

GenAI Basics

CSE585

Insu Jang

August 29, 2024



Insu Jang

- Your GSI, Only GSI
- <https://insujang.github.io>

Better Tomorrow with Computer Science

About Research Posts

Posts

2024

Flash Attention

Jan 21, 2024

dl parallelism

Tensor Parallelism and Sequence Parallelism: Detailed Analysis

Jan 11, 2024

dl parallelism

LLM Inference: Continuous Batching and PagedAttention

Jan 7, 2024

dl inference attention

LLM Inference: Autoregressive Generation and Attention KV Cache

Jan 7, 2024

dl inference attention

Agenda

- Transformer architecture
- Transformer-based large language models
- Characteristics of LLMs

Transformer Architecture

- Introduced in: [Attention is All You Need \[NIPS'17\]](#)

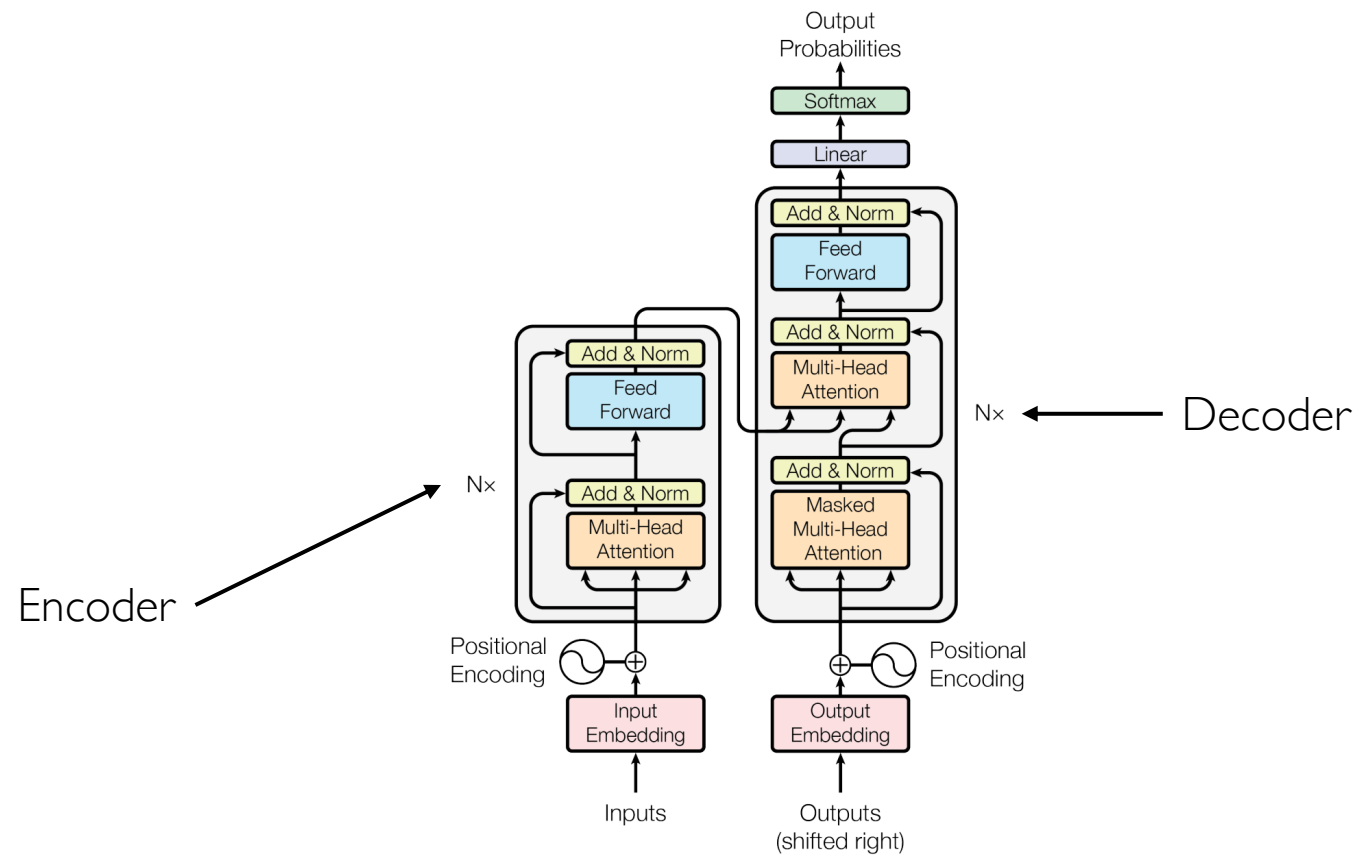
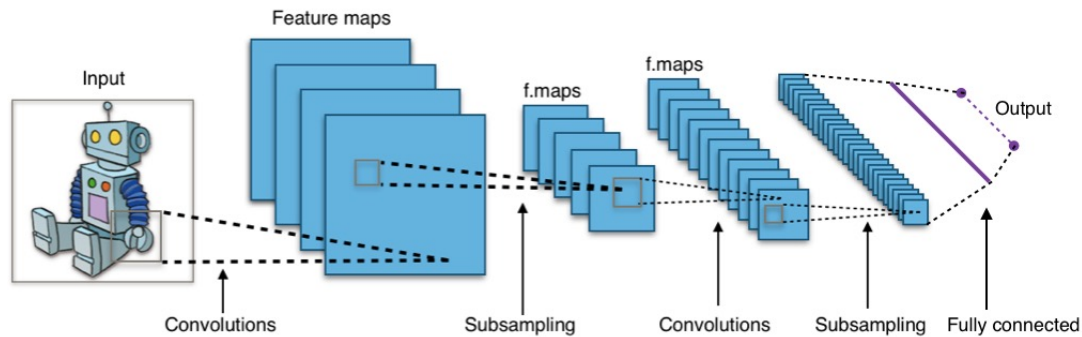


Figure 1: The Transformer - model architecture.

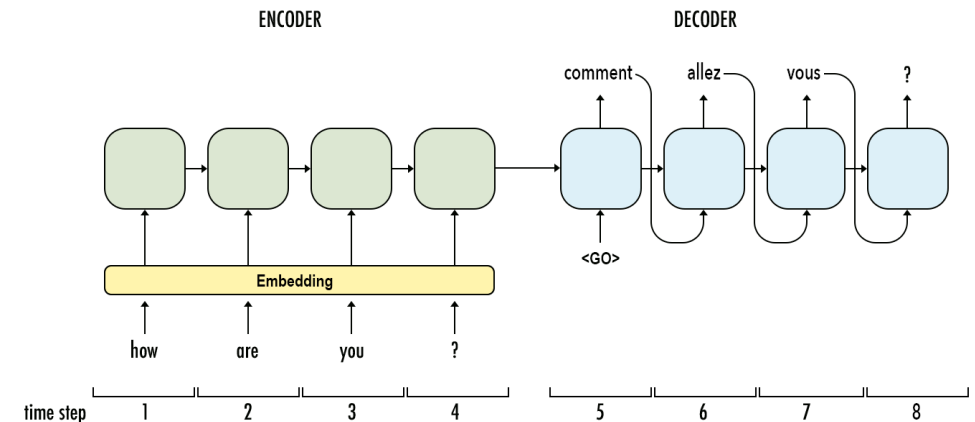
Why Transformer Was Introduced

- Before Transformer era

Computer Vision: Convolutional Neural Networks (CNN)



Natural Language Processing: Recurrent Neural Networks (RNN)



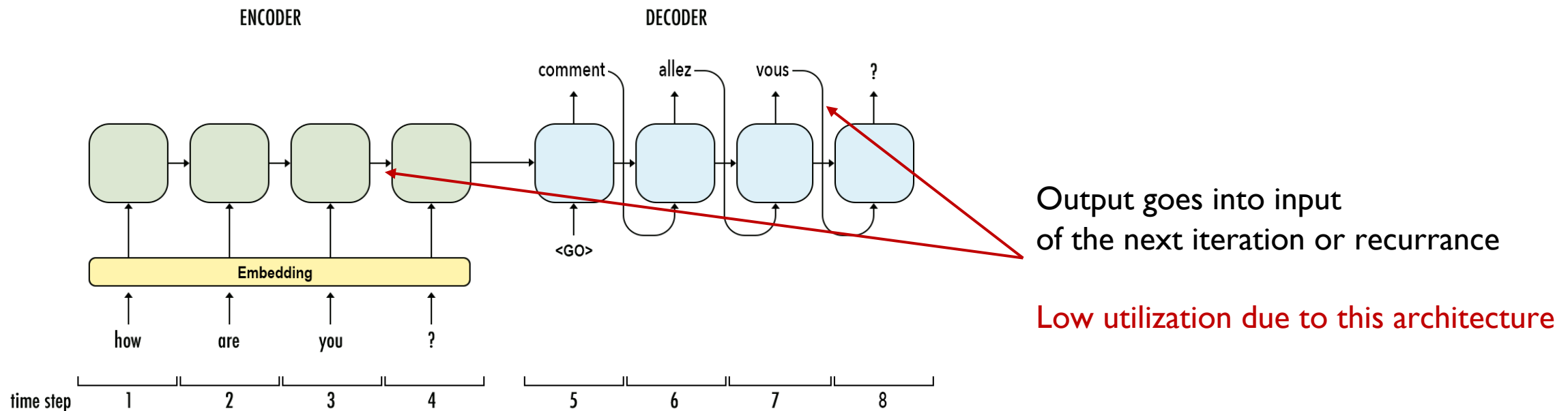
CNN image: https://en.wikipedia.org/wiki/Convolutional_neural_network

RNN image: <https://classroom.udacity.com/nanodegrees/nd101/parts/4f636f4e-f9e8-4d52-931f-a49a0c26b710/modules/c1558ffb-9afd-48fa-bf12-b8f29dcb18b0/lessons/43ccf91e-7055-4833-8acc-0e2cf77696e8/concepts/f999d8f6-b4c1-4cd0-811e-4767b127ae50>

modules/c1558ffb-9afd-48fa-bf12-b8f29dcb18b0/lessons/43ccf91e-7055-4833-8acc-0e2cf77696e8/concepts/f999d8f6-b4c1-4cd0-811e-4767b127ae50

Why Transformer Was Introduced

Natural Language Processing: Recurrent Neural Networks (RNN)



Why Transformer Was Introduced

Natural Language Processing: Recurrent Neural Networks (RNN)

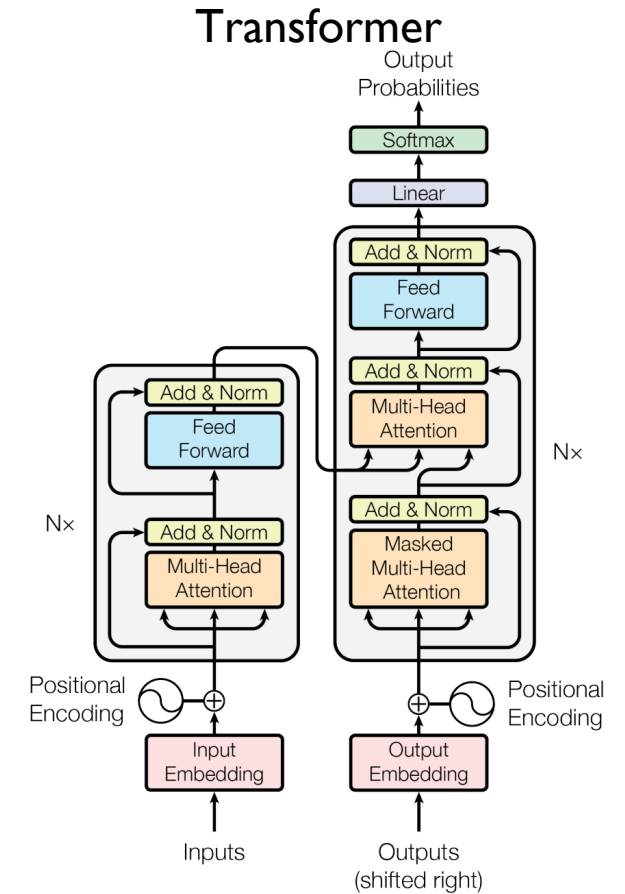
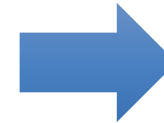
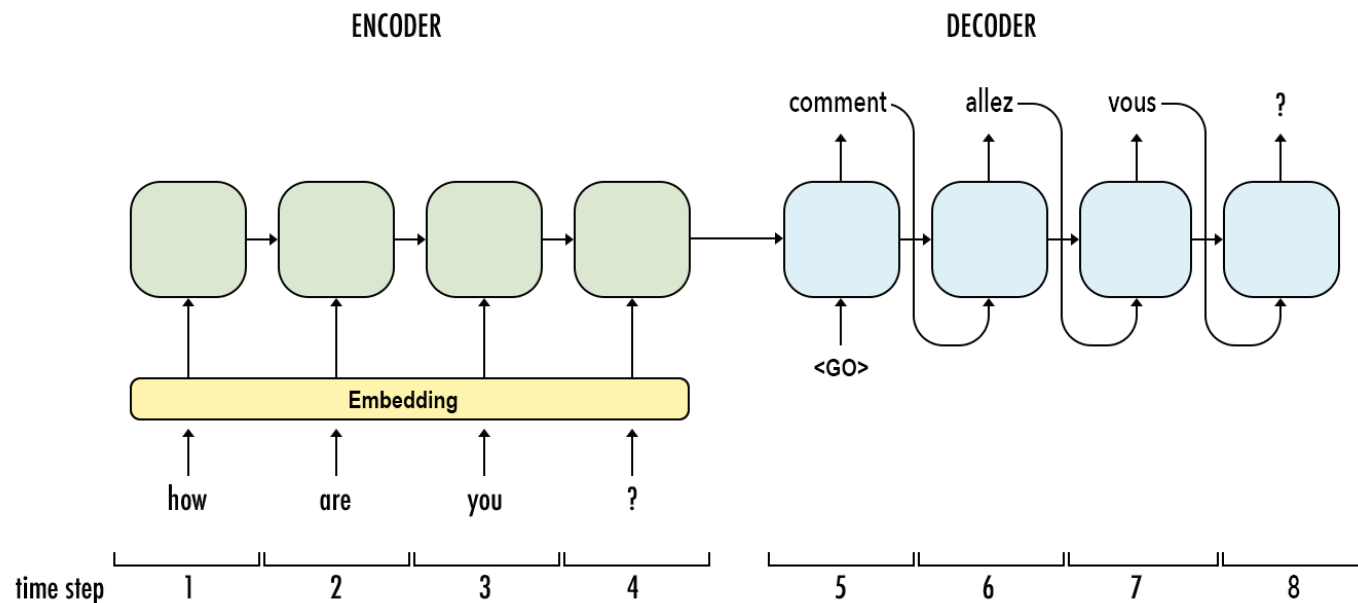


Figure 1: The Transformer - model architecture.

Attention is All You Need

- Avoid sequence translation → parallelizable and scalable
- Long term memory

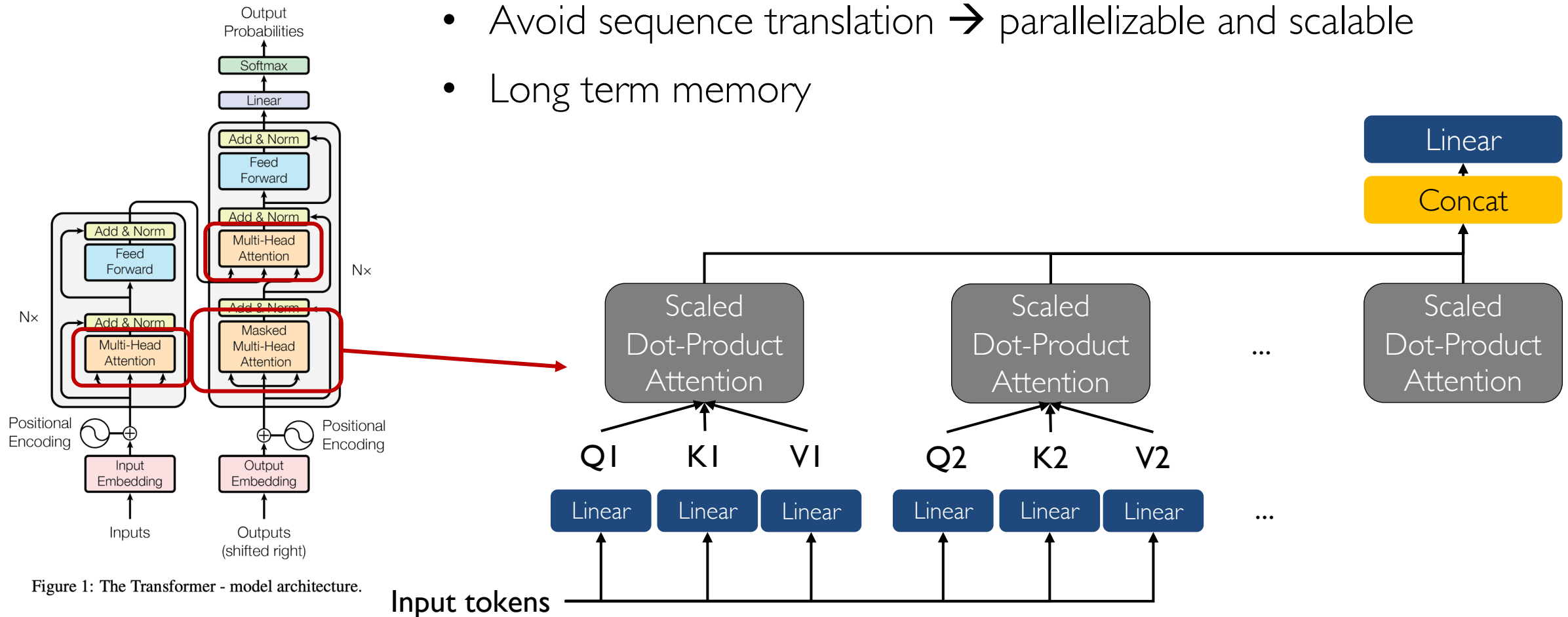


Figure 1: The Transformer - model architecture.

Input tokens

Attention is All You Need

- Avoid sequence translation → parallelizable and scalable
- Long term memory

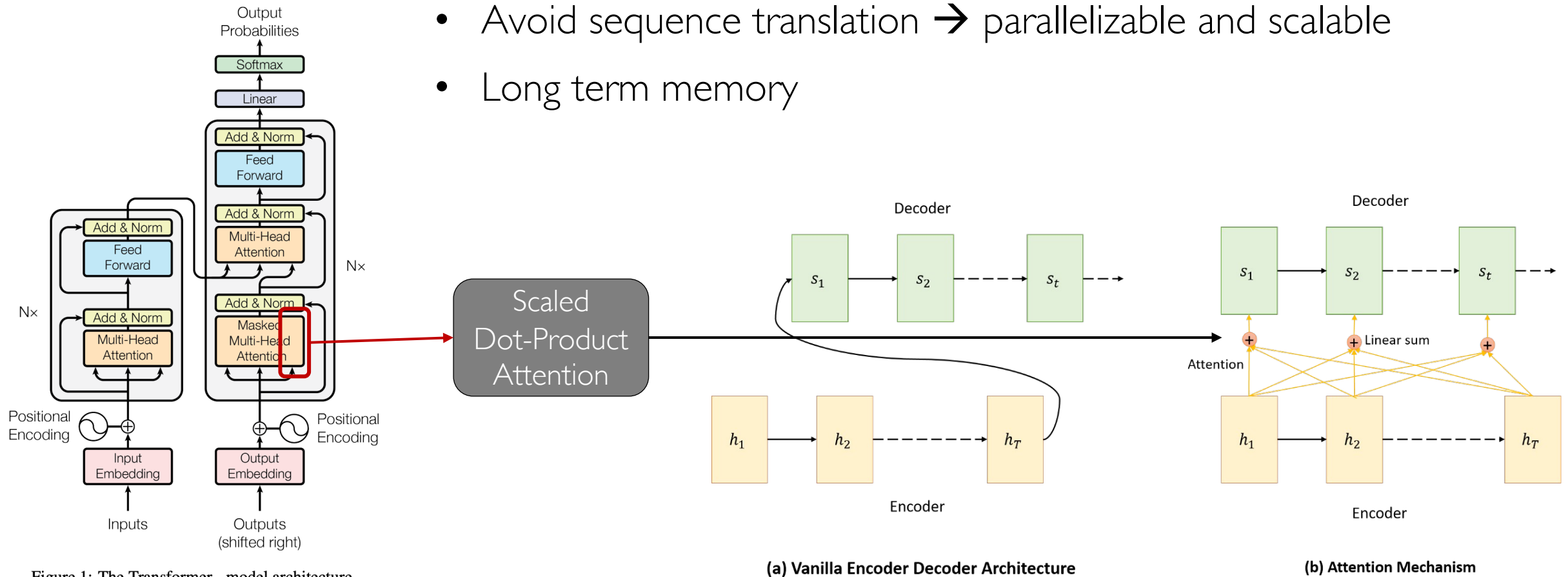


Figure 1: The Transformer - model architecture.

Attention is All You Need

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Input tokens

the		Q1, K1, V1
train	Q _{weight}	Q2, K2, V2
left	K _{weight}	Q3, K3, V3
the	V _{weight}	Q4, K4, V4
station		Q5, K5, V5

Attention is All You Need

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Input tokens

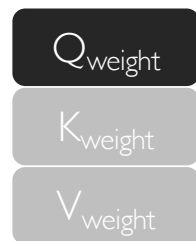
the

train

left

the

station



Q1, K1, V1

Q2, K2, V2

Q3, K3, V3

Q4, K4, V4

Q5, K5, V5

Attention is All You Need

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Input tokens

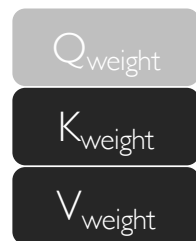
the

train

left

the

station



$Q_1, \mathbf{K}_1, \mathbf{V}_1$

$Q_2, \mathbf{K}_2, \mathbf{V}_2$

$Q_3, \mathbf{K}_3, \mathbf{V}_3$

$Q_4, \mathbf{K}_4, \mathbf{V}_4$

$Q_5, \mathbf{K}_5, \mathbf{V}_5$

Attention is All You Need

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Input tokens

K1, V1

K2, V2

K3, V3

K4, V4

K5, V5

the

Q1

Attention(Q1,
K1, V1)

Attention(Q1,
K2, V2)

Attention(Q1,
K3, V3)

Attention(Q1,
K4, V4)

Attention(Q1,
K5, V5)

train

Q2

left

Q3

the

Q4

station

Q5

$$\text{Attention}(\text{Token}_i, \text{Token}_k) = \text{softmax}\left(\frac{Q_i K_k^T}{\sqrt{d_k}}\right) V_k$$

Calculate relationship between this query token (i)
and all the other tokens (k)

Attention is All You Need

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Input tokens

K1, V1

K2, V2

K3, V3

K4, V4

K5, V5

the

Q1

Attention Score Vector for Q1 (the)

train

Q2

left

Q3

the

Q4

station

Q5

$$\text{Attention}(\text{Token}_i, \text{Token}_k) = \text{softmax}\left(\frac{Q_i K_k^T}{\sqrt{d_k}}\right) V_k$$

Calculate relationship between this query token (i) and all the other tokens (k)

Attention is All You Need

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Input tokens

the
train
left
the
station
on
time

	the	train	left	the	station	on	time
the	1.0	0.3	0.1	0.5	0.2	0.1	0.1
train	0.3	1.0	0.6	0.3	0.8	0.1	0.2
left	0.1	0.6	1.0	0.1	0.6	0.1	0.1
the	0.5	0.3	0.1	1.0	0.3	0.1	0.2
station	0.2	0.8	0.6	0.3	1.0	0.2	0.2
on	0.1	0.1	0.1	0.1	0.2	1.0	0.5
time	0.1	0.2	0.1	0.2	0.2	0.5	1.0

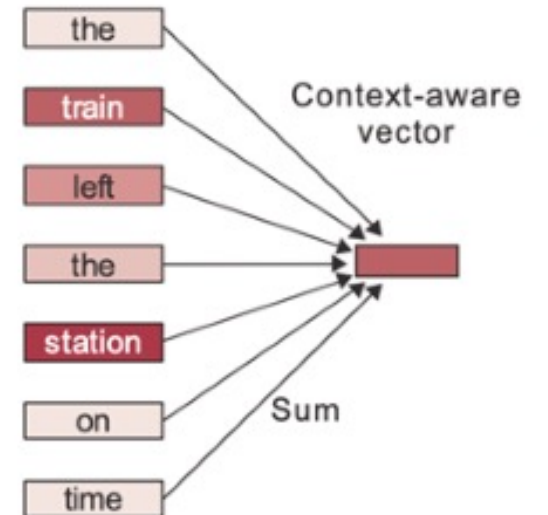
Attention scores

Scores for
"station"

0.2
0.8
0.6
0.3
1.0
0.2
0.2

Softmax,
scaling, and
multiplication

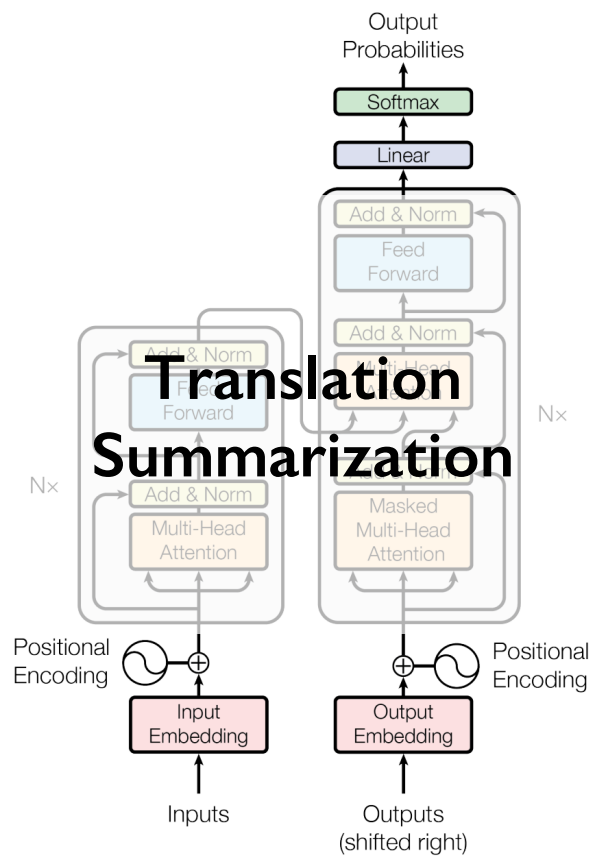
Weighted
token vectors



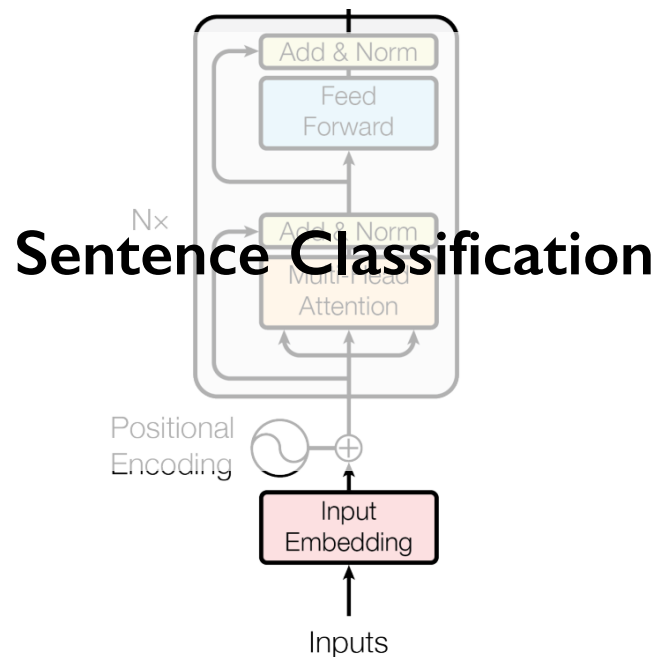
Transformer-based Large Language Models

Encoder-decoder
(or sequence-to-sequence)

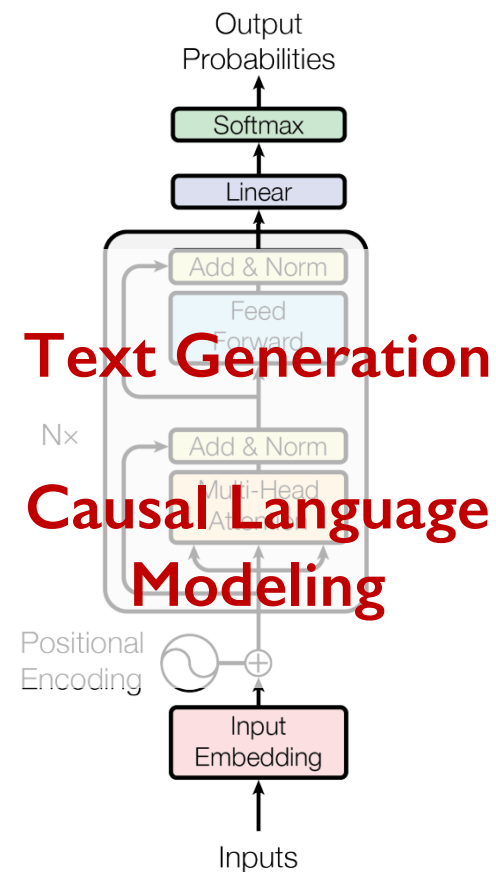
T5



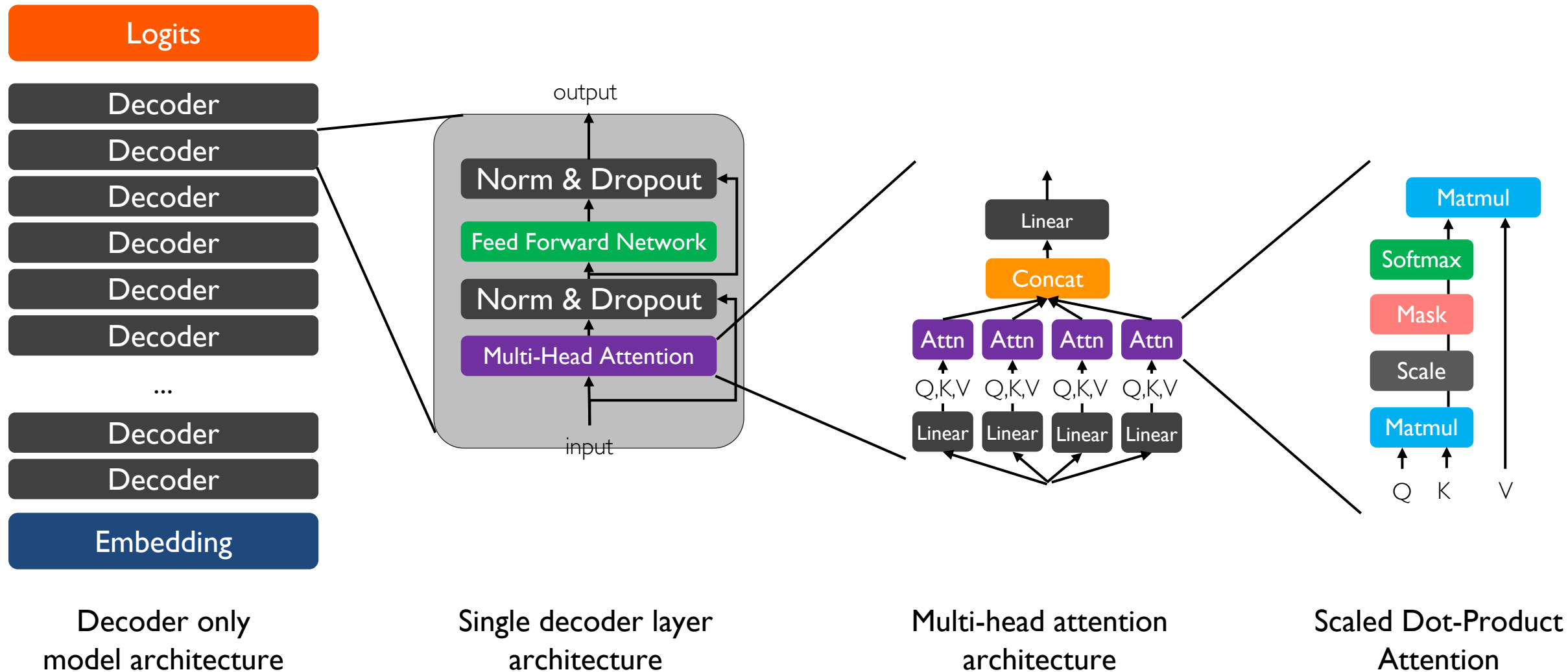
Encoder-only
BERT



Decoder-only
GPT



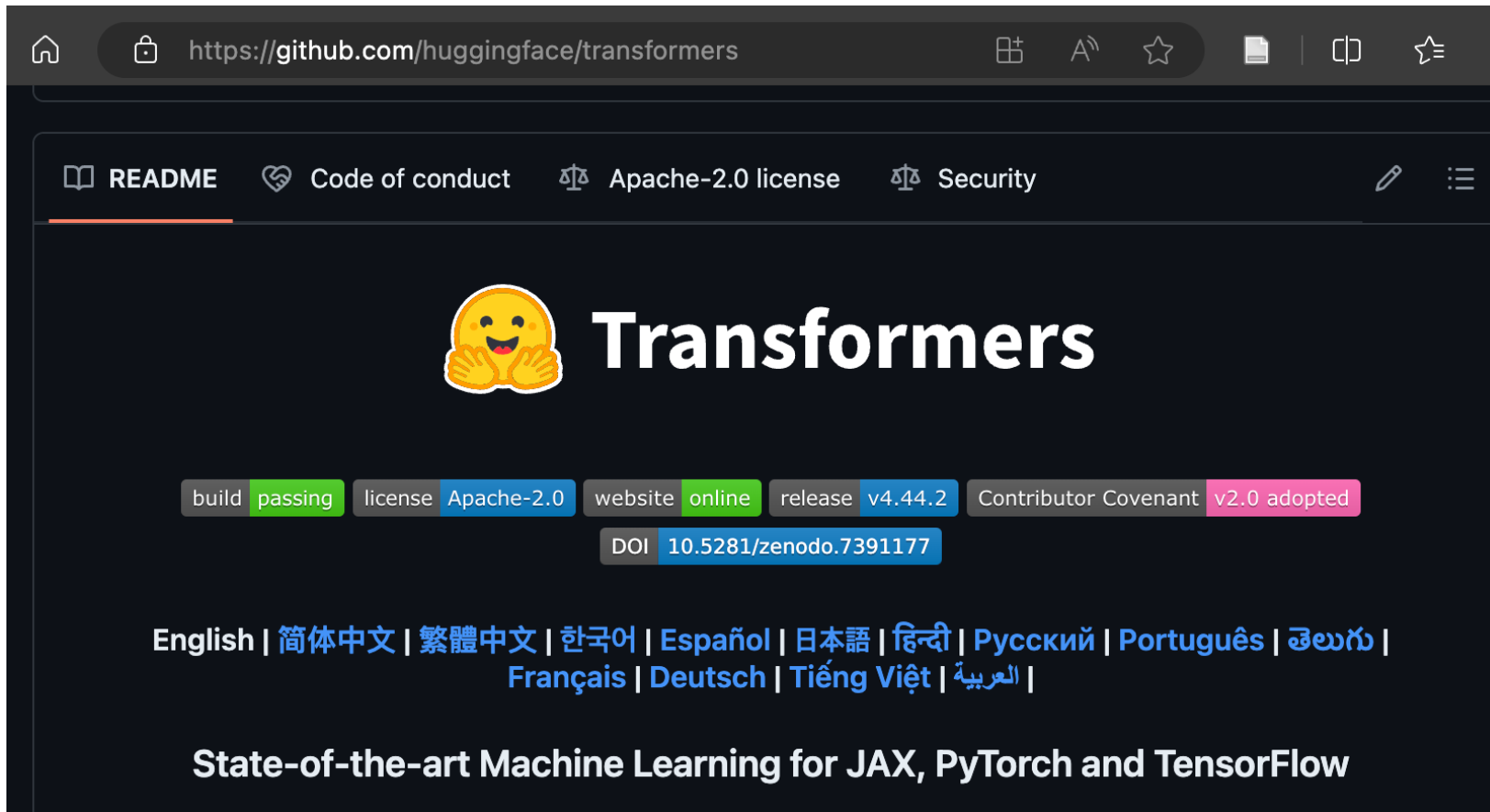
Transformer-based Large Language Models



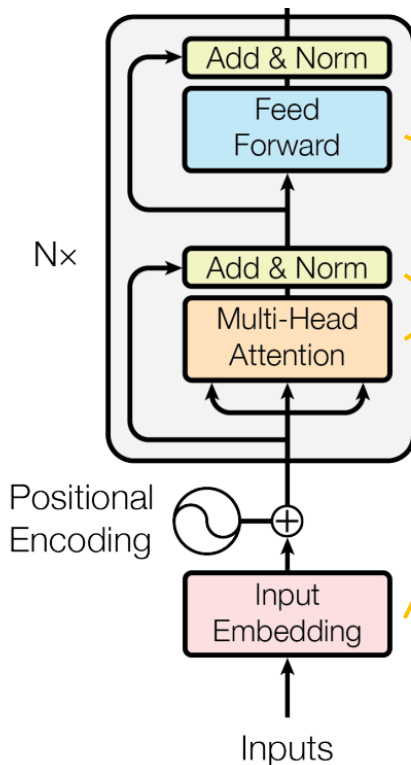
Let's See in the Code

Using 🤗 **Transformers** (<https://github.com/huggingface/transformers>)

A python library implemented by HuggingFace



Let's See in the Code



```
from transformers import LlamaForCausalLM

model = LlamaForCausalLM.from_pretrained("meta-llama/Meta-Llama-3.1-
8BInstruct")

LlamaForCausalLM(
  (model): LlamaModel(
    (embed_tokens): Embedding(128256, 4096)
    (layers): ModuleList(
      (0-31): 32 x LlamaDecoderLayer(
        (self_attn): LlamaSdpaAttention(
          (q_proj): Linear(in_features=4096, out_features=4096, bias=False)
          (k_proj): Linear(in_features=4096, out_features=1024, bias=False)
          (v_proj): Linear(in_features=4096, out_features=1024, bias=False)
          (o_proj): Linear(in_features=4096, out_features=4096, bias=False)
          (rotary_emb): LlamaRotaryEmbedding()
        )
        (mlp): LlamaMLP(
          (gate_proj): Linear(in_features=4096, out_features=14336, bias=False)
          (up_proj): Linear(in_features=4096, out_features=14336, bias=False)
          (down_proj): Linear(in_features=14336, out_features=4096, bias=False)
          (act_fn): SiLU()
        )
        (input_layernorm): LlamaRMSNorm((4096,), eps=1e-05)
        (post_attention_layernorm): LlamaRMSNorm((4096,), eps=1e-05)
      )
    )
    (norm): LlamaRMSNorm((4096,), eps=1e-05)
    (rotary_emb): LlamaRotaryEmbedding()
  )
  (lm_head): Linear(in_features=4096, out_features=128256, bias=False)
)
```

Let's See in the Code

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

```
class LlamaAttention(nn.Module):  
    def forward(  

```

```
        query_states = self.q_proj(hidden_states)  
        key_states = self.k_proj(hidden_states)  
        value_states = self.v_proj(hidden_states)
```

```
        attn_weights = torch.matmul(query_states, key_states.transpose(2, 3)) / math.sqrt(self.head_dim)
```

```
        attn_weights = nn.functional.softmax(attn_weights, dim=-1, dtype=torch.float32).to(query_states.dtype)  
        attn_weights = nn.functional.dropout(attn_weights, p=self.attention_dropout, training=self.training)  
        attn_output = torch.matmul(attn_weights, value_states)
```

Let's See in the Code

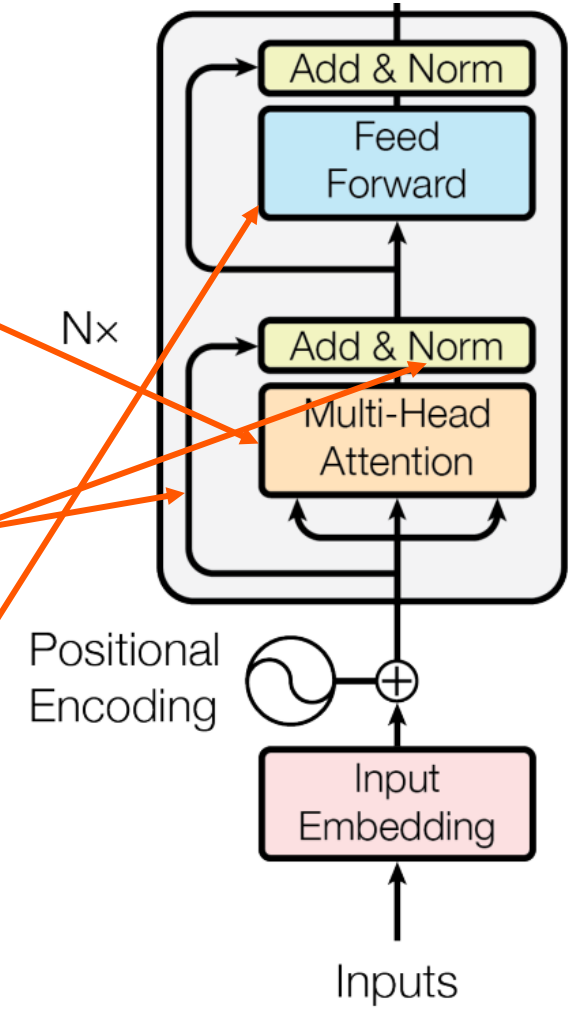
```
class LlamaDecoderLayer(nn.Module):
    def forward(

        # Self Attention
        hidden_states, self_attn_weights, present_key_value = self.self_attn(
            hidden_states=hidden_states,
            attention_mask=attention_mask,
            position_ids=position_ids,
            past_key_value=past_key_value,
            output_attentions=output_attentions,
            use_cache=use_cache,
            cache_position=cache_position,
            position_embeddings=position_embeddings,
            **kwargs,
        )
        hidden_states = residual + hidden_states

        # Fully Connected
        residual = hidden_states
        hidden_states = self.post_attention_layernorm(hidden_states)
        hidden_states = self.mlp(hidden_states)
        hidden_states = residual + hidden_states
```

```
class LlamaAttention(nn.Module):
    def forward(
```

```
class LlamaMLP(nn.Module):
    def forward(self, x):
```



Let's See in the Code

```
from transformers import LlamaTokenizerFast

tokenizer = LlamaTokenizerFast.from_pretrained("meta-llama/Meta-Llama-3.1-8B-Instruct")
input = "Hi there! Welcome to CSE 585: Advanced Scalable Systems for Generative AI!"
input_ids = tokenizer(input, return_tensors="pt").input_ids.to("cuda")
```

```
output = model.generate(input_ids, max_length=100, do_sample=True)
tokenizer.decode(output[0])
```

"<|begin_of_text|>Hi there! Welcome to CSE 585: Advanced Scalable Systems for Generative AI! This course is designed to help you master the skills needed to build, deploy, and manage large-scale generative AI systems. In this course, we'll cover the fundamental concepts, technologies, and best practices for building scalable and efficient generative AI systems.\nCourse Overview:\nIn this course, we'll cover the following topics:\n1. **Generative AI Fundamentals**: We'll start by covering the"

Let's See in the Code

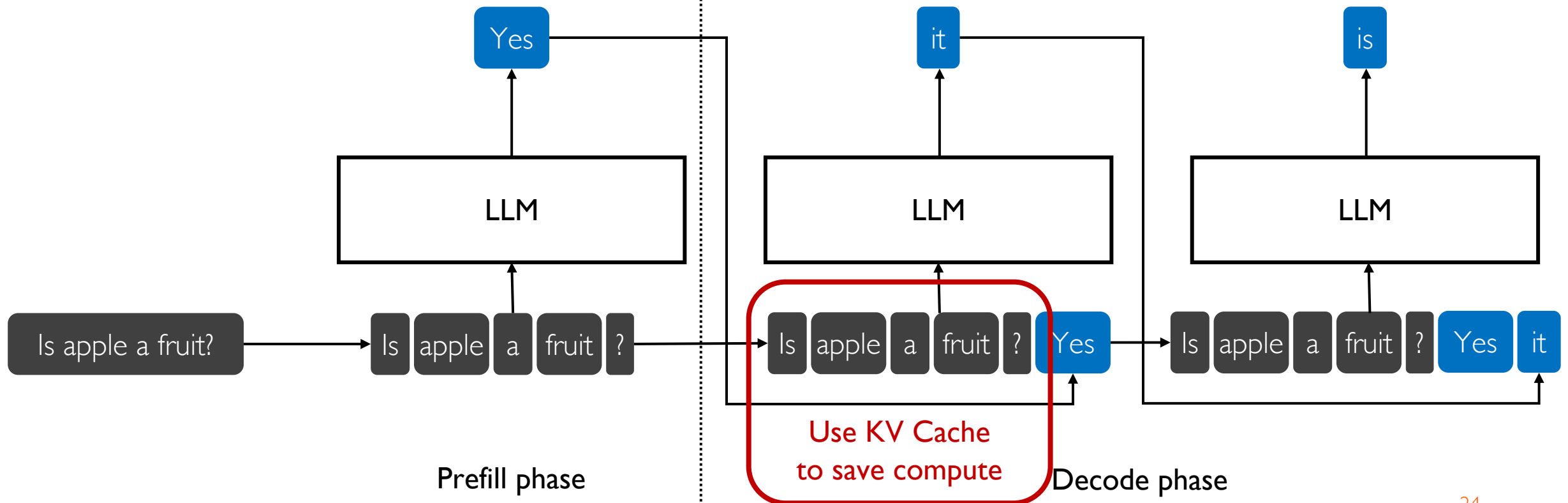
So many models and datasets available in HuggingFace hub (<https://huggingface.co>)

The screenshot shows the Hugging Face website interface. At the top, there's a navigation bar with the Hugging Face logo, a search bar, and links to Models, Datasets, Spaces, Posts, Docs, and Pricing. Below the navigation bar, the left sidebar contains filters for Tasks, Libraries, Datasets, Languages, and Licenses. The 'Tasks' section is expanded, showing categories like Multimodal and Computer Vision with various sub-tasks. The main content area displays a list of models under the 'Models' tab, sorted by trending. The models listed are:

- black-forest-labs/FLUX.1-dev**: Text-to-Image, Updated 10 days ago, 507k downloads, 3.21k likes.
- microsoft/Phi-3.5-MoE-instruct**: Text Generation, Updated 6 days ago, 12.5k downloads, 391 likes.
- microsoft/Phi-3.5-mini-instruct**: Text Generation, Updated 4 days ago, 61.1k downloads, 340 likes.
- black-forest-labs/FLUX.1-schnell**: Text-to-Image, Updated 10 days ago, 1.58M downloads, 1.89k likes.
- microsoft/Phi-3.5-vision-instruct**: Text Generation, Updated 3 days ago, 18.7k downloads, 304 likes.

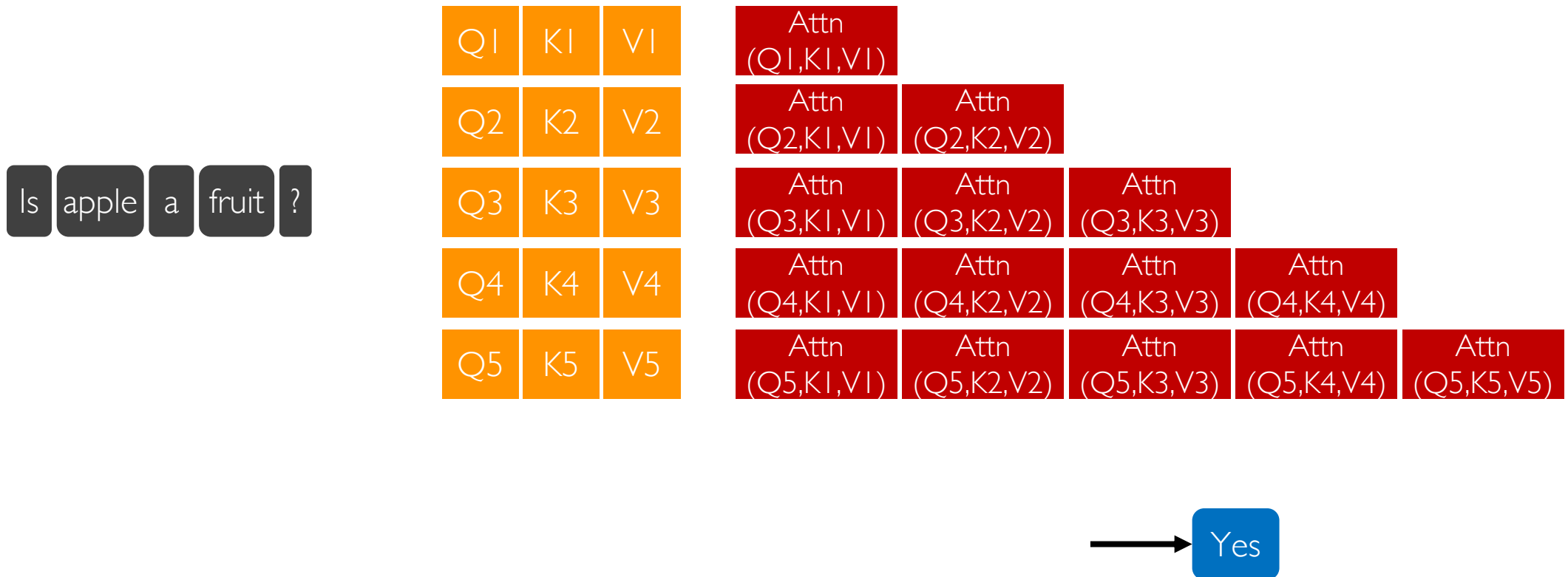
Characteristics of Transformer LLMs

- **Autoregressive decoding:** only one token is generated at a time
- Two phases have different characteristics



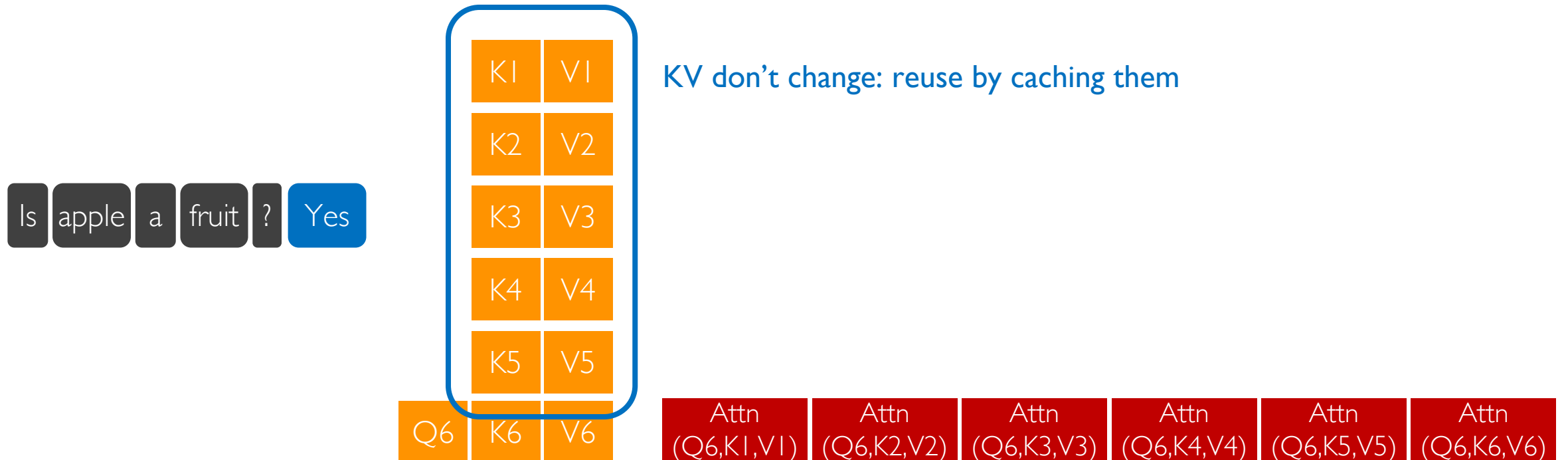
Characteristics of Transformer LLMs

- **KV Cache** to save compute in inference



Characteristics of Transformer LLMs

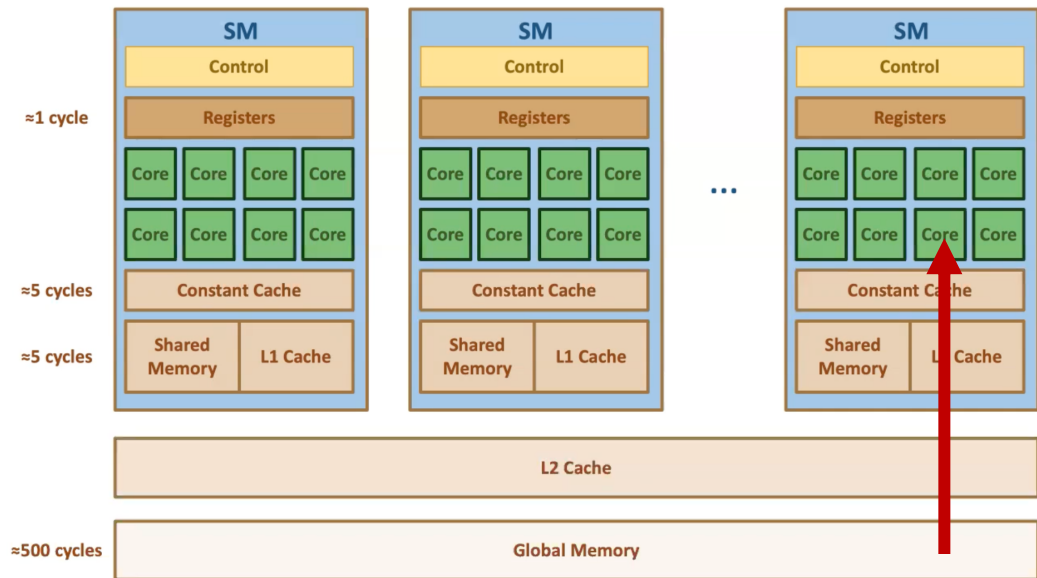
- **KV Cache** to save compute in inference



Characteristics of Transformer LLMs

- **Different arithmetic intensity** in prefill vs decode

Memory in the GPU Architecture

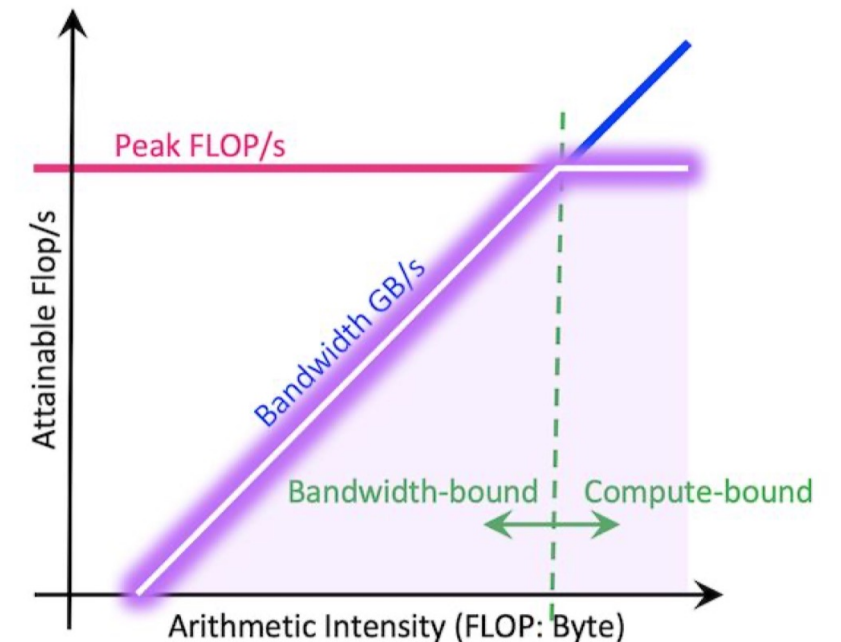


Slide credit: Izzat El Hajj

24

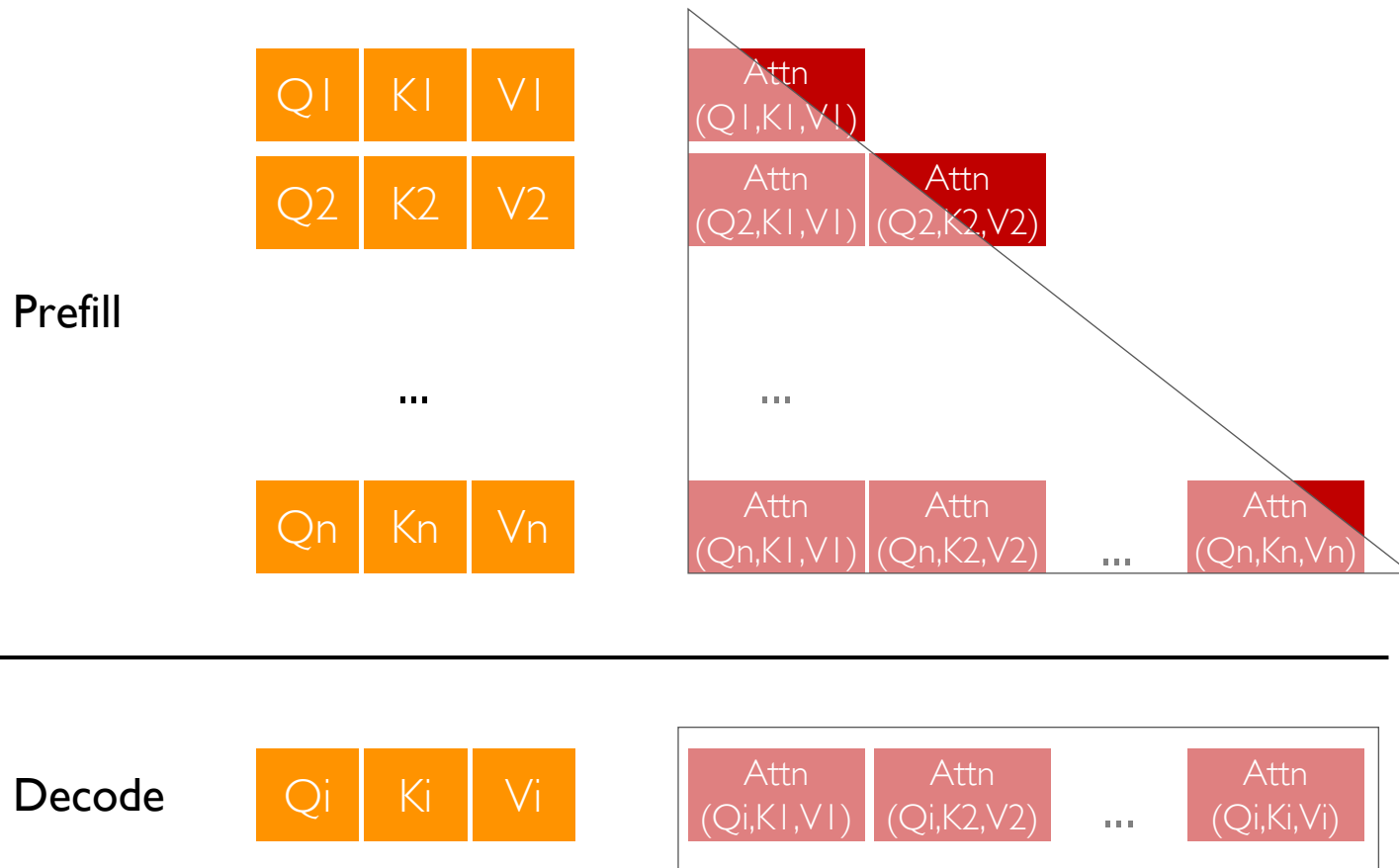
Arithmetic intensity:

compute operations / # byte accesses



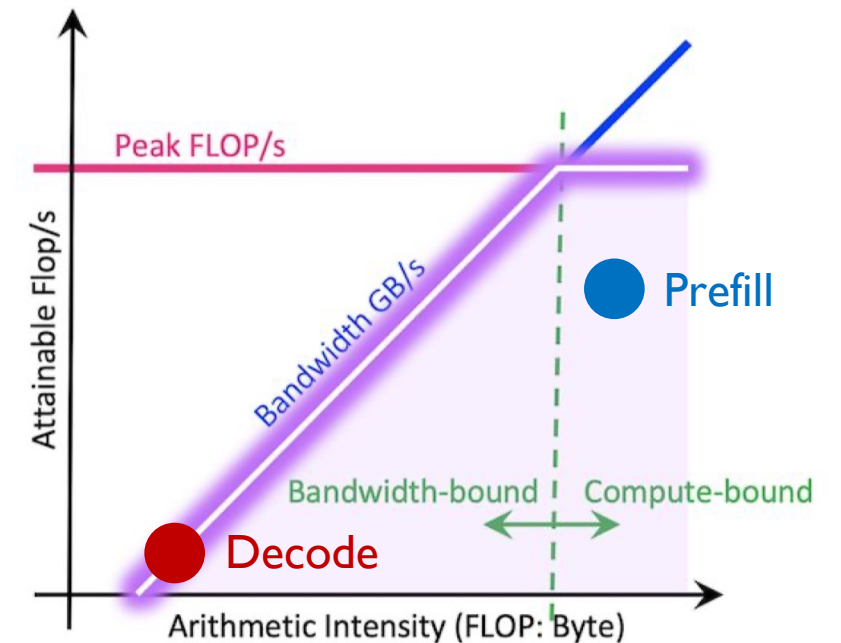
Characteristics of Transformer LLMs

- **Different arithmetic intensity** in prefill vs decode



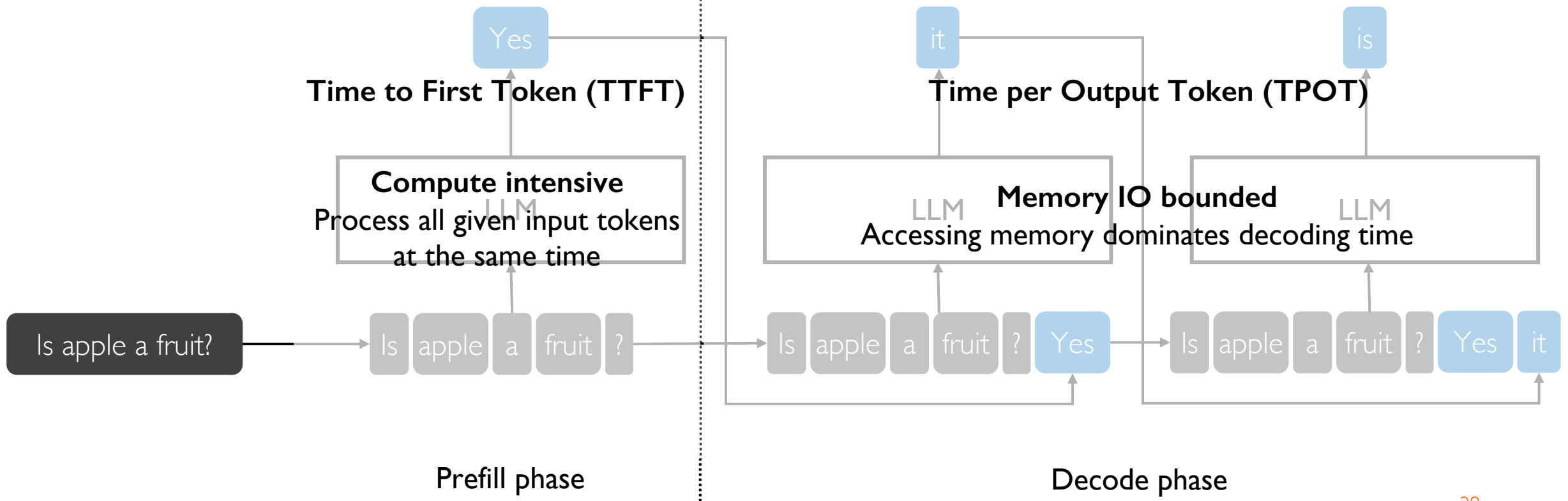
Arithmetic intensity:

compute operations / # byte accesses



Characteristics of Transformer LLMs

- **Autoregressive decoding:** only one token is generated at a time
- Two phases have different characteristics



Characteristics of Transformer LLMs

- Huge model size

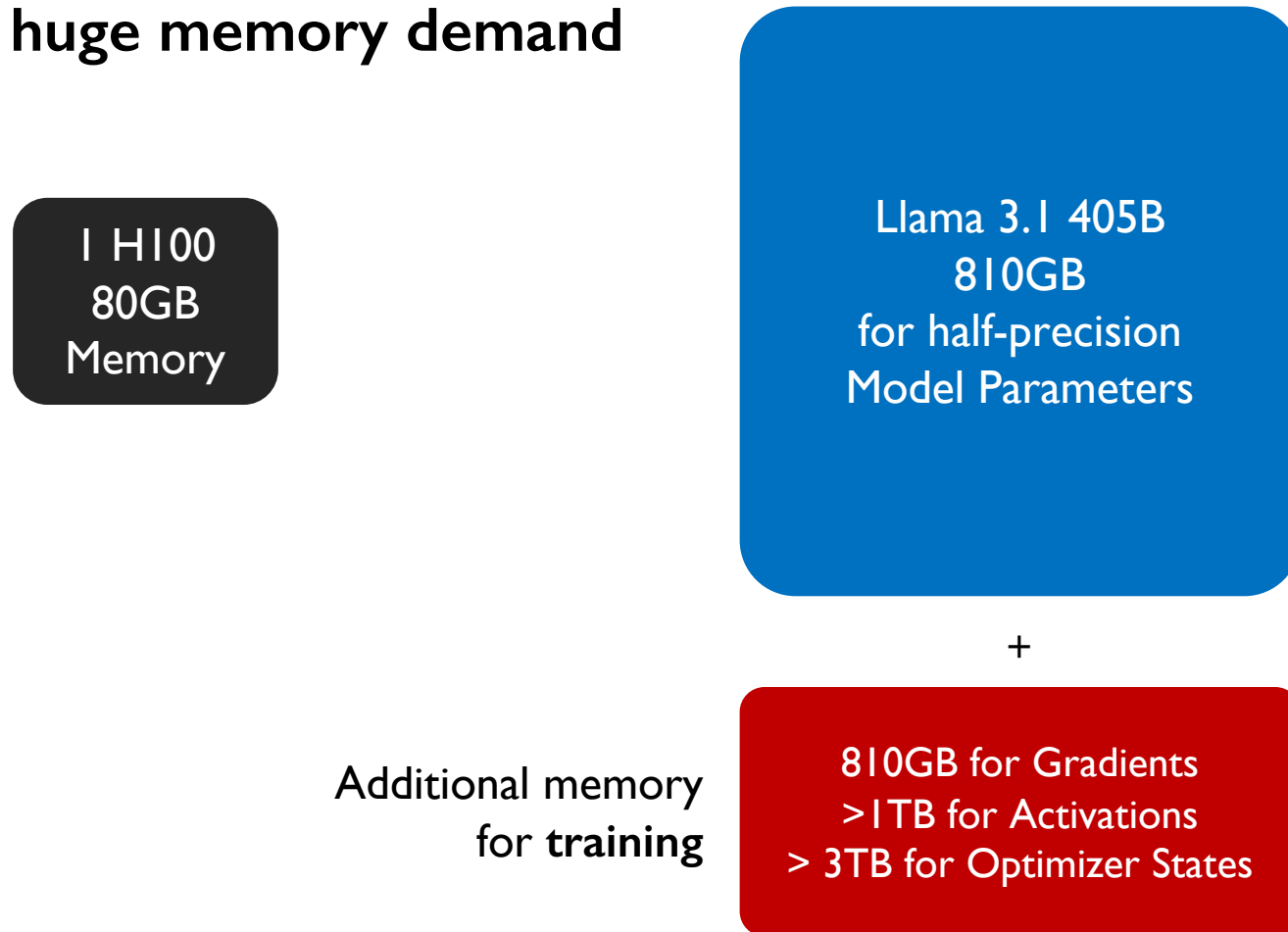


1 H100
80GB
Memory

Llama 3.1 405B
810GB
for half-precision
Model Parameters

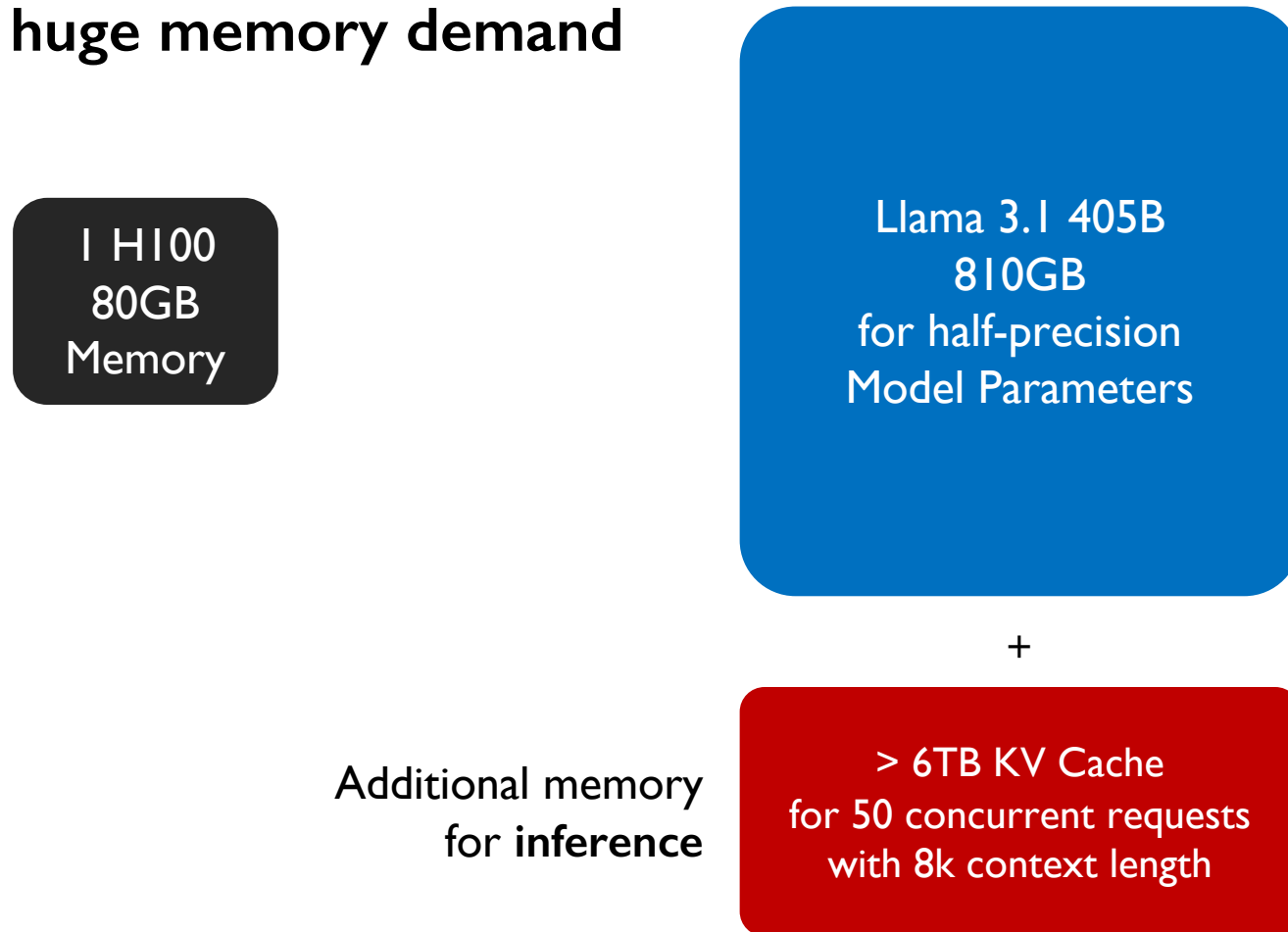
Characteristics of Transformer LLMs

- Huge model size + **huge memory demand**



Characteristics of Transformer LLMs

- Huge model size + **huge memory demand**



System Designs for LLMs

- Distributed ML



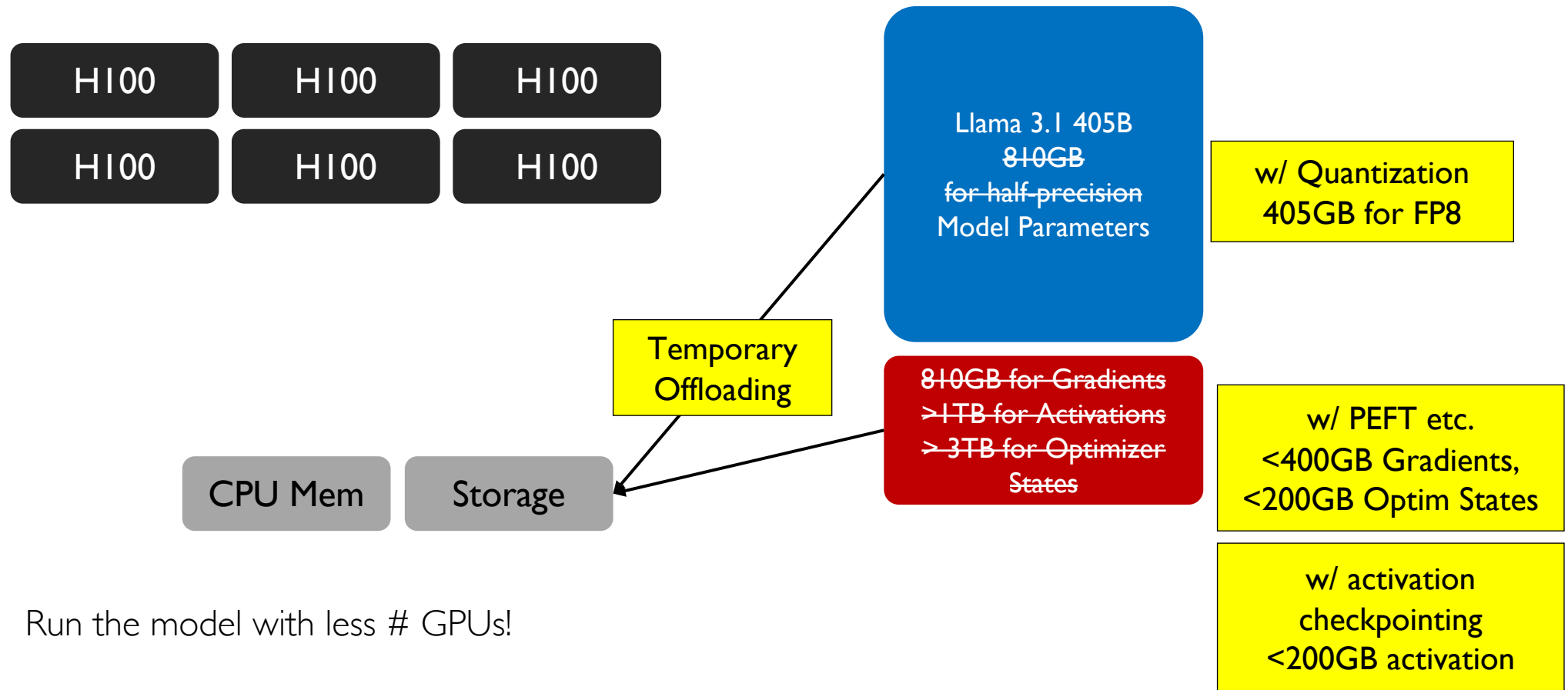
For Llama 3.1 405B: 16k H100s are used to train

Llama 3.1 405B
810GB
for half-precision
Model Parameters

810GB for Gradients
>1TB for Activations
> 3TB for Optimizer States

System Designs for LLMs

- Memory efficient ML

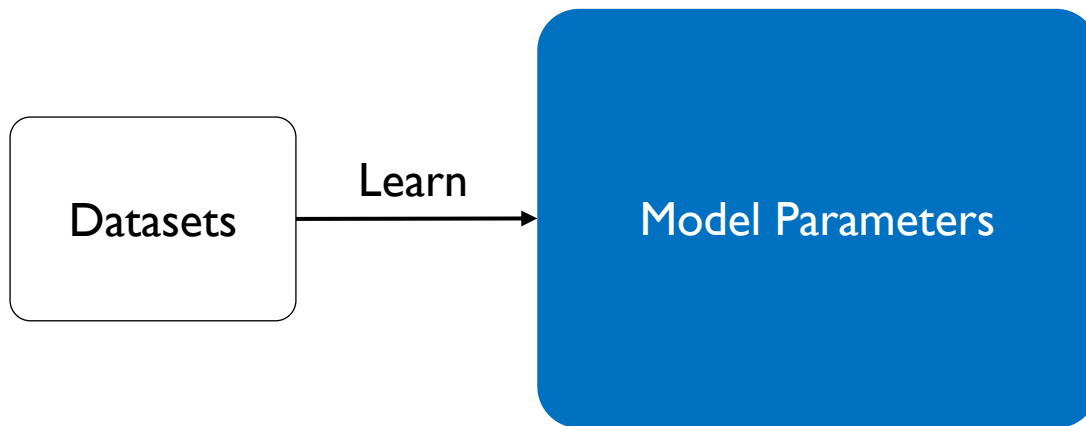


Run the model with less # GPUs!

Characteristics of Transformer LLMs

- Outdated Knowledge & hallucinations

Knowledge cutoff: model doesn't know later information
Generate "plausible sounding but factually incorrect response"



Model: GPT-4

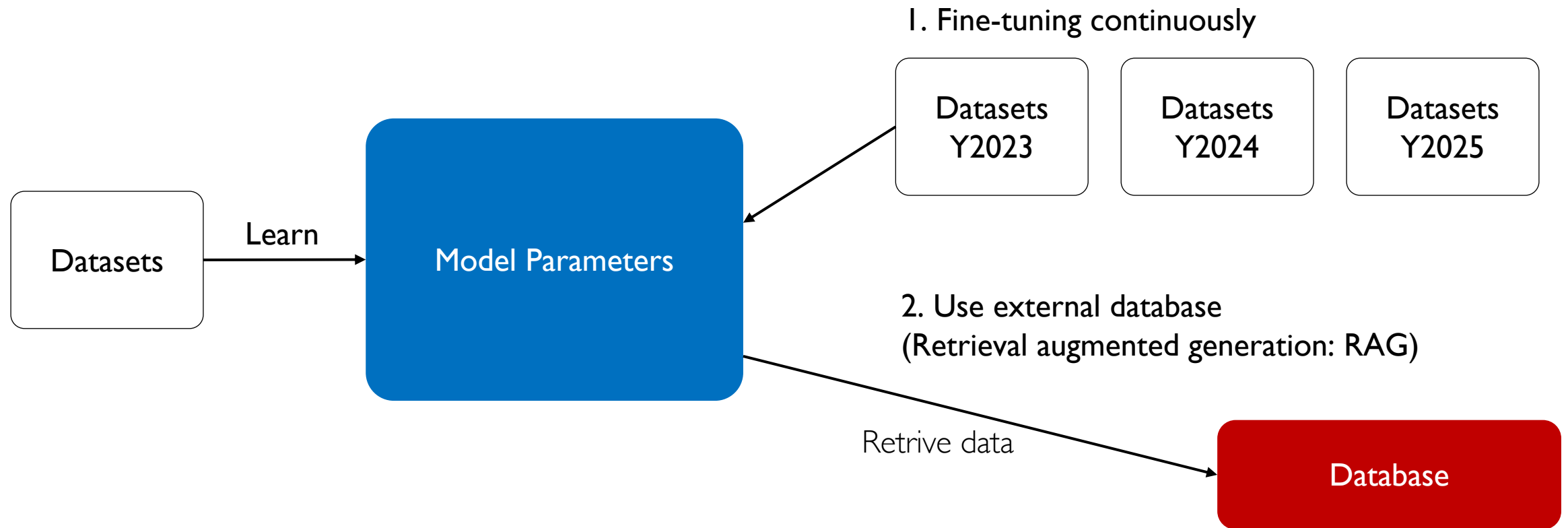
JE Which review papers discuss challenges and applications of large language models?

As of my knowledge cutoff in September 2021, there were several influential papers discussing the challenges and applications of large language models. Here are some key resources that were available at that time:

1. "Language Models are Few-Shot Learners" by Tom B. Brown, Benjamin Mann, Nick Ryder, et al., 2020. This is not exactly a review paper, but it's a seminal work by OpenAI that discusses the potential and challenges of large-scale language models. ✓ Correct!
2. "The Power and Limitations of Language Models" by Jacob Devlin, 2021. Although this is a blog post, it provides a good summary of the current understanding of the power and limitations of large language models, especially the Transformer-based models like BERT and GPT-3. ✗ Does not exist!
3. "A Survey of Transformers" by Tianyu Liu, Xiang Li, Zhengyang Liu, et al., 2021. This is a survey paper that reviews the Transformer architecture, which is the basis for large language models such as GPT-3. It discusses various applications and challenges of these models. ✗ Wrong authors!

Characteristics of Transformer LLMs

- Outdated Knowledge & hallucinations: mitigations



GenAI != Transformer

- Transformer architecture is dominating text generation as of now
- Diffusion models: generate data by denoising diffusion
- State space models: improve Transformer's computational inefficiency
- and more

Denoising diffusion models

● Forward / noising process

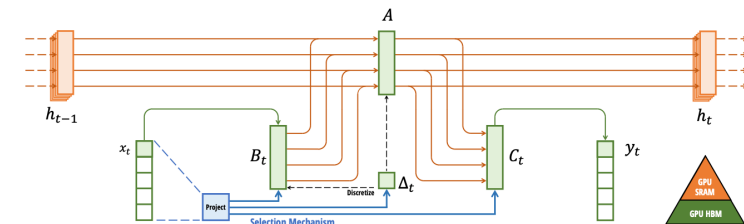
- Sample data $p(x_0) \rightarrow$ turn to noise



● Reverse / denoising process

- Sample noise $p_T(x_T) \rightarrow$ turn into data

Selective State Space Model with Hardware-aware State Expansion



Q&A