



CS 639: Foundation Models **Specialization: Fine-Tuning & Adaptation II**

Fred Sala

University of Wisconsin-Madison

March 12, 2026



Announcements

- **Exam:** complete---good work!

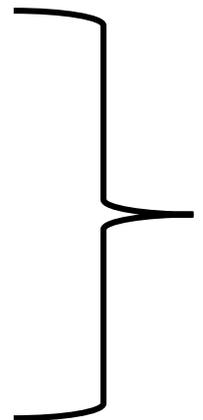
- **Homework 3:** out.

- **Project information:** released

https://pages.cs.wisc.edu/~fredsala/cs639/files/project_info_639.pdf

- **Class outline:**

Thursday March 12	Specialization II
Tuesday March 15	Alignment I
Thursday March 17	Alignment II
Tuesday March 22	More on Reasoning



Modifying and Improving
FMS

Outline

- **Fine-Tuning and Adapters**

- Fine-tuning vs. prompting, linear probing, etc. Full vs partial fine tuning vs adapting. Popular adapters

- **Cross-Modal Adaptation**

- Frozen transformers, ORCA, aligning via optimal transport dataset distance

- **Foundation Models + Tools**

- Beyond unaided LLMs and FMs, simple tool integration, Toolformer

Outline

- **Fine-Tuning and Adapters**

- Fine-tuning vs. prompting, linear probing, etc. Full vs partial fine tuning vs adapting. Popular adapters

- **Cross-Modal Adaptation**

- Frozen transformers, ORCA, aligning via optimal transport dataset distance

- **Foundation Models + Tools**

- Beyond unaided LLMs and FMs, simple tool integration, Toolformer

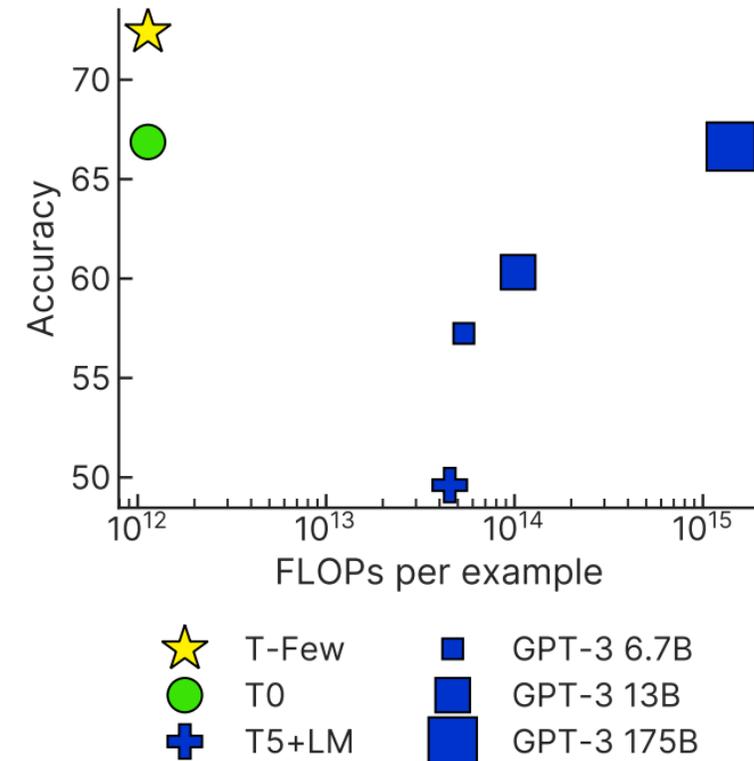
Before: Prompting

With prompting, we didn't change the model

- To improve performance, we used few-shot/ICL
- But, this might be **worse** than changing our model weights

Few-Shot Parameter-Efficient Fine-Tuning is Better and Cheaper than In-Context Learning

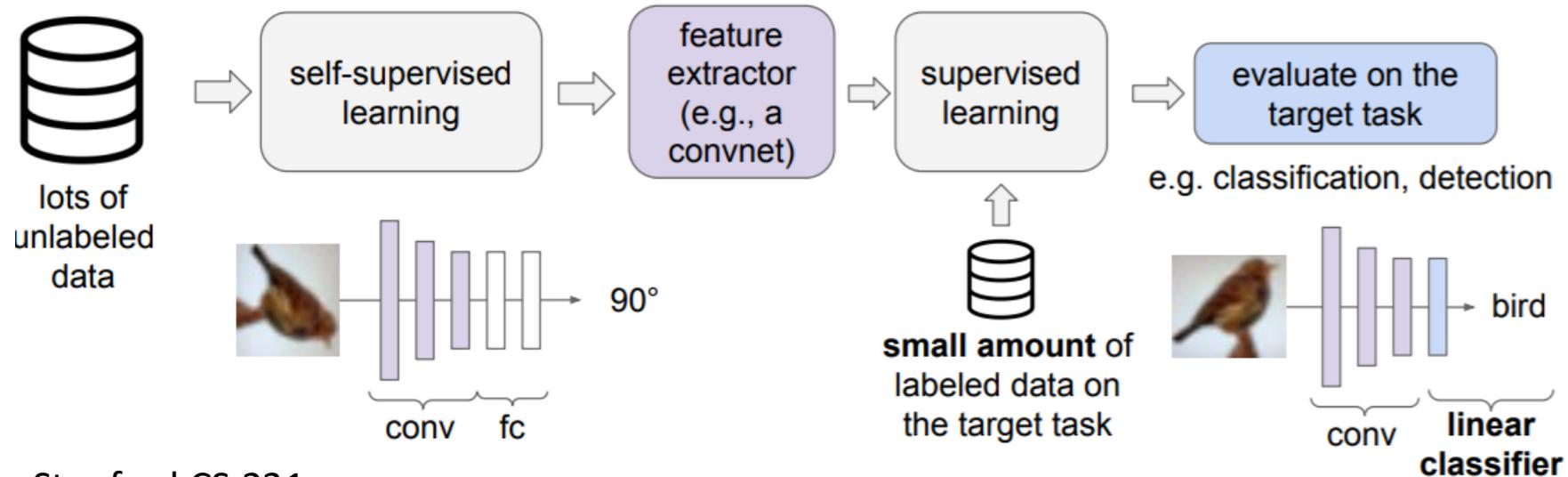
Liu et al '22



Before: Frozen Models/Linear Probing

We previously discussed freezing our model, and using just some trainable heads

- E.g., a linear model on top (called **linear probing**)
- Our self-supervised learning example



Full Fine-Tuning

Performance might still be bottlenecked,

- Frozen representations might not be suitable for task
- Might need lots of capacity on top to adapt
- **Change all the weights!**

```
>>> from transformers import AutoModelForSequenceClassification  
  
>>> model = AutoModelForSequenceClassification.from_pretrained("bert-base-cased", num_labels=5)  
  
>>> trainer.train()
```

<https://huggingface.co/docs/transformers/training>

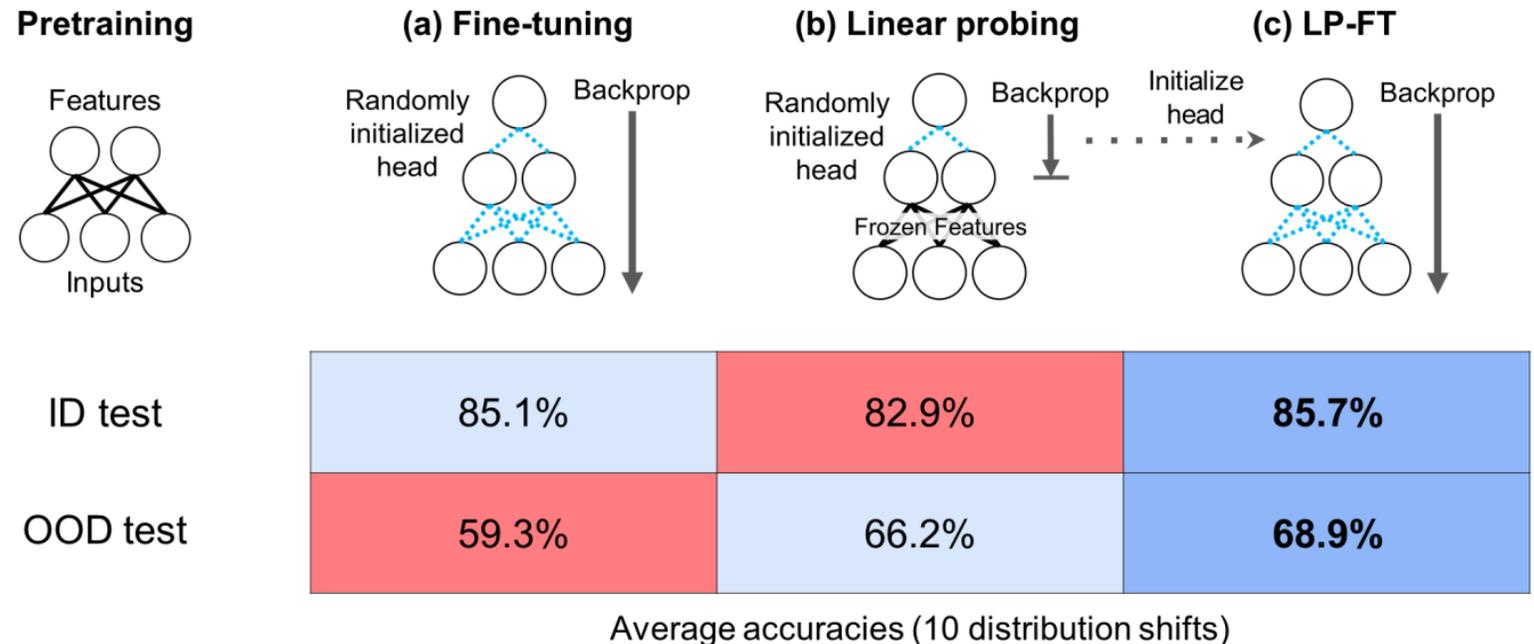
Full Fine-Tuning: Downsides

Fine-tuning all parameters can be tough:

1. Expensive: just like training a full model

2. Known to cause issues on OOD data...

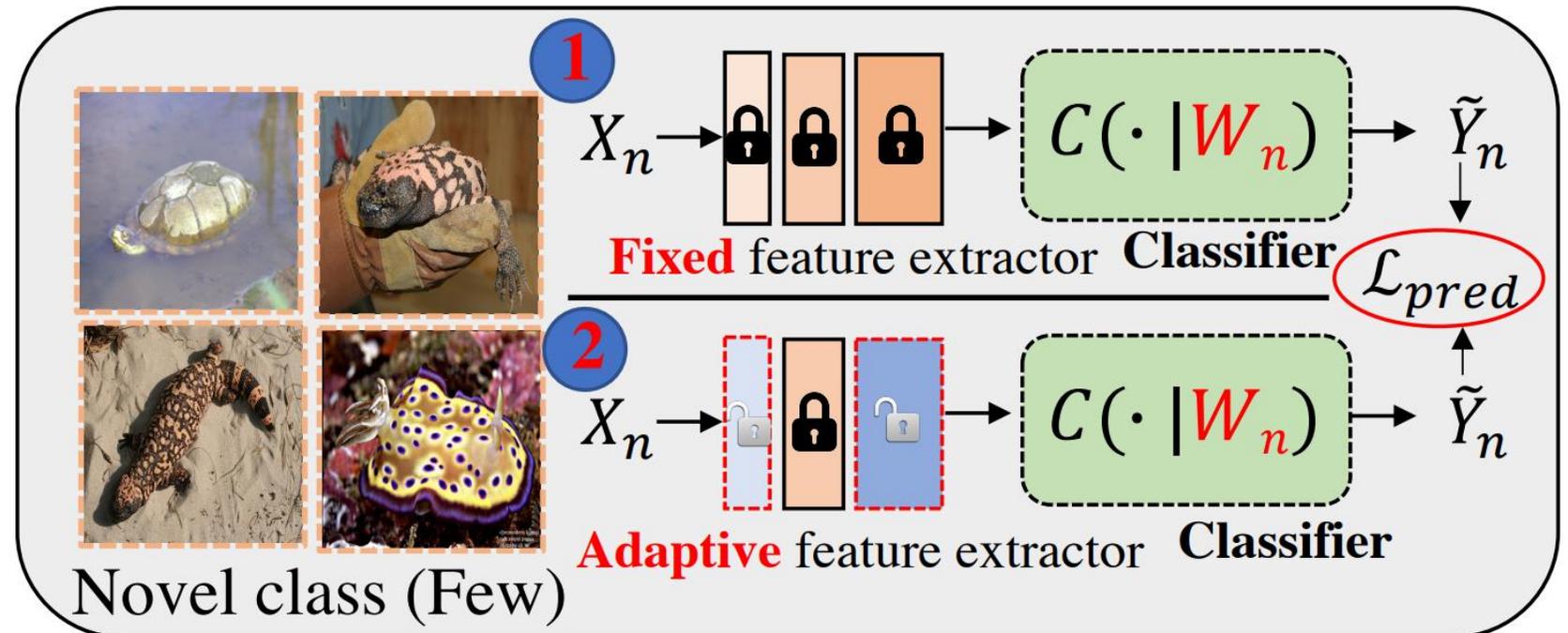
- Fine-Tuning can Distort Pretrained Features and Underperform Out-of-Distribution



Partial Fine-Tuning

Full fine-tuning might be expensive

- Partial fine-tuning might be a good choice
- Only some layers change

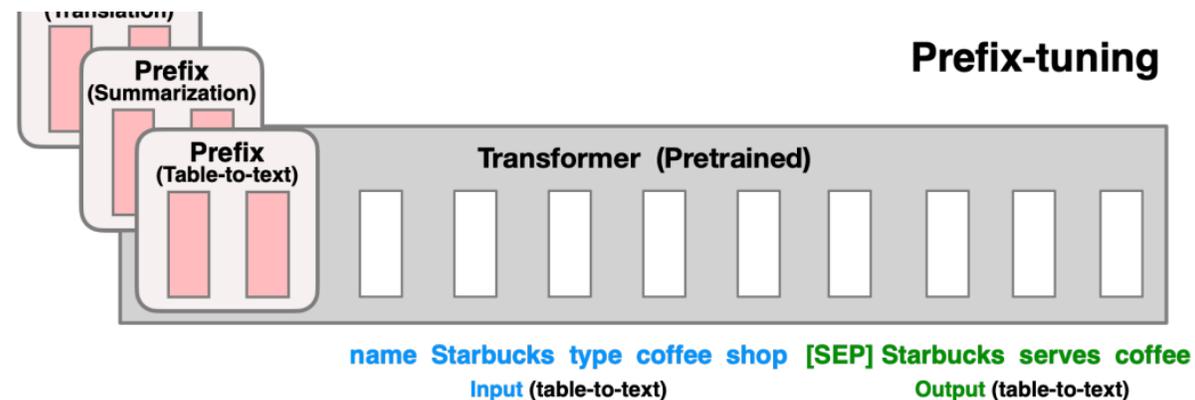


Fine-tuning stage

Prefix-Tuning

Recall this *soft prompting* method.

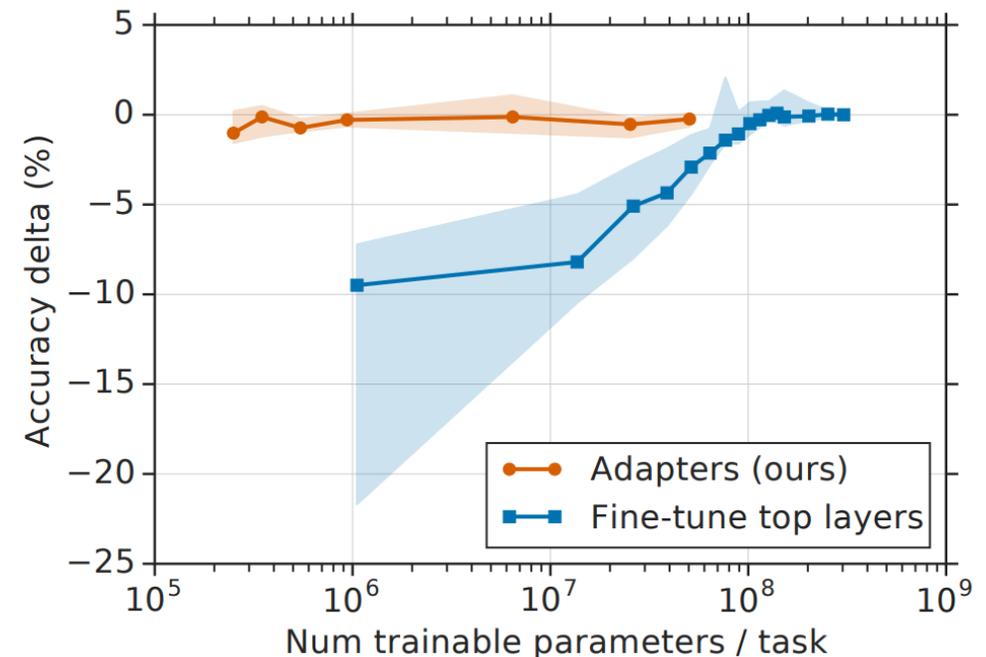
- Prefixes are trainable parameters
- Train one for each goal task, only store these new parameters
- Enables cheap **adaptation** of frozen language model



Parameter-Efficient Fine-Tuning (PEFT)

None of these methods were fully satisfying

- Have to figure out what layers to train, have to interpolate with prompts, etc.
 - Lots of choices!
- If we fine-tune too many parameters, that gets expensive...
 - Alternative, tune fewer parameters by fine-tuning top layers only, however performance isn't **great**
- Houlsby et al '19:



PEFT: Adapters

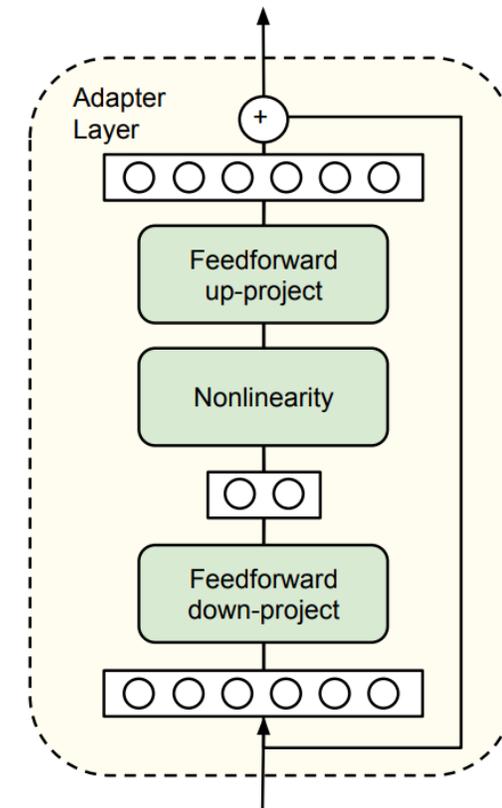
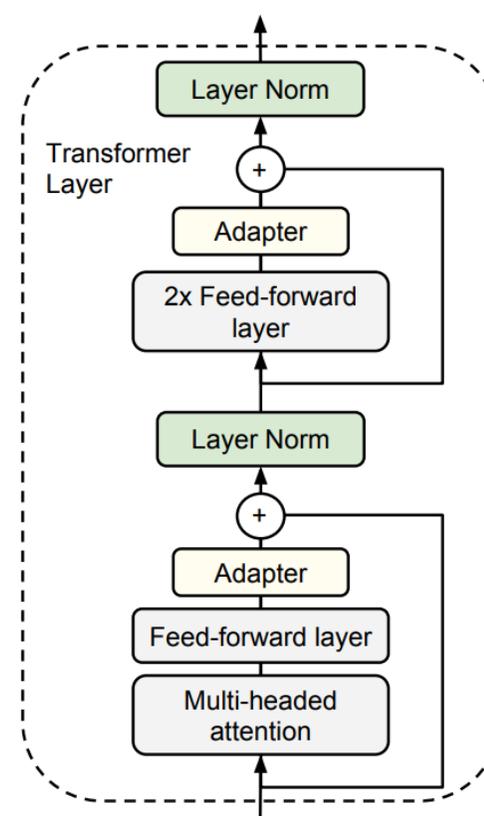
Want two things in PEFT

- Good performance (accuracy, etc.)
- Parameter efficiency

- **Solution: Adapters**

- Small modules, inserted in between model and trained

Another **advantage**: no change to model, new modules for tasks

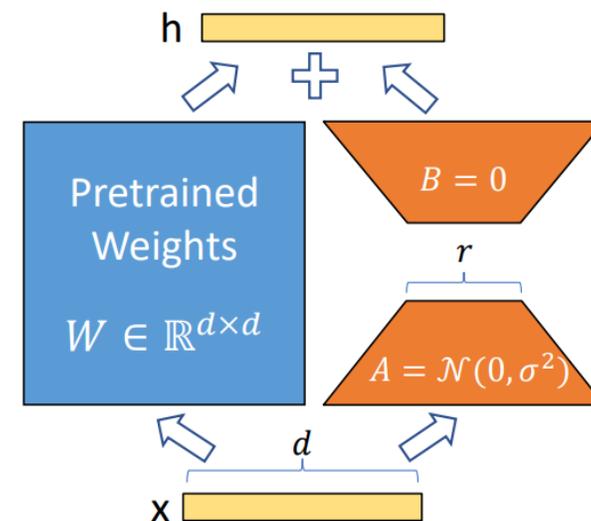


Houlsby et al '19

PEFT: Low-Rank Adapters (LoRA)

Perhaps the most popular variant

- LoRA suggests **adding** directly to pretrained weights
 - Instead of placing in a new module
 - The matrix to be added should be low-rank
 - Intuition: the weight matrices already live close to a low-rank manifold
- Transformers, initially applied only to a Attention weight matrices
 - Now everywhere



Takeaways

Specialization approaches:

- **Linear probing** – don't need to change anything, sits on top (but not usable for prompting-based methods)
- **Full fine-tuning** – expensive, may hurt OOD performance
 - Though some approaches counter the latter
- **Partial fine-tuning** – cheaper than full fine-tuning, but potentially worse performance, many choices to make
- **Parameter efficient fine-tuning** – good compromise.
 - Many techniques, **LoRA** is a popular choice.
 - Can combine with quantization (QLoRA) for maximum efficiency



Break & Questions

Outline

- **Fine-Tuning and Adapters**

- Fine-tuning vs. prompting, linear probing, etc. Full vs partial fine tuning vs adapting. Popular adapters

- **Cross-Modal Adaptation**

- Frozen transformers, ORCA, aligning via optimal transport dataset distance

- **Foundation Models + Tools**

- Beyond unaided LLMs and FMs, simple tool integration, Toolformer

What About Other Modalities?

So far, mostly talked about fine-tuning language models.

- Suppose we want tasks that are not directly language-based
- Could just train a new model...
 - As we did for multimodal models – but might be expensive.

Can we adapt language models? Lots of **challenges**:

- Must change data types
- How do we know modalities are usable together?

Cross-Modal: FPTs

Frozen language-pretrained transformers (Lu et al '21)

Basic idea:

- Change the **input/output layers** (here, linear)
- Layer norm parameters
- Everything else frozen

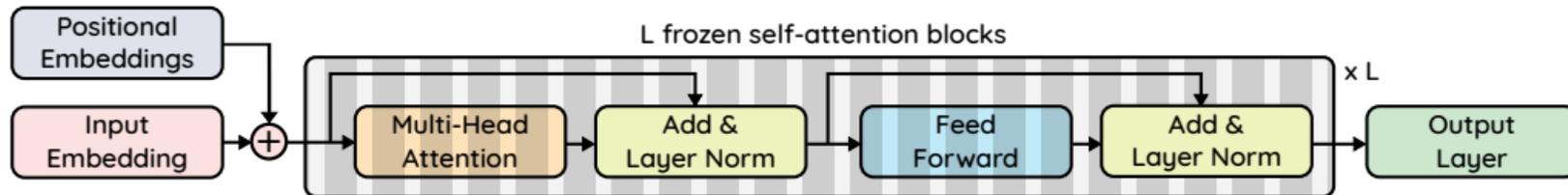


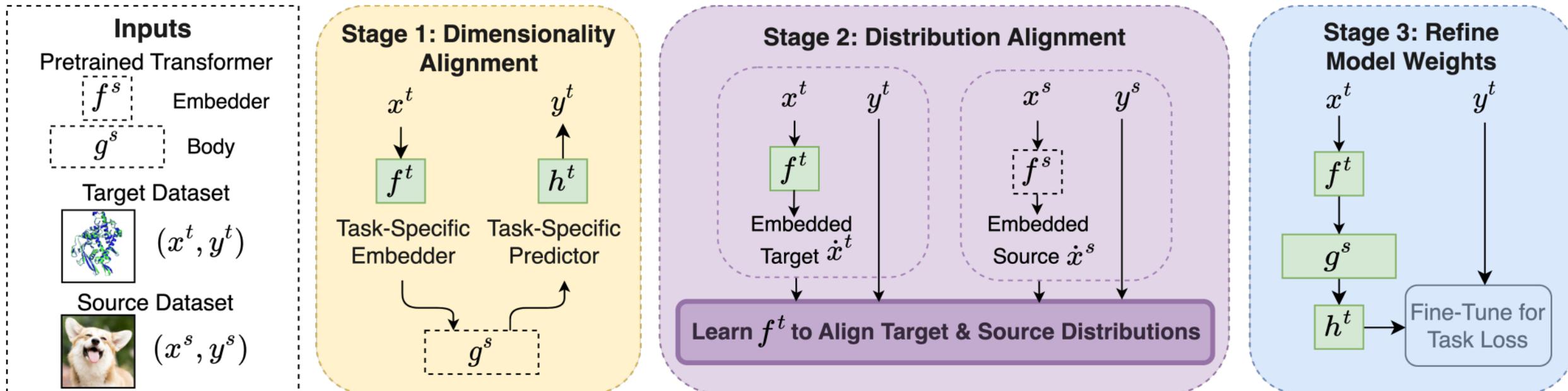
Figure 2: Frozen Pretrained Transformer (FPT). The self-attention & feedforward layers are frozen.

Cross-Modal: ORCA

Performance bottleneck in FPTs

A more powerful approach: ORCA (Shen et al '23)

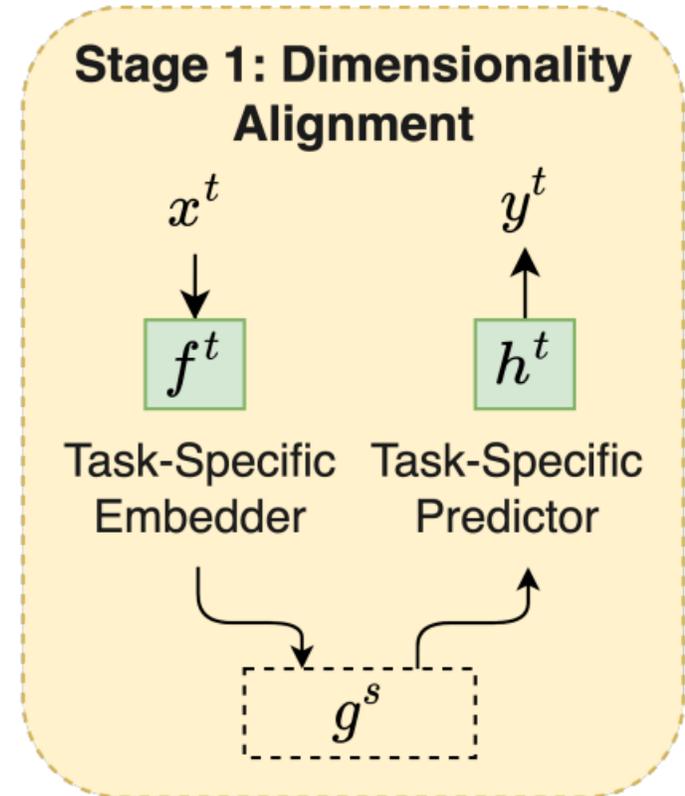
- Adds: distribution alignment step (align then refine)



ORCA: Stage 1

Let's understand each stage of ORCA

- Stage 1: compatibility for inputs and outputs
- Custom input and output embedders that depend on the task
 - Input example: convolutional layers for image settings
 - Output example: average pooling+linear layer for classification

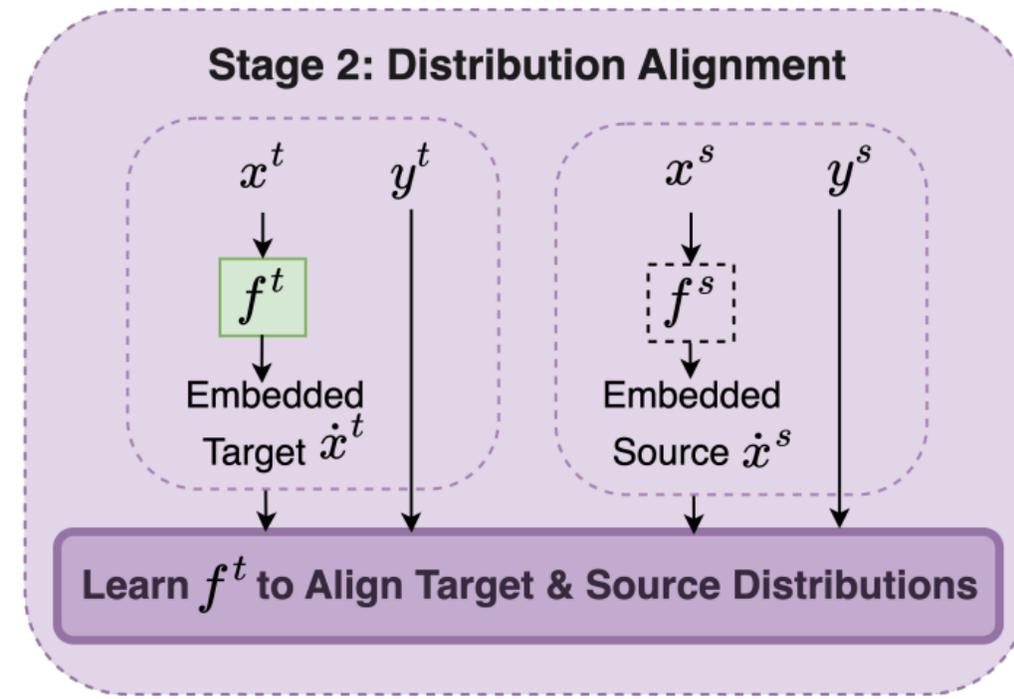


ORCA: Stage 2

Let's understand each stage of ORCA

- Stage 2: distribution alignment
- Intuition:
 - Change embeddings so target features **resemble** source features
- Learn the function f^t that **minimizes distance between**
 $(f^t(x^t), y^t)$ and $(f^s(x^s), y^s)$

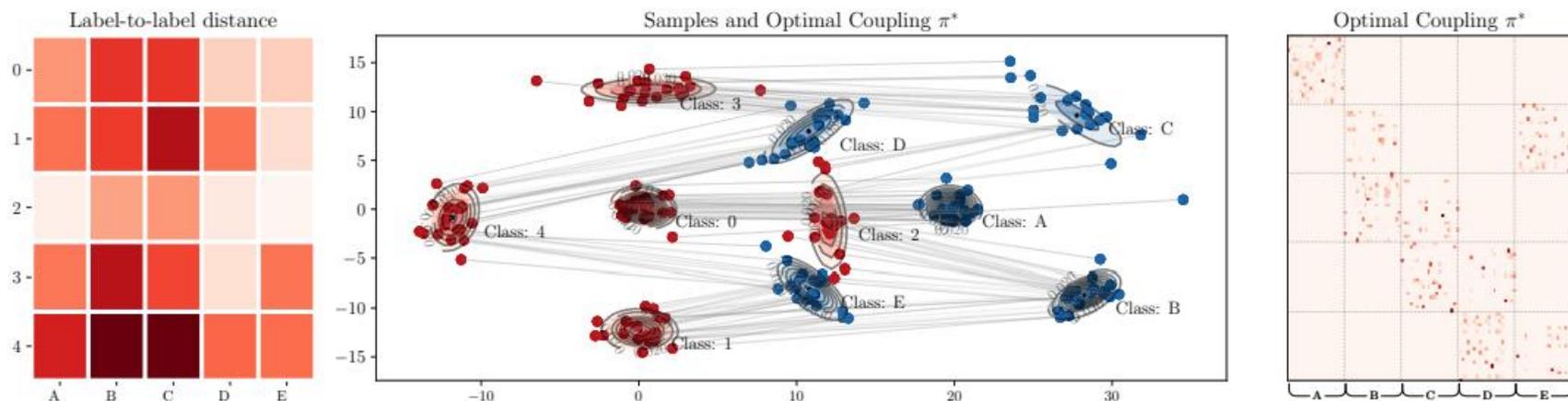
Warning: this part is math-heavy, but we just need the intuition



ORCA: Distributional Distances

Want: learn the function f^t that minimizes distance between $(f^t(x^t), y^t)$ and $(f^s(x^s), y^s)$

- How?
- Need a distance function on these distributions
- Here, **optimal transport dataset distance (OTDD)**



Interlude: Optimal Transport

In optimal transport, we solve

$$\inf \left\{ \int_{X \times Y} c(x, y) d\gamma(x, y) \mid \gamma \in \Gamma(\mu, \nu) \right\},$$

Cost or distance
of moving x to y

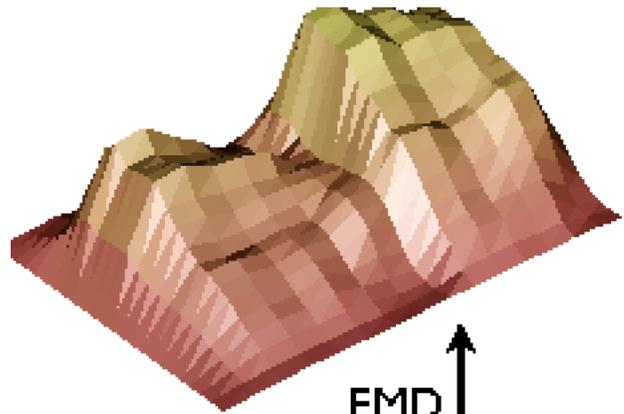
The two **marginals** we care
about, i.e., on x and y

- Want to “move” distribution on x to one on y
 - Output is a joint distribution with the original source and target
- But there’s a cost to moving x to y , given by $c(x, y)$

Interlude: Optimal Transport

In optimal transport, we solve

$$\inf \left\{ \int_{X \times Y} c(x, y) d\gamma(x, y) \mid \gamma \in \Gamma(\mu, \nu) \right\},$$



EMD
↕



Cost or distance
of moving x to y

The two **marginals** we care
about, i.e., on x and y

Applegate et al '11

Interlude: **Optimal Transport**

In optimal transport, we solve

$$\inf \left\{ \int_{X \times Y} c(x, y) \, d\gamma(x, y) \mid \gamma \in \Gamma(\mu, \nu) \right\},$$

- Cost given by **distance**: Wasserstein distance
- Gives a distance on distributions, i.e.,

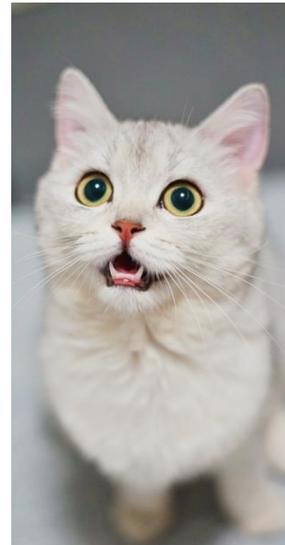
$$W_p(\mu, \nu) = \left(\inf_{\gamma \in \Gamma(\mu, \nu)} \mathbf{E}_{(x, y) \sim \gamma} d(x, y)^p \right)^{1/p}$$

Interlude: Dataset Distance

What should this cost/distance $c(x,y)$ be for us?

- For inputs x , pretty easy: feature vectors in spaces that have distances, e.g., $\|x-x'\|$

- For outputs y , not so easy



—



- A clever idea:

- Replace y with $P(X|y)$

- Even harder? No, just use Wasserstein: $W(P(X|y),P(X|y'))$

- Approximate this with a Gaussian: closed form too!

ORCA: Distributional Distances

Want: learn the function f^t that minimizes distance between $(f^t(x^t), y^t)$ and $(f^s(x^s), y^s)$

- Need a distance function on these distributions
- Here, **optimal transport dataset distance (OTDD)**

$$d_{\mathcal{Z}}((x, y), (x', y')) \triangleq \left(d_{\mathcal{X}}(x, x')^p + \mathbf{W}_p^p(\alpha_y, \alpha_{y'}) \right)^{1/p}$$



i.e., Euclidean
distance

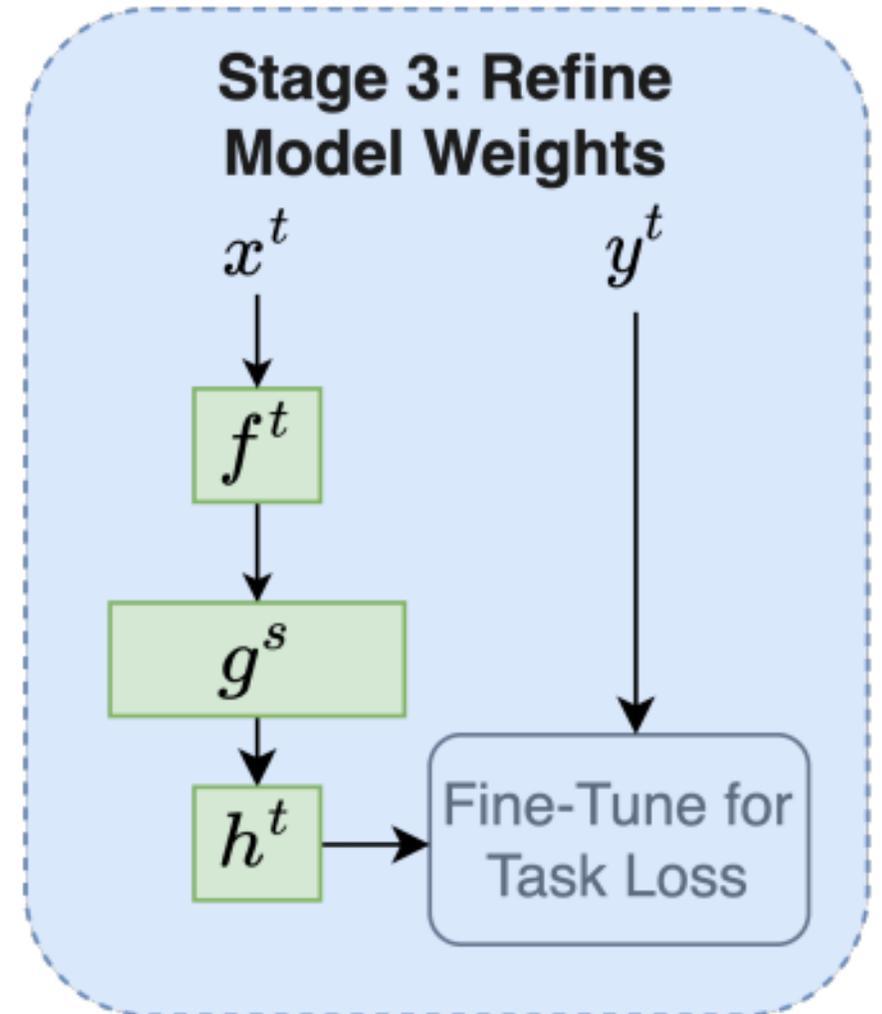


p-Wasserstein distance on
 $P(x|y)$

ORCA: Stage 3

Let's understand each stage of ORCA

- Stage 3: fine-tune the input and output network weights
 - For particular tasks
 - Or, could do any other variant of what we've talked about...



ORCA: Results

Extremely good, even against state-of-the-art results

- Compare to Neural Architecture Search (NAS)
 - Produces custom architectures that hit state-of-the-art for various tasks
 - Same procedure on many types of tasks works well:

	CIFAR-100 0-1 error (%)	Spherical 0-1 error (%)	Darcy Flow relative ℓ_2	PSICOV MAE ₈	Cosmic 1-AUROC	NinaPro 0-1 error (%)	FSD50K 1- mAP	ECG 1 - F1 score	Satellite 0-1 error (%)	DeepSEA 1- AUROC
Hand-designed	19.39	67.41	8E-3	3.35	0.127	8.73	0.62	0.28	19.80	0.30
NAS-Bench-360	23.39	48.23	2.6E-2	2.94	0.229	7.34	0.60	0.34	12.51	0.32
DASH	24.37	71.28	7.9E-3	3.30	0.19	6.60	0.60	0.32	12.28	0.28
Perceiver IO	70.04	82.57	2.4E-2	8.06	0.485	22.22	0.72	0.66	15.93	0.38
FPT	10.11	76.38	2.1E-2	4.66	0.233	15.69	0.67	0.50	20.83	0.37
ORCA	6.53	29.85	7.28E-3	1.91	0.152	7.54	0.56	0.28	11.59	0.29



Break & Questions

Outline

- **Fine-Tuning and Adapters**

- Fine-tuning vs. prompting, linear probing, etc. Full vs partial fine tuning vs adapting. Popular adapters

- **Cross-Modal Adaptation**

- Frozen transformers, ORCA, aligning via optimal transport dataset distance

- **Foundation Models + Tools**

- Beyond unaided LLMs and FMs, simple tool integration, Toolformer

Beyond Unaided Language Models

Even when we fine-tune or optimize prompts (or do CoT), the language model can get things wrong.

- Often simple things... like **arithmetic**.

Data Formatting

Plain

128+367=495

Reverse

\$128+367=594\$

Simplified Scratchpad

Input: 128+367

Target:

A->5, C->1

A->9, C->0

A->4, C->0.

495

Detailed Scratchpad

Input:

128+367

Target:

<scratch>

[1,2,8] has 3 digits.

[3,6,7] has 3 digits.

[1,2,8] + [3,6,7], C=0, 8+7+0=15, A->5, C->1

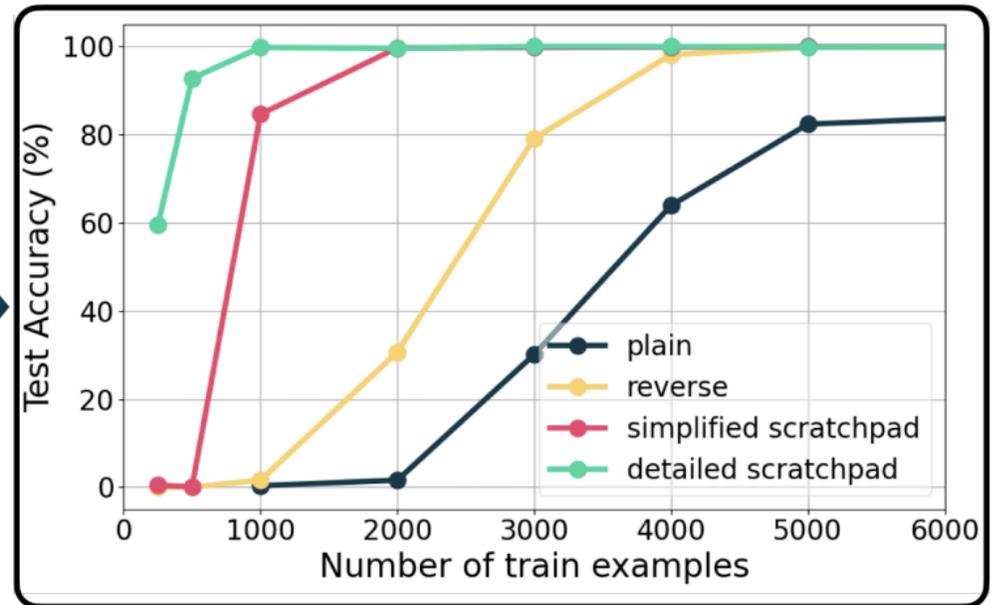
[1,2] + [3, 6], A= [5], 2+6+1=9, A->9, C->0

[1] + [3], A= [9,5], C=0, 1+3+0=4, A->4, C->0

[] + [], A= [4,9,5], C=0, END

</scratch>

4 9 5



Beyond Unaided Language Models

Even when we fine-tune or optimize prompts (or do CoT), the language model can get things wrong.

- Often simple things... like **arithmetic**.
- How else can we help it?
- **A:** Use external tools



Tools: Program-aided LMs

Use external tools:

- Python interpreter
- How? *Interleave* the text explanations in CoT steps with lines of Python code
- LMs can already output code
 - Just need to *prompt* the right way

Chain-of-Thought (Wei et al., 2022)

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 tennis balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?

Model Output

A: The bakers started with 200 loaves. They sold 93 in the morning and 39 in the afternoon. So they sold $93 + 39 = 132$ loaves. The grocery store returned 6 loaves. So they had $200 - 132 - 6 = 62$ loaves left. The answer is 62.



Gao et al '23

Program-aided Language models (this work)

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 tennis balls.
`tennis_balls = 5`
`2 cans of 3 tennis balls each is`
`bought_balls = 2 * 3`
`tennis_balls`. The answer is
`answer = tennis_balls + bought_balls`

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?

Model Output

A: The bakers started with 200 loaves
`loaves_baked = 200`
`They sold 93 in the morning and 39 in the afternoon`
`loaves_sold_morning = 93`
`loaves_sold_afternoon = 39`
`The grocery store returned 6 loaves.`
`loaves_returned = 6`
The answer is
`answer = loaves_baked - loaves_sold_morning`
`- loaves_sold_afternoon + loaves_returned`

`>>> print(answer)`
74



Tools: Program-aided LMs Prompts

Just need to *prompt* the right way

- Craft examples that interleave code and text

Q: Olivia has \$23. She bought five bagels for \$3 each. How much money does she have left?

```
money_initial = 23
bagels = 5
bagel_cost = 3
money_spent = bagels * bagel_cost
money_left = money_initial - money_spent
answer = money_left
```

Figure 3: Example prompt for the mathematical reasoning tasks, from the GSM8K benchmark.

Q: On the table, you see a bunch of objects arranged in a row: a purple paperclip, a pink stress ball, a brown keychain, a green scrunchiephone charger, a mauve fidget spinner, and a burgundy pen. What is the color of the object directly to the right of the stress ball?

```
...
stress_ball_idx = None
for i, object in enumerate(objects):
    if object[0] == 'stress ball':
        stress_ball_idx = i
        break
# Find the directly right object
direct_right = objects[stress_ball_idx+1]
# Check the directly right object's color
answer = direct_right[1]
```

Tools: Program-of-Thoughts

Similar idea: program-of-thoughts

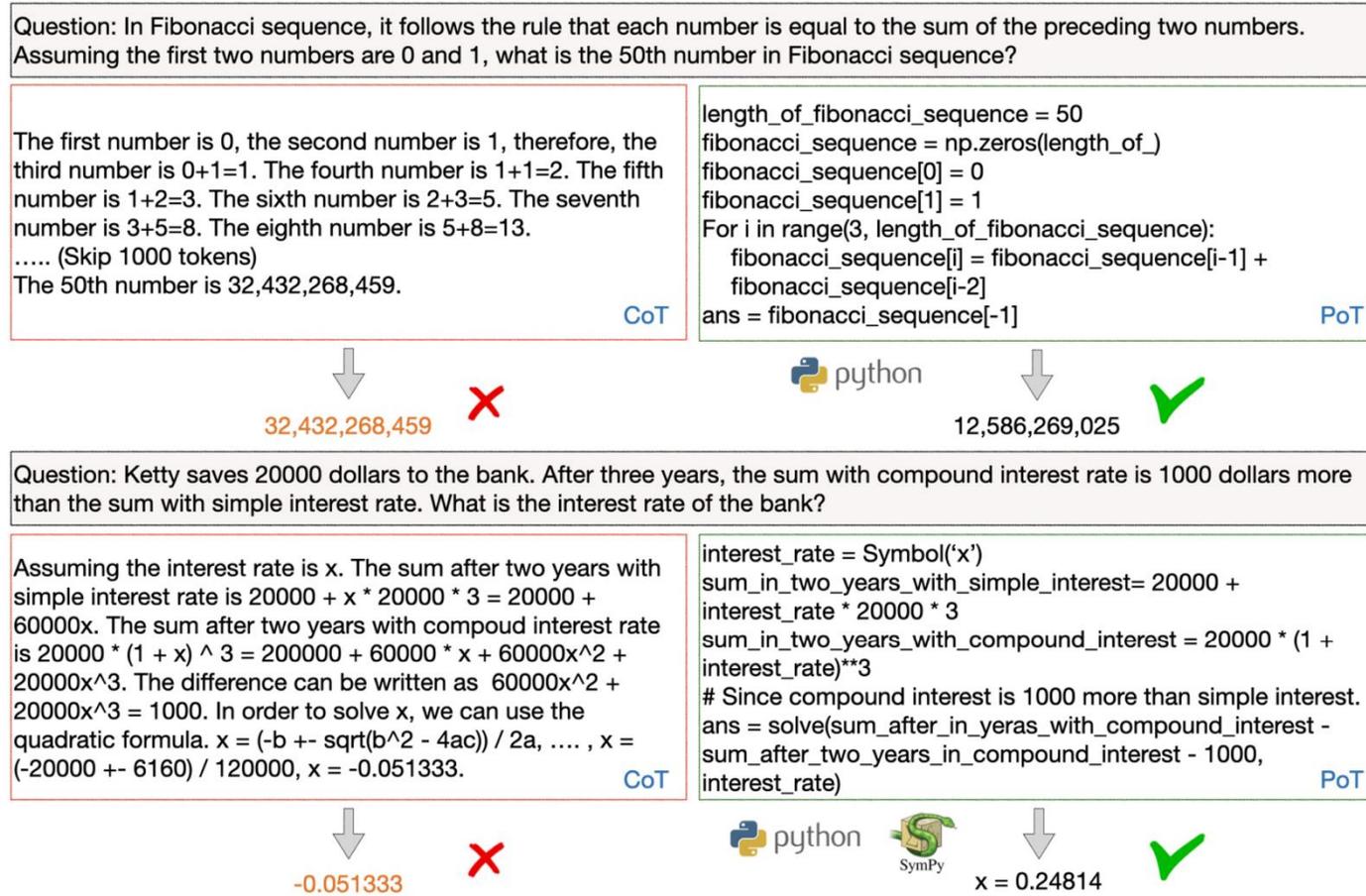


Figure 1: Comparison between Chain of Thoughts and Program of Thoughts.

Tools: More General Tools

Ideally, use more general external tools

- Without lots of human annotation
- Model should decide on its own which tool to use
- **Toolformer**: introduces API calls into the model
 - But these API calls aren't already there... so need to **fine-tune**

Your task is to add calls to a Question Answering API to a piece of text. The questions should help you get information required to complete the text. You can call the API by writing "[QA(question)]" where "question" is the question you want to ask. Here are some examples of API calls:

Input: Joe Biden was born in Scranton, Pennsylvania.

Output: Joe Biden was born in [QA("Where was Joe Biden born?")] Scranton, [QA("In which state is Scranton?")] Pennsylvania.

Input: Coca-Cola, or Coke, is a carbonated soft drink manufactured by the Coca-Cola Company.

Output: Coca-Cola, or [QA("What other name is Coca-Cola known by?")] Coke, is a carbonated soft drink manufactured by [QA("Who manufactures Coca-Cola?")] the Coca-Cola Company.

Input: x

Output:



Thank You!