mapping info (GPU, CPU, SSD) is passed to the engine. Although HiFC uses Flash blocks primarily, fallback to DRAM blocks is also supported.
- **block_manager.py / class CpuGpuBlockAllocator:** Allocates physical FC block space using a simple append-only strategy. Supports both naive and prefix-caching block modes for flexible allocation.
- **scheduler.py / class Scheduler:** Decides swap timing and target sequences based on GPU memory pressure and scheduling priority. During decoding, it selects victim sequences for eviction to Flash Cache. Swapped sequences are reloaded when GPU space becomes available. Swap overhead is partially masked by GPU parallelism, reducing visible performance degradation.
- **cache_kernels.cu / function swap_flash_blocks:** Implements the CUDA kernel for KV cache transfer between GPU and Flash Cache using cuFile APIs. Supports up to 16 parallel threads for GDS I/O throughput optimization. KV tensors are allocated contiguously per layer and KV type, and I/O commands are issued with 4KB-aligned offsets to maximize SSD sequential throughput. GDS behavior is GPU-dependent; refer to the README for system configuration.

2 Experimental and Build Environment

2.1 System and Software Condition

- GPU: NVIDIA A100 or GDS supported GPU
- Test SSD: 1TB NVMe Gen4 SSD (pseudo-SLC)
- PyTorch: 2.5.1
- CUDA: 12.3
- GDS: 1.8.1.2
- libcufile: 2.12 with nvidia-fs: 2.22
- OS: Ubuntu 22.04, Kernel 5.15+

2.2 Conda and PIP Setting

- **Conda:** hifc_vllm/conda_environment.yml
- **PIP:** hifc_vllm/requirements-hifc.txt:

2.3 Install GDS

Reference NVIDIA's official GDS guide.

[NVIDIA GPUDirect Storage Installation and Troubleshooting Guide](#)

2.4 File System and Initialize Test SSD

```
1 #!/bin/bash
2 # Check NVMe SSD list
3 ~/hifc_vllm$ nvme list
4 Node              Usage              Format
5 -----
6 /dev/nvme0n1      1.00TB/1.00TB    512 Bytes + 0 B
7
8 # NVMe format 4KB LBA unit size
9 ~/hifc_vllm$ nvme format /dev/nvme0n1 -l 1 -f
10
11 # Check NVMe format type
12 ~/hifc_vllm$ nvme list
13 Node              Usage              Format
14 -----
15 /dev/nvme0n1      1.00TB/1.00TB    4 KiB + 0 B
16
17 # Make XFS file system for GDS
18 ~/hifc_vllm$ mkfs.xfs -f -d su=1m,sw=1 /dev/nvme0n1
19
20 # Make a directory for flash cache
21 ~/hifc_vllm$ mkdir /mnt/ssd0
22
23 # Mount flash cache directory
24 ~/hifc_vllm$ mount -t xfs -o noatime,logbufs=8,allocsize=1m /dev/nvme0n1
25 /mnt/ssd0
26
27 # Make a large flash cache file in target directory
28 ~/hifc_vllm$ fallocate -l 900G /mnt/ssd0/fc0
29
30 # Check sequential LBA mapping for flash cache
31 ~/hifc_vllm$ hdparm --fibmap /mnt/ssd0/fc0
32 /mnt/ssd0/fc0:
33 filesystem blocksize 4096, begins at LBA 0; assuming 512 byte sectors.
34 byte_offset  begin_LBA    end_LBA      sectors
35 0           11329216    28106423    16777208
36 8589930496  28106424    44883631    16777208
37 17179860992 44883632    61660839    16777208
38 25769791488 67103424    69200599    2097176
39 26843545600 69200600    85975767    16775168
40 35432431616 85975768    102750935   16775168
41 ....
```

2.5 Build vLLM and Developer Mode

- Reference vLLM's official install guide. [vLLM: NVIDIA GPU - Full Build](#)

- Simple build method.
`: /hifc_vllm$ export MAX_JOBS=8`
`: /hifc_vllm$ pip install -e .`
`: /hifc_vllm$ python setup.py develop --user`

3 Benchmark LLM Inference Performance with HiFC

3.1 HiFC Engine Configure

- **-swap-fc-space:** 0 to the full size of the test SSD (in GiB), default is 0 (disabled).
 - Please create a swap file on the SSD larger than the intended swap space.
 - A positive integer specifies the size (in GiB) of the Flash Cache (FC) swap space. A value of 0 disables HiFC.
 - When using pseudo-SLC SSDs, setting a value less than or equal to the pSLC region size improves performance and endurance.
- **-num-ssd-for-fc:** 0 or 1, default is 0 (disabled).
 - Make sure to initialize the SSD before enabling this option. See the SSD setup section.
 - A positive integer specifies the number of SSD-backed swap files to use as flash cache.
 - Only one flash cache file is supported per SSD.
 - The file paths for the cache can be set in the `FlashCacheConfig` class.
 - Currently, only one SSD or one RAID0 group is supported.
- **-num-threads:** 0 to 16, default is 0 (disabled).
 - A positive integer allocates the specified number of CPU threads for GDS I/O operations.
 - Check your system’s available threads before setting this value.
 - For server-grade CPUs with lower clock speeds, 4 or more threads are recommended.
- **-io-size-mb:** 0 to 32, default is 0 (disabled).
 - A positive integer determines the maximum I/O transfer size (in MiB) for each GDS operation.
 - This value is applied to the continuous range merge algorithm, which selects I/O size based on the contiguity of source/destination blocks.
 - The actual maximum I/O size may depend on the GPU configuration and GDS version.
 - Refer to the `/etc/cufile.json` configuration file for the `max_direct_io_size_kb` setting.

3.2 Benchmark Scripts

- `bm_swap_throughput_base.sh`: Base benchmark script for TPS.
- `bm_swap_throughput_inc_prompts.sh`: Benchmark TPS by increasing prompts(batch size)
- `bm_swap_throughput_input_length.sh`: Benchmark TPS by increasing input length.
- `bm_swap_throughput_output_length.sh`: Benchmark TPS by increasing output length.
- `bm_swap_throughput_length_max_seq.sh`: Benchmark TPS by max sequence number. (5K fixed)
- `bm_swap_throughput_length_block_size.sh`: Benchmark TPS by block sizes.

4 License and Attribution

This supplementary code is based on the open-source project **vLLM**, which is licensed under the GNU Lesser General Public License v3.0 (LGPL-3.0). The original repository is available at:

<https://github.com/vllm-project/vllm>

We include a copy of the original `LICENSE` file in this supplement.

Modifications in this work include:

- High-efficiency Flash-based KV Cache swapping (HiFC)
- Direct GPU-to-SSD GDS interface
- Contiguous block allocator for flash access locality
- Benchmark scripts for long-context inference

This supplement is provided exclusively for NeurIPS 2025 reviewing and academic reproducibility purposes. These modifications are licensed under the GNU LGPL-3.0.