# Llama3 Inferencing (part 3)

## Architecture | Inference

Dr. Karthik Mohan, March 11th 2026

# Previous Lecture

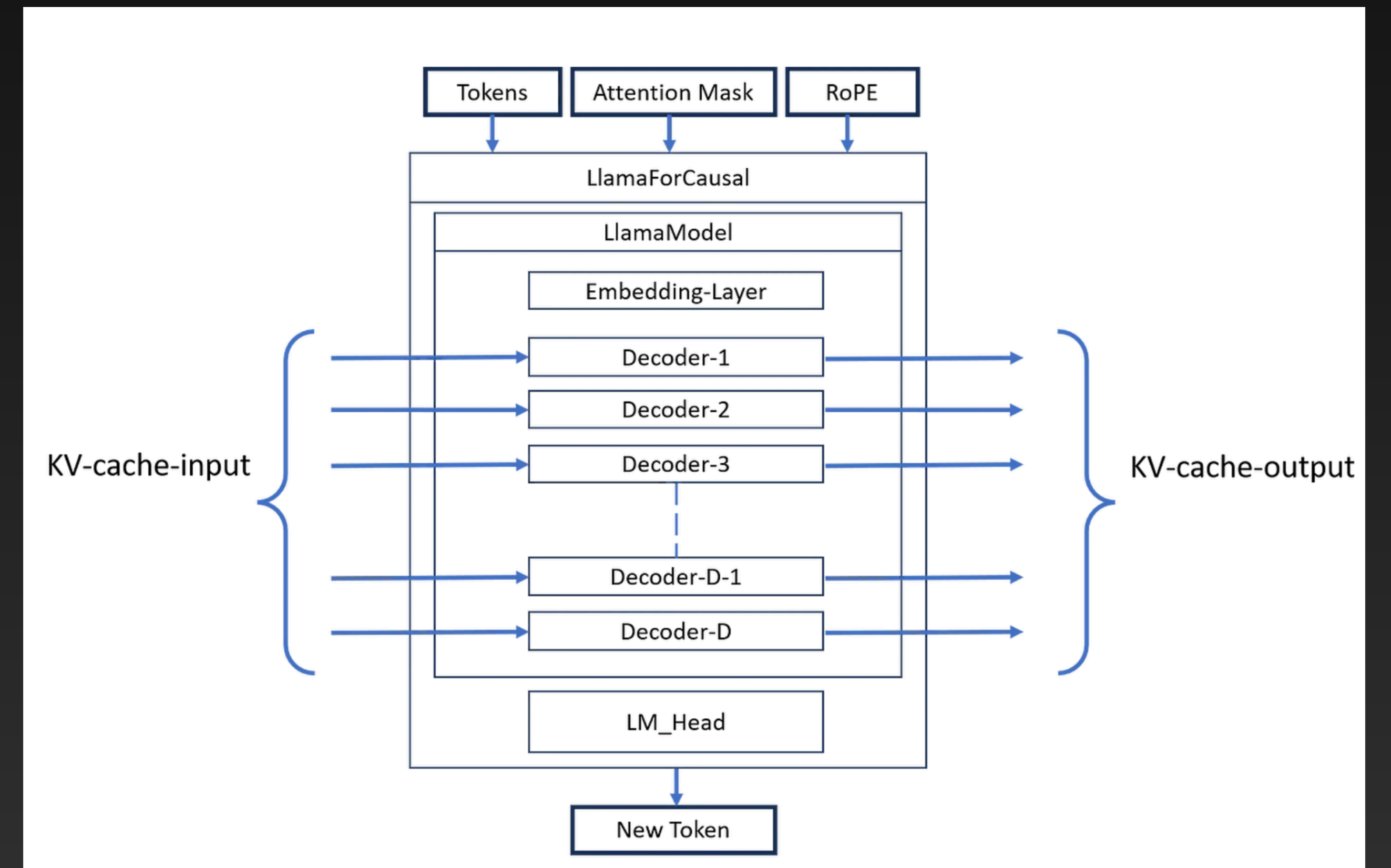
- Llama3 models
- KV-Caching
- Faster Inferencing

# Today's Lecture

- KV-Cache recap
- VLLM
- DeepSeek Models
- MoE and MLA (mixture of experts & Multi-head latent attention)

# Inference Without KV Cache

Assume “Quick Brown Fox jumped over….”
T1 t2 t3 t4 t5
Need to predict t6
I only need hidden state h5 of last layer to predict t6.
But that requires knowing the keys, values at every single Layer!

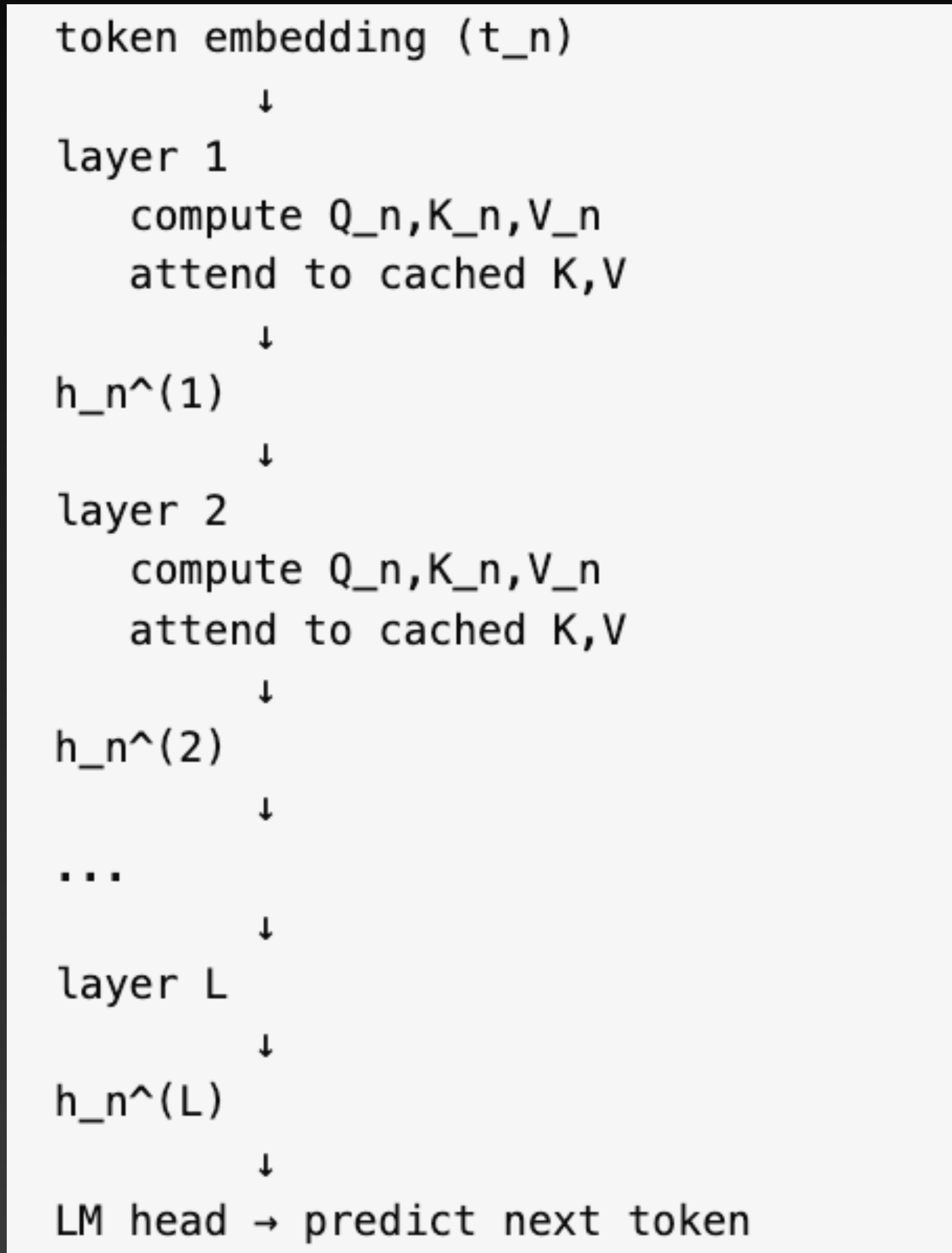



**Without KV cache**

Every generation step recomputes the entire triangular matrix:

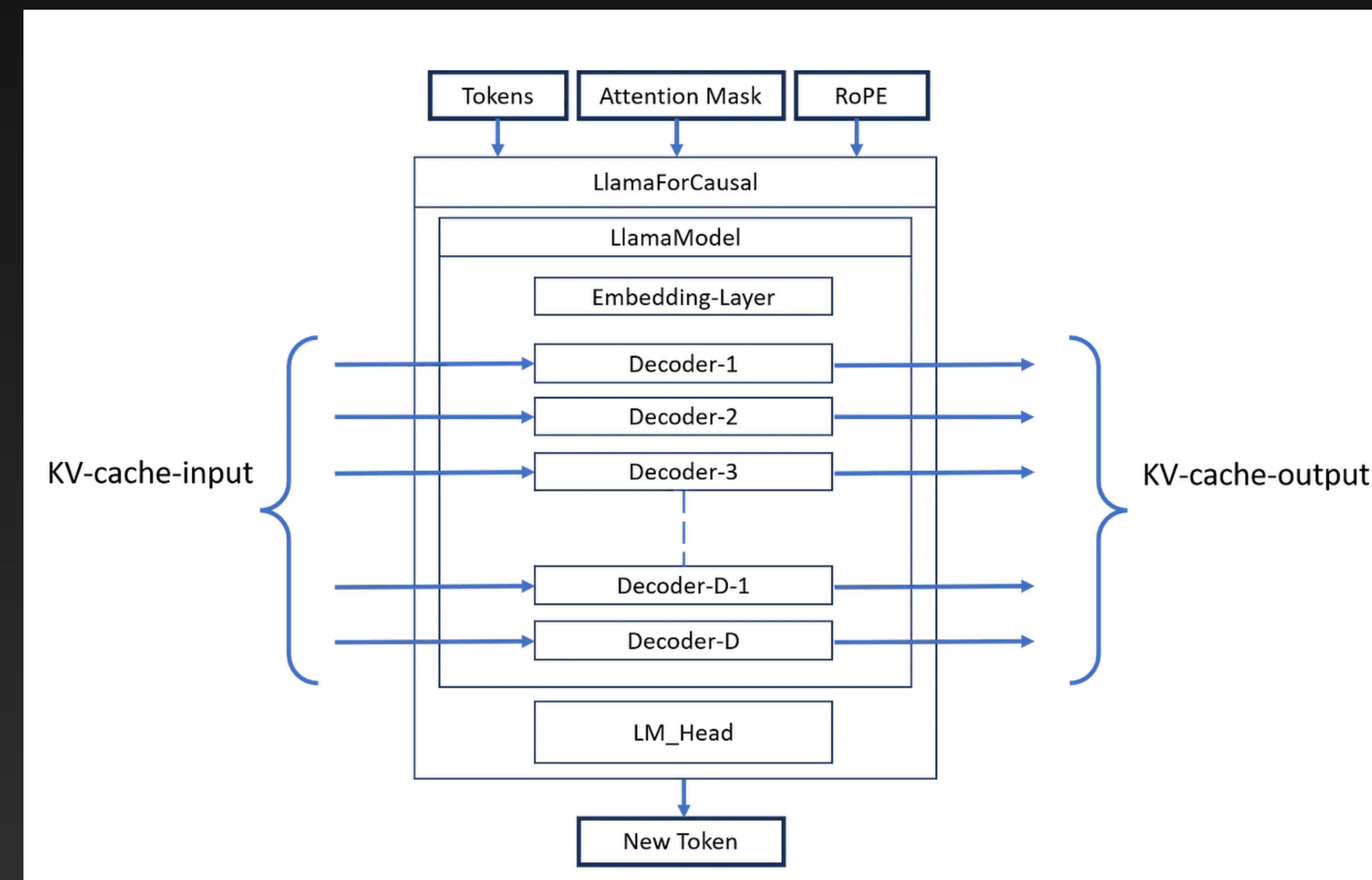
```
Q1 → K1
Q2 → K1 K2
Q3 → K1 K2 K3
Q4 → K1 K2 K3 K4
Q5 → K1 K2 K3 K4 K5
```

# KV Cache usage




**Current Cached Memory**

K_1,K_2,...,K_(n-1)
V_1,V_2,....,V_(n-1)



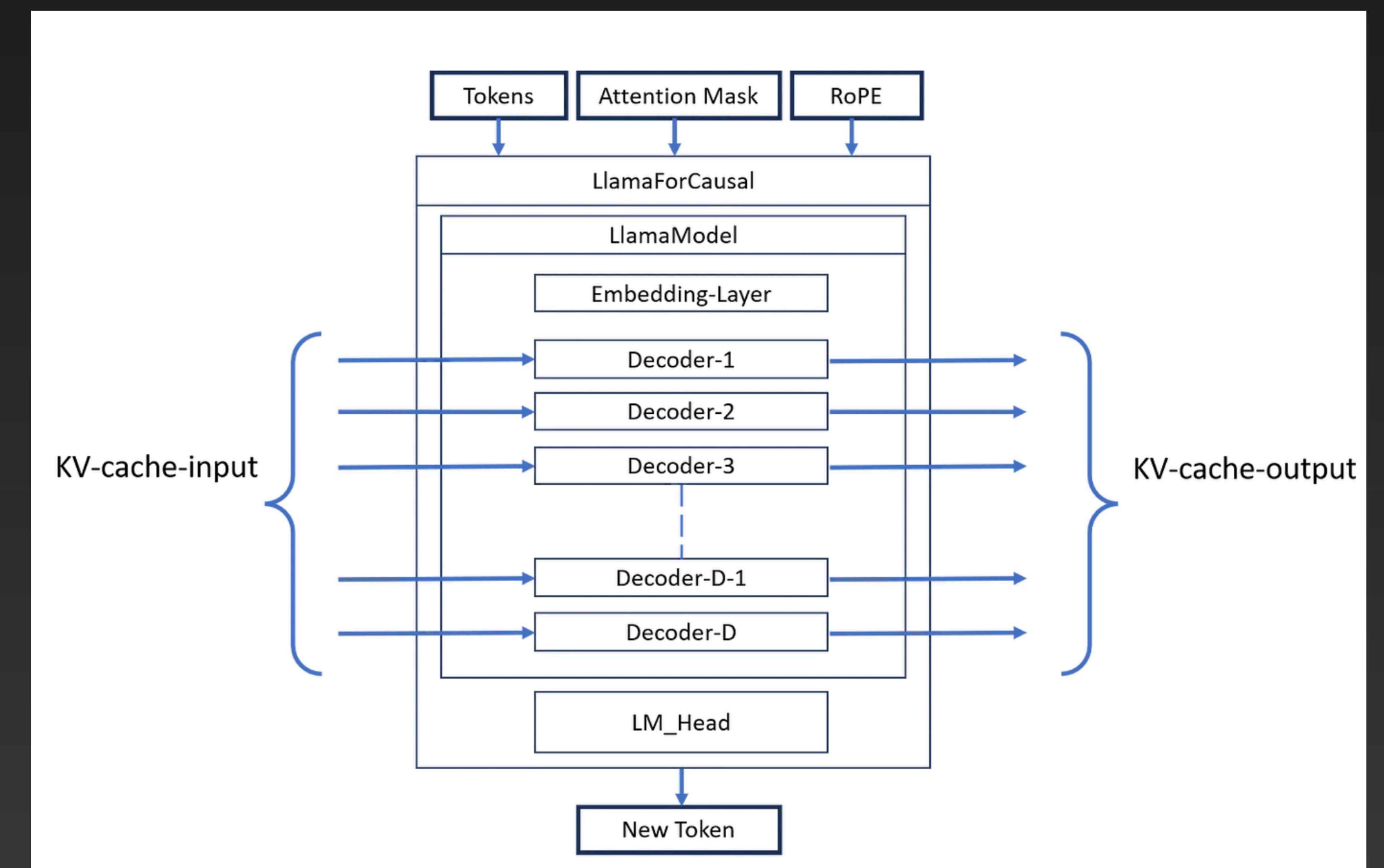
# Inference With KV Cache

Assume “Quick Brown Fox jumped over....”
T1 t2 t3 t4 t5
Need to predict t6
I only need hidden state h5 of last layer to predict t6.
But that requires knowing the keys, values at every single Layer!

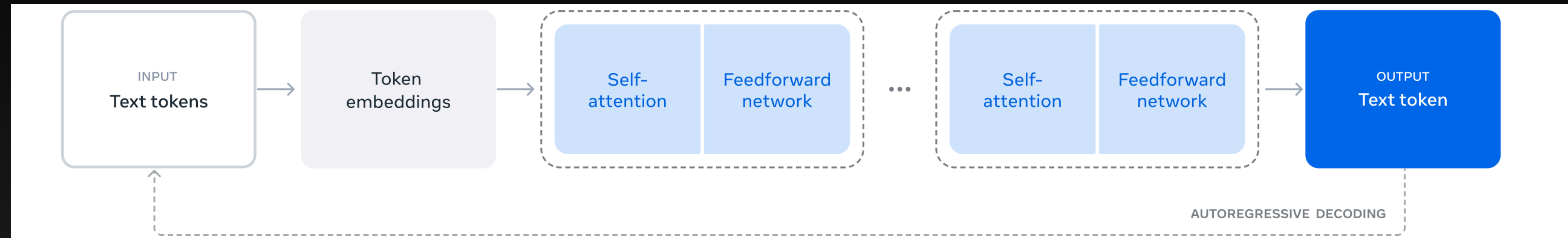
**With KV cache**

Only the new row is computed:

Q5 → K1 K2 K3 K4 K5



# KV Cache compute example

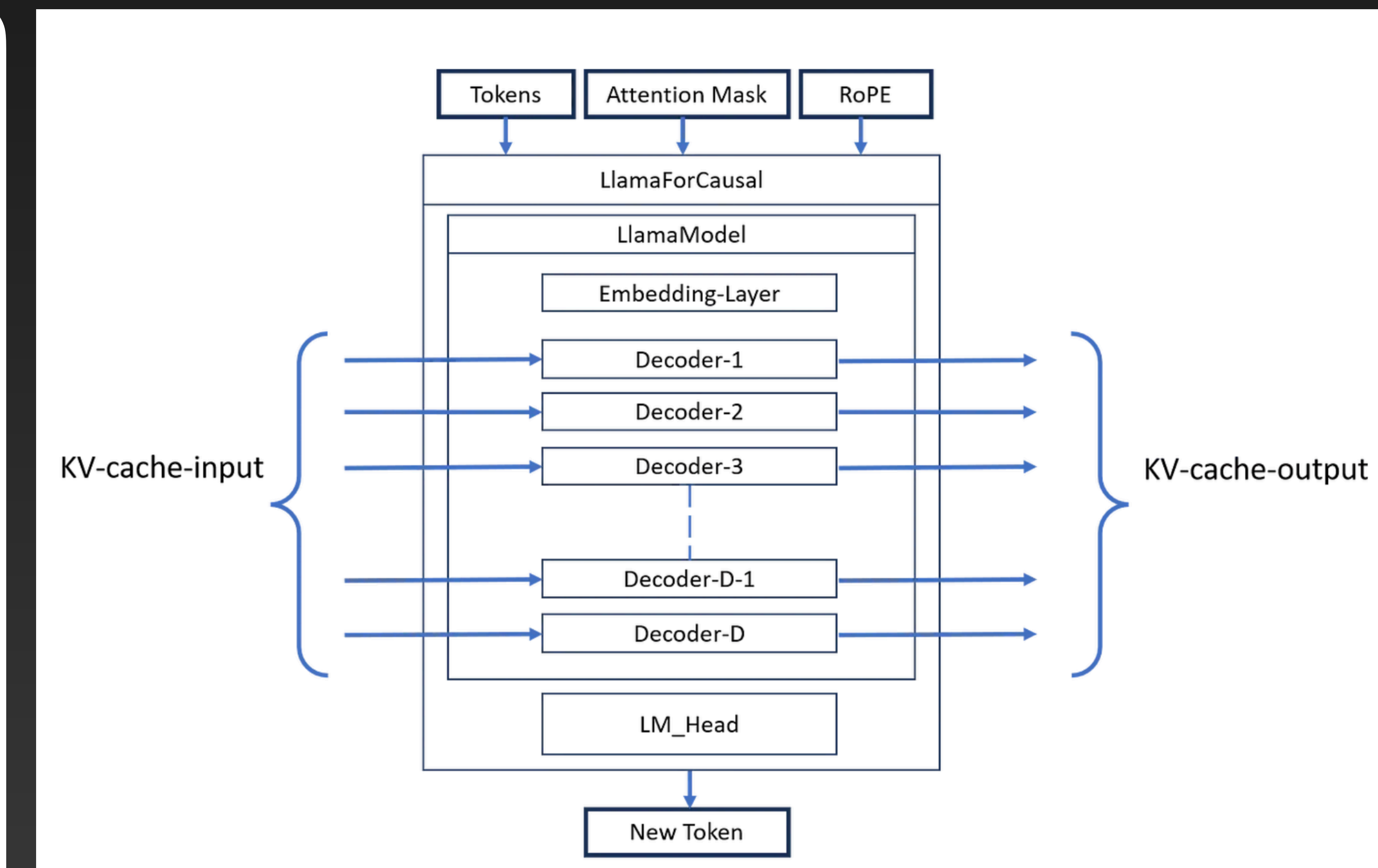



KV-Cache Memory consumption:
#layers x #heads/layer x dim/head x seq length x bytes/parameter x 2 (one for K and one for V)

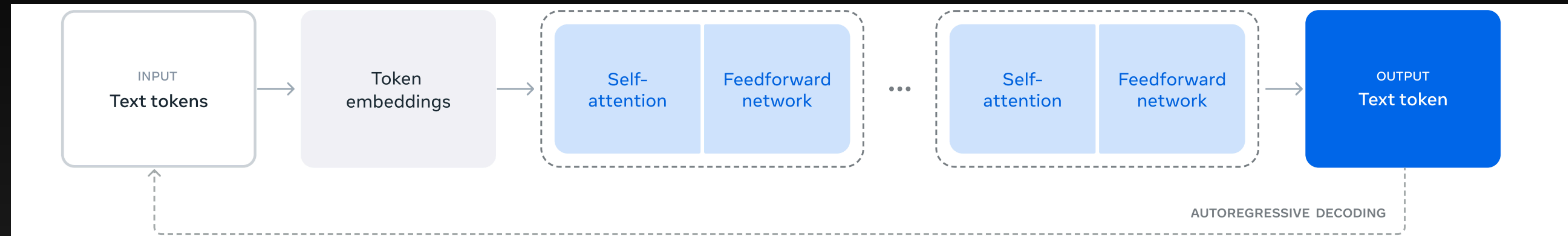
E.g. Llama3-70b model
Model ram size at fp32 -> 70*4 bytes (fp32 precision) = 280GB (or 4 H100 80GB ram GPUs)!

KV-Cache Memory Consumption
80 layers, 64 heads, 128 head dim, 8k hidden dimensions, 8k context window

Mem consumption = 80*64*128*8k*2*4 = 40GB



# KV Cache compute example (Long Context)

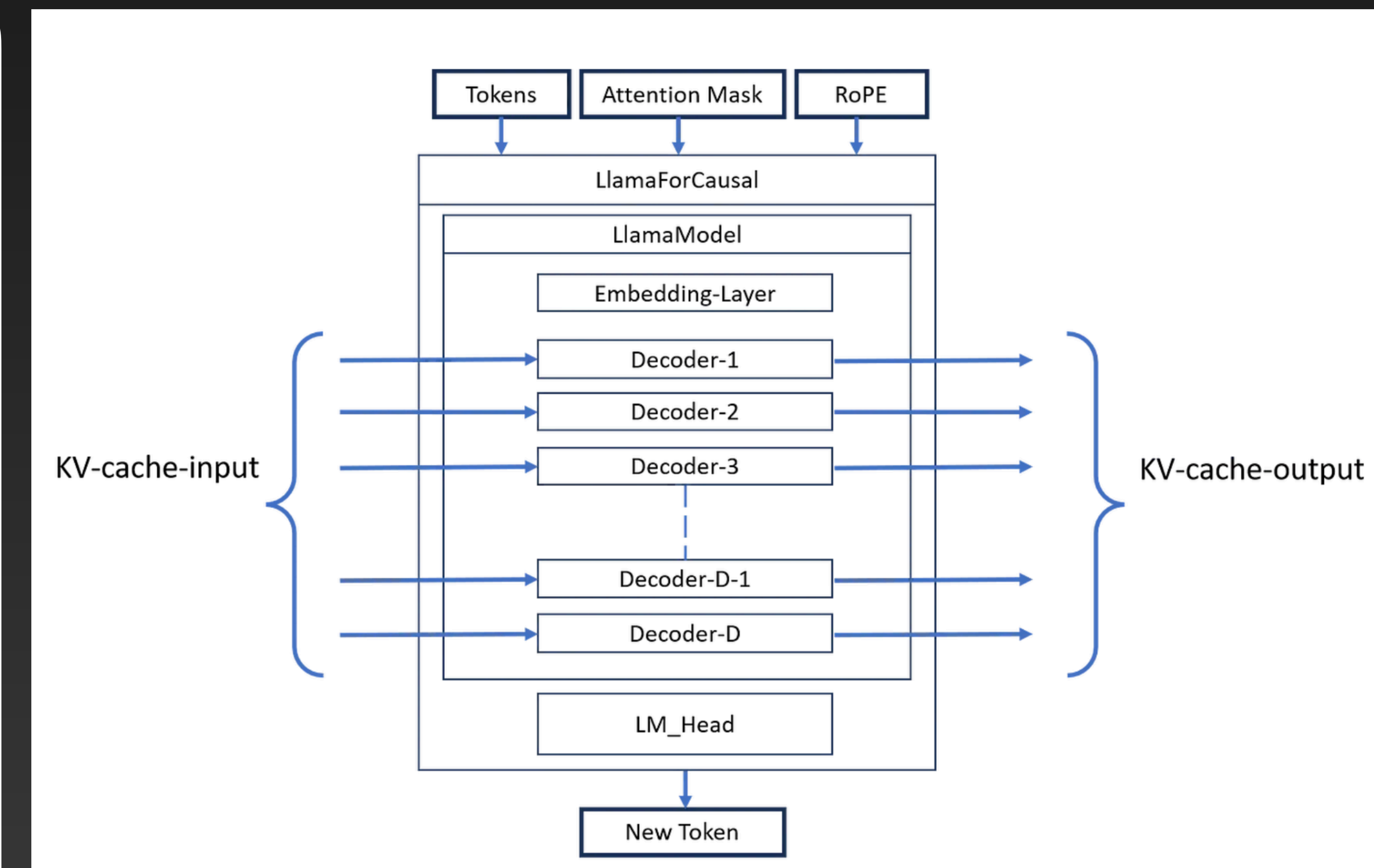



KV-Cache Memory consumption:
#layers x #heads/layer x dim/head x seq length x bytes/parameter x 2 (one for K and one for V)

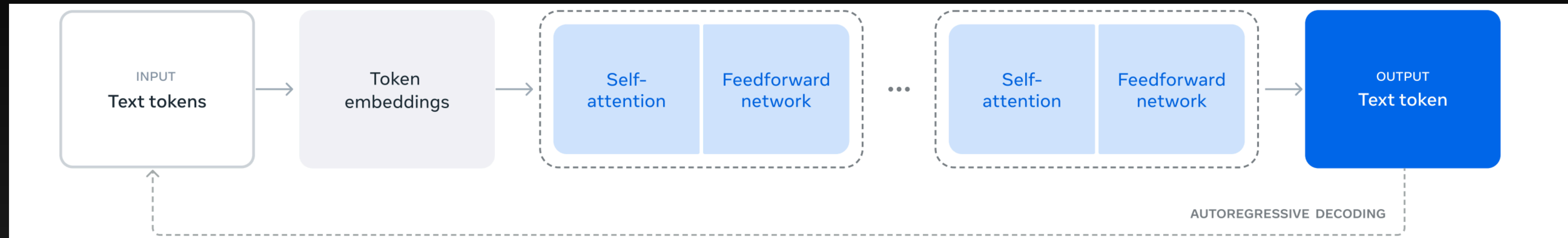
E.g. Llama3-70b model
Model ram size at fp32 -> 70*4 bytes (fp32 precision) = 280GB (or 4 H100 80GB ram GPUs)!

KV-Cache Memory Consumption
80 layers, 64 heads, 128 head dim, 8k hidden dimensions, 128k long context window

Mem consumption = 80*64*128*128k*2*4 = 670 GB



# KV Cache Efficiency



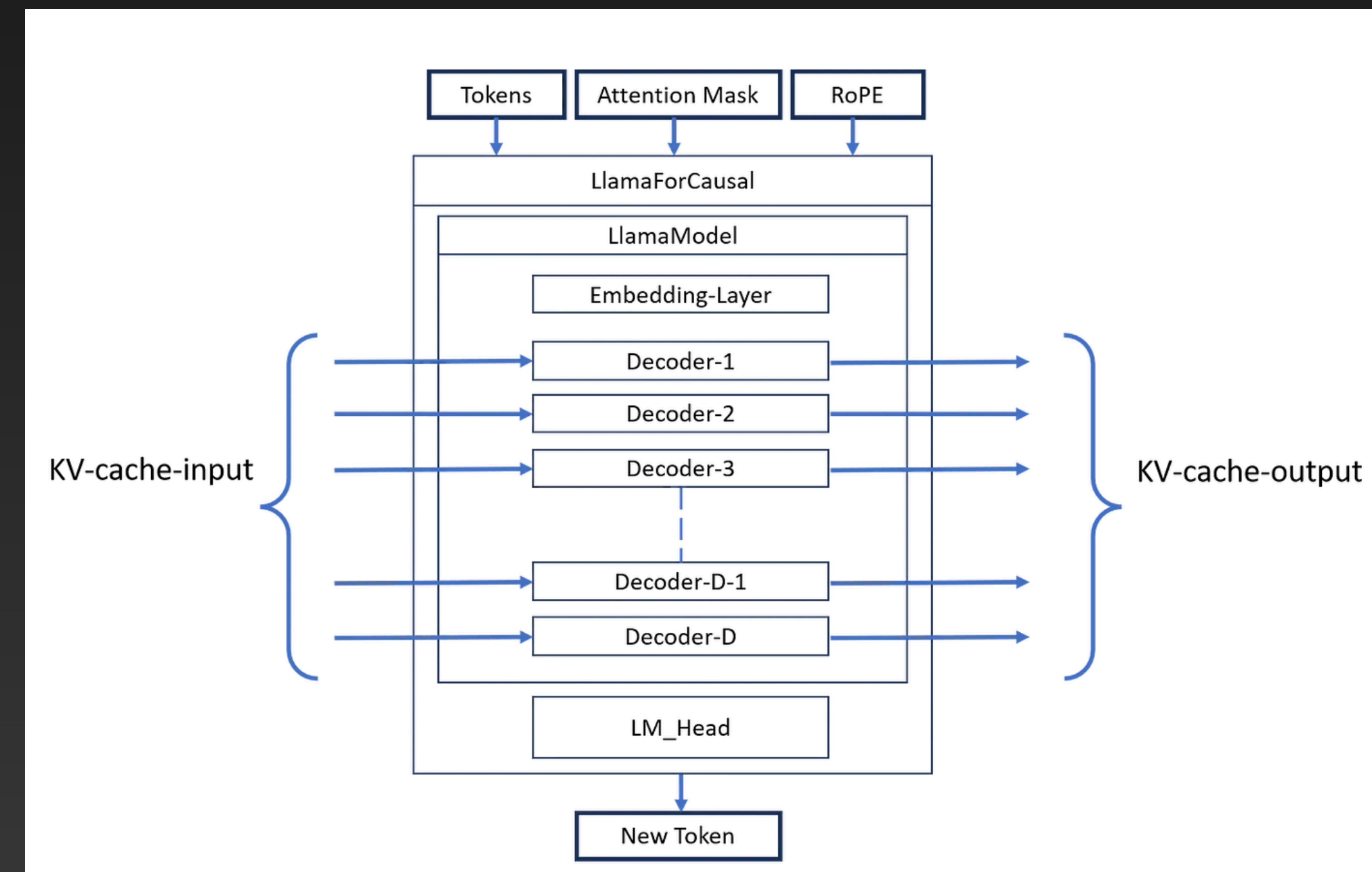

Assume "Quick Brown Fox jumped ...."
T1 t2 t3 t4
Need to predict t5

**Without KV-cache**

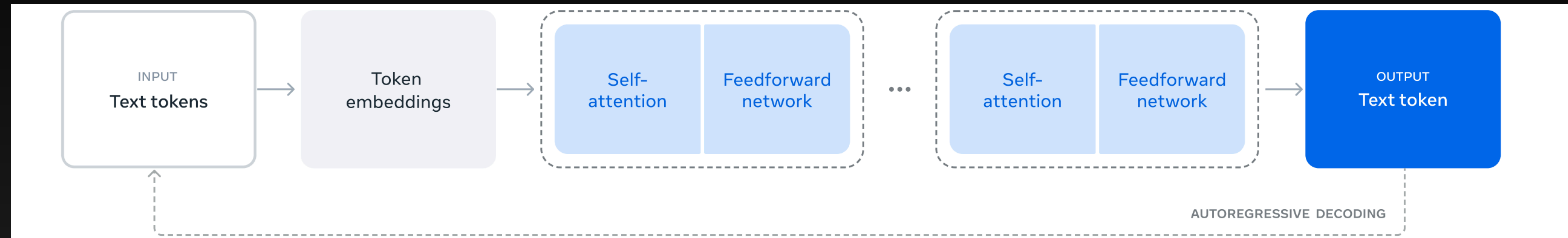
Compute K1,K2,K3,K4 and V1,V2,V3,V4 at each layer! And then finally with the last hidden representation of token 4, h4 -> Pass into the LM_head to get the new prediction t5

**With KV-cache**

K1...K3 and V1..V3 are already cached into memory for each of 64 layers of Llama3. Just compute attention for 4th token in each layer. Get h4 -> Pass into LM_head and get new prediction t5



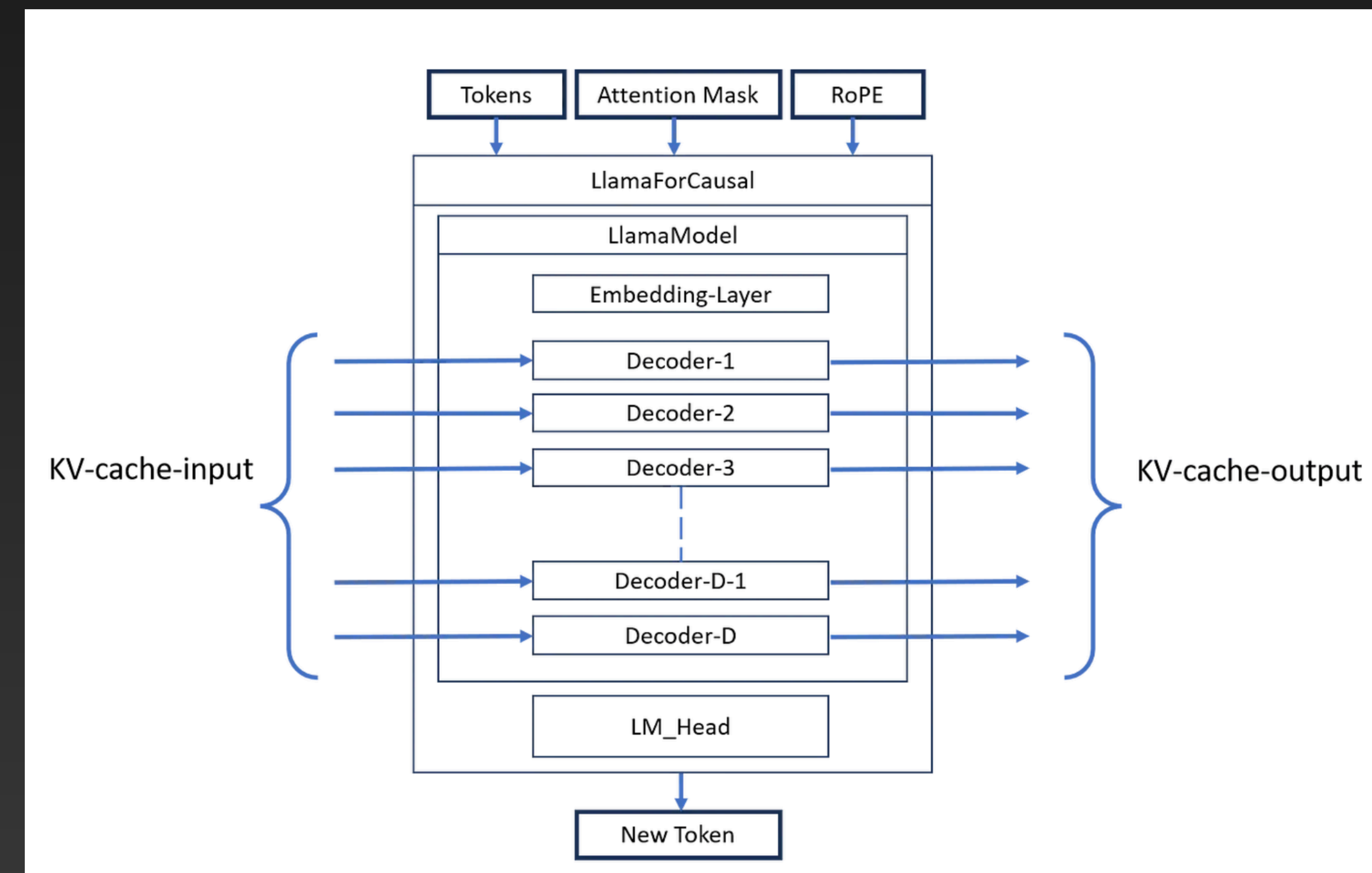
# Inference phases



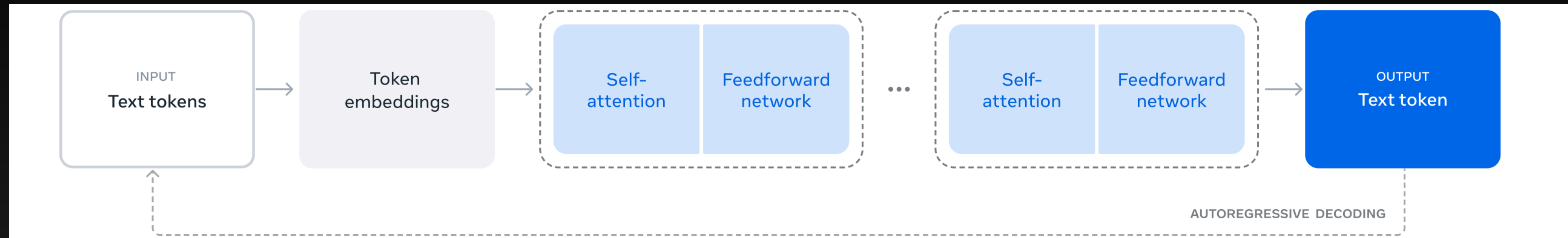

Assume “Quick Brown Fox jumped ....”
T1 t2 t3 t4
Need to predict t5

Pre-fill phase for KV-cache build
Compute K1,K2,K3,K4 and V1,V2,V3,V4 at each layer by passing all tokens at once

Next Token Phase
Predict next token given K-V cache + add new Key, Value at each layer for the new token and repeat



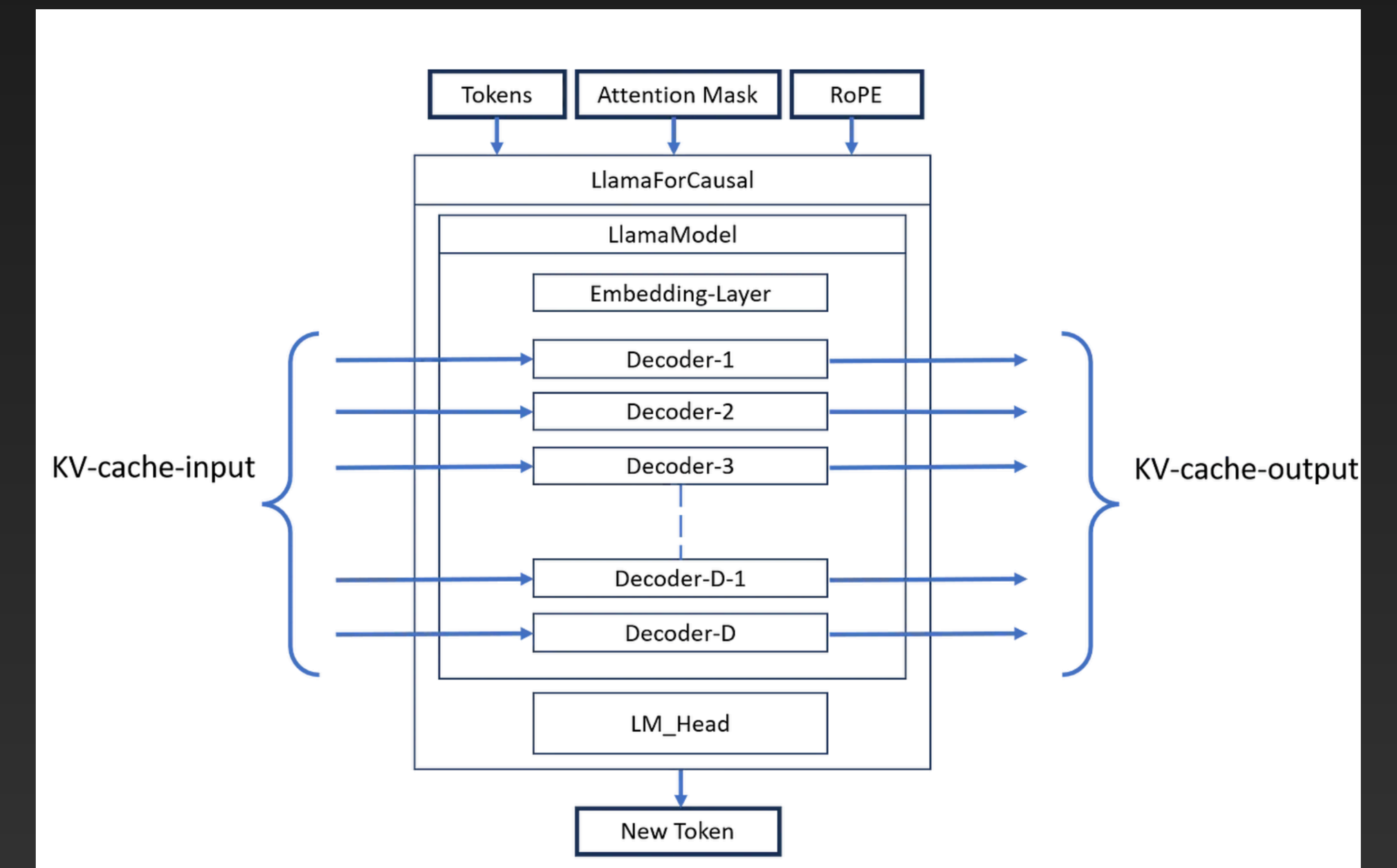
# Parsing Inference Metrics



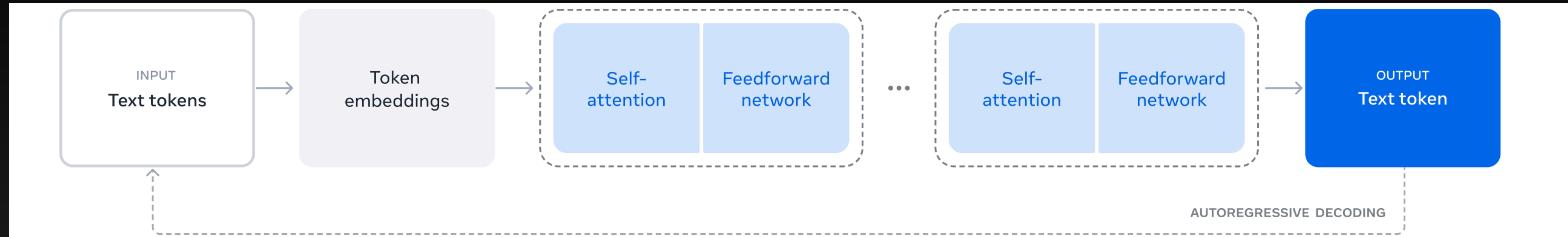

Assume "Quick Brown Fox jumped …."
T1 t2 t3 t4
Need to predict t5

TTFT: What is time to first token? (Pre-fill phase)
It's time to generate first token, which in this case is t5 (over)

TPT: What is time per token? (Next token prediction phase)
Time between successive tokens t5->t6, t6->t7, etc

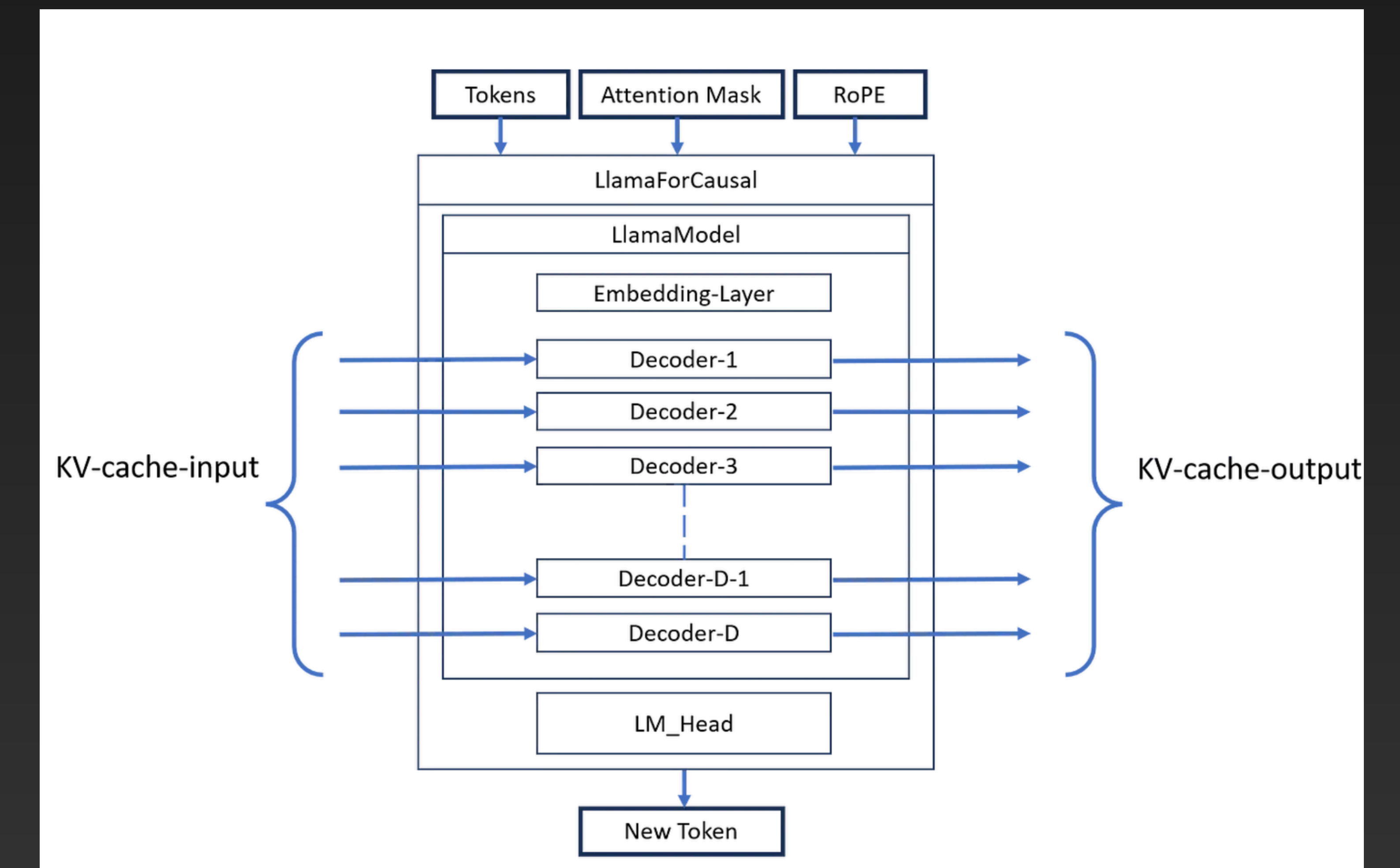


# Parsing Inference Metrics




## Typical TTFT and Time-per-Token (Llama-3)

| Model | Typical TTFT (prompt ~100–500 tokens) | Tokens/sec (generation) | Time per token |
|---|---|---|---|
| **Llama-3-8B** | ~0.4 – 1.2 s | ~80 – 150 tok/s | ~7 – 12 ms |
| **Llama-3-70B** | ~1.5 – 4 s | ~15 – 40 tok/s | ~25 – 65 ms |



# Inference Pipeline

- Prompt -> Tokenization -> Transformer -> Next Token -> Repeat

| Metric | Meaning |
| --- | --- |
| TTFT | Time to First Token |
| TPOT | Time Per Output Token |
| Throughput | Tokens/sec across requests |
| Latency | End-to-end response time |

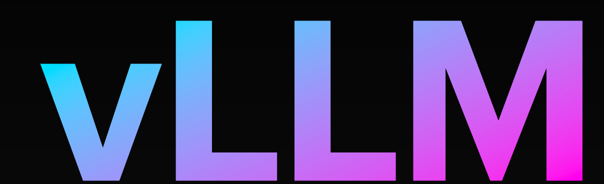
# LLM Serving

Imagine 100 users send prompts at the same time? What happens?

# LLM Serving

Imagine 100 users send prompts at the same time? What happens?

KV-cache explosion. For llama3-8b - 1 request can consume 2GB ram for 2000 token request. With 100 requests - peak KV-cache memory can go to 200 GB




**vLLM is a high-throughput LLM inference engine that efficiently manages KV cache memory and batching.**

Key innovations:

| Feature | Purpose |
| --- | --- |
| PagedAttention | Efficient KV memory management |
| Continuous batching | GPU always busy |
| Fast scheduling | High throughput |
| Prefix caching | reuse prompt computation |
| Speculative decoding | faster generation |

# Paged Attention

Instead of contiguous memory:

Split KV cache into **blocks/pages**.

Example:

```
Page size = 16 tokens
```

Memory layout:

```
Page1: tokens 1–16
Page2: tokens 17–32
Page3: tokens 33–48
```

Pages stored anywhere in GPU memory.

# Continuous Batching

Traditional batching:

Batch fixed requests together.

```
Batch1 = 4 requests
wait until finished
Batch2 = next 4
```

Bad for latency.

---

### vLLM: Continuous batching

Requests join and leave batch dynamically.

Example timeline:

```
Step1: A B C
Step2: A B C D
Step3: B C D E
Step4: C D E
```

Batch constantly updated.

# Prefix Caching

Many prompts share prefixes.

Example:

```
User1:
Explain reinforcement learning

User2:
Explain reinforcement learning with examples

User3:
Explain reinforcement learning in robotics
```

Shared prefix:

```
Explain reinforcement learning
```

---

Instead of recomputing:

vLLM caches prefix KV.

New request reuses it.

Huge speedup.

# Speculative Decoding

Idea:

Use a **small draft model**.

Example:

Small model predicts:

```
token1 token2 token3 token4
```

Large model verifies.

If correct → accept all tokens.

Otherwise → recompute.

---

Benefit:

Large model runs **less frequently**.

Speed improvement.