

# ECE7115 ~~Multimodal VLM~~ LLM

## 3. LLM Basics

Spring 2026

Namhyuk Ahn, Inha University



# Last Week: FLOPs

- Suppose we have  $N = 16384$ ,  $D_{in} = 32768$ ,  $D_{out} = 8192$
- $X = [N, D_{in}]$ ,  $W = [D_{in}, D_{out}] \# X \xrightarrow{W} \text{loss}$
- FLOPs of forward =  $2 \times N \times D_{in} \times D_{out}$
- FLOPs of backward =  $4 \times N \times D_{in} \times D_{out}$ 
  - 2 for calculating grads of  $W$  ( $w.\text{grad}[j,k] = \sum_i x[i,j] * h.\text{grad}[i,k]$ )
  - 2 for calculating grads of  $X$  ( $x.\text{grad}[i,j] = \sum_k w[j,k] * h.\text{grad}[i,k]$ )
- Total:  $6 \times (\# \text{ data or tokens}) \times (\# \text{ params})$

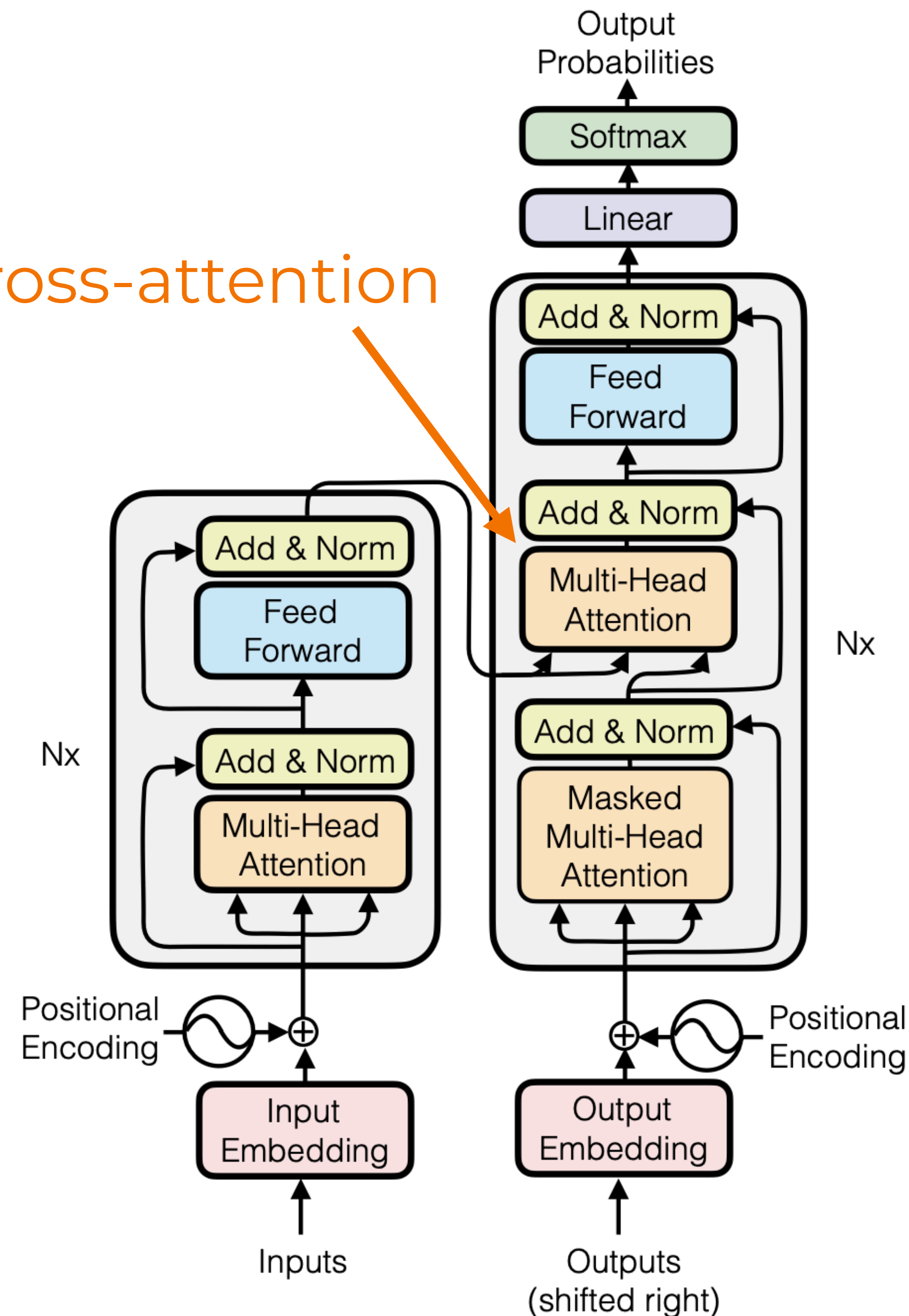
# Last Week: Memory

- Linear layer  $[D_{in} \times D_{out}]$  (no bias)
- # params =  $D_{in} \times D_{out}$
- # activations =  $N \times D_{out}$
- # grads = # params
- # optimizer states =  $2 \times \# \text{ params}$  (assume we use Adam family)
- **Total memory (bytes) =  $4 \times (4 \times \# \text{ params} + \# \text{ activations})$** 
  - 4 for fp32
  - But in reality, we use mixed precision; more complicated

# Last Week: Transformer

- We learned
  - Tokenization (word strings to indices)
  - Embedding (indices to matrix)
  - Positional Encoding (add sequence order)
  - Attentions:
    - Multi-Head Attention
    - Masked Multi-Head Attention
  - Residual Connections (standard approach)
  - Layer Normalization (increase stability)
  - Pointwise FFN (add nonlinearities)

Cross-attention



# Traditional Paradigm

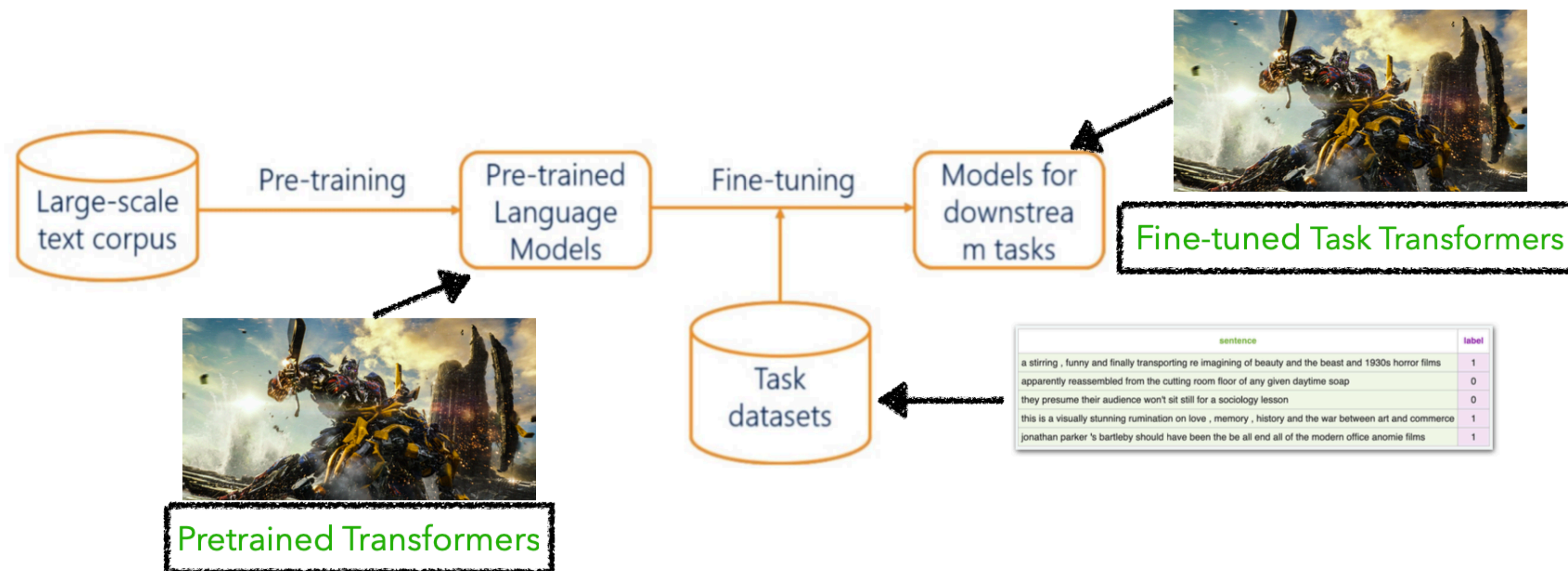
- Supervised training on a dedicated the dataset
  - Collect  $(x, y)$  task training pairs
  - Randomly initialize your model  $f(x)$  (e.g. vanilla Transformers)
  - Train  $f(x)$  on  $(x, y)$  pairs



Then you get a trained Transformer ONLY for sentiment analysis

# Modern Paradigm

- Pre-training → fine-tuning
  - **(1) Pre-training**: train Transformer using a lot of general text using unsupervised (or self-supervised) learning
  - **(2) Fine-tuning**: Train the pre-trained Transformer for a specific task using supervised learning

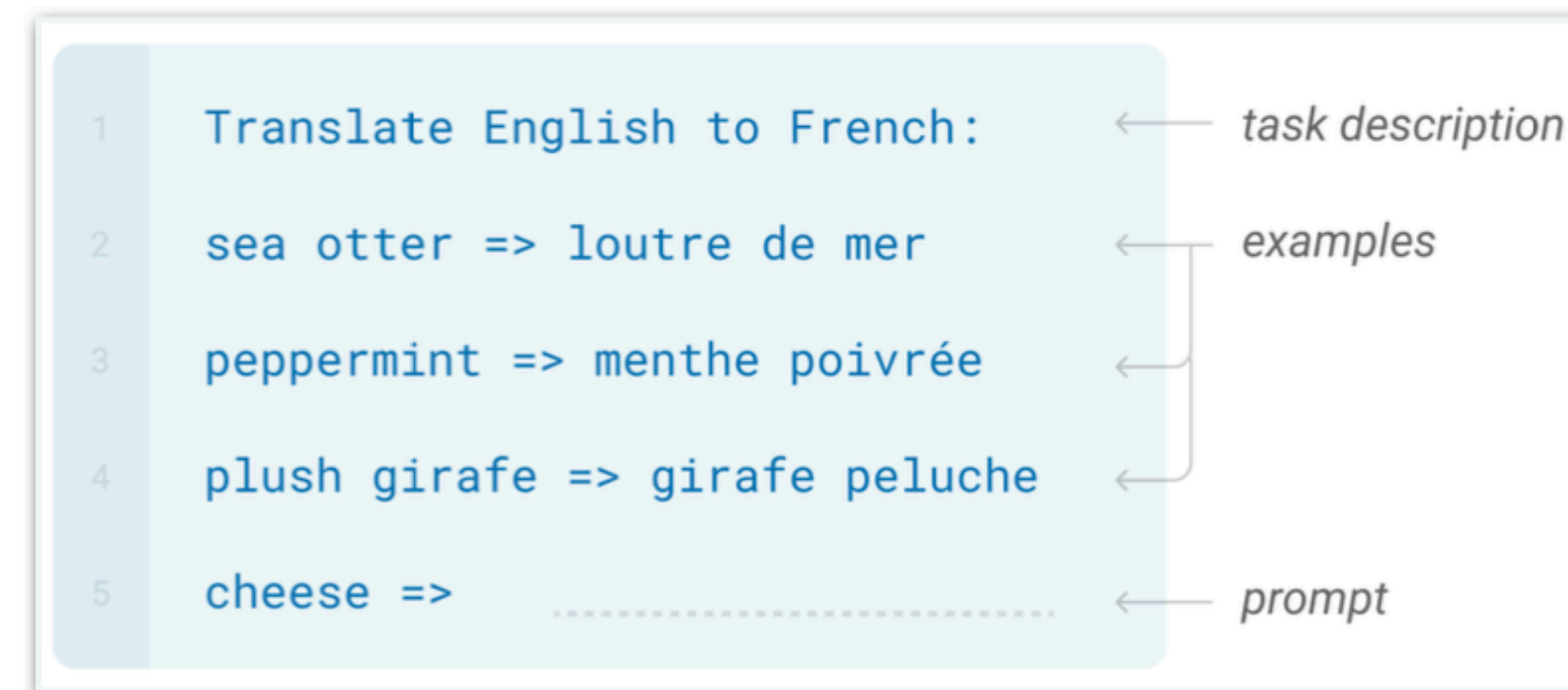


# Latest Paradigm with LLMs

- Pre-training + **prompting / in-context learning** (no training here)
  - **(1) LLM pre-training**: train a large (>7~175B) Transformer using a lot of general text using unsupervised (or self-supervised) learning
  - **(2) Prompting/in-context learning**: Then directly use the pre-trained large Transformer (no further fine-tuning/training) for any different task given only a natural language description of the task or a few task (x, y) examples



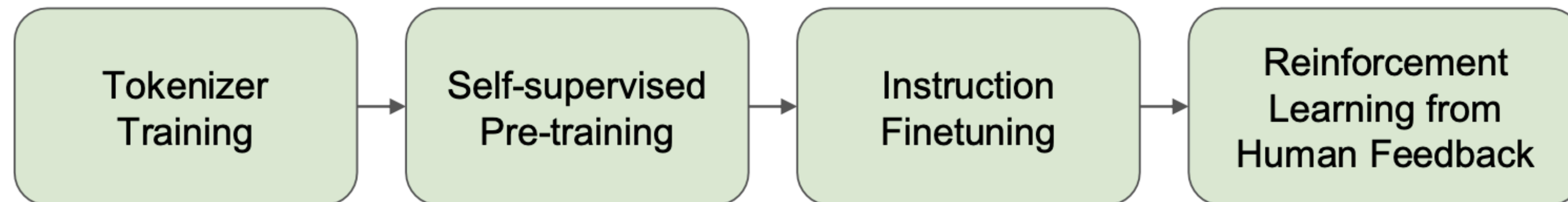
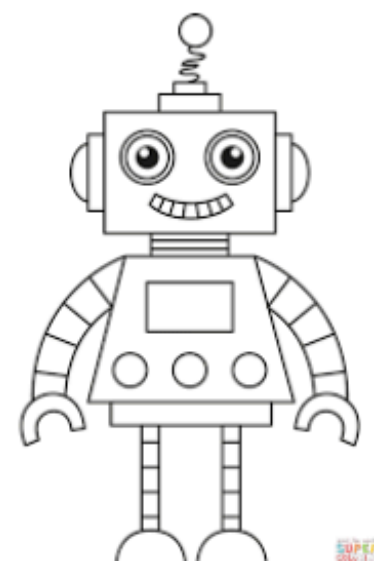
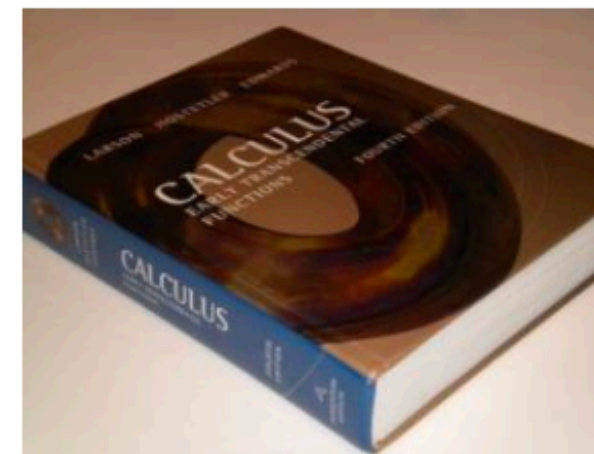
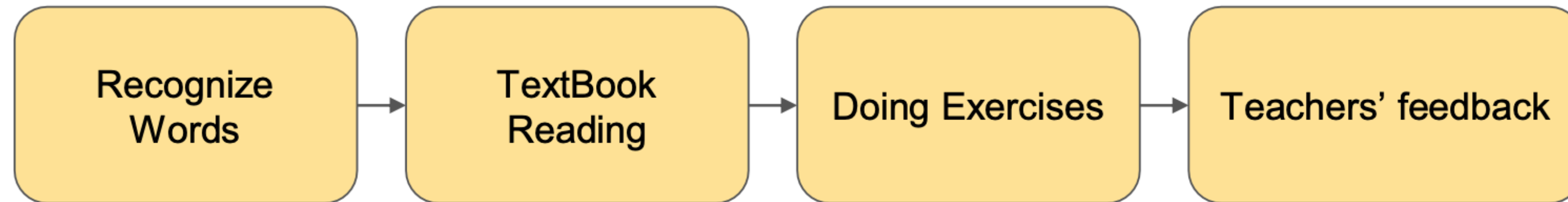
Zero-shot prompting



Few-shot prompting / in-context learning

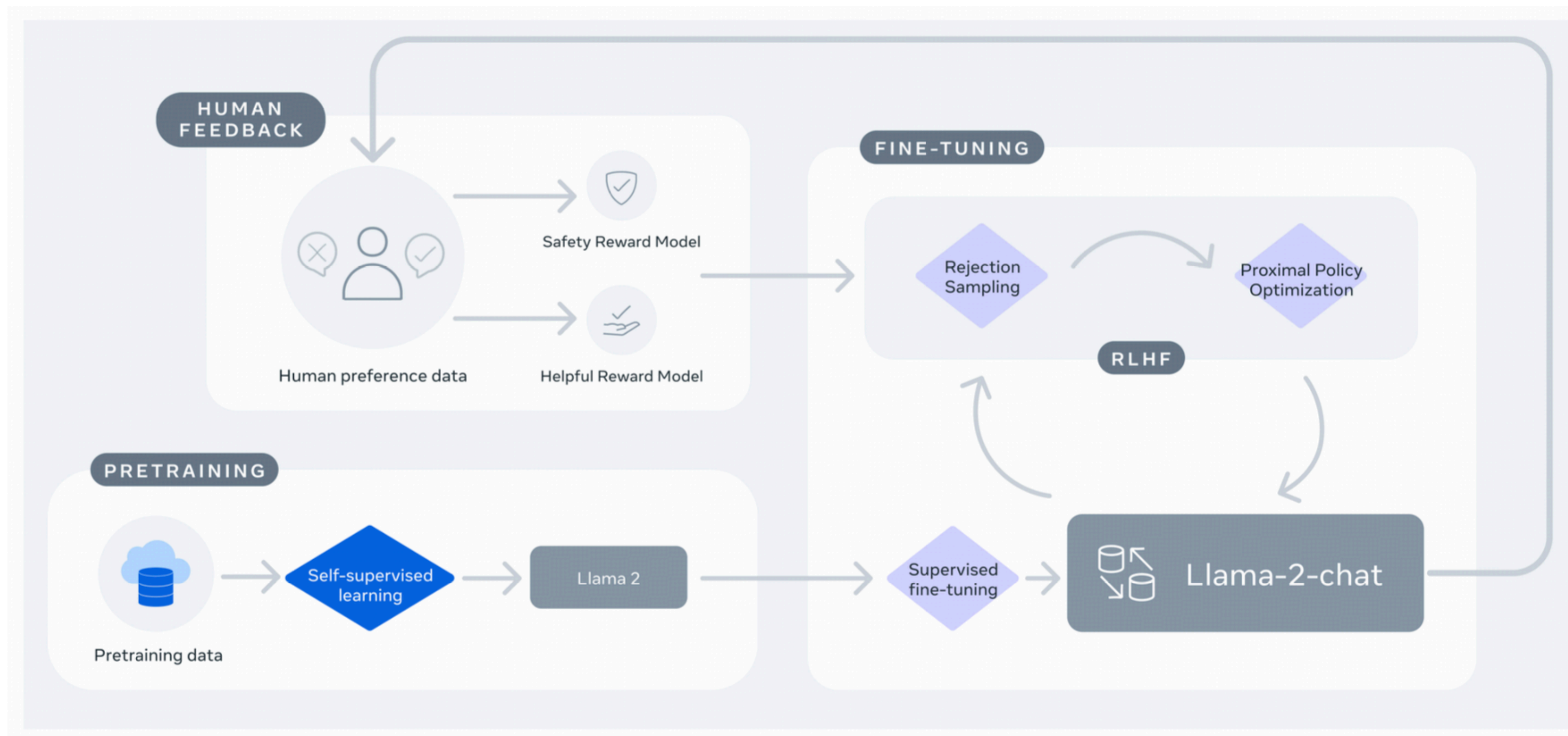
# Steps of LLM Training

- Pre-training → Supervised fine-tuning (SFT) → RL-based training



# Steps of LLM Training

- Pre-training → Supervised fine-tuning (SFT) → RL-based training
- Pre-training → Mid-training → Post-training



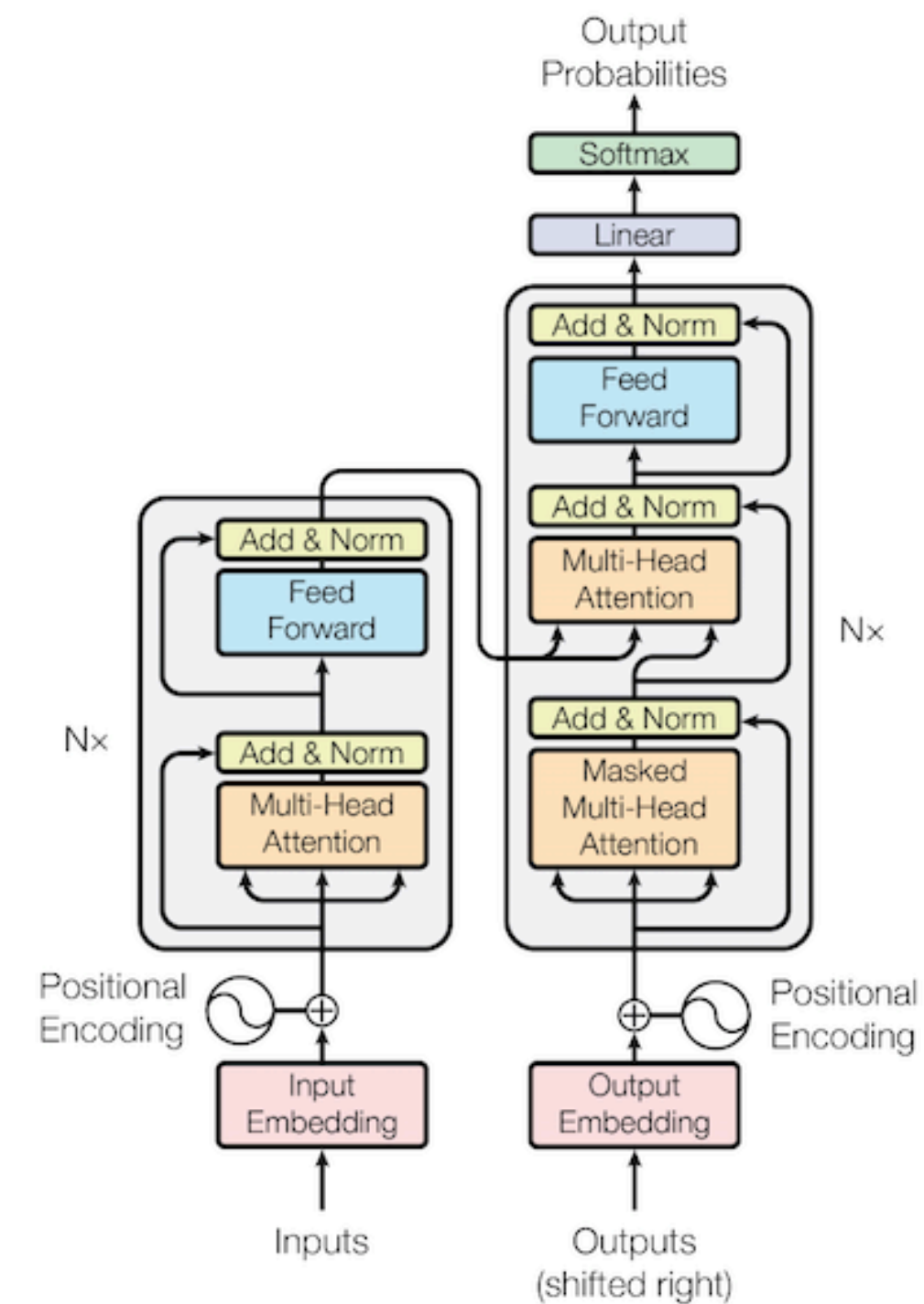
# Lecture Overview

- LLM Pre-Training
  - BERT
  - T5
  - GPT
- LLM Post-Training
  - Supervised Fine-tuning (SFT) or Instruction Tuning
  - RL-Based Training
- Fine-Tuning and Prompting

# LLM Pre-Training

# Pre-training: Training Objectives

- We have the Transformer
- During pre-training, we have a large text corpus (no labels; just text)
- Question: what labels or objectives are used to train the Transformer?



# Pre-training: Training Objectives



BERT (Encoder-only)

Devlin et al., 2018

The cabs \_\_\_ the same rates as those \_\_\_ by horse-drawn cabs and were \_\_\_ quite popular, \_\_\_ the Prince of Wales (the \_\_\_ King Edward VII) travelled in \_\_\_. The cabs quickly \_\_\_ known as "hummingbirds" for \_\_\_ noise made by their motors and their distinctive black and \_\_\_ livery. Passengers \_\_\_ the interior fittings were \_\_\_ when compared to \_\_\_ cabs but there \_\_\_ some complaints \_\_\_ the \_\_\_ lighting made them too \_\_\_ to those outside \_\_\_.

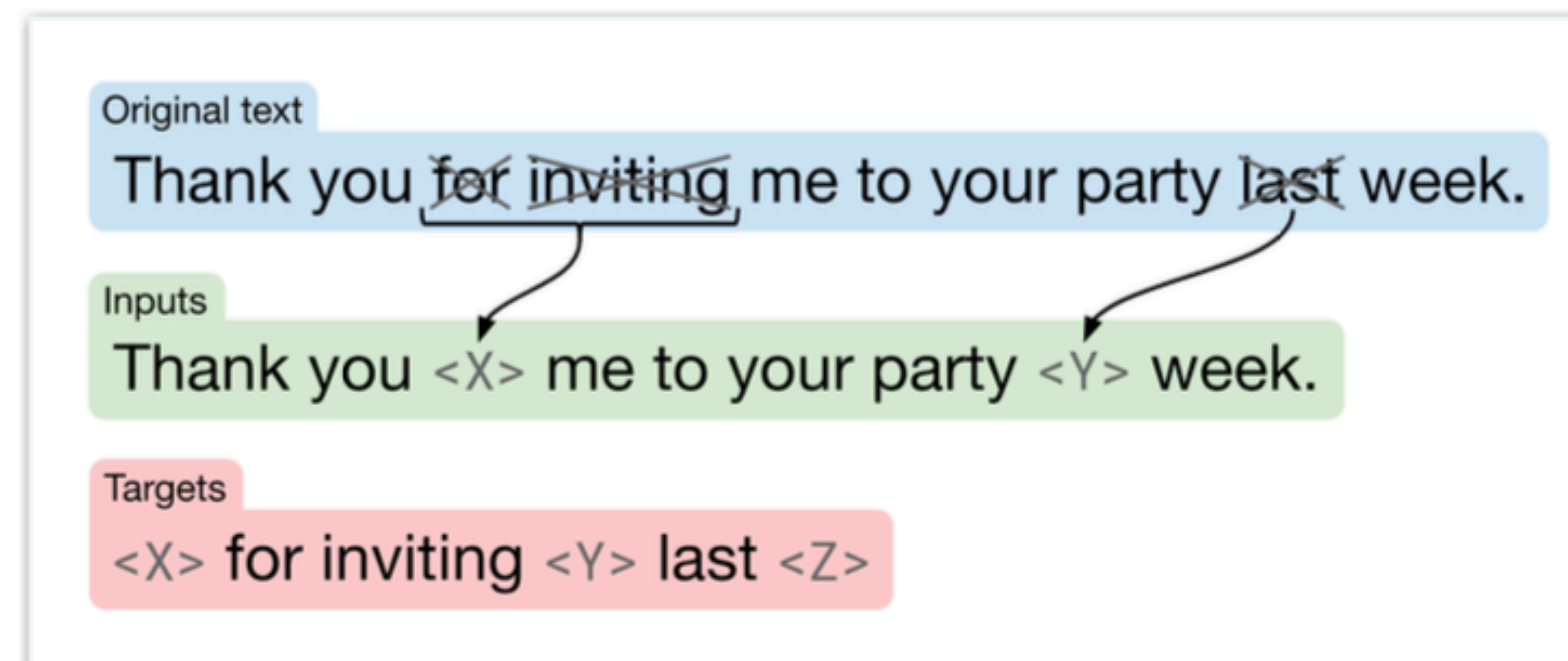
charged, used, initially, even, future, became, the, yellow, reported, that, luxurious, horse-drawn, were that, internal, conspicuous, cab

Masked token prediction



T5 (Encoder-decoder)

Raffel et al., 2019

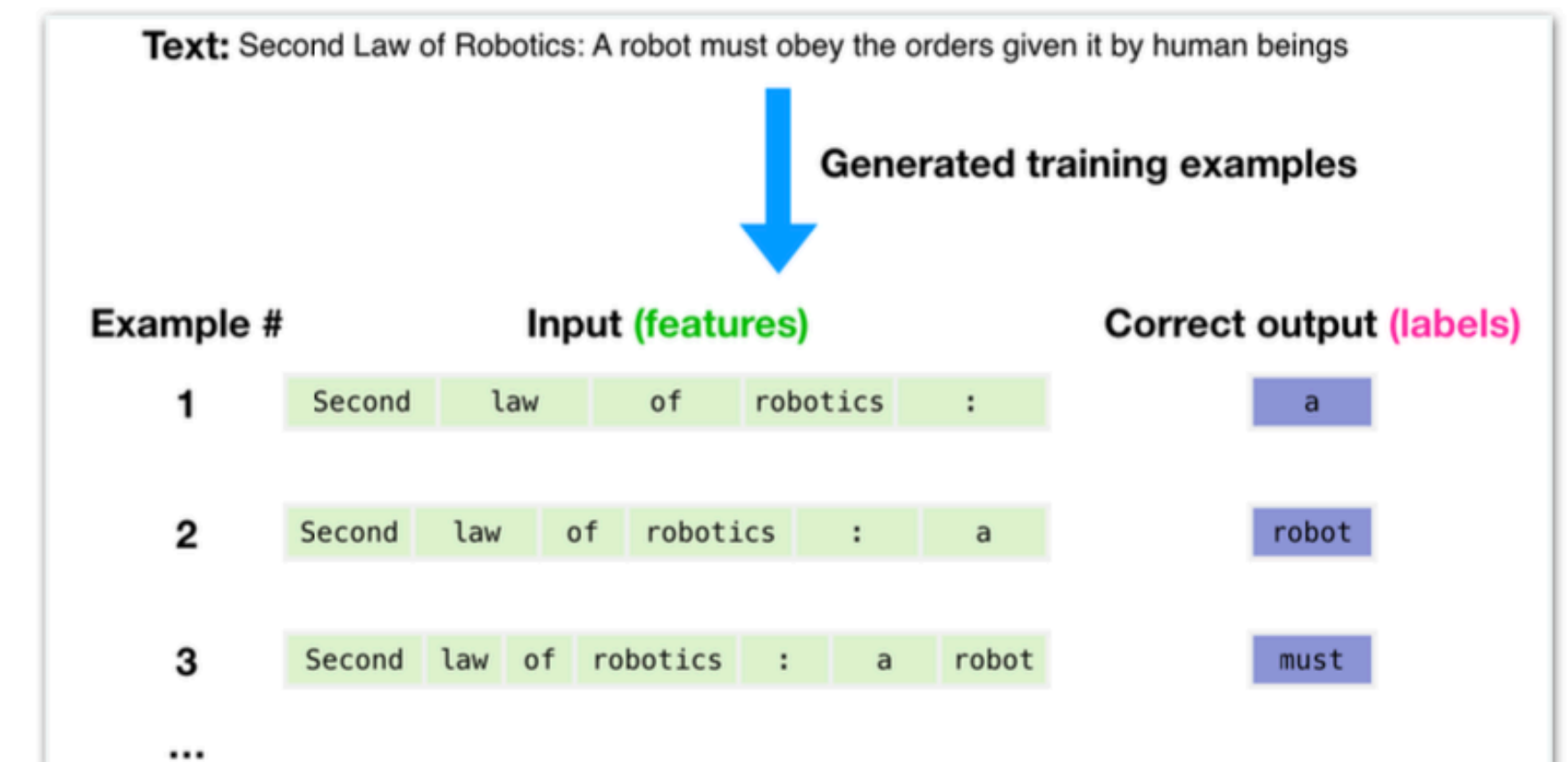


Denosing span-mask prediction



GPT - 4

Decoder-only



Next token prediction

# Pre-training: Transformer Architectures

## Encoder

- E.g., BERT, RoBERTa, DeBERTa, ...
- **Autoencoder** model
- **Masked** language modeling

## Encoder-Decoder

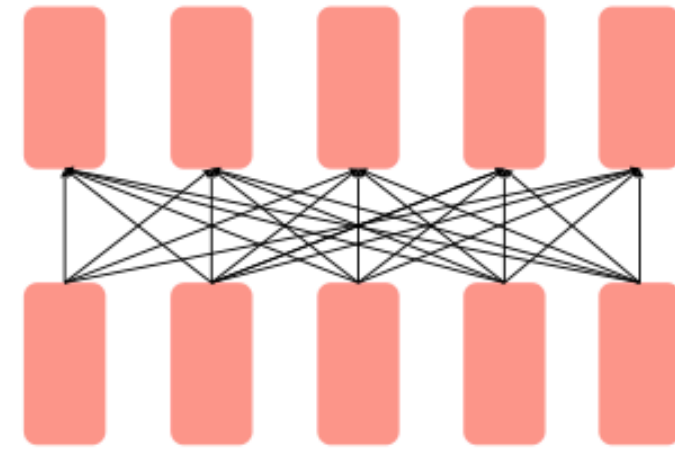
- E.g., T5, BART, ...
- **seq2seq** model

## Decoder

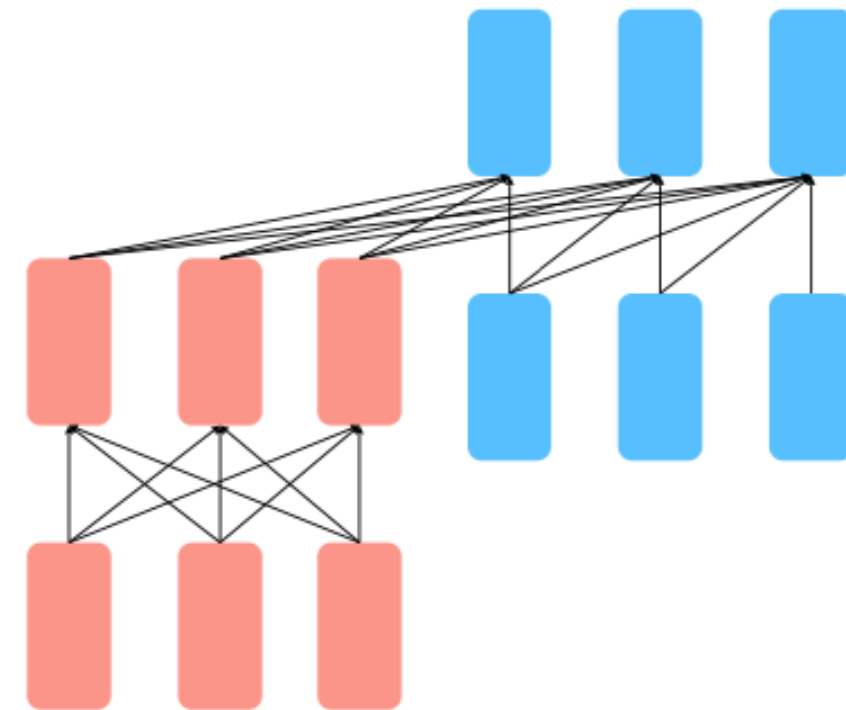
- E.g., GPT, GPT2, GPT3, ...
- **Autoregressive** model
- **Left-to-right** language modeling

# Pre-training: Transformer Architectures

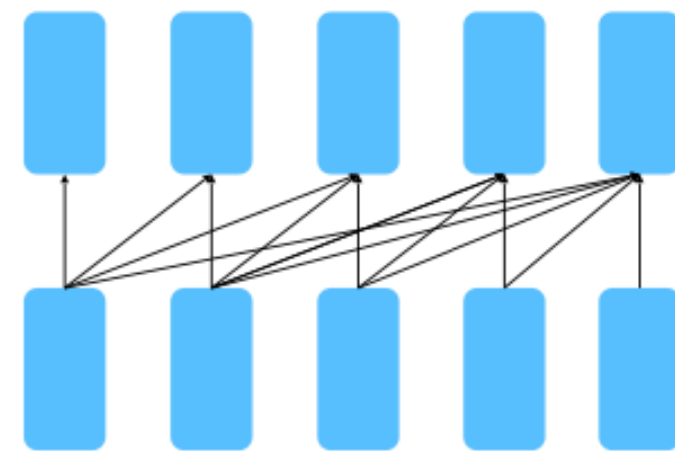
**Encoder**



**Encoder-Decoder**



**Decoder**

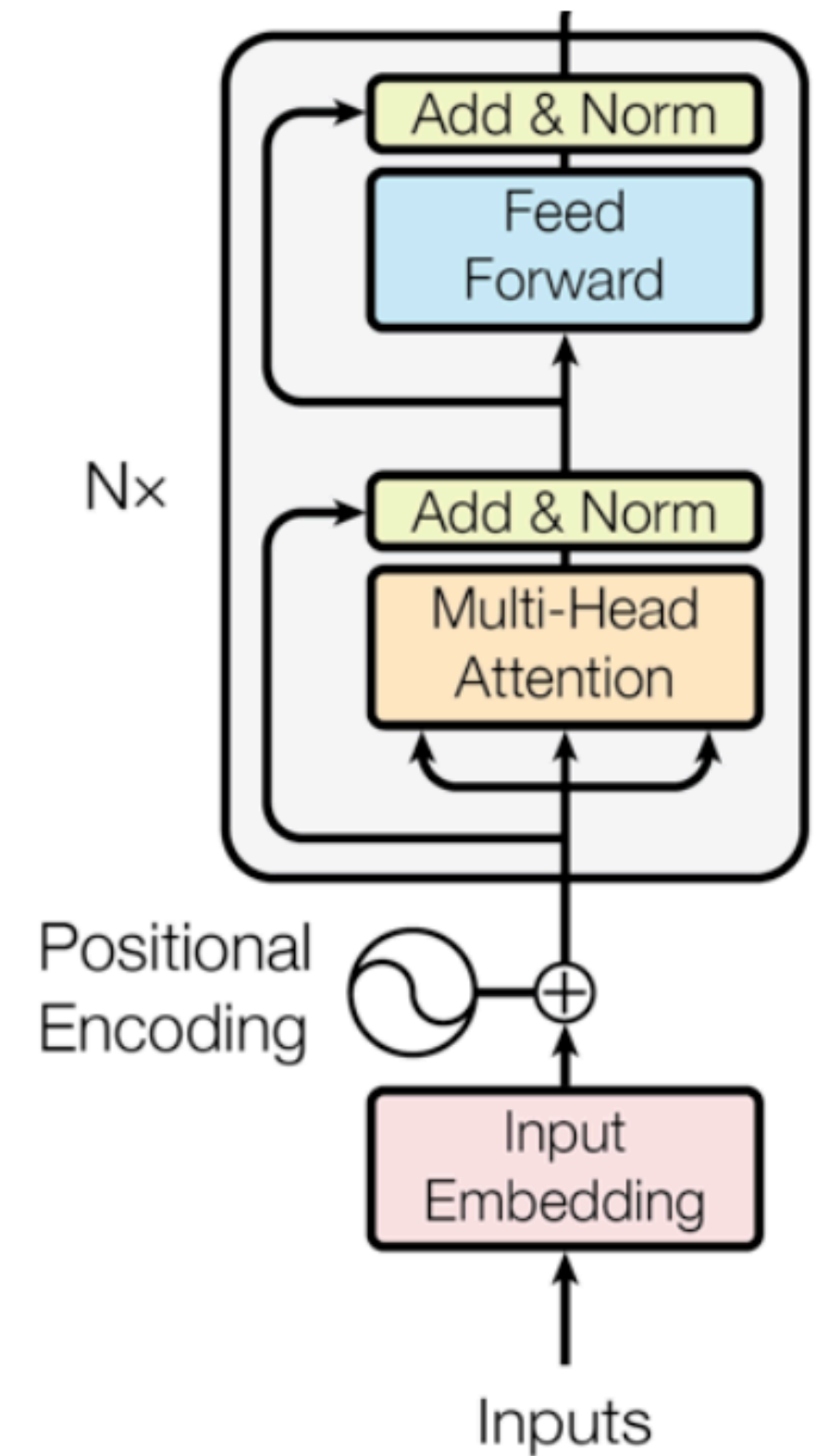


- Bidirectional; can condition on the future context
- Map two sequences of different length together
- Language modeling; can only condition on the past context

# Encoder-Only Models: BERT

# BERT [Devlin+ 2019]

- Bidirectional Encoder Representations from Transformers
- A fine-tuning approach based on a **bidirectional Transformer encoder** instead of a Transformer decoder
- The key: learn representations based on **bidirectional contexts**
  - Example #1: we went to the river bank
  - Example #2: I need to go to bank to make a deposit
- Two new pre-training objectives:
  - **Masked language modeling (MLM)**
  - Next sentence prediction (NSP) - Later work shows that NSP hurts performance though...



# Masked Language Modeling [Devlin+ 2019]

- Why we can't do language modeling with bidirectional models?



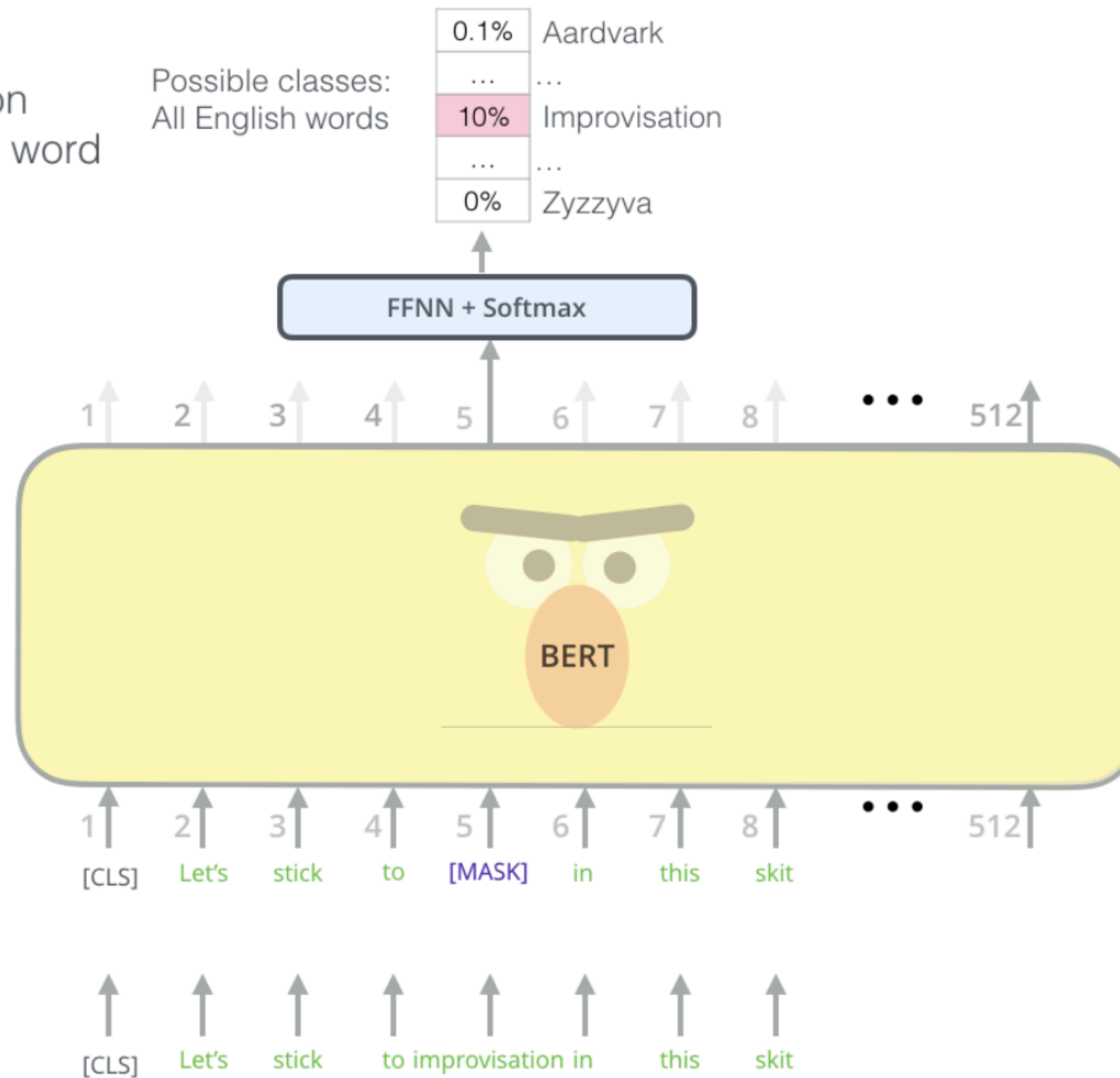
- Solution: Mask out  $k\%$  of the input, and predict the masked words

store                      gallon  
↑                              ↑  
the man went to [MASK] to buy a [MASK] of milk

**$k = 15\%$  in practice**

# BERT [Devlin+ 2019]

Use the output of the masked word's position to predict the masked word



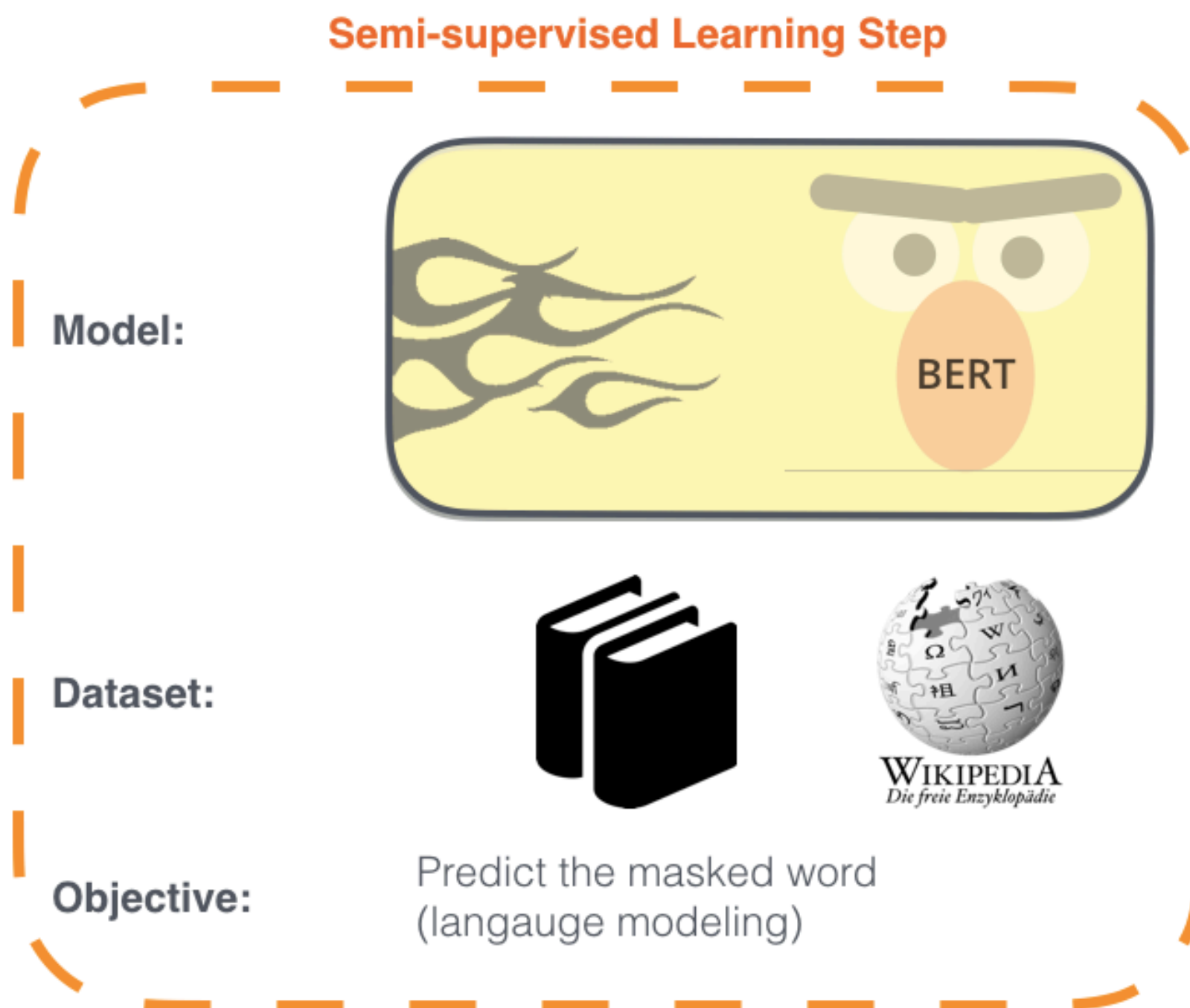
Randomly mask 15% of tokens

Input

# BERT Pre-Training → Fine-Tuning [Devlin+ 2019]

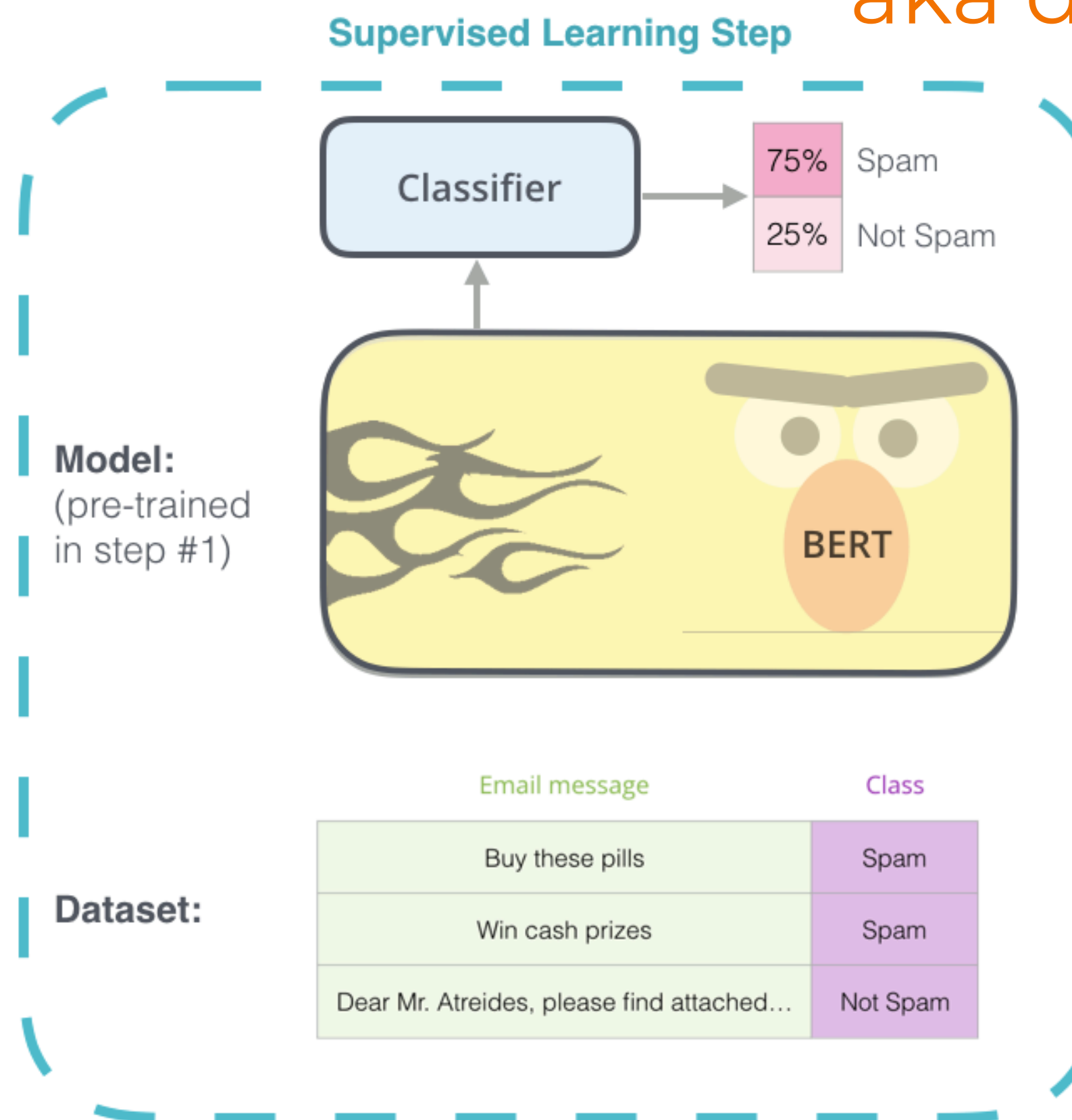
1 - **Semi-supervised** training on large amounts of text (books, wikipedia..etc).

The model is trained on a certain task that enables it to grasp patterns in language. By the end of the training process, BERT has language-processing abilities capable of empowering many models we later need to build and train in a supervised way.



2 - **Supervised** training on a specific task with a labeled dataset.

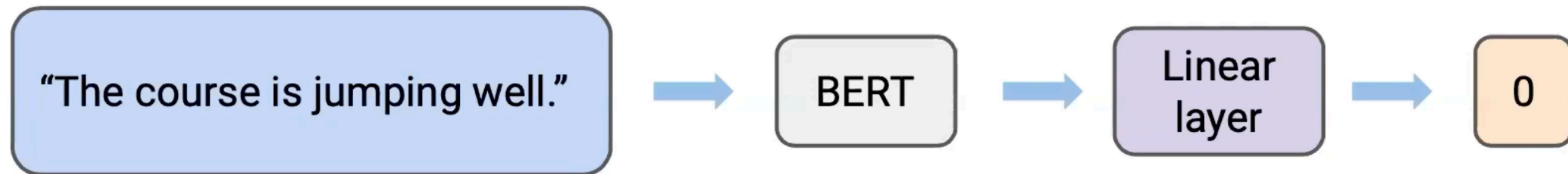
aka downstream task



# BERT Pre-Training → Fine-Tuning [Devlin+ 2019]

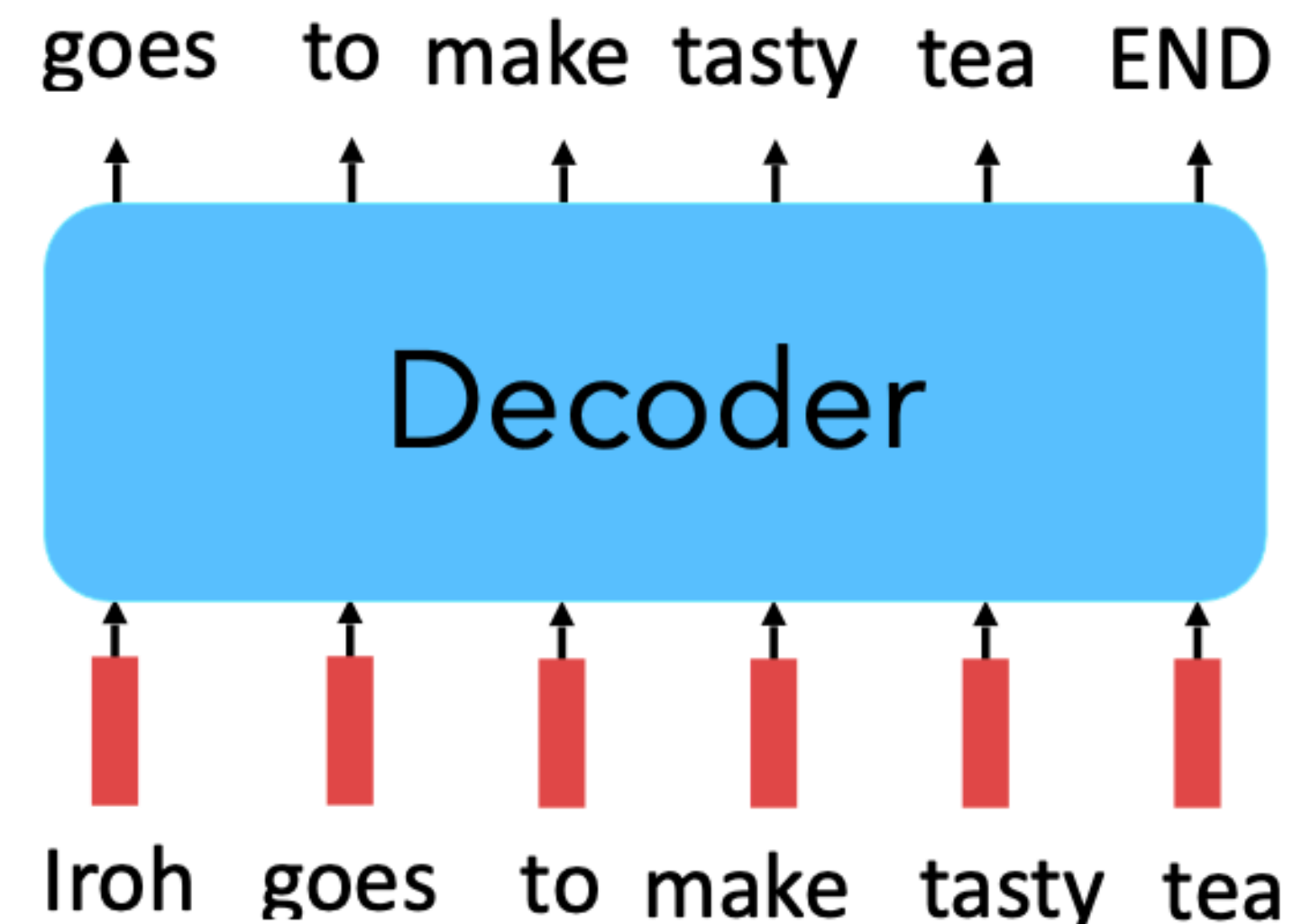
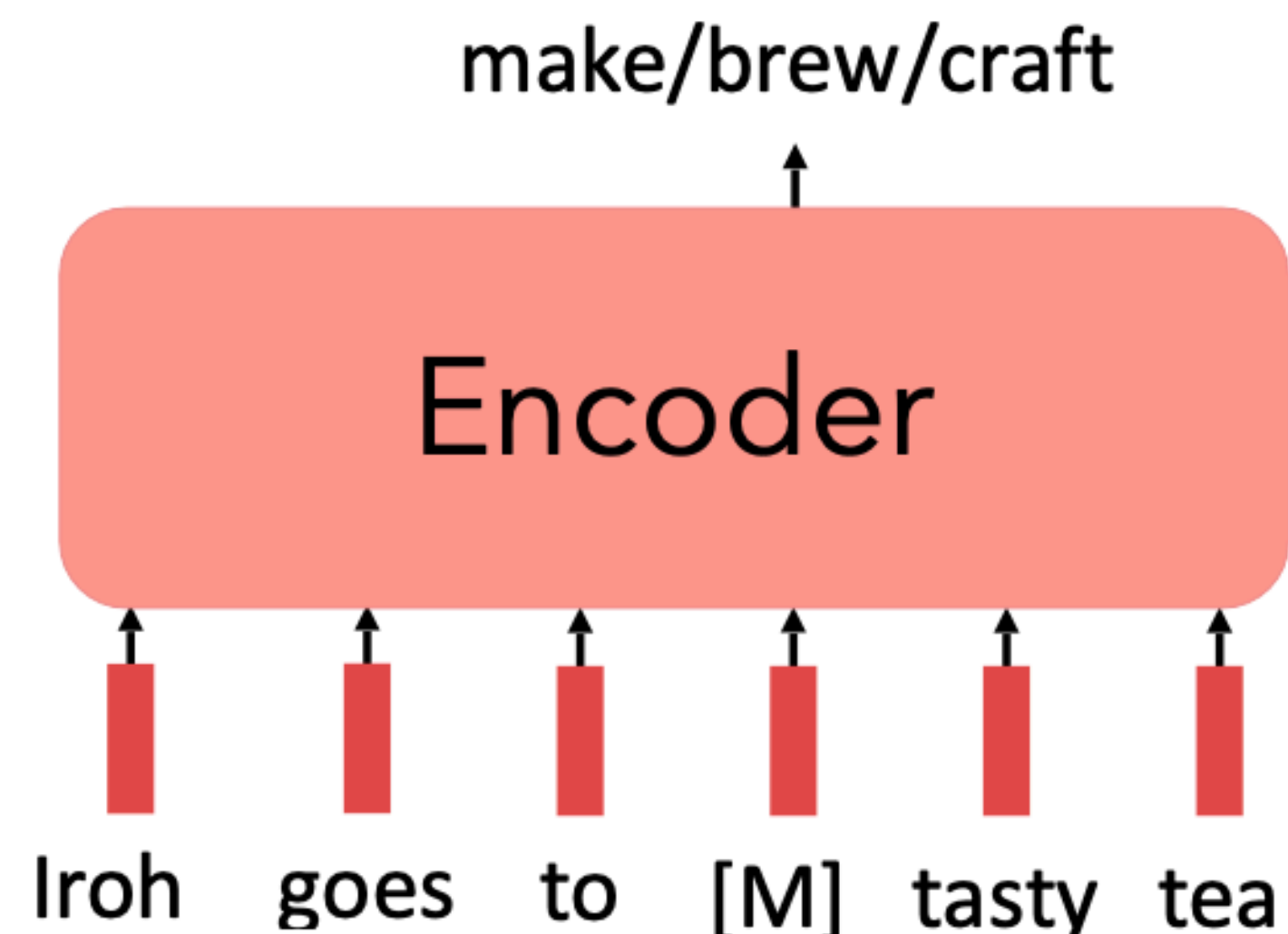
**Input: text**

**Output: label**



# Encoder-Only Models (BERT): Pros & Cons

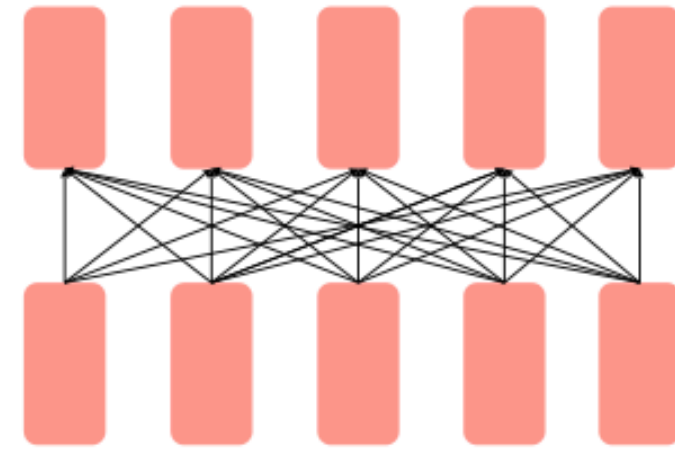
- 👍 Consider both left and right context (bidirectional)
- 👍 Capture intricate contextual relationships
- 👎 Not good at generating text from left-to-right, one token at a time



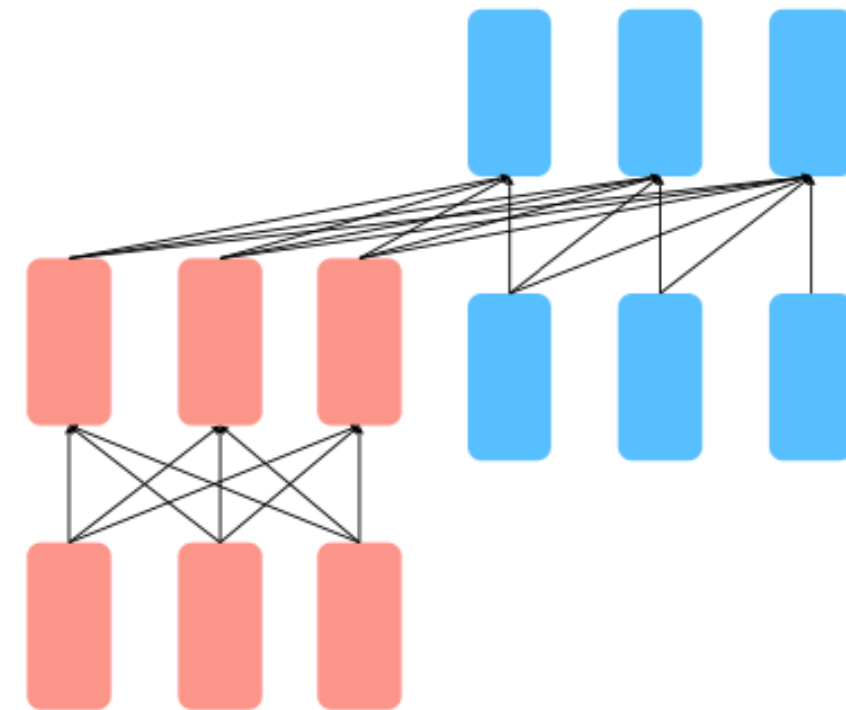
# Encoder-Decoder Models: T5

# Pre-training: Transformer Architectures

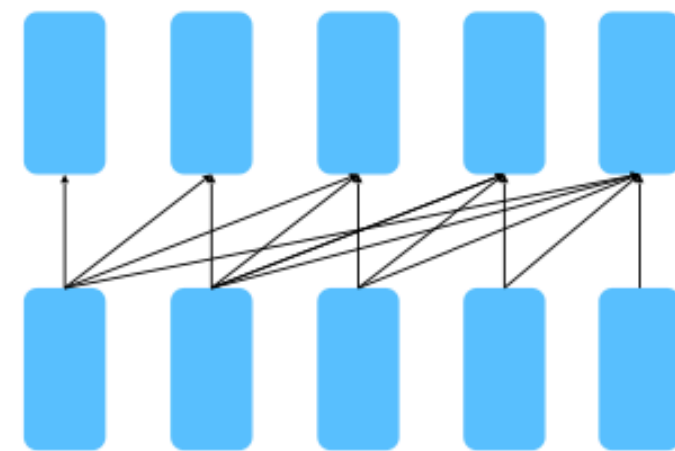
**Encoder**



**Encoder-Decoder**



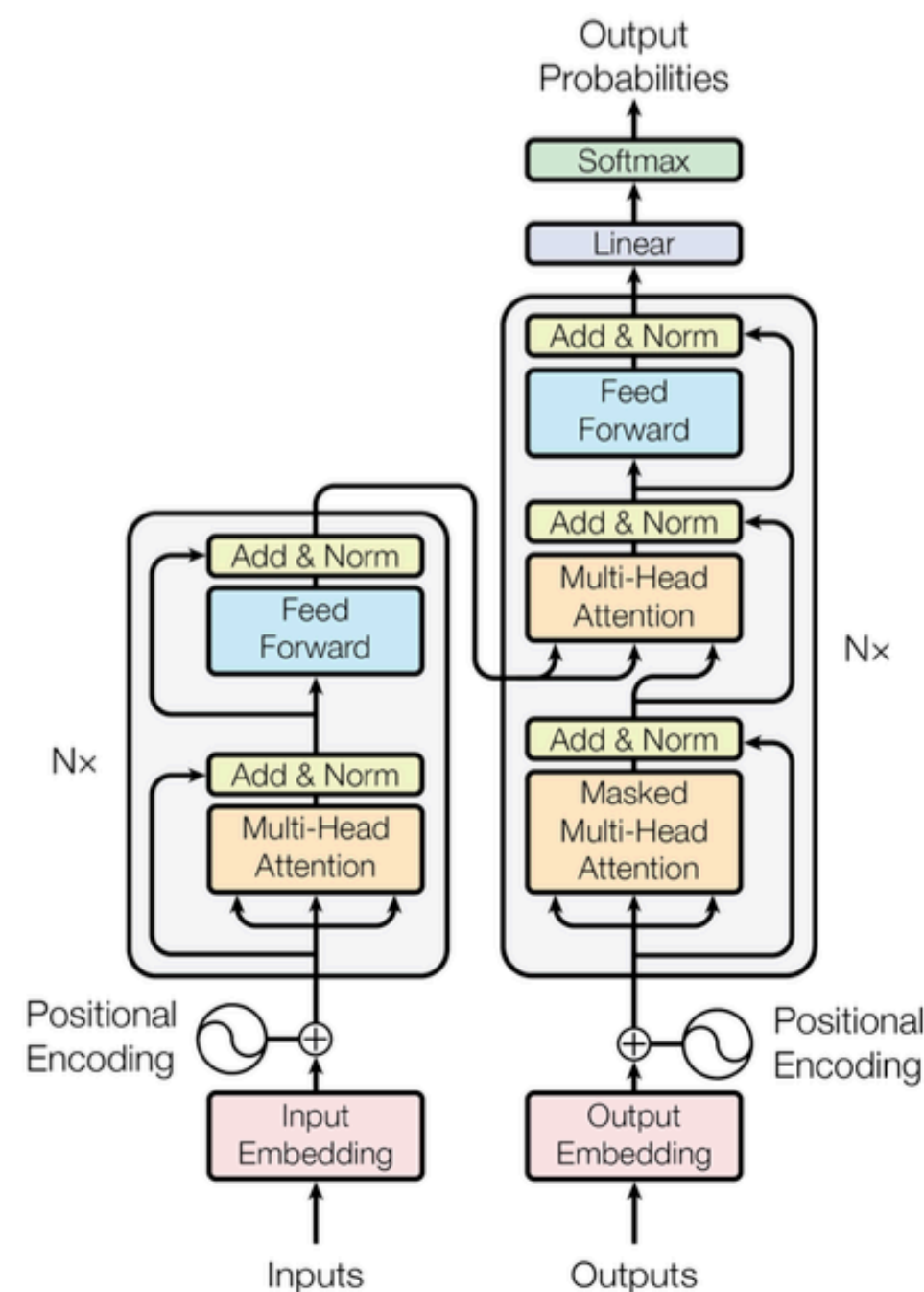
**Decoder**



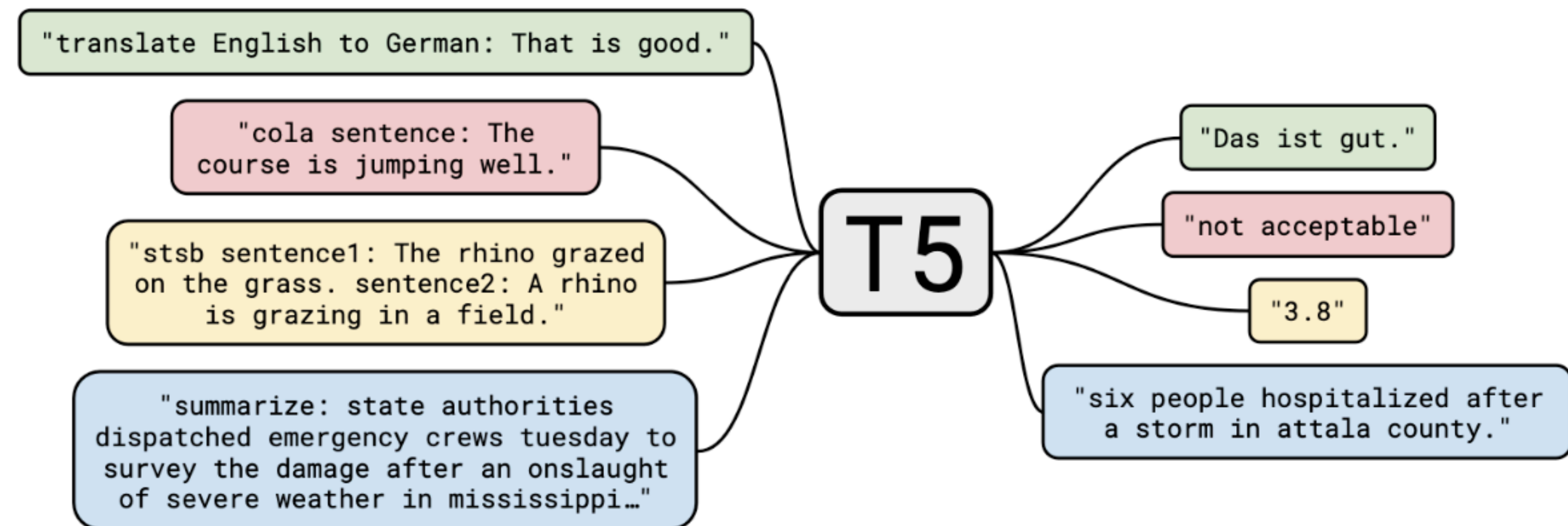
- Bidirectional; can condition on the future context
- Map two sequences of different length together
- Language modeling; can only condition on the past context

# Text-to-Text Transfer Transformer [Raffel+ 2020]

- Encoder-only models enjoy the benefits of bidirectionality, but they can't be used to generate text
- Decoder-only models can do generation but they are left-to-right LMs
- **Text-to-text models combine the best of both worlds!**



T5 = Text-to-Text Transfer Transformer



# T5 Training Objective [Raffel+ 2020]

- Text span corruption (denoising)
- Replace different-length spans from the input with unique placeholders (e.g., `<extra_id_0>`); decode out the masked spans

Original text

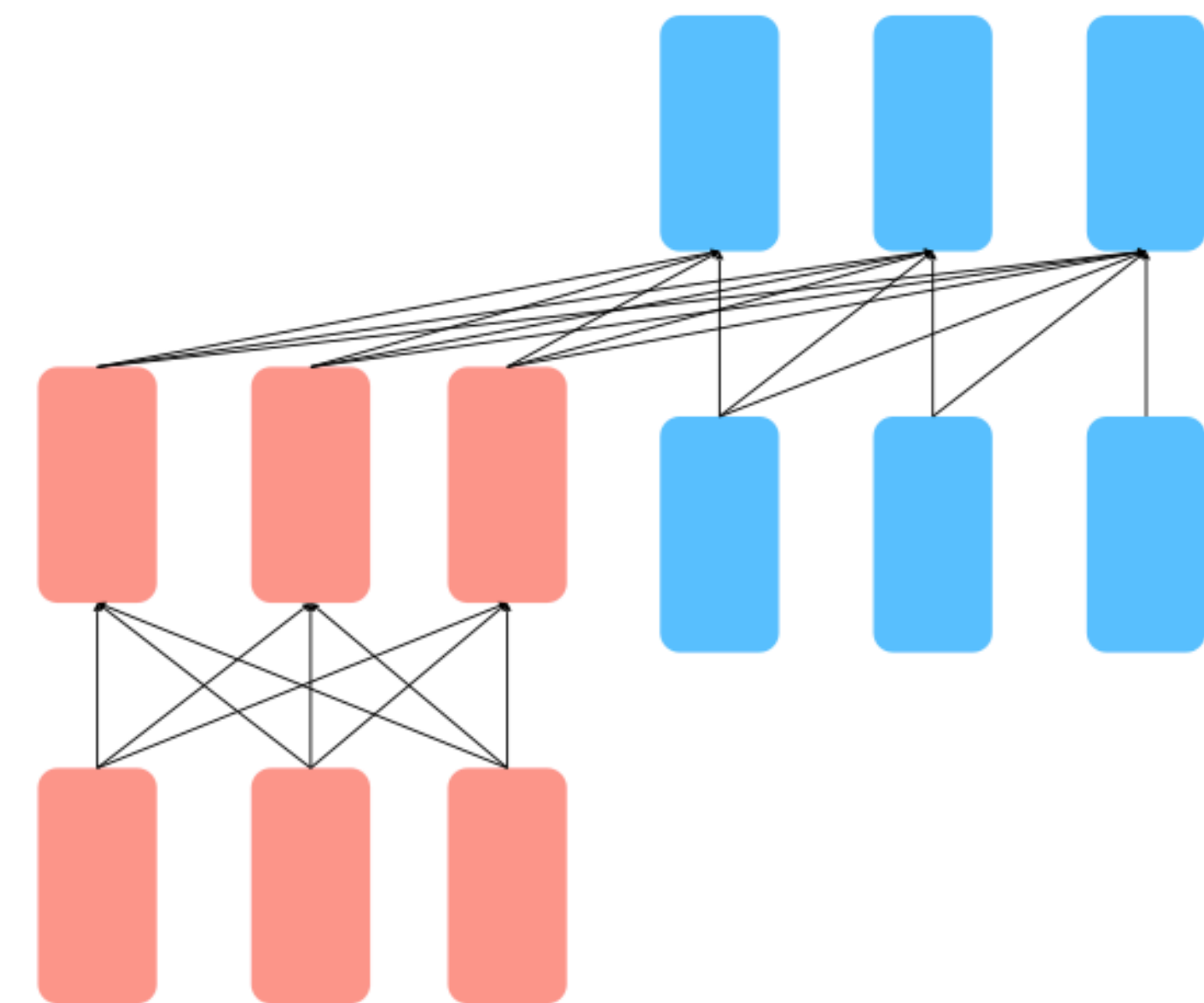
Thank you ~~for inviting~~ me to your party ~~last~~ week.

Inputs

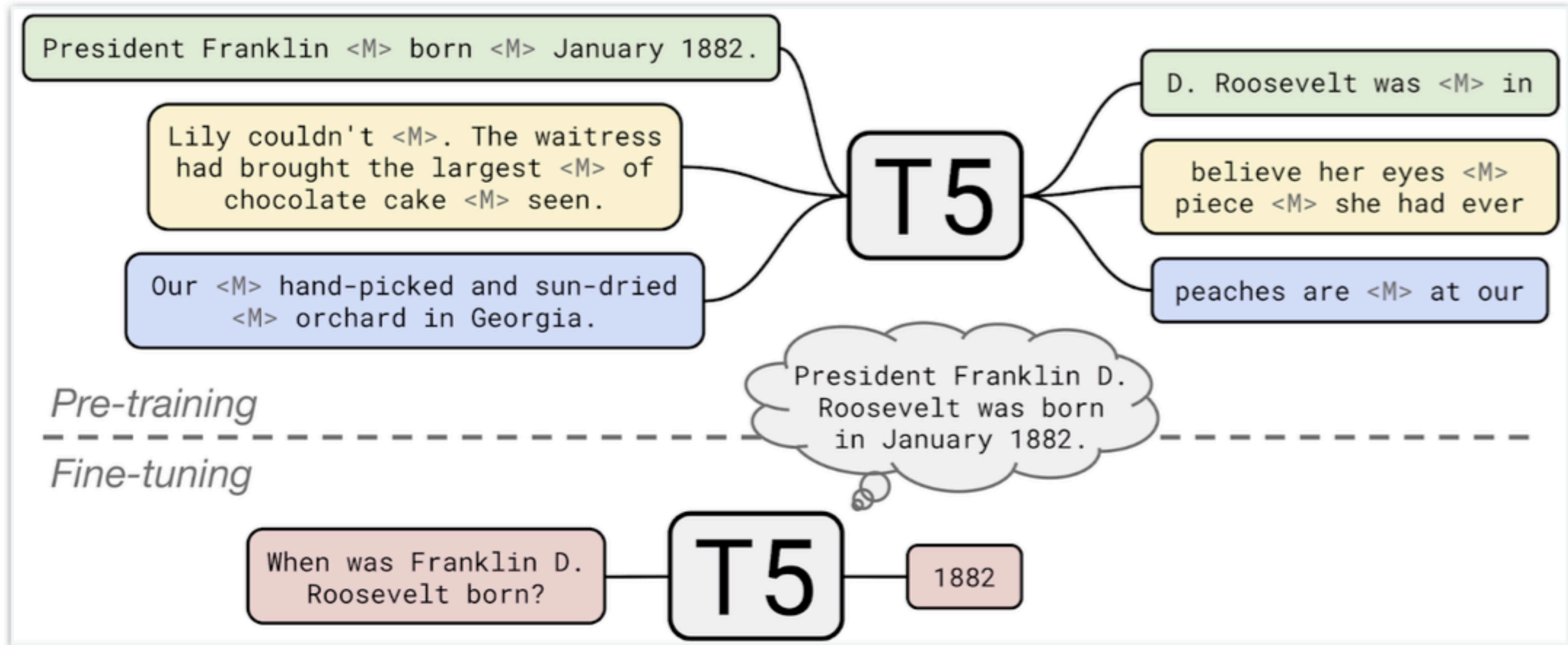
Thank you `<X>` me to your party `<Y>` week.

Targets

`<X>` for inviting `<Y>` last `<Z>`



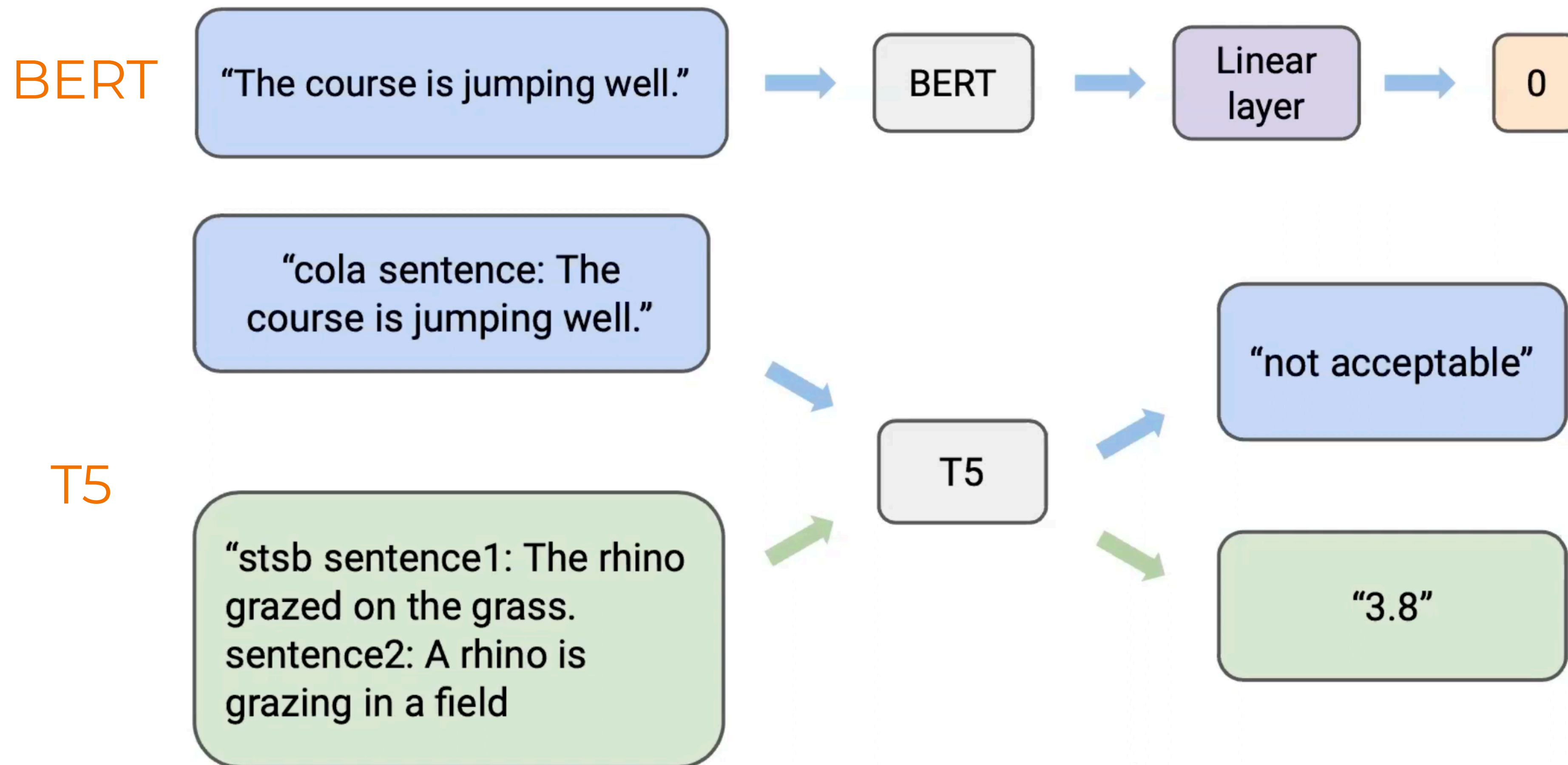
# T5 Pre-Training → Fine-Tuning [Raffel+ 2020]



# Encoder-Decoder Models (T5): Pros & Cons

- 👍 A nice middle ground between leveraging bidirectional contexts and open-text generation
- 👍 Good for multi-task fine-tuning
  - They treat all the NLP tasks as text-to-text transfer task
- 👎 Require more text preprocessing
- 👎 Harder to train
- 👎 Less flexible for natural language generation

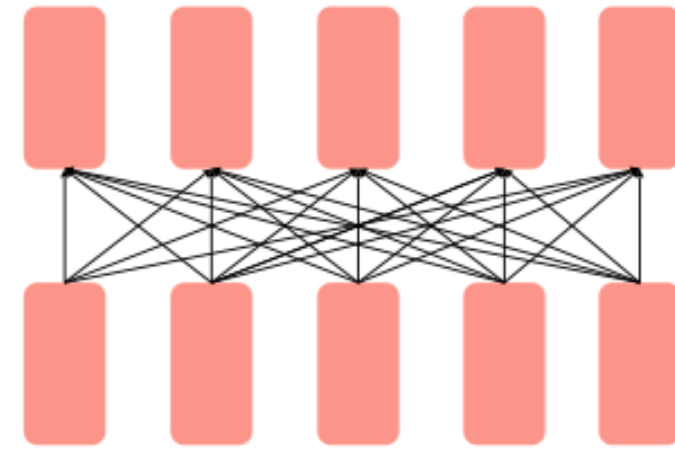
# BERT → T5: A Multi-Task Learning View



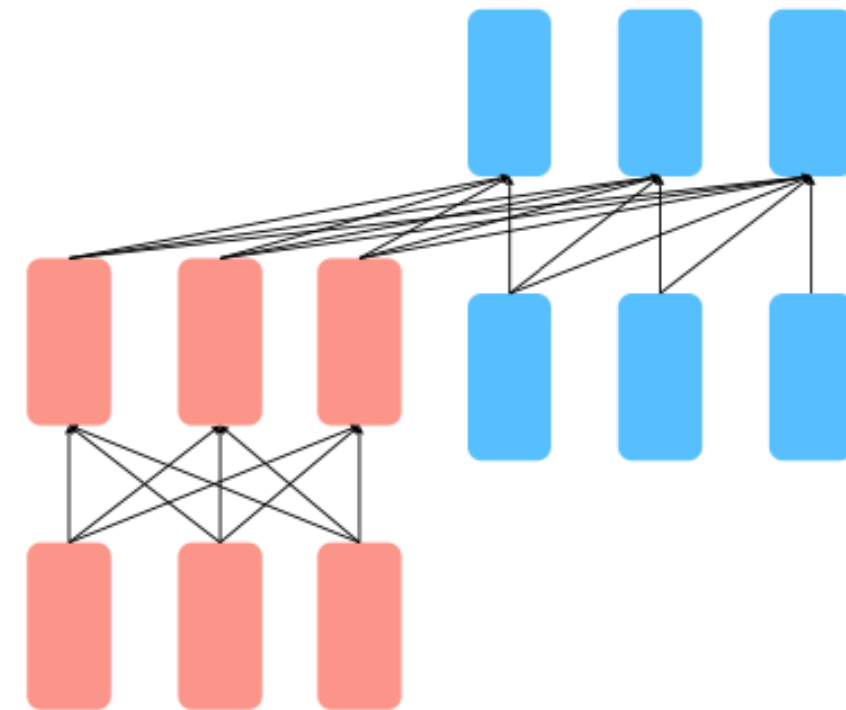
# Decoder-Only Models: GPT

# Pre-training: Transformer Architectures

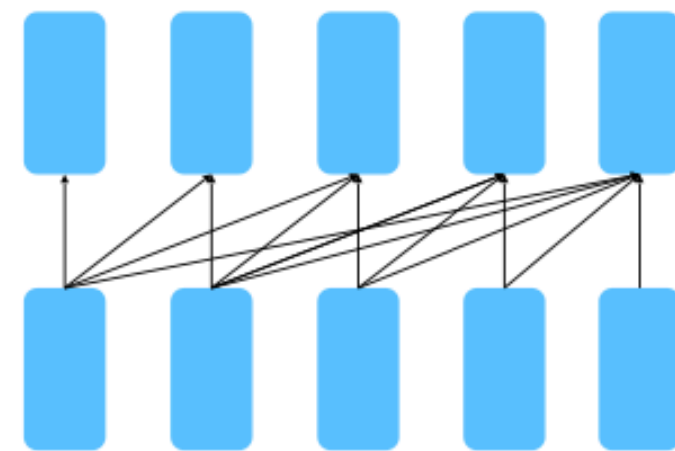
**Encoder**



**Encoder-Decoder**



**Decoder**



- Bidirectional; can condition on the future context
- Map two sequences of different length together
- Language modeling; can only condition on the past context

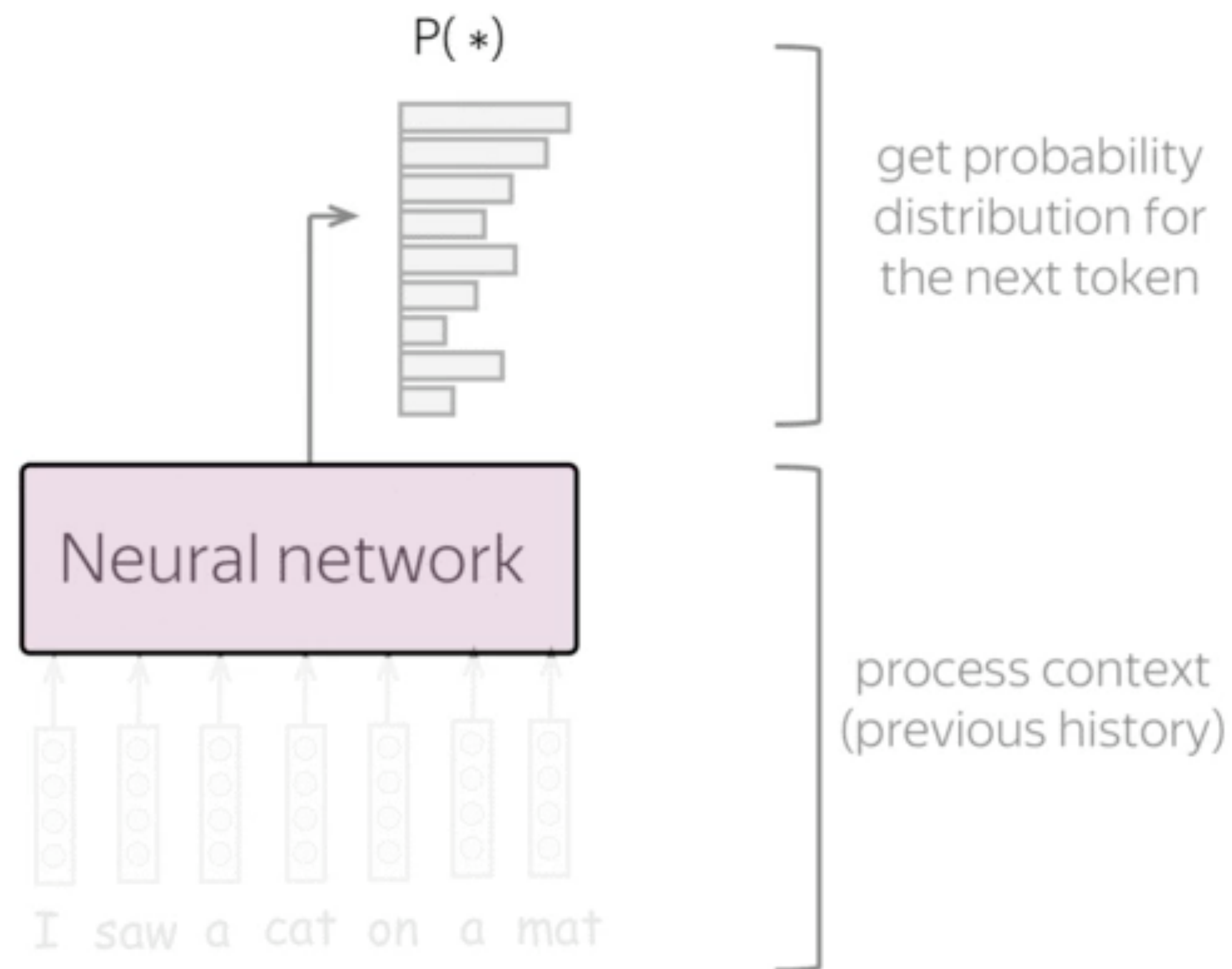
# Auto-Regressive Language Models

$$P(X) = \prod_{i=1}^I P(x_i \mid x_1, \dots, x_{i-1})$$

Next Token      Context

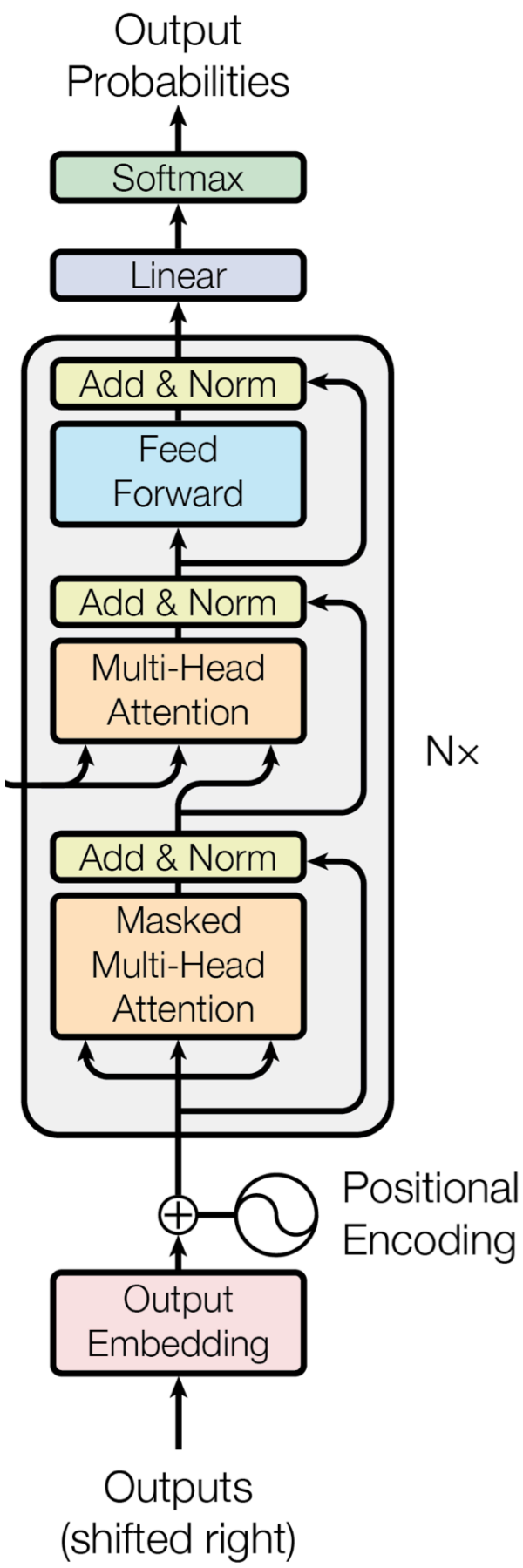
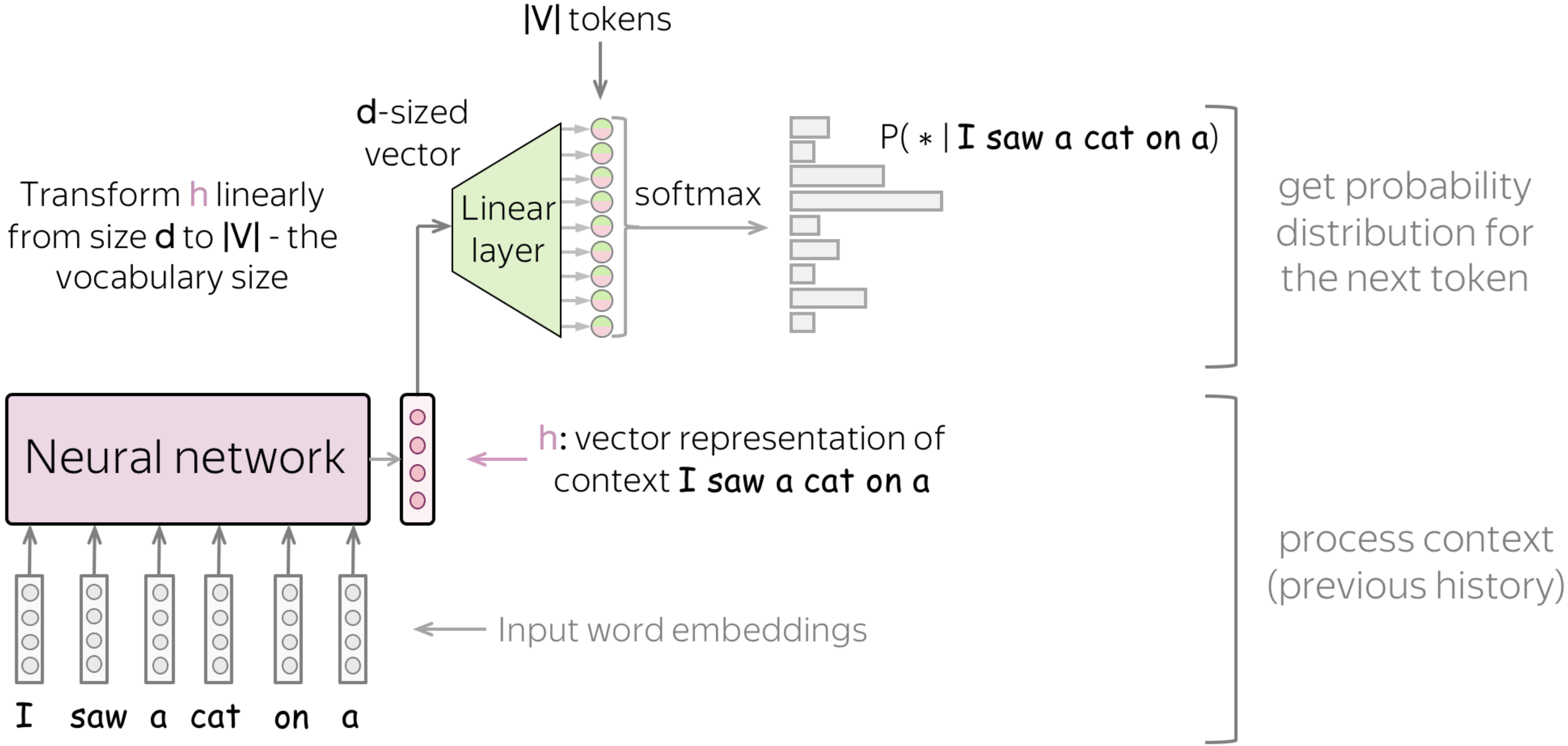
- Predict next token (word) from all previous tokens
  - e.g. given a context "Inha U", what would be the next token?

# Next Token Prediction



- This is **classification** over the entire vocabulary
  - We can think of neural language models as neural classifiers
  - They classify a prefix of a text into  $|V|$  classes, where the classes are vocabulary tokens

# Next Token Prediction



# Pre-Training Dataset

- Example: LLaMA 1 pre-training dataset mixture

Dataset	Sampling prop.	Epochs	Disk size
CommonCrawl	67.0%	1.10	3.3 TB
C4	15.0%	1.06	783 GB
Github	4.5%	0.64	328 GB
Wikipedia	4.5%	2.45	83 GB
Books	4.5%	2.23	85 GB
ArXiv	2.5%	1.06	92 GB
StackExchange	2.0%	1.03	78 GB

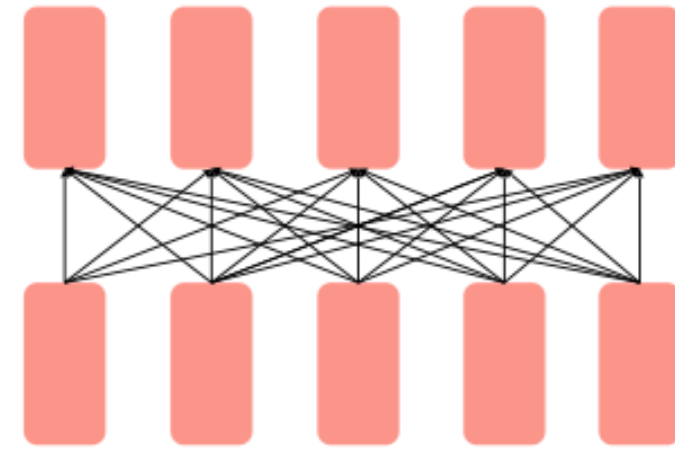
1.4 Trillion tokens

# Note: How Large are 1T Tokens?

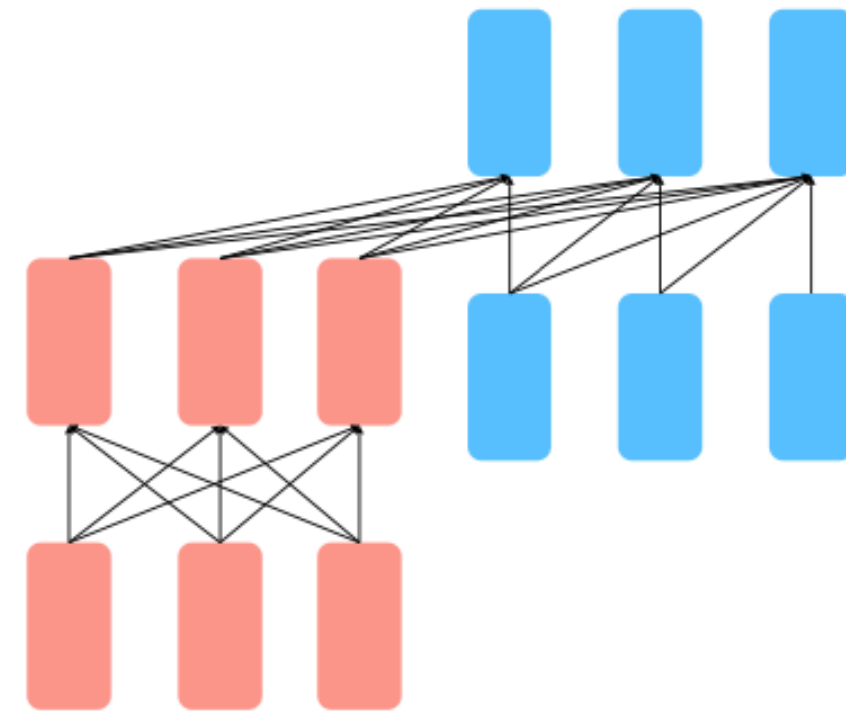
- Physical size (if printed)
  - A typical page contains about 300-500 words
  - Assume 1T tokens are 750B words, and 400 words per page
  - Total (book) pages is around 1.875B
- Reading time
  - The average reading speed is about 200-250 words per minute
  - To read 750B words, it would take about 3.75 billion minutes, or around 7,125 years of reading

# Summary

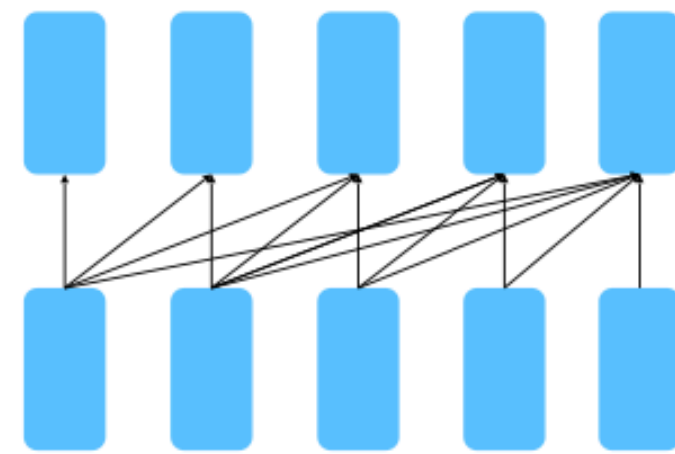
**Encoder**



**Encoder-Decoder**



**Decoder**



- Bidirectional; can condition on the future context
- Map two sequences of different length together
- Language modeling; can only condition on the past context

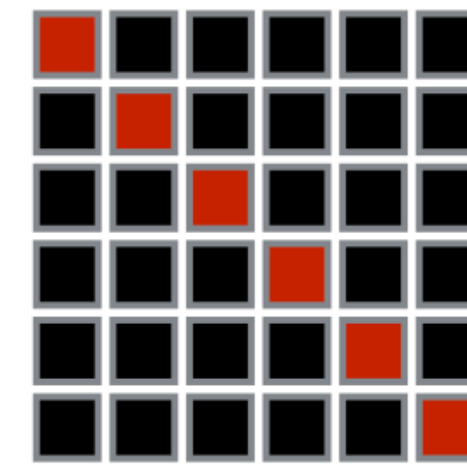
# Summary

BERT

*Bidirectional Prediction*

$$P(X) \neq \prod_{i=1}^{|X|} P(x_i | x_{\neq i})$$

(e.g. masked language model)

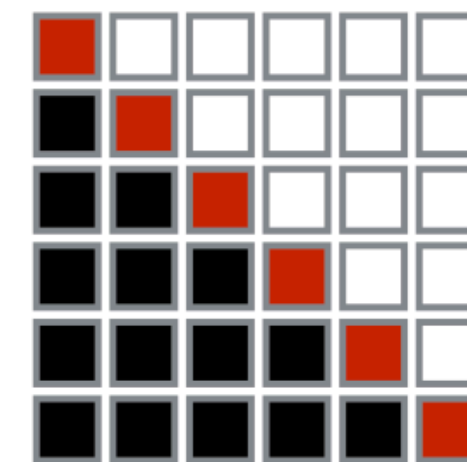


GPT

*Left-to-right Autoregressive Prediction*

$$P(X) = \prod_{i=1}^{|X|} P(x_i | x_1, \dots, x_{i-1})$$

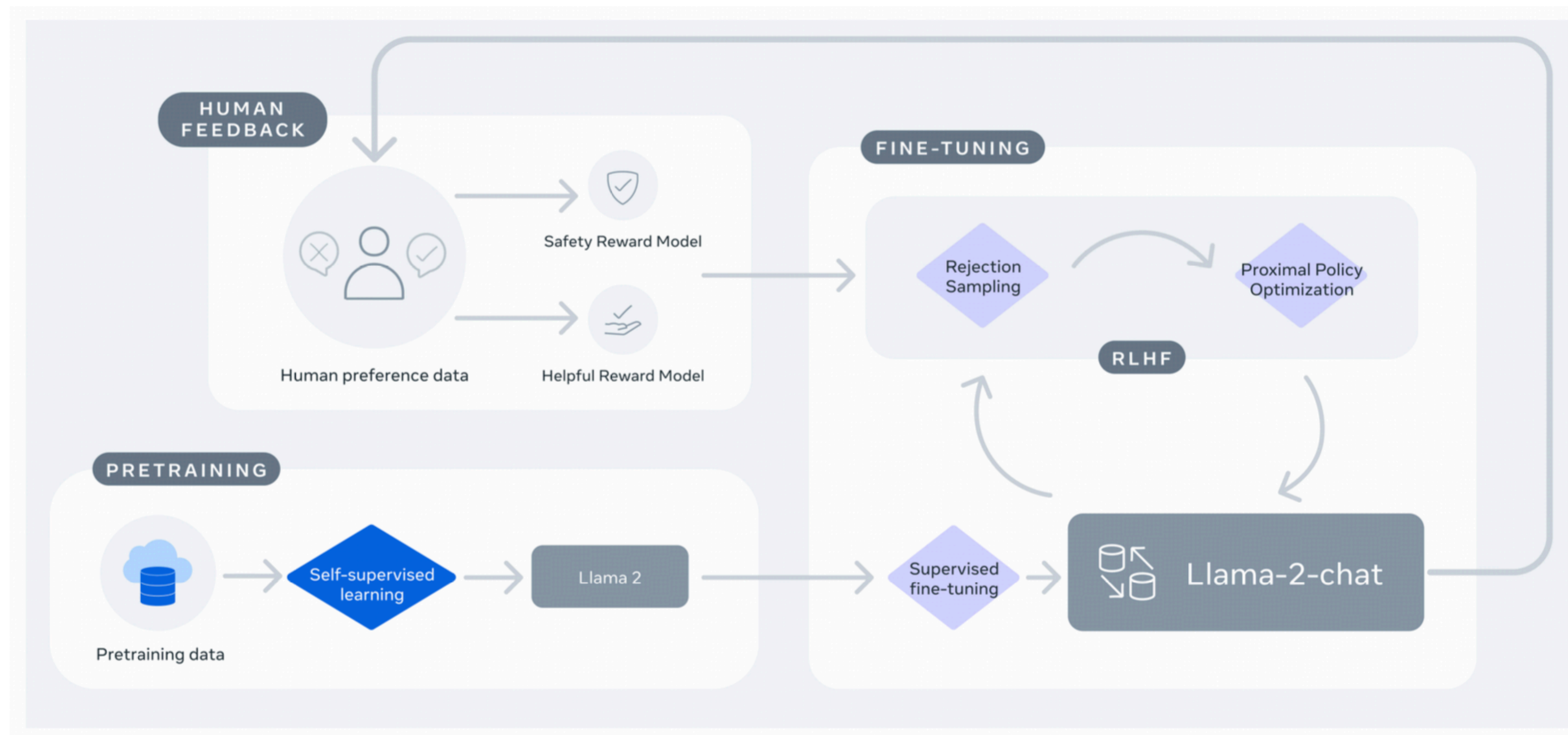
(e.g. RNN or Transformer LM)



# LLM Post-Training

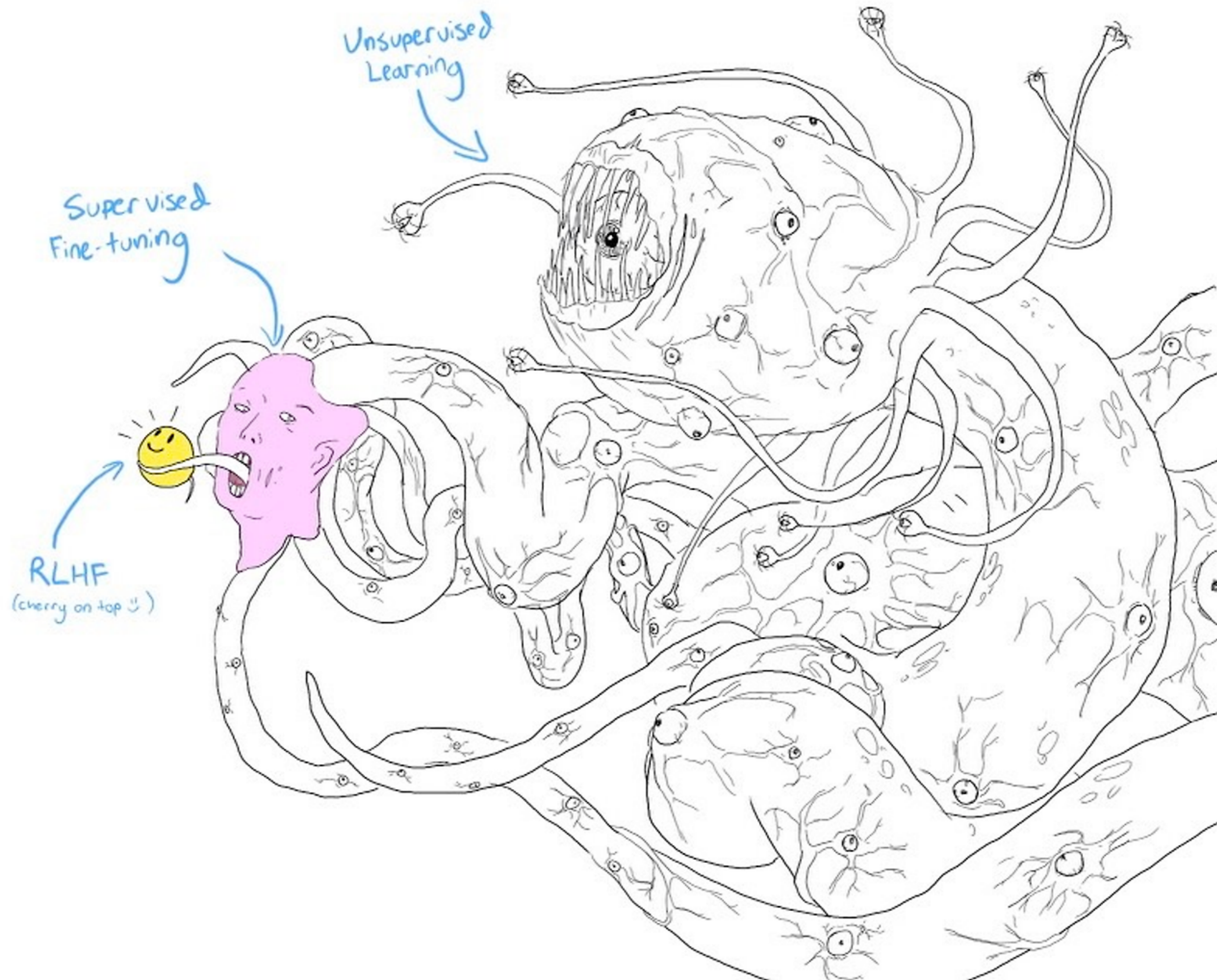
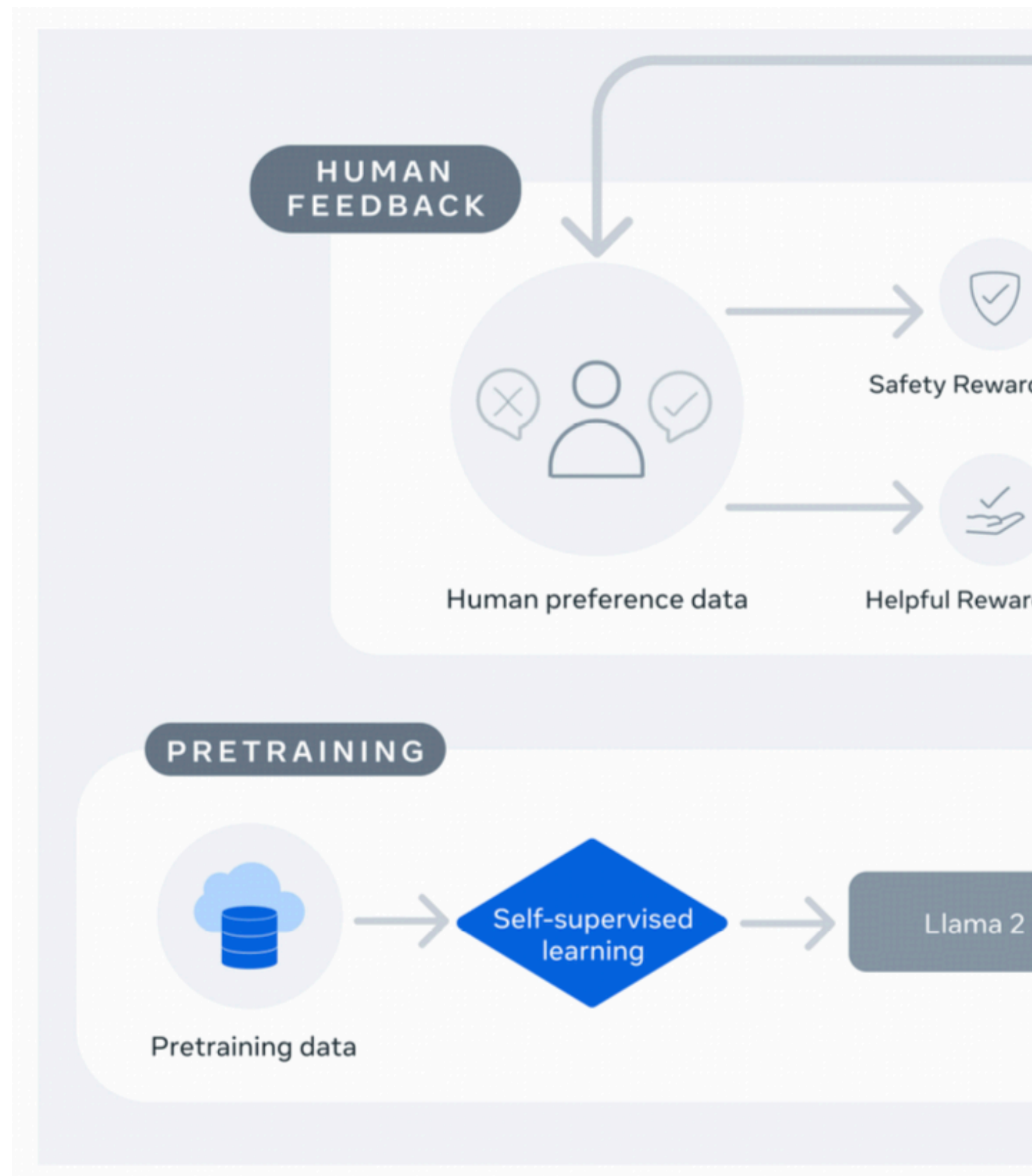
# Recall: Steps of LLM Training

- Pre-training → Supervised fine-tuning (SFT) → RL-based training
- Pre-training → Mid-training → Post-training



# Recall: Steps of LLM

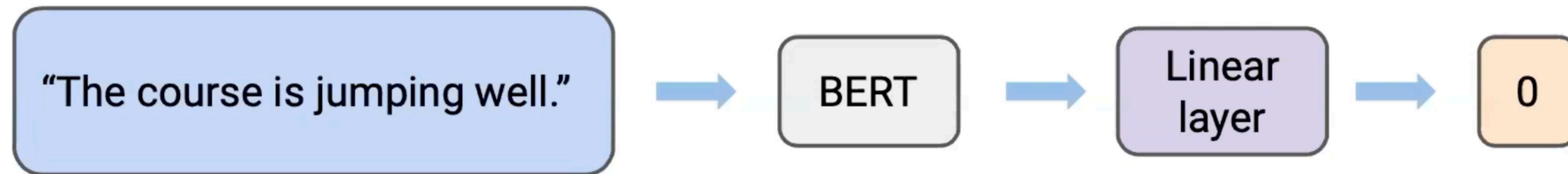
- Pre-training → Supervised fine-tuning
- Pre-training → Mid-training →



# Recall: BERT

**Input:** text

**Output:** label

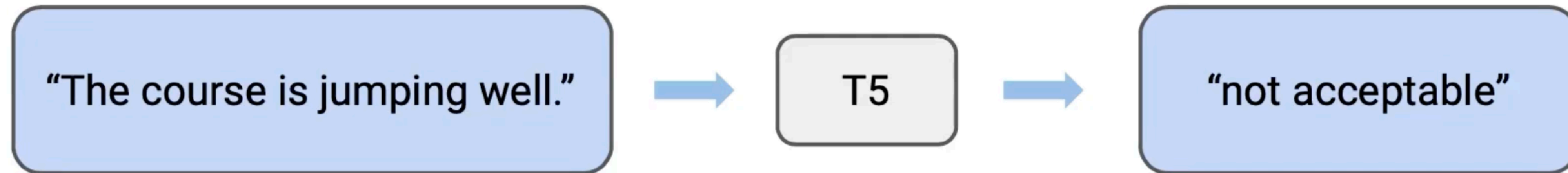


Task specific linear layer is necessary

# Recall: T5

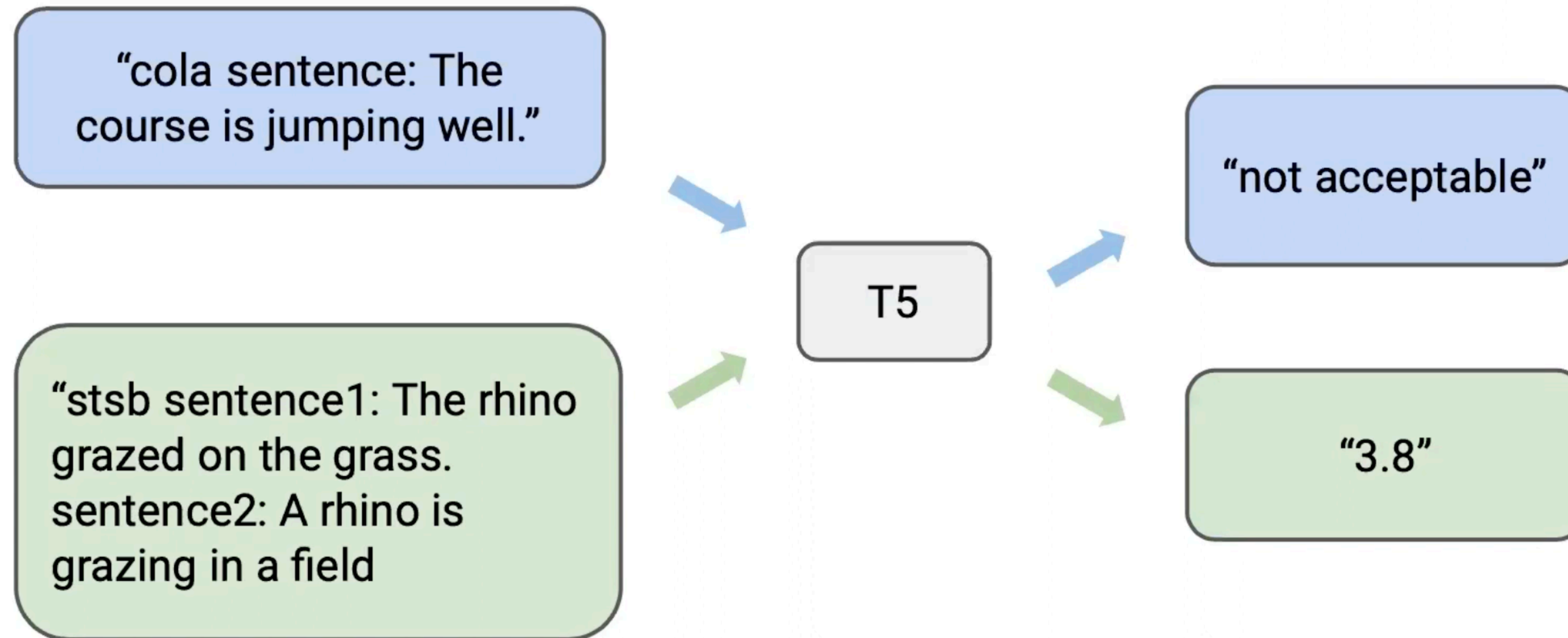
**Input: text**

**Output: text**



Architecture is unified across tasks with *text-to-text* format

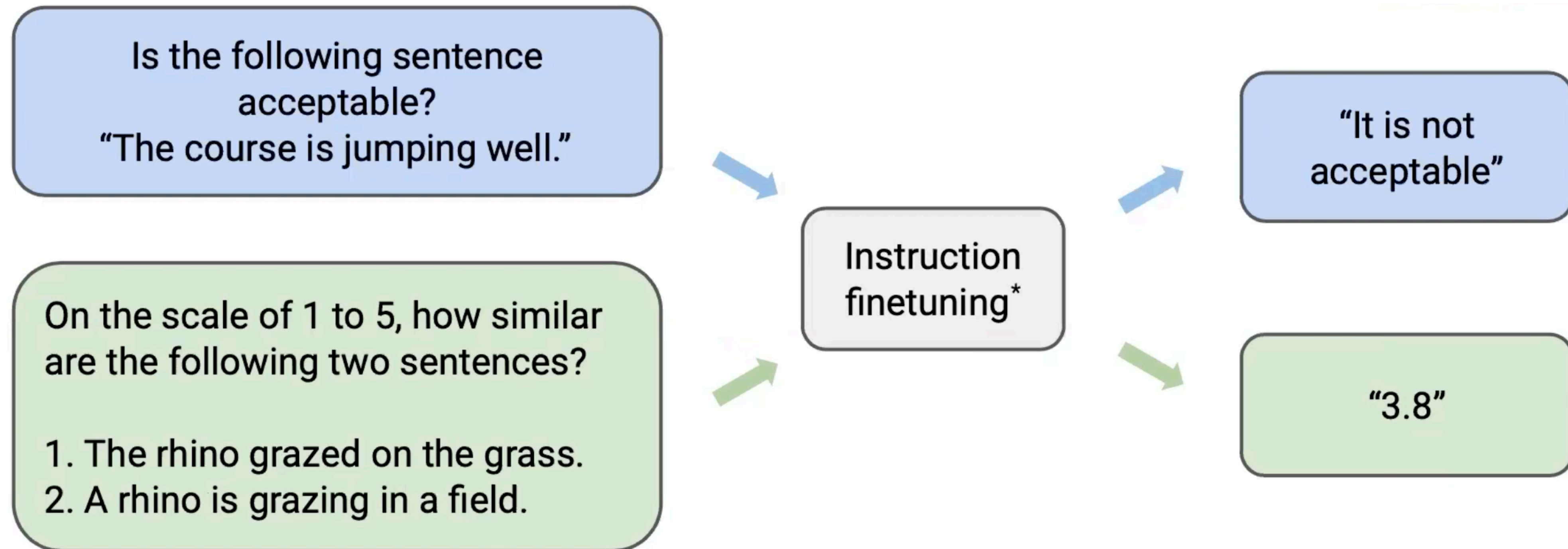
# T5: Limitations



Tasks are not semantically related

# Instruction Fine-Tuning

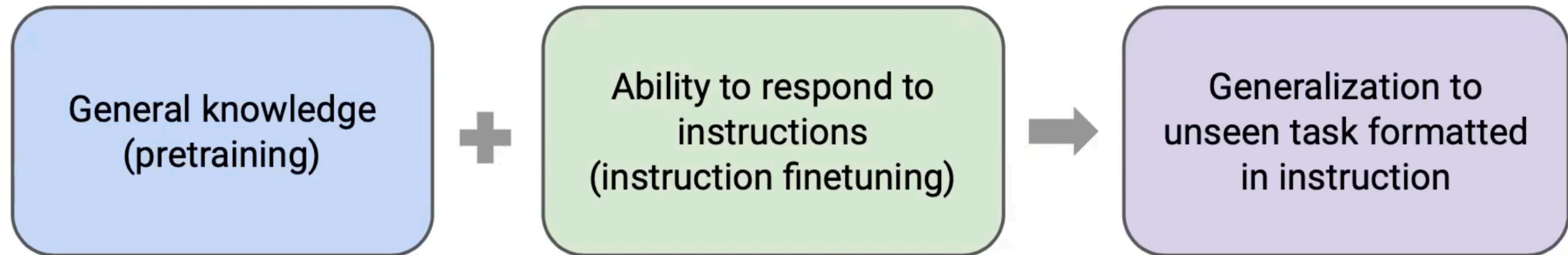
[Wei+ 2021,  
Sanh+ 2021,  
Ouyang+ 2022]



Tasks are unified. So for an unseen task, the model just needs to respond to the natural language instruction

# Instruction Fine-Tuning

[Wei+ 2021,  
Sanh+ 2021,  
Ouyang+ 2022]



# Why Instruction Tuning?

- (Pre-trained) LLMs are not aligned with user intention
  - They have a massive knowledge from the web-scaled dataset
  - But they don't learn to follow users' instructions

**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.

# Why Instruction Tuning?

- Solution: Finetune LLMs to follow the human instruction

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

**COMPLETION**

**Human**

A giant rocket ship blasted off from Earth carrying astronauts to the moon. The astronauts landed their spaceship on the moon and walked around exploring the lunar surface. Then they returned safely back to Earth, bringing home moon rocks to show everyone.

- This fine-tuning process is also called **alignment**

# Instruction Tuning

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

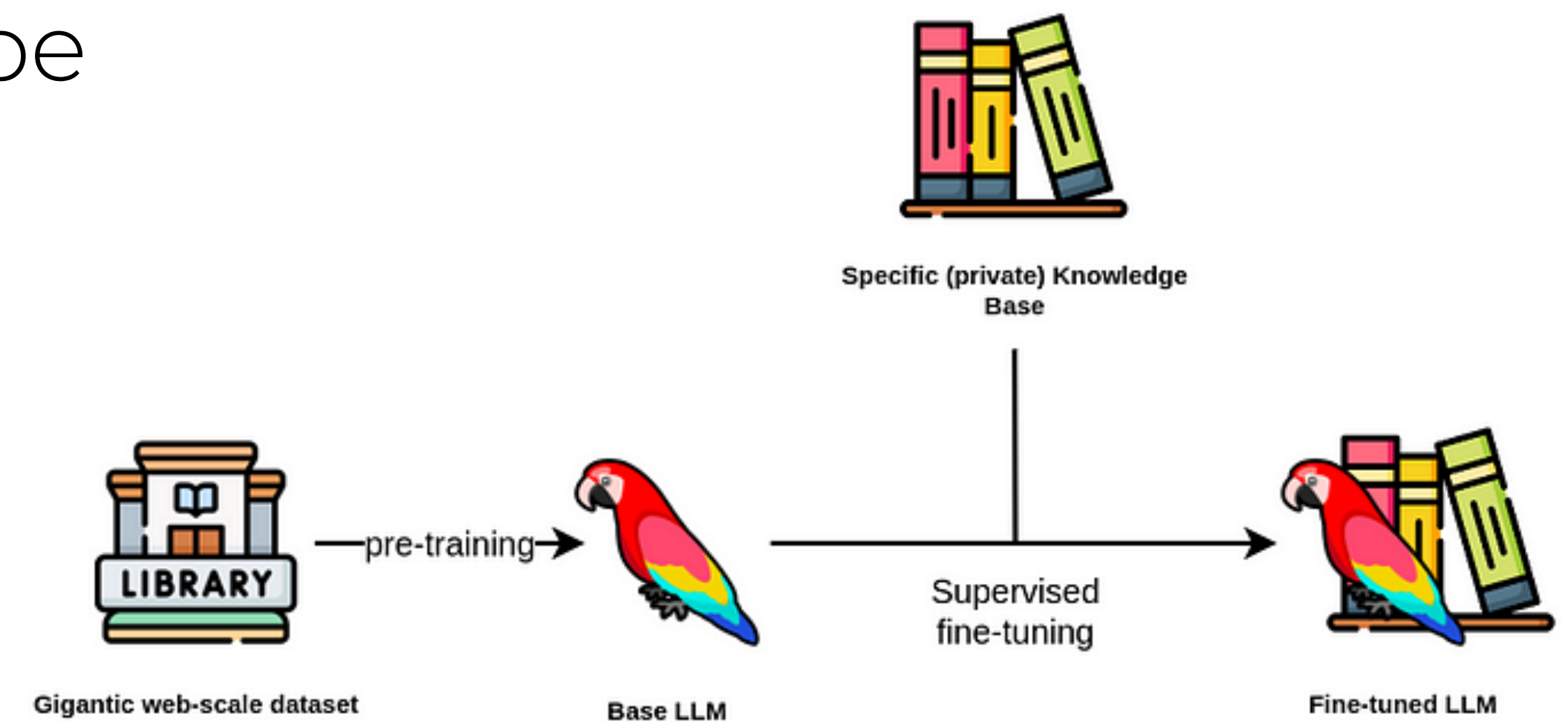
**COMPLETION** **Human**

A giant rocket ship blasted off from Earth carrying astronauts to the moon. The astronauts landed their spaceship on the moon and walked around exploring the lunar surface. Then they returned safely back to Earth, bringing home moon rocks to show everyone.

- Collect prompt and completion (response) set
- Now, the prompt is an input  $X$  and completion is a target  $Y$
- With  $(X, Y)$ , fine-tune the pre-trained LLM with NTP loss

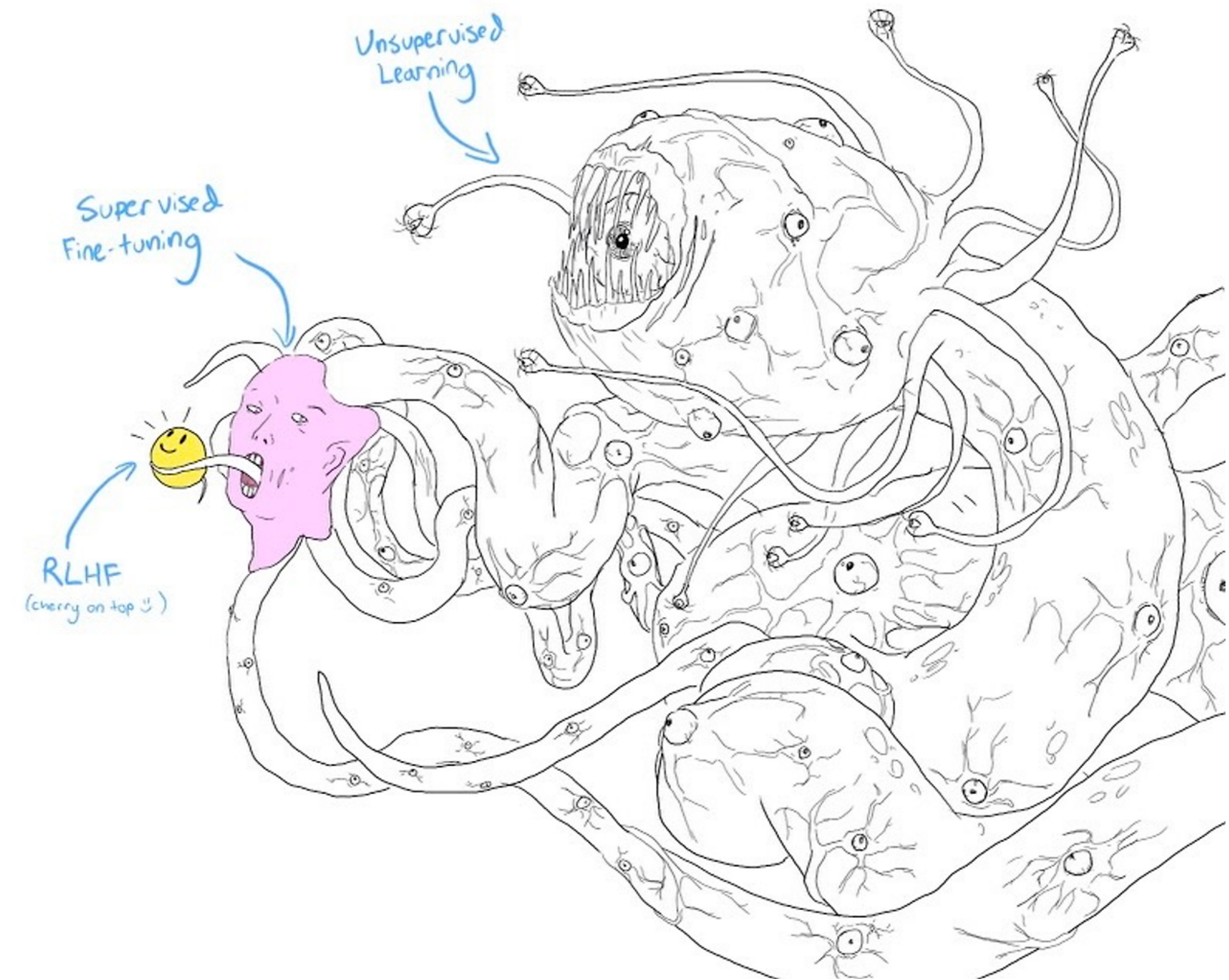
# Supervised Fine-Tuning (SFT)

- The term "**supervised fine-tuning (SFT)**" is more widely used than "**instruction fine-tuning**" (or instruction tuning)
- The logic behind these two terms is the same, but
  - Instruction fine-tuning is called "instruction fine-tuning" because this specifically fine-tunes models to follow user instructions
  - SFT literally refers to a "supervised" fine-tuning approach; it can cover instruction dataset, but has to be



# Recap

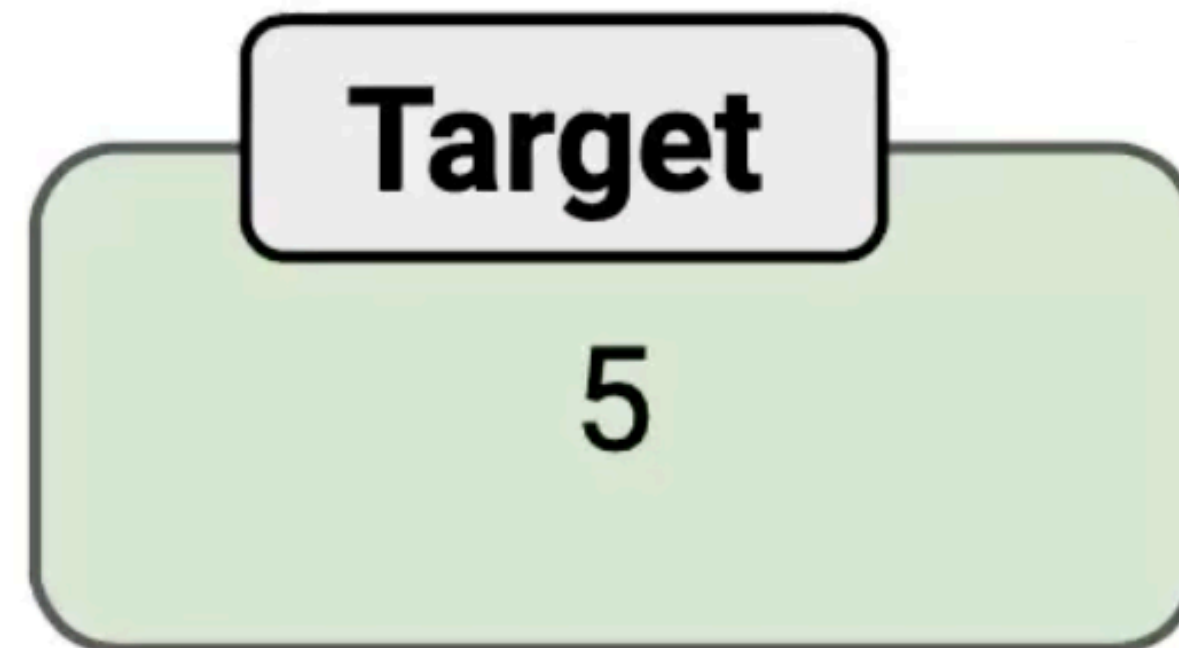
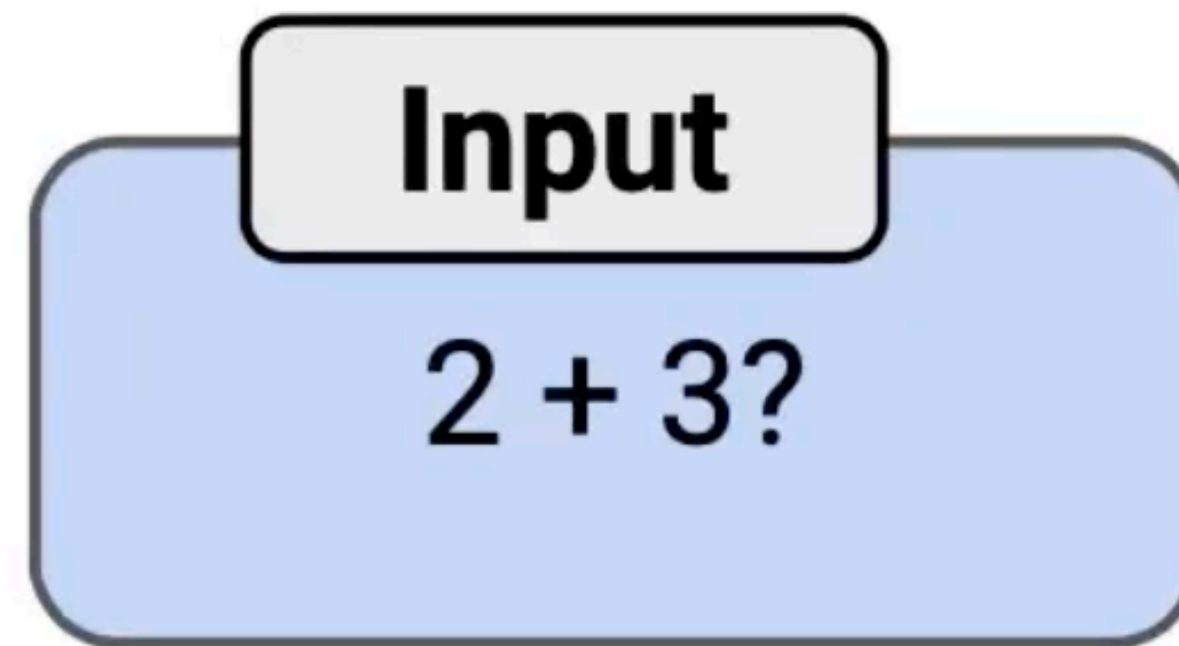
- Pre-training
  - Grasp a massive knowledge from the web-scale dataset
  - Unaligned, hard to use
- SFT
  - Align pre-trained LLM to follow instructions from the users
  - Further training to learn extra knowledge
- Both training procedures use next-token prediction



# Recall: Instruction Fine-Tuning

- For a given input, **the target is the single correct answer**
- In RL, this is called behavior cloning
  
- Hope is that, if we have enough data, the model can learn to generalize
- This requires formalizing the correct behavior for a given input

# Single Correct Answer?



# Single Correct Answer?

**Input**

What is the sentiment of "I should have studied instead of watching this movie"?

**Target**

negative

# Single Correct Answer?

**Input**

Translate this to Korean:  
“I should have studied instead of watching this movie”

**Target**

나는 이 영화 보는 대신 공부를 했어야 했다

# Single Correct Answer?

## Input

Write a letter to a 5-year-old boy from Santa Clause explaining that Santa is not real. Convey gently so as not to break his heart

## Target



# Single Correct Answer?

## Input

Implement logistic regression with gradient descent in Python

## Target

```
class LogisticRegression:
```

```
...
```

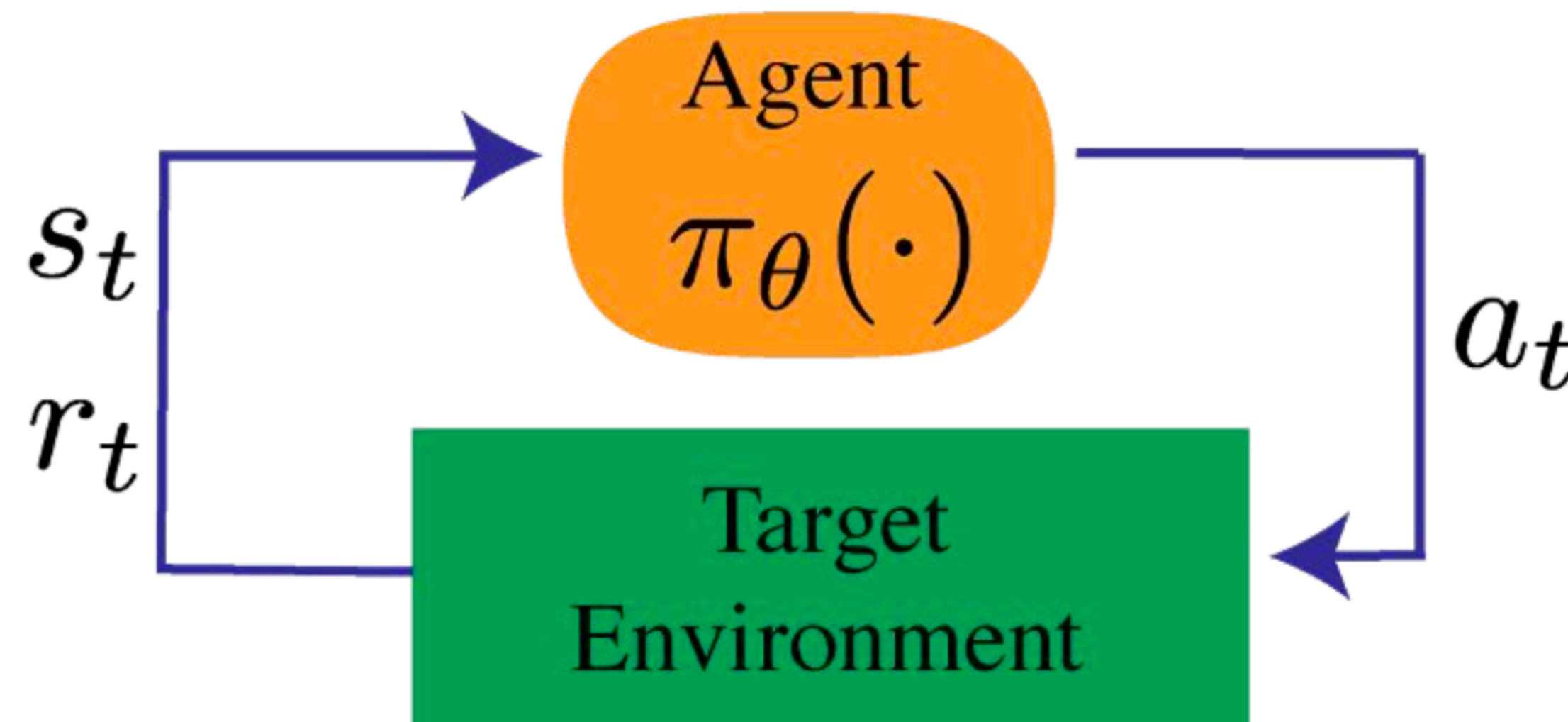
# Another Aspect

- Increasingly we want to teach models more abstract behaviors
- Objective function of instruction fine-tuning (NTP from supervised dataset) seems to be the "bottleneck" of teaching these behaviors
- The maximum likelihood objective is a **predefined** function (i.e. no learnable parameters in the loss function)
- Can we parameterize the objective function and **learn** it?

# TL;DR: Use Reinforcement Learning

- Instead of using labeled text, we use **human preference dataset**
  - We want to make LLM to maximize human preference
  - How to train the model? use RL
- In RL, we try to maximize the expected **reward function**  $\max_{\theta} \mathbb{E}[R]$
- Reward is the objective function and we can learn the reward
  - By training the reward model using the human preference dataset
  - So, reward model predicts the human preference
  - RL-trained LLM maximizes reward from the reward model

# Reinforcement Learning



Some notation:

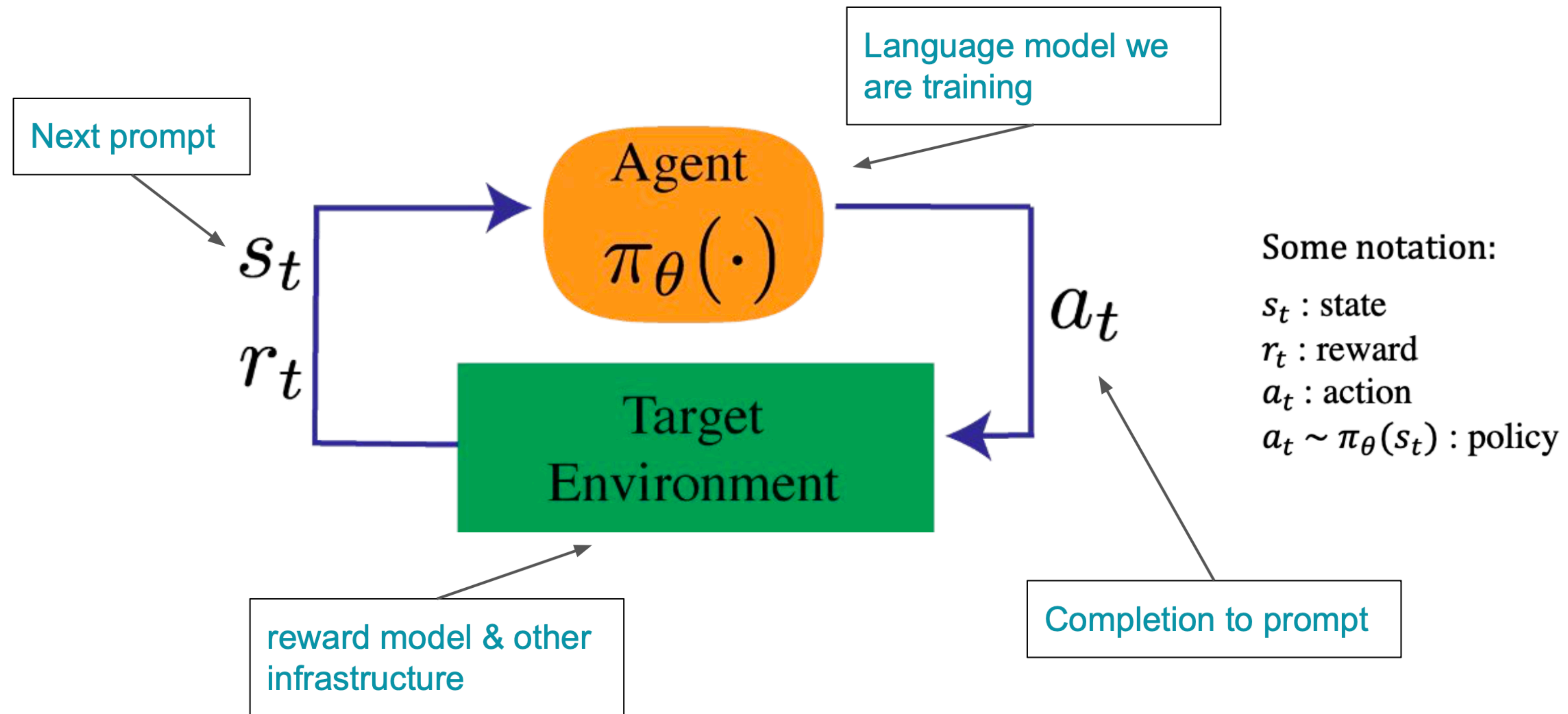
$s_t$  : state

$r_t$  : reward

$a_t$  : action

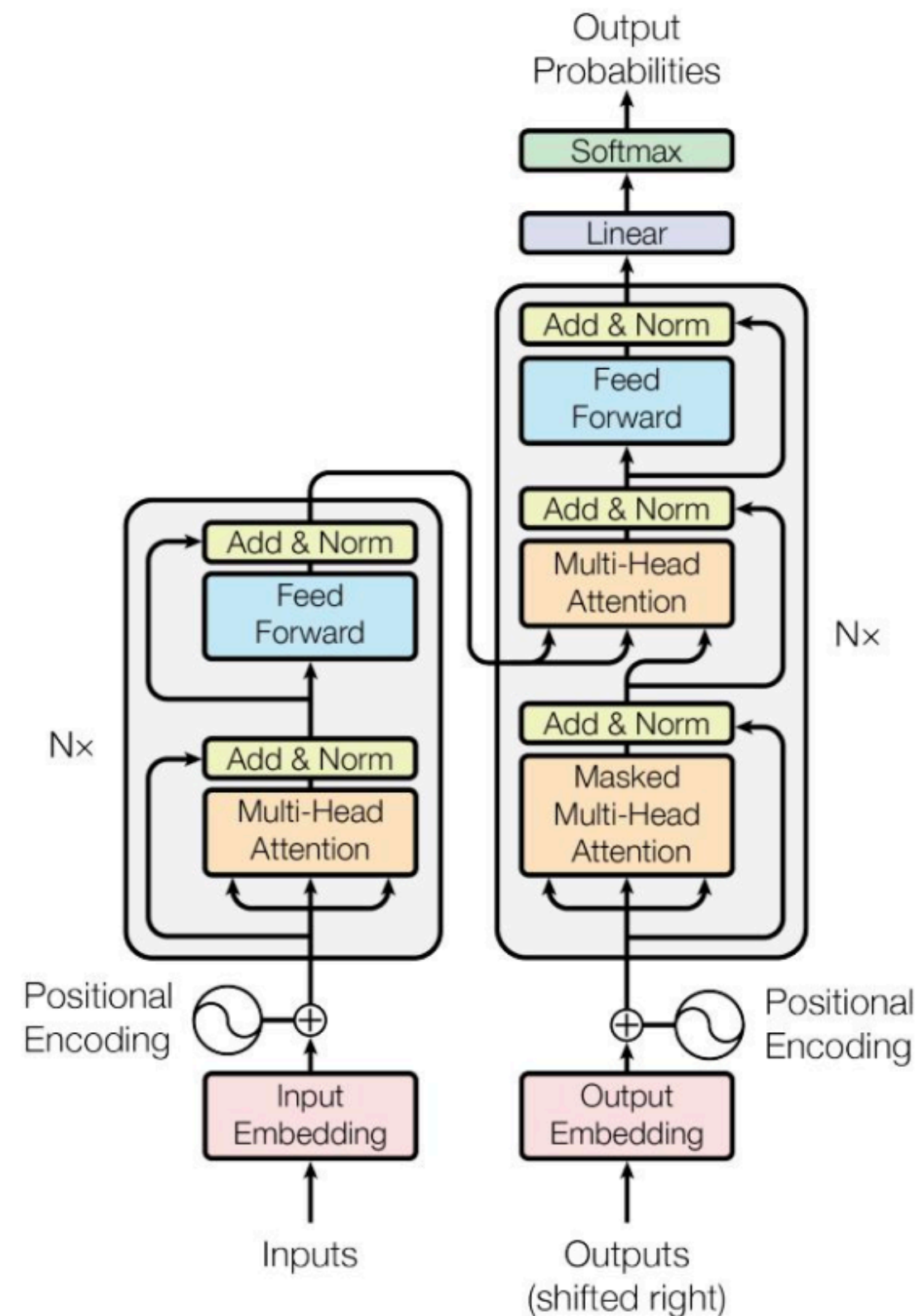
$a_t \sim \pi_{\theta}(s_t)$  : policy

# Reinforcement Learning



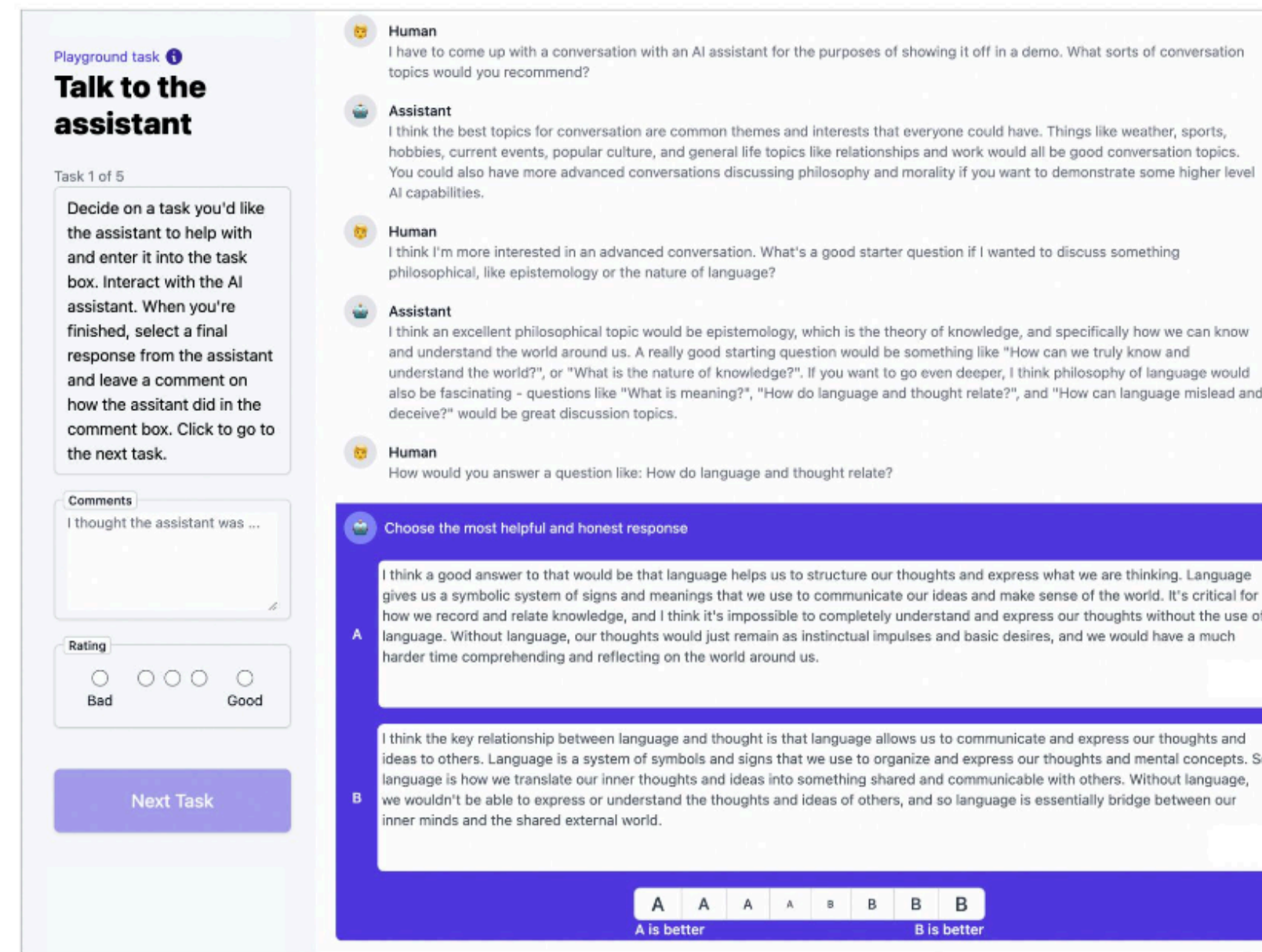
# RL from Human Feedback

base model (instruction, helpful, chatty etc.)

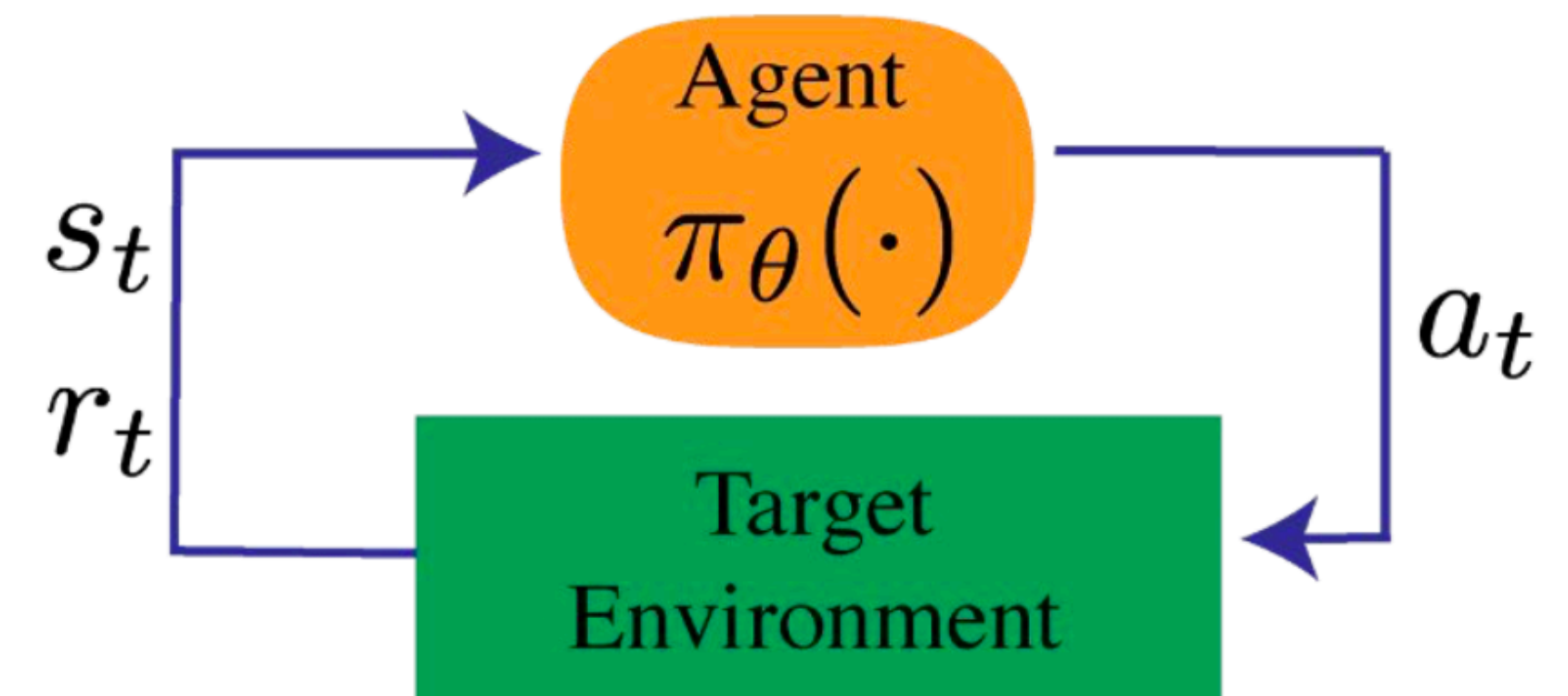


Vaswani et al. 2017

## preference collection & training

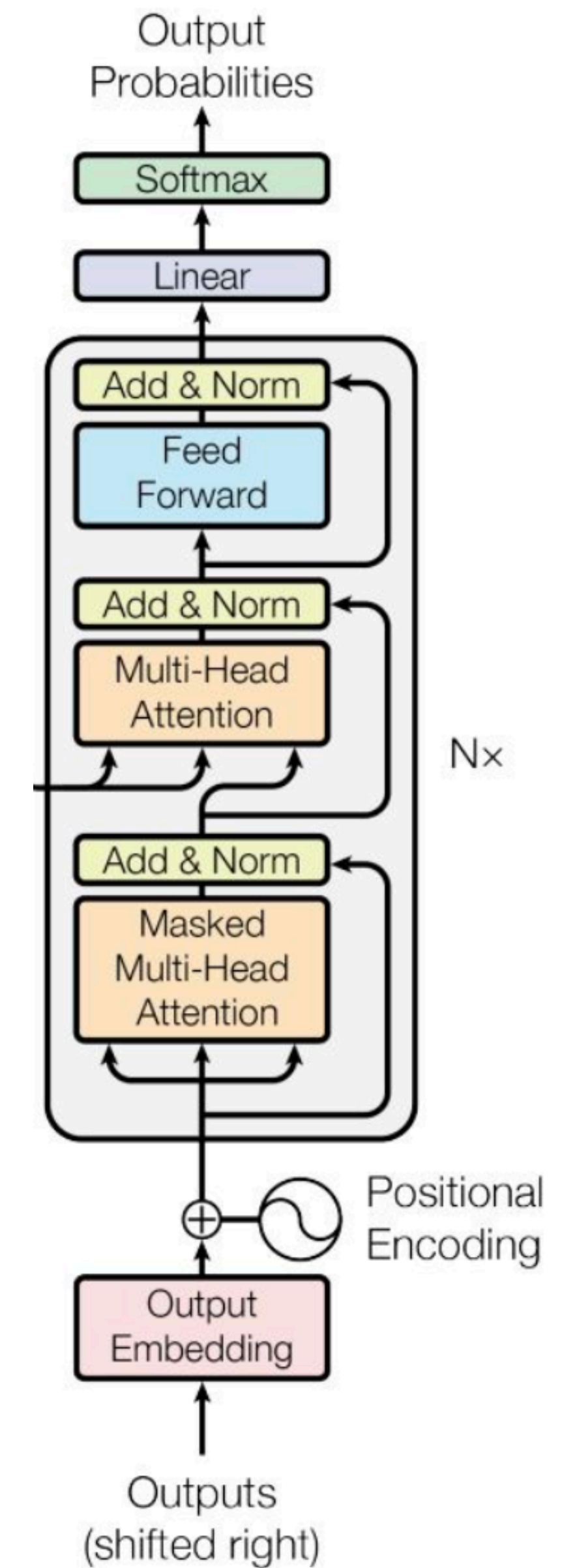


## reinforcement learning optimization



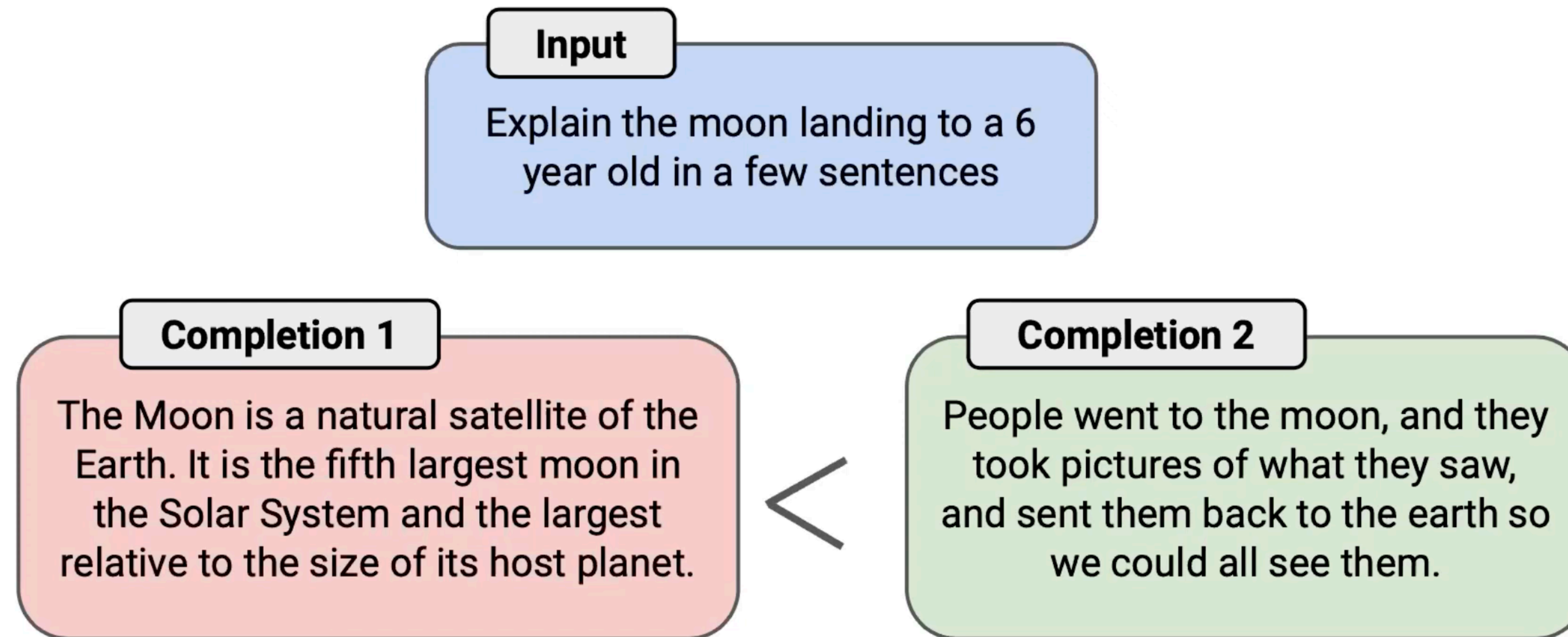
# 1. Instruction Fine-Tuned Model

- Starting point: a base LLM + instruction fine-tuning



## 2. Preference Collection

- Which completion (answer from the input) is better?



- Humans label which completion is "preferred"
- This setup aims to align models to the human preference

# 2. Preference Collection

- Feedback collection interface

**Playground task** ⓘ  
**Talk to the assistant**

Task 1 of 5

Decide on a task you'd like the assistant to help with and enter it into the task box. Interact with the AI assistant. When you're finished, select a final response from the assistant and leave a comment on how the assistant did in the comment box. Click to go to the next task.

**Comments**  
I thought the assistant was ...

**Rating**

○ ○ ○ ○ ○  
Bad Good

**Next Task**

**Human**  
I have to come up with a conversation with an AI assistant for the purposes of showing it off in a demo. What sorts of conversation topics would you recommend?

**Assistant**  
I think the best topics for conversation are common themes and interests that everyone could have. Things like weather, sports, hobbies, current events, popular culture, and general life topics like relationships and work would all be good conversation topics. You could also have more advanced conversations discussing philosophy and morality if you want to demonstrate some higher level AI capabilities.

**Human**  
I think I'm more interested in an advanced conversation. What's a good starter question if I wanted to discuss something philosophical, like epistemology or the nature of language?

**Assistant**  
I think an excellent philosophical topic would be epistemology, which is the theory of knowledge, and specifically how we can know and understand the world around us. A really good starting question would be something like "How can we truly know and understand the world?", or "What is the nature of knowledge?". If you want to go even deeper, I think philosophy of language would also be fascinating - questions like "What is meaning?", "How do language and thought relate?", and "How can language mislead and deceive?" would be great discussion topics.

**Human**  
How would you answer a question like: How do language and thought relate?

**Choose the most helpful and honest response**

**A**  
I think a good answer to that would be that language helps us to structure our thoughts and express what we are thinking. Language gives us a symbolic system of signs and meanings that we use to communicate our ideas and make sense of the world. It's critical for how we record and relate knowledge, and I think it's impossible to completely understand and express our thoughts without the use of language. Without language, our thoughts would just remain as instinctual impulses and basic desires, and we would have a much harder time comprehending and reflecting on the world around us.

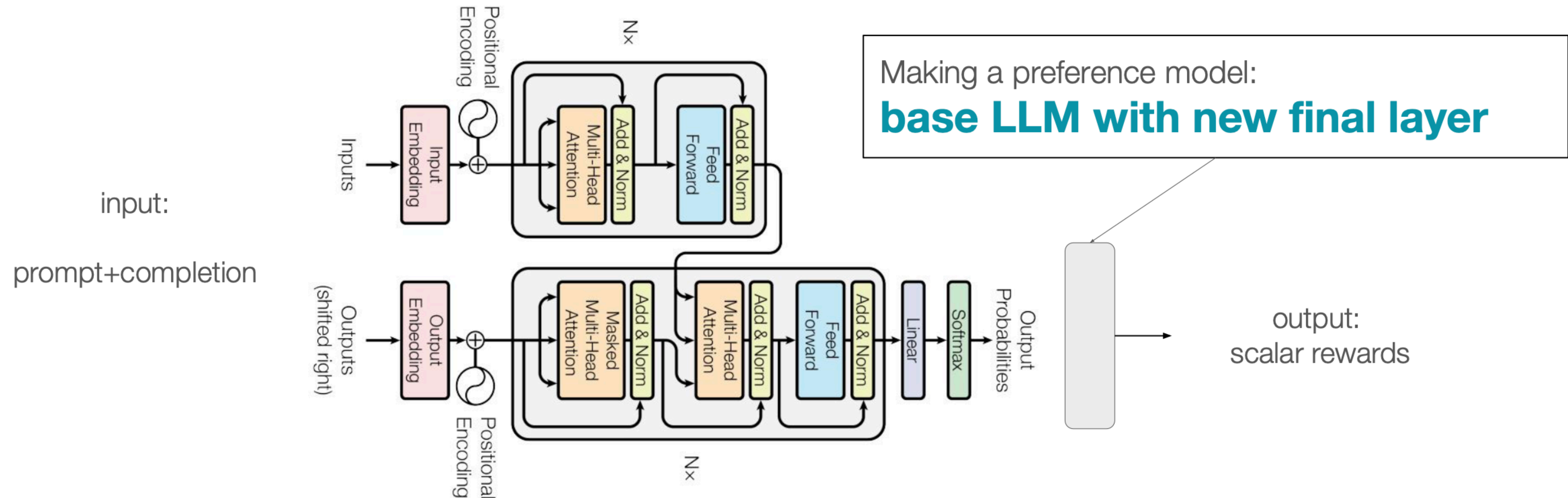
**B**  
I think the key relationship between language and thought is that language allows us to communicate and express our thoughts and ideas to others. Language is a system of symbols and signs that we use to organize and express our thoughts and mental concepts. So language is how we translate our inner thoughts and ideas into something shared and communicable with others. Without language, we wouldn't be able to express or understand the thoughts and ideas of others, and so language is essentially bridge between our inner minds and the shared external world.

**A is better** **B is better**

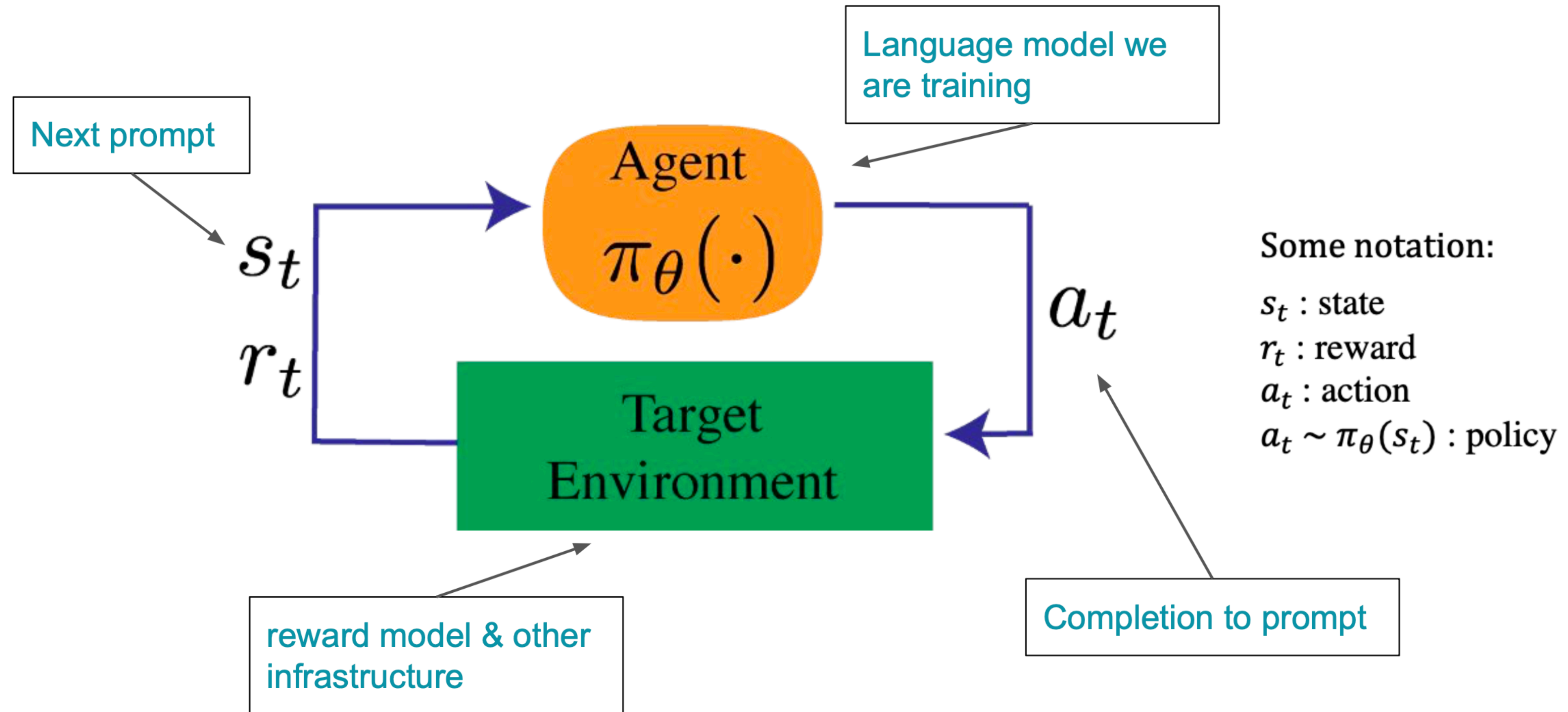
RLHF at ICML 2023, 34

# 3. Train Reward Model

- It is also called preference model
- Starting point: a base instruction-tuned LLM
  - We train the reward model to predict human preference



# 4. RL Training

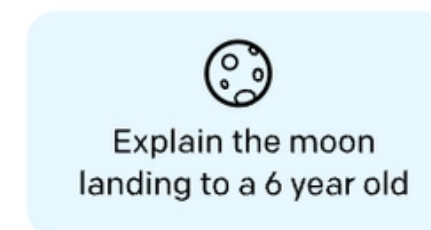


# Summary

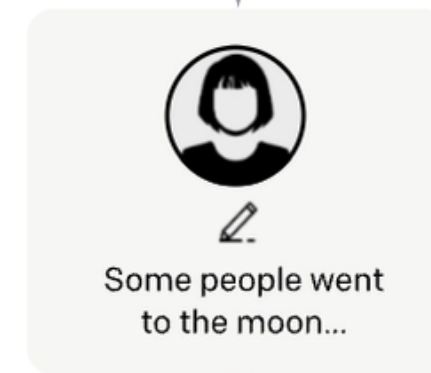
Step 1

**Collect demonstration data, and train a supervised policy.**

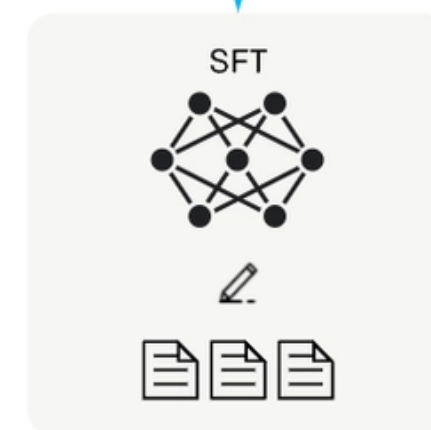
A prompt is sampled from our prompt dataset.



A labeler demonstrates the desired output behavior.



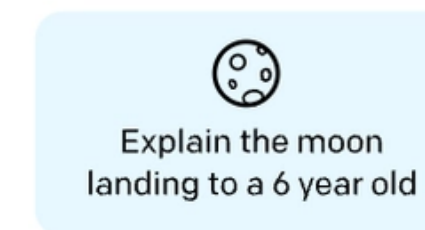
This data is used to fine-tune GPT-3 with supervised learning.



Step 2

**Collect comparison data, and train a reward model.**

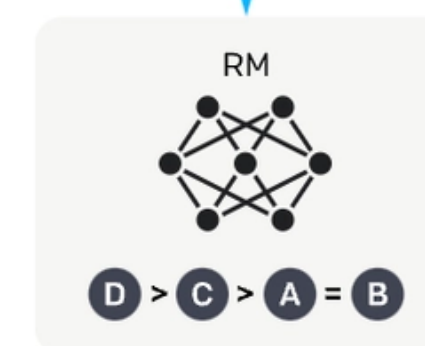
A prompt and several model outputs are sampled.



A labeler ranks the outputs from best to worst.



This data is used to train our reward model.



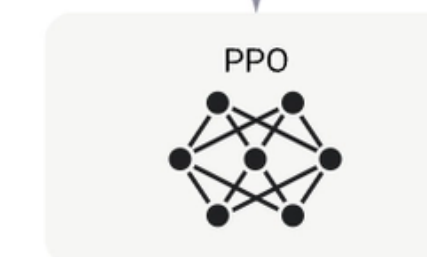
Step 3

**Optimize a policy against the reward model using reinforcement learning.**

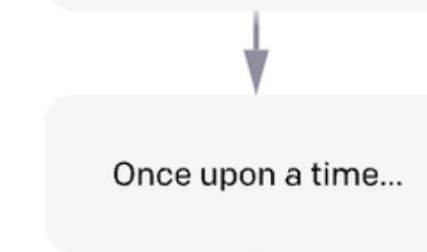
A new prompt is sampled from the dataset.



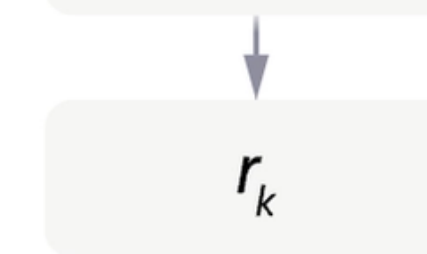
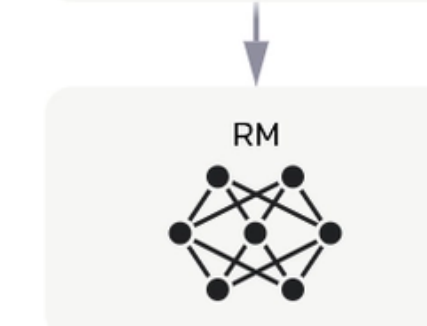
The policy generates an output.



The reward model calculates a reward for the output.



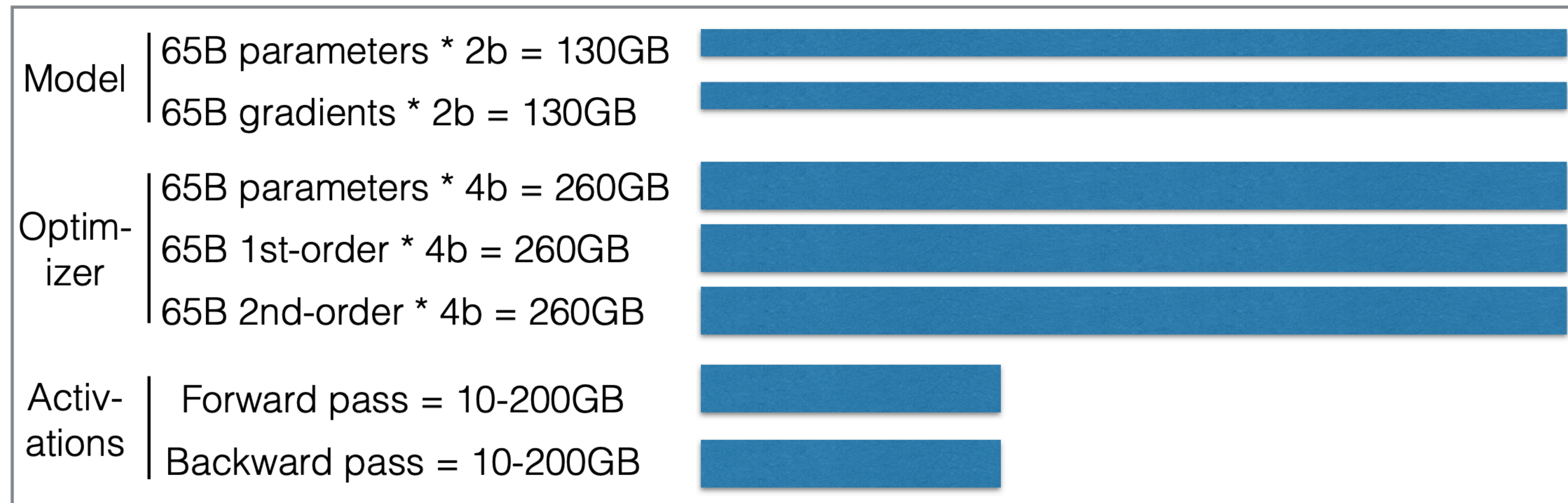
The reward is used to update the policy using PPO.



# Fine-Tuning

# Full Fine-Tuning

- Simply continue training the (pre-trained) LM on the new output
  - But it can take lots of memory!
  - Example: Training 65B params model with FP16 mixed precision



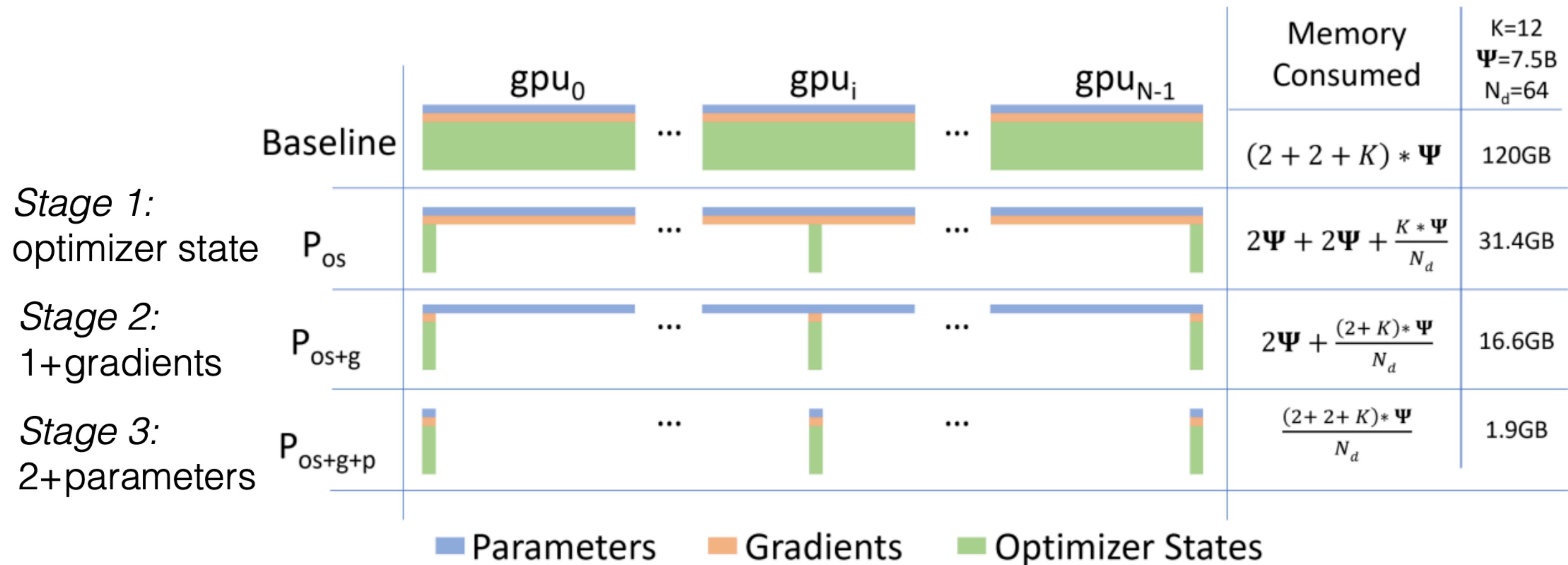
1000-1400GB of GPU memory! [Rajbhandari+ 2019]

# Notes: GPU Specs

GPU	Memory	Cost (2/2024)	(Cloud) Machines
T40 / K80	24GB	\$150	Google Colab, AWS p2.*
V100	32GB	\$2,500	Google Colab
A100	40GB or 80GB	\$8,000/\$16,000	Google Colab, AWS p3.*
H100	80GB	\$44,000	AWS p4.*
6000 Ada, L40	48GB	\$8000	N/A
Mac M*	Same as CPU	\$2000	N/A

# Multi-GPU Training

- One solution: buy and use more hardware at it!
- We will see parallelism, but this takes lots of GPUs

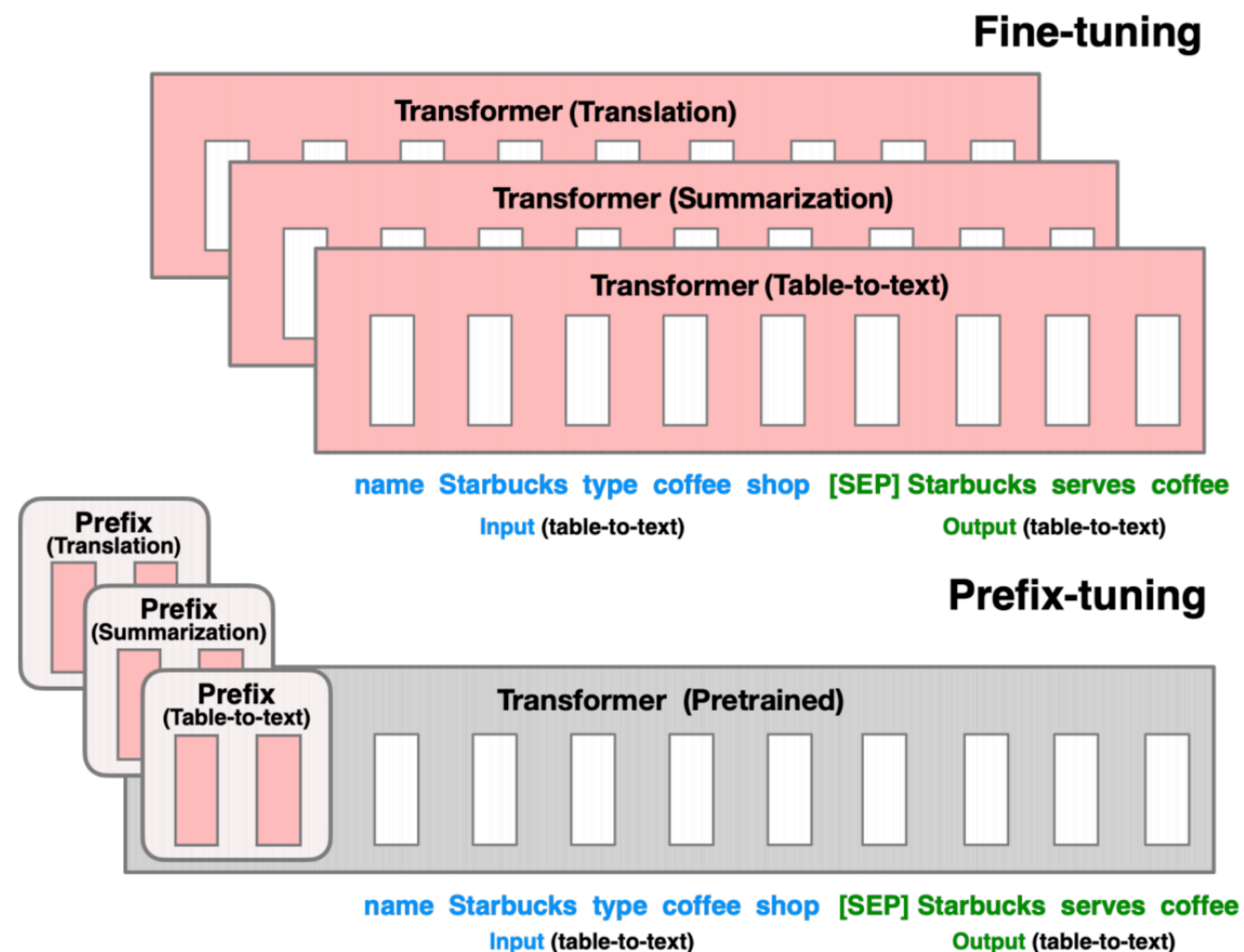


# Parameter-Efficient Fine-Tuning (PEFT)

- We don't have money to buy all this GPUs
- Don't tune all of the parameters, but just some
  - Prompt/prefix tuning
  - Adapters
  - LoRA

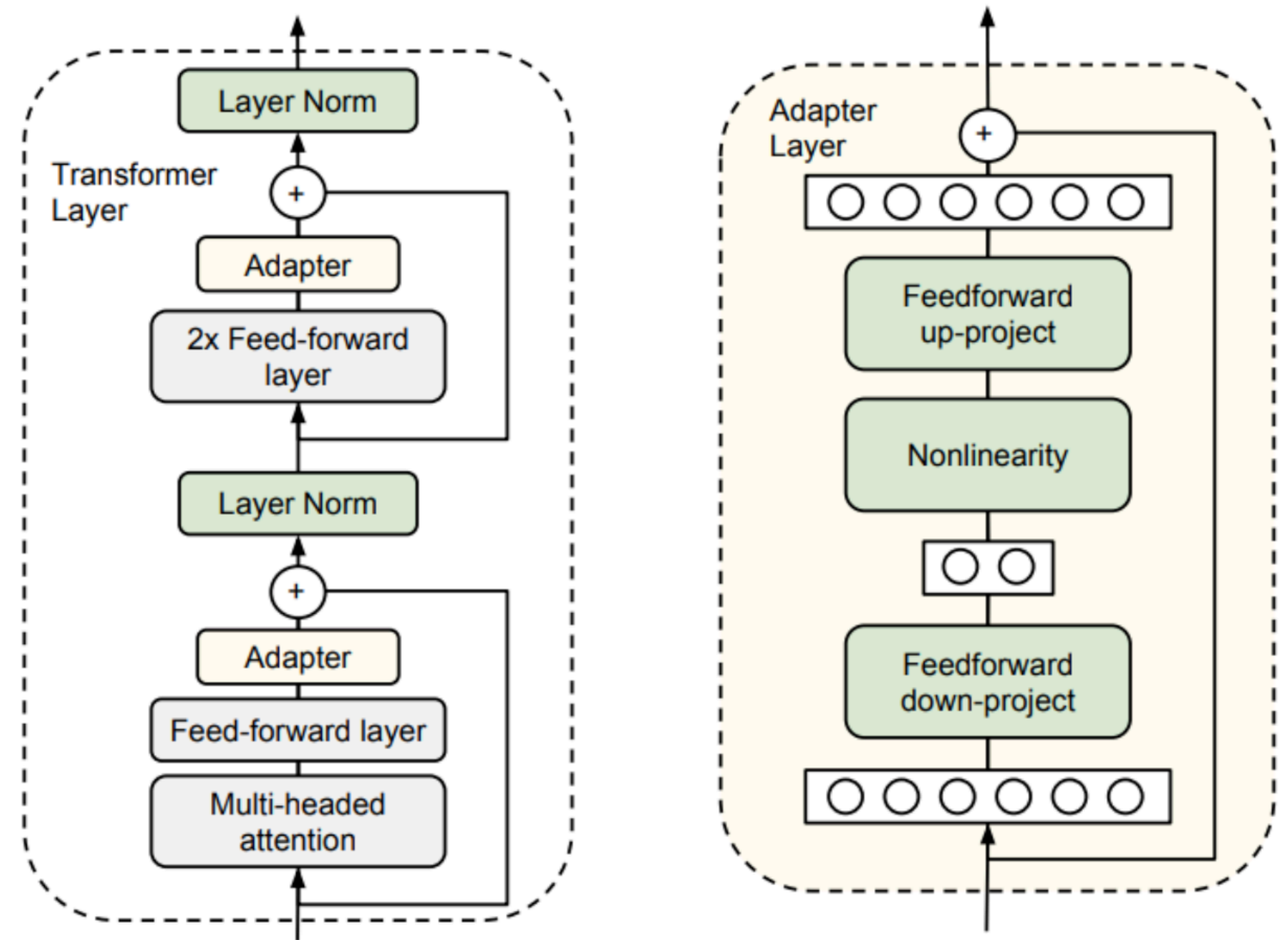
# Prompt/Prefix Tuning [Li+ 2021, Lester+ 2021]

- Prompt tuning optimizes only the embedding layer
- Prefix tuning optimizes the prefix of all layers



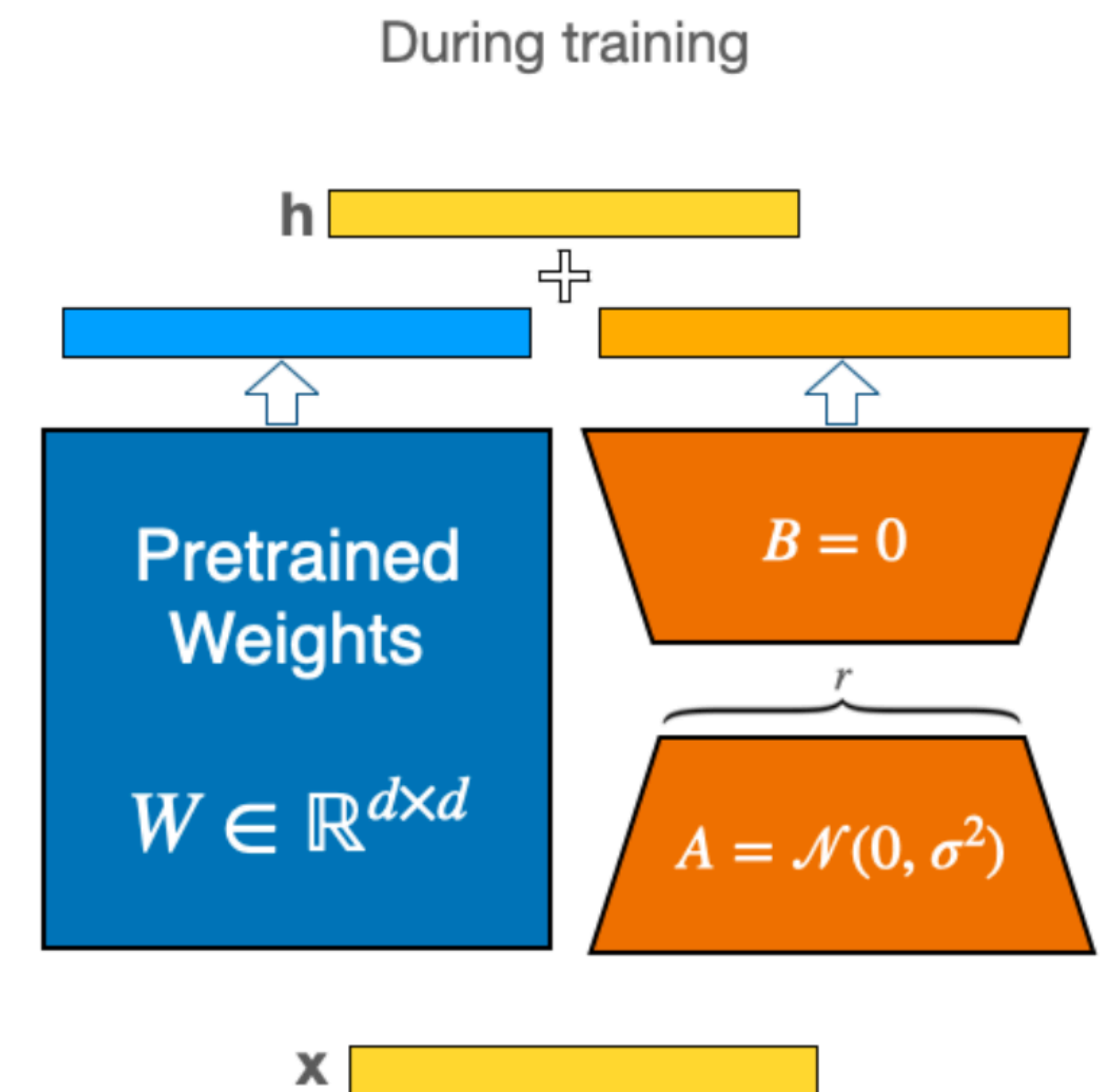
# Adapters [Houlsby+ 2019]

- Sandwich in layers in a pre-trained LLM and only update the adapters
  - These layers only have  $2 \times \text{model\_dim} \times \text{adapter\_dim}$  params
- 🙌 It modifies the model structure

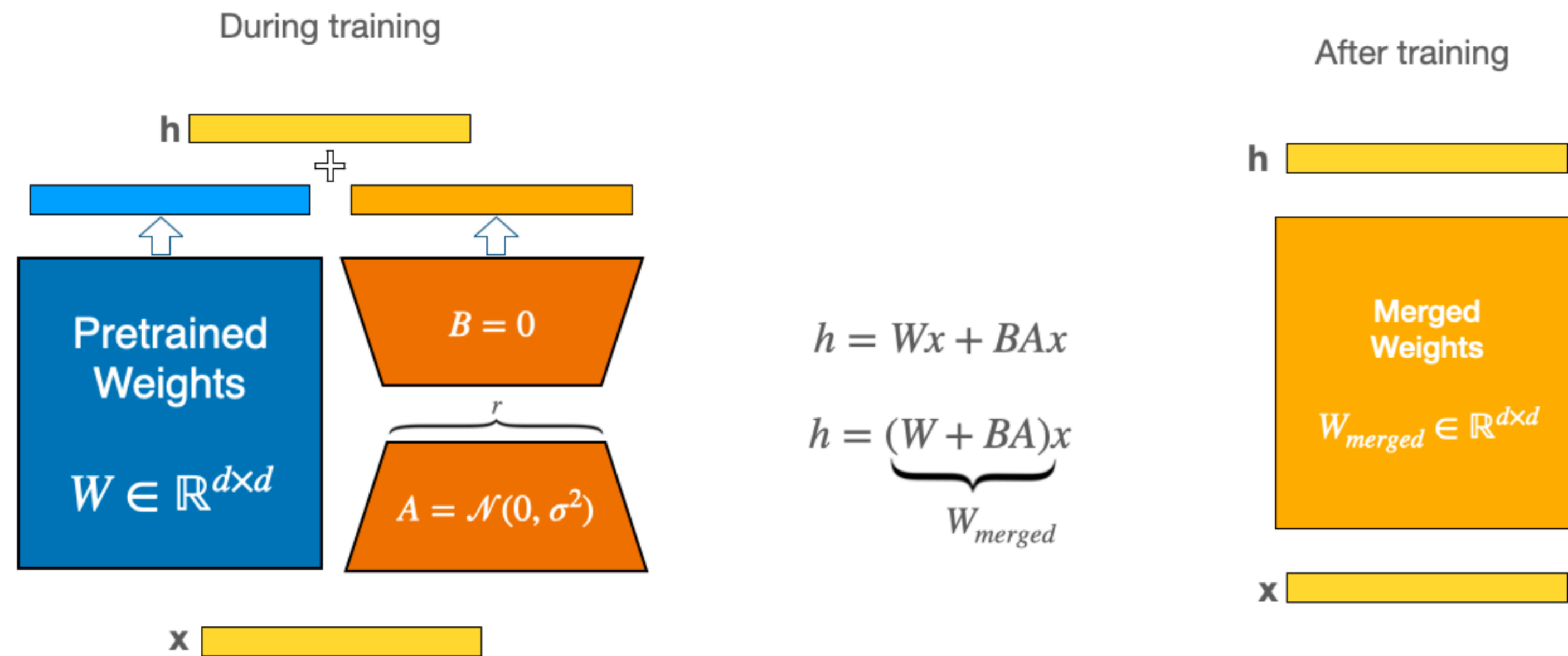


# Low-Rank Adaptation (LoRA) [Hu+ 2021]

- $W_0 \in \mathbb{R}^{d \times k}$ : a pre-trained weight matrix
- Constrain update with a **low-rank** decomposition
  - $W' = W_0 + \Delta W = W_0 + \alpha BA$
  - where  $B \in \mathbb{R}^{d \times r}$ ,  $A \in \mathbb{R}^{r \times k}$ ,  $r \ll \min(d, k)$  (row-rank)
- $\alpha$  is the trade-off between pre-trained knowledge and task-specific knowledge
- Only  $A, B$  contain trainable parameters
  - $A$  = random initialization
  - $B$  = zero initialization (so  $\Delta W$  starts from zero)



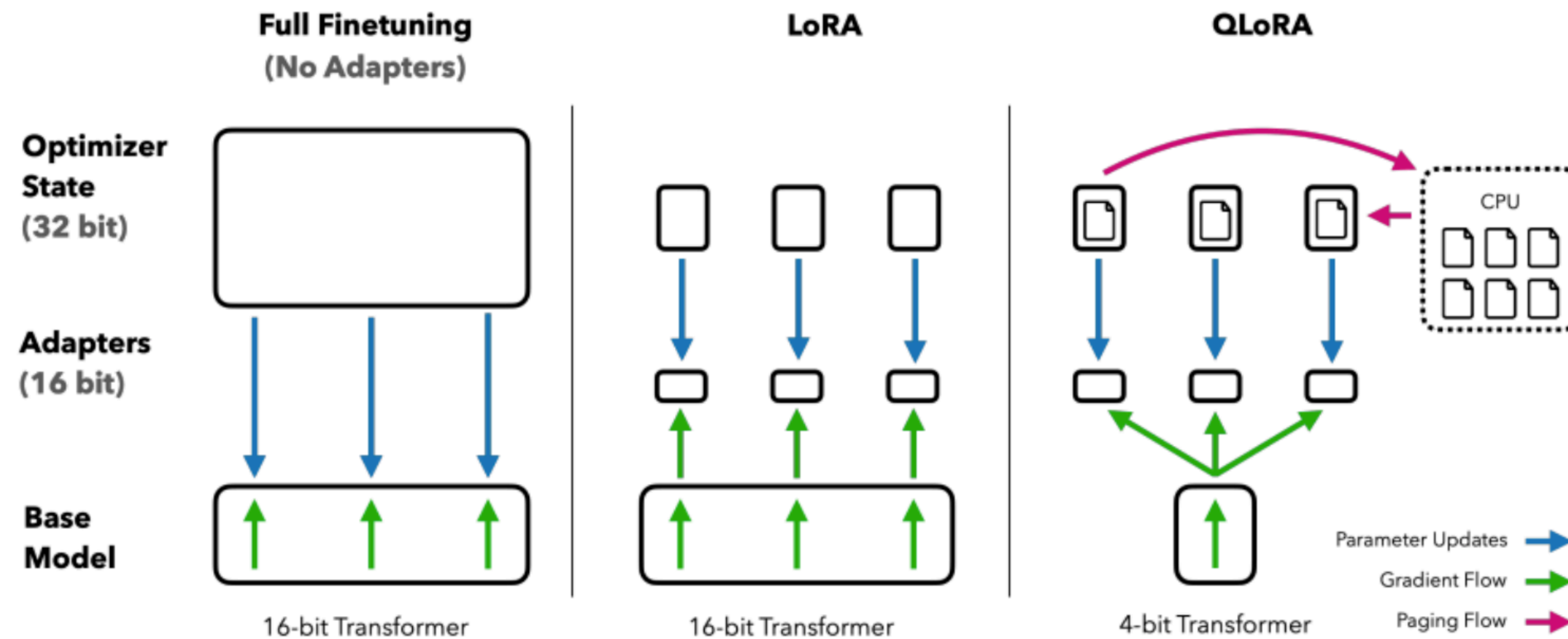
# Low-Rank Adaptation (LoRA) [Hu+ 2021]



- Freeze pre-trained weights, train only the low-rank approximation of difference (residual) from the pre-trained weights
- 👍 after training, just add in to pre-trained weights; no new components!

# Q-LoRA [Dettmers+ 2023]

- Further compress memory requirements for training by
  - 4-bit quantization of the pre-trained model
  - Use of GPU memory paging to prevent OOM



- We can train a 65B model on a 48GB GPU

# A Unified View of PEFT [He+ 2021]

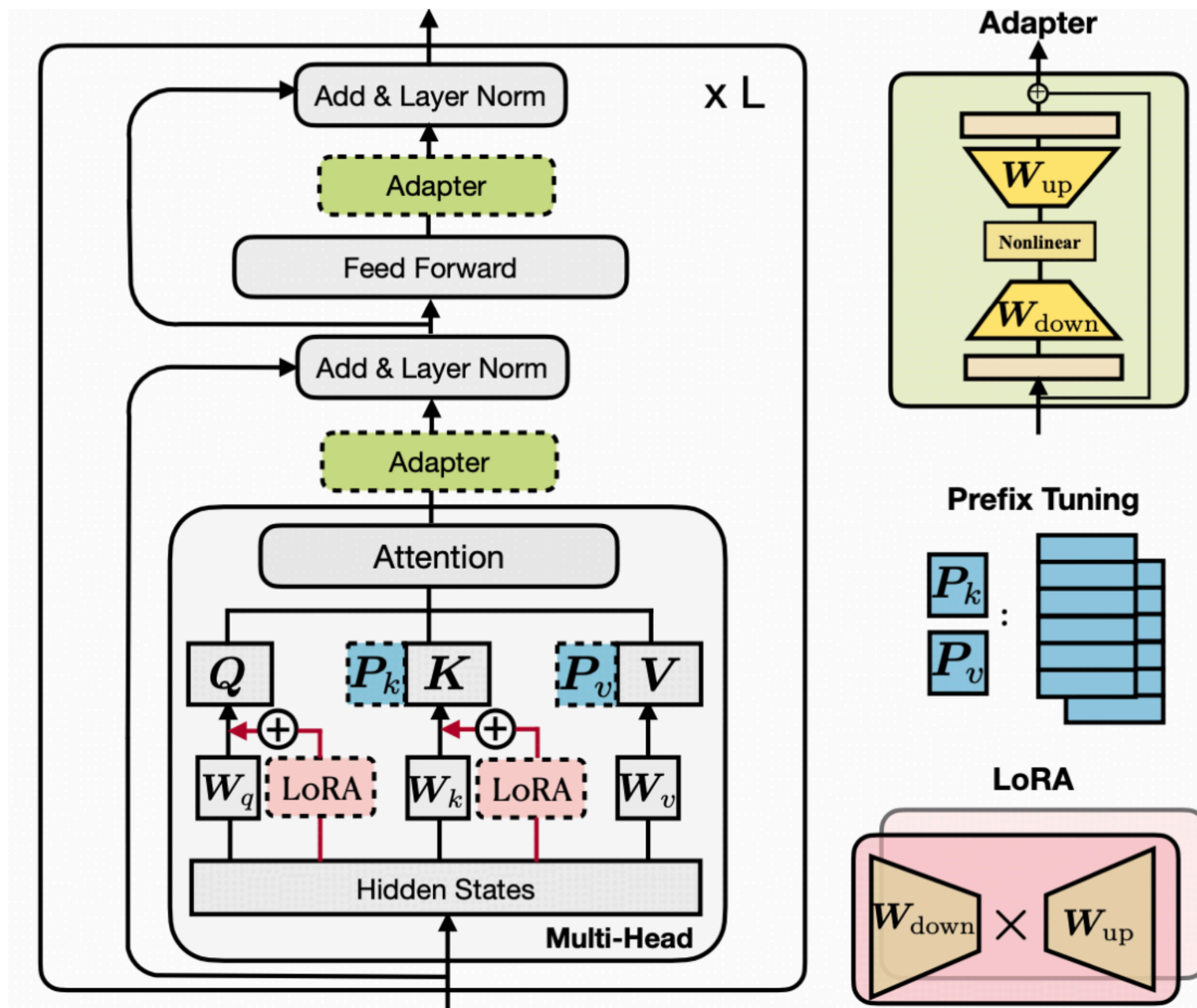


Figure 1: Illustration of the transformer architecture and several state-of-the-art parameter-efficient tuning methods. We use blocks with dashed borderlines to represent the added modules by those methods.

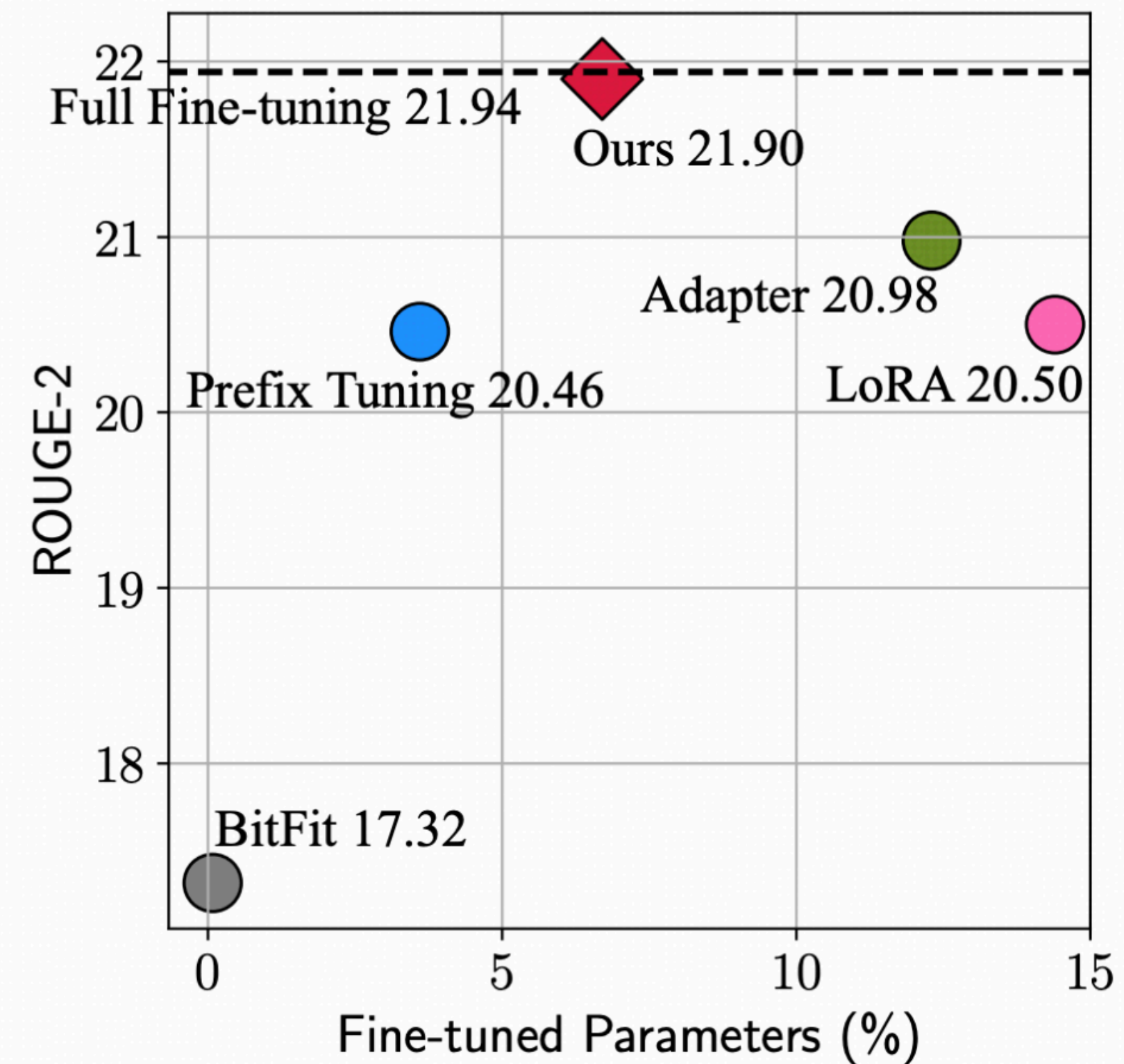


Figure 2: Performance of different methods on the XSum (Narayan et al., 2018) summarization task. The number of fine-tuned parameters is relative to the tuned parameters in full fine-tuning.

# A Unified View of PEFT [He+ 2021]

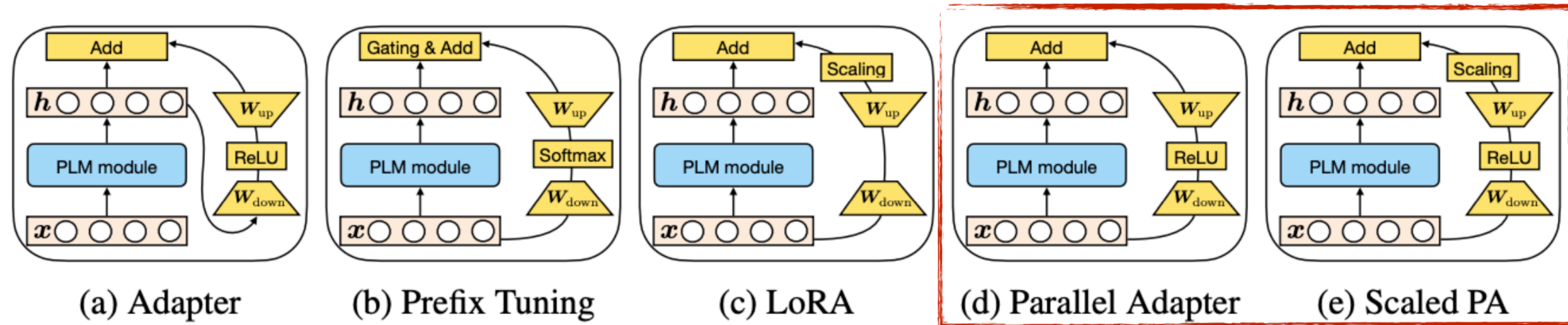


Table 1: Parameter-efficient tuning methods decomposed along the defined design dimensions. Here, for clarity, we directly write the adapter nonlinear function as ReLU which is commonly used. The bottom part of the table exemplifies new variants by transferring design choices of existing approaches.

Method	$\Delta h$ functional form	insertion form	modified representation	composition function
<b>Existing Methods</b>				
Prefix Tuning	$\text{softmax}(xW_q P_k^\top) P_v$	parallel	head attn	$h \leftarrow (1 - \lambda)h + \lambda \Delta h$
Adapter	$\text{ReLU}(hW_{\text{down}})W_{\text{up}}$	sequential	ffn/attn	$h \leftarrow h + \Delta h$
LoRA	$xW_{\text{down}}W_{\text{up}}$	parallel	attn key/val	$h \leftarrow h + s \cdot \Delta h$
<b>Proposed Variants</b>				
Parallel adapter	$\text{ReLU}(hW_{\text{down}})W_{\text{up}}$	parallel	ffn/attn	$h \leftarrow h + \Delta h$
Muti-head parallel adapter	$\text{ReLU}(hW_{\text{down}})W_{\text{up}}$	parallel	head attn	$h \leftarrow h + \Delta h$
Scaled parallel adapter	$\text{ReLU}(hW_{\text{down}})W_{\text{up}}$	parallel	ffn/attn	$h \leftarrow h + s \cdot \Delta h$

# Which One to Choose?

- **Convenience:**
  - LoRA doesn't change model architecture
- **Accuracy:**
  - Simpler tasks (e.g. classification): probably doesn't matter much
  - More complex tasks + small budget: prefix tuning seems favorable
  - More complex tasks + larger budget: adapters and LoRA
- **In most case, LoRA-family is the best option to choose**

# DoRA: Weight-Decomposed LoRA [Liu+ 2024]

- Decompose weight into **magnitude and direction**

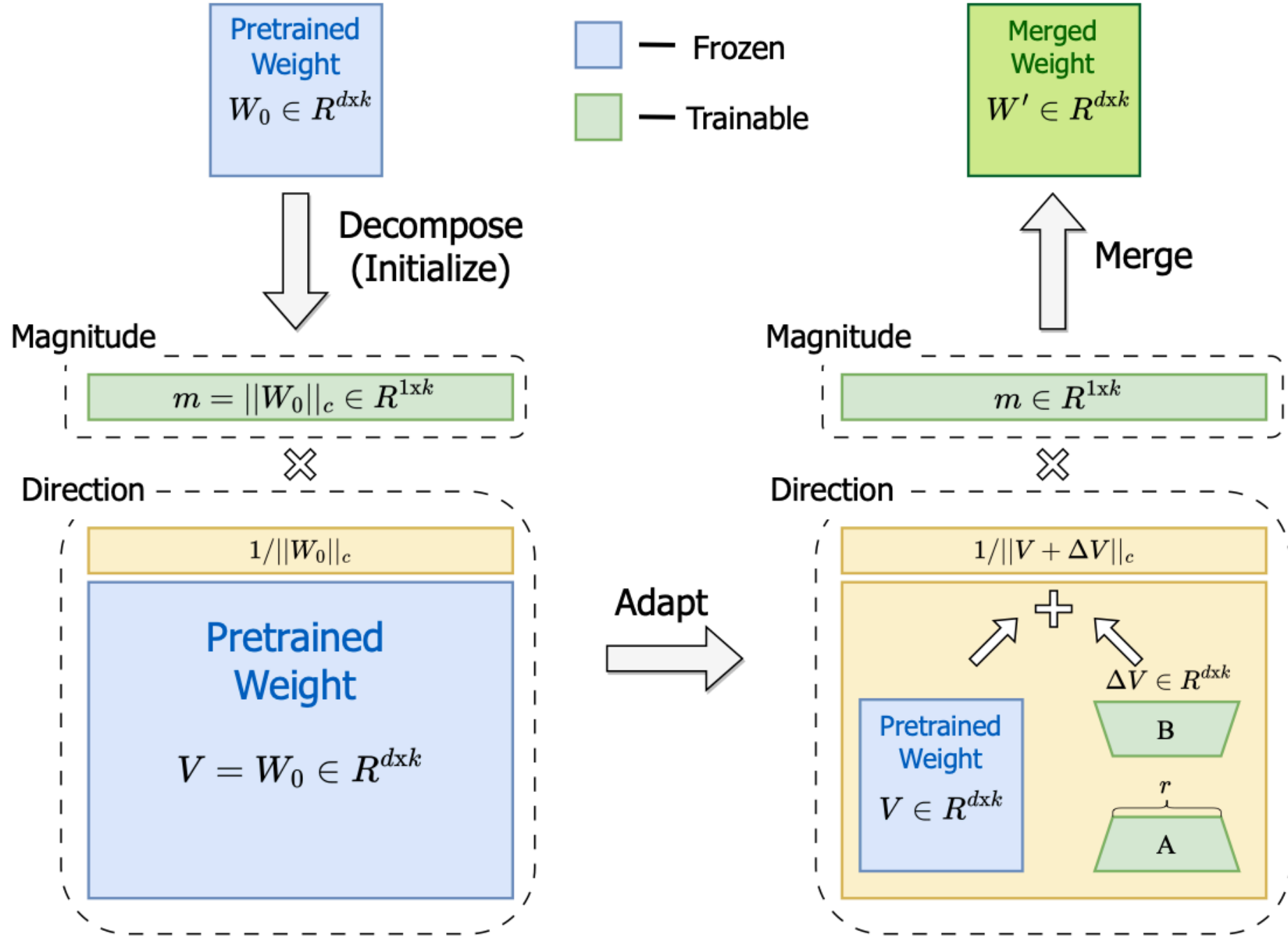
- Learning new thing: large  $\Delta D$  small  $\Delta M$
- Using prior knowledge: large  $\Delta M$  small  $\Delta D$

- LoRA:  $W' = W_0 + \Delta W = W_0 + \underline{BA}$

- Weight decomposition:

$$W = m \frac{V}{\|V\|_c} = \|W\|_c \frac{W}{\|W\|_c}$$

- DoRA:**  $W' = \underline{m} \frac{V + \Delta V}{\|V + \Delta V\|_c} = \underline{m} \frac{W_0 + \underline{BA}}{\|W_0 + \underline{BA}\|_c}$



# PiSSA: SVD + LoRA [Meng+ 2024]

- Poor initialization of LoRA

---

## LoRA

---

Forward  $Y = X(\mathbf{W} + \underline{\Delta W}) = X(\mathbf{W} + \underline{AB})$

---

Initialization  $\underline{A} \sim \mathcal{N}(0, \sigma^2) \in \mathbb{R}^{m \times r}$   
 $\underline{B} = 0 \in \mathbb{R}^{r \times n}$

---

Gradient  $\frac{\partial L}{\partial \underline{A}} = X^\top \left( \frac{\partial L}{\partial Y} \right) \underline{B}^\top \rightarrow \underline{0}$   
 $\frac{\partial L}{\partial \underline{B}} = \underline{A}^\top X^\top \left( \frac{\partial L}{\partial Y} \right) \rightarrow \underline{\text{Random Direction}}$

---

Comparison Fine-tunes noise while freezing  $\mathbf{W}$ .  
Slow convergence and underperformance.  
QLoRA cannot reduce quantization error.

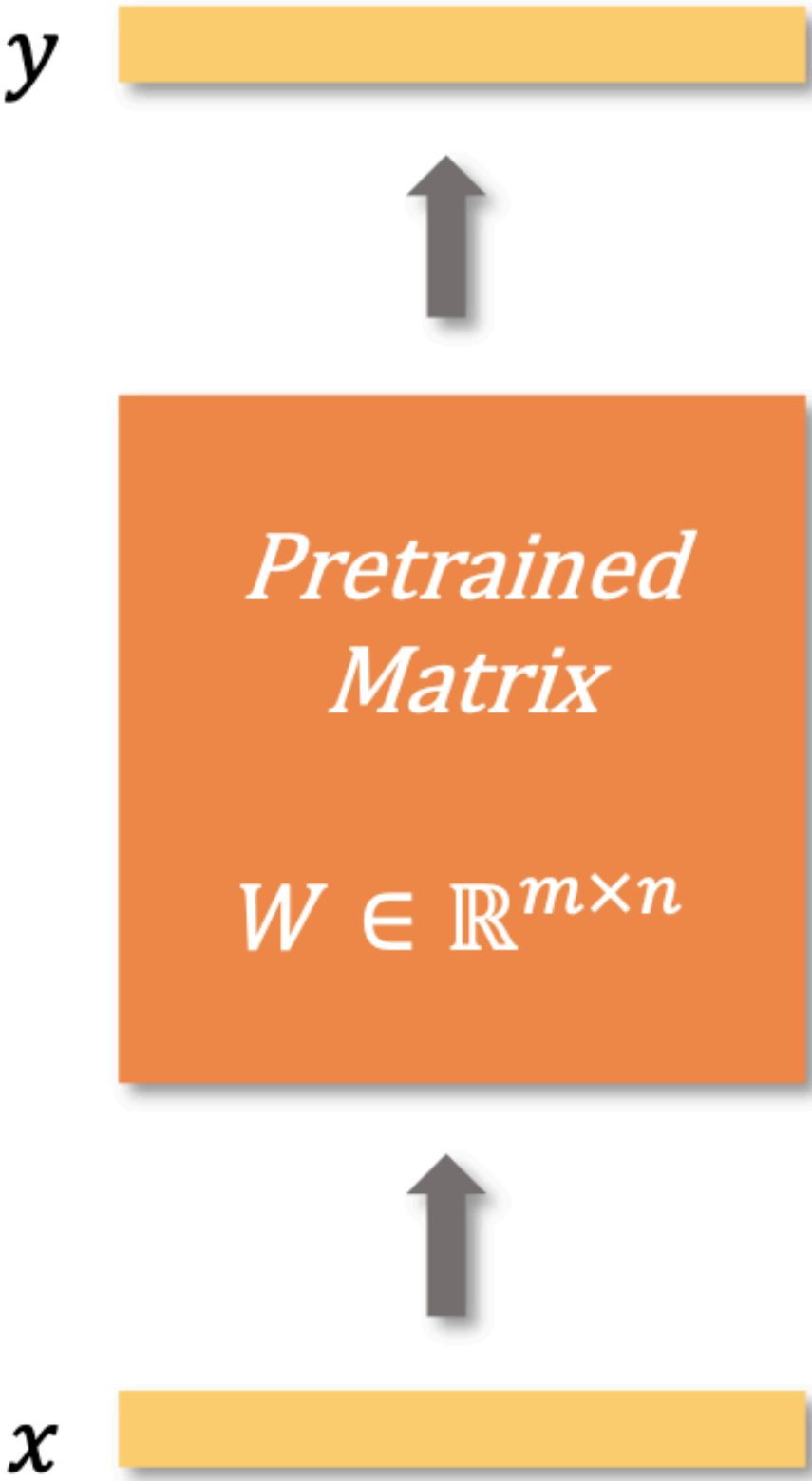
---

# PiSSA: SVD + LoRA [Meng+ 2024]

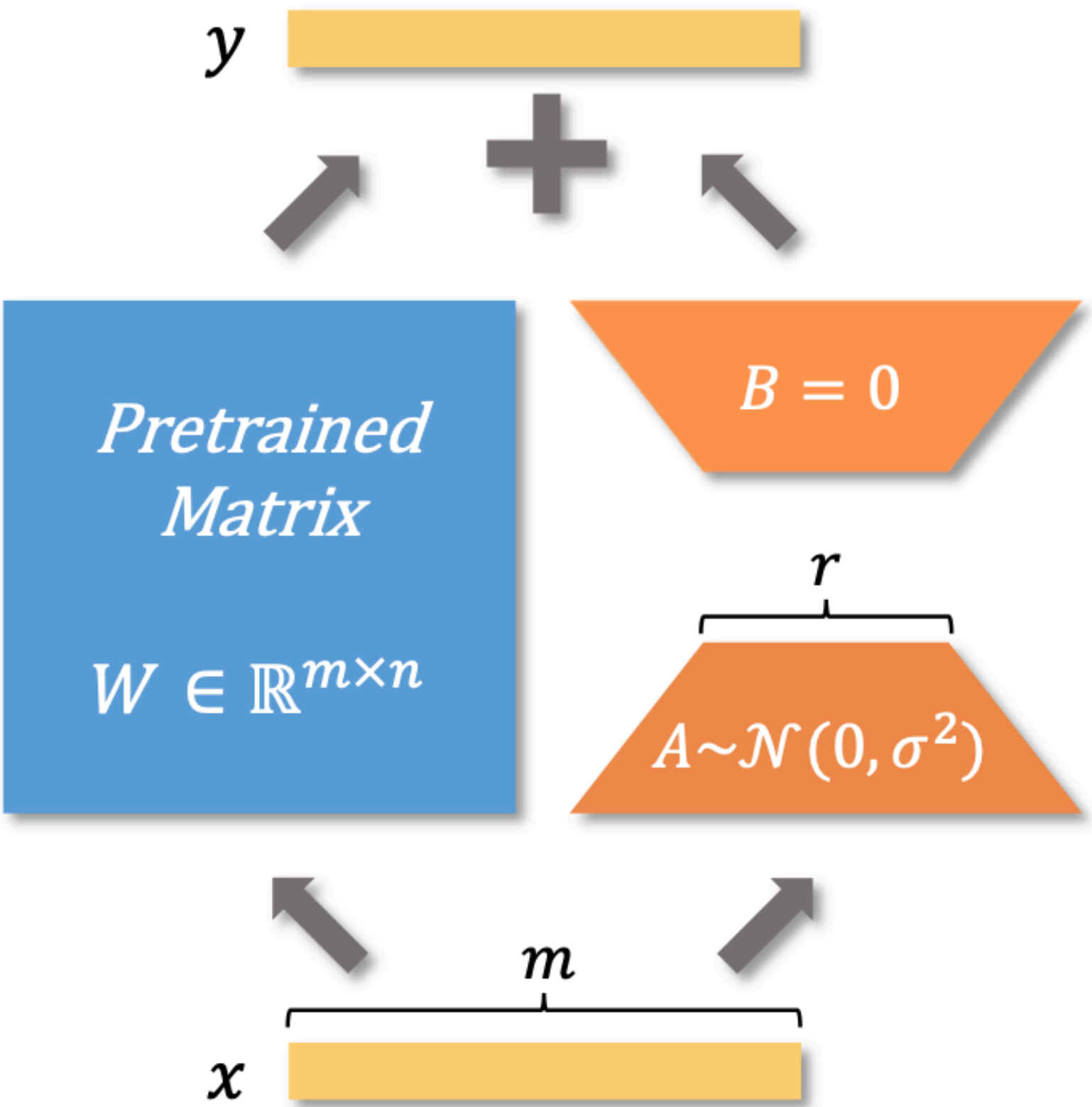
- Better initialization → better performance

	LoRA	PiSSA
Forward	$Y = X(\mathbf{W} + \underline{\Delta W}) = X(\mathbf{W} + \underline{AB})$	$Y = X(\underline{W}^{res} + \mathbf{W}^{pri}) = X(\underline{W}^{res} + \mathbf{AB})$
Initialization	$\underline{A} \sim \mathcal{N}(0, \sigma^2) \in \mathbb{R}^{m \times r}$ $\underline{B} = \mathbf{0} \in \mathbb{R}^{r \times n}$	$\mathbf{A} = U_{[:,\mathbf{r}]} S_{[\mathbf{r},\mathbf{r}]}^{1/2} \in \mathbb{R}^{m \times r}$ $\mathbf{B} = S_{[\mathbf{r},\mathbf{r}]}^{1/2} V_{[:,\mathbf{r}]}^\top \in \mathbb{R}^{r \times n}$ $\underline{W}^{res} = U_{[:,\underline{r}]} S_{[\underline{r},\underline{r}]} V_{[:,\underline{r}]}^\top \in \mathbb{R}^{m \times n}$
Gradient	$\frac{\partial L}{\partial \underline{A}} = X^\top \left( \frac{\partial L}{\partial Y} \right) \underline{B}^\top \rightarrow \underline{\mathbf{0}}$ $\frac{\partial L}{\partial \underline{B}} = \underline{A}^\top X^\top \left( \frac{\partial L}{\partial Y} \right) \rightarrow \underline{\text{Random Direction}}$	$\frac{\partial L}{\partial \mathbf{A}} = X^\top \left( \frac{\partial L}{\partial Y} \right) \mathbf{B}^\top \rightarrow \mathbf{Principal}$ $\frac{\partial L}{\partial \mathbf{B}} = \mathbf{A}^\top X^\top \left( \frac{\partial L}{\partial Y} \right) \rightarrow \mathbf{Principal}$
Comparison	<p>Fine-tunes <u>noise</u> while freezing <math>\mathbf{W}</math>.</p> <p><u>Slow</u> convergence and <u>underperformance</u>.</p> <p>QLoRA <u>cannot</u> reduce quantization error.</p>	<p>Fine-tunes <b>principal</b> parts freezing <math>\underline{W}^{res}</math>.</p> <p><b>Fast</b> convergence and <b>better</b> performance.</p> <p>QPiSSA <b>can</b> reduce quantization error.</p>

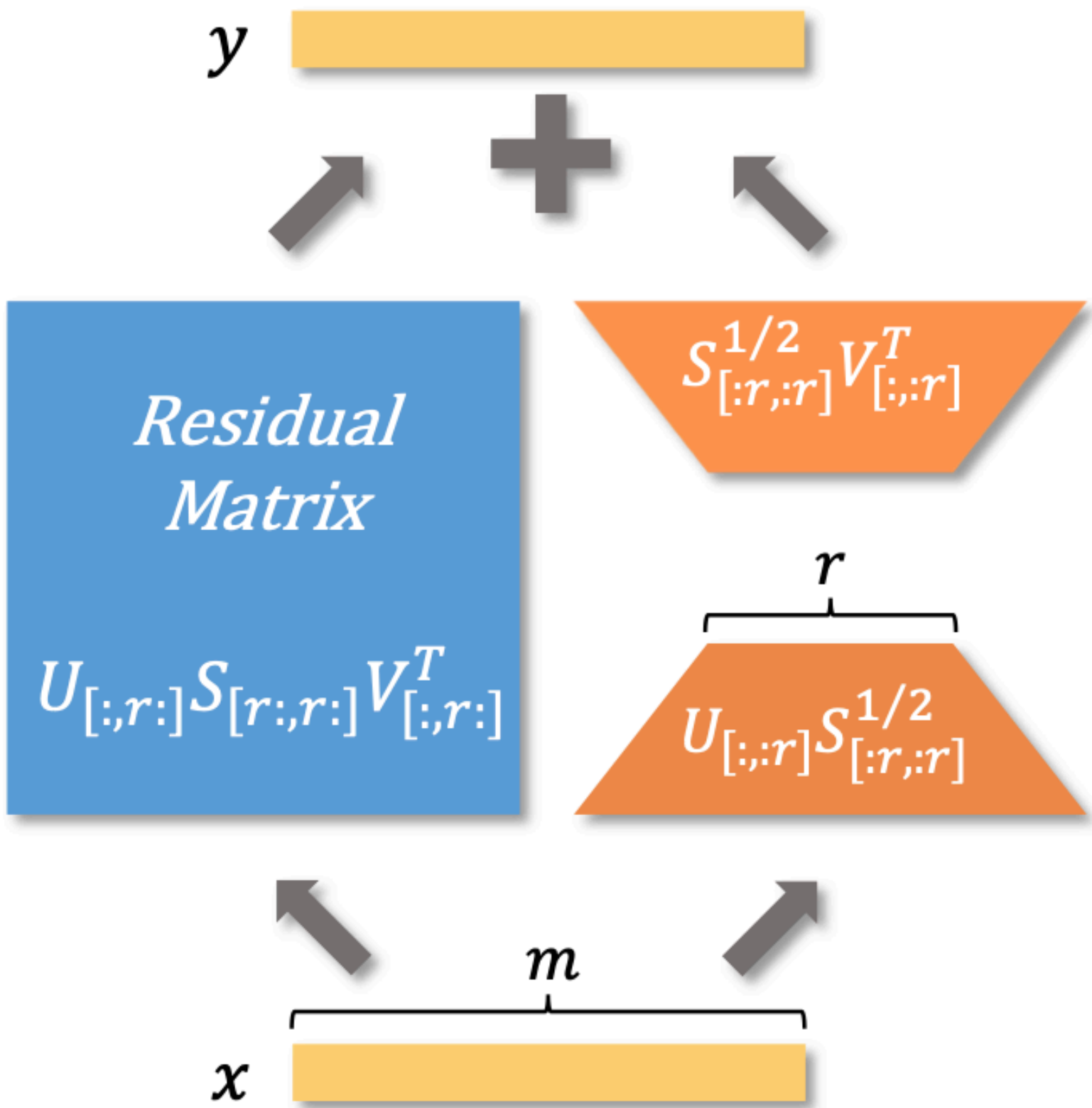
# PiSSA: SVD + LoRA [Meng+ 2024]



(a) Full Fine-tuning



(b) LoRA

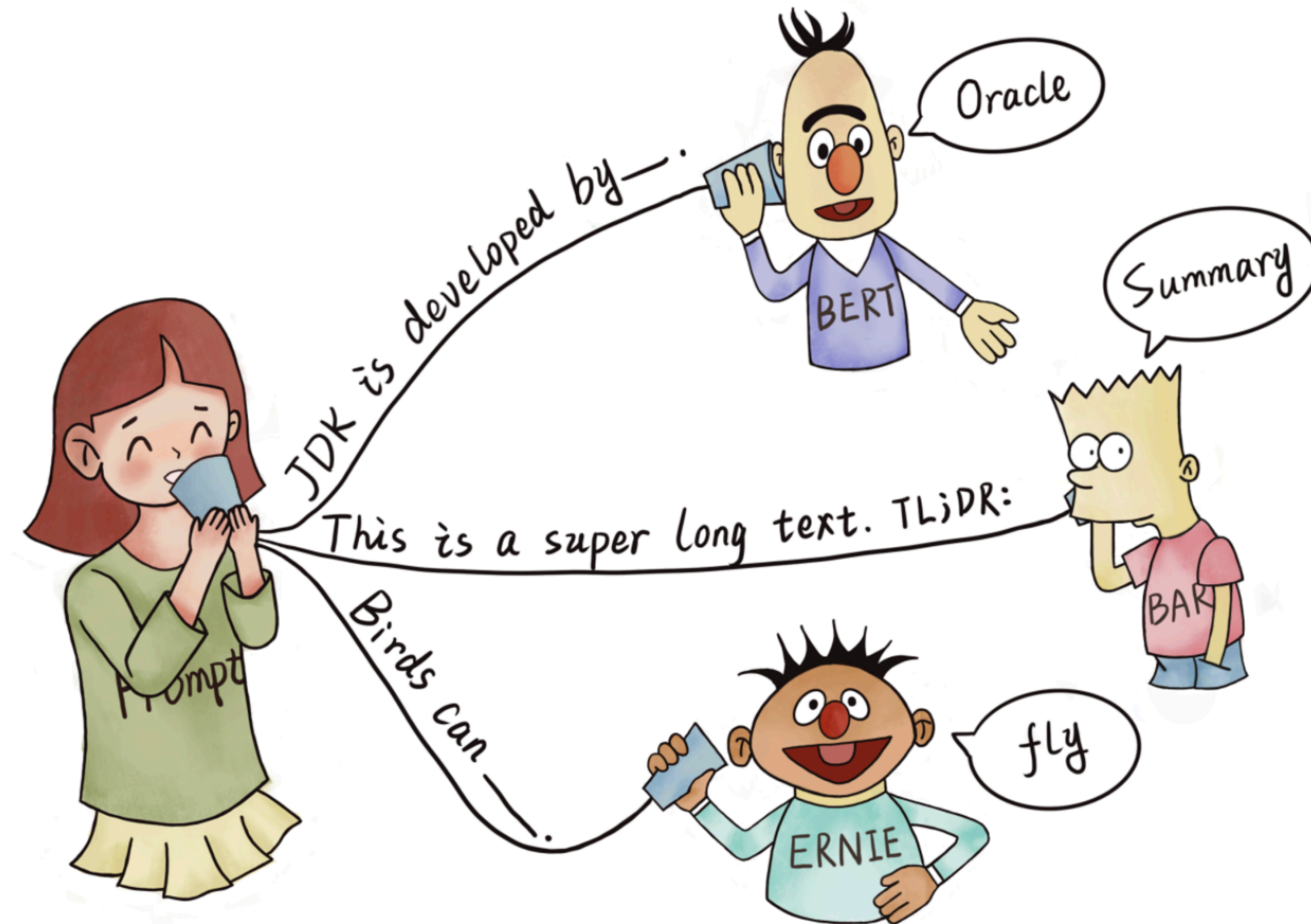


(c) PiSSA

# Prompting

# What is Prompting?

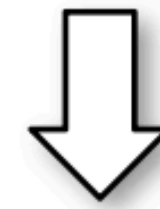
- Encouraging a pre-trained model to make particular predictions by providing a **textual "prompt"** specifying the task to be done



# Prompt Templates

- A template where you fill in with an actual input

**Input:**  $x = \text{"I love this movie"}$



**Template:**  $[x]$  Overall, it was  $[z]$



**Prompting:**  $x' = \text{"I love this movie. Overall it was [z]"}$

# Chat Prompts

- Recently, many models are trained as chatbots
- Usually inputs are specified in OpenAI messages format

```
messages=[  
  {  
    "role": "system",  
    "content": "Please classify movie reviews as 'positive' or 'negative'."  
  },  
  {  
    "role": "user",  
    "content": "This movie is a banger."  
  },  
]
```

- Roles:
  - **“system”**: message provided to the system to influence behavior
  - **“user”**: message input by the user
  - **“assistant”**: message output by the system

# Chat Prompts

- Behind the scenes, messages are converted to token strings

## LLaMa

```
Sys. | [INST]
      | <<SYS>>
      | You are an assistant that ...
      | <</SYS>>
      | [/INST]
User | [INST]This movie is great.[/INST]
Asst.| Positive.
```

## Alpaca

```
Sys. | ### Instruction:
      | You are an assistant that ...
User | ### Instruction:
      | This movie is great.
Asst.| ### Response:
      | Positive.
```

# In-Context Learning [Brown+ 2021]

- GPT-3: Language Models are Few-Shot Learners
- Provide a few examples of the task together with the instruction

**Instruction** | Please classify movie reviews as 'positive' or 'negative'.

## Examples

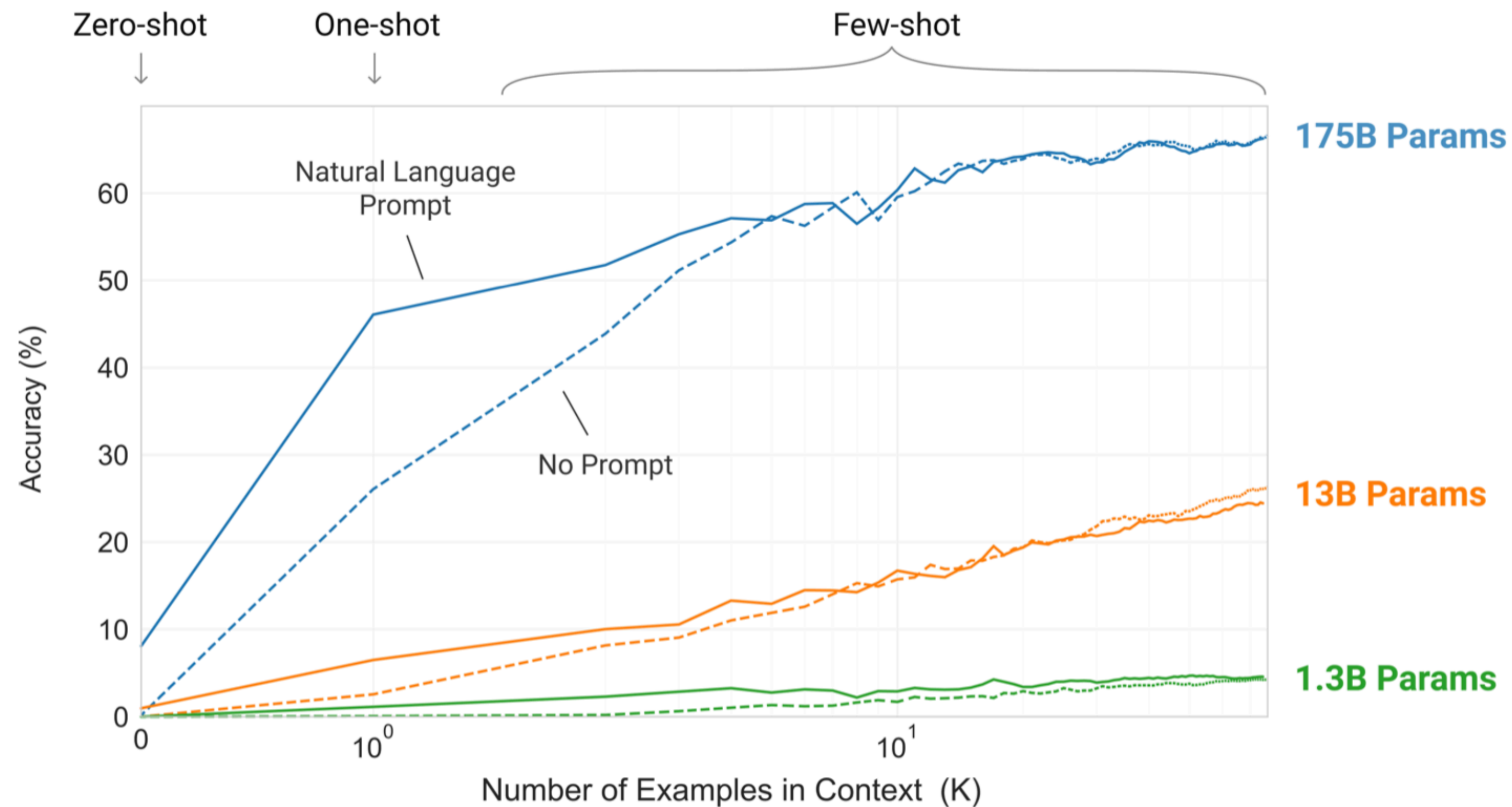
Input: I really don't like this movie.

Output: negative

Input: This movie is great!

Output: positive

# In-Context Learning [Brown+ 2021]



**Figure 1.2: Larger models make increasingly efficient use of in-context information.** We show in-context learning performance on a simple task requiring the model to remove random symbols from a word, both with and without a natural language task description (see Sec. 3.9.2). The steeper “in-context learning curves” for large models demonstrate improved ability to learn a task from contextual information. We see qualitatively similar behavior across a wide range of tasks.

# In-Context Lear

## The three settings we explore for in-context learning

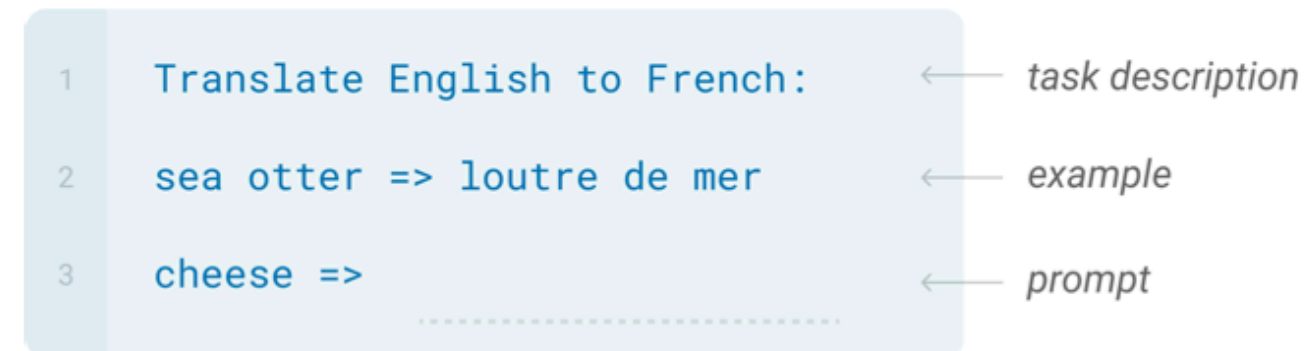
### Zero-shot

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



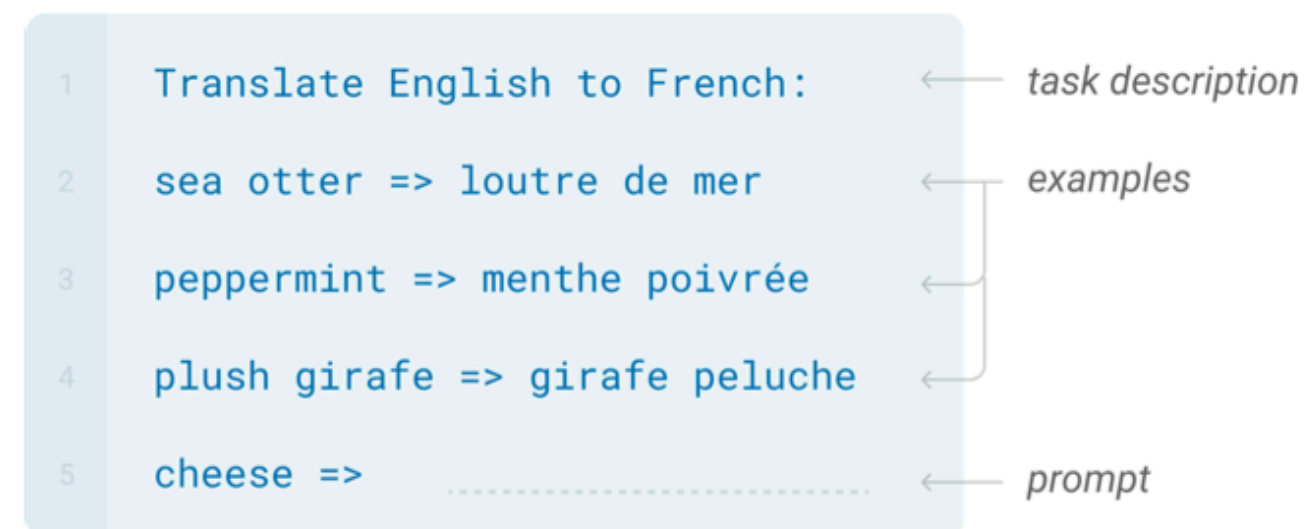
### One-shot

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



### Few-shot

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



## Traditional fine-tuning (not used for GPT-3)

### Fine-tuning

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



# Standard Prompting

- Provide instruction + role + example
- Before the advancement in post-training techniques, standard prompting performance is poor on reasoning benchmarks
- Issue: standard few-shot exemplars only provide **information on the final solution** format, but not the rationale to derive the solution

## 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?

**Exemplar**

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 (CoT) [Wei+ 2022]

## 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?

**Exemplar**

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? **Thought**

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

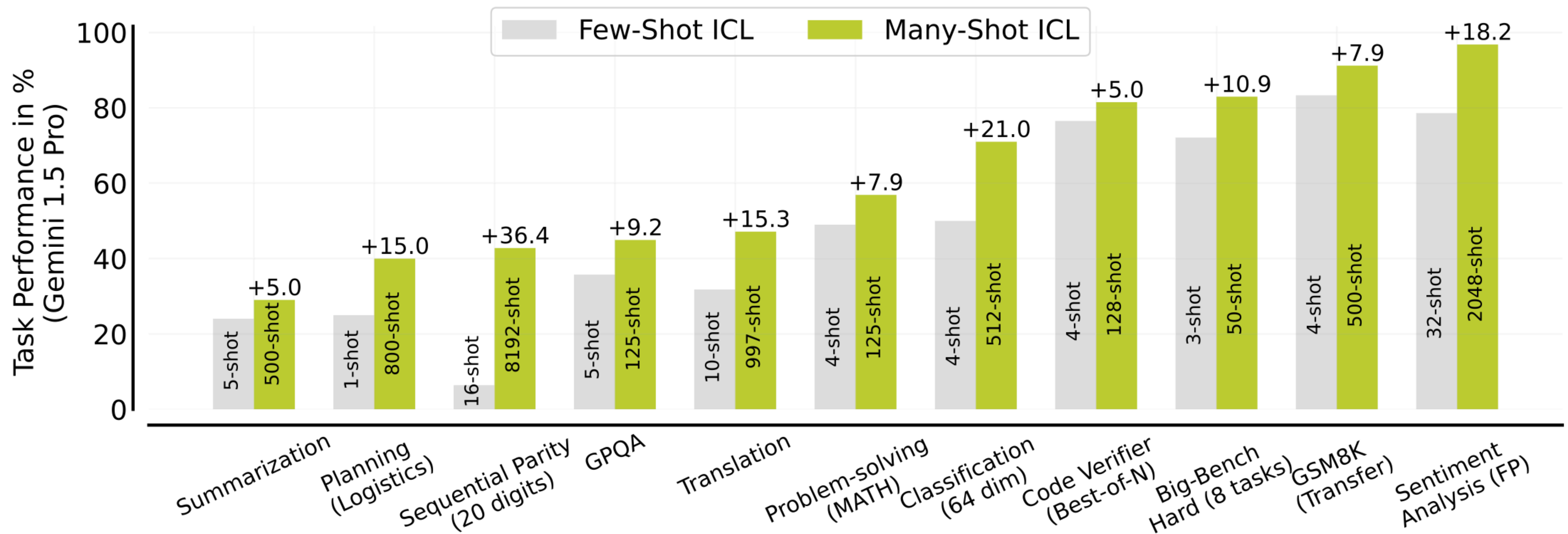
Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

### 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. ✅

# Many-Shot ICL [Agarwal+ 2024]

- Previously, many-shot was bottlenecked by limited context length
  - e.g. GPT-3 (2048 tokens), but now we ~ 1M tokens!

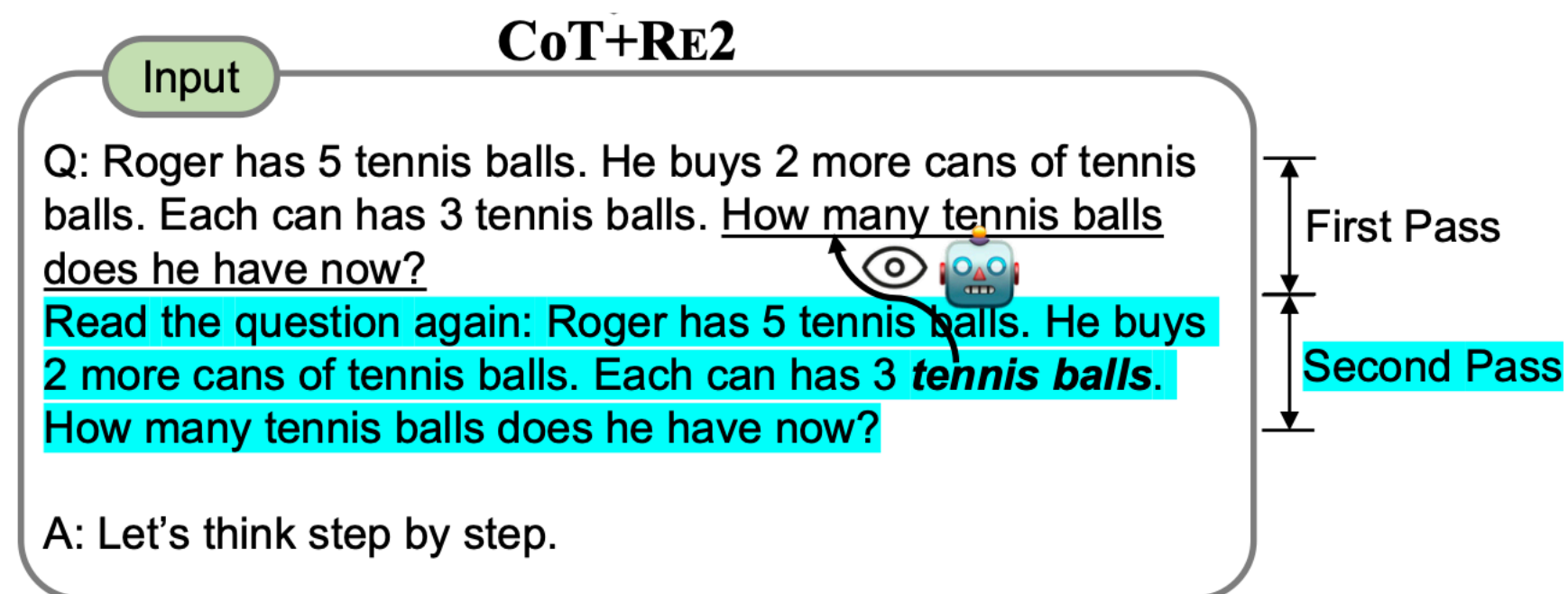
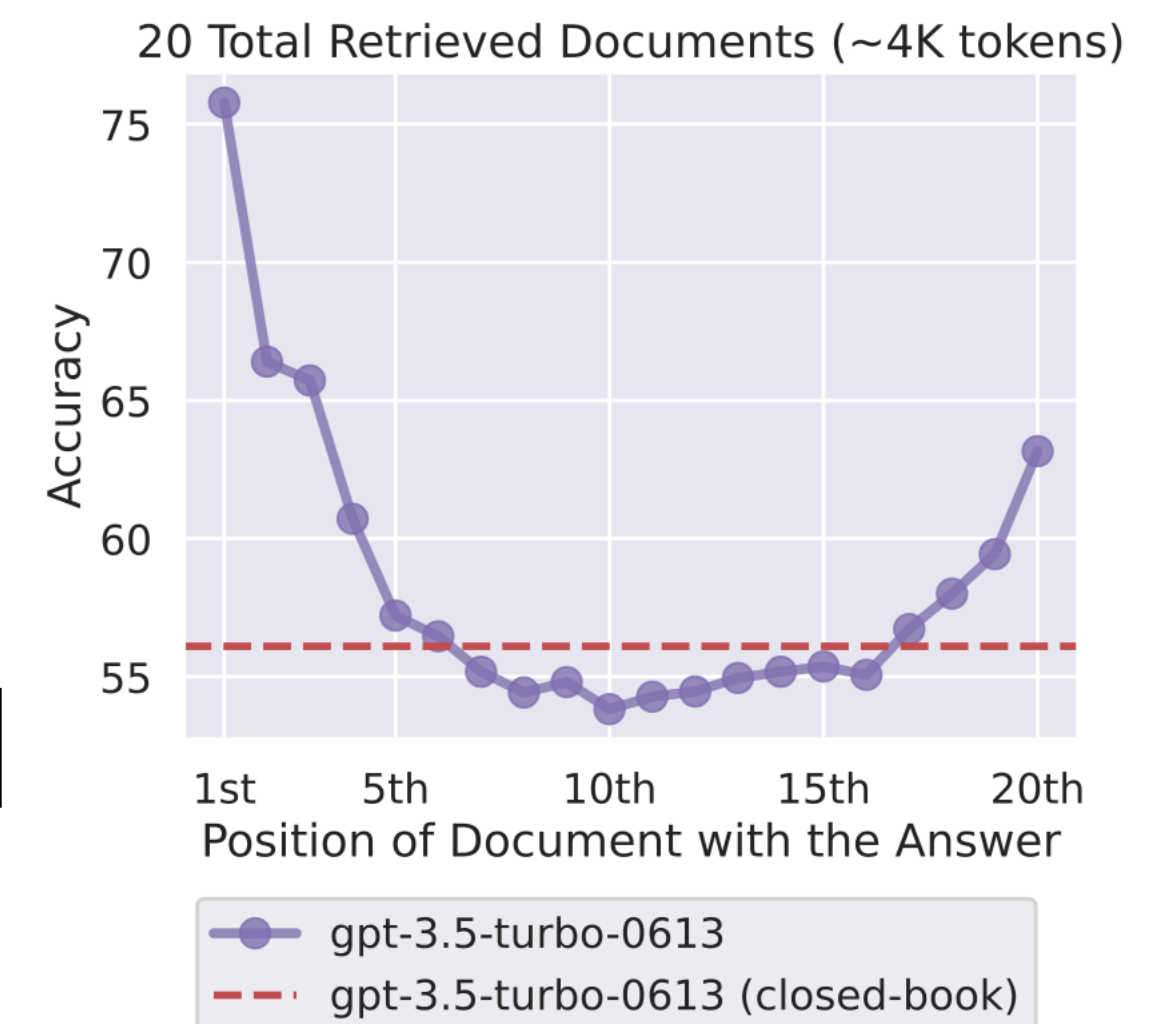
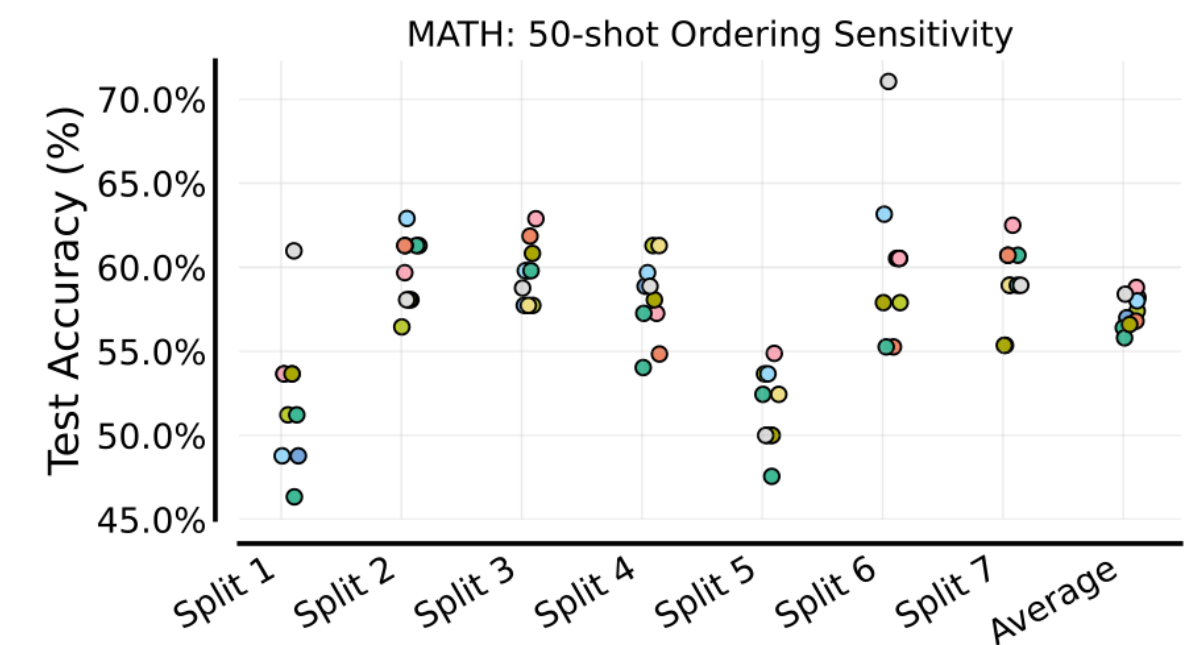


# Many-Shot ICL [Agarwal+ 2024]

- Some important observations
  - Many-shot ICL can overcome pre-training biases (small-shot cannot)
  - Many-shot ICL enables to learn high-dimensional functions
  - Many-shot ICL shows comparable performance to SFT
  - Many-shot ICL wins against repeating the same few-shot

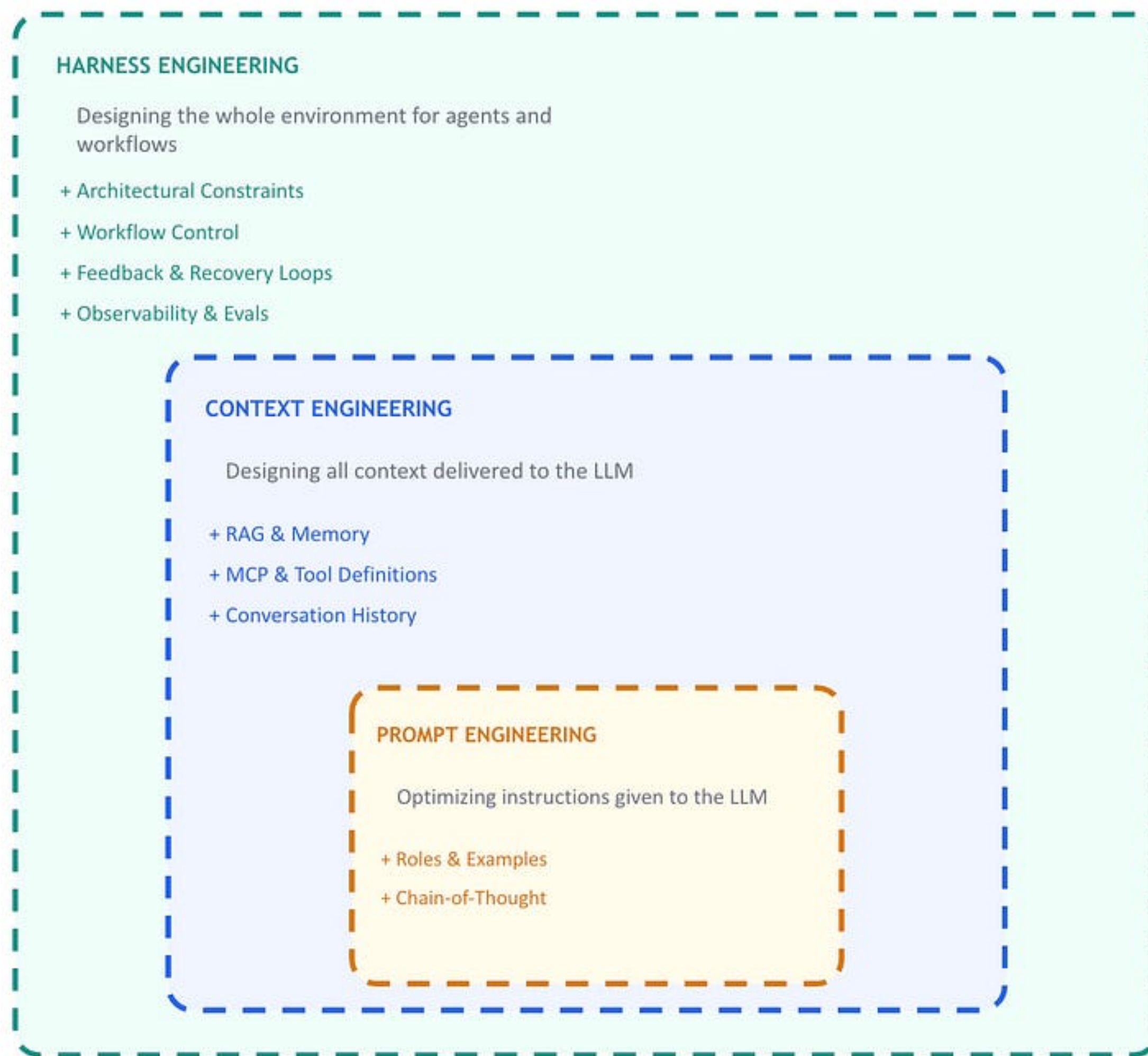
# Subtlety of Prompting

- Even with many-shot, example ordering is important
  - This is long-standing limitation of ICL [Lu+ 2021, Agarwal+ 2024]
- "Lost in the middle" [Liu+ 2023]
  - The model tends to forget middle context
- Prompt repetition helps [Xu+ 2023, Leviathan+ 2025]



Therefore, the order of the tokens in a user's query can affect prediction performance. For example, a query of the form "`<CONTEXT> <QUESTION>`" often performs differently from a query of the form "`<QUESTION> <CONTEXT>`" (see *options-first* vs. *question-first* in Figure 1). We propose to *repeat the prompt*, i.e. transform the input from "`<QUERY>`" to "`<QUERY><QUERY>`". This enables each

# In practice... ChatGPT → LLM + RAG → Claude code



## THREE LAYERS OF AI AGENTS ENGINEERING

# From Prompts to context to Harness

### ● Prompt

What should I ask?  
Optimizes the instruction.

### ● Context

What should the model see?  
Optimizes the input window.

### ● Harness

How should the whole environment be designed?  
Optimizes the system around the model.

AGENT = MODEL + HARNESS

# Summary

- Fine-tuning
  - Train an LLM with your data using SFT
  - Instead of training all parameters, use LoRA or its recent variants
    - Unless you have massive server or intend to make your own LLM
- Prompting
  - The "emergent abilities" of LLM
  - Providing examples helps because LLM learns from the "context"
  - When utilizing AI, building a good context is really important
    - And now people focus on building good AI systems (aka harness)
      - Agent, workflow, etc; see how people use Claude Code recently