CS310 Natural Language Processing
自然语言处理
Lecture 10 - Instruction Tuning

Instructor: Yang Xu

主讲人：徐炀

xuyang@sustech.edu.cn

Overview

- Motivation
- Instruction Tuning
- Parameter Efficient Fine-Tuning

What a pretrained LM can do and cannot?

- With properly designed decoding methods, an LM can respond to prompt like this:

Prompt: Teach me how to bake cake _____

Response: in a home oven

- This is grammatically sound, but not what a user want
- What we actually want:

Response: First, you make a dough with flour and water; …

Pretrained LM ≠ Assisting Users

- Reason: A pretrained model is not (yet) *aligned* with user **intent**

Prompt *Explain the moon landing to a 6 year old in a few sentences.*

Completion

GPT-3

```
Explain the theory of gravity to a 6 year old.

Explain the theory of relativity to a 6 year old in a few sentences.

Explain the big bang theory to a 6 year old.

Explain evolution to a 6 year old.
```

Not aligned

InstructGPT

```
People went to the moon, and they took pictures of what they saw, and sent them
back to the earth so we could all see them.
```

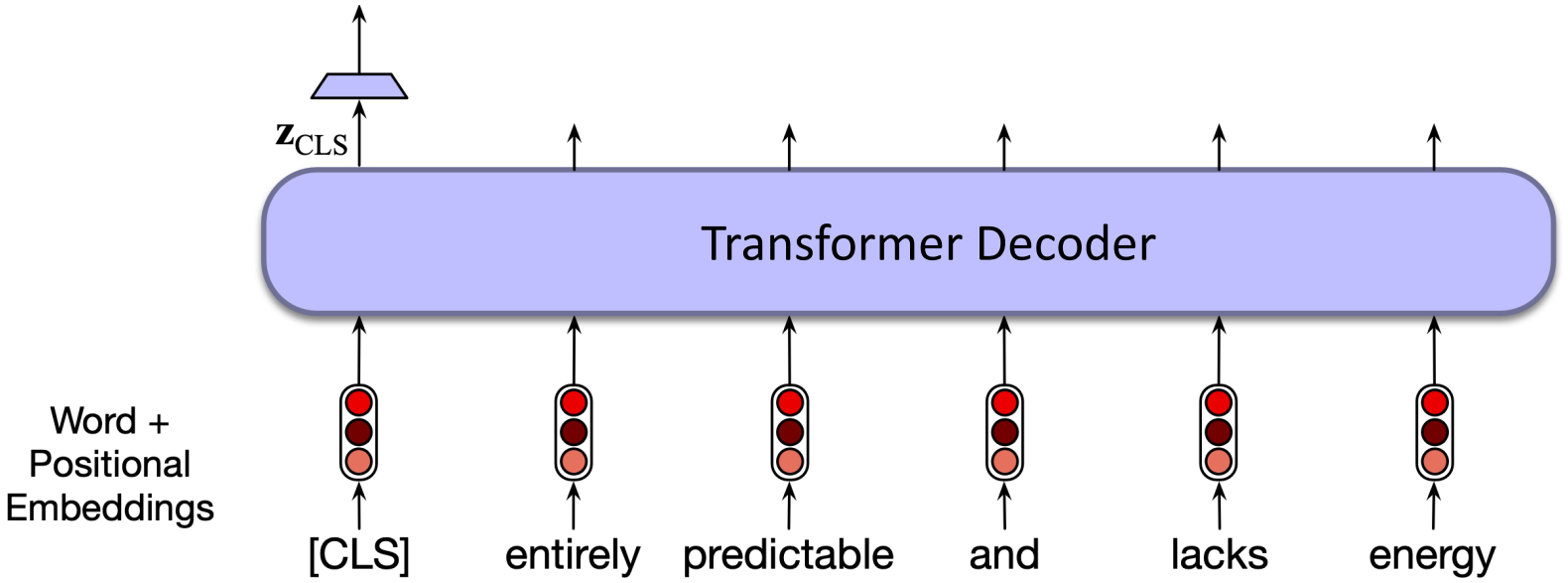
Aligned

Source: https://openai.com/research/instruction-following; Ouyang et al., 2022

How to enable LM to follow instructions?

- Fine-tuning to the rescue!

Fine-tune the pretrained model on specific **tasks** → consisting of pairs of instruction and output



Instruction tuning

- Collect **(instruction, output) pairs** across many tasks and finetune
  - A.k.a., supervised fine-tuning (SFT) or multi-task prompt tuning

- Where to get the data?
- formatting from three sources:
- 1. NLP task datasets
- 2. Daily chat data
- 3. Synthetic data

1. NLP tasks datasets

- A lot of useful datasets for NLP tasks exist in academia
- .. long before LLM emerged
- These data can be used for instruction-tuning, with minor *formatting*

Formatting by adding a task description:

"Please translate this sentence to Chinese."

Input: The sun raises every morning.

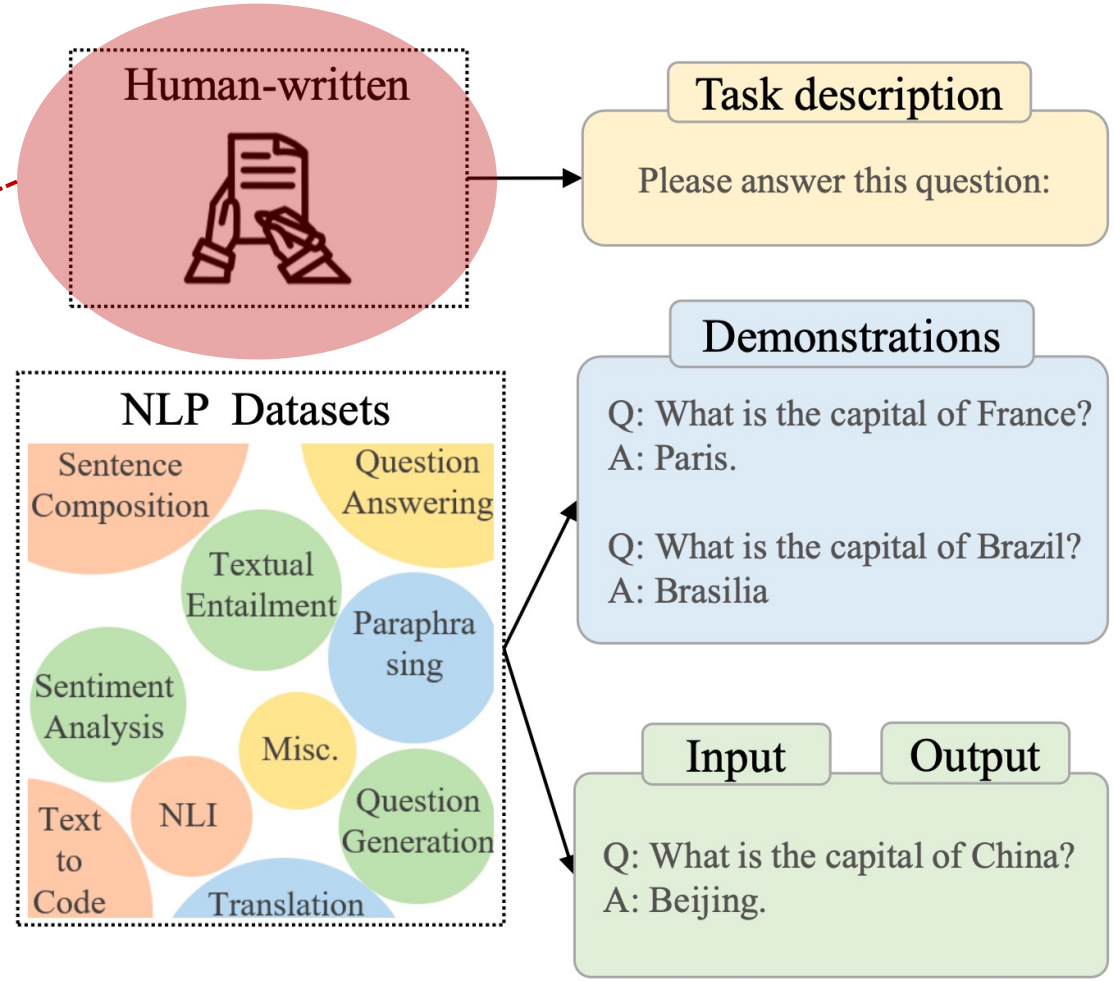
Output: 太 阳 每 天 照 常 升 起。

A sample from machine translation dataset

1. NLP tasks datasets

- General workflow

This part can be crowd sourced



Current instruction-tuning datasets

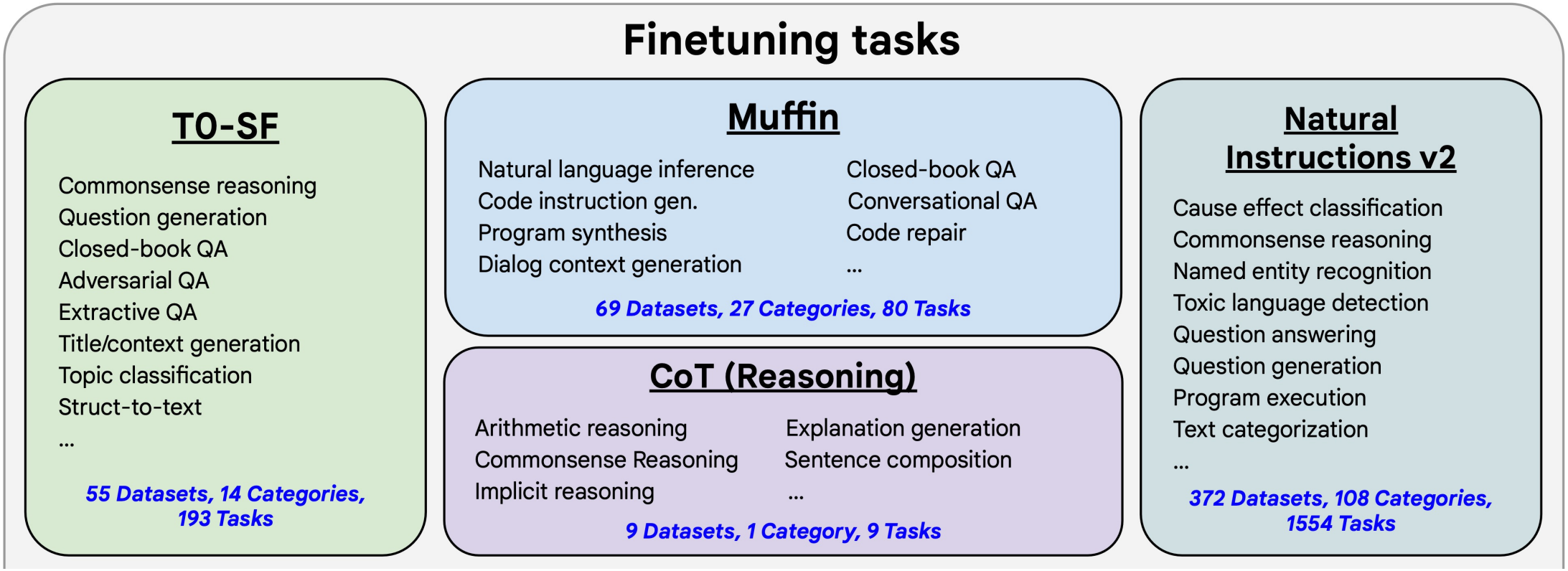
- Formatting from NLP tasks datasets

	类别	集合	时间	# 样本数量	来源
NLP	任务	Nat. Inst.	2021 年 04 月	193K	Allen Institute for AI
		FLAN	2021 年 09 月	4.4M	Google
		P3	2021 年 10 月	12.1M	BigScience
		Super Nat. Inst.	2022 年 04 月	5M	Allen Institute for AI
		MVPCorpus	2022 年 06 月	41M	Renmin University of China
		xP3	2022 年 11 月	81M	BigScience
		OIG	2023 年 03 月	43M	LAION-AI
		UnifedSKG	2022 年 03 月	812K	The University of Hong Kong

Source: https://llmbook-zh.github.io/LLMBook.pdf

Sample Tasks from FLAN

Source: Chung et al., 2024



FLAN-T5 used 473 datasets, 146 task categories, and **1836 total tasks**

Alternative way of formatting

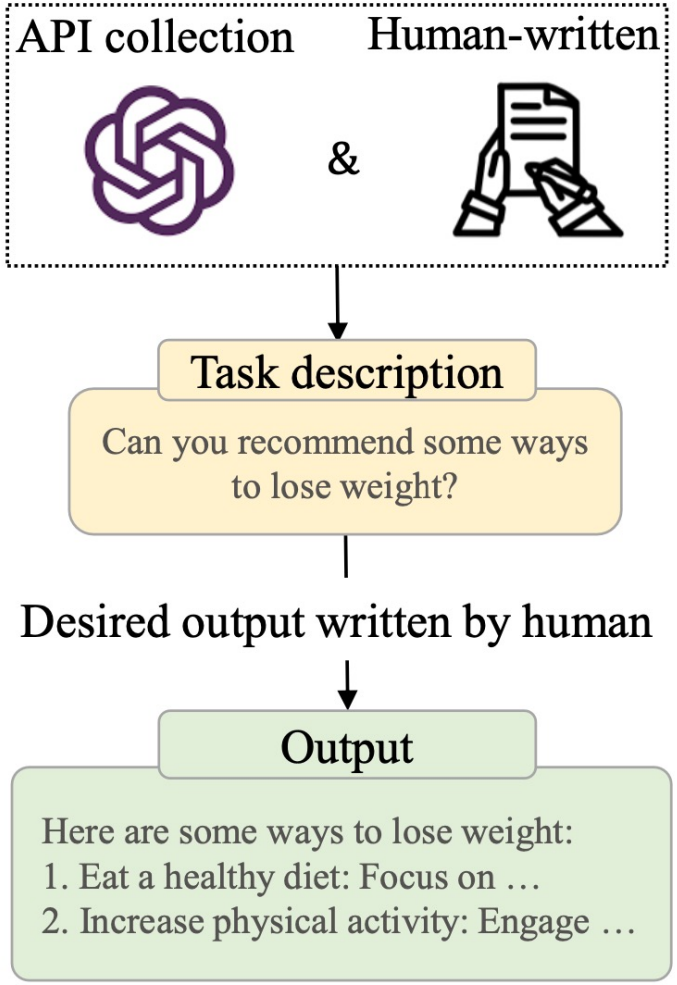
- Given a (question, answer) pair, create a new instance by adding this task description:
- ***"Please generate a question based on the answer:"***

2. Daily chat data

- NLP datasets lack instruction diversity, or mismatch real human needs

- **InstructGPT** (Ouyang et al., 2022) proposes to take the queries that real users have submitted to the OpenAI API as the task descriptions.
  - In addition:
  - OpenAI hired human labelers to compose instructions for real-life tasks:
    - open-ended generation, open QA, brain-storming, chatting, …
  - Let another group of labelers to respond to these instructions

  (OpenAI never published these data)



2. Daily chat data

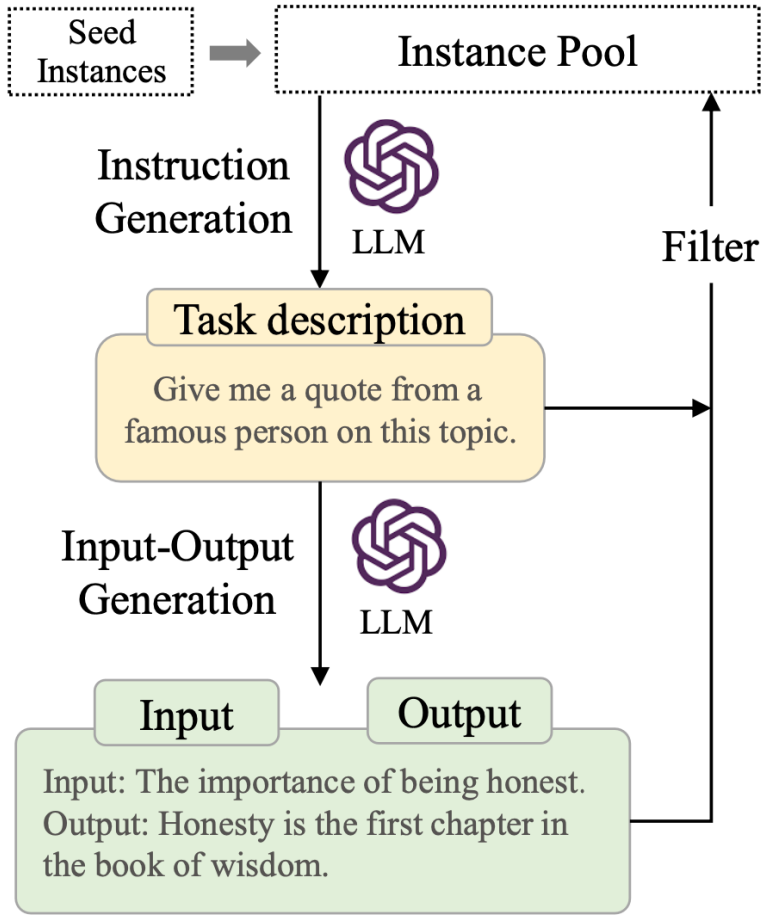
- **Databricks' Dolly** and **OpenAssistant** have open-sourced human labeled daily chat data
- **ShareGPT**: collected users' chat requests as input data, and then utilized ChatGPT or GPT-4 to generate responses as output data

Chat	对话	HH-RLHF	2022 年 04 月	160K	Anthropic
		HC3	2023 年 01 月	87K	SimpleAI
		ShareGPT	2023 年 03 月	90K	TechCrunch
		Dolly	2023 年 04 月	15K	Databricks
		OpenAssistant	2023 年 04 月	161K	LAION-AI
		InstructWild v2	2023 年 04 月	111K	National University of Singapore
		LIMA	2023 年 06 月	1K	Meta AI

Source: https://llmbook-zh.github.io/LLMBook.pdf

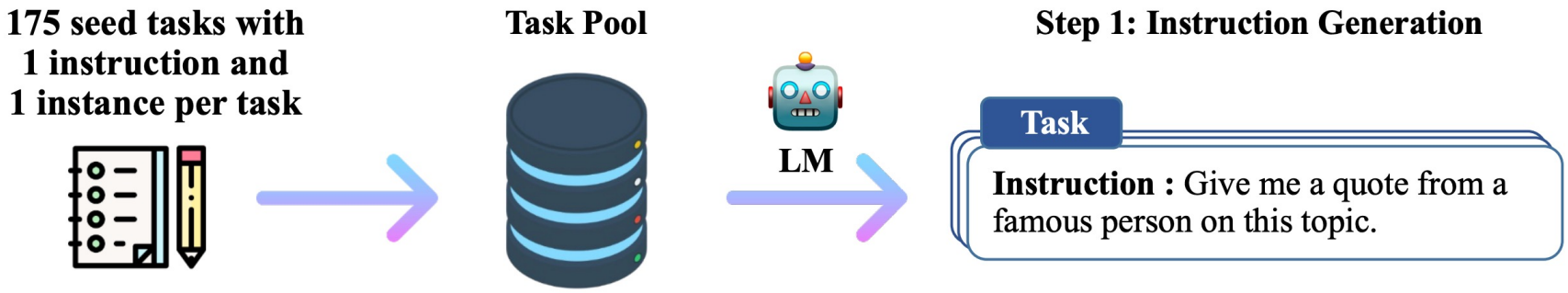
3. Synthetic data

- To reduce the burden of human annotation or manual collection
- people feed existing instances into LLMs to synthesize diverse task descriptions and instances
- Examples:
- Self-Instruct, Evol-Instruct



3. Synthetic data

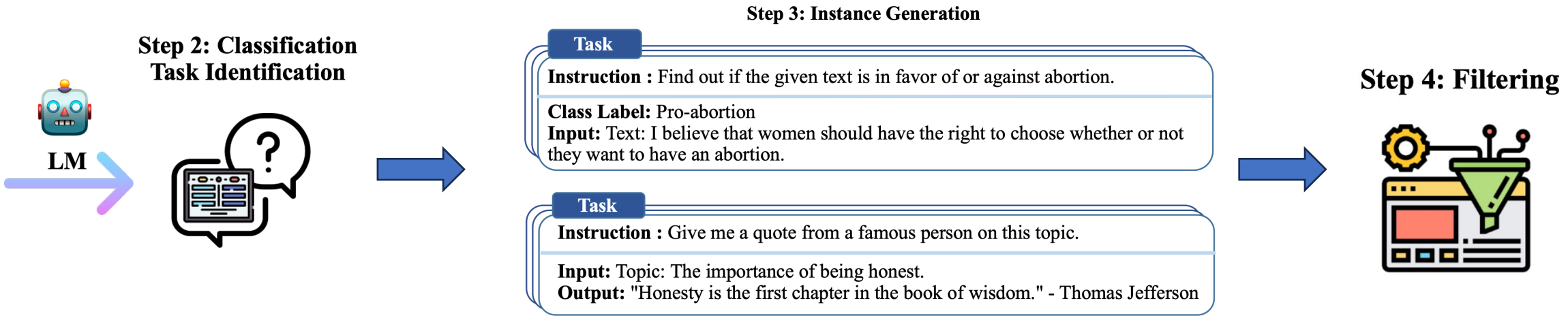
- **Self-Instruct**: Use LLMs (e.g., ChatGPT) to generate instructions
- Phase 1: Starts with a small seed set of tasks as the task pool.
- Randomly sample tasks from the pool, and use them to prompt an off-the-shelf LM to generate both new instructions and corresponding instances



Source: Self-Instruct paper, https://arxiv.org/pdf/2212.10560

3. Synthetic data

- **Self-Instruct**: Use LLMs (e.g., ChatGPT) to generate instructions
- Phase 2: After the quality and diversity filtering, newly generated instances would be added into the task pool.



Source: Self-Instruct paper, https://arxiv.org/pdf/2212.10560

3. Synthetic data

- Self-Instruct uses GPT-3 to generate 52K instruction dataset
- Following Self-Instruct's pipeline, Stanford Alpaca project generates a 52K dataset using the `text-davinci-003` model,
- based on which, a 7B LLaMA model is fine-tuned, resulting in Alpaca-7B.

Source: https://github.com/tatsu-lab/stanford_alpaca

3. Synthetic data

Synthetic	合成	Self-Instruct	2022 年 12 月	82K	University of Washington
		Alpaca	2023 年 03 月	52K	Standford
		Guanaco	2023 年 03 月	535K	-
		Baize	2023 年 04 月	158K	University of California, San Diego
		Belle	2023 年 04 月	1.5M	LianjiaTech
		Alpaca-GPT4	2023 年 04 月	52K	Microsoft
		Evol-Instruct	2023 年 06 月	52K	Microsoft
		UltraChat	2023 年 06 月	675K	Tsinghua University

Source: https://llmbook-zh.github.io/LLMBook.pdf

Training strategies for instruction fine-tuning

- **Loss function**: sequence-to-sequence loss, with loss computed only on the output sequence, not on the input sequence.
- **Batch size** and **learning rate**: relatively small, because pretraining already learns the parameters well
- Use **multi-turn** dialogue data efficiently.

Example of instruction tuning data

- Alpaca 52k data

```
{
    "instruction": "What are the three primary colors?",
    "input": "",
    "output": "The three primary colors are red, blue, and yellow."
},
```

Case 1: The input is empty

Example of instruction tuning data

- Alpaca 52k data

```
{
    "instruction": "Evaluate this sentence for spelling and grammar mistakes",
    "input": "He finnished his meal and left the resturant",
    "output": "He finished his meal and left the restaurant."
},
```

Case 2: The input is not empty

Example of instruction tuning data

- Use the template for fine-tuning

```
instruction_template = "\n### Instruction:\n"
response_template = "\n### Output:\n"
```

For Case 1, empty input

```
    "prompt_no_input": (
        "Below is an instruction that describes a task. " +
        "Write a response that appropriately completes the request." +
        instruction_template + "{instruction}" +
        response_template
    ),
}
```

Example of instruction tuning data

- Use the template for fine-tuning

```
instruction_template = "\n### Instruction:\n"
response_template = "\n### Output:\n"
```

For Case 2, non-empty input

```
format_template = {
    "prompt_input": (
        "Below is an instruction that describes a task, paired with an input that
        provides further context. " +
        "Write a response that appropriately completes the request." +
        instruction_template + "{instruction}" +
        "{input}" + response_template
    ),
```

Example of instruction tuning data

- Full template

```
Below is an instruction that describes a task. Write a response that
appropriately completes the request.
### Instruction:
What are the three primary colors?
### Output:

The three primary colors are red, blue, and yellow.
```

Example of instruction tuning data

- Full template

```
Below is an instruction that describes a task, paired with an input
that provides further context. Write a response that appropriately
completes the request.
### Instruction:
Evaluate this sentence for spelling and grammar mistakes
He finnished his meal and left the resturant
### Output:

He finished his meal and left the restaurant.
```

Limitations of Instruction Tuning

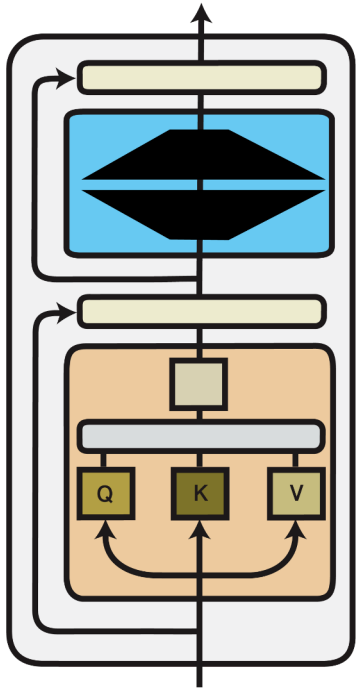
- Obvious limit: **Expensive** to collect task data
- Subtler limits:
- 1. Tasks like open-ended creative writing have no correct answer
- 2. Even after being fine-tuned, there is mismatch between LM's object and the **human preferences**
- 3. fine-tuning penalizes all token-level mistakes equally, but some errors are worse than others

- **Question**: Can we explicitly ask the model to satisfy human preferences?

Overview

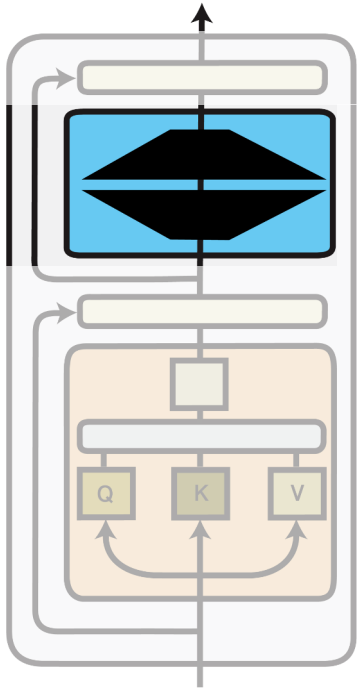
- Motivation
- Instruction Tuning
- **Parameter Efficient Fine-Tuning**
- Alignment to Human

What is Parameter Efficient Fine-Tuning (PEFT)?



Full Fine-tuning

Update **all model parameters**



Parameter-efficient Fine-tuning

Update a **small subset** of parameters

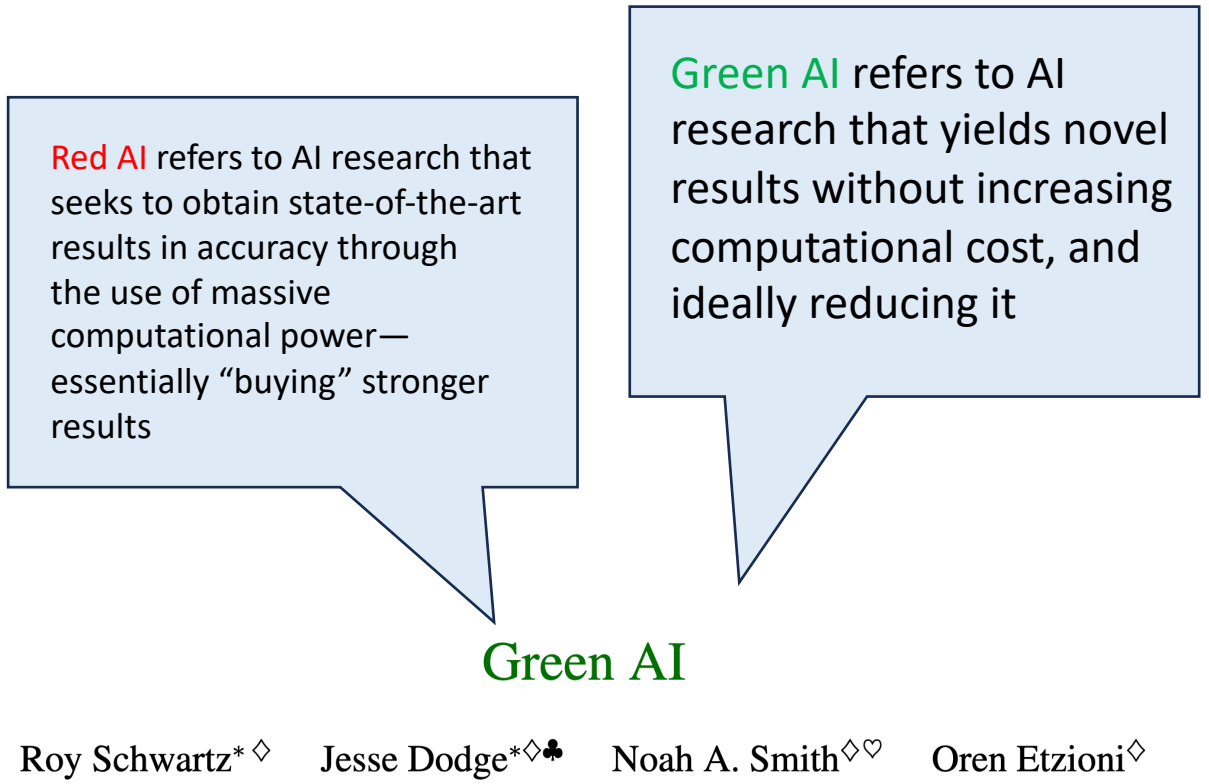
Why fine-tuning *only some* parameters?

- Fine-tuning all parameters is impractical with large models
- SOTA models are massively over-parameterized: PEFT matches performance of full fine-tuning

Slide credit: Modular and Parameter-efficient Fine-tuning for NLP Models EMNLP 2022 Tutorial https://www.modulardeeplearning.com/

Why Parameter Efficient Fine-Tuning (PEFT)?

- Emphasis on accuracy over efficiency in current AI paradigm
- Hidden environmental costs of training (and fine tuning) LLMs
- As costs of training go up, AI development becomes concentrated in well-funded organizations, especially in industry



Revisit Full Fine-Tuning

- Assume we have a pretrained model language model $P_\phi(y|x)$
- Goal: to fine-tune this model to downstream tasks (e.g., summarization, translation, …)
  - On training dataset: $\{(x_i, y_i)\}_{i=1,\ldots,N}$

- During full fine-tuning, we update ϕ_0 to $\phi_0 + \Delta\phi$, where $\Delta\phi$ is the gradient, to maximize the log probability:

$$\max_{\phi} \sum_{(x,y)} \sum_{t=1}^{|y|} \log\left(P_\phi(y_t|x, y_{<t})\right)$$

Cost of full fine-tuning and how to reduce it

- For each downstream task, need to learn a different $\Delta\phi$
  - $|\Delta\phi| = |\phi_0|$, that it, $|\boldsymbol{\phi_0}| = \mathbf{175\ billion}$ for GPT-3!
  - Expensive for storing and deployment for many tasks

- **Key Idea**: Encode the task-specific gradient $\Delta\phi = \Delta\phi(\Theta)$ by a much smaller set of parameters Θ, $\Theta \ll |\phi_0|$
- The fine-tuning task becomes optimizing over Θ:

$$\max_{\Theta} \sum_{(x,y)} \sum_{t=1}^{|y|} \log\left(P_{\Delta\phi=\Delta\phi(\Theta)}(y_t|x, y_{<t})\right)$$

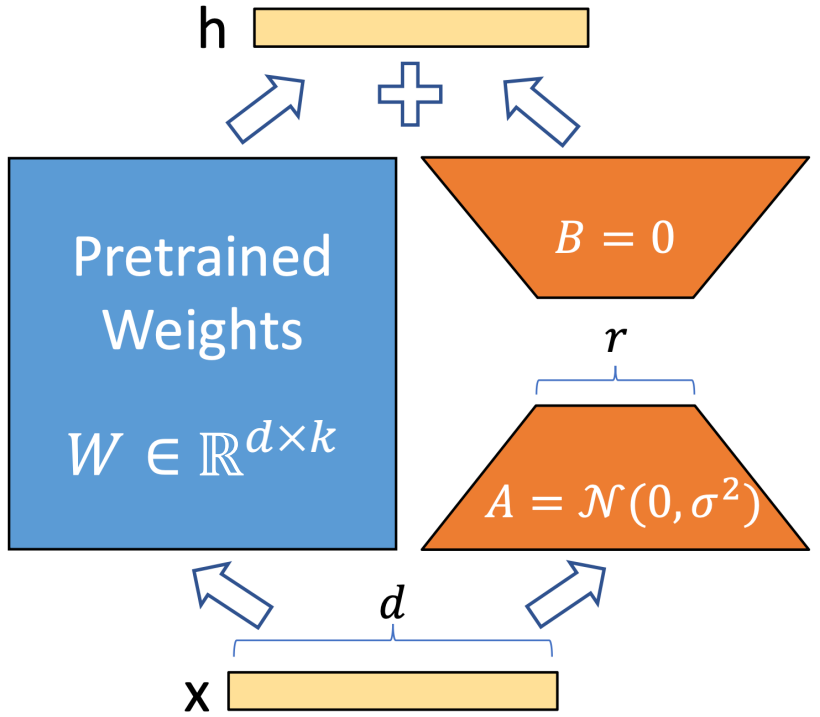
Low Rank Adaptation (LoRA)

Hu et al., 2021

$\Delta\phi(\Theta)$ in last slide

- LoRA: freezes the pre-trained model weights and injects trainable **rank decomposition** matrices into each layer
- For a pretrained weight $\mathbb{R}^{d\times k}$
- Constrain its update with low-rank decomposition:
  $$W_0 + \Delta W = W_0 + BA$$
  where $B \in \mathbb{R}^{d\times r}, A \in \mathbb{R}^{r\times d}, r \ll \min(d, r)$
- Only A and B contain **trainable** parameters
- Random Gaussian initialization for A
  zero for B, so $\Delta W = BA = 0$ at the beginning

$$h = W_0x + \Delta x = W_0x + BAx$$

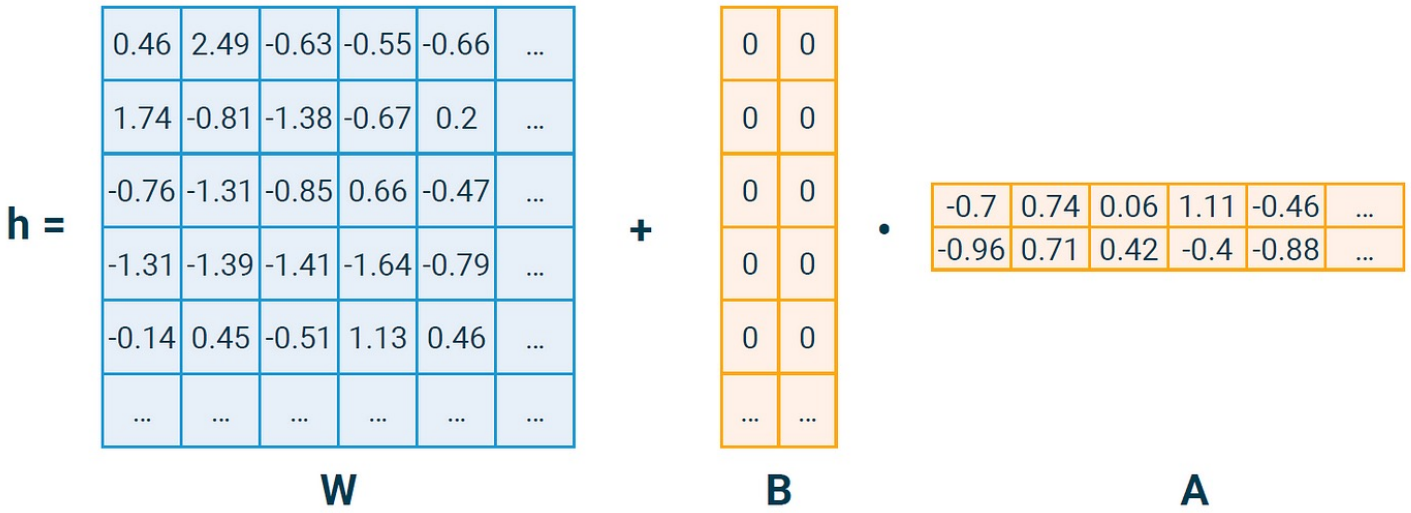


LoRA example

$$h = W_0 x + \Delta x = W_0 x + BAx$$

Check more impl. here:

- https://github.com/microsoft/LoRA
- https://github.com/cccntu/minLoRA



"Non-LoRA" query result

LoRA query result

```
# Initialize LoRA matrices for query and value
self.lora_query_matrix_B = nn.Parameter(torch.zeros(d, r))
self.lora_query_matrix_A = nn.Parameter(torch.randn(r, d))
self.lora_value_matrix_B = nn.Parameter(torch.zeros(d, r))
self.lora_value_matrix_A = nn.Parameter(torch.randn(r, d))
```

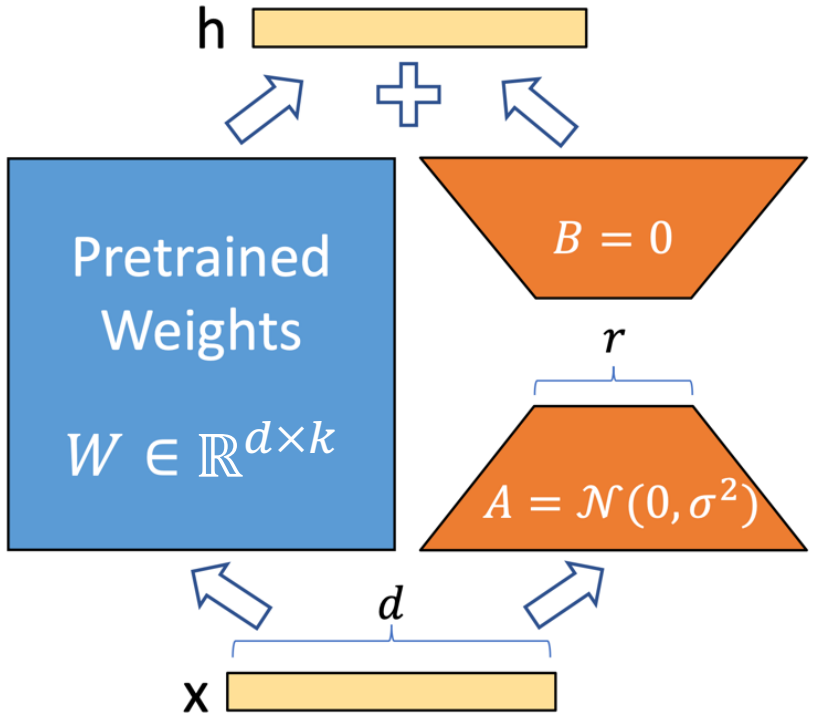
```
def lora_query(self, x):
    lora_query_weights =
        torch.matmul(self.lora_query_matrix_B,
                     self.lora_query_matrix_A)
    return self.query(x) + F.linear(x,
lora_query_weights)
```

Source: https://towardsdatascience.com/implementing-lora-from-scratch-20f838b046f1

LoRA Propertities

- As one increases r, LoRA generalizes to full fine-tuning
- **No additional inference latency**:
  For some task T_1,
  compute and store $W = W_0 + BA$
  when switching to a another task T_2,
  recover W_0 by subtracting BA and adding a different $B'A'$
- LoRA is applied to attention matrices:
  W_q and W_v

$$h = W_0x + \Delta x = W_0x + BAx$$



LoRA Performance

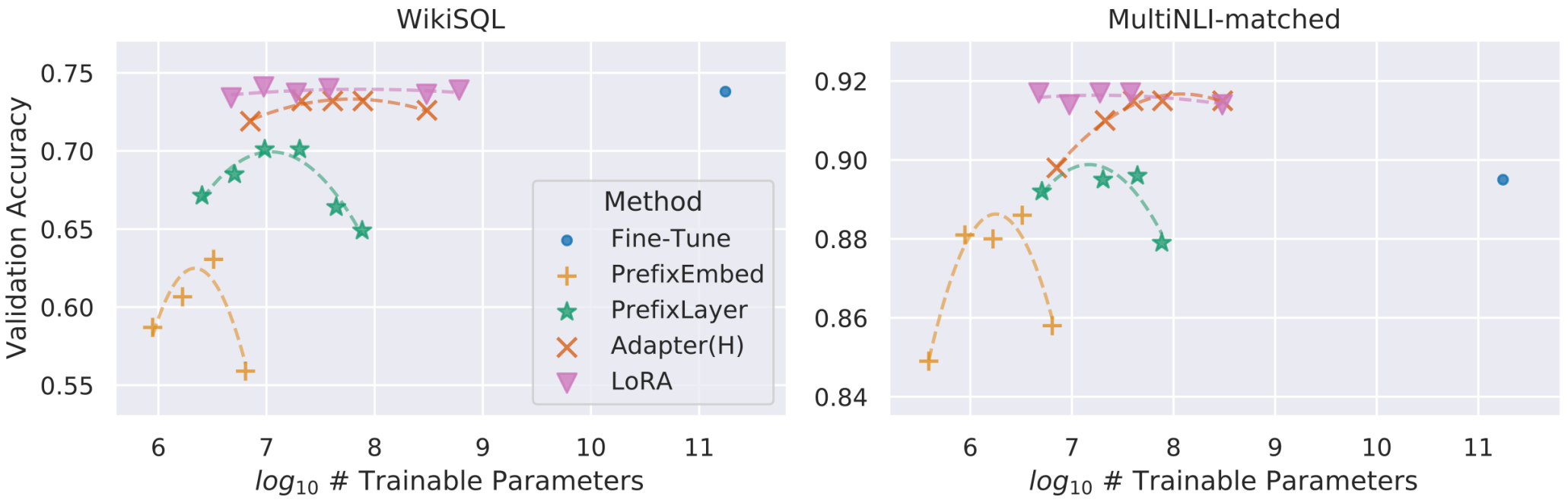
Hu et al., 2021

Model&Method	# Trainable Parameters	WikiSQL Acc. (%)	MNLI-m Acc. (%)	SAMSum R1/R2/RL
GPT-3 (FT)	175,255.8M	**73.8**	89.5	52.0/28.0/44.5
GPT-3 (BitFit)	14.2M	71.3	91.0	51.3/27.4/43.5
GPT-3 (PreEmbed)	3.2M	63.1	88.6	48.3/24.2/40.5
GPT-3 (PreLayer)	20.2M	70.1	89.5	50.8/27.3/43.5
GPT-3 ($\text{Adapter}^{\text{H}}$)	7.1M	71.9	89.8	53.0/28.9/44.8
GPT-3 ($\text{Adapter}^{\text{H}}$)	40.1M	73.2	**91.5**	53.2/29.0/45.1
GPT-3 (LoRA)	4.7M	73.4	**91.7**	**53.8/29.8/45.9**
GPT-3 (LoRA)	37.7M	**74.0**	**91.6**	53.4/29.2/45.1

LoRA performs better than prior approaches, including full fine-tuning with much fewer params!

LoRA Performance

Hu et al., 2021



LoRA exhibits better scalability and task performance

LoRA details

Hu et al., 2021

- Which weight matrices to apply LoRA to?

	# of Trainable Parameters = 18M						
Weight Type	W_q	W_k	W_v	W_o	W_q, W_k	W_q, W_v	W_q, W_k, W_v, W_o
Rank r	8	8	8	8	4	4	2
WikiSQL ($\pm 0.5\%$)	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI ($\pm 0.1\%$)	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

Applying to W_q and W_v gives the best performance

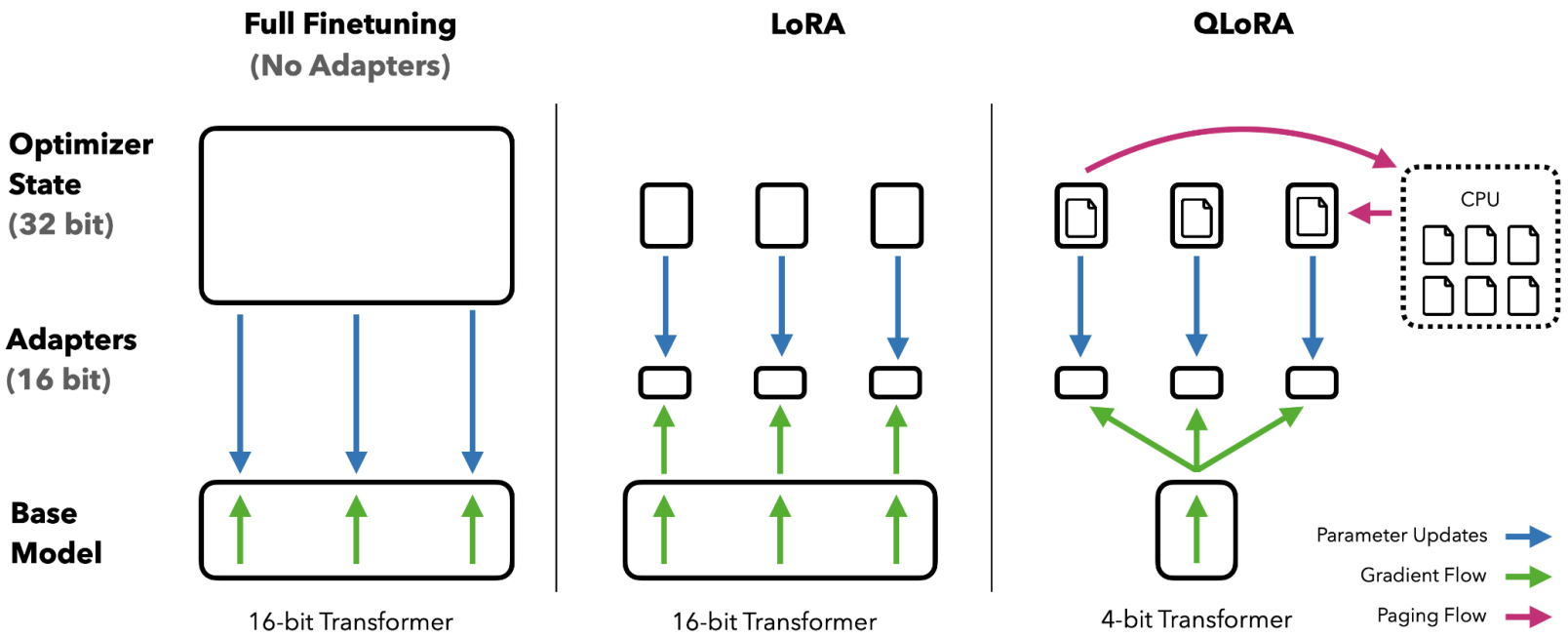
- What is the optimal rank r?

	Weight Type	$r = 1$	$r = 2$	$r = 4$	$r = 8$	$r = 64$
WikiSQL($\pm 0.5\%$)	W_q	68.8	69.6	70.5	70.4	70.0
	W_q, W_v	73.4	73.3	73.7	73.8	73.5
	W_q, W_k, W_v, W_o	74.1	73.7	74.0	74.0	73.9
MultiNLI ($\pm 0.1\%$)	W_q	90.7	90.9	91.1	90.7	90.7
	W_q, W_v	91.3	91.4	91.3	91.6	91.4
	W_q, W_k, W_v, W_o	91.2	91.7	91.7	91.5	91.4

Already performs decently with very small r

From LoRA to QLoRA

- **QLoRA**: improves over LoRA by quantizing the transformer model to 4- bit precision and using paged optimizer to handle memory spikes
- 4-bit NormalFloat (NF4), a new data type that is information theoretically optimal for normally distributed weights



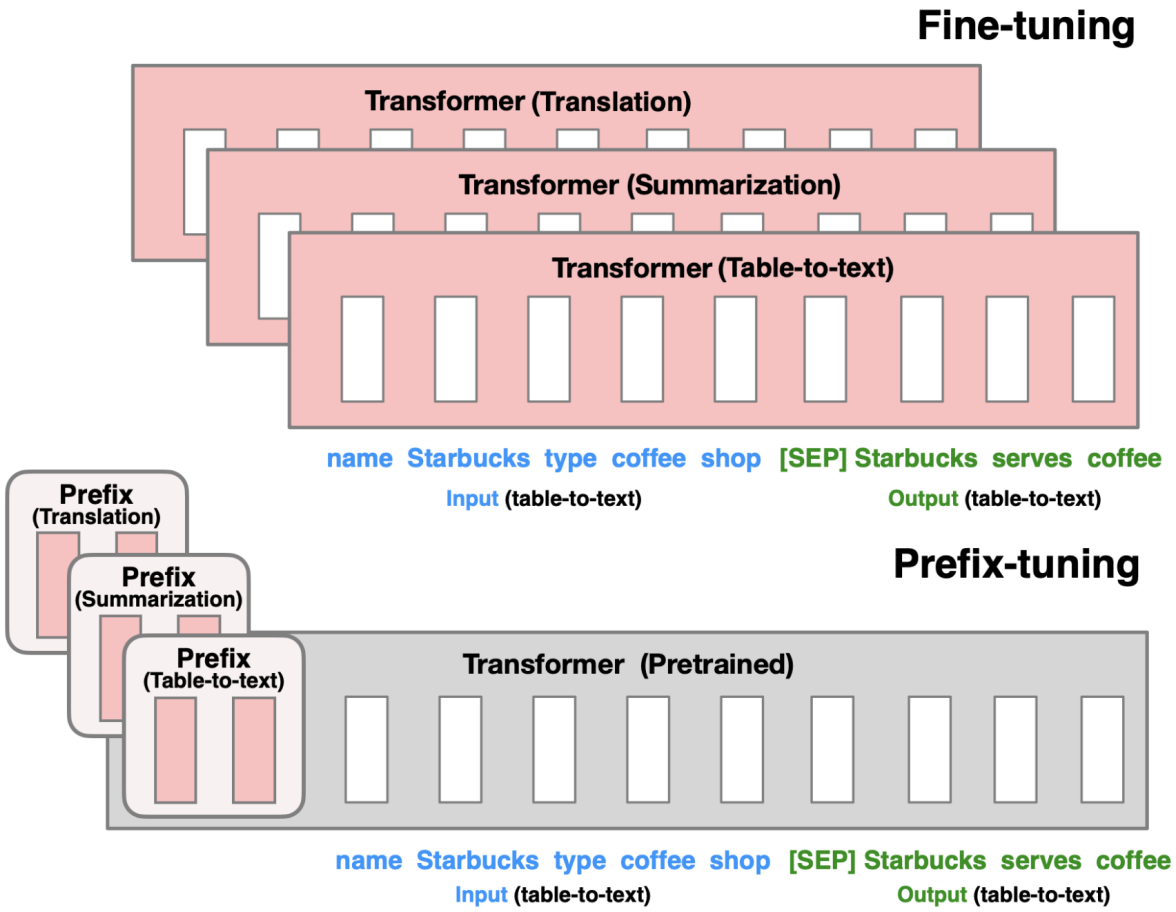
Other PEFT Techniques

- Prefix tuning
- Prompt tuning
- Adapter

Prefix Tuning

Li and Liang, 2021

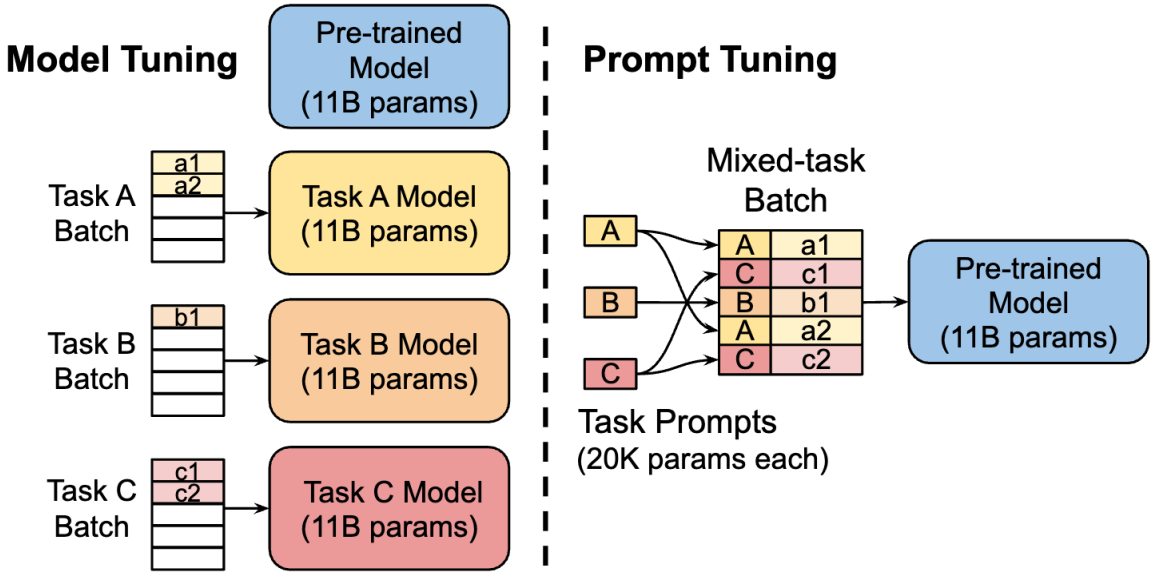
- Prefix-Tuning adds a prefix of parameters (*learnable*), and freezes all pretrained parameters
- Optimizes a sequence of continuous **task-specific vectors** -- called ***prefix***
- Consequently, only need to store the prefix for each task, making prefix-tuning modular and space-efficient



Prompt Tuning

Lester et al., 2021; Qin et al., 2021

- "**Soft prompt**": Unlike the discrete text prompts, soft prompts are learned through backpropagation and can be tuned to incorporate signals from any number of labeled examples

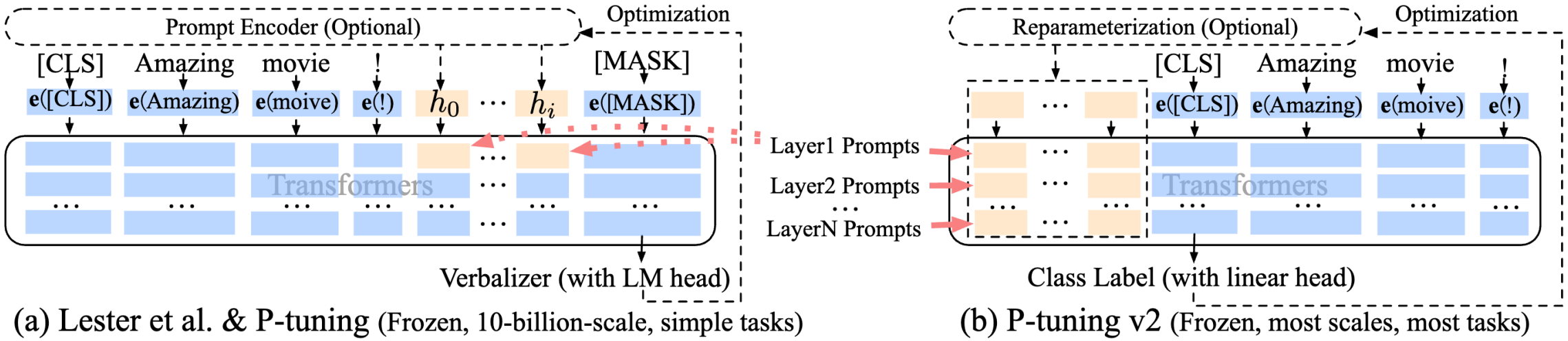


Prompt tuning requires storing a small task-specific prompt for each task, and enables mixed-task inference

Prompt Tuning

Liu et al., 2022

- P-tuning v2 (Liu et al., 2022): Add prompts as prefix tokens to different layers



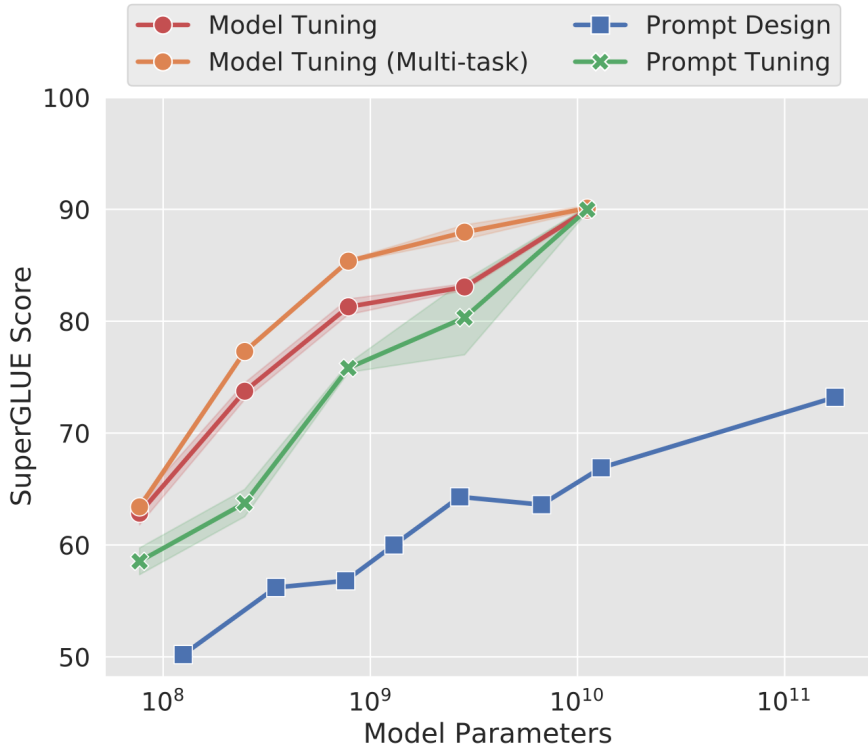
(a) Lester et al. & P-tuning (Frozen, 10-billion-scale, simple tasks)

(b) P-tuning v2 (Frozen, most scales, most tasks)

Recap of prefix/prompt tuning

Method	Task	Re-param.	Deep PT	Multi-task	No verb.
P-tuning (Liu et al., 2021)	KP NLU	LSTM	-	-	-
PROMPTTUNING (Lester et al., 2021)	NLU	-	-	✓	-
Prefix Tuning (Li and Liang, 2021)	NLG	MLP	✓	-	-
SOFT PROMPTS (Qin and Eisner, 2021)	KP	-	✓	-	-
P-tuning v2 (Ours)	NLU SeqTag	*(depends)*	✓	✓	✓

Table from Liu et al., 2022

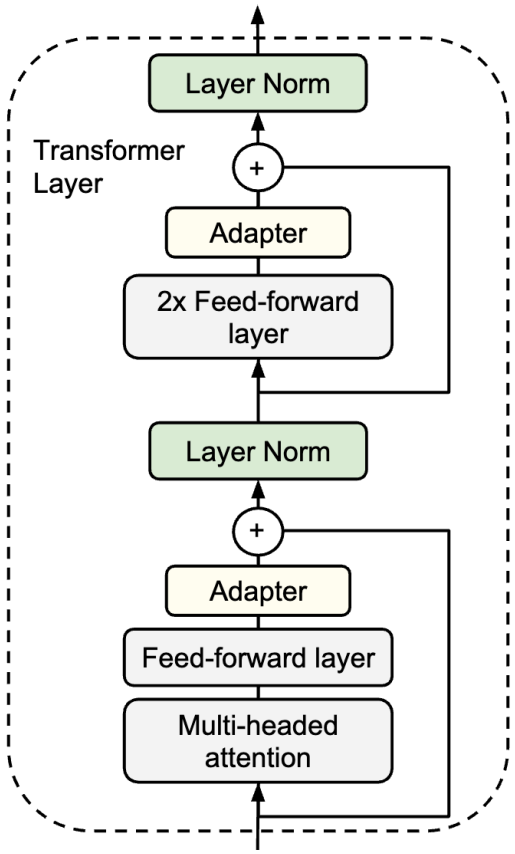


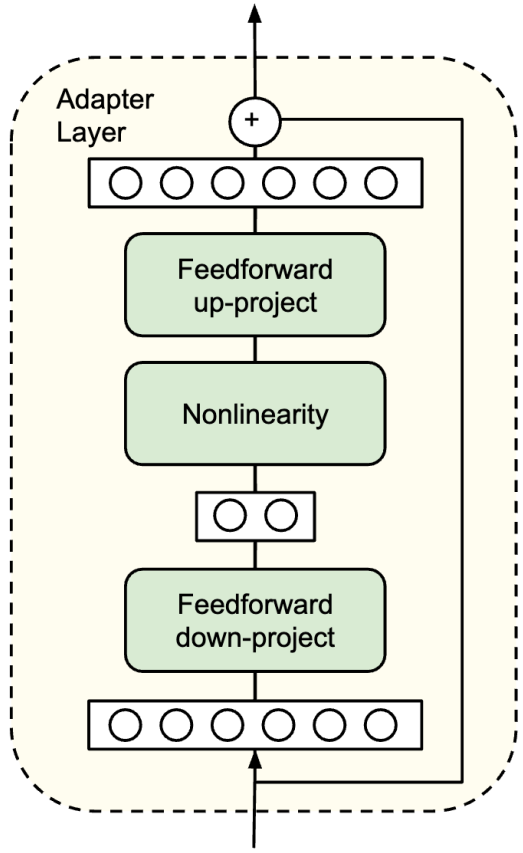
General performance of prompt tuning (Figure from Lester et al., 2021)

Adapter

Houlsby et al., 2019

- Insert a new function f_ϕ between layers of a pretrained model to adapt to a downstream task --- known as "**adapters**"
- Add only a few trainable parameters per task; new tasks can be added without revisiting previous ones
- The parameters of the original network remain fixed

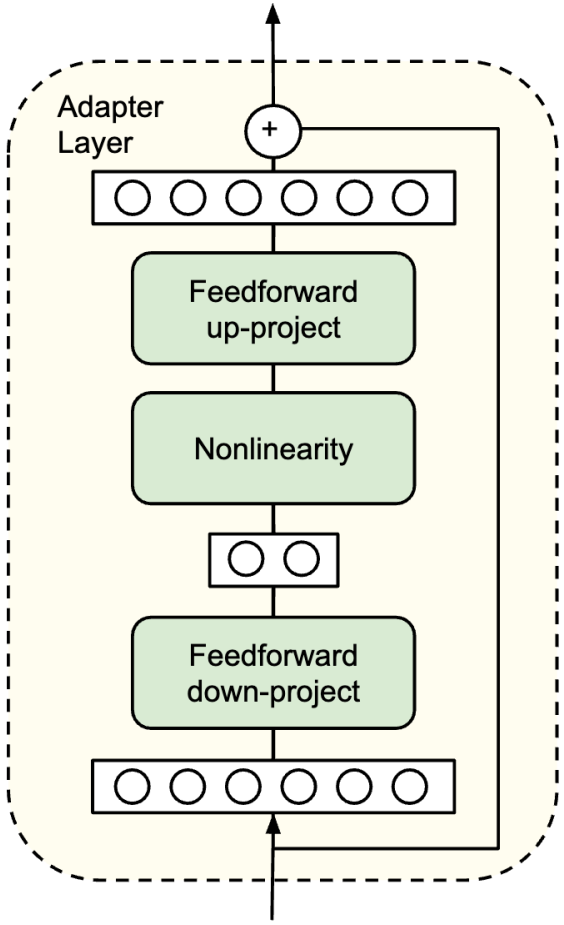




Adapter

Houlsby et al., 2019

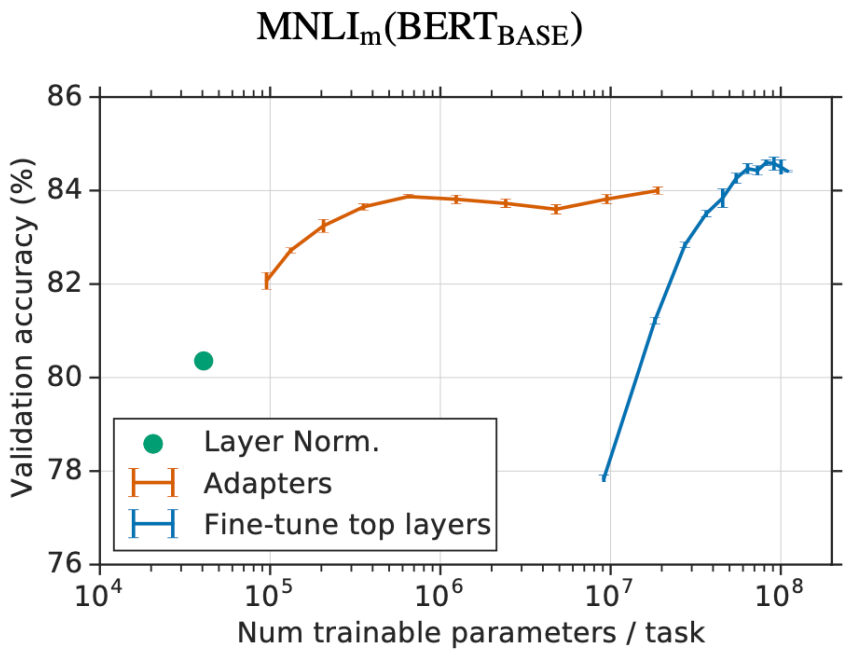
- An adapter in a Transformer layer consists of:
- A feed-forward down-projection $W^D \in \mathbb{R}^{k \times d}$
- A non-linearity
- A feed-forward up-projection $W^U \in \mathbb{R}^{d \times k}$
- $f_\phi(x) = W^U \sigma(W^D x)$

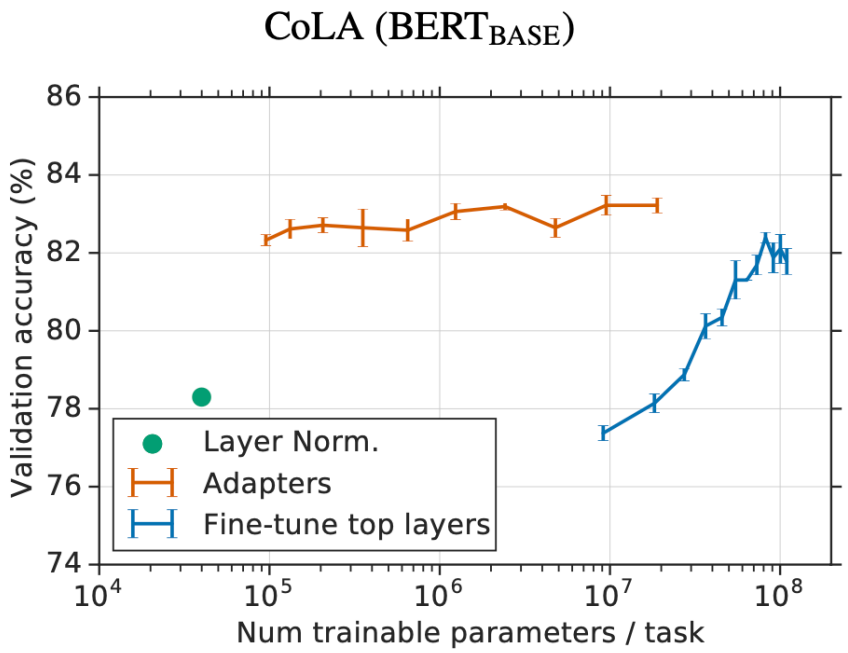


Adapter performance

Houlsby et al., 2019

- Adapters are earlier work than LoRA and prompt-tuning
- Thus only compared with partial fine-tuning





Recap of PEFT

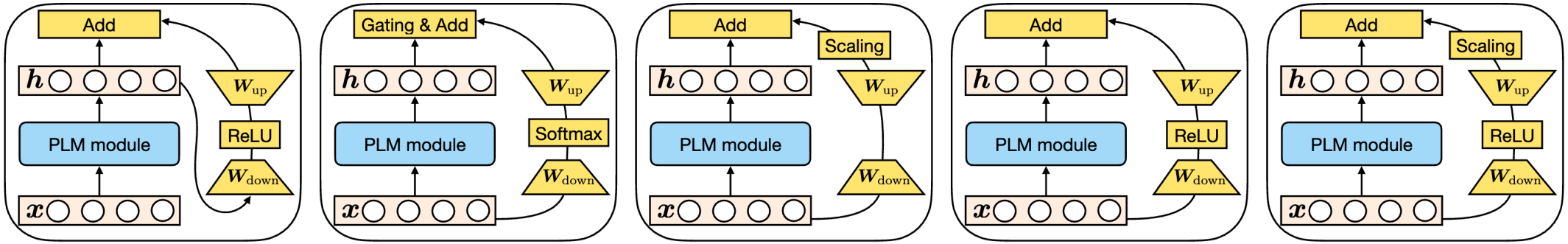
- Parameter Efficient Fine-Tuning
  - LoRA
  - Prefix-Tuning and Prompt Tuning
  - Adapter

Directly optimizing the prompt is hard

Adapter layers introduce inference latency

Unified View of PEFT

- He et al. (2022) show that LoRA, prefix tuning, and adapters can be expressed with a similar functional form
- All methods can be expressed as modifying a model's hidden representation $\boldsymbol{h}$

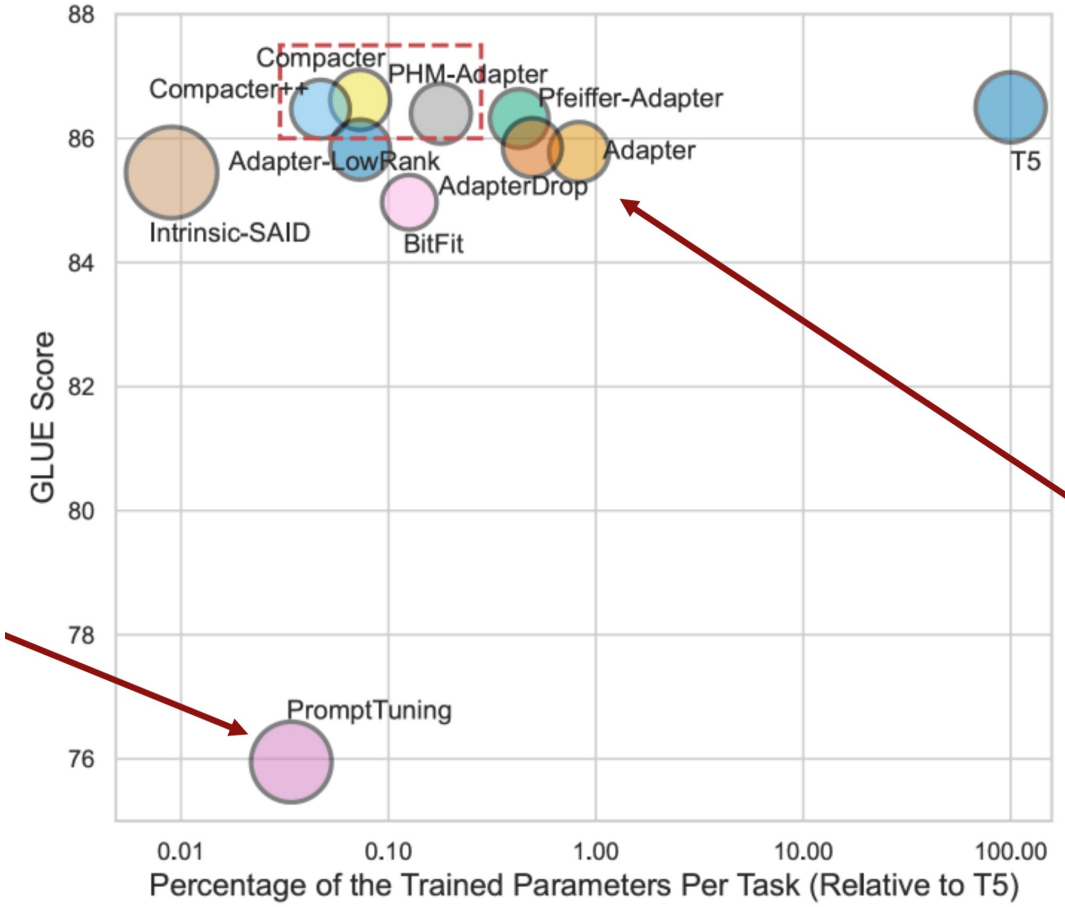


(a) Adapter (b) Prefix Tuning (c) LoRA (d) Parallel Adapter (e) Scaled PA

"PLM" for *pretrained language modeling*

PEFT Performance Comparison

Prompt tuning underperforms the other methods due to limited capacity



Adapter achieves better performance but add more parameters

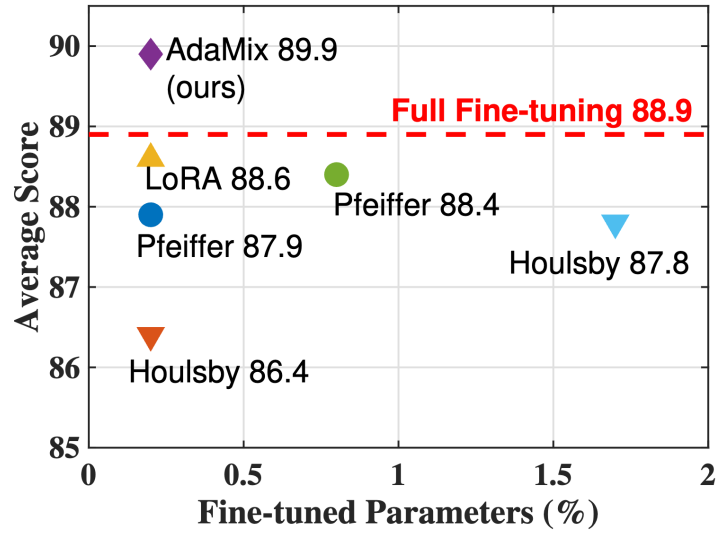


Figure source: Wang et al., 2022, AdaMix: Mixture-of-Adaptations for Parameter-efficient Model Tuning

Figure source: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/