innovate achieve lead

Fine-Tuning & Preference Optimization

Lecture 7 | Module 2

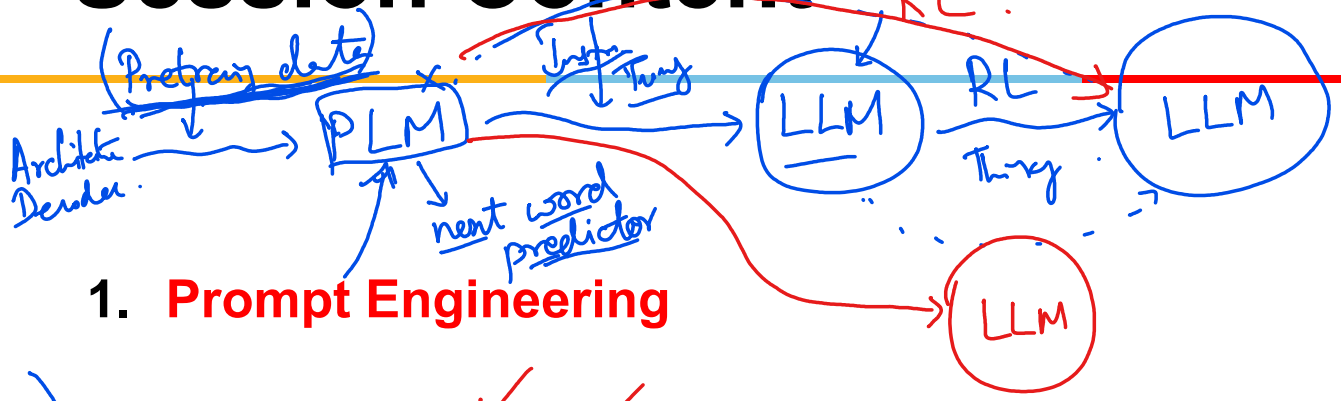
Course: AIMLCZG521

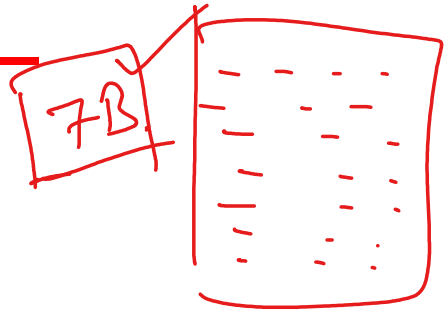
- Bhagath S

BITS Pilani

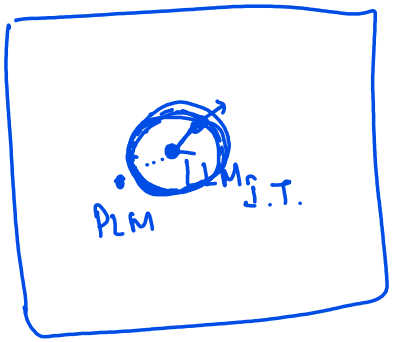
Pilani Campus

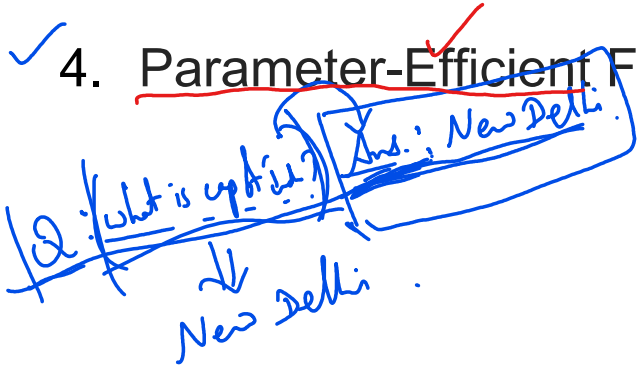
Session Content





1. **Prompt Engineering**
2. Instruction Tuning
3. Reinforcement Learning
4. Parameter-Efficient Fine-Tuning





Prompt Engineering

Prompt engineering is the art and science of crafting effective inputs (prompts) to guide Large Language Models (LLMs) toward desired outputs.

To ensure clarity, consistency, and control when interacting with LLMs, enabling effective communication and reliable output.

Prompt structure:

- **Priming:** This clearly defines the role of the LLM. (Assign a Persona)
- **Style and Tone:** It sets expectations for clear, concise, and jargon-free language with a sought tone.
- **Error Handling:** This guides the LLM to politely decline requests outside its defined scope.
- **Content (static or dynamic or both):** Query, examples, data, etc.
- **Output Formatting:** The format ensures the response can be easily used by an application or displayed in a structured way.

Prompt Engineering

Detailed prompt:

You are a financial analyst specializing in summarizing quarterly earnings reports for technology companies.

Use clear and concise language, avoiding financial jargon. Explain complex financial concepts in a way that is easy for a non-expert to understand. Maintain a formal and objective tone.

If a request is unrelated to summarizing quarterly earnings reports or providing information about them, politely decline and state that it falls outside your area of expertise.

User inquiry: "Can you summarize the main points of Apple's Q3 2024 earnings report and explain its significance?“

Format: Summary: “your response goes here”

Prompt Engineering

Zero-Shot Prompting: This technique involves prompting a language model to generate responses without providing any specific training examples. The model relies solely on its pre-existing knowledge to produce relevant output.

Example:

"Translate the following English text to French: 'Hello, how are you?'"

Few-Shot Prompting: Provide a few examples (input/output pairs) in the prompt to guide the LLM.

Example:

```
"'
English: Sea otter \n French: Loutre de mer

English: Peppermint \n French: Menthe poivrée

English: Cheese \n French: Fromage

English: Thank you \n French:
"'
```

Chain-of-Thought (CoT) Prompting

Encourage the LLM to "think step by step" by providing examples where the reasoning process is explicitly laid out. This is especially useful for tasks requiring arithmetic, commonsense, or symbolic reasoning.

Example for a math problem:

‘‘‘

Question:
Ram has 5 tennis balls. He buys 2 more cans of tennis balls. Each can have 3 tennis balls.
How many tennis balls does he have now?
Answer:
Roger started with 5 balls.
He bought 2 cans, and each can have 3 balls, so that's 2 * 3 = 6 more balls.
Therefore, he now has 5 + 6 = 11 balls.
So, the answer is 11.
Question: <Your complex question here>
Answer:

’’’

Tree-of-Thought (ToT) Prompting

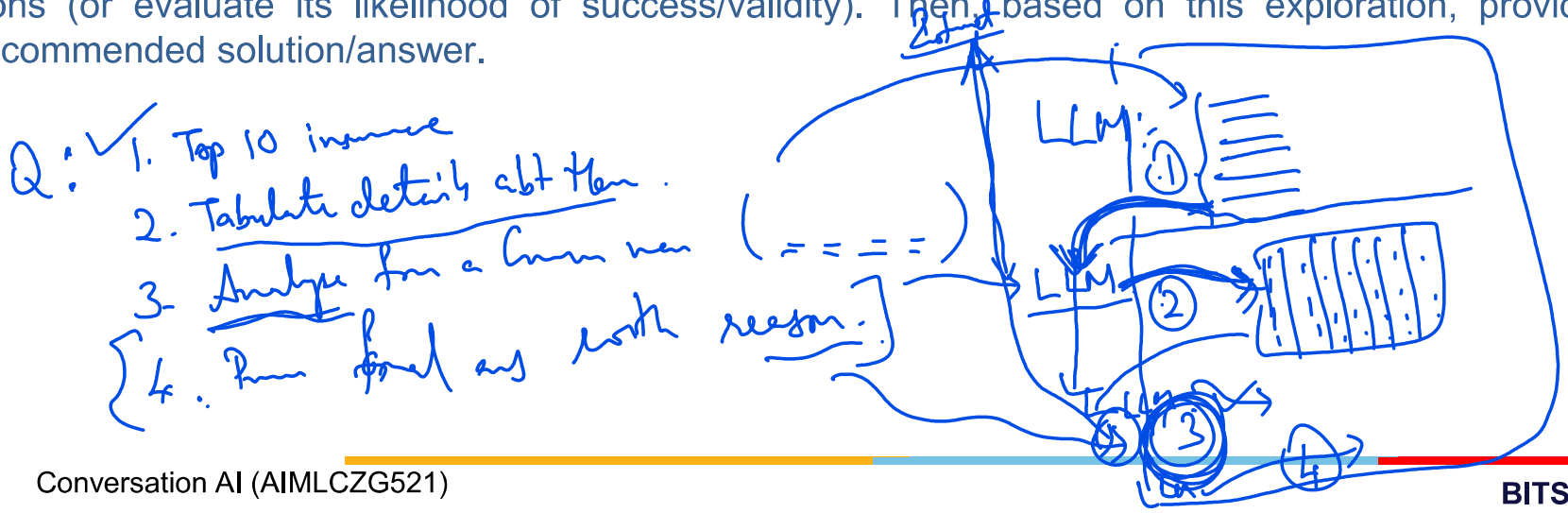
LLM explores multiple reasoning paths (thoughts) as a tree. It self-evaluates intermediate thoughts and uses search algorithms (like BFS or DFS) to decide which path to explore further, enabling more deliberate problem-solving.

Example:

‘‘‘

I'm trying to solve [complex problem/question]. Before giving me a final answer, please explore at least three different potential approaches or lines of reasoning. For each approach, briefly explain it, analyze its pros and cons (or evaluate its likelihood of success/validity). Then, based on this exploration, provide your final recommended solution/answer.

’’’



Prompt Engineering

APE (Automatic Prompt Engineer): Using an LLM to automatically discover and optimize prompts for specific tasks (Executed by Target LLM)

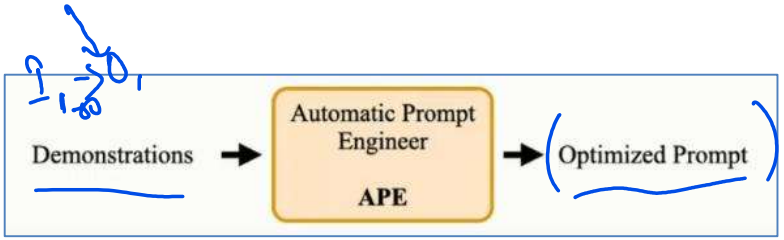
General Guidelines for Prompting for APE:

1. Clearly Define the APE's Meta-Role
2. Describe the Target LLM's Task in Detail
3. Provide High-Quality Examples for the Target Task
4. Specify Qualities of an Effective Target Prompt
5. Define the APE LLM's Deliverables
6. Encourage Iteration and Exploration
7. Output Format for APE LLM

Step-1: Instruction Generation
- A language model is given a task and a few examples (input-output pairs).
- It generates multiple candidate prompts or instructions.

Step-2: Prompt Evaluation
- These prompts are tested by running them through a target model.
- The outputs are scored based on performance metrics (e.g., accuracy, relevance, coherence).

Step-3: Selection
- The best-performing prompt is selected for use.

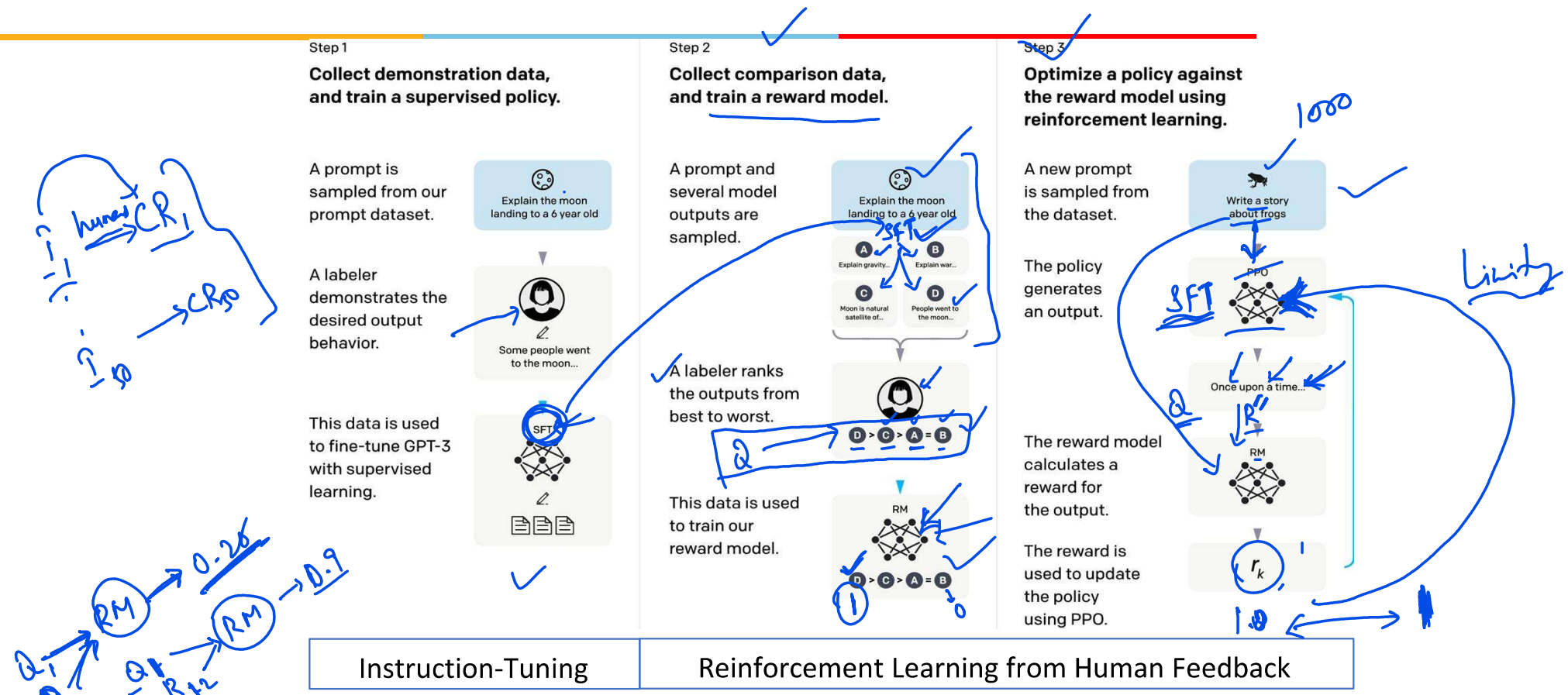


Large Language Models Are Human-Level Prompt Engineers - https://arxiv.org/pdf/2211.01910

Session Content

1. Prompt Engineering
2. **Instruction Tuning**
3. Reinforcement Learning
4. Parameter-Efficient Fine-Tuning

Introduction

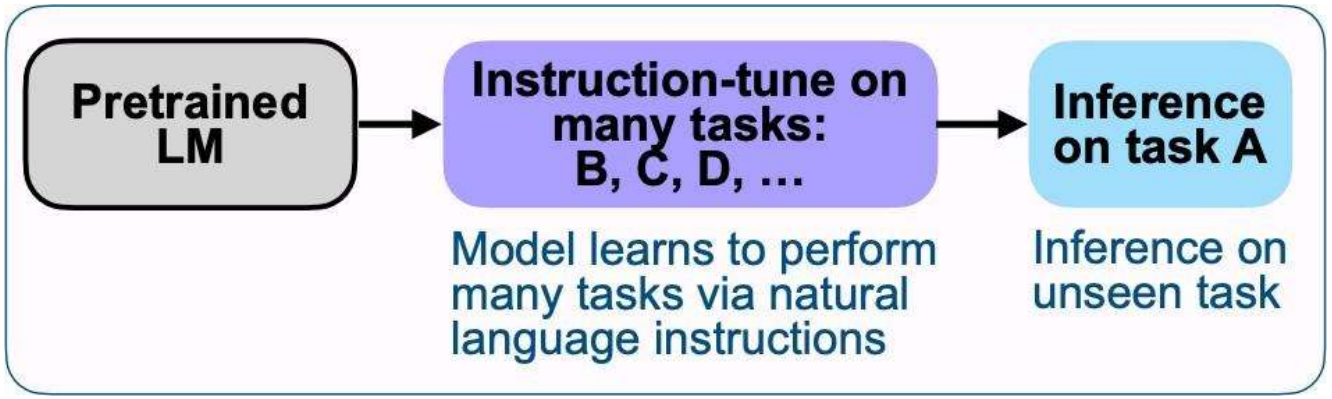


Instruction-Tuning	Reinforcement Learning from Human Feedback

Training language models to follow instructions with human feedback. Ouyang et al. 2022.

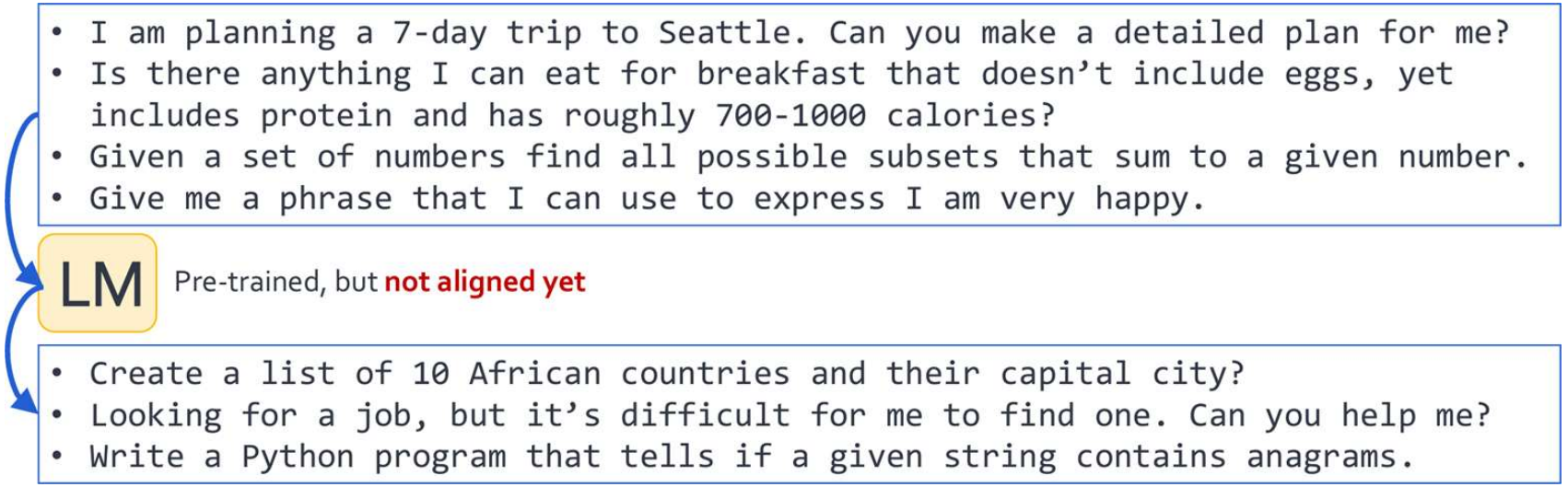
Instruction Tuning

- Fine-tuning on many tasks! Teach language models to follow different natural language instructions, so that it could perform better on downstream tasks and generalize to unseen tasks!
- Fine-tuning -> Instruction Pre-training



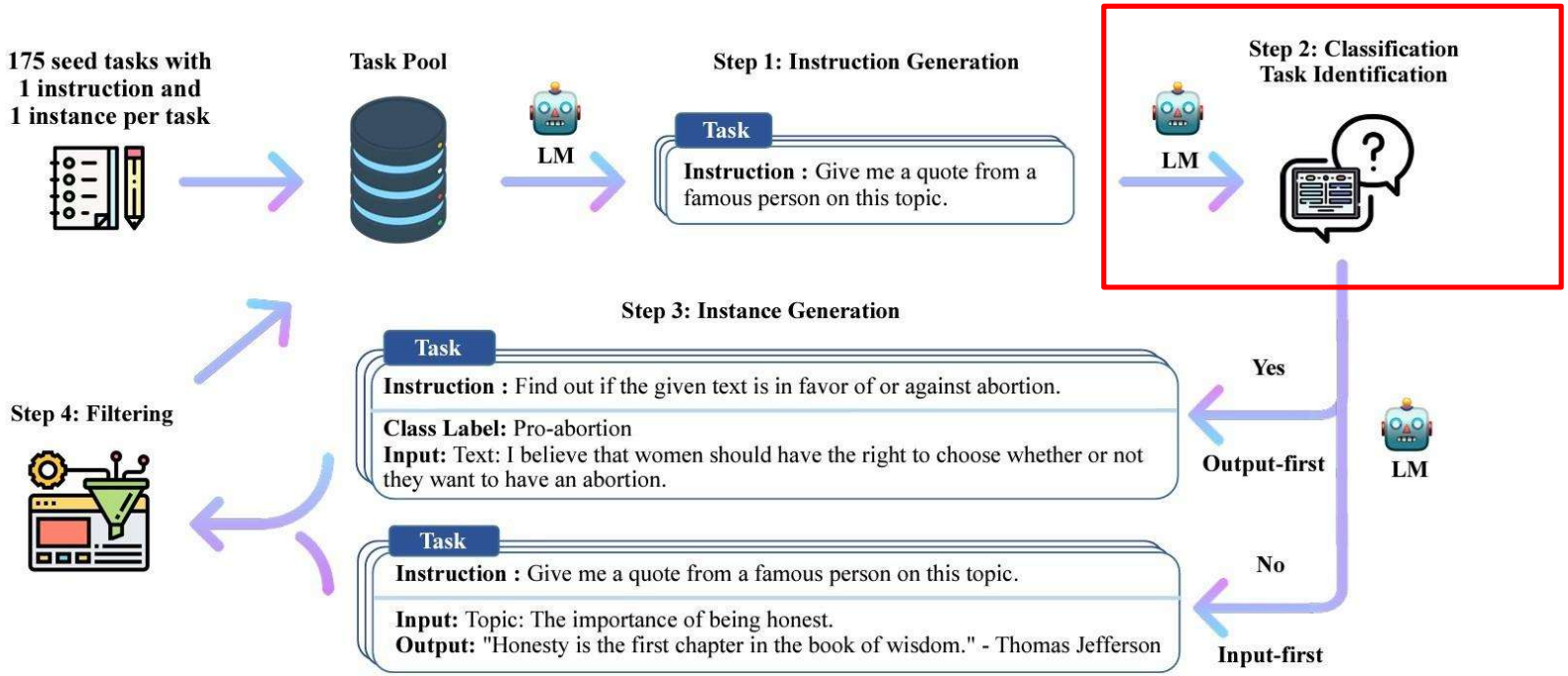
Self-Instruct: Aligning Language Models with Self- Generated Instructions (Wang et. al, 2022)

- Human written seed tasks to bootstrap off-the-shelf language models (GPT-3)



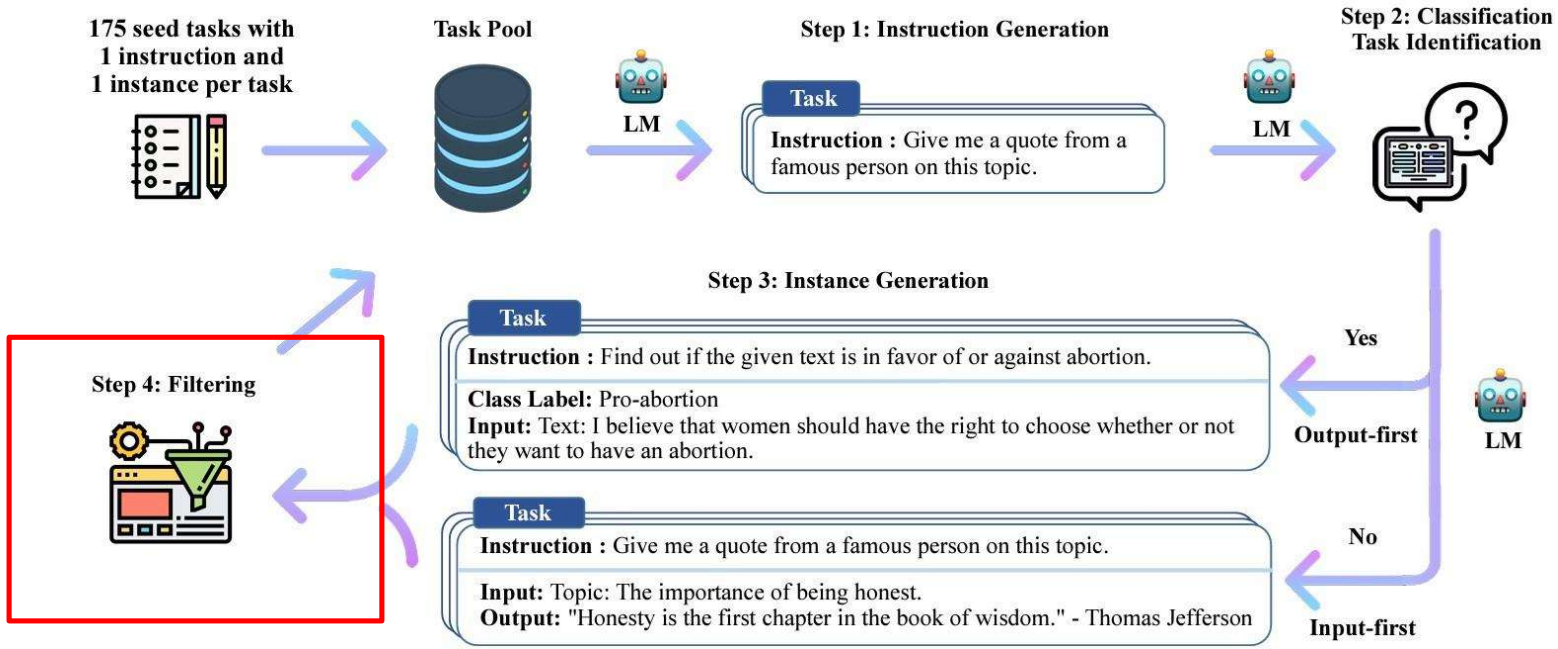
Self-Instruct Framework

- Classify whether the generated instruction is a classification task
- Output-first: avoid bias towards one class label



Self-Instruct Framework

- Filter out instructions similar with existing ones
- Add newly generated tasks into the task pool for next iteration



Experiment Results

- Use a GPT-3 (“davinci”) model to generate new instruction tasks, and fine-tune the GPT-3 model itself
- 175 seed tasks -> 52K instructions and 82K instances

statistic	
# of instructions	52,445
- # of classification instructions	11,584
- # of non-classification instructions	40,861
# of instances	82,439
- # of instances with empty input	35,878
ave. instruction length (in words)	15.9
ave. non-empty input length (in words)	12.7
ave. output length (in words)	18.9

Self-Instruct: Aligning Language Models with Self- Generated Instructions (Wang et. al, 2022)

- Human-written instruction data can be very expensive!
- Can we reduce the human annotations?
- Idea: bootstrap from off-the-shelf LMs

Comparison of Using Different Instruction Tuning Datasets

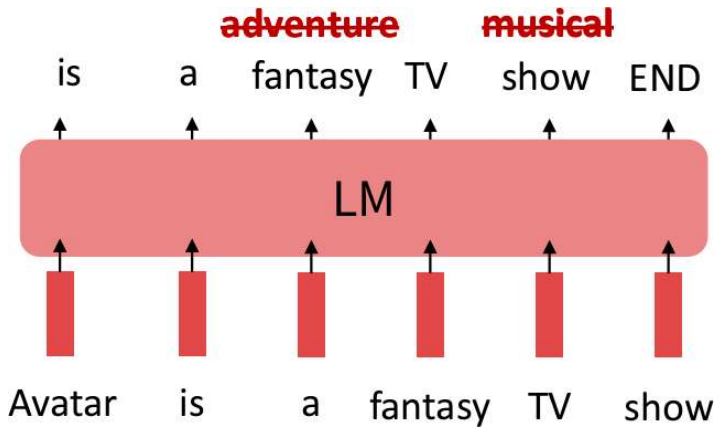
- There is not a single best instruction tuning dataset across all tasks
- Combining datasets results in the best overall performance

	MMLU (factuality)	GSM (reasoning)	BBH (reasoning)	TydiQA (multilinguality)	Codex-Eval (coding)	AlpacaEval (open-ended)	Average
	EM (0-shot)	EM (8-shot, CoT)	EM (3-shot, CoT)	F1 (1-shot, GP)	P@10 (0-shot)	Win % vs Davinci-003	
Vanilla LLaMa 13B	42.3	14.5	39.3	43.2	28.6	-	-
+SuperNI	49.7	4.0	4.5	**50.2**	12.9	4.2	20.9
+CoT	44.2	40.0	41.9	47.8	23.7	6.0	33.9
+Flan V2	**50.6**	20.0	40.8	47.2	16.8	3.2	29.8
+Dolly	45.6	18.0	28.4	46.5	31.0	13.7	30.5
+Open Assistant 1	43.3	15.0	39.6	33.4	31.9	58.1	36.9
+Self-instruct	30.4	11.0	30.7	41.3	12.5	5.0	21.8
+Unnatural Instructions	46.4	8.0	33.7	40.9	23.9	8.4	26.9
+Alpaca	45.0	9.5	36.6	31.1	29.9	21.9	29.0
+Code-Alpaca	42.5	13.5	35.6	38.9	34.2	15.8	30.1
+GPT4-Alpaca	46.9	16.5	38.8	23.5	**36.6**	63.1	37.6
+Baize	43.7	10.0	38.7	33.6	28.7	21.9	29.4
+ShareGPT	49.3	27.0	40.4	30.5	34.1	**70.5**	42.0
+Human data mix.	50.2	38.5	39.6	47.0	25.0	35.0	39.2
+Human+GPT data mix.	49.3	**40.5**	**43.3**	45.6	35.9	56.5	**45.2**

How Far Can Camels Go? Exploring the State of Instruction Tuning on Open Resources, Yizhong Wang et. al., 2023

Limitations of Instruction-Tuning

- Human-written pairs are very expensive
- Mismatch between LM objectives and human preferences
  - factual error vs. imprecise adjectives



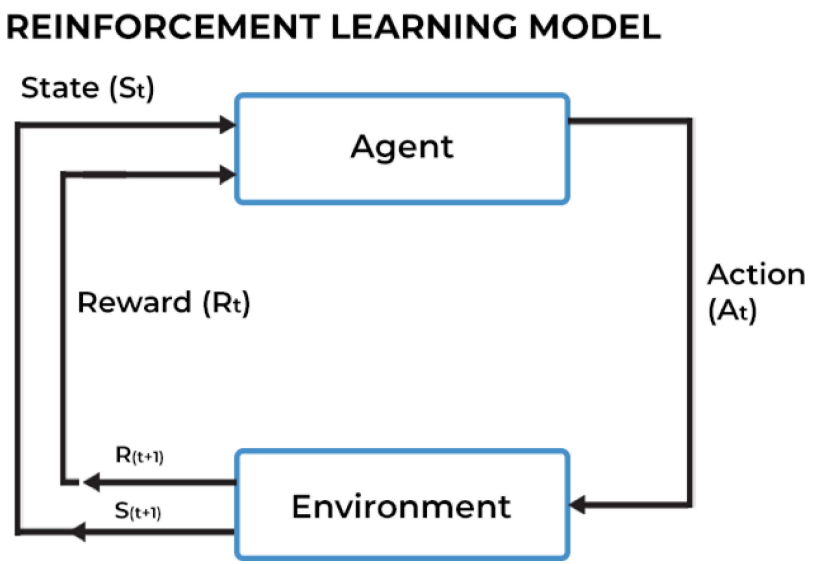
Session Content

1. Prompt Engineering
2. Instruction Tuning
3. **Reinforcement Learning**
4. Parameter-Efficient Fine-Tuning

Common Objectives of Learning from Human Feedback

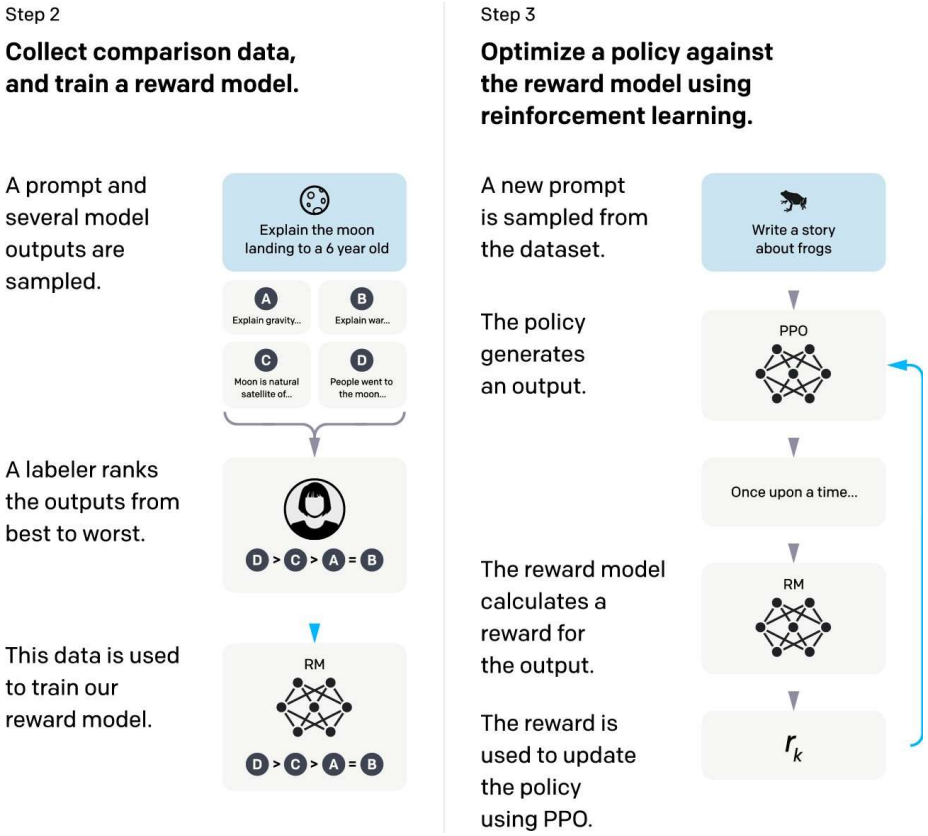
- Align model output with our values
- Trustworthy and robust on factualness
- Fairness on social values
- Explainable with logical rationales

Reinforcement Learning Model



- An agent has a policy function, which can act A_t according to the current state S_t.
- As a result of the action, the agent receives a reward Rt from the environment and transit to the next state St+1.

InstructGPT: Training language models to follow instructions with human feedback. (Ouyang et. al, 2022)



- Agent: language model
- Action: predict the next token
- Policy π_θ : the output distribution of the next token
- Reward: r_ϕ a reward model trained by human evaluations on model responses, so no more human-in-the-loop is needed

Reward Model Training

- Prompt supervised fine-tuned language model with to produce pairs of answers

$$(y_1, y_2) \sim \pi^{\text{SFT}}(y \mid x)$$

- Human annotators decide which one wins / is preferred

$$y_w \succ y_l \mid x$$

- A reward model is trained to score y_w higher than y_l

$$\mathcal{L}_R(r_\phi, \mathcal{D}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))\right]$$

- A reward model is often initialized from π^{SFT} with a linear layer to produce a scalar reward value

Fine-Tuning with RL: PPO

- Optimize the language model π_θ with feedback from the reward model r_ϕ

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(y|x)} \left[r_\phi(x, y) \right] - \beta \mathbb{D}_{\mathrm{KL}} \left[\pi_\theta(y \mid x) \,||\, \pi_{\mathrm{ref}}(y \mid x) \right]$$

prefer responses with high rewards

control the deviation from the reference policy, the π^{SFT} model

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.

Fine-Tuning with RL: PPO

- Optimize the language model π_θ with feedback from the reward model r_ϕ

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(y|x)} \left[r_\phi(x, y) \right] - \beta \mathbb{D}_{\mathrm{KL}} \left[\pi_\theta(y \mid x) \,||\, \pi_{\mathrm{ref}}(y \mid x) \right]$$

prefer responses with high rewards

control the deviation from the reference policy, the π^{SFT} model

- prevent mode-collapse to single high reward answers
- prevent the model deviating too far from the distribution where the reward model is accurate

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.

Fine-Tuning with RL: PPO

- Optimize the language model π_θ with feedback from the reward model r_ϕ

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(y|x)} \left[r_\phi(x, y) \right] - \beta \mathbb{D}_{\mathrm{KL}} \left[\pi_\theta(y \mid x) \,||\, \pi_{\mathrm{ref}}(y \mid x) \right]$$

- sample y from the current policy
- prefer responses with high rewards
- control the deviation from the reference policy, the π^{SFT} model

- prevent mode-collapse to single high reward answers
- prevent the model deviating too far from the distribution where the reward model is accurate

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.

Fine-Tuning with RL: PPO-ptx

- Training objective

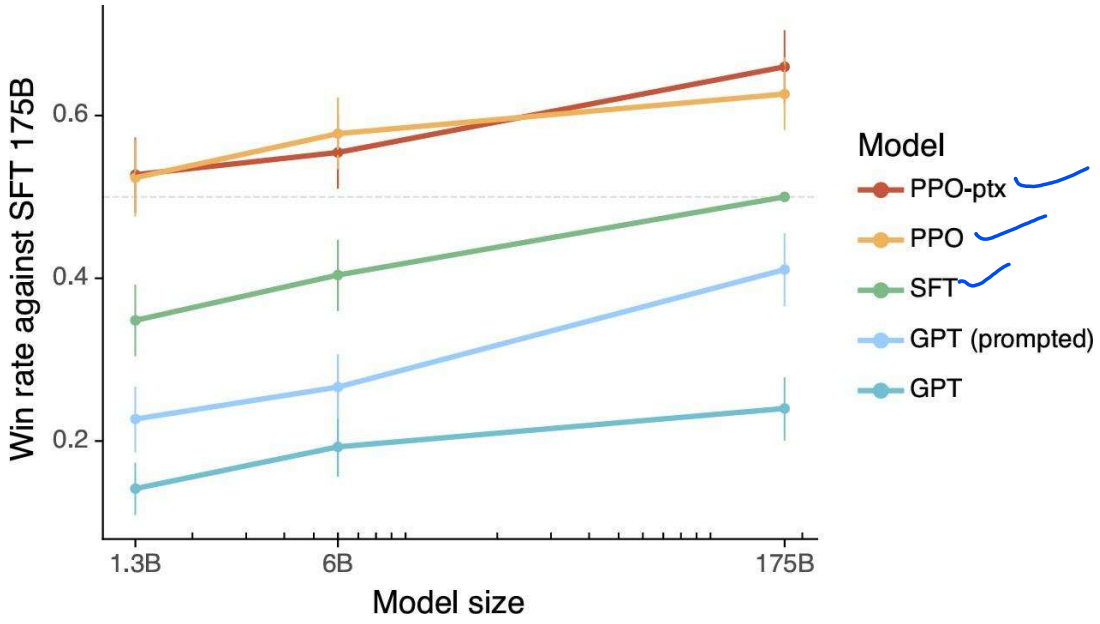
$$E_{(x,y)\sim D_{\pi_\phi^{\mathrm{RL}}}}\left[r_\theta(x,y) - \beta \log\left(\pi_\theta\ (y \mid x)/\pi_{\mathrm{ref}}(y \mid x)\right)\right] +$$
$$\gamma E_{x\sim D_{\mathrm{pretrain}}}\left[\log(\pi_\theta\ (x))\right]$$

- Add pre-training gradients to fix the performance regressions on public NLP tasks
- For PPO models, γ is set to 0

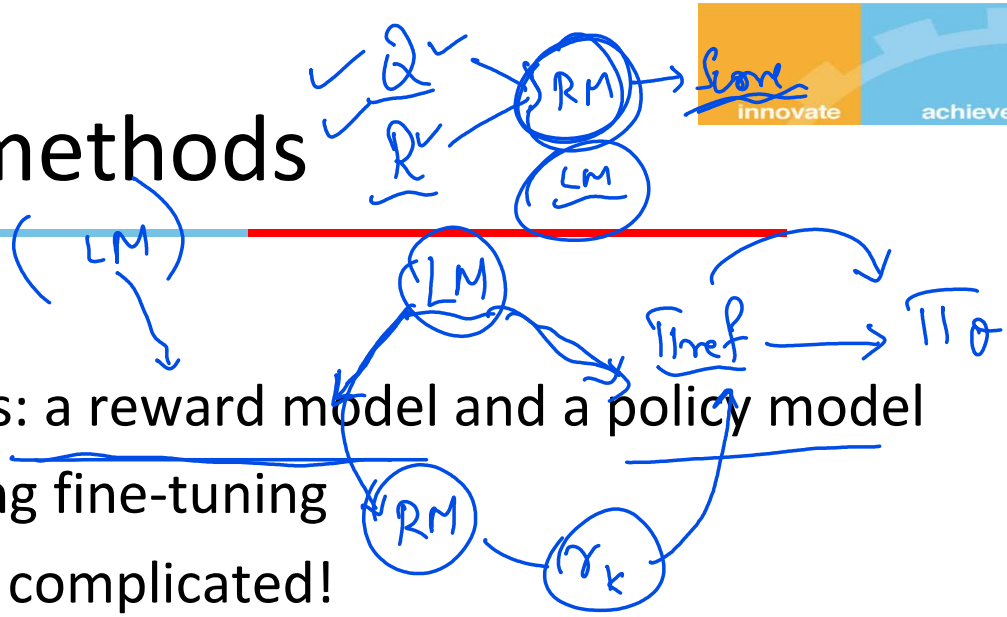
J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.

Comparison with Baselines

- RLHF models are more preferred by human labelers

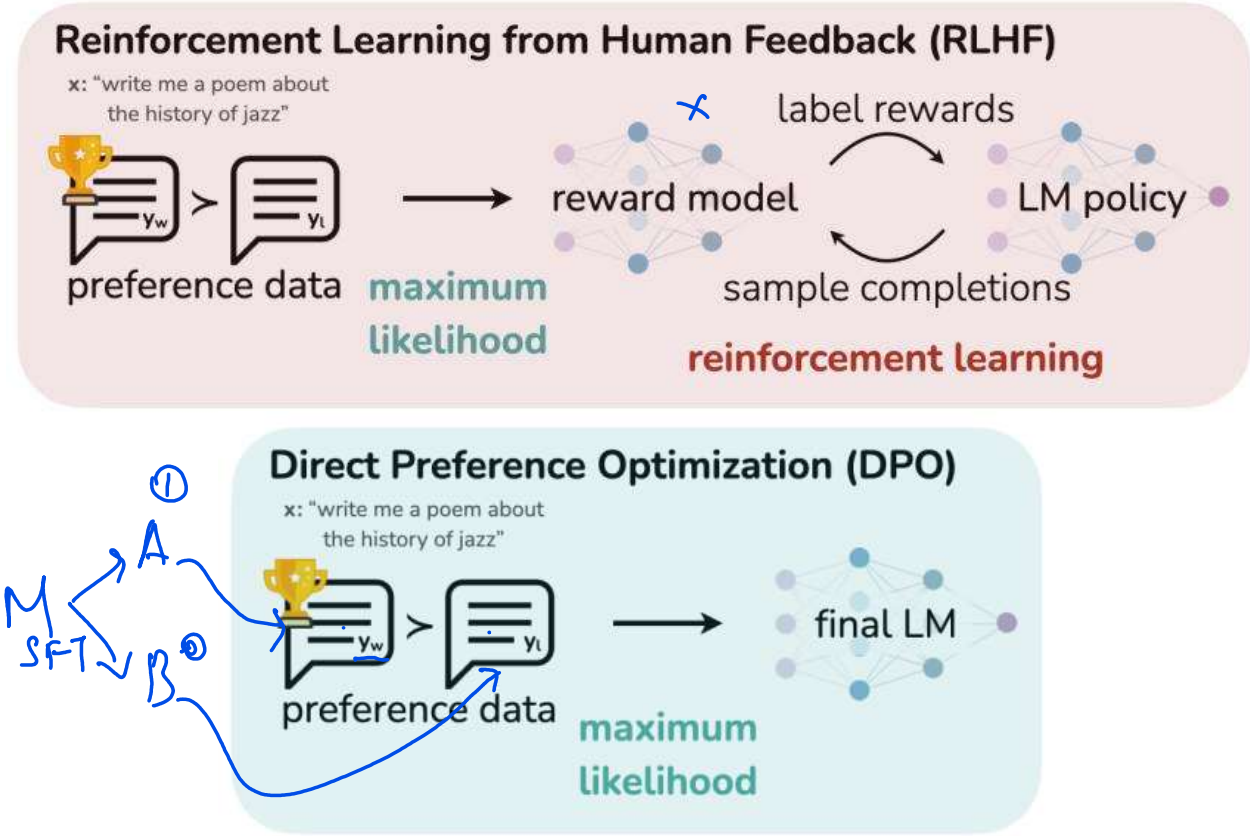


Limitation of PPO methods



- Need to train multiple models: a reward model and a policy model
- Need sampling from LM during fine-tuning
- The RL training process is too complicated!
- Is it possible to directly train a language model from the human preference annotations?

DPO: Direct Preference Optimization: Your Language Model is Secretly a Reward Model (Rafailov et. al, 2023)



DPO: Direct Preference Optimization: Your Language Model is Secretly a Reward Model (Rafailov et. al, 2023)

- Looking into the PPO objective

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(y|x)}\left[r_\phi(x,y)\right] - \beta\mathbb{D}_{\text{KL}}\left[\pi_\theta(y \mid x) \,||\, \pi_{\text{ref}}(y \mid x)\right]$$

- Deriving optimal closed-form solution

$$\begin{aligned}
&\max_{\pi} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi}\left[r(x,y)\right] - \beta\mathbb{D}_{\text{KL}}\left[\pi(y|x) \,||\, \pi_{\text{ref}}(y|x)\right] \\
&\quad= \max_{\pi} \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{y\sim\pi(y|x)}\left[r(x,y) - \beta\log\frac{\pi(y|x)}{\pi_{\text{ref}}(y|x)}\right] \\
&\quad= \min_{\pi} \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{y\sim\pi(y|x)}\left[\log\frac{\pi(y|x)}{\pi_{\text{ref}}(y|x)} - \frac{1}{\beta}r(x,y)\right] \\
&\quad= \min_{\pi} \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{y\sim\pi(y|x)}\left[\log\frac{\pi(y|x)}{\frac{1}{Z(x)}\pi_{\text{ref}}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)} - \log Z(x)\right]
\end{aligned}$$

partition function:

$$Z(x) = \sum_y \pi_{\text{ref}}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)$$

Direct Preference Optimization

- PPO Objective

$$\min_{\pi} \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{y\sim\pi(y|x)}\left[\log \frac{\pi(y|x)}{\frac{1}{Z(x)}\pi_{\text{ref}}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)} - \log Z(x)\right]$$

partition function:

$$Z(x) = \sum_{y} \pi_{\text{ref}}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)$$

- Partition function is a function of only x and π_{ref} , but does not depend on the policy π
- Therefore, we can define $\pi^*(y|x) = \frac{1}{Z(x)}\pi_{\text{ref}}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)$

- $\pi^*(y|x)$ is a valid probability, therefore the objective can be seen as a KL divergence between two probability distribution
- The optimal solution of the objective $\pi(y|x) = \pi^*(y|x) = \frac{1}{Z(x)}\pi_{\text{ref}}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)$

Direct Preference Optimization

- Every reward function induce an optimal policy

$$\pi_r(y \mid x) = \frac{1}{Z(x)} \pi_{\text{ref}}(y \mid x) \exp\left(\frac{1}{\beta} r(x, y)\right)$$

- Every policy is the optimal policy of some reward function

$$r(x, y) = \beta \log \frac{\pi_r(y \mid x)}{\pi_{\text{ref}}(y \mid x)} + \beta \log Z(x)$$

This term is intractable!

- Key idea: train the policy model so that r(x,y) fits the human preference data!

Direct Preference Optimization

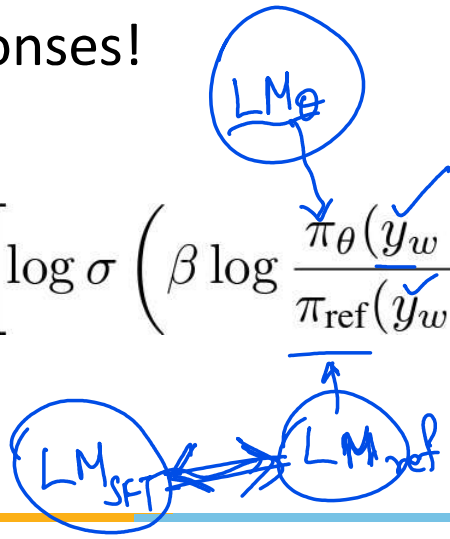
- Recall the reward model training loss

$$\mathcal{L}_R(r_\phi, \mathcal{D}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))\right]$$

- The partition function cancels out when we take the difference between the reward of a pair of responses!
- DPO training objective:

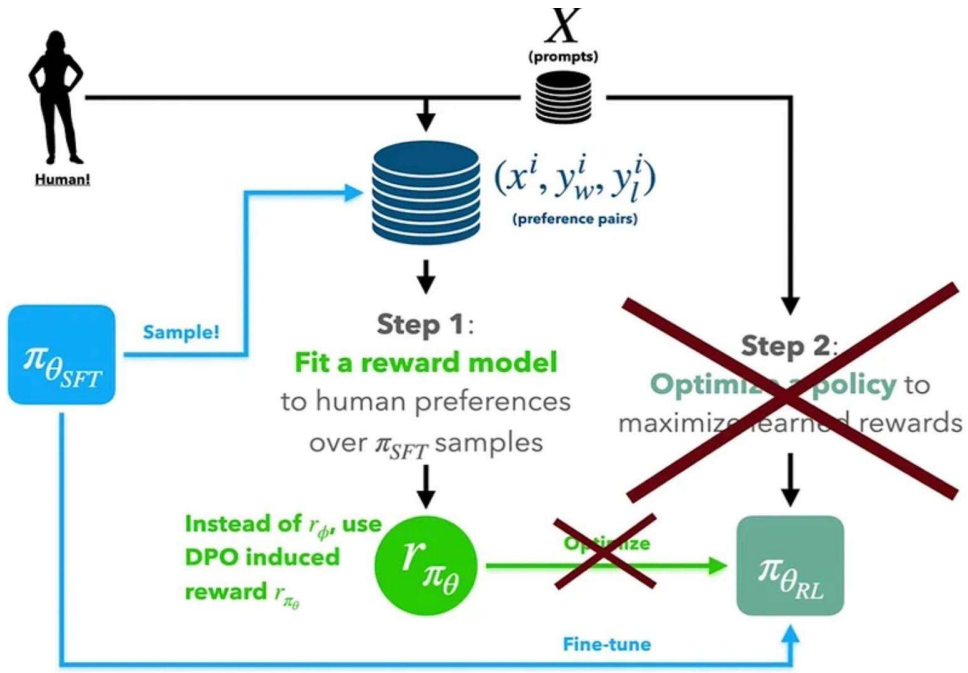
$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right)\right]$$

- A simple classification loss!

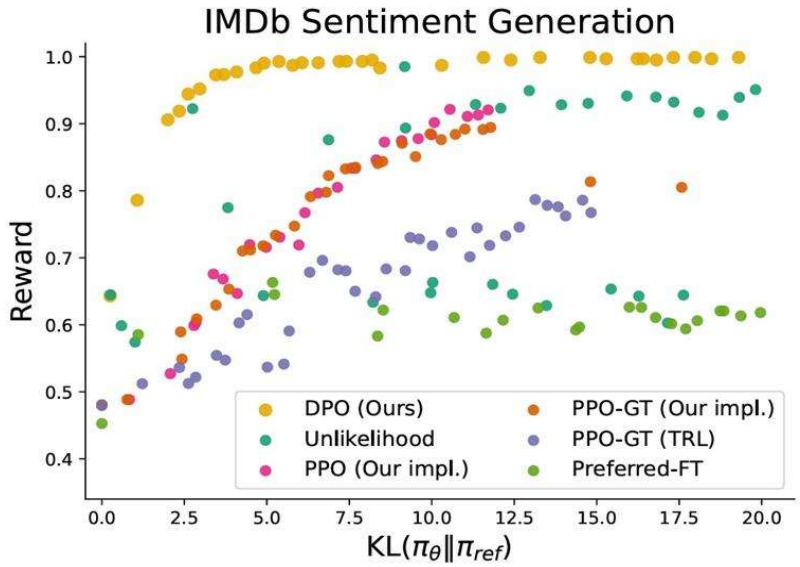


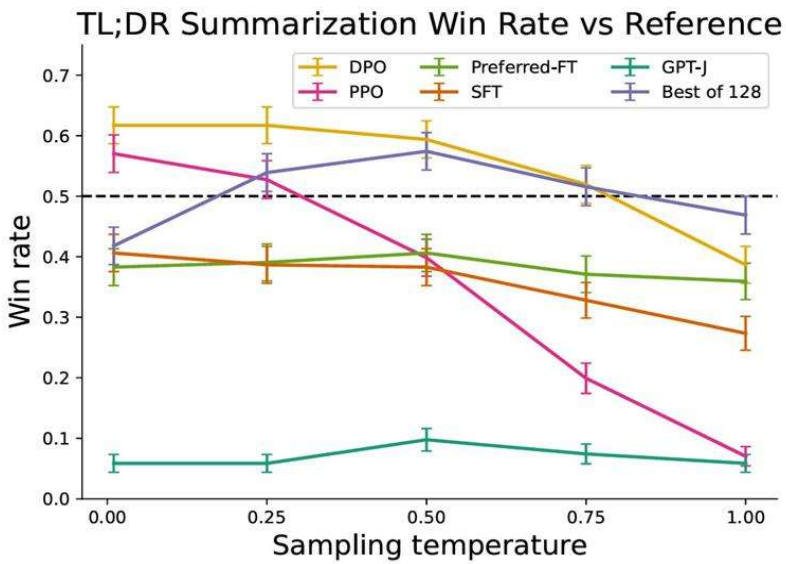
What does DPO do?

- DPO eliminates the need to train a reward model, sample from the LLM during fine-tuning, or perform significant hyperparameter search.



Comparison with Baseline Models

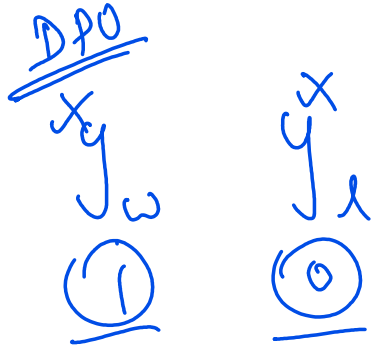


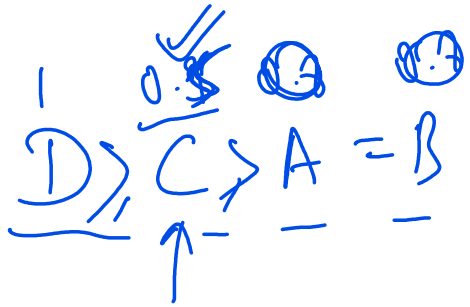


- Preferred-FT: Fine-tune the model on y_w
- PPO-GT: reward model is the ground truth of the sentiment
- Unlikelihood: optimize the policy model to maximize $P(y_w)$ and minimize $P(y_l)$
- Best of N: sampling N responses from the SFT model (very inefficient)

Comparison between PPO and DPO

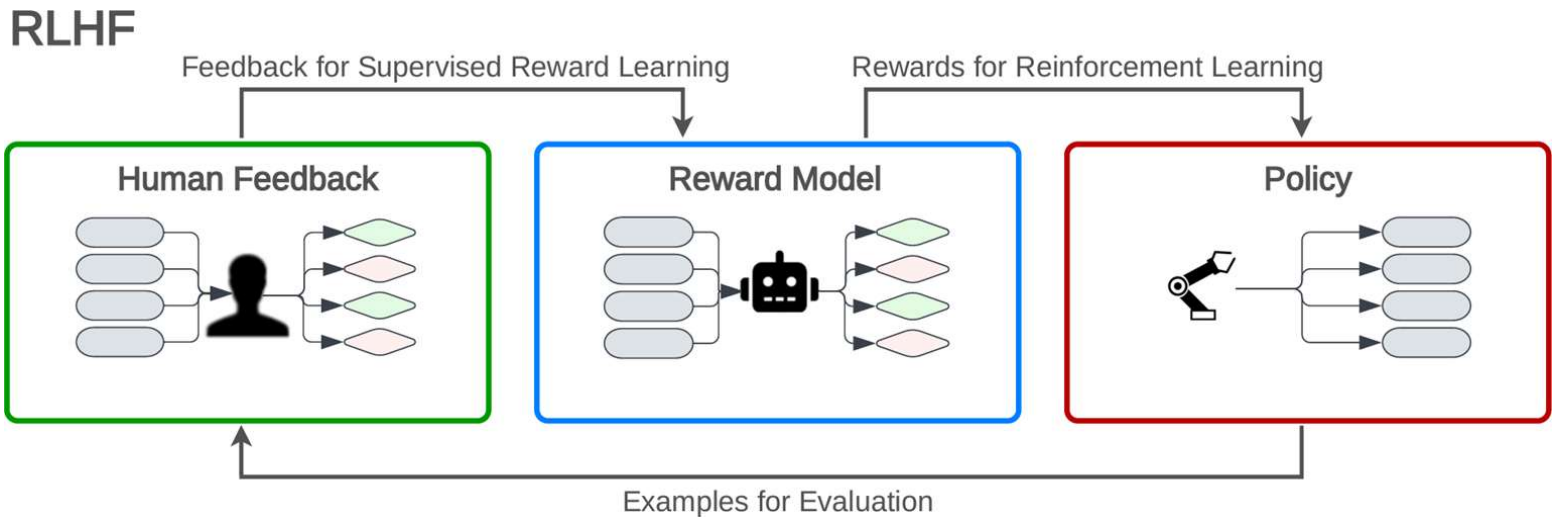
- DPO training is cheaper and more stable than PPO training
- PPO can handle more informative human feedback (e.g., numerical ratings) while DPO can only handle binary signals





Open Problems and Fundamental Limitations of RLHF (Casper et. al, 2023)

- Challenges within each step: human feedback, reward model and policy



Challenges with Obtaining Human Feedback

- Human evaluators may have biases
  - Studies found that ChatGPT models became politically biased post RLHF.
- Good oversight is difficult
  - Evaluators are paid per example and may make mistakes given time constraints; poor feedback on evaluating difficult tasks
- Data quality
  - cost / quality tradeoff
- Tradeoff between richness and efficiency of feedback types
  - comparison-based feedback, scalar feedback, correction feedback, language feedback, …

Challenges with the Reward Model

- A single reward function cannot represent a diverse society of humans
- Reward misgeneralization: reward models may fit with human preference data with unexpected features
- Evaluation of a reward model is difficult and expensive

Challenges with the Policy

- Robust reinforcement learning is difficult
  - balance between exploring new actions and exploiting known rewards
  - the challenge intensifies in high-dimensional or sparse reward settings
- Policy mis generalization: training and deployment environment is difference

Group Relative Policy Optimization (GRPO)

https://arxiv.org/pdf/2402.03300

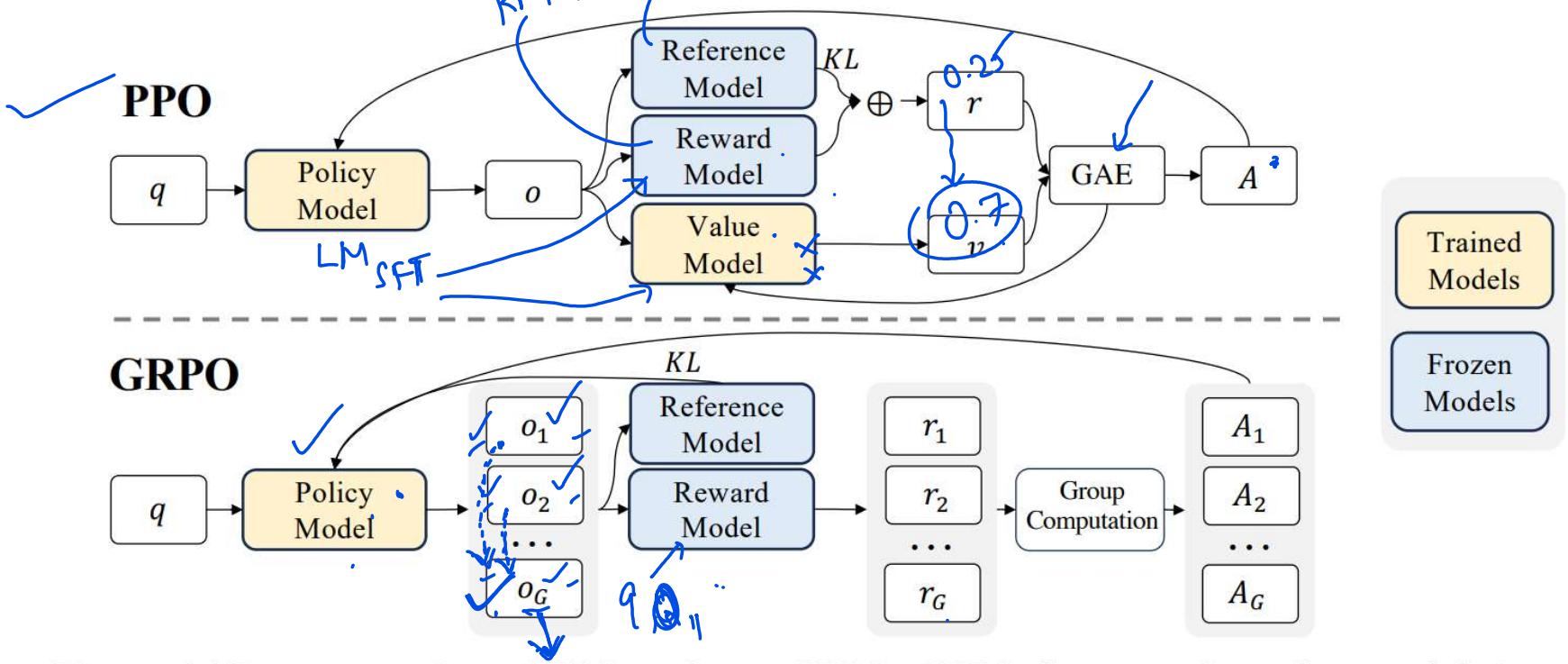


Figure 4 | Demonstration of PPO and our GRPO. GRPO foregoes the value model, instead estimating the baseline from group scores, significantly reducing training resources.

Comparison of PPO and GRPO

Feature / Aspect	Proximal Policy Optimization (PPO)	Group Relative Policy Optimization (GRPO)
Core Mechanism	Actor-Critic; optimizes clipped surrogate objective.	Critic-less; optimizes based on relative rewards within a generated group.
Advantage Estimation	Learns a separate state-value function (critic).	Calculates advantage using a reward model on a group of policy outputs.
Computational Cost	Higher (actor + critic networks).	Lower (actor only, but group generation adds some cost).
Memory Usage	Higher (two networks).	Lower (one main network for policy).
Training Simplicity	More complex (tuning actor & critic, their interaction).	Simpler training loop (no critic-related hyperparameters/instability).
Stability Focus	Robust due to objective clipping; prevents large policy shifts.	Stable due to relative ranking within groups; avoids critic errors.
Data Generation / Update	Typically, simpler sample collection per update.	Generates multiple responses per input for group comparison.
Reward Sensitivity	Performance sensitive to critic accuracy and reward shaping.	Performance highly sensitive to the quality of the external reward model.
Algorithm Maturity	Mature and widely adopted	Newer, gaining traction, particularly for LLM alignment and reasoning.

Comparison of PPO and GRPO

Best Scenario for Proximal Policy Optimization (PPO):

Robotics Control Tasks:
Example: Training a robotic arm to pick and place objects using simulated environments

PPO is highly effective in environments where stability and sample efficiency are crucial, such as training robotic arms or locomotion agents in simulation. Its clipped objective prevents large policy updates, which is essential for maintaining safe and stable learning in continuous control tasks.

Best Scenario for Group Relative Policy Optimization (GRPO):

Large Language Model (LLM) Alignment via Preference Optimization:
Example: Fine-tuning a chatbot to generate helpful and safe responses by comparing multiple completions and optimizing based on human or model preferences.

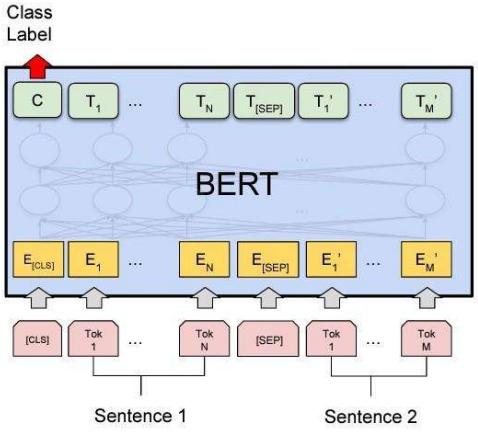
GRPO excels in scenarios where absolute rewards are hard to define, but relative preferences among outputs are available. This makes it ideal for aligning LLMs with human preferences by comparing multiple generated responses and selecting the best one.

Session Content

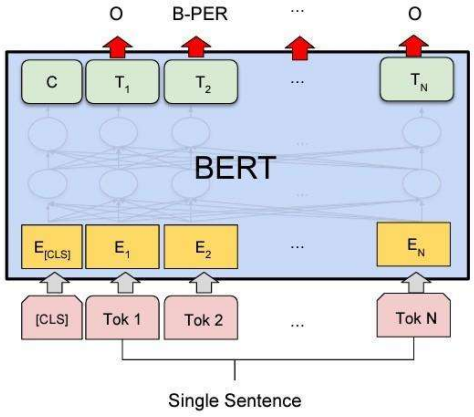
1. Prompt Engineering
2. Instruction Tuning
3. Reinforcement Learning
4. **Parameter-Efficient Fine-Tuning**

Background: Vanilla Fine-tuning

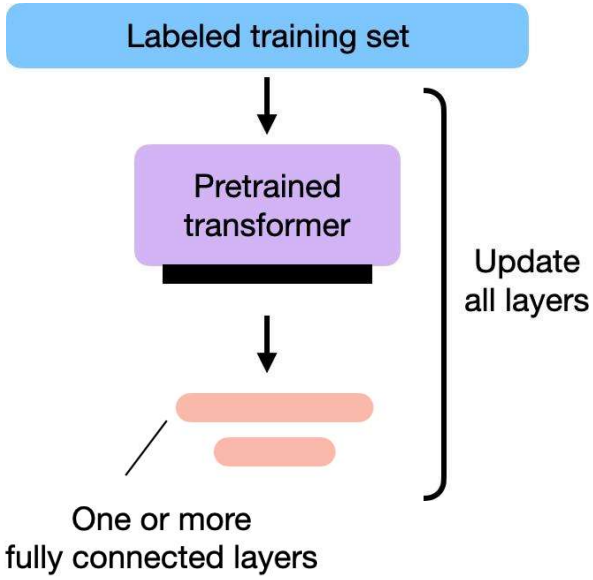
- Attach a task-specific layer to the last layer of the pre-trained transformer output
- Update the weights of all the parameters by backpropagating gradients on a downstream task



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



(d) Single Sentence Tagging Tasks: CoNLL-2003 NER



Prompting

- Prompting a language model with a natural description of the task, and possibly several few-shot examples. No gradient updates.

```
1  Translate English to French:        ← task description
2  cheese =>                           ← prompt
```

```
1  Translate English to French:        ← task description
2  sea otter => loutre de mer          ← examples
3  peppermint => menthe poivrée        ←
4  plush girafe => girafe peluche      ←
5  cheese =>                           ← prompt
```

Vanilla Fine-Tuning vs. Prompting

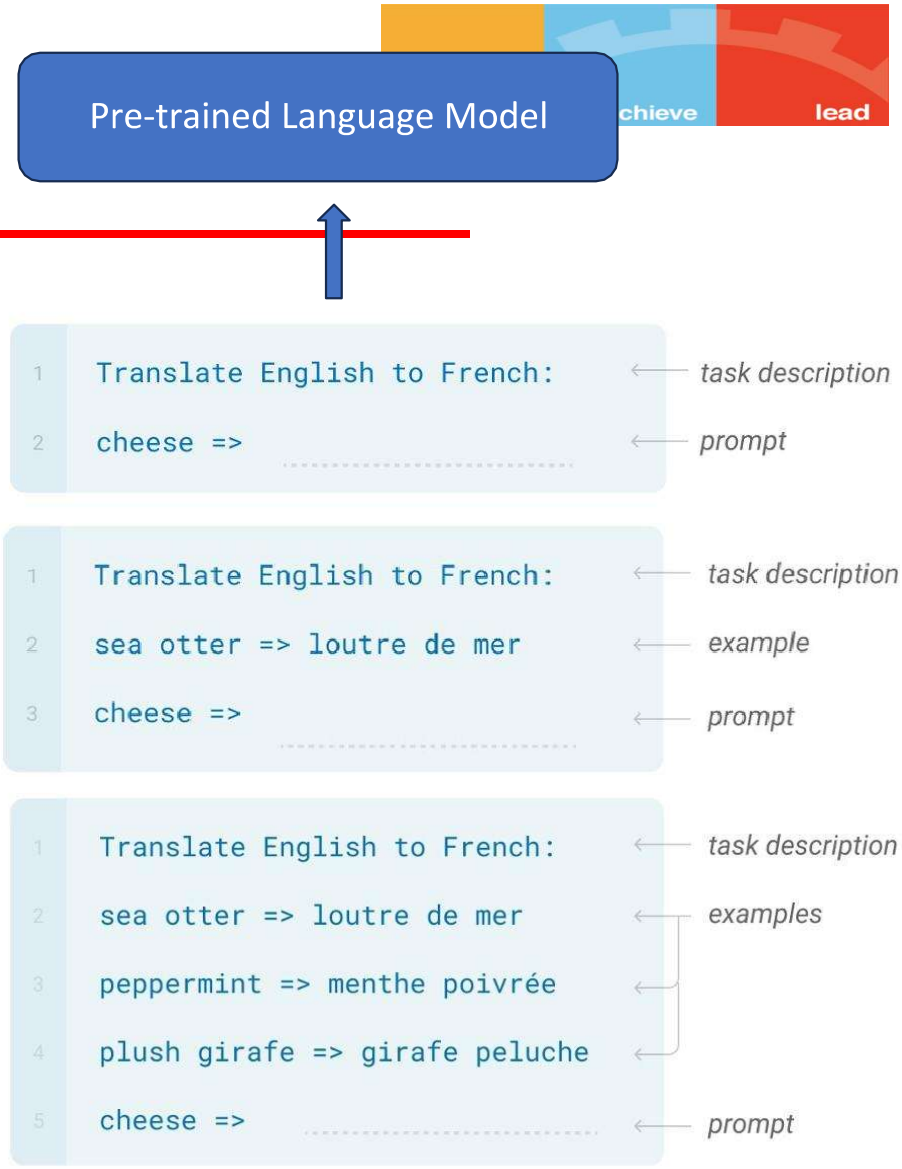
- Vanilla fine-tuning
  - Pros: Can utilize more training data
  - Pros: Lead to stronger performance with more training data
  - Cons: Computationally expensive to train the complete network
  - Cons: Need to store a full set of model weights per task
- Prompting
  - Pros: Training-data efficient
  - Pros: Computational efficient
  - Cons: Performance depends on prompts and examples
  - Cons: Finding a good prompt could be challenging

Parameter-Efficient Fine-Tuning

- Rather than fine-tuning the parameters in the entire model, only fine-tune a small set of weights.
  - Addition: add a small external network for each task
    - Prompt-based Methods
    - Adapter-based Methods
  - Reparameterization: reparametrize the model parameter to be more efficient for training
    - LoRA

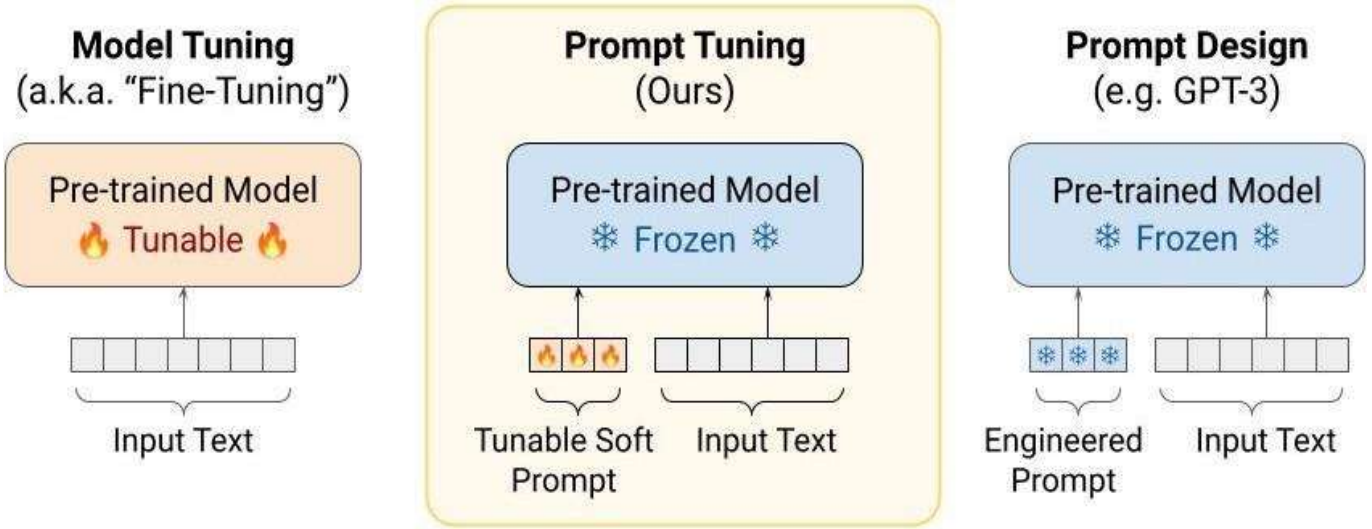
Prompt Engineering

- Paraphrasing the task instruction
- Adding detailed examples
- For each new task, search over the possible sequence space to find the prompt with the best output performance. -> **computationally expensive!**
- Can we use a set of parameters to replace these prompts and train them with labeled sets?



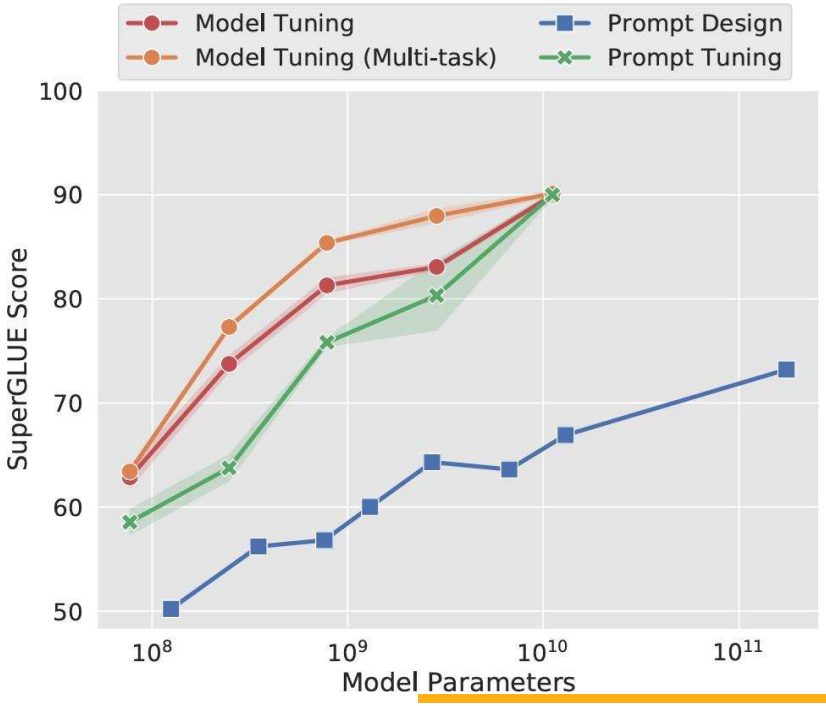
Prompt-Tuning

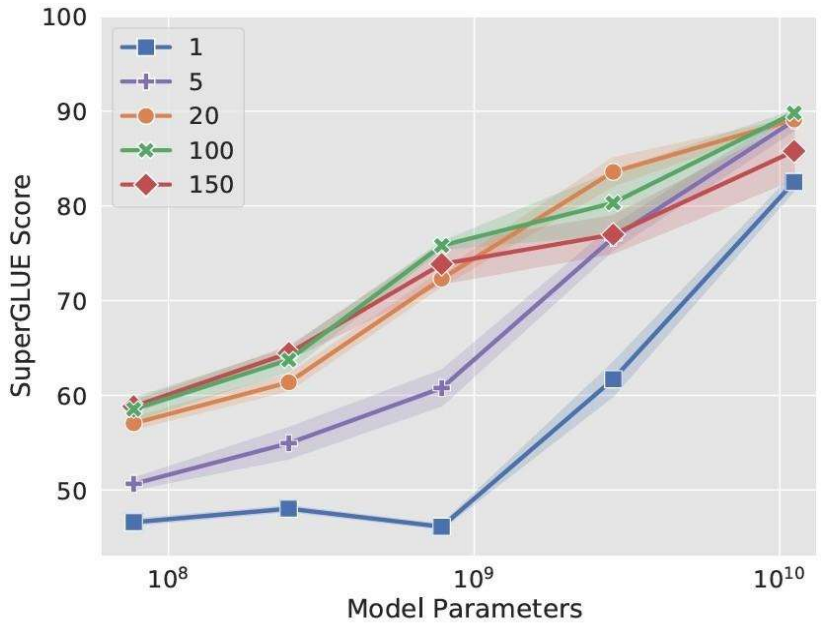
- Prepend a sequence of tokens as tunable embeddings to the input data (as soft prompts)
- freeze the whole Transformer model during training, and only tune the prepended soft prompts
- Only a small set of parameters need to be stored for each task



The Power of Scale for Parameter-Efficient Prompt Tuning (Lester et. al, 2021)

- Prompt-tuning becomes more effective when the pre-trained model becomes larger
- Larger models perform well even with a small number of prompt tokens

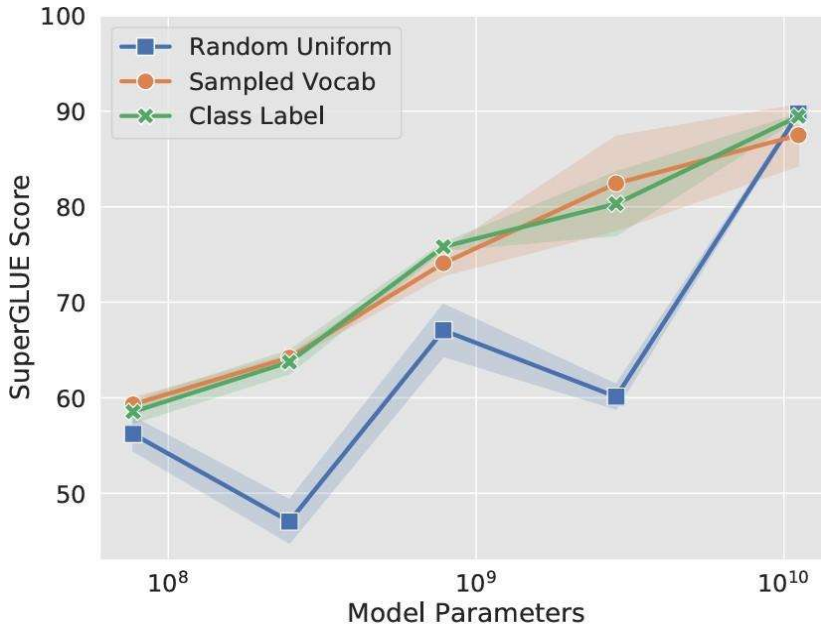




(a) Prompt length

The Power of Scale for Parameter-Efficient Prompt Tuning (Lester et. al, 2021)

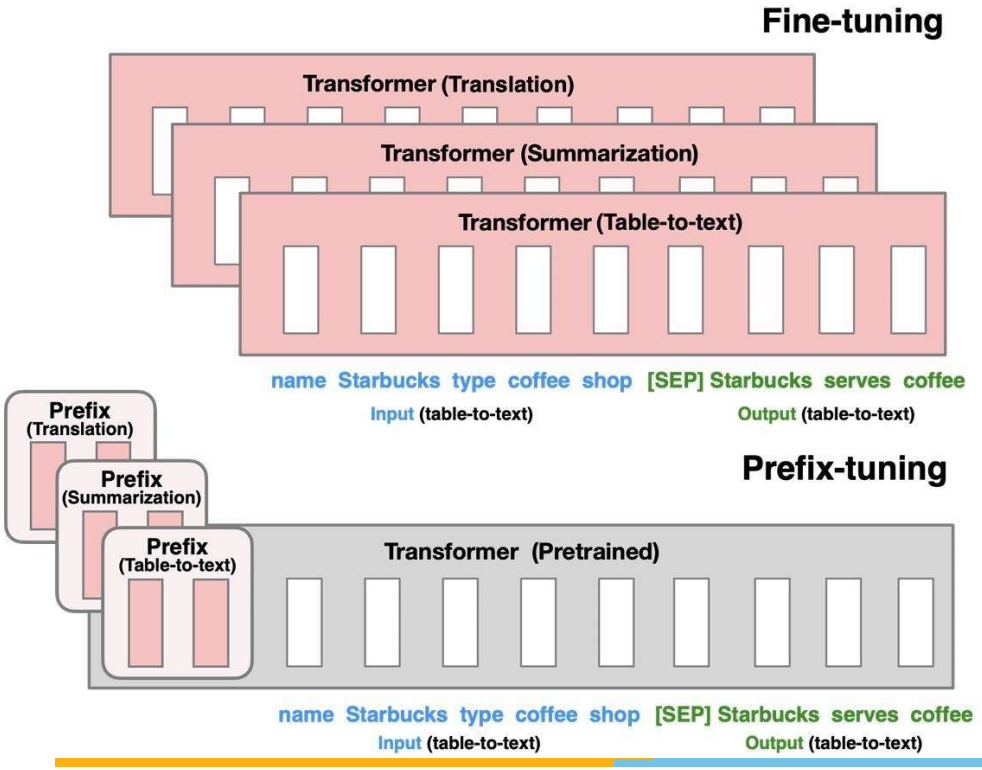
- Initializing prompt tokens with real tokens in vocabulary is helpful



(b) Prompt initialization

Prefix-Tuning: Optimizing Continuous Prompts for Generation (Li et. al, 2021)

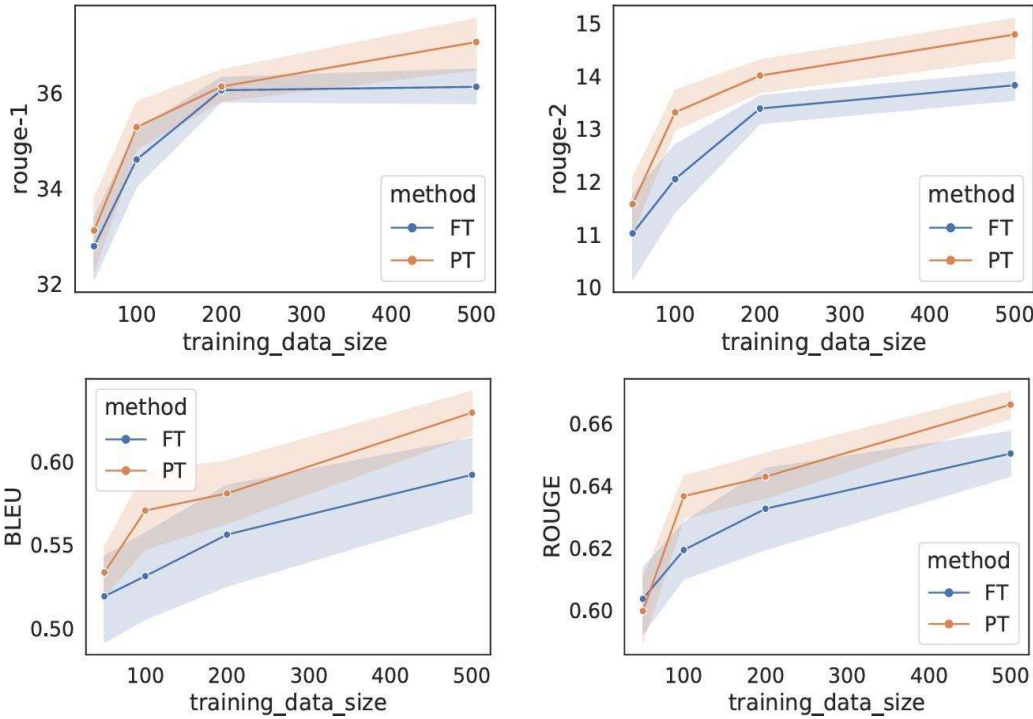
Similar with prompt-tuning, except that the soft prompt tokens are prepended to each layer in the Transformer instead of just the input layer.



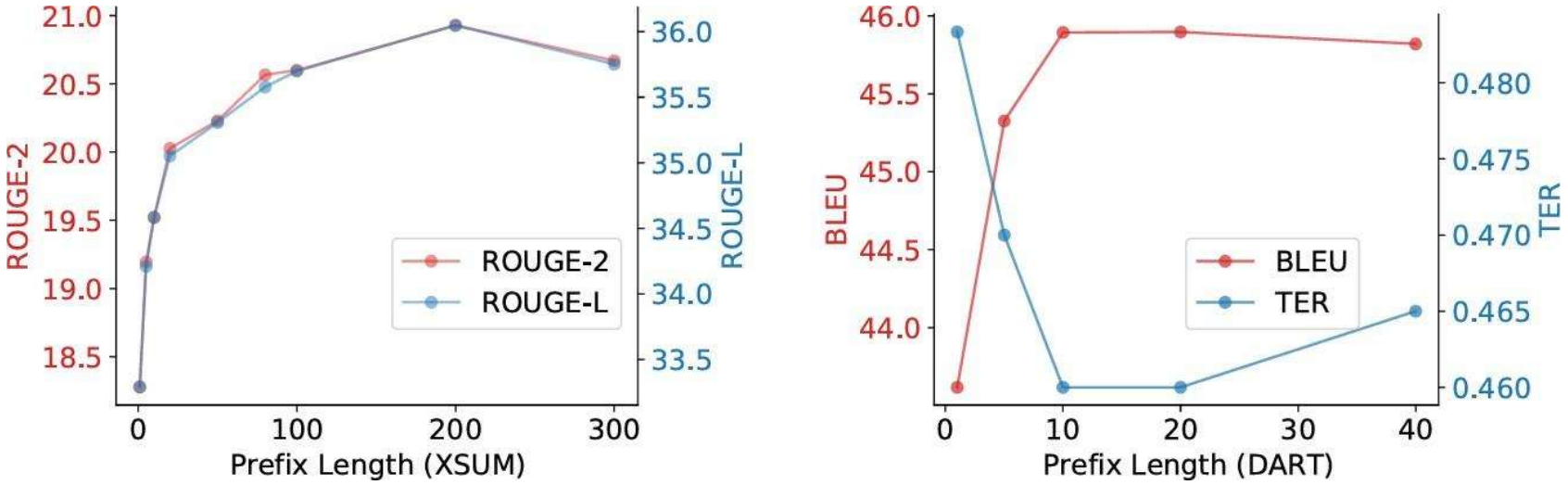
Experiments on Text Generation

Prefix-tuning works better than fine-tuning under low-data settings.

Source	name : The Eagle | type : coffee shop | food : Chinese | price : cheap | customer rating : average | area : riverside | family friendly : no | near : Burger King
Prefix (50)	The Eagle is a cheap Chinese coffee shop located near Burger King.
Prefix (100)	The Eagle is a cheap coffee shop located in the riverside near Burger King. It has average customer ratings.
Prefix (200)	The Eagle is a cheap Chinese coffee shop located in the riverside area near Burger King. It has average customer ratings.
Prefix (500)	The Eagle is a coffee shop that serves Chinese food. It is located in the riverside area near Burger King. It has an average customer rating and is not family friendly.
FT (50)	The Eagle coffee shop is located in the riverside area near Burger King.
FT (100)	The Eagle is a cheap coffee shop near Burger King in the riverside area. It has a low customer rating and is not family friendly.
FT (200)	The Eagle is a cheap Chinese coffee shop with a low customer rating. It is located near Burger King in the riverside area.
FT (500)	The Eagle is a cheap Chinese coffee shop with average customer ratings. It is located in the riverside area near Burger King.



Prefix Length



- Performance increases as the prefix length increases up to athreshold (200 for summarization and 10 for table-to-text) and then a slight performance drop occurs.

Prefix Initialization

- Random initialization leads to low performance with high variance.
- Initializing the prefix with real words significantly improves generation, as shown in Figure5.
- Initializing with task relevant words such as “summarization” and “table-to-text” obtains slightly better performance than task irrelevant words such as “elephant” and “divide”.

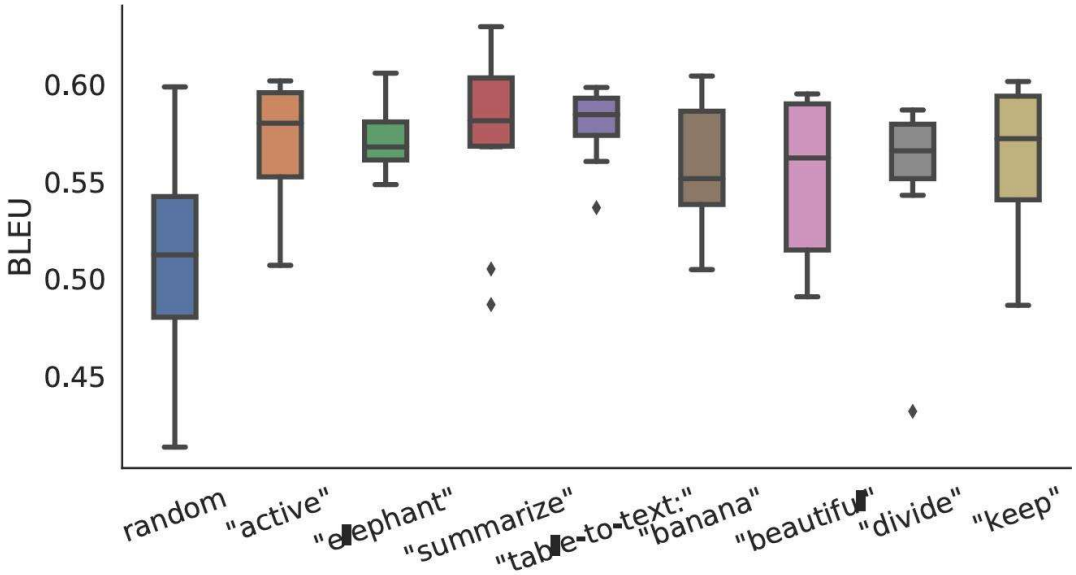


Figure 5: Initializing the prefix with activations of real words significantly outperforms random initialization, in low-data settings.

Issues with Prompt/Prefix-Tuning

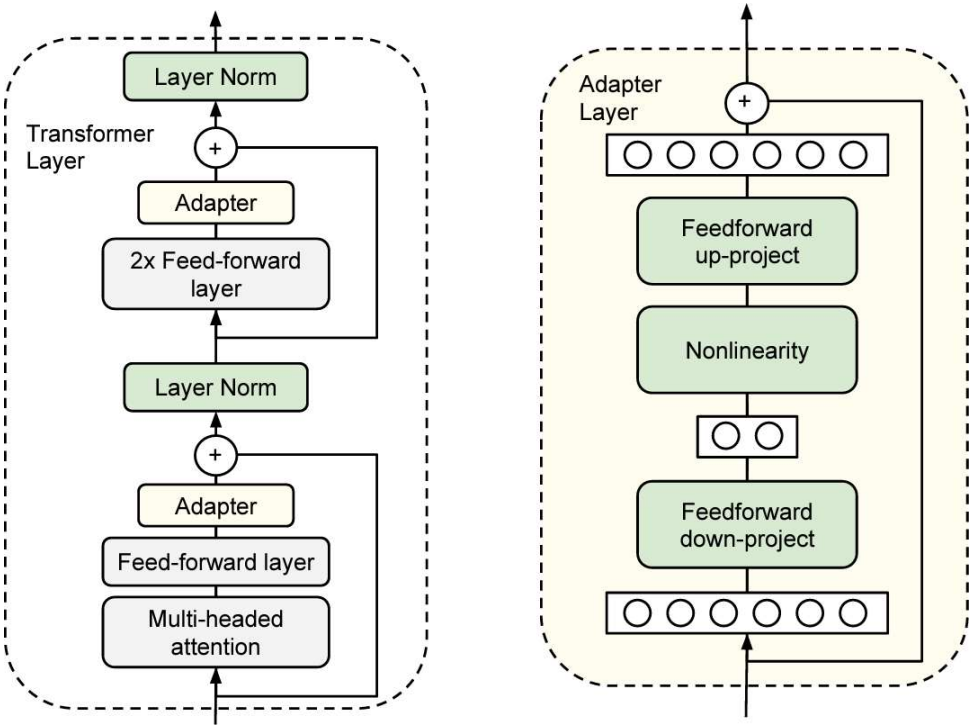
- Optimal prefix length may be different for tasks
- The prefix occupies the length of your input context to the Transformer

What are Adapters?

- Vanilla fine-tuning can be seen as adding an extra layer to the top of a Transformer
- Adapter modules perform more general architectural modifications: injecting new layers/modules into the original network.
- During training, the original network weights are untouched, only the adapter weights are updated.

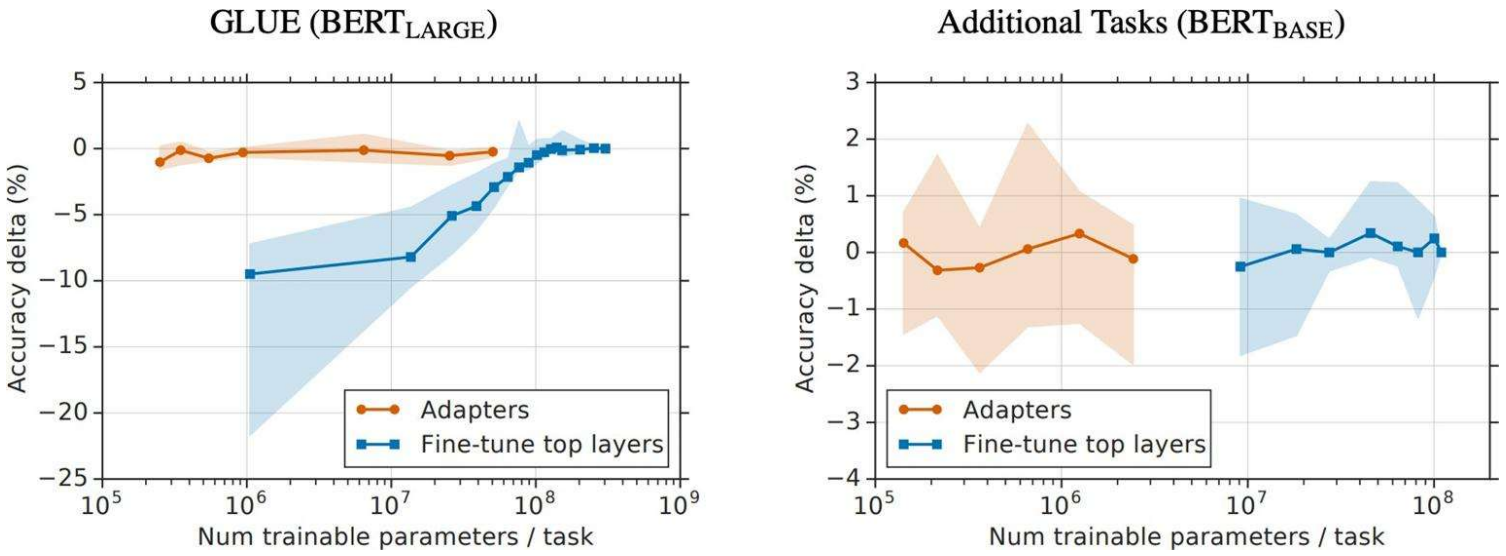
Parameter-Efficient Transfer Learning for NLP (Houlsby et. al, 2019)

- Adding adapter layers to each transformer layer: after the self-attention layer and the feed-forward layer
- Adapter modules have two main features
  - a small number of parameters
  - a near-identity initialization
- Only adapter layers and the final classification layer is updated during training



Comparison with Fine-Tuning

- Adapter-based tuning achieves a similar performance to full fine- tuning with several orders of magnitude fewer trained parameters.



Pros and Cons of Adapter-based Methods

- Pros:
  - Empirically very effective in multi-task settings
  - Computationally efficient compared to full fine-tuning
- Cons:
  - Adding in new layers makes the model slower during inference time
  - Make the model size larger

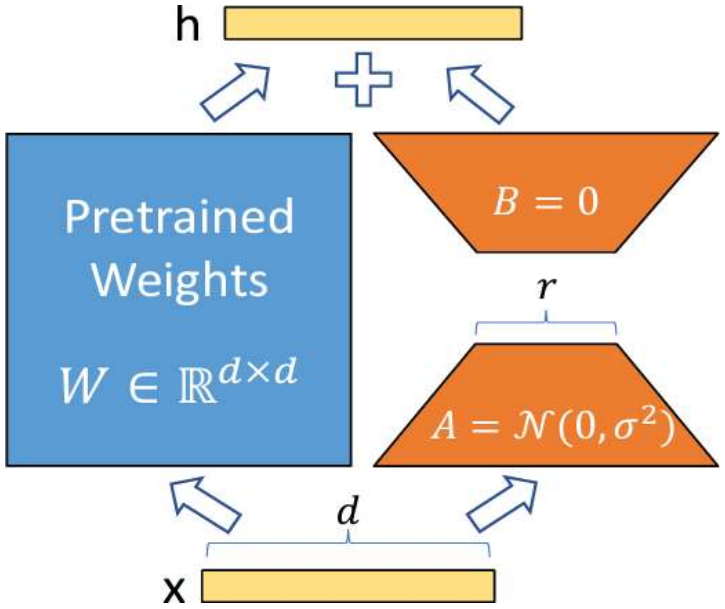
LoRA: Low-Rank Adaptation of Large Language Models (Hu et. al, 2021)

- A neural network contains many dense layers which perform matrix multiplication.
- Inspired by the low intrinsic dimension assumption, hypothesize that the update weights can also have a low intrinsic rank
- Pre-trained matrix to be update:

$$W_0 \in \mathbb{R}^{d \times k}$$

  - Updated matrix

$$W_0 + \Delta W$$



LoRA: Low-Rank Adaptation of Large Language Models (Hu et. al, 2021)

- Reparametrize the updated weight with low-rank decomposition

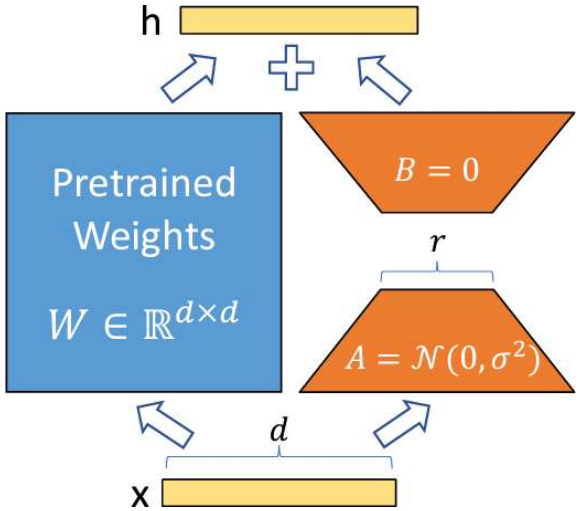
$$W_0 + \Delta W = W_0 + BA$$

- where A and B are low rank matrices

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

- For the hidden state h of an input x , $h = W_0 x$
- The updated hidden state is now

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



Applying LoRA to Transformers

- In principle, LoRA can be applied to any weight matrices in deep learning
- In this study, they focus on applying LoRA to attention matrices in Transformers
- r ranges from 2 to 64
- For GPT3-175B
- VRAM: 1.2TB -> 350GB
- Checkpoint storage: 350GB -> 35MB (10000x smaller)

Comparison with Other Fine-Tuning Methods

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 outperforms several baselines with comparable or fewer trainable parameters.

Which Matrices? & Optimal Rank?

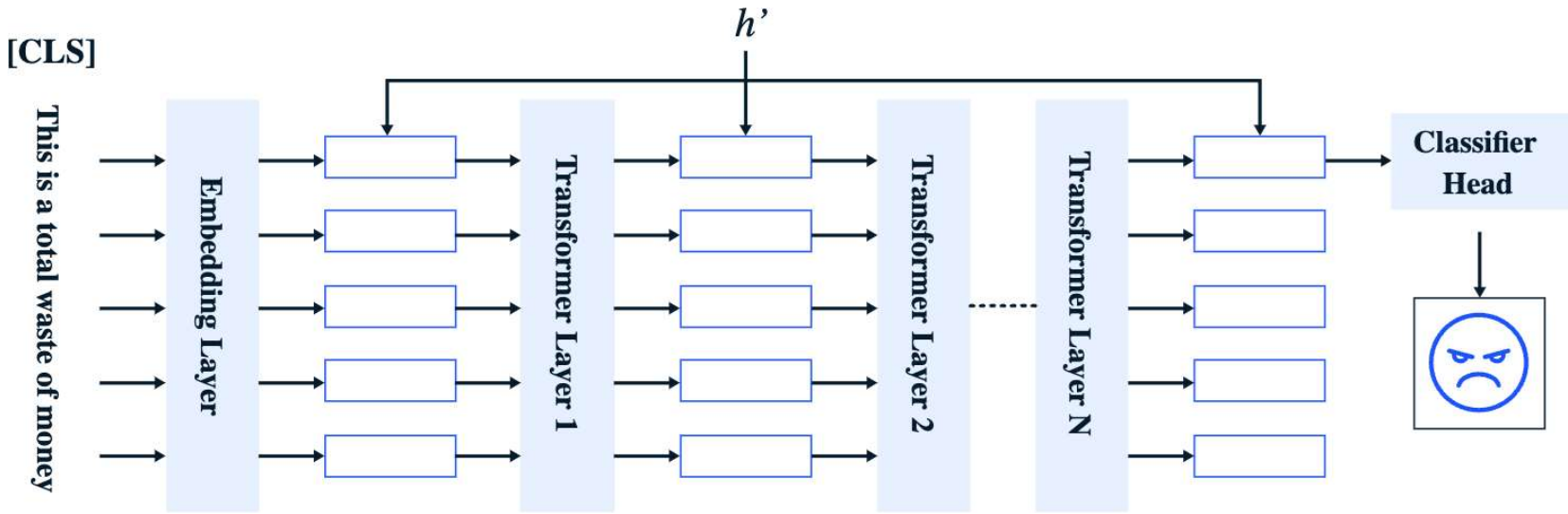
	# of Trainable Parameters = 18M						
Weight Type	W_q	W_k	W_v	W_o	W_q, W_k	W_q, W_v	W_q, W_k, W_v, W_o
Rank r	8	8	8	8	4	4	2
WikiSQL ($\pm$0.5%)	70.4	70.0	73.0	73.2	71.4	**73.7**	**73.7**
MultiNLI ($\pm$0.1%)	91.0	90.8	91.0	91.3	91.3	91.3	**91.7**

- r = 4 and r = 8 already give a good result, and increasing r does not cover more meaningful subspaces

	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

Summary of Parameter-Efficient Fine-Tuning

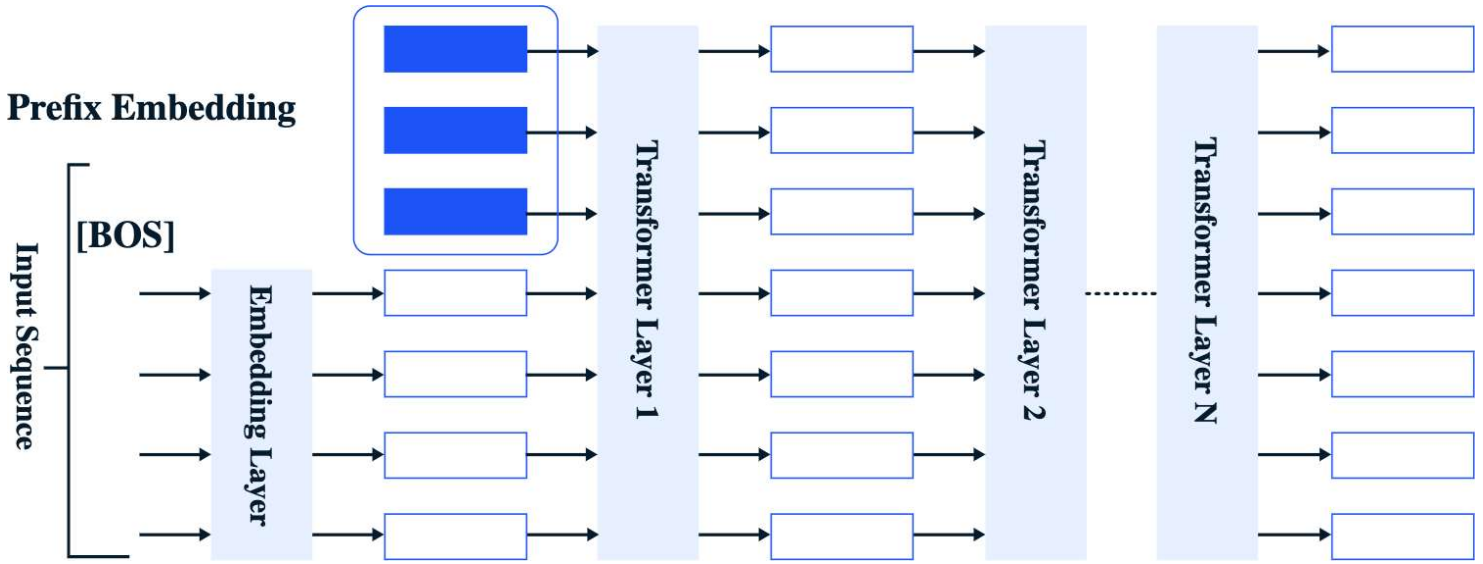
- Vanilla Fine-Tuning



Source: https://www.leewayhertz.com/parameter-efficient-fine-tuning/

Summary of Parameter-Efficient Fine-Tuning

- Prompt-Tuning

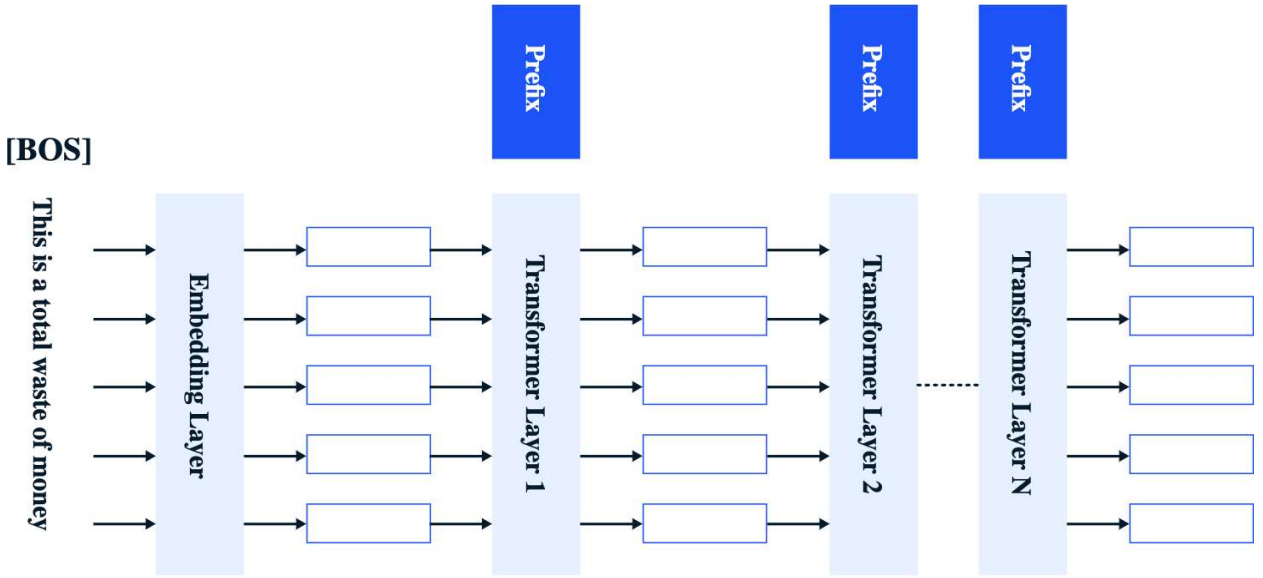



```
def prompt_tuning_attention(input_ids):
    q = x @ W_q
    k = cat([s_k, x]) @ W_k   # prepend a
    v = cat([s_v, x]) @ W_v   # soft prompt
    return softmax(q @ k.T) @ V
```

Source: https://www.leewayhertz.com/parameter-efficient-fine-tuning/

Summary of Parameter-Efficient Fine-Tuning

- Prefix-Tuning

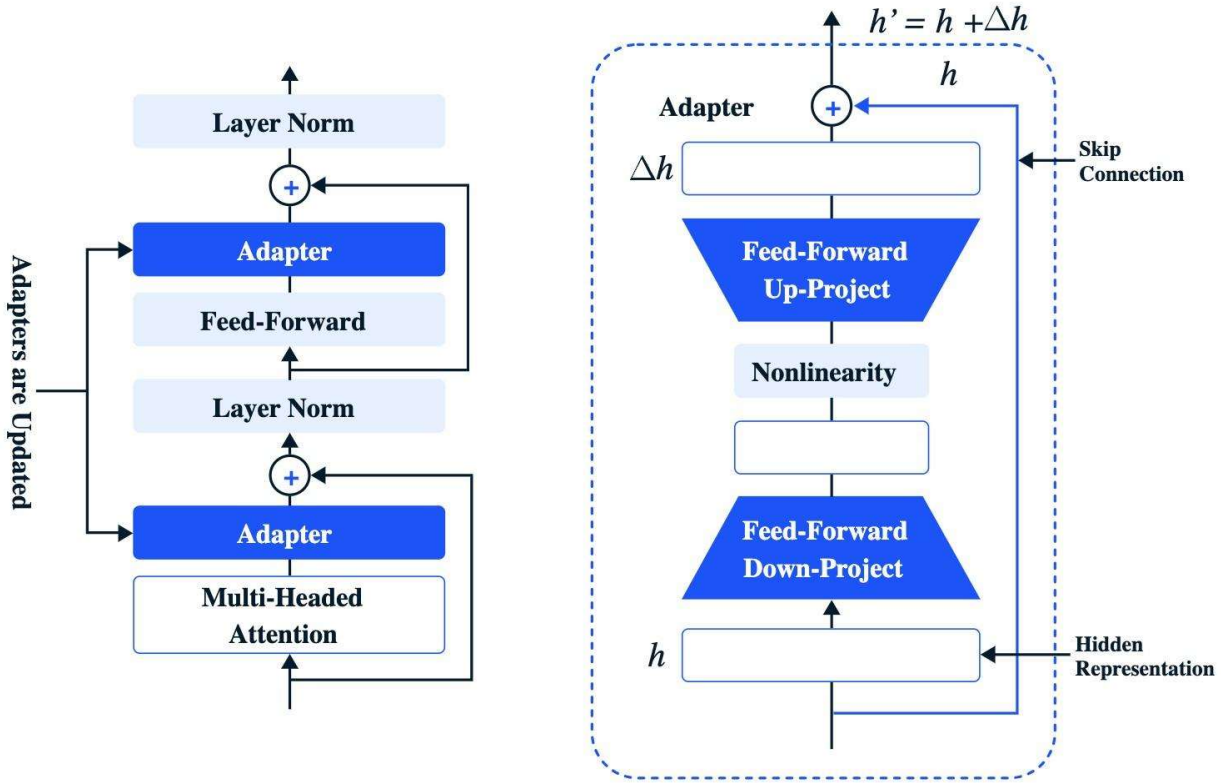



```
def transformer_block_for_prefix_tuning(x):
    soft_prompt = FFN(soft_prompt)
    x = concat([soft_prompt, x], dim=seq)
    return transformer_block(x)
```

Source: https://www.leewayhertz.com/parameter-efficient-fine-tuning/

Summary of Parameter-Efficient Fine-Tuning

- Adapters

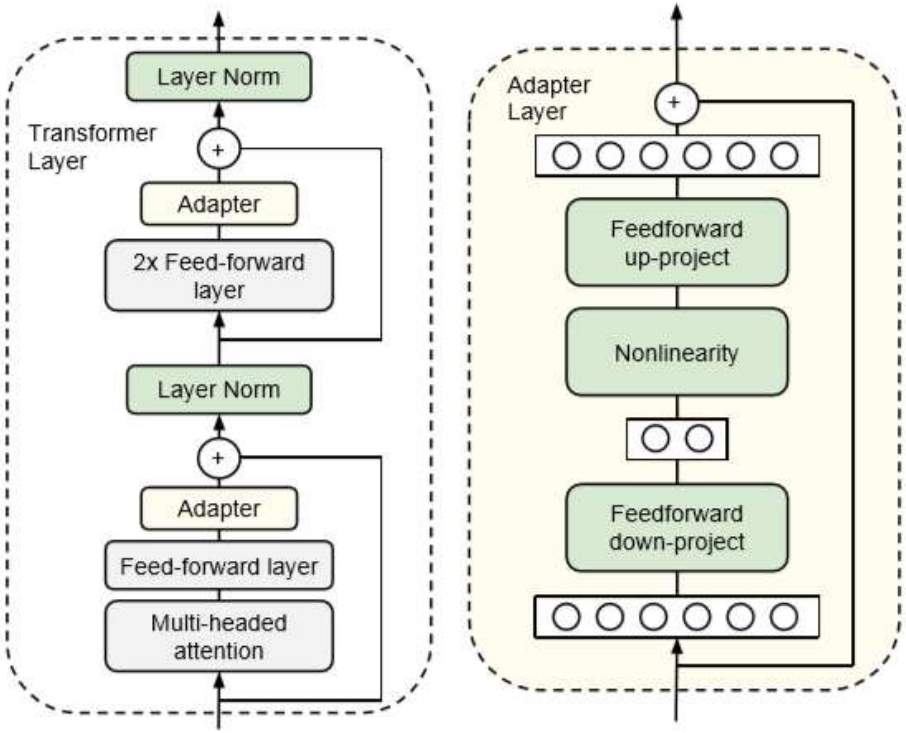


Source: https://www.leewayhertz.com/parameter-efficient-fine-tuning/

Adapter-based Methods

- Instead of changing the original model, adapters are added like plug-ins, preserving the pre-trained knowledge.
- Adapters allow the same base model to be used for different tasks by just training these small, task-specific modules.

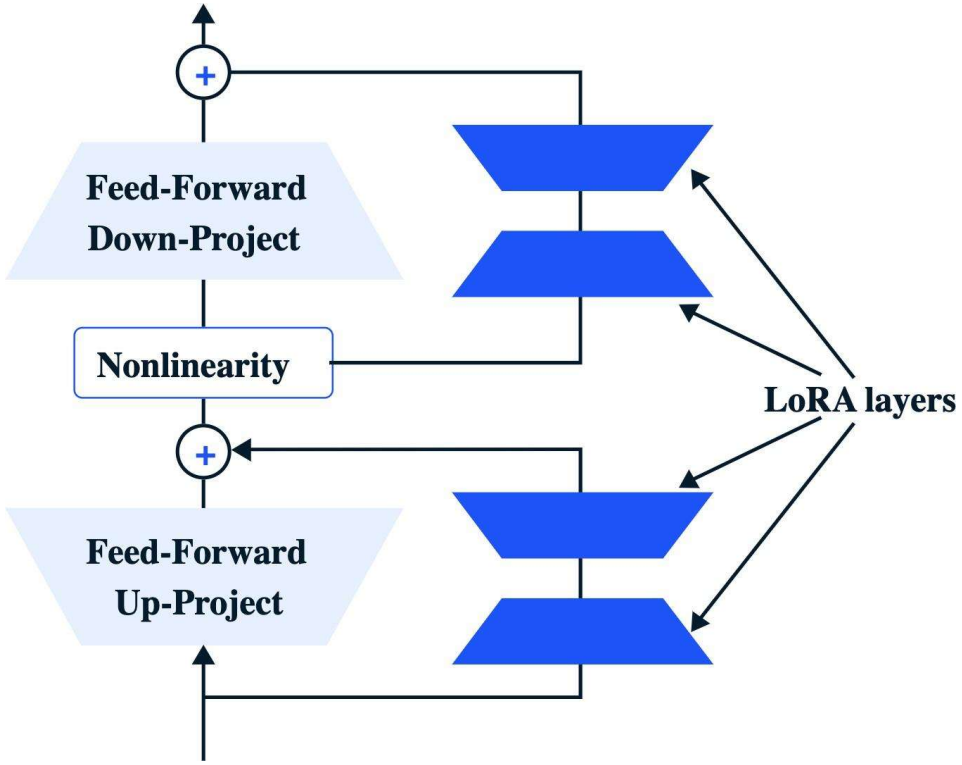
```
def transformer_block_with_adapter(x):
    residual = x
    x = SelfAttention(x)
    x = FFN(x)
    x = Adapter(x)  # Adapter after FFN
    x = LN(x + residual)
```



Parameter-Efficient Transfer Learning for NLP, Houlsby et al.

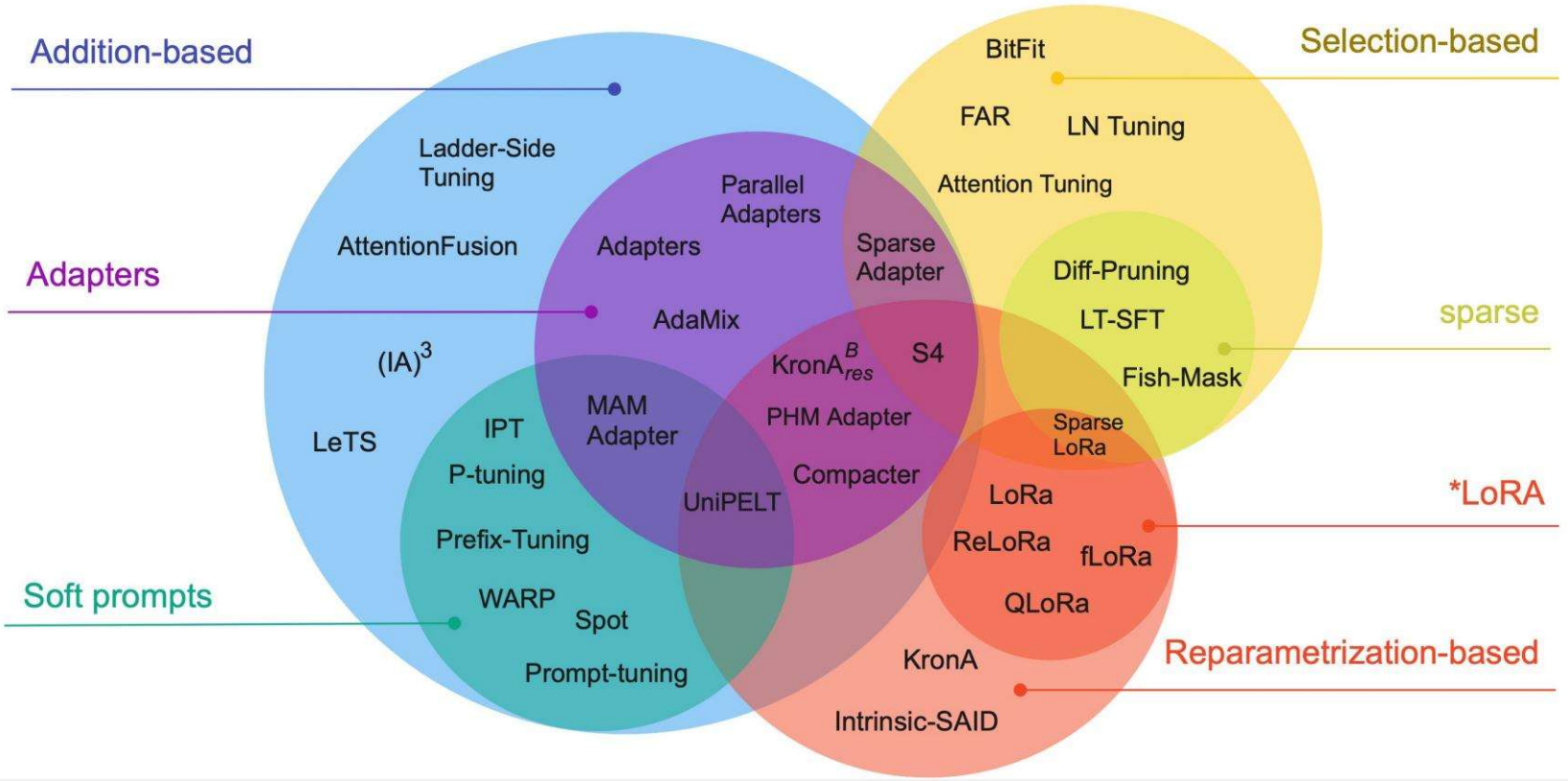
Summary of Parameter-Efficient Fine-Tuning

- LoRA



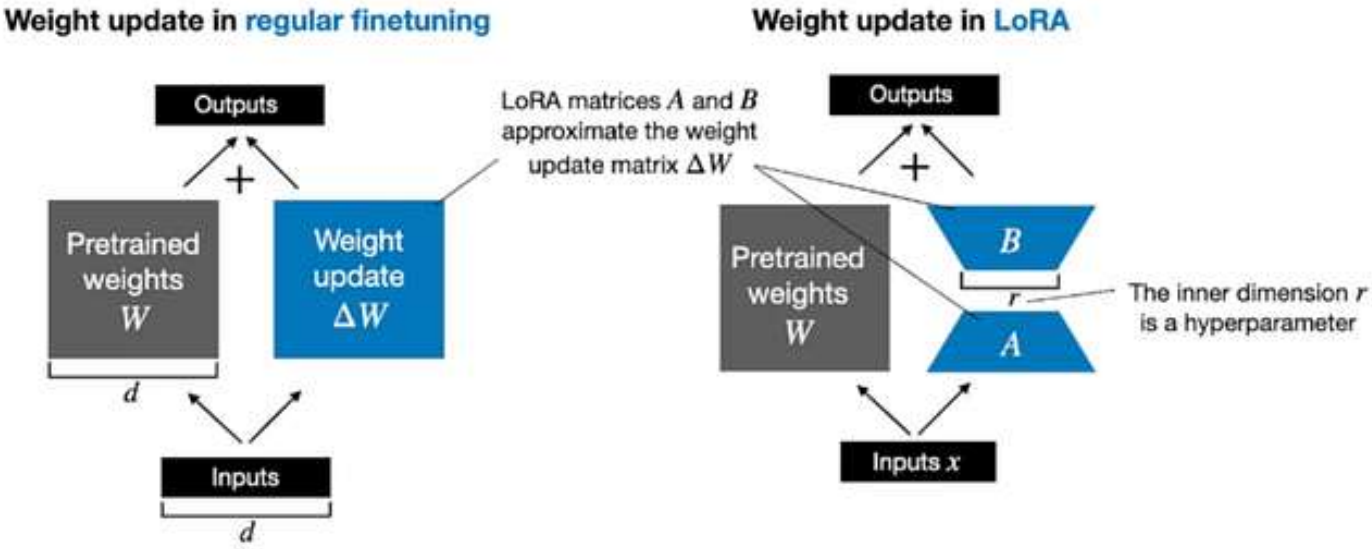
Source: https://www.leewayhertz.com/parameter-efficient-fine-tuning/

Parameter-efficient fine-tuning methods taxonomy



Scaling Down to Scale Up:A Guide to Parameter-Efficient Fine-Tuning, Lialin et al., 2023

Reparametrization-based Method (LoRA)




```
def lora_linear(x):
    h = x @ W             # regular linear
    dh = x @ W_A @ W_B    # low-rank update
    h += scale * dh       # scaling
    return h
```

Scaling Down to Scale Up:A Guide to Parameter-Efficient Fine-Tuning, Lialin et al., 2023

Comparison of PEFT categories

		Additive			Selective		Reparam.	Hybrid
		Adapters	Soft prompts	Other	Structured	Sparse		
Storage efficiency		✓	✓	✓	✓	✓	✓	✓
Training	RAM	✓	✓	✓	✓	✗	✓	maybe
	Speed	✗	✗	maybe	maybe	✗	✓	maybe
Inference	Single-task	✗ Adds overhead			✓ No overhead		✓	maybe
	Mutli-task	✓Allows efficient inference			maybe	✗	maybe	maybe
Implemented in		🤗	🤗	🤗	✗	✗	🤗	🤗

- For training, we evaluate methods by their memory (RAM) efficiency and speed relative to full training.
- ✓ means more effective than regular training, ✗ means less effective, and maybe suggests it might vary based on the specific method within the category.
- For inference, the table shows whether a method introduces single-task overhead and if it facilitates efficient multi-task inference.
- These methods are implemented in popular open-source libraries: HuggingFace PEFT and AdapterHub.

Memory requirements for serving LLMs

$$M = \frac{(P * 4B)}{(32/Q)} * 1.2$$

Symbol	Description
M	GPU memory expressed in Gigabyte
P	The amount of parameters in the model. E.g. a 7B model has 7 billion parameters.
4B	4 bytes, expressing the bytes used for each parameter
32	There are 32 bits in 4 bytes
Q	The amount of bits that should be used for loading the model. E.g. 16 bits, 8 bits or 4 bits.
1.2	Represents a 20% overhead of loading additional things in GPU memory.

For Llama 70B that we will load in 16 bit

$$\frac{70 * 4\text{bytes}}{32/16} * 1.2 = 168\text{GB}$$

2 x A100 80GB should be enough to serve the Llama 2 70B model in 16-bit mode.

Memory requirements

For Meta-Llama-3-8B-Instruct (7.6B):

	float32	float16/bfloat16	int8	int4
Total Size (GB)	27.96	13.98	6.99	3.49
Inference (GB)	33.55	16.77	8.39	4.19
Training using Adam (GB)	111.83	55.92	27.96	13.98
LoRA Fine-Tuning (GB)	34.99	18.22	9.83	5.63

https://huggingface.co/spaces/Vokturz/can-it-run-llm

Quantization

Computations are performed using 32-bit floating-point numbers. This is because 32-bit floating-point numbers provide a good balance between precision (the ability to represent numbers accurately) and range (the range of numbers that can be represented).
Using 32-bit floating-point numbers for all computations can be memory-intensive.

Quantization is a technique to reduce the precision of the numbers used in the model.

In the case of 4-bit quantization, the weights and activations of the network are compressed from 32-bit floating-point numbers to 4-bit integers.

- A 4-bit integer can range from -8 to 7 (16 levels)
- Int-8 tensor with range [-127, 127]
- A 32-bit float can represent the interval from 1.18e-38 to 3.4e38

Quantization

General Formula for Quantization:

X[Int8] = round (127 / absmax(X[FP32])) * X[FP32]
= round(c[FP32] * X[FP32])
where, X[Int8] is the quantized 8-bit integer representation.
X[FP32] is the original 32-bit floating-point value.
c[FP32] is the quantization constant or scale.

Dequantization is simply the inverse:

dequant(c[FP32], X[Int8]) = X[Int8] / c[FP32] = X[FP32]

Quantization with outliers leads to larger quantization errors.

https://doi.org/10.48550/arXiv.2305.14314

4-bit NormalFloat Quantization

Normalization: Transform all weights to a single fixed distribution by scaling with σ and they set an arbitrary range of [-1,1]. Hence it is a mean of 0 and a standard deviation of 1.

Quantization: The normalized weights are quantized to 4 bits. This involves mapping the original high-precision weights (typically 32-bit floating point) to a smaller set of low-precision values. In NF4, these quantization levels are evenly spaced within the range of the normalized weights.

Example:

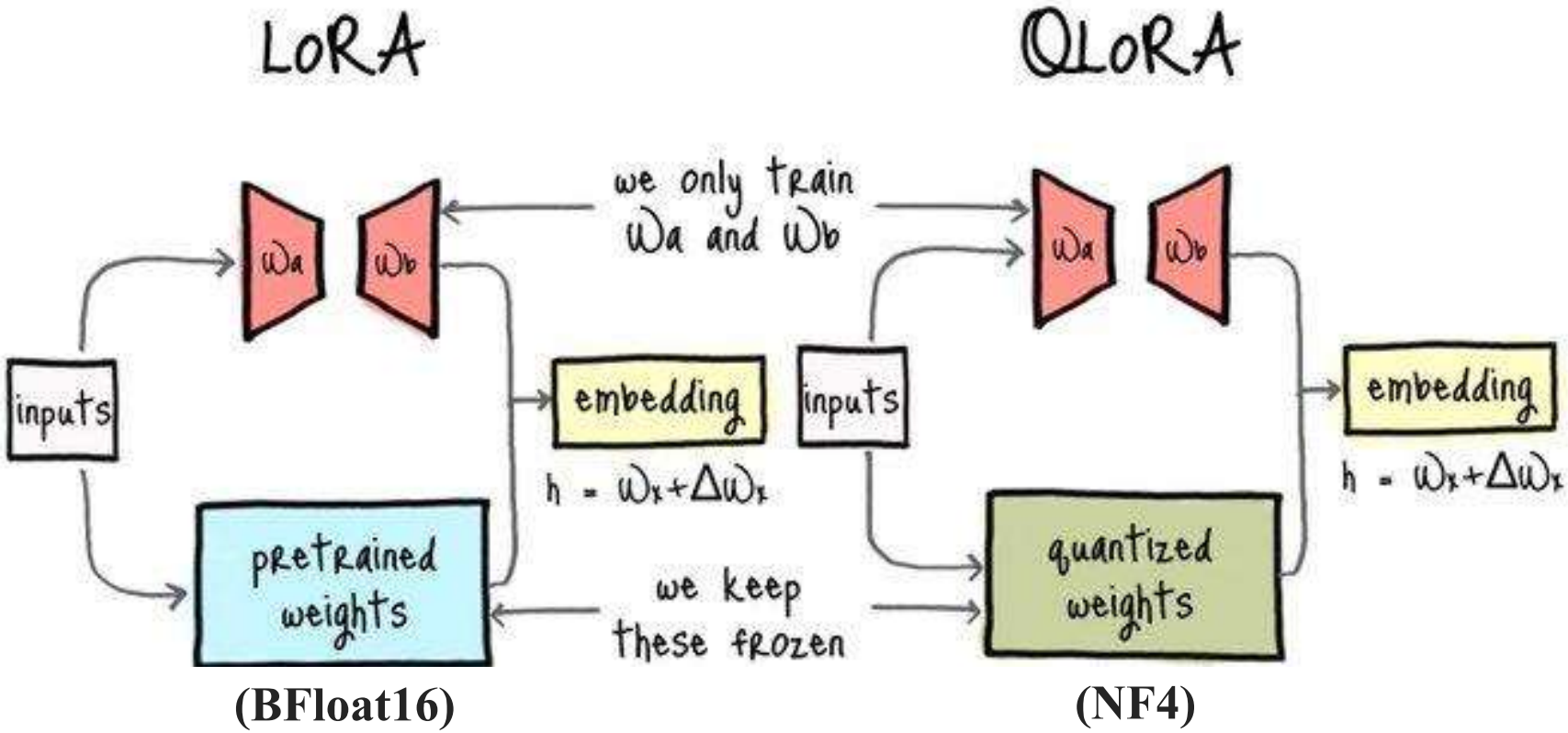
- Imagine a weight with a value of 0.1832
- If we use 4-bit integers to represent 16 levels between -1 and 1, the levels would be:
  -1.0, -0.8667, -0.7333, -0.6, -0.4667, -0.3333, -0.2, -0.0667,
  0.0667, 0.2, 0.3333, 0.4667, 0.6, 0.7333, 0.8667, 1.0
- Quantize 0.1832 to 0.2 (the closest), which might be represented by the integer 10 in our 4-bit system.

4-bit NormalFloat Quantization

Dequantization: During computations, the quantized weights are dequantized back to their original precision (e.g., 32-bit floating point) by mapping the 4-bit values back to their original range.

This introduces a small dequantization error, as the quantized value is an approximation of the original. In our example, the error would be 0.2 - 0.1832 = 0.0168.

QLoRA: Quantization and Low-Rank Adapters



QLoRA: Quantization and Low-Rank Adapters

QLoRA combines quantization with **Low-Rank Adapters (LoRA)** to enable efficient fine-tuning of large language models.

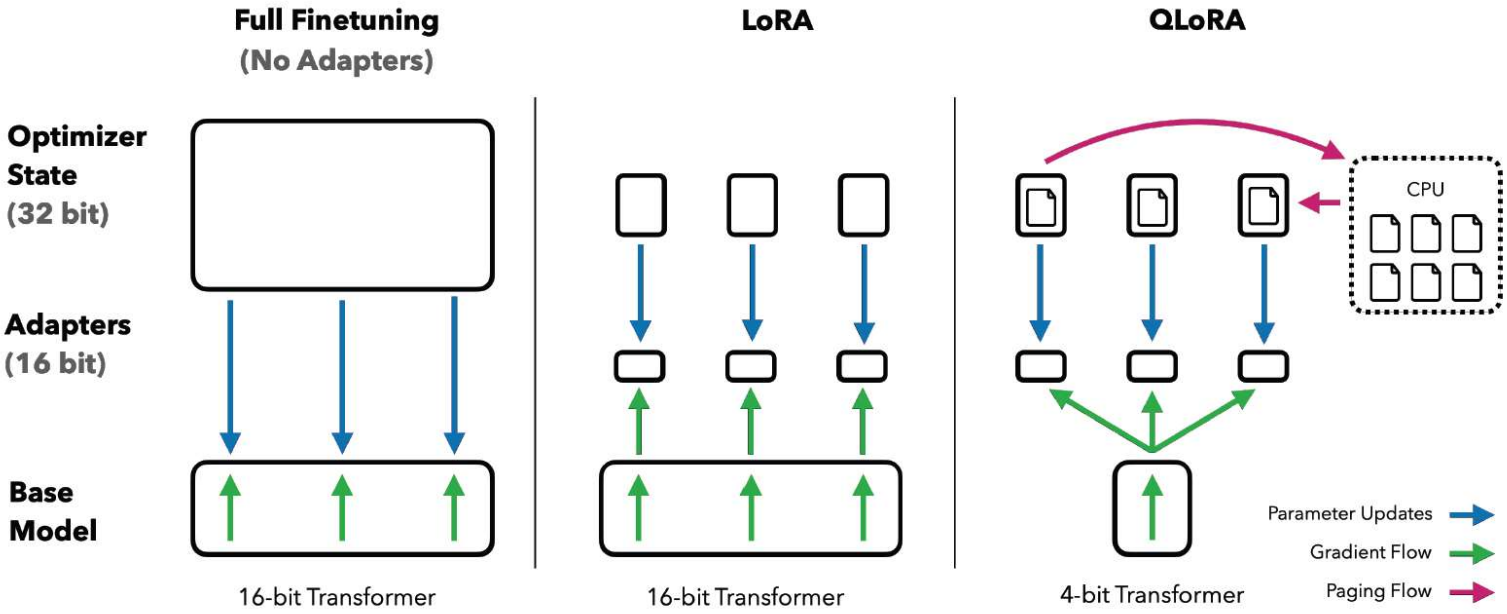
- **Quantization:** The pre-trained model weights are quantized to 4-bit (typically using NF4) and frozen. This significantly reduces memory usage.
- **Low-Rank Adapters:** Small, trainable parameters in the form of low-rank adapters are added to each layer of the model. These adapters are trained in full precision (e.g., 32-bit floating point or bfloat16) to adapt the model to a specific task.
- **Fine-tuning:** The model is fine-tuned on the target task. During computations, the quantized weights are dequantized back to full precision.
- **Final Model:** The final model consists of the original weights in their quantized 4-bit form and the additional low-rank adapters in their higher precision format.

https://doi.org/10.48550/arXiv.2305.14314

Efficient Finetuning of Quantized LLMs

Dettmers et al.

QLoRA combines LoRA with quantization techniques reduces the average memory requirements of finetuning a 65B parameter model from >780GB of GPU memory to <48GB without degrading the runtime or predictive performance compared to a 16-bit fully finetuned baseline.



https://doi.org/10.48550/arXiv.2305.14314

Efficient Finetuning of Quantized LLMs

Dettmers et al.

1. **4-bit NormalFloat**, an information theoretically optimal quantization data type for normally distributed data that yields better empirical results than 4-bit Integers and 4-bit Floats.

2. **Double Quantization**, a method that quantizes the quantization constants, saving an average of about 0.37 bits per parameter (approximately 3 GB for a 65B model).

3. **Paged Optimizers**, using NVIDIA unified memory to avoid the gradient checkpointing memory spikes that occur when processing a mini-batch with a long sequence length.

https://doi.org/10.48550/arXiv.2305.14314

Efficient Finetuning of Quantized LLMs

Dettmers et al.

MMLU accuracy for LLaMA 7-65B models finetuned with adapters on Alpaca and FLAN v2 for different data types:

	Mean 5-shot MMLU Accuracy								
LLaMA Size	7B		13B		33B		65B		Mean
Dataset	Alpaca	FLAN v2	Alpaca	FLAN v2	Alpaca	FLAN v2	Alpaca	FLAN v2	
BFloat16	38.4	45.6	47.2	50.6	57.7	60.5	61.8	62.5	53.0
Float4	37.2	44.0	47.3	50.0	55.9	58.5	61.3	63.3	52.2
NFloat4 + DQ	39.0	44.5	47.5	50.7	57.3	59.2	61.8	63.9	53.1

https://doi.org/10.48550/arXiv.2305.14314

Thank you