vllm入门

参考：b站 RethinkFun

VLLM 是一个用于LLM**推理**的高性能库，适用于需要快速响应和高吞吐量的大规模应用场景。可以比直接调用模型的推理速度快24倍。

回顾一下之前学习过的**kv cache**：

推理阶段最常用的一种缓存机制，本质上是用空间换时间。

原理：

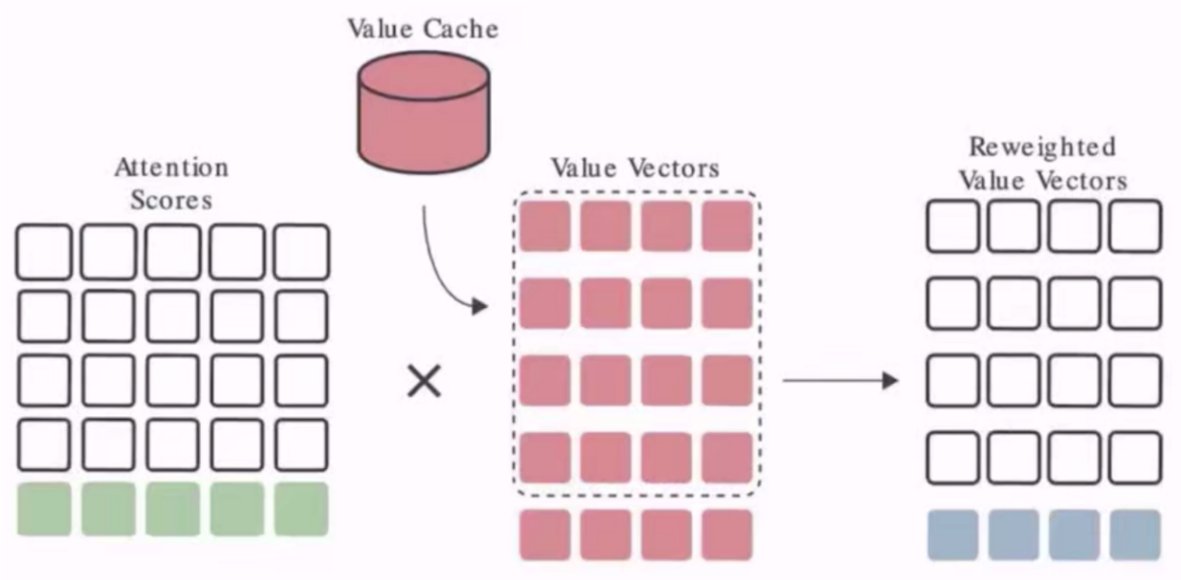
在进行自回归解码的时候，新生成的token会加入序列，一起作为下一次解码的输入。

由于单向注意力的存在，新加入的token并不会影响前面序列的计算，因此可以把已经计算过的每层的kv值保存起来，这样就节省了和本次生成无关的计算量。

通过把kv值存储在速度远快于显存的L2缓存中，可以大大减少kv值的保存和读取，这样就极大加快了模型推理的速度。

分别做一个k cache和一个v cache，把之前计算的k和v存起来

以v cache为例：

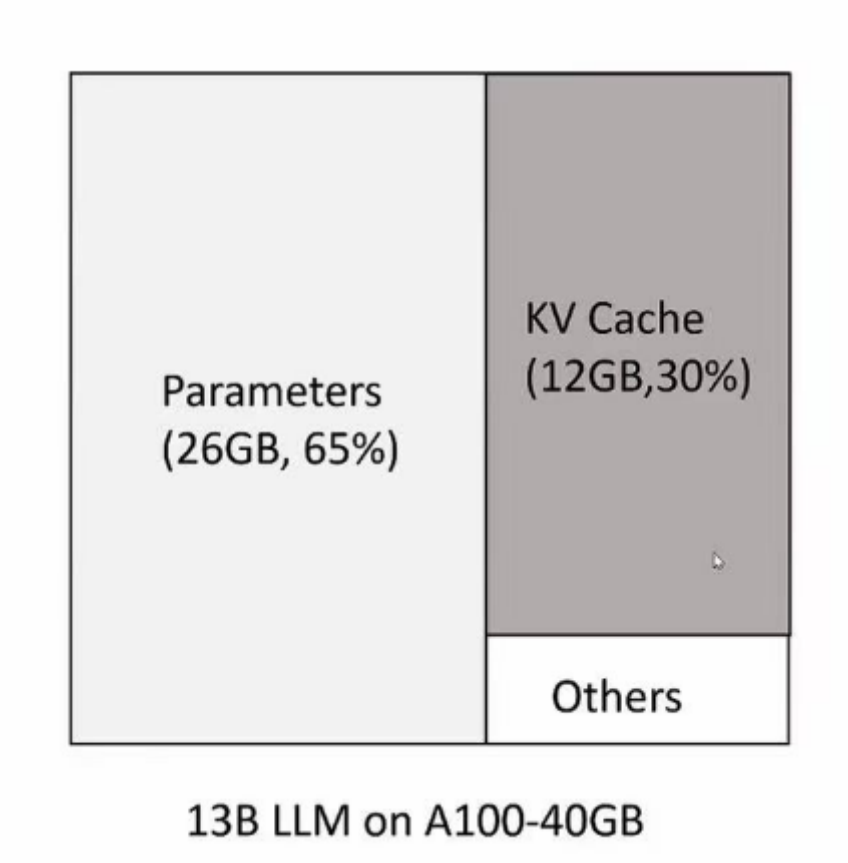


这样LLM在回答时只需要计算当前token的k、v，前面token的k、v都缓存了，因此随着模型输出的不断变长，我们也没有感觉到速度变慢。

kv cache存在的问题

以13B的模型为例：

（others表示forward过程中产生的activation的大小，这些activation可以认为是用完则废的，因此它们占据的显存不大）



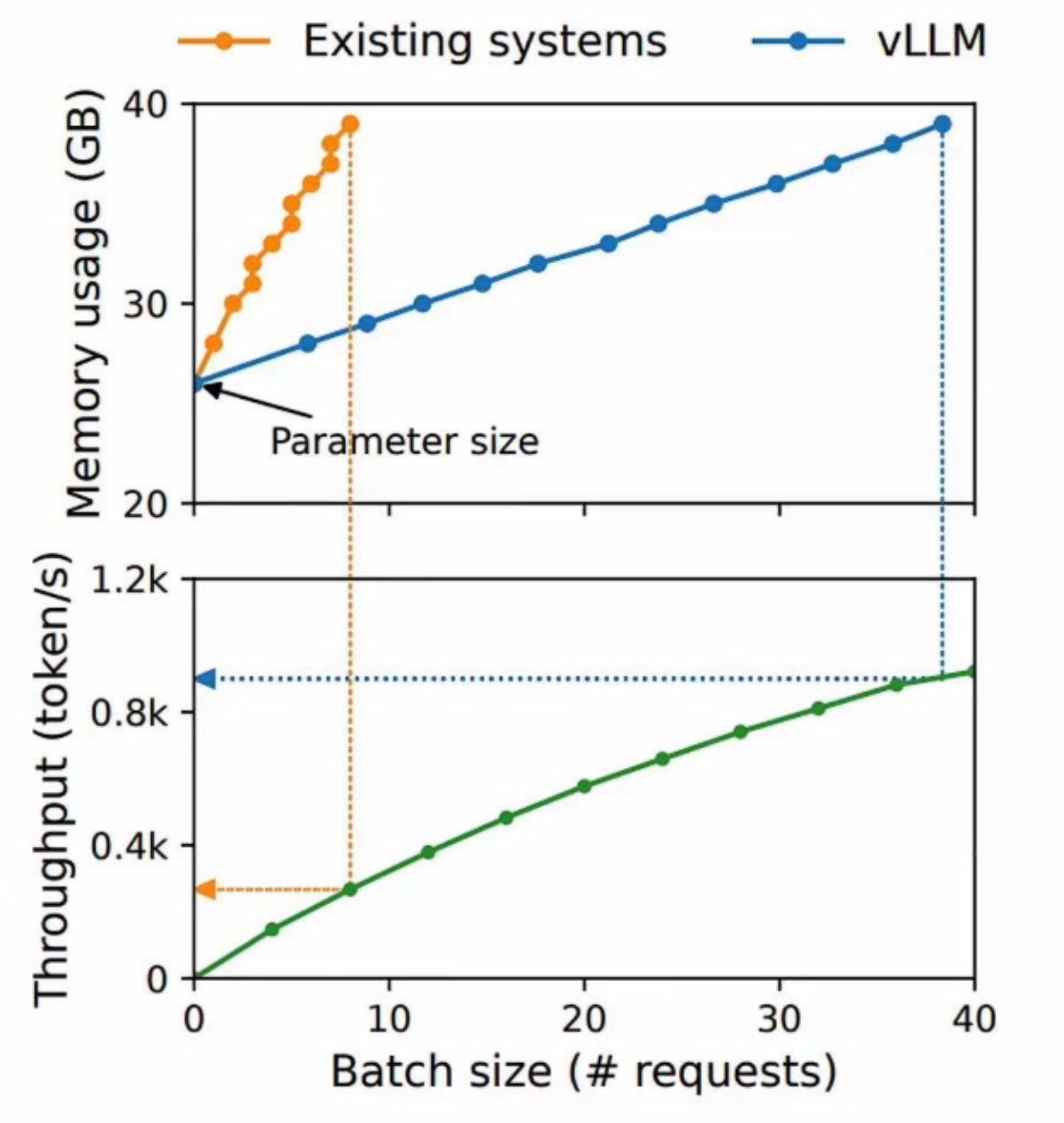
但kv cache实际的**显存利用率只有20%-40%**，大部分都被浪费了：

1、LLM在推理时，总是按照可生成最长序列长度预分配显存。但实际上可能并不会用到那么长。

2、LLM在推理时，当生成第一个token时，剩下的显存已经被预分配了，这就导致无法同时响应其他的请求，不能并行处理多个请求

3、输入的prompt的长度不同，会导致显存之间的间隔碎片，而这些碎片又不足以预分配给下一个文本生成

vllm就是用来解决kv cache中存在的显存浪费问题，用更大的batch size来处理请求，从而提高了系统的吞吐量：



从图中可以看出，原本的batch size为8，优化后batch size为40；原本响应速度是300token/s，优化后响应速度是900token/s。

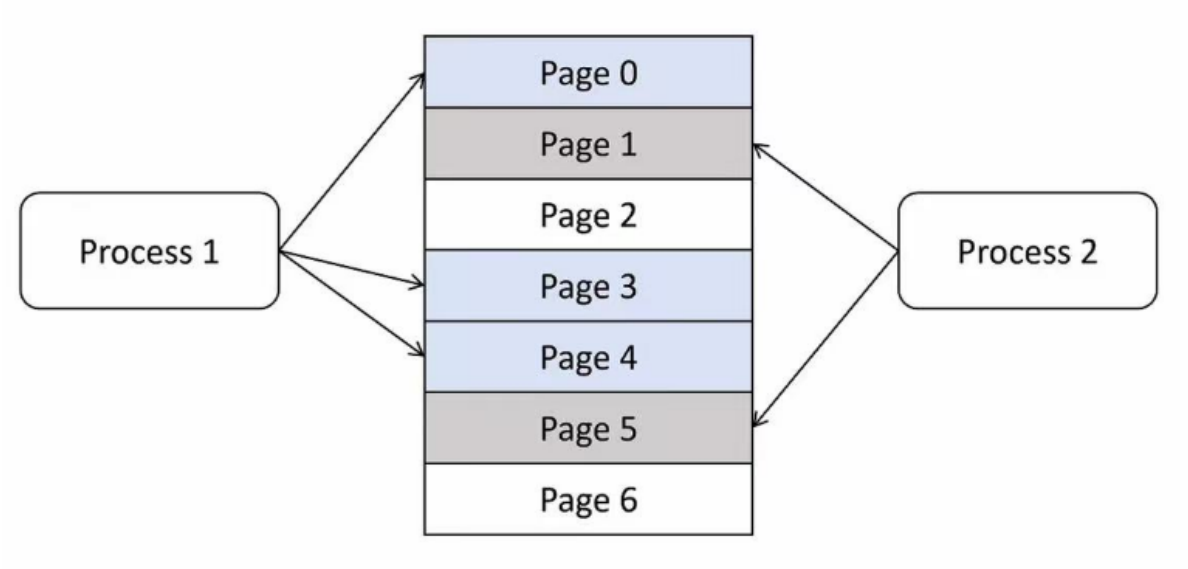
vllm优化的点：

Page Attention

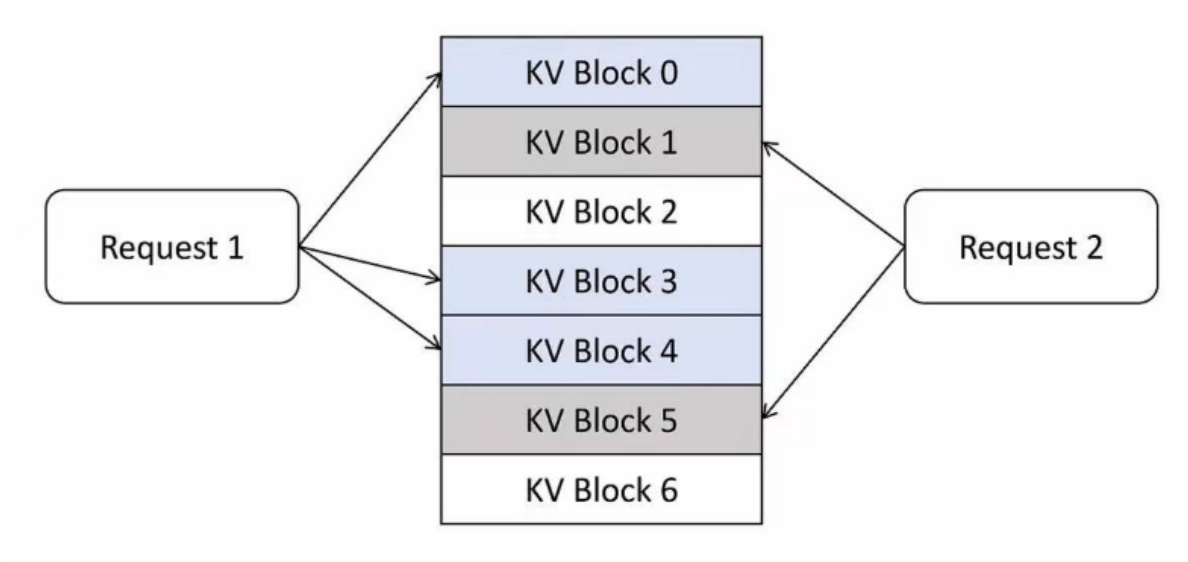
借鉴了操作系统中的虚拟内存和页管理技术：

操作系统给每个程序怎么分配内存？要不要预分配内存？程序关闭后怎么回收内存？内存碎片怎么处理？怎么管理内存？

以页为最小单位来分配内存，物理内存被划分为很多页。



类似的，page attention也把显存划分为kv block：

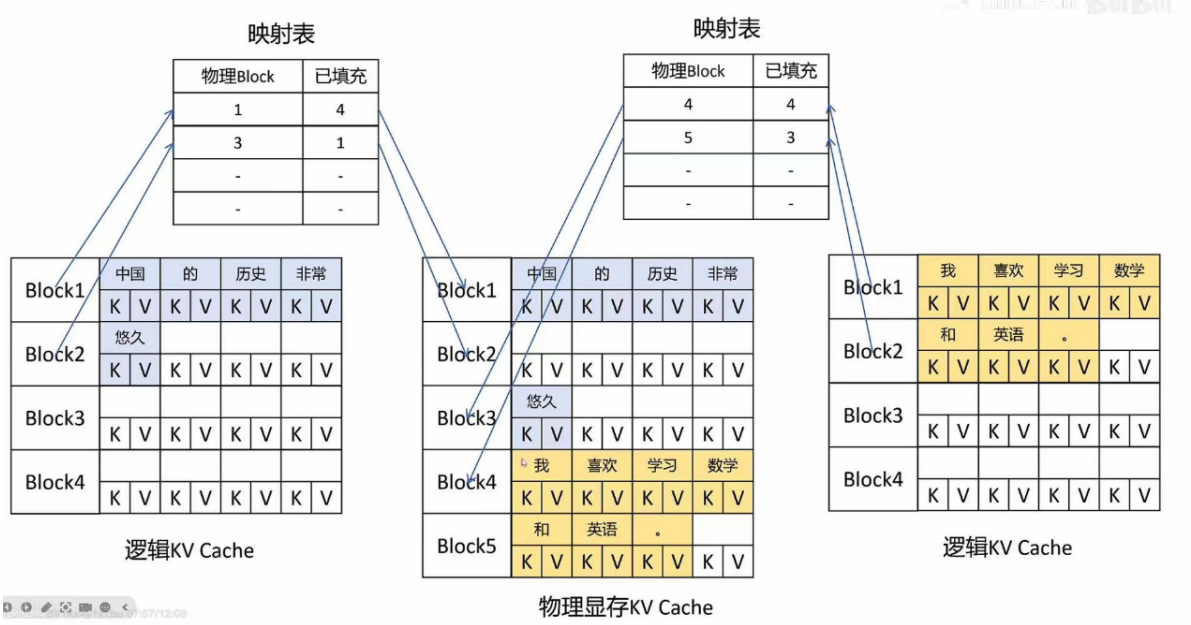


例如，每个kv block可以缓存4个token的k、v向量

1、按需分配，不提前预分配

2、按block分配，解决了显存碎片的问题（因为碎片最大也只有3个token的大小）

3、虚拟内存：维护了逻辑kv cache，让系统看上去是使用的连续的显存，通过映射表来实现，方便block的调用。



通过page attention的优化，kv cache的显存利用率从20%-40%提升到了96%。

Sharing KV Blocks

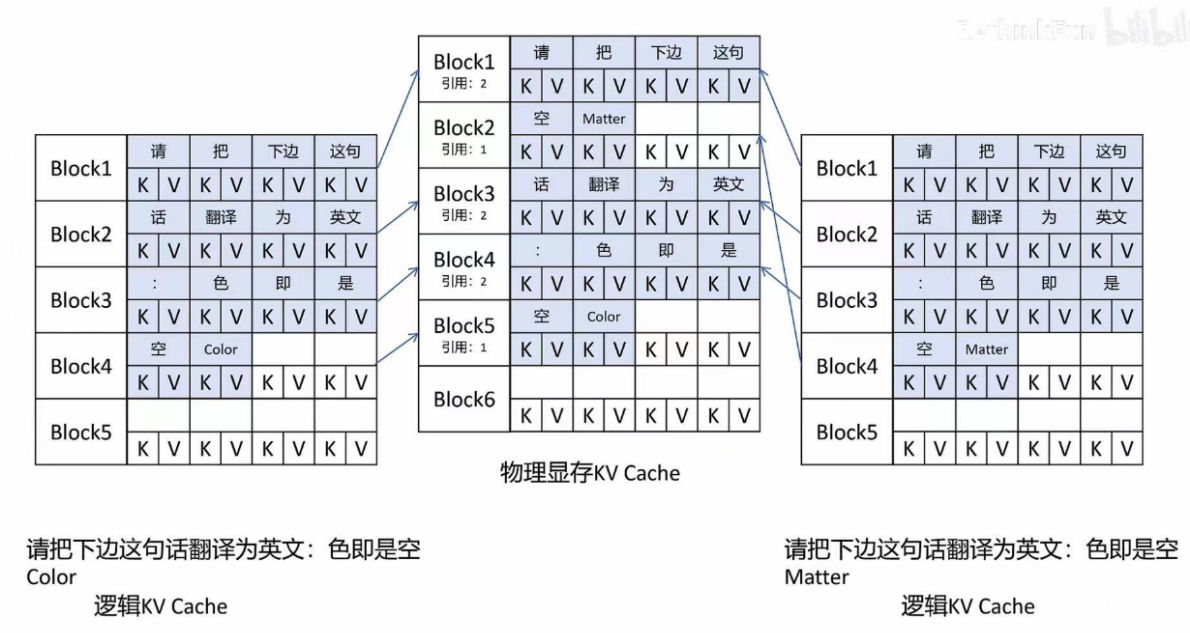
主要在一下两种场景中生效：

1、用LLM同一个prompt，希望生成多个Output时

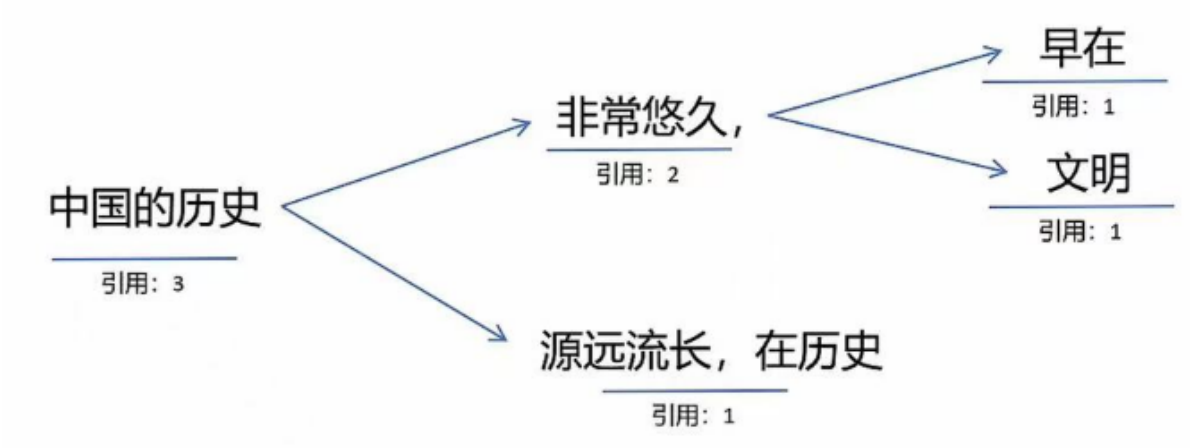
Prompt：

请把下面这句话翻译为英文：{item}

假设要生成2个Output，设置n=2，prompt在kv cache中只存放了一份，每个block标记引用数为2，只有当引用数为0时，这个显存所占用的block才会被释放。



2、优化beam search里的显存占用



代码示例

```
from vllm import LLM, SamplingParams
prompts = [
    "Hello, a nice day",
    "What a perfect solution",
    "AIGC developed rapidly",
]
sampling_params = SamplingParams(temperature=0.8, top_p=0.95, max_tokens=10)
llm = LLM(model="qwen1.5-13B-chat")
outputs = llm.generate(prompts, sampling_params)

for output in outputs:
    prompt = output.prompt
    generated_text = output.outputs[0].text
    print(f"Prompt:{prompt!r}, Generated text:{generated_text!r}")
```