KIT Lecture - AI (ML) Application II

Structure and Mechanism of ChatGPT - Generative AI

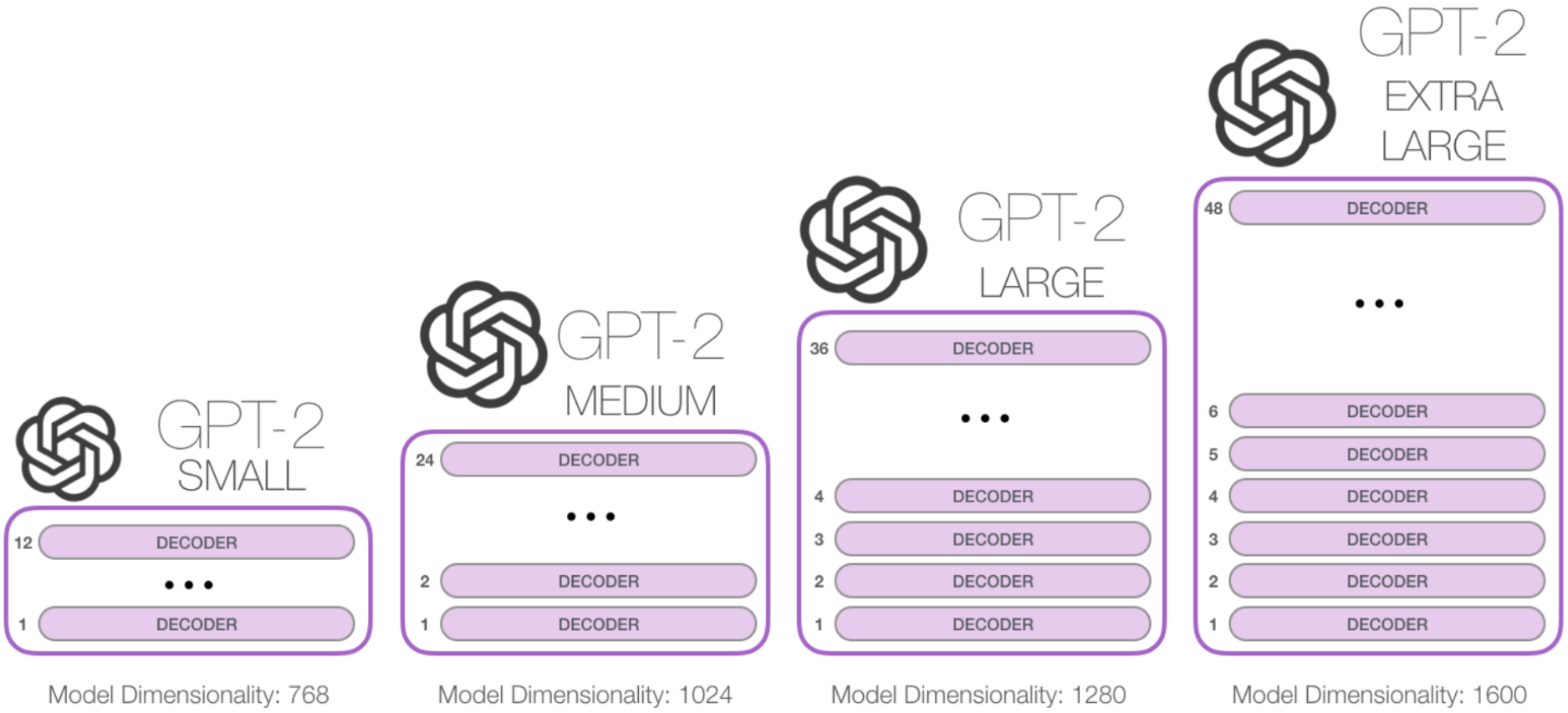
Incheon Paik
University of Aizu

Contents

- ChatGPT
  - SFT
  - RLHF
- Introduction to Transformer Reinforcement Learning
- Introduction to Reinforcement Learning (RL)
  - Q-Learning and Value-Based Deep RL
  - Policy-Based Deep RL
  - Model-Based Deep RL
  - Example of RL Code
- Code Explanation and Demonstration of TRL

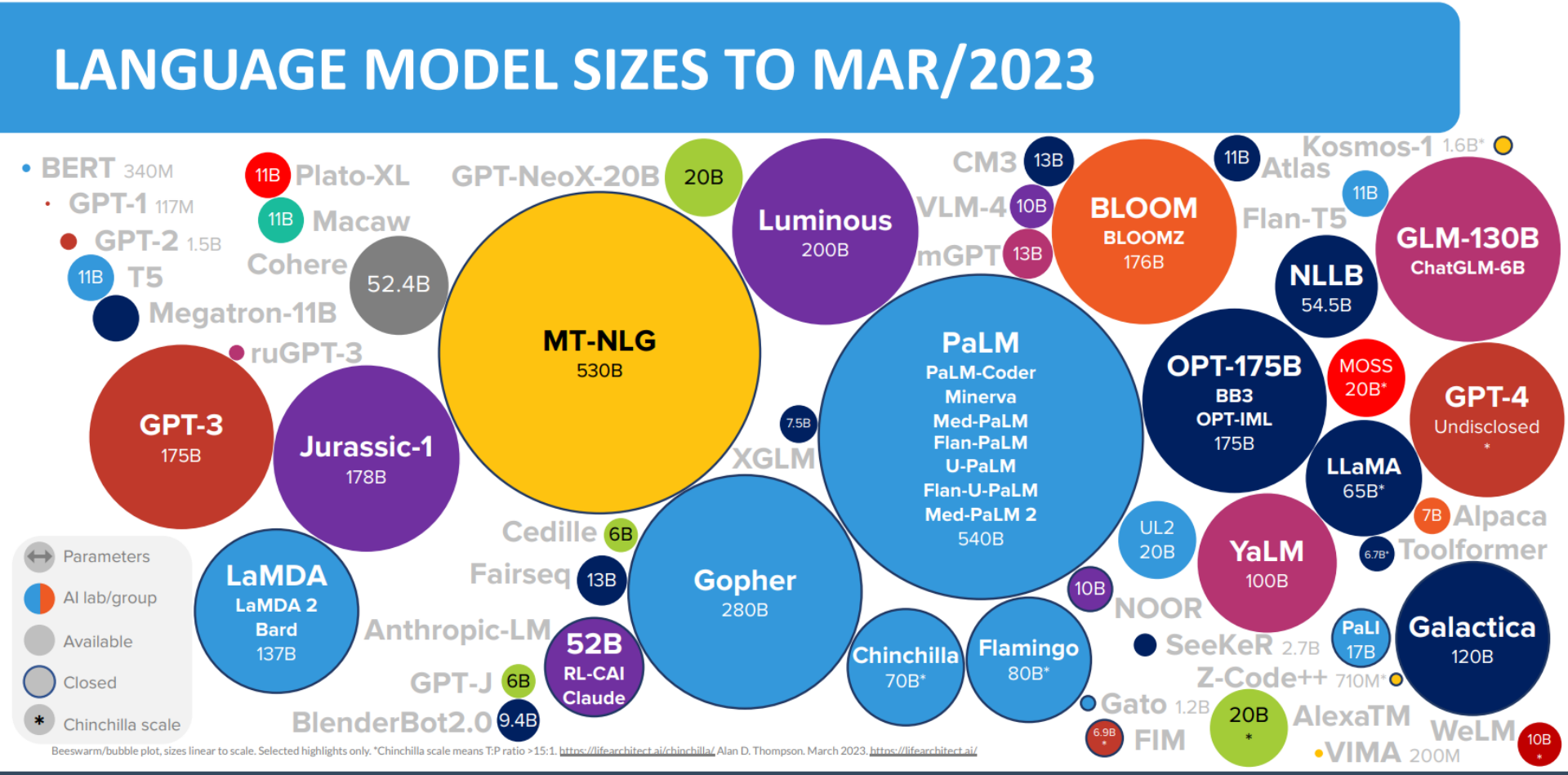
GPT-2

- Several Models of GPT-2



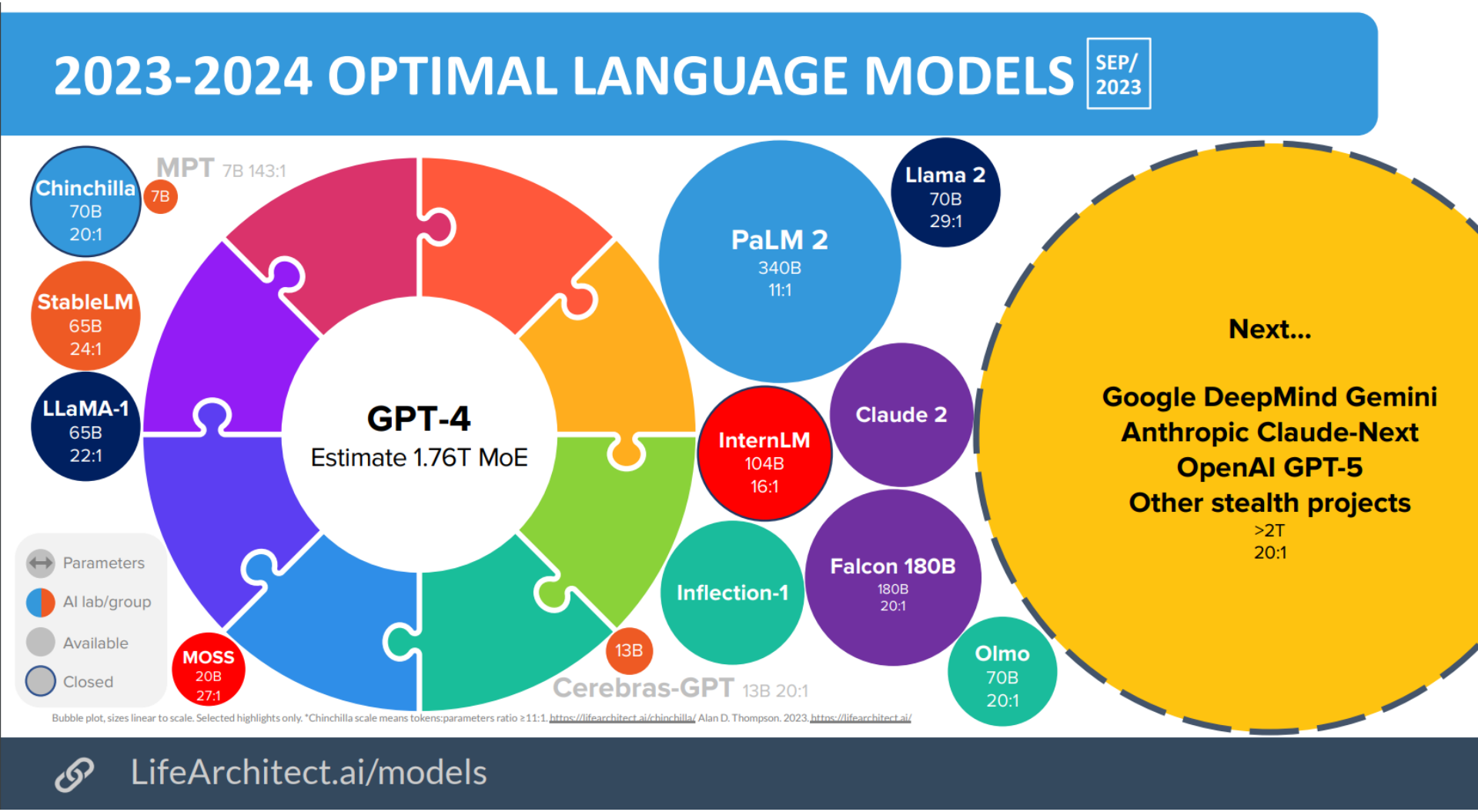
Reference: http://jalammar.github.io/illustrated-gpt2/

Size of LLM s



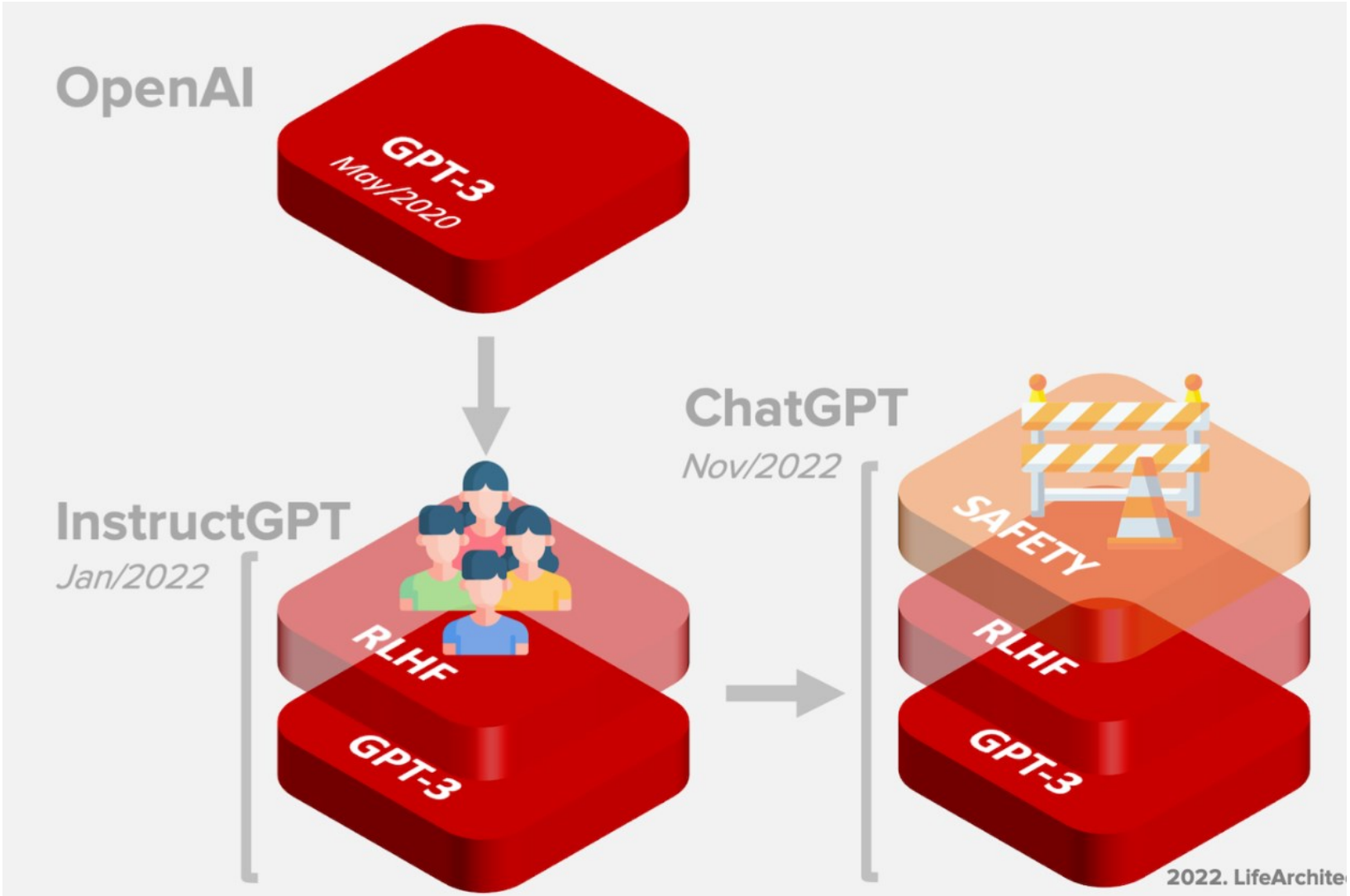
LifeArchitect.ai/models

Size of LLM s


2023-2024 OPTIMAL LANGUAGE MODELS
SEP/ 2023
Chinchilla 70B 20:1
MPT 7B 143:1
7B
StableLM 65B 24:1
LLaMA-1 65B 22:1
GPT-4
Estimate 1.76T MoE
PaLM 2 340B 11:1
Llama 2 70B 29:1
InternLM 104B 16:1
Claude 2
Inflection-1
Falcon 180B 180B 20:1
Olmo 70B 20:1
Next...
Google DeepMind Gemini
Anthropic Claude-Next
OpenAI GPT-5
Other stealth projects
>2T
20:1
MOSS 20B 27:1
13B
Cerebras-GPT 13B 20:1
Parameters
AI lab/group
Available
Closed
Bubble plot, sizes linear to scale. Selected highlights only. *Chinchilla scale means tokens:parameters ratio ≥11:1. https://lifearchitect.ai/chinchilla/ Alan D. Thompson. 2023. https://lifearchitect.ai/
LifeArchitect.ai/models

- ChatGPT

ChatGPT (GPT 3.5?)



ChatGPT: 3Steps for FT with RL

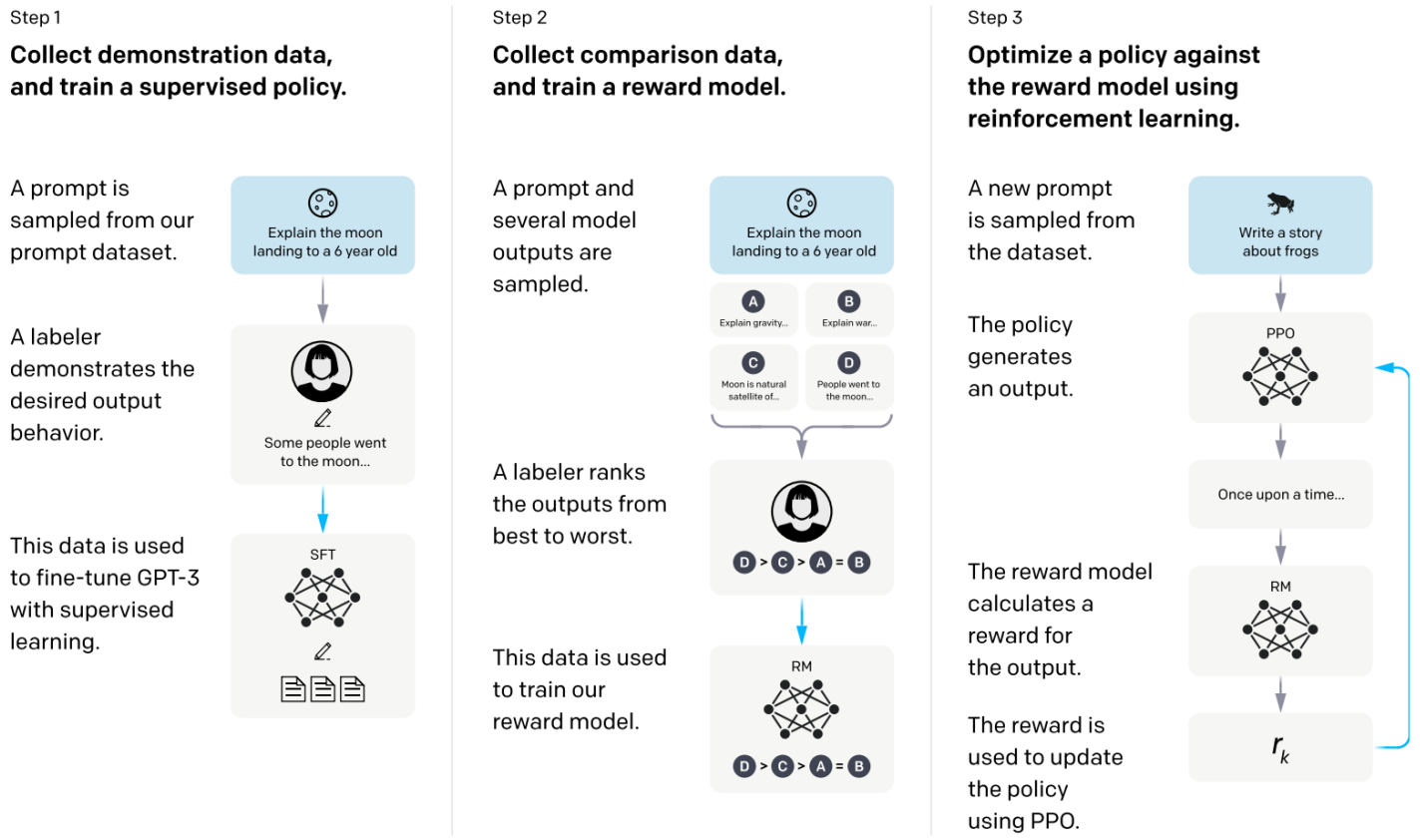
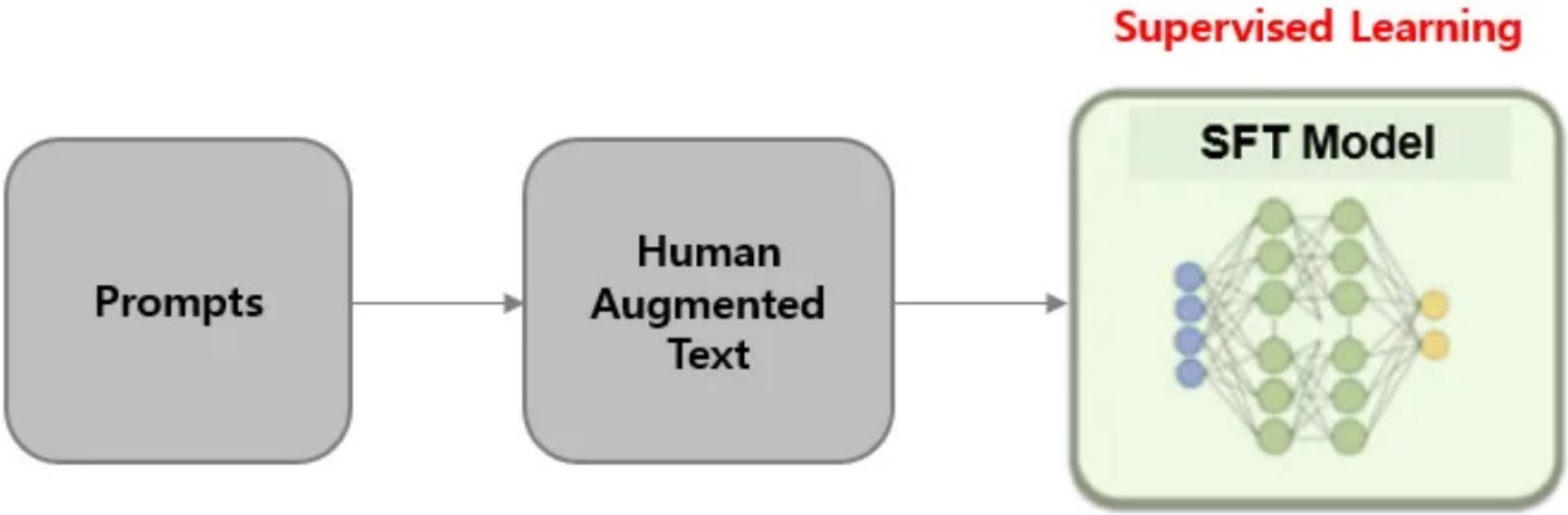


Figure 2: A diagram illustrating the three steps of our method: (1) supervised fine-tuning (SFT), (2) reward model (RM) training, and (3) reinforcement learning via proximal policy optimization (PPO) on this reward model. Blue arrows indicate that this data is used to train one of our models. In Step 2, boxes A-D are samples from our models that get ranked by labelers. See Section 3 for more details on our method.

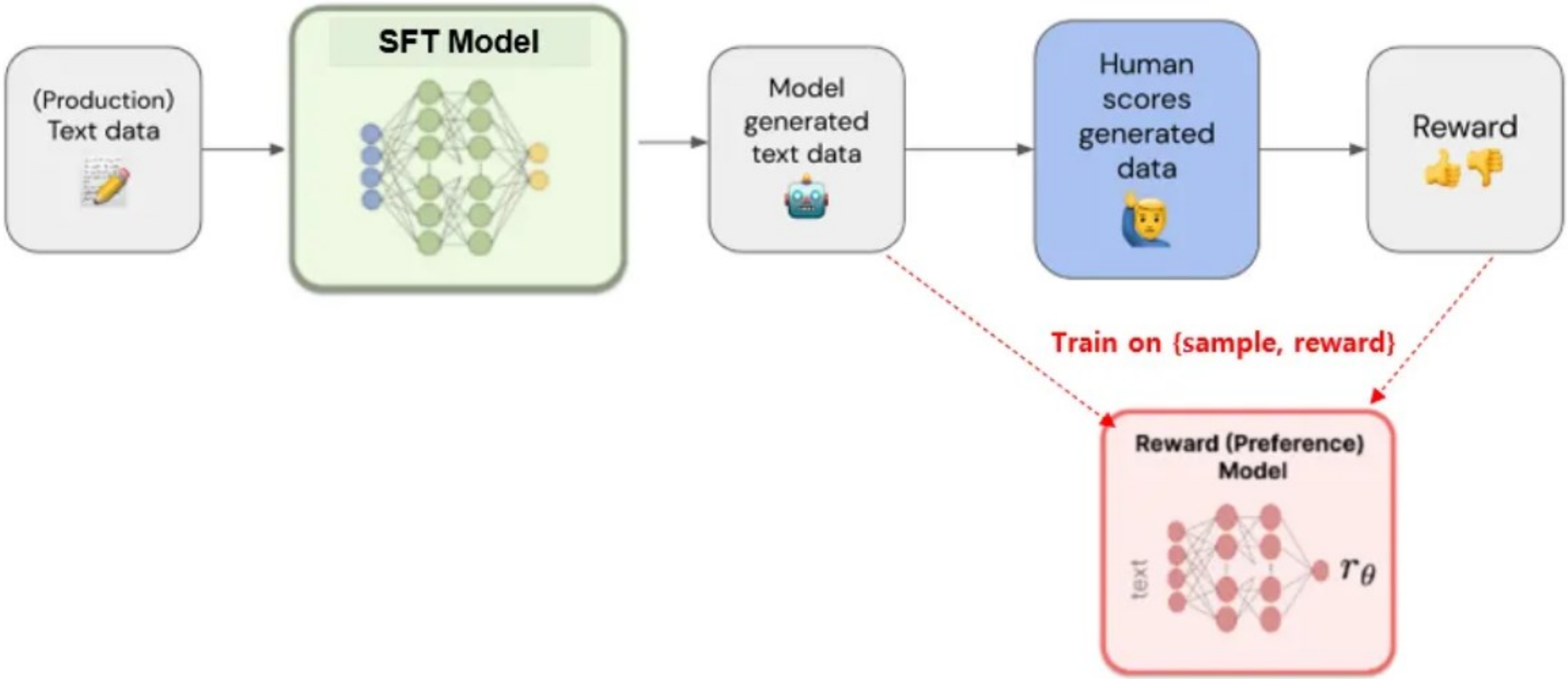
ChatGPT: 3 Steps for FT with RL

1. Supervised Fine-Tuning



ChatGPT: 3 Steps for FT with RL

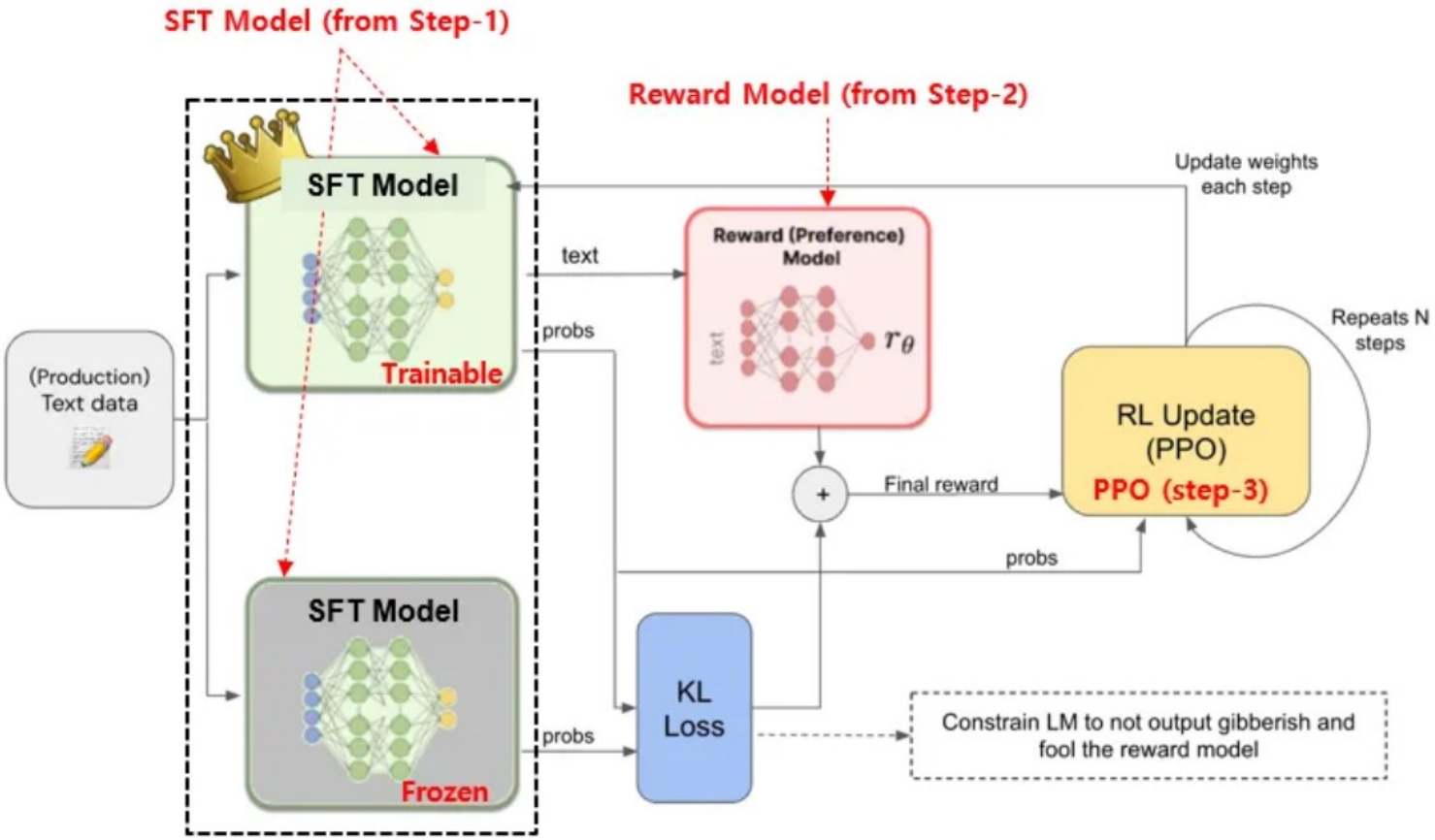
2. Reward Model (Mimic Human Preference)



ChatGPT: 3 Steps for FT with RL

3. Reinforcement Learning of SFT Model using PPO(Proximal Policy Optimization)

Fine-Tuning SFT Model using Reward Model and PPO Loss Calculation

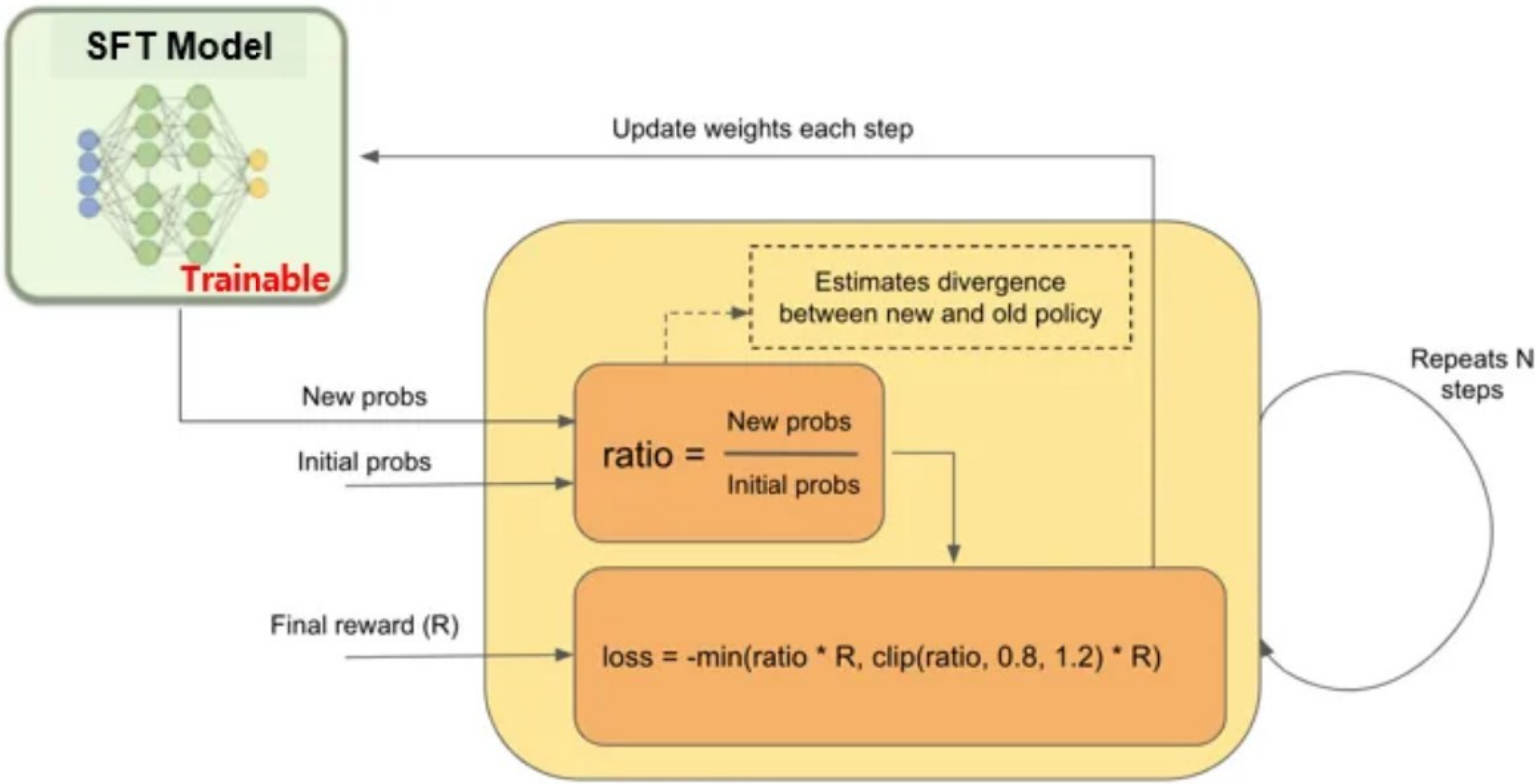


Source: Daewoo Kim, "Principle of RLHF Applied to ChatGPT", https://moon-walker.medium.com/chatgpt%EC%97%90-%EC%A0%81%EC%9A%A9%EB%90%9C-rlhf-%EC%9D%B8%EA%B0%84-%ED%94%BC%EB%93%9C%EB%B0%B1-%EA%B8%B0%EB%B0%98-%EA%B0%95%ED%99%94%ED%95%99%EC%8A%B5-%EC%9D%98-%EC%9B%90%EB%A6%AC-eb456c1b0a4a

ChatGPT: 3 Steps for FT with RL

3. Reinforcement Learning of SFT Model using PPO(Proximal Policy Optimization)

PPO (Proximal Policy Optimization) Algorithm



Source: Daewoo Kim, "Principle of RLHF Applied to ChatGPT", https://moon-walker.medium.com/chatgpt%EC%97%90-%EC%A0%81%EC%9A%A9%EB%90%9C-rlhf-%EC%9D%B8%EA%B0%84-%ED%94%BC%EB%93%9C%EB%B0%B1-%EA%B8%B0%EB%B0%98-%EA%B0%95%ED%99%94%ED%95%99%EC%8A%B5-%EC%9D%98-%EC%9B%90%EB%A6%AC-eb456c1b0a4a

ChatGPT: 3 Steps for FT with RL (details)

1. Pretraining for Completion

The result of the pretraining phase is a large language model (LLM), often known as the pretrained model. Examples include GPT-x (OpenAI), Gopher (DeepMind), LLaMa (Meta), StableLM (Stability AI).

Language model

A language model encodes statistical information about language. For simplicity, statistical information tells us how likely something (e.g. a word, a character) is to appear in a given context. The term **token** can refer to a word, a character, or a part of a word (like `-tion`), depending on the language model. You can think of tokens as the **vocabulary** that a language model uses.

Fluent speakers of a language subconsciously have statistical knowledge of that language. For example, given the context `My favorite color is __`, if you speak English, you know that the word in the blank is much more likely to be `green` than `car`.

Similarly, language models should also be able to fill in that blank. You can think of a language model as a "*completion machine*": given a text (prompt), it can generate a response to complete that text. Here's an example:

- **Prompt (from user):** `I tried so hard, and got so far`
- **Completion (from language model):** `But in the end, it doesn't even matter.`

Source: Chip Huyen, "RLHF: Reinforcement Learning from Human Feedback", https://huyenchip.com/2023/05/02/rlhf.html (Slide 10 ~23)

ChatGPT: 3 Steps for FT with RL (details)

1. Pretraining for Completion (Mathematical Formulation)

- ML task: language modeling
- Training data: low-quality data
- Data scale: usually in the order of trillions of tokens as of May 2023.
  - GPT-3's dataset (OpenAI): 0.5 trillion tokens. I can't find any public info for GPT-4, but I'd estimate it to use an order of magnitude more data than GPT-3.
  - Gopher's dataset (DeepMind): 1 trillion tokens
  - RedPajama (Together): 1.2 trillion tokens
  - LLaMa's dataset (Meta): 1.4 trillion tokens
- Model resulting from this process: LLM

- LLM_ϕ: the language model being trained, parameterized by ϕ. The goal is to find ϕ for which the cross entropy loss is minimized.
- $[T_1, T_2, \ldots, T_V]$: vocabulary – the set of all unique tokens in the training data.
- V: the vocabulary size.
- $f(x)$: function mapping a token to its position in the vocab. If x is T_k in the vocab, $f(x) = k$.
- Given the sequence $(x_1, x_2, \ldots, x_n)$, we'll have n training samples:
  - Input: $x = (x_1, x_2, \ldots, x_{i-1})$
  - Ground truth: x_i
- For each training sample (x, x_i):
  - Let $k = f(x_i)$
  - Model's output: $LLM(x) = [\bar{y}_1, \bar{y}_2, \ldots, \bar{y}_V]$. Note: $\sum_j \bar{y}_j = 1$
  - The loss value: $CE(x, x_i; \phi) = -\log \bar{y}_k$
- Goal: find ϕ to minimize the expected loss on all training samples. $CE(\phi) = -E_x \log \bar{y}_k$

ChatGPT: 3 Steps for FT with RL (details)

2. SFT for Dialogue – Why SFT

Pretraining optimizes for completion. If you give the pretrained model a question, say, `How to make pizza`, any of the following could be valid completion.

1. Adding more context to the question: `for a family of six`
2. Adding follow-up questions: `? What ingredients do I need? How much time would it take?`
3. Actually giving the answer

The third option is preferred if you're looking for an answer. The goal of SFT is to optimize the pretrained model to generate the responses that users are looking for.

How to do that? We know that a model mimics its training data. During SFT, we show our language model examples of how to appropriately respond to prompts of different use cases (e.g. question answering, summarization, translation). The examples follow the format (prompt, response) and are called demonstration data. OpenAI calls supervised finetuning *behavior cloning*: you demonstrate how the model should behave, and the model clones this behavior.

Table 1: Distribution of use case categories from our API prompt dataset.

Use-case	(%)
Generation	45.6%
Open QA	12.4%
Brainstorming	11.2%
Chat	8.4%
Rewrite	6.6%
Summarization	4.2%
Classification	3.5%
Other	3.5%
Closed QA	2.6%
Extract	1.9%

Table 2: Illustrative prompts from our API prompt dataset. These are fictional examples inspired by real usage—see more examples in Appendix A.2.1.

Use-case	Prompt
Brainstorming	List five ideas for how to regain enthusiasm for my career
Generation	Write a short story where a bear goes to the beach, makes friends with a seal, and then returns home.
Rewrite	This is the summary of a Broadway play: """ {summary} """ This is the outline of the commercial for that play: """

ChatGPT: 3 Steps for FT with RL (details)

2. SFT for Dialogue – Demonstration Data

Demonstration data

Demonstration data can be generated by humans, like what OpenAI did with InstructGPT and ChatGPT. Unlike traditional data labeling, demonstration data is generated by highly educated labelers who pass a screen test. Among those who labeled demonstration data for InstructGPT, ~90% have at least a college degree and more than one-third have a master's degree.

What is your highest attained level of education?	
Less than high school degree	0%
High school degree	10.5%
Undergraduate degree	52.6%
Master's degree	36.8%
Doctorate degree	0%

OpenAI's 40 labelers created around 13,000 (prompt, response) pairs for InstructGPT. Here are a few examples:

Prompt	Response
Serendipity means the occurrence and development of events by chance in a happy or beneficial way. Use the word in a sentence.	Running into Margaret and being introduced to Tom was a fortunate stroke of serendipity.
ELI5: What's the cause of the "anxiety lump" in our chest during stressful or disheartening experiences?	The anxiety lump in your throat is caused by muscular tension keeping your glottis dilated to maximize airflow. The clenched chest or heartache feeling is caused by the vagus nerve which tells the organs to pump blood faster, stop digesting, and produce adrenaline and cortisol.
Create a shopping list from this recipe: Trim the ends off zucchini. Cut zucchini in half lengthwise; scoop out pulp, leaving 1/2-in. shells. Finely chop pulp. In a skillet, cook beef, zucchini pulp, onion, mushrooms and peppers over medium heat until meat is no longer pink; drain. Remove from the heat. Add 1/2 cup cheese, ketchup, salt and pepper; mix well. Spoon into the zucchini shells. Place in a greased 13x9-in. baking dish. Sprinkle with remaining cheese.	Zucchini, beef, onion, mushroom, peppers, cheese, ketchup, salt, pepper

ChatGPT: 3 Steps for FT with RL (details)

3. RLHF

»»Side note: Hypotheses on why RLHF works««

> *Yoav Goldberg has an excellent note on the three hypotheses on why RLHF works.*

- ***The diversity hypothesis**: during SFT, the model's output is expected to somewhat match the demonstrated responses. For example, given the prompt "what's an example of a language?", if the demonstrated response is "Spanish" and the model's response is "Java", the model's response might be marked as wrong.*
- ***The negative feedback hypothesis**: demonstration only gives the model positive signals (e.g. only showing the model good responses), not negative signals (e.g. showing models what bad responses look like). RL allows us to show models negative signals.*
- ***The hallucination hypothesis**: RLHF is supposed to help with hallucination, which we'll go into in the **RLHF and hallucination** section.*

ChatGPT: 3 Steps for FT with RL (details)

3. RLHF – Reward Model

There might be some variations, but here's the core idea.

- Training data: high-quality data in the format of (prompt, winning_response, losing_response)
- Data scale: 100K - 1M examples
  - InstructGPT: 50,000 prompts. Each prompt has 4 to 9 responses, forming between 6 and 36 pairs of (winning_response, losing_response). This means between 300K and 1.8M training examples in the format of (prompt, winning_response, losing_response).
  - Constitutional AI, which is suspected to be the backbone of Claude (Anthropic): 318K comparisons – 135K generated by humans, and 183K generated by AI. Anthropic has an older version of their data open-sourced (hh-rlhf), which consists of roughly 170K comparisons.

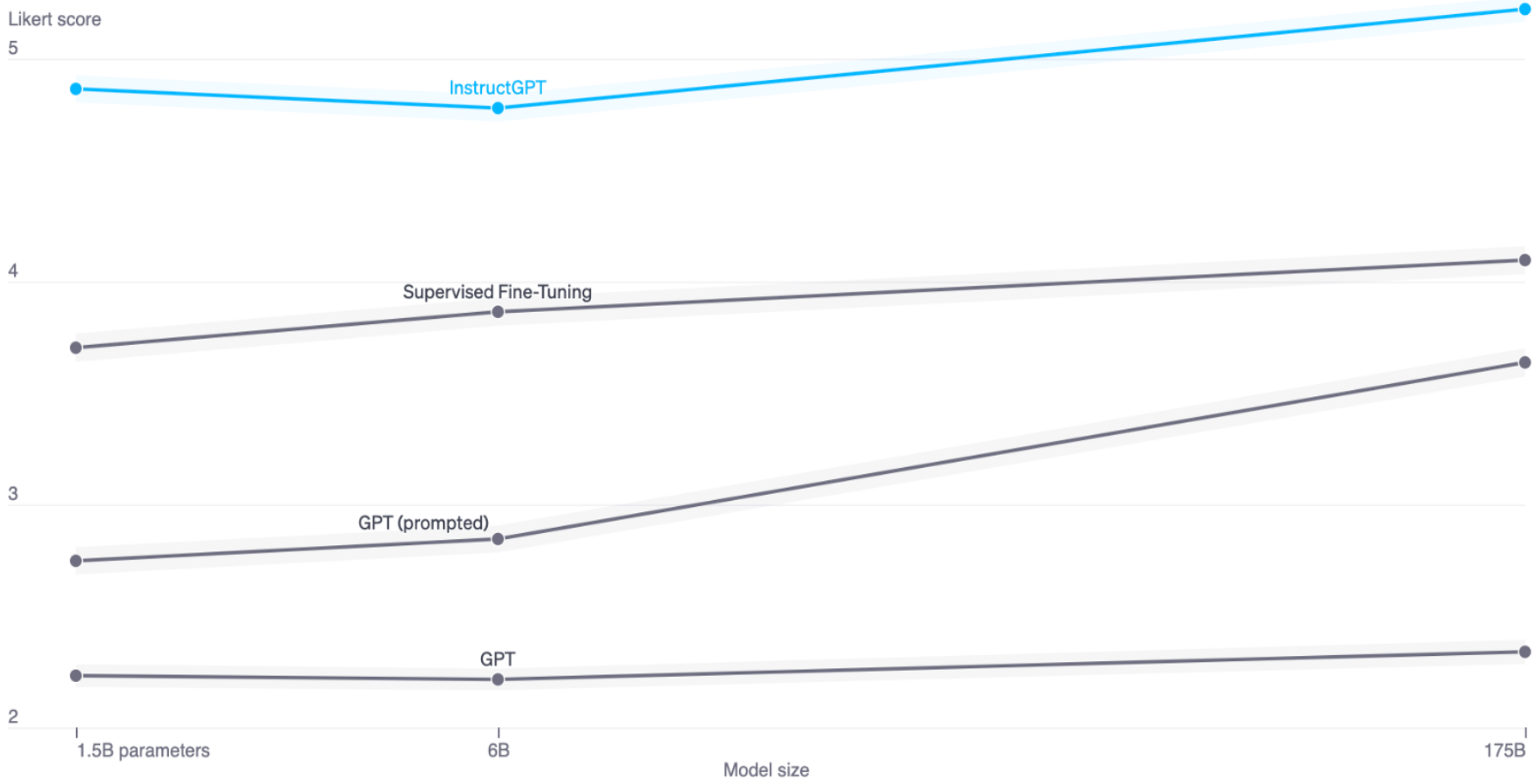
- r_θ: the reward model being trained, parameterized by θ. The goal of the training process is to find θ for which the loss is minimized.
- Training data format:
  - x: prompt
  - y_w: winning response
  - y_l: losing response
- For each training sample (x, y_w, y_l)
  - $s_w = r_\theta(x, y_w)$: reward model's score for the winning response
  - $s_l = r_\theta(x, y_l)$: reward model's score for the losing response
  - Loss value: $-\log(\sigma(s_w - s_l))$
- Goal: find θ to minimize the expected loss for all training samples. $-E_x \log(\sigma(s_w - s_l))$

To get more intuition how this loss function works, let's visualize it.

Let $d = s_w - s_l$. Here's the graph for $f(d) = -\log(\sigma(d))$. The loss value is large for negative d, which incentivizes the reward model to not give the winning response a lower score than the losing response.

ChatGPT: Performance Comparison

GPT/GPT(prompted)/SFT/InstructGPT

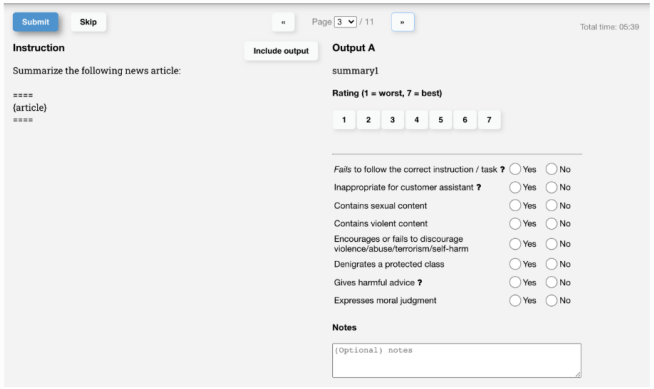


ChatGPT: 3 Steps for FT with RL (details)

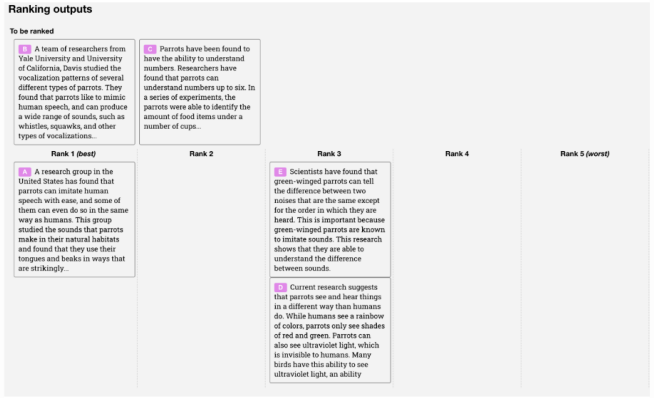
3. RLHF – Reward Model. UI to collect comparison data

UI to collect comparison data

Below is a screenshot of the UI that OpenAI's labelers used to create training data for InstructGPT's RM. Labelers both give concrete scores from 1 to 7 and rank the responses in the order of preference, but only the ranking is used to train the RM. **Their inter-labeler agreement is around 73%, which means if they ask 10 people to rank 2 responses, 7 of them will have the same ranking.**



(a)



(b)

To speed up the labeling process, they ask each annotator to rank multiple responses. 4 ranked responses, e.g. A > B > C > D, will produce 6 ranked pairs, e.g. (A > B), (A > C), (A > D), (B > C), (B > D), (C > D).

ChatGPT: 3Steps for FT with RL

Fine-Tuning Reward Model

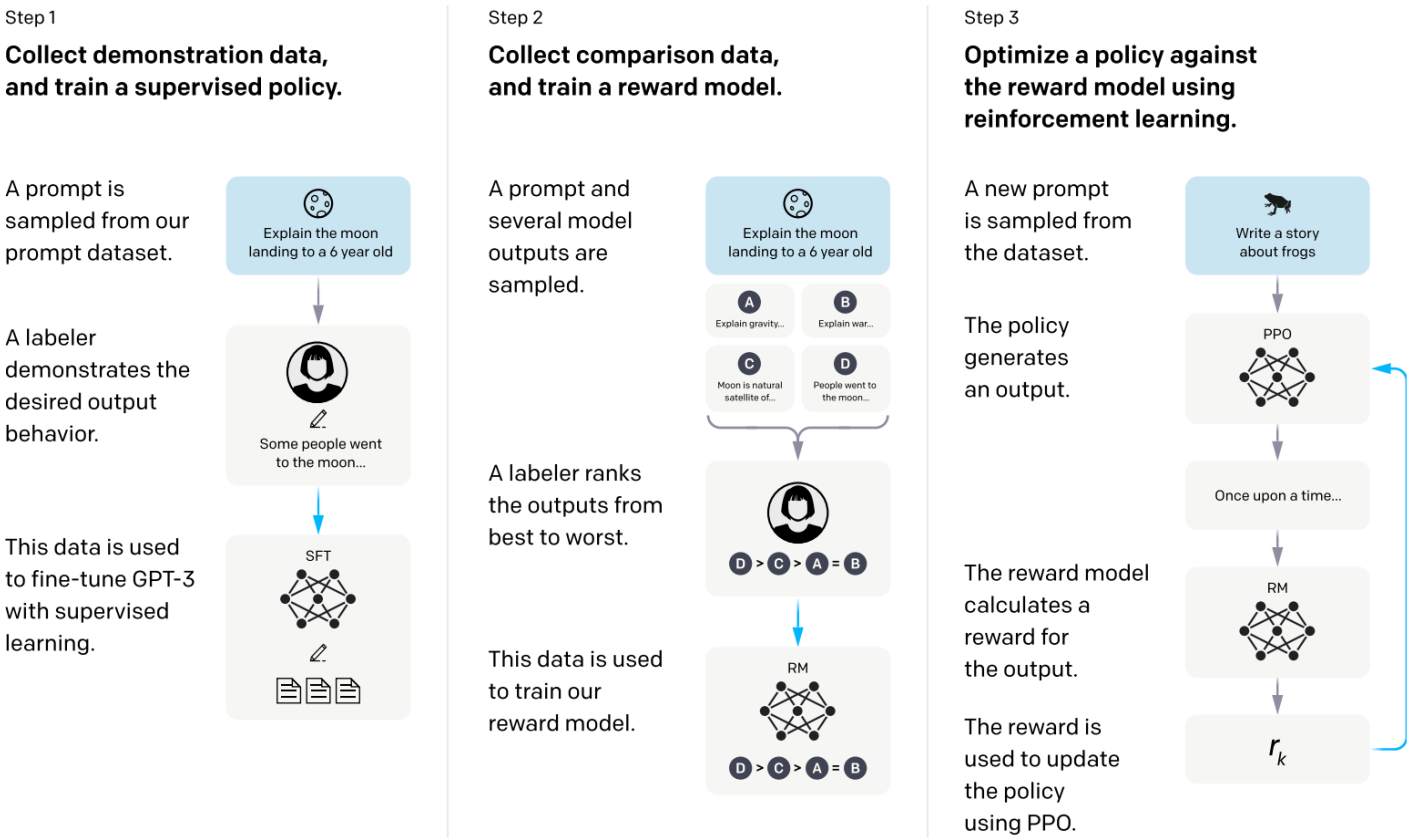


Figure 2: A diagram illustrating the three steps of our method: (1) supervised fine-tuning (SFT), (2) reward model (RM) training, and (3) reinforcement learning via proximal policy optimization (PPO) on this reward model. Blue arrows indicate that this data is used to train one of our models. In Step 2, boxes A-D are samples from our models that get ranked by labelers. See Section 3 for more details on our method.

ChatGPT: 3 Steps for FT with RL (details)

- ML task: reinforcement learning
  - Action space: the vocabulary of tokens the LLM uses. Taking action means choosing a token to generate.
  - Observation space: the distribution over all possible prompts.
  - Policy: the probability distribution over all actions to take (aka all tokens to generate) given an observation (aka a prompt). An LLM constitutes a policy because it dictates how likely a token is to be generated next.
  - Reward function: the reward model.
- Training data: randomly selected prompts
- Data scale: 10,000 - 100,000 prompts
  - InstructGPT: 40,000 prompts

3. RLHF – Mathematical Notation

- RM: the reward model obtained from phase 3.1.
- LLM^{SFT}: the supervised finetuned model obtained from phase 2.
  - Given a prompt x, it outputs a distribution of responses.
  - In the InstructGPT paper, LLM^{SFT} is represented as π^{SFT}.
- LLM_ϕ^{RL}: the model being trained with reinforcement learning, parameterized by ϕ.
  - The goal is to find ϕ to maximize the score according to the RM.
  - Given a prompt x, it outputs a distribution of responses.
  - In the InstructGPT paper, LLM_ϕ^{RL} is represented as π_ϕ^{RL}.
- x: prompt
- D_{RL}: the distribution of prompts used explicitly for the RL model.
- $D_{pretrain}$: the distribution of the training data for the pretrain model.

ChatGPT: 3 Steps for FT with RL (details)

3. RLHF – Mathematical Notation

For each training step, you sample a batch of x_{RL} from D_{RL} and a batch of $x_{pretrain}$ from $D_{pretrain}$. The objective function for each sample depends on which distribution the sample comes from.

1. For each x_{RL}, we use LLM_ϕ^{RL} to sample a response: $y \sim LLM_\phi^{RL}(x_{RL})$. The objective is computed as follows. Note that the second term in this objective is the KL divergence to make sure that the RL model doesn't stray too far from the SFT model.

$$\text{objective}_1(x_{RL}, y; \phi) = RM(x_{RL}, y) - \beta \log \frac{LLM_\phi^{RL}(y|x)}{LLM^{SFT}(y|x)}$$

2. For each $x_{pretrain}$, the objective is computed as follows. Intuitively, this objective is to make sure that the RL model doesn't perform worse on text completion – the task the pretrained model was optimized for.

$$\text{objective}_2(x_{pretrain}; \phi) = \gamma \log LLM_\phi^{RL}(x_{pretrain})$$

The final objective is the sum of the expectation of two objectives above. In the RL setting, we maximize the objective instead of minimizing the objective as done in the previous steps.

$$\text{objective}(\phi) = E_{x \sim D_{RL}} E_{y \sim LLM_\phi^{RL}(x)} [RM(x, y) - \beta \log \frac{LLM_\phi^{RL}(y|x)}{LLM^{SFT}(y|x)}] + \gamma E_{x \sim D_{pretrain}} \log LLM_\phi^{RL}(x)$$

Note:

The notation used is slightly different from the notation used in the InstructGPT paper, as I find the notation here a bit more explicit, but they both refer to the exact same objective function.

$$\begin{aligned}\text{objective}(\phi) =& E_{(x,y) \sim D_{\pi_\phi^{\text{RL}}}} \left[r_\theta(x, y) - \beta \log \left(\pi_\phi^{\text{RL}}(y \mid x) / \pi^{\text{SFT}}(y \mid x) \right) \right] + \\ & \gamma E_{x \sim D_{\text{pretrain}}} \left[\log(\pi_\phi^{\text{RL}}(x)) \right]\end{aligned} \tag{2}$$

The objective function as written in the InstructGPT paper.

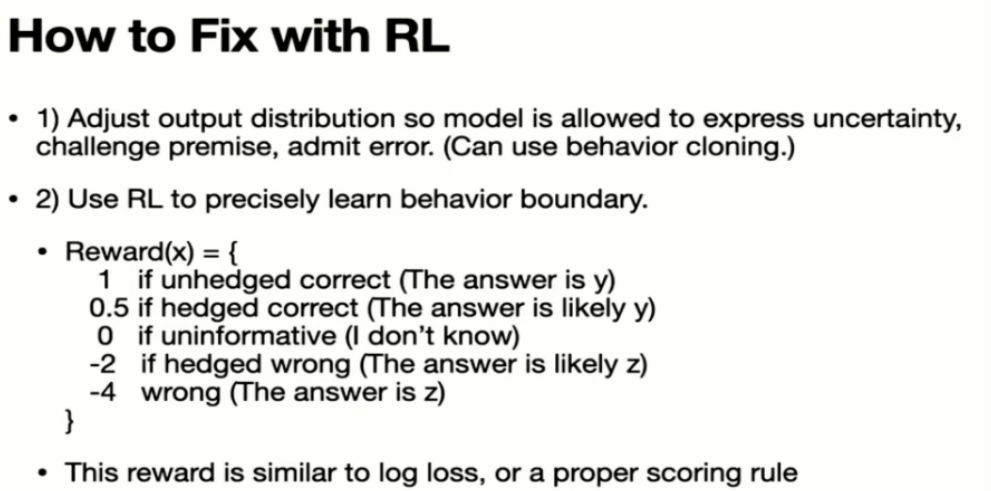
ChatGPT: RLHF and Hallucination

About Hallucination

Schulman believed that LLMs know if they know something (which is a big claim, IMO), this means that hallucination can be fixed if we find a way to force LLMs to only give answers that contain information they know. He then proposed a couple of solutions.

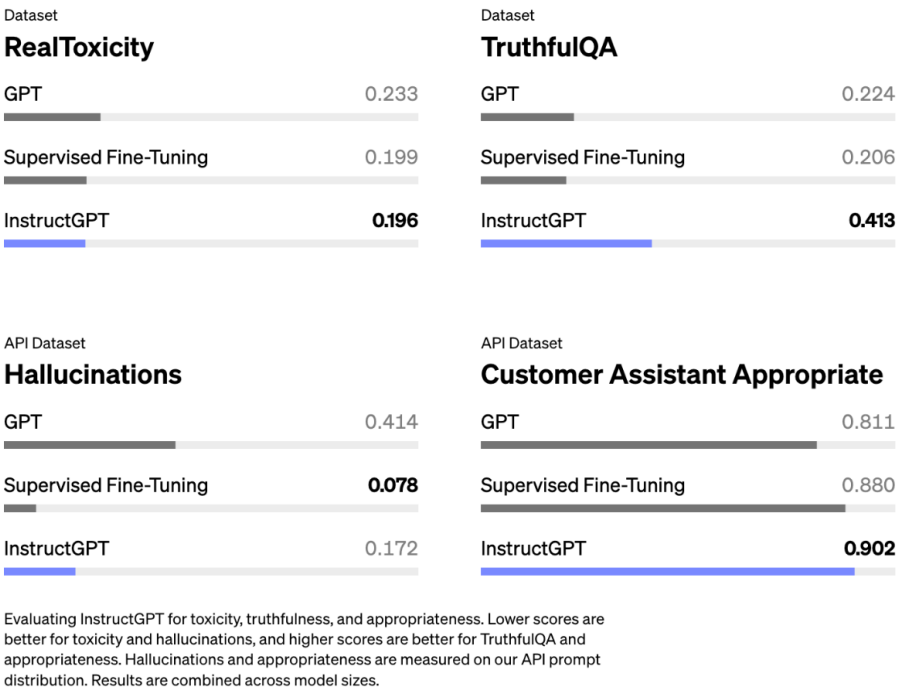
1. Verification: asking the LLM to explain (retrieve) the sources where it gets the answer from.
2. RL. Remember that the reward model in phase 3.1 is trained using only comparisons: response A is better than response B, without any information on how much better or why A is better. Schulman argued that we can solve hallucination by having a better reward function, e.g. punishing a model more for making things up.

Here's a screenshot from John Schulman's talk in April 2023.



ChatGPT: RLHF and Hallucination

From Schulman's talk, I got the impression that RLHF is supposed to help with hallucination. However, the InstructGPT paper shows that RLHF actually made hallucination worse. Even though RLHF caused worse hallucination, it improved other aspects, and overall, human labelers prefer RLHF model over SFT alone model.



Hallucination is worse for InstructGPT (RLHF + SFT) compared to just SFT (Ouyang et al., 2022)

Based on the assumption that LLMs know what they know, some people try to reduce hallucination with prompts, e.g. adding `Answer as truthfully as possible, and if you're unsure of the answer, say "Sorry, I don't know"`. Making LLMs respond concisely also seems to help with hallucination – the fewer tokens LLMs have to generate, the less chance they have to make things up.

Source: Chip Huyen, "RLHF: Reinforcement Learning from Human Feedback", https://huyenchip.com/2023/05/02/rlhf.html (Slide 10 ~23)

Implementation Case: Transformer Reinforcement Learning

Step 1: **SFTTrainer**
Train your model on your favorite dataset

```
from trl import SFTTrainer

trainer = SFTTrainer(
    "facebook/opt-350m",
    train_dataset=dataset,
    dataset_text_field="text",
    max_seq_length=512,
)

trainer.train()
```

Step 2: **RewardTrainer**
Train a preference model on a comparison data to rank generations from the supervised fine-tuned (SFT) model

```
from trl import RewardTrainer

trainer = RewardTrainer(
    model=model,
    args=training_args,
    tokenizer=tokenizer,
    train_dataset=dataset,
)

trainer.train()
```

Step 3: **PPOTrainer**
Further optimize the SFT model using the rewards from the reward model and PPO algorithm

```
from trl import PPOConfig, PPOTrainer

trainer = PPOTrainer(
    config,
    model,
    tokenizer=tokenizer,
)

for query in dataloader:
  response = model.generate(query)
  reward = reward_model(response)
  trainer.step(query, response, reward)
```

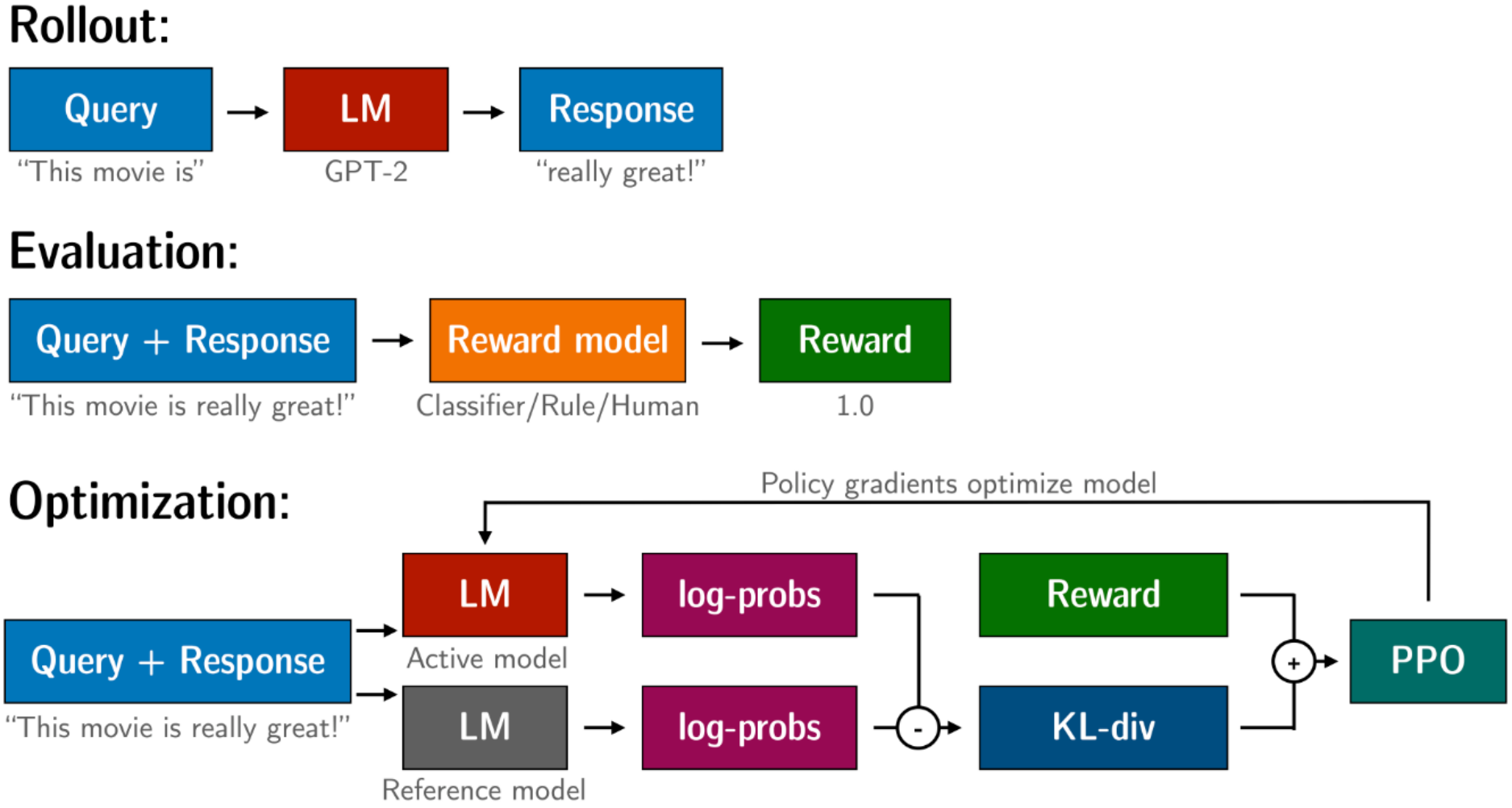
Source: https://github.com/huggingface/trl

Implementation Case: Transformer Reinforcement Learning

Fine-tuning a language model via PPO consists of roughly three steps:

1. **Rollout**: The language model generates a response or continuation based on query which could be the start of a sentence.
2. **Evaluation**: The query and response are evaluated with a function, model, human feedback or some combination of them. The important thing is that this process should yield a scalar value for each query/response pair.
3. **Optimization**: This is the most complex part. In the optimisation step the query/response pairs are used to calculate the log-probabilities of the tokens in the sequences. This is done with the model that is trained and and a reference model, which is usually the pre-trained model before fine-tuning. The KL-divergence between the two outputs is used as an additional reward signal to make sure the generated responses don't deviate to far from the reference language model. The active language model is then trained with PPO.

Implementation Case: Transformer Reinforcement Learning



- Reinforcement Learning

Reinforcement Learning (RL)

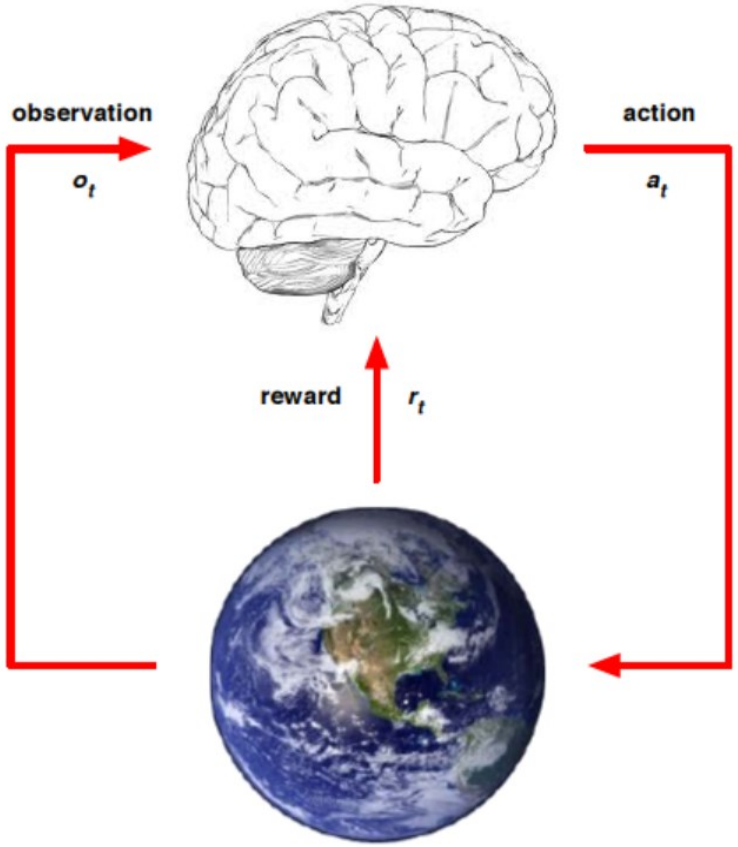
RL is a general-purpose framework for decision-making

- RL is for an agent with the capacity to act
- Each action influences the agent's future state
- Success is measured by a scalar reward signal
- Goal: select actions to maximise future reward

Deep Reinforcement Learning

- We seek a single agent which can solve any human-level task.
- RL defines the objective.
- DL gives the mechanism.
- RL + DL = general intelligence

Agent and Environment



- At each step t the agent:
  - Executes action a_t
  - Receives observation o_t
  - Receives scalar reward r_t
- The environment:
  - Receives action a_t
  - Emits observation o_{t+1}
  - Emits scalar reward r_{t+1}

State

- Experience is a sequence of observations, actions, rewards

$$o_1, r_1, a_1, ..., a_{t-1}, o_t, r_t$$

- The state is a summary of experience

$$s_t = f(o_1, r_1, a_1, ..., a_{t-1}, o_t, r_t)$$

- In a fully observed environment

$$s_t = f(o_t)$$

Major Components of an RL Agent

- An RL agent may include one or more of these components:
  - Policy: agent's behavior function
  - Value function: how good is each state and/or action
  - Model: Agent's representation of the environment

Policy

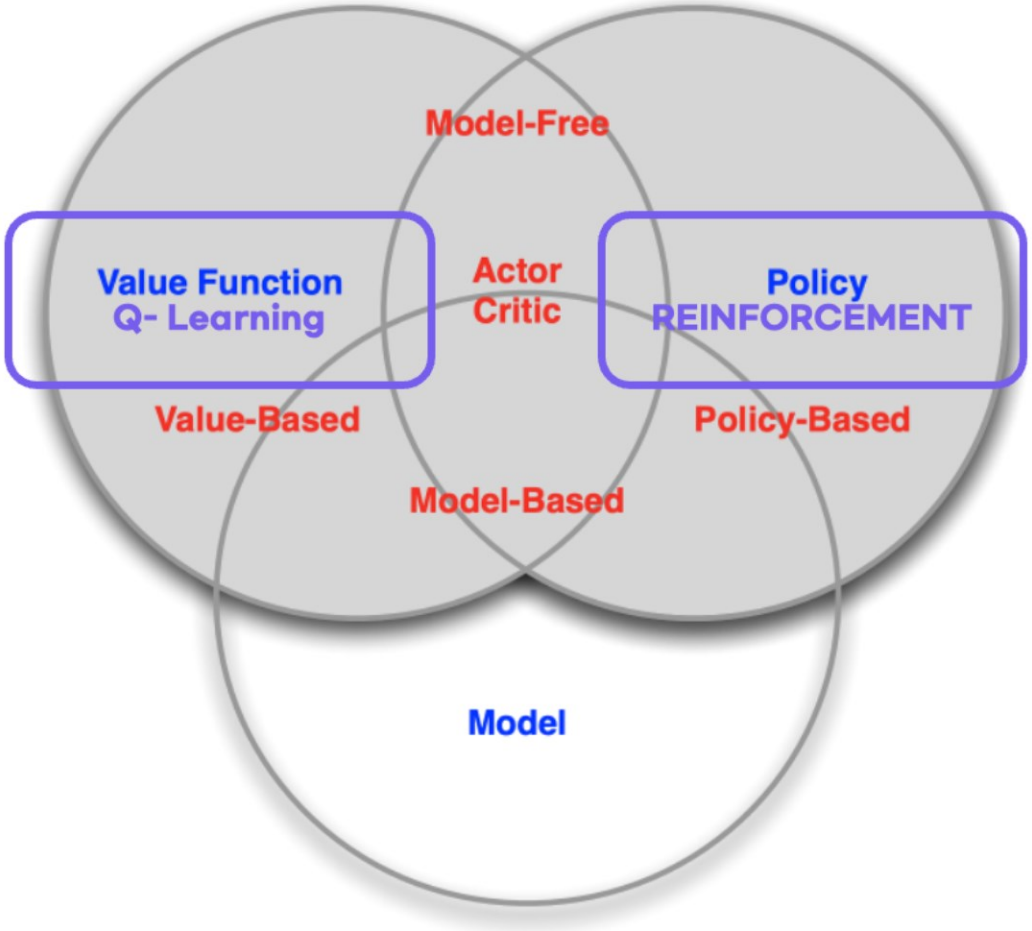
- A policy is the agent's behaviour
- It is a map from state to action:
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a|s) = \mathbb{P}[a|s]$

RL-Algorithms

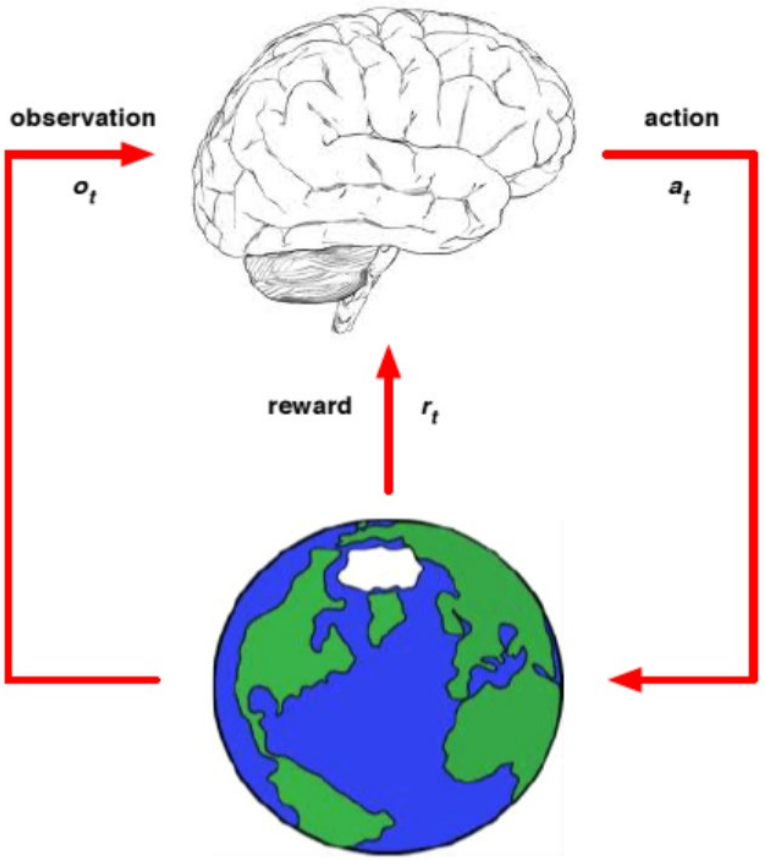
Policy Gradient Learning

REINFORCE

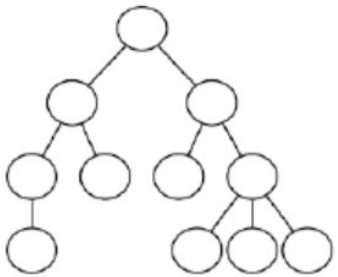
REward **I**ncrement = **N**onnegative **F**actor **O**ffset **R**einforcement **C**haracteristic **E**ligibility



Model



- Model is learnt from experience
- Acts as proxy for environment
- Planner interacts with model
- e.g. using lookahead search



Approaches to Reinforcement Learning

Value-based RL

- Estimate the optimal value function $Q^*(s, a)$
- This is the maximum value achievable under any policy

Policy-based RL

- Search directly for the optimal policy π^*
- This is the policy achieving maximum future reward

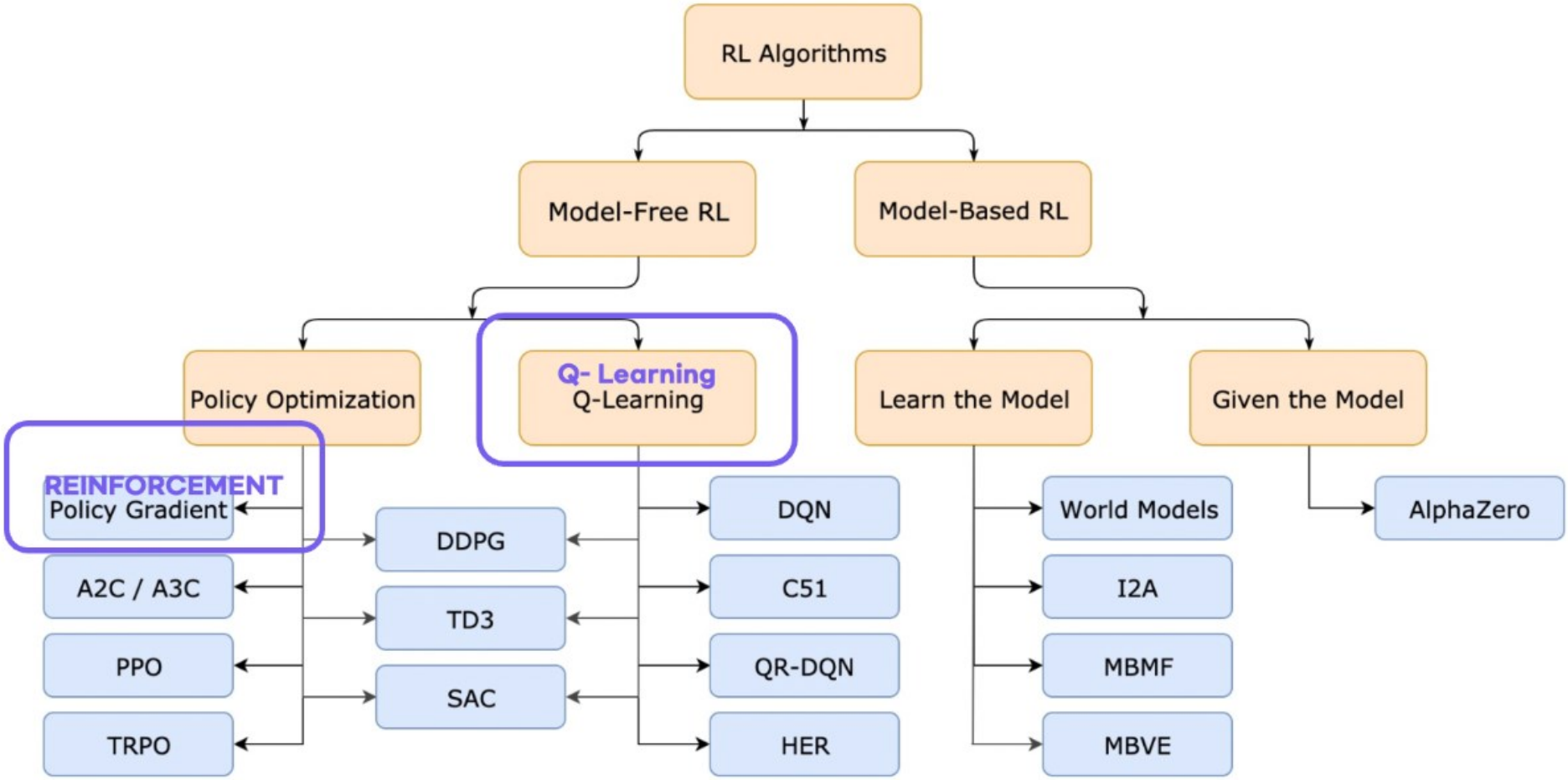
Model-based RL

- Build a model of the environment
- Plan (e.g. by lookahead) using model

Deep Reinforcement Learning

- Use deep neural networks to represent
  - Value function
  - Policy
  - Model
- Optimize loss function by stochastic gradient descent

RL-Algorithms



- Value-based RL

Value Function

- A value function is a prediction of future reward
  - "How much reward will I get from action a in state s?"
- Q-value function gives expected total reward
  - from state s and action a
  - under policy π
  - with discount factor γ

$$Q^{\pi}(s,a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s, a\right]$$

- Value functions decompose into a Bellman equation

$$Q^{\pi}(s,a) = \mathbb{E}_{s',a'}\left[r + \gamma Q^{\pi}(s',a') \mid s, a\right]$$

Optimal Value Function

- An optimal value function is the maximum achievable value

$$Q^*(s,a) = \max_\pi Q^\pi(s,a) = Q^{\pi^*}(s,a)$$

- Once we have Q^* we can act optimally,

$$\pi^*(s) = \operatorname*{argmax}_a Q^*(s,a)$$

- Optimal value maximises over all decisions. Informally:

$$\begin{aligned} Q^*(s,a) &= r_{t+1} + \gamma \max_{a_{t+1}} r_{t+2} + \gamma^2 \max_{a_{t+2}} r_{t+3} + \ldots \\ &= r_{t+1} + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}) \end{aligned}$$

- Formally, optimal values decompose into a Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^*(s',a') \mid s,a \right]$$

Q Learning

- A model-free reinforcement learning technique. Specifically, Q-learning can be used to find an optimal action-selection policy for any given (finite) Markov decision process (MDP).
- It works by learning an action-value function that ultimately gives the expected utility of taking a given action in a given state and following the optimal policy thereafter.
- A policy is a rule that the agent follows in selecting actions, given the state it is in. When such an action-value function is learned, the optimal policy can be constructed by simply selecting the action with the highest value in each state.
- One of the strengths of Q-learning is that it is able to compare the expected utility of the available actions without requiring a model of the environment.
- Q-learning can handle problems with stochastic transitions and rewards, without requiring any adaptations. It has been proven that for any finite MDP, Q-learning eventually finds an optimal policy, in the sense that the expected value of the total reward return over all successive steps, starting from the current state, is the maximum achievable.

Q Learning

The algorithm therefore has a function that calculates the Quality of a state-action combination:

$$Q : S \times A \to \mathbb{R} \ .$$

Before learning has started, Q returns an (arbitrary) fixed value, chosen by the designer. Then, at each time *t* the agent selects an action at and observes a reward r_t and a new state s_{t+1} that may depend on both the previous state s_t and the selected action, Q is updated. The core of the algorithm is a simple value iteration update, using the weighted average of the old value and the new information:

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

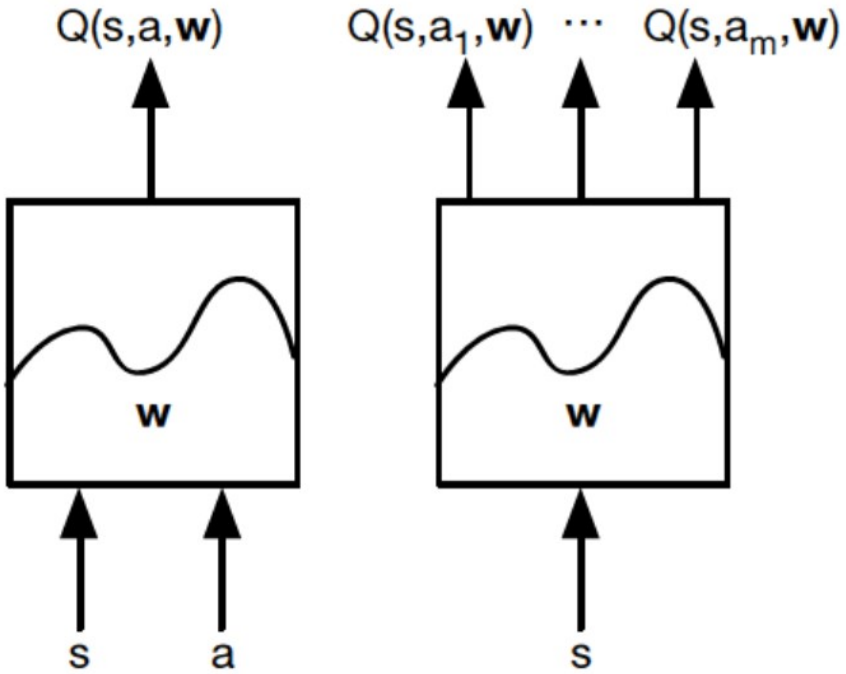
where r_t is the reward observed for the current state s_t, and α is the learning rate (0< α <= 1).

- Value-based Deep RL

Q-Network

Represent value function by Q-network with weights **w**

$$Q(s, a, \mathbf{w}) \approx Q^*(s, a)$$



Q-Learning

- Optimal Q-values should obey Bellman equation

$$Q^*(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q(s',a')^* \mid s,a\right]$$

- Treat right-hand side $r + \gamma \max_{a'} Q(s',a',\mathbf{w})$ as a target
- Minimise MSE loss by stochastic gradient descent

$$I = \left(r + \gamma \max_{a} Q(s',a',\mathbf{w}) - Q(s,a,\mathbf{w})\right)^2$$

- Converges to Q^* using table lookup representation
- But diverges using neural networks due to:
  - Correlations between samples
  - Non-stationary targets

Deep Q-Network(DQN): Experience Replay

To remove correlations, build data-set from agent's own experience

$s_t, a_t, r_{t+1}, s_{t+1} \rightarrow$

s_1, a_1, r_2, s_2
s_2, a_2, r_3, s_3
s_3, a_3, r_4, s_4
...
$s_t, a_t, r_{t+1}, s_{t+1}$

$\rightarrow \quad s, a, r, s'$

Sample experiences from data-set and apply update

$$I = \left(r + \gamma \max_{a'} Q(s', a', \mathbf{w}^-) - Q(s, a, \mathbf{w}) \right)^2$$

To deal with non-stationarity, target parameters $\mathbf{w}^-$ are held fixed

- Policy-based Deep RL

Deep Policy Networks

- Represent policy by deep network with weights $\mathbf{u}$

$$a = \pi(a|s, \mathbf{u}) \text{ or } a = \pi(s, \mathbf{u})$$

- Define objective function as total discounted reward

$$L(\mathbf{u}) = \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \ldots \mid \pi(\cdot, \mathbf{u})\right]$$

- Optimise objective end-to-end by SGD
- i.e. Adjust policy parameters $\mathbf{u}$ to achieve more reward

Policy Gradients

How to make high-value actions more likely:

- The gradient of a stochastic policy $\pi(a|s, \mathbf{u})$ is given by

$$\frac{\partial L(\mathbf{u})}{\partial u} = \mathbb{E}\left[\frac{\partial \log \pi(a|s, \mathbf{u})}{\partial \mathbf{u}} Q^{\pi}(s, a)\right]$$

- The gradient of a deterministic policy $a = \pi(s)$ is given by

$$\frac{\partial L(\mathbf{u})}{\partial \mathbf{u}} = \mathbb{E}\left[\frac{\partial Q^{\pi}(s, a)}{\partial a} \frac{\partial a}{\partial \mathbf{u}}\right]$$

- if a is continuous and Q is differentiable

Policy Gradients

- The policy gradient has many equivalent forms

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ v_t\right] && \text{REINFORCE} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ Q^w(s,a)\right] && \text{Q Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ A^w(s,a)\right] && \text{Advantage Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta\right] && \text{TD Actor-Critic} \\
&= \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta e\right] && \text{TD}(\lambda)\text{ Actor-Critic} \\
G_\theta^{-1}\nabla_\theta J(\theta) &= w && \text{Natural Actor-Critic}
\end{aligned}
$$

- Each leads a stochastic gradient ascent algorithm
- Critic uses policy evaluation (e.g. MC or TD learning) to estimate $Q^\pi(s,a)$, $A^\pi(s,a)$ or $V^\pi(s)$

Policy Gradients

- For the true value function $V^{\pi_\theta}(s)$, the TD error δ^{π_θ}

$$\delta^{\pi_\theta} = r + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s)$$

- is an unbiased estimate of the advantage function

$$\begin{aligned}\mathbb{E}_{\pi_\theta}\left[\delta^{\pi_\theta}|s,a\right] &= \mathbb{E}_{\pi_\theta}\left[r + \gamma V^{\pi_\theta}(s')|s,a\right] - V^{\pi_\theta}(s) \\ &= Q^{\pi_\theta}(s,a) - V^{\pi_\theta}(s) \\ &= A^{\pi_\theta}(s,a)\end{aligned}$$

- So we can use the TD error to compute the policy gradient

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a)\ \delta^{\pi_\theta}\right]$$

- In practice we can use an approximate TD error

$$\delta_v = r + \gamma V_v(s') - V_v(s)$$

Policy Gradients

Goal: $\nabla_\theta \log \pi_\theta(s,a) r = r \times \nabla_\theta \log \pi_\theta(s,a)$

Before Derivation $r \times \log \pi_\theta(s,a)$

For Loss $-r \times \log \pi_\theta(s,a)$

Add “-” for maximization with the loss

Actor-Critic Algorithm

- Estimate value function $Q(s, a, \mathbf{w}) \approx Q^{\pi}(s, a)$
- Update policy parameters $\mathbf{u}$ by stochastic gradient ascent

$$\frac{\partial l}{\partial \mathbf{u}} = \frac{\partial \log \pi(a|s, \mathbf{u})}{\partial \mathbf{u}} Q(s, a, \mathbf{w})$$

or

$$\frac{\partial l}{\partial \mathbf{u}} = \frac{\partial Q(s, a, \mathbf{w})}{\partial a} \frac{\partial a}{\partial \mathbf{u}}$$

Asynchronous Advantage Actor-Critic (A3C)

- Estimate state-value function

$$V(s, \mathbf{v}) \approx \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + ... | s\right]$$

- Q-value estimated by an n-step sample

$$q_t = r_{t+1} + \gamma r_{t+2} ... + \gamma^{n-1} r_{t+n} + \gamma^n V(s_{t+n}, \mathbf{v})$$

- Actor is updated towards target

$$\frac{\partial l_u}{\partial \mathbf{u}} = \frac{\partial \log \pi(a_t | s_t, \mathbf{u})}{\partial \mathbf{u}} (q_t - V(s_t, \mathbf{v}))$$

- Critic is updated to minimise MSE w.r.t. target

$$l_v = (q_t - V(s_t, \mathbf{v}))^2$$

- $4x$ mean Atari score vs Nature DQN

Proximal Policy Optimization (PPO) Algorithm

- Characteristics of PPO

1. Rewriting the consumed data (Data reuse)

2. We want to maximize the step without loss of the performance using the first order derivation.

3. Rather than reflecting the results after an episode ends, incorporating them into the learning on a step-by-step basis

4. Using Importance Sampling, importance is calculated as the ratio = (recent log P / old log P) [higher ratio indicates higher importance]. This ratio is used to adjust values and create upper and lower bounds. In PPO, it extends the reflection to the next future. Additionally, there are two neural networks, Actor and Critic, in which the Critic network evaluates whether the actions are performed correctly, complementing the performance.

- Bigger ratio, Larger Effect.
- Larger the recent log P, smaller old log P --> Larger effect.

PPO-Algorithms

- Policy gradients

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \right]$$

- Can differentiate the following loss

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t \mid s_t) \hat{A}_t \right].$$

but don't want to optimize it too far

- Equivalently differentiate

$$L^{IS}_{\theta_{\text{old}}}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right].$$

at $\theta = \theta_{\text{old}}$, state-actions are sampled using θ_{old}. (IS = importance sampling)

Just the chain rule: $\nabla_\theta \log f(\theta)\big|_{\theta_{\text{old}}} = \frac{\nabla_\theta f(\theta)\big|_{\theta_{\text{old}}}}{f(\theta_{\text{old}})} = \nabla_\theta \left(\frac{f(\theta)}{f(\theta_{\text{old}})} \right)\Big|_{\theta_{\text{old}}}$

PPO-Algorithms

$$U(\theta) = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{P(\tau|\theta)}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

$$\nabla_\theta U(\theta) = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{\nabla_\theta P(\tau|\theta)}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

$$\nabla_\theta \, U(\theta)|_{\theta=\theta_{\text{old}}} = \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\frac{\nabla_\theta \, P(\tau|\theta)|_{\theta_{\text{old}}}}{P(\tau|\theta_{\text{old}})} R(\tau) \right]$$

$$= \mathbb{E}_{\tau \sim \theta_{\text{old}}} \left[\nabla_\theta \log P(\tau|\theta)|_{\theta_{\text{old}}} \, R(\tau) \right]$$

PPO-Algorithms

Importance Sampling

- Estimate the expectation of a different distribution

$$\begin{aligned}\mathbb{E}_{X\sim P}[f(X)] &= \sum P(X)f(X) \\ &= \sum Q(X)\frac{P(X)}{Q(X)}f(X) \\ &= \mathbb{E}_{X\sim Q}\left[\frac{P(X)}{Q(X)}f(X)\right]\end{aligned}$$

$$\sum_a \pi_\theta(a|s_n)A_{\theta_{\text{old}}}(s_n,a) = \mathbb{E}_{a\sim q}\left[\frac{\pi_\theta(a|s_n)}{q(a|s_n)}A_{\theta_{\text{old}}}(s_n,a)\right]$$

q is sampling distribution.

PPO-Algorithms

TRPO

- **Constrained Form**

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t[\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta. \end{aligned}$$

- **Penalized Form**

$$\underset{\theta}{\text{maximize}} \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t - \beta \, \mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)] \right]$$

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \quad \text{so} \quad r(\theta_{\text{old}}) = 1.$$

PPO-Algorithms

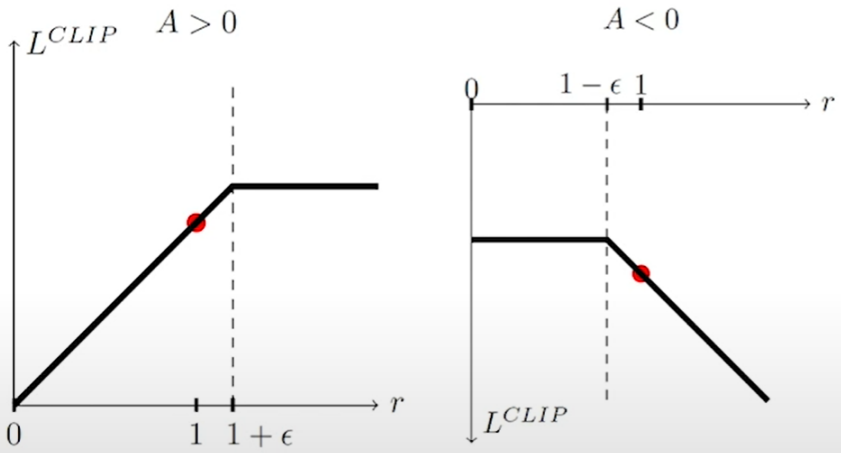
Method1 Clipped Surrogate Objective

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

Original Loss

Clipped Loss

Lower bound of unclipped objective = L^{CPI}



PPO-Algorithms

Algorithm – PPO Clipped

for iteration=$1, 2, \ldots$ **do**
  Run policy for T timesteps or N trajectories
  Estimate advantage function at all timesteps
  Do SGD on $L^{CLIP}(\theta)$ objective for some number of epochs
end for

PPO-Algorithms

Method2 Adaptive KL Penalty

- Using several epochs of minibatch SGD, optimize the KL-penalized objective

$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t - \beta\,\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]\right]$$

- Compute $d = \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]]$
  - If $d < d_{\text{targ}}/1.5$, $\beta \leftarrow \beta/2$
  - If $d > d_{\text{targ}} \times 1.5$, $\beta \leftarrow \beta \times 2$

- Pseudocode:

for iteration=1, 2, . . . **do**
  Run policy for T timesteps or N trajectories
  Estimate advantage function at all timesteps
  Do SGD on above objective for some number of epochs
  If KL too high, increase β. If KL too low, decrease β.
end for

PPO-Algorithms

1. Surrogate Objectives Experiment

algorithm	avg. normalized score
No clipping or penalty	-0.39
Clipping, $\epsilon = 0.1$	0.76
Clipping, $\epsilon = 0.2$	**0.82**
Clipping, $\epsilon = 0.3$	0.70
Adaptive KL $d_{\text{targ}} = 0.003$	0.68
Adaptive KL $d_{\text{targ}} = 0.01$	0.74
Adaptive KL $d_{\text{targ}} = 0.03$	0.71
Fixed KL, $\beta = 0.3$	0.62
Fixed KL, $\beta = 1.$	0.71
Fixed KL, $\beta = 3.$	0.72
Fixed KL, $\beta = 10.$	0.69

No clipping or penalty: $L_t(\theta) = r_t(\theta)\hat{A}_t$

Clipping: $L_t(\theta) = \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta)), 1-\epsilon, 1+\epsilon)\hat{A}_t$

KL penalty (fixed or adaptive) $L_t(\theta) = r_t(\theta)\hat{A}_t - \beta\,\text{KL}[\pi_{\theta_{\text{old}}}, \pi_\theta]$

Example code for Q-Learning

```
def choose_action(state, q_table):
    # This is how to choose an action
    state_actions = q_table.iloc[state, :]
    if (np.random.uniform() > EPSILON) or (state_actions.all() == 0):
        action_name = np.random.choice(ACTIONS)
    else:   # act greedy
        action_name = state_actions.argmax()
    return action_name


def get_env_feedback(S, A):
    # This is how agent will interact with the environment
    if A == 'right':    # move right
        if S == N_STATES - 2:   # terminate
            S_ = 'terminal'
            R = 1
        else:
            S_ = S + 1
            R = 0
    else:   # move left
        R = 0
        if S == 0:
            S_ = S  # reach the wall
        else:
            S_ = S - 1
    return S_, R
```

Example Code

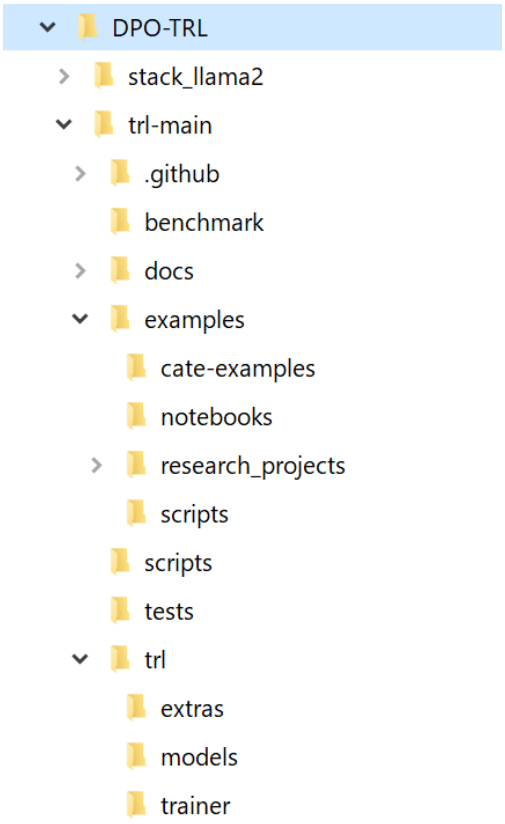
```
def rl():
    # main part of RL loop
    q_table = build_q_table(N_STATES, ACTIONS)
    for episode in range(MAX_EPISODES):
        step_counter = 0
        S = 0
        is_terminated = False
        update_env(S, episode, step_counter)
        while not is_terminated:

            A = choose_action(S, q_table)
            S_, R = get_env_feedback(S, A)  # take action & get n
            q_predict = q_table.ix[S, A]
            if S_ != 'terminal':
                q_target = R + GAMMA * q_table.iloc[S_, :].max()
            else:
                q_target = R     # next state is terminal
                is_terminated = True    # terminate this episode

            q_table.ix[S, A] += ALPHA * (q_target - q_predict)  #
            S = S_  # move to next state

            update_env(S, episode, step_counter+1)
            step_counter += 1
    return q_table
```

Code Examples of RL Algorithms

- REINFORCE Algorithm
  - https://github.com/seungeunrho/minimalRL/blob/master/REINFORCE.py
- A3C Algorithm
  - https://github.com/seungeunrho/minimalRL/blob/master/a3c.py
- Actor-Critic Algorithm
  - https://github.com/seungeunrho/minimalRL/blob/master/actor_critic.py
- PPO Algorithm
  - https://github.com/seungeunrho/minimalRL/blob/master/ppo.py

Code for TRL

- TRL - Transformer Reinforcement Learning
  - https://github.com/huggingface/trl



Explanation of TRL Code

- Explanation of TRL Code Details and Demonstration!

Thanks!