https://hao-ai-lab.github.io/cse234-w25/

CSE 234: Data Systems for Machine Learning
Winter 2025

LLMSys

Optimizations and Parallelization

MLSys Basics

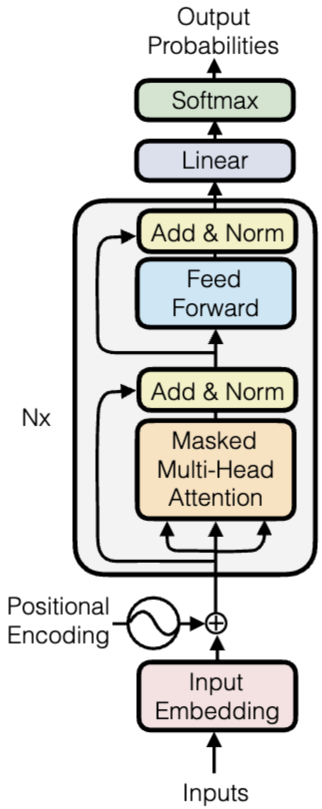
Course Eval

- If 80% of you finish the course eval, all get +2 points in final score!
- No reading summary for week10. Points go to PA3.
- Guest lecture by the author of the ***best speculative decoding*** method on Thursday. Please attend!
- TA will hold recitations for PA3 and exams in week 10 or 11.
  - date TBD.

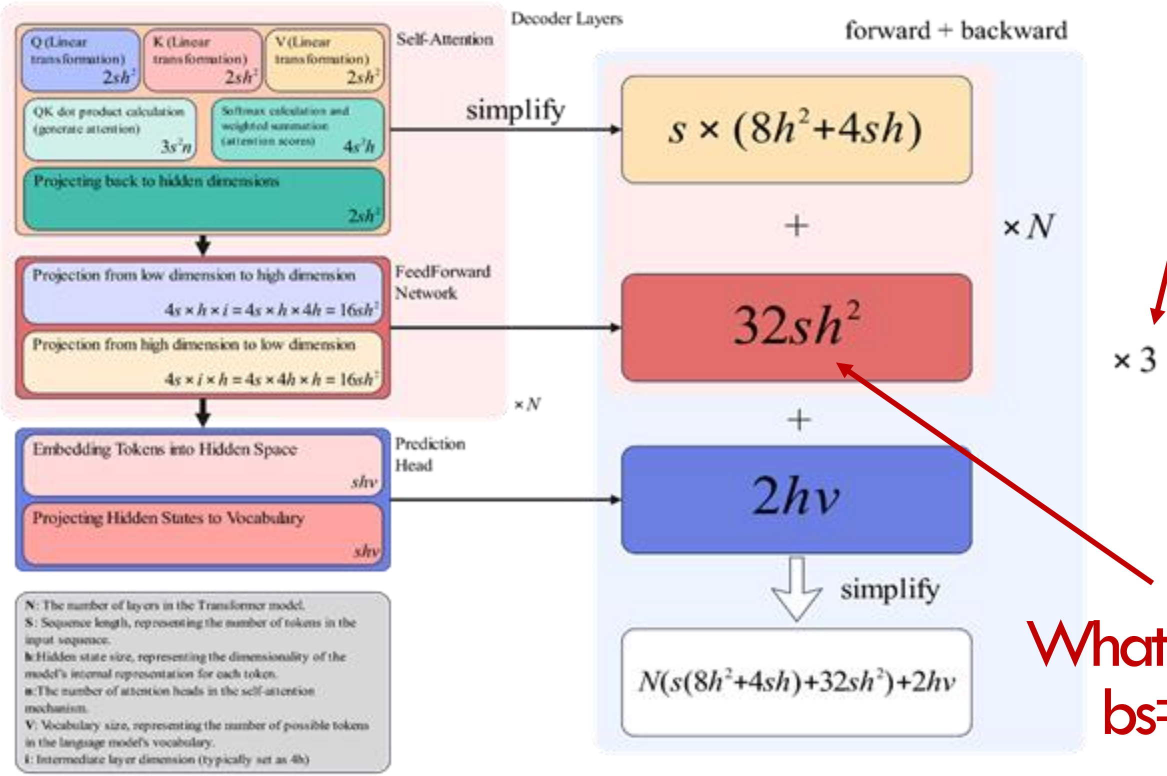
Connecting the Dots: Compute/Comm characteristic of LLMs

Key characteristics: compute, memory, communication

- calculate the number of parameters of an LLM?
- calculate the flops needed to train an LLM?
- calculate the memory needed to train an LLM?



Recap: Compute -- Where is the Potential Bottleneck?



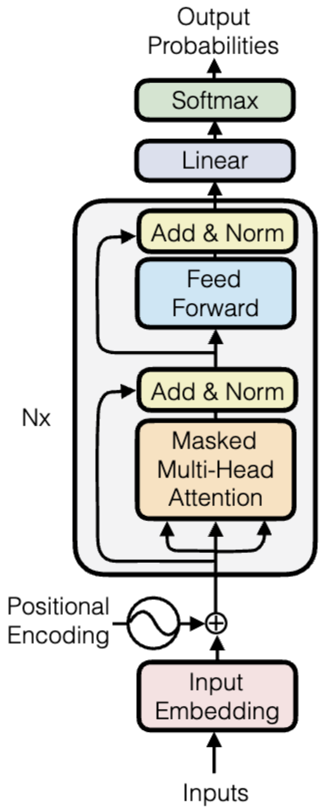
Q1: why x3?

What happens when bs=1 and s=1

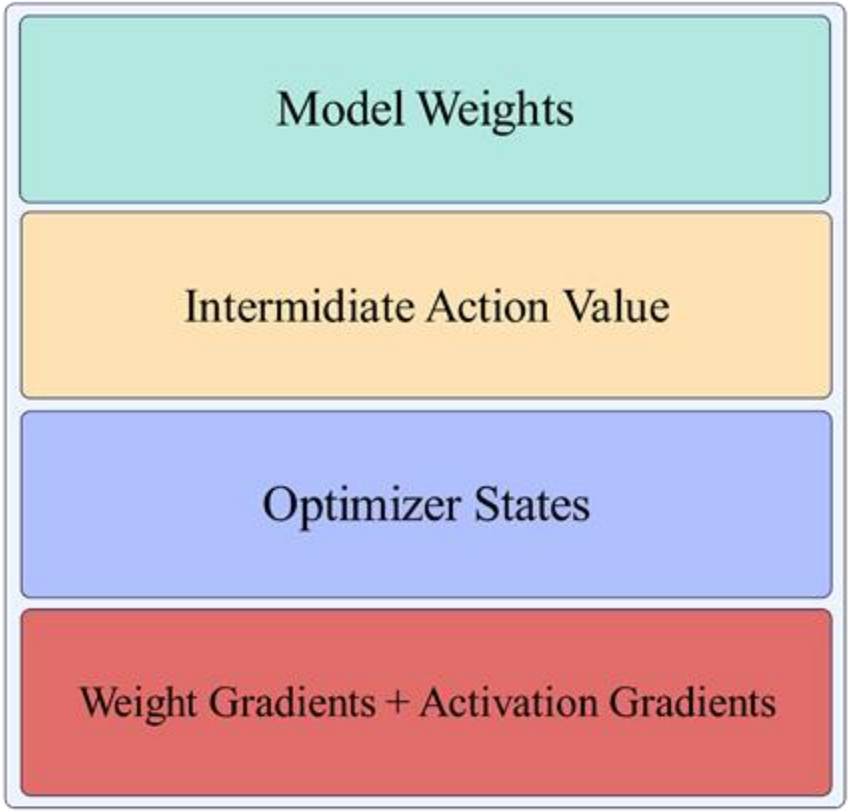
Connecting the Dots: Compute/Comm characteristic of LLMs

Key characteristics: compute, memory, communication

- calculate the number of parameters of an LLM?
- calculate the flops needed to train an LLM?
- calculate the memory needed to train an LLM?



Composition of Memory Usage (Training)



✅ 2*M (fp16)

❓

✅ 12*M (fp16-32 mixed precision)

✅ 2*M

❓

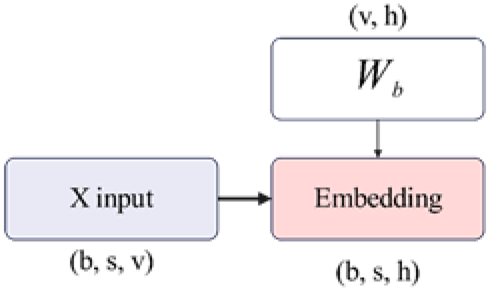
Llama2-7b Mix Precision(16bit-32bit)

X input

(b, s, v)

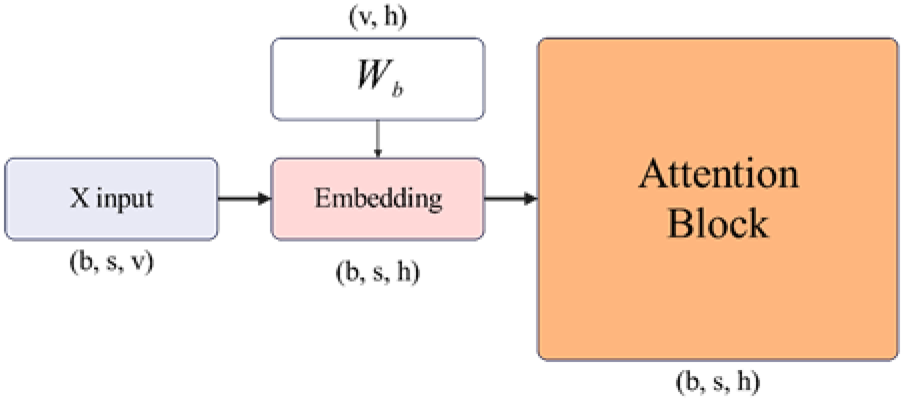
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension ($n \times d = h$)
v: Vocabulary Size

Llama2-7b Mix Precision(16bit-32bit)



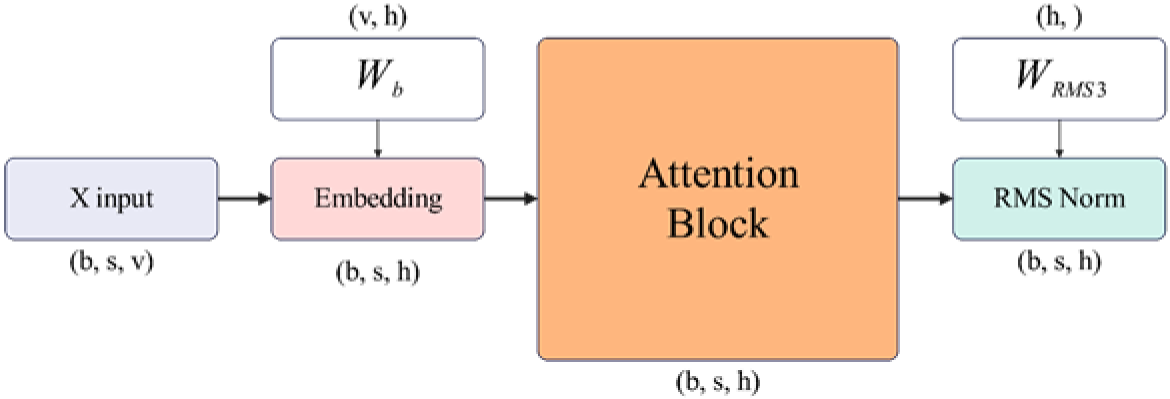
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension ($n \times d = h$)
v: Vocabulary Size

Llama2-7b Mix Precision(16bit-32bit)



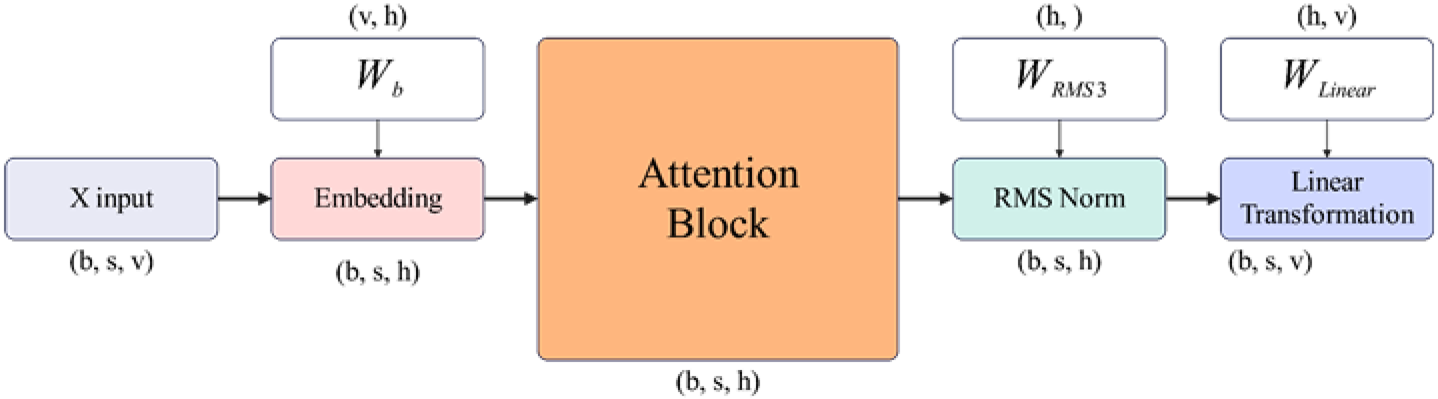
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension ($n \times d = h$)
v: Vocabulary Size

Llama2-7b Mix Precision(16bit-32bit)



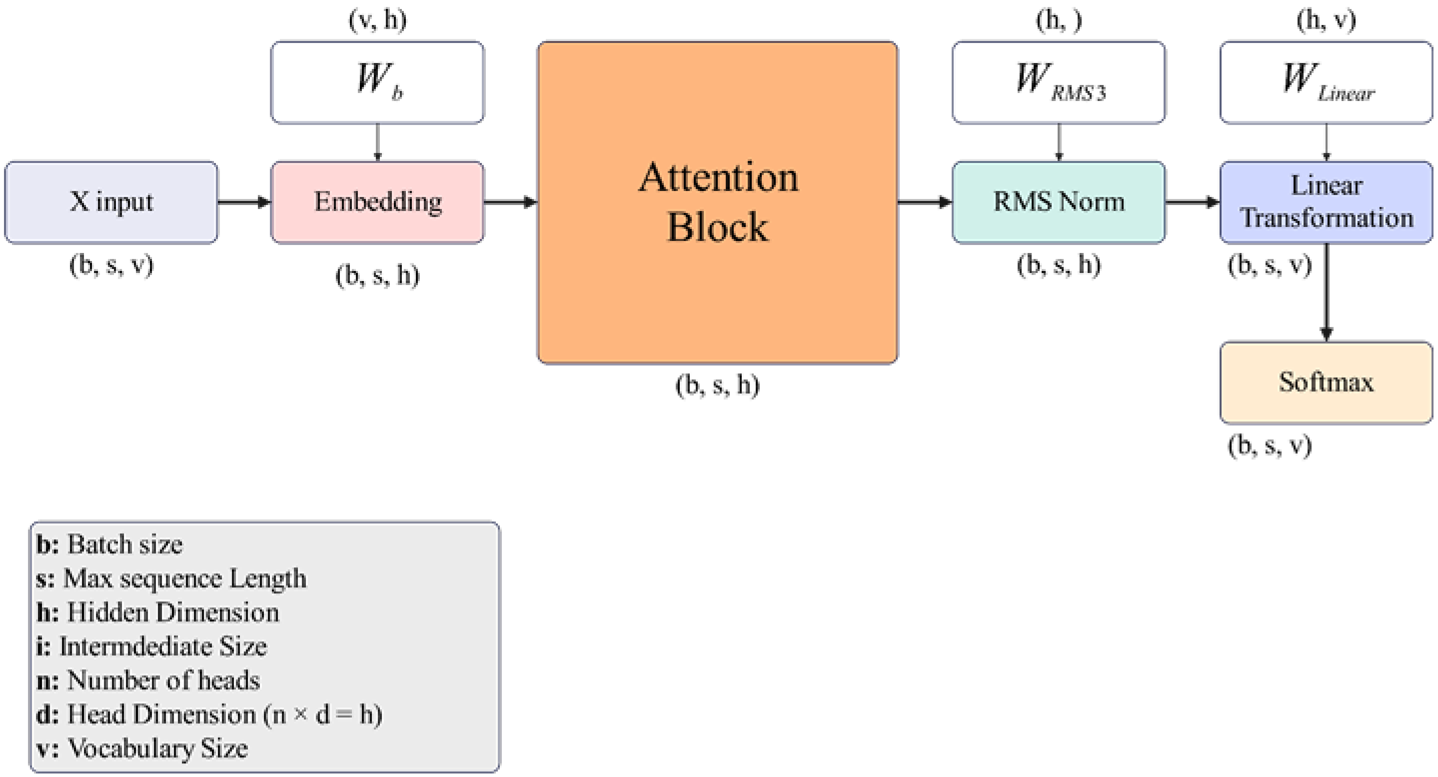
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension ($n \times d = h$)
v: Vocabulary Size

Llama2-7b Mix Precision(16bit-32bit)

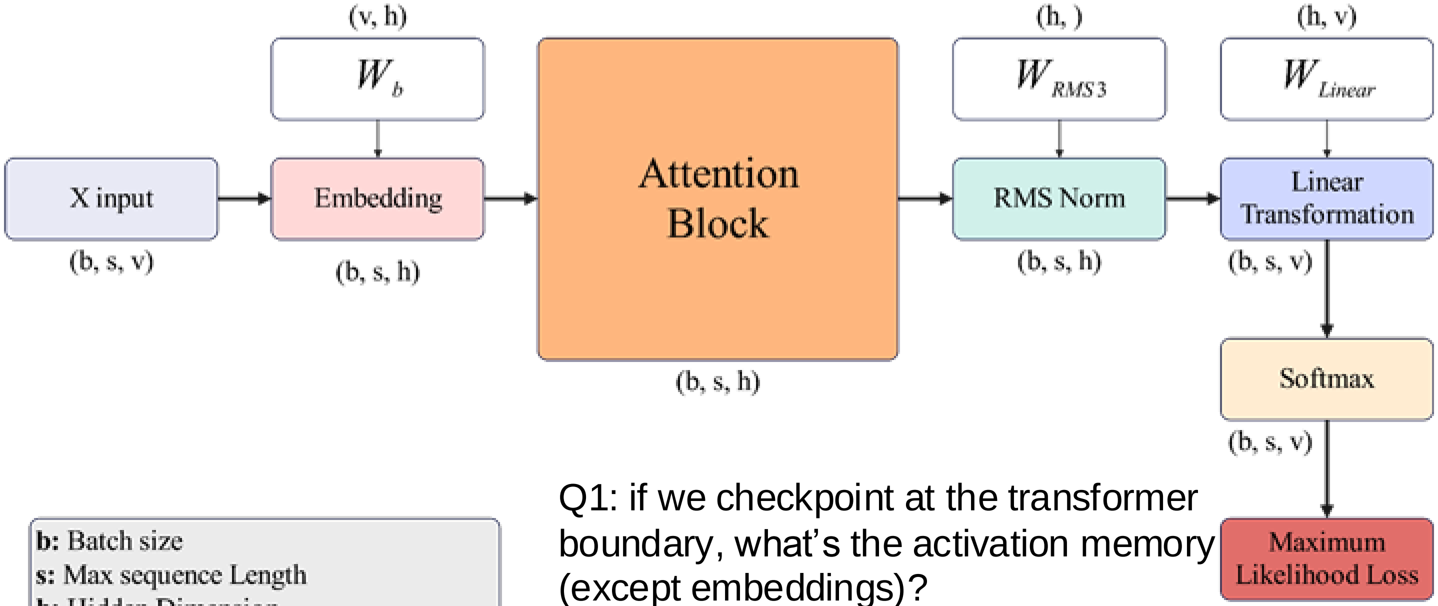


b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension ($n \times d = h$)
v: Vocabulary Size

Llama2-7b Mix Precision(16bit-32bit)



Llama2-7b Mix Precision(16bit-32bit)



b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension ($n \times d = h$)
v: Vocabulary Size

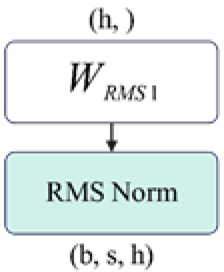
Q1: if we checkpoint at the transformer boundary, what's the activation memory (except embeddings)?

Q2: if we equally assign layers to devices, what's the P2P communication overhead?

Llama2-7b Attention Block (Self-Attention)

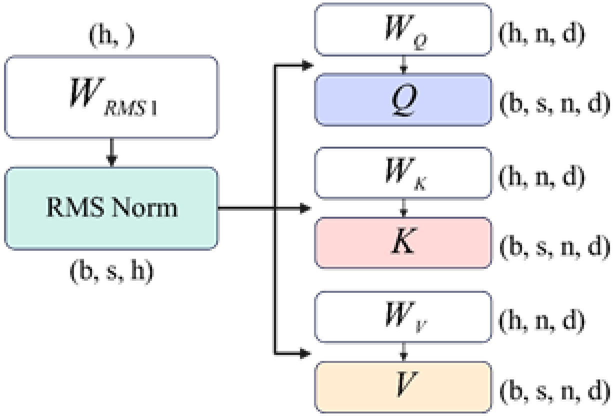
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension ($n \times d = h$)
v: Vocabulary Size

Llama2-7b Attention Block (Self-Attention)



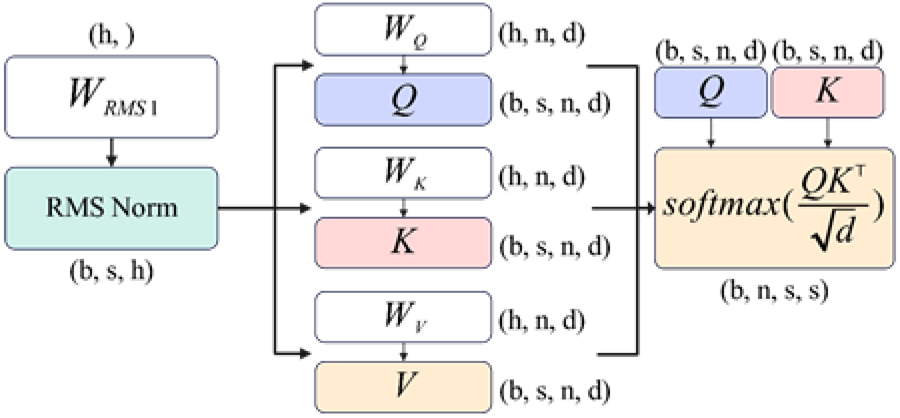
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension ($n \times d = h$)
v: Vocabulary Size

Llama2-7b Attention Block (Self-Attention)



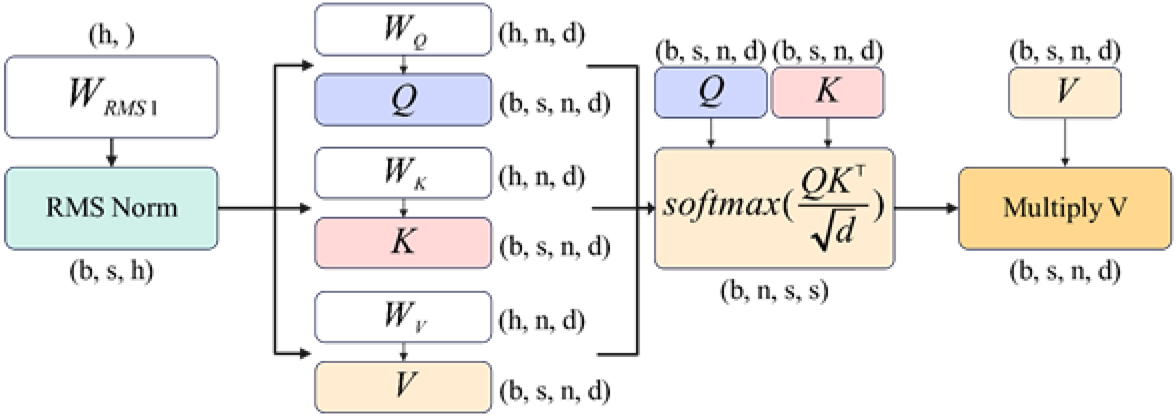
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension ($n \times d = h$)
v: Vocabulary Size

Llama2-7b Attention Block (Self-Attention)



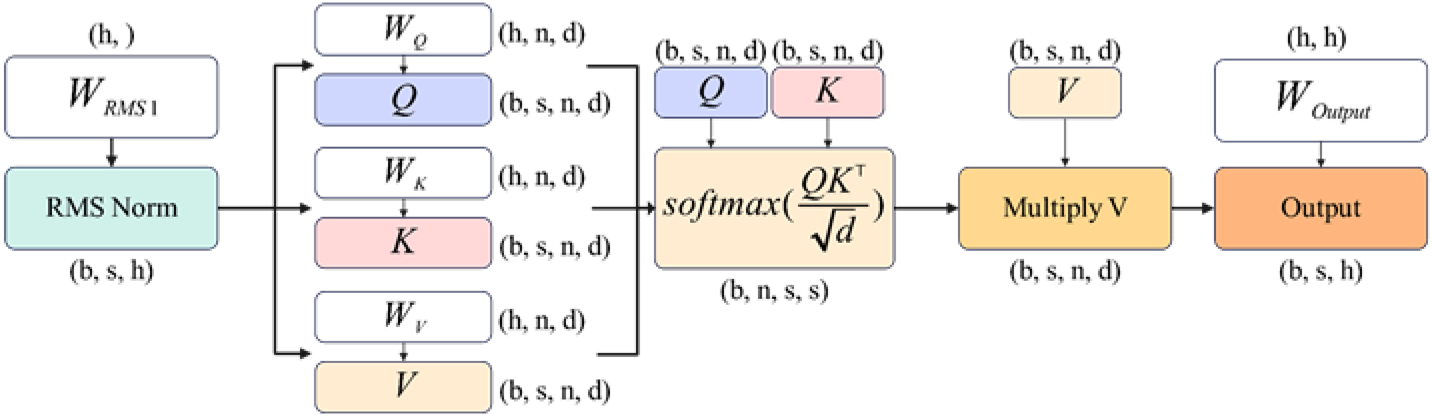
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension (n × d = h)
v: Vocabulary Size

Llama2-7b Attention Block (Self-Attention)



b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension ($n \times d = h$)
v: Vocabulary Size

Llama2-7b Attention Block (Self-Attention)

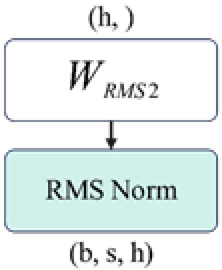


b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension (n × d = h)
v: Vocabulary Size

Llama2-7b Attention Block (FeedForward)

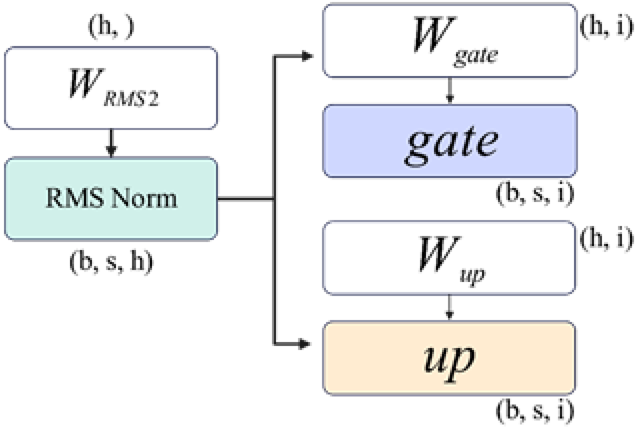
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension ($n \times d = h$)
v: Vocabulary Size

Llama2-7b Attention Block (FeedForward)



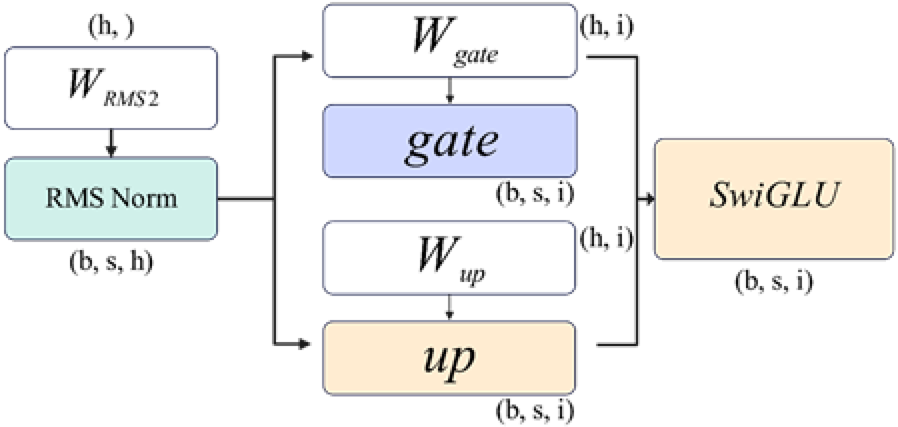
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension ($n \times d = h$)
v: Vocabulary Size

Llama2-7b Attention Block (FeedForward)



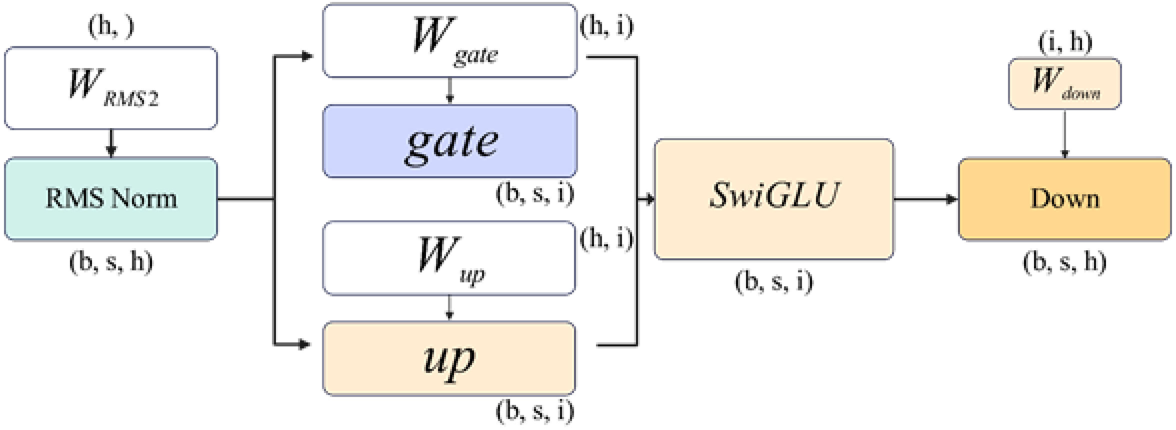
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension ($n \times d = h$)
v: Vocabulary Size

Llama2-7b Attention Block (FeedForward)



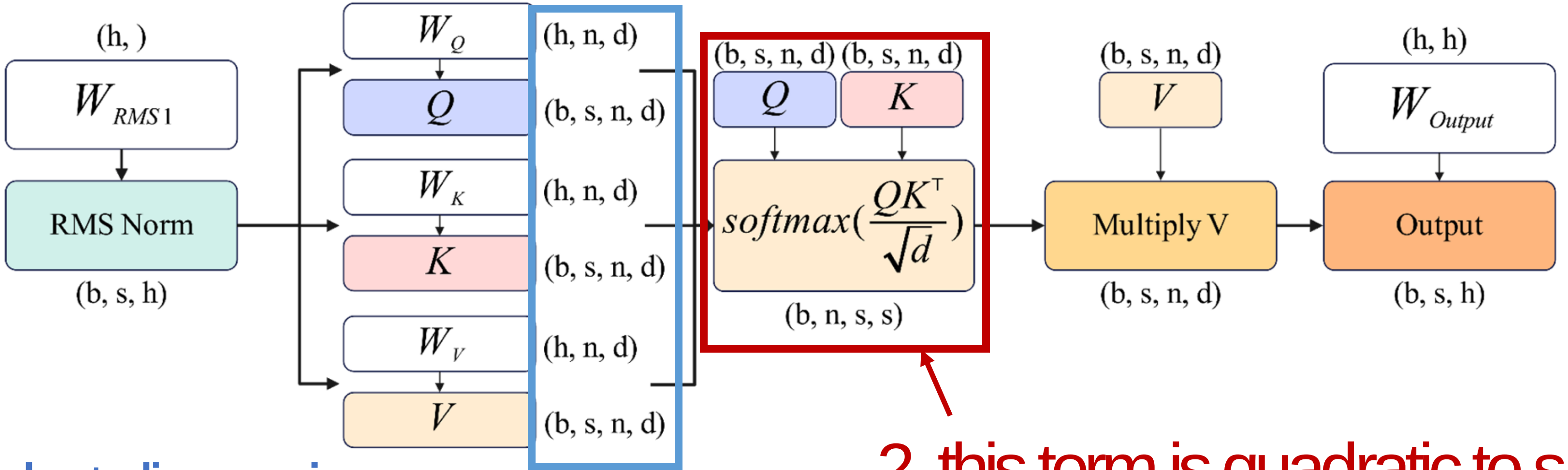
b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension ($n \times d = h$)
v: Vocabulary Size

Llama2-7b Attention Block (FeedForward)



b: Batch size
s: Max sequence Length
h: Hidden Dimension
i: Intermdediate Size
n: Number of heads
d: Head Dimension ($n \times d = h$)
v: Vocabulary Size

Scale Up: Potential Problems?

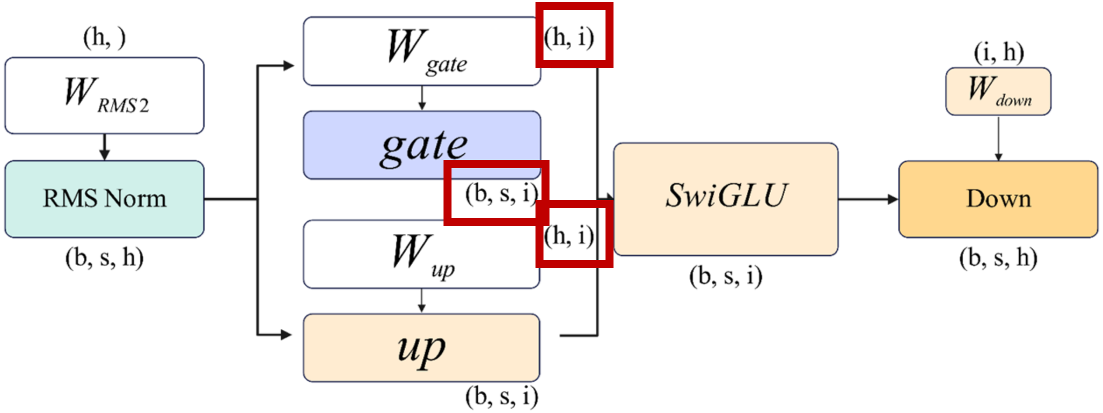


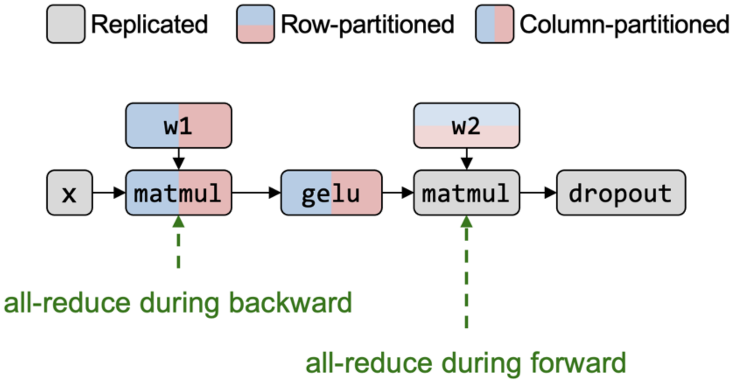
3. what dimension can we partition along?

2. this term is quadratic to s

1. all terms are at least linear with h

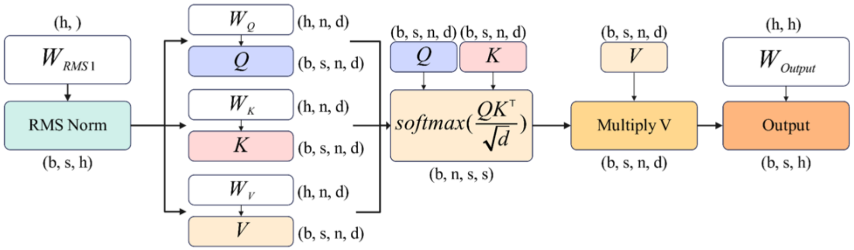
Scale Up: Potential Problems if h/i is large

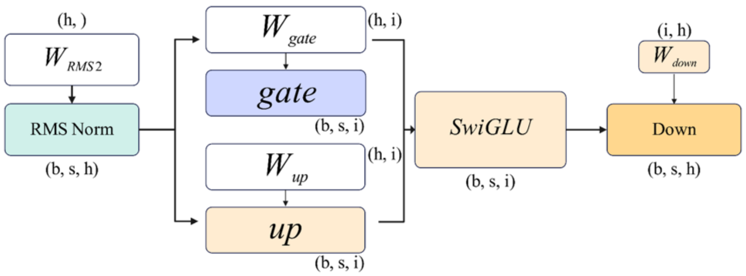




1. Recap: tensor parallel
2. can we partition along s if s is large?

Advanced Topics





- You already know megatron-style tensor parallelism
- s is large: can we partition along s?
  - Partition n in attention (the number of head dimension) and s in MLP
    - Deepspeed ulysses sequence parallelism
  - What communication is needed?
- What if #head << # GPUs or is not a multiple of 8
  - partition s in both attention and MLP
    - Ring Attention

Large Language Models

- Transformers, Attentions
- **Scaling Law**
  - MoE
- Connecting the dots: Training Optimizations
  - Flash attention
  - Long context, parallelism
- Serving and inference optimization
  - Continuous batching and Paged attention
  - Speculative decoding (Guest Lecture)
- Connecting the dots: Deepseek-v3
- Hot topics

Some Observations After the first part

- Compute is a function of: h, i, b
- #parameter is a function of: h, i
- Hence: compute correlates with #parameters
  - more parameters, more compute
  - more data, more compute (of course)
- Problem: we have limited compute ($)
- how should we allocate our limited resources:
  - Train models longer vs train bigger models?
  - Collect more data vs get more GPUs?
  - How to choose the exact h, i, etc.?

Motivation of Scaling Laws

- We want to know:
  - how large a model (detailed specs) should we train…
  - How many data should we use…
  - To achieve a given performance…
  - Subject to a compute budget ($)?

How do we do that in traditional ML: data scaling law

Input: $x_1 \ \dots x_n \ \sim N(\mu, \sigma^2)$

Task: estimate the average as $\hat{\mu} = \frac{\sum_i x_i}{n}$

What's the error? By standard arguments..

$$E[(\hat{\mu} - \mu)^2] = \frac{\sigma^2}{n}$$

This is a scaling law!!

$$\log(Error) = -\log n + 2 \log \sigma$$

More generally, any polynomial rate $1/n^{\alpha}$ is a scaling law

- Can we do this for transformers LLMs?
- **Unfortunately NO**

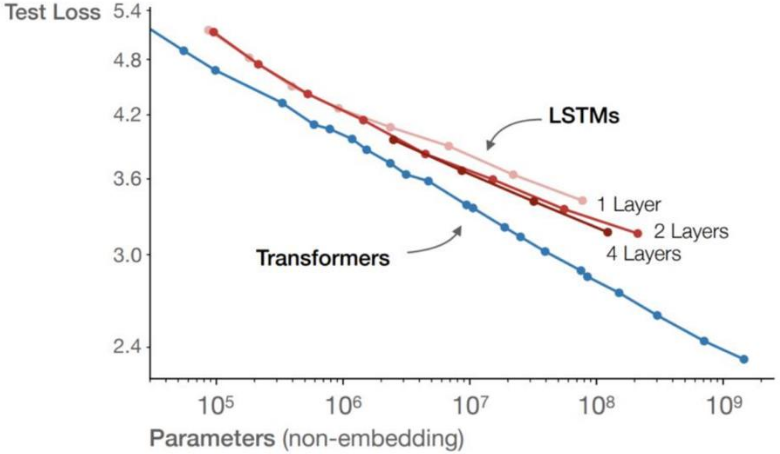
Think in this way

Mathematics vs.

Physics

Transformers vs LSTMs

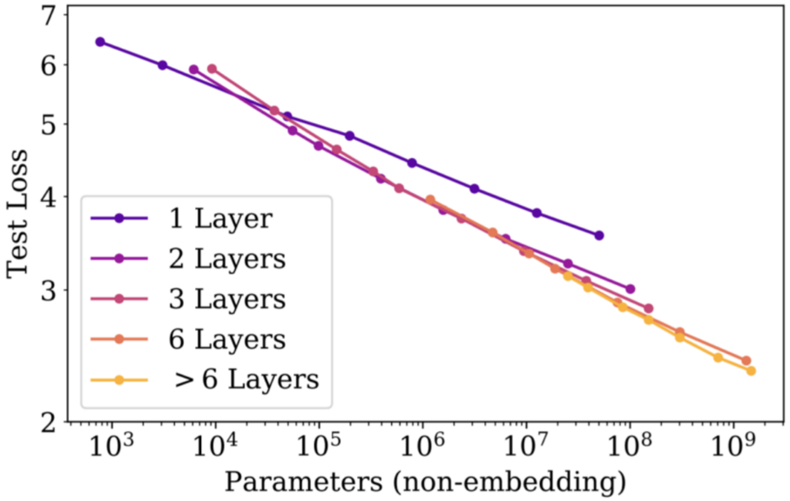
- Q: Are transformers better than LSTMs?
  - Brute force way: spend tens of millions to train a LSTM GPT-3
- Scaling law way:



[Kaplan+ 2021]

Number of Layers

- Does depth or width make a huge difference?
  - 1 vs 2 layers makes a huge difference.
  - More layers have diminishing returns below 107 params



The Scaling law way: Physics Way

- Approach:
  - Train a few smaller models
  - Establish a scaling law (LSTM vs. transformers)
  - Select optimal hyperparams based on the scaling law prediction.

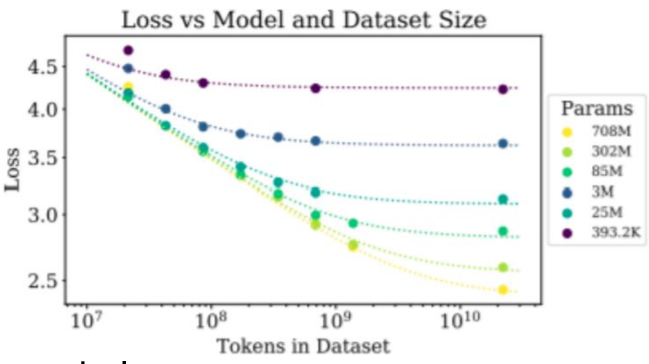
- Rationale
  - The effect of hyperparameters on big LMs can be predicted before training!
    - Optimizer choice
    - Model Depth
    - Arechitecture choice

Back to our problem:

- how large a model should we train…
- How many data should we use…
- To achieve a given performance…
- Subject to a compute budget?

- Approach: estimate a law between model size data joint scaling

Model size data joint scaling



- Do we need more data or bigger model
  - Clearly, lots of data is wasted on small models
- Joint data-model scaling laws describe how the two relate

From Rosenfeld+ 2020,

$$Error = n^{-\alpha} + m^{-\beta} + C$$

From Kaplan+ 2021

$$Error = [m^{-\alpha} + n^{-1}]^{\beta}$$

Provides surprisingly good fits to model-data joint error.

Compute Trade-offs

- Q: what about other resources? Compute vs. performance?

- For a fixed compute budget…
  - Big models that's undertrained vs small model that's well trained?
  - Solving the following optimization?

$$N_{opt}(C), D_{opt}(C) = \underset{N, D \text{ s.t. FLOPs}(N,D)=C}{\operatorname{argmin}} L(N, D).$$

Approach: empirical scaling law

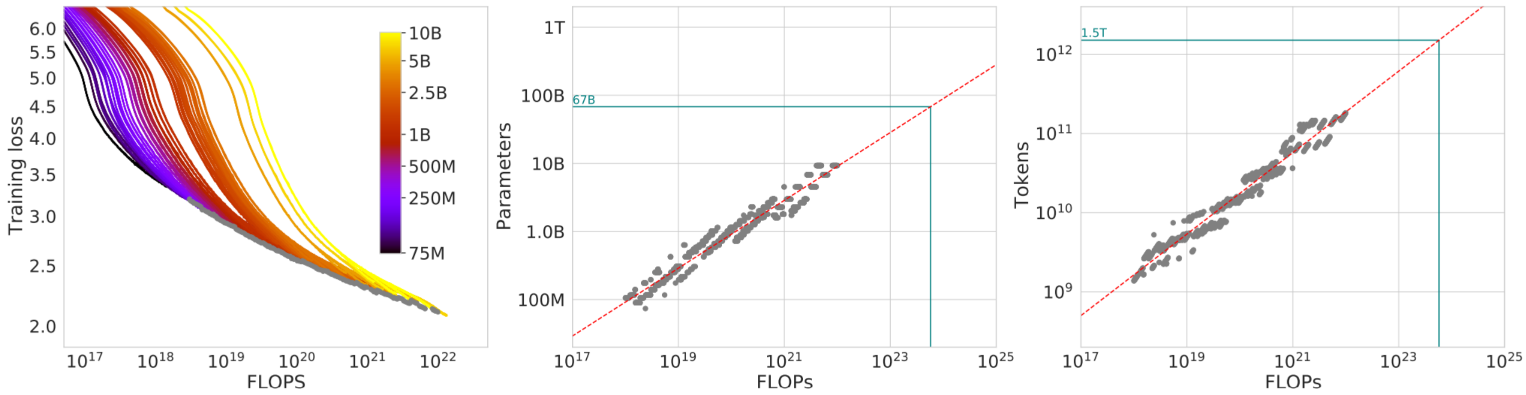


Figure 2 | **Training curve envelope.** On the **left** we show all of our different runs. We launched a range of model sizes going from 70M to 10B, each for four different cosine cycle lengths. From these curves, we extracted the envelope of minimal loss per FLOP, and we used these points to estimate the optimal model size (**center**) for a given compute budget and the optimal number of training tokens (**right**). In green, we show projections of optimal model size and training token count based on the number of FLOPs used to train *Gopher* (5.76×10^{23}).

Today's SoTA Law

$$L(N, D) = \frac{406.4}{N^{0.34}} + \frac{410.7}{D^{0.29}} + 1.69$$

Summary

- Scaling law: the physics of ML
- Scaling law marks a new era of ML research:
  - Rigorous theoretical analysis -> empirical laws
  - Exploration of different model architectures -> Scaling transformers
  - Due to scaling law: ML systems become essential

PA3: Hints

You already know:

- How to estimate the number of parameters of an LLM?
- How to estimate the flops needed to train an LLM?
- How to estimate the memory needed to train a transformer?

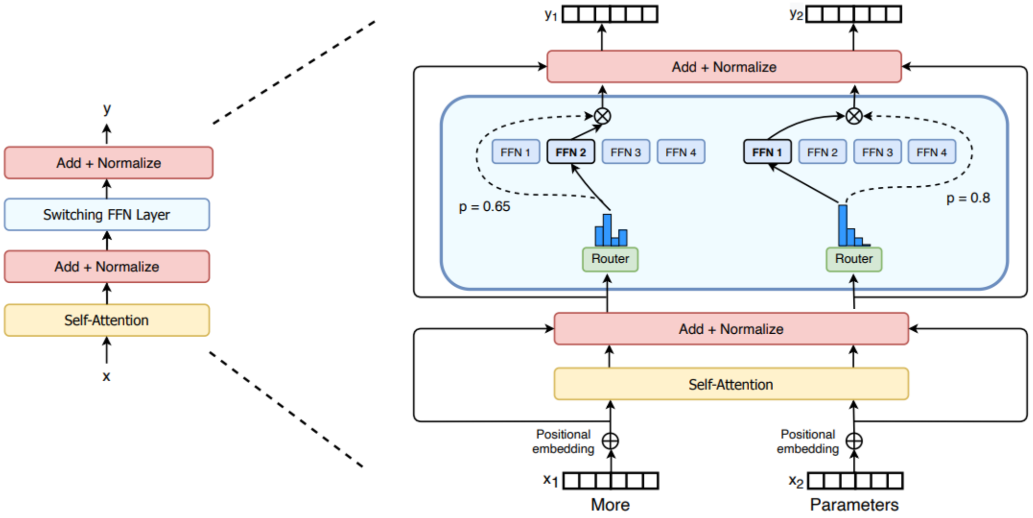
- We will give you a scaling law and compute budget
  - Task: design your optimal LLM

Large Language Models

- Transformers, Attentions
- Scaling Law
  - **MoE**
- Connecting the dots: Training Optimizations
  - Flash attention
- Serving and inference optimization
  - Continuous batching and Paged attention
  - Speculative decoding (Guest Lecture)
- Connecting the dots: Deepseek-v3
- Hot topics

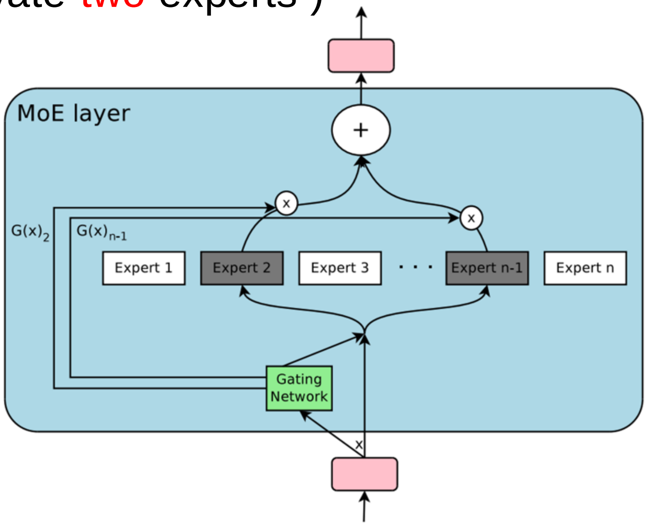
MoE-based LLMs

- Superficially: mixture of experts
- **Key idea**: make each expert focus on predicting the right answer for a subset of ca:



A Closer Look at Mixture-of-Experts

A typical MoE layer (assume single instance and activate two experts)



Gating: $G = \mathrm{Softmax}(W_G X)$

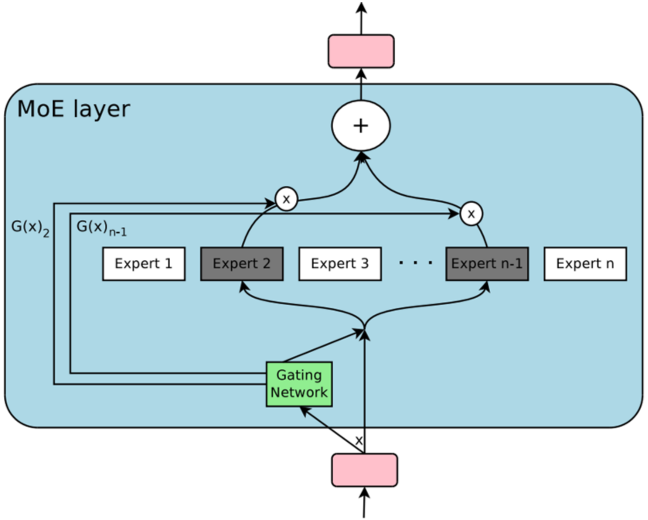
Expert indices: $I = \{i_0, i_1\} = \mathrm{Top}K(G, k = 2)$

Output weight: $s_0 = \frac{G_{i_0}}{(G_{i_0} + G_{i_1})}, s_1 = \frac{G_{i_1}}{(G_{i_0} + G_{i_1})}$

Output: $Y = s_0 \mathrm{FFN}_{i_0}(X) + s_1 \mathrm{FFN}_{i_1}(X)$

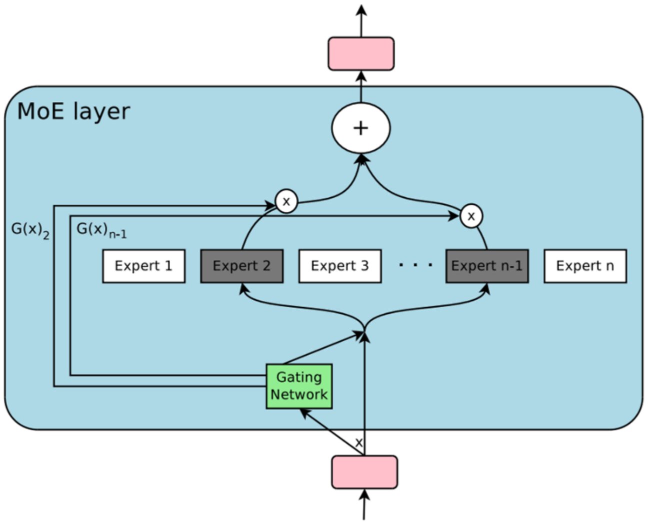
MoE from the Scaling Law Perspective

- Parameters
  - MLP params dominate LLMs
  - MLP params x N/2
  - Increase drastically
- Memory
  - parameter related x N/2
  - activation?
- Compute
  - Only increase mildly



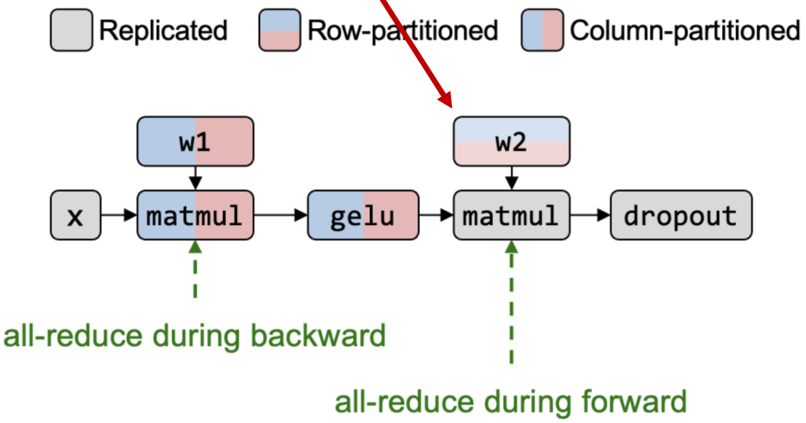
MoE from the Scaling Law Perspective

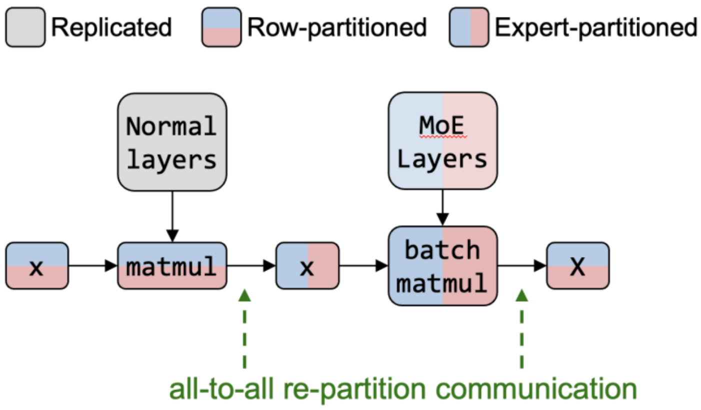
- Essentially, MoE is a more compute-efficient Model
- I.e., MoE has a better scaling law



Parallelization of MoE

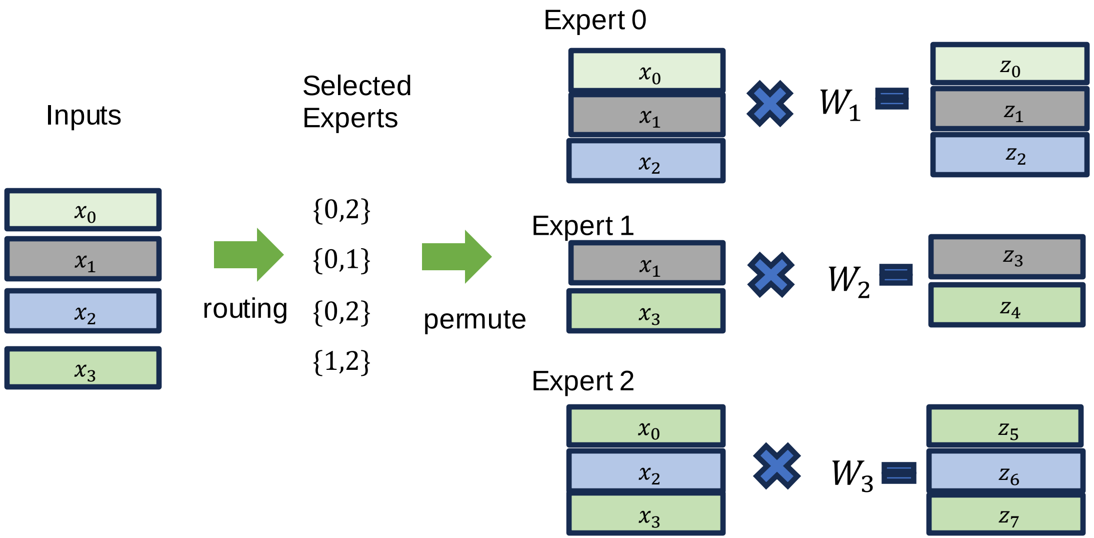
What if we still do TP in face of MoE?





Potential problems of MoE?

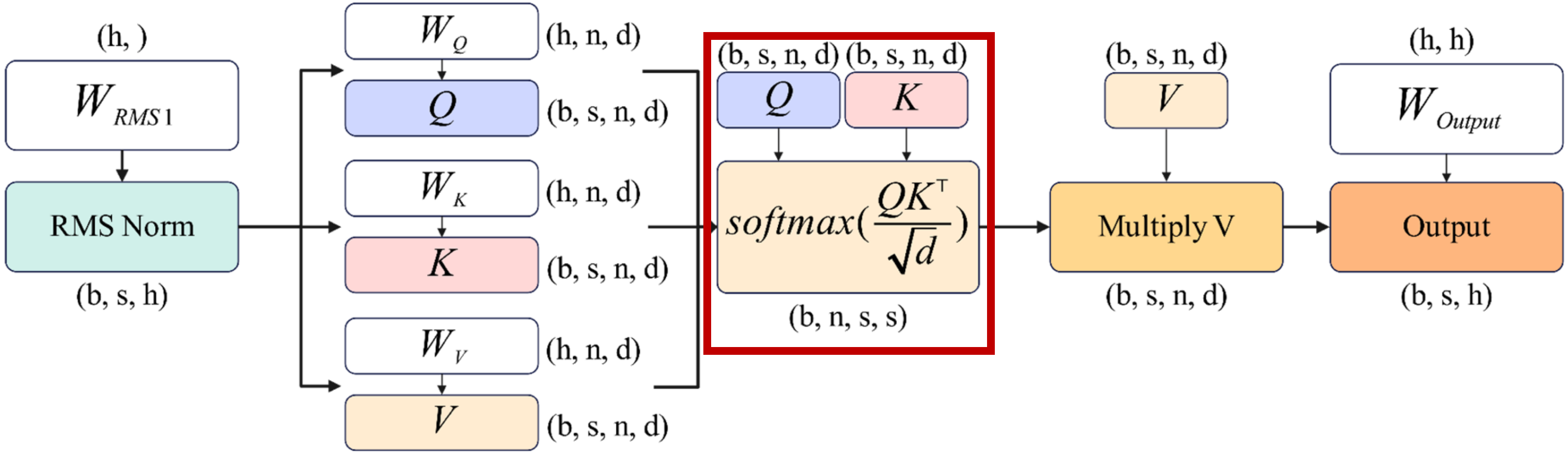
Potential Problems of MoE?



Color tracks instance indices

Gale et.al MegaBlocks: Efficient Sparse Training with Mixture-of-Experts

The rest of bottleneck of LLMs



✌️Rule of thumb: in many computer systems and algorithms, anything more complex than quadratic is less likely to be adapted at large scale

Large Language Models

- Transformers, Attentions
- Scaling Law
  - MoE
- Connecting the dots: Training Optimizations
  - Flash attention ← will fix the s^2 to some extend next week
- Serving and inference optimization
  - Continuous batching and Paged attention
  - Speculative decoding (Guest Lecture)
- Connecting the dots: Deepseek-v3
- Hot topics

Reality Check: LLMs are Slow and Expensive to Serve

- **At least ten** A100-40GB GPUs to serve 175B GPT-3 in half precision
- Generating 256 tokens takes **~20 seconds**

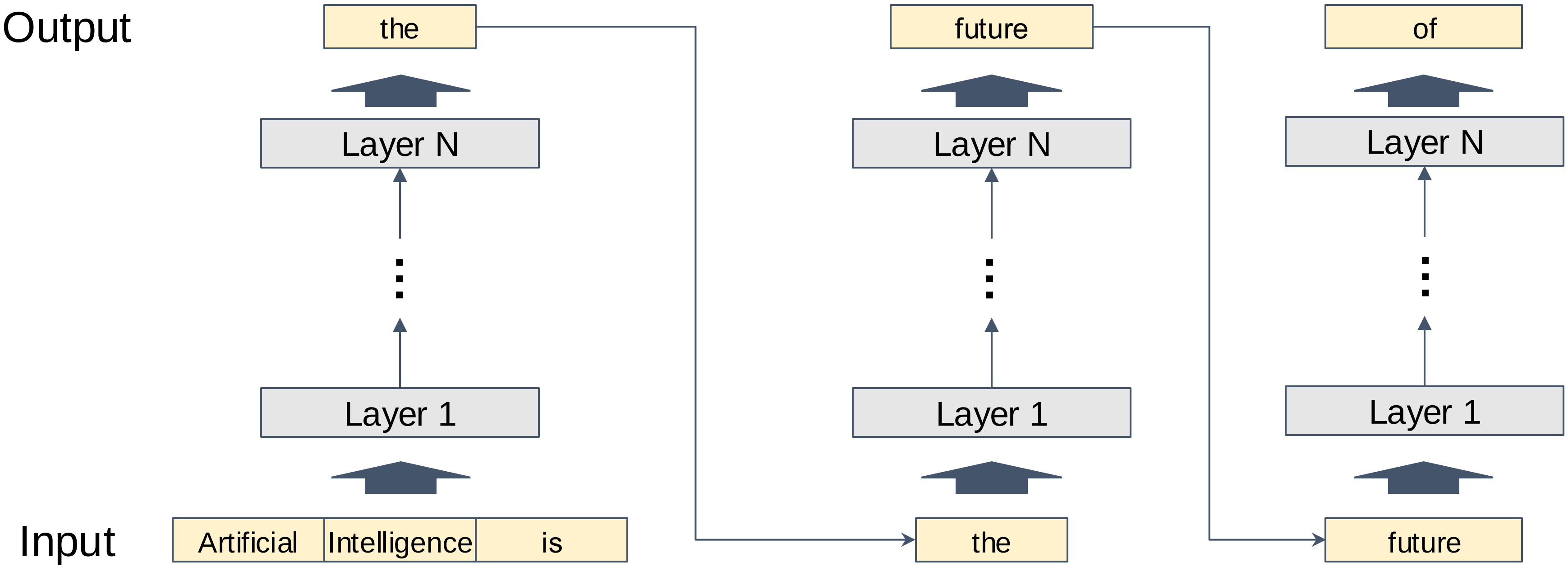
Next Token Prediction

Probability("San Diego has very nice weather")
= P("San Diego") P("has" | "San Diego")P("very" | "San Diego has")P("city" | . . .). . .P("weather" | . . .)

$$\text{Max}\, Prob(x_{1:T}) = \boxed{\prod_{t=1}^{T} P(x_{t+1} | x_{1 \ldots t})}$$

This is model we got – capable of "predicting the next token".

Inference process of LLMs



Repeat until the sequence

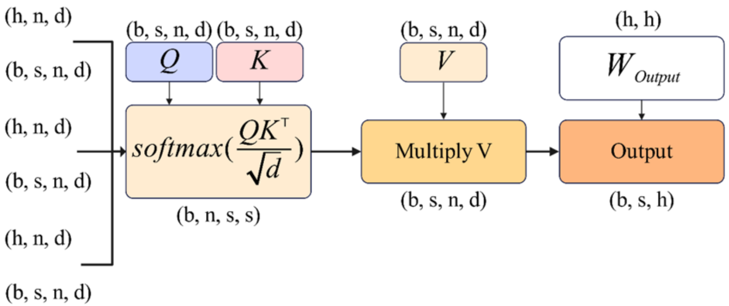
- Reaches its pre-defined maximum length (e.g., 2048 tokens)
- Generates certain tokens (e.g., “<|*end of sequence*|>”)

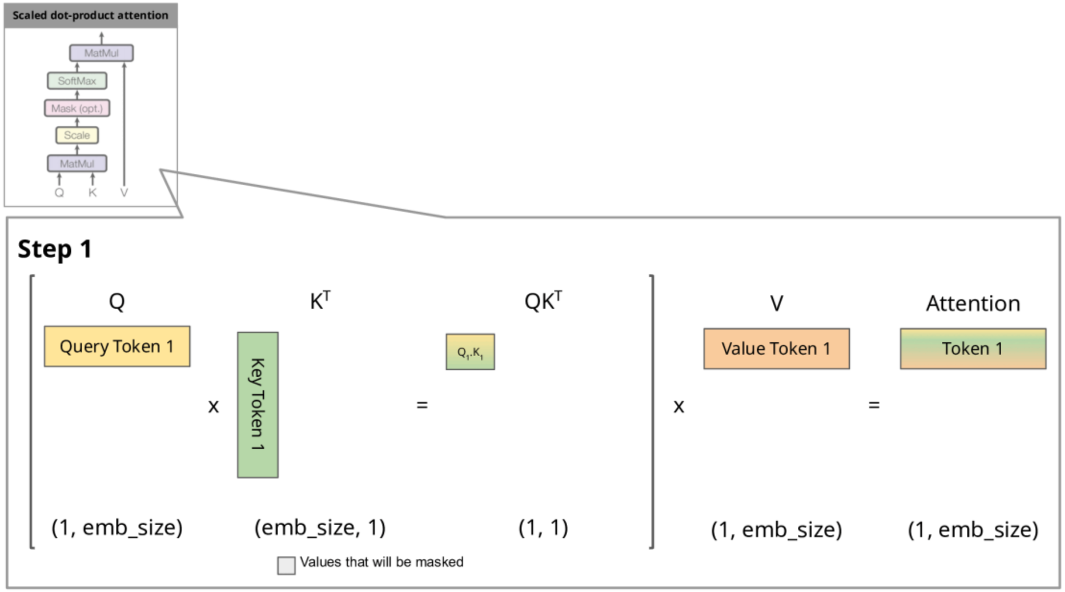
Generative LLM Inference: Autoregressive Decoding

- Pre-filling phase (0-th iteration):
  - Process *all* input tokens at once
- Decoding phase (all other iterations):
  - Process a *single* token generated from previous iteration

- Key-value cache:
  - Save attention keys and values for the following iterations to avoid recomputation
  - what is KV cache essentially?

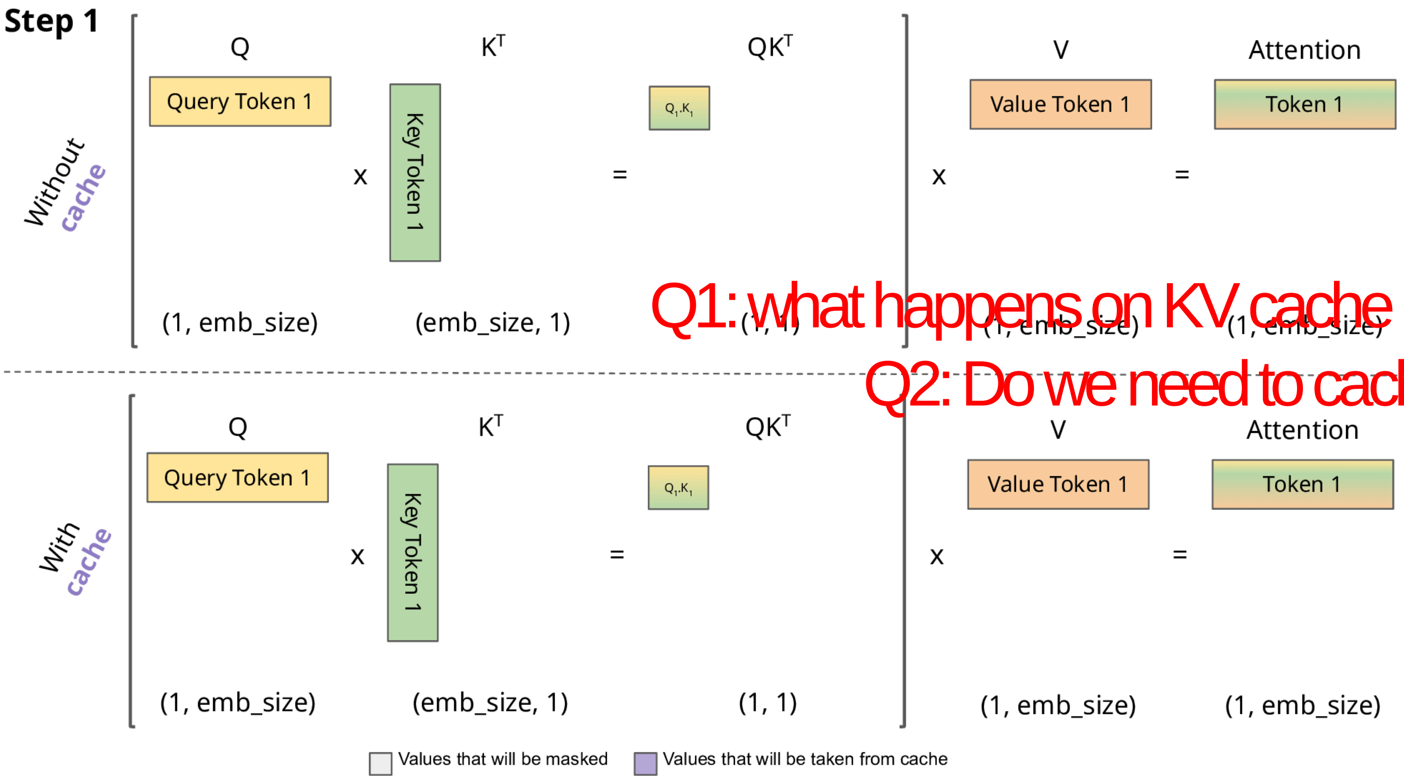
w/ KV Cache vs. w/o KV Cache





Zoom-in! (simplified without Scale and Softmax)

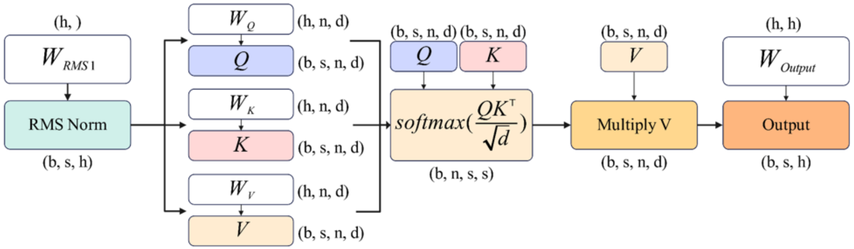
w/ KV Cache vs. w/o KV Cache

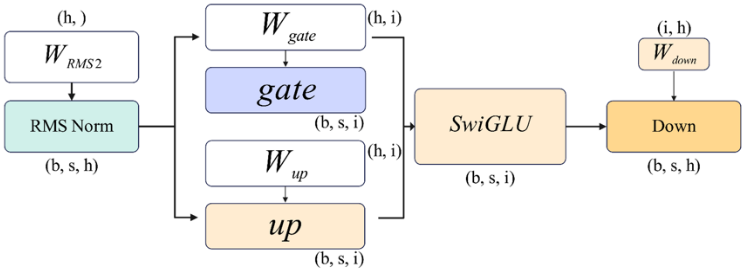


Q1: what happens on KV cache in prefill phase?

Q2: Do we need to cache Q?

Potential Bottleneck of LLM Inference?





- Compute:
  - Prefill: largely same with training
  - Decode: s = 1
- Memory
  - New: KV cache
- Communication
  - mostly same with training

Q? how about batch size b?

Serving vs. Inference

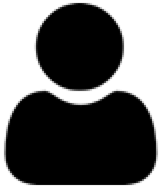
large b

Serving: many requests, online traffic, emphasize cost-per-query.

s.t. some mild latency constraints

emphasize **throughput**

b=1

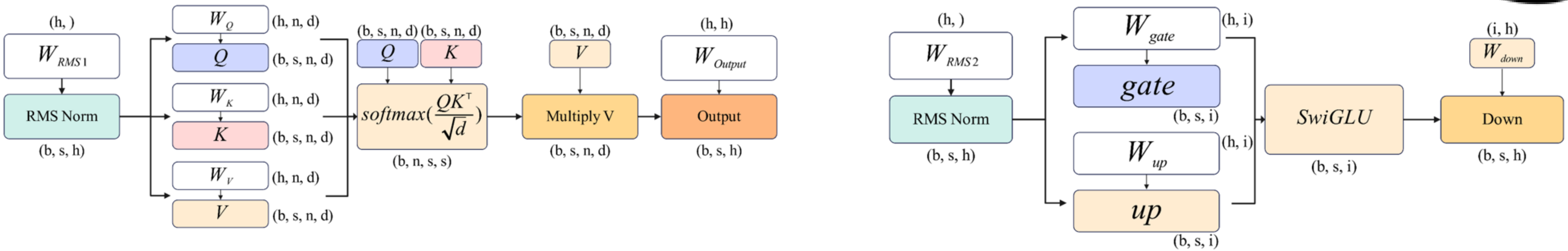


Inference: fewer request, low or offline traffic,

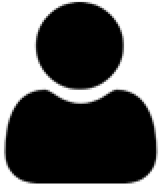
emphasize **latency**

large b

Potential Bottleneck of LLM Inference in Serving

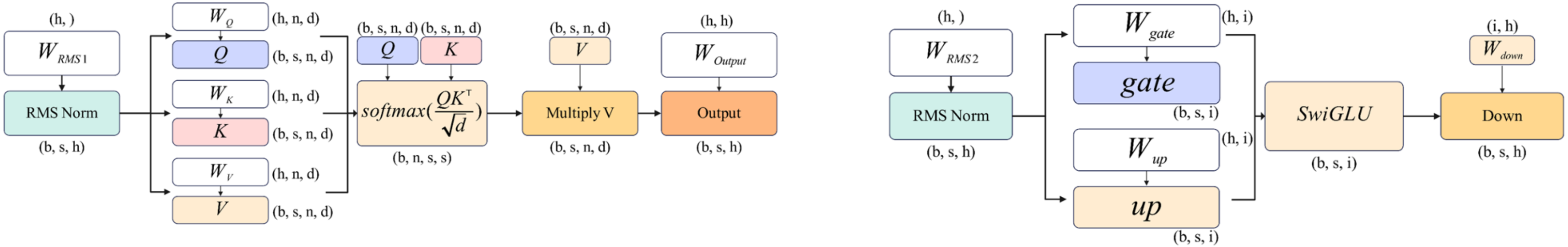


- Compute:
  - Prefill:
    - Different prompts have different length: how to batch?
  - Decode
    - Different prompts have different, unknown #generated tokens
    - s = 1, b is large
- Memory
  - New: KV cache
  - b is large -> KV is linear with b -> will KVs be large?
- Communication
  - mostly same with training



b=1

Potential Bottleneck of LLM Inference in Serving

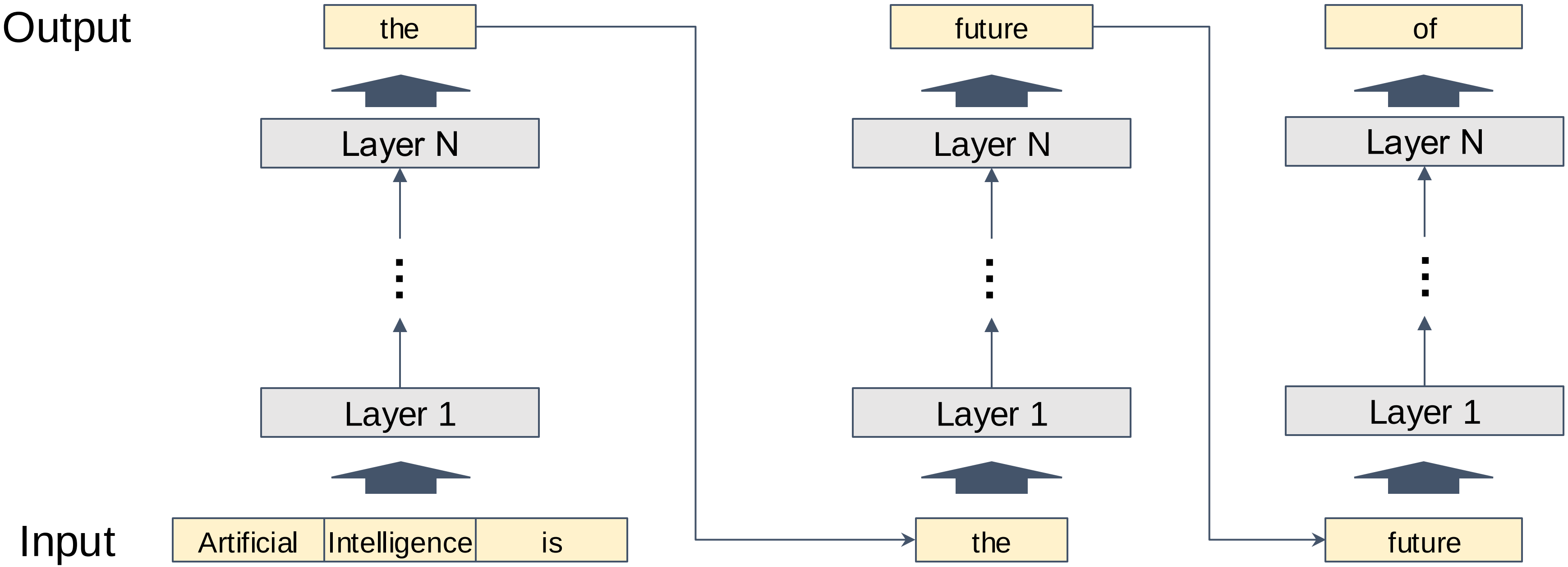


- Compute:
  - Prefill:
    - ~~Different prompts have different length: how to batch?~~
  - Decode
    - Different prompts have different, unknown #generated tokens
    - s = 1, b=1
- Memory
  - New: KV cache
  - ~~b =1 -> KV is linear with b -> will KVs be large?~~
- Communication
  - mostly same with training

Problems of bs = 1

max AI = #ops / #bytes

Recap: Inference process of LLMs

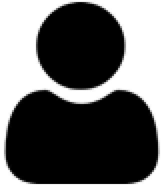


Repeat until the sequence

- Reaches its pre-defined maximum length (e.g., 2048 tokens)
- Generates certain tokens (e.g., “<|*end of sequence*|>”)

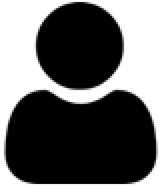
Problem of bs = 1

b=1



Latency = step latency * # steps

b=1



Large Language Models

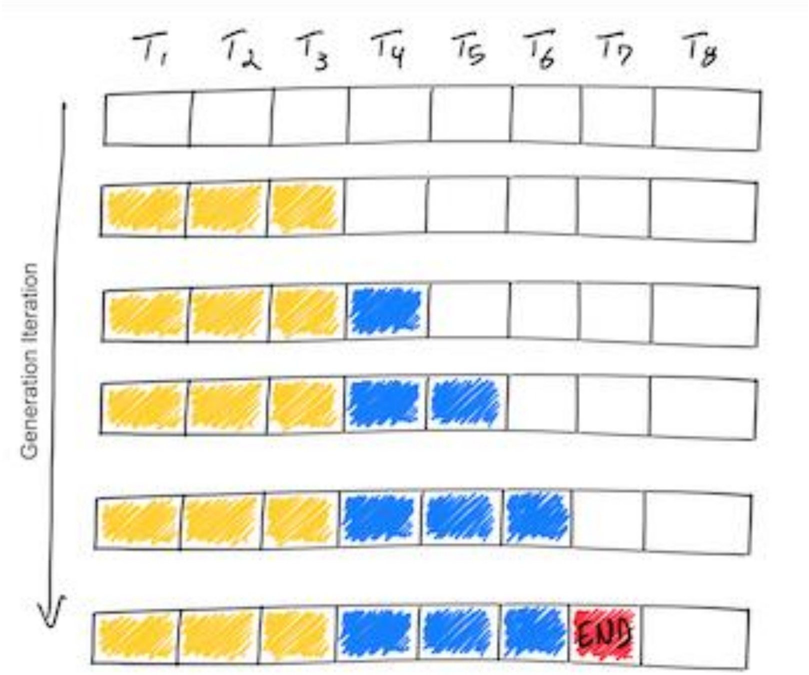
- Transformers, Attentions
- Scaling Law
  - MoE
- Connecting the dots: Training Optimizations
  - Flash attention ← come back to this later next week
- Serving and inference optimization
  - Continuous batching and Paged attention
  - **Speculative decoding (Guest Lecture)**
- Connecting the dots: Deepseek-v3
- Hot topics

large b

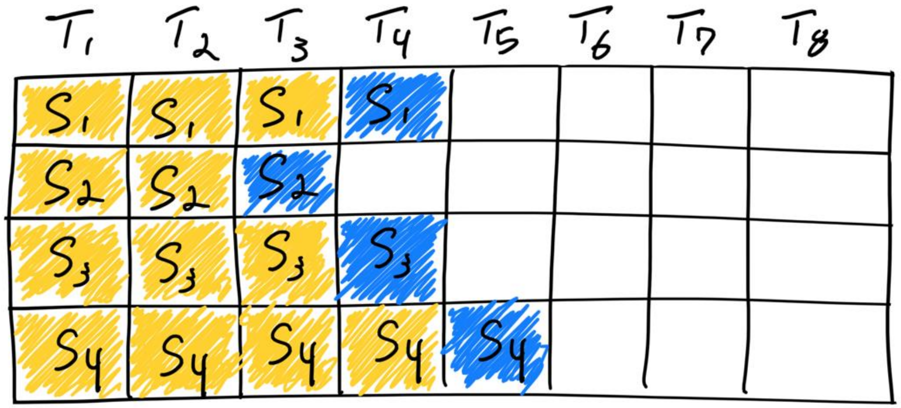
Large Language Models

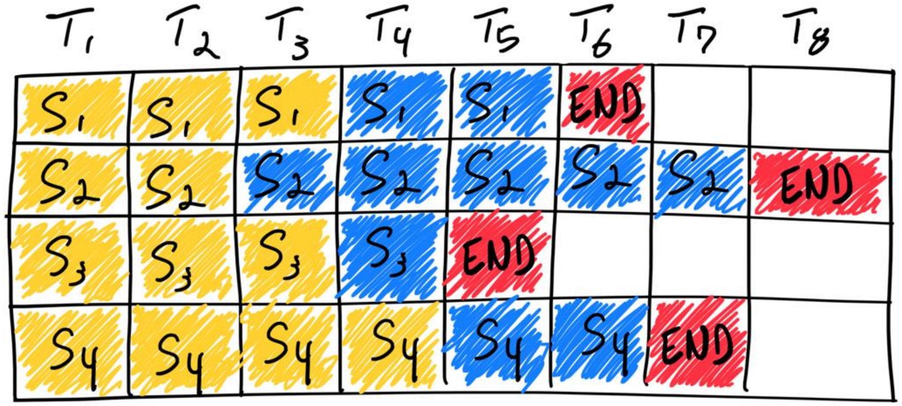
- Transformers, Attentions
- Scaling Law
  - MoE
- Connecting the dots: Training Optimizations
  - Flash attention ← come back to this later next week
- Serving and inference optimization
  - **Continuous batching and Paged attention**
  - Speculative decoding (Guest Lecture)
- Connecting the dots: Deepseek-v3
- Hot topics

LLM Decoding Timeline



Batching Requests to Improve GPU Performance

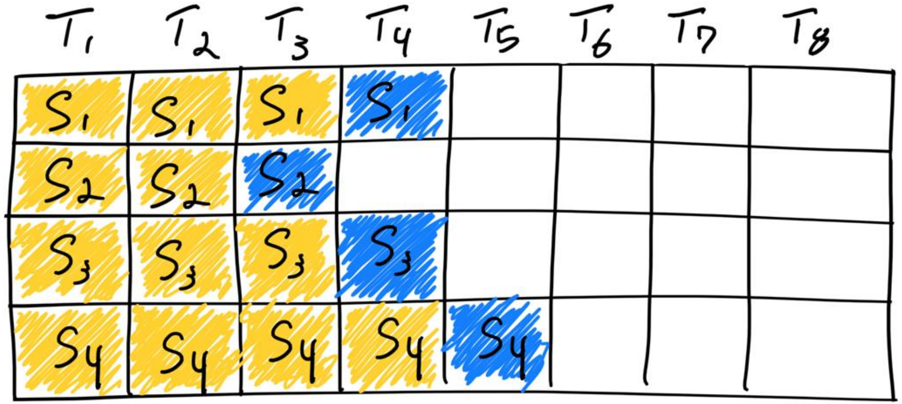


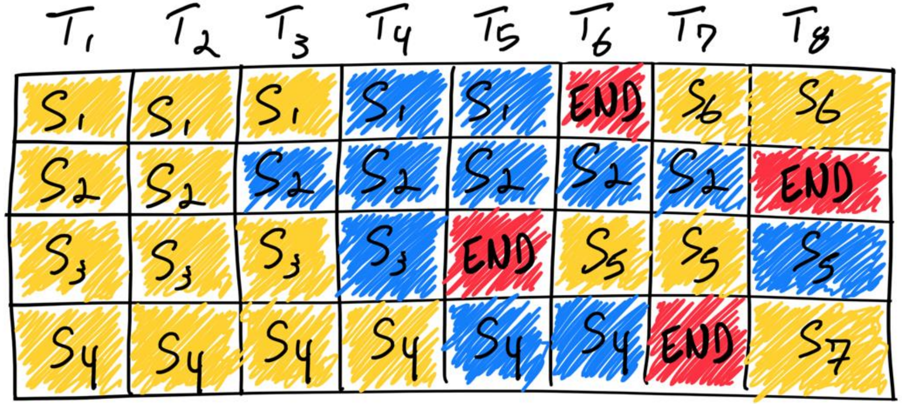


Issues with static batching:

- Requests may complete at different iterations
- Idle GPU cycles
- New requests cannot start immediately

Continuous Batching





Benefits:

- Higher GPU utilization
- New requests can start immediately

Continuous Batching Step-by-Step

- Receives two new requests R1 and R2

R1: optimizing ML systems

R2: LLM serving is

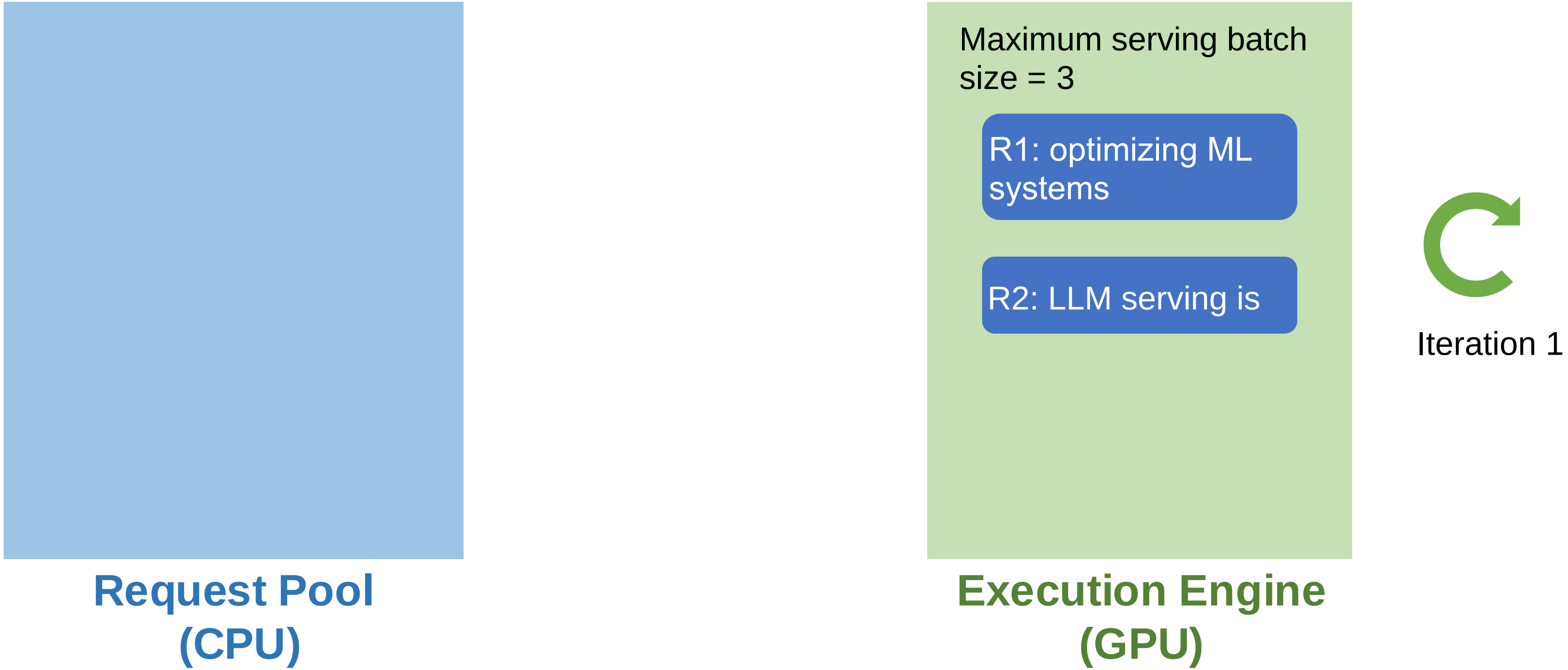
Maximum serving batch size = 3

Request Pool (CPU)

Execution Engine (GPU)

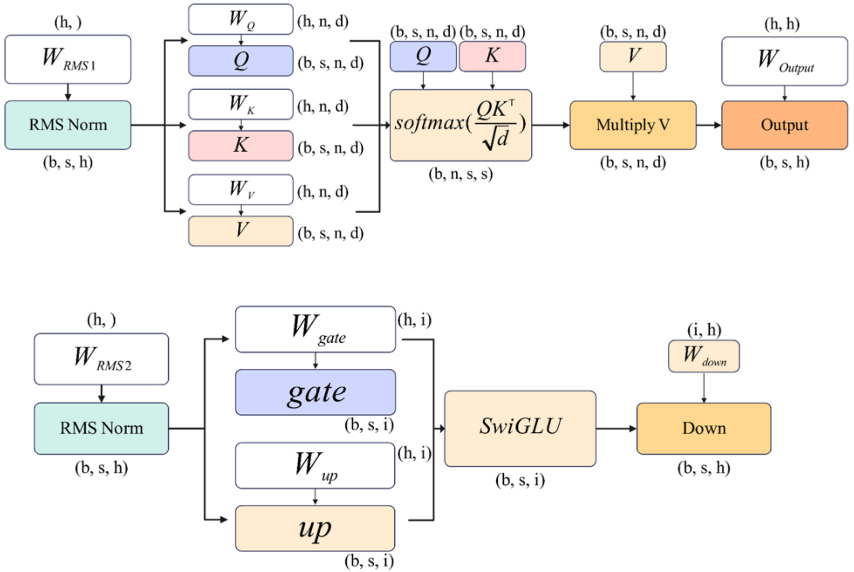
Continuous Batching Step-by-Step

- Iteration 1: decode R1 and R2



Continuous Batching Step-by-Step

- Iteration 1: decode R1 and R2



Q: How to batch these?

Maximum serving batch size = 3

R1: optimizing ML systems

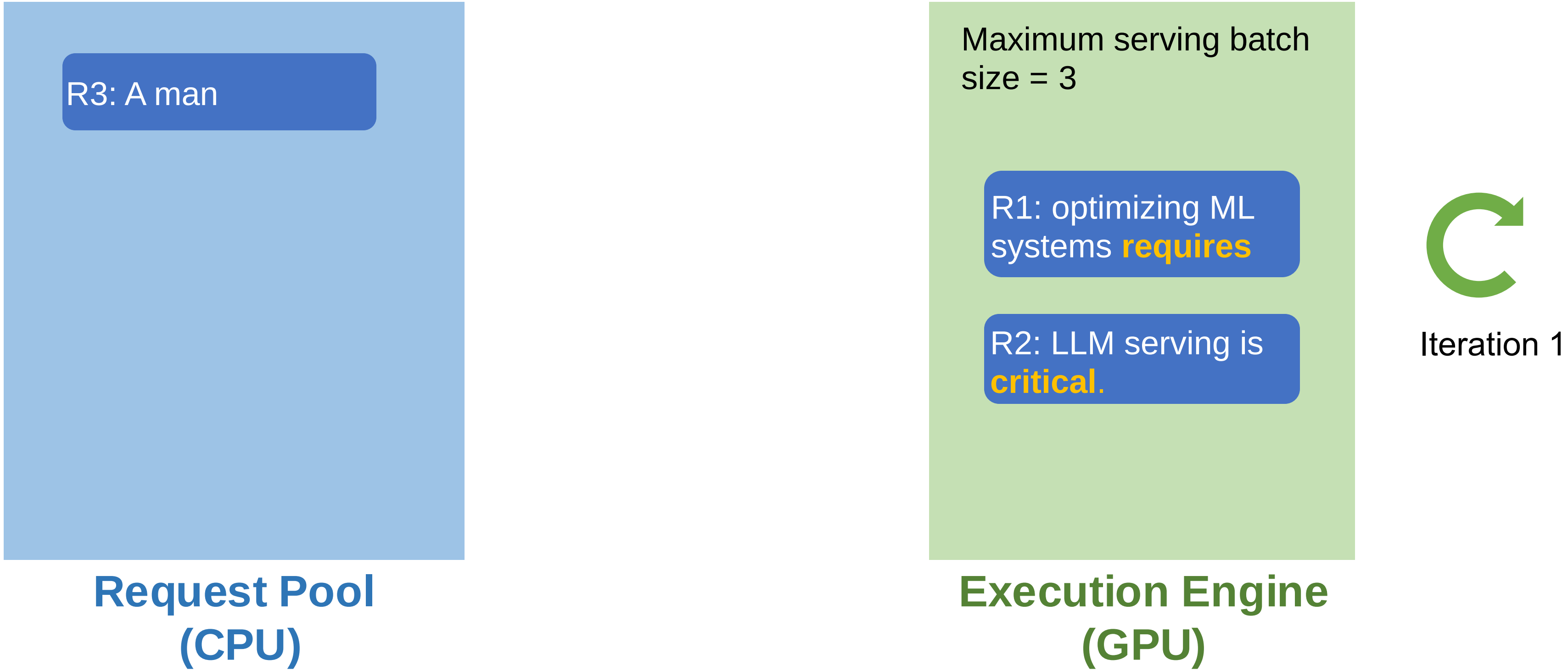
R2: LLM serving is

Iteration 1

Execution Engine (GPU)

Continuous Batching Step-by-Step

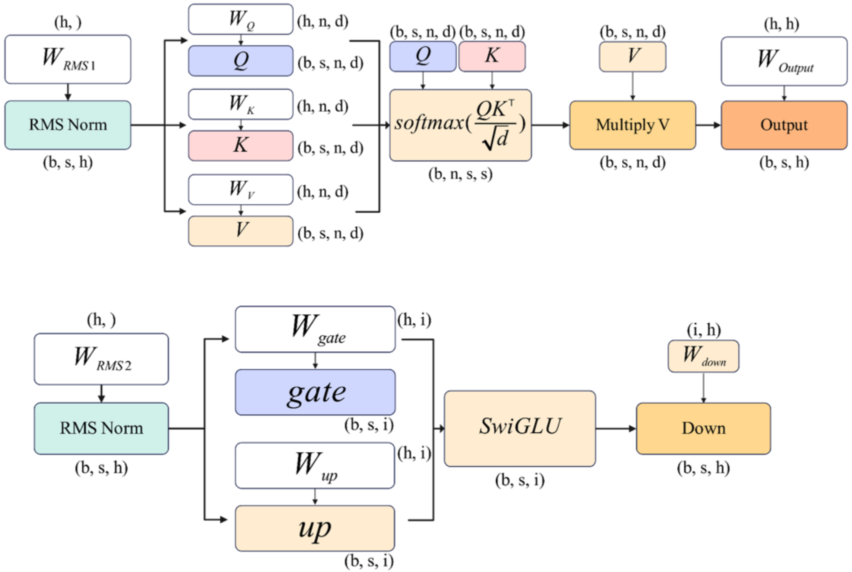
- Receive a new request R3; finish decoding R1 and R2



Continuous Batching Step-by-Step

Q: How to batch these?

- Receive a new request R3; finish decoding R1 and R2



Maximum serving batch size = 3

R1: optimizing ML systems **requires**

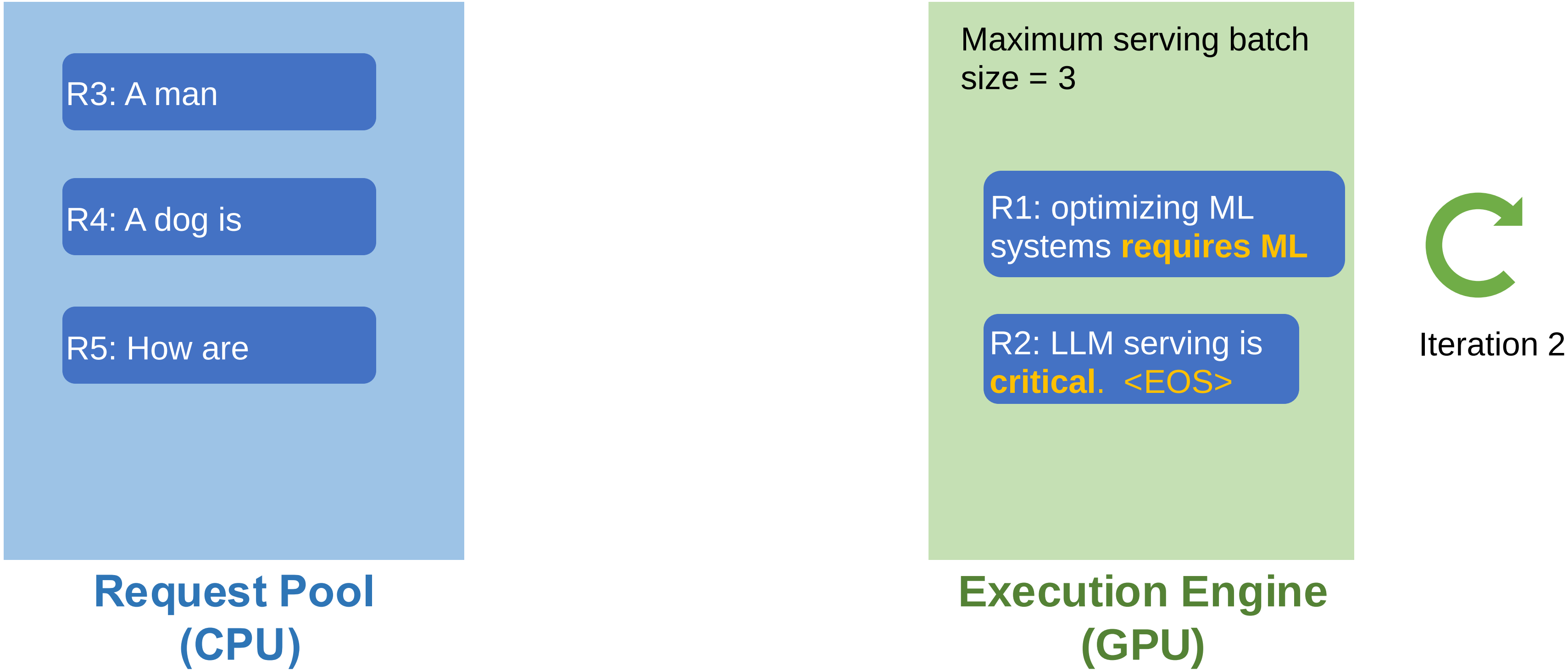
R2: LLM serving is **critical.**

Iteration 1

Execution Engine (GPU)

Traditional Batching

- Receive a new request R3; finish decoding R1 and R2



Continuous Batching

- Iteration 2: decode R1, R2, R3; receive R4, R5; R2 completes

R4: A dog is

R5: How are

Request Pool (CPU)

Maximum serving batch size = 3

R3: A man

R1: optimizing ML systems **requires ML**

R2: LLM serving is **critical**. <EOS>

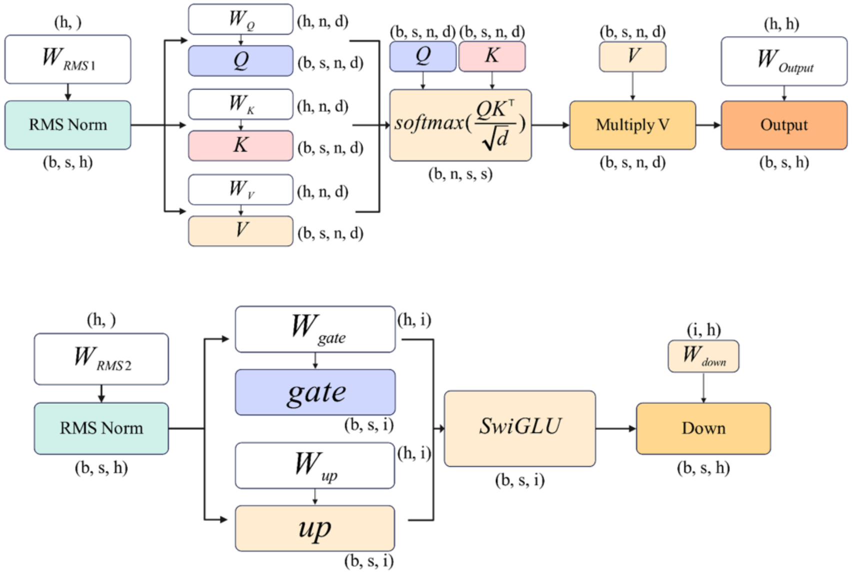
Execution Engine (GPU)

Iteration 2

Continuous Batching

Q: How to batch these?

- Iteration 2: decode R1, R2, R3; receive R4, R5; R2 completes



Maximum serving batch size = 3

R3: A man

R1: optimizing ML systems **requires ML**

R2: LLM serving is **critical** <EOS>

Iteration 2

Execution Engine (GPU)

Traditional vs. Continuous Batching

- Iteration 2: decode R1, R2, R3; receive R4, R5; R2 completes

R4: A dog is

R5: How are

R3: A man

Request Pool (CPU)

Maximum serving batch size = 3

R1: optimizing ML systems **requires ML**

R2: LLM serving is **critical**. <EOS>

Execution Engine (GPU)

Maximum serving batch size = 3

R3: A man

R1: optimizing ML systems **requires ML**

R2: LLM serving is **critical**. <EOS>

Execution Engine (GPU)

Iteration 2

Continuous Batching

- Iteration 2: decode R1, R2, R3; receive R4, R5; R2 completes

R4: A dog is

R5: How are

Request Pool (CPU)

Maximum serving batch size = 3

R3: A man

R1: optimizing ML systems **requires ML**

R2: LLM serving is **critical**. <EOS>

Execution Engine (GPU)

Iteration 2

Continuous Batching Step-by-Step

- Iteration 3: decode R1, R3, R4

R5: How are

Request Pool (CPU)

Maximum serving batch size = 3

R3: A man **is**

R1: optimizing ML systems **requires ML**

R4: A dog is

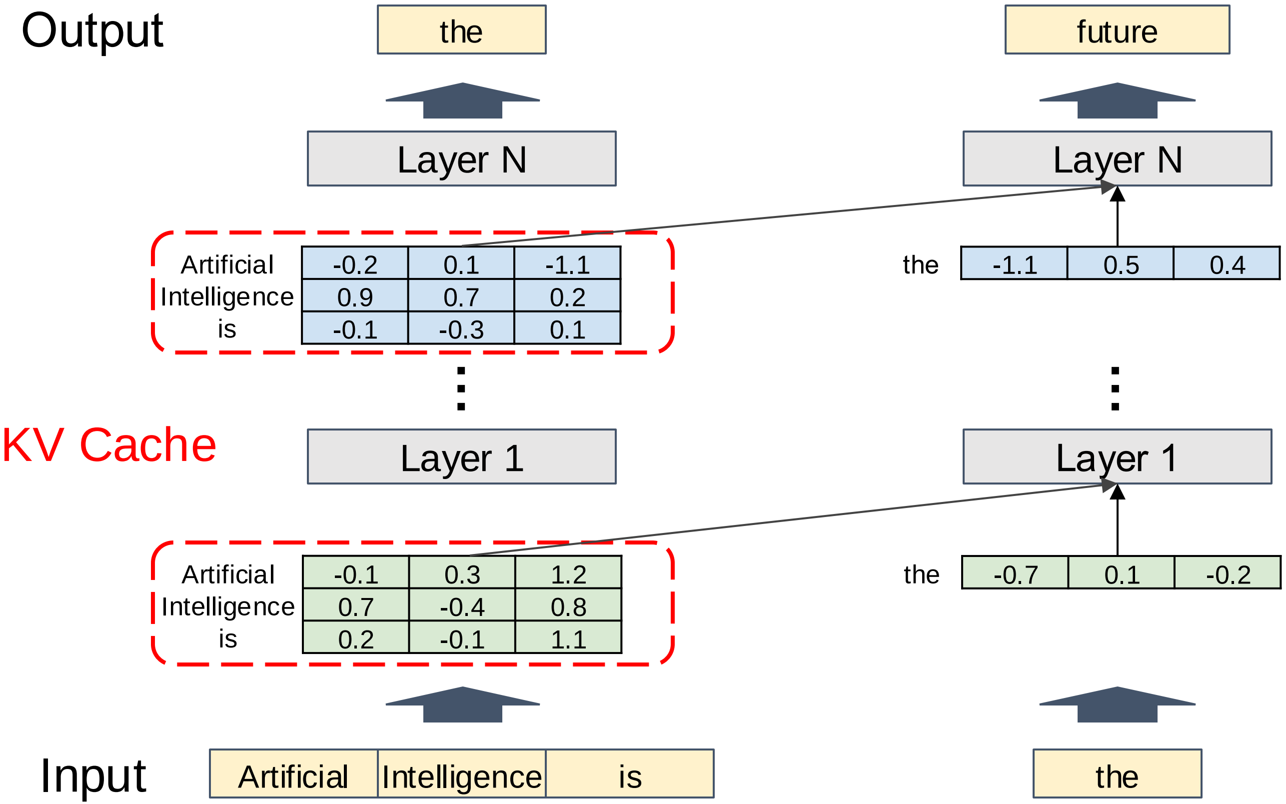
Execution Engine (GPU)

Iteration 3

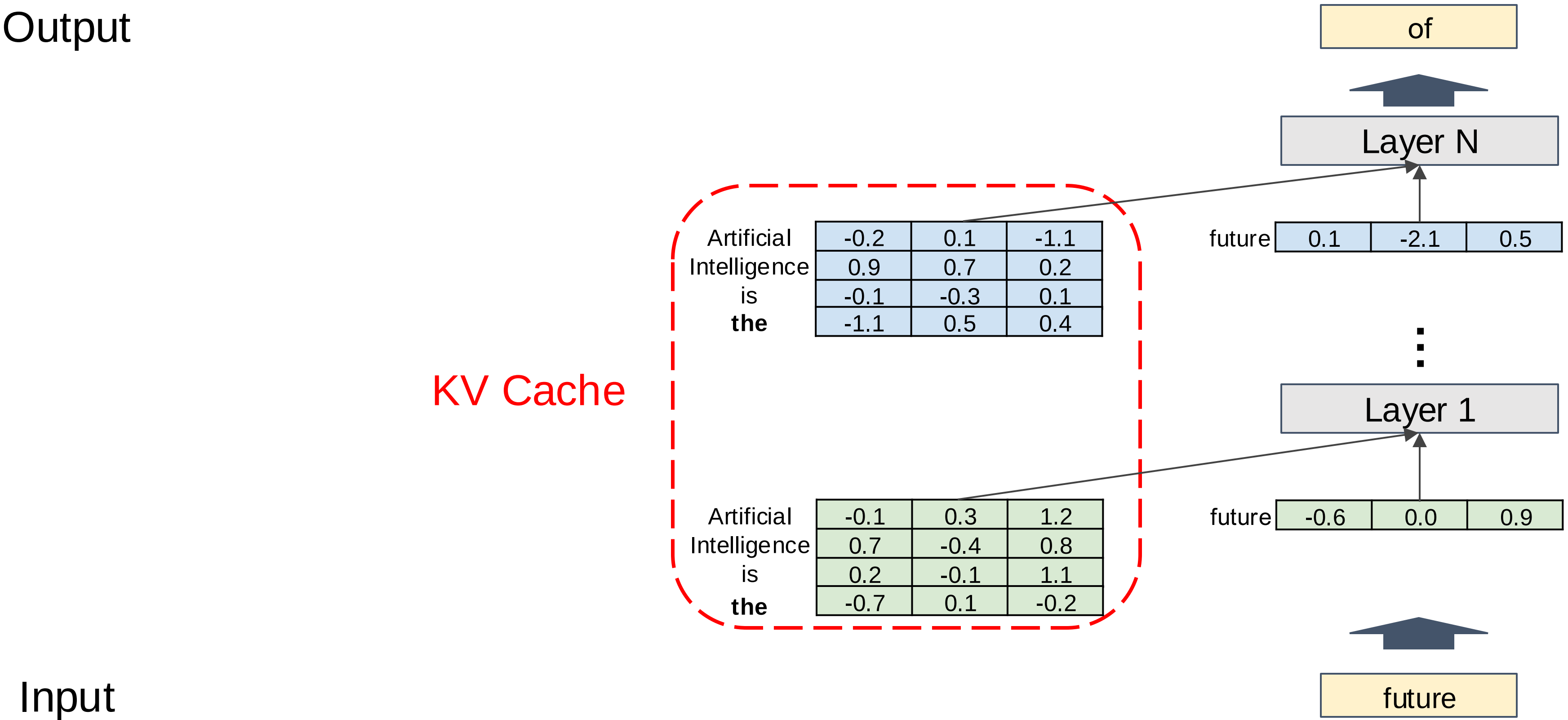
Summary: Continuous Batching

- Handle early-finished and late-arrived requests more efficiently
- Improve GPU utilization
- Key observation
  - MLP kernels are agnostic to the sequence dimension

KV Cache

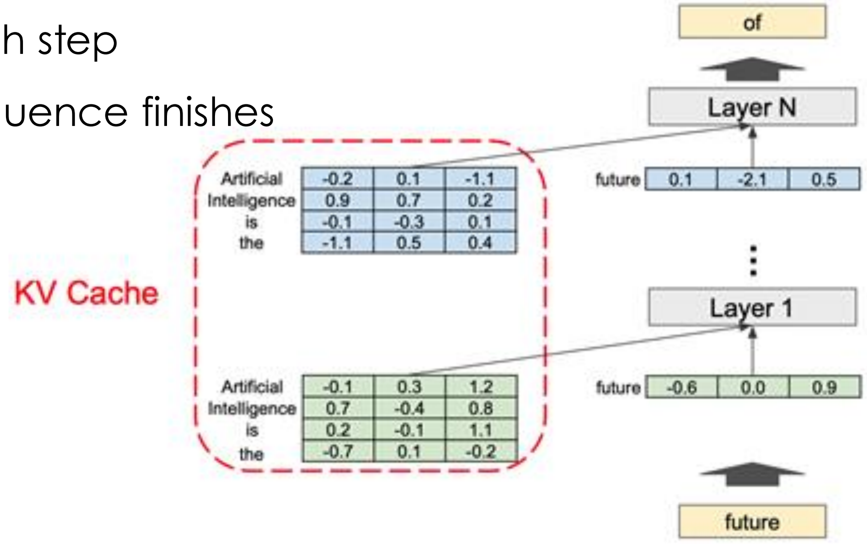

Output
the
future
Layer N
Layer N
Artificial
Intelligence
is
-0.2 0.1 -1.1
0.9 0.7 0.2
-0.1 -0.3 0.1
the
-1.1 0.5 0.4
KV Cache
Layer 1
Layer 1
Artificial
Intelligence
is
-0.1 0.3 1.2
0.7 -0.4 0.8
0.2 -0.1 1.1
the
-0.7 0.1 -0.2
Input
Artificial
Intelligence
is
the

KV Cache


Output
of
Layer N
Artificial -0.2 0.1 -1.1
Intelligence 0.9 0.7 0.2
is -0.1 -0.3 0.1
the -1.1 0.5 0.4
future 0.1 -2.1 0.5
KV Cache
Layer 1
Artificial -0.1 0.3 1.2
Intelligence 0.7 -0.4 0.8
is 0.2 -0.1 1.1
the -0.7 0.1 -0.2
future -0.6 0.0 0.9
future
Input

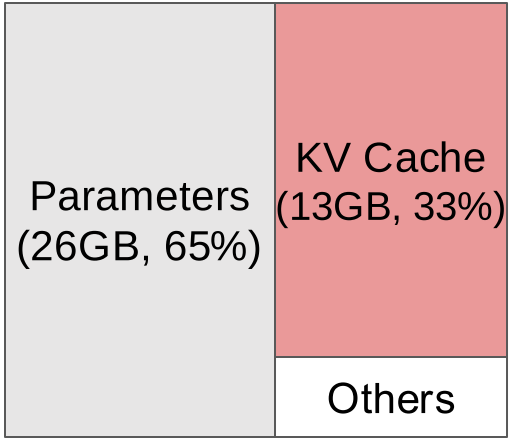
KV Cache

- Memory space to store intermediate vector representations of tokens
  - **Working set** rather than a "cache"
- The size of KV Cache dynamically grows and shrinks
  - A new token is appended in each step
  - Tokens are deleted once the sequence finishes

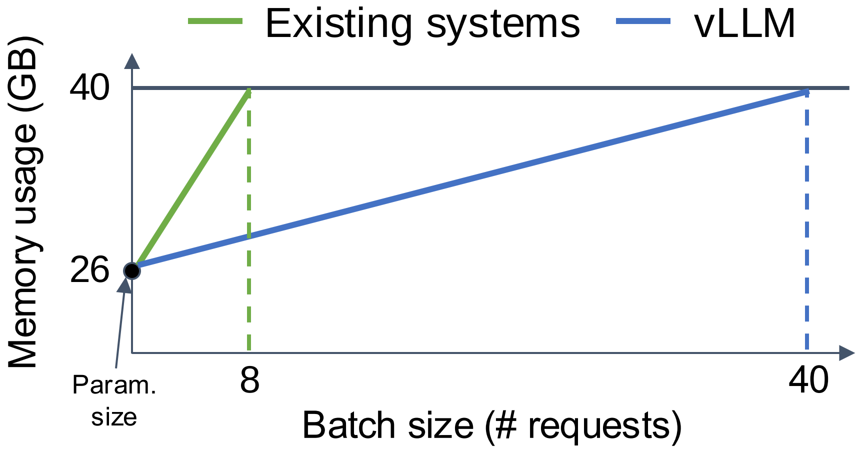


Key insight

Efficient management of KV cache is crucial for high-throughput LLM serving

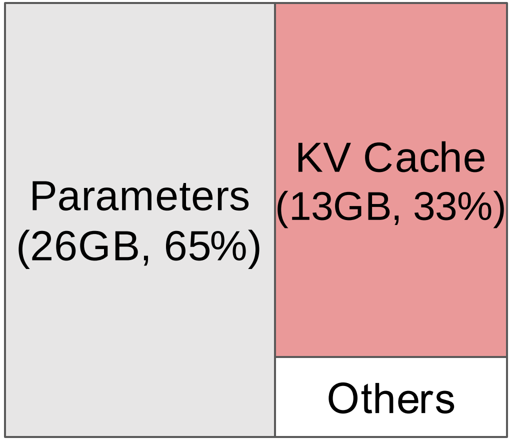


13B LLM on A100-40GB

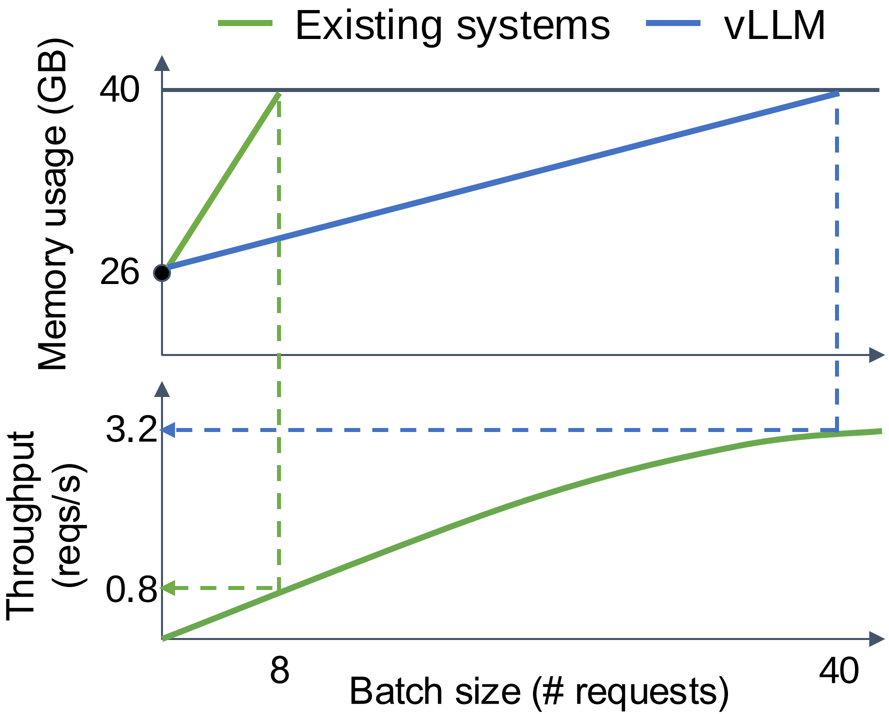


Key insight

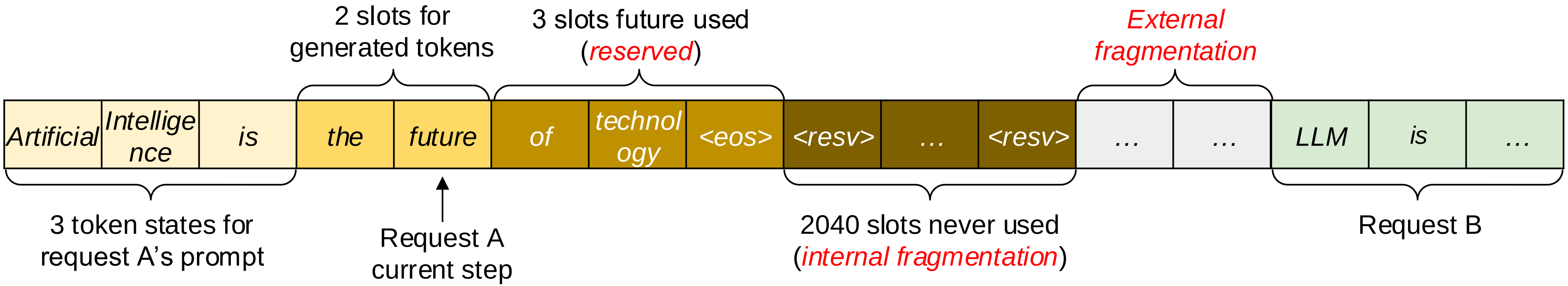
Efficient management of KV cache is crucial for high-throughput LLM serving



13B LLM on A100-40GB

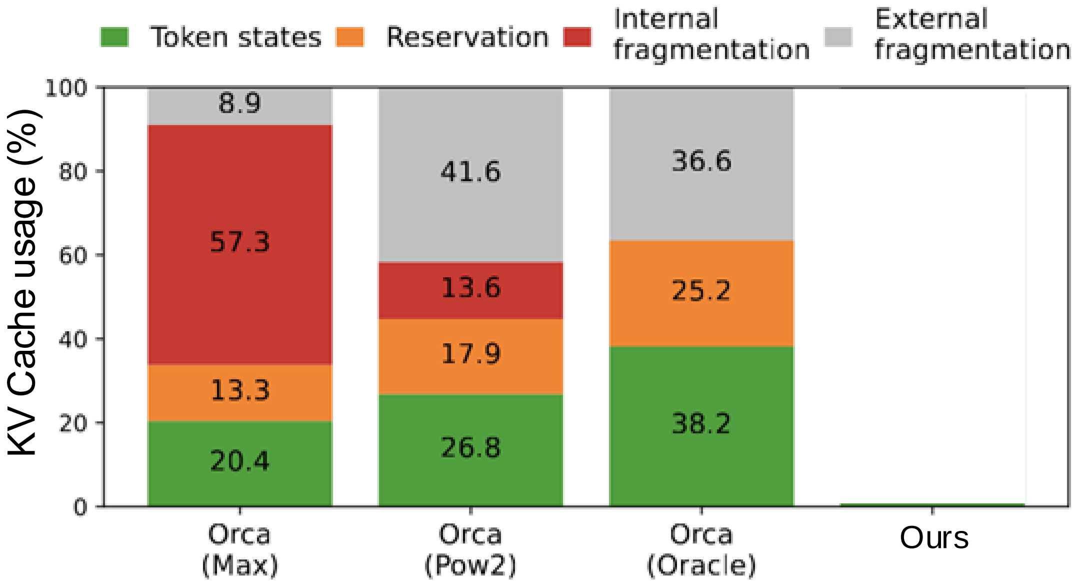


Memory waste in KV Cache



- Reservation: not used at the current step, but used in the future
- Internal fragmentation: over-allocated due to the unknown output length.

Memory waste in KV Cache

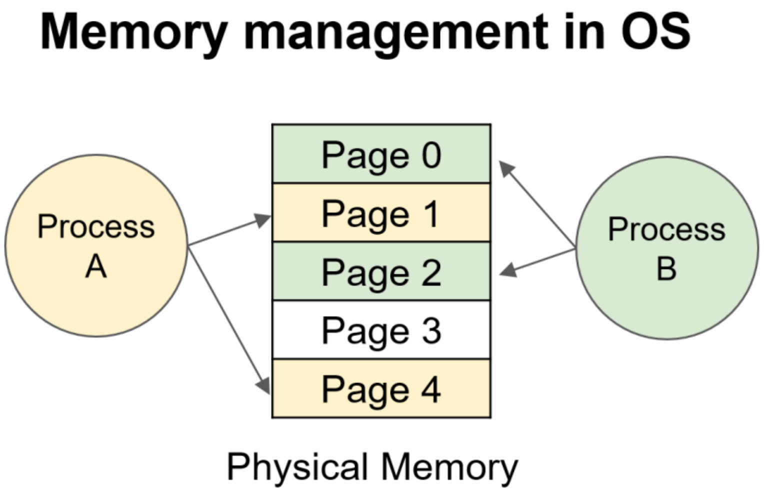


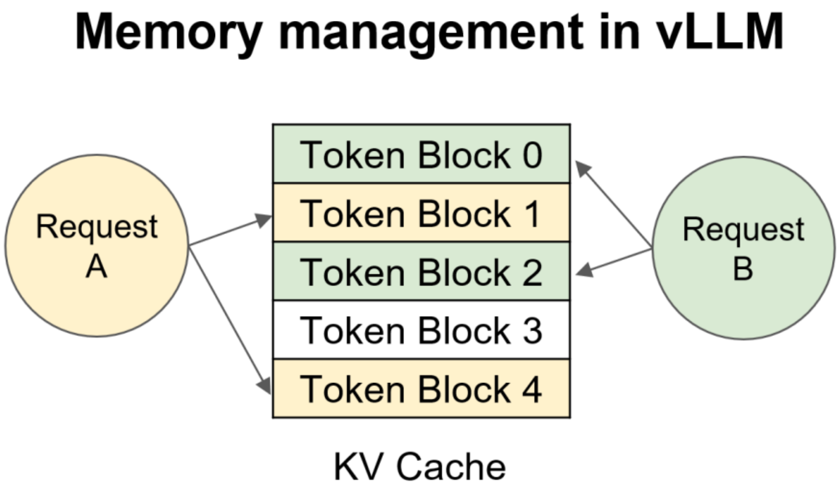
Only **20–40%** of KV cache is utilized to store token states

* Yu, G. I., Jeong, J. S., Kim, G. W., Kim, S., Chun, B. G. "Orca: A Distributed Serving System for Transformer-Based Generative Models" (OSDI 22).

vLLM: Efficient memory management for LLM inference

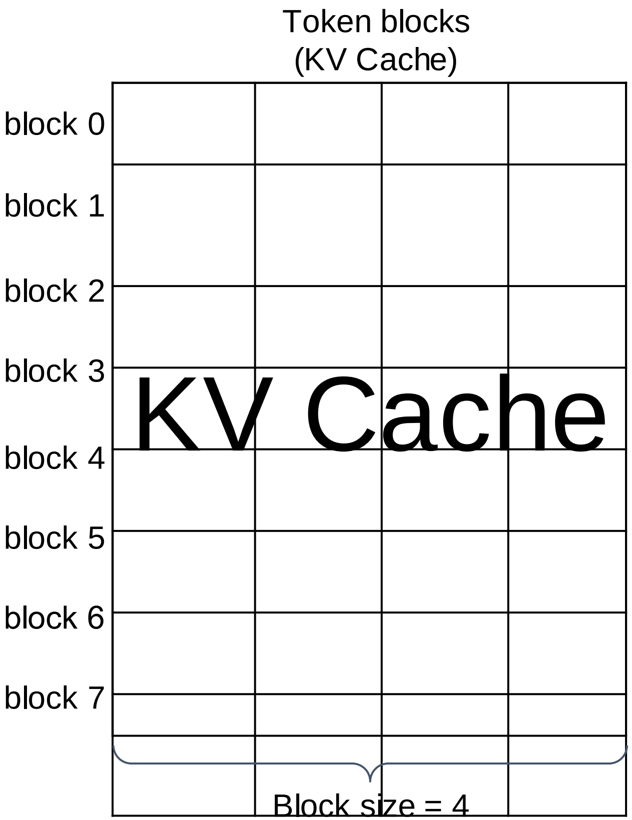
Inspired by **virtual memory** and **paging**





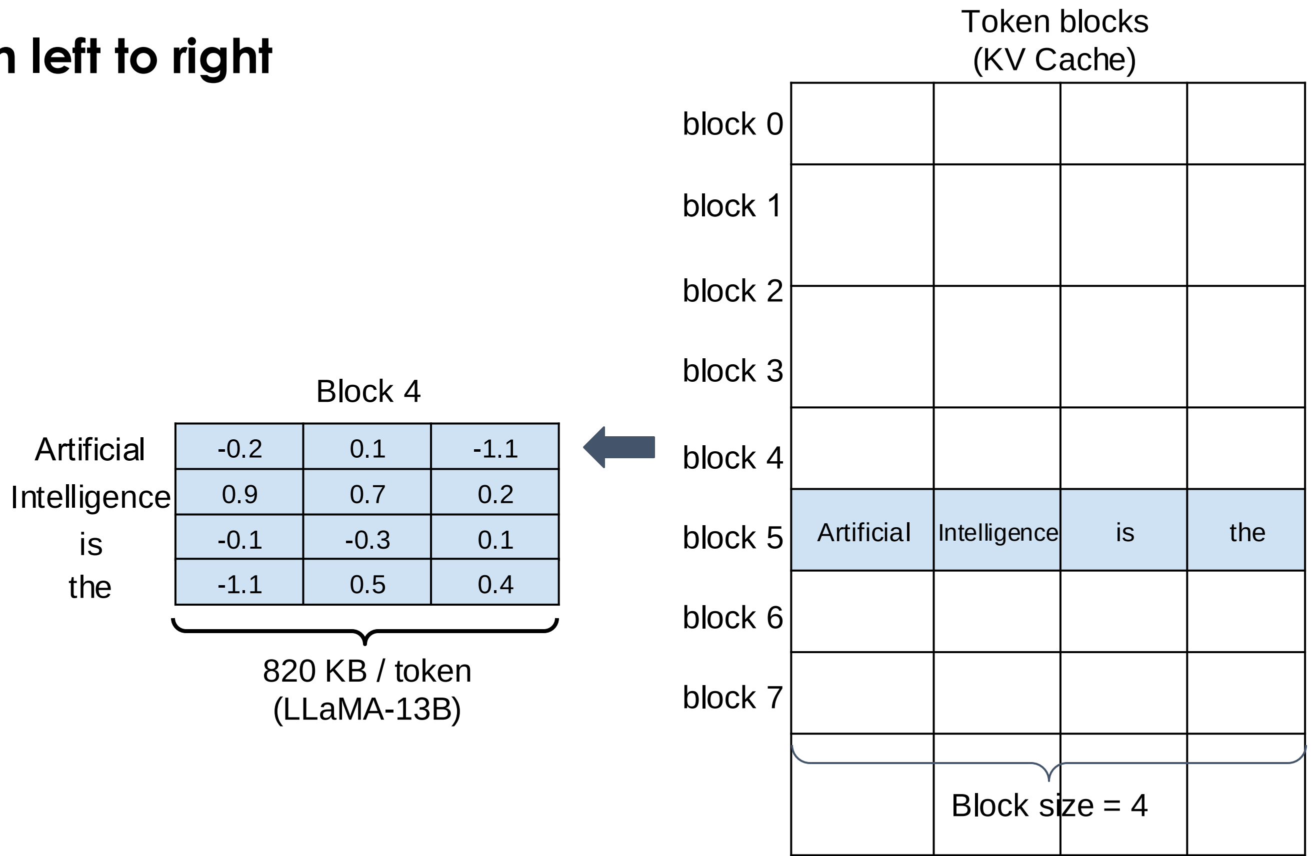
Token block

- A **fixed-size** contiguous chunk of memory that can store token states **from left to right**



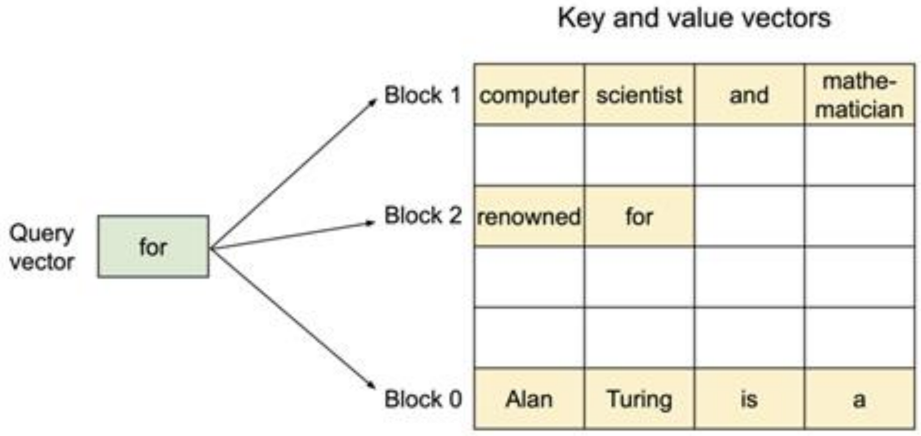
Token block

- A **fixed-size** contiguous chunk of memory that can store token states **from left to right**



Paged Attention

- An attention algorithm that allows for storing continuous keys and values in non-contiguous memory space



Logical & physical token blocks

Request A

Prompt: “Alan Turing is a computer scientist”

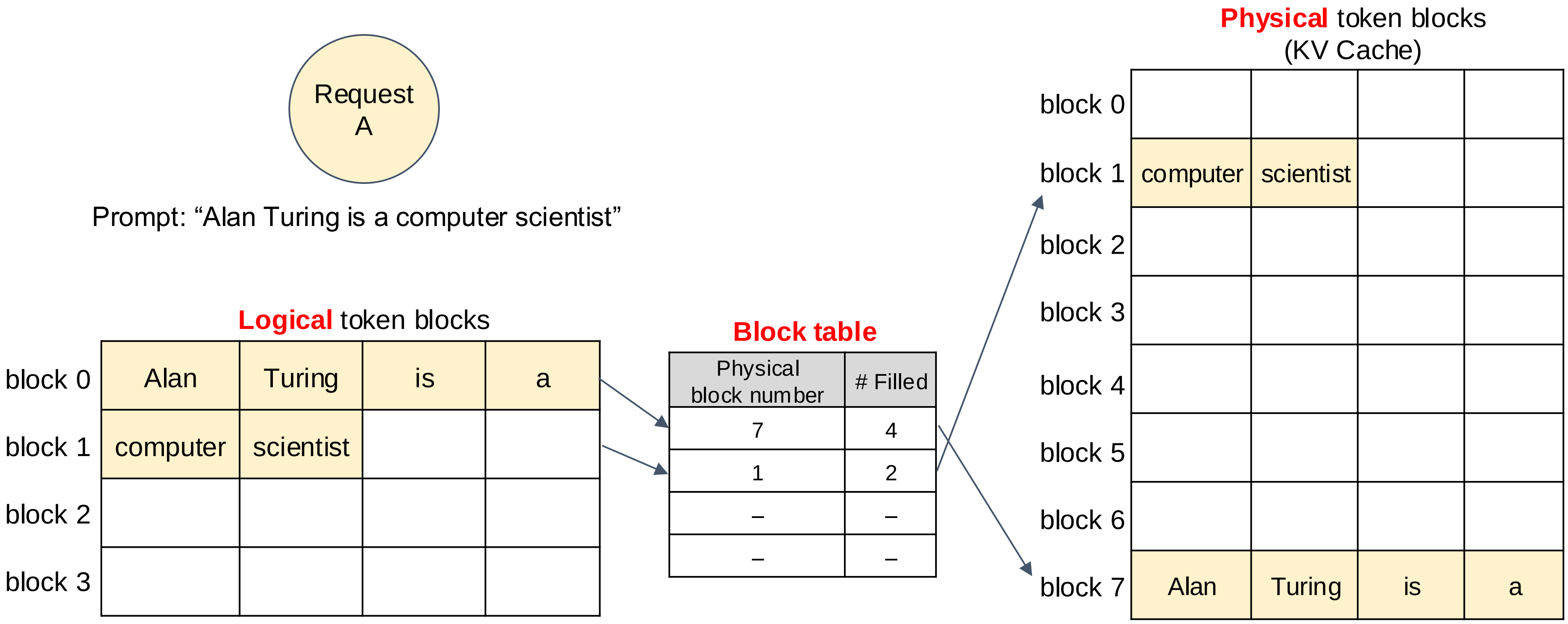
Logical token blocks

block 0	Alan	Turing	is	a
block 1	computer	scientist		
block 2				
block 3				

Physical token blocks
(KV Cache)

block 0				
block 1				
block 2				
block 3				
block 4				
block 5				
block 6				
block 7				

Logical & physical token blocks



Logical & physical token blocks

Request
A

Prompt: “Alan Turing is a computer scientist”
Completion: “<u>and</u>”

Logical token blocks

block 0	Alan	Turing	is	a
block 1	computer	scientist		
block 2				
block 3				

Block table

Physical block number	# Filled
7	4
1	2
–	–
–	–

Physical token blocks
(KV Cache)

block 0				
block 1	computer	scientist		
block 2				
block 3				
block 4				
block 5				
block 6				
block 7	Alan	Turing	is	a

Logical & physical token blocks

Request A

Prompt: “Alan Turing is a computer scientist”
Completion: “<u>and</u>”

Logical token blocks

block 0	Alan	Turing	is	a
block 1	computer	scientist	and	
block 2				
block 3				

Block table

Physical block number	# Filled
7	4
1	2
–	–
–	–

Physical token blocks (KV Cache)

block 0				
block 1	computer	scientist		
block 2				
block 3				
block 4				
block 5				
block 6				
block 7	Alan	Turing	is	a

Logical & physical token blocks

Request
A

Prompt: “Alan Turing is a computer scientist”
Completion: “<u>and</u>”

Logical token blocks

block 0	Alan	Turing	is	a
block 1	computer	scientist	and	
block 2				
block 3				

Block table

Physical block number	# Filled
7	4
1	3
–	–
–	–

Physical token blocks
(KV Cache)

block 0				
block 1	computer	scientist	and	
block 2				
block 3				
block 4				
block 5				
block 6				
block 7	Alan	Turing	is	a

Logical & physical token blocks

Request A

Prompt: “Alan Turing is a computer scientist”
Completion: “and <u>mathematician</u>”

Logical token blocks

block 0	Alan	Turing	is	a
block 1	computer	scientist	and	mathematician
block 2				
block 3				

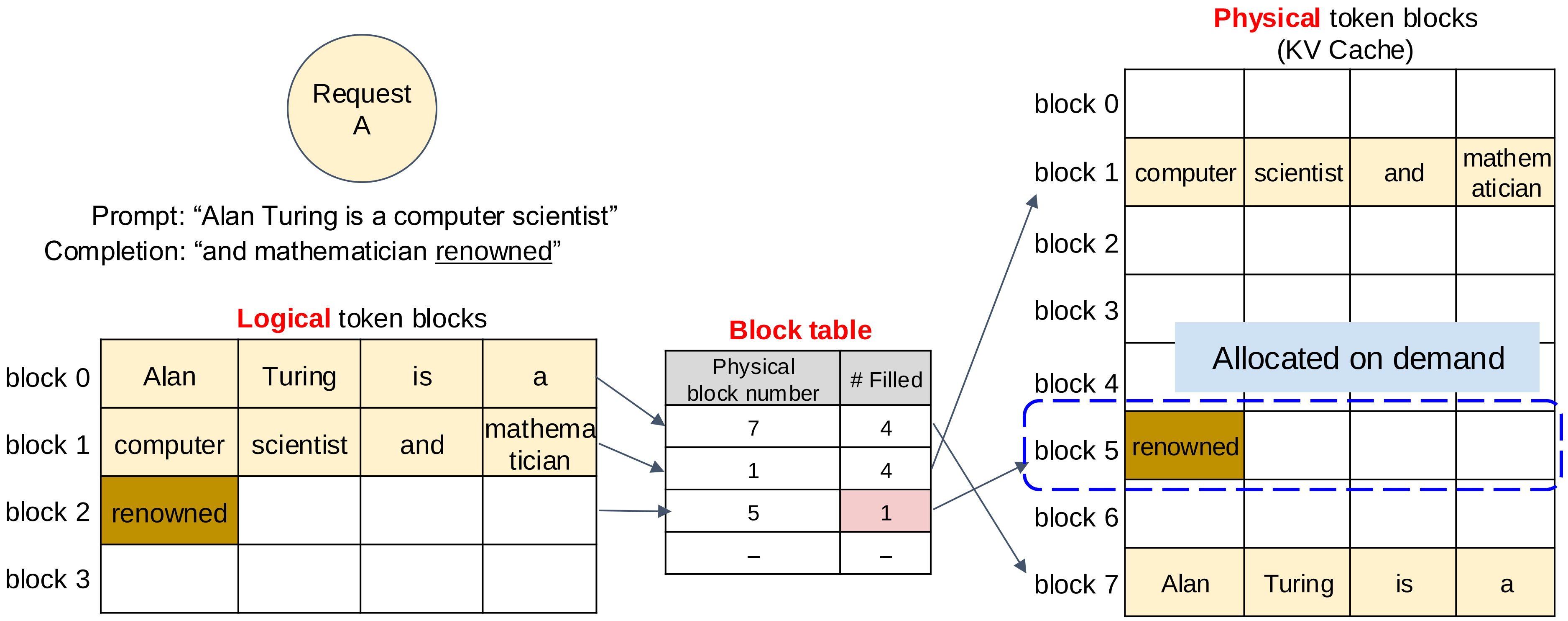
Block table

Physical block number	# Filled
7	4
1	4
–	–
–	–

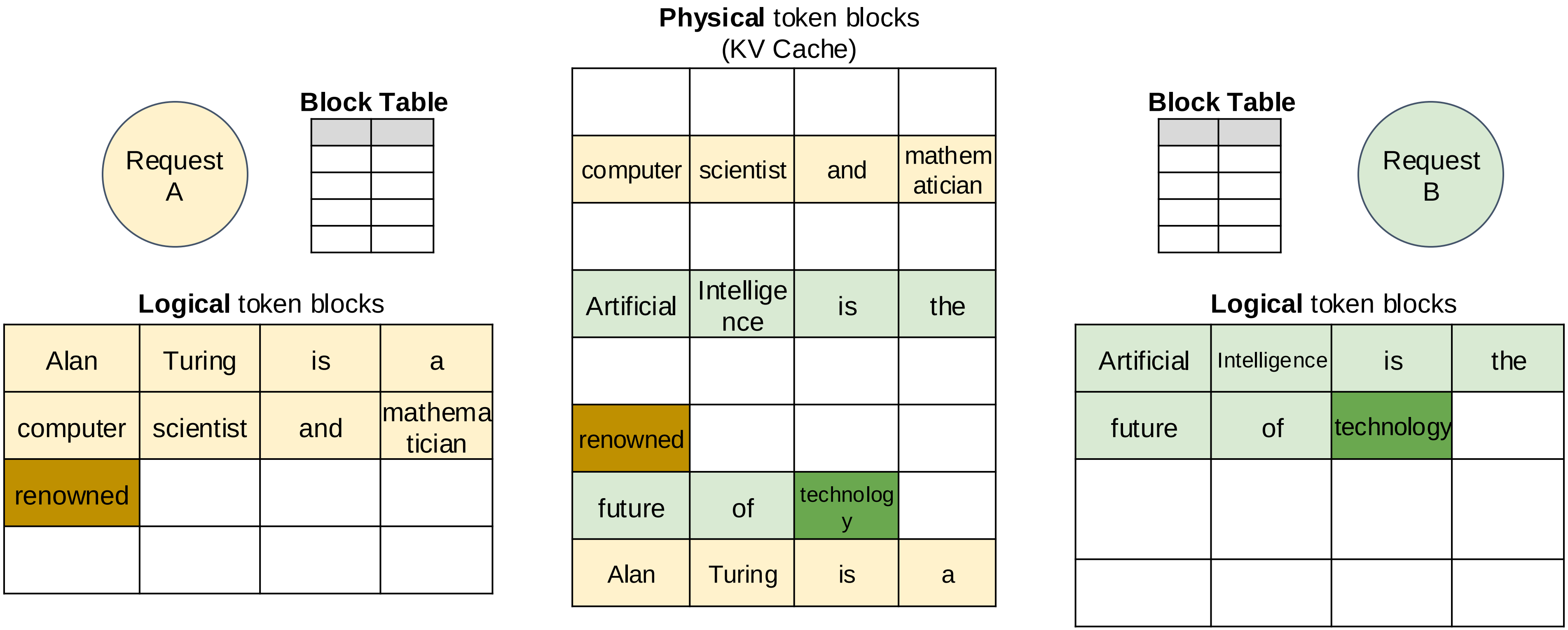
Physical token blocks (KV Cache)

block 0				
block 1	computer	scientist	and	mathematician
block 2				
block 3				
block 4				
block 5				
block 6				
block 7	Alan	Turing	is	a

Logical & physical token blocks



Serving multiple requests



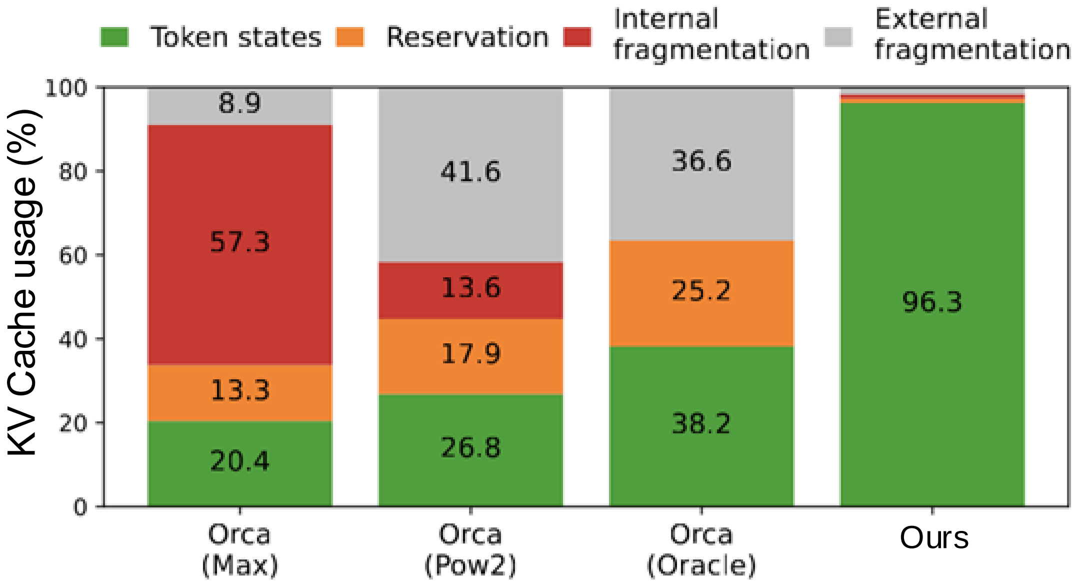
Memory efficiency of vLLM

- Minimal internal fragmentation
  - Only happens at the last block of a sequence
  - **# wasted tokens / seq < block size**
    - Sequence: O(100) – O(1000) tokens
    - Block size: 16 or 32 tokens
- No external fragmentation

Alan	Turing	is	a
computer	scientist	and	mathemati cian
renowned			

Internal fragmentation

Effectiveness of PagedAttention



96.3% KV cache utilization