



大模型 - AI 集群

# 大模型推理显存分析



ZOMI

# 关于本内容



# 关于本内容

- LLMs 大模型推理显存分析 & KV cache 原理
  - Transformer 训练显存回顾
  - 大模型推理显存分析
  - 大模型推理过程理解
  - KV Cache 原理



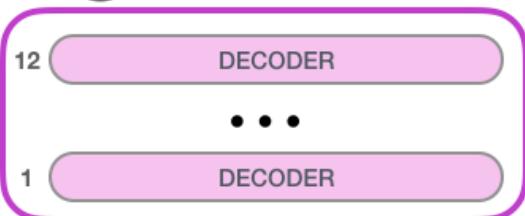
# 1. Transformer

训练显存占用回顾

# 大模型 Transformers 堆叠



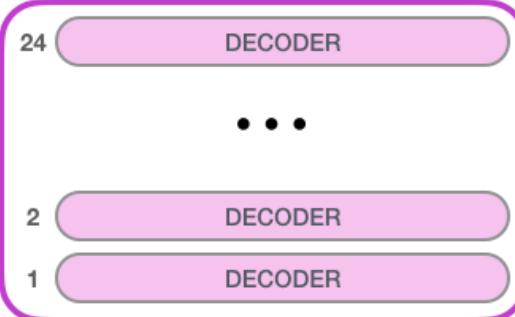
GPT-2  
SMALL



Model Dimensionality: 768



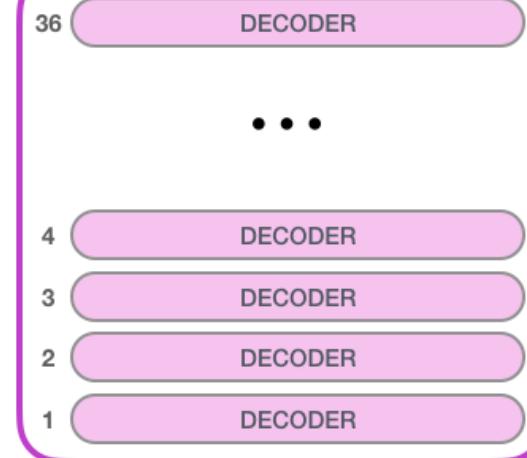
GPT-2  
MEDIUM



Model Dimensionality: 1024



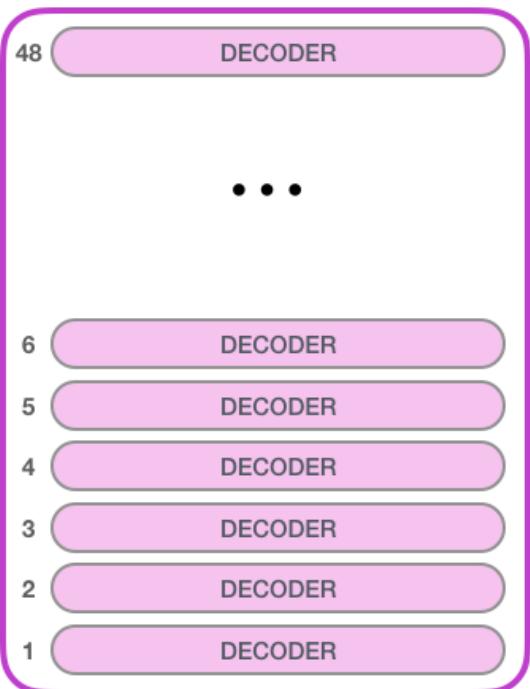
GPT-2  
LARGE



Model Dimensionality: 1280



GPT-2  
EXTRA  
LARGE



Model Dimensionality: 1600



# Transformers 结构

- transformer 中最核心的为两个部分：Self-attention 层和 MLP 层

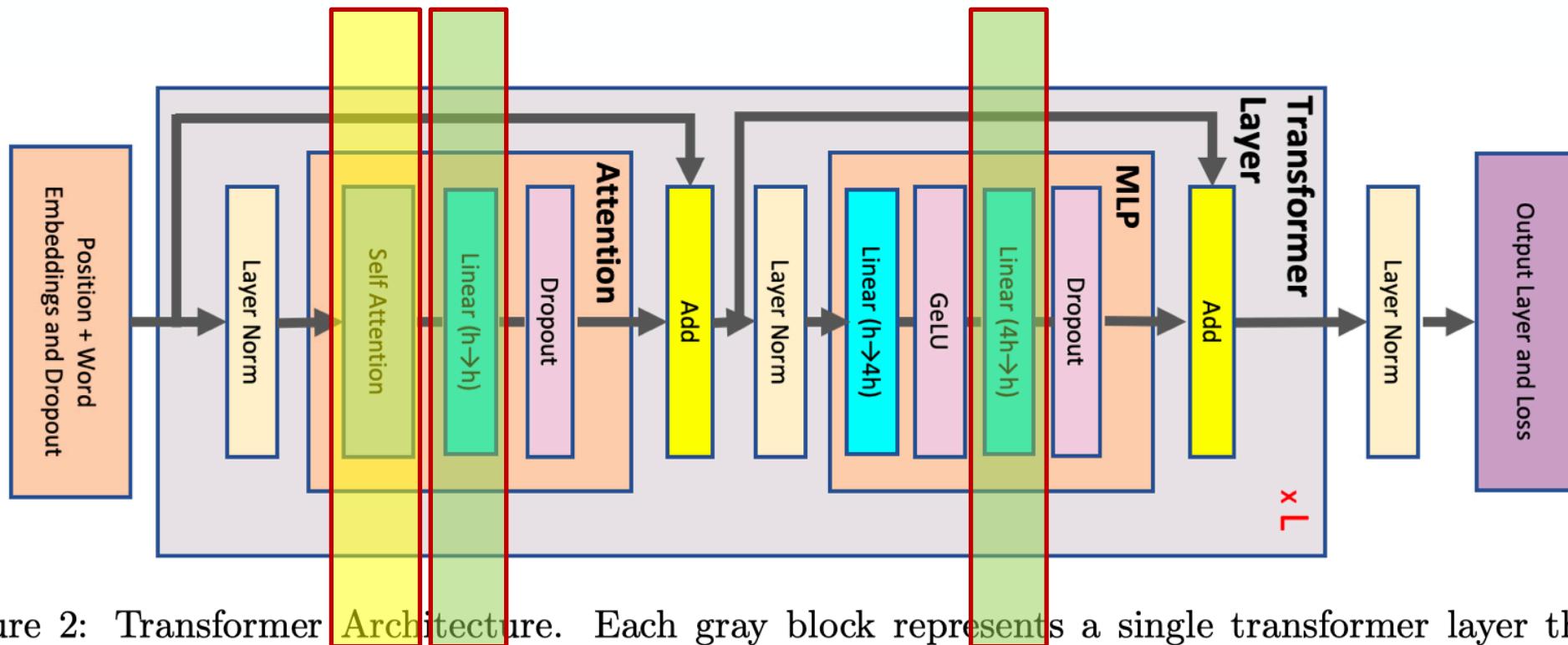


Figure 2: Transformer **Arch**itecture. Each gray block represents a single transformer layer that is replicated  $L$  times.

# 模型参数量与模型大小关系

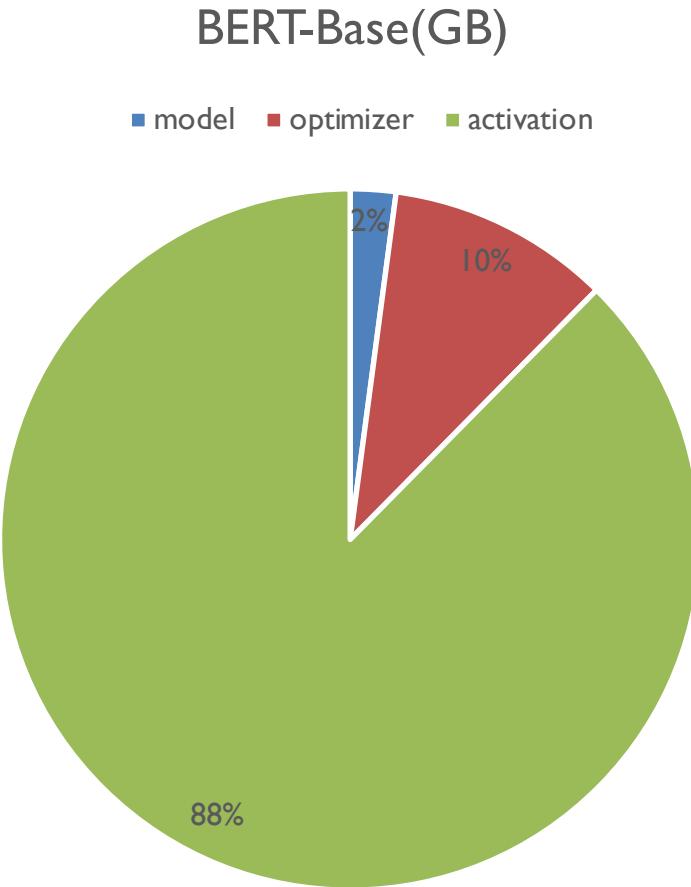
- $l$  层 Transformer 结构可训练参数量为  $l(12h^2) + Vh$
- 模型使用 FP16/BF16 方式保存和加载到显存，那么占用 2 个 Byte
- 模型使用 FP32 方式保存和加载到显存，那么占用 4 个 Byte

模型名称	隐藏层维度 $h$	层数 $l$	$12h^2$	实际参数量	模型大小 (FP16)
LLAMA-6B	4096	32	6442450944	6.7B	12 GB
LLAMA-13B	5120	40	12582912000	13.0B	23.4 GB
LLAMA-33B	6656	60	31897681920	32.5B	59.4 GB
LLAMA-65B	8192	80	64424509440	65.2B	120 GB



# 大模型训练内存占用

- $l$  层 Transformer 结构可训练参数量为  $l(12h^2) + Vh$
- **模型 Model**
  - Parameters 权重参数 (half) 2 bytes ,
  - Gradient 梯度参数 (half) 2 bytes ,
- **优化器状态 Optimizers status**
  - Master Weight (FP32) 4 bytes
  - Adam m (FP32) 4 bytes
  - Adam v (FP32) 4 bytes
- **激活值：forward 中保存，用于反向传播**



# 2. 大模型推理

## 显存分析

# 模型训练的总内存

- 在一次训练迭代中，每个可训练参数都对应 1 个梯度，2 个优化器状态（Adam）。设模型参数量为  $\varphi(FP16)$ ，那么梯度的参数量为  $2\varphi(FP32)$ ，Adam 优化器的参数量为  $4\varphi(FP32)$
- 大模型混合精度训练过程中，使用 BF16 进行前向传递，FP32 反向传递梯度信息；优化器更新模型参数时，使用 FP32 优化器状态、FP32 的梯度来更新模型参数。

$$\text{训练总内存} = \underbrace{\text{模型内存}}_{\varphi} + \underbrace{\text{梯度内存}}_{2\varphi} + \underbrace{\text{优化器内存}}_{4\varphi} + \underbrace{\text{激活内存}}_{X\varphi} + \underbrace{\text{其他内存}}_{1.X\varphi}$$

# 模型推理的总内存

- 神经网络推理（Inference）阶段，没有优化器状态和梯度信息，也不需要保存中间激活。因此推理阶段占用的显存要远远小于训练阶段。
- 模型推理阶段，占用显存大头是模型参数，一次推理过程中，每个可训练参数都对应 1 个梯度，设模型参数数量为  $\varphi$ ，使用 float16 来进行推理，模型参数占用的显存大概是  $2\varphi$  bytes。

$$\text{推理总内存} = \underbrace{\text{模型内存}}_{\varphi} + \underbrace{\text{其他内存}}_{0.X\varphi}$$

# 模型推理的总内存

- 目前业界流行的是使用 KV cache 来加速推理，而 KV cache 也需要占用显存。
- 此外，输入的数据也需要放到 NPU 上。

$$\text{推理总内存} = \underbrace{\text{模型内存}}_{\varphi} + \underbrace{\text{KV Cache内存}}_{0.5X \varphi} + \underbrace{\text{其他内存}}_{0.2X \varphi}$$

# 目标思考

1. LLMs 大模型推理性能的最大瓶颈在于显存；
2. 不希望在 LLMs 大模型推理有太多的并行，尽可能减少通信；

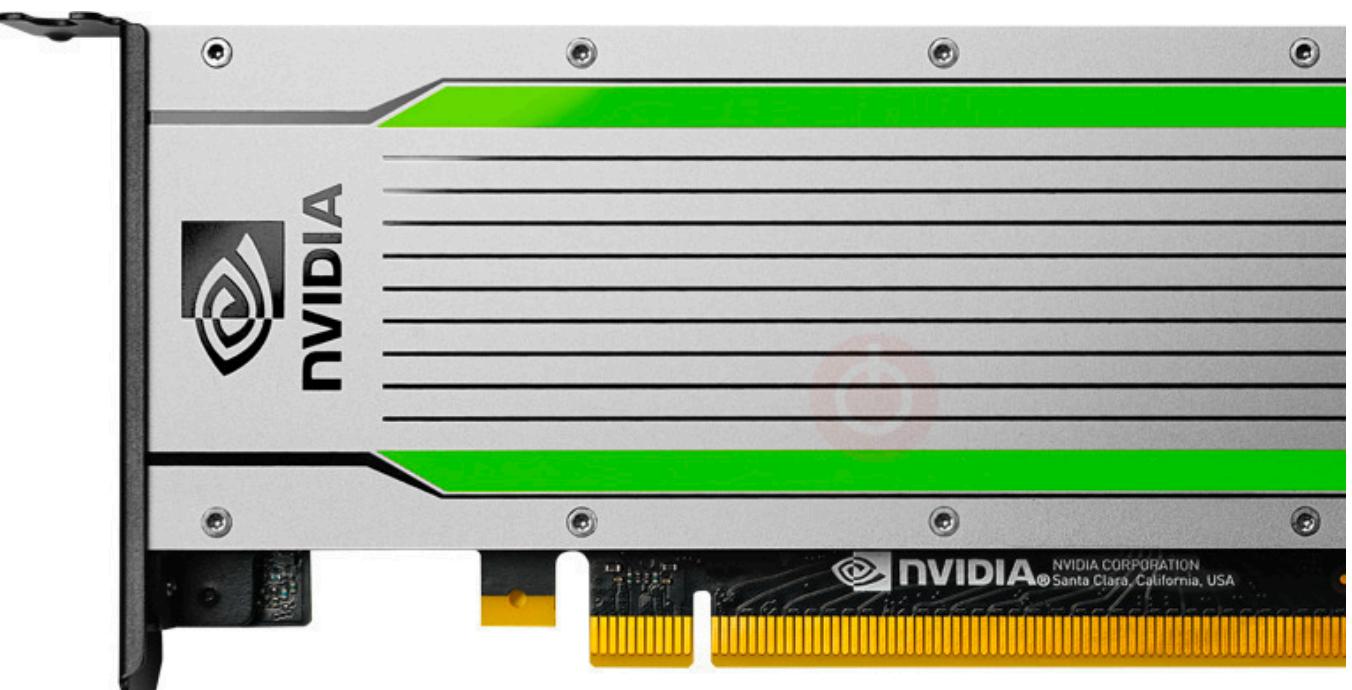


# 推理硬件介绍

- NVIDIA 英伟达推理加速卡 T4 , <https://www.nvidia.cn/data-center/tesla-t4/> , 因为成本的原因主要采用GDDR , 而非 HBM ( DRAM 芯粒三维堆叠 )

## 规格

GPU 架构	<b>NVIDIA Turing</b>
NVIDIA Turing Tensor 核心数量	<b>320</b>
NVIDIA CUDA® 核心数量	<b>2560</b>
单精度	<b>8.1 TFLOPS</b>
混合精度 (FP16/FP32)	<b>65 TFLOPS</b>
INT8	<b>130 TOPS</b>
INT4	<b>260 TOPS</b>
GPU 显存	<b>16 GB GDDR6</b> <b>300 GB/s</b>
ECC	<b>支持</b>
互联带宽	<b>32 GB / 秒</b>
系统接口	<b>x16 PCIe Gen3</b>



# 3. 大模型 推理过程

# 符号

- Transformer 模型的层数为  $l$ ，隐藏层维度为  $h$ ，注意力头数为  $a$ ，词表大小为  $V$ ，训练数据的批次 batch 大小为  $b$ ，序列长度为  $s$ ，当前生成的 token 代表的向量表示为  $t$ 。

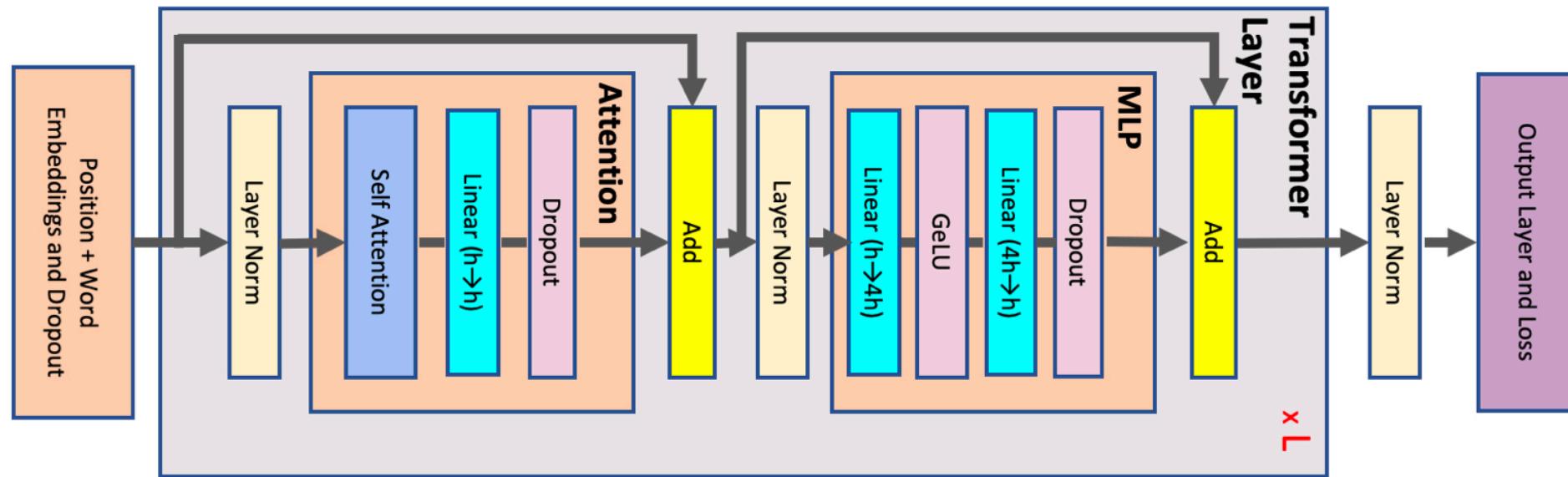
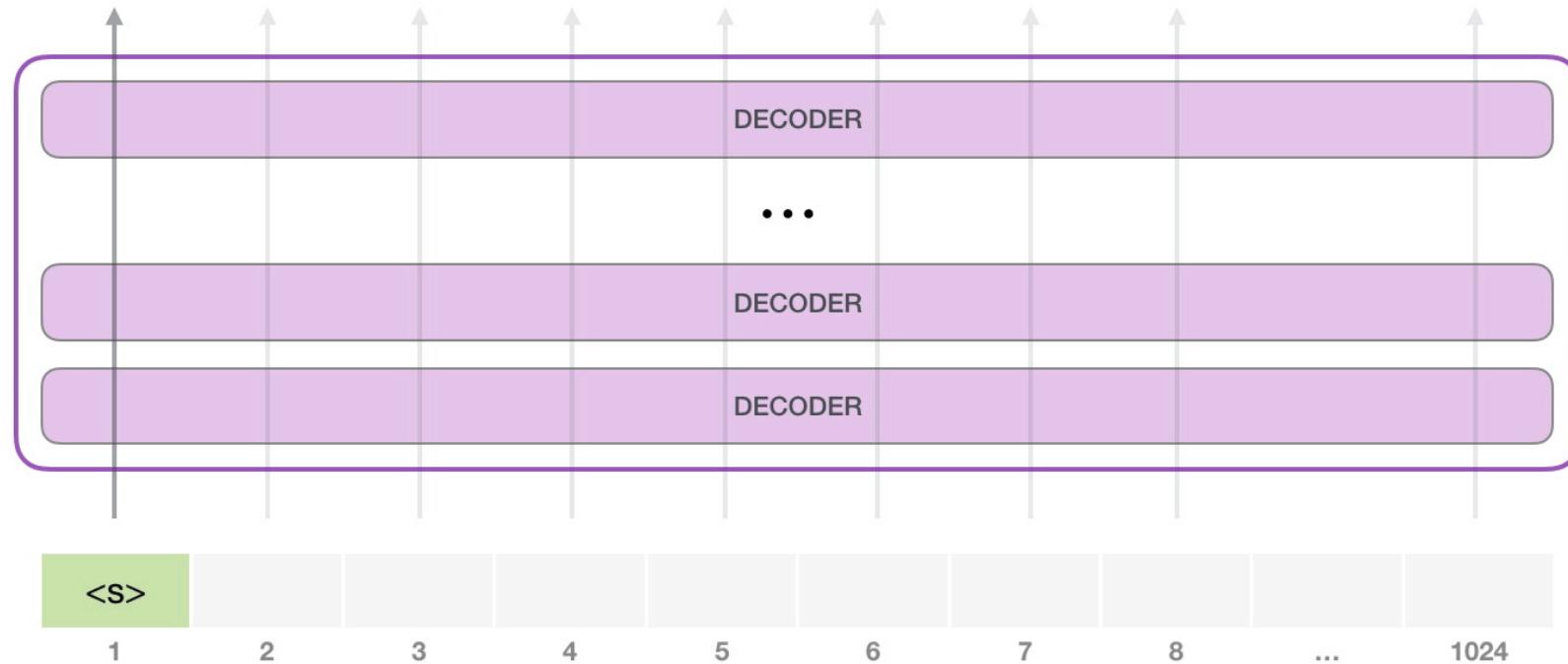


Figure 2: Transformer Architecture. Each gray block represents a single transformer layer that is replicated  $L$  times.

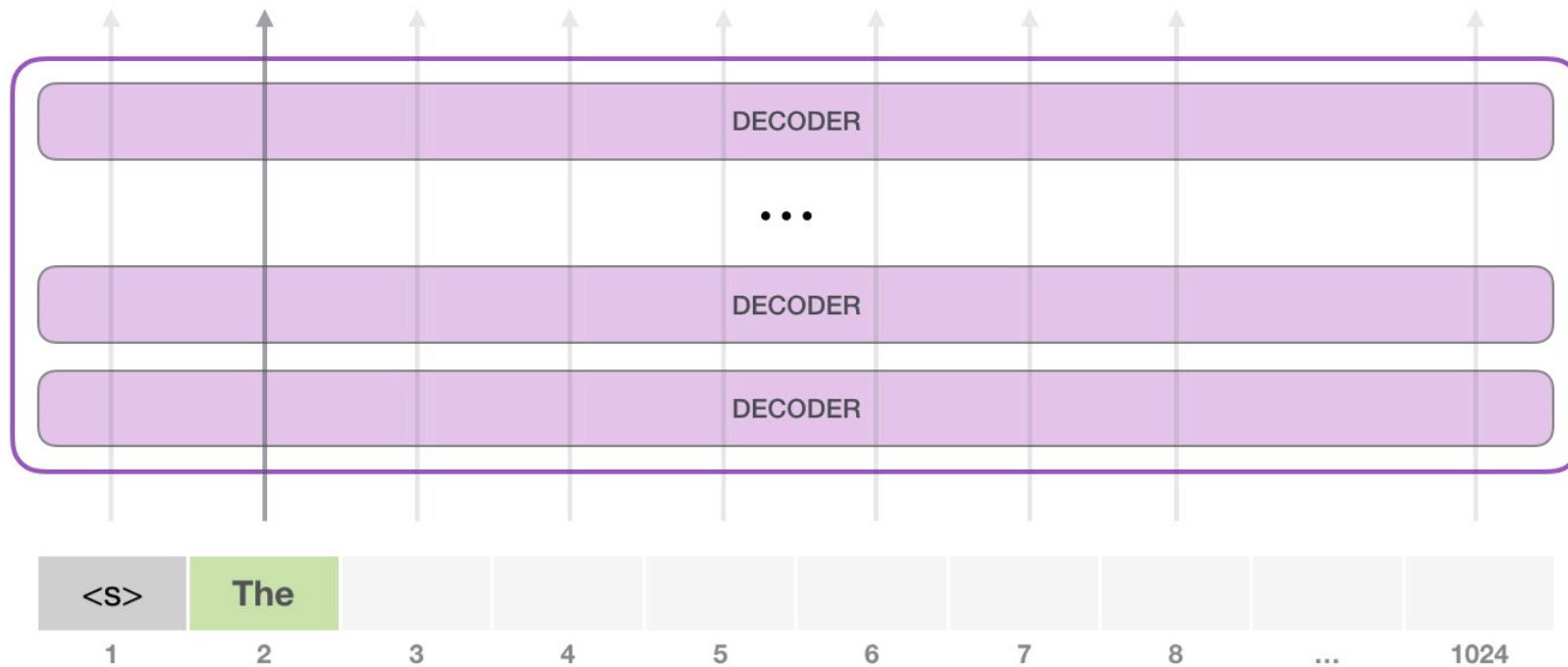
# LLMs 大模型推理过程

- LLMs 大模型一次推理只输出一个 token，输出 token 会与输入 tokens 拼接一起，作为下一次推理的输入，不断反复直到遇到终止符。



# LLMs 大模型推理过程

- LLMs 大模型一次推理只输出一个 token，输出 token 会与输入 tokens 拼接一起，作为下一次推理的输入，不断反复直到遇到终止符。



# 使用 Huggingface 的 GPT2 模型进行推理

```
5 model = GPT2LMHeadModel.from_pretrained("/WORK/Test/gpt", torchscript=True).eval()
6
7 # tokenizer
8 tokenizer = GPT2Tokenizer.from_pretrained("/WORK/Test/gpt")
9 in_text = "Lionel Messi is a"
10 in_tokens = torch.tensor(tokenizer.encode(in_text))
11
12 # inference
13 token_eos = torch.tensor([198]) # line break symbol
14 out_token = None
15 i = 0
16 with torch.no_grad():
17     while out_token != token_eos:
18         logits, _ = model(in_tokens)
19         out_token = torch.argmax(logits[-1, :], dim=0, keepdim=True)
20         in_tokens = torch.cat((in_tokens, out_token), 0)
21         text = tokenizer.decode(in_tokens)
22         print(f'step {i} input: {text}', flush=True)
23         i += 1
24
25 out_text = tokenizer.decode(in_tokens)
26 print(f' Input: {in_text}')
27 print(f'Output: {out_text}'')
```



# 使用 Huggingface 的 GPT2 模型进行推理

```
1
2 step 0 input: Lionel Messi is a player
3 step 1 input: Lionel Messi is a player who
4 step 2 input: Lionel Messi is a player who has
5 step 3 input: Lionel Messi is a player who has been
6 step 4 input: Lionel Messi is a player who has been a
7 step 5 input: Lionel Messi is a player who has been a key
8 step 6 input: Lionel Messi is a player who has been a key part
9 step 7 input: Lionel Messi is a player who has been a key part of
10 step 8 input: Lionel Messi is a player who has been a key part of the
11 step 9 input: Lionel Messi is a player who has been a key part of the team
12 step 10 input: Lionel Messi is a player who has been a key part of the team's
13 step 11 input: Lionel Messi is a player who has been a key part of the team's success
14 step 12 input: Lionel Messi is a player who has been a key part of the team's success.
15 step 13 input: Lionel Messi is a player who has been a key part of the team's success.
16
17 Input: Lionel Messi is a
18 Output: Lionel Messi is a player who has been a key part of the team's success.
19
```

# 推理过程

- 推理过程中每 step 输入一个 token 序列，经过 Embedding 层将输入 token 序列变为一个三维张量  $[b, s, h]$ ，经过 Transformers 网络模型计算，最后 logits 层将计算结果映射至词表空间，输出张量维度为  $[b, s, V]$ 。
- 当前轮  $i$  输出 token 与输入 tokens 拼接，并作为下一轮的输入 tokens，反复到终止符。第  $i + 1$  轮输入只比第  $i$  轮输入数据新增一个 token，其他相同。因此第  $i + 1$  轮推理必然包含第  $i$  轮部分计算。

# 推理过程

- 推理过程中每 step 输入一个 token 序列，经过 Embedding 层将输入 token 序列变为一个三维张量  $[b, s, h]$ ，经过 Transformers 网络模型计算，最后 logits 层将计算结果映射至词表空间，输出张量维度为  $[b, s, V]$ 。
  - 当前轮  $i$  输出 token 与输入 tokens 拼接，并作为下一轮的输入 tokens，反复到终止符。第  $i + 1$  轮输入只比第  $i$  轮输入数据新增一个 token，其他相同。因此第  $i + 1$  轮推理必然包含第  $i$  轮部分计算。
- KV Cache 的原理在于缓存当前轮  $i$  可重复利用的计算结果， $i + 1$  轮计算时直接读取缓存结果。



# 4. KV Cache 原理

# 推理过程

- Transformers 结构里的 keys 和 values 通常被称为 KV，这些 tensors 会存在 NPU 显存中，用于生成下一个 token；
- 这些 KV 中间值占用很大的内存，而且大小动态变化，导致有接近 60%-80% 预留显存（Workspaces）浪费；
- 因此 Transformers 结构的推理加速常用的优化手段使用 KV Cache 算法，LLMs 大模型生成式推理包含两个阶段：1) 预填充 Cache 阶段；2) 使用 KV Cache 阶段。

# 思考

1. KV Cache 节省了 Self-Attention 层中哪部分的计算？
2. KV Cache 对 MLP 层计算量有影响吗？
3. KV Cache 对 Transformers 结构间的数据传输量有影响吗？

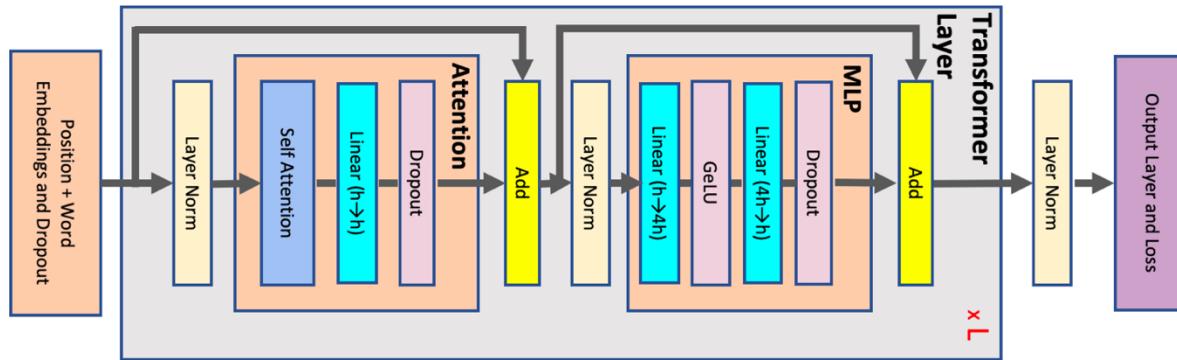
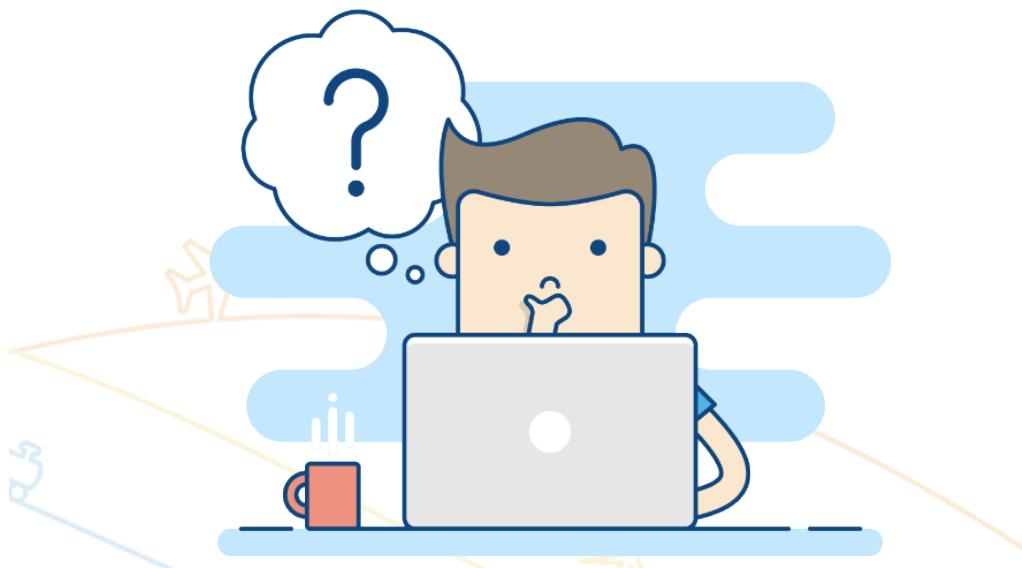
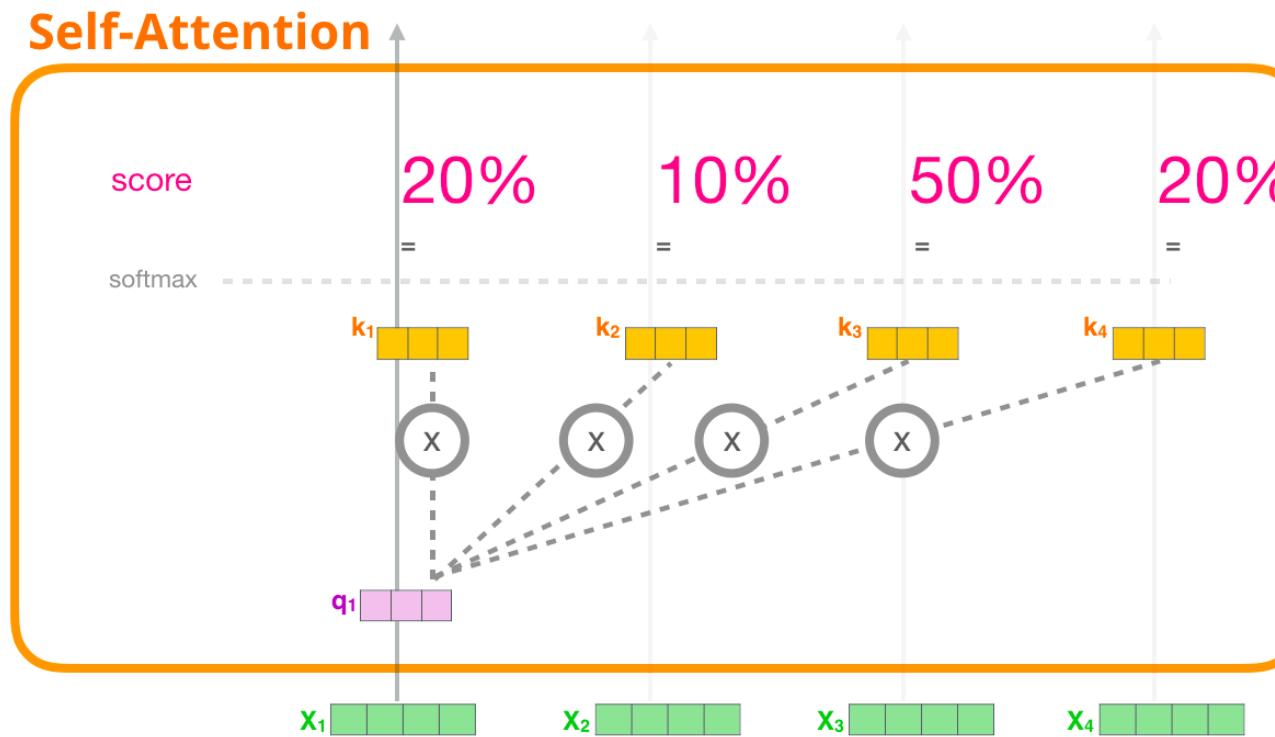


Figure 2: Transformer Architecture. Each gray block represents a single transformer layer that is replicated  $L$  times.



# Self-attention 自注意力层

- LLMs 大模型生成式推理包含两个阶段：1 ) 预填充 Cache 阶段；2 ) 使用 KV Cache 阶段。
  - **预填充阶段**：输入一个 prompt 序列，为每个 transformer 层生成 key cache 和 value cache，即 KV cache。
  - **解码阶段**：使用并更新 KV cache，一个接一个地生成词，当前生成的词依赖于之前已经生成的词。



# KV Cache 原理 I

- **预填充阶段**：输入一个 prompt 序列，计算第一个输出 token 过程中，这时 Cache 是空的，为每个 transformer 层生成 key cache 和 value cache，即 KV cache，在输出 token 时第一次 KV Cache 完成填充。
- **具体计算**（FLOPs 计算过程中存在大量 GEMM 操作，推理速度相对较慢）
  - 假设第  $i$  个 Transformers 层输入为  $x^i$
  - self-attention 中 key、value、query 和 output 分别表示为  $x_K^i, x_V^i, x_Q^i, x_O^i, x^i \in R^{b \times s \times h}$
  - NPU 显存缓存下 key cache 和 value cache，即  $x_K^i = x^i \cdot W_K^i, x_V^i = x^i \cdot W_V^i$
  - 最后，计算第  $i$  个 Transformers 层剩余的计算过程（传统 Transformers 结构的计算）

## KV Cache 原理 II

- **解码阶段**：计算第二个输出 token 至最后一个 token，此时 Cache 有值，接下来是使用并更新 KV cache，每轮推理读取缓存中的 KV Cache，一个接一个地生成词，当前生成的词依赖于之前已经生成的词，同时将当前轮输出的新 Key、Value 追加写入至缓存 Cache 中。
- **计算量** ( FLOPs降低，GEMM 变为 GEMV 操作，推理速度相对第一阶段变快 )
  - 假设当前生成词在第  $i$  个 Transformers 层的向量表示为  $t^i$ ， $t \in R^{b \times 1 \times h}$ ；
  - 更新 key cache 和 value cache 为即  $x_K^i = \text{Concat}(x_K^{i-1}, t^i \cdot W_K^i)$ ， $x_V^i = \text{Concat}(x_V^{i-1}, t^i \cdot W_V^i)$
  - 最后，计算第  $i$  个 Transformers 层的输出（传统Transformers结构的计算）

# KV cache 显存占用分析

- 在推理的过程中，以 FPI6 保存 KV cache，那么峰值显存占用大小为：

$$b(s + n) \times h \times l \times 2 \times 2 = 4blh(s + n)$$

- 第一个 2 分别表示 K cache 和 V Cache
- 第二个 2 表示 FPI6 占用 2 个 Bytes



# KV cache 显存占用分析

- 在推理的过程中，以 FPI6 保存 KV cache，那么峰值显存占用大小为：

$$b(s + n) \times h \times l \times 2 \times 2 = 4blh(s + n)$$

- 其中，第一个 2 分别表示 K cache 和 V Cache
  - 第二个 2 表示 FPI6 占用 2 个 Bytes
- 
- e.g.，GPT3 175B 模型占用显存为 ~340 GB
    - $b = 64, s = 512$ , 输出序列长度为 32
    - KV Cache 为  $4blh(s + n) = 164282499072$  Bytes  $\approx 164GB$ ，显存占用约为模型 0.5X。



# 小结 & 思考



# 小结

- 回顾了大模型训练显存占用主要由模型参数、梯度、优化器状态和中间激活值 4 部分组成。
- 了解大模型推理显存仅由模型参数和 KV Cache 组成，推理性最大瓶颈在于显存。
- 根据代码回顾大模型推理的过程中，tokens 是反复迭代生成的。
- 深入了解 KV Cache 的算法原理，整体占用模型参数显存的 0.5X。



# 引用 & 参考

1. Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM
2. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism
3. Reducing Activation Recomputation in Large Transformer Models
4. A Survey of Large Language Models
5. <https://medium.com/@amirhossein.abaskohi/text-to-text-models-in-natural-language-processing-f203afb34a8b>
6. [https://twitter.com/cwolferearch...](https://twitter.com/cwolferesearch/status/1649476518811148314)
7. <https://mdnice.com/writing/ce291e46450e415abd0c71f7282f3f20>
8. <https://medium.com/@ngiengkianyew/multi-headed-attention-8b940b76c351>
9. <https://stats.stackexchange.com/questions/620002/why-is-the-layer-normalization-same-with-the-instance-normalization-in-transformer>
10. <https://www.kaggle.com/code/gabedossantos/killer-bert-tutorial>
11. <https://zhuanlan.zhihu.com/p/630832593>
12. <http://jalammar.github.io/illustrated-gpt2/>
13. <http://jalammar.github.io/how-gpt3-works-visualizations-animations/>
14. <http://jalammar.github.io/illustrated-gpt2/>





# Thank you

把AI系统带入每个开发者、每个家庭、  
每个组织，构建万物互联的智能世界

Bring AI System to every person, home and  
organization for a fully connected,  
intelligent world.

Copyright © 2023 XXX Technologies Co., Ltd.  
All Rights Reserved.

The information in this document may contain predictive statements including, without limitation, statements regarding the future financial and operating results, future product portfolio, new technology, etc. There are a number of factors that could cause actual results and developments to differ materially from those expressed or implied in the predictive statements. Therefore, such information is provided for reference purpose only and constitutes neither an offer nor an acceptance. XXX may change the information at any time without notice.



Course [chenzomi12.github.io](https://chenzomi12.github.io)

GitHub [github.com/chenzomi12/DeepLearningSystem](https://github.com/chenzomi12/DeepLearningSystem)