Inference Basics

October 2nd, 2025

Jae-Won Chung

Jae-Won Chung

- Bio
  - 5th year PhD student working with Mosharaf
  - https://jaewonchung.me/about
- Background
  - 2017 – 2019: Machine learning, Computer vision, Meta-learning, Few-shot learning
  - 2019 – now: Systems for machine learning, Power & energy as first-class systems resources
- Three lectures
  - 08/29 (Thu) | Introduction to GenAI and Systems for GenAI
  - 09/04 (Thu) | Distributed training basics
  - 10/02 (Thu) | Inference basics

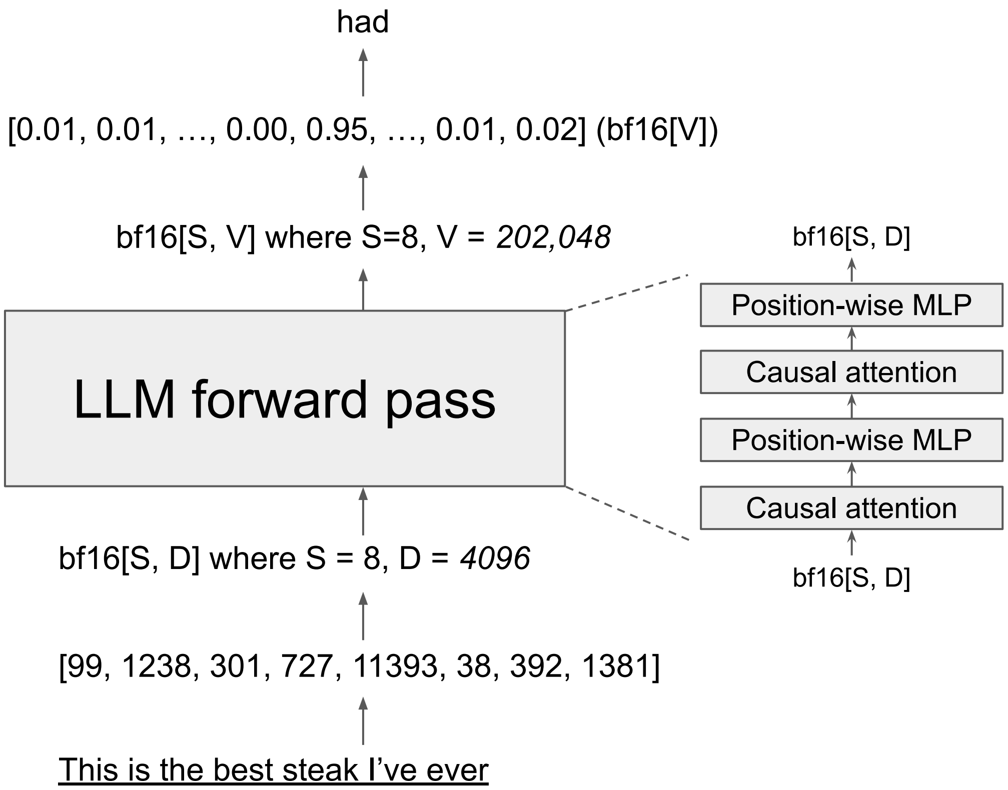
(Unofficial) Textbooks

- How to Scale Your Model (Google DeepMind)
  - Theoretical analysis of computations that are important to LLMs
  - Arithmetic intensity, compute- and memory-bound, roofline, back-of-the-envelope estimations
- The Ultra-Scale Playbook (Hugging Face)
  - Training LLMs on GPU Clusters
  - Various types of training parallelisms, implications on compute, memory, and communication
- Use as references
  - Each thing will take at least a week of full-time reading (it's worth it, though)
  - It's a fast-evolving field; the only way to be relevant is to continuously read

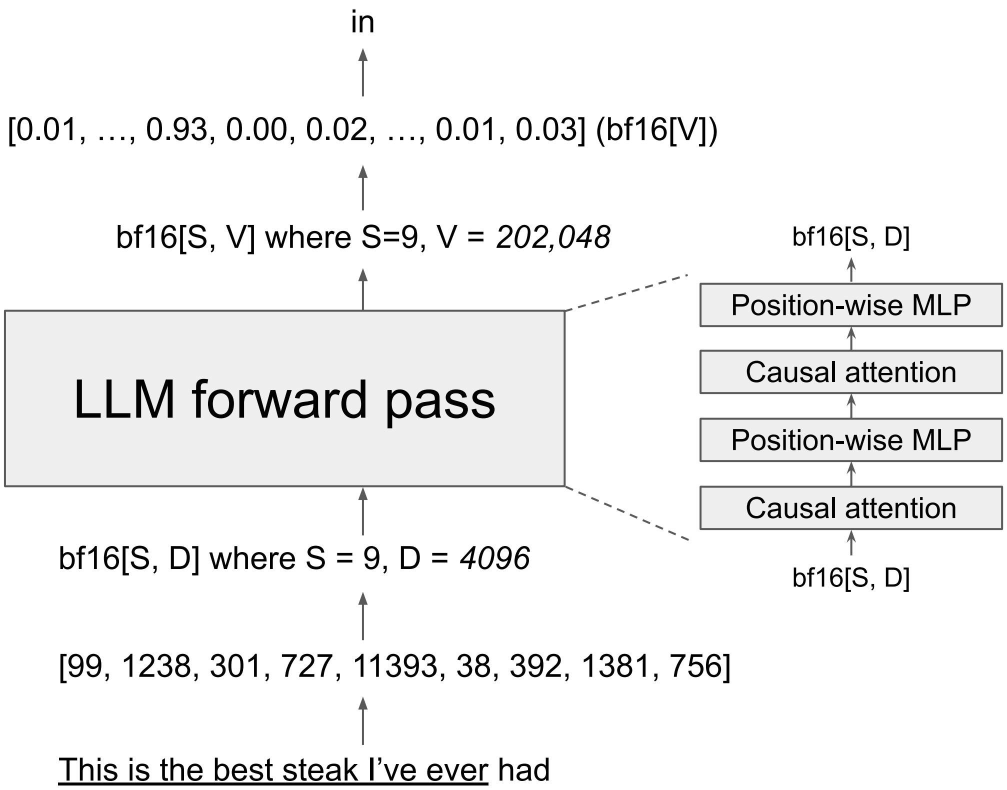
Today

- What are the computations?
- What do we want to optimize for?
- Essential optimizations and why they make sense

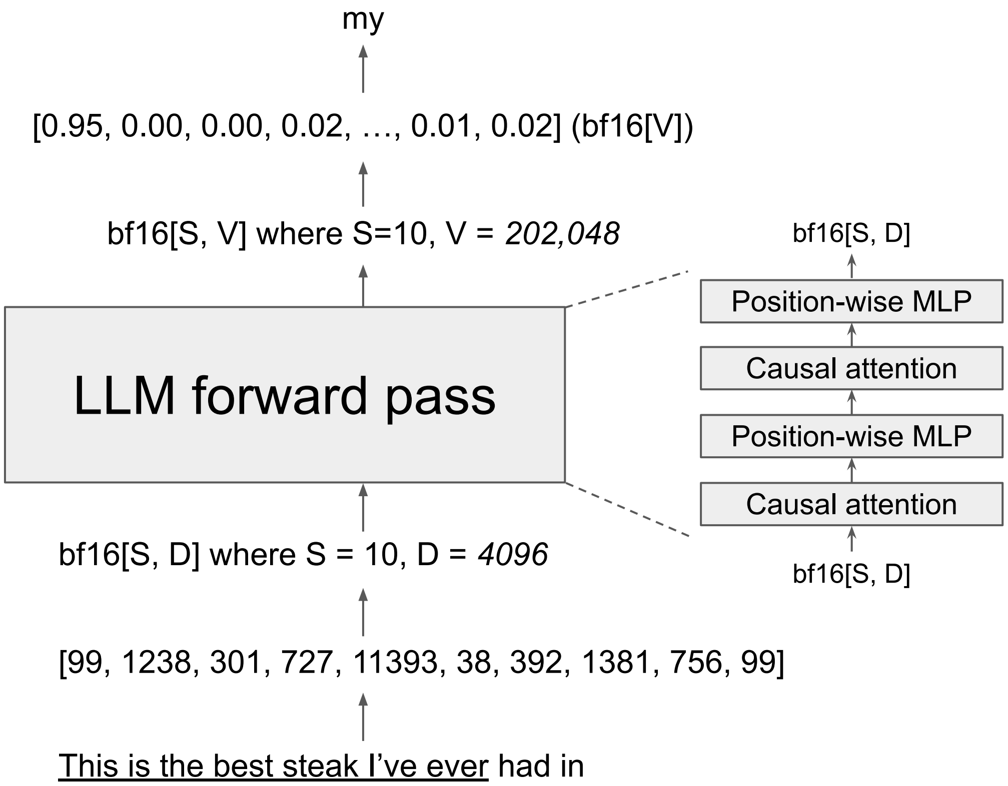
LLM Inference



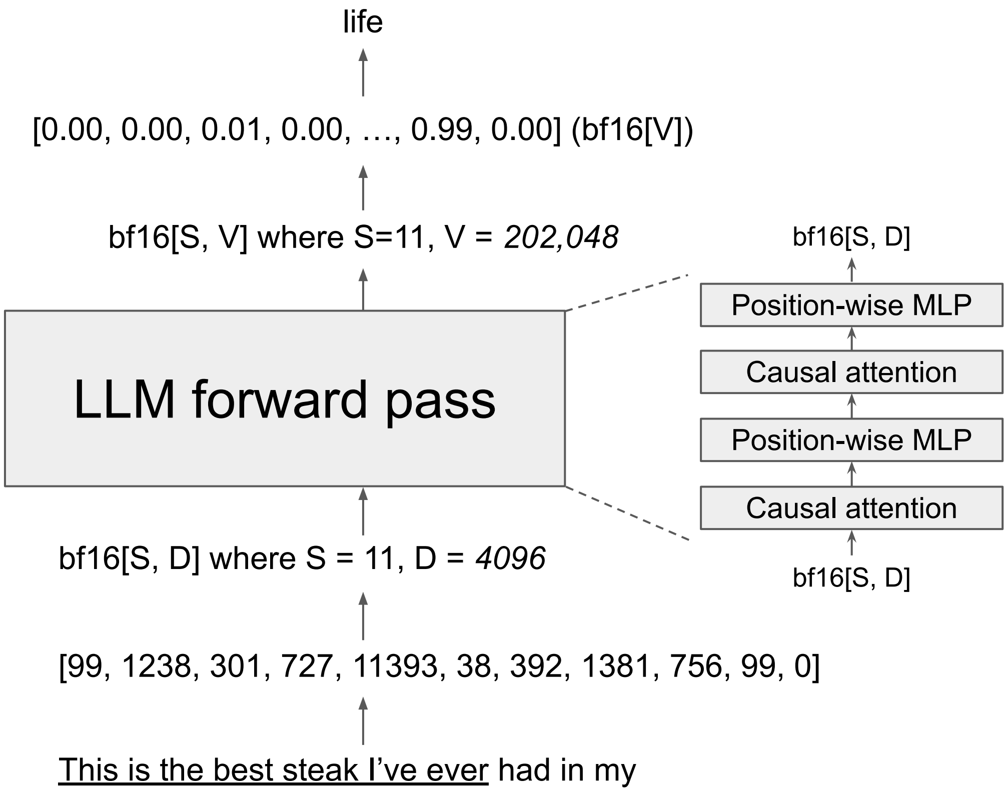
LLM Inference



LLM Inference

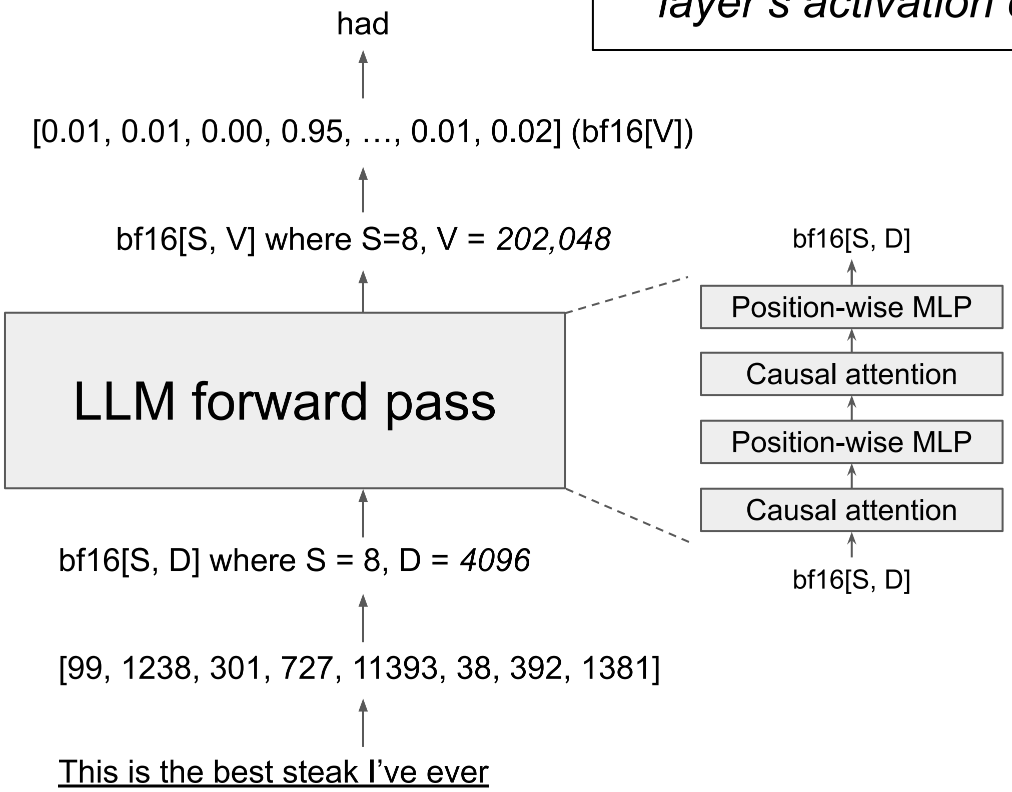


LLM Inference



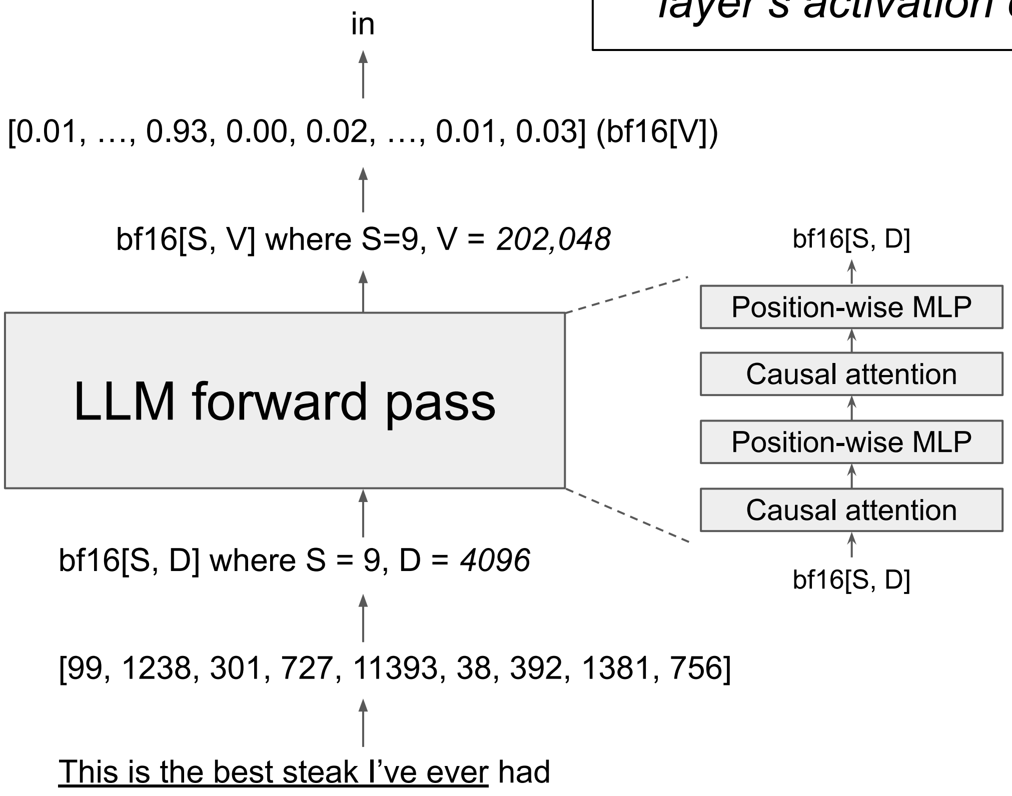
LLM Inference

Would the first 8 positions of each layer's activation change at all?



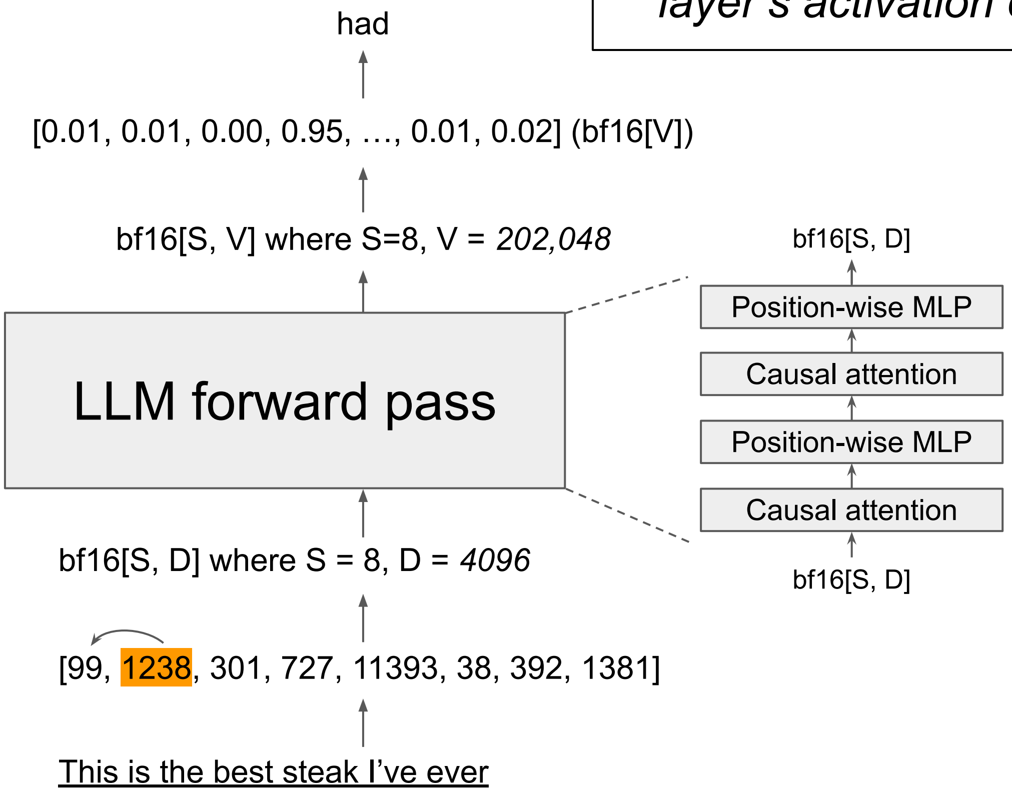
LLM Inference

Would the first 8 positions of each layer's activation change at all?



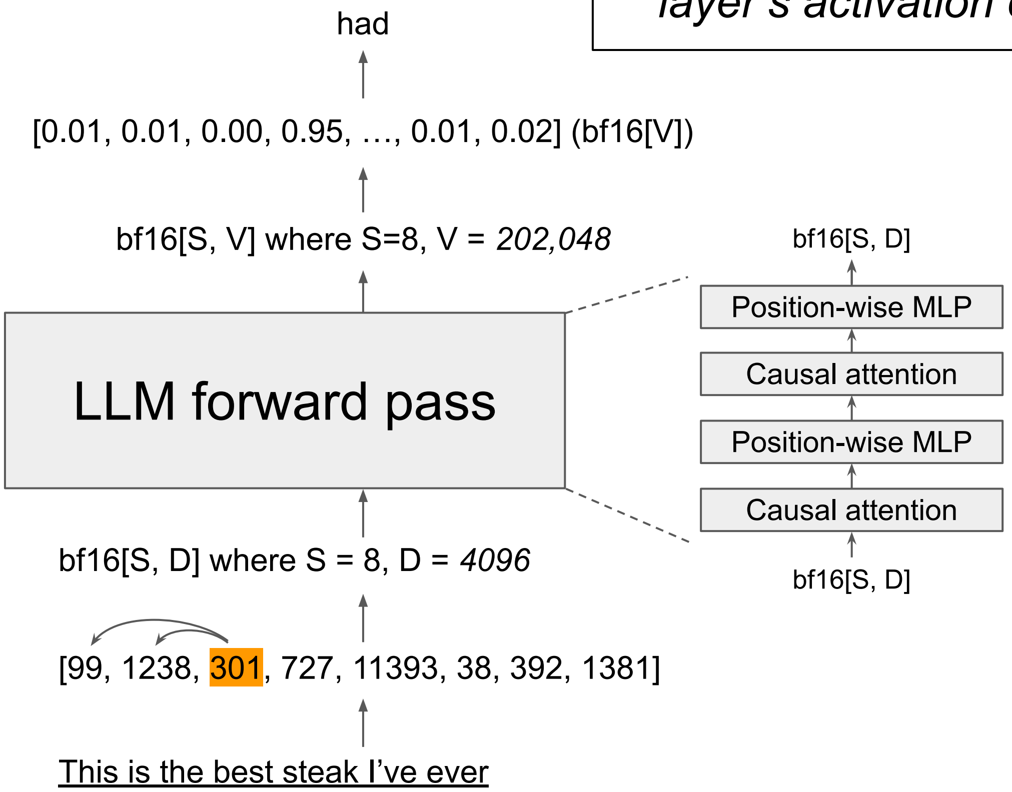
LLM Inference

Would the first 8 positions of each layer's activation change at all?



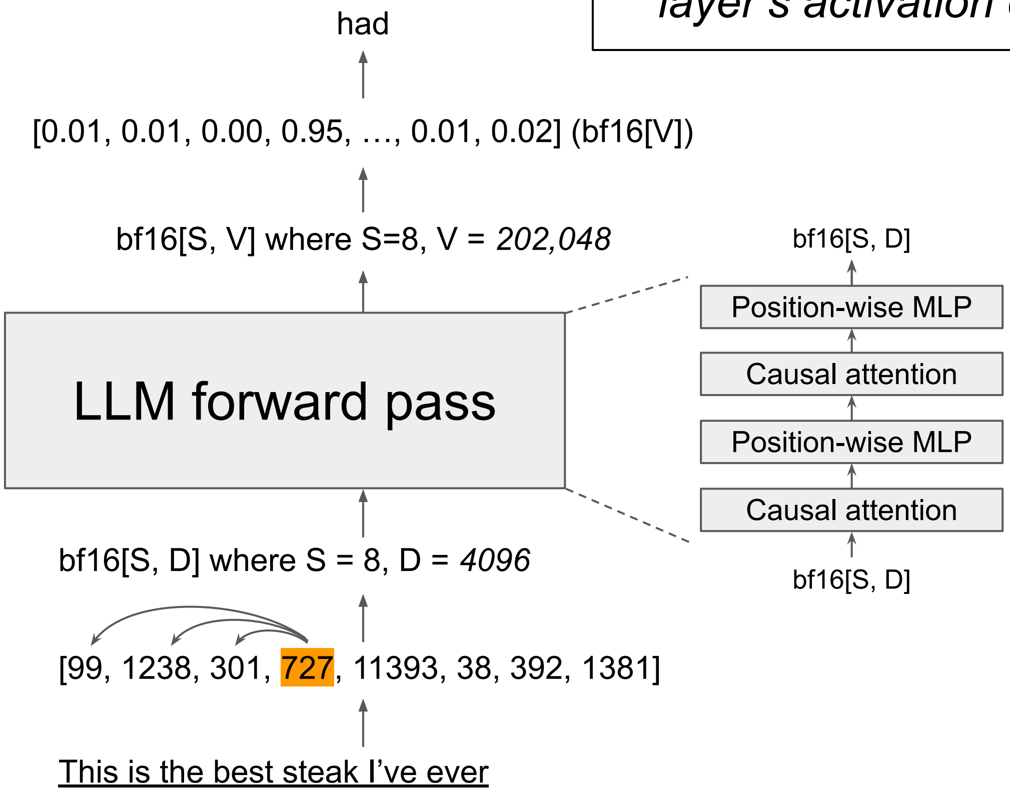
LLM Inference

Would the first 8 positions of each layer's activation change at all?



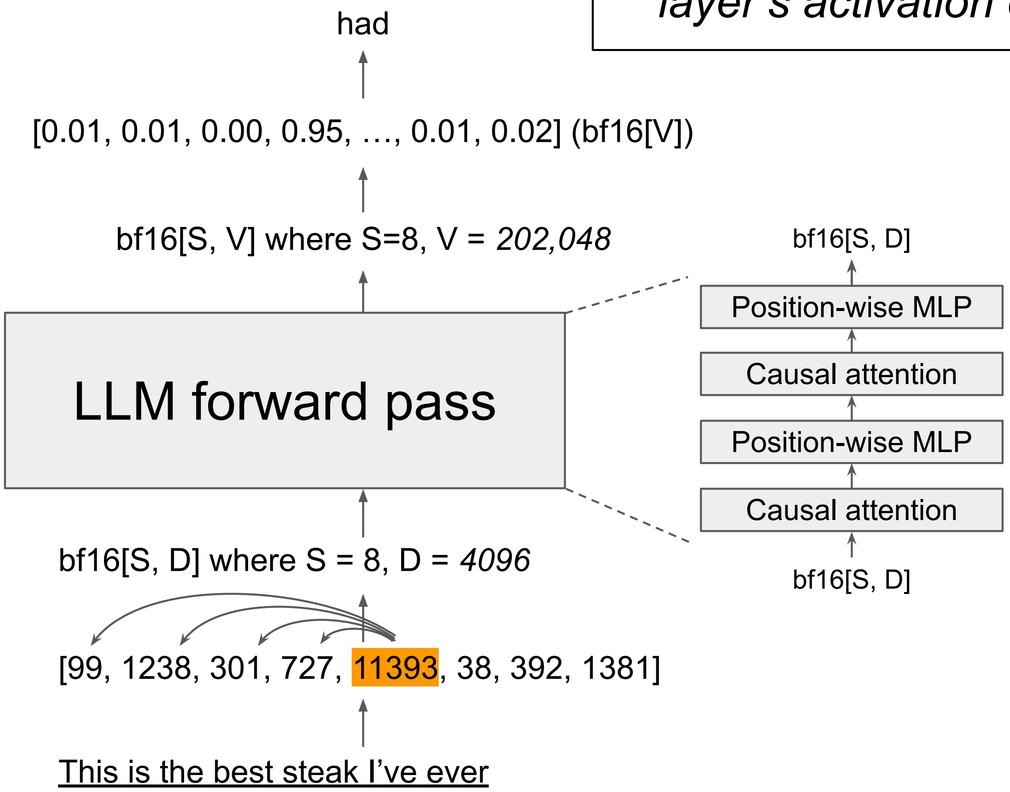
LLM Inference

Would the first 8 positions of each layer's activation change at all?



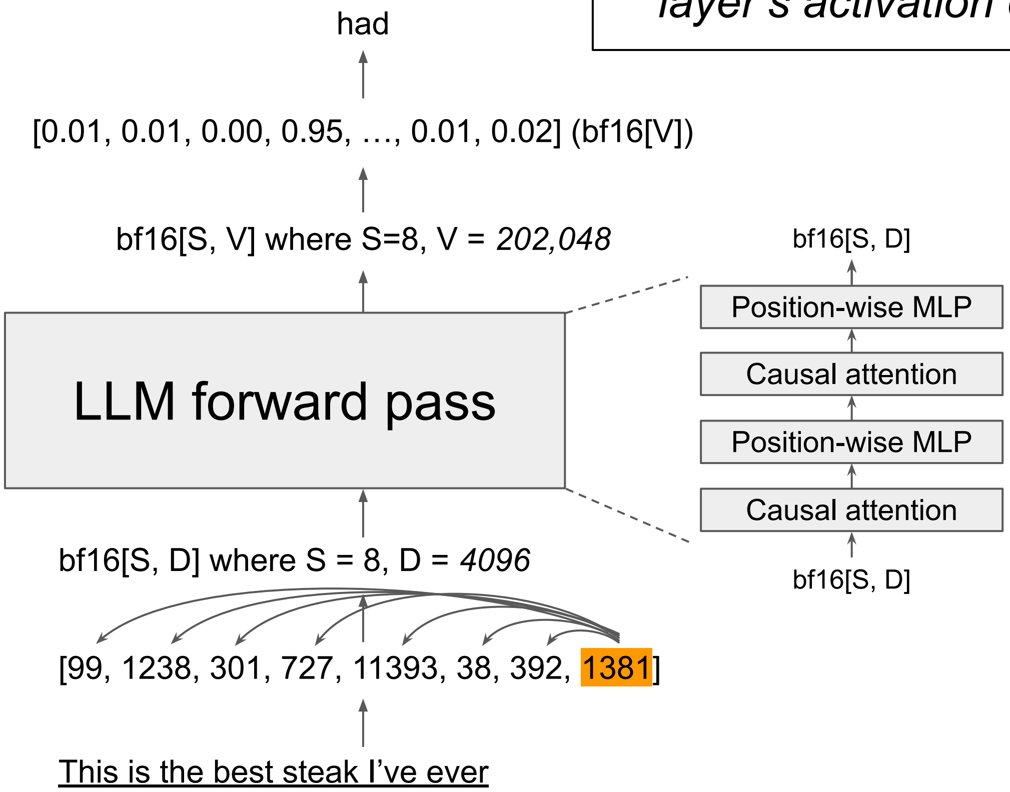
LLM Inference

Would the first 8 positions of each layer's activation change at all?



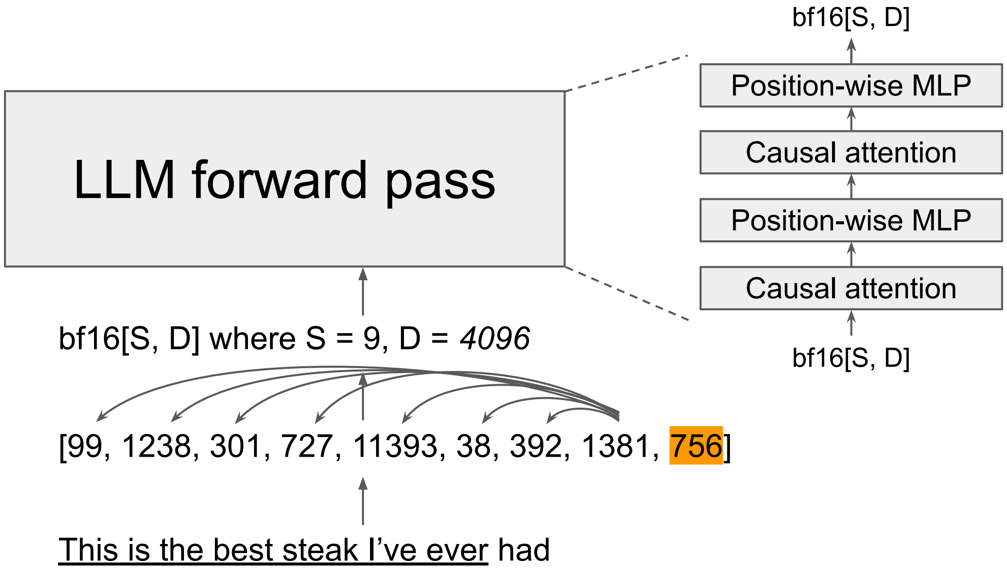
LLM Inference

Would the first 8 positions of each layer's activation change at all?



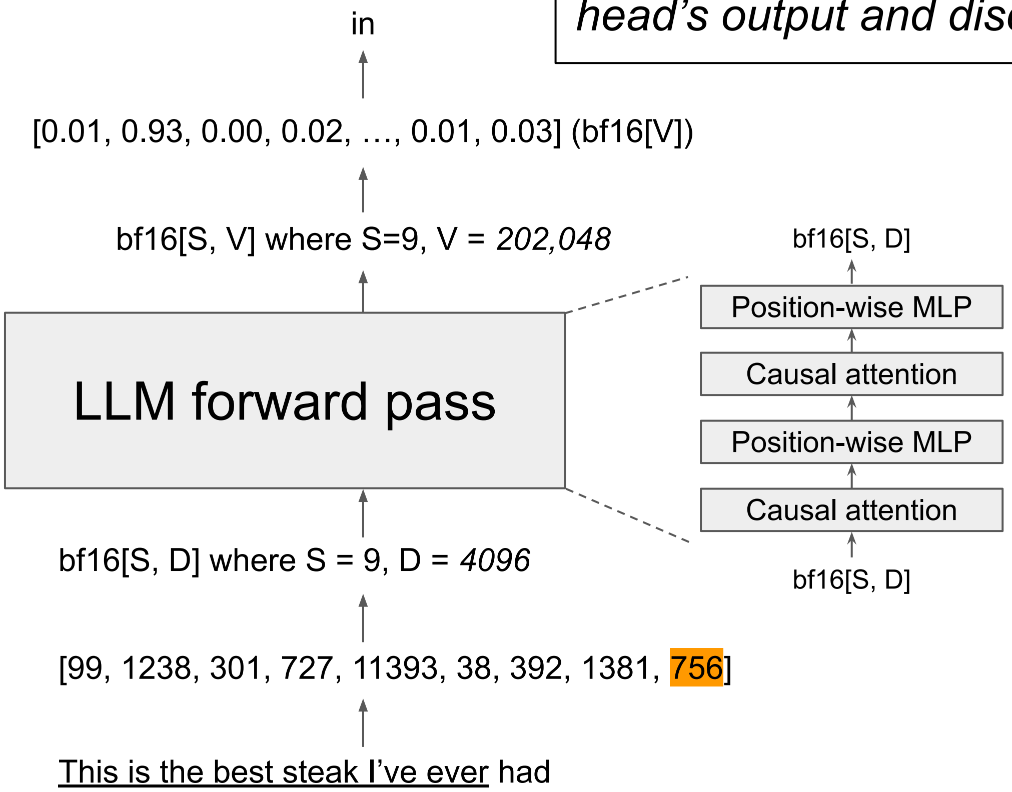
LLM Inference

*The first 8 positions of intermediate activations are **the same**.*



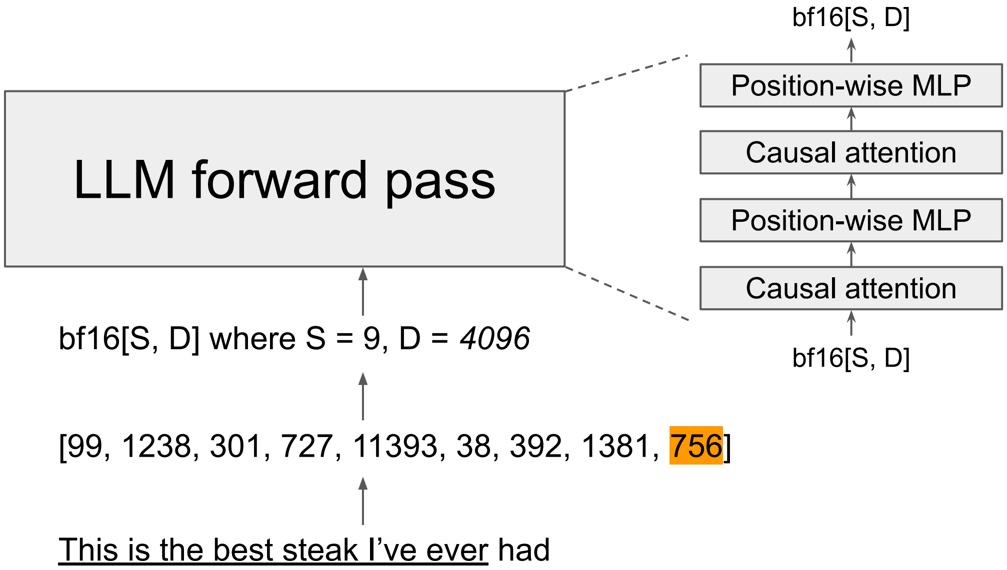
LLM Inference

We're taking the 9th position of the head's output and discarding the rest.



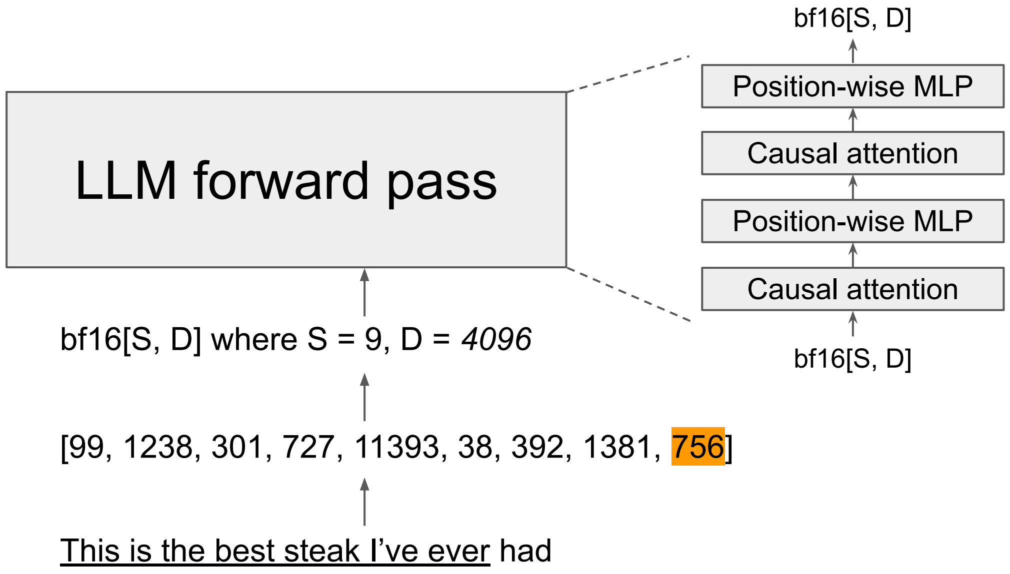
LLM Inference

*Re-computing earlier positions seems super **redundant**.*

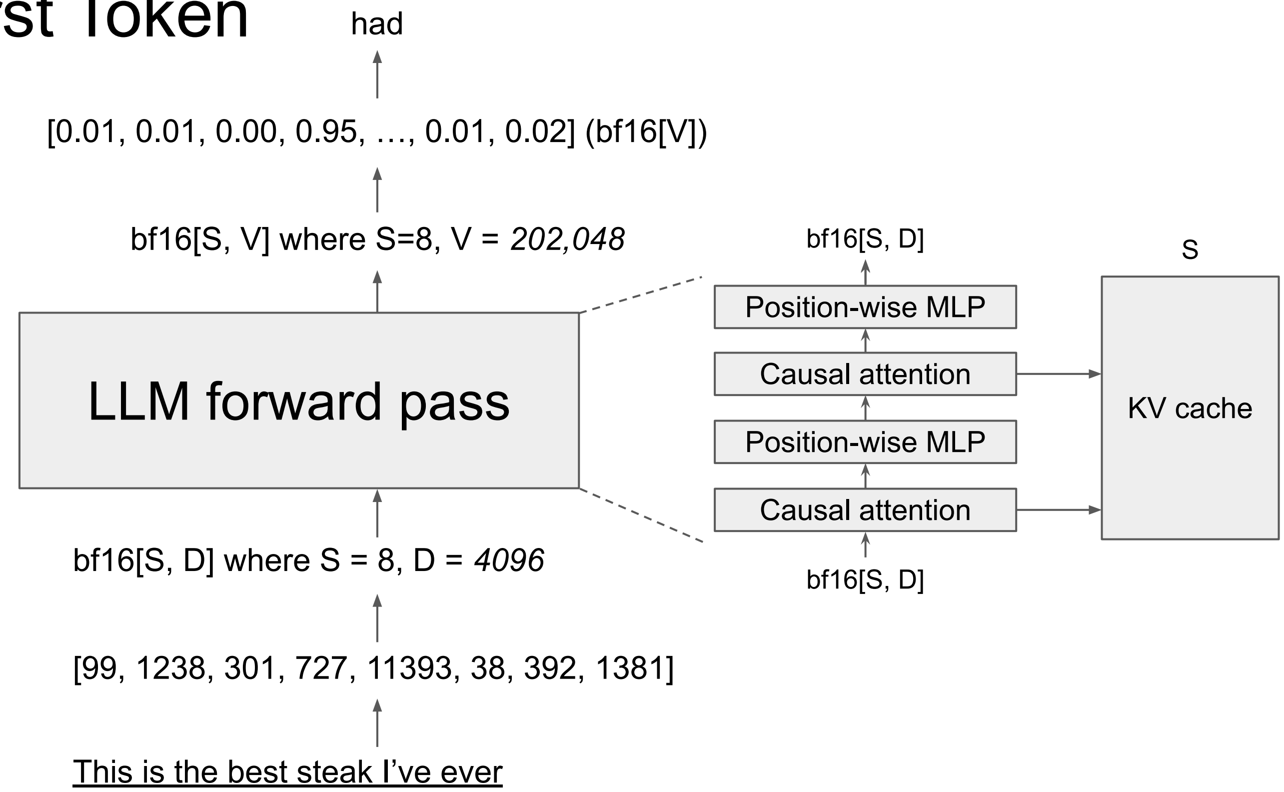


Reducing Computation for the Second & Later Tokens

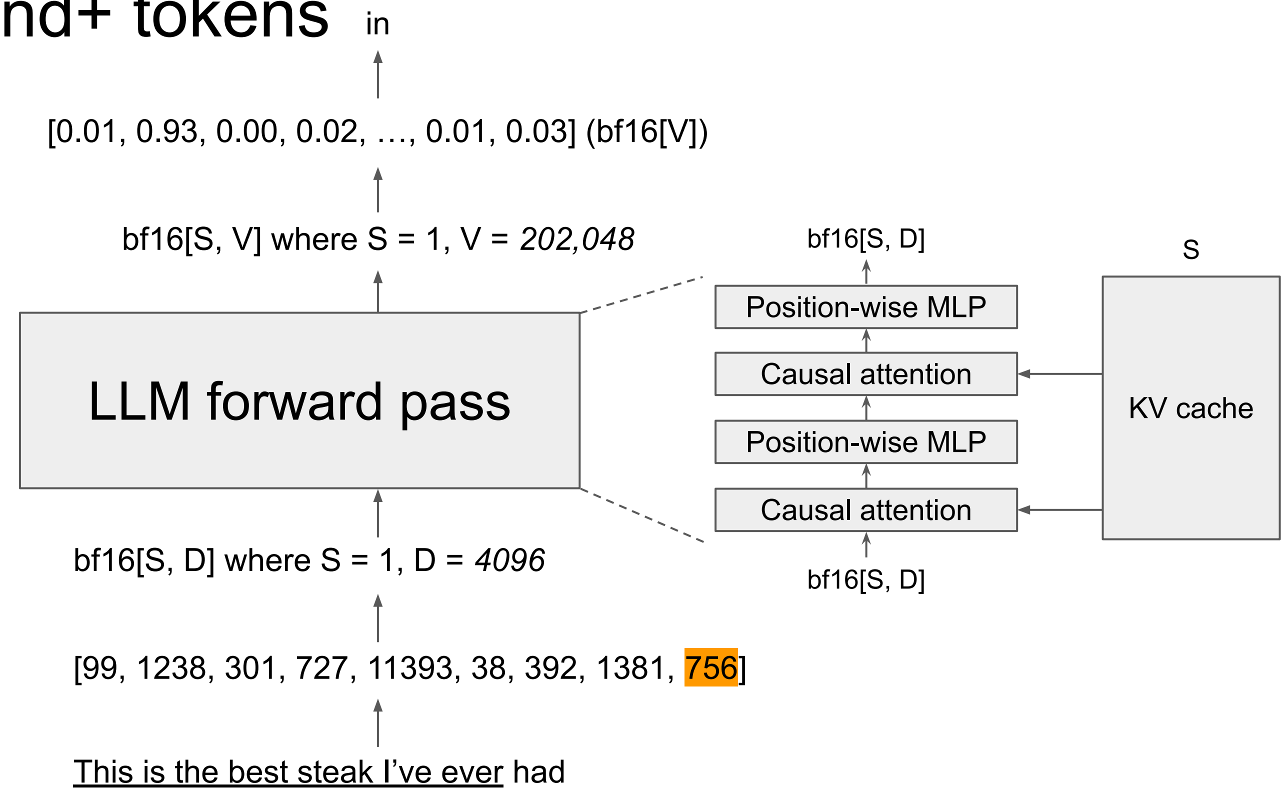
- Position-wise MLP
  - As long as you get the bf16[D] activations for the 9th position, the MLP can be computed
- Causal attention
  - It *does* depend on earlier tokens
  - You just needs two things for each token position:
    - Key tensor
    - Value tensor



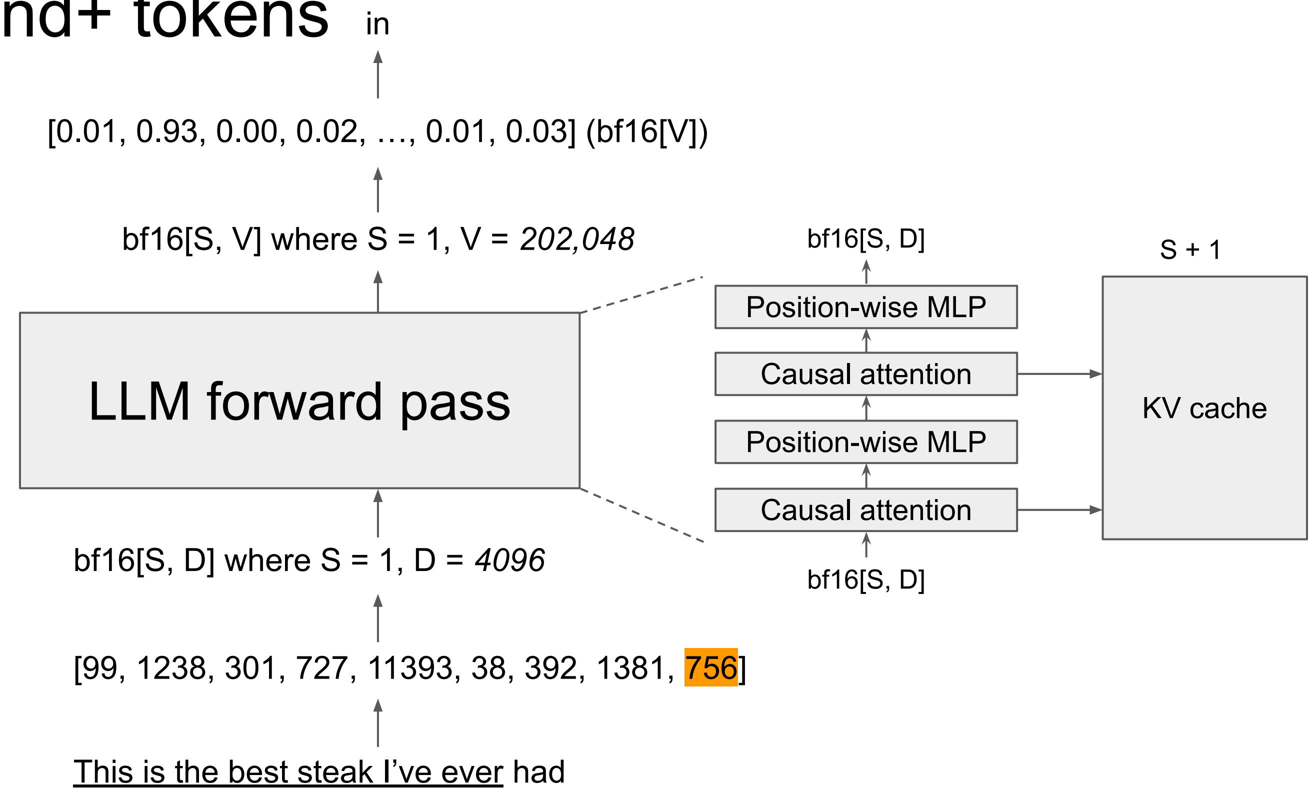
Prefill: The First Token



Decode: Second+ tokens



Decode: Second+ tokens



KV Cache

- Characteristics
  - Constant size for **each token in the context** (input or output)
  - So, for each output token generated, KV cache size grows **linearly**
- Size calculation for a single token
  - (# Transformer layers) x (# KV heads) x 2 (K and V) x (head output dimension) x 2 bytes

Model	1 token	4096 tokens
Llama 3.1 8B	32 x 8 x 2 x 128 x 2 bytes = 128 KiB	0.50 GiB
Llama 3.1 70B	80 x 8 x 2 x 128 x 2 bytes = 320 KiB	1.25 GiB
Llama 3.1 405B	126 x 8 x 2 x 128 x 2 bytes = 504 KiB	1.97 GiB

Optimization Metrics

- Prefill latency
  - **Time To First Token (TTFT)**
  - TTFT + queuing delay is the user-experienced wait time
- Decode latency
  - **Time Per Output Token (TPOT)**: Average decode latency for all output tokens
  - **Inter-Token Latency (ITL)**: Latencies between each output token
  - Long decode latency leads to a laggy chat experience and will annoy users
  - It also doesn’t have to be infinitely fast if the LLM is user-facing
- Throughput
  - Request throughput (reqs/s)
  - Token generation throughput (tokens/s)

A Potential Optimization: Batching

- Training had batch size; what about for inference?
  - Running more than one requests together
- LLMs are pretty weird
  - Very large in size
  - Prefill and decode
- Questions
  - Is batching a no-brainer optimization, or does it only sometimes make sense?
  - How should we *reason* about batching for LLMs on modern GPUs?

Matrix Multiplication

- It's everywhere in Transformers and takes up most of the compute
  - Some in attention, and the MLP is basically two *big* matmuls
- Focus: Position-wise MLP
  - Batch size (B) is the total number of tokens being forwarded through the MLP
  - Say we batched together three requests with input sequence length: 3, 4, and 5
  - Prefill: B = 12 – total number of input tokens
  - Decode: B = 3 – number of tokens generated (== number of requests)

Input	bf16[B, D]
Weight	bf16[D, F]
Output	bf16[B, F]

Analyzing Matrix Multiplication

Input	bf16[B, D]
Weight	bf16[D, F]
Output	bf16[B, F]

- Computation and memory accesses
    - B = 128, D = 4096, F = 16384

	Computation	Memory
Matmul	2BDF = 16 x 10^9 FLOPs	2BD + 2DF + 2BF = 0.13 x 10^9 Bytes
H100 spec	1979 TFLOPs/s / 2 = 989.5 x 10^{12} FLOPs/s	3.35 TB/s = 3.35 x 10^{12} Bytes/s

Which one is bottlenecking our matmul?
***Computation throughput**, or **memory throughput**?*

Arithmetic Intensity

- A single number that tells us which one we're bottlenecked by
- **Arithmetic Intensity**: Computation per byte of memory access
    - Amount of computation / amount of memory access (FLOPs/bytes)

	Computation	Memory	Computation per byte
Matmul	2BDF = 16×10^9 FLOPs	2BD + 2DF + 2BF = 0.13×10^9 Bytes	Exactly: 123 **Approximate with B**: 128
H100 spec	1979 TFLOPs/s / 2 = 989.5×10^{12} FLOPs/s	3.35 TB/s = 3.35×10^{12} Bytes/s	Exactly: 295

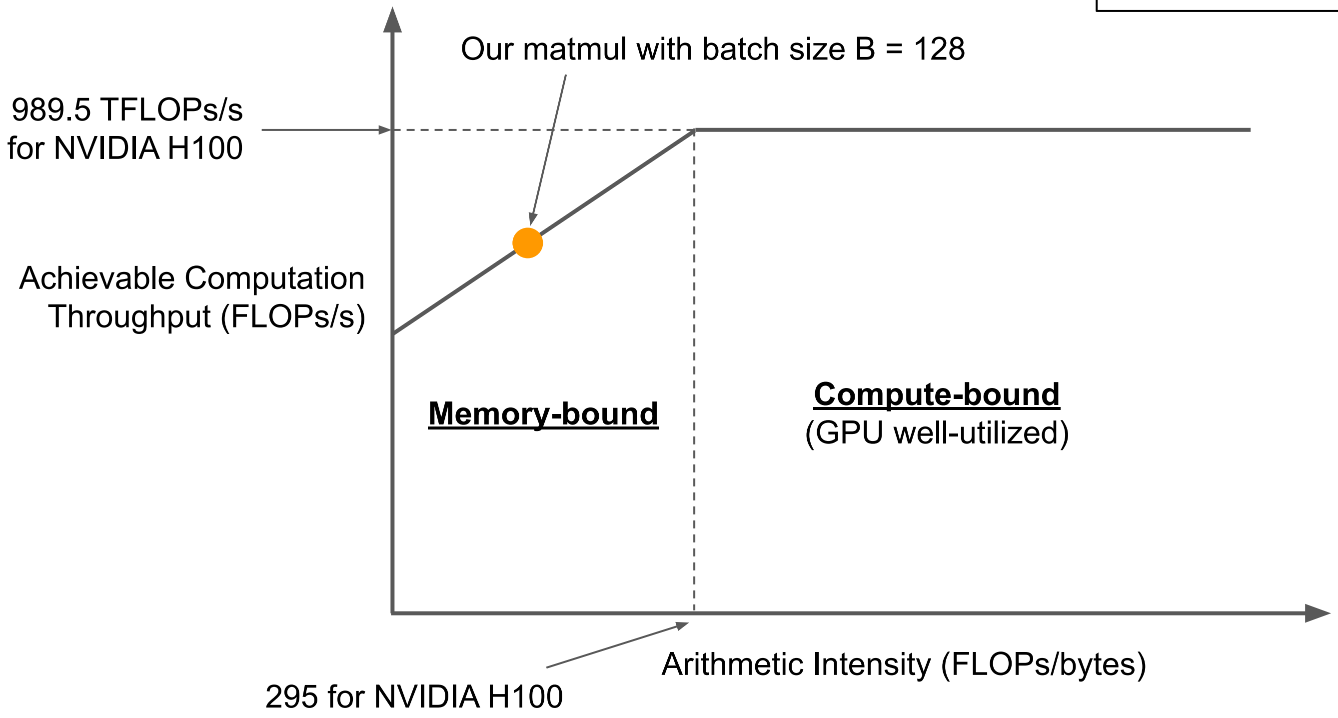
B = 128, D = 4096, F = 16384

*It's **memory bound**.*

The Roofline Plot

You get a different roofline for each

- Hardware
- Floating point precision
- Dense vs. sparse operation
- etc.

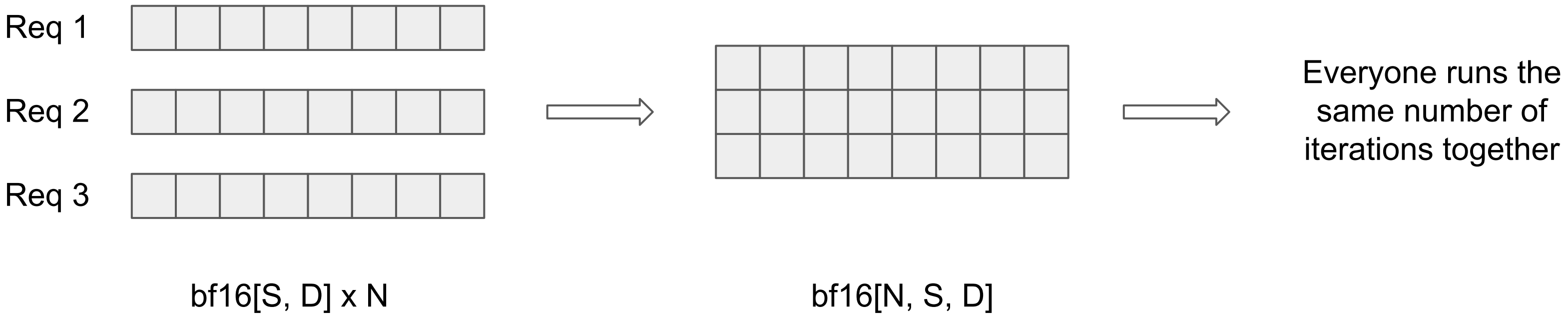


What All This Means for Prefill and Decode

- Position-wise MLP
  - Batch size (B) is the total number of tokens being forwarded through the MLP
  - Say we batched together three requests with input sequence length: 3, 4, and 5
  - Prefill: B = 12 – total number of input tokens
  - Decode: B = 3 – number of tokens generated (== number of requests)
- Prefill and Decode
  - **<u>Prefill</u>**: 295 tokens across input sequences is easy, so you're **<u>nearly always compute-bound</u>**
    - You actually don't have to run multiple *requests* together for prefill to be compute-bound
  - **<u>Decode</u>**: Running 295 *requests* together is *not easy*, so you're **<u>usually memory-bound</u>**
    - We always want to **<u>increase</u>** the number of *requests* running together for decode

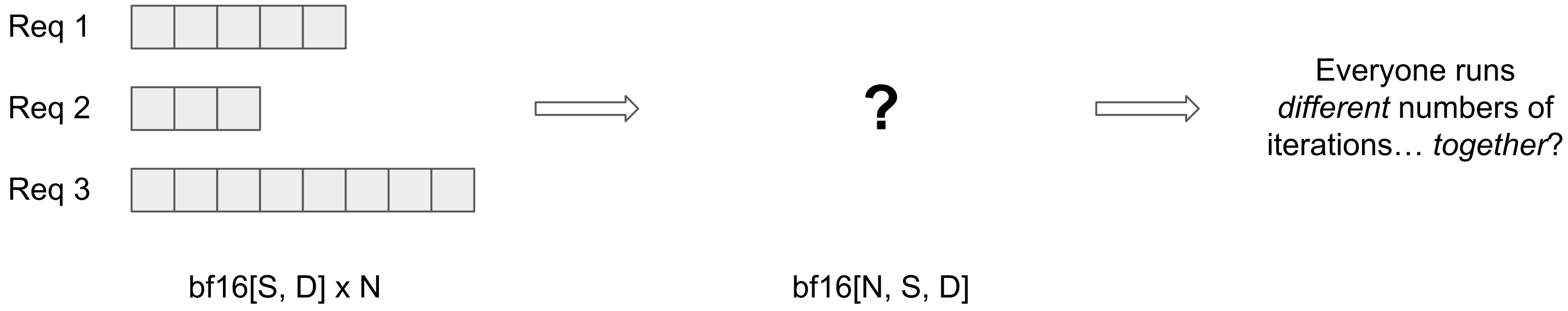
Orca (OSDI '22): Efficient Batching for LLMs

- If LLM inference requests
  - had the same number of input tokens, and
  - had the same number of output tokens (i.e., same number of decode iterations),
  - then batching is trivial



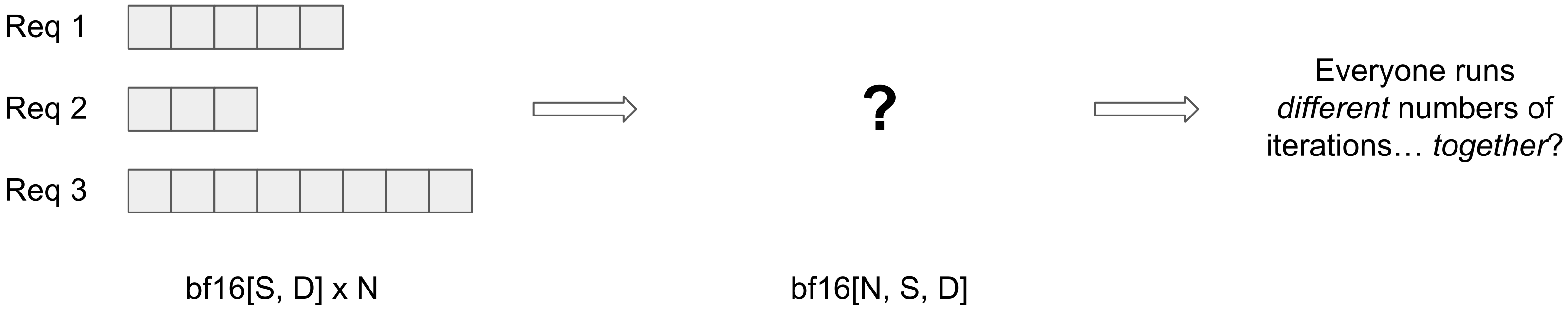
Orca (OSDI ’22): Efficient Batching for LLMs

- If LLM inference requests
  - had the same number of input tokens, and
  - had the same number of output tokens (i.e., same number of decode iterations),
  - then batching is trivial



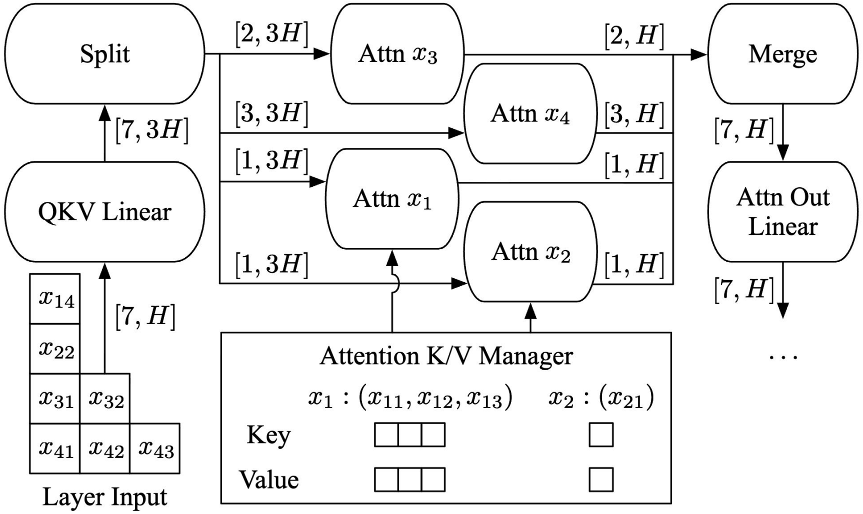
Orca (OSDI ’22): Efficient Batching for LLMs

- Position-wise MLP
  - This is position-wise anyway
  - Just flatten all the tokens together into the batch dimension, and run forward



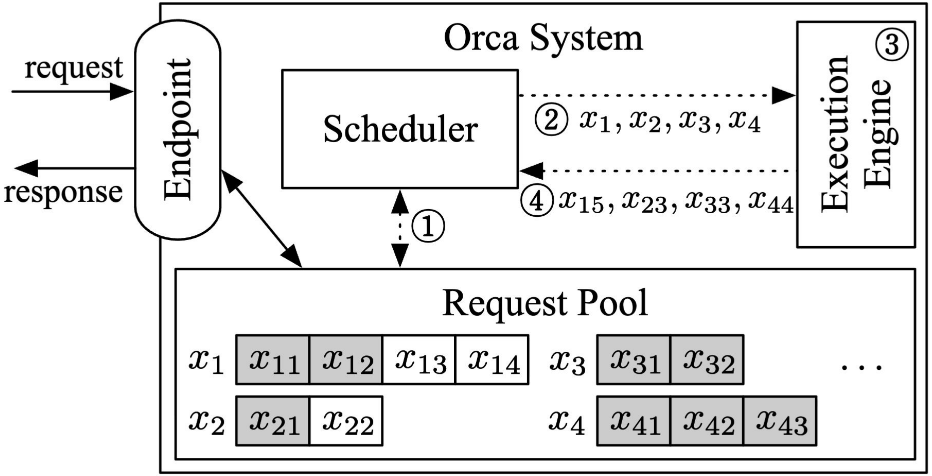
Orca (OSDI '22): Efficient Batching for LLMs

- Attention
  - GPUs are parallel enough
  - Just dispatch the attention operation of each sequence in parallel



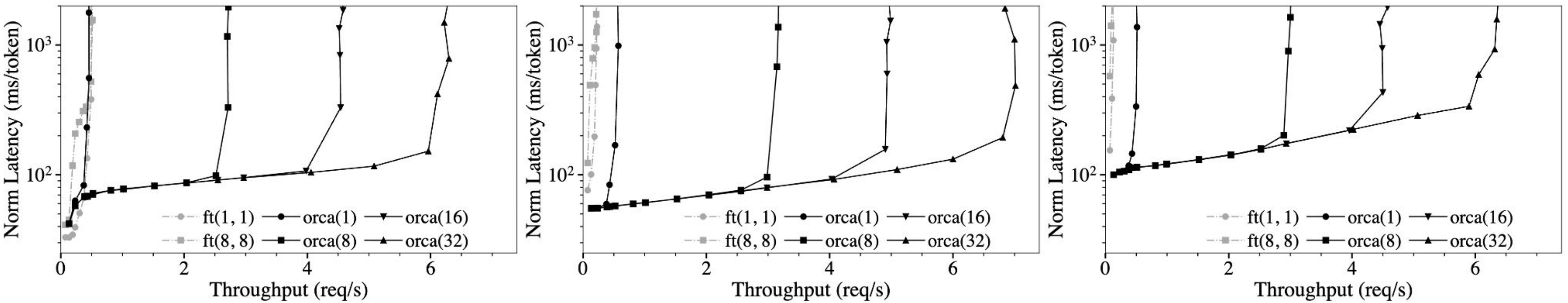
Orca (OSDI '22): Efficient Batching for LLMs

- Scheduling benefits
  - Requests may enter and leave the system at iteration boundaries
  - Run one iteration → Back to scheduler, finished ones leave the system
  - New request → Enqueued into scheduler, can start processing when there's an empty slot



Orca (OSDI ’22): Efficient Batching for LLMs

- Crushes baselines
  - NVIDIA A100-40GB GPUs
  - Batch size is in the order of *tens*

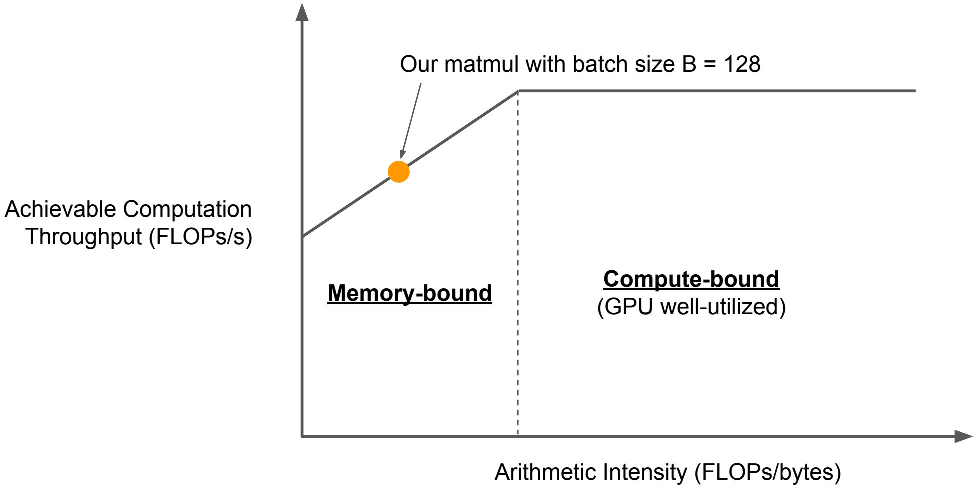


(a) 101B model, 8 GPU. (b) 175B model, 16 GPUs. (c) 341B model, 32 GPUs.

Figure 10: Median end-to-end latency normalized by the number of generated tokens and throughput. Label “orca(*max_bs*)” represents results from ORCA with a max batch size of *max_bs*. Label “ft(*max_bs*, *mbs*)” represents results from FasterTransformer with a max batch size of *max_bs* and a microbatch size of *mbs*.

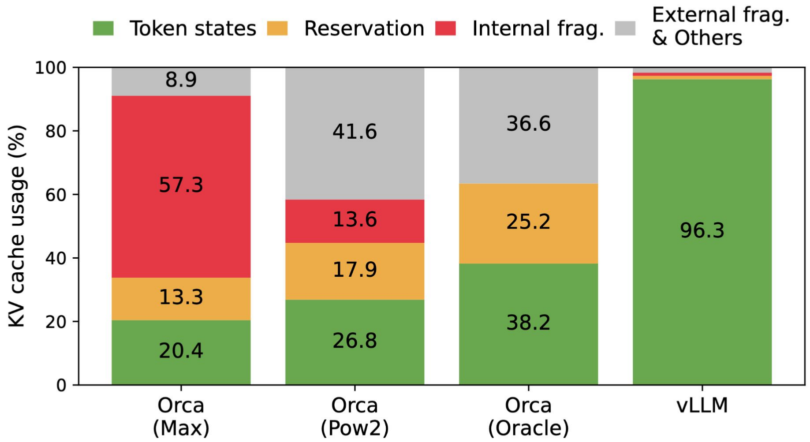
vLLM (SOSP '23): Memory is Key!

- Recall arithmetic intensity of matrix multiplication
  - We wanted a **larger batch size** for better hardware utilization & throughput
- Why can't we increase batch size very easily for Decode?
  - Each request needs to have its **KV cache** in GPU memory – hits **VRAM capacity**



vLLM (SOSP '23): Memory is Key!

- You don't know how much KV cache memory to allocate for a request
  - Because you don't know **how many output tokens** it'll generate
- KV cache memory pre-allocation leads to memory wastage
  - Max, Pow2 (Half-oracle), and Oracle: Policies for how much memory is pre-allocated
  - In all cases, memory is pre-allocated during the **entire lifetime** of the request

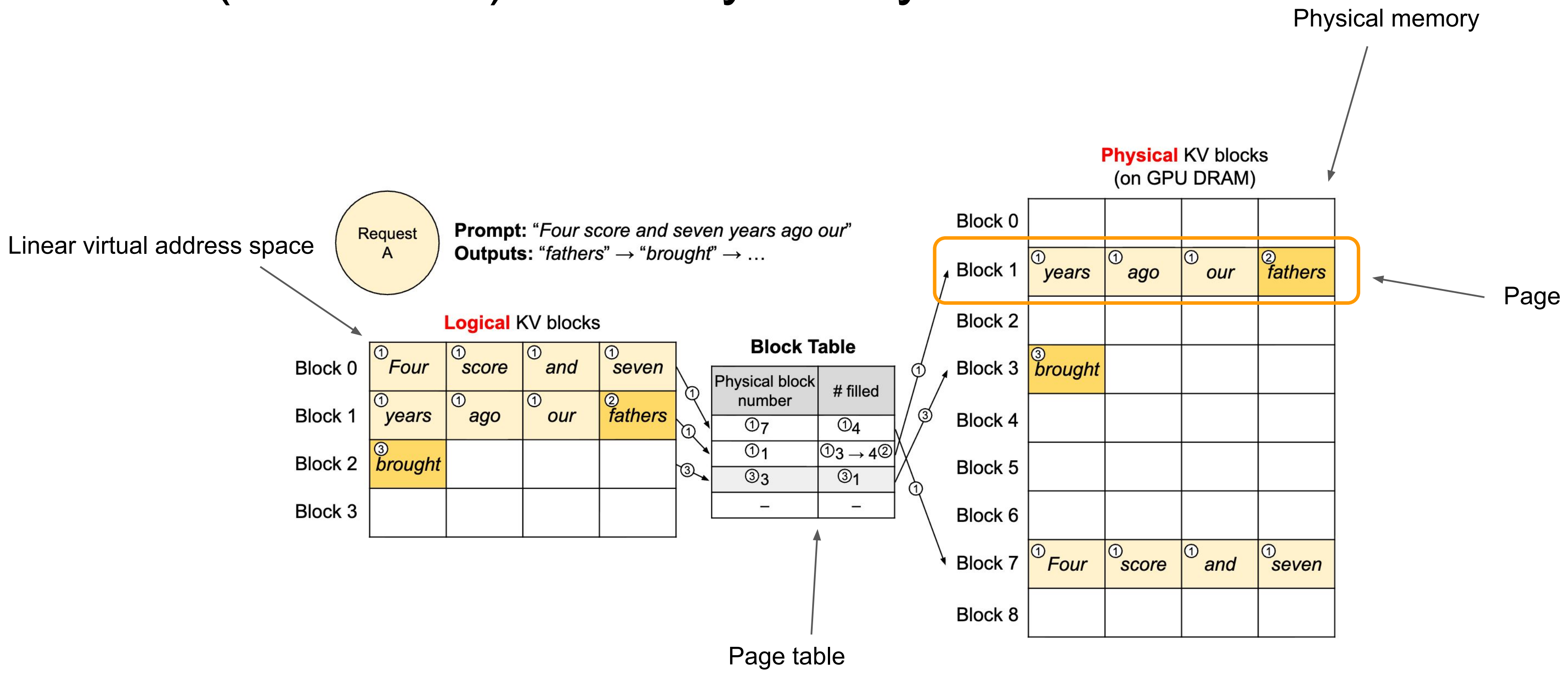


vLLM (SOSP ’23): Memory is Key!

- This sounds familiar
  - There are units of work, and they come and go
  - You don’t know how much memory these units will consume (they can vary quite a bit)
  - When the unit goes away, you deallocate all memory associated with it
  - Like… *processes*

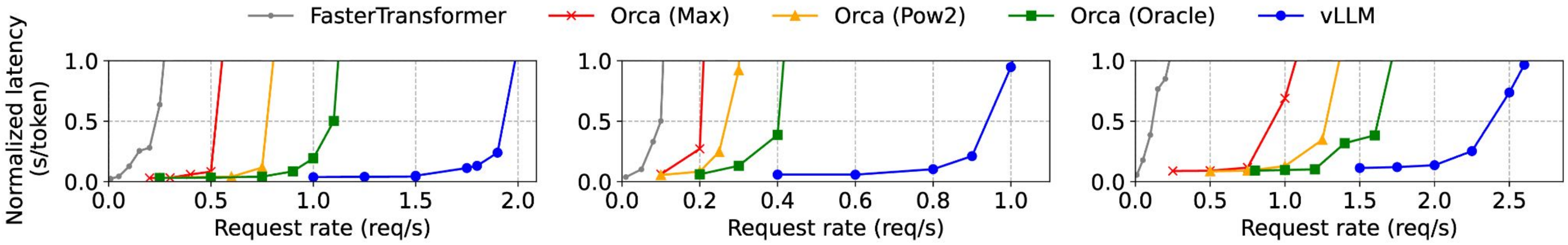
Virtual memory and pages

vLLM (SOSP '23): Memory is Key!



vLLM (SOSP '23): Memory is Key!

- Crushes baselines
  - NVIDIA A100-40GB and A100-80GB GPUs
  - For each system, uses the largest batch size that fits in memory



(a) OPT-13B, 1 GPU, ShareGPT (b) OPT-66B, 4 GPUs, ShareGPT (c) OPT-175B, 8 GPUs, ShareGPT

In the Coming Weeks

- You will learn about
  - Different workloads
  - Parallelism
  - Disaggregation
  - Resource management
  - Optimization metric design
  - Various sources of heterogeneity
  - … and more!