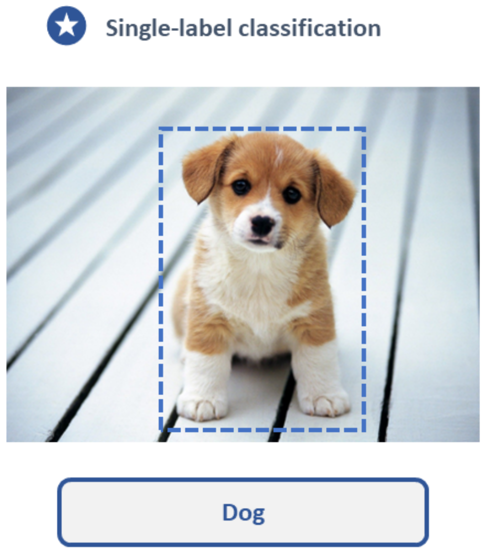
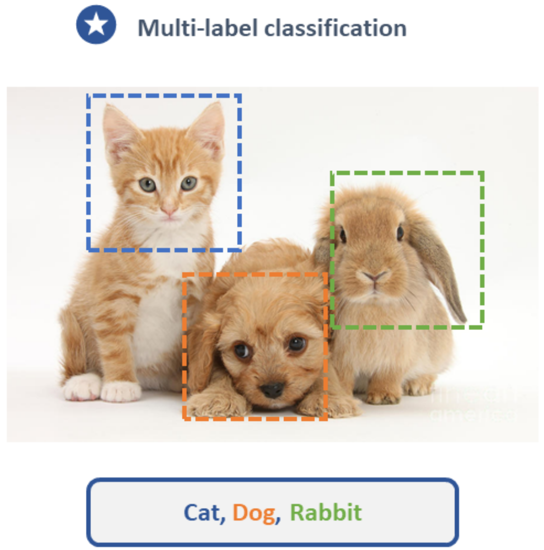
Deep Reinforcement Learning

Supervised Learning

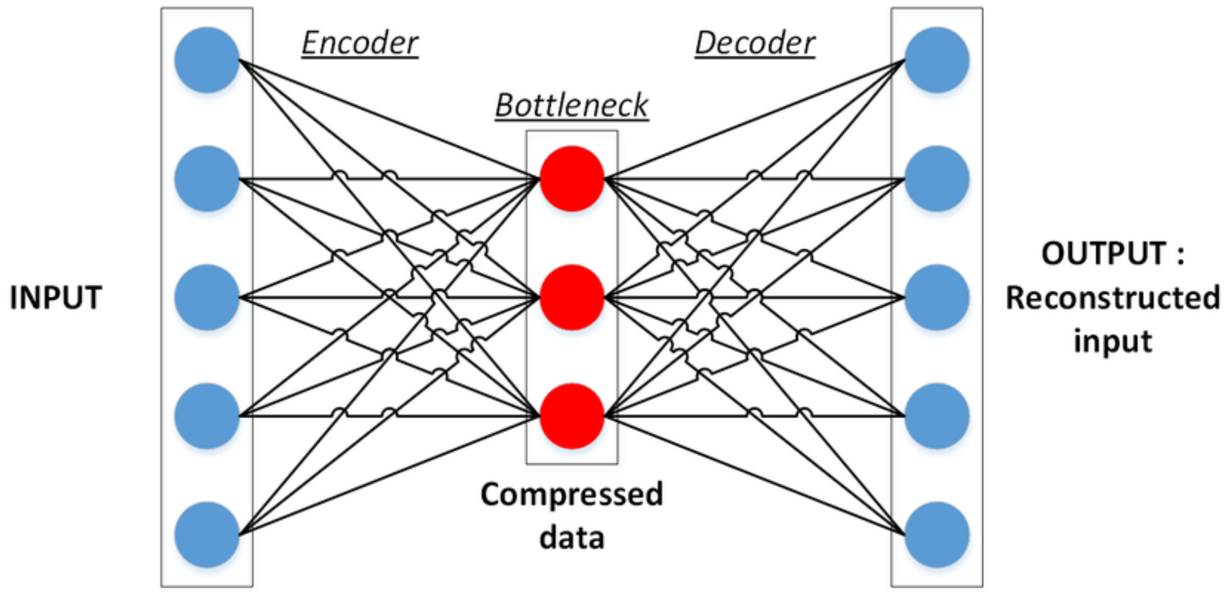
- Data: (x, y)
- Goal: Learn a function f(x)=y
- Examples: Classification, Regression, …





Self-supervised Learning

- Data: x
- Goal: Learn underlying structure of the data
- Examples: Representation Learning, Contrastive Learning, Autoregressive Pretraining



Reinforcement Learning

- Goal: Learn a policy to maximize reward
- Examples: Chess, Go, Poker, Self-driving, LLM

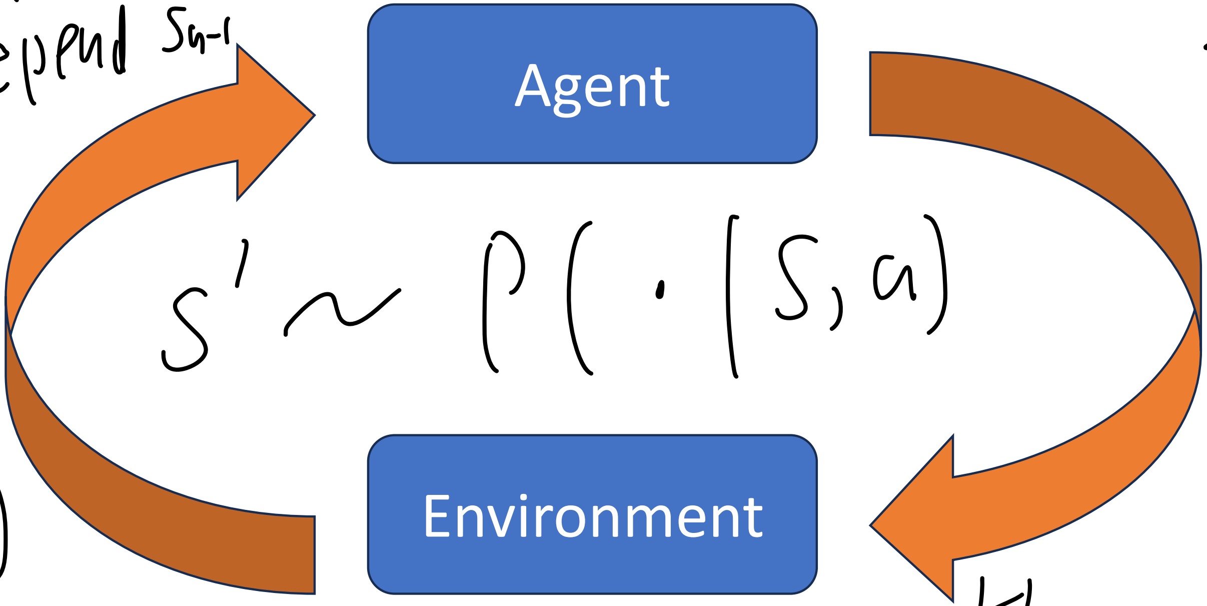
Markov Decision Process

MDP

S_{n+1} only depends on S_n, a_n

does not depend S_{n-1} given S_n

State



$s' \sim P(\cdot \mid s, a)$

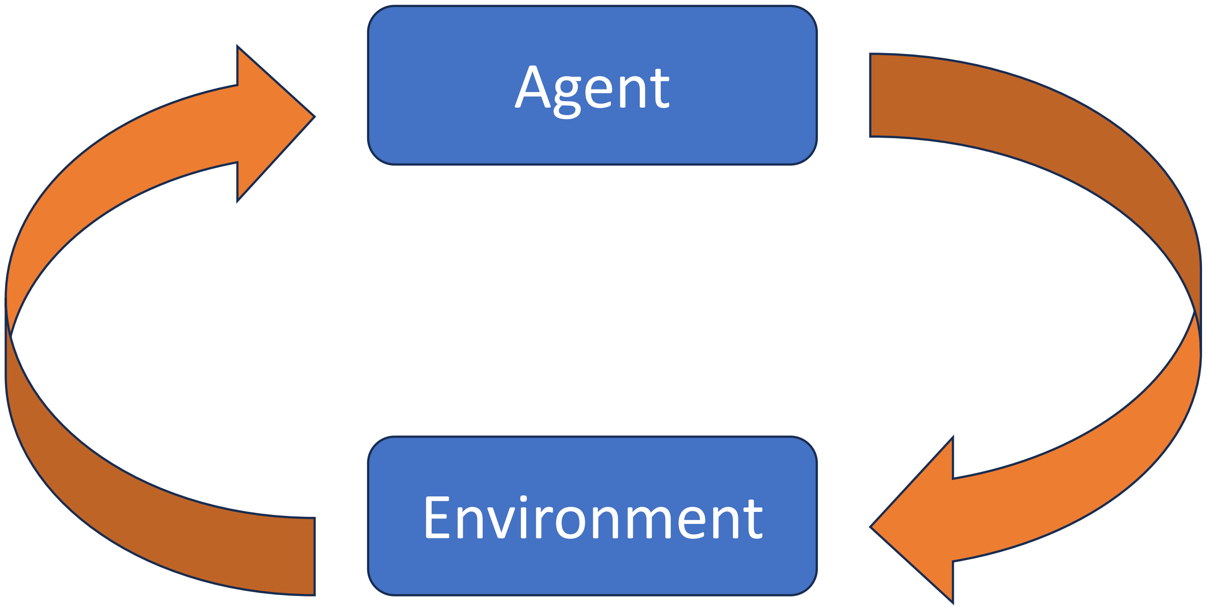
$R(s,a)$

$\pi : S \to A$
policy

- Goal: Collect as much reward as possible. $\mathbb{E}_{P,\pi}\left[\sum_{n=1}^{H} R(s_n, a_n)\right]$

H: total steps

Markov Decision Process



State:

$s_{t+1} \sim P(\cdot | s_t, a_t)$

Reward:

$r_{t+1} = r(s_t, a_t)$

Action $a_t = \pi(s_t)$

Maximize total discounted reward $\sum \gamma^t r_t$.

Markov Decision Process

$\pi : S \to A$

- Policy: $\pi(s) = a$.
- Discount factor: $\gamma \in (0,1)$.
- Value function: $V^{\pi}(s_0) = \mathbb{E}_{\pi}[\sum_t \gamma^t r_t]$, where $s_0, a_0, r_0, s_1, a_1, r_1, \ldots$ is a trajectory sampled by using policy π.
- Q function: $Q^{\pi}(s_0, a_0) = \mathbb{E}_{\pi}[\sum_t \gamma^t r(s_t, a_t)]$.
- Optimal policy: $\pi^* = \operatorname{argmax}_{\pi} V^{\pi}(s)$.
- There exists an optimal policy that achieves the argmax for all s **simultaneously**!

S: initial state

Optimal Q Function

$\mathcal{A} = \{a_0, a_1, \overline{a_2}\}$

- Optimal Q function: $Q^{\pi^*}(s_0, a_0) = \mathbb{E}_{\pi^*}[\sum_t \gamma^t r(s_t, a_t)]$.
- Property: $\pi^*(s) = \mathrm{argmax}_a Q^{\pi^*}(s, a)$.
- If we know Q^*, we know π^*.

$$\begin{cases} Q^{\pi^*}(s, a_0) \\ Q^{\pi^*}(s, a_1) \\ Q^{\pi^*}(s, a_2) \end{cases}$$

Reinforcement Learning

- If we know $r(s, a)$ and $P(s' \mid s, a)$, we can use dynamic programming to solve the optimal policy.
- How to learn the optimal policy **without** the knowledge of $r(s, a)$ and $P(s' \mid s, a)$?
- Collect **samples**!

Challenge: Large State Space

S⟩

19 x 19

- 3^{361} possible board configurations in Go.
- Impossible to enumerate.

- Theorem: $\Omega(SA)$ samples are necessary for learning MDP without structures, where S is # of states and A is # of actions.

Function Approximation

- Challenge in RL: large state and action space.
- Many states and actions are similar and have similar $\underbrace{Q^{\pi^*}}$.
- Use a function class $\mathcal{F} = \{f_\theta\}$ to $\underbrace{\text{approximate Q function.}}$

Q-learning

↓

Deep Q-learning

- Suppose we have a dataset $\mathcal{D} = \underbrace{\{Q^{\pi^*}(s, a)\},}_{\text{don't have this}}$ then we can fit a f_θ to approximate Q^{π^*}:

DQN

$$\theta^* = \operatorname{argmin}_\theta \sum_{(s,a)\in\mathcal{D}} \left(f_\theta(s, a) - Q^{\pi^*}(s, a)\right)^2.$$

Offline Reinforcement Learning

$a_0 \sim \pi(s_0)$ $s_1 \sim P(\cdot \mid s_0, a_0)$

$r_0 \sim R(a_0, s_0)$

- Dataset: trajectories $s_0, a_0, r_0, s_1, a_1, r_1, \ldots, s_T$ sampled from some behavior policy π_b.

- Challenge: unknown $Q^{\pi^*}(s, a)$.

off line: use a $\underline{\text{fixed}}$ Policy π_b to collect data

on line: use adaptive policis ~~to~~ $\pi_1, \ldots, \pi_N$ collect data

Q-learning

finite states
finite actions

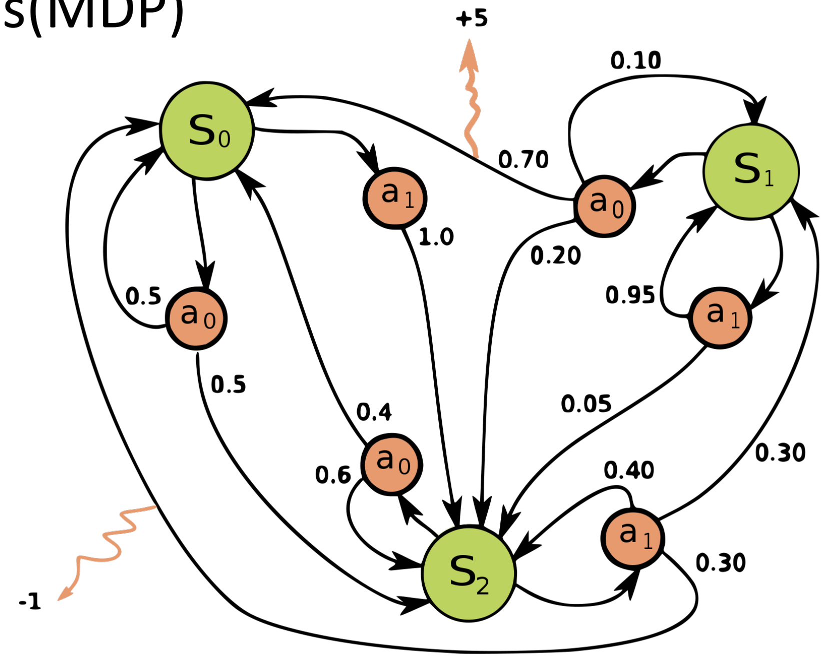
- Reminder: Markov-Decision Process(MDP)

State:

$$s_{h+1} = P(\cdot | s_h, a_h)$$

Reward:

$$r_{h+1} = r(s_h, a_h)$$



Q-learning

- Value-based method:
  - Evaluate all the states, then find the action leading to the best state.
- Reminder: Value function and Q function:

  - $V_\pi(s) = E_\pi[\sum_h \gamma^h r_h \mid s]$

- We need to know which action leads to the given reward:

  - $\mathrm{Q}_\pi(s, a) = E_\pi[\sum_h \gamma^h r_h \mid s, a]$

Q-learning

Q-function:

$$\mathrm{Q}_{\pi}(s,a) = E_{\pi}[\sum_{h} \gamma^{h} r_{h} \mid s,a]$$

- Target: derive the Q function for the optimal policy π^*, Q^*
- How to solve this system?

- Of course, we can use Monte Carlo's Method to estimate Q function.
- But it takes $\Omega(SA^h)$ sample trajectories.

- Can we do better?

Q-learning: Tabular learning

Q-function:

$$Q_\pi(s,a) = E_\pi[\sum_h \gamma^h r_h \mid s,a]$$

- Notice that Q function should satisfy the successor relationship
- Bellman's Equation:
  - $Q^*(s,a) = r(s,a) + \gamma E_{\pi^*}[V^*(s') \mid s,a]$
  - $V^*(s) = \max_a Q^*(s,a)$

$s' \sim p(\cdot \mid s,a)$

- Then we can solve it with polynomial samples!

Q-learning: Tabular learning

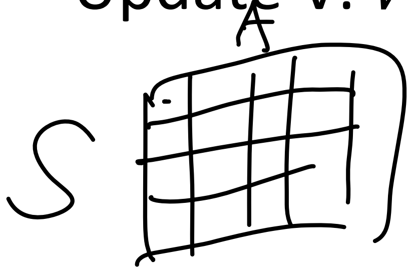
(inf samples)

- First, initialize $Q(\cdot) = 0$; $\Rightarrow V(\cdot) \Leftarrow$
- Then we do iterative DP:
  - Until convergency, do:
    - For $(s, a) \in S \times A$:
      - Update Q: $\underline{Q(s,a)} \leftarrow \frac{1}{\underset{\sim}{N_{s,a}}} \sum_{s_i = s, a_i = a} \left(r_i + \gamma V(s_{i+1})\right)$
    - For $s \in S$:
      - Update V: $V(s) \leftarrow \max_a Q(s,a)$

if finite sample

$N_{s,a}$ for (s,a) pairs

Here $N_{s,a}$ is the counter of (s,a) in dataset.



Deep Q Network (DQN)

- When we combine Deep Learning with Q-learning, we get DQN.
- Reminder: function approximation
  - Structure/function class: MLP, CNN, Transformer, etc.
  - Solve the Bellman's Equation with gradient descent!
    - $Q^*(s,a) = r(s,a) + \gamma E_{\pi^*}[V^*(s') \mid s,a]$ $\Rightarrow$ regression
    - $V^*(s) = \max_a Q^*(s,a)$
  - Loss function:

$$L(\theta) = \mathbf{E}_\theta\Big[(Q_\theta(s,a) - r(s,a) - \gamma\mathbf{E}[V_\theta(s') \mid s,a])^2\Big]$$

$$(s,a,r,s') \xrightarrow[]{\text{given } Q_\theta, V_\theta} \{((s,a),\ r(s,a) + \gamma \cdot V_\theta(s'))\}$$

Deep Q Network (DQN)

- Loss function:

$$L(\theta) = \mathbf{E}_{\theta}\left[(Q_{\theta}(s,a) - r(s,a) - \gamma \mathbf{E}[V_{\theta}(s') \mid s,a])^2\right]$$

- Estimated loss:

$$\mathscr{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}[Q_{\theta}(s_i,a_i) - r_i - \gamma \max_{a'} Q_{\theta}(s_i', a')]^2$$

- Other tricks:
  1. Double network trick for stronger stability;
  2. Replay buffer for higher sample efficiency.

DQN: double network structure

$$\min_{\theta} \quad \mathscr{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}[Q_\theta(s_i, a_i) - r_i - \gamma \max_{a'} Q_{\theta'}(s_i', a')]^2$$

- Evaluate network: trained network θ
  - Updated in each iteration
  - The first Q is the evaluate network
- Target network: temporal copy of evaluate network θ'
  - Updated at regular intervals
  - The second Q is fixed to be target network

- Avoid overfitting problem.
- Don't need to solve a max problem in each iteration.
- Stabilize the training process.

DQN: experience replay

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{i=1}^{N}[Q_\theta(s_i, a_i) - r_i - \gamma \max_{a'} Q_{\theta'}(s_i', a')]^2$$

- Problem: batch size is very small compared with the dataset
  - Each batch may only contain the transitions from a single trajectory
  - Not mutually independent!

- Notice that we only need transitions $\{s_i, a_i, r_i, s_i'\}$,instead of complete trajectories.
- Solution: In each iteration, we randomly sample data from the replay buffer to form the training batch.
- The replay buffer can be the offline dataset, or the data collected with latest policy, which gives better sample efficiency.

Policy-Gradient

$\pi_\theta(s,a) \rightarrow (0,1)$

- Sometimes we don't want to estimate the Value function!
  - Value function approximation can be extremely tricky;
    - Empirical experiments tell us simpler algorithm leads to better performance;
  - We need to solve an argmax/max problem for each update, which can be very expensive.

$$\pi(s) \leftarrow \arg\max_a \{\mathbf{E}_\pi[\sum \gamma^h r_h \mid s, a]\}$$

- Policy-Gradient(PG) directly optimize the policy!
- Directly approximate $\pi^*(\cdot)$ with DNN.
  - Now we use π_θ to denote the policy we want to learn.

Policy-Gradient

- Denote the probability of getting a certain trajectory τ as $P(\tau, \theta)$, and the corresponding reward as $R(\tau)$.

$$P(\tau, \theta) = \prod_h \pi_\theta(\, a_h \mid s_h \,)$$
$$R(\tau) = \sum_h \gamma^h r_h$$

- Target: maximize $J(\theta) = E_{\pi_\theta}[\sum_h \gamma^h r_h] = \sum_h P(\tau, \theta) R(\tau)$
- Gradient ascent: $\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$

- The problem lies in the estimation of $\nabla_\theta J(\theta)$.

Policy-Gradient

- Target: maximize $J(\theta) = E_{\pi_\theta}[\sum_h \gamma^h r_h] = P(\tau, \theta)R(\tau)$
- Gradient ascent: $\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$
- Directly calculation of the gradient of empirical reward gives:

$$J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N} R(\tau_i),$$

$$\nabla_\theta J(\theta) \approx \nabla_\theta [\frac{1}{N}\sum_{i=1}^{N} R(\tau_i)]?$$

- Remember that $R(\tau)$ doesn't depend on θ directly:
  - $P(\tau, \theta) = \prod_h \pi_\theta(a_h \mid s_h)$
  - $R(\tau) = \sum_h \gamma^h r_h$

Policy-Gradient

- Target: maximize $J(\theta) = E_{\pi_\theta}[\sum_h \gamma^h r_h] = P(\tau,\theta)R(\tau)$

$$\left.\begin{aligned}
\nabla_\theta J(\theta) &= \sum_\tau \nabla_\theta P(\tau,\theta)R(\tau) \\
&= \sum_\tau \underbrace{\overset{=1}{\frac{P(\tau,\theta)}{P(\tau,\theta)}}} \nabla_\theta P(\tau,\theta)R(\tau) \\
&= \sum_\tau P(\tau,\theta) \underbrace{\boxed{\frac{\nabla_\theta P(\tau,\theta)}{P(\tau,\theta)}}} R(\tau) \\
&= \sum_\tau P(\tau,\theta)\nabla_\theta \log P(\tau,\theta)R(\tau) \\
&= \mathbf{E}_{\pi_\theta}[R(\tau)\nabla_\theta \log P(\tau,\theta)]
\end{aligned}\right\}$$

policy gradient theorem

- Good! The gradient can be also understood as an expectation!
- Therefore, the empirical update function is:

$$\theta \leftarrow \theta + \frac{\eta}{N}\sum_{i=1}^{N} R(\tau_i)\,\nabla_\theta \log P(\tau_i,\theta)$$

Decision Transformers

Decision making

Architecture in language modeling

- Language modeling: autoregressive conditional sequence modeling
  - Predict next **token** (≈word) with some probability

$$P(\text{“you”}|[\text{“How”}, \text{“ ”}, \text{“are”}, \text{“ ”}])$$

  - **Autoregressive:** sample, and predict next

$$P(\text{“? ”}|[\text{“How”}, \text{“ ”}, \text{“are”}, \text{“ ”}, \text{“you”}])$$

- Just like **policy** in RL!

$$\pi(a_t|s_1, a_1, r_1, \ldots, s_t)$$

Decision Transformers for Offline RL[🔗]

- Offline dataset:
  - Consider **deterministic reward, finite horizon $\boldsymbol{H}$, and discount $\boldsymbol{\gamma = 1}$**

$$D = \{\tau^i = (s_0^i, a_0^i, r_0^i; s_1^i, a_1^i, r_1^i; \cdots; s_H^i, a_H^i, r_H^i)\}_{i=1}^N$$

- Decision Transformers:
  - **Return-to-go:** $\hat{R}_t = \sum_{h=t}^{H} r_h$

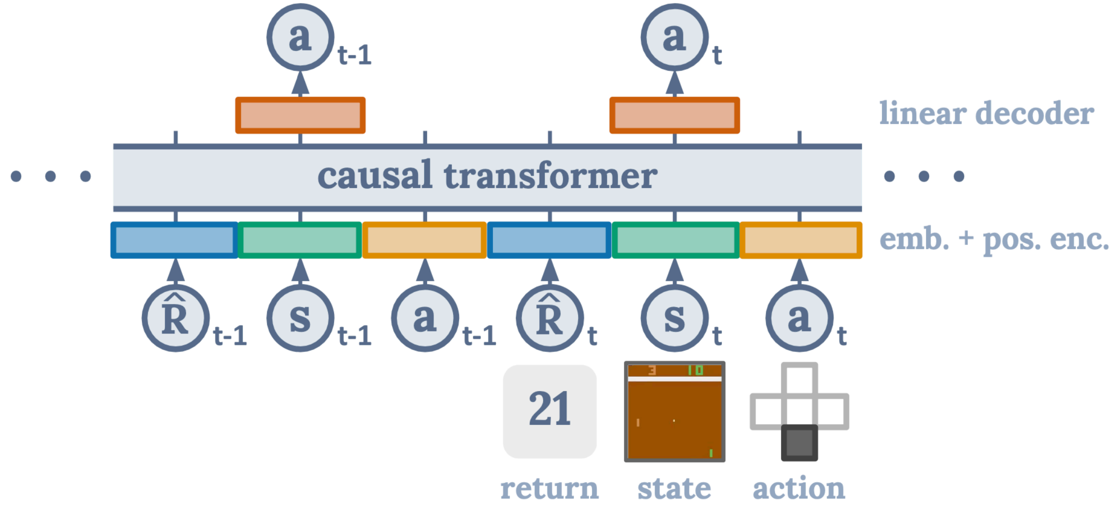
$$D = \{\tau^i = (\hat{R}_0^i, s_0^i, a_0^i; \hat{R}_1^i, s_1^i, a_1^i; \cdots; \hat{R}_H^i, s_H^i, a_H^i)\}_{i=1}^N$$

$$\hat{R}_0^i = \sum_{h=0}^{H} r_h^i$$

Decision Transformers for Offline RL

- Decision Transformers:
  - **Return-to-go (RTG):** $\hat{R}_t = \sum_{h=t}^{H} r_h$

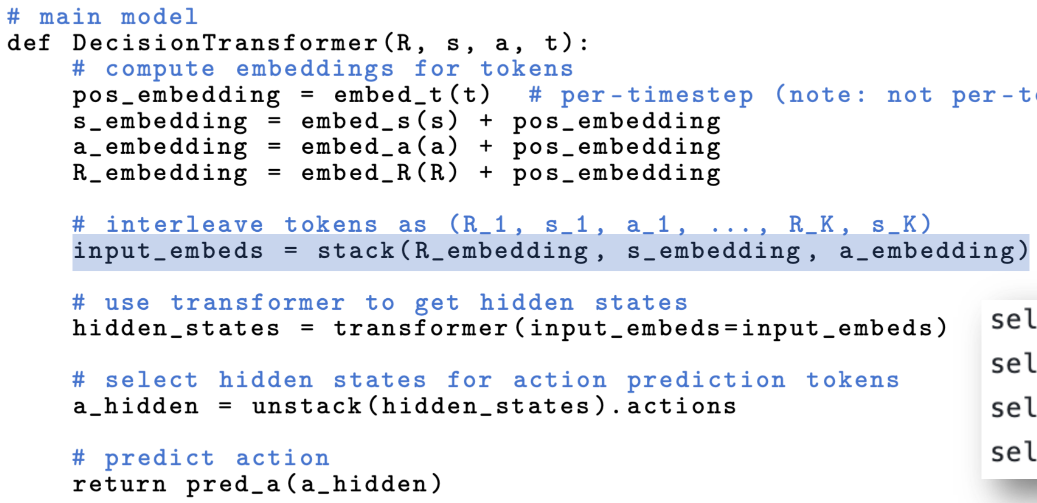
$$D = \left\{\tau^i = (\hat{R}_0^i, s_0^i, a_0^i; \hat{R}_1^i, s_1^i, a_1^i; \cdots; \hat{R}_H^i, s_H^i, a_H^i)\right\}_{i=1}^{N}$$



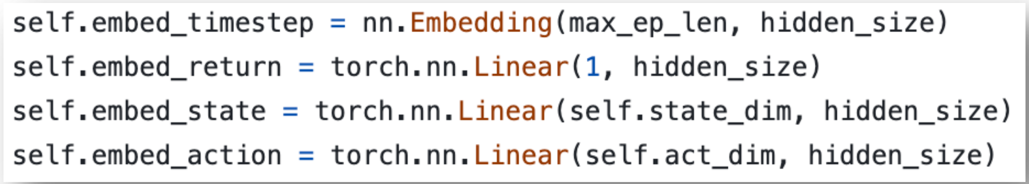
Decision Transformers for Offline RL

- Decision Transformers:
  - **Return-to-go (RTG):** $\hat{R}_t = \sum_{h=t}^{H} r_h$

$$D = \left\{\tau^i = \left(\hat{R}_0^i, s_0^i, a_0^i; \hat{R}_1^i, s_1^i, a_1^i; \cdots; \hat{R}_H^i, s_H^i, a_H^i\right)\right\}_{i=1}^{N}$$



```
# main model
def DecisionTransformer(R, s, a, t):
    # compute embeddings for tokens
    pos_embedding = embed_t(t)  # per-timestep (note: not per-token)
    s_embedding = embed_s(s) + pos_embedding
    a_embedding = embed_a(a) + pos_embedding
    R_embedding = embed_R(R) + pos_embedding

    # interleave tokens as (R_1, s_1, a_1, ..., R_K, s_K)
    input_embeds = stack(R_embedding, s_embedding, a_embedding)

    # use transformer to get hidden states
    hidden_states = transformer(input_embeds=input_embeds)

    # select hidden states for action prediction tokens
    a_hidden = unstack(hidden_states).actions

    # predict action
    return pred_a(a_hidden)
```



```
self.embed_timestep = nn.Embedding(max_ep_len, hidden_size)
self.embed_return = torch.nn.Linear(1, hidden_size)
self.embed_state = torch.nn.Linear(self.state_dim, hidden_size)
self.embed_action = torch.nn.Linear(self.act_dim, hidden_size)
```

Training

- Minibatch of sequence with length K
  - Context length K: use previous K steps to predict next action
  - Slice τ^i into $\tau^i_{[\max\{j-K+1,1\}:j]}$ for $j = 1,2,\dots,H$

$$\tau^i_{[l:r]} = \left(\hat{R}^i_l, s^i_l, a^i_l; \dots; \hat{R}^i_r, s^i_r, a^i_r\right)$$
$$\check{\tau}^i_{[l:r]} = \left(\hat{R}^i_l, s^i_l, a^i_l; \dots; \hat{R}^i_r, s^i_r\right)$$

Training

- Loss function
  - **Cross-entropy loss** for discrete action space

$$\mathcal{L}_{\text{decision}} = \sum_{i=1}^{N} \sum_{j=1}^{H} - \log \pi\left(a_j^i \middle| \check{\tau}^i_{[\max\{j-K+1,1\}:j]}\right)$$

  - ***L*2 loss** for continuous action space

$$\mathcal{L}_{\text{decision}} = \sum_{i=1}^{N} \sum_{j=1}^{H} \mathbb{E}_{a \sim \pi(\cdot | \check{\tau}^i_{[\max\{j-K+1,1\}:j]})} \left(a_j^i - a\right)^2$$

```
# training loop
for (R, s, a, t) in dataloader:  # dims: (batch_size, K, dim)
    a_preds = DecisionTransformer(R, s, a, t)
    loss = mean((a_preds - a)**2)  # L2 loss for continuous actions
    optimizer.zero_grad(); loss.backward(); optimizer.step()
```

Evaluation

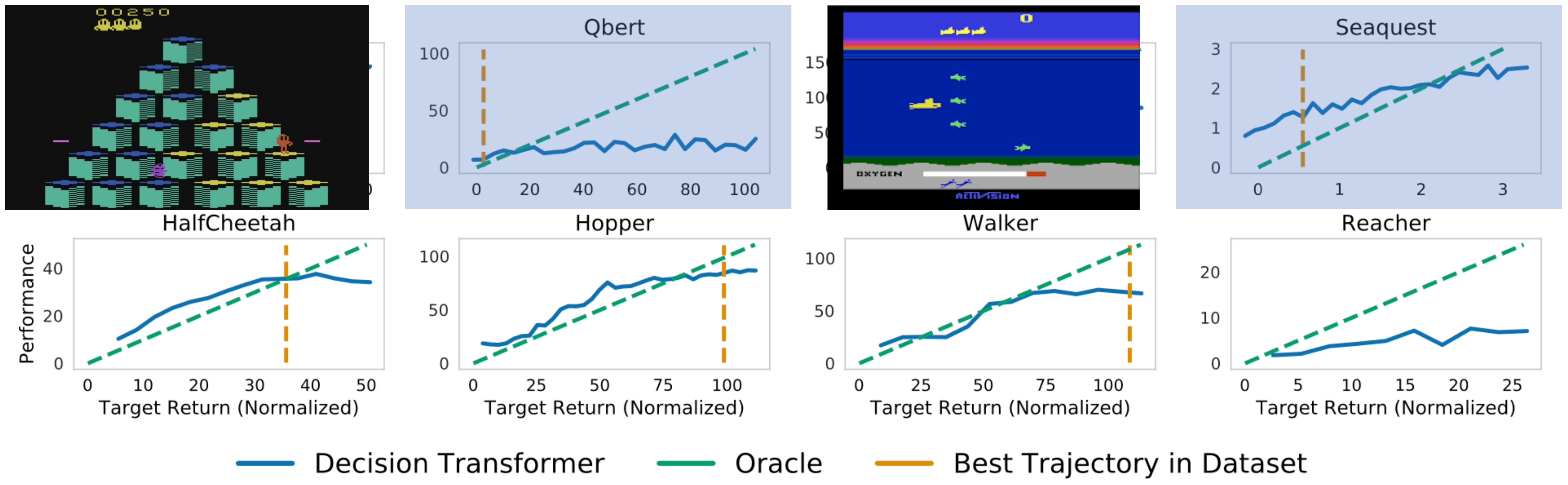
- Set an **initial RTG** (large enough)
- Run the DT and subtract the current return-to-go with the observed reward
- Crop the sequence to length K

at time t $\sum_{h=t}^{H} r_h$, already get $r_1, \dots, r_{t-1}$

init RTG $- \sum_{h=1}^{t-1} r_h$

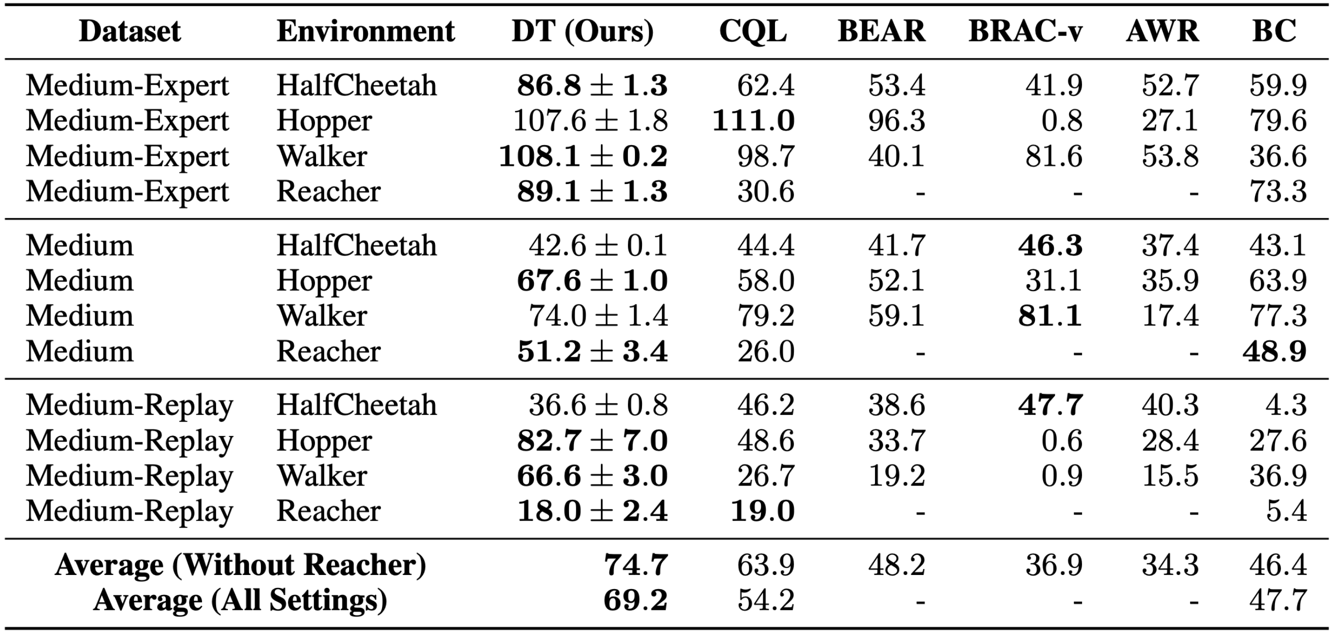
```
# evaluation loop
target_return = 1  # for instance, expert-level return
R, s, a, t, done = [target_return], [env.reset()], [], [1], False
while not done:  # autoregressive generation/sampling
    # sample next action
    action = DecisionTransformer(R, s, a, t)[-1]  # for cts actions
    new_s, r, done, _ = env.step(action)

    # append new tokens to sequence
    R = R + [R[-1] - r]  # decrement returns-to-go with reward
    s, a, t = s + [new_s], a + [action], t + [len(R)]
    R, s, a, t = R[-K:], ...  # only keep context length of K
```

Results



- **Possible** to outperform the best trajectory in dataset

Results

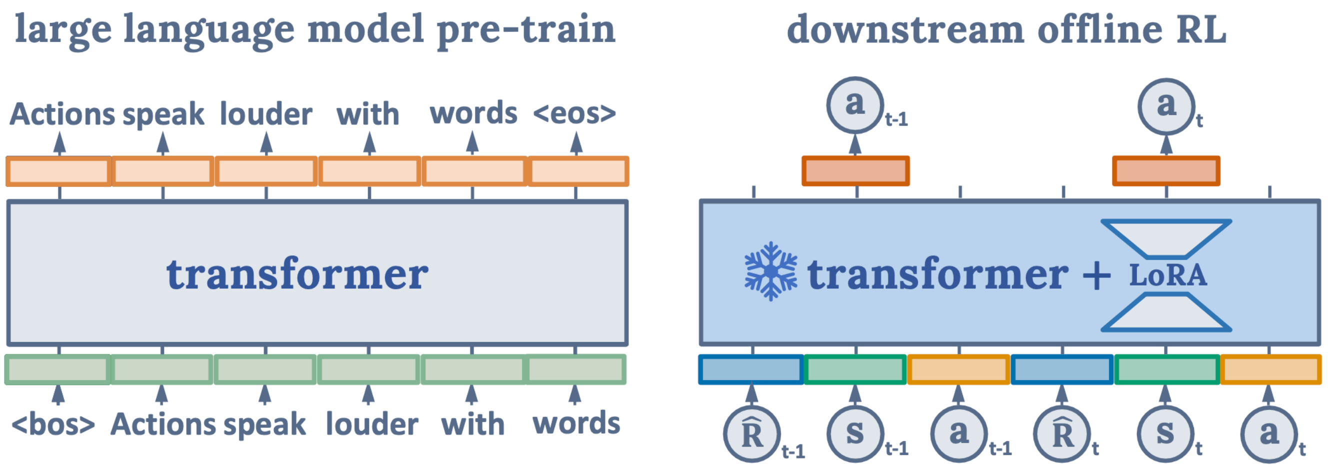


Dataset	Environment	DT (Ours)	CQL	BEAR	BRAC-v	AWR	BC
Medium-Expert	HalfCheetah	**86.8 ± 1.3**	62.4	53.4	41.9	52.7	59.9
Medium-Expert	Hopper	107.6 ± 1.8	**111.0**	96.3	0.8	27.1	79.6
Medium-Expert	Walker	**108.1 ± 0.2**	98.7	40.1	81.6	53.8	36.6
Medium-Expert	Reacher	**89.1 ± 1.3**	30.6	-	-	-	73.3
Medium	HalfCheetah	42.6 ± 0.1	44.4	41.7	**46.3**	37.4	43.1
Medium	Hopper	**67.6 ± 1.0**	58.0	52.1	31.1	35.9	63.9
Medium	Walker	74.0 ± 1.4	79.2	59.1	**81.1**	17.4	77.3
Medium	Reacher	**51.2 ± 3.4**	26.0	-	-	-	**48.9**
Medium-Replay	HalfCheetah	36.6 ± 0.8	46.2	38.6	**47.7**	40.3	4.3
Medium-Replay	Hopper	**82.7 ± 7.0**	48.6	33.7	0.6	28.4	27.6
Medium-Replay	Walker	**66.6 ± 3.0**	26.7	19.2	0.9	15.5	36.9
Medium-Replay	Reacher	**18.0 ± 2.4**	**19.0**	-	-	-	5.4
Average (Without Reacher)		**74.7**	63.9	48.2	36.9	34.3	46.4
Average (All Settings)		**69.2**	54.2	-	-	-	47.7

- CQL: conservative Q-learning
- BEAR: off-policy Q-learning
- BRAC-v: behavior regularized offline RL
- AWR: advantage-weighted regression
- BC: behavior cloning

Pretraining DTs on Language Tasks[🔗]

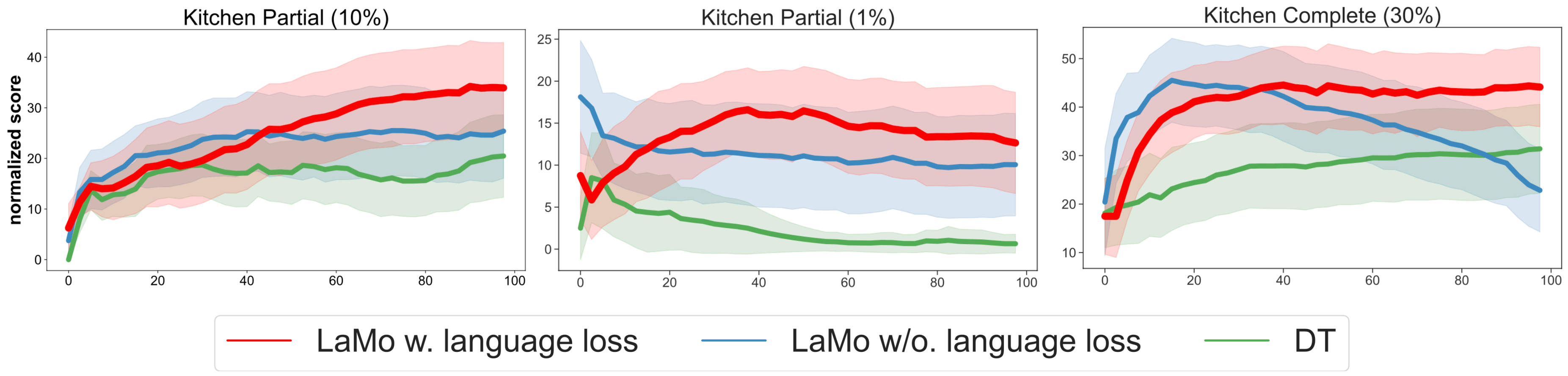
- Use a pretrained language model (GPT2) as initialization


large language model pre-train
Actions speak louder with words <eos>
transformer
<bos> Actions speak louder with words
downstream offline RL
a_{t-1}
a_t
transformer + LoRA
$\hat{R}_{t-1}$ s_{t-1} a_{t-1} $\hat{R}_t$ s_t a_t

Pretraining DTs on Language Tasks

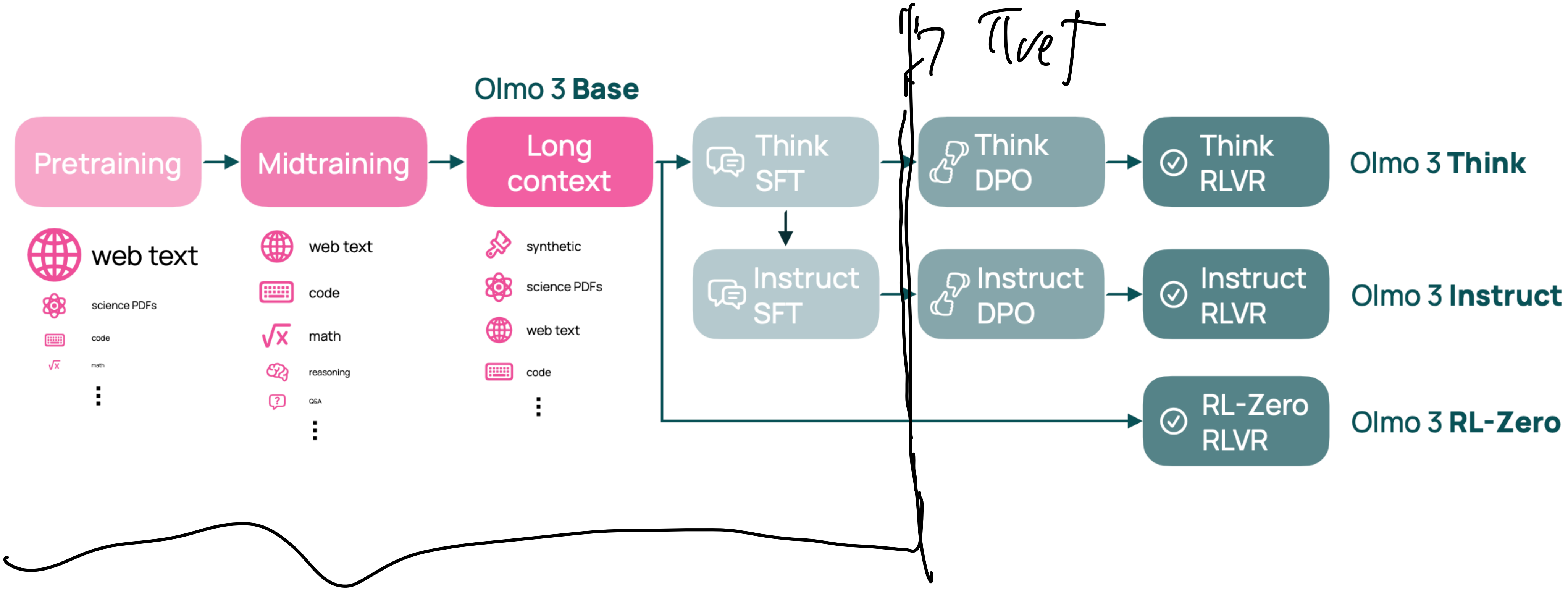
- Language prediction as an auxiliary objective
  - WikiText dataset
  - $\mathcal{L}_{\text{language}} = \sum_i -\log T(w_{i+1} | w_1, \dots, w_i)$

$$\mathcal{L} = \mathcal{L}_{\text{decision}} + \lambda \mathcal{L}_{\text{language}}$$



Reinforcement Learning for Large Language Models

LLM Pipeline



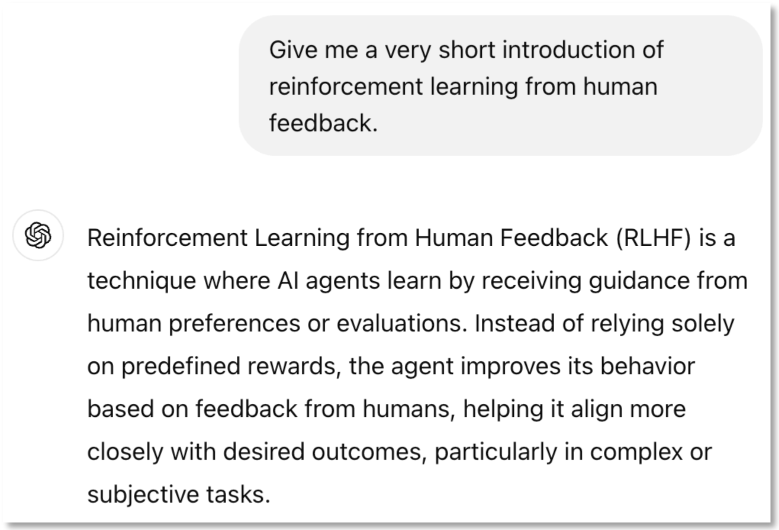
From Imitation to Optimization

- SFT: imitation
  - fit $\hat{P}(y|x) \approx p^*(y|x)$ for some reference distribution p^*.
  - Use auto-regressive modeling.
  - Collect data from p^*.

- Optimization:
  - Find $\hat{P}(y|x)$ to maximize some reward.
  - LLMs are policies, not a model of some distribution.
  - We need a reward function.
  - Why:
    - Can be less costly than SFT.
    - Generation vs. Verification.

Language Generation

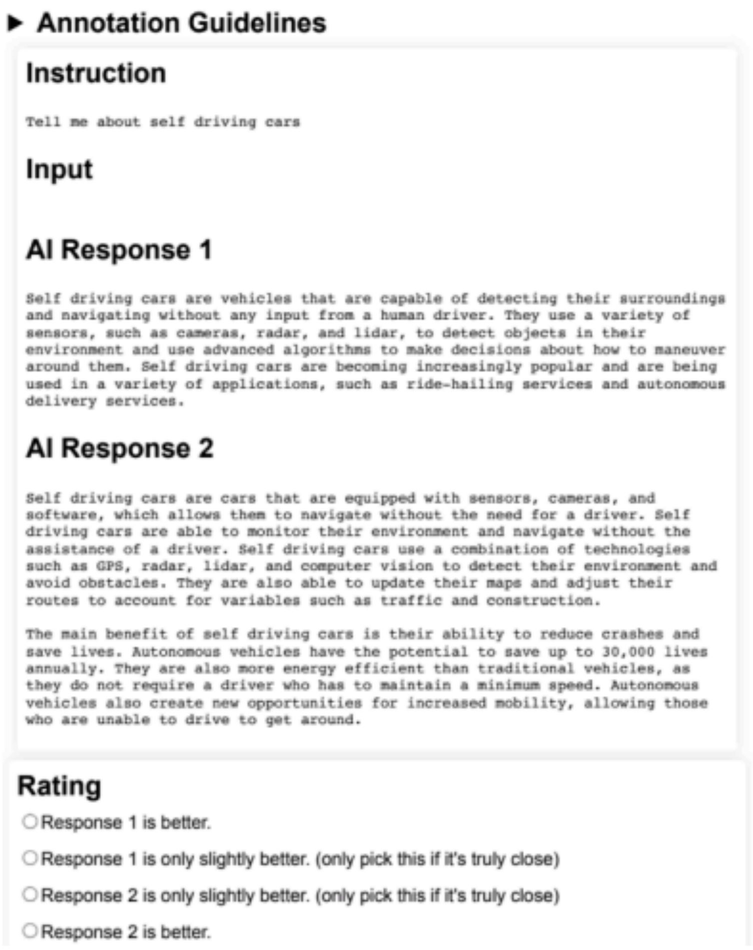
- **Prompt (State)**: user queries
- **Response (Action)**: language model generation result



Give me a very short introduction of reinforcement learning from human feedback.

Reinforcement Learning from Human Feedback (RLHF) is a technique where AI agents learn by receiving guidance from human preferences or evaluations. Instead of relying solely on predefined rewards, the agent improves its behavior based on feedback from humans, helping it align more closely with desired outcomes, particularly in complex or subjective tasks.

Comparison Data



▶ **Annotation Guidelines**

Instruction

```
Tell me about self driving cars
```

Input

AI Response 1

```
Self driving cars are vehicles that are capable of detecting their surroundings
and navigating without any input from a human driver. They use a variety of
sensors, such as cameras, radar, and lidar, to detect objects in their
environment and use advanced algorithms to make decisions about how to maneuver
around them. Self driving cars are becoming increasingly popular and are being
used in a variety of applications, such as ride-hailing services and autonomous
delivery services.
```

AI Response 2

```
Self driving cars are cars that are equipped with sensors, cameras, and
software, which allows them to navigate without the need for a driver. Self
driving cars are able to monitor their environment and navigate without the
assistance of a driver. Self driving cars use a combination of technologies
such as GPS, radar, lidar, and computer vision to detect their environment and
avoid obstacles. They are also able to update their maps and adjust their
routes to account for variables such as traffic and construction.

The main benefit of self driving cars is their ability to reduce crashes and
save lives. Autonomous vehicles have the potential to save up to 30,000 lives
annually. They are also more energy efficient than traditional vehicles, as
they do not require a driver who has to maintain a minimum speed. Autonomous
vehicles also create new opportunities for increased mobility, allowing those
who are unable to drive to get around.
```

Rating

○ Response 1 is better.

○ Response 1 is only slightly better. (only pick this if it's truly close)

○ Response 2 is only slightly better. (only pick this if it's truly close)

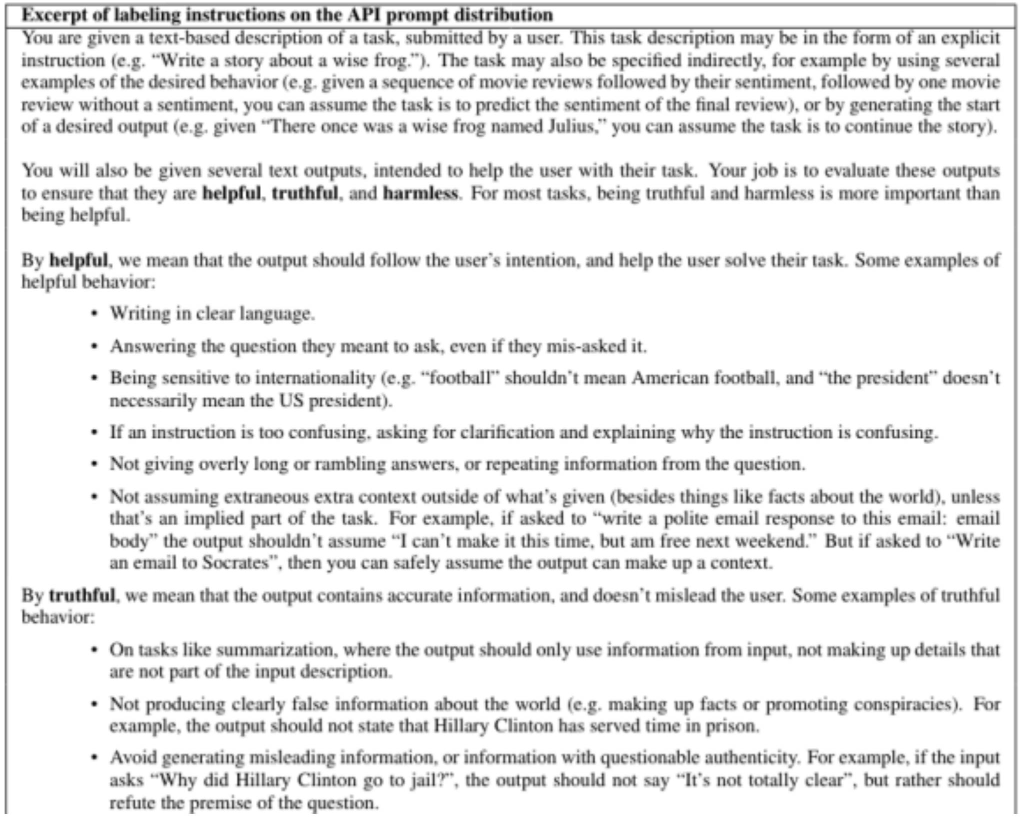
○ Response 2 is better.

RLHF

RL with

human feedback

Comparison Data



Excerpt of labeling instructions on the API prompt distribution

You are given a text-based description of a task, submitted by a user. This task description may be in the form of an explicit instruction (e.g. "Write a story about a wise frog."). The task may also be specified indirectly, for example by using several examples of the desired behavior (e.g. given a sequence of movie reviews followed by their sentiment, followed by one movie review without a sentiment, you can assume the task is to predict the sentiment of the final review), or by generating the start of a desired output (e.g. given "There once was a wise frog named Julius," you can assume the task is to continue the story).

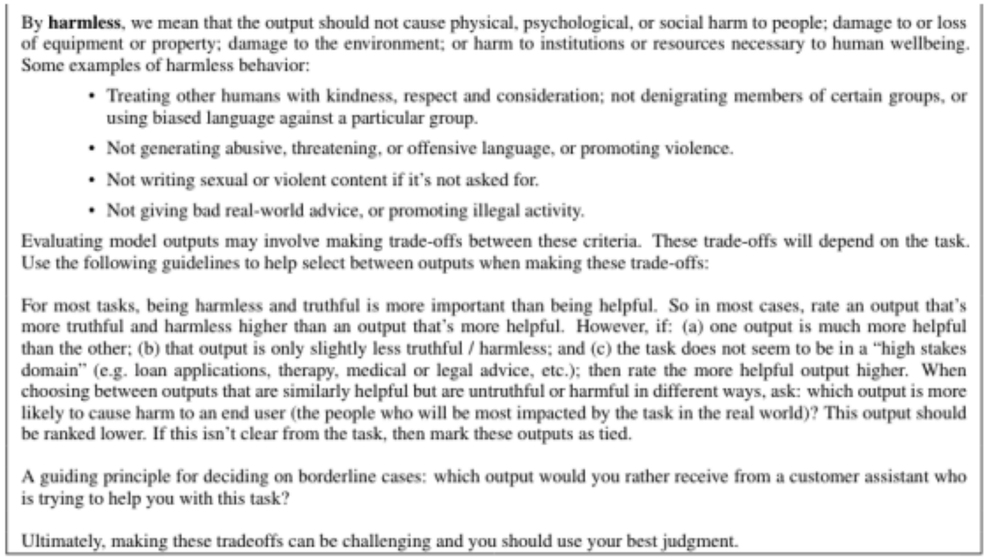
You will also be given several text outputs, intended to help the user with their task. Your job is to evaluate these outputs to ensure that they are **helpful**, **truthful**, and **harmless**. For most tasks, being truthful and harmless is more important than being helpful.

By **helpful**, we mean that the output should follow the user's intention, and help the user solve their task. Some examples of helpful behavior:

- Writing in clear language.
- Answering the question they meant to ask, even if they mis-asked it.
- Being sensitive to internationality (e.g. "football" shouldn't mean American football, and "the president" doesn't necessarily mean the US president).
- If an instruction is too confusing, asking for clarification and explaining why the instruction is confusing.
- Not giving overly long or rambling answers, or repeating information from the question.
- Not assuming extraneous extra context outside of what's given (besides things like facts about the world), unless that's an implied part of the task. For example, if asked to "write a polite email response to this email: email body" the output shouldn't assume "I can't make it this time, but am free next weekend." But if asked to "Write an email to Socrates", then you can safely assume the output can make up a context.

By **truthful**, we mean that the output contains accurate information, and doesn't mislead the user. Some examples of truthful behavior:

- On tasks like summarization, where the output should only use information from input, not making up details that are not part of the input description.
- Not producing clearly false information about the world (e.g. making up facts or promoting conspiracies). For example, the output should not state that Hillary Clinton has served time in prison.
- Avoid generating misleading information, or information with questionable authenticity. For example, if the input asks "Why did Hillary Clinton go to jail?", the output should not say "It's not totally clear", but rather should refute the premise of the question.



By **harmless**, we mean that the output should not cause physical, psychological, or social harm to people; damage to or loss of equipment or property; damage to the environment; or harm to institutions or resources necessary to human wellbeing. Some examples of harmless behavior:

- Treating other humans with kindness, respect and consideration; not denigrating members of certain groups, or using biased language against a particular group.
- Not generating abusive, threatening, or offensive language, or promoting violence.
- Not writing sexual or violent content if it's not asked for.
- Not giving bad real-world advice, or promoting illegal activity.

Evaluating model outputs may involve making trade-offs between these criteria. These trade-offs will depend on the task. Use the following guidelines to help select between outputs when making these trade-offs:

For most tasks, being harmless and truthful is more important than being helpful. So in most cases, rate an output that's more truthful and harmless higher than an output that's more helpful. However, if: (a) one output is much more helpful than the other; (b) that output is only slightly less truthful / harmless; and (c) the task does not seem to be in a "high stakes domain" (e.g. loan applications, therapy, medical or legal advice, etc.); then rate the more helpful output higher. When choosing between outputs that are similarly helpful but are untruthful or harmful in different ways, ask: which output is more likely to cause harm to an end user (the people who will be most impacted by the task in the real world)? This output should be ranked lower. If this isn't clear from the task, then mark these outputs as tied.

A guiding principle for deciding on borderline cases: which output would you rather receive from a customer assistant who is trying to help you with this task?

Ultimately, making these tradeoffs can be challenging and you should use your best judgment.

Approach 1: Train a Reward Model then Optimize

(x, y_1, y_2)

- Reward model $r_\theta(x, y)$, x: prompt, y: good/bad comparison.
  - Also use an LLM.
- Sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

- Bradley-Terry (BT) preference model

$$p^\star(y_1 \succ y_2) = \sigma\big(r_\theta(x, y_1) - r_\theta(x, y_2)\big) = \frac{e^{r_\theta(x,y_1)}}{e^{r_\theta(x,y_1)} + e^{r_\theta(x,y_2)}}$$

- Loss: $\min_\theta -\log\big(r_\theta(x, y_1) - r_\theta(x, y_2)\big)$

From Policy Gradient to PPO

x: prompt

- Say we have trained a reward model: r(x,y). Want to train π_θ (LLM).
- Loss: $\underset{\theta}{argmax} \sum_x \mathbb{E}_{y\sim\pi_\theta}[r(x,y)] - \beta KL(\pi_\theta || \underline{\pi_{ref}})$ init checkpoint
- Policy gradient algo: $\theta \leftarrow \theta - \eta \cdot r(x,y) \cdot \nabla_\theta \log \pi_\theta(x,y)$
  - Data (x,y) needs to be sampled from the **current policy π_θ.** $\boxed{\text{on-policy}}$ policy gradient
  - Practice:
    - Collect data from π_{old} (policy from several iterations in the past).
    - Reasons: parallel sampling, async update and sampling, larger throughput.

at iter t, use π_{θ_t} to sample then use data update

$\pi_{\theta_{t-10}}$

- Correction via importance sampling:
  - if (x,y) is sampled from an **old policy π_{old}.**
  - $\theta \leftarrow \theta - \eta \cdot r(x,y) \cdot \nabla_\theta \log \pi_\theta(x,y) \cdot \boxed{\frac{\pi_\theta(x,y)}{\pi_{old}(x,y)}}$

From Policy Gradient to PPO

- Problem 2: variance of gradient can be large:
  - Even with the same input x: $\theta \leftarrow \theta - \eta \cdot \underbrace{r(x,y)} \cdot \nabla_\theta \log \pi_\theta(x,y) \cdot \boxed{\frac{\pi_\theta(x,y)}{\pi_{old}(x,y)}}$
  - Solution: subtract a baseline b(x) Variance - reduction
    - $\theta \leftarrow \theta - \eta \cdot (r(x,y) - b(x)) \cdot \nabla_\theta \log \pi_\theta(x,y) \cdot \frac{\pi_\theta(x,y)}{\pi_{old}(x,y)}$
    - Simple b(x): a running average.
    - Call $A(x,y) = r(x,y) - b(x)$ the **advantage.**
- Problem 3: training-inference mismatch
  - $\frac{\pi_\theta(x,y)}{\pi_{old}(x,y)}$ can be very large or small => large variance.
  - TRPO (Trust-Region Policy Optimization):
    - Add a constraint in the loss: $KL(\pi_{old}(\cdot|x) \,||\, \pi_\theta(\cdot|x)) \le \delta$
    - Restrict the deviation from π_{old}.

From Policy Gradient to PPO

- PPO (Proximal Policy Optimization)
  - Original gradient: $(r(x,y) - b(x)) \cdot \nabla_\theta \log \pi_\theta(x,y) \cdot \frac{\pi_\theta(x,y)}{\pi_{old}(x,y)}$
  - Just use simple clipping to control:
  - Let's call $C_\theta(x,y) = (r(x,y) - b(x)) \cdot \nabla_\theta \log \pi_\theta(x,y) \cdot$
  - PPO update: $\boldsymbol{min}\left(\frac{\pi_\theta(x,y)}{\pi_{old}(x,y)} \cdot C_\theta(x,y),\ \boldsymbol{clip}\left(\frac{\pi_\theta(x,y)}{\pi_{old}(x,y)}, 1+\epsilon, 1-\epsilon\right) C_\theta(x,y)\right)$
  - Intuition 1: If $C_\theta(x,y) > 0$, increasing the prob helps, but stops rewarding once $\frac{\pi_\theta(x,y)}{\pi_{old}(x,y)} > 1+\epsilon$
  - Intuition 2: If $C_\theta(x,y) < 0$, decreasing the prob helps, but stops rewarding once $\frac{\pi_\theta(x,y)}{\pi_{old}(x,y)} < 1-\epsilon$

Approach 2: Direct Preference Optimization

DPO

- A **tabular softmax** policy π_θ satisfies

$$\pi_\theta(y) = \frac{\mathrm{e}^{\theta_y}}{\sum_{y'} \mathrm{e}^{\theta_{y'}}}$$

- Human preference dataset $\mathcal{D} = \left\{\left(y_w^{(i)}, y_l^{(i)}\right)\right\}_{i=1}^{N}$
  - In the ith sample, $y_w^{(i)}$ is preferred over $y_l^{(i)}$
- **Step 1**: Learn reward function by minimizing negative log-likelihood

$$\mathcal{L}_r(\phi) = -\frac{1}{N}\sum_{i=1}^{N} \log \sigma\left(r_\phi\left(y_w^{(i)}\right) - r_\phi\left(y_l^{(i)}\right)\right)$$

Approach 2: Direct Preference Optimization (DPO)

- A **tabular softmax** policy π_θ satisfies

$$\pi_\theta(y) = \frac{\mathrm{e}^{\theta_y}}{\sum_{y'} \mathrm{e}^{\theta_{y'}}}$$

- Under tabular softmax parametrization

$$\underset{\theta}{argmax} \sum_x \mathbb{E}_{y\sim\pi_\theta}[r(x,y)] - \beta KL(\pi_\theta || \pi_{ref})$$

is equivalent to

$$\pi^\star_\phi(y) = \frac{1}{Z_\phi} \pi_{\text{ref}}(y) \mathrm{e}^{r_\phi(y)/\beta}$$

where Z is the normalizing factor

Direct Preference Optimization (DPO)

- For any y,

$$r_\phi(y) = \beta\left(\log Z_\phi + \log \frac{\pi_\phi^\star(y)}{\pi_{\mathrm{ref}}(y)}\right)$$

- Plug into reward loss and Z_ϕ cancels out!

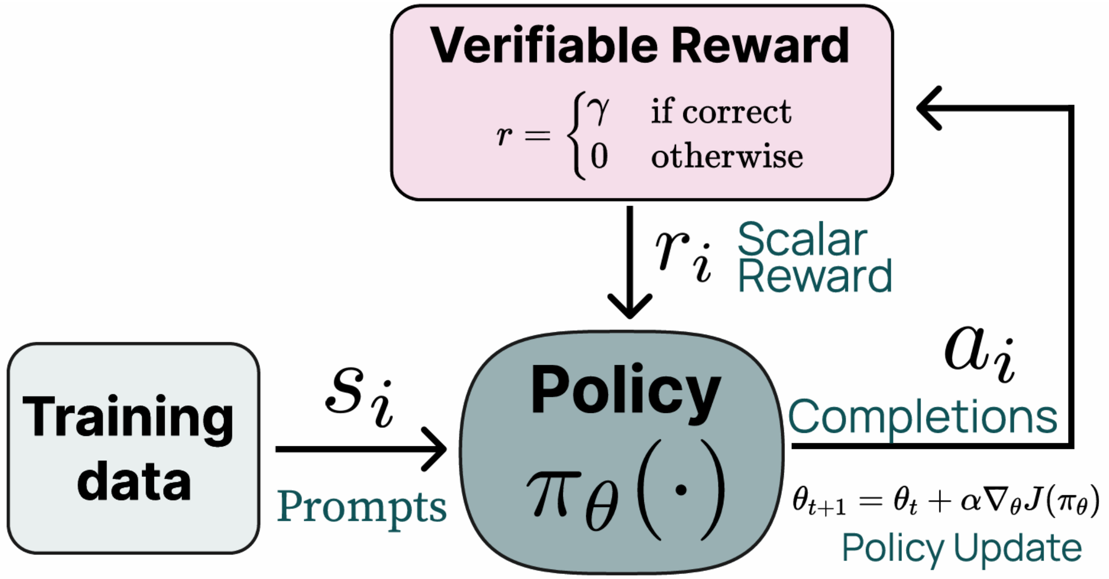
$$\mathcal{L}_\pi(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log \sigma\left(\beta \log \frac{\pi_\theta\left(y_w^{(i)}\right)}{\pi_{\mathrm{ref}}\left(y_w^{(i)}\right)} - \log \frac{\pi_\theta\left(y_l^{(i)}\right)}{\pi_{\mathrm{ref}}\left(y_l^{(i)}\right)}\right)$$

RLVR

- **Reinforcement Learning with verifiable reward**

- RLVR is widely used in improving LLM performance in reasoning tasks (math, code, etc.)

- Use verifiable outcome reward in RL training (e.g. 0-1 correctness reward for math data)


Verifiable Reward
$r = \begin{cases} \gamma & \text{if correct} \\ 0 & \text{otherwise} \end{cases}$
r_i
Scalar Reward
Training data
s_i
Prompts
Policy
$\pi_\theta(\cdot)$
a_i
Completions
$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\pi_\theta)$
Policy Update

Evaluation Dataset

Mathematical reasoning tasks:

- MATH500
- AIME2024
- AIME2025
- AMC2023
- Minerva Math
- OlympiadBench

Non-mathematical reasoning tasks:

- ARC-Easy/Challenge

Example from MATH500:

Convert the point \$(0,3)\$ in rectangular coordinates to polar coordinates. Enter your answer in the form \$(r,\theta),\$ where \$r > 0\$ and \$0 \le \theta < 2 \pi.\$

response . . . ~ . . . \$(r, \theta)\$

Example from ARC-Challenge:

George wants to warm his hands quickly by rubbing them. Which skin surface will produce the most heat?

A. Dry palms
B. Wet palms
C. Palms covered with oil
D. Palms covered with lotion

GRPO (from DeepSeek)

GRPO:

- q: question from dataset.
- o_i: i-th generated outputs from old policy model θ_{old}
- $\underbrace{G\text{: group size}}$
- ϵ: clipping hyperparameter
- r_i: reward for o_i
- A_i: group advantage for o_i
- D_{KL}: KL divergence between current policy θ and reference policy θ_{ref}.

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^{G} \sim \pi_{\theta_{old}}(O|q)]$$

$$\frac{1}{G}\sum_{i=1}^{G}\left(\min\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}A_i, \text{clip}\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}, 1-\varepsilon, 1+\varepsilon\right)A_i\right) - \beta\mathbb{D}_{KL}\left(\pi_\theta||\pi_{ref}\right)\right)$$

$$\mathbb{D}_{KL}\left(\pi_\theta||\pi_{ref}\right) = \frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - \log\frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - 1,$$

$$\boxed{A_i = \frac{r_i - \text{mean}(\{r_1, r_2, \cdots, r_G\})}{\text{std}(\{r_1, r_2, \cdots, r_G\})}.}$$

Rubrics-based Reward

Rubrics-Based Reward Example

Task

Question:
If a train travels **120 km in 2 hours**, what is its **average speed**?

Model Output

> The average speed is **60 km/h** because speed equals **distance divided by time**.

Evaluation Rubric

Criterion	Description	Score Range
Correctness	Final answer is correct	0–5
Reasoning	Explanation is logically sound	0–3
Clarity	Answer is clear and concise	0–2

Total possible score = 10

prompt to $\underbrace{\text{LLM}}_{\text{LLM as a judge}}$

Scoring the Output

Criterion	Score	Reason
Correctness	5	$120/2 = 60$
Reasoning	3	Correct formula used
Clarity	2	Concise explanation

$$\textbf{Reward} = 5 + 3 + 2 = 10$$

Reward Used for RL

$$r(x, y) = \frac{10}{10} = 1.0$$

instruction following

RL Data

Olmo 3

LLM as judge

Category	Prompt Dataset	# Prompts Used in Think RL	# Prompts Used in Instruct RL	Reference
Precise IF	IF-RLVR	30,186	38,000	Pyatkin et al. (2025)
Math	Open-Reasoner-Zero	3,000	14,000	Hu et al. (2025)
	DAPO-Math	2,584	7,000	Yu et al. (2025)
	AceReason-Math	6,602	–	Chen et al. (2025)
	Polaris-Dataset	–	14,000	An et al. (2025)
	KlearReasoner-MathSub	3,000	9,000	Su et al. (2025c)
	OMEGA-train	15,000	20,000	Sun et al. (2025)
Coding	AceCoder	9,767	20,000	Zeng et al. (2025a)
	KlearReasoner-Code	8,040	–	Su et al. (2025c)
	Nemotron Post-training Code	2,303	–	NVIDIA AI (2025)
	SYNTHETIC-2	3,000	–	PrimeIntellect (2025)
General Chat	Tulu 3 SFT	7,129	18,955	Lambert et al. (2024)
	Wildchat-4.8M	7,129	18,761	-
	Multi-Subject RLVR	7,129	12,234	Su et al. (2025b)
Total		104,869	171,950	