CMSC828G: Special Topics in Systems for Machine Learning

KV Cache Compression II

A system-perspective for KV Cache Management

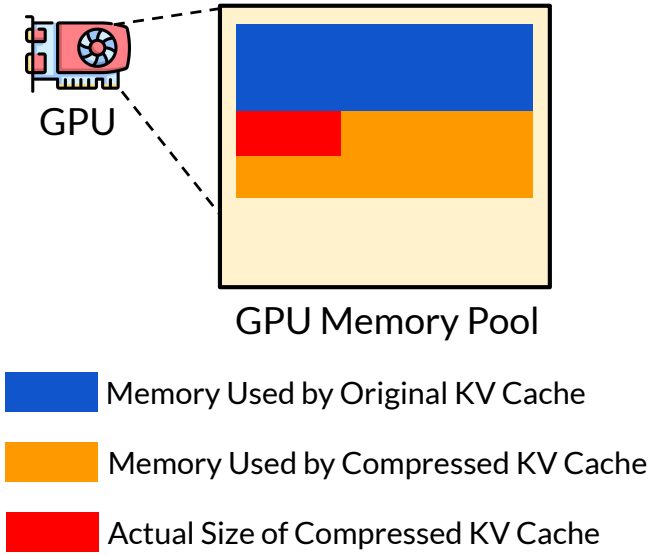
Instructor: Yafan Huang



DEPARTMENT OF
COMPUTER SCIENCE

System Problems for KV Cache

- Life cycle for KV cache: generate -> use -> retain -> reuse / evict.
- System abstraction for KV Cache management:
  - Memory allocation and sharing
  - Cache retention and eviction
  - Data placement and movement
  - Distributed cache management
- Compression can be used across problems/stacks.



KV Cache as a Data Object

- Dynamic: KV cache grows as new tokens are generated.
- Reusable: Previously computed KV may be shared across requests.
- Memory-intensive: KV competes with model weights and other requests for GPU memory.
- Movable: KV can be stored, transferred, or recomputed when needed.

- In system research, it is critical to understand the features of your object.

Direction 1: Dynamic KV Cache Allocation

- Challenge: Request lengths are unpredictable and KV cache grows dynamically.
  - Contiguous allocation can lead to over-reservation and fragmentation.
- Solution: Paged memory allocation.
  - Divide KV cache into fixed-size blocks.
  - Map logical token positions to physical memory blocks.
- Tradeoff: Block size vs. memory utilization and management overhead.

Direction 2: Making KV Cache Shareable

- Challenge: Multiple requests may use identical KV states.
  - Duplicating KV wastes GPU memory.
- Solution: Separate logical KV references from physical storage.
  - Identify reusable blocks through content-based indexing.
  - Allow multiple requests to reference the same physical blocks.
- Tradeoff: Reuse must preserve the original computation semantics.

Direction 3: Cache Retention and Eviction

- Challenge: GPU memory cannot retain all previously computed KV caches.
- Cache admission: Which KV blocks are worth keeping after a request?
- Cache eviction: Which inactive blocks should be removed under memory pressure?
- Possible policies can be learnt from system research:
    - Recency-based (LRU), frequency-based (LFU).
    - Reuse- and cost-aware policies.

Direction 4: KV Cache Across the Memory Hierarchy

- Challenge: GPU HBM provides high bandwidth but limited capacity.
- Solution: Hierarchical KV cache storage.
  - L1, GPU HBM/GDDR: active and latency-critical KV.
  - L2, CPU DRAM: larger, lower-cost cache capacity.
  - L3, SSD/Remote memory: larger-scale persistent or shared caching.
- Operations: Offload, prefetch, and on-demand loading.
- Tradeoff: Storage capacity vs. data access latency.

Direction 5: Data Movement vs. Recomputation

- Challenge: Retrieving cached KV may be slower than recomputing it.
- Data retrieval cost:
    - Depends on cache size, bandwidth, and storage/transfer overhead.
- Recomputation cost:
    - Depends on the number of tokens and available compute resources.
- Key question: When should we load cached KV, and when should we recompute?
- This is a traditional HPC question: computation vs. communication.

Direction 6: Cache-Aware Scheduling and Routing

- Challenge: KV caches and available GPU compute are distributed across workers.
- Load-aware routing: Prefer less-loaded workers.
- Cache-aware routing: Prefer workers holding reusable KV.
- Tradeoff: Queueing delay vs. cache locality and data movement cost.
- This brings new challenges to design effective communication strategies for cache-aware LLM serving on distributed systems.

Takeaways

- Memory management: Allocate and share KV efficiently.
- Cache management: Decide what to retain or evict.
- Data management: Place, retrieve, or recompute KV.
- Distributed management: Balance cache locality and compute load.

Student Presentations

- [SIGCOMM’24] CacheGen: KV Cache Compression and Streaming for Fast Large Language Model Serving
- [EuroSys’25] CacheBlend: Fast Large Language Model Serving for RAG with Cached Knowledge Fusion