



Race to the Bottom? LLM Inference in 2024



Hao Zhang

<https://cseweb.ucsd.edu/~haozhang/>

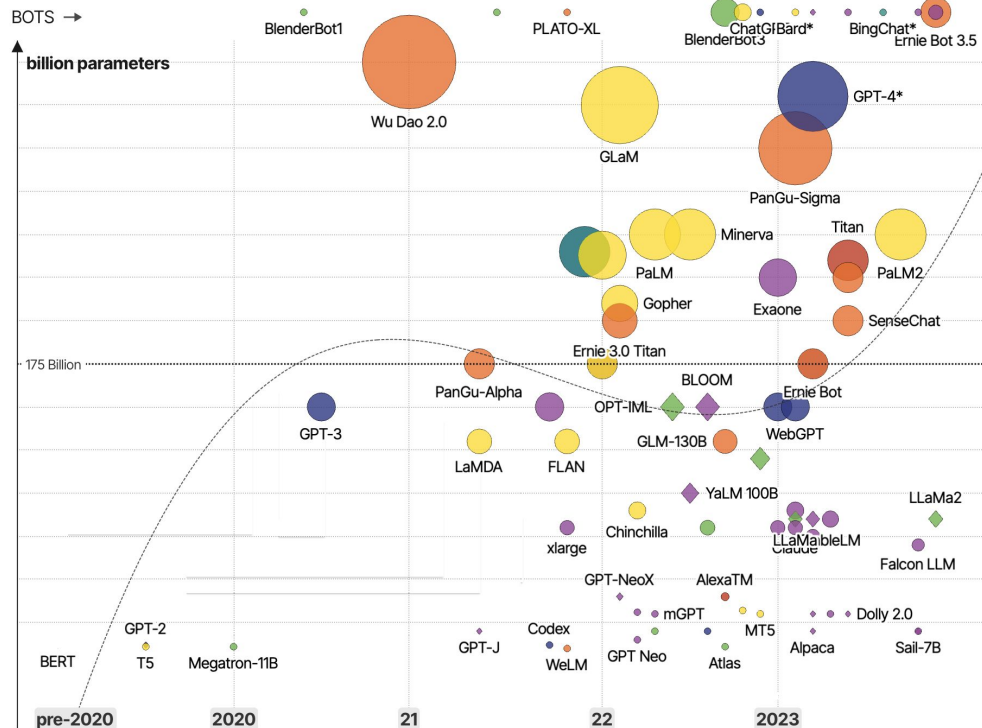
Assistant Professor, UCSD

 ChatGPT

GitHub Copilot

size = no. of parameters open-access

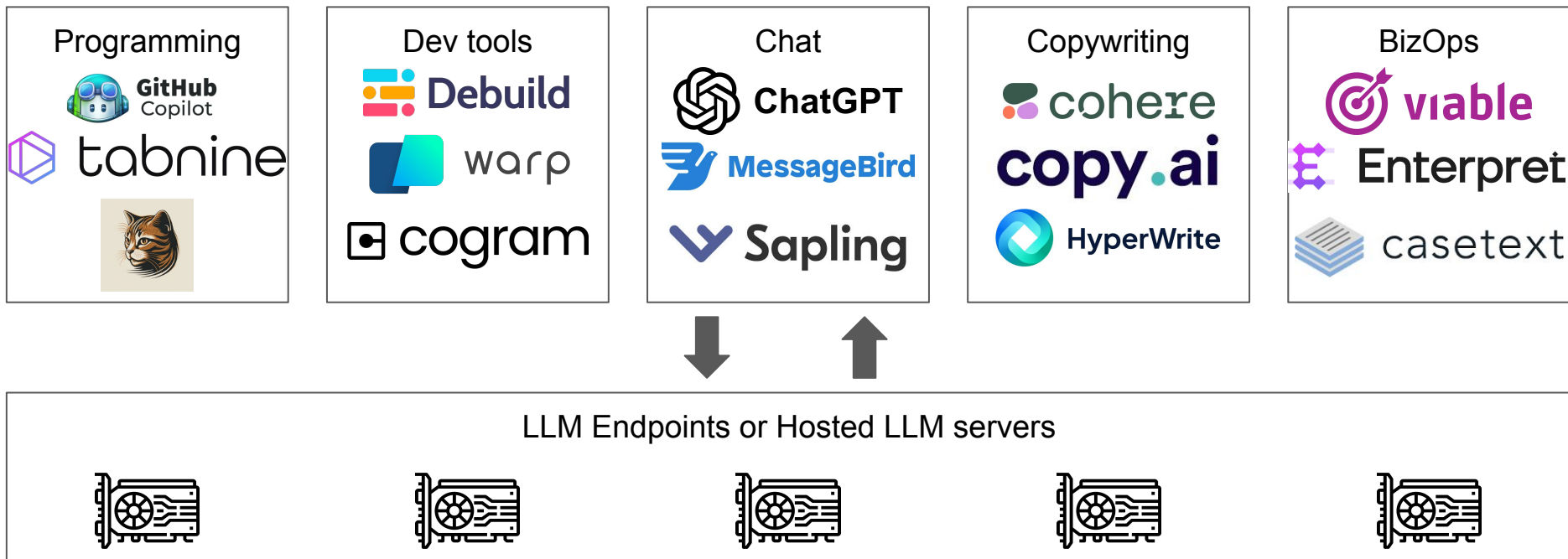
● Amazon-owned ● Chinese ● Google ● Meta / Facebook ● Microsoft ● OpenAI ● Other



David McCandless, Tom Evans, Paul Barton
Information is Beautiful // UPDATED 27th Jul 23

source: news reports, [LifeArchitect.ai](https://life-architect.ai)
* = parameters undisclosed // see the data

LLM-powered services



Serving LLMs is (surprisingly) expensive

- LLMs run on high-end GPUs such as NVIDIA A100
- Nevertheless, each GPU can only serve a handful of requests per second
 - For LLaMA-13B and moderate-size inputs, 1 A100 can process **< 1** requests per second
- A ton of GPUs are required for production-scale LLM services



Paige Bailey ✓
@DynamicWebPaige

...

Inference on LLMs is slow (frustratingly high latency), expensive (multiple GPUs or TPUs), & engineering intensive (requires specialized skills to do it well)



r/LocalLLaMA · 8 days ago
by Financial_Stranger52

Join

...

Is local LLM cheaper than ChatGPT API?

ChatGPT api only costs 0.002 dollar for 1k token. I found that LLMs like llama output only 10-20 tokens per second, which is very slow. And such machines costs over 1 dollar per hour. It seems that using api is much cheaper. Based on these observations, it seems that utilizing the ChatGPT API might be a more affordable option.

Consider Two Cases



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



Inference: fewer request,
low or offline traffic,
emphasize latency

LLM Inference and Serving: Two Typical Workloads

Q1 Serving: How to efficiently serve large amount of LLM requests while minimizing cost-per-query?

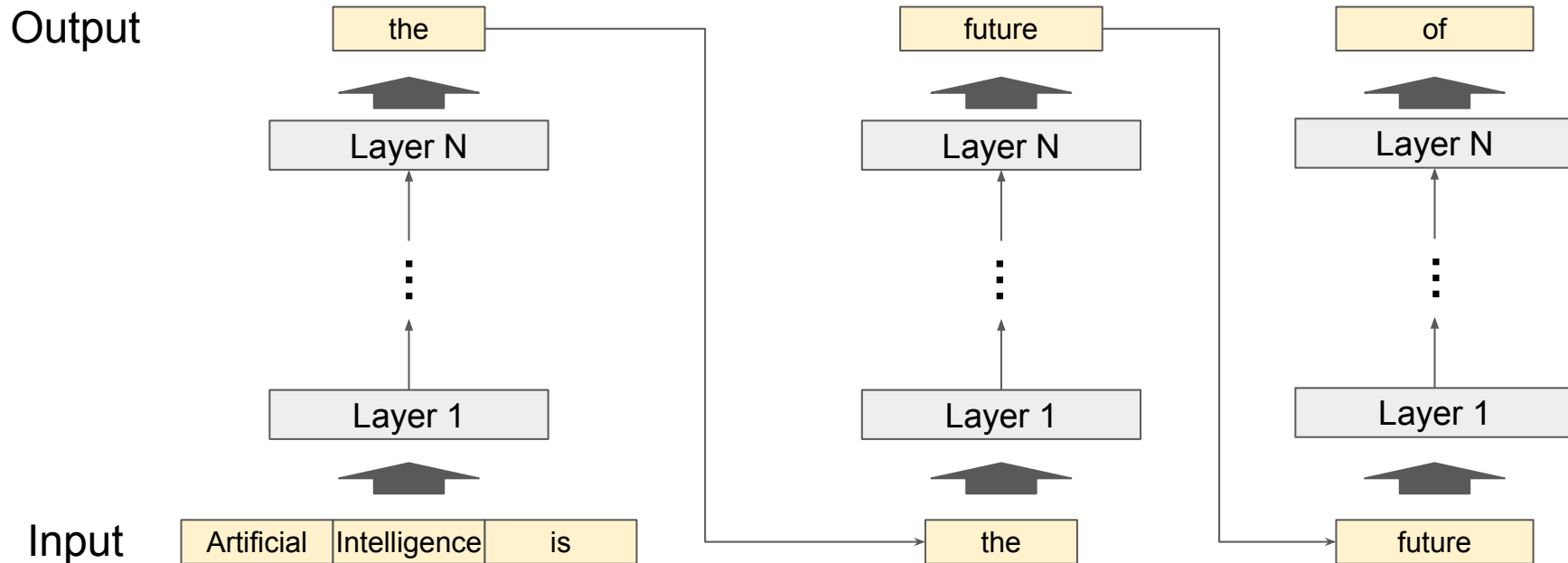
Q2 Inference: How to perform LLM inference on a single request while maximizing performance?

Serving LLMs: Core Problems

Q1 Serving: How to efficiently serve large amount of LLM requests while minimizing cost-per-query?

A: vLLM with **Paged Attention** (Next).

Inference process of LLMs

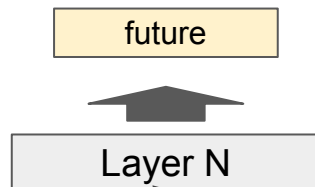
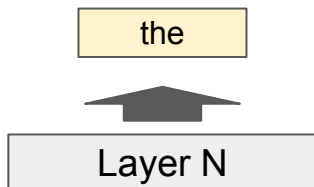


Repeat until the sequence

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

KV Cache

Output



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
------	-----	-----

⋮

⋮



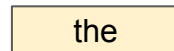
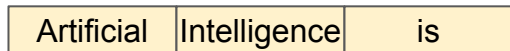
Artificial	-0.1	0.3	1.2
Intelligence	0.7	-0.4	0.8
is	0.2	-0.1	1.1

the

-0.7	0.1	-0.2
------	-----	------



Input

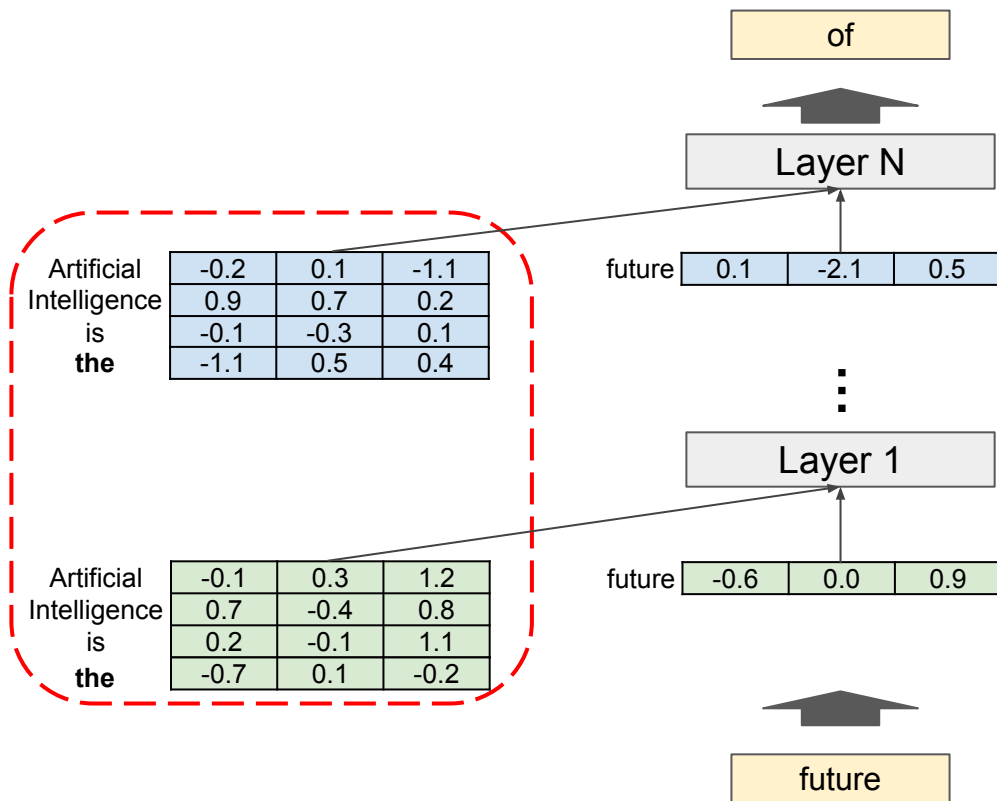


KV Cache

Output

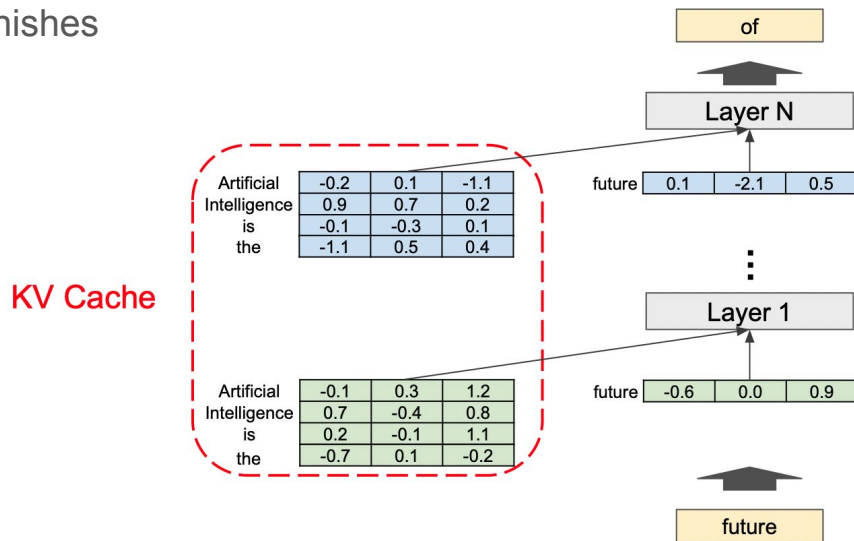
KV Cache

Input



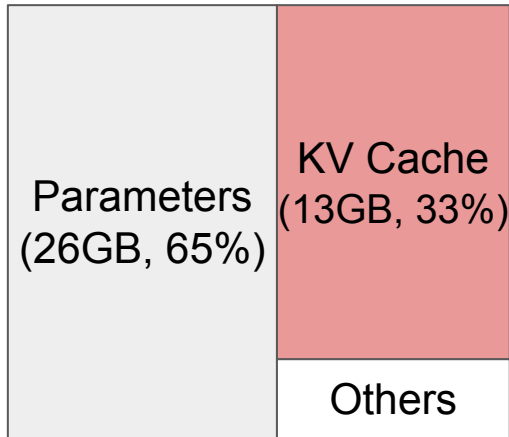
KV Cache

- 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

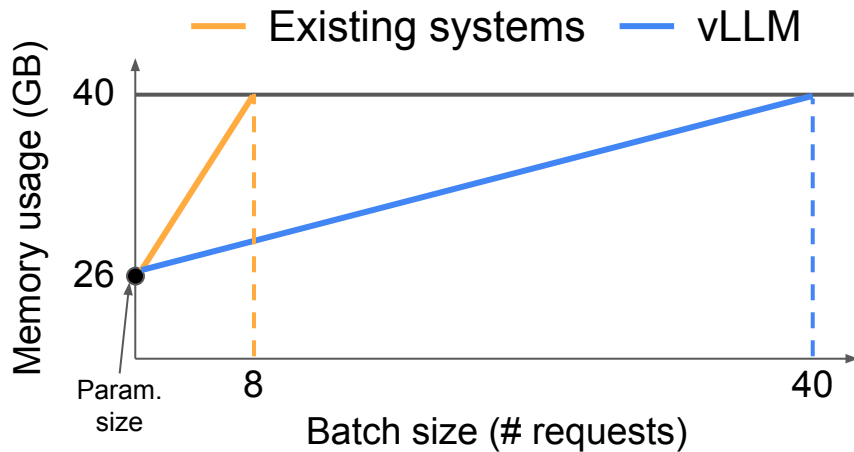


Key insight

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

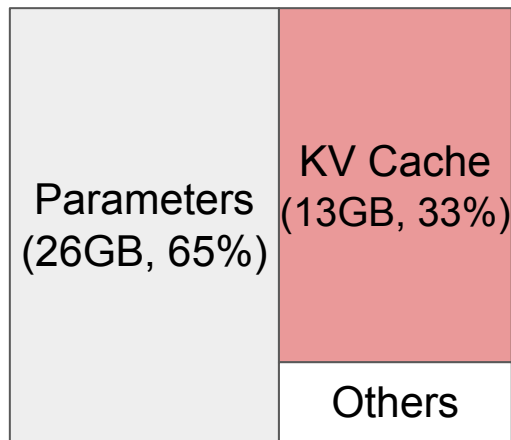


13B LLM on A100-40GB

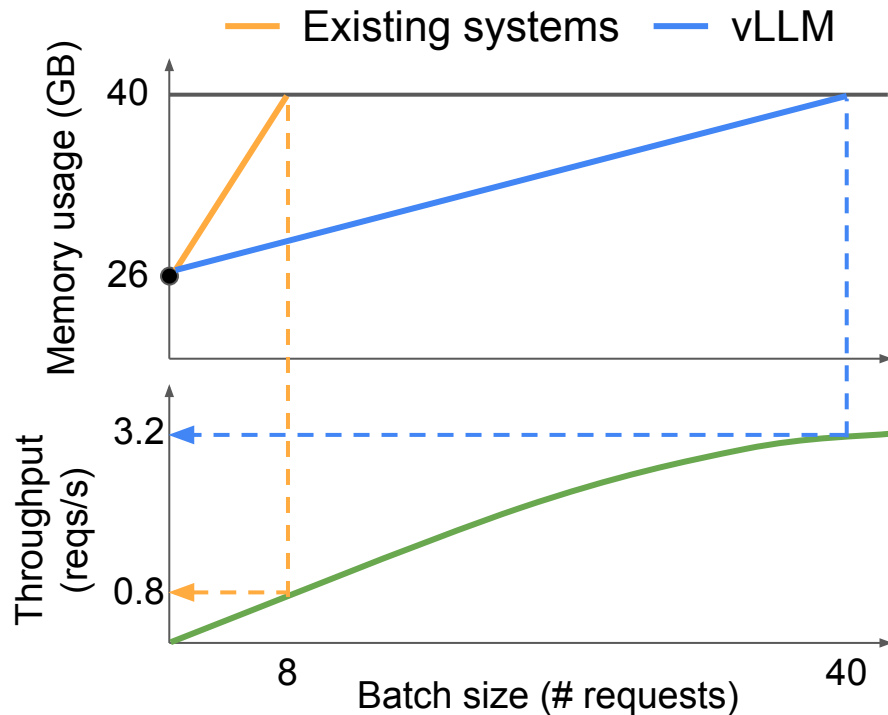


Key insight

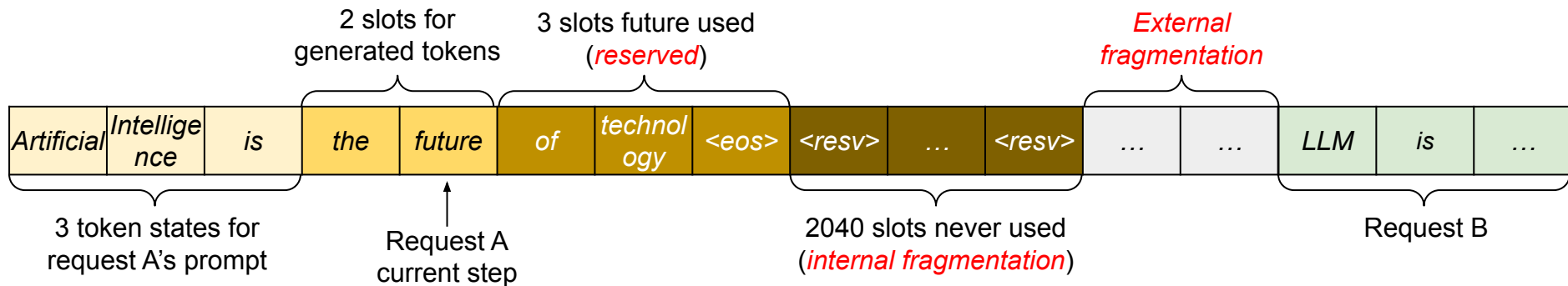
Efficient management of KV cache is crucial for high-throughput LLM serving



13B LLM on A100-40GB

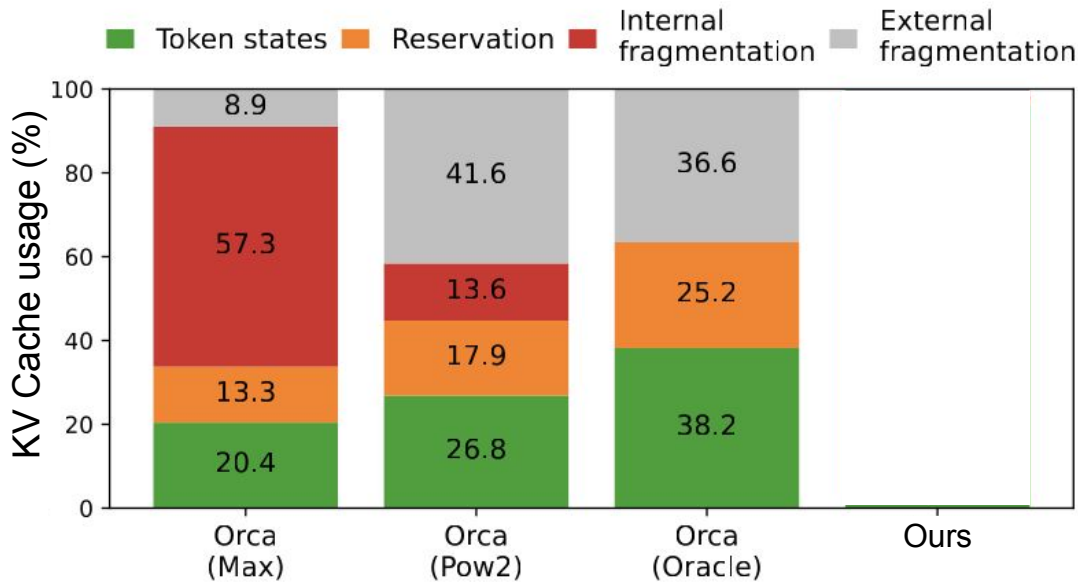


Memory waste in KV Cache



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

Memory waste in KV Cache

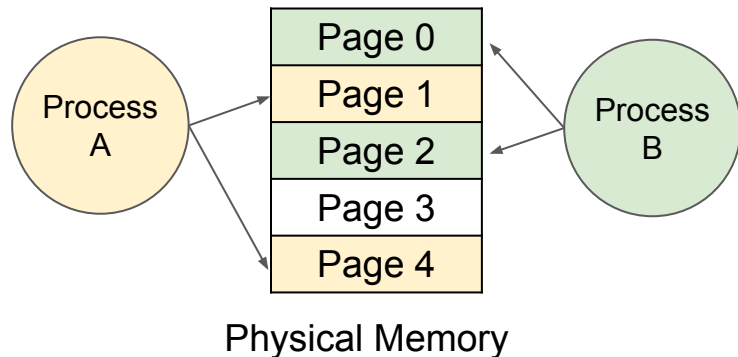


Only **20–40%** of KV cache is utilized to store token states

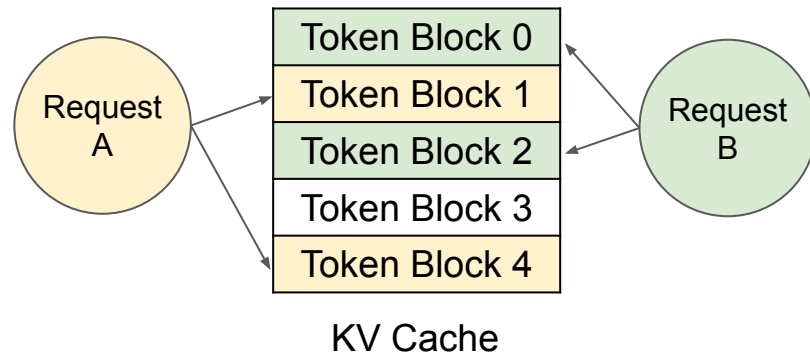
vLLM: Efficient memory management for LLM inference

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

Memory management in OS

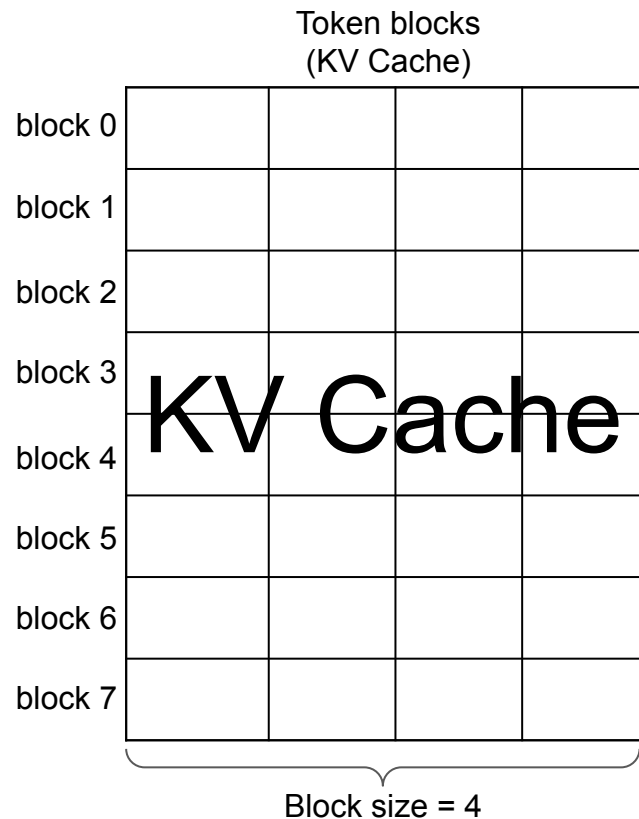


Memory management in vLLM



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**

Block 4

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

820 KB / token
(LLaMA-13B)



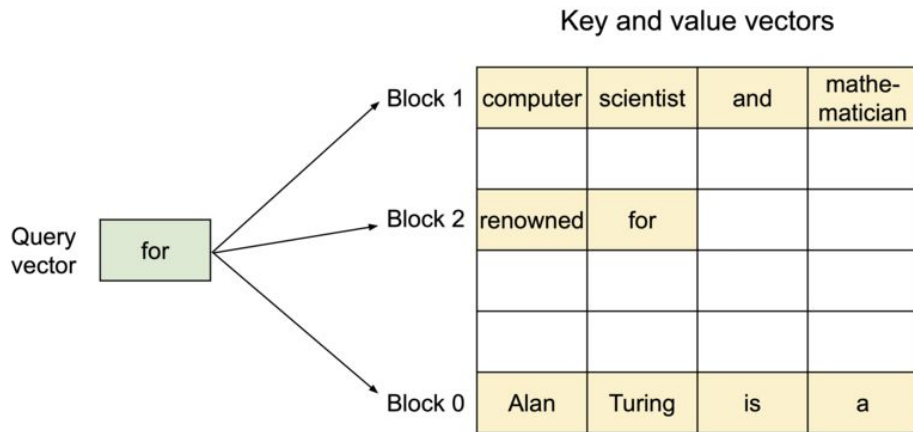
Token blocks
(KV Cache)

block 0				
block 1				
block 2				
block 3				
block 4	Artificial	Intelligence	is	the
block 5				
block 6				
block 7				

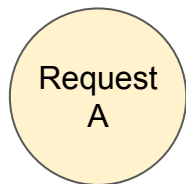
Block size = 4

Paged Attention

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



Logical & physical token blocks



Prompt: "Alan Turing is a computer scientist"

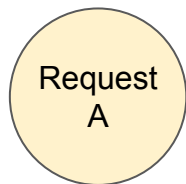
Logical token blocks

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



Prompt: "Alan Turing is a computer scientist"

Logical token blocks

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

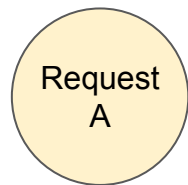
Block table

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



Prompt: "Alan Turing is a computer scientist"

Completion: "and"

Logical token blocks

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

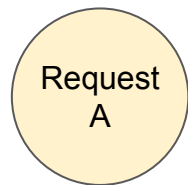
Block table

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



Prompt: "Alan Turing is a computer scientist"
Completion: "and"

Logical token blocks

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

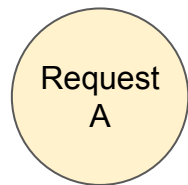
Block table

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



Prompt: "Alan Turing is a computer scientist"
Completion: "and"

Logical token blocks

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

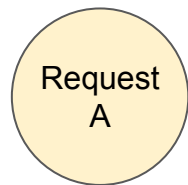
Block table

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



Prompt: "Alan Turing is a computer scientist"

Completion: "and mathematician"

Logical token blocks

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

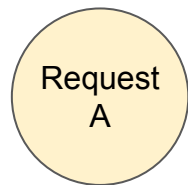
Block table

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



Prompt: "Alan Turing is a computer scientist"
Completion: "and mathematician renowned"

Logical token blocks

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

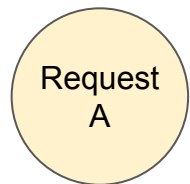
Block table

Physical block number	# Filled
7	4
1	4
5	1
—	—

Physical token blocks
(KV Cache)

block 0				
block 1	computer	scientist	and	mathem atician
block 2				
block 3				
block 4	Allocated on demand			
block 5	renowned			
block 6				
block 7	Alan	Turing	is	a

Serving multiple requests



Block Table

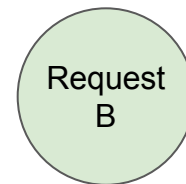
Logical token blocks

Alan	Turing	is	a
computer	scientist	and	mathematician
renowned			

Physical token blocks
(KV Cache)

computer	scientist	and	mathematician
Artificial	Intelligence	is	the
renowned			
future	of	technology	
Alan	Turing	is	a

Block Table



Logical token blocks

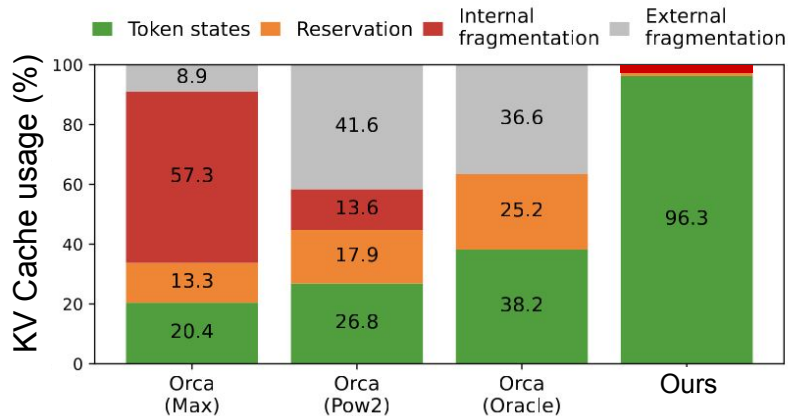
Artificial	Intelligence	is	the
future	of	technology	

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

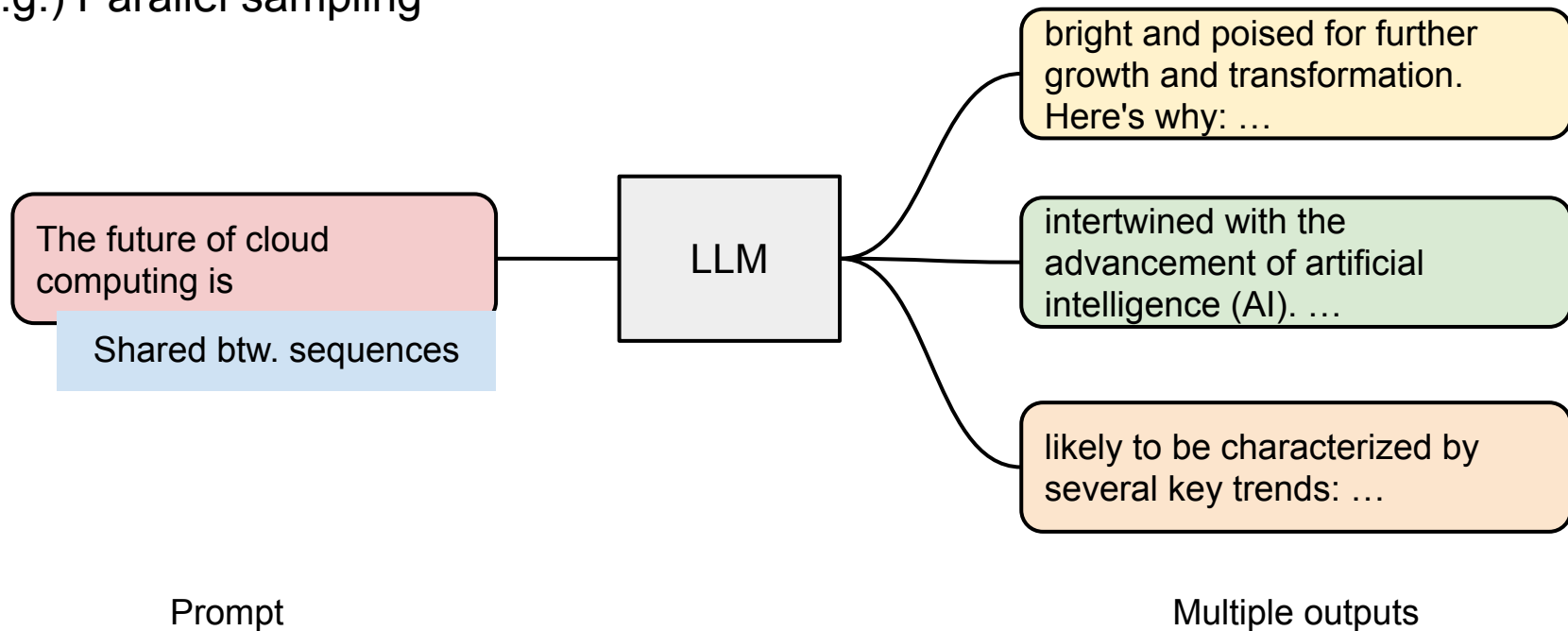
Alan	Turing	is	a
computer	scientist	and	mathemati cian
renowned			

Internal fragmentation

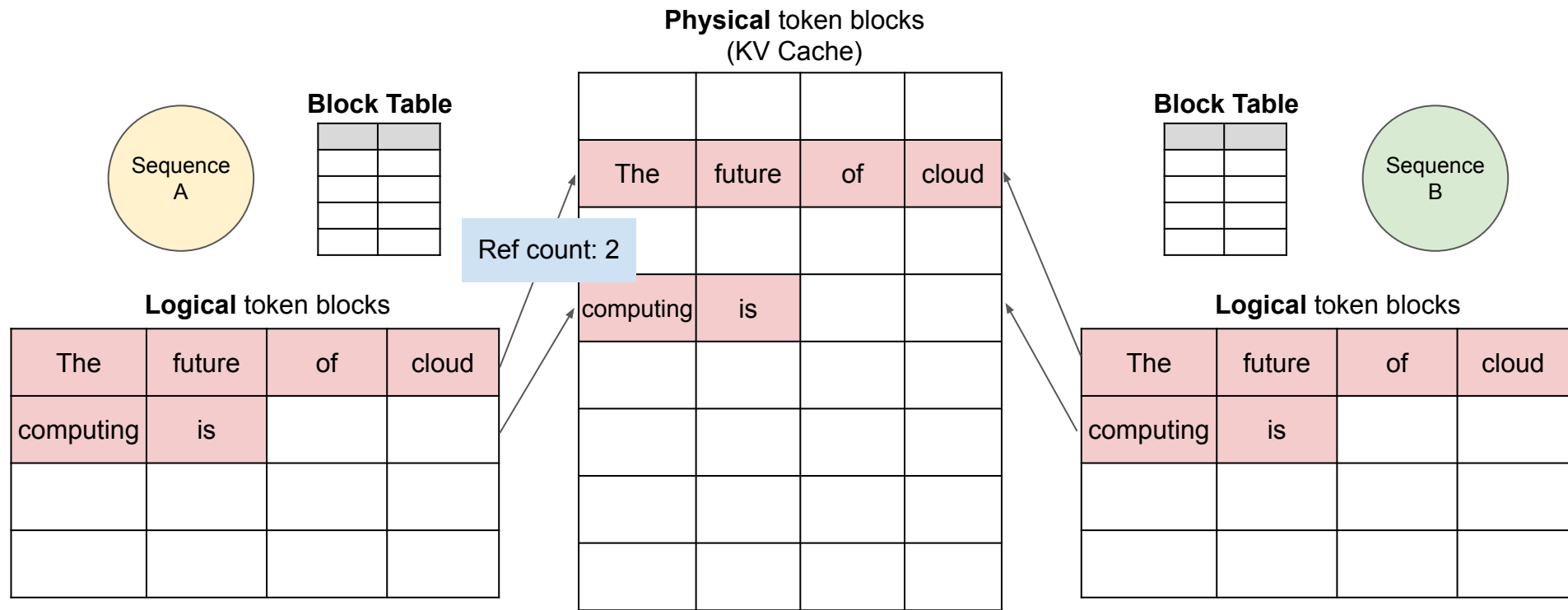


Dynamic block mapping enables sharing

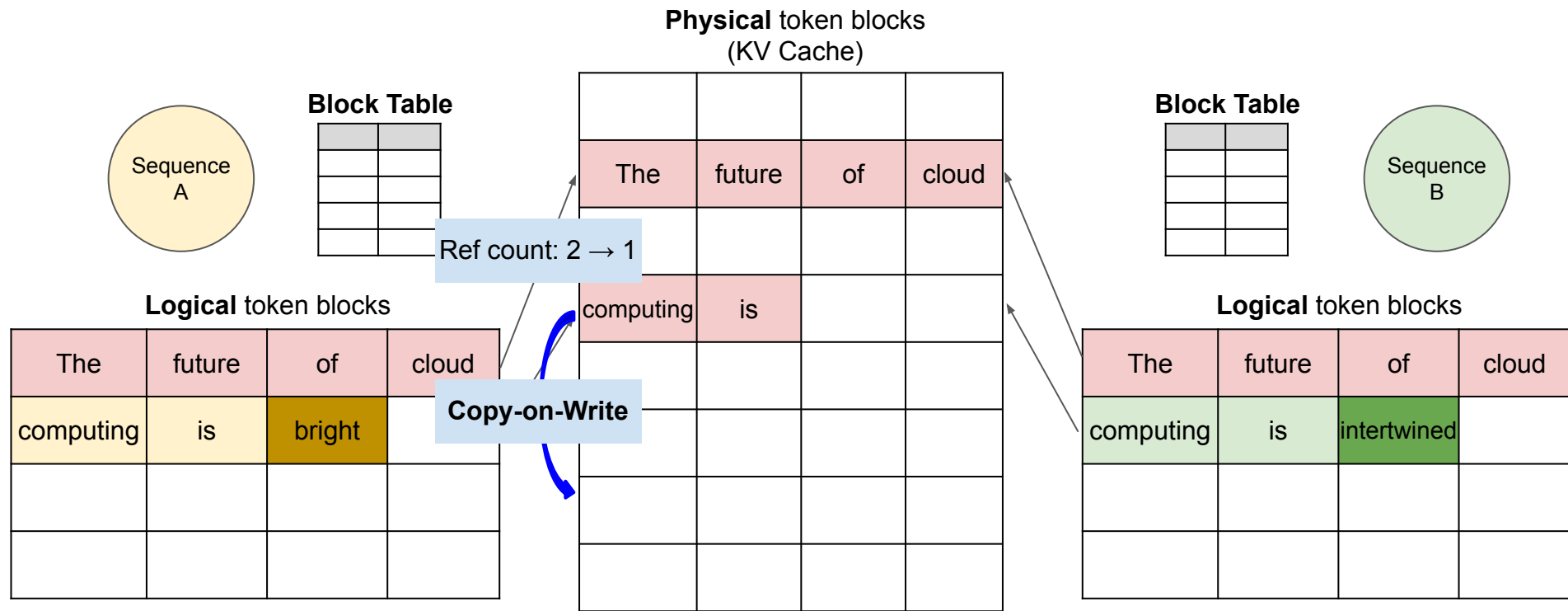
E.g.) Parallel sampling



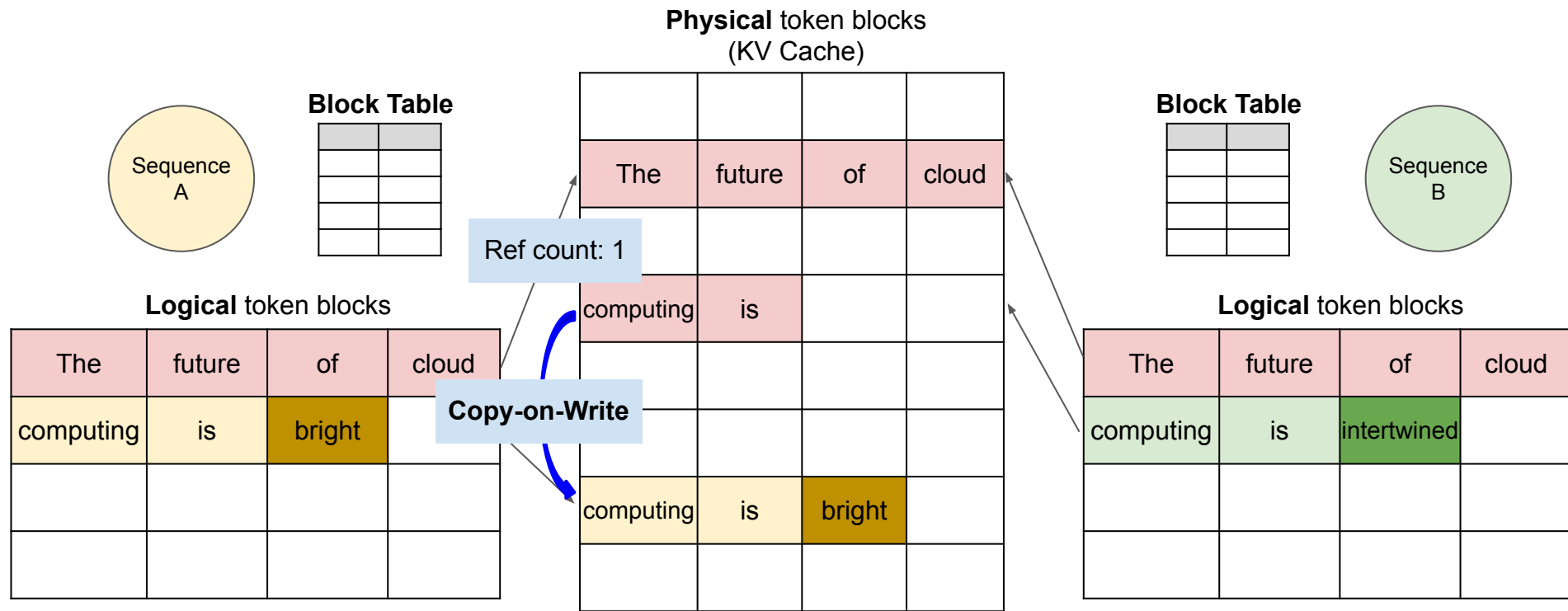
Sharing token blocks



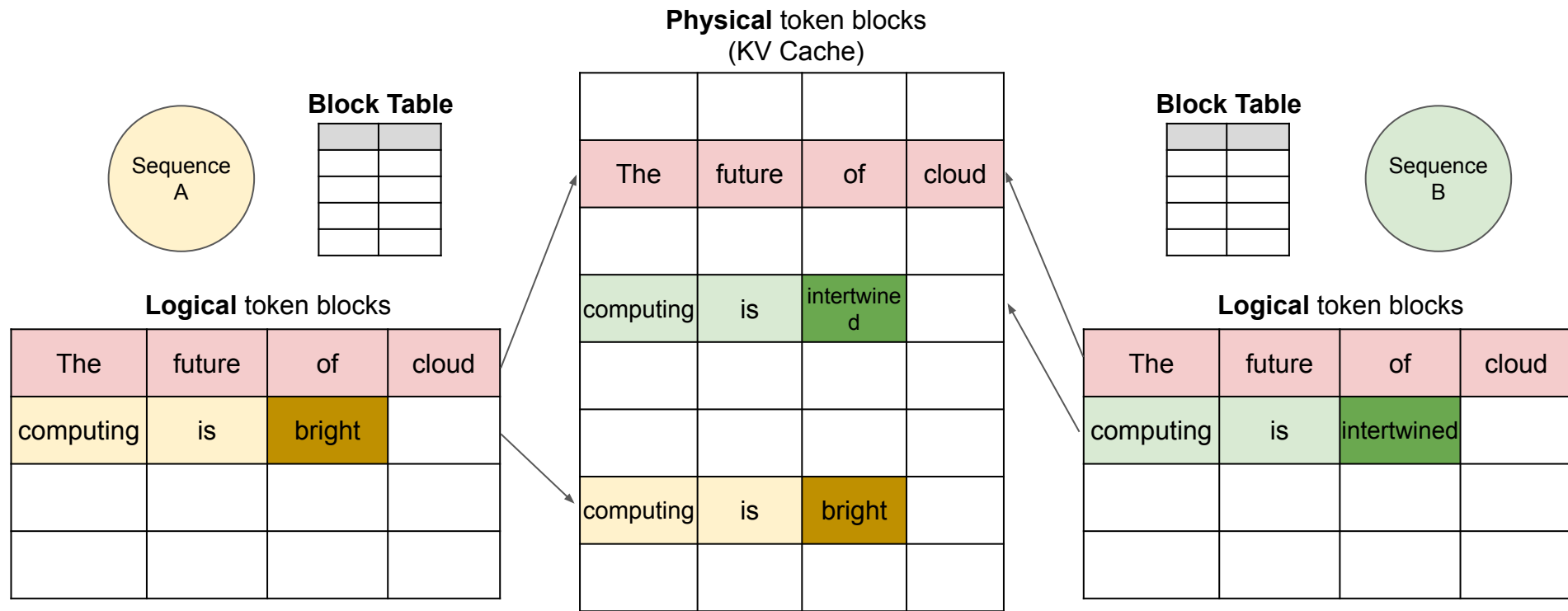
Sharing token blocks



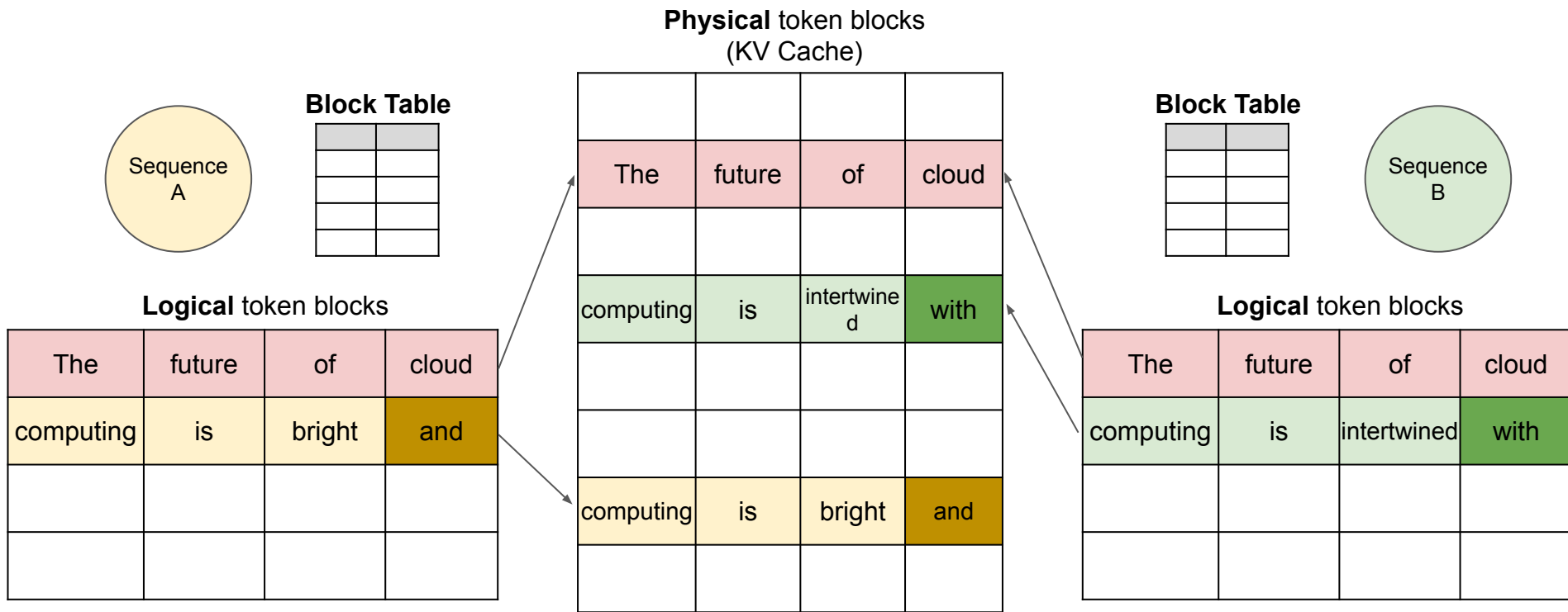
Sharing token blocks



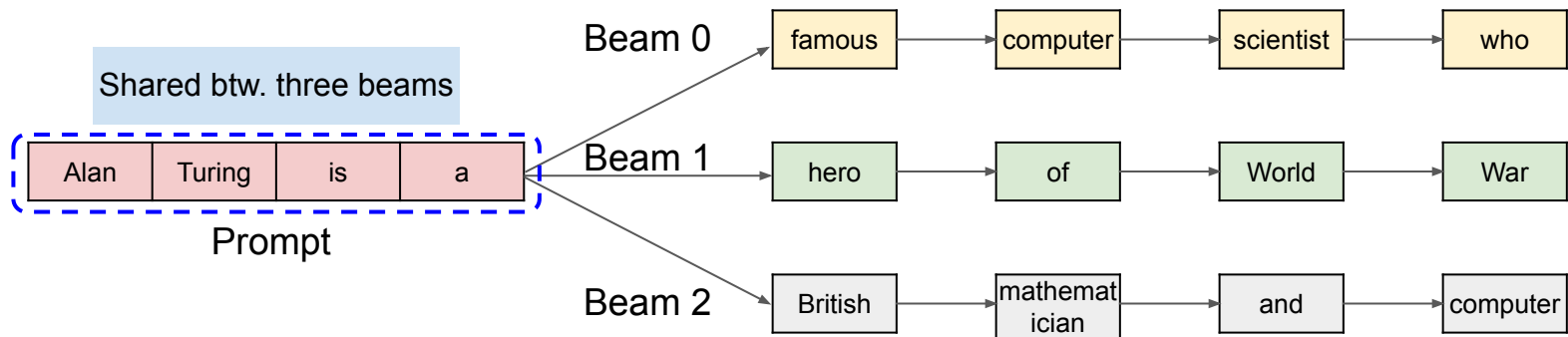
Sharing token blocks



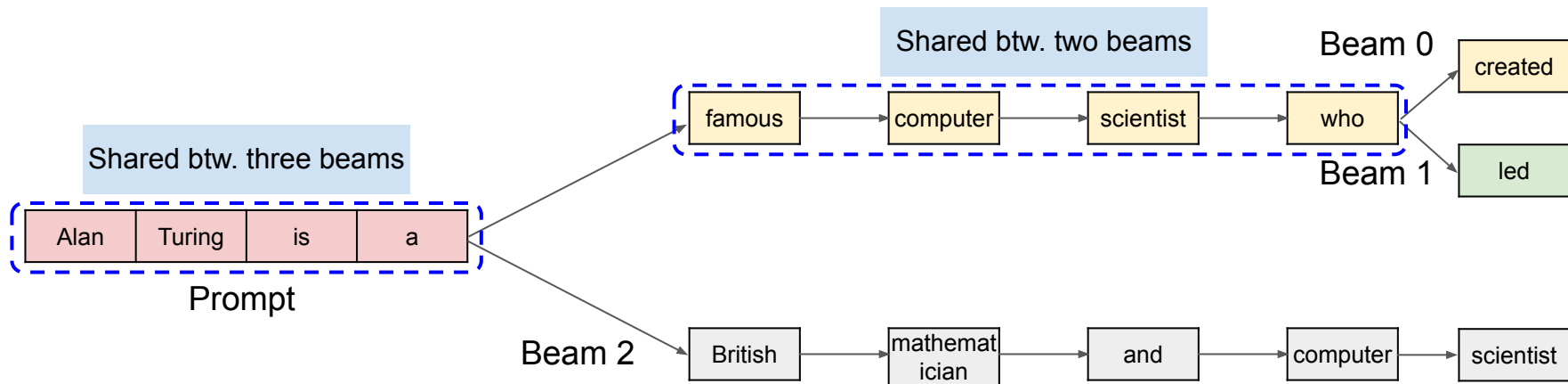
Sharing token blocks



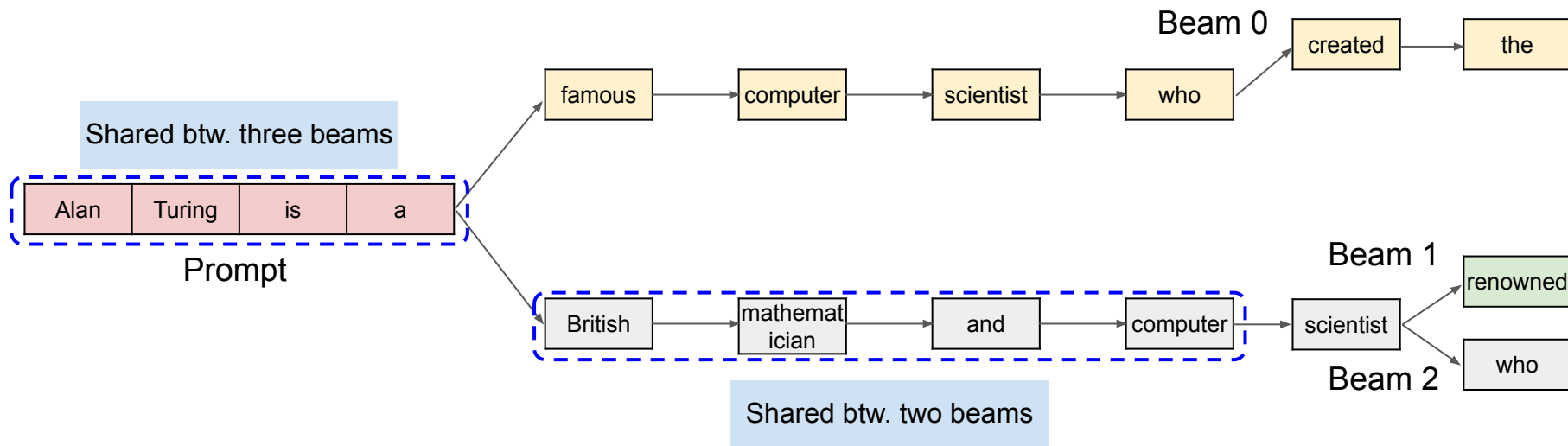
More complex sharing: beam search



More complex sharing: beam search

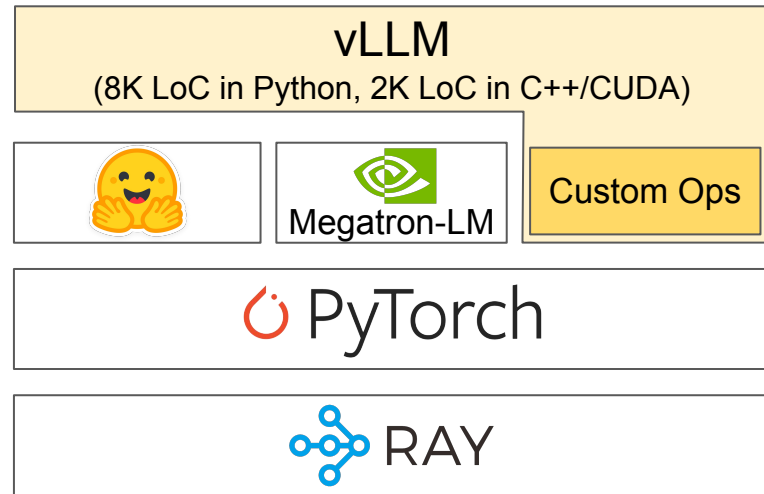
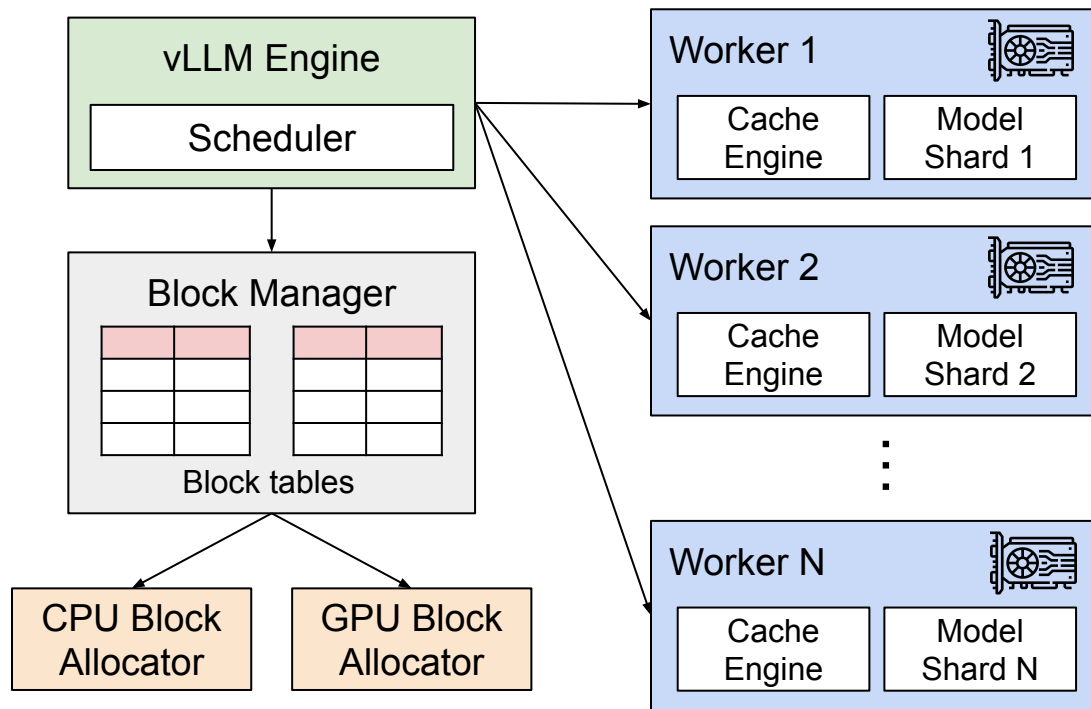


More complex sharing: beam search



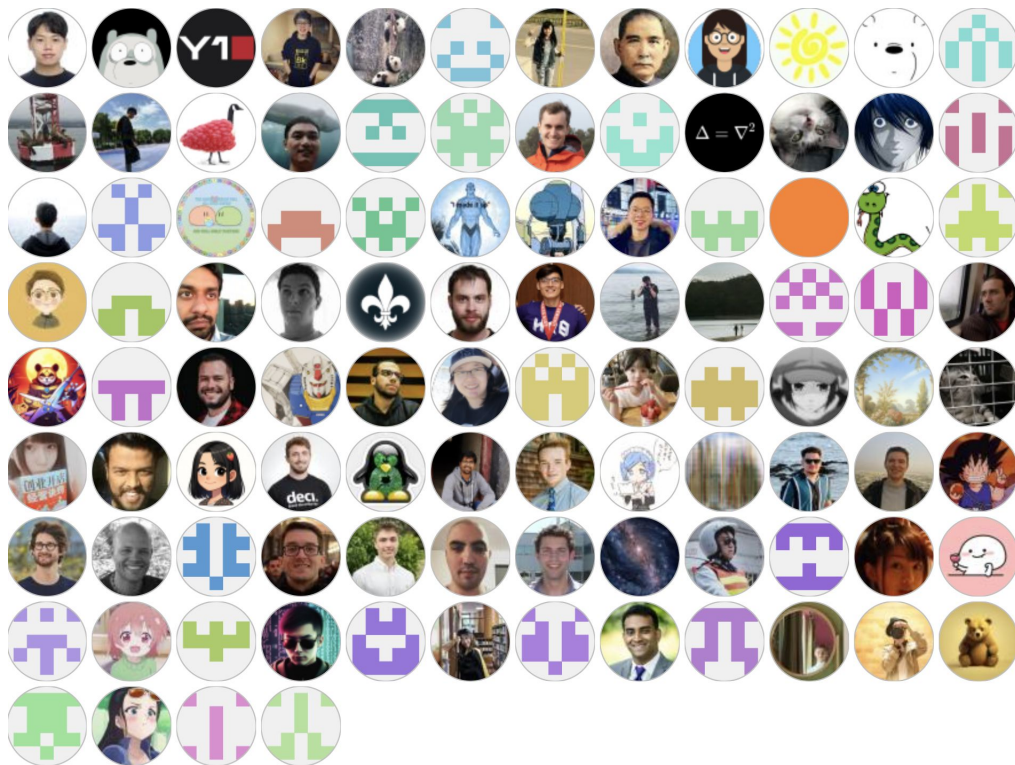
- Similar to process tree (fork & kill)
- Efficiently supported by dynamic block mapping and copy-on-write mechanism

System architecture & implementation



Contributors

Thanks to 190+ contributors who raised issues, participated in discussions, and submitted PRs!



Adopters



Open-Source Projects



Im-sys/FastChat



OPENCHAT



EleutherAI/Im-evaluation-harness



allenai/open-instruct



outlines-dev/outlines



SkyPilot

...

Companies



anyscale



aws



Google Cloud



Meta



NVIDIA



snowflake



Microsoft



Uber



databricks



Lepton AI



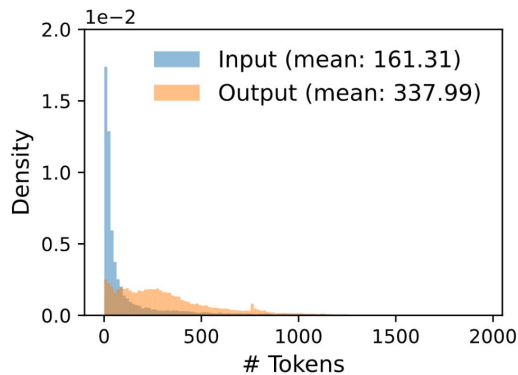
Modal

scale

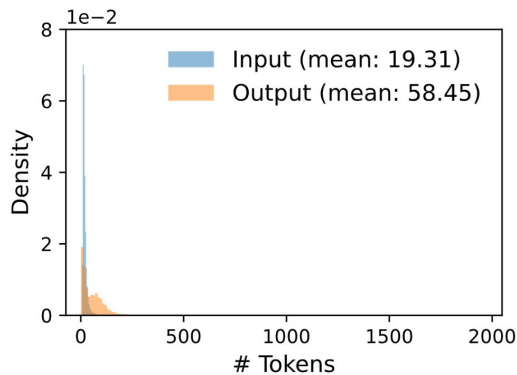
...

Experimental setup

- Metric: serving throughput
 - The maximum request rate that the system can serve without exploding the queue (latency)
- Model: OPT-13B, GPU: A100-40GB
- Trace: synthesized using Poisson arrival process
 - Datasets

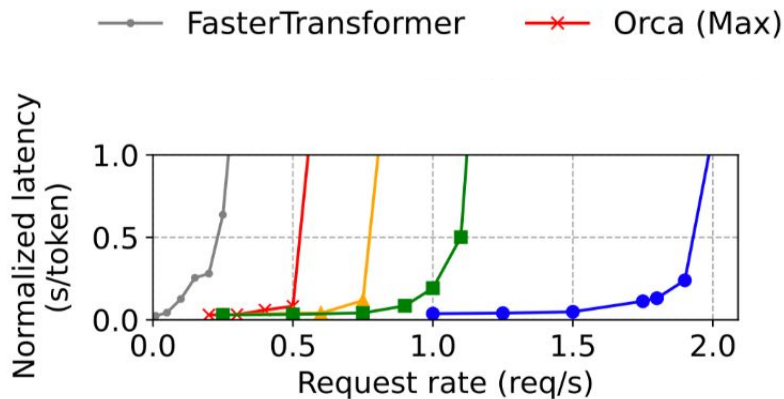


(a) ShareGPT

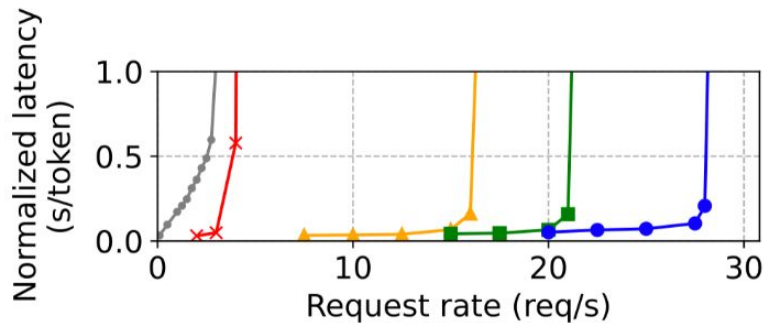


(b) Alpaca

Single sequence generation



(a) OPT-13B, 1 GPU, ShareGPT



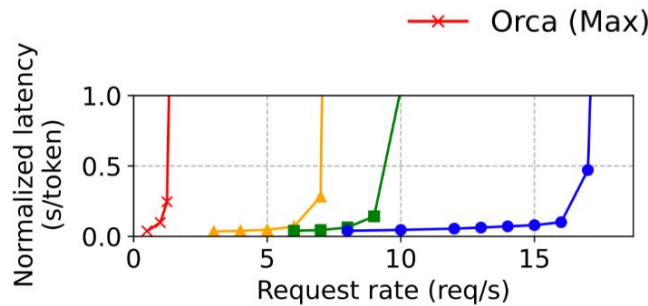
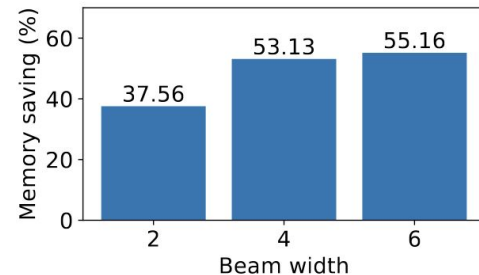
(d) OPT-13B, 1 GPU, Alpaca

Throughput bound by 100 ms per-token latency

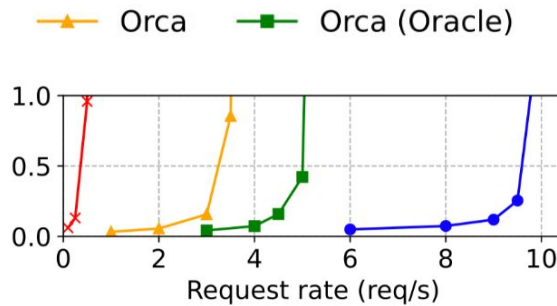
6.3x higher than FasterTransformer
2.5x higher than Orca

13x higher than FasterTransformer
1.7x higher than Orca

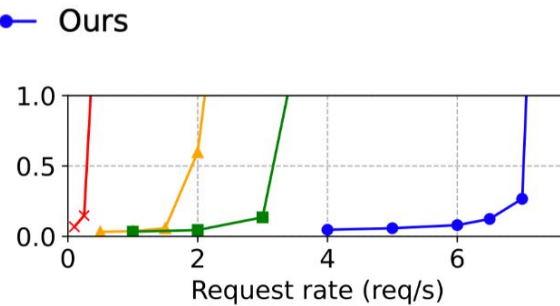
Generation with beam search



(d) beam search (beam width = 2)



(e) beam search (beam width = 4)



(f) beam search (beam width = 6)

Throughput bound by 100 ms per-token latency

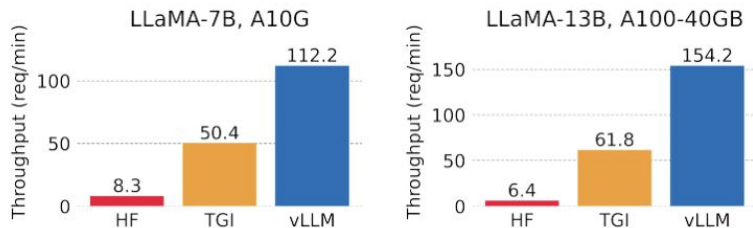
2.6x higher than Orca

3x higher than Orca

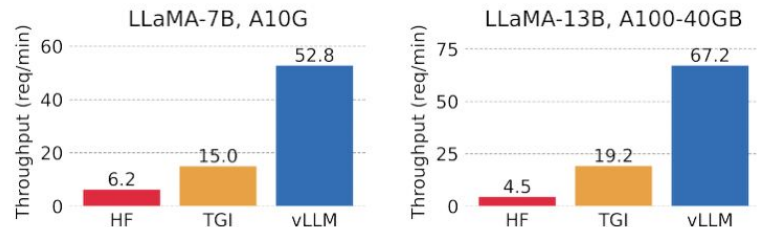
4x higher than Orca

Comparison with HuggingFace and TGI

- Up to **24x** higher throughput than HuggingFace (HF)
- Up to **3.5x** higher throughput than Text Generation Inference (TGI)



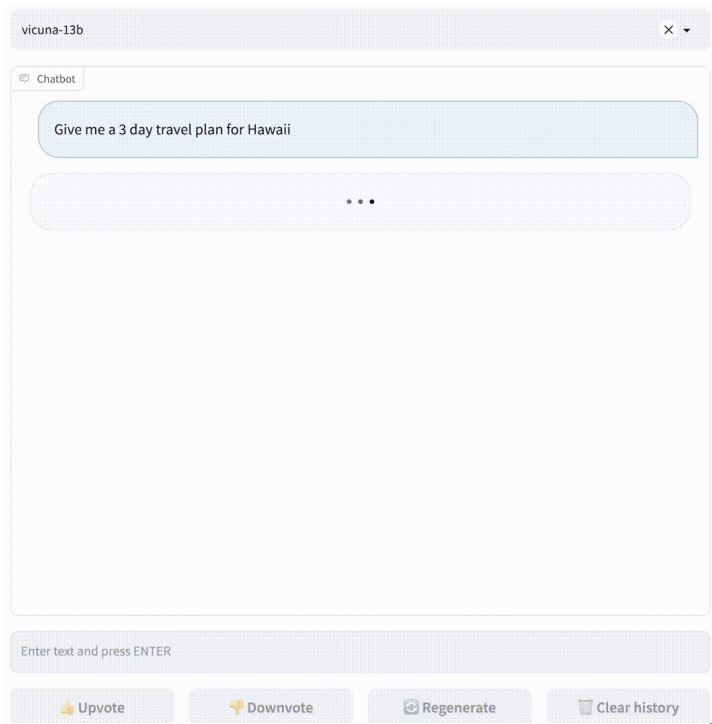
Serving throughput when each request asks for 1 output completion.



Serving throughput when each request asks for 3 output completions.

Powering Vicuna & Chatbot Arena

- Deployed for **3** months
 - Backend for LLaMA-based models (including Vicuna, Alpaca, Koala, etc.)
- Average traffic **40K** conversations/day
- Reduced the total number of GPUs by **50%** while serving **2x–3x** more requests/sec
 - The gain will be larger as vLLM is used for more models



Conclusion

- vLLM improves the memory efficiency of LLM serving by **2.5x–5x** by **reducing memory fragmentation** and **enabling sharing** token states
- vLLM outperforms SOTA by **1.7x–4x** in terms of **serving throughput**
- vLLM is more effective for large models, long sequences, and complex sampling methods

Key Idea

Reduce memory
fragmentation with
paging

Further reduce
memory usage with
sharing

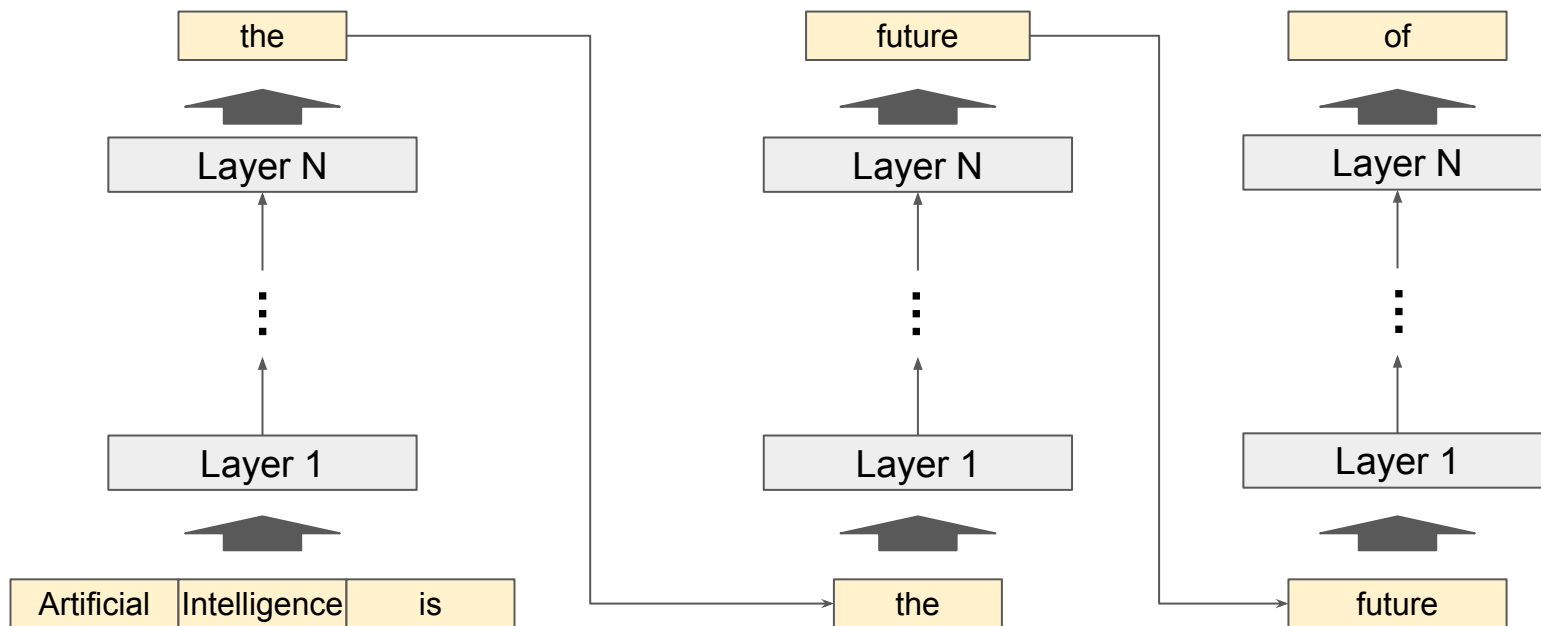
Behind Vicuna and Chatbot Arena: System Problems

Q1 Serving: How to efficiently serve large amount of LLM requests while minimizing cost-per-query?

Q2 Inference: How to perform LLM inference on a single request while maximizing performance?

Inference process of LLMs

Output



Input

Repeat until the sequence

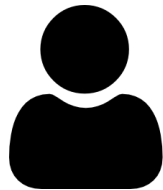
- Reaches its pre-defined maximum length (e.g., 2048 tokens)
- Generates certain tokens (e.g., "<|end of sequence|>")

The Problem is harder than Thought

Even if only one request (and the system is not busy), we still *cannot* do better

$$\text{Latency} = \text{step latency} * \# \text{ steps}$$

Can we do better?

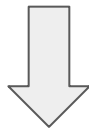


Inference: fewer request

Rethink Autoregressive Decoding

Autoregressive Decoding (Greedy): decoding m tokens

$$y_i = \arg \max p_{\theta}(y_i \mid \mathbf{y}_{1:i-1}, \mathbf{x})$$



$$\left\{ \begin{array}{l} y_1 = \arg \max p_{\theta}(y_1 \mid \mathbf{x}) \\ y_2 = \arg \max p_{\theta}(y_2 \mid y_1, \mathbf{x}) \\ \vdots \\ y_m = \arg \max p_{\theta}(y_m \mid \mathbf{y}_{1:m-1}, \mathbf{x}) \end{array} \right.$$

Rethink Autoregressive Decoding

$\mathbf{x}$: prompt, $\mathbf{y} = [y_1, y_2, \dots, y_m]$: m tokens to decode, $p(\mathbf{y}|\mathbf{x})$: LLM distribution

Define: $f(y_i, \mathbf{y}_{1:i-1}, \mathbf{x}) = y_i - \operatorname{argmax} p(y_i | \mathbf{y}_{1:i-1}, \mathbf{x})$

$$\begin{cases} y_1 = \operatorname{argmax} p(y_1 | \mathbf{x}) \\ y_2 = \operatorname{argmax} p(y_2 | y_1, \mathbf{x}) \\ \vdots \\ y_m = \operatorname{argmax} p(y_m | \mathbf{y}_{1:m-1}, \mathbf{x}) \end{cases}$$

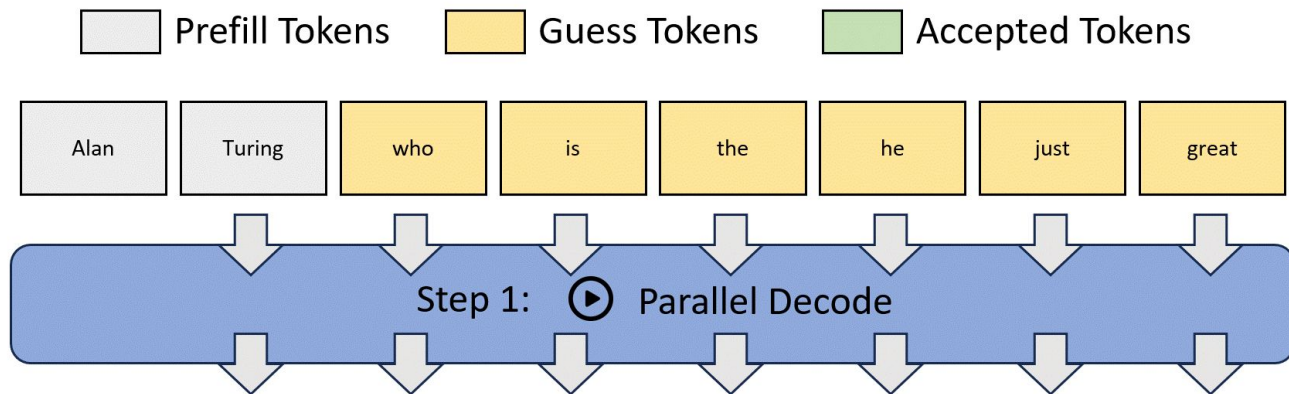
Autoregressive decoding

$\equiv$

$$\begin{cases} f(y_1, \mathbf{x}) = 0 \\ f(y_2, y_1, \mathbf{x}) = 0 \\ \vdots \\ f(y_m, \mathbf{y}_{1:m-1}, \mathbf{x}) = 0 \end{cases}$$

Non-linear system with
 m variables and m equations

Jacobi Decoding Illustrated



Total Steps: 0

Total Accepted Tokens: 0

Does Jacobi Iteration Really Work?

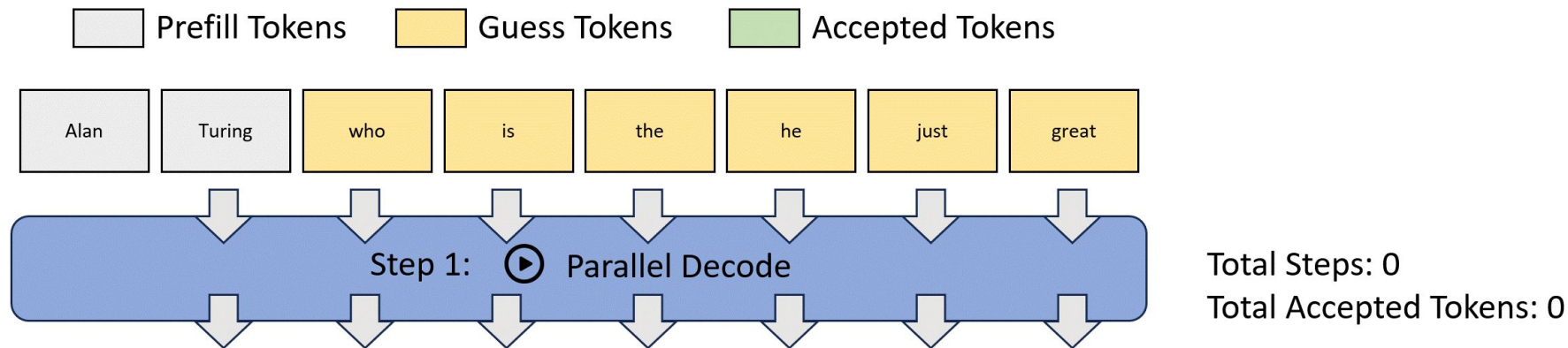
Pros:

- It can reduce decoding steps opportunistically
- Every Jacobi step has roughly the same speed as every decoding step (though slightly more cost on FLOPs)

Cons:

- However, it is hard to obtain wallclock speedups:
- The generated tokens are often put in the wrong positions of the seq.
- Correctly placed tokens are frequently replaced by subsequent Jacobi iterations.

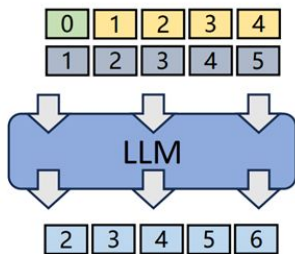
Observations: Many meaningful N-Grams are generated



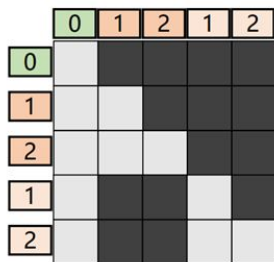
Solution: Lookahead Decoding

Lookahead/Verification Branch

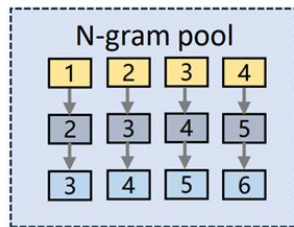
1: lookahead branch



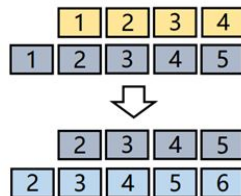
2: verification branch



3: collect n-grams



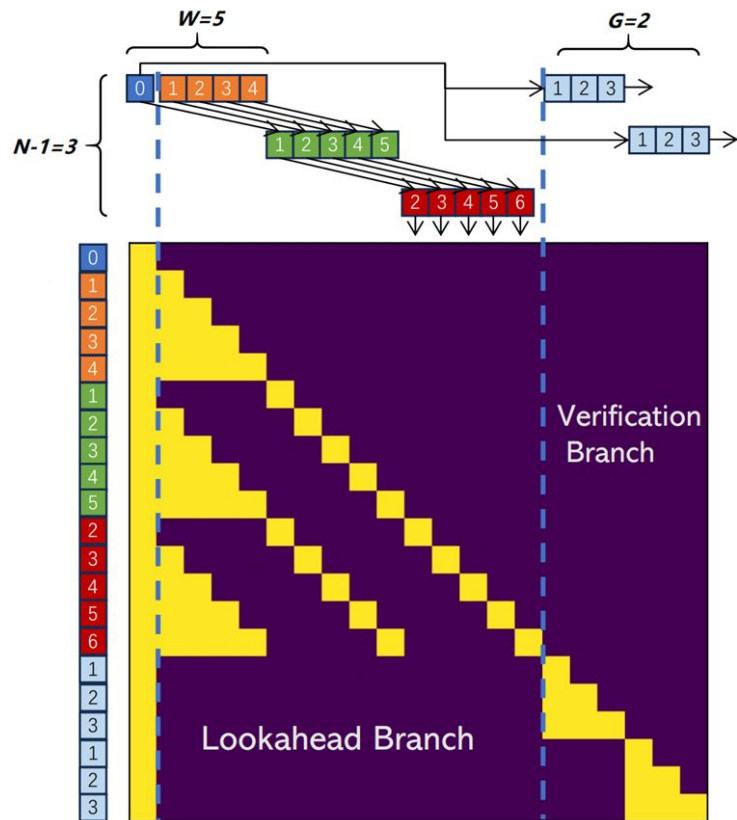
4: update lookahead branch



Input token (step t)
 Generated tokens
 Unwanted compute
 Tokens from $t-1$ steps
 Tokens from $t-2$ steps
 3-grams from pool
 n -gram pool
 Wanted compute

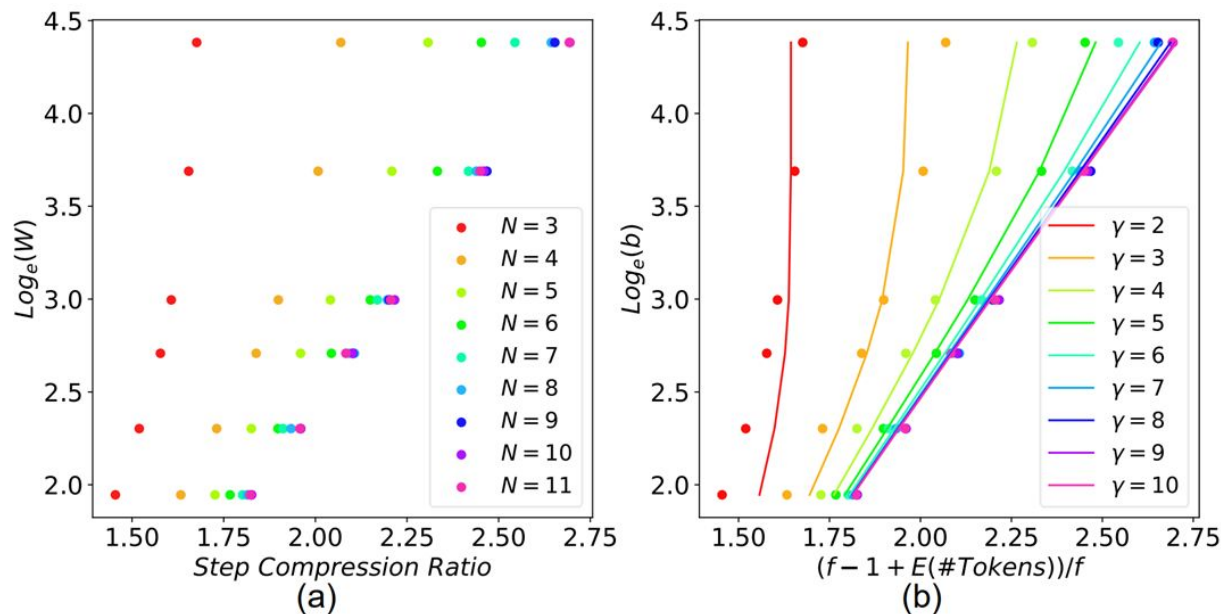
- A $W \times N$ lookahead branch to generate n-grams
- A verification branch of size G to verify n-grams from the pool

Implement Lookahead Decoding



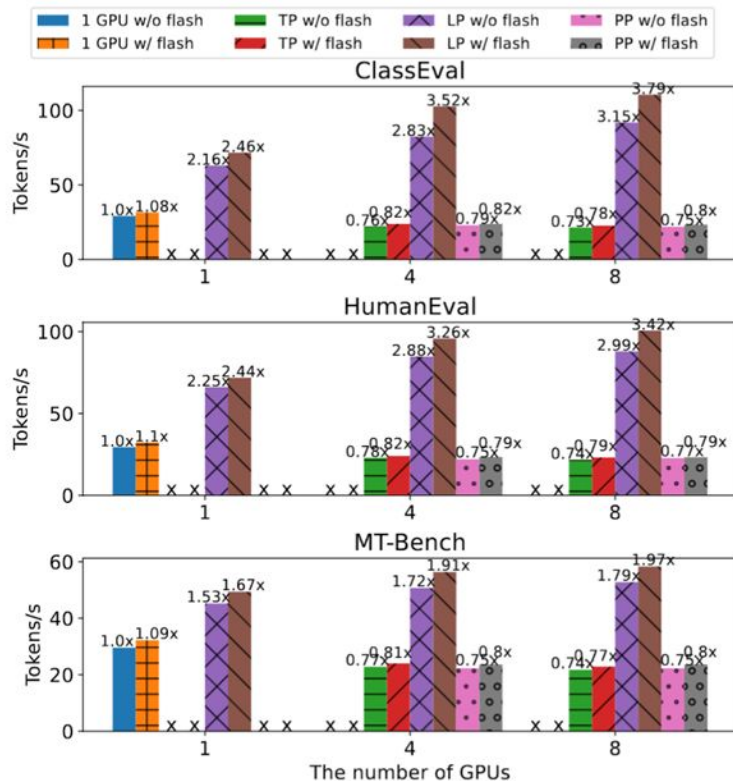
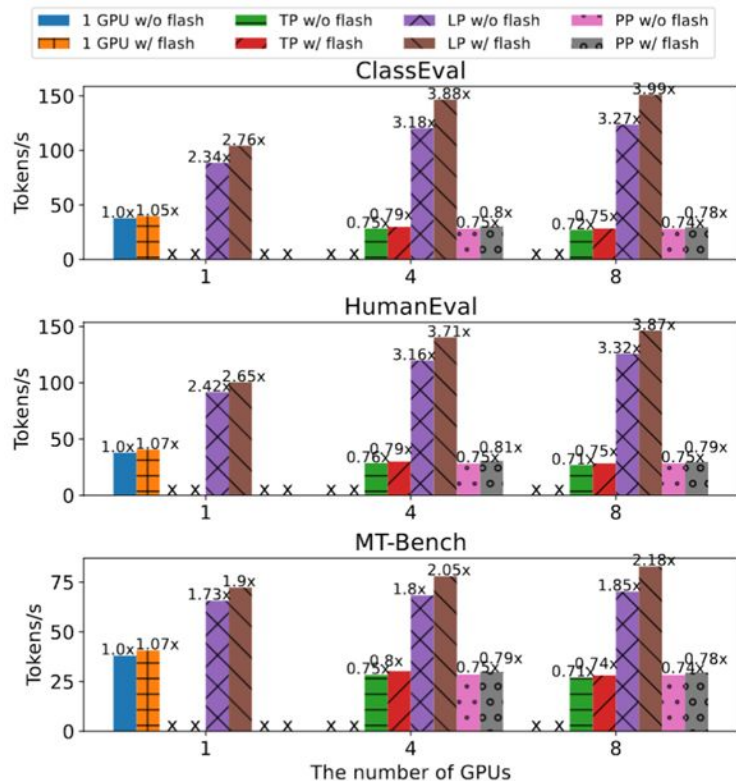
- Decoding progress is memory bandwidth bounded, which means the cost of decoding one token is similar to decoding several tokens.
- We can merge the lookahead and verification branches into one decoding step. And the cost should be similar to decoding only one token.
- Steps (largely $\downarrow$) x Per-Step Time (slightly $\uparrow$) = Wall Clock Time (largely $\downarrow$)

Lookahead Decoding Scaling Law



We can **exponentially** scale the computation budget (b) to **linearly** scale the *step compression ratio* (S) as in the figure(b).

Performance



Summary

Q1: How to efficiently serve large amount of LLM requests while minimizing cost-per-query?

A: vLLM with **Paged Attention** (Next).

Q2: How to perform LLM inference on a single request while maximizing performance?

A: Lookahead Decoding

LLM Inference Stack

The current stack:

- Continuous batching
- Paged attention
- Kernel optimization, quantization, etc.
- Speculative decoding, lookahead decoding, prompt lookup, etc.

Problems that will be solved In the next 6 - 12 month

- Better scheduling: prefill-decoding disaggregation, optimal scheduling
- Diverse outputs: structured output, API,

Prediction

- LLM Inference will be free (like google search) in 2025

Thanks

Browse my lab:

<https://cseweb.ucsd.edu/~haozhang/>

LLM inference and serving systems: [AlpaServe](#) [OSDI'23], [vLLM](#) [SOSP'23]

Model-parallel ML Systems: [Alpa](#) [OSDI'22, MLSys'23], [AMP](#) [NeurIPS'22], [Terapipe](#) [ICML'22]

Open-source LLMs and evaluation: [Vicuna](#), [LongChat](#), [MT-bench](#), [Chatbot Arena](#) [Preprint'23]

Secure inference: [MPCFormer](#) [ICLR'23]