# CS310 Natural Language Processing
# 自然语言处理
# Lecture 11 - Prompting and Parameter Efficient Fine-Tuning

Instructor: Yang Xu

主讲人：徐炀

xuyang@sustech.edu.cn

# Overview

- Prompting
- Parameter Efficient Fine-Tuning

# Emergent Abilities of LLMs: GPT (2018)

- Generative Pretrained Transformer (**GPT**) (OpenAI, Radford et al., 2018)
- 117M params
- 12 layers of transformer decoders
- Trained on BooksCorpus (4.6GB text)

- GPT showed that language model at scale (LLM) is an effective pretraining technique for downstream tasks
  - such as entailment, question answering, commonsense reasoning
- Pretraining + fine-tuning

outshined by BERT in the same year

涌现

# Emergent Abilities of LLMs: GPT-2 (2019)

- GPT-2 (Radford et al., 2019)
- 1.5B params; same architecture as GPT (bigger, 117M ⇒ 1.5B)
- Trained on **much more data**: 4.6GB ⇒ 40GB of internet text data

- One key emergent ability in GPT-2 is **zero-shot learning**:
- The ability to do many tasks with **no examples**, and **no gradient updates**

# Emergent zero-shot learning of GPT-2

- Question answering:

```
Passage: The 2008 Summer Olympics torch relay was run
from March 24 until August 8, 2008, prior to the 2008
Summer Olympics, with the theme of “one world, one dream”.

Q: What was the theme
A:
```

# Emergent zero-shot learning of GPT-2

- Comparing sentence probabilities

The cat couldn't fit into the hat because it was too big.

Does "it" = "the cat" or "the hat"?

Is P(... because the cat was too big) >=
P(... because the hat was too big)?

# Emergent zero-shot learning of GPT-2

- Summarization by adding **prompts**

"To induce summarization behavior we add the text **TL;DR:** after the article and generate 100 tokens with Top-k random sampling with k = 2"

**TL; DR ⇒ Too Long; Didn't Read**

"… approach the performance of classic neural baselines and just barely outperforms selecting 3 random sentences from the article."

- Zero-shot behavior comes from creative ways of specifying the task

# Emergent Abilities of LLMs: GPT-3 (2019)

- GPT-3 (175B parameters; Brown et al., 2020); same model and architecture as GPT-2
- Big increase in size: 1.5B ⇒ 175B
- Further increase in data: 40GB ⇒ 600+GB

"For all tasks, GPT-3 is applied without any gradient updates or fine-tuning, with tasks and few-shot demonstrations specified purely via text interaction with the model. "

# Emergent few-shot learning of GPT-3

- Few-shot learning: Specify a task by simply **prepending examples before the question**
  - Also known as **in-context learning**; no gradient updates are performed when learning a new task

```
5 + 8 = 13
7 + 2 = 9
1 + 0 = 1
3 + 4 = 7
5 + 9 = 14
9 + 8 = 17
```

In-context learning

```
gaot => goat
sakne => snake
brid => bird
fsih => fish
dcuk => duck
cmihp => chimp
```

In-context learning

```
thanks => merci
hello => bonjour
mint => menthe
wall => mur
otter => loutre
bread => pain
```

In-context learning

Source: Brown et al., 2020

# Explanation: Zero-shot

**Zero-shot**

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.

```
Translate English to French:
cheese =>
```

*task description*

*prompt*

Source: Brown et al., 2020

# Explanation: One-shot

**One-shot**

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.

```
Translate English to French:
sea otter => loutre de mer
cheese =>
```

*task description*

*example*

*prompt*

Source: Brown et al., 2020

# Explanation: Few-shot

**Few-shot**

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

```
Translate English to French:
sea otter => loutre de mer
peppermint => menthe poivrée
plush girafe => girafe peluche
cheese =>
```

*task description*

*examples*

*prompt*

Source: Brown et al., 2020

# Compared to traditional fine-tuning (not use in GPT-3)

**Fine-tuning**

The model is trained via repeated gradient updates using a large corpus of example tasks.

```
sea otter => loutre de mer
```

*example #1*

↓

**gradient update**

↓

```
peppermint => menthe poivrée
```

*example #2*

↓

**gradient update**

↓

```
plush giraffe => girafe peluche
```

*example #N*

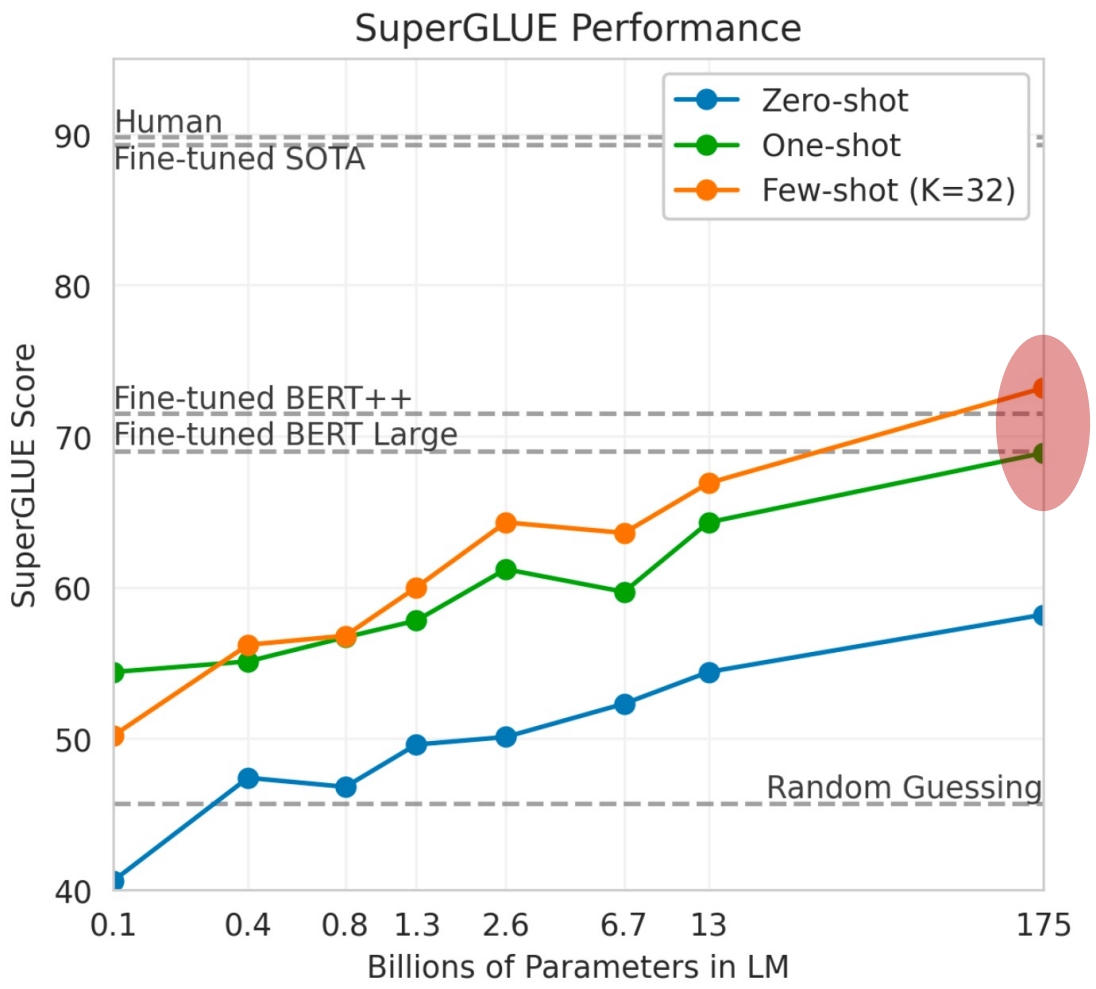
**gradient update**

```
cheese =>
```

*prompt*

Source: Brown et al., 2020

# Performance Comparison

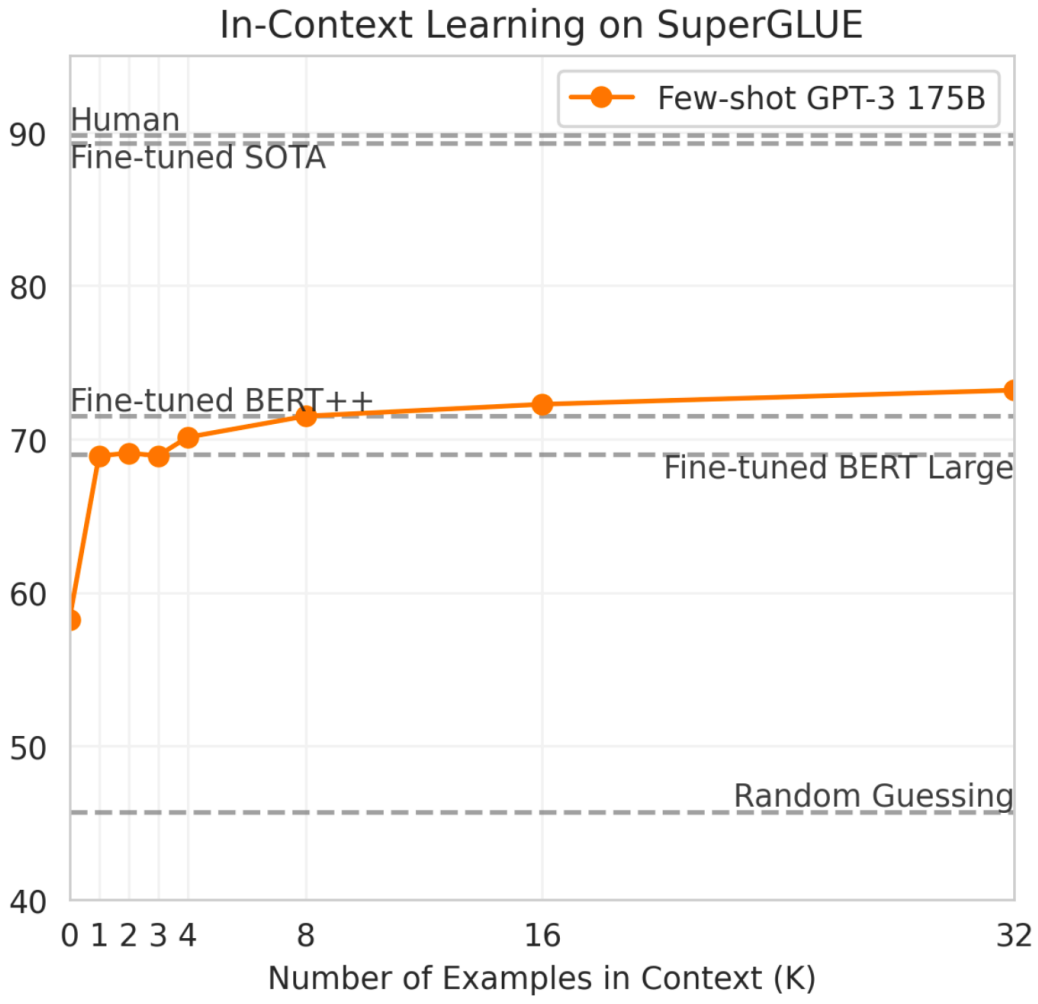



- BERT Large: fine-tuned on SuperGLUE training set (125K examples)
- BERT++: fine-tuned on MultiNLI (392K), SWAG (113K), and SuperGLUE (630K)

Difference between BERT-Large and BERT++ is roughly equivalent to the difference between GPT-3 with **one-shot** and **few-shot**

Source: Brown et al., 2020

# Performance Comparison

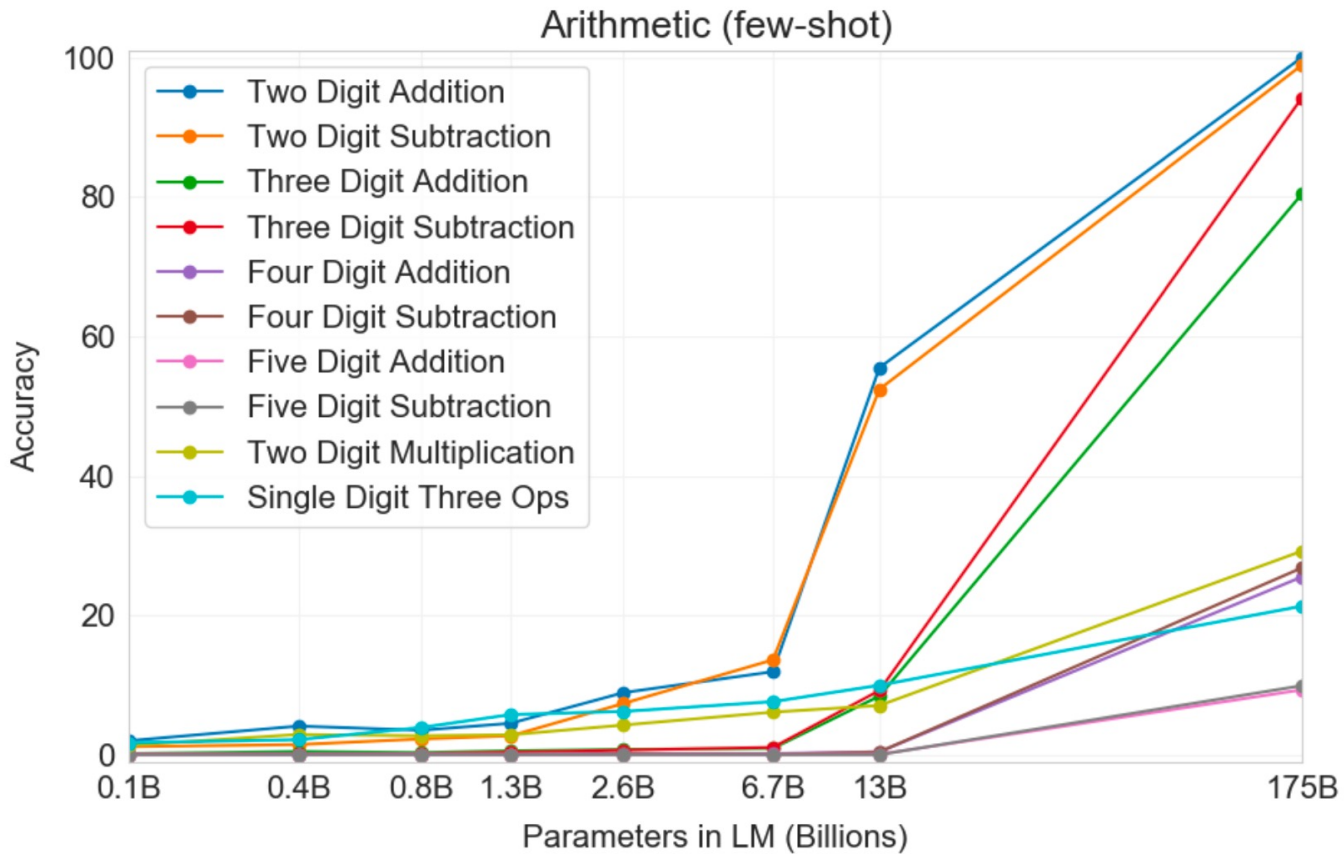



Difference between BERT-Large and BERT++ is roughly equivalent to the difference between GPT-3 with **one-shot** (1 example) and **few-shot** (8 examples)

Source: Brown et al., 2020

涌现

# Few-shot learning: An emergent property of model size




Significant jump from the second largest model (GPT-3 13B) to the largest model (GPT-3 175B)

| Model Name | $n_{\text{params}}$ |
|---|---|
| GPT-3 Small | 125M |
| GPT-3 Medium | 350M |
| GPT-3 Large | 760M |
| GPT-3 XL | 1.3B |
| GPT-3 2.7B | 2.7B |
| GPT-3 6.7B | 6.7B |
| GPT-3 13B | 13.0B |
| GPT-3 175B or “GPT-3” | 175.0B |

Source: Brown et al., 2020

# Limits and improvement of prompting

- Some tasks seem too hard for even large LMs to learn through prompting alone
- Especially tasks involving richer, **multi-step reasoning**

- Solution: **Chain-of-thought** prompting

# Chain of thought: a series of intermediate reasoning steps

**Standard Prompting**

**Model Input**

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

**Model Output**

A: The answer is 27. ✗

**Chain-of-Thought Prompting**

**Model Input**

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

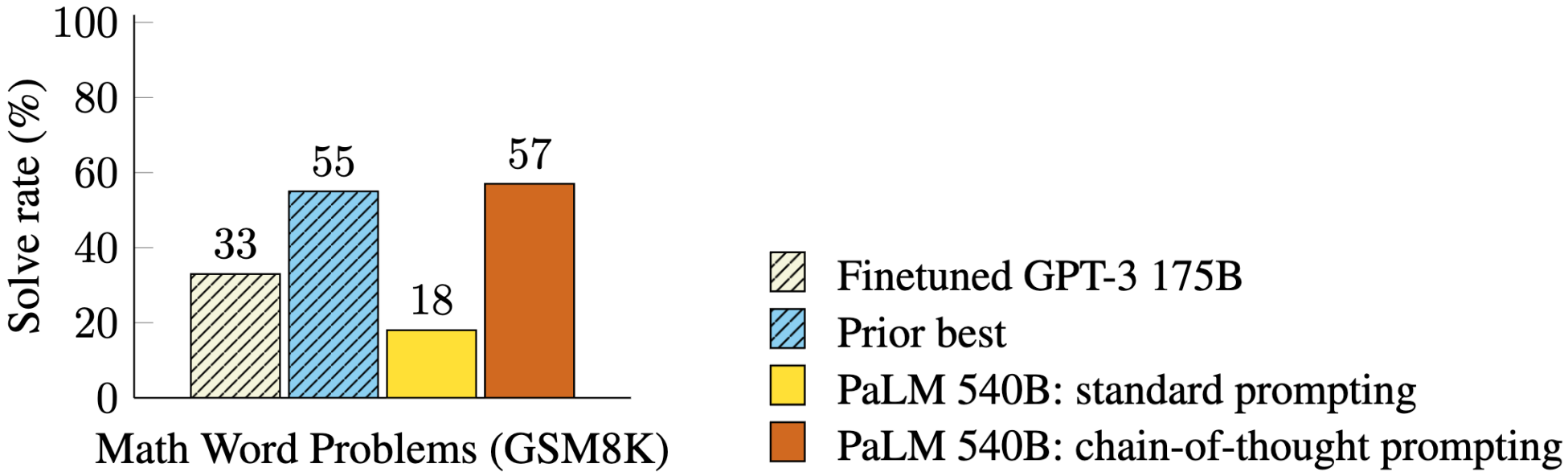
**Model Output**

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

Wei et al., 2022

# Chain of thought: a series of intermediate reasoning steps

- Prompting a **PaLM 540B** with just eight chain-of-thought exemplars achieves state-of-the-art accuracy on the GSM8K benchmark of math word problems (middle school math), surpassing even finetuned GPT-3 with a verifier.

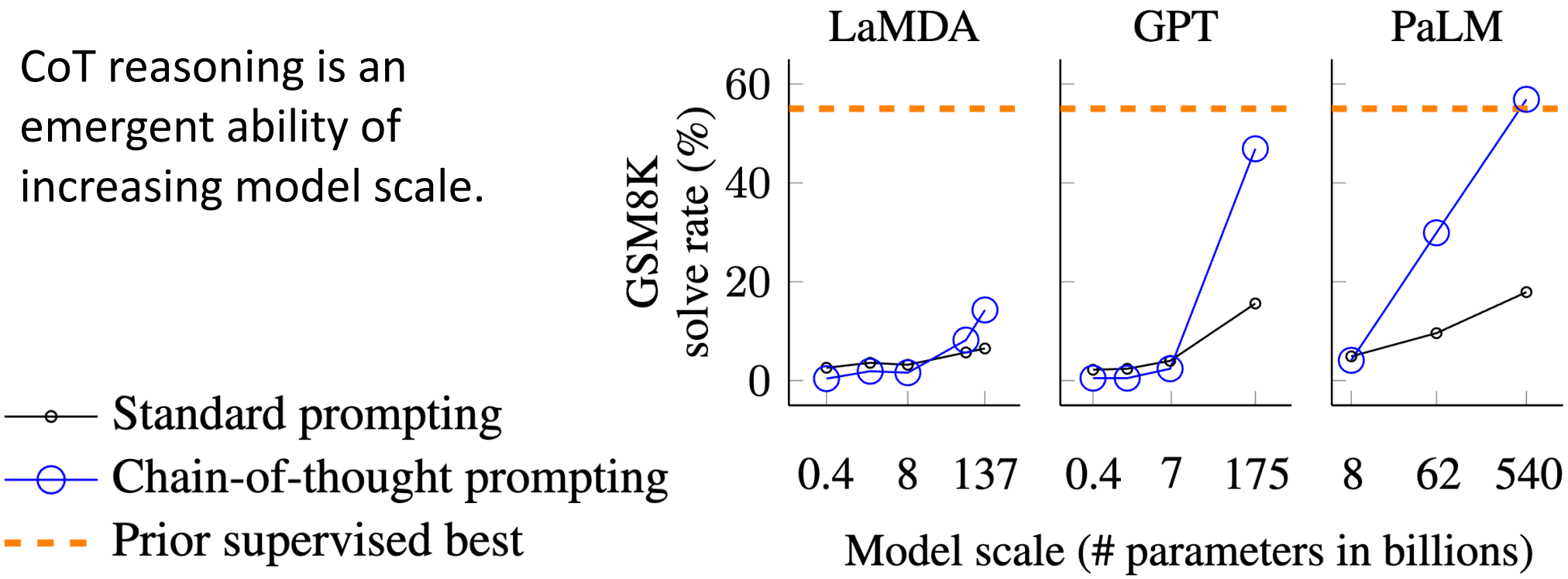



Wei et al., 2022

# CoT prompting is also an emergent property

涌现

- CoT reasoning is an emergent ability of increasing model scale.



# Zero-shot Chain of Thought

- Do we even need few-shot examples of reasoning?
- Can we just "**ask**" the model to reason through steps?
- Kojima et al., 2022: "LLMs are decent **zero-shot reasoners** by simply adding "Let's think step by step" before each answer."

# Zero-shot Chain of Thought

Kojima et al., 2022

(a) Few-shot

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:

---

*(Output) The answer is 8.* ✗

(b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:

---

*(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are 16 / 2 = 8 golf balls. Half of the golf balls are blue. So there are 8 / 2 = 4 blue golf balls. The answer is 4.* ✓

(c) Zero-shot

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: The answer (arabic numerals) is

---

*(Output) 8* ✗

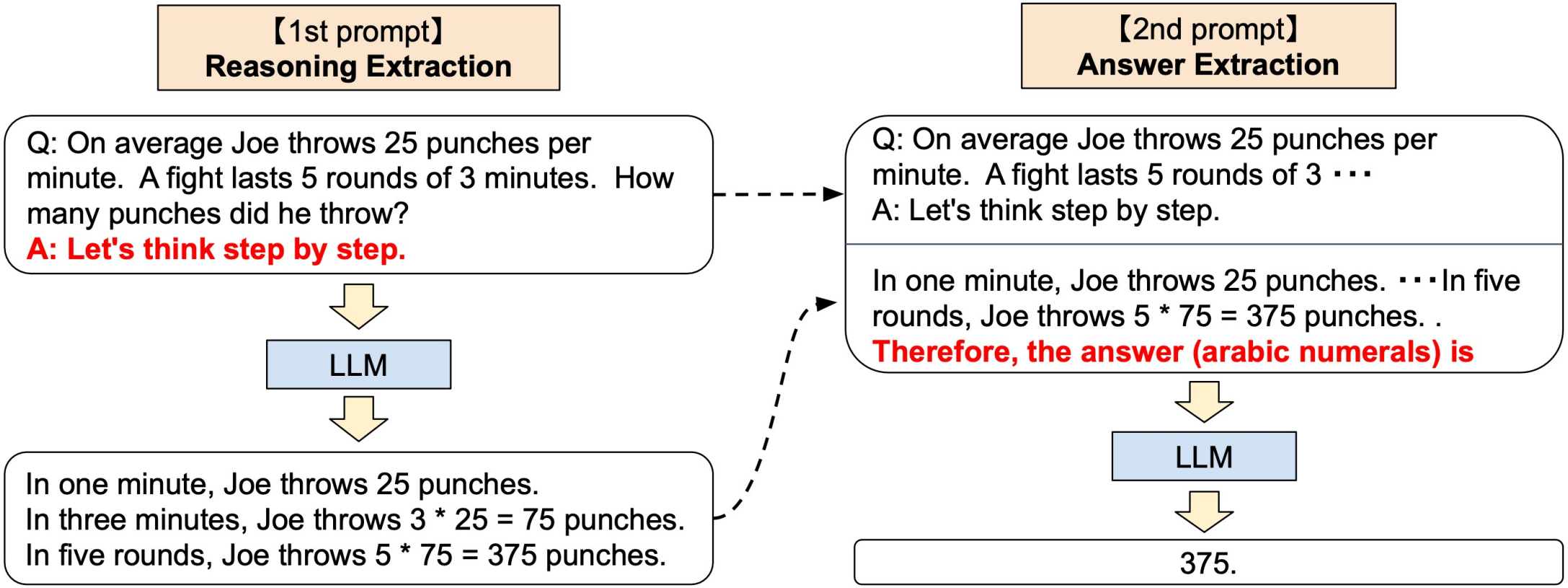
(d) Zero-shot-CoT (Ours)

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: ***Let's think step by step.***

---

*(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls.* ✓

# Detailed Method of CoT (Kojima et al., 2022)




- First use the first "reasoning" prompt to extract a full reasoning path from a language model
- Then use the second "answer" prompt to extract the answer

# Zero-shot CoT performance

Kojima et al., 2022

- Greatly outperform zero-shot; Manual CoT (with few-shot) is still better

| | MultiArith | GSM8K |
|---|---|---|
| **Zero-Shot** | **17.7** | **10.4** |
| Few-Shot (2 samples) | 33.7 | 15.6 |
| Few-Shot (8 samples) | 33.8 | 15.6 |
| **Zero-Shot-CoT** | **78.7** | **40.7** |
| Few-Shot-CoT (2 samples) | 84.8 | 41.3 |
| Few-Shot-CoT (4 samples : First) (*1) | 89.2 | - |
| Few-Shot-CoT (4 samples : Second) (*1) | 90.5 | - |
| Few-Shot-CoT (8 samples) | 93.0 | 48.7 |
| **Zero-Plus-Few-Shot-CoT (8 samples)** (*2) | **92.8** | **51.5** |

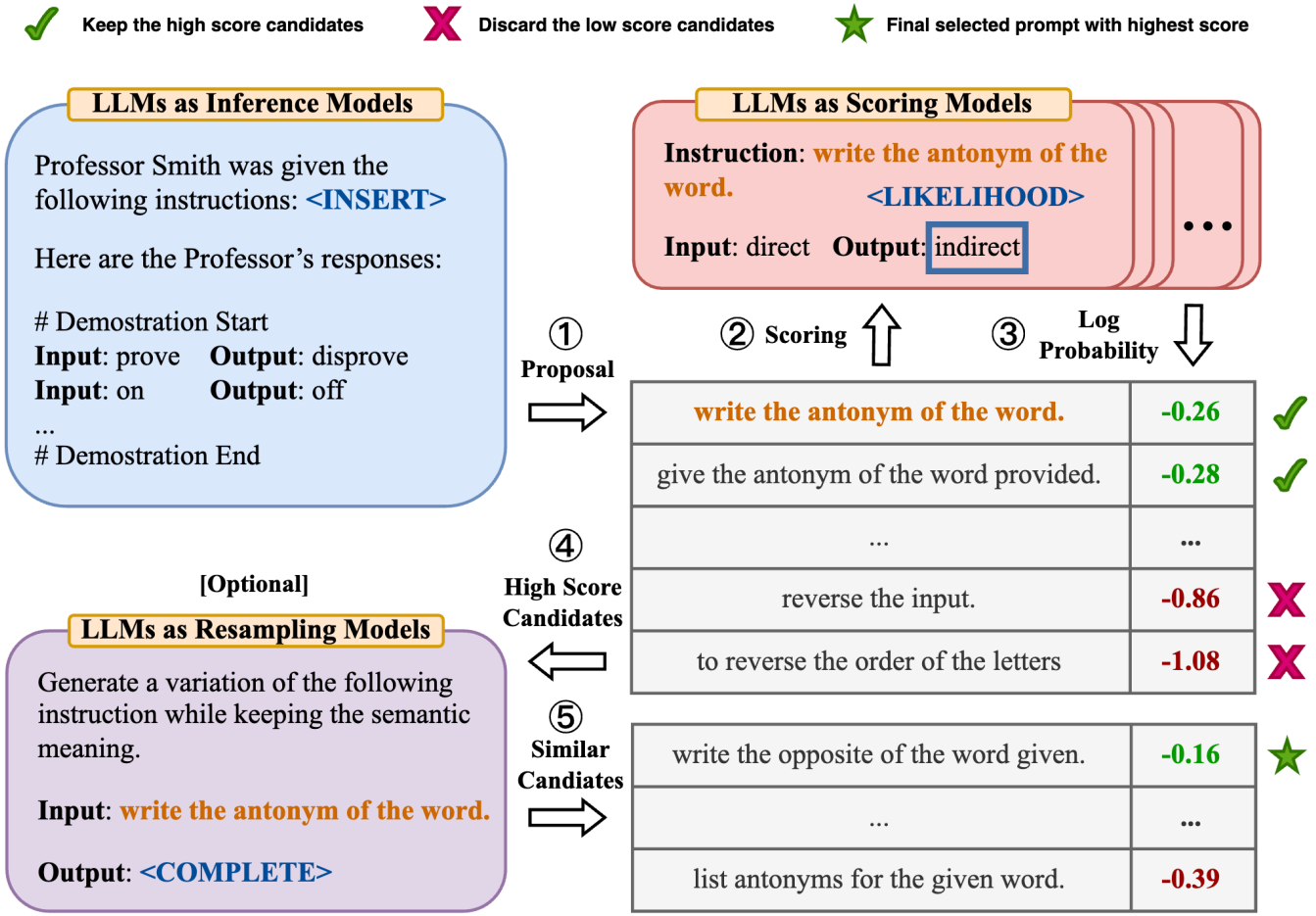
Few shot CoT is still better

# How does prompt selection affect 0-S CoT?

| No. | Category | Template | Accuracy |
|---|---|---|---|
| 1 | instructive | Let's think step by step. | **78.7** |
| 2 | | First, (*1) | 77.3 |
| 3 | | Let's think about this logically. | 74.5 |
| 4 | | Let's solve this problem by splitting it into steps. (*2) | 72.2 |
| 5 | | Let's be realistic and think step by step. | 70.8 |
| 6 | | Let's think like a detective step by step. | 70.3 |
| 7 | | Let's think | 57.5 |
| 8 | | Before we dive into the answer, | 55.7 |
| 9 | | The answer is after the proof. | 45.7 |
| 10 | misleading | Don't think. Just feel. | 18.8 |
| 11 | | Let's think step by step but reach an incorrect answer. | 18.7 |
| 12 | | Let's count the number of "a" in the question. | 16.7 |
| 13 | | By using the fact that the earth is round, | 9.3 |
| 14 | irrelevant | By the way, I found a good restaurant nearby. | 17.5 |
| 15 | | Abrakadabra! | 15.5 |
| 16 | | It's a beautiful day. | 13.1 |
| - | | (Zero-shot) | 17.7 |

Kojima et al., 2022

# What's the best CoT prompt?




- Zhou et al., 2023: LLMs are human level prompt engineers
- **Automatic Prompt Engineer (APE)**: Automatic instruction generation and selection

1. It generates several instruction candidates
2. Executes them using the target model, and selects the most appropriate instruction based on the evaluation score

# Best prompt from APE

Zhou et al., 2023

| No. | Category | Zero-shot CoT Trigger Prompt | Accuracy |
|---|---|---|---|
| 1 | APE | Let's work this out in a step by step way to be sure we have the right answer. | **82.0** |
| 2 | Human-Designed | Let's think step by step. (*1) | 78.7 |
| 3 | | First, (*2) | 77.3 |
| 4 | | Let's think about this logically. | 74.5 |
| 5 | | Let's solve this problem by splitting it into steps. (*3) | 72.2 |
| 6 | | Let's be realistic and think step by step. | 70.8 |
| 7 | | Let's think like a detective step by step. | 70.3 |
| 8 | | Let's think | 57.5 |
| 9 | | Before we dive into the answer, | 55.7 |
| 10 | | The answer is after the proof. | 45.7 |
| - | | (Zero-shot) | 17.7 |

# Take-home message from zero-shot CoT

- There is enormous zero-shot knowledge hidden inside LLMs
- LLMs can be seen as general-purpose computers that execute programs specified by prompts (in natural language)
- Prompt engineering process can be automated

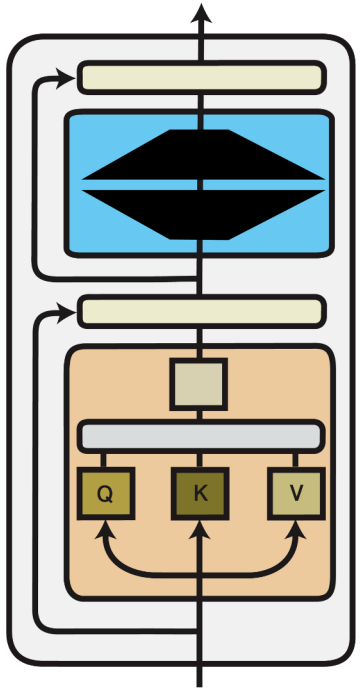
# Downside of Prompting

- **Inefficiency**: The prompt needs to be processed every time to make a prediction.
- **Suboptimal performance**: Prompting generally performs worse than fine-tuning
- **Sensitivity**: to the wording of the prompt (Webson & Pavlick, 2022), order of examples (Zhao et al., 2021; Lu et al., 2022), etc.
- **Lack of clarity**: about what the model learns from the prompt.
  - Even random labels work! (Zhang et al., 2022; Min et al., 2022)

# Overview

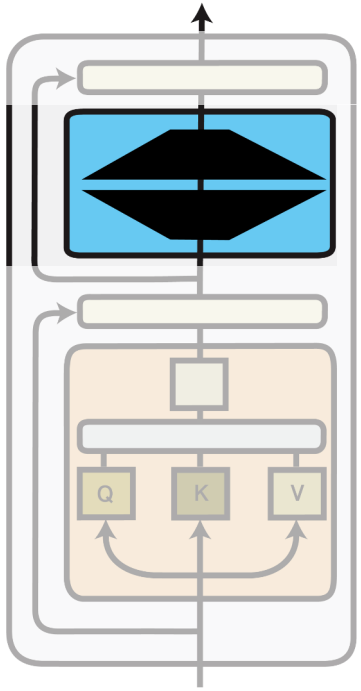
- Prompting
- **Parameter Efficient Fine-Tuning**
  - **LoRA**
  - **Prefix-Tuning and Prompt Tuning**
  - **Adapter**

# 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

Red AI refers to AI research that seeks to obtain state-of-the-art results in accuracy through the use of massive computational power—essentially "buying" stronger results

Green AI refers to AI research that yields novel results without increasing computational cost, and ideally reducing it

Green AI

Roy Schwartz*◇ Jesse Dodge*◇♣ Noah A. Smith◇♡ Oren Etzioni◇

# 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 $\phi_0$ 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 $\Theta$, $\Theta \ll |\phi_0|$
- The fine-tuning task becomes optimizing over $\Theta$:

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

# Low Rank Adaptation (LoRA)

Hu et al., 2021

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

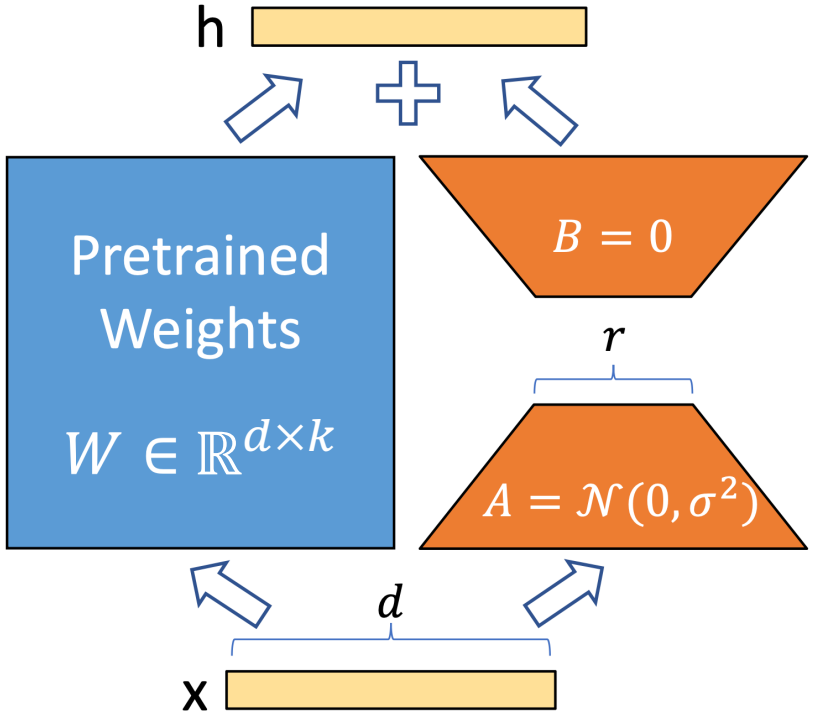
- LoRA: freezes the pre-trained model weights and injects trainable **rank decomposition** matrices into each layer
- For a pretrained weight $\mathbb{R}^{d\times k}$
- Constrain its update with low-rank decomposition:

$$W_0 + \Delta W = W_0 + BA$$

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

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



# 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

h =

| | | | | | |
|---|---|---|---|---|---|
| 0.46 | 2.49 | -0.63 | -0.55 | -0.66 | ... |
| 1.74 | -0.81 | -1.38 | -0.67 | 0.2 | ... |
| -0.76 | -1.31 | -0.85 | 0.66 | -0.47 | ... |
| -1.31 | -1.39 | -1.41 | -1.64 | -0.79 | ... |
| -0.14 | 0.45 | -0.51 | 1.13 | 0.46 | ... |
| ... | ... | ... | ... | ... | ... |

W

+

| | |
|---|---|
| 0 | 0 |
| 0 | 0 |
| 0 | 0 |
| 0 | 0 |
| 0 | 0 |
| ... | ... |

B

•

| | | | | | |
|---|---|---|---|---|---|
| -0.7 | 0.74 | 0.06 | 1.11 | -0.46 | ... |
| -0.96 | 0.71 | 0.42 | -0.4 | -0.88 | ... |

A

"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))
```

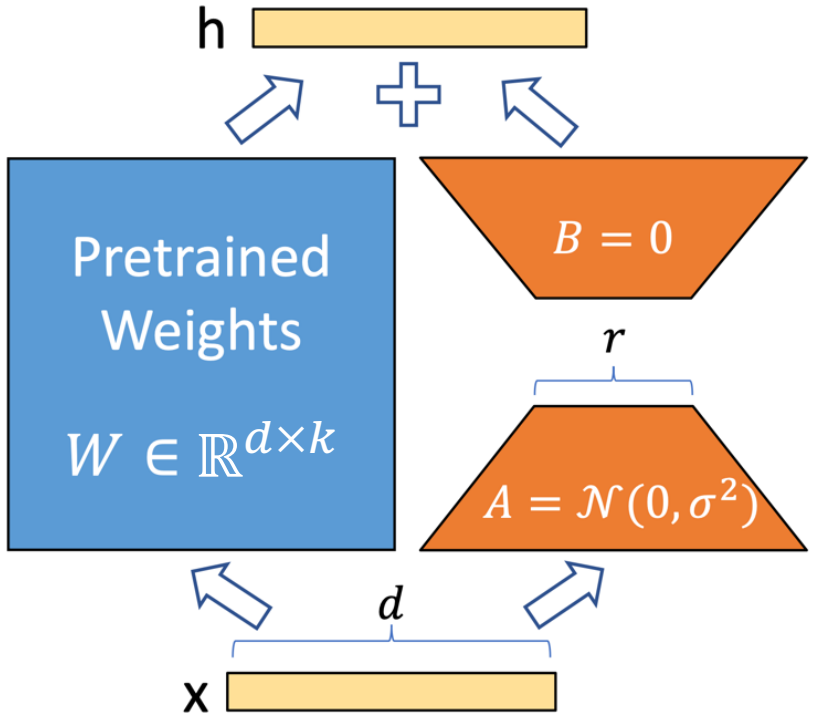
```
def lora_query(self, x):
    lora_query_weights =
        torch.matmul(self.lora_query_matrix_B,
                     self.lora_query_matrix_A)
    return self.query(x) + F.linear(x,
lora_query_weights)
```

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 $W_v$

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



# LoRA Performance

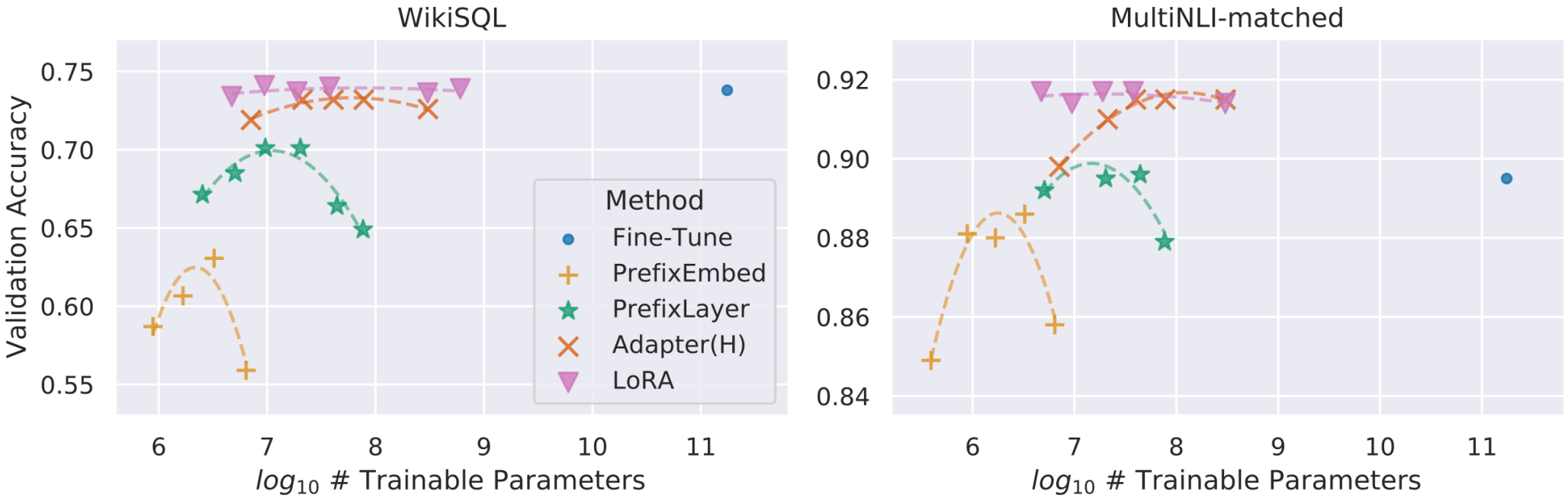
Hu et al., 2021

| 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 ($Adapter^H$) | 7.1M | 71.9 | 89.8 | 53.0/28.9/44.8 |
| GPT-3 ($Adapter^H$) | 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

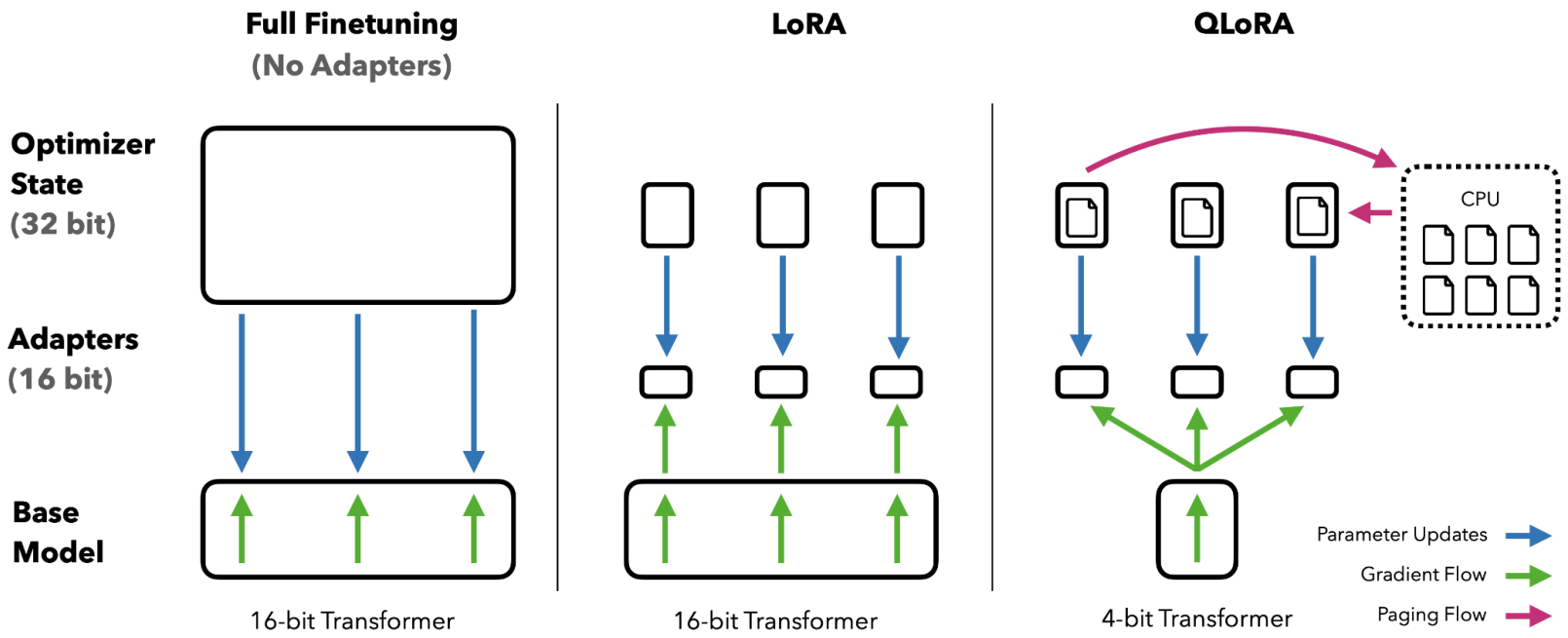
- What is the optimal rank $r$?

| | 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

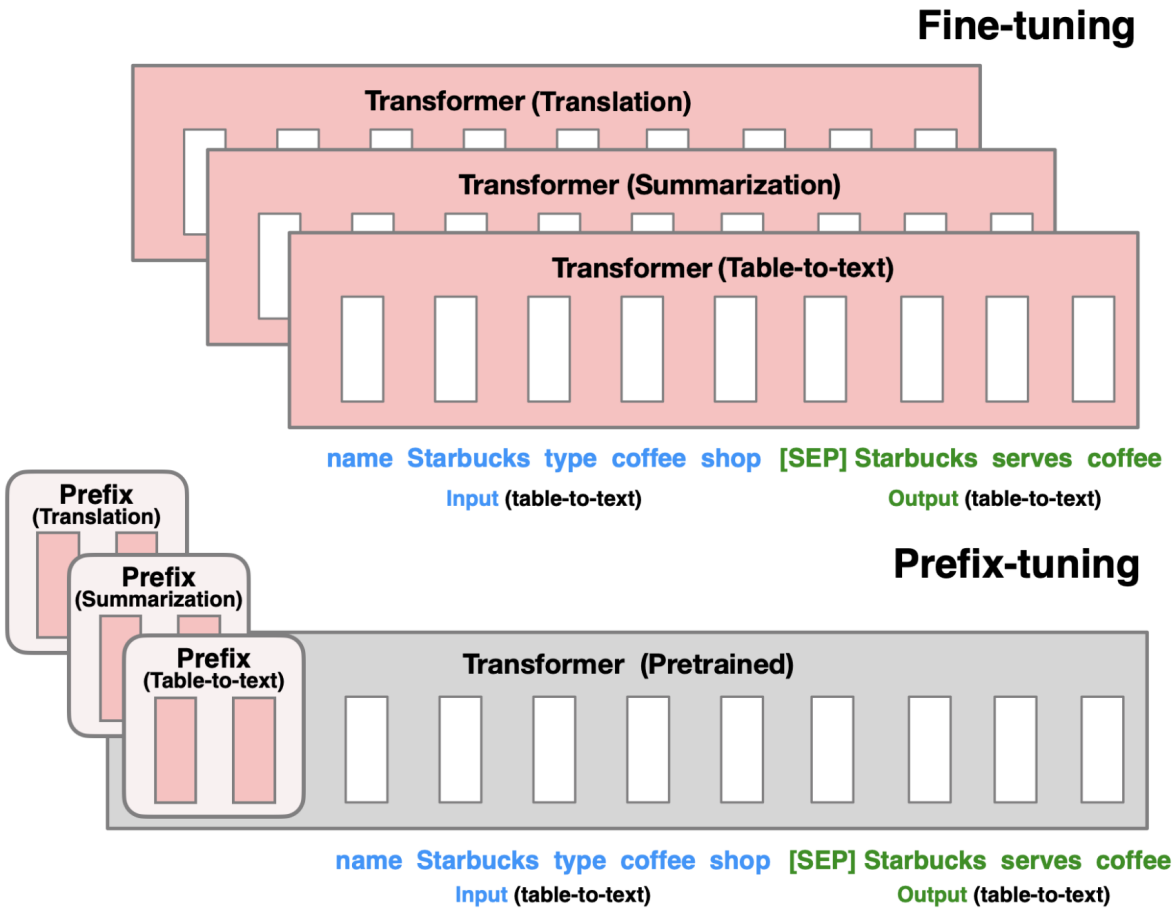
- **QLoRA**: improves over LoRA by quantizing the transformer model to 4- bit precision and using paged optimizer to handle memory spikes
- 4-bit NormalFloat (NF4), a new data type that is information theoretically optimal for normally distributed weights



# Prefix Tuning

Li and Liang, 2021

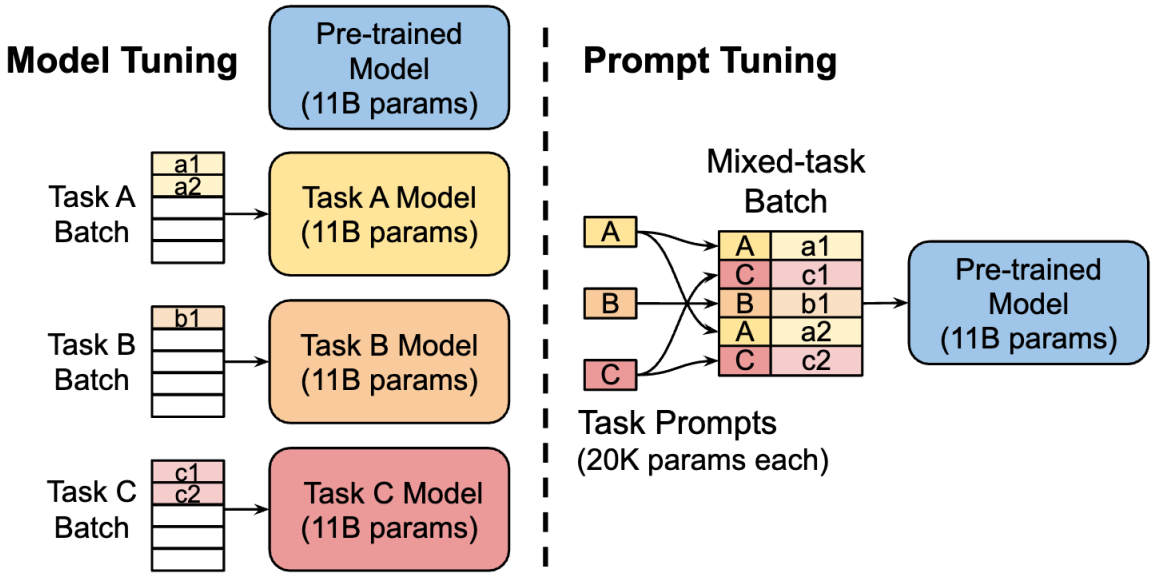
- Prefix-Tuning adds a prefix of parameters (*learnable*), and freezes all pretrained parameters
- Optimizes a sequence of continuous **task-specific vectors** -- called ***prefix***
- Consequently, only need to store the prefix for each task, making prefix-tuning modular and space-efficient



# 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

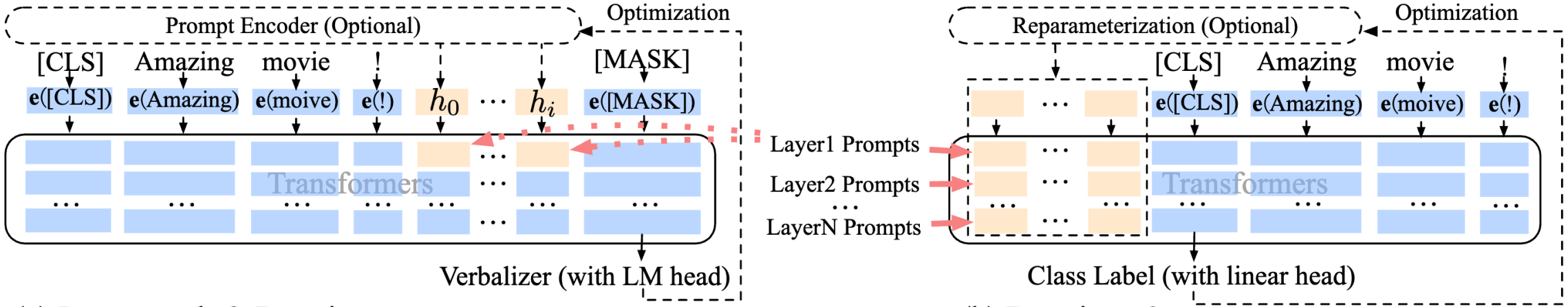



Prompt tuning requires storing a small task-specific prompt for each task, and enables mixed-task inference

# Prompt Tuning

Liu et al., 2022

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



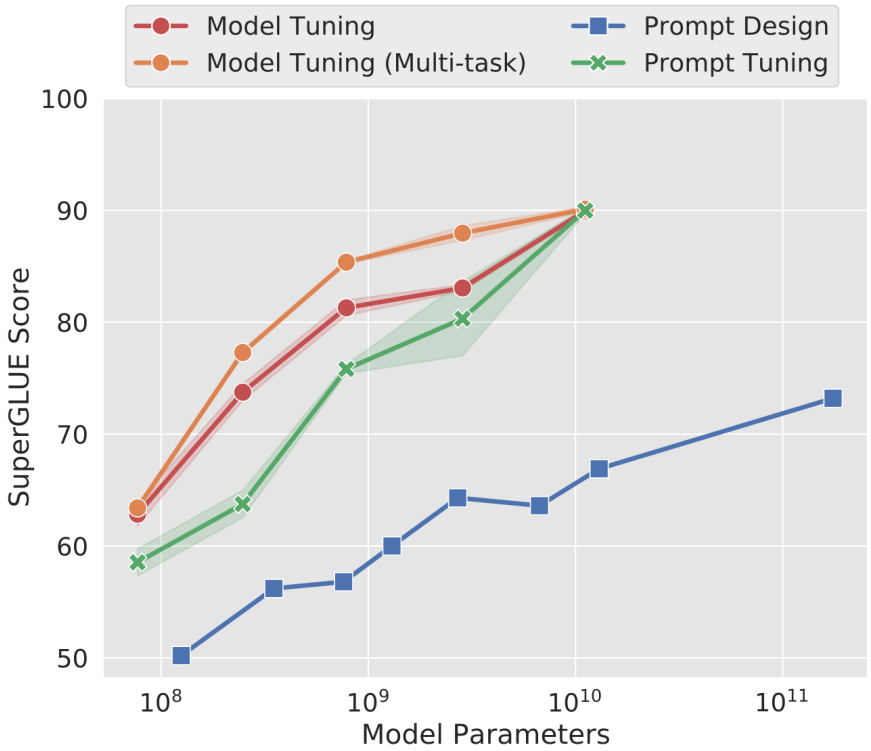

(a) Lester et al. & P-tuning (Frozen, 10-billion-scale, simple tasks)

(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



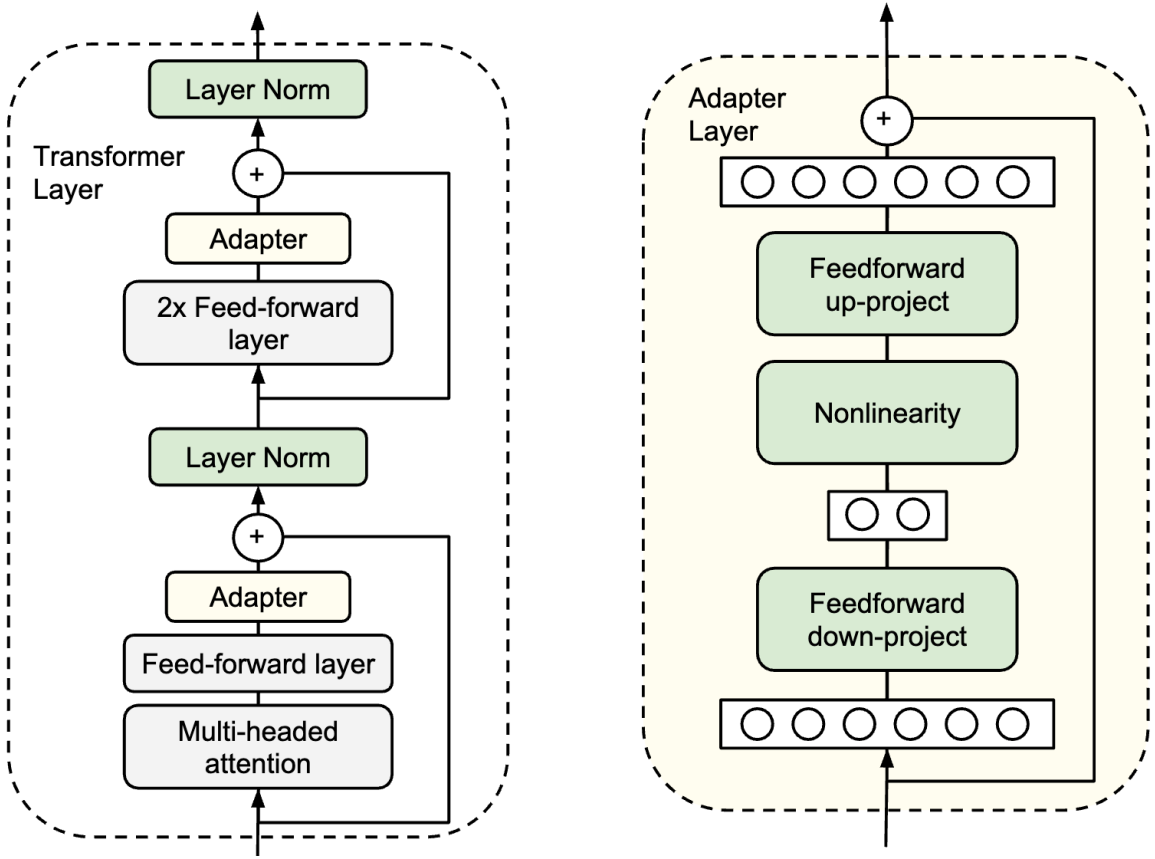

General performance of prompt tuning (Figure from Lester et al., 2021)

# Adapter

Houlsby et al., 2019

- Insert a new function $f_\phi$ between layers of a pretrained model to adapt to a downstream task --- known as "**adapters**"

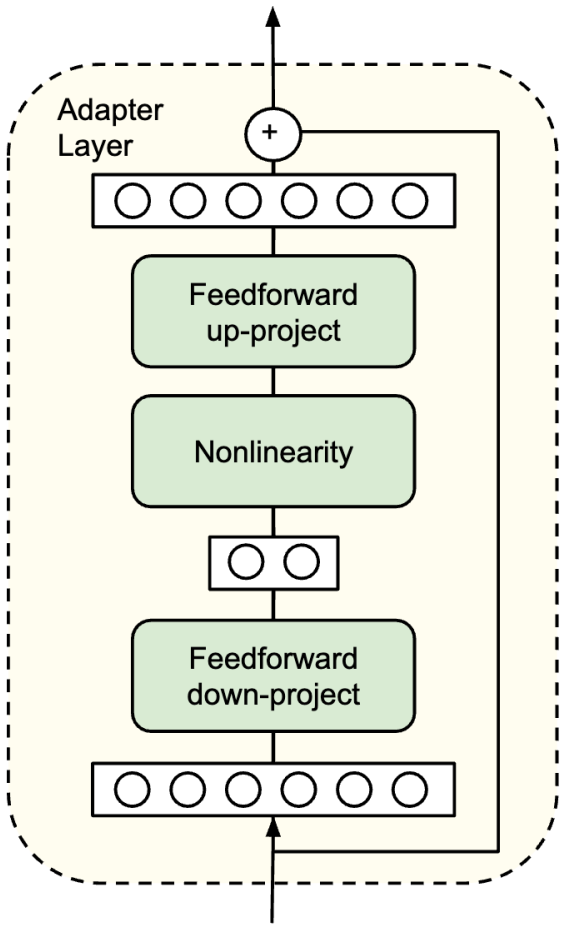
- Add only a few trainable parameters per task; new tasks can be added without revisiting previous ones
- The parameters of the original network remain fixed



# Adapter

Houlsby et al., 2019

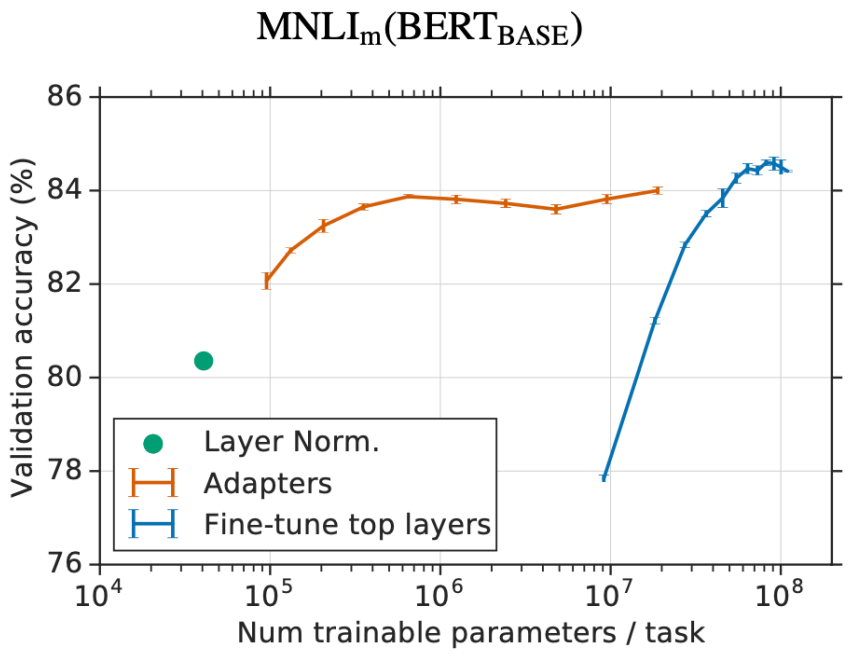
- An adapter in a Transformer layer consists of:
- A feed-forward down-projection $W^D \in \mathbb{R}^{k \times d}$
- A non-linearity
- A feed-forward up-projection $W^U \in \mathbb{R}^{d \times k}$
- $f_\phi(x) = W^U \sigma(W^D x)$

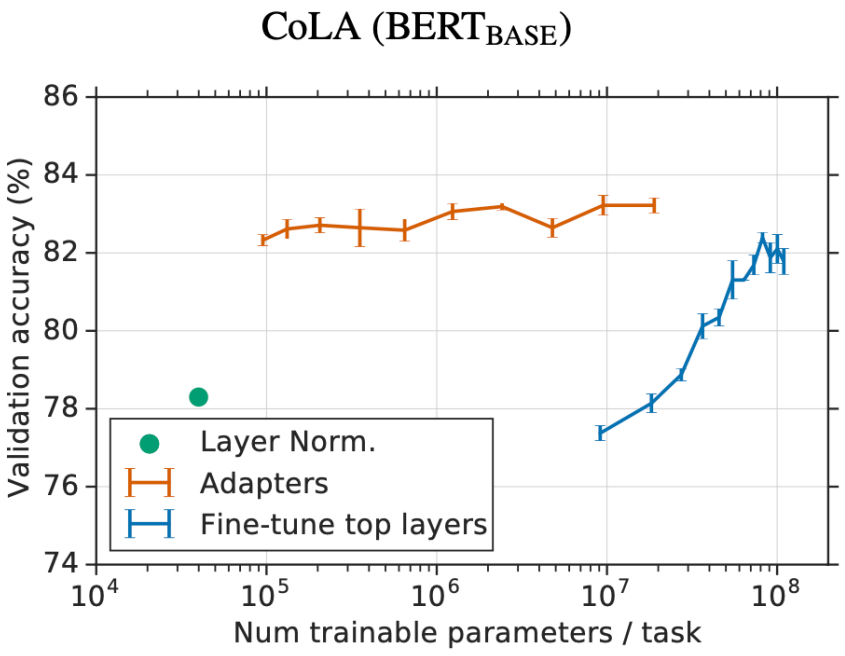


# Adapter performance

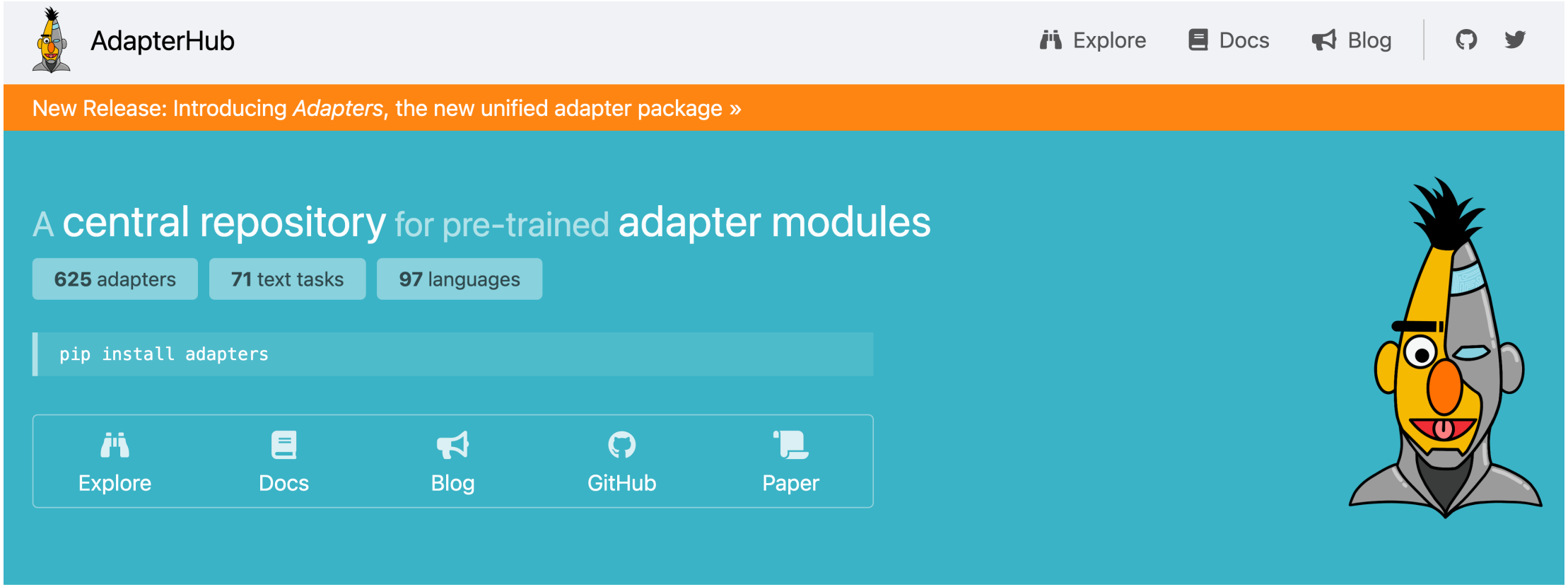
Houlsby et al., 2019

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



# Community-wide shared adapters




AdapterHub: [https://adapterhub.ml/](https://adapterhub.ml/)

# Recap of PEFT

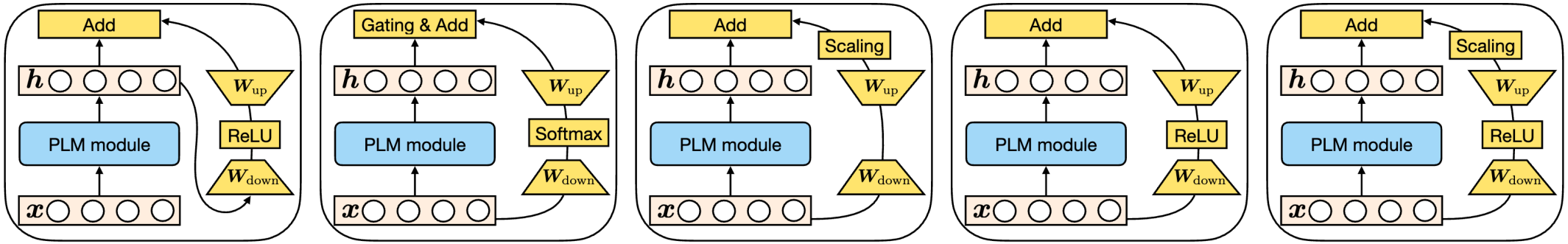
- Parameter Efficient Fine-Tuning
  - LoRA
  - Prefix-Tuning and Prompt Tuning
  - Adapter

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 $\boldsymbol{h}$

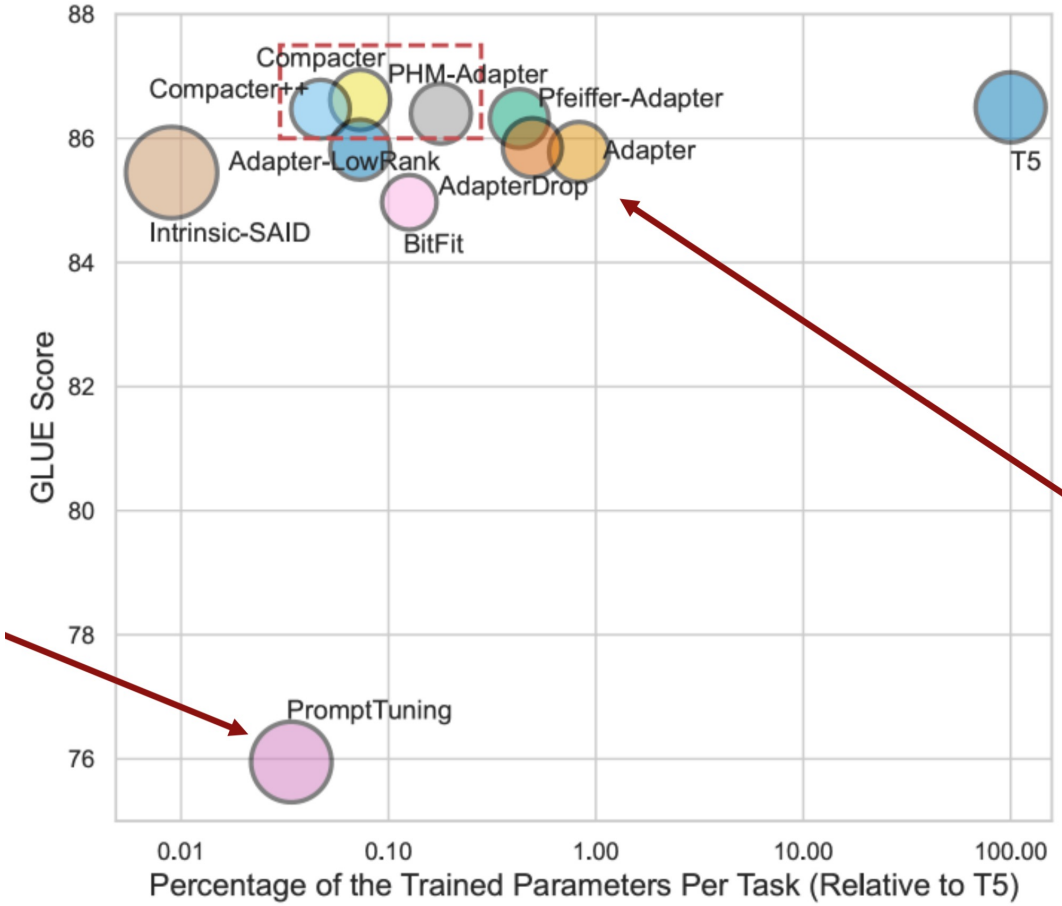



(a) Adapter (b) Prefix Tuning (c) LoRA (d) Parallel Adapter (e) Scaled PA

"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

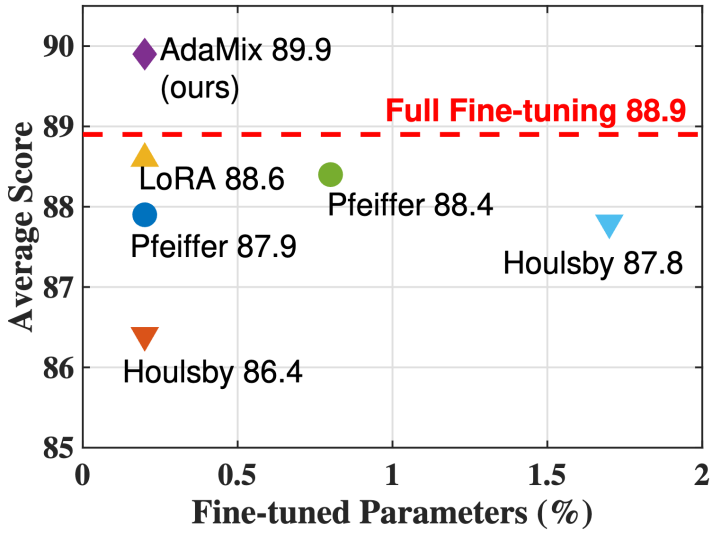



Figure source: Wang et al., 2022, AdaMix: Mixture-of-Adaptations for Parameter-efficient Model Tuning

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

# To-Do List

- Attend Lab 12

# References


- Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving language understanding by generative pre-training.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, *1*(8), 9.
- Schwartz, R., Dodge, J., Smith, N. A., & Etzioni, O. (2020). Green ai. *Communications of the ACM*, *63*(12), 54-63.
- Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., ... & Gelly, S. (2019, May). Parameter-efficient transfer learning for NLP. In *International conference on machine learning* (pp. 2790-2799). PMLR.
- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877-1901.
- Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., & Iwasawa, Y. (2022). Large language models are zero-shot reasoners. *Advances in neural information processing systems*, *35*, 22199-22213.
- Zhou, Y., Muresanu, A. I., Han, Z., Paster, K., Pitis, S., Chan, H., & Ba, J. (2022). Large language models are human-level prompt engineers. *arXiv preprint arXiv:2211.01910*.