Efficiency

CSE 5525: Foundations of Speech and Natural Language Processing

https://shocheen.github.io/courses/cse-5525-spring-2025

The Ohio State University

Slide Credits: Graham Neubig, Greg Durrett, Nishant Subramani, Sofia Serrano

Logistics

- Final project:
  - Mid-project report is due March 28.
  - Project presentations: April 16, 18*
  - Final project report due date: Tentatively April 25.

- There will be a quiz every week (either or both days) starting next week.
  - Next week quiz: Multimodal LMs (reading announced on teams)
- Mid-semester feedback: shared a Google form on teams.

(We know that) Training big models is expensive

Table 1: We developed our models in five groups, based on parameter count and architecture: less than 1 billion, 1 billion, 7 billion, and 13 billion parameters, and our mixture-of-experts model with 1 billion active and 7 billion total parameters. We found that ~70% of our developmental environmental impact came from developing the 7B and 13B models, and the total impact was emissions equivalent to 2.1 tanker trucks' worth of gasoline, and equal to about 7 and a half years of water used by the average person in the United States.

	GPU Hours	Total MWh	# Runs	Carbon Emissions (tCO_2eq)	Equivalent to... (energy usage, 1 home, U.S.)	Water Consumption (kL)	Equivalent to... (water usage, 1 person)
<1B	29k	19	20	6	1 yr, 4 mo	24	3 mo
7B	269k	196	375	65	13 yrs, 6 mo	252	2 yrs, 7 mo
13B	191k	116	156	46	9 yrs, 7 mo	402	3 yrs, 7 mo
MoE	27k	19	35	6	1 yr, 4 mo	24	3 mo
Total	680k	459	813	159	33 yrs, 1 mo	843	7 yrs, 5 mo

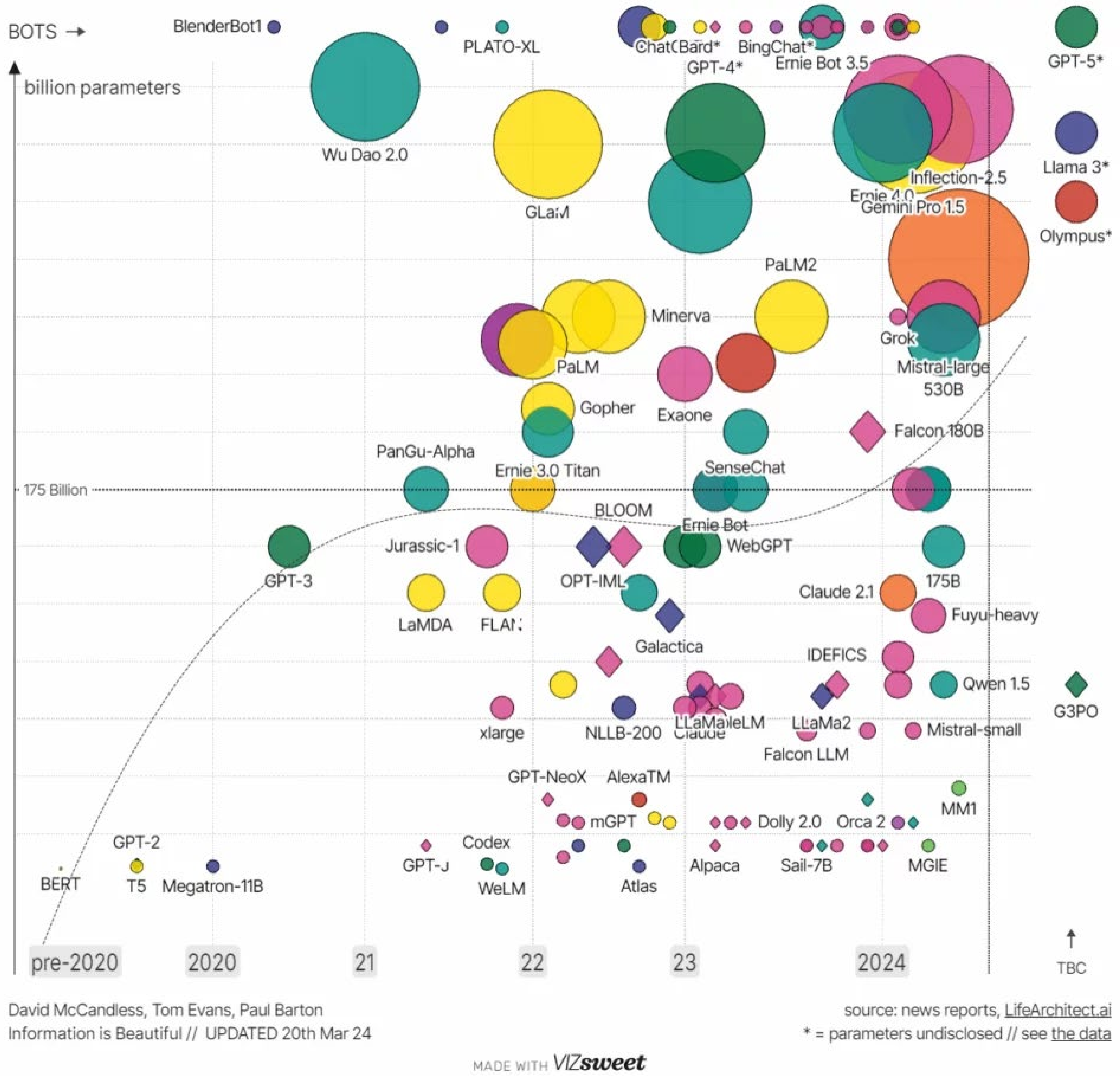
HOLISTICALLY EVALUATING THE ENVIRONMENTALIMPACT OF CREATING LANGUAGE MODELS

But inference is even more expensive

More importantly, inference costs far exceed training costs when deploying a model at any reasonable scale. In fact, the costs to inference ChatGPT exceed the training costs on a weekly basis.

https://www.semianalysis.com/p/the-inference-cost-of-search-disruption

Models aren't getting much smaller



The rise and rise of AI-based Large Language Models (LLMs) like GPT4, LaMDA, LLaMa, PaLM and Jurassic-2.

Today’s Topic

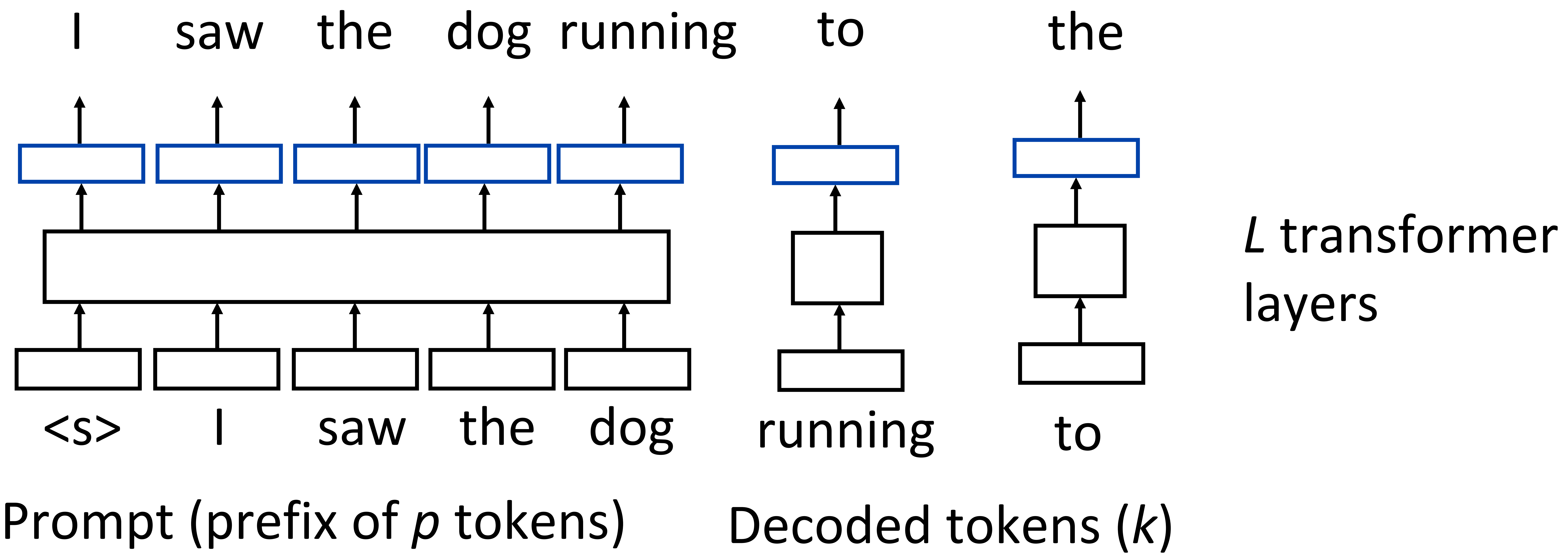
- **How can we cheaply, efficiently, and equitably deploy NLP systems without sacrificing performance?**

This Lecture

- Decoding optimizations: exact decoding, but faster
  - Speculative decoding
  - Medusa heads
  - Flash attention
- Model compression
  - Pruning LLMs
  - Distilling LLMs
- Parameter-efficient tuning
- LLM quantization

Decoding Optimizations

Decoding Basics

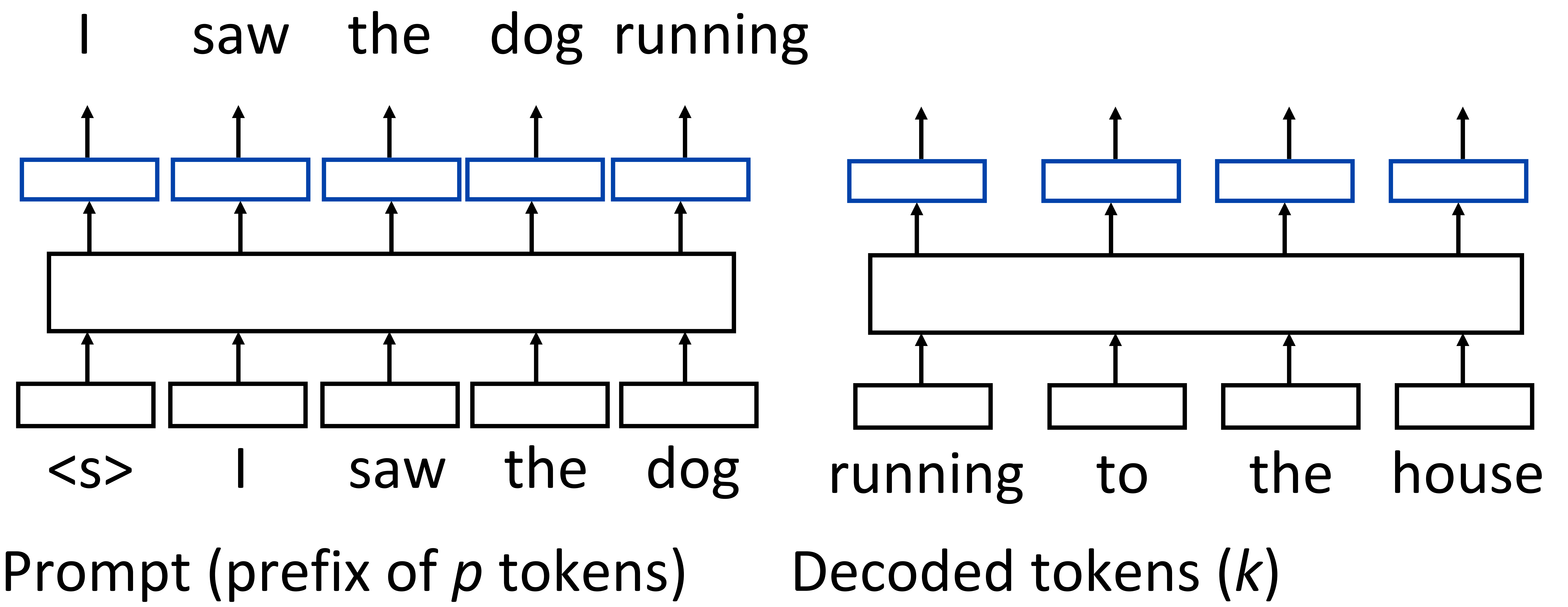


Operations for one decoder pass: $O(pL)$

Operations for k decoder passes: $O(pk^2L)$

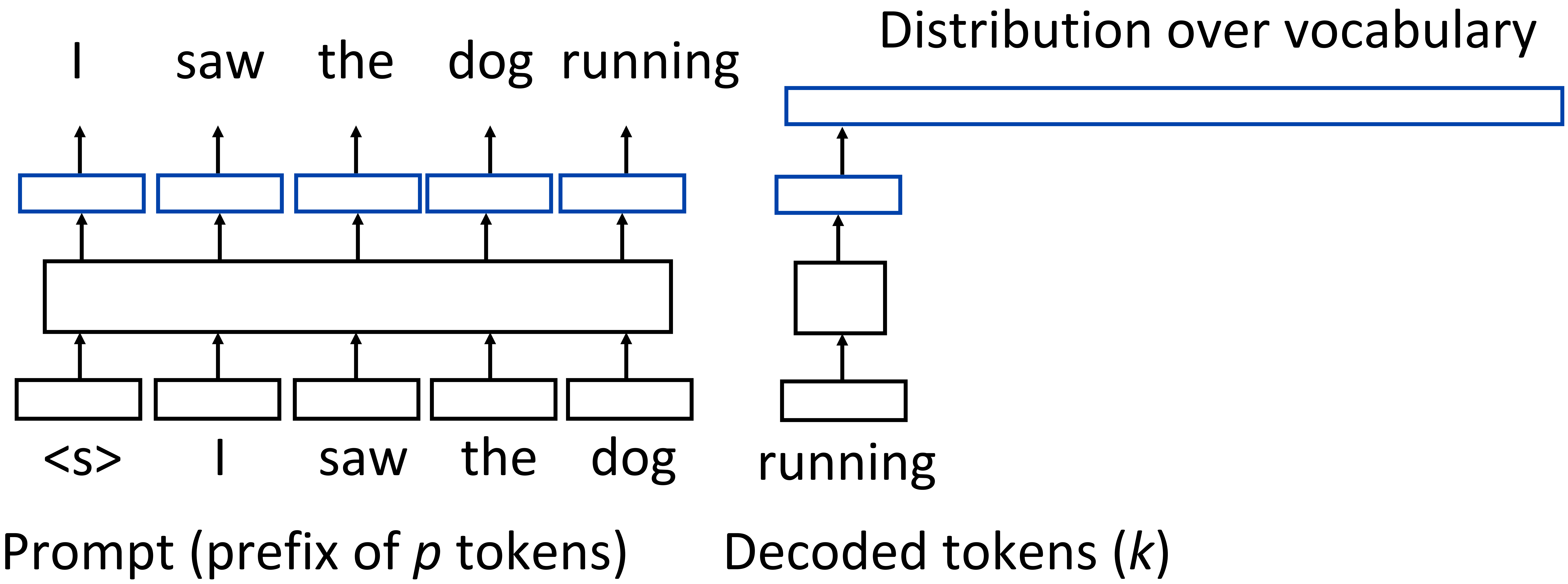
Number of **layers** in decoder (non-parallelizable): $O(kL)$

Speculative Decoding



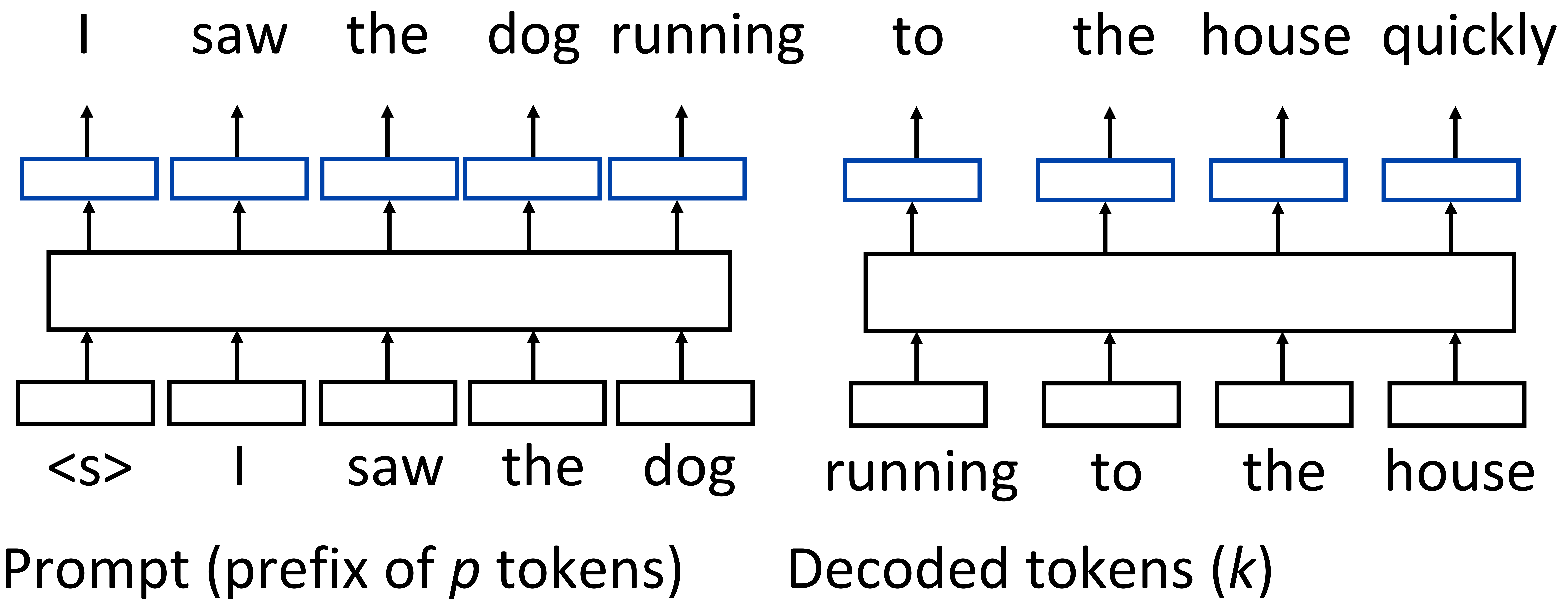
- Key idea a forward pass for several tokens at a time is O(L) serial steps, since the tokens can be computed in parallel
- Can we predict many tokens with a weak model and then "check" them with a single forward pass?

Speculative Decoding



- When sampling, we need the whole distribution
- When doing greedy decoding, we only need to know what token was the max

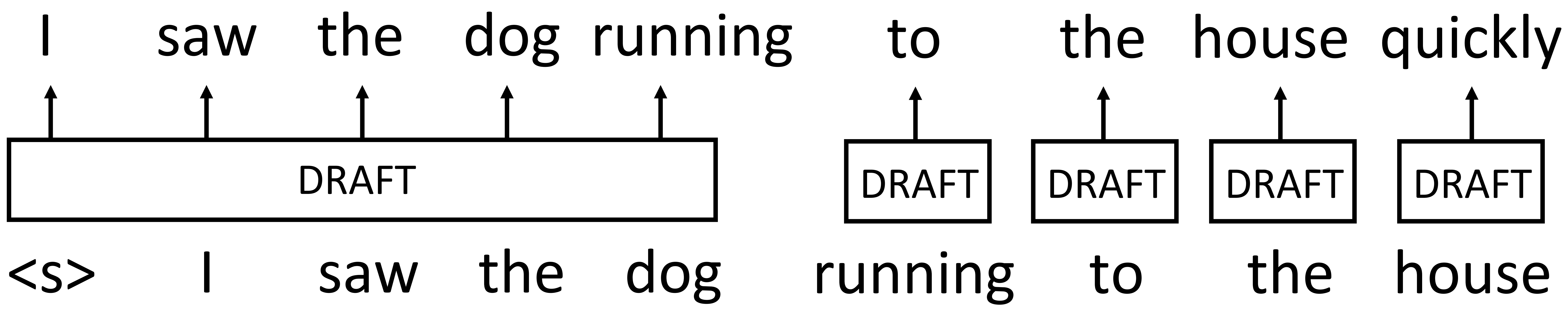
Speculative Decoding



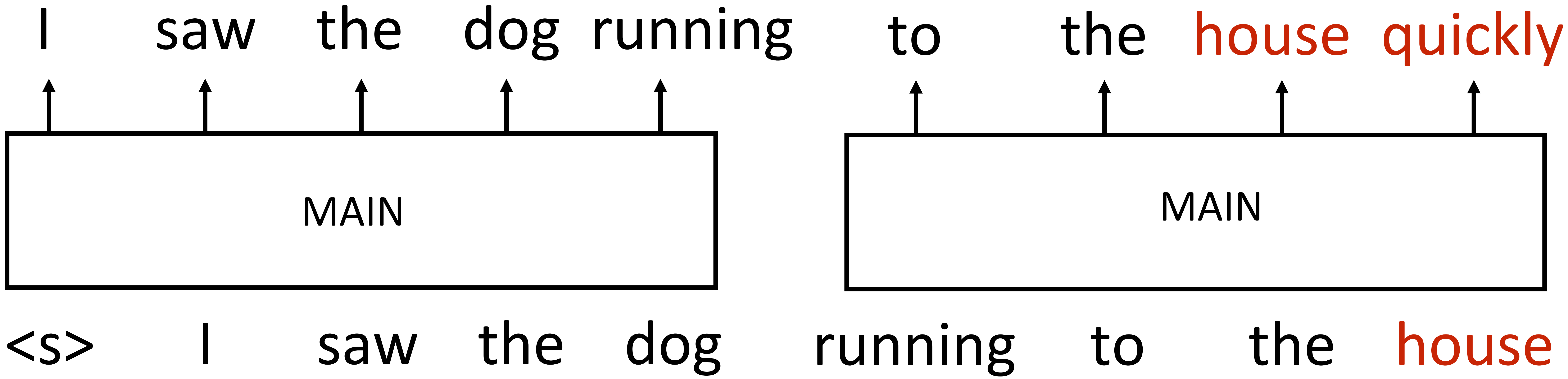
- **We can use a small, cheap model to do inference, then check that “to”, “the”, “house”, “quickly” are really the best tokens from a bigger model**

Leviathan et al. (2023)

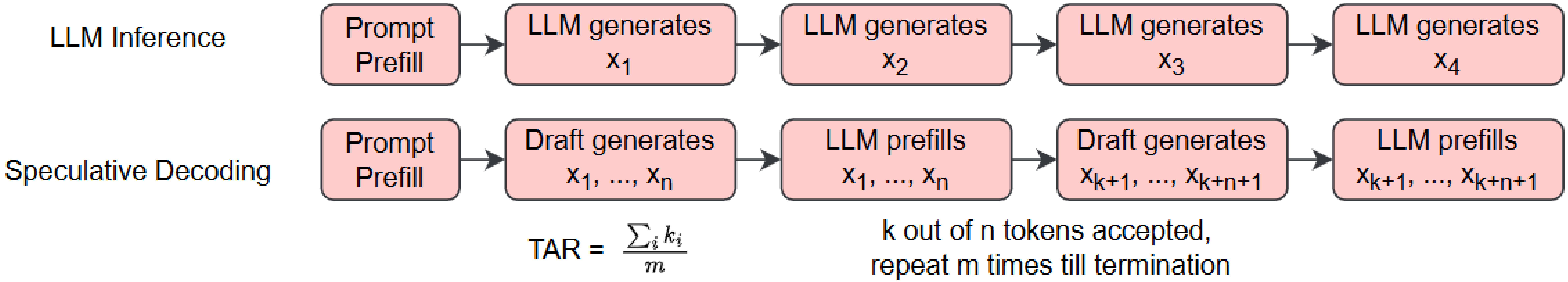
Speculative Decoding: Flow



- Produce decoded tokens one at a time from a fast draft model…



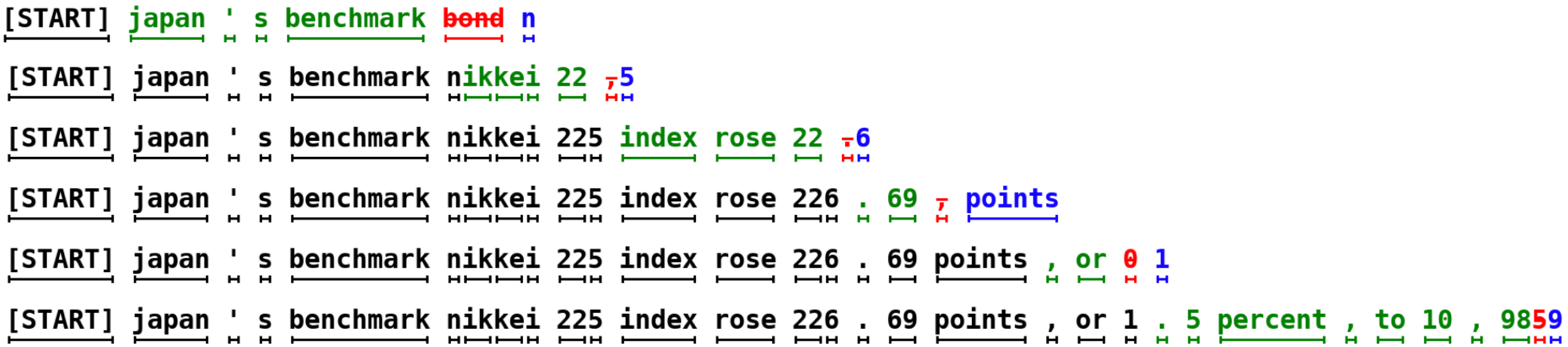
- Confirm that the tokens are the max tokens from the slower main model. Any “wrong” token invalidates the rest of the sequence



LLM Inference
Prompt Prefill
LLM generates x_1
LLM generates x_2
LLM generates x_3
LLM generates x_4
Speculative Decoding
Prompt Prefill
Draft generates $x_1, ..., x_n$
LLM prefills $x_1, ..., x_n$
Draft generates $x_{k+1}, ..., x_{k+n+1}$
LLM prefills $x_{k+1}, ..., x_{k+n+1}$
TAR = $\frac{\sum_i k_i}{m}$
k out of n tokens accepted,
repeat m times till termination

Speculative Decoding

Leviathan et al. (2023)



- Can also adjust this to use sampling. Treat this as a proposal distribution $q(x)$ and may need to reject + resample (rejection sampling)

Speculative Decoding

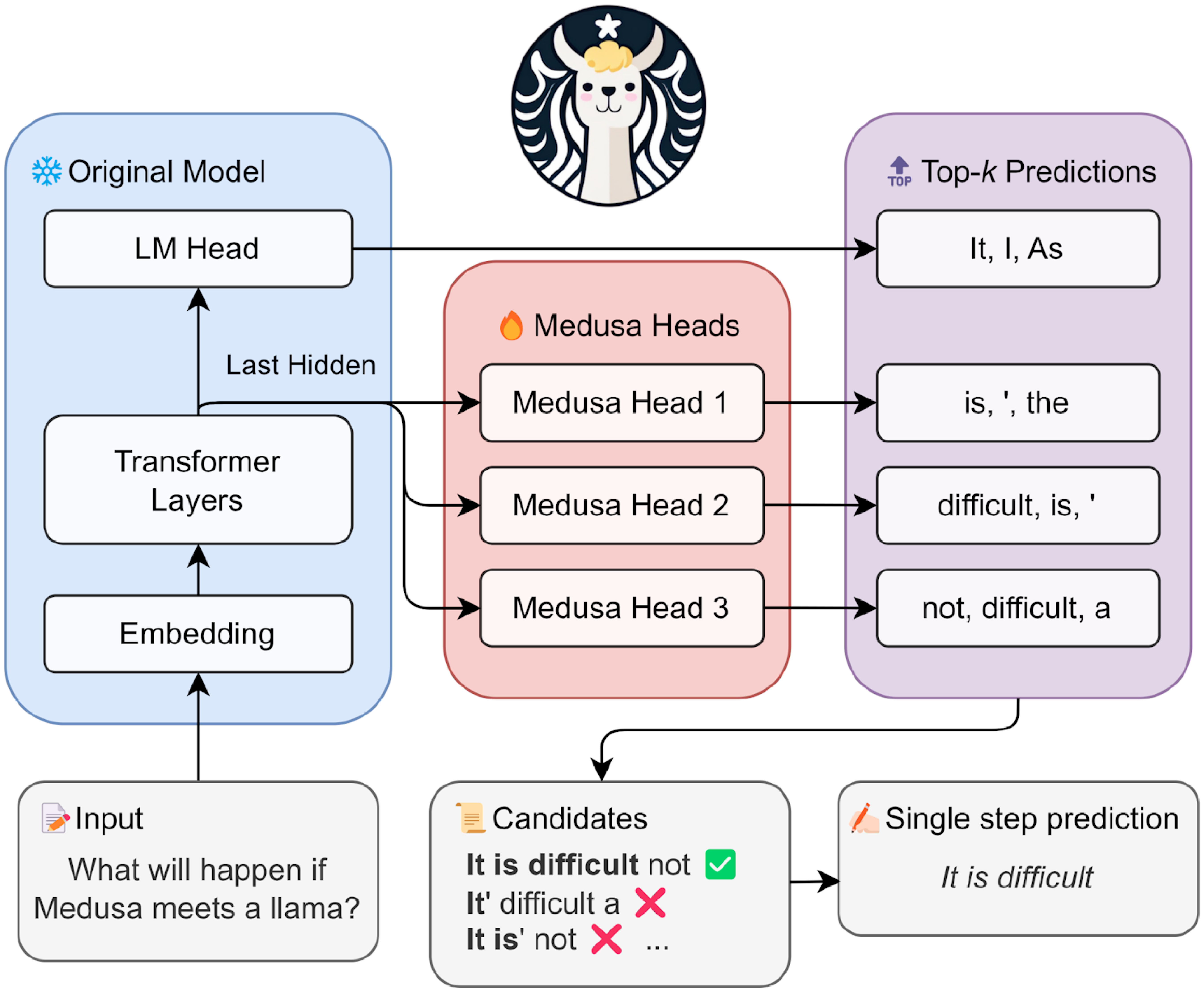
- Find the first index that was rejected by the sampling procedure, then resample from there

Leviathan et al. (2023)

Inputs: $M_p, M_q, prefix$.
$\triangleright$ Sample γ guesses $x_{1,\ldots,\gamma}$ from M_q autoregressively.
for $i = 1$ **to** γ **do**
  $q_i(x) \leftarrow M_q(prefix + [x_1, \ldots, x_{i-1}])$
  $x_i \sim q_i(x)$
end for
$\triangleright$ Run M_p in parallel.
$p_1(x), \ldots, p_{\gamma+1}(x) \leftarrow$
  $M_p(prefix), \ldots, M_p(prefix + [x_1, \ldots, x_\gamma])$
$\triangleright$ Determine the number of accepted guesses n.
$r_1 \sim U(0,1), \ldots, r_\gamma \sim U(0,1)$
$n \leftarrow \min(\{i - 1 \mid 1 \leq i \leq \gamma, r_i > \frac{p_i(x)}{q_i(x)}\} \cup \{\gamma\})$
$\triangleright$ Adjust the distribution from M_p if needed.
$p'(x) \leftarrow p_{n+1}(x)$
if $n < \gamma$ **then**
  $p'(x) \leftarrow norm(max(0, p_{n+1}(x) - q_{n+1}(x)))$
end if
$\triangleright$ Return one token from M_p, and n tokens from M_q.
$t \sim p'(x)$
return $prefix + [x_1, \ldots, x_n, t]$

Medusa Heads

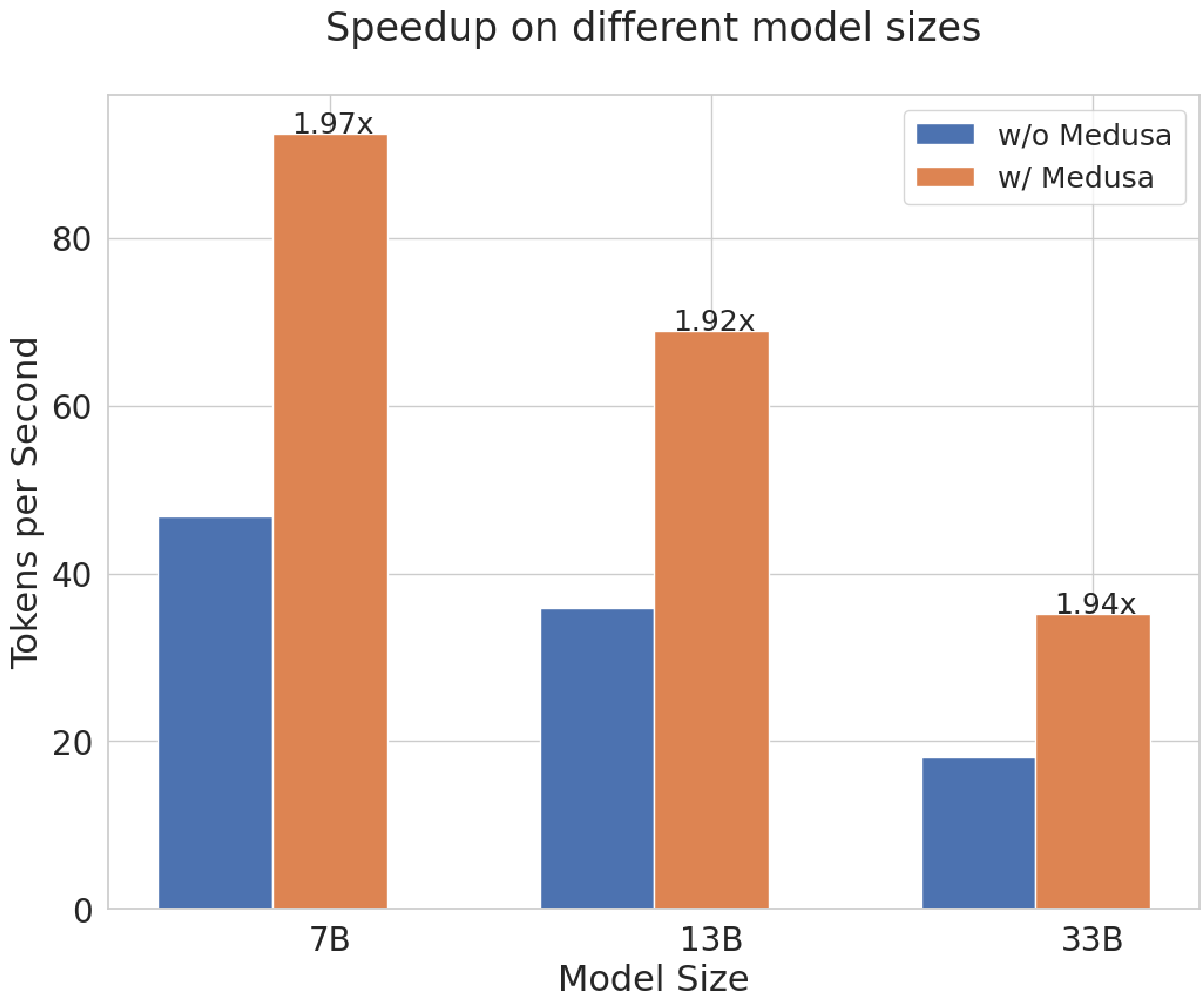
- The “draft model” consists of multiple prediction heads trained to predict the next k tokens



https://www.together.ai/blog/medusa

Medusa Heads

- Speedup with no loss in accuracy!



https://www.together.ai/blog/medusa

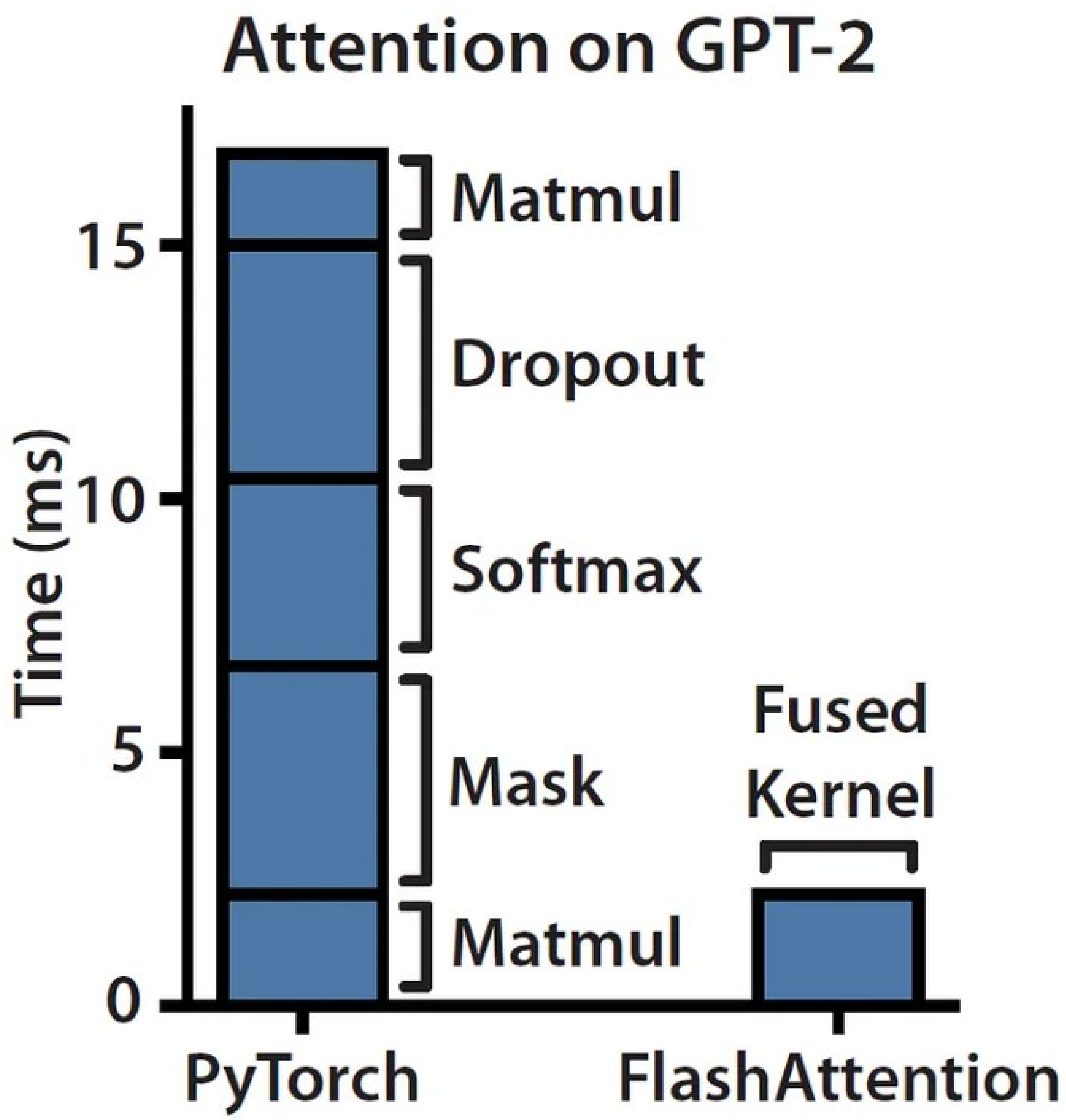
Other Decoding Improvements

- Most other approaches to speeding up require changing the model (making a faster Transformer) or making it smaller (distillation, pruning; discussed next)
- Batching parallelism: improve throughput by decoding many examples in parallel. (Does not help with latency, and it's a little bit harder to do in production if requests are coming in asynchronously)
- Low-level hardware optimizations?
  - Easy things like caching (KV cache: keys + values for context tokens are cached across multiple tokens)

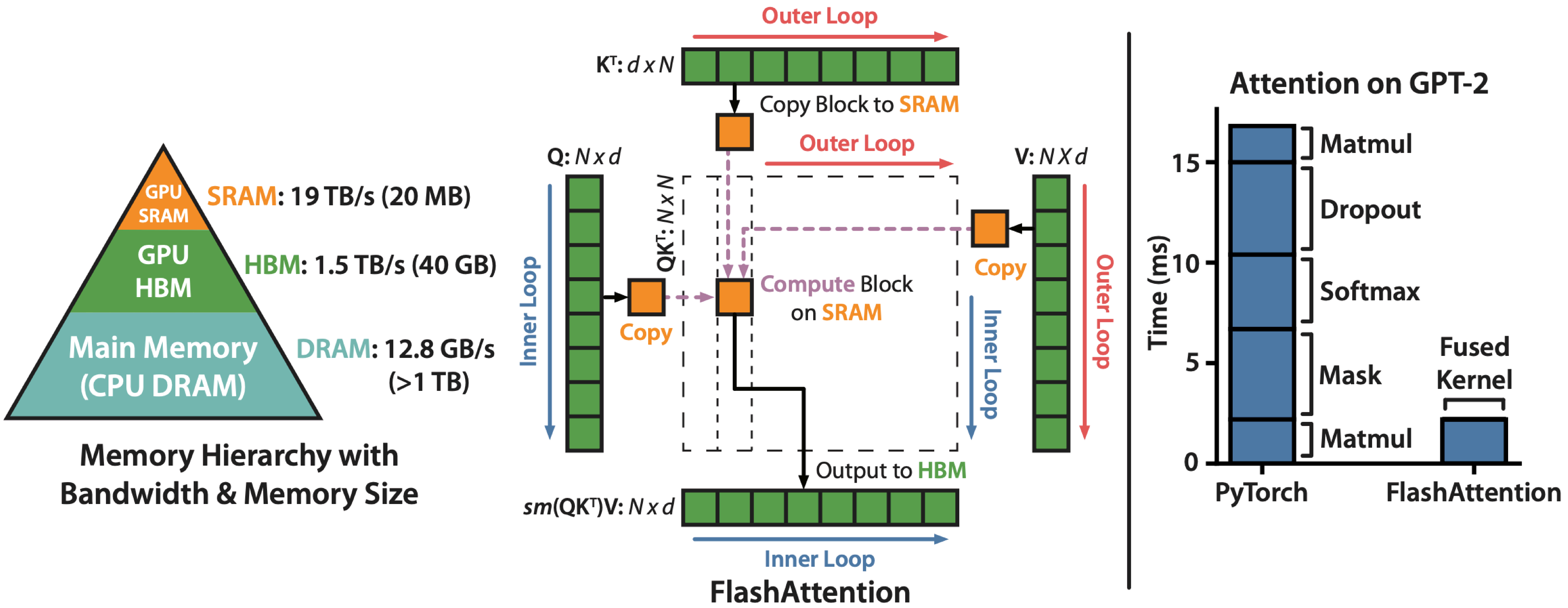
Flash Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Operation	Cost	Bound
$QK^\top$	$\mathcal{O}(nmd_k)$	**Compute-bound**
Scaling $\div\sqrt{d_k}$	$\mathcal{O}(nm)$	Memory-bound
Softmax	$\mathcal{O}(nm)$	Memory-bound
$\text{softmax}(...)V$	$\mathcal{O}(nmd_v)$	**Compute-bound**



Flash Attention



- Does extra computation during attention, but avoids expensive reads/writes to GPU “high-bandwidth memory.” Recomputation is all in SRAM and is very fast
- Essentially: store a running sum for the softmax, compute values as needed

Flash Attention

Models	ListOps	Text	Retrieval	Image	Pathfinder	Avg	Speedup
Transformer	36.0	63.6	81.6	42.3	72.7	59.3	-
FlashAttention	37.6	63.9	81.4	43.5	72.7	59.8	2.4×
Block-sparse FlashAttention	37.0	63.0	81.3	43.6	73.3	59.6	**2.8×**
Linformer [84]	35.6	55.9	77.7	37.8	67.6	54.9	2.5×
Linear Attention [50]	38.8	63.2	80.7	42.6	72.5	59.6	2.3×
Performer [12]	36.8	63.6	82.2	42.1	69.9	58.9	1.8×
Local Attention [80]	36.1	60.2	76.7	40.6	66.6	56.0	1.7×
Reformer [51]	36.5	63.8	78.5	39.6	69.4	57.6	1.3×
Smyrf [19]	36.1	64.1	79.0	39.6	70.5	57.9	1.7×

- Gives a speedup for free — with no cost in accuracy (modulo numeric instability)
- Outperforms the speedup from many other approximate Transformer methods, which perform substantially worse

Model Compression

Model Compression

1. Quantization
   - keep the model the same but reduce the number of bits
2. Pruning
   - remove parts of a model while retaining performance
3. Distillation
   - train a smaller model to imitate the bigger model

Why is this even possible?

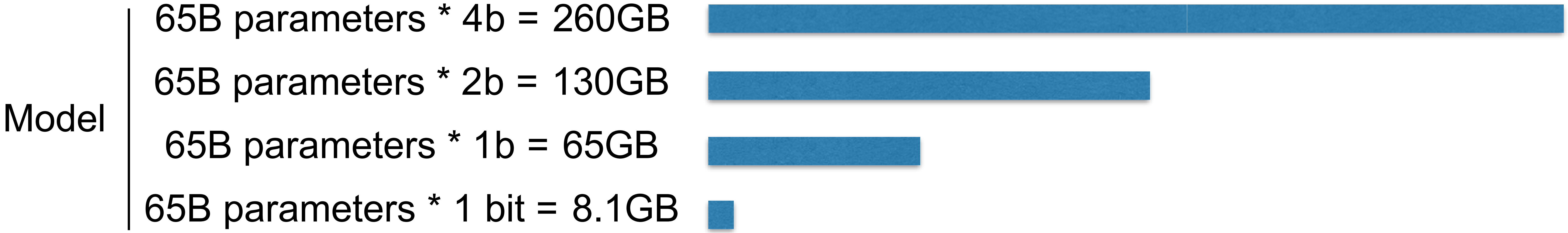
Overparameterized models are easier to optimize (Du and Lee 2018)

networks. For a k hidden node shallow network with quadratic activation and n training data points, we show as long as $k \geq \sqrt{2n}$, over-parametrization enables local search algorithms to find a *globally* optimal solution for general smooth and convex loss functions. Further, de-

Quantization

Post-Training Quantization

- **Example:** Train a 65B-param model with whatever precision you like, then quantize the weights

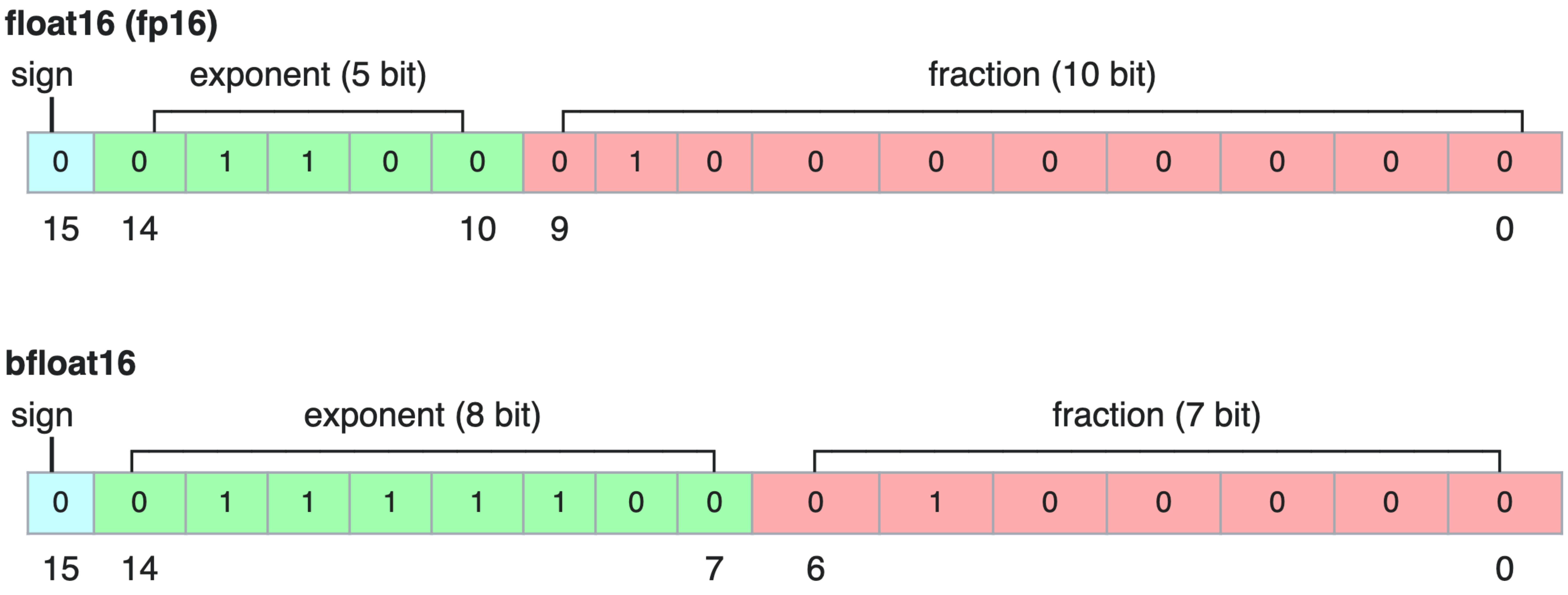


Floating point numbers

Floating point number is stored as $(-1)^s M 2^E$

- Sign bit *s*
- Fractional part *M* = frac
- Exponential part *E* = exp - bias

Reduced-precision floating point types



Int8 quantization

- Absolute Maximum (absmax) quantization:

$$\mathbf{X}_{i8} = \left\lfloor \frac{127 \cdot \mathbf{X}_{f16}}{\max_{ij}(|\mathbf{X}_{f16_{ij}}|)} \right\rceil$$

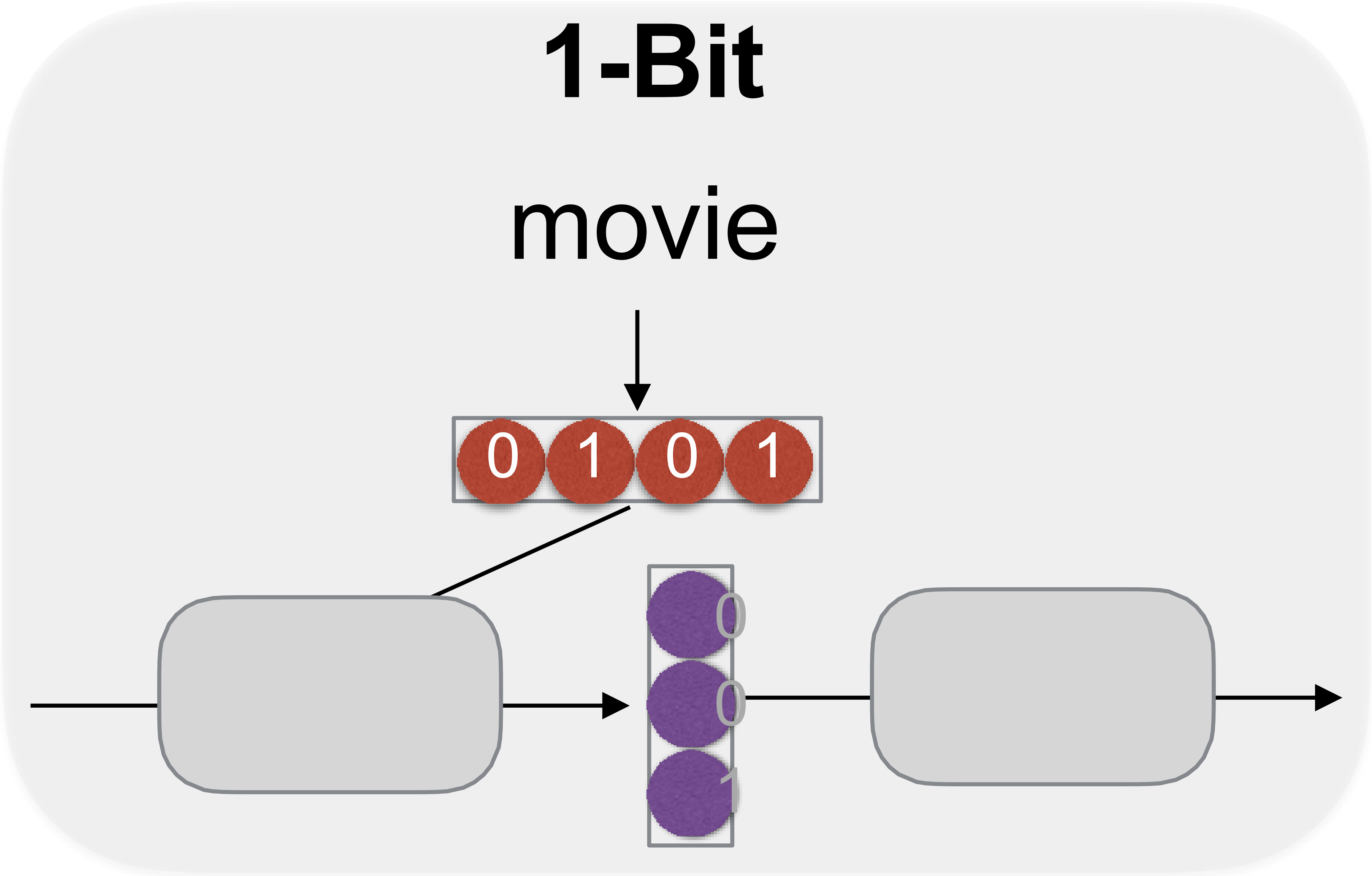
- This scales inputs to [-127, 127]

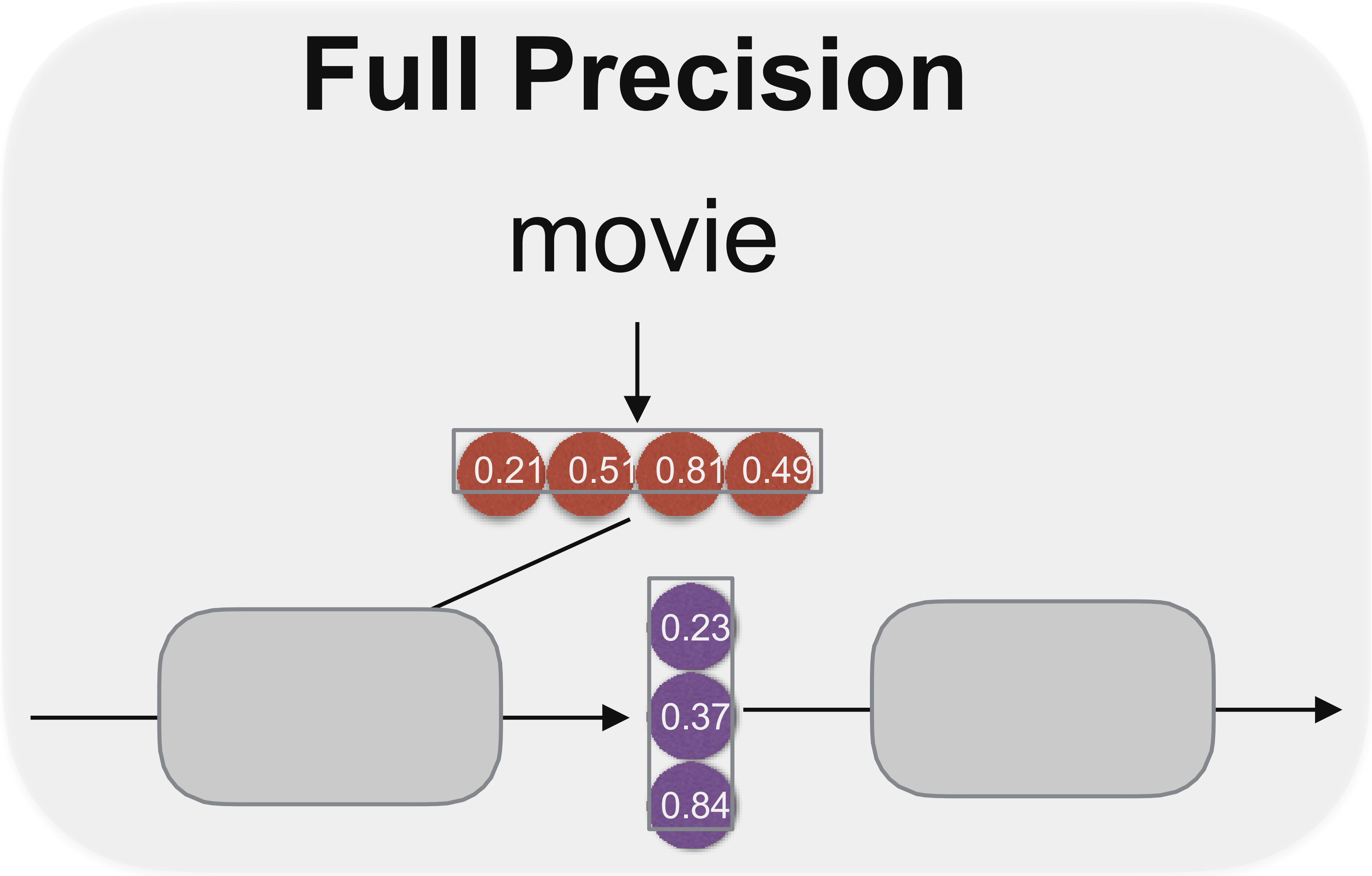
[0.5, 20, -0.0001, -.01, -0.1]

- Maximum entry is 20

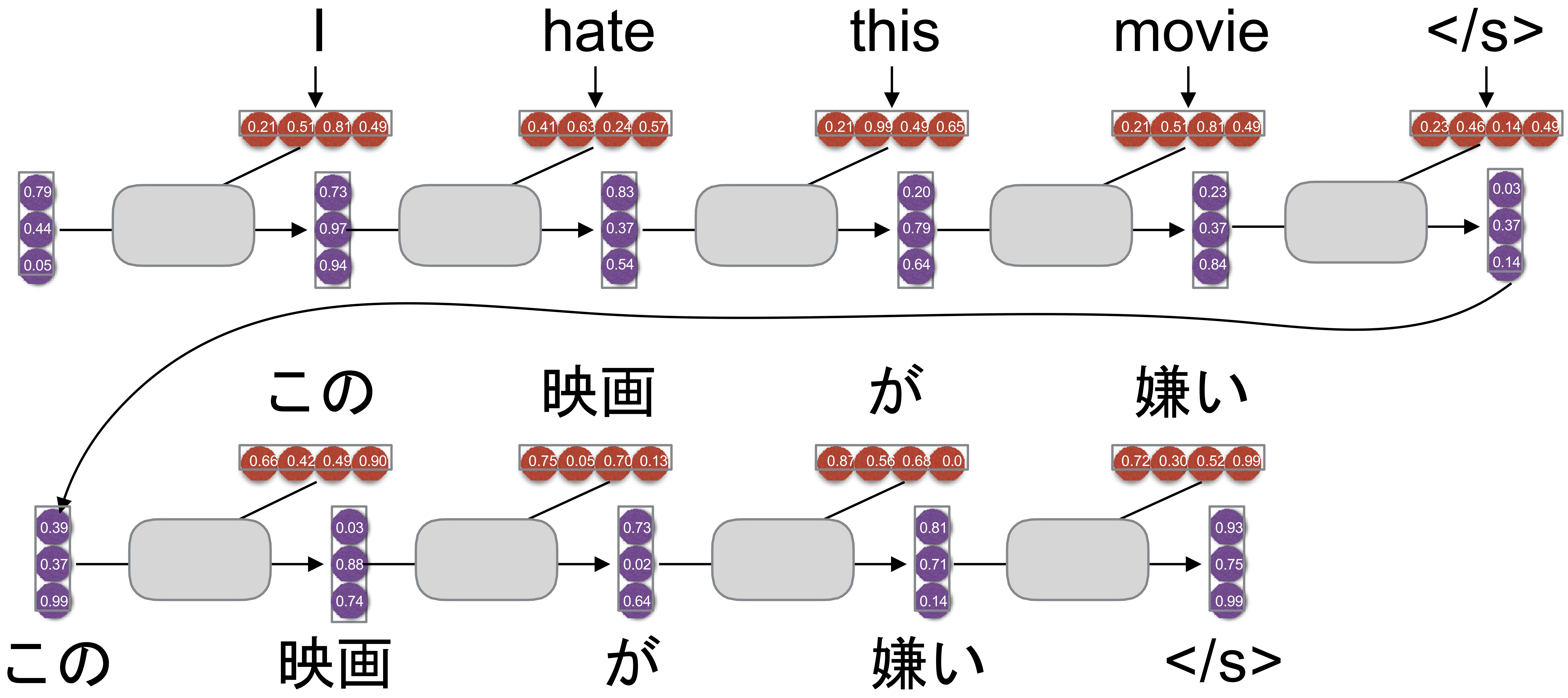
- round(127/20 * [0.5, 20, -0.0001, -.01, -0.1]) -> [3, 127, 0, 0, -1]

Extreme Example: Binarized Neural Networks

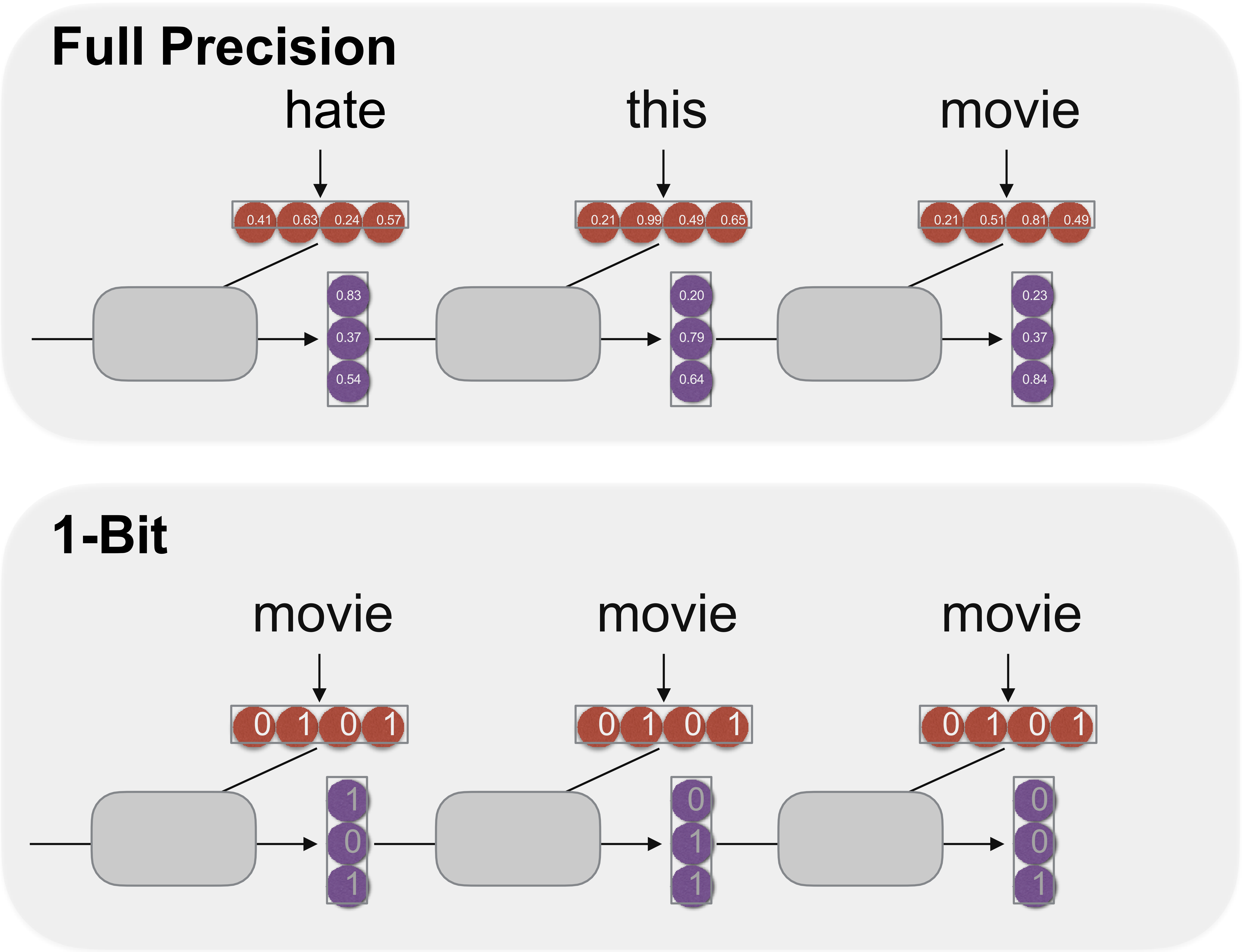




Extreme Example: Binarized Neural Networks



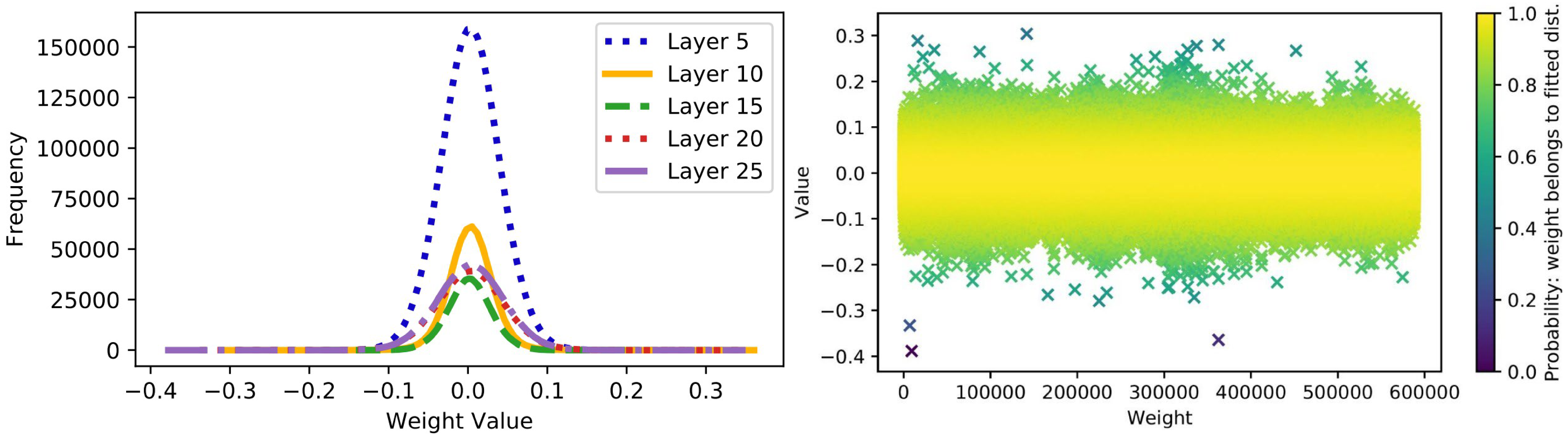
Extreme Example: Binarized Neural Networks



Model-Aware Quantization: GOBO

(Zadeh et al. 2020)

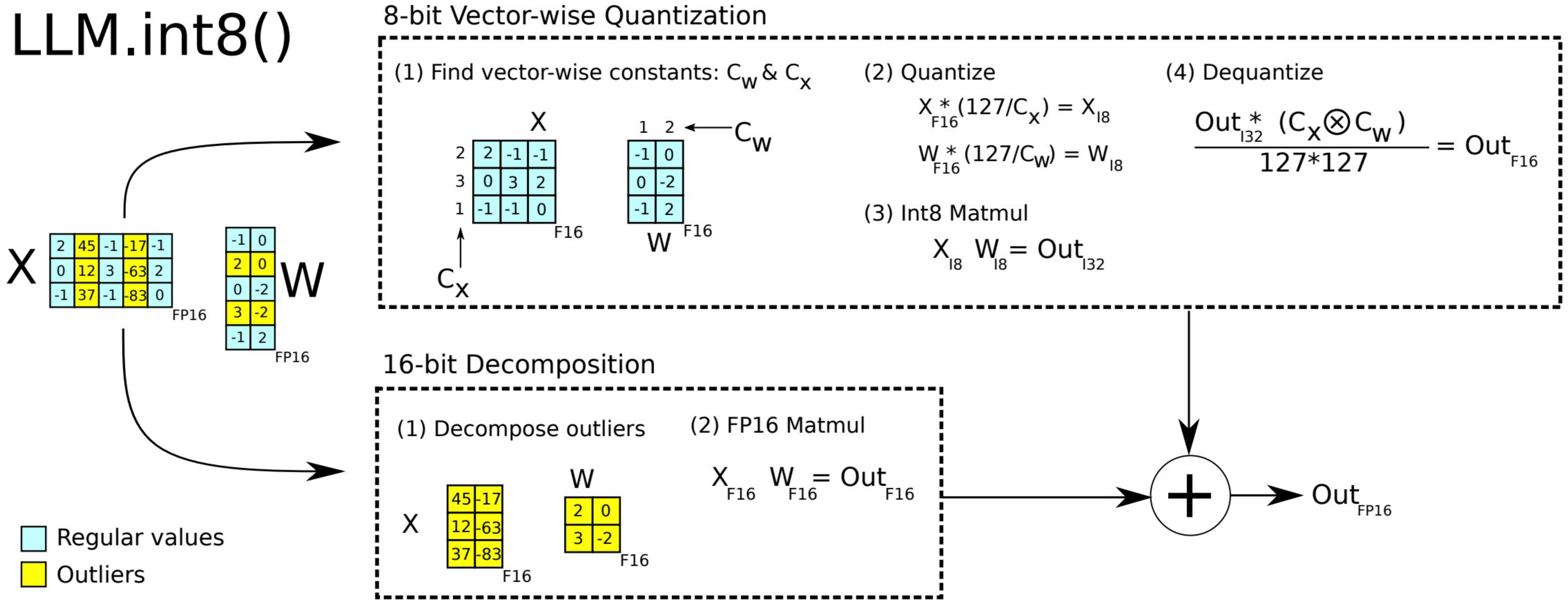
- BERT weights in each layer tend to lie on a Gaussian
  - Only small fraction of weights in each layer are in the tails of the distribution



- Quantize the 99.9% of weights in the body of the disribution into 8 buckets
  - Do not quantize the remaining 0.01%

Model-Aware Quantization: LLM.int8
(Dettmers et al. 2022)

- Problem with prev approach: quantizing each layer uniformly
- 95% of params in Transformer LLMs are matrix multiplication



- Quantization overhead slowns down <6.7B models, but enables inference of 175B models on single GPUs (in half the time)

Hardware Concerns
(Shen et al. 2019)

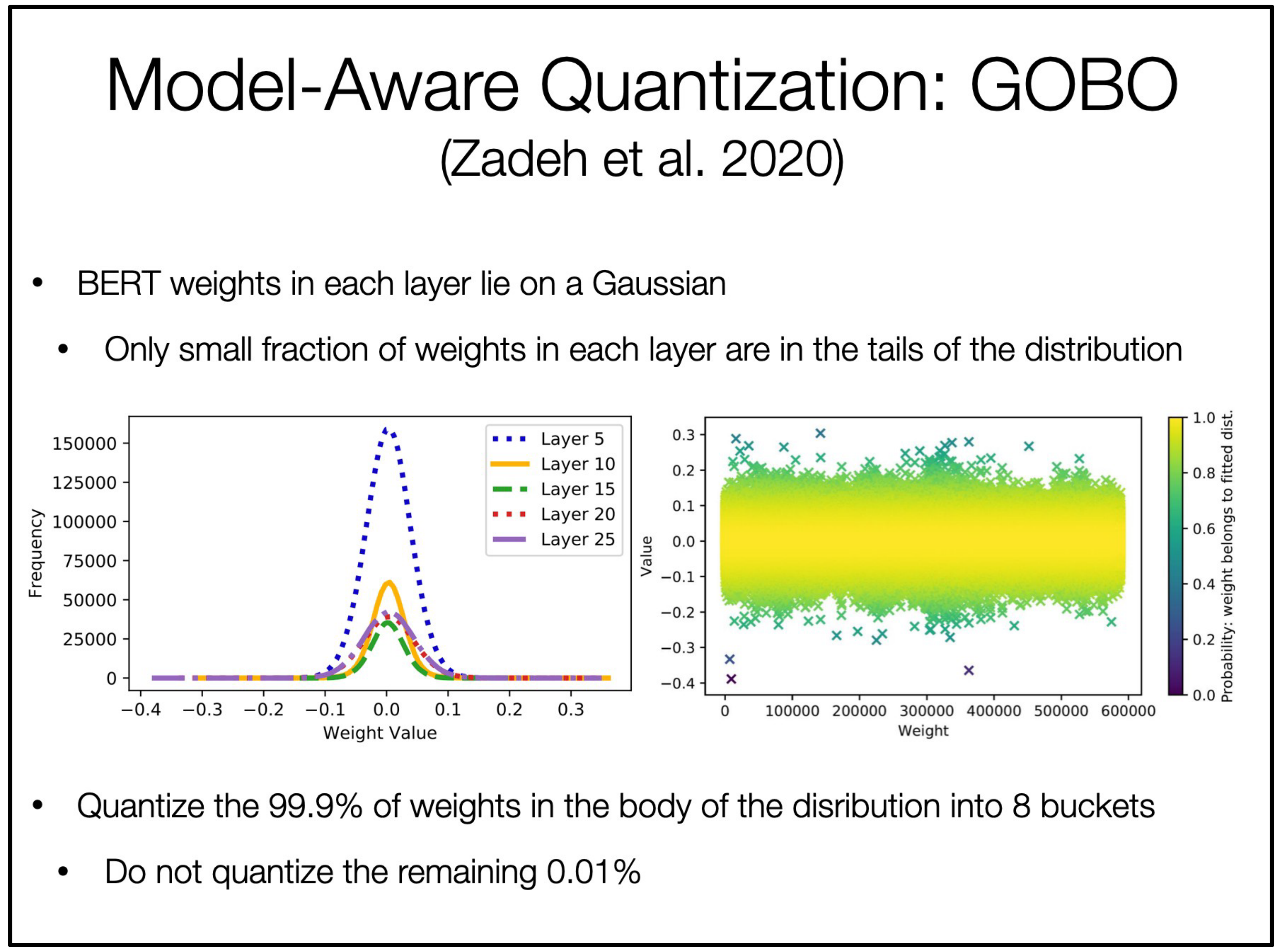
- Not all data types (e.g. "Int3") are supported by most hardware
- PyTorch only supports certain data types (e.g. no support for Int4)

PyTorch Docs > Quantization

	Static Quantization	Dynamic Quantization
nn.Linear	Y	Y
nn.Conv1d/2d/3d	Y	N
nn.LSTM	Y (through custom modules)	Y
nn.GRU	N	Y
nn.RNNCell	N	Y
nn.GRUCell	N	Y
nn.LSTMCell	N	Y
nn.EmbeddingBag	Y (activations are in fp32)	Y
nn.Embedding	Y	Y
nn.MultiheadAttention	Y (through custom modules)	Not supported
Activations	Broadly supported	Un-changed, computations stay in fp32

Hardware Concerns
(Shen et al. 2019)

- Not all data types (e.g. “Int3”) are supported by most hardware
- PyTorch only supports certain data types (e.g. no support for Int4)
- Some quantization methods require writing bespoke hardware accelerators



Quantization-Aware Training

Binarized Neural Networks

(Courbariaux et al. 2016)

- Weights are -1 or 1 everywhere
- Activations are also binary
- Defined stochastically: choose 0 with probability σ(x) and 1 with probability 1 - σ(x)
- Backprop is also discretized

Binarized Neural Networks

(Courbariaux et al. 2016)

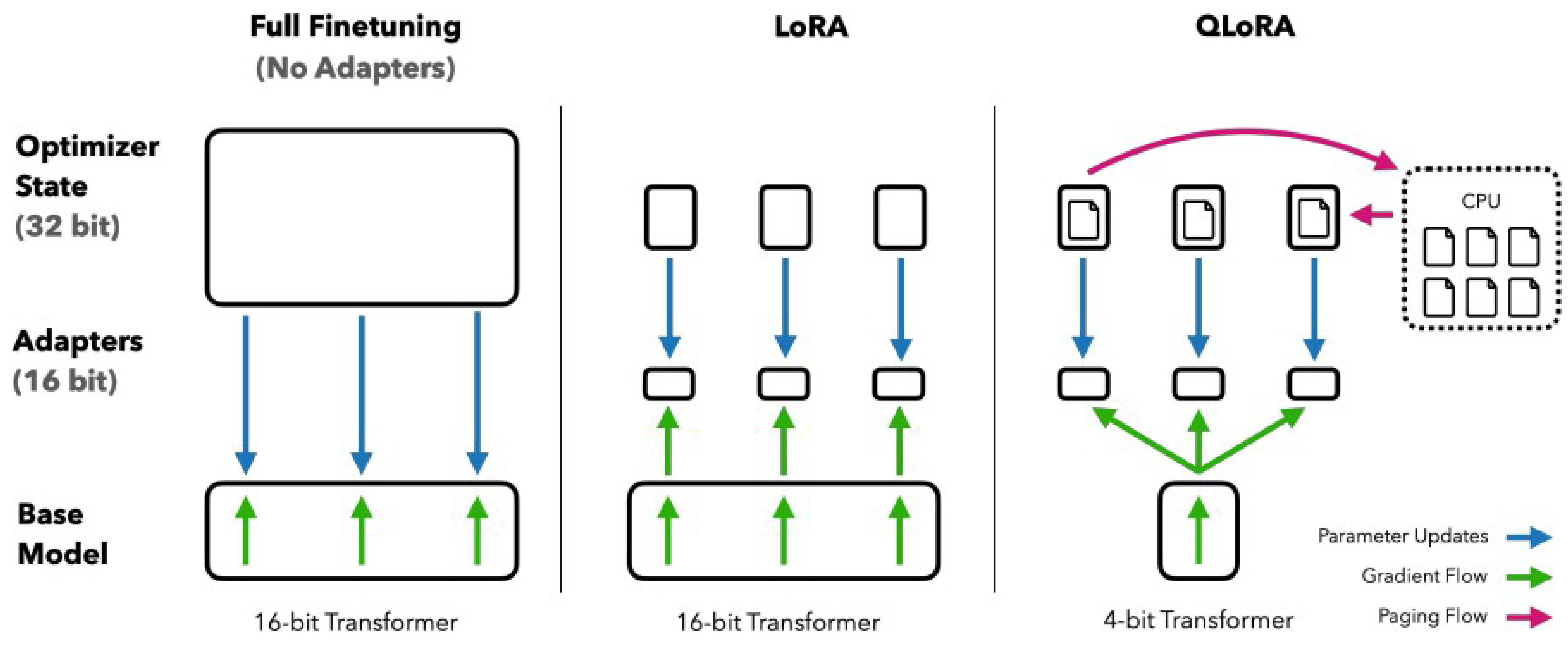
Data set	MNIST	SVHN	CIFAR-10
Binarized activations+weights, during training and test			
BNN (Torch7)	1.40%	2.53%	10.15%
BNN (Theano)	0.96%	2.80%	11.40%
Committee Machines' Array (Baldassi et al., 2015)	1.35%	-	-
Binarized weights, during training and test			
BinaryConnect (Courbariaux et al., 2015)	1.29± 0.08%	2.30%	9.90%
Binarized activations+weights, during test			
EBP (Cheng et al., 2015)	2.2± 0.1%	-	-
Bitwise DNNs (Kim & Smaragdis, 2016)	1.33%	-	-
No binarization (standard results)			
Maxout Networks (Goodfellow et al.)	0.94%	2.47%	11.68%
Network in Network (Lin et al.)	-	2.35%	10.41%
Gated pooling (Lee et al., 2015)	-	1.69%	7.62%

Q-LORA

(Dettmers et al. 2023)

Further compress memory requirements for training by

- 4-bit quantization of the model (please see the class on LoRA) Use of
  - GPU memory paging to prevent OOM



- Can train a 65B model on a 48GB GPU!

Pruning

Pruning

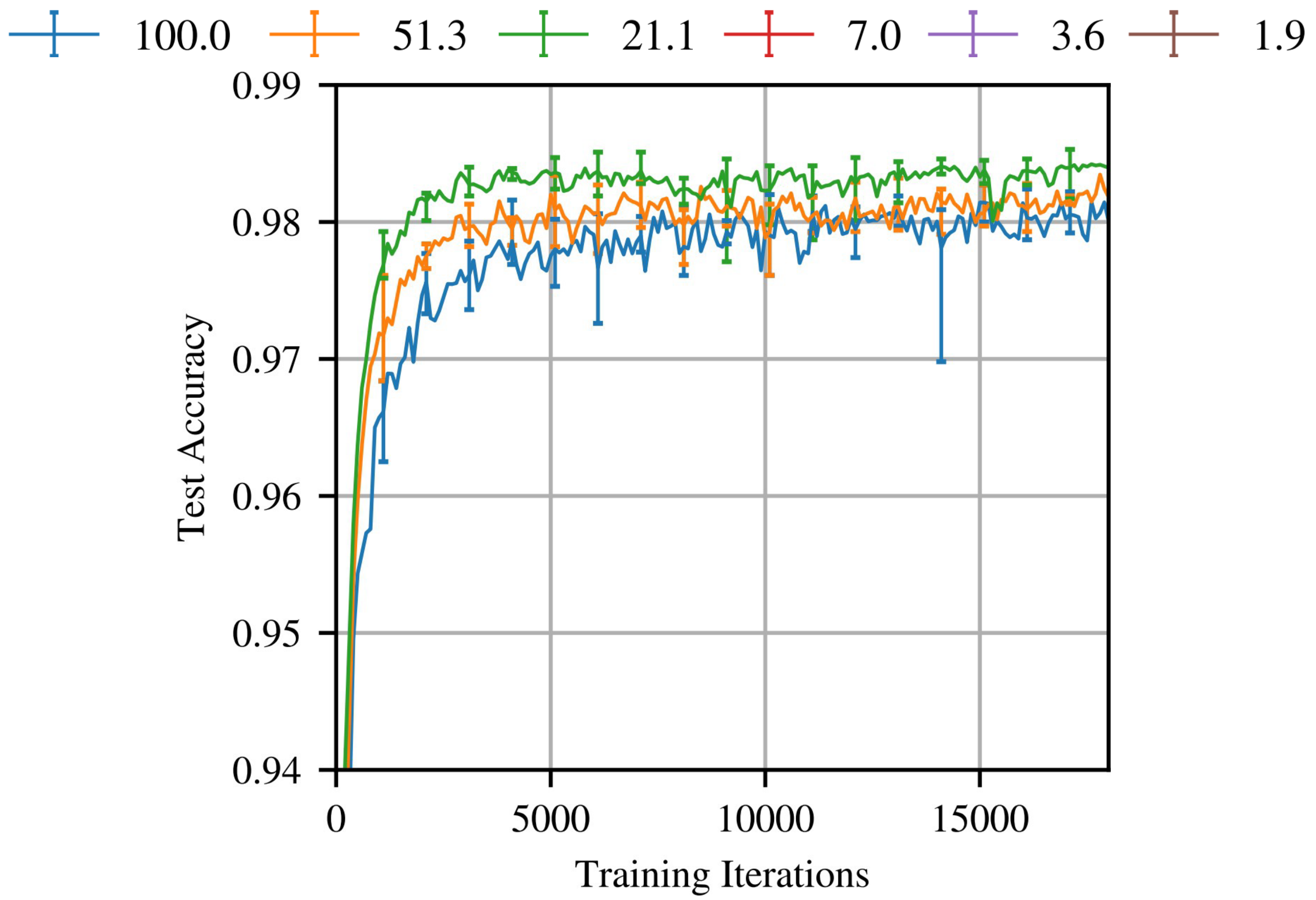
- Remove parameters from the model after training

Pruning vs Quantization

- **Quantization:** no parameters are changed*, up to *k bits of precision*
- **Pruning:** a number of parameters are set to zero, the rest are unchanged

Lottery Ticket Hypothesis

Within a randomly initialized dense neural network, there exists a small subnetwork (a "winning ticket") that, when trained in isolation with the same initialization, can match or even outperform the original network.



Model Compression

Approaches to Compression

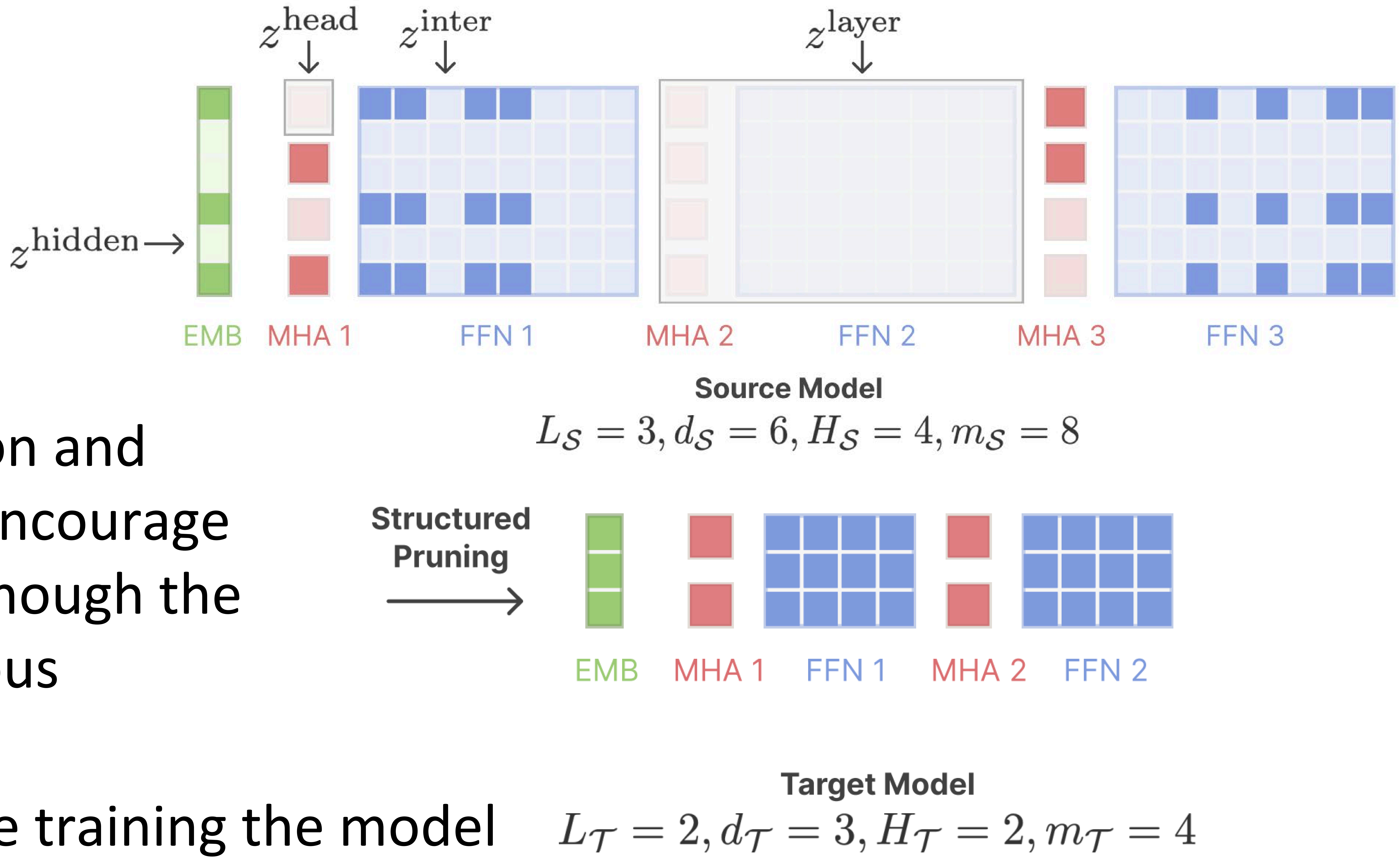
- Pruning: can we reduce the number of neurons in the model?
  - Basic idea: remove low-magnitude weights
  - Issue: sparse matrices are not fast, matrix multiplication is very fast on GPUs so you don't save any time!

Approaches to Compression

- Pruning: can we reduce the number of neurons in the model?
  - ~~Basic idea: remove low-magnitude weights~~
  - Instead, we want some kind of structured pruning. What does this look like?
- Still a challenge: if different layers have different sizes, your GPU utilization may go down

Sheared Llama

- Idea 1: targeted structured pruning



- Parameterization and regularization encourage sparsity, even though the z's are continuous

- Idea 2: continue training the model in its pruned state

Mengzhou Xia et al. (2023)

Sheared Llama

Model (#tokens for training)	Continued		LM	World Knowledge		Average
	LogiQA	BoolQ (32)	LAMBADA	NQ (32)	MMLU (5)	
LLaMA2-7B (2T)†	30.7	82.1	28.8	73.9	46.6	64.6
OPT-1.3B (300B)†	**26.9**	57.5	58.0	6.9	24.7	48.2
Pythia-1.4B (300B)†	27.3	57.4	**61.6**	6.2	**25.7**	48.9
Sheared-LLaMA-1.3B (50B)	**26.9**	**64.0**	61.0	**9.6**	**25.7**	**51.0**
OPT-2.7B (300B)†	26.0	63.4	63.6	10.1	25.9	51.4
Pythia-2.8B (300B)†	28.0	66.0	64.7	9.0	26.9	52.5
INCITE-Base-3B (800B)	27.7	65.9	65.3	14.9	**27.0**	54.7
Open-LLaMA-3B-v1 (1T)	28.4	70.0	65.4	**18.6**	**27.0**	55.1
Open-LLaMA-3B-v2 (1T)†	28.1	69.6	66.5	17.1	26.9	55.7
Sheared-LLaMA-2.7B (50B)	**28.9**	**73.7**	**68.4**	16.5	26.4	**56.7**

- (Slightly) better than models that were "organically" trained at these larger scales

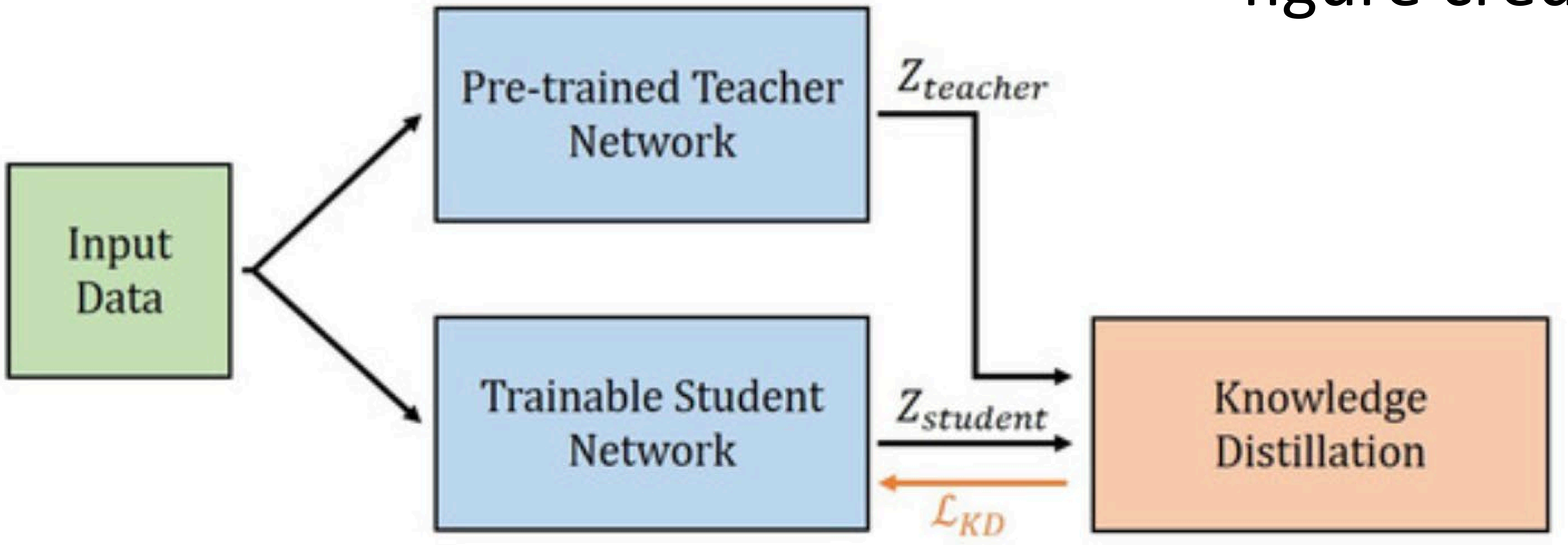
Mengzhou Xia et al. (2023)

Approaches to Compression

- Pruning: can we reduce the number of neurons in the model?
  - ~~Basic idea: remove low-magnitude weights~~
  - Instead, we want some kind of structured pruning. What does this look like?
- Knowledge distillation
  - Classic approach from Hinton et al.: train a *student* model to match distribution from *teacher*

DistilBERT

figure credit: Tianjian Li



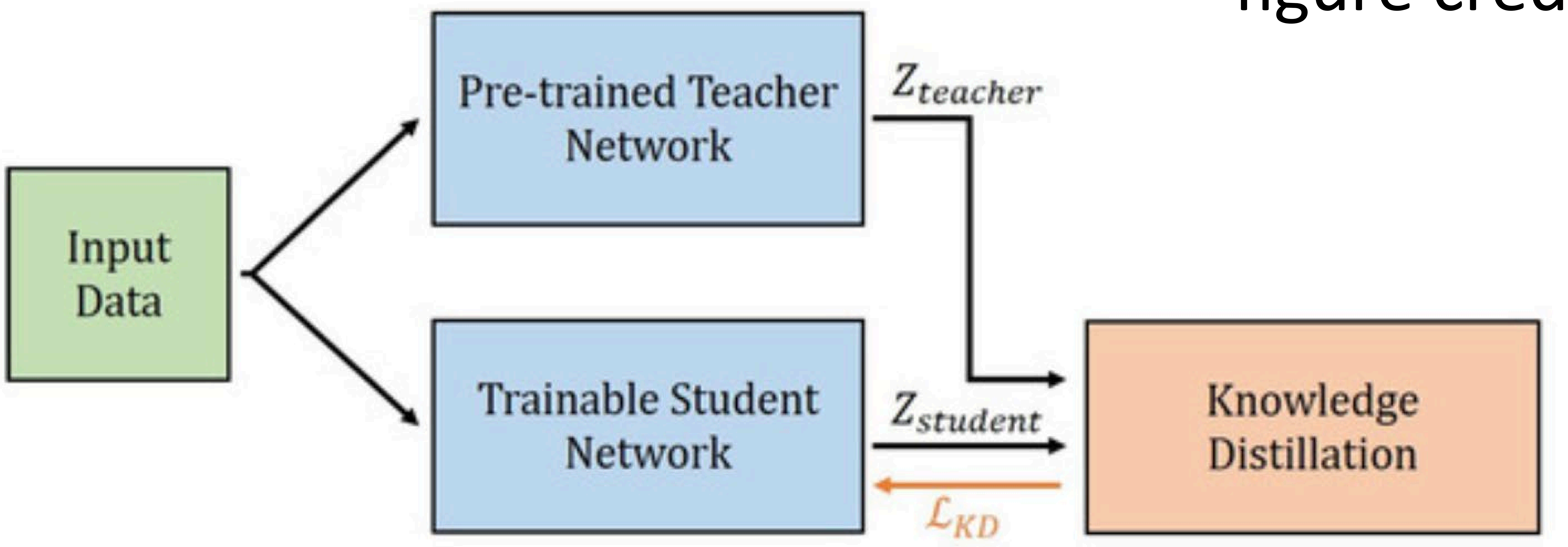
Suppose we have a classification model with output $P_{teacher}(y \mid \boldsymbol{x})$

Minimize KL($P_{teacher}$ || $P_{student}$) to bring student dist close to teacher

Note that this is not using labels — it uses the teacher to "pseudo-label" data, and we label an entire distribution, not just a top-one label

DistilBERT

figure credit: Tianjian Li



- Use a teacher model as a large neural network, such as BERT
- Make a small student model that is half the layers of BERT. Initialize with every other layer from the teacher

Sanh et al. (2019)

DistilBERT

Model	**Score**	CoLA	MNLI	MRPC	QNLI	QQP	RTE	SST-2	STS-B	WNLI
ELMo	68.7	44.1	68.6	76.6	71.1	86.2	53.4	91.5	70.4	56.3
BERT-base	79.5	56.3	86.7	88.6	91.8	89.6	69.3	92.7	89.0	53.5
DistilBERT	77.0	51.3	82.2	87.5	89.2	88.5	59.9	91.3	86.9	56.3

Table 2: **DistilBERT yields to comparable performance on downstream tasks.** Comparison on downstream tasks: IMDb (test accuracy) and SQuAD 1.1 (EM/F1 on dev set). D: with a second step of distillation during fine-tuning.

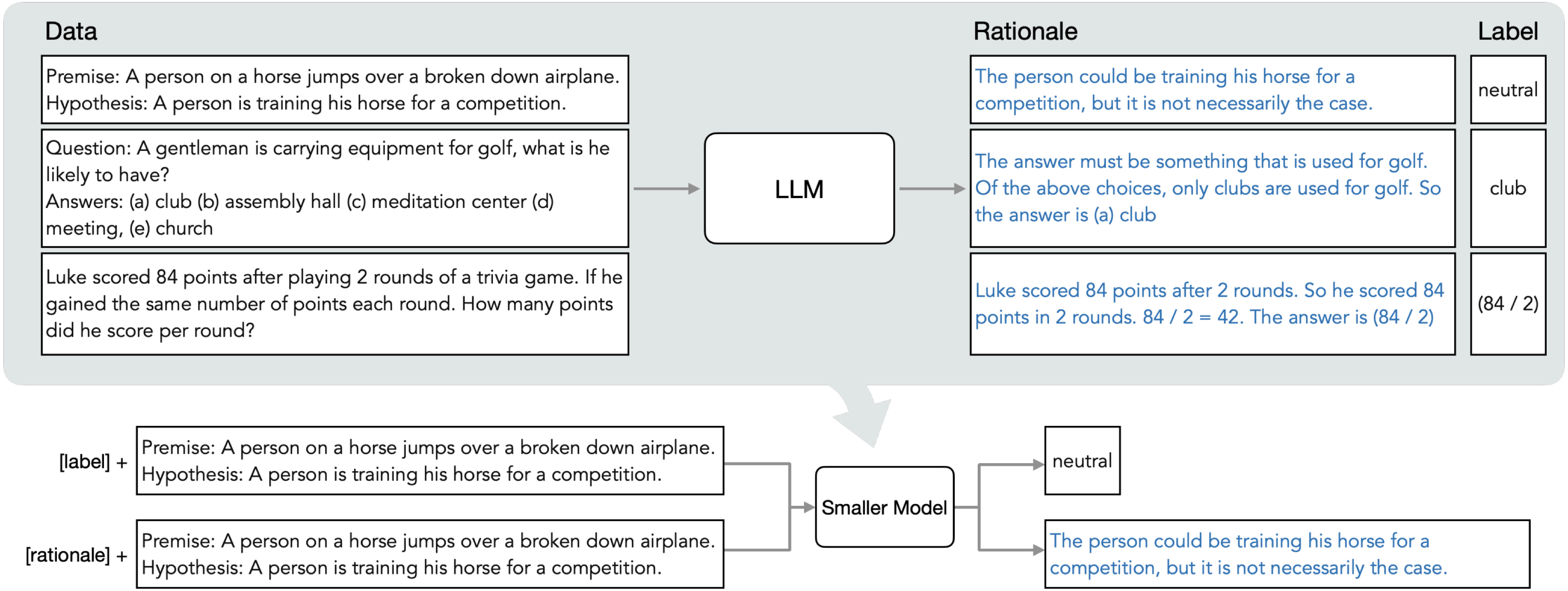
Model	IMDb (acc.)	SQuAD (EM/F1)
BERT-base	93.46	81.2/88.5
DistilBERT	92.82	77.7/85.8
DistilBERT (D)	-	79.1/86.9

Table 3: **DistilBERT is significantly smaller while being constantly faster.** Inference time of a full pass of GLUE task STS-B (sentiment analysis) on CPU with a batch size of 1.

Model	# param. (Millions)	Inf. time (seconds)
ELMo	180	895
BERT-base	110	668
DistilBERT	66	410

Sanh et al. (2019)

Other Distillation



- How to distill models for complex reasoning settings? Still an open problem!

Cheng-Yu Hsieh et al. (2023)

Where is this going?

- **Better GPU programming:** as GPU performance starts to saturate, we'll probably see more algorithms tailored very specifically to the affordances of the hardware
- **Small models**, either distilled or trained from scratch: as LLMs gets better, we can do with ~7B scale what used to be only doable with ChatGPT (GPT-3.5)
- **Continued focus on faster inference**: faster inference can be highly impactful across all LLM applications

Takeaways

- Decoding optimizations: speculative decoding gives a fast way to exactly sample from a smaller model. Also techniques like Flash Attention

- Model compression and quantization: standard compression techniques, but adapted to work really well for GPUs

- Model optimizations to make models smaller: pruning, distillation