Maciej Besta, Lorenzo Paleari

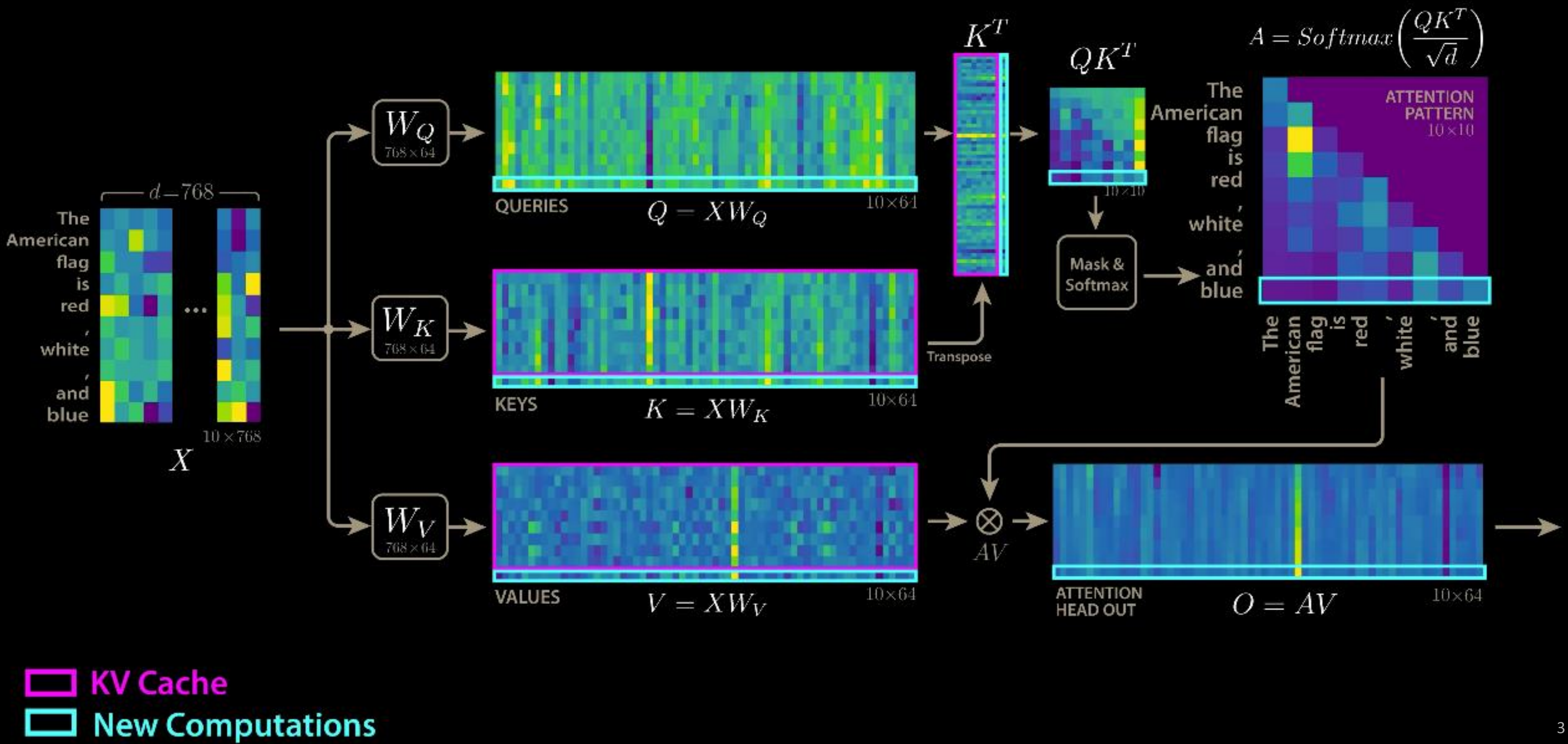
09.07.2025

Graphs & LLMs: Synergy

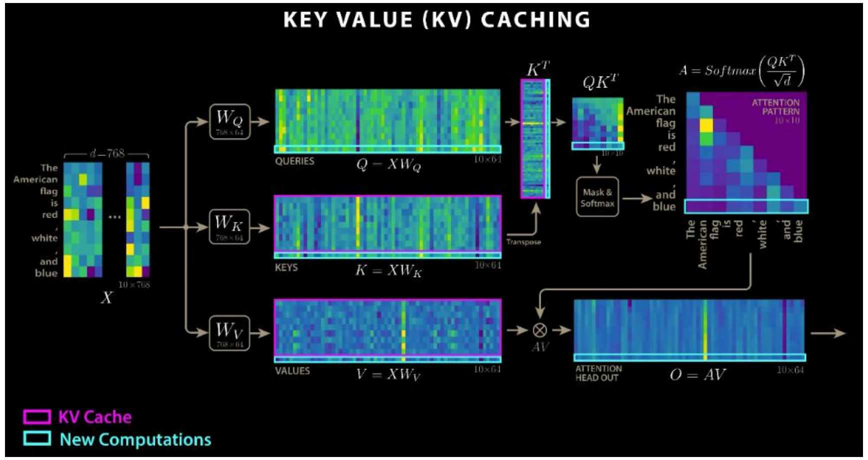
Recent advances

- Preparing AAAI and TPAMI submissions,
- Investigate Asynchronous Pipelining (explore how to reduce latency and improve GPU utilization through off-policy or decoupled training strategies),
- „Distilling” Transformers through the Information Theory Bottleneck (in progress),
- KV cache investigations (Lorenzo).

KEY VALUE (KV) CACHING



KV-Cache: Towards Performance-Centric Formalization



Objective

- Faster Inference

How we obtain this

- Caching Key and Values Matrices
- Save computation

KV-Cache Downside

- Huge amount of memory consumption
  - $2\ x\ N_{head}\ x\ d_{head}\ x\ N_{layers}\ x\ C_{length}\ x\ 2\ (16\ bit)$
  - Memory needed
  - Data movement needed
- What happens when it does not fit in a GPU?
  - Ring Attention
    - Many GPUs used → Reduce number of GPUs
    - Reduce Communication Overhead

- **How do we reduce KV-Cache, while still maintaining same performance/expressiveness?**

How much computation we save in an ideal setting – B=1

- $X \times W_K,\ X \times W_V \rightarrow 2 \times (C_{length}, d_{model}) \times (d_{model}, d_{head}) = C_s \rightarrow O(C_{length} \times d_{head} \times d_{model}))$
- Per each Head and Layer → $C_s \times L \times H$
- Per each Forward Pass
- What is left to be computed for each head are 6 VMM→$O(C_{length} \times d_{head})$

KV-Cache: Overview of Optimizations

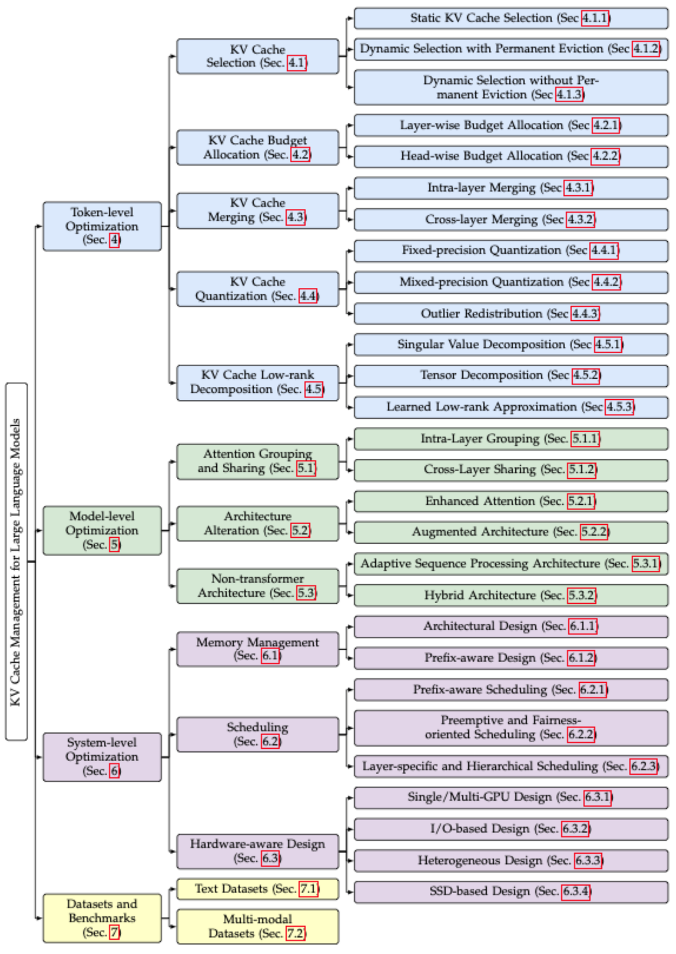


Fig. 2. Taxonomy of KV Cache Management for Large Language Models.

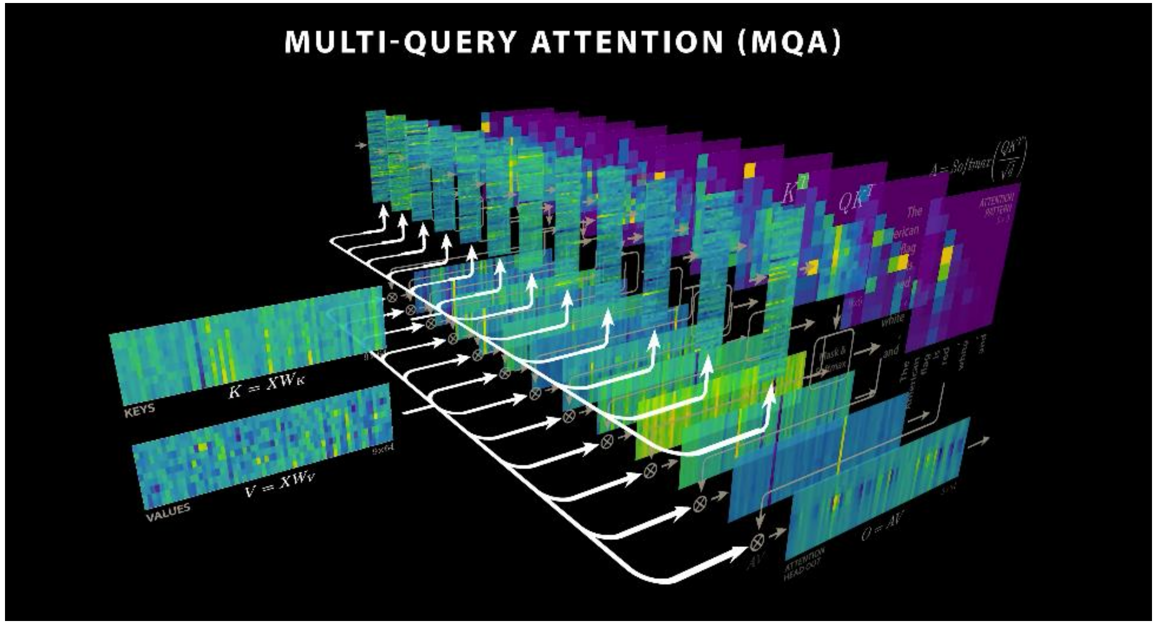
- Token-Level Optimizations
- Model-Level Optimizations
- System-Level Optimizations

- Token-Level
  - Selection / Merging / Quantization / Low-Rank Reduction
  - Fine-Grained focus
  - No-architectural changes
  - **Always Applicable**
- **Model-Level**
  - Grouping / Sharing / Architectural Changes
  - Model-Structure Changes – Transformer / Attention
  - **NOT Always Applicable**
- System-Level
  - Memory Management / Scheduling / Multi-GPU / I-O improvements
  - vLLM – Paged Attention
  - Block-Wise Attention / Ring Attention / Flash Attention

KV-Cache - MQA

- 1 KV Head shared for all the Attention Mechanism in a single Layer.
- Need Retraining
- Very low representation power. To achieve good performance in benchmark we need bigger heads and thus more flops than required by standard KV-Cache.

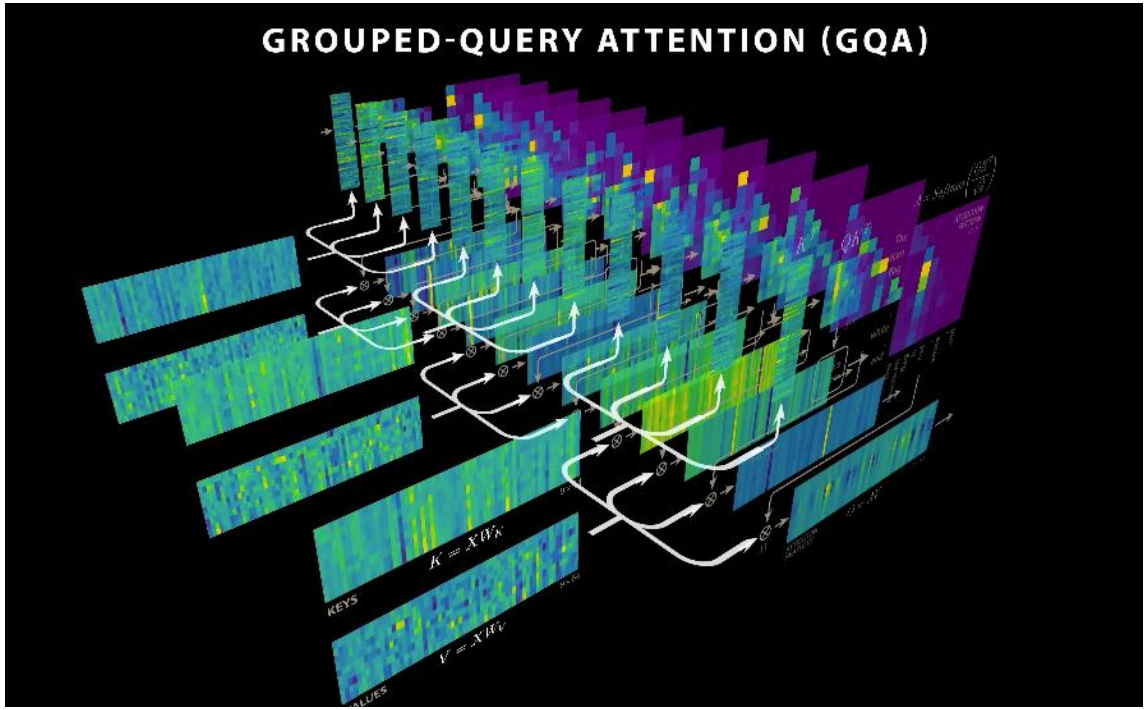
- Reduction of KV Cache size proportional to the number of Attention heads in each layer.



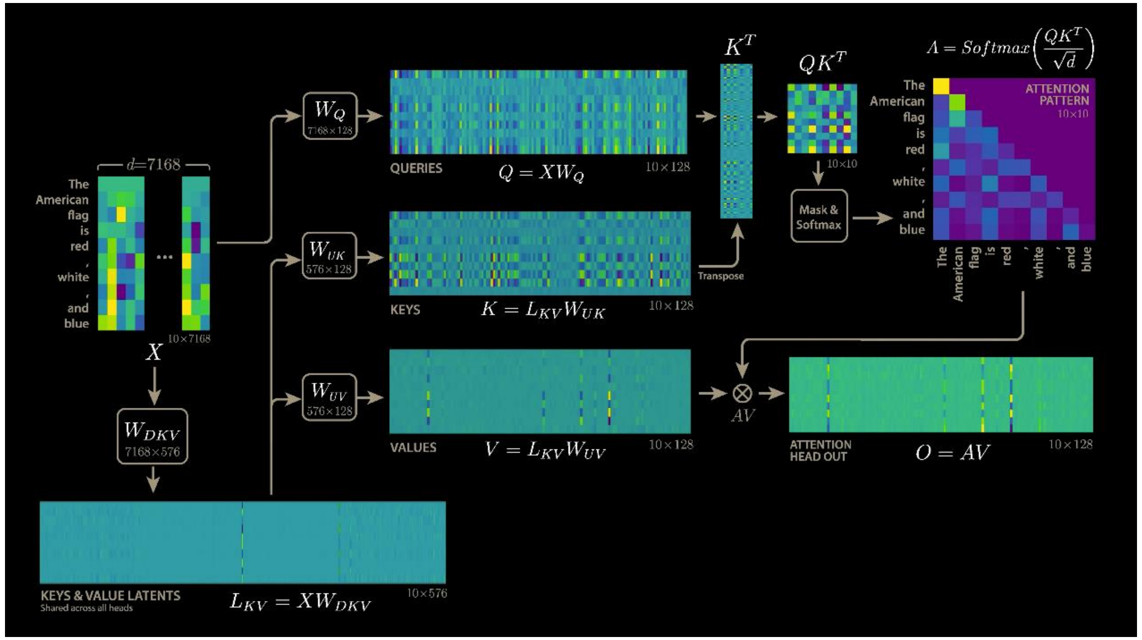
KV-Cache - GQA

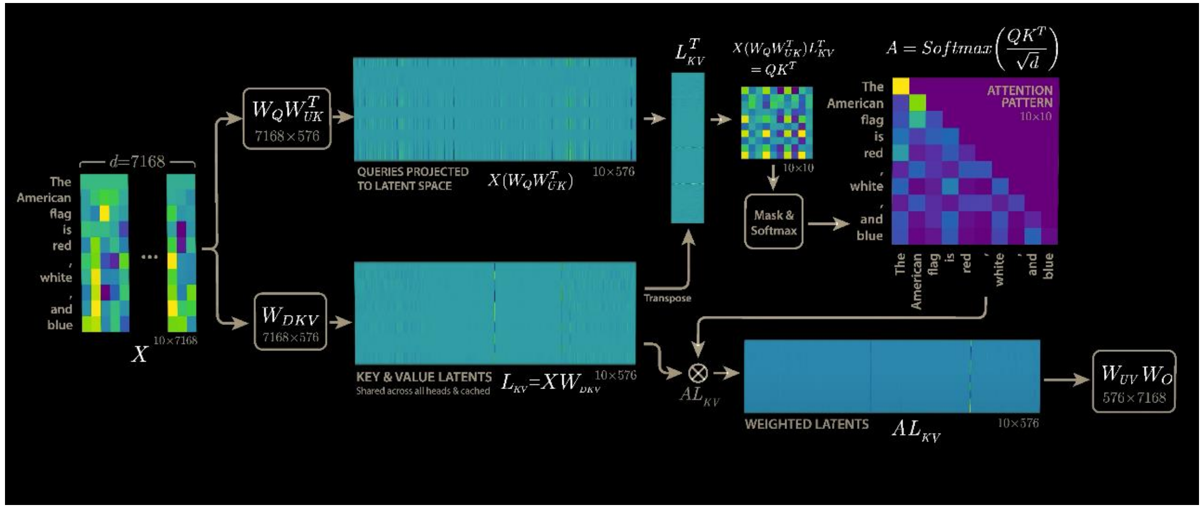
- K KV matrices shared for all the Attention Mechanism in a single Layer
- Needs Uptraining/Retraining - 5% of original training data needed.
- Proposed method to convert existing MHA into GQA by mean-pooling
- Proposed method to uptrain MQA from MHA after mean-pooling

- Reduction of KV Cache size proportional to the number of Attention heads in each layer divided by the number of shared KV matrices



KV-Cache - MLA





- 1 Latent representation shared between all the Head in a layer
- Low-Rank Decomposition
- Need Retraining/Up Training

- *TransMLA: MLA Is All You Need*
    - Proposed method to convert GQA to MLA
    - MLA >> GQA (on same dimension)

KV-Cache – Notable Others

MLKV – Multi-Layer Key-Value

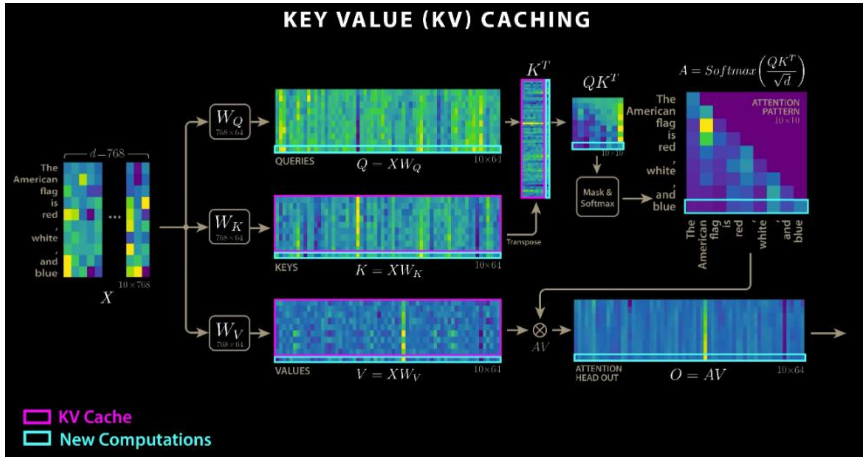
- 1 KV Head for ALL
- Need Retraining/Up Training (Quite large)
- FFN emulate key-value memories, so not all is lost in this passages
- Variable number of Heads per-layer
- Good score retention until 4 head maintained. 2 Starts to lose more and 1 is completely trash

Method	KV Cache Size (# Elements)	Cache Size of OPT-175B (GB)
MHA	$2bslhd_k$	144.0
MQA	$2bsld_k$	1.5
GQA	$2bslgd_k$	36.0
MLKV	$2bsmgd_k$	0.375

CLLA – Cross-Layer Latent Attention

- MLA
- Latent Attention used Cross Layer also, in grouping or alone
- Quantization gets applied
- 2% of original KV-Cache for good performance

KV-Cache



Token-Level Optimizations

- How do we “Retain” the most amount of important information while “discarding” all the other information

- Selection / Merging / Quantization / Low-Rank Decomposition

- Reduce dimension of KV-Cache
- Higher/Lower amount of computation

Model-Level Optimizations

- How do we ”Rearrange” the Attention Mechanism to be more efficient while preserving expressiveness?

- Grouping / Sharing / Enhancing

- Reduce dimension of KV-Cache
- Higher/**Lower** amount of computation
- Methods are not at Inference Time → Orthogonal to **Token-Level Optimizations**

System-Level Optimizations

- Given KV-Caching. How do we use at its best the “Outer World” to be as efficient as possible.

- FlashAttention / PageAttention / RingAttention…

- More efficient communication, use of bandwidth, computation