Fundamentals of Reinforcement Learning

Luka Meladze
National University of Singapore
Project: **LLM Post-training with Reinforcement Learning**

Contents

Abstract

This paper provides an overview of key introductory concepts in Reinforcement Learning (RL). It defines the Markov Decision Process (MDP) and its components, elaborates on the distinctions between RL and Supervised Learning (including Imitation Learning) and describes a real-world application using RL terminology. Finally, the paper provides a mathematical formulation of the RL objective function under both finite and infinite horizon settings and provides a detailed explanation of the derivation of policy gradients, including methods for reducing variance and a simplified algorithmic outline for practical implementation.

1 Reinforcement Learning

1.1 Markov Decision Process (MDP)

A **Markov Decision Process (MDP)** is a framework used to describe sequential decision-making problems, where a decision-maker (agent) interacts with an environment over time. At each time step t, the agent observes an observation of the state of the world s_t (possibly partial o_t), takes an action a_t, and then receives a reward r_t while making a transition to the new state s_{t+1}. This cycle continues, and the agent's ultimate goal is to find a **policy** (a strategy for choosing actions) that maximises the total expected reward.

The key idea and assumption of MDP is rooted in the **Markov property**, which states that the next state depends **only** on the current state and action, not on the history of previous states or actions. That doesn't mean that the future is perfectly predictable; there might still be randomness, but knowing the past doesn't help at all to resolve that randomness.

An MDP is defined by a **5-tuple**: $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$.

- **States ($\mathcal{S}$)**: The set of all possible states the environment of an agent can be in. Each state $s \in \mathcal{S}$ is a **complete** description of the environment, while an observation o is a partial description of a state. A state s must satisfy the **Markov Property**: If the current state s_t is given, the future state s_{t+1} and future rewards are independent of past states $(s_1, \ldots, s_{t-1})$ and actions $(a_1, \ldots, a_{t-1})$. Mathematically, $P(s_{t+1}|s_t, a_t, \ldots, s_1, a_1) = P(s_{t+1}|s_t, a_t)$.
- **Actions ($\mathcal{A}$)**: The set of all possible & valid actions (moves or decisions) that the agent can make in any state is often called the action space. Some environments have discrete action spaces, while others have continuous action spaces where actions are real-valued vectors.
- **Transition Probability Operator/Function(T)**: It is a **tensor** which specifies the probability of transitioning from the current state s to the next state s' after taking action a. $T_{i,j,k} = p(s_{t+1} = i|s_t = j, a_t = k)$ also denoted as $P(s'|s, a)$. The environment's response to an action can be **deterministic** (always leads to the same next state) or **stochastic** (leads to different next states).
- **Reward Function (R)**: The reward function R is very important in reinforcement learning, which is a mapping from $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. It depends on the current state of the world and the action just taken. It quantifies the desirability of an action or state, typically denoted as $R(s)$ or $R(s, a)$. The agent's ultimate goal in RL is to maximise ***not** the immediate* but the *cumulative* reward over time (over a trajectory).
- **Discount Factor (γ)**: For tasks that continue indefinitely, the cumulative sum of rewards could be infinite. We need ways to ensure the objective remains finite. Discount Factor is a value between 0 and 1 ($\gamma \in [0, 1)$ typically 0.9 to 0.99) which weights rewards received sooner more heavily than rewards received later to guarantee that the infinite sum converges to a finite value.

1.2 How RL Differs from Supervised / Imitation Learning

Reinforcement Learning (RL) and Supervised Learning (SL), including Imitation Learning (IL), learn in fundamentally different ways and are distinguishable by their data sources, goals, inherent assumptions, and feedback mechanisms. The Supervised Learning works fine under the assumption that each data example is Independent and Identically Distributed. **Imitation Learning (Behavioural Cloning)** is a form of Supervised Learning applied to sequential decision-making. It trains a policy π_θ to mimic an expert's actions. However, IL inherits SL's limitations, particularly the distributional shift problem, meaning it cannot guarantee optimal performance outside the expert's observed trajectory space and cannot discover behaviours superior to the expert's. RL directly addresses these limitations by learning through interaction and optimising for long-term outcomes.

- In supervised learning, including imitation learning, the agent learns from a fixed dataset (**Independent and Identically Distributed data**) of input-output pairs $\mathcal{D} = \{(X_i, Y_i)\}_{i=1}^{N}$. The model learns from these static examples and receives direct, immediate, and explicit "ground truth" labels (correct answers) for each input.

  In contrast, reinforcement learning does not assume access to an expert or labelled actions. Instead, the agent sequentially interacts with the environment directly. It observes the state, selects actions, and receives rewards based on those actions. The agent's goal is not to imitate an expert but to maximise the total cumulative reward it receives over time.
- **Learning Objective**: In supervised learning, the goal is to generalise patterns from given data and copy the behaviour or labels. i.e. learn mapping $f_\theta : \mathcal{X} \to \mathcal{Y}$ that accurately predicts outputs for unseen inputs and minimises prediction error.
  In Reinforcement Learning, the goal is to learn a **policy** $\pi_\theta : \mathcal{S} \to \mathcal{A}$ that maximises the **total future reward** over a sequence of interactions. It's about figuring out what the agent should do to reach a goal, potentially discovering solutions better than any human expert could show (like AlphaGo's surprising "Move 37").
  It maximises $J(\theta) = \mathbb{E}_{\tau \sim P_\pi(\tau)}\left[\sum_{t=1}^{T} \gamma^t R(s_t, a_t)\right]$.

1.3 The Objective of Reinforcement Learning in Math

The fundamental objective in Reinforcement Learning is to find an **optimal policy (π^*)** that is defined by parameters (θ^*) and maximises the **expected cumulative reward** (also known as "return") over time. This expectation accounts for the stochasticity of both the environment and the agent's policy.

1.3.1 Finite Horizon (Total Reward)

For tasks that have a fixed, finite number of time steps T, the objective is to maximise the expected sum of rewards the agent collects until the end of completing the task.

The **return** for a single trajectory $\tau = (s_1, a_1, \ldots, s_T, a_T, s_{T+1})$ is:

$$G_0 = \sum_{t=1}^{T} R(s_t, a_t)$$

The **objective function**, which we aim to maximise by optimising the policy's parameters θ, is the **expected return**:

$$J(\theta) = \mathbb{E}_{\tau \sim P_\pi(\tau)}\left[\sum_{t=1}^{T} R(s_t, a_t)\right]$$

Where:

- $P_\pi(\tau)$: The probability of a specific trajectory τ occurring under policy π_θ:

$$P_\pi(\tau) = P(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) P(s_{t+1}|s_t, a_t)$$

- $R(s_t, a_t)$: The immediate reward received at time t in state s_t taking an action a_t.

An equivalent formulation for finding the optimal policy parameters θ^* in the finite horizon case is to maximise the sum of expected rewards at each time step. Here, $p_\theta(s_t, a_t)$ represents the probability of visiting state s_t and taking action a_t at time t under policy π_θ.

$$\theta^* = \arg\max_\theta \sum_{t=1}^{T} \mathbb{E}_{(s_t,a_t)\sim p_\theta(s_t,a_t)}[R(s_t, a_t)]$$

1.3.2 Infinite Horizon (Average Reward)

For tasks where the agent operates continuously without a natural ending point and without a discount factor, the objective is to maximise the long-term **average** reward per time step.

The **average reward objective** is defined as:

$$J_{avg}(\theta) = \lim_{T\to\infty} \frac{1}{T} \mathbb{E}_{\tau\sim P_\pi(\tau)} \left[\sum_{t=1}^{T} R(s_t, a_t) \right]$$

When the system converges to a stationary distribution $p_\theta(s, a)$, this objective simplifies to the expected immediate reward under that distribution:

$$J_{avg}(\theta) = \mathbb{E}_{(s,a)\sim p_\theta(s,a)}[R(s, a)]$$

The optimal policy parameters θ^* for the average reward case are therefore found by maximising this expected immediate reward under the stationary distribution.

$$\theta^* = \arg\max_\theta \mathbb{E}_{(s,a)\sim p_\theta(s,a)}[r(\mathbf{s}, \mathbf{a})]$$

1.3.3 Infinite Horizon (Discounted Total Reward)

For continuous tasks without a natural ending point, we also use a **discounted sum of rewards** to ensure the total return remains finite and converges.

The **discounted return** for a single trajectory $\tau = (s_1, a_1, s_2, a_2, \ldots)$ is:

$$G_0 = \sum_{t=1}^{\infty} \gamma^t R(s_t, a_t)$$

The **objective function** to maximise is the **expected discounted return**:

$$J(\theta) = \mathbb{E}_{\tau\sim P_\pi(\tau)} \left[\sum_{t=1}^{\infty} \gamma^t R(s_t, a_t) \right]$$

Where:

- γ: The **discount factor**, a value in $[0, 1)$ that exponentially weighs future rewards less than immediate ones.

The ultimate goal in Reinforcement Learning is to find the policy parameters θ^* that maximise this objective function:

$$\theta^* = \arg\max_\theta J(\theta)$$

This means we are searching for the best possible strategy that, on average, leads to the highest accumulation of rewards over the long run, considering both the agent's actions and the environment's stochastic (random) nature.

1.4 Policy Gradients

1.4.1 Deriving the Basic Policy Gradient

To achieve the main goal in Reinforcement Learning, i.e. to find the best settings (parameters θ) for a policy π_θ so that the average total reward $J(\theta) = \mathbb{E}_{\tau \sim P_\pi(\tau)} \left[\sum_{t=1}^{T} R(s_t, a_t) \right]$ is as high as possible, policy gradient methods are used that directly optimize a parameterized policy. The underlying principle of policy gradients involves iteratively adjusting policy parameters to increase the probabilities of actions that lead to higher returns while simultaneously decreasing the probabilities of actions that result in lower returns. This iterative refinement process inherently embodies a "trial and error" (Levine, 2021) learning paradigm, where the agent continuously adapts its policy based on observed outcomes. The optimisation of $J(\theta)$ is achieved through an iterative process known as **gradient ascent**, that calculates "steepest way up" given by the **gradient** of the objective function, $\nabla_\theta J(\theta)$. Once this gradient is known, policy parameters θ are adjusted in that direction:

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Here, α (alpha) is a small step size, called the **learning rate**.

The challenge is to calculate $\nabla_\theta J(\theta)$ because the trajectory τ depends on the rules of the environment, such as the unknown transition probabilities $P(s_{t+1}|s_t, a_t)$ or initial state $P(s_1)$. We can only interact with the world to collect samples. Mathematically, $J(\theta)$ can be expressed as an average over all possible trajectories τ:

$$J(\theta) = E_{\tau \sim p_\theta(\tau)}[r(\tau)] = \int p_\theta(\tau) r(\tau) d\tau$$

Next, taking the gradient with respect to θ, since the gradient operation is linear, it can be moved inside the integral:

$$\nabla_\theta J(\theta) = \nabla_\theta \int p_\theta(\tau) r(\tau) d\tau = \int \nabla_\theta p_\theta(\tau) r(\tau) d\tau$$

Now, a very useful trick called the "convenient identity" (log-derivative trick) is employed. This trick states that for any probability distribution $P(\tau)$:

$$\nabla_\theta P(\tau) = P(\tau) \nabla_\theta \log P(\tau)$$

Applying this convenient identity is useful because it essentially converts a gradient of a probability into a probability times the gradient of its logarithm. The equation becomes:

$$\nabla_\theta J(\theta) = \int p_\theta(\tau) \nabla_\theta \log p_\theta(\tau) r(\tau) d\tau = E_{\tau \sim p_\theta(\tau)}[\nabla_\theta \log p_\theta(\tau) r(\tau)]$$

Now, let's determine $\nabla_\theta \log p_\theta(\tau)$. The probability of a trajectory is defined as:

$p_\theta(\tau) = p(s_1) \prod_{t=1}^{T} \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t)$

Taking the logarithm of both sides (which turns products into sums) gives the following equation:

$$\log p_\theta(\tau) = \log p(s_1) + \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \sum_{t=1}^{T} \log p(s_{t+1}|s_t, a_t)$$

A crucial simplification occurs when taking the gradient ∇_θ with respect to θ, since the terms $\log p(s_1)$ (initial state distribution) and $\log p(s_{t+1}|s_t, a_t)$ (environment dynamics & transition probabilities) vanish, as they are independent of the policy parameters θ. This is because the environment's rules do not depend on the policy parameters θ. So, only the policy term remains:

$$\nabla_\theta \log p_\theta(\tau) = \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

This means the gradient computation now only requires differentiating the known, parameterised policy $pi_\theta(a_t|s_t)$, not the unknown environment. Substituting this back into the policy gradient equation yields the fundamental **basic policy gradient formula** (the REINFORCE Algorithm). This seminal algorithm was formally introduced by (Williams, 1992), providing a foundational method for estimating the policy gradient through a sample mean derived from collected trajectories.

$$\nabla_\theta J(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \right) \left(\sum_{t=1}^{T} r(s_t, a_t) \right) \right]$$

In practice, since this expectation cannot be calculated perfectly, it is estimated **(unbiased)** by running the policy π_θ in the environment N times and collecting many trajectories. Then, for each sampled trajectory, the calculated terms are averaged to get an estimate of the gradient. So, the **MONTE Carlo approximation formula for policy gradient (MIT-6.7920, 2024)**:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \right) \left(\sum_{t=1}^{T} r(s_{i,t}, a_{i,t}) \right) \right]$$

1.4.2 Adding a Baseline for the Policy Gradient and Why It is Needed

The basic policy gradient, while mathematically correct, has a significant practical issue: it can be very **noisy** and have **high variance** (Afachao, 2023). This means that if a small number of trajectories are collected, the estimate of the gradient can jump around a lot, making it hard for policy parameters to learn & improve consistently. Additionally, if all the rewards are positive, even for ”bad” trajectories, the basic policy gradient method will try to increase the probability of all sampled actions, simply because their reward is positive. This goes against the intuition that the goal is to increase probabilities for ”better than average” actions and decrease them for ”worse than average” ones.

To avoid this problem and make learning more stable and efficient, two key tricks, reward-to-go and baselines (Afachao, 2023) help reduce the **high variance** of policy gradients:

Exploiting Causality - ”Reward-to-Go” ($\hat{Q}_t$) An action taken at time t cannot affect rewards that were received in the past (before time t) (Schulman, 2016). This is the principle of **causality**. So, when evaluating how good an action a_t was, only the rewards that came **after** that action, from time t until the end of the trajectory, need to be considered. This sum of future rewards is called the **reward-to-go**:

$$\hat{Q}_t = \sum_{t'=t}^{T} r(s_{t'}, a_{t'})$$

Using this reward-to-go term instead of the total reward from the entire trajectory reduces noise by discarding irrelevant past rewards. The policy gradient now looks like:

$$\nabla_\theta J(\theta) \approx E_{\tau \sim p_\theta(\tau)} \left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \hat{Q}_t \right]$$

Baselines $b(s_t)$ To further reduce noise and make the learning more intuitive, a value called a **baseline**, $b(s_t)$, is subtracted from the reward-to-go term. This baseline typically represents the average or expected reward-to-go from a given state s_t. The use of such a baseline is formally justified by the EGLP (Expected Gradient of Log Policy) Lemma, which states that adding a baseline $b(s_t)$ that is independent of the action a_t does not change the expectation of the policy gradient, while significantly reducing its variance (OpenAI, Achiam, 2018).

By using a baseline, the actual reward received ($\hat{Q}_t$) is essentially compared to what was expected from that state ($b(s_t)$).

- If $(\hat{Q}_t - b(s_t))$ is positive, it means the action taken led to a **better-than-expected** outcome. So, the policy will be adjusted to make that action more likely.
- If $(\hat{Q}_t - b(s_t))$ is negative, it means the action led to a **worse-than-expected** outcome. So, the policy will be adjusted to make that action less likely.

This ”centering” provides a more stable learning signal, leading to more reliable policy improvement. So, the formula (MIT-6.7920, 2024) with reward-to-go and a baseline is:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \left(\hat{Q}_{i,t} - b\right)$$

1.4.3 On-policy algorithm for the Policy Gradient

This algorithm for the policy gradient outlines the steps for how an agent learns using policy gradients with a basic baseline. It focuses on the ”trial and error” learning process that policy gradients formalise, allowing an agent to learn complex behaviours by continually refining its strategy based on the outcomes it experiences.

Algorithm: Policy Gradient Method (Simplified version of REINFORCE algorithm (Achiam, 2018))

1. **Input**:
   - Initial policy parameters θ_0 (the starting settings for the ”brain”)
   - Learning rate α (how big of a step to take when adjusting settings)
2. **For** $k = 0, 1, 2, \ldots$ (repeat for many learning cycles) **do**:
   (a) **Collect Trajectories (Run the Policy)**:
      - Use the current policy $\pi(\theta_k)$ to interact with the environment.
      - Collect a set of trajectories $\mathcal{D}_k = \{\tau_i\}_{i=1}^{N}$ (many paths the robot takes). Each τ_i includes a sequence of states, actions, and rewards: $(s_{i,1}, a_{i,1}, r_{i,1}, s_{i,2}, a_{i,2}, r_{i,2}, \ldots, s_{i,T}, a_{i,T}, r_{i,T})$.

   (b) **Compute Rewards-to-Go ($\hat{Q}_t$)**:
      - For each time step t in every collected trajectory $\tau_i \in \mathcal{D}_k$, calculate the reward-to-go $\hat{Q}_{i,t}$:

$$\hat{Q}_{i,t} = \sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})$$

(c) **Compute Simple Baseline (b)**:

- Calculate the average of all $\hat{Q}_{i,t}$ values across all time steps in all trajectories in the current batch $\mathcal{D}_k$. This average will be our baseline b:

$$b = \frac{1}{\sum_{\tau \in \mathcal{D}_k} |\tau|} \sum_{\tau \in \mathcal{D}_k} \sum_{t=1}^{|\tau|} \hat{Q}_{i,t}$$

(where $|\tau|$ is the length of trajectory τ)

(d) **Estimate Policy Gradient ($\hat{g}_k$)**:

- Calculate the estimated policy gradient $\hat{g}_k$ using the collected data, reward-to-go values, and the baseline:

$$\hat{g}_k = \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})(\hat{Q}_{i,t} - b)$$

(e) **Update Policy Parameters**:

- Adjust the policy parameters θ by taking a step in the direction of the estimated gradient:

$$\theta_{k+1} = \theta_k + \alpha \hat{g}_k$$

2 Overview and Deep Dives into LLMs like ChatGPT

This section provides an overview of the principles, architecture, and training methodologies behind Large Language Models (LLMs) like ChatGPT and the role of Reinforcement Learning (RL) in improving their outputs. The primary source of the concepts and interesting LLM insights mentioned below is the YouTube Video by Andrej Karpathy (Karpathy, 2025). The report outlines the entire LLM pipeline, from the initial pre-training on vast internet datasets to the post-training stages of Supervised Fine-Tuning (SFT) and RL. The SFT models have inherent computational limitations, and we will explore the specific reasons ”why?” they excel at certain (even very hard for humans) tasks while failing at much simpler ones (for humans) like counting or spelling. Finally, we will see how RL is used to unlock reasoning capabilities, moving these models from simple text predictors to more sophisticated problem-solving agents.

2.1 Base-Model

Before a model can learn to ”think” for itself or answer questions to become an ”assistant”, it must first process a huge amount of diverse data in different languages and knowledge sources. This training is a two-stage process: First, training a base model, and then turning this base model into a sophisticated assistant.

- 1.1. The first step in pre-training is data curation, usually from the public internet. The raw data is noisy and requires filtering like URL filtering, Text Extraction, Language Filtering, and PII Removal. After this extensive cleaning, the vast internet is condensed into a more manageable & accurate dataset. For instance, the FineWeb dataset contains approximately 44 terabytes of text (15 trillion tokens), which can then be used as the ”textbook” from which the model will learn and be trained. (Penedo et al., 2024)
- 1.2 Tokenisation: Neural networks do not process raw text or bytes. They process sequences of numbers, i.e. tokens (OpenAI, 2025). Tokenisation is the crucial step of converting a stream of text into a sequence of integers (token = Unique ID). Instead of using individual characters (which would create very long sequences) or whole

words (which would create an unlimited vocabulary), LLMs use an algorithm like **Byte-Pair Encoding (BPE)** (Hugging Face, 2025) to find commonly occurring chunks of text and assign them a unique token ID. This results in a fixed vocabulary. Tokenisation is important for understanding LLM limitations. The model does not see individual letters; it sees a sequence of text chunks (tokens).

- 1.3 This tokenised data from a diverse collection of texts is then used to train a massive neural network (a Transformer). The initial base model's **objective** is simple: to predict the next token in a sequence. To train an initial base model:
  - The model is given a chunk of text (a "context window"), for example, "The capital of **Georgia** is". (Of course, using tokens like [464, 3139, 286, 7859, 318])
  - It then predicts the most probable next token (in this case, likely the tokens for " Tbilisi"). The model makes its prediction, which is initially random.
  - It compares its prediction to the actual next token in the training data and then, using a process called backpropagation, it calculates its error and slightly adjusts its internal parameters (billions of "weights") to make its prediction for that specific example slightly better next time.

  By doing this trillions of times in parallel (using tens of thousands of GPUs running for months), the model develops a base model—a massive neural network (like Meta's Llama 3 or GPT-4's base) whose parameters have been tuned to be a "internet document simulator", but is not yet a helpful assistant. If you prompt it, it will simply continue your text as if completing a webpage it found online.

2.2 From Base-Model to Assistant - Supervised Fine-Tuning (SFT)

The base model can only statistically complete texts but is not capable of having conversations and answering questions. To make it a helpful assistant, we need to teach it a new behaviour: how to have a conversation. This is done by further using new tokens specifically for signalling the start & end of conversations, and training models on a much smaller, high-quality dataset of example conversations. In these conversations, human labellers & experts (following detailed instructions & templates) have written "ideal" responses to a wide variety of prompts. It's not learning new factual knowledge (that came from pre-training), but rather a new style of interaction. So, when we ask LLM simple models like GPT-2 or GPT-4o a question, they essentially give back the style of a statistical simulation of what an average human labeller, following OpenAI's guidelines, would have written.

2.3 Errors & Limitations: Why LLMs Can't Spell or Count Characters?

2.3.1 Models are not good at "mental" computations

To generate each token, the model performs a single forward pass through its neural network. This means it takes the current sequence of tokens as input and, after a fixed number of computational layers (e.g., 100 layers), outputs the probabilities for the next token. The critical implication is that the model has a limited computational budget for "thinking" about each token it produces. It cannot loop, ponder indefinitely, or perform a complex multi-step calculation in its "head" before producing a single token. All reasoning & computation must be spread across the multiple steps of generating next tokens.

2.3.2 Why LLMs Can't Spell or Count Characters?

The two characteristics of LLMs: 1. Token-based view & 2. finite computation per token, cause them to have some holes in performing basic tasks, even though they are usually very good at much more advanced problem-solving. As models don't see letters, and they see tokens, the words might be seen as the tokens of text chunks that the model must first recall (from its training), then has to recall the "length" or symbols of these tokens and e.g. add them up. This is a multi-step task that is not native to its architecture, especially in one pass. It's doing "mental arithmetic" on the composition of tokens and often fails. Example prompts where it fails due to tokenisation include **counting** the number of dots, spelling, etc. This is why asking it to use a tool like the Python code interpreter often works: the model can better copy-paste the string (easy task for LLM) and let Python do the actual counting.

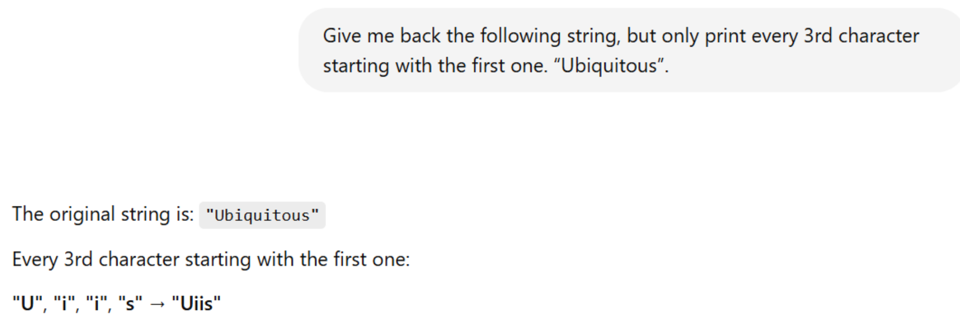

Figure 1: GPT-4o

2.4 Reinforcement Learning (RL) for LLMs

While SFT creates a competent assistant, it's limited by imitation and its training data. The model only stochastically imitates how a human would write a solution based on the internet data. It doesn't truly "solve" the problem itself. RL addresses this by allowing the model to practice. Instead of just imitating a "perfect" (expert-based, training data-driven) solution, the model generates thousands of its own potential solutions and is rewarded for the ones that reach the "correct" or "better" answers. This practice of models without human supervision is important because the optimal reasoning path for an LLM (a sequence of tokens) is often different from a human's.

2.4.1 RL in Verifiable Domains (e.g. Math & Code)

In domains with a clear right or wrong answer (like math or coding), the process of RL is straightforward. For a single prompt (e.g., a math problem), the model generates many different potential solutions (e.g., 100 different chains of thought). A model then automatically checks which of these solutions arrive at the correct final answer, and the final step is to reinforce, i.e. the model is then trained and its parameters are updated to increase the probability of generating the successful token sequences. A key paper in this area, DeepSeek-R1 (DeepSeek-AI et al., 2025), showed that this process leads models to distribute their "thinking" over many tokens: It writes out intermediate steps, re-evaluates its work (e.g. "Wait, let me double-check that"), and tries multiple approaches. So, RL discovers reasoning strategies that are effective for models' architectures, even if a human wouldn't have written them that way.

2.4.2 RL in Unverifiable Domains like writing a joke (RLHF)

For creative tasks like writing a joke, where there's no single "correct" answer, Reinforcement Learning happens from Human Feedback (RLHF). The model generates several responses to a prompt (e.g., X different jokes). Then a human labeller ranks these responses from best to worst. However, instead of using humans to directly grade the LLM's performance billions of times for each sequence of responses, we use humans for a much smaller task to rank maybe 10,000 to 100,000 sets of responses and eventually train an AI judge model that can instantly rank given responses based on how a human would likely perceive it (by using same pre-training and post-training steps as mentioned above for LLM models). This judge, called a Reward Model, is a separate neural network and it can be used billions of times automatically. Its only job is to learn to predict human's preferences and is rewarded for getting the relative ordering correct.

Limitations of RLHF: While powerful, RLHF has a significant flaw. The reward model is just a simulation of a human, and like any neural network, it can be ”gamed.” If you run the RL process for too long, the LLM will find ”adversarial examples”—nonsensical outputs that trick the reward model into giving a high score. For this reason, RLHF is more of a ”fine-tuning” step to slightly improve model alignment and helpfulness, rather than a method for indefinite capability improvement like the RL used in verifiable domains. This is what the video means when it says ”RHLF is not RL in the magical sense,” unlike the systems like AlphaGo (Silver et al., 2016) which could practice indefinitely against a perfect game simulator to achieve superhuman performance.

2.5 Stages from raw internet predictor to assistant

The creation of a large language model is a multi-stage process, transforming a raw internet text predictor (base model) into a sophisticated, reasoning assistant.

- Pre-training gets training data and builds knowledge by predicting the next token across trillions of examples.
- SFT teaches the model how to converse like a helpful assistant by imitating human-written conversations.
- RL allows the model to move beyond imitation and discover effective, sometimes novel, reasoning strategies through trial-and-error practice.

However, it is important to remember the model's nature. It is a stochastic, token-based system with a finite computational budget for each step. Its knowledge has ”mistakes,” and its reasoning can be broken. It does not ”know” things in the human sense but is a master of statistical pattern-matching. As such, LLMs should be treated as incredibly powerful tools—for brainstorming, for first drafts, for summarising, and for accelerating work—but their output should be verified by a human user who remains ultimately responsible for the final product.

3 InstructGPT: Training LLM by RLHF

Building on the foundational RL principles and the general LLM training pipeline, this section provides an analysis of the OpenAI paper ”Training language models to follow instructions with human feedback” (Ouyang et al., 2022). We will explore the specific procedures and results of the InstructGPT study, with a focus on a comparative analysis of the different models developed and the key limitations derived from their performance.

3.1 RL Application in InstructGPT Development

The InstructGPT model is developed through a three-step process that applies Reinforcement Learning principles to the (**unverifiable:** see section 2.4.2) domain of human language. This Reinforcement Learning from Human Feedback (RLHF), aims to maximally align the model's outputs with user intent while being helpful and honest, a goal not inherently captured by the standard next-token prediction objective of **base** language models. The process begins with a pretrained GPT-3 model and takes 3 main steps.

1. **Supervised Fine-Tuning (SFT).** The first step is to create a baseline for instruction-following behavior. A dataset of high-quality responses is collected, where human labelers write the desired output for a given prompt (Ouyang et al., 2022). A pre-trained GPT-3 model is then fine-tuned on this demonstration data using standard supervised learning techniques. This SFT model learns the style and format of a helpful assistant but is limited by the scope of the labelers demonstration data.

2. **Reward Model (RM) Training.** A key difference from fully human super-vised model or verifiable domains is the absence of a precisely predefined reward function. Instead of asking humans to evaluate responses, a **Reward Model (RM)** is trained to predict human preferences. To move beyond simple imitation, the system needs to learn a generalized notion of what humans prefer. To achieve this, a dataset of human ranking is collected and then used to train a separate Reward Model (RM). Labelers rank model outputs for a given prompt and then the RM uses a reward to update the policy. The RM's loss function in the study was designed to maximize log-likelihood of matching human preferences & ranking.

3. **Proximal Policy Optimization (PPO).** In the final step, the SFT model from Step 1 is further fine-tuned using the Proximal Policy Optimization (PPO) algorithm. In this RL setup, the policy generates a response, and the **RM from Step 2** calculates a reward for that response. To prevent the RL policy from deviating too far (so called "performance-regression" challenge in the study) from the distribution learned during SFT and over-optimizing for the RM, a per-token KL penalty was added to the reward function but the study still reveals the **"alignment tax"** challenge of using PPO. While the standard PPO model excelled on API tasks, its performance on several public NLP benchmarks *regressed* compared to the original GPT-3 model. This trade-off between alignment and capability motivated the development of the final model, PPO-ptx. The objective function for this model was modified to include a term from the pretraining data distribution, effectively "reminding" the model of its **foundational** knowledge.

3.2 Interesting insights about Model Performance

- Alignment can be more valuable than scale: The study reveals that a smaller model fine-tuned with human feedback can significantly outperform a much larger, un-aligned model. For instance, the 1.3B parameter InstructGPT model was preferred by human labelers over the 175B parameter base GPT-3 model. This highlights that the method of fine-tuning is a more critical factor for user satisfaction than a 100x increase in model parameters alone. But of course, there are some trade-offs too when solely focusing on human preferences.

- All fine-tuned models hallucinate less, but RL models are the most reliable: The base GPT-3 model had a high rate of making up information ("hallucinating") in closed-domain tasks, at around 41%. While all fine-tuning methods improved this, the PPO models showed the greatest reduction in hallucinations. The PPO models were also rated as being significantly better at following explicit constraints within a prompt compared to both SFT and GPT-3 baselines.

3.3 Research Questions and Limitations

The paper is clear about the limitations of its approach, and it providing insights and a motivation to research the open directions in this area. There are several directions that may be explored for future research.

3.3.1 Limitations of InstructGPT & Future Directions

- The models, with their heavy focus on human alignment, may often prioritize helpfulness at the expense of harmlessness. This creates a critical need to balance these objectives and avoid generating toxic, biased, or factually incorrect outputs. The InstructGPT paper suggests that one path forward is adversarial data collection, where labelers actively find the model's worst-case behaviors to add to the training data. Another research has explored more scalable solutions, such as **Constitutional AI**, where a model learns from AI-generated feedback based on a set of fundamental principles (a "constitution"). This method trains the model to be harmless by having it critique and revise its own outputs according to the constitution, significantly reducing the need for human labeling of safety-critical prompts (Bai et al., 2022).

- The model's alignment is based on a small, non-representative group of 40 paid labelers and the researchers who wrote their instructions. This raises significant questions about whose values are being encoded and deployed. Given the limited and expensive nature of human annotation, this bottleneck has spurred research into minimizing direct human involvement. A promising direction is **Reinforcement Learning from AI Feedback (RLAIF)**, where preferences are generated by a powerful "teacher" LLM instead of humans. Studies have shown that RLAIF can achieve performance comparable to or even exceeding RLHF, suggesting a scalable method to generalize preferences and reduce the dependency on intensive human data collection (Lee et al., 2023).
- The models can be easily confused by instructions that contain false premises, often accepting the premise as true and generating factually incorrect answers. This issue of "hallucination" is a major barrier to using LLMs in high-stakes domains. To address this, research is moving beyond standard Outcome Reward Models (ORMs), which only score the final answer ($R_{ORM} = r(x, y_{final})$), towards **Process-based Reward Models (PRMs)**. PRMs provide a reward for each correct step in a model's reasoning chain, with the total reward being a sum over the individual rewards for each intermediate step, y_i:

$$R_{PRM} = \sum_{i=1}^{N} r(x, y_{1..i-1}, y_i)$$

  By rewarding the *process* of arriving at an answer, this method encourages the model to produce more faithful and verifiable outputs (Lightman et al., 2023). However, developing these PRMs is not trivial; very recent works research about significant challenges in evaluation and propose new mechanisms to mitigate biases and improve the effectiveness (Zhang et al., 2025).

References

Joshua Achiam. Vanilla policy gradient — spinning up documentation, 2018. URL https://spinningup.openai.com/en/latest/algorithms/vpg.html#pseudocode.

Kevin Afachao. Deep dive into reinforcement learning: Policy gradient algorithms. *Medium*, 2023. URL https://medium.com/@thekevin.afachao/deep-dive-into-reinforcement-learning-policy-gradient-algorithms-bbeabe2fc989. Accessed: 2025-05-23.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, , et al. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*, 2022.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, et al. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning, 2025.

Hugging Face. Byte-Pair Encoding (BPE) Tokenization, in: The LLM Course. https://huggingface.co/learn/llm-course/en/chapter6/5, 2025.

Andrej Karpathy. Deep dive into llms like chatgpt, 2025. URL https://www.youtube.com/watch?v=zjkBMFhNj_g.

Harrison Lee, Sam Phatale, Kyo Tworkowski, Kexin Zhang, Abel Mainsah, and Vineet Kosaraju. Rlaif: Scaling reinforcement learning from human feedback with ai feedback. *arXiv preprint arXiv:2309.00267*, 2023.

Sergey Levine. Policy gradients (uc berkeley cs 285 lecture notes), 2021. URL https://rail.eecs.berkeley.edu/deeprlcourse/.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

MIT-6.7920. Policy gradient. https://web.mit.edu/6.7920/www/lectures/L13&14-2024fa-PolicyGradient.pdf, 2024. Lecture notes for MIT 6.7920: Reinforcement Learning, Fall 2024.

OpenAI. What are tokens and how to count them? https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them, 2025.

OpenAI, Achiam. Spinning up in deep rl. https://spinningup.openai.com/en/latest/spinningup/rl_intro3.html, 2018. Accessed: 2025-05-23.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022. URL https://arxiv.org/abs/2203.02155.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

John Schulman. *Optimizing Expectations: From Deep Reinforcement Learning to Stochastic Computation Graphs*. PhD thesis, University of California, Berkeley, 2016. Specifically Chapter 2, as referenced in OpenAI Spinning Up documentation.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992. doi: 10.1007/BF00992696. URL https://link.springer.com/article/10.1007/BF00992696.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning. *arXiv preprint arXiv:2501.07301*, 2025. URL https://arxiv.org/abs/2501.07301.