LLM Inference Optimization

Spring 2026

Lecturer: Yuedong (Steven) Xu

Fudan University

ydxu@fudan.edu.cn

Disclaimer

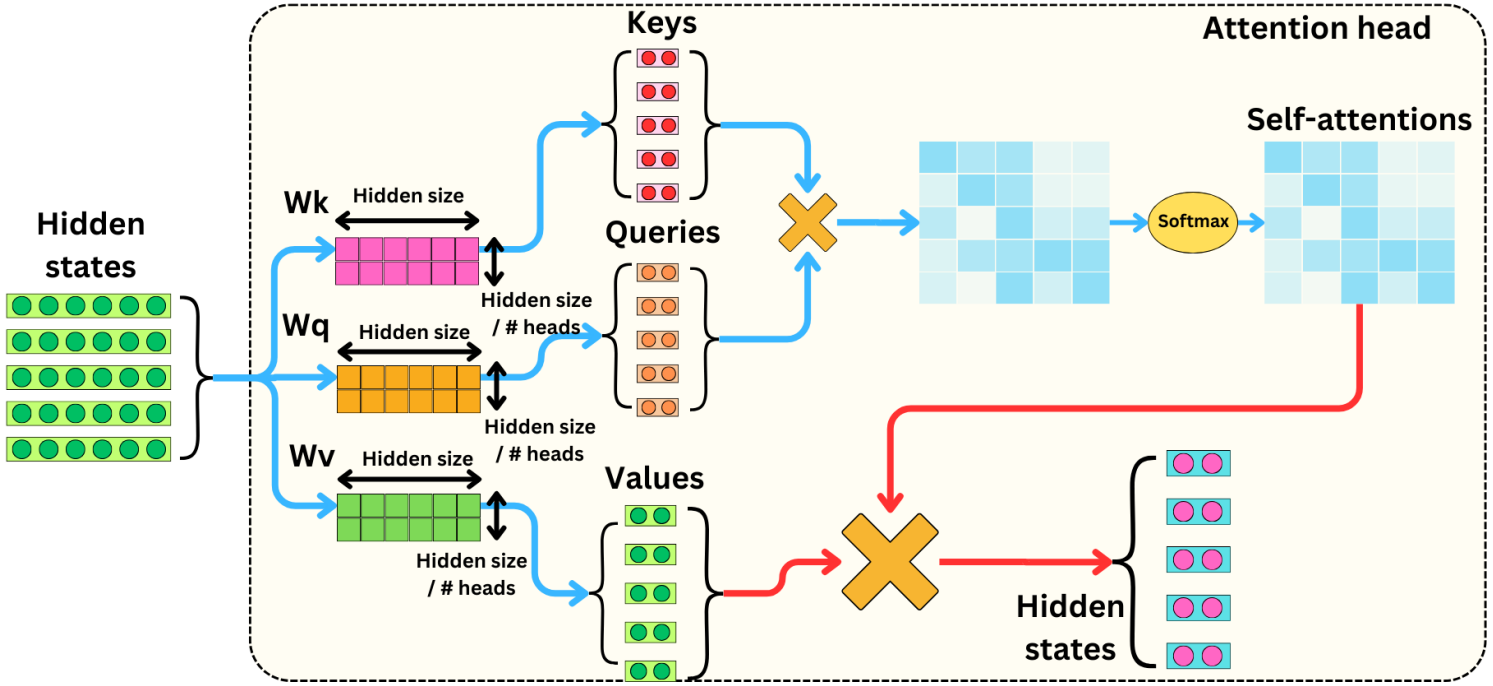
Machine learning systems is a broad and rapidly evolving field. The course material has been developed using a broad spectrum of resources, including research papers, lecture slides, blogposts, research talks, tutorial videos, and other materials shared by the research community. Sometimes animations and pictures from external sources are heavily reused.

Inference Optimization: Outline

- Overview
- Attention Optimization
- Continuous Batching
- KV Cache Optimization
  - **Model Architecture Optimization**
  - KV Cache Data Structure Optimization
  - KV Cache Memory Optimization
- Speculative Decoding
- Distributed Serving (Extended Learning)

Attention Architecture Optimization

- Multi-Head Attention
  - Each head has its own Query, Key and Value matrices



Source: The Multi-head Attention Mechanism Explained!

Attention Architecture Optimization

- Multi-Query Attention (MQA)
  - All heads in a layer share a single set of W_K and W_V
  - Size of Q unchanged, and sizes of K and V shrink by h times

Component	Weight matrix shape	Number of parameters (exact)
Query projection	$W_Q \in \mathbb{R}^{d\times(h\cdot d_h)}$	$d\cdot(h\cdot d_h)=d^2$
Key projection	$W_K \in \mathbb{R}^{d\times d_h}$	$d\cdot d_h=d\cdot(d/h)=d^2/h$
Value projection	$W_V \in \mathbb{R}^{d\times d_h}$	$d\cdot d_h=d^2/h$
Output projection	$W_O \in \mathbb{R}^{(h\cdot d_h)\times d}$	$(h\cdot d_h)\cdot d=d^2$
Total	—	$2d^2+2\cdot\frac{d^2}{h}$

Shared

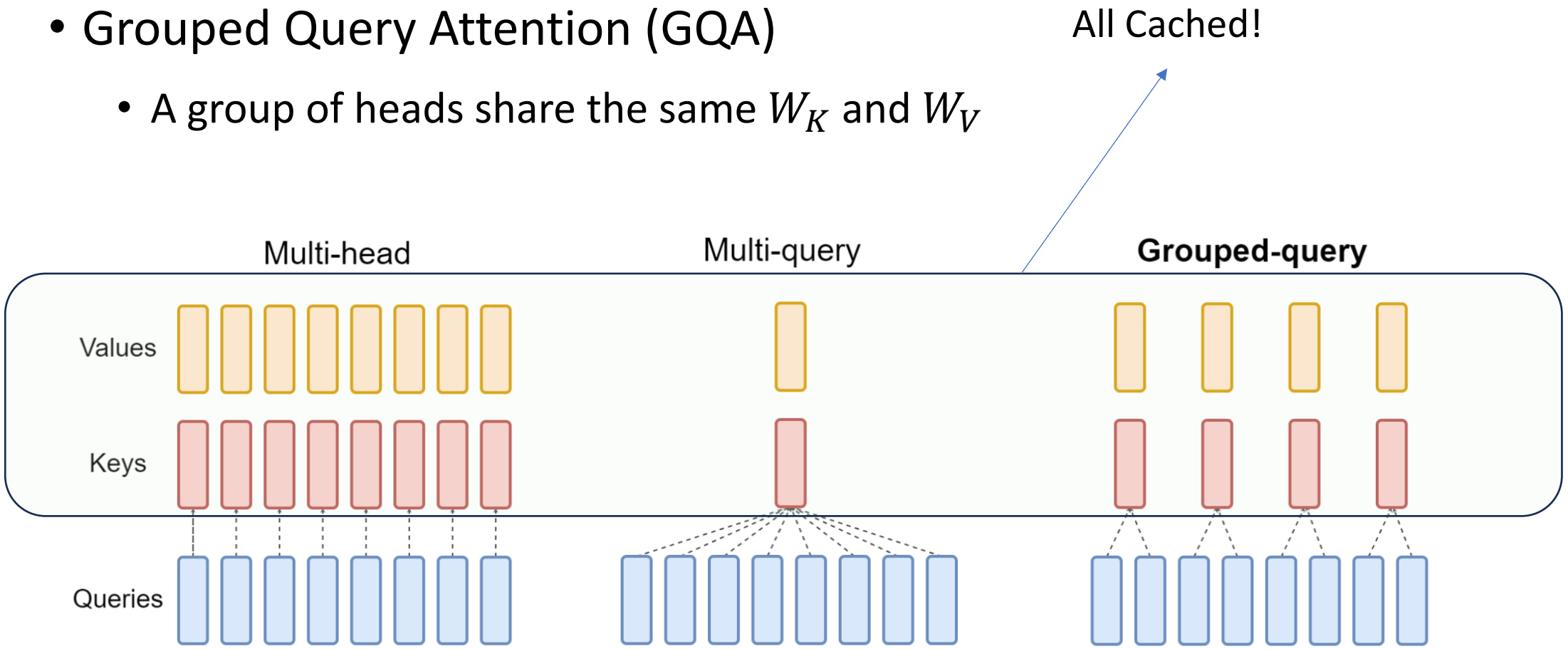
$$\frac{MQA}{MHA}=\frac{1+1/h}{2}$$

Number of parameters in MQA

Source: Fast Transformer Decoding: One Write-Head is All You Need

Attention Architecture Optimization

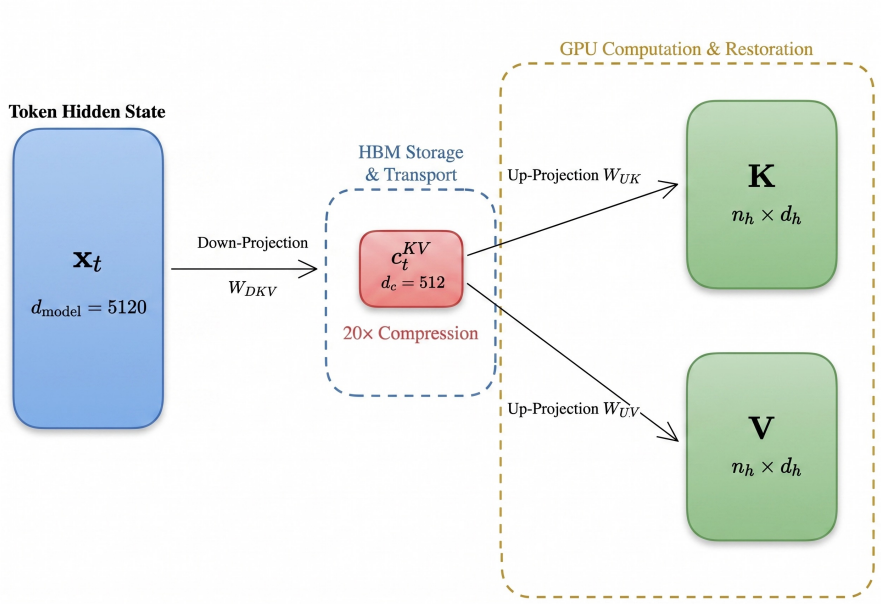
- Grouped Query Attention (GQA)
  - A group of heads share the same W_K and W_V



(Extended Learning)

Attention Architecture Optimization

- Multi-Head Latent Attention (MLA)
  - Compressing K and V into a small Latent Vector $c_t^{KV} \in \mathbb{R}^{d_c}$
  - Decompressing it into full Key and Value during inference
  - Only c_t^{KV} is **cached** where $d_c \ll d_h \cdot n_h$



$$W^K = W^{DKV} \cdot W^{UK}$$

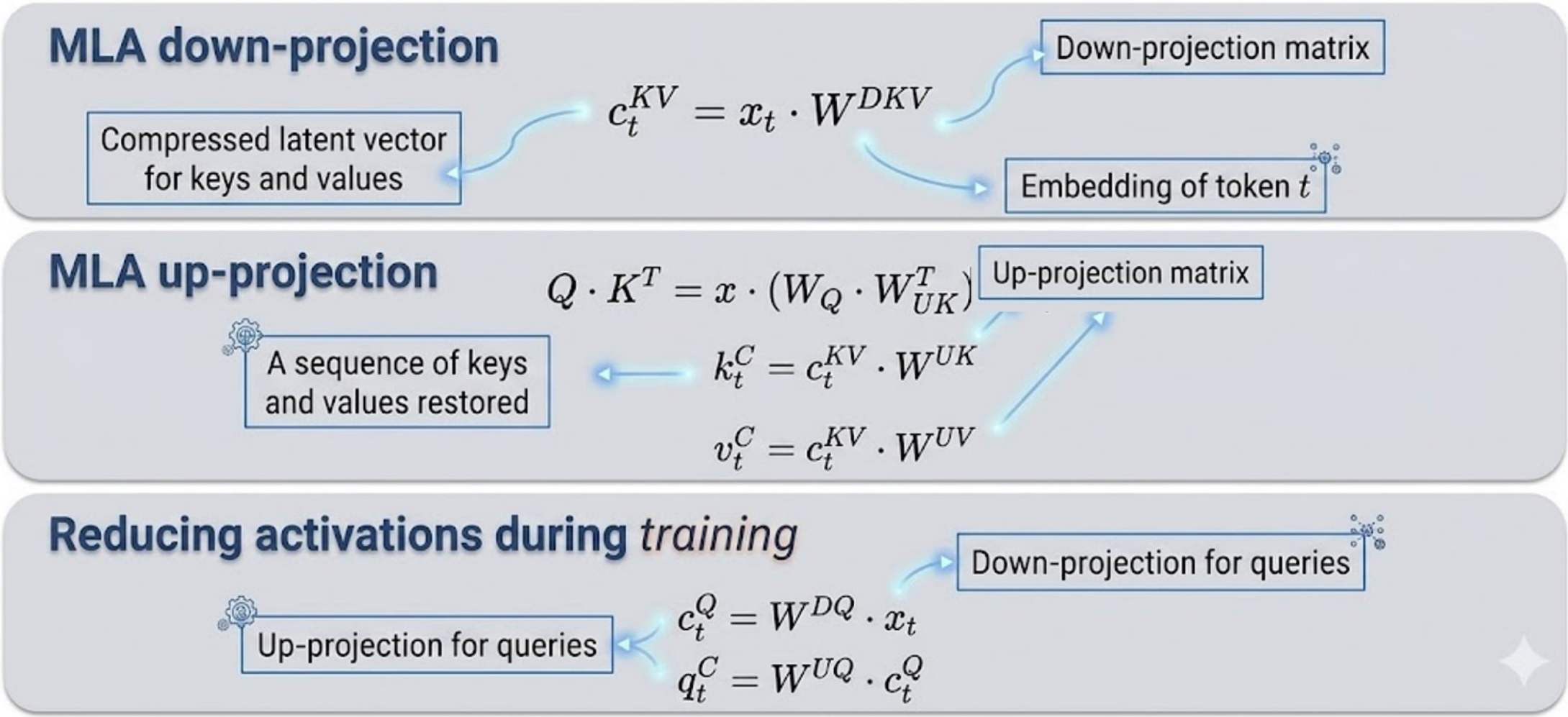
$$W^V = W^{DKV} \cdot W^{UV}$$

Trained parameters, not handcrafted

Source:DeepSeek-v3

(Extended Learning)

Attention Architecture Optimization



(Extended Learning)

Attention Architecture Optimization

- **Weight Absorption:** not explicitly computing up-projections

$$Q \cdot K^T = \underbrace{(x \cdot W_Q)}_{[1,d_h]} \cdot \underbrace{(c^{KV} \cdot W_{UK})^T}_{[d_h,L]}$$

Standard Attention Score Computation:
High computational & I/O latency

$$Q \cdot K^T = (x \cdot W_Q) \cdot W_{UK}^T \cdot (c^{KV})^T$$

$$Q \cdot K^T = x \cdot \underbrace{(W_Q \cdot W_{UK}^T)}_{W'_Q \in \mathbb{R}^{d_{model} \times d_c}} \cdot (c^{KV})^T$$

$W'_Q = W_Q \cdot W_{UK}^T$ **Precomputed Matrix**

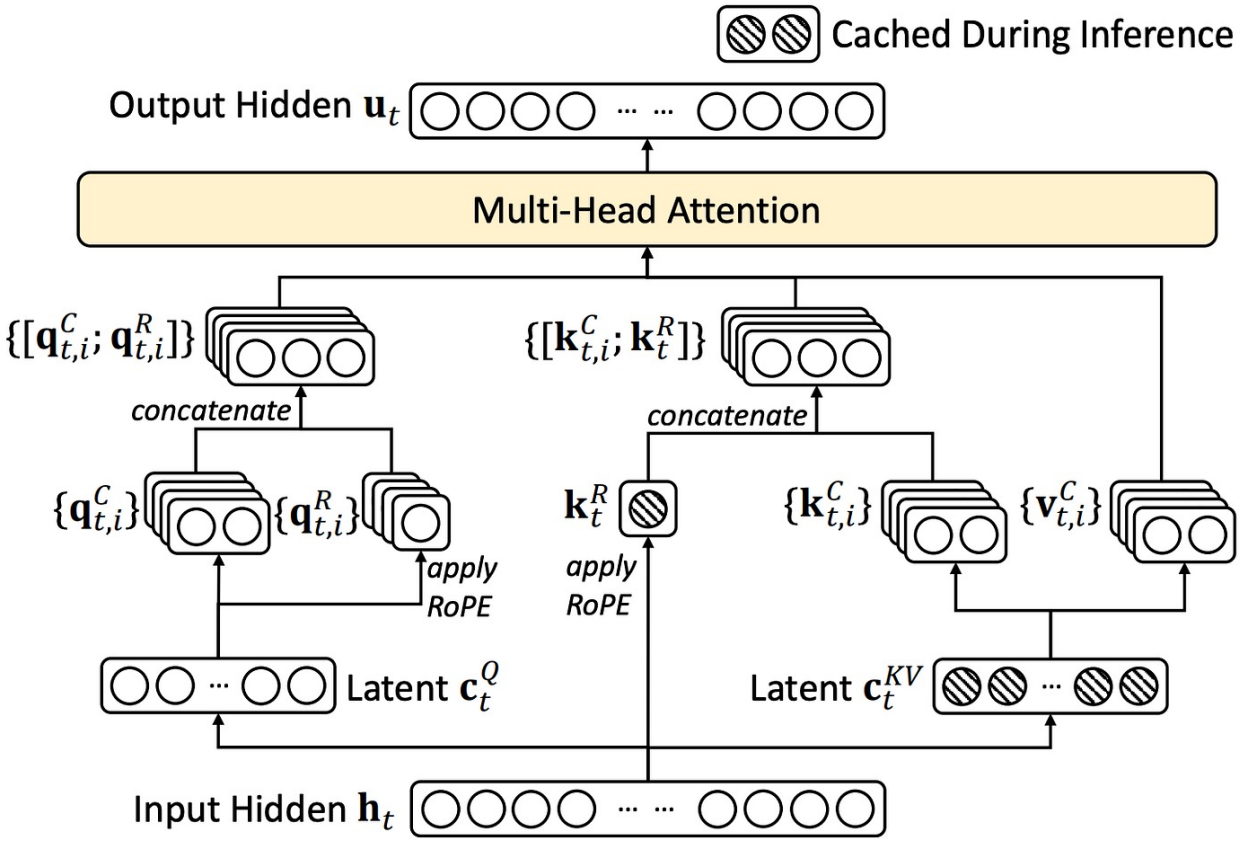
$$Q' = x \cdot W'_Q, \quad \text{Score} = Q' \cdot (c^{KV})^T$$

Result: Dot Product for Final Score

(Extended Learning)

Attention Architecture Optimization

- Decoupled Rotary Position Embedding (RoPE) (for reference) **cached**



The architecture of Multi-head Latent Attention (MLA)

$$q_t^R = \mathrm{RoPE}(W^{QR} c_t^Q)$$

$$q_{t,i} = [q_{t,i}^C, q_{t,i}^R]$$

$$k_t^R = \mathrm{RoPE}(W^{KR} h_t)$$

$$k_{t,i} = [k_{t,i}^C, k_{t,i}^R]$$

$$v_t^C = W^{UV} c_t^{KV}$$

$$q_{t,i} = \sum_{j=1}^{t} Softmax_j\left(\frac{q_{t,i}^T q_{j,i}^C}{\sqrt{d_h + d_h^R}}\right) v_{j,i}^C$$

$$u_t = W^O [o_{t,1}; o_{t,2}; \cdots; o_{t,n_h}]$$

Source:DeepSeek-v3

Attention Architecture Optimization

- KV Cache Per-Token

MHA	GQA	MQA	MLA
$2d$	$2n_{kv}d_h$	$2d_h$	d_c+d_h^{rope}
32768 $(n_h = 128, d_h = 128)$	256~32768	256	576 $(d_c = 512)$

Source:DeepSeek-v3

(Extended Learning)

Attention Architecture Optimization

DeepSeek v4: Compressed Sparse Attention (CSA)

Extended Reading

- Sequence dimension compression: not all historic tokens are important

Linear projection of tokens

$$C_a = H \cdot W_a^{KV}, \quad C_b = H \cdot W_b^{KV}$$

$$Z_a = H \cdot W_a^{Z}, \quad Z_b = H \cdot W_b^{Z}$$

where $W^{aKV}, W^{bKV}, W^{aZ}, W^{bZ} \in R^{d \times c}$ are trainable parameters

Compressing m consecutive tokens

$$[S^a_{mi:m(i+1)-1}; S^b_{m(i-1):mi-1}]$$

$$= \mathrm{Softmax}_{\mathrm{row}}([Z^a_{mi:m(i+1)-1} + B^a; Z^b_{m(i-1):mi-1} + B^b])$$

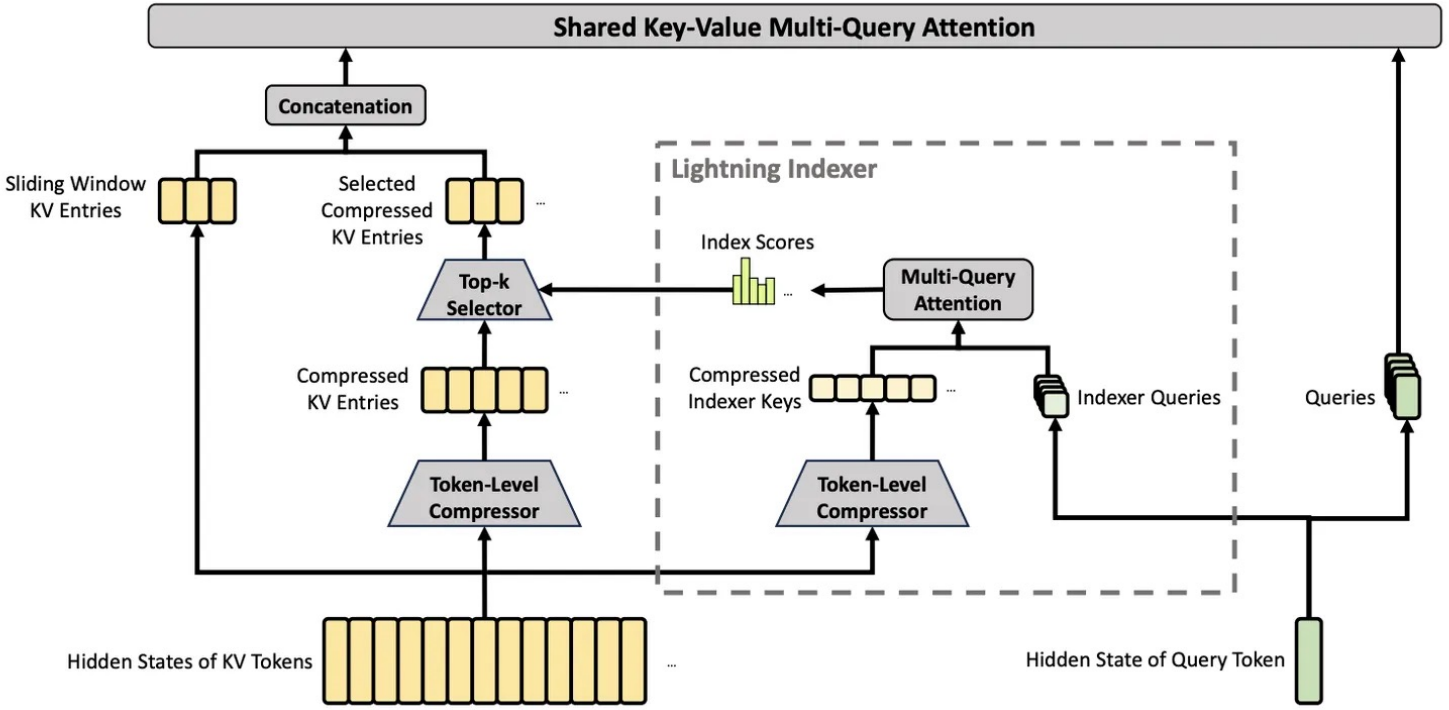
$$C_i^{Comp} = \sum_{j=mi}^{m(i+1)-1} S_j^a \odot C_j^a + \sum_{j=m(i-1)}^{mi-1} S_j^b \odot C_j^b$$

Lightning indexer Sparsification to select most relevant KV

(Extended Learning)

Attention Architecture Optimization

- DeepSeek v4: Compressed Sparse Attention

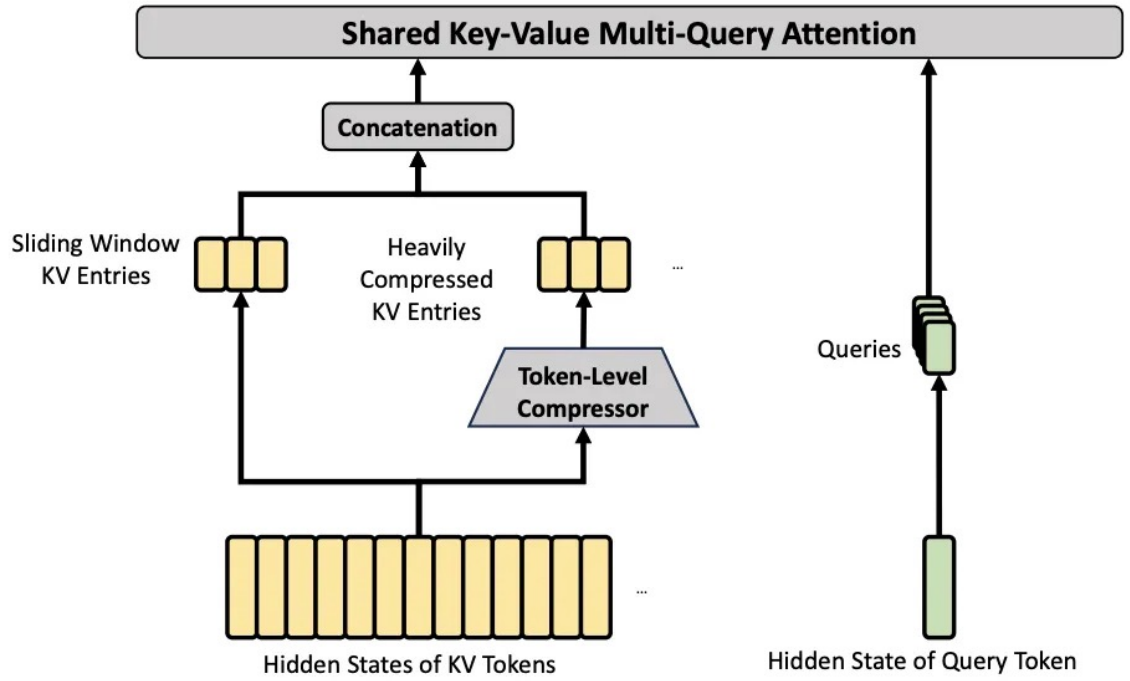


Source:DeepSeek-v4

(Extended Learning)

Attention Architecture Optimization

- DeepSeek v4: Heavily Compressed Attention (HCA)
  - Compressing m consecutive tokens (e.g. 128) as a minimal summary



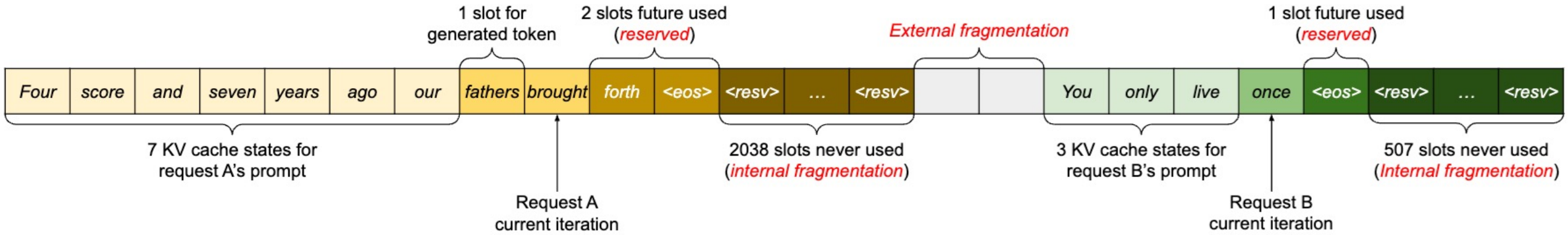
Source:DeepSeek-v4

Inference Optimization: Outline

- Overview
- Attention Optimization
- Continuous Batching
- KV Cache Optimization
  - Model Architecture Optimization
  - **KV Cache Data Structure Optimization**
  - KV Cache Memory Optimization
- Speculative Decoding
- Distributed Serving (Extended Learning)

Memory Management

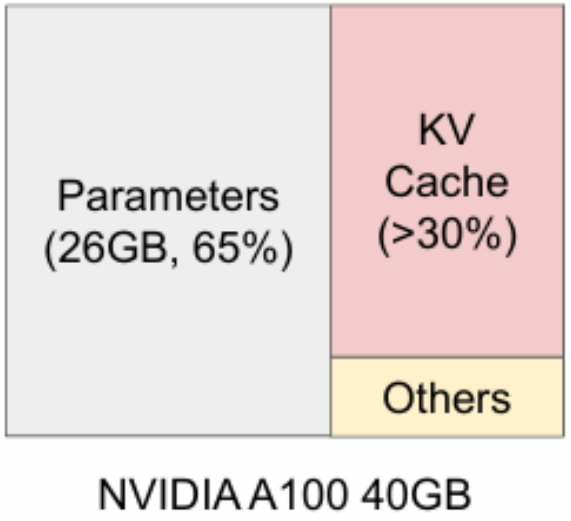
- Paged Attention
  - Uncertainty in ***Decode***: no one know how many tokens will be generated (remember continuous batching), and how much space needs to be allocated beforehand



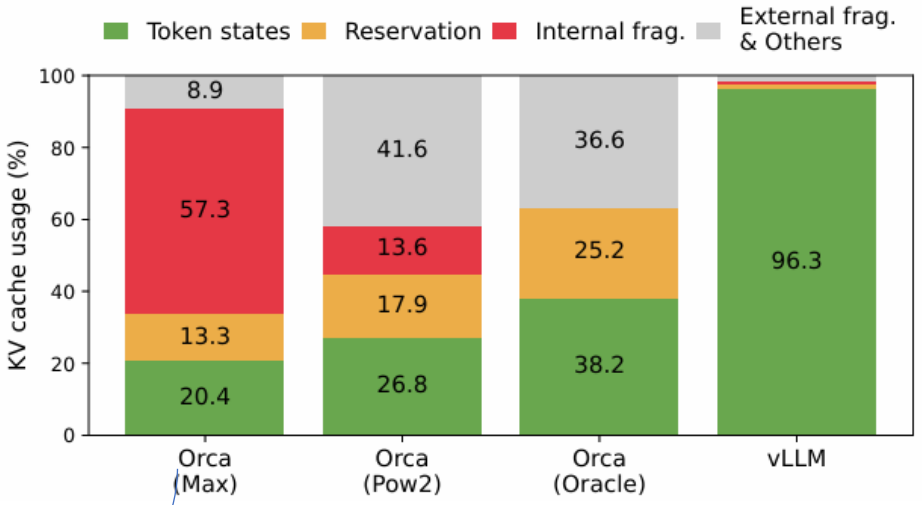
  - Memory fragmentations
    - Reserved fragmentation : reserved memory space not filled with output tokens temporarily
    - Internal fragmentation: reserved but never used even after the token generation completes
    - External fragmentation: memory space insufficient for inserting new KV cache

Memory Management

- Paged Attention
  - An example of memory fragmentation



Grey: parameters persisting in GPU memory throughout serving
Red: KV cache (de)allocated per serving request
Yellow: used ephemerally for activation.



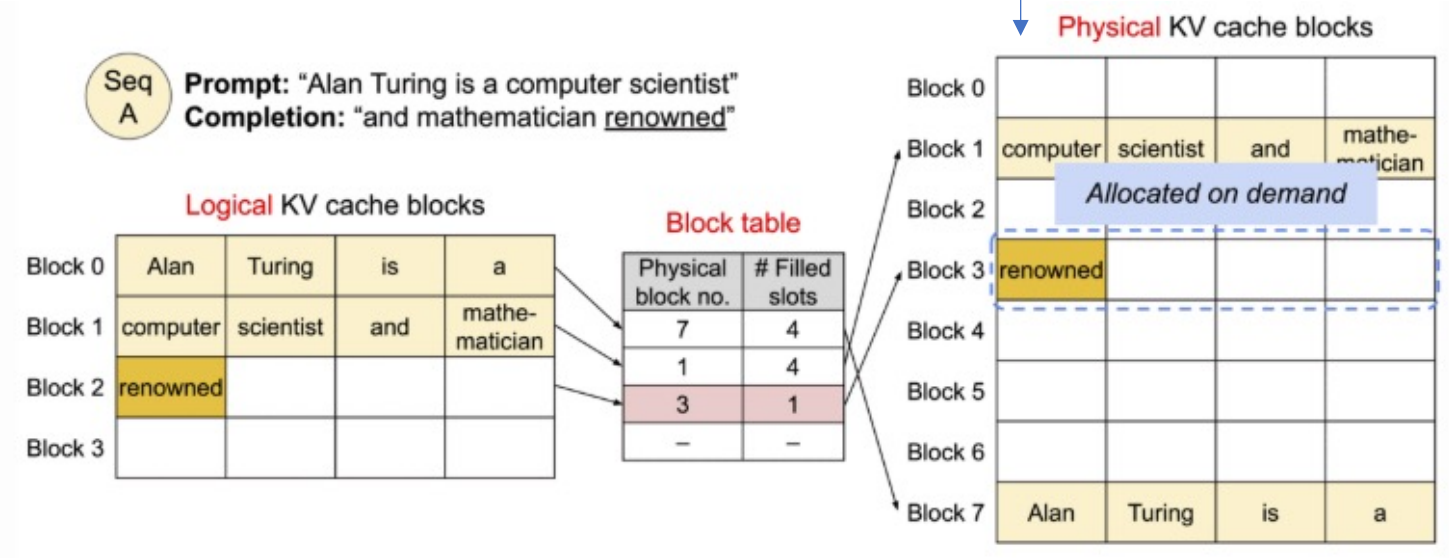
NVIDIA A100 with 40GB HBM

Orca: another serving framework

Memory Management

- Paged Attention
  - Inspired by paged virtual memory
    - KV block table: translation between logical KV blocks (what the AI model sees) and their actual physical locations in GPU memory

What is the size of a physical block?



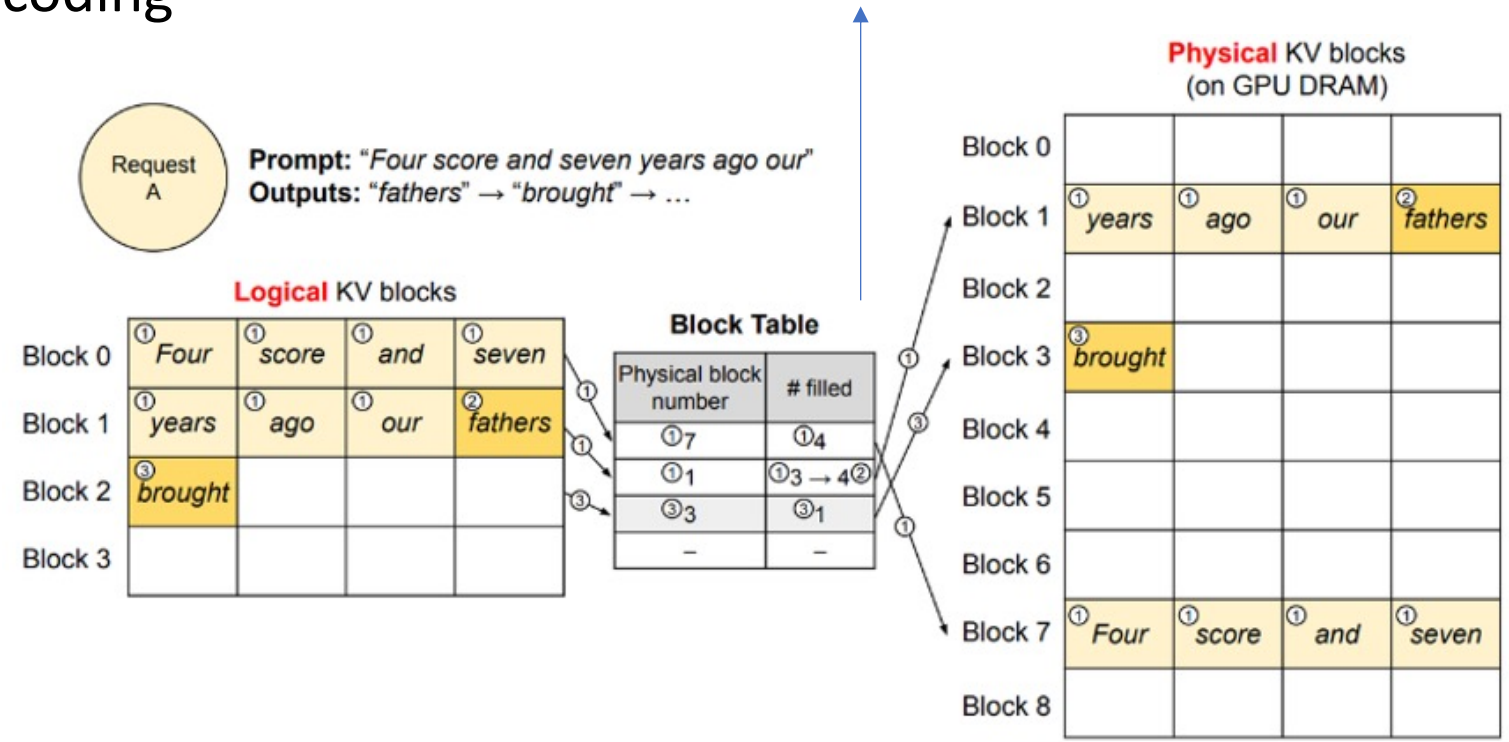
Logica block #0 (#2) is stored physical block #7 (#3) where the KV of 4 (only 1) tokens are filled

Memory Management

- Paged Attention
  - An example of decoding

The last slot of logical block 1 is still available, vLLM stores the newly generated token there, and the block table is updated

A prompt containing 7 tokens: vLLM initially allocates 2 logical blocks to store the KV (block 0 and block 1), then maps them to 2 physical KV blocks (blocks 1 and 7) and records this mapping in the block table.

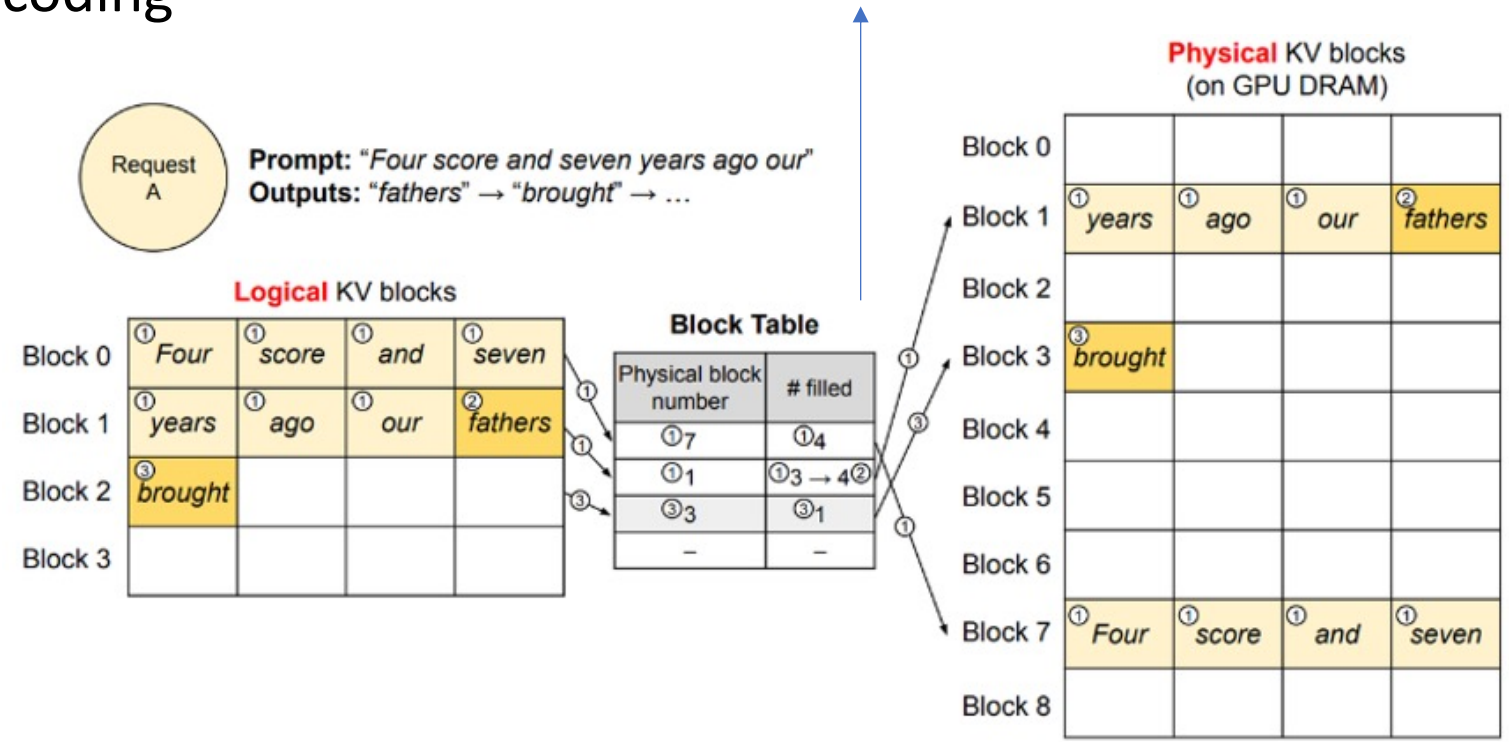


Memory Management

- Paged Attention
  - An example of decoding

The last slot of logical block 1 is still available, vLLM stores the newly generated token there, and the block table is updated

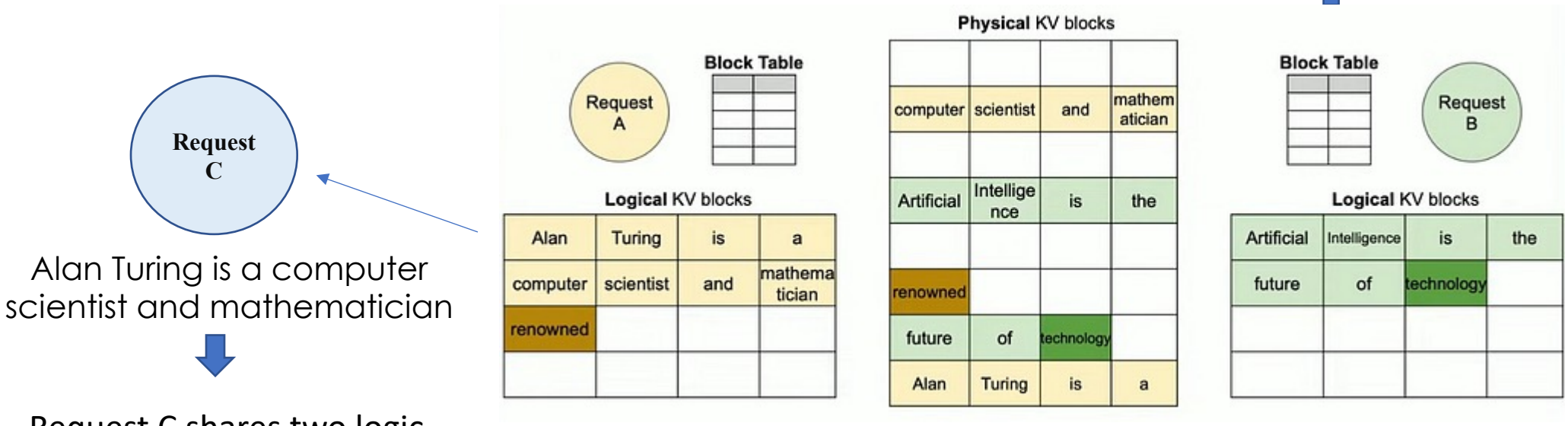
A prompt containing 7 tokens: vLLM initially allocates 2 logical blocks to store the KV (block 0 and block 1), then maps them to 2 physical KV blocks (blocks 1 and 7) and records this mapping in the block table.



Memory Management

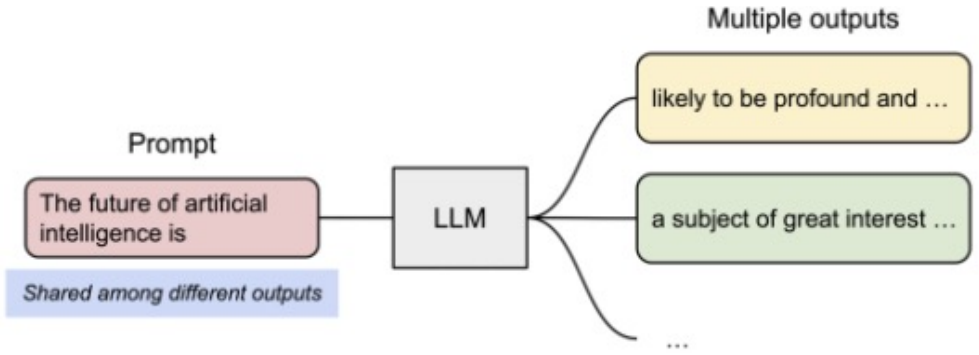
- Paged Attention
  - An example of decoding

Logical blocks of different requests will be mapped to different physical blocks

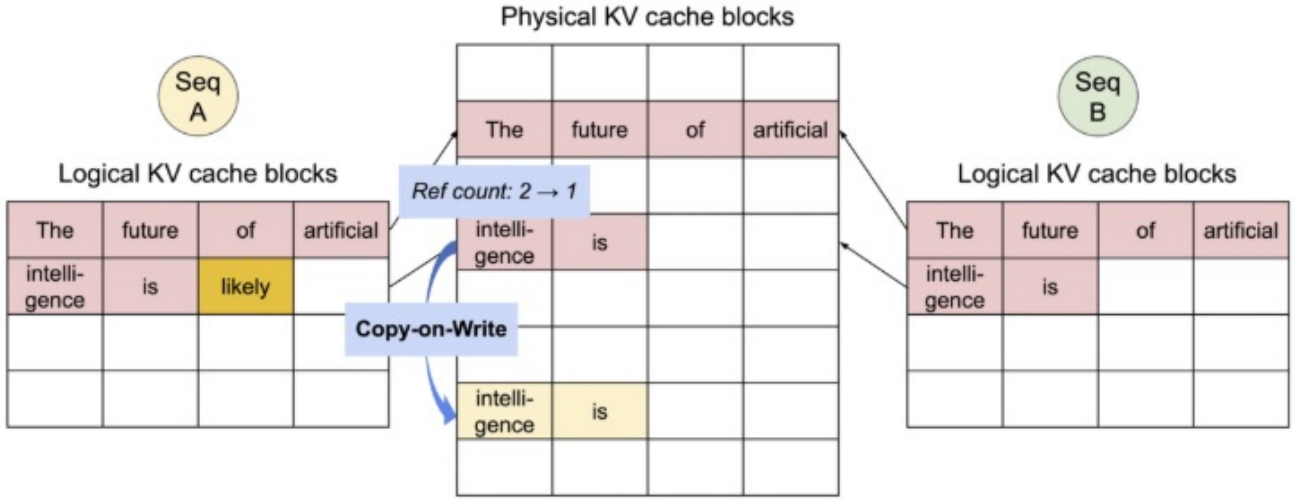


Memory Management

- Paged Attention
  - Parallel sampling or beam search
    - Parallel sampling: multiple generation paths originate from the same prompt
    - Copy-on-Write (CoW): a memory optimization strategy where data isn't immediately duplicated when a copy is requeste



Parallel sampling

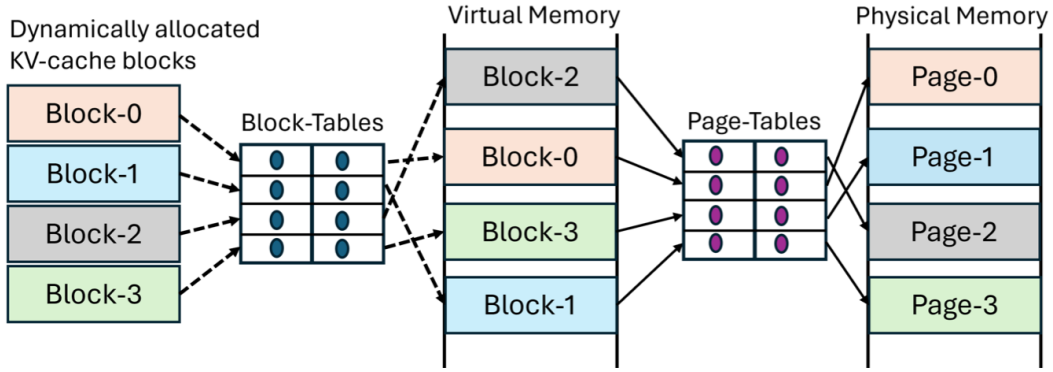


logical blocks for the prompts of both sequences are mapped to the same physical blocks

(Extended Learning)

Memory Management

- Revisiting PagedAttention
  - Requires re-writing the attention kernel and adds redundancy in the serving framework



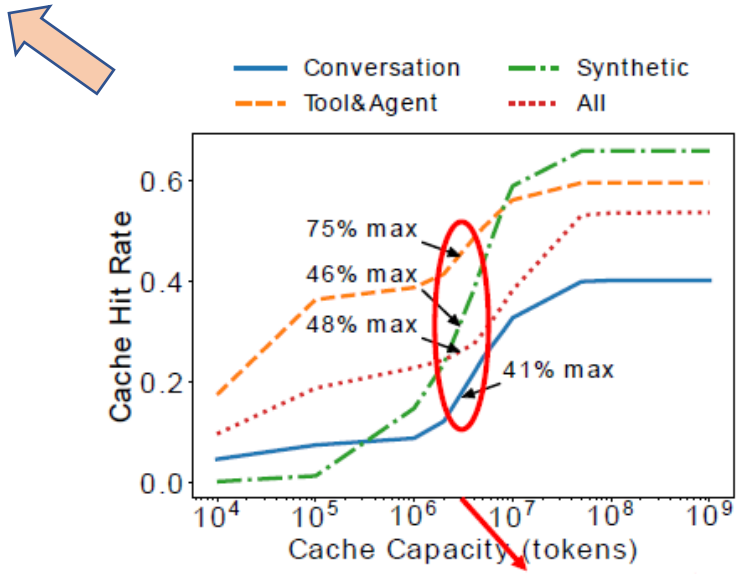
A request allocates four KV cache blocks over time that are usually non-contiguous in virtual memory. The serving system needs to track the virtual memory addresses of KV cache blocks and pass them to the attention kernel at runtime.

- vAttention
  - Mitigates fragmentation in physical memory while retaining the virtual memory contiguity of the KV cache (supplementary information)

Source:vAttention: Dynamic Memory Management for Serving LLMs without PagedAttention

Memory Management

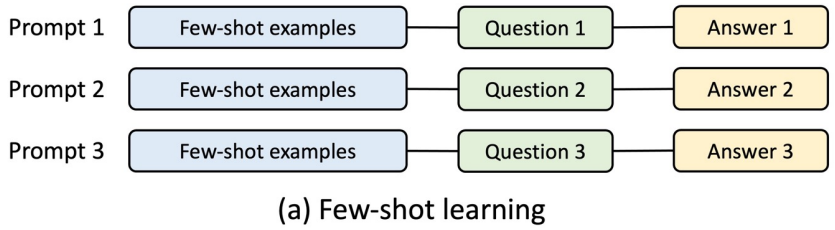
- Prefix Caching
  - **20-80 rule**: a small portion of requests accounts for the majority of usage
  - KV cache reuse across requests
- Application Scenarios
  - Multi-Turn Conversations
  - Retrieval-Augmented Generation
  - Coding Assistants and IDE Integration
  - Few-Shot Prompting / In-Context Learning
  - ……



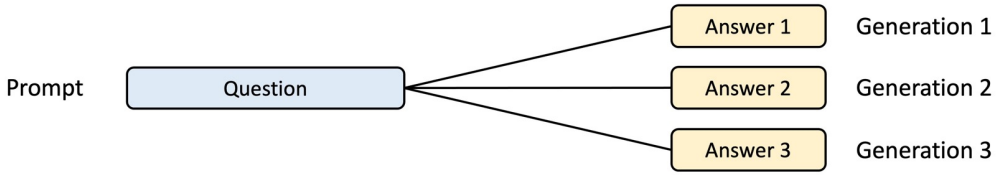
Around 50% of the tokens' KVCache in the real-world workloads can be reused

Memory Management

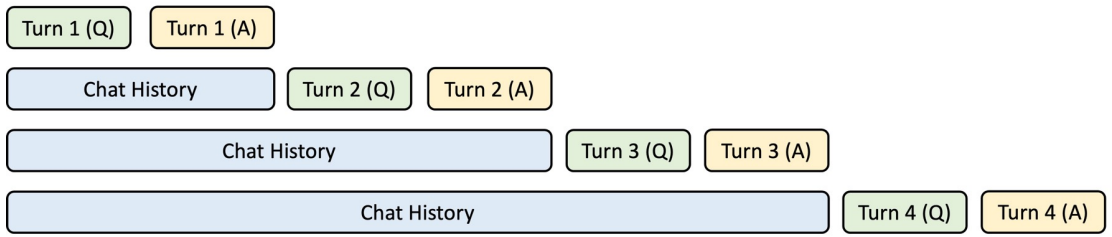
- KV Cache Reuse



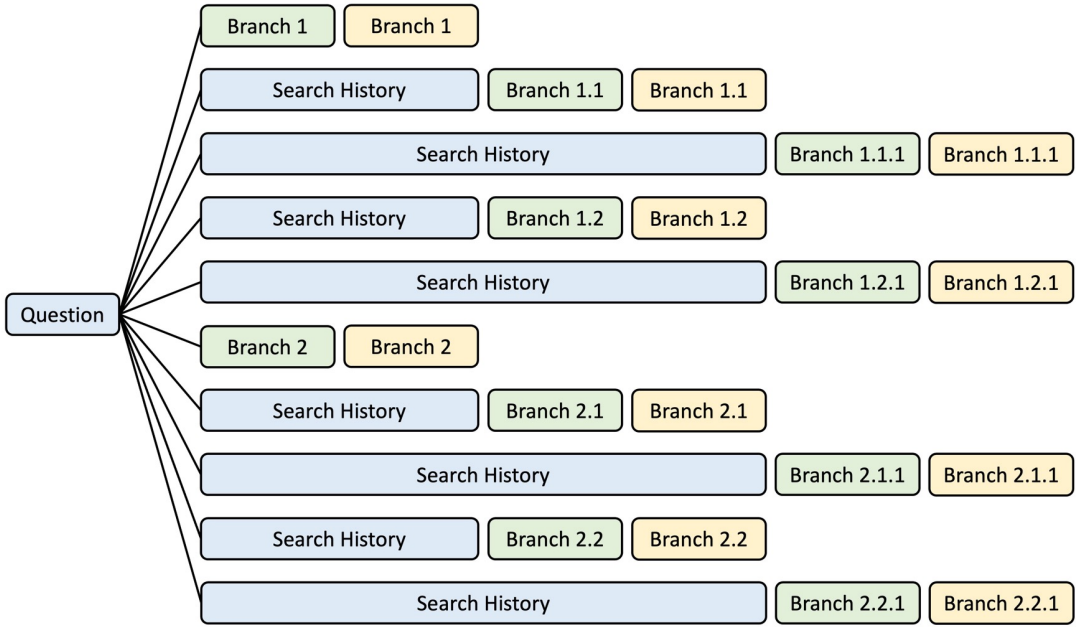
(a) Few-shot learning



(b) Self-consistency



(c) Multi-turn chat

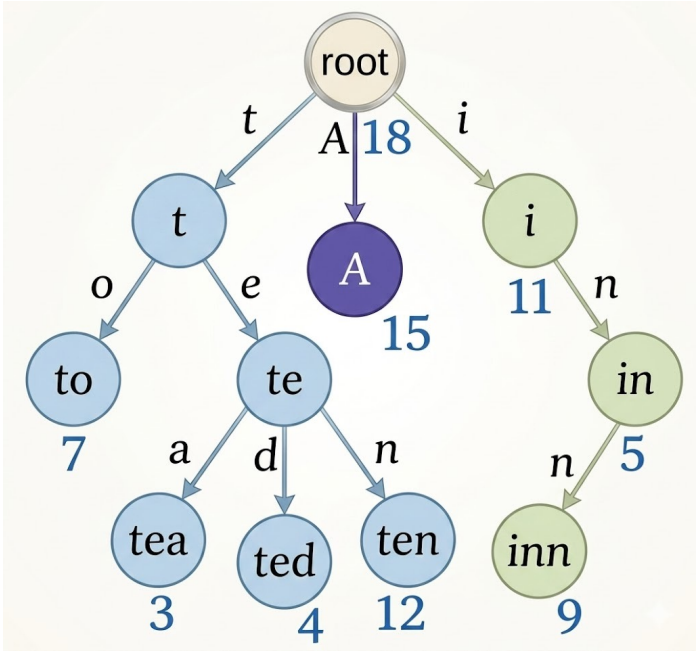


(d) Tree-of-thought

https://medium.com/@marketing_novita.ai/accelerating-llm-workflows-with-sglang-on-novita-ai-051a35292cf1

Memory Management

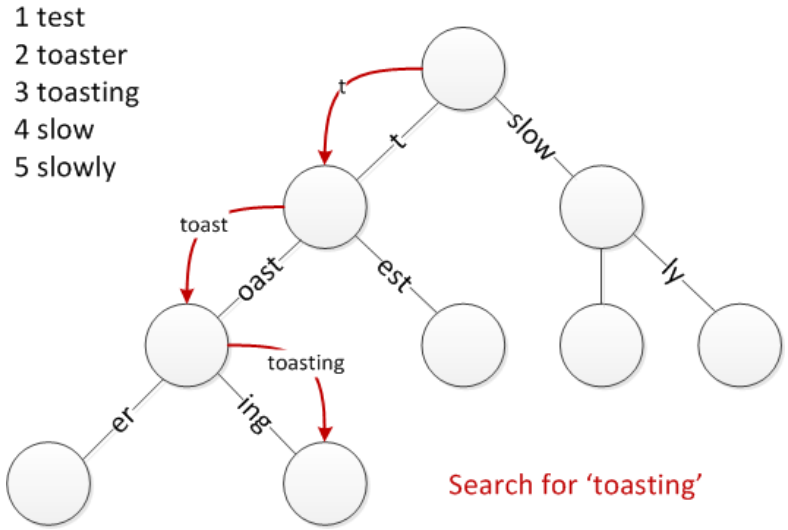
- Radix Tree
  - Trie, a specialized search tree data structure used to store and retrieve strings
  - Root does not store any character, each edge represents a character, and any path from the root to a node corresponds to a prefix of some string



A trie for keys “A”, “to”, “tea”, “ted”, “ten”, “i”, “in”, and “inn”. Each complete English word has an arbitrary integer value associated with it.

Memory Management

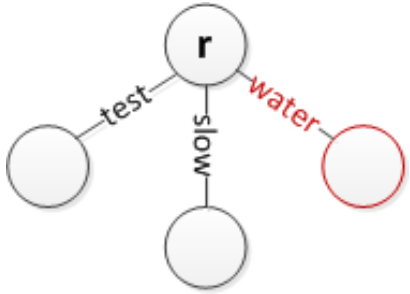
- Radix Tree
  - A space-optimized trie (prefix tree)
    - The number of children of every internal node is at most the radix r of this tree
    - If a node is the only child, it will be merged to its parent; an edge contains one or more characters
    - Efficient for small sets and for sets of strings that share long prefixes



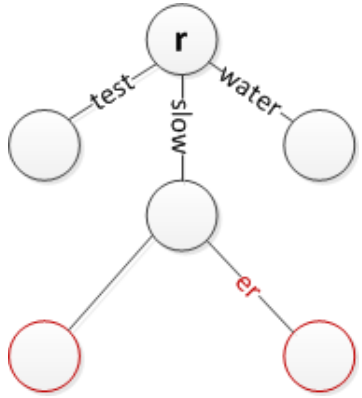
An illustration of storing five words and querying “toasting”

Memory Management

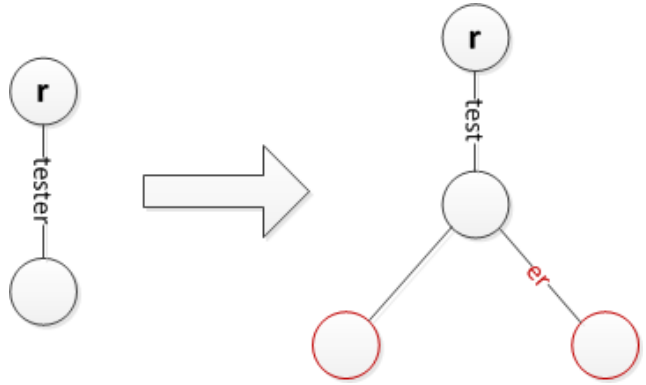
- Radix Tree
  - Insert and delete



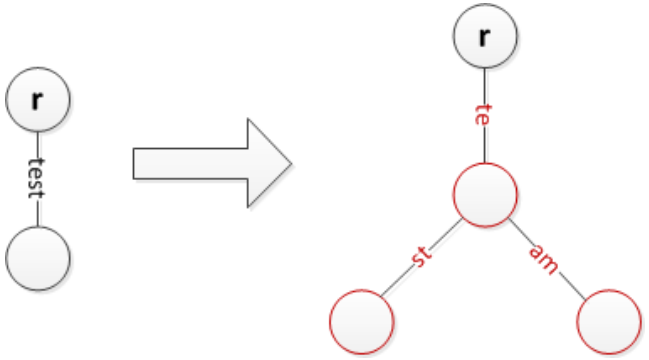
Insert 'water' at the root



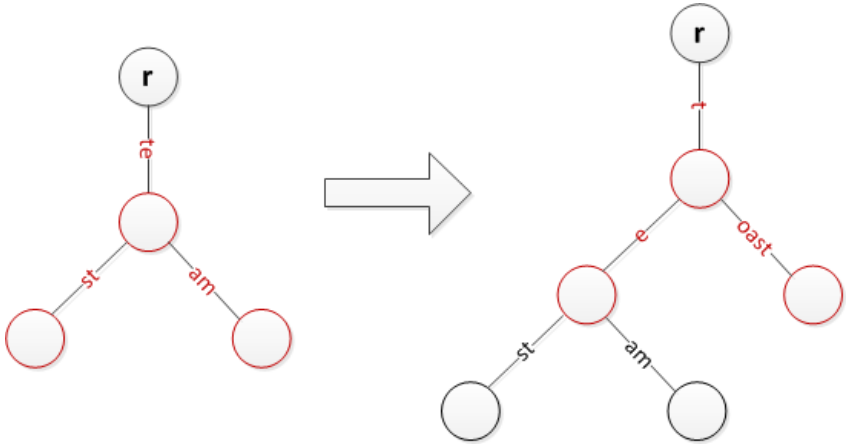
Insert 'slower' while keeping 'slow'



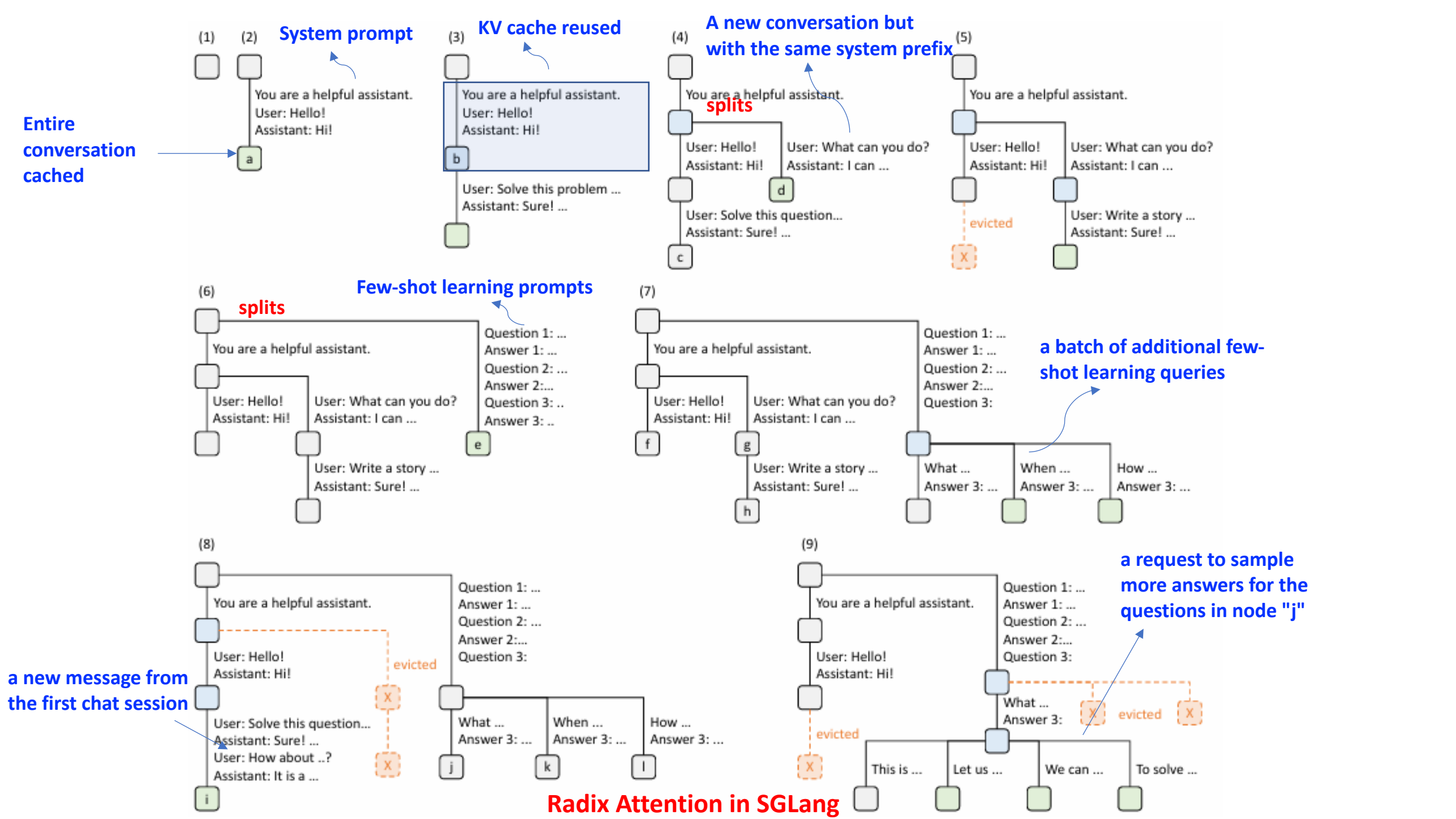
Insert 'test' which is a prefix of 'tester'



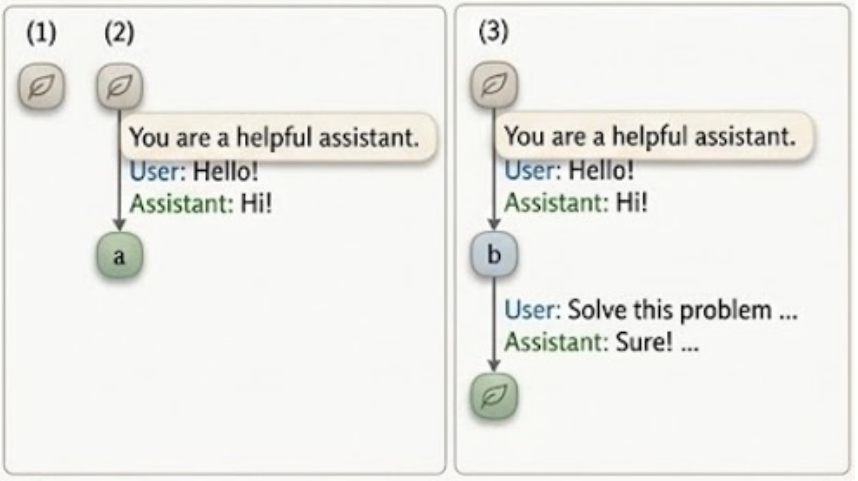
Insert 'team' while splitting 'test'
and creating a new edge label 'st'



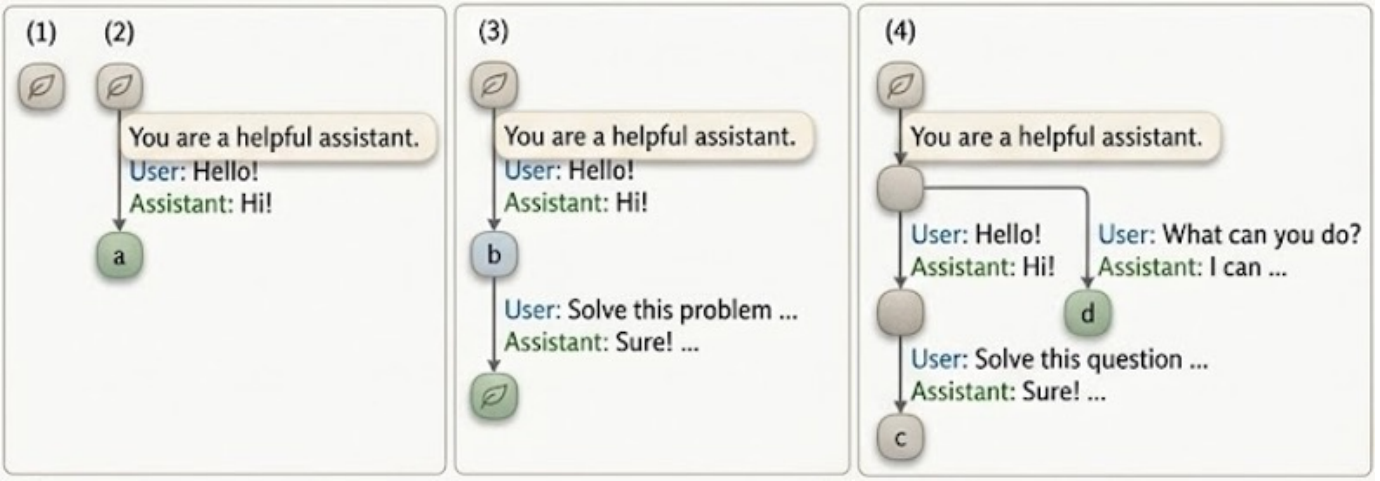
Insert 'toast' while splitting 'te' and
moving previous strings a level lower


Entire conversation cached
(1)
(2)
System prompt
You are a helpful assistant.
User: Hello!
Assistant: Hi!
a
(3)
KV cache reused
You are a helpful assistant.
User: Hello!
Assistant: Hi!
b
User: Solve this problem ...
Assistant: Sure! ...
(4)
A new conversation but with the same system prefix
You are a helpful assistant.
splits
User: Hello!
Assistant: Hi!
User: What can you do?
Assistant: I can ...
d
User: Solve this question...
Assistant: Sure! ...
c
(5)
You are a helpful assistant.
User: Hello!
Assistant: Hi!
User: What can you do?
Assistant: I can ...
evicted
X
User: Write a story ...
Assistant: Sure! ...
(6)
splits
Few-shot learning prompts
You are a helpful assistant.
User: Hello!
Assistant: Hi!
User: What can you do?
Assistant: I can ...
User: Write a story ...
Assistant: Sure! ...
Question 1: ...
Answer 1: ...
Question 2: ...
Answer 2:...
Question 3: ..
Answer 3: ..
e
(7)
You are a helpful assistant.
User: Hello!
Assistant: Hi!
User: What can you do?
Assistant: I can ...
f
g
User: Write a story ...
Assistant: Sure! ...
h
Question 1: ...
Answer 1: ...
Question 2: ...
Answer 2:...
Question 3:
a batch of additional few-shot learning queries
What ...
Answer 3: ...
When ...
Answer 3: ...
How ...
Answer 3: ...
(8)
a new message from the first chat session
You are a helpful assistant.
User: Hello!
Assistant: Hi!
evicted
X
User: Solve this question...
Assistant: Sure! ...
User: How about ..?
Assistant: It is a ...
i
Question 1: ...
Answer 1: ...
Question 2: ...
Answer 2:...
Question 3:
What ...
Answer 3: ...
When ...
Answer 3: ...
How ...
Answer 3: ...
j
k
l
(9)
You are a helpful assistant.
User: Hello!
Assistant: Hi!
evicted
X
Question 1: ...
Answer 1: ...
Question 2: ...
Answer 2:...
Question 3:
What ...
Answer 3:
X
evicted
X
a request to sample more answers for the questions in node "j"
This is ...
Let us ...
We can ...
To solve ...
Radix Attention in SGLang

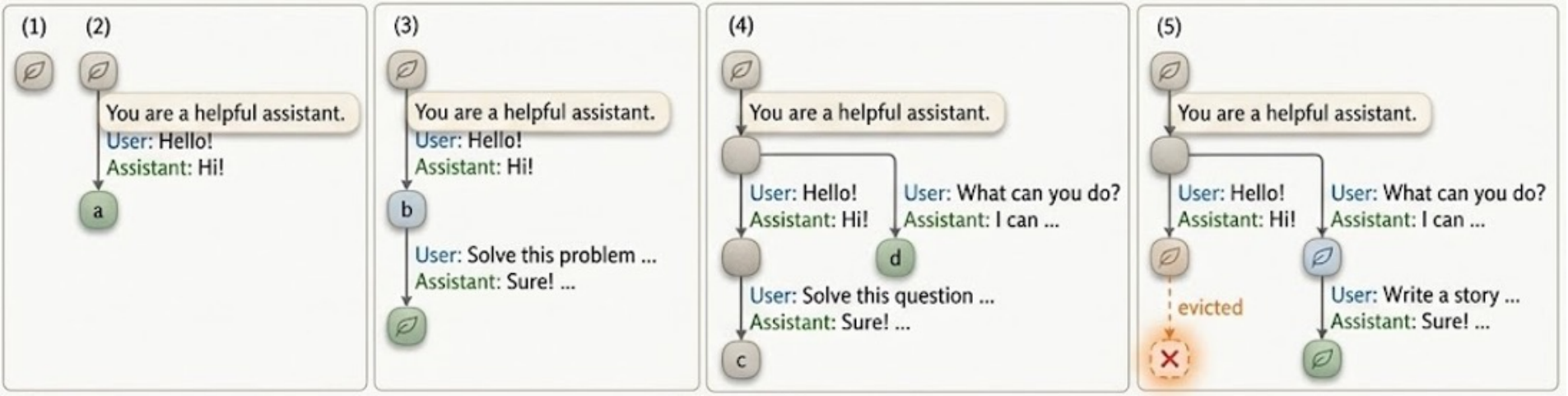
Step 1: the radix tree is initially **empty**; Step 2: the server processes an incoming user message "Hello" and responds with the LLM output "Hi". The **system prompt** "You are a helpful assistant", the **user message** "Hello!", and **the LLM reply** "Hi!" are consolidated into the tree as a single edge linked to a **new node**.



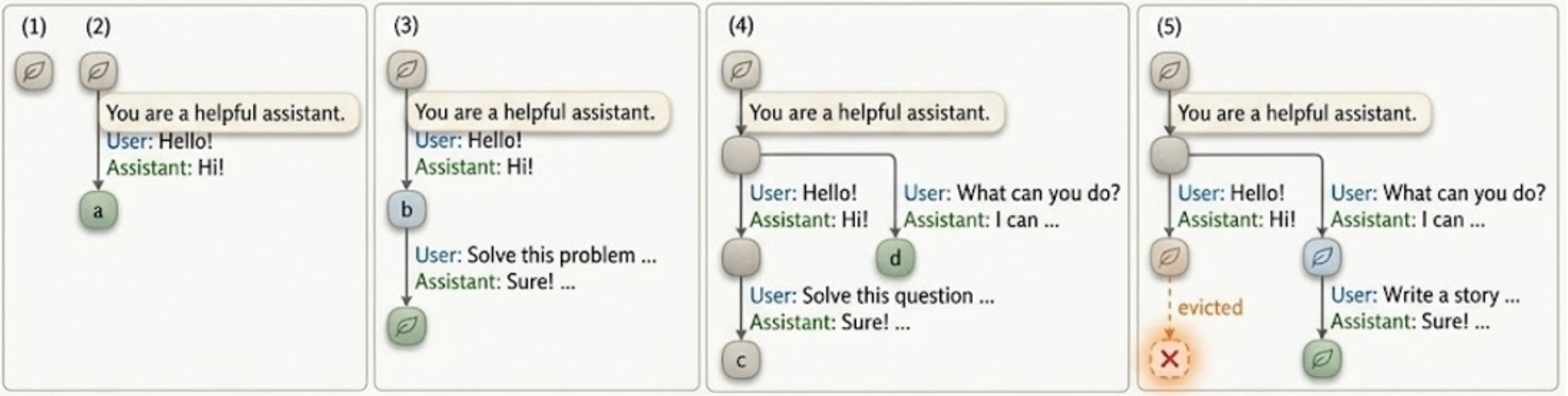
Step 3: a new prompt arrives and the server finds the prefix of the prompt in the radix tree and reuses its KV cache. The new turn is appended to the tree as a new node.

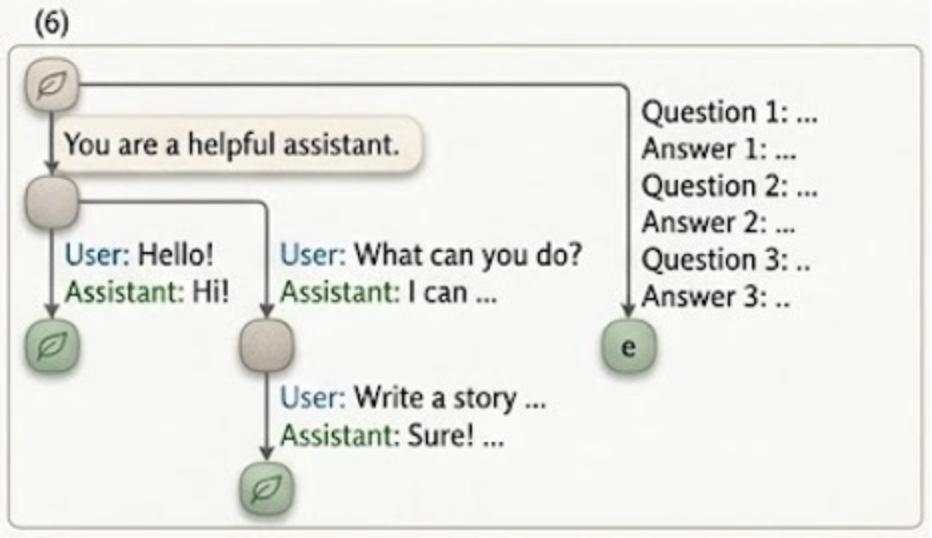


Step 4: a new chat session begins. The node ``b'' from (3) is split into two nodes to allow the two chat sessions to share the system prompt.

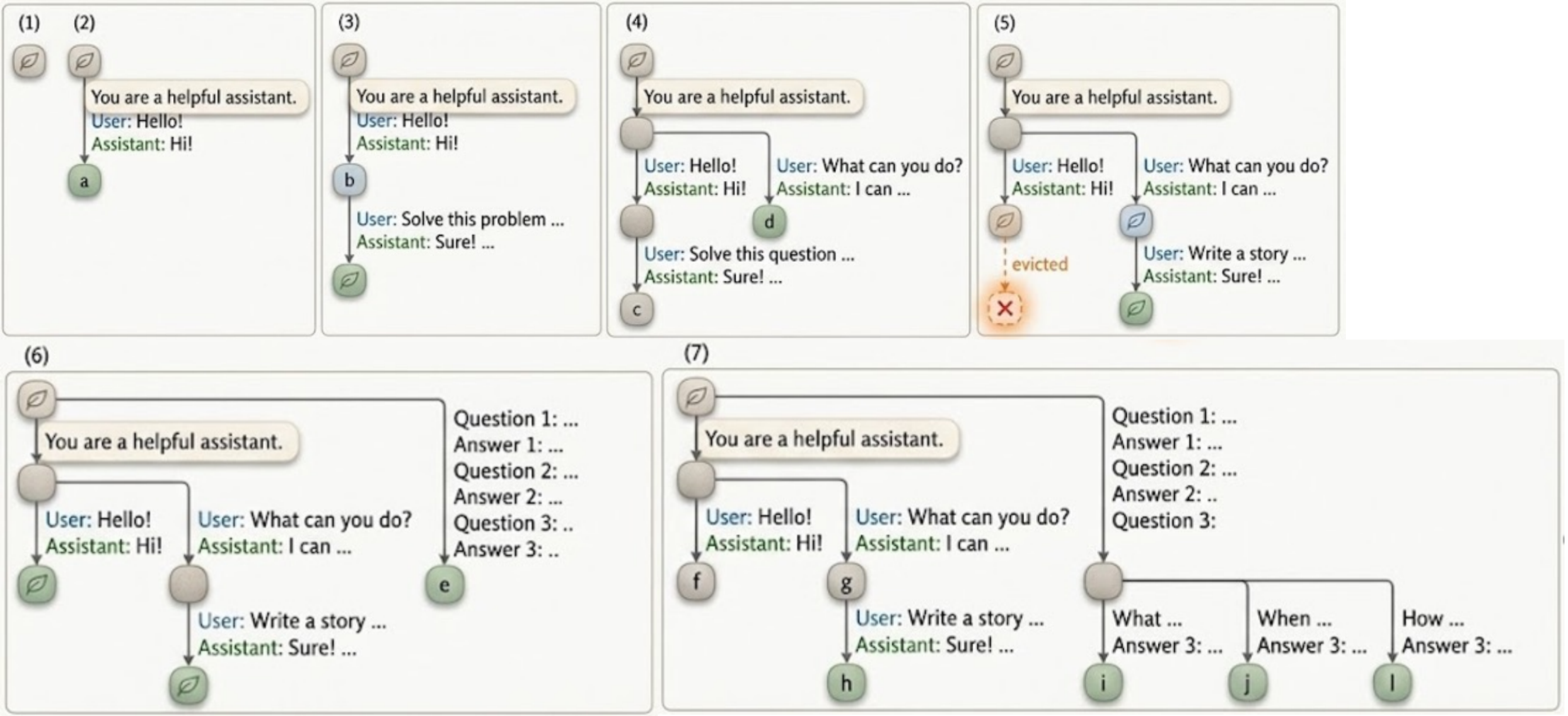


Step 5: the second chat session continues. However, due to the memory limit, node "c" from (4) must be evicted.

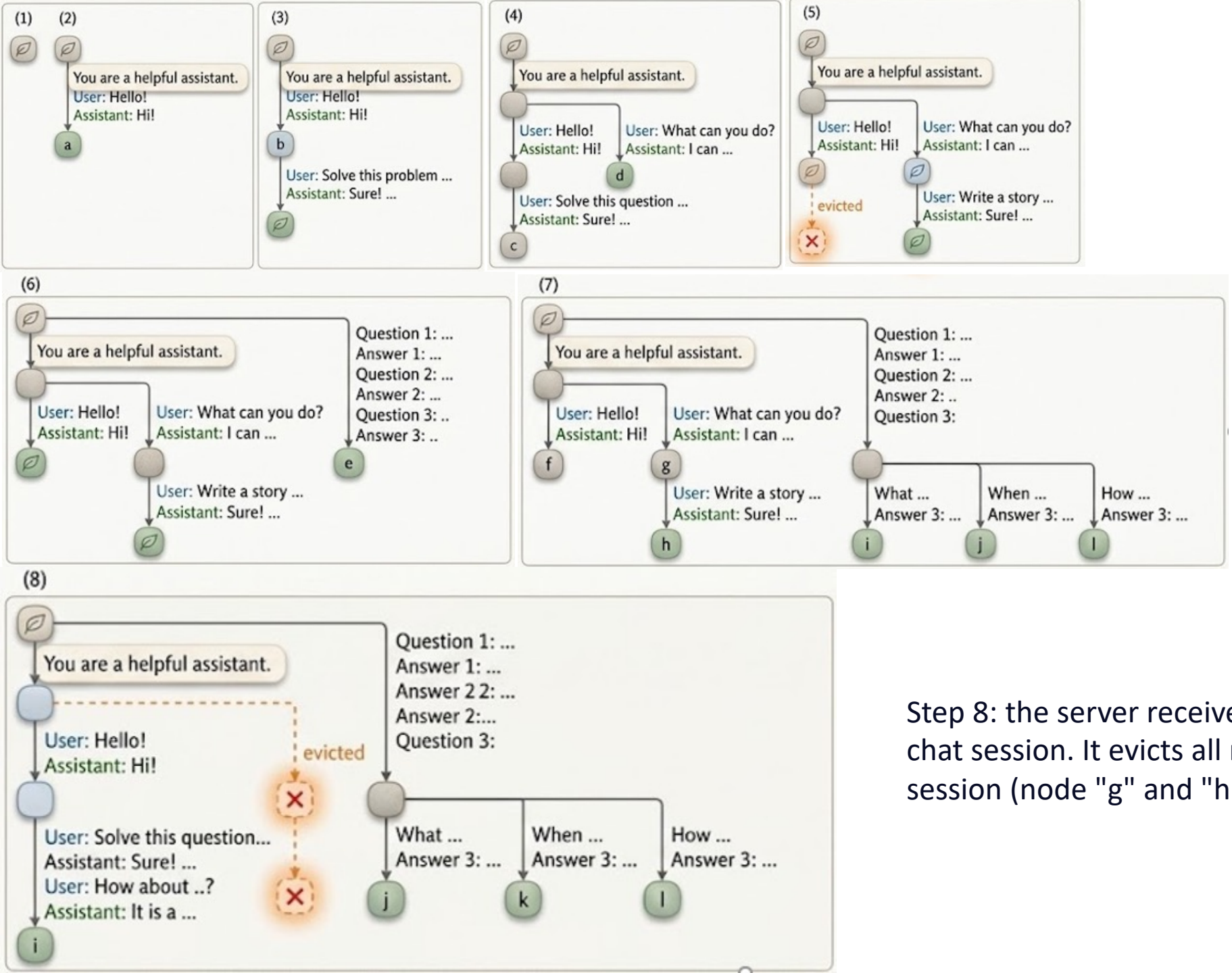




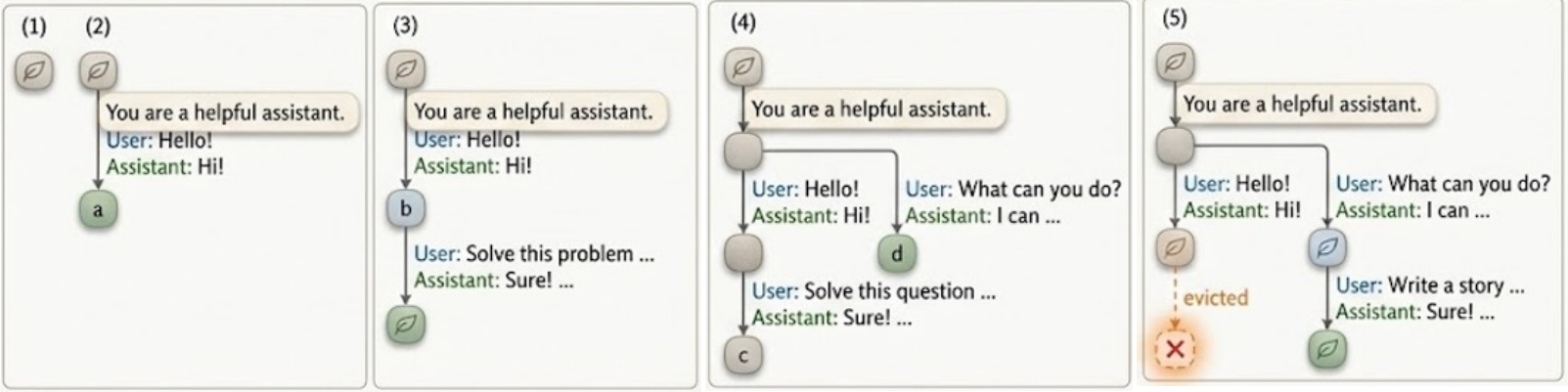
Step 6: the server receives a few-shot learning query, processes it, and inserts it into the tree. The root node is split because the new query does not share any prefix with existing nodes.

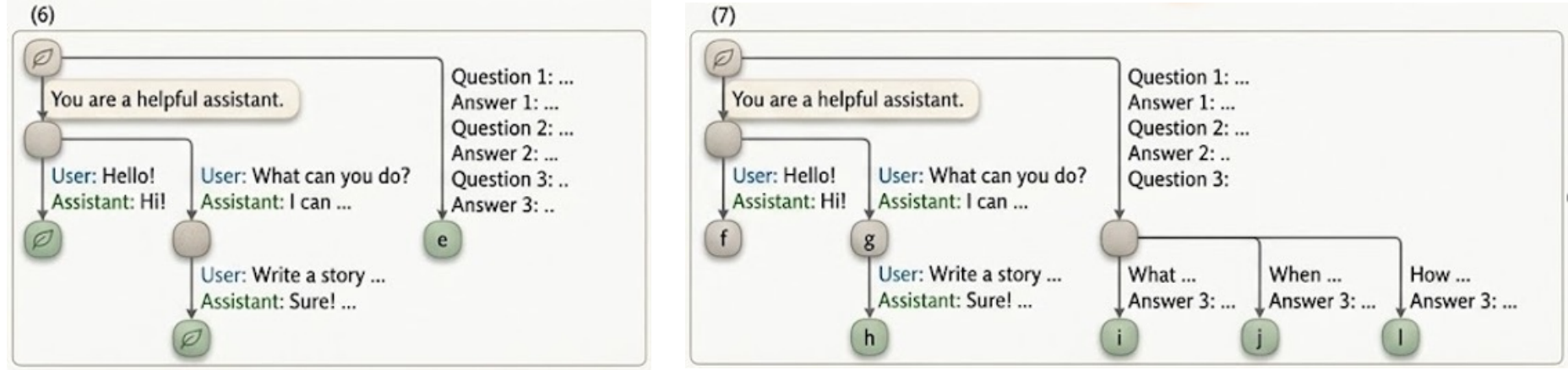


Step 7: the server receives a batch of additional few-shot learning queries. These queries share the same set of few-shot examples, so we split node 'e' from (6) to enable sharing.

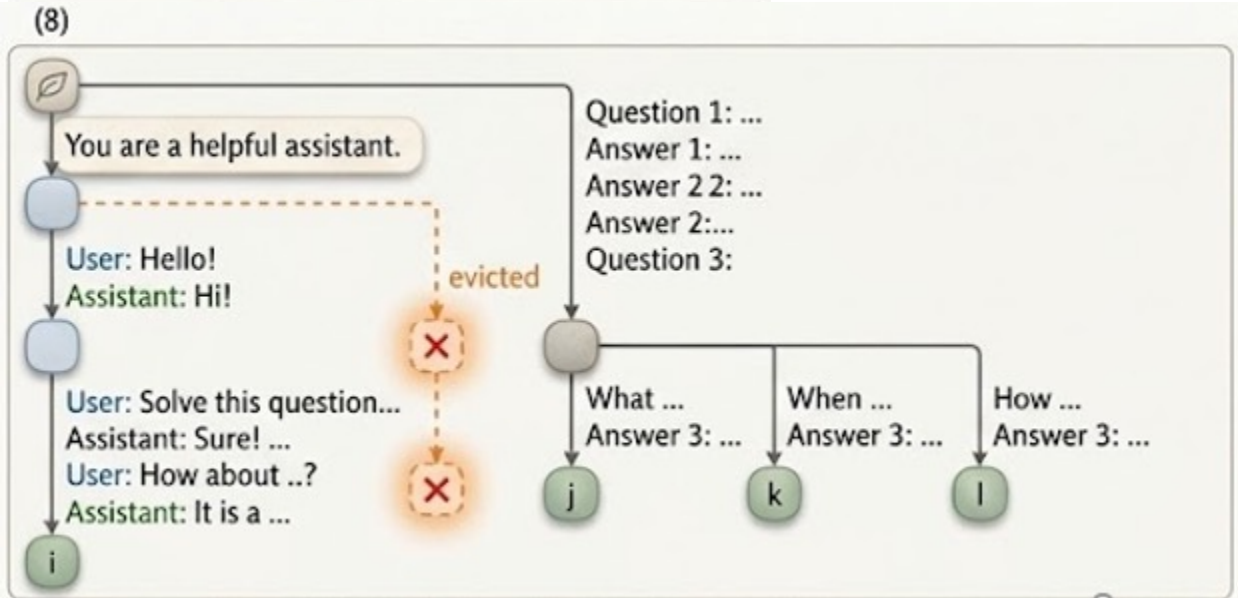


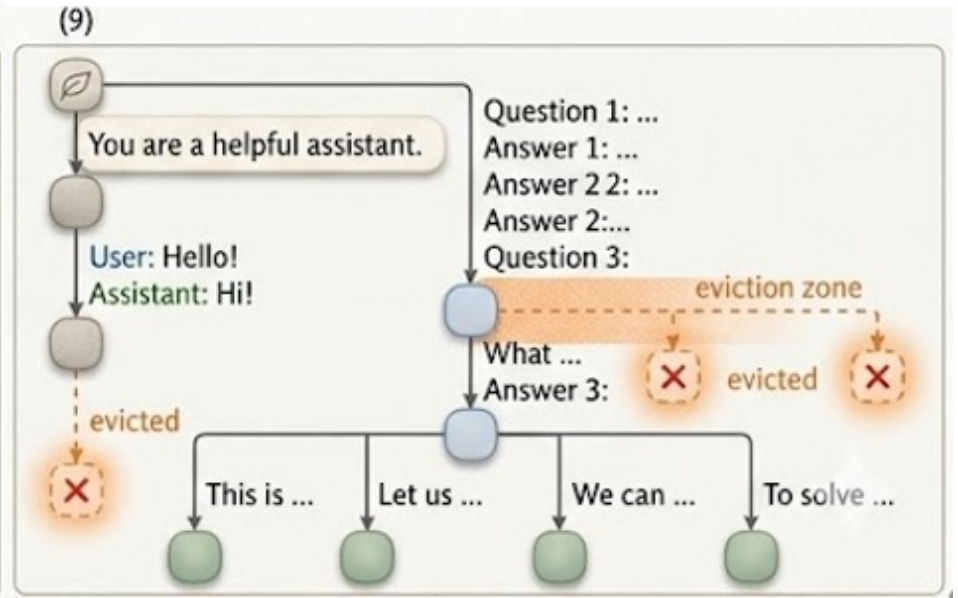
Step 8: the server receives a new message from the first chat session. It evicts all nodes from the second chat session (node "g" and "h") as they are least recently used.





Step 9: the server receives a request to sample more answers for the questions in node "j" from (8)

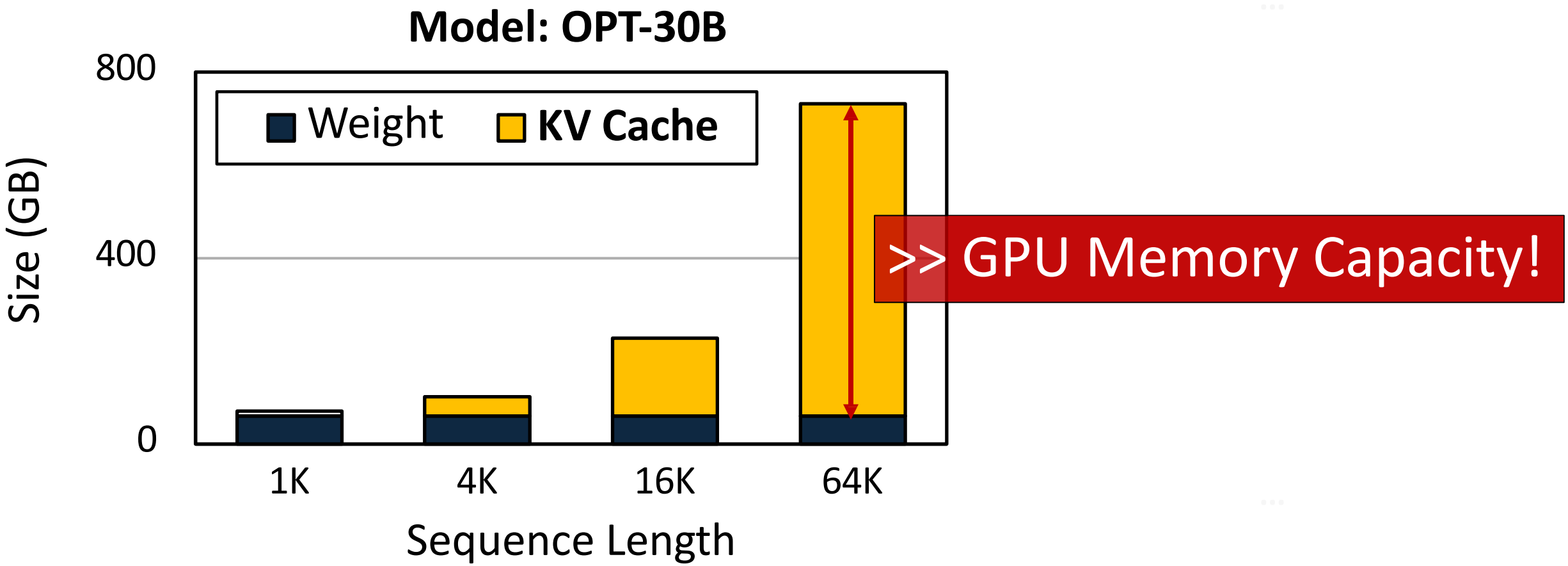




Inference Optimization: Outline

- Overview
- Attention Optimization
- Continuous Batching
- KV Cache Optimization
  - Model Architecture Optimization
  - KV Cache Data Structure Optimization
  - **KV Cache Memory Optimization**
- Speculative Decoding
- Distributed Serving (Extended Learning)

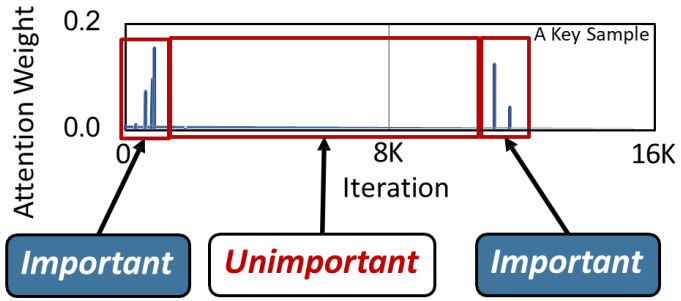
Memory Optimization

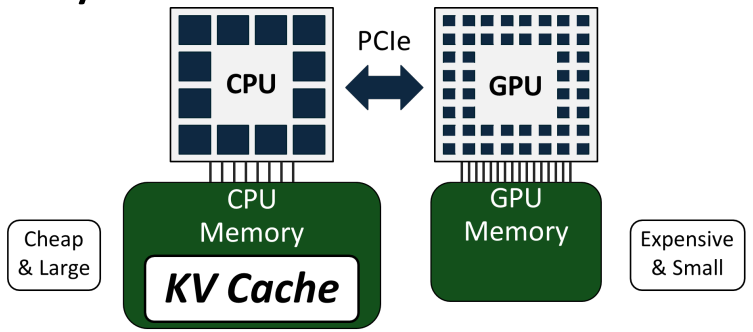


Source: InfiniGen Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management

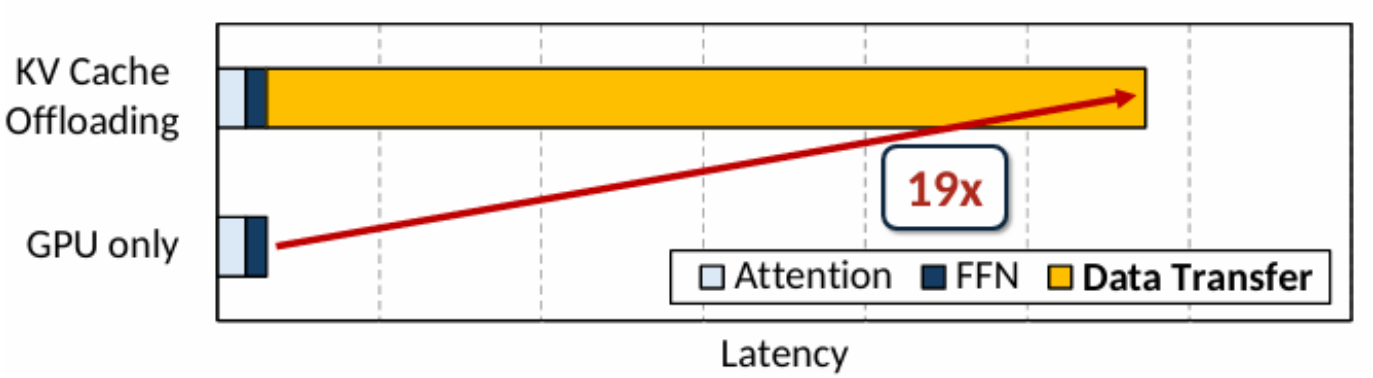
Memory Optimization

- Eviction
  - Permanent eviction: simple, but may cause an accuracy drop
  - Temporary eviction: offloads to CPU memory or SSD with limited bandwidth





  - ***Not*** all KV cache data needs to stay in GPU memory at all times



Source: InfiniGen Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management

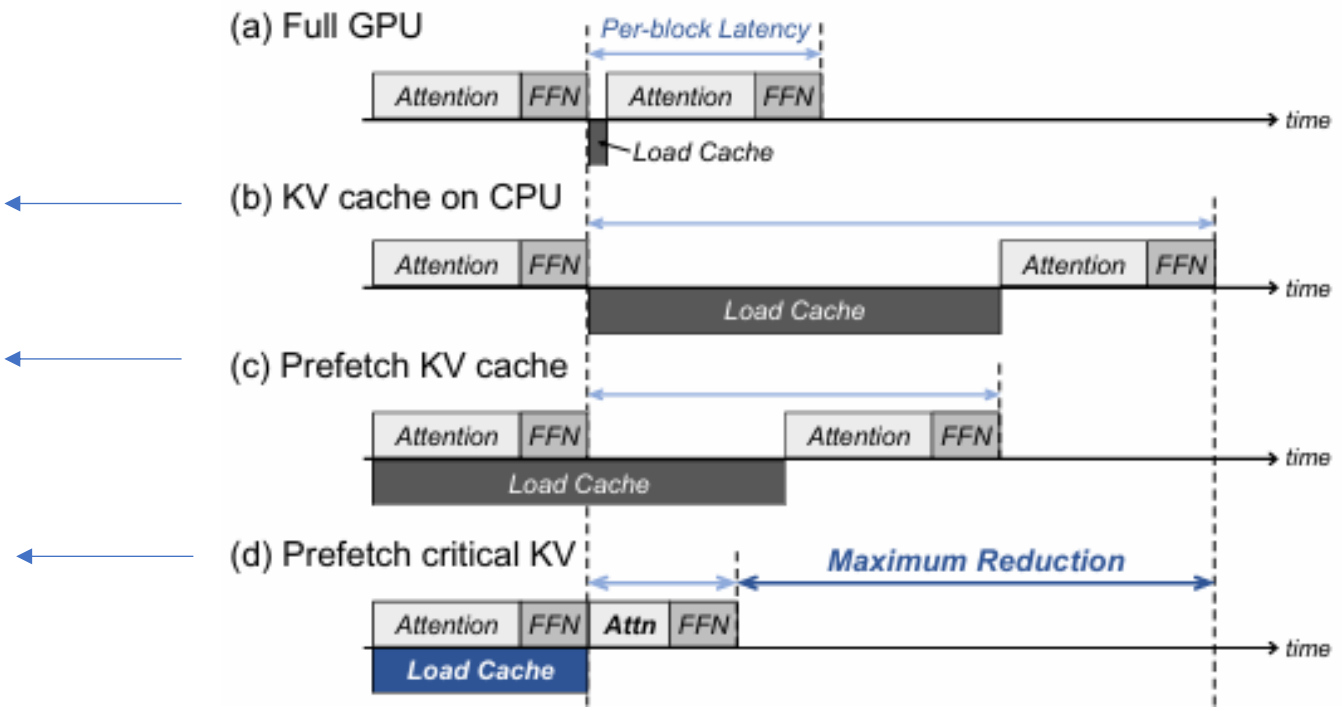
KV Cache Offload: InfiniGen

- Idea KV Cache Offloading

Naïve offloading blocks the computation

Prefetching large KV matrices cannot fully overlap communication and computation

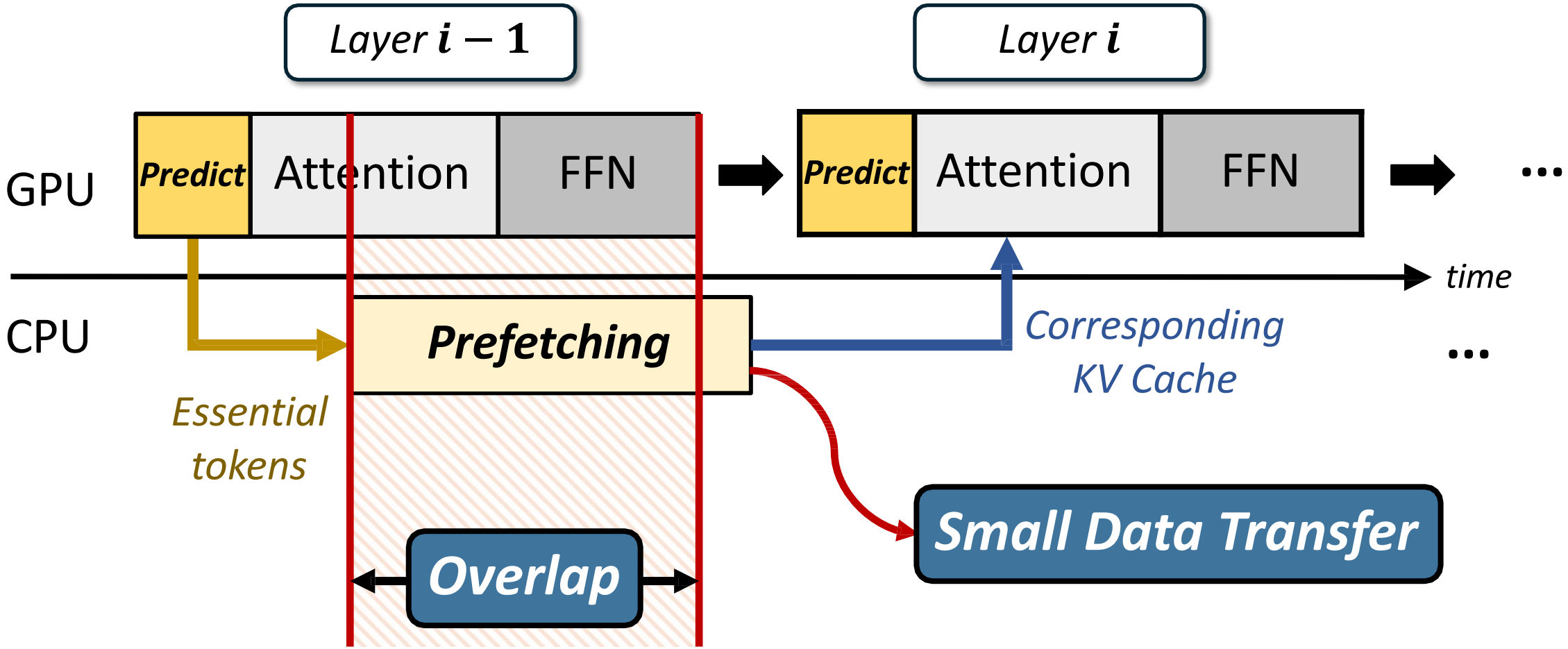
Only prefetching important KV pairs



A high-level illustration of the key idea

Source: InfiniGen: Efficient generative inference of large language models with dynamic {KV} cache management

KV Cache Offload: InfiniGen



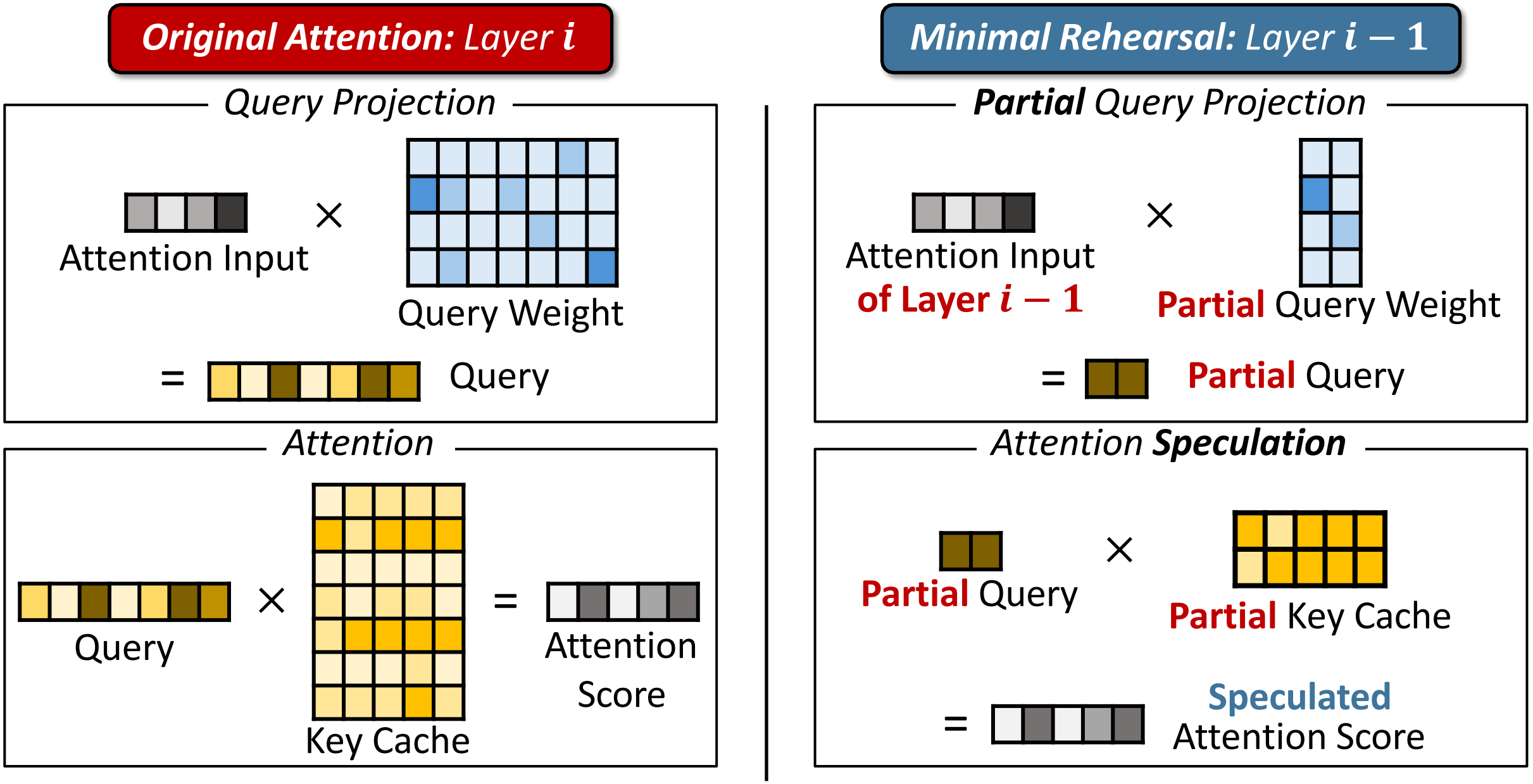
Source: InfiniGen: Efficient generative inference of large language models with dynamic KV cache management

KV Cache Offload: InfiniGen

Q: Using attention pattern of layer i to estimate that of layer i+1?

A: NO! The tokens deemed unimportant in the current iteration could become important in subsequent iterations.

Focusing on the key idea for general KV Cache evictions!



Speculative KV prefetching: predicting the **important** tokens (Extended Learning)

Source: InfiniGen: Efficient generative inference of large language models with dynamic {KV} cache management

Original Attention: Layer i

Minimal Rehearsal: Layer $i-1$

Query Projection

Attention Input

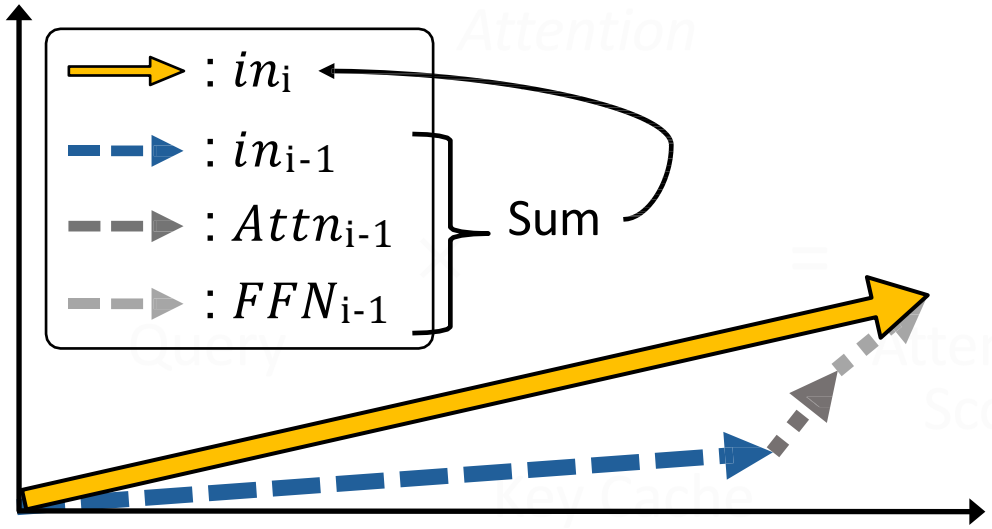
Highly Similar!

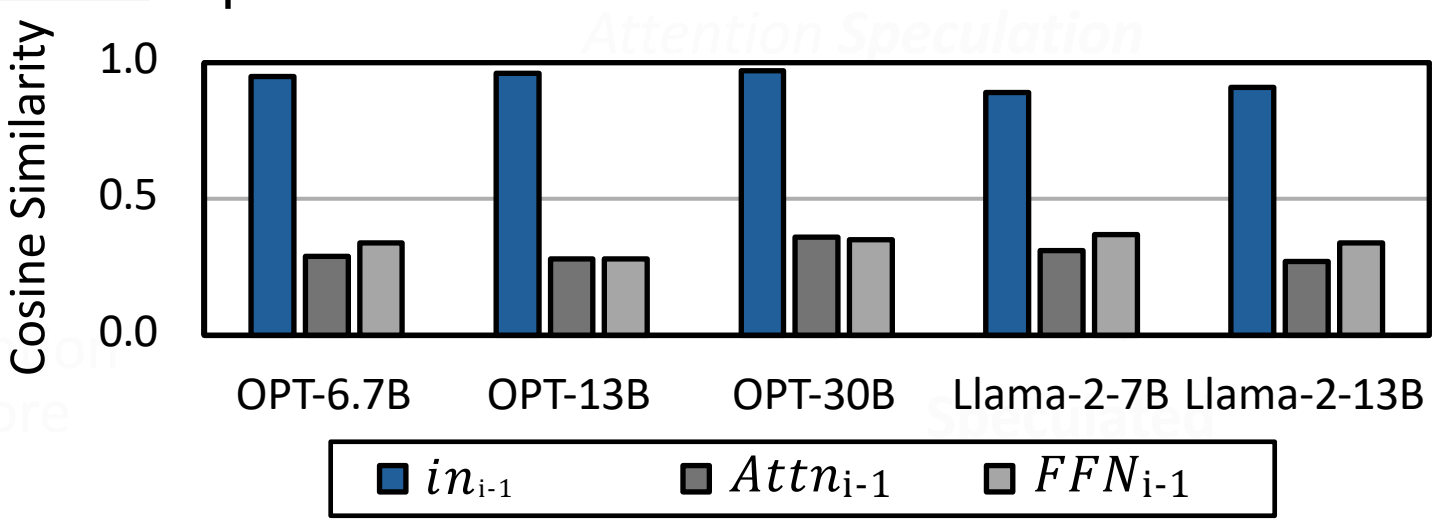
(Extended Learning)

Partial Query Projection

Attention Input
of Layer $i-1$

(Extended Learning)

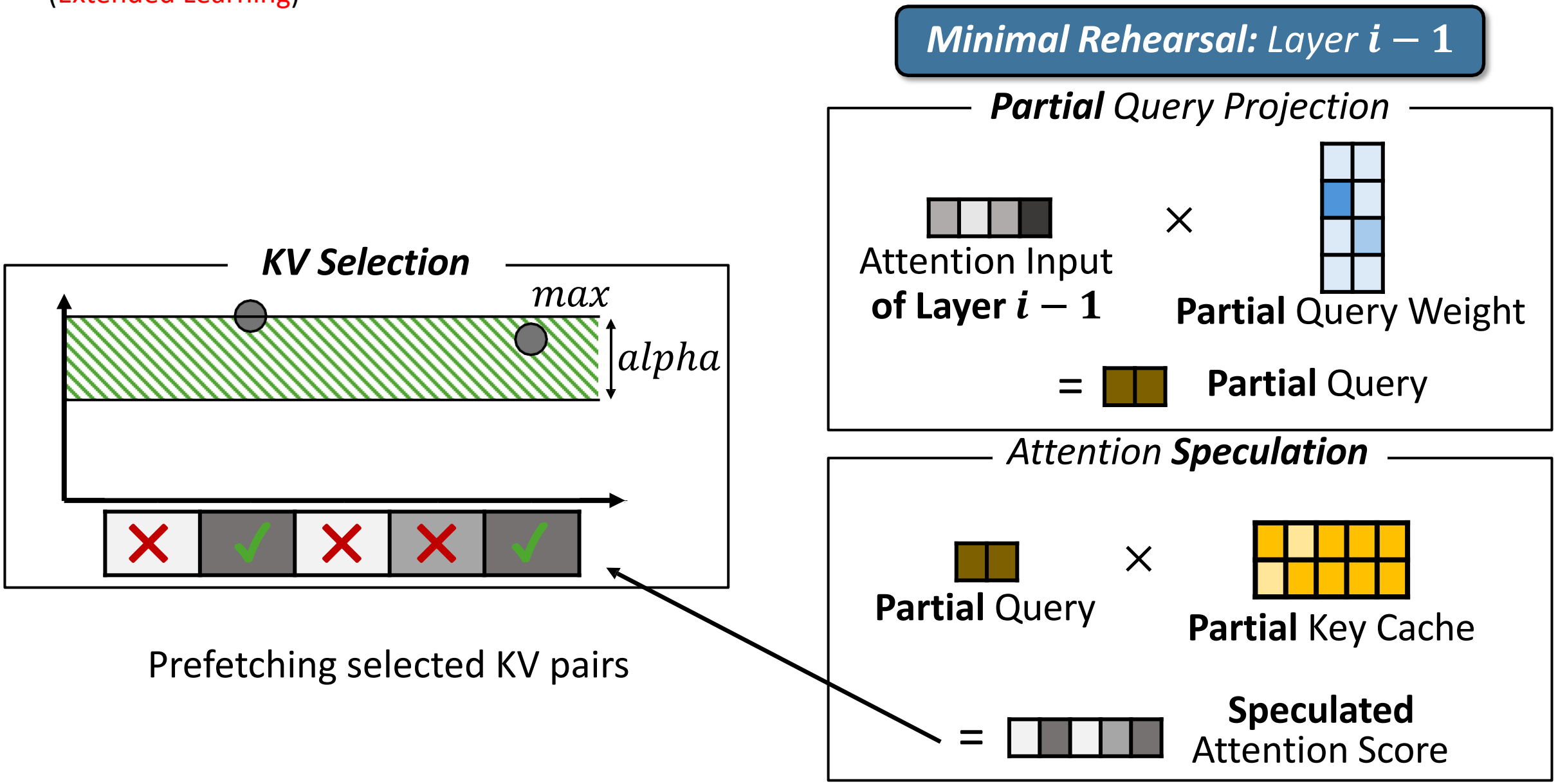




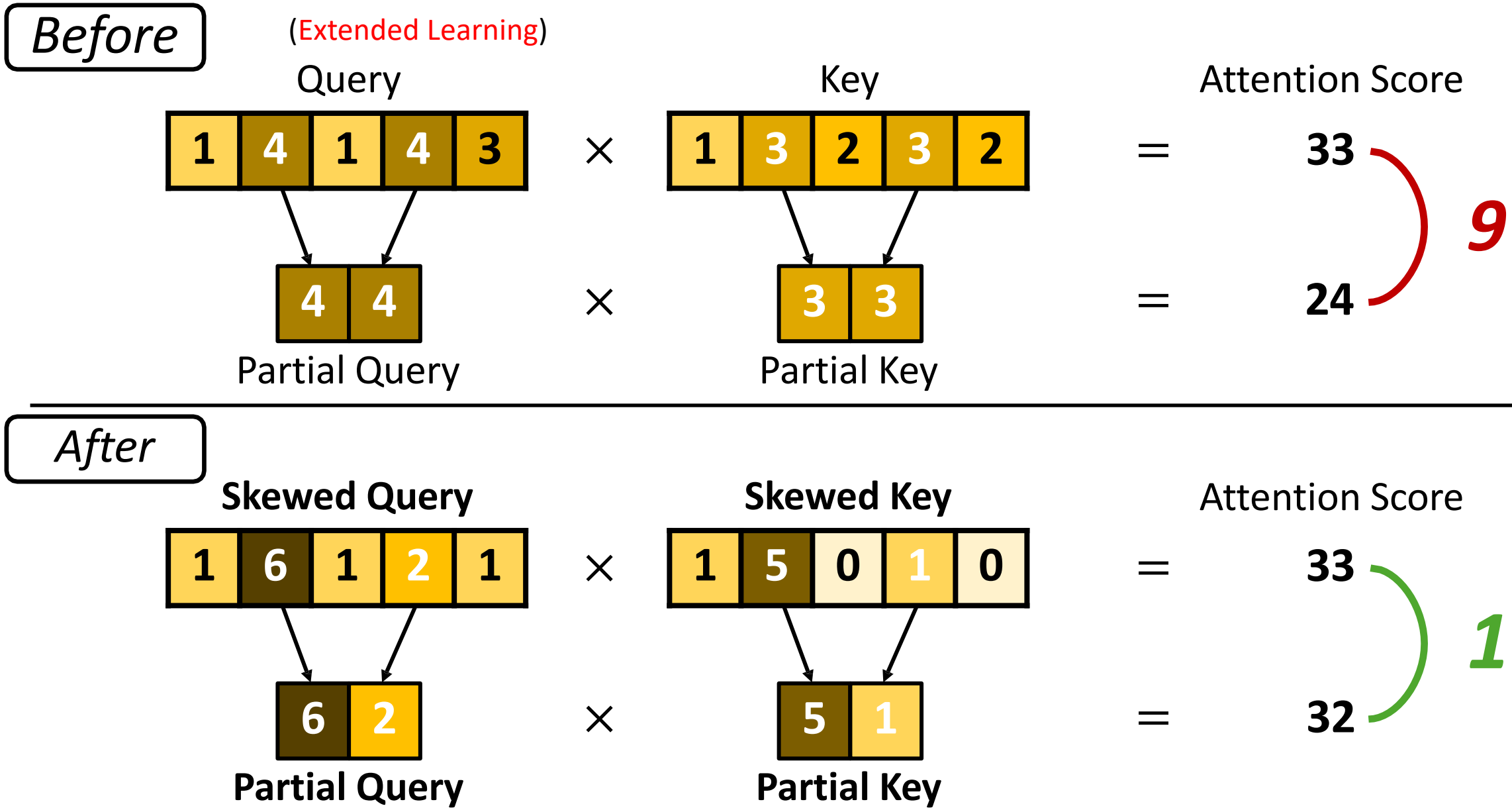
Observation: attention inputs of consecutive attention layers are highly similar

Source: InfiniGen: Efficient generative inference of large language models with dynamic {KV} cache management

(Extended Learning)



Source: InfiniGen: Efficient generative inference of large language models with dynamic {KV} cache management



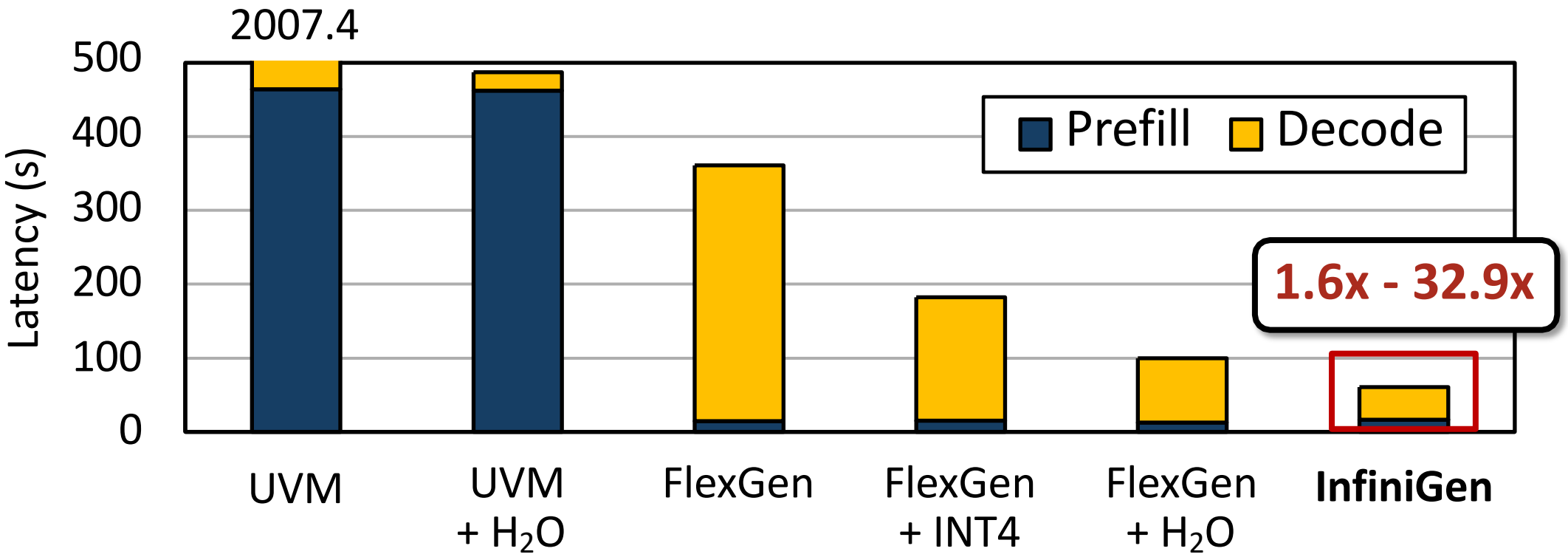
Key/Query skewing using singular value decomposition

Source: InfiniGen: Efficient generative inference of large language models with dynamic {KV} cache management

(Extended Learning)

KV Cache Offload: InfiniGen

- Experimental results: OPT and LLama2 on NVIDIA RTX A6000 with PCIe 3.0 Gen

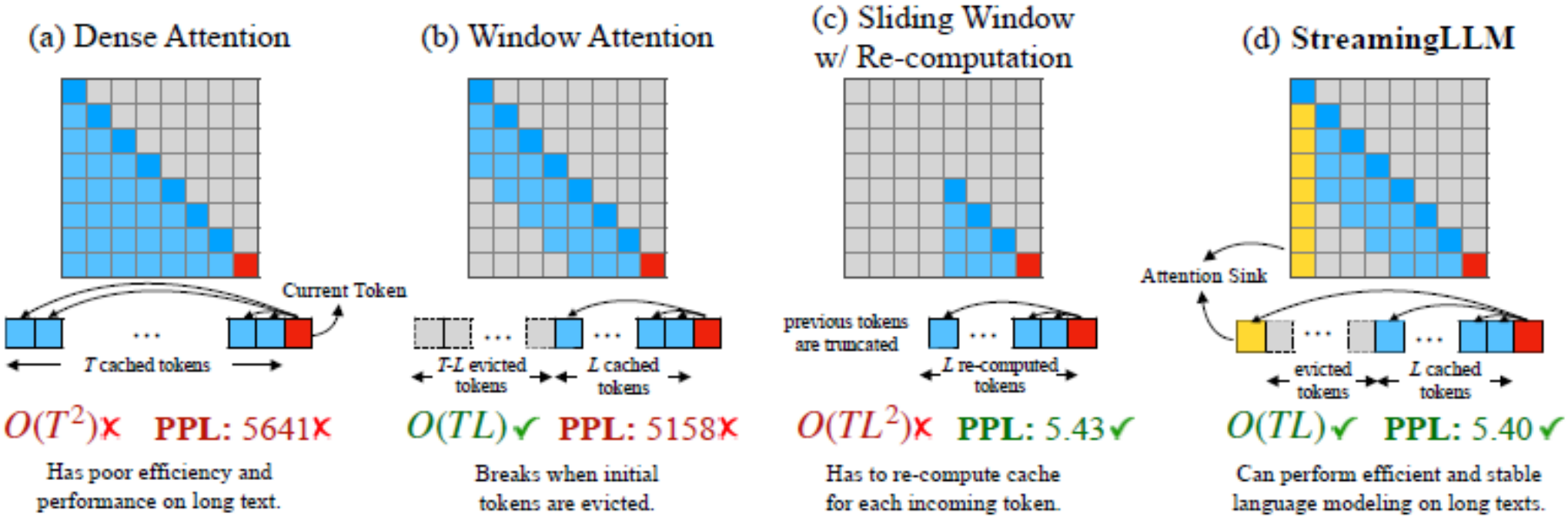


Source: InfiniGen: Efficient generative inference of large language models with dynamic {KV} cache management

(Extended Learning)

KV Cache Eviction: StreamingLLM

- Inference for Infinite Token Length
  - Insufficient GPU HBM, and high latency of KV cache offloading
  - Poor perplexity (PPL) for truncated attention methods (e.g. evicting old KV)

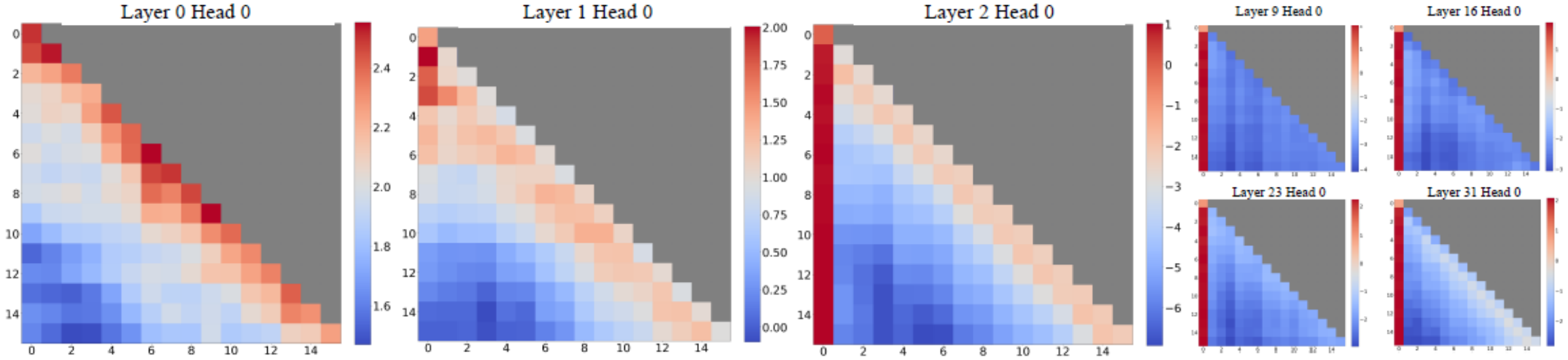


Source: Efficient Streaming Language Models with Attention Sinks

(Extended Learning)

KV Cache Eviction: StreamingLLM

- Key Insights
  - Initial tokens have large attention scores, even if not semantically significant
  - ***Attention sinks***: tokens that disproportionately attract attention irrespective of their relevance
  - Attributed to ***Softmax*** of autoregressive language modeling



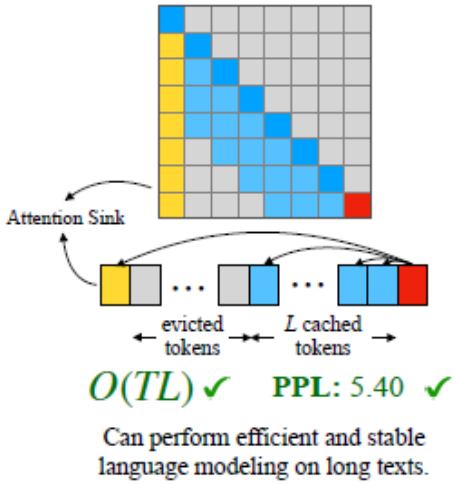
Visualization of the average attention logits in Llama-2-7B over 256 sentences

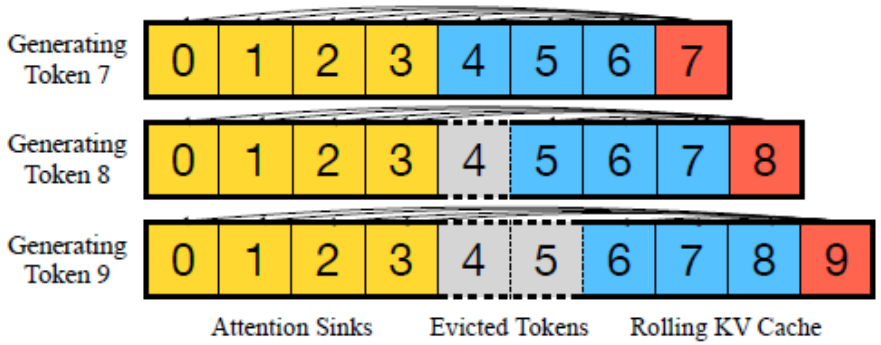
Source: Efficient Streaming Language Models with Attention Sinks

(Extended Learning)

KV Cache Eviction: StreamingLLM

- Serving Infinite Streams
  - Preserve the KV of attention sink tokens, keep the KV of tokens in the sliding window, and evict remaining tokens





An illustration of KV cache eviction

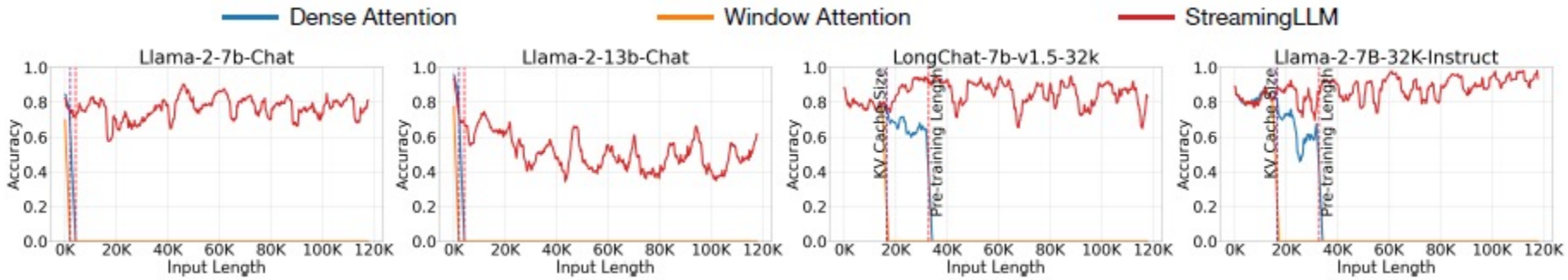
  - Adjust the position of tokens. StreamingLLM focuses on positions within the cache rather than those in the original text

Source: Efficient Streaming Language Models with Attention Sinks

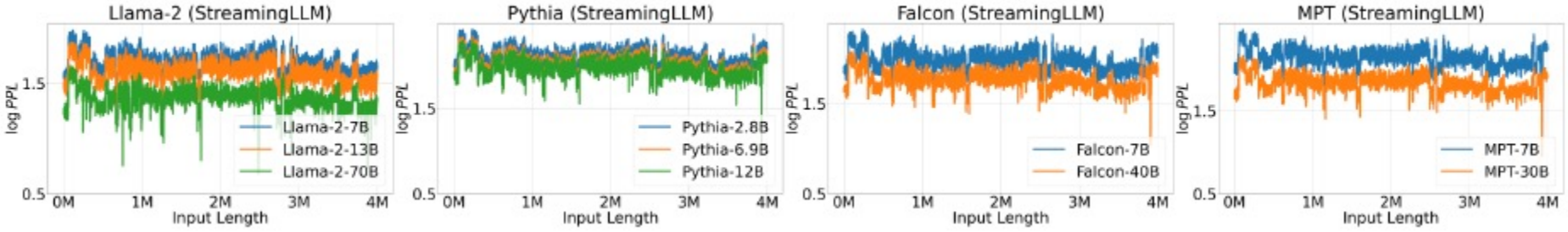
(Extended Learning)

KV Cache Eviction: StreamingLLM

- Serving Infinite Streams



Accuracy on the StreamEval benchmark

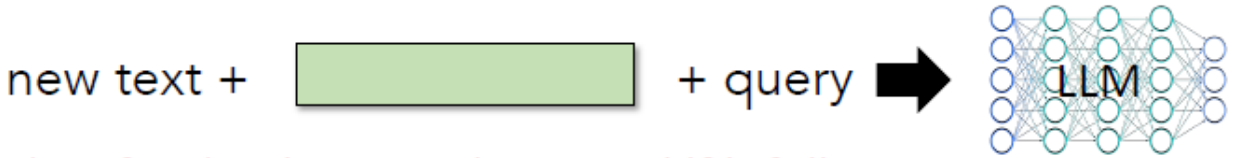


Language modeling perplexity of StreamingLLM on super long texts with 4 million tokens

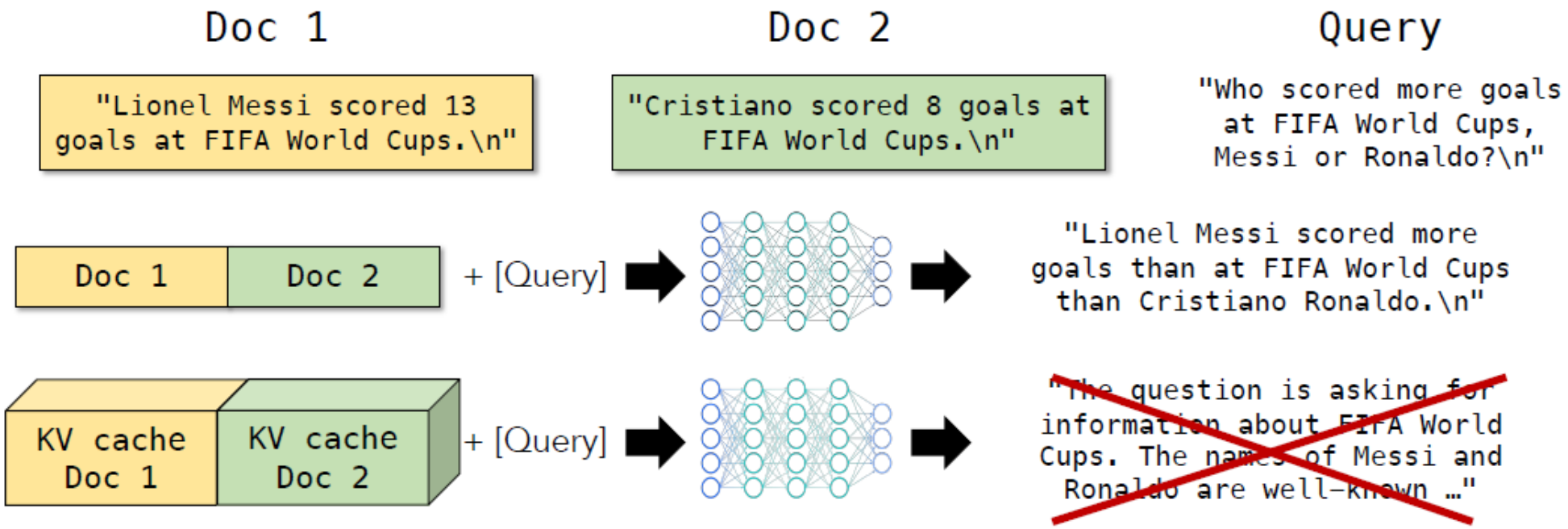
(Extended Learning)

KV Cache for RAG: CacheBlend

- RAG: repetitive token sequences, but without common prefix



KV cache of a chunk cannot be reused if it follows some new text

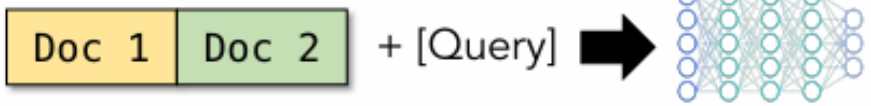


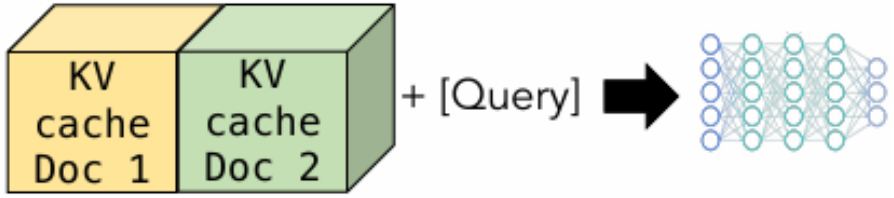
Source: Faster Large Language Model Serving with Knowledge Delivery Networks, By Jiang Junchen@APNet2024

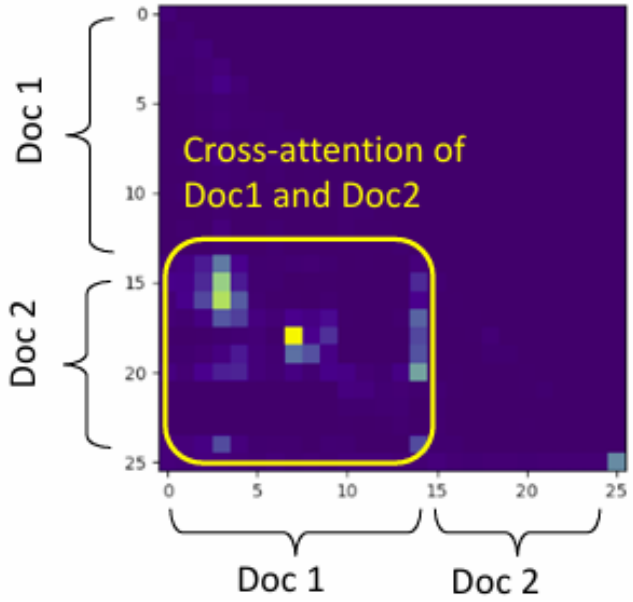
(Extended Learning)

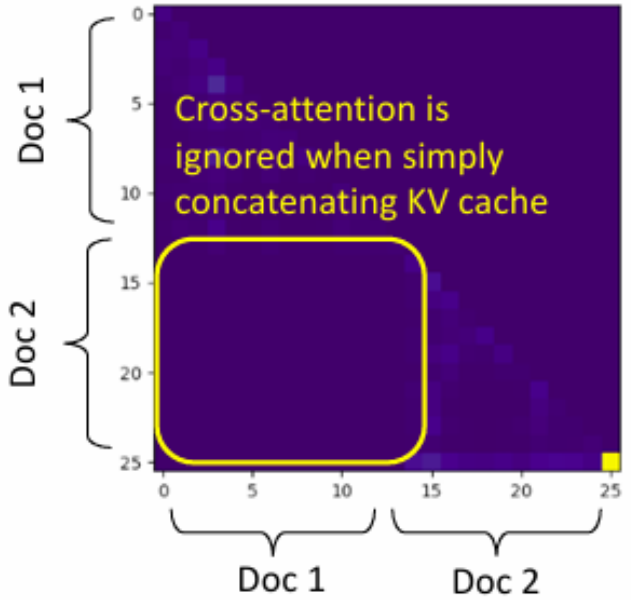
KV Cache for RAG: CacheBlend

- CacheBlend for RAG
  - Missing cross-attention (think why?)







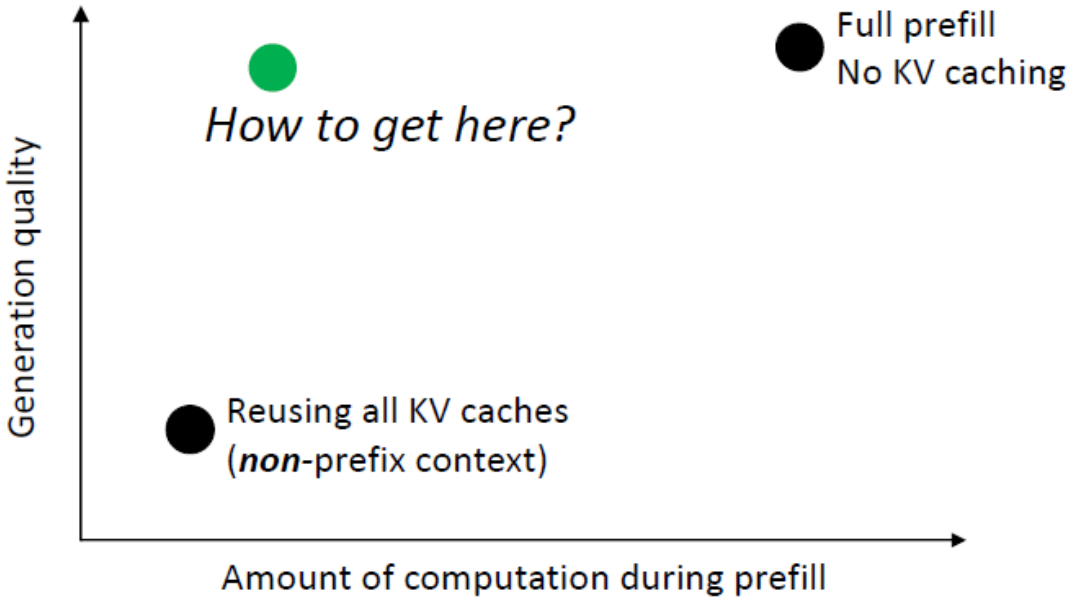


Source: Faster Large Language Model Serving with Knowledge Delivery Networks, By Jiang Junchen@APNet2024

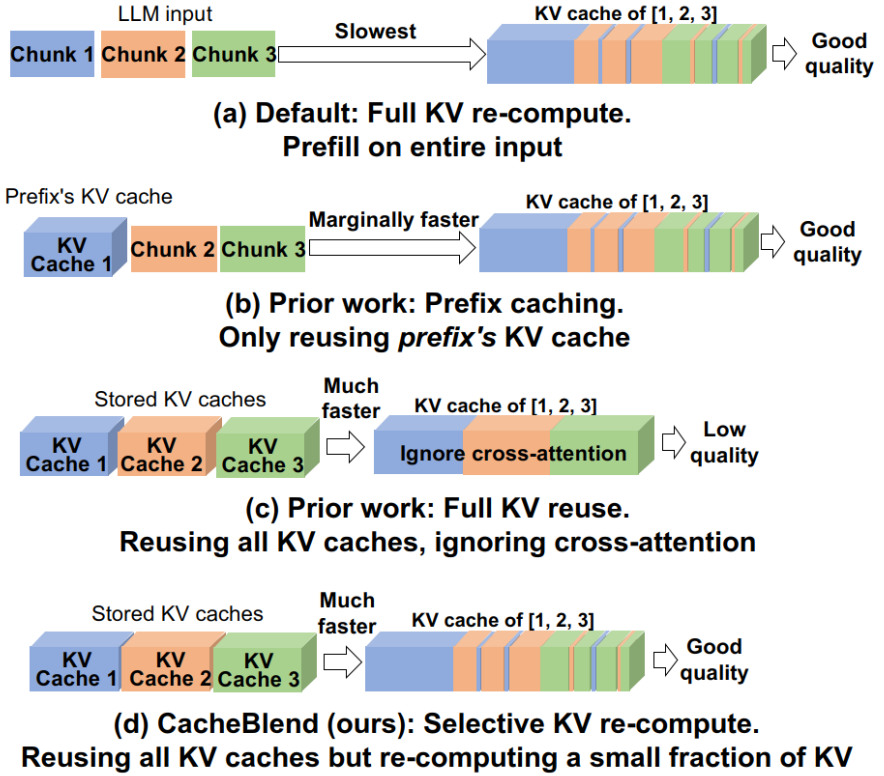
(Extended Learning)

KV Cache for RAG: CacheBlend

- CacheBlend for RAG
  - Selectively recomputing KV



Challenge and goal



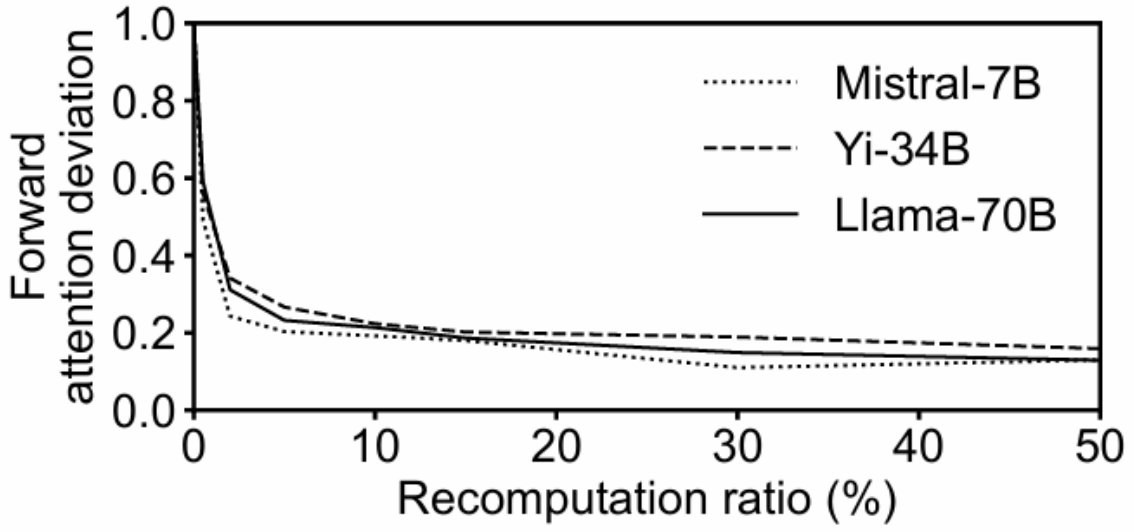
Contrasting full KV recompute, prefix caching, full KVreuse, and CacheBlend's selective KV recompute

Source: Faster Large Language Model Serving with Knowledge Delivery Networks, By Jiang Junchen@APNet2024

(Extended Learning)

KV Cache for RAG: CacheBlend

- Selectively recomputing KV
  - Key insight: ***sparsity*** of attention matrix and ***similarity*** of attention matrices on two consecutive layers
  - Attention deviation reduces as KV of more tokens on each layer are recomputed

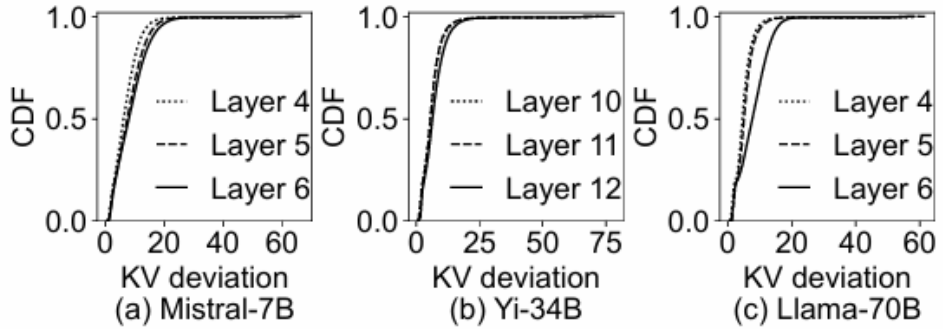


Source: Faster Large Language Model Serving with Knowledge Delivery Networks, By Jiang Junchen@APNet2024

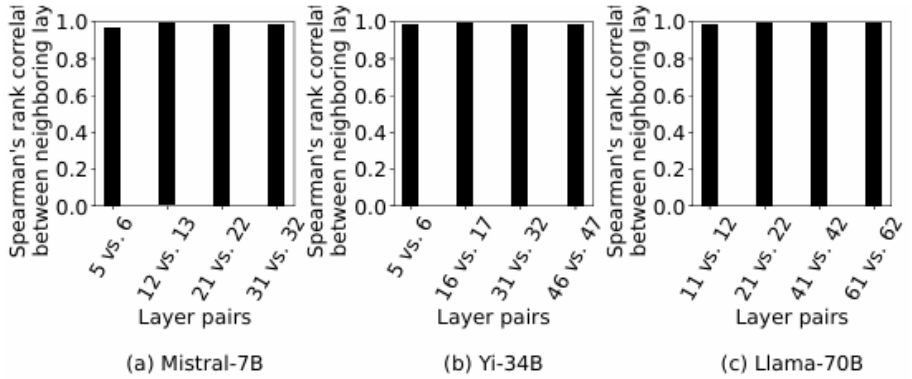
(Extended Learning)

KV Cache for RAG: CacheBlend

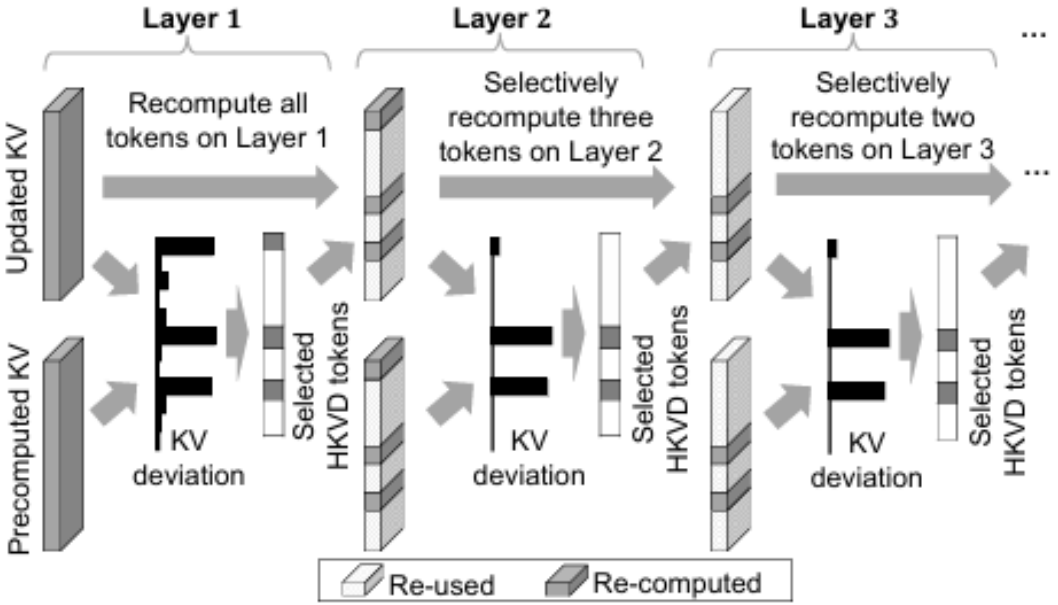
- CacheBlend for RAG
  - Selectively recomputing KV



Distribution of KV deviation of tokens on one layer



Rank correlation of KV deviation per token between two consecutive layers



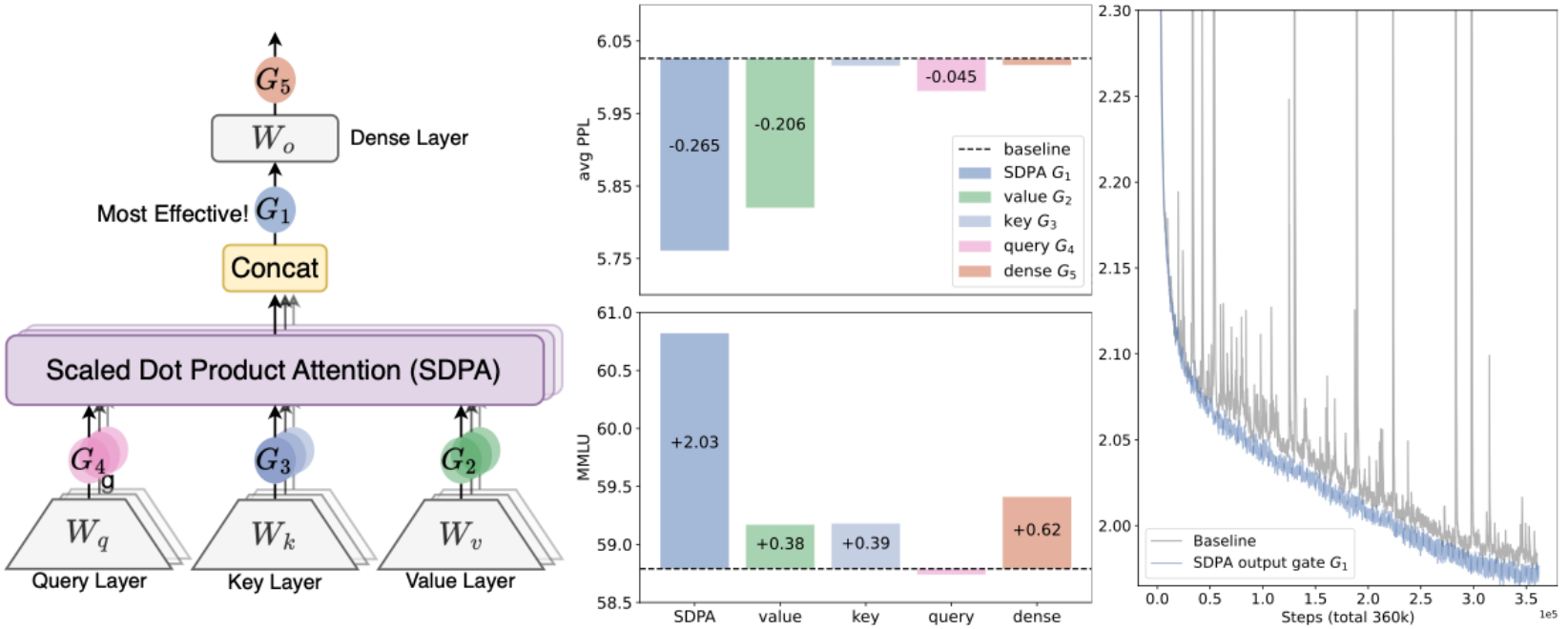
CacheBlend selects high KV deviation tokens of one layer according to inter-layer dependency property

Source: Faster Large Language Model Serving with Knowledge Delivery Networks, By Jiang Junchen@APNet2024

(Extended Learning)

Supplementary Reading on Attention Sinks

- Do we need attention sinks? Voices from a different perspective
  - "Sparse gating mechanism mitigates 'attention sink' and enhances long-context extrapolation performance" – From Qwen



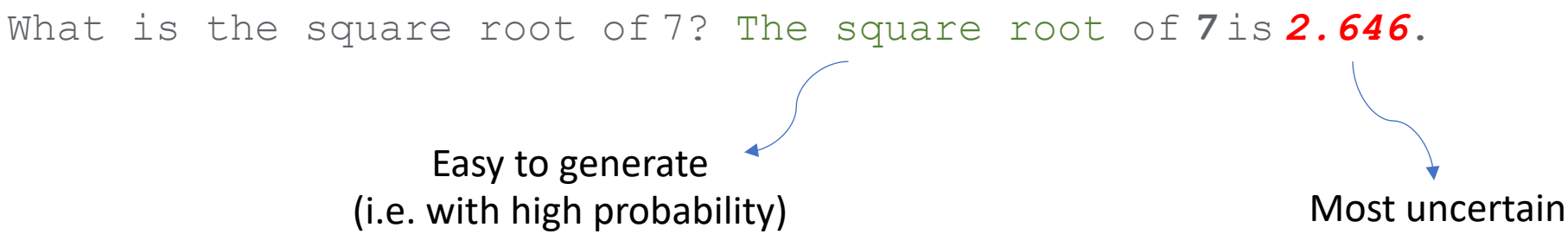
Source: Gated Attention for Large Language Models: Non-linearity, Sparsity, and Attention-Sink-Free

Inference Optimization: Outline

- Overview
- Attention Optimization
- Continuous Batching
- KV Cache Optimization
  - Model Architecture Optimization
  - KV Cache Management
  - KV Offloading and Transmission
- **Speculative Decoding**
- Distributed Serving (Extended Learning)

Speculative Decoding

- Autoregressive models decode one token at a time
  - Decoding is memory-bound—a factual observation rather than an assumption
- Some tokens are easier to generate than others
  - Using a smaller model to do the easy job



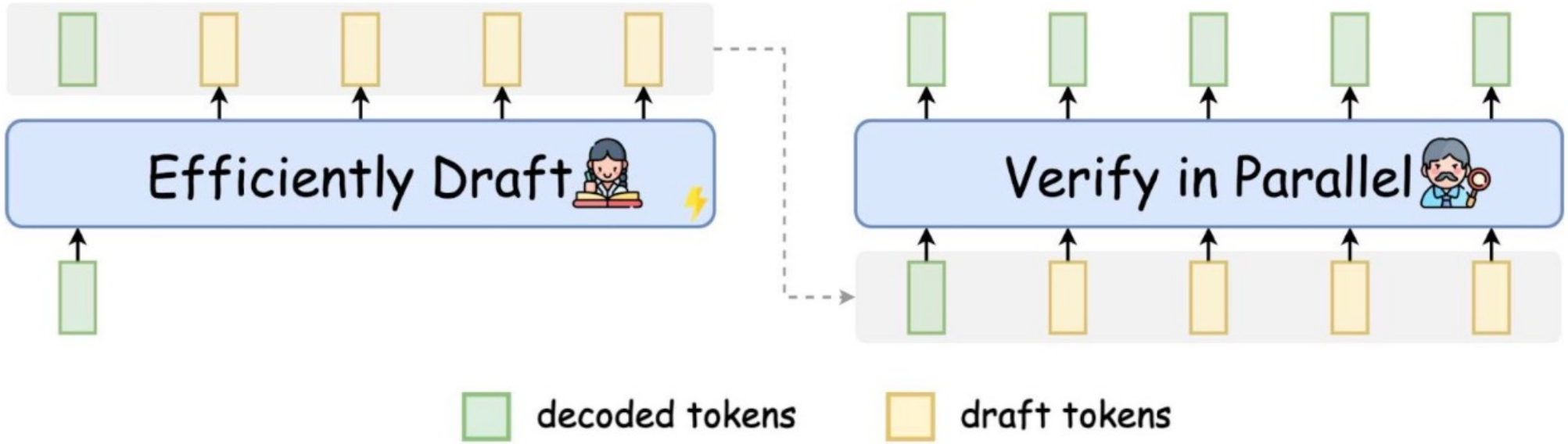
Speculative Decoding

- **Draft model:** a *smaller, lightweight* model that runs alongside to help speed up the main LLM's inference process
- **Target model:** an LLM verifies these proposed tokens *in parallel* and accepts those that match its own predictions

- Key assumption: draft model speculates a sequence of tokens very fast, target model accepts multiple tokens through one-round verification (inference).

Speculative Decoding

- Draft-then-Verify Paradigm



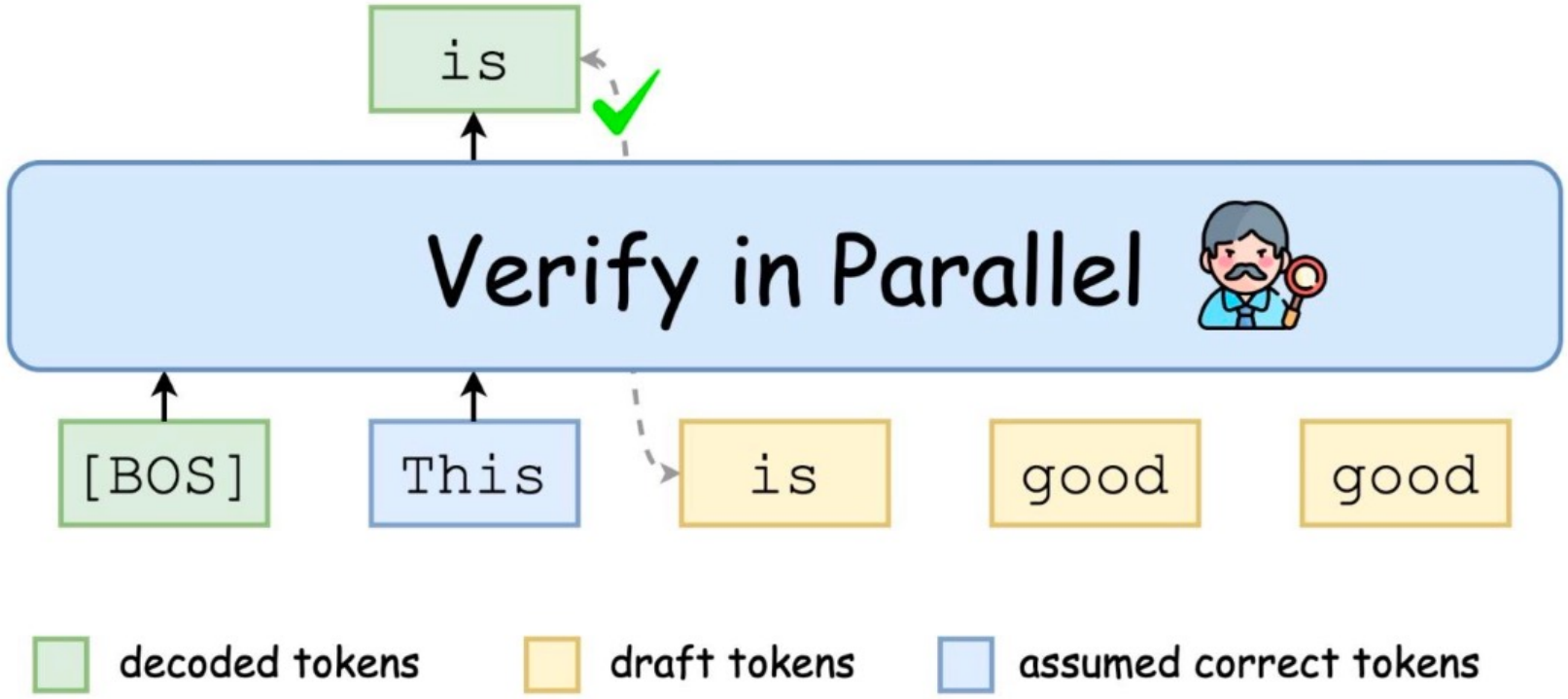
Speculative Decoding

- Verify In Parallel

The original LLM output: [BOS] This is good news. [EOS]

Speculative Decoding

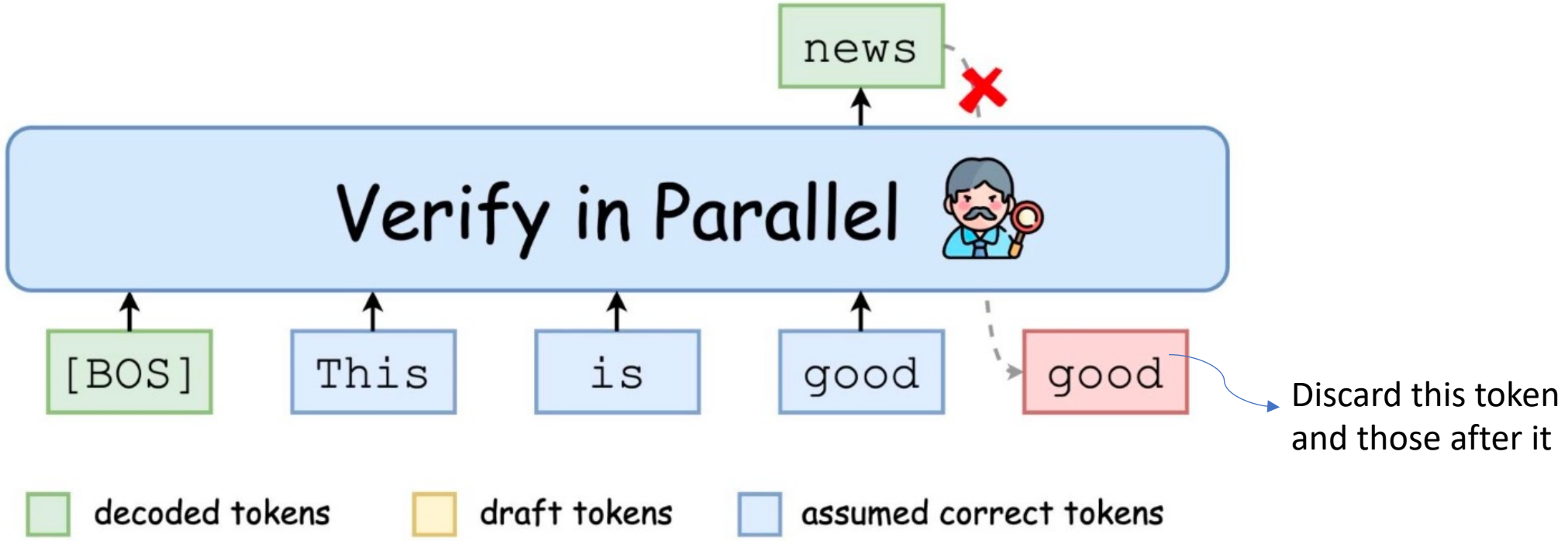
- Verify In Parallel



Draft output coincides with original LLM by far

Speculative Decoding

- Verify In Parallel



Bifurcation: The index of the first draft token that fails verification.

Speculative Decoding

- Principle of parallel verification
  - **Draft model** generates a sequence of speculated tokens

    $[\tilde{x}_1, \tilde{x}_2, ...\tilde{x}_K]$ with sampling probabilities $q_i = P_{Draft}(\tilde{x}_i \mid C, \tilde{x}_{1:i-1})$
  - **Target model** performs forwarding
    - Let the confirmed context be C, concatenate it to be $C + \tilde{x}_1 + \tilde{x}_2 + ... + \tilde{x}_K$
    - Compute conditional probability distributions using mask matrix K simultaneously

$$
\begin{aligned}
p_1(\cdot) &= P_{Target}(\cdot \mid C) \\
p_2(\cdot) &= P_{Target}(\cdot \mid C, \tilde{x}_1) \\
p_3(\cdot) &= P_{Target}(\cdot \mid C, \tilde{x}_1, \tilde{x}_2) \\
&\cdots \\
p_K(\cdot) &= P_{Target}(\cdot \mid C, \tilde{x}_1, ..., \tilde{x}_{K-1})
\end{aligned}
$$

Speculative Decoding

- Principle of parallel verification
  - **Standard Validation & Acceptance Rule**

$$Accept_{prob} = min\left(1, \frac{p_i}{q_i}\right)$$

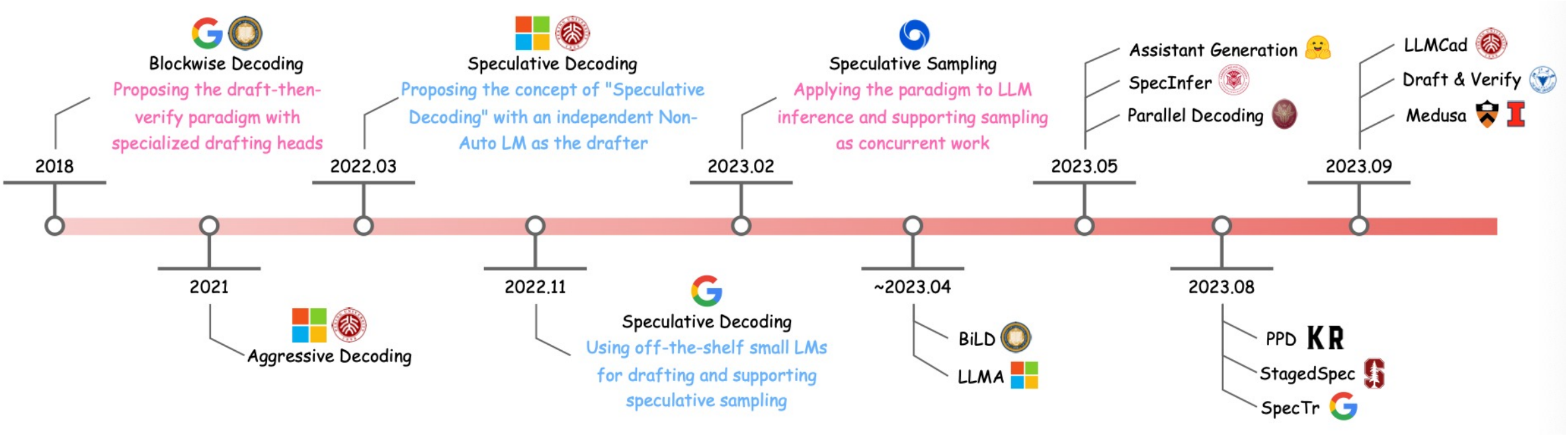
    - If $p_i \geq q_i$ the draft token is accepted unconditionally
    - Otherwise, accept the token with the probability of p_i/q_i ; otherwise reject it.
    - Strict sequential constraint: Probability calculation is parallel, while acceptance judgment remains serial. All subsequent draft tokens are invalidated if one token gets rejected.
  - Simplified Validation Strategies
    - Greedy Decoding (Top-1 Exact Matching), etc.

Speculative Decoding

- Speculative decoding metrics
  - **Speculation accuracy** (high) and **drafting time** (low)
- Drafting
  - Self drafting
    - **Extra head**, layer skipping, mask predict
  - Independent drafting
    - **Non-Autoregressive LM**, **Small LM**, Context Retrieval, Copy
- Verification:
  - Strict matching, approximate matching

Speculative Decoding

- A brief history

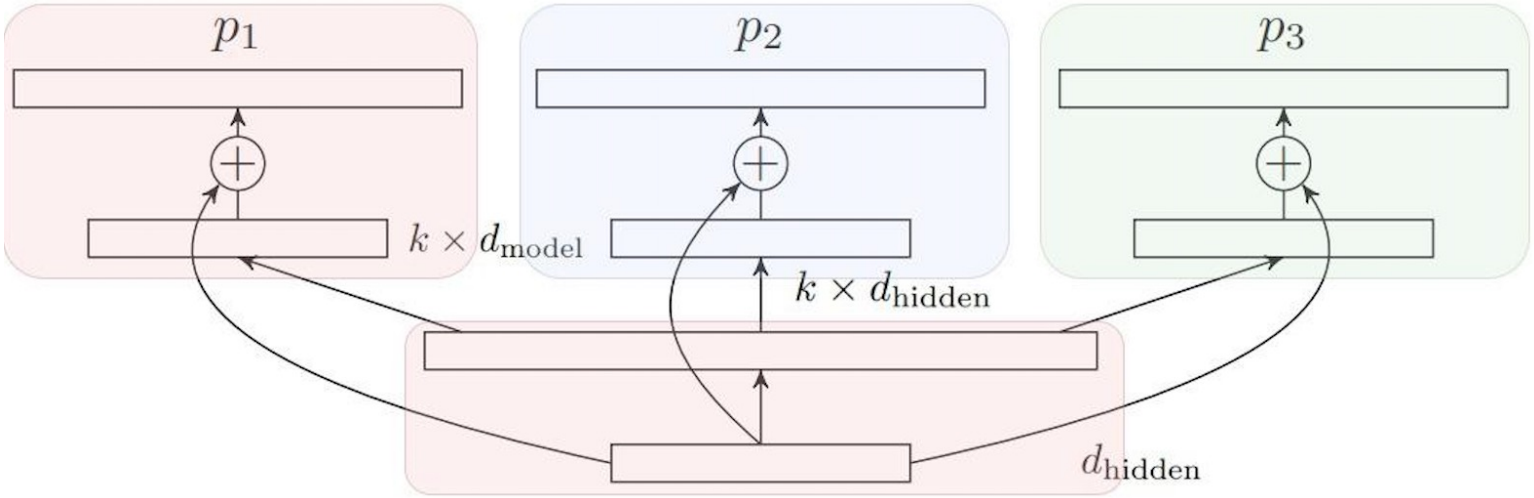


Source: Unlocking Efficiency in Large Language Model Inference: A Comprehensive Survey of Speculative Decoding

(Extended Learning)

Self-Drafting

- Block-wise Decoding (2018)
  - Augmenting the dimensionality of the final projection layer by a factor of k



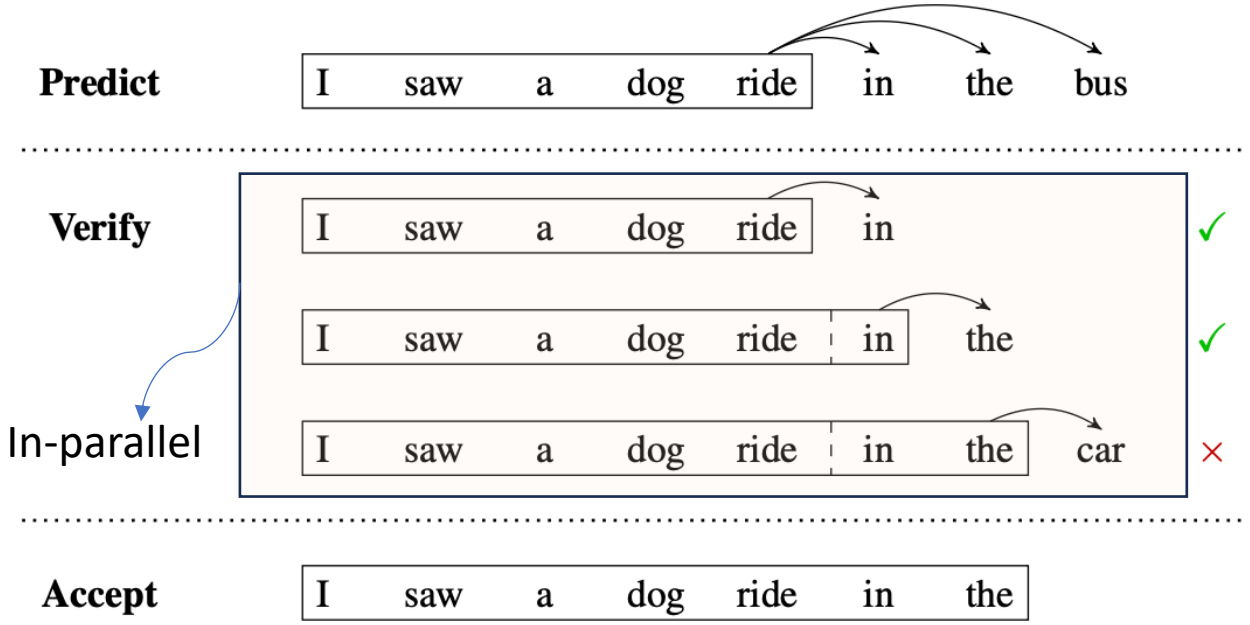
- Apply the original vocabulary projection, add k output layers, add a hidden layer, compute k separate softmaxes per position
- Train the extra parameters or fine-tune them

Source: Blockwise Parallel Decoding for Deep Autoregressive Models

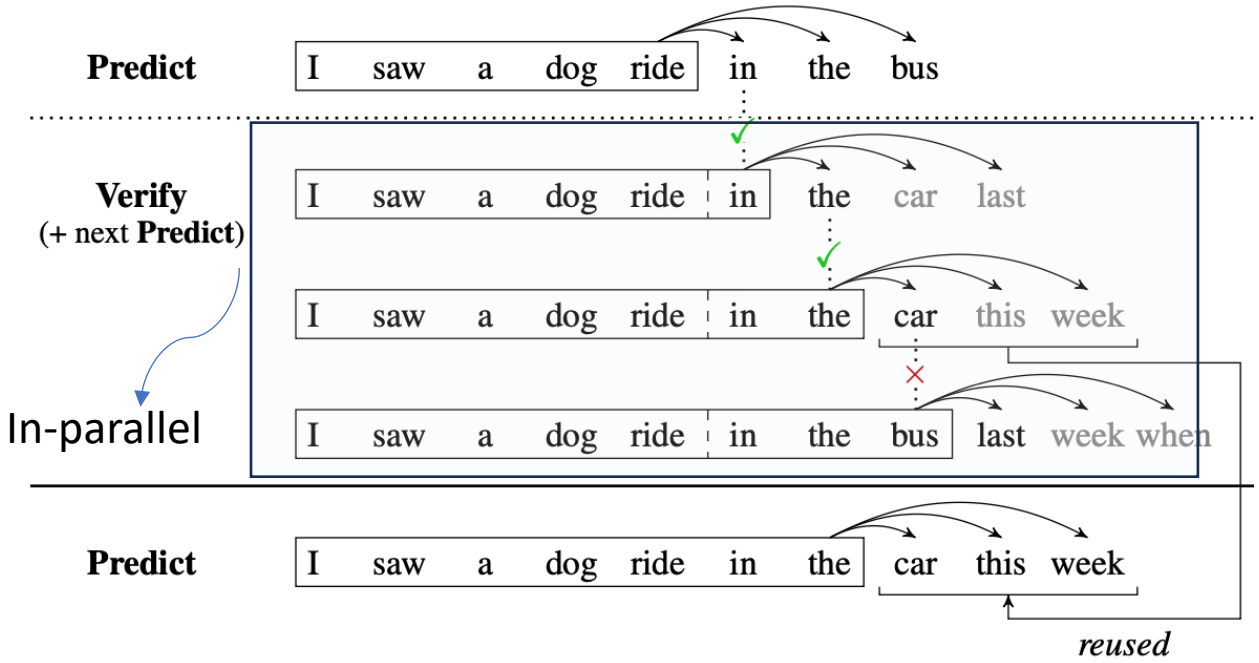
(Extended Learning)

Self-Drafting

- Block-wise Decoding (2018)
  - Predict-Verify-Accept versus Predict-Verify



Two forward passes



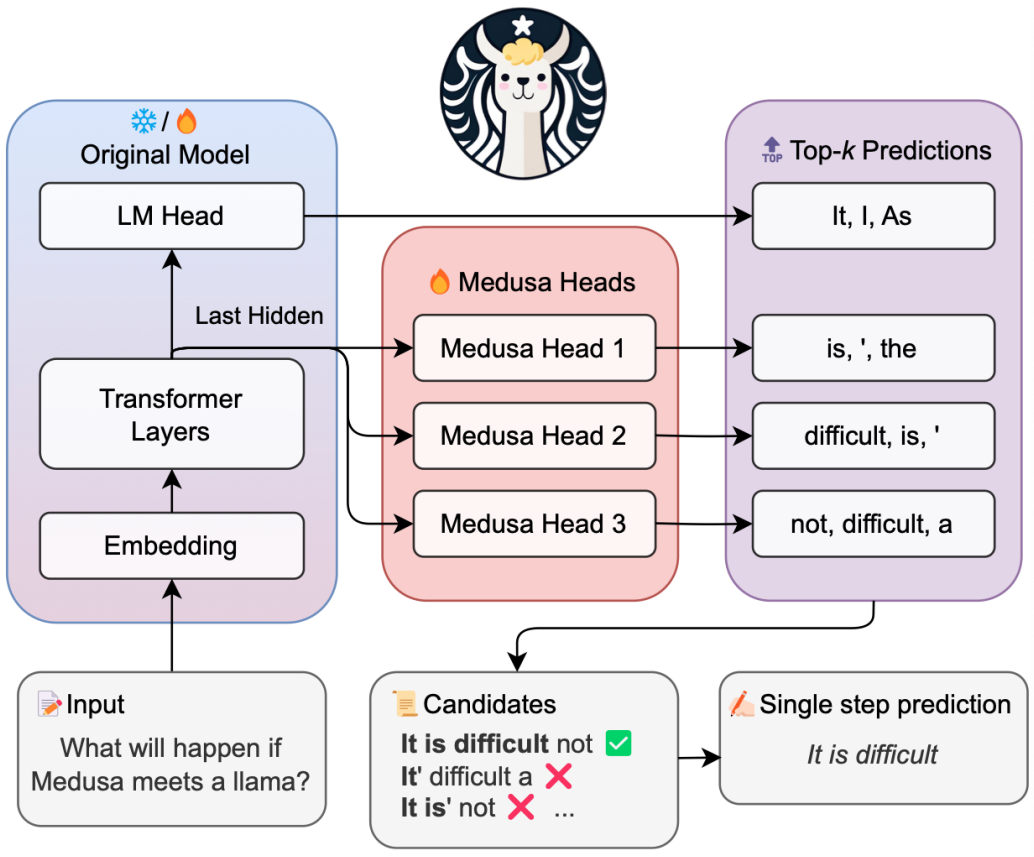
One forward pass

Source: Blockwise Parallel Decoding for Deep Autoregressive Models

(Extended Learning)

Self-Drafting

- Medusa (2023)



Medusa appends k project layers (heads) to generate $k + 1$ tokens

Keeping backbone model fixed, and training or finetuning Medusa heads

Similar to blockwise parallel decoding (almost) if top-1 greedy decoding is adopted

Source: Medusa: Simple LLM inference acceleration framework with multiple decoding heads.

(Extended Learning)

Self-Drafting

- Medusa (2023)
  - Model training
    - The original model stays untouched, and only the new heads are trained/fine-tuned

probability of token y predicted by the k-th head

Medusa-1: Frozen Backbone $\mathcal{L}_{\text{MEDUSA-1}} = \sum_{k=1}^{K} -\lambda_k \log p_t^{(k)}(y_{t+k+1})$

Medusa-2: Joint Training $\mathcal{L}_{\text{MEDUSA-2}} = \mathcal{L}_{\text{LM}} + \lambda_0 \mathcal{L}_{\text{MEDUSA-1}}$

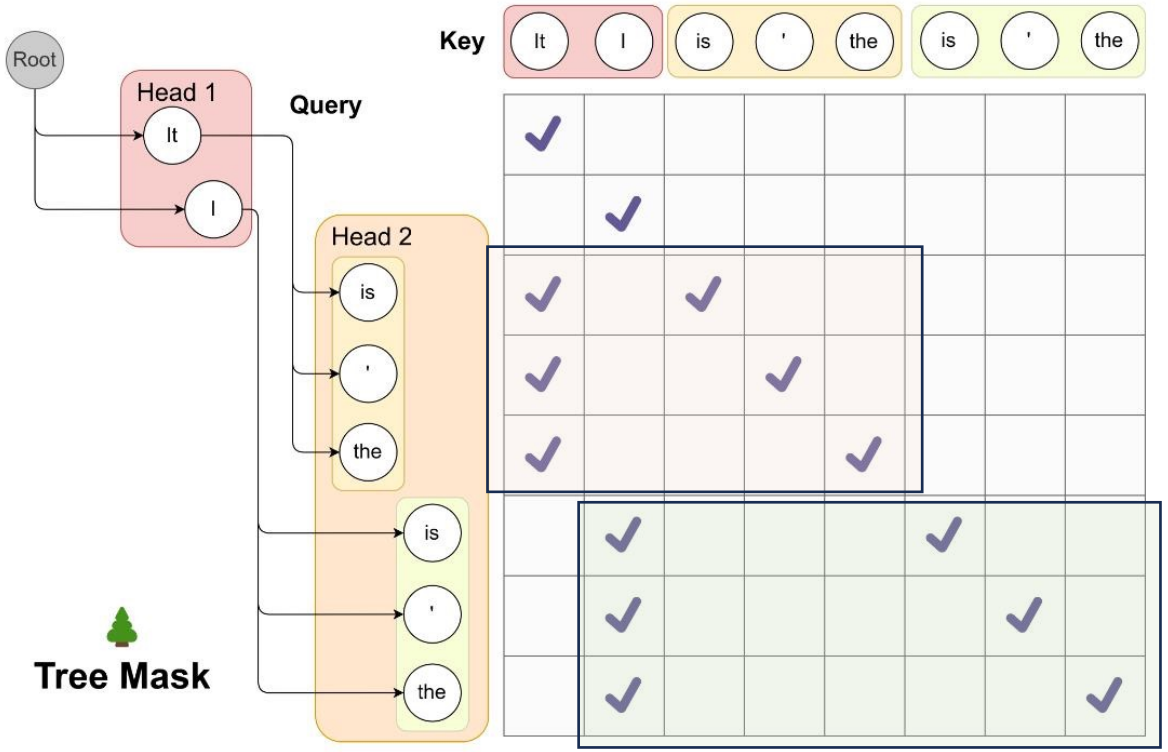
  - Prediction
    - Medusa heads predict multiple likely words for the corresponding position
    - The predicted tokens are combined and processed using a tree-based attention mechanism
  - Verification
    - A typical acceptance scheme is employed to pick the longest plausible prefix from the candidates for further decoding

Source: Medusa: Simple LLM inference acceleration framework with multiple decoding heads.

(Extended Learning)

Self-Drafting

- Medusa (2023)
  - Tree-based attention



- Using tree attention to process multiple candidates concurrently.
  - The top-2 predictions from the first MEDUSA head and the top-3 from the second result in a total of 6 candidates. Each of these candidates corresponds to a distinct branch within the tree structure.
- Devising an attention mask.
  - We exclusively permits attention flow from the current token back to its antecedent tokens.

Source: Medusa: Simple LLM inference acceleration framework with multiple decoding heads.

(Extended Learning)

Self-Drafting

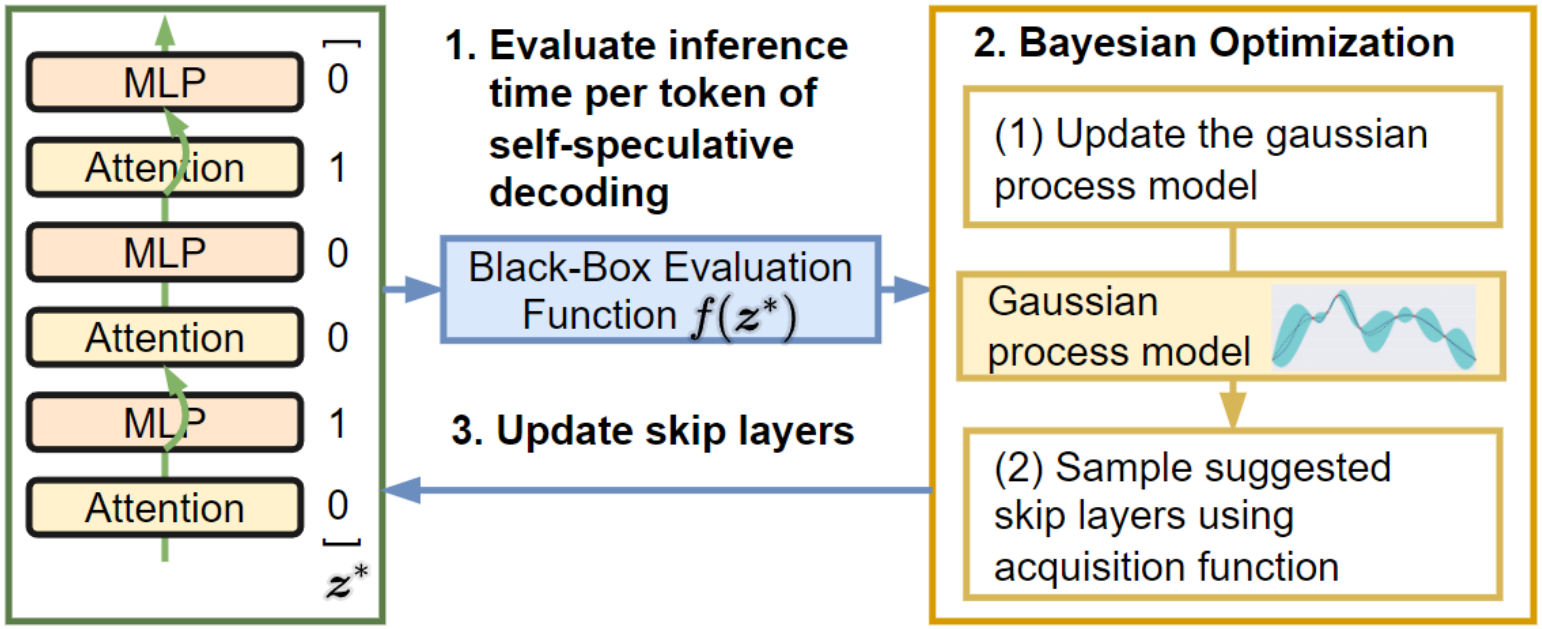
- Draft & Verify (2023) (for reference only)
  - Drafting Stage: selectively skipping certain intermediate layers
  - Verification Stage: evaluating all drafted tokens in a full-model forward pass
  - Unable to Draft and Verify in one pass

Source: Draft & Verify: Lossless Large Language Model Acceleration via Self-Speculative Decoding

(Extended Learning)

Self-Drafting

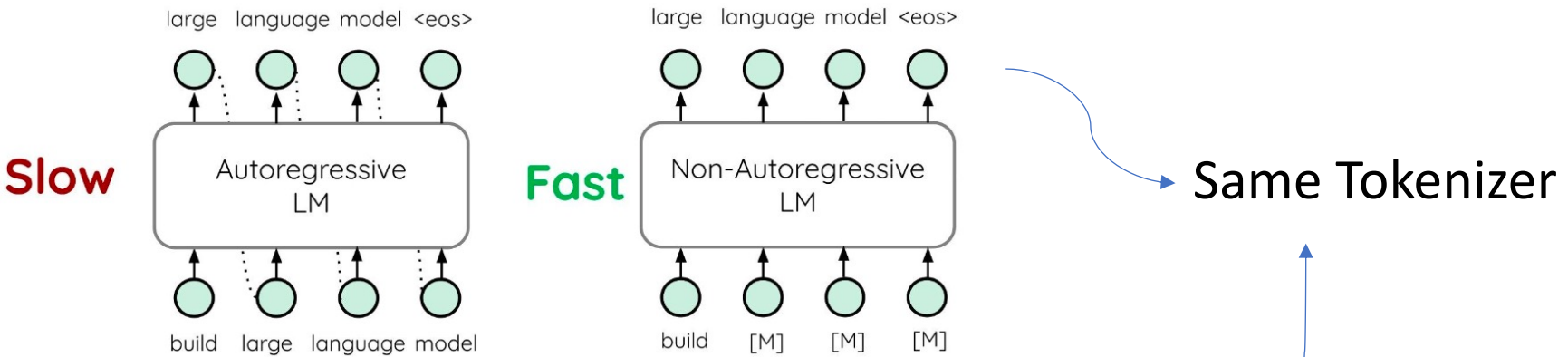
- Draft & Verify (2023) (for reference only)
  - Objective: the average inference time per verified token on a development set
  - Method: Using Bayesian optimization to find which layers to skip



Source: Draft & Verify: Lossless Large Language Model Acceleration via Self-Speculative Decoding

Independent Drafting

- Non-autoregressive LMs: usually less performant



- Smaller Models for Drafting: tradeoff between model size and acceptance rate

increasing

Draft Model	Target Model	Acceptance Rate	Speed Up
T5-Small (77M)	T5-XXL (11B)	0.75	3.4x
T5-Base (250M)	T5-XXL (11B)	0.8	2.8x
T5-Large (800M)	T5-XXL (11B)	0.82	1.7x

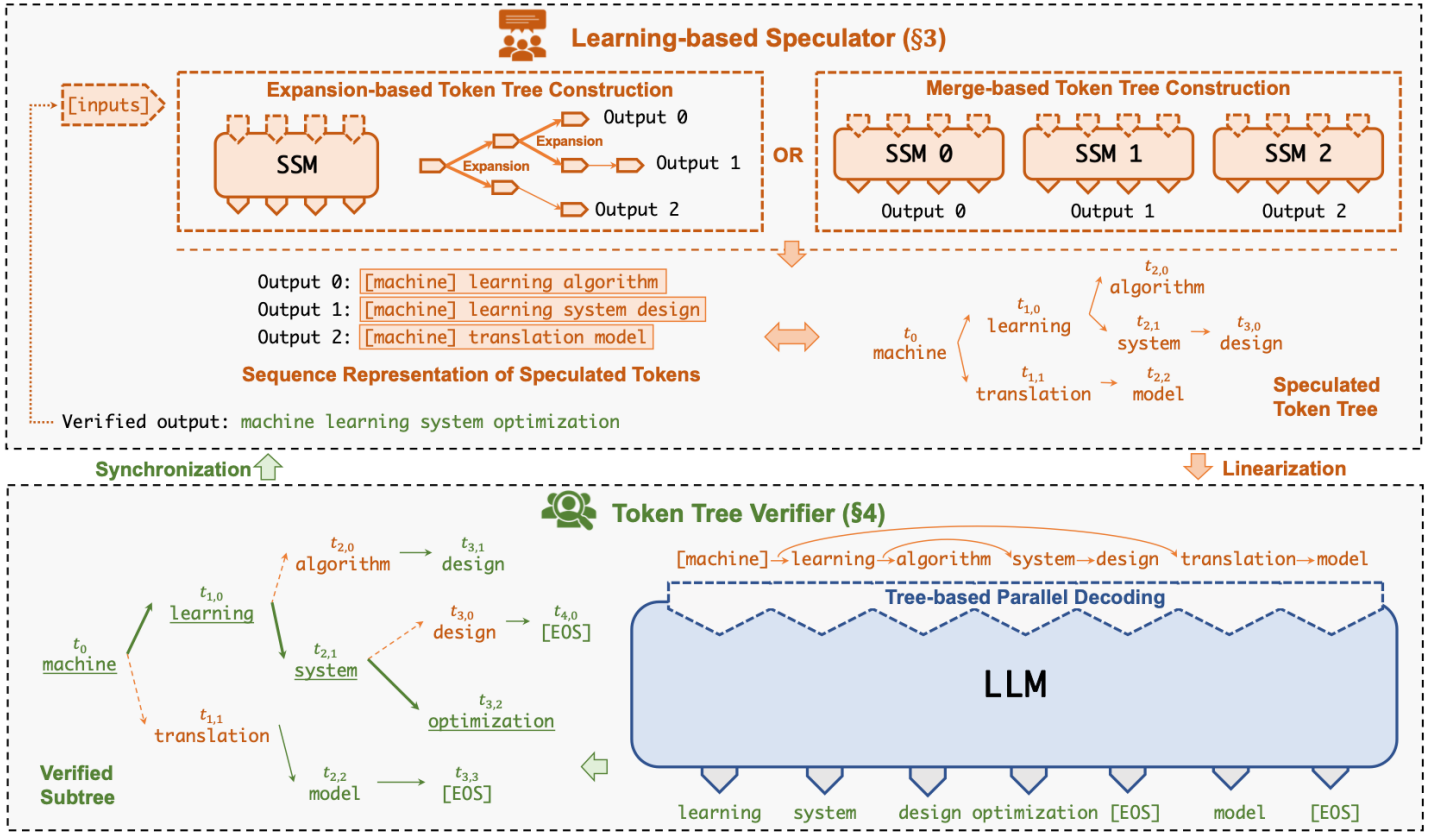
increasing

Source: Fast Inference from Transformers via Speculative Decoding, and the tutorial by Dr. Heming Xia
Source: Speculative Decoding: Exploiting Speculative Execution for Accelerating Seq2seq Generation,

(Extended Learning)

Independent Drafting

- SpecInfer (2024) (for reference only)



Learning-based speculator: predict the LLM's output by maximizing the overlap between the speculated token tree and the tokens generated by the LLM using incremental decoding

Source: SpecInfer: Accelerating Large Language Model Serving with Tree-based Speculative Inference and Verification

Speculative Decoding: Brief Summary

- Key Idea
  - Employing small models to predict multiple tokens, and verify them via LLM
  - Requiring accurate prediction and low speculative latency
- Feasibility
  - Some tasks are easy, while some are difficult
  - Decoding is usually not compute-bound, allowing fast computing of small models
- Methods
  - Drafting: self-drafting and independent drafting
  - Verification: greedy decoding (exact match) and speculative sampling (e.g. output-distribution match) and token-free verification

Source: Unlocking Efficiency in Large Language Model Inference: A Comprehensive Survey of Speculative Decoding

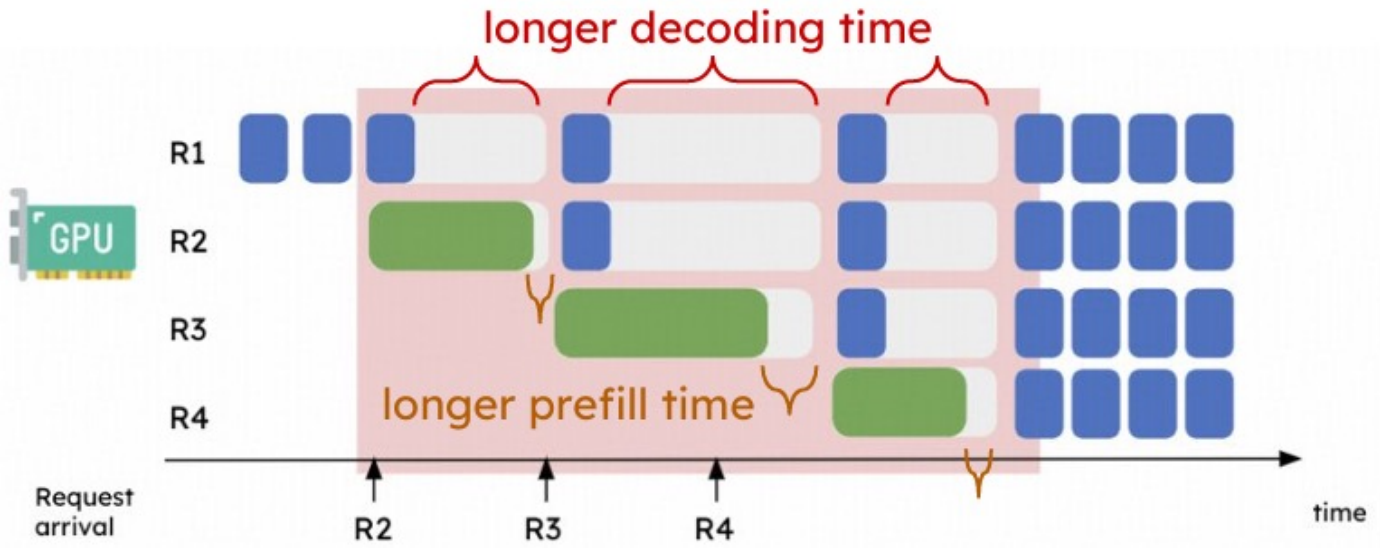
Inference Optimization: Outline

- Overview
- Attention Optimization
- Continuous Batching
- KV Cache Optimization
  - Model Architecture Optimization
  - KV Cache Management
  - KV Offloading and Transmission
- Speculative Decoding
- **Distributed Serving** (Extended Learning)

(Extended Learning)

Prefill-Decode Disaggregation

- Prefill and decode colocation negatively impact each other.
  - Batching prefill and decoding phase together hurts both TTFT and TPOT
  - Three representative approaches: ***Orca***, ***Chunked Prefill*** and ***Prefill-Decode disaggregation***



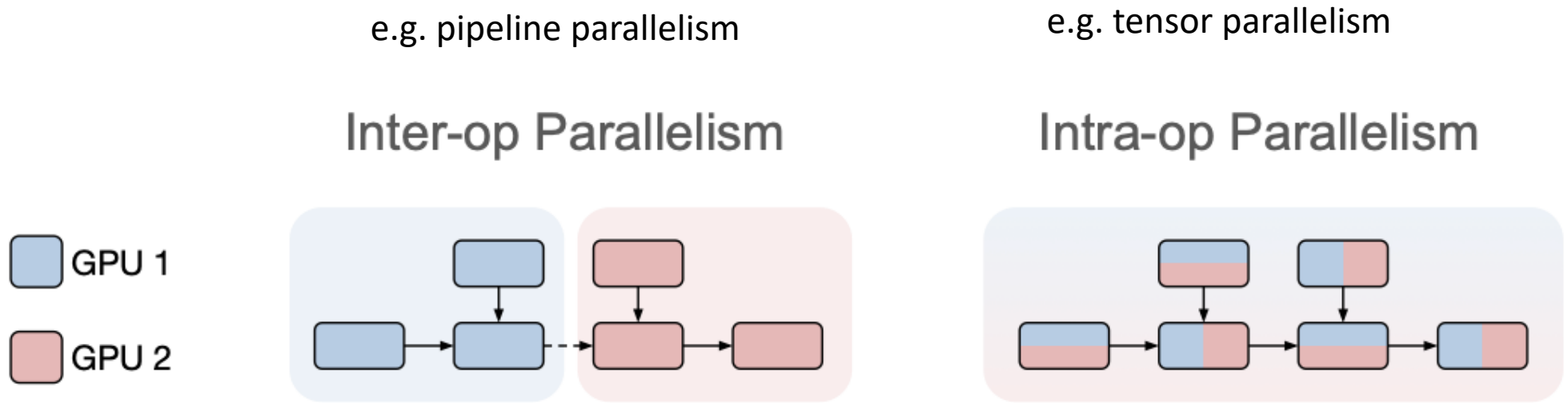
Prefill and decode times are prolonged because of mutual interference

Source: DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving

(Extended Learning)

Prefill-Decode Disaggregation

- Prefill and decode colocation enforces the same parallel strategy



Prefill-Decode decoupling is the KEY!

Source: DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving

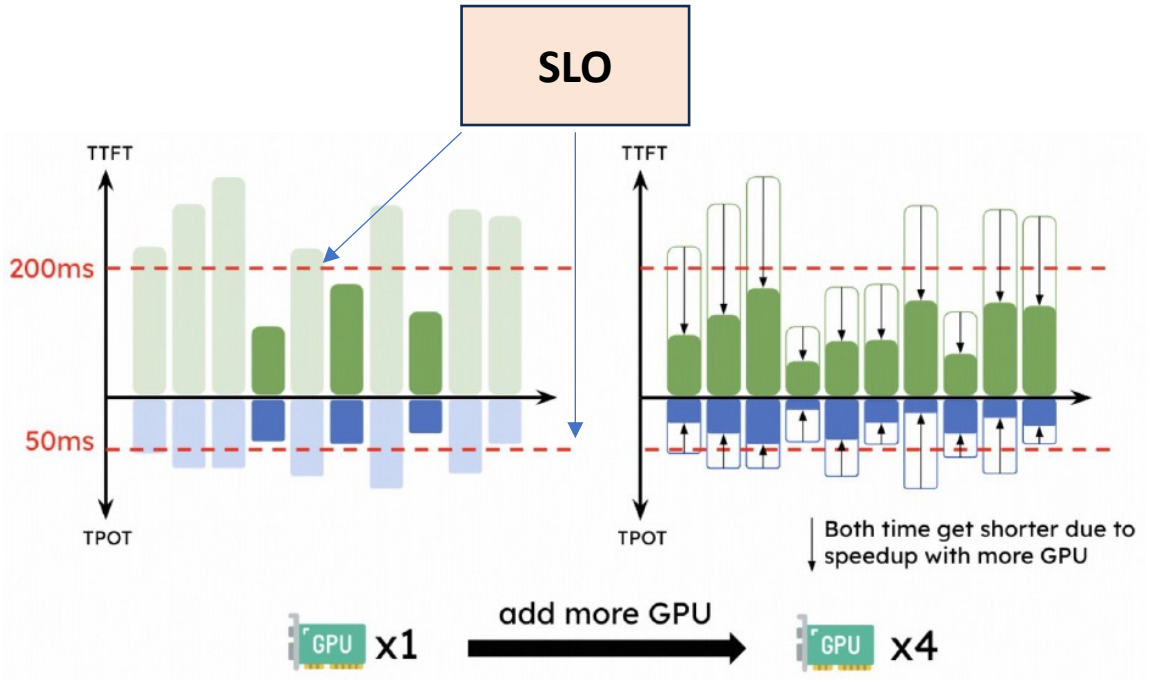
(Extended Learning)

Prefill-Decode Disaggregation

- Application-dependent SLO
  - **High throughput != high goodput:** Different apps have various latency requirements

	TTFT	TPOT
Chat	Low (< 1s)	Match read speed (~ 100ms)
Search	Very Low (~ 200ms)	Match read speed (~ 100ms)
Program	Very Low (~ 200ms)	Very Low (~ 50ms)

LLM as a service: diverse SLO



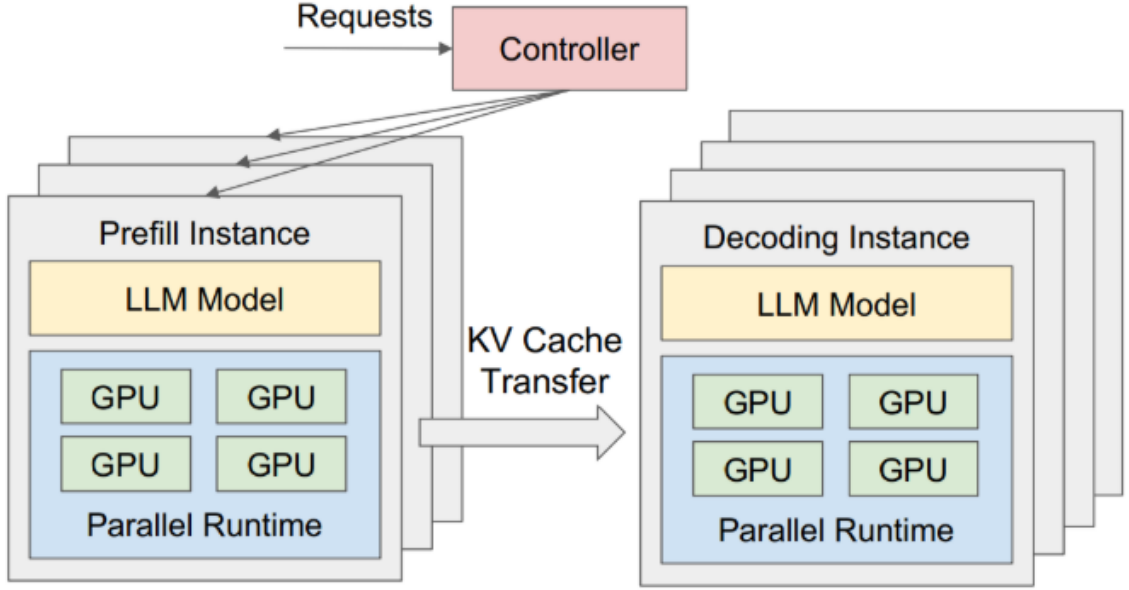
Overprovisioning for LLM serving:
using 4 GPUs increases overall cost

Source: https://hao-ai-lab.github.io/blogs/distserve/

(Extended Learning)

Prefill-Decode Disaggregation

- PD Disaggregation
  - Compute-bound prefill and memory-bound decode on different GPU pools
  - Prefill GPUs ***transfer*** KV cache to Decode GPUs
  - Optimizing prefill and decode separately via the most suitable parallelism



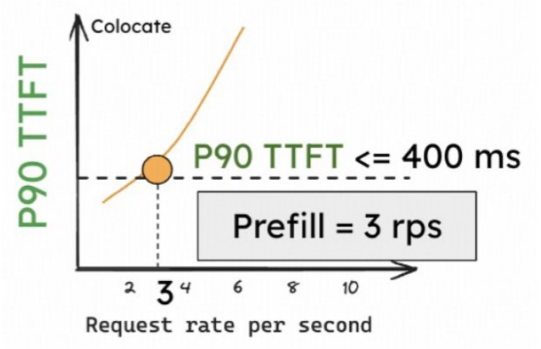
Source: DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving

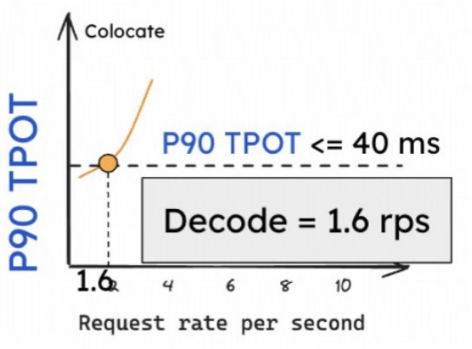
(Extended Learning)

Prefill-Decode Disaggregation

- Application-dependent SLO

Per-GPU throughput **does not change**

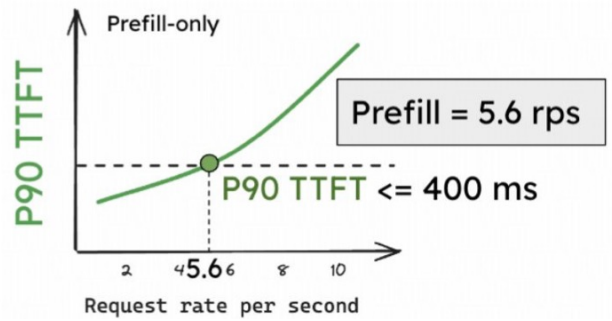


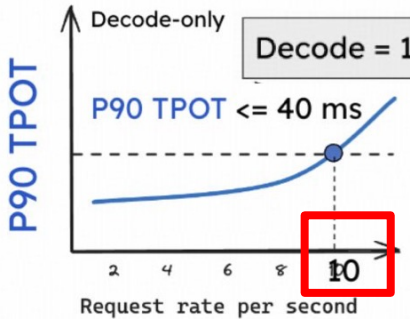


Colocation

GPU

Max System rps
= Min(Prefill, Decode)
= 1.6 rps / GPU





Disaggregation (2P1D)

GPU GPU GPU

Max System rps
= Min (5.6 x **2**, 10) rps / 3 GPU
= 3.3 rps / GPU

Improved throughput

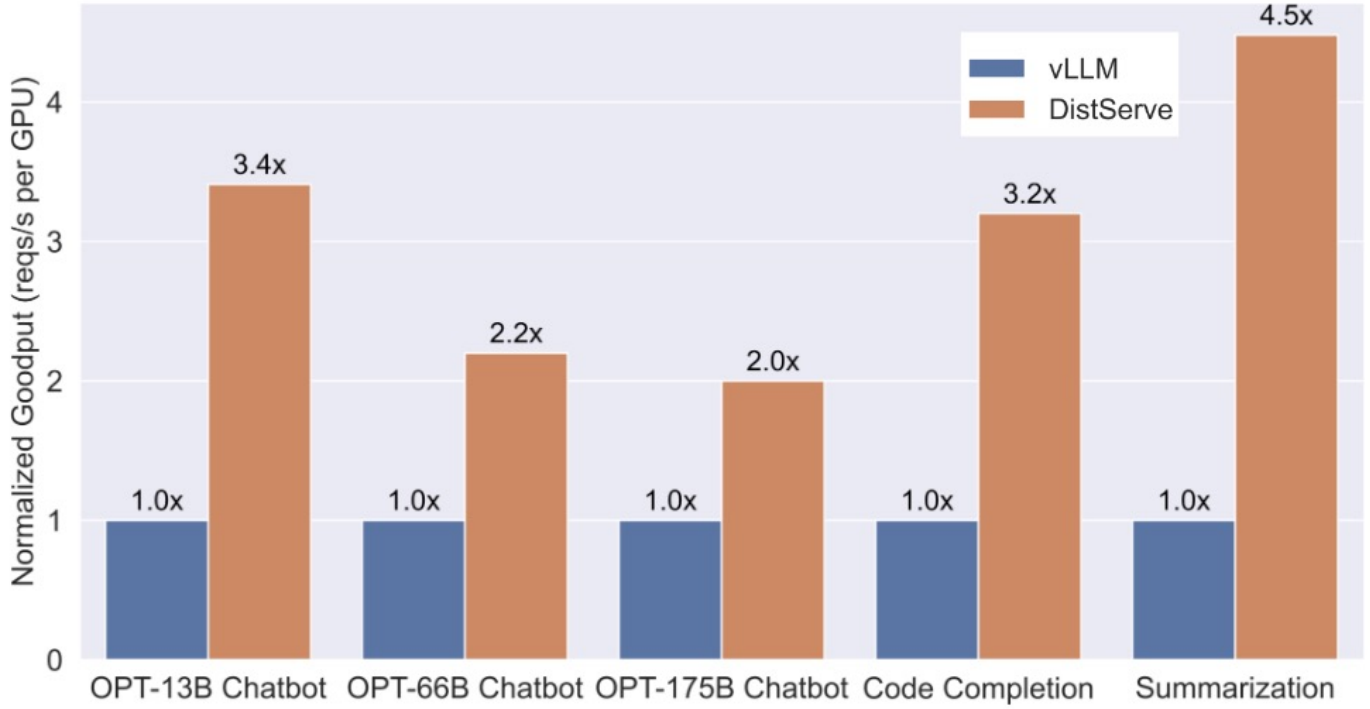
Doing only one type of job: decoding

Source: DistServe: Disaggregating Prefill and Decoding for Goodput-optimiz

(Extended Learning)

Prefill-Decode Disaggregation

- Experimental Analysis

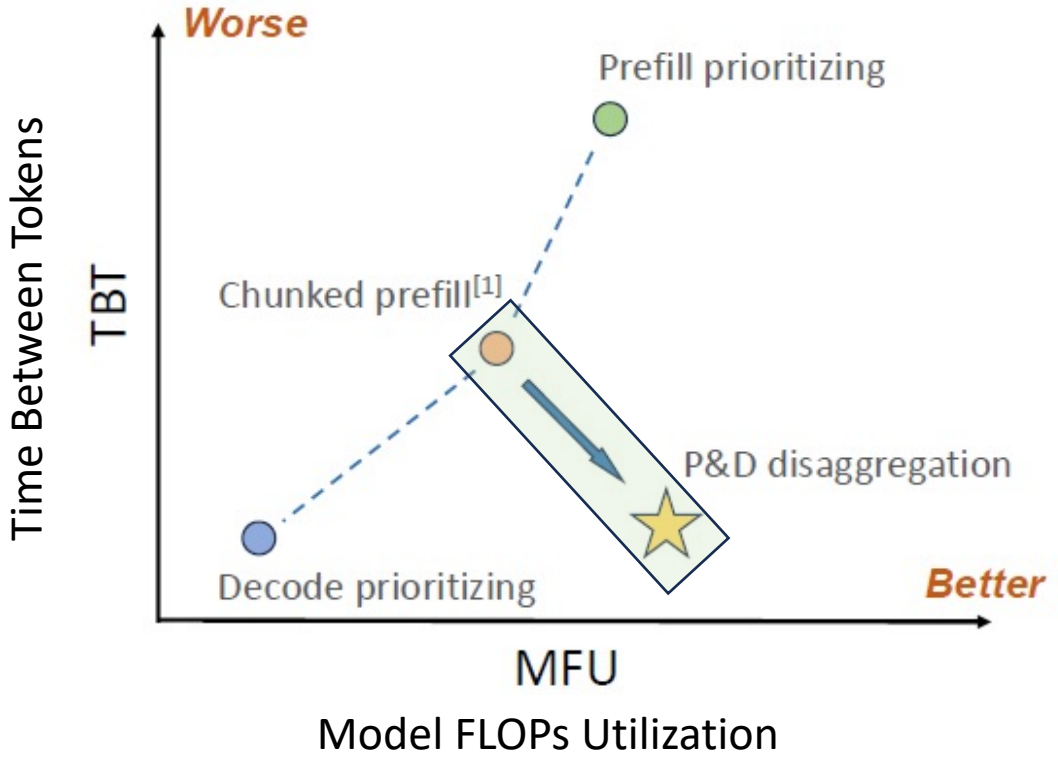


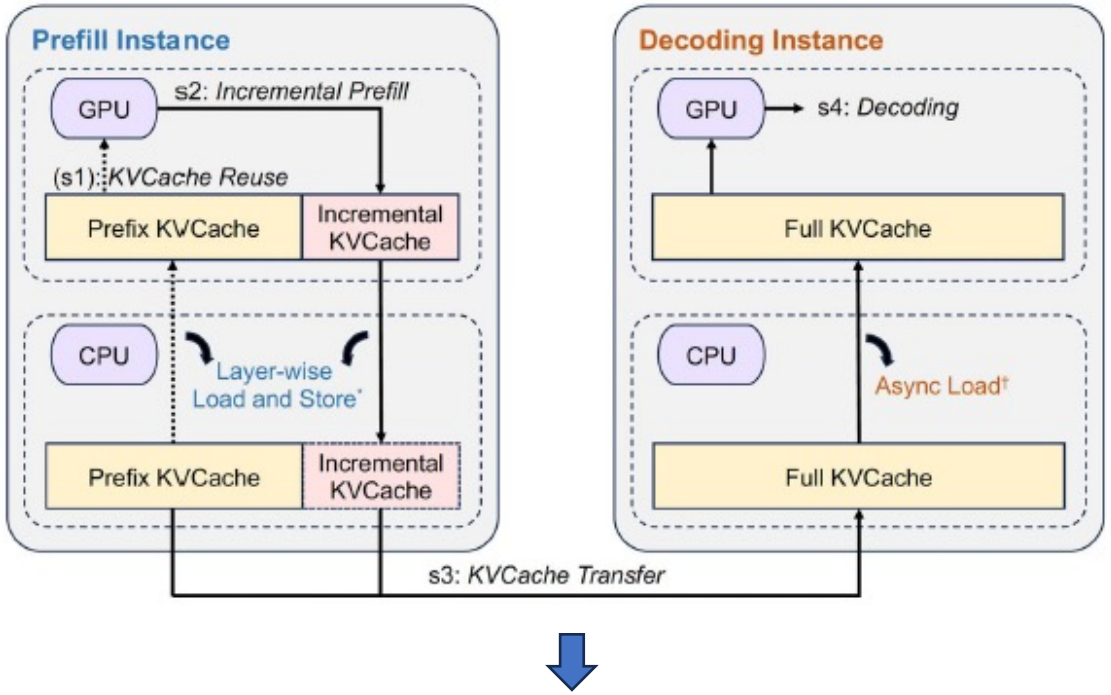
Evaluation of DistServe against vLLM on various datasets

(Extended Learning)

Prefill-Decode Disaggregation

- Chunked Prefill versus PD Disaggregation
  - Best: high MFU low TBT





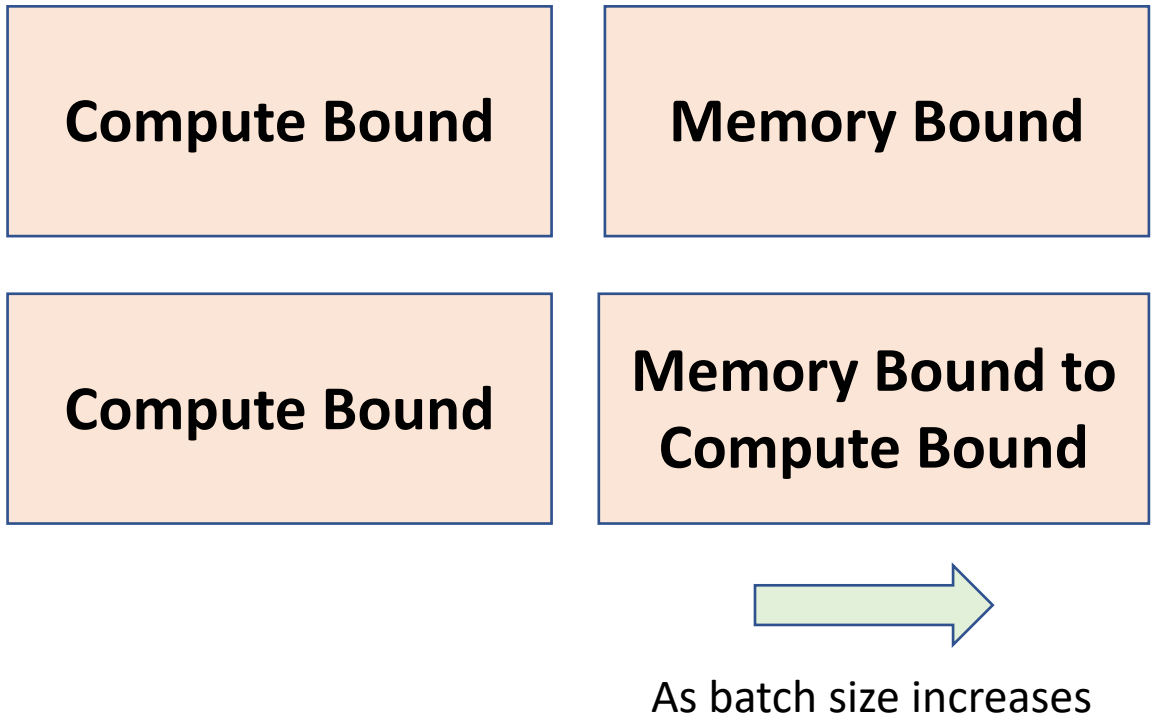
KV Cache transfer on the same layer

Source: Trading More Storage for Less Computation A KV Cache-centric Architecture for Serving LLM Chatbot

(Extended Learning)

Attention-FFN Disaggregation

- MoE Inference

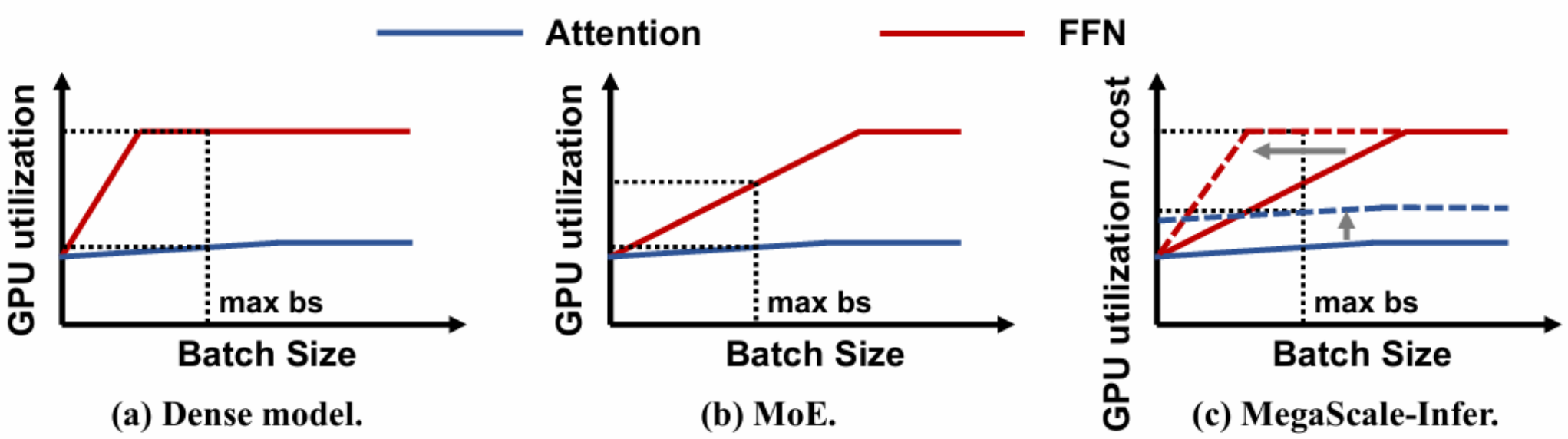


Source: MegaScale-Infer: Serving Mixture-of-Experts at Scale with Disaggregated Expert Parallelism

(Extended Learning)

Attention-FFN Disaggregation

- Motivation of AF Disaggregation
  - Attention is memory intensive and FFN is compute intensive in dense models
  - Sparse MoE models reduce the batch size on an expert



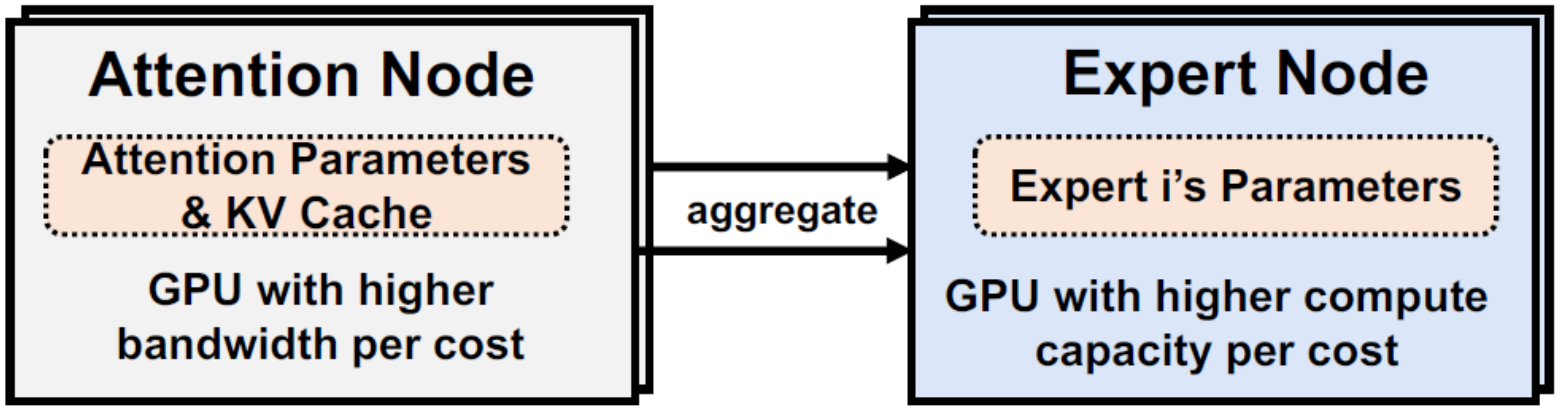
GPU utilization of Attention and FFN vs. batch size in dense model, MoE

Source: MegaScale-Infer: Serving Mixture-of-Experts at Scale with Disaggregated Expert Parallelism

(Extended Learning)

Attention-FFN Disaggregation

- Key Idea of MegaScale-Infer
  - Independent scaling: feed more requests to FFN
  - Cost-effective, heterogeneous hardware for Attention and FFN

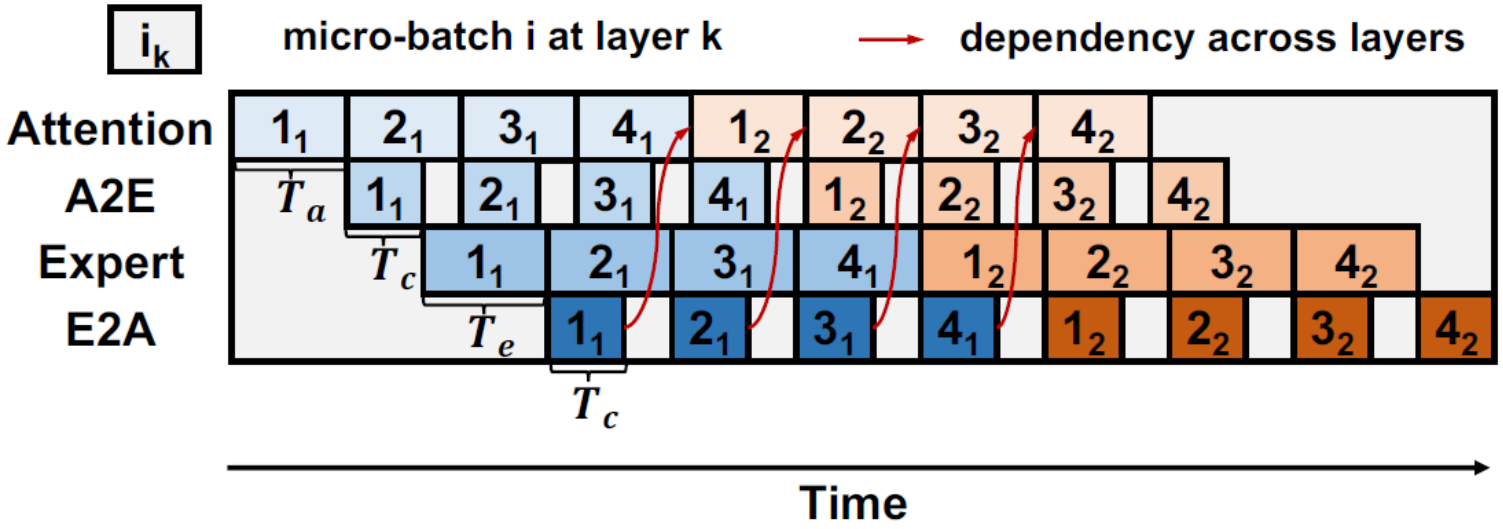


Source: MegaScale-Infer: Serving Mixture-of-Experts at Scale with Disaggregated Expert Parallelism

(Extended Learning)

Attention-FFN Disaggregation

- Challenges
  - Pipeline bubbles: sequential computation of a batch will result in pipeline bubbles, i.e. not all the GPUs being utilized at the same time
  - Ping-Pong pipeline parallelism: dividing a large batch into multiple microbatches

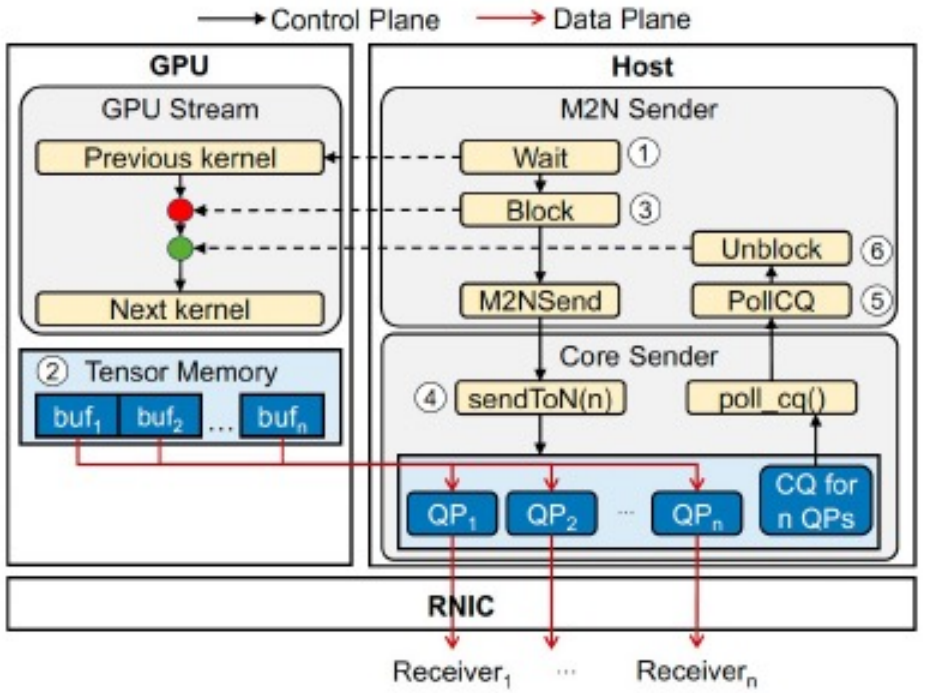


Source: MegaScale-Infer: Serving Mixture-of-Experts at Scale with Disaggregated Expert Parallelism

(Extended Learning)

Attention-FFN Disaggregation

- Challenges
  - Simple idea, yet definitely non-trivial implementation
    - Minimize overhead such as data copies
    - CPU manages send/receive operations
    - GPU Direct over RDMA data transfer

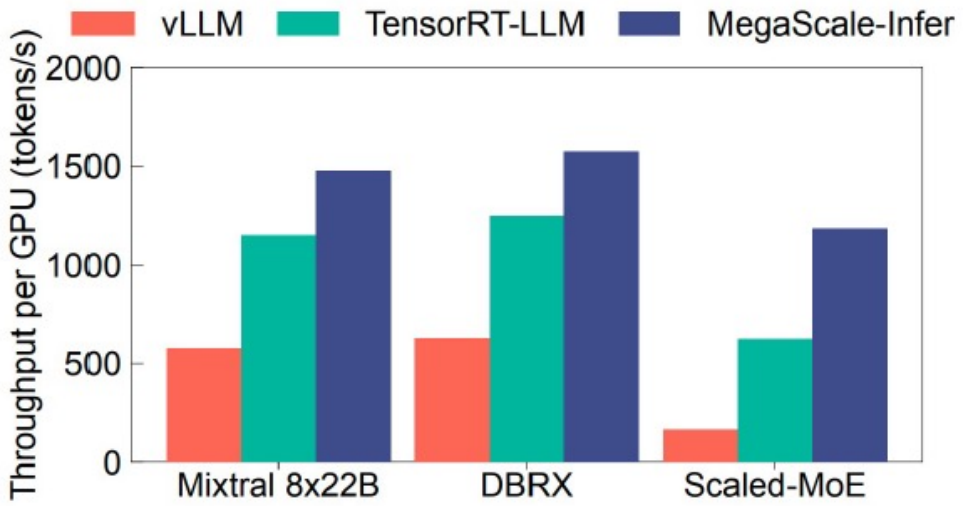


Source: MegaScale-Infer: Serving Mixture-of-Experts at Scale with Disaggregated Expert Parallelism

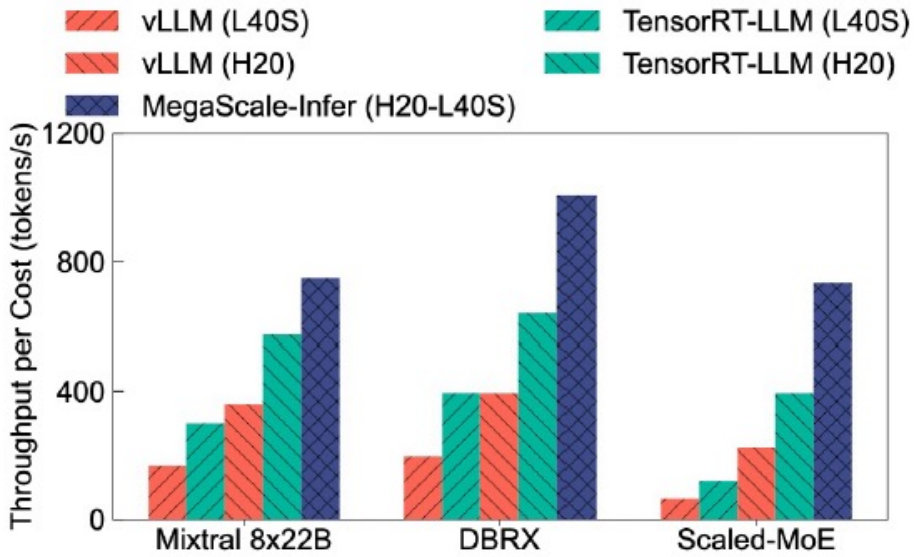
(Extended Learning)

Attention-FFN Disaggregation

- Experimental Analysis



Homogeneous clusters



Heterogeneous clusters

Source: MegaScale-Infer: Serving Mixture-of-Experts at Scale with Disaggregated Expert Parallelism

Thanks!