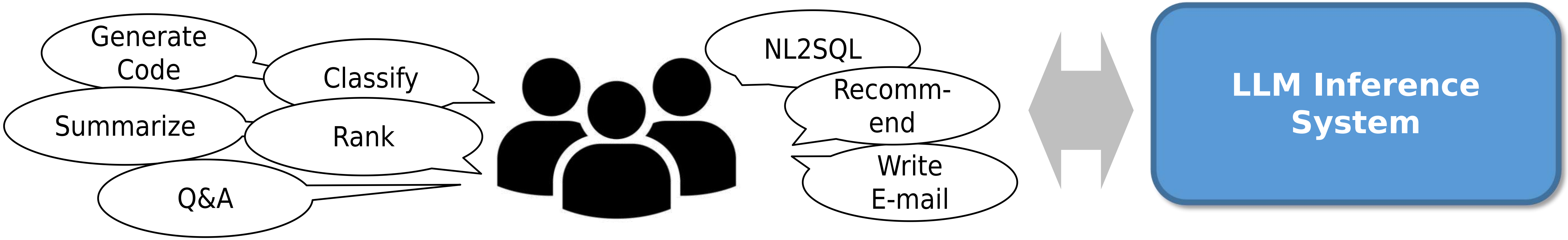
Database Perspective on LLM Inference Systems

James Pan, Guoliang Li

Department of Computer Science, Tsinghua University

LLMs: General Computing Interface

- ***Widespread LLM adoption leads to High-Volume, High-Velocity, & High-Variety inference workloads***


Generate Code
Classify
Summarize
Rank
Q&A
NL2SQL
Recomm-end
Write E-mail
LLM Inference System

LLM-Powered Applications

Information Retrieval

- Question & Answering
  - Customer Support
  - Role-based, e.g. Travel Agent
  - Translation
- Recommendation

Data Analytics

- Spam detection
- Attribute extraction
- Classification
- Ranking
- Summarization

Content Creation

- Code generation
  - NL2SQL
- Document/text generation
  - Emails, reports, etc.

Large Language Models

LLM Inference Systems

- ***Widespread LLM adoption leads to High-Volume, High-Velocity, & High-Variety inference workloads***

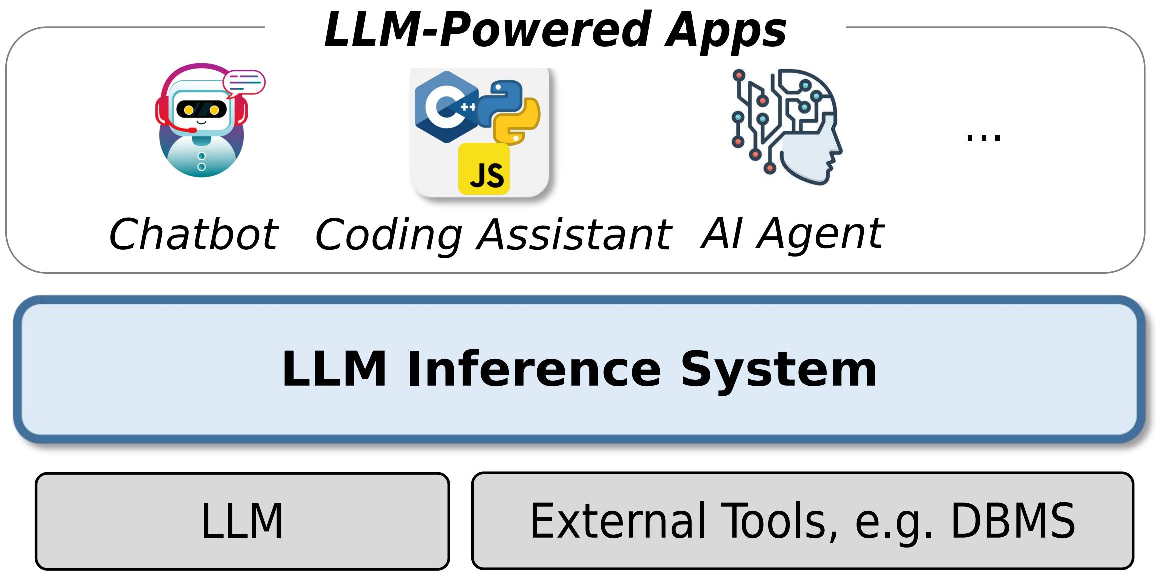
Goal: Build a system for High-Performance and High-Quality inference

High Performance

- Low latency, i.e. time-to-first-token (**TTFT**), time-between-tokens (**TBT, TPOT**), end-to-end lat.
- High throughput, i.e. **requests/sec**, **tokens/sec**

High Quality

- E.g. **correctness** (NL2SQL, Q&A, code gen), **relevance** (recommendation, customer support), **accuracy** (classification, ranking), etc.



LLM Inference Systems: Key Challenges

- ***Widespread LLM adoption leads to High-Volume, High-Velocity, & High-Variety inference workloads***

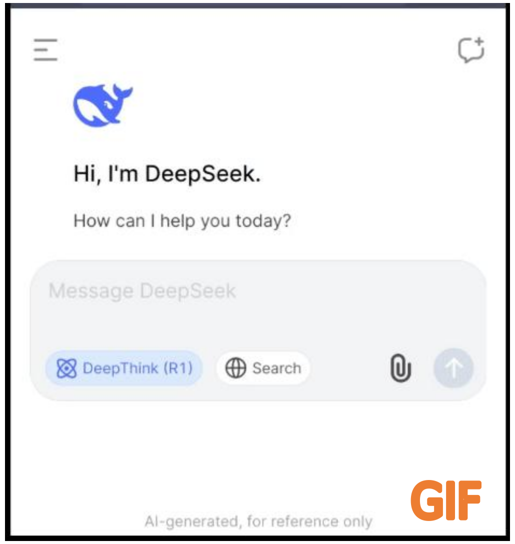
Goal: Build a system for High-Performance and High-Quality inference

😈

1) LLM Uncertainty Principle: Can't know what you'll get until you run it

2) Autoregressive Generation: Output generated one token at a time

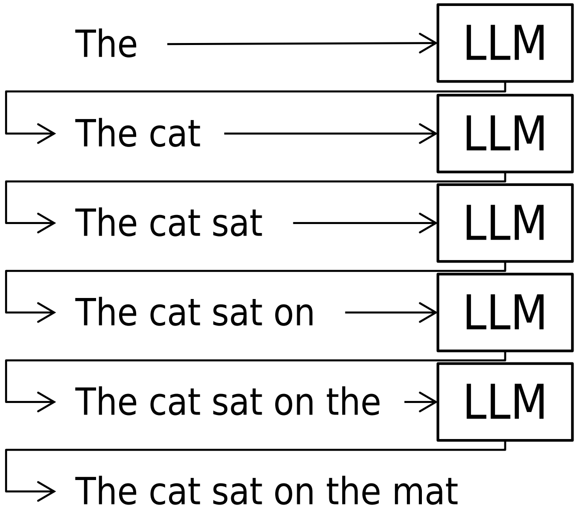
Latency | **Throughput** | **Memory** | **Quality**



Hi, I'm DeepSeek.

How can I help you today?

(a) DeepSeek-R1 picking a random number



The → LLM

The cat → LLM

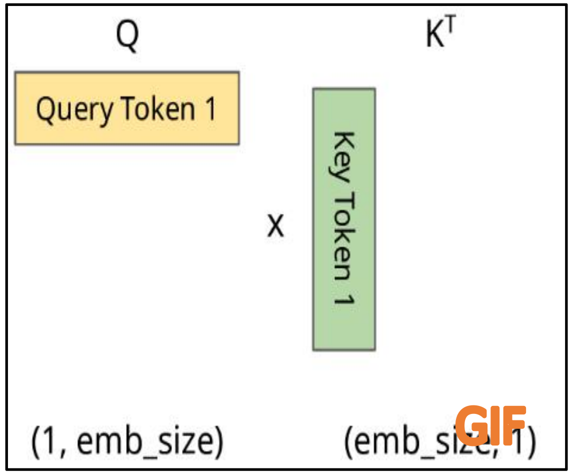
The cat sat → LLM

The cat sat on → LLM

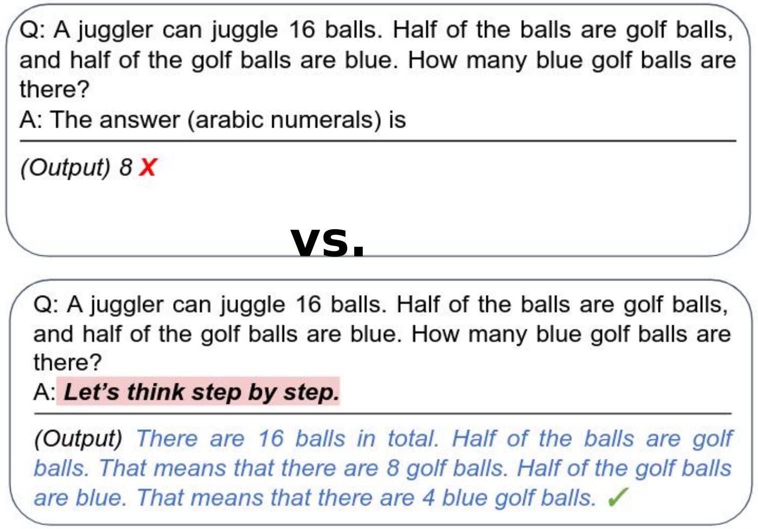
The cat sat on the → LLM

The cat sat on the mat

(b) Autoregressive Generation



(c) KV cache growth



Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: The answer (arabic numerals) is

(Output) 8 ✗

vs.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: ***Let's think step by step.***

(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓

(d) Output sensitivity to small changes in prompt [Kojima '23]

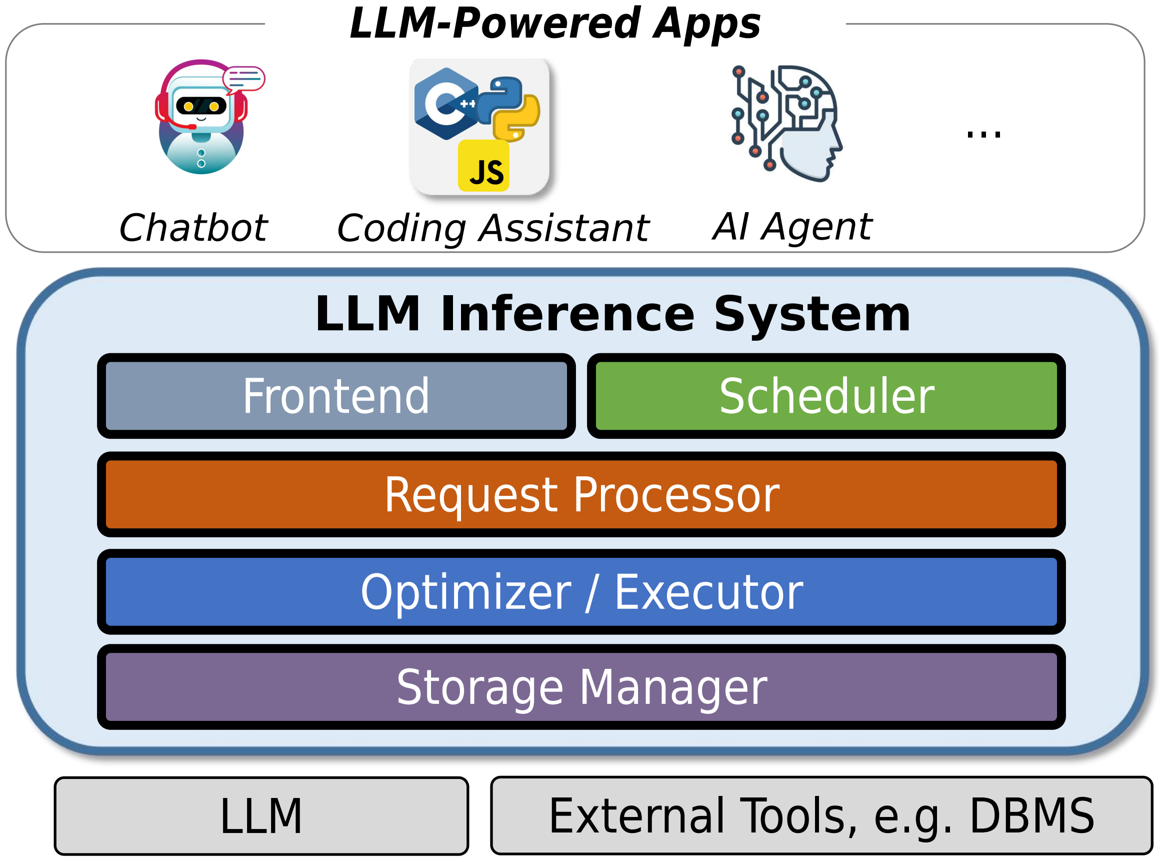
LLM Inference Systems: Architecture

- ***Widespread LLM adoption leads to High-Volume, High-Velocity, & High-Variety inference workloads***

Goal: Build a system for High-Performance and High-Quality inference

1) LLM Uncertainty Principle: Can't know what you'll get until you run it

2) Autoregressive Generation: Output generated one token at a time



LLM-Powered Apps

Chatbot *Coding Assistant* *AI Agent* …

LLM Inference System

Frontend | Scheduler

Request Processor

Optimizer / Executor

Storage Manager

LLM | External Tools, e.g. DBMS

Latency
- Fast, Available

Throughput
- Scalable

Memory
- Memory Efficient, Elastic Resources

Quality
- Correct, Accurate, Relevant, Trustworthy, Secure

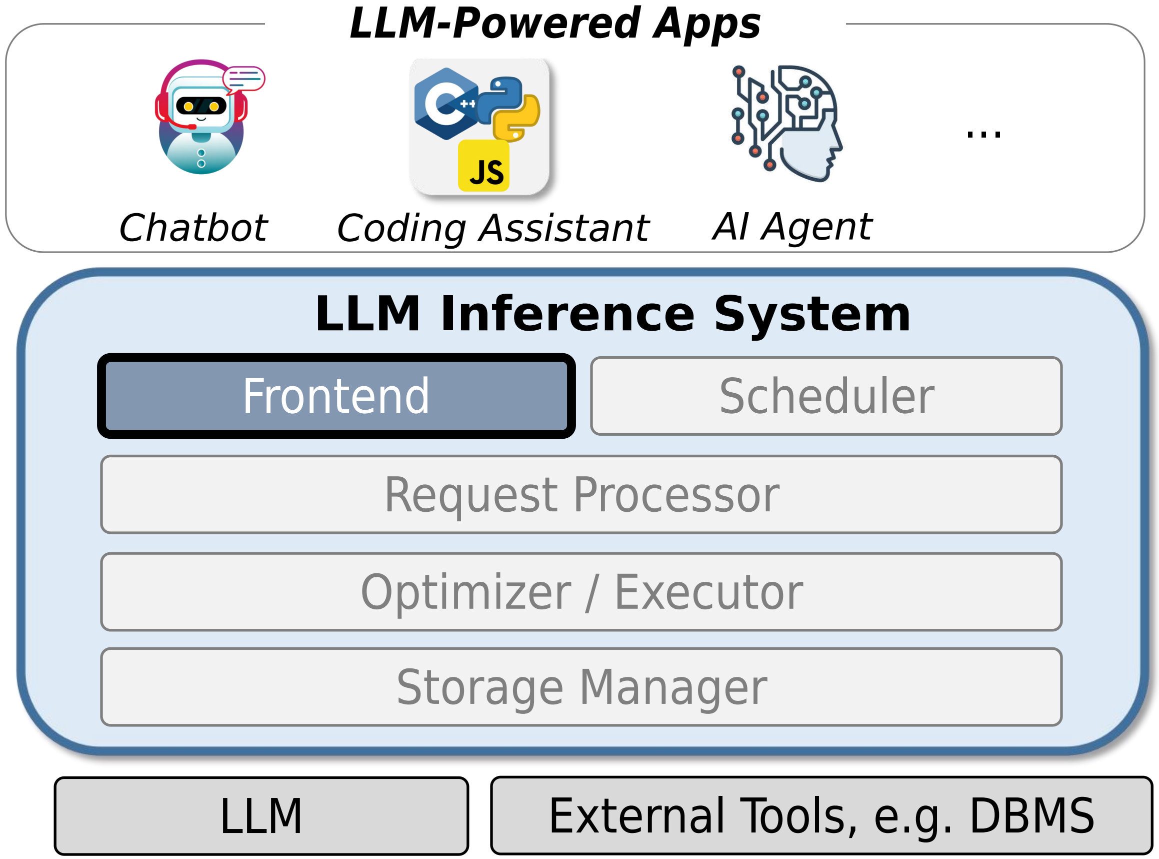
LLM Inference Systems: Frontend

- ***Widespread LLM adoption leads to High-Volume, High-Velocity, & High-Variety inference workloads***

Goal: Build a system for High-Performance and High-Quality inference

1) LLM Uncertainty Principle: Can't know what you'll get until you run it

2) Autoregressive Generation: Output generated one token at a time



User Interface
- Declarative Modules
- Language Extensions

- *Parse user requests into **effective prompt workflow***

I/O Interpreter
- Prompt Generator
- Constraint Checker

- *Build **optimized prompts**, e.g. prompt engineering*

Seq. Generation
- Streaming Generation
- Structured Generation

- *Coordinate seq. gen. to **balance quality and performance***

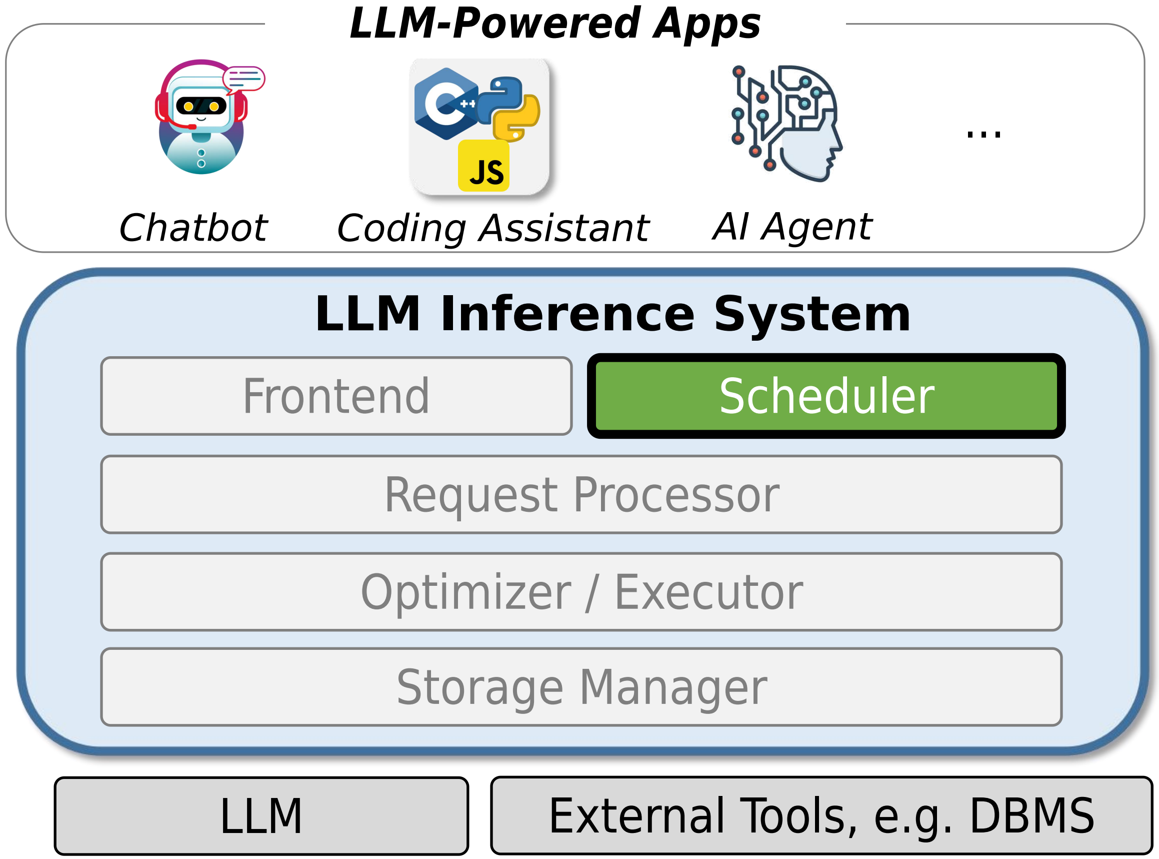
LLM Inference Systems: Scheduler

- ***Widespread LLM adoption leads to High-Volume, High-Velocity, & High-Variety inference workloads***

Goal: Build a system for High-Performance and High-Quality inference

1) LLM Uncertainty Principle: Can't know what you'll get until you run it

2) Autoregressive Generation: Output generated one token at a time



Load Balancer
- Job Assignment Module
- Load Prediction Model

- *Assign requests to workers to **maximize utilization***

Scheduler
- Job Prioritizer
- Job Cost Model

- *Prioritize jobs to **minimize queuing delays***

Batch Controller
- Chunking Module
- Batch Size Control

- *Compose batches to **balance TTFT & TBT with throughput***

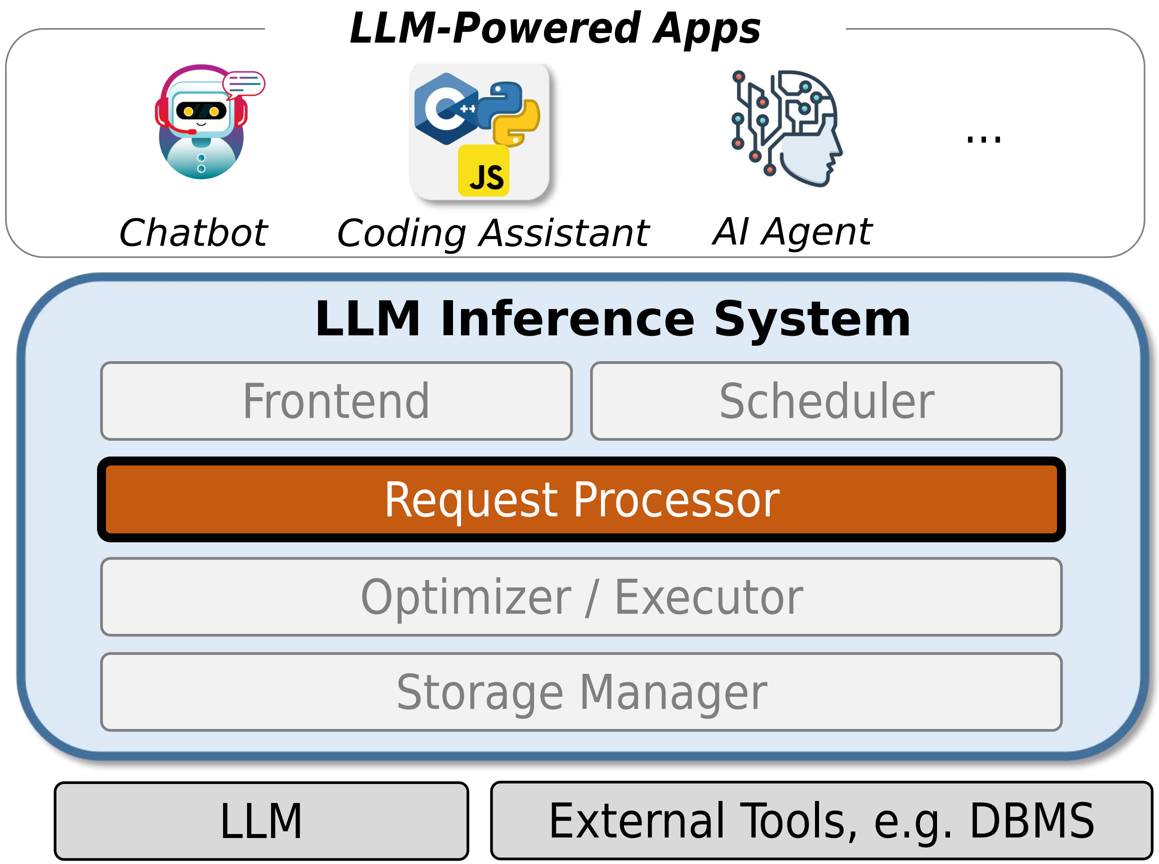
LLM Inference Systems: Req. Proc.

- ***Widespread LLM adoption leads to High-Volume, High-Velocity, & High-Variety inference workloads***

Goal: Build a system for High-Performance and High-Quality inference

<u>1) LLM Uncertainty Principle</u>: Can't know what you'll get until you run it

<u>2) Autoregressive Generation:</u> Output generated one token at a time



LLM-Powered Apps

Chatbot *Coding Assistant* *AI Agent* …

LLM Inference System

Frontend | Scheduler

Request Processor

Optimizer / Executor

Storage Manager

LLM | External Tools, e.g. DBMS

Inference Workflow

- Prefill
- Decode

Operators

- Attention
- FFN / Mixture-of-Experts
- Token Sampler / Speculative Decoder
- GeMM

- ***Efficiently generate next token*** *given partial text seq.*

- ***Effectively perform token prediction*** *by contextualizing token embeddings with minimal CPU / mem. cost*

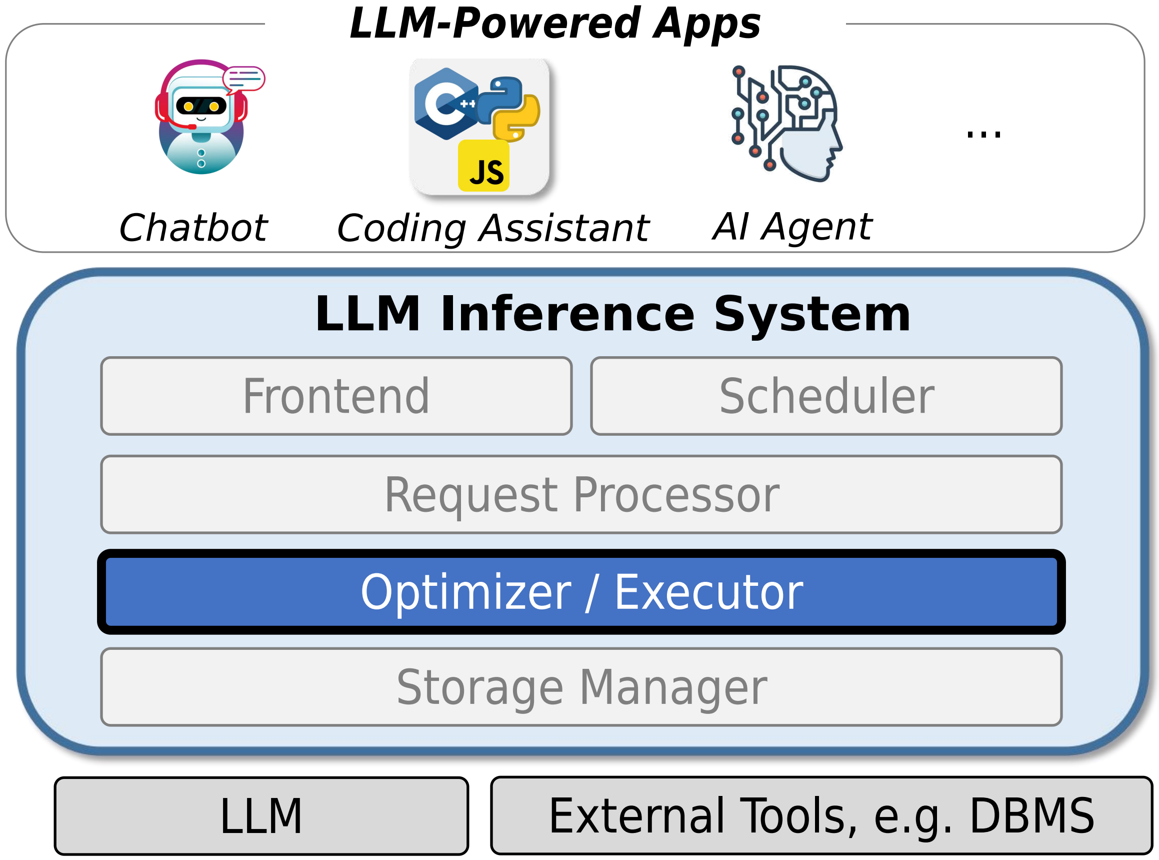
LLM Inference Systems: Executor

- ***Widespread LLM adoption leads to High-Volume, High-Velocity, & High-Variety inference workloads***

Goal: Build a system for High-Performance and High-Quality inference

1) LLM Uncertainty Principle: Can't know what you'll get until you run it

2) Autoregressive Generation: Output generated one token at a time



Hardware Acceleration

- FlashAttention
- FlashDecoding, RingAttention, LeanAttention

- ***Minimize operator costs** by exploiting special hardware*

Batch Executor

- Continuous Batching
- Bursted Attention

- ***Balance latency & throughput** by coordinating batch execution timing*

Distributed Executor

- Data (PD-Disagg.) / Model / Pipeline Parallel Executor

- ***Maximize throughput** by coordinating execution over distributed workers*

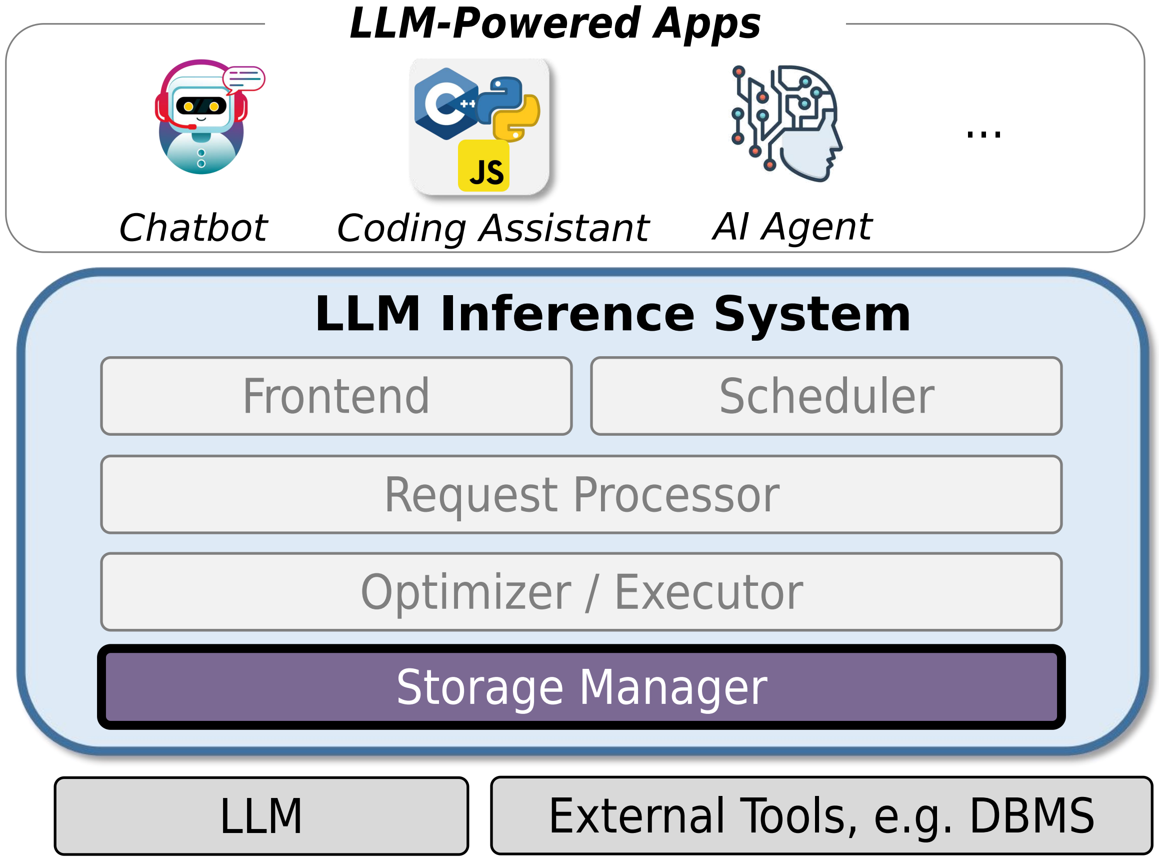
LLM Inference Systems: Storage

- ***Widespread LLM adoption leads to High-Volume, High-Velocity, & High-Variety inference workloads***

Goal: Build a system for High-Performance and High-Quality inference

1) LLM Uncertainty Principle: Can't know what you'll get until you run it

2) Autoregressive Generation: Output generated one token at a time



LLM-Powered Apps

Chatbot *Coding Assistant* *AI Agent* …

LLM Inference System

Frontend | Scheduler

Request Processor

Optimizer / Executor

Storage Manager

LLM | External Tools, e.g. DBMS

Block Manager
- Block Storage
- Block Search & Retrieval
- Block Sharing & Eviction

- *Manage KV cache blocks to **minimize wasted memory***

Quantizer
- Quantizer Design
- Outlier Protection

- *Compress model weights, activations, KV to **minimize memory usage***

Physical Storage
- Tiered Storage & Offloading
- Distributed Storage

- *Store model weights and KV caches for **efficient retrieval***

Part 1: Request Processing

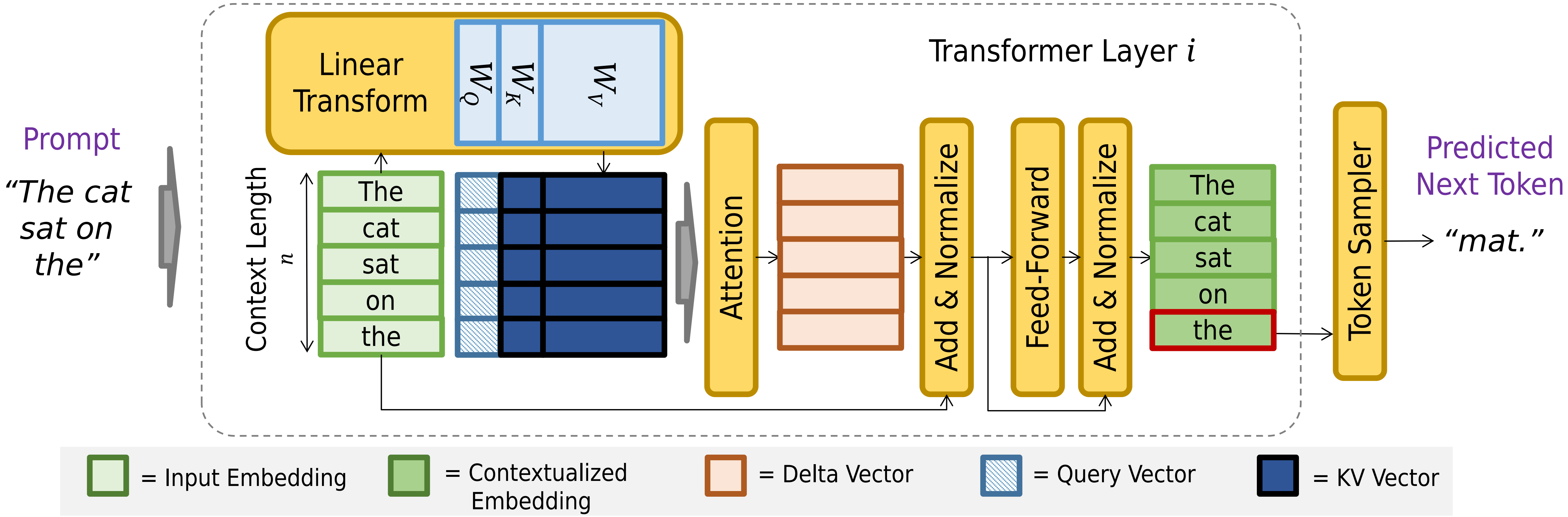
Efficiently and effectively generate next token by using contextualized embeddings

Request Processor	Technique Classification	Technique Description / Key Idea
Inference Workflow		
• Prefill	Workflow	
• Decode	Optimization	• Reduce compute complexity by exploiting KV cache
Operators		
• Attention		
• Naive Attention	Operator Design	
• Multi-Headed Attention	Operator Design	• Parallelized attention
• Grouped Attention	Operator Design	• Parallelized attention with shared heads
• Shared Attention	Optimization	• Reduce memory by sharing KV vectors
• Sparse Attention	Optimization	• Reduce memory & compute by discarding KVs
• FFN		
• Naive FFN	Operator Design	
• Mixture-of-Experts	Optimization	• Increase param. count (quality) w/o increasing cost
• Token Sampler		
• Greedy / Stochastic	Operator Design	
• Speculative Decoding	Optimization	• Increase token/sec via fast drafter with parallel verif.

Inference Workflow: Prefill

Inference Workflow: How to efficiently perform LLM inference?

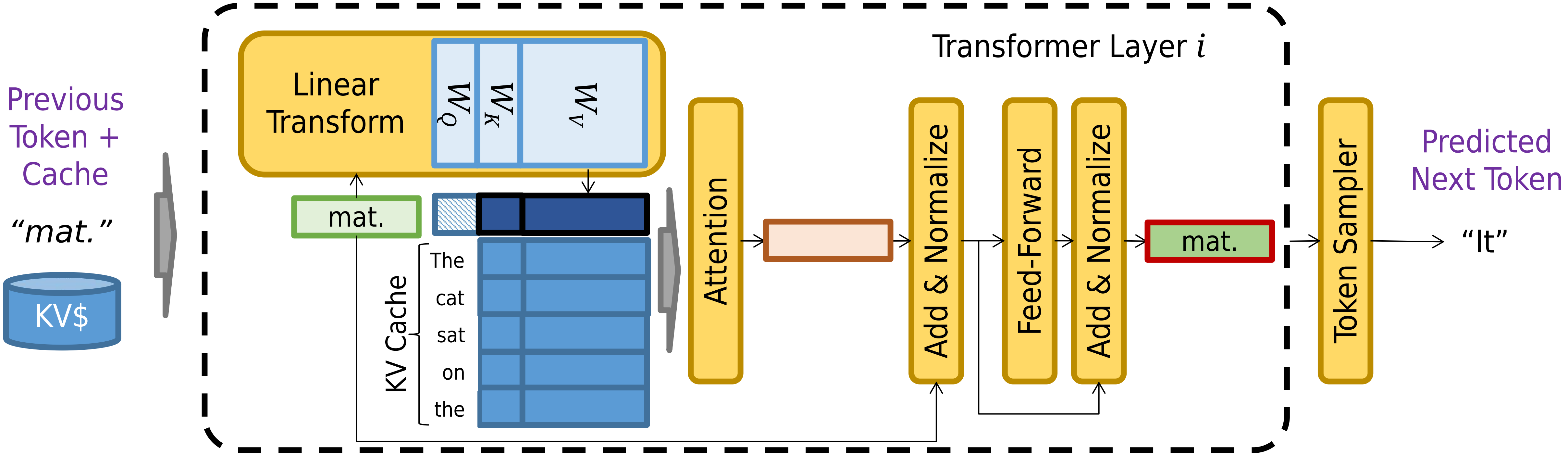
- ***Prefill**: Exploit GPU matmul to contextualize multiple tokens at once*



Inference Workflow: Decode

Inference Workflow: How to efficiently perform LLM inference?

- ***Decode: After prefill, exploit KV Cache to avoid reconstructing KVs***

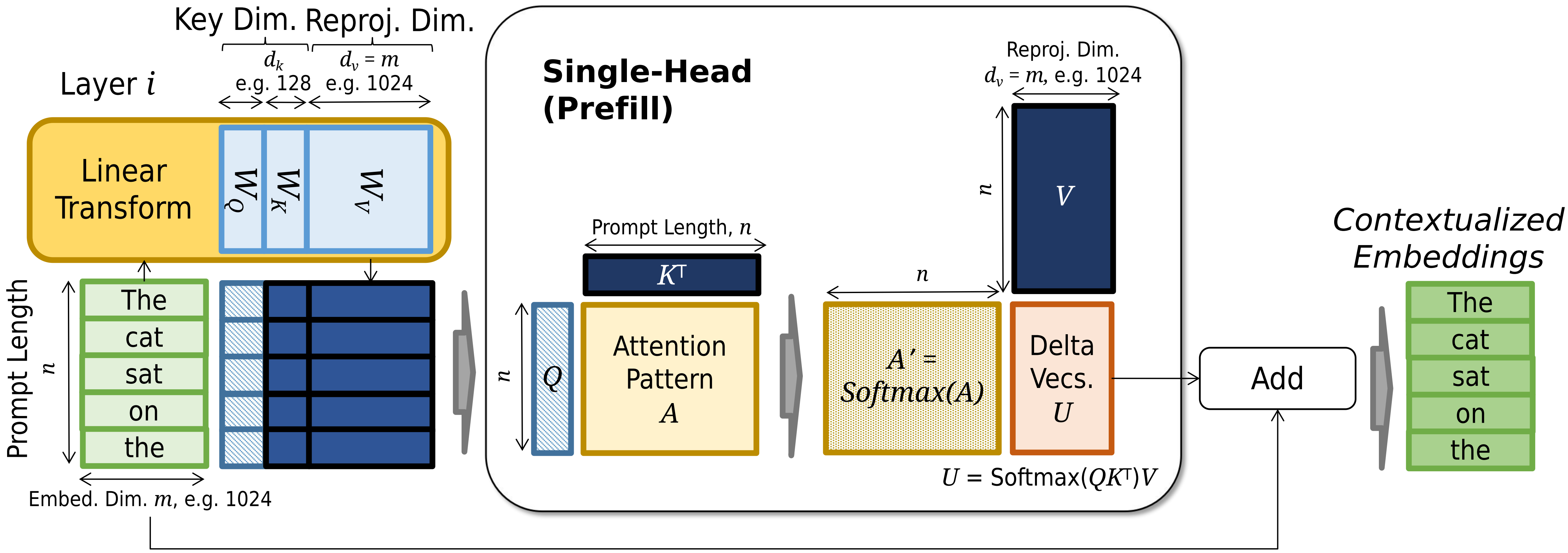


🟩 = Input Embedding 🟩 = Contextualized Embedding 🟧 = Delta Vector ▨ = Query Vector ■ = KV Vector 🟦 = Cached KVs

Operators: Naive Attention

Attention: How to efficiently contextualize an embedding vector?

- ***Naive**: Weight contributions of other tokens by learned query-key similarity*

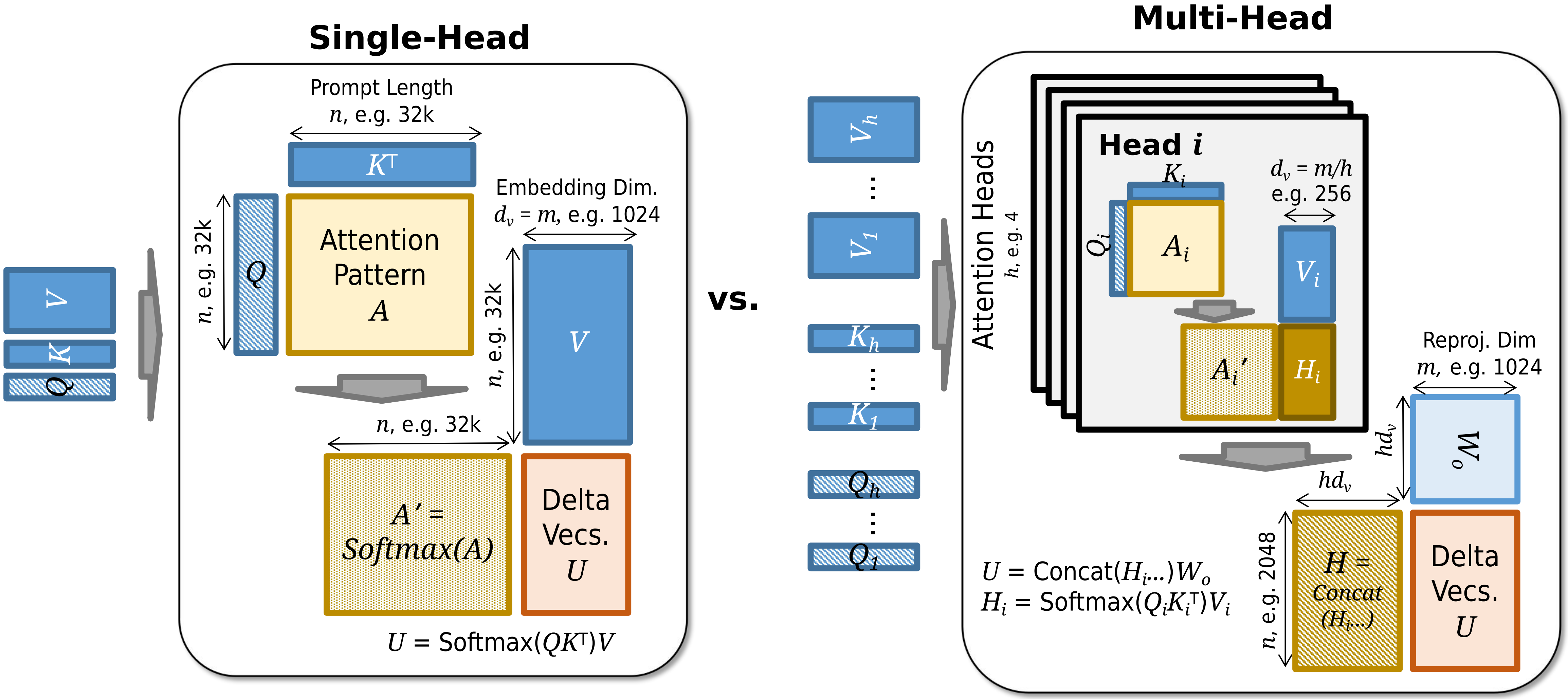


- Compute Cost: two matmuls + row-wise softmax
- Memory Cost: |Q|, |K|, |V|, |A|

Operators: Multi-Headed Attention

Attention: How to efficiently contextualize an embedding vector?

- ***Multi-Head (MHA): Split V across parallel "heads"***



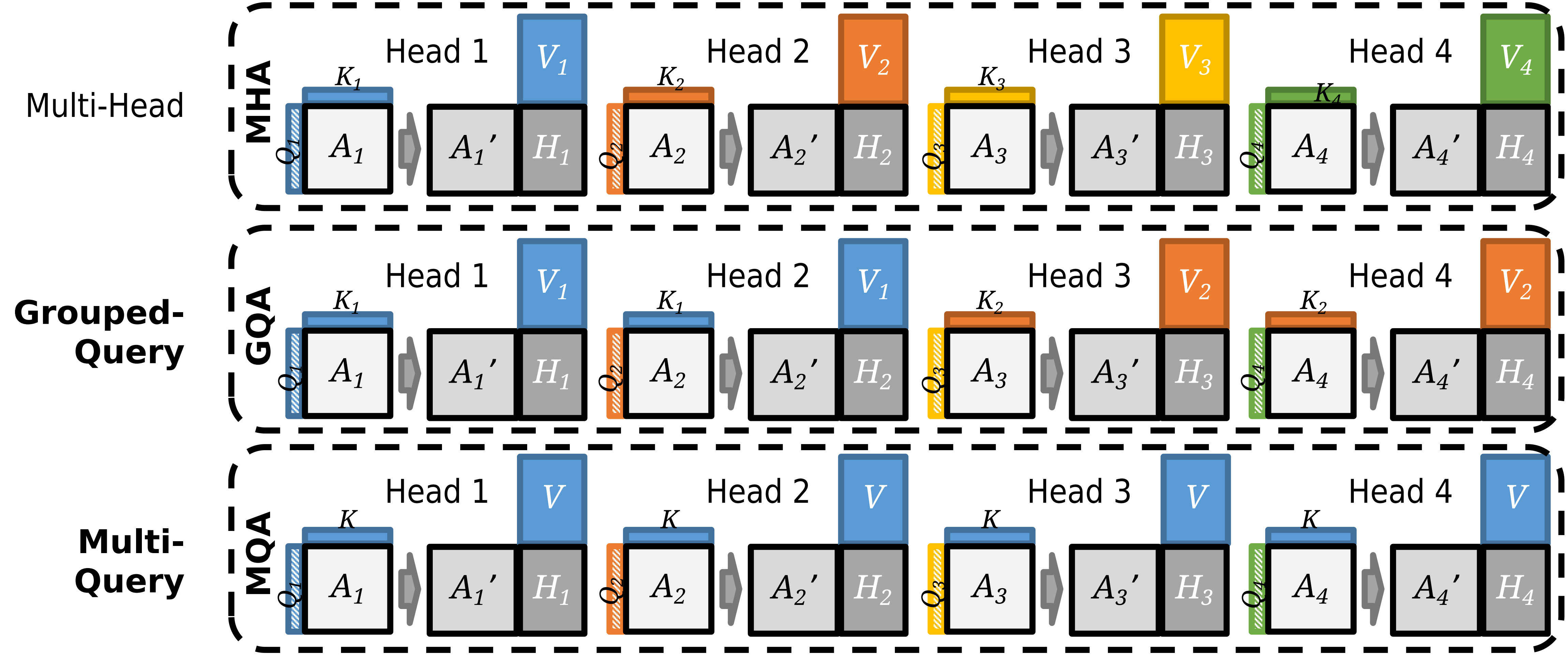
$U = \text{Softmax}(QK^\top)V$

$U = \text{Concat}(H_i\ldots)W_o$
$H_i = \text{Softmax}(Q_iK_i^\top)V_i$

Operators: Grouped Attention

Attention: How to efficiently contextualize an embedding vector?

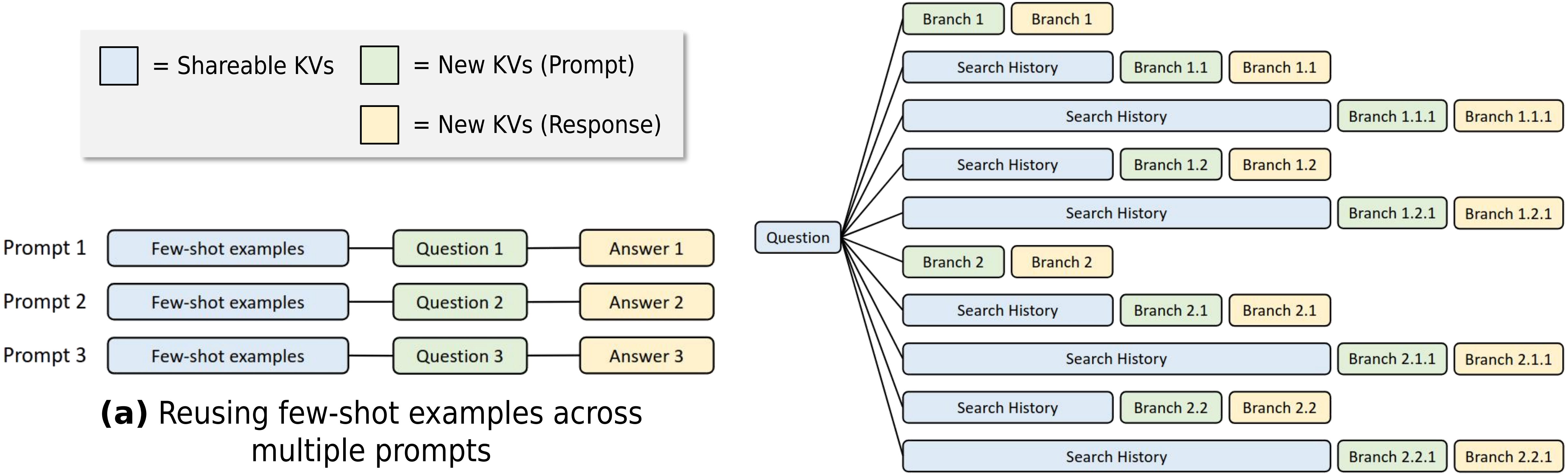
- ***Grouped Attention (GQA, MQA)**: Share KV projections across the heads*



Operators: Shared Attention

Attention: How to efficiently contextualize an embedding vector?

- ***Shared Attention**: Share KVs across multiple (sub)-requests*



(a) Reusing few-shot examples across multiple prompts

(b) Reusing ”thoughts” across multiple branches of a Tree-of-Thoughts process

Zheng, L et al. (2025) *SGLang: Efficient Execution of Structured Language Model Programs*, arXiv:2312.07104

Operators: Sparse Attention

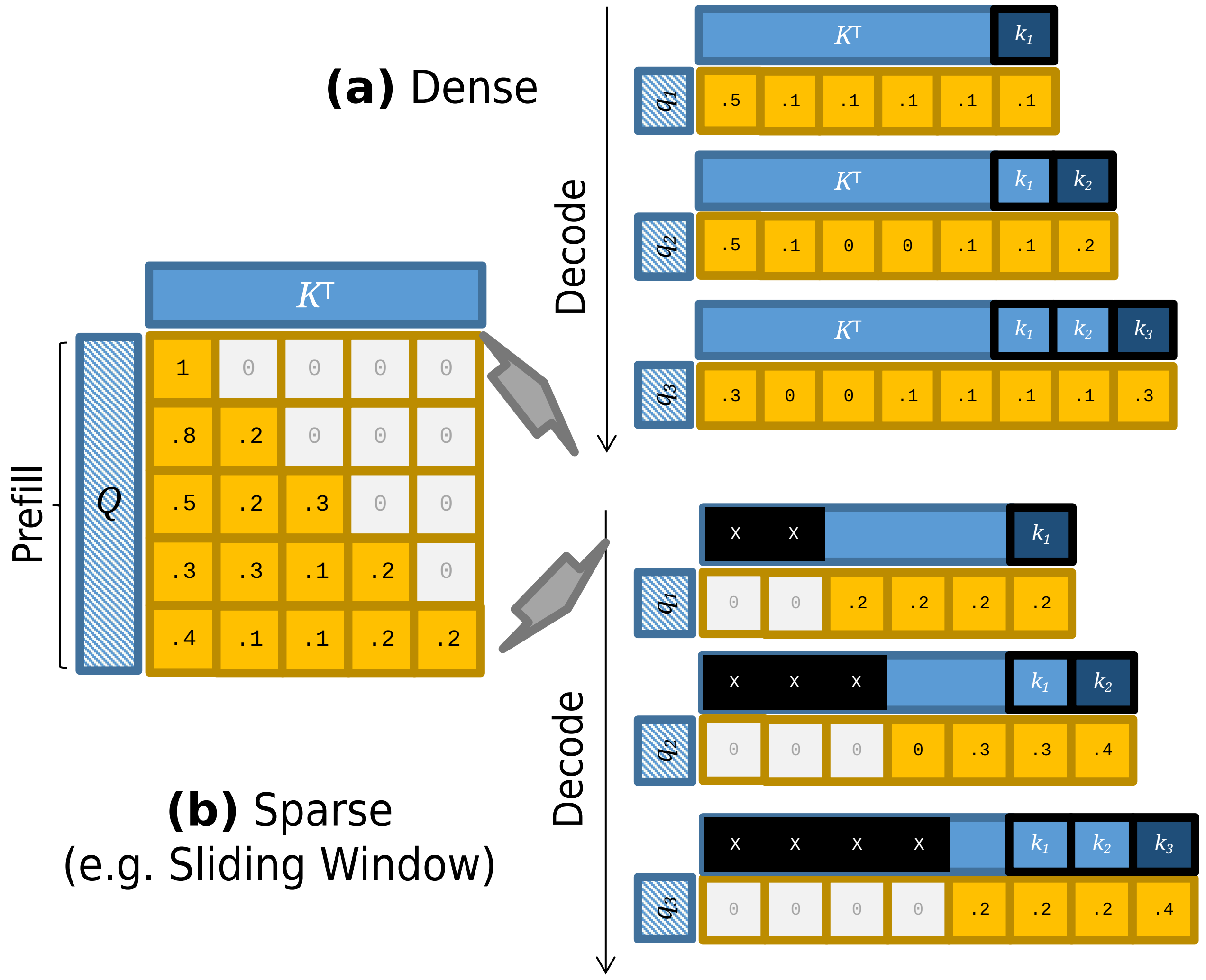
Attention: How to efficiently contextualize an embedding vector?

- ***Sparse Attention*: Compute QK similarities for only small subset of tokens**

Token Pruning

- *Heuristic Mask*
  - Sliding Window (Sparse Transformers)
  - Attention Sink (StreamingLLM)
- *Score-Based Pruning*
  - Attention Threshold (Scissorhands)
  - Accum. Attention (H2o "Heavy Hitters")
  - Approx. Attention (Loki, SparQ)
- *Learned Pruning*
  - Block Gating (SeerAttention)

🟦	= Cached Keys	X	= Evicted Keys
■	= New Keys	◩	= Attention Score



1	0	0	0	0
.8	.2	0	0	0
.5	.2	.3	0	0
.3	.3	.1	.2	0
.4	.1	.1	.2	.2

(a) Dense

q_1: .5 .1 .1 .1 .1 .1

q_2: .5 .1 0 0 .1 .1 .2

q_3: .3 0 0 .1 .1 .1 .1 .3

(b) Sparse
(e.g. Sliding Window)

q_1: 0 0 .2 .2 .2 .2

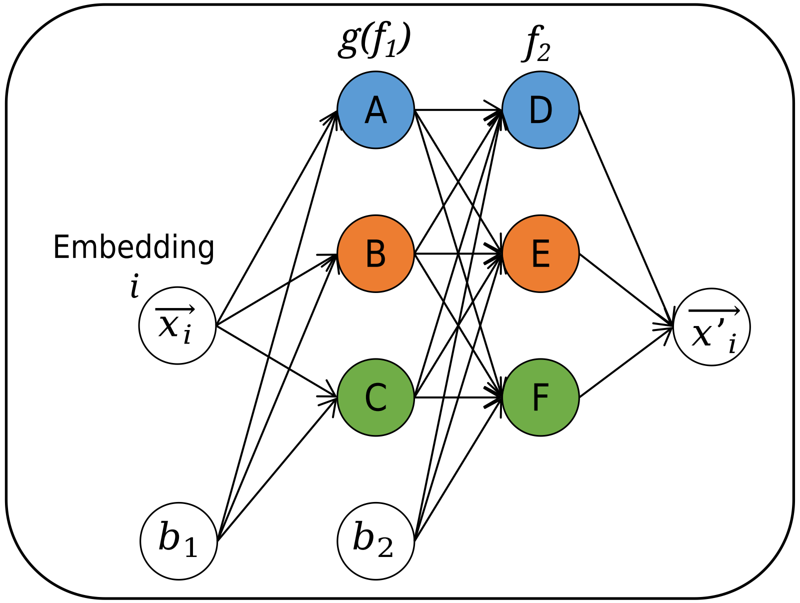
q_2: 0 0 0 0 .3 .3 .4

q_3: 0 0 0 0 .2 .2 .2 .4

Operators: Feed-Forward Network

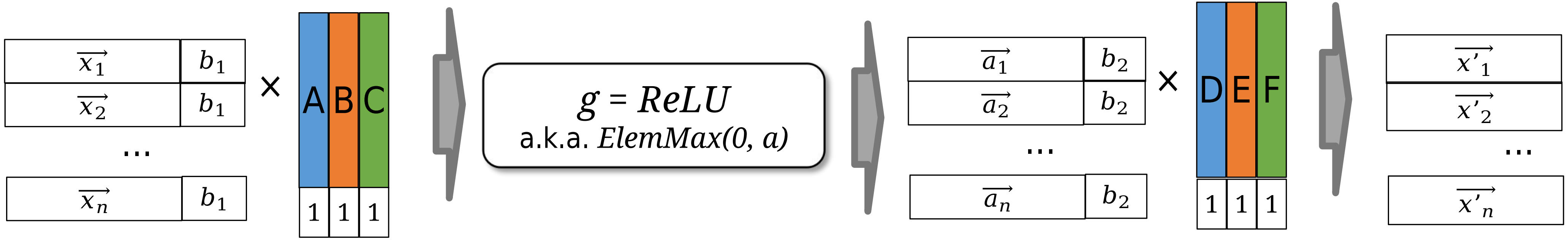
Feed-Forward: How to predict next token given contextualized token?

- ***Naive**: Construct next-token embedding via multi-layer perceptrons*



$$f_1(X) = XW_1 + b_1$$

$$f_2(A) = AW_2 + b_2$$

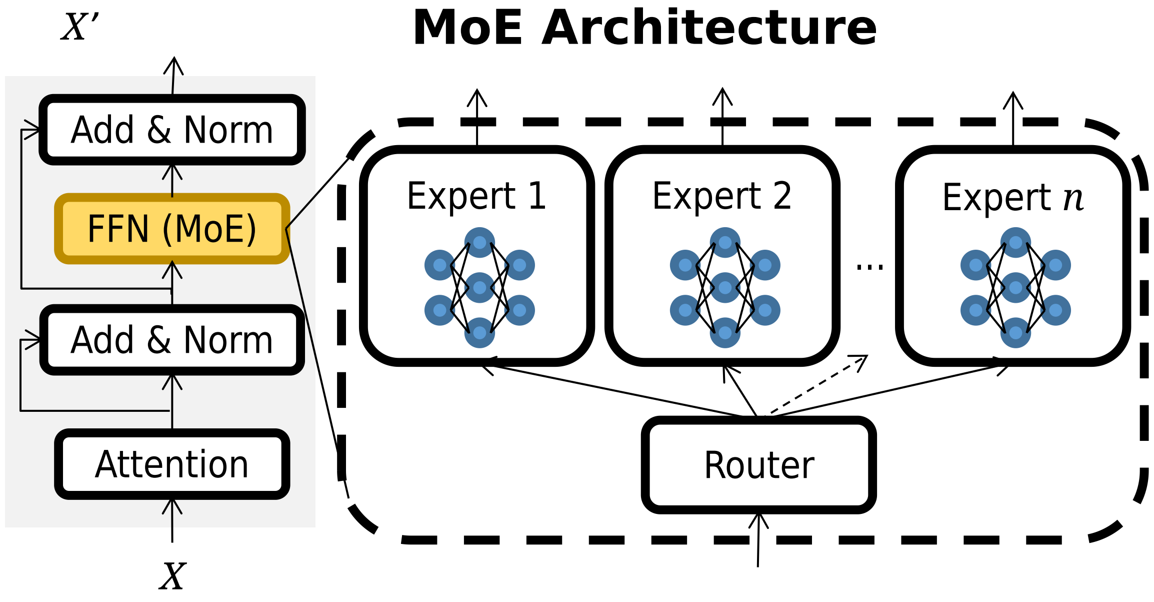

$\overrightarrow{x_1}$ b_1
$\overrightarrow{x_2}$ b_1
...
$\overrightarrow{x_n}$ b_1
×
A B C
1 1 1
$\overrightarrow{a_1}$ b_2
$\overrightarrow{a_2}$ b_2
...
$\overrightarrow{a_n}$ b_2
×
D E F
1 1 1
$\overrightarrow{x'_1}$
$\overrightarrow{x'_2}$
...
$\overrightarrow{x'_n}$

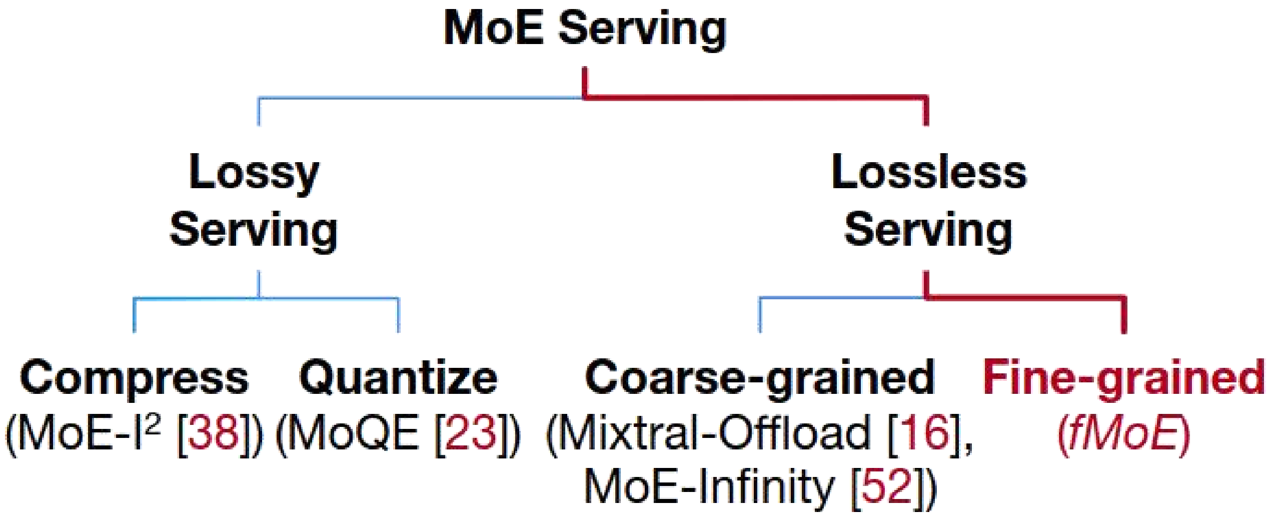
g = ReLU
a.k.a. *ElemMax(0, a)*

Operators: Mixture-of-Experts

Feed-Forward: How to predict next token given contextualized token?

- ***Mixture-of-Experts: Replace FFN with a m different "experts"***
  - **Single FFN**: n total parameters, n activated parameters during inference
  - **m Experts**: $\boldsymbol{m \times n}$ total parameters, $\boldsymbol{k \times n}$ activated parameters during inference





MoE Serving

Lossy Serving

Compress (MoE-I² [38])

Quantize (MoQE [23])

Lossless Serving

Coarse-grained (Mixtral-Offload [16], MoE-Infinity [52])

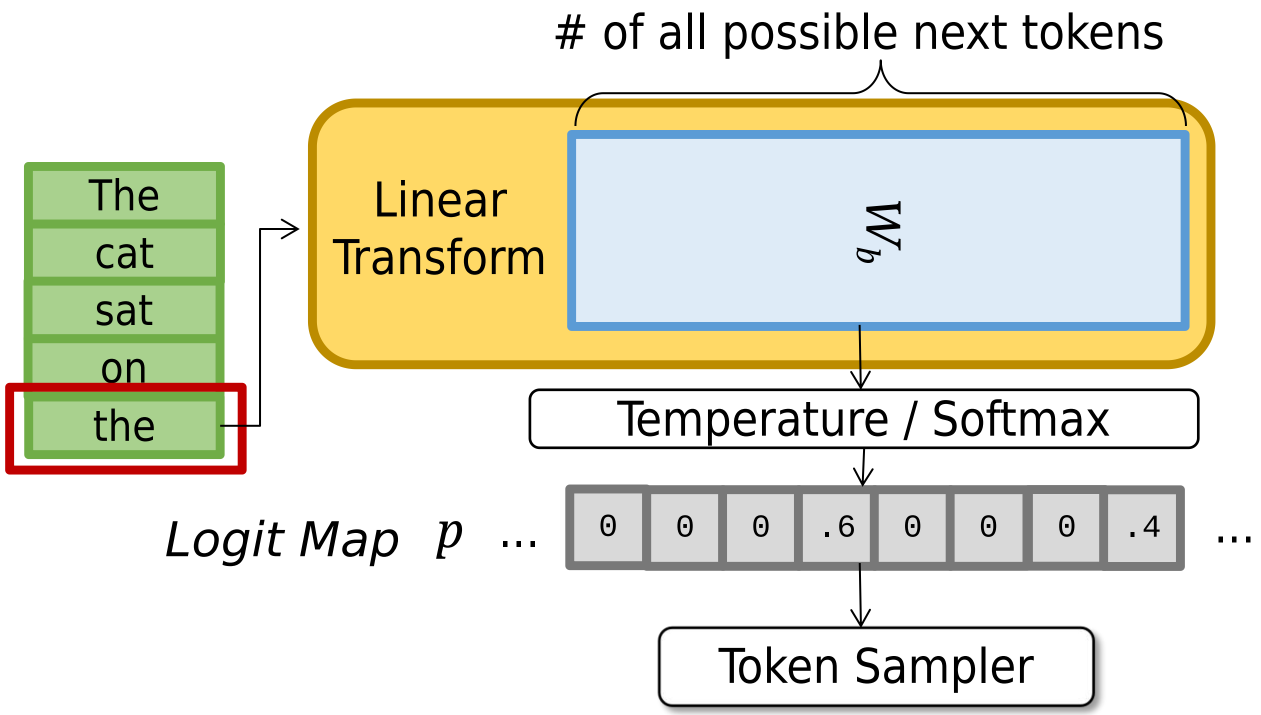
Fine-grained (*fMoE*)

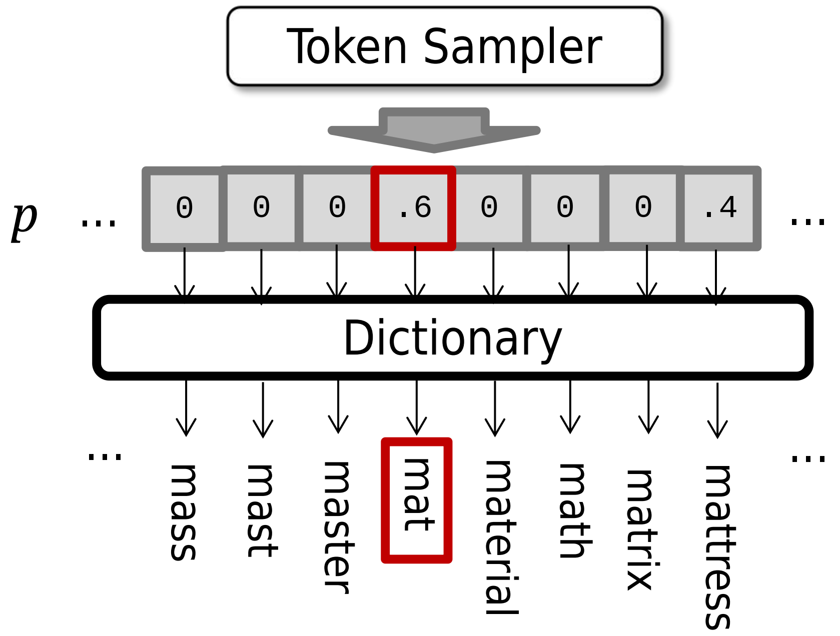
Yu, H et al. (2025) *fMoE: Fine-Grained Expert Offloading for Large Mixture-of-Experts Serving*, arXiv:2502.05370

Operators: Greedy / Stochastic Sampler

Token Sampler: How to select next token given predicted next-token embedding?

- ***Greedy*: Map from embedding onto token set & select max logit**
  - **Stochastic**: Randomly sample from the logit map according to logit value
  - **Top-K**: Randomly sample from k-largest logits
  - **p-Nucleus**: Set k so that logits sum to p

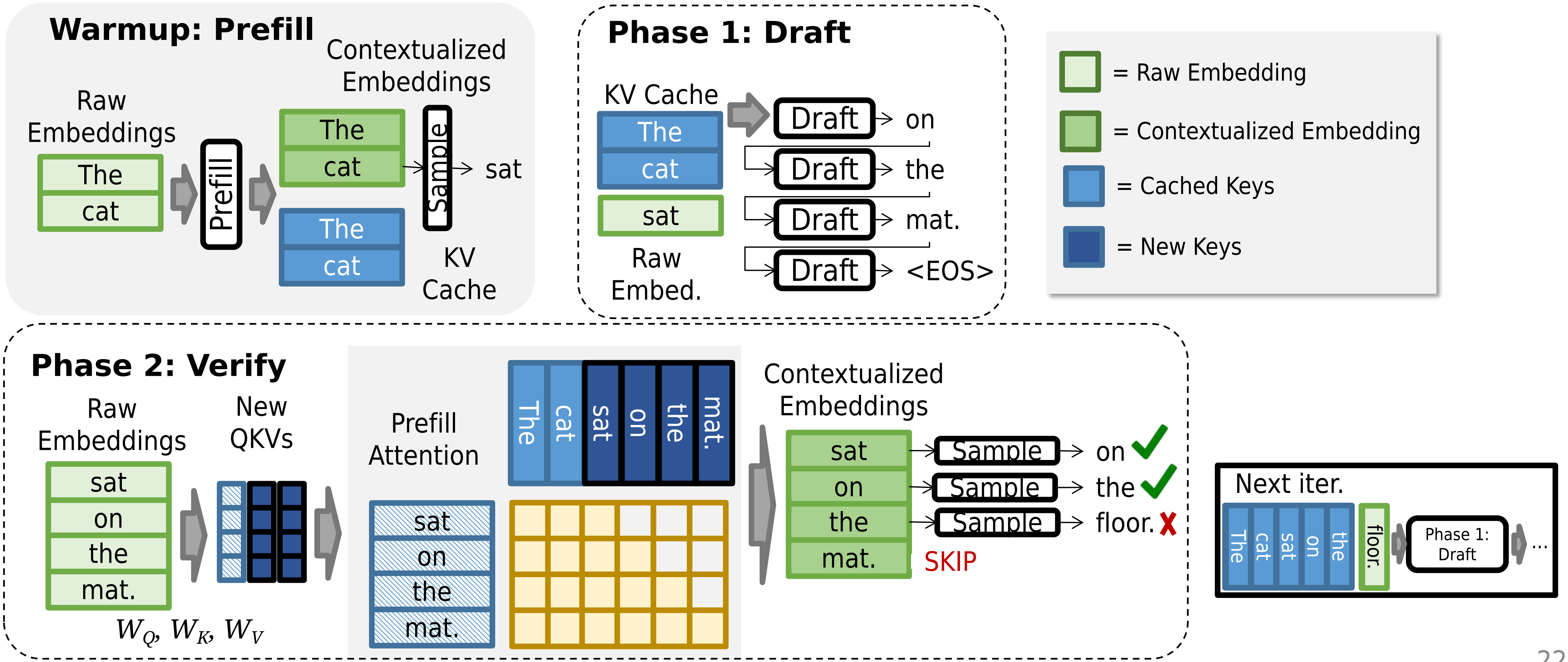




Operators: Speculative Decoding

Token Sampler: How to select next token given predicted next-token embedding?

- ***Speculative Decoding**: Quickly draft next k tokens, then quickly verify*



Request Processing: Summary

Efficiently and effectively generate next token by using contextualized embeddings

Request Processor	Technique Classification	Latency	Throughput	Memory	Quality
Inference Workflow					
• Prefill	Workflow				
• Decode	Optimization	↓	↑		
Operators					
• Attention					
• Naive Attention	Operator Design				
• Multi-Headed Attention	Operator Design	↓	↑	↑	
• Grouped Attention	Operator Design	↓	↑	↓	↓
• Shared Attention	Optimization	↓	↑	↓	
• Sparse Attention	Optimization	↓	↑	↓	↓
• FFN					
• Naive FFN	Operator Design				
• Mixture-of-Experts	Optimization			↑	↑
• Token Sampler					
• Greedy / Stochastic	Operator Design				
• Speculative Decoding	Optimization	↓	↑	↑	

Part 2: Optimizer / Execution

Minimize op. costs via hardware kernels; balance throughput / lat. by coordinating execution

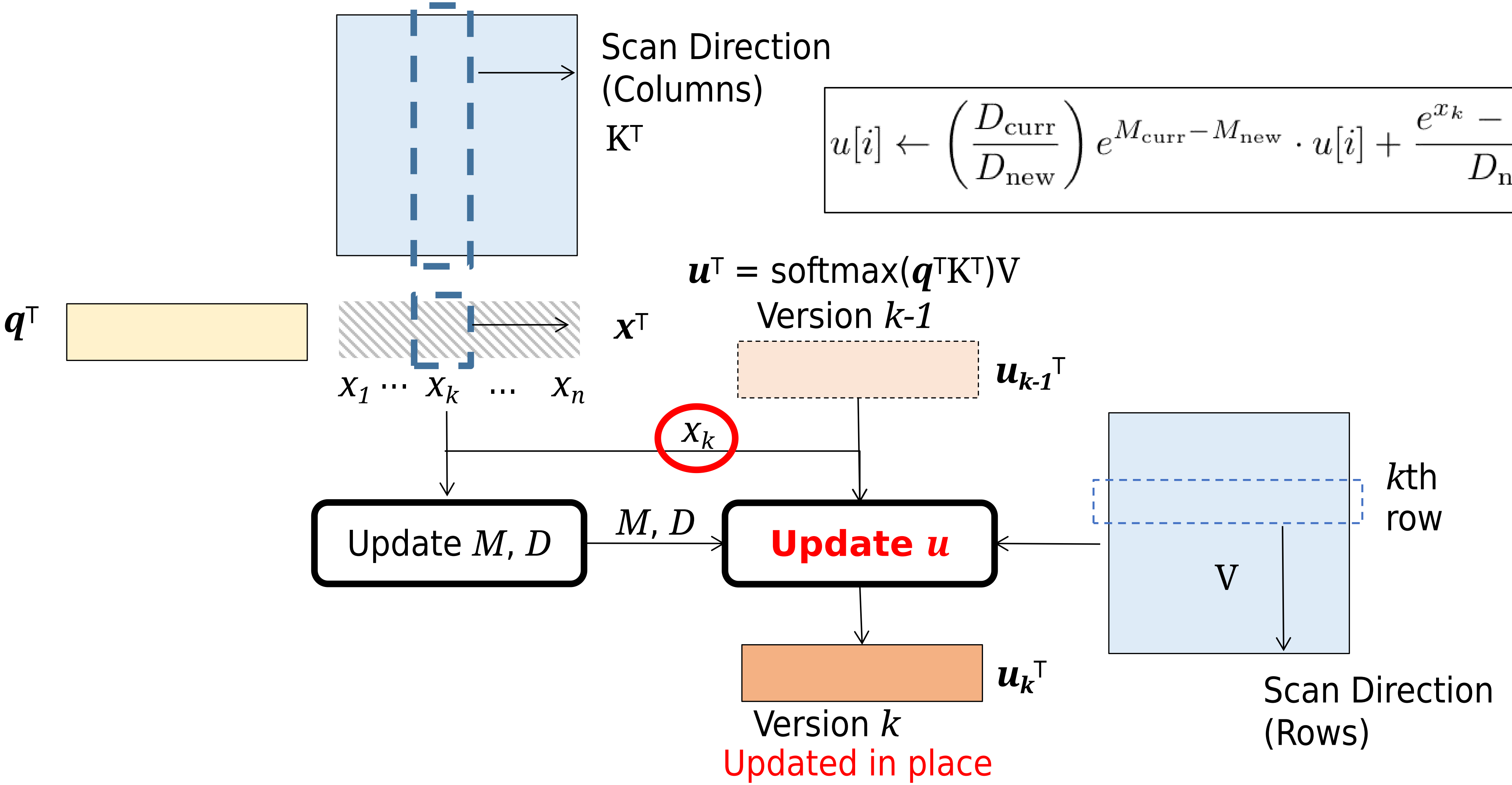
Optimizer / Execution	Technique Classification	Technique Description / Key Idea
Hardware Acceleration		
• FlashAttention	Kernel Design	• Reduce memory & I/O via kernel fusion
• FlashDecoding, RingAttention	Kernel Design	• Parallelized blockwise attention
• LeanAttention	Optimization	• Maximize core utilization via streaming load balanc.
Batch Executor		
• Static Batching	Workflow	
• Continuous Batching	Workflow	• Mitigate straggler effects via dynamic rebatching
• Bursted Attention	Workflow	• Batch splitting and merging
Distributed Executor		
• Model Parallelism	Workflow	• Parallelize across layers
• Pipeline Parallelism	Workflow	• Parallelize across requests in different stages
• Data Parallelism		
• Multi-Replica	Architecture	• Add multiple LLM replicas to increase throughput
• PD-Disaggregated	Architecture	• Decouple P and D replicas to allow flexibility

Hardware Accel.: FlashAttention

Hardware Accel.: How to implement efficient operators over specialized hardware?

- ***FlashAttention**: Update delta vector in place via online softmax & matmul*

$$u[i] \leftarrow \left(\frac{D_{\text{curr}}}{D_{\text{new}}}\right) e^{M_{\text{curr}} - M_{\text{new}}} \cdot u[i] + \frac{e^{x_k} - M_{\text{new}}}{D_{\text{new}}} \cdot V_i[k]$$

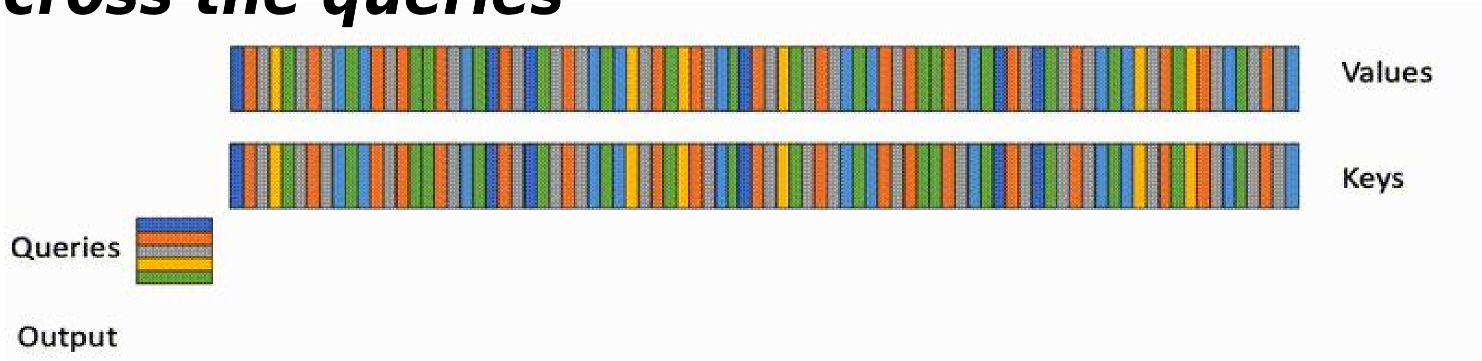


Hardware Accel.: FlashDecoding

Hardware Accel.: How to implement efficient operators over specialized hardware?

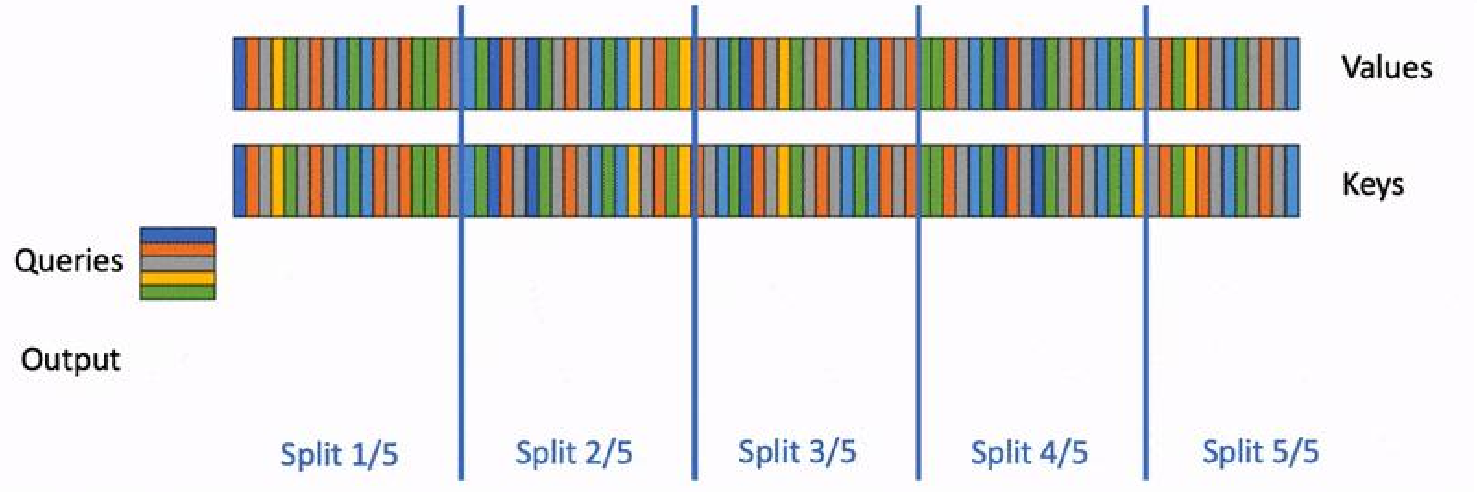
- ***FlashAttention: Shard across the queries***

Inter-query: Each worker gets different query block but share key-value blocks



- ***FlashDecoding: Shard across KV followed by global reduction***

Intra-query: Each worker gets different key-value blocks followed by global reduction step

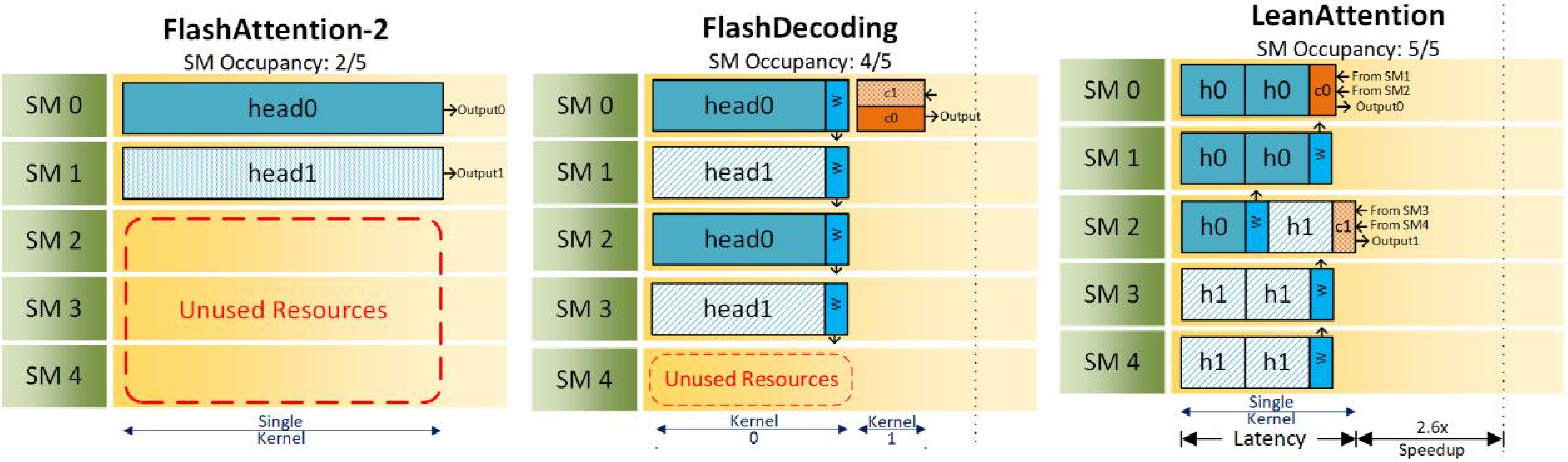


Dao, T., Haziza, D., Massa, F., and Sizov, G. *Flash-decoding for long-context inference*, 2023

Hardware Accel.: LeanAttention

Hardware Accel.: How to implement efficient operators over specialized hardware?

- ***LeanAttention*: Stream mini-blocks to GPU cores followed by global reduct.**

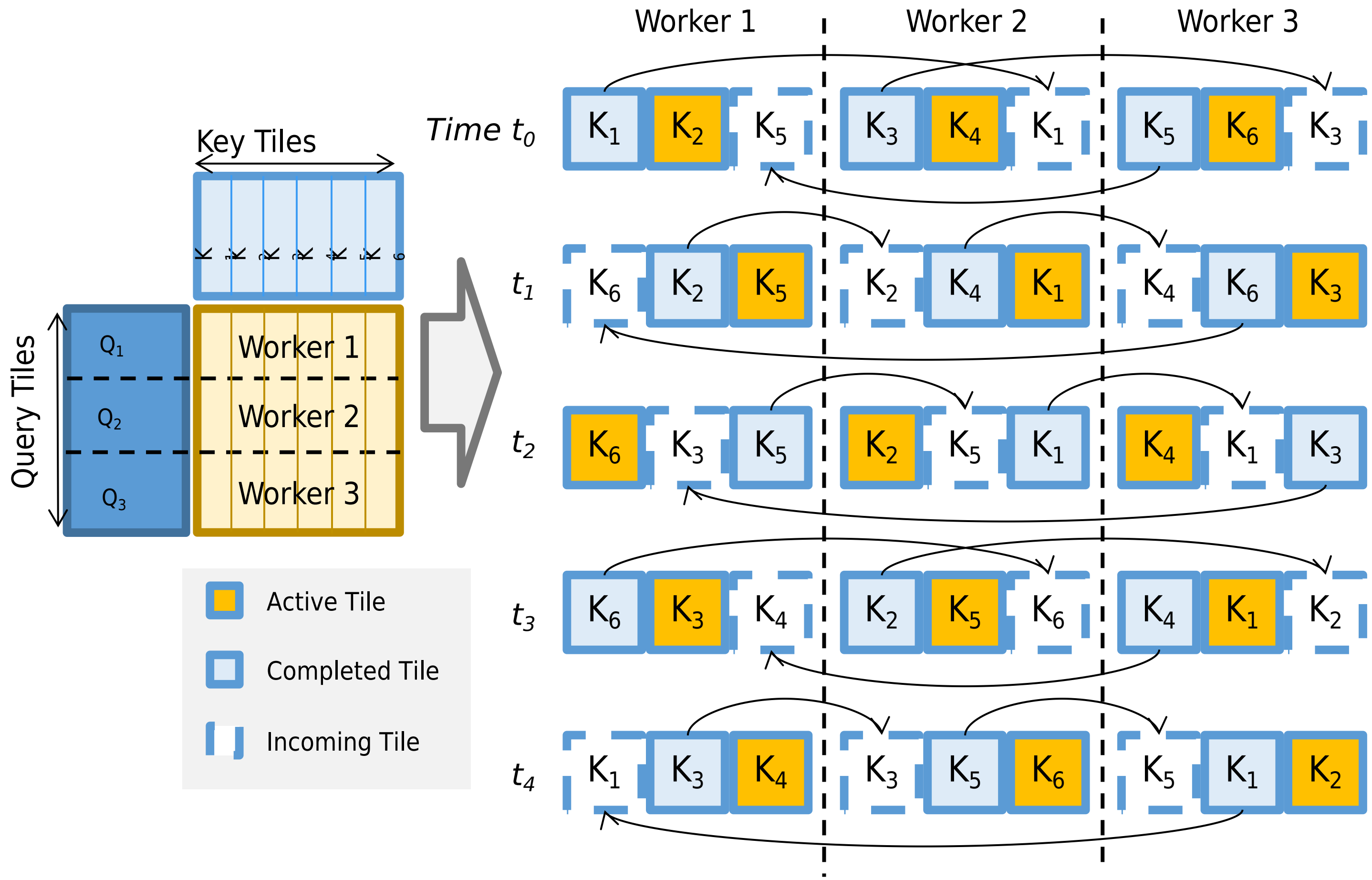


Rya S., Srikant B., Renee SA., Victor R., Saravan R. *Lean Attention: Hardware-Aware Scalable Attention Mechanism for the Decode-Phase of Transformers.* *arXiv:2405.10480*

Hardware Accel.: RingAttention

Hardware Accel.: How to implement efficient operators over specialized hardware?

- ***RingAttention*: Distributed blocks + fixed transfer sequence**
  - Each worker needs to read every cache block, but what to do if cache exceeds worker memory?
  - Distribute blocks across workers, then use fixed transfer sequence to hide transfer overhead



Key Tiles
K_1 K_2 K_3 K_4 K_5 K_6
Query Tiles
Q_1
Q_2
Q_3
Worker 1
Worker 2
Worker 3
Active Tile
Completed Tile
Incoming Tile
Worker 1
Worker 2
Worker 3
Time t_0
K_1 K_2 K_5 K_3 K_4 K_1 K_5 K_6 K_3
t_1
K_6 K_2 K_5 K_2 K_4 K_1 K_4 K_6 K_3
t_2
K_6 K_3 K_5 K_2 K_5 K_1 K_4 K_1 K_3
t_3
K_6 K_3 K_4 K_2 K_5 K_6 K_4 K_1 K_2
t_4
K_1 K_3 K_4 K_3 K_5 K_6 K_5 K_1 K_2

Batching: Continuous Batching

Batching: How to avoid stragglers during batch formation?

- ***Continuous Batching**: Reconstitute the batch after each round*

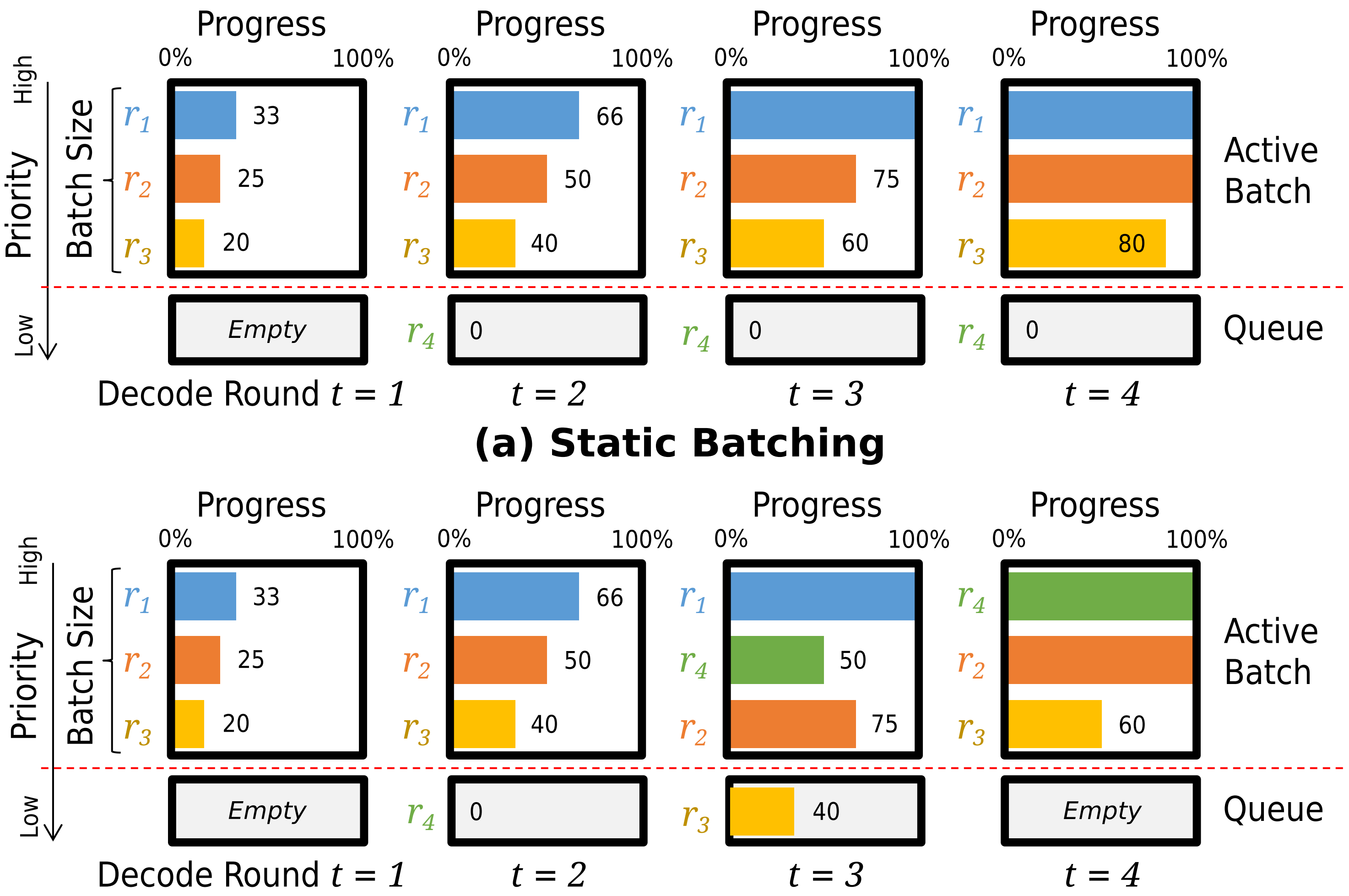
Static Batching

- Requests 1 and 2 are held up by Request 3 (*straggler*)
- Request 4 cannot start until the R1R2R3 batch completes

Continuous Batching

e.g. Shortest-Job First

- Request 4 starts immediately b.c. higher priority than e.g. R3
- Requests 1 and 2 can return immediately once they finish
- Request 3 takes longer b.c. it got preempted by R4



(a) Static Batching

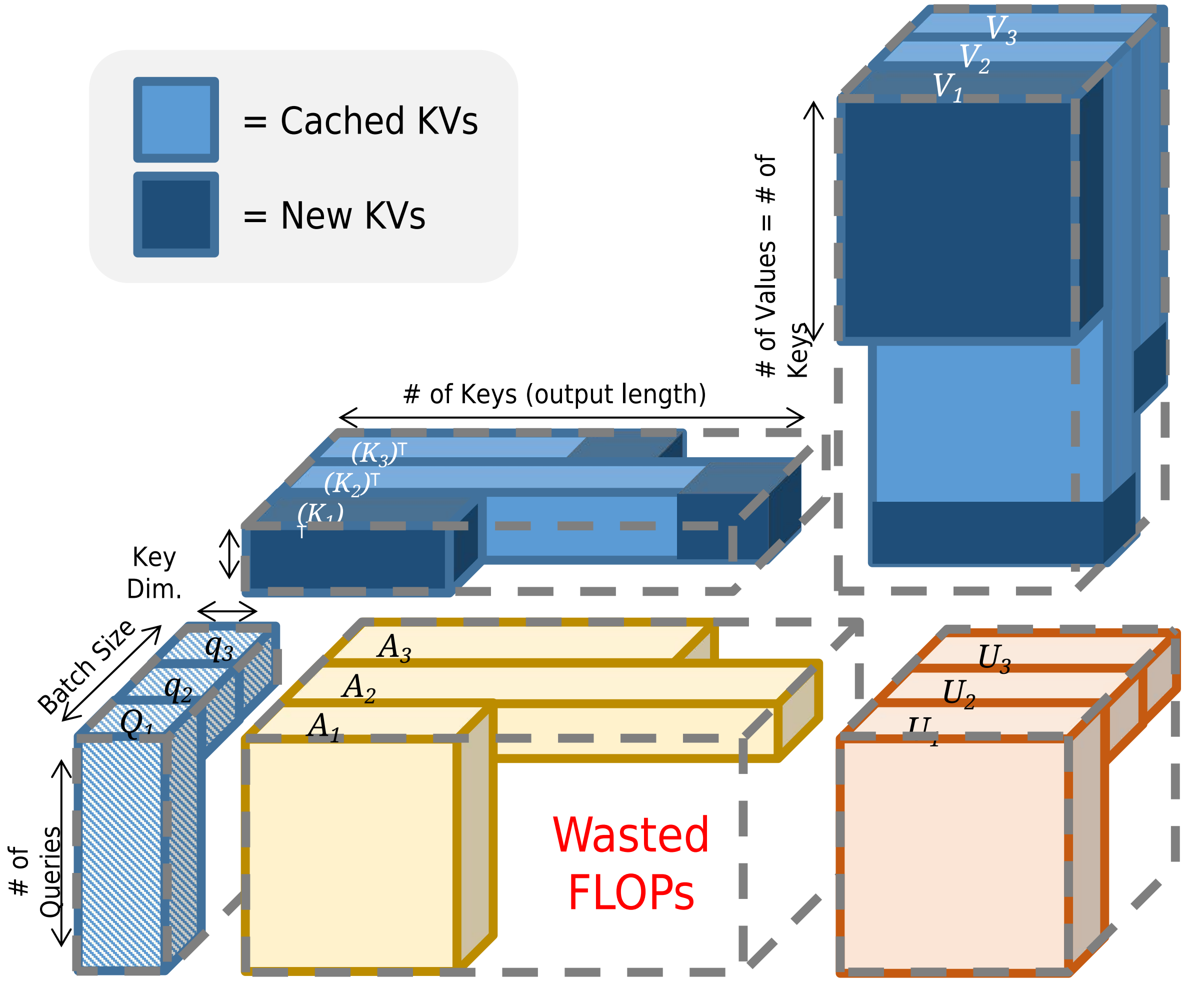
(b) Continuous Batching

Batching: Bursted Attention

Batching: How to avoid stragglers during batch formation?

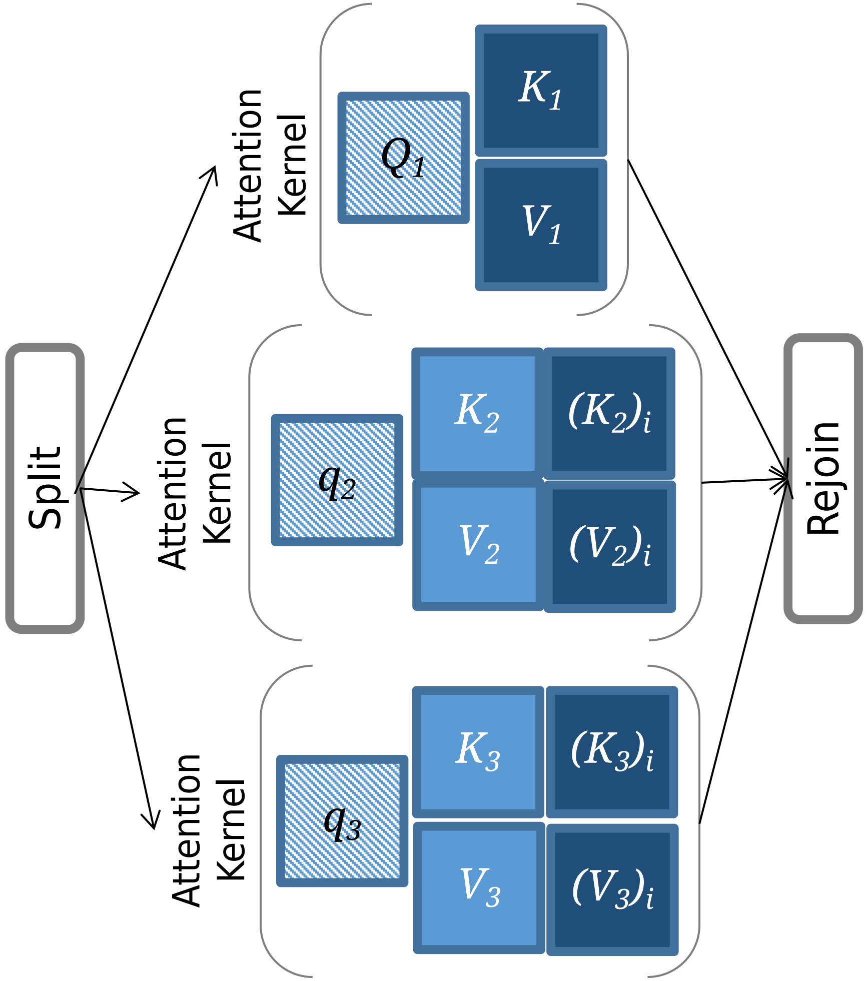
- ***Bursted Attention: Split for attention and rejoin for matrix ops.***

Batched Attention



vs

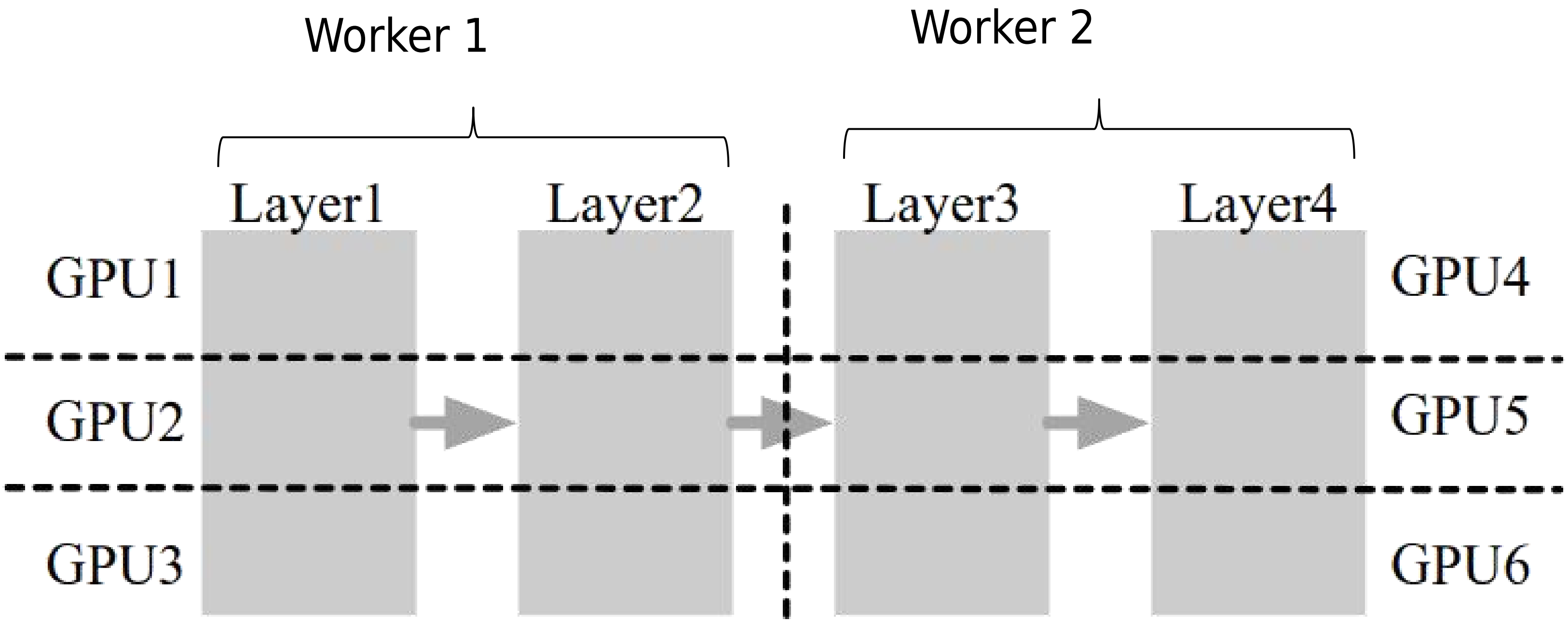
Bursted Attention



Distributed Exec.: Model Parallelism

Distributed Exec.: How to take advantage of multiple executors?

- ***Model Parallelism: Split large model across transformer layers***
  - Avoid memory pressure on a single worker

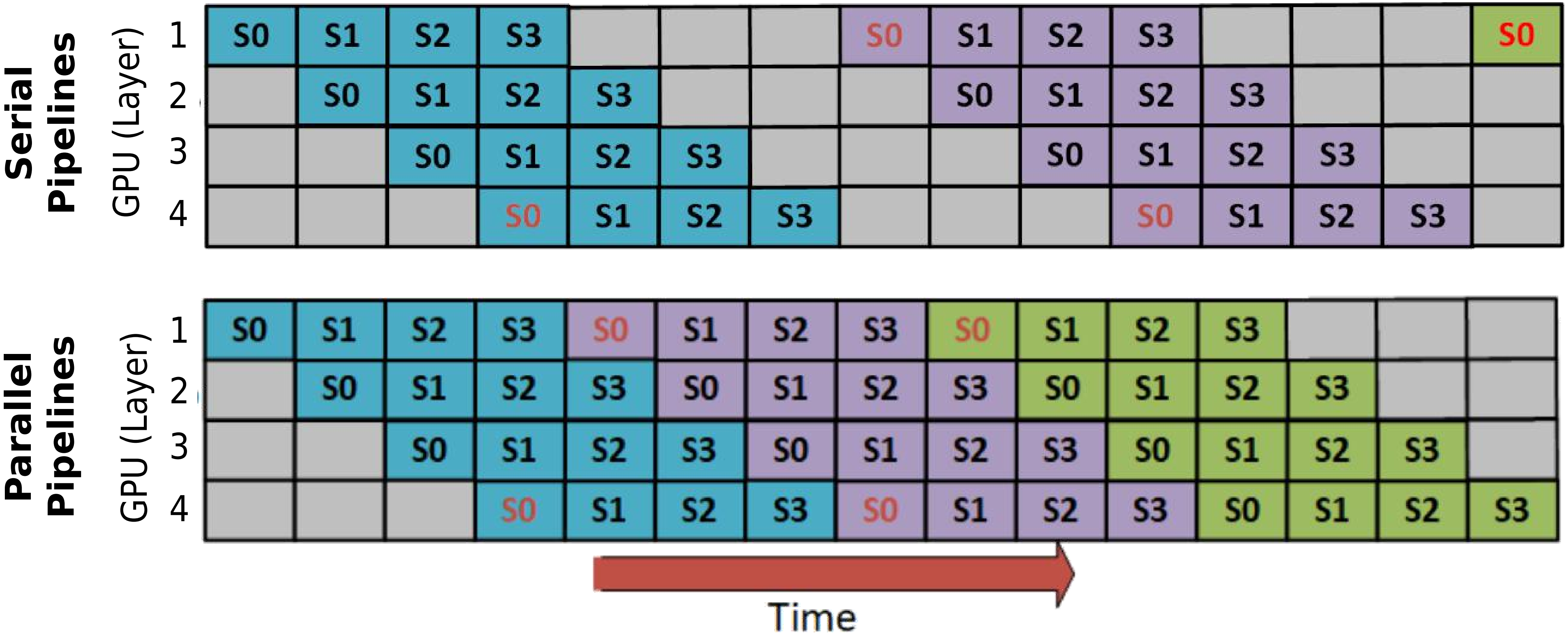


Yu G. I., Jeong J. S., Kim G. W., Kim S., Chun B. G. *ORCA: A Distributed Serving System for Transformer-Based Generative Models. OSDI'22*

Distributed Exec.: Pipeline Parallelism

Distributed Exec.: How to take advantage of multiple executors?

- ***Pipeline Parallelism*: Concurrently execute multiple pipelines**



Serial Pipelines

GPU (Layer)															
1	S0	S1	S2	S3				S0	S1	S2	S3				S0
2		S0	S1	S2	S3				S0	S1	S2	S3			
3			S0	S1	S2	S3				S0	S1	S2	S3		
4				S0	S1	S2	S3				S0	S1	S2	S3	

Parallel Pipelines

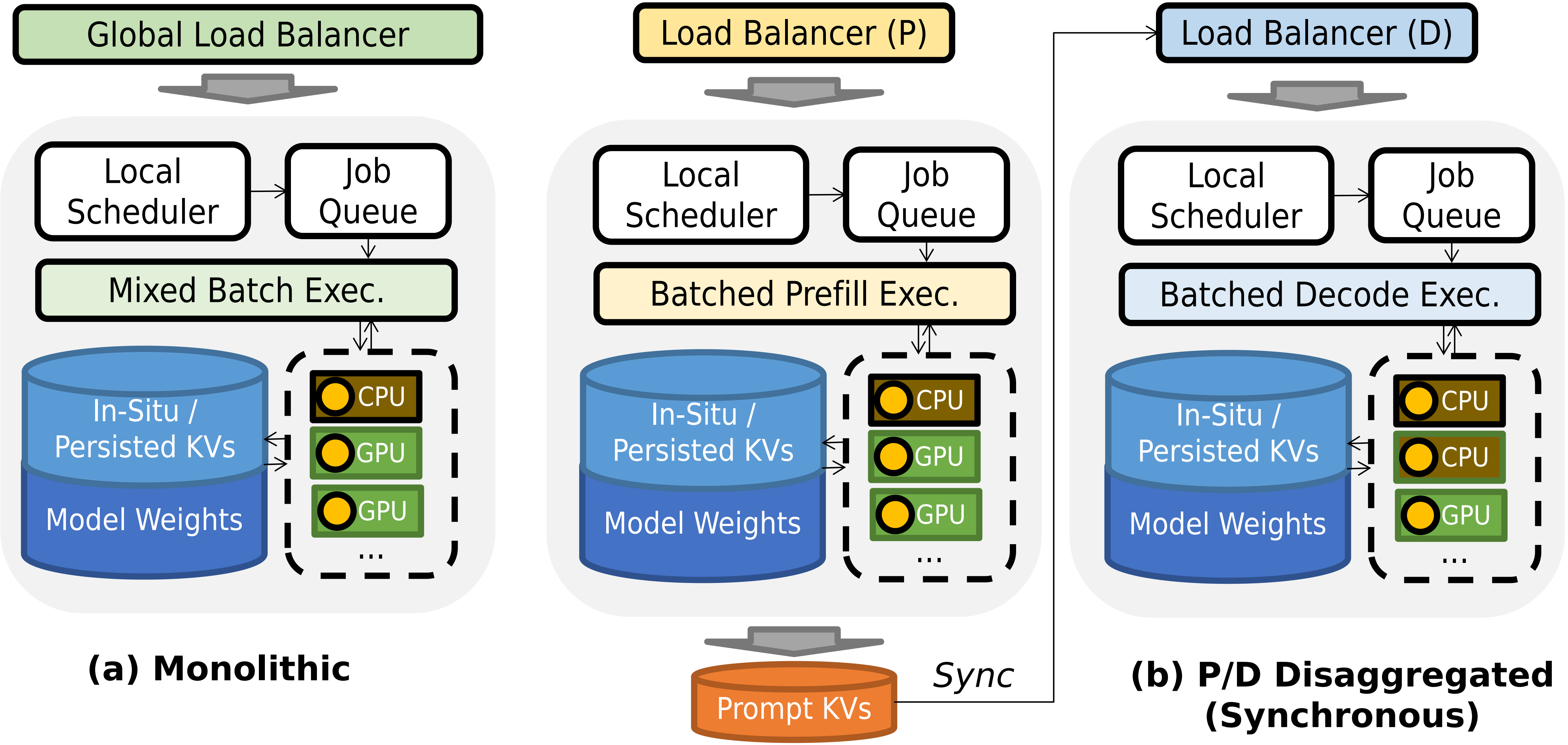
GPU (Layer)															
1	S0	S1	S2	S3	S0	S1	S2	S3	S0	S1	S2	S3			
2		S0	S1	S2	S3	S0	S1	S2	S3	S0	S1	S2	S3		
3			S0	S1	S2	S3	S0	S1	S2	S3	S0	S1	S2	S3	
4				S0	S1	S2	S3	S0	S1	S2	S3	S0	S1	S2	S3

Aminabadi R. Y., Rajbhandari S., Zhang M., Awan A. A., Li C., Li D., Zheng E., Rasley J., Smith S., Ruwase O., He Y. *DeepSpeed Inference: Enabling Efficient Inference of Transformer Models at Unprecedented Scale. arXiv:2207.00032*

Distributed Exec.: Data Parallelism

Distributed Exec.: How to take advantage of multiple executors?

- ***Data Parallelism**: Deploy multiple LLM replicas to increase throughput*



(a) Monolithic

(b) P/D Disaggregated (Synchronous)

Optimizer / Execution: Summary

Minimize op. costs via hardware kernels; balance throughput / lat. by coordinating execution

Optimizer / Execution	Technique Classification	Latency	Throughput	Memory	Quality
Hardware Acceleration					
• FlashAttention	Kernel Design	↓	↑	↓	
• FlashDecoding, RingAttention	Kernel Design	↓	↑	↓	
• LeanAttention	Optimization	↓	↑	↓	
Batch Executor					
• Static Batching	Workflow				
• Continuous Batching	Workflow		↑		
• Bursted Attention	Workflow	↓	↑		
Distributed Executor					
• Model Parallelism	Workflow		↑	↓	
• Pipeline Parallelism	Workflow	↓	↑	↓	
• Data Parallelism					
• Multi-Replica	Architecture	↓	↑	↑	
• PD-Disaggregated	Architecture	↓	↑	↑	

Part 3: Scheduler

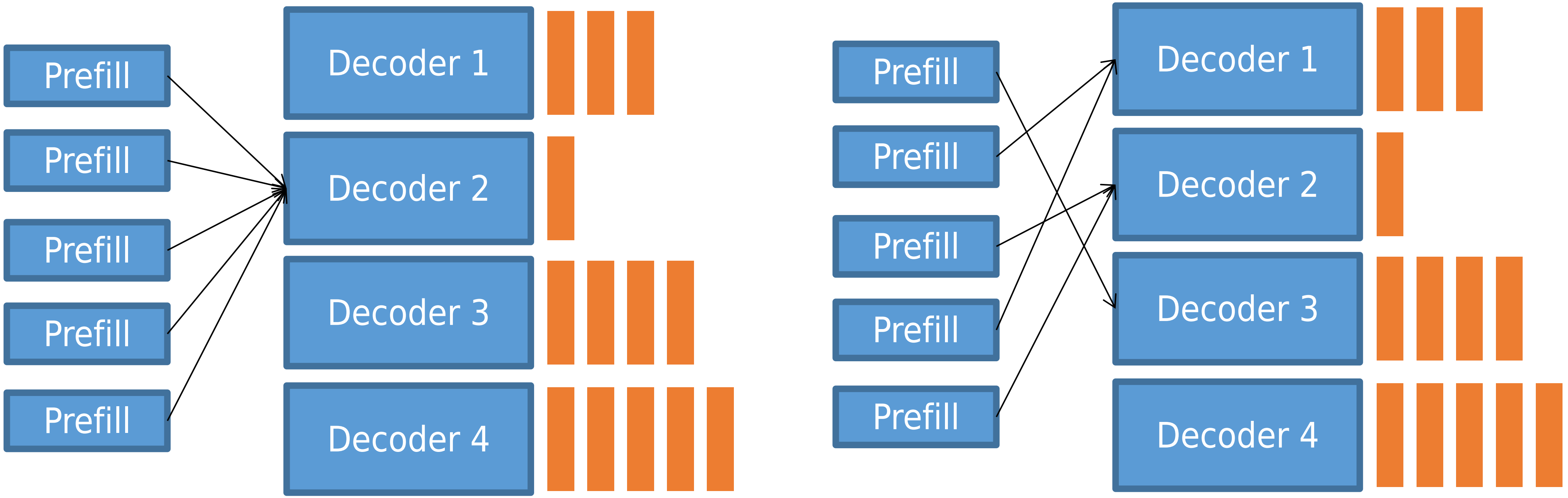
Minimize queuing delays and maximize resource utilization by balancing the load

Scheduler	Technique Classification	Technique Description / Key Idea
Load Balancer		
• Job Assignment		
• Greedy	Algorithm	
• Power-of-2	Algorithm	• Reduce overloading by 2-phase assignment
• Load Prediction (SAL)	Model (Heuristic)	• Develop a model for predicting worker load
Scheduler		
• Job Prioritizer		
• First-Come First-Serve	Algorithm	
• Shortest-Job	Algorithm	• Minimize queueing delays by prioritizing fast jobs
• Multi-Level Queue	Algorithm	• Simulate shortest-job by using multiple queues
• Job Cost Prediction		
• Cache / Prompt Based	Model (Heuristic)	• Use cache / prompt length as proxy for job cost
• Learning-Based	Model (Learned)	• Train a model to predict job cost
Batch Controller		
• Chunking Module	Optimization	• Balance latency / throughput via chunk sizing
• Batch Size Control	Optimization	• Balance latency / throughput via batch sizing

Load Balancer: Job Assignment

Job Assignment: How to assign jobs to workers under dynamic and uncertain loads?

- ***Greedy*: Assign requests to least-load worker at time of assignment**
  - Under static loads, this is 2-competitive in worst-case but requires accurate load prediction
- ***Power-of-Two*: Assign to greedy worker out of <u>random 2</u>** [Hu et al 2024 "TetriInfer"]
  - Exponentially smaller makespan compared to random (but not as good as greedy) [Mitzenmacher 2001]
  - Under dynamic loads, avoids overloading workers



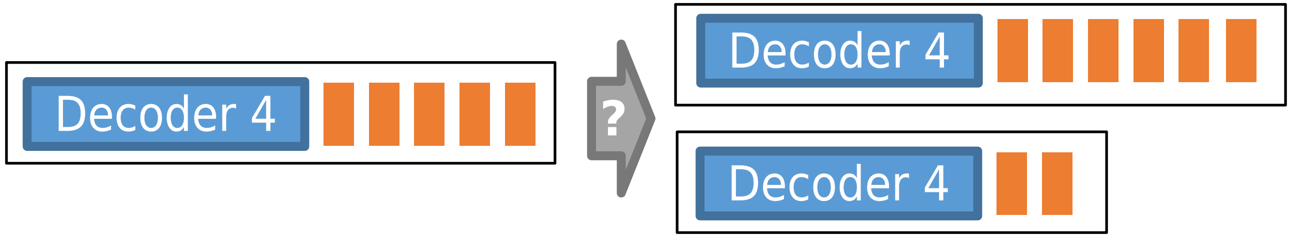
(a) Greedy **(b) Power-of-Two**

Load Balancer: Load Prediction

Load Prediction: How to measure worker load while considering dynamic job costs?

- Sources of Uncertainty:
  - Dynamic memory growth:
    - In-situ KV caches from existing / new requests
    - Reloaded caches from request resumptions
  - Dynamic memory reclamation:
    - Offloaded or evicted caches from preempted / finished requests
- ***Naive: Sum cost of in-situ jobs using request-level job cost prediction***
- ***SAL: Factor in memory reclamation rate*** [Kossman et al 2025]

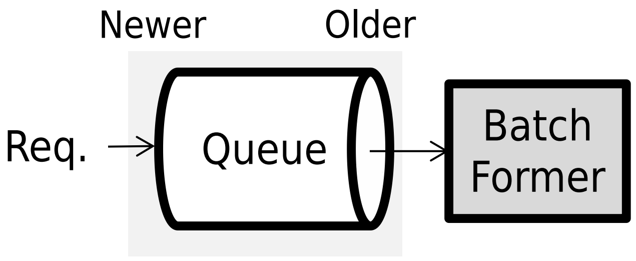
$$load(s,r) = \max(\beta * (memory(r) - free\_mem(s)), \; queued\_tokens(s,r) / max\_tokens\_per\_batch)$$

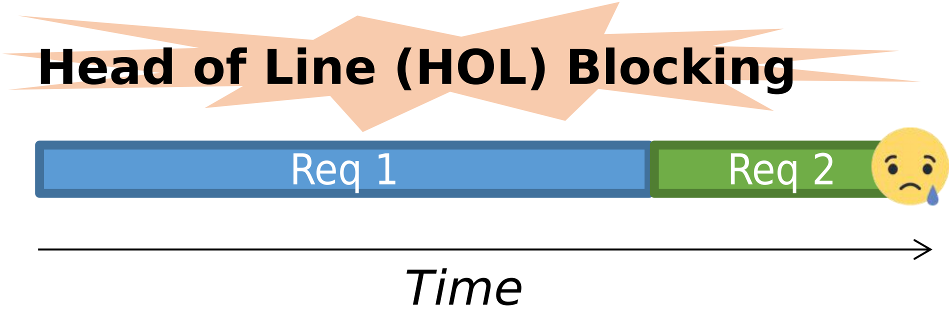


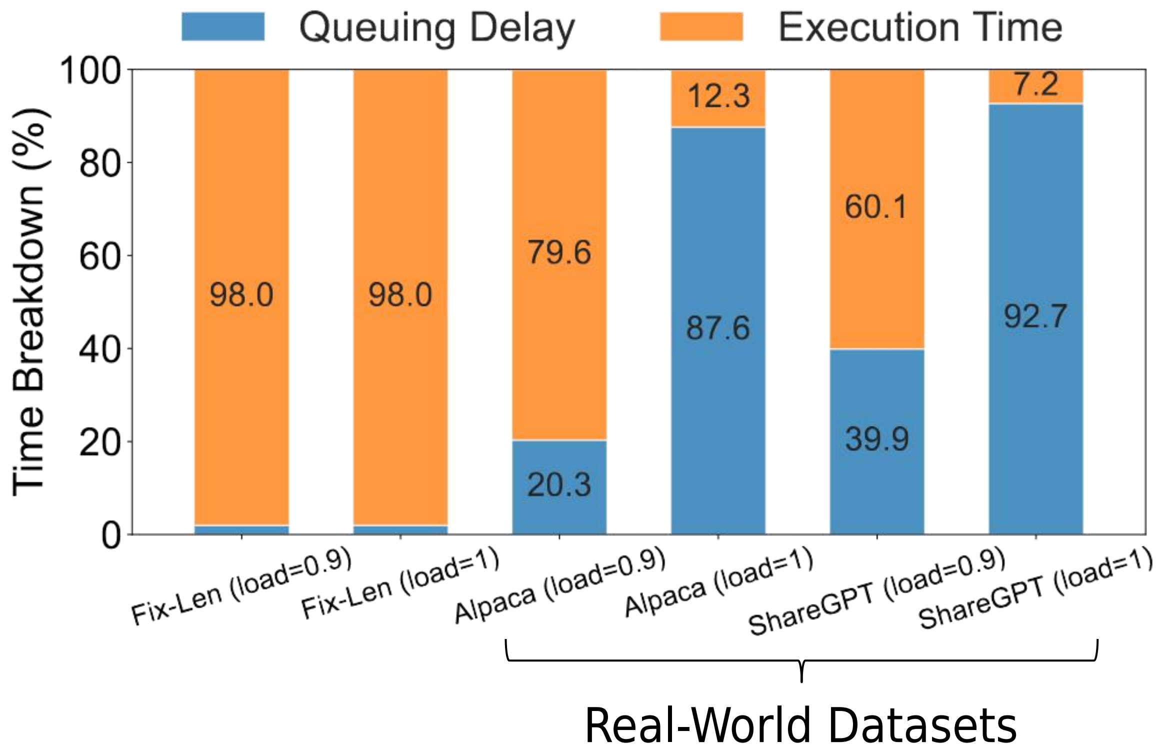
Scheduler: Job Prioritization

Job Prioritization: How to prioritize jobs to minimize queuing time?

- ***First-Come First-Serve (FCFS): Process requests in order of arrival***





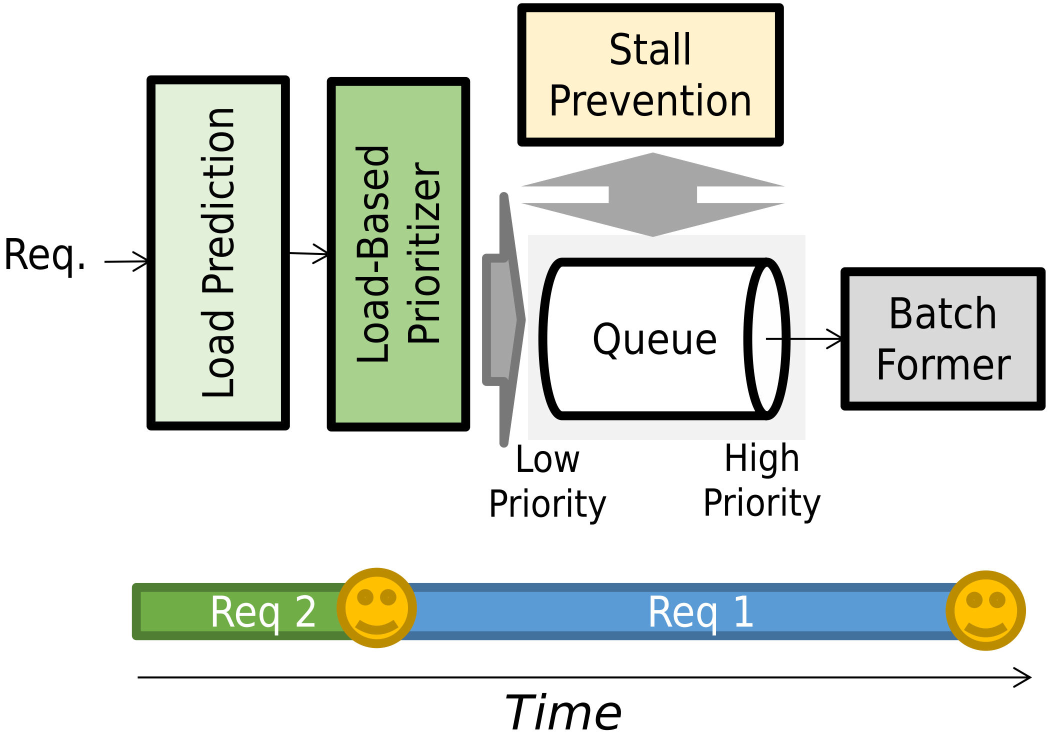


Wu B., Zhong Y., Zhang Z., Liu S., Liu F., Sun Y., Huang G., Liu X., Jin X. *Fast Distributed Inference Serving for Large Language Models. arXiv:2305.05920*

Scheduler: Job Prioritization

Job Prioritization: How to prioritize jobs to minimize queuing time?

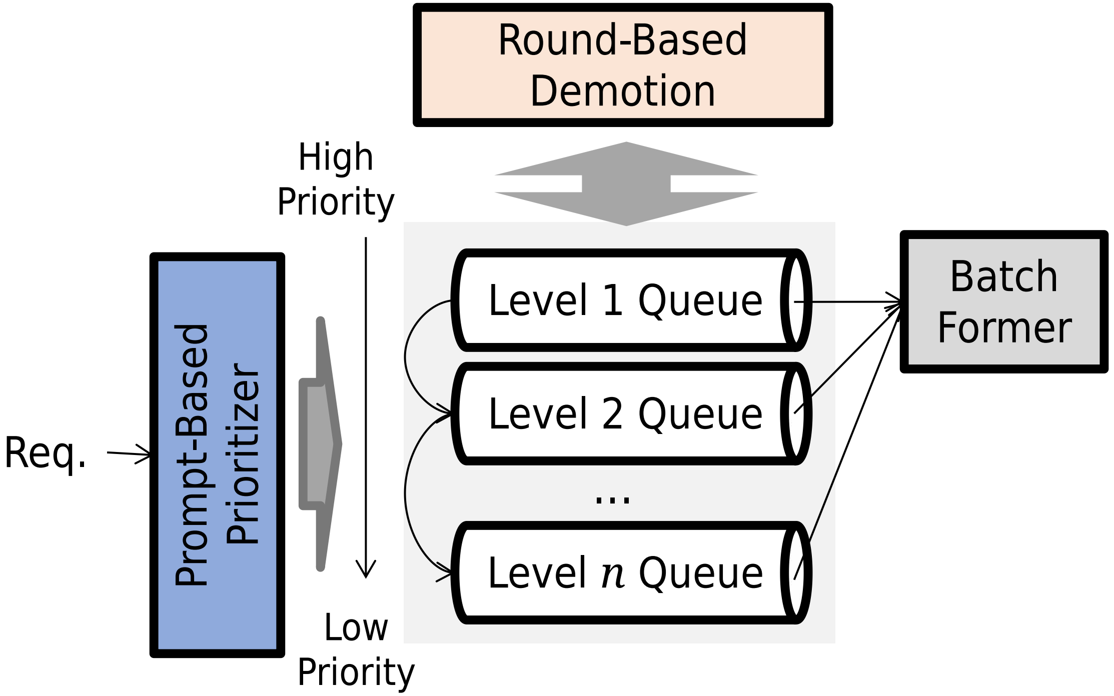
- ***Shortest-Job First (SJF): Process requests in order of remaining time***
  - Guarantees minimum average latency (incl. queuing time) but requires accurate completion time pred.
  - Preemptive SJF:
    - Can lead to **stalls** for perpetually low-priority requests
    - **Context-switch cost** (offloading / evicting in-situ cache + reloading the cache upon resumption)

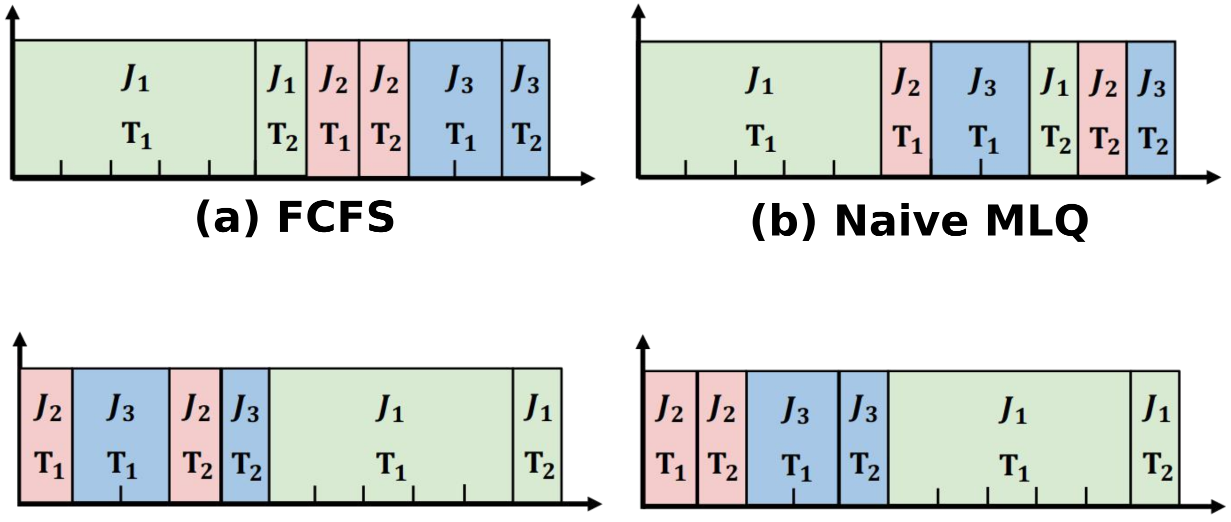


Scheduler: Job Prioritization

Job Prioritization: How to prioritize jobs to minimize queuing time?

- ***Multi-Level Queue (MLQ): Gradually demote requests to simulate SJF***
  - **Naive MLQ**: place all new jobs in highest priority queue, then gradually demote
  - **Skip-Join MLQ**: place all new jobs in queue based on prefix length





(a) FCFS

(b) Naive MLQ

(c) Skip-Join MLQ

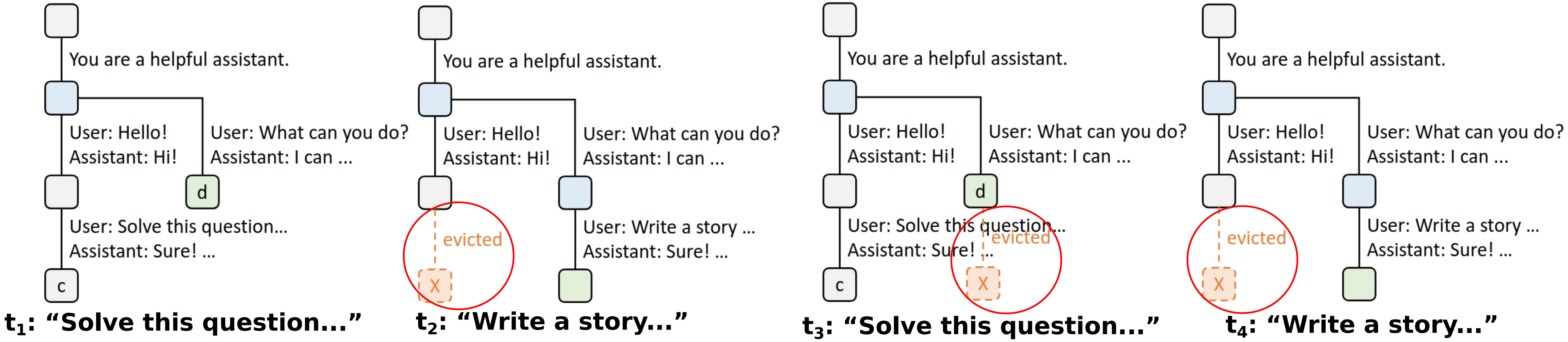
(d) Shortest Remaining Processing Time (Optimal)

Wu B., Zhong Y., Zhang Z., Liu S., Liu F., Sun Y., Huang G., Liu X., Jin X. *Fast Distributed Inference Serving for Large Language Models.* *arXiv:2305.05920*

Scheduler: Job Prioritization

Job Prioritization: How to prioritize jobs to minimize queuing time?

- ***Maximum Cache Hits: Process requests based on cache hits***
  - Simulates SJF since large cache hit could mean low job cost
  - Avoids cache thrashing



t_1: "Solve this question..." **t_2: "Write a story..."** **t_3: "Solve this question..."** **t_4: "Write a story..."**

Zheng L., Yin L., Xie Z., Sun C., Huang J., Yu CH., Cao S., Kozyrakis C., Stoica I., Gonzalez JE., Barrett C., Sheng Y. *SGLang: Efficient Execution of Structured Language Model Programs*. arXiv:2312.07104

Scheduler: Job Cost Prediction

Job Cost Prediction: How to measure job cost without knowing final output length?

- ***Ask the LLM: Add output length prediction request to original prompt***

E.g. Perception-in-Advance (PiA):

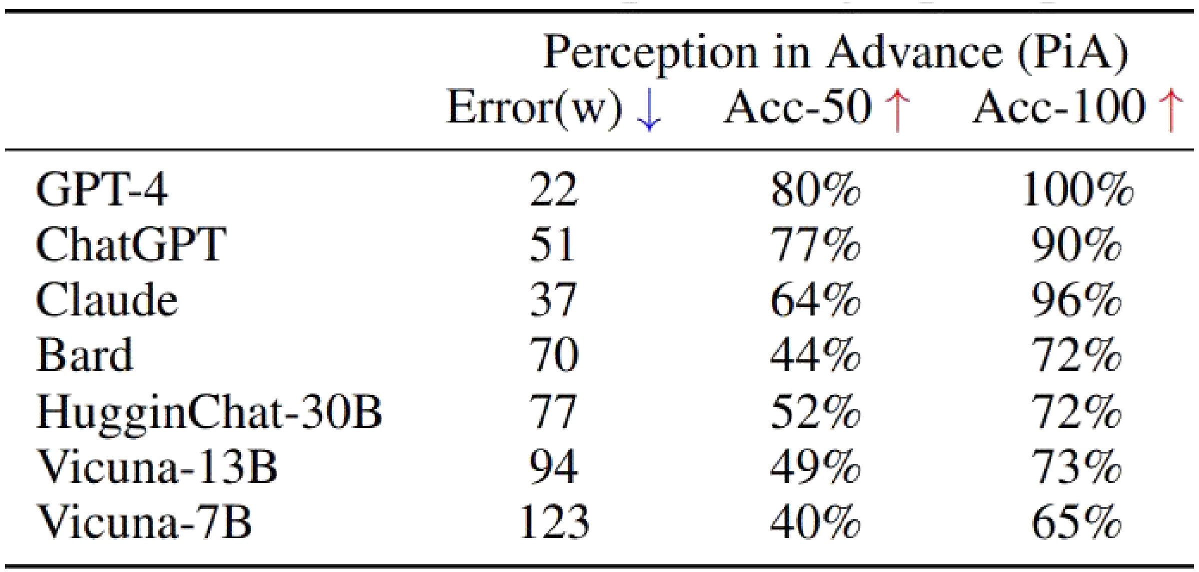
Prompt

Create a fun math question for children. **Before responding to the above instruction, you have to predict the length of your response. Print the estimated number of words in your response in the first line.** Then change to a new line to respond to the instruction.

GPT-4

Estimated response length: 60 words.

Sure, here’s a fun math problem: There are 7 apples in a basket. A friendly squirrel comes and...



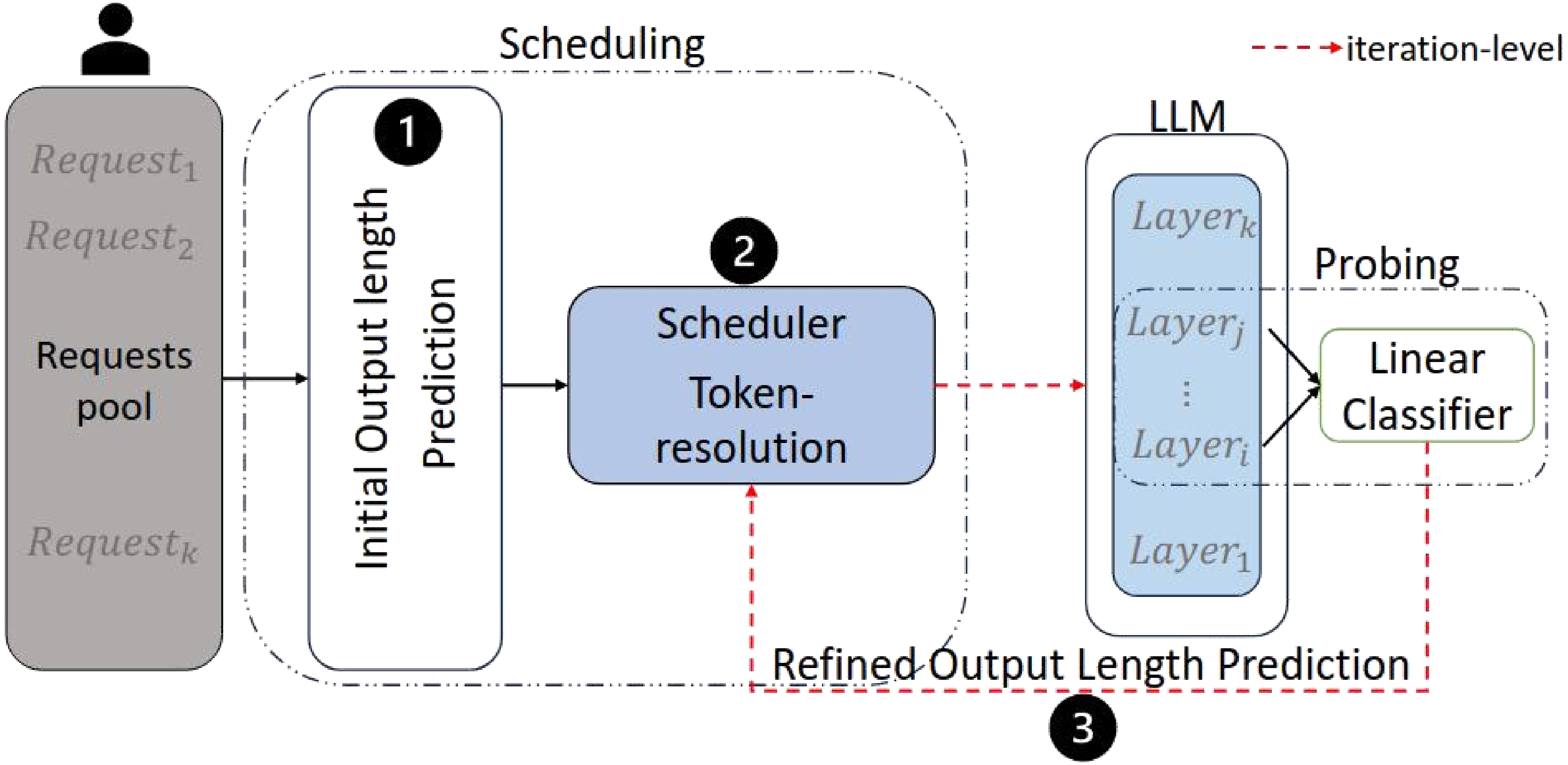
	Perception in Advance (PiA)		
	Error(w) ↓	Acc-50 ↑	Acc-100 ↑
GPT-4	22	80%	100%
ChatGPT	51	77%	90%
Claude	37	64%	96%
Bard	70	44%	72%
HugginChat-30B	77	52%	72%
Vicuna-13B	94	49%	73%
Vicuna-7B	123	40%	65%

Zheng Z., Ren X., Xue F., Luo Y., Jiang X., You Y. *Response Length Perception and Sequence Scheduling: An LLM-Empowered LLM Inference Pipeline. NeurIPS’23*

Scheduler: Job Cost Prediction

Job Cost Prediction: How to measure job cost without knowing final output length?

- ***Train an Estimator: Use separate estimator to predict output length***



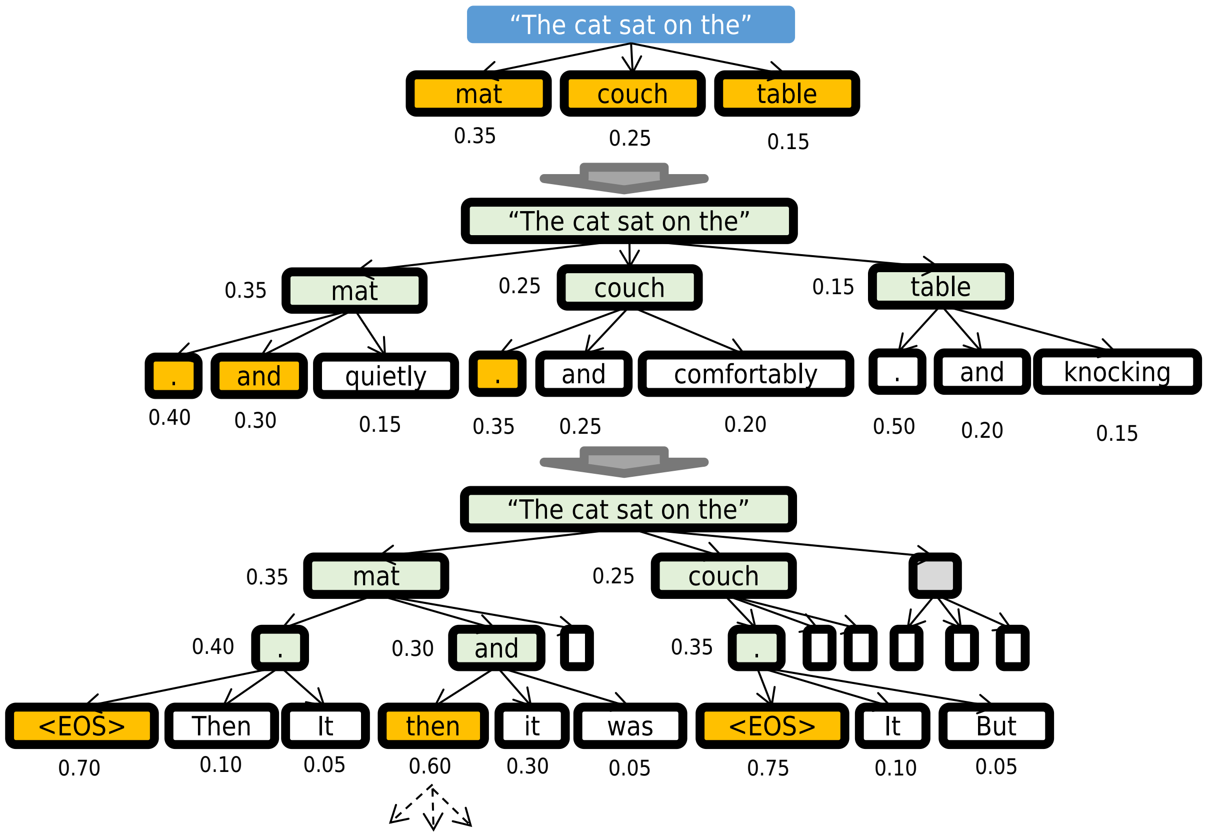
Shahout R., Malach E., Liu C., Jiang W., Yu M., Mitzenmacher M. *Don't Stop Me Now: Embedding Based Scheduling for LLMs.* *arXiv:2410.01035*

Scheduler: Job Cost Prediction

Job Cost Prediction: How to measure job cost without knowing final output length?

- ***Certaindex**: Use beam consistency as heuristic for remaining job time*

Beam Search (k > 1, e.g. k = 3)



Group beams into m clusters based on similarity

Measure cluster entropy using size of each cluster $|C_i|$ relative to number of beams, n

$$\mathcal{H} = -\sum_{i=1}^{m} \frac{|C_i|}{n} \log \frac{|C_i|}{n}$$

$$\tilde{\mathcal{H}} = \frac{\log n - \mathcal{H}}{\log n} \in [0, 1]$$

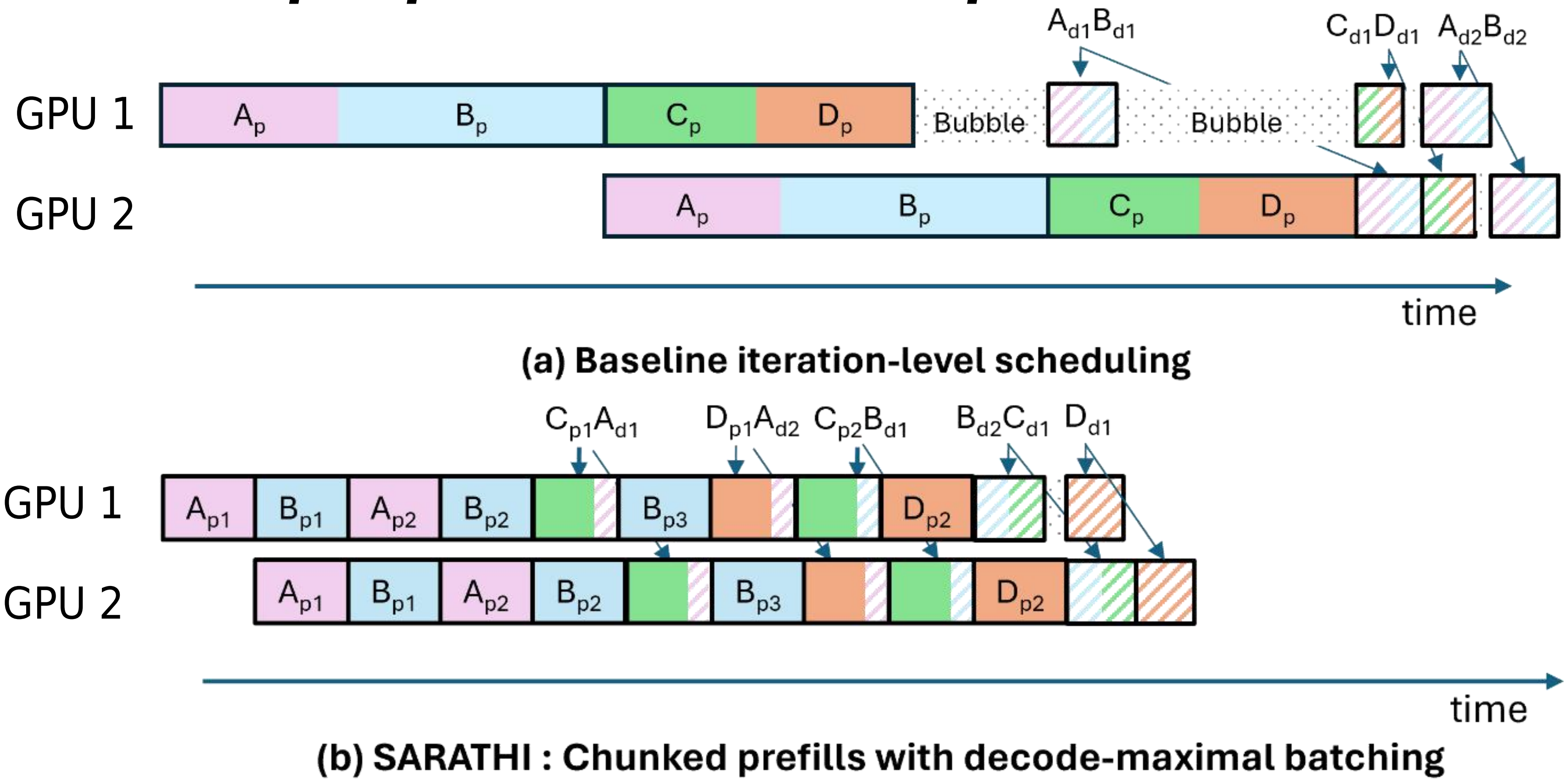
Normalize to yield a score between [0, 1]

Fu Y., Chen J., Zhu S., Fu Z., Dai Z., Zhuang Y., Ma Y., Qiao A., Rosing T., Stoica I., Zhang H. *Efficiently Scaling LLM Reasoning with Certaindex. arXiv:2412.20993*

Batch Controller: Prefix Chunking

Batch Controller: How to compose the batch to balance throughput and latency?

- ***Chunked Prefills: Split prefill across multiple rounds***



(a) Baseline iteration-level scheduling

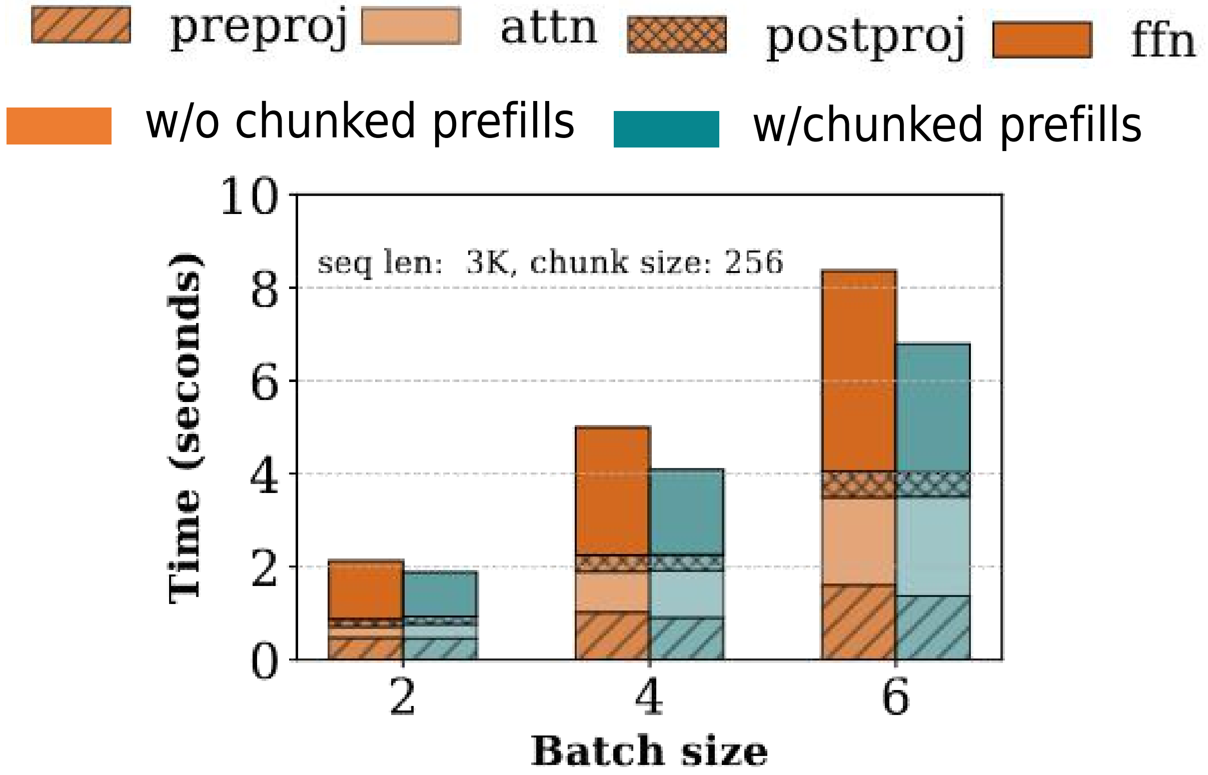
(b) SARATHI : Chunked prefills with decode-maximal batching

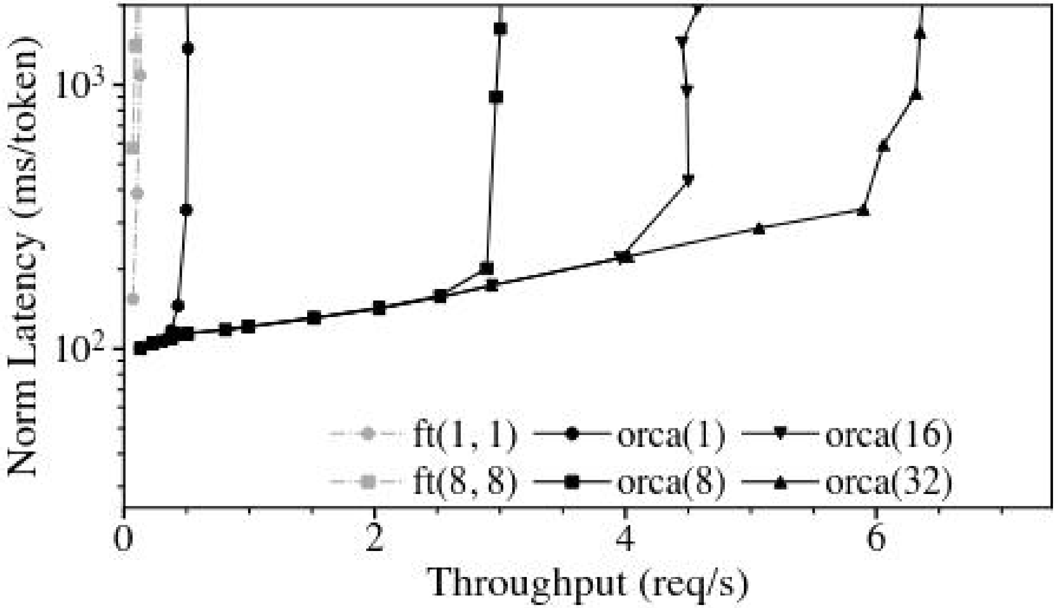
Agrawal, A, Panwar, A, Mohan, J, Kwatra, N, Gulavani, BS, Ramjee, R. *SARATHI: Efficient LLM Inference by Piggybacking Decodes with Chunked Prefills.* *arXiv:2308.16369*

Batch Controller: Batch Sizing

Batch Controller: How to compose the batch to balance throughput and latency?

- ***Batch Sizing**: Inc. batch size to raise throughput & dec. to lower latency*





Agrawal, A, Panwar, A, Mohan, J, Kwatra, N, Gulavani, BS, Ramjee, R. *SARATHI: Efficient LLM Inference by Piggybacking Decodes with Chunked Prefills. arXiv:2308.16369*

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

Scheduler: Summary

Minimize queuing delays and maximize resource utilization by balancing the load

Scheduler	Technique Classification	Latency	Throughput	Memory	Quality
Load Balancer					
• Job Assignment					
• Greedy	Algorithm				
• Power-of-2	Algorithm	↓	↑		
• Load Prediction (SAL)	Model (Heuristic)	↓	↑		
Scheduler					
• Job Prioritizer					
• First-Come First-Serve	Algorithm				
• Shortest-Job	Algorithm	↓	↑		
• Multi-Level Queue	Algorithm	↓	↑		
• Job Cost Prediction					
• Cache / Prompt Based	Model (Heuristic)	↓	↑		
• Learning-Based	Model (Learned)	↓	↑	↑	
Batch Controller					
• Chunking Module	Optimization	↓	↑		
• Batch Size Control	Optimization	↓	↑		

Part 4: Storage Manager

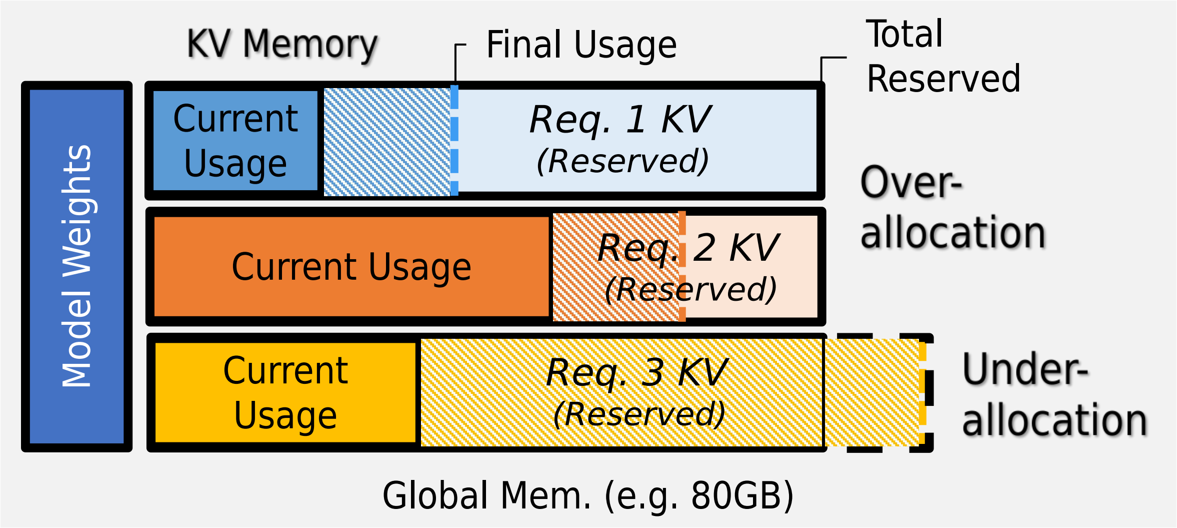
Efficiently store KV caches to minimize wasted memory; reduce memory usage via compression

Storage Manager	Technique Classification	Technique Description / Key Idea
Block Manager		
• Block Storage (Paged)	Framework	• Dynamic block-based memory allocation
• Block Sharing & Eviction		
• Prefix Sharing	Optimization	
• Partial Reconstruction	Optimization	• Reconstruct KV vectors for imperfect matches
• Long Context Eviction	Optimization	• Reduce memory by discarding unimportant KVs
• Block Search & Retrieval		
• Radix Tree	Index	• Organize blocks by prefix to support efficient search
Physical Storage		
• Tiered Storage & Offloading	Framework	• Increase capacity by exploiting tiered storage
• Distributed Storage	Framework	• Increase capacity by storing across multiple workers
• Hot Blocks	Optimization	• Replicate hot blocks to avoid block transfer
Quantizer		
• Quantizer Design	Operator Design	• Reduce memory by lowering numerical precision
• Outlier Smoothing	Optimization	• Reduce quantization error by smoothing outliers

Block Manager: Block Storage

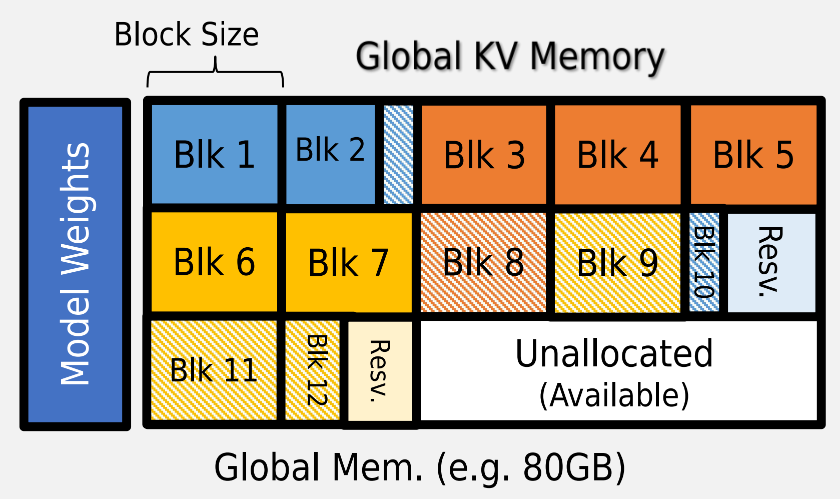
Block Storage: How to allocate memory for tasks with dynamic memory usage?

- ***PagedAtten.*: *Dynamically allocate small blocks managed by block table***
  - **vAttention** [Prabhu et al 2025], **vTensor** [Xu et al 2024 FlexInfer]: use GPU native memory management capabilities to keep track of blocks



(a) Static Allocation

vs.

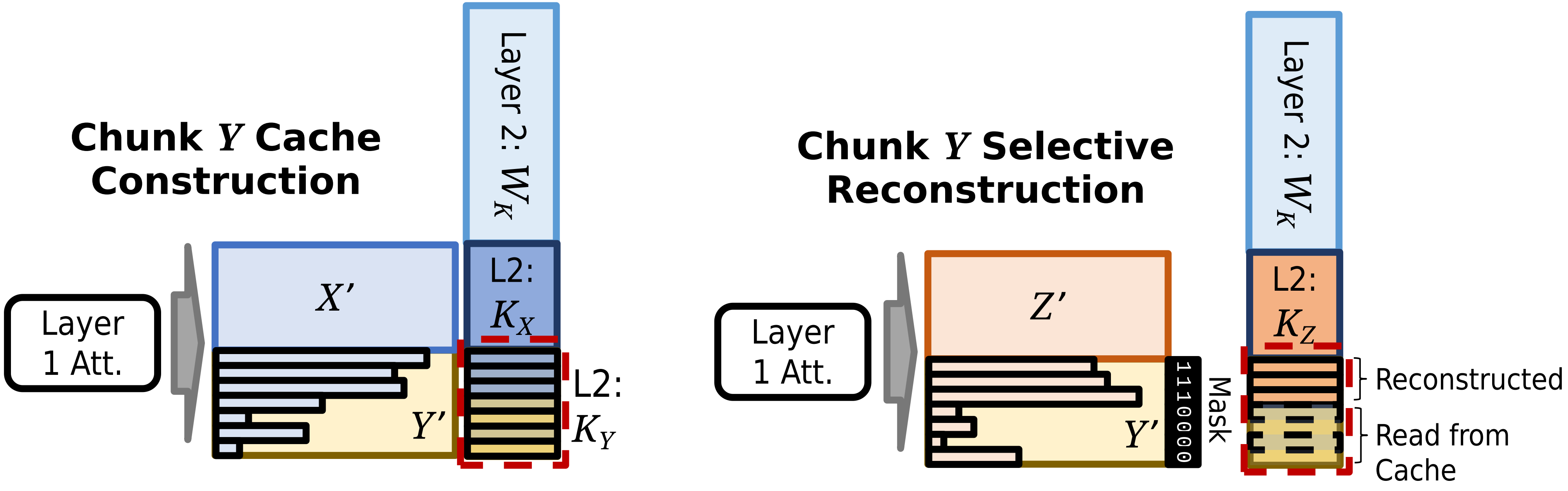


(b) Paged Allocation

Block Manager: Sharing & Eviction

Block Sharing: How to reuse cache blocks when KV vectors are context-dependent?

- Key vectors K_Y for Chunk Y are influenced by value vectors from the prefix X

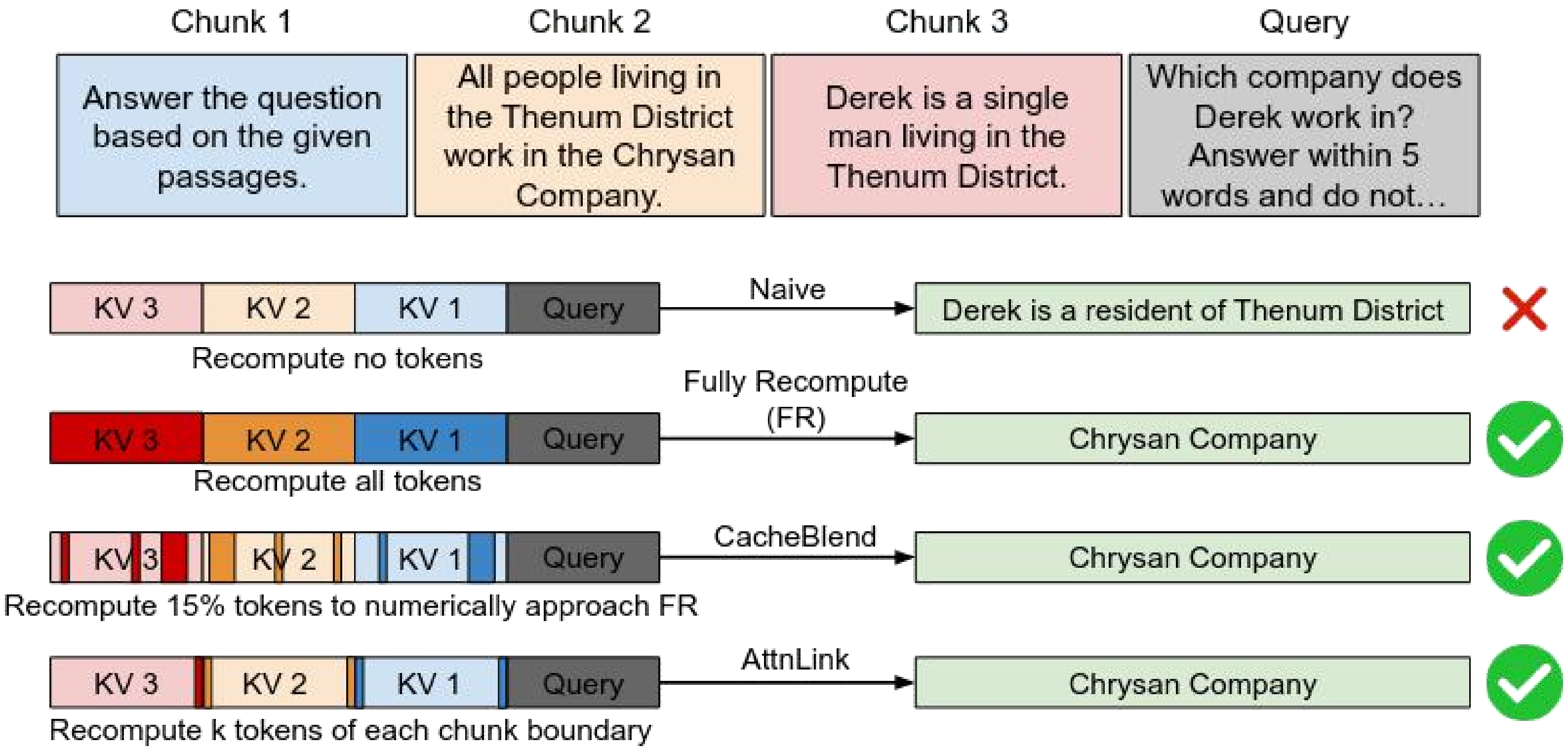


- ***Prefix Sharing*: Reuse up to longest exact-match prefix**
- ***Cache Reconstruction*: Recalculate KV vectors for a few significant tokens**
  - E.g. position-based, template-based, score-based

Block Manager: Sharing & Eviction

Block Sharing: How to reuse cache blocks when KV vectors are context-dependent?

- ***Cache Reconstruction*****: *Recalculate KV vectors for a few significant tokens***
  - **Position-Based** [Hu et al 2024 **Epic**]: Recalculate at fixed positions, e.g. chunk boundaries

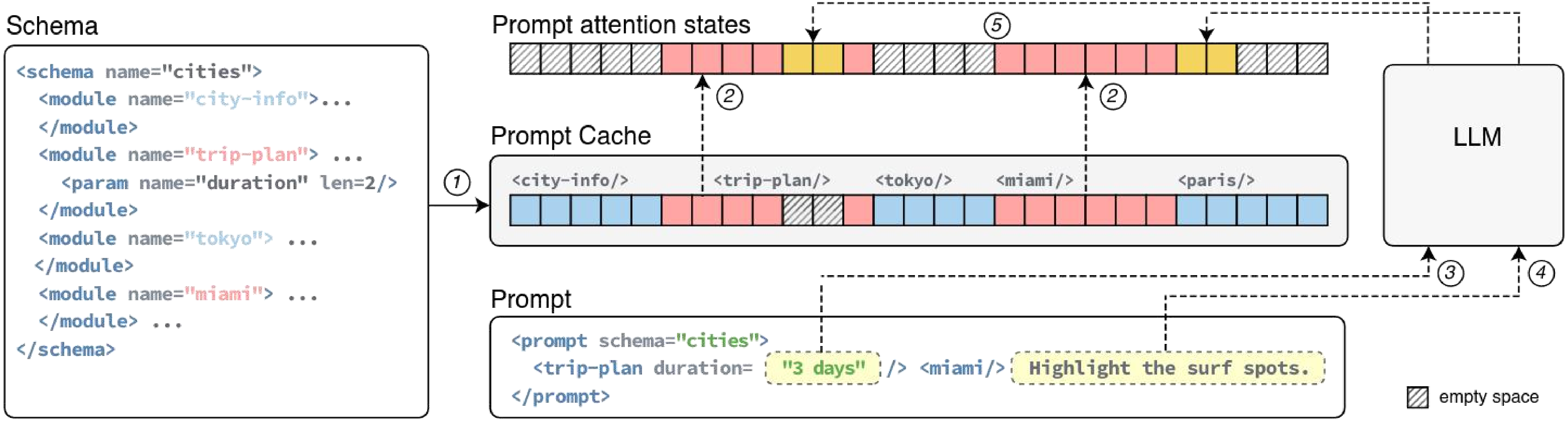

Chunk 1
Answer the question based on the given passages.
Chunk 2
All people living in the Thenum District work in the Chrysan Company.
Chunk 3
Derek is a single man living in the Thenum District.
Query
Which company does Derek work in? Answer within 5 words and do not...
KV 3
KV 2
KV 1
Query
Naive
Derek is a resident of Thenum District
✗
Recompute no tokens
Fully Recompute (FR)
Chrysan Company
✓
Recompute all tokens
CacheBlend
Recompute 15% tokens to numerically approach FR
AttnLink
Recompute k tokens of each chunk boundary

Hu J., Huang W., Wang H., Wang W., Hu T., Zhang Q., Feng H., Chen X., Shan Y., Xie T. *EPIC: Efficient Position-Independent Caching for Serving Large Language Models. arXiv:2410.15332*

Block Manager: Sharing & Eviction

Block Sharing: How to reuse cache blocks when KV vectors are context-dependent?

- ***Cache Reconstruction*: *Recalculate KV vectors for a few significant tokens***
  - **Template-Based** [Gim et al 2024 **Prompt Cache**]: Recalculate only the “parameter” tokens of a template



```
<schema name="cities">
  <module name="city-info">...
  </module>
  <module name="trip-plan"> ...
    <param name="duration" len=2/>
  </module>
  <module name="tokyo"> ...
 </module>
  <module name="miami"> ...
  </module> ...
</schema>
```

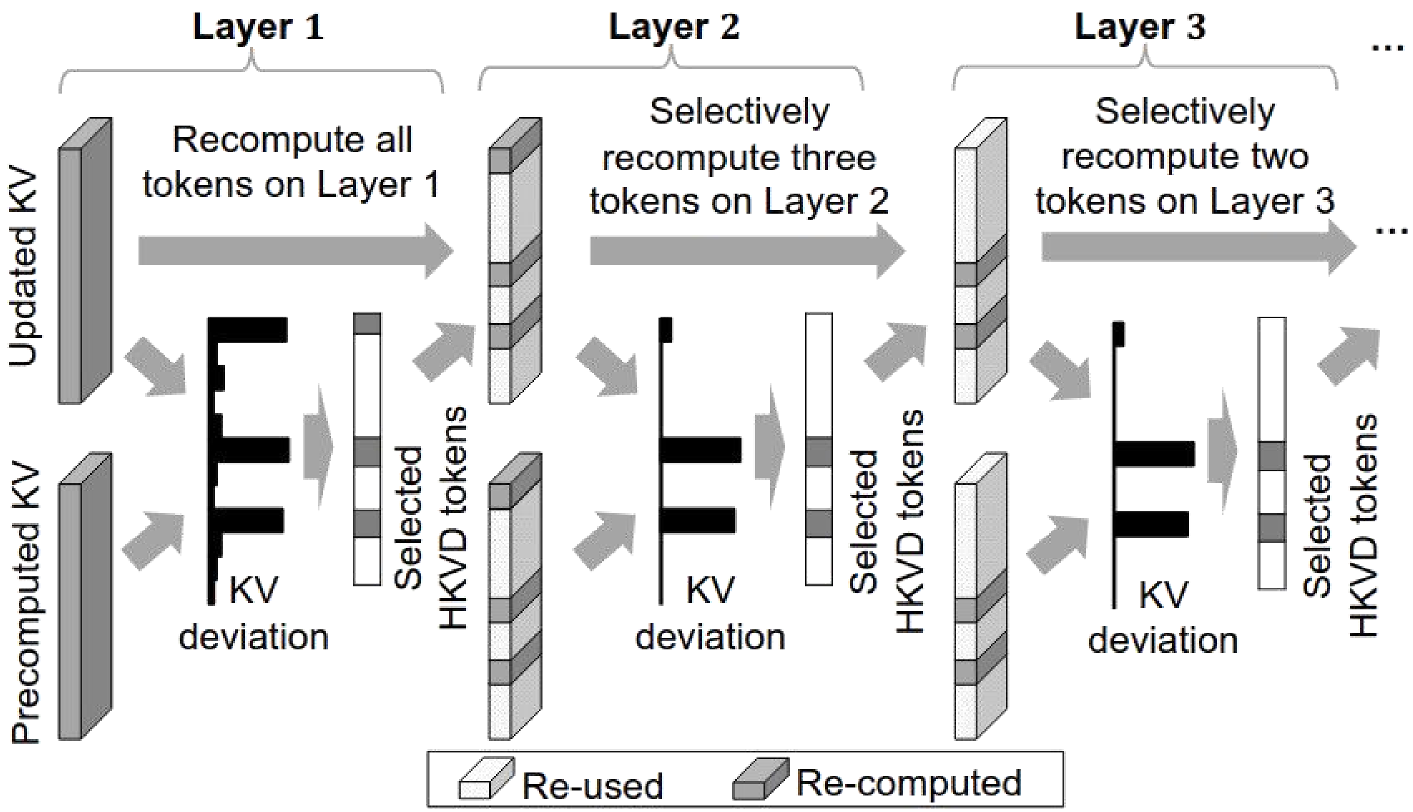
```
<prompt schema="cities">
  <trip-plan duration= "3 days" /> <miami/> Highlight the surf spots.
</prompt>
```

Gim I., Chen G., Lee S., Sarda N., Khandelwal A., Zhong L. *Prompt Cache: Modular Attention Reuse for Low-Latency Inference. arXiv:2311.04934*

Block Manager: Sharing & Eviction

Block Sharing: How to reuse cache blocks when KV vectors are context-dependent?

- ***Cache Reconstruction*: Recalculate KV vectors for a few significant tokens**
  - **Score-Based** [Yao et al 2024 **CacheBlend**]: Identify significant tokens based on attention score deviation

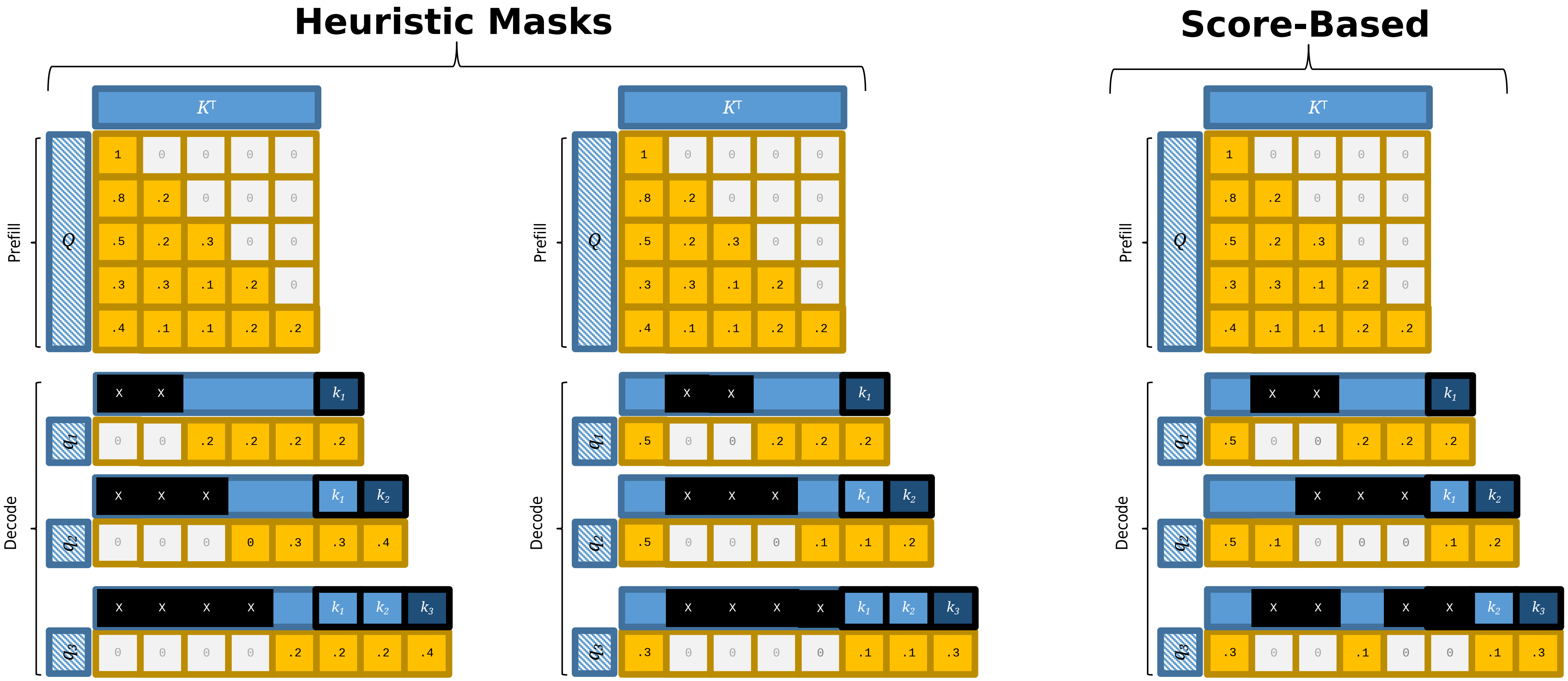


Yao J., Li H., Liu Y., Ray S., Cheng Y., Zhang Q., Du K., Lu S., Jiang J. *CacheBlend: Fast Large Language Model Serving for RAG with Cached Knowledge Fusion.* arXiv:2405.16444

Block Manager: Sharing & Eviction

Block Eviction (Long Context): How to reduce cache size without reducing quality?

- ***Sparse Attention**: Compute QK similarities for small subset of tokens*



(a) Sliding Window
e.g. LongFormer

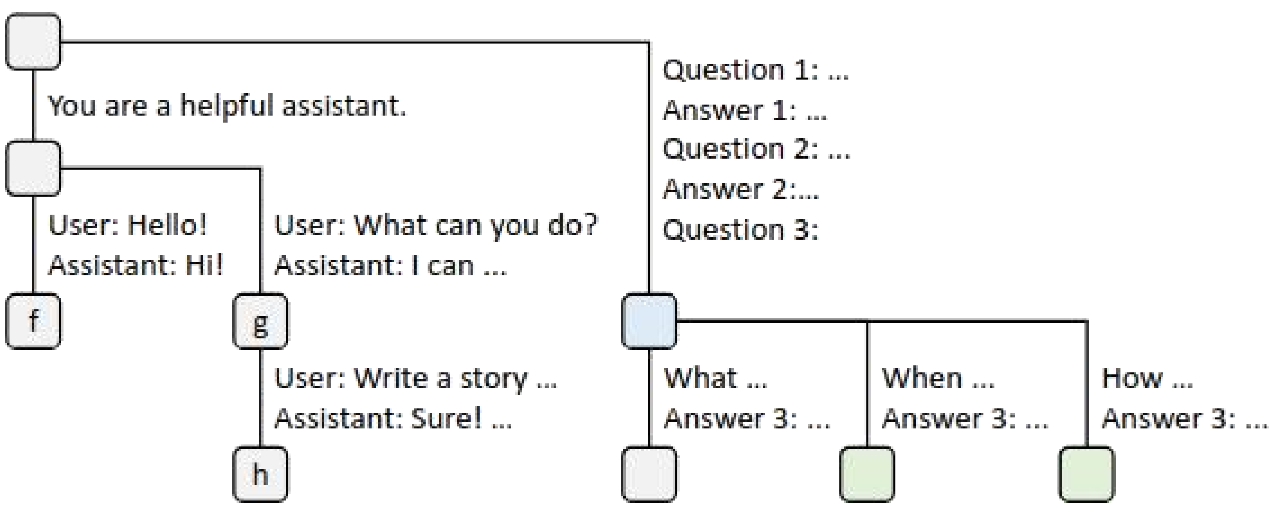
(b) Attention Sink
e.g. StreamingLLM

(c) Least-Score
e.g. TOVA, Keyformer, H2O

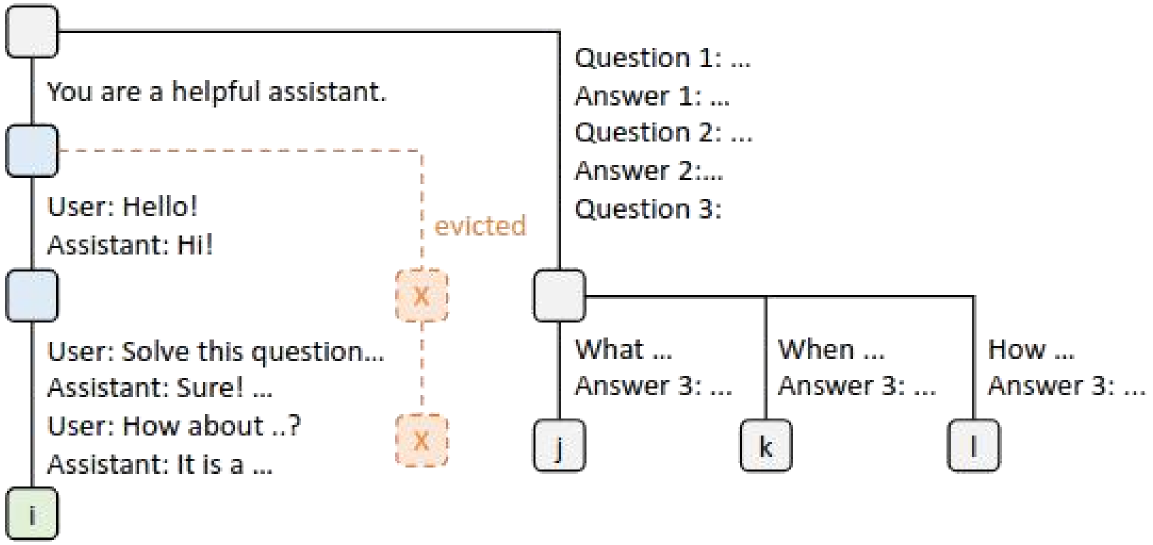
Block Manager: Block Search & Retriev.

Block Search & Retrieval: How to find and retrieve reusable blocks from a persisted cache?

- ***Radix Tree**: Split persisted prefixes along shared prefix branches*



(a) Each branch stores a matchable prefix



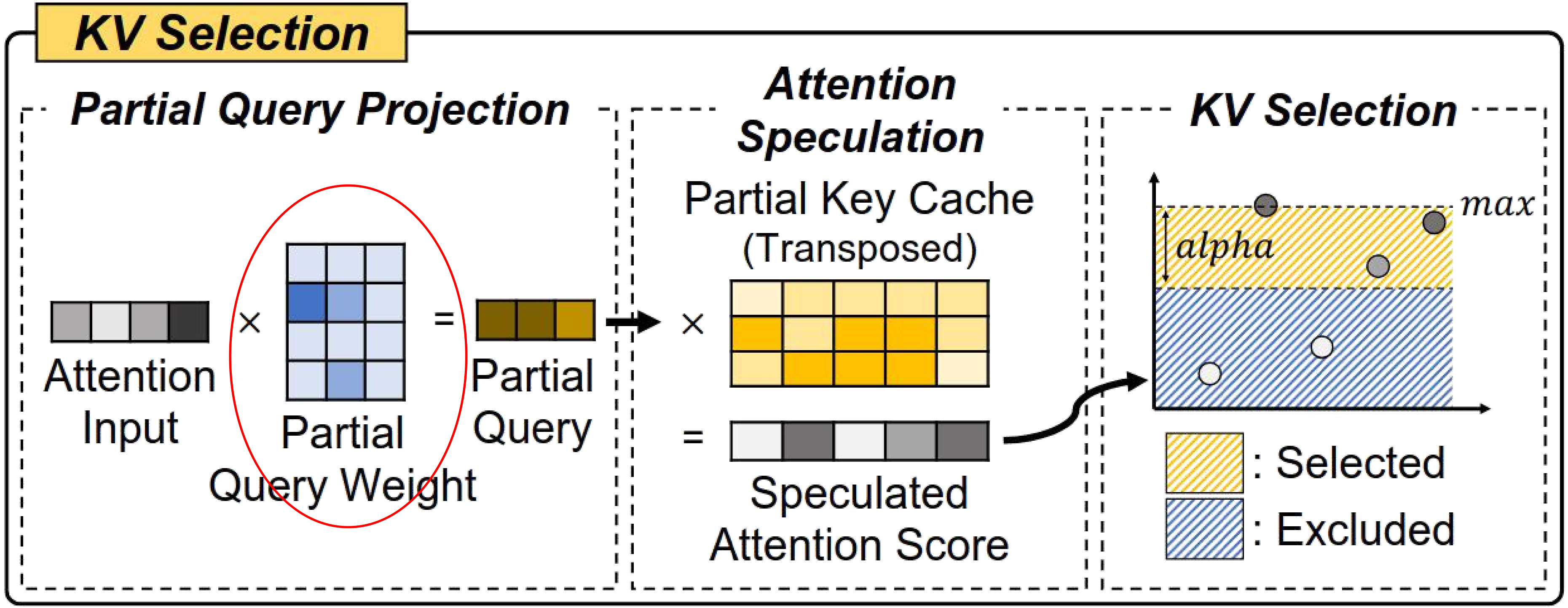
(b) To keep cache size under control, whole least-used branches can be evicted as the tree grows

Zheng L., Yin L., Xie Z., Sun C., Huang J., Yu CH., Cao S., Kozyrakis C., Stoica I., Gonzalez JE., Barrett C., Sheng Y. *SGLang: Efficient Execution of Structured Language Model Programs*. arXiv:2312.07104

Physical Storage: Tiered & Offloading

Cache Offloading (Long Context): How to simultaneously reduce memory and reload costs?

- ***Entry-Wise*: *Store cache on cold storage and load significant tokens only***
  - Partial Query Weight: Modified W_q that returns truncated query vector with few “significant” dims.
  - Partial Key Cache: Key vectors truncated to few “significant” dims.

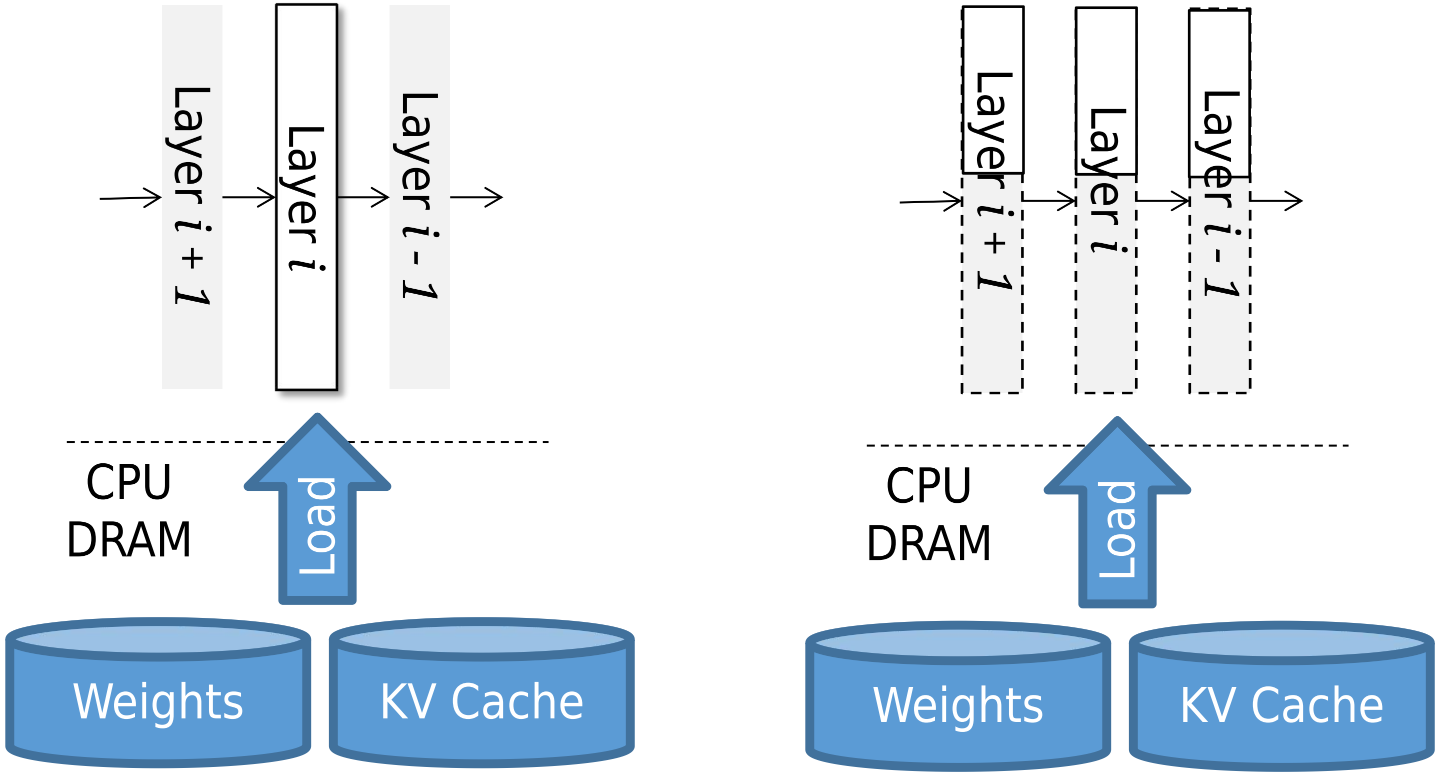


Lee W., Lee J., Seo J., and Sim J. *InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management. OSDI’24*

Physical Storage: Tiered & Offloading

Cache Offloading (Long Context): How to simultaneously reduce memory and reload costs?

- ***Layer/Model-Wise:* Store % of model/layers across tiered storage**
  - FlexGen: Define a cost model and minimize via LP formulation
    - Considerations: read/write costs, CPU-side computation

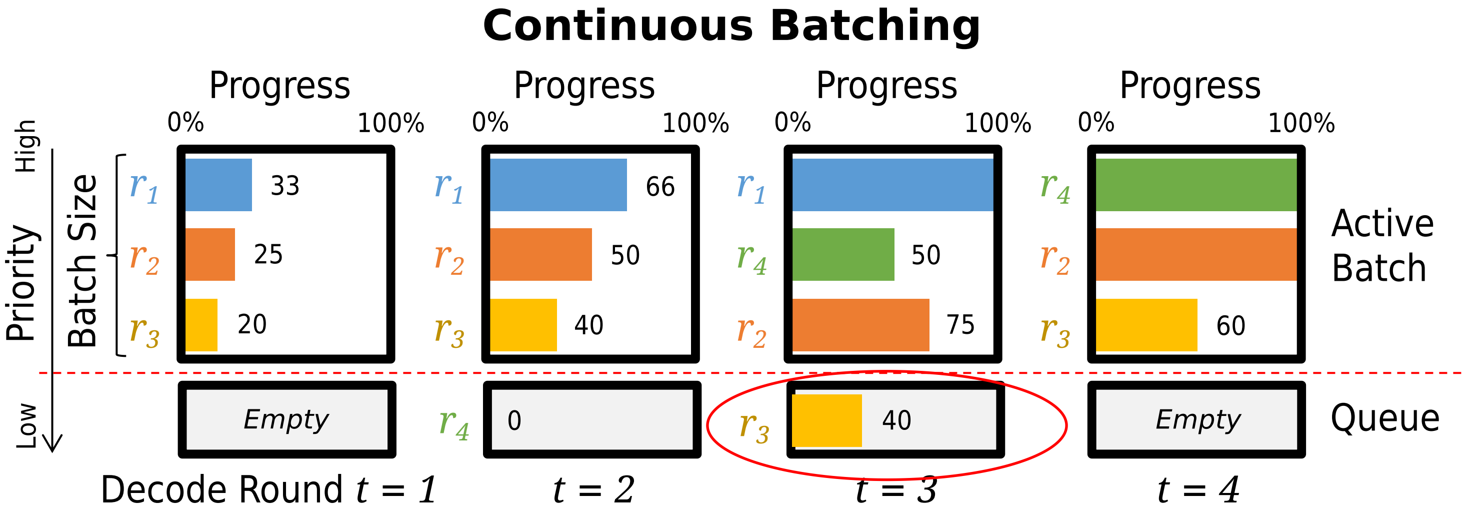


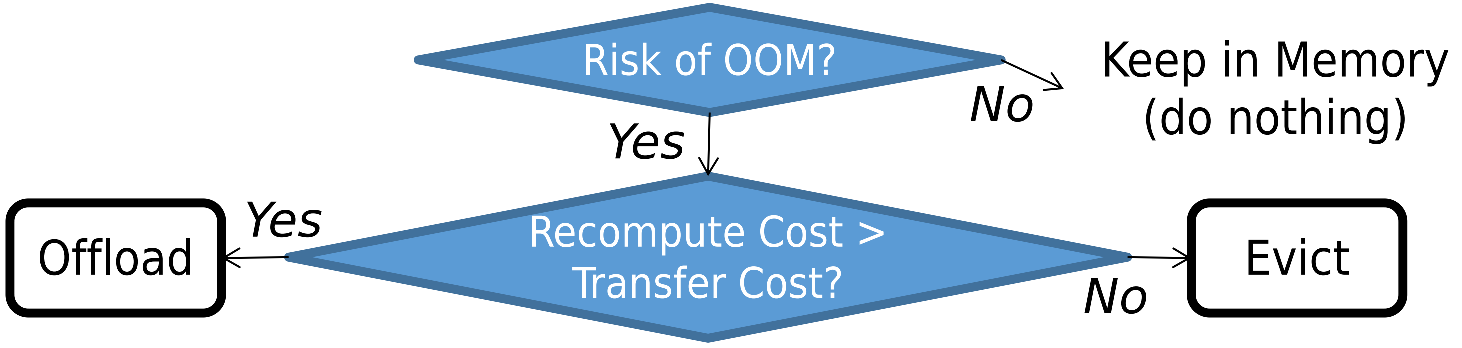
(a) Model-Wise **(b) Layer-Wise**

Physical Storage: Tiered & Offloading

Cache Offloading (Preemption): For preempted requests, when to evict and when to offload?

- ***Cost-Aware Preemption: Use resumption cost to decide evict or offload***





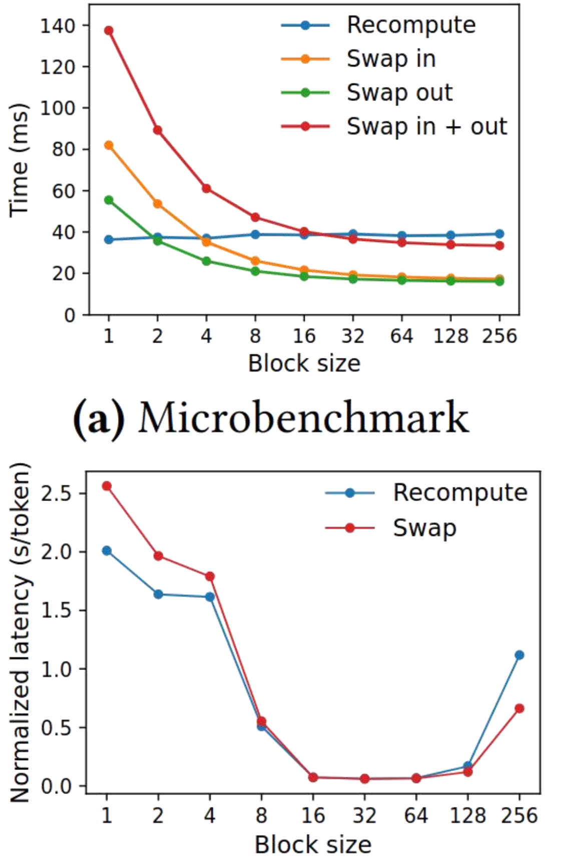
Risk of OOM?

No → Keep in Memory (do nothing)

Yes → Recompute Cost > Transfer Cost?

Yes → Offload

No → Evict



(a) Microbenchmark

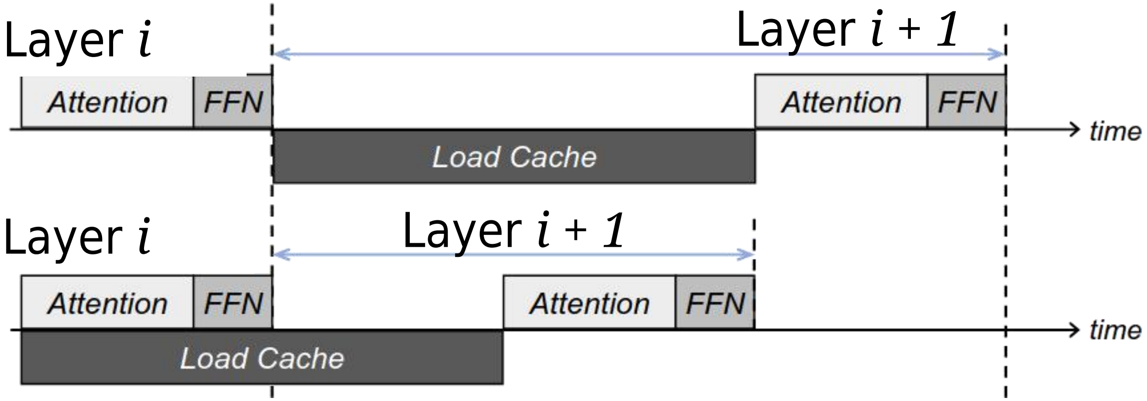
(b) End-to-end performance

Kwon W., Li Z., Zhuang S., Sheng Y., Zheng L., Yu C. H., Gonzalez J. E., Zhang H., Stoica I. *Efficient Memory Management for Large Language Model Serving with PagedAttention. arXiv:2309.06180*

Physical Storage: Tiered & Offloading

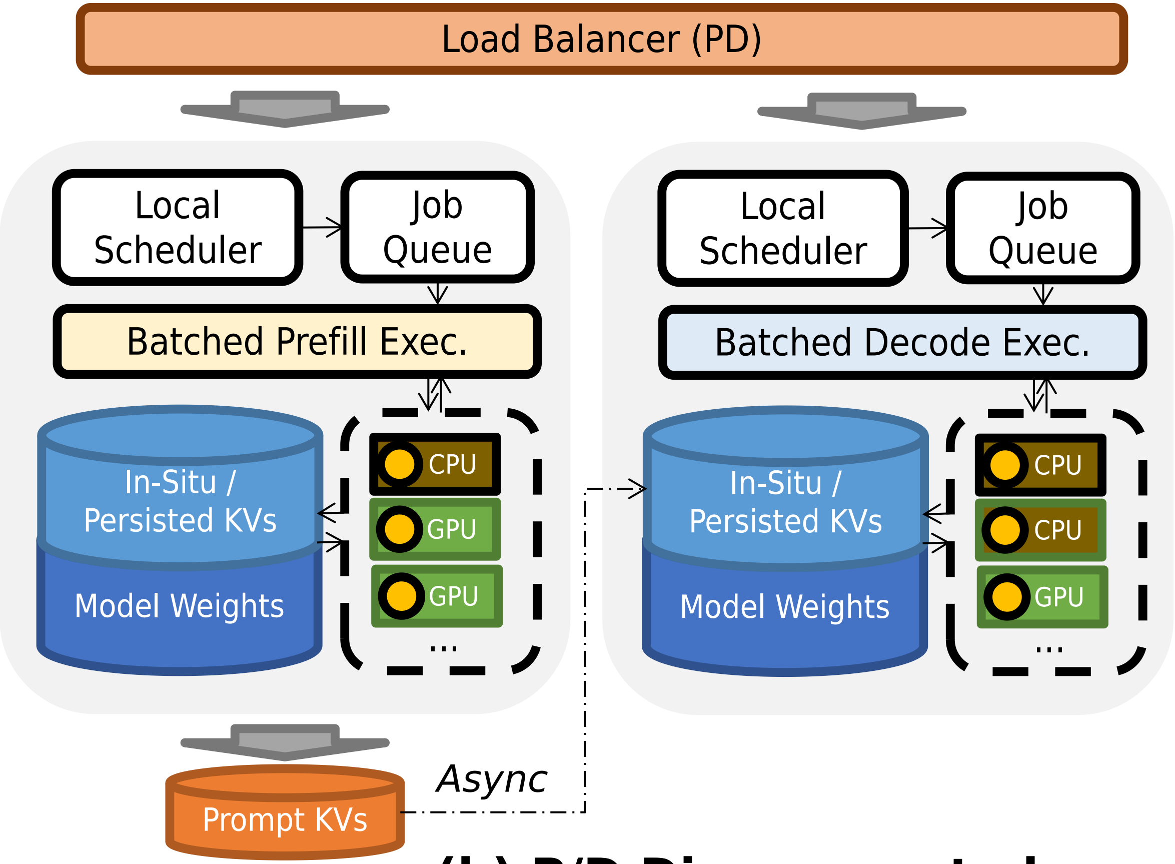
Cache Offloading (Preemption): For preempted requests, when to evict and when to offload?

- ***Async Recovery**: Prefetch Layer $i + 1$ during computation of Layer i*
- ***Disaggregated Async Transfer**: Stream cache from prefill to decode*



(a) Async Recovery/Onloading

Lee W., Lee J., Seo J., and Sim J. *InfiniGen: Efficient Generative Inference of Large Language Models with Dynamic KV Cache Management. OSDI'24*

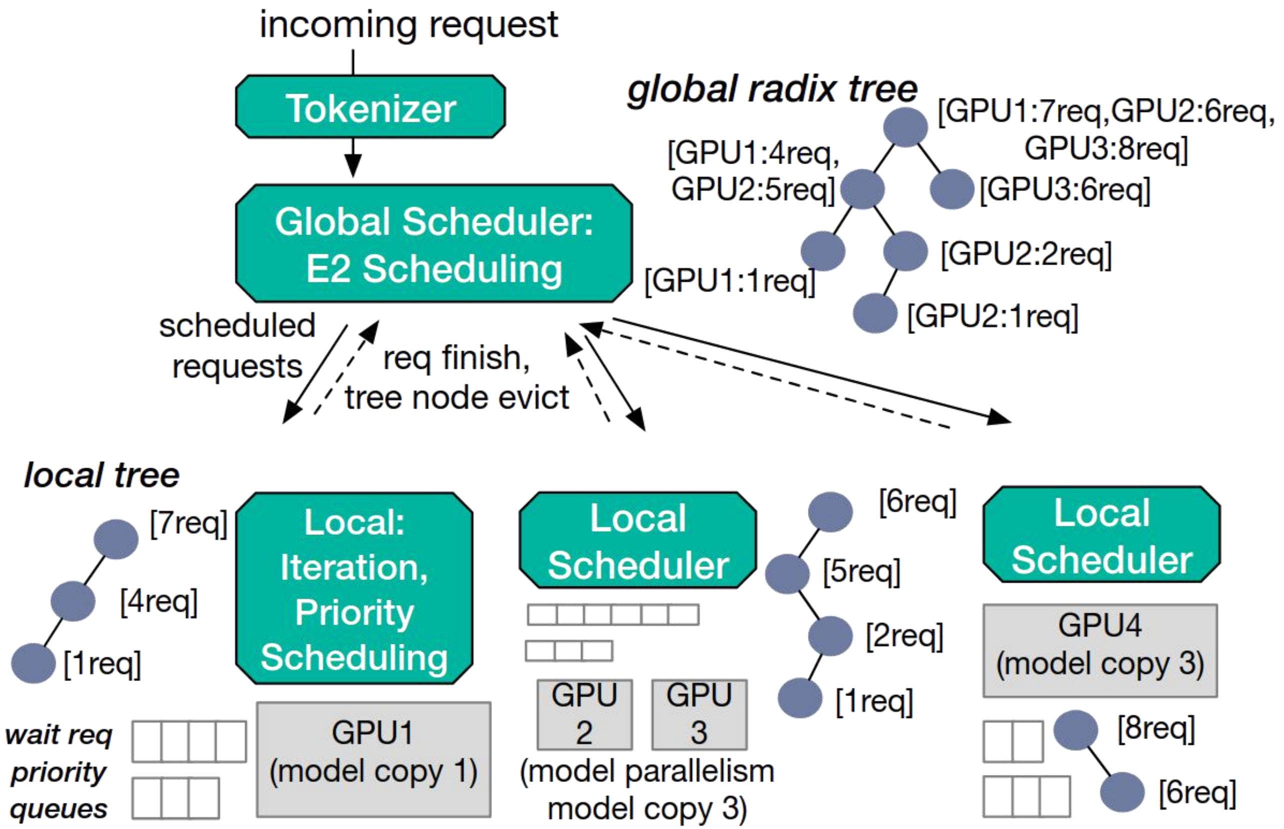


(b) P/D Disaggregated (Asynchronous)

Physical Storage: Distributed Cache

Distributed Cache: How to partition blocks to workers to balance the workload & reduce transfers?

- ***Cache-Aware Load Balancing*: Assign jobs based on cache hits**
  - **Preble**: Use distributed radix tree to search matching blocks

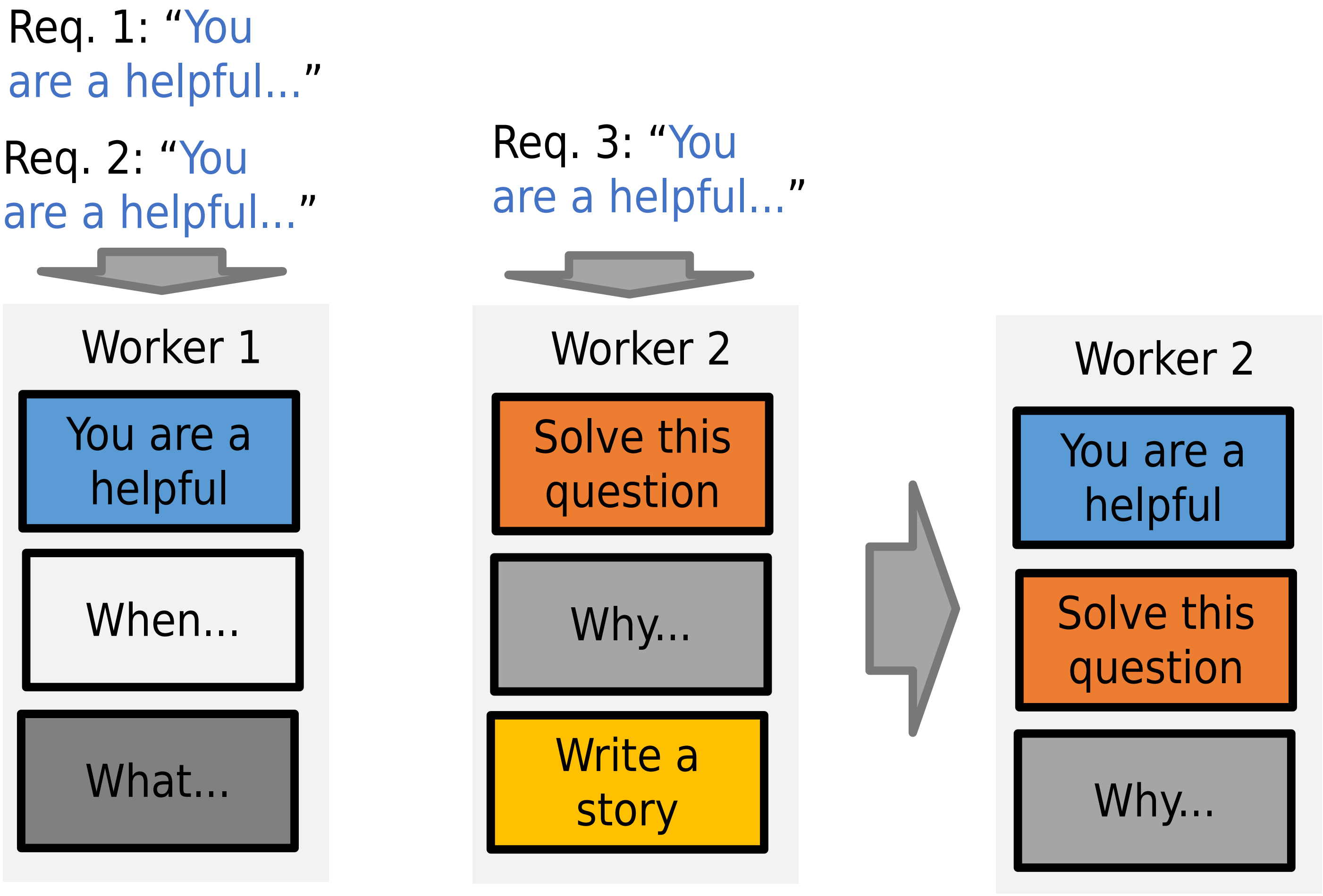


Srivatsa V., He Z., Abhyankar R., Li D., Zhang Y. *Preble: Efficient Distributed Prompt Scheduling for LLM Serving. arXiv:2407.00023*

Physical Storage: Distributed Cache

Distributed Cache: How to partition blocks to workers to balance the workload & reduce transfers?

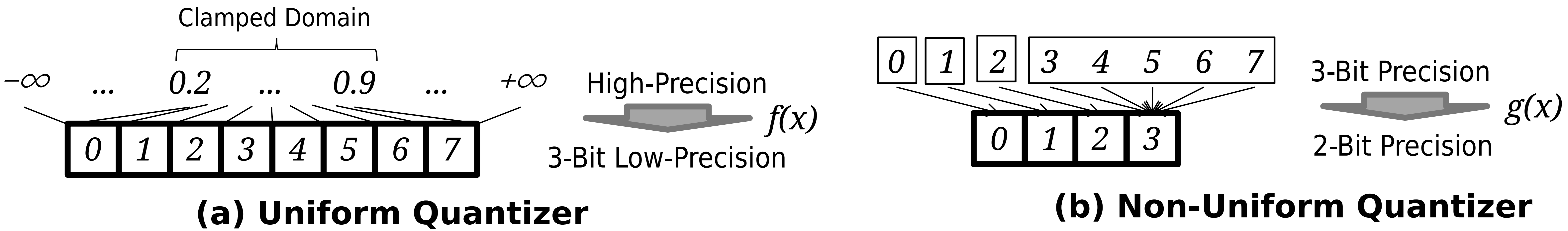
- ***Hot Blocks: Store hot block replicas on multiple workers***
  - **Mooncake**: To replicate blocks "naturally", occasionally assign requests while ignoring worker blocks



Quantization: Quantizer Design

Quantizer Design: How to find error-minimizing map from high to low-precision domain?

- ***Uniform*: Discretize a high-precision domain into low-bit numbers**
  - E.g. $q(x) = \lfloor x/s \rfloor + z$ where s is a step size and z is offset
- ***Non-Uniform*: Directly solve for error minimization mapping**
  - E.g. k-means clustering



(a) Uniform Quantizer

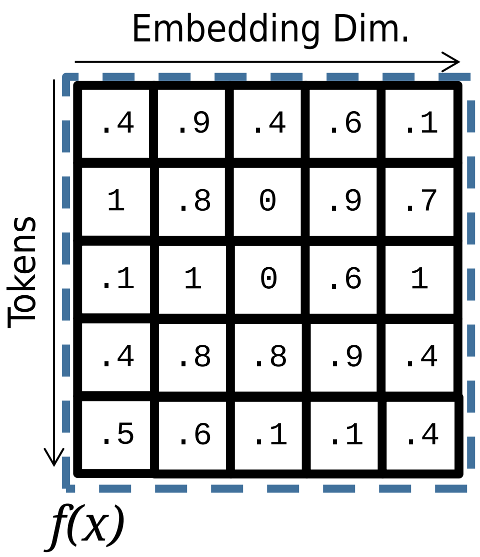
(b) Non-Uniform Quantizer

Survey: **Gholami A.**, Kim S., Dong Z., Yao Z., Mahoney M. W., Keutzer K. *A Survey of Quantization Methods for Efficient Neural Network Inference.* *arXiv:2103.13630*

Quantization: Quantizer Design

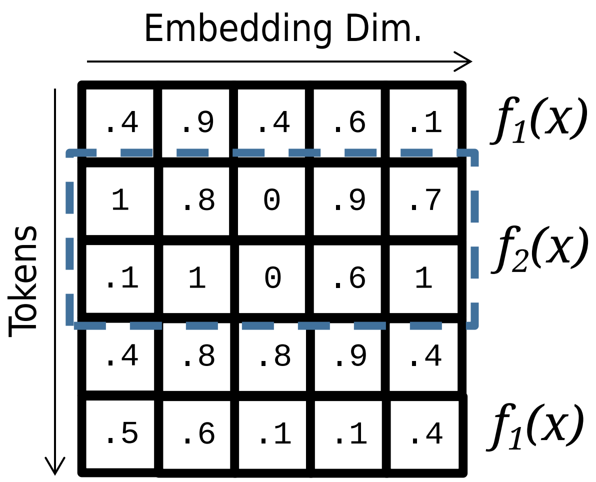
Quantizer Design: How to find error-minimizing map from high to low-precision domain?

- ***Tensor-Wise**: Apply one quantizer over a whole tensor*
- ***Vector-Wise**: Apply different quantizers per token/KV or dim (“channel”)*
- ***Dimension-Wise**: Apply different quantizers per group of dimensions*



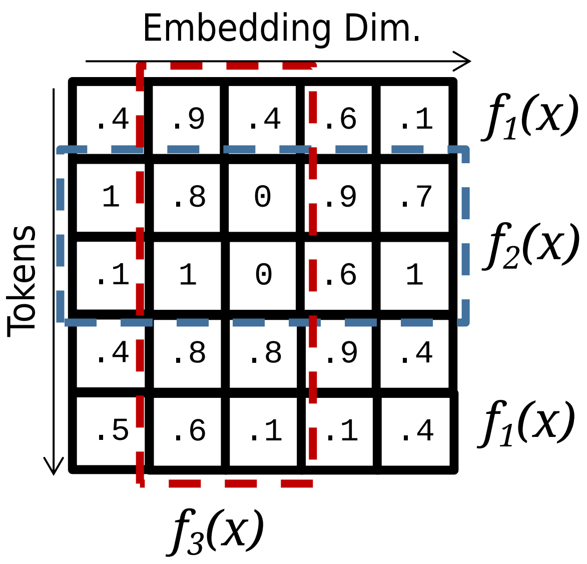
.4	.9	.4	.6	.1
1	.8	0	.9	.7
.1	1	0	.6	1
.4	.8	.8	.9	.4
.5	.6	.1	.1	.4

(a) Tensor-Wise



.4	.9	.4	.6	.1
1	.8	0	.9	.7
.1	1	0	.6	1
.4	.8	.8	.9	.4
.5	.6	.1	.1	.4

(b) Vector-Wise



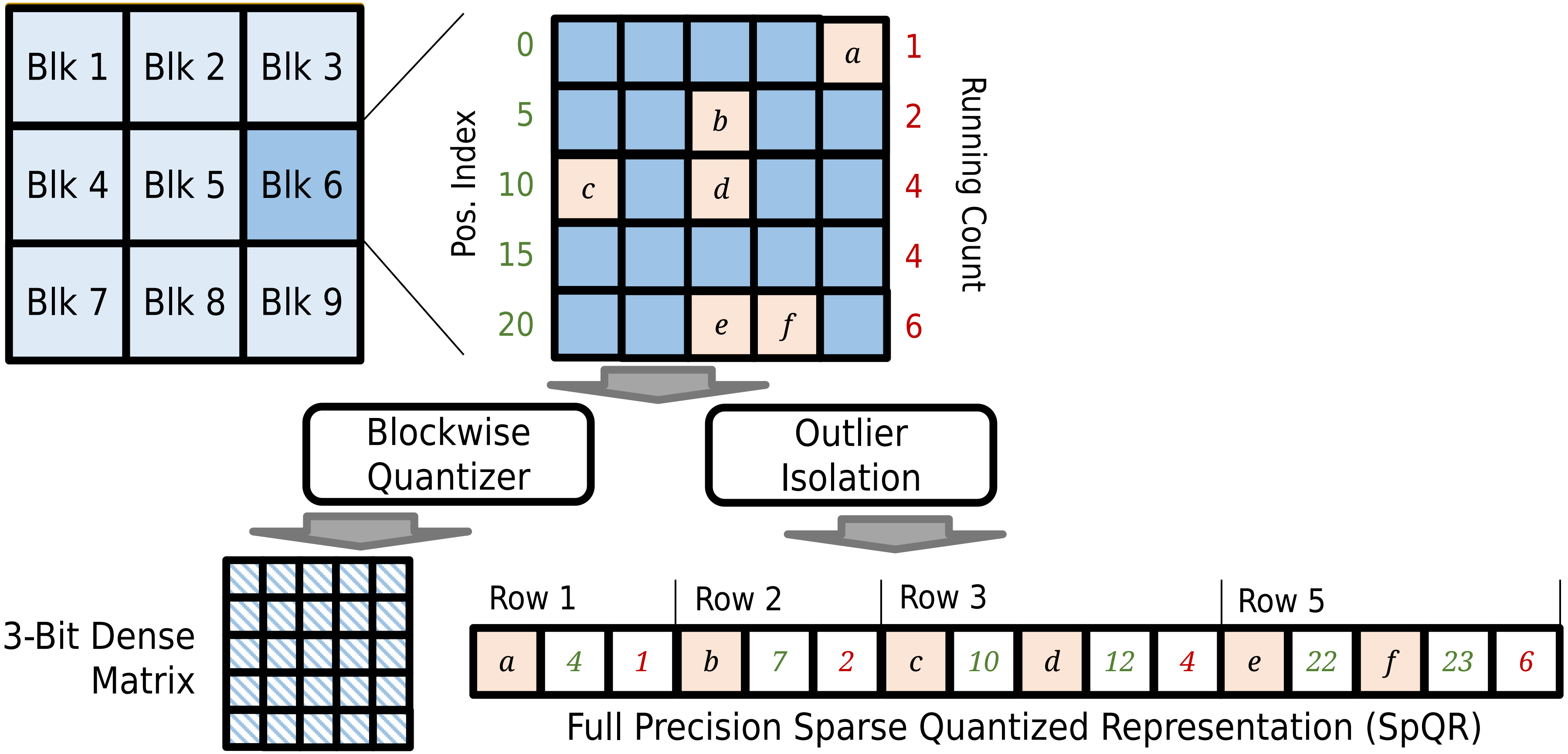
.4	.9	.4	.6	.1
1	.8	0	.9	.7
.1	1	0	.6	1
.4	.8	.8	.9	.4
.5	.6	.1	.1	.4

(c) Dimension-Wise

Quantization: Outlier Protection

Outlier Protection: How to identify & preserve information in outliers?

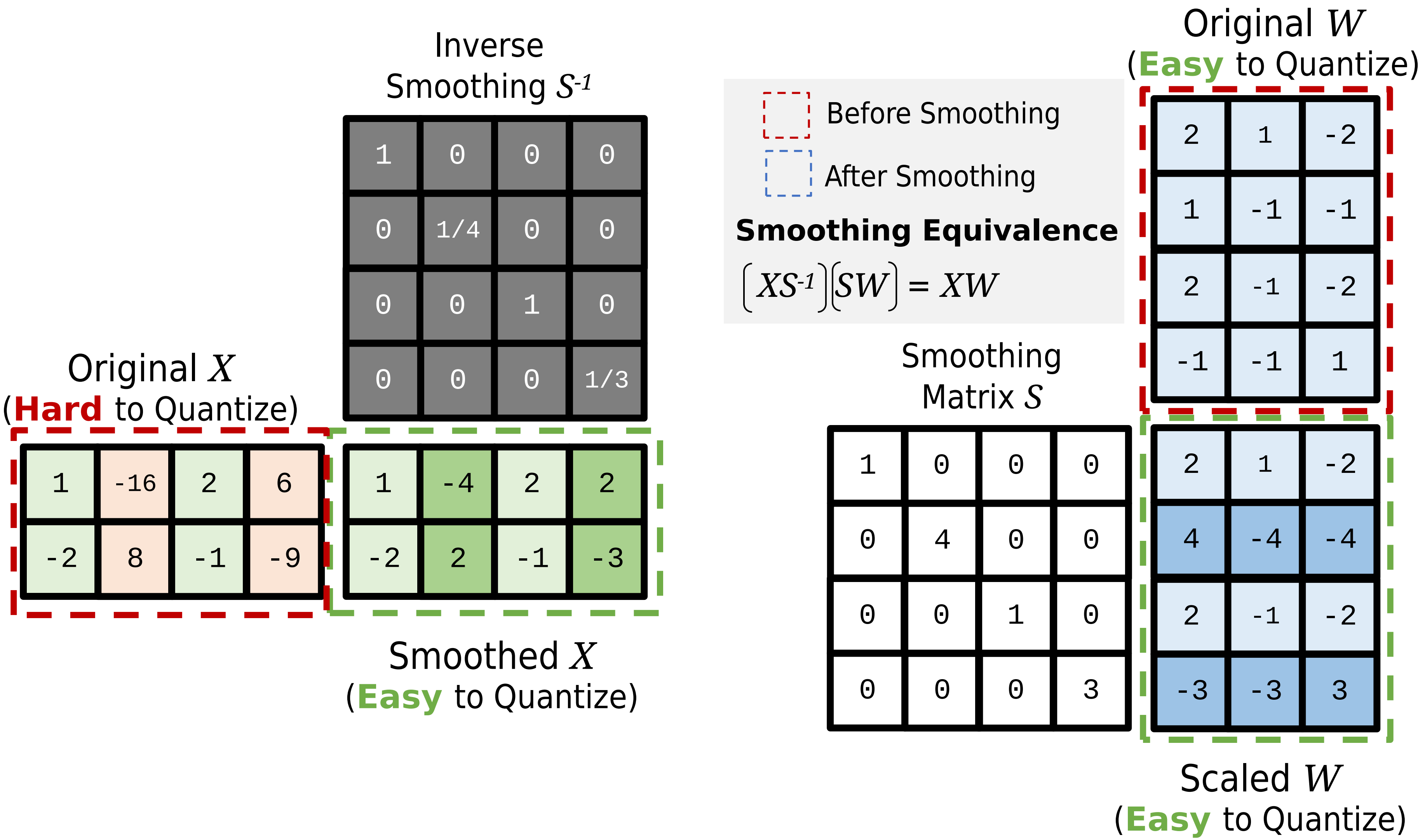
- ***Mixed-Precision*: Keep outliers in raw high-precision form**
  - **SpQR** [Dettmers et al 2023]: Use a sparse representation to hold raw values + special matmul kernel



Quantization: Outlier Protection

Outlier Protection: How to identify & preserve information in outliers?

- ***Outlier Smoothing: Smooth outliers to yield more uniform tensor***



Inverse Smoothing S^{-1}

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1/3 \end{bmatrix}$$

Original X (**Hard** to Quantize)

$$\begin{bmatrix} 1 & -16 & 2 & 6 \\ -2 & 8 & -1 & -9 \end{bmatrix}$$

Smoothed X (**Easy** to Quantize)

$$\begin{bmatrix} 1 & -4 & 2 & 2 \\ -2 & 2 & -1 & -3 \end{bmatrix}$$

Before Smoothing

After Smoothing

Smoothing Equivalence

$$\left(XS^{-1}\right)\left(SW\right) = XW$$

Smoothing Matrix S

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

Original W (**Easy** to Quantize)

$$\begin{bmatrix} 2 & 1 & -2 \\ 1 & -1 & -1 \\ 2 & -1 & -2 \\ -1 & -1 & 1 \end{bmatrix}$$

Scaled W (**Easy** to Quantize)

$$\begin{bmatrix} 2 & 1 & -2 \\ 4 & -4 & -4 \\ 2 & -1 & -2 \\ -3 & -3 & 3 \end{bmatrix}$$

Storage Manager: Summary

Efficiently store KV caches to minimize wasted memory; reduce memory usage via compression

Storage Manager	Technique Classification	Latency	Throughput	Memory	Quality
Block Manager					
• Block Storage (Paged)	Framework		↑	↓	
• Block Sharing & Eviction					
• Prefix Sharing	Optimization	↓	↑	↓	
• Partial Reconstruction	Optimization	↓	↑	↓	↓
• Long Context Eviction	Optimization	↓	↑	↓	↓
• Block Search & Retrieval					
• Radix Tree	Index	↓	↑		
Physical Storage					
• Tiered Storage & Offloading	Framework	↑	↓	↓	
• Distributed Storage	Framework	↑	↑	↓	
• Hot Blocks	Optimization	↓	↑	↑	
Quantizer					
• Quantizer Design	Operator Design	↓	↑	↓	↓
• Outlier Smoothing	Optimization	↓	↑	↓	↓

Part 5: Frontend

Capture user intents in order to automatically optimize prompts and workflows

Frontend	Technique Classification	Technique Description / Key Idea
User Interface		
• Declarative Modules	API	• Capture user intent to support prompt optimization
• Language Extensions	API	• Facilitate programmatic prompting
I/O Interpreter		
• Control Flow	API Feature	
• Prompt Generator		
• Prompt Optimization	Optimization	• Provide automatic prompt engineering
• Template Completion	Optimization	• PD interleave for fast and accurate templates
Seq. Generation		
• Streaming Generation		
• 0-Shot CoT	Optimization	• Increase quality by generating more context
• Few-Shot, 1-Shot CoT	Optimization	• Increase quality by providing more context
• Internalized CoT	Optimization	• Increase quality via fine-tuning
• Structured Generation		
• Beam Search	Framework	• Increase quality via multiple candidate sequences
• x-of-Thoughts	Framework	• Increase quality via multiple candidate sequences

User Interface: Declarative Modules

Declarative Modules: How to capture intent of a request in order to support automatic prompts?

- ***LMQL: Use SQL-like syntax to express intent via output constraints***

```
# use constrained variable to produce a classification
"Based on this, the overall sentiment of the message\
 can be considered to be[CLS]" where CLS in [" positive", " neutral", " negative"]
```

- ***DSPy: Provide callable modules for common requested tasks***

```
math = dspy.ChainOfThought("question -> answer: float")
math(question="Two dice are tossed. What is probability that the sum equals 2?")
```

```
class ExtractInfo(dspy.Signature):
    """Extract structured information from text."""
    text: str = dspy.InputField()
    title: str = dspy.OutputField()
    headings: list[str] = dspy.OutputField()
    entities: list[dict[str, str]] = dspy.OutputField(desc="a list of entities and 
their metadata")
module = dspy.Predict(ExtractInfo)
```

User Interface: Declarative Modules

Declarative Modules: How to capture intent of a request in order to support automatic prompts?

- ***DSPy: Provide callable modules for common requested tasks***

User-Submitted Program

```
cot = dspy.ChainOfThought(BasicGenerateAnswer)
```

System-Generated Prompt

```
Your input fields are:
1. `question` (str)

Your output fields are:
1. `reasoning` (str)
2. `answer` (str)

All interactions will be structured in the following way, with the appropriate
values filled in.

[[ ## question ## ]]
{question}

[[ ## reasoning ## ]]
{reasoning}
```

Automatic zero-shot CoT prompting

User Interface: Language Extensions

Language Extensions: How to intuitively incorporate LLM generation into imperative languages?

- ***SGLang: Provide LLM API with parameterized calling***

```
s += LLM("To answer "+q+", I need "+gen("tool", choices=["calc", "www"]))
if s["tool"] == "calc":
    // .. do something
elif s["tool"] == "www":
    // .. do something
```

Example 1: Using LLM API plus imperative control flow to build a tool-using agent

```
character_regex=(...)
def character_gen(s, name):
    s += user(
        f"{name} is a character in Harry Potter. Please fill in the
following information about this character."
    )
    s += LLM(gen("json_output", max_tokens=256, regex=character_regex))
```

Example 2: The LLM API includes features e.g. regex constrained outputs

I/O Interpreter: Control Flow

Control Flow: How automatically format LLM outputs to enable value-based control flow?

- ***SGLang: Provide LLM API with parameterized calling***

```
s += LLM("To answer "+q+", I need "+gen("tool", choices=["calc", "www"]))
if s["tool"] == "calc":
    // .. do something
elif s["tool"] == "www":
    // .. do something
```

Generated Prompt

Complete the following with one word only: “calc” or “www”. To answer (question here), I need:

I/O Interpreter: Prompt Generator

Prompt Generator: How to automatically optimize a prompt to decr. lat & increase quality?

- ***Declarative Modules: Optimize prompts based on the called module***

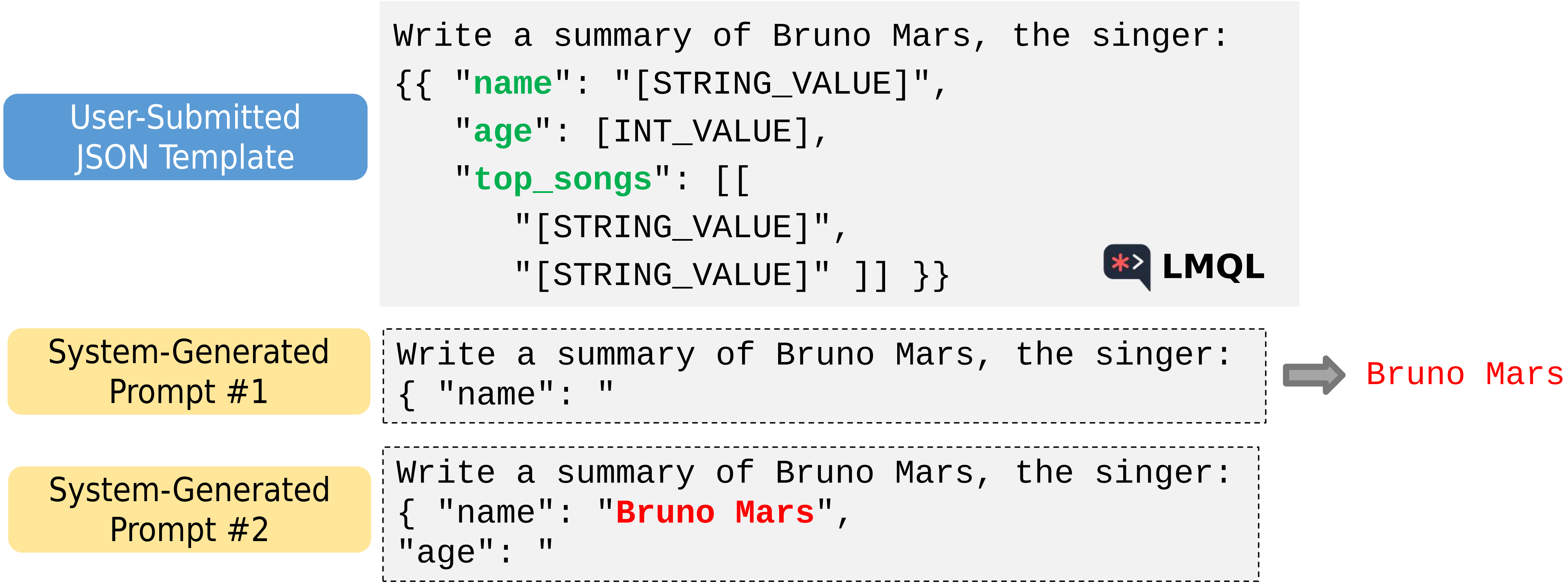
```
# Initialize KNNFewShot with a sentence transformer model
knn_few_shot = KNNFewShot(k=3, trainset=trainset,
vectorizer=dspy.Embedder(xyz).encode))

# Compile the QA module with few-shot learning
compiled_qa = knn_few_shot.compile(qa)

# Use the compiled module
result = compiled_qa("What is the capital of Belgium?")
```

Example: Automatic few-shot prompting

I/O Interpreter: Prompt Generator

Prompt Generator: How to automatically optimize a prompt to decr. lat & increase quality?

- ***Staggered Templates*: Build progressive prompts by interleaved decode**



User-Submitted JSON Template

```
Write a summary of Bruno Mars, the singer:
{{ "name": "[STRING_VALUE]",
   "age": [INT_VALUE],
   "top_songs": [[
      "[STRING_VALUE]",
      "[STRING_VALUE]" ]] }}
```

LMQL

System-Generated Prompt #1

```
Write a summary of Bruno Mars, the singer:
{ "name": "
```

Bruno Mars

System-Generated Prompt #2

```
Write a summary of Bruno Mars, the singer:
{ "name": "Bruno Mars",
"age": "
```

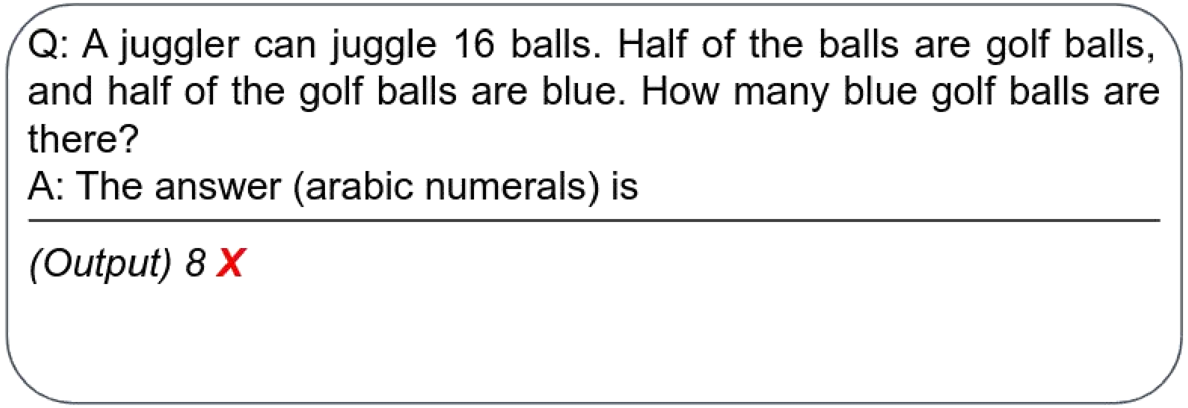
Automatic "staggered" template completion workflow from LMQL

Seq. Generation: Streaming

Streaming Generation: Adding which key phrases illicit high-quality responses?

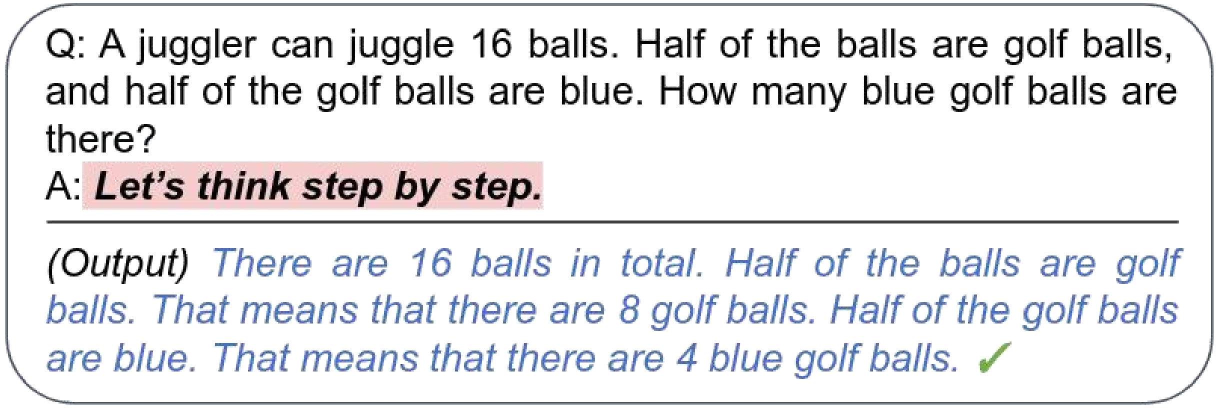
- ***Zero-Shot CoT: Use phrases that yield responses mirroring reasoning***

Base Prompt **vs.** **Zero-Shot Chain-of-Thought (Cot)**



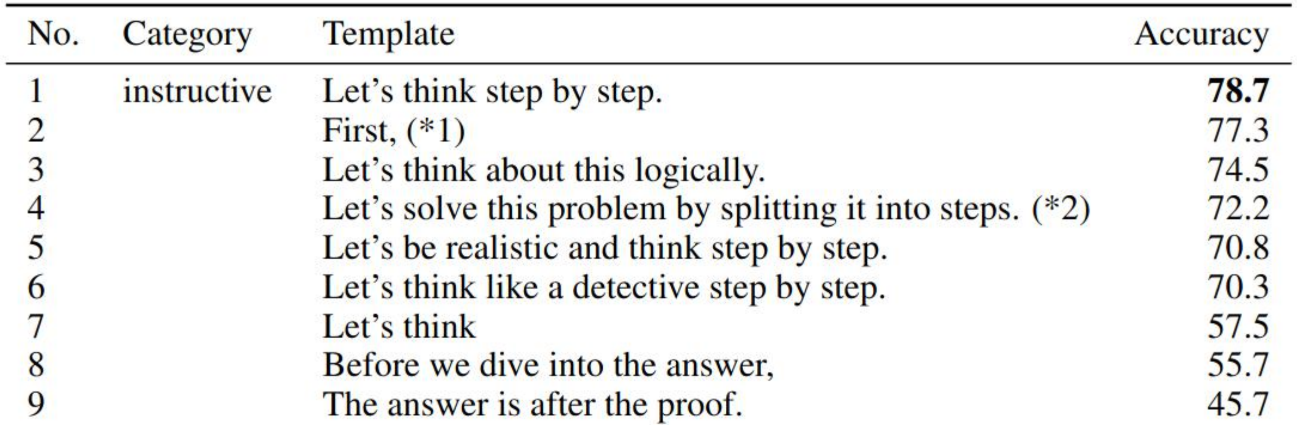
Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: The answer (arabic numerals) is

(Output) 8 ✗



Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: ***Let's think step by step.***

(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓



No.	Category	Template	Accuracy
1	instructive	Let's think step by step.	**78.7**
2		First, (*1)	77.3
3		Let's think about this logically.	74.5
4		Let's solve this problem by splitting it into steps. (*2)	72.2
5		Let's be realistic and think step by step.	70.8
6		Let's think like a detective step by step.	70.3
7		Let's think	57.5
8		Before we dive into the answer,	55.7
9		The answer is after the proof.	45.7

Effect of different phrases on accuracy for math word problems (MultiArith)

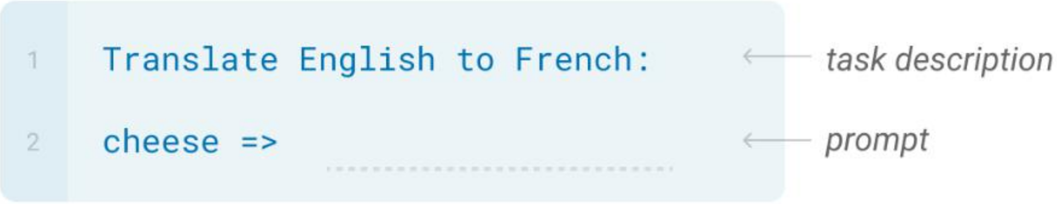
Kojima, T et al. (2022) *Large Language Models are Zero-Shot Reasoners*, arxiv:2205.11916

Seq. Generation: Streaming

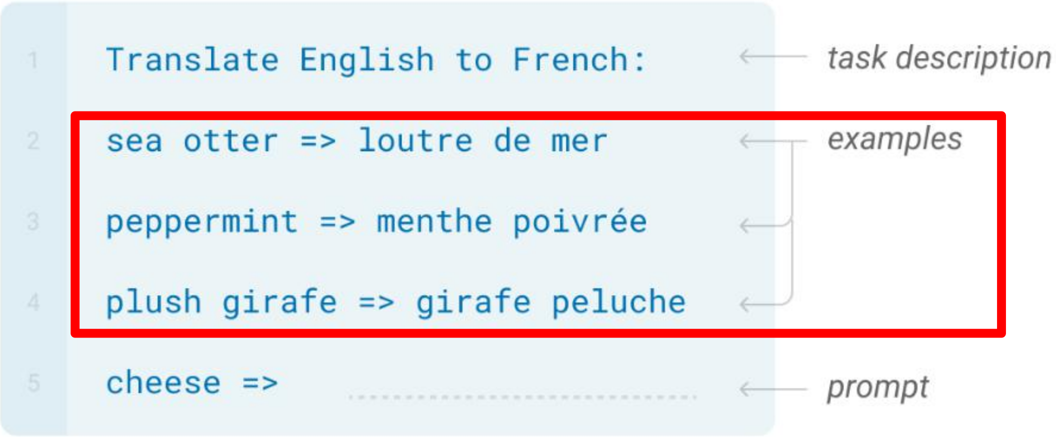
Streaming Generation: Adding which key phrases illicit high-quality responses?

- ***Few-Shot Examples**: Use examples to yield pattern-matching outputs*

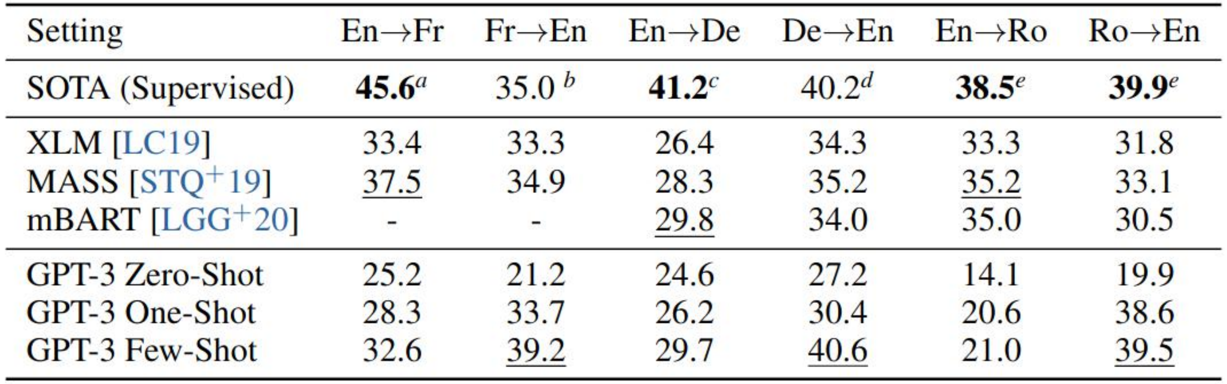
Base Zero-Shot Prompt **vs.** **Few-Shot Prompt**



```
Translate English to French:
cheese => ........................
```



```
Translate English to French:
sea otter => loutre de mer
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese => ........................
```



Setting	En→Fr	Fr→En	En→De	De→En	En→Ro	Ro→En
SOTA (Supervised)	**45.6**[a]	35.0 [b]	**41.2**[c]	40.2[d]	**38.5**[e]	**39.9**[e]
XLM [LC19]	33.4	33.3	26.4	34.3	33.3	31.8
MASS [STQ+19]	37.5	34.9	28.3	35.2	35.2	33.1
mBART [LGG+20]	-	-	29.8	34.0	35.0	30.5
GPT-3 Zero-Shot	25.2	21.2	24.6	27.2	14.1	19.9
GPT-3 One-Shot	28.3	33.7	26.2	30.4	20.6	38.6
GPT-3 Few-Shot	32.6	39.2	29.7	40.6	21.0	39.5

Providing few-shot examples increases BLEU score for translation tasks

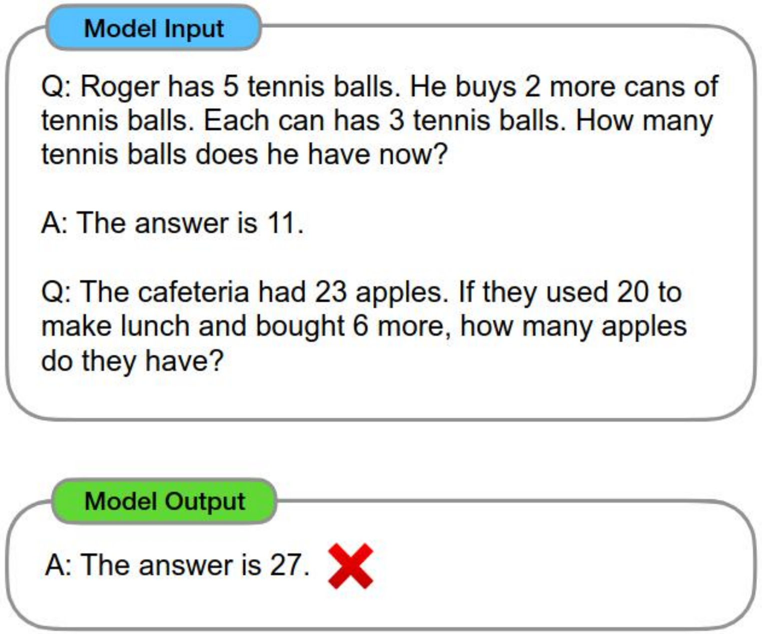
Brown, T et al. (2020) *Language Models are Few-Shot Learners*, NeurIPS'20

Seq. Generation: Streaming

Streaming Generation: Adding which key phrases illicit high-quality responses?

- ***One-Shot CoT**: Add example reasoning to yield reasoning-like output*

Base Zero-Shot Prompt **vs.** **One-Shot CoT Prompt**



Model Input

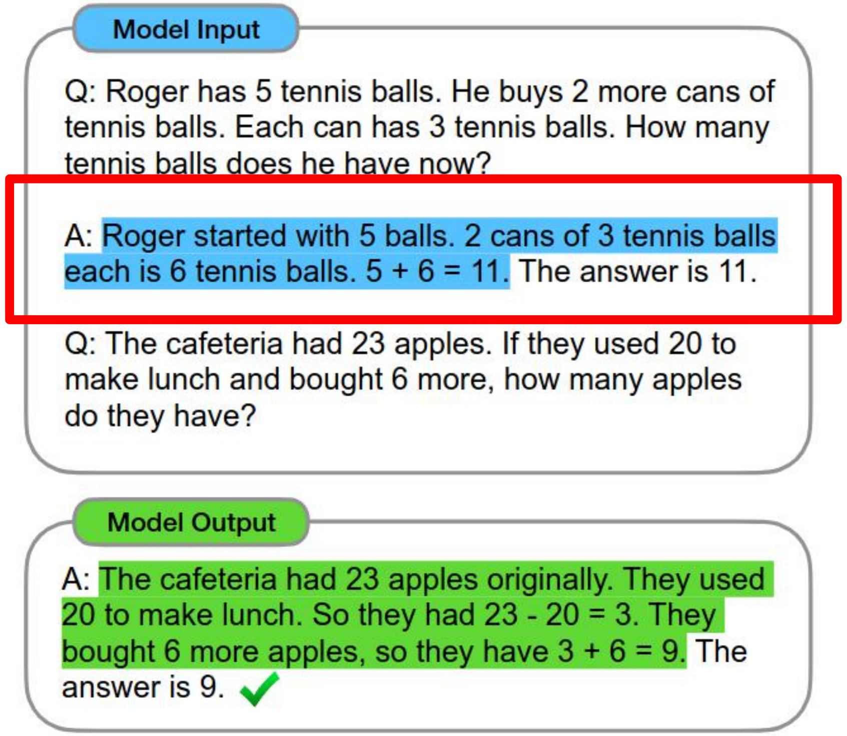
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗



Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

Wei, J et al. (2022) *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*, NeurIPS’22

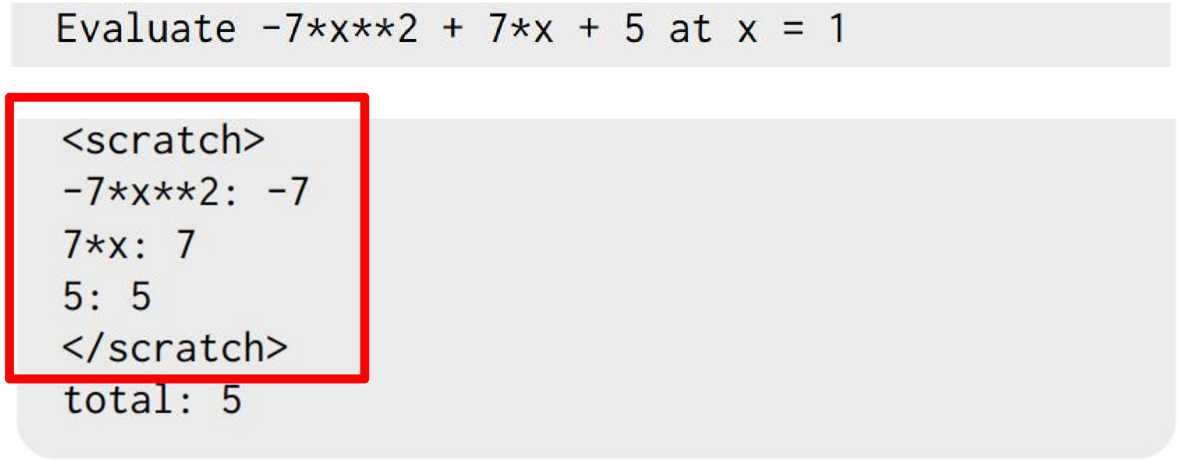
Seq. Generation: Streaming

Streaming Generation: Adding which key phrases illicit high-quality responses?

- ***Internalized CoT**: Fine-tune to yield reasoning-like output w/o key phrases*

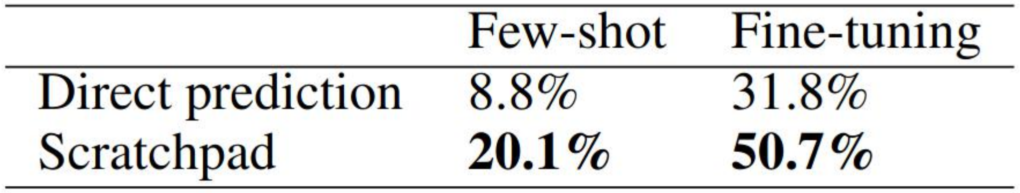
Prompt:

Model Output:



```
Evaluate -7*x**2 + 7*x + 5 at x = 1
```

```
<scratch>
-7*x**2: -7
7*x: 7
5: 5
</scratch>
total: 5
```



	Few-shot	Fine-tuning
Direct prediction	8.8%	31.8%
Scratchpad	**20.1%**	**50.7%**

Fine-tuning with supervised scratchpad increases accuracy over few-shot (i.e. one-shot CoT) alone

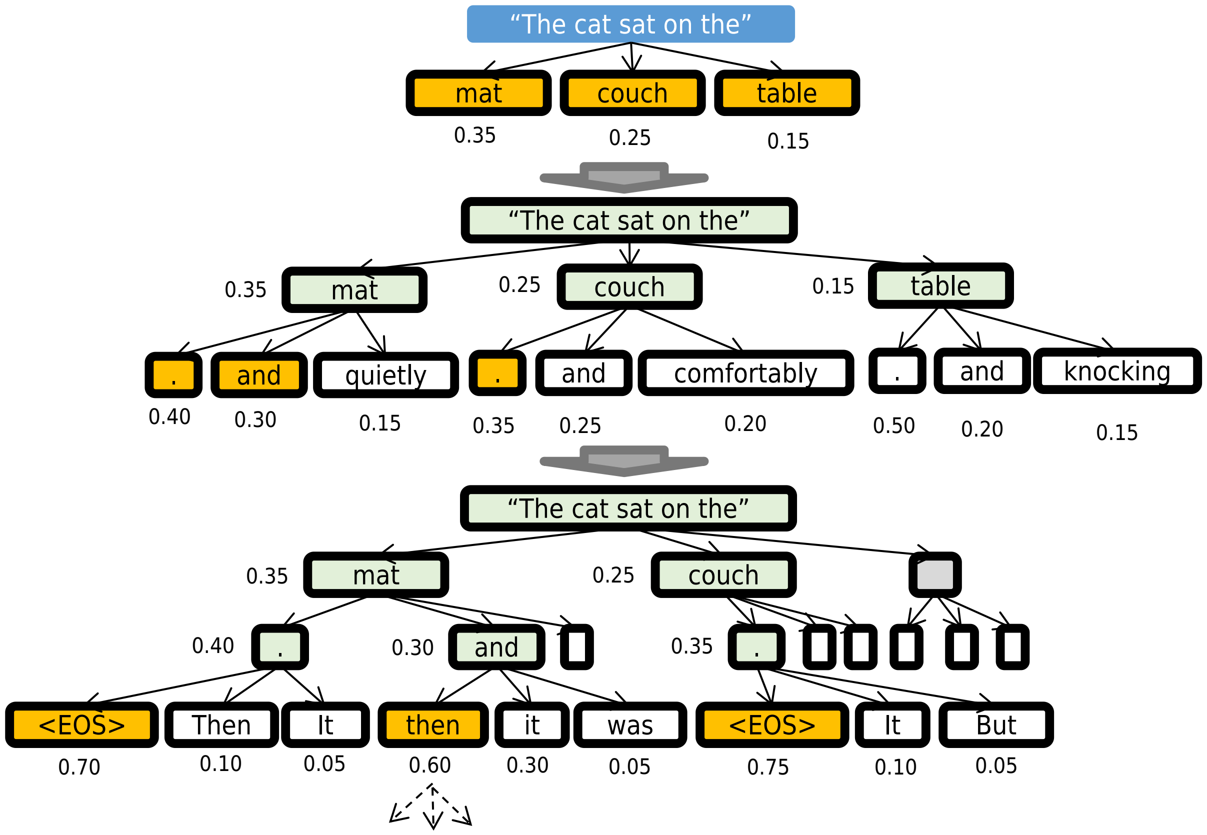
Nye, M et al. (2021) *Show Your Work: Scratchpads for Intermediate Computation with Language Models*, ICLR’21

Seq. Generation: Structured

Structured Generation: Which candidate sequences to generate and how to organize?

- ***Beam Search*: Advance the top-k sequences based on logit score**

Beam Search (k > 1, e.g. k = 3)



Score	Candidate
0.35	The cat sat on the mat
0.25	The cat sat on the couch
0.15	The cat sat on the table

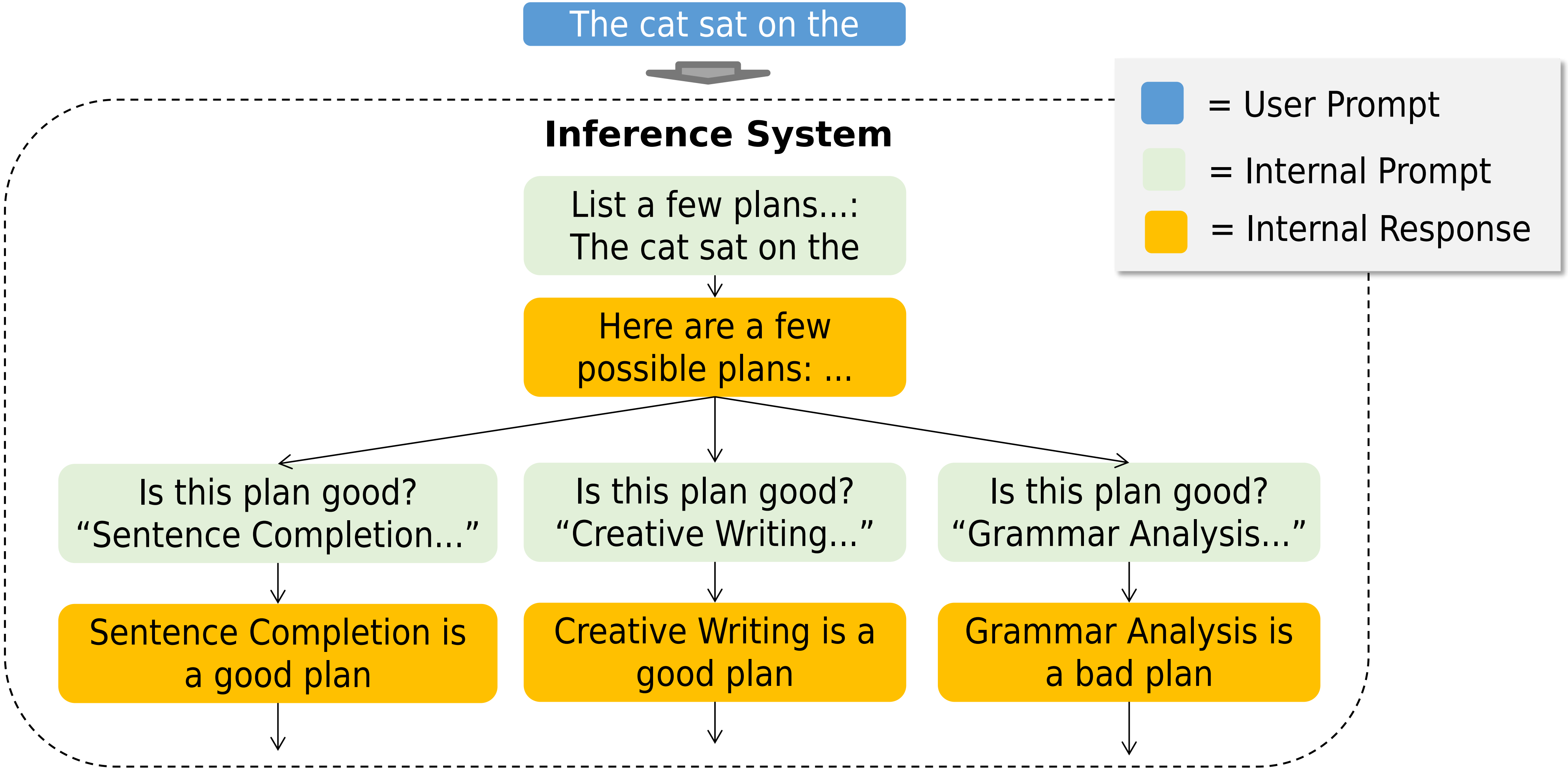
Score	Candidate
0.14	The cat sat on the mat.
0.11	The cat sat on the mat and
0.09	The cat sat on the couch.

Score	Candidate
0.10	The cat sat on the mat.<EOS>
0.07	The cat sat on the mat and then
0.07	The cat sat on the couch.<EOS>

Seq. Generation: Structured

Structured Generation: Which candidate sequences to generate and how to organize?

- ***Tree-of-Thoughts: Advance multiple "thought chains", i.e. sub-requests***



The cat sat on the

Inference System

List a few plans...: The cat sat on the

Here are a few possible plans: ...

Is this plan good? "Sentence Completion..."

Is this plan good? "Creative Writing..."

Is this plan good? "Grammar Analysis..."

Sentence Completion is a good plan

Creative Writing is a good plan

Grammar Analysis is a bad plan

🟦 = User Prompt

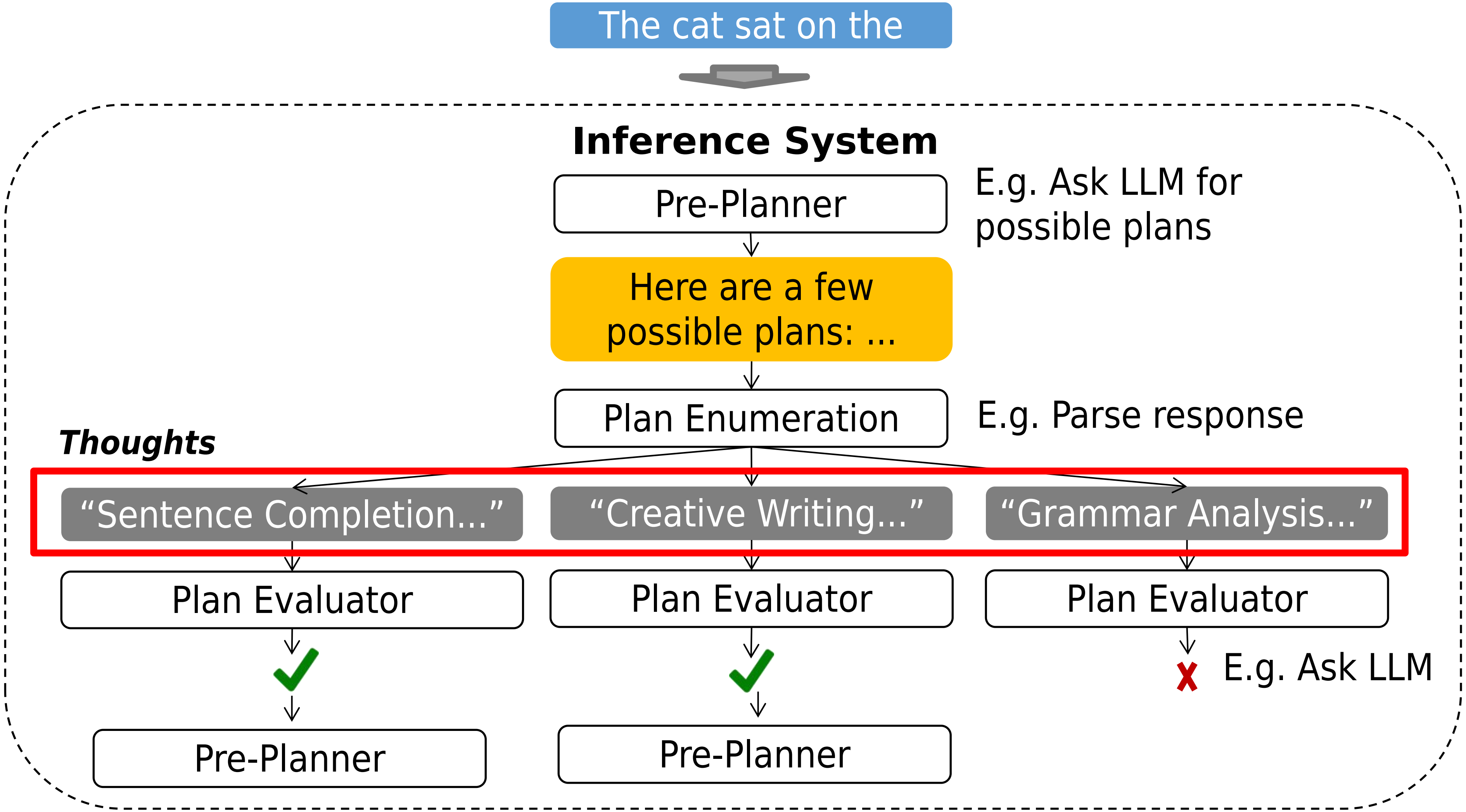
🟩 = Internal Prompt

🟨 = Internal Response

Seq. Generation: Structured

Structured Generation: Which candidate sequences to generate and how to organize?

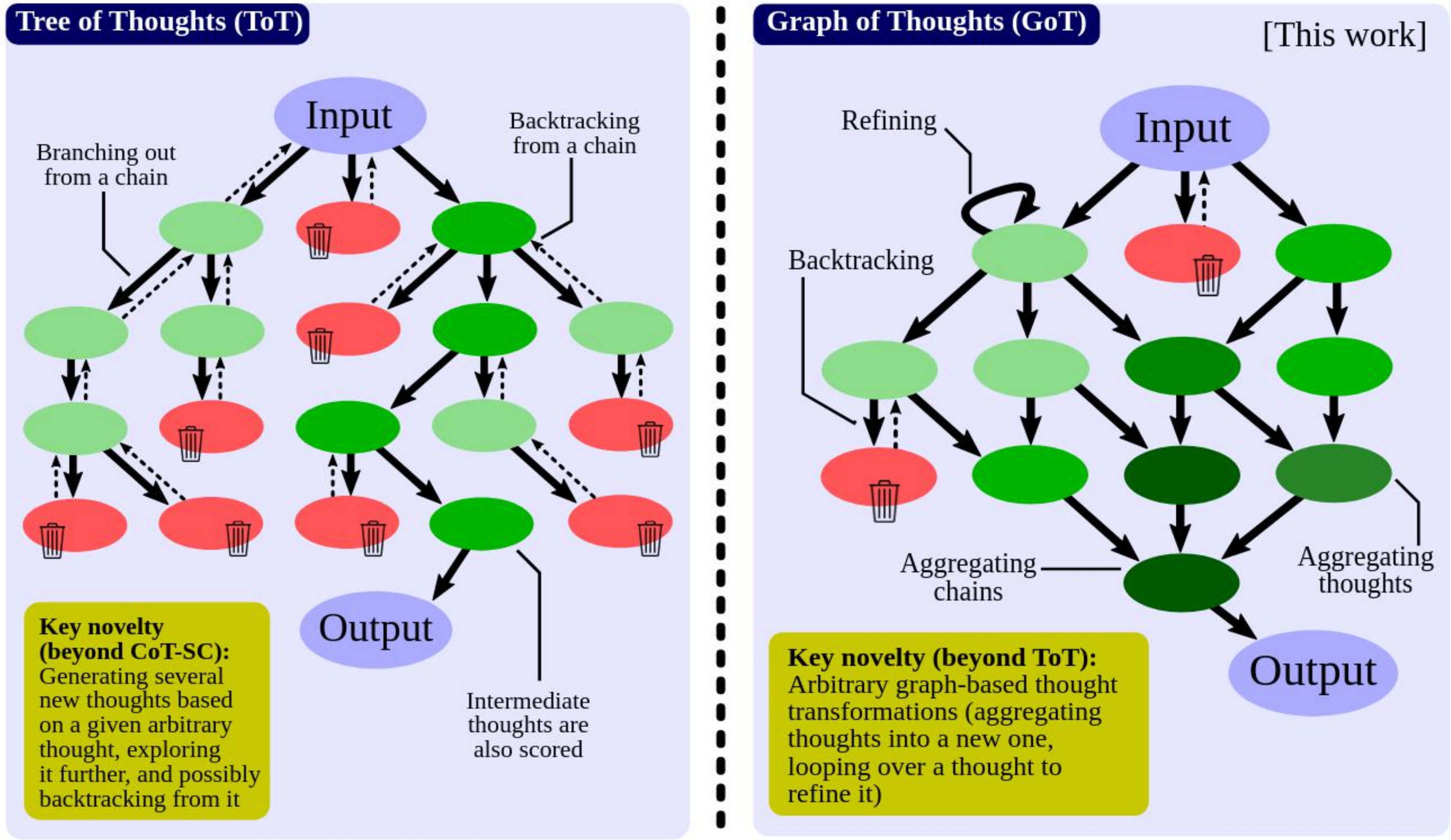
- ***Tree-of-Thoughts*: *Advance multiple “thought chains”, i.e. sub-requests***



Seq. Generation: Structured

Structured Generation: Which candidate sequences to generate and how to organize?

- ***Graph-of-Thoughts*: *ToT with more ops., e.g. “aggregation”, “refine”***



Key novelty (beyond CoT-SC): Generating several new thoughts based on a given arbitrary thought, exploring it further, and possibly backtracking from it

Key novelty (beyond ToT): Arbitrary graph-based thought transformations (aggregating thoughts into a new one, looping over a thought to refine it)

M. **Besta**, N. Blach, A. Kubicek, R. Gerstenberger, M. Podstawski, L. Gianinazzi, J. Gajda, T. Lehmann, H. Niewiadomski, P. Nyczyk, and T. Hoefler. *Graph of thoughts: Solving elaborate problems with large language models*. AAAI’24, 38(16):17682–17690, 2024

Frontend: Summary

Capture user intents in order to automatically optimize prompts and workflows

Frontend	Technique Classification	Latency	Throughput	Memory	Quality
User Interface					
- Declarative Modules	API				
- Language Extensions	API				
I/O Interpreter					
- Control Flow	API Feature				
- Prompt Generator					
- Prompt Optimization	Optimization				↑
- Template Completion	Optimization				↑
Seq. Generation					
- Streaming Generation					
- 0-Shot CoT	Optimization	↑	↓	↑	↑
- Few-Shot, 1-Shot CoT	Optimization	↑	↓	↑	↑
- Internalized CoT	Optimization	↑	↓	↑	↑
- Structured Generation					
- Beam Search	Framework	↑	↓	↑	↑
- x-of-Thoughts	Framework	↑	↓	↑	↑

Part 6: Inference Systems

Build a system for High-Performance and High-Quality inference

	Examples	Key Features	Key Design Aims
Single-Replica	• Orca (2022) • vLLM (2023) • Sarathi (2024) • SGLang (2024) • FastServe (2024)	• Single copy of LLM weights • **Fundamental Scalability Limitation**: Linear Transform (W_Q, W_K, W_V matmul) and FFN cannot be scaled up → Low Throughput	• Increase throughput via latency and memory reduction → faster request processing & larger batch sizes
Multi-Replica	• Preble (2024) • DistServe (2024) • TetriInfer (2024) • SplitWise (2024) • Mooncake (2024) • DeepServe (2025)	• Multiple copies of LLM weights • **Raises total system mem.** • Allows **Data Parallelism** & **Distributed Cache** for larger in-memory persisted KV caches	• Increase throughput and reduce latency via techniques for distributed execution, e.g. **Load Balancing**, **PD Disaggregation**, & **Hot Block Replicas**

Single-Replica Systems

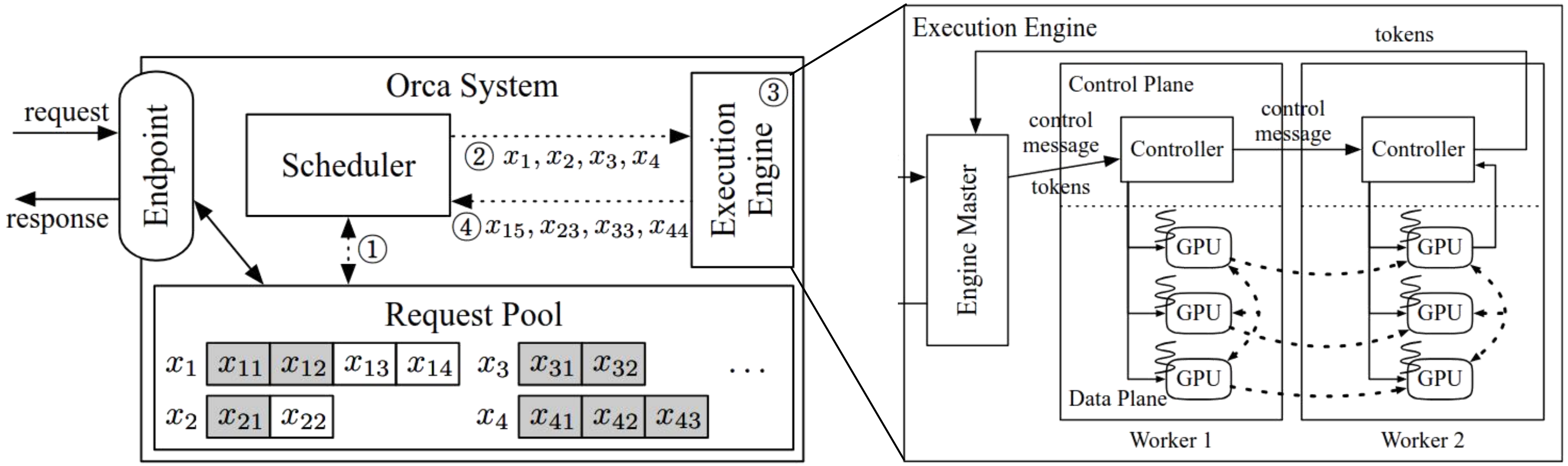
Increase throughput via lat. and mem. reduction → faster request processing & larger batch sizes

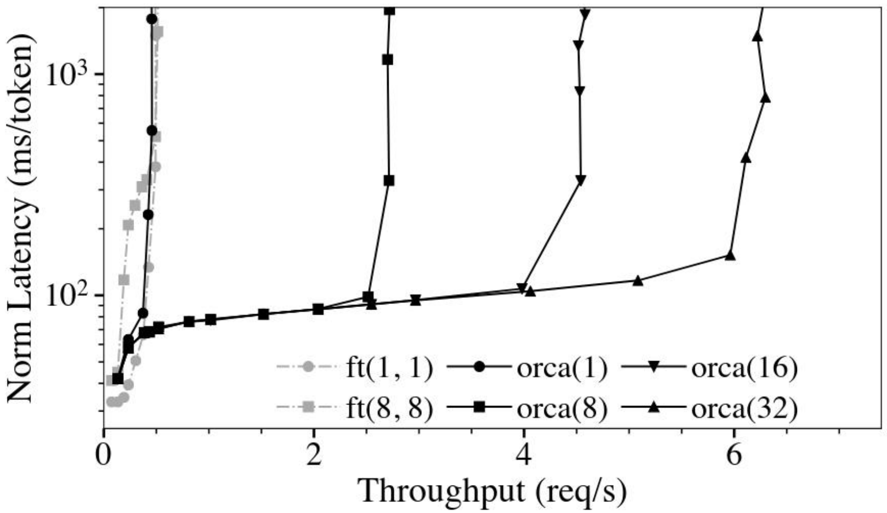
	Latency	Memory	Throughput	Quality
Request Processing	• KV Cache (decode) • Efficient attention	• Grouped / Shared / Sparse Attention	• Speculative Decoding	• MoE
Optimizer / Execution	• Fused / Blockwise Kernels • Cont. Batching • Pipeline Parallelism	• Fused Kernels • Model Parallelism (device mem.)	*Low lat. → greater throughput*	*N/A*
Scheduler	• Job Prioritization supported by Job Cost Prediction • Chunked Prefills	*Low lat. → faster reclamation*		*N/A*
Storage Manager	• Cache Sharing • Block Search • Quantization	• Paged Memory • Cache Sharing • Offloading • Quantization	*Low mem. → larger batch sizes*	*N/A*
Frontend	• Constrained Outputs • Staggered Templ.	*Low lat. → faster reclamation*	*Low lat. → greater throughput*	• Prompt Opt/Eng. • Structured Gen.

Single-Replica: Orca (2022)

- ***Orca (2022)*: Reduce TTFT via continuous batching and reduce TBT via model/pipeline par.**

Request Processing	Optimization / Execution	Scheduling	Storage	Frontend
• KV Cache	• Fused Attention • **Cont. Batching** • **Bursted Attention** • **Model/Pipeline Par**.	• FCFS	• Static Preallocated Memory	N/A



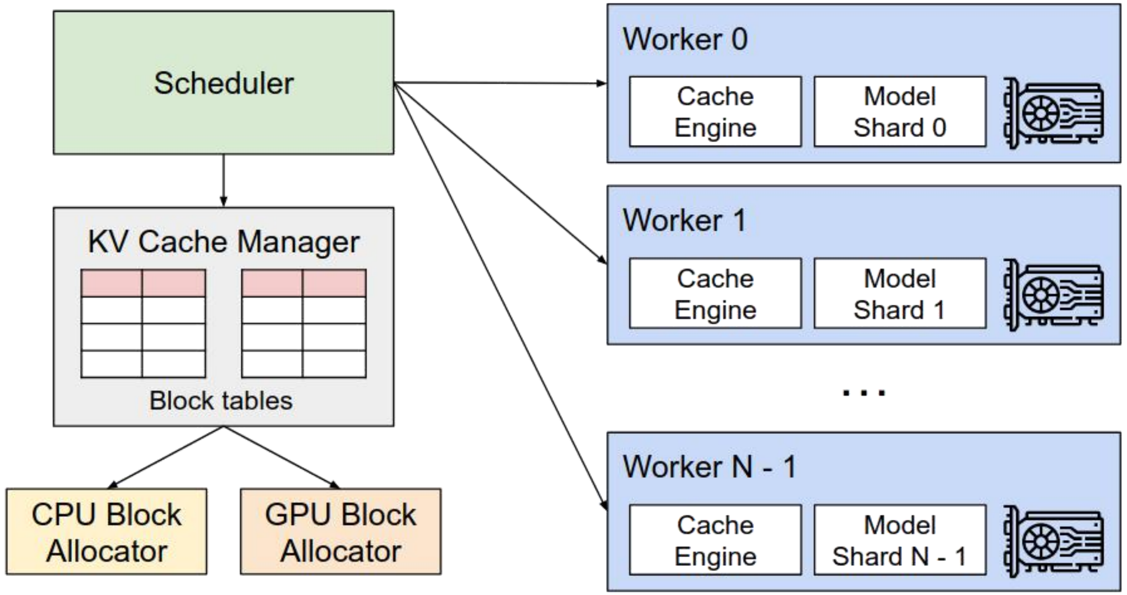


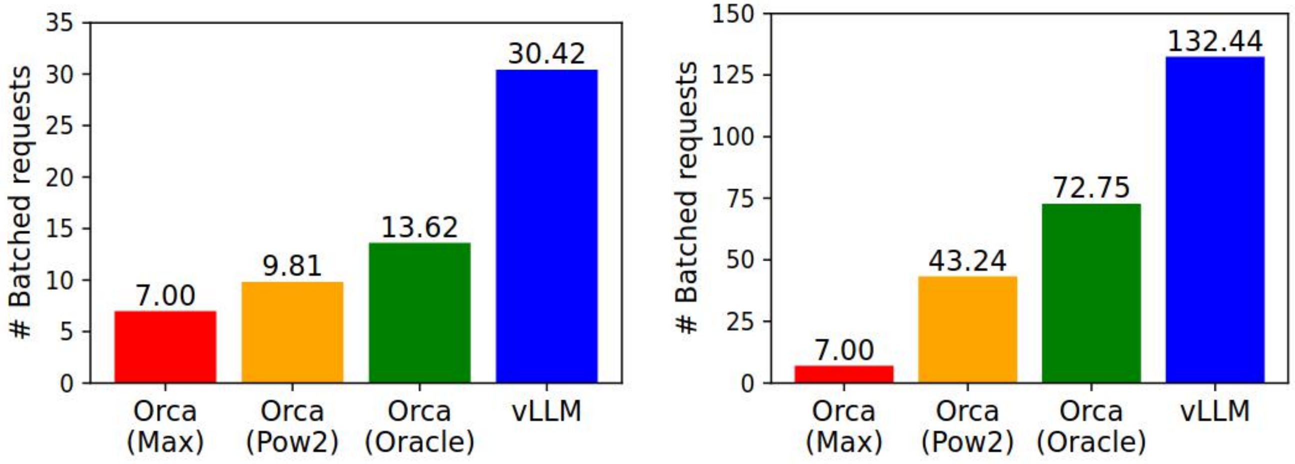
Yu G. I., Jeong J. S., Kim G. W., Kim S., Chun B. G. *ORCA: A Distributed Serving System for Transformer-Based Generative Models. OSDI'22*

Single-Replica: vLLM (2023)

- *vLLM (2023)*: ***Reduce memory waste via paged memory and block sharing***

Request Processing	Optimization / Execution	Scheduling	Storage	Frontend
• KV Cache • Multi-Head Attn. • **Shared Attn.**	• Fused Attention • Cont. Batching • Model/Pipeline Par.	• FCFS	• **Paged Memory** • **Cache Sharing** • **Offloading (Preemption)**	N/A





(a) ShareGPT

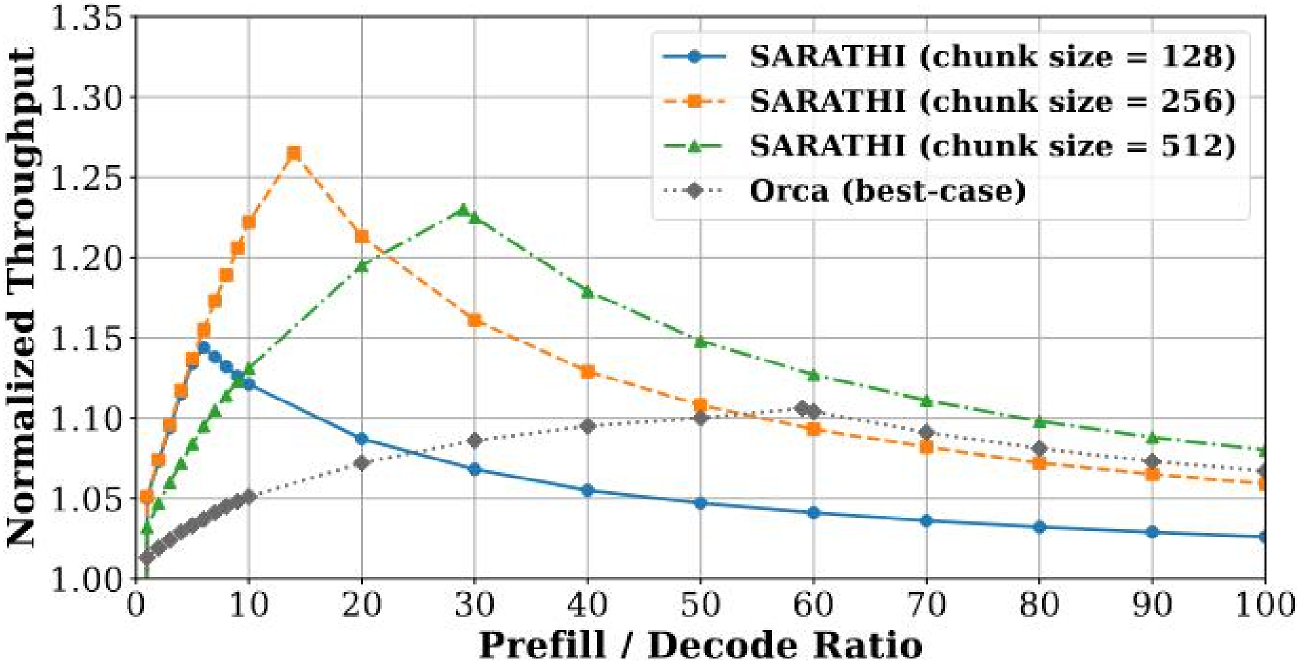
(b) Alpaca

Kwon W., Li Z., Zhuang S., Sheng Y., Zheng L., Yu C. H., Gonzalez J. E., Zhang H., Stoica I. *Efficient Memory Management for Large Language Model Serving with PagedAttention. arXiv:2309.06180*

Single-Replica: Sarathi (2024)

- ***Sarathi (2024)*: Use Chunked Prefills to reduce TBT from straggler batches**

Request Processing	Optimization / Execution	Scheduling	Storage	Frontend
• KV Cache • Multi-Head Attn.	• Fused Attention • **Cont. Batching** • Model/Pipeline Par.	• FCFS • **Chunked Prefills**	• Paged Memory	N/A

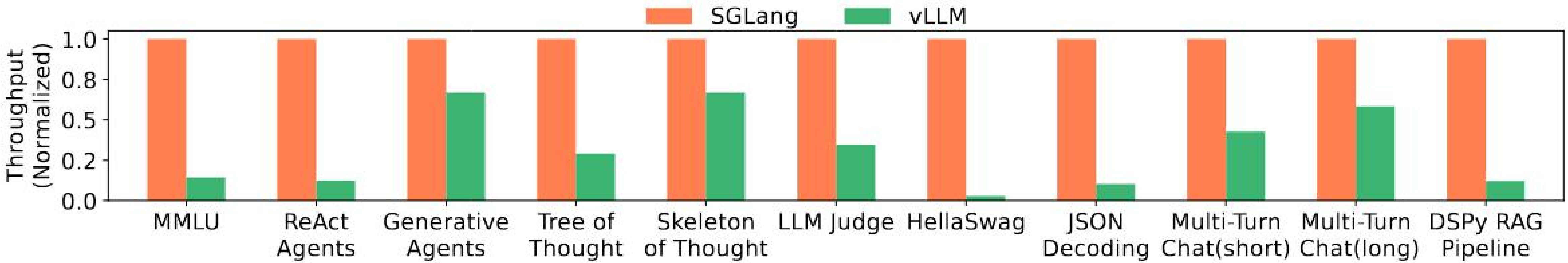


Agrawal, A, Panwar, A, Mohan, J, Kwatra, N, Gulavani, BS, Ramjee, R. *SARATHI: Efficient LLM Inference by Piggybacking Decodes with Chunked Prefills. arXiv:2308.16369*

Single-Replica: SGLang (2024)

- ***SGLang (2024)*****: *Co-design frontend to support fast/accurate template completion, structured gen.***

Request Processing	Optimization / Execution	Scheduling	Storage	Frontend
• KV Cache • Multi-Head Attn. • **Shared Attn.**	• Fused Attention • Cont. Batching • Model/Pipeline Par.	• **Cache Hits Priority**	• Paged Memory • **Cache Sharing** • **Block Search (Radix Tree)**	• **Constrained Gen.** • **Staggered Temp.** • **Structured Gen.**

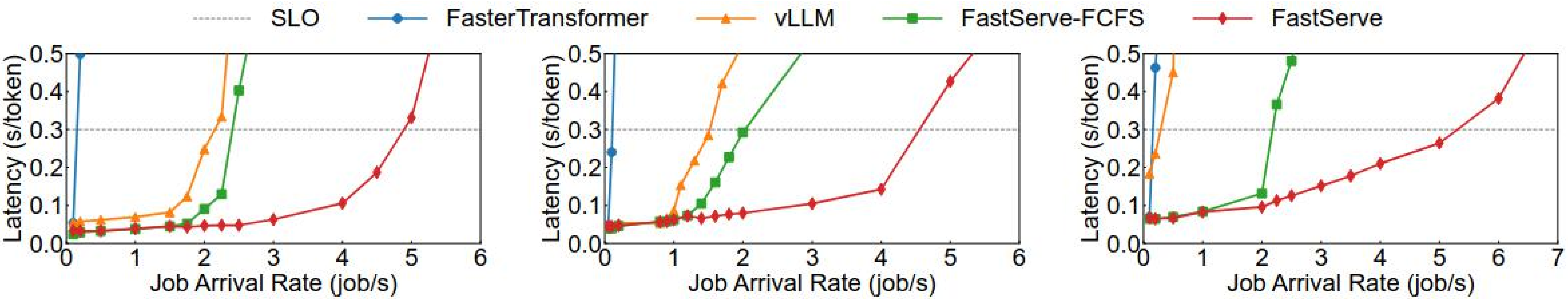


Zheng L., Yin L., Xie Z., Sun C., Huang J., Yu CH., Cao S., Kozyrakis C., Stoica I., Gonzalez JE., Barrett C., Sheng Y.
SGLang: Efficient Execution of Structured Language Model Programs. arXiv:2312.07104

Single-Replica: FastServe (2024)

- ***FastServe (2024)*: Reduce latency from Head-of-Line blocking using MLQ**

Request Processing	Optimization / Execution	Scheduling	Storage	Frontend
• KV Cache • Multi-Head Attn.	• Fused Attention • Cont. Batching • Model/Pipeline Par.	• **Multi-Level Queue**	• Paged Memory • Offloading (Preemption)	N/A



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

(b) OPT-66B, 4 GPUs, ShareGPT.

(c) OPT-175B, 16 GPUs, ShareGPT.

Wu B., Zhong Y., Zhang Z., Liu S., Liu F., Sun Y., Huang G., Liu X., Jin X. *Fast Distributed Inference Serving for Large Language Models. arXiv:2305.05920*

Multi-Replica Systems

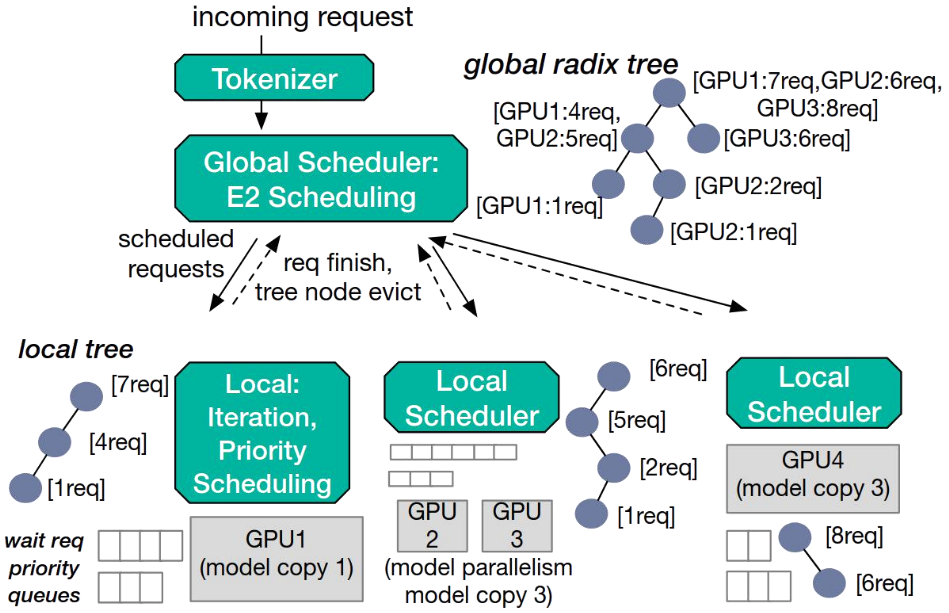
Increase throughput and reduce latency via techniques for distributed execution

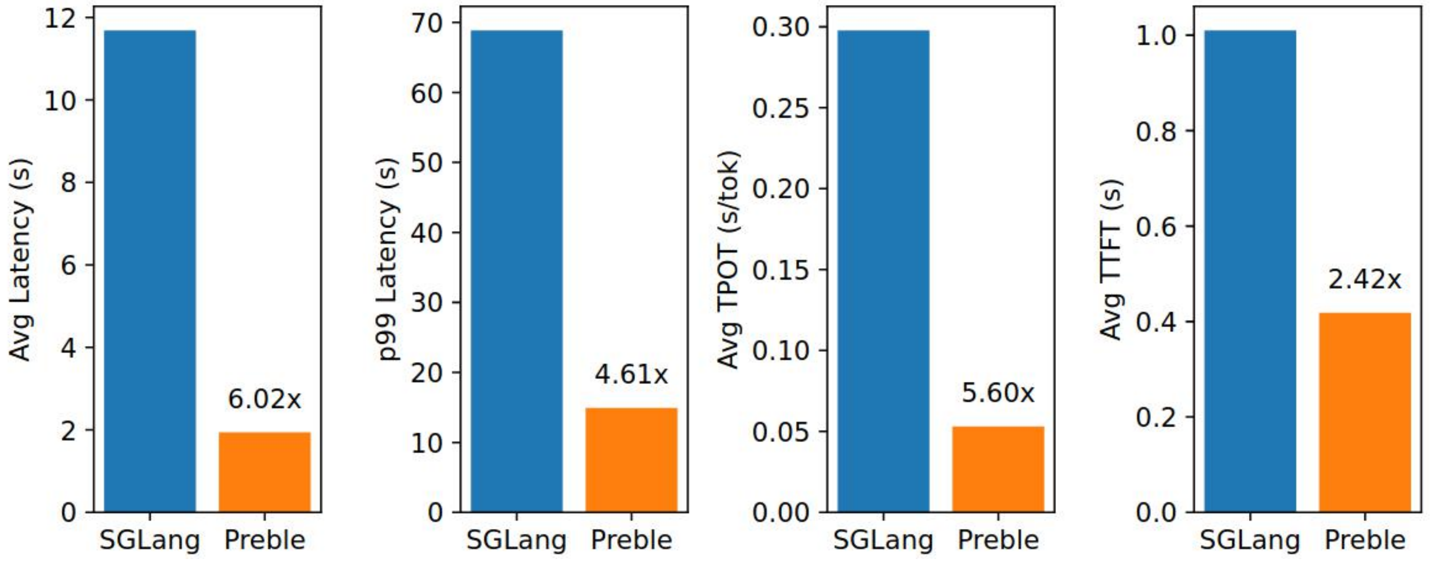
	Latency	Memory	Throughput	Quality
Request Processing	• KV Cache (decode) • Efficient attention	• Grouped / Shared / Sparse Attention	• Speculative Decoding	• MoE
Optimizer / Execution	• Fused / Blockwise Kernels • Cont. Batching • Pipeline Parallelism • **Data Parallelism** • **PD Disaggregation**	• Fused Kernels • Model Parallelism (device mem.)	• **Data Parallelism** • **PD Disaggregation (low lat.)**	*N/A*
Scheduler	• Job Prioritization supported by Job Cost Prediction • Chunked Prefills • **Job Assignment** supported by **Load Prediction**	*Low lat. → faster reclamation*	*Low lat. → greater throughput*	*N/A*
Storage Manager	• Cache Sharing • Block Search • Quantization • **Hot Block Replicas**	• Paged Memory • Cache Sharing • Offloading • Quantization • **Distributed Cache**	• **Hot Block Replicas (low lat.)**	*N/A*
Frontend	• Constrained Outputs • Staggered Templ.	*Low lat. → faster reclamation*	*Low lat. → greater throughput*	• Prompt Opt/Eng. • Structured Gen.

Multi-Replica: Preble (2024)

- ***Preble (2024)*: Decrease workload latency by assigning requests based on cache hits**

Request Processing	Optimization / Execution	Scheduling	Storage	Frontend
- KV Cache - Multi-Head Attn. - **Shared Attn.**	- Fused Attention - Cont. Batching - Model/Pipeline Par. - **Data Parallelism**	- **Cache Hits Priority** - **Cache Hits Load Balancing**	- Paged Memory - Offloading (Preemption) - **Block Search (Radix Tree)**	- *SGLang*



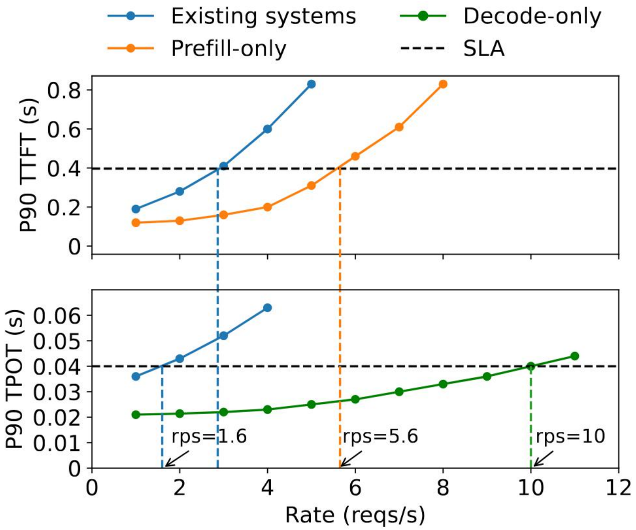


Srivatsa V., He Z., Abhyankar R., Li D., Zhang Y. *Preble: Efficient Distributed Prompt Scheduling for LLM Serving. arXiv:2407.00023*

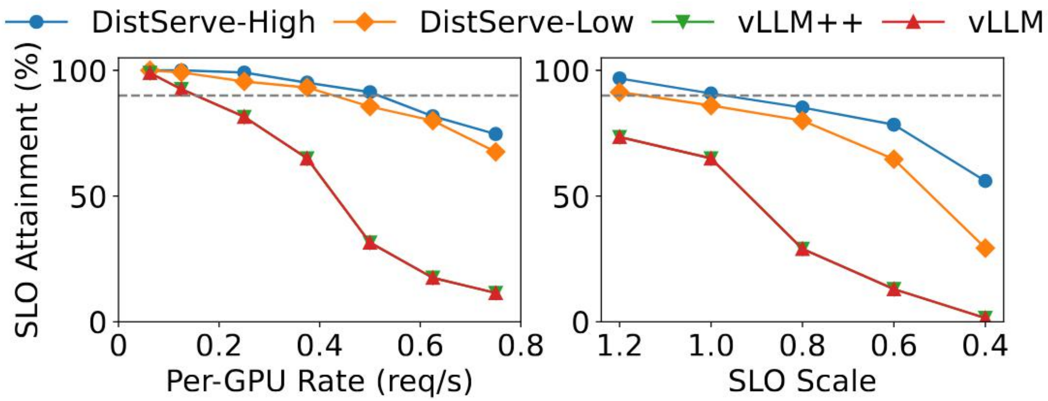
Multi-Replica: DistServe (2024)

- *DistServe (2024)*: ***Provision GPUs in a cluster to P/D in order to maximize goodput***

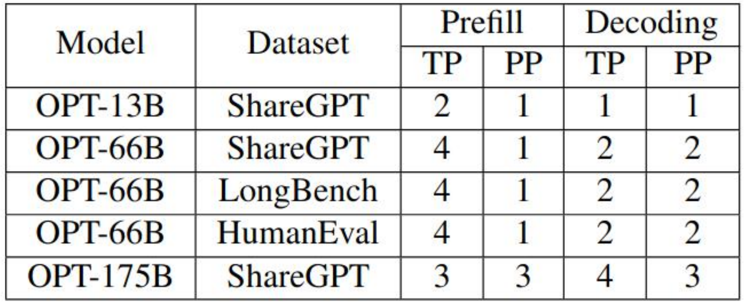
Request Processing	Optimization / Execution	Scheduling	Storage	Frontend
• KV Cache • Multi-Head Attn.	• Fused Attention • Cont. Batching • Model/Pipeline Par. • **Data Parallelism (PD-Disagg.)**	• FCFS • **Greedy Job Assignment (P: Shortest-Queue, D: Least-Load)**	• Paged Memory	N/A



(a) Mixed vs Pure Batches



(b) Allocation Strategy



Model	Dataset	Prefill		Decoding	
		TP	PP	TP	PP
OPT-13B	ShareGPT	2	1	1	1
OPT-66B	ShareGPT	4	1	2	2
OPT-66B	LongBench	4	1	2	2
OPT-66B	HumanEval	4	1	2	2
OPT-175B	ShareGPT	3	3	4	3

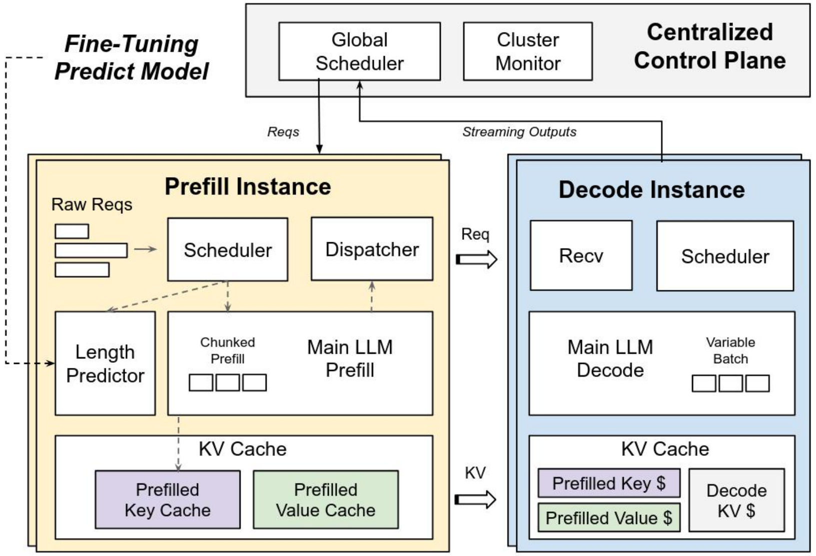
(c) Example Allocations

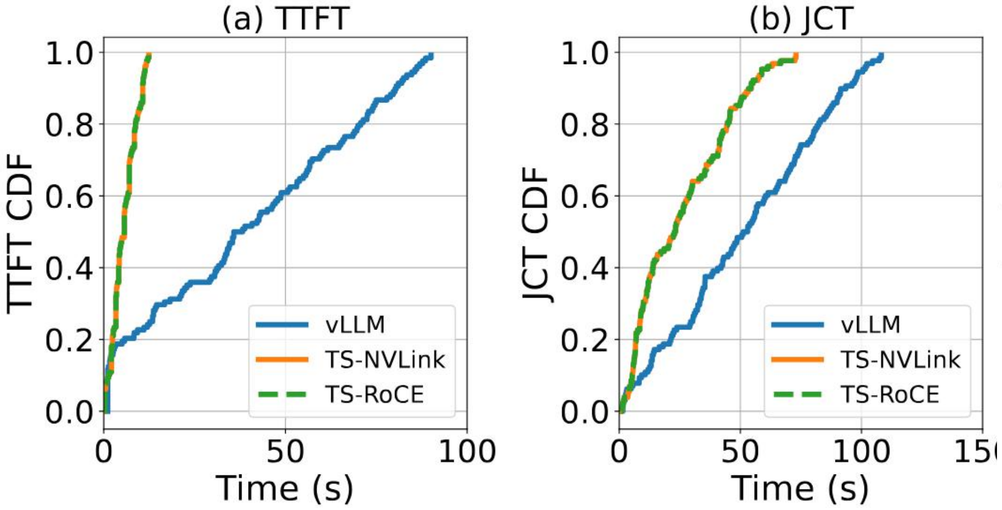
Zhong Y., Liu S., Chen J., Hu J., Zhu Y., Liu X., Jin X., Zhang H. *DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. arXiv:2401.09670*

Multi-Replica: TetriInfer (2024)

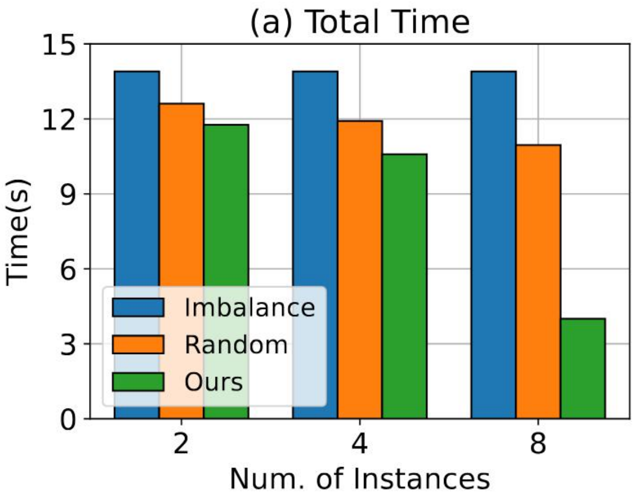
- **TetriInfer (2024)**: ***Decouple P and D scheduling to allow workload targeted scheduling***

Request Processing	Optimization / Execution	Scheduling	Storage	Frontend
• KV Cache • Multi-Head Attn.	• Fused Attention • Cont. Batching • Model/Pipeline Par. • **Data Parallelism (PD-Disagg.)**	• Chunked Prefills • **Job Priority (P: SJF, D: Conservative FCFS)** • **Job Assignment (P: Least-Load, D: Power-2)**	• Paged Memory • Cache Sharing • Offloading (Preemption)	N/A





(a) Disaggregation vs. vLLM



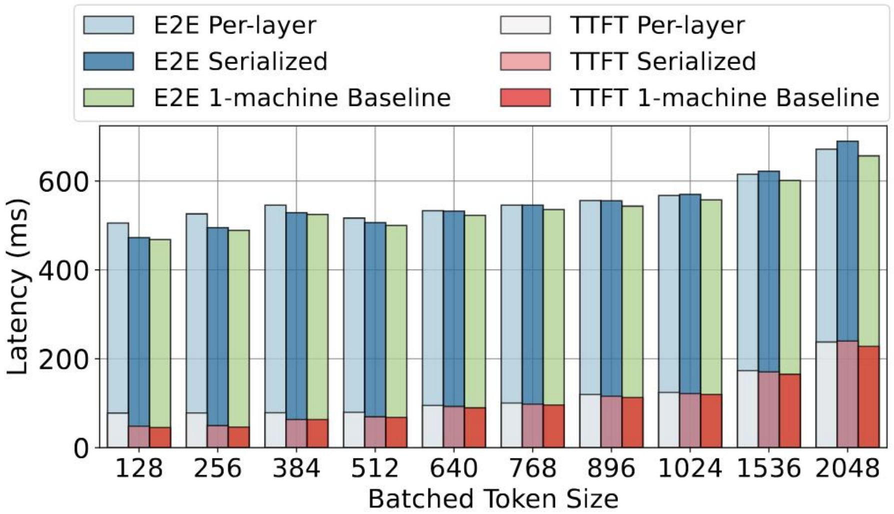
(b) Power-2 vs. Random

Hu C., Huang H., Xu L., Chen X., Xu J., Chen S., Feng H., \Wang C., Wang S., Bao Y., Sun N., Shan Y. *Inference without Interference: Disaggregate LLM Inference for Mixed Downstream Workloads.* *arXiv:2401.11181*

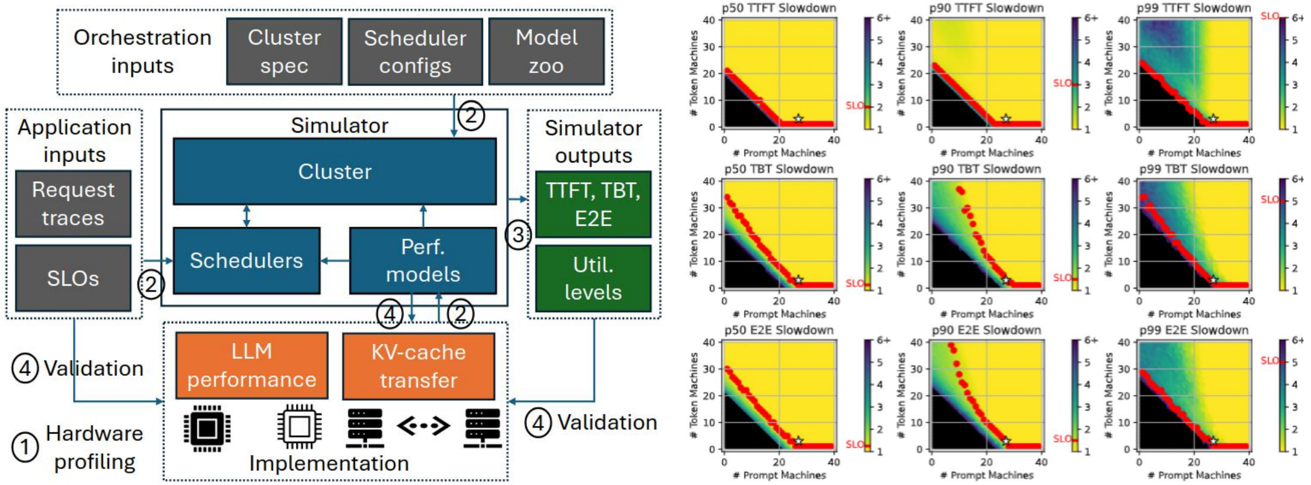
Multi-Replica: SplitWise (2024)

- ***SplitWise (2024)*: Use one-shot load balancing to allow asynchronous PD cache transfer**

Request Processing	Optimization / Execution	Scheduling	Storage	Frontend
• KV Cache • Multi-Head Attn. • Shared Attn.	• Fused Attention • Cont. Batching • Model/Pipeline Par. • **Data Parallelism (PD-Disagg.)**	• FCFS • **One-Shot Greedy Job Assignment (Shortest Queue)**	• Paged Memory • Cache Sharing • Offloading (Preemption)	N/A



(a) Async vs Serial Transfer



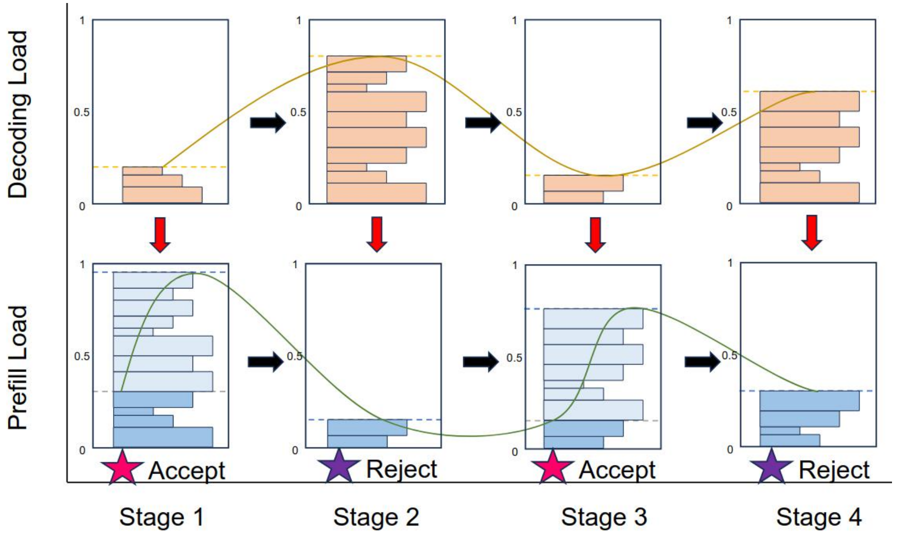
(b) Provisioning Simulator and Results

Patel P., Choukse E., Zhang C., Shah A., Goiri I., Maleki S., Bianchini R. *Splitwise: Efficient Generative LLM Inference Using Phase Splitting. ISCA'24*

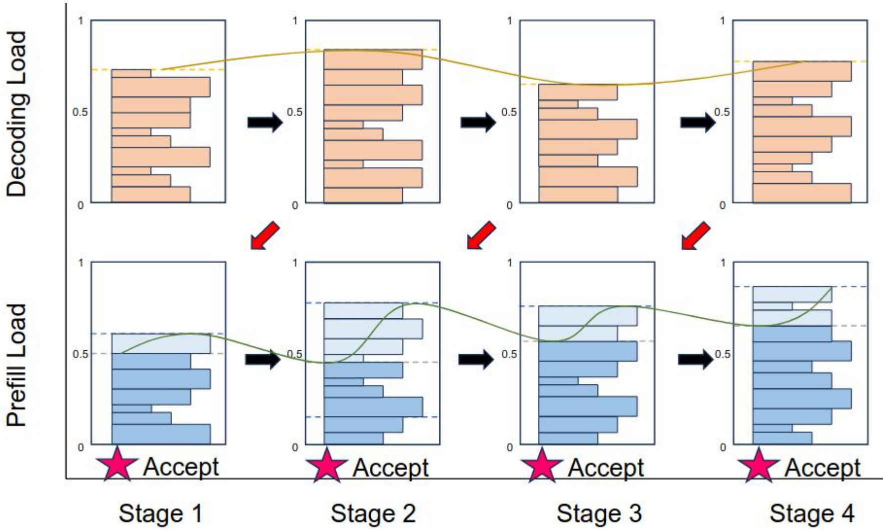
Multi-Replica: Mooncake (2024)

- ***Mooncake (2024)*: Hot blocks & one-shot load balancing with early rejection for overload protection**

Request Processing	Optimization / Execution	Scheduling	Storage	Frontend
• KV Cache • Multi-Head Attn. • **Shared Attn.**	• Fused Attention • Cont. Batching • Model/Pipeline Par. • **Data Parallelism (PD-Disagg.)**	• FCFS • **One-Shot Greedy Job Assignment (P: Cache Hits, D: Least-Load)** • **Early Rejection**	• Paged Memory • Cache Sharing • Offloading (Preemption, **Distributed Cache**) • **Hot Blocks**	N/A



(a) Early Rejection (Instantaneous Load)



(b) Early Rejection (Predicted Load)

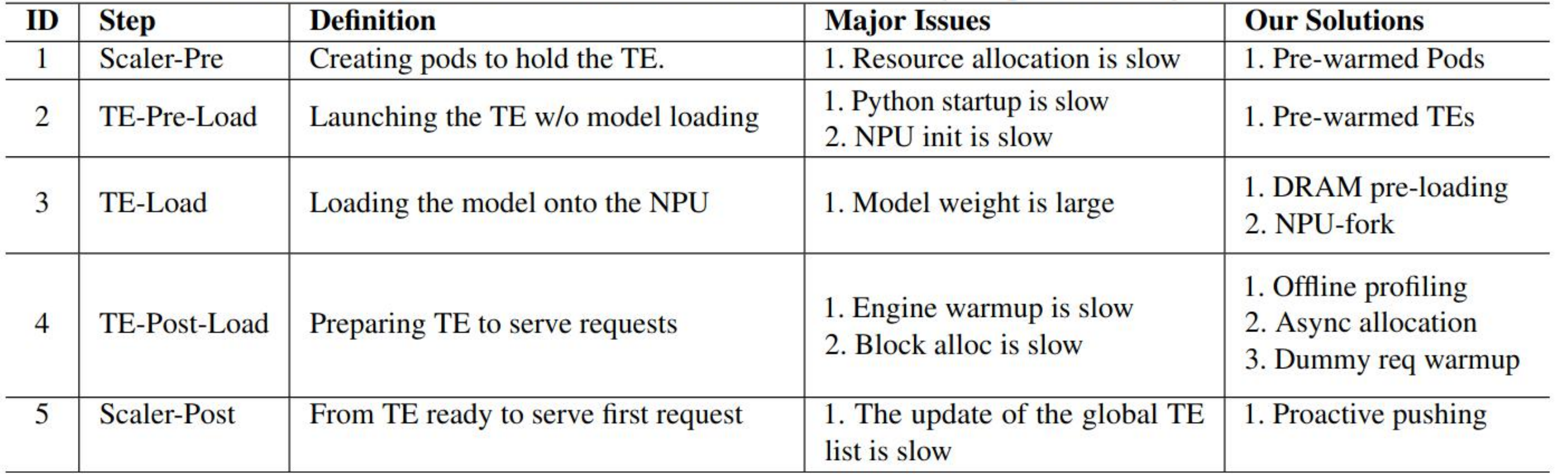
Qin R., Li Z., He W., Zhang M., Wu Y., Zheng W., Xu X. *Mooncake: A KVCache-centric Disaggregated Architecture for LLM Serving. arXiv:2407.00079*

Multi-Replica: DeepServe (2025)

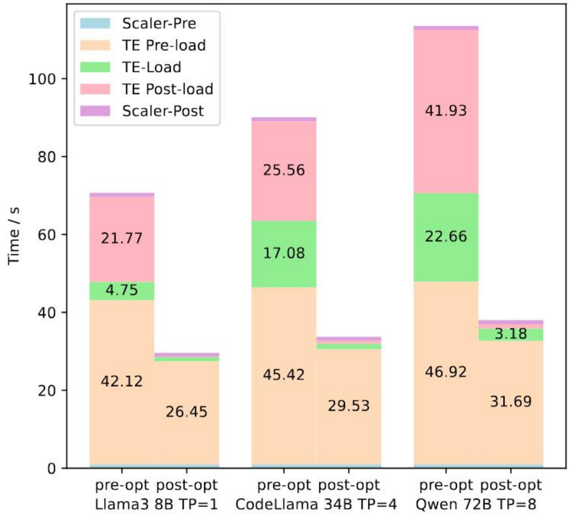
- ***DeepServe (2025)*: Serverless inference system over shared AI infrastructure**

Request Processing	Optimization / Execution	Scheduling	Storage	Frontend
• KV Cache • Multi-Head Attn. • **Shared Attn.**	• Fused Attention • Cont. Batching • Model/Pipeline Par. • **Data Parallelism (PD-Disagg.)**	• **One-Shot Greedy Job Assignment (Heuristic)**	• Paged Memory • Cache Sharing • Offloading (Preemption, **Distributed Cache**) • **Block Search (Radix Tree)**	N/A

Table 2: **A Summary of DeepServe's End-to-End Scaling Steps, Challenges, and Solutions.**



ID	Step	Definition	Major Issues	Our Solutions
1	Scaler-Pre	Creating pods to hold the TE.	1. Resource allocation is slow	1. Pre-warmed Pods
2	TE-Pre-Load	Launching the TE w/o model loading	1. Python startup is slow 2. NPU init is slow	1. Pre-warmed TEs
3	TE-Load	Loading the model onto the NPU	1. Model weight is large	1. DRAM pre-loading 2. NPU-fork
4	TE-Post-Load	Preparing TE to serve requests	1. Engine warmup is slow 2. Block alloc is slow	1. Offline profiling 2. Async allocation 3. Dummy req warmup
5	Scaler-Post	From TE ready to serve first request	1. The update of the global TE list is slow	1. Proactive pushing



Hu J., Xu J., Liu Z., He Y., Chen Y., Xu H., Liu J., Meng J., Zhang B., Wan S., Dan G., Dong Z., Ren Z., Liu C., Xie T., Lin D., Zhang Q., Yu Y., Feng H., Chen X., Shan Y. *DeepServe: Serverless Large Language Model Serving at Scale. arXiv:2501.14417*

Inference Systems: Summary

Fundamental techniques + workload/performance-driven design and system configuration

Fundamental Techniques

Fundamentally efficient techniques

- KV Cache
- Fused/Blockwise Kernels
- Continuous Batching
- Paged Memory

Design Choices

Based on workload or resource considerations

- Job Priority/Assignment
    - Cost-Based vs. Cost-Agnostic
- Cache Management
    - Persisted vs. Non-Persisted
    - In-Memory vs. Tiered Storage
    - Replicated vs. Non-Replicated
- Frontend
    - Specialized vs. General Reqs.
- Architecture
    - Single vs. Multi-Replica
    - Mono. vs. Disaggregated
- Quantization
    - Quantized vs. Raw

Configuration Tuning

Based on performance objectives

- Batch Size
- Chunk Size
- Resource Provisioning (e.g. # of P and D workers, # of GPUs per layer, etc.)
- Quantization Scheme

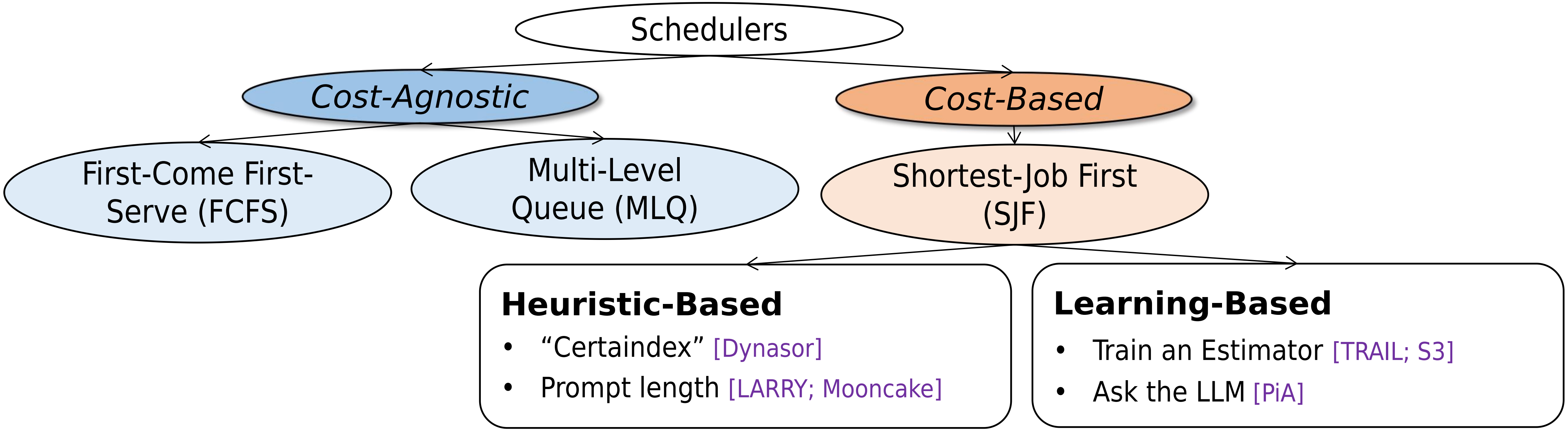
Inference Systems: Summary

Existing systems are general-purpose and tend towards memory-rich environments

System	Architecture	Job Priority/Assign.	Cache Management	Frontend
Single-Replica				
• Orca (2022)	Single	Cost-Agnostic	In-Mem	General
• vLLM (2023)	Single	Cost-Agnostic	Persisted In-Mem	General
• Sarathi (2024)	Single	Cost-Agnostic	In-Mem	General
• SGLang (2024)	Single	Cost-Agnostic	Persisted In-Mem	Special + Gen
• FastServe (2024)	Single	Cost-Agnostic	In-Mem	General
Multi-Replica				
• Preble (2024)	Multi Mono	Cost-Agnostic	Persisted In-Mem	General
• DistServe (2024)	Multi Disagg	Cost-Agnostic	In-Mem	General
• TetriInfer (2024)	Multi Disagg	Cost-Based	Persisted In-Mem	General
• SplitWise (2024)	Multi Disagg	Cost-Agnostic	Persisted In-Mem	General
• Mooncake (2024)	Multi Disagg	Cost-Base	Persisted Tiered Repl	General
• DeepServe (2025)	Multi Disagg	Cost-Agnostic	Persisted In-Mem	General

Future Opportunities: Scheduling

Scheduling techniques raise throughput by minimizing queueing delays



Heuristic-Based

- "Certaindex" [Dynasor]
- Prompt length [LARRY; Mooncake]

Learning-Based

- Train an Estimator [TRAIL; S3]
- Ask the LLM [PiA]

Key Challenges for the DB Community

- **Scheduler Design**
  - **Robust Schedulers**: Stall Prevention, Rebalancing
- **Job Cost & Load Prediction**
- **System Integration**: Co-design scheduler + batcher, e.g. adaptive chunk/batch size & job priority while balancing TTFT, TBT, SLO

Future Opportunities: Storage Manager

Paged memory increases memory efficiency via dynamic memory allocation & block sharing

Key Challenges for the DB Community

Stage	Techniques	Things to Consider
Block Storage	- Direct Storage, e.g. GPU Shared Memory - Tiered Storage, i.e. Offloading	Hot blocks, search & retrieval costs, transfer cost
Block Search	- Exact-match hash table - Exact-match radix tree	Block granularity, partial matches, searching by other than matched tokens, integrating with entry-based techniques
Block Retrieval	- GPU to GPU - DRAM to GPU (offloaded blocks) - Remote DRAM (distributed blocks)	For offloaded / distributed blocks, balancing retrieval + reconstruction cost with savings from reuse
Block Reuse	- Use without modification (i.e. prefix sharing) - Selective Reconstruction	Balancing accuracy with overhead from reuse, e.g. amount of reconstructed vectors
Block Eviction	- LRU, score-based	Potentially useful vs. historically useful blocks

Future Opportunities: Frontend

Seq. Gen. techniques can increase quality by increasing context but raises inference cost

	Prompt Eng.			*Structured Gen.*					
Frontend	**Auto CoT**	**Auto Few-Shot**	**Auto Reasoning**	**Control Flow**	**Structured Output**	**Template Comp.**	**Auto Beam**	**Auto ToT**	**Auto GoT**
LMQL (Declarative)		Random		✓	✓	✓	Manual		
DSPy (Declarative)	Module	Random, k-NN		✓	✓		Module		
SGLang				✓	✓	✓			
Guidance				✓	✓				
LangChain		Random, k-NN		✓	✓				

Manual ↔ *Auto*

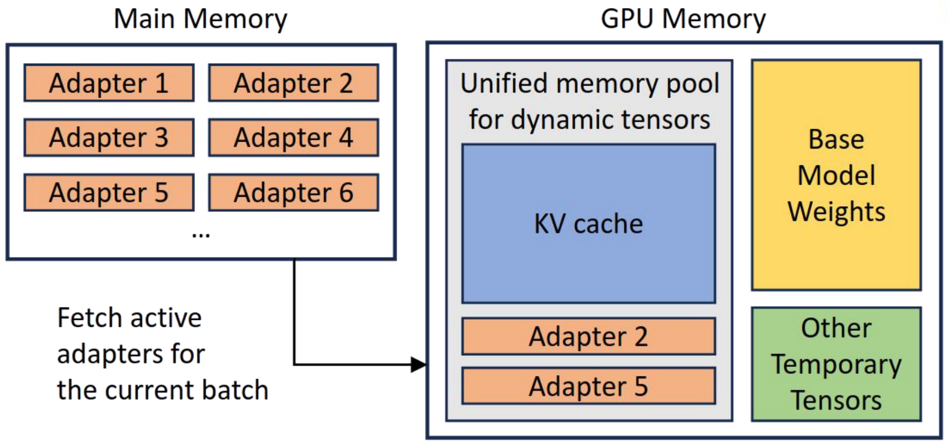
Key Challenges for the DB Community

- **LLM Query Optimization**: Which generation technique to use given a user request?
  - Capturing user intent (Query Parsing)
  - Optimizing prompt contents (Prompt Engineering)
  - Optimizing prompt workflows (Structured Generation)

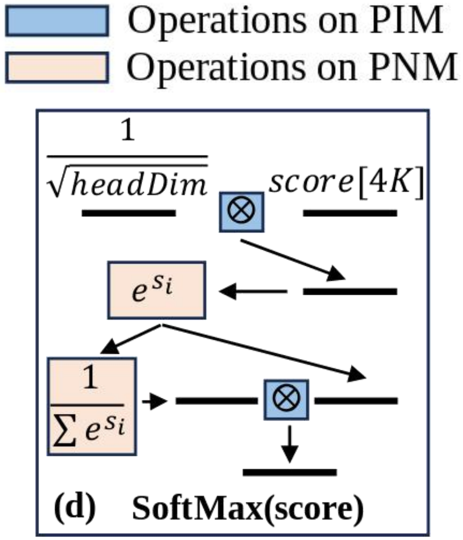
Future Opportunities: Other

Key Challenges for the DB Community

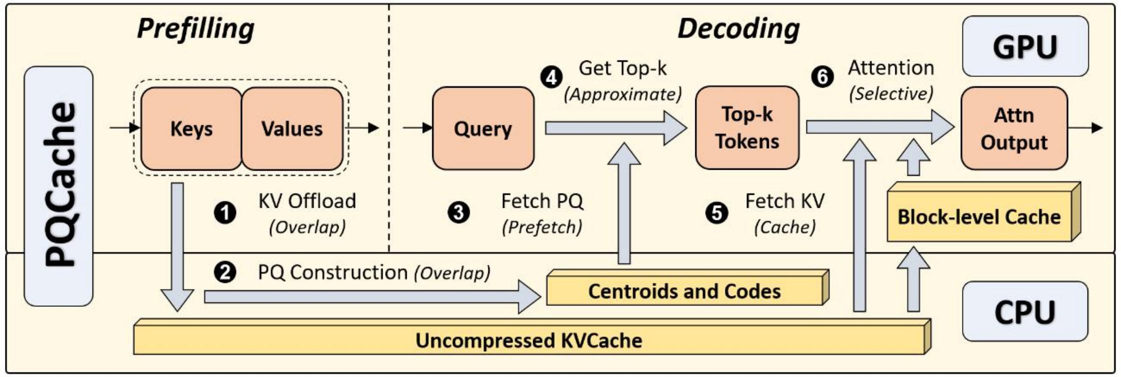
- **LLM Query Execution**: How to coordinate memory / compute resources?
  - Managing experts / low-rank adapters for **MoE & LoRA** (Model Offloading)
  - Integrating speculative drafters / small models for **SpecDec** (Model Management)
- **Data Structures + Algorithms**: How to design operators for modern hardware?
  - Heterogenous hardware; **CXL**; PIM (Processing-In-Memory) DRAM
- **Quantization**: How to effectively quantize weights / KV cache / activations?



Swappable Low-Rank (**LoRA**) adapters.
[Sheng et al ‘25 S-LoRA]



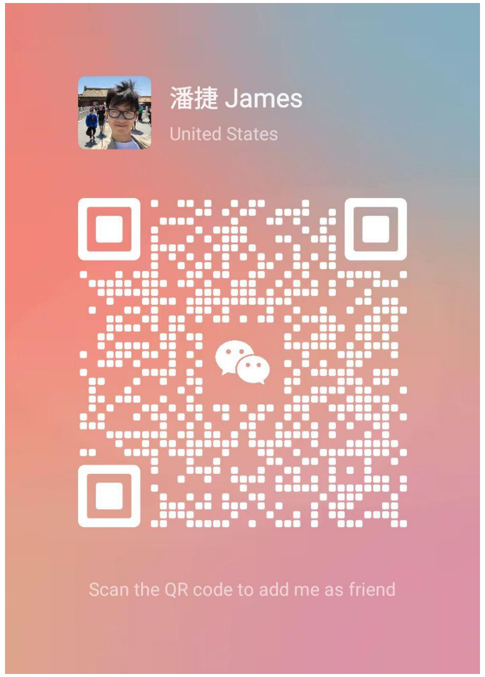
Softmax with **CXL**.
[Gu et al ‘25]



Product quantization KV compression.
[Zhang et al ‘25]

Thanks!

Survey of LLM Inference Systems arXiv:2506.21901



潘捷 James

United States

Scan the QR code to add me as friend

James Pan

jpan@tsinghua.edu.cn

Slides:

https://dbgroup.cs.tsinghua.edu.cn/ligl/activities.html