Machine Learning Model Management and Inference (MLMMI)
08 KV Cache Management

Dr. Arnab Phani
BIFOLD, TU Berlin
DEEM Lab

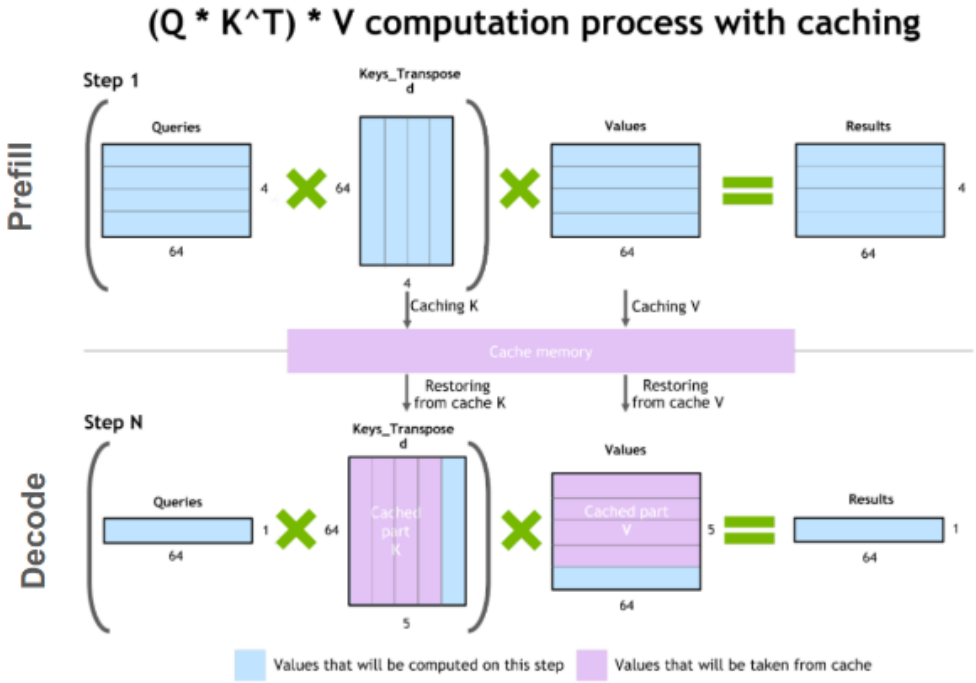
24.06.2026

Agenda

- **KV Cache Memory Management**
- **Prompt Cache**
- **Long-Context LLM Serving**
- **Quantization for KV Cache**

Efficient Management of KV Cache

KV Cache Recap



Self-attention:

$$Q = XW^Q; K = XW^K; V = XW^V$$

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Attention head:

$$H_i = \text{Attention}(XW_i^Q, XW_i^K, XW_i^Q)$$

$$\text{Multi-Head Attention} = \text{Concat}(H_1, H_2, ..., H_h)W^O$$

- **Overview**
  - KV cache stores Key (K) and Value (V) tensors computed at each layer to avoid recomputation
  - Prefill stores computed KVs of input sequence in KV cache
  - In each iteration in decode phase each new token only needs to attend to cached KV states + the latest token

* https://developer.nvidia.com/blog/mastering-llm-techniques-inference-optimization

KV Cache Management is an Active Research Area

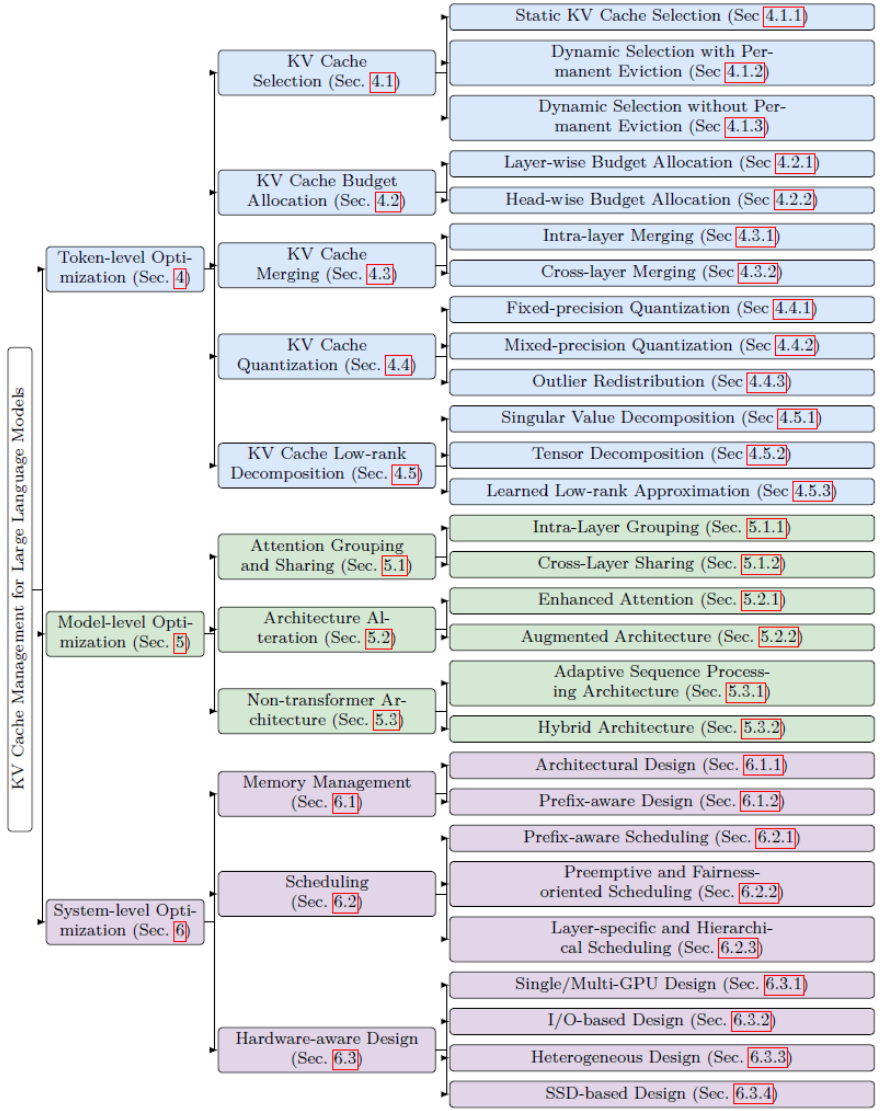


Figure 2: Taxonomy of KV Cache Management for Large Language Models.

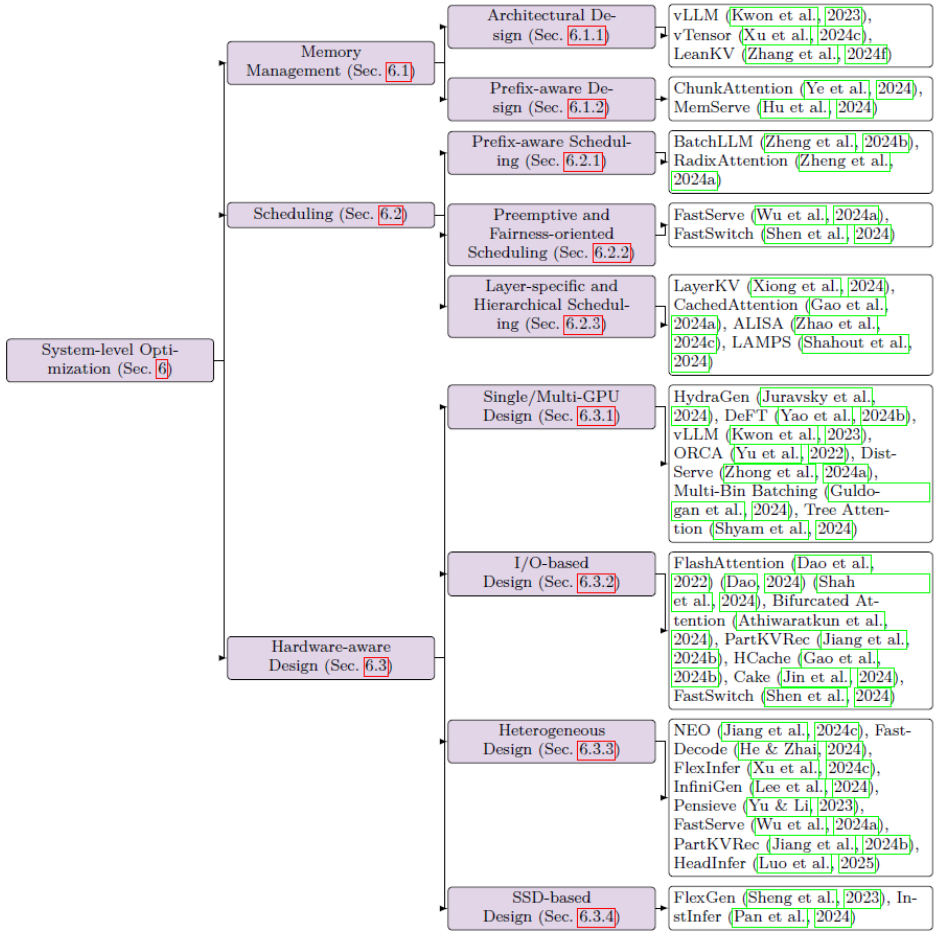
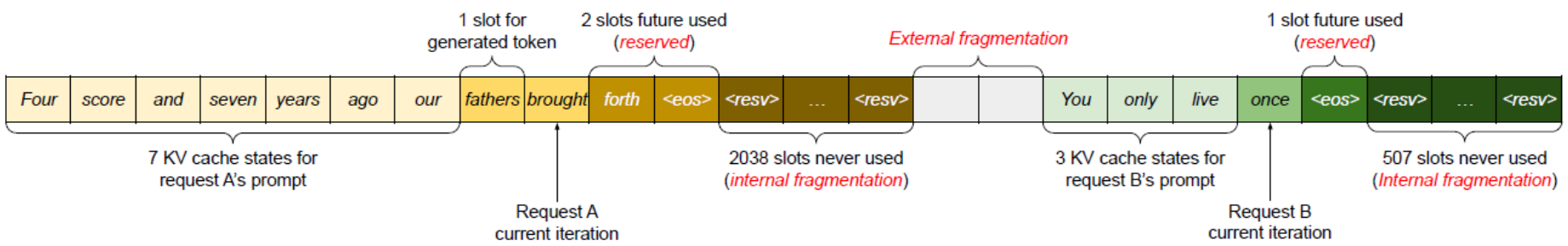
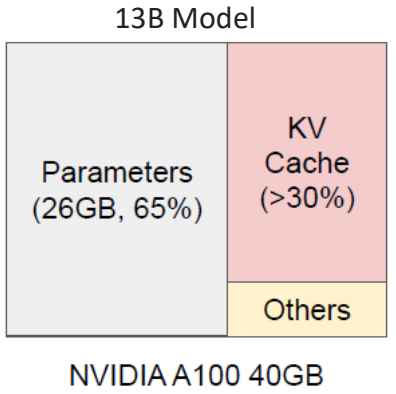


Figure 7: Taxonomy of the System-level Optimization for KV Cache Management.

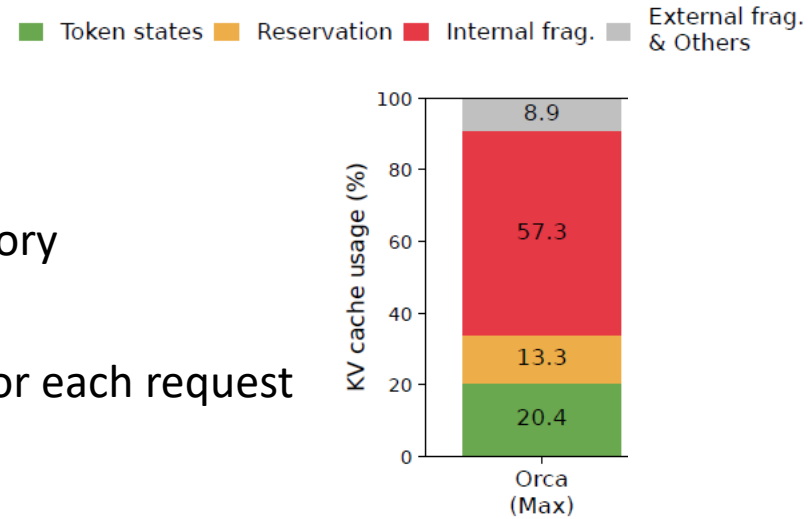
* A Survey on Large Language Model Acceleration based on KV Cache Management. 2025.

KV Cache Memory Challenges





- **Contiguous KV Cache**
  - Model takes constant memory
  - Large KV cache significantly limits the batch size due to limited GPU memory
  - KV cache grows and shrinks dynamically
  - Prior work pre-allocate a contiguous memory chunk for entire KV cache for each request
  - Problem: memory waste (reserved, internal and external fragmentations)



* Efficient Memory Management for Large Language Model Serving with PagedAttention. SOSP 2023.

PagedAttention

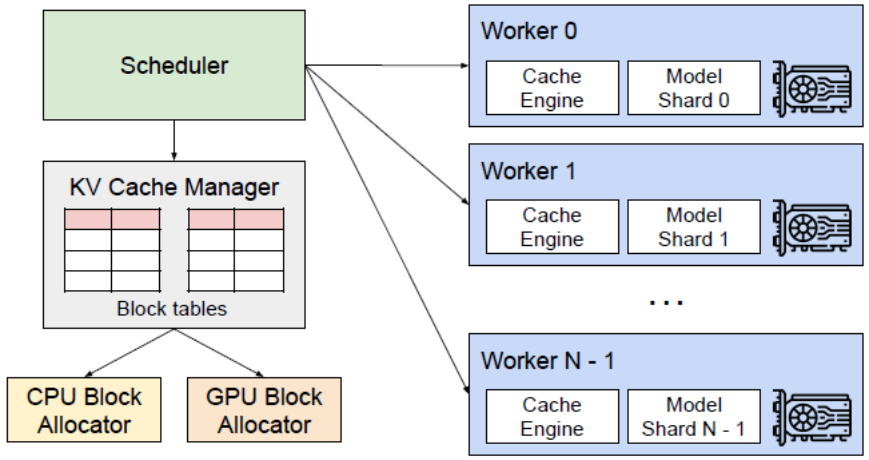


Figure 4. vLLM system overview.

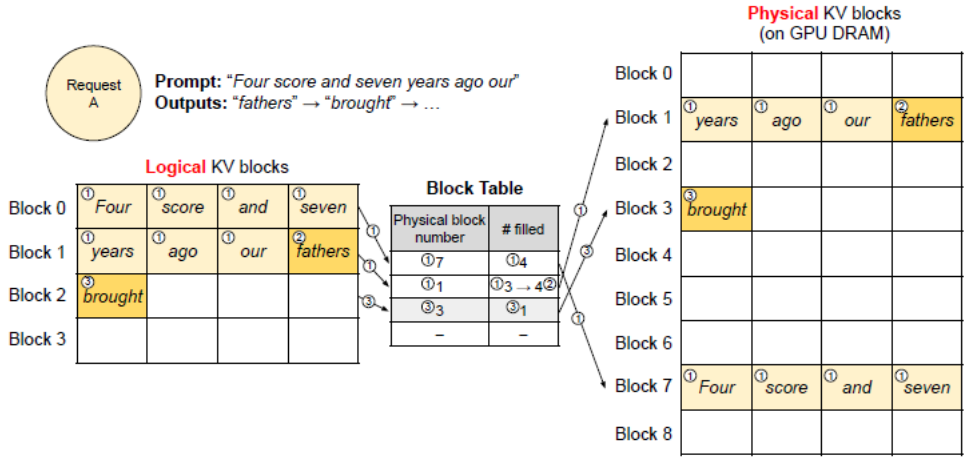
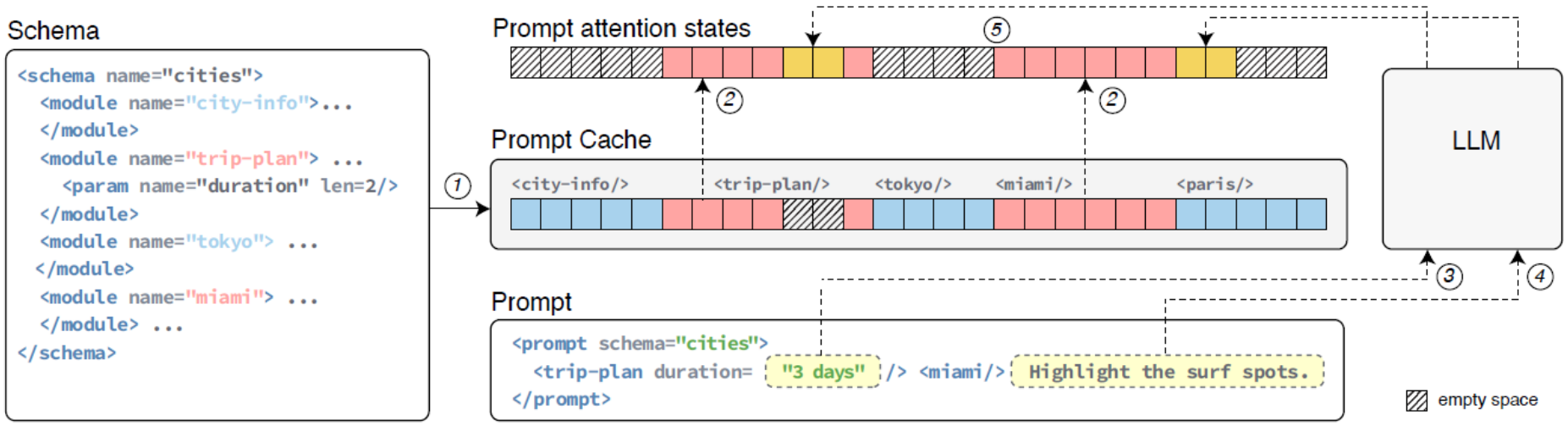


Figure 6. Block table translation in vLLM.

- **PagedAttention Overview**
  - Follows OS's virtual memory principles
  - Splits KV cache into same-sized non-contiguous blocks; on-demand allocation of blocks
  - A block engine allocates a contiguous chunk of GPU memory and divides it physical KV blocks
  - A block table maintains a mapping from logical to physical KV blocks
  - vLLM inference and serving engine

* Efficient Memory Management for Large Language Model Serving with PagedAttention. SOSP 2023.

Prompt Cache

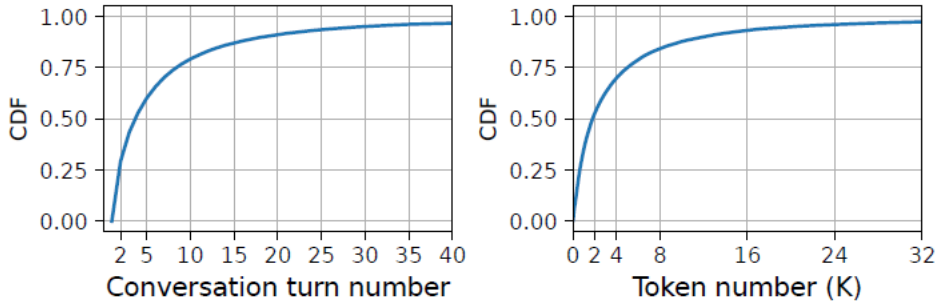


Prompt template for a trip plan

- **Overview**
  - LLM applications insert identical system prompts and use reusable prompt templates
  - This leads to overlapping text segments; large fraction of prompts are reused frequently
  - Solution: precompute and cache KV attention states for the reusable text segments
  - Challenge: reuse possible only if the text segment appears at the same position
  - Prompt Markup Language (PML); users write prompt in PML and derive prompts from a schema

* Prompt Cache: Modular Attention Reuse for Low-Latency Inference. MLSys 2024.

CachedAttention for Multi-turn Conversations



(a) CDF for conversation turn number (b) CDF for session length

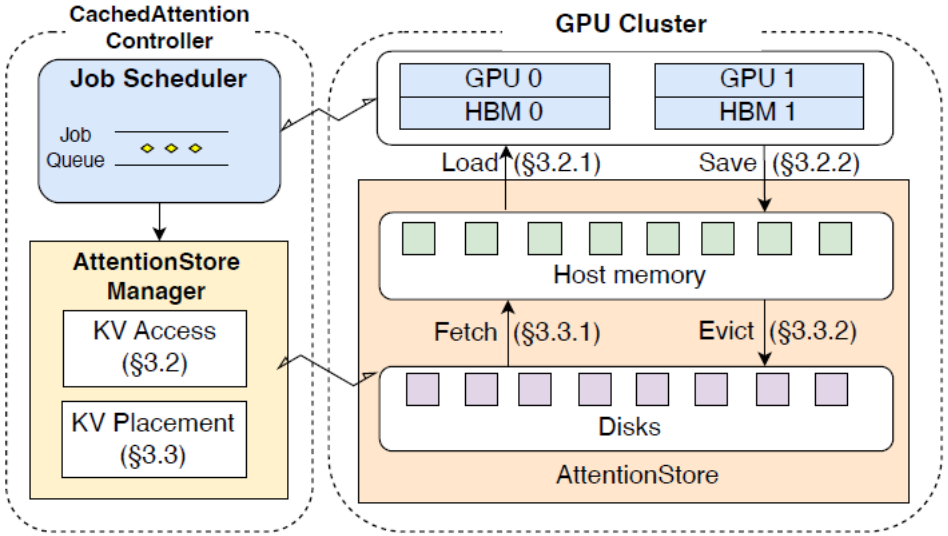


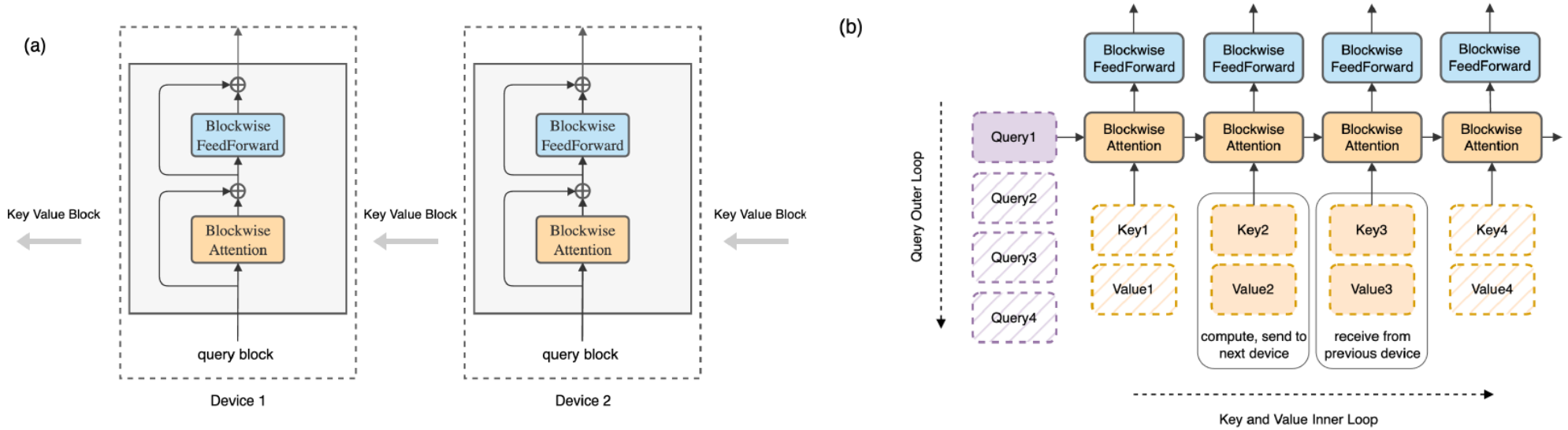
Figure 5: The system architecture of CachedAttention.

- **Overview**
  - ~73% conversations involve multiple turns; serving engines discard KV cache after each session and regenerate when becomes active again
  - CachedAttention stores the KV caches in host memory and disks
  - Layer-wise pre-loading to overlap host-to-device copy and layer computation
  - Challenge: prompts come randomly; corresponding KV cache are likely to be in disk
  - Job scheduler applies look-ahead prefetching window for the waiting jobs and prefetch the KV cache of the waiting jobs from the disk

* Cost-Efficient Large Language Model Serving for Multi-turn Conversations with CachedAttention. USENIX ATC 2024.

Long-Context Serving

RingAttention



- **Overview**
  - For batch size 1, 100M tokens require 100GB memory (for a model with a hidden size 1024)
  - FlashAttention avoids materializing full matrix but still needs to store the outputs of all layers in memory
  - Distribute KV blocks among multiple devices, each managing its input query block
  - key-value blocks traverse through a ring of hosts for attention and FFN computations block-by-block
  - Overlap attention compute with sending KV block to the next host and receiving from the previous host

* Ring Attention with Blockwise Transformers for Near-Infinite Context. ICLR 2024.

InfiniGen for KV Cache Offloading

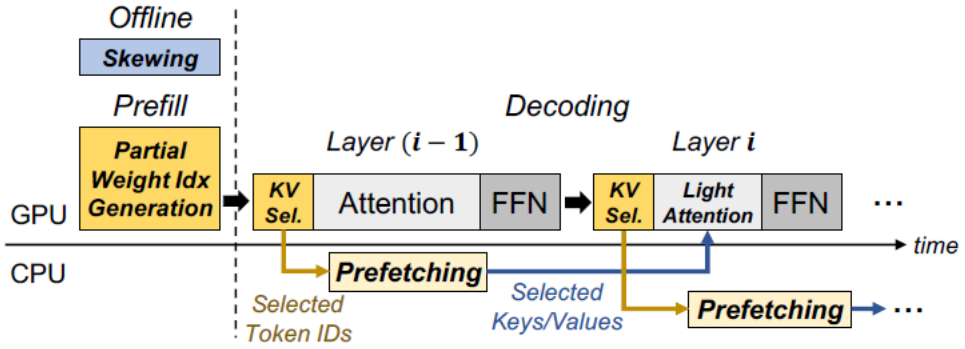


Figure 8: Operation flow of the prefetching module of InfiniGen.

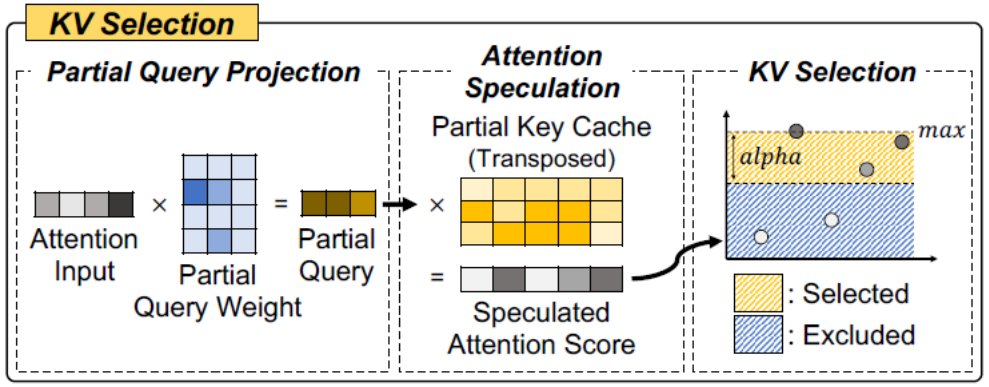


Figure 10: Attention score speculation in the decoding stage.

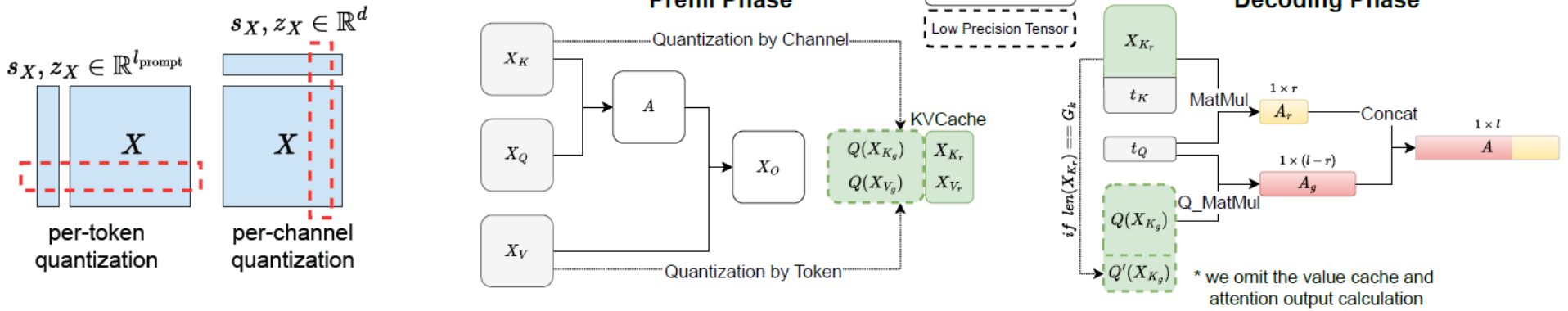
- **Overview**
  - Offloading entire layer takes significantly longer than layer computation
  - Offloads KV cache to host memory and prefetch only the essential entries
  - Speculates KV cache entries that are critical to produce the next token
  - Conduct a *rehearsal* of attention computation for layer i at layer $i-1$
  - Observation: attention inputs of consecutive attention layers are highly similar
  - Requires an offline phase to modify weights to generate skewed Q and K matrices to make speculation precise

$$
\begin{aligned}
\tilde{Q} &= X_a \times W_Q \times A, \qquad \tilde{K} = X_a \times W_K \times A \\
\tilde{Q} \times \tilde{K}^T &= X_a \times W_Q \times A \times (X_a \times W_K \times A)^T \\
&= X_a \times W_Q \times A \times A^T \times W_K^T \times X_a^T \\
&= X_a \times W_Q \times W_K^T \times X_a^T \\
&= X_a \times W_Q \times (X_a \times W_K)^T \\
&= Q \times K^T,
\end{aligned}
$$

Skewed Partial Weight

* InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management. OSDI 2024.

KIVI: Quantization for KV Cache



- **Overview**
  - KV cache quantization is simple and effective; 4bit / 2bit round-to-nearest quantization
  - Streaming nature of KV cache; per-token quantization to both K and V is natural
  - Observation: K tensor has significant pattern in channel (feature), but V does not
  - Proposes per-channel quantization for K and per-token quantization for V
  - Group key cache every *G* tokens for quantization
  - Fuse dequantization and matmul at tiling level

Quantization Recap

$$x_q = \text{clip}\left(\text{round}\left(\frac{x}{s}\right), \alpha_q, \beta_q\right)$$

$$x \approx x' = s \cdot x_q$$

- x is the original value
- x_q is the quantized value
- s is the scale factor, a positive real number

* KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache. ICML 2024.

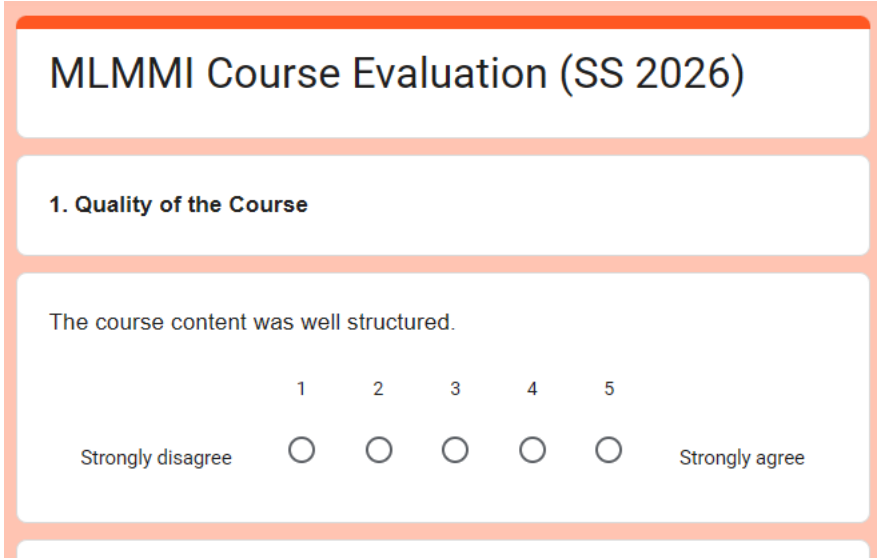
Course Summary

- **Model Management and Deployment**
  - Model versioning, sharing, governance, reproducibility
  - Dataset versioning and provenance tracking; feature stores
  - MLOps, lifecycle management, monitoring, data drift and validation
  - AutoML, model selection optimizations, transfer learning
- **AI Agents**
  - Compound AI systems, prompt optimization
  - AI agents, agent context and memory, tooling, MCP, RAG
  - Text-to-SQL and ML code generation use cases
- **Model and LLM Serving**
  - Traditional model serving optimizations, in-database serving, serverless
  - LLM serving systems, KV cache memory management, long-context serving

Thank You and Feedback

- **Thank You**
  - My first course designed from scratch
  - Active research topic
  - Really appreciated your engagement
  - I hope you enjoyed and learned something new
  - I apologize for the hiccups

- **Course Evaluation**
  - Help us improve the course
  - We will appreciate your participation
  - **Deadline: July 24**



https://forms.gle/YVK2nXJzb2NHoBig6

Course Evaluation Form

Project Presentations : July 15 (2-2.5 hours)
Project Report and Code: July 29