泛在计算与智能系统研究中心
Research Center of Ubiquitous Computing and Intelligent Systems
哈爾濱工業大學
Harbin Institute of Technology

论文分享

SpecInfer:Accelerating Generative Large Language Model Serving with Tree-based Speculative Inference and Verification

ASPLOS 2024

乔浩宇

2024.12.31

前置知识

LLM的推理过程

以一组对话在Llama-2-7b中的推理为例：

Human: remember the boy names Tom is five years old
Assistant: ok,I remember
Human: how old is Tom
Assistant:

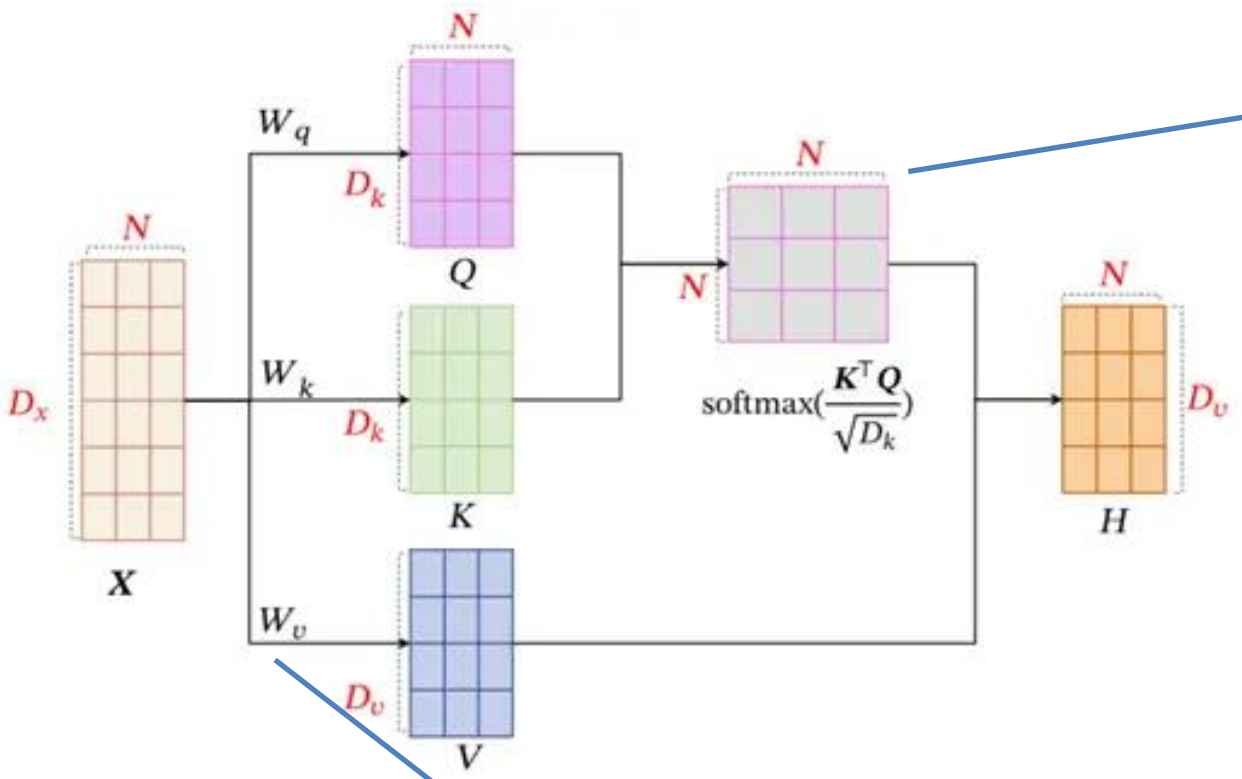
1. **嵌入**：将句子嵌入到向量空间，例如对话包含30个词元(token)，根据查词表将每个token映射成一个长度为4096的向量
2. **transformer layers**：将嵌入后的矩阵交给transformer-decoder执行矩阵乘等计算，Llama-2-7b有32层，也就是32个decoder拼在一起，每个层的输入是上一层的输出
3. **预测**：transformer layers计算结果的最后一行蕴含了对下一个token的预测信息，把这个长度为4096的向量通过全连接层映射回词表大小的长度。每个位置的数值表示对该位置token的预测概率

前置知识

LLM的推理过程-自注意力计算

Attention的计算：

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



N*N的矩阵
第(i,j)个元素表示该序列第i个token对第j个token的注意力分数

输入矩阵的大小是N*4096
三个权重矩阵，每个矩阵的大小都是4096*4096
因此输入和权重矩阵相乘后的大小仍然是N*4096

前置知识

LLM的推理过程-KV缓存

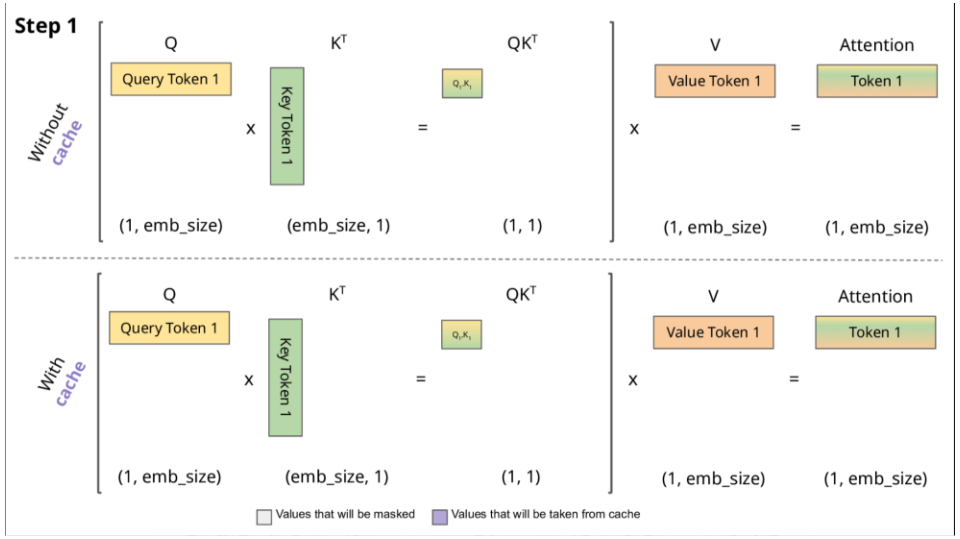
LLM的推理是自回归的
某次推理中，30个token作为输入，得到对下一个token的预测；那么下次推理的输入就是30+1个token

这会在自注意力过程中产生重复计算：
回顾计算QKV的过程，每次计算出的KV矩阵，前面的N-1行都是和上一次推理重复的
不如每次计算得到KV之后，都把它们存到显存里，那么下次推理的时候只需要计算新的一行的KV就可以了

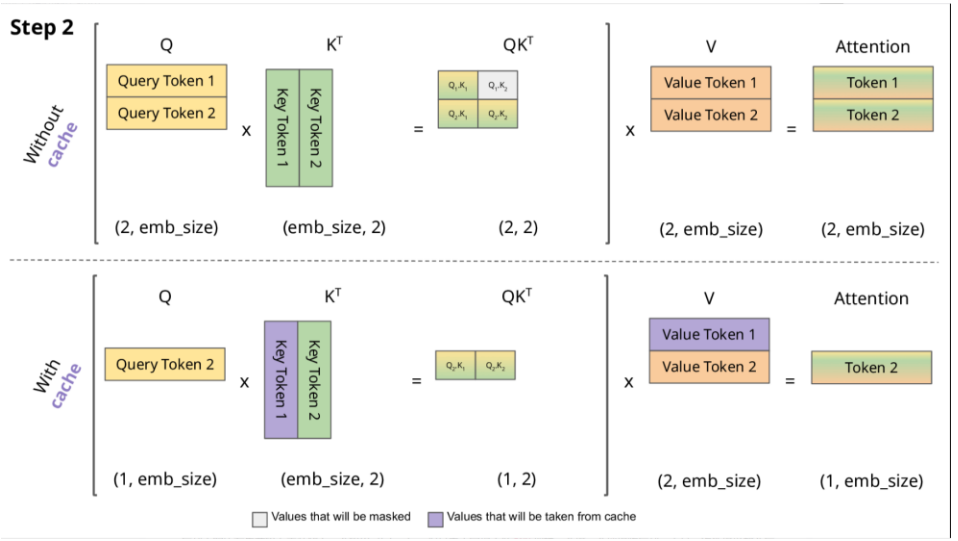
前置知识

LLM的推理过程-KV缓存

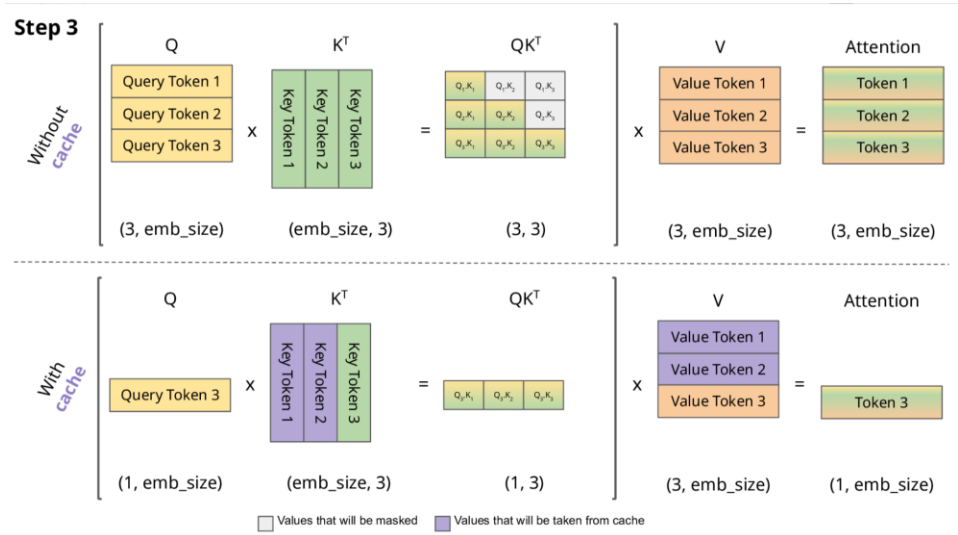
1



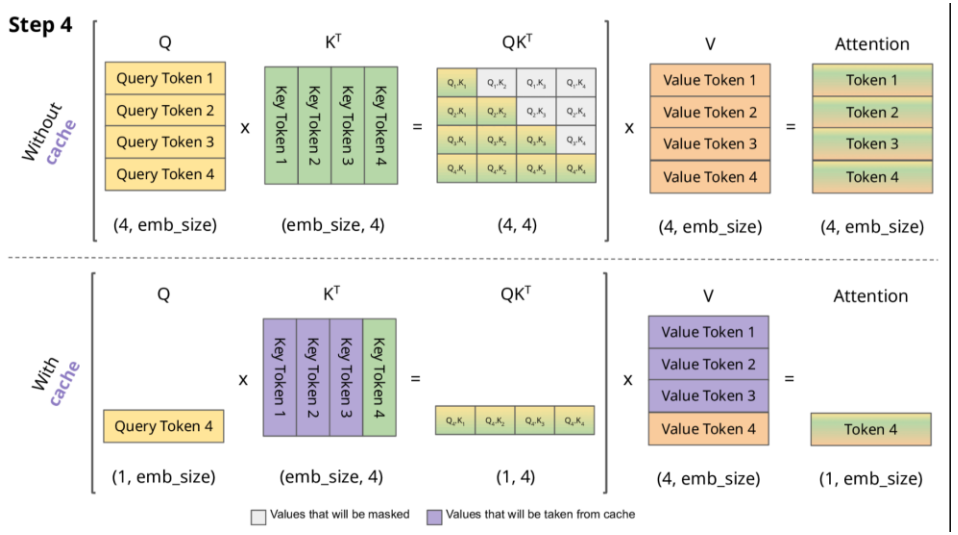
2



3



4



投机解码

动机

- 基于前面介绍的KV缓存，可以将LLM生成一个新的序列的过程分成两个阶段：预填充prefill和解码decode
  预填充阶段，就是将用户的输入和上下文的若干个token一起交给LLM执行一次推理，推理过程中缓存所有token的KV
  解码阶段，将上一次推理得到的一个token作为输入执行一次推理，推理过程中缓存这个token的KV

- 于是，在decode阶段很容易产生内存瓶颈问题：
  GPU核心想要执行计算只能从SRAM中读取数据，而模型参数存在显存HBM中。每一次推理都要把把参数从显存取到SRAM
  从显存中取参数的时间，远远长于矩阵乘法计算的时间

- 内存瓶颈被形容成memory-wall，在这个墙之内，能堆的有效计算越多越好
  理想的情况就是，将完整的参数取一次，能够生成很多个后续的token，于是就有了投机解码

投机解码

标准范式

为了只取一次参数、生成多个token，最初的想法是在训练的时候就使用非自回归的方式，比如一次推理直接生成后面的5个token，但这种方式效果不好。

投机解码的标准范式是： **draft-then-verify,先打草稿后验证**

- 用一个小模型(草稿模型)生成接下来若干个token作为草稿，然后让大模型(目标模型)来只取一次参数同时验证这些token是否可以被接受
- 比如：草稿模型的单次推理速度是目标模型的0.1，那么小模型连续推理5个token，之后目标模型验证一次，接受了前3个
- 总耗时1.5，生成了3个token，那么加速比就达到了2x

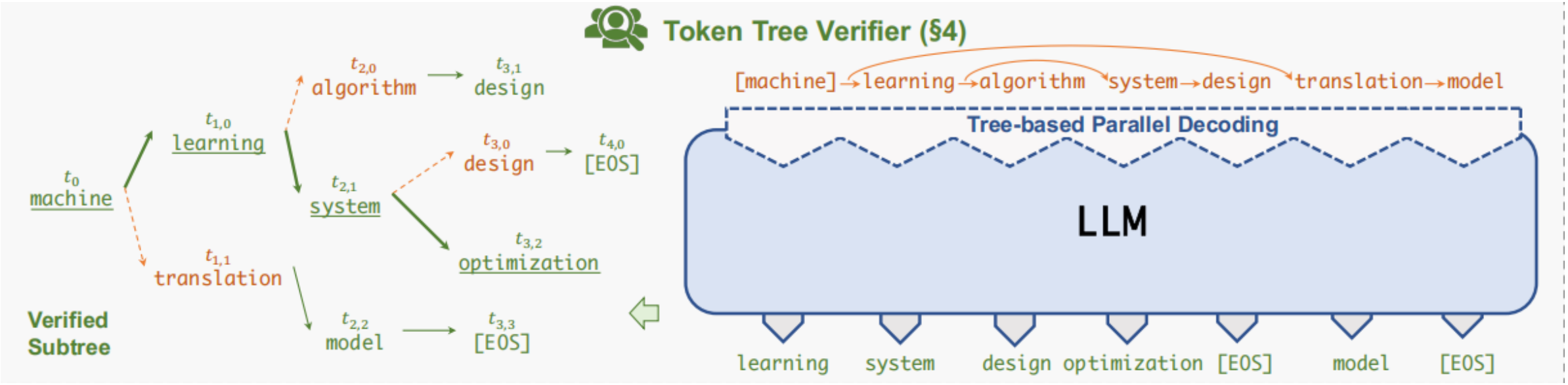
```
[START] japan ' s benchmark bond n
[START] japan ' s benchmark nikkei 22 ,5
[START] japan ' s benchmark nikkei 225 index rose 22 ,6
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 , points
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 0 1
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 9859
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 , in
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 in tokyo late
[START] japan ' s benchmark nikkei 225 index rose 226 . 69 points , or 1 . 5 percent , to 10 , 989 . 79 in late morning trading . [END]
```

SpecInfer

动机

让草稿模型一次只生成一个序列，效率还是有点低
以输入“machine”为例，草稿模型第一次推理之后，“learning”和”translation”的概率都比较高
如果只是选择概率最高的token拿去给目标模型验证，那么就可能造成潜在的浪费

于是可以把概率高的token都拿去给目标模型验证，提高接受概率
目标模型需要一次性验证整个token树，那怎么处理不在同一个序列中的不同token呢？

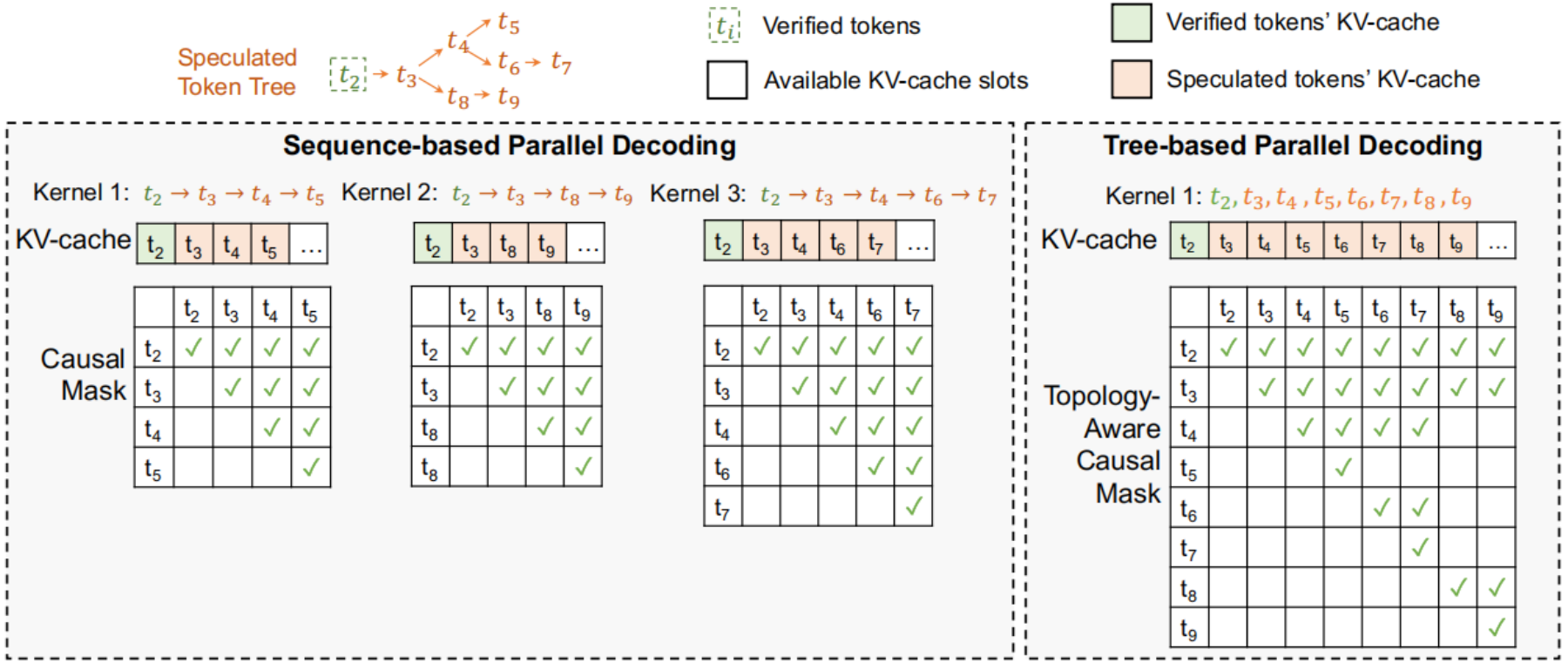


SpecInfer

tree attention

目标模型需要一次性验证整个token树，为了让每个token只能看到自己序列前面的token，不对其他序列中的token施加注意力
提出了tree attention，将不属于自己序列并且不在自己前面的token给遮盖掉
比如t7只对t2/t3/t4/t6和自身施加注意力，那么t5/t8/t9都被mask

下图左边是之前的验证方式，对每个序列单独进行注意力计算；右边是使用tree attention，一次验证整个树所有token



SpecInfer

其他

关于草稿模型生成序列，本文使用两种策略：

- 训练一个草稿模型，通过调节temperature使其生成不同的序列
- 训练多个草稿模型，通过控制训练数据使不同草稿模型的生成序列不同

控制训练数据的策略：

- 先微调完一个SSM，然后把SSM的输出与LLM输出相同的样本进行过滤；剩下的样本去微调下一个SSM。
- 保证每一个SSM的推理结果能尽可能少的重叠