Modern LLM Recipe: Efficient Adaptation

EECS 183/283a: Natural Language Processing

Recipe So Far…

1. Pretraining
    1. Collect training documents and preprocess them
    2. Optimize language modeling objective
    3. **Outcome:** really good *language* model, but:
        1. Doesn’t have a useful (natural language) interface
        2. Doesn’t necessarily exhibit desired behavior (alignment)

Recipe So Far…

2. Posttraining
    1. Instruction-tuning to solve the interface problem
        1. Collect examples of natural language instructions paired with demonstrations
        2. Fine-tune base language model to generate response conditioned on instruction
    2. Reinforcement learning from human feedback to solve the alignment problem
        1. For some new instructions, sample candidate responses from the instruction-tuned model
        2. Ask human annotators to rank the set of candidates
        3. Train a reward model to, for a pair of candidates, assign a higher score to the candidate that the annotator ranked higher
        4. Fine-tune the instruction-tuned model via RL, using reward model
3. **Outcome:** model that follows natural language instructions directly

Recipe So Far…

- What's left?
- For complex tasks, model may not "know" the best way to solve it
- Model might be bad at some target task, for example:
  - Really challenging math problems
  - Very domain-specific problems, e.g., medical reasoning, new programming languages, etc.
- Running inference may be inefficient or impossible due to model size
- **Can we solve these problems without respending all of the compute we used to get our instruction-tuned model?**

Inference-Time Adaptation

- Too expensive to fine-tune a model?
- Too little (or no) data available for fine-tuning?
- No access to model weights?
- No access to output probabilities?
- No problem

Recall: In-Context Learning

Zero-shot prompting (base LM)

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

The restaurant had 15 oranges. If they used 2 to make dinner and
bought 3 more, how many oranges do they have?
```

Few-shot prompting

```
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?

A: The answer is 27.
```

prompt, task input, model output

Wei et al. 2022

Chain-of-Thought Prompting

Main idea: "prime" model to generate step-by-step solution to input problem

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?

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.

prompt, task input, model output

Wei et al. 2022

Zero-Shot Chain-of-Thought

Main idea: format input to prime model to generate a step-by-step solution

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

A: Let's think step by step. 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.

prompt, task input, model output

Kojima et al. 2022

Zero-Shot Chain-of-Thought

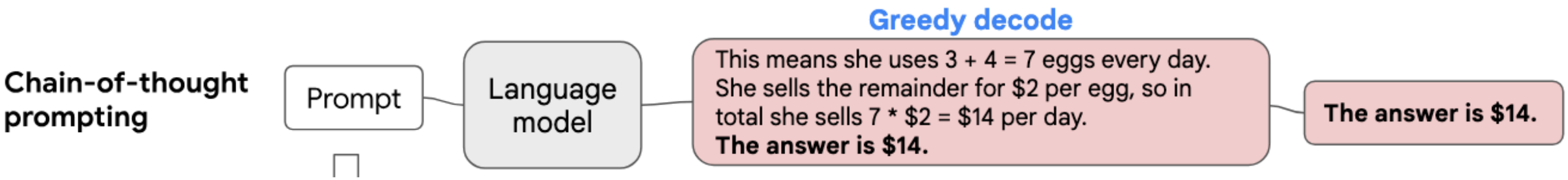
Main idea: format input to prime model to generate a step-by-step solution

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

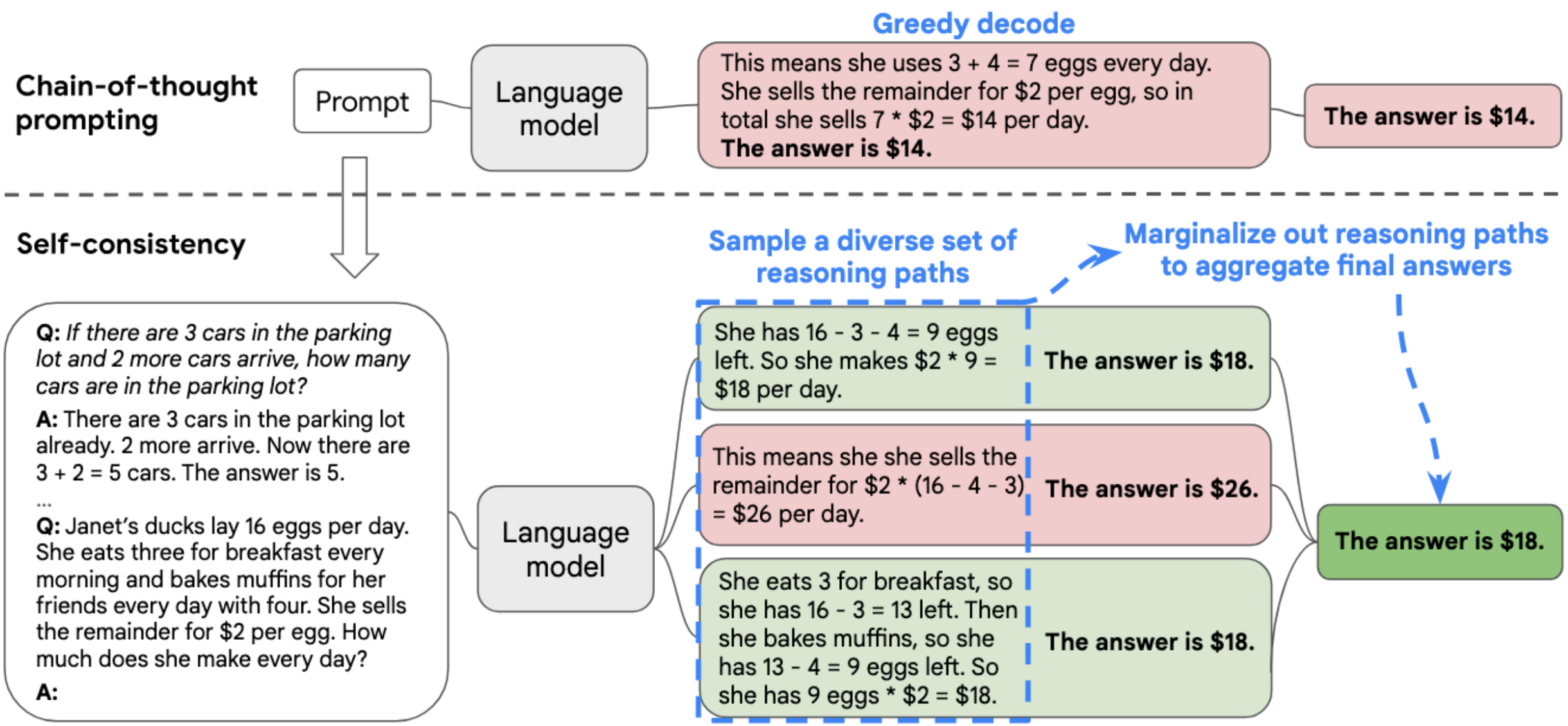
Structured Prompting

Self-Consistency



Structured Prompting

Self-Consistency



Wang et al. 2023

Structured Prompting

- Why are we expecting the models to do arithmetic directly? Why not just give them a calculator?
- Main idea: prompt LMs to "call" tools, e.g., by interleaving language output with calls to a calculator:

```
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 = 9

The answer is
answer = loaves_baked - loaves_sold_morning -
         loaves_sold_afternoon + loaves_returned
```

prompt, model text output, model program output

Gao et al. 2022

Prompt Tuning

- Instead of designing a prompting method ourselves, why not train a model to do it?

- **Training data:** examples from our target task

- **Goal:** use the training data to find a prompt that, for some particular model, we perform as well as possible on held-out task data

Discrete Prompt Tuning

- Space of prompts: sequences of wordtypes! $\mathcal{V}^\dagger$
- Goal during training: find a prompt that maximizes some reward (e.g., accuracy) over the training dataset

$$\arg\max_{p \in \mathcal{V}^\dagger} \mathop{\mathbb{E}}_{x \in \mathcal{D}} \mathcal{R}(y \sim \mathrm{LLM}(px))$$

Discrete Prompt Tuning

- Space of prompts: sequences of wordtypes! $\mathcal{V}^\dagger$
- Goal during training: find a prompt that maximizes some reward (e.g., accuracy) over the training dataset

$$\arg\max_{p \in \mathcal{V}^\dagger} \mathop{\mathbb{E}}_{x \in \mathcal{D}} \mathcal{R}(y \sim \mathrm{LLM}(px))$$

- How to optimize?
- Reinforcement learning!

Discrete Prompt Tuning

ID	Template [to negative | to positive]	Content	Style	Fluency	$J(C, S, F)$	$GM(C, S, F)$	BLEU	BERTScore	PPL↓
Null Prompt									
1	"{input}" "	37.4 (0.1)	94.8 (0.1)	**97.6 (0.1)**	33.6 (0.1)	70.2 (0.1)	6.6 (0.1)	35.8 (0.1)	59.5 (2.0)
Manual Prompt									
1	Here is some text: "{input}". Here is a rewrite of the text, which is more [negative | positive]: "	72.1 (0.1)	94.8 (0.3)	91.6 (0.1)	62.3 (0.2)	**85.6 (0.1)**	23.9 (0.1)	58.8 (0.1)	**29.6 (0.3)**
2	Change the following sentence from [positive | negative] sentiment to [negative | positive] sentiment but keep its semantics. "{input}" "	60.4 (0.1)	91.9 (0.2)	94.0 (0.1)	50.5 (0.1)	80.5 (0.1)	17.4 (0.1)	51.3 (0.1)	31.0 (0.4)
3	"{input}". Rewrite the sentence to be [sadder | happier] but have the same meaning. "	60.2 (0.2)	87.7 (0.4)	94.0 (0.2)	47.4 (0.3)	79.2 (0.1)	16.2 (0.1)	49.3 (0.1)	45.8 (0.7)

Deng et al. 2022

Discrete Prompt Tuning

Fluent Prompt									
1	[I don't like having | I love my life (] "{input}" "	54.1 (0.5)	95.2 (0.4)	93.9 (0.7)	47.4 (0.4)	78.5 (0.3)	13.4 (0.4)	45.7 (0.2)	52.3 (1.9)
2	[This is not an example | The best is good\n] "{input}" "	51.5 (0.1)	**96.8 (0.4)**	94.2 (0.6)	46.0 (0.4)	77.7 (0.1)	11.9 (0.3)	46.2 (0.2)	35.4 (2.3)
3	[I don't like | I love my work (] "{input}" "	51.5 (0.4)	96.6 (0.7)	95.7 (0.5)	46.7 (0.5)	78.1 (0.2)	12.3 (0.3)	46.2 (0.3)	43.5 (1.3)
RLPROMPT *(Ours)*									
1	[Fixed (– contrasts (– contrasts | Dutch English excellent Correct (>] "{input}" "	71.5 (0.1)	96.6 (0.2)	90.1 (0.2)	**62.8 (0.9)**	85.4 (0.1)	23.5 (0.1)	58.7 (0.1)	34.1 (0.2)
2	[Fixed RemovedChanged Prevent outcomes | Parameters Comparison)=(Compare either] "{input}" "	71.0 (0.1)	91.9 (0.3)	89.3 (0.2)	58.9 (1.1)	83.5 (0.1)	23.7 (0.1)	58.3 (0.1)	35.3 (0.5)
3	[Affect differed judgments (– analysis | Difference experiences (– contrasting experience] "{input}" "	**73.8 (0.1)**	94.0 (0.2)	89.2 (0.2)	62.6 (1.1)	85.2 (0.1)	**25.6 (0.1)**	**59.9** (0.1)	33.5 (0.5)

Deng et al. 2022

Continuous Prompt Tuning

- Task: summarization
- One possible (probably suboptimal) prompt: “summarize”
- What is this like for the network?

Continuous Prompt Tuning

- What if our “prompts” are just embeddings in the same space as all of the other wordtypes?

- Optimize:

$$\arg\max_{p \in \mathbb{R}^d} \mathop{\mathbb{E}}_{(x,y)\in\mathcal{D}} \mathrm{LLM}(y \mid [p; \phi(x)])$$

- At inference time, always prepend embedding *p* to inputs

Continuous Prompt Tuning

- Initialize prompt embeddings with pretrained embeddings corresponding to the task (e.g., “summarize”)

- Embeddings are very small, and we don’t need to finetune any model parameters, so easy to learn

- However, it could be slower to converge than fully finetuning a model (why?)

Prompt Sensitivity

Prompt 1

```
Review: <negative review>
Answer: Negative

Review: <positive review>
Answer: Positive
```

88.5%

Prompt 2

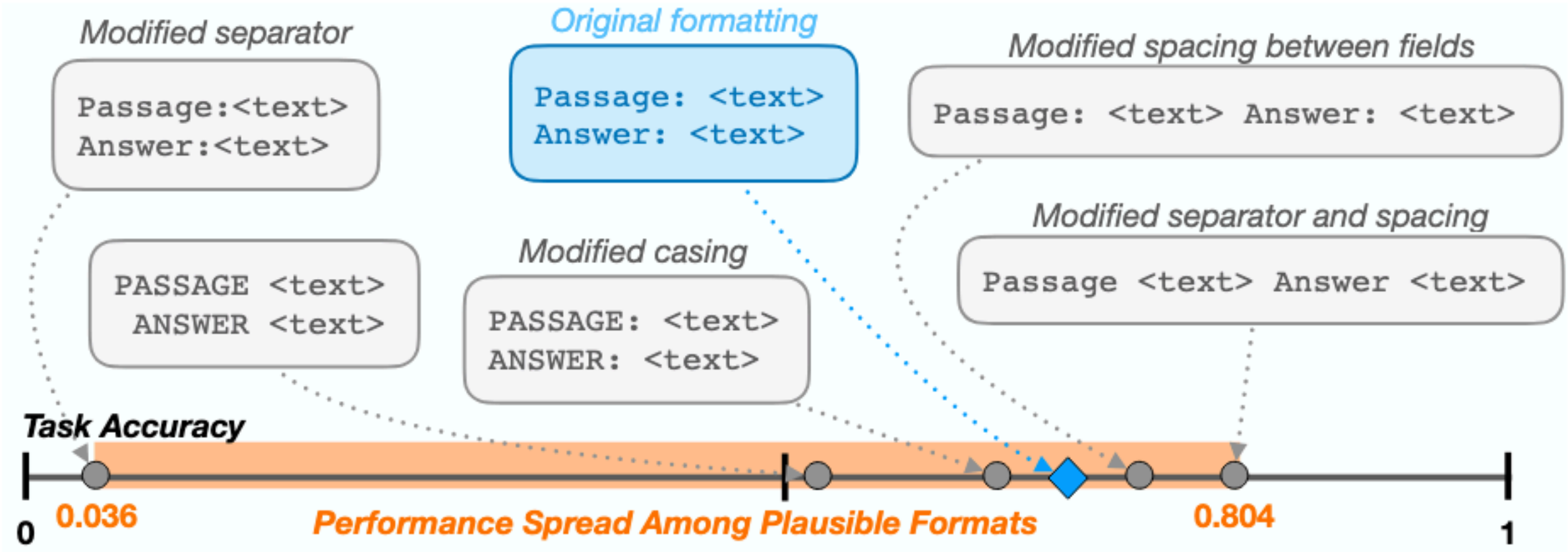
```
Review: <positive review>
Answer: Positive

Review: <negative review>
Answer: Negative
```

51.3%

Golen et al. 2022

Prompt Sensitivity

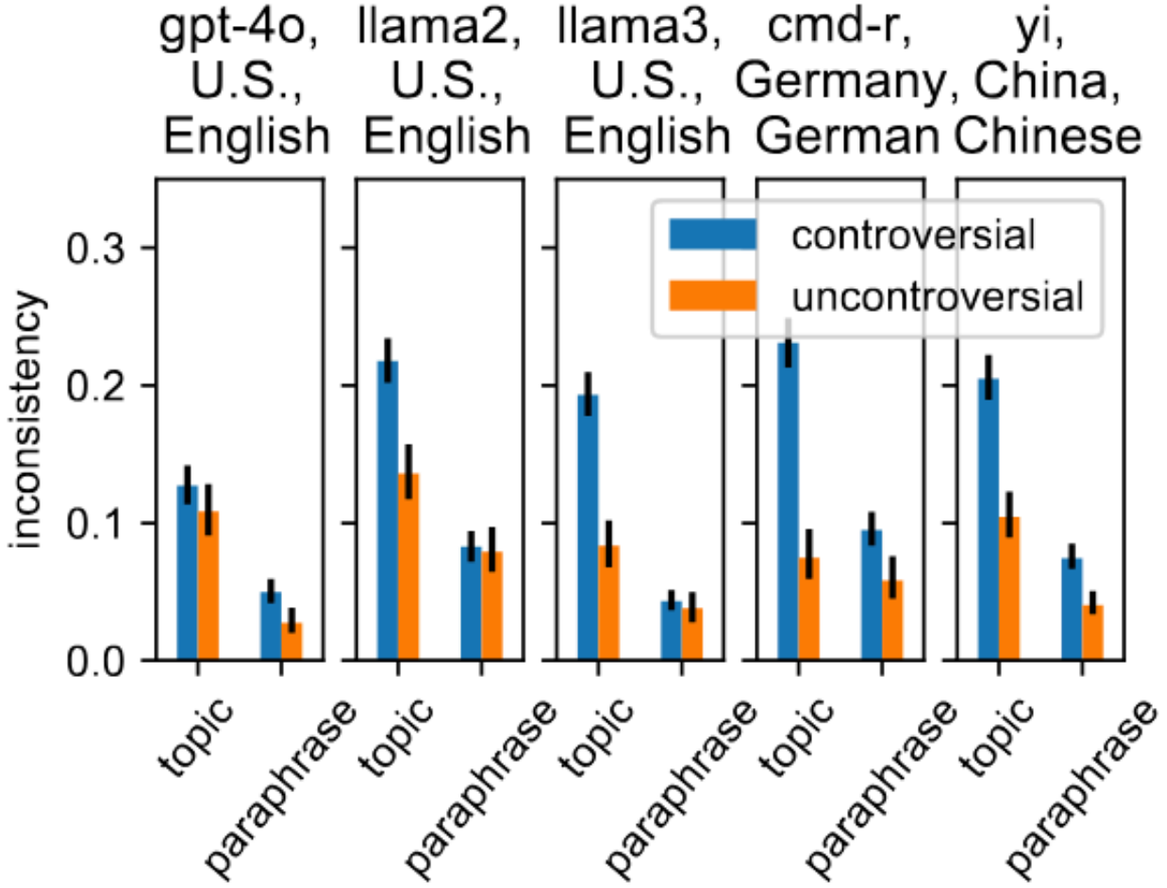
- Lots of possible features in prompt design…
- Formatting and wording
- Choices of few-shot examples, labels provided with examples, ordering of examples

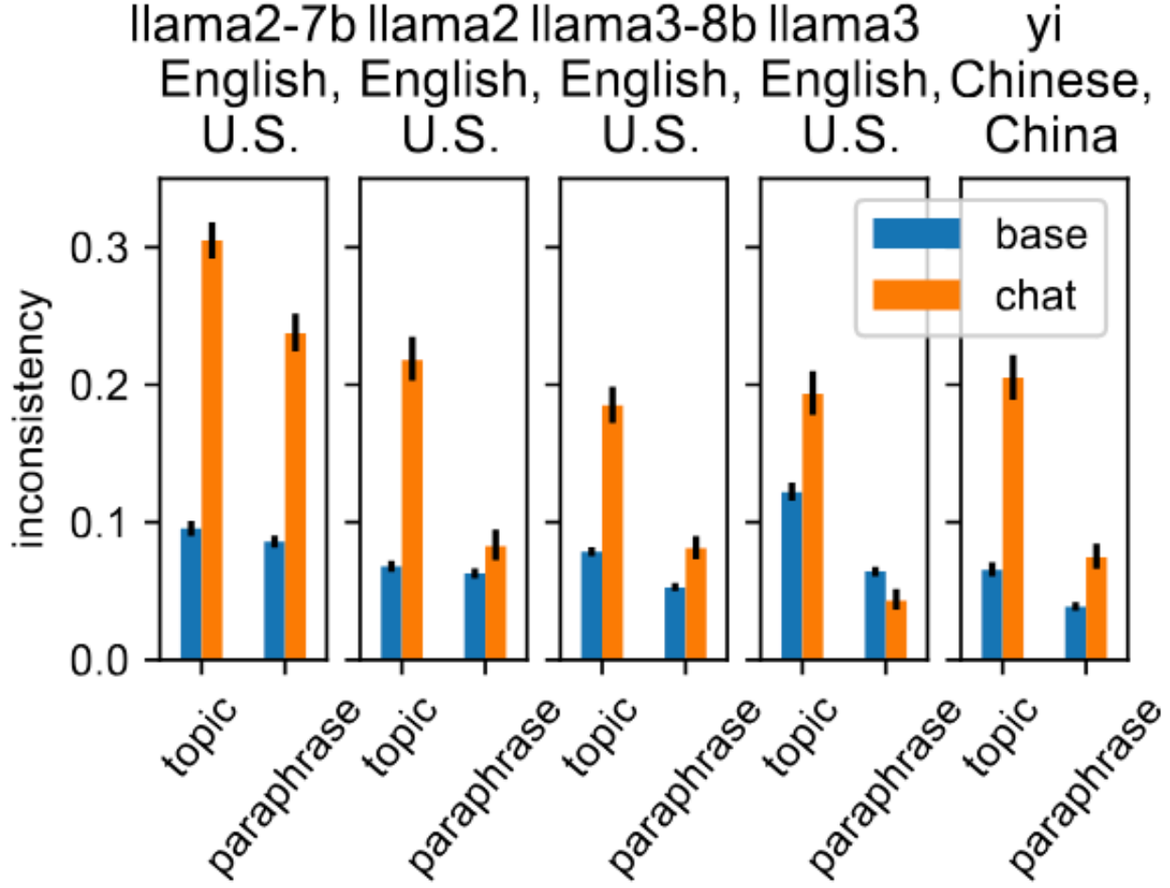


Sclar et al. 2023

Prompt Sensitivity

- Models are more sensitive to prompt changes for controversial vs. non-controversial social questions
- Chat (instruction-tuned) models are more sensitive than base models!





Moore et al. 2024

Back to the Interface Problem…

- If post-trained models are optimized to follow user instructions, why do these methods work so much better than simple instruction?
  - Chain-of-thought prompting
  - Self-consistency
  - Program-aided language models
- If models are so sensitive to trivial changes to prompt design… have we solved the interface problem?

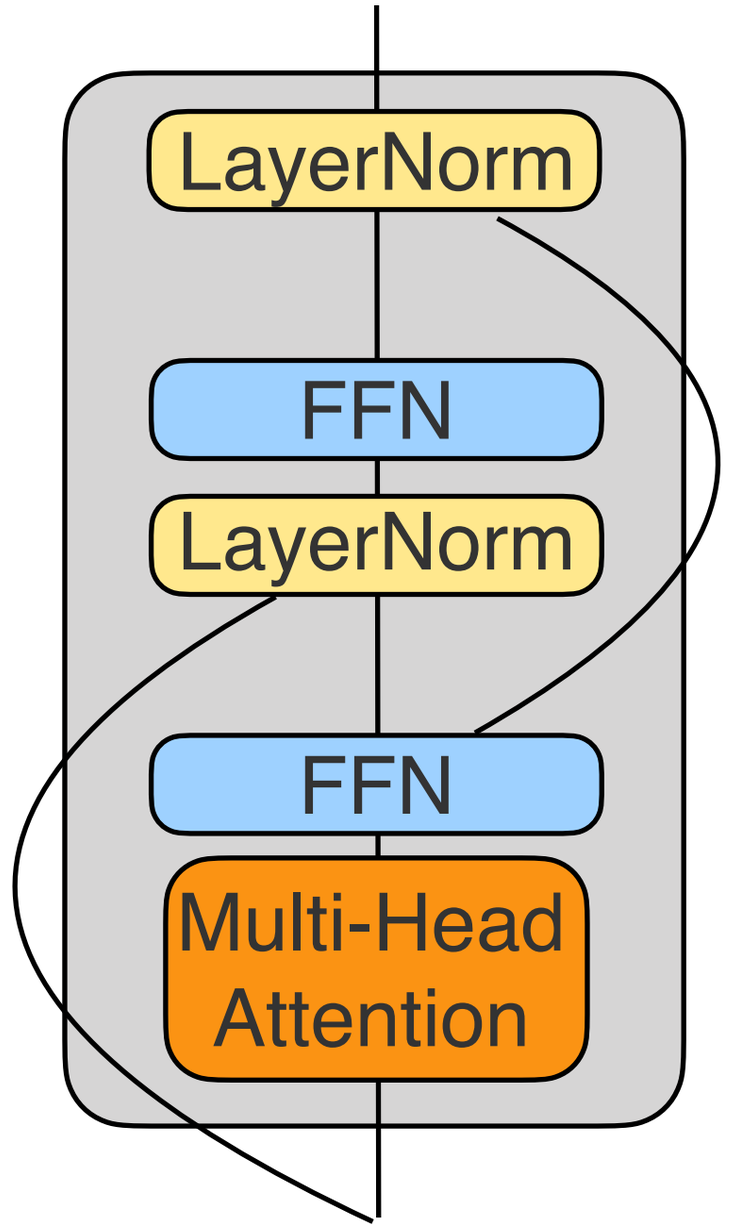
Fine-Tuning

- **Main problem:** model capabilities are bounded by their training data
  - Pretraining data
  - Instruction-tuning data
  - Preference data
- If a particular task doesn't have adequate support in the training data, the resulting model won't have the target capability
- Instead, we should finetune the model directly

Full Fine-Tuning

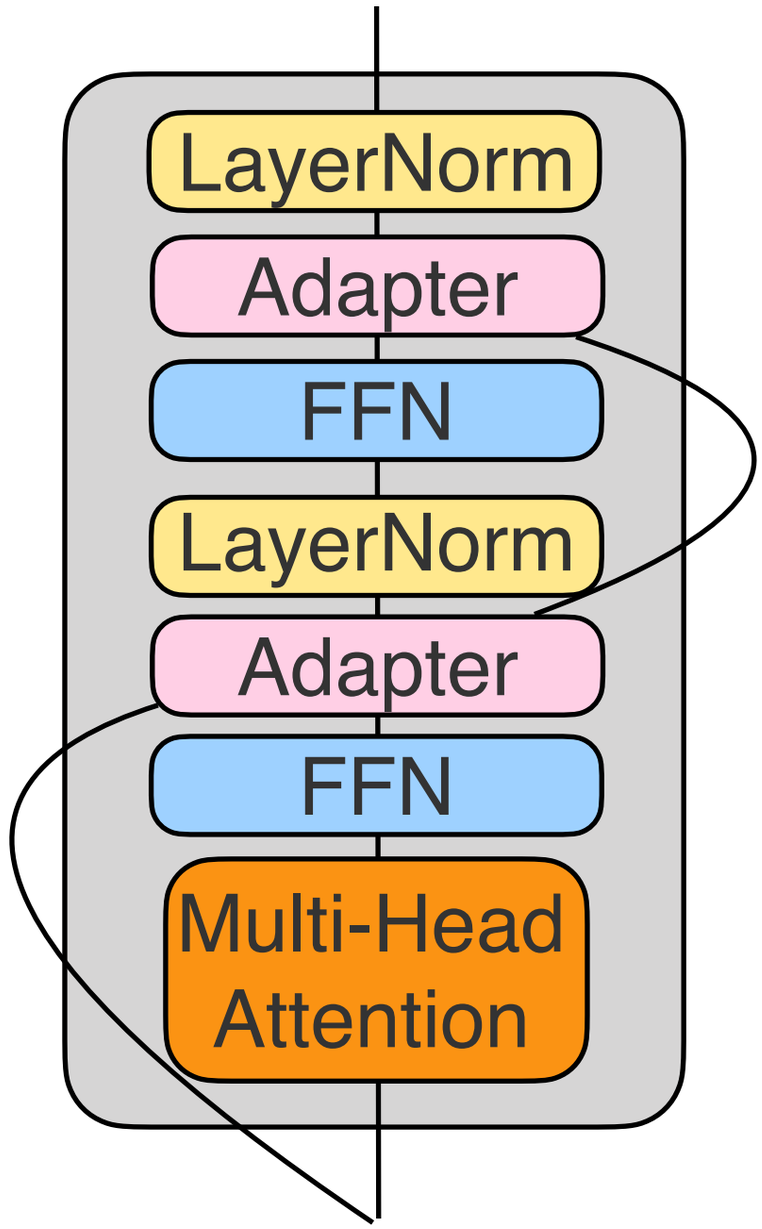
- Yet another phase of fine-tuning, except this time we train on input/output pairs from our target task
- Basic setup: allow all model parameters to be updated
- However, this can be expensive (why?)
- Instead, to speed up convergence, we can “freeze” a subset of parameters (“parameter-efficient fine-tuning”, PEFT)
  - Keep their values fixed during fine-tuning (though allowing backpropagation through them)
  - Which parameters to freeze? Some work proposes to just learn a second network from scratch whose parameters represent a “diff” of the original network, regularized to have values of mostly 0 (DiffPruning, Guo et al. 2021)

Adapters



Adapters

- Modify the network directly by injecting additional parameters into transformer cells
- Initialize the adapter as an identity function
- Finetune **only** the adapter parameters, keeping everything else frozen
- Pretty fast to train (especially compared to full fine-tuning)
- But adding layers makes the model larger, and inference slower



Low-Rank Adaptation (LoRA)

- Let's say we want to fine-tune some weight matrix $W \in \mathbb{R}^{d \times k}$
- We can express the new value as $W' = W + \Delta W$
  - In DiffPruning: we'd just learn ΔW directly
- Can we learn even fewer parameters?

$$\Delta W = BA$$

$$B \in \mathbb{R}^{d \times r}$$

$$A \in \mathbb{R}^{r \times k}$$

Low-rank: $r \ll \min(d, k)$

At the beginning of finetuning, initialize:

$$B = \mathbf{0}$$

$$A \sim \mathcal{N}(0, \sigma^2)$$

(so that ΔW behaves as identity function)

Low-Rank Adaptation (LoRA)

$$\Delta W = BA$$

$$B \in \mathbb{R}^{d \times r}$$

$$A \in \mathbb{R}^{r \times k}$$

Low-rank: $r \ll \min(d, k)$

- Significantly fewer parameters to fine-tune than full fine-tuning or adapters
- But still roughly approximates full fine-tuning, as long as *r* is the “intrinsic rank” of the original weight matrix
- No additional inference latency because we can precompute $W' = W + BA$
- In practice: adapt attention weights

Model Compression

- **Main problem:** model is too big!
  - Inference takes too long
  - Model doesn't fit on the device (e.g., VRAM on GPU; CPU on a mobile device)
- Can we take a big model and make it smaller?

Pruning

- Not all model parameters are necessary to keep for some target task
  - We could identify the important parameters using a binary mask: $b \in \{0,1\}^{|\theta|}$
  - Which parameters to keep?

Pruning

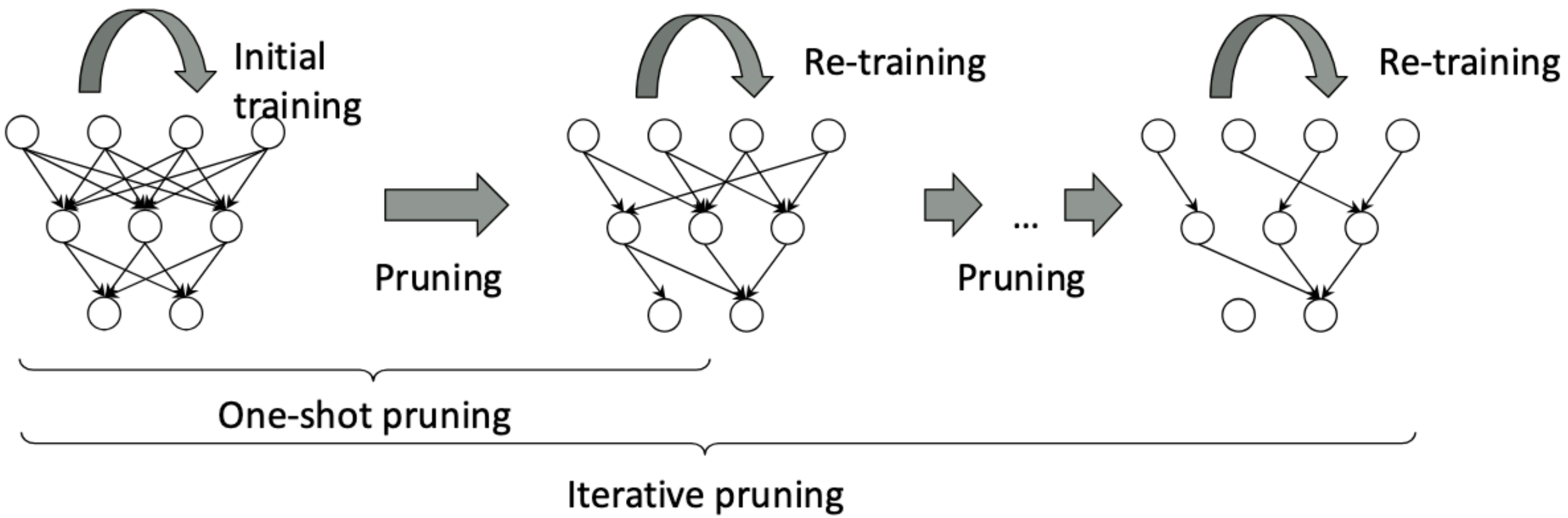
- Not all model parameters are necessary to keep for some target task
  - We could identify the important parameters using a binary mask: $b \in \{0, 1\}^{|\theta|}$
  - Which parameters to keep?

- **Lottery ticket hypothesis:** dense, randomly-initialized models contain subnetworks that, when trained in isolation, reach test accuracy comparable to the original network in a similar number of iterations

Frankle and Carbin 2019

Pruning

- Remove lowest-magnitude weights (set values to 0)
- Re-train network (freezing removed weights)
- Iterate between pruning and re-training



Frankle and Carbin 2019

Quantization

- **Main principle:** use lower-precision representations of network parameters during inference
- Reduces the space required to store the model during inference
- If your model has 65B parameters…
  - float32 (single-precision) —> 260 GB
  - float16 (half-precision) —> 130 GB
    - Usually doesn’t influence performance significantly!
  - 1-byte precision —> 65 GB
  - 1-bit precision —> 8.1 GB

Quantization-Aware Training

- Why might training quantization at inference time cause degraded performance?

- Activations at input to each layer will be increasingly out-of-distribution!

- Instead: train model to expect quantized inputs at each layer
  - Forward pass: quantize
  - Backward pass: don’t quantize

Distillation

- **Main idea:** just train a new network (possibly from scratch) on task-specific data sampled from a much larger model

- No need for access to larger model's weights or output probabilities, just its outputs

- You can get a much smaller network that you have full control over and access to!

- Why not just train on "naturally-available" task-specific data?

Model Compression

- **Quantization:** no parameters are changed (via learning, only via lowering precision), can cut model size in half without significant performance drops!

- **Pruning:** set some parameters to zero

- **Distillation:** train a smaller model using data sampled from a larger model