GenAI Basics

CSE585

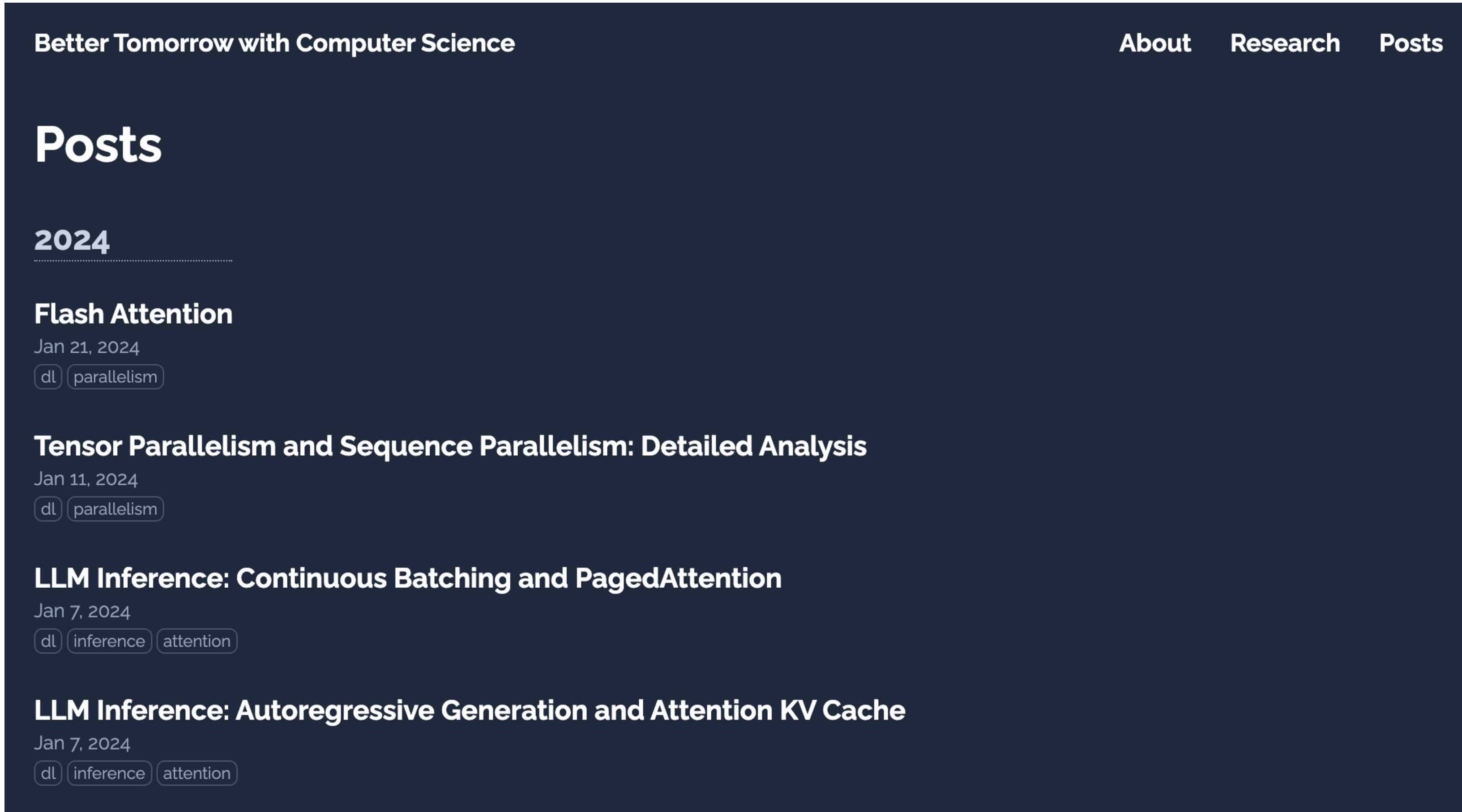
Insu Jang

August 29, 2024

Insu Jang

- Your GSI, Only GSI

- https://insujang.github.io



Agenda

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

Transformer Architecture

- Introduced in: [Attention is All You Need [NIPS'17]]

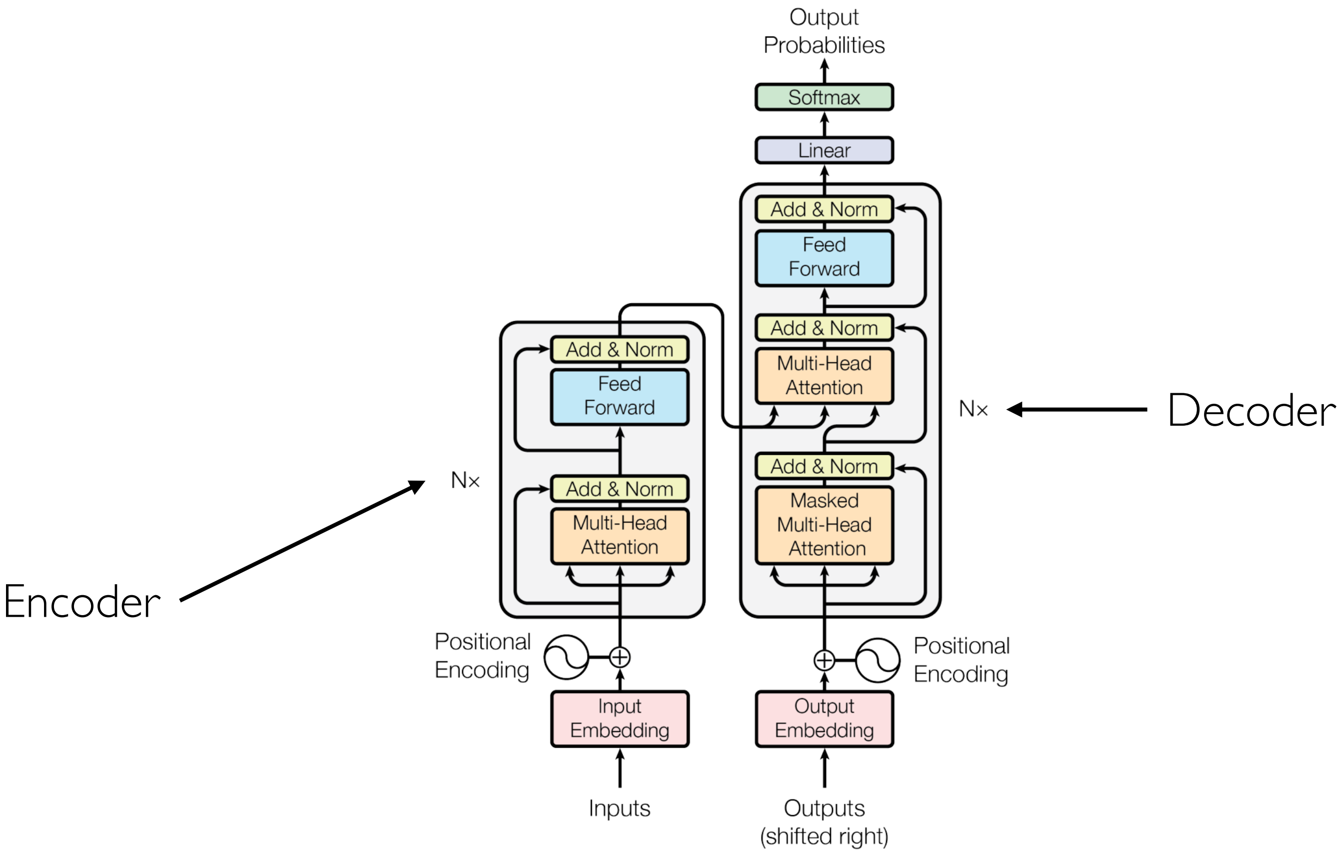
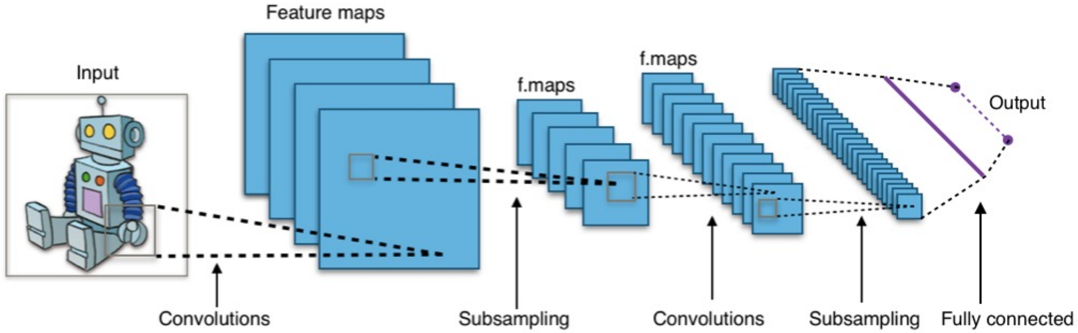


Figure 1: The Transformer - model architecture.

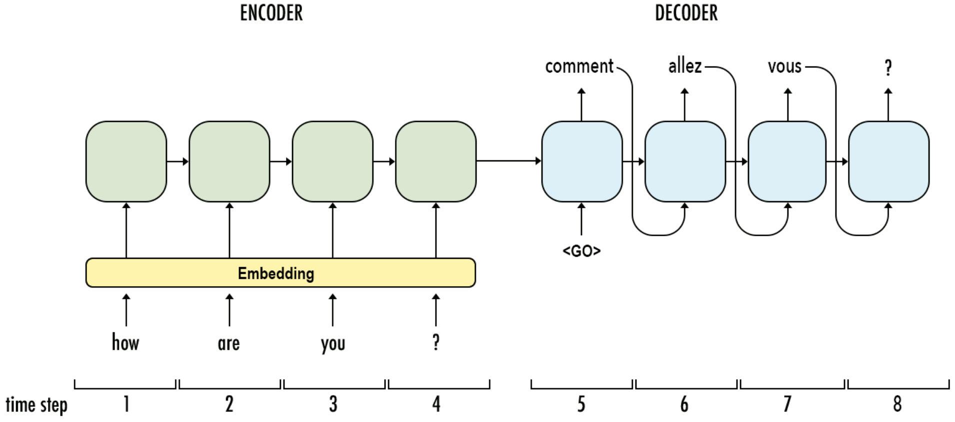
Why Transformer Was Introduced

- Before Transformer era

Computer Vision:
Convolutional Neural Networks (CNN)



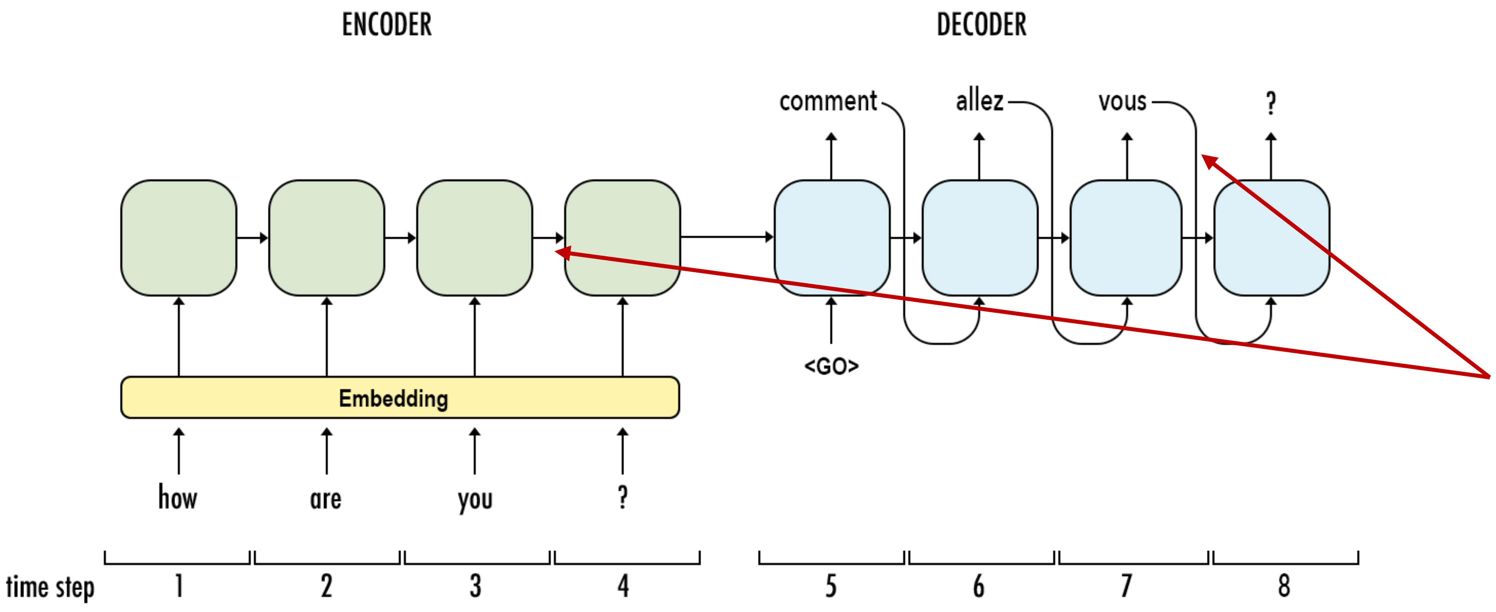
Natural Language Processing:
Recurrent Neural Networks (RNN)



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

Why Transformer Was Introduced

Natural Language Processing:
Recurrent Neural Networks (RNN)



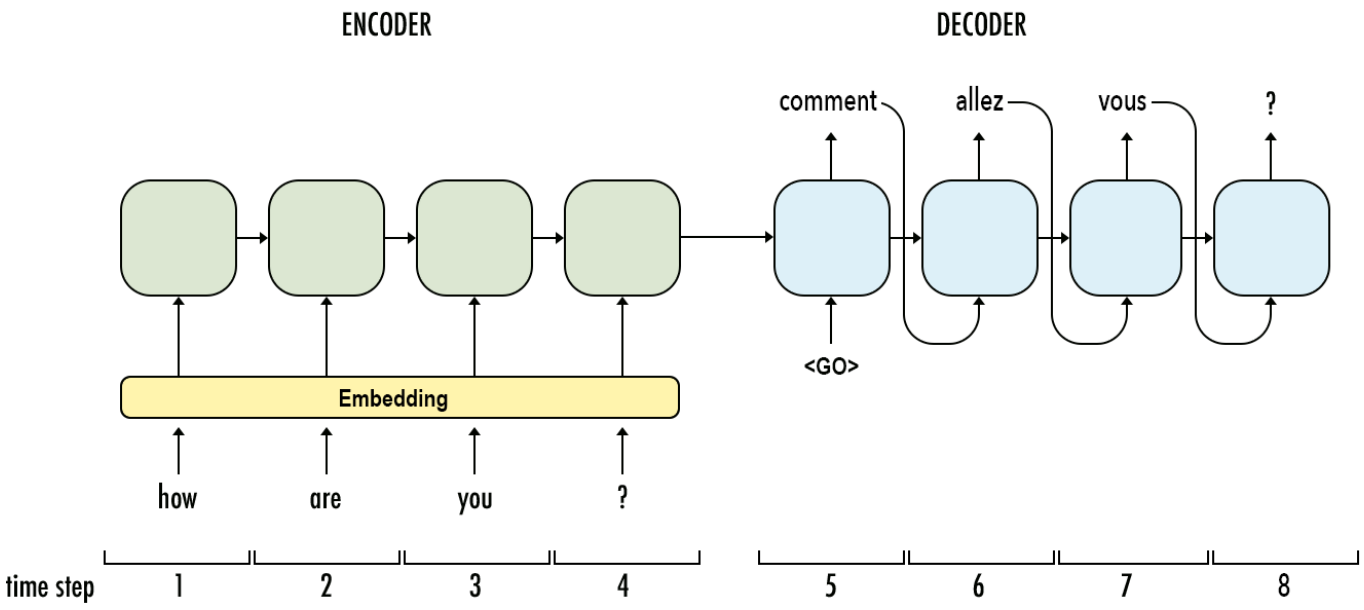
Output goes into input
of the next iteration or recurrance

Low utilization due to this architecture

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

Why Transformer Was Introduced

Natural Language Processing: Recurrent Neural Networks (RNN)



Transformer

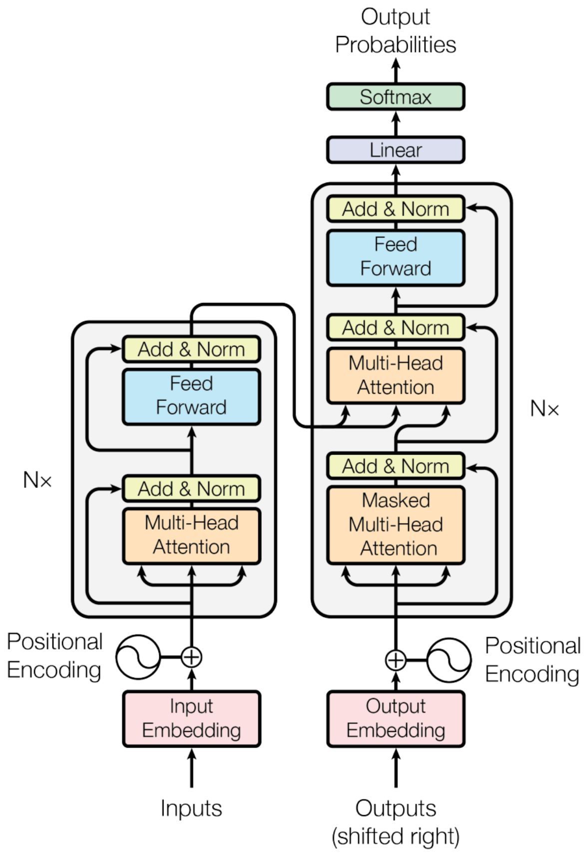


Figure 1: The Transformer - model architecture.

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

Attention is All You Need: https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

Attention is All You Need

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

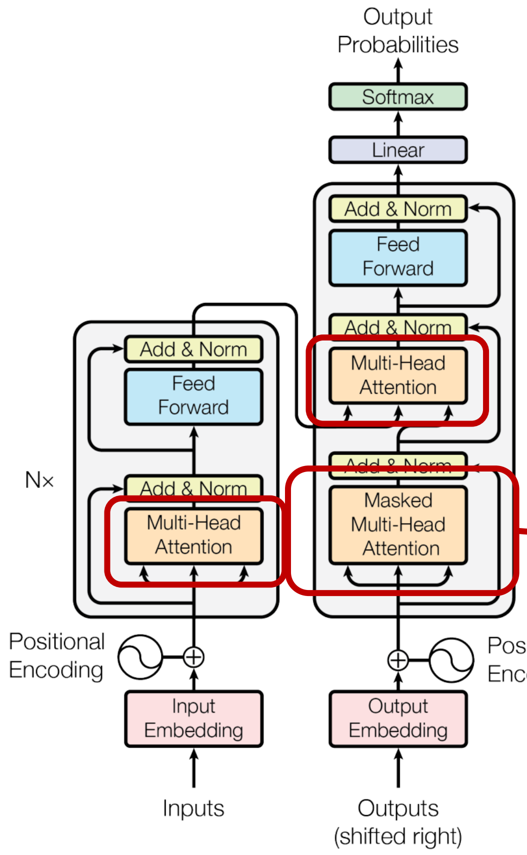
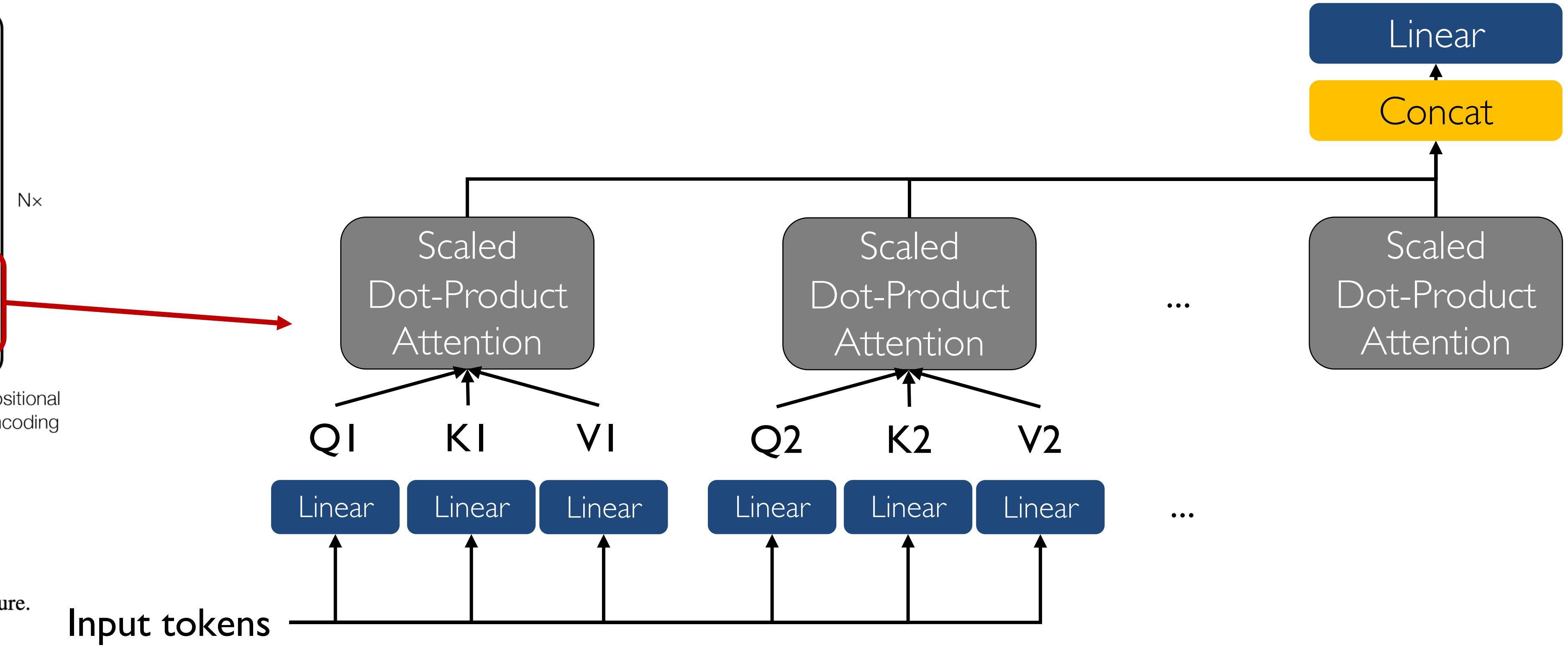


Figure 1: The Transformer - model architecture.



Attention is All You Need

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

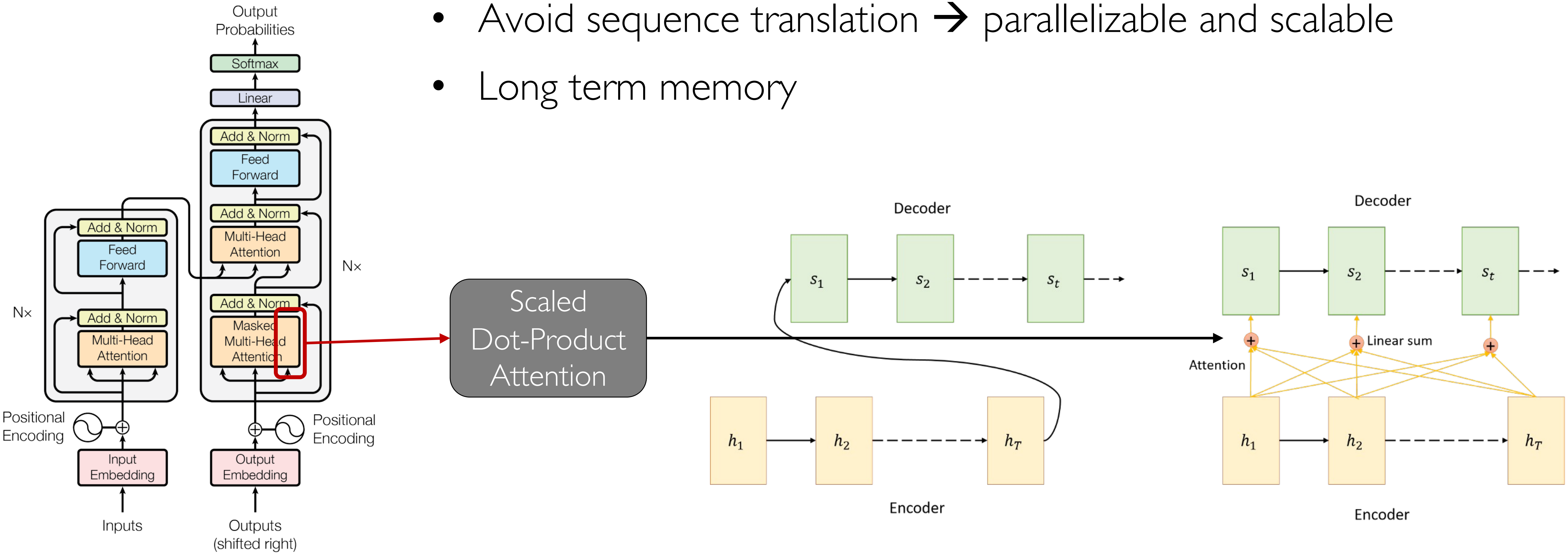


Figure 1: The Transformer - model architecture.

(a) Vanilla Encoder Decoder Architecture

(b) Attention Mechanism

Simple vs attention based encoder-decoder architecture: https://medium.com/swlh/a-simple-overview-of-rnn-lstm-and-attention-mechanism-9e844763d07b

Attention is All You Need

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

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

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

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		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

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

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					

Attention(Token_i, 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

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

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					

Attention(Token$_i$, 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

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

Input tokens

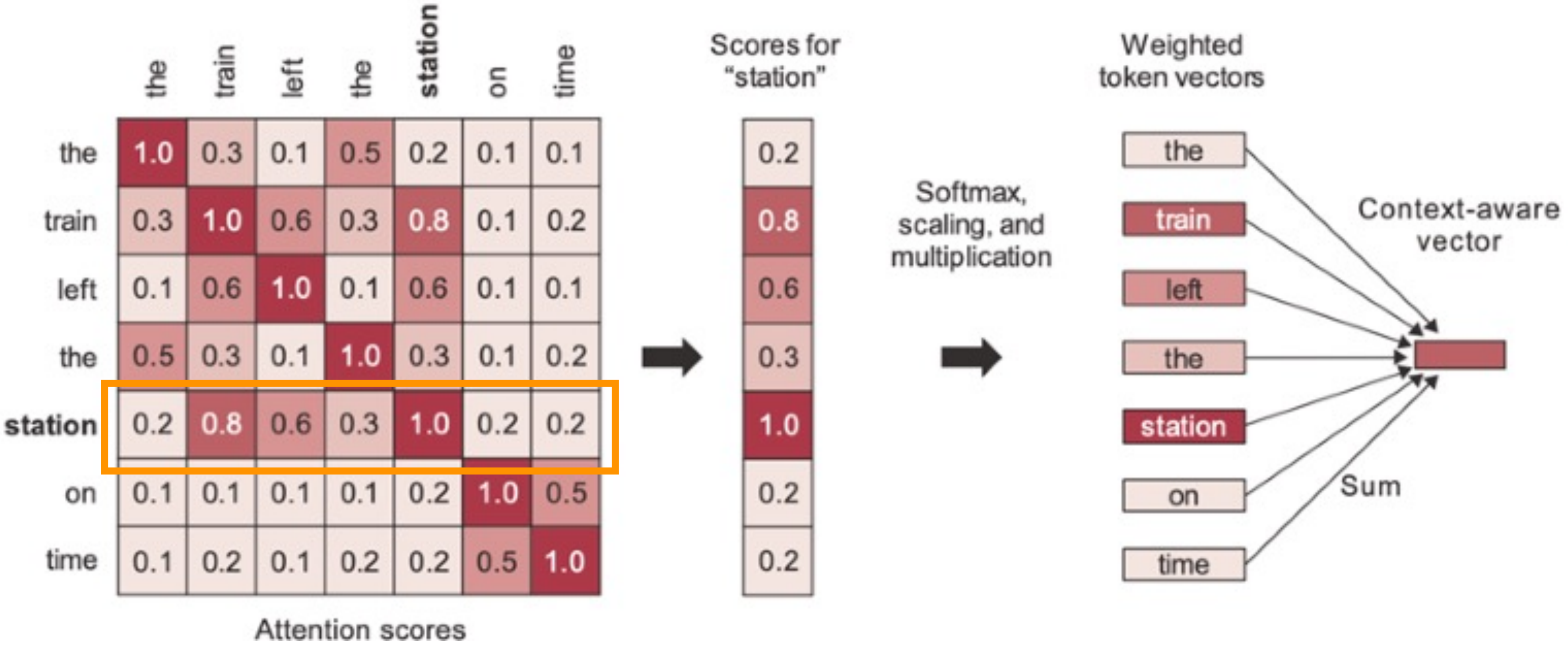
the

train

left

the

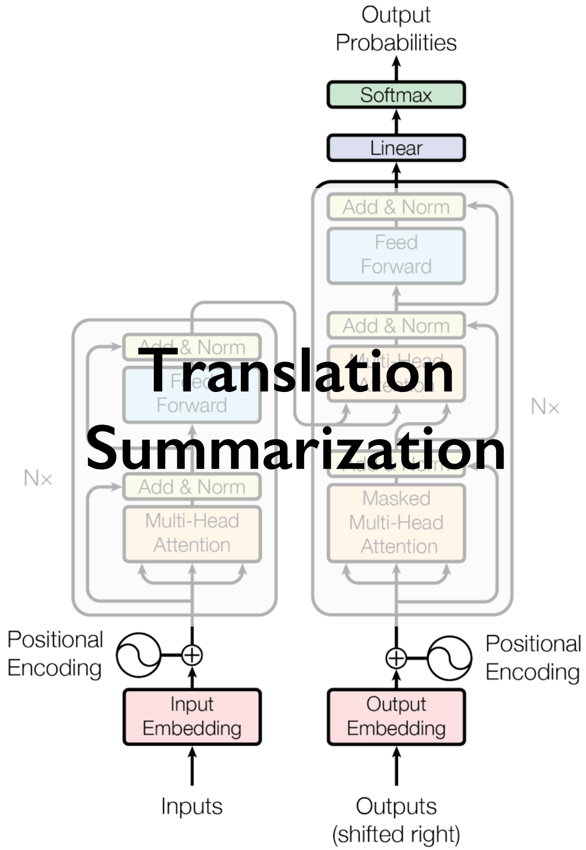
station



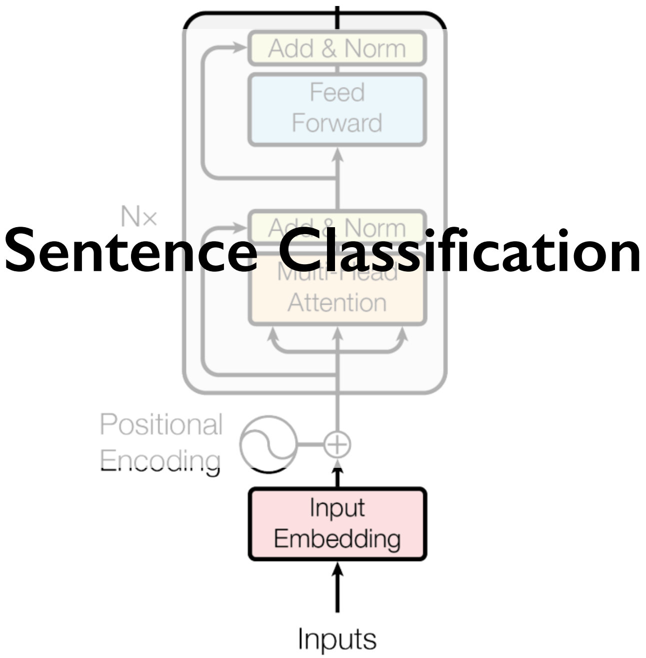
https://medium.com/@ramendrakumar/self-attention-d8196b9e9143

Transformer-based Large Language Models

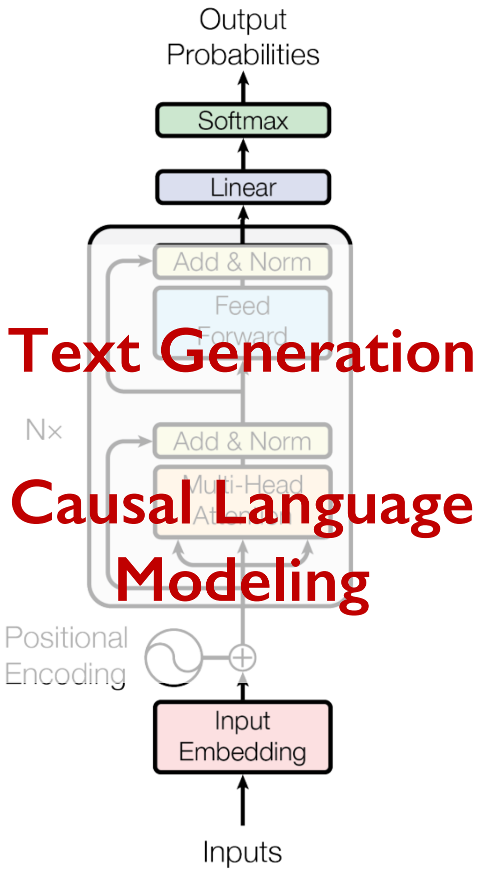
Encoder-decoder
(or sequence-to-sequence)
T5



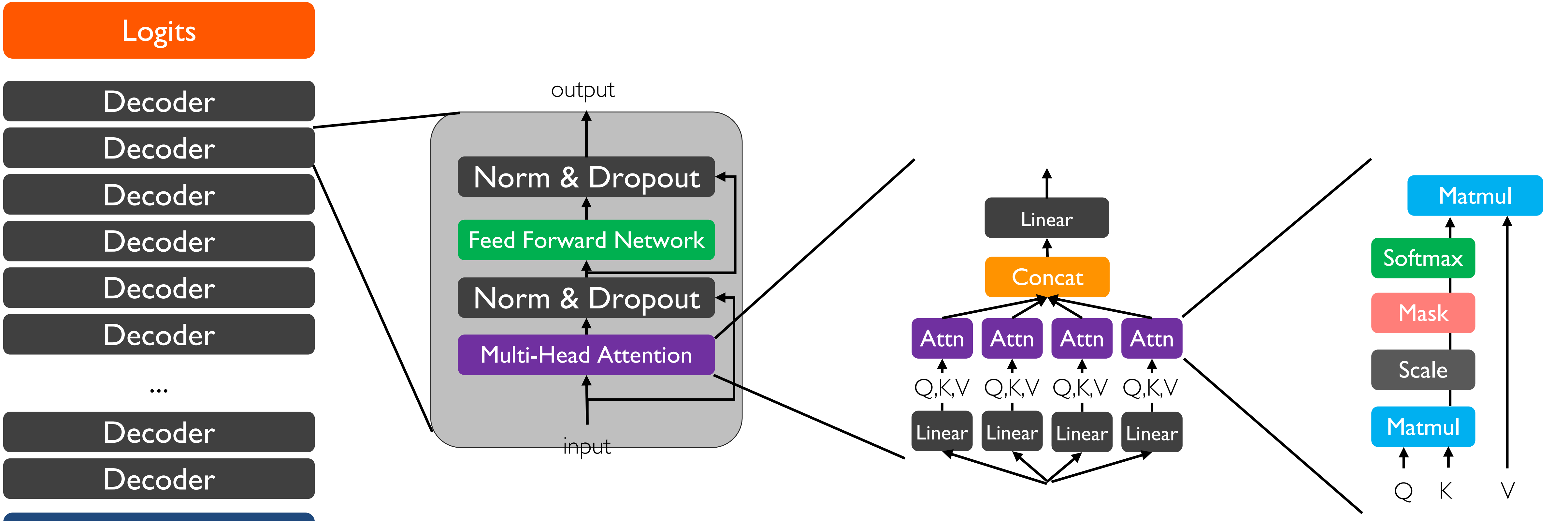
Encoder-only
BERT



Decoder-only
GPT



Transformer-based Large Language Models



Decoder only model architecture

Single decoder layer architecture

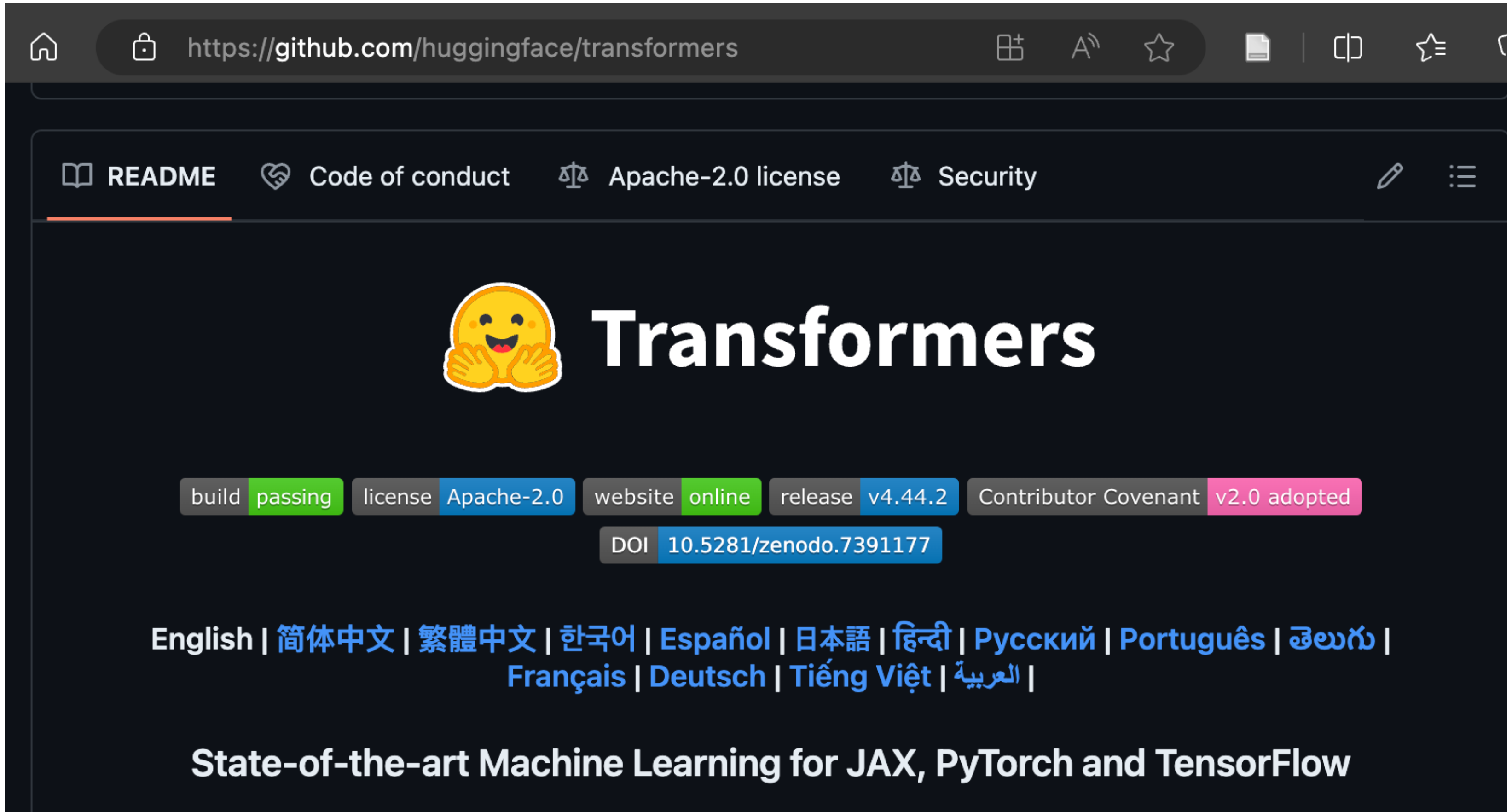
Multi-head attention architecture

Scaled Dot-Product Attention

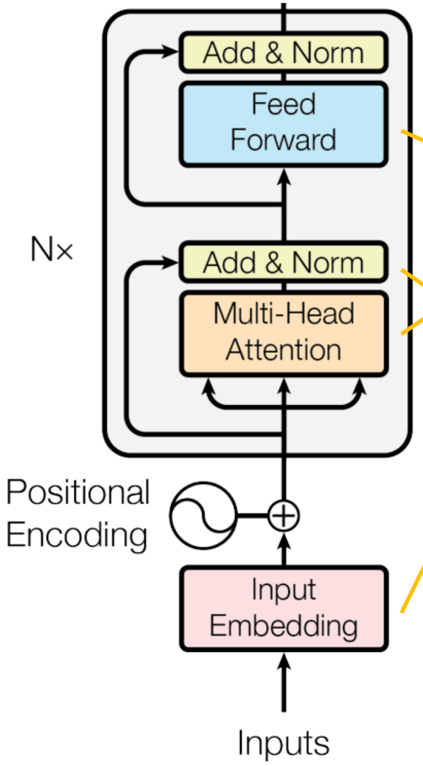
Let's See in the Code

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-
8B").to("cuda")

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

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

```
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

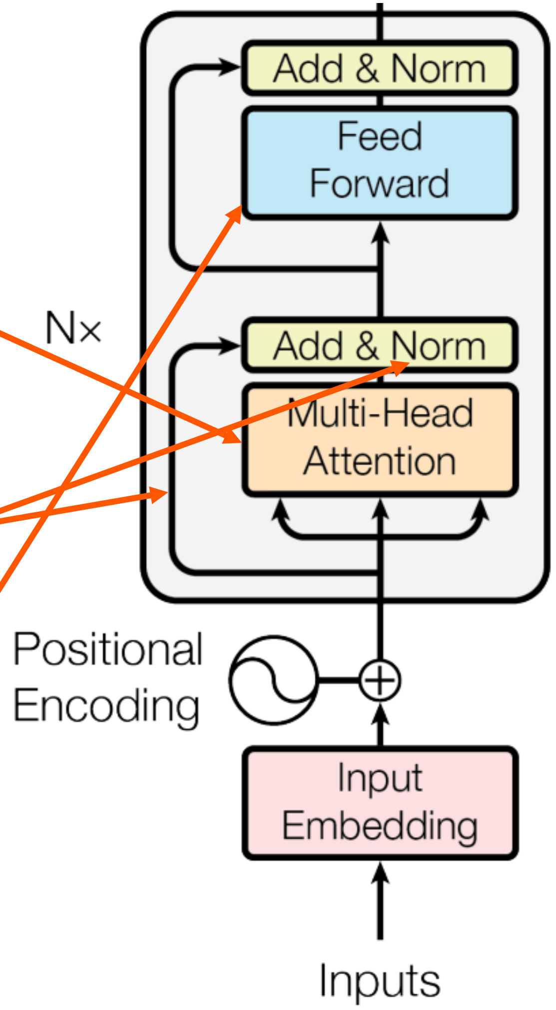
```python
class LlamaDecoderLayer(nn.Module):
    def forward(

        # 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
```

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

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



Let's See in the Code

```python
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")
```

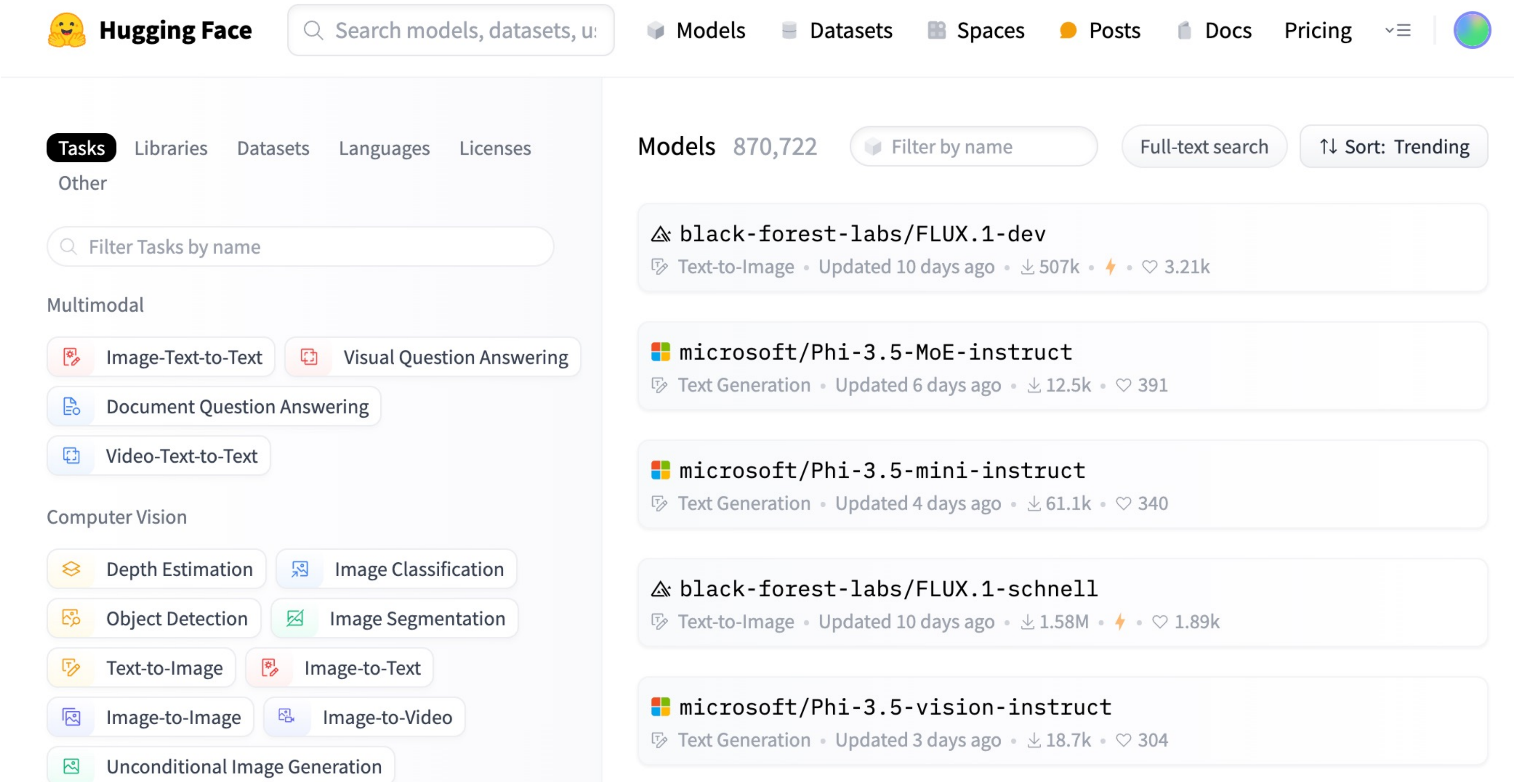
```python
output = model.generate(input_ids, max_length=100, do_sample=True)
tokenizer.decode(output[0])
```

```
"<|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"
```

https://huggingface.co/spaces/Xenova/the-tokenizer-playground

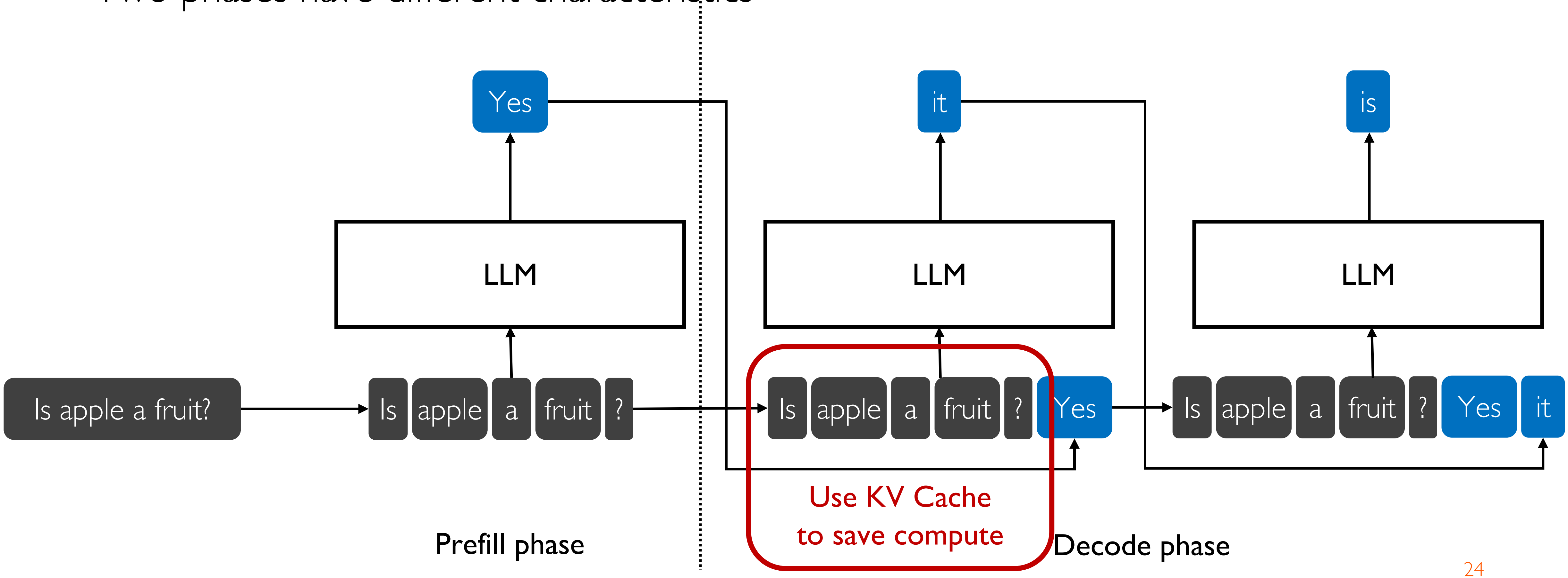
Let's See in the Code

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



Characteristics of Transformer LLMs

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

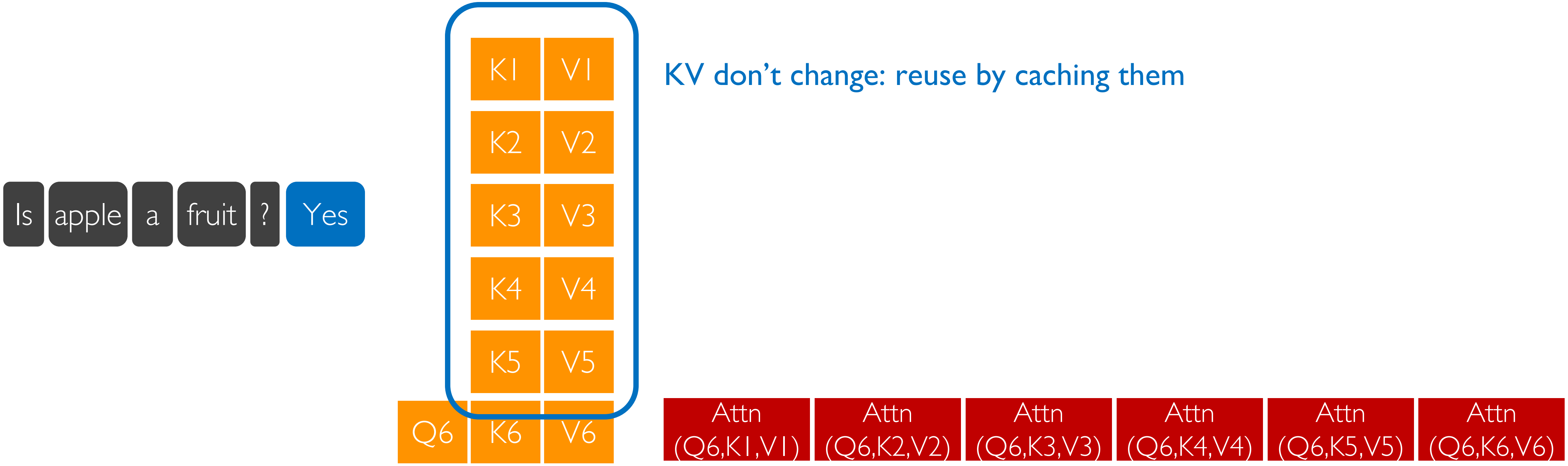


Characteristics of Transformer LLMs

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

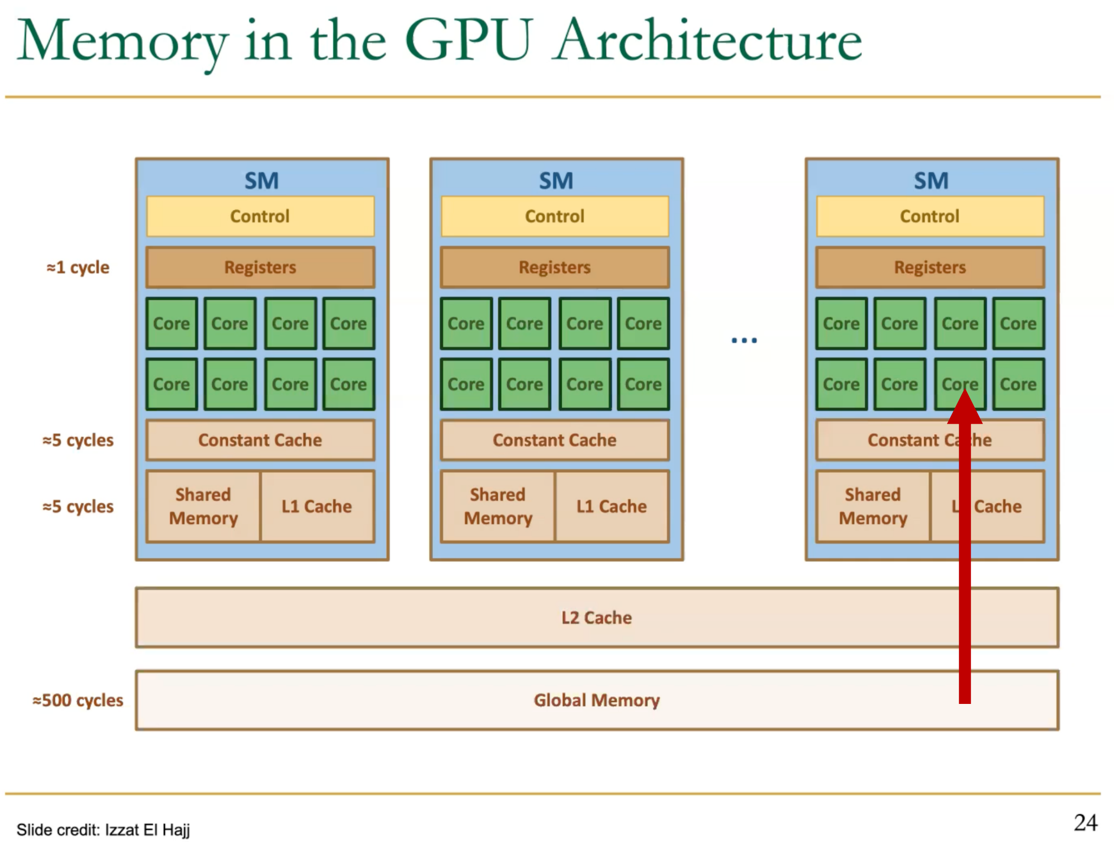
Characteristics of Transformer LLMs

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



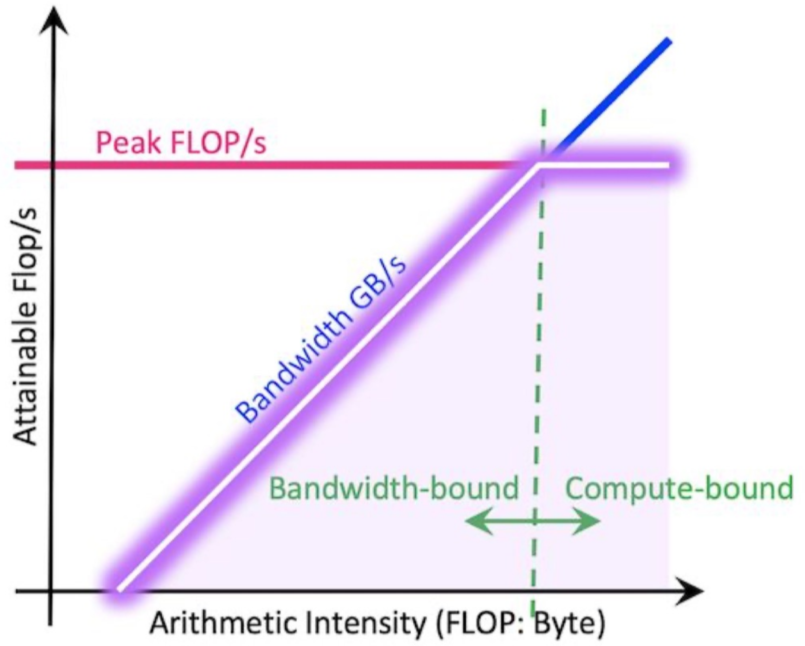
Characteristics of Transformer LLMs

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



Arithmetic intensity:

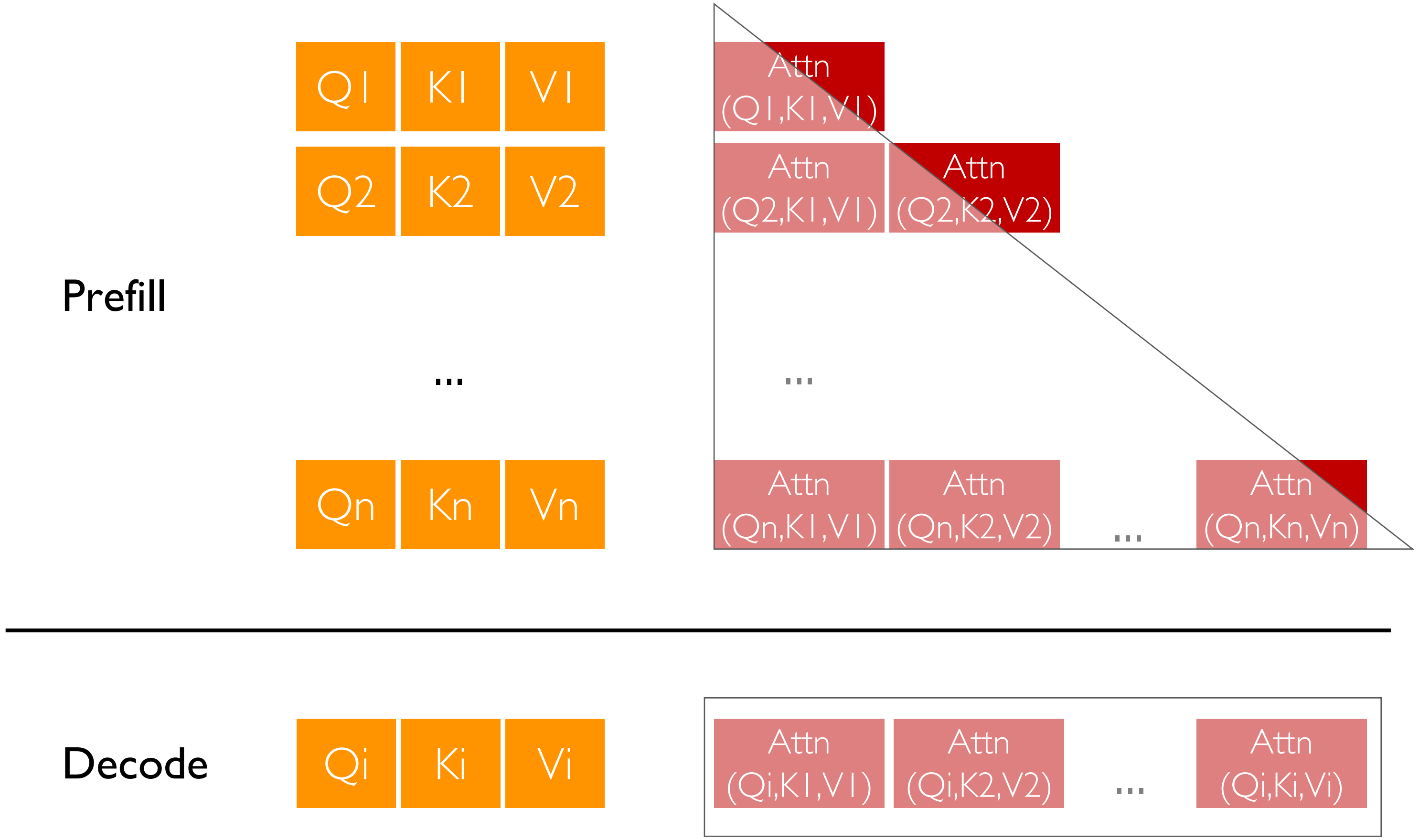
compute operations / # byte accesses



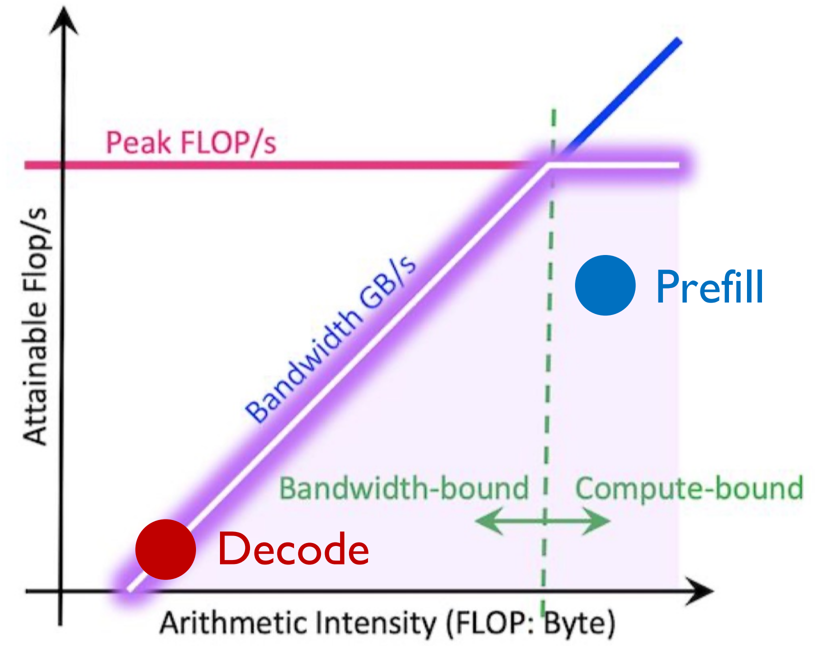
Memory in the GPU Architecture: https://www.youtube.com/watch?v=ZQKMZIP3Fzg

Characteristics of Transformer LLMs

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

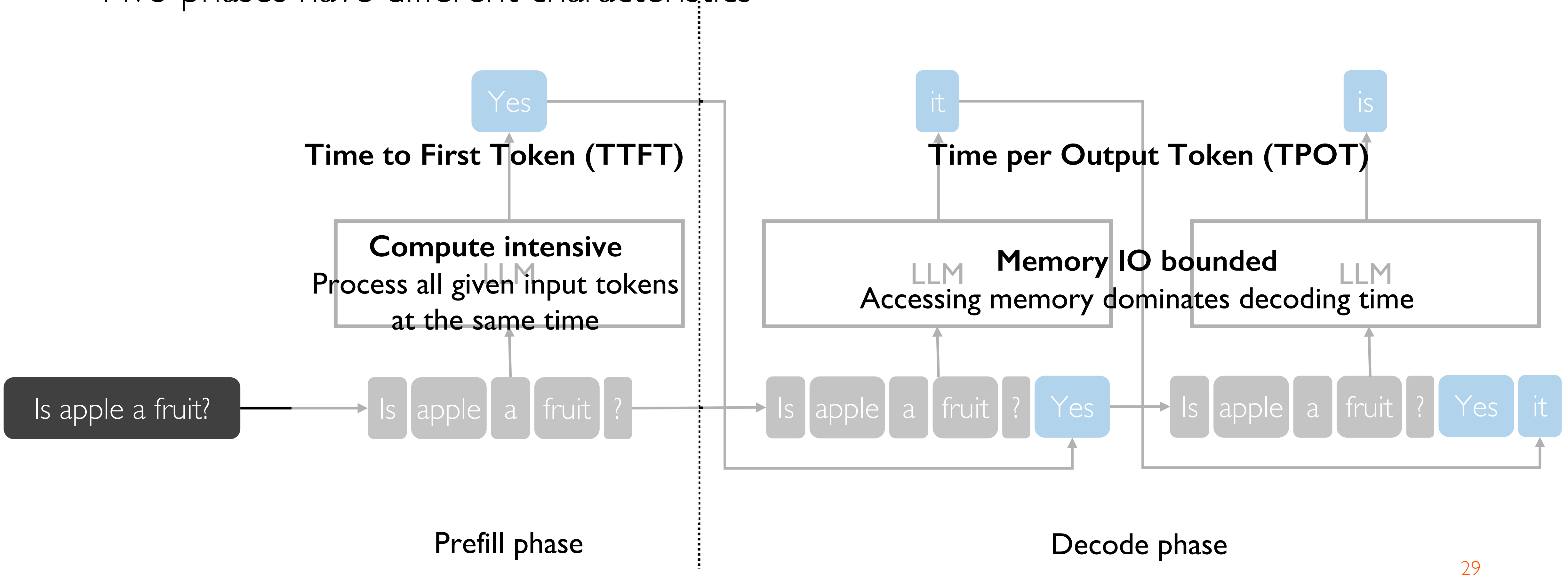


Arithmetic intensity:
\# compute operations / # byte accesses



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**

1 H100
80GB
Memory

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

+

Additional memory
for **training**

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

Characteristics of Transformer LLMs

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

1 H100
80GB
Memory

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

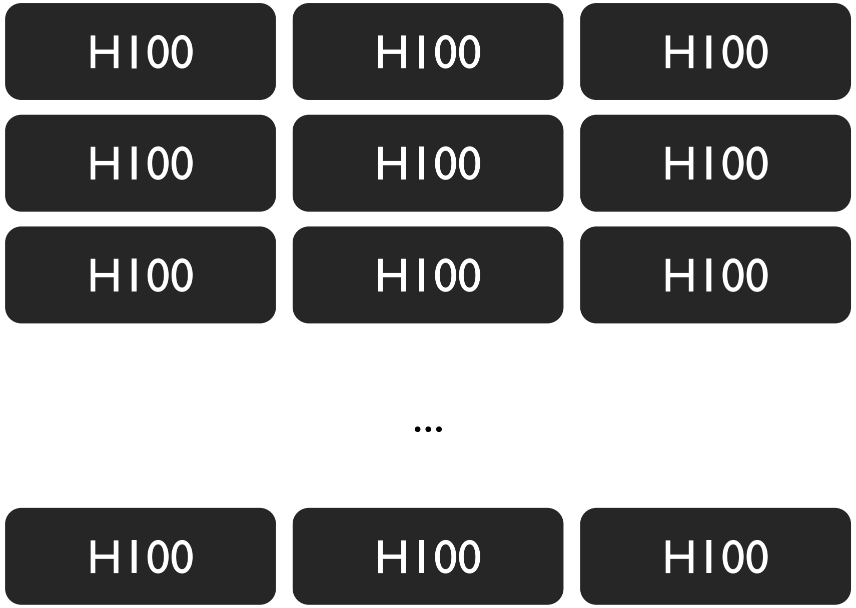
+

Additional memory
for **inference**

> 6TB KV Cache
for 50 concurrent requests
with 8k context length

System Designs for LLMs

- Distributed ML



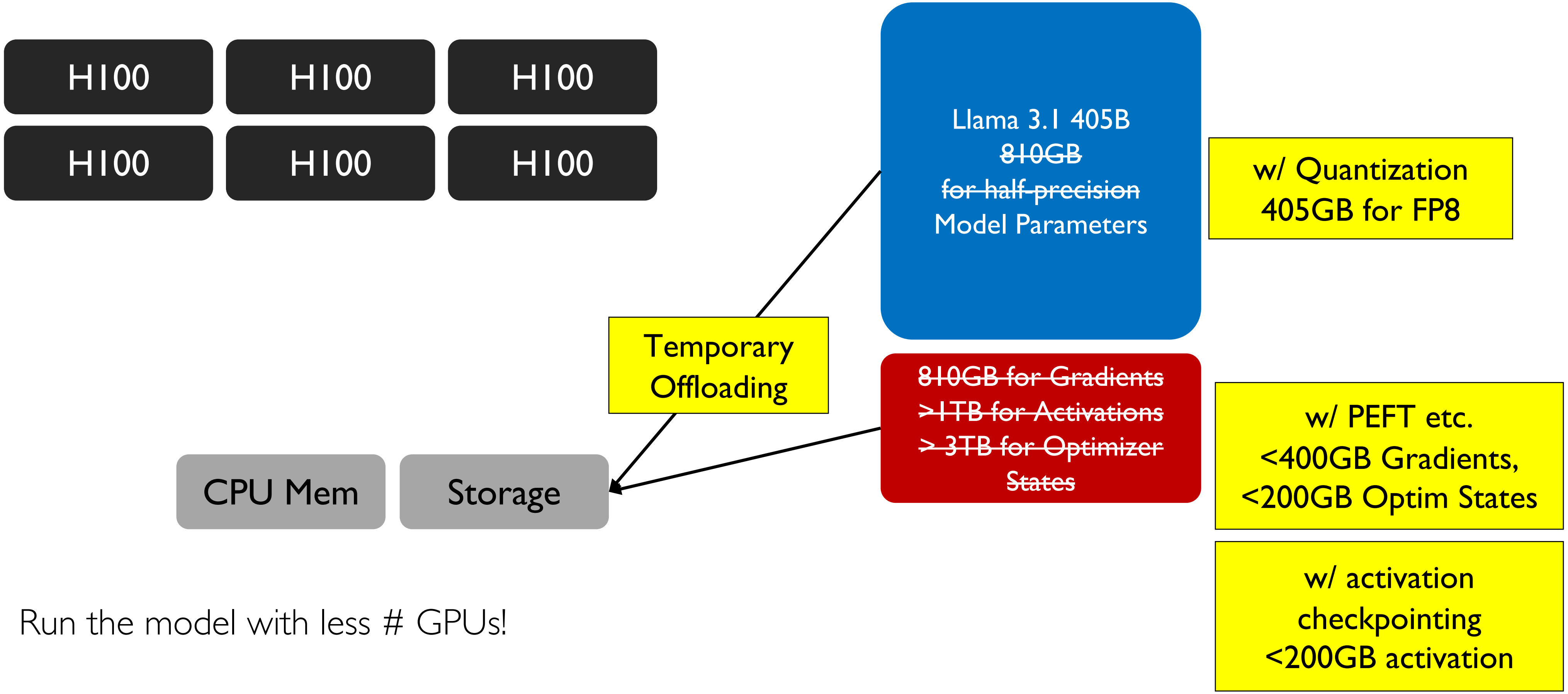
For Llama 3.1 405B: 16k H100s are used to train

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

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

System Designs for LLMs

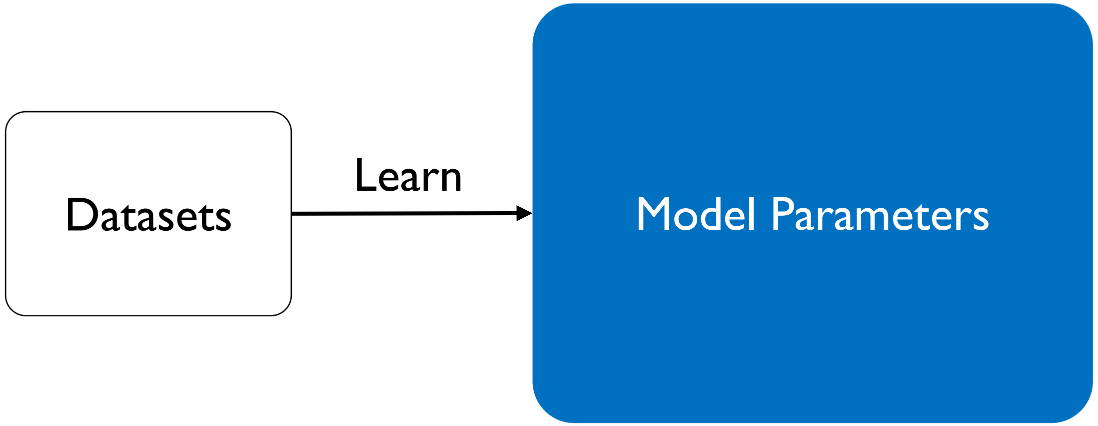
- Memory efficient ML



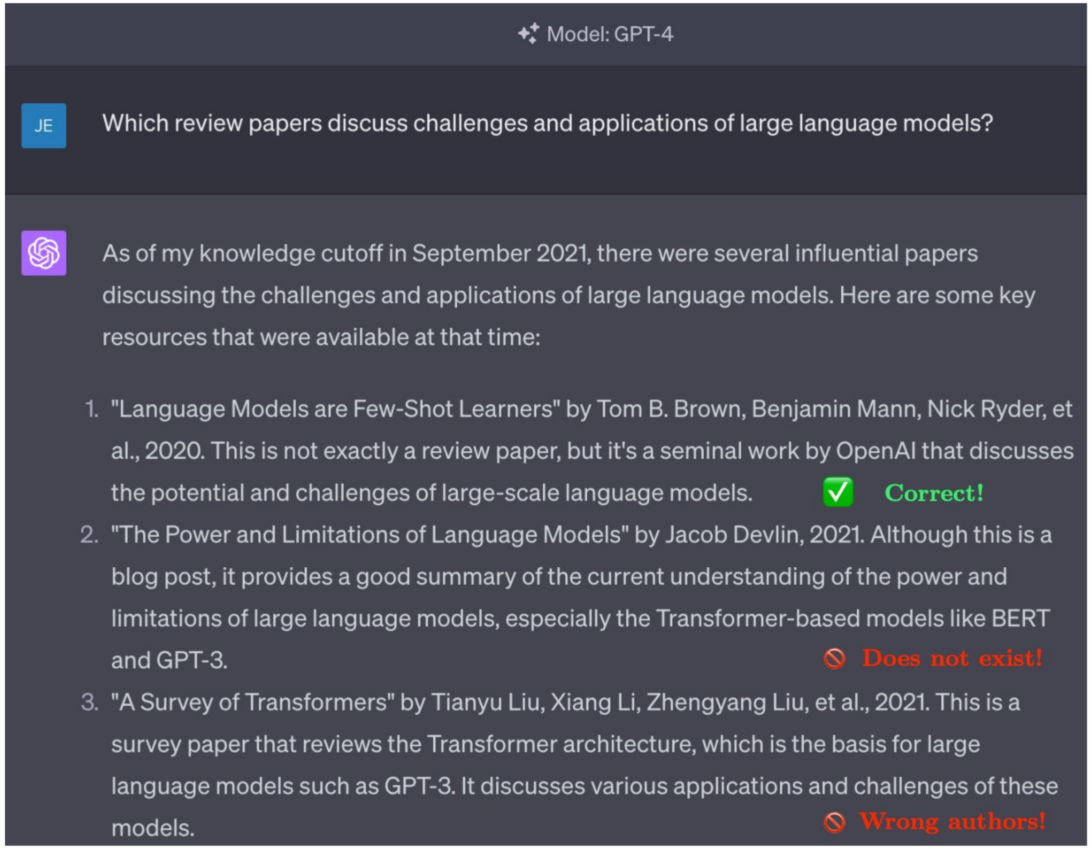
Run the model with less # GPUs!

Characteristics of Transformer LLMs

- Outdated Knowledge & hallucinations

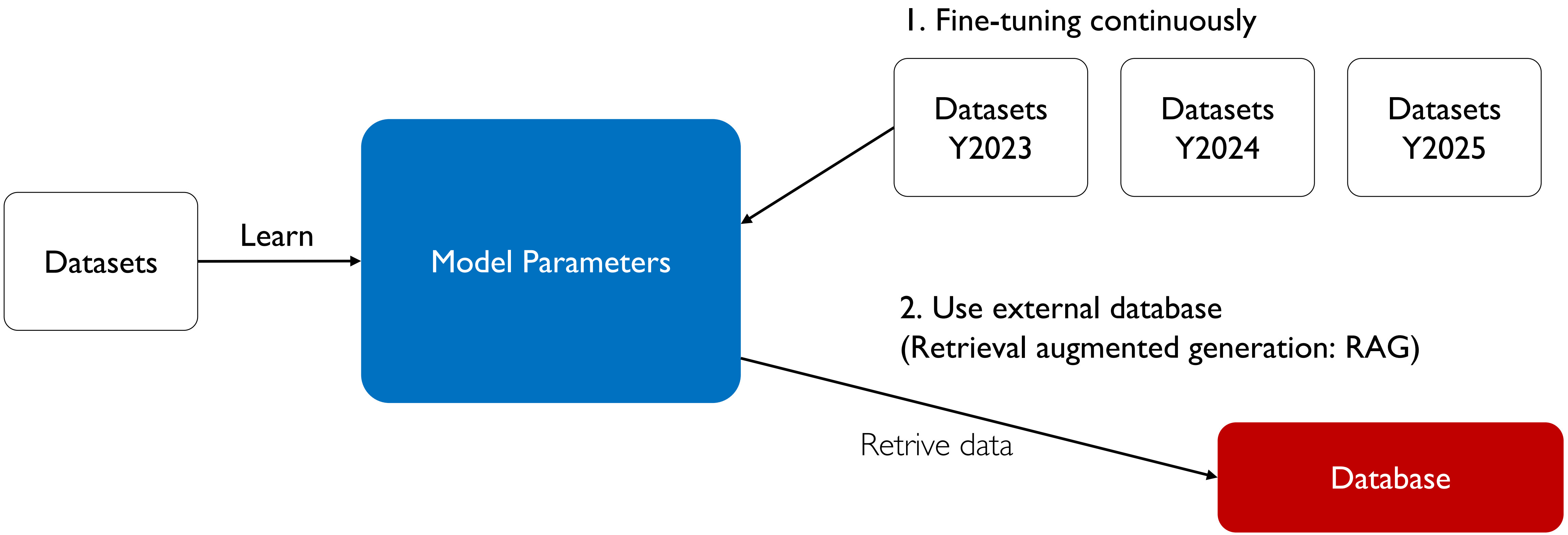


Knowledge cutoff: model doesn't know later information
Generate "plausible sounding but factually incorrect response"



Characteristics of Transformer LLMs

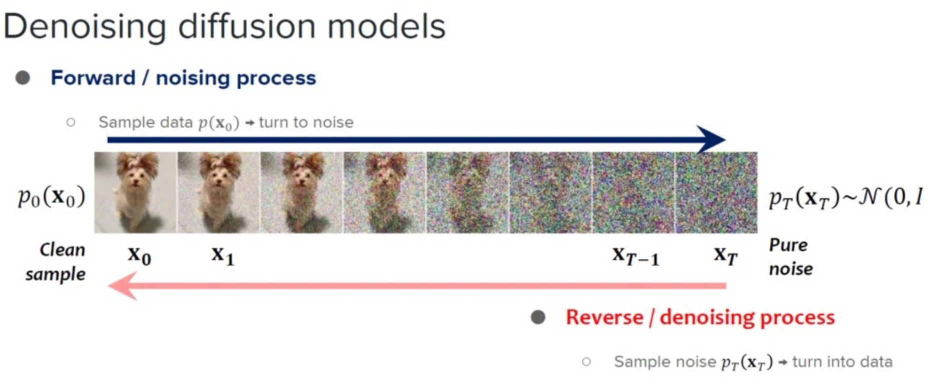
- Outdated Knowledge & hallucinations: mitigations



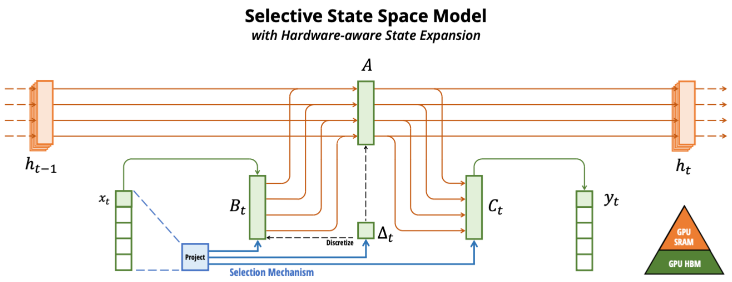
GenAI != Transformer

- Transformer architecture is dorminating <u>text generation</u> <u>as of now</u>

- Diffusion models: generate data by denoising diffusion



- State space models: improve Transformer's computational inefficiency



- and more

https://www.iguazio.com/glossary/diffusion-models/
https://arxiv.org/abs/2312.00752

Q&A