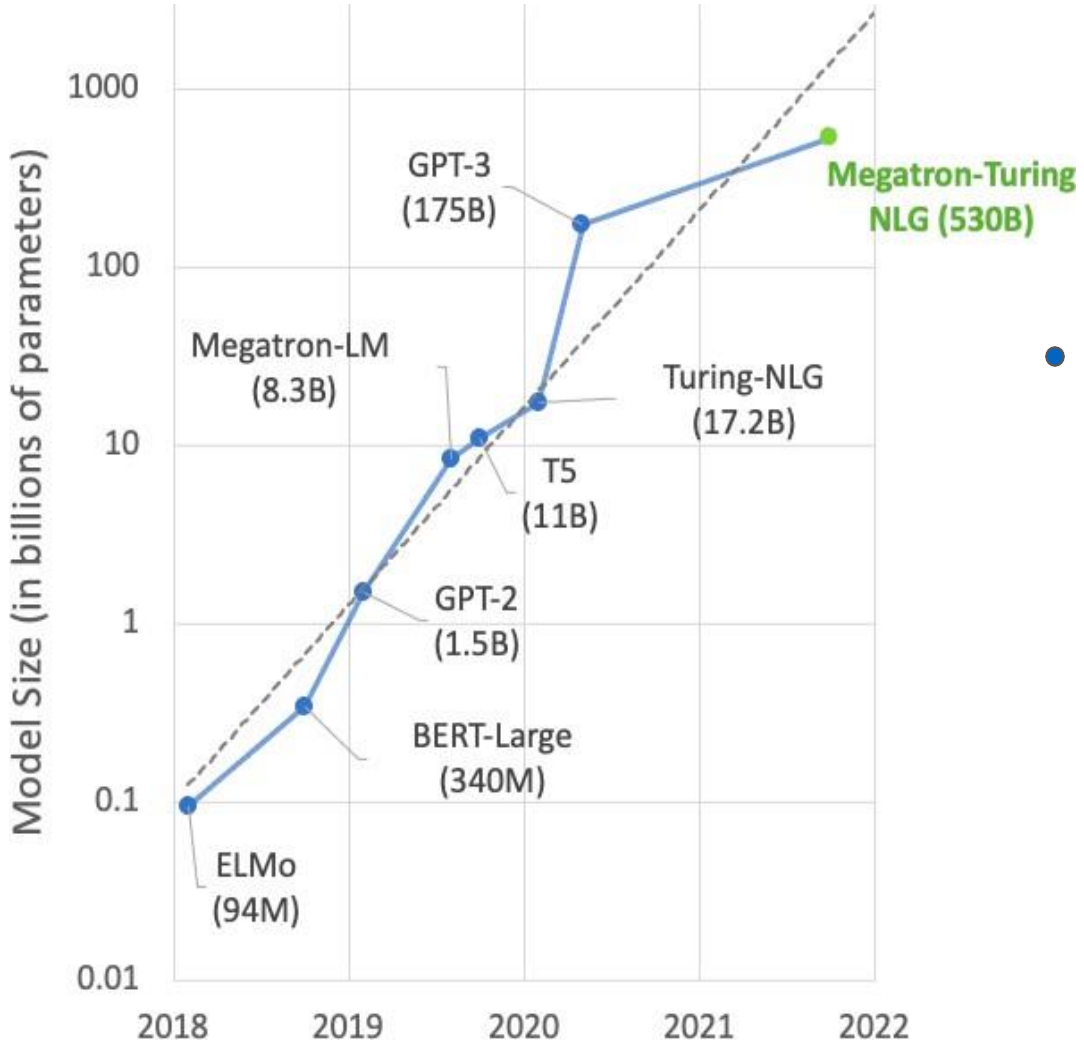
Efficiency!

Quantization, Pruning, & Distillation

Bigger is better paradigm

- Recent models follow the "bigger-better" approach.
- Both **Training and inference** are computationally challenging due to the models' size



Efficiency!

How should large models be trained and deployed while significantly reducing computational cost without hurting performance (too much)

Alternatively – *how come we can run LLM on our laptops!*?

Efficiency!

The key mechanisms we'll discuss in this lecture:

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

All can be considered as *model compression*, though via different means

Quantization

- Current models are huge! Often in the 100's Billion parameters range!

But… **the model's size in memory is also dependent on the representation of these parameters in memory.**

A parameter is real-valued, represented as a floating point.

Key idea: reduce memory requirements by using "cheaper" less exact floating point representation.

Quantization

- **Let’s start with some math:**

Typical representation: 2B (16b) floating point for each parameter.

Let’s assume an 8B model, storing it takes up 16GB.

Running inference requires more memory~ 24GB

That’s a significant memory requirement !

Now, assume each parameter can be represented using 4b

6GB is enough!

my laptop can handle it!

Reminder: number representation

- Unsigned integer Representation: each bit is a power of 2

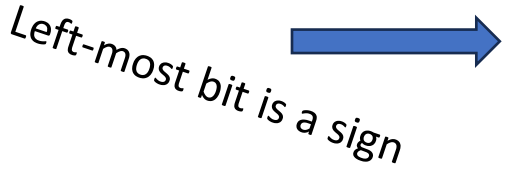
0	0	0	1	0	0	0	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

= 17

- U8: 8-bit unsigned integer: 0 to $2^8 - 1 = 255$
- U16: 16-bit unsigned integer: 0 to $2^{16} - 1 = 65{,}535$
- U32: 32-bit unsigned integer: 0 to 4,294,967,295

Reminder: number representation

- **Signed** integer Representation



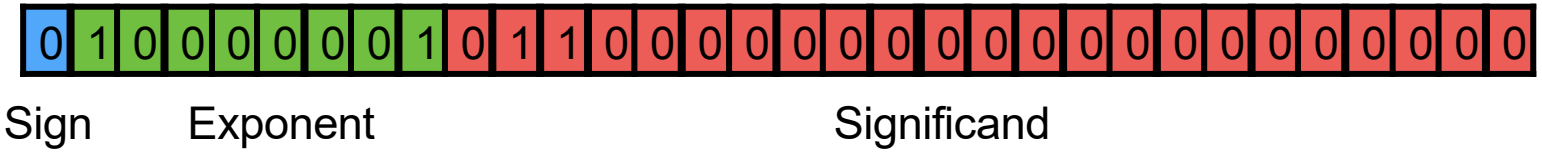
$$1\ 0\ 1\ 0 \quad = -6$$
$$2^3\ 2^2\ 2^1\ 2^0$$

- **2-complement :**
  - The most significant bit (MSB) acts as a sign bit (0 for positive, 1 for negative).
  - Positive numbers are stored as standard binary
  - Negative numbers are formed by flipping all bits of the absolute value and adding 1.

Two's complement binary	Decimal
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

Floating point

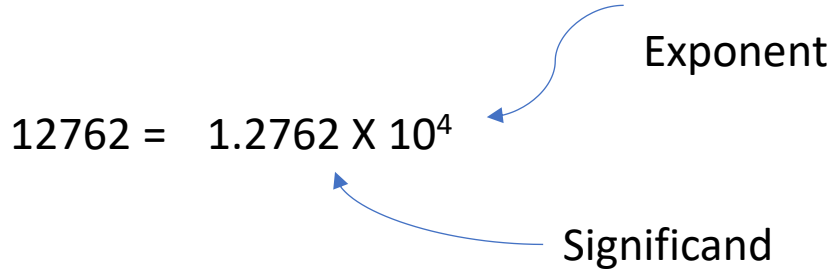
- Represents a subset of real-valued numbers
- Sign, exponent, and significand bits

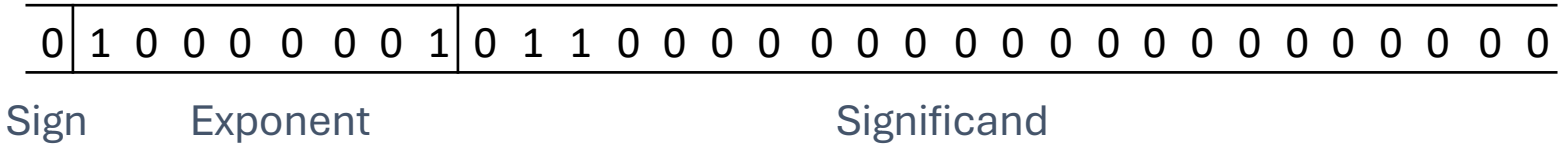


- value = $(-1)^{\text{sign}} \times (1.\text{significand}) \times 2^{\text{exponent}-\text{bias}}$
  - Bias allows for storing the exponent as an unsigned int
  - Binary scientific notation: 1.101×2^3 = (1+0.5+0.125)*8

Floating point

- Representing real-valued numbers
  - Limited range of values and precision level
- Representation breaks vector into three parts: sign, exponent and significand bits





Example

01000000010011001100110011001101

- **Sign bit**: 0 → positive
- **Exponent**: 10000000 → 128 → = 128 – 127 = 1
  - **Actual exponent** = exponent-bias (Bias term: 127)
- **Significand:** 10011001100110011001101 ≈0.6000000238
- **Final Step:**

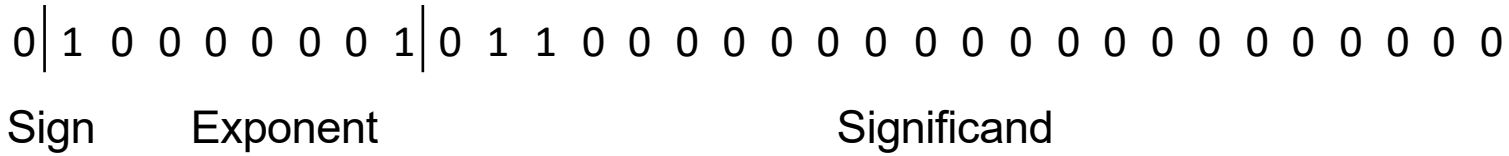
(1+0.6000000238)×2=1.6000000238×2≈3.2000000477

= 1×(1/2)
+ 0×(1/4)
+ 0×(1/8)
+ 1×(1/16)
+ 1×(1/32)
+ 0×(1/64)
+ 0×(1/128)
+ 1×(1/256)
...

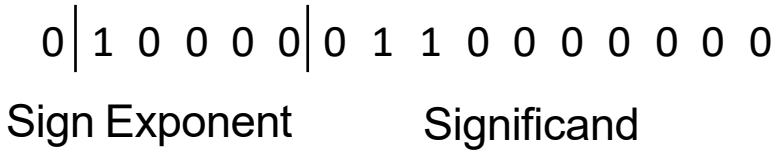
Floating points

- Float32 (“full precision”)

Min (normal): ≈1.18×10^{-38}
Max: ≈3.4×10^{38}



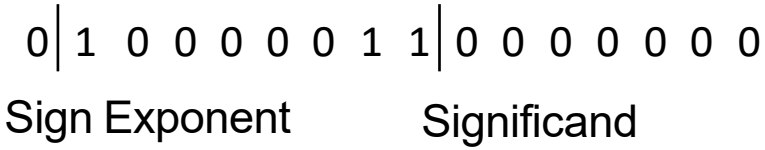
- Float16 (“half precision”)



Min (normal): ≈6.1×10^{-5}
Max: ≈6.55×10^{4}

- BFloat16

Same exponent as 32b, but only 7 significand bits. i.e., less detail



Min (normal): ≈1.18×10^{-38}
Max: ≈3.4×10^{38}

Quantization

- Compress high-precision numbers (e.g., float32) **into lower-precision representations, e.g., Int8**

- **This is a lossy process!**
- Quantization error: the difference between the original and compressed numbers

- **Simple approach: Round to nearest integer**
  - Q(x) = round(x): round(1.49) = round(0.51)

Quantization

- int8 operations are significantly faster than floating-point.
- Rather than rounding up, methods can be sensitive to the actual range of values that have to be represented (added expressivity within that range).
- Absmax Quantization: scales into values [-127, 127] :

$$X_{i8} = \text{round}\left(\frac{127 \cdot X_{f16}}{\max_{ij}(|X_{f16_{ij}}|)} \right)$$

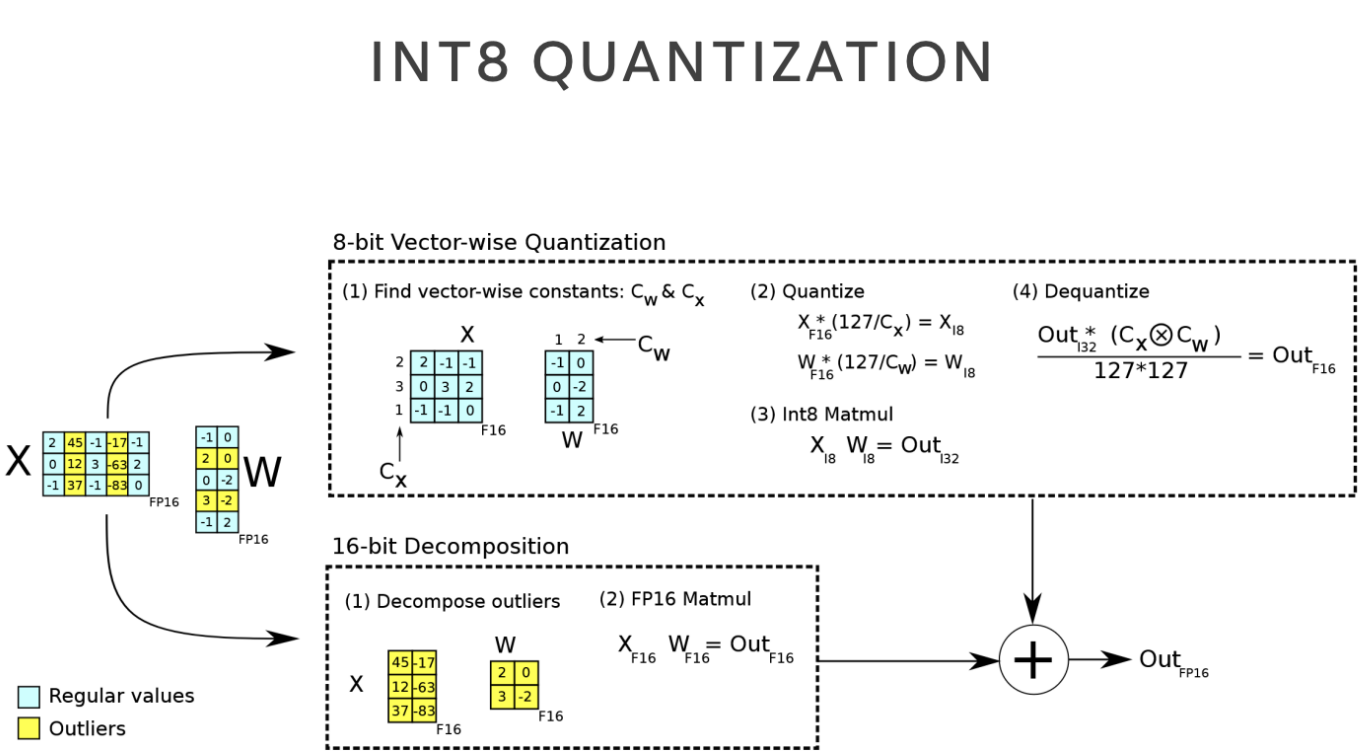
Original 16b FP matrix

Dealing with outliers

- Absmax normalization depends on min/max values. If the data contains outliers – significant quantization error.
- LLM.int8 : separate the rows/columns of matrices with outliers.
- Use fp16 representation for operations over outliers
- Other values- int8
- Combine the outputs and convert the result back into fp16

Dettmers et al., 2022

INT8 Quantization



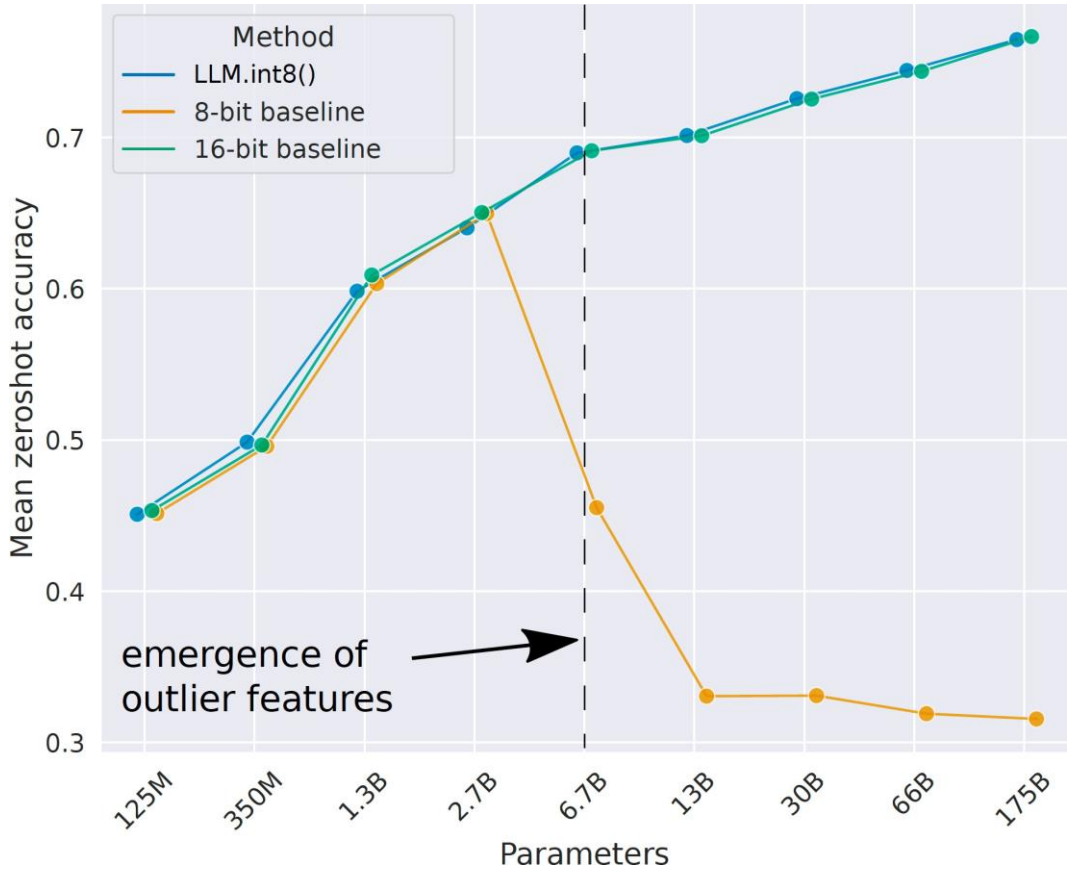
Dettmers et al., 2022

INT 8 Quantization

- The process does incur a higher computational cost

- Significantly less memory intensive: running inference with 175B parameter models on 80GB of VRAM.

INT 8 Quantization



INT 8 Quantization

Parameters	125M	1.3B	2.7B	6.7B	13B
32-bit Float	25.65	15.91	14.43	13.30	12.45
Int8 absmax	87.76	16.55	15.11	14.59	19.08
Int8 zeropoint	56.66	16.24	14.76	13.49	13.94
Int8 absmax row-wise	30.93	17.08	15.24	14.13	16.49
Int8 absmax vector-wise	35.84	16.82	14.98	14.13	16.48
Int8 zeropoint vector-wise	25.72	15.94	14.36	13.38	13.47
Int8 absmax row-wise + decomposition	30.76	16.19	14.65	13.25	12.46
Absmax LLM.int8() (vector-wise + decomp)	25.83	15.93	14.44	**13.24**	**12.45**
Zeropoint LLM.int8() (vector-wise + decomp)	**25.69**	**15.92**	**14.43**	**13.24**	**12.45**

Validation perplexity

Quantization-Aware Training (QAT)

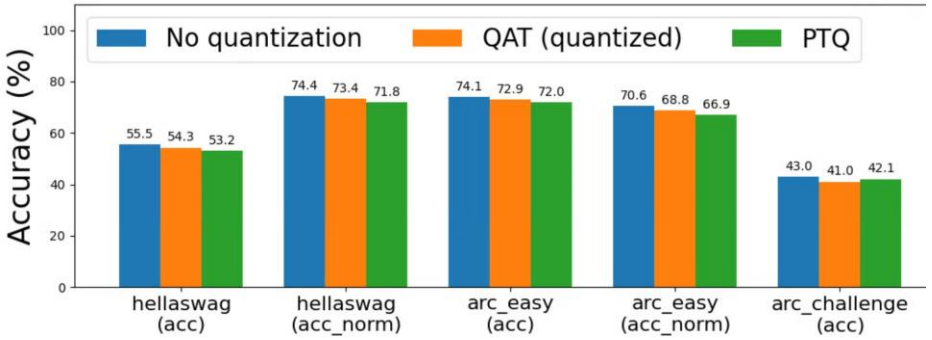
- Quantization after training compresses the model.
- It can lead to performance degradation, as the parameter values are not the same as those resulting in training.
- **Alternative**: Quantization aware training.

- Simulate quantization during forward passes.
- Model learns to be robust to quantization errors, instead of being quantized only after training.

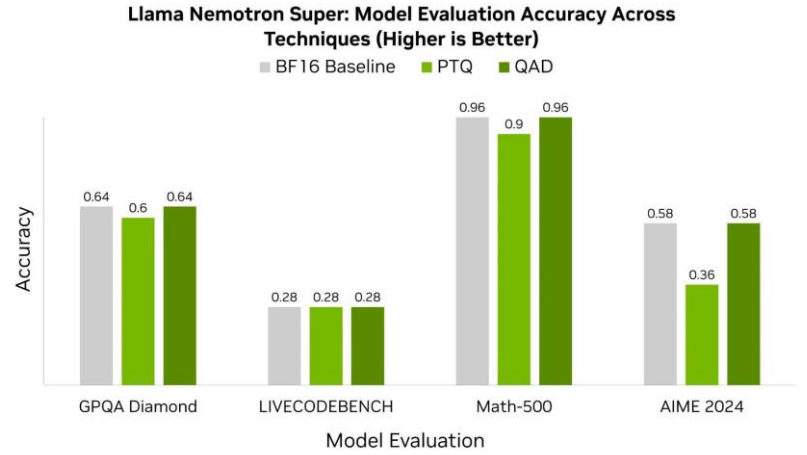
Quantization-Aware Training (QAT)

- Add a quantize-dequantize operation in model layers

```
# QAT: x_fq is still in float
# Fake quantize simulates the numerics of quantize + dequantize
x_fq = (x_float / scale + zp).round().clamp(qmin, qmax)
x_fq = (x_fq - zp) * scale
```



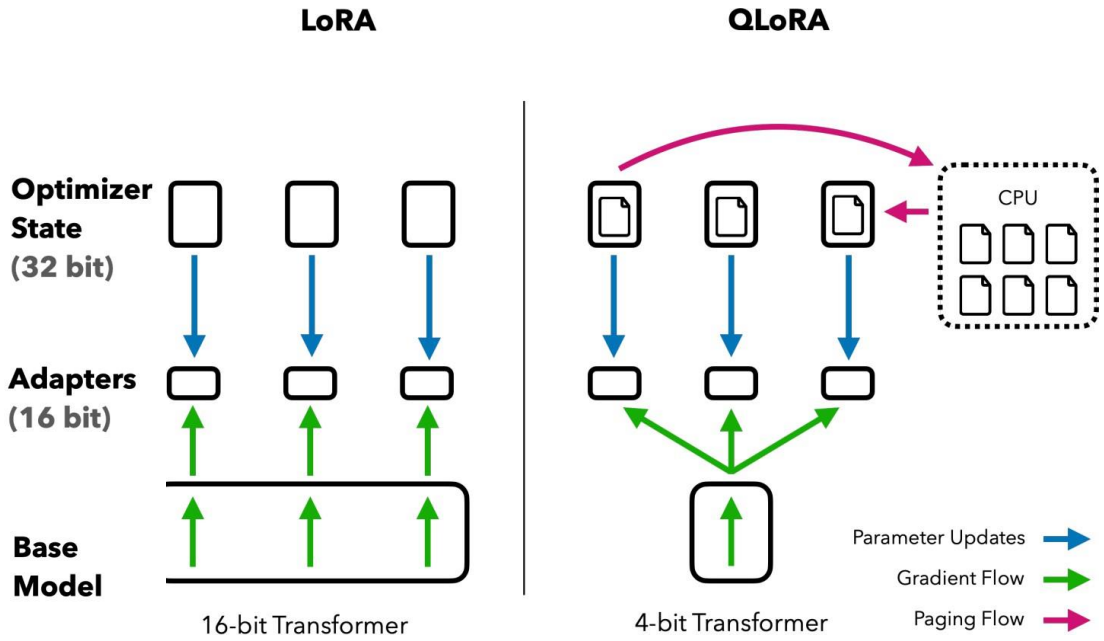
https://pytorch.org/blog/quantization-aware-training/



https://developer.nvidia.com/blog/how-quantization-aware-training-enables-low-precision-accuracy-recovery/

Fine-tuning: QLora (Quantized Low-Rank Adaptation)

- Basic idea: freeze and quantize model's weights, train LoRA adapters over the quantized model.



[Dettmers et al 2023]

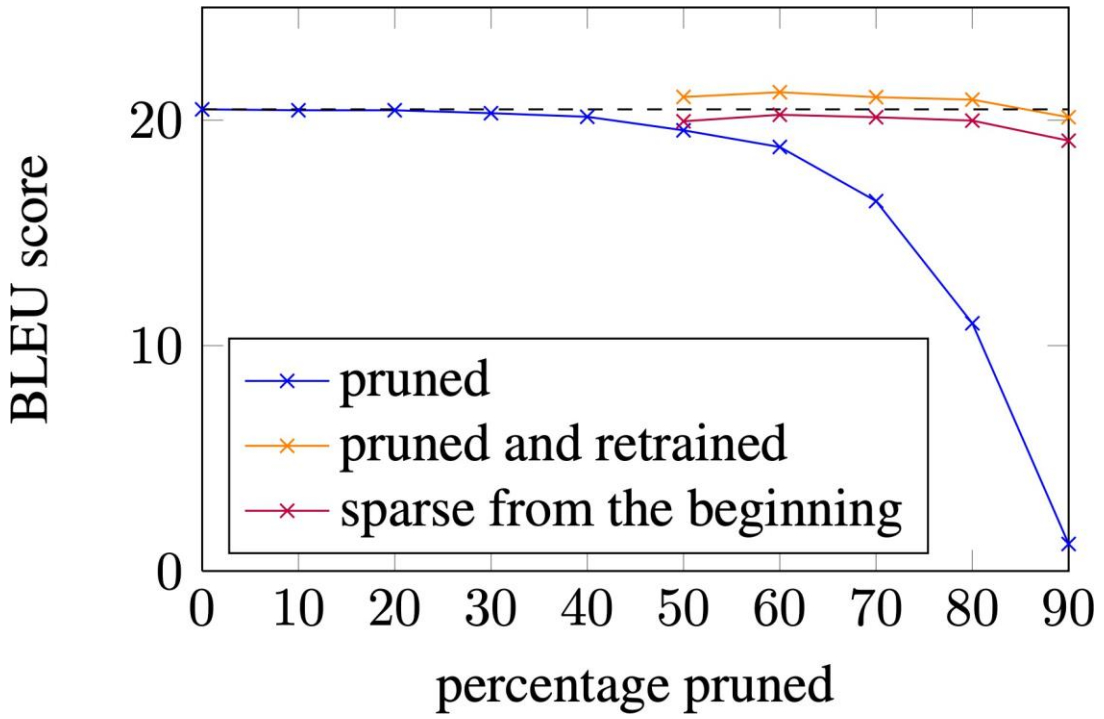
Pruning

- Prune parameters after training.

- Pruning vs. Quantization
- **Quantization**: same number of parameters, precision is reduced.
- **Pruning**: remove some parameters (i.e., set value to 0), remaining parameters untouched.

Magnitude Pruning

- Remove the bottom K% of parameters, closest values to 0



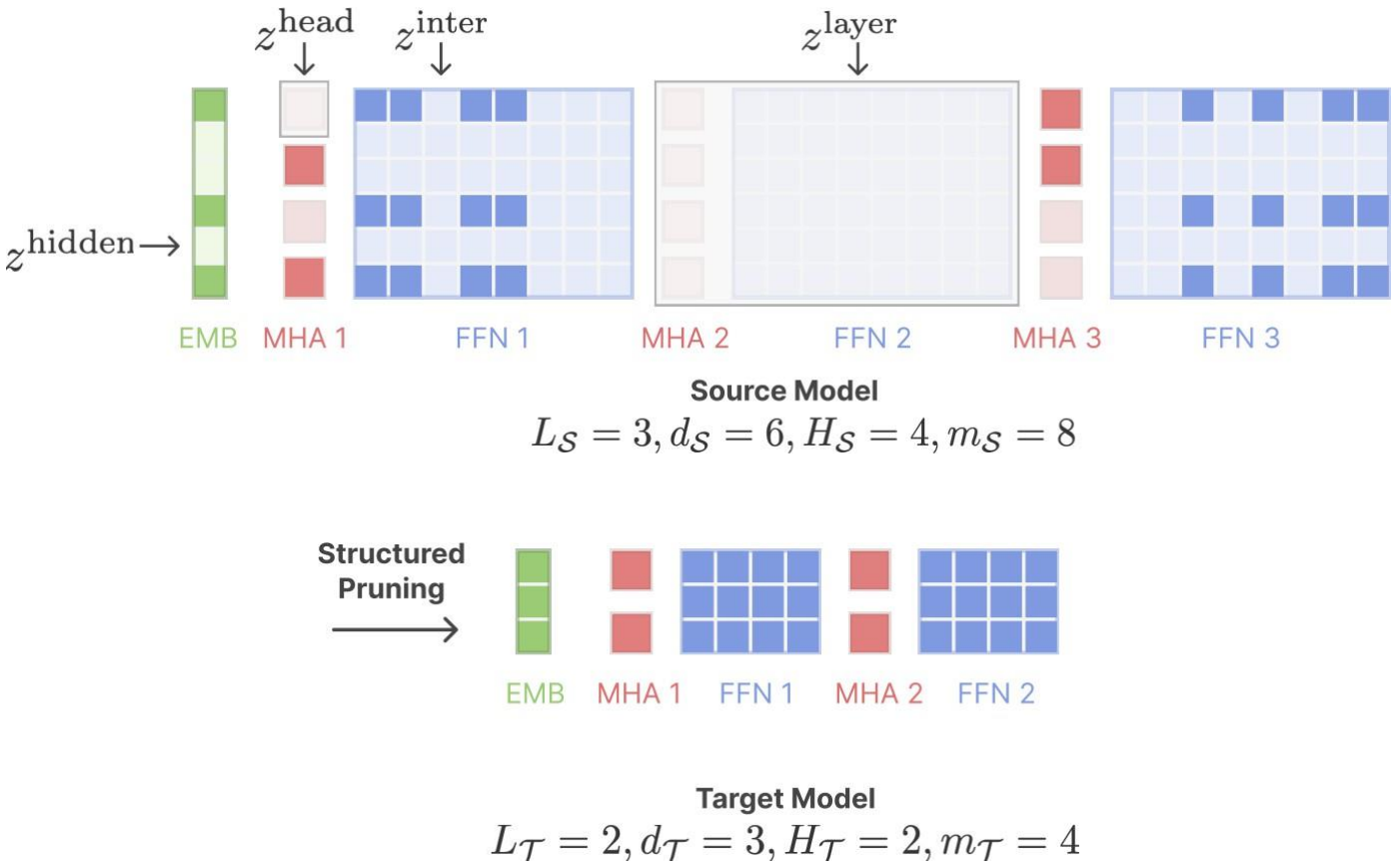
Han et al. 2015

Unstructured Pruning

- While removing almost 50% of parameters resulted in small performance loss, it's hard to directly translate it to faster computation or less memory requirements.

- Instead, removing complete modules (i.e., structured pruning) can be directly used.

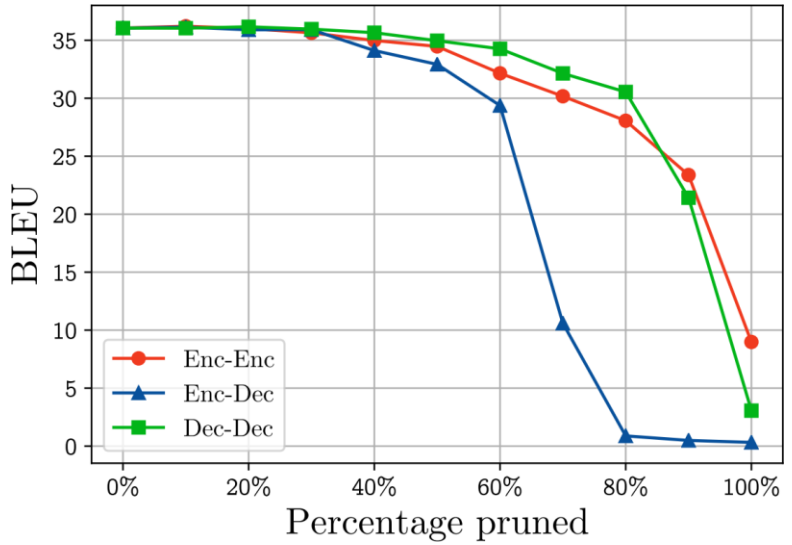
Structured Pruning



Xia et al. 2022

Structured Pruning

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



Michel and Neubig 2019

Structured Pruning

- Bonsai, a gradient-free structured pruning method that removes parts of large language models
- using only forward passes (no backprop needed). Enables efficient compression of LLMs by identifying and removing less important components while keeping performance high.

Table 3: Pruning can match highly optimized small, pretrained models while delivering better speedups. Bonsai-pruned LLaMA-2 achieves accuracy comparable to Phi-2 (a carefully engineered 3B model) with superior inference efficiency. Semi-structured pruning loses speedup benefits due to incompatible sparsity patterns.

Model	~Size	Fine-tune	PPL	Speedup
LlaMA-2	7B	✗	5.11	1×
Phi-2	3B	✓	8.69	1.24×
Wanda 2:4	3B	✗	10.52	1.14×
+ PPA		✓	8.34	0.75×
Bonsai	3B	✗	19.47	**1.58**×
+ PPA		✓	8.89	**1.58**×

Michel and Neubig 2019

Distillation

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

Distillation

- Not really model compression, rather training a simpler model based on the output from a stronger model.
- So.. Effectively a form of compression..
- Teacher model: strong model, providing labels (soft or hard).
- Student model: trained model

- An extension of old ideas in AI: one model used to train another (e.g., self-training, etc.)

Distillation

- Train a model based on “hard targets” and “soft targets”
  - Predicted labels vs. probability distribution over the output labels

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%

Hinton et al 2015

DistilBERT

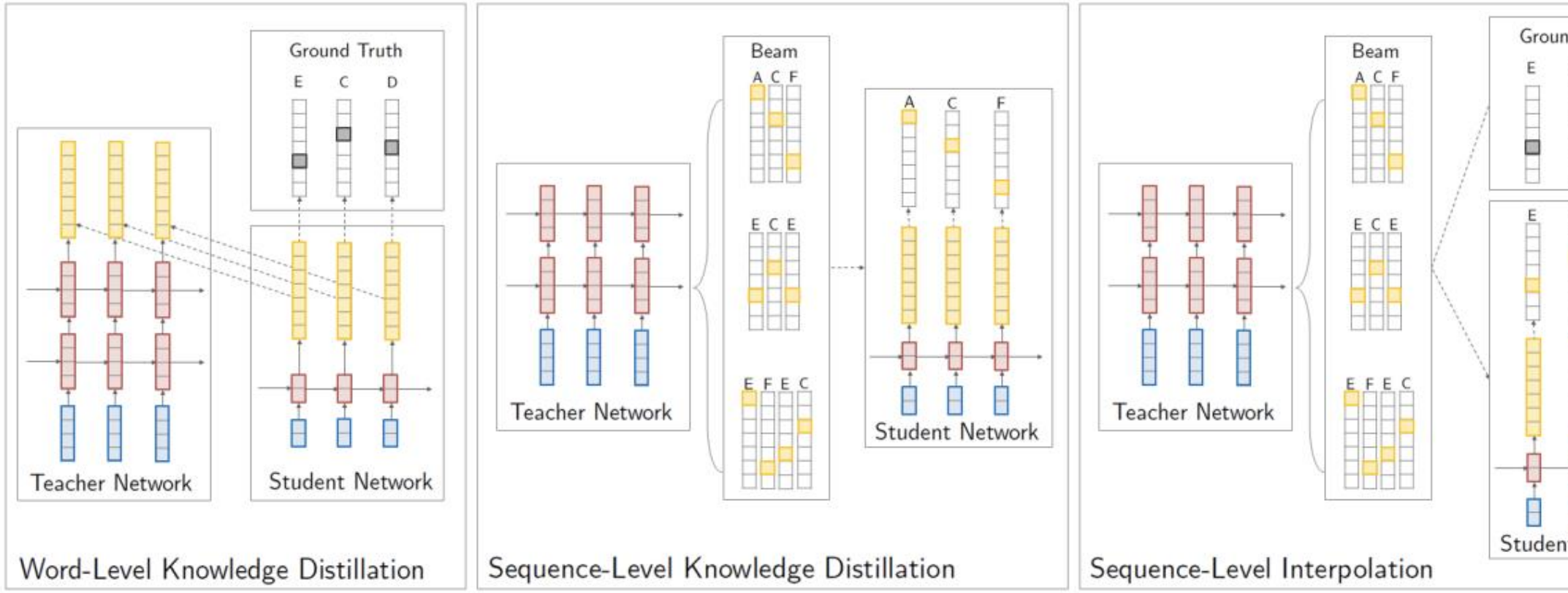
- Distill an encoder from BERT, reduce the number of layers to 6 (50%), while keeping most of the performance.

Table 1: **DistilBERT retains 97% of BERT performance.** Comparison on the dev sets of the GLUE benchmark. ELMo results as reported by the authors. BERT and DistilBERT results are the medians of 5 runs with different seeds.

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

Sanh et al 2019

Sequence-Level Distillation



Kim and Rush 2016

Sequence-Level Distillation

- Extend soft labels to sequences, combining word-level distillation and sequence-level distillation (maximize probability of the output generated by the teacher)

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

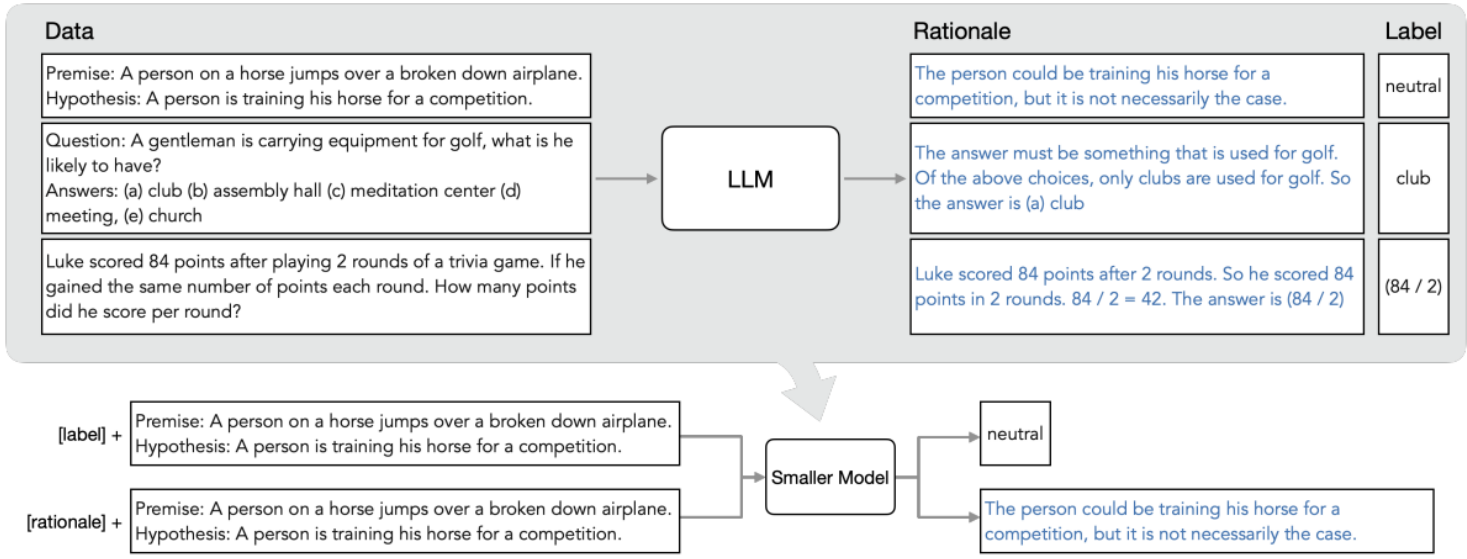
$$\begin{aligned}\mathcal{L}_{\text{SEQ-KD}} &\approx -\sum_{\mathbf{t}\in\mathcal{T}} \mathbb{1}\{\mathbf{t} = \hat{\mathbf{y}}\} \log p(\mathbf{t} \mid \mathbf{s}) \\ &= -\log p(\mathbf{t} = \hat{\mathbf{y}} \mid \mathbf{s})\end{aligned}$$

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

Kim and Rush 2016

Distilling step-by-step!

- Instead of only distilling final answers, also distills **LLM-generated rationales** to train smaller models.
- Extra “reasoning supervision” helps student models **learn the teacher’s process, not just the output**, helping it generalize better.



Hsieh et-al 2023

Distilling step-by-step!

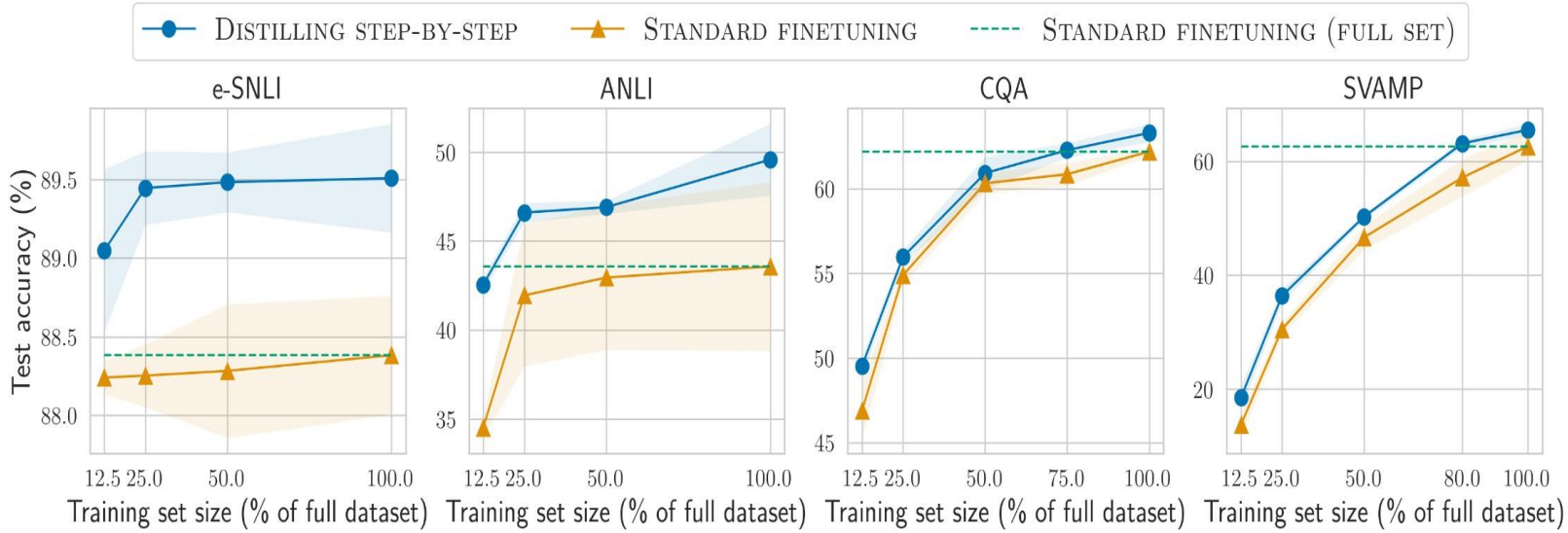
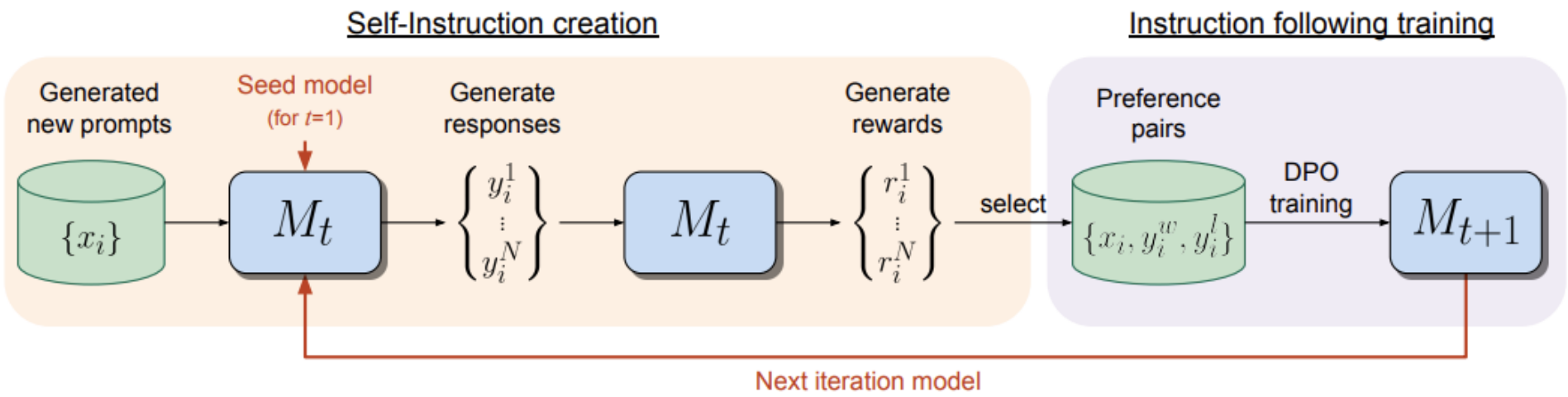


Figure 4: We compare Distilling step-by-step and Standard finetuning using 220M T5 models on varying sizes of human-labeled datasets. On all datasets, Distilling step-by-step is able to outperform Standard finetuning, trained on the full dataset, by using much less training examples (e.g., 12.5% of the full e-SNLI dataset).

Hsieh et-al 2023

Self-Rewarding Language Models

Improve by generating evaluation signals over the output.
Iteratively produce responses, critique them using an internal self-judgment mechanism



Yuan et-al 2024

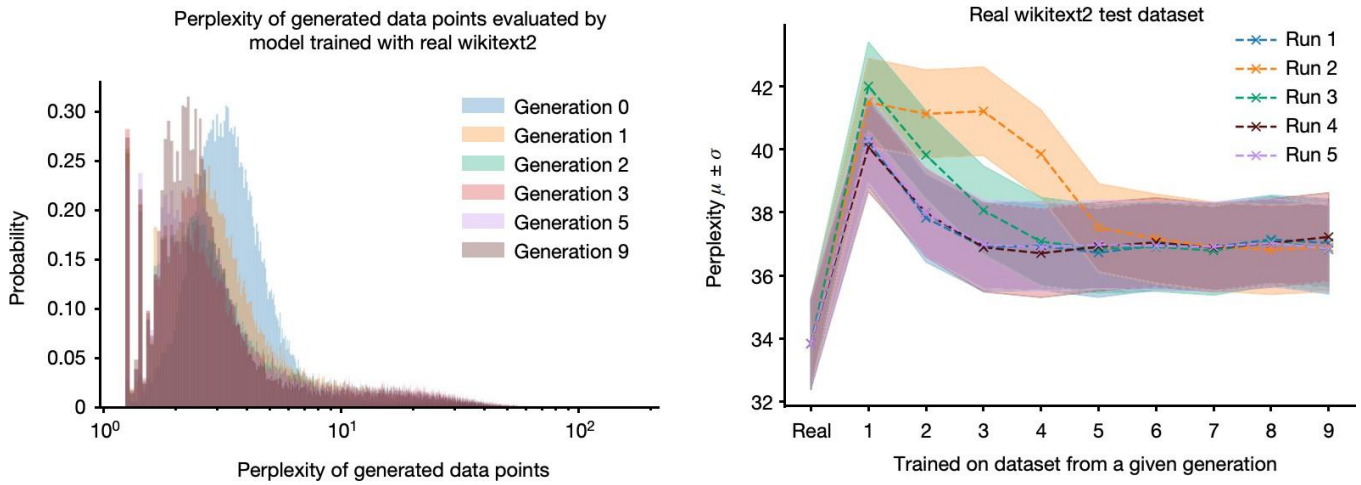
Training on Synthetic Data

- Alternative view: generate synthetic data for the model to train on
- Can be extended to generating more text.
- Gunasekar et al 2023: *Textbooks are all you need*. Careful use of little synthetic data: **textbooks**

Date	Model	Model size (Parameters)	Dataset size (Tokens)	HumanEval (Pass@1)	MBPP (Pass@1)
2021 Jul	Codex-300M [CTJ+21]	300M	100B	13.2%	-
2021 Jul	Codex-12B [CTJ+21]	12B	100B	28.8%	-
2022 Mar	CodeGen-Mono-350M [NPH+23]	350M	577B	12.8%	-
2022 Mar	CodeGen-Mono-16.1B [NPH+23]	16.1B	577B	29.3%	35.3%
2022 Apr	PaLM-Coder [CND+22]	540B	780B	35.9%	47.0%
2022 Sep	CodeGeeX [ZXZ+23]	13B	850B	22.9%	24.4%
2022 Nov	GPT-3.5 [Ope23]	175B	N.A.	47%	-
2022 Dec	SantaCoder [ALK+23]	1.1B	236B	14.0%	35.0%
2023 Mar	GPT-4 [Ope23]	N.A.	N.A.	67%	-
2023 Apr	Replit [Rep23]	2.7B	525B	21.9%	-
2023 Apr	Replit-Finetuned [Rep23]	2.7B	525B	30.5%	-
2023 May	CodeGen2-1B [NHX+23]	1B	N.A.	10.3%	-
2023 May	CodeGen2-7B [NHX+23]	7B	N.A.	19.1%	-
2023 May	StarCoder [LAZ+23]	15.5B	1T	33.6%	52.7%
2023 May	StarCoder-Prompted [LAZ+23]	15.5B	1T	40.8%	49.5%
2023 May	PaLM 2-S [ADF+23]	N.A.	N.A.	37.6%	50.0%
2023 May	CodeT5+ [WLG+23]	2B	52B	24.2%	-
2023 May	CodeT5+ [WLG+23]	16B	52B	30.9%	-
2023 May	InstructCodeT5+ [WLG+23]	16B	52B	35.0%	-
2023 Jun	WizardCoder [LXZ+23]	16B	1T	57.3%	51.8%
2023 Jun	**phi-1**	1.3B	7B	50.6%	55.5%

Training on Synthetic Data

- This idea has limitations!
- **Consider an internet dominated by "AI-slop", what would happen if we trained over that data?**
- Train over self-generated data, repeat over multiple "generations"



Shumailov et al 2024