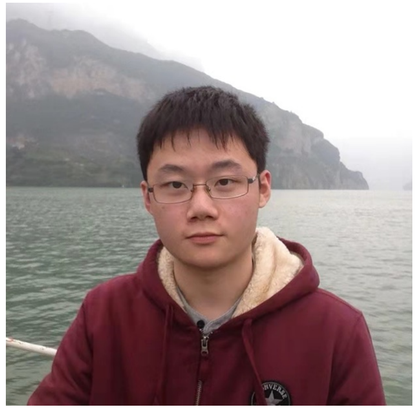
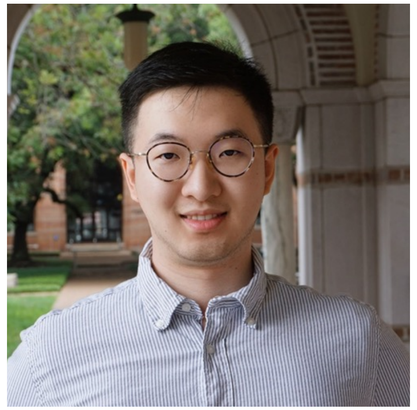
The 39th Annual AAAI Conference on Artificial Intelligence
FEBRUARY 25 – MARCH 4, 2025 | PHILADELPHIA, PENNSYLVANIA
Feb 25, 4:15pm-6:00pm, Room 116

# TQ08: KV Cache Compression for Efficient Long Context LLM Inference: Challenges, Trade-Offs, and Opportunities



Zhaozhuo Xu


Zirui Liu


Shaochen (Henry) Zhong


Jiayi Yuan


Beidi Chen


Anshumali Shrivastava


Xia Hu

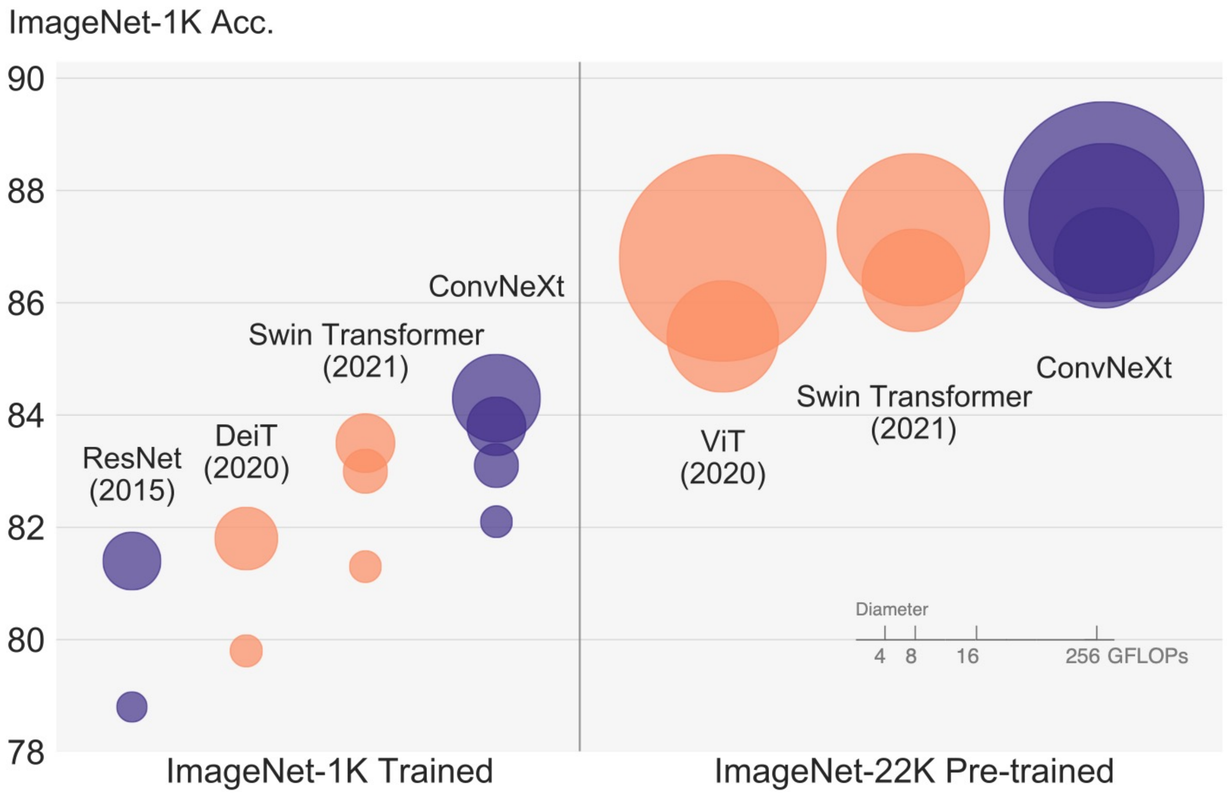
# Agenda

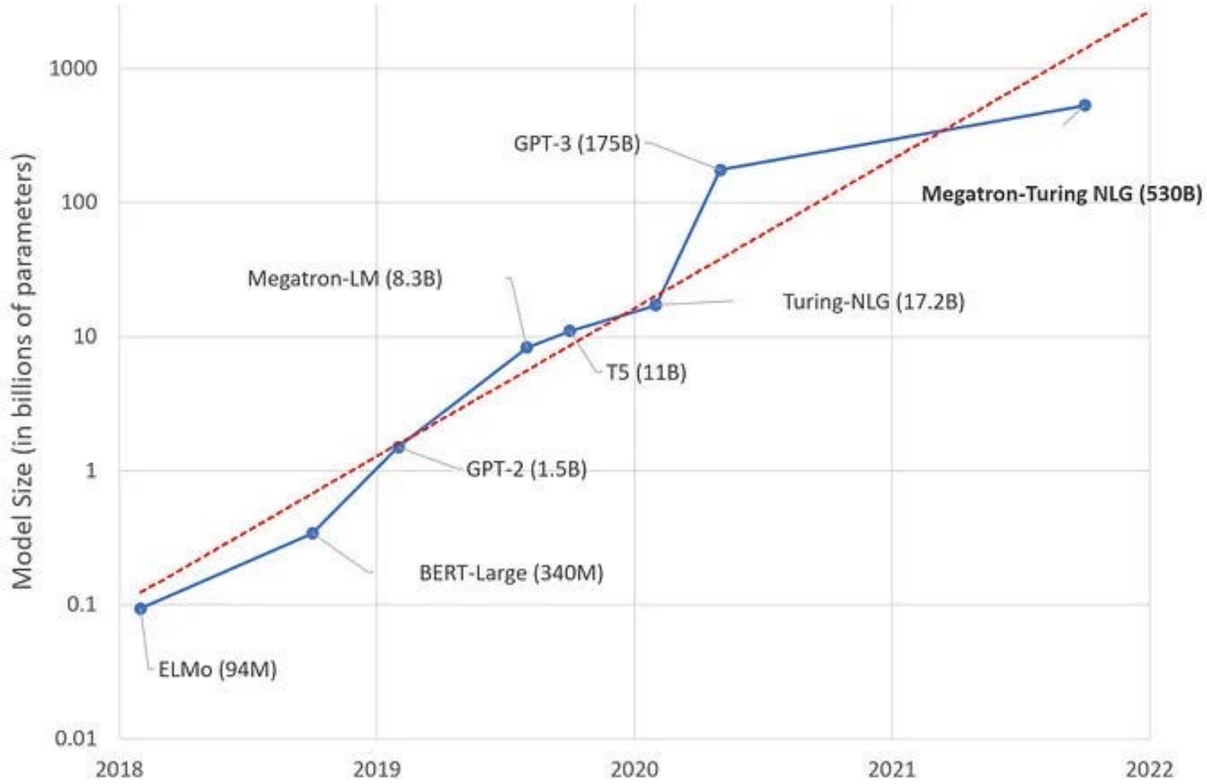
- Motivation: Why efficiency matters?
- Overview of (established) KV cache compression techniques and benchmarks
- Highlight of MagicPIG, a recent study
- Insights and future directions

# Motivation:
# Why efficiency matters?

# Why do we care about efficiency

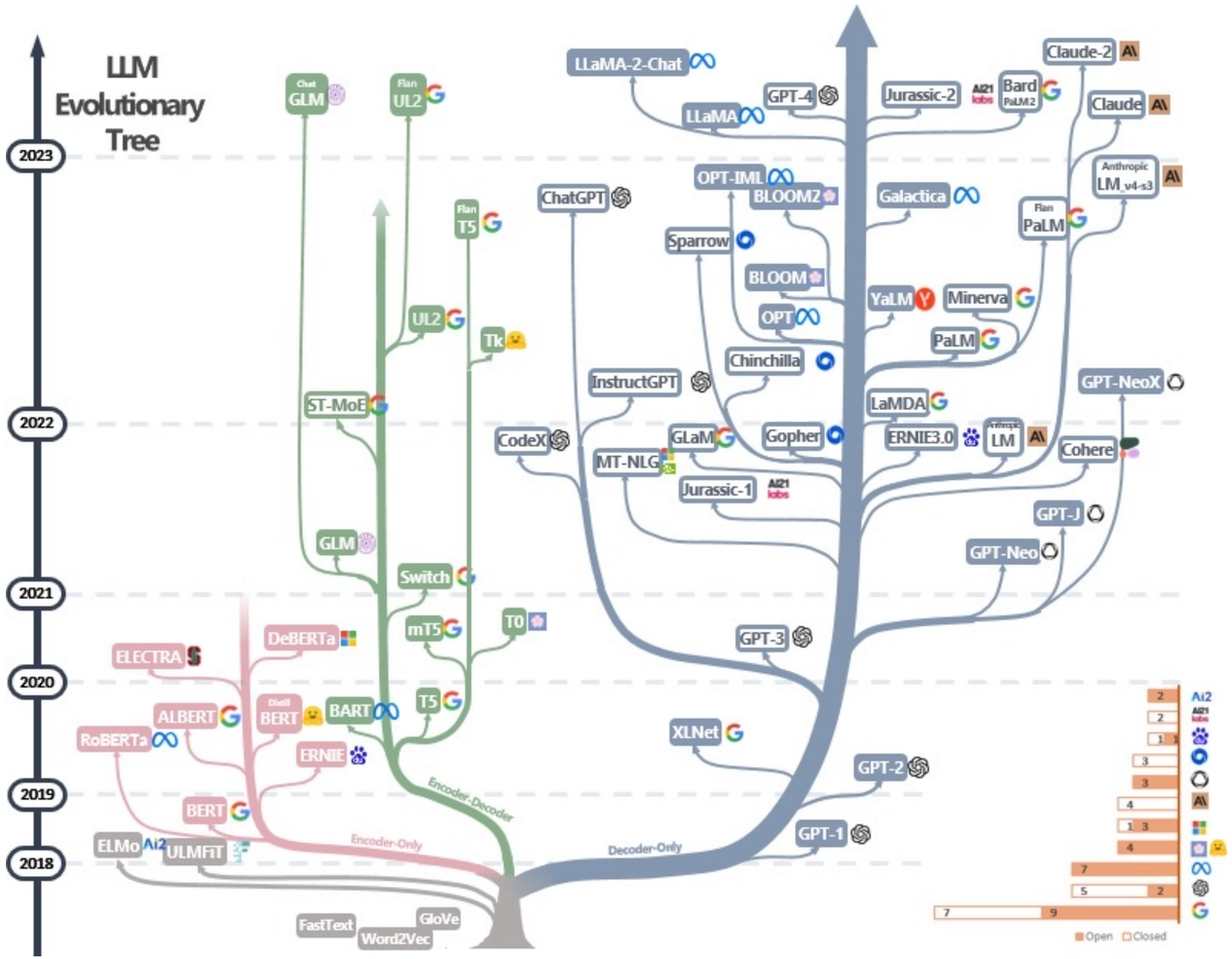
## And why this is one of the major topics in the LLM era?

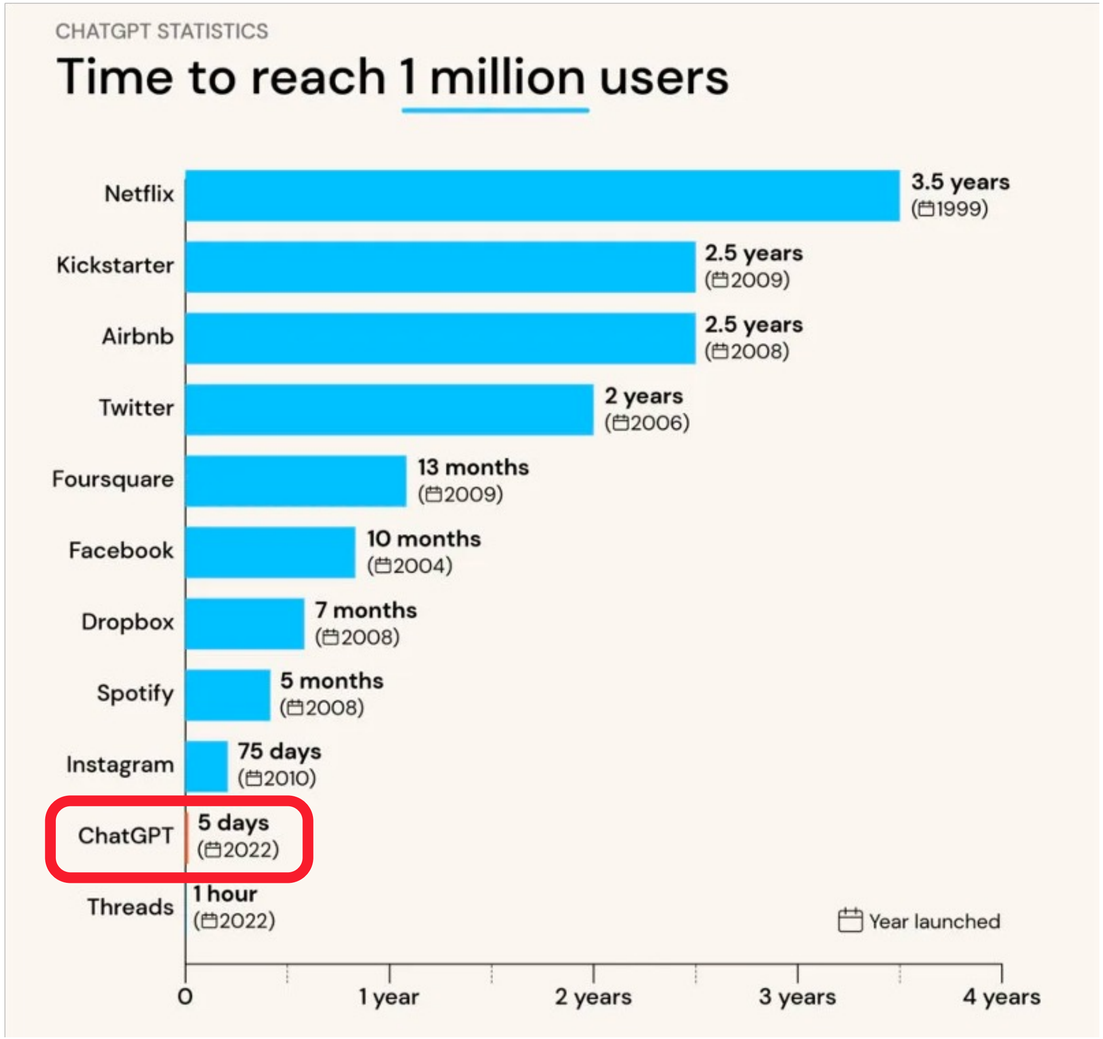



A ConvNet for the 2020s.
The LLM Multiplier (L2M2) Phenomenon: Navigating the Exponential Growth of Large Language Models

# LLM is one of the fastest growing industries



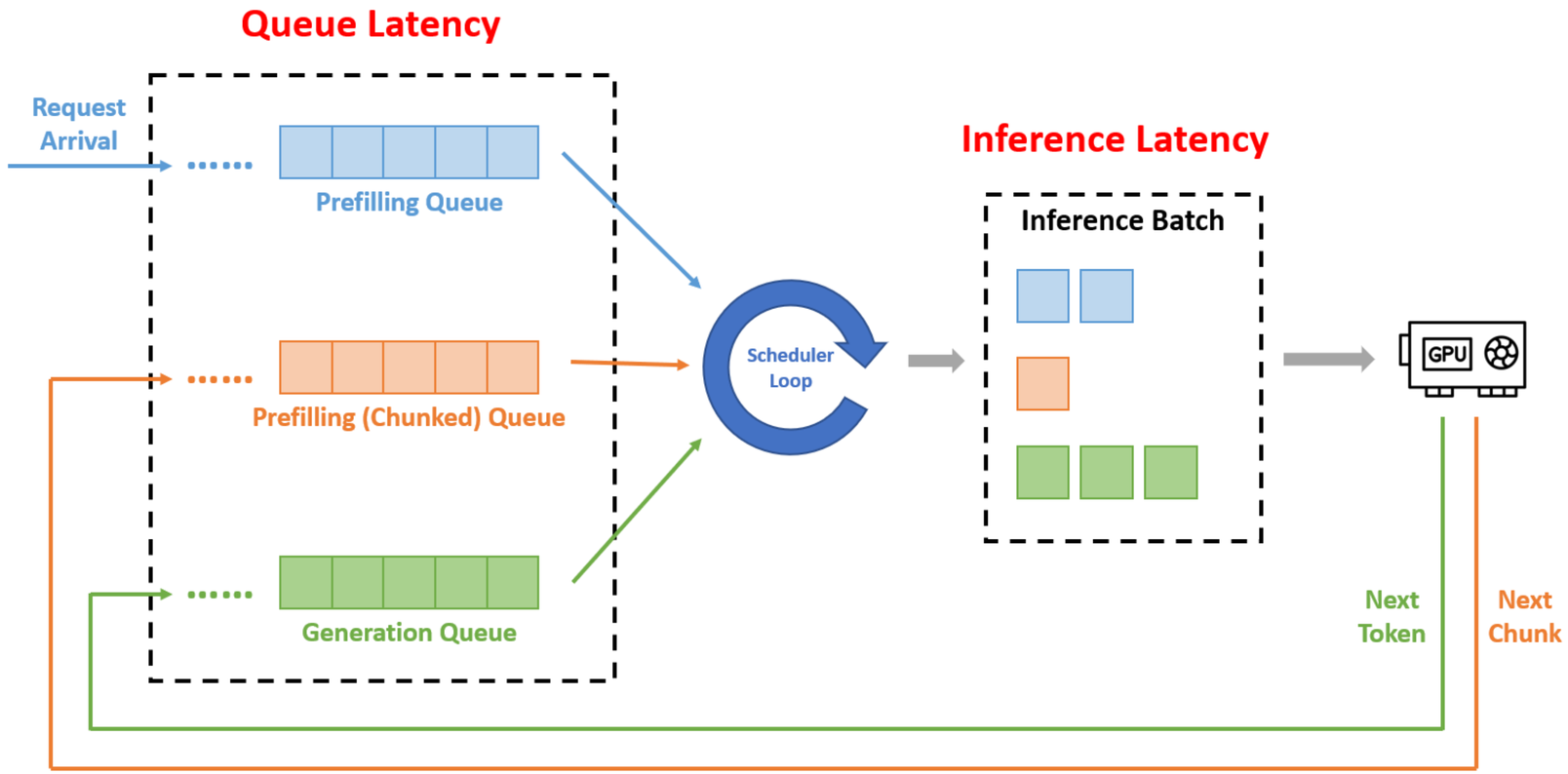

Harnessing the Power of LLMs in Practice: A Survey on ChatGPT and Beyond.
The role of generative AI in cyber security

# How are LLMs trained

## Pretrain / SFT / RLHF (or it is still the case?)


State of GPT, Microsoft Build 2023

# How are LLM served?




- Throughput (tokens per time)
- TTFT (time to first token)
- TBT (time between tokens)
- …

Latency in LLM Serving, Monsoon's Blog

# What are the bottlenecks?

## In general

- **Compute Bottlenecks**
  - Mostly happen during pretraining and prefilling
- **Memory Bottlenecks (as a rule-of-thumb)**
  - 1x model weight for weight itself
  - 1x model weight for gradient
  - 2x model weight for optimizer states (assume AdamW)
  - Activation (batch size & sequence length dependent)

# What are the bottlenecks?

## For inference

- **Compute Bottlenecks**
  - Mostly happen during ~~pretraining and~~ prefilling
- **Memory Bottlenecks (as a rule-of-thumb)**
  - 1x model weight for weight itself
  - ~~1x model weight for gradient~~
  - ~~2x model weight for optimizer states (assume AdamW)~~
  - Activation (batch size & sequence length dependent)

# Inference Efficiency Through Lower Memory Cost

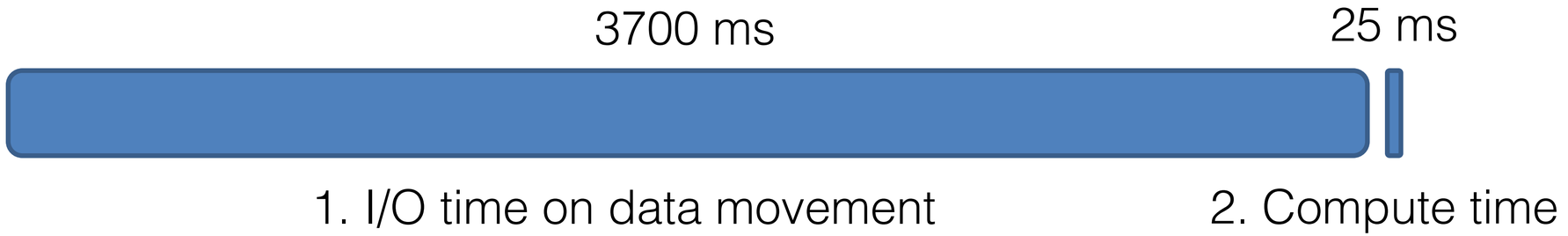
## Today's focus

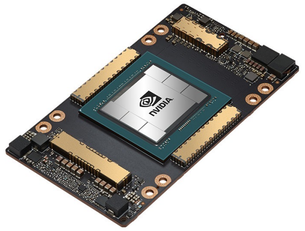
- **Weight**
  - Sparsity
  - Pruning
  - Quantization
- **Activation**
  - **Key-Value cache compression**

# Memory efficiency, but why?

**Challenge : I/O time dominates inference due to large KV cache size**

Time to generate one answer for a Llama-2-7B



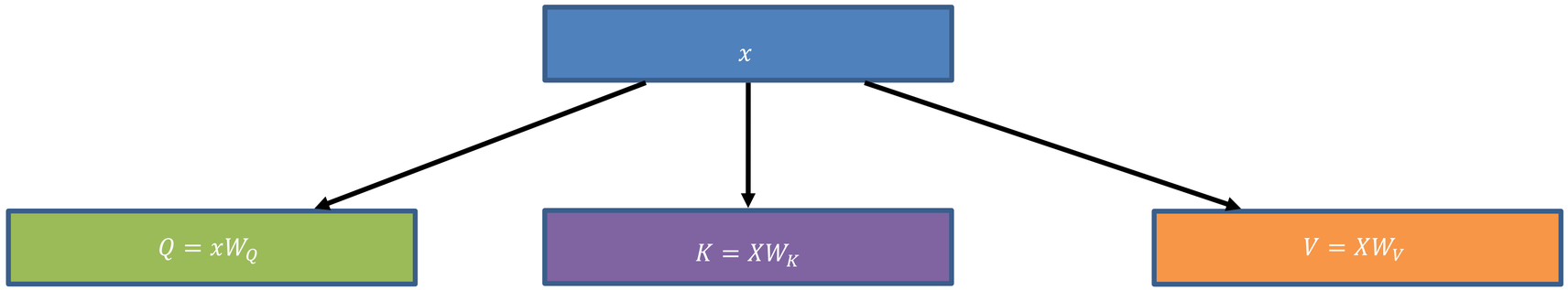



A100-80GB

1. Data movement: device memory -> registers
2: Compute with registers

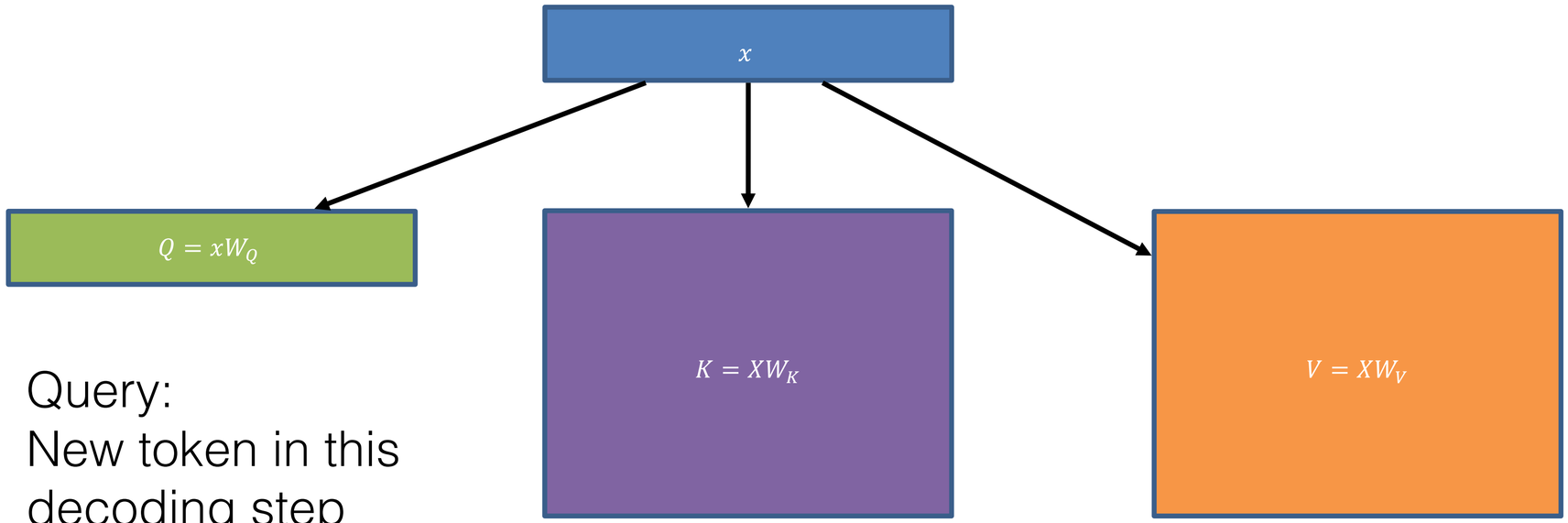
# The KV Cache Mechanism

$$A = \text{Softmax}(QK^T)V$$



# The KV Cache Mechanism

$$A = \text{Softmax}(QK^T)V$$



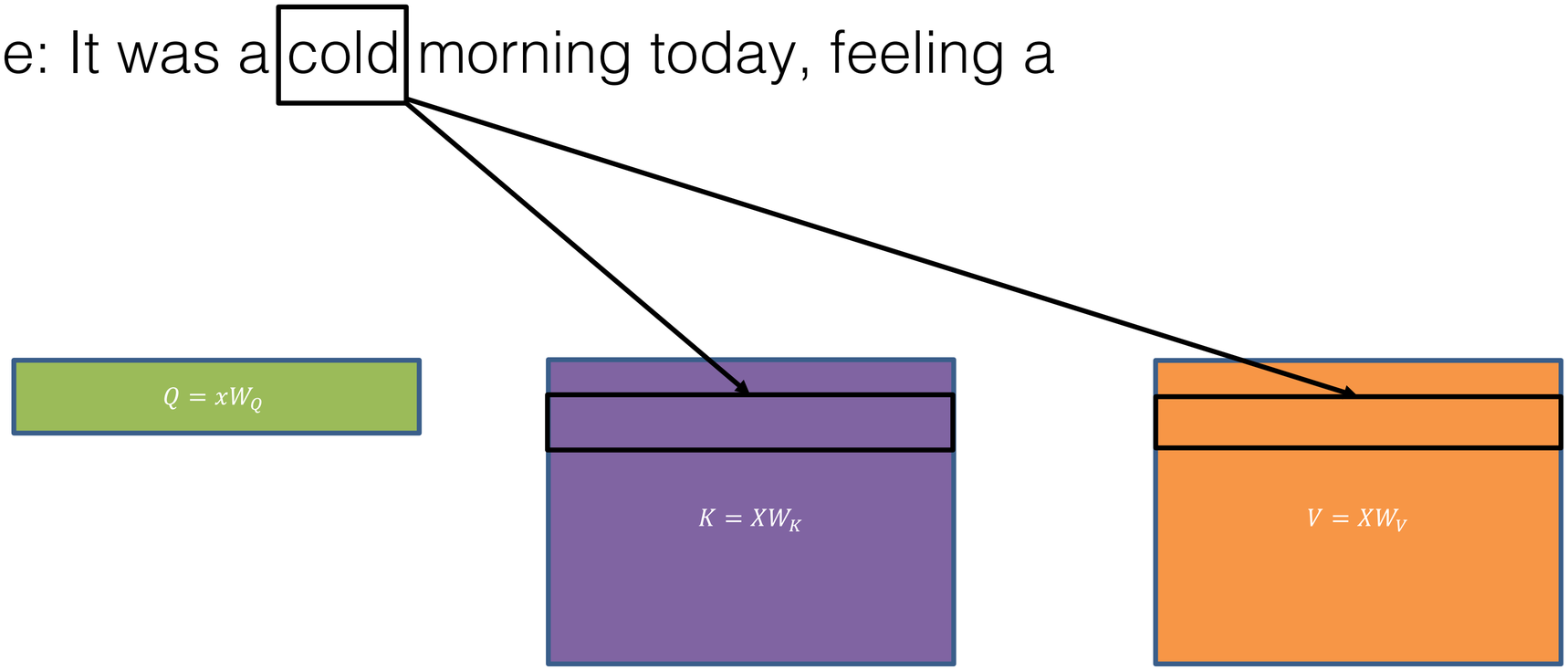

Query:
New token in this decoding step

Key:
Previous context that model should attend

Value:
Previous context but weighted by attention score

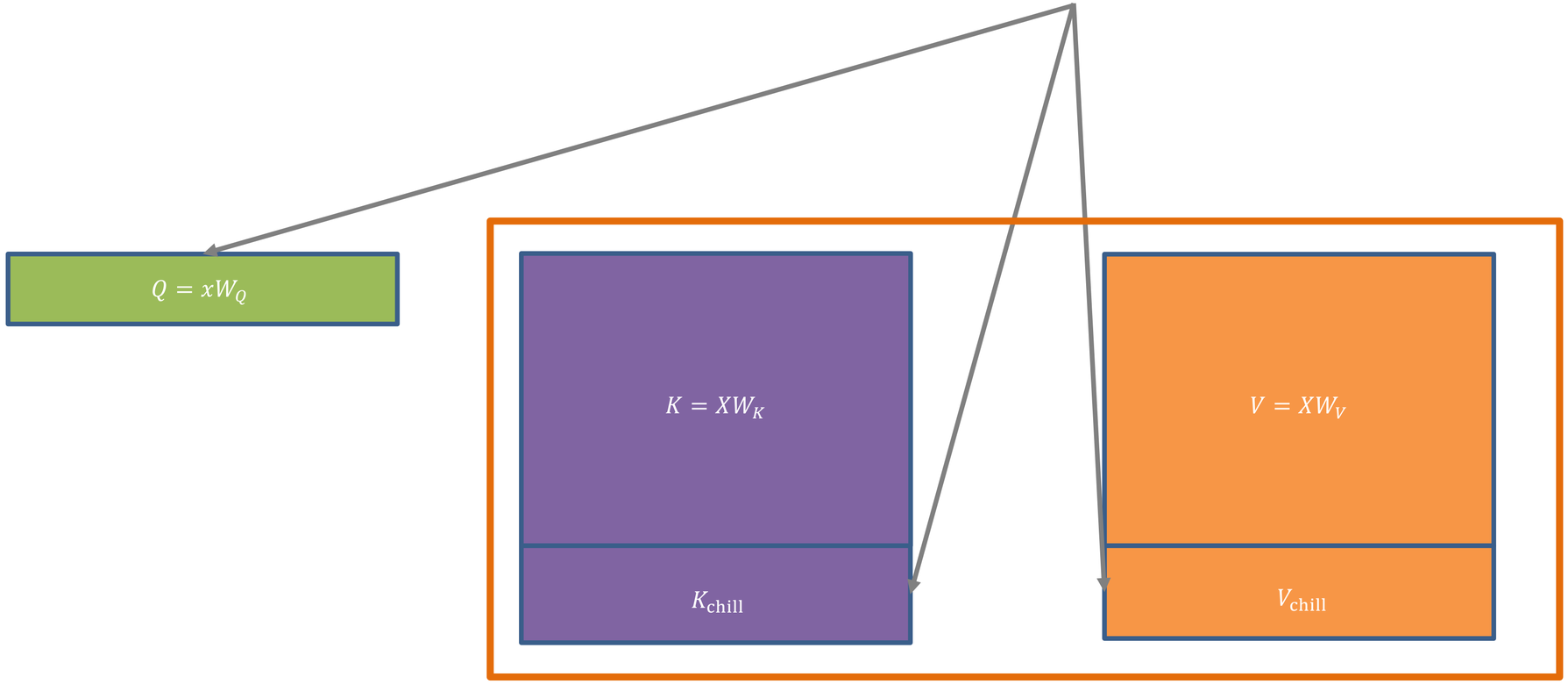
# The KV Cache Mechanism

Example: It was a cold morning today, feeling a



# The KV Cache Mechanism

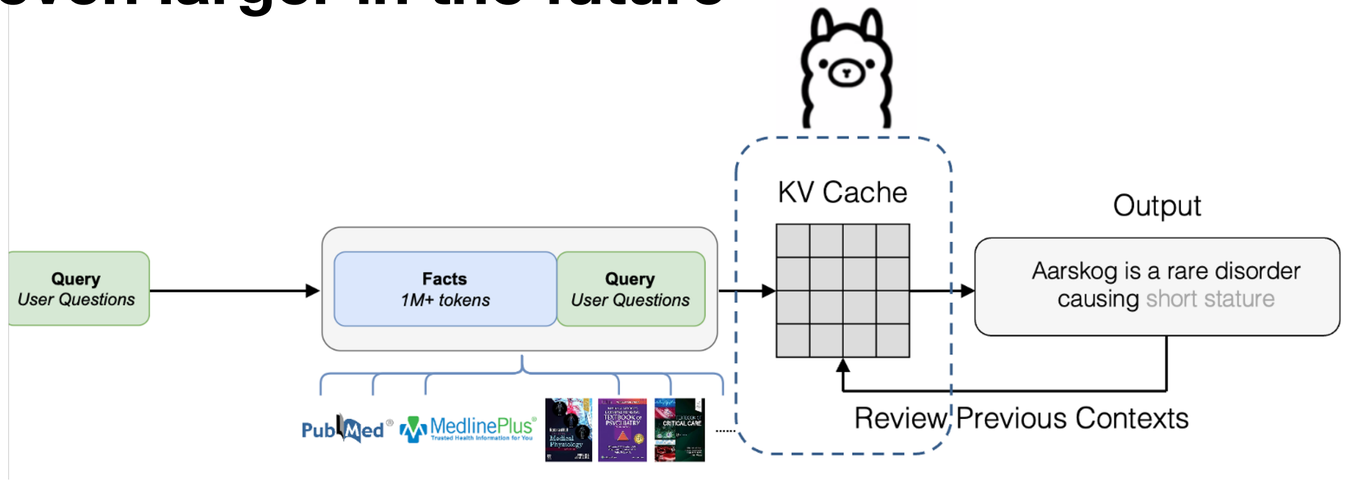
Example: It was a cold morning today, feeling a chill

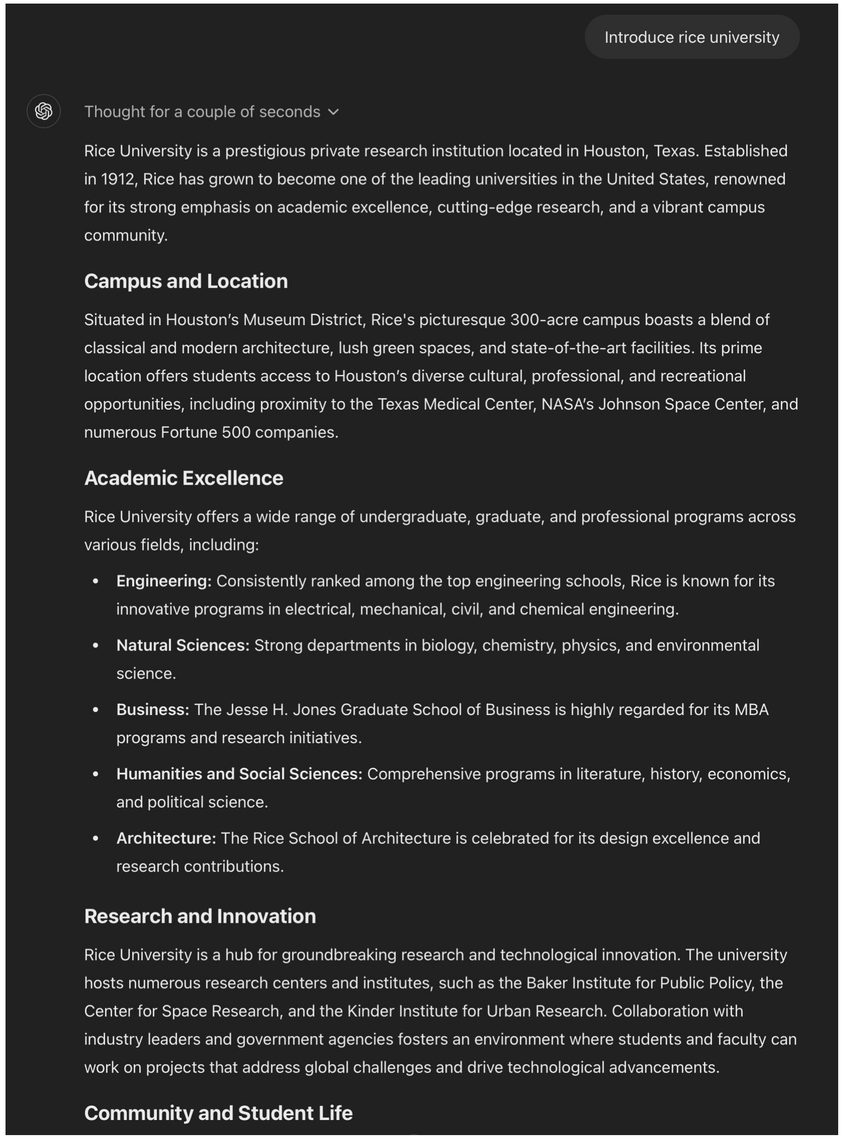



Streaming Data Structure: KV Cache

# Takeaway

- KV cache is a must have for efficient serving that trades space for speed
- KV cache grows linearly as of batch size
- KV cache grows linearly as of sequence length
- **… so it is huge, and we safely predict it will be even larger in the future**

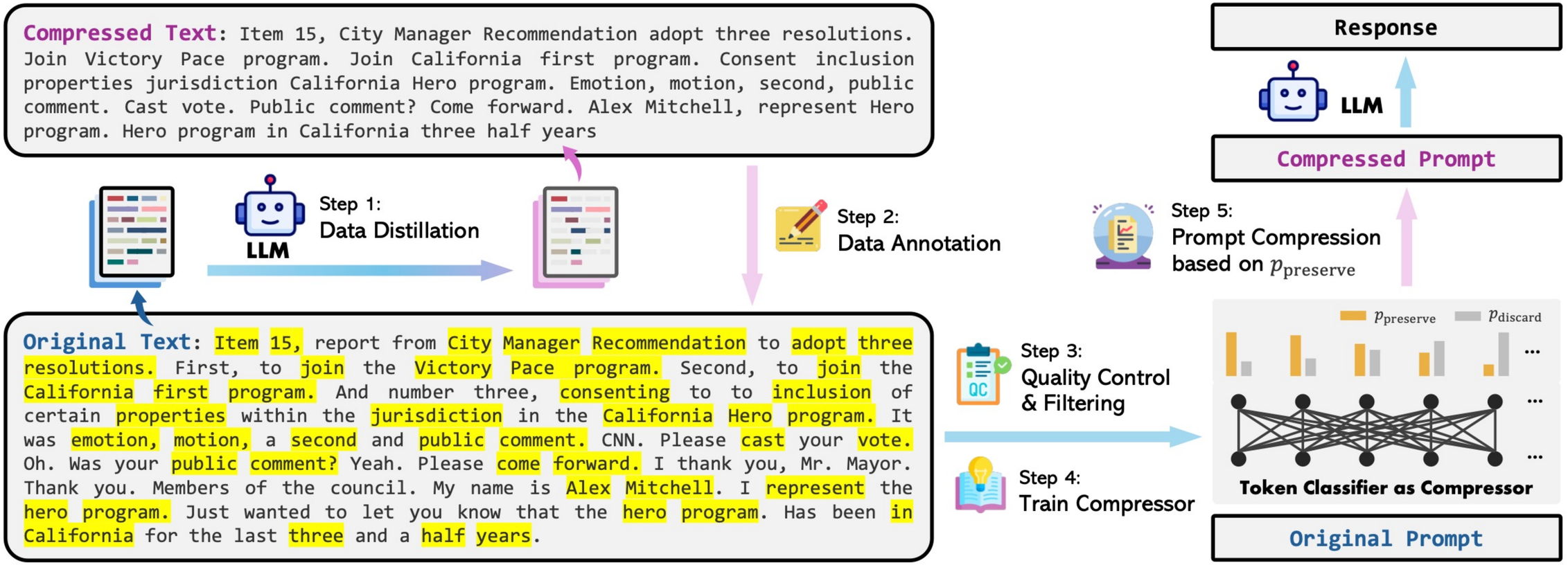


# Schools of Thought in KV Cache Compression

# Prompt Compression

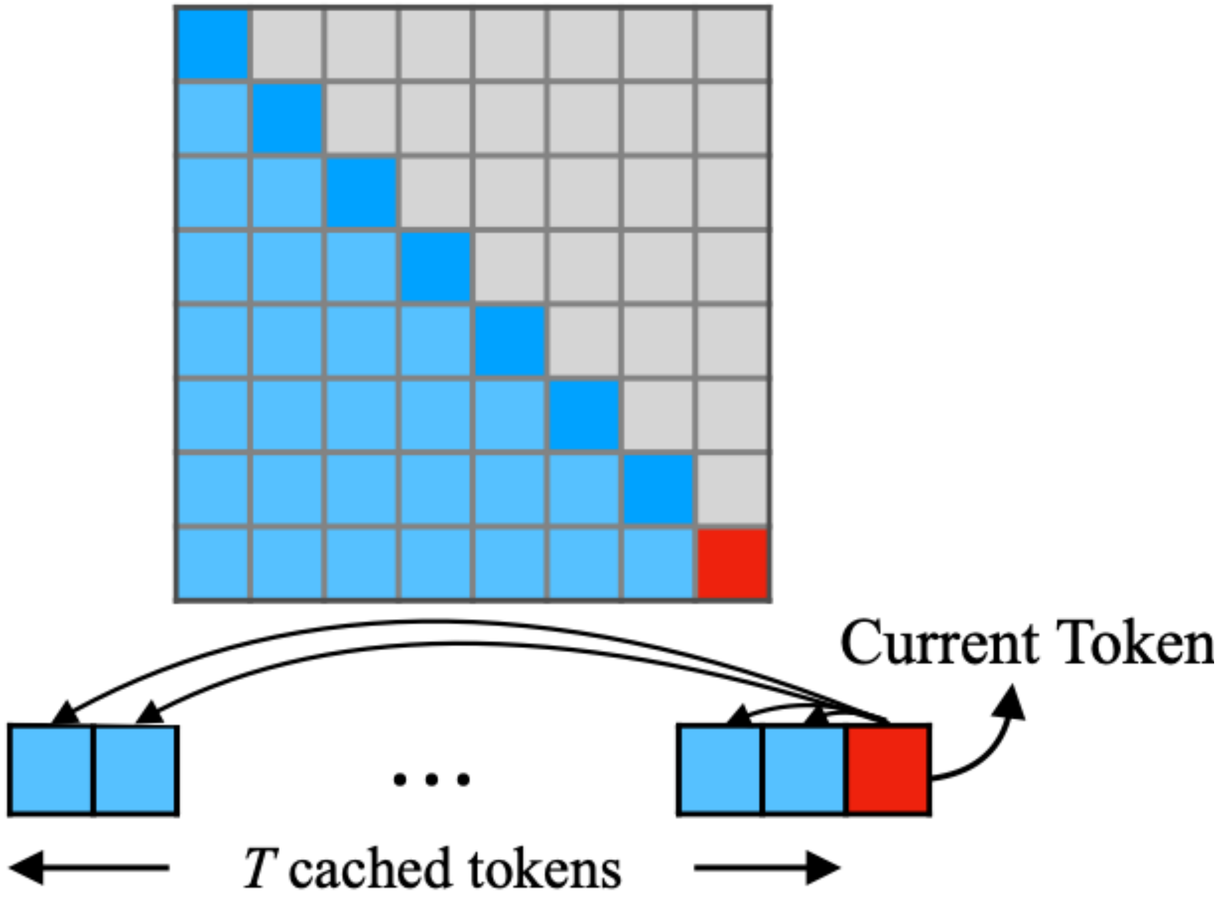
## LLMLingua2




LLMLingua-2: Data Distillation for Efficient and Faithful Task-Agnostic Prompt Compression

# Token Dropping

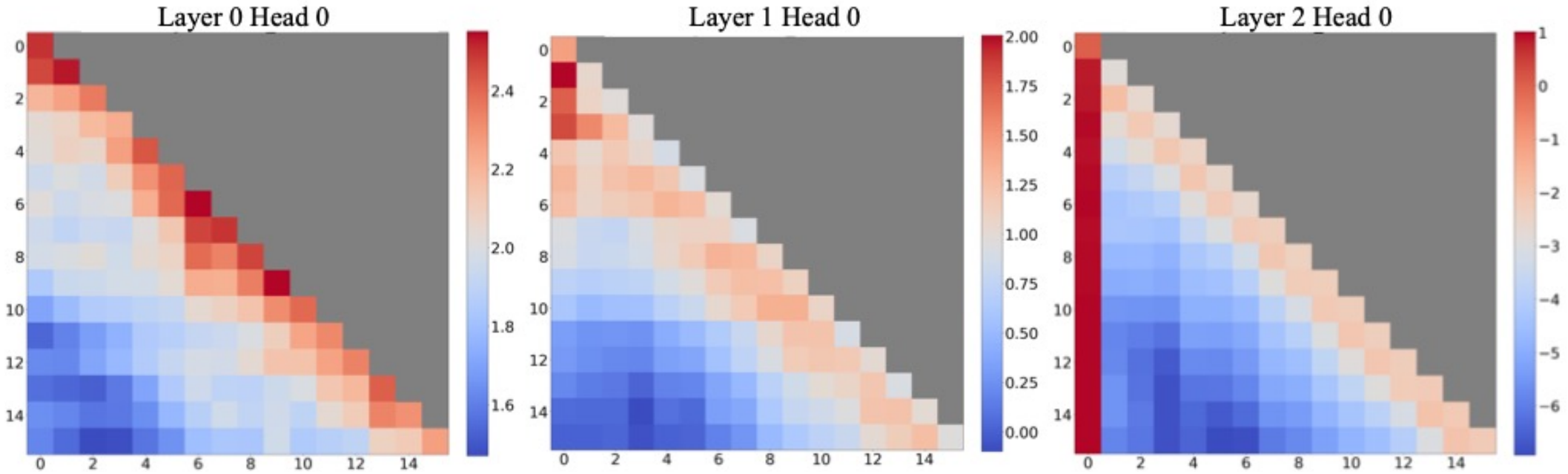
## Do we need the full attention matrix?




Efficient Streaming Language Models with Attention Sinks
Infinite-LLM: Efficient LLM Service for Long Context with DistAttention and Distributed KVCache

# Token Dropping

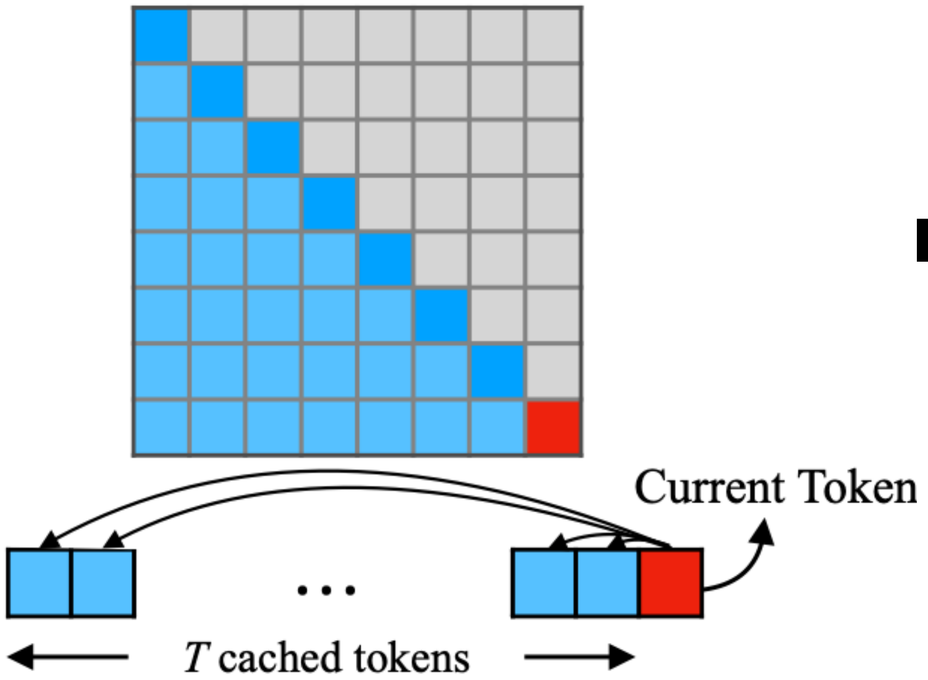
## Initial and recent tokens are heavily emphasized

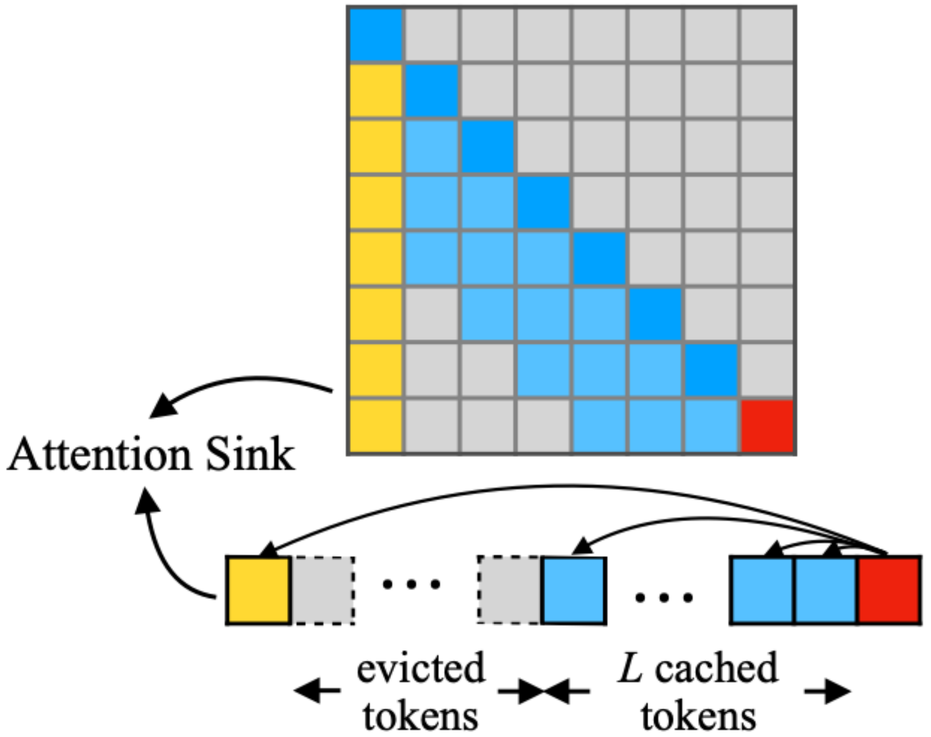



Efficient Streaming Language Models with Attention Sinks
Infinite-LLM: Efficient LLM Service for Long Context with DistAttention and Distributed KVCache

# Token Dropping

## StreamingLLM / Infinite-LLM



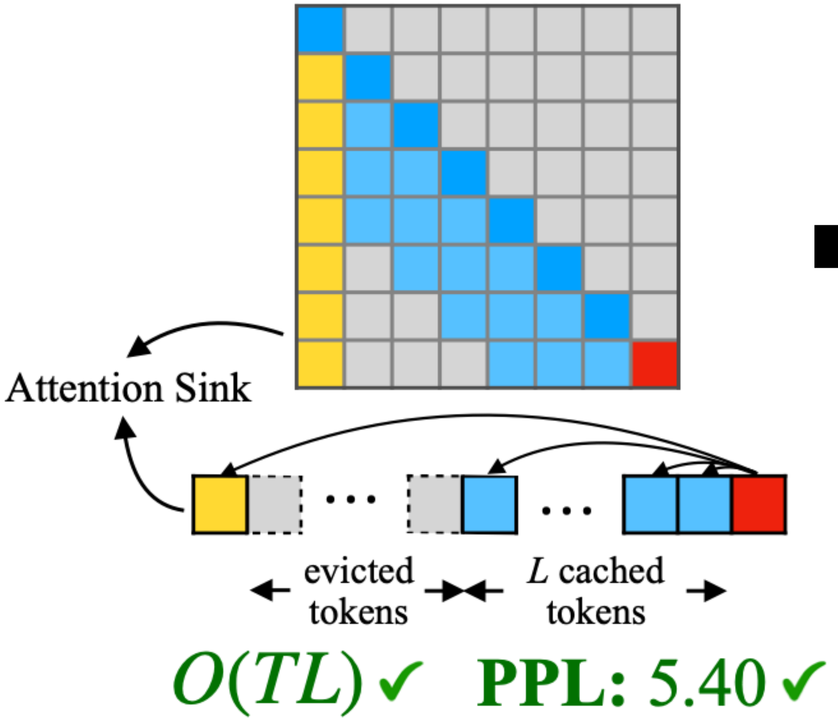

$O(T^2)$ ✗ **PPL: 5641** ✗

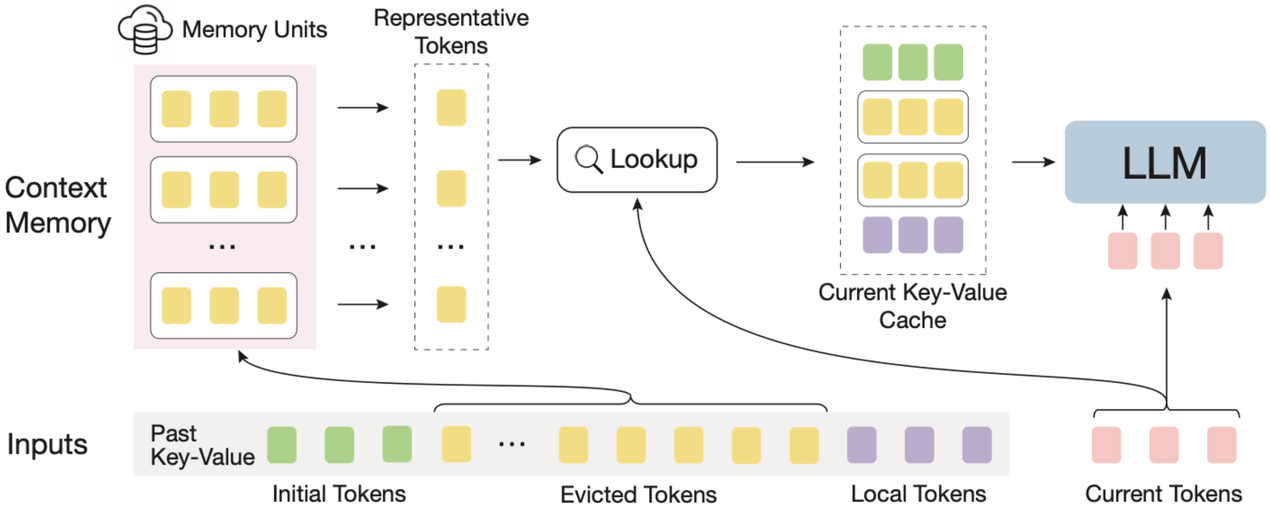
$O(TL)$ ✓ **PPL: 5.40** ✓

Efficient Streaming Language Models with Attention Sinks
Infinite-LLM: Efficient LLM Service for Long Context with DistAttention and Distributed KVCache

# Token Dropping

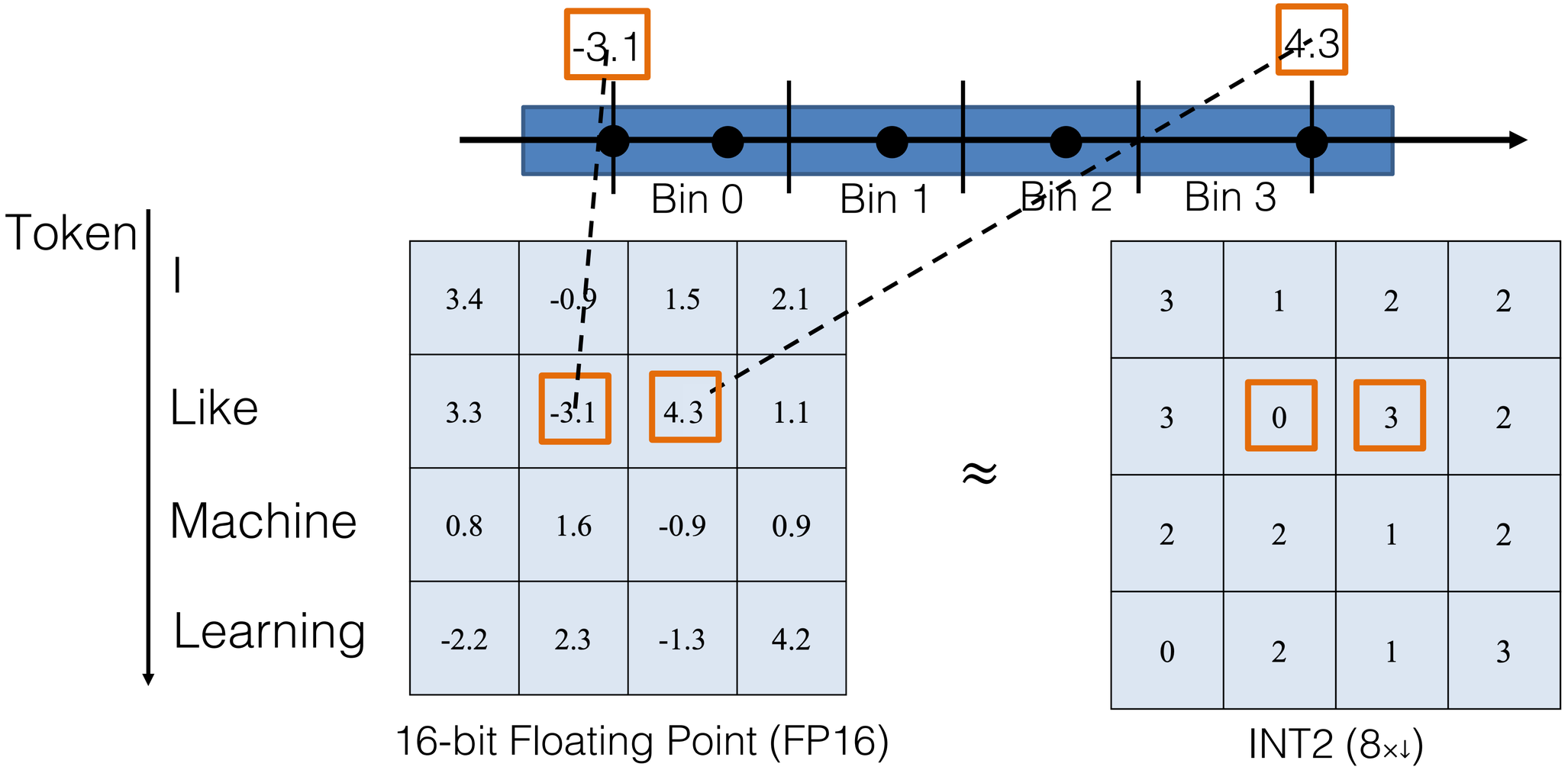
## InfLLM




InfLLM: Training-Free Long-Context Extrapolation for LLMs with an Efficient Context Memory

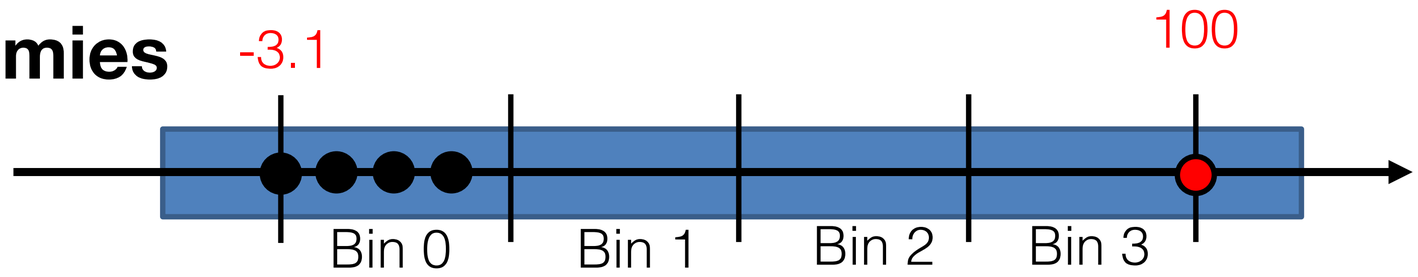
# Quantization




Example: 2bit quantization

# Quantization

## Outliers are the enemies




Token

I

Like

Machine

Learning

| | | | |
|---|---|---|---|
| **100** | -0.9 | 1.5 | 2.1 |
| 3.3 | -3.1 | 4.0 | 1.1 |
| 0.8 | 1.6 | -0.9 | 0.9 |
| -2.2 | 2.3 | -1.3 | 4.2 |

16-bit Floating Point (FP16)

≈

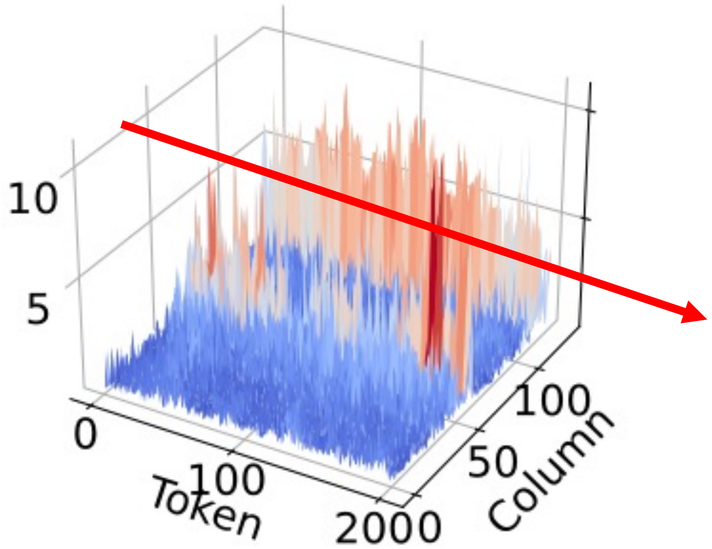
| | | | |
|---|---|---|---|
| 3 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |

INT2 (8×↓)

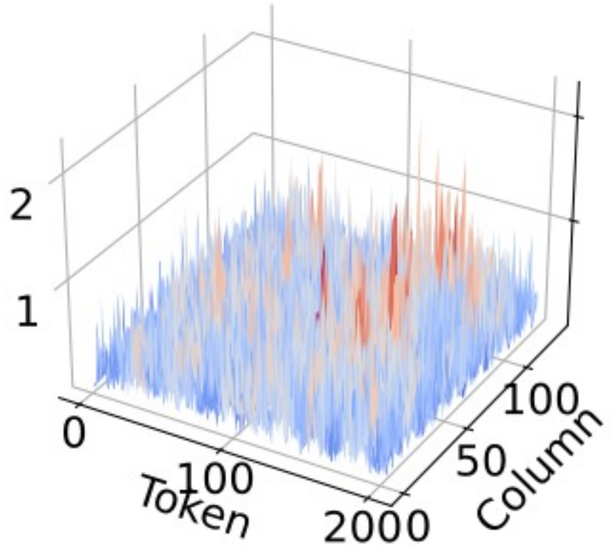
No difference after quantization!

# Quantization

## Outliers come in pattern




Key Cache

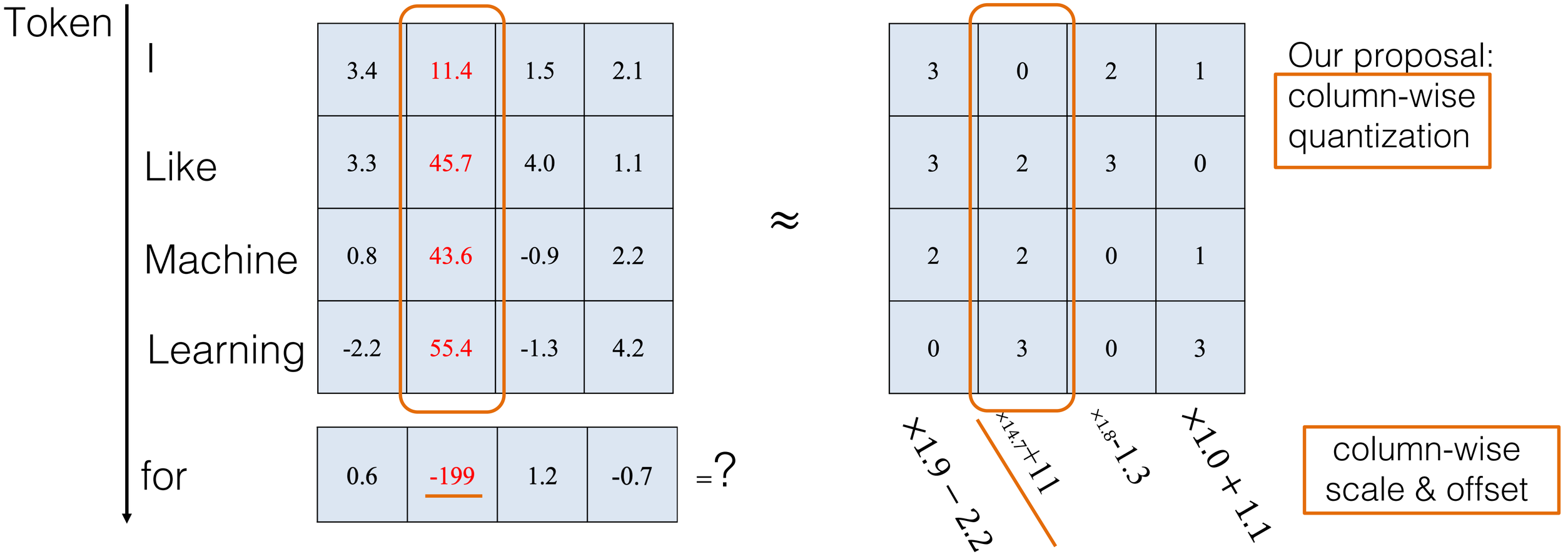



Value Cache

KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache

# Quantization

## Outliers come in pattern — so we leveraged them (KIVI)

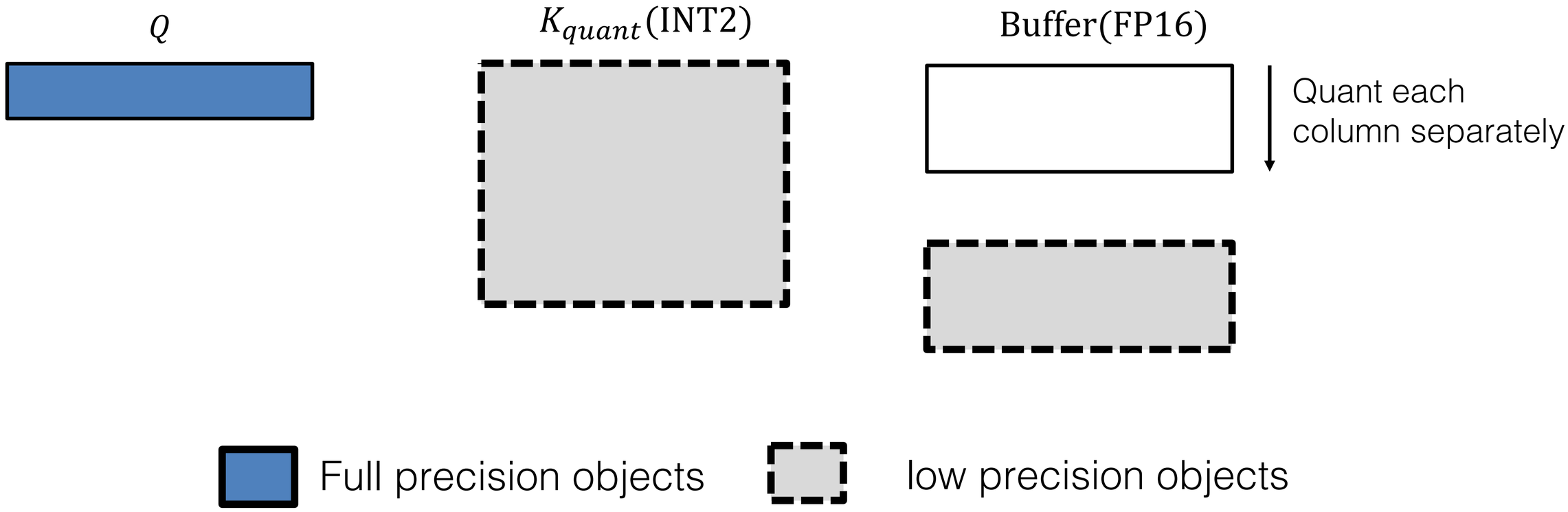



KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache

# Quantization

## How to make column-wise quantization streaming capable?

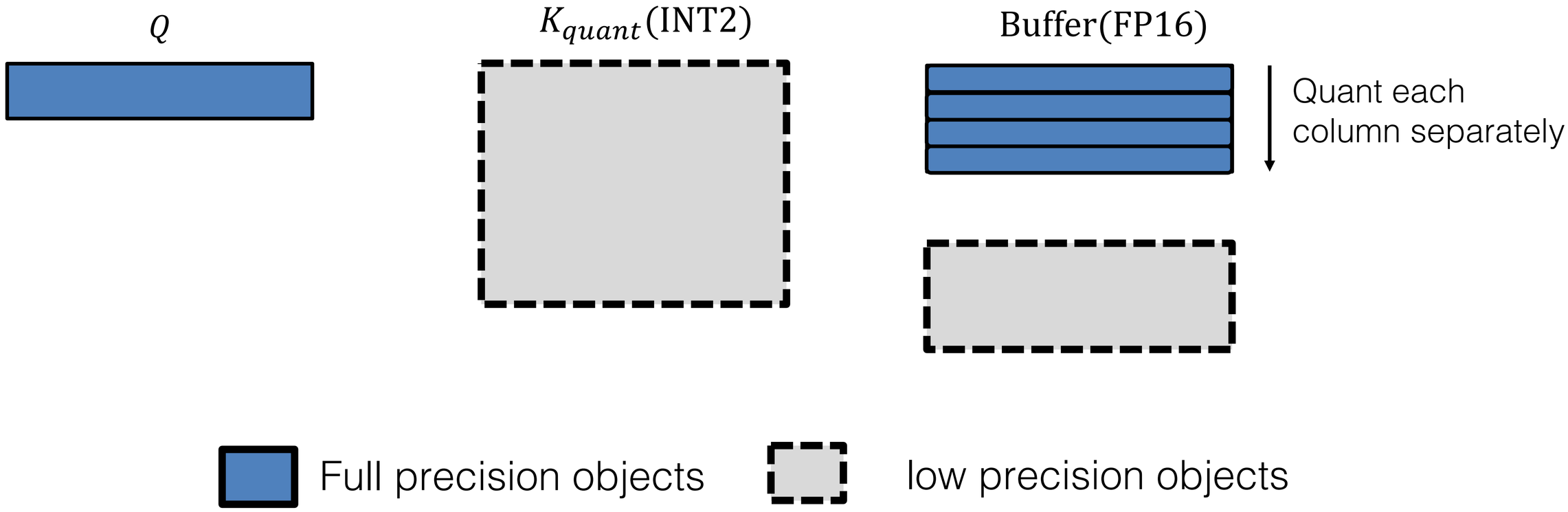
…It was



# Quantization

## How to make column-wise quantization streaming capable?

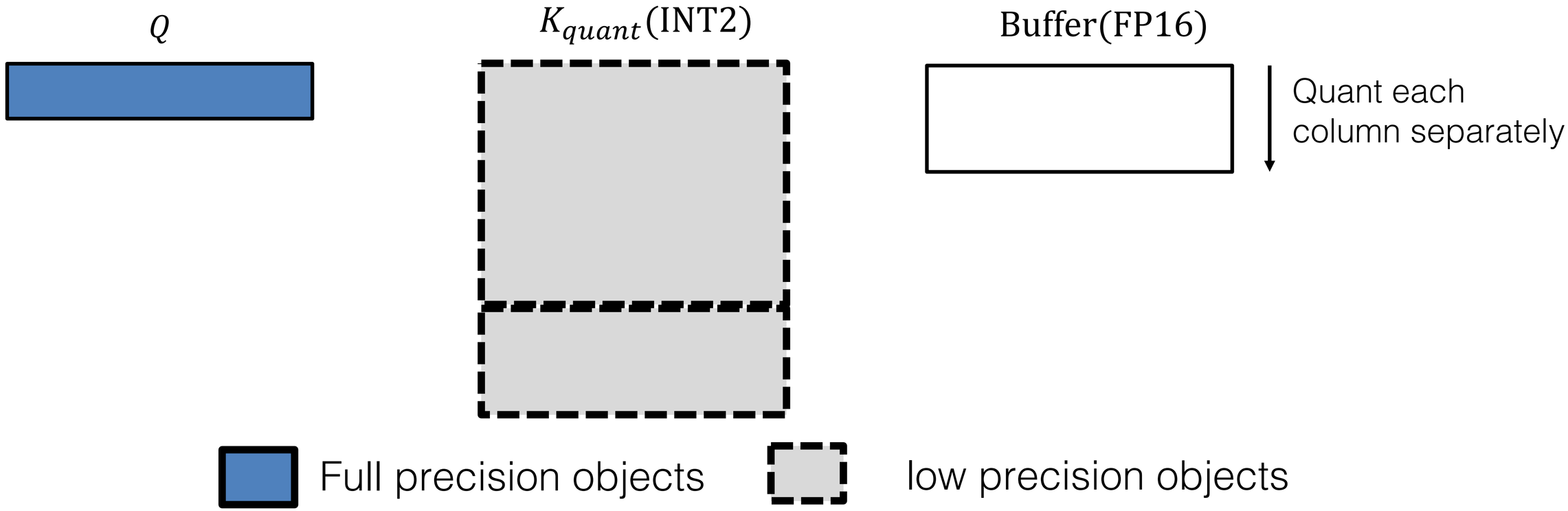
…It was a cold morning today



# Quantization

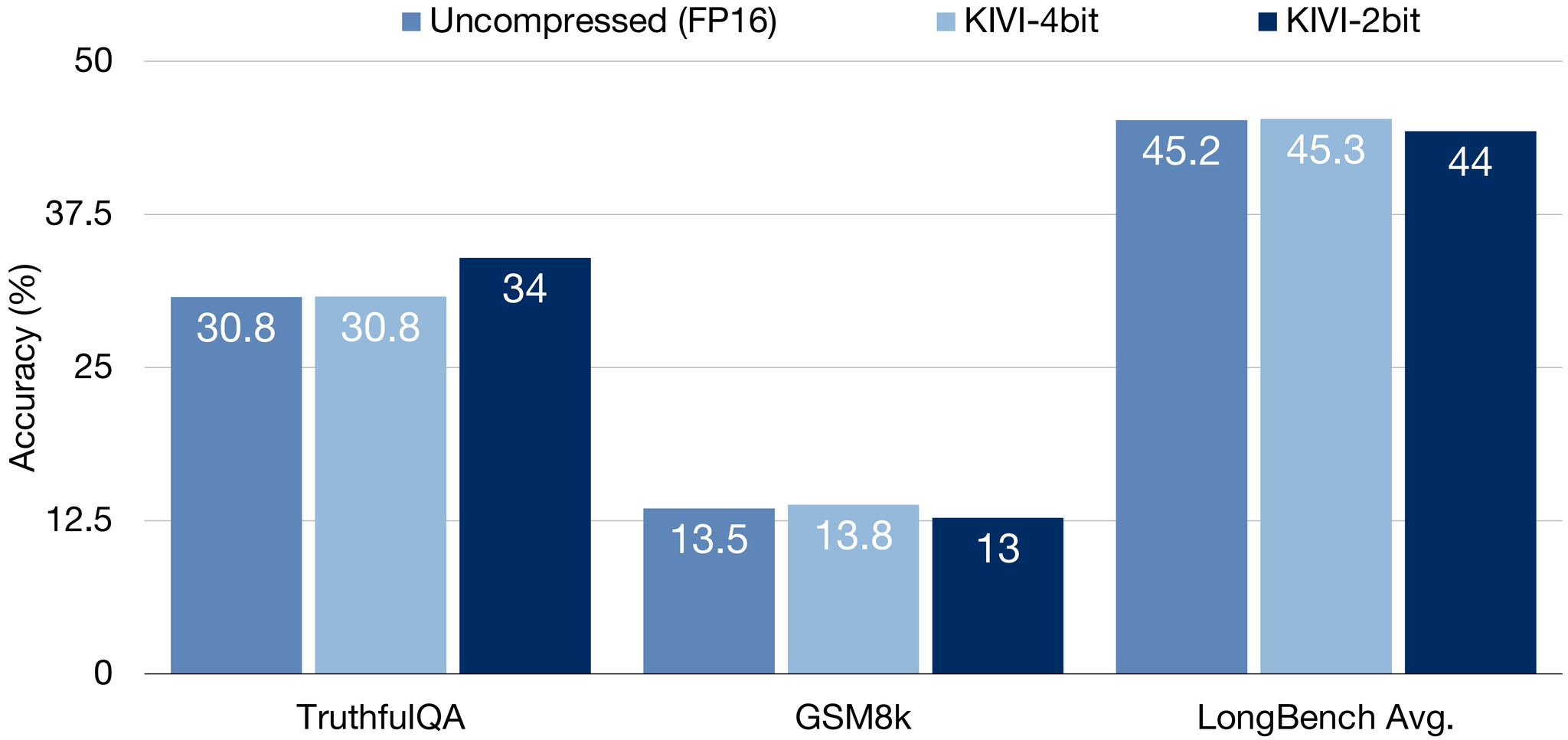
## How to make column-wise quantization streaming capable?

…It was a cold morning today



# Quantization

## KIVI's Task Results

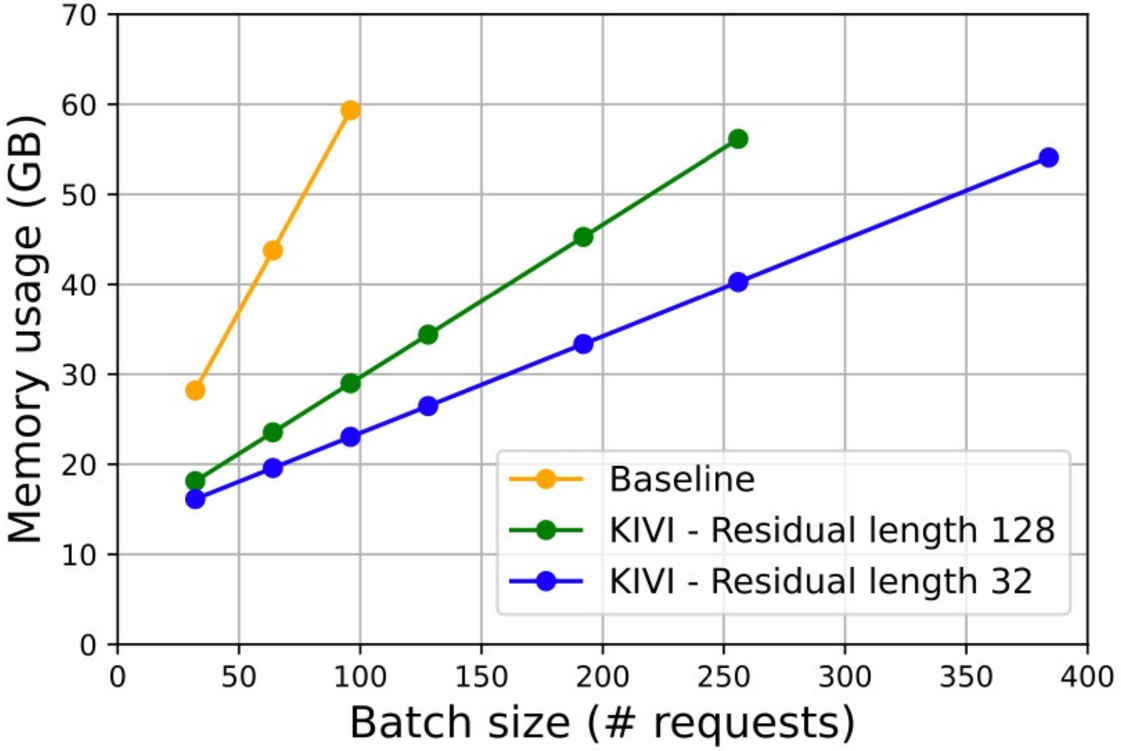


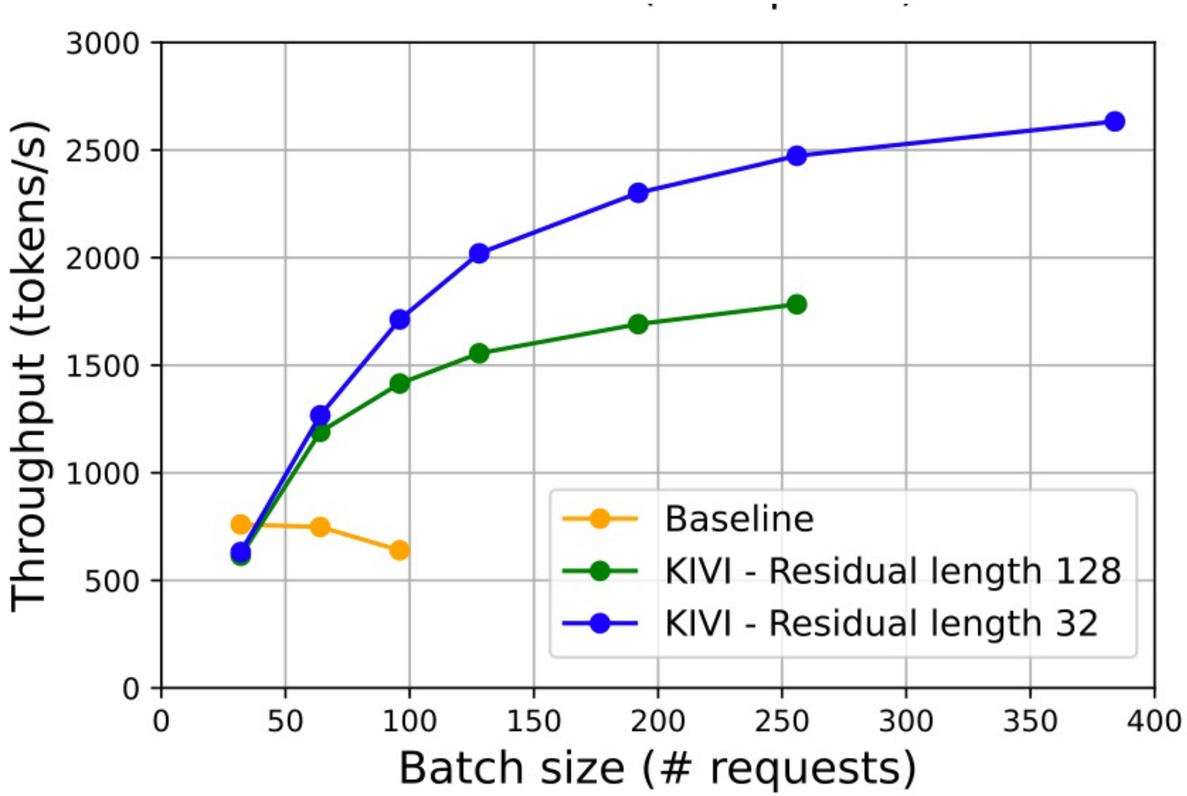

KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache

KV Cache Compression, But What Must We Give in Return? A Comprehensive Benchmark of Long Context Capable Approaches

# Quantization

## KIVI's Efficiency Results

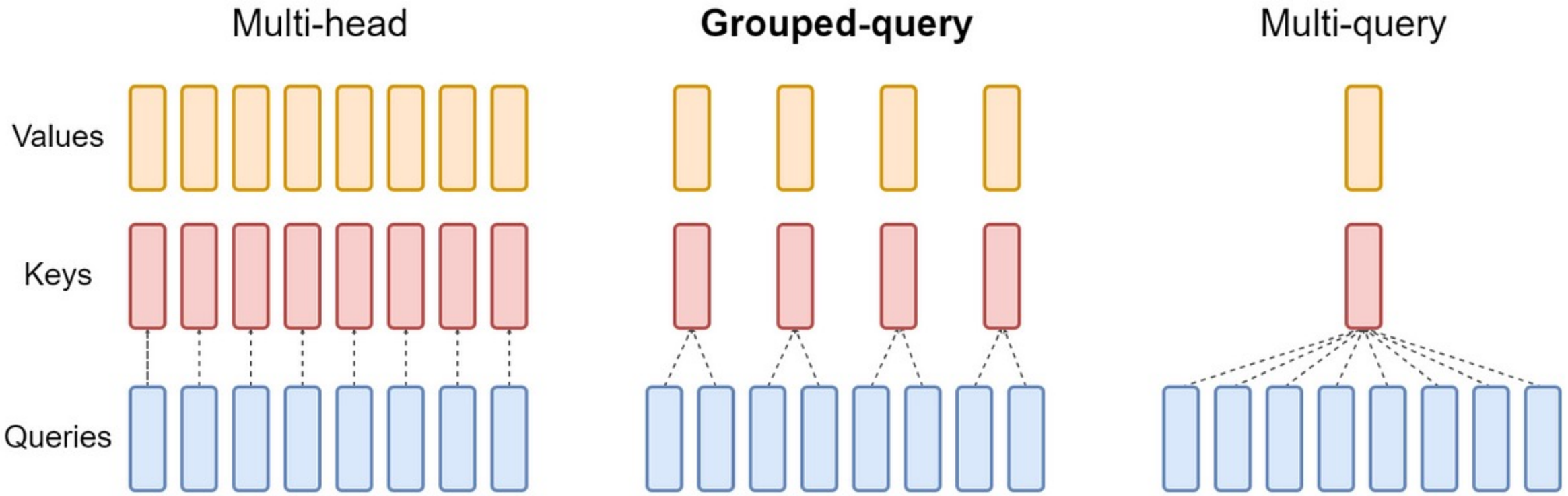



KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache

# Architecture Tweak

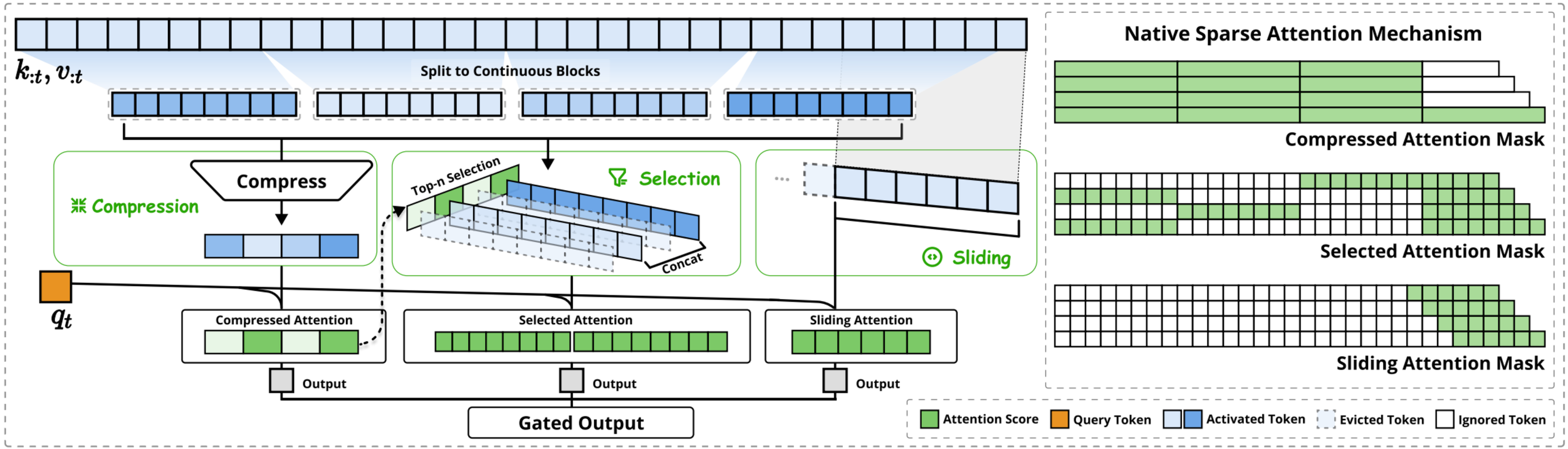
## MHA, GQA, etc.




GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints

# Architecture Tweak

## Native Sparse Attention (NSA) and MoBA




Native Sparse Attention: Hardware-Aligned and Natively Trainable Sparse Attention
MoBA: Mixture of Block Attention for Long-Context LLMs
Long Context Compression with Activation Beacon

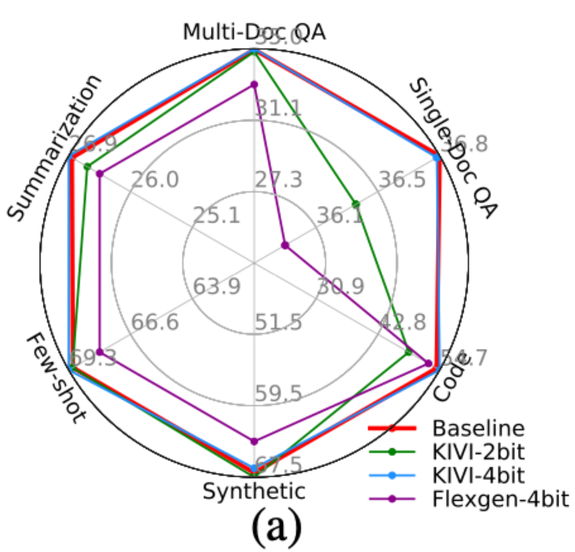
# KV cache free & hybird architectures

## Mamba, RWKV, Jamba, RecurrentGemma, GLA, DeltaNet…

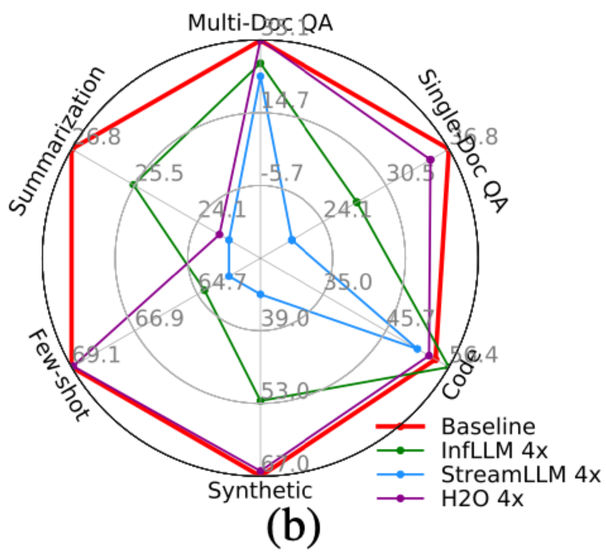

Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality
RWKV: Reinventing RNNs for the Transformer Era
Griffin: Mixing Gated Linear Recurrences with Local Attention for Efficient Language Models
Jamba: A Hybrid Transformer-Mamba Language Model
Gated Linear Attention Transformers with Hardware-Efficient Training
Parallelizing Linear Transformers with the Delta Rule over Sequence Length

# Benchmark

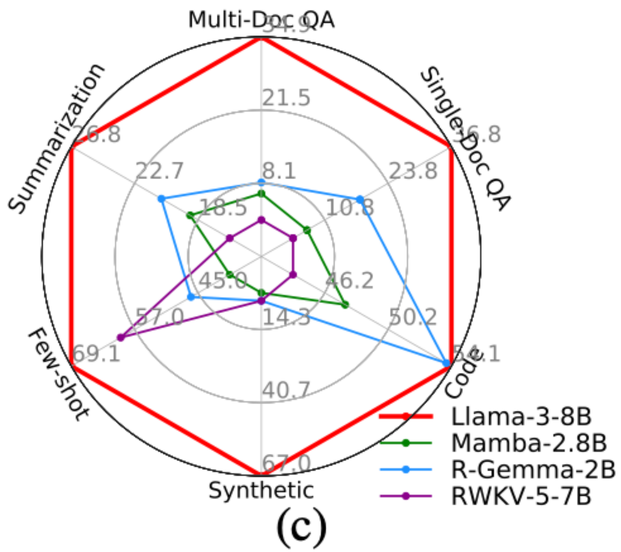
## LongBench Average



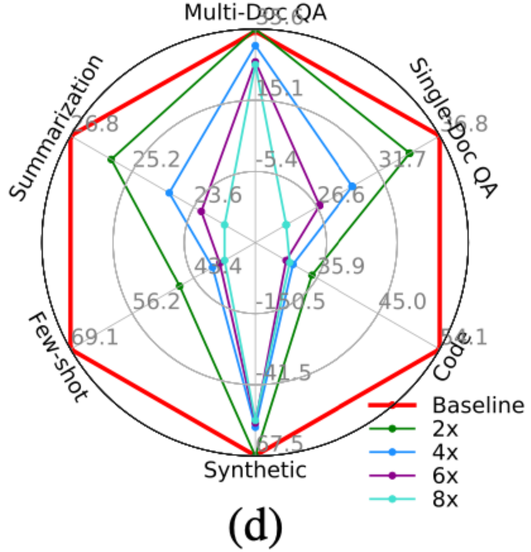

Quantization




Token Dropping



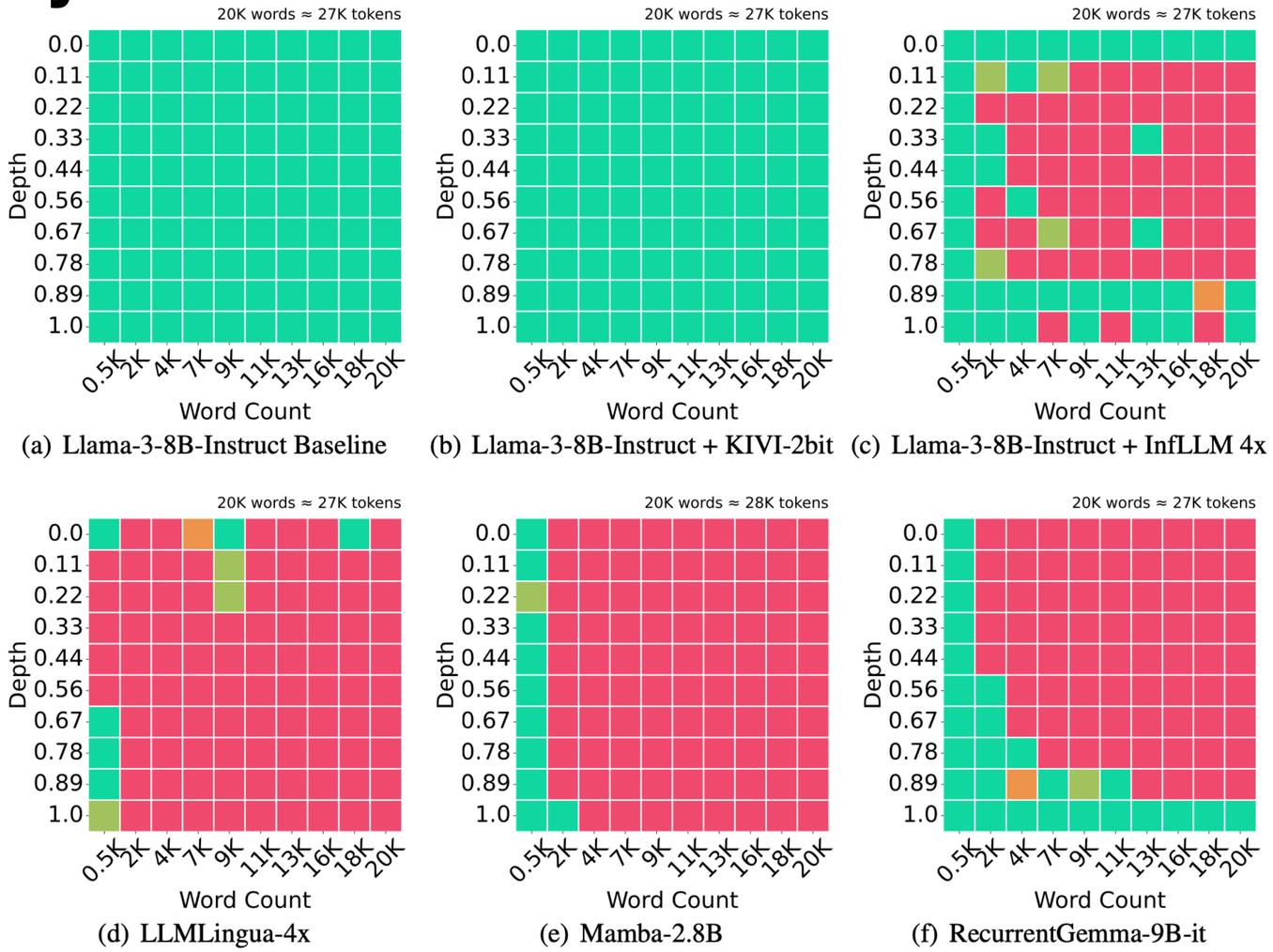

Linear Sequence




Prompt Compression

# Benchmark

## Needle-in-a-Haystack




(a) Llama-3-8B-Instruct Baseline (b) Llama-3-8B-Instruct + KIVI-2bit (c) Llama-3-8B-Instruct + InfLLM 4x

(d) LLMLingua-4x (e) Mamba-2.8B (f) RecurrentGemma-9B-it

# What’s Missing in Current Benchmarks?

**Automatic metrics vs LLM-as-a-Judge**

*HELMET: How to Evaluate Long-Context Language Models Effectively and Thoroughly*

**Natural variant-length input vs Synthetic length-controllable input**

*LongBench v2: Towards Deeper Understanding and Reasoning on Realistic Long-context Multitasks*

*100-LongBench: Are de facto Long-Context Benchmarks Literally Evaluating Long-Context Ability?*

**Single-round vs Multi-round evaluations**

*SharedContextBench: How Lossy are Long-Context Methods in KV Cache Reuse*

**Retrieval-only vs Some reasoning components**

*Same Task, More Tokens: the Impact of Input Length on the Reasoning Performance of Large Language Models*

# Takeaway

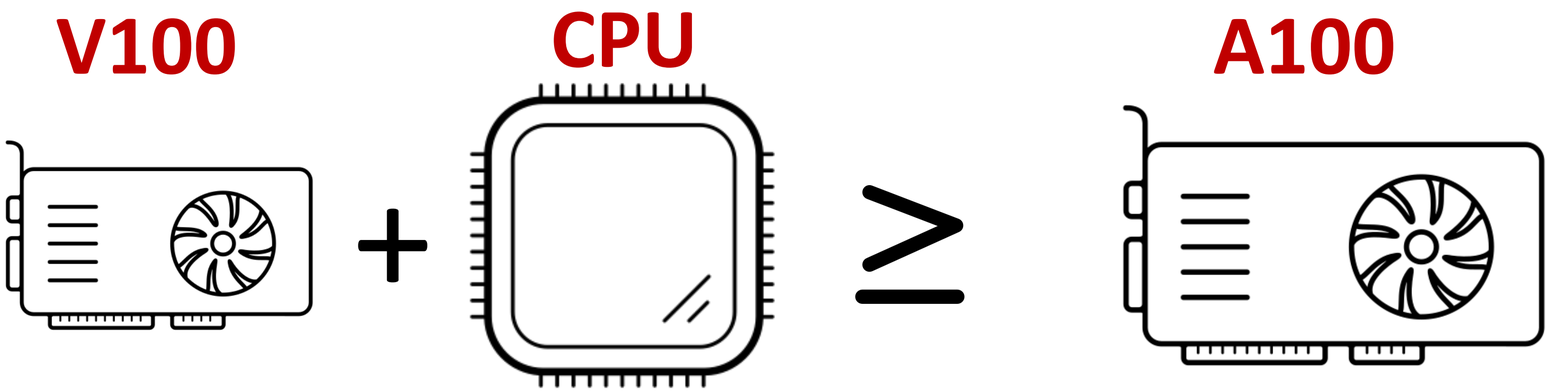
- **Token Dropping** methods can be highly effective at certain scenarios, but likely not a fail-safe solution unless some offload + retrieval mechanisms are also implemented.
- **Quantization** can be near loss-less at 4bit, but exact prefill require extra considerations.
- **Constant KV cache architectures** still struggle at medium compression ratio, though some promising improvement are showing.
- **Hard prompt compression** methods are unlikely to retain as much of information as compression along other dimensions. We will likely see a rediscovery of this under the “CoT pruning” realm.
- **Pretraining** with compression show promising results.

- **More comprehensive and more distinguishable benchmarks** are needed to evaluate the true effectiveness of models.
- **More co-design approaches** would likely achieve great trade-offs: compression-extension, pretrain-compression, posttrain-compression…

# MagicPIG, A Recent Work

# MagicPIG (🔥)

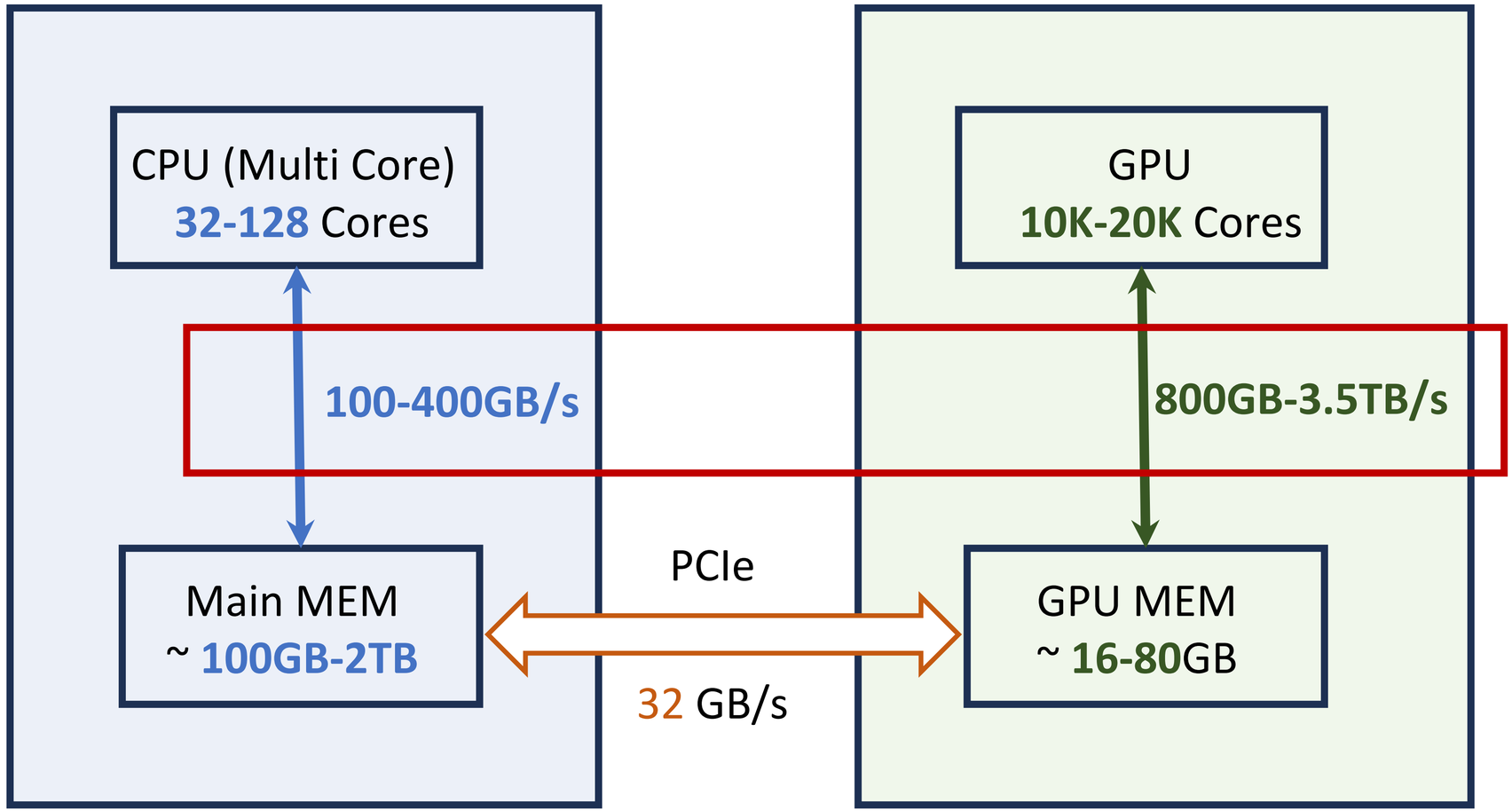
## CPU as your Infinite Memory




AI Agents

| Model | A100-40G | V100 + MagicPIG |
|---|---|---|
| CodeLlama-7B-16K | 80 token/s | **182** token/s |
| CodeLlama-13B-16K | OOM | **43** token/s |
| Llama3.1-8B-128K | 38 token/s | **44** token/s |

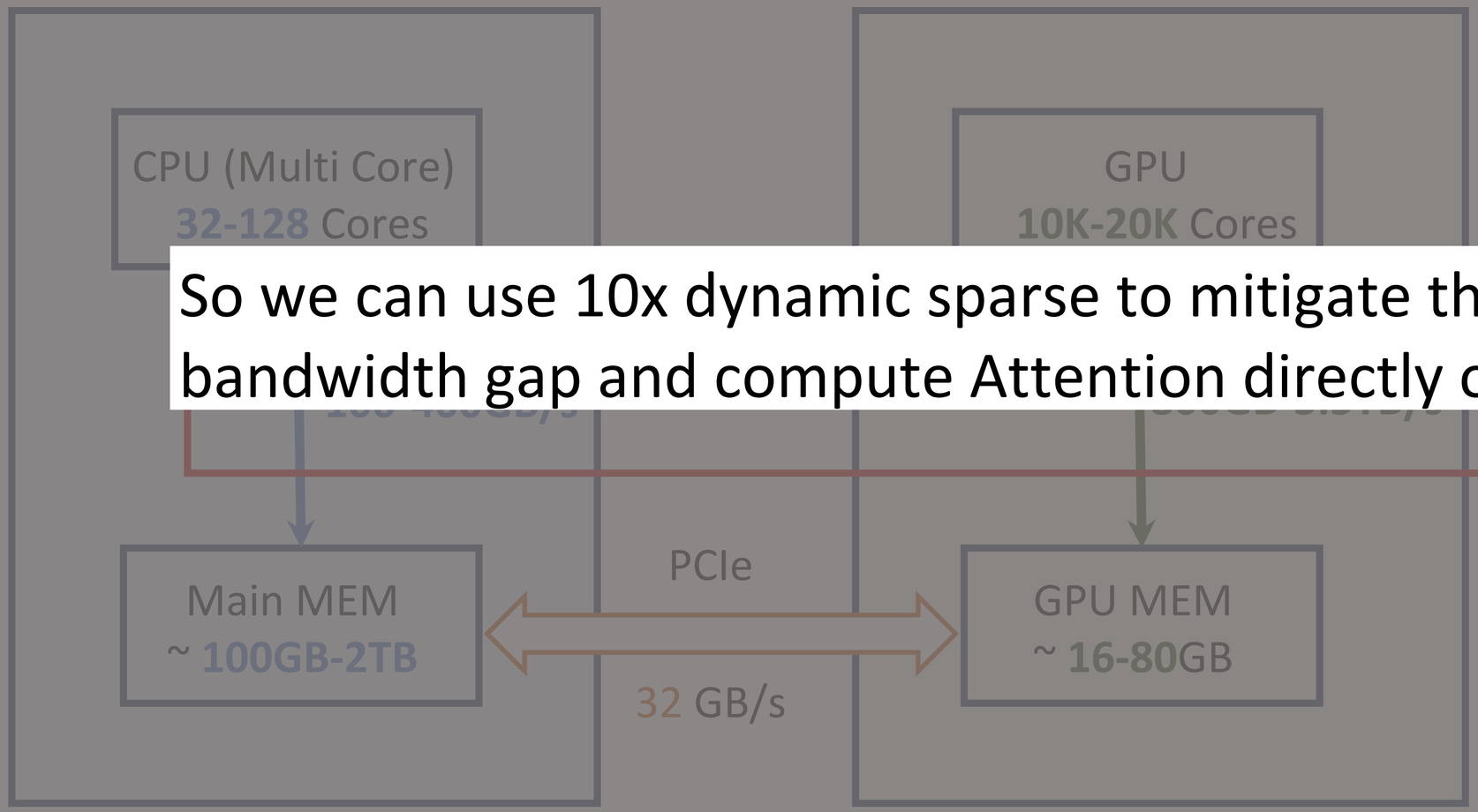
# Key Observation: CPU's Mem Bandwidth is Only 10x Slower!




Attention is always memory bound, so CPU's 100x worse flops does not matter!

Deepspeed, FastDecode, InfiniGen, InfiniLLM AttentionStore, Mooncake

# Key Observation: CPU's Mem Bandwidth is Only 10x Slower!

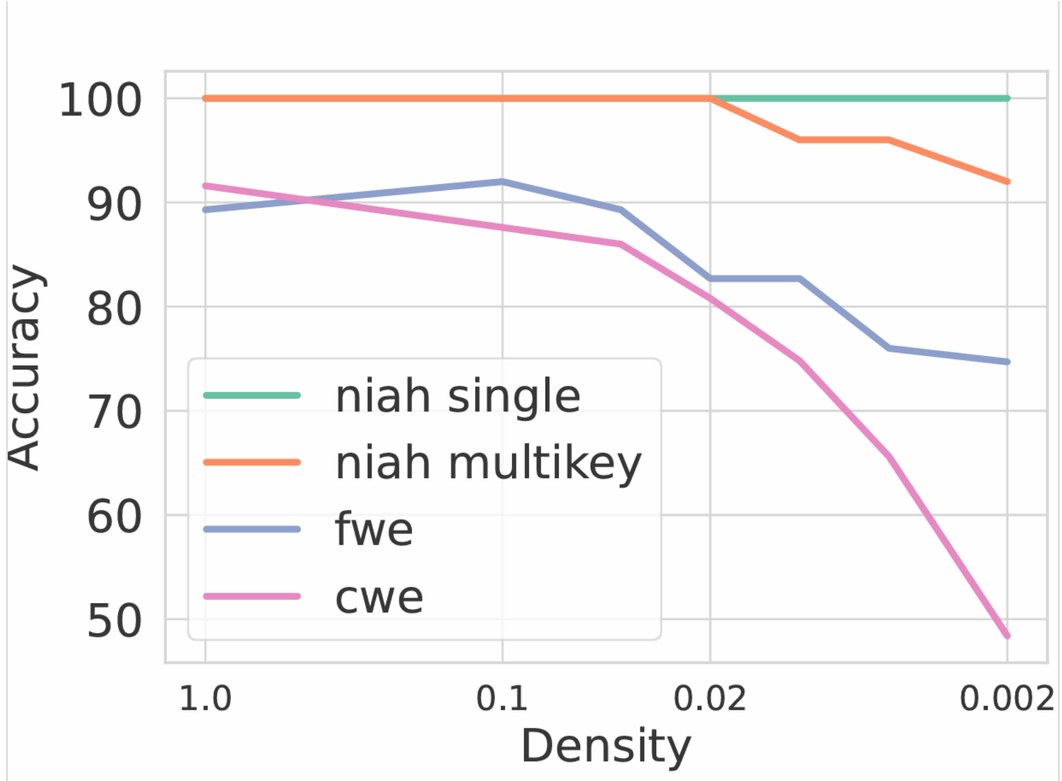



So we can use 10x dynamic sparse to mitigate the GPU CPU bandwidth gap and compute Attention directly on CPU.

Attention is always memory bound, so CPU's 100x worse flops does not matter!

Deepspeed, FastDecode, InfiniGen, InfiniLLM AttentionStore, Mooncake

# Challenge: TopK also fails!




RULER Benchmark

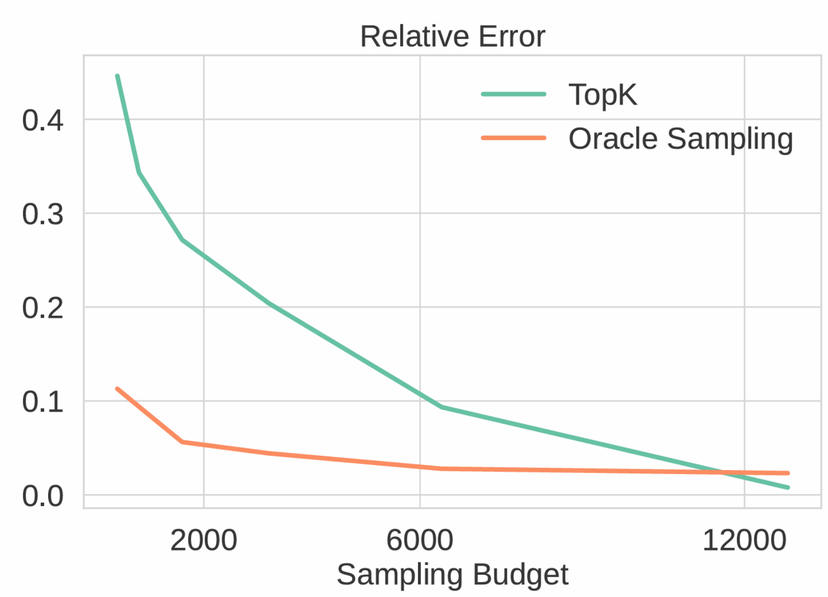
| Methods | Config | 2-Ops | 4-Ops | 5-Ops |
|---|---|---|---|---|
| *Llama-3.1-8B-Instruct* | Full | 87.4 | 71.4 | 26.8 |
| MagicPIG | (10,300) | 83.1 | 67.2 | 20.7 |
| MagicPIG | (10,220) | 79.8 | 58.9 | 17.9 |
| MagicPIG | (10,150) | 68.3 | 43.5 | 11.7 |
| TopK | 0.06 | 78.6 | 62.9 | 20.8 |
| TopK | 0.04 | 76.2 | 59.0 | 19.2 |
| TopK | 0.02 | 71.5 | 44.0 | 11.3 |

Reasoning Benchmark

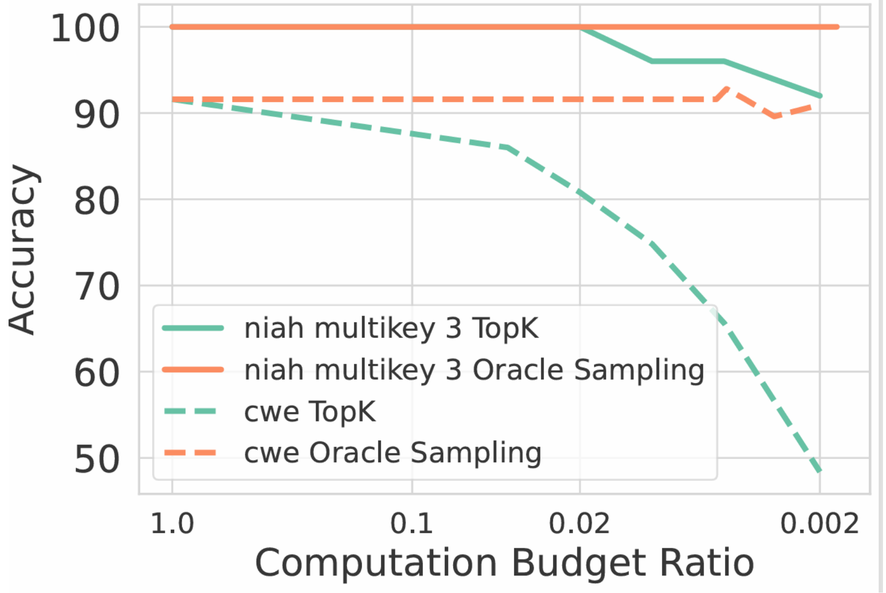
# We need to go beyond TopK attention!

**Definition 3.1** (Oracle Sampling Estimation)**.** Given a sampling budget $\mathcal{B}$ and normalized attention score $w$, $\mathcal{B}$ elements are sampled independently from $w$ (i.e. $i_1, i_2, ..., i_{\mathcal{B}} \overset{\text{iid}}{\sim} w$). Then the attention output is estimated as

$$\bar{o} = \frac{1}{\mathcal{B}} \sum_{j=1}^{\mathcal{B}} v_{i_j} \tag{4}$$



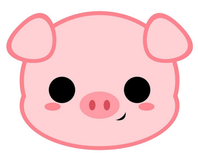

Estimation Error

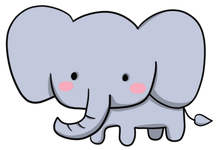



Downstream Tasks

# We need to go beyond TopK attention!

## Zoo


X 10


X 10


X 10

Other 70 animals

… X 1


50 lbs


20 lbs


10 lbs


1 lb

What is avg weight of the food every animal eats in this zoo?

$$\frac{50\times10 + 20\times10 + 10\times10 + 1\times80}{100} = 8.7\ lbs$$

If estimation budget is 10,

**Topk:**

- K=10 selects   

and other 7 animals and use them to estimate avg:

$$\frac{50\times10 + 20\times10 + 10\times10 + 1\times7}{37} = 22\ lbs$$

**Sampling:**

- Sample with replacement from:

$[0.1, 0.1, 0.1, 0.01\times70]$

10 times. Example trial:

$[elephant, pig, tiger, others\times7]$

Estimated avg:

$$\frac{50 + 20 + 10 + 1\times7}{10} = 8.7\ lbs$$

# We need to go beyond TopK attention!

## Zoo




Sampling! Not only Search!

What is avg weight of the food every animal eats in this zoo?

$$\frac{50\times10 + 20\times10 + 10\times10 + 1\times80}{100} = 8.7\ lbs$$

If estimation budget is 10,

**Topk:**

estimate avg:

$$\frac{50\times10 + 20\times10 + 10\times10 + 1\times7}{37} = 22\ lbs$$

**Sampling:**

- Sample with replacement from:

$[0.1, 0.1, 0.1, 0.01\times70]$

10 times. Example trial:

$[elephant, pig, tiger, others\times7]$

Estimated avg:

$$\frac{50 + 20 + 10 + 1\times7}{10} = 8.7\ lbs$$

# Locality-Sensitive Hashing (late 90s)


- Randomly split hyperplane
- **Observation**: query and its far-away point ( ) are more likely to split than query and its neighbor ( )

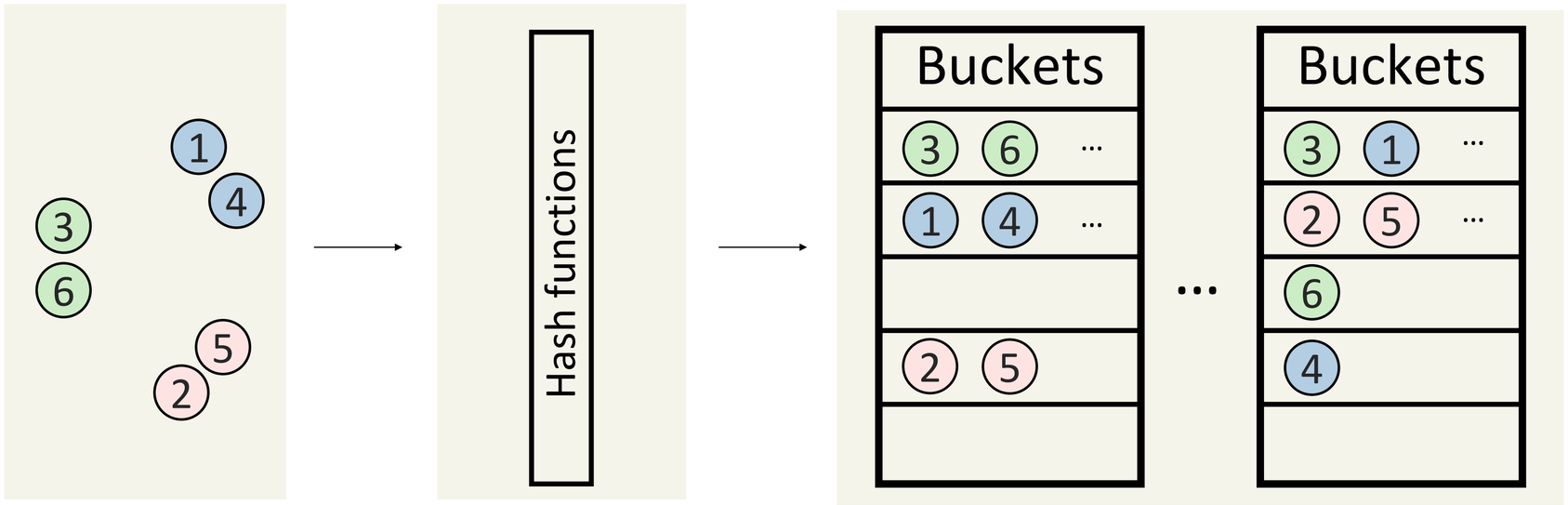
**Likely** to separate **far** points; **Unlikely** to separate **near** points

$$Pr(h(x) = h(y)) = 1 - \frac{1}{\pi}\cos^{-1}(\theta)$$

(Broder SEQUENCES '97, Indyk-Motwani STOC '98, Charikar STOC '02)

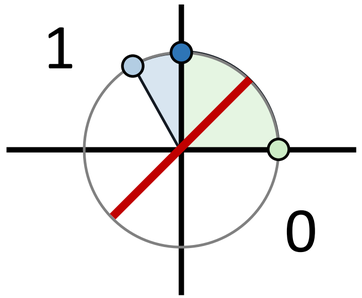
# LSH for Near-Neighbor Search

Preprocessing



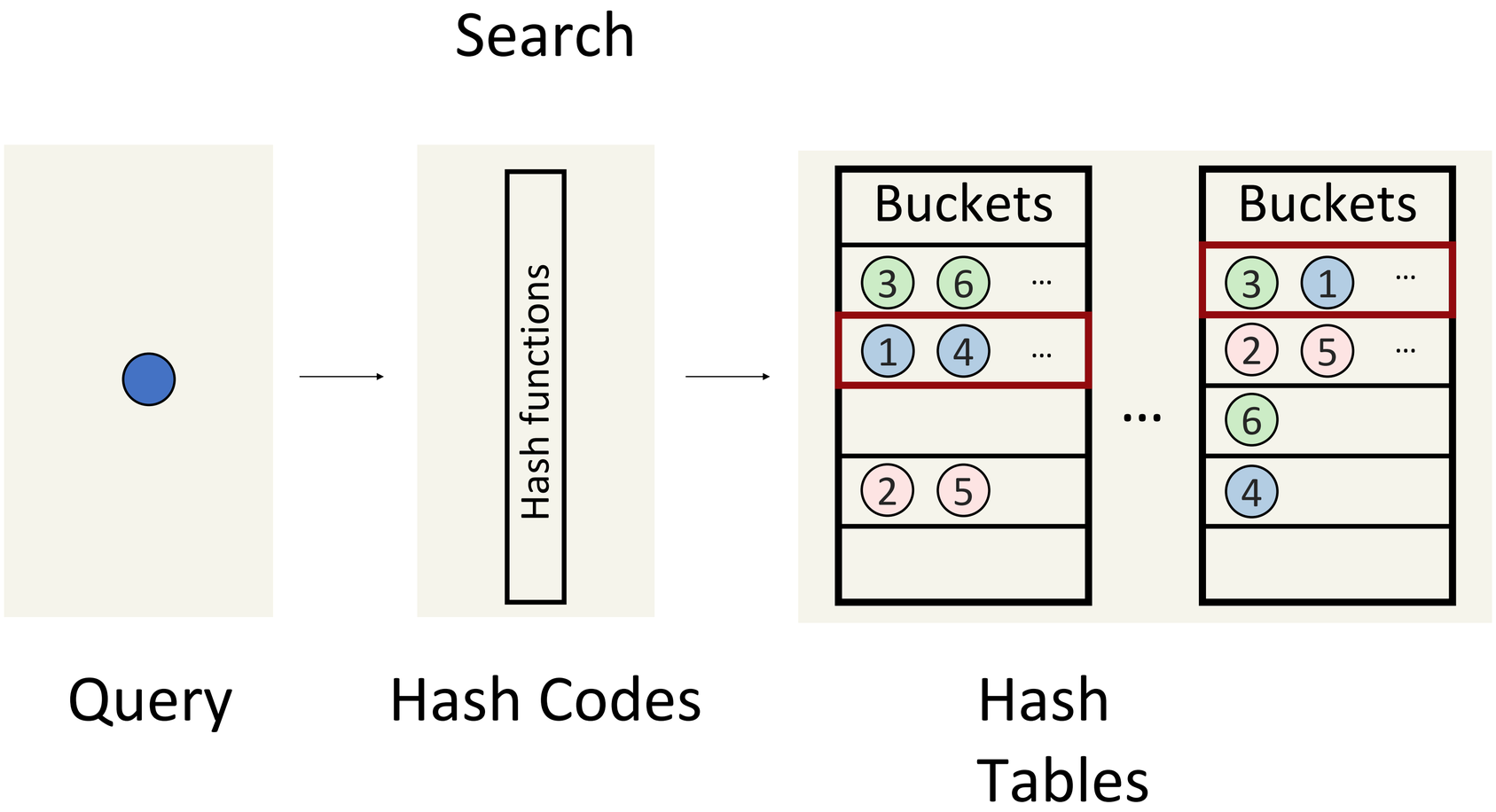

Data

Hash Codes



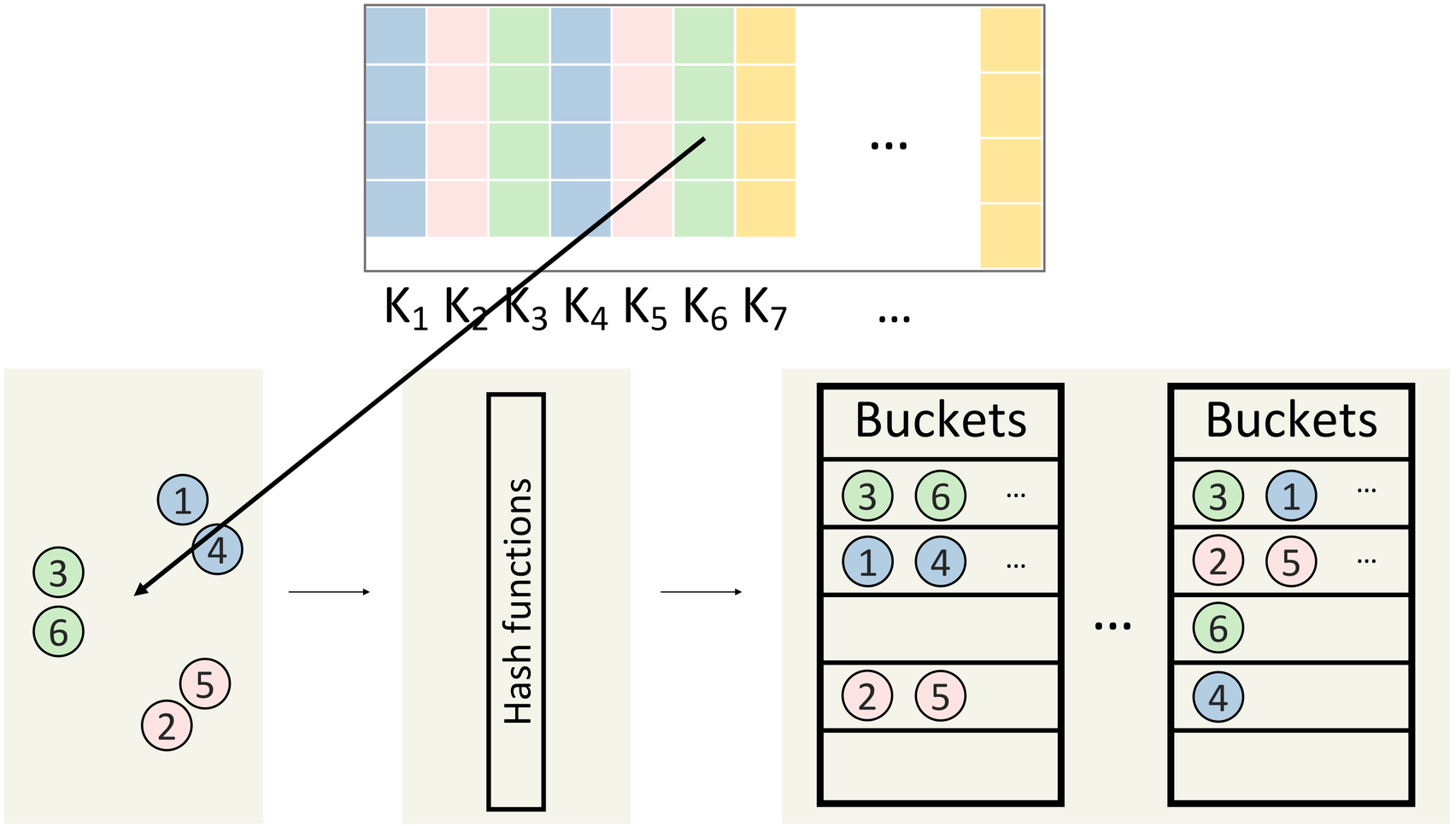

Hash Tables

# LSH for Near-Neighbor Search

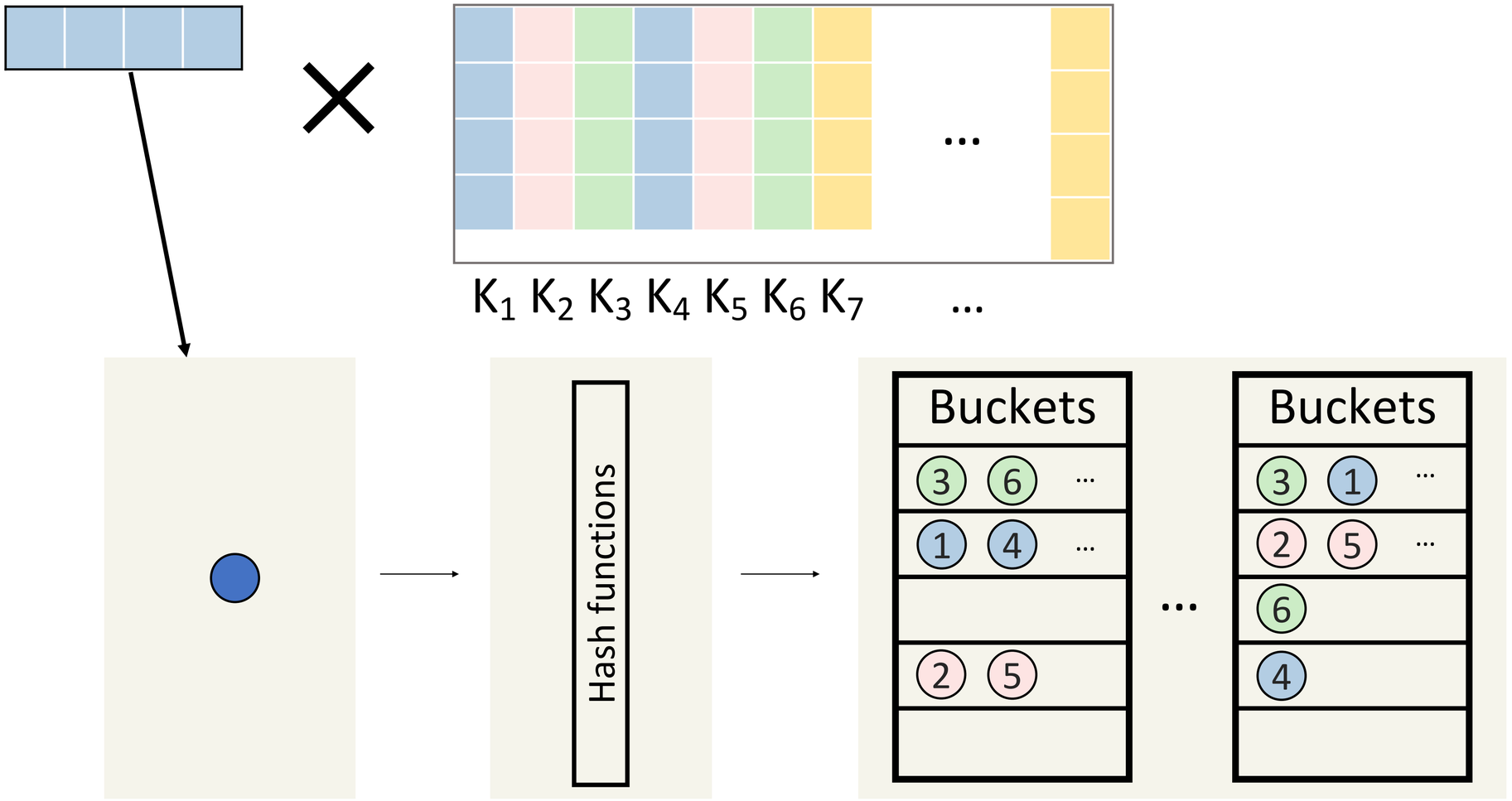



(Shrivastava-Li *NeurIPS'14,* Neyshabur-Srebro *ICML'15* )

# LSH for Approx. Matmul



# LSH for Approx. Matmul



# LSH as Samplers

**Challenge**: LSH need many hash functions and tables in practice!

Theory

- Super-linear $O(N^{1+\rho})$ memory
- Sub-linear query time, $O(N^{\rho})$
- $\rho < 1$ but generally large (close to 1) and often hard to determine

Practical Issues

- Needs lot of hash tables and distance computations for good accuracy on near-neighbors
- Buckets can be quite heavy. Poor randomness, or unfavorable data distributions

Intuition: LSH Sampling is more robust and efficient than search!

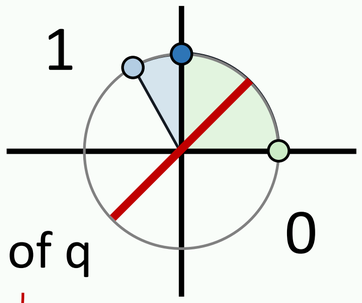
# MagicPIG: SParse Inference EnGine

## LSH Retriever

## Attention Server

GPU

Hash Functions




Model

q

① 

Static KV

Static KV

Static KV

② Hash code of q

④ Aggregated attention

CPU

Hash Table Slots

Hash Table Slots

Hash Table Slots

Hash Table Slots

③KV Indices

KV Cache

KV Cache

KV Cache

KV Cache

**MagicPIG**:sParse Inference enGine for LLM. Zhuoming Chen, Ranajoy Sadhukhan, Zihao Ye, Jianyu Zhang, Matthijs Douze, Leon Bottou, Zhihao Jia, Beidi Chen

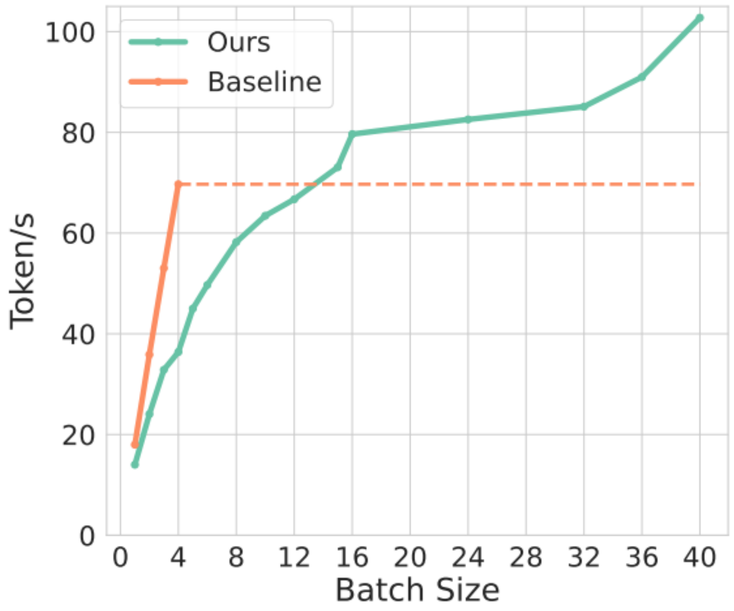
# MagicPIG on RULER Benchmark

| Methods | Config | 16K | 32K | 64K | 96K | Avg. | $\text{Cost}_1$ | $\text{Cost}_2$ | $\text{Cost}_{total}$. |
|---|---|---|---|---|---|---|---|---|---|
| *Llama-3.1-8B-Instruct* | Full | 94.2 | 91.5 | 86.1 | 83.0 | 88.7 | 0.00 | 1.00 | 1.00 |
| MagicPIG | (10,150) | 91.8 | 88.9 | 84.8 | 80.0 | 86.4 | 0.00 | 0.02 | 0.02 |
| MagicPIG | (9,120) | 93.4 | 90.6 | 84.7 | 81.5 | **87.6** | 0.00 | 0.04 | 0.04 |
| MagicPIG | (8,75) | 92.9 | 90.2 | 84.9 | 81.7 | 87.4 | 0.00 | 0.05 | 0.05 |
| Quest | (16,0.04) | 86.3 | 85.4 | 81.9 | 74.9 | 82.1 | 0.06 | 0.04 | 0.10 |
| Quest | (32,0.06) | 84.3 | 84.0 | 80.1 | 74.4 | 80.7 | 0.03 | 0.06 | 0.09 |
| Quest | (64,0.08) | 85.2 | 84.3 | 77.0 | 74.2 | 80.2 | 0.02 | 0.08 | 0.10 |
| *MegaBeam-Mistral-7B-512K* | Full | 91.7 | 88.1 | 83.5 | 83.7 | 86.8 | 0.00 | 1.00 | 1.00 |
| MagicPIG | (10,150) | 89.8 | 86.5 | 81.7 | 80.7 | 84.7 | 0.00 | 0.02 | 0.02 |
| MagicPIG | (9,120) | 90.7 | 88.5 | 82.9 | 82.4 | **86.1** | 0.00 | 0.04 | 0.04 |
| MagicPIG | (8,75) | 90.6 | 86.4 | 82.8 | 81.6 | 85.4 | 0.00 | 0.05 | 0.05 |
| Quest | (16,0.04) | 83.3 | 83.2 | 79.3 | 78.6 | 81.1 | 0.06 | 0.04 | 0.10 |
| Quest | (32,0.06) | 81.5 | 80.8 | 76.7 | 74.4 | 78.4 | 0.03 | 0.06 | 0.09 |
| Quest | (64,0.08) | 79.6 | 77.5 | 73.8 | 73.7 | 76.1 | 0.02 | 0.08 | 0.10 |

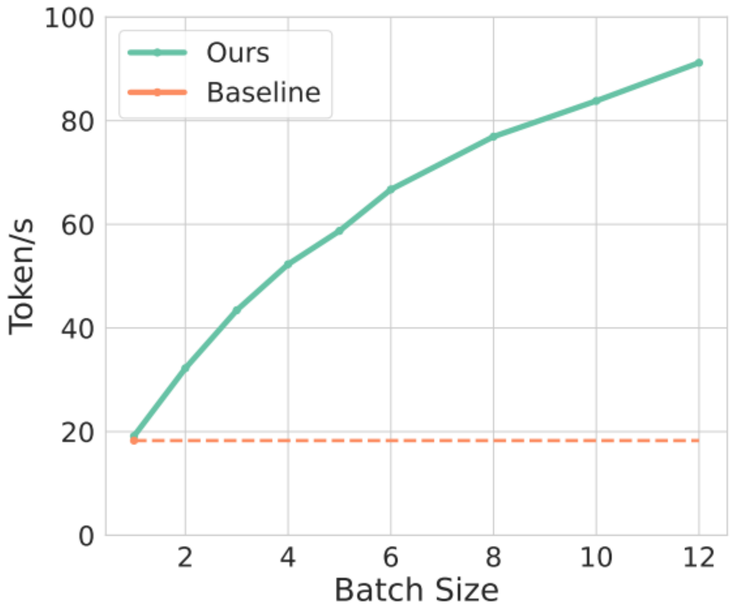
# MagicPIG on Reasoning Benchmark

| Methods | Config | 2-Ops | 4-Ops | 5-Ops | $\text{Cost}_1$ | $\text{Cost}_2$ | $\text{Cost}_{total}$. |
|---|---|---|---|---|---|---|---|
| *Llama-3.1-8B-Instruct* | Full | 87.4 | 71.4 | 26.8 | 0.00 | 1.00 | 1.00 |
| MagicPIG | (10,300) | 83.1 | 67.2 | 20.7 | 0.00 | 0.06 | 0.06 |
| MagicPIG | (10,220) | 79.8 | 58.9 | 17.9 | 0.00 | 0.04 | 0.04 |
| MagicPIG | (10,150) | 68.3 | 43.5 | 11.7 | 0.00 | 0.02 | 0.02 |
| TopK | 0.06 | 78.6 | 62.9 | 20.8 | 0.50 | 0.06 | 0.56 |
| TopK | 0.04 | 76.2 | 59.0 | 19.2 | 0.50 | 0.04 | 0.54 |
| TopK | 0.02 | 71.5 | 44.0 | 11.3 | 0.50 | 0.02 | 0.52 |
| Oracle Sampling | 0.3 | 88.1 | 72.4 | 27.6 | 0.50 | 0.02 | 0.52 |
| Oracle Sampling | 0.1 | 88.5 | 69.2 | 26.2 | 0.50 | 0.01 | 0.51 |
| Oracle Sampling | 0.02 | 83.1 | 57.9 | 11.9 | 0.50 | 0.005 | 0.505 |
| Quest | (16,0.06) | 55.8 | 23.2 | 5.2 | 0.06 | 0.06 | 0.12 |

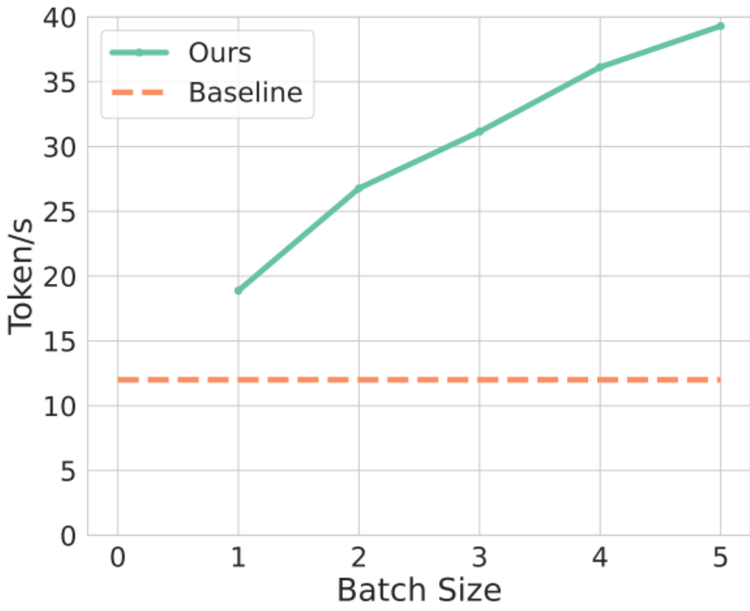
**Beat exact TopK attention!**




(a) A100 with 34B model




(b) L20 with 13B model



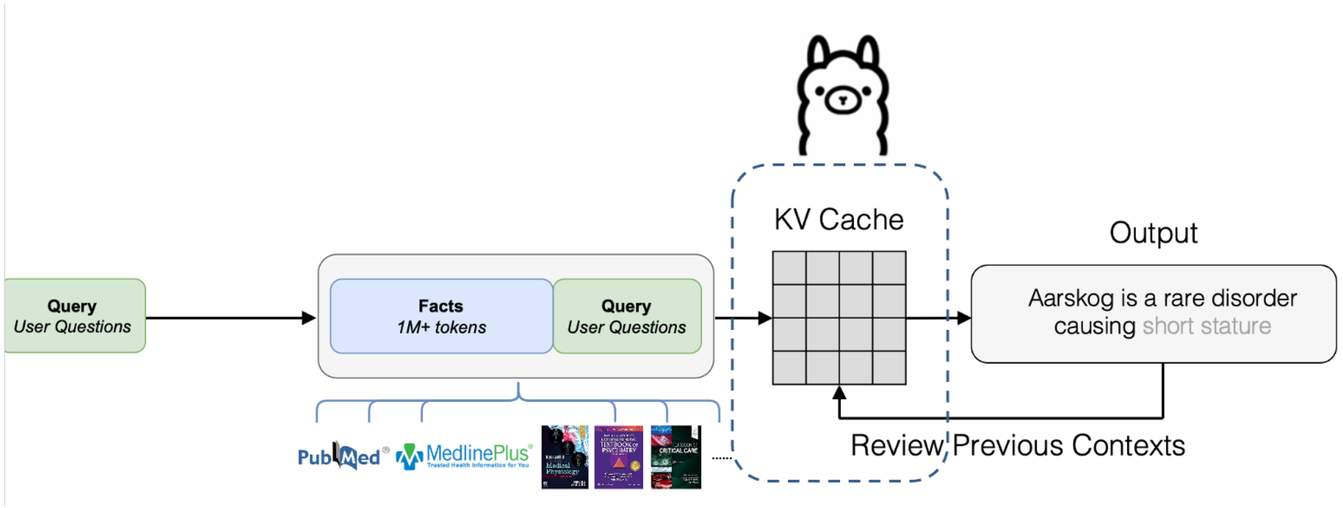

(c) RTX 4090 with 8B model

1.5-5X throughput improvement!

| | | | | | | |
|---|---|---|---|---|---|---|
| Oracle Sampling | 0.1 | 88.5 | 69.2 | 26.2 | 0.50 | 0.01 | 0.51 |
| Oracle Sampling | 0.02 | 83.1 | 57.9 | 11.9 | 0.50 | 0.005 | 0.505 |
| Quest | (16,0.06) | 55.8 | 23.2 | 5.2 | 0.06 | 0.06 | 0.12 |

# Takeaway

- We should also take CPU into consideration for system design
- Sometimes exact TopK is worse than sampling the TopK
- LSH is an effective and efficient sampling strategy

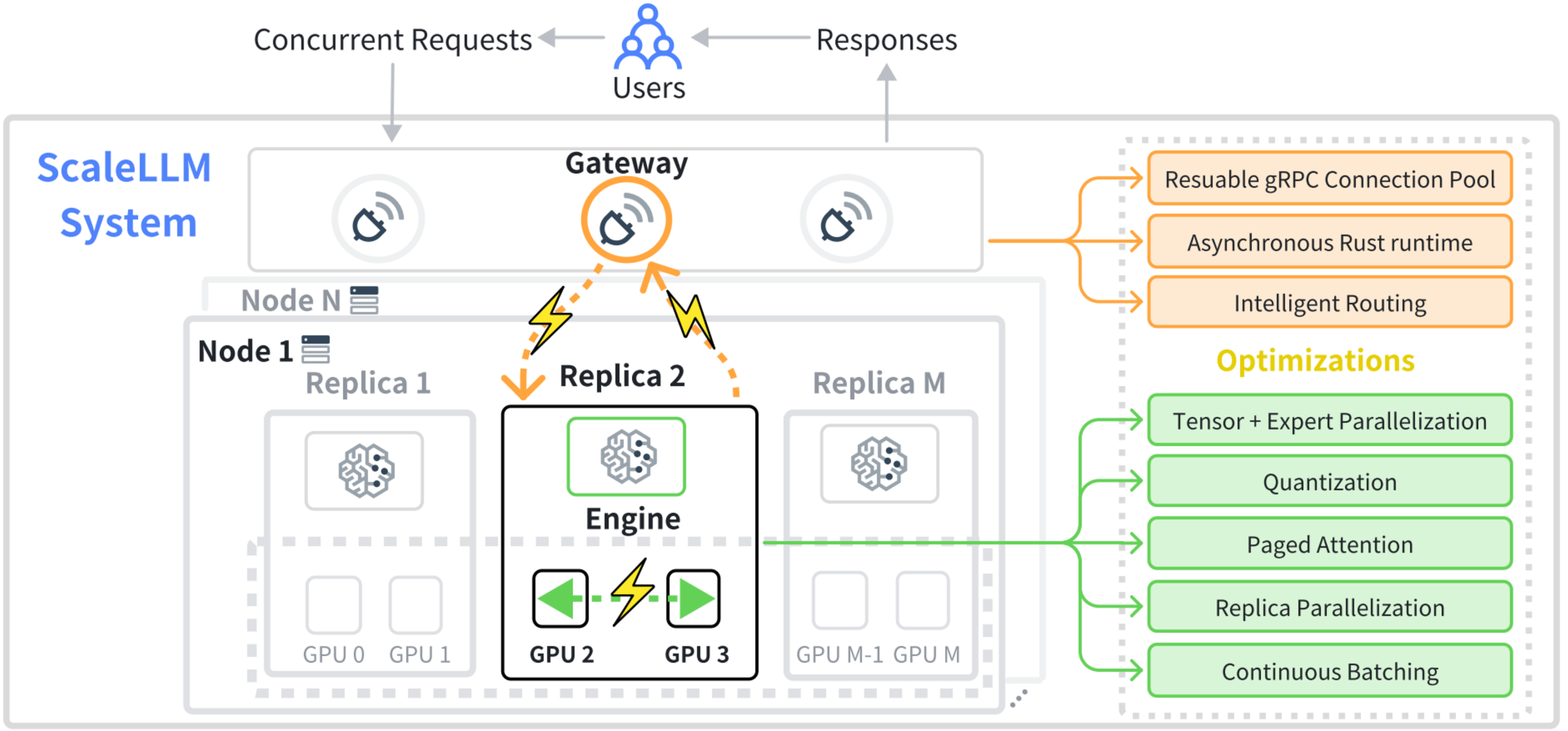


# Insights & Future

# Insights

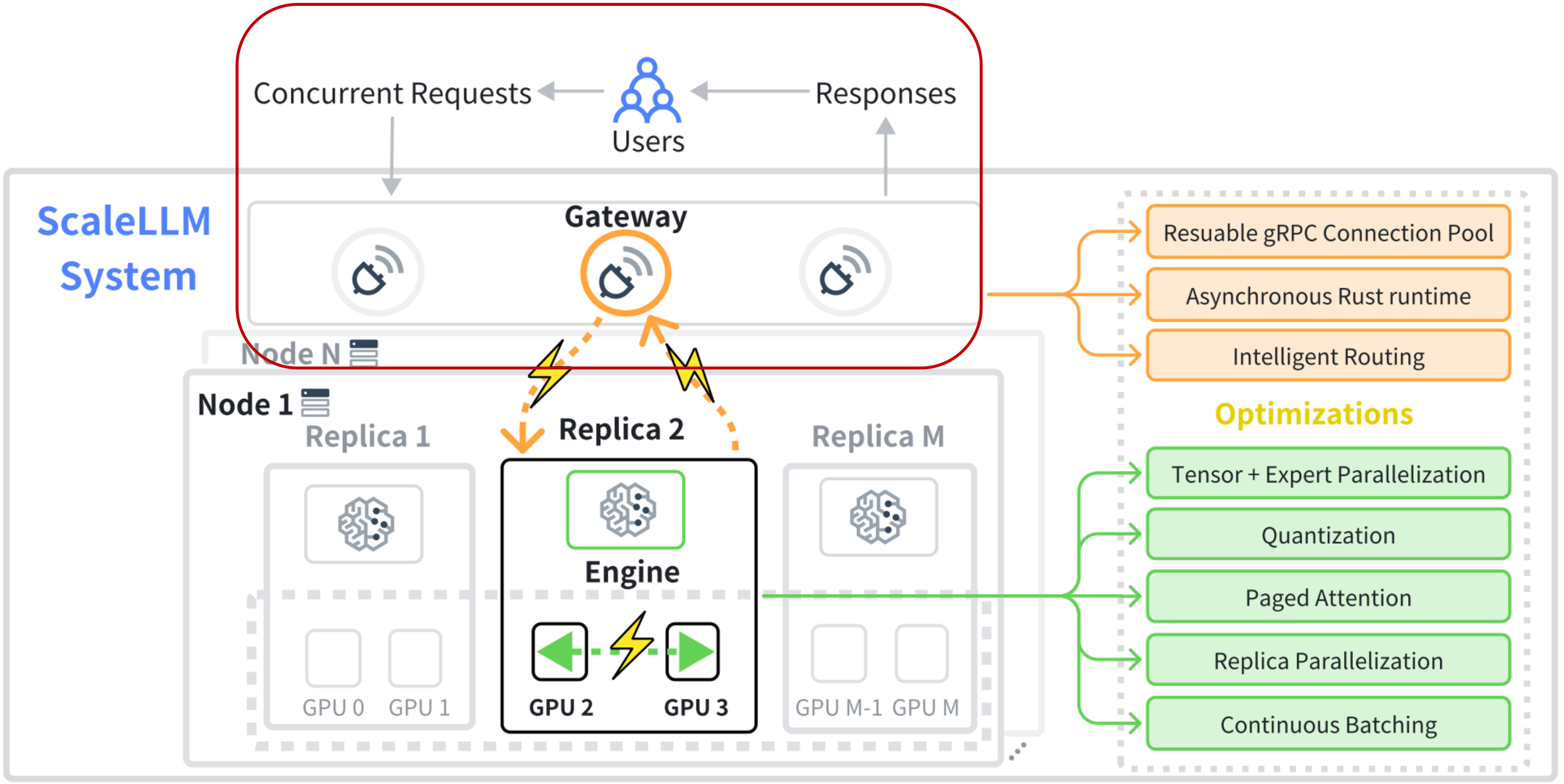
- Full attention, and thus KV Cache, is fundamental
- Full attention is necessary, but not at every layer
- Compressing tokens directly at/before prefill is hard
- Maybe we should design efficient KV cache at pre-training

# LLM Inference to LLM Endpoint



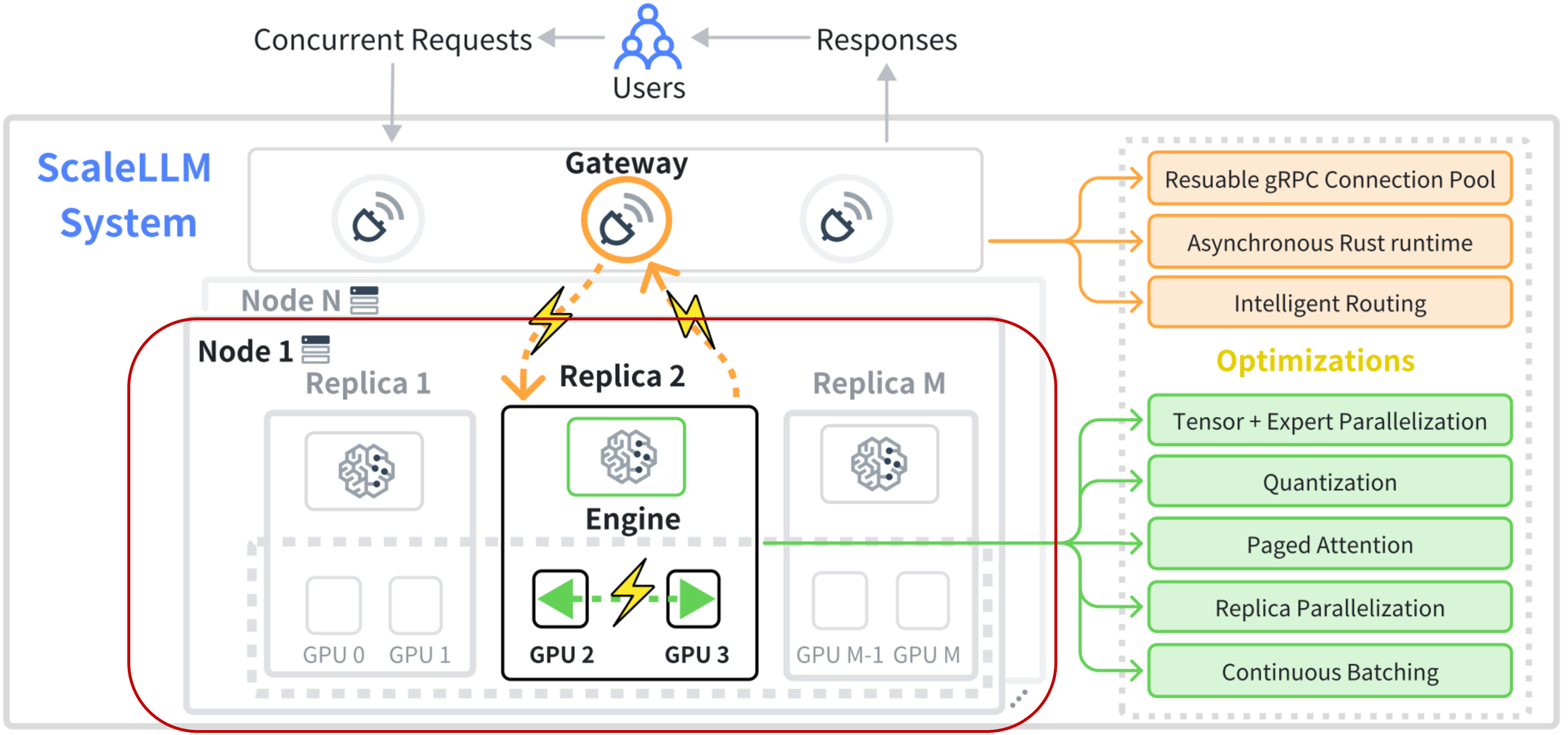

ScaleLLM: A Resource-Frugal LLM Serving Framework by Optimizing End-to-End Efficiency

# Future: Efficient Gateway



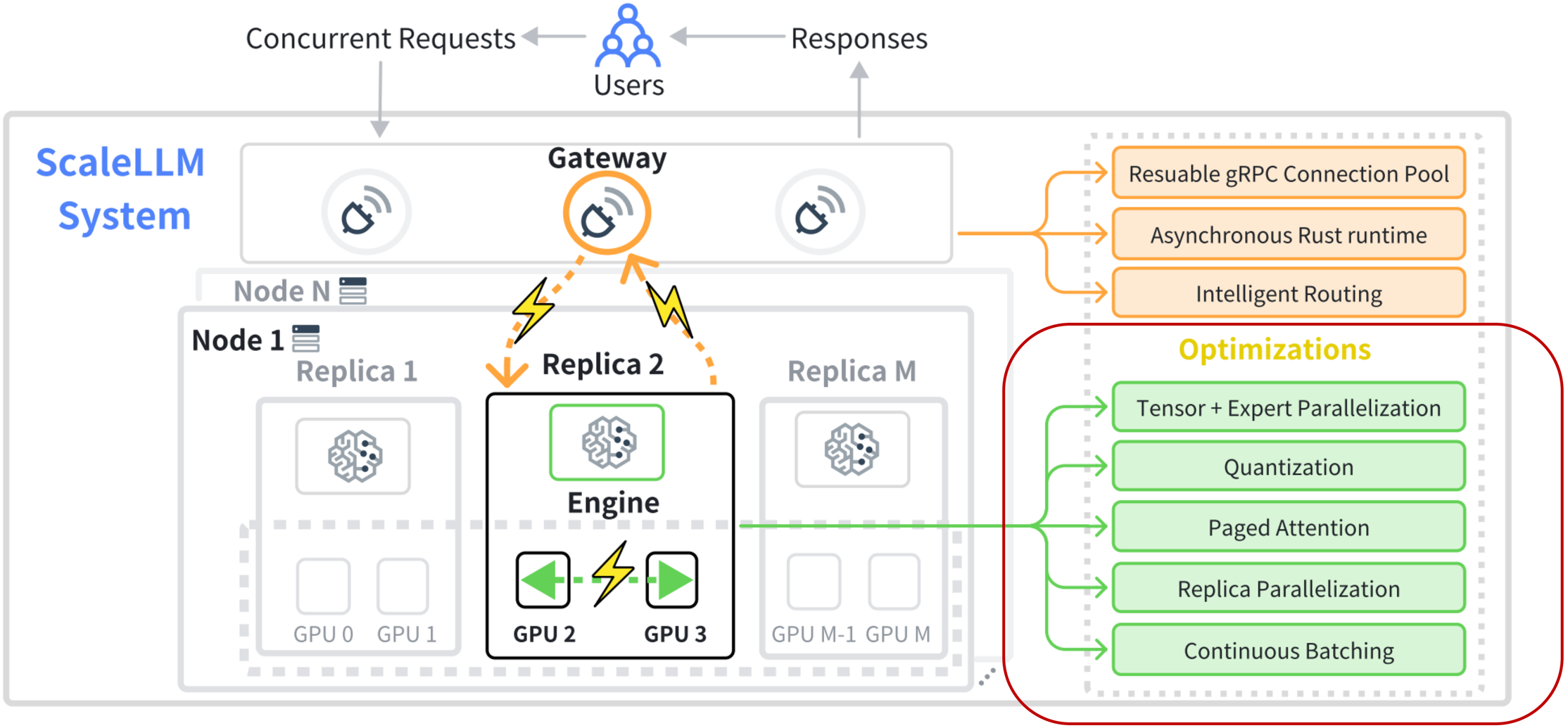

ScaleLLM: A Resource-Frugal LLM Serving Framework by Optimizing End-to-End Efficiency

# Future: Autoscaling of Replica




ScaleLLM: A Resource-Frugal LLM Serving Framework by Optimizing End-to-End Efficiency

# Future: Integration of Multiple Techniques




ScaleLLM: A Resource-Frugal LLM Serving Framework by Optimizing End-to-End Efficiency

# Thank you!
# Q&A