

Making Long-Context Attention Efficient

Lecture 6: Compact Representations

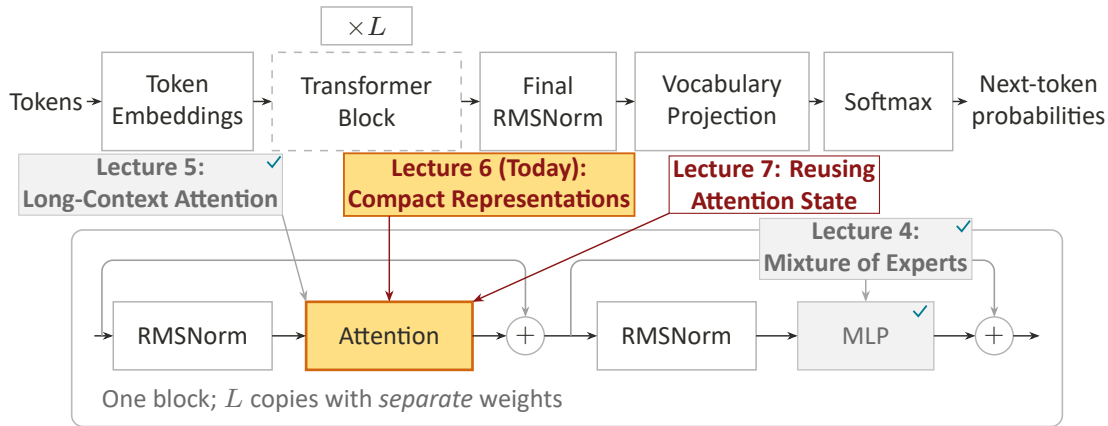
Anay Mehrotra

MS&E 319

Autumn 2026



Topic 2: Innovations in architecture

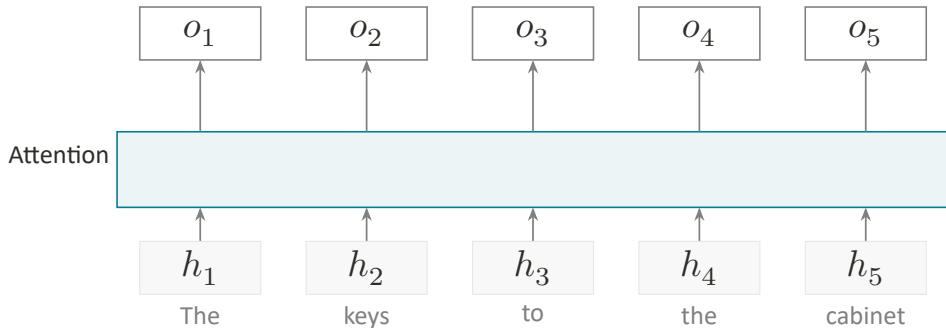


Can we reduce the cost per token while keeping quality fixed?



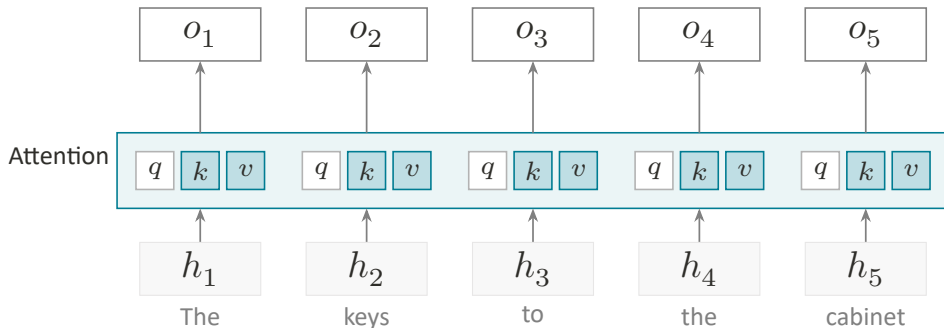
Attention: inputs and outputs

Recall (Lectures 2 and 5): attention maps T inputs to T outputs.



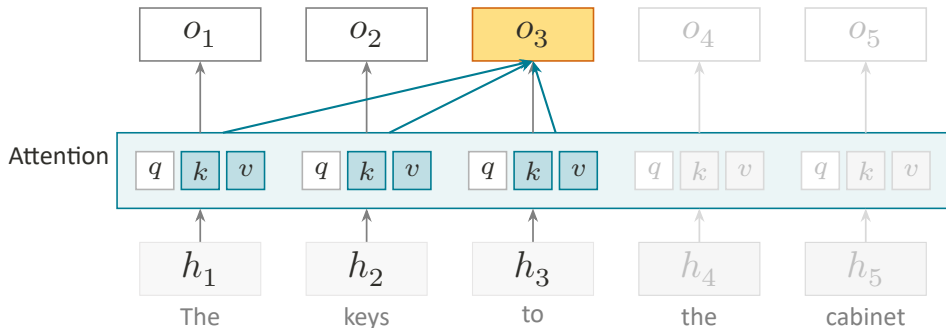
Attention: inputs and outputs

Recall (Lectures 2 and 5): attention maps T inputs to T outputs.



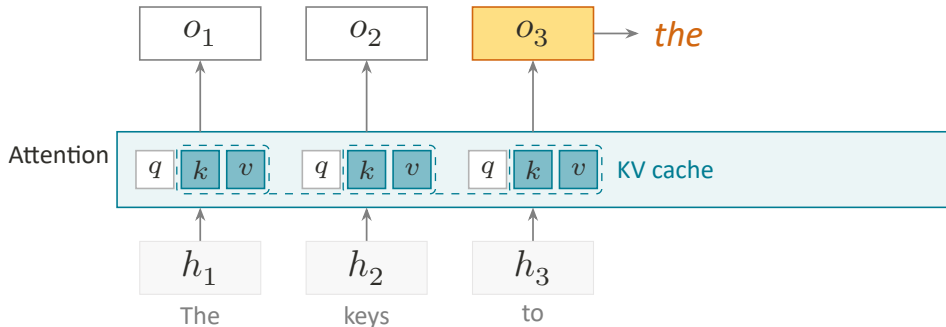
Attention: inputs and outputs

Recall (Lectures 2 and 5): attention maps T inputs to T outputs.



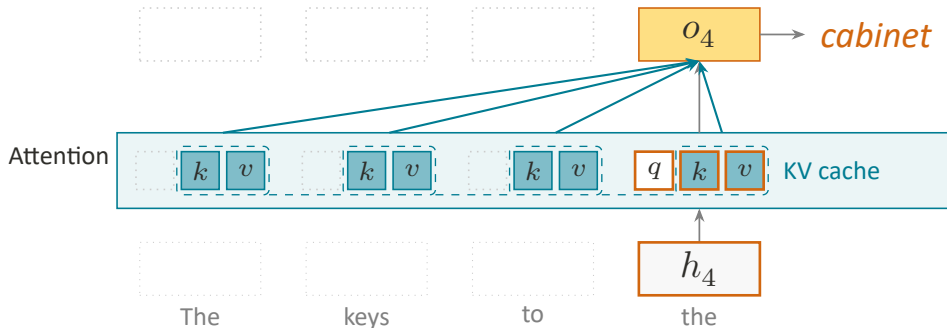
Recap: KV caching

Prefill: compute and store k_j, v_j for each prompt position once.



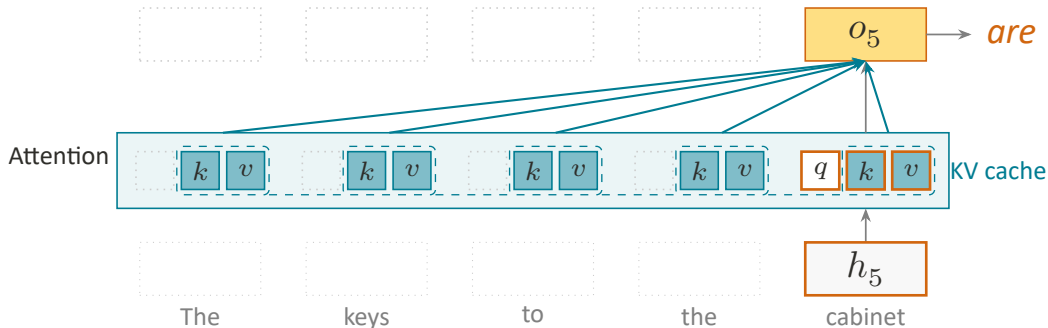
Recap: KV caching

At position 4, append (k_4, v_4) and read all four cached entries.



Recap: KV caching

At position 5, append (k_5, v_5) and read all five cached entries.

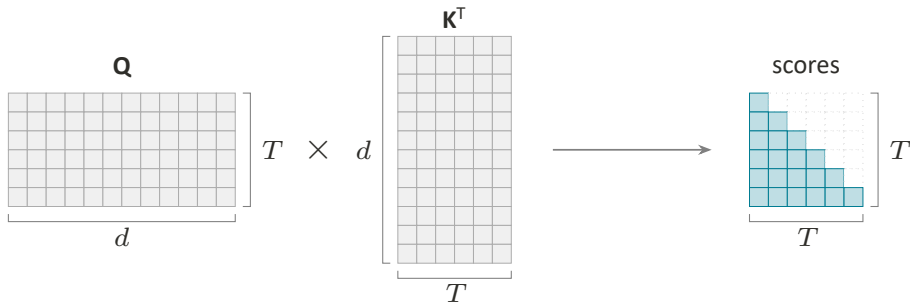


The KV cache stores earlier keys and values in every layer.



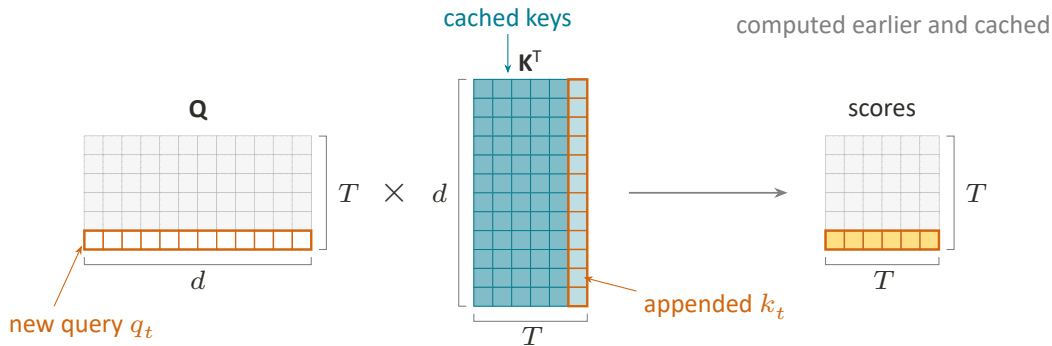
KV caching in the score matrix

Prefill computes $\mathbf{QK}^T$ at once, with a causal mask.



KV caching in the score matrix

Decode: append query 6 and key 6, then compute the last score row.



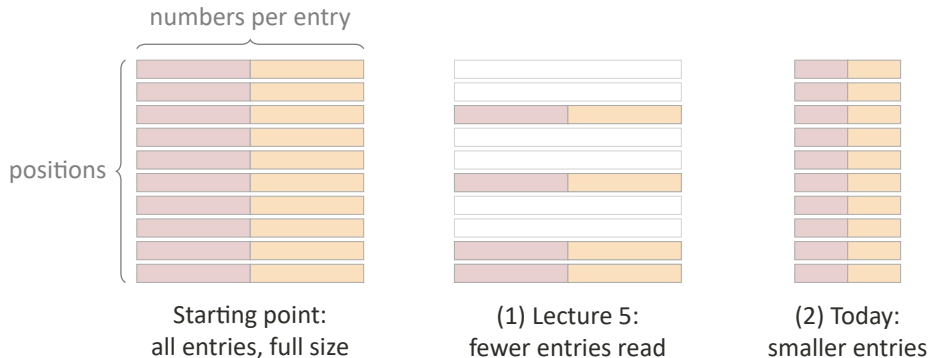
The new query row and key column require one new row of scores.

Likewise, row t of the weights multiplies every cached value.



Two ways to reduce the KV cache

Each layer caches keys and values for every position.



(1) Lecture 5: reduce the number of entries.

(2) Today: reduce the size of each entry.



Size of the KV cache

Each layer stores a key and a value for every cached position:

$$M_{\text{KV}} = \underbrace{B}_{\text{sequences}} \cdot \underbrace{T}_{\text{positions}} \cdot \underbrace{L}_{\text{layers}} \cdot \underbrace{2d}_{k_j, v_j} \cdot \underbrace{b}_{\text{bytes per number}}$$

d : the width of each key and value, as in Lectures 2 and 5.

Generating each new token requires reading this cache in every layer.



How to reduce the KV cache

We can reduce the cache by reducing these factors:

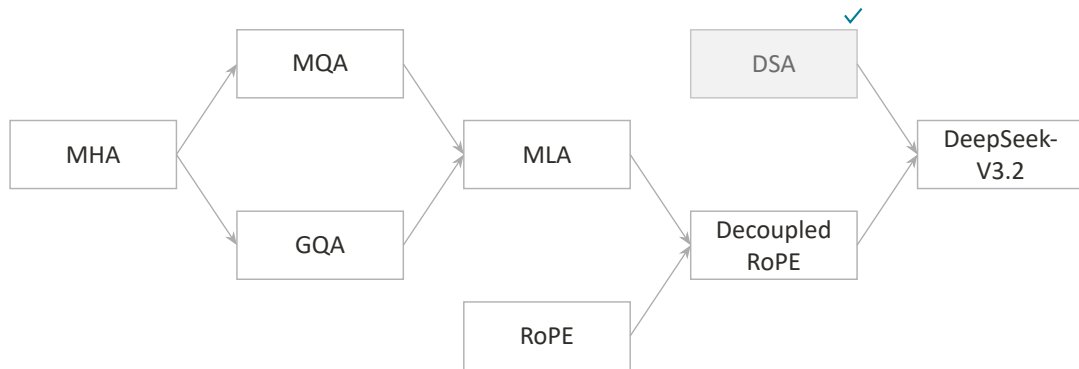
$$M_{\text{KV}} = B \cdot T \cdot L \cdot 2d \cdot b$$

Factor	Shrink it by	When
T : positions	fewer entries	Lecture 5
$2d$: one entry	smaller entries	today
L : layers	share across layers	Lecture 7
b : bytes per number	fewer bits per number	later

How many numbers do we need to store for each entry?



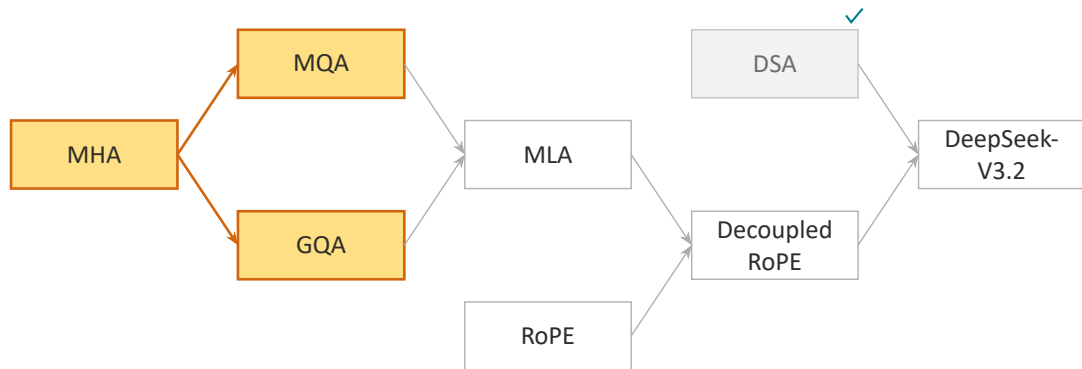
Compact cache entries



We first share keys and values across attention heads.



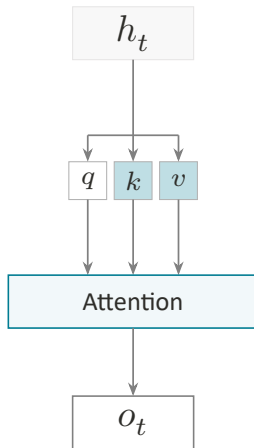
Compact cache entries



We first share keys and values across attention heads.



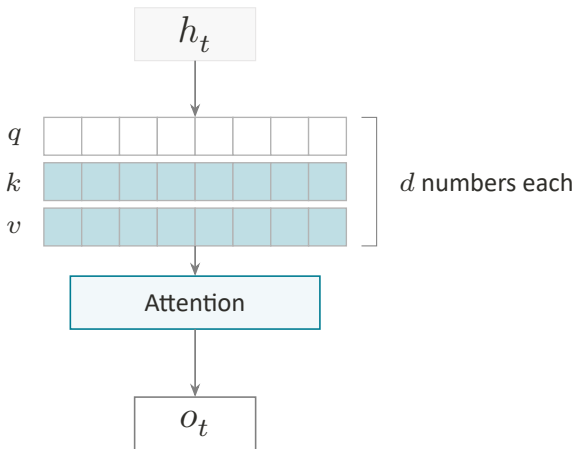
Splitting attention into heads



So far, each token has one query, key, and value.



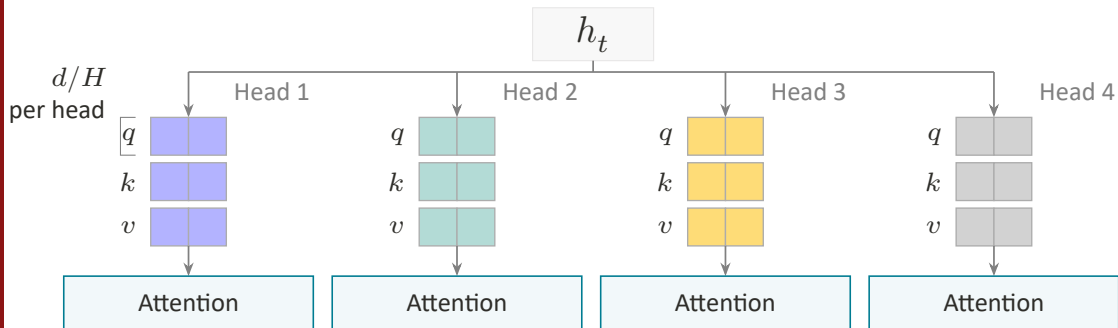
Splitting attention into heads



Project h_t to q, k, v , each with d numbers.



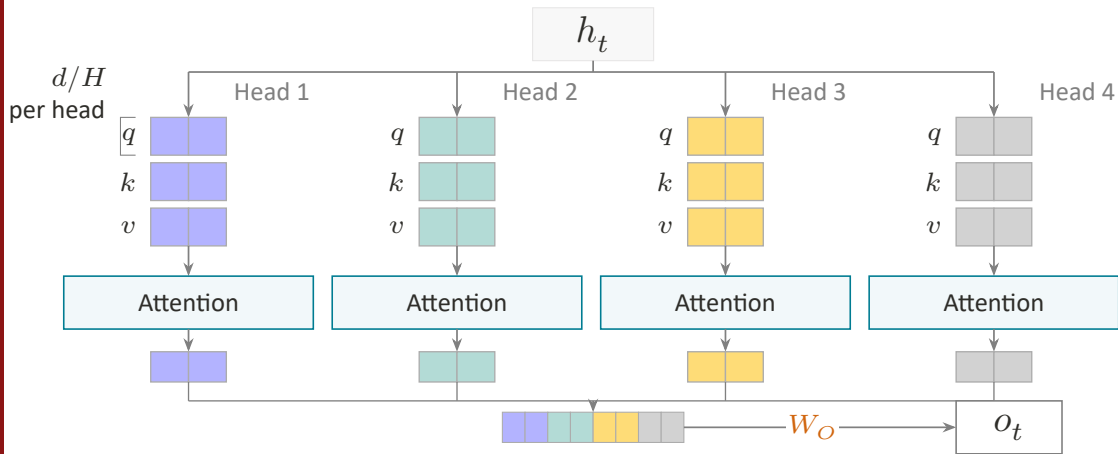
Splitting attention into heads



Split each into H slices, with head i using slice i .



Splitting attention into heads

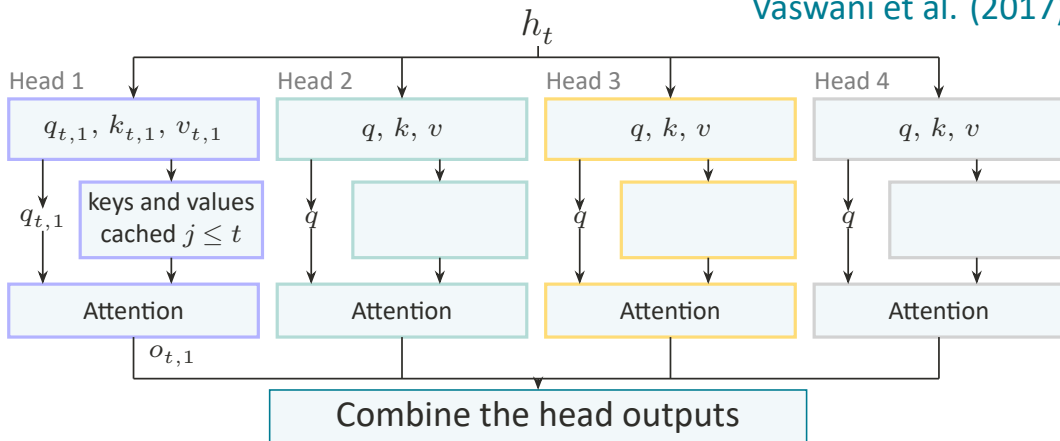


Concatenate the head outputs and apply W_O .



Multi-Head Attention (MHA)

Vaswani et al. (2017)



Each head uses its own queries, keys, and values.



Multi-Head Attention (MHA)

Each head $i = 1, \dots, H$ has its own query, key, and value maps:

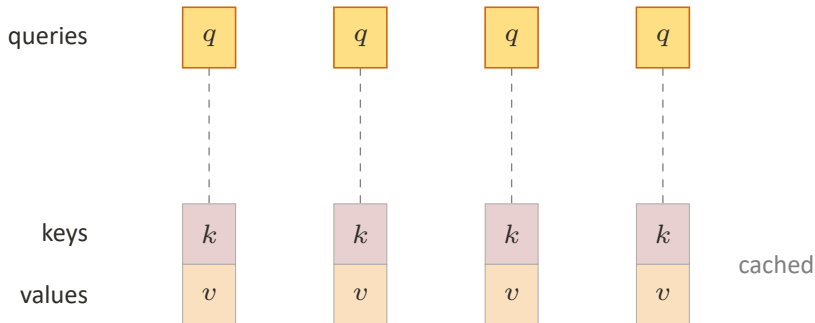
$$q_{t,i} = W_{Q,i}^T h_t, \quad k_{j,i} = W_{K,i}^T h_j, \quad v_{j,i} = W_{V,i}^T h_j.$$



Multi-Head Attention (MHA)

Each head $i = 1, \dots, H$ has its own query, key, and value maps:

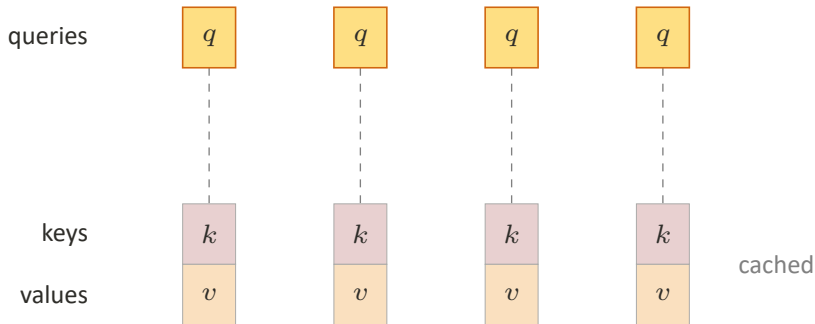
$$q_{t,i} = W_{Q,i}^T h_t, \quad k_{j,i} = W_{K,i}^T h_j, \quad v_{j,i} = W_{V,i}^T h_j.$$



Multi-Head Attention (MHA)

Each head $i = 1, \dots, H$ has its own query, key, and value maps:

$$q_{t,i} = W_{Q,i}^T h_t, \quad k_{j,i} = W_{K,i}^T h_j, \quad v_{j,i} = W_{V,i}^T h_j.$$



All heads' keys and values: $H(d_k + d_v) = 2d$ numbers.



Multi-Query Attention (MQA)

Shazeer (2019)

The H query heads share one key map and one value map:

$$k_j = W_K^T h_j, \quad v_j = W_V^T h_j \quad \text{for every head.}$$

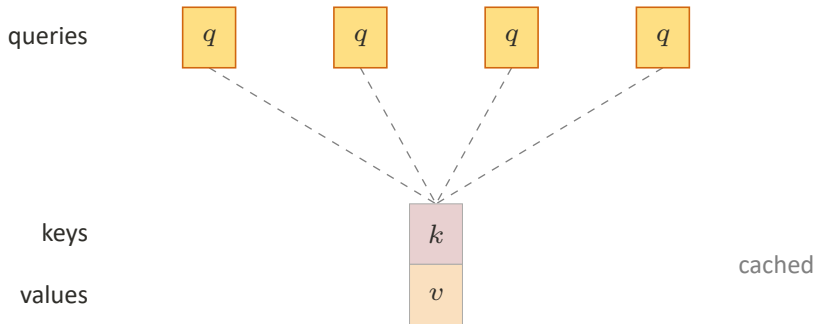


Multi-Query Attention (MQA)

Shazeer (2019)

The H query heads share one key map and one value map:

$$k_j = W_K^T h_j, \quad v_j = W_V^T h_j \quad \text{for every head.}$$

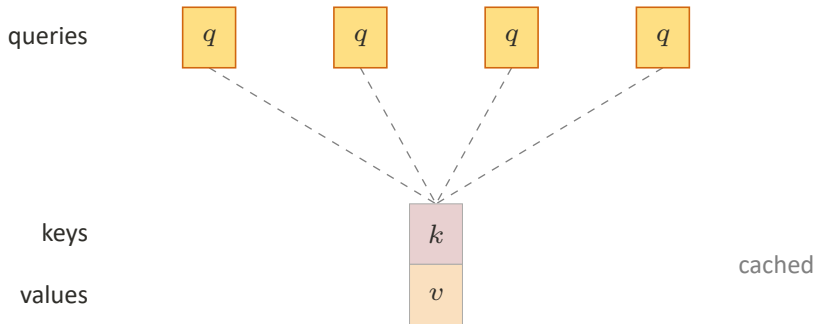


Multi-Query Attention (MQA)

Shazeer (2019)

The H query heads share one key map and one value map:

$$k_j = W_K^T h_j, \quad v_j = W_V^T h_j \quad \text{for every head.}$$



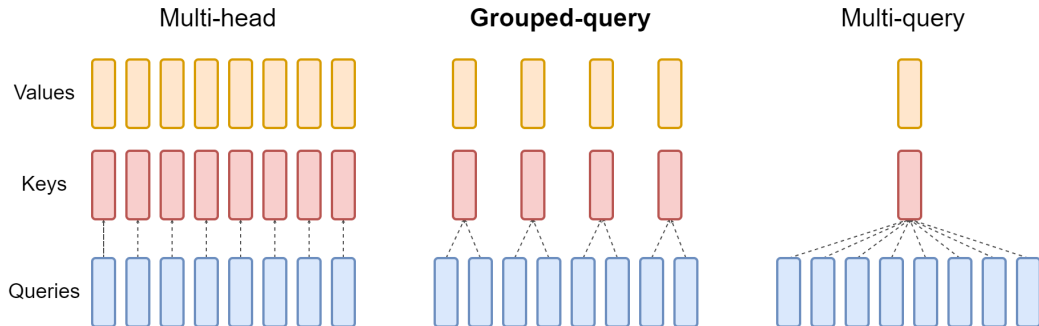
Shared keys and values reduce the KV cache by a factor of H .



Grouped-Query Attention (GQA)

Ainslie et al. (2023)

Each group of query heads shares one key and one value per position.



Source: Ainslie et al. (2023), Fig. 2.

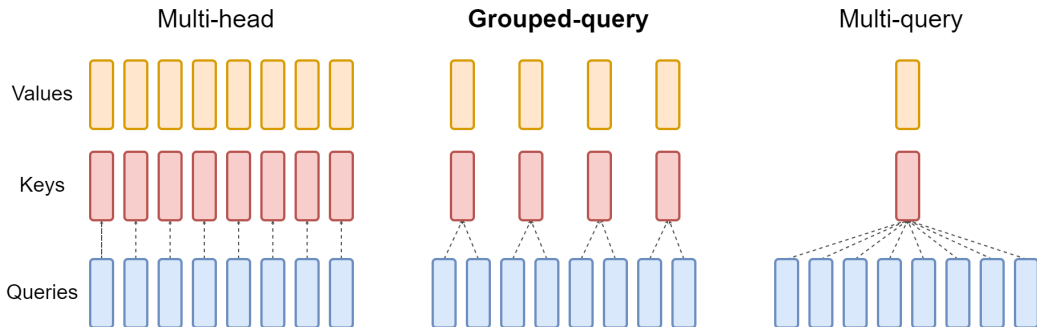
MS&E 319 Lecture 6



Grouped-Query Attention (GQA)

Ainslie et al. (2023)

Each group of query heads shares one key and one value per position.



With one group, GQA is MQA; with H groups, it is MHA.

Source: Ainslie et al. (2023), Fig. 2.

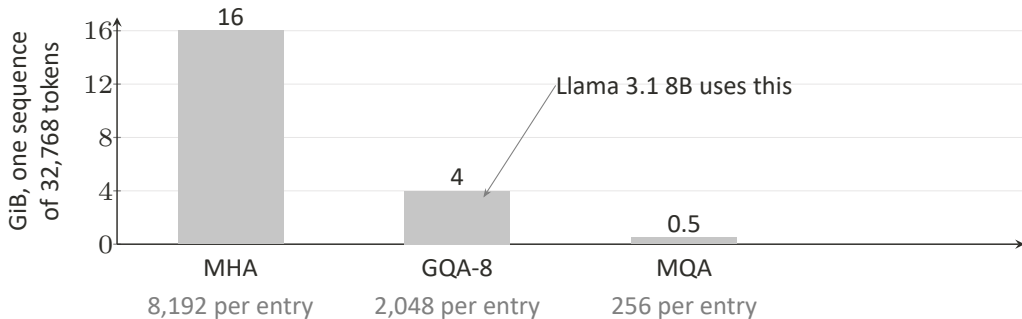
MS&E 319 Lecture 6



KV Sharing: MHA, GQA, and MQA

Example: $L = 32$; $H = 32$ heads of width 128; $T = 32,768$; 2 bytes.

Only the number of key-value heads, H_{KV} , changes:



KV Sharing: MHA, GQA, and MQA

Per decode step and layer; divide each count by $T(d_k + d_v)$:

	MHA	GQA	MQA
Numbers stored	H	H_{KV}	1
Numbers read	H	H_{KV}	1
Multiply-adds	H	H	H



KV Sharing: MHA, GQA, and MQA

Per decode step and layer; divide each count by $T(d_k + d_v)$:

	MHA	GQA	MQA
Numbers stored	H	H_{KV}	1
Numbers read	H	H_{KV}	1
Multiply-adds	H	H	H

KV sharing reduces storage and reads, with the same arithmetic.

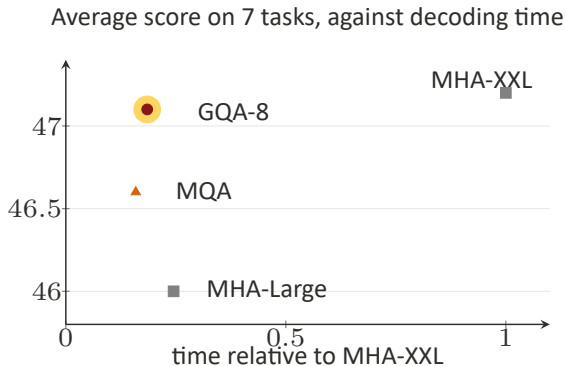


Converting T5 to MQA and GQA

Start: T5-XXL (MHA).

Convert to MQA or GQA-8:
average the KV heads.

Continue pretraining
for 5% of the original steps.



Source: [Ainslie et al. \(2023\)](#), Table 1 and Fig. 3; times relative to MHA-XXL.

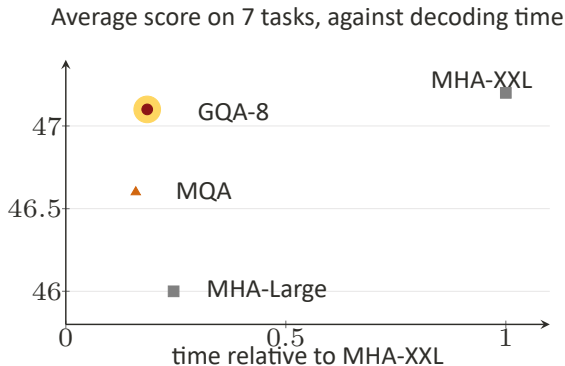


Converting T5 to MQA and GQA

Start: T5-XXL (MHA).

Convert to MQA or GQA-8:
average the KV heads.

Continue pretraining
for 5% of the original steps.



GQA-8 scores higher and decodes faster than the smaller MHA model.

Scores after conversion, continued pretraining, then fine-tuning on each task.

Source: [Ainslie et al. \(2023\), Table 1 and Fig. 3](#); times relative to MHA-XXL.



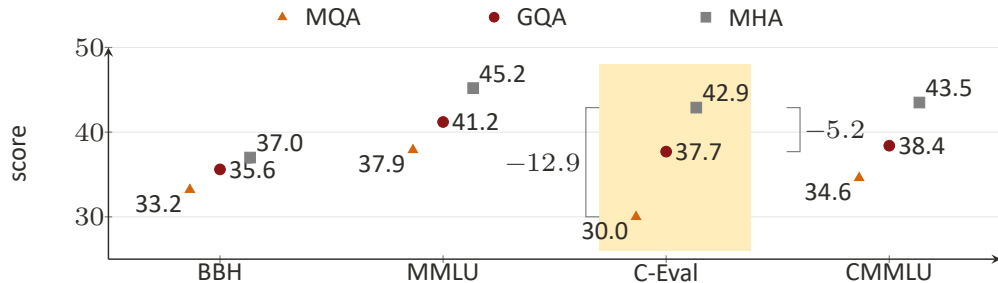
How KV sharing affects quality

Heads share keys and values within a group, with separate queries.



How KV sharing affects quality

Heads share keys and values within a group, with separate queries.

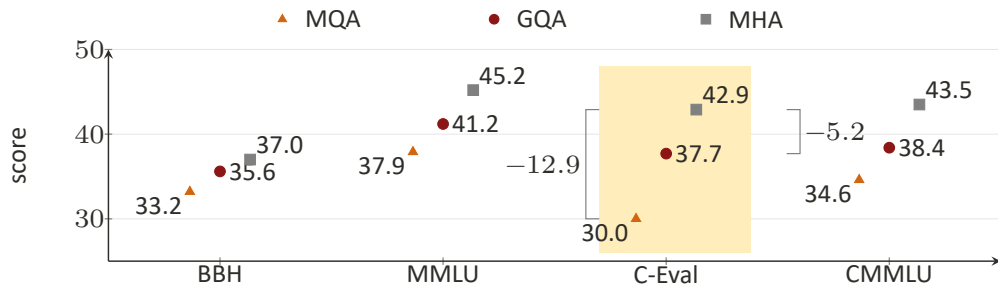


DeepSeek-AI (2024): 7B dense models from scratch; depth adjusted to match parameters.



How KV sharing affects quality

Heads share keys and values within a group, with separate queries.



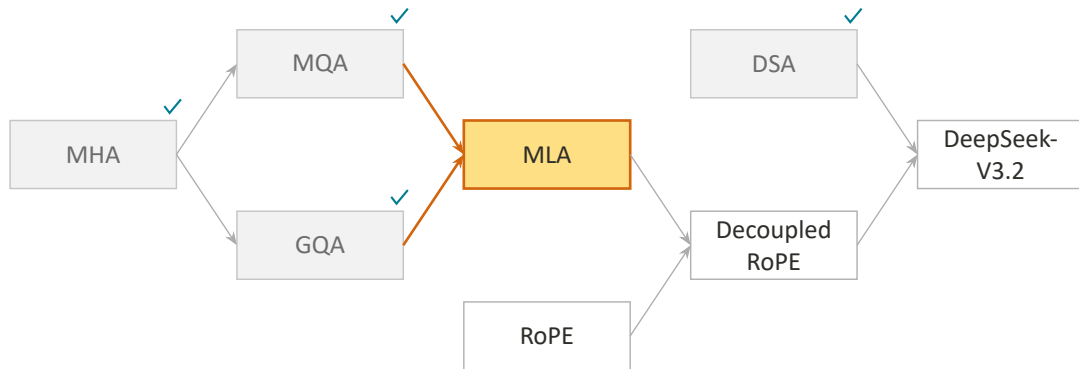
DeepSeek-AI (2024): 7B dense models from scratch; depth adjusted to match parameters.

MQA trails MHA by 3.8 points on BBH, but by 12.9 on C-Eval.

Sharing saves memory; the quality cost depends on task and setup.



Compact cache entries

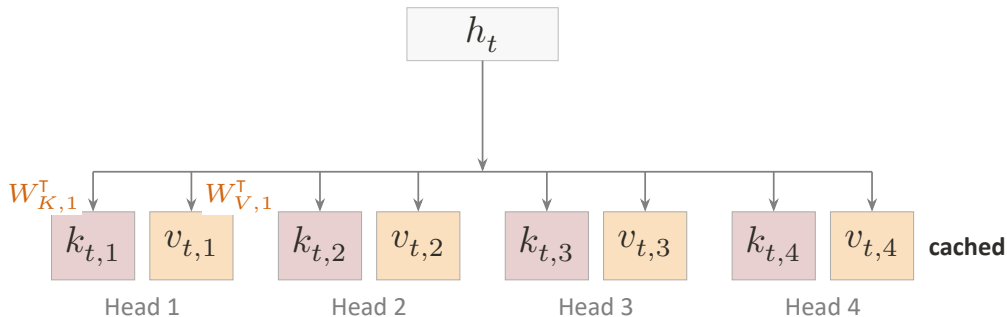


Can we recover every head's keys and values from one short vector?



Recap: Multi-Head Attention (MHA)

In MHA, each head computes its own key and value from h_t .



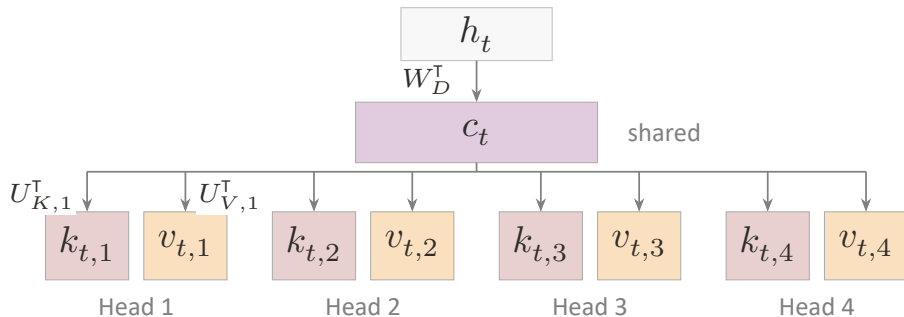
All H keys and values are cached: $H(d_k + d_v)$ numbers per token.



Multi-Head Latent Attention (MLA)

DeepSeek-AI (2024)

MLA learns a short vector for each token, shared across heads.



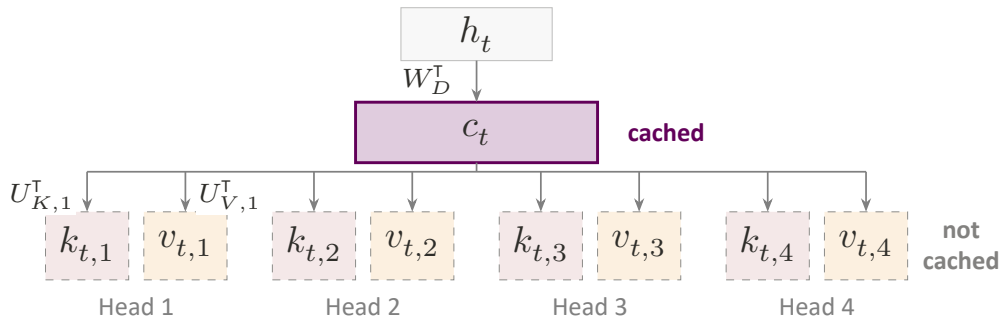
Each head maps the shared latent to its own key and value.



MLA: why caching c_t is enough

DeepSeek-AI (2024)

The latent c_t determines each head's key and value.



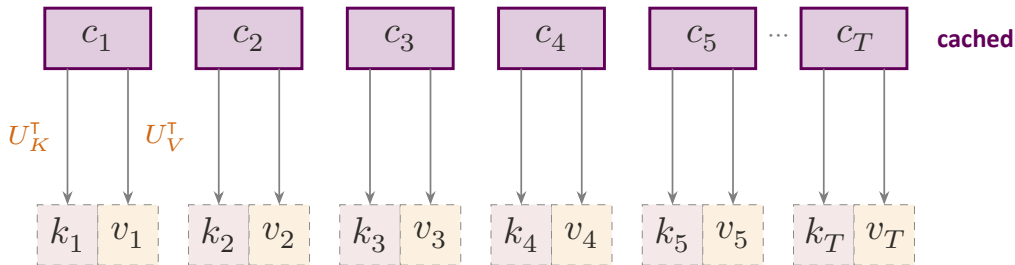
$$k_{t,i} = U_{K,i}^T c_t, \quad v_{t,i} = U_{V,i}^T c_t.$$

Numbers cached per token and layer: MHA, $H(d_k + d_v)$; MLA, d_c .



MLA: reconstructing keys at every step

At each step, we could reconstruct every k_j and v_j from c_j .



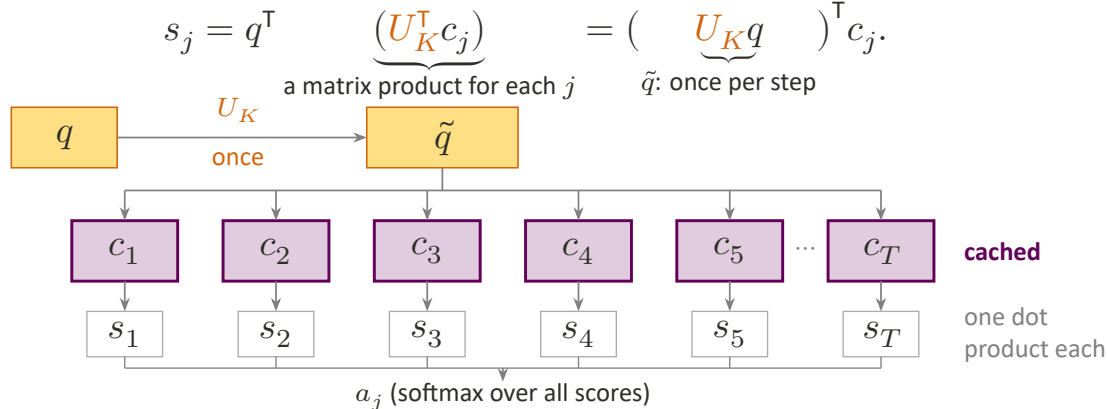
Per head and step: $Td_c(d_k + d_v)$ multiply-adds, or $O(Td^2)$.

Can we avoid reconstructing all these vectors?



MLA: transform the query once

DeepSeek-AI (2024)



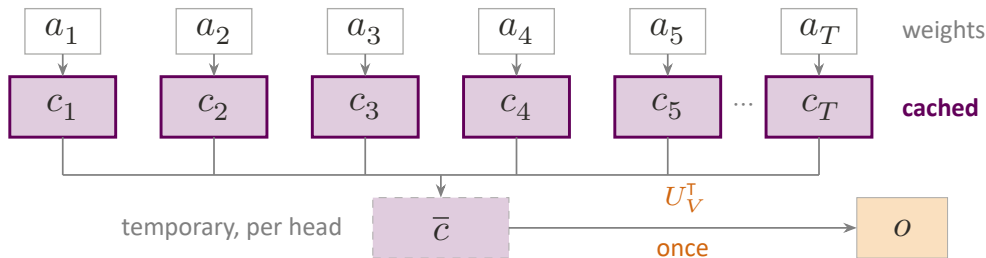
After computing $\tilde{q}$ once, we need one dot product with each c_j .

We can also combine U_K with W_Q ahead of time.



MLA: average before expanding

$$o = \sum_j a_j \underbrace{(U_V^T c_j)}_{\text{a matrix product for each } j} = \underbrace{U_V^T}_{\text{once}} \underbrace{\left(\sum_j a_j c_j \right)}_{\bar{c}}.$$



Average the c_j before applying one matrix product per head.

We can also combine U_V with the output map W_O ahead of time.



MLA: cost per decode step

Cost for one head at one decode step, with T cached positions:

	Natural approach	Transform once, expand once
Matrix–vector products	$2T$	2
Multiply-adds	$Td_c(d_k + d_v) + T(d_k + d_v)$	$d_c d_k + 2Td_c + d_c d_v$
Widths $\approx d$	$O(Td^2)$	$O(d^2 + Td)$

Scores, weights, and outputs are unchanged; the Td^2 cost is gone.

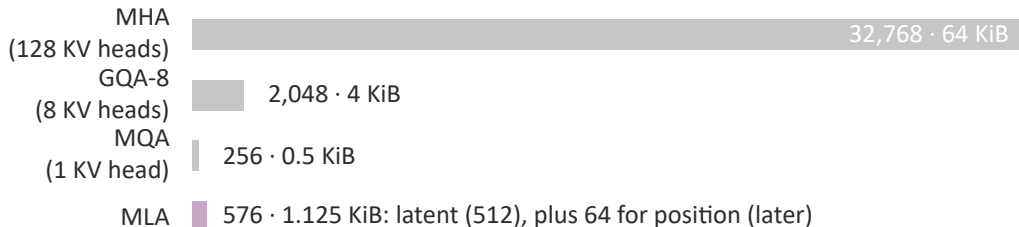
Example versus MHA: $16\times$ fewer numbers read, $4\times$ more multiply-adds.



MLA: cache savings in DeepSeek-V3

DeepSeek-V3: 128 heads; key/value content width 128; $d_c = 512$.

Numbers cached per entry, per layer (KiB at 2 bytes each)



MLA: $3.6\times$ smaller than GQA-8, and $57\times$ smaller than MHA.

Source: Zhao et al. (2025), Table 1; values recomputed from configurations.



MLA performance

DeepSeek-AI (2024)

DeepSeek-V2 MoE ($\sim 250\text{B}$, 420B tokens), with MHA or MLA.

	MHA	MLA
Total parameters	250.8B	247.4B
Cached numbers per token, all layers	860.2K	34.6K
BBH	46.6	50.7
MMLU	57.5	59.0
C-Eval	57.9	59.2
CMMLU	60.7	62.5

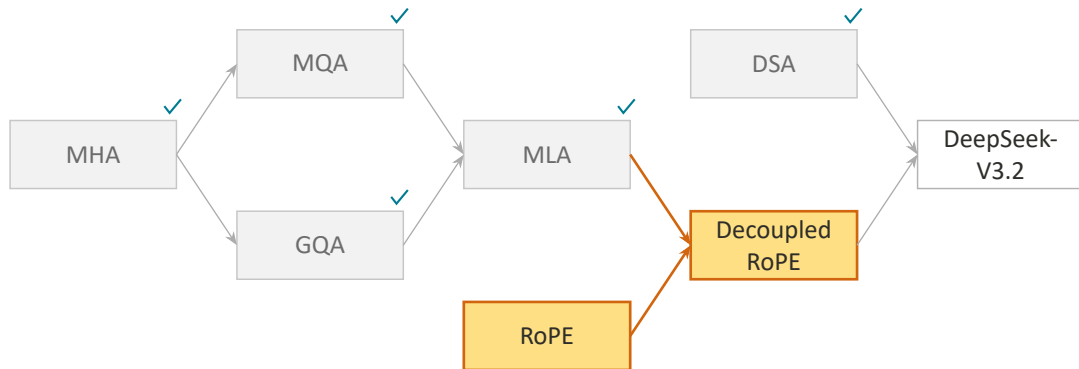
MLA improves all four scores while using 4% of MHA's cache.

Source: [DeepSeek-AI \(2024\)](#), App. D, Tables 8–9.

MS&E 319 Lecture 6



Compact cache entries

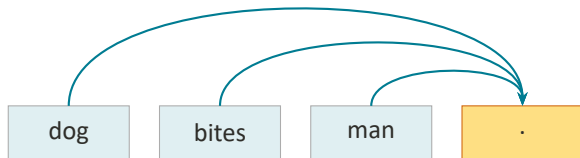


How do we include position while retaining MLA's computation?

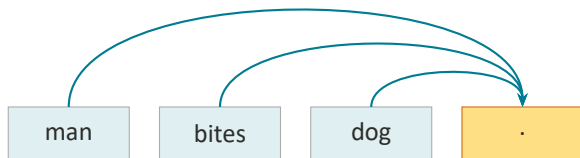


Why attention needs position

Without position, attention cannot distinguish these word orders.



same keys, values, and scores:
the same output for "." in layer 1



We want the score to use position, ideally through the offset $t - j$.



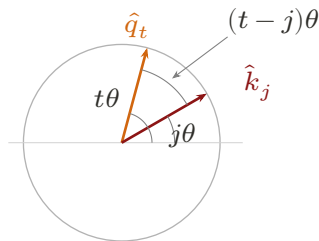
Rotary Position Embedding (RoPE)

Su et al. (2021)

RoPE rotates each query and key by an angle set by its position.

$\hat{q}_t = R_t q_t$: rotate the query by $t\theta$

$\hat{k}_j = R_j k_j$: rotate the key by $j\theta$



In d dimensions, each coordinate pair rotates at its own rate.

drawn with q_t, k_j initially aligned



Rotary Position Embedding (RoPE)

Su et al. (2021)

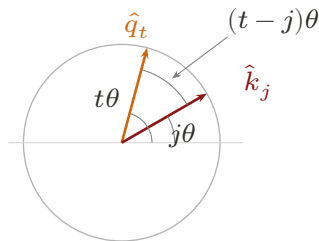
RoPE rotates each query and key by an angle set by its position.

$\hat{q}_t = R_t q_t$: rotate the query by $t\theta$

$\hat{k}_j = R_j k_j$: rotate the key by $j\theta$

$$\hat{q}_t^\top \hat{k}_j = q_t^\top R_t^\top R_j k_j = q_t^\top \underbrace{R_{j-t}}_{\text{only the offset}} k_j$$

In d dimensions, each coordinate pair rotates at its own rate.



drawn with q_t, k_j initially aligned

The score depends on the two positions only through $t - j$.



Why RoPE prevents query reuse

MLA uses the same transformed query to score every cached latent.

$$q^T (U_K^T c_j) = \left(\underbrace{U_K q}_{\tilde{q}, \text{ same for every } j} \right)^T c_j.$$

With RoPE, the rebuilt key is rotated by its own R_j :

$$\hat{q}_t^T R_j U_K^T c_j = \left(\underbrace{U_K R_j^T \hat{q}_t}_{\text{changes with } j} \right)^T c_j.$$

The query depends on j , preventing reuse across positions.

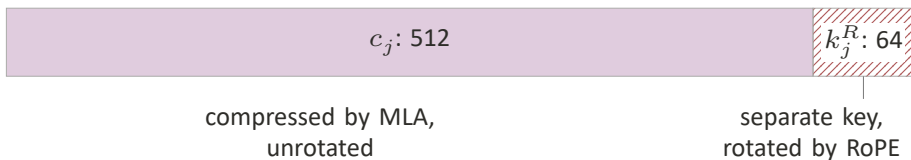


Decoupled RoPE

DeepSeek-AI (2024)

Store a small positional key alongside the content latent.

one cached entry, shared by all heads



$$\text{Score} = \text{content score} + \text{positional score}.$$

MLA keeps its content shortcut and adds a small positional key.



Adding content and position

For head h and position j , add the content and positional terms.

$$s_{t,h,j} = \underbrace{\tilde{q}_{t,h}^\top c_j}_{\text{content: 512 dimensions}} + \underbrace{(q_{t,h}^R)^\top k_j^R}_{\text{position: 64 dimensions}}.$$

$$a_{t,h,j} = \text{softmax}_j(s_{t,h,j}/\tau) \quad z_{t,h} = \sum_j a_{t,h,j} c_j.$$

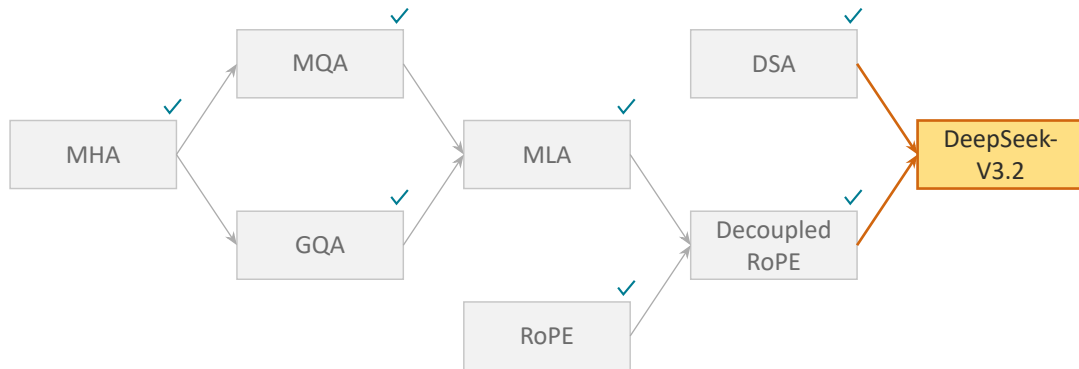
τ : attention scaling factor; softmax runs over allowed cached positions j .

RoPE affects the weights; the weighted sum has 512 dimensions.

Each head applies one softmax to the combined scores.



Compact cache entries



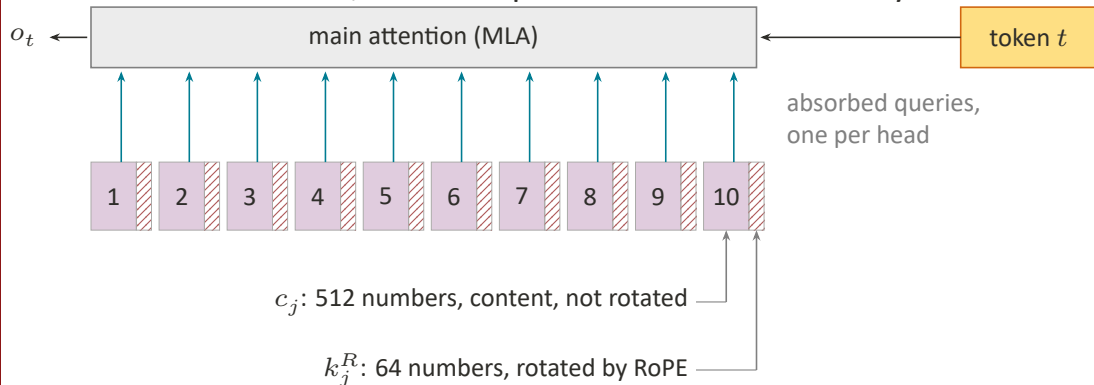
Can Lecture 5's indexer select which latents attention reads?



DeepSeek-V3.2: MLA

DeepSeek-AI (2025)

V3.2 uses V3's MLA, with two parts in each cached entry.



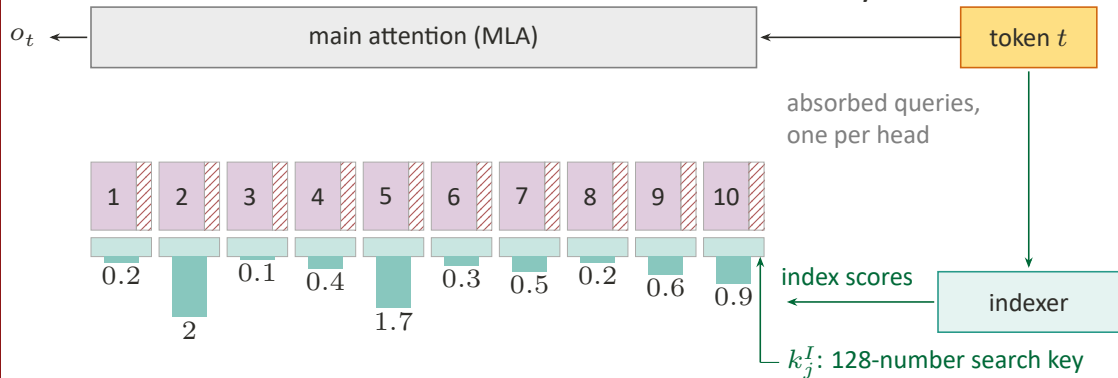
Main attention in MLA reads all 576 numbers of every entry.



DeepSeek-V3.2: MLA plus DSA

DeepSeek-AI (2025)

The indexer from Lecture 5 selects the entries used by main attention.



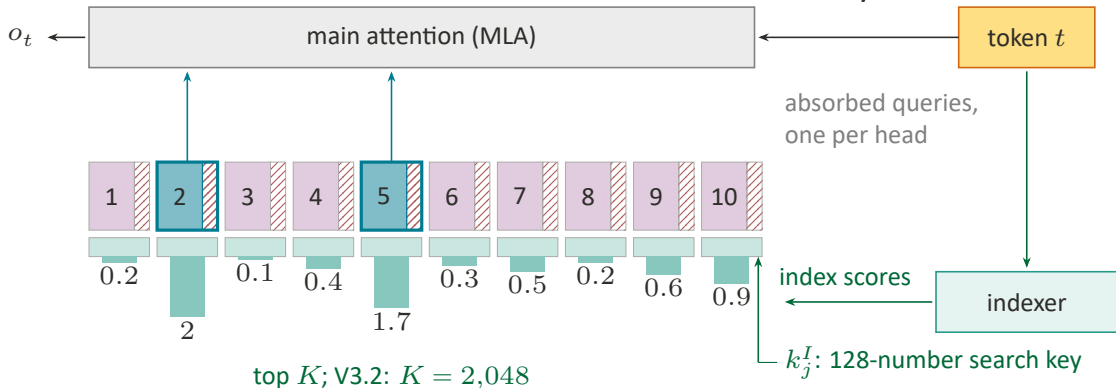
The indexer uses the search keys to score every position.



DeepSeek-V3.2: MLA plus DSA

DeepSeek-AI (2025)

The indexer from Lecture 5 selects the entries used by main attention.



Main attention then reads the K selected entries.



DSA: the cost of scanning small keys

$$I_{tj} = \sum_{h=1}^{H_I} w_{t,h} \text{ReLU}(q_{t,h}^I \cdot k_j^I)$$

64 small heads (width 128), in FP8

one small search key
per position



DSA: the cost of scanning small keys

$$I_{tj} = \sum_{h=1}^{H_I} w_{t,h} \text{ReLU}(q_{t,h}^I \cdot k_j^I)$$

64 small heads (width 128), in FP8

Per cached position, one new token, one layer

one small search key
per position

	Main attention (MLA)	Indexer
Numbers read	576	128, in FP8
Multiply-adds	$128 \times (576 + 512) \approx 139,000$	$64 \times 128 \approx 8,200$

The indexer uses about $17\times$ fewer multiply-adds per position.



DeepSeek V3.2: cache and reads

one cached entry per position and layer: 704 numbers



c_j : content, not rotated

k_j^R : rotated by RoPE

k_j^I : search key

	Who	Per new token
Store	every position	704 numbers
Select	indexer, all T positions	128-number search keys
Attend	main attention	$\leq 2,048$ entries of 576

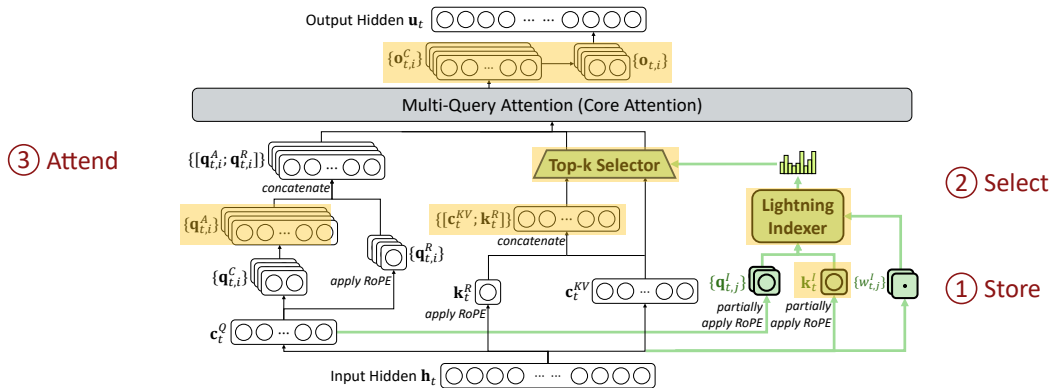
MLA reduces entry size; DSA reduces the number of entries read.



DeepSeek V3.2: summary

DeepSeek-AI (2025)

Cache, indexer, and main attention in the full model.



Source: DeepSeek-AI (2025), Fig. 2.

MS&E 319 Lecture 6



Reducing the KV Cache

Goal: Store fewer numbers for each token.



Reducing the KV Cache

Goal: Store fewer numbers for each token.

MQA / GQA: Heads share keys and values, with separate queries.

MLA: Cache a shared latent. Absorb projections into the query and output to avoid reconstructing keys and values.

Decoupled RoPE: Unrotated content, small rotated positional key.

DeepSeek V3.2: MLA + DSA; select at most 2,048 entries.

DeepSeek V4: Sparse and compressed attention; compressed entries serve as both keys and values.



Reducing the KV Cache

Goal: Store fewer numbers for each token.

MQA / GQA: Heads share keys and values, with separate queries.

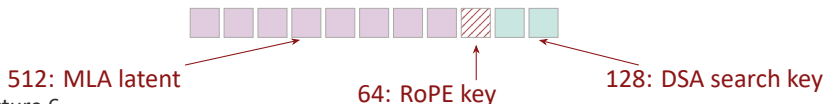
MLA: Cache a shared latent. Absorb projections into the query and output to avoid reconstructing keys and values.

Decoupled RoPE: Unrotated content, small rotated positional key.

DeepSeek V3.2: MLA + DSA; select at most 2,048 entries.

DeepSeek V4: Sparse and compressed attention; compressed entries serve as both keys and values.

V3.2, per position and layer: 576 numbers read by all 128 heads



Questions?

