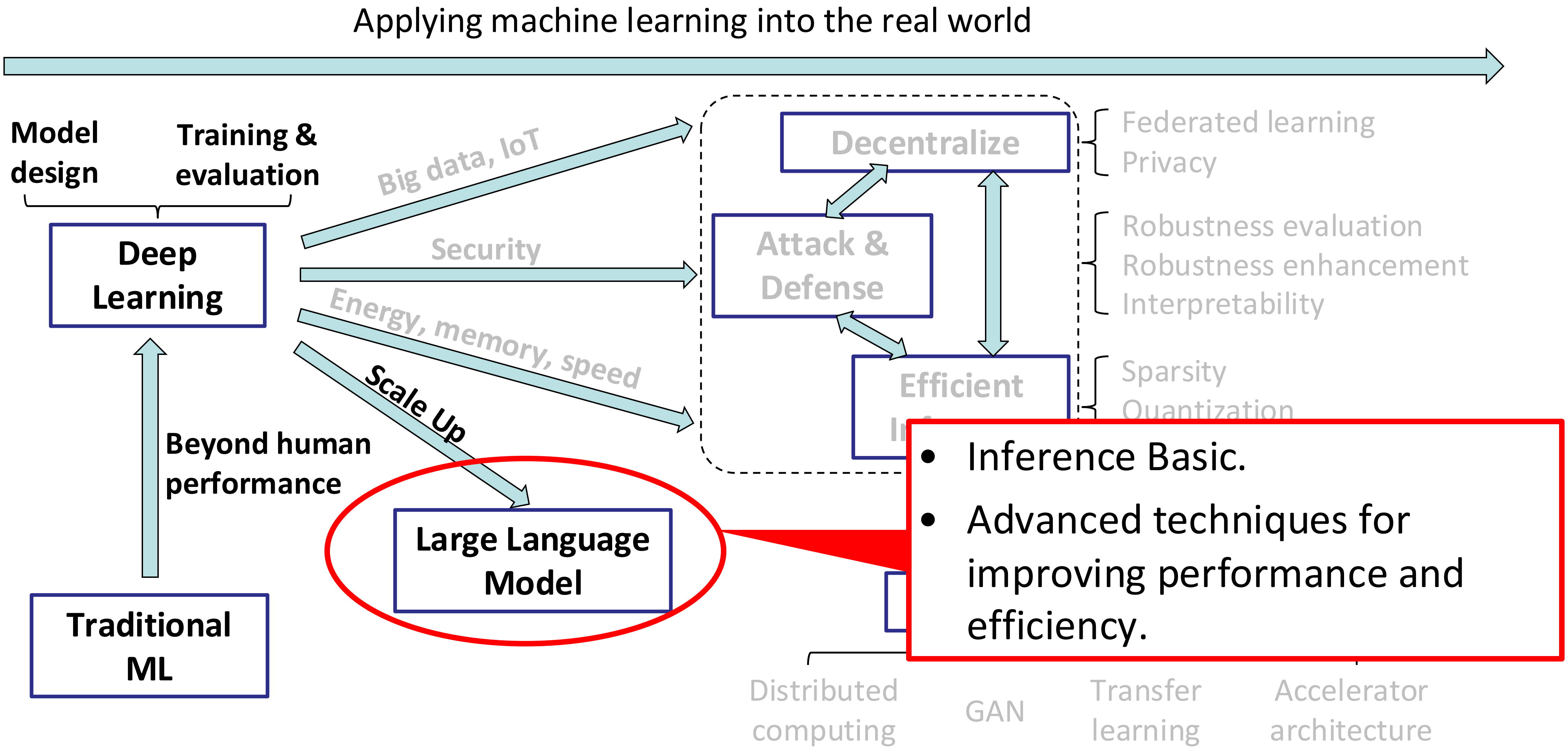
ECE 661 COMP ENG ML & DEEP NEURAL NETS

11. LARGE LANGUAGE MODELS INFERENCE

HAI "HELEN" LI, SPRING 2025

This lecture



- Inference Basic.
- Advanced techniques for improving performance and efficiency.

Outline

Lecture 11: Large Language Model Inference

- **Inference Basic**
- Additional Transformer Designs
  - KV cache
  - Attention mechanisms optimization
- Advanced Inference Systems
  - FlashAttention
  - vLLM
- LLM Inference Randomness

LLM Inference

- What is Inference in Machine Learning?
  - The process of inputting new data into a trained machine learning model to generate a prediction.
  - Example: Inputting an image into a CNN model to recognize its class

- What is Inference in context of LLMs?
  - The process of generating text outputs based on input prompts, by iteratively predicting the next token in a sequence.

LLM Inference Procedure

- Loading Weight to GPU
- Tokenizing the input text sequence (Prompt)
- Prefill Phase
- Decoding Phase

Key Phases

- Detokenize output tokens

Pierre Lienhart, Medium. https://medium.com/@plienhar/llm-inference-series-2-the-two-phase-process-behind-llms-responses-1ff1ff021cd5

LLM Inference Procedure

- Loading Weight to GPU
  - LLaMa-2-7B (FP32 ~ 28GB)

- Tokenizing the input text sequence (Prompt)
  - Tokenizer breaks down text into tokens (e.g word, subword, characters)
  - Tokens are converted into vectors that model can understand
  - Text -> tokens -> vector

What is LLM inference?

↓

↓

[3923, 374, 445, 11237, 45478, 30]

OpenAI. https://platform.openai.com/tokenizer

Tokenization

- Tokenization is the process of dividing text into smaller units called tokens, which are typically words or sub-words.
- Tokens are mapped to vectors for use in neural networks.

Tokenization is an important step.

⇩

Tokenization	is	an	important	step	.	[SEP]

Two Approaches :

- **Top-Down (Rule-based tokenization)** uses predefined rules to segment text into tokens, typically based on grammar and syntax, e.g., splitting sentences at punctuation marks or spaces.
- **Bottom-up (Subword tokenization)** breaks down words into smaller units, such as subwords or characters, allowing for the handling of unknown words and variations, e.g., Byte Pair Encoding used in BERT and GPT.

Byte-Pair Encoding

Byte Pair Encoding is a compression-based tokenization method that iteratively merges the most frequent character pairs to create subword units.

Step 1: Start with a vocabulary containing the individual characters present in the training corpus.

Step 2: Examine the training corpus and identify the two most frequently adjacent symbols.

Step 3: Add a new merged symbol representing the combined pair to the vocabulary. Replace every instance of the adjacent pair in the corpus with the new merged symbol.

Step 4: Continue counting and merging the most frequent pairs. Repeat until you've performed k merges, resulting in k novel tokens.

Step 5: The final vocabulary consists of the original set of characters plus the k new symbols created through merging.

https://web.stanford.edu/~jurafsky/slp3/ed3book.pdf

Byte-Pair Encoding

Initial vocabulary:
characters

↓

Split each word
into characters

Words in the data:

word	count
c a t	4
m a t	5
m a t s	2
m a t e	3
a t e	3
e a t	2

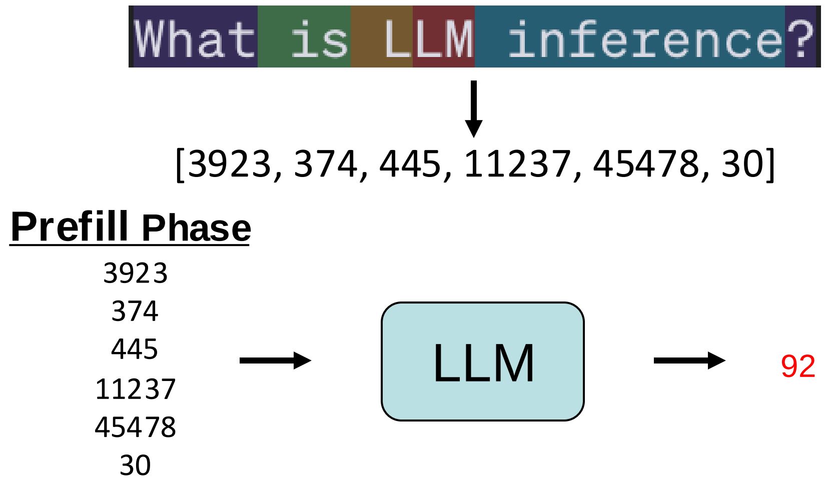
Current merge table:

(empty)

https://colab.research.google.com/github/catalyst-team/dl-course/blob/master/week-09/seminar_done.ipynb

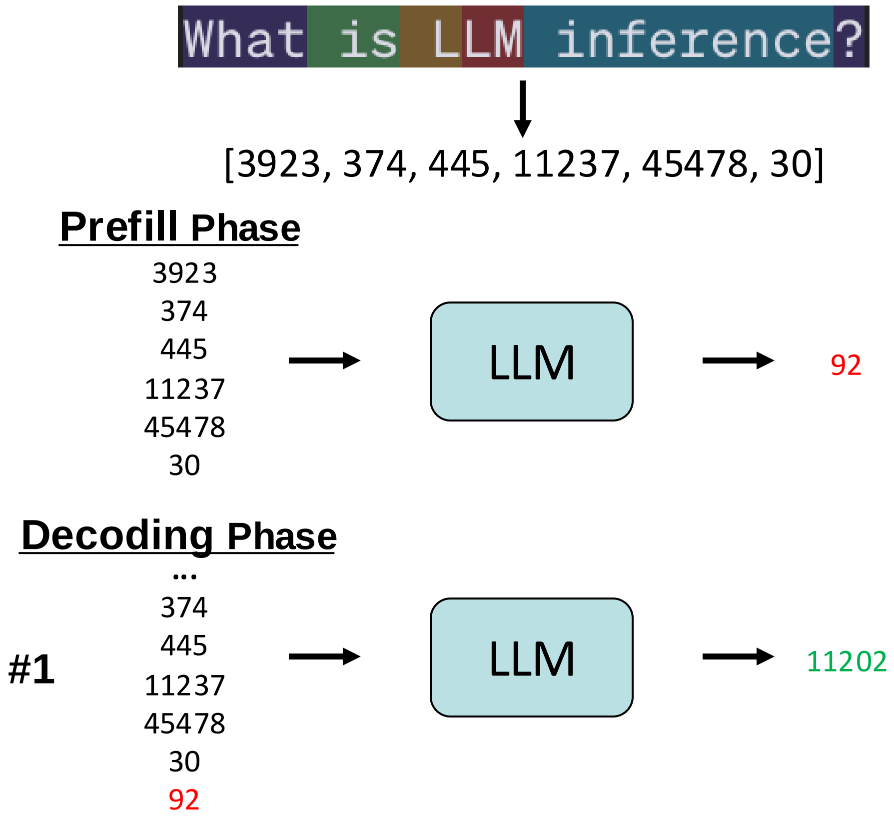
LLM Inference Procedure

- Prefill Phase (Single-step Phase)
  - Running the tokenized prompt through the LLM Model to generate the first token



LLM Inference Procedure

- Prefill Phase (Single-step Phase)
  - Running the tokenized prompt through the LLM Model to generate the first token
- Decoding Phase (Multi-step Phase)
  - Appending the generated token to the sequence of input tokens and using it as a new input to generate the next token

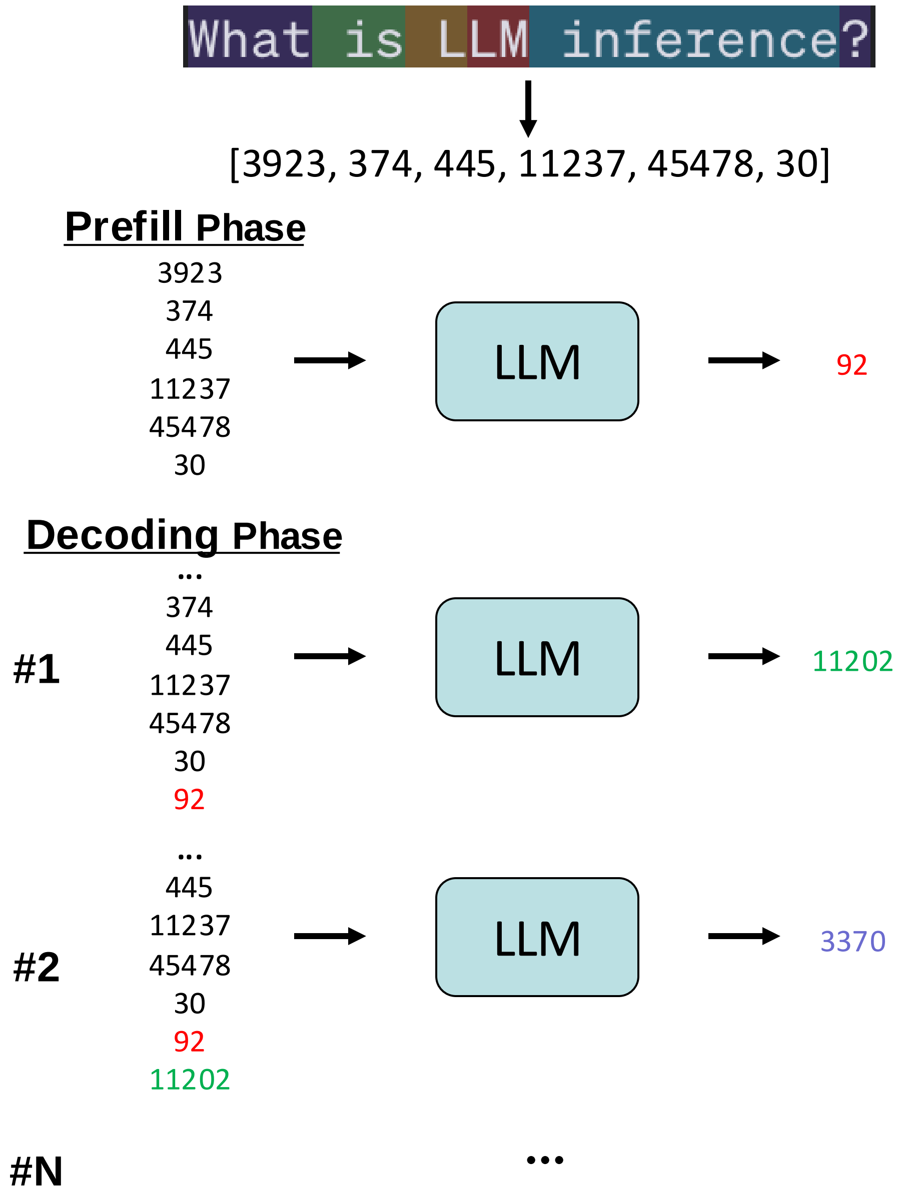


LLM Inference Procedure

- **Prefill Phase** (Single-step Phase)
  - Running the tokenized prompt through the LLM Model to generate the first token

- **Decoding Phase** (Multi-step Phase)
  - Appending the generated token to the sequence of input tokens and using it as a new input to generate the next token

Repeat decoding until meeting a stopping criteria
  - Generating end-of-sequence token
  - Reaching maximum sequence length



LLM Inference Scenarios

- **Inference** - Fewer request, offline traffic, latency

  Take a series of tokens as inputs, and generate subsequent tokens autoregressively until they meet a stopping criteria

  – Prefill Phase (Process the input)

  – Decoding Phase (Generate the output)

- **Serving** - Many requests, online traffic, cost-per-query
  – Co-locate the two phases and batch the computation of prefill and decoding across all users and requests

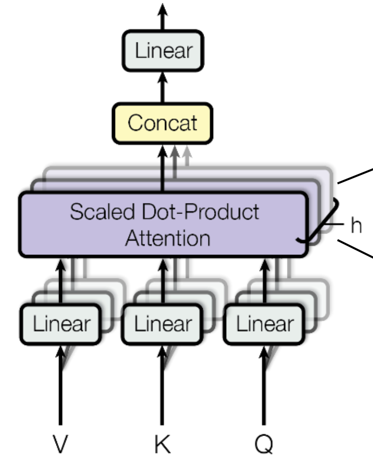
Outline

Lecture 11: Large Language Model Inference

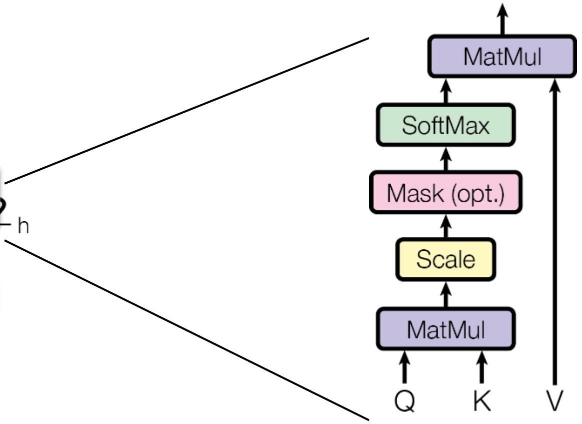
- Inference Basic
- Additional Transformer Designs
  - **KV cache**
  - Attention mechanisms optimization
- Advanced Inference Systems
  - FlashAttention
  - vLLM
- LLM Inference Randomness

KV Cache

- Recaps Attention Function



Multi-head attention



Scaled dot-product attention

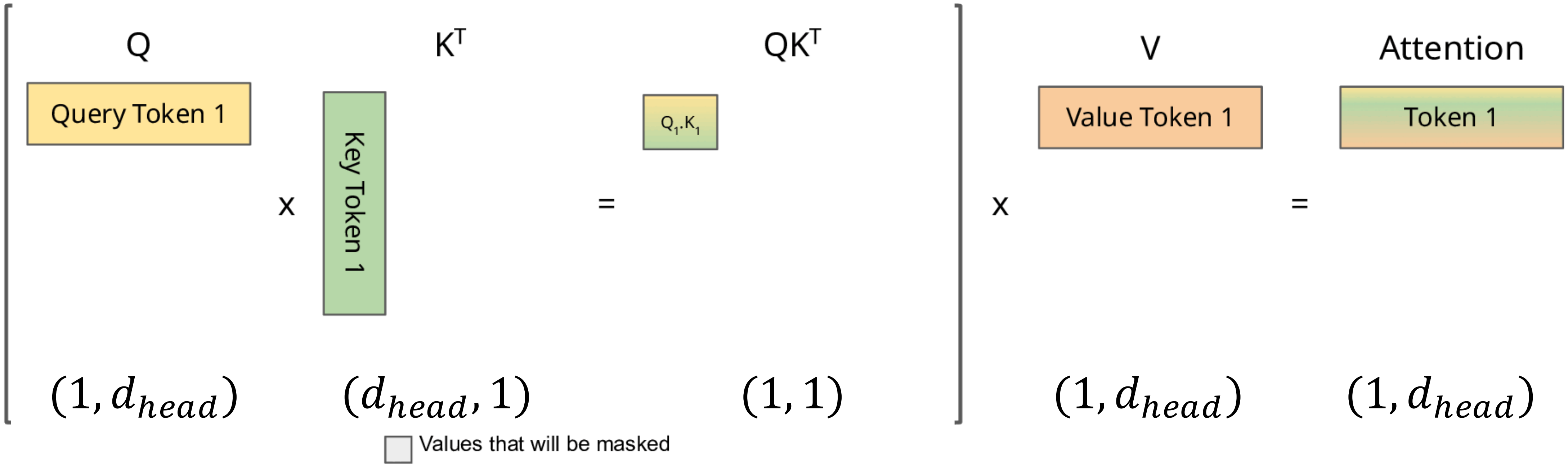
- Token generation (Attention computation)
  - Keys and Values of all preceding tokens
  - Query from current token
  - Recalculating previous tokens' attention

- Token generation only occurs in Decoder

KV cache

- Without Cache

Step 1



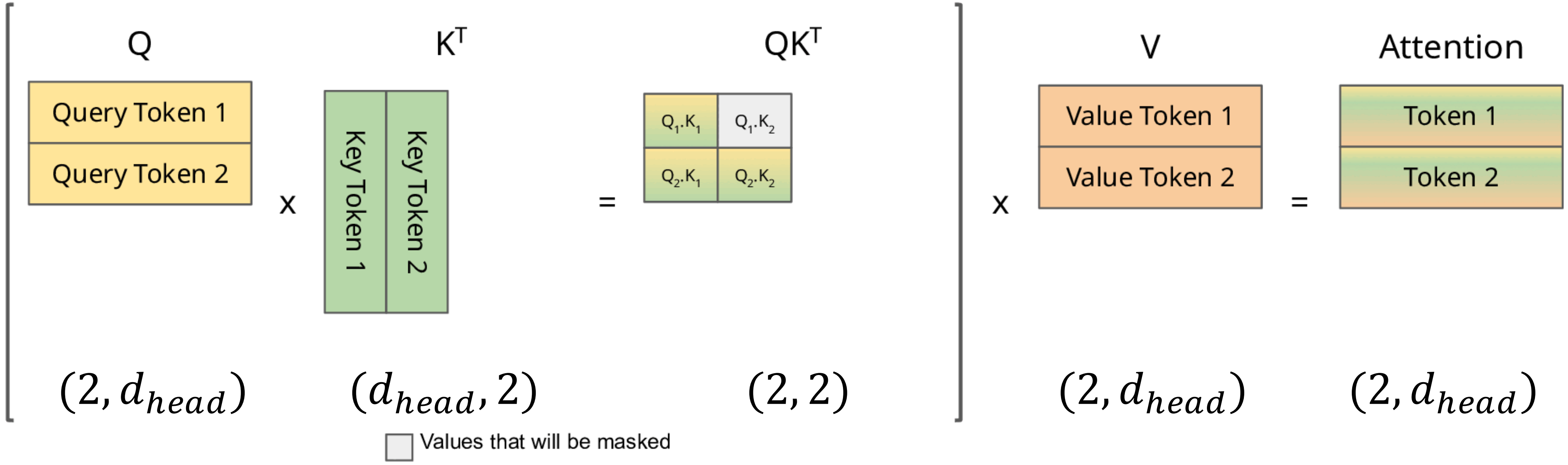
d_{head}: the hidden dimension of the attention head

Lages, J. (n.d.). Medium. https://medium.com/@joaolages/kv-caching-explained-276520203249

KV cache

- Without Cache

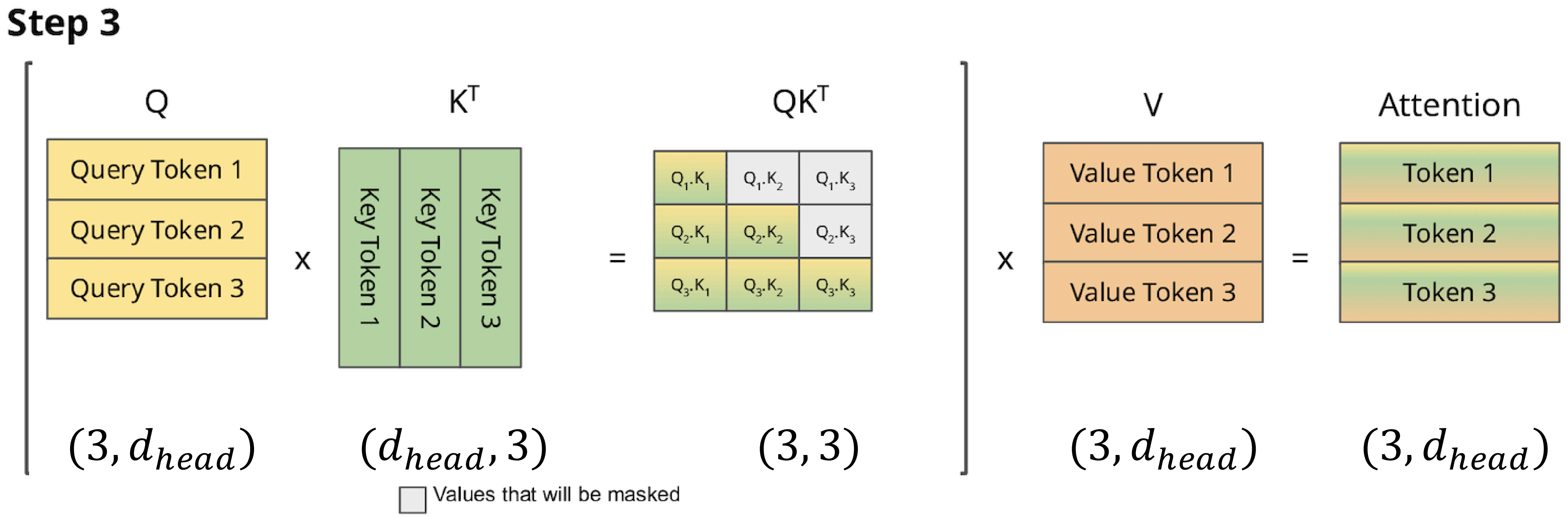
Step 2



Lages, J. (n.d.). Medium. https://medium.com/@joaolages/kv-caching-explained-276520203249

KV cache

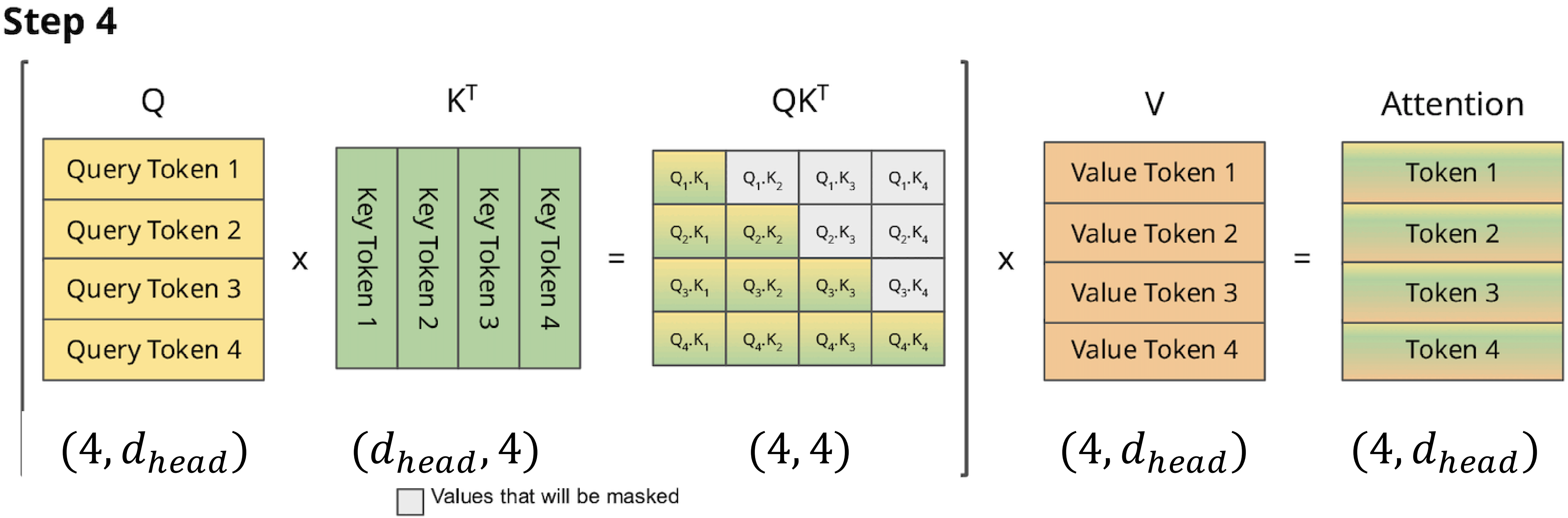
- Without Cache



Lages, J. (n.d.). Medium. https://medium.com/@joaolages/kv-caching-explained-276520203249

KV cache

- Without Cache



Lages, J. (n.d.). Medium. https://medium.com/@joaolages/kv-caching-explained-276520203249

KV cache

- Function
  - Storing previously calculated Keys and Values

- Benefits
  - Reducing the size of the matrices involved (compute attention only for new tokens.)
  - Leading to faster matrix multiplication and overall improved efficiency.

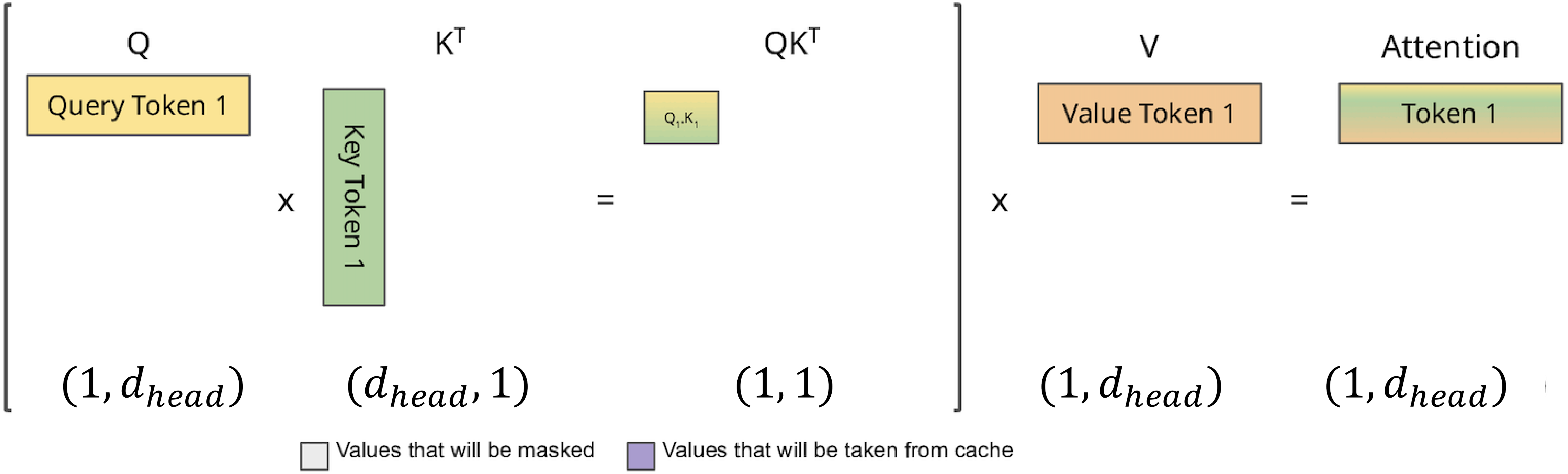
- Drawbacks
  - Requiring EXTRA memory to store the KV cache

Lages, J. (n.d.). Medium. https://medium.com/@joaolages/kv-caching-explained-276520203249

KV cache

- With Cache

Step 1

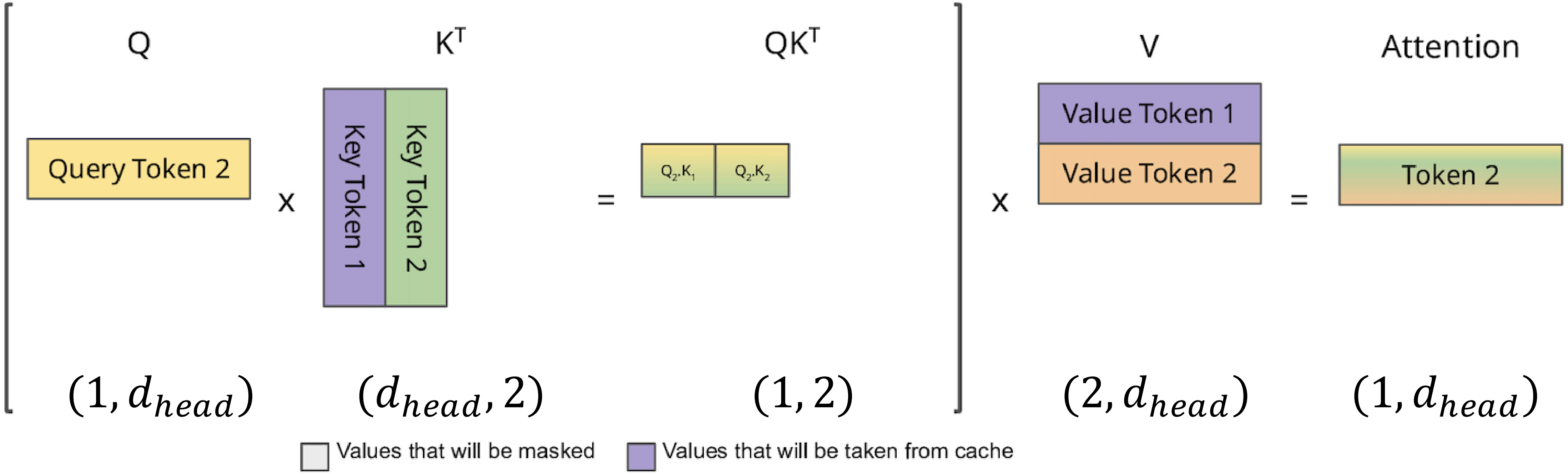


Lages, J. (n.d.). Medium. https://medium.com/@joaolages/kv-caching-explained-276520203249

KV cache

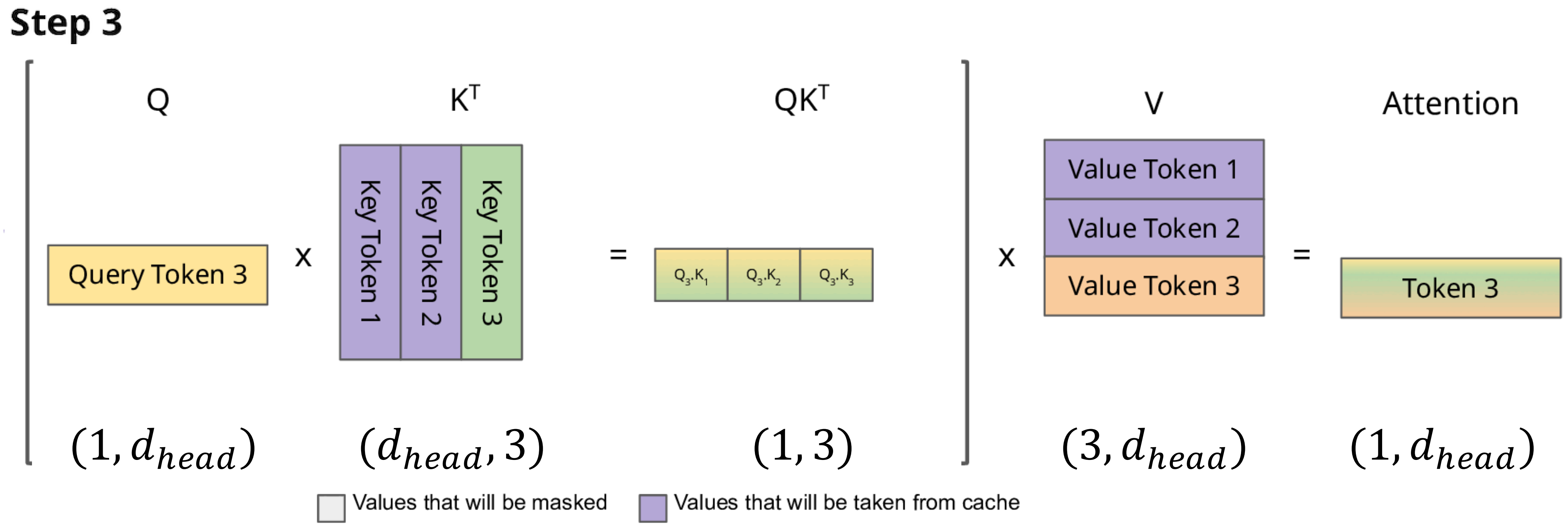
- With Cache

Step 2



KV cache

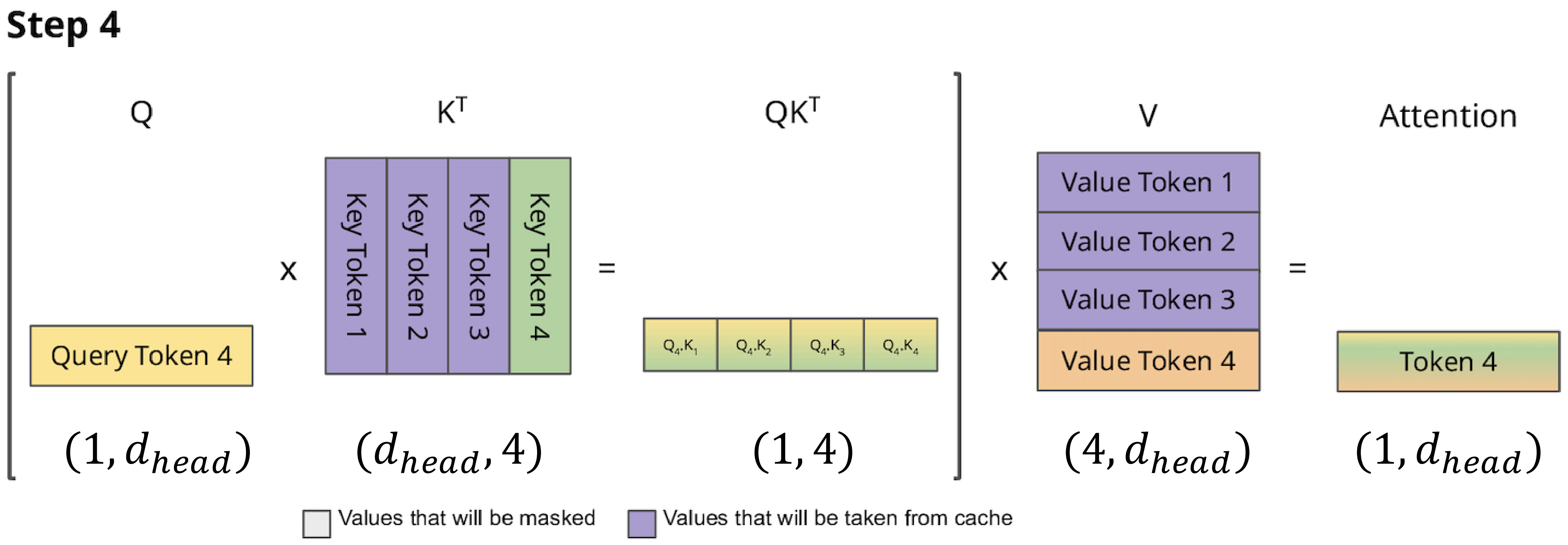
- With Cache



Lages, J. (n.d.). Medium. https://medium.com/@joaolages/kv-caching-explained-276520203249

KV cache

- With Cache



Lages, J. (n.d.). Medium. https://medium.com/@joaolages/kv-caching-explained-276520203249

Outline

Lecture 11: Large Language Model Inference

- Inference Basic
- Additional Transformer Designs
  - KV cache
  - **Attention mechanisms optimization**
- Advanced Inference Systems
  - FlashAttention
  - vLLM
- LLM Inference Randomness

KV cache Memory Usage

- How Big is KV cache?
  - Total size of KV cache (FP16 = 2 bytes):

$$Size_{KV} = 2 * n_{batch} * n_{seq} * n_{layers} * (n_{heads} * d_{head}) * 2(bytes)$$

    - n_{batch}: batch size,
    - n_{seq} : total sequence length
    - n_{layers}: the number of decoder attention layers,
    - n_{heads} : the number of attention heads per attention layer
    - d_{head}: the hidden dimension of the attention head
    - $(n_{heads} * d_{head})$: generally called embedding dimension *"d"*

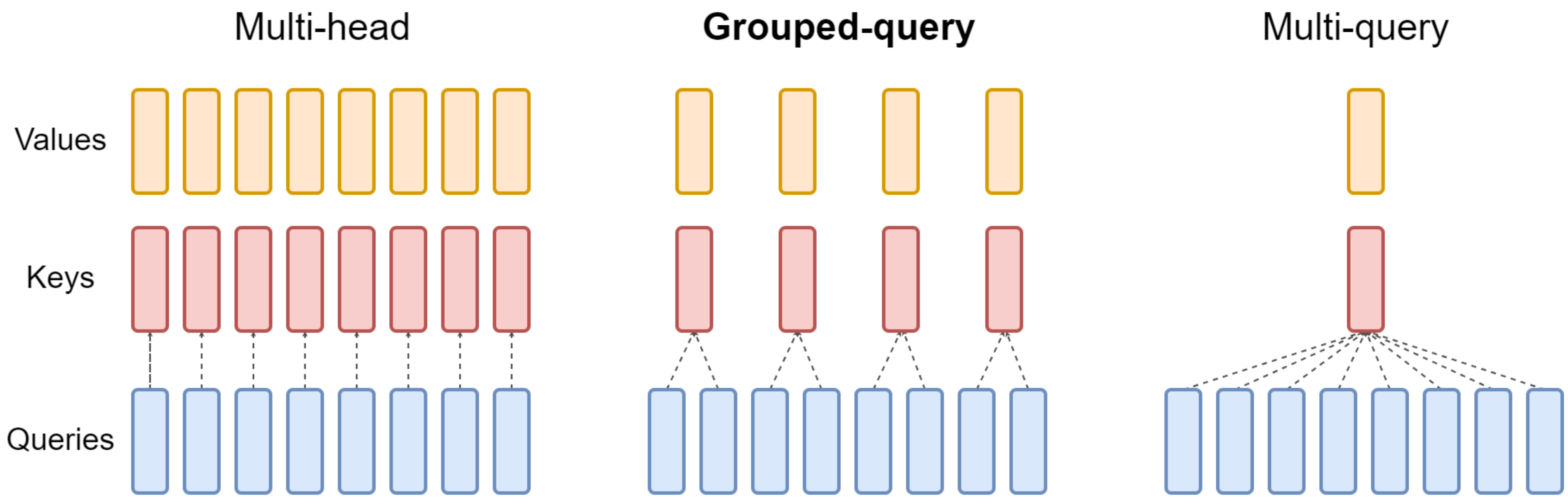
  - Llama-2-7B: $n_{layers} = 32, n_{heads} = 32, d_{head} = 128$
    - $n_{batch} = 1, \ n_{seq} = 100 \rightarrow Size_{KV} = 0.05$GB
    - $n_{batch} = 16, n_{seq} = 100 \rightarrow Size_{KV} = 0.8$GB
    - $n_{batch} = 16, n_{seq} = 10000 \rightarrow Size_{KV} = 80$GB

A100 GPU Memory = 80GB

Pierre Lienhart, Medium. https://medium.com/@plienhar/llm-inference-series-4-kv-caching-a-deeper-look-4ba9a77746c8

Attention mechanisms optimization

- Reduce KV Cache Memory Usage with n_{heads}



  - Multi-head Attention (MHA): N head for Query, Key, Value
  - Grouped-query attention (GQA): N head for Query, G head for Key and Value
  - Multi-query attention (MQA): N head for Query, 1 head for Key and Value

Ainslie, Joshua, and et, al. "Gqa: Training generalized multi-query transformer models from multi-head checkpoints." (2023).

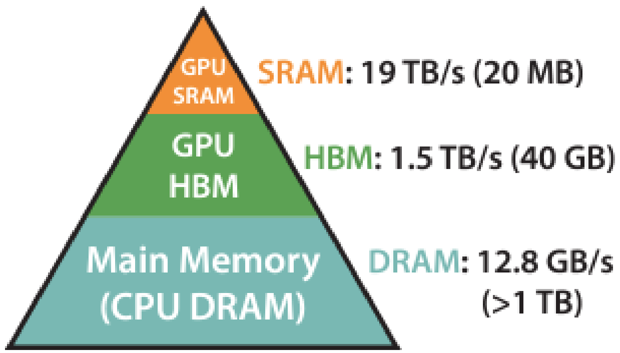
Outline

Lecture 11: Large Language Model Inference

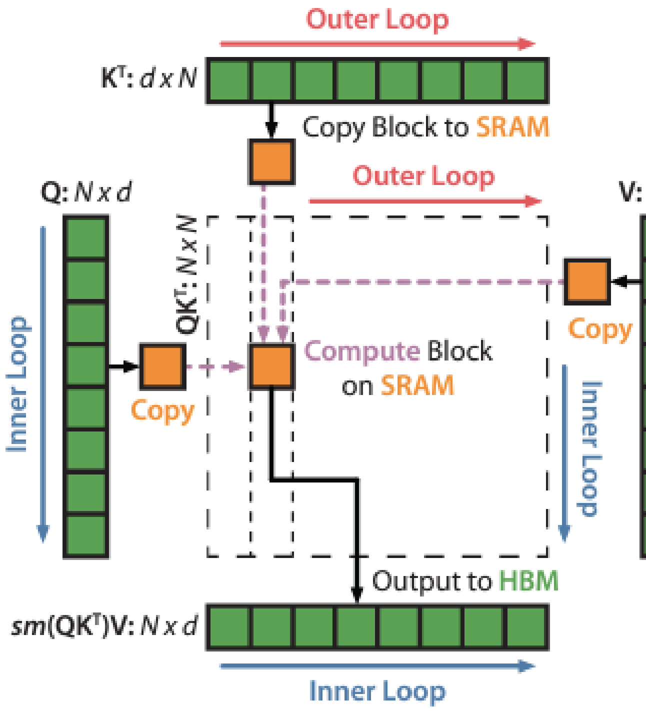
- Inference Basic
- Additional Transformer Designs
  - KV cache
  - Attention mechanisms optimization
- Advanced Inference Systems
  - **FlashAttention**
  - vLLM
- LLM Inference Randomness

FlashAttention

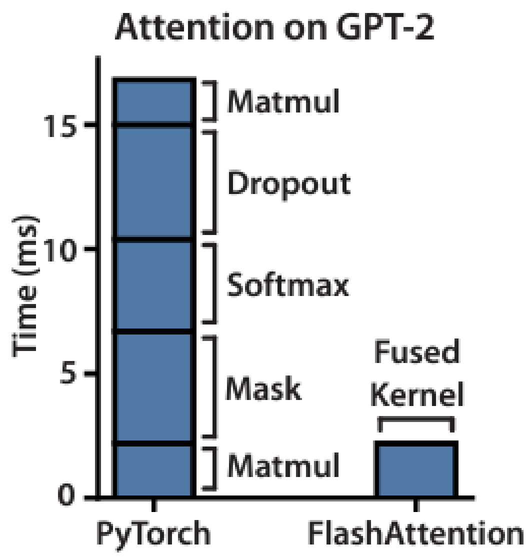
- Uses tiling to **reduce the number of memory reads/writes** between GPU high bandwidth memory (HBM) and GPU on-chip SRAM



Memory Hierarchy with Bandwidth & Memory Size



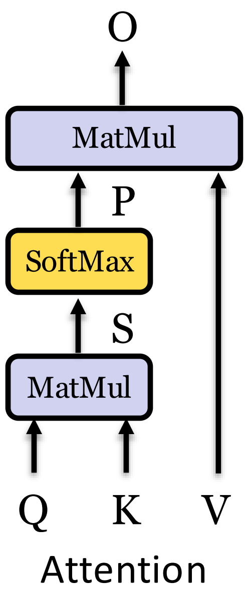
FlashAttention



Dao, Tri, et al. "Flashattention: Fast and memory-efficient exact attention with io-awareness."

Attention Memory Access

- Input stage
  - Load ***Q, K*** $(n_{seq} * d)$ from High Bandwidth Memory (HBM)
  - Calculate and write back ***S*** $(n_{seq} * n_{seq})$ to HBM
- Intermediate stage
  - Load ***S*** $(n_{seq} * n_{seq})$ from HBM
  - Calculate and write back ***P*** $(n_{seq} * n_{seq})$ to HBM.
- Output stage
  - Load ***P*** $(n_{seq} * n_{seq})$ and ***V*** $(n_{seq} * d)$ from HBM,
  - Calculate and write back ***O*** $(n_{seq} * d)$ to HBM



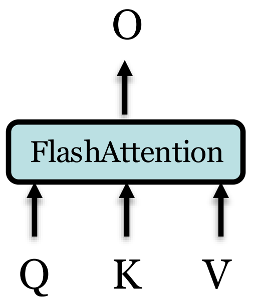
- HBM Memory Access Complexity
  - $\Theta(n_{seq} * d + n_{seq} * n_{seq})$

Dao, Tri, et al. "Flashattention: Fast and memory-efficient exact attention with io-awareness."

FlashAttention Memory Access

- Key Idea
  - Breaking down the large attention matrix into smaller sub-matrices (tiles).
  - Tile fits within SRAM (access faster than HBM)
  - **Significantly reducing the need to access HBM during computation**

- Breakdown of FlashAttention
  - Load a tile (part of ***Q, K, V***) from HBM into SRAM **(size: M)**
  - Perform all operations for the given tile in SRAM
  - Eliminate the need to load and write back of ***S*** and ***P*** to HBM
  - Write ***O*** back to HBM once the computation is complete



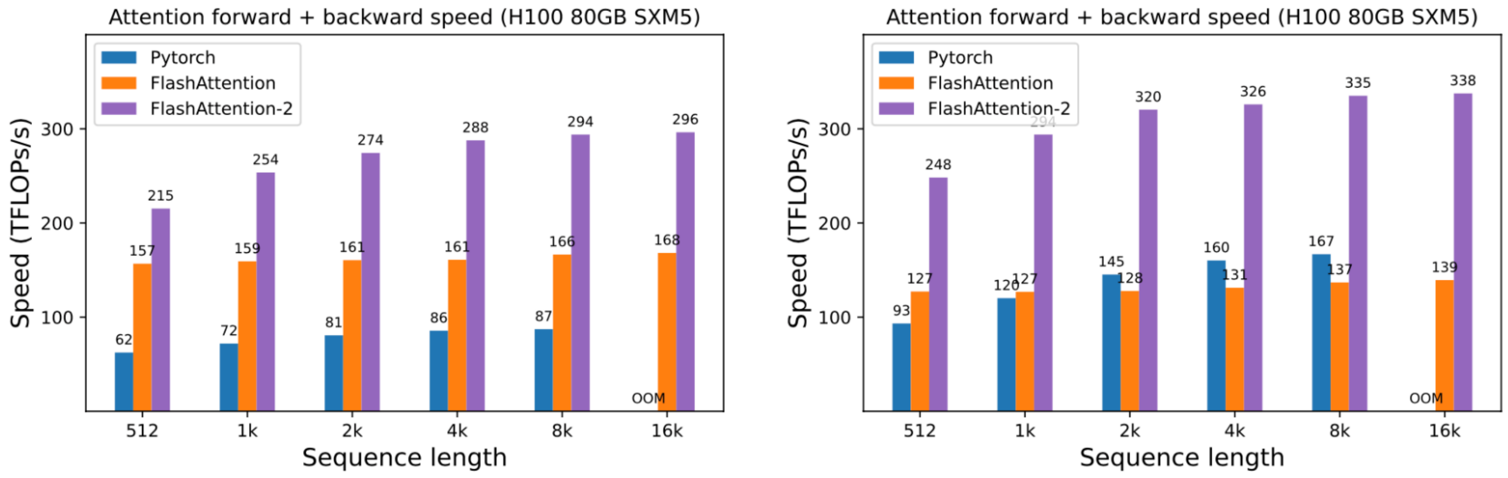
Flash attention

- HBM Memory Access Complexity
  - $\Theta(\frac{{n_{seq}}^2 * d^2}{M})$

Dao, Tri, et al. "Flashattention: Fast and memory-efficient exact attention with io-awareness."

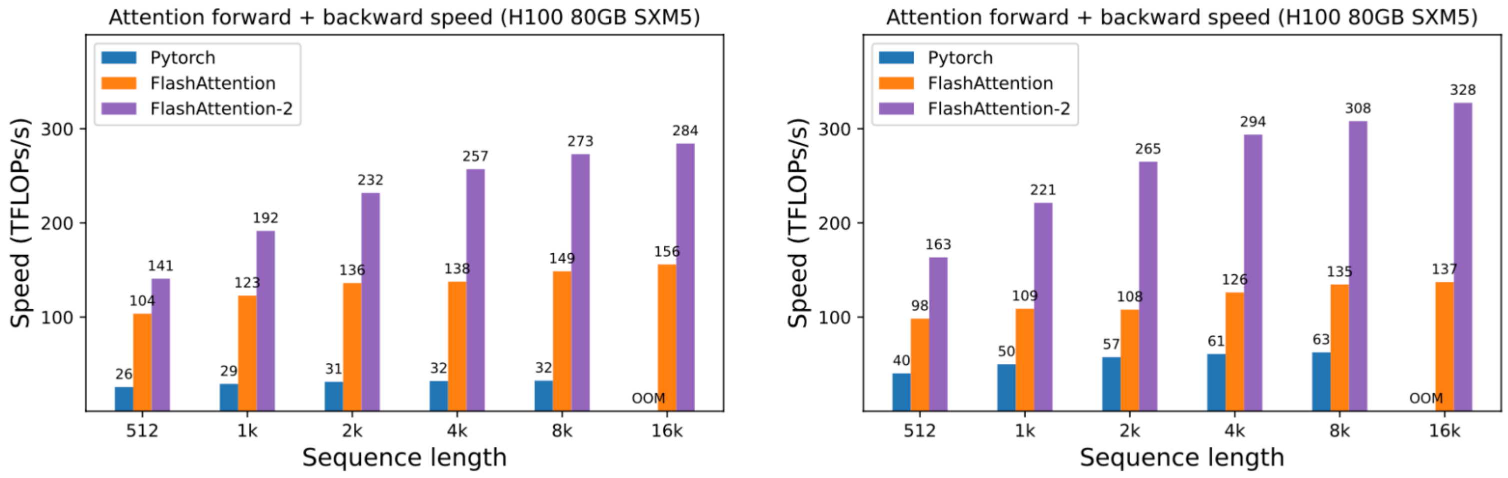
FlashAttention

- Results on NVDIA H100



(a) Without causal mask, head dimension 64

(b) Without causal mask, head dimension 128



Dao, Tri, et al. "Flashattention: Fast and memory-efficient exact attention with io-awareness."

Outline

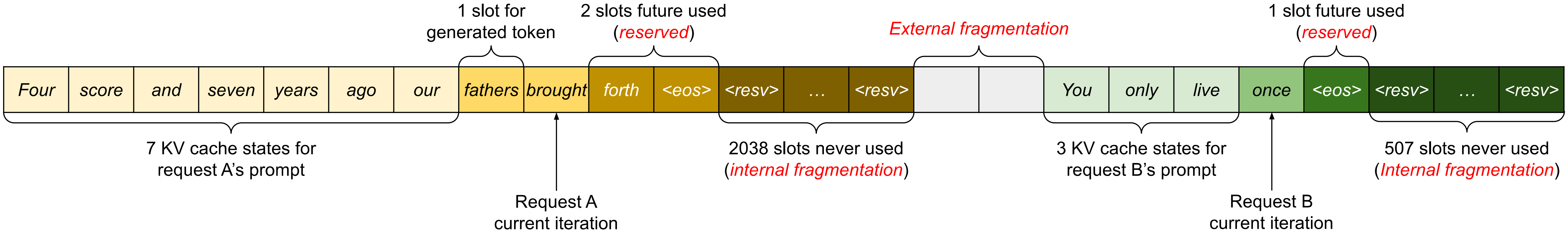
Lecture 11: Large Language Model Inference

- Inference Basic
- Additional Transformer Designs
  - KV cache
  - Attention mechanisms optimization
- Advanced Inference Systems
  - FlashAttention
  - **vLLM**
- LLM Inference Randomness

vLLM

- A high-throughput and memory-efficient inference and serving engine for LLMs

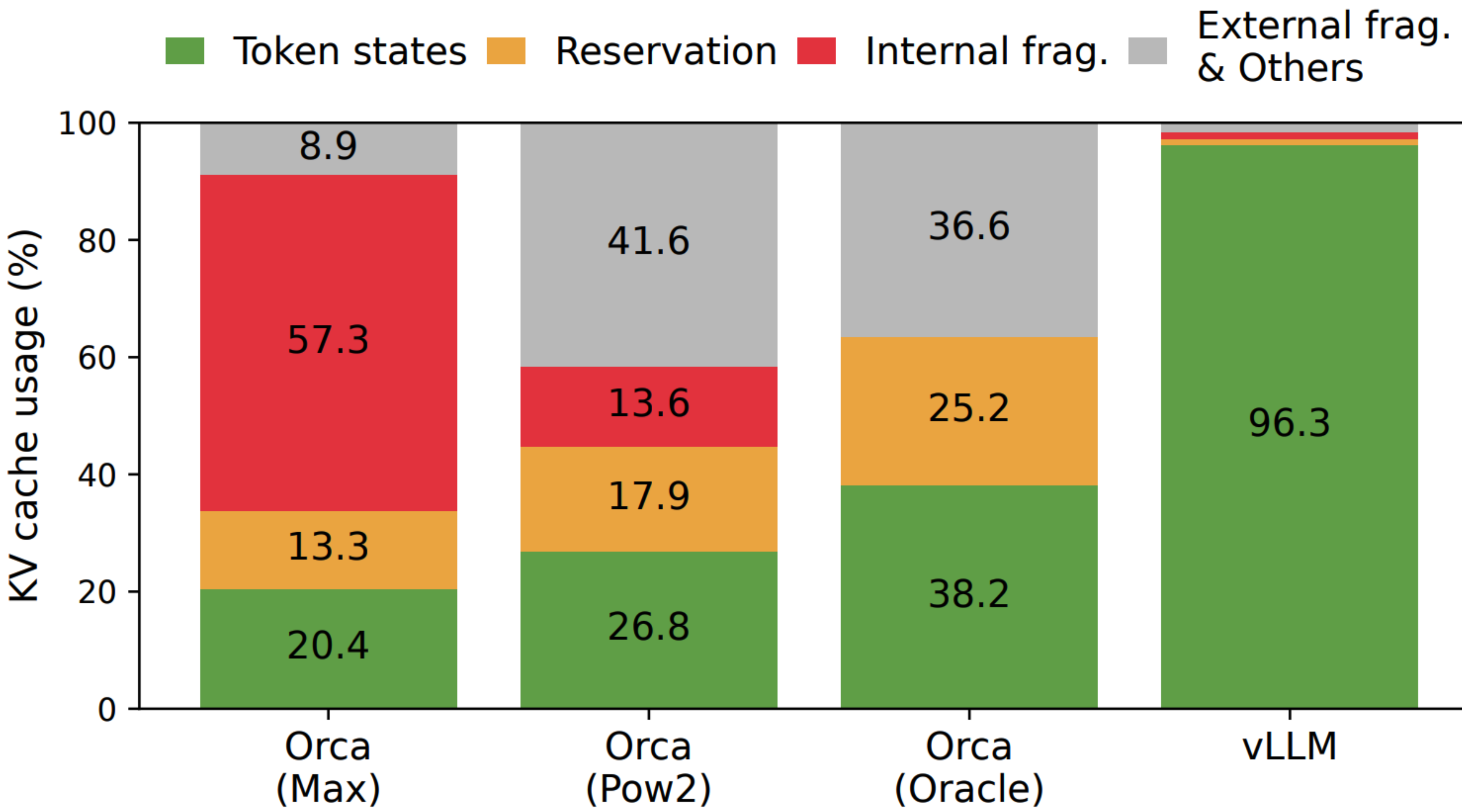
- Motivation
  - KV cache Memory Usage problem



  - **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.

Woosuk, Kwon, et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention."

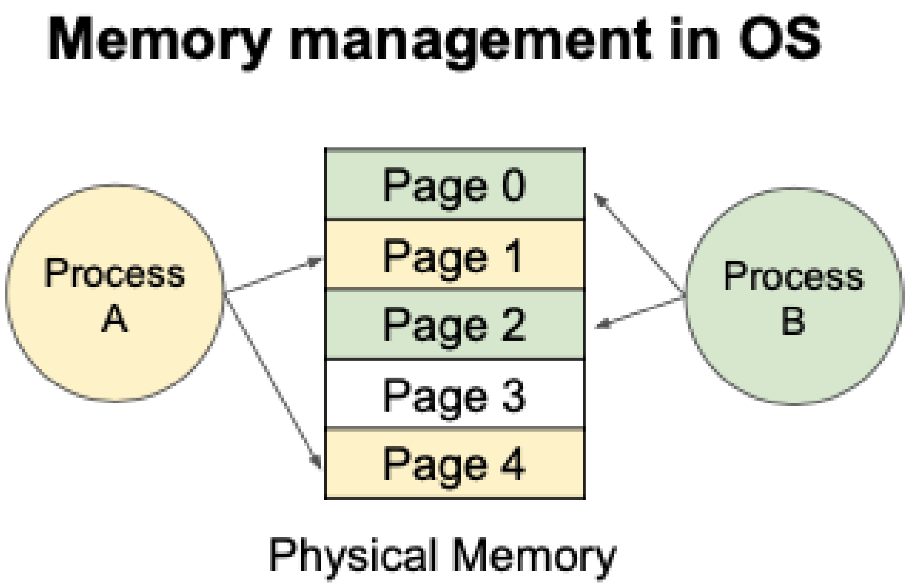
Memory Wastes in LLM KV Cache

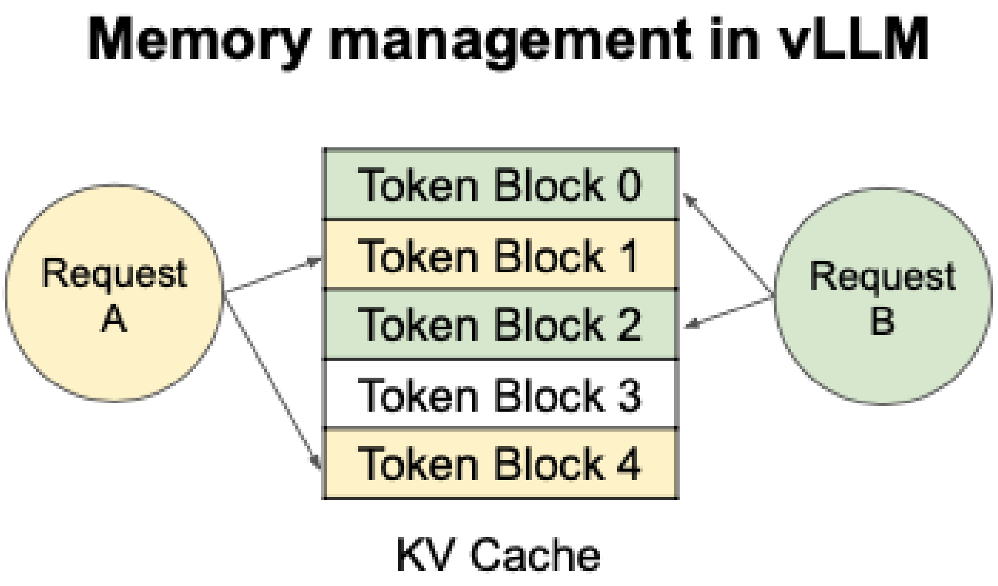


Woosuk, Kwon, et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention."
PagedAttention, Hopsworks, https://www.hopsworks.ai/dictionary/pagedattention

vLLM Analogy

- Inspired by Virtual Memory Management in Operating system
- Key Algorithm: **PagedAttention**

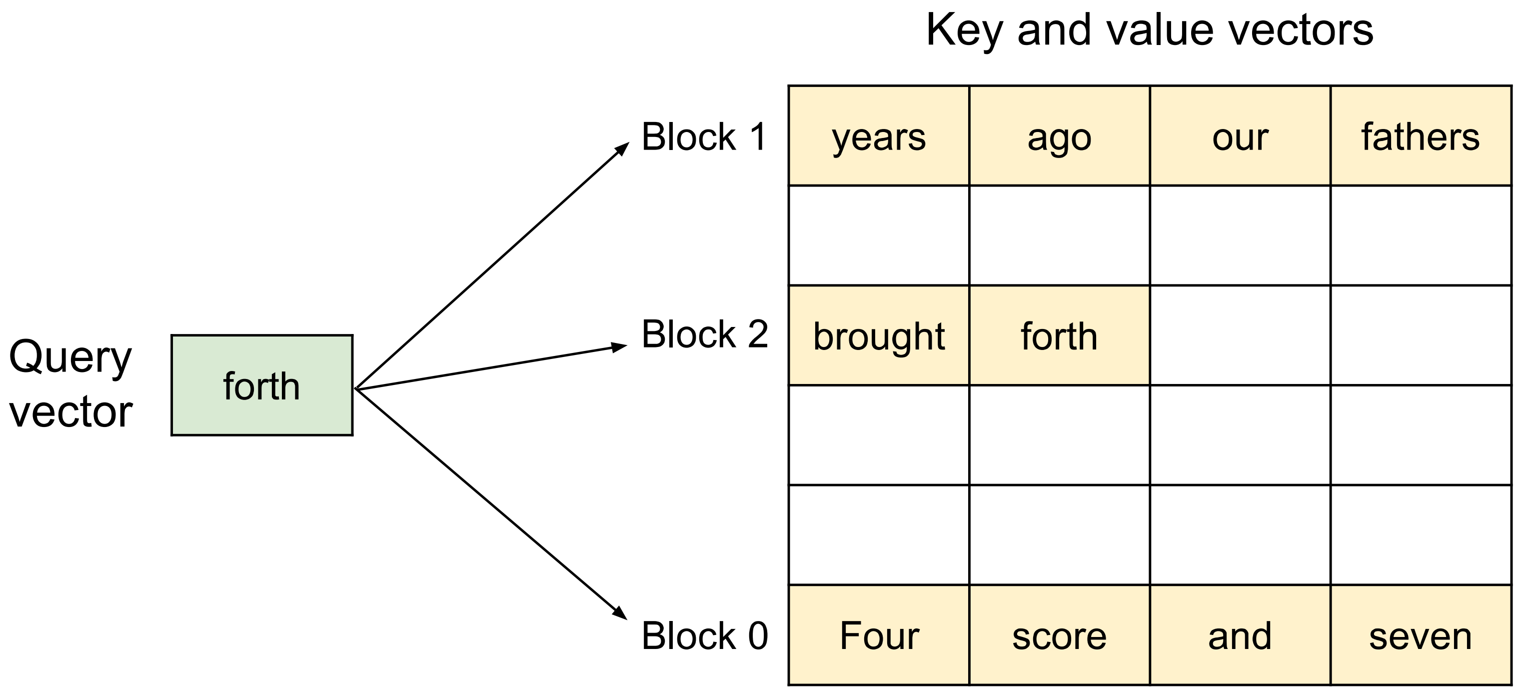




Woosuk, Kwon, et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention."

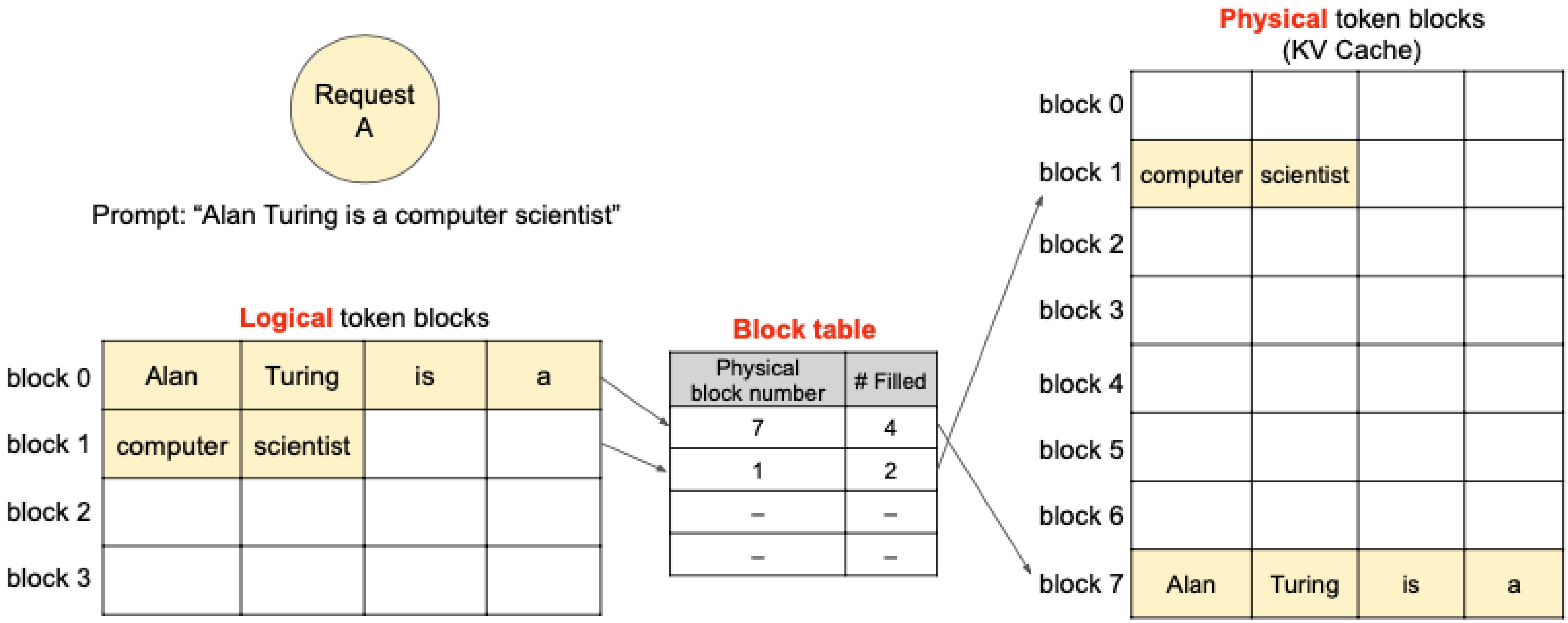
PagedAttention

- Storing Continuous Keys and Values in non-contiguous memory space



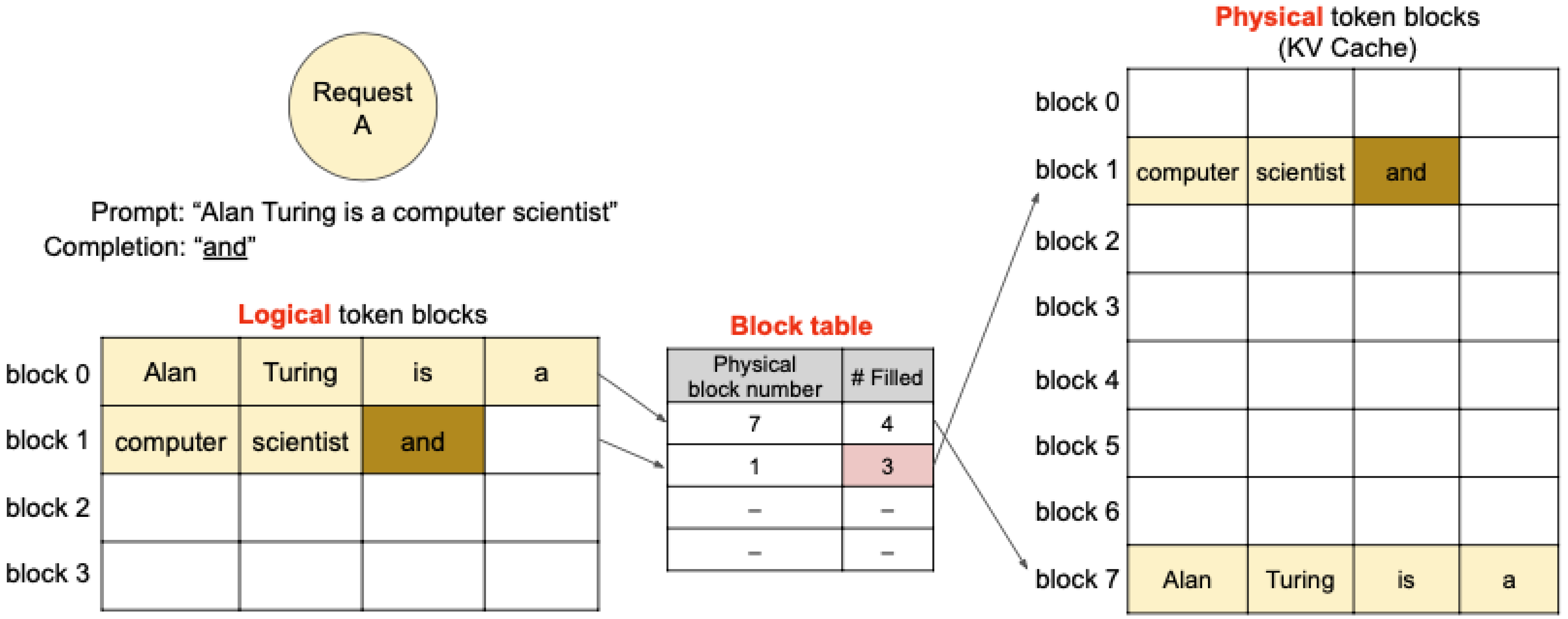
Woosuk, Kwon, et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention."

Logical & Physical KV blocks



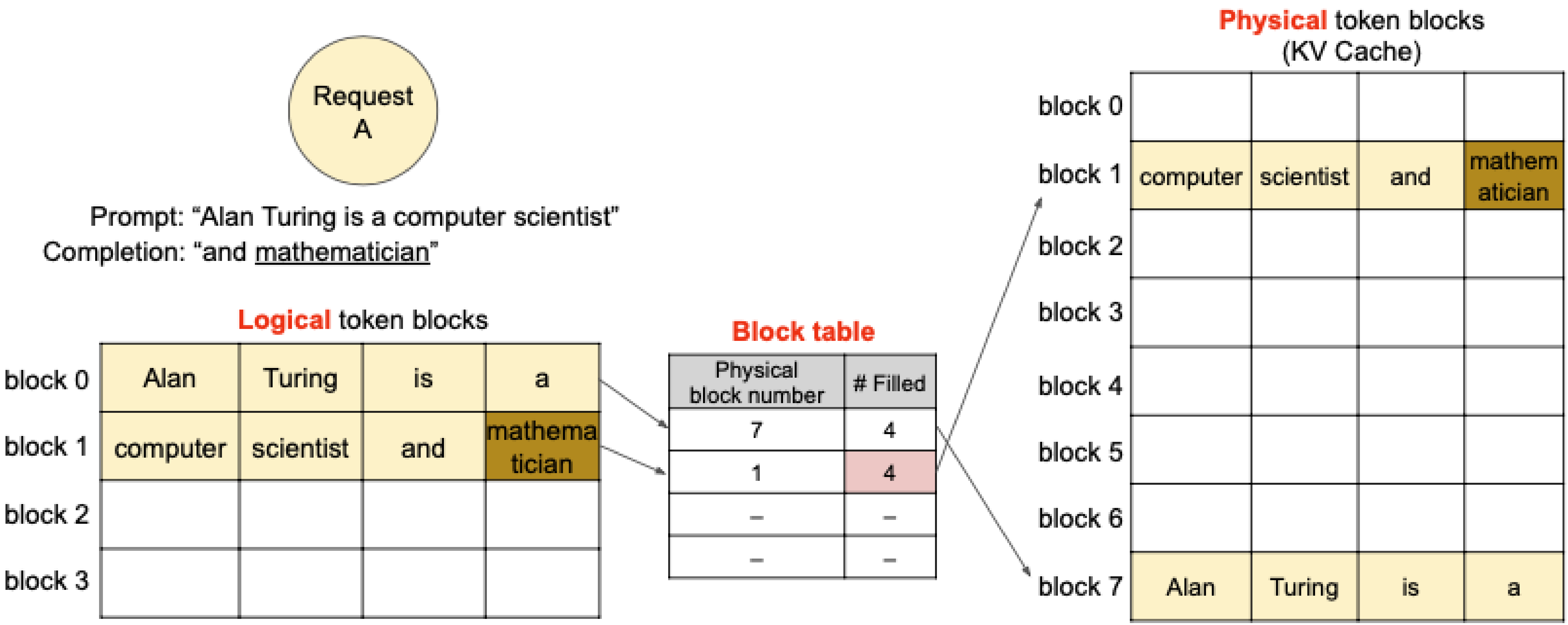
Woosuk, Kwon, et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention."

Logical & Physical KV blocks



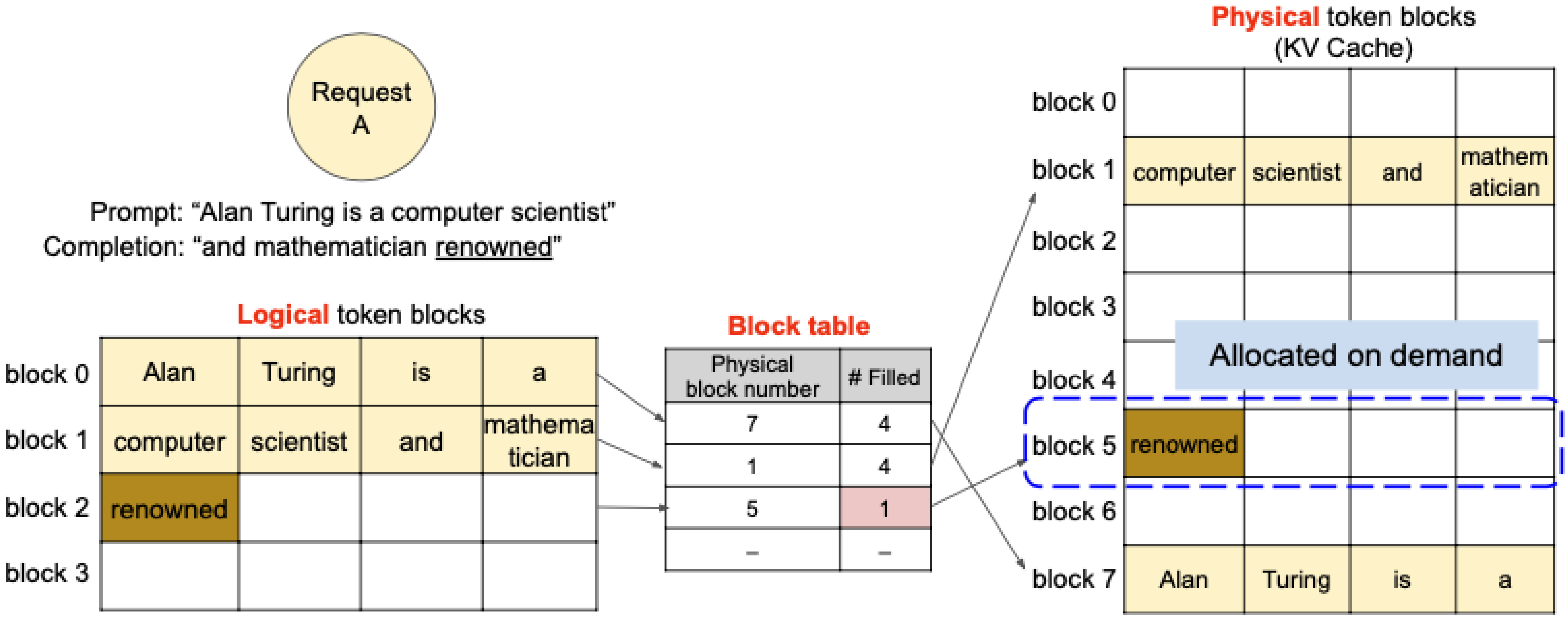
Woosuk, Kwon, et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention."

Logical & Physical KV blocks



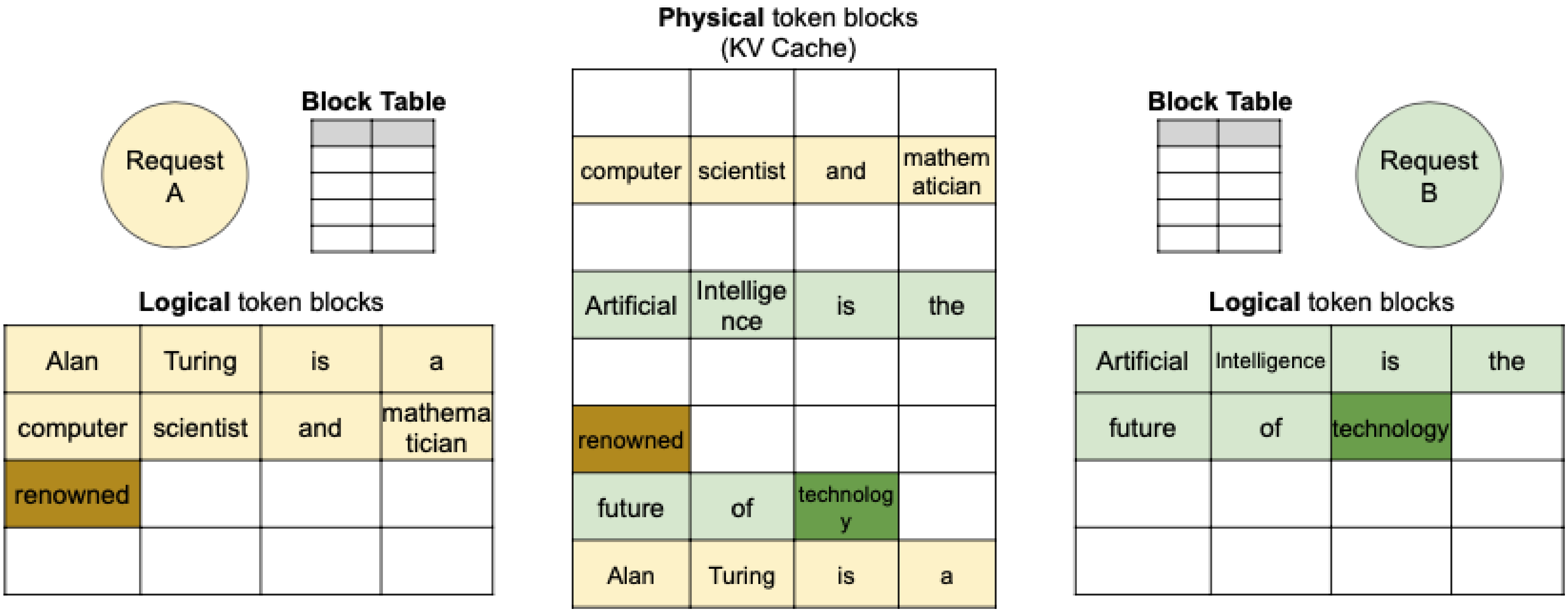
Woosuk, Kwon, et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention."

Logical & Physical KV blocks



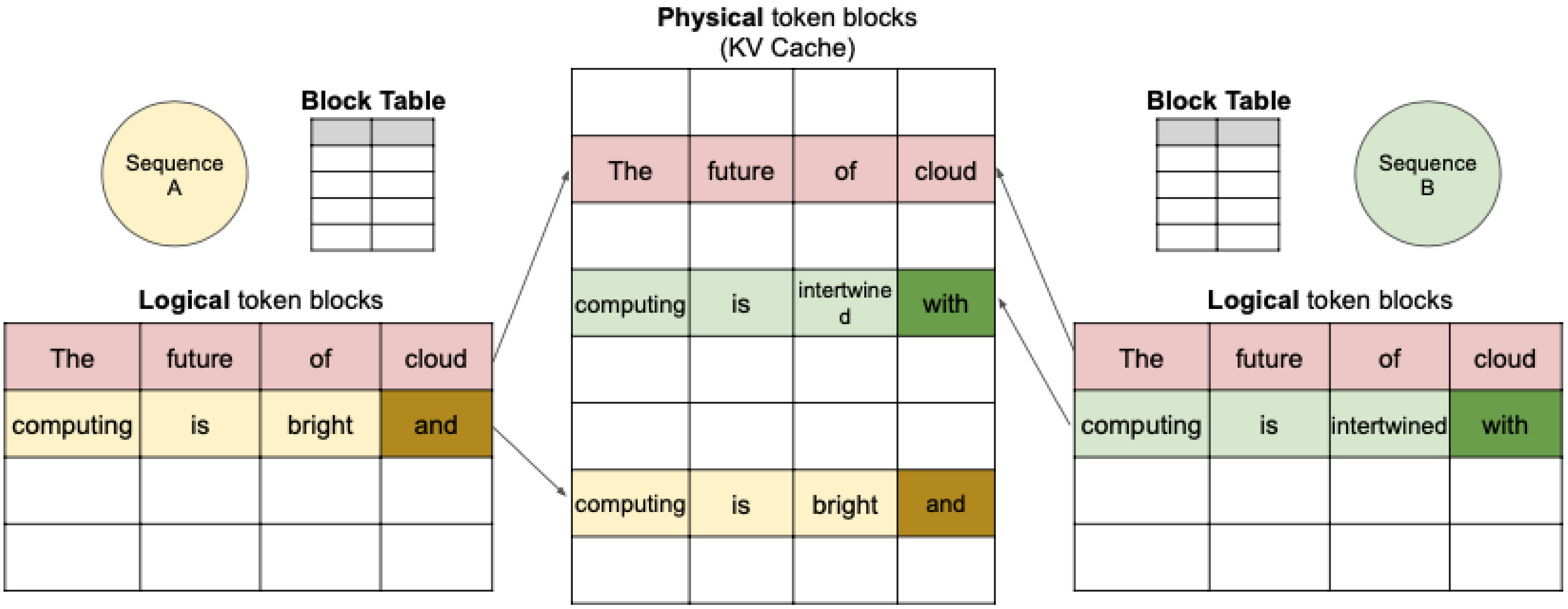
Woosuk, Kwon, et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention."

Multiple Request Serving



Woosuk, Kwon, et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention."

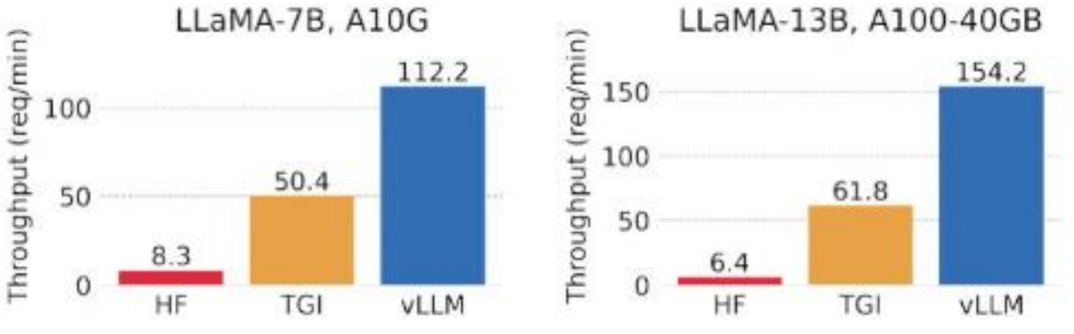
Token Block Sharing



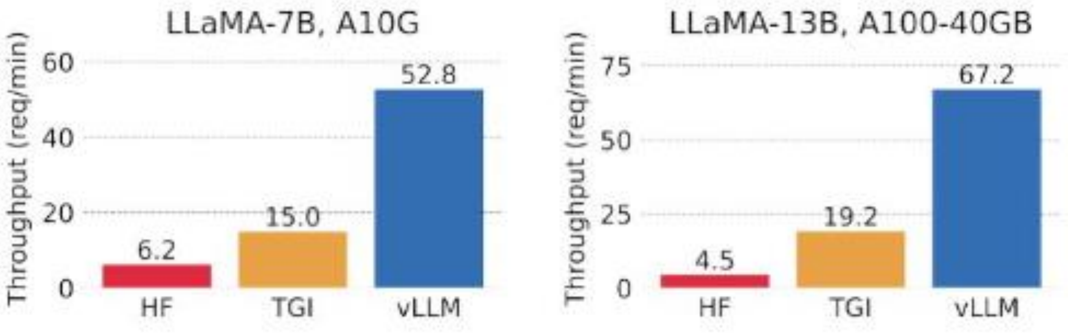
Woosuk, Kwon, et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention."

Performance Comparison with HuggingFace and TGI

- Throughput
  - 24x higher than HuggingFace
  - 3.5x higher 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.

Woosuk, Kwon, et al. "Efficient Memory Management for Large Language Model Serving with PagedAttention."

Outline

Lecture 11: Large Language Model Inference

- Inference Basic
- Additional Transformer Designs
  - KV cache
  - Attention mechanisms optimization
- Advanced Inference Systems
  - FlashAttention
  - vLLM
- **LLM Inference Randomness**

Temperature

- A crucial hyperparameter in fine-tuning the output of LLMs
  - control the randomness and creativity of generated text by adjusting word probability distributions.

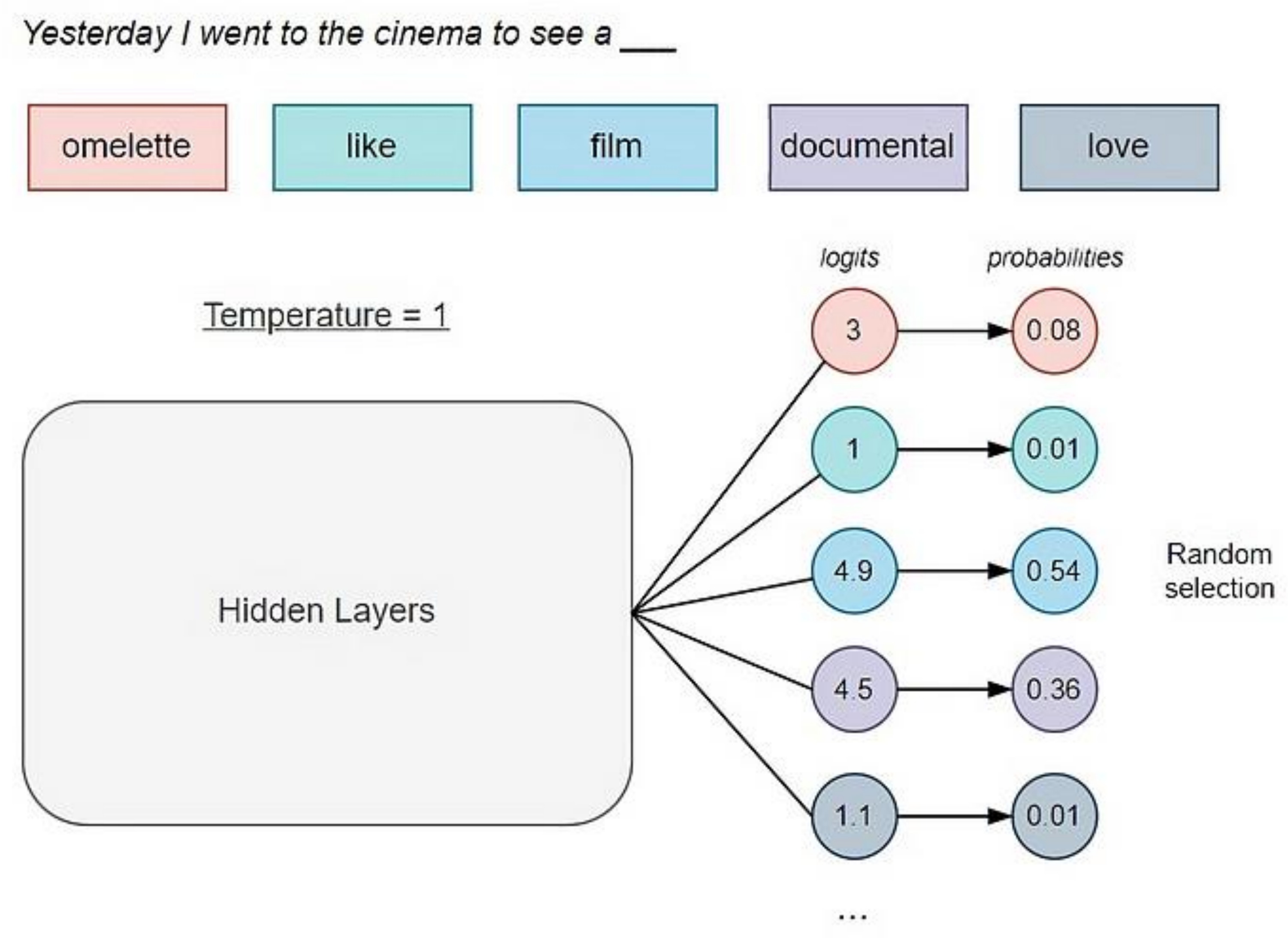
- How it is Implemented
  - Softmax (same as Temperature = 1)

$$P_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

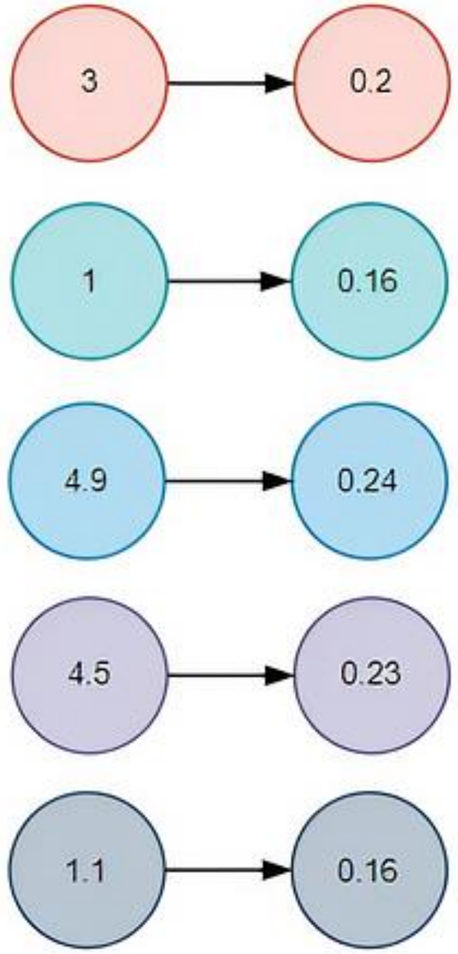
  - Softmax with Temperature (T)

$$P_i = \frac{e^{x_i/T}}{\sum_j e^{x_j/T}}$$

Temperature (T=10)

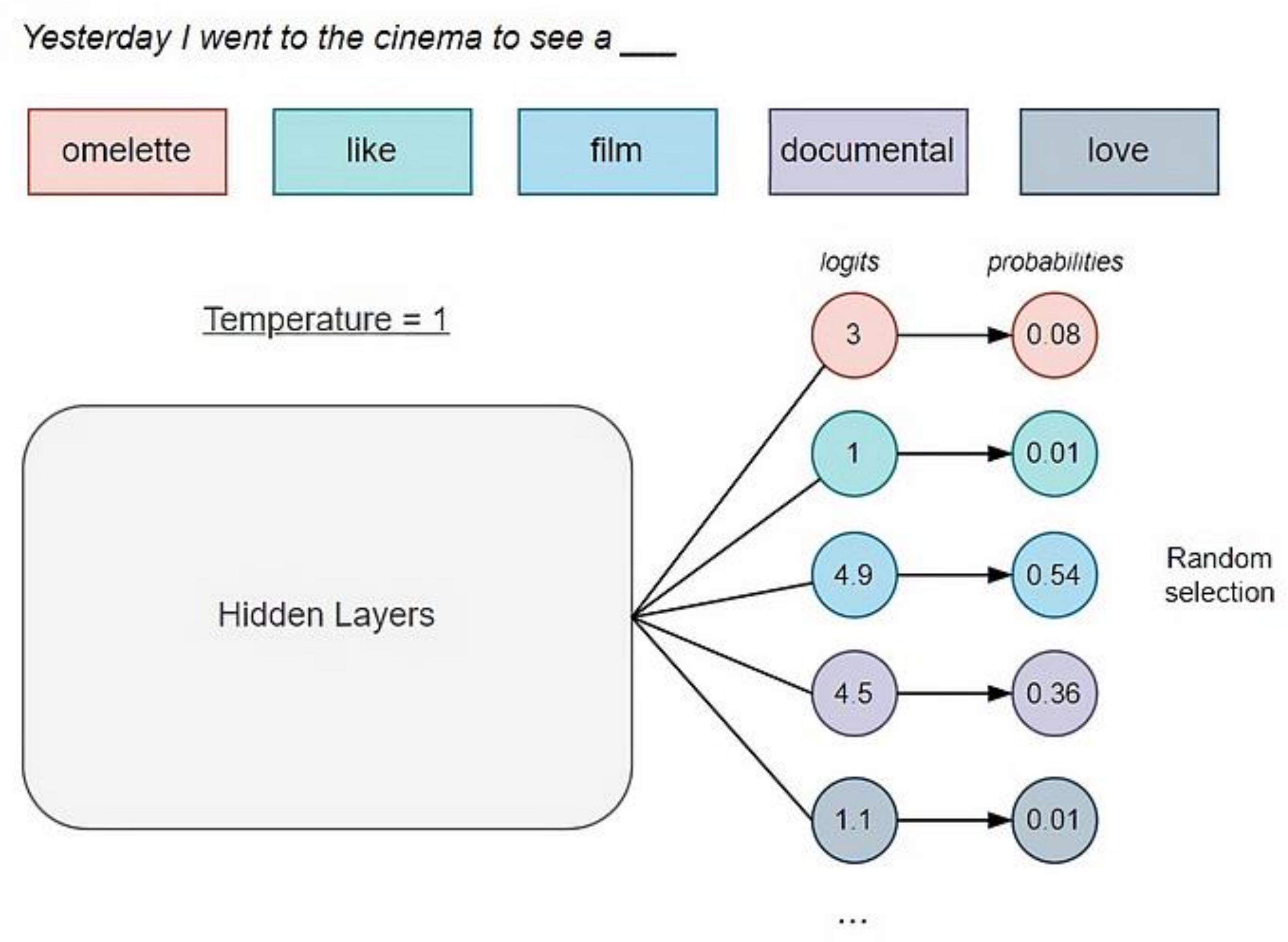


Temperature (T) = 10

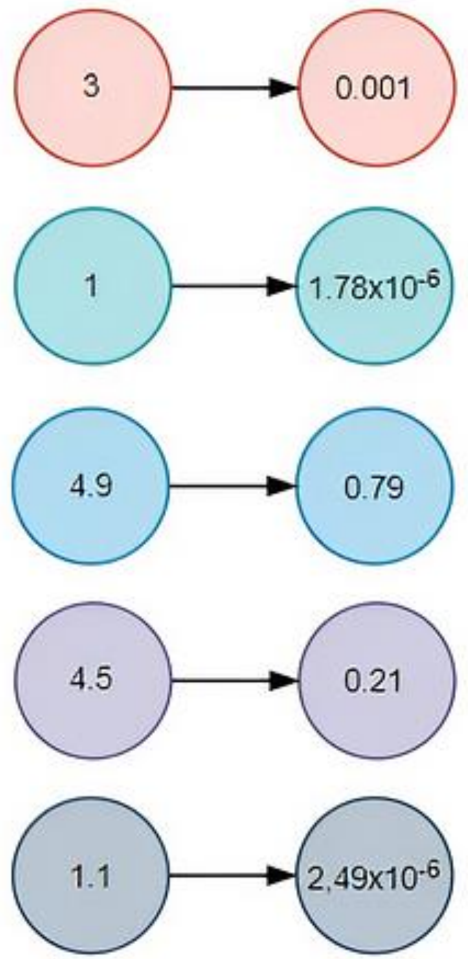


https://medium.com/the-science-of-control-how-temperature-top-p-and-top-k-shape-large-language-models

Temperature (T=0.3)

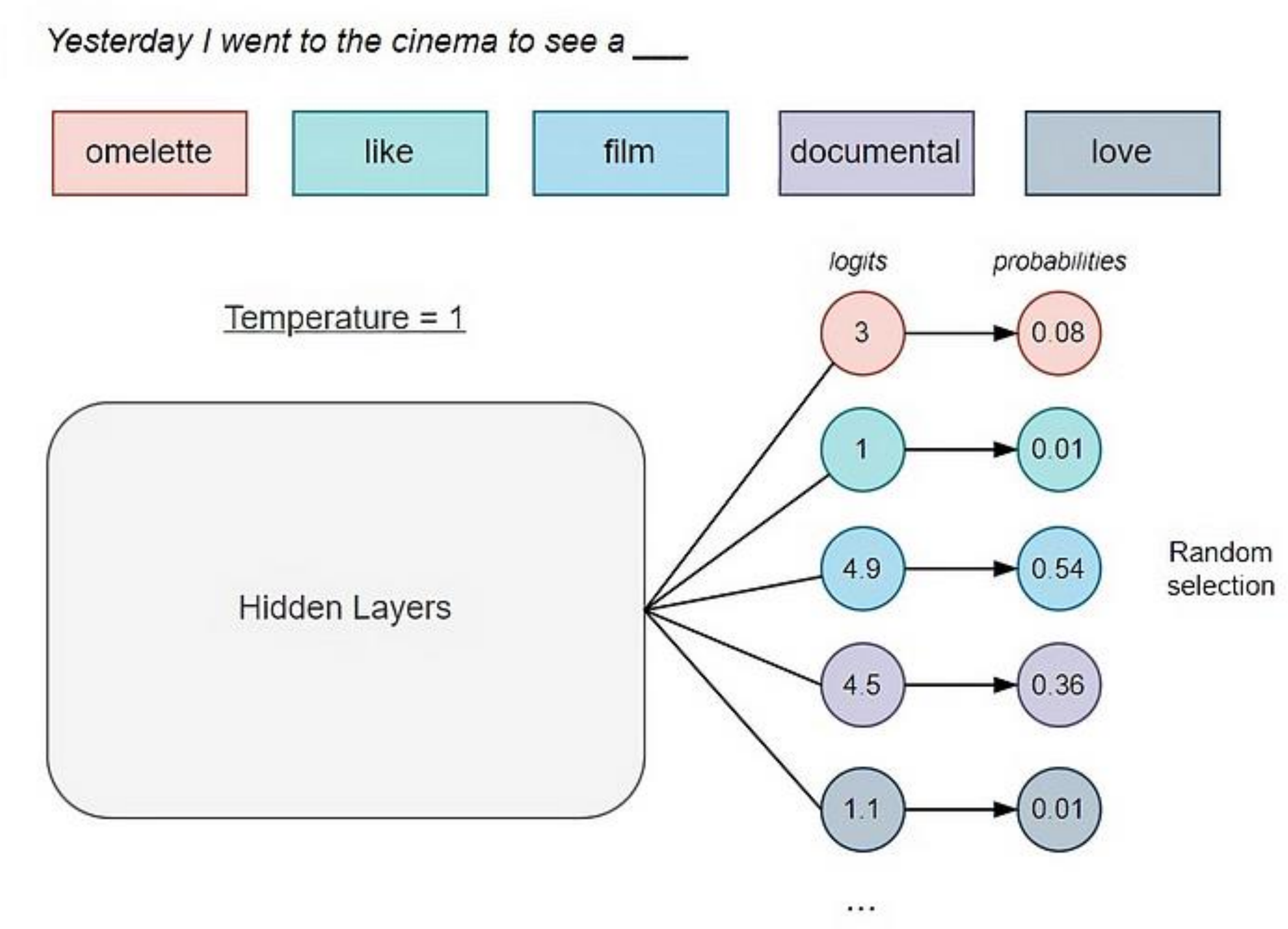


Temperature (T) = 0.3

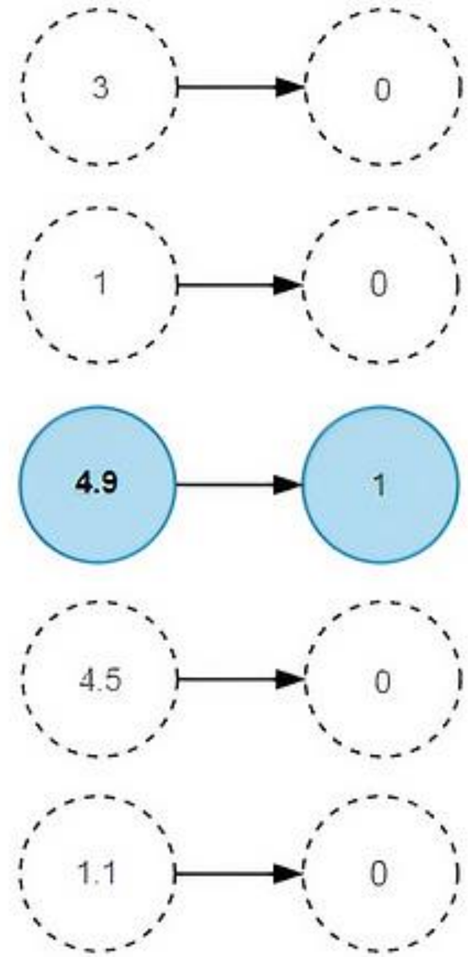


https://medium.com/the-science-of-control-how-temperature-top-p-and-top-k-shape-large-language-models

Temperature (T=0)



Temperature (T) = 0



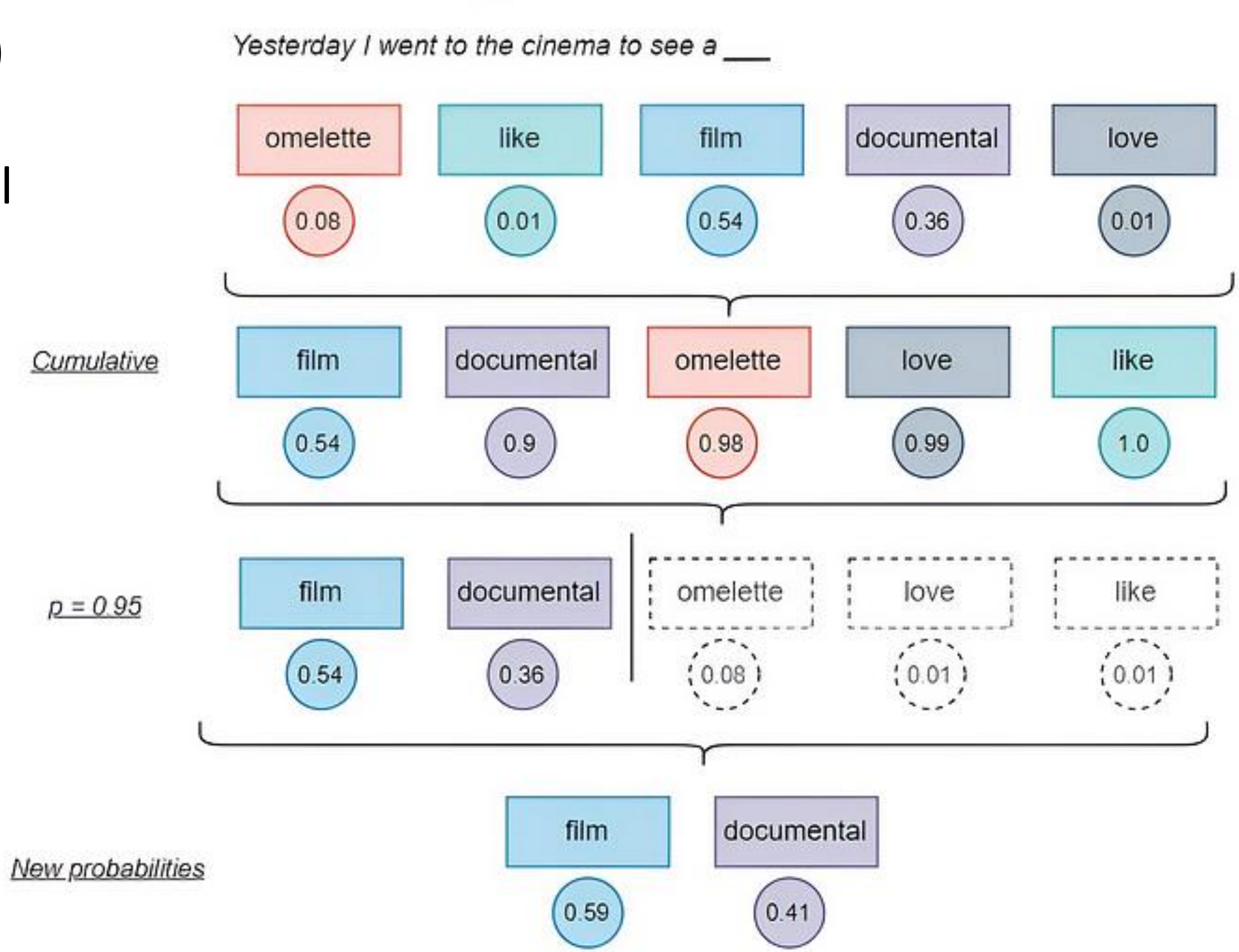
https://medium.com/the-science-of-control-how-temperature-top-p-and-top-k-shape-large-language-models

Temperature Impact

- Low Temperature ($T \to 0$)
  - Makes the model more deterministic
  - Higher probability words become even more likely
- High Temperature ($T > 1$):
  - Increases randomness.
  - Less probable words have a higher chance of selection
- Practical Usage:
  - Low T: For precise and factual responses
  - High T: For creative writing and idea generation

Top_p

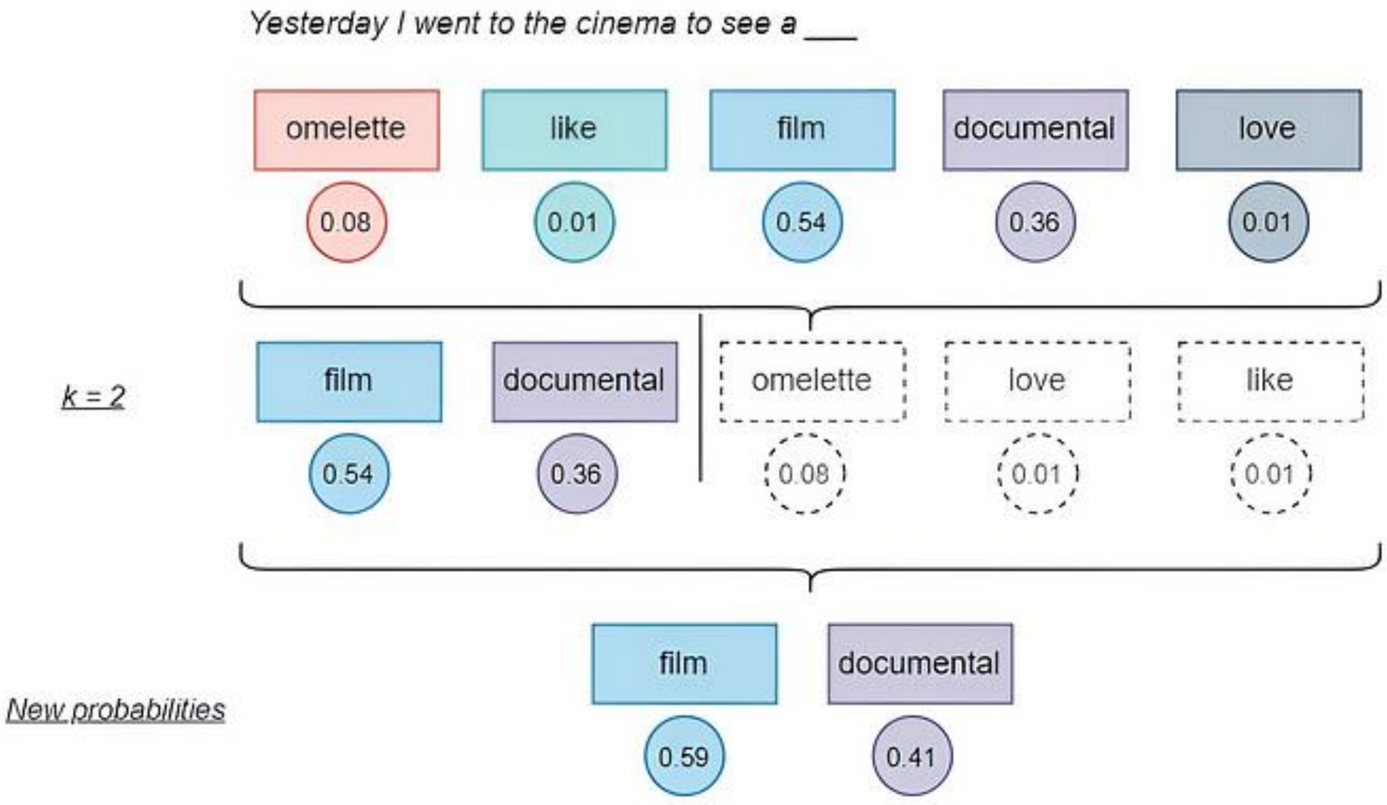
- Top_p (Nucleus Sampling)
  - Selects tokens based on cumulative probability until reaching a predefined threshold **p**.
  - Considers the smallest set of top tokens whose probabilities sum up to **p**.



https://medium.com/the-science-of-control-how-temperature-top-p-and-top-k-shape-large-language-models

Top_k

- Top_k (Top_k Sampling)
  - Considers only the top **k** most probable tokens.
  - Ignores all tokens outside the top **k** probabilities.



https://medium.com/the-science-of-control-how-temperature-top-p-and-top-k-shape-large-language-models

In this lecture we learned

- LLM Inference
  - Prefill and decoding phase
- Additional Transformer Design
  - KV cache
  - Group Query Attention, Multi-query Attention
- Advanced Inference Systems
  - FlashAttention
  - vLLM
- Temperature, Top-k & Top-p