CS11-711 Advanced NLP

Quantization, Pruning, and Distillation

Vijay Viswanathan and Graham Neubig

Carnegie Mellon University
Language Technologies Institute

Site
https://phontron.com/class/anlp2024/

NLP systems are now deployed at scale

OpenAI's ChatGPT now has 100 million weekly active users

Aisha Malik @aiishamalik1 / 1:49 PM EST • November 6, 2023 Commer

Article: [TechCrunch (2023)]

We know that training big models is expensive

		Time (GPU hours)	Power Consumption (W)	Carbon Emitted (tCO_2eq)
Llama 2	7B	184320	400	31.22
	13B	368640	400	62.44
	34B	1038336	350	153.90
	70B	1720320	400	291.42
Total		3311616		539.00

Table 2: CO_2 emissions during pretraining. Time: total GPU time required for training each model

Llama 2: Open Foundation and Fine-Tuned Chat Models. Touvron et al. 2023.

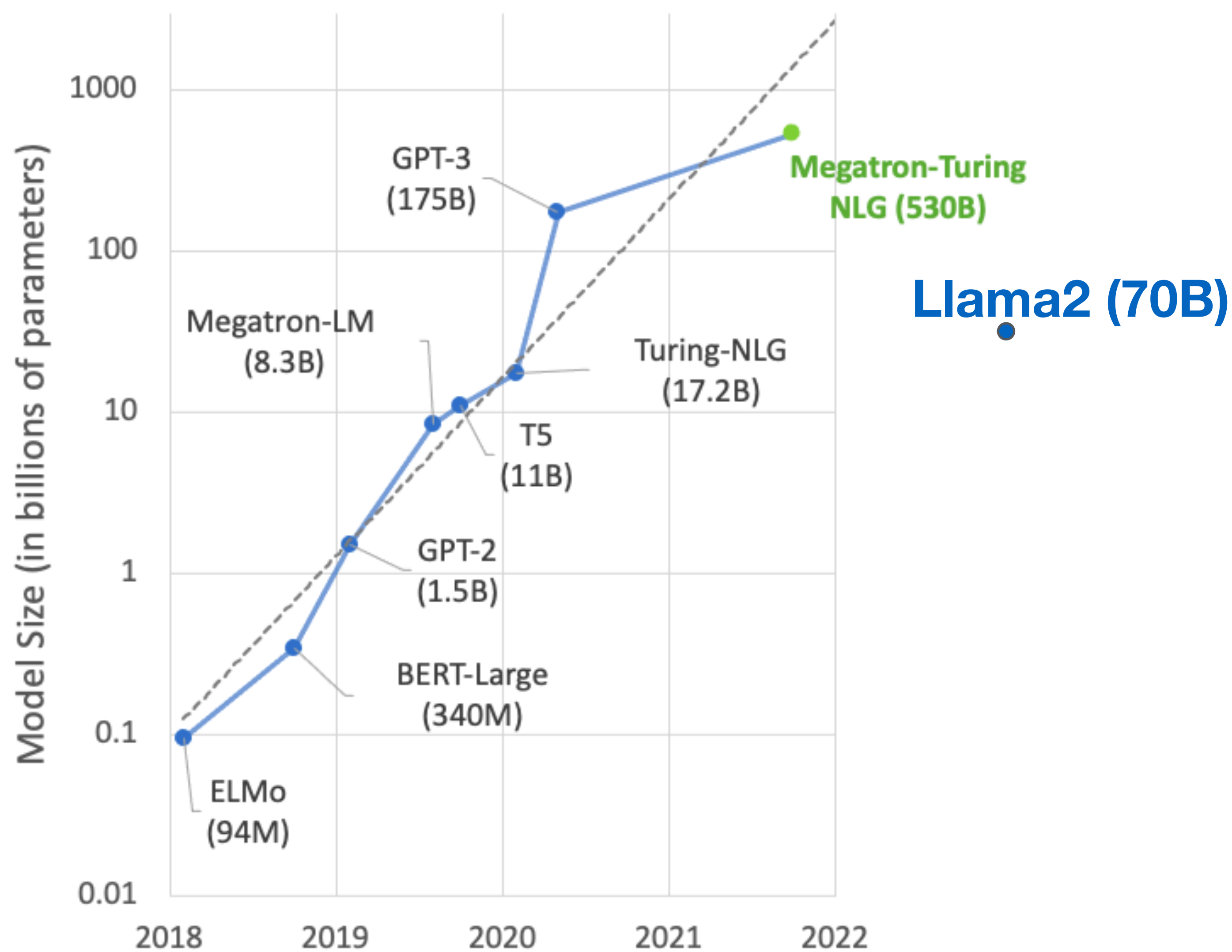
But inference is even more expensive

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 top models for most NLP tasks are massive



Main Question

- The top models for most NLP tasks are massive
- **How can we cheaply, efficiently, and equitably deploy NLP systems without sacrificing performance?**

Answer: Model Compression

Answer: 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

Answer: Model Compression

1. Quantization
    1. keep the model the same but give up some precision

Why is this even possible?

3. Distillation
    1. train a smaller model to imitate the bigger model

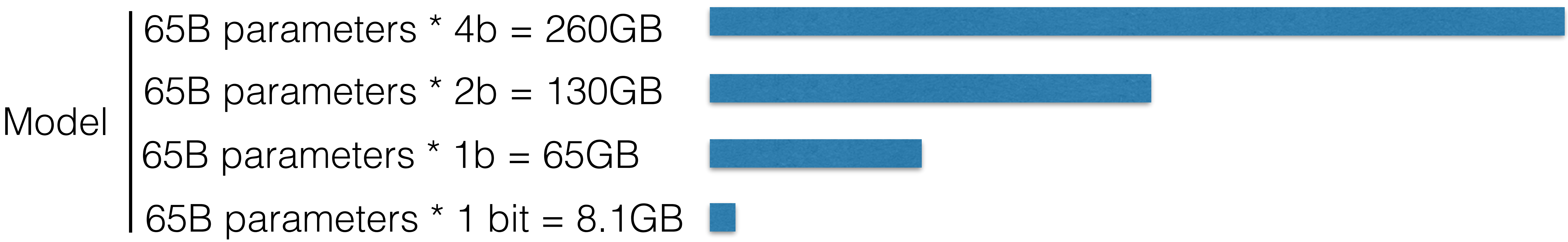
Overparameterized models are easier to optimize (Du and Lee 2018)

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

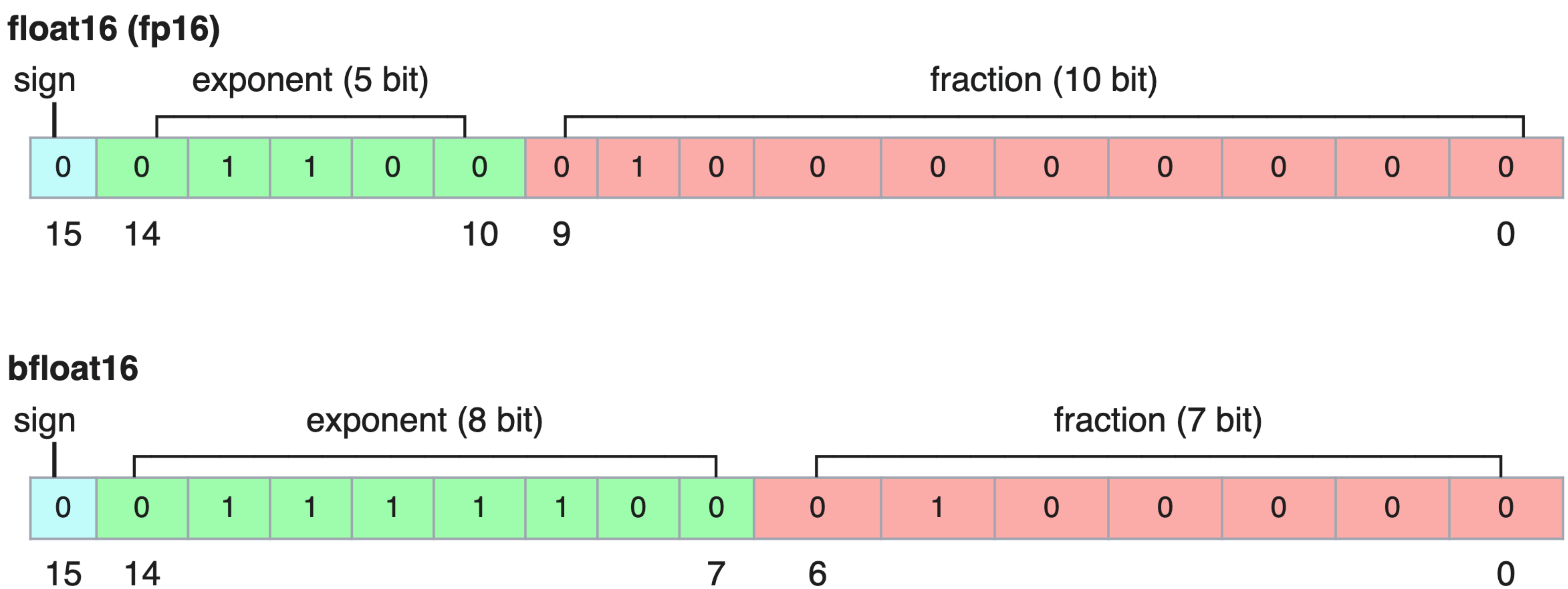


Floating point numbers

- Floating point number is stored as $(-1)^s M 2^E$
  - Sign bit *s*
  - Fractional part *M* = frac
  - Exponential part *E* = exp - bias

Source: Lecture 4 from 15-213, taught in Summer 2022

Reduced-precision floating point types



Int8 quantization

- Absolute Maximum (absmax) quantization:

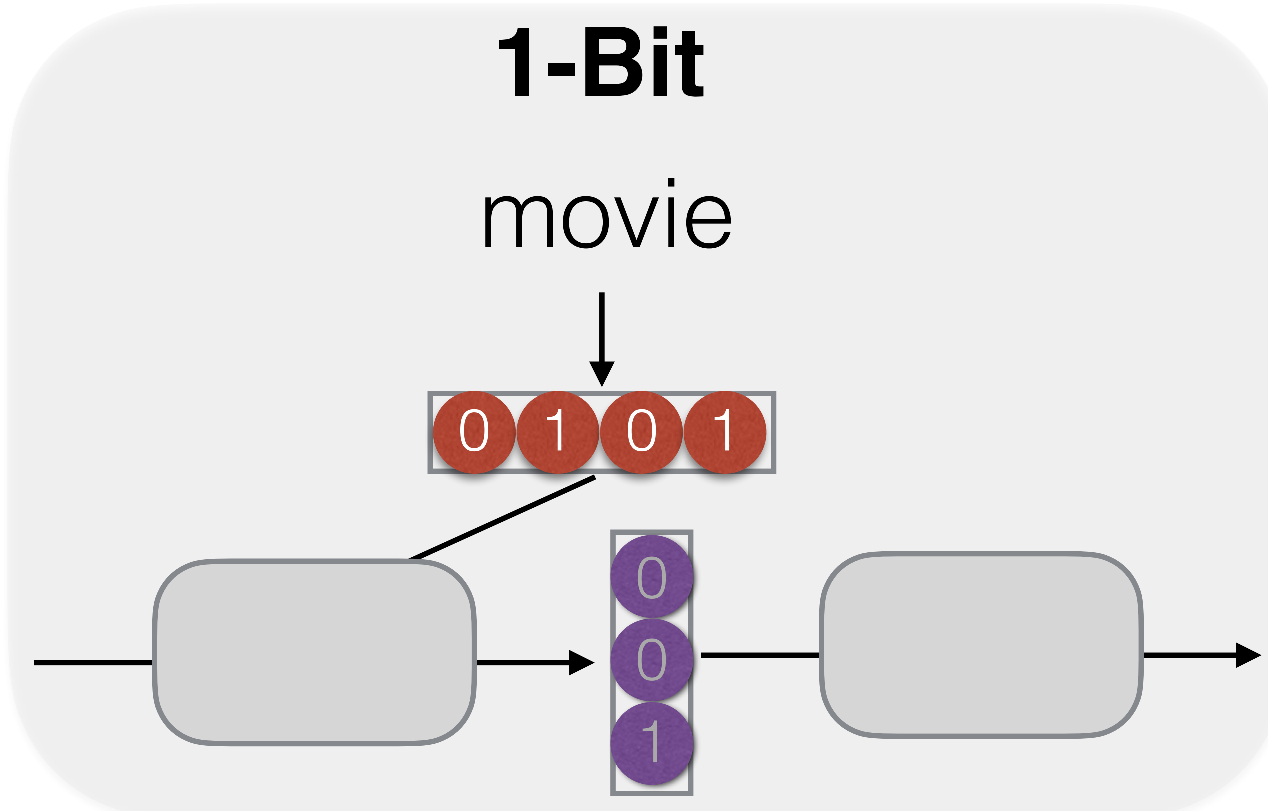
$$\mathbf{X}_{i8} = \left\lfloor \frac{127 \cdot \mathbf{X}_{f16}}{\max_{ij}(|\mathbf{X}_{f16_{ij}}|)} \right\rceil$$

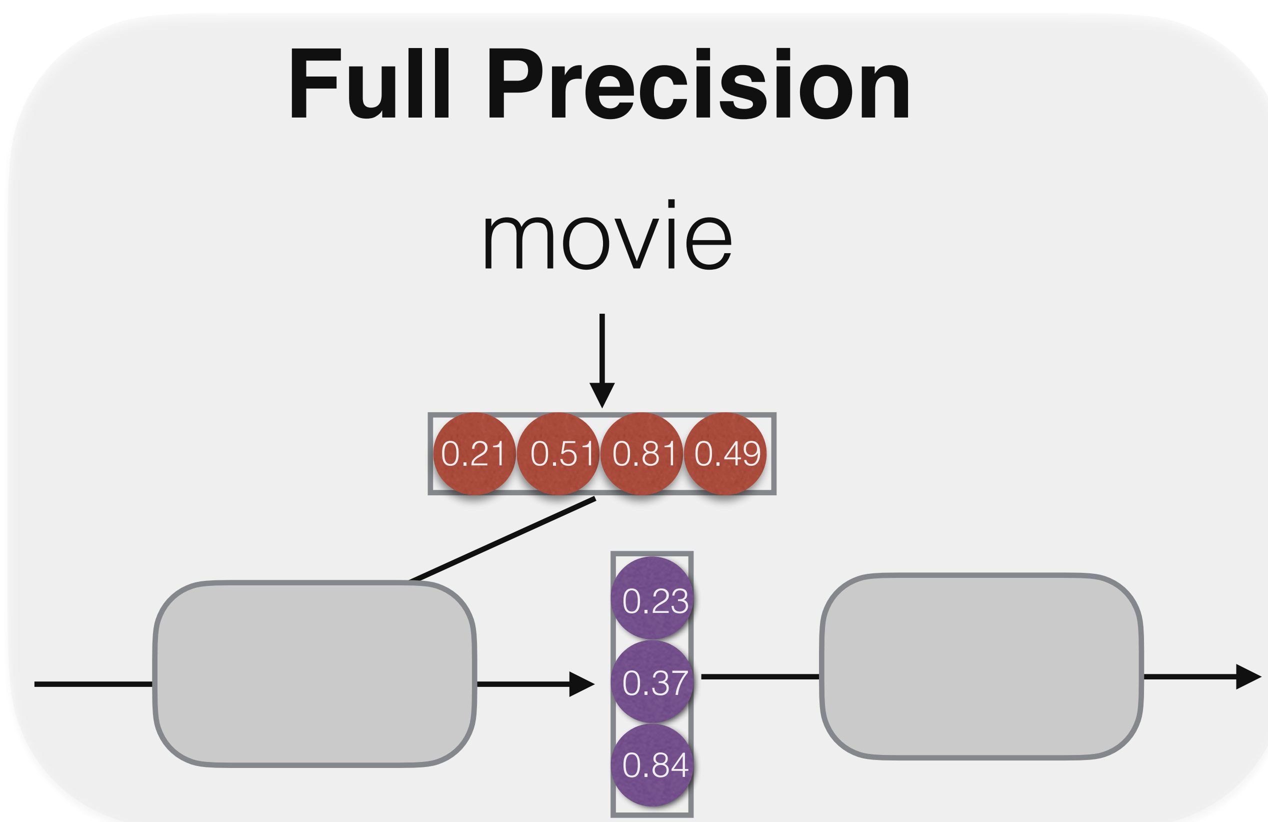
  - This scales inputs to [-127, 127]

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

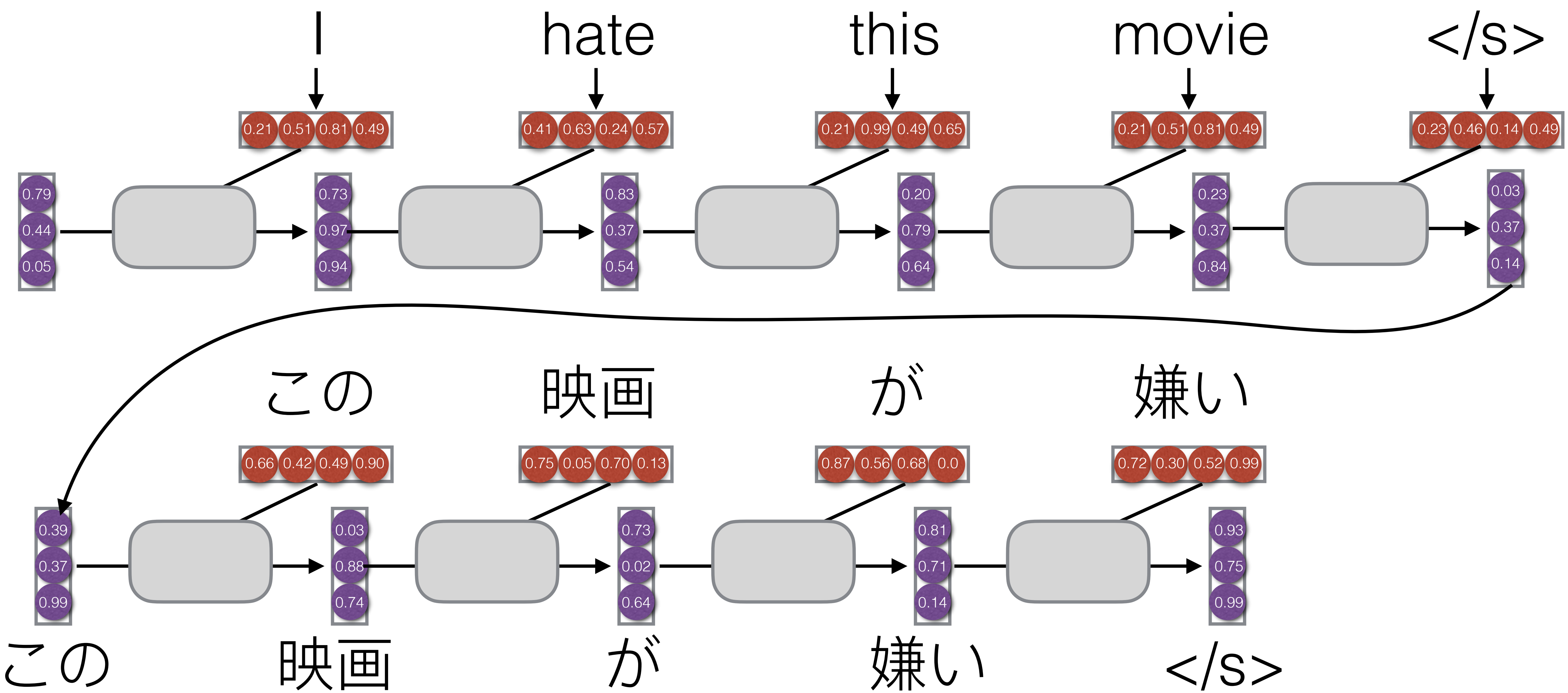
- Maximum entry is 20
- round(127/20 * [0.5, 20, -0.0001, -.01, -0.1]) -> [3, 127, 0, 0, -1]

Extreme Example: Binarized Neural Networks

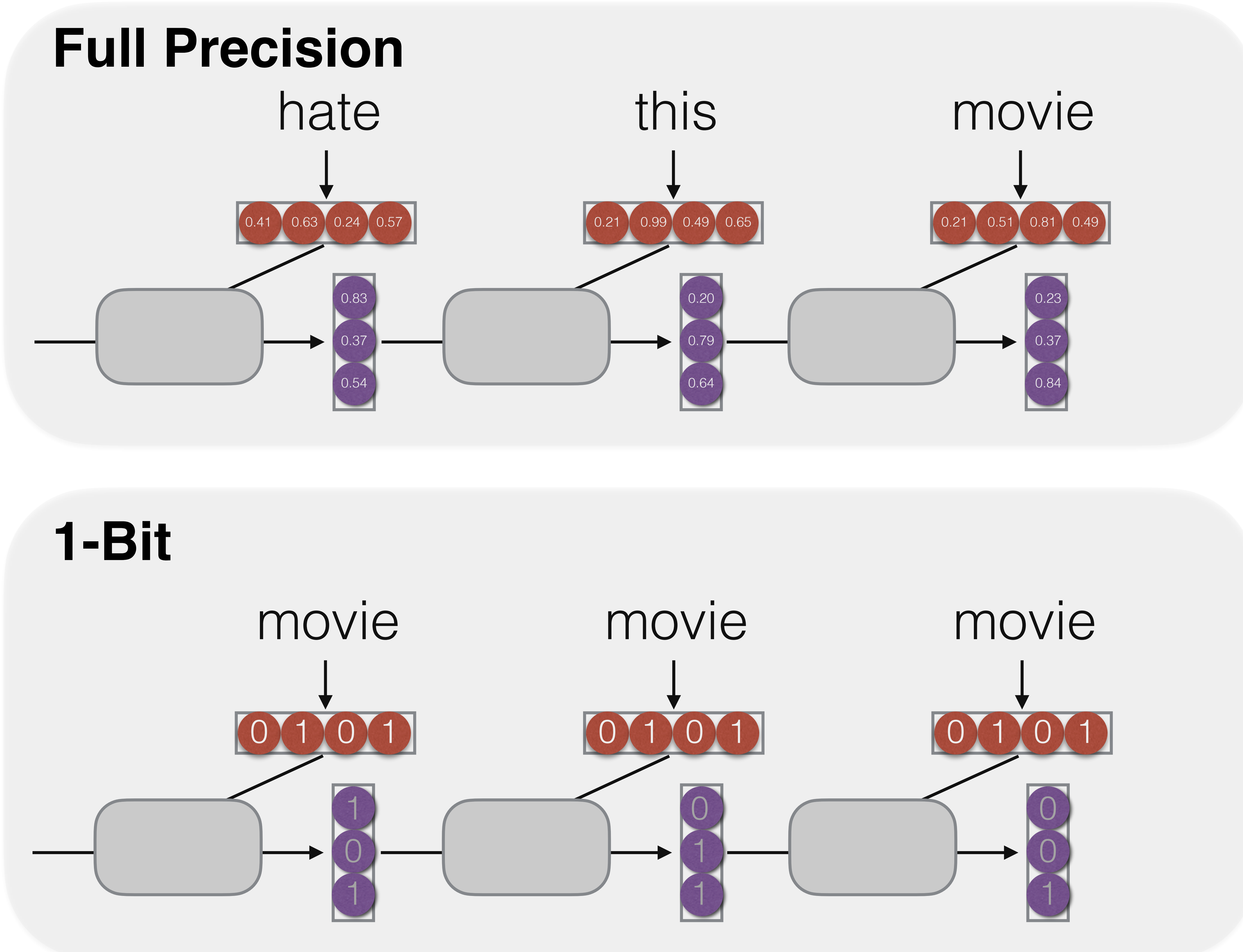




Extreme Example: Binarized Neural Networks

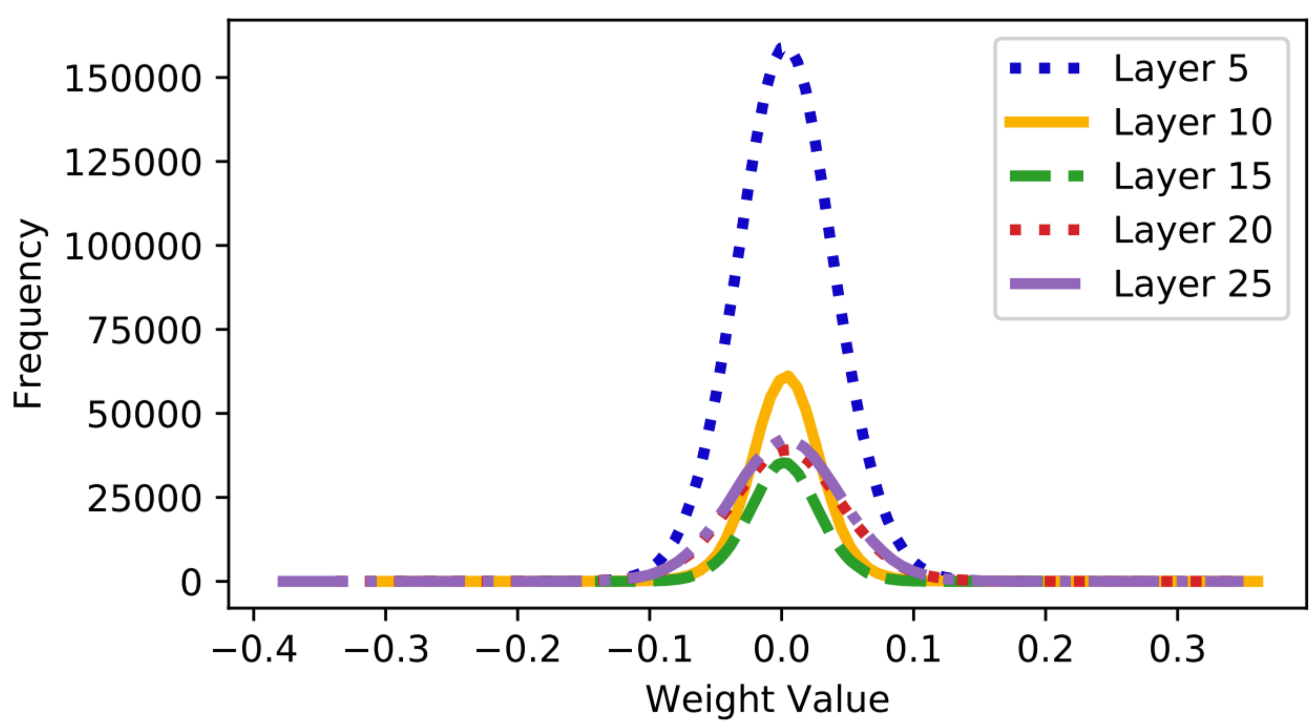


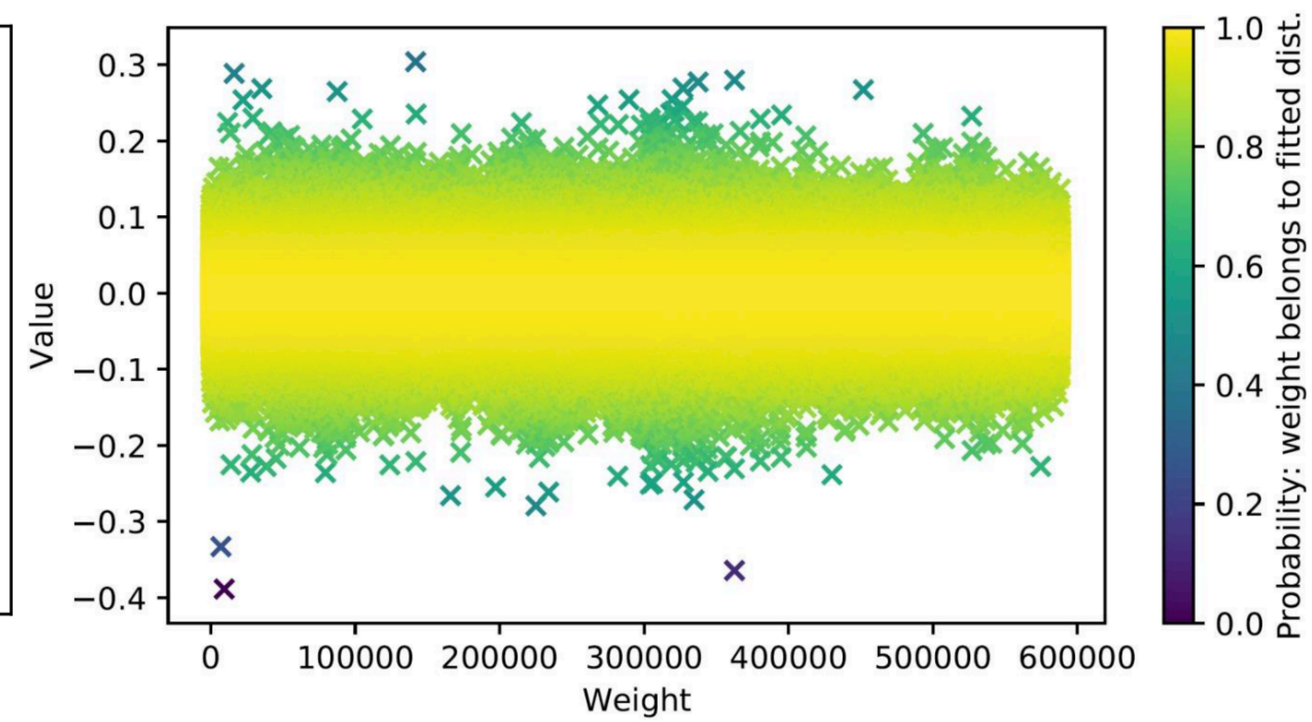
Extreme Example: Binarized Neural Networks



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



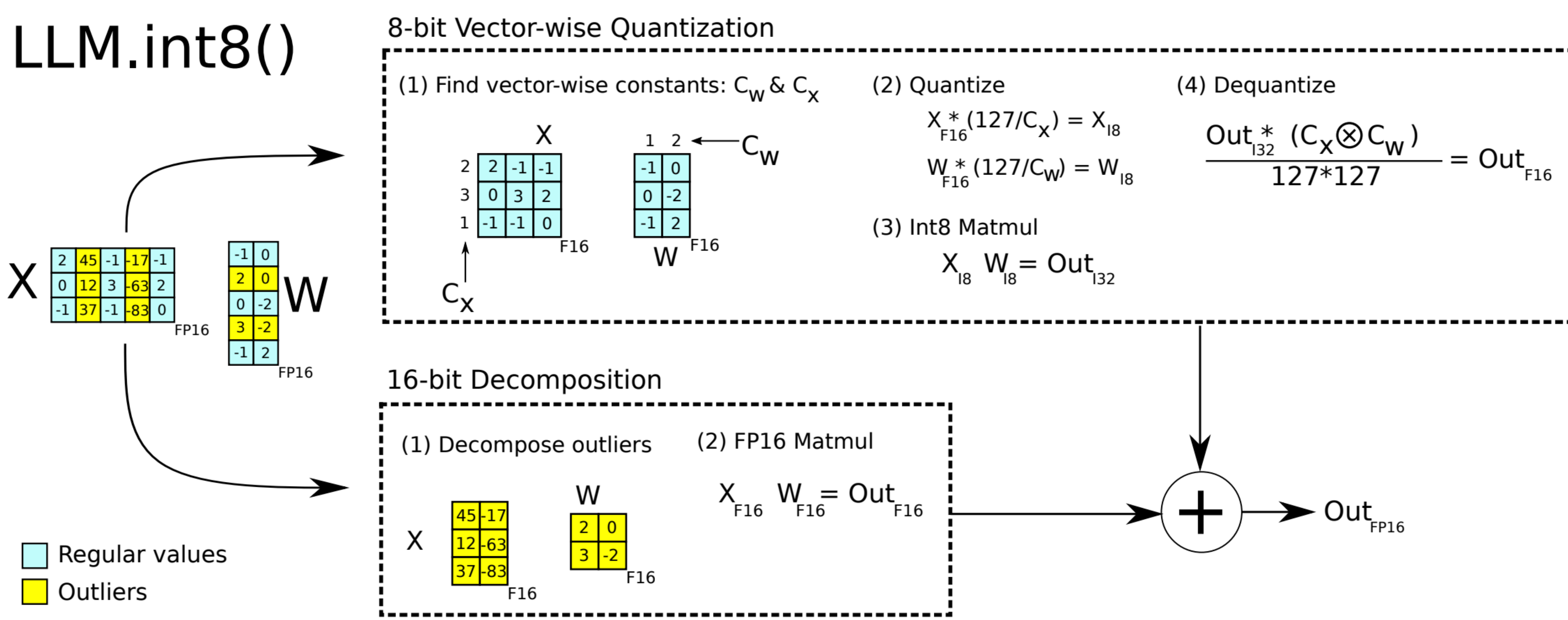


- Quantize the 99.9% of weights in the body of the disribution into 8 buckets
  - Do not quantize the remaining 0.01%

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)

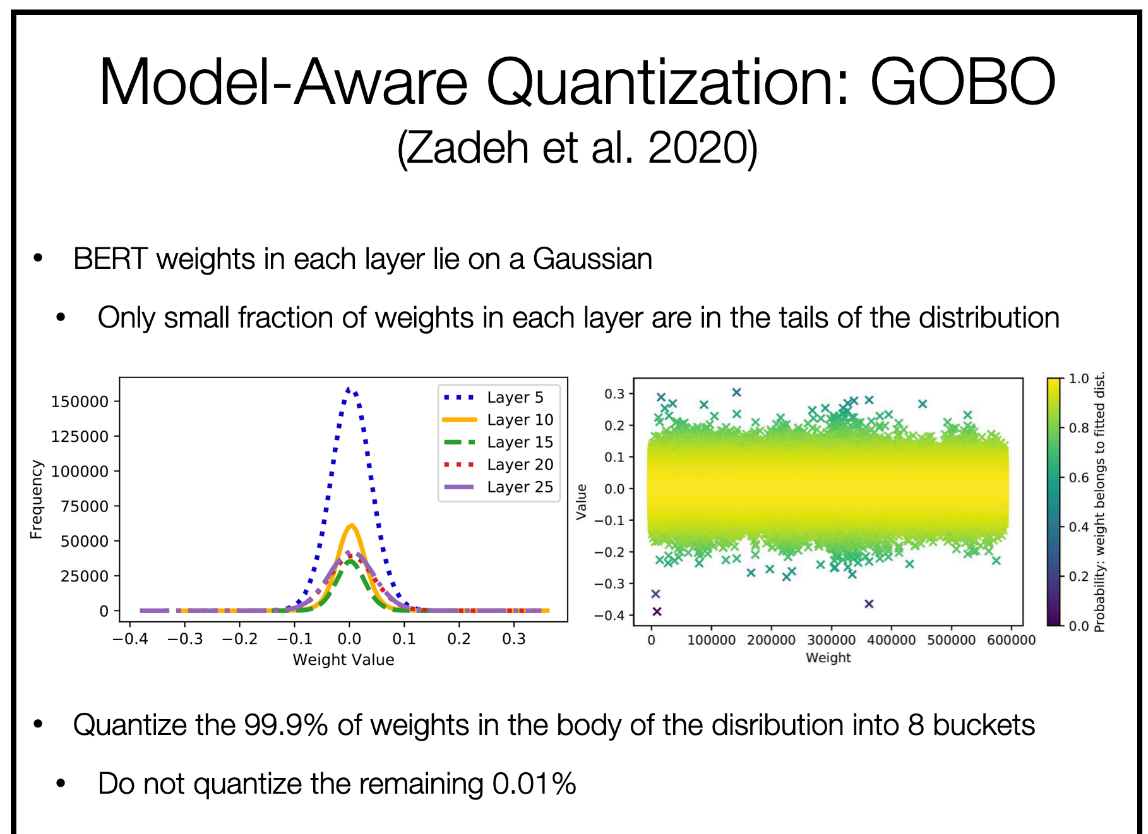
- Not all data types (e.g. “Int3”) are supported by most hardware
- PyTorch only supports certain data types (e.g. no support for Int4)

PyTorch Docs > Quantization

	Static Quantization	Dynamic Quantization
nn.Linear nn.Conv1d/2d/3d	Y Y	Y N
nn.LSTM nn.GRU	Y (through custom modules) N	Y Y
nn.RNNCell nn.GRUCell nn.LSTMCell	N N N	Y Y 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 $\sigma(x)$ and 1 with probability $1 - \sigma(x)$
- 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%

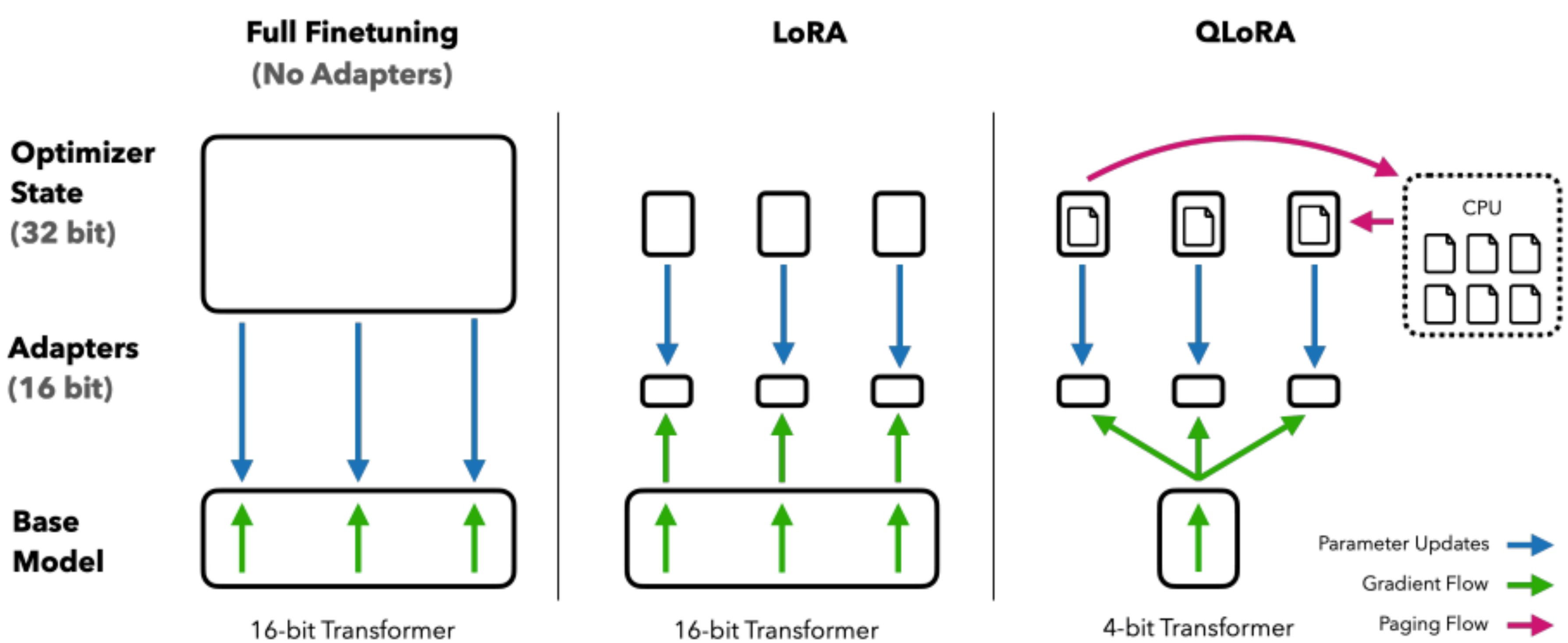
Layer-by-Layer Quantization-Aware Distillation

(Yao et al. 2022)

- Initialize the quantized network with the same architecture as the original
- Train each layer of the quantized network to mimic the output of its full-precision counterpart

Q-LORA
(Dettmers et al. 2023)

- Further compress memory requirements for training by
  - 4-bit quantization of the model (later class for details)
  - Use of GPU memory paging to prevent OOM



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

Pruning

Pruning

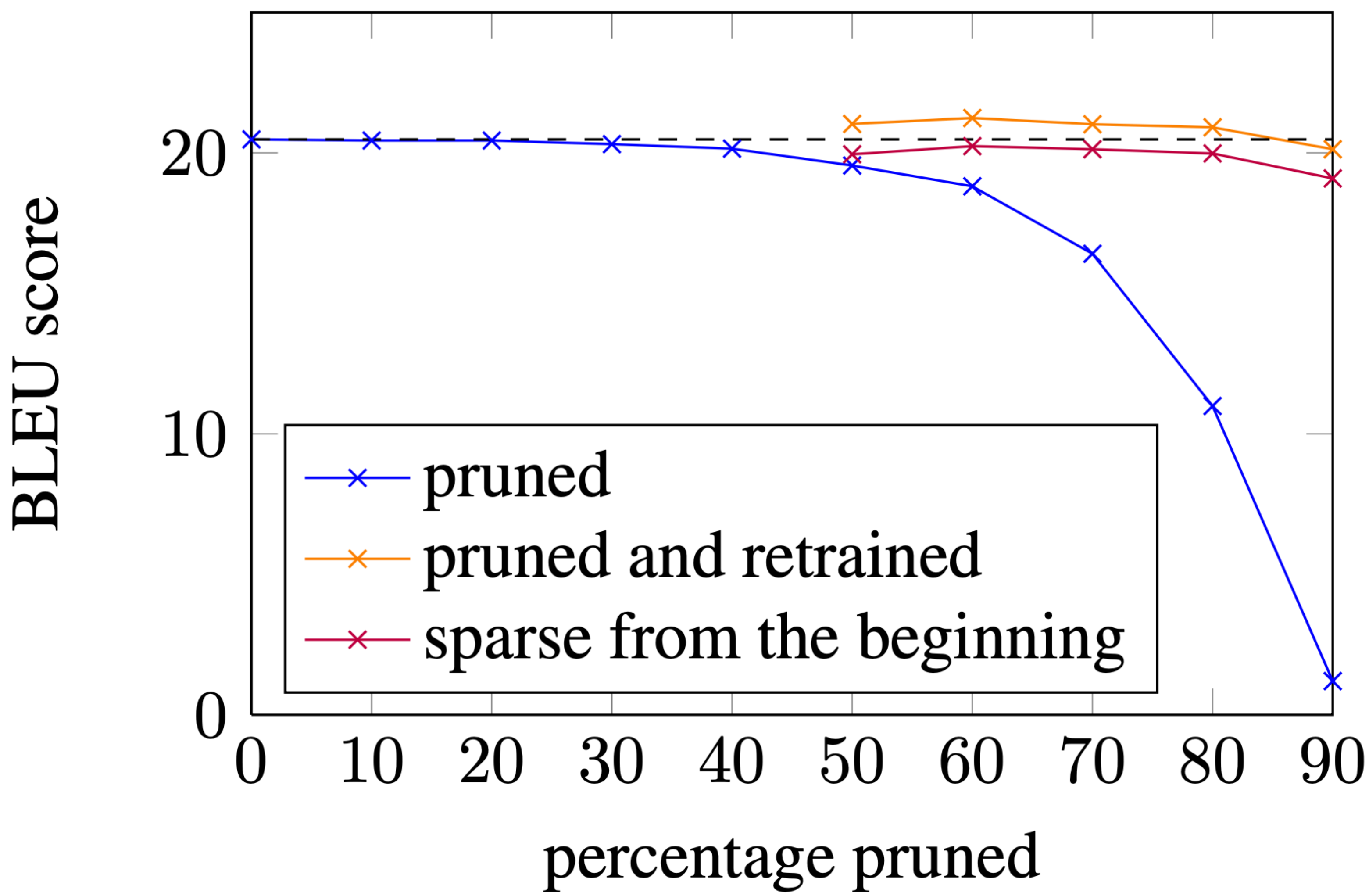
- Remove parameters from the model after training

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

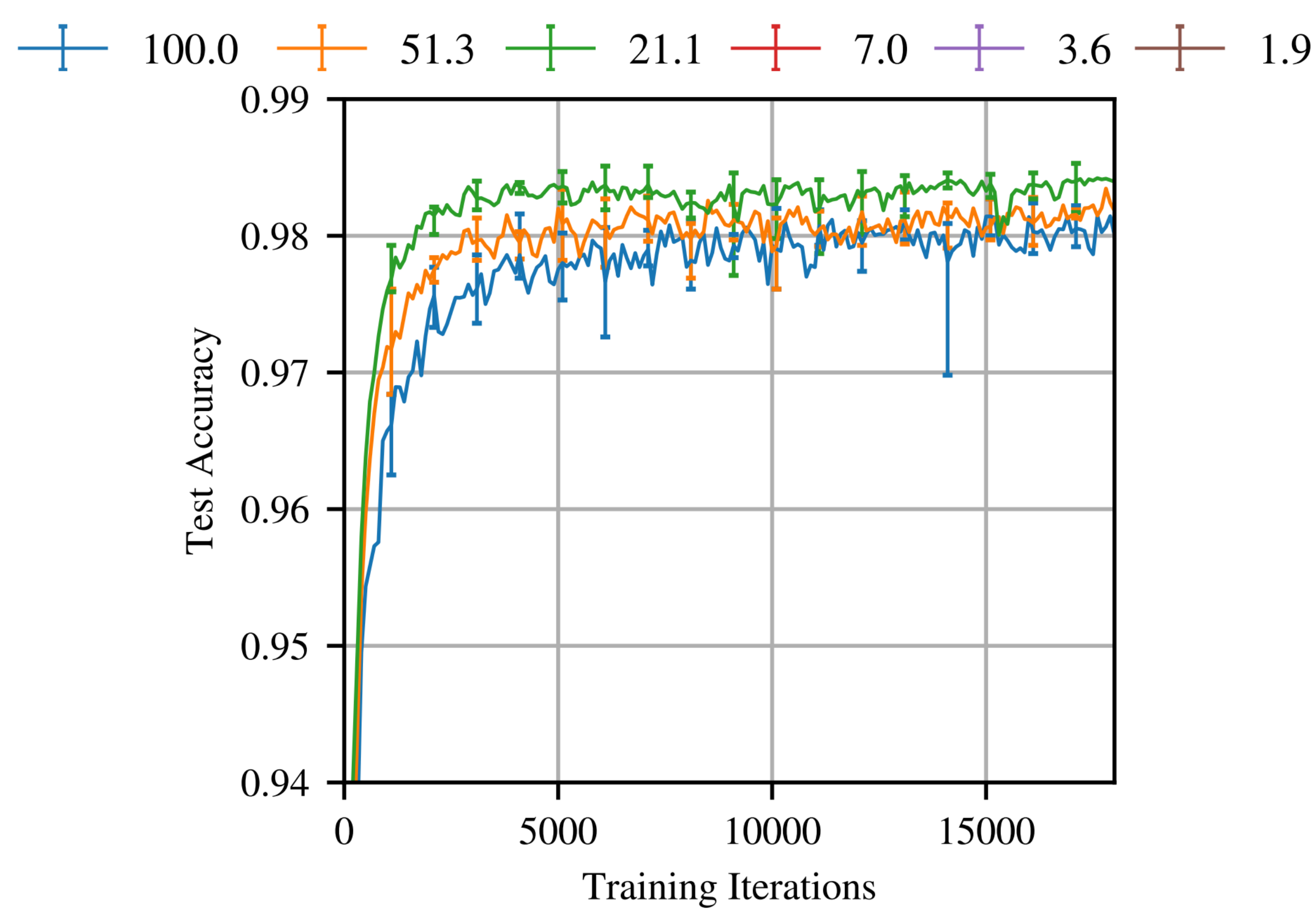
Magnitude Pruning
(Han et al. 2015, See et al. 2016)

- Zero out the X% of parameters with least magnitude
- A type of *unstructured pruning*

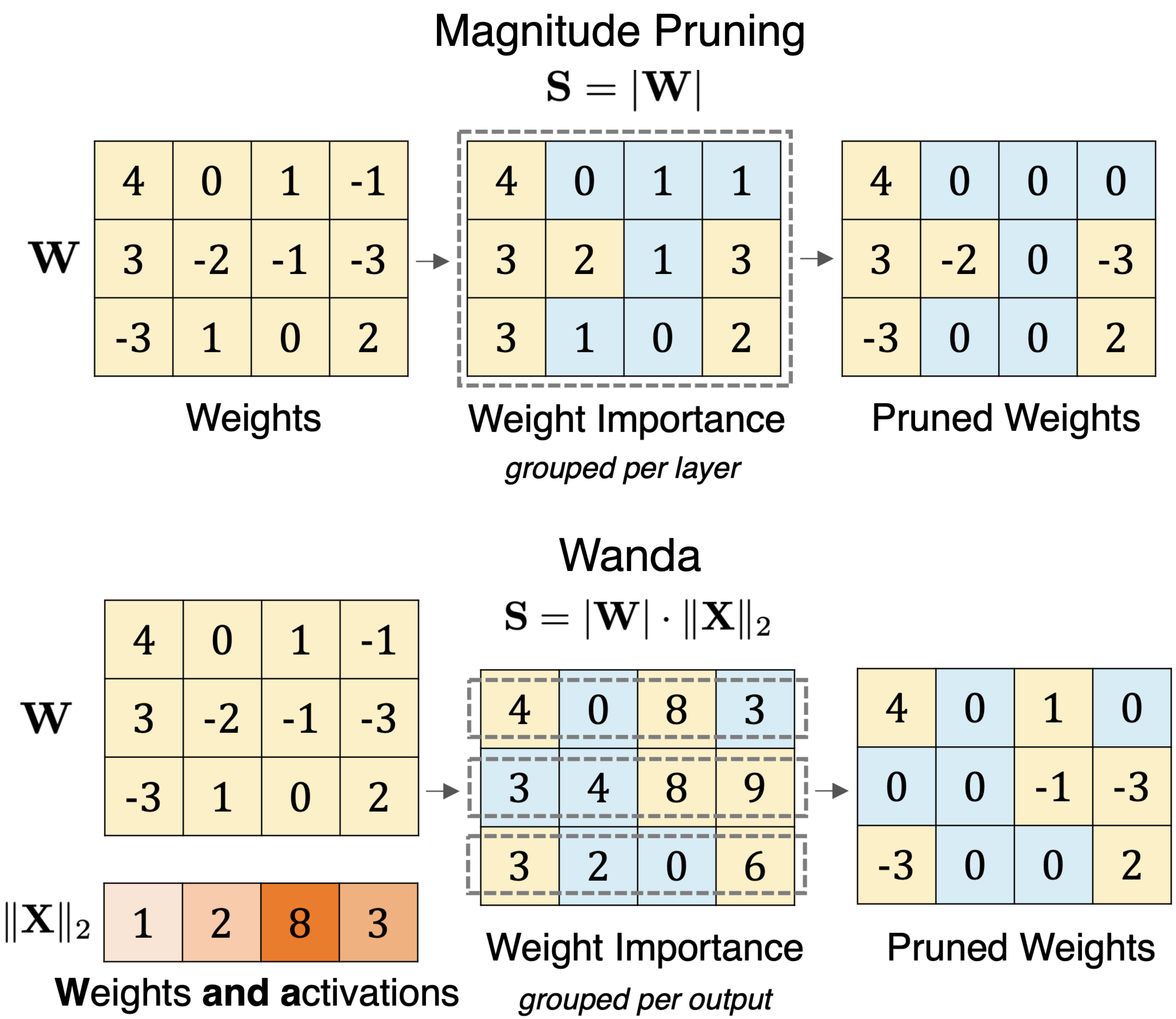


Lottery Ticket Hypothesis
(Frankle et al. 2018)

- Training a *pruned randomly-initialized* networks can be better than training the full randomly-initialized network



Wanda
(Sun et al. 2023)

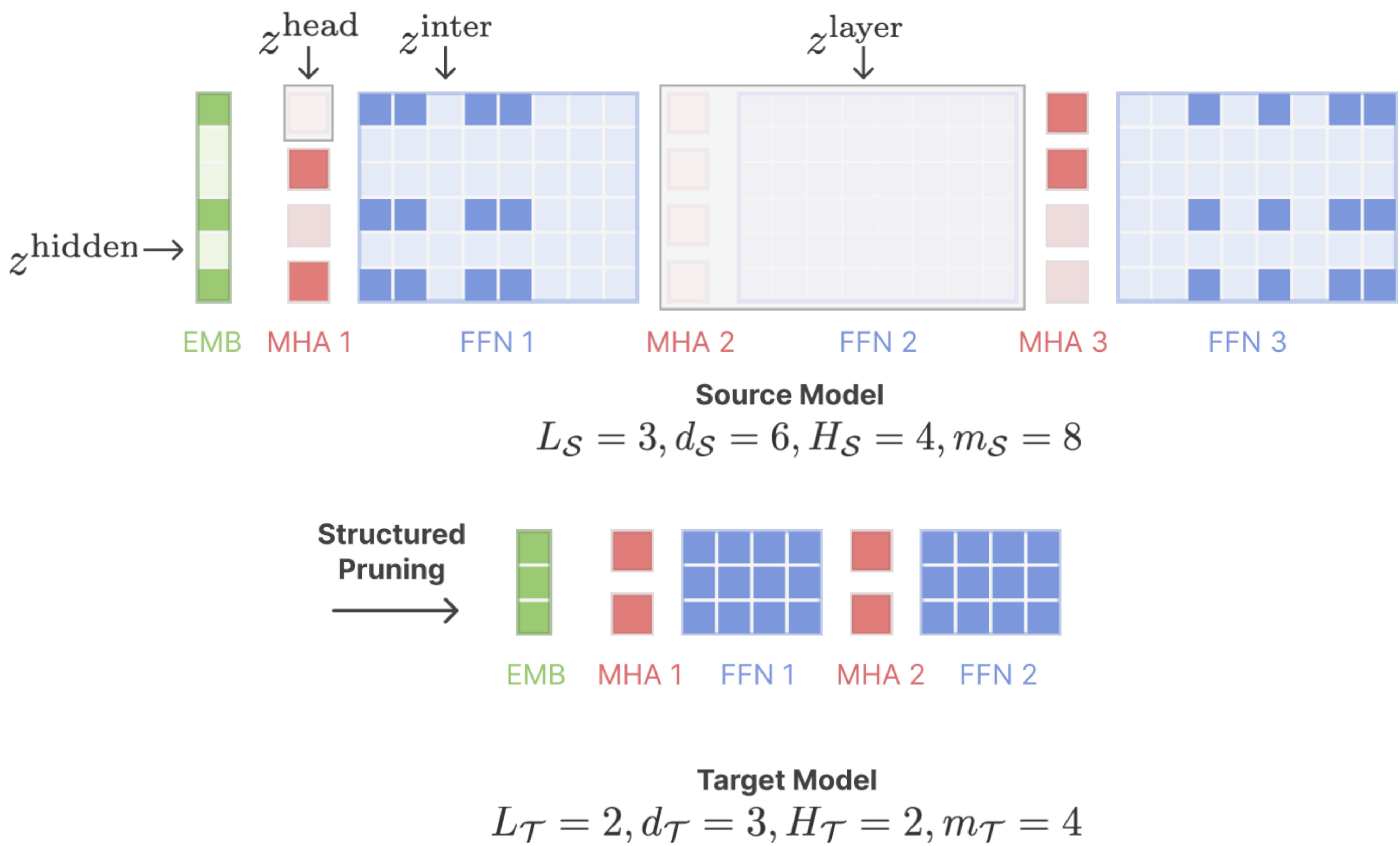


Problem with Unstructured Pruning

- Unstructured sparsity doesn’t necessarily improve memory or speed
  - Hardware that supports sparse data structures and multiplications are needed
  - This is currently an active area of work but not common in commodity hardware

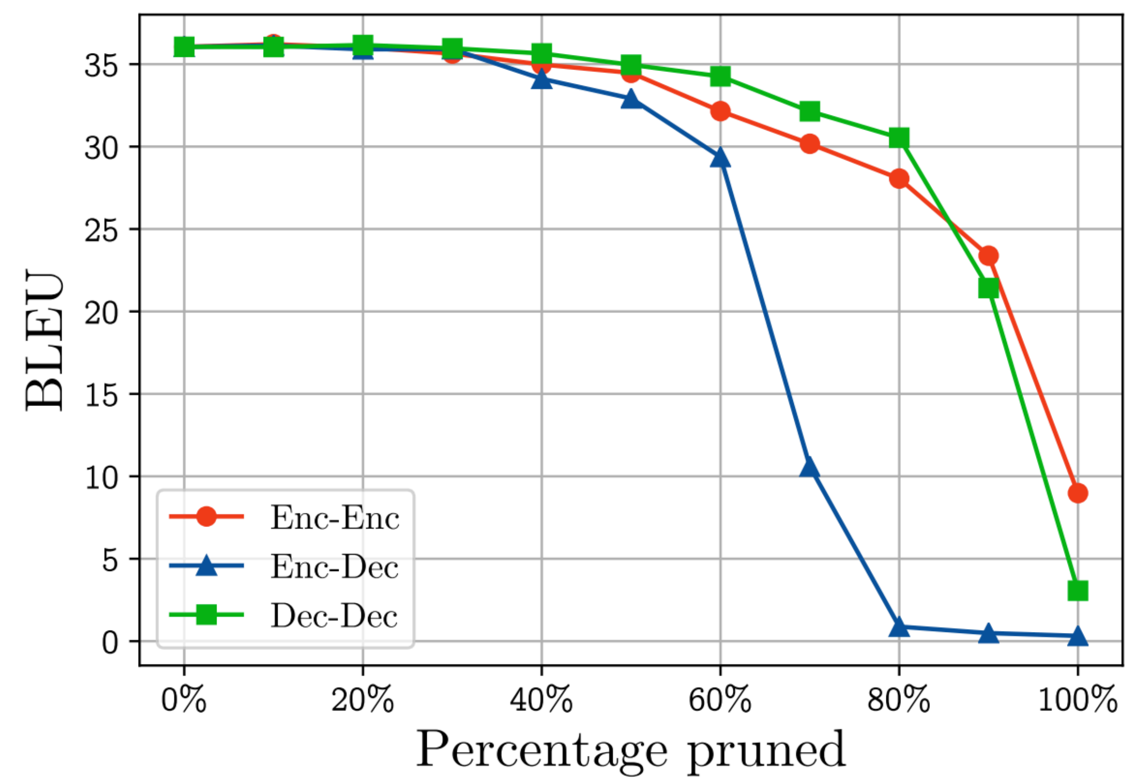
Structured Pruning
(Xia et al. 2022)

- Remove entire components
- Remaining components aren't pruned



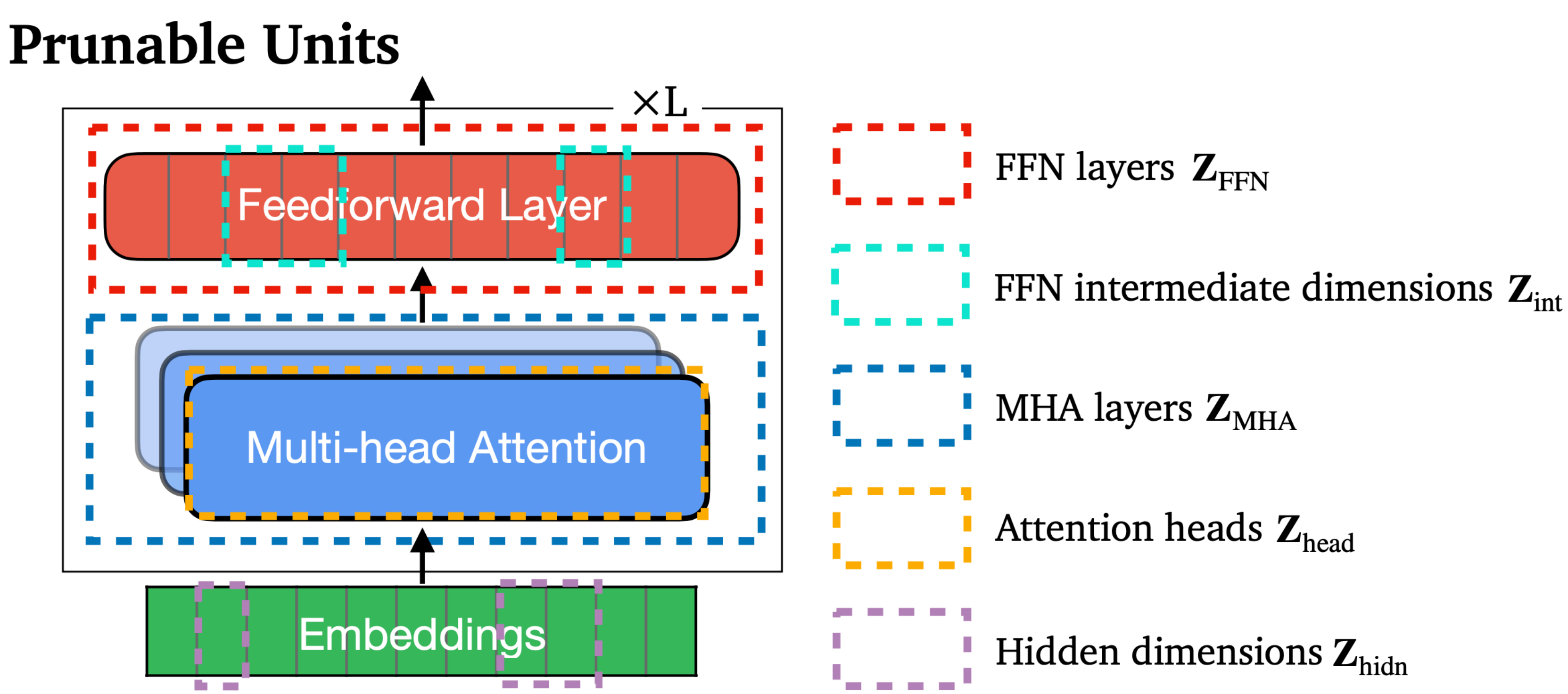
Are Sixteen Heads Really Better than One?
(Michel and Neubig 2019)

Layer \ Head	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0.03	0.07	0.05	-0.06	0.03	**-0.53**	0.09	**-0.33**	0.06	0.03	0.11	0.04	0.01	-0.04	0.04	0.00
2	0.01	0.04	0.10	**0.20**	0.06	0.03	0.00	0.09	0.10	0.04	**0.15**	0.03	0.05	0.04	0.14	0.04
3	0.05	-0.01	0.08	0.09	0.11	0.02	0.03	0.03	-0.00	0.13	0.09	0.09	-0.11	**0.24**	0.07	-0.04
4	-0.02	0.03	0.13	0.06	-0.05	0.13	0.14	0.05	0.02	0.14	0.05	0.06	0.03	-0.06	-0.10	-0.06
5	**-0.31**	-0.11	-0.04	0.12	0.10	0.02	0.09	0.08	0.04	**0.21**	-0.02	0.02	-0.03	-0.04	0.07	-0.02
6	0.06	0.07	**-0.31**	0.15	-0.19	0.15	0.11	0.05	0.01	-0.08	0.06	0.01	0.01	0.02	0.07	0.05



Coarse-to-Fine Structured Pruning
(Xia et al. 2022)

- Transformer layers consist of two components: self-attention and feed-forward
- Idea: learn "masks" that control which components to turn off
  - *Coarse masks:* entire self-attention or feed-forward components
  - *Fine masks:* attention heads and hidden state dimensions



Pruning w/ Forward Passes
(Dery et al. 2024)

- Structured pruning big models requires a lot of memory
- Can we avoid using gradients?
- **Idea**
  1. measure the performance of a model with different modules masked
  2. learn the impact of each module mask via regression

Pruning w/ Forward Passes
(Dery et al. 2024)

Model	~Size	Fine-tune	PPL	Speedup
Phi-2	3B	✓	8.69	1.24×
		LLaMA-2 7B Pruned		
Wanda 2:4	3B	✗	10.52	1.14×
		✓	8.34	0.75×
Bonsai	3B	✓	8.89	**1.58**×

Distillation

Distillation

- Train one model (the “student”) to replicate the behavior of another model (the “teacher”)

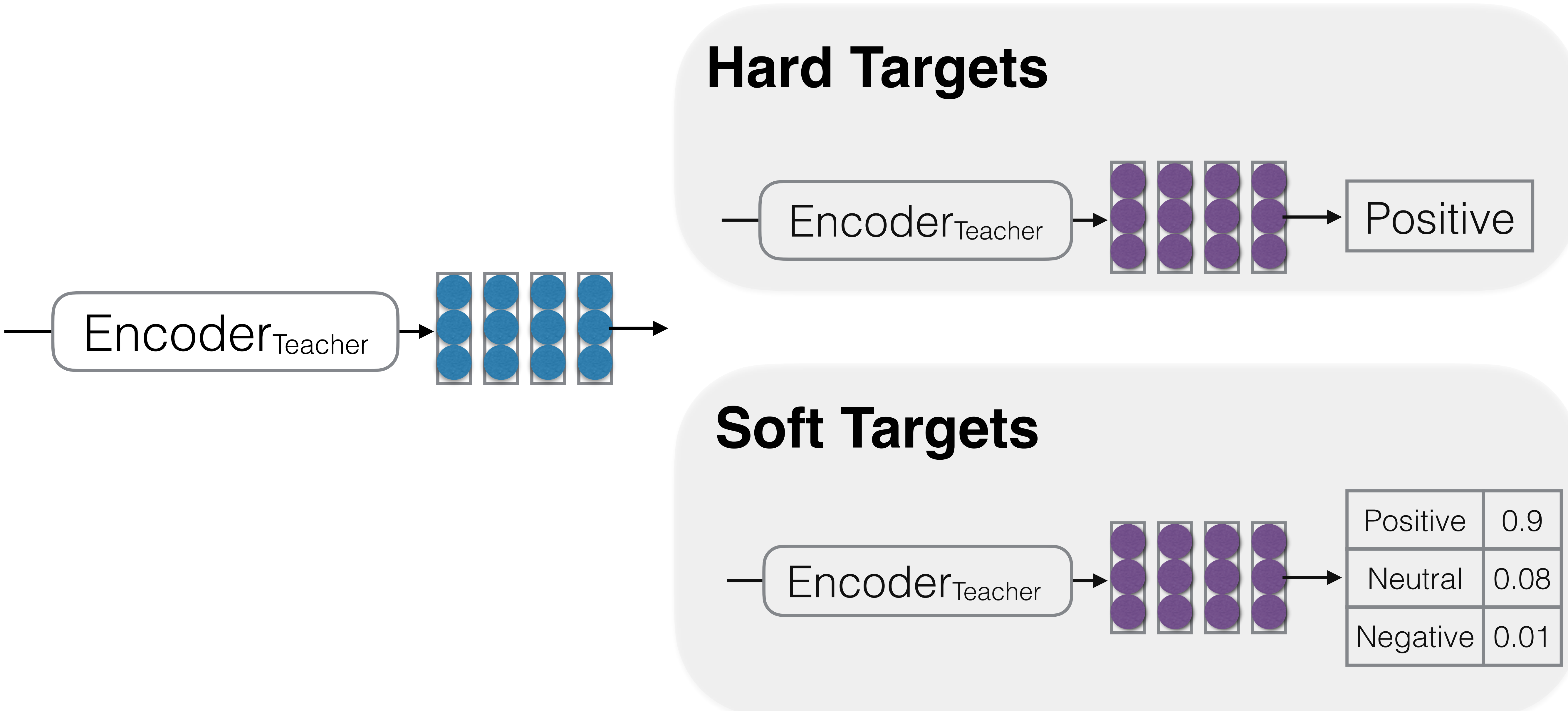
Distillation vs Quantization vs Pruning

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

Weak Supervision
(Yarowski 1995)

- *Pseudo-labels* are targets generated for unlabeled text
    - We can train on *pseudo-labels* as though they are labels
- This idea is old and used in many ideas
    - Self-training (Yarowski 1995)
    - Co-training (Blum and Mitchell 1998)
    - Meta Pseudo Labels (Pham et al 2020)

Hard vs Soft Targets
(Hinton et al 2015)



System & training set	Train Frame Accuracy	Test Frame Accuracy
Baseline (100% of training set)	63.4%	58.9%
Baseline (3% of training set)	67.3%	44.5%
Soft Targets (3% of training set)	65.4%	57.0%

Sequence-Level Distillation
(Kim and Rush 2016)

- Can we extend *soft targets* to sequences?
- 2 ways:
  - *Word-level distillation:* match distribution of words at each step with the teacher's distribution
  - *Sequence-level distillation:* maximize probability of the output generated by the teacher

$$\mathcal{L}_{\text{WORD-KD}} = -\sum_{j=1}^{J}\sum_{k=1}^{|\mathcal{V}|} q(t_j = k \mid \mathbf{s}, \mathbf{t}_{<j}) \times \log p(t_j = k \mid \mathbf{s}, \mathbf{t}_{<j})$$

$$\begin{aligned}\mathcal{L}_{\text{SEQ-KD}} &\approx -\sum_{\mathbf{t}\in\mathcal{T}} \mathbb{1}\{\mathbf{t} = \hat{\mathbf{y}}\} \log p(\mathbf{t} \mid \mathbf{s}) \\ &= -\log p(\mathbf{t} = \hat{\mathbf{y}} \mid \mathbf{s})\end{aligned}$$

$$\mathcal{L} = (1-\alpha)\mathcal{L}_{\text{SEQ-NLL}} + \alpha\mathcal{L}_{\text{SEQ-KD}}$$

DistilBERT
(Sanh et al 2019)

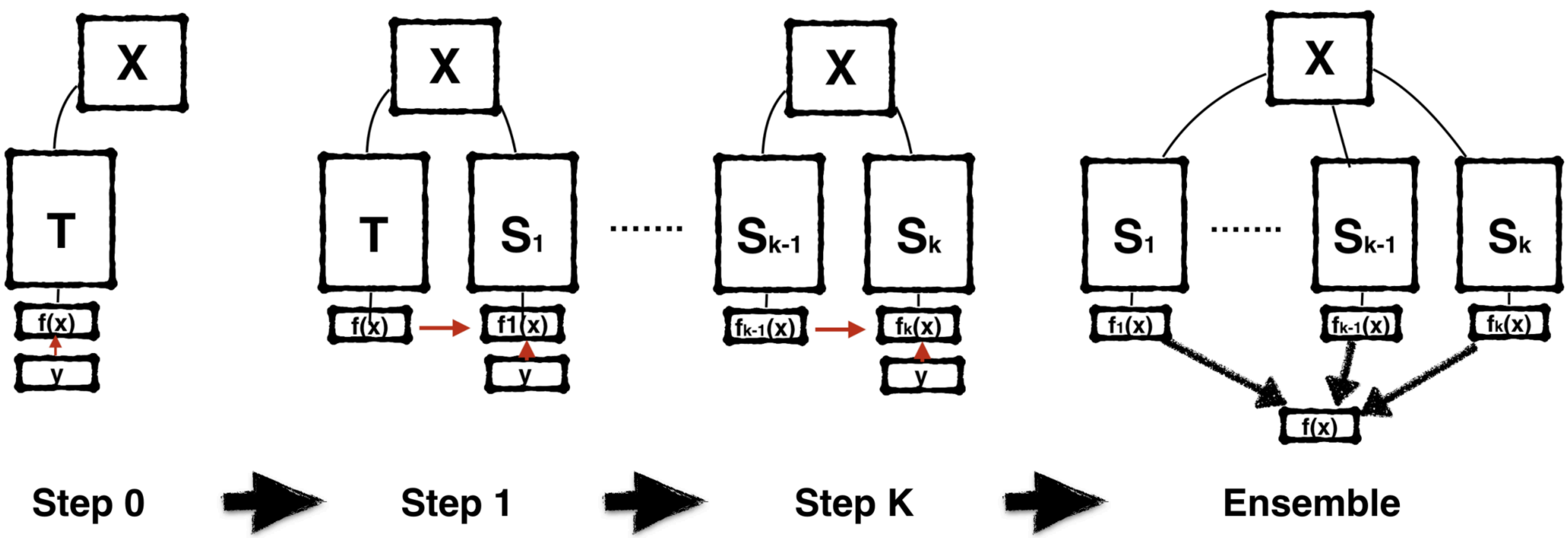
- Uses half the layers and 60% of total parameters
- Tricks:
  - Initialize DistilBERT with alternating layers from BERT
  - Use both supervised and distillation-based losses
    - Supervised loss doesn't help much
  - Add cosine similarity of hidden state vectors between teacher and student

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

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

Born Again Neural Networks

(Furlanello, Lipton, et al 2018)



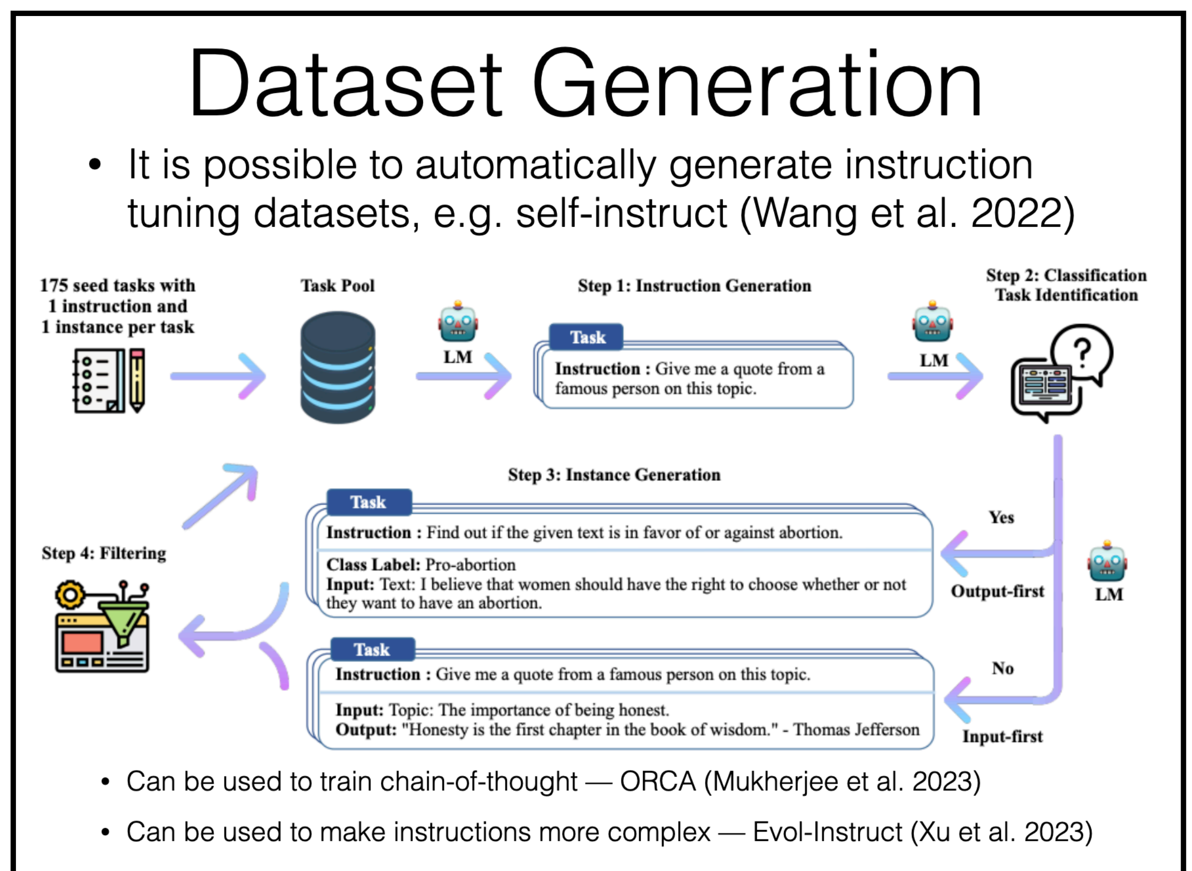
Test error on CIFAR-100

Network	Teacher	BAN
DenseNet-112-33	18.25	**16.95**
DenseNet-90-60	17.69	**16.69**
DenseNet-80-80	17.16	**16.36**
DenseNet-80-120	16.87	**16.00**

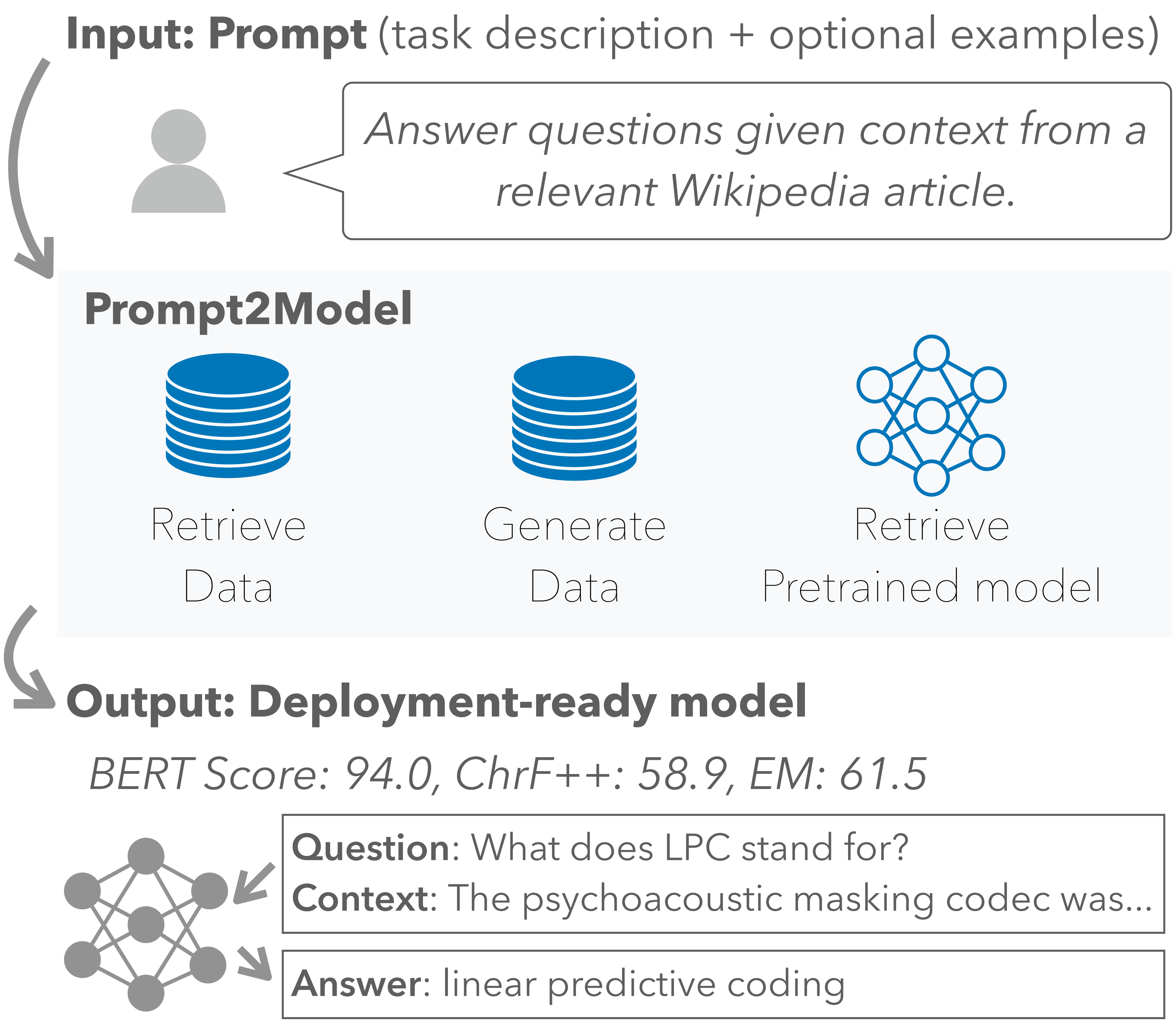
Self-Instruct

(Wang et al 2022)

- Use distillation to train a vanilla LM to follow instructions by synthesizing and pseudo-labeling instructions using itself



Prompt2Model
(Viswanathan et al 2023)



A Toolkit for Synthetic Data Generation

(Patel et al 2024)

Type		Examples
Steps	Load a Dataset	`DataSource`, `HFHubDataSource`, `JSONDataSource`, `CSVDataSource`, ...
	Prompting	`Prompt`, `RAGPrompt`, `ProcessWithPrompt`, `FewShotPrompt`, `DataFromPrompt`, `DataFromAttributedPrompt`, `FilterWithPrompt`, `RankWithPrompt`, `JudgeGenerationPairsWithPrompt`, ...
	Other	`Embed`, `Retrieve`, `CosineSimilarity`, ...
Models		`OpenAI`, `OpenAIAssistant`, `HFTransformers`, `CTransformers`, `VLLM`, `Petals`, `HFAPIEndpoint`, `Together`, `MistralAI`, `Anthropic`, `Cohere`, `AI21`, `Bedrock`, `Vertex`, ...
Trainers		`TrainOpenAIFineTune`, `TrainHFClassifier`, `TrainHFFineTune`, `TrainSentenceTransformer`, `TrainHFDPO`, `TrainHFPPO`, ...

Questions?