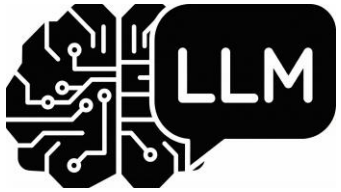
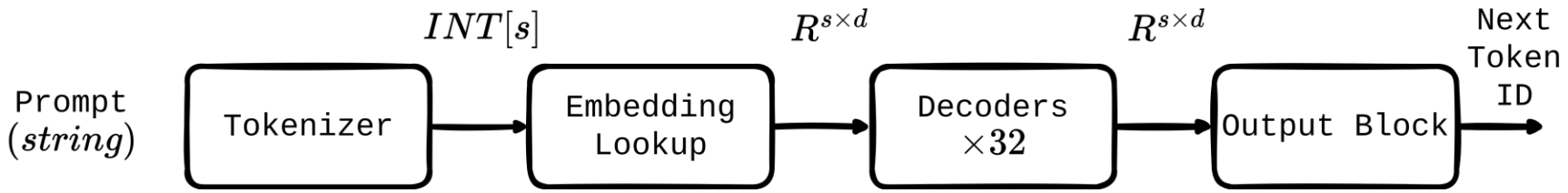
LLM Inference Lectures
Part 2: LLM Operators

(Harvard CS265 – DATA/AI Systems LAB)



Milad Rezaei Hajidehi
(milad@seas.harvard.edu)

Overview of an LLM



s: number of tokens in our prompt.
d: dimension of vectors we use for embeddings.

We Inspect All of Them!

Tokenizer

Goal:
Raw text -> Sequence of Discrete Tokens

"future belongs to" -> "future belongs to" -> [21733, 17623, 311]

Vocabulary (size: v):
List of all possible tokens.
Determined at the training time.

Coarse vs. Fine Grained Tokens?

future belongs to
future belongs to
future belongs to

Smaller Sequence Length.

Bigger Vocabulary Size. (Last Output Layer)

Weaker Generalization.

Byte-Pair Encoding

Vocabulary Size (v) = Hyperparameter
How to Construct Vocabulary Set?

Start with Each Byte as a Token
Greedily Merge Tokens and Add a New Token
Based on Frequencies.

be -> be

At Inference: Start with Bytes, Greedily Merge as Much as You Can.
(based on *merge-scores*)

128000 Tokens in LLAMA3.

At Inference: Start with Characters(Bytes), Greedily Merge as Much as You Can.
(based on *merge-scores*)

128000 Tokens in LLAMA3.

b_e_l_o_n_g -> be_l_o_n_g -> be_lo_n_g -> be_lon_g -> be_long

id->token

id	token
1	a
2	b
…	…
43700	long
…	…

token->id

token	id
a	1
aa	264
…	…
long	43700
…	…

<id,id> -> rank
(frequency)

<id,id>	rank
<1,2>	42
<1,3>	824
…	…
<43700,1>	9999
…	…

Embedding Lookup

Goal: Map Token IDs to Vectors with Semantic.

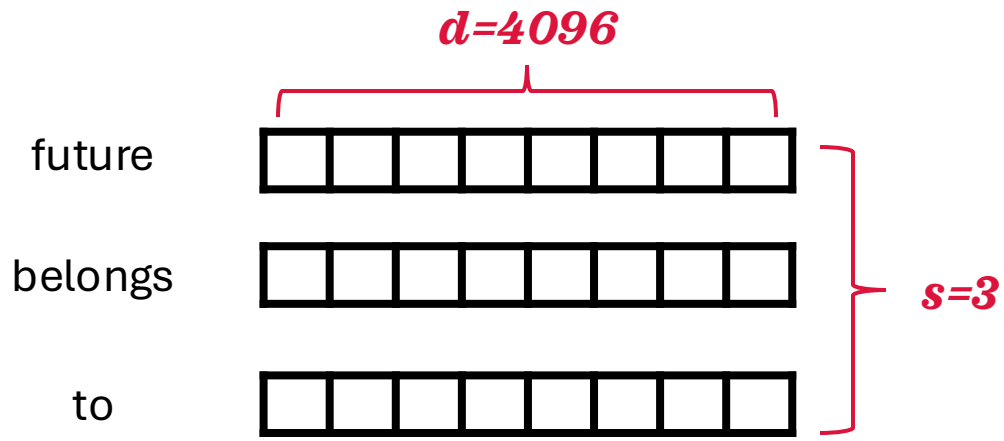
Embeddings: A Lookup Table of $(v \times d)$

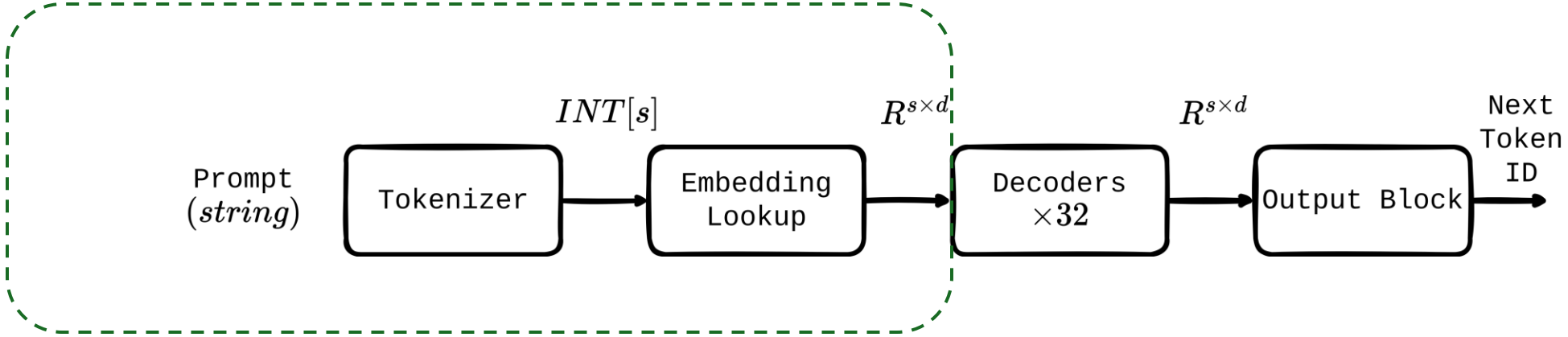
Afterward we have a $(s \times d)$ Tensor.

Where Should We Store Embedding Table?

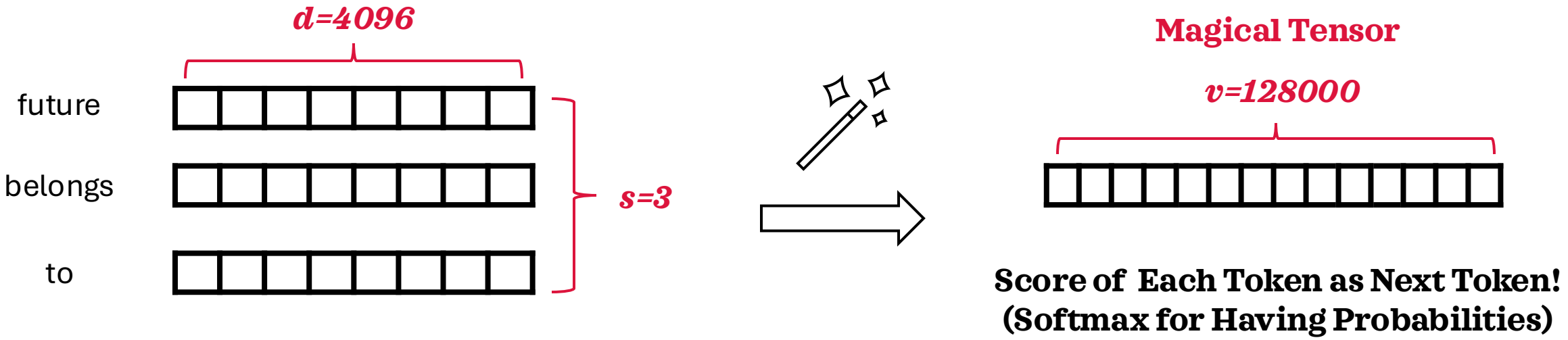
GPU!!
Avoid large tensor data-movement
Easier Duplicate Token Management

What We Have so Far...

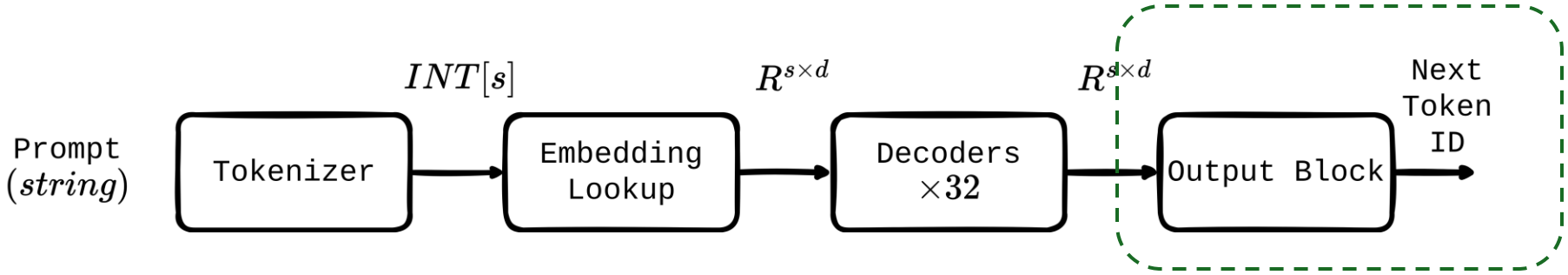


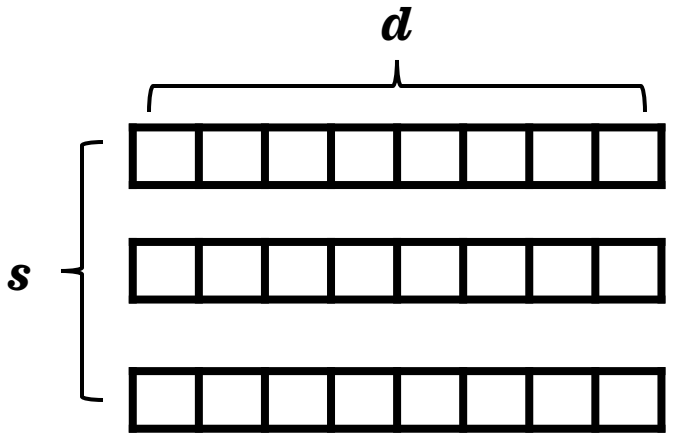


Our Goal: Best Next Token

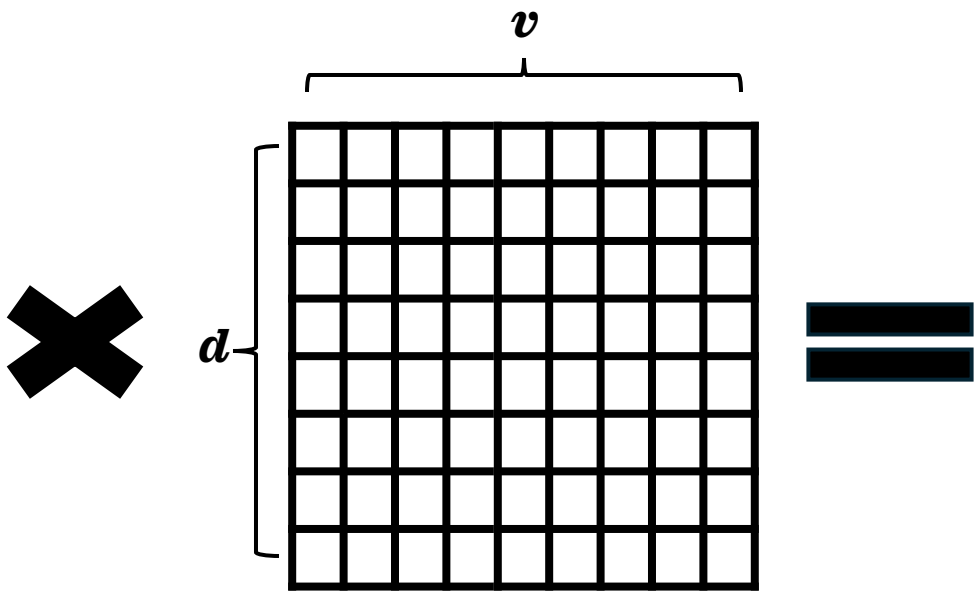


Output Block: Finding Magical Tensor

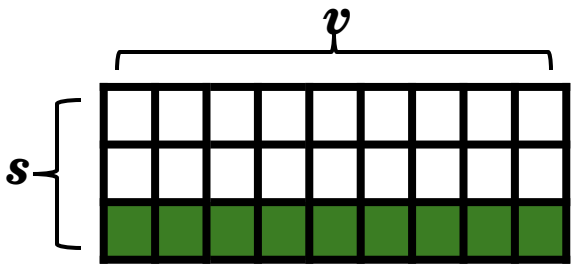




The <s,d> matrix after decoders



Learned Matrix in Output Block (Vocabulary Projection)



Last Tensor is Magical Tensor!

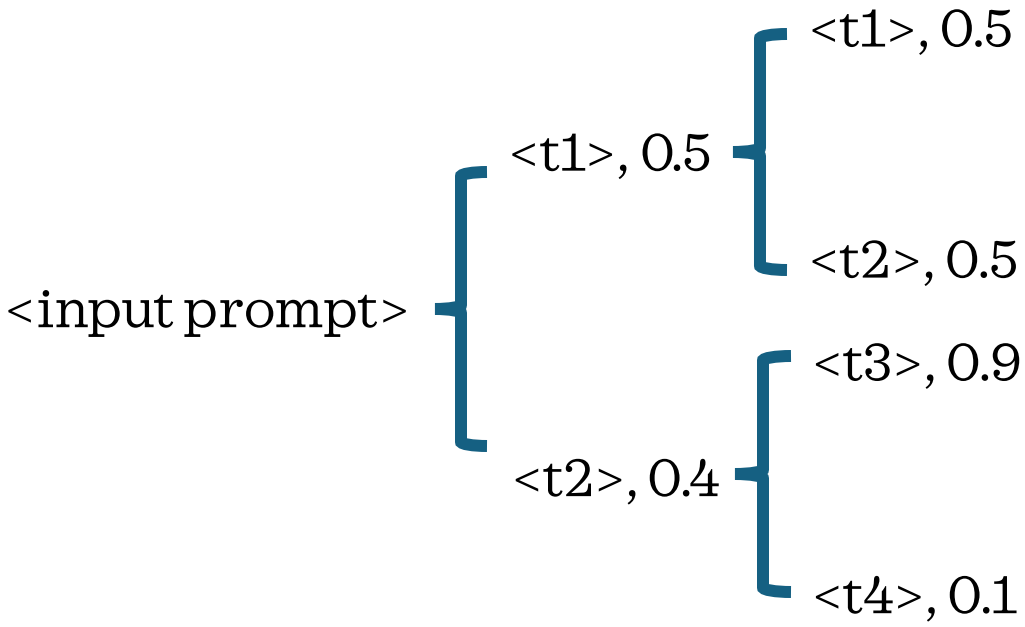
Softmax give us Probabilities.

DASlab @ Harvard SEAS

Decoding Algorithms: A Short Detour

Greedy Sampling: We always pick next token that has highest score/probability!

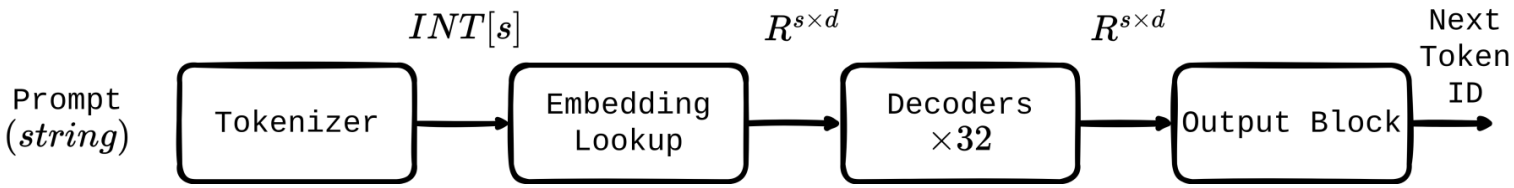
Is It Optimal?

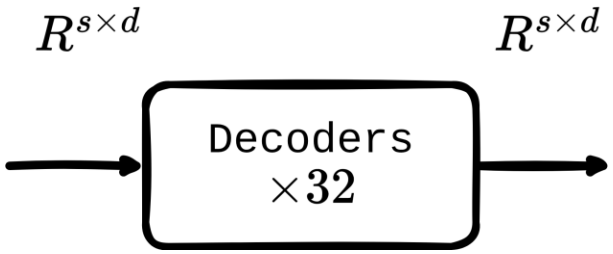


Issue #1: Diversity of Answers.

Issue #2: Locally Optimal != Globally Optimal

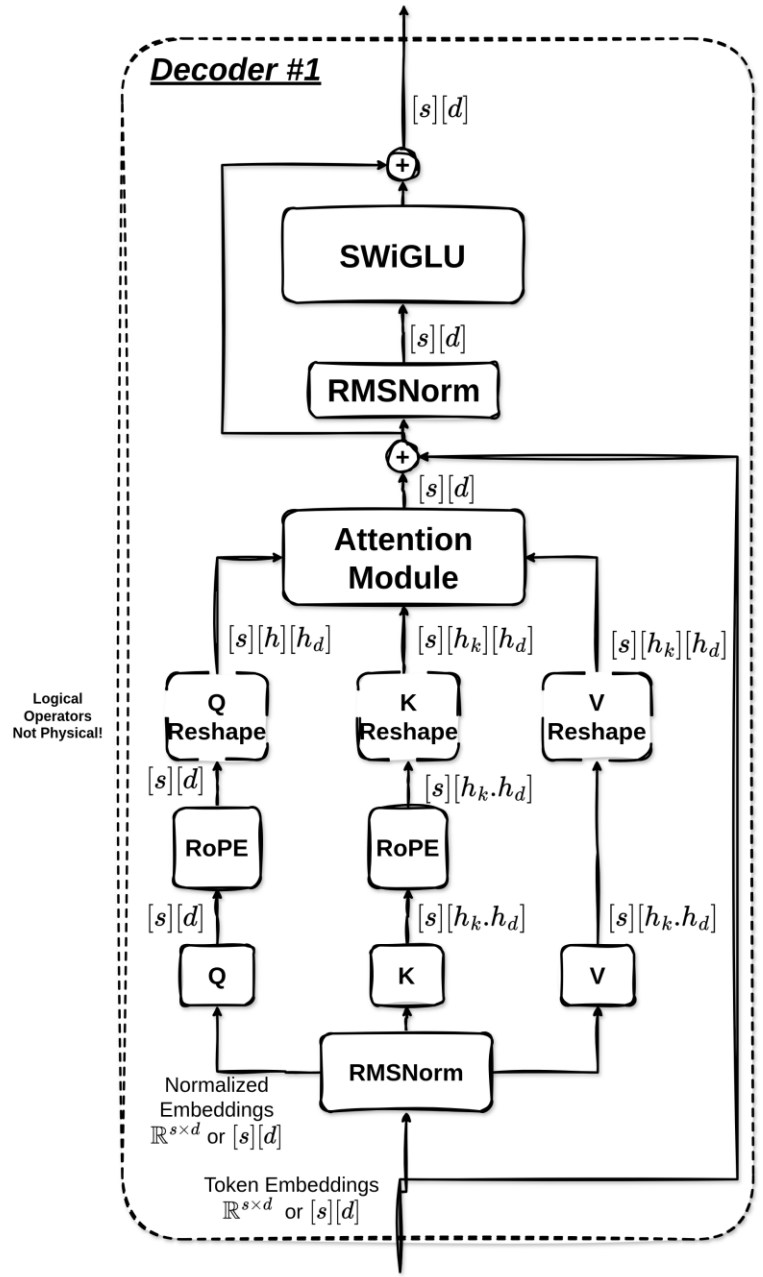
Beam Search & Temperature are Solution To This.

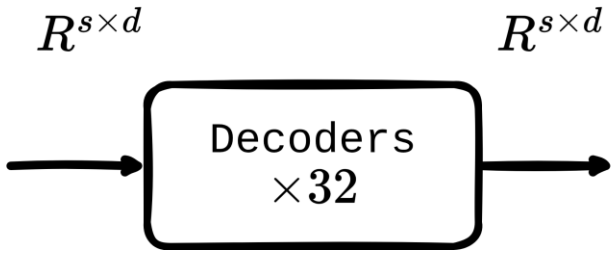




Last Missing Piece

Goal: Transform Initial Embedding -> Contextual Representation for Output Block





Operators Inside Decoder:

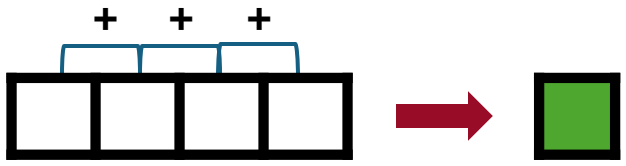
RMSNorm, QKV, KV-Cache, Attention, MLP
(90% of the Time)

$$\text{RMSNorm}(x) = \frac{x}{\text{RMS}(x)} \odot \gamma, \qquad \text{RMS}(x) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2 + \varepsilon}$$

x: Embedding Per Token, *s* Norms in Total

Purpose: Normalize Layer Inputs, Faster Convergence, Stability

Requires A Reduction (sum) Kernel!



How to Calculate Sum of a Vector? (Threads are Independent!)

Step1: Each Thread Calculate Sum for its Own Color

Per Thread Sum in Shared Memory

T_id	Color	Sum
0		17
1		29
2		14
4		11

Sync. Barrier

Second ½ of Threads Finish, First ½ Threads Merge With Second Half Values (0 with 2, 1 with 4)

Sync. Barrier

Second 1/4 of Threads Finish, First 1/4 Threads Merge With Second Half Values (0 with 1)

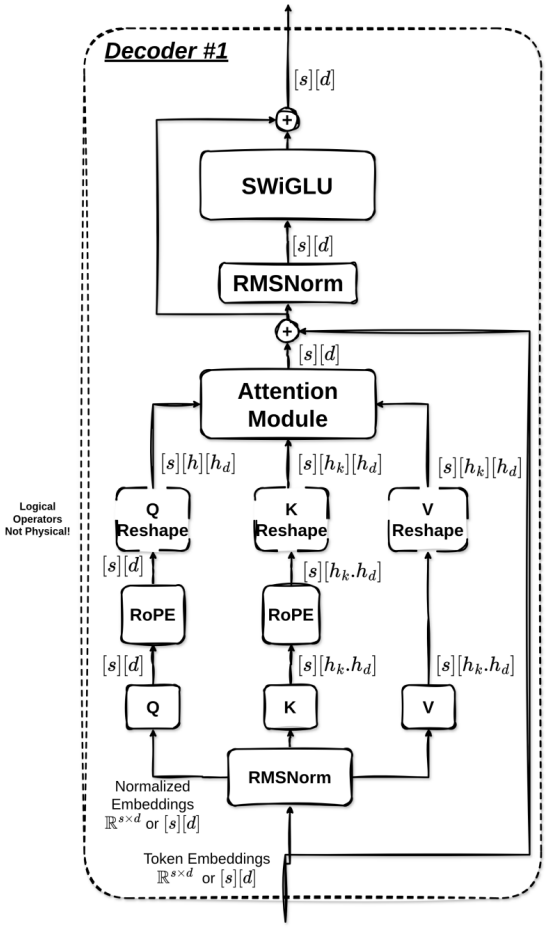
……

Original Goal of Decoder: Transforming Embeddings to be *Contextual* Information.

"future belongs to self-"
- "motivated" -> 70%
- "driving" -> 29%
- "designing" -> 1%

"**elon said** future belongs to self-"
- "motivated" -> 5%
- "driving" -> 95%
- "designing" -> 0%

"**stratos said** future belongs to self-"
- "motivated" -> 10%
- "driving" -> 0%
- "designing" -> 90%



DASlab @ Harvard SEAS

Input of Decoder: One embedding per token.

Output of Decoder/Attention(*H*): One embedding per token. But transformed!

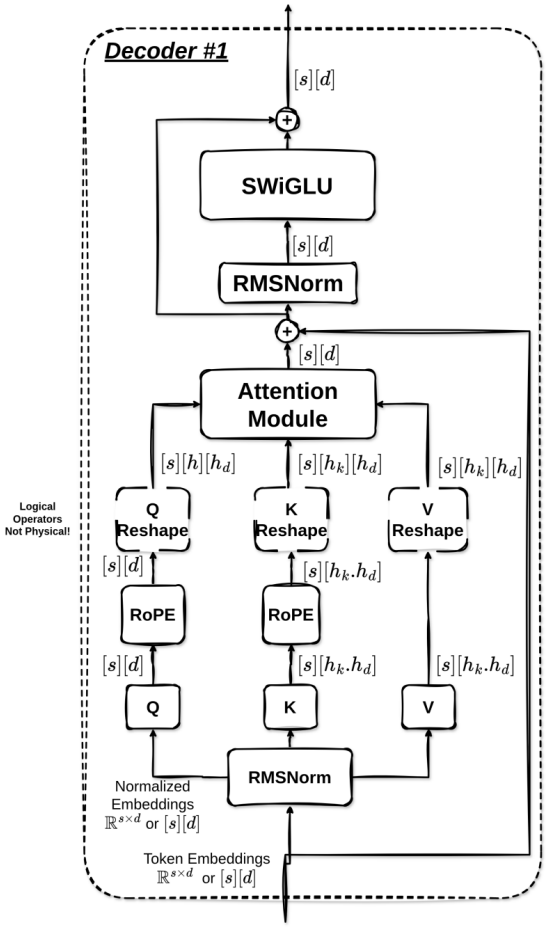
based on context and previous tokens!

(K)ey: What kind of query would match me?

(Q)uery: What am I looking for in previous tokens?

Token

(V)alue: What information do I pass if selected?



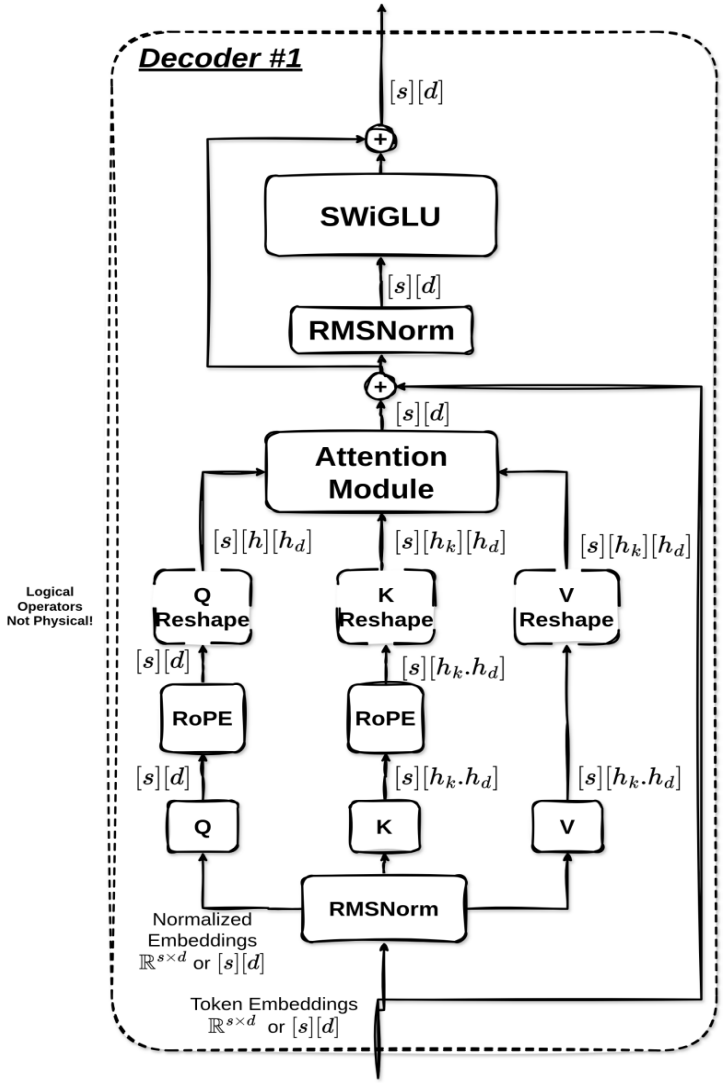
DASlab @ Harvard SEAS

How much j can impact i? (normalized)

$$\alpha_{ij} = \frac{\exp\left(\frac{Q_i K_j^\top}{\sqrt{128}}\right)}{\sum_k \exp\left(\frac{Q_i K_k^\top}{\sqrt{128}}\right)}$$

Embeddings Transformed.

$$\text{Attention}(Q, K, V) = AV$$



!

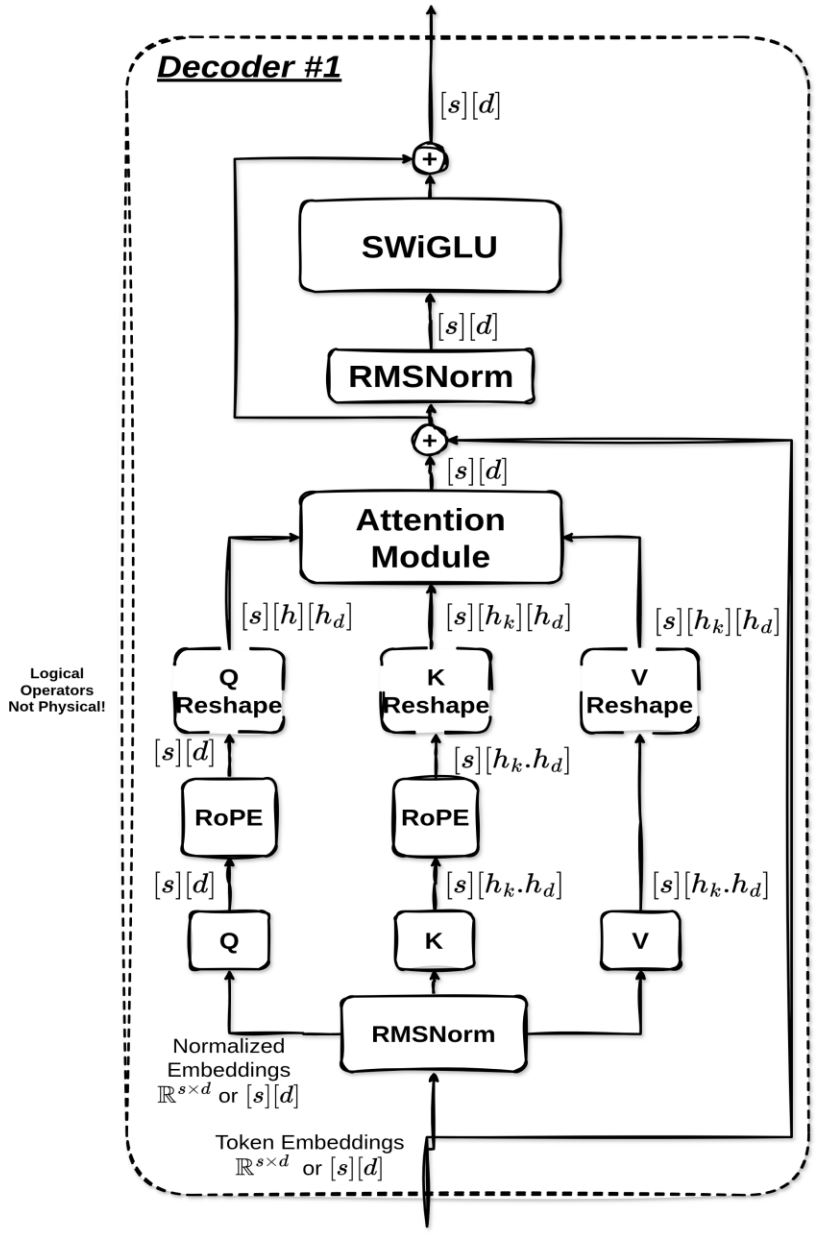
In LLAMA3, We use GQA rather than Vanilla-Transformer, which is slightly different.

We need QKV for all tokens!

After first decoder, all QKVs are affected by their previous tokens!

KV-Cache:
Cache previously computed K,V to avoid recomputing them at every decoding step.

After QKV and Attention we have bunch of Matrix Multiplication and elementwise operators.



You Are Ready For The Project!

Step 0	Step 1	Step 2	Step 3
Read CUDA Book (selected chapters)	Finish Tokenizer	Write a good generic Matrix Multiplication kernel.	Write your Controllers (operators)
Understand LLAMA Operators. (Slides, Office Hours, Youtube, LLM, Torch Codes)	Write dumper.py and loader.cpp to transform weights. (mmap)	Write a Reduce kernel.	Write rest of easy kernels.
	Load embeddings and prepare the input for first Kernel (RMSNorm)		

Test, Test, Test!

DASlab @ Harvard SEAS

You Are Ready For The Project!

Step 0	Step 1	Step 2	Step 3
Codes)	Load embeddings and prepare the input for first Kernel (RMSNorm)		Write rest of easy kernels.

First Make it Work.
Then Optimizations:
KV-Cache, Fusing, Performance Tuning, Quantization....

Test, Test, Test!

DASlab @ Harvard SEAS

Challenges & Bottlenecks.
What Makes Inference Hard?

✓ **Opportunities For Problem-Solving!**

Performance Metrics in Inference

Phase 1 – Prefill: Reading prompt – Compute QKVs for each token – Find the first "next token"
Metric -> Time-to-First-Token (TTFT) = T_queue + T_prefill
Compute Bound!

Phase2 – Decode: Predict next tokens one by one – Read QKV from cache.
Metric -> Tokens Decoded per Second (token/s) or Latency = TTFT + Full Decode Latency
Memory Bound!

Inference Challenges

Highly variable sequence lengths (prompt, answer)

Prompt Tokens	Output Tokens
10	300
170	289
16	29
290	46

Why It Cause a Problem?

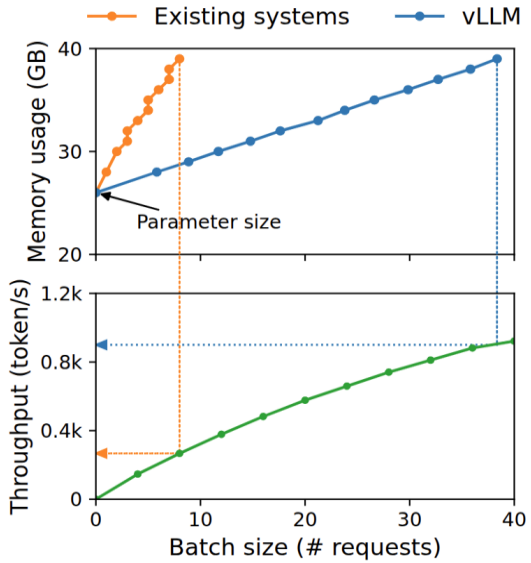
Inference Challenges

Highly variable sequence lengths (prompt, answer)

Memory is Under Pressure.

For a 500 token request:
500 * 32 * 2 * 4096 * 4b ~= **500MB**!!

GPU Memory



Inference Challenges

Highly variable sequence lengths (prompt, answer)

Memory is Under Pressure.

Too Many Operators/Kernel Launches!

Why It Cause a Problem?

Inference Challenges

- Highly variable sequence lengths (prompt, answer)
- Memory is Under Pressure.
- Too Many Operators/Kernel Launches!
- **Bursty, Irregular, On/Off Workload**

And of Course Everything We Said About **Kernel Tuning.**

DASlab @ Harvard SEAS

Every Paper We Read:
A Solution to These Challenges.
General or Application-Specific

Inference Challenges

Too Many Operators/Kernel Launches!

Bursty, Irregular, On/Off Workload

And of Course Everything We Said About Kernel Tuning.

Problem: We launch many kernels for one inference.
~25 kernel launch per decoder if done naively.

Why We Care: GPU Underutilization.
Low throughput due to frequent data movement.

FLASHATTENTION: Fast and Memory-Efficient Exact Attention with IO-Awareness

Tri Dao[†], Daniel Y. Fu[†], Stefano Ermon[†], Atri Rudra[‡], and Christopher Ré[†]

Mirage: A Multi-Level Superoptimizer for Tensor Programs

Mengdi Wu Xinhao Cheng Shengyu Liu[†] Chunan Shi[†] Jianan Ji
Man Kit Ao Praveen Velliengiri[‡] Xupeng Miao[♯] Oded Padon[◇] Zhihao Jia

Problem: Many design-decisions and parameters for each kernel.

Why We Care: Optimizing kernels independently is necessary for end-to-end optimization of LLM.

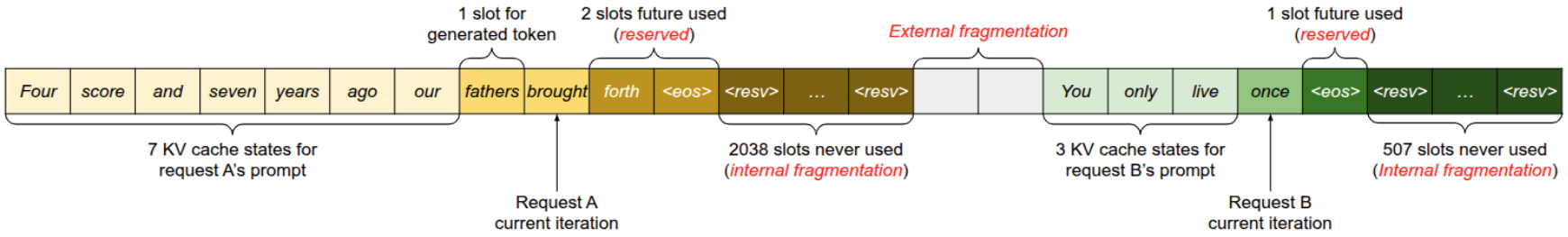
TVM: An Automated End-to-End Optimizing Compiler for Deep Learning

Tianqi Chen[1], Thierry Moreau[1], Ziheng Jiang[1,2], Lianmin Zheng[3], Eddie Yan[1]

Meghan Cowan[1], Haichen Shen[1], Leyuan Wang[4,2], Yuwei Hu[5], Luis Ceze[1], Carlos Guestrin[1], Arvind Krishnamurthy[1]

Problem: Naive KV cache allocation wastes GPU memory.

Why We Care: Lack of memory limit batch-size and hence throughput.



Efficient Memory Management for Large Language Model Serving with *PagedAttention*

Woosuk Kwon[1,*] Zhuohan Li[1,*] Siyuan Zhuang[1] Ying Sheng[1,2] Lianmin Zheng[1] Cody Hao Yu[3]
Joseph E. Gonzalez[1] Hao Zhang[4] Ion Stoica[1]

Problem: Naive KV cache allocation wastes GPU memory.

And Many More Papers....

Efficient Memory Management for Large Language Model Serving with *PagedAttention*

Woosuk Kwon[1,*] Zhuohan Li[1,*] Siyuan Zhuang[1] Ying Sheng[1,2] Lianmin Zheng[1] Cody Hao Yu[3]
Joseph E. Gonzalez[1] Hao Zhang[4] Ion Stoica[1]

DASlab @ Harvard SEAS

How to Assess LLM Inference Techniques?

Adaptation Overhead
- Pretraining (Grouped Query Attention)
- Fine-Tuning (Quantization Methods)
- No Overhead! (vLLM)

Fidelity
- Lossy
- Lossless

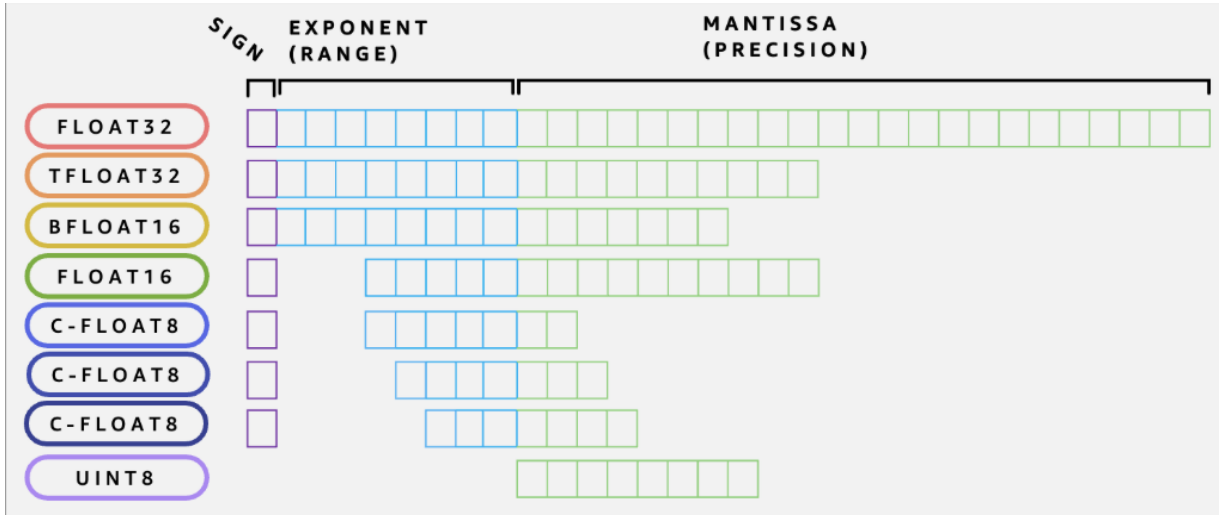
What is Sacrificed
- Latency (mean)
- Throughput
- p99, Tail-Latency

Generality!

DASlab @ Harvard SEAS

Quantization

Using Less Bits for Weights, Activations, KV-Cache -> **More Memory, More Batch Size, More Throughput**



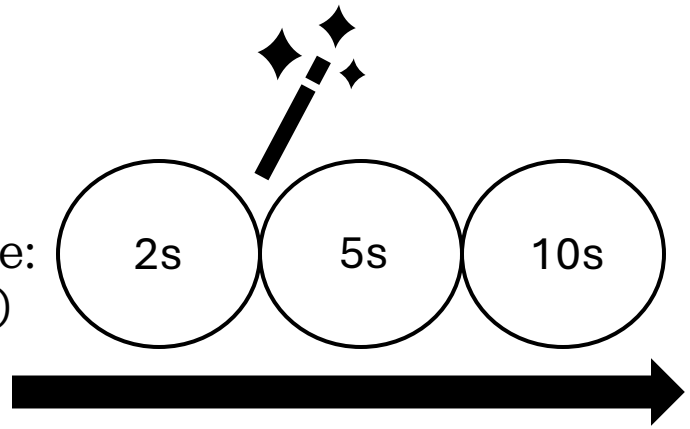
$$x = (-1)^{\text{sign}} \times (1 + \text{mantissa}) \times 2^{(\text{exponent} - \text{bias})}$$

Format	Smallest	Largest	Machine Precision
FP32	1.175×10^{-38}	3.403×10^{38}	$2^{-23} \approx 1.19 \times 10^{-7}$
FP16	6.104×10^{-5}	6.550×10^{4}	$2^{-10} \approx 9.77 \times 10^{-4}$
BF16	1.175×10^{-38}	3.390×10^{38}	$2^{-7} \approx 7.81 \times 10^{-3}$
FP8 (E4M3)	1.5625×10^{-2}	2.40×10^{2}	$2^{-3} = 1.25 \times 10^{-1}$
FP8 (E5M2)	6.104×10^{-5}	5.734×10^{4}	$2^{-2} = 2.5 \times 10^{-1}$

Possible Problems:
Overflow, Underflow, Low Precision

Lossy, May Require Fine-tuning.

Scheduling Requests in LLM



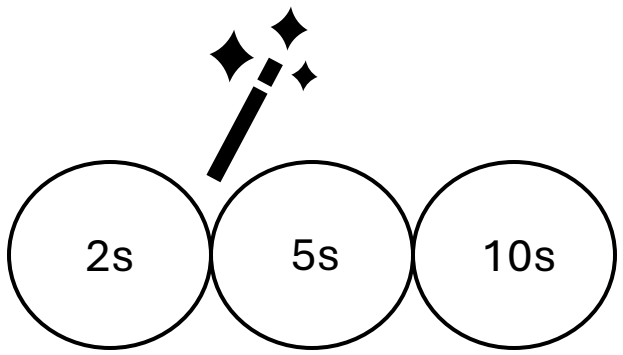
Original Order: 2s 5s 10s

Re-order: 10s 5s 2s

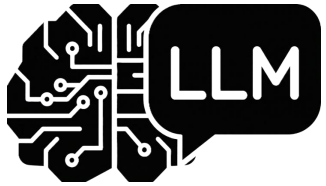
? P99, Throughput, Chicken-Egg Problem

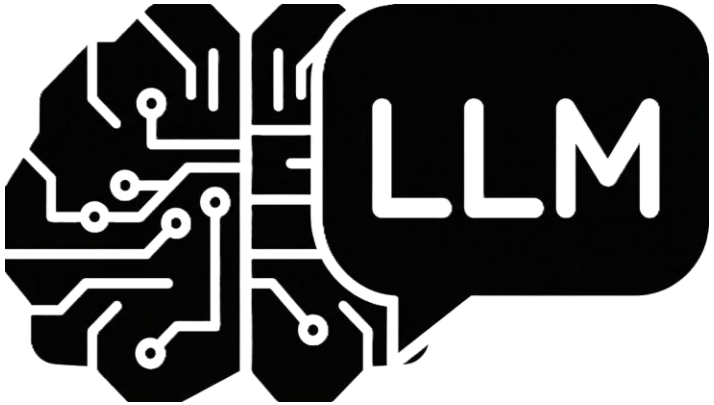
✓ Can Improve Mean Latency.
Loseless, Only Train the Magical Part!

Routing Requests in LLM Inference



?





✓ Improve Everything but Lossy!