Llama3 Inferencing

Architecture | Inference

Dr. Karthik Mohan, March 9th 2026

Previous Lecture

- Types of Training
- Model parallelism and Data parallelism for training

Today's Lecture

- Llama3 Arch
- Pre-Training/Post-Training
- KV-Cache

Llama3 Herd

Herd of models including 405B LM, 70B, 8B, 1B versions and also Llama Guard 3 for input/output safety

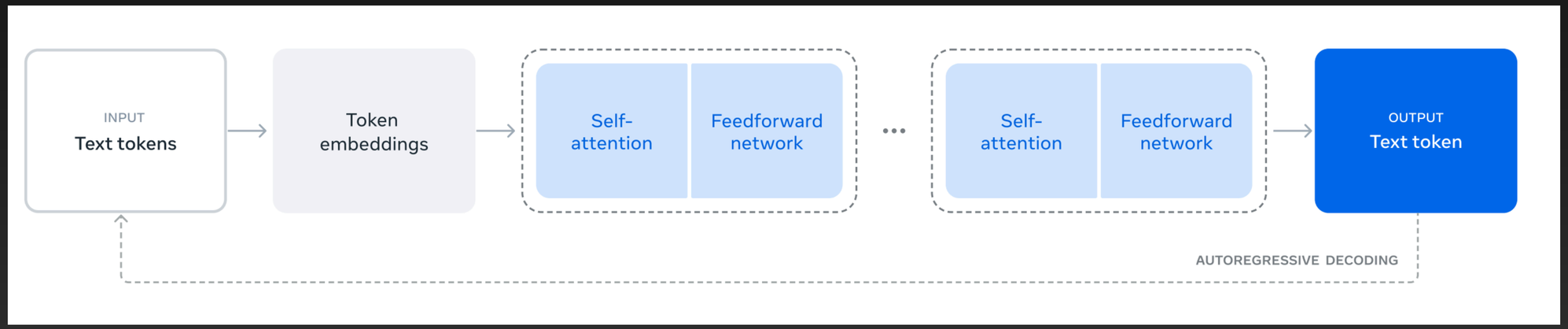
Reference: https://arxiv.org/pdf/2407.21783

Llama 3 Herd of Models

	Finetuned	Multilingual	Long context	Tool use	Release
Llama 3 8B	✗	✗[1]	✗	✗	April 2024
Llama 3 8B Instruct	✓	✗	✗	✗	April 2024
Llama 3 70B	✗	✗[1]	✗	✗	April 2024
Llama 3 70B Instruct	✓	✗	✗	✗	April 2024
Llama 3.1 8B	✗	✓	✓	✗	July 2024
Llama 3.1 8B Instruct	✓	✓	✓	✓	July 2024
Llama 3.1 70B	✗	✓	✓	✗	July 2024
Llama 3.1 70B Instruct	✓	✓	✓	✓	July 2024
Llama 3.1 405B	✗	✓	✓	✗	July 2024
Llama 3.1 405B Instruct	✓	✓	✓	✓	July 2024

Reference: https://arxiv.org/pdf/2407.21783

Llama3 Architecture



Llama3 Training Phases

Pre-Training

- Train 405B parameters
- 15.6T tokens ~ 150 M books
- Context window 8k with continued pre-training of 128k tokens
- Learns language structure and knowledge about the world

Llama3 Training Phases

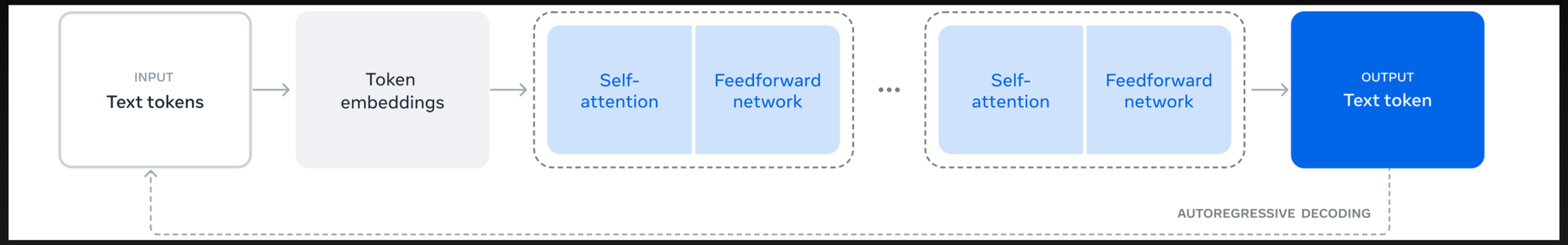
Post-Training

- Pre-Trained model doesn’t follow instructions
- Doesn’t behave an assistant is expected to
- Post-training has several rounds including SFT on instruction tuning data, and Direct Preference Optimization (DPO)
- Post-Training strong improvements in coding and reasoning capabilities
- Safety is also handled in the post-training stage
- Final model can handle questions in 8 languages, write quality code, solve reasoning out of the box

Llama3 Key Features

Context Window: 128k tokens
Vocab size: 128k tokens
Training data: 15T tokens
Decoder blocks: 32
Positional Embedding: RoPE

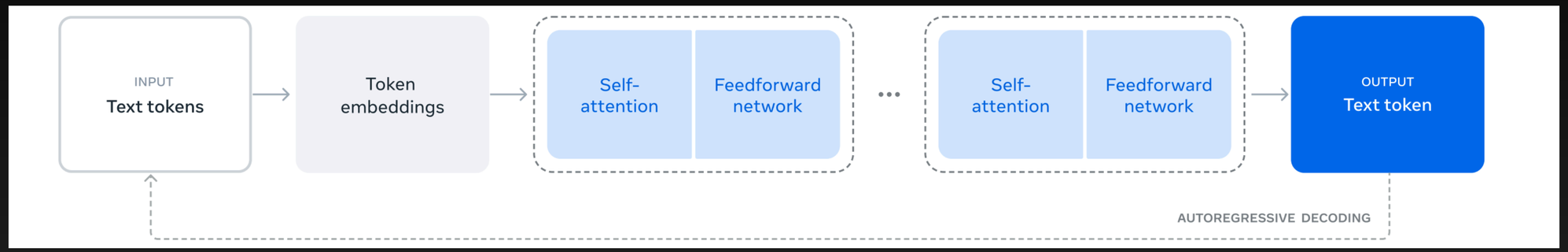
ICE #1

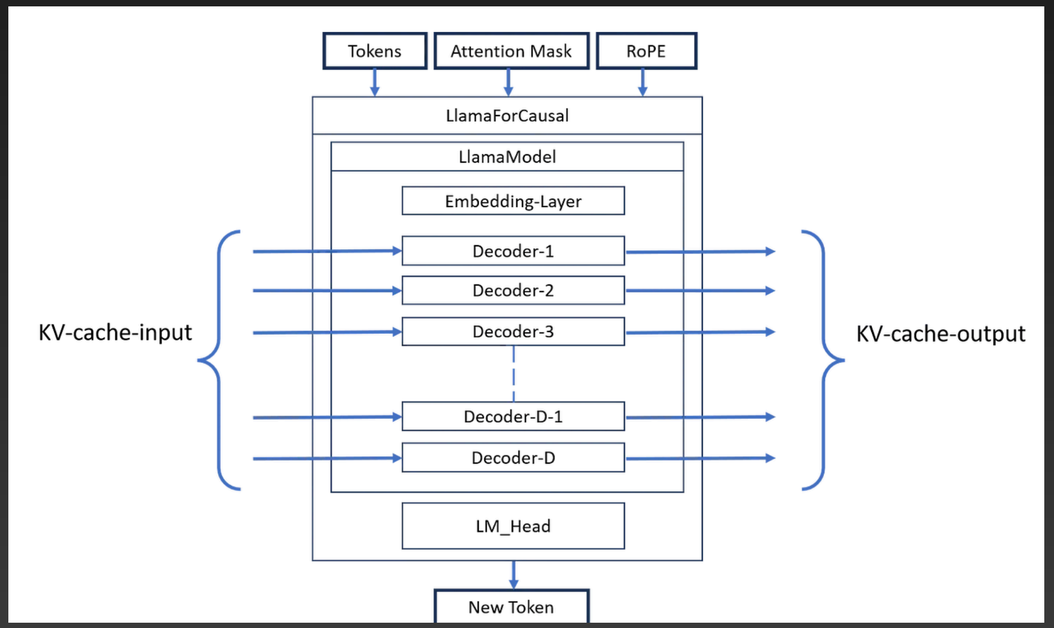


Why is increasing the input context window been a challenge for LLM models. Only recently have LLM models increased it from 4000 tokens to 100k tokens

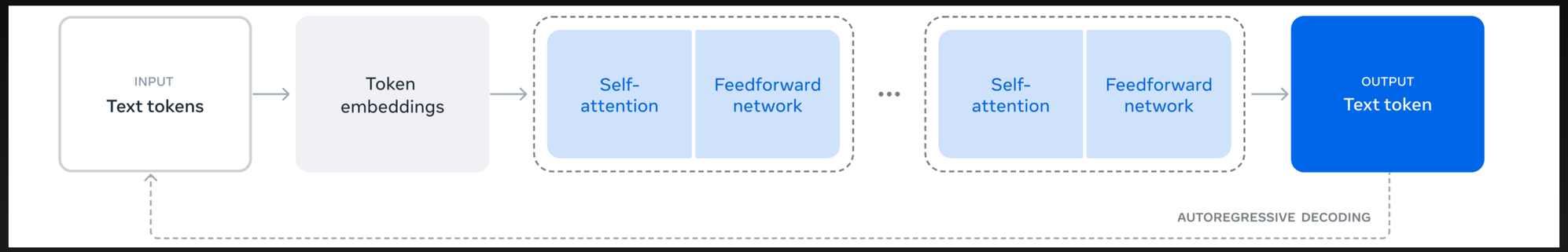
a) Compute increases linearly with context window size
b) Compute increases quadratically with context window size
c) Compute increases exponentially with context window size

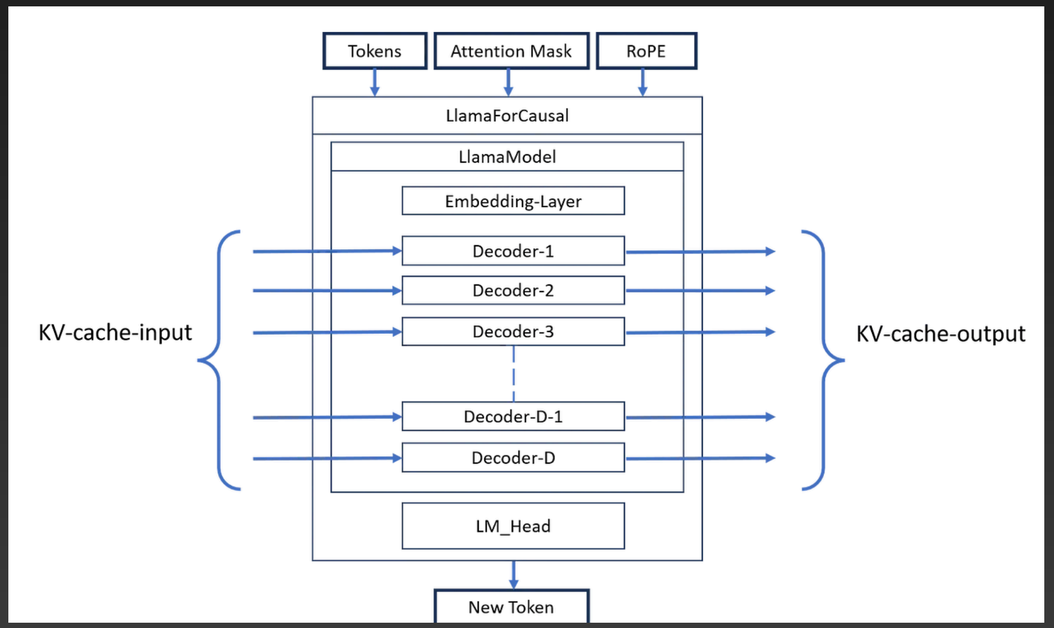
Llama3 Inferencing

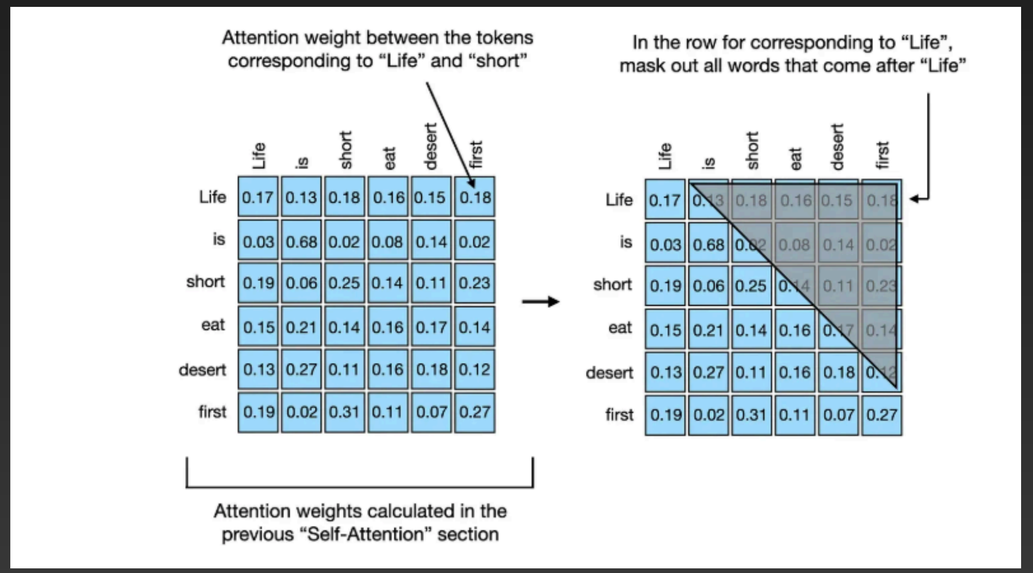




Llama3 Inferencing

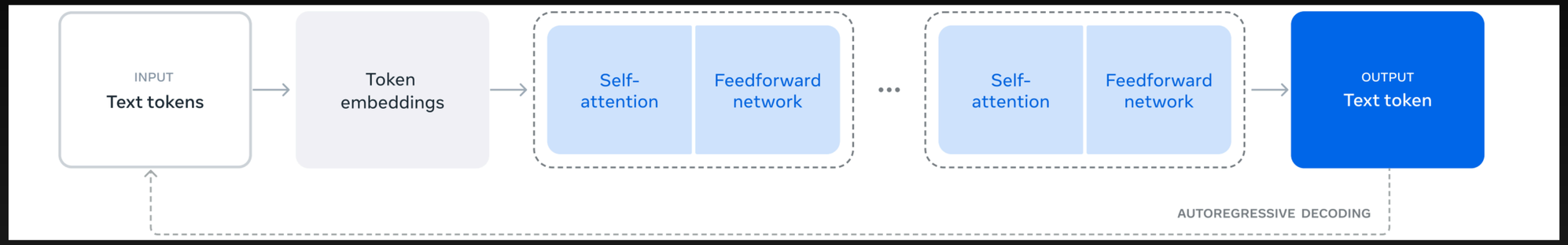






https://www.kaggle.com/code/aisuko/causal-self-attention

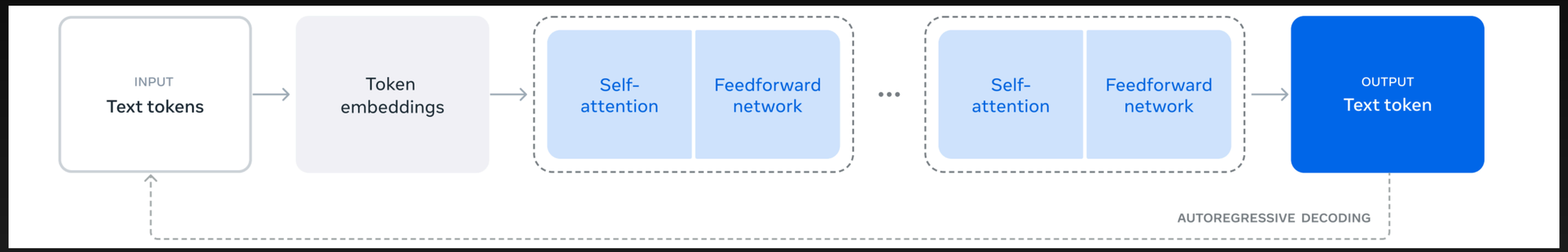
ICE #2

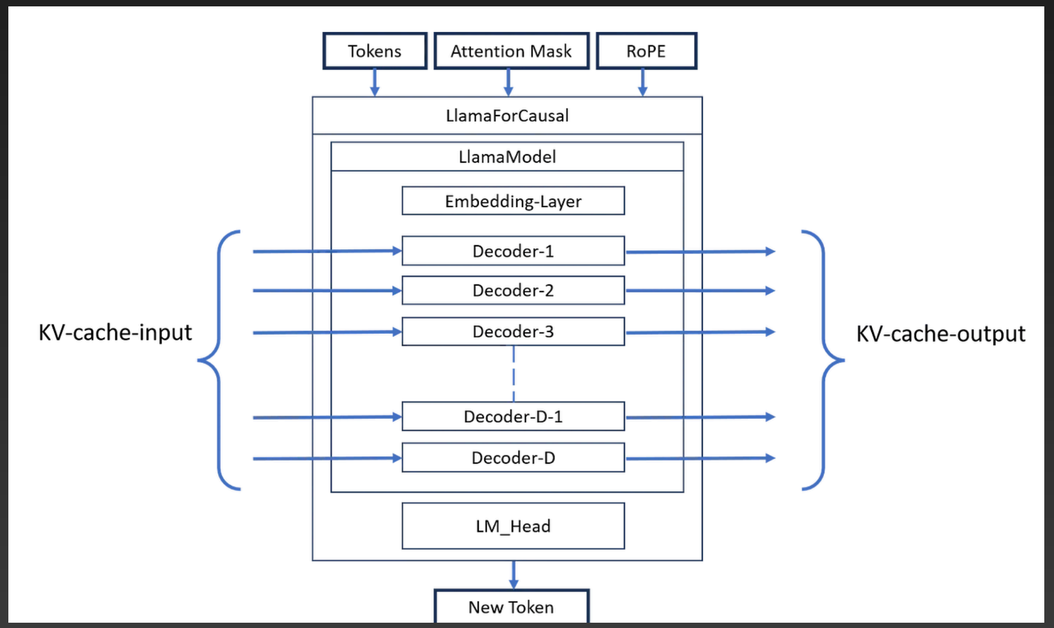


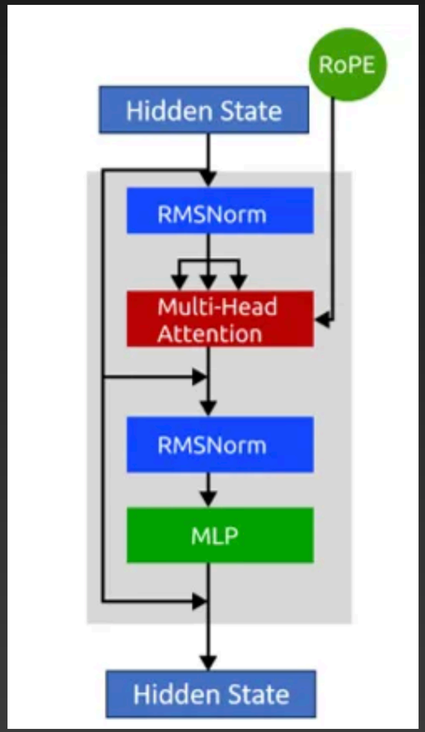
Assume that each word in a sentence corresponds to a token (for simplicity). Consider the sentence: “The sun rises”. Passing this into llama produces the output token as “in”. How many tokens will be passed in to the next step of autoregressive decoding and what will the expected output?

a) 3,”east”
b) 4,”east”
c) 3,”the
d) 4, “the”

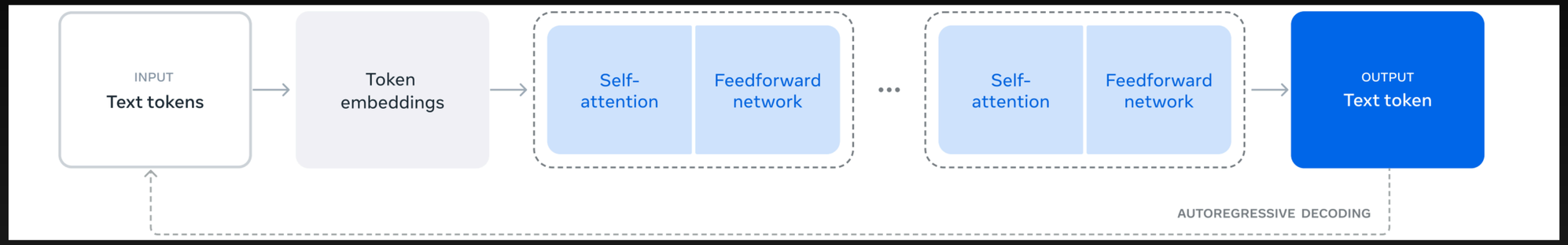
Llama3 Inferencing







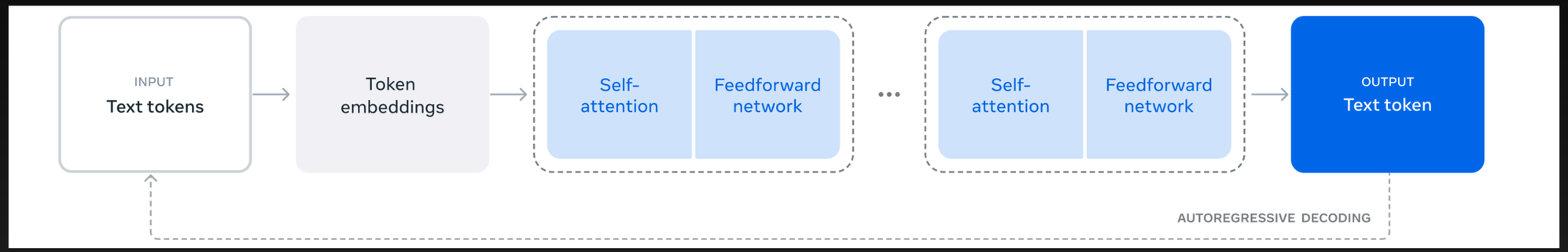
ICE #3

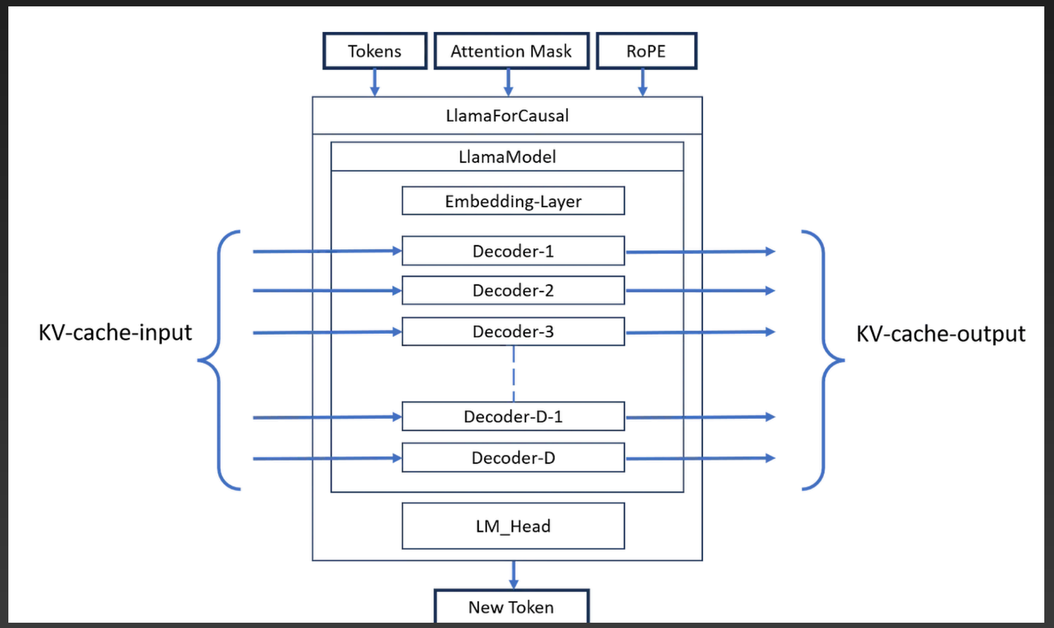


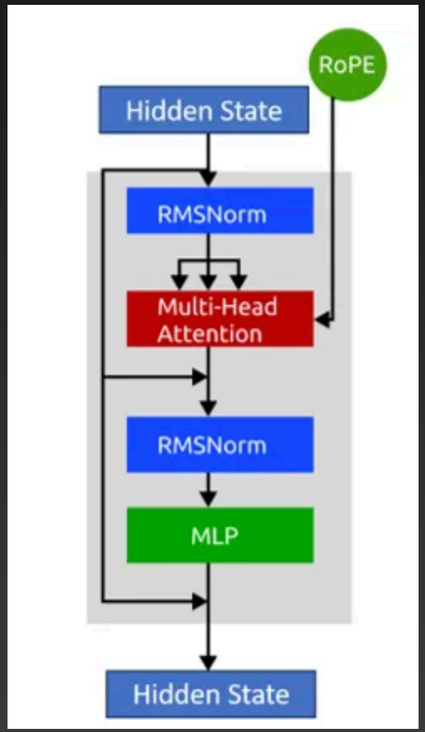
RMS norm is the root mean square of a vector of values, normalized by number of values in vector. E.g vec = [1,4,5] RMS(vec) = sqrt((1^2 + 4^2 + 5^2)/3). Which quantity in statistics is RMS norm closest to?

a) Mean
b) Variance
c) Standard Deviation
d) Expectation

Llama3 Inferencing

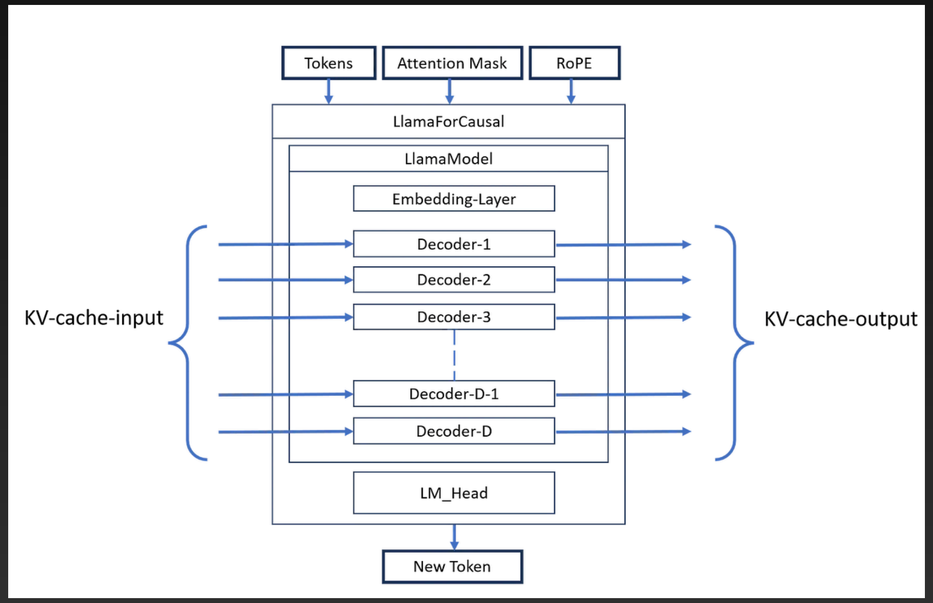






Inference Without KV Cache

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:

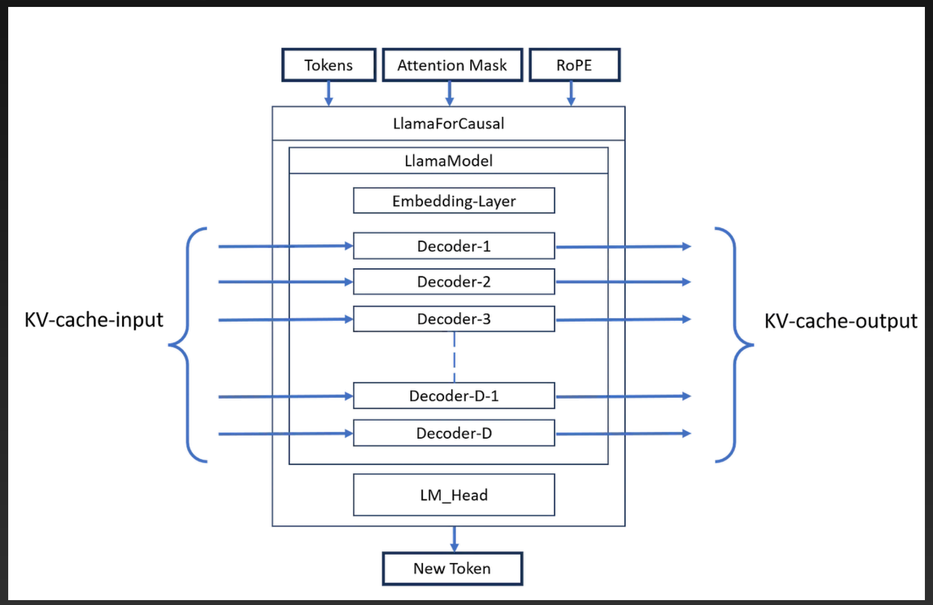
```
Q1 → K1
Q2 → K1 K2
Q3 → K1 K2 K3
Q4 → K1 K2 K3 K4
Q5 → K1 K2 K3 K4 K5
```

KV Cache usage

```
token embedding (t_n)
        ↓
layer 1
   compute Q_n,K_n,V_n
   attend to cached K,V
        ↓
h_n^(1)
        ↓
layer 2
   compute Q_n,K_n,V_n
   attend to cached K,V
        ↓
h_n^(2)
        ↓
...
        ↓
layer L
        ↓
h_n^(L)
        ↓
LM head → predict next token
```

Current Cached Memory

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



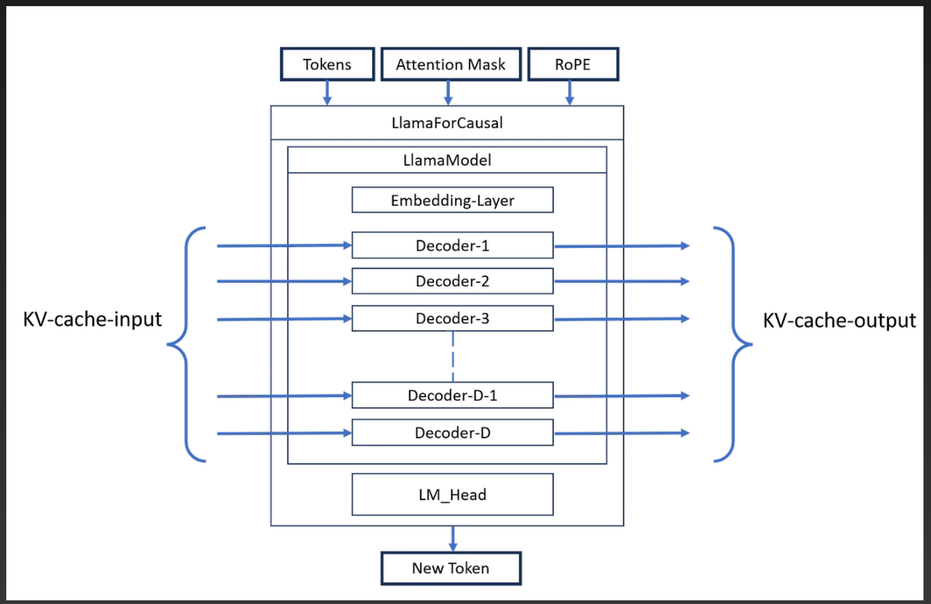
Inference With KV Cache

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!

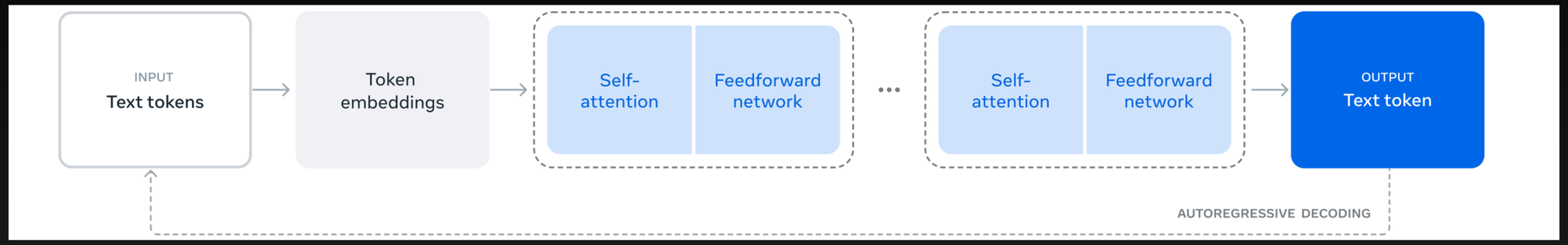
With KV cache

Only the new row is computed:

```
Q5 → K1 K2 K3 K4 K5
```

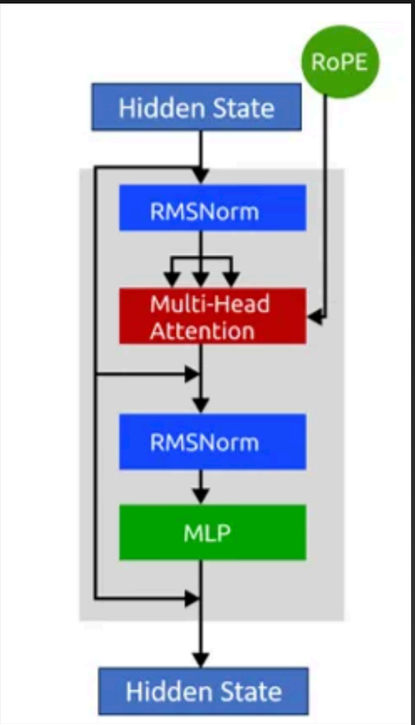


ICE #4

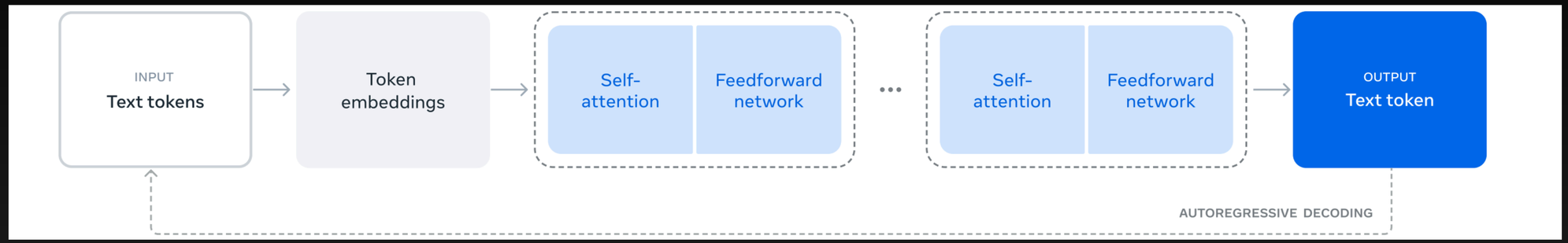


What is the role of KV-cache in the Llama architecture?

a) To compute attention using query, keys and values
b) To cache all the model parameters in memory
c) To speed up computation in the auto-regressive decoding process
d) To cache intermediate query, hidden embeddings of each layer to be used in the next decoding phase
e) All of the above



KV Cache compute example

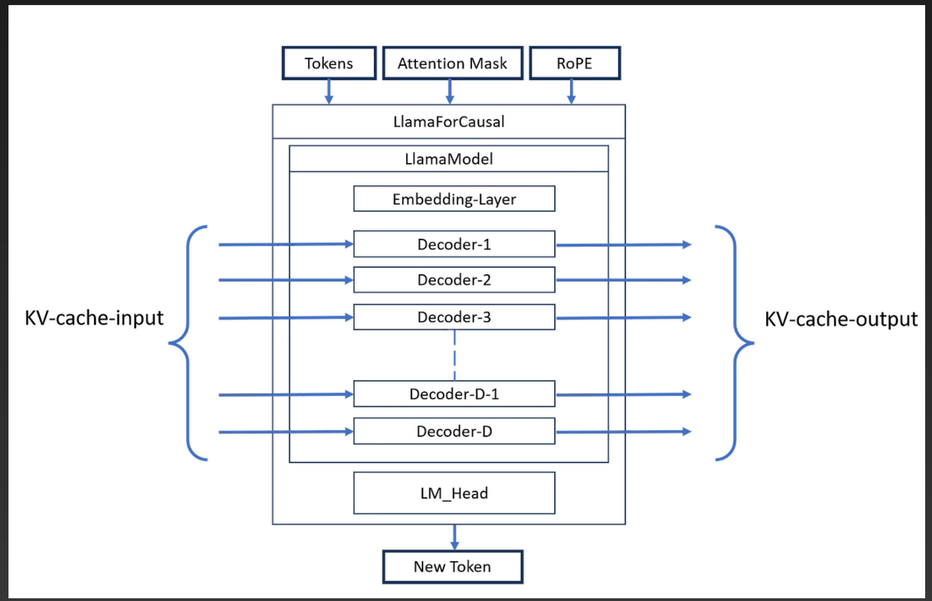


KV-Cache Memory consumption:
#layers x #heads/layer x dim/head x seq length x bytes/parameter x 2 (one for K and one for V)

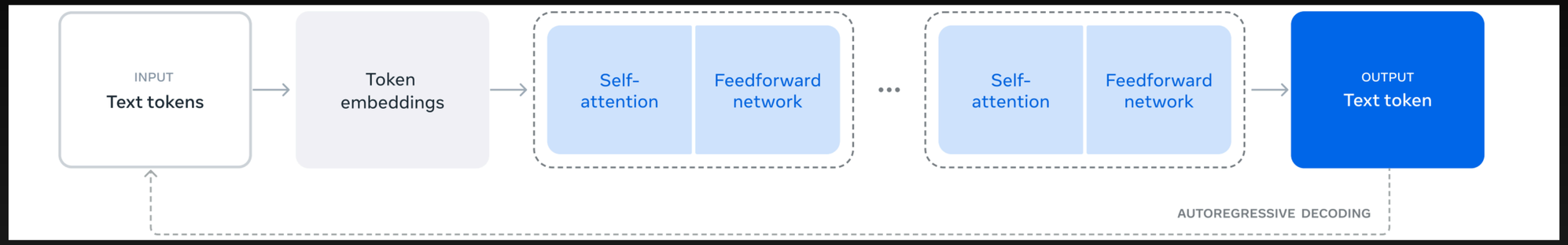
E.g. Llama3-70b model
Model ram size at fp32 -> 70*4 bytes (fp32 precision) = 280GB (or 4 H100 80GB ram GPUs)!

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)

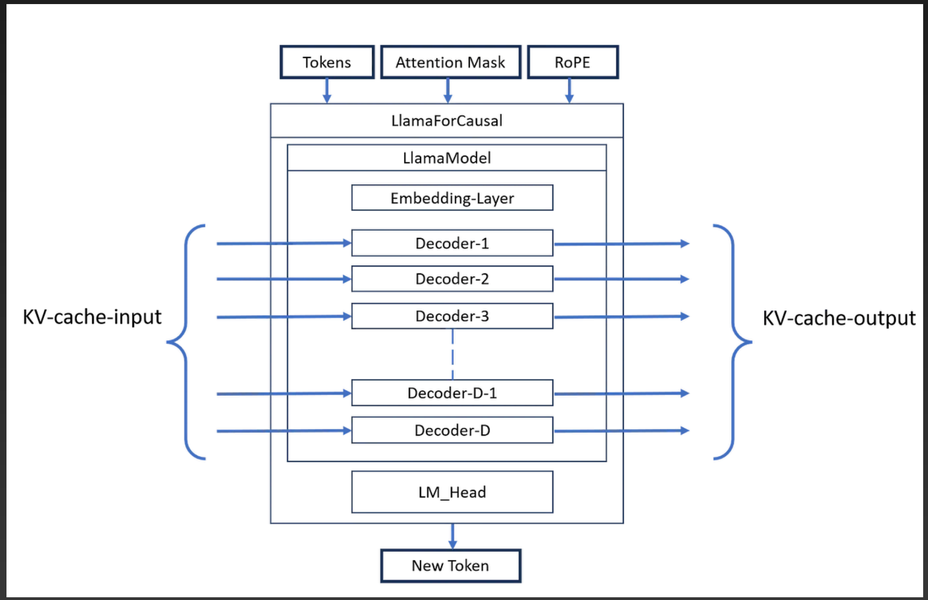


KV-Cache Memory consumption:
#layers x #heads/layer x dim/head x seq length x bytes/parameter x 2 (one for K and one for V)

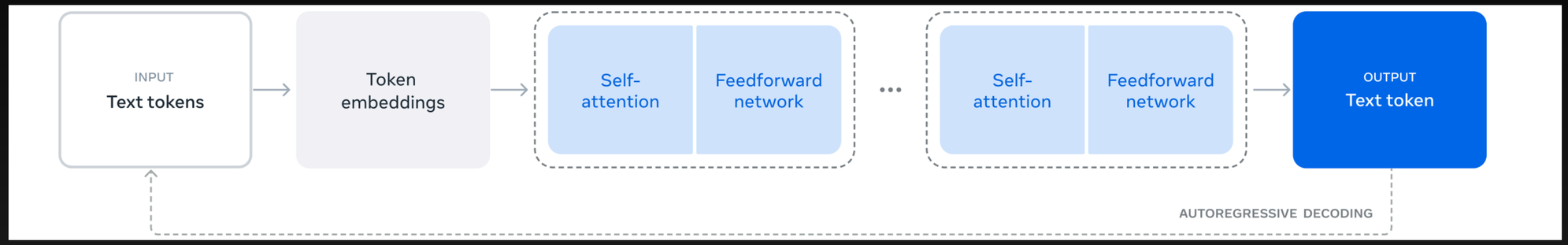
E.g. Llama3-70b model
Model ram size at fp32 -> 70*4 bytes (fp32 precision) = 280GB (or 4 H100 80GB ram GPUs)!

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

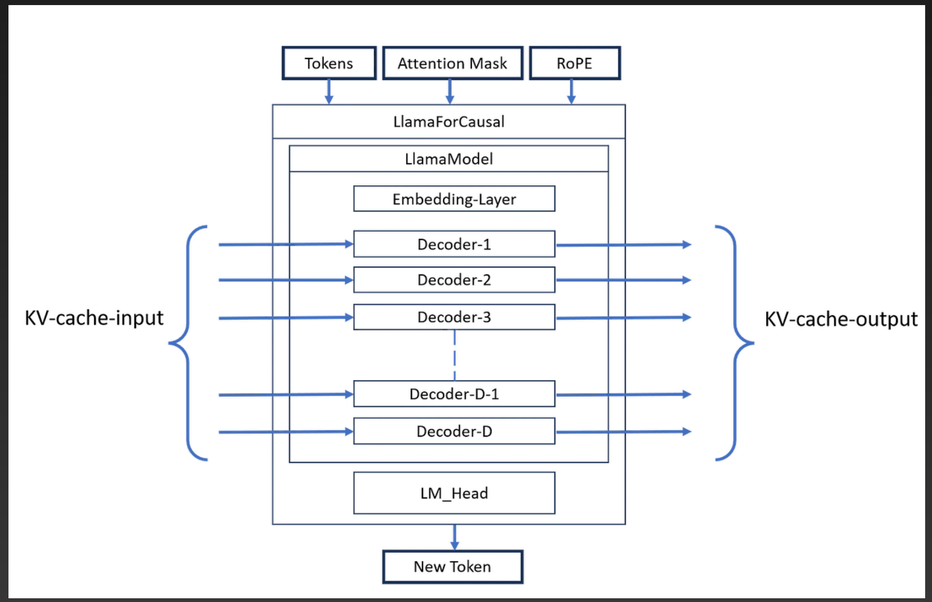


ICE #5

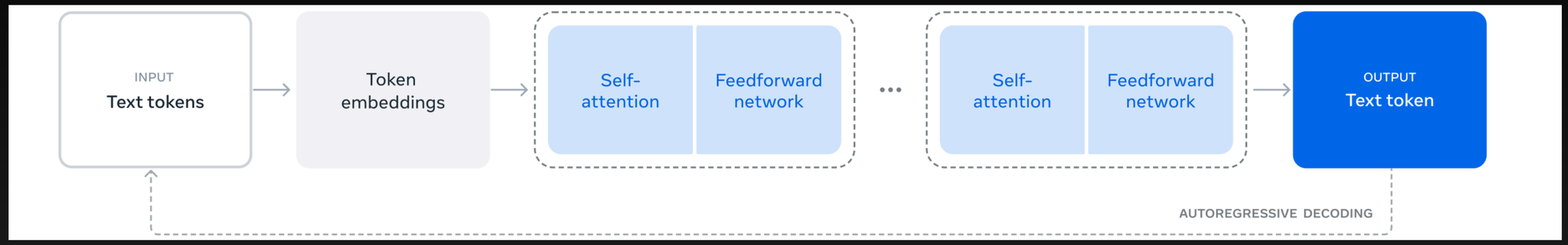


For Llama3-8b, with a key/value embedding dimension of 128 per head (4k in total) and 32 attention heads per layer and 32 layers and context window of 8k - What's the max RAM usage by the kv cache? Assume FP16 representation

a) 1 GB
b) 2 GB
c) 4 GB
d) 8 GB



KV Cache Efficiency



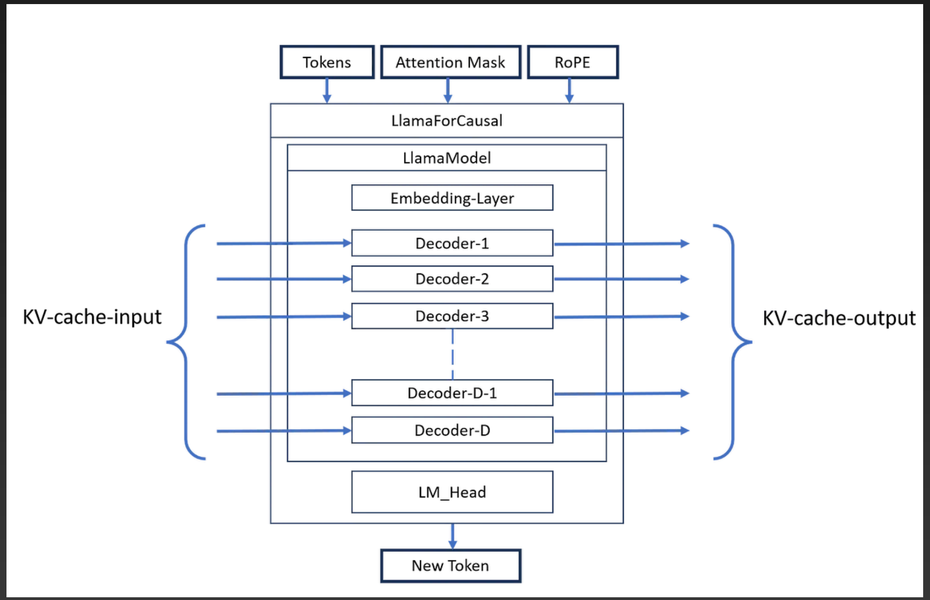
Assume “Quick Brown Fox jumped ....”
T1 t2 t3 t4
Need to predict t5

Without KV-cache

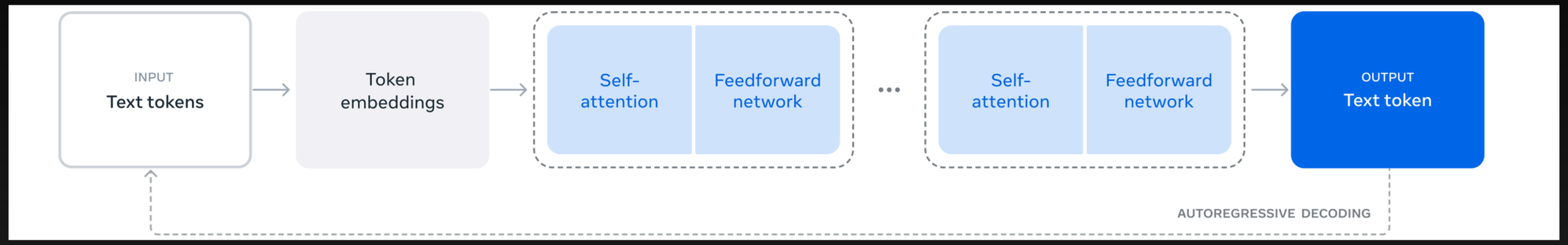
Compute K1,K2,K3,K4 and V1,V2,V3,V4 at each layer! And then finally with the last hidden representation of token 4, h4 -> Pass into the LM_head to get the new prediction t5

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

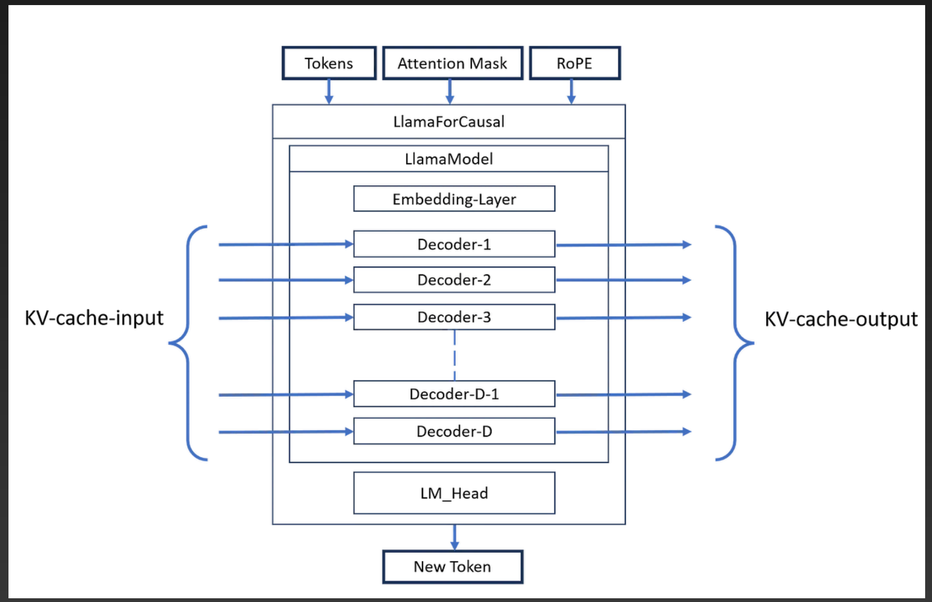


ICE #6: KV Cache Efficiency

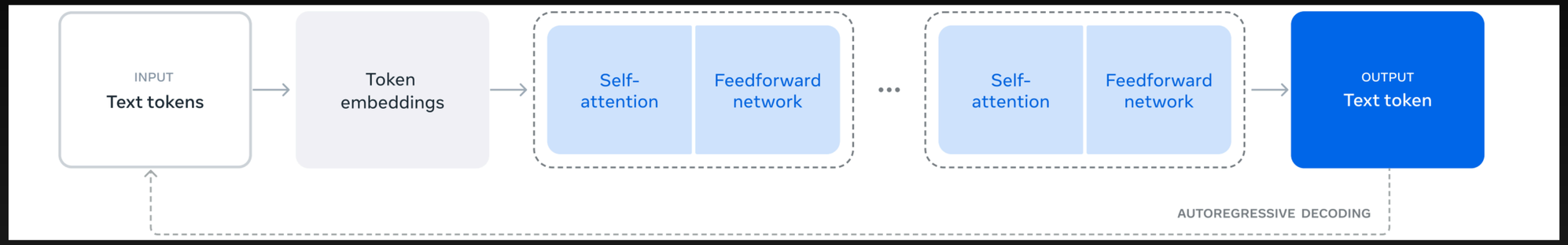


What is the compute complexity with/without kv-cache per layer in terms of the sequence length, N

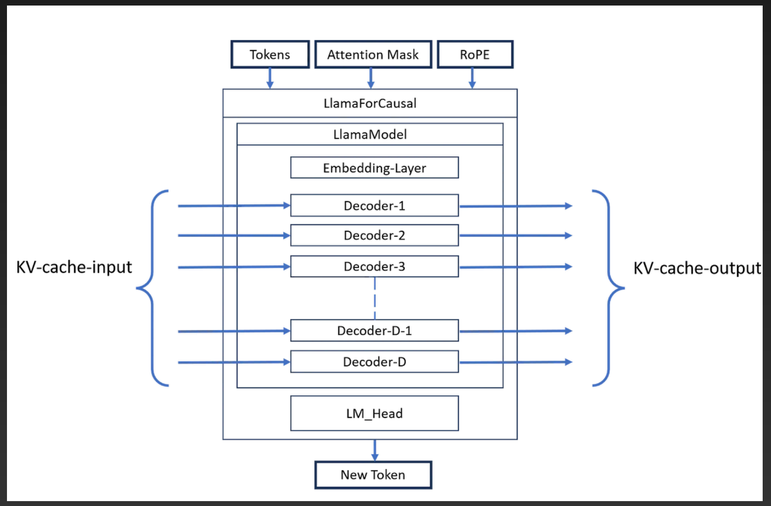
a) O(N), O(N)
b) O(N), O(N^2)
c) O(N^2), O(N^2)
d) O(log(N)), O(N^2)



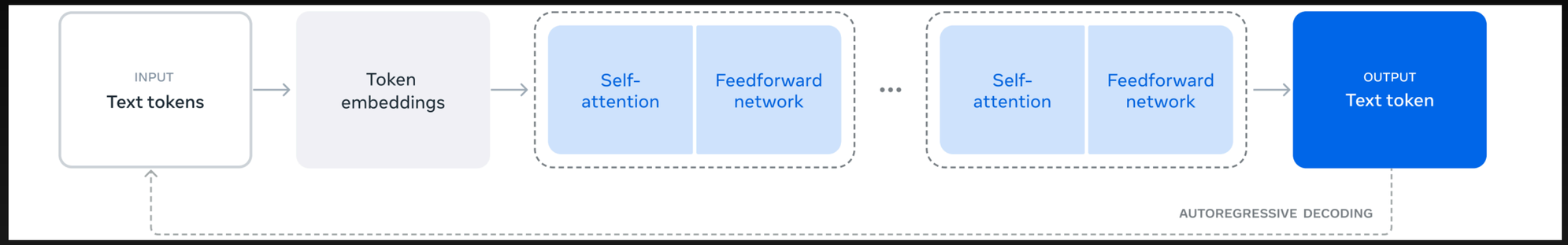
LM head



Takes the last hidden token from the last decoder layer, D of the llama arch and predicts the next token

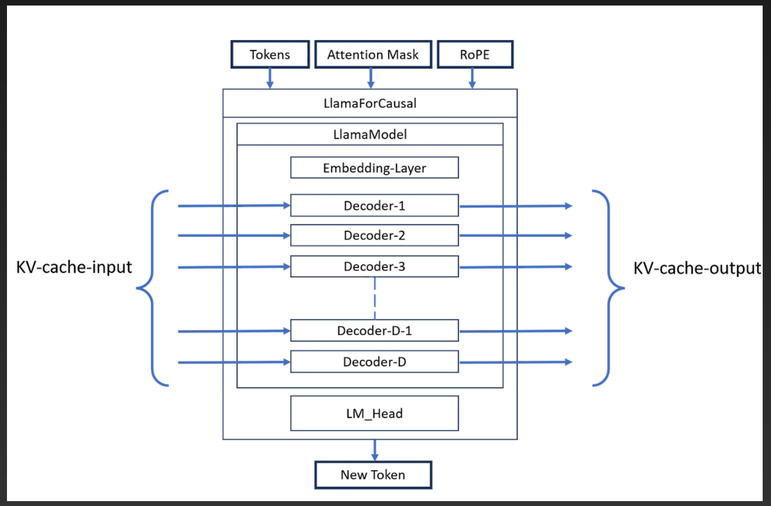


ICE #7 | LM head

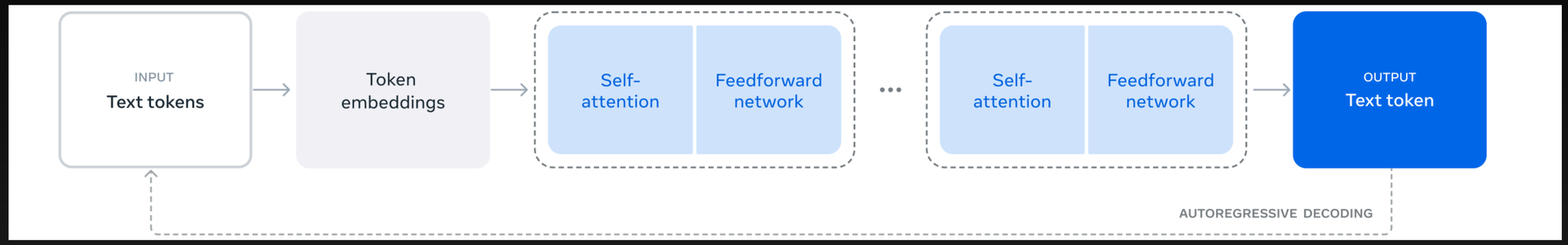


What is the composition of LM head?

a) Linear transformation + softmax activation
b) Non-linear transformation + relu activation
c) Non-linear transformation + softmax activation
d) Linear transformation + relu activation



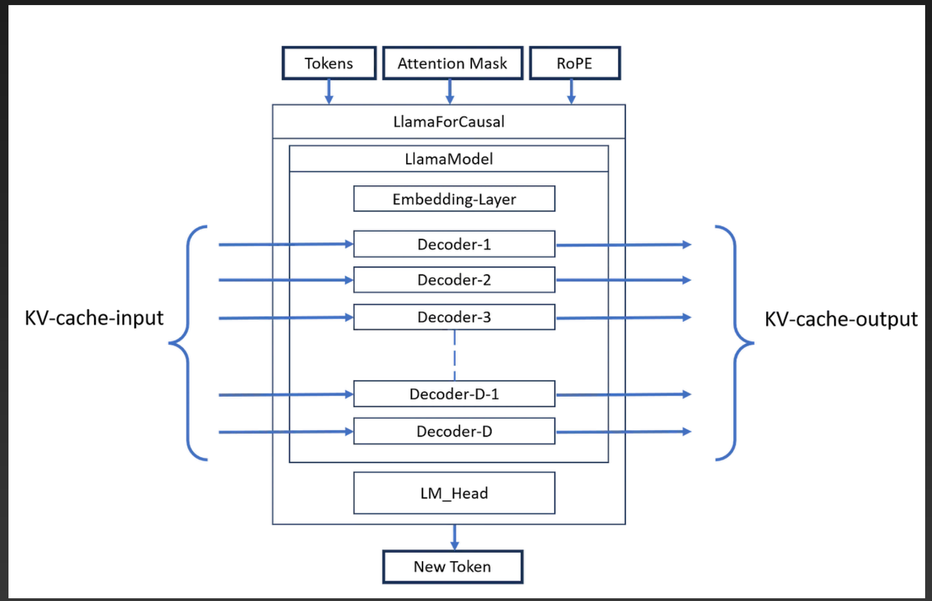
Inference phases



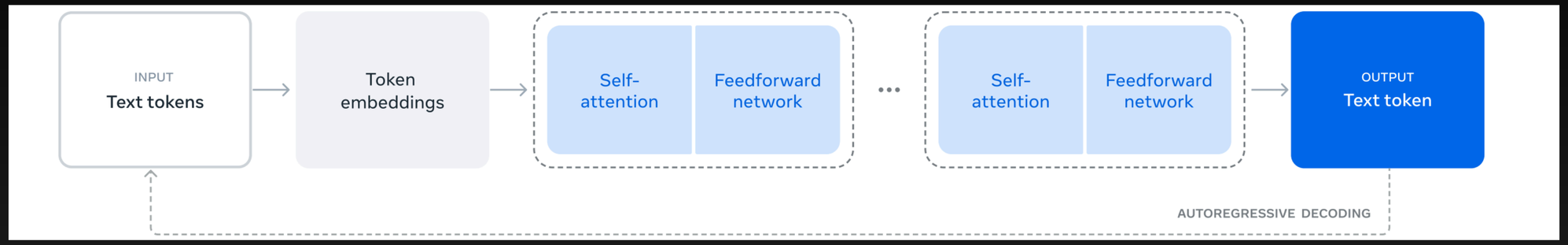
Assume “Quick Brown Fox jumped ….”
T1 t2 t3 t4
Need to predict t5

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



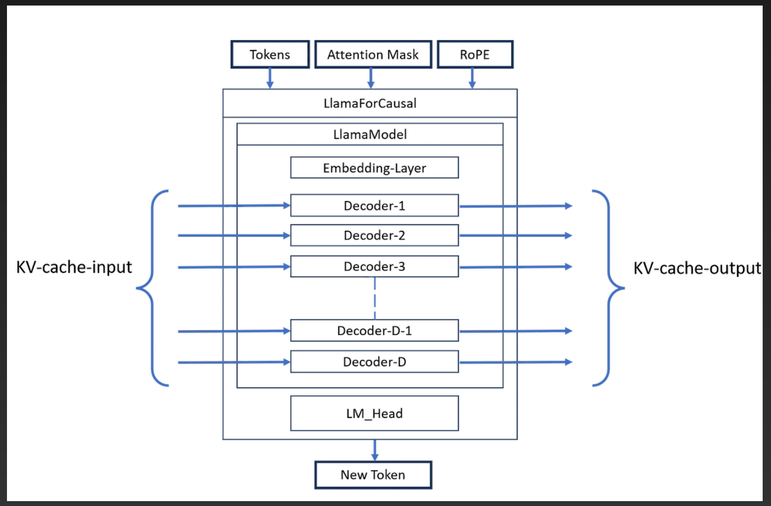
Parsing Inference Metrics



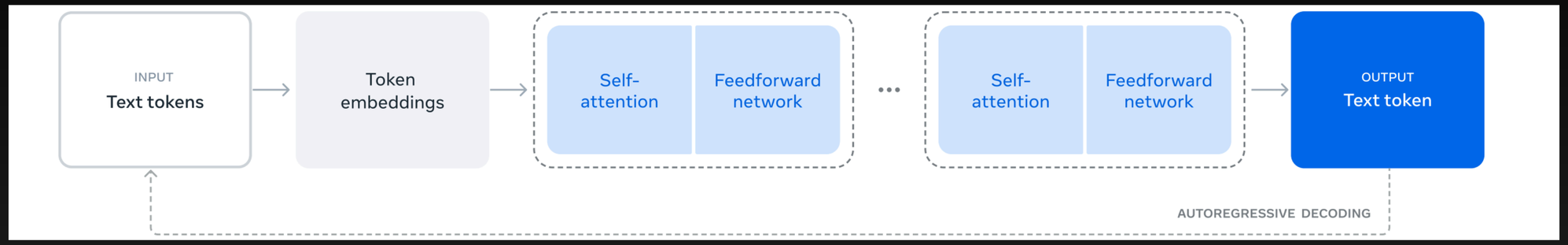
Assume “Quick Brown Fox jumped ….”
T1 t2 t3 t4
Need to predict t5

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