LLM Serving

Nitya Agarwal & Benjamin Xia

Table of contents

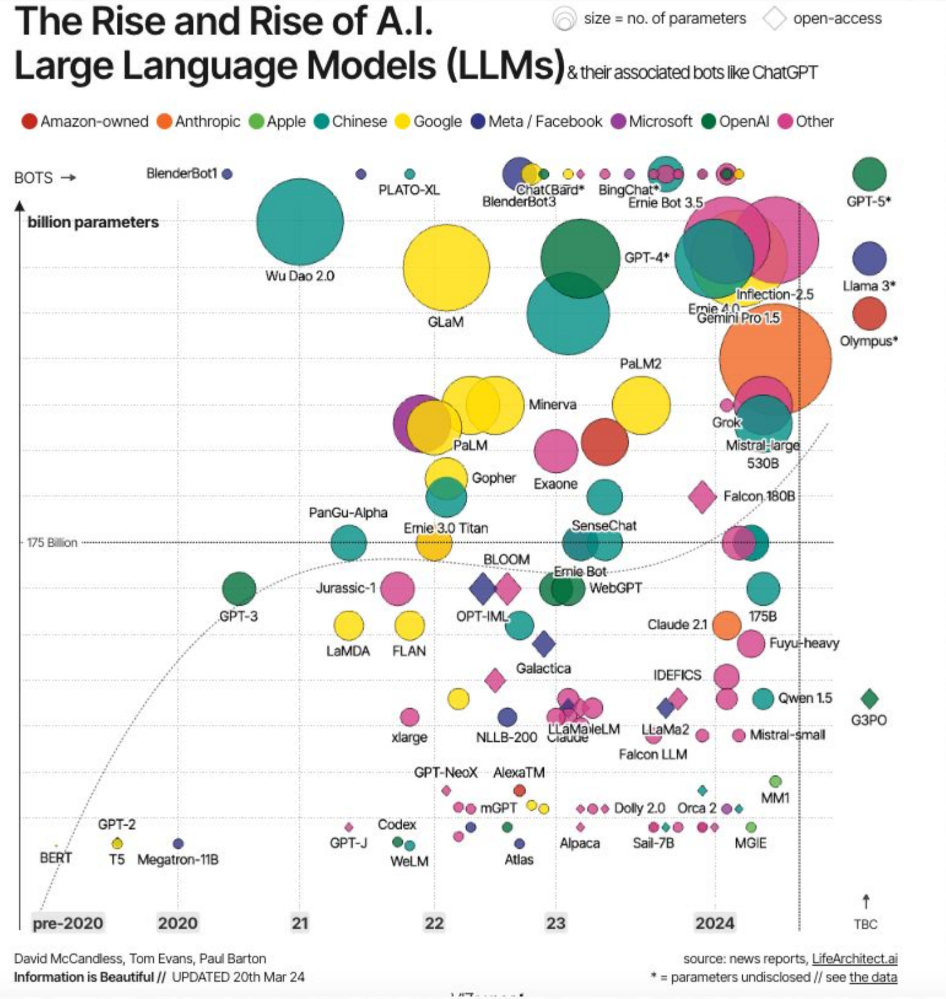
01

Background

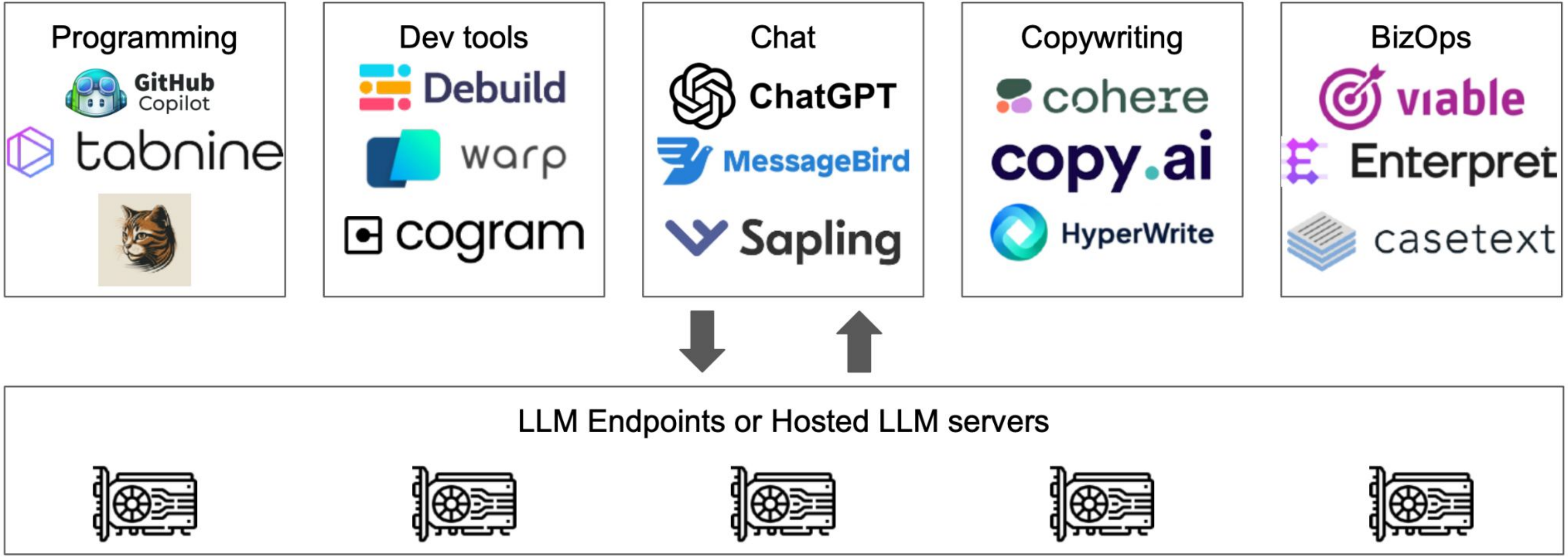
LLMS + Self Attention + Model Size

The era of LLMS

- Increasing **model size**
- Increasing popularity



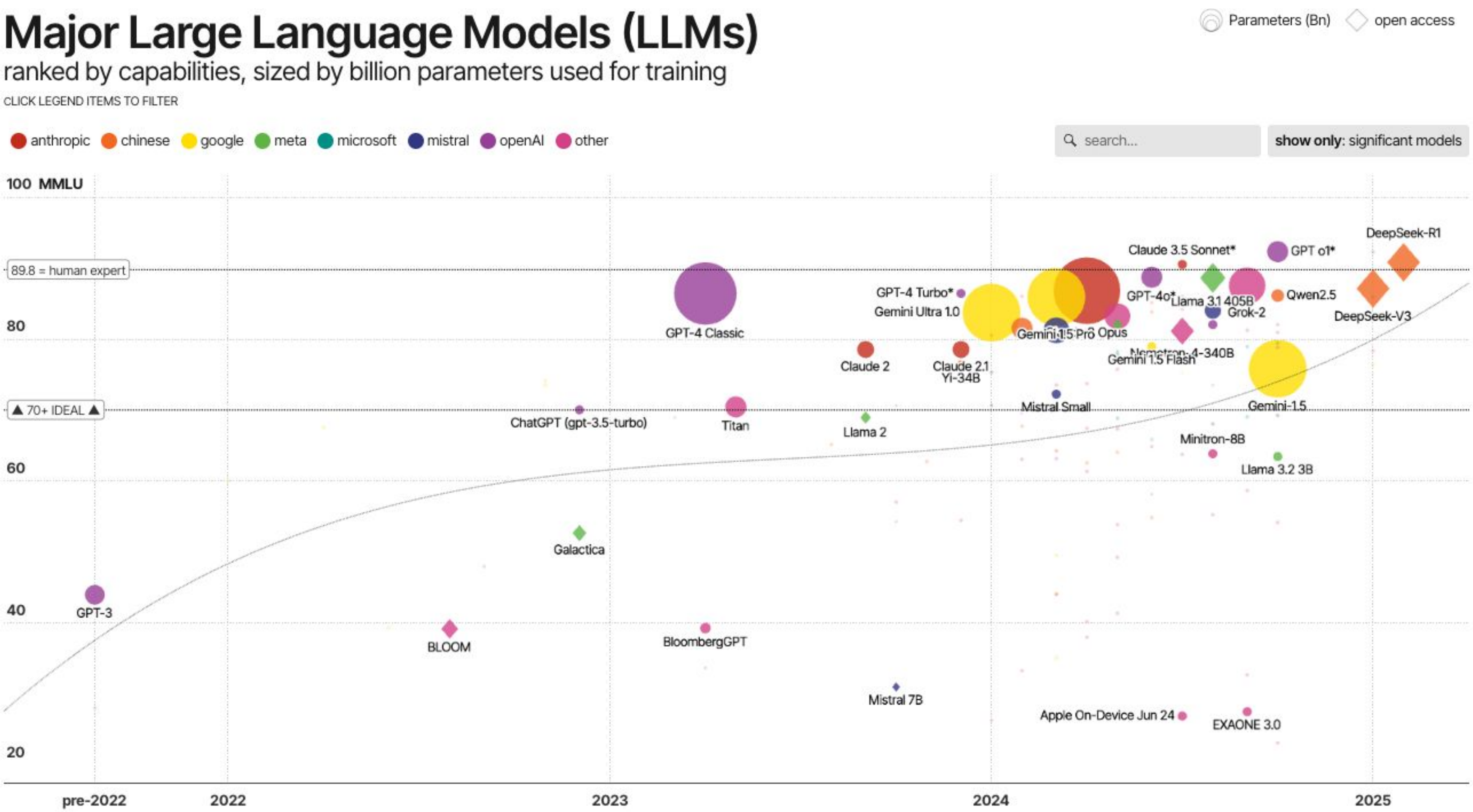
LLM-powered services



Major Large Language Models (LLMs)

ranked by capabilities, sized by billion parameters used for training

CLICK LEGEND ITEMS TO FILTER



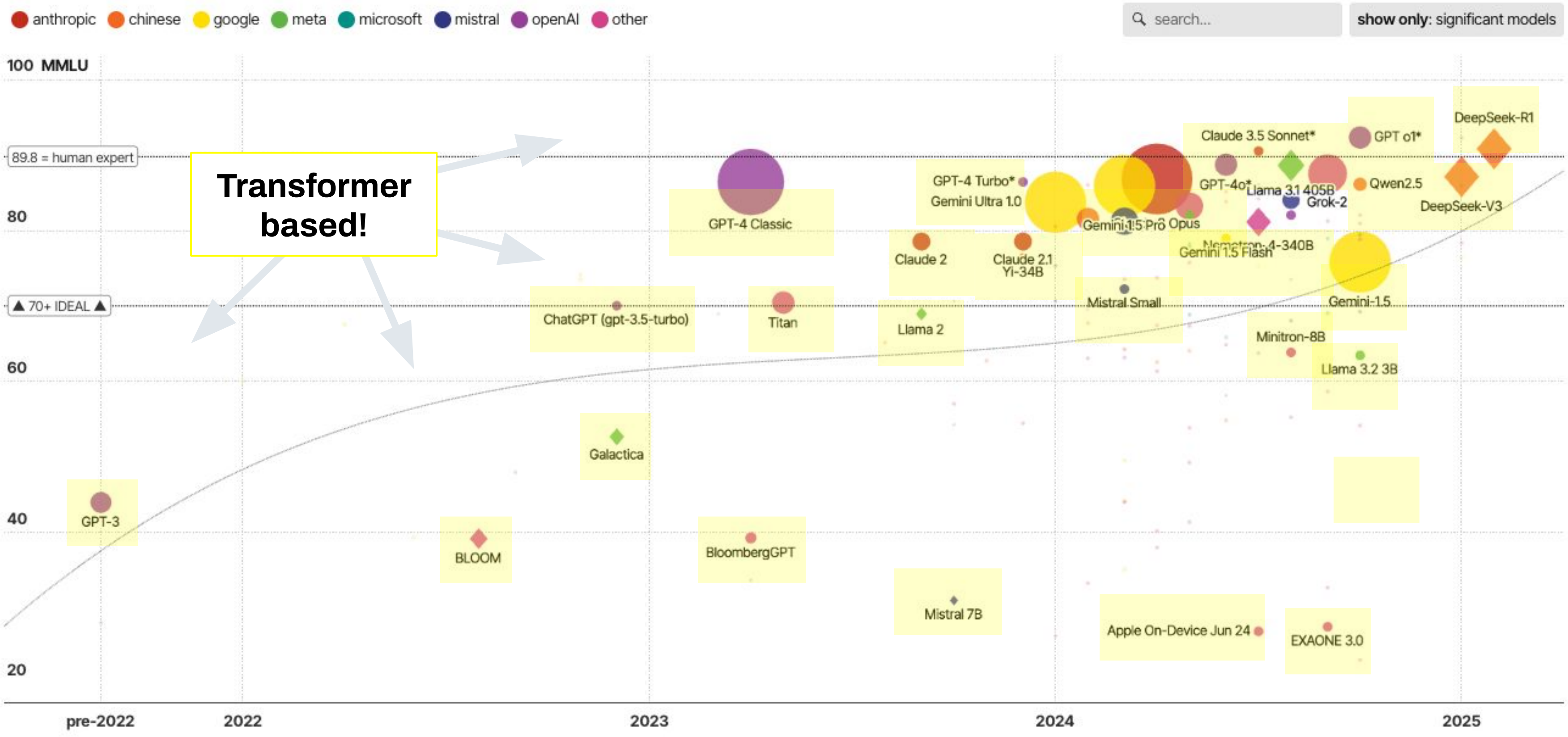
David McCandless, Tom Evans, Paul Barton
Informationisbeautiful // Jan 2024

MMLU = benchmark for measuring LLM capabilities
* = parameters undisclosed // source: LifeArchitect // data

Major Large Language Models (LLMs)

ranked by capabilities, sized by billion parameters used for training

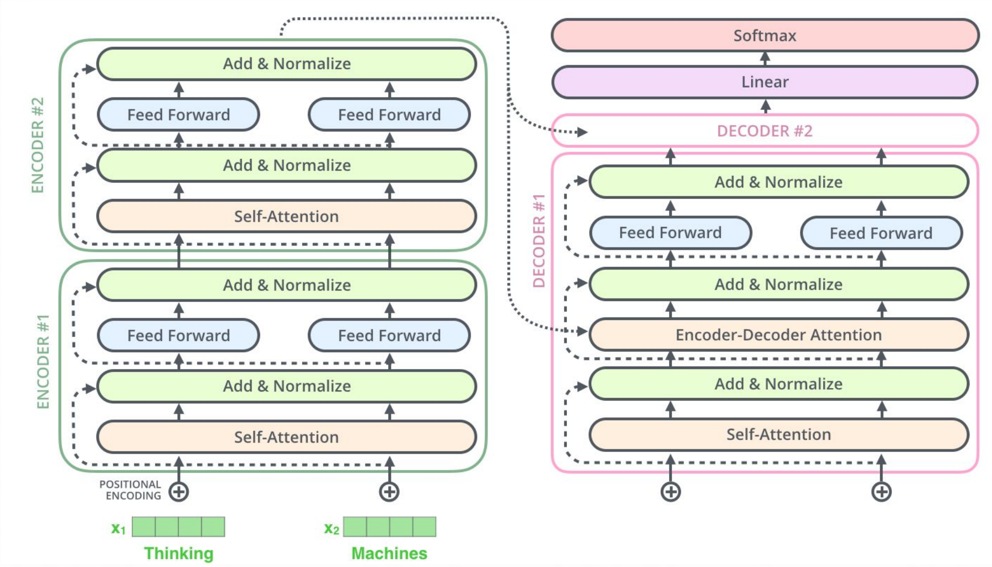
CLICK LEGEND ITEMS TO FILTER



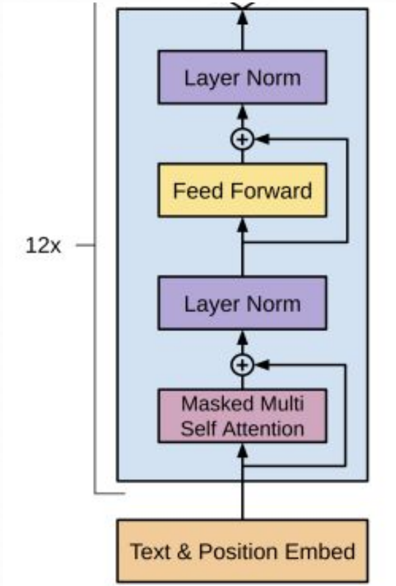
David McCandless, Tom Evans, Paul Barton
Informationisbeautiful // Jan 2024

MMLU = benchmark for measuring LLM capabilities
* = parameters undisclosed // source: LifeArchitect // data

Transformer Architecture



Encoder-Decoder Transformer

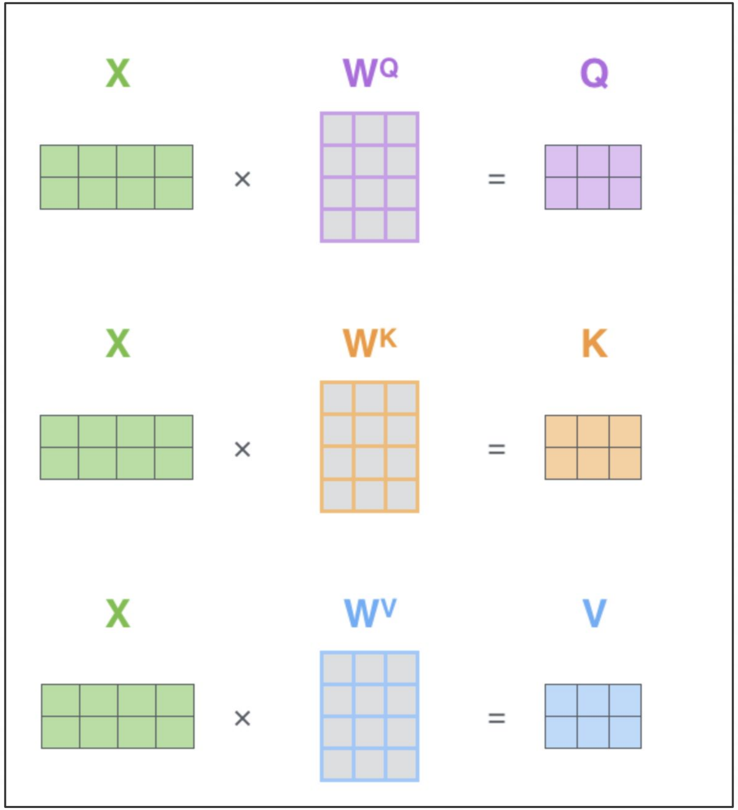


Original GPT-1 Architecture
(Decoder-only)

Source: https://jalammar.github.io/illustrated-transformer/

Source: Language Models are Few-Shot Learners https://arxiv.org/abs/2005.14165

Attention Basics - Self Attention

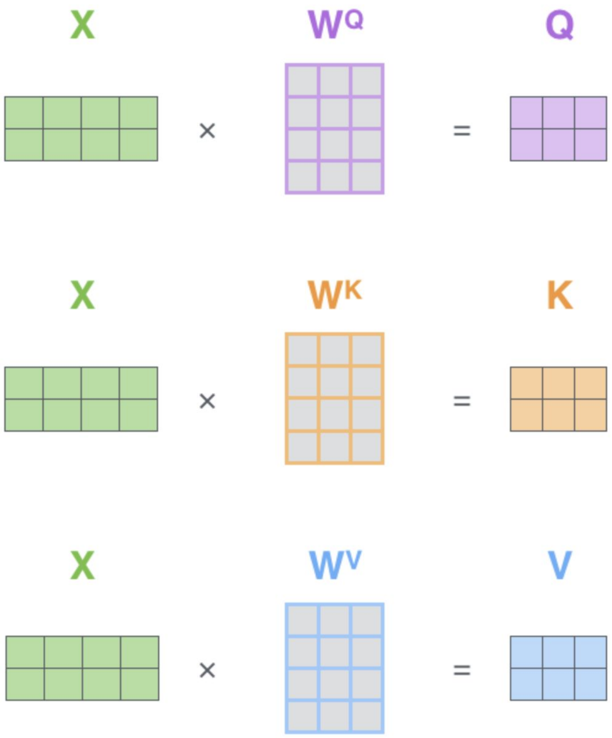


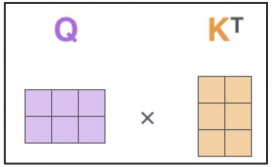
Every row in the X matrix corresponds to a word in the input sentence.

Learn the W matrices to obtain Query Q, Key K and Value V matrices.

Source: https://jalammar.github.io/illustrated-transformer/

Attention Basics - Self Attention



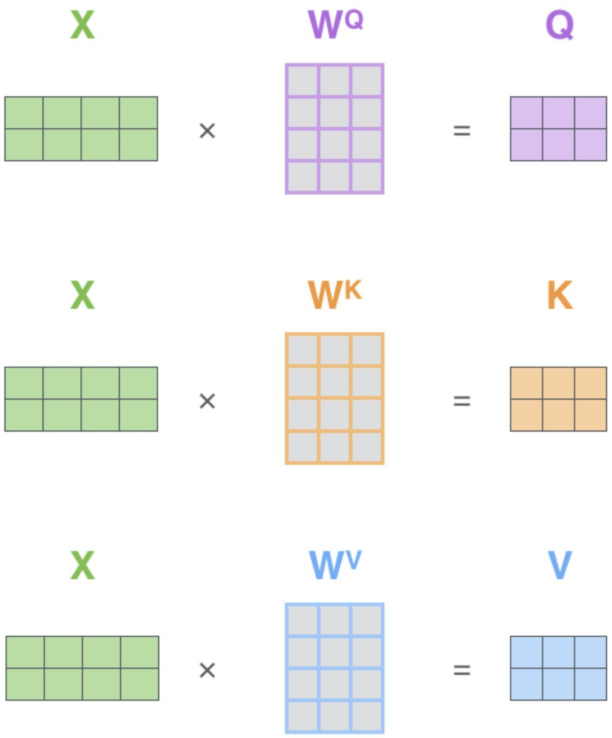


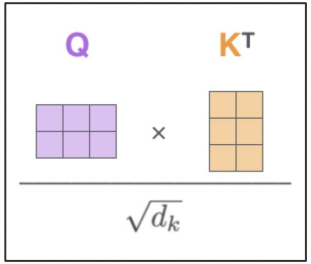
Calculate a score for how much focus to place on other parts of the input sentence as we encode a word at a certain position.

Take dot product of the **query vector** with the **key vector** of the respective word we're scoring.

Source: https://jalammar.github.io/illustrated-transformer/

Attention Basics - Self Attention



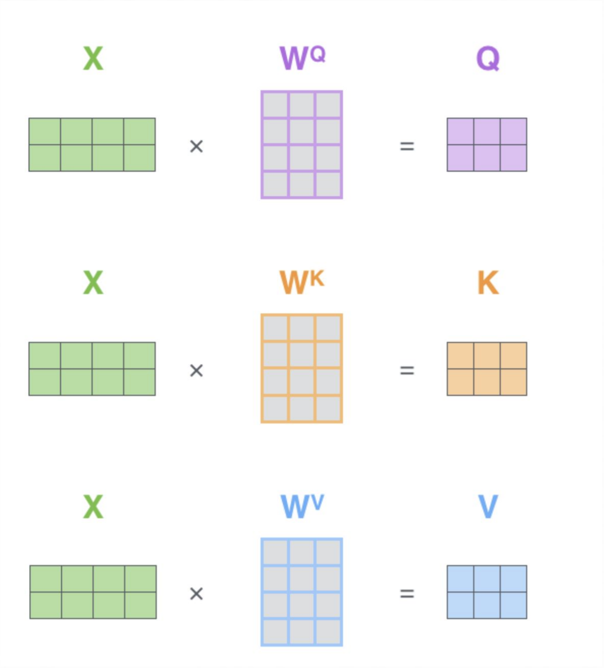


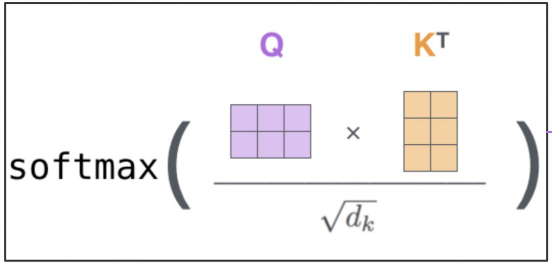
Divide scores by the square root of the dimension of the **key vectors** (more stable gradients).

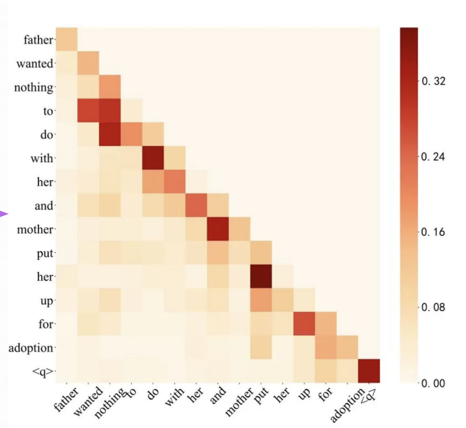
Taking the dot product of two random vectors of length d with mean 0 and variance 1 will have variance d.
Dividing by sqrt(d) normalizes the dot product to have variance 1, which is a desirable property in neural networks.

Source: https://jalammar.github.io/illustrated-transformer/

Attention Basics - Self Attention





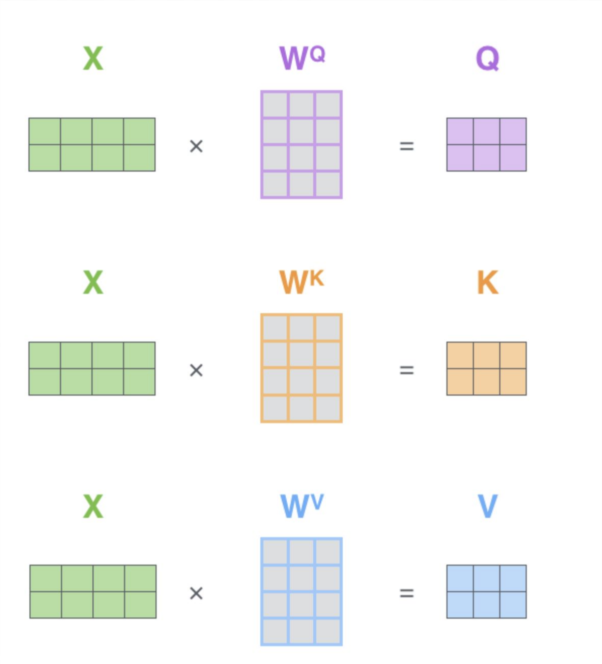


Pass result through a softmax operation (all positive and add up to 1).

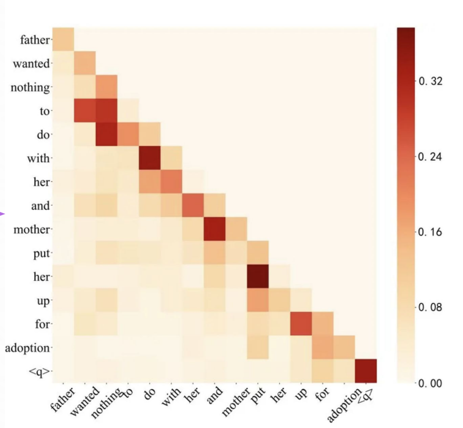
Why is the softmax operation applied after computing attention scores?

Source: https://jalammar.github.io/illustrated-transformer/

Attention Basics - Self Attention



$$\text{softmax}\left(\frac{Q \times K^{T}}{\sqrt{d_k}}\right)$$

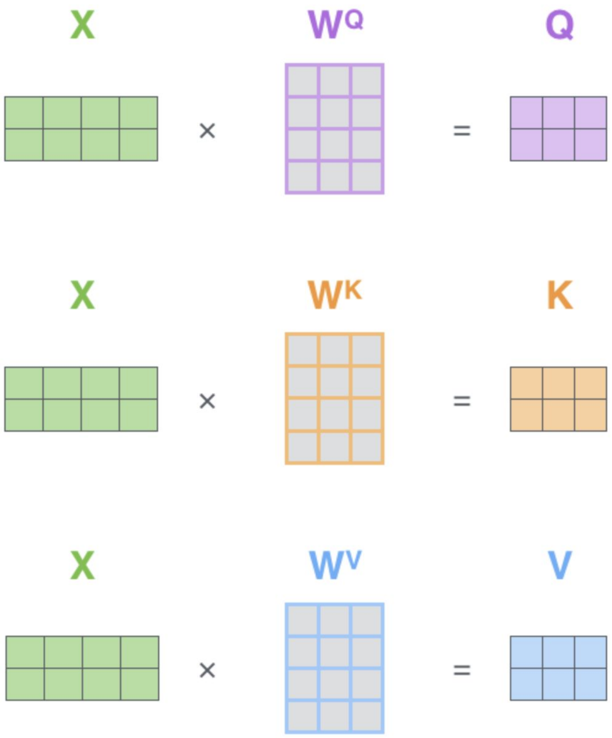


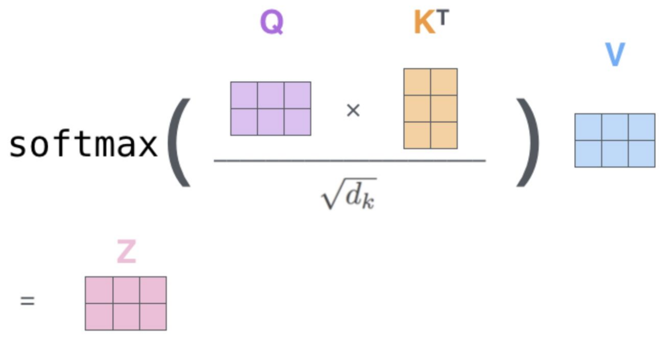
Pass result through a softmax operation (all positive and add up to 1).

Intuition: softmax score determines how much each word will be expressed at this position.

Source: https://jalammar.github.io/illustrated-transformer/

Attention Basics - Self Attention



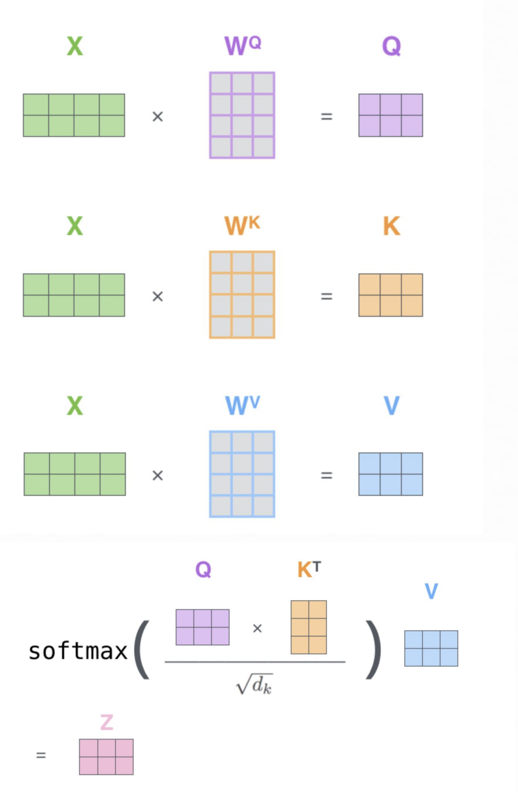


The self-attention calculation in matrix form

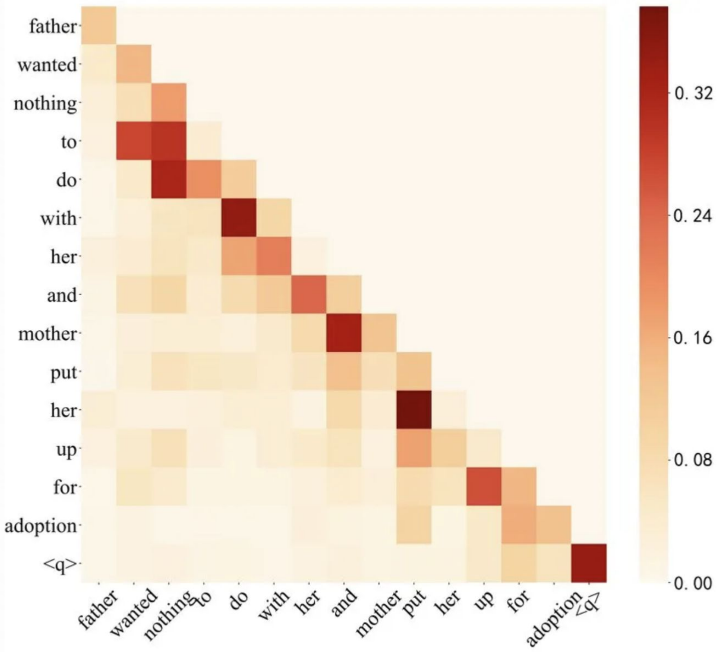
Sum up the weighted **value vectors**. This produces the output of **the self-attention layer at this position.**

Source: https://jalammar.github.io/illustrated-transformer/

Attention Basics - Self Attention



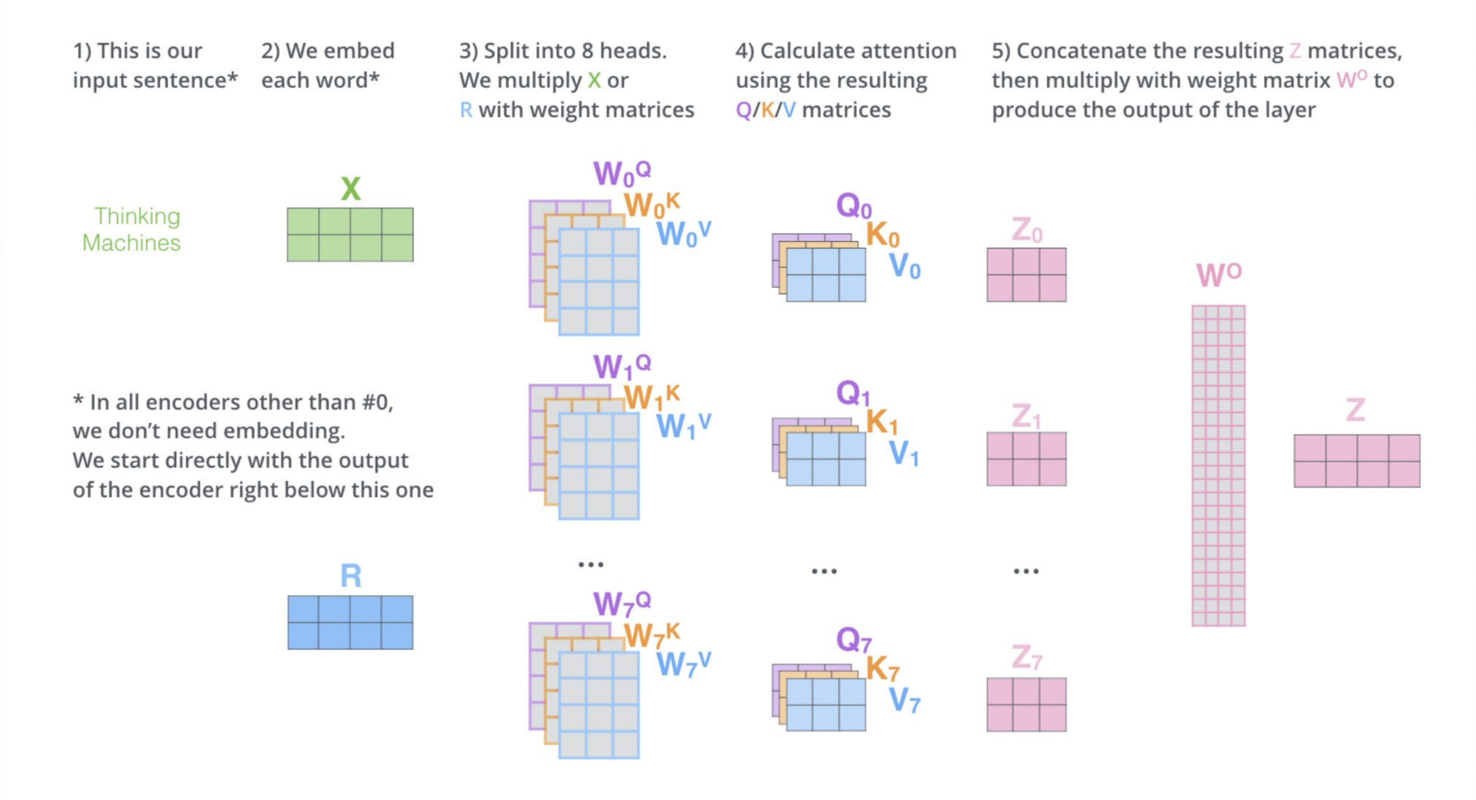
The self-attention calculation in matrix form



Example Output

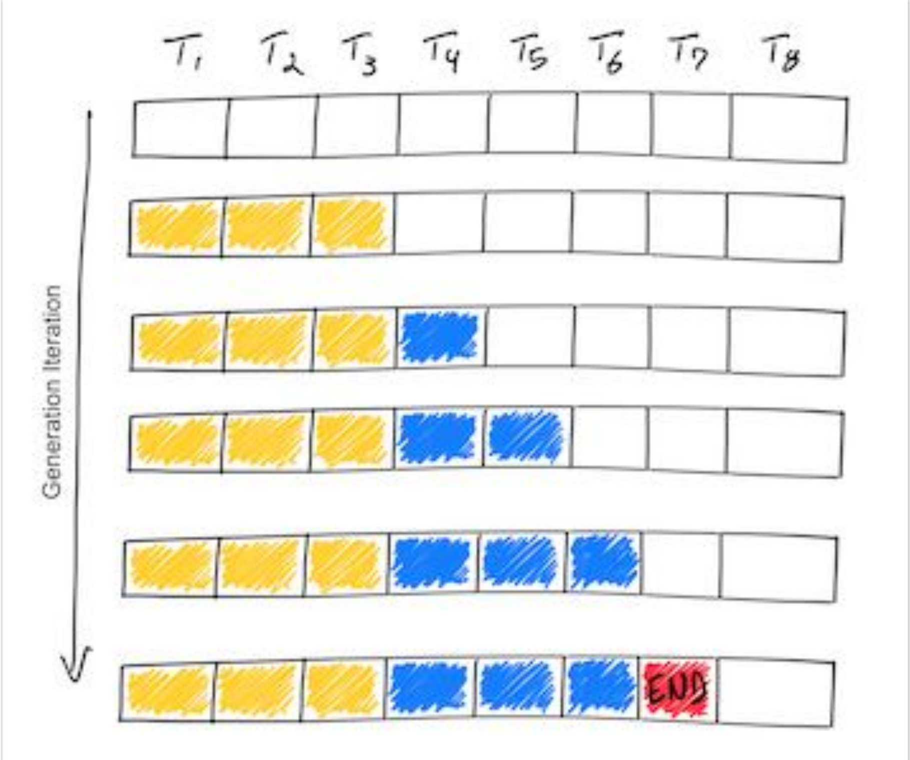
Source: https://medium.com/@AIExplainedML/how-does-the-attention-mechanism-in-gpt-models-work-5f489a59346b

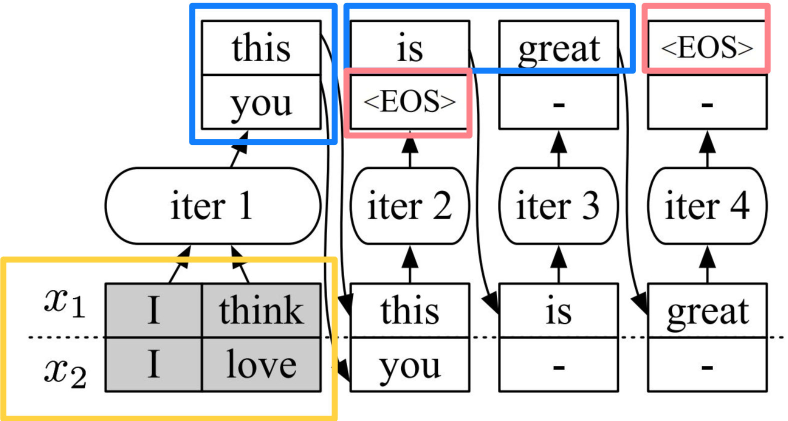
Multi-Headed Attention



Source: https://jalammar.github.io/illustrated-transformer/

LLM Inference Background





Legend

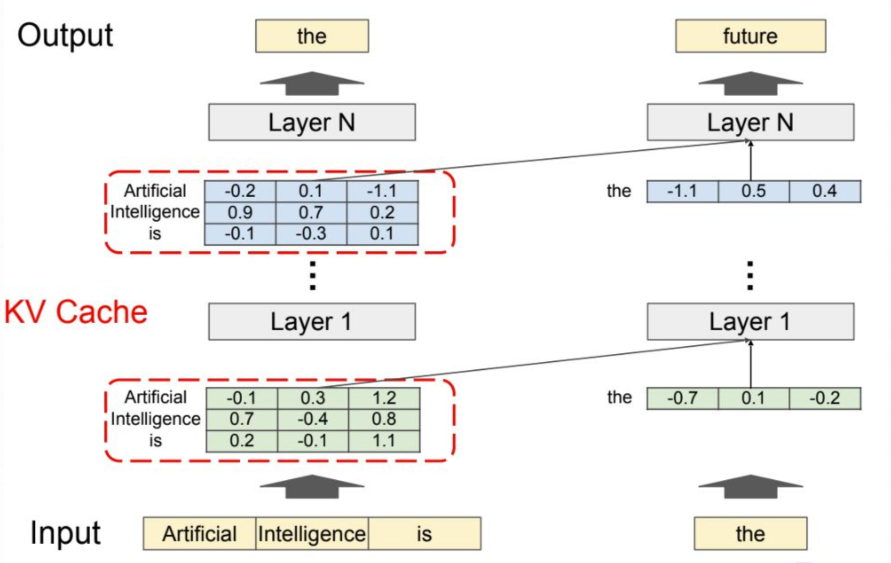
- Yellow: Prompt Token
- Blue: Generated Token
- Red: <EOS> Token

Source: https://www.anyscale.com/blog/continuous-batching-llm-inference

KV Caching

Observation: We don’t need to recompute the keys and values of previous tokens!

- Cache the previous Keys and Values, compute current token’s attention scores/context with cached Keys and Values.
- KV Cache dynamically grows with newly generated tokens, and shrinks as tokens are deleted upon sequence completion.



LLM Serving

- Handles multiple user requests for LLM inference (e.g., ChatGPT)
- Runs on high-end GPUs (e.g., NVIDIA A100, H100)
- Limited throughput:
  - 1 A100 GPU processes **< 1 request/second** for LLaMA-13B with moderate inputs
- Production-scale services require **thousands of GPUs**

GPT-3 175B

Cost of serving

$476,491.44 per year

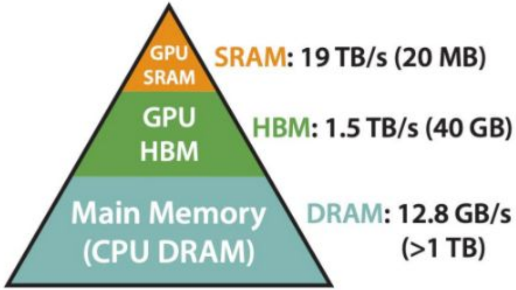
$53.934 per hour for 1 instance

$190.6M per year

for 400 instances

System Challenges That Increase Cost

- **Size of LLMS**
    - LLaMA-13B: 13GB to store float16 parameters
    - 7xA100-80GB GPUs needed to maximize throughput
- **Memory I/O**
    - Single token generation requires loading 13GB to compute cores
    - CPU Memory I/O: 10-40 GB/s
    - GPU Memory I/O: 2000 GB/s (A100-80GB)
- **High Throughput Requires Many FLOPs**
    - CPUs: Can generate a single sequence in real-time
    - GPUs: Can generate many sequences in real-time

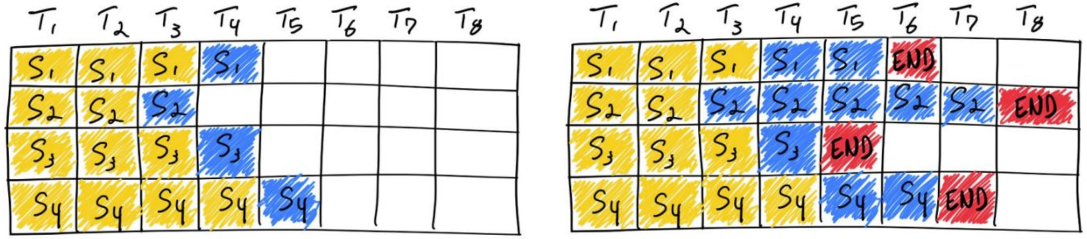


Memory Hierarchy with Bandwidth & Memory Size

From the FlashAttention paper
https://arxiv.org/pdf/2205.14135.pdf

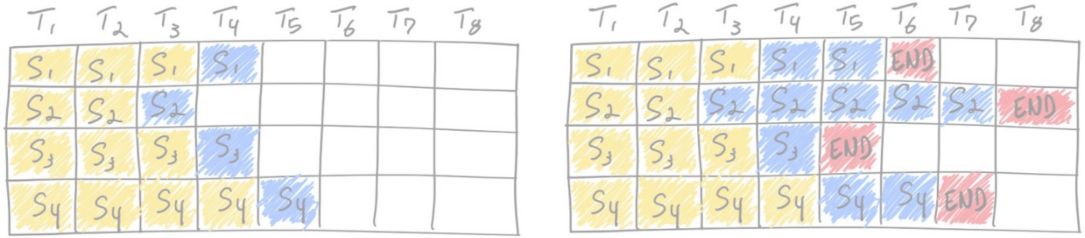
Batching

- Batching multiple **<u>sequences</u>** together on a GPU "**static batching**"
- Problem: GPU utilization drops as some sequences are completed earlier than others.

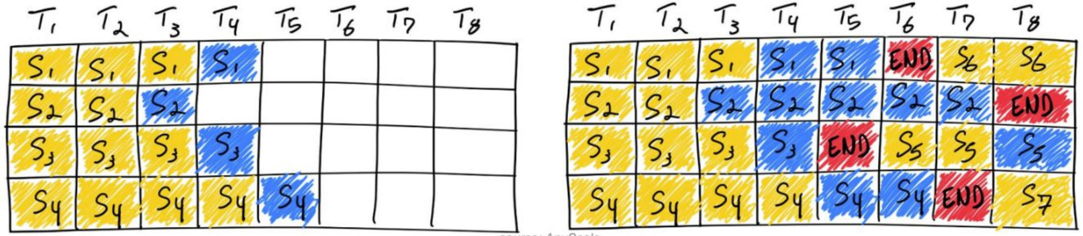


Batching

- Batching multiple **sequences** together on a GPU “**static batching**”
- Problem: GPU utilization drops as some sequences are completed earlier than others.



- Batching multiple **iterations** together on a GPU **“continuous batching”** aka **“iteration-level scheduling”**



source: AnyScale

02

Orca

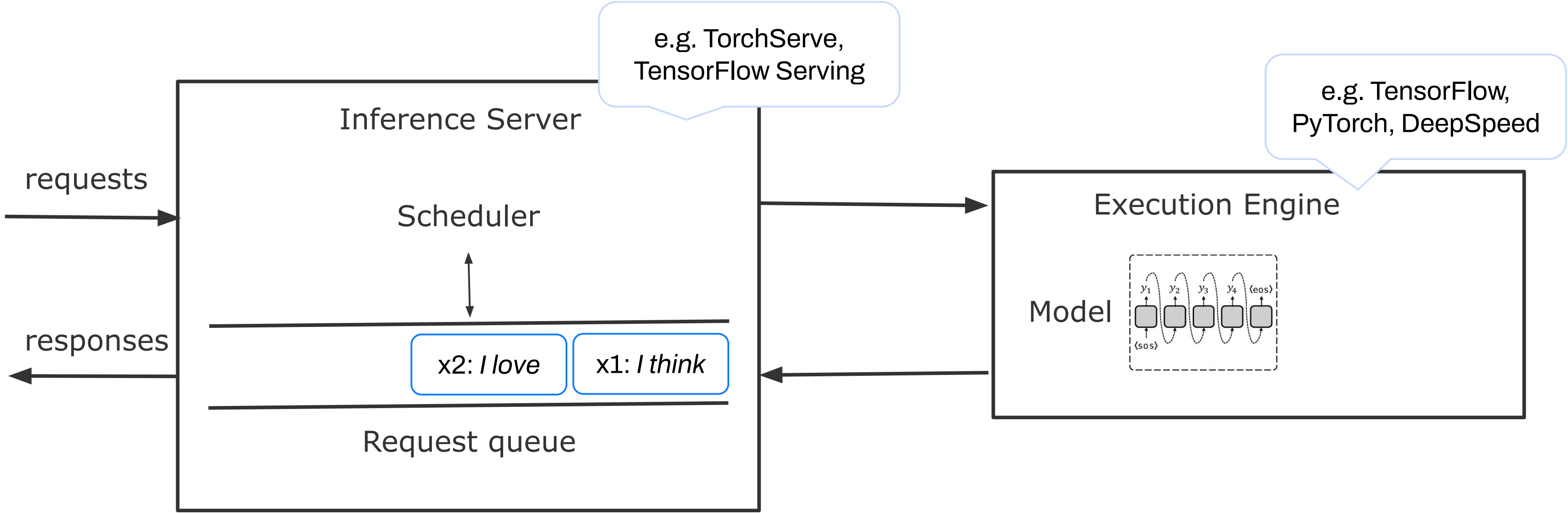
A Distributed Serving System for Transformer-Based Generative Models

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun

We focus on how to improve the throughput of serving transformer-based generative models to reduce the cost.

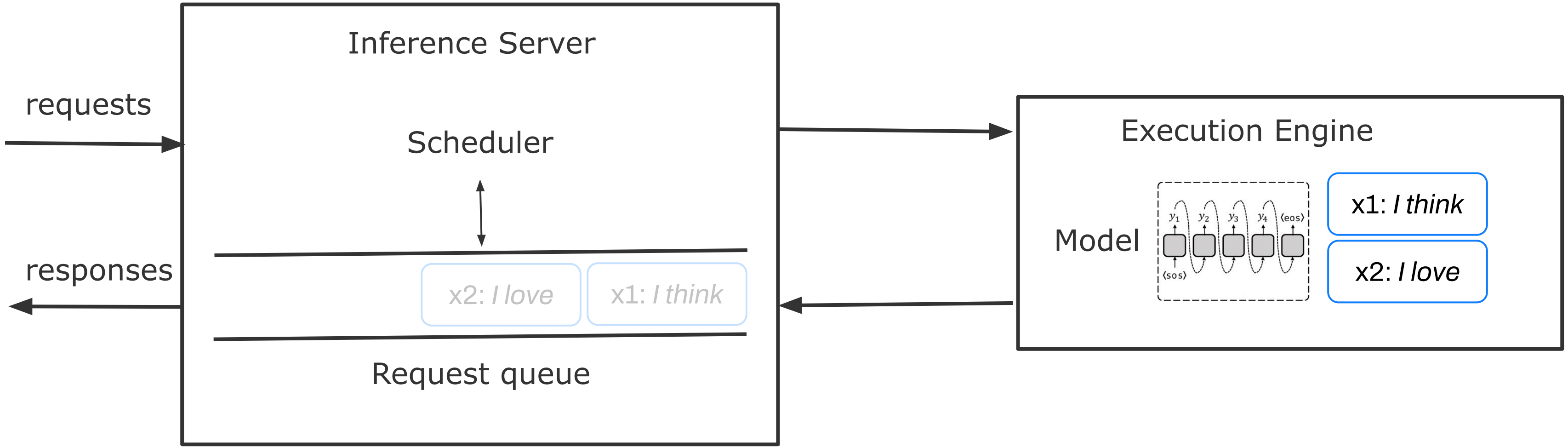
- **Authors of *Orca***

LLM Inference
General Architecture



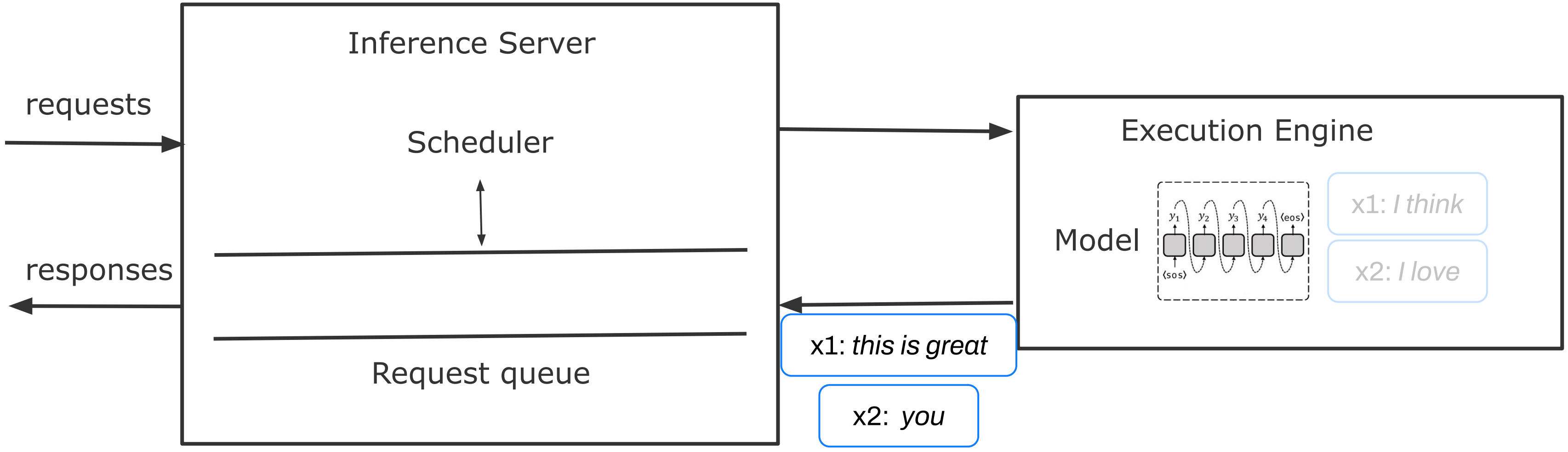
* maximum batch size = 3

LLM Inference
General Architecture



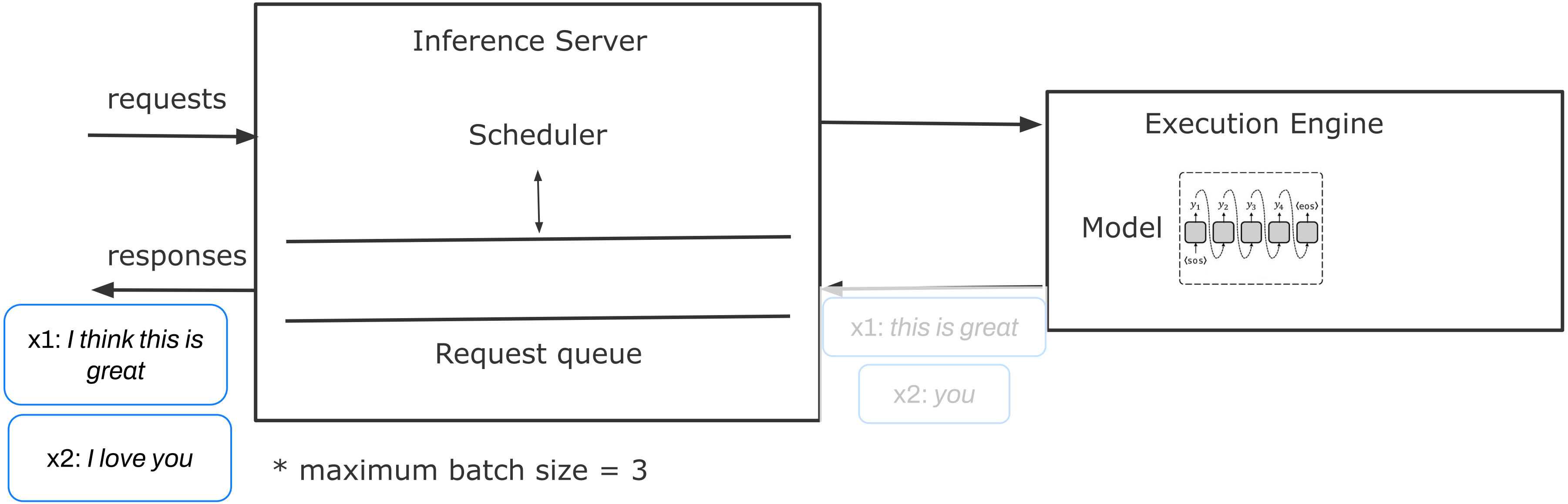
* maximum batch size = 3

LLM Inference
General Architecture

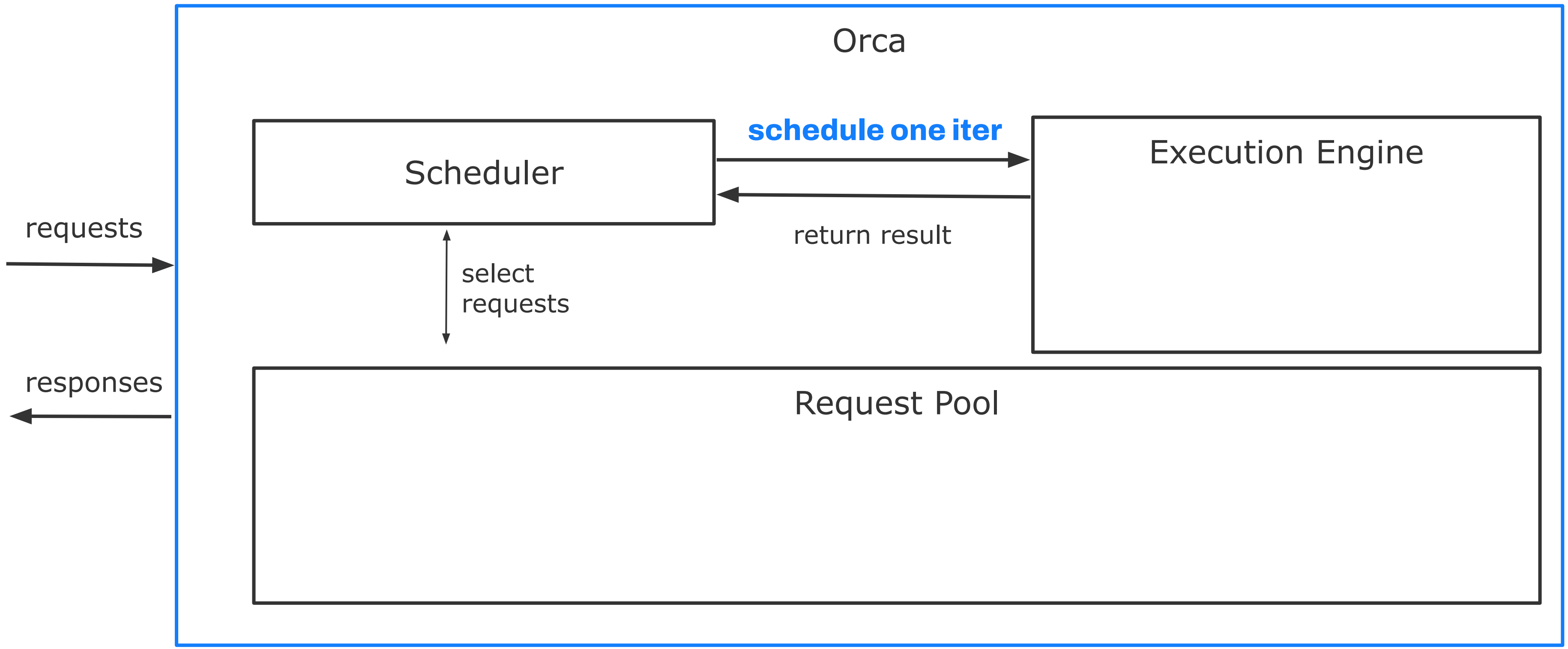


* maximum batch size = 3

LLM Inference
General Architecture

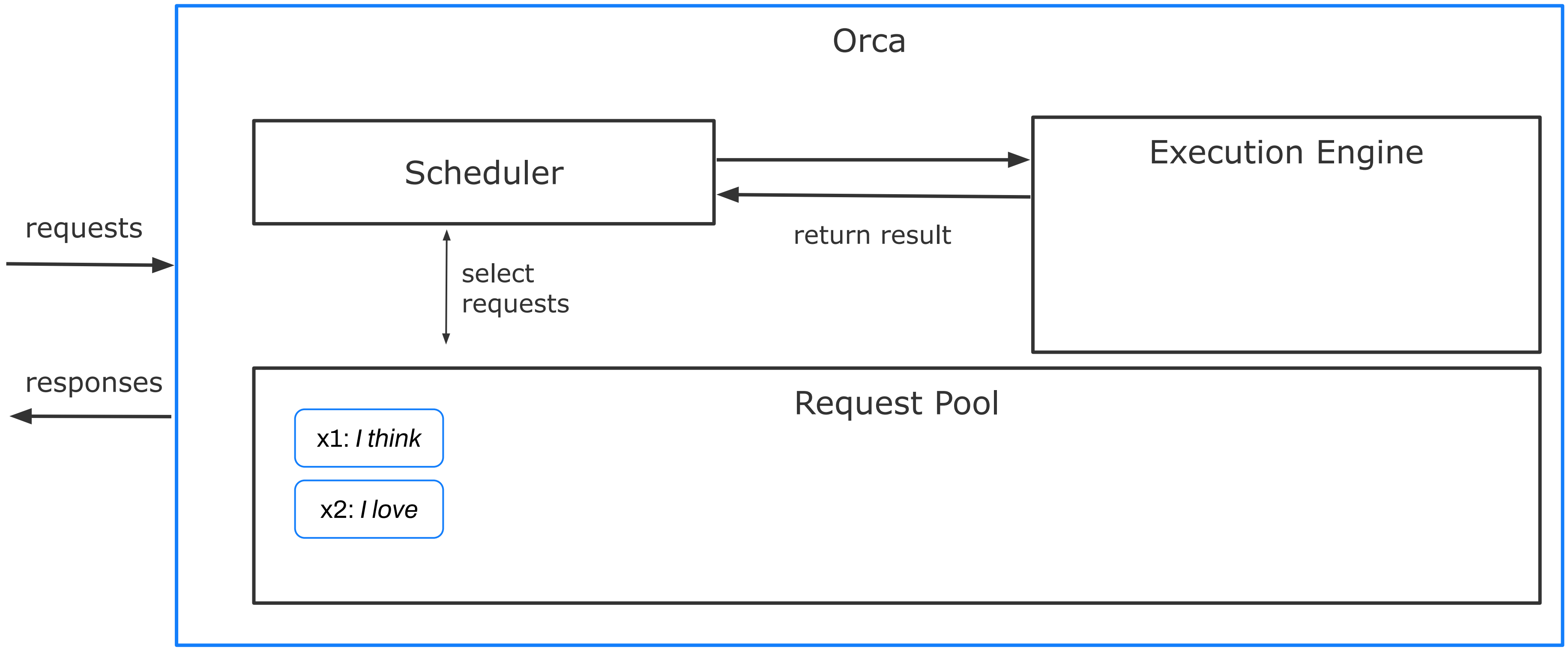


Solution 1: Iteration-Level Scheduling



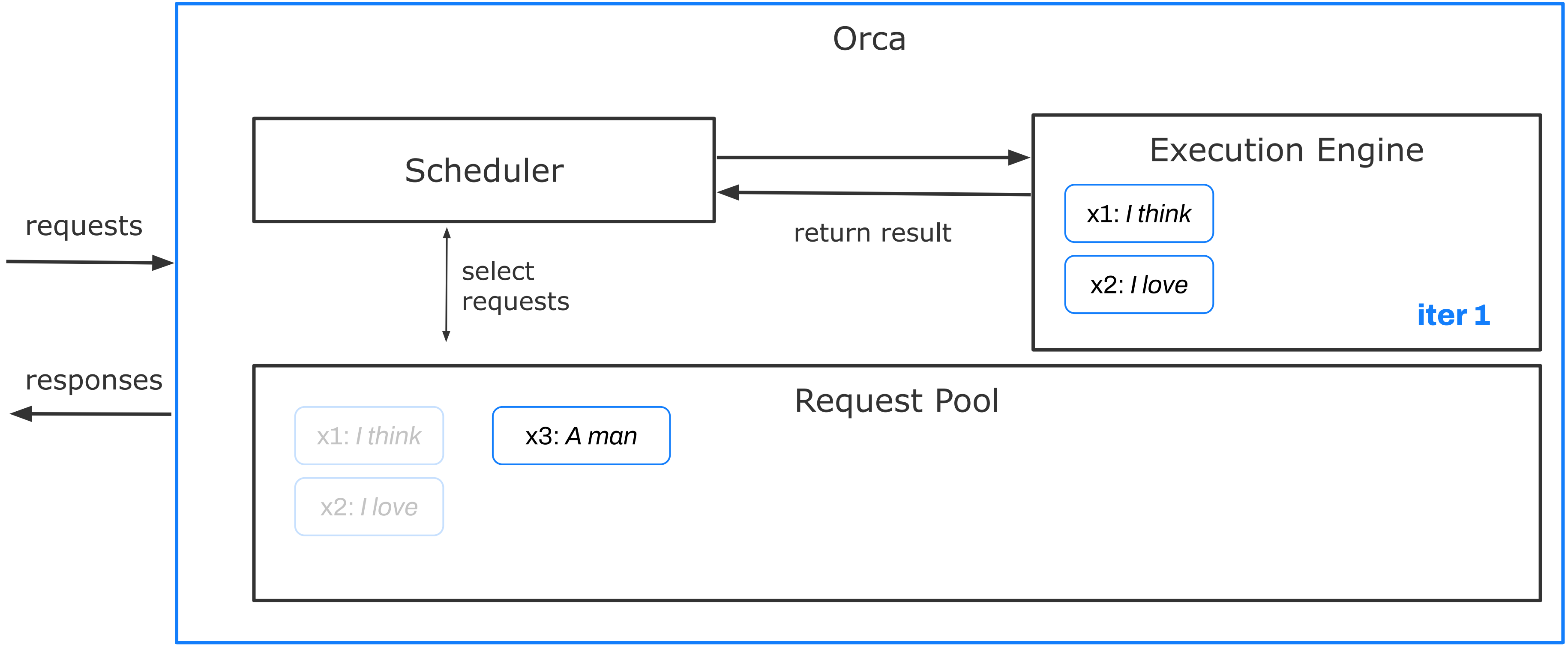
* maximum batch size = 3

Solution 1: Iteration-Level Scheduling



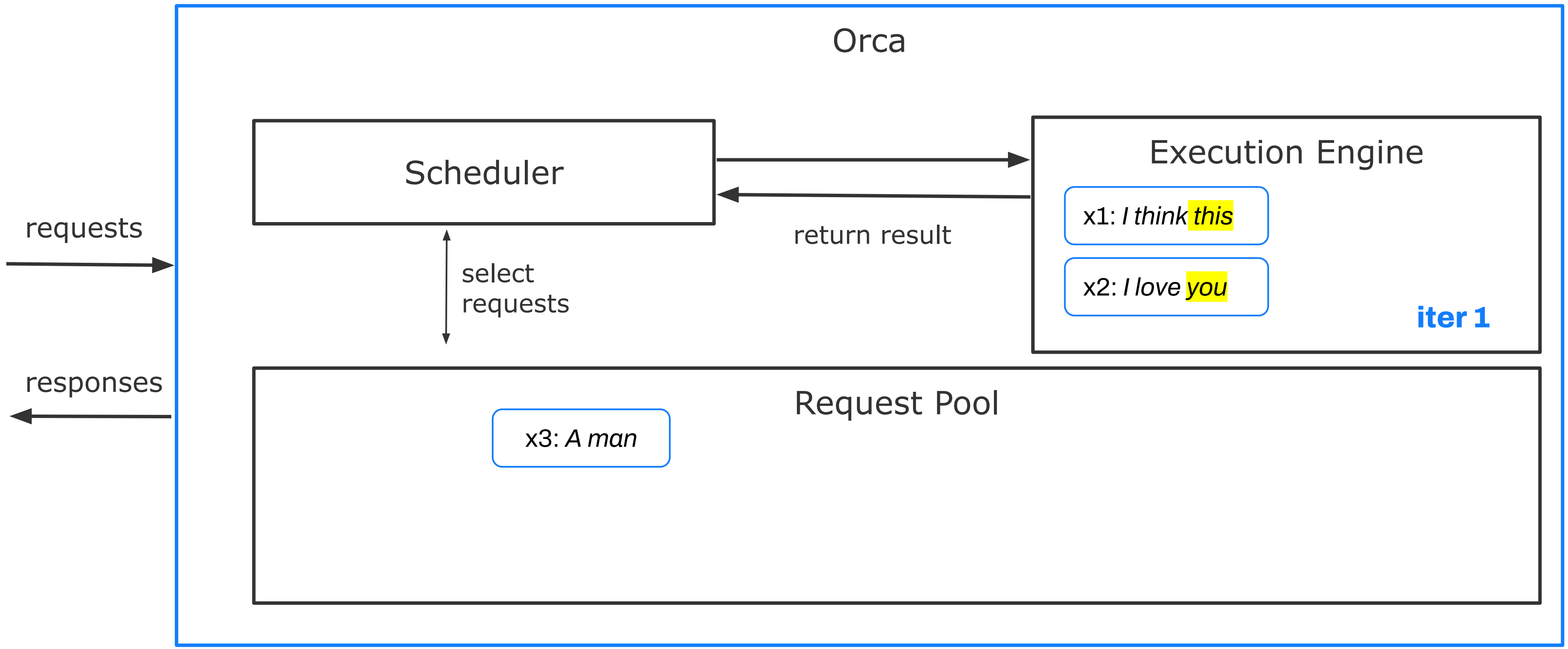
* maximum batch size = 3

Solution 1: Iteration-Level Scheduling



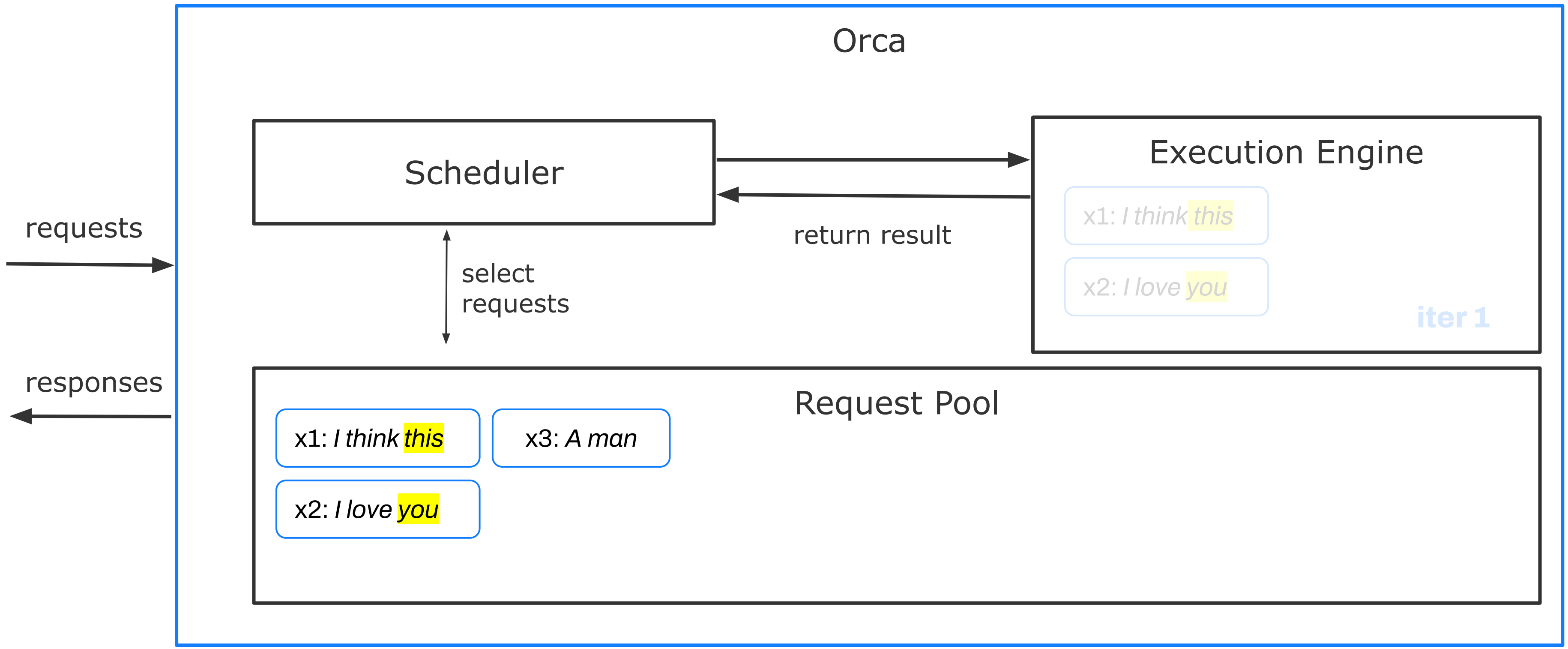
* maximum batch size = 3

Solution 1: Iteration-Level Scheduling



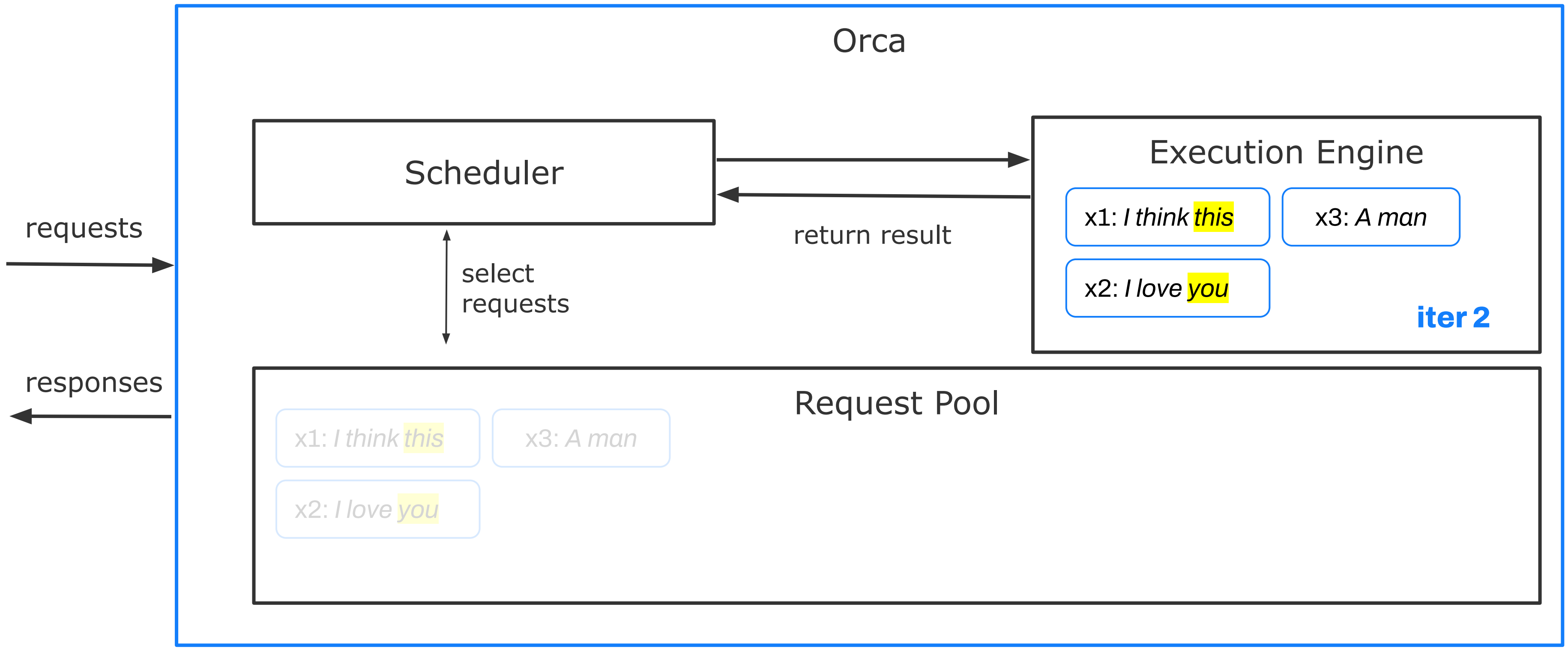
* maximum batch size = 3

Solution 1: Iteration-Level Scheduling



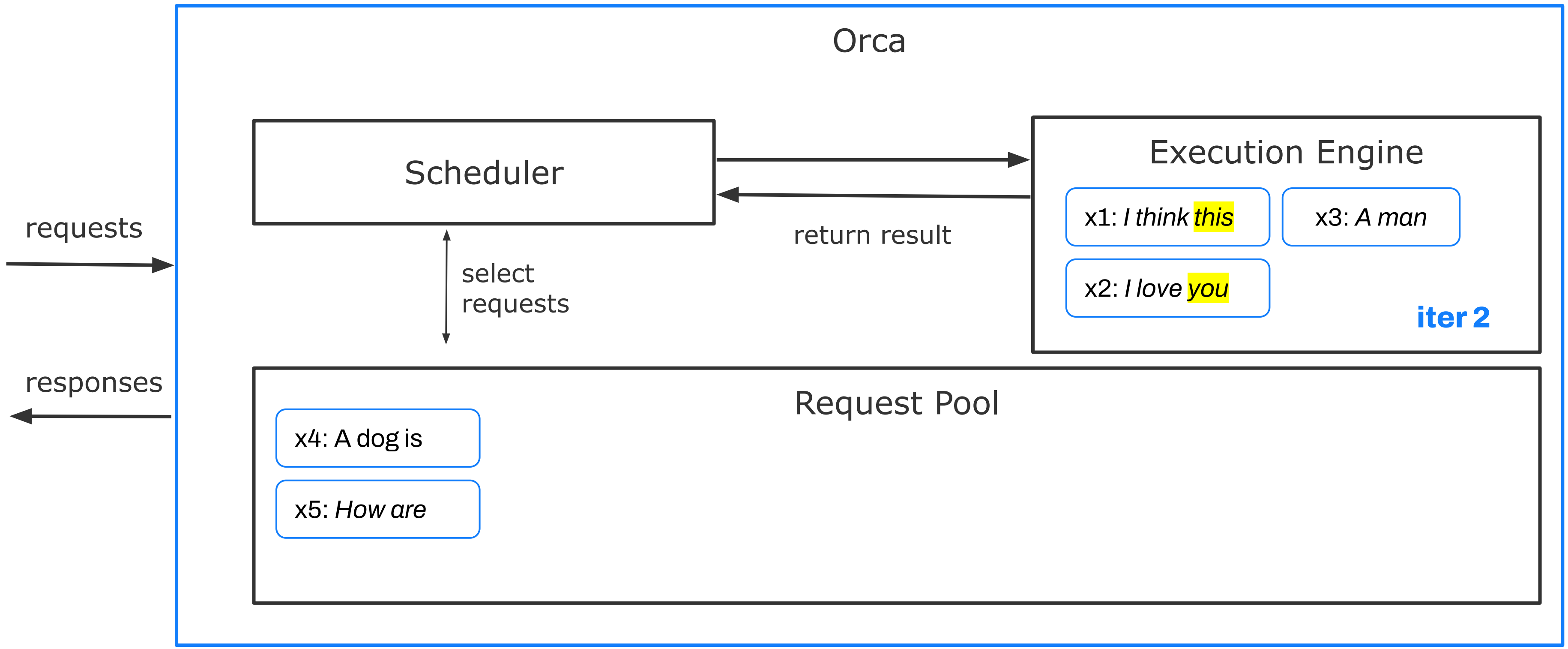
* maximum batch size = 3

Solution 1: Iteration-Level Scheduling



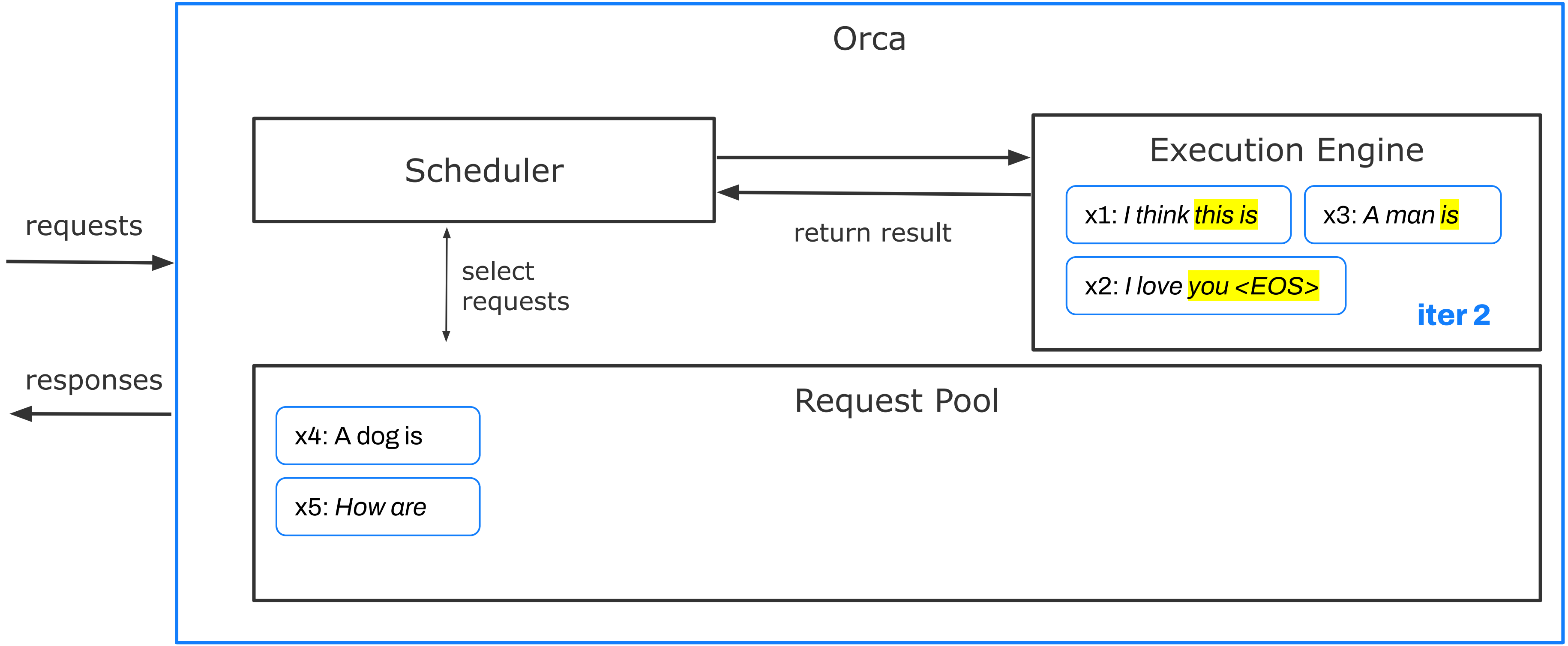
* maximum batch size = 3

Solution 1: Iteration-Level Scheduling



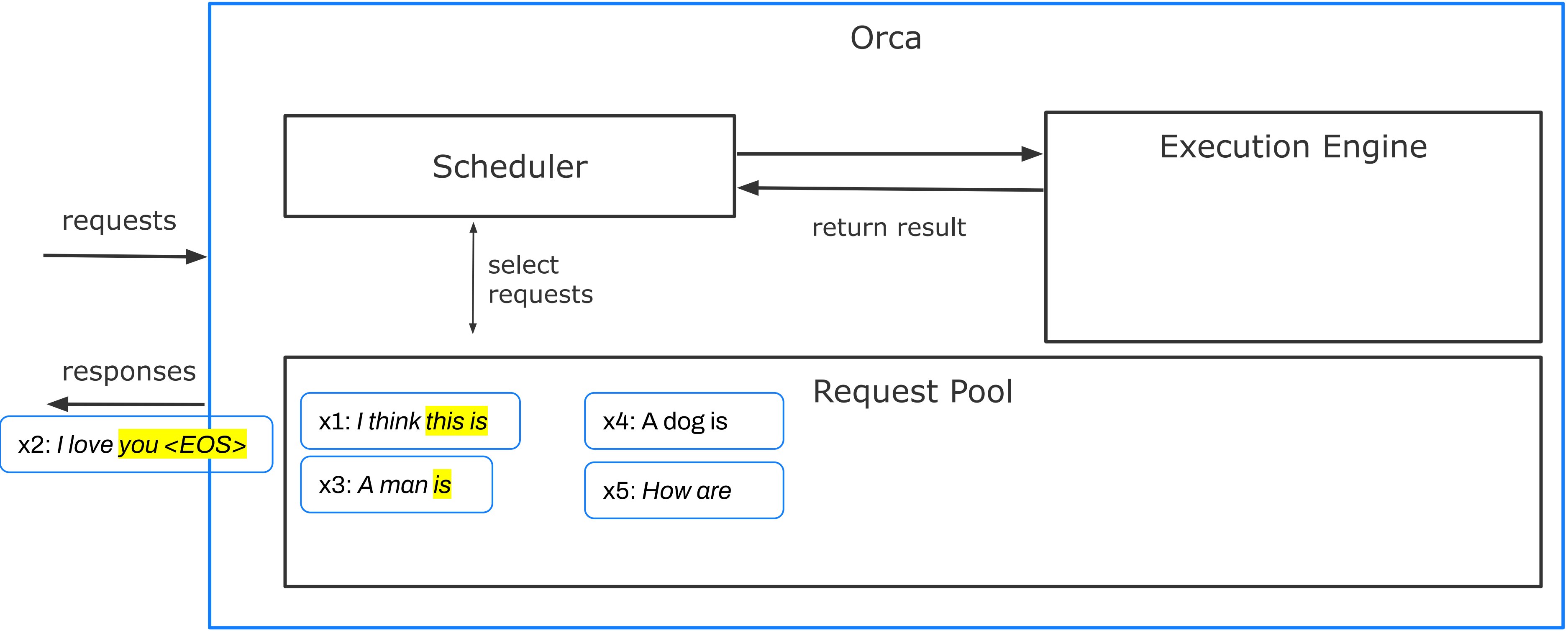
* maximum batch size = 3

Solution 1: Iteration-Level Scheduling



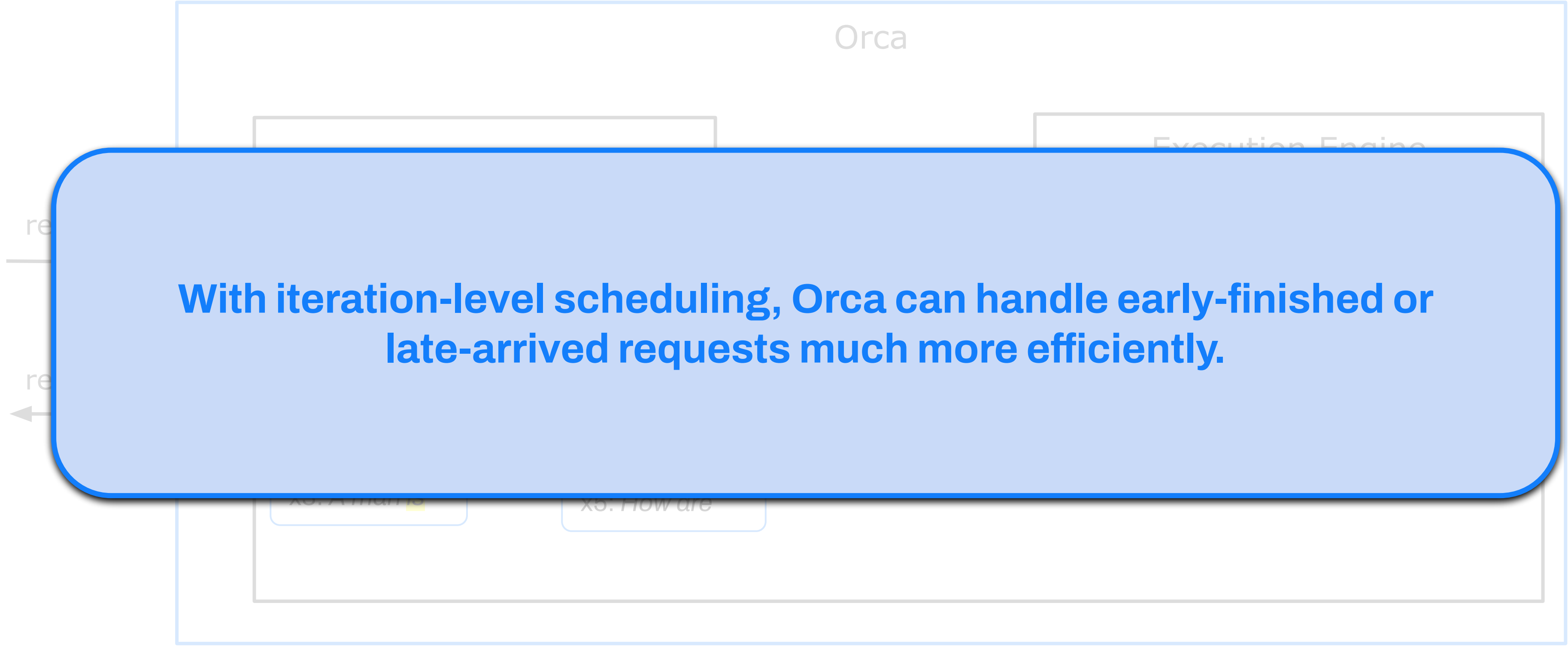
* maximum batch size = 3

Solution 1: Iteration-Level Scheduling



* maximum batch size = 3

Solution 1: Iteration-Level Scheduling

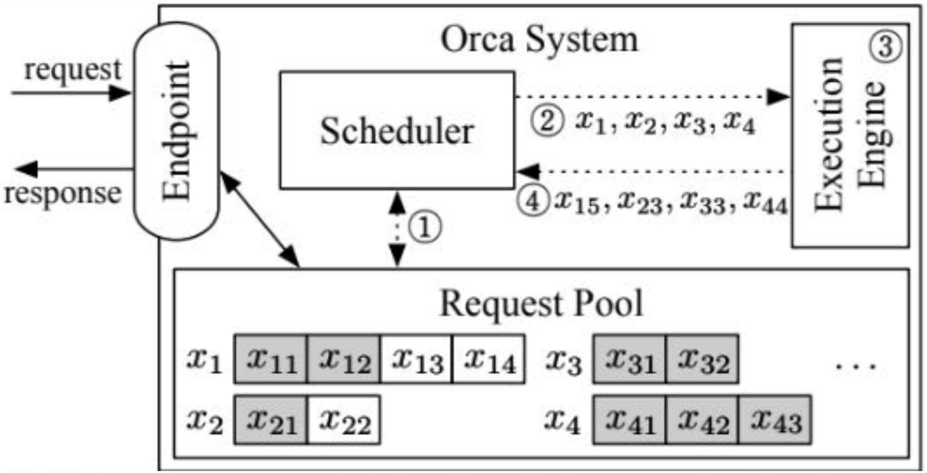


With iteration-level scheduling, Orca can handle early-finished or late-arrived requests much more efficiently.

* maximum batch size = 3

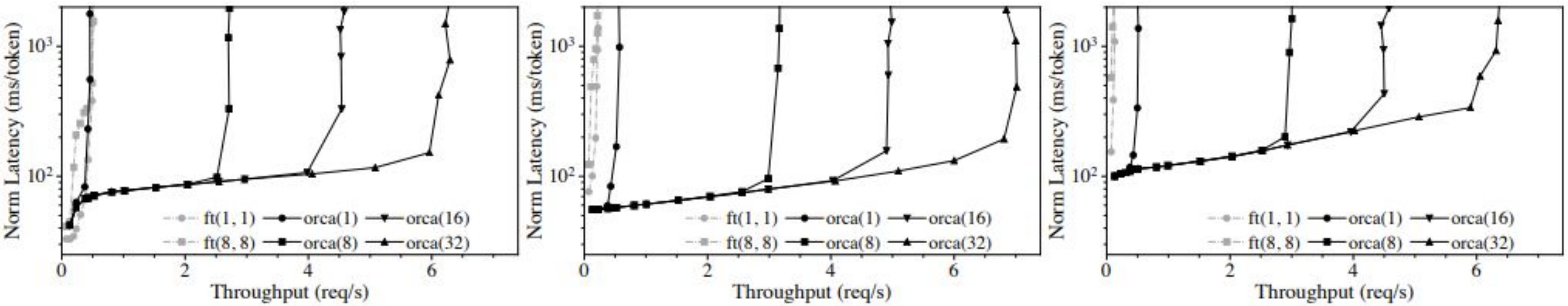
Orca Scheduling Summary

1. Requests are added to the request pool.
2. Scheduler fetches requests from pool in FIFO order.
3. Scheduled requests are run for a single iteration.
4. New token appended to sequence.
5. If sequence complete, return result to endpoint.
6. Otherwise, add sequence back to request pool.



Source: Orca A Distributed Serving System for Transformer-Based Generative Models

Results



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

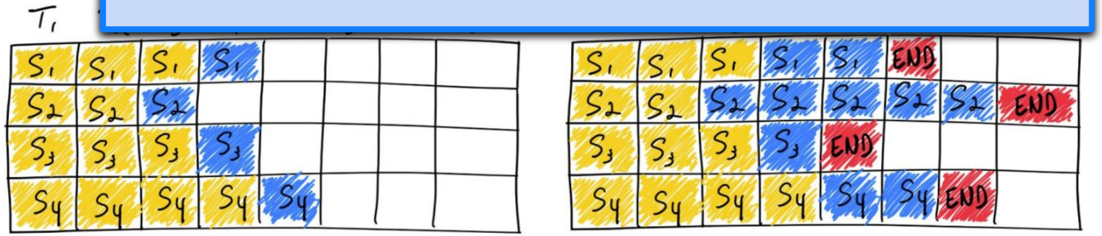
Figure 10: Median end-to-end latency normalized by the number of generated tokens and throughput. Label "orca(*max_bs*)" represents results from ORCA with a max batch size of *max_bs*. Label "ft(*max_bs*, *mbs*)" represents results from FasterTransformer with a max batch size of *max_bs* and a microbatch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

Source: Orca A Distributed Serving System for Transformer-Based Generative Models

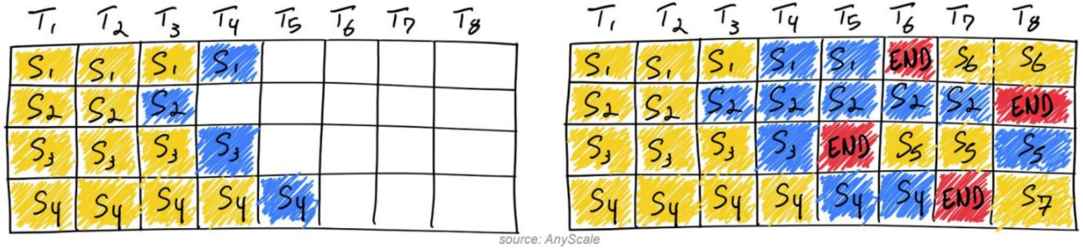
Recall: Batching

What are the trade-offs of iteration-level scheduling? Could there be scenarios where it performs worse than static batching?

- Batching multiple **sequences** together on a GPU "**static batching**"
- Problem: GPU utilization drops as some sequences are completed earlier than others.



- Batching multiple **iterations** together on a GPU **"continuous batching"** aka **"iteration-level scheduling"**



source: AnyScale

03

Paged Attention

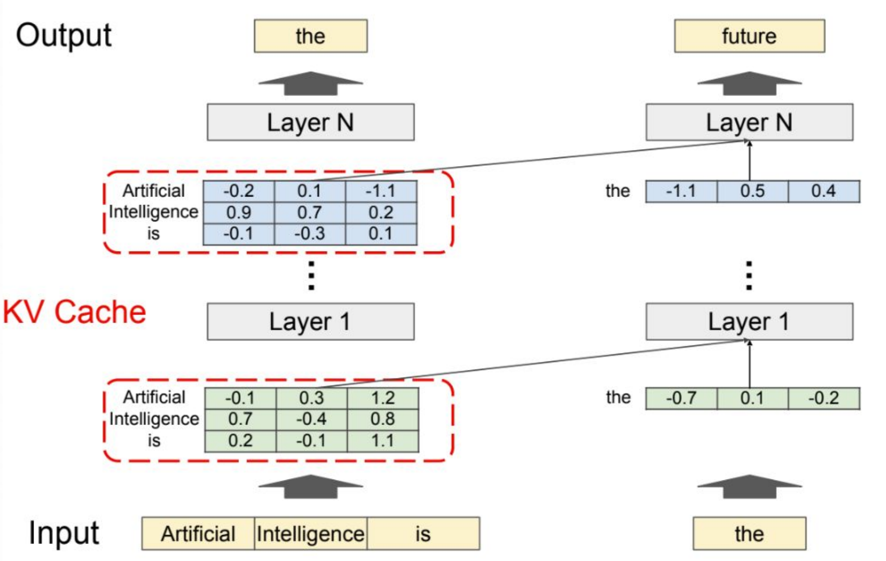
Efficient Memory Management for Large Language Model Serving with PagedAttention

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, Ion Stoica

KV Caching

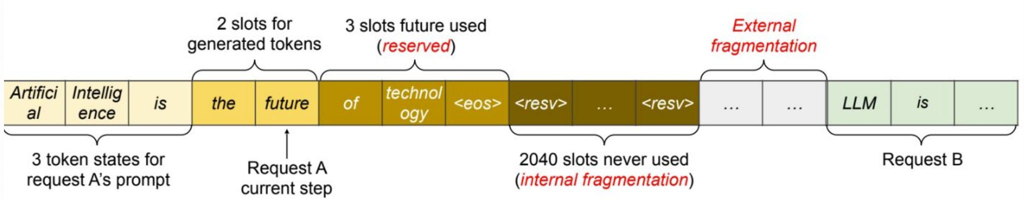
Observation: We don't need to recompute the keys and values of previous tokens!

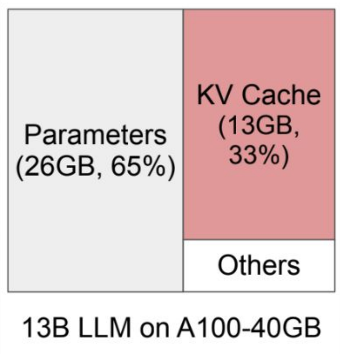
- Cache the previous Keys and Values, compute current token's attention scores/context with cached Keys and Values.
- KV Cache dynamically grows with newly generated tokens, and shrinks as tokens are deleted upon sequence completion.
- Add each newly generated token to sequence's KV cache.

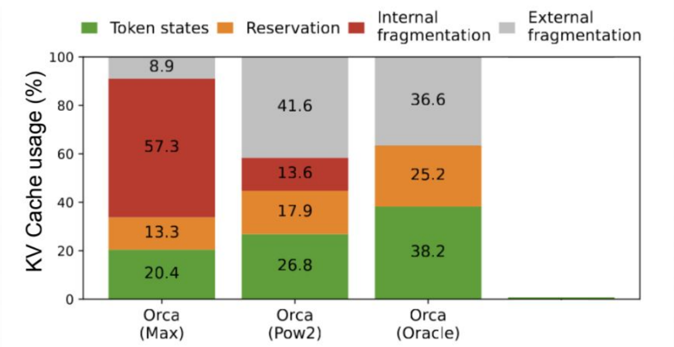


KV Cache Memory Waste

- Only about **20-40%** of KV Cache memory is utilized to store token states
- **Reservation**: Memory allocated for cache, but currently unused.
- **Internal Fragmentation**: Over-allocated memory due to unknown sequence lengths.
- **External Fragmentation**: Due to different sequence lengths.



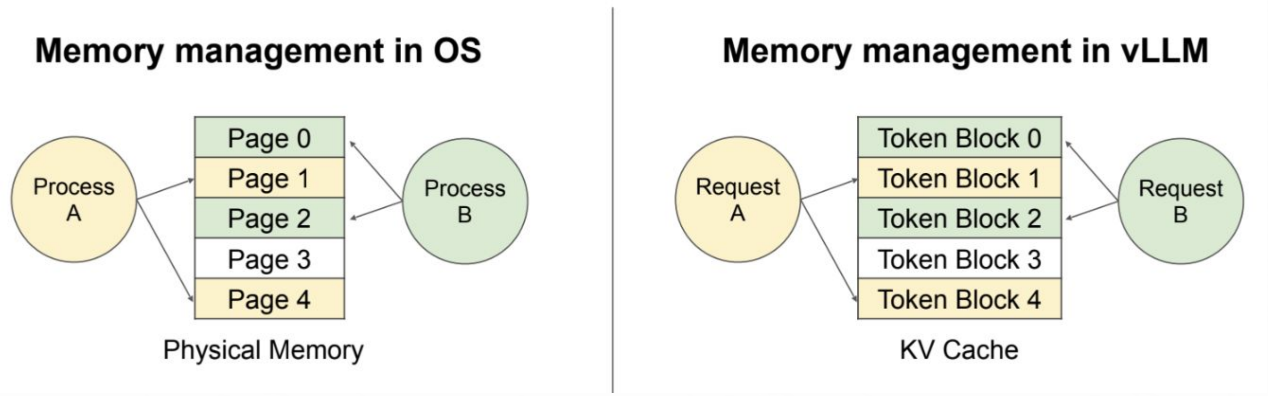




Source: Efficient Memory Management for Large Language Model Serving with *PagedAttention*

Paged Attention: Motivation

- KV Caching has a lot of memory waste, can we do better?
- Operating systems use virtual addressing and paging to efficiently manage memory to eliminate external fragmentation, while limiting internal fragmentation.
  - Non-contiguous memory allocation
- Page sharing: Multiple processes' page tables point to the same underlying physical frame.
  - Multiple sequences could share the same prefix/prompt. (ex. Beam Search decoding)

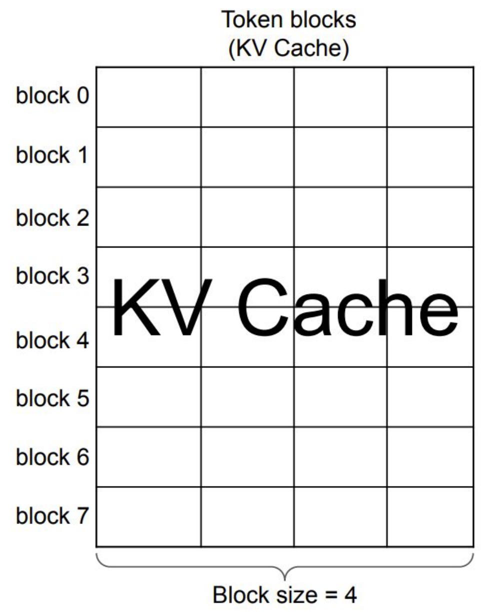


Source: Efficient Memory Management for Large Language Model Serving with *PagedAttention*

Token Blocks

Store tokens in small, contiguous chunks/blocks of memory.

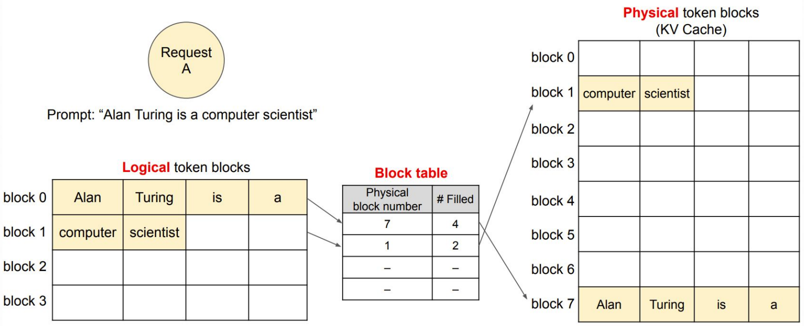
Allocate new blocks as existing blocks get filled.



Paged Attention

- Each request's KV cache is now a series of **Logical KV Blocks** (analogous to virtual memory addresses).
- When computing attention scores, the **KV Block Manager** maps Logical KV Blocks to **Physical KV Blocks**.
- While decoding add tokens from left to right to the last token block. Allocate a new token block if all previous blocks are full.

Below: Block size = 4



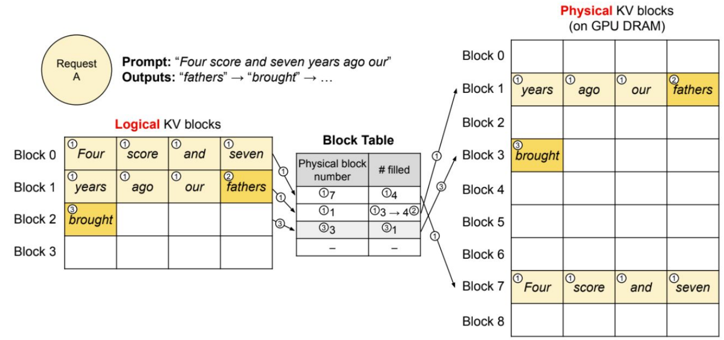


Figure 6. Block table translation in vLLM.

Block Translation

Prompt: “According to all known laws of”

Logical Token Blocks

Logical Block				
0	According	to	all	known
1	laws	of		
2				
3				

Block Table

Logical Block #	Physical Block #	# Filled
0	1	4
1	4	2

Physical Token Blocks

Physical Block				
0				
1	According	to	all	known
2				
3				
4	laws	of		
5				
6				
7				

Block Translation

Prompt: “According to all known laws of”
Completion: “**aviation**”

Logical Token Blocks

Logical Block				
0	According	to	all	known
1	laws	of	aviation	
2				
3				

Block Table

Logical Block #	Physical Block #	# Filled
0	1	4
1	4	3

Physical Token Blocks

Physical Block				
0				
1	According	to	all	known
2				
3				
4	laws	of	aviation	
5				
6				
7				

Block Translation

Prompt: “According to all known laws of”
Completion: “aviation **<u>there</u>**”

Logical Token Blocks

Logical Block				
0	According	to	all	known
1	laws	of	aviation	there
2				
3				

Block Table

Logical Block #	Physical Block #	# Filled
0	1	4
1	4	4

Physical Token Blocks

Physical Block				
0				
1	According	to	all	known
2				
3				
4	laws	of	aviation	there
5				
6				
7				

Block Translation

Prompt: “According to all known laws of”
Completion: “aviation there **is**”

Logical Token Blocks

Logical Block				
0	According	to	all	known
1	laws	of	aviation	there
2	is			
3				

Block Table

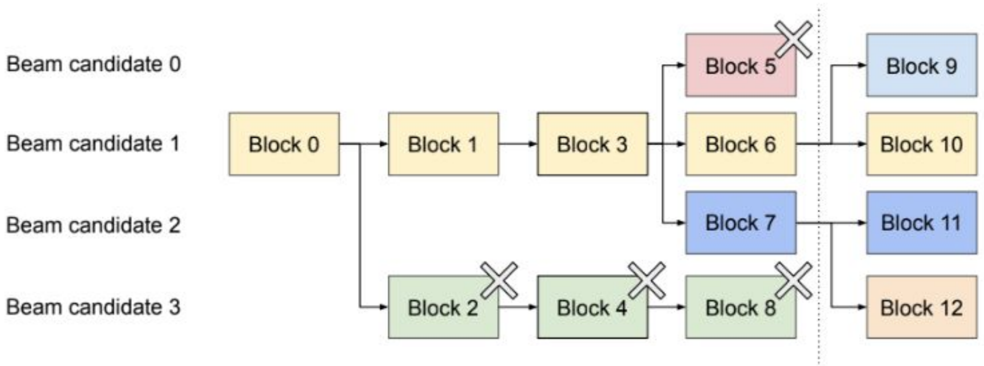
Logical Block #	Physical Block #	# Filled
0	1	4
1	4	4
2	0	1

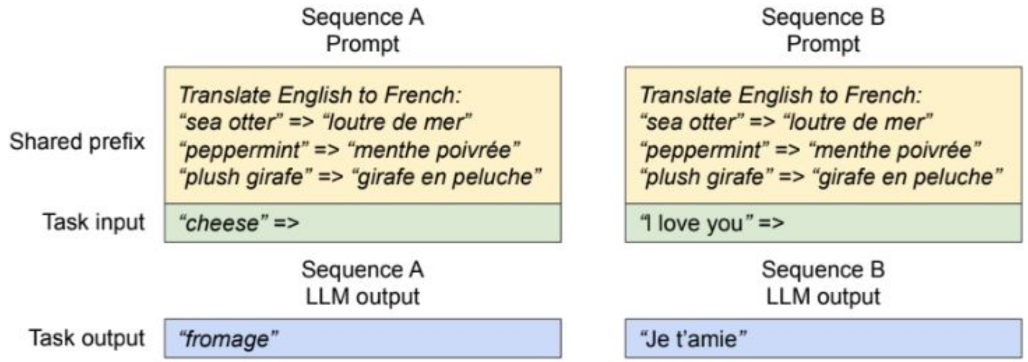
Physical Token Blocks

Physical Block				
0	is			
1	According	to	all	known
2				
3				
4	laws	of	aviation	there
5				
6				
7				

Can we save even more memory?

Can you think of some more optimizations?





Source: Efficient Memory Management for Large Language Model Serving with *PagedAttention*

Block Sharing

- Enables blocks to be shared across multiple requests/decoding streams.
  - Shared prompts or Beam Search

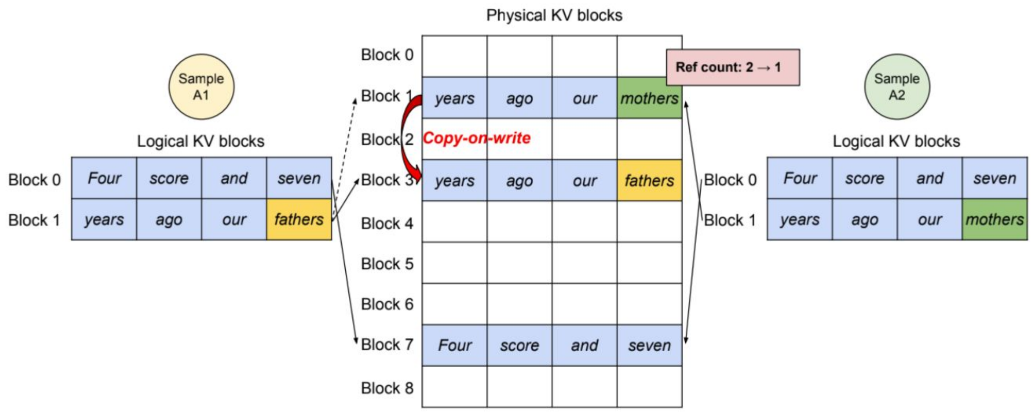


Figure 8. Parallel sampling example.

Source: Efficient Memory Management for Large Language Model Serving with *PagedAttention*

Block Sharing

- **Reference count** in block table to support copy-on-write.

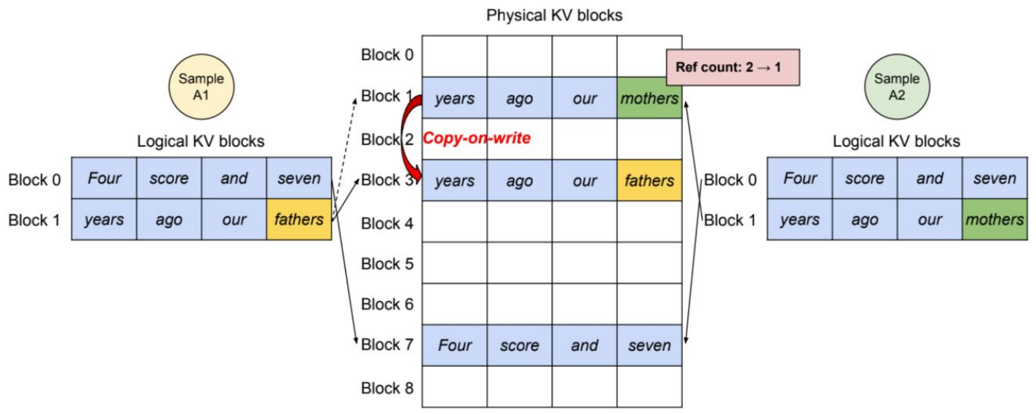
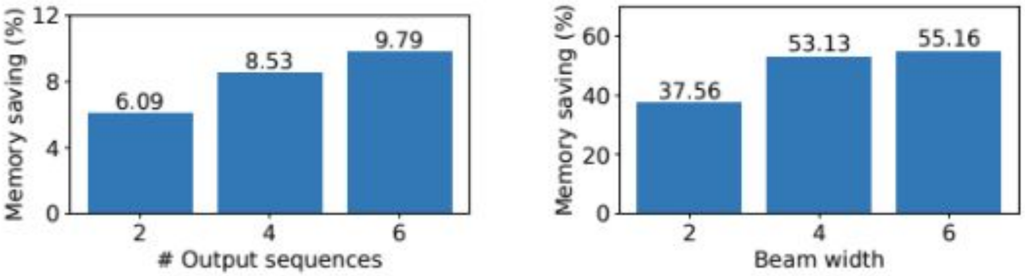


Figure 8. Parallel sampling example.

Source: Efficient Memory Management for Large Language Model Serving with *PagedAttention*

Block Sharing Results



(a) Parallel sampling **(b)** Beam search

Figure 15. Average amount of memory saving from sharing KV blocks, when serving OPT-13B for the Alpaca trace.

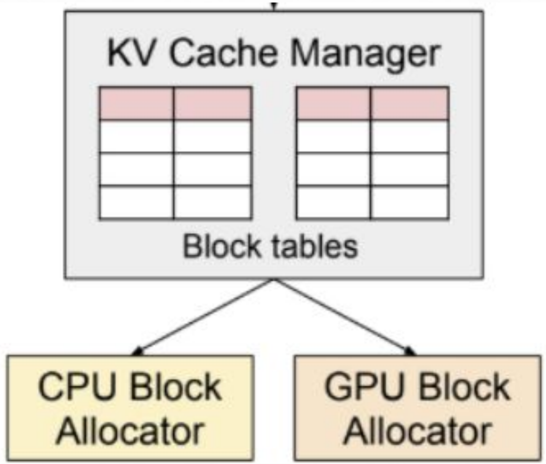
Source: Efficient Memory Management for Large Language Model Serving with *PagedAttention*

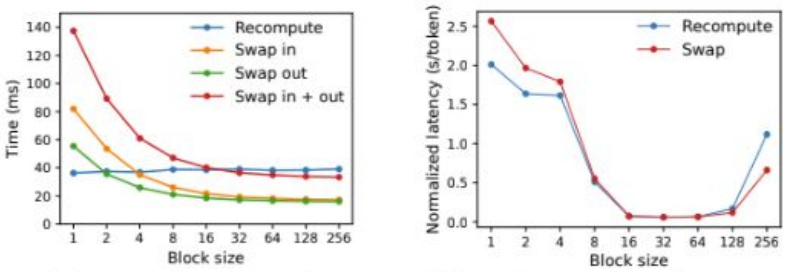
Block Swapping

When GPU memory is exhausted, vLLM evicts blocks from GPU memory.

Evicted blocks are copied to CPU memory.

Why do some block sizes perform better than others?





(a) Microbenchmark (b) End-to-end performance

Source: Efficient Memory Management for Large Language Model Serving with *PagedAttention*

vLLM System Architecture

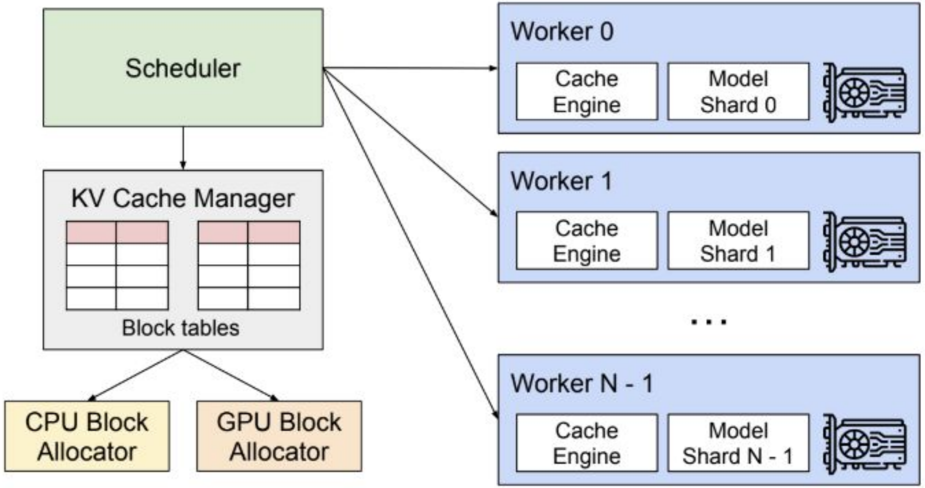
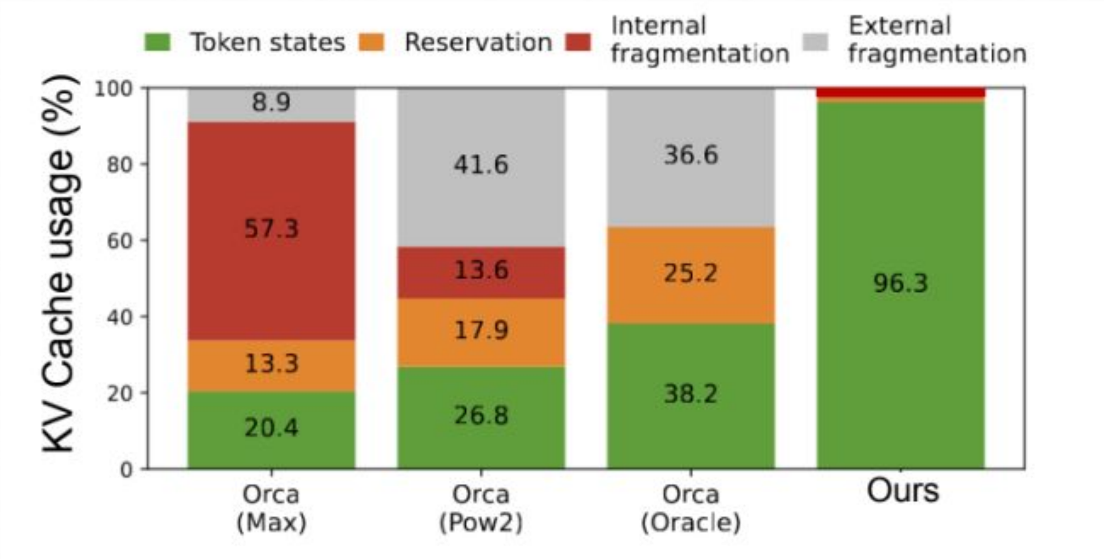


Figure 4. vLLM system overview.

Source: Efficient Memory Management for Large Language Model Serving with *PagedAttention*

vLLM Memory Usage



It’s 1.5-4x better than Orca and other state of the art approaches :)

Source: Efficient Memory Management for Large Language Model Serving with *PagedAttention*

vLLM Performance

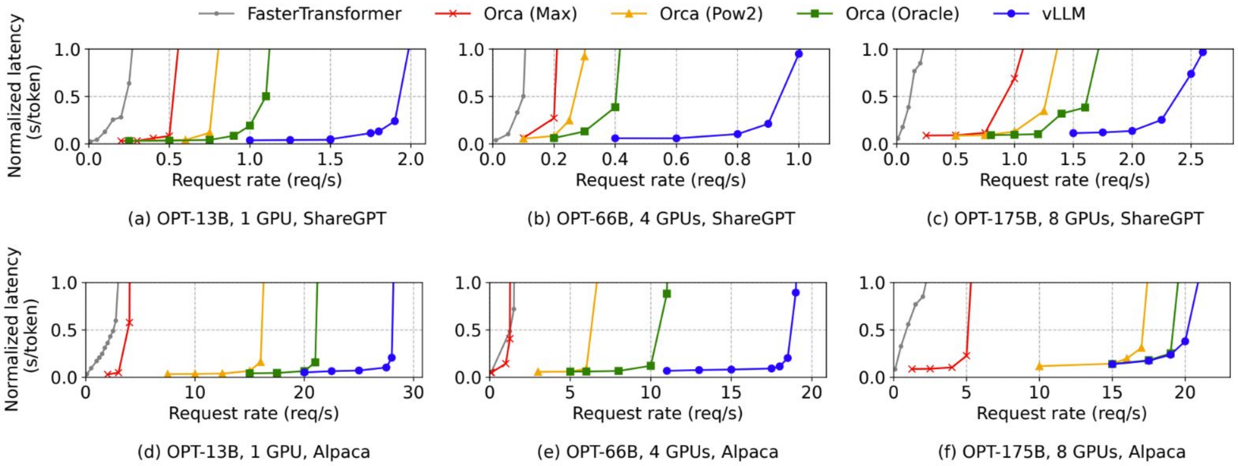


Figure 12. Single sequence generation with OPT models on the ShareGPT and Alpaca dataset

It's 1.5-4x better than Orca and other state of the art approaches :)

Source: Efficient Memory Management for Large Language Model Serving with *PagedAttention*