Multi-Head Latent Attention

Deep Dive into DeepSeek — Chapter 4

Yin Yang

Foundation Books Project

The one rule of this chapter

MLA changes *what is stored* per earlier position.
It does not change the causal obligation.

- Autoregressive decode reads attention state for every earlier position, again, at every step.
- So the per-position payload is not a modeling detail — it is the byte volume the accelerator re-reads T times.
- MLA keeps a low-rank latent state plus a small RoPE-sensitive component, and reconstructs what attention needs.

The one rule of this chapter

MLA changes *what is stored* per earlier position.
It does not change the causal obligation.

- Autoregressive decode reads attention state for every earlier position, again, at every step.
- So the per-position payload is not a modeling detail — it is the byte volume the accelerator re-reads T times.
- MLA keeps a low-rank latent state plus a small RoPE-sensitive component, and reconstructs what attention needs.

Goal: follow one decode step through MHA, GQA, and MLA — cache shape, projection paths, the NoPE/RoPE split, and the decode-time representation — and see where each piece lives in the DeepSeek source.

Roadmap

Cache shape as a systems object

Baselines: MHA, MQA, GQA

The low-rank latent KV state

Projection paths and the NoPE/RoPE split

Decode-time representation

Cost, accuracy, and what comes next

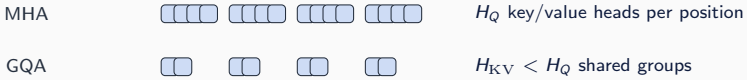
Cache shape as a systems object

The same causal reach, three payloads

MHA  H_Q key/value heads per position

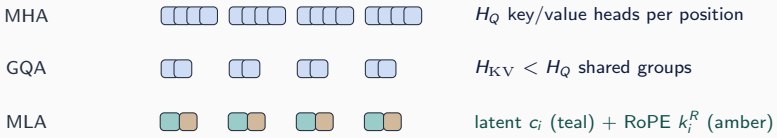
Four causal positions, held fixed. Only the stored representation changes.

The same causal reach, three payloads



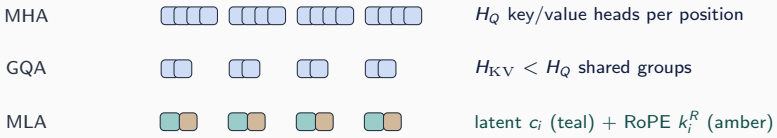
Four causal positions, held fixed. Only the stored representation changes.

The same causal reach, three payloads



Four causal positions, held fixed. Only the stored representation changes.

The same causal reach, three payloads



Four causal positions, held fixed. Only the stored representation changes.

Cache read volume (Def.): stored elements read from previous positions $\times$ storage byte width — one layer, one decode step.

The running toy layer, carried all chapter

Quantity	Value	Role
T	4	prompt positions already cached
H_Q	4	query heads
$d_k = d_v$	2	key and value widths per head
r	—	low-rank latent dimension (MLA)

The running toy layer, carried all chapter

Quantity	Value	Role
T	4	prompt positions already cached
H_Q	4	query heads
$d_k = d_v$	2	key and value widths per head
r	—	low-rank latent dimension (MLA)

- Dense MHA cache read: $TH_Q(d_k + d_v) = 4 \cdot 4 \cdot 4 = 64$ scalar elements, or 128 bytes in BF16.

The running toy layer, carried all chapter

Quantity	Value	Role
T	4	prompt positions already cached
H_Q	4	query heads
$d_k = d_v$	2	key and value widths per head
r	—	low-rank latent dimension (MLA)

- Dense MHA cache read: $TH_Q(d_k + d_v) = 4 \cdot 4 \cdot 4 = 64$ scalar elements, or 128 bytes in BF16.
- Decoding x_5 may read cached state for $x_1, \ldots, x_4$, then appends state for x_5. Every variant obeys that.

The running toy layer, carried all chapter

Quantity	Value	Role
T	4	prompt positions already cached
H_Q	4	query heads
$d_k = d_v$	2	key and value widths per head
r	—	low-rank latent dimension (MLA)

- Dense MHA cache read: $TH_Q(d_k + d_v) = 4 \cdot 4 \cdot 4 = 64$ scalar elements, or 128 bytes in BF16.
- Decoding x_5 may read cached state for $x_1, \ldots, x_4$, then appends state for x_5. Every variant obeys that.

The chapter's question, made precise: can the model retain less state per previous position while still reconstructing the attention quantities the next causal step requires?

Naming discipline: non-RoPE vs. NoPE

- RoPE rotates only *part* of each query and key.
- This book says **non-RoPE** for the unrotated coordinates in explanatory prose.
- It reserves **NoPE** for the DeepSeek source label — `qk_nope_head_dim`, `q_nope`.
- Elsewhere in the literature, "NoPE" can name a whole decoder with *no* positional encoding. That model-wide meaning is not the meaning here.

Naming discipline: non-RoPE vs. NoPE

- RoPE rotates only *part* of each query and key.
- This book says **non-RoPE** for the unrotated coordinates in explanatory prose.
- It reserves **NoPE** for the DeepSeek source label — `qk_nope_head_dim`, `q_nope`.
- Elsewhere in the literature, "NoPE" can name a whole decoder with *no* positional encoding. That model-wide meaning is not the meaning here.

The non-RoPE branch may still receive position-aware hidden states from the surrounding model. The RoPE branch is explicitly rotated, so it stays tied to the positions being scored.

Baselines: MHA, MQA, GQA

Baselines priced by per-position payload

Variant	Stored per position	Cache pressure
MHA	per-head K, V for all H_Q heads	highest

Baselines priced by per-position payload

Variant	Stored per position	Cache pressure
MHA	per-head K, V for all H_Q heads	highest
MQA	one shared K, V head	lowest of the three

Baselines priced by per-position payload

Variant	Stored per position	Cache pressure
MHA	per-head K, V for all H_Q heads	highest
MQA	one shared K, V head	lowest of the three
GQA	H_{KV} groups, $1 < H_{\mathrm{KV}} < H_Q$	tunable middle

Baselines priced by per-position payload

Variant	Stored per position	Cache pressure
MHA	per-head K, V for all H_Q heads	highest
MQA	one shared K, V head	lowest of the three
GQA	H_{KV} groups, $1 < H_{\mathrm{KV}} < H_Q$	tunable middle
MLA	latent c_i plus RoPE k_i^R	set by rank r, not head count

- MQA and GQA reduce the payload by *sharing heads*: the knob is H_{KV}.
- MLA reduces it by *changing the object*: the knob is a rank.

Baselines priced by per-position payload

Variant	Stored per position	Cache pressure
MHA	per-head K, V for all H_Q heads	highest
MQA	one shared K, V head	lowest of the three
GQA	H_{KV} groups, $1 < H_{\mathrm{KV}} < H_Q$	tunable middle
MLA	latent c_i plus RoPE k_i^R	set by rank r, not head count

- MQA and GQA reduce the payload by *sharing heads*: the knob is H_{KV}.
- MLA reduces it by *changing the object*: the knob is a rank.

That difference is why MLA's savings do not have to be bought with fewer distinct attention heads.

The low-rank latent KV state

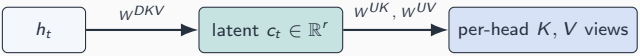
One compact state, projected back up



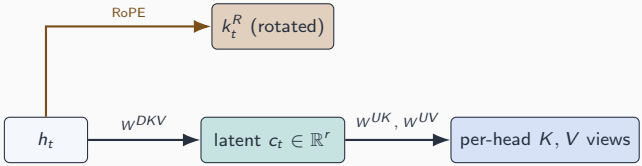
One compact state, projected back up



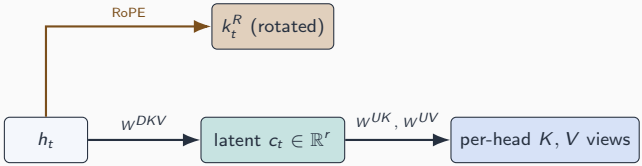
One compact state, projected back up



One compact state, projected back up

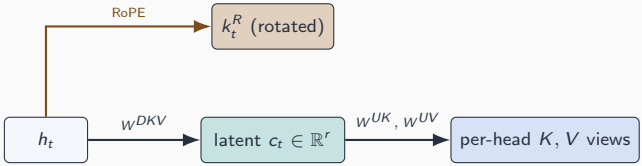


One compact state, projected back up



cached: c_t and k_t^R *not cached*: the reconstructed per-head views

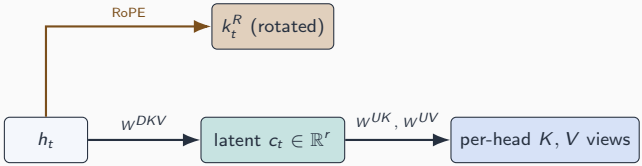
One compact state, projected back up



cached: c_t and k_t^R *not cached*: the reconstructed per-head views

- The down-projection W^{DKV} writes the only KV object that persists.
- Up-projections reconstruct what the score and value paths need.
- The RoPE branch is kept separately, because rotation does not commute with an arbitrary up-projection.

One compact state, projected back up



cached: c_t and k_t^R *not cached*: the reconstructed per-head views

- The down-projection W^{DKV} writes the only KV object that persists.
- Up-projections reconstruct what the score and value paths need.
- The RoPE branch is kept separately, because rotation does not commute with an arbitrary up-projection.

Storage is $O(r + d_{\text{rope}})$ per position, independent of H_Q.

Projection paths and the NoPE/RoPE split

Two branches, one score

$$\text{score}(t, i) \;=\; \underbrace{\langle q_t^C,\; k_i^C \rangle}_{\text{non-RoPE, absorbable}}$$

Two branches, one score

$$\text{score}(t,i) \;=\; \underbrace{\langle q_t^C,\, k_i^C \rangle}_{\text{non-RoPE, absorbable}} \;+\; \underbrace{\langle q_t^R,\, k_i^R \rangle}_{\text{RoPE, position-tied}}$$

Two branches, one score

$$\text{score}(t,i) \;=\; \underbrace{\langle q_t^C,\, k_i^C \rangle}_{\text{non-RoPE, absorbable}} \;+\; \underbrace{\langle q_t^R,\, k_i^R \rangle}_{\text{RoPE, position-tied}}$$

- The non-RoPE term is bilinear in the latent, so W^{UK} can be *absorbed* into the query side — the per-head K view need never be materialized at decode.

Two branches, one score

$$\text{score}(t,i) \;=\; \underbrace{\langle q_t^C,\, k_i^C \rangle}_{\text{non-RoPE, absorbable}} \;+\; \underbrace{\langle q_t^R,\, k_i^R \rangle}_{\text{RoPE, position-tied}}$$

- The non-RoPE term is bilinear in the latent, so W^{UK} can be *absorbed* into the query side — the per-head K view need never be materialized at decode.
- The RoPE term cannot be absorbed the same way: rotation depends on position, so k_i^R is stored explicitly.

Two branches, one score

$$\text{score}(t,i) \;=\; \underbrace{\langle q_t^C,\, k_i^C \rangle}_{\text{non-RoPE, absorbable}} \;+\; \underbrace{\langle q_t^R,\, k_i^R \rangle}_{\text{RoPE, position-tied}}$$

- The non-RoPE term is bilinear in the latent, so W^{UK} can be *absorbed* into the query side — the per-head K view need never be materialized at decode.
- The RoPE term cannot be absorbed the same way: rotation depends on position, so k_i^R is stored explicitly.

This split is the entire reason MLA has two cached tensors instead of one.

Decode-time representation

From concept to code: two caches, not one

The reference implementation registers exactly the two objects the split predicts — and the naive path registers the full views instead.

from `DeepSeek-V3/inference/model.py`

```
if attn_impl == "naive":
    self.register_buffer("k_cache", torch.zeros(..., self.qk_head_dim), ...)
    self.register_buffer("v_cache", torch.zeros(..., self.v_head_dim), ...)
else:
    self.register_buffer("kv_cache", torch.zeros(..., self.kv_lora_rank), ...)
    self.register_buffer("pe_cache", torch.zeros(..., self.qk_rope_head_dim), ...)
```

From concept to code: two caches, not one

The reference implementation registers exactly the two objects the split predicts — and the naive path registers the full views instead.

from `DeepSeek-V3/inference/model.py`

```
if attn_impl == "naive":
    self.register_buffer("k_cache", torch.zeros(..., self.qk_head_dim), ...)
    self.register_buffer("v_cache", torch.zeros(..., self.v_head_dim), ...)
else:
    self.register_buffer("kv_cache", torch.zeros(..., self.kv_lora_rank), ...)
    self.register_buffer("pe_cache", torch.zeros(..., self.qk_rope_head_dim), ...)
```

`kv_cache` is c_i at width `kv_lora_rank` $= r$; `pe_cache` is k_i^R at width `qk_rope_head_dim`. The `naive` branch is the materialized baseline, kept for reference. [INSPECTED: DeepSeek-V3 `inference/model.py`]

Cost, accuracy, and what comes next

What MLA does and does not buy

Buys

- Per-position payload set by a rank, not a head count.
- Lower repeatedly-read byte volume during decode.
- Head count stays free to serve model quality.

What MLA does and does not buy

Buys

- Per-position payload set by a rank, not a head count.
- Lower repeatedly-read byte volume during decode.
- Head count stays free to serve model quality.

Does not buy

- Fewer positions to attend to — that is *sparse* attention, a different axis.
- Automatic runtime support: kernels, paging, and layout are a Volume II problem.
- A free lunch in accuracy; the rank is a modeling choice.

What MLA does and does not buy

Buys

- Per-position payload set by a rank, not a head count.
- Lower repeatedly-read byte volume during decode.
- Head count stays free to serve model quality.

Does not buy

- Fewer positions to attend to — that is *sparse* attention, a different axis.
- Automatic runtime support: kernels, paging, and layout are a Volume II problem.
- A free lunch in accuracy; the rank is a modeling choice.

MLA compresses the per-position payload. Sparse attention shortens the position list. DeepSeek-V3.2 combines both — Chapter 11.

Concept-to-code map for the chapter

- **MLA layer and its two cache buffers** → `DeepSeek-V3/inference/model.py`

Concept-to-code map for the chapter

- **MLA layer and its two cache buffers** → `DeepSeek-V3/inference/model.py`
- **latent rank, NoPE and RoPE widths** → `DeepSeek-V3/inference/configs/*.json`

Concept-to-code map for the chapter

- **MLA layer and its two cache buffers** → `DeepSeek-V3/inference/model.py`
- **latent rank, NoPE and RoPE widths** → `DeepSeek-V3/inference/configs/*.json`
- **released cache-shape contract** → `DeepSeek-V3/README.md`

Concept-to-code map for the chapter

- **MLA layer and its two cache buffers** → `DeepSeek-V3/inference/model.py`
- **latent rank, NoPE and RoPE widths** → `DeepSeek-V3/inference/configs/*.json`
- **released cache-shape contract** → `DeepSeek-V3/README.md`
- **sparse-attention kernel surface** → `FlashMLA/flash_mla/flash_mla_interface.py`

Concept-to-code map for the chapter

- **MLA layer and its two cache buffers** → `DeepSeek-V3/inference/model.py`
- **latent rank, NoPE and RoPE widths** → `DeepSeek-V3/inference/configs/*.json`
- **released cache-shape contract** → `DeepSeek-V3/README.md`
- **sparse-attention kernel surface** → `FlashMLA/flash_mla/flash_mla_interface.py`

Store a latent, reconstruct the rest, keep the rotation separate.

Next: the serving chapter turns this cache contract into pages, kernels, and a decode loop (Volume II).