Efficient Memory Management for Large Language Model Serving with PagedAttention

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, Ion Stoica

Presented by Yuqi Xue
Feb 15, 2024

Paper Published at SOSP’23. Many slide pages are borrowed from the MLsys Seminar @SG talk: https://www.youtube.com/watch?v=Oq2SN7uutbQ
The Era of LLMs

ChatGPT

GitHub Copilot
LLM-Powered Services
LLM-Powered Services

- Programming: GitHub, Copilot, tabnine
- Dev tools: Debug, warp, cogram
- Chat: ChatGPT, MessageBird, Sapling
- Copywriting: cohere, copy.ai, HyperWrite
- BizOps: viable, Enterpret, casetext

LLM Endpoints or Hosted LLM servers
Serving LLMs is extremely expensive

- LLMs run on high-end GPUs such as NVIDIA A100
- Each GPU can only serve a handful of requests per second
  - For LLaMA-13B and moderate-size inputs, one A100 can process < 1 requests per second
- A ton of GPUs are required for production-scale LLM services
Inference process of LLMs

Input

Artificial Intelligence is
Inference process of LLMs

Output

Input

Artificial Intelligence is
Inference process of LLMs

Output

Layer N
...
Layer 1

Input

Artificial Intelligence is

the
Inference process of LLMs

• Repeat until the sequence
  • Reaches its pre-defined max length (e.g., 4K tokens)
  • Generates an EOS (end of sequence) token
KV Cache

Input:
- Artificial Intelligence is

Output:
- The
- Future

Layer N

Layer 1

Weights:
- Artificial: -1.1, 0.5, 0.4
- Intelligence: -0.7, 0.1, -0.2
KV Cache

Output

Input
KV Cache

Output

KV Cache

Input
KV Cache

20 KB / layer / token
= 800 KB / token (LLaMA-13B)
KV Cache

**Step 1**

**Without cache**

- \( Q \) (Query Token 1)
- \( K^T \) (Key Token 1)
- \( QK^T \)
- \( V \) (Value Token 1)
- Attention

\[
\begin{align*}
(1, \text{emb}_\text{size}) & \times (\text{emb}_\text{size}, 1) = (1, 1) \\
& \times (1, \text{emb}_\text{size}) = (1, \text{emb}_\text{size})
\end{align*}
\]

**With cache**

- \( Q \) (Query Token 1)
- \( K^T \) (Key Token 1)
- \( QK^T \)
- \( V \) (Value Token 1)
- Attention

\[
\begin{align*}
(1, \text{emb}_\text{size}) & \times (\text{emb}_\text{size}, 1) = (1, 1) \\
& \times (1, \text{emb}_\text{size}) = (1, \text{emb}_\text{size})
\end{align*}
\]

- Values that will be masked
- Values that will be taken from cache

Animation source: https://medium.com/@joaolages/kv-caching-explained-276520203249
Key Insight

Parameters (26GB, 65%)  KV Cache (>30%)  Others

13B LLM on A100-40GB

Graph showing memory usage (GB) and throughput (tokens/s) against batch size (# requests).
Memory Waste in KV Cache

- **Internal fragmentation**: over-allocated due to the unknown output length
- **Reservation**: not used at the current step, but used in the future
- **External fragmentation**: due to different sequence lengths
Memory Waste in KV Cache

Only 20% - 40% of KV cache is utilized to store token states
vLLM: Efficient Memory Management for LLM Inference
KV Block

• A fixed-size contiguous chunk of memory that stores the tokens’ KV states
• Similar to the concept of a memory page
PagedAttention

- Manages KV cache in block granularity instead of sequence (i.e., request) granularity
- Allows storing logically contiguous KV blocks in non-contiguous physical memory

Example sequence: “Four score and seven years ago our fathers brought forth”
Logical-to-Physical KV Block Translation

Prompt: “Four score and seven years ago our”

Outputs: “fathers” → “brought” → …
Serving Multiple Requests
Memory Efficiency of vLLM

• Minimizes **internal fragmentation**
  • Only happens at the last block of a sequence
  • # wasted tokens per sequence < block size
    • Seq len: $O(100) – O(1000)$ tokens
    • Block size: 16 or 32 tokens
Dynamic Block Mapping Enables Sharing

E.g.) Parallel sampling

The future of cloud computing is intertwined with the advancement of artificial intelligence (AI). ... likely to be characterized by several key trends: ...

bright and poised for further growth and transformation. Here's why: ...

Prompt

Multiple outputs
KV Block Sharing for Parallel Sampling

Physical KV blocks

Copy-on-write

Ref count: 2 → 1
KV Block Sharing for Beam Search

![Diagram showing KV Block Sharing for Beam Search]

Candidate Sequences: A, C

Position 1
- A
- C

Position 2
- AB
- AE

Position 3
- ABC
- AED

KV Block Sharing for Beam Search

- Similar to process `fork` and `kill`
• Similar to shared libraries in OS
vLLM System Architecture & Implementation
Evaluation

System configuration.

<table>
<thead>
<tr>
<th>Model size</th>
<th>13B</th>
<th>66B</th>
<th>175B</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPUs</td>
<td>A100</td>
<td>4xA100</td>
<td>8xA100-80GB</td>
</tr>
<tr>
<td>Total GPU memory</td>
<td>40 GB</td>
<td>160 GB</td>
<td>640 GB</td>
</tr>
<tr>
<td>Parameter size</td>
<td>26 GB</td>
<td>132 GB</td>
<td>346 GB</td>
</tr>
<tr>
<td>Memory for KV cache</td>
<td>12 GB</td>
<td>21 GB</td>
<td>264 GB</td>
</tr>
<tr>
<td>Max. # KV cache slots</td>
<td>15.7K</td>
<td>9.7K</td>
<td>60.1K</td>
</tr>
</tbody>
</table>

Figure 11. Input and output length distributions of the (a) ShareGPT and (b) Alpaca datasets.
vLLM Improves Inference Throughput by Enabling Larger Batch Size

Figure 12. Single sequence generation with OPT models on the ShareGPT and Alpaca dataset.
vLLM Improves Inference Throughput by Enabling Larger Batch Size

**Figure 13.** Average number of batched requests when serving OPT-13B for the ShareGPT (2 reqs/s) and Alpaca (30 reqs/s) traces.
Memory Saving of vLLM

(a) Parallel sampling

(b) Beam search

Figure 15. Average amount of memory saving from sharing KV blocks, when serving OPT-13B for the Alpaca trace.
Takeaways

• Strength
  • Interesting observation on the KV cache memory inefficiency
  • Analogy between KV cache management and OS paging
  • Open-source implementation

• Weakness
  • Cannot fundamentally improve inference latency
  • For multi-chip execution, vLLM assumes attention heads are sharded
    • If even a single attention head is too large, or we want to split it across multiple chips to improve latency, how can vLLM support sharding the KV cache?
  • The fundamental bottlenecks faced by LLM serving, memory capacity due to large model weights, and memory bandwidth due to low compute intensity of auto-regressive decoding, remain unsolved.
    • Speculative decoding?
    • New model architectures? (e.g., SSM, Mamba)
    • New hardware/architectural innovations? (e.g., processing-in-memory, NVIDIA's new patent that proposes stacking HBM dies on the processor die to expose a wider mem interface)
Q & A