



# LLM MODEL SERVING

**Team 6**

Fengyu Gao, Shunqiang Feng, Wei Shen, Zihan Zhao

Team5 Previous KV Survey



# Towards Efficient Generative Large Language Model Serving: A Survey from Algorithms to Systems

**Zihan Zhao (rxy6cc)**

# LLM Lifecycle

- Training
  - **Build** a model
- Goals
  - Accuracy
  - Efficiency
- Inference ★
  - **Deploy** a model
- Goals
  - Latency & Throughput
  - Efficiency

# LLM Inference

- Prefill

- Compute QKVs of prompts
- Save all KV pairs
- Generate the first token

- Decode

- Compute the next QKV
- Save this KV pair
- Generate the next token



# Challenges

- Service-Level Objectives (SLOs)
  - Latency
  - Throughput
- Resource Allocation
  - Memory
  - CPU/GPU Compute
  - Power Efficiency
- Scalability
  - Auto Scaling
  - Load Balancing
- Accelerator Compatibility
  - NVIDIA Superchip
  - NVIDIA GPU Direct RDMA

# Challenges - LLM-Specific SLOs

- Latency
  - Time To First Token (TTFT)
  - Time Between Token (TBT)
  - Time Per Output Token (TPOT)
- Throughput
  - Tokens per second

# Solutions

- Algorithms

- Decoding Algorithm
- Architecture Design
- Model Compression

- Systems

- Parallel Computation
- Memory Management ★
- Request Scheduling
- Kernel Optimization
- Low-bit Quantization
- Disaggregated Serving



# Algorithmic Solutions

# Decoding Algorithms

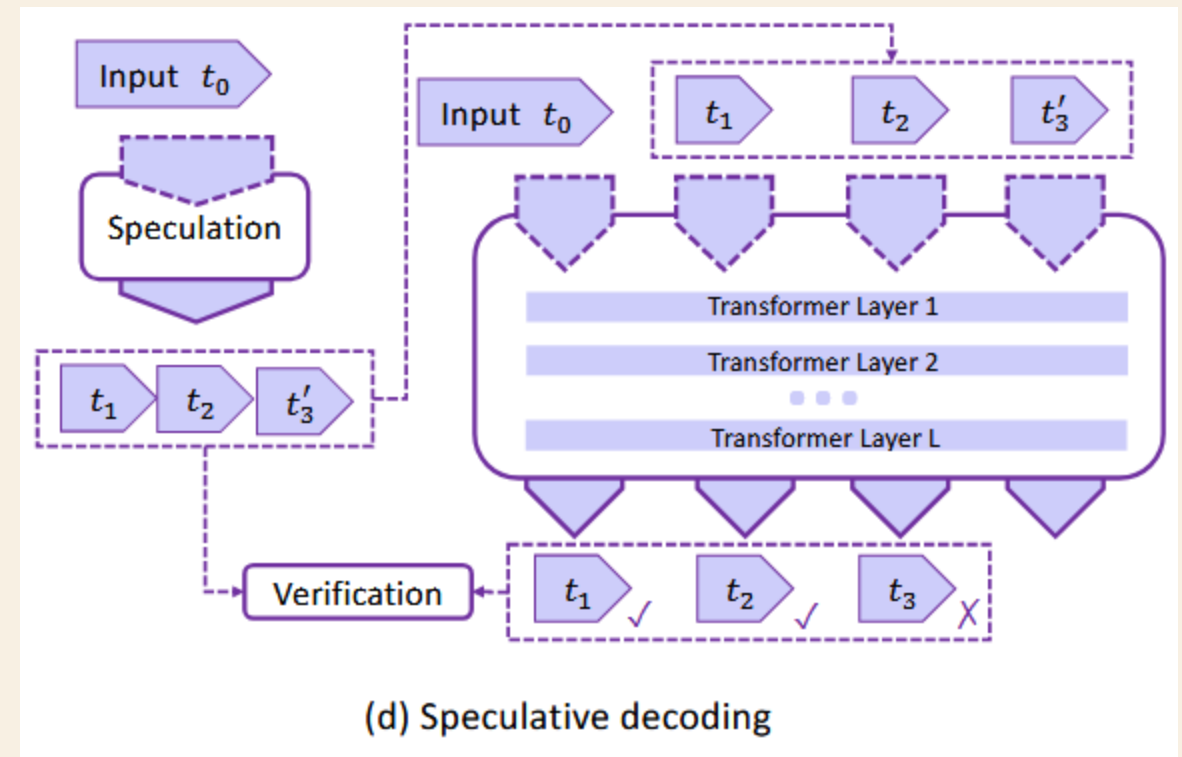
- **Speculative Decoding**

- *Inspired by Speculative Execution*
- Predict the next a few tokens
- Verify they are correct

- Example

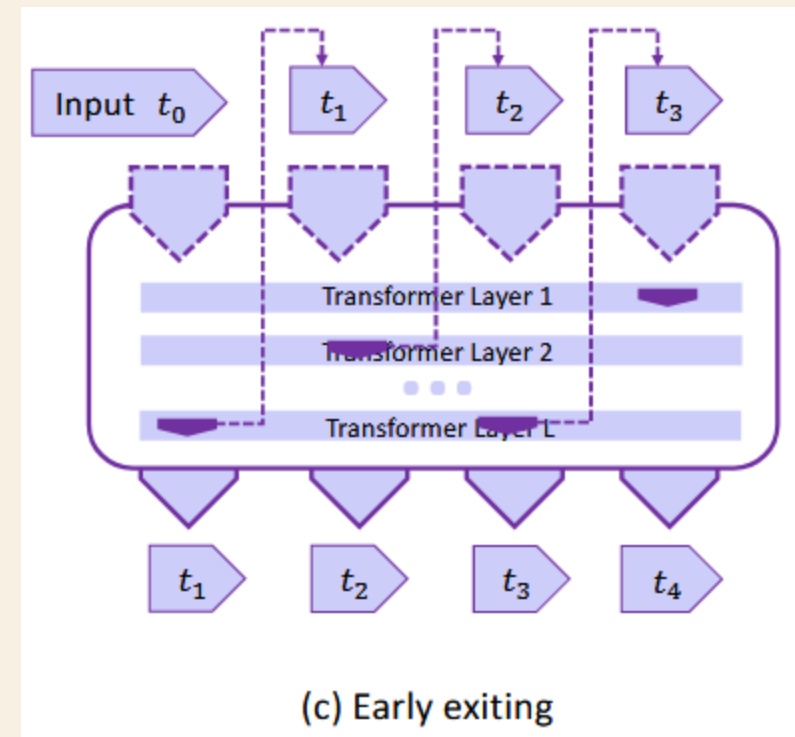
- SpecInfer (ASPLOS '24)

```
if (x > 10)
  y = a + b;
```



# Decoding Algorithms

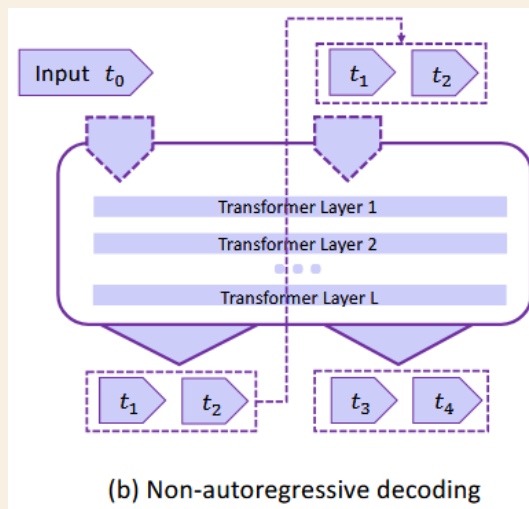
- **Early Exiting** / Adaptive Computation
  - Outputs of early model layers suffice



# Decoding Algorithms

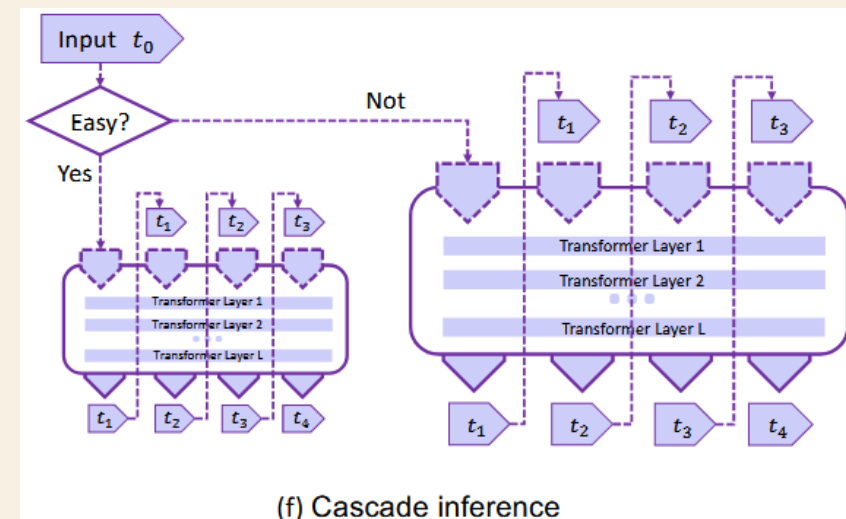
- **Parallel Decoding**

- Generate multi tokens at a time



- **Cascade Inference**

- Use models of different sizes



# Architecture Designs

- **Configuration Downsizing**

- Reduce parameters

- **Attention Simplification**

- Reduce quadratic computational complexity, *e.g. FlashAttention, SparseTransformer*

- **Attention Sharing**

- Reuse common attention matrices, *e.g. Multi-Query A., Grouped-Query A.*

- **Conditional Computing**

- Use only activated parameters, *e.g. DeepSeek-V3, GPT-4\*, Gemini-1.5*

# Model Compression

- **Knowledge Distillation**

- Teach a small model by a large model, *e.g. DeepSeek-R1-Distill-\**

- **Network Pruning**

- Drop a subset of network weights

# Limitations of Algo Solutions

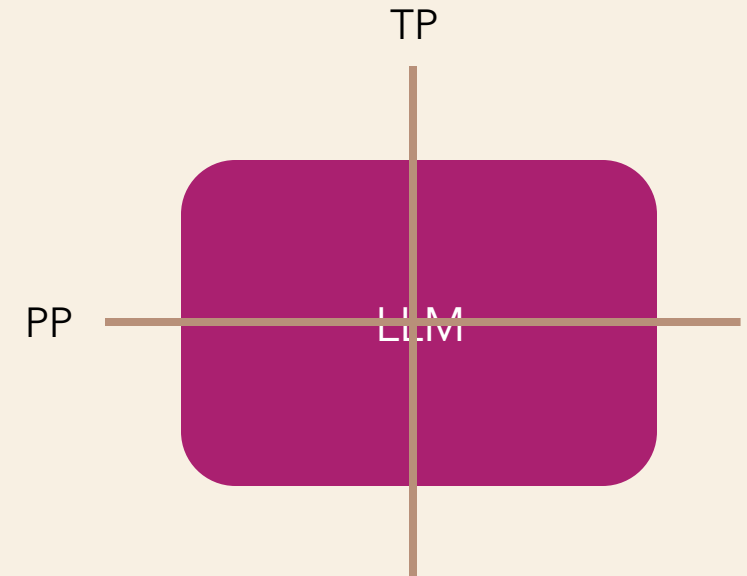
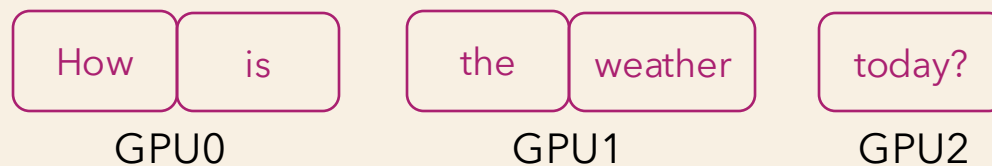
- Usually require retraining
- Usually trade accuracy for efficiency

# **System Solutions**



# Parallel Computation

- **Pipeline Parallelism (PP)**
  - Move different layers onto different GPUs
- **Tensor Parallelism (TP)**
  - Move *slices of layers* onto different GPUs
- **Sequence Parallelism (SP)**
  - Move parts of a long sequence onto different GPUs



# Memory Management

- Model Weights and Activations
- KV Cache
  - **PagedAttention** (vLLM, *40k stars*)
    - Partitions KV cache into non-contiguous memory blocks
  - **TokenAttention** (LightLLM, *3k stars*)
    - Optimizes page-level attention by using token-level attention
  - **RadixAttention** (SGLang, *11.3k stars*)
    - Use prefix tree to enable efficient KV cache reuse

# Request Scheduling

- **Request-level Scheduling**
  - Treat a request as the smallest unit
- **Iteration-level Scheduling**
  - Treat one iteration of a request as the smallest unit
  - Examples
    - Continuous Batching (OSDI '22)
    - Chunked Prefill (OSDI '24)

# Kernel Optimization

- **Kernel Fusion**
  - Fuse GEMMs and addition whenever possible
- **Tailored Attention**
  - Optimize kernel implementations for attention calculations
- **Variable Sequence Length**
  - Batch-process sequences of various lengths

# Low-bit Quantization

- Use fewer bits to represent model weights and activations
- Examples
  - **Quantization-Aware Training (QAT)**
  - **Post-Training Quantization (PTQ)**
    - Reduce precision from FP32/FP16 to *INT8* or *INT4* or *FP8*

# Disaggregated Serving

- Split a LLM based on execution stages across GPUs/Nodes
  - Disaggregate prefill and decode stages
- Examples
  - Splitwise (ISCA '24)
  - DistServe (OSDI '24)

# Limitations of System Solutions

- Unable to address model inefficiencies
- Usually require a new system to deploy

# Frameworks

Table 2. Comparison of state-of-the-art open-sourced GPU-based LLM serving systems.

Name Github Ref.	Parallel Computation			Itera- tion- Sche.	Attention Kernel		Prioritized Opt. Target		Main Features
	TP	PP	Offload		Initial	Incremental	$L_{at}$	$T_{pt}$	
FasterTrans- former [2]	✓	✓			cuBLAS GEMM	Fused attention	✓		<ul style="list-style-type: none"> <li>Manually-written kernel</li> <li>Lightweight runtime</li> </ul>
FlexFlow- Serve [12]	✓	✓	✓	✓	cuBLAS GEMM	Tree attention	✓		<ul style="list-style-type: none"> <li>SpecInfer [177]</li> <li>Extremely low <math>L_{at}</math></li> </ul>
vLLM [29]	✓		✓	✓	xFormers	Paged attention		✓	<ul style="list-style-type: none"> <li>Block-level KV cache [150]</li> <li>Enlarging batch size &amp; <math>T_{pt}</math></li> </ul>
FlexGen [13]		✓	✓		torch. bmm	torch. bmm		✓	<ul style="list-style-type: none"> <li>CPU&amp;Disk Offload [224]</li> <li>Maximizing single GPU <math>T_{pt}</math></li> </ul>
TGI [18]	✓			✓	Flash attention	Paged attention		✓	<ul style="list-style-type: none"> <li>Huggingface integration</li> </ul>
DeepSpeed- Inference [3]	✓	✓			cuBLAS GEMM	cuBLAS GEMM	✓		<ul style="list-style-type: none"> <li>Kernel auto-injection [10]</li> <li>Multi-GPU &amp; Multi-Node</li> </ul>
ZeRO- Inference [3]	✓	✓	✓		cuBLAS GEMM	cuBLAS GEMM		✓	<ul style="list-style-type: none"> <li>CPU&amp;NVMe Offload [35]</li> <li>Maximizing single GPU <math>T_{pt}</math></li> </ul>
Light- LLM [21]	✓			✓	Flash attention	Token attention		✓	<ul style="list-style-type: none"> <li>Token-level KV cache</li> <li>Enlarging batch size &amp; <math>T_{pt}</math></li> </ul>
MLC- LLM [242]	✓			✓	compiled MatMul	Paged attention	✓		<ul style="list-style-type: none"> <li>Universal deployment</li> <li>Multiple types of GPUs</li> </ul>
TensorRT- LLM [25]	✓	✓	✓	✓	cuBLAS/ Flash-attn	Paged attention	✓		<ul style="list-style-type: none"> <li>NVIDIA Triton integration</li> <li>Rich features supported</li> </ul>



# Future Work

- Development and Enhancement of Hardware Accelerators
- Efficient and Effective Decoding Algorithms
- Long Context/Sequence Scenarios Optimization
- Investigating Alternative Architectures
- Exploration of Deployment in Complex Environments
- Automatic Adaptation to Specific Requirements



# Efficient Memory Management for Large Language Model Serving with PagedAttention (vLLM)

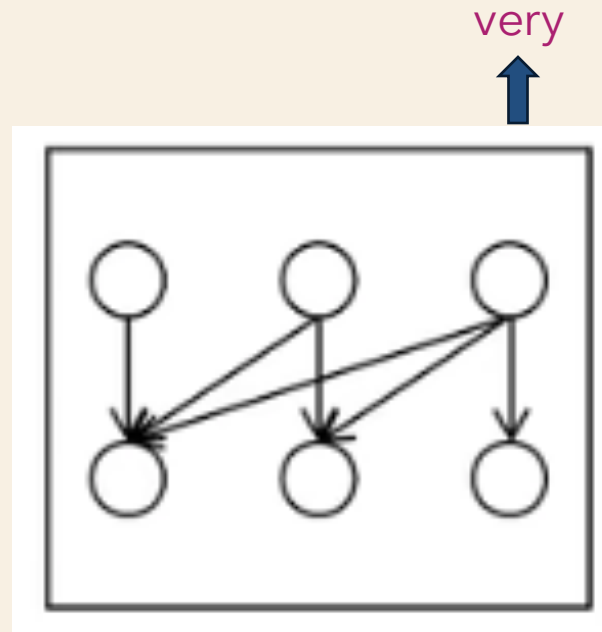
<https://github.com/vllm-project/vllm>

★40K +

**Shunqiang Feng (mpp7ez)**

# Start with KV cache

## How does LLM generate?



very  
↑

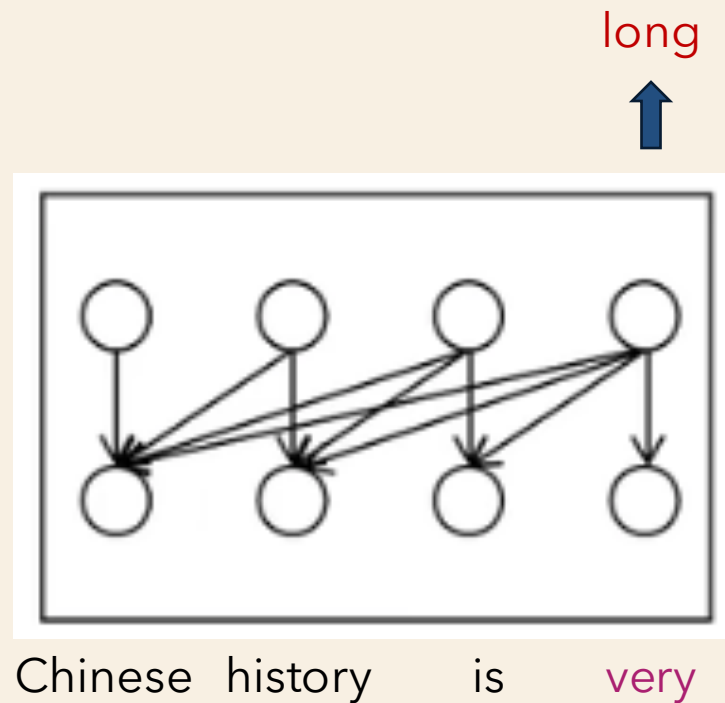
Chinese history is

Suppose we input "Chinese history is"  
It's split into three tokens.

Then, it generates the next token "very".

# Start with KV cache

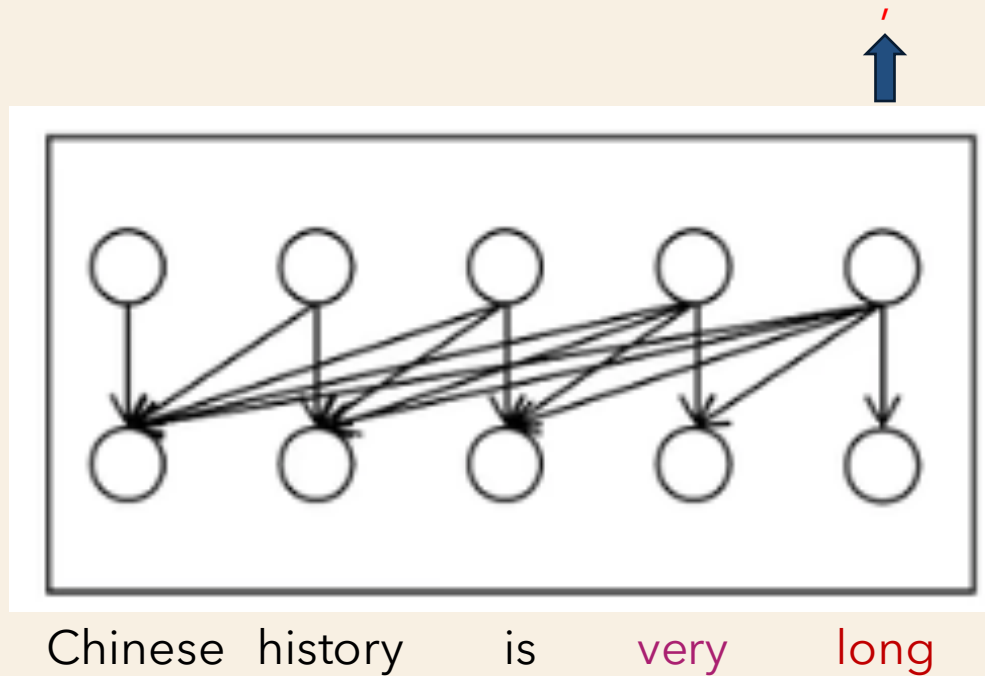
## How does LLM generate?



The word "very" is appended to the input, which predicts the next token "long"

# Start with KV cache

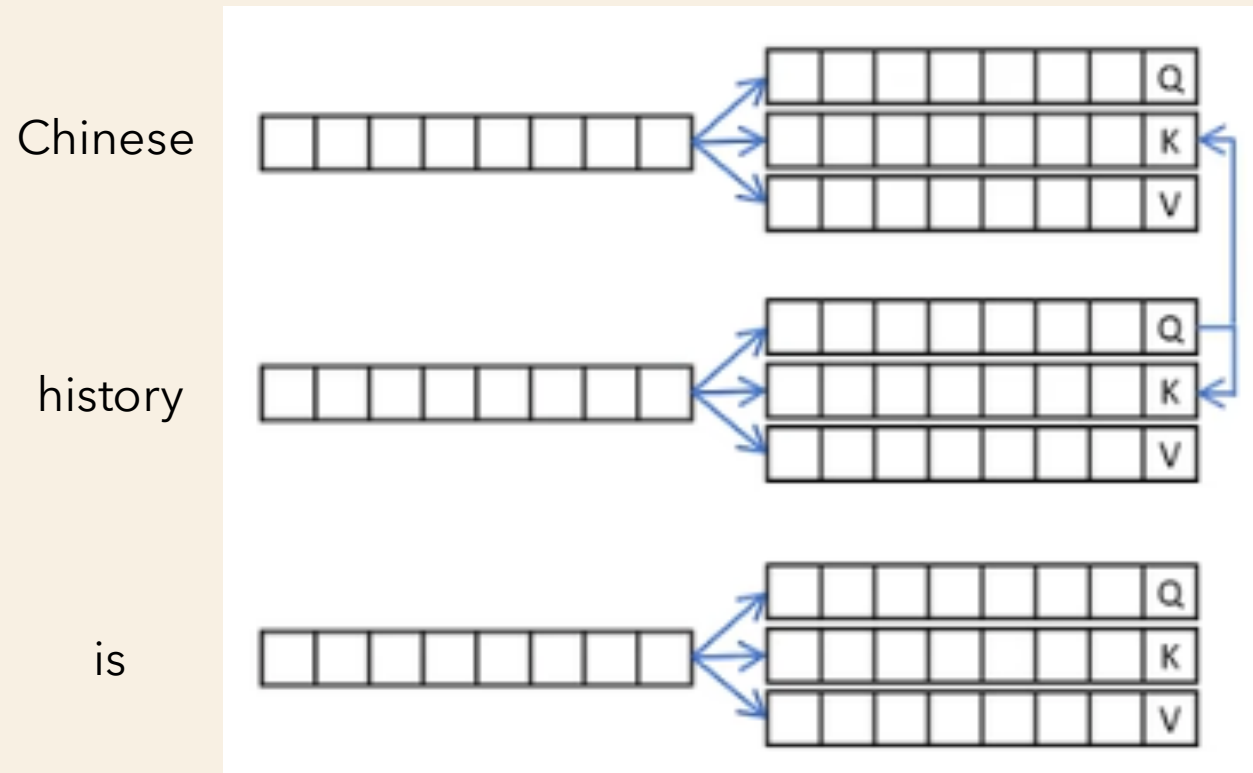
## How does LLM generate?



Then, "long" is appended to the input;  
With "Chinese history is very long", the model predicts a comma

# Start with KV cache

## How does self-attention work?

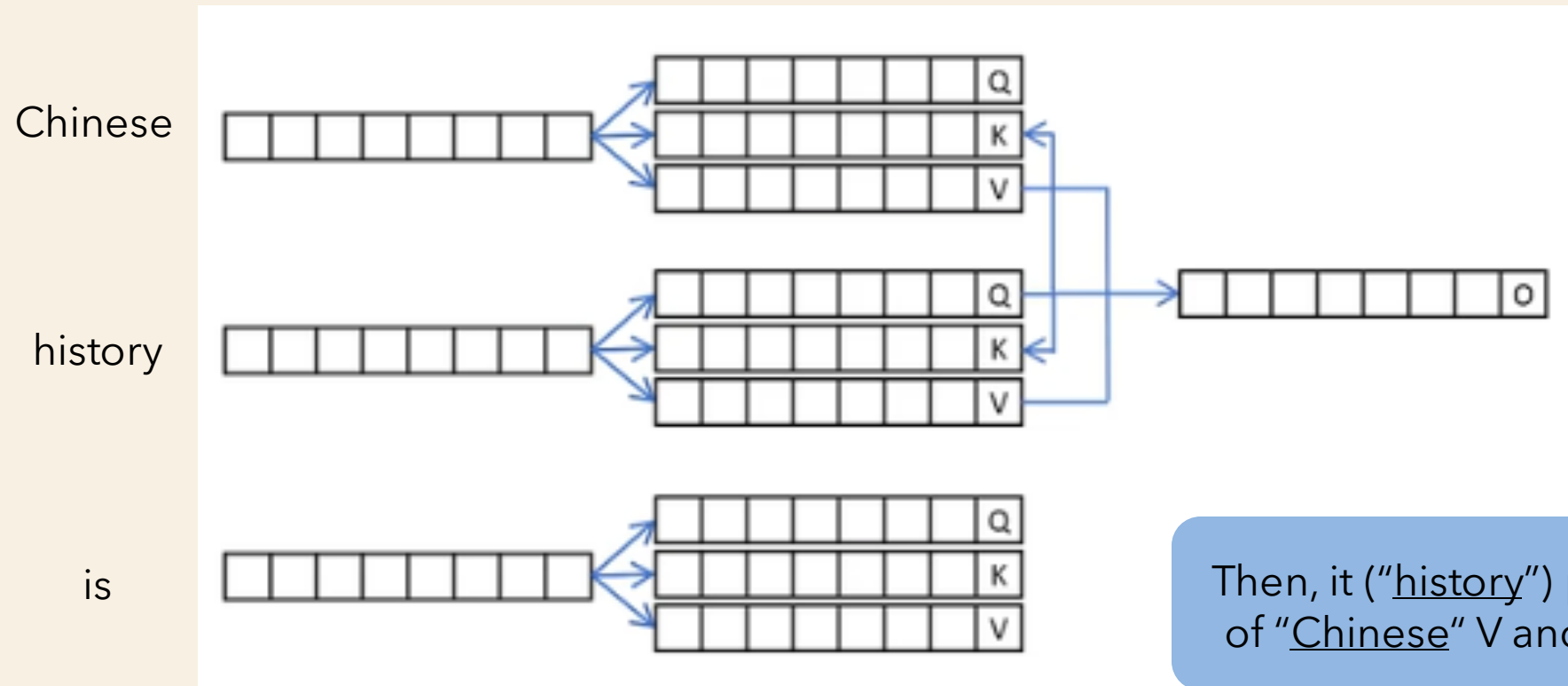


For the first token "Chinese", it can only attend to itself

For the second token "history", it uses its Q to query the K of "Chinese" & "history" to calculate the attention weights;

# Start with KV cache

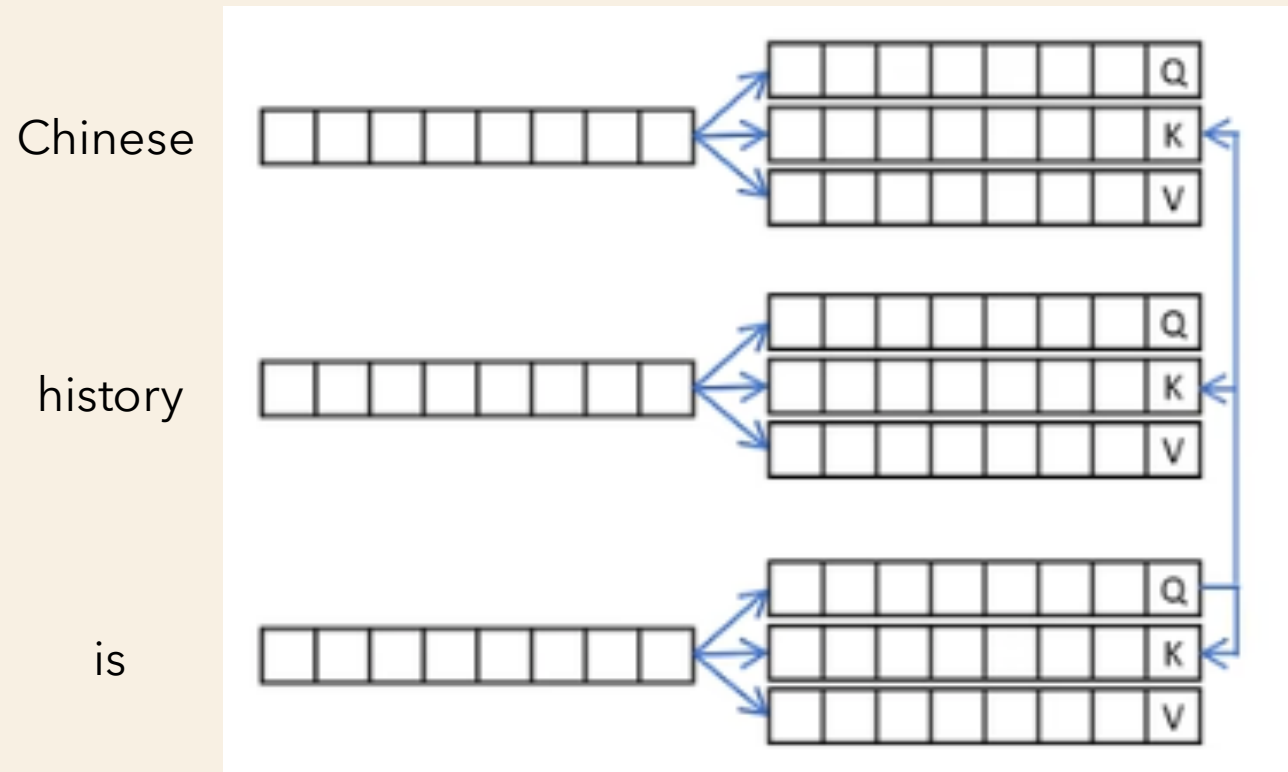
## How does Transformer work?





# Start with KV cache

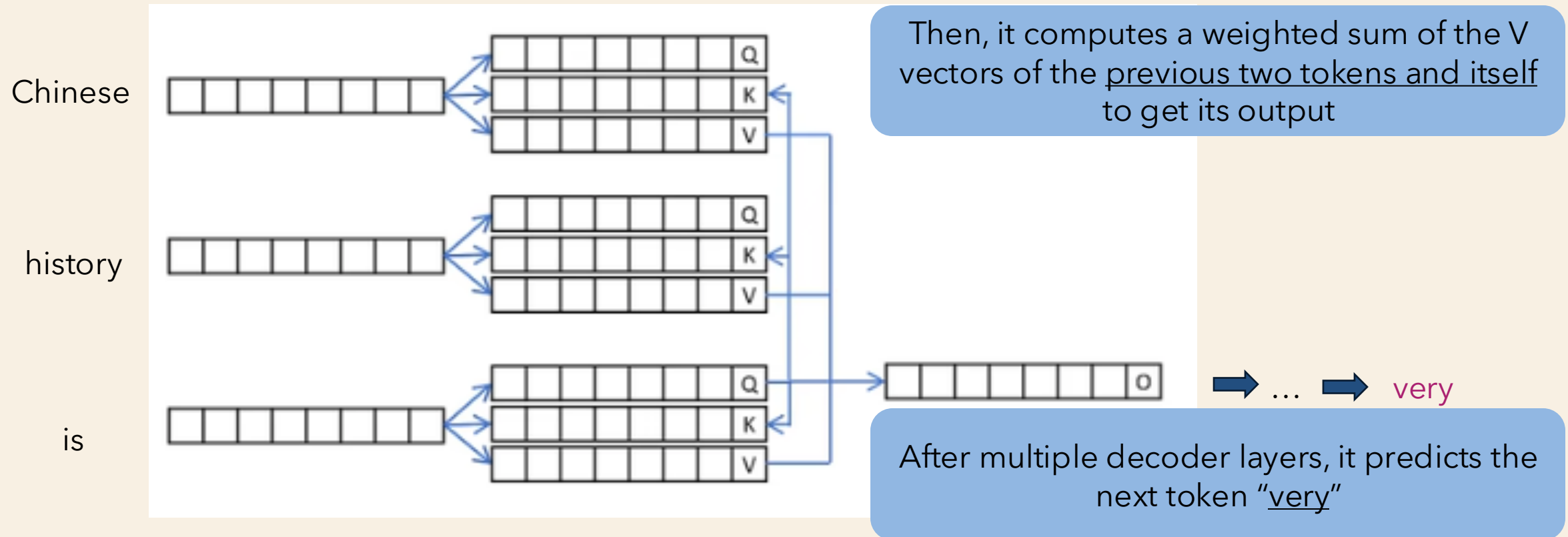
## How does Transformer work?



Similarly, for the token "is", its Q queries the K vectors of the previous two tokens and itself to generate attention weights

# Start with KV cache

## How does Transformer work?



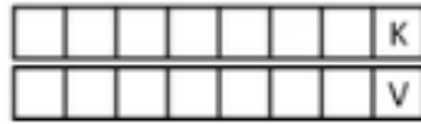
# Start with KV cache

## How does Transformer work?

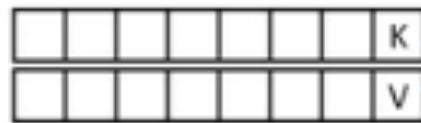
Chinese



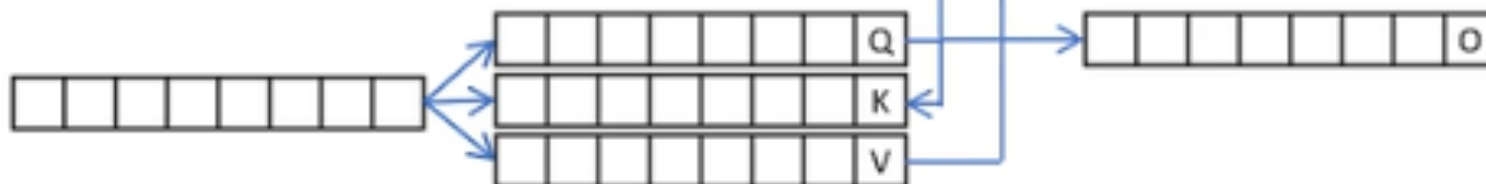
history



is



very

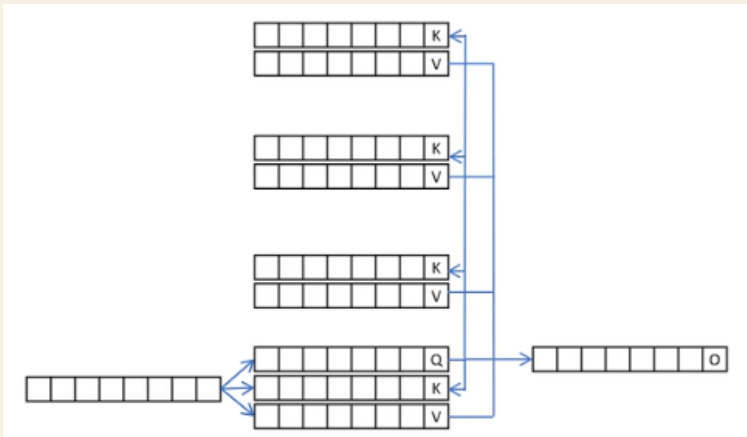


Then, "very" is appended to the input, and then predict next token "long" similarly.

... → long

# Start with KV cache

## KV cache & Its benefit



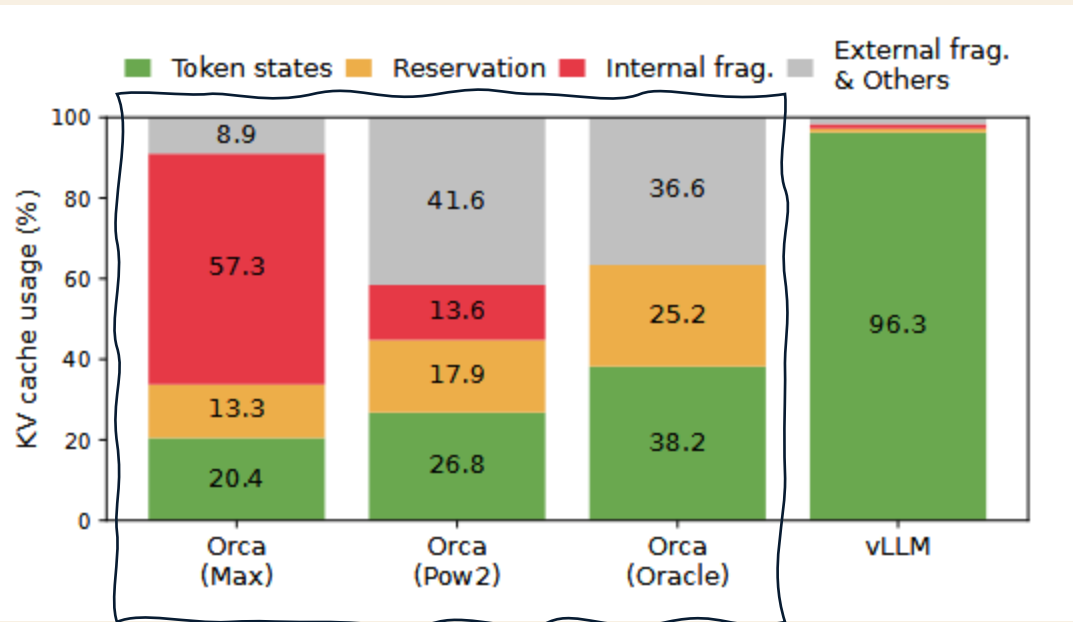
When a new token appended to the input:

- The KV vectors of the previous tokens have already been computed;
- There's no need to recompute them again. (Because each token only attends to the tokens before it, their values aren't affected by the new token)
- In Hugging Face's generate method, it saves the K and V vectors of previously generated tokens by default to speeds up computation, which is called **KV cache**.

That's why when you have a conversation with a LLM model, you don't notice a significant slowdown as the model's output grows longer.

# Problem

## Low utilization rate of KV cache



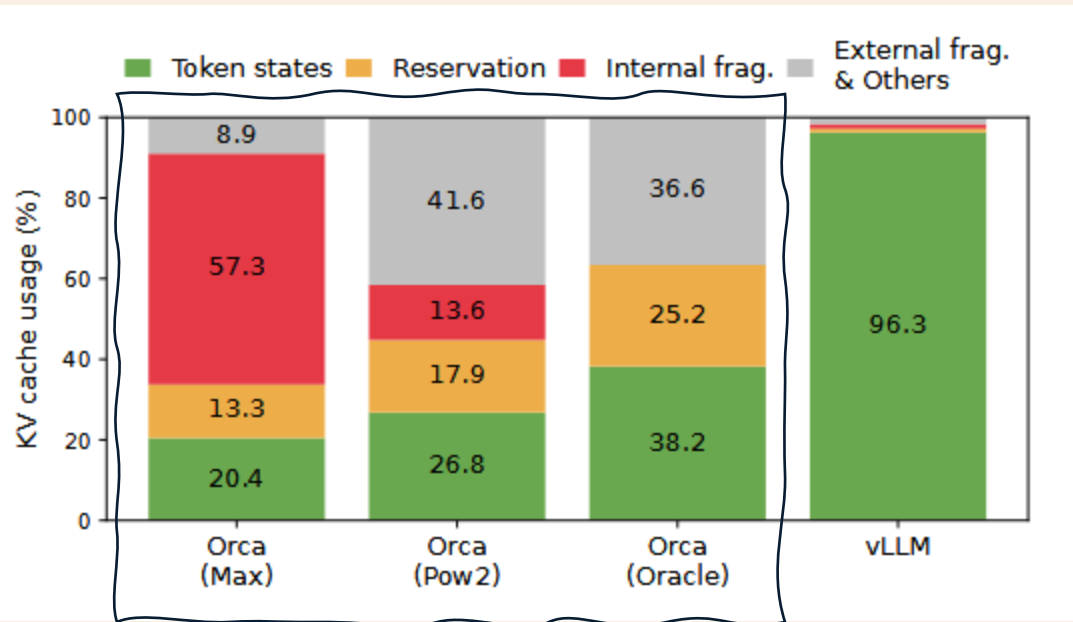
The actual utilization rate of the KV cache is only 20% ~ 40%, most of the GPU memory in the KV cache is wasted for **3 reasons**.

### (1) Pre-allocation for max tokens

For example, if the max token count is 1000, but the model stops at the 100th token, when it outputs an end-of-sequence symbol, then the KV cache for the remaining 900 tokens is wasted.

# Problem

## Low utilization rate of KV cache



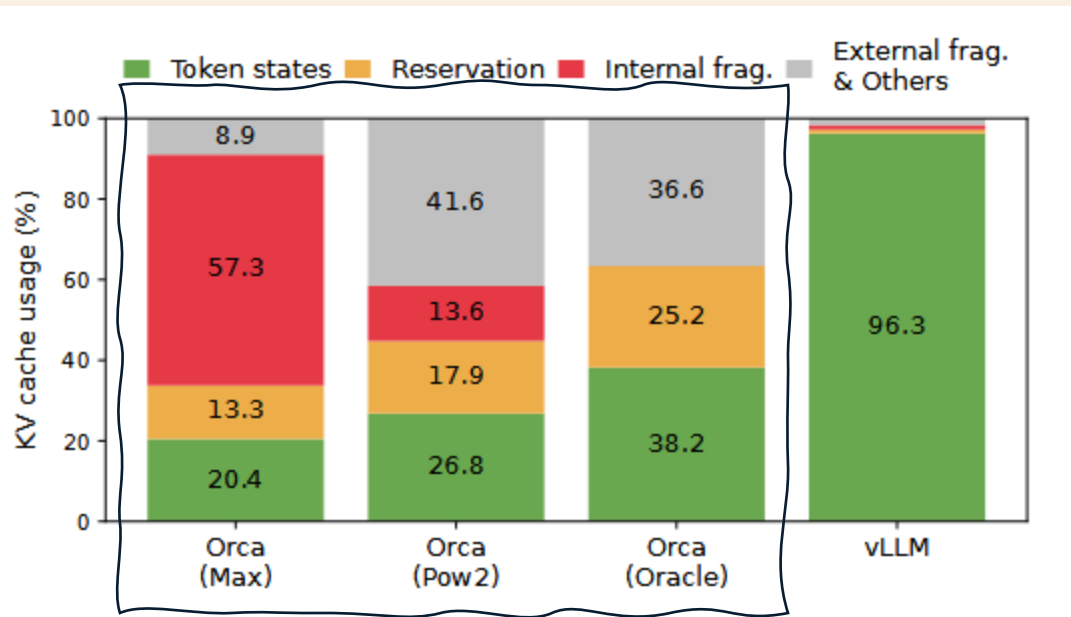
The actual utilization rate of the KV cache is only 20% ~ 40%, most of the GPU memory in the KV cache is wasted for **3 reasons**.

### (2) Unused reserved space blocks other requests

Suppose a sample does output 1000 tokens, when it's just starting to output the 1st token, the remaining tokens haven't been used yet, which could have been processed in parallel with the ongoing sample.

# Problem

## Low utilization rate of KV cache



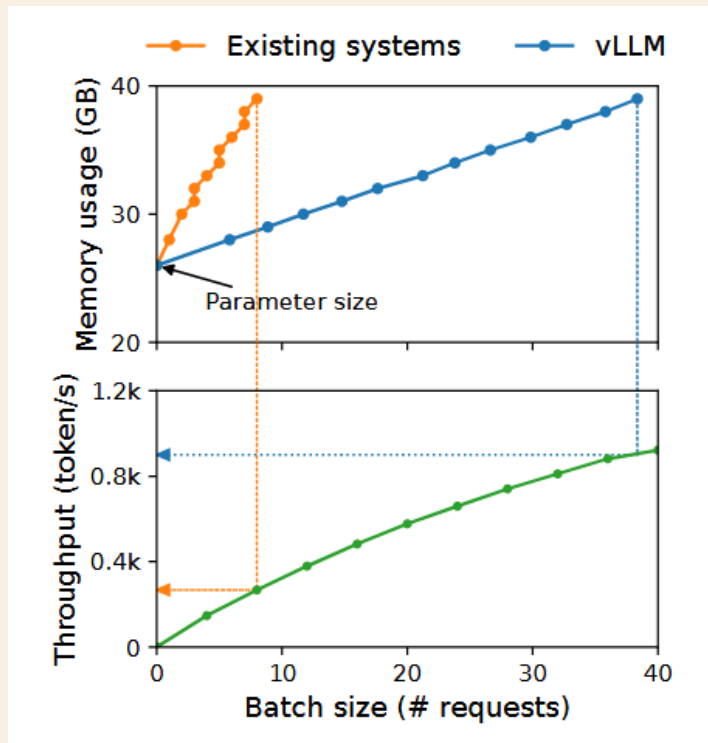
The actual utilization rate of the KV cache is only 20% ~ 40%, most of the GPU memory in the KV cache is wasted for **3 reasons**.

### (3) External fragmentation from varying prompt lengths

When a request finishes generation and releases its cache, but the next request's prompt length is greater than the prompt length of the released request, it can't fit into the freed cache space, which is external fragmentation.

# Method: vLLM

## vLLM & its benefits

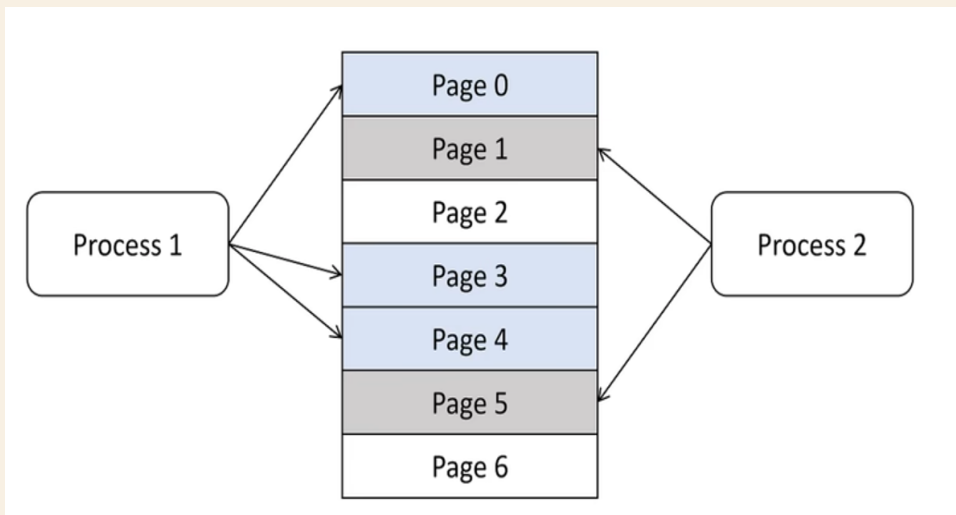


- **Definition:** vLLM is a **high-throughput** distributed LLM serving engine on top of **PagedAttention** that achieves **near-zero waste** in KV cache memory.
- It allows larger batch size to process requests thereby improving system throughput.
- **Benefits:**
  - Batch Size: 8 → 40
  - Throughput: 300 tokens/s → 900 tokens/s



# Main module: PagedAttention

## Inspired by OS Virtual Memory & Paging

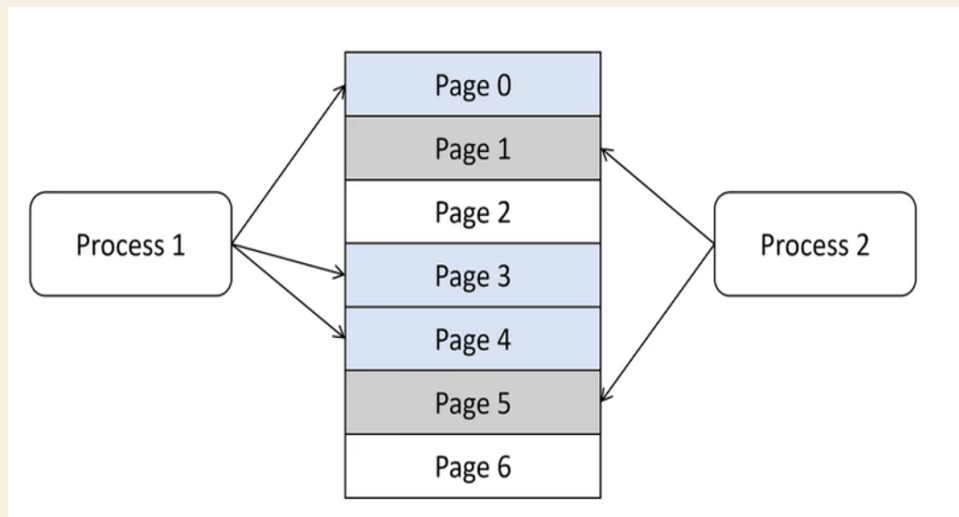


Problems similar to the KV cache have been encountered in operating systems:

- Whether pre-allocate memory for each program or not?
- How to reclaim memory after a program closes?
- How to handle memory fragmentation?
- How to maximize memory utilization?

# Main module: PagedAttention

## Inspired by OS Virtual Memory & Paging

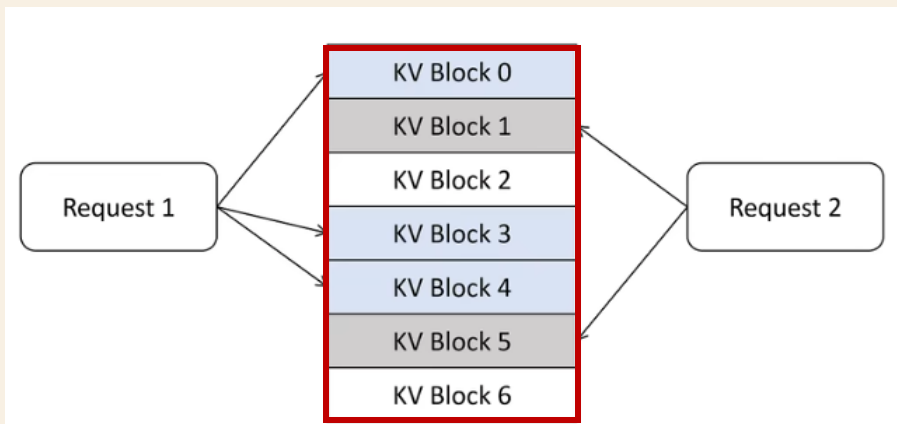


Operating systems solve this using **virtual memory** and **paging** techniques.

- Operating systems allocate memory in minimal units called **pages**.
- Each page is 4KB, and physical memory is divided into many pages.
- The memory each process needs is mapped to different pages.

# Main module: PagedAttention

## Inspired by OS Virtual Memory & Paging



Similarly, PagedAttention divides GPU memory **into KV blocks**.

- The KV cache is managed using KV blocks in GPU memory.
- The KV cache required by each request is split across different KV blocks in the GPU memory.

# Main module: PagedAttention

## KV block

Block1	Chinese		history		is		very	
	K	V	K	V	K	V	K	V
Block2								
	K	V	K	V	K	V	K	V
Block3	long							
	K	V	K	V	K	V	K	V
Block4	I		have		a		dog	
	K	V	K	V	K	V	K	V
Block5	and		two		cats			
	K	V	K	V	K	V	K	V

For example, each KV block can cache the KV vectors of four tokens.

For "Chinese history is very long", these five tokens would correspond to two blocks, which can be non-contiguous in physical GPU memory.

# Main module: PagedAttention

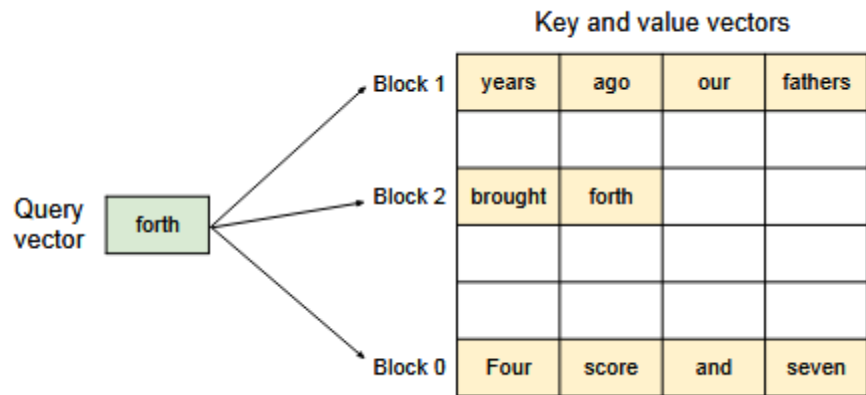
## KV block

Block1	Chinese		history		is		very	
	K	V	K	V	K	V	K	V
Block2								
	K	V	K	V	K	V	K	V
Block3	long		,					
	K	V	K	V	K	V	K	V
Block4	I		have		a		dog	
	K	V	K	V	K	V	K	V
Block5	and		two		cats			
	K	V	K	V	K	V	K	V

When inferencing, it generates a new token, like a comma here. It continues to add it to the unfilled block, until the current block is full.

# Main module: PagedAttention

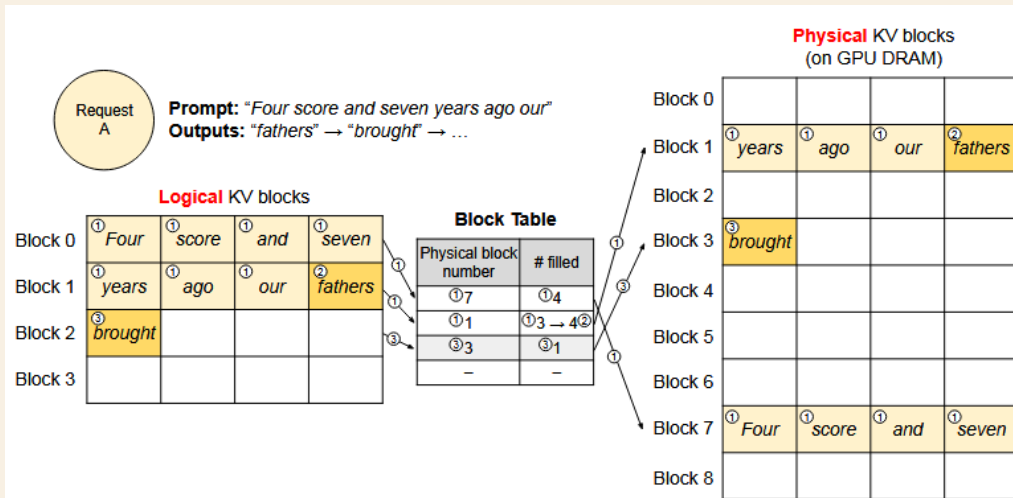
## Benefits



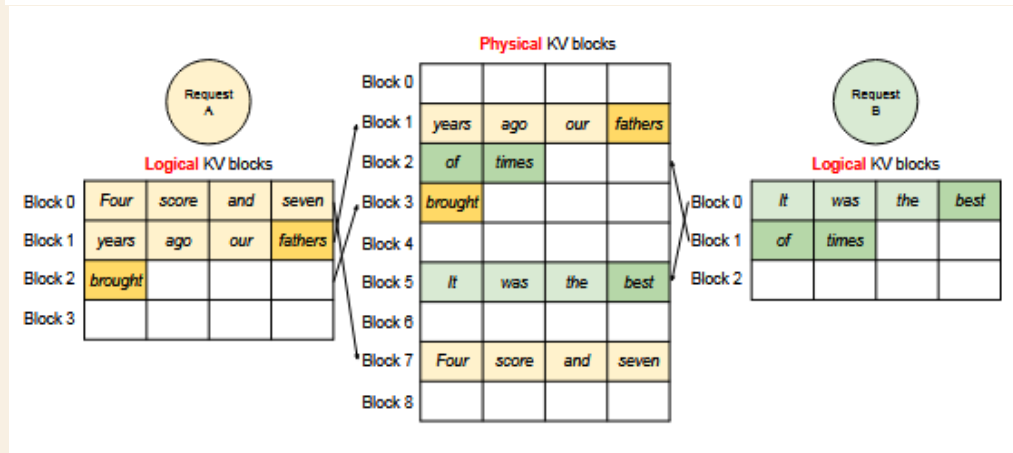
With PagedAttention, vLLM

- Overcomes the pre-allocation problem
  - *It doesn't occupy GPU memory in advance*
- Reduces memory fragmentation
  - *It's all allocated in units of four tokens per block;*
  - *Maximum fragmentation is 3 tokens.*

# Main module: PagedAttention Virtual Memory (Similar to OS)



- **Logical KV cache:** Appears contiguous to each request.
- **Block table:** Translates logical KV cache to physical KV blocks.

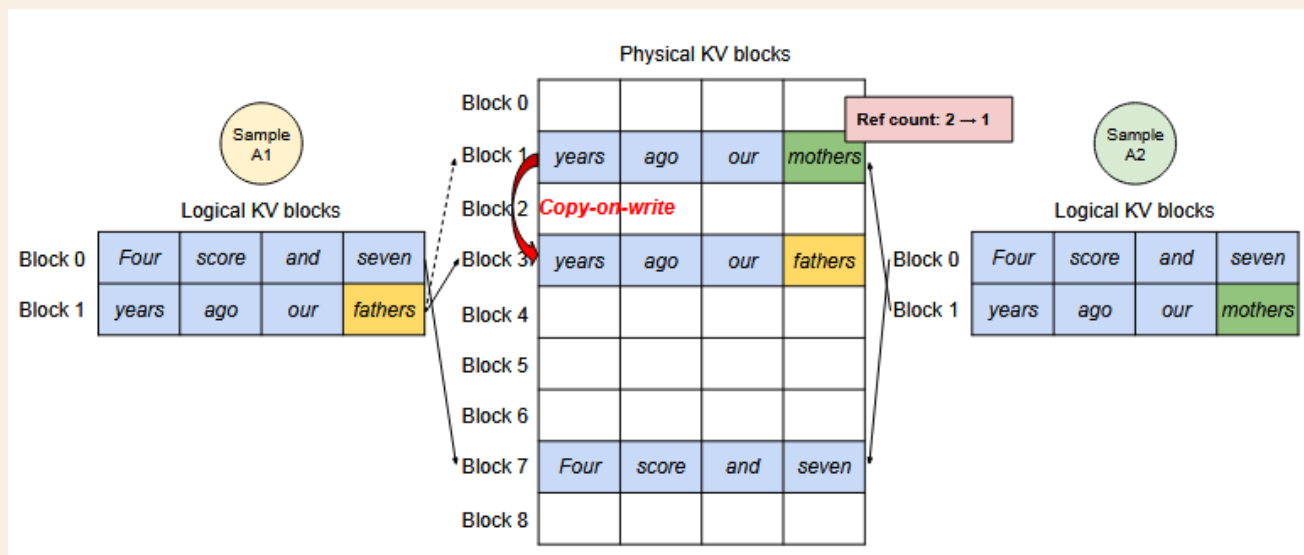


- Store the KV cache of two requests at the same time in vLLM

# vLLM

## Application to Parallel sampling

- **Parallel sampling** generates multiple output sequences from a single prompt, enabling diverse outputs (e.g., translations, completions).



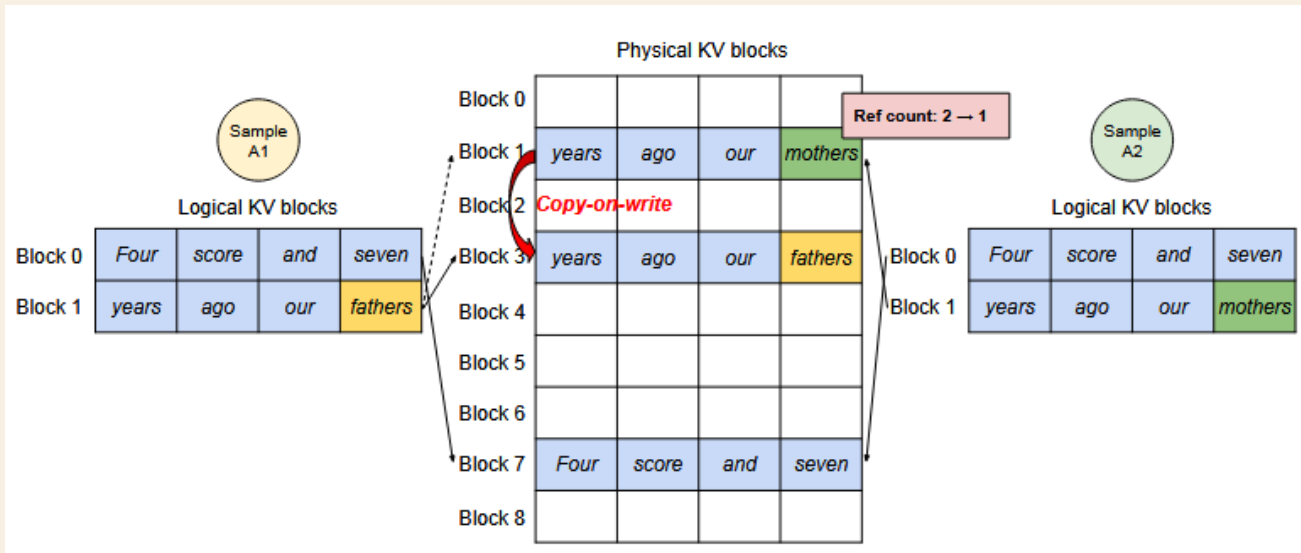
- The same prompt produces two different sequences
- In the GPU memory, only one copy of the prompt tokens' KV blocks is stored
- Each block is marked as being referenced by two sequences
- Only when the reference count drops to zero is the memory occupied by that block released



# vLLM

## Application to Parallel sampling

- **Divergence begins:** Samples A1 and A2 generate different tokens ("fathers" vs. "mothers")



### Copy-on-write mechanism

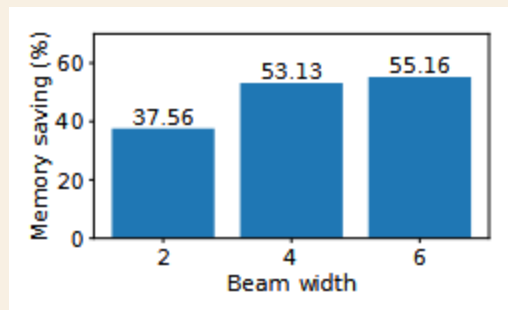
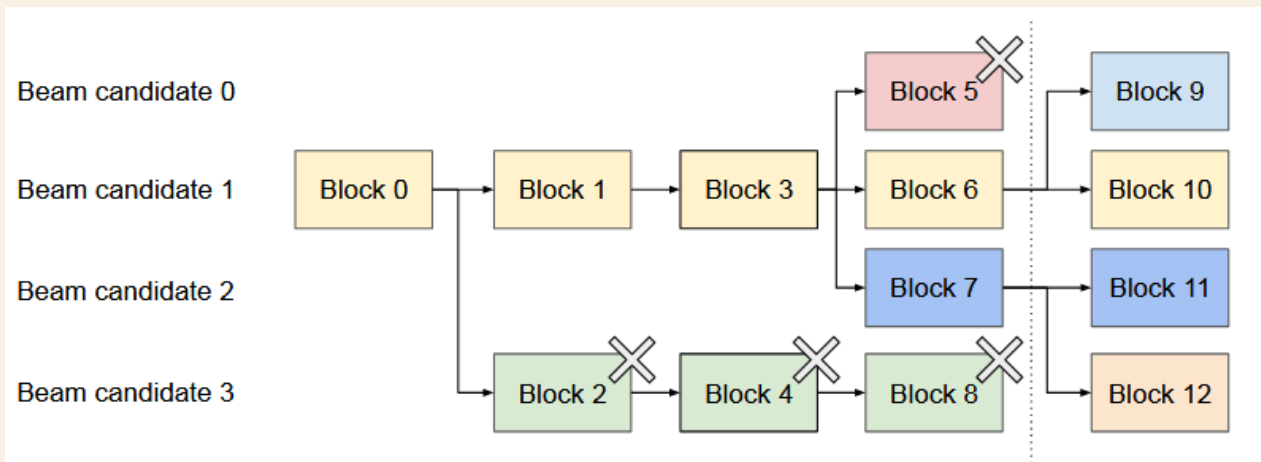
- Sample A1 tries to write to logical block 1 (physical block 1, count > 1) → Allocates new block 3, copies data from block 1, reduces block 1's count to 1.
- Sample A2 writes to physical block 1 (count now 1) → Direct write of new KV cache ("mothers").

- vLLM optimizes parallel sampling by sharing KV blocks for identical prompts, reducing memory waste.

# vLLM

## Application to Beam Search

- **Definition:** Beam search decodes top-k most probable sequences in LLM tasks like machine translation, reducing computational complexity of traversing the full sample space.

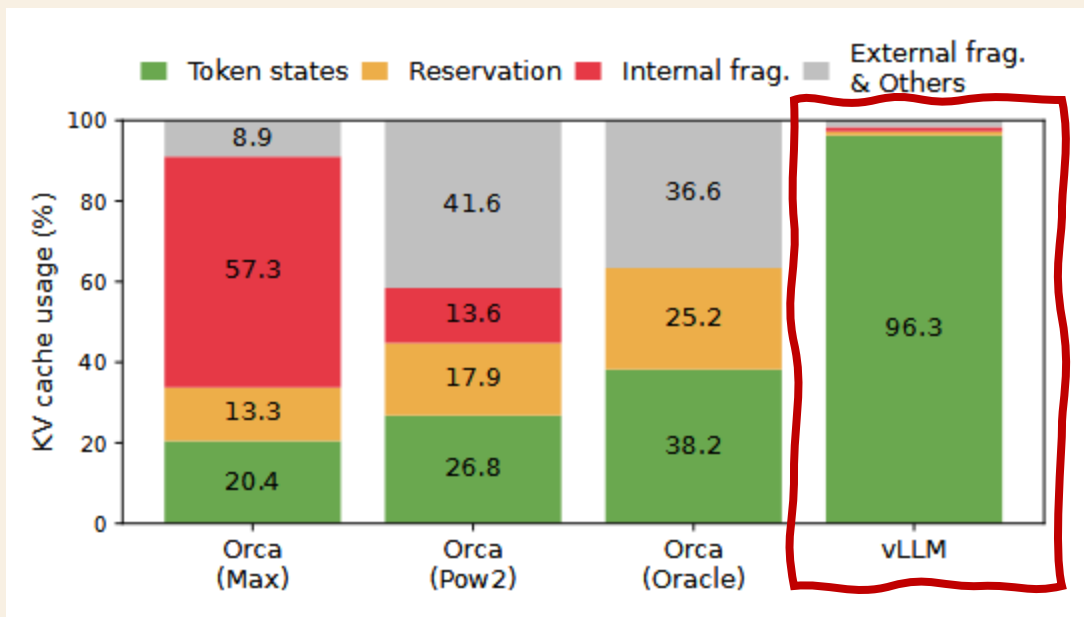


**vLLM Optimization:** Extends KV block sharing beyond prompts, dynamically across beam candidates.

- Shares initial prompt block (block 0) and other blocks (e.g., blocks 1, 3, 6, 7) as decoding progresses
- Reduces frequent memory copies in traditional systems (e.g., candidate 3 copying candidate 2's KV cache).

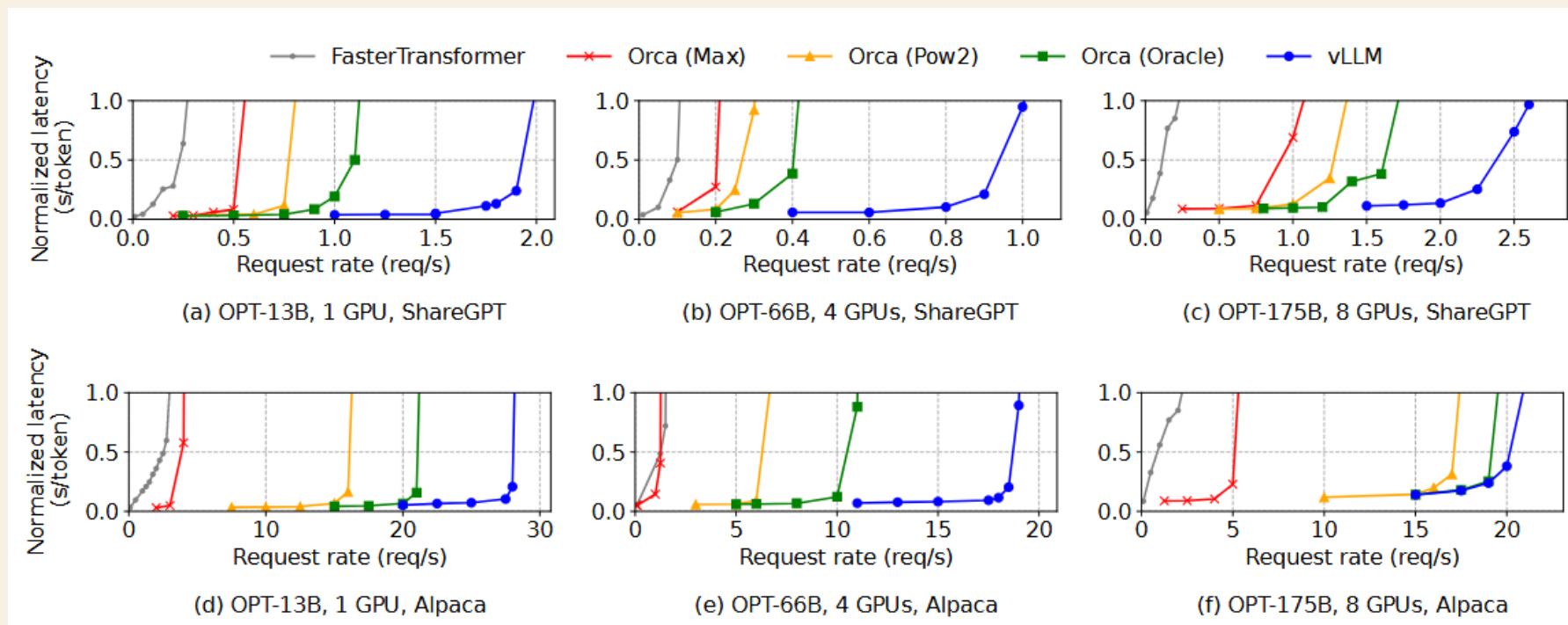
# Key Results

- **Near-zero memory waste** (20.4%-38.2% → ~100% usage)




# Key Results

- **higher request rates:** Compared to FasterTransformer, vLLM can sustain up to **22x** higher request rates



- More: Lower costs, higher scalability for LLM applications...



# A SURVEY ON LARGE LANGUAGE MODEL ACCELERATION BASED ON KV CACHE MANAGEMENT

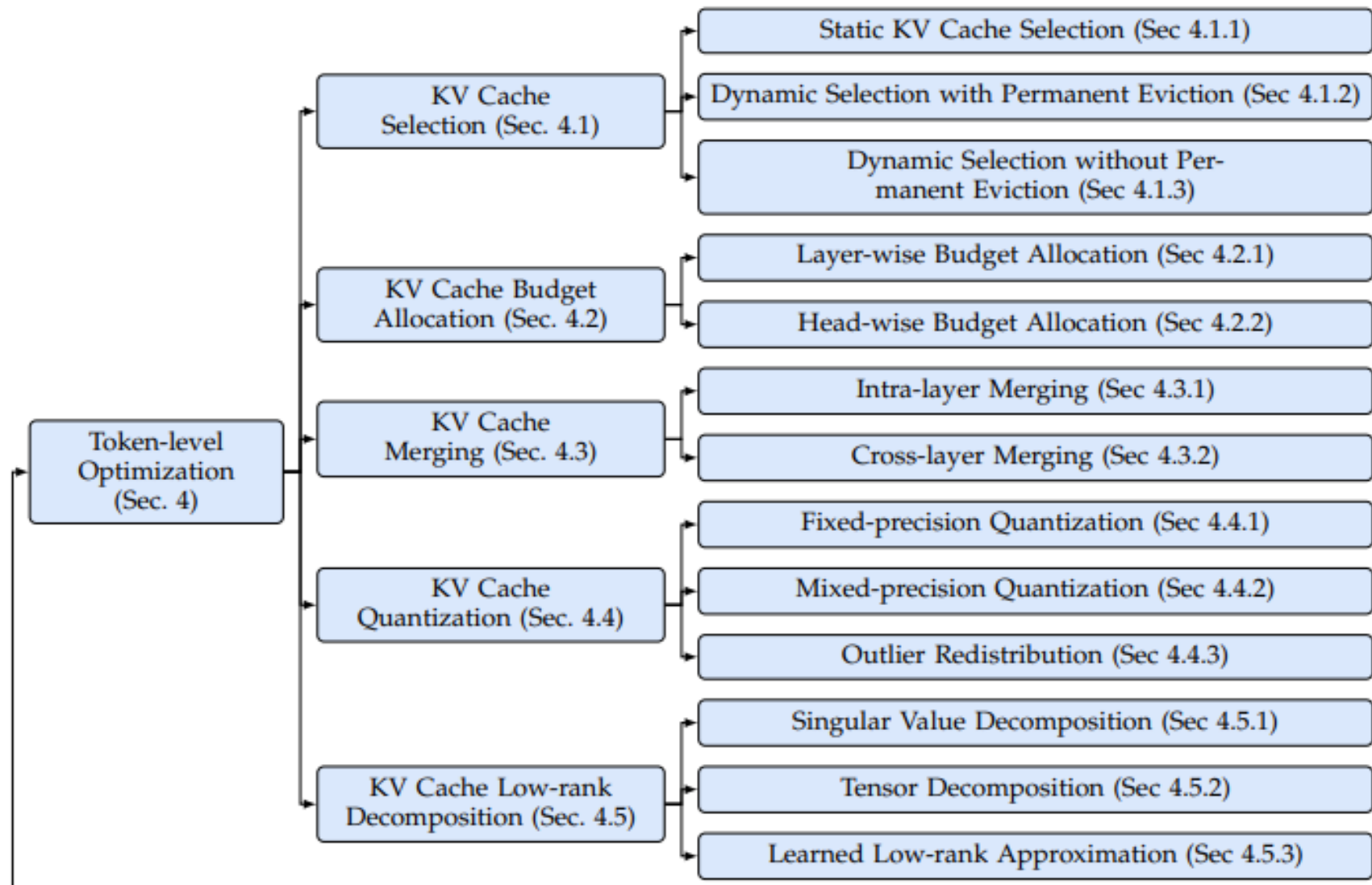
# A Survey on Large Language Model Acceleration based on KV Cache Management

- Haoyang Li, Yiming Li, Anxin Tian, Tianhao Tang, Zhanchao Xu, Xuejia Chen, Nicole Hu, Wei Dong, Qing Li Fellow, IEEE, Lei Chen Fellow, IEEE
- Improving LLMs through KV Cache
  - Heavy hardware demands by LLMs
  - Challenge to scale up
  - Make LLMs aware of resources used
- KV Cache Management Strategies
  - Token level
  - Model level
  - System level

Symbol	Definition
$X$	Input sequence of tokens
$\mathbf{X}$	Dense representations of $X$
$d_x$	Dimensionality of the input embeddings.
$\mathbf{E}$	Embedding matrix $\mathbf{E} \in \mathbb{R}^{d_{\text{vocab}} \times d_x}$ .
$PE(X)$	Positional encoding
$\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i$	Query, Key, and Value matrices
$d_k, d_v$	Query/Key and Value dimension
$\mathbf{W}_{Q_i}, \mathbf{W}_{K_i}, \mathbf{W}_{V_i}$	Weight matrices for computing $\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i$ .
$\mathbf{Z}_i$	Self-attention Output
$\mathbf{W}_O$	Weight matrix
$\mathbf{W}_1, \mathbf{W}_2$	Weight matrices
$\mathbf{b}_1, \mathbf{b}_2$	Bias vectors
$t$	Sequence length index
$t_c$	Number of tokens stored in the KV cache.
$\mathbf{K}_i^t, \mathbf{V}_i^t$	Key and Value at step $t$
$\hat{\mathbf{K}}_i^{t-1}, \hat{\mathbf{V}}_i^{t-1}$	Cached Key and Value
$h$	Number of attention heads per layer
$L$	Number of transformer layers
$P(x_{t+1} x_1, \dots, x_t)$	Conditional probability

**Akira Durham**  
**zup9su**

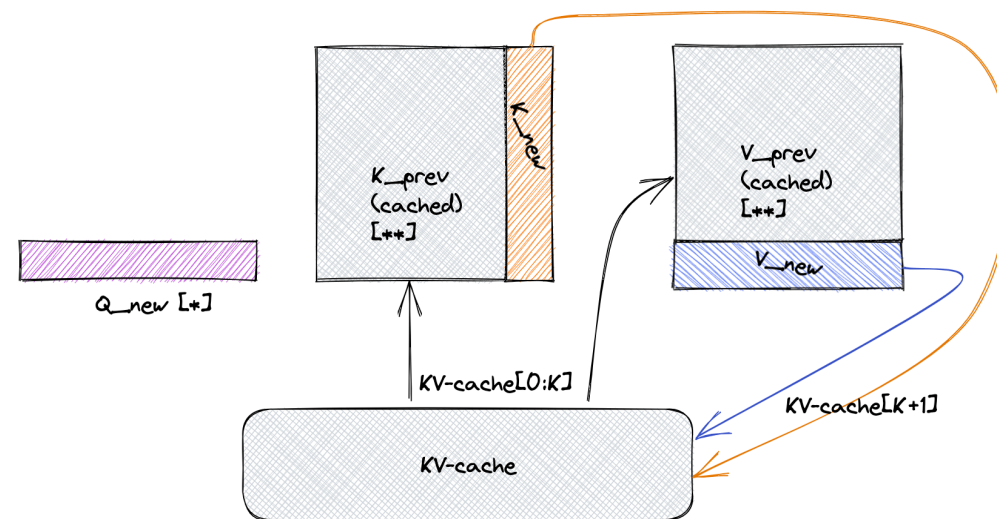
Introduction  
Preliminary  
Taxonomy  
Token Level Optimization





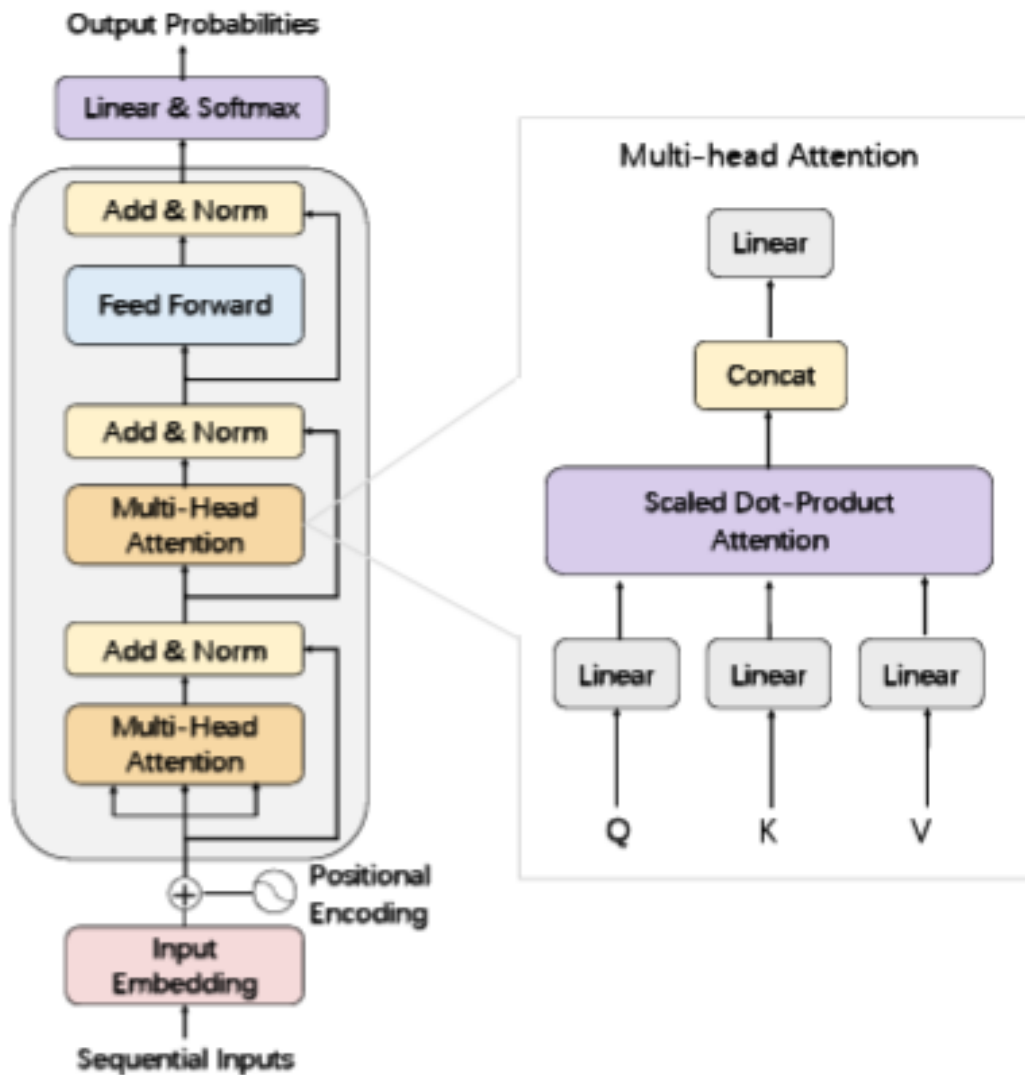
# Introduction

- Transformer Architecture
  - Excels at capturing long-term dependencies
  - Heavy computation and memory demands
- Key-Value Pairs (KV)
  - Critical bottleneck in LLM inference
  - Caching technique that allows model to use past results



Notes:

- \* When processing token  $[K]$ , we only need the  $K$ 'th row of  $Q$
- \*\* When processing token  $[K]$ , we require the full  $K$  &  $V$  tensors, but we can mostly reuse the cached values (This enables skipping the computation of  $K$  &  $V$ )



# Preliminary

- Transformer Architecture
  - Most LLMs follow a decoder only component
  - Composed of stacked Transformer blocks
- Auto-regressive Generation Mechanism
  - LLMs generate text token by token
  - Tokens depend on previously generate tokens
  - Predict next token by applying a softmax
  - Repeat until EOS or max length of response

# KV Cache in Transformer Models

- How KV caching accelerates LLMs' inferencing
  - LLM performs self-attention over the entire token sequence every token
  - Saves previous KV matrices, and reuses instead of recalculating again

- Time and Space Analysis

- Time saved is directly proportional to cached tokens
- Space depends on number of cached tokens and price

- Challenges

- Managing memory as sequence lengths grow
- Cache Eviction Policies, Memory Management, Latency Bottleneck
- Compression Trade-offs, Dynamic Workloads, Distributed Coordination

Formally, at decoding step  $t$ , the new token embedding  $\mathbf{x}_t$  is used to compute the query vector  $\mathbf{q}_i^t$ , key vector  $\mathbf{k}_i^t$ , and value vector  $\mathbf{v}_i^t$  as follows:

$$\mathbf{q}_i^t = \mathbf{x}_t \mathbf{W}_{Q_i}, \quad \mathbf{k}_i^t = \mathbf{x}_t \mathbf{W}_{K_i}, \quad \mathbf{v}_i^t = \mathbf{x}_t \mathbf{W}_{V_i}, \quad (7)$$

The newly computed  $\mathbf{k}_i^t$  and  $\mathbf{v}_i^t$  are then appended to the cached key and value matrices from previous steps:

$$\mathbf{K}_i^t = \text{Concat}(\hat{\mathbf{K}}_i^{t-1}, \mathbf{k}_i^t), \quad \mathbf{V}_i^t = \text{Concat}(\hat{\mathbf{V}}_i^{t-1}, \mathbf{v}_i^t), \quad (8)$$

$$\mathbf{z}_i^t = \text{Softmax} \left( \frac{\mathbf{q}_i^t \mathbf{K}_i^{t \top}}{\sqrt{d_k}} \right) \mathbf{V}_i^t,$$

# Formulas of Time and Space Analysis

• Time

$$O(L \cdot h \cdot t_c \cdot t \cdot (d_k + d_v) + L \cdot h \cdot t_c (\Delta_1 + \Delta_2)) \quad (10)$$

- **QKV Computation.** The time of computing Queries, Keys and Values for each token in Equation (1) is  $\Delta_1 = O(2d_x d_k + d_x d_v)$ .
- **Self-attention Result.** Additionally, computing each attention result  $\mathbf{z}_i$  in Equation (2) takes  $O(t(d_k + d_v))$ .
- **Linear Transformation.** To merge the  $h$  attention results in Equation (3) the time is  $\Delta_2 = O(hd_v + d_v d_o)$ .

Therefore, for  $t_c$  cached tokens across  $h$  attention heads and  $L$  layers, the total saved computation time is:

Space

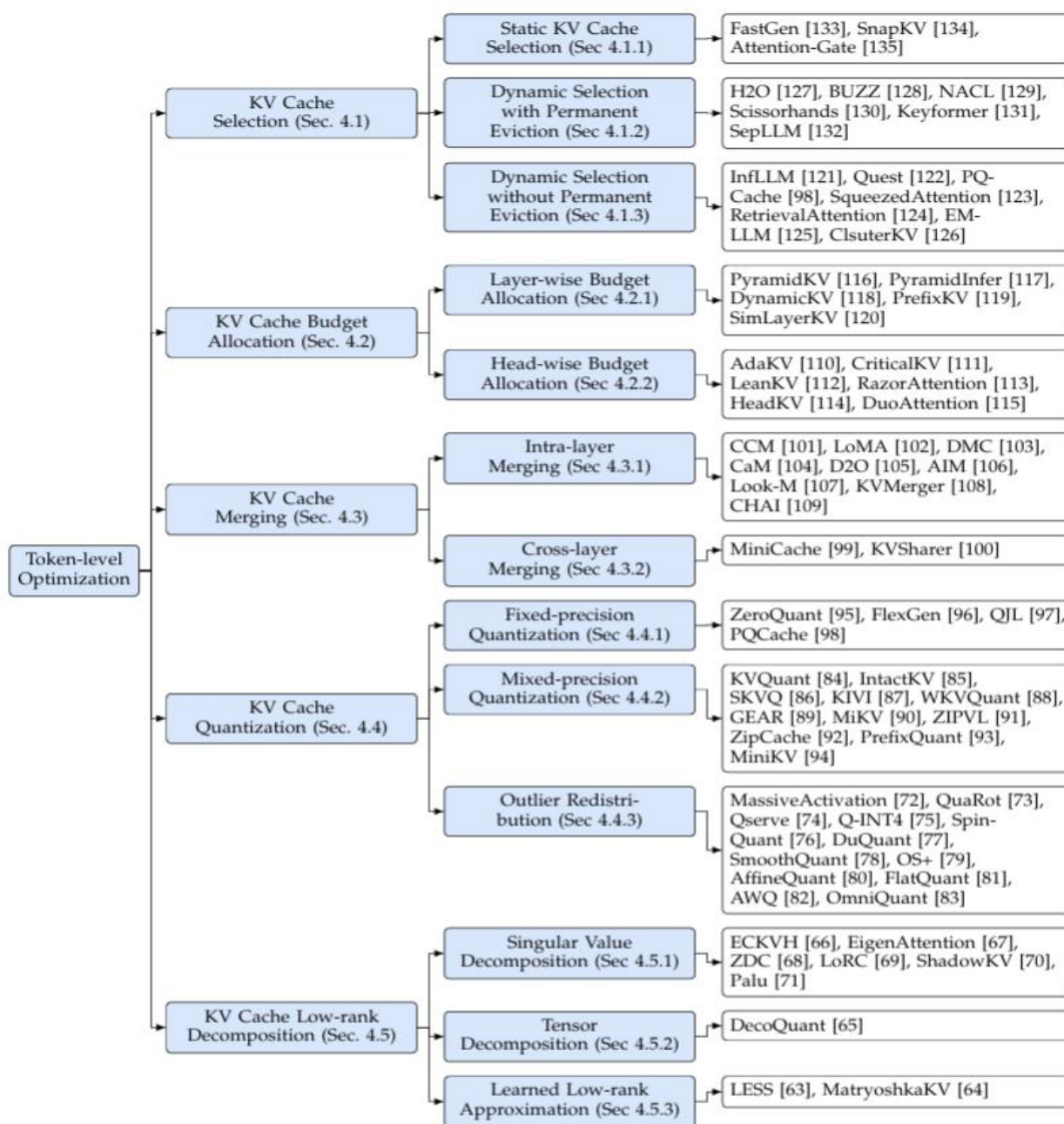
$$O(L \cdot h \cdot t_c \cdot 2 \cdot \text{sizeof}(\text{Float16})) \quad (11)$$

$$\mathbf{Q}_i = \mathbf{XW}_{Q_i}, \quad \mathbf{K}_i = \mathbf{XW}_{K_i}, \quad \mathbf{V}_i = \mathbf{XW}_{V_i}, \quad (1)$$

$$\mathbf{Z}_i = \text{Attention}(\mathbf{Q}_i, \mathbf{K}_i, \mathbf{V}_i) = \text{Softmax}\left(\frac{\mathbf{Q}_i \mathbf{K}_i^\top}{\sqrt{d_k}}\right) \mathbf{V}_i, \quad (2)$$

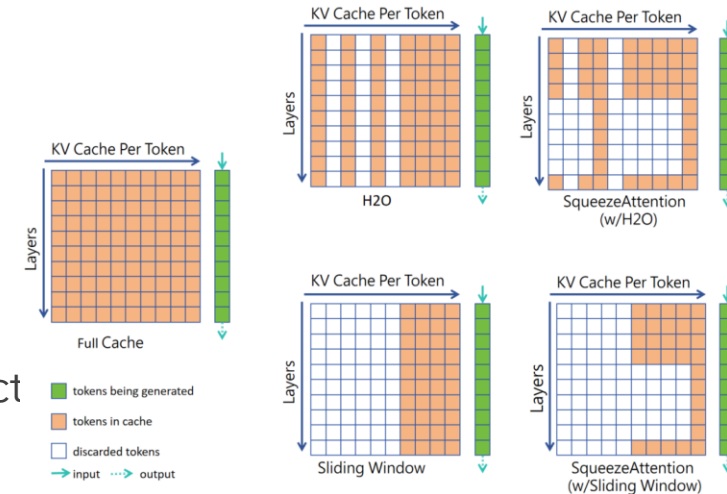
$$\mathbf{Z} = \text{Concat}(\mathbf{Z}_1, \mathbf{Z}_2, \dots, \mathbf{Z}_h) \mathbf{W}_O, \quad (3)$$

# Token Level Optimization



# KV Cache Selection

- Goals: Reduce memory utilization, inference latency, enhance throughput
- Static KV Cache Selection
  - One time compression on KV Cache after initial caching
  - Pattern aware and importance scoring
- Dynamic Selection with Permanent Eviction
  - Continuously update KV Cache during decoding phase
  - Sliding window, accumulative attention scores, diversified random evict
- Dynamic Selection without Permanent Eviction
  - Irreversible eviction of tokens potentially impairs model performance on long sequence tasks
  - Block-level caching, multi-tier storage, clustering methods
- Challenges: Validation on multi-turn dialogue and extended decoding lengths



Method	Initial tokens	Top- $k$ tokens	Recent tokens	Permanent eviction	Dynamic selection	Selection granularity	Remark
FastGen [133]	✓	✓	✓	✓		token	five attention structures
SnapKV [134]		✓	✓	✓		token	observation window-based
Attention-Gate [135]		✓		✓		token	learned eviction policy
StreamingLLM [136]	✓		✓	✓	✓	token	initial and recent tokens
LM-Infinite [137]	✓		✓	✓	✓	token	distance ceiling
H2O [127]		✓	✓	✓	✓	token	accumulative attention score
BUZZ [128]	✓	✓	✓	✓	✓	token	beehive-like structure
Scissorhands [130]		✓	✓	✓	✓	token	persistence of importance
NACL [129]		✓	✓	✓	✓	token	diversified random eviction
Keyformer [131]		✓	✓	✓	✓	token	gumbel logit adjustment
InfLLM [121]	✓	✓	✓		✓	block	block-level KV management
Quest [122]		✓			✓	block	new block representation
PQCache [98]	✓	✓	✓		✓	block	product quantization
SqueezedAttention [123]		✓			✓	cluster	hierarchical clusters
RetrievalAttention [124]	✓	✓	✓		✓	Token	ANN search
EM-LLM [125]	✓	✓	✓		✓	event	episodic events
SparQ [138]		✓	✓		✓	token	low-dimension retrieval
InfiniGen [139]		✓			✓	token	asynchronous prefetching

# KV Cache Budget Allocation

- Goals: Improve inherent heterogeneity across LLM layers' KV Caches
- Layer-wise Budget Allocation
  - Assign different compression ratios across model layers
  - Pyramid shaped memory, attention patterns, per layer token identification
- Head-wise Budget Allocation
  - Finer allocations, precise distribution across individual attention heads within each layer
  - Retrieval head-based methods are specialized category - key information extraction
  - Thresholding, minimize output deviations, retrieval head support
- Challenges: Pyramid vs. Retrieval



# KV Cache Merging

- Goals: Compress KV Caches without degrading accuracy
- Intra-layer Merging
  - Consolidating KV Caches within individual layers
  - Special indicator compression, merging tokens, attention head clusters
- Cross-layer Merging
  - Targets redundancy across layers
  - Combine middle to deep layers and combines very dissimilar layers
- Challenges: Adaptive merging and Preservation of critical information guarantee

Model	Merge Layer		Merge Unit	Merge Metric	Merge Type	Training-free
	Intra-layer	Cross-layer				
CCM [101]	✓		Token	Sliding Window	Many-to-One	×
LoMA [102]	✓		Token	Sliding Window	Many-to-Many	×
DMC [103]	✓		Token	Learned Merge Indicator	Many-to-One	×
D2O [105]	✓		Token	Cosine Similarity	Two-to-One	✓
CaM [104]	✓		Token	Attention Score	Many-to-One	✓
AIM [106]	✓		Token	Cosine Similarity	Many-to-One	✓
Look-M [107]	✓		Token	Cosine Similarity	Many-to-One	✓
KVMerger [108]	✓		Token	Weighted Gaussian Kernel	Many-to-One	✓
CHAI [109]	✓		Head	Attention Score	Many-to-One	✓
MinCache [99]		✓	Token	Angular Distance	Two-to-One	✓
KVSharer [100]		✓	Layer	Euclidean distance	Many-to-One	✓

# KV Cache Quantization

- Goals: Reduce numeric precision to drastically reduce memory size
- Fixed-precision
  - All KVs are quantized to the same bit-width: often suboptimal
  - Per-token individual, product quantization
- Mixed-precision
  - Higher precision to critical parts of the cache
  - Per channel, per impact, per layer
- Outlier redistribution
  - Smooths the outliers in KVs to improve quantization quality
  - Virtual tokens, redistribute outlier values, transformations
- Challenges: Real-time adaptive, multi-modal, hybrid methods

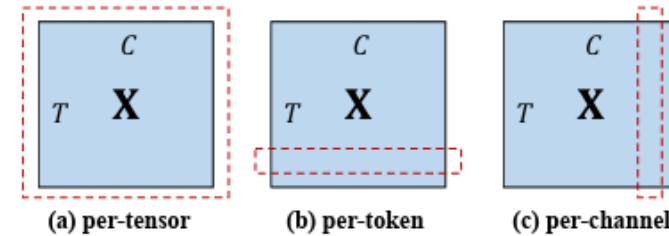


Fig. 4. Three types of quantization. Then matrix  $\mathbf{X} \in \mathbb{R}^{T \times C}$ , where  $T$  is the number of tokens and  $C$  is the feature dimension.

Model	Operation	Formula	Learn	Remarks
MassiveAct. [72]	Add virtual tokens	$\text{softmax}\left(\frac{\mathbf{Q}[\mathbf{K}^T, \mathbf{k}']}{\sqrt{d}}\right) \begin{bmatrix} \mathbf{V} \\ \mathbf{v}'^T \end{bmatrix}$	✓	Learnable $\mathbf{k}', \mathbf{v}'$
QuaRot [73]	Hadamard rotation	$\mathbf{XW}^\top = (\mathbf{XH})(\mathbf{H}^\top \mathbf{W}^\top)$	×	$\mathbf{H}^\top \mathbf{H} = \mathbf{I}$
Qserve [74]	Hadamard rotation	$\mathbf{XW}^\top = (\mathbf{XH})(\mathbf{H}^\top \mathbf{W}^\top)$	×	$\mathbf{H}^\top \mathbf{H} = \mathbf{I}$
Q-INT4 [75]	Hadamard rotation	$\mathbf{XW}^\top = (\mathbf{XH})(\mathbf{H}^\top \mathbf{W}^\top)$	×	$\mathbf{H}^\top \mathbf{H} = \mathbf{I}$
SmoothQuant [78]	Scaling	$(\mathbf{X} \text{diag}(\mathbf{s})^{-1}) \cdot (\text{diag}(\mathbf{s}) \mathbf{W}^\top)$	×	$\mathbf{s} \in \mathbb{R}^{c_i}$
QS+ [79]	Scaling, Shifting	$((\mathbf{X} - \mathbf{z}) \text{diag}(\mathbf{s})^{-1} \cdot \text{diag}(\mathbf{s}) + \mathbf{z}) \mathbf{W}^\top$	×	$\mathbf{s} \in \mathbb{R}^{c_i}$
AWQ [82]	Scaling	$\arg \min_{\mathbf{s}} \ \mathbf{XW}^\top - \mathbf{X} \text{diag}(\mathbf{s})^{-1} Q(\text{diag}(\mathbf{s}) \mathbf{W}^\top)\ $	✓	Quantization $Q(\cdot)$
OmniQuant [83]	Scaling, shifting	$Q_a\left(\frac{\mathbf{X}-\delta}{\mathbf{s}}\right) Q_w(\mathbf{s} \odot \mathbf{W}^\top) + \mathbf{B} + \delta \mathbf{W}^\top$	✓	Learnable $Q_a(\cdot), Q_w(\cdot)$
DuQuant [77]	Rotation, permutation	$[(\mathbf{X} \cdot \mathbf{\Lambda}) \hat{\mathbf{R}}_{(1)} \cdot \mathbf{P} \cdot \hat{\mathbf{R}}_{(2)}] \cdot [\hat{\mathbf{R}}_{(2)}^\top \cdot \mathbf{P}^\top \cdot \hat{\mathbf{R}}_{(1)}^\top (\mathbf{\Lambda}^{-1} \cdot \mathbf{W}^\top)]$	×	Matrices $\mathbf{P}, \mathbf{R}$
AffineQuant [80]	Affine transform	$\arg \min_{\mathbf{P}} \ \mathbf{XW}^\top - \mathbf{XP}^{-1} Q(\mathbf{PW}^\top)\ _F^2$	✓	Quantization $Q(\cdot)$
FlatQuant [81]	Affine transform	AffineQuant + $\mathbf{P} = \mathbf{P}_1 \otimes \mathbf{P}_2$	✓	Decomposition

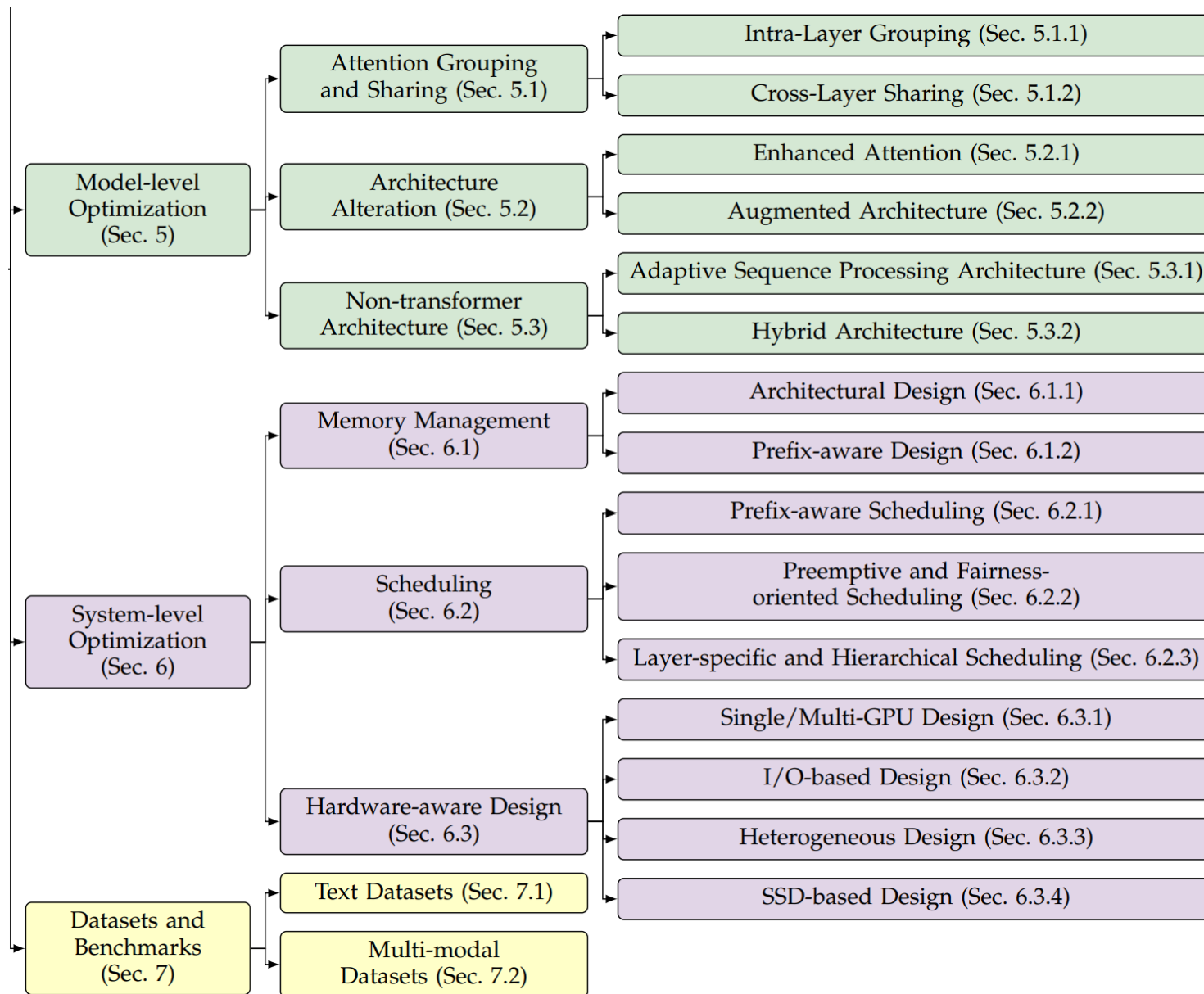
# KV Cache Low-Rank Decomposition

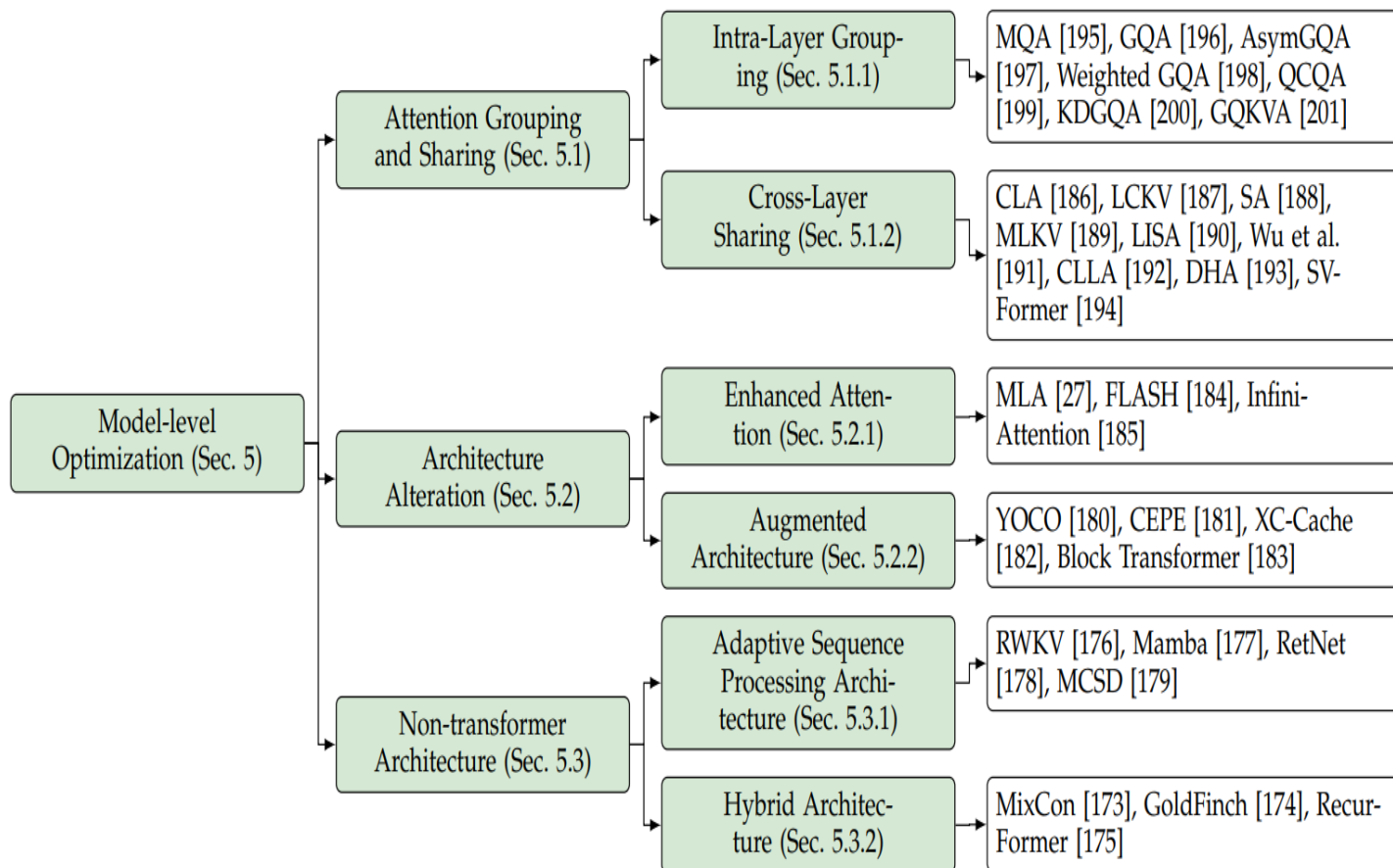
- Goals: Reduce memory requirements while preserving critical information
- Singular Value Decomposition
  - Use low-rank structure of KV matrices to retain most critical singular value
  - Group heads, adaptive hybrid compression, weight matrix replacement
- Tensor Decomposition
  - Factorizes KV matrices into smaller components to reduce redundancy
  - Matrix product operator, KV to local tensors, quantization combination
- Learned Low-Rank Approximation
  - Incorporates adaptive mechanisms to optimize compression with learned representations
  - Learned-kernel-based low rank approximation to approximate the softmax function
- Challenges: Dynamic rank adjustment, real-time/streaming applications

$$\text{TD}(\mathbf{W}) = \prod_{k=1}^n \mathcal{T}_{(k)}[d_{k-1}, i_k, j_k, d_k],$$

$\phi(\mathbf{q}_t)\psi(\mathbf{K}_t)^\top$ , where  $\phi$  and  $\psi$  are row-wise functions. Here,  $\mathbf{q}_t \in \mathbb{R}^{1 \times D}$  represents the query, and  $\mathbf{K}_t \in \mathbb{R}^{t \times D}$  represents the keys at step  $t$ . To elaborate, if  $\phi$  and  $\psi$  are such that:  $a_t = \text{softmax}\left(\frac{\mathbf{q}_t \mathbf{K}_t^\top}{\sqrt{D}}\right) \mathbf{V}_t \approx \frac{\phi(\mathbf{q}_t)\psi(\mathbf{K}_t)^\top \mathbf{V}_t}{\phi(\mathbf{q}_t)\psi(\mathbf{K}_t)^\top \mathbf{1}_{S \times 1}}$ , then we only need to cache the hidden states  $\mathbf{H}_t = \psi(\mathbf{K}_t)^\top \mathbf{V}_t \in \mathbb{R}^{R \times D}$  and the normalization factor  $\mathbf{z}_t = \sum_{s=1}^t \psi([\mathbf{K}_t]_s) \in \mathbb{R}^{1 \times R}$

**Sahlar Salehi**  
**rmh7ce**





# Model Level Optimization



# Attention Grouping and Sharing

- Intra-Layer Grouping
  - Grouping query, key, and value heads within layers -> reduce redundancy
- Cross-Layer Sharing
  - Sharing query, key, and value components across layers
- Goals: Reduce redundancy, improve efficiency/reuse, reduce KV cache requirements
- Challenges: Performance/efficiency tradeoff, scalability, timestep variations in transformer

# Intra-Layer Grouping: MQA/GQA

- Multi-Query Attention (MQA)
  - All attention heads in transformer block share a single key and value
  - Fast decoding + low cache requirements, but unstable
- Grouped Query Attention (GQA) improves on MQA
  - Divide attention heads into groups, share key and values within groups
  - Uptraining processes proposed to convert traditional multiheaded attention to GQA
- Result: GQA model performed as well as MHA and as fast as MQA

Method	Applied Location		Intra-layer Grouped Component	Cross-layer Shared Component	Retraining Required
	Intra-layer	Cross-layer			
MQA [195]	✓		K, V	-	✓
GQA [196]	✓		K, V	-	Uptrain
AsymGQA [197]	✓		K,V	-	Finetune
Weighted GQA [198]	✓		K,V	-	Uptrain & Finetune
QCQA [199]	✓		K, V	-	✓
KDGQA [200]	✓		K, V	-	✓
GQKVA [201]	✓		Q, K, V	-	✓

# Cross-Layer Sharing

- Cross-Layer Attention (CLA)
  - Share key and value heads across transformer layers

◦ 2X |

Method	Applied Location		Intra-layer Grouped Component	Cross-layer Shared Component	Retraining Required
	Intra-layer	Cross-layer			
CLA [186]	✓	✓	K, V	K, V	✓
LCKV [187]		✓	-	K, V	✓
SA [188]		✓	-	Attention Weight	✓
MLKV [189]	✓	✓	K, V	K, V	Uptrain
LISA [190]		✓		Q, K, V	Lightweight adaption
Wu et al. [191]		✓	-	Q, K, V	✓
CLLA [192]		✓	-	Q, K, V	✓
DHA [193]	✓	✓	K, V	Q, K, V	Lightweight adaption
SVFormer [194]		✓	-	V	✓

# Architecture Alteration

- Enhanced Attention Mechanisms
  - DeepSeek-V2 Multi-Head Latent Attention (MLA)
- Augmented Architectures
- Enables longer context window and faster inference time
- Difficult to implement into existing pretrained models

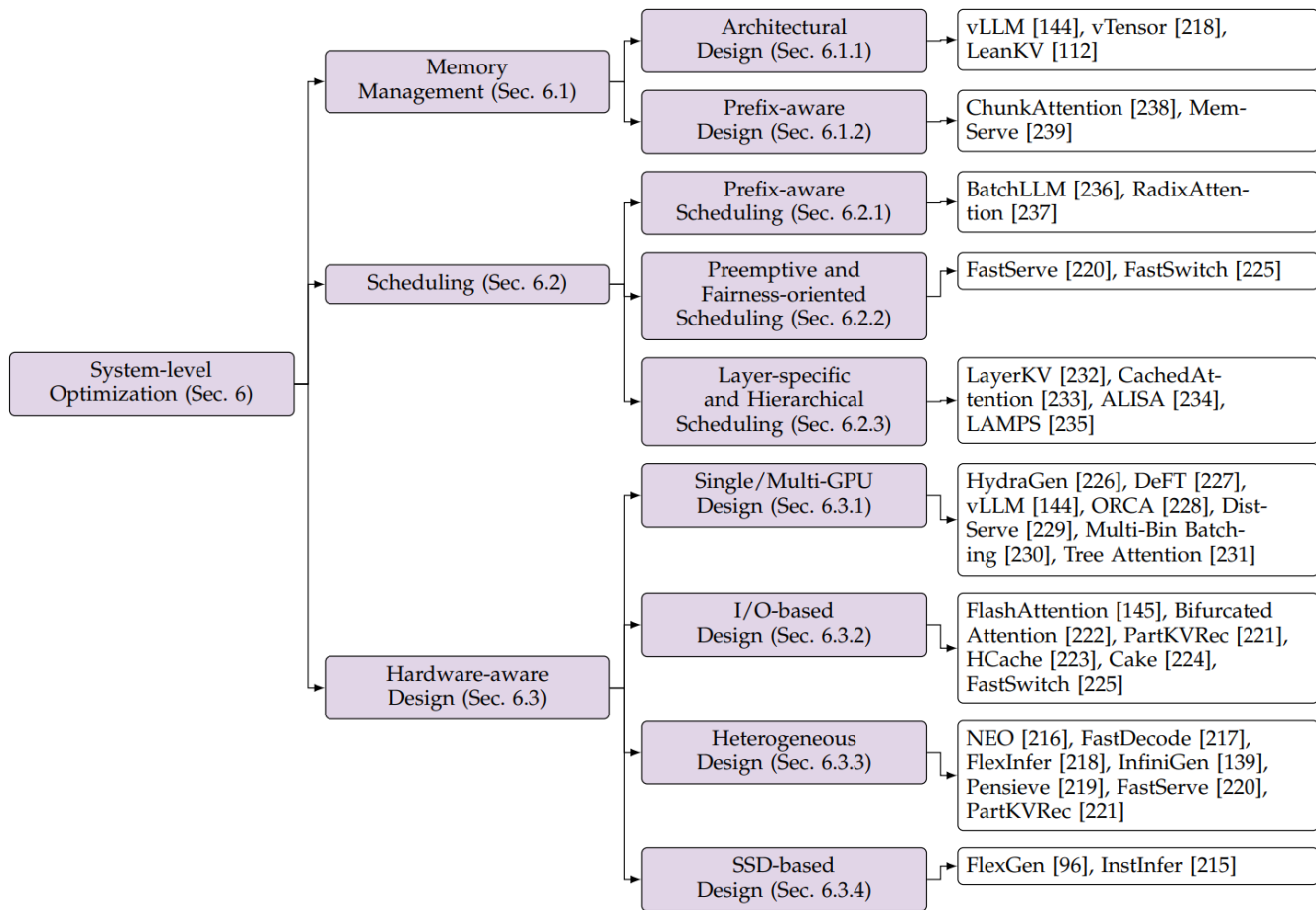
Method	Alteration Type		KV Cache Management	Retraining Requirement
	Enhanced Attention	Augmented Architecture		
MLA [27]	✓		Latent compression	✓
FLASH [184]	✓		Linear approximation	✓
Infini-Attention [185]	✓		Compressive cache	✓
YOCO [180]		✓	Single global KV cache	✓
CEPE [181]		✓	Parallel encoding with cross-attn	Lightweight
XC-Cache [182]		✓	Encoder cross-attention	✓
Block Transformer [183]		✓	Hierarchical local KV	Lightweight

# Non-Transformer Architectures

- Paper focused on architectures that highly compress or compensate for having KV cache
- Combine RNN efficient sequence processing + attention mechanisms parallelizable training
  - Receptance Weighted Key Value (RWKV)
  - Mamba: selectively propagate/forget parameters, performs well on 1M token sequence
- Hybrid Models
  - MixCon: dynamic and high control
  - RecurFormer: identify and replace weak attention heads

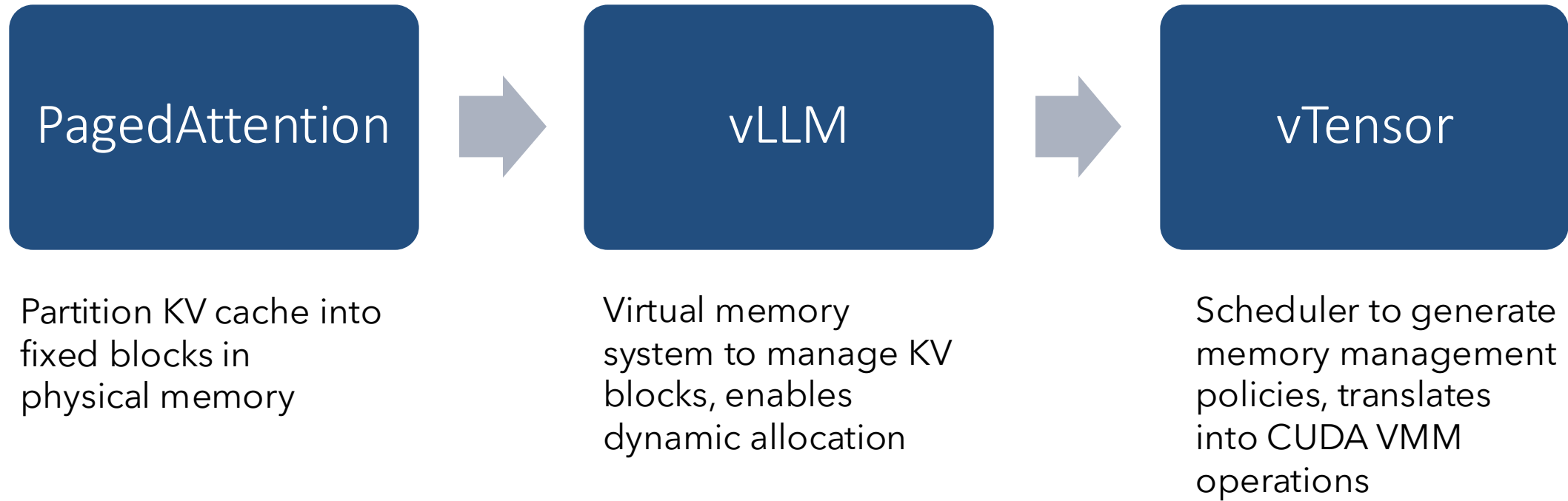
Method	Key Mechanism	No Traditional KV Cache	KV Cache Compression
RWKV [176]	RNN-like with Transformer parallelism	✓	
Mamba [177]	Selective state-space model	✓	
RetNet [178]	Retention mechanism		✓
MCSD [179]	Slope-decay fusion	✓	
MixCon [173]	Transformer + Conba + MoE	✓	
GoldFinch [174]	RWKV + Modified Transformer		✓
RecurFormer [175]	Mamba replacing some attention heads		✓





# System Level Optimization

# Memory Management: Architectural Designs



Method	Paged Memory	Virtual Memory	Dynamic Sparsity	Prefix Sharing	Distributed Memory
vLLM [144]	✓	✓			
vTensor [218]		✓			
LeanKV [112]	✓		✓		
ChunkAttention [238]				✓	
MemServe [239]				✓	✓

# Scheduling

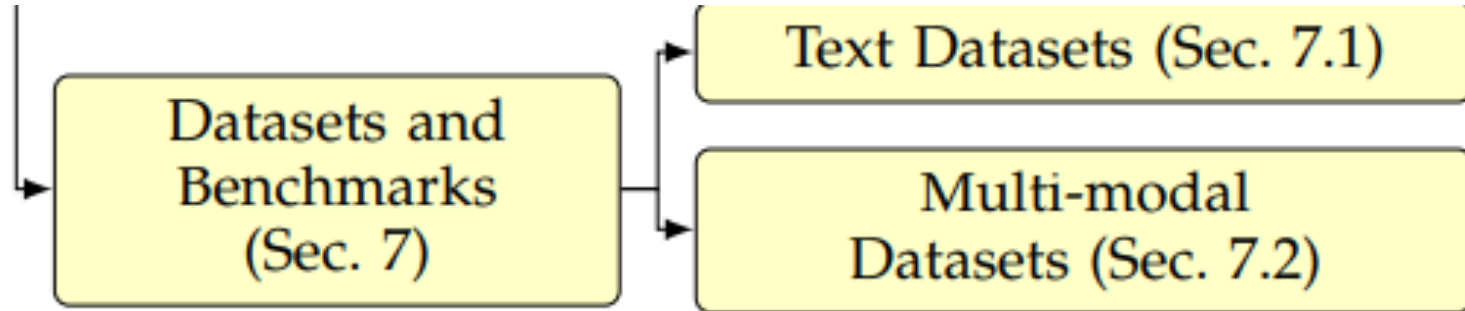
- Prefix Aware
  - BatchLLM: identify global prefixes, schedule cache based on common prefixes
- Preemptive and Fairness Oriented
  - FastServe coordinates cache movement between GPU/host memory
  - FastSwitch balances efficient memory with smooth context switches
- Layer-Specific and Hierarchical
  - LayerKV allocates cache block by layers rather than whole prompt level
- Goals: reduce latency, maximize resource availability

Method	Prefix-aware	Preemptive	Fairness-oriented	Layer-specific	Hierarchical	Dynamic
BatchLLM [236]	✓					
RadixAttention [237]	✓					✓
FastServe [220]		✓	✓			
FastSwitch [225]		✓	✓			
LayerKV [232]				✓		
CachedAttention [233]				✓	✓	
ALISA [234]				✓		✓
LAMPS [235]					✓	✓

# Hardware-Aware Design

- Goal: Optimize KV cache/cache management based on hardware specifications
- Single/Multi GPU designs
  - Efficient memory access patterns and load balancing
- IO-Based Designs
  - Optimize data movement across memory hierarchies (CPU, GPU, disk, etc)
- Heterogenous Designs
  - Maximize resource utilization via CPU-GPU collaboration
- SSD-Based Solutions
  - Extending hierarchy across GPU, CPU => optimize LLM inference on constrained hardware

Method	Single/Multi-GPU	I/O-aware	Heterogeneous	SSD-based
Bifurcated Attention [222]		✓		
Cake [224]				✓
DeFT [227]	✓			
DistServe [229]			✓	
FastDecode [217]		✓		
FastSwitch [225]	✓			
FlexGen [96]		✓		
FlexInfer [218]				✓
FlashAttention [145]	✓		✓	
HCache [223]			✓	
HydraGen [226]	✓			
InfiniGen [139]			✓	
InstInfer [215]				
Multi-Bin Batching [230]				✓
NEO [216]			✓	
ORCA [228]	✓			
PartKVRec [221]		✓		
Pensieve [219]		✓		
Tree Attention [231]		✓		
vLLM [144]	✓			



# Datasets and Benchmark



# Question Answering Tasks

- Model given document(s) and question(s) as input
- Answer either closed (multiple choice) or open ended depending on question
- Single document (QA-SG) vs multi document (QA-MT)

Task	Name	Source	Instances	Avg Len	Metric	Lang.
QA	AltQA [253]	Wikipedia	200/200	3243/13,084 Tok	Acc	EN
QA	PaperQA(BAMBOO) [246]	Paper	100/100	3101/6838 Tok	Acc	EN
QA	MeetingQA(BAMBOO) [246]	Meeting	100/100	2738/9838 Tok	Acc	EN
QA	TriviaQA [254]	Web Question, Wiki	95,956 Q, 662,659 Doc	17,370 W	EM, F1	EN
QA	TOEFL(L-Eval) [244]	TOFEL-QA [255]	15 Doc, 269 Inst	3907 Tok	Rouge-L, GPT-4, $\Delta L$	EN
QA	Coursera(L-Eval) [244]	Video Subtitles	15 Doc, 172 Inst	9075 Tok	Rouge-L, GPT-4, $\Delta L$	EN
QA	SFiction(L-Eval) [244]	SFGram [256], fiction	7 Doc, 64 Inst	16,381 Tok	Rouge-L, GPT-4, $\Delta L$	EN
QA	LongFQA(L-Eval) [244]	Financial Transcripts	6 Doc, 52 Inst	6032 Tok	Rouge-L, GPT-4, $\Delta L$	EN
QA	CUAD(L-Eval) [244]	CUAD [257]	20 Doc, 130 Inst	30,966 Tok	Rouge-L, GPT-4, $\Delta L$	EN
QA	DuoRC [245]	Movie		3572 W	Acc	EN
QA	NQ [258]	Wiki	307,373	9005 W	Rouge	EN
QA-SG	NarrativeQA [259]	Story	1572 Doc	62,528 Tok	BLEU, METEOR, Rouge-L, MRR	EN
QA-SG	NarrativeQA(LongBench) [247]	Story	200	18,409 W	F1	EN
QA-SG	Qasper [260]	Paper	1585	5001 W	F1	EN
QA-SG	Qasper(LongBench) [247]	Paper	200	3619 W	F1	EN
QA-SG	MultifieldQA-en [247]	Paper, Legal, Gov, Google	200	4459 W	F1	EN
QA-SG	MultifieldQA-zh [261]	Paper, Legal, Gov, Google	200	6701 W	F1	ZH
QA-SG	QuALITY [262]	Story, magazine	381 Doc, 6737 Q	4203 W	EM	EN
QA-MT	HotpotQA [261]	Wiki	112,779	1138 W	EM, F1	EN
QA-MT	HotpotQA(LongBench) [247]	Wiki	200	9151 W	F1	EN
QA-MT	2WikiMultihopQA [263]	Wiki	192,606 Q	639 W	EM, F1	EN
QA-MT	MuSiQue [264]	Wiki	24,814	1827 W	F1	EN
QA-MT	DuReader [265]	Baidu	200,000 Q, 1,000,000 Doc	396 W	BLEU, Rouge-L	ZH,EN
QA+RET	NewsQA(M4LE) [245]	News	-	3679 W	Acc	EN
QA+RET	C3(M4LE) [245]	Textbook	-	3797 W	Acc	ZH

# Text Summarization Tasks

- Datasets include curated selection of texts and corresponding summaries

Task	Name	Source	Instances	Avg Len	Metric	Lang.
SUM	CNN/Dailymail [266]	News	300,000	766 W	Rouge-1/2/L	EN
SUM	XSum [267]	News	400,000	431 W	Rouge-1/2/L	EN
SUM	QMSum [268]	Meeting	232 Meets, 1808 Q	9070 W	Rouge-1/2/L	EN
SUM	MultiNews [269]	News	51,216	5866 W	Rouge-1/2/SU	EN
SUM-QB+ Reasoning+ QA	LooGLE [251]	Papers, Wiki, Movie, TV	776 Doc, 6448 Q	19,367 W 24,005 Tok	BLEU, Rouge, METEOR, BERT, GP14, EM, PM	EN,ZH
SUM	GovReport [270]	Gov	19,466	9409.4 W	Rouge-1/2/L	EN
SUM	VCSUM [271]	Meeting	239	14,107 Tok	F1, Gold Rouge-1	ZH
SUM	SummScreenFD [272]	TV	269,000	6613 Tok	Rouge	EN
SUM	BigPatent [273]	Patent	1,341,362	3573 W	Rouge-1/2/L	EN
SUM	SPACE [274]	Review	50 Entities, 1,140,000 Reviews, 100R/Ent, 1050 Sum	15,532 W	Rouge-1/2/L	EN
SUM	SQuALITY [275]	Story	625	5200 W	Rouge-1/2/L, METEOR, BERT	EN
SUM+RET	CNNNews(M4LE) [245]	News	-	3754 W	Rouge-L	EN
SUM+RET	CEPSUM(M4LE) [245]	E-Commerce	-	4003 W	Rouge-L	ZH
SUM+RET	LCSTS(M4LE) [245]	News	-	4102 W	Rouge-L	ZH
SUM+RET	NCLS(M4LE) [245]	NCLS [276]	-	3470 W	Rouge-L	EN,ZH
SUM+RET	WikiHow [245]	Wiki	-	3514 W	Rouge-L	EN
SUM+RET	News2016 [245]	News	-	3785 W	Rouge-L	ZH
SUM	Pubmed(M4LE) [245]	Medical	1267	3678 W	Rouge-L	EN
SUM	BookSum(M4LE) [245]	Book	-	2643 W	Rouge-L	EN
SUM	CNewsum(M4LE) [245]	News	690	1883 W	Rouge-L	ZH
SUM	CLTS+(M4LE) [245]	News	-	3158 W	Rouge-L	ZH
SUM	Arxiv(M4LE) [245]	Paper	1550	3748 W	Rouge-L	EN

# Text Reasoning Tasks

- Given text, model tested on solving problems, drawing logical conclusions, making inferences
- Finding patterns, relationships rules

- Natural Language Inference (NLI)

- Determining premises

Task	Name	Source	Instances	Avg Len	Metric	Lang.
CLS/Reasoning	Long ListOps [248]	Generated	100,003	3106 W	Acc	EN
Reasoning	ContractNLI [277]	Legal	10,319	2254 Tok	EM	EN
CLS	LSHT(LongBench) [247]	News	200	22,337 W	Acc	ZH
Reasoning	GSM(16 shot) [244]	GSM8K [278]	100 Doc, 100 Inst	5557 Tok	Rouge-L, GPT-4, $\Delta L$	EN
Reasoning	SenHallu(BAMBOO) [246]	Paper	200/200	3170/6357 Tok	Precision, Recall, F1	EN
Reasoning	AbsHallu(BAMBOO) [246]	Paper	200/200	3314/6445 Tok	Precision, Recall, F1	EN
CLS	MNDS News [279]	News	10,917	637 W	Acc	EN

# Text Retrieval Tasks

- Retrieve information from a large amount of data, tests

que  
effi  
rele

Task	Name	Source	Instances	Avg Len	Metric	Lang.
CLS/RET	TREC(LongBench) [247]	Web Question	200	5177 W	Acc	EN
RET	LongEval [252]	Conversations	-	-	Acc	EN
RET	StreamingEval [136]	LongChat [252]	-	-	Acc	EN
RET	TopicRet(L-Eval) [244]	LongChat [252]	-	-	Acc	EN
RET	DRCD(M4LE) [245]	Wiki	-	3617 W	Acc	ZH
CLS+RET	MARC [245]	E-Commerce	2200	3543 W	F1	EN,ZH
CLS+RET	Online Shopping(M4LE) [245]	E-Commerce	2200	3714 W	F1	ZH
CLS+RET	MNDS News(M4LE) [245]	MNDS News [279]	-	3805 W	Acc	EN
CLS+RET	THUCNews(M4LE) [245]	News	-	3721 W	Acc	ZH

# Text Generation Tasks

- Generate content based on task specifications

Task	Name	Source	Instances	Avg Len	Metric	Lang.
GEN	LCC [282]	Code	360000	1337 W	EM, Edit Sim	Python/CSharp/Java
GEN	RepoBench-P(LongBench) [247]	Code	500	4206 W	Edit Sim	Python/Java
GEN/RET	MultiDoc2Dial [283]	Doc2Dial [284]	488 Doc, 4796 Dialogues	4283 T	F1, EM, SacreBLEU, Recall	EN
GEN	OpenReview(L-Eval) [244]	ASAP-Review [285]	20 Doc 60 Inst	11,170 Tok	Rouge-L, GPT-4, $\Delta L$	EN
GEN	ASAP-Review [285]	Paper	8877 Papers, 25,986 Reviews	6782 W/Paper	Rouge-1/2/L, BERT	EN
GEN	ShowsPred [246]	TV Shows	100/100	2389/4860 Tok	Acc	EN
GEN	MeetingPred [246]	Meeting	100/100	3689/11578 Tok	Acc	EN
GEN-Code	PrivateEval [246]	Code	152/152	3149/6230 Tok	Pass@1	EN, Python
GEN-Code	CodeU(L-Eval) [244]	Code	90 Doc 10 Inst	31,575 Tok	Rouge-L, GPT-4, $\Delta L$	Python

# Aggregation Tasks

- Aggregate varying information from dataset to answer complex questions

○ E:

Task	Name	Source	Instances	Avg Len	Metric	Lang.
AGG	SpaceDigest [250]	Reviews	500	5481 W	ES	EN
AGG	BookSumSort [250]	Literature	500	6840 W	CI	EN
AGG	PassageRetrieval-en [247]	Wiki	200	9289 W	Acc	EN
AGG	PassageRetrieval-zh [247]	C4 Dataset	200	6745 W	Acc	ZH
AGG	PassageCount [247]	Wiki	200	11,141 W	Acc	EN
AGG	ShowsReport(BAMBOO) [246]	TV Shows	200/200	2992/6411 Tok	CI	EN
AGG	ReportSumSort(BAMBOO) [246]	Reports	150/150	3753/8309 Tok	CI	EN

# Multimodal Dataset Tasks

- Datasets include image, text, and video formats
- Testing description, reasoning, conversation, perception, prediction among other tasks

Tasks	Name	Data	Source	Instance	Average	Metric	Language
Conv, Desc, Reas	LLaVA-Bench [294]	Img, T	COCO, In-The-Wild	54 Img, 150 Q	1 Img, 59.9 W	Relative Score	EN
Perc, Reas	MMBench [295]	Img, T	Internet	2948 Q	1 Img, 114.5 W	Acc	EN/CN
Pred, Count, NIH, Retrieval	MileBench [296]	Img, T	Public, self-building	6440 Q	15.2 Img, 422.3 W	Acc, ROUGE-L	EN
Reas, NIH, SUMM, Desc, Order, Count	MLVU [297]	V, T	Public, self-collection	1334 V, 2593 Q	704.6s V, 39.7 W	M-Avg, G-Avg	EN
Reas, Retrieval	LongVideoBench [298]	V, T	web-collected	3763 V, 6678 Q	730.5s V, 49.5 W	Acc	EN
Perc, Recognition, Reas	Video-MME [299]	V, T	YouTube	900 V, 2700 Q	1017.9s V	Acc	EN
Desc, Reas	NExT-QA [300]	V, T	YouTube, TV Show, Public	1000 V, 47962 Q	44s V, 25.5 W	Acc, WUPS	EN
Perc, Count, Reas	MVBench [301]	V, T	Public	4000 Q	16.7s V, 31.3 W	Acc	EN
Decs	MSVD-QA [302]	V, T	MSVD	1970 V, 50505 Q	10s V	Acc	EN
Desc	MSRVYY-QA [302]	V, T	MSRVTT	10000 V, 243690 Q	15s V	Acc	EN



**Thank you!**