Making Inference Faster

Zhenyan(Luke) Zhu, Daniel Hou, Oh Jun Kweon

Background: Models are getting Larger!

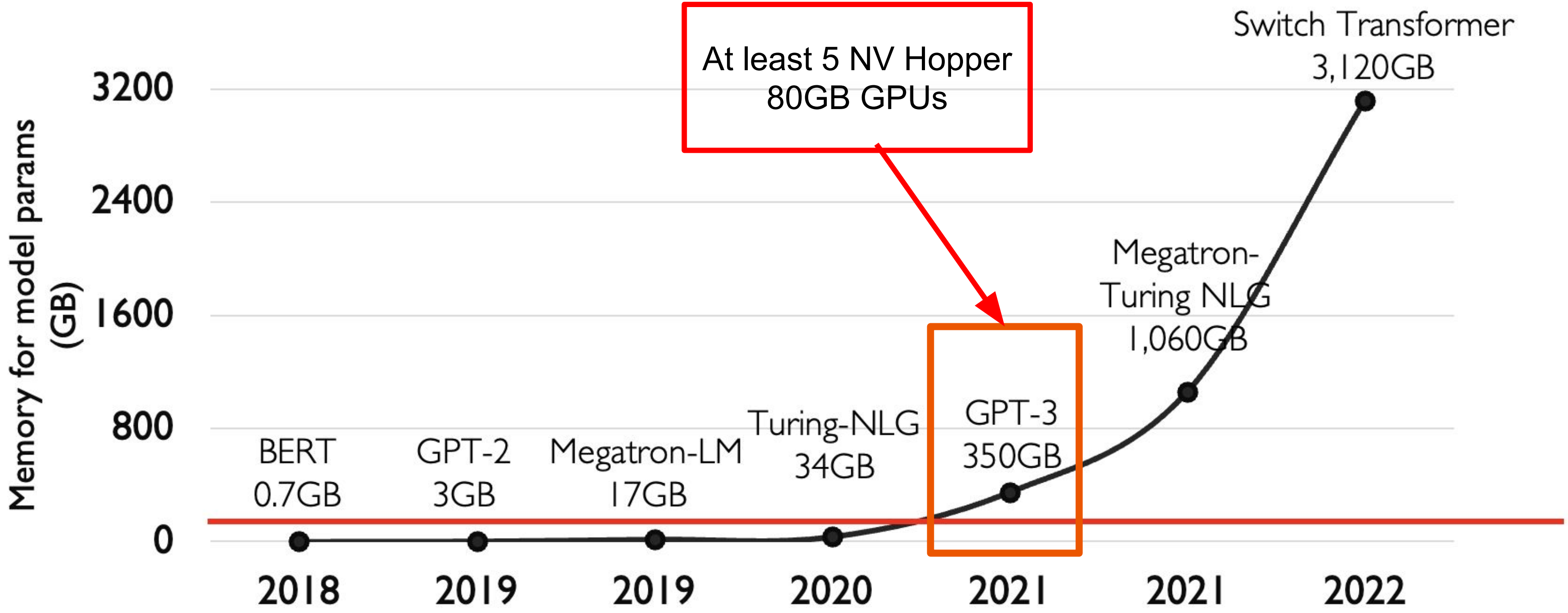
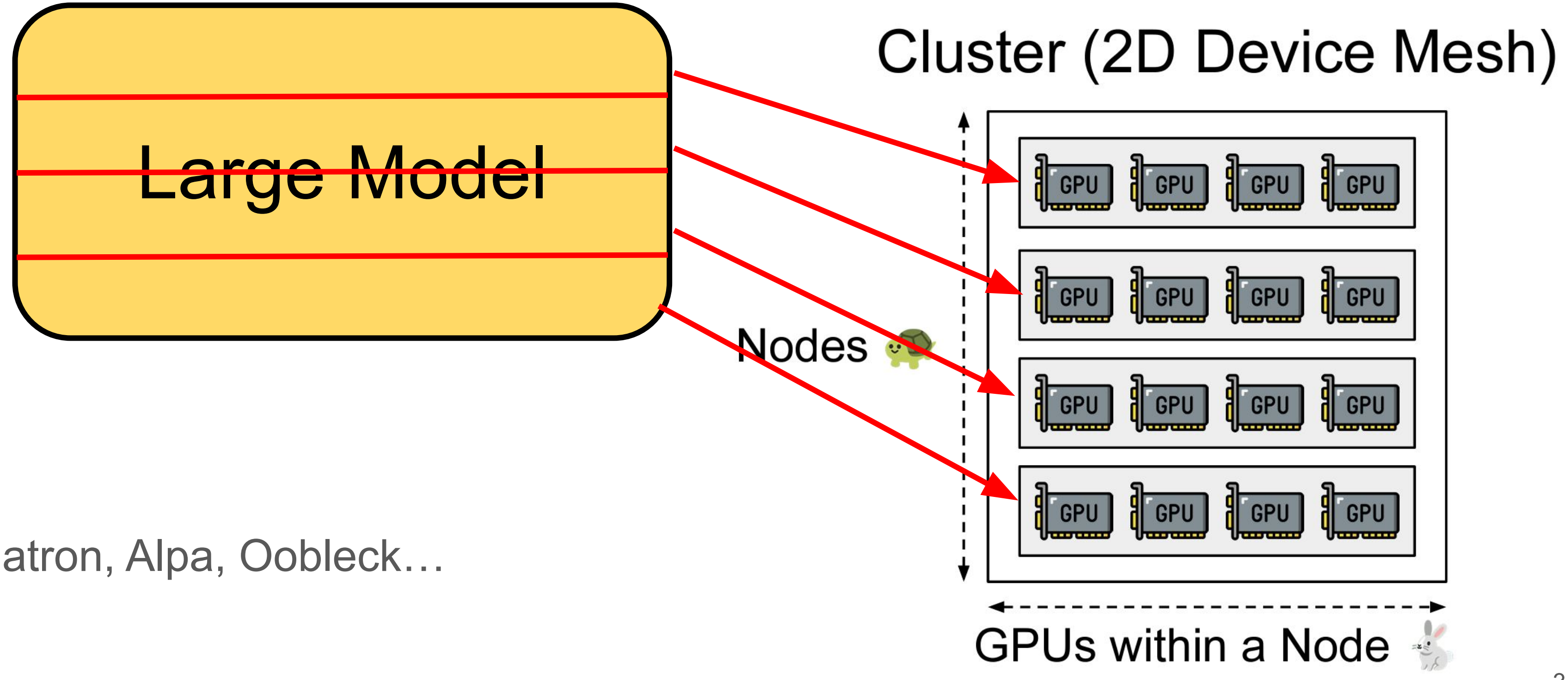


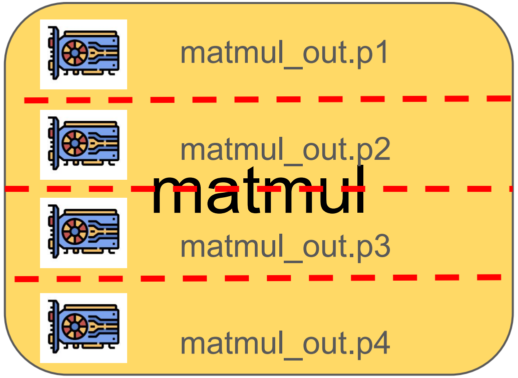
Image source:https://github.com/mosharaf/eecs598/blob/w24-genai/Slides/020624-alpa-oobleck-Insu.pptx

Background: Need to partition the model



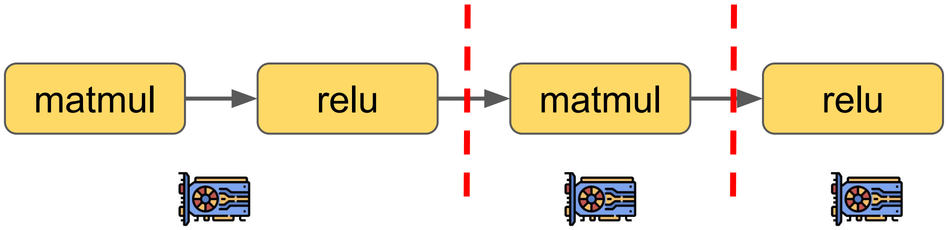
Megatron, Alpa, Oobleck…

Background: recap on tensor/pipeline level parallelism



Tensor level parallelism: partition a layer to multiple devices

Background: recap on tensor/pipeline level parallelism



Pipeline level parallelism: partition a model to different stages, assign stages to devices.

Terminology

We use model level parallelism to refer both Tensor-level parallelism and Pipeline-level parallelism.

Background: Training VS Serving

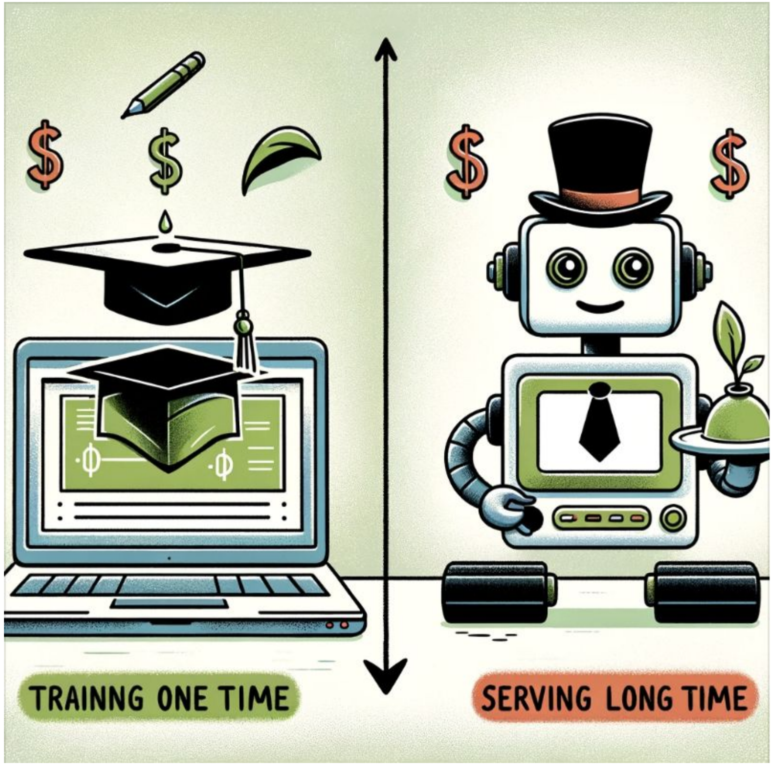


Image source: Chatgpt

AlpaServe: Statistical Multiplexing with Model Parallelism for Deep Learning Serving

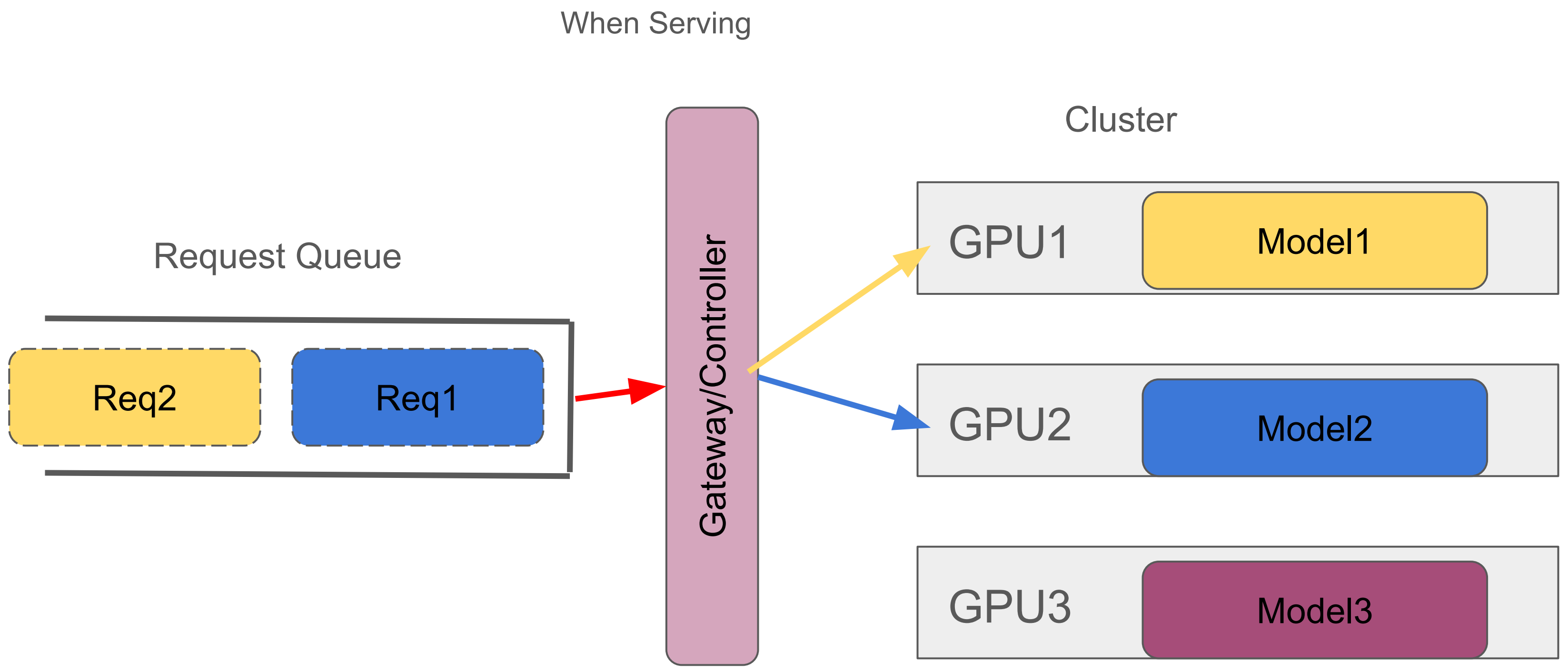
Zhuohan Li, UC Berkeley, Lianmin Zheng, UC Berkeley, Yinmin Zhong, Peking University, Vincent Liu, University of Pennsylvania, Ying Sheng, Stanford University, Xin Jin, Peking University, Yanping Huang, Google, Zhifeng Chen, Google, Hao Zhang

Presenter: Zhenyan(Luke) Zhu

Li, Z., Zheng, L., Zhong, Y., Liu, V., Sheng, Y., Jin, X., Huang, Y., Chen, Z., Zhang, H., Gonzalez, J. E., & Stoica, I. (2023). AlpaServe: Statistical Multiplexing with Model Parallelism for Deep Learning Serving. In 17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23) (pp. 663-679). USENIX Association. https://www.usenix.org/conference/osdi23/presentation/li-zhouhan

Contents:

Background: simple placement



Background: Serving Request Patterns

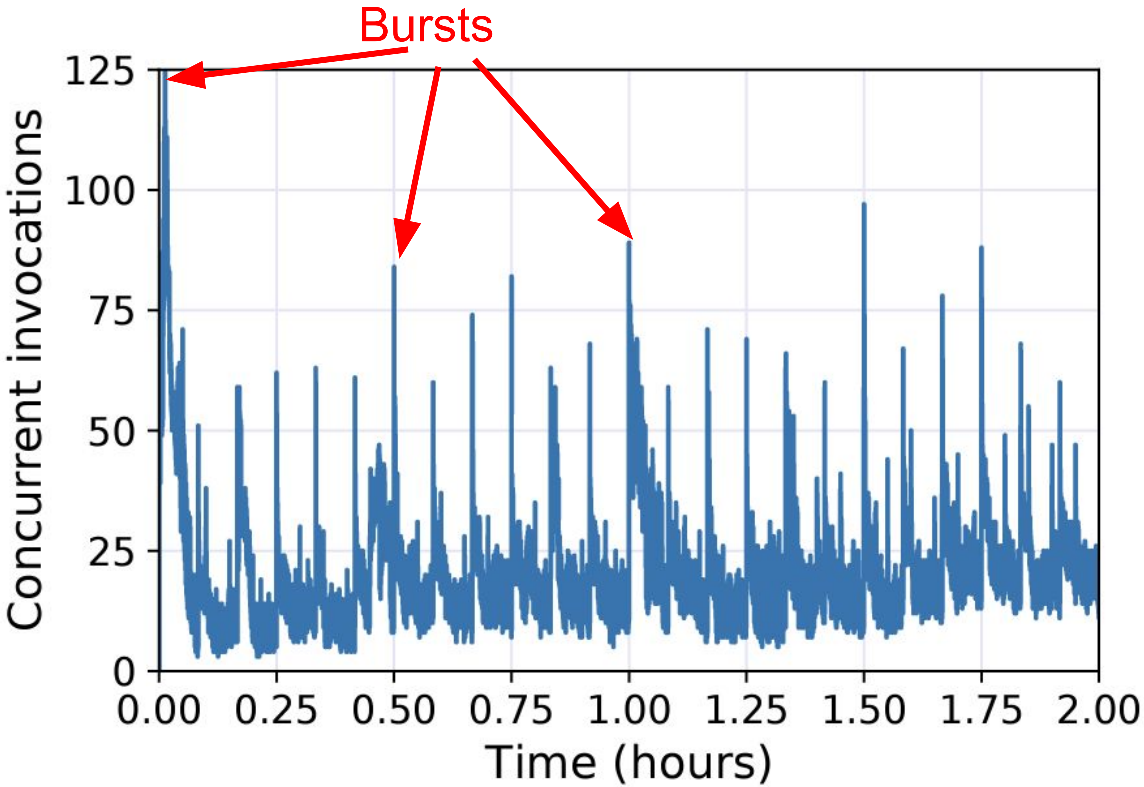
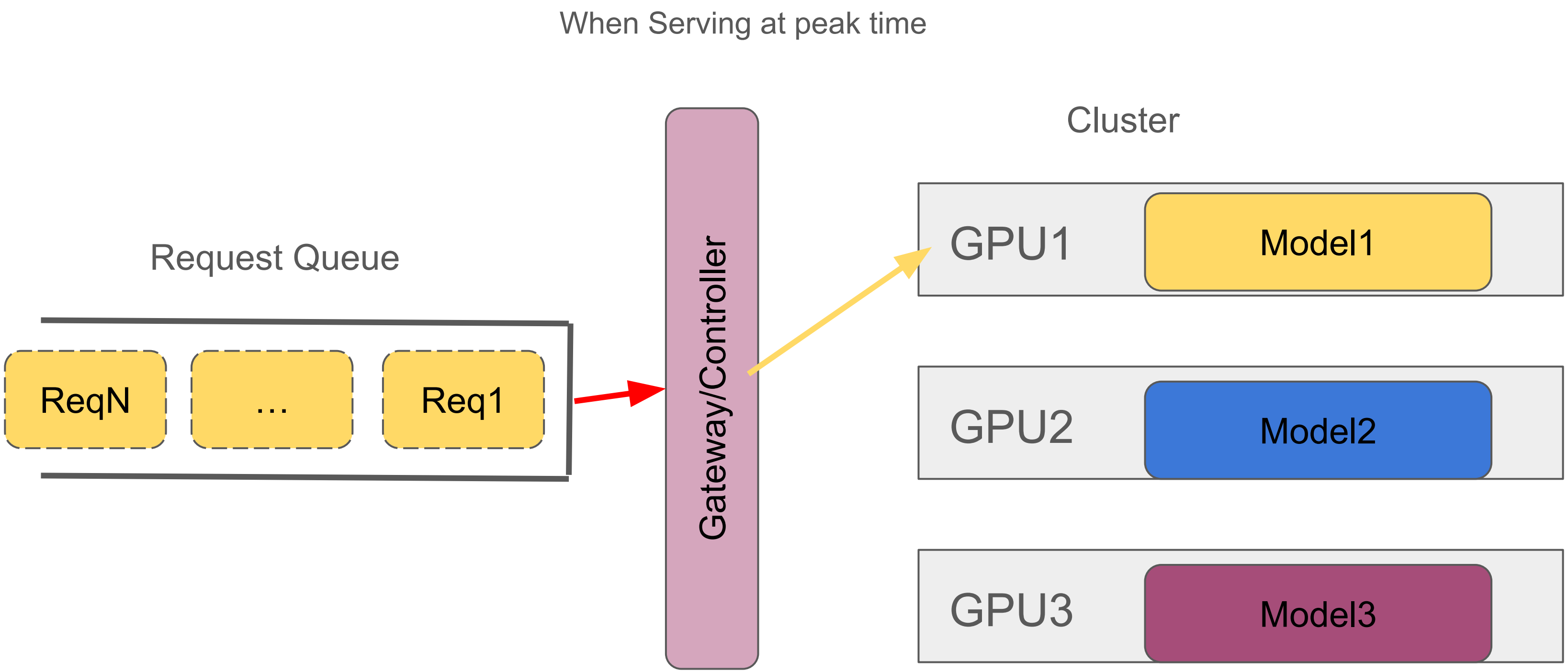
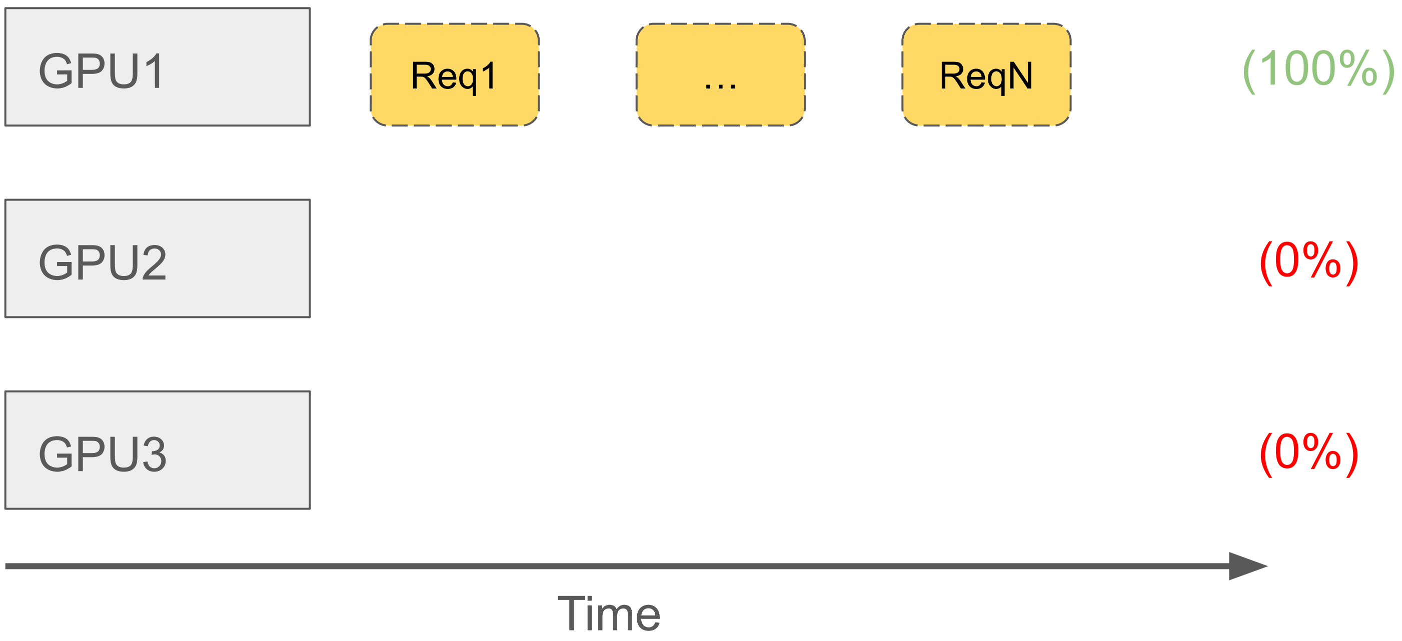


Image source: Zhang, Yanqi, et al. “Faster and Cheaper Serverless Computing on Harvested Resources”

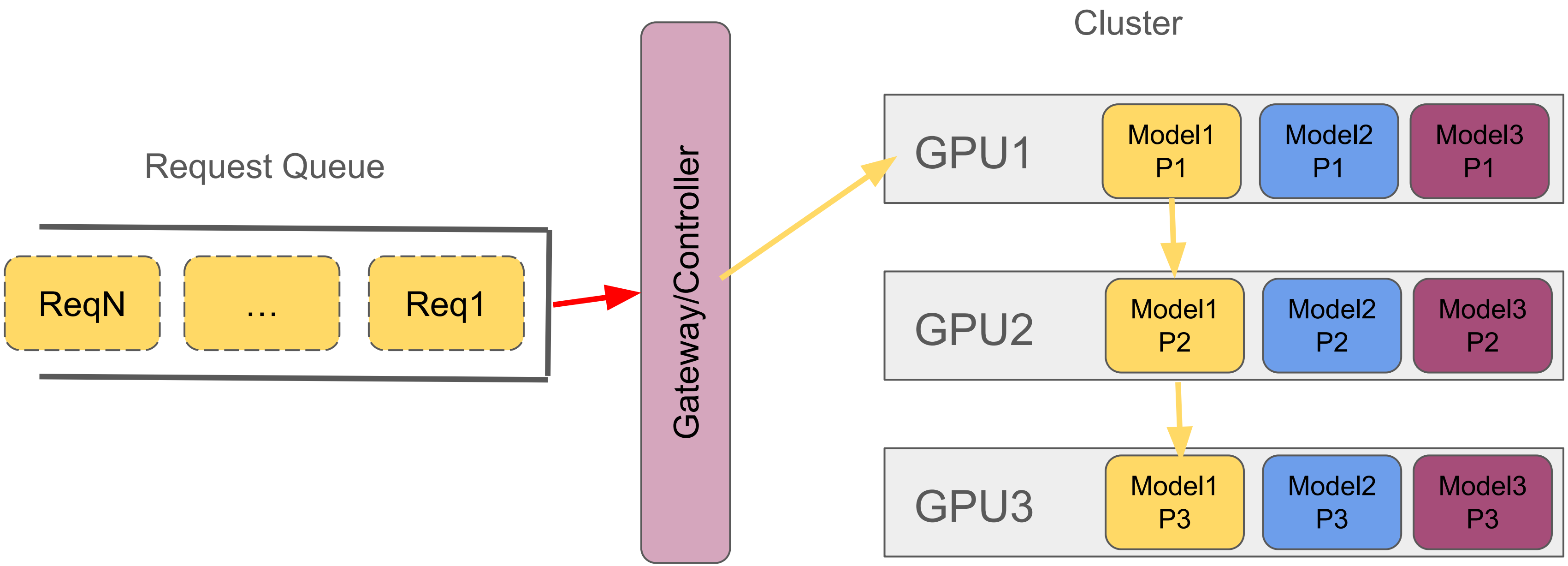
Background: Real-world Serving Pattern



Background: The Problem

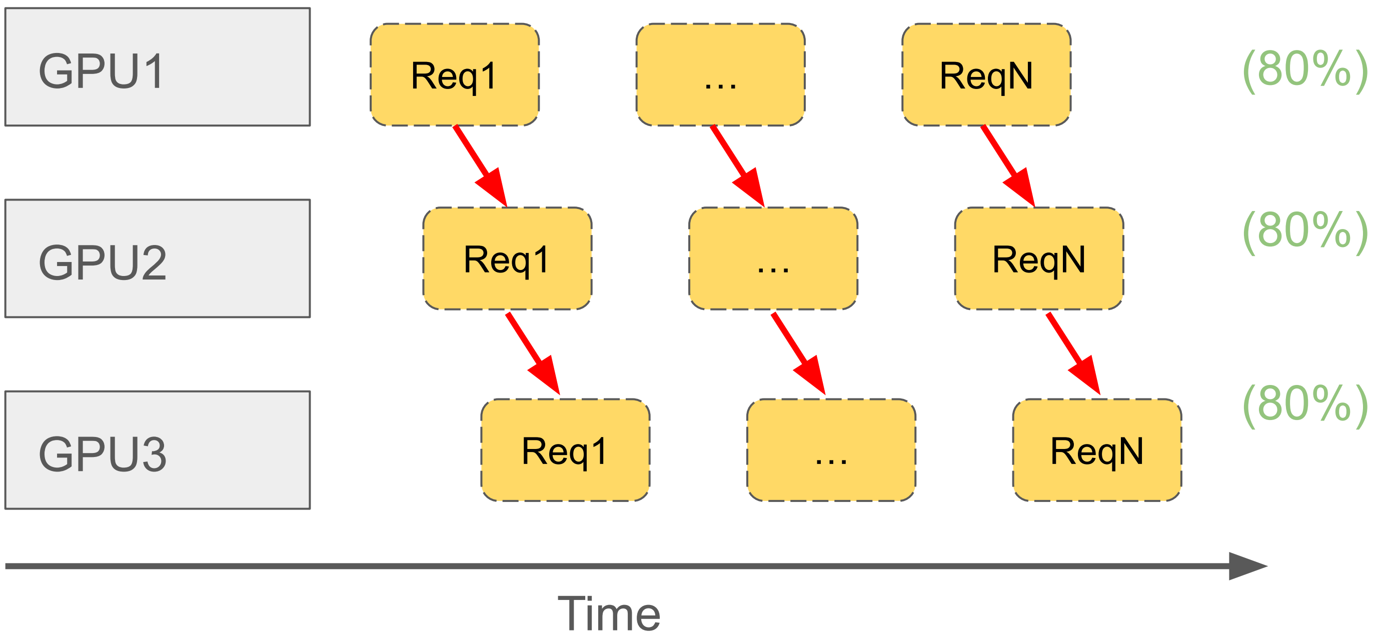


Motivation: Multiplex the GPU fully utilize resource



Motivation: Multiplex the GPU fully utilize resource

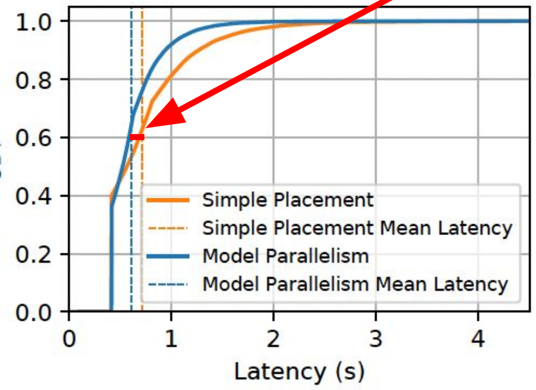
≈ 3x throughput!



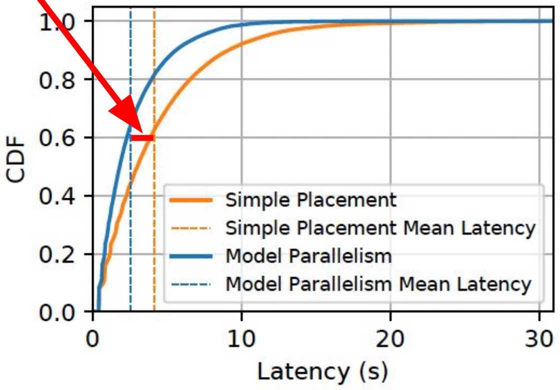
2 Model Example Comparison

2 Transformer Model(13.4GB)
2 16GB GPU

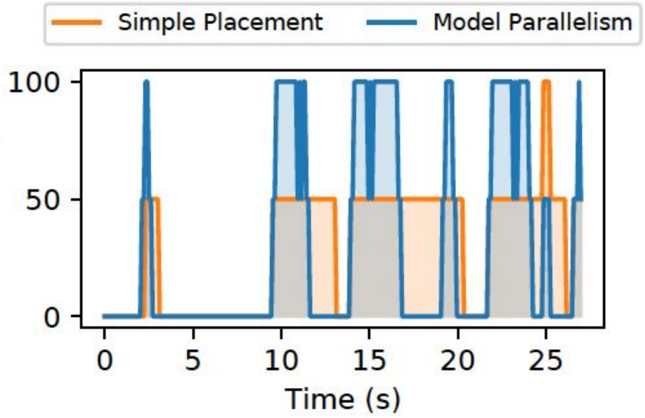
Model Parallelism outperforms greatly if request are brusty



(a) Poisson arrival.

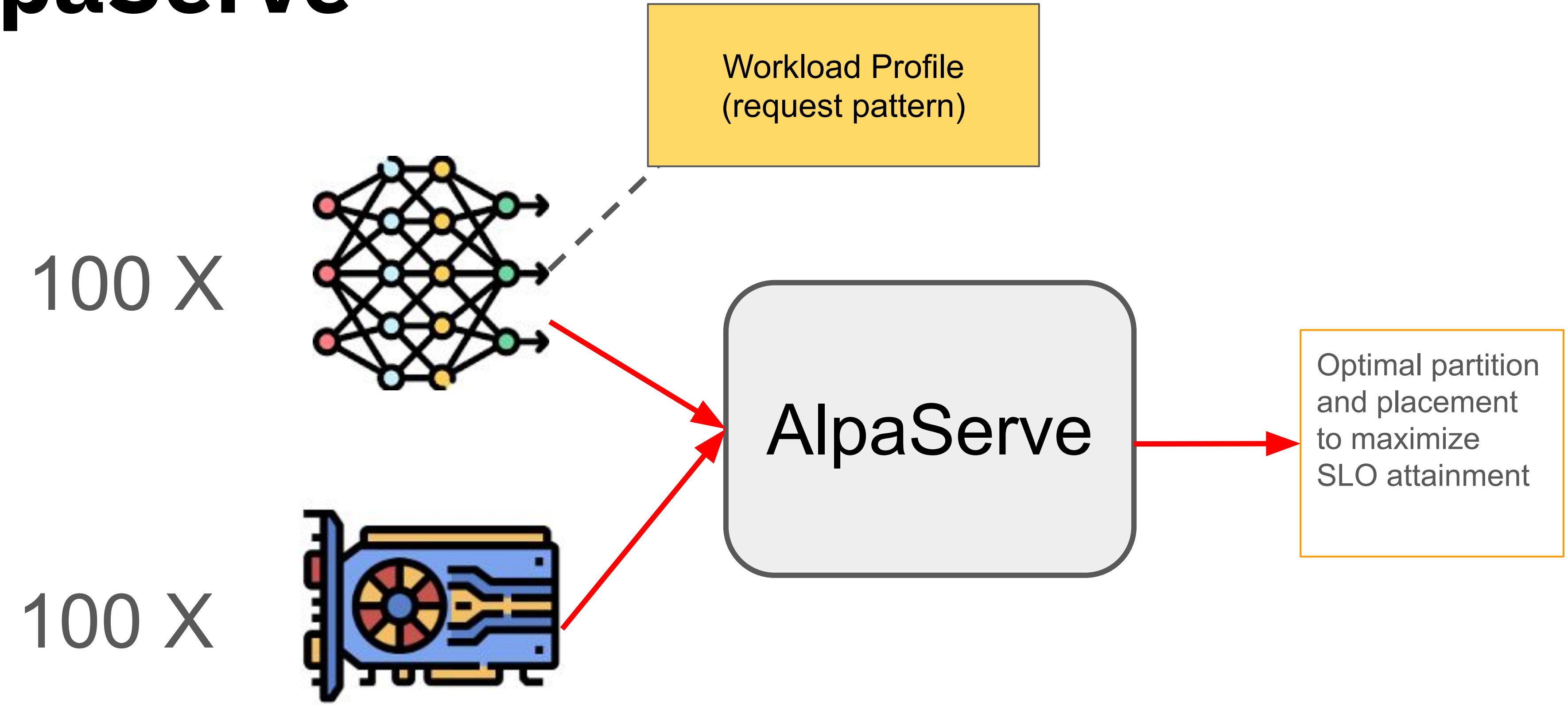


(b) High CV Gamma arrival.



(d) Cluster utilization.

AlpaServe

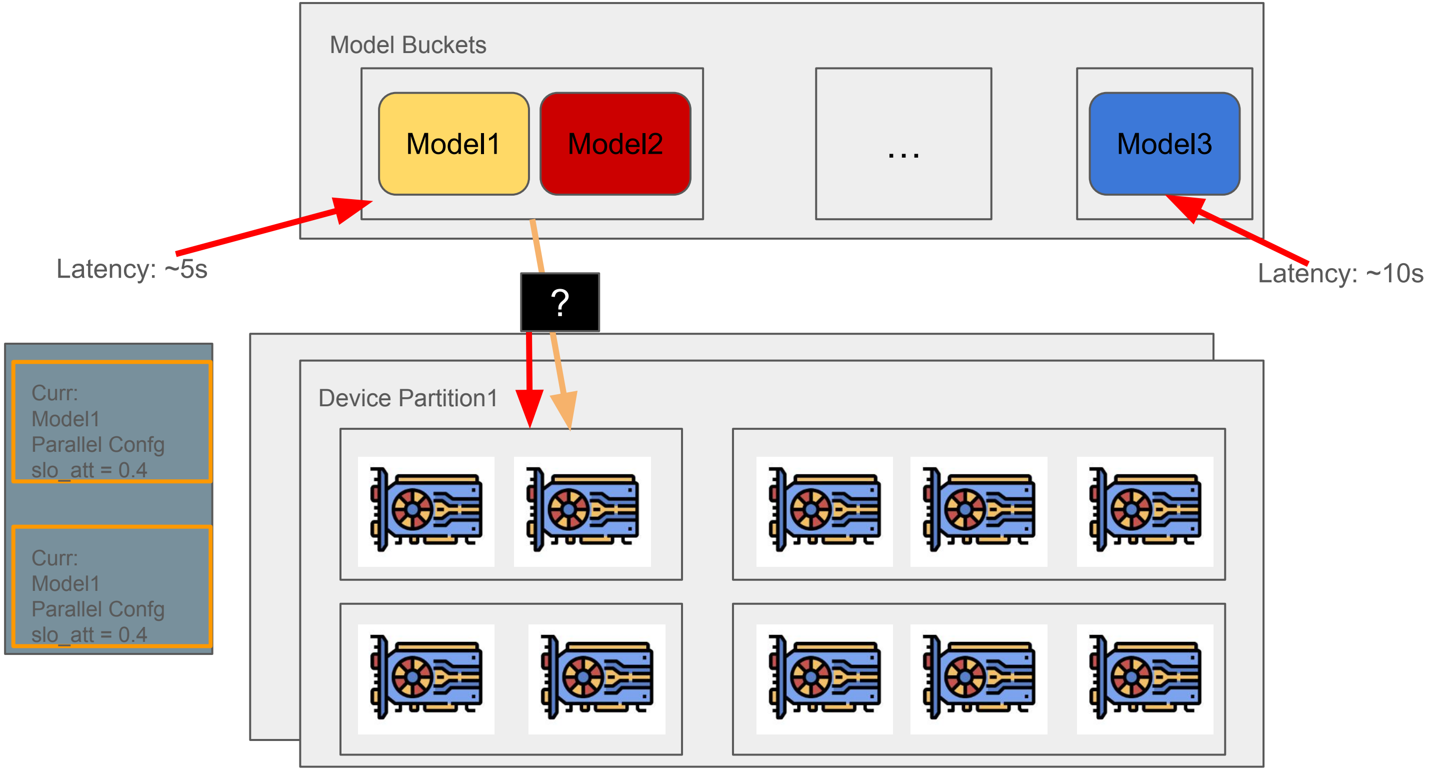

100 X
100 X
Workload Profile
(request pattern)
AlpaServe
Optimal partition
and placement
to maximize
SLO attainment

What is SLO attainment?

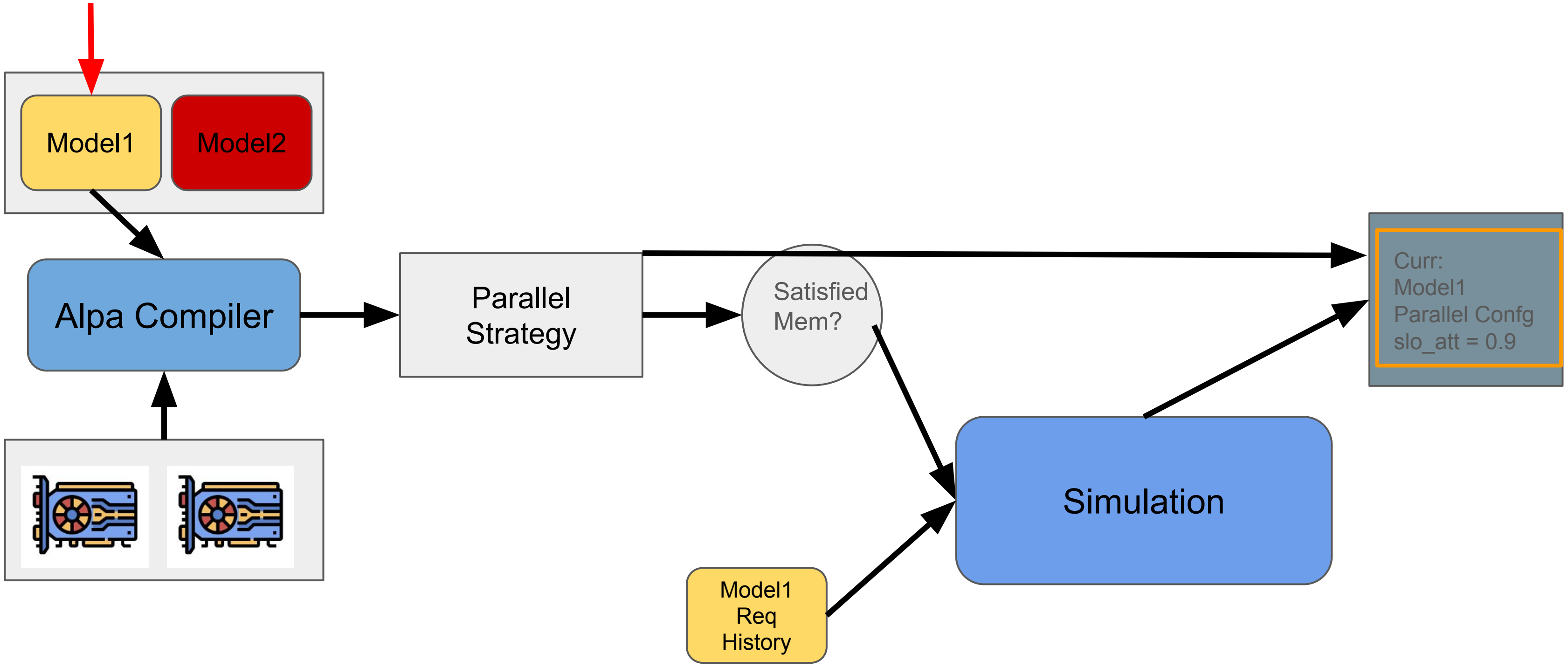
SLO: Service Level Objective (normally on latency)

Different requests may have different SLO, SLO attainment is the percentage of requests you serve within SLO.

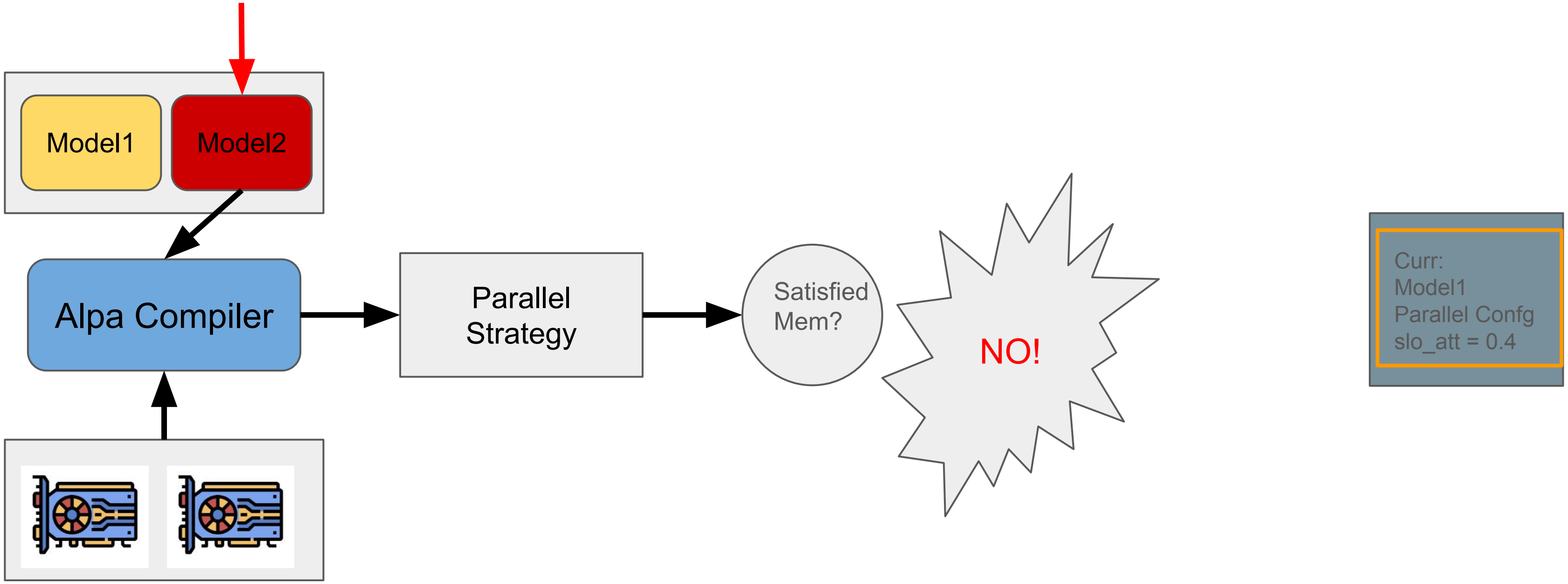
How AlpaServe solves the problem



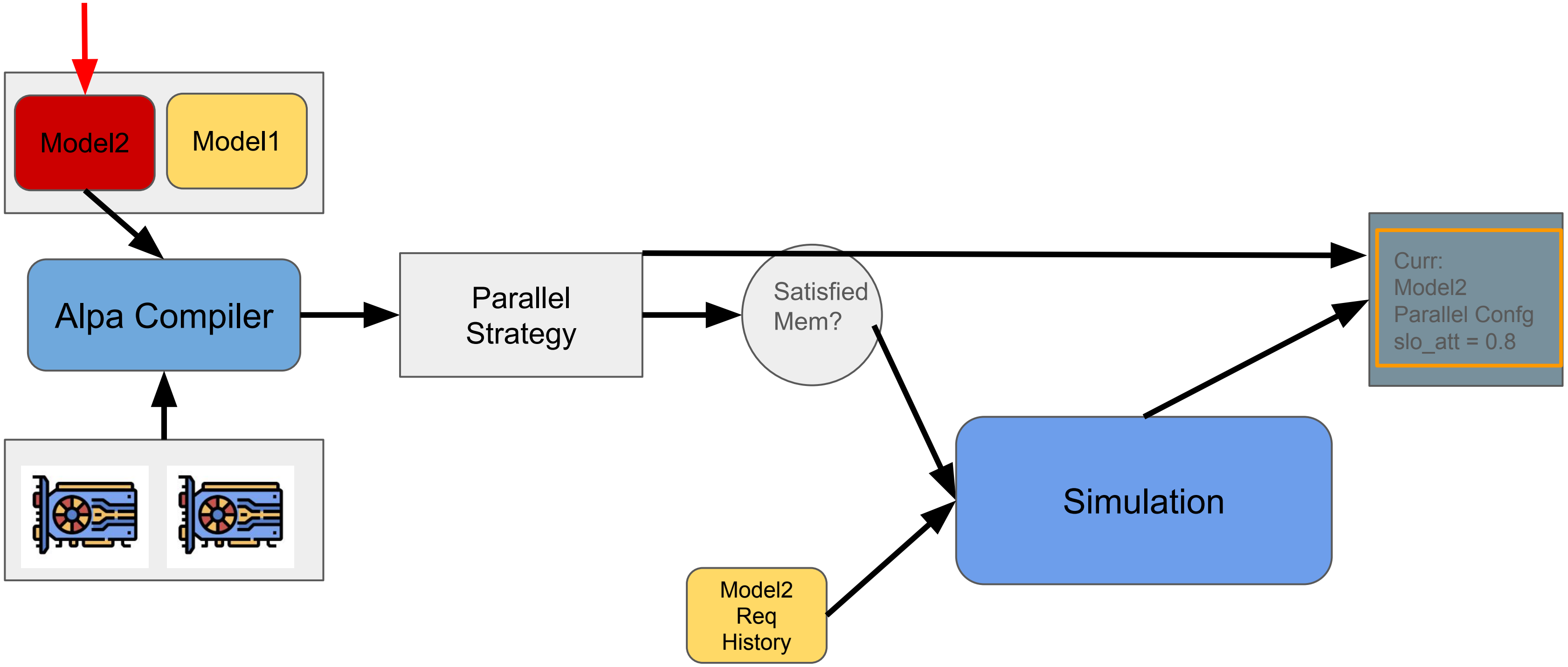
How the black box works?



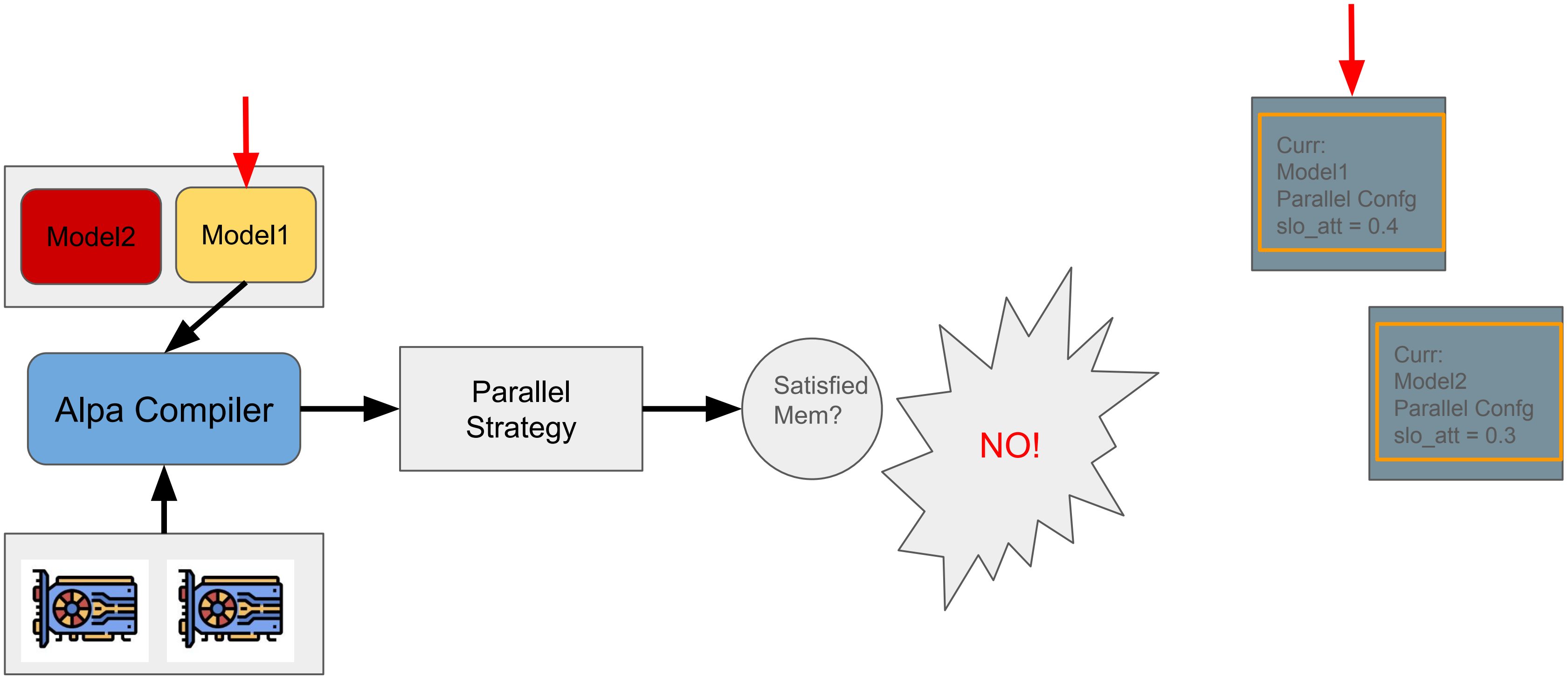
How the black box works?



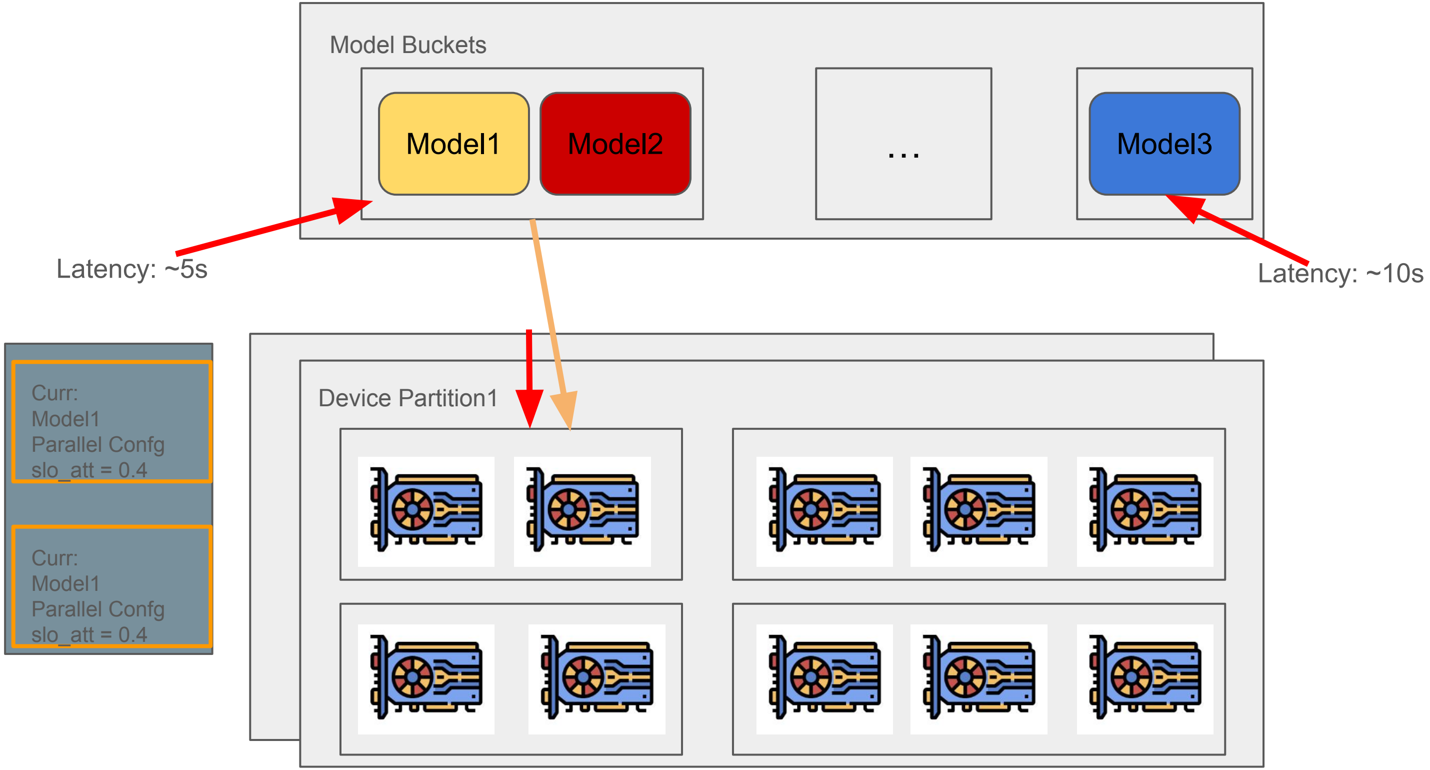
How the black box works?



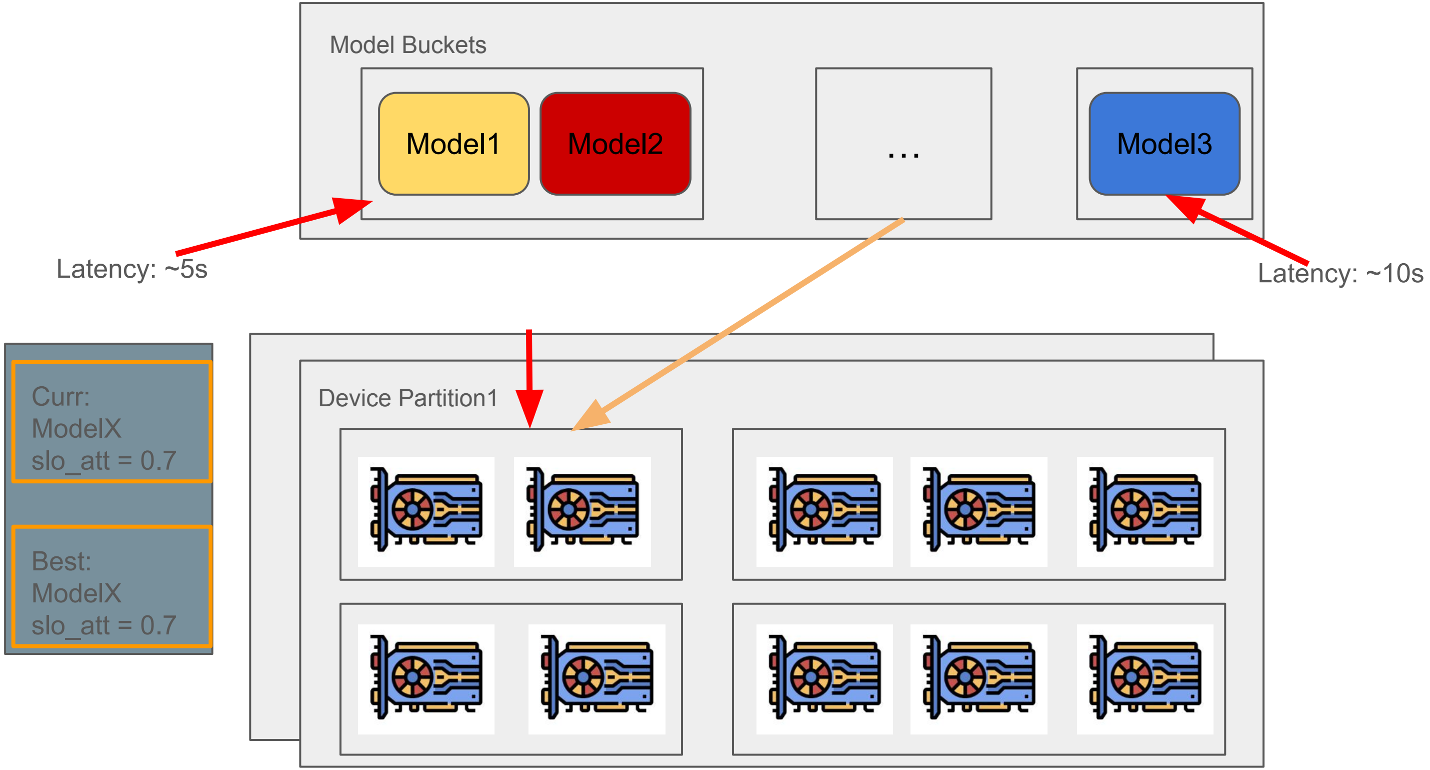
How the black box works?



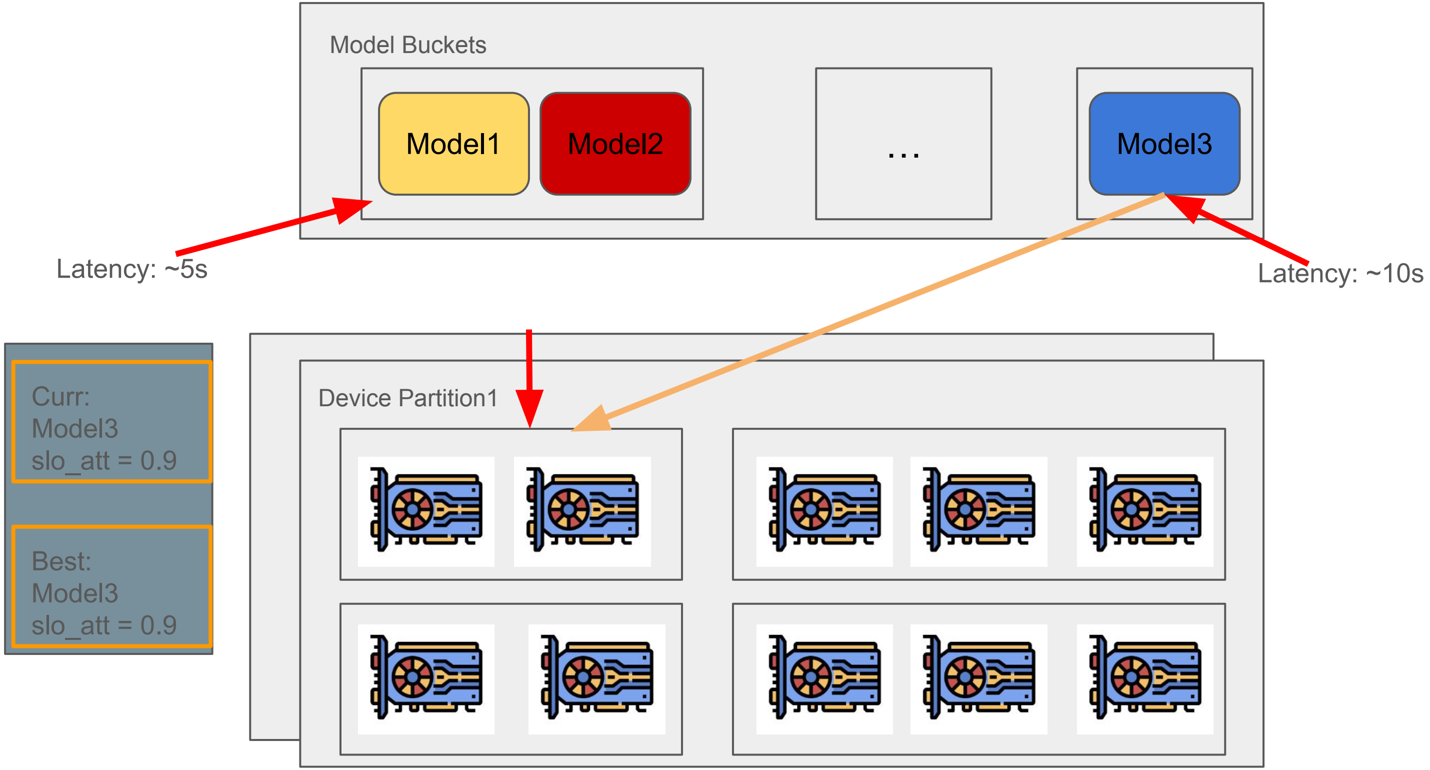
How AlpaServe solves the problem



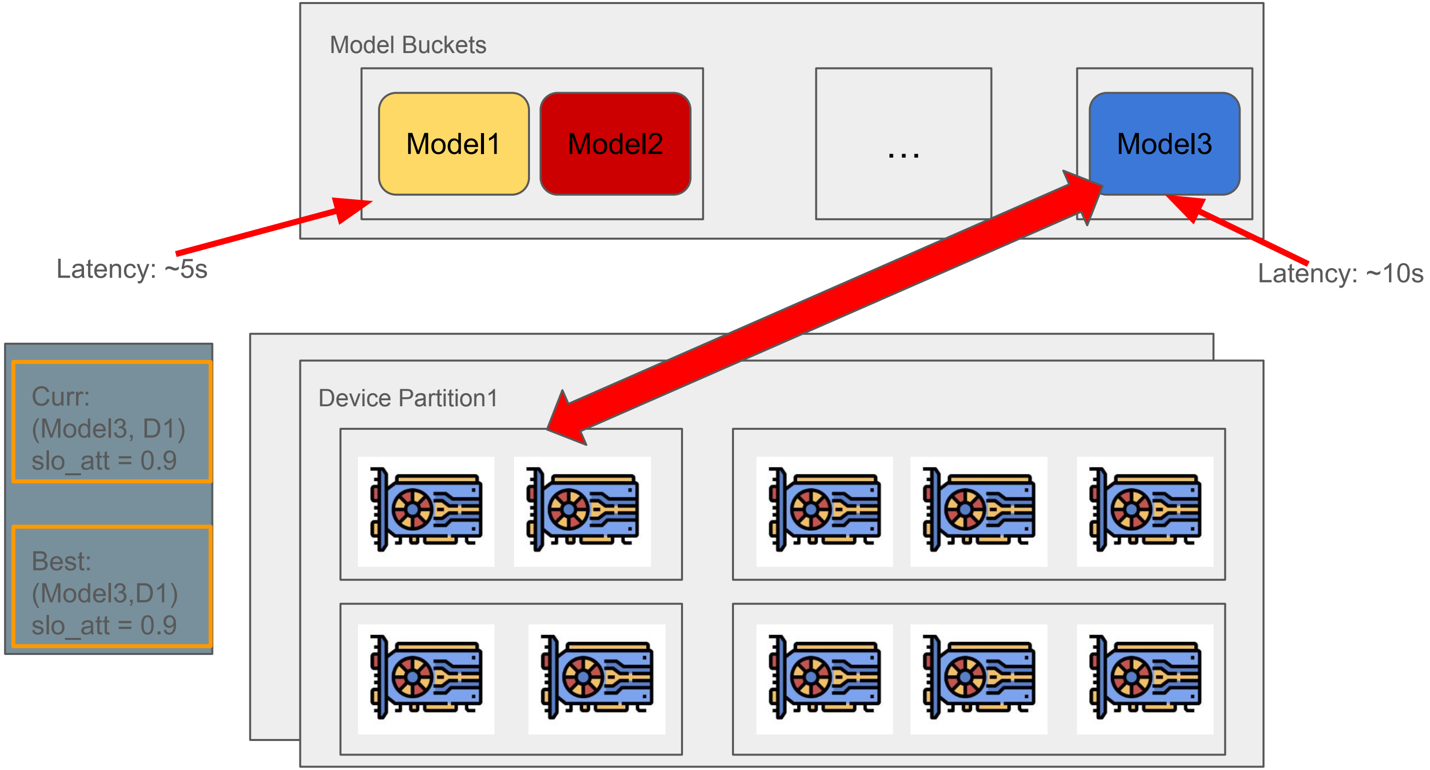
How AlpaServe solves the problem



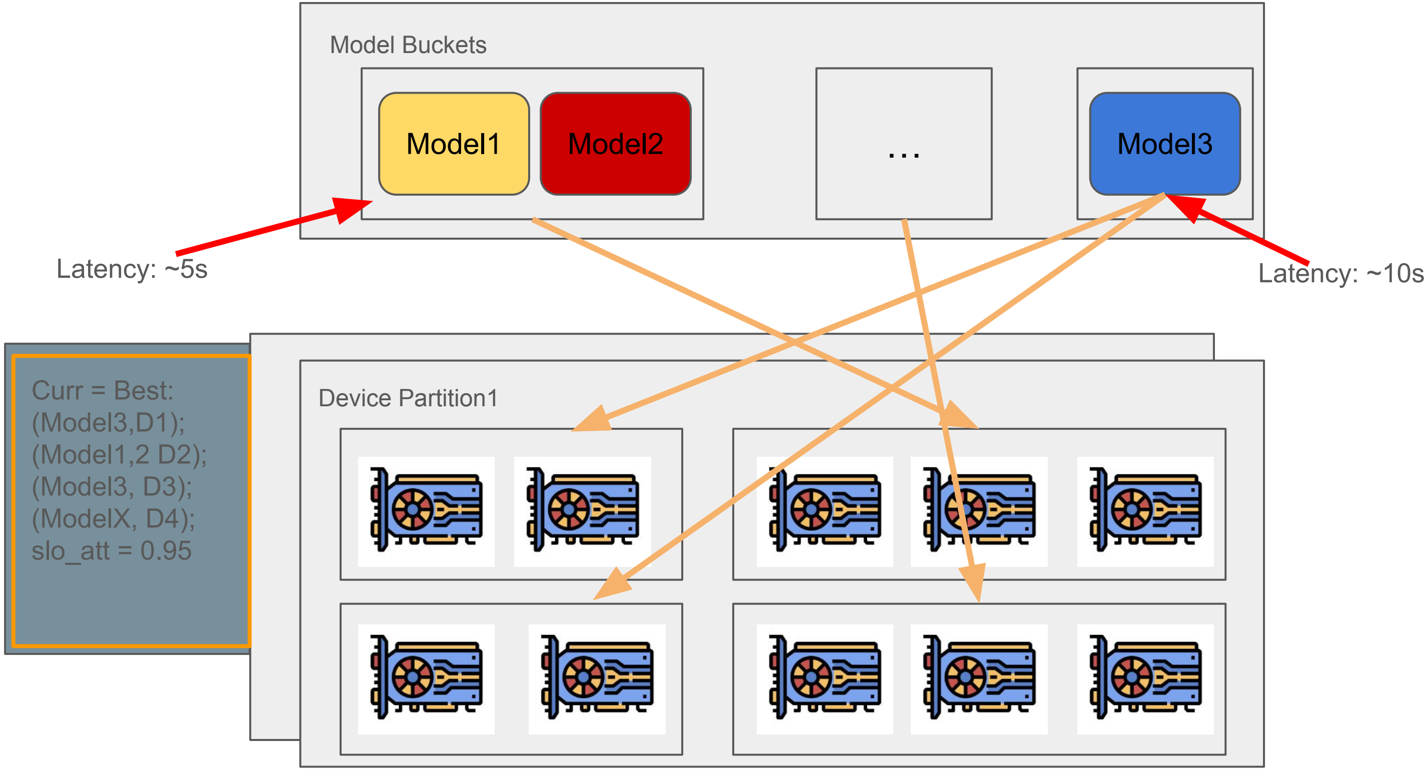
How AlpaServe solves the problem



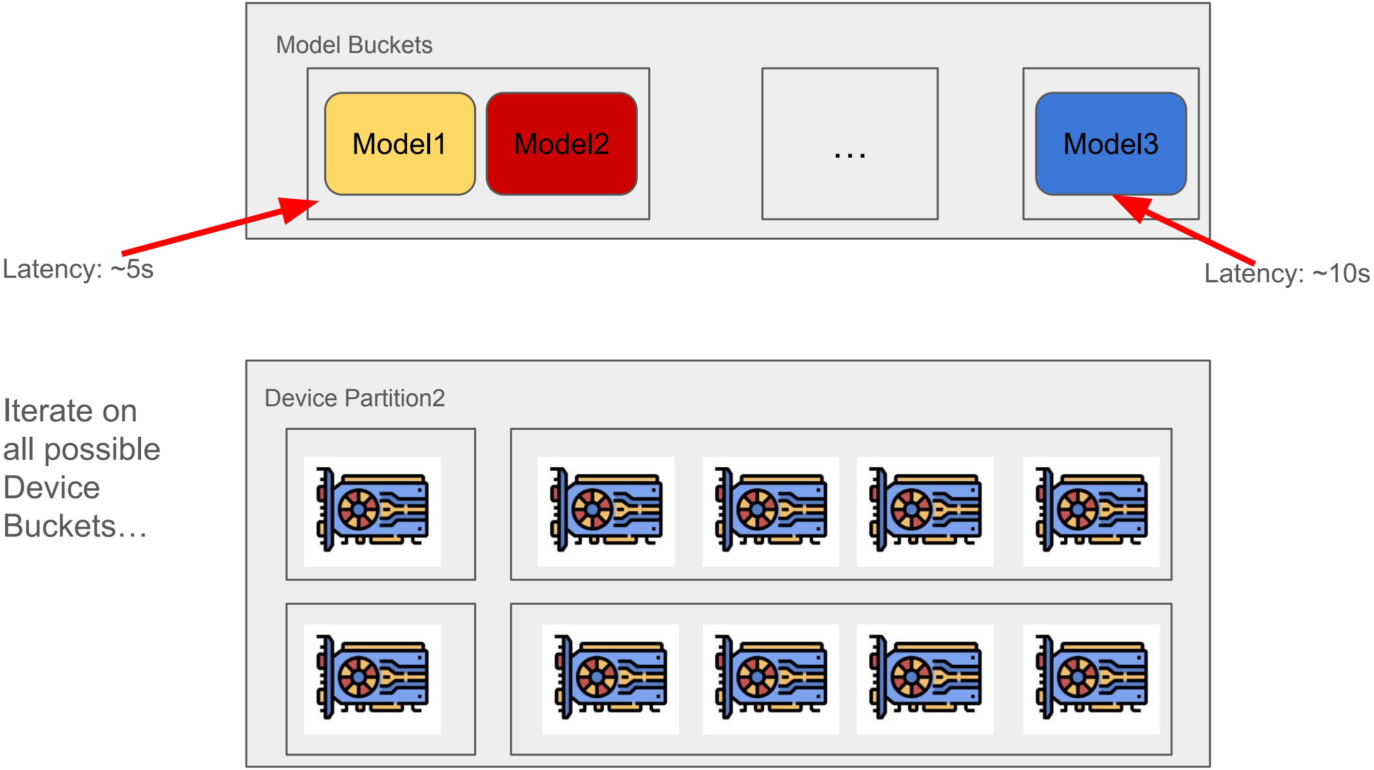
How AlpaServe solves the problem



How AlpaServe solves the problem



How AlpaServe solves the problem



Iterate on all possible Device Buckets…

Evaluation Setup

Cluster: 8 nodes and 64 GPUs. Each node is an AWS EC2 p3.16xlarge instance with 8 NVIDIA Tesla V100 (16GB) GPUs

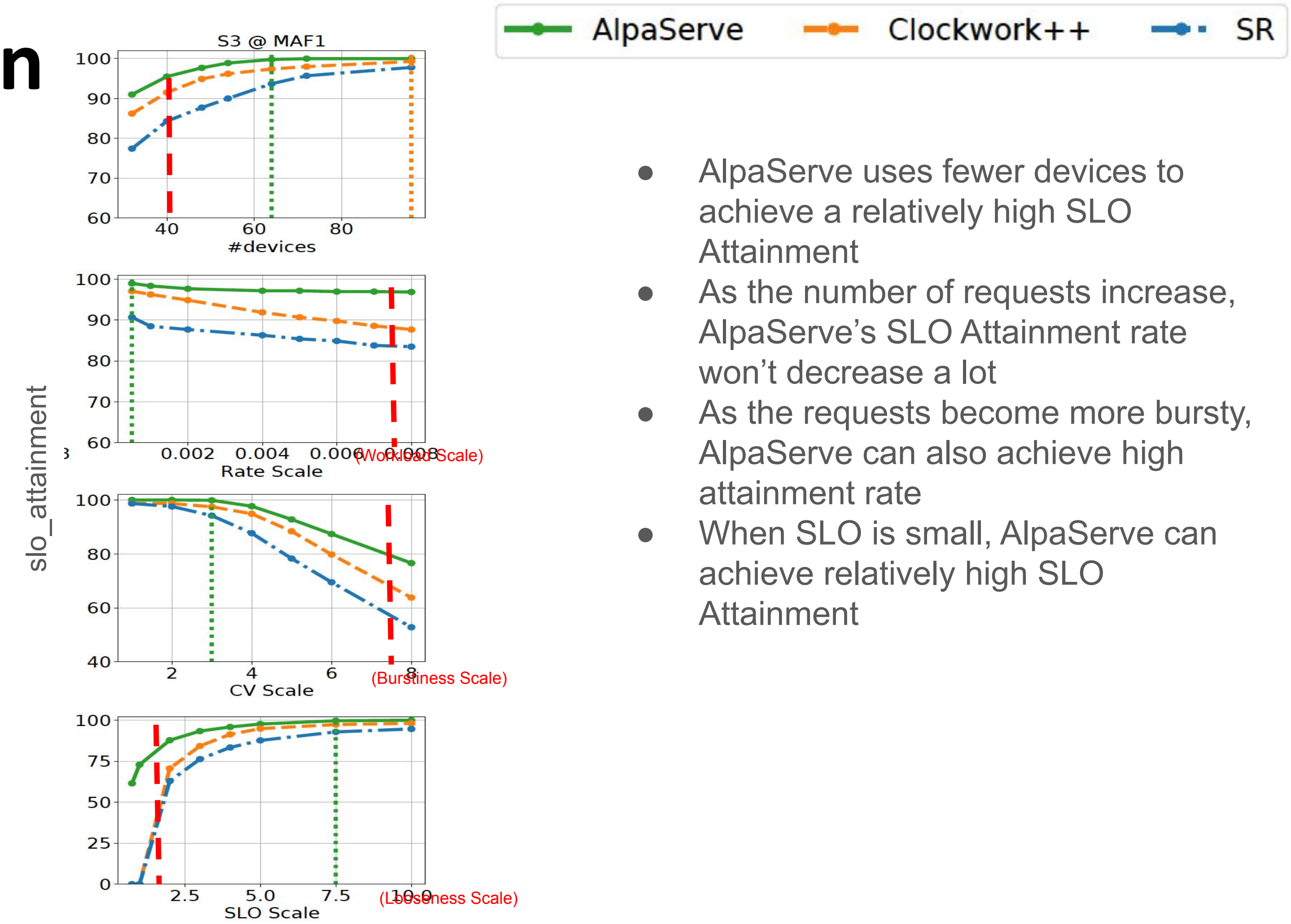
Model: Many BERT and Transformer MoE models of 6 different sizes/latency

Req History: 2 Microsoft Azure function traces(one steady and one brusty)

Evaluation Setup

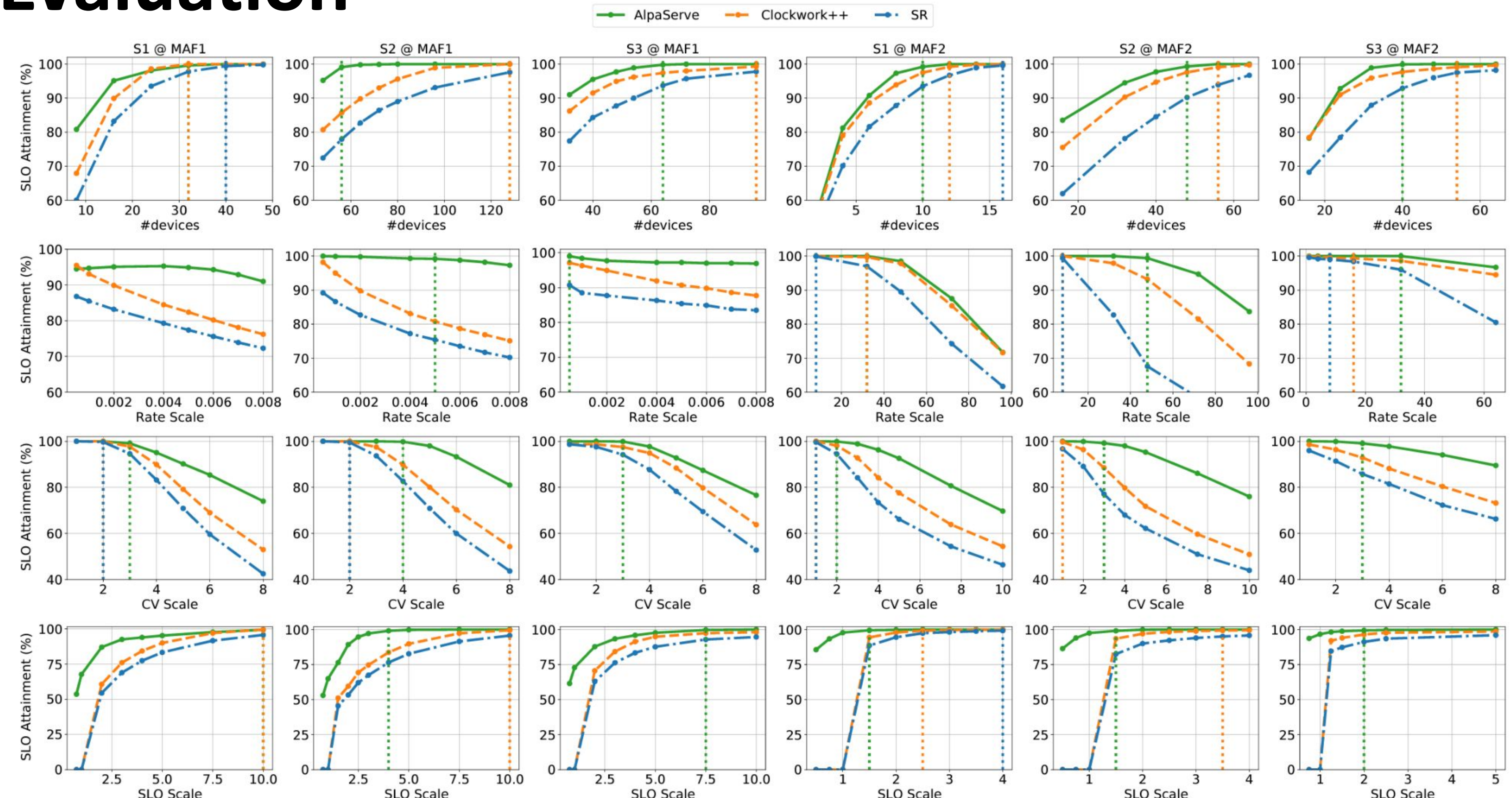
Name	Size	Latency (ms)	S1	S2	S3	S4
BERT-1.3B	2.4 GB	151	32	0	10	0
BERT-2.7B	5.4 GB	238	0	0	10	0
BERT-6.7B	13.4 GB	395	0	32	10	0
BERT-104B	208 GB	4600	0	0	0	4
MoE-1.3B	2.6 GB	150	0	0	10	0
MoE-2.4B	4.8 GB	171	0	0	10	0
MoE-5.3B	10.6 GB	234	0	0	10	0

Evaluation



- AlpaServe uses fewer devices to achieve a relatively high SLO Attainment
- As the number of requests increase, AlpaServe's SLO Attainment rate won't decrease a lot
- As the requests become more bursty, AlpaServe can also achieve high attainment rate
- When SLO is small, AlpaServe can achieve relatively high SLO Attainment

Evaluation



AlpaServe
Clockwork++
SR
S1 @ MAF1
S2 @ MAF1
S3 @ MAF1
S1 @ MAF2
S2 @ MAF2
S3 @ MAF2
SLO Attainment (%)
#devices
Rate Scale
CV Scale
SLO Scale

Evaluation: Very Large Model (Set 4)

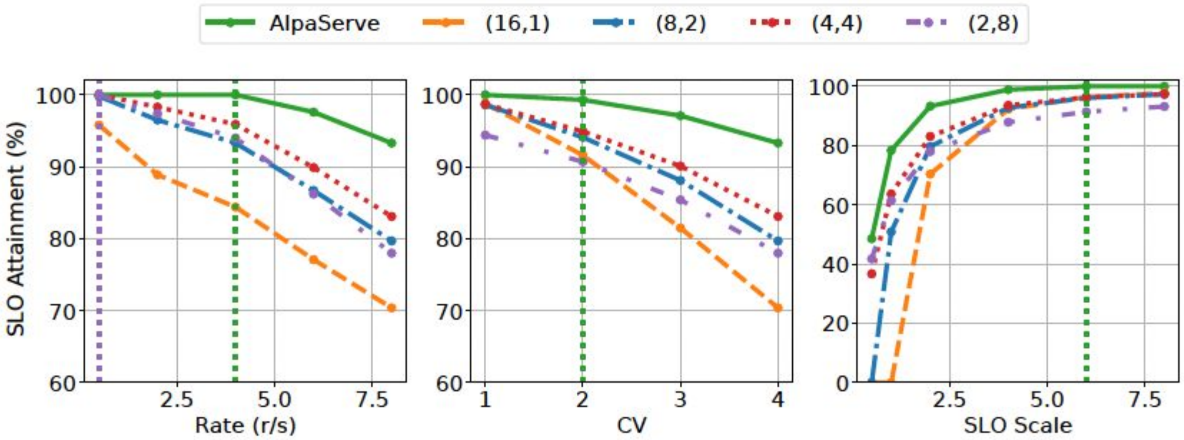


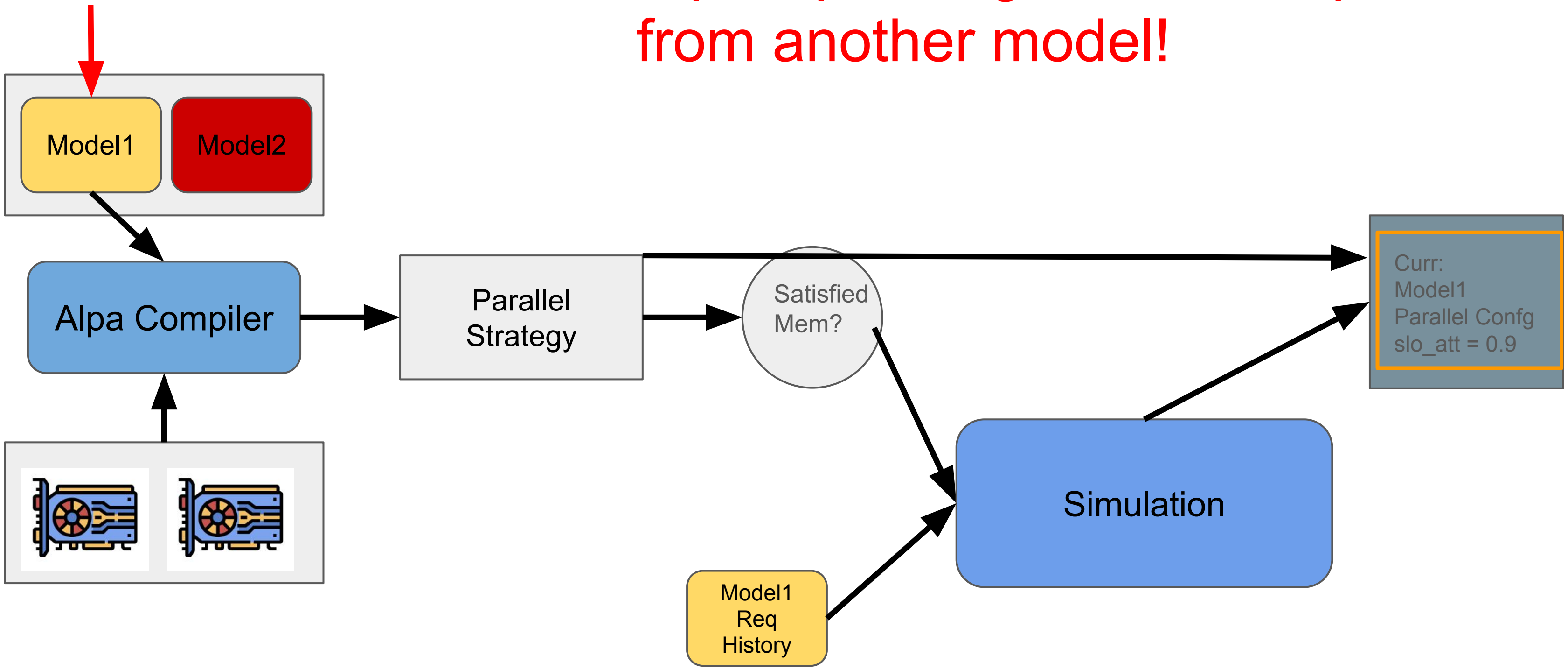
Figure 13: SLO attainment as we vary the rate, CV, and SLO scale. (8,2) means 8-way inter-op parallelism and in each pipeline stage using 2-way intra-op parallelism.

Discussion

1. When using Alpa Compiler to generate best parallel strategy, does it consider the interaction/impact between other models?

How the black box works?

Alpa's profiling omit the impact from another model!



Discussion

1. When using Alpa Compiler to generate best parallel strategy, it will do profiling. Does it consider the interaction between other models?

2. Planning time is not included in their evaluation. Given such a high complexity, it cannot be directly ignored.

Discussion

1. When using Alpa Compiler to generate best parallel strategy, it will do profiling. Does it consider the interaction between other models?

2. Planning time is not included in their evaluation. Given such a high complexity, it cannot be directly ignored.

3. When model-level parallelism is most beneficial?

Discussion

A1: Model parallelism benefits model serving through statistical multiplexing at the following scenario:

- Limited per-device memory
- Below system peak throughput overall request rate
- High request burstiness
- Tight SLO

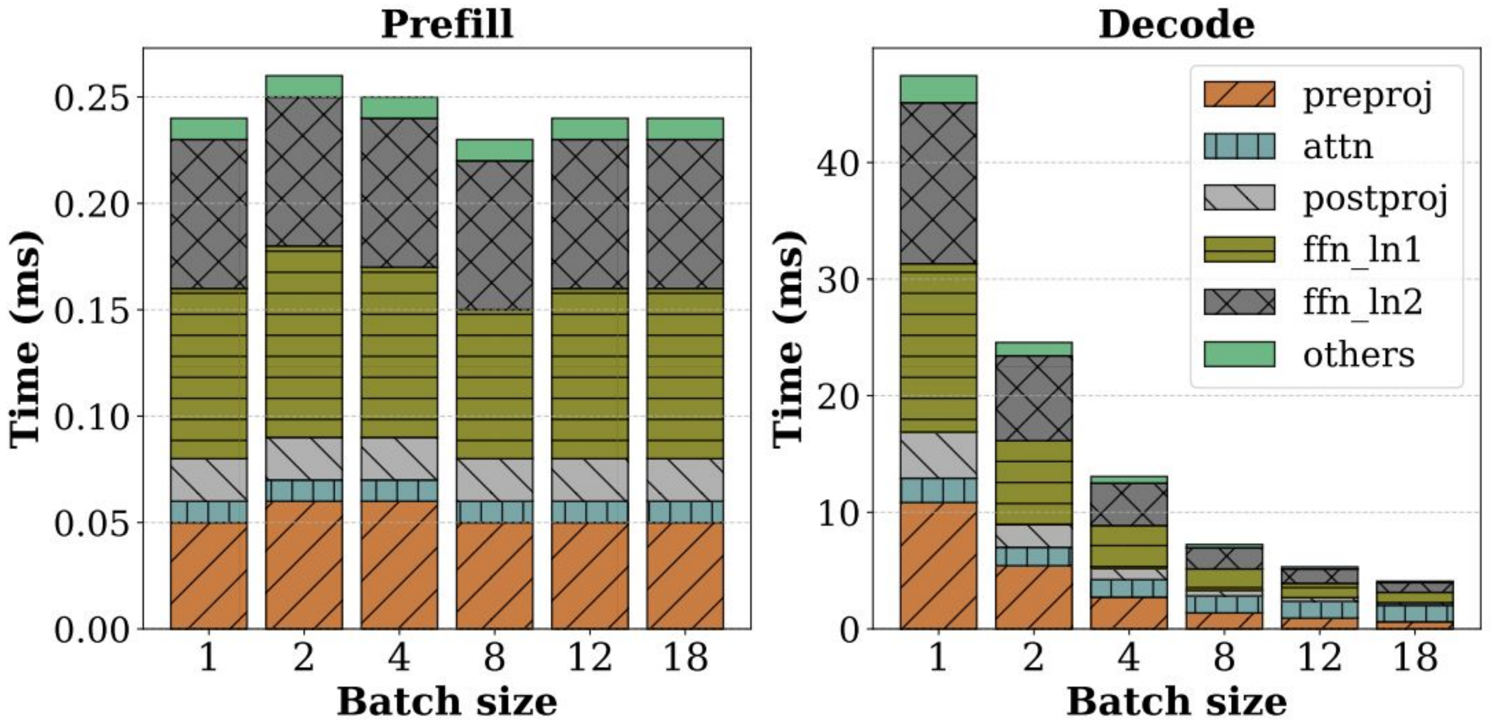
SARATHI: Efficient LLM Inference by Piggybacking Decodes with Chunked Prefills

Amey Agrwal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S. Gulavani, Ramachandran Ramjee
(Microsoft Research India, Georgia Institute of Technology)

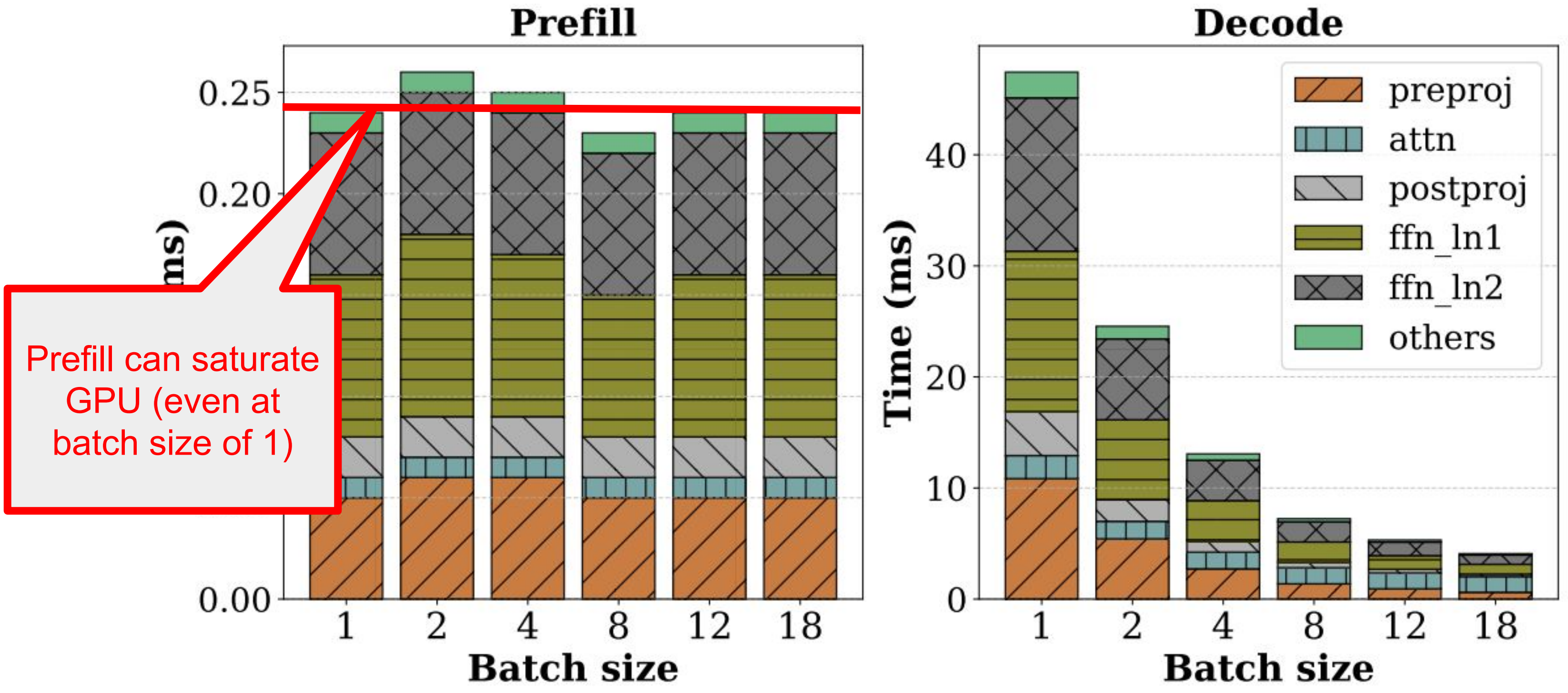
Recall: Prefill and Decode

- Transformer Architecture: 2 phases
  - Prefill
  - Decode
- Prefill
  - Digest Prompt → Process **all** tokens in input sequence in parallel
  - High GPU utilization for small batch sizes
- Decode
  - Predict Next Token → Processes only a **single** token in each autoregressive pass
  - Low GPU utilization at small batch sizes
- A single request goes through 1 prefill pass and multiple decode pass
  - Prefill pass + decode passes = inefficient

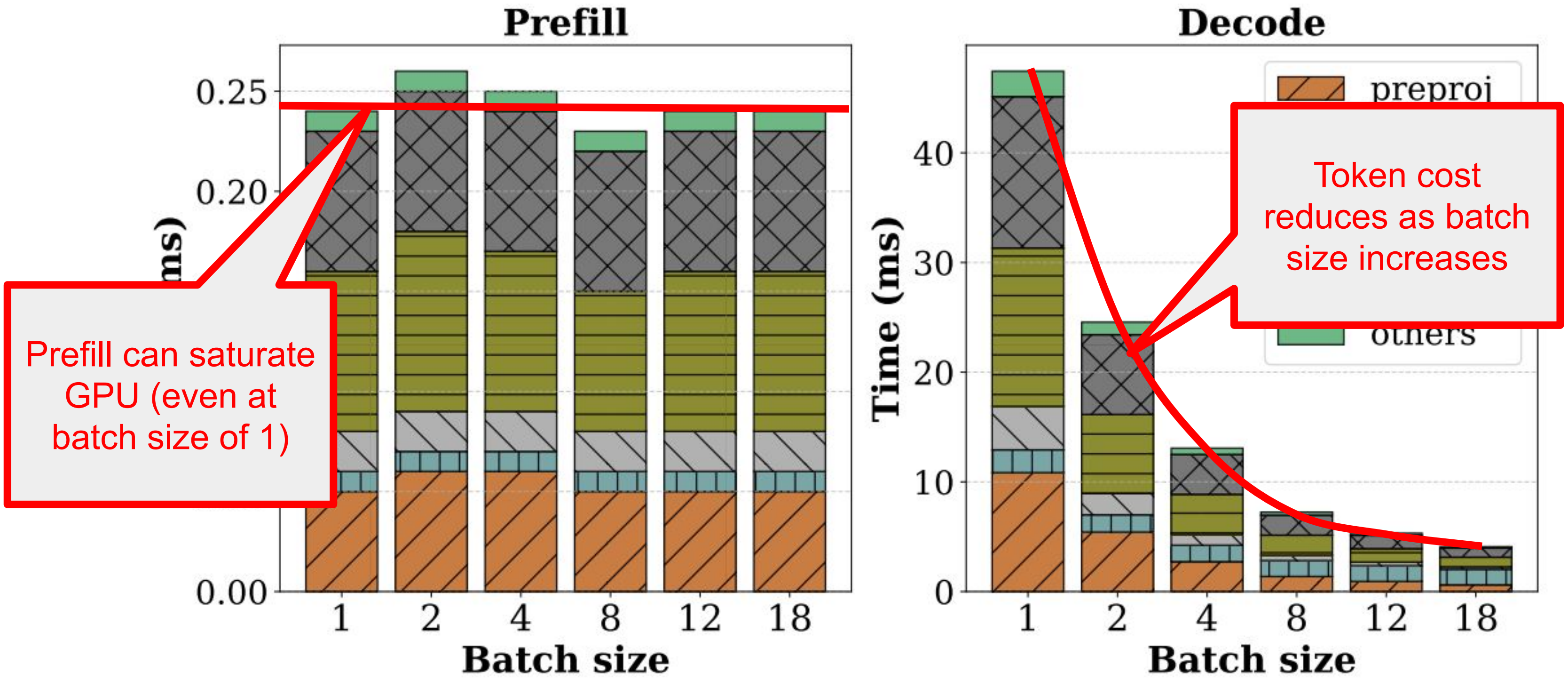
Prefill and Decode: Per token time



Prefill and Decode: Per token time



Prefill and Decode: Per token time

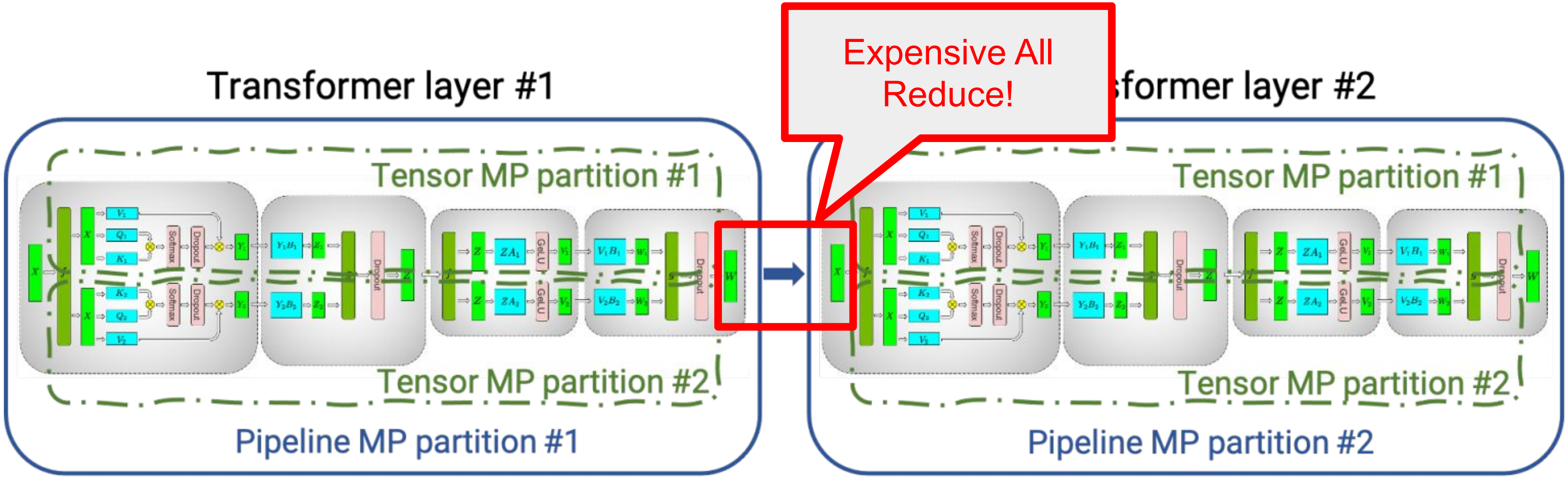


Parallelism: Improving LLM Inefficiencies

- Tensor Parallelism
- Pipeline Parallelism

Tensor Parallelism

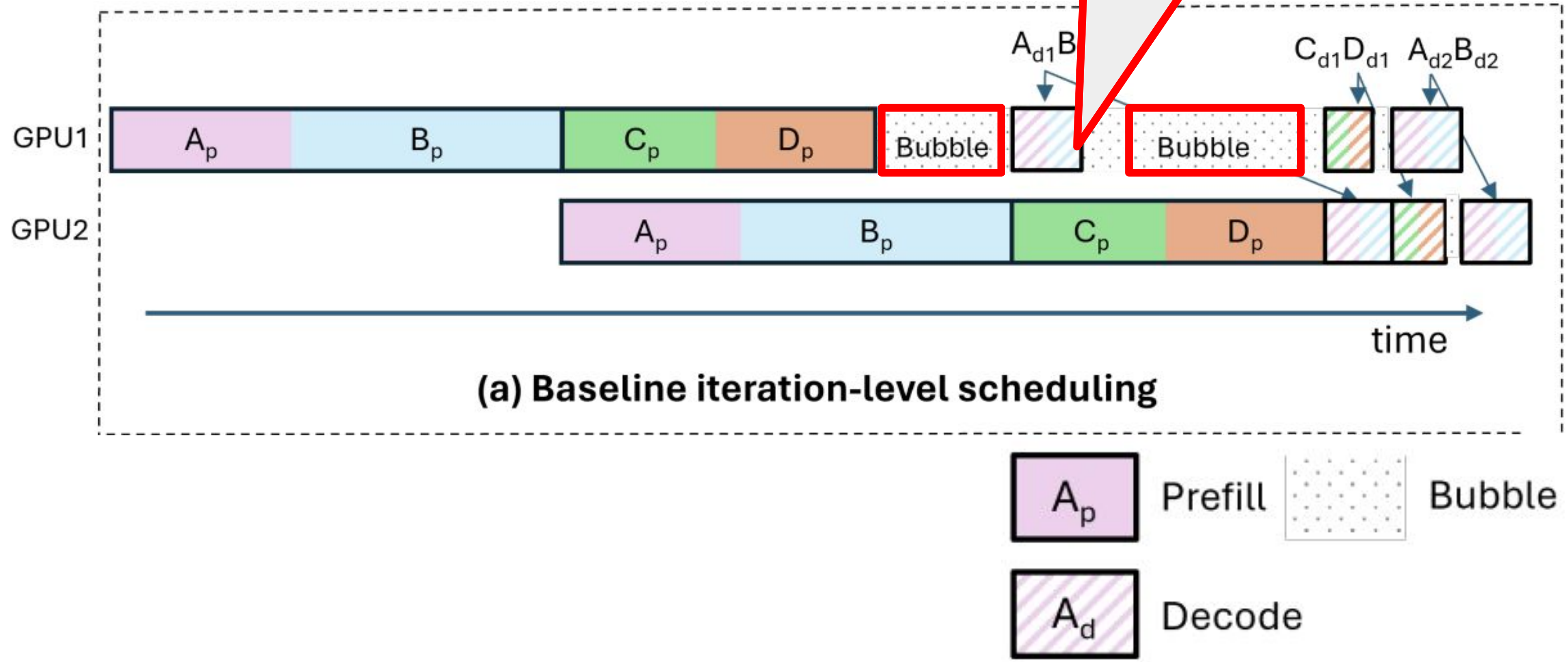
Shard each individual layer across multiple GPU



Pipeline Parallelism

Shard model layer wise → each GPU responsible

Bubbles lead to idle time!
Bubbles occur from different compute times



(a) Baseline iteration-level scheduling

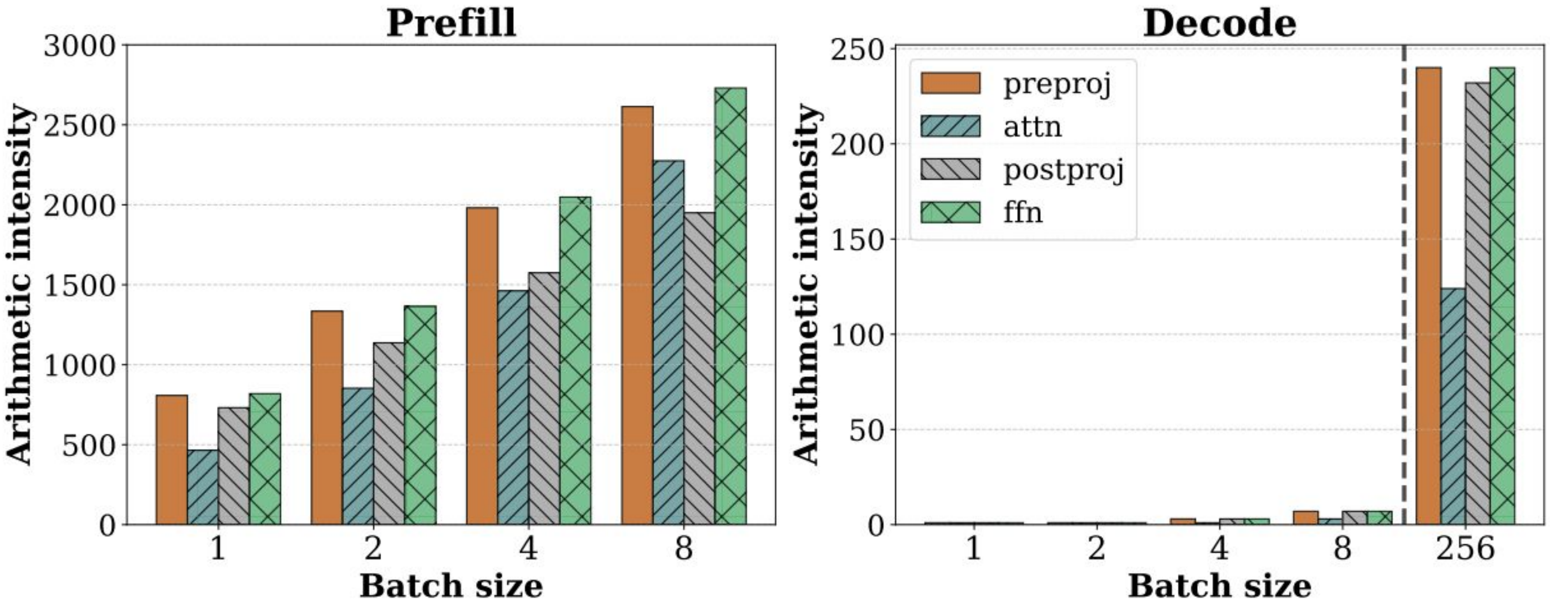
Parallelism Insights

- Tensor Parallelism
    - Communication needed for all-reduce operation
    - Only viable with high-bandwidth connectivity
- Pipeline Parallelism
    - No all-reduce operation needed
    - Viable even if high-bandwidth connectivity not available
    - Pipeline Bubbles from **varying compute times** → Inefficiencies

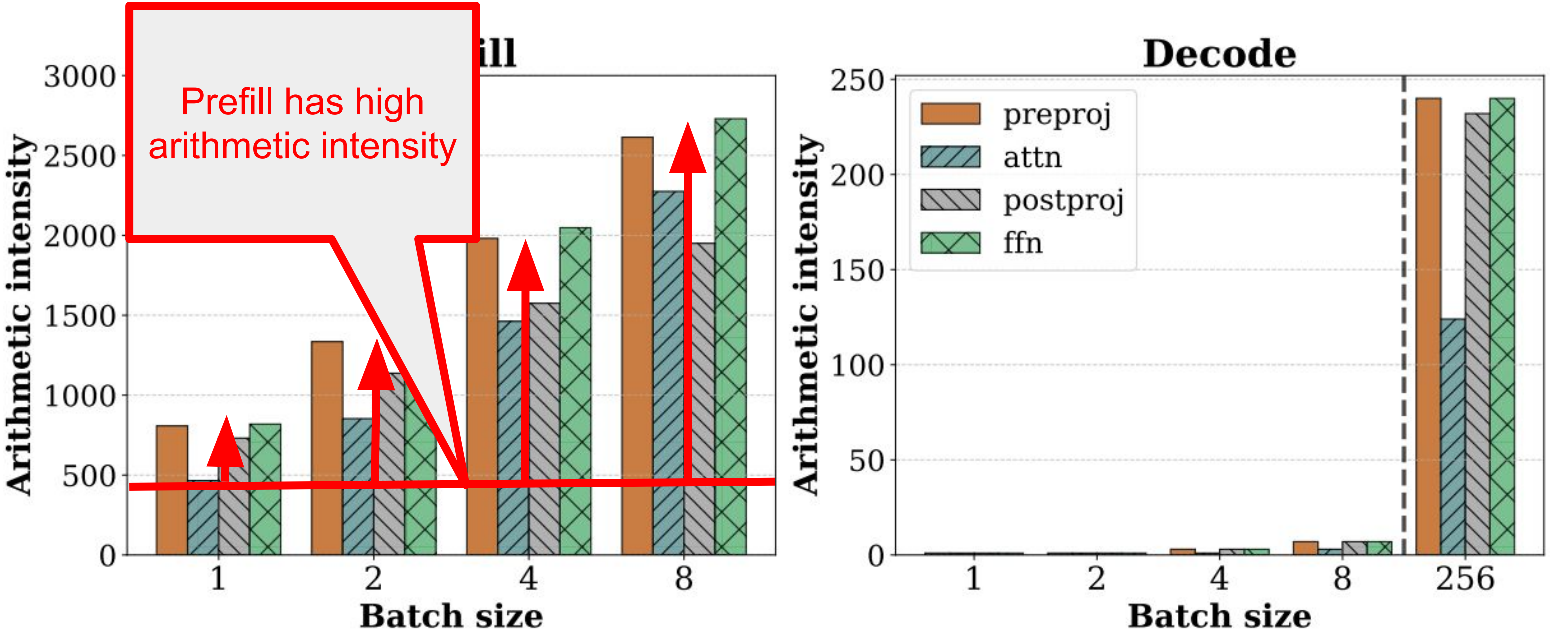
LLM Inefficiencies

1. Pipeline Parallelism leads to big bubbles

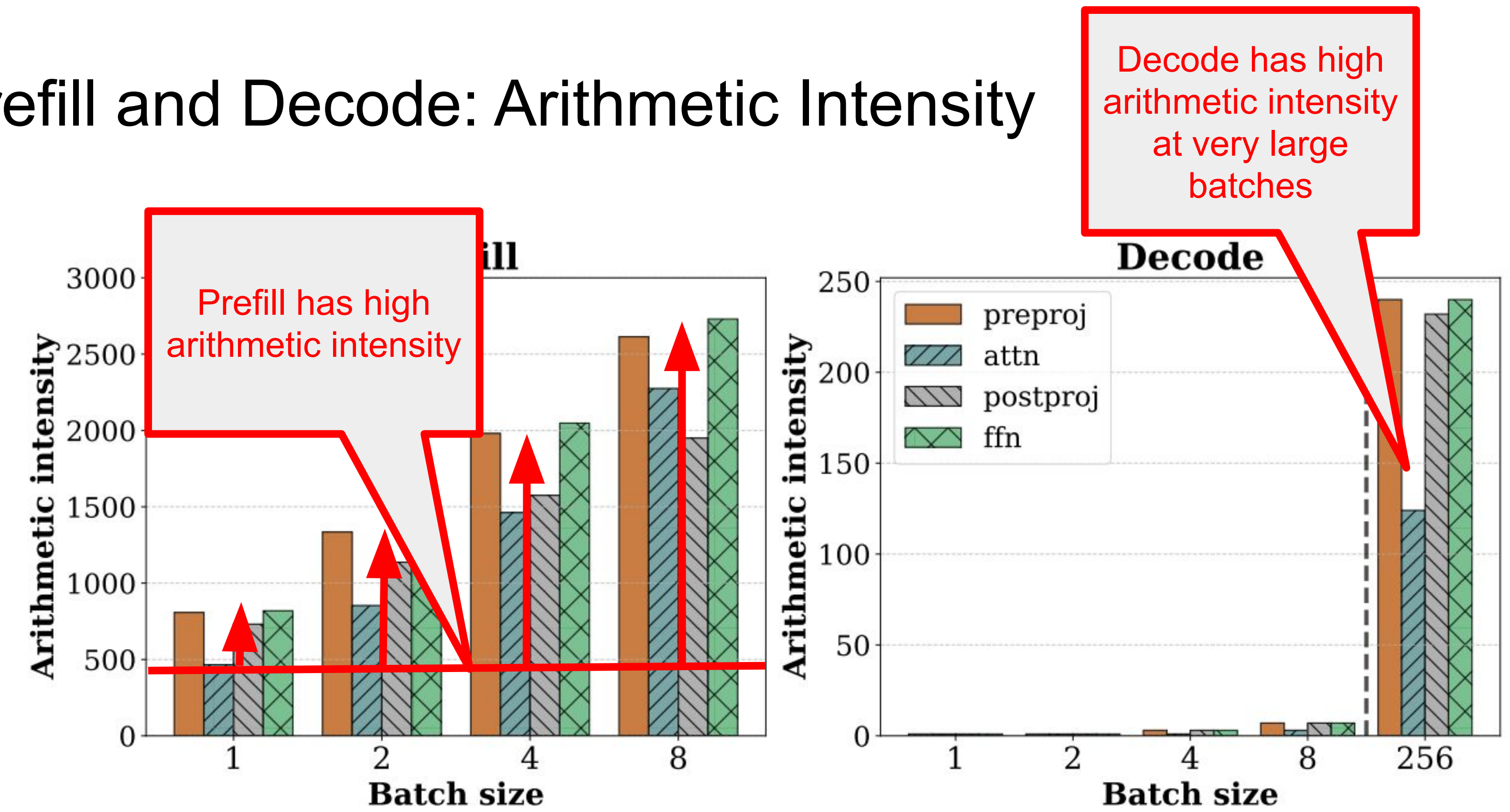
Prefill and Decode: Arithmetic Intensity



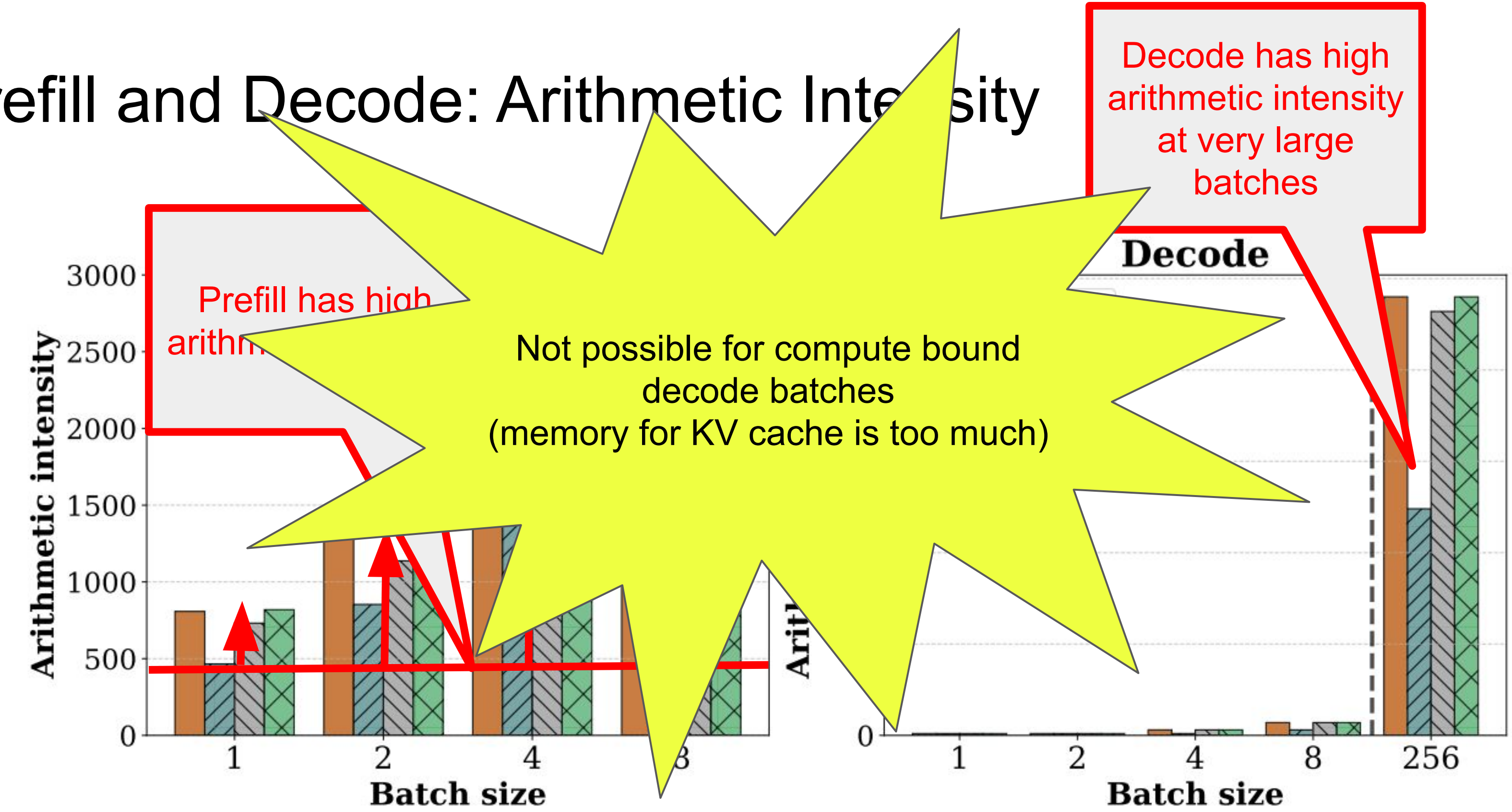
Prefill and Decode: Arithmetic Intensity



Prefill and Decode: Arithmetic Intensity



Prefill and Decode: Arithmetic Inte sity



Notes

- Increasing batch size of prefill does not affect cost (per-token)
- Increasing batch size of decode reduces cost (per-token)
- Decode can be compute-bound at very high batch sizes
    - Very high batch sizes not practical today
    - Memory needed for KV-cache for each individual request → a lot

LLM Inefficiencies

1. Pipeline Parallelism leads to big bubbles
2. Decoding is memory-bound

LLM Inefficiencies

1. Pipeline Parallelism leads to big bubbles
2. Decoding is memory-bound

What if we can create uniformly compute-intensive batches

LLM Inefficiencies

1. Pipeline Parallelism will have smaller bubbles
2. Decoding is memory-bound

What if we can create uniformly compute-intensive batches

LLM Inefficiencies

1. Pipeline Parallelism will have smaller bubbles
2. Decoding is compute-bound

What if we can create uniformly compute-intensive batches

LLM Inefficiencies (Gone)

1. Pipeline Parallelism will have smaller bubbles
2. Decoding is compute-bound

What if we can create uniformly compute-intensive batches

LLM Inefficiencies (Gone)

1. Pipeline Parallelism will have smaller bubbles
2. Decoding is compute-bound

What if we can create uniformly compute-intensive batches

SARATHI

Proposal for 2 new concepts:

- **Chunked-Prefills:** Split a prefill into multiple chunks
- **Decode-Maximal Batching:** Construct a batch by using a single prefill chunk and “piggybacking” that chunk with multiple decode tokens

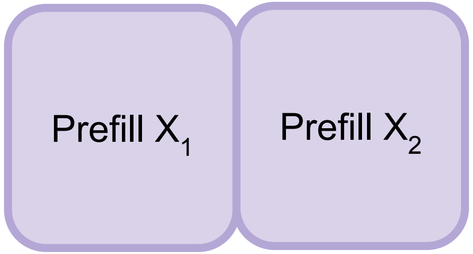
Chunked-Prefills + Decode-Maximal Batching → Hybrid batches of prefill and decode

SARATHI

Given Prefill X:

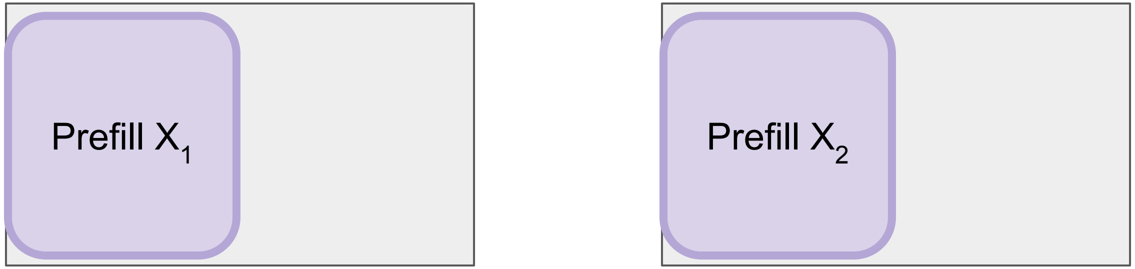
SARATHI

We can split X



SARATHI

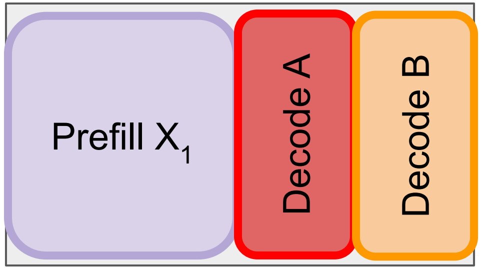
With a predetermined batch size,

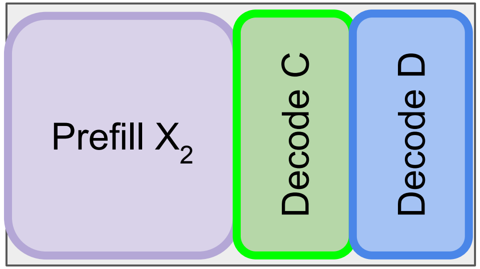


SARATHI

We can fill the remaining space in batch with “piggybacked” decodes

Uniform Hybrid Batches:



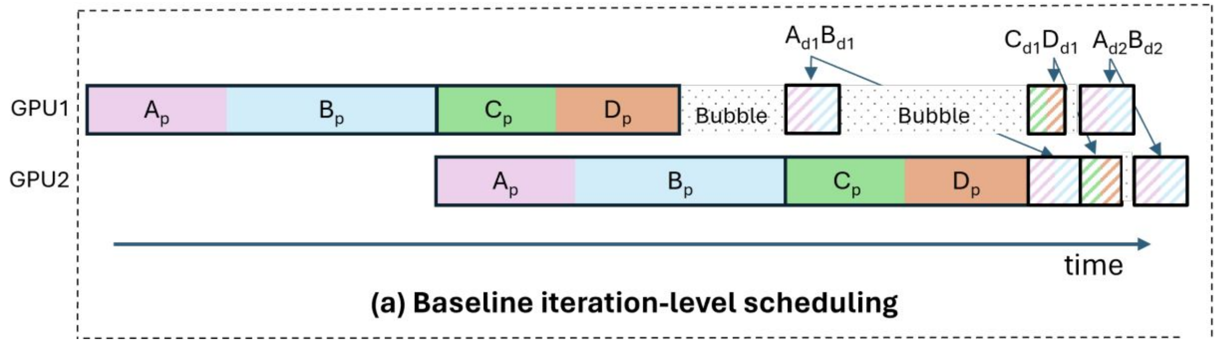


LLM Inefficiencies

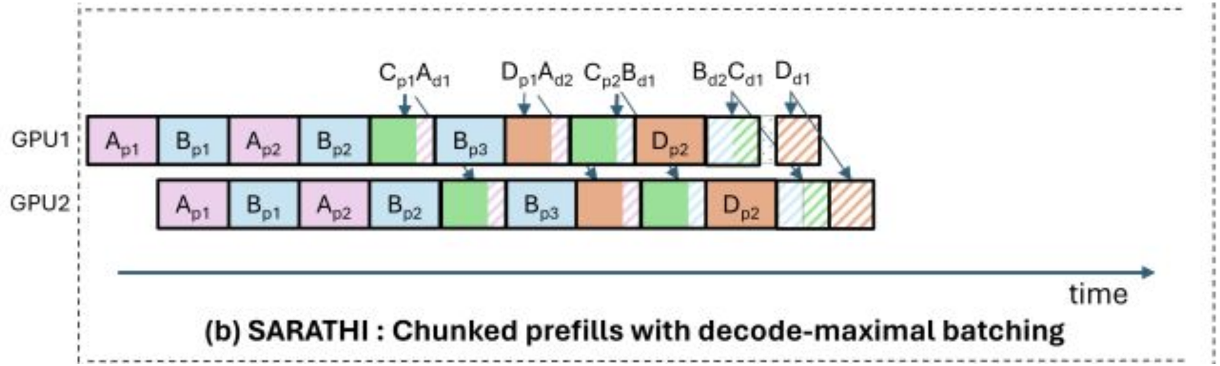
1. Pipeline Parallelism leads to large bubbles
    a. Uniform Hybrid batch size
2. Decoding is memory-bound

SARATHI: Results

Baseline Iteration (like ORCA):



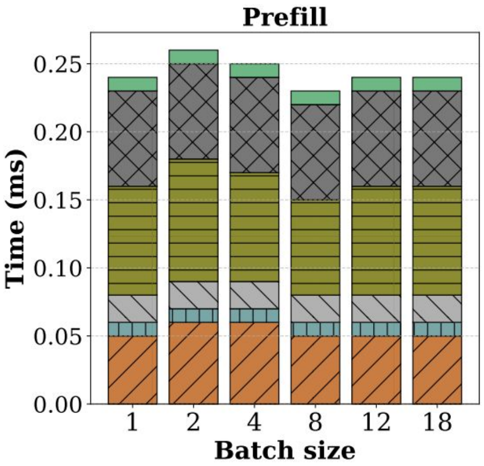
SARATHI:



SARATHI: Chunked Prefills

Hinged on 2 key insights:

- Decreasing # of prefill tokens → gives around the same throughput
- Larger hidden dimension → chunk size of prefill needed to saturate GPU lowers
  - More computations per data element (in input)



Conclusion: Compute-saturating batch is possible

- Most practical scenarios: prefill is large
- Can easily split prefill into multiple chunks

SARATHI: Decode-Maximal Batching

Combine linear operation computations for prefill chunk and decodes into single operation

- Result: Decodes are more efficient because of GPU-saturating prefill computation

What is maximum possible number of decodes?

$$B = \left\lfloor \left(\frac{M_G - M_S}{L * m_{kv}} \right) \right\rfloor$$

What is maximum possible number of decodes?

Available GPU memory

$$B = \left\lfloor \left(\frac{\boxed{M_G} - M_S}{L * m_{kv}} \right) \right\rfloor$$

What is maximum possible number of decodes?

Available GPU memory

Model Parameter Memory Requirement

$$B = \lfloor \left(\frac{\boxed{M_G} - \boxed{M_S}}{L * m_{kv}} \right) \rfloor$$

What is maximum possible number of decodes?

Memory left over

$$B = \left\lfloor \left(\frac{\boxed{M_G - M_S}}{L * m_{kv}} \right) \right\rfloor$$

What is maximum possible number of decodes?

$$B = \left\lfloor \left(\frac{\boxed{M_G - M_S}}{\boxed{L} * m_{kv}} \right) \right\rfloor$$

Memory left over

Maximum seq len supported by model

What is maximum possible number of decodes?

Memory left over

$$B = \left\lfloor \left(\frac{\boxed{M_G - M_S}}{\boxed{L} * \boxed{m_{kv}}} \right) \right\rfloor$$

Maximum seq len supported by model

Memory required per pair of K and V for a token

What is maximum possible number of decodes?

$$B = \left\lfloor \left(\frac{\overbrace{M_G - M_S}^{\text{Memory left over}}}{\underbrace{L * m_{kv}}_{\text{Memory needed for KV cache per request}}} \right) \right\rfloor$$

What is maximum possible number of decodes?

$$\underbrace{\boxed{B}}_{\text{\# of Prefill + \# of Decodes}} = \left\lfloor \left(\frac{\overbrace{\boxed{M_G - M_S}}^{\text{Memory left over}}}{\underbrace{\boxed{L * m_{kv}}}_{\text{Memory needed for KV cache per decode}}} \right) \right\rfloor$$

What is maximum possible number of decodes?

of Prefill + # of Decodes

Memory left over

$$\boxed{B} = \left\lfloor \left(\frac{\boxed{M_G - M_S}}{\boxed{L * m_{kv}}} \right) \right\rfloor$$

Memory needed for KV cache per decode

Since only 1 prefill chunk followed by multiple decodes, # of decodes = B - 1

SARATHI: Decode-Maximal Batching

Prefill and Decode phases follow same computation path

- Linear operations use same weight tensors in both prefill and decode phases
    - Only need to load model weights once onto GPU!
    - Before: had to separately load weights for decoding
    - After: Decoding becomes compute-bound instead of memory bound

LLM Inefficiencies (Gone)

1. Pipeline Parallelism leads to large bubbles
    a. Uniform Hybrid batch size
2. Decoding is memory bound
    a. Loading weights only once
    b. Efficient combination of linear ops
    c. Larger batch of decodes

SARATHI: Tradeoff to prefill chunk size

As prefill chunk size decrease:

- More decode tokens can be “piggybacked” (Good)
- Arithmetic intensity of chunked-prefills computation decrease (Bad)

How to determine optimal prefill chunk size?

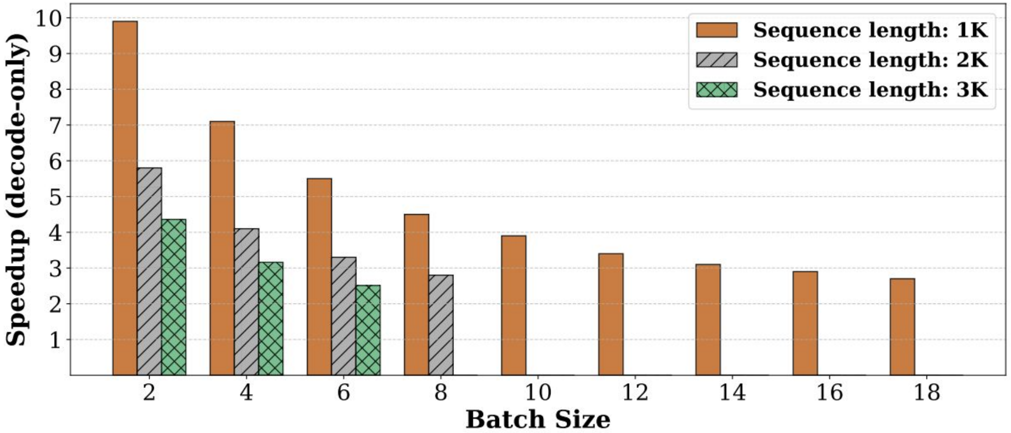
- What is expected of workload?
  - More prefills or decodes?
- Varies depending on model and hardware

Answer: **Profiling**

Evaluation

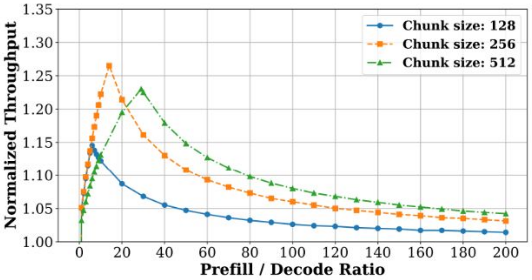
Batching Scheme	Operation(s)		Total Time	Per-token Time	
	Linear	Attn		Prefill	Decode
Prefill-only	224.8	10	234.8	0.229	-
Decode-only	44.28	5.68	49.96	-	**12.49**
Decode-maximal	223.2	15.2	238.4	0.229	**1.2**

Evaluation

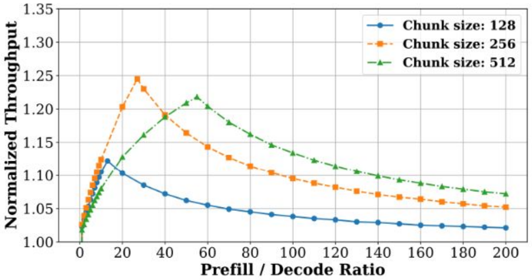


Model (GPU)	Sequence Length	Batch Size	P:D Ratio	Decode Speedup	Throughput Gain
LLaMA-13B (A6000)	1K	6	50:1	5.45×	1.33×
	2K	6	50:1	3.26×	1.26×
	3K	6	50:1	2.51×	1.22×
LLaMA-33B (A100)	1K	10	28:1	3.83×	1.25×
	2K	5	63:1	4.25×	1.22×
	3K	3	127:1	3.51×	1.14×

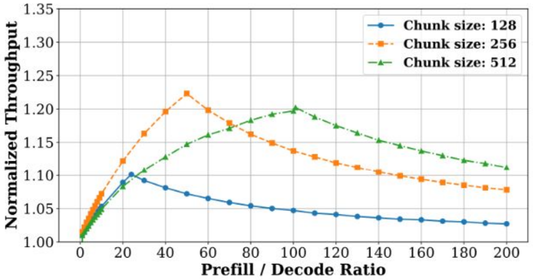
Evaluation over different workloads



(a) Sequence length = 1K, batch size = 18.

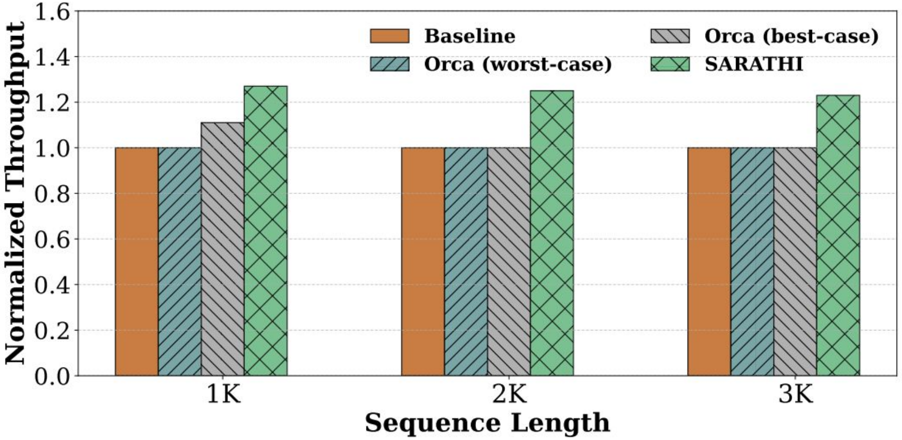


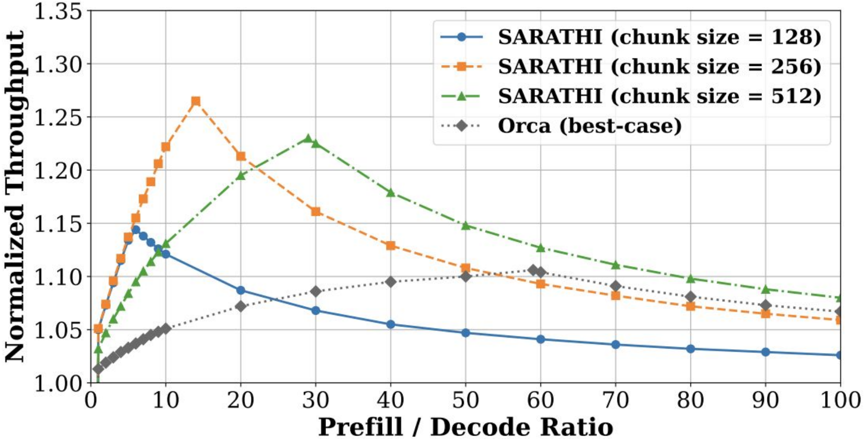
(b) Sequence length = 2K, batch size = 10.



(c) Sequence length = 3K, batch size = 6.

Comparison to Iteration Level Scheduling (ORCA)





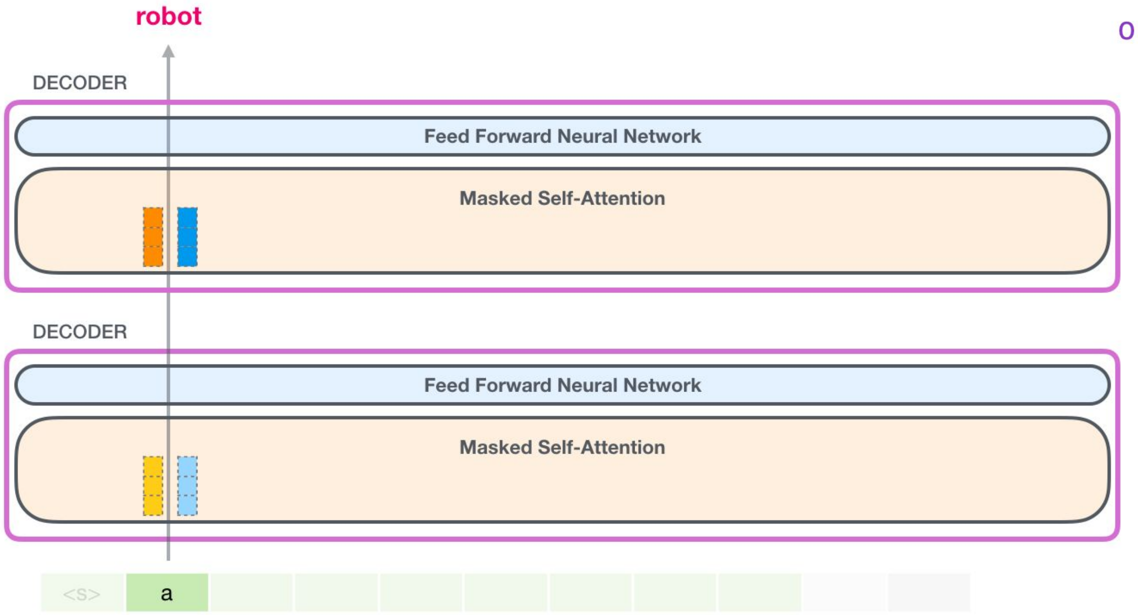
Accelerating Large Language Model Decoding with Speculative Sampling

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, Jon Jumper
(DeepMind)

Feb 2023

LLM Inference

- Generating the “next words” through transformer decoders



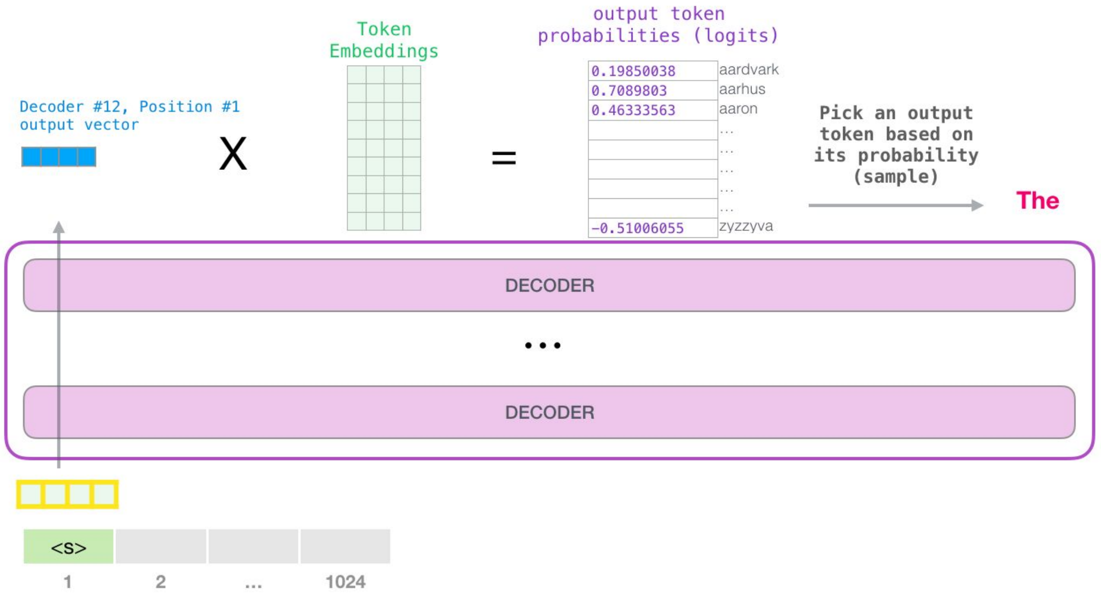
output token probabilities (logits)

0.19850038	aardvark
0.7089803	aarhus
0.46333563	aaron
	...
	...
	...
	...
	...
-0.51006055	zyzzyva

https://jalammar.github.io/illustrated-gpt2/

LLM Inference

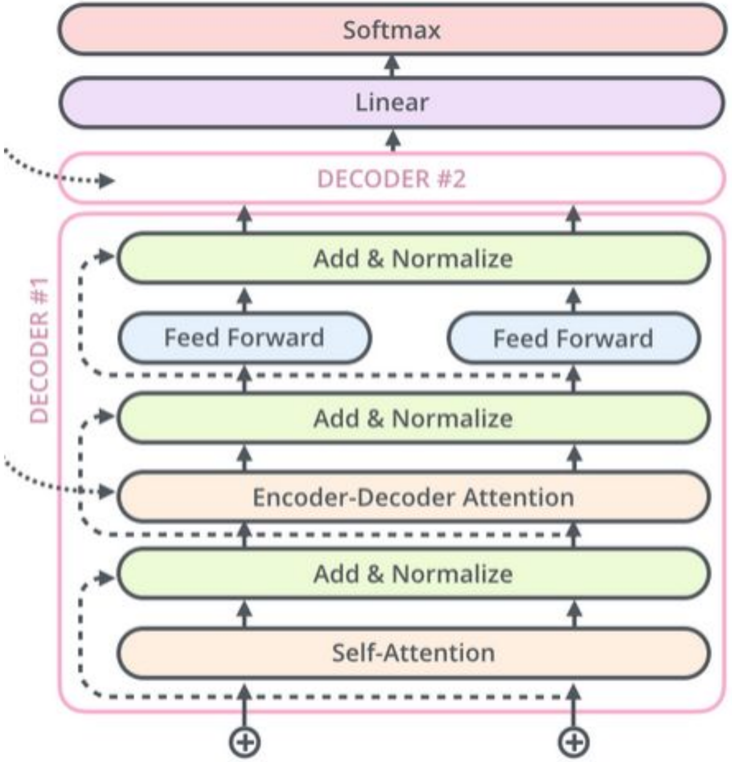
- Transformers are cool: multiple requests can be run in parallel through elegant matrix multiplications
    - New request = “add a row to the matrix”



https://jalammar.github.io/illustrated-gpt2/

Problem: LLM Inference is slow

- Generating response of *k* tokens requires *k* **sequential** decoder runs.



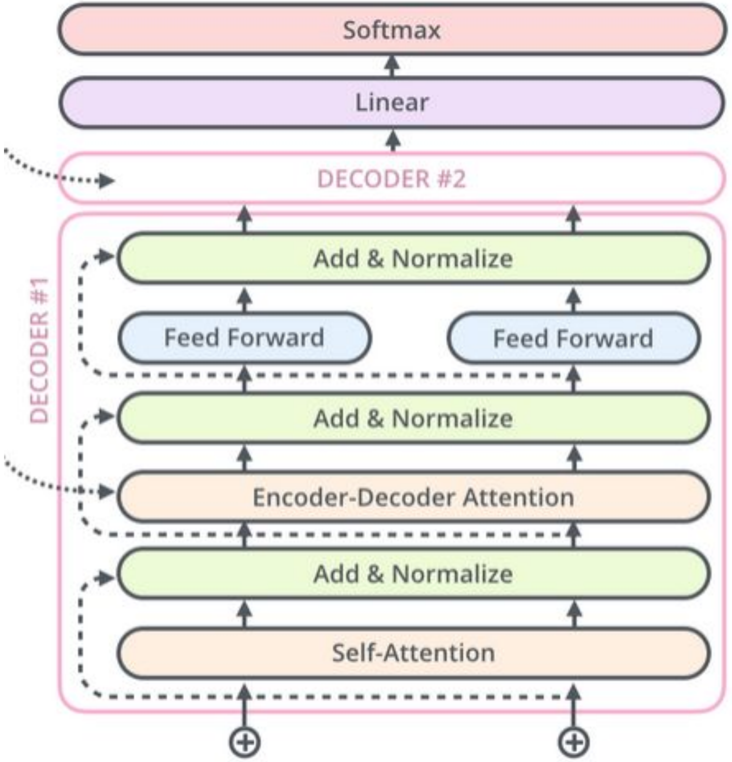
https://jalammar.github.io/illustrated-transformer/

Problem: LLM Inference is slow

- Generating response of *k* tokens requires *k* **sequential** decoder runs.

Can we do this in parallel?

- Well, we would need to know the future…



https://jalammar.github.io/illustrated-transformer/

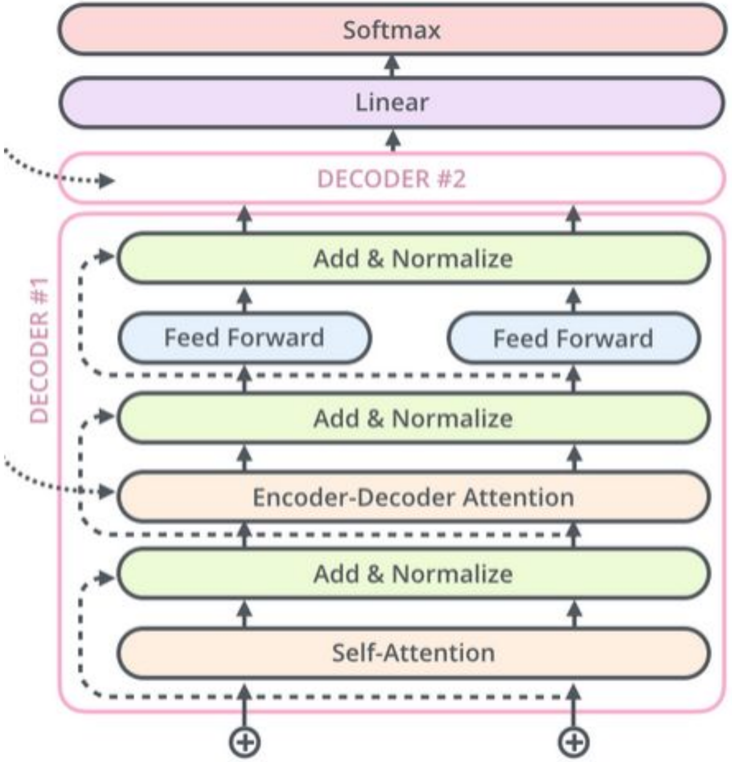
Problem: LLM Inference is slow

- Generating response of *k* tokens requires *k* **sequential** decoder runs.

Can we do this in parallel?

- Well, we would need to know the future…

Can we speculate the future?



https://jalammar.github.io/illustrated-transformer/

Observations

- Some inference steps are “easier” than others
- LLM inference is not bottlenecked by arithmetic operations
    - Bottleneck is memory bandwidth and communication

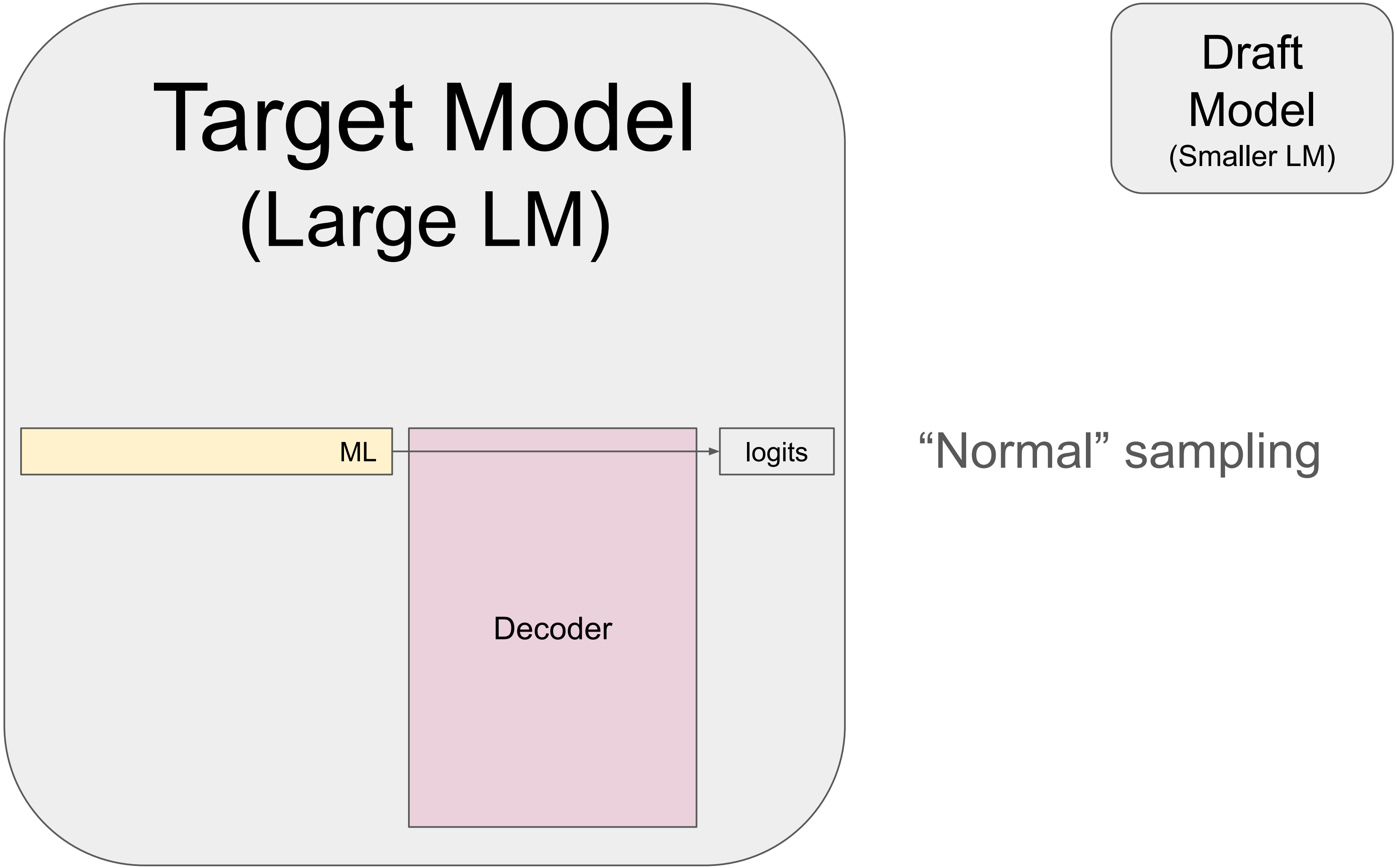
Observations

- Some inference steps are “easier” than others
- LLM inference is not bottlenecked by arithmetic operations
    - Bottleneck is memory bandwidth and communication

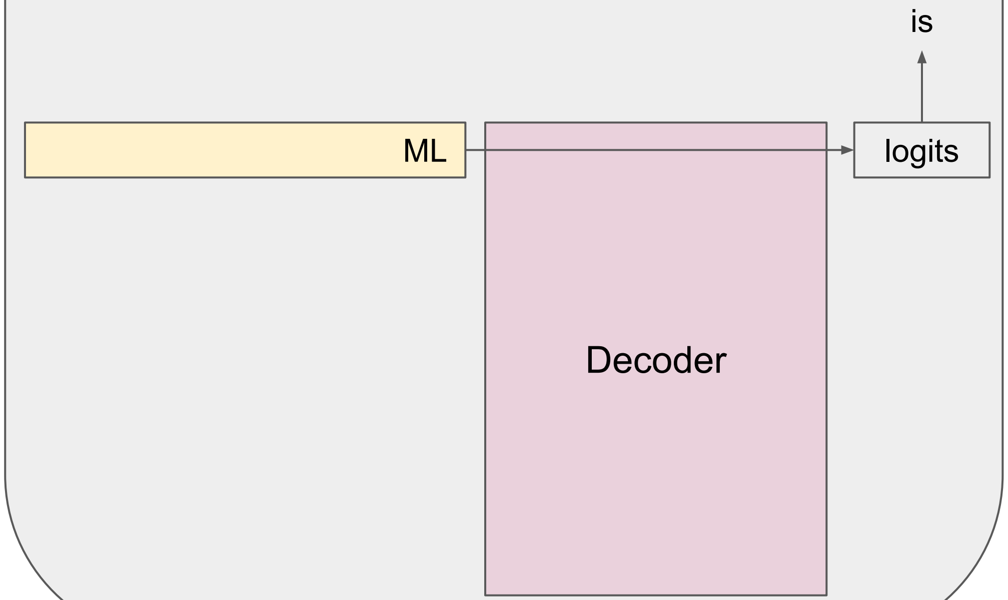
Idea: Use a smaller model to speculate the future

Target Model
(Large LM)

Draft
Model
(Smaller LM)

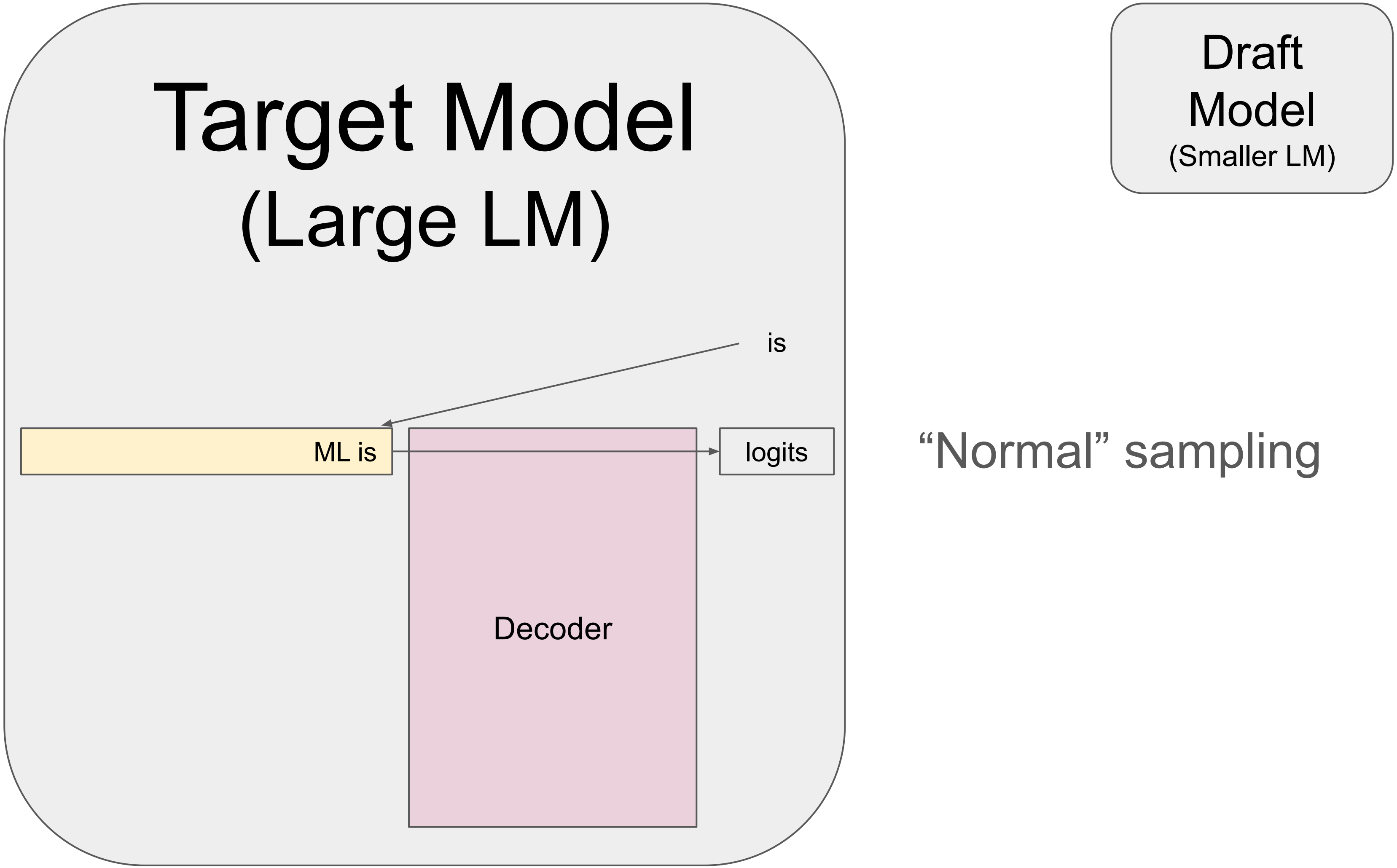

Target Model
(Large LM)
ML
logits
Decoder
Draft
Model
(Smaller LM)
"Normal" sampling

Target Model
(Large LM)

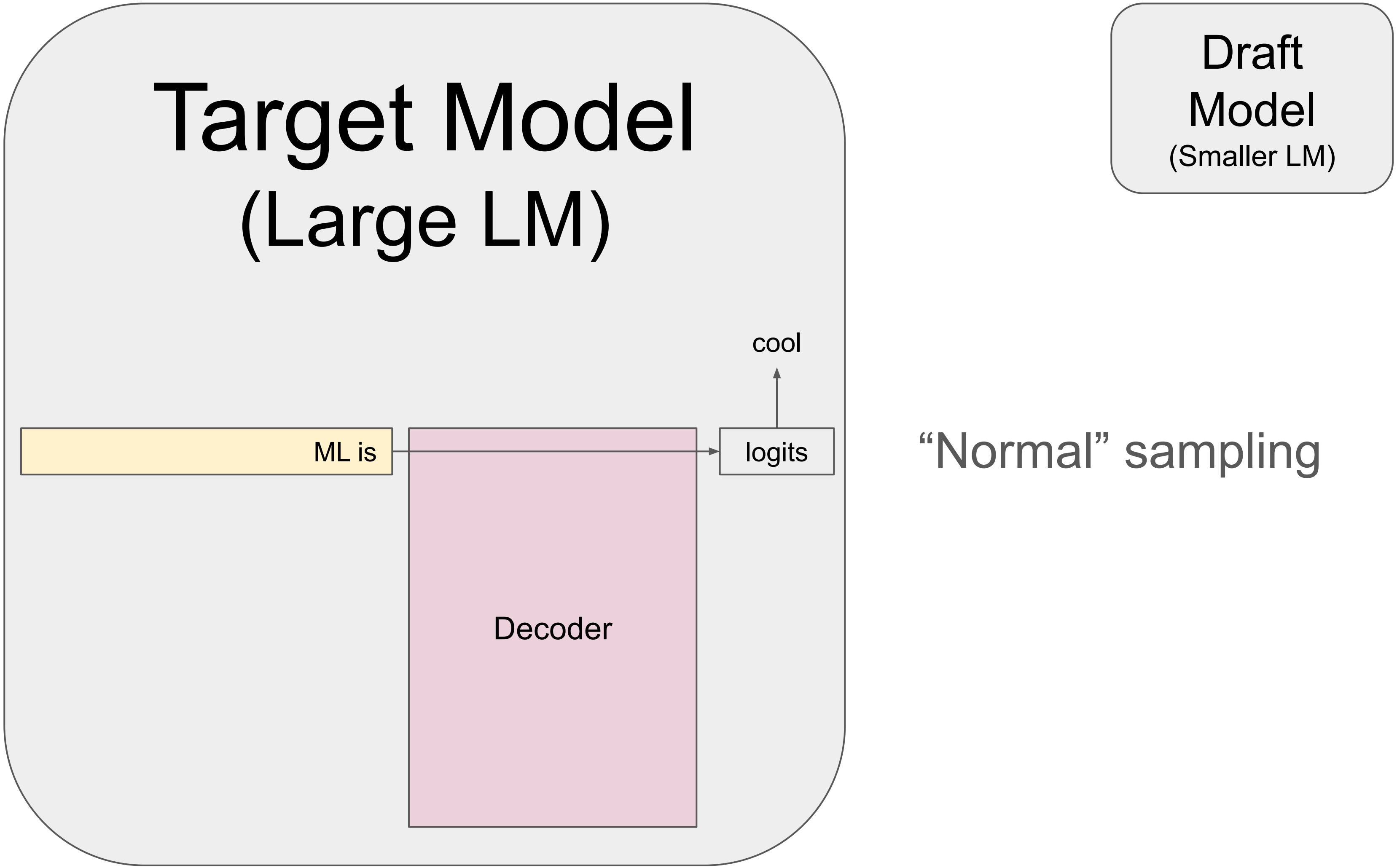


Draft
Model
(Smaller LM)

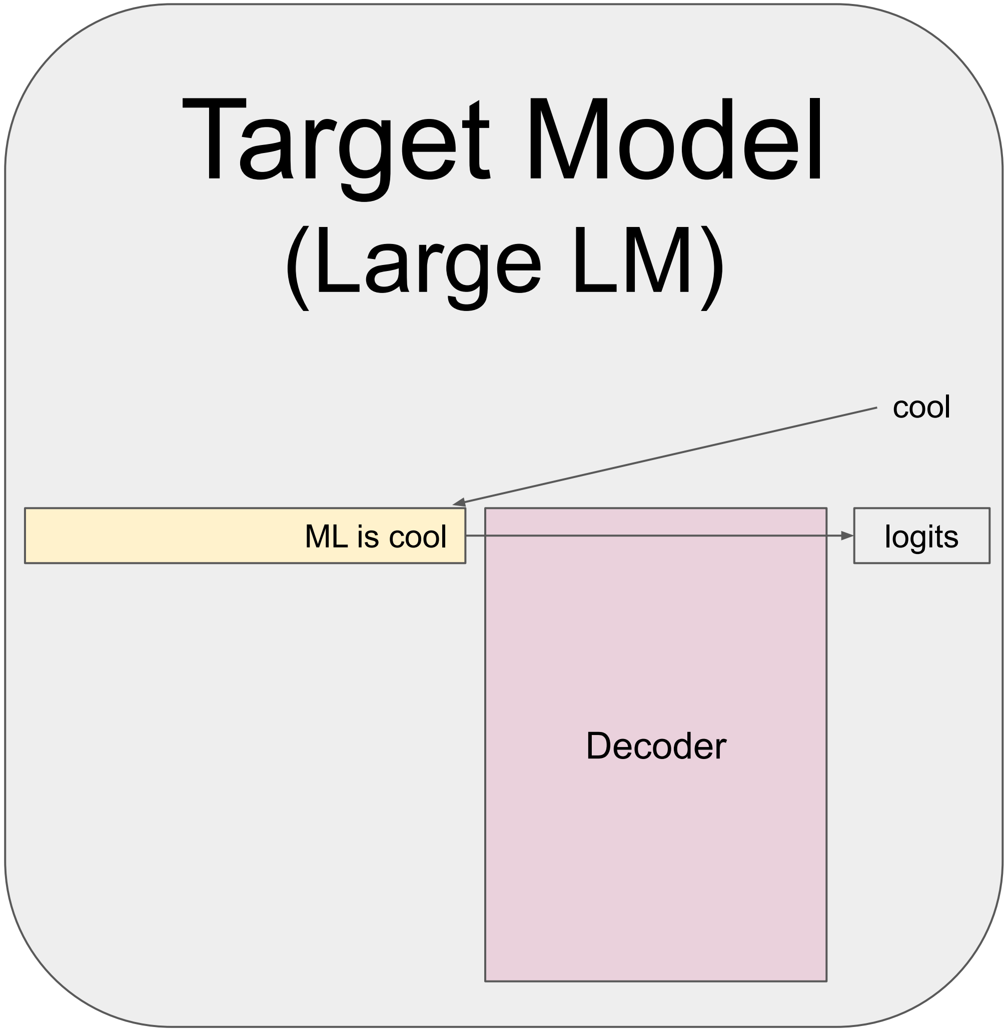
“Normal” sampling



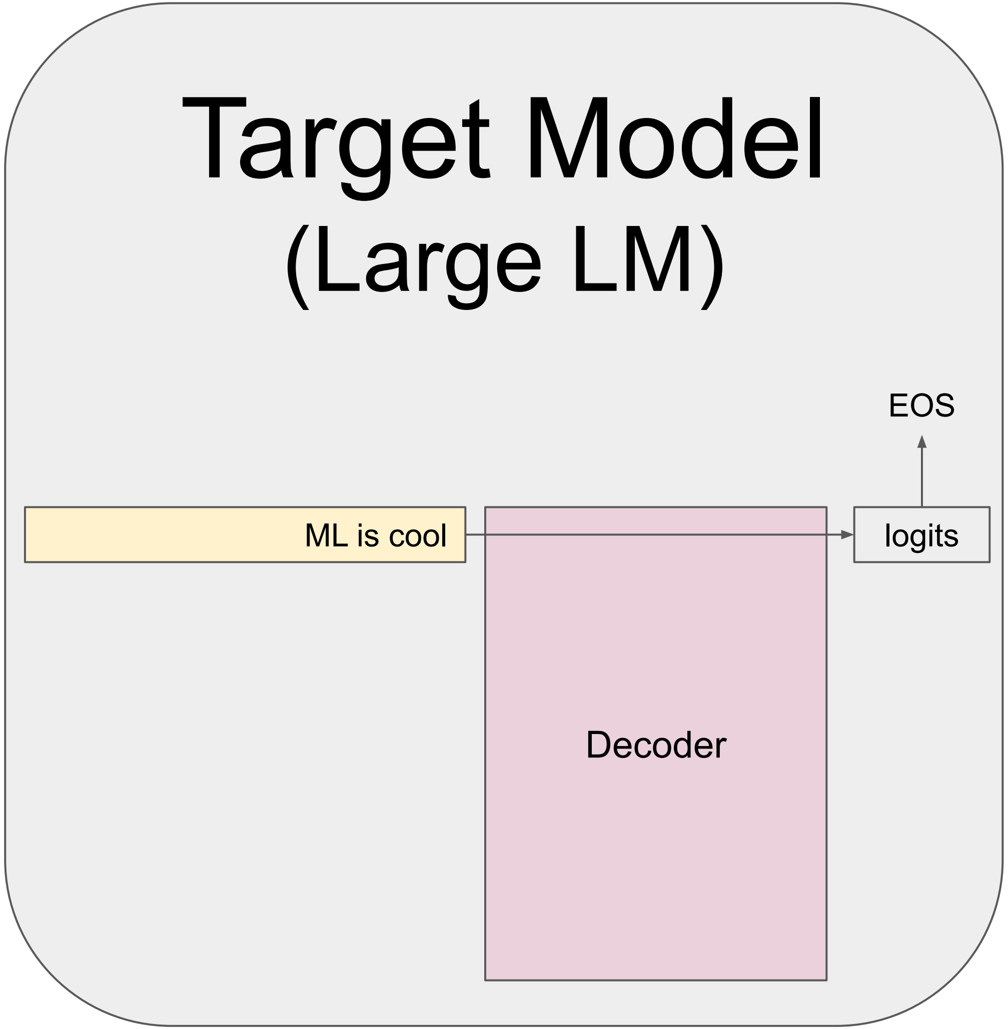
"Normal" sampling



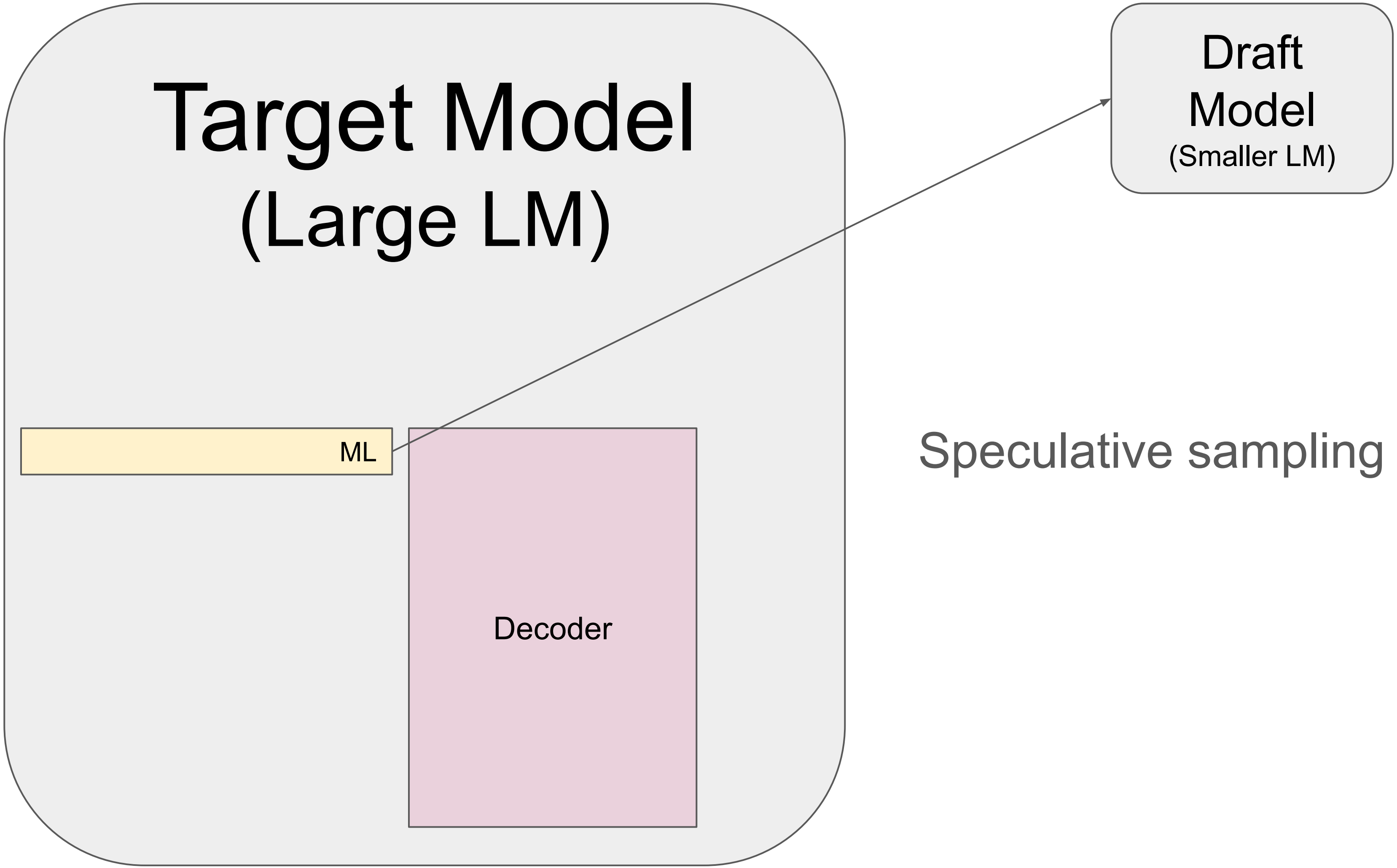
"Normal" sampling



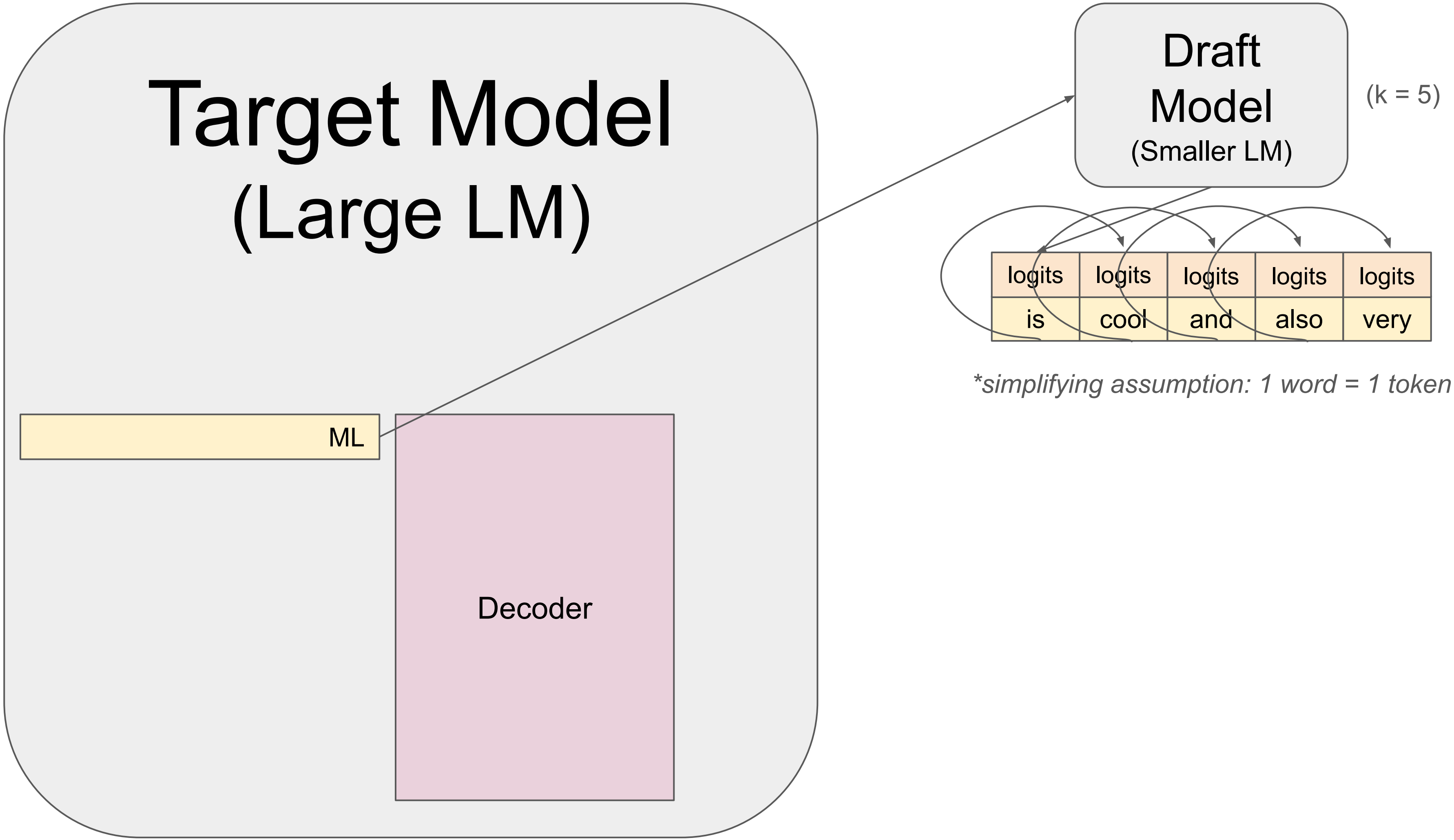
“Normal” sampling



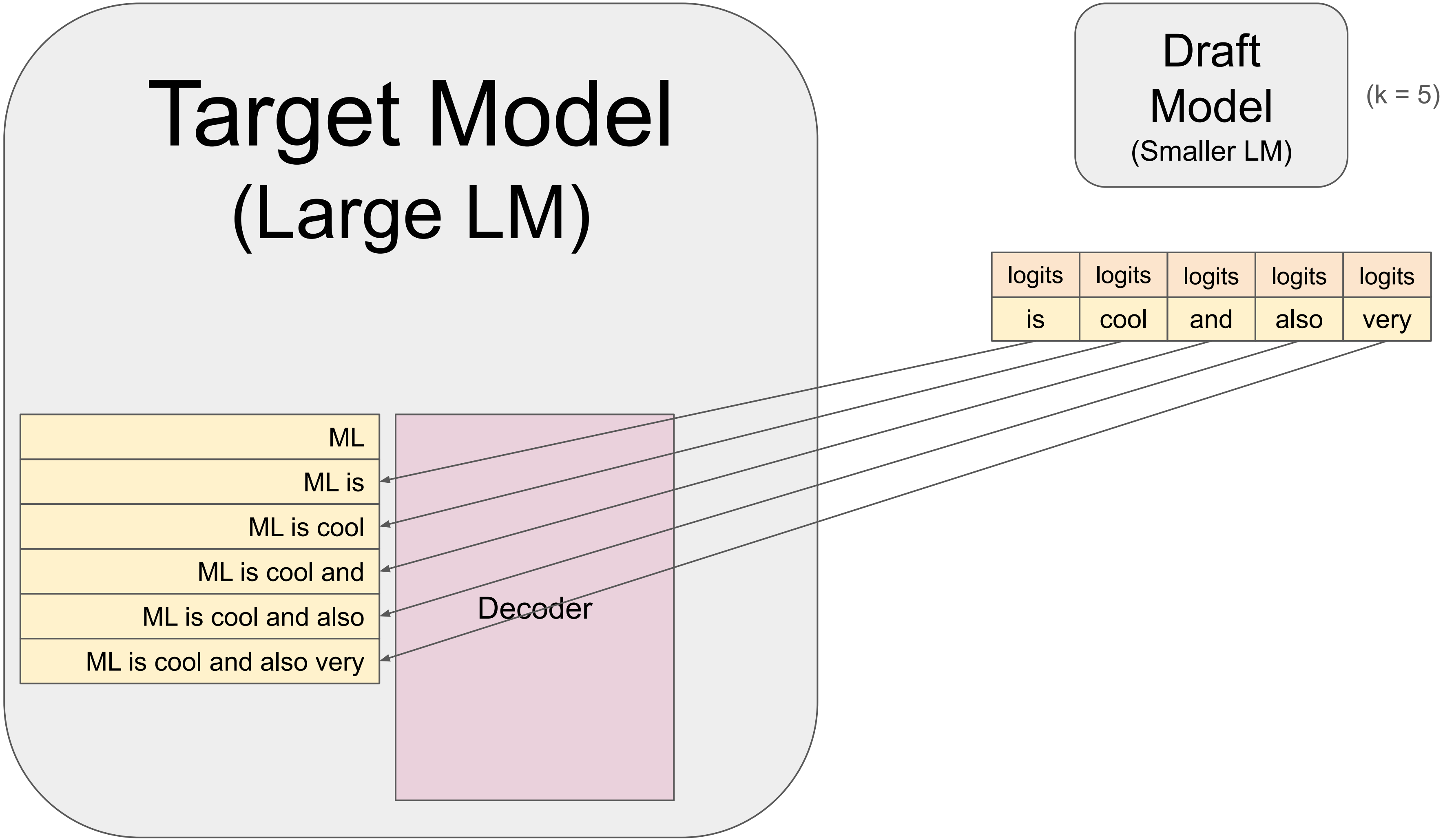
“Normal” sampling



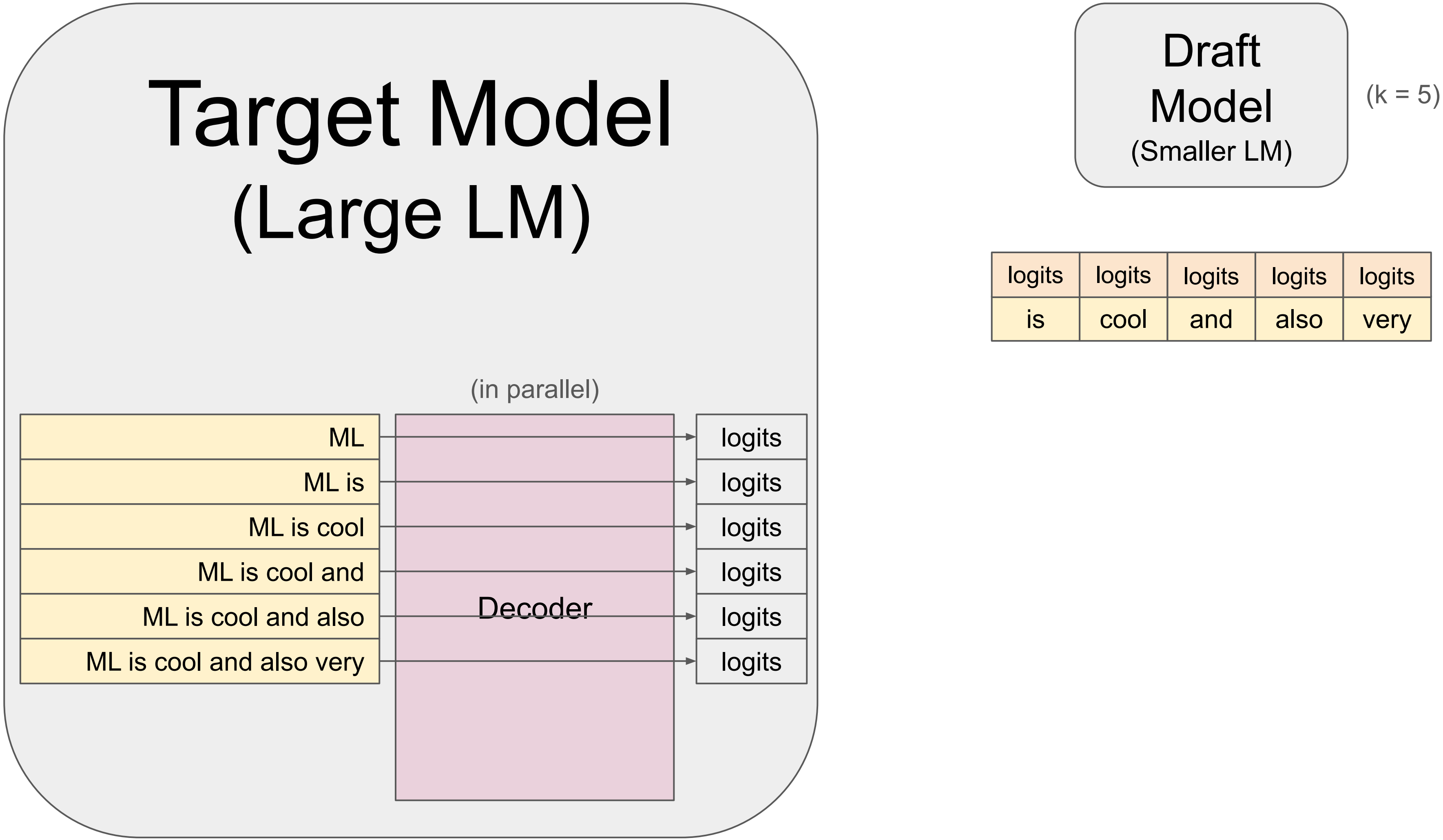
Speculative sampling

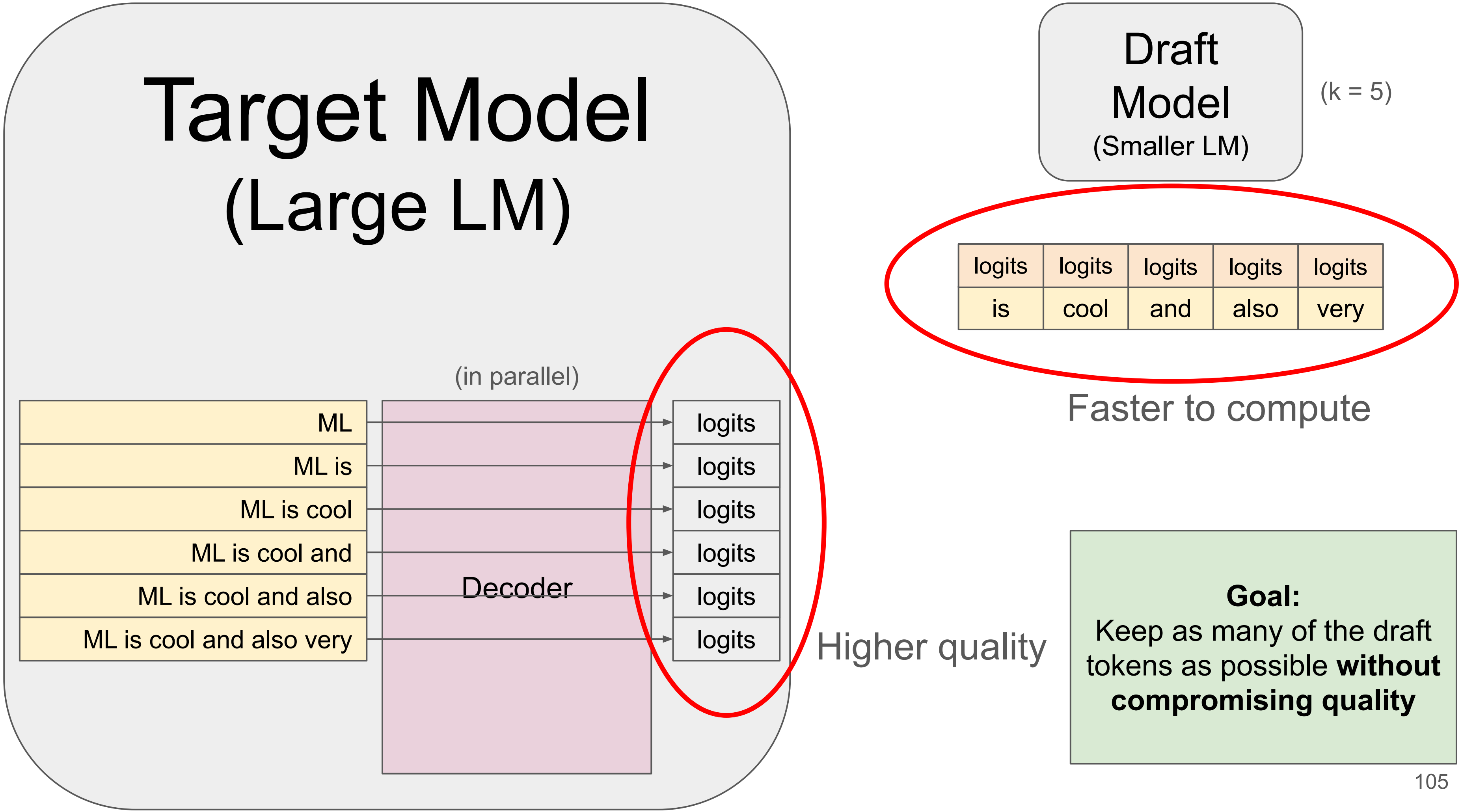


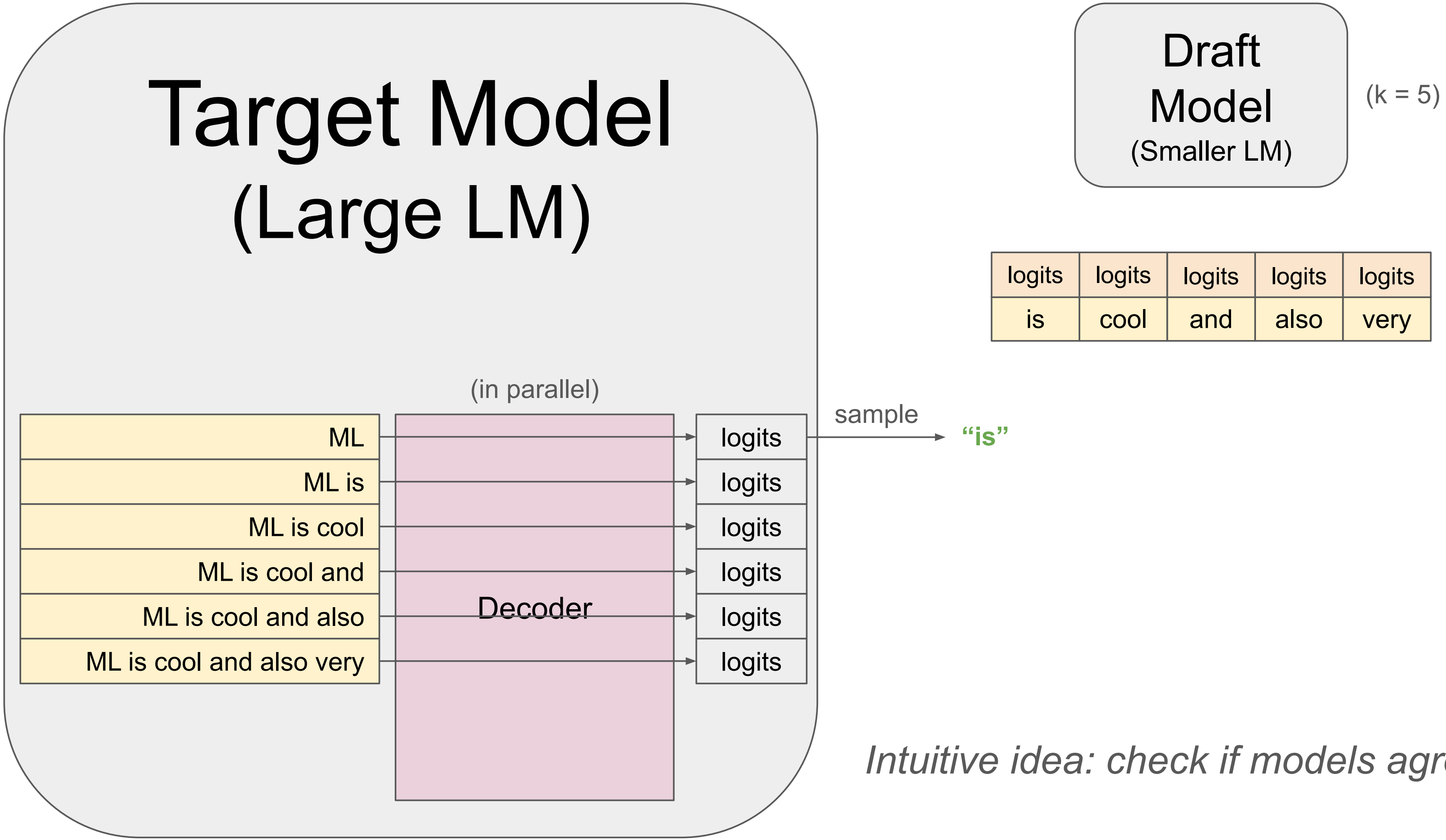
simplifying assumption: 1 word = 1 token



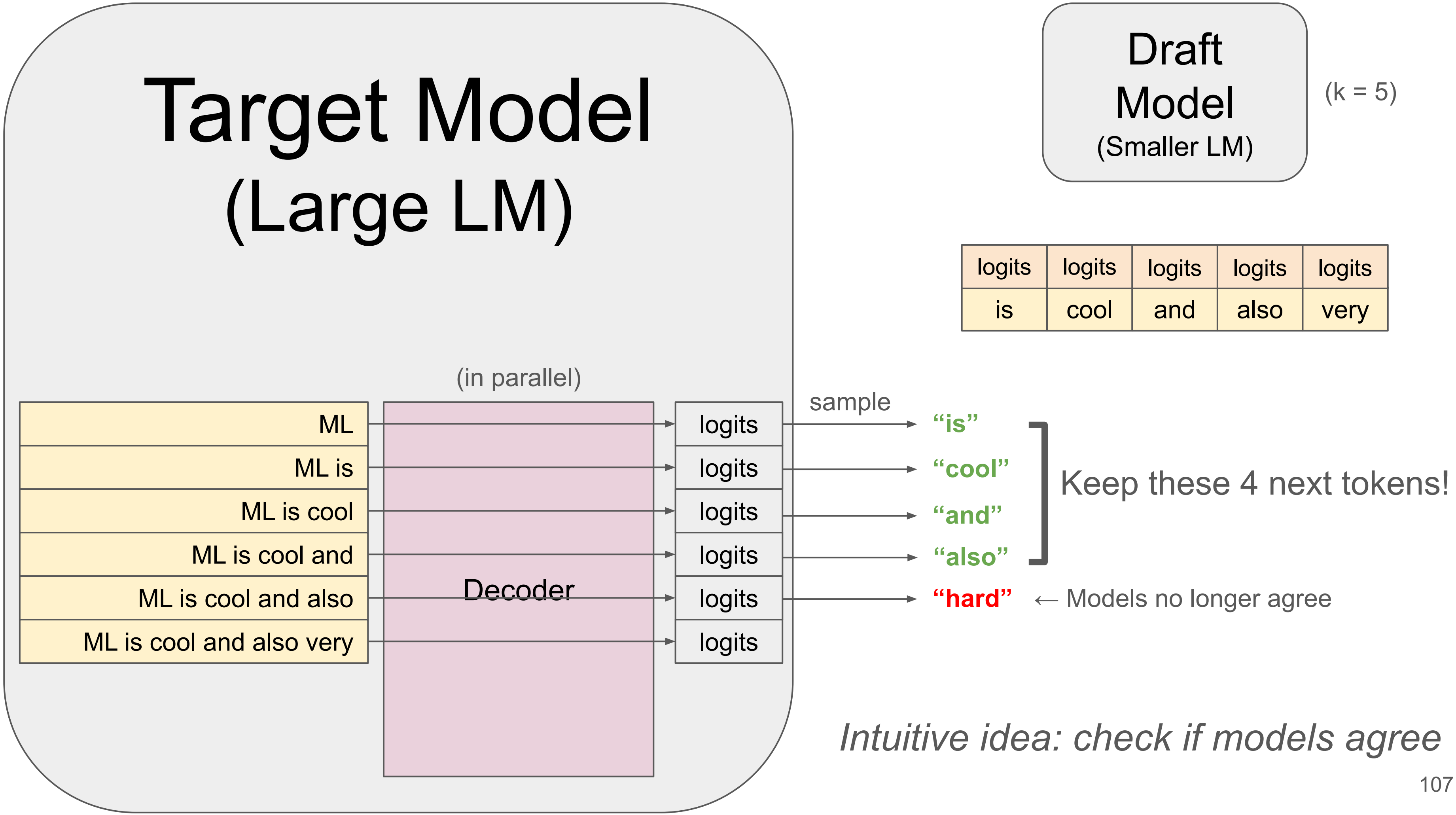
Target Model
(Large LM)
ML
ML is
ML is cool
ML is cool and
ML is cool and also
ML is cool and also very
Decoder
Draft
Model
(Smaller LM)
(k = 5)
logits
logits
logits
logits
logits
is
cool
and
also
very


Target Model
(Large LM)
(in parallel)
ML
ML is
ML is cool
ML is cool and
ML is cool and also
ML is cool and also very
Decoder
logits
logits
logits
logits
logits
logits
Draft
Model
(Smaller LM)
(k = 5)
logits
logits
logits
logits
logits
is
cool
and
also
very


Target Model
(Large LM)
(in parallel)
ML
ML is
ML is cool
ML is cool and
ML is cool and also
ML is cool and also very
Decoder
logits
logits
logits
logits
logits
logits
Higher quality
Draft
Model
(Smaller LM)
(k = 5)
logits logits logits logits logits
is cool and also very
Faster to compute
Goal:
Keep as many of the draft tokens as possible without compromising quality



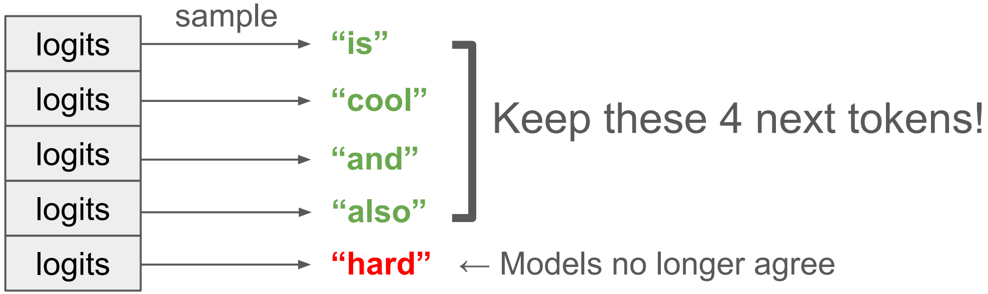
Intuitive idea: check if models agree


Target Model
(Large LM)
Draft
Model
(Smaller LM)
(k = 5)
logits | logits | logits | logits | logits
is | cool | and | also | very
(in parallel)
ML
ML is
ML is cool
ML is cool and
ML is cool and also
ML is cool and also very
Decoder
logits
logits
logits
logits
logits
logits
sample
"is"
"cool"
"and"
"also"
"hard"
Keep these 4 next tokens!
← Models no longer agree
Intuitive idea: check if models agree

How do we do this efficiently?

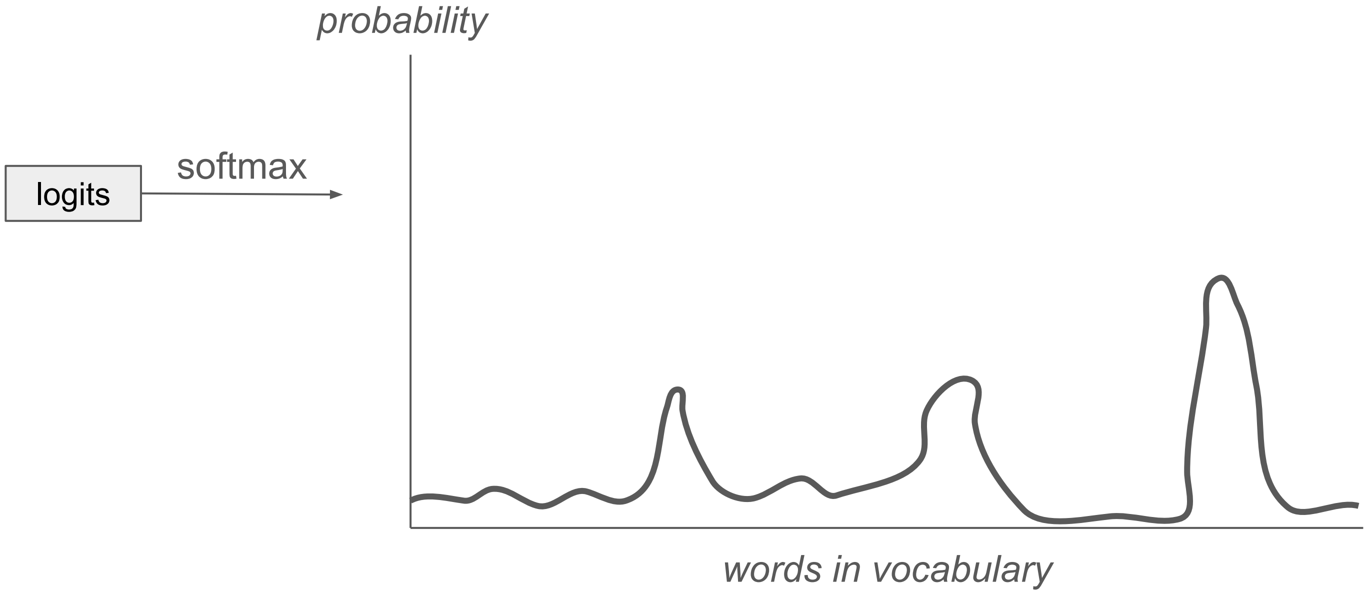
>> Modified Rejection Sampling

logits	logits	logits	logits	logits
is	cool	and	also	very



How do we do this efficiently?

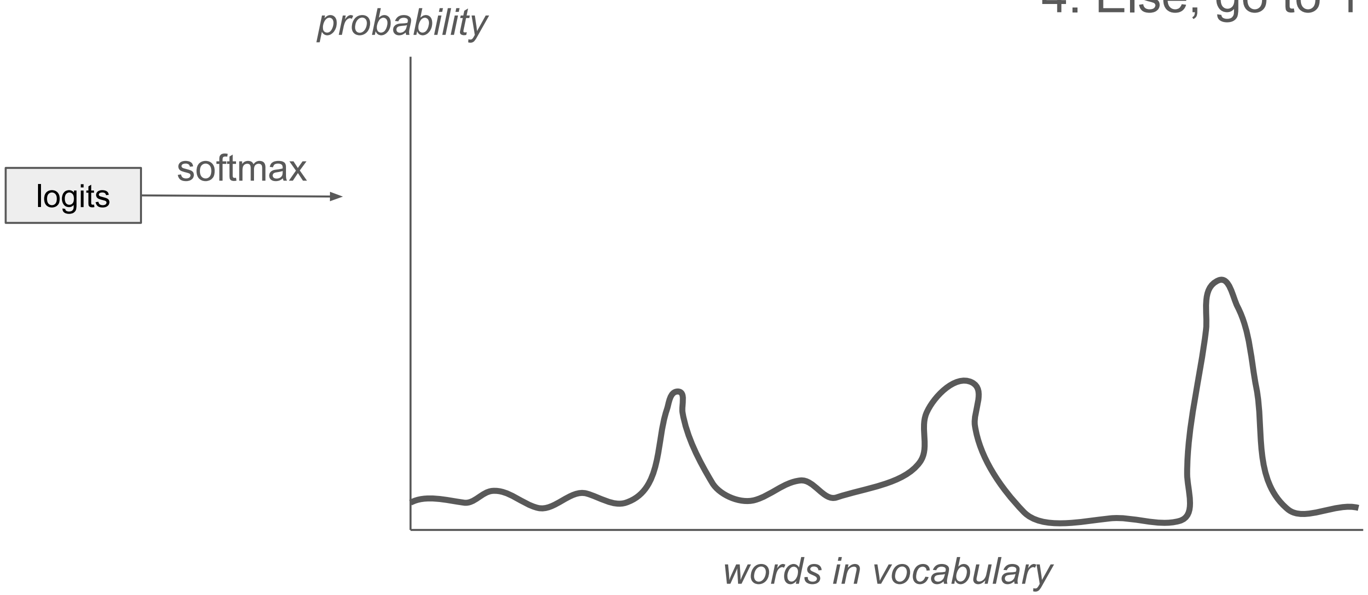
>> Modified Rejection Sampling



How do we do this efficiently?

>> Modified Rejection Sampling

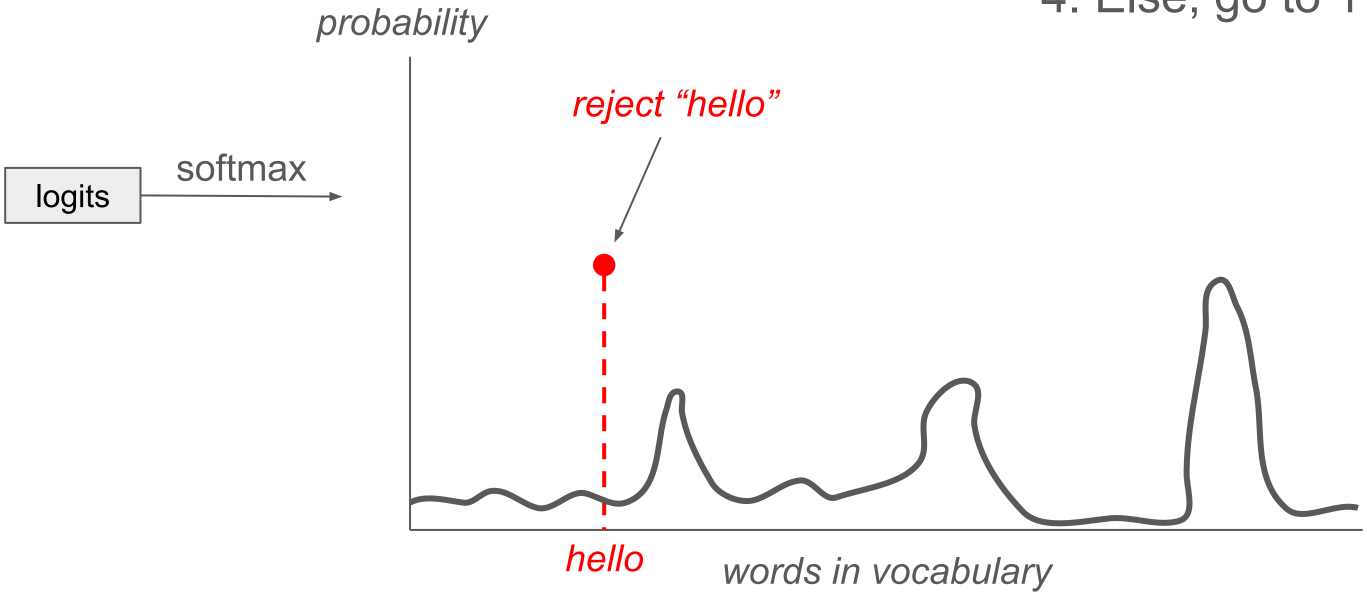
1. Pick a word *w*
2. Pick *r ~ U(0, 1)*
3. If *r* <= P(*w*), choose *w*
4. Else, go to 1



How do we do this efficiently?

>> Modified Rejection Sampling

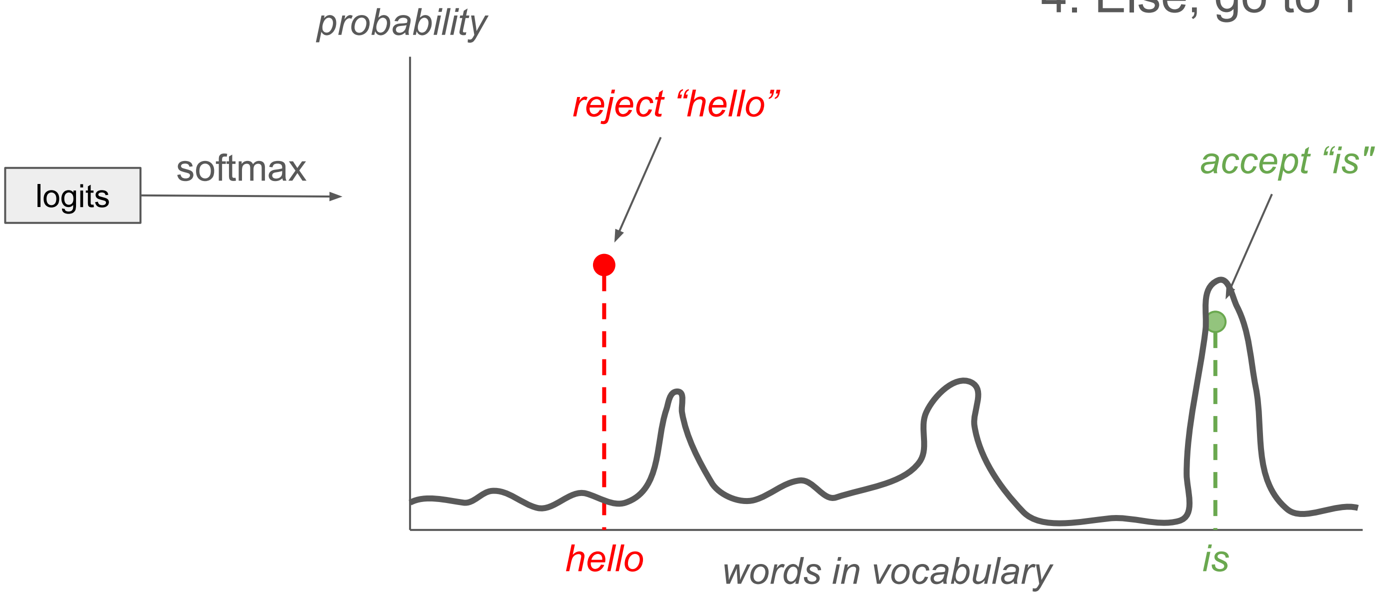
1. Pick a word *w*
2. Pick *r* ~ *U(0, 1)*
3. If *r* <= P(*w*), choose *w*
4. Else, go to 1



How do we do this efficiently?

>> Modified Rejection Sampling

1. Pick a word *w*
2. Pick *r* ~ *U(0, 1)*
3. If *r* <= P(*w*), choose *w*
4. Else, go to 1



How do we do this efficiently?

>> Modified Rejection Sampling

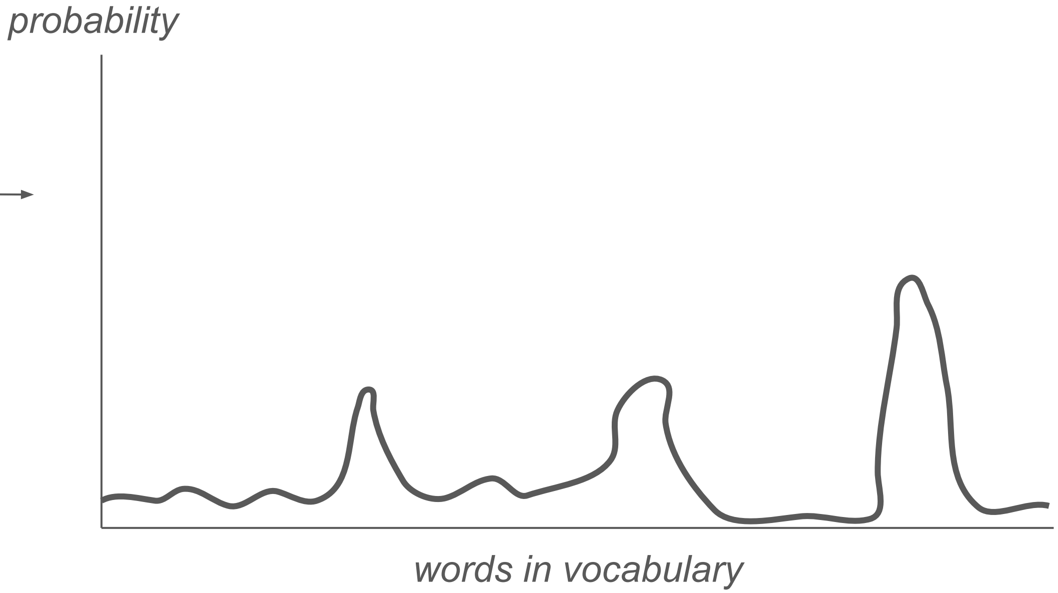
Draft Model

logits	logits	logits	logits	logits
is	cool	and	also	very

Target Model

softmax

logits
logits
logits
logits
logits
logits



How do we do this efficiently?

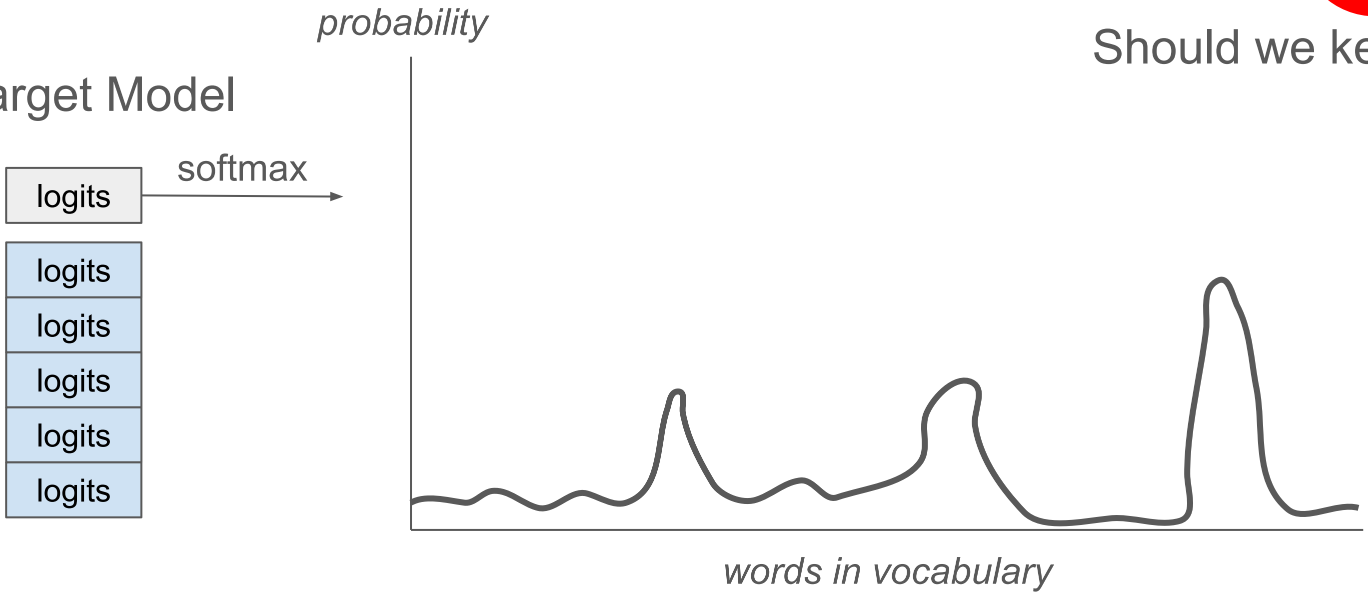
>> Modified Rejection Sampling

Draft Model

logits	logits	logits	logits	logits
is	cool	and	also	very

Should we keep this draft token?

Target Model



How do we do this efficiently?

>> Modified Rejection Sampling

Draft Model

softmax

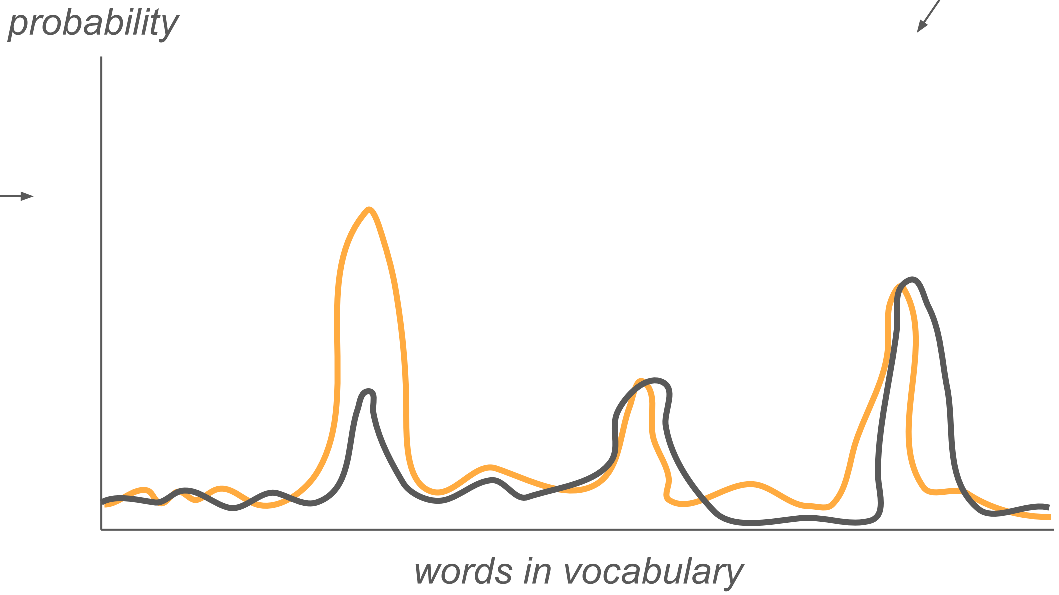
logits	logits	logits	logits	logits
is	cool	and	also	very

Target Model

logits

softmax

logits
logits
logits
logits
logits



How do we do this efficiently?

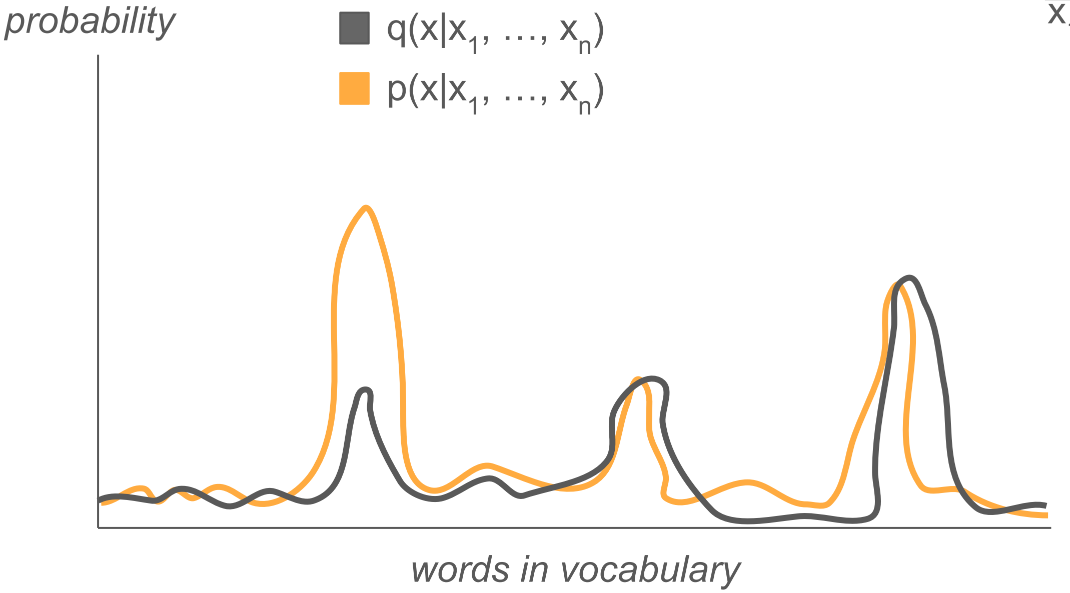
>> Modified Rejection Sampling

Draft Model

logits	logits	logits	logits	logits
is	cool	and	also	very
$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$	$\bar{x}_4$	$\bar{x}_5$

Target Model

logits
logits
logits
logits
logits
logits



How do we do this efficiently?

>> Modified Rejection Sampling

Draft Model

logits	logits	logits	logits	logits
is	cool	and	also	very
$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$	$\bar{x}_4$	$\bar{x}_5$

Target Model

logits

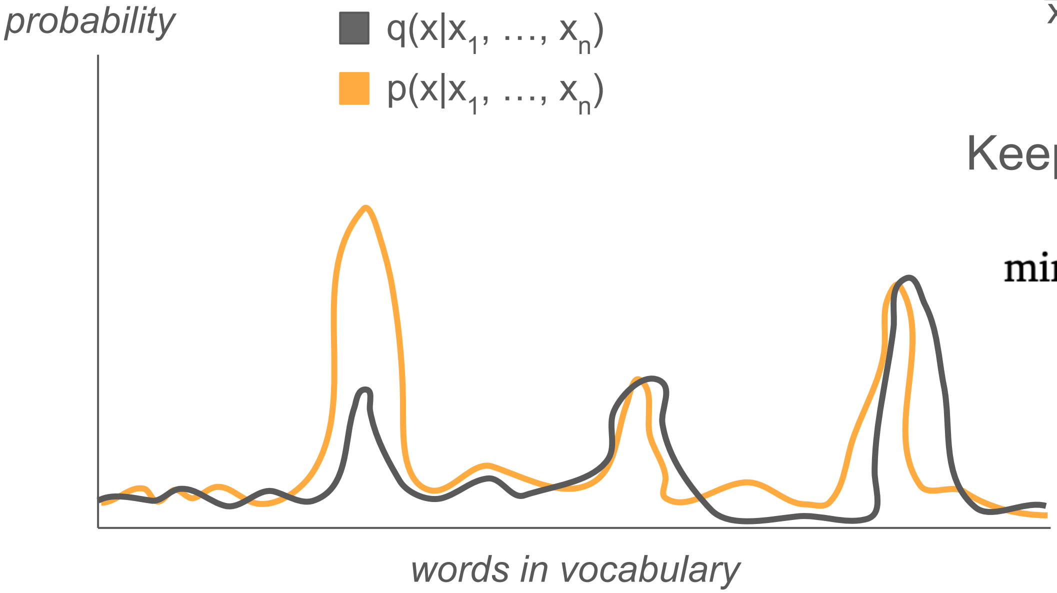
logits

logits

logits

logits

logits



Keep $\bar{x}_1$ with probability:

$$\min\left(1, \frac{q(\bar{x}_1|x_1,\ldots,x_n)}{p(\bar{x}_1|x_1,\ldots,x_n)}\right)$$

Captures:
(1) probability of word
(2) agreement of models

How do we do this efficiently?

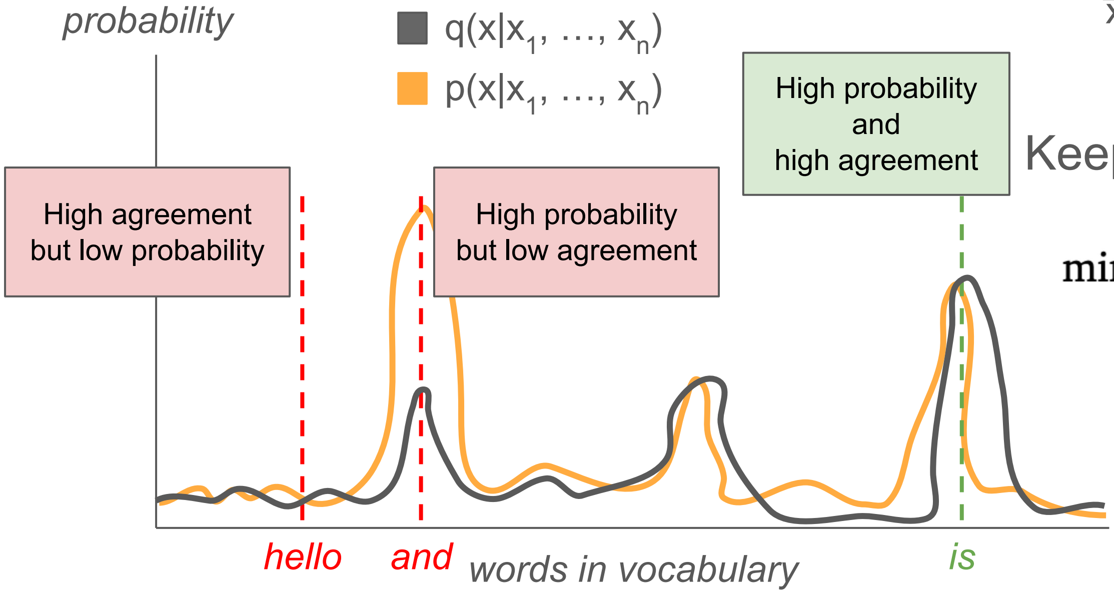
>> Modified Rejection Sampling

Draft Model

logits	logits	logits	logits	logits
is	cool	and	also	very
$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$	$\bar{x}_4$	$\bar{x}_5$

Target Model

logits
logits
logits
logits
logits
logits



Keep $\bar{x}_1$ with probability:

$$\min\left(1, \frac{q(\bar{x}_1|x_1,\ldots,x_n)}{p(\bar{x}_1|x_1,\ldots,x_n)}\right)$$

Captures:
(1) probability of word
(2) agreement of models

How do we do this efficiently?

>> Modified Rejection Sampling

Draft Model

logits	logits	logits	logits	logits
is	cool	and	also	very
$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$	$\bar{x}_4$	$\bar{x}_5$

Target Model

logits

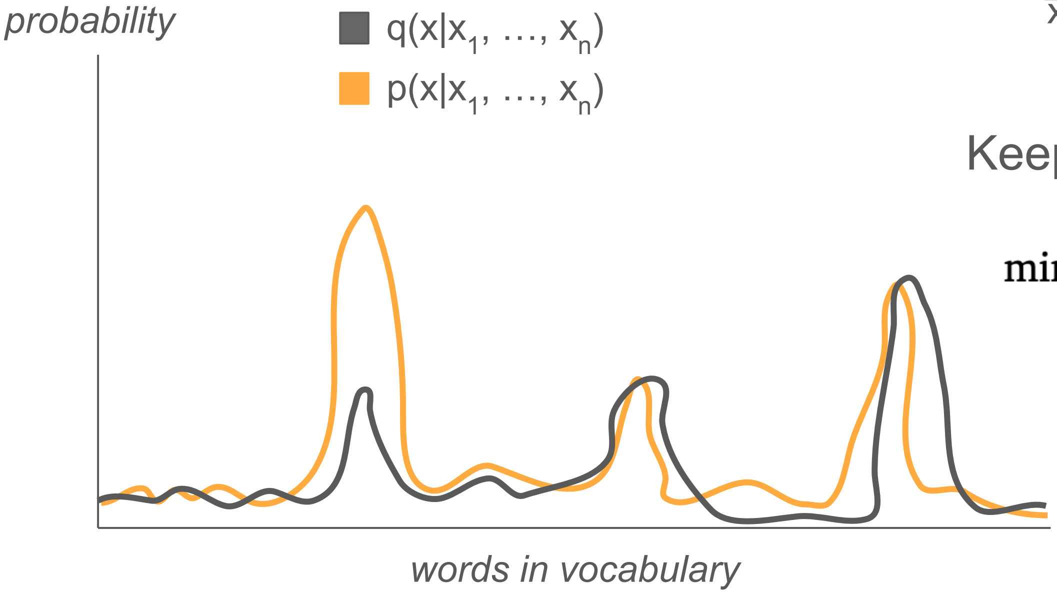
logits

logits

logits

logits

logits



Keep $\bar{x}_1$ with probability:

$$\min\left(1, \frac{q(\bar{x}_1 \mid x_1, \ldots, x_n)}{p(\bar{x}_1 \mid x_1, \ldots, x_n)}\right)$$

If accept, continue!

How do we do this efficiently?

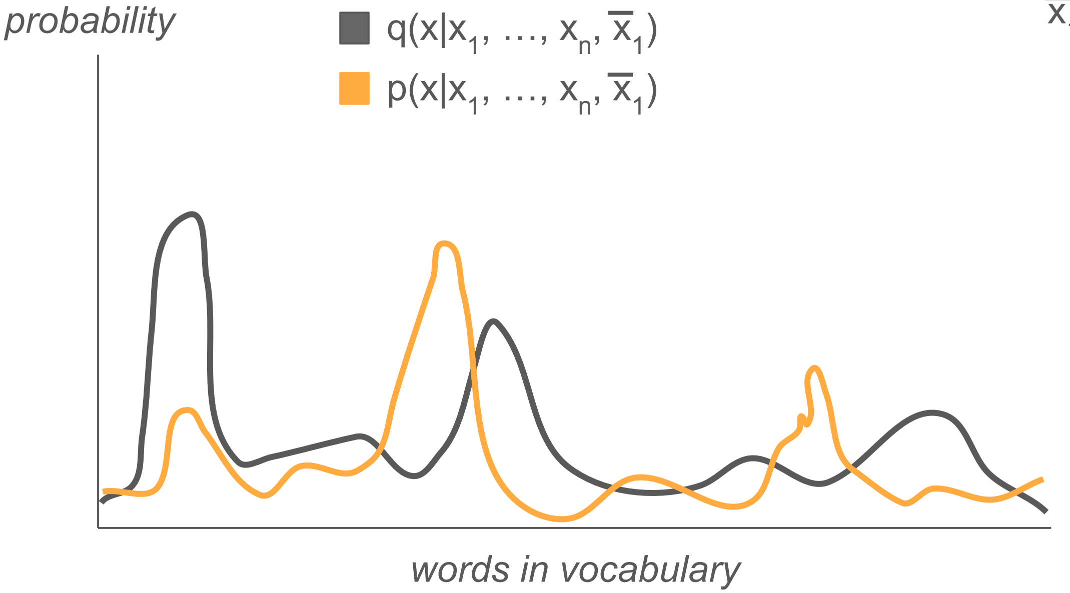
>> Modified Rejection Sampling

Draft Model

logits	logits	logits	logits	logits
is	cool	and	also	very
$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$	$\bar{x}_4$	$\bar{x}_5$

Target Model

logits
logits
logits
logits
logits
logits



If accept, continue!

How do we do this efficiently?

>> Modified Rejection Sampling

Draft Model

logits	logits	logits	logits	logits
is	cool	and	also	very
$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$	$\bar{x}_4$	$\bar{x}_5$

Target Model

logits

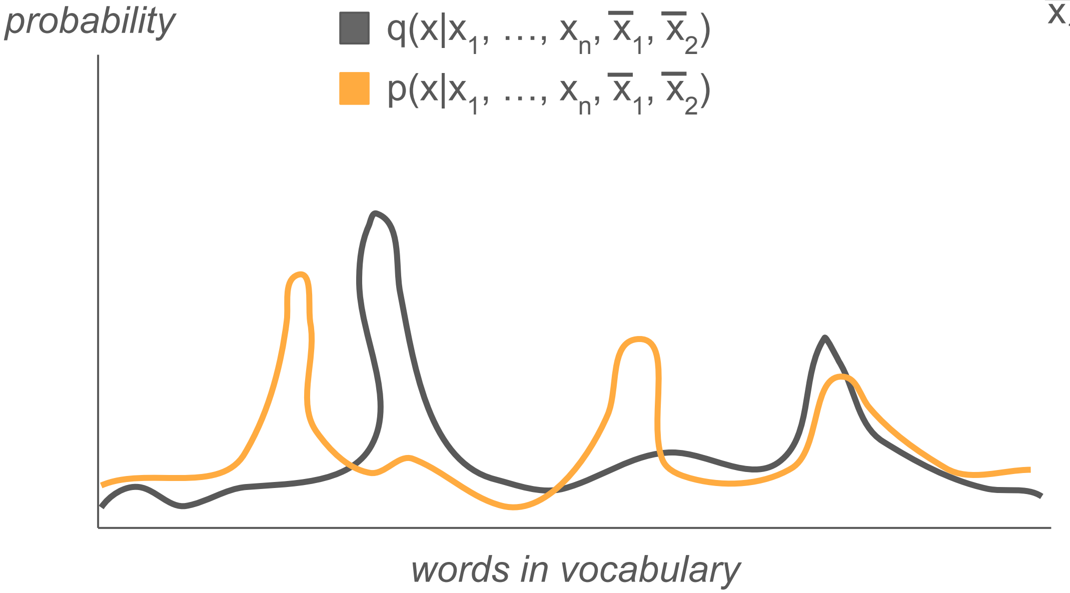
logits

logits

logits

logits

logits



If accept, continue!

How do we do this efficiently?

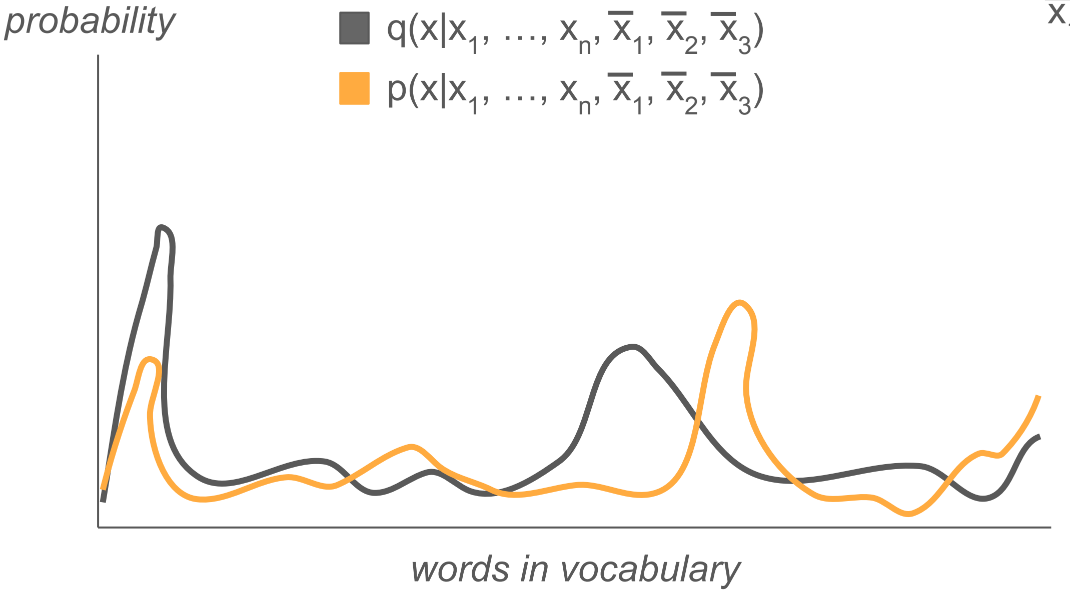
>> Modified Rejection Sampling

Draft Model

logits	logits	logits	logits	logits
is	cool	and	also	very
$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$	$\bar{x}_4$	$\bar{x}_5$

Target Model

logits
logits
logits
logits
logits
logits



If accept, continue!

How do we do this efficiently?

>> Modified Rejection Sampling

Draft Model

logits	logits	logits	logits	logits
is	cool	and	also	very
$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$	$\bar{x}_4$	$\bar{x}_5$

Target Model

logits

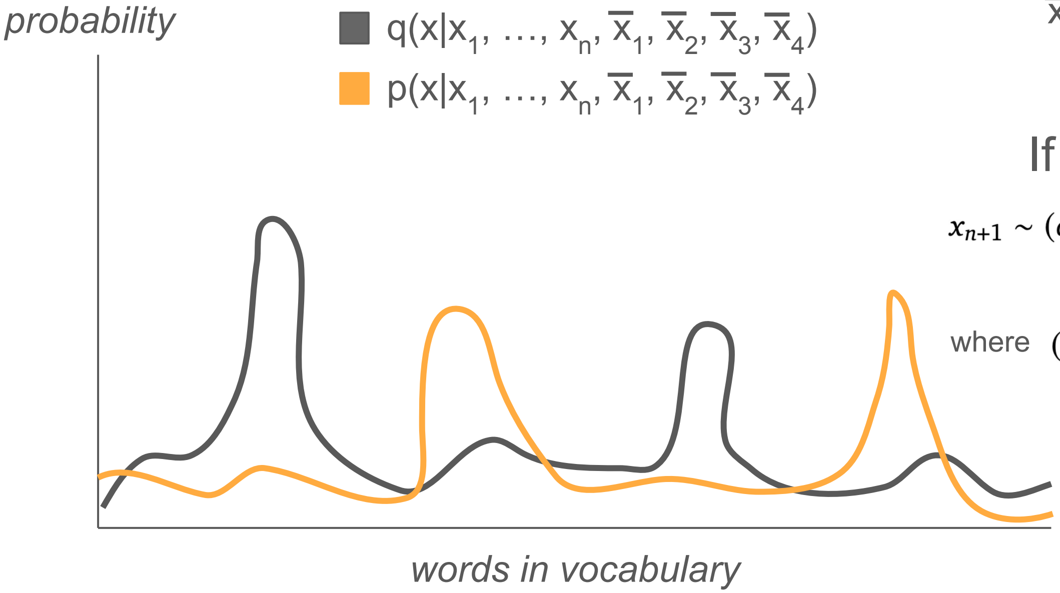
logits

logits

logits

logits

logits



If reject, sample from:

$$x_{n+1} \sim (q(x|x_1, \ldots, x_n) - p(x|x_1, \ldots, x_n))_+$$

where $$(f(x))_+ = \frac{\max(0, f(x))}{\sum_x \max(0, f(x))}$$

How do we do this efficiently?

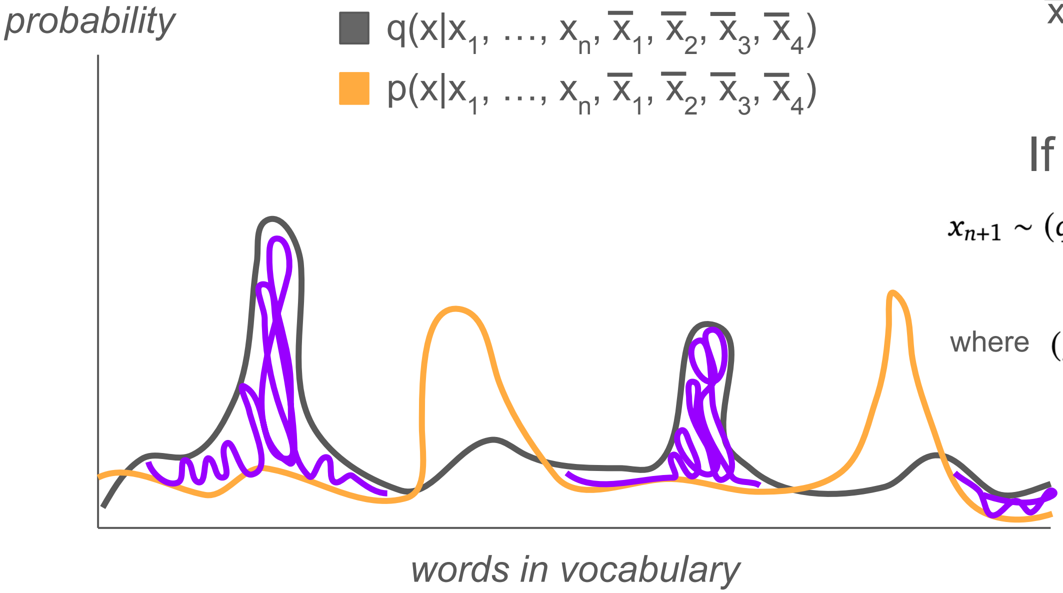
>> Modified Rejection Sampling

Draft Model

logits	logits	logits	logits	logits
is	cool	and	also	very
$\bar{x}_1$	$\bar{x}_2$	$\bar{x}_3$	$\bar{x}_4$	$\bar{x}_5$

Target Model

logits
logits
logits
logits
logits
logits



If reject, sample from:

$$x_{n+1} \sim (q(x|x_1, \ldots, x_n) - p(x|x_1, \ldots, x_n))_+$$

where $(f(x))_+ = \dfrac{\max(0, f(x))}{\sum_x \max(0, f(x))}$

Intuitively, this is like rejection sampling on purple regions

TL;DR - Modified Rejection Sampling

- Worst case: guarantees 1 token is generated
- Best case: can generate *k* tokens at once

- Mathematically proven that tokens are sampled according to target model distribution *(Theorem 1 in Appendix)*
  - a.k.a No reduction in model quality!

TL;DR - Speculative Sampling

- Key insights:
    - Decode **k tokens in 1 step** (instead of just 1 token)
    - Use a **smaller model** to speculate the larger model
- Key assumption:
    - Computing logits of *k* tokens in parallel **has similar latency** to sampling 1 token

- No modification to original (target) model
- Effective for **small batch sizes**
    - Batch size of *n* requires *n*k* speculative logits computations
    - → OOM for large *n*

Evaluation - Set Up

- Target model: Chinchilla (70b params)
- Draft model: smaller Chinchilla (4b params)
  - Generally, smaller version of target model works well
  - But must be careful in distributed setups

- Eg. Chinchilla-70b used 16 TPU v4s
  - Naively sharding Chichilla-4b across 16 TPUs **increases** latency
  - Chinchilla-4b has lowest latency on 4 TPUs
  - So a “wider” version of Chinchilla was used for draft, allowing it to be sharded across 16 TPUs effectively

Evaluation - Performance & Speed

Table 1 | **Chinchilla performance and speed on XSum and HumanEval with naive and speculative sampling at batch size 1 and $K = 4$.** XSum was executed with nucleus parameter $p = 0.8$, and HumanEval with $p = 0.95$ and temperature 0.8.

Sampling Method	Benchmark	Result	Mean Token Time	Speed Up
ArS (Nucleus)	XSum (ROUGE-2)	0.112	14.1ms/Token	1×
SpS (Nucleus)		0.114	7.52ms/Token	1.92×
ArS (Greedy)	XSum (ROUGE-2)	0.157	14.1ms/Token	1×
SpS (Greedy)		0.156	7.00ms/Token	2.01×
ArS (Nucleus)	HumanEval (100 Shot)	45.1%	14.1ms/Token	1×
SpS (Nucleus)		47.0%	5.73ms/Token	2.46×

ArS: Autoregressive Sampling (“Normal” Sampling)
SpS: Speculative Sampling
Sampling methods: https://towardsdatascience.com/decoding-strategies-that-you-need-to-know-for-response-generation-ba95ee0faadc

Evaluation - Trade offs with larger *k*

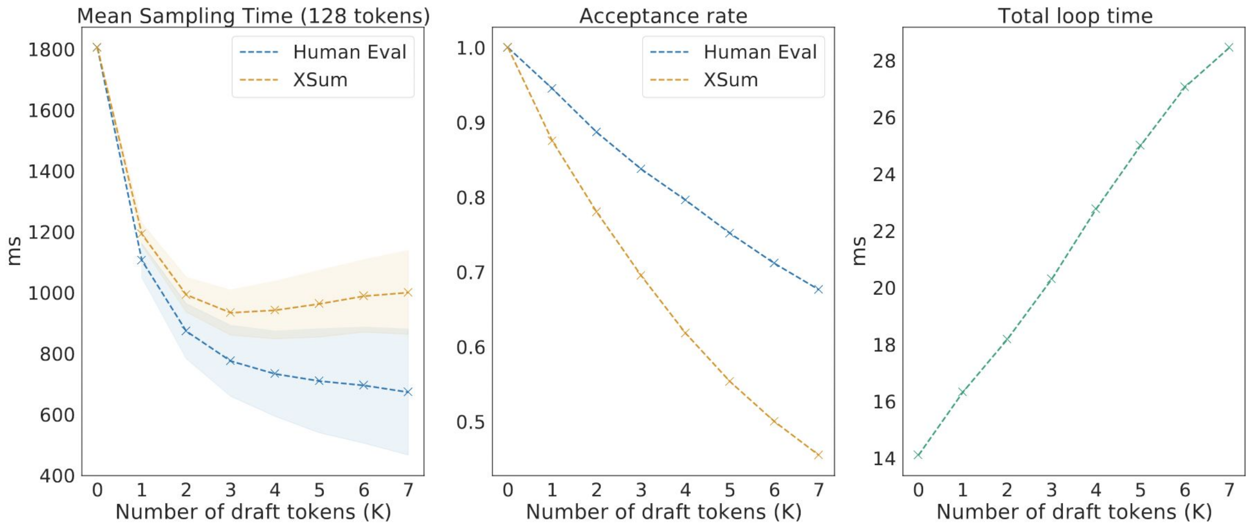


Figure 1 | **Left:** The average time to generate 128 tokens, with standard deviation. Note that as K increases, the overall speedup plateaus or even regresses, with XSum being optimal at $K = 3$. The variance consistently increases with K. **Middle:** The average number of tokens accepted divided by $K + 1$ – this serves as a measure of the overall efficiency of the modified rejection scheme, which decreases with the lookahead. **Right:** Average time per loop increases approximately linearly with K due to the increased number of model calls. Note that the gradient is slightly higher than the sampling speed of the draft model, due to additional overheads in nucleus decoding.

Conclusion

Speculative sampling generates **multiple tokens** in 1 step

- with **no reduction in quality**
- by speculating on a **smaller model**
- using a novel **modified rejection sampling** technique
- achieving **2-2.5x decoding speedup** in a distributed setup

Questions?

(hidden) Proof that target distribution is preserved

Proofs

Theorem 1 (Modified Rejection Sampling recovers the target distribution)**.** *Given discrete distributions q, p and a single draft sample $\tilde{x} \sim p$, let X be the final resulting sample. For $X = x$ to be true, we must either sample $\tilde{x} = x$ and then accept it, or resample it after $\tilde{x}$ (of any value) is rejected. Hence:*

$$\mathbb{P}(X = x)$$

$$= \mathbb{P}(\tilde{x} = x)\mathbb{P}(\tilde{x}\ accepted|\tilde{x} = x) + \mathbb{P}(\tilde{x}\ rejected)\mathbb{P}(X = x|\tilde{x}\ rejected)$$

For the first term, we apply the acceptance rule:

$$\mathbb{P}(\tilde{x} = x)\mathbb{P}(\tilde{x}\ accepted|\tilde{x} = x)$$

$$= p(x) \min\left(1, \frac{q(x)}{p(x)}\right)$$

$$= \min\left(p(x), q(x)\right)$$

For the second conditional term, we apply the resampling rule:

$$\mathbb{P}(X = x|\tilde{x}\ rejected) = (q(x) - p(x))_+$$

Where $(.)_+$ *denotes:*

$$(f(x))_+ = \frac{\max(0, f(x))}{\sum_x \max(0, f(x))}$$

Finally, we calculate the probability of rejection:

$$\mathbb{P}(\tilde{x}\ rejected) = 1 - \mathbb{P}(\tilde{x}\ accepted)$$

$$= 1 - \sum_{x'} \mathbb{P}(X = x', \tilde{x}\ accepted)$$

$$= 1 - \sum_{x'} \min(p(x'), q(x'))$$

$$= \sum_{x'} \max(0, q(x') - p(x'))$$

$$= \sum_{x'} q(x') - \min(p(x'), q(x'))$$

$$= \sum_{x'} \max(0, q(x') - p(x'))$$

This is equal to the denominator of $(q(x) - p(x))_+$*, so:*

$$\mathbb{P}(\tilde{x}\ rejected)\mathbb{P}(X = x|\tilde{x}\ rejected) = \max(0, q(x) - p(x))$$

Hence:

$$\mathbb{P}(X = x)$$

$$= \min(p(x), q(x)) + \max(0, q(x) - p(x))$$

$$= q(x)$$

and we have recovered the desired target.

(hidden) Real world implementations

TGI gh issue:

- https://github.com/huggingface/text-generation-inference/issues/729
- https://github.com/huggingface/text-generation-inference/issues/1169
- Discusses using Medusa (multi-head) instead of separate draft model

vLLM gh issue:

- https://github.com/vllm-project/vllm/pull/2188
- Discusses implementing with draft model