IN5640: Large Language Models Development
Post-training: Supervised fine-tuning and reinforcement learning

David Samuel

Language Technology Group, University of Oslo

October 19, 2026



Contents

1. Introduction
   - Recap: Pretraining of language models
   - LLM lifecycle
   - Post-training of language models
2. Supervised finetuning
3. Reinforcement learning
4. Where does the reward come from?
5. Direct preference optimization
6. Reasoning and tool use
7. Summary

Recap: Pretraining of language models

Statistical models $p_\theta(\cdot)$ of language distribution in some training corpus $\mathcal{D}$.

- The training corpus consists of sequences $\boldsymbol{x} = (x_1, x_2 \dots x_N) \in \mathcal{D}$ of tokens.

$$p_\theta(\textit{That dog barks.}) = 0.001\%$$
$$p_\theta(\textit{That dog meows.}) = 0.00001\%$$
$$p_\theta(\textit{meow this Dogs.}) = 0.000000001\%$$

- The training corpus consists of sequences $\boldsymbol{x} = (x_1, x_2 \dots x_N) \in \mathcal{D}$ of tokens
- The language models are trained by finding such parameters $\boldsymbol{\theta}$ that maximize the (log-)likelihood of the corpus – maximum likelihood estimation (MLE):

$$\operatorname*{argmax}_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{x} \sim \mathcal{D}} \Big[\log\, p_{\boldsymbol{\theta}}(\boldsymbol{x}) \Big]. \tag{1}$$

Recap: Pretraining of language models

Autoregressive language models – loss function

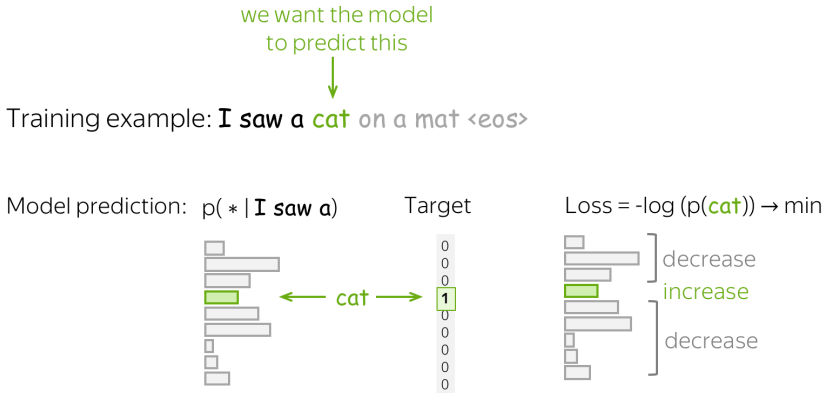


Figure: Source: `https://lena-voita.github.io/nlp_course/language_modeling.html`

Recap: Pretraining of language models

But how do we train the model to give us *good answers to user queries?**

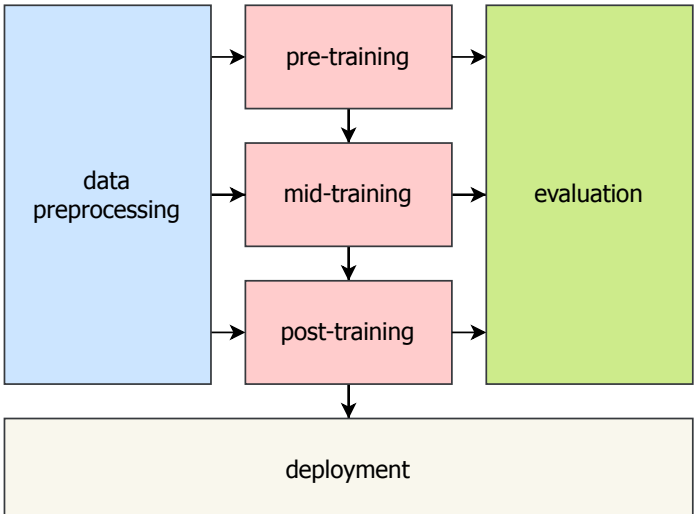
* good == helpful, honest, harmless (Askell et al., 2021),

Recap: Pretraining of language models

But how do we train the model to give us *good answers to user queries?**

* good == helpful, honest, harmless (Askell et al., 2021), *increasing the company's income*, ...

LLM lifecycle



LLM lifecycle – zoom on post-training

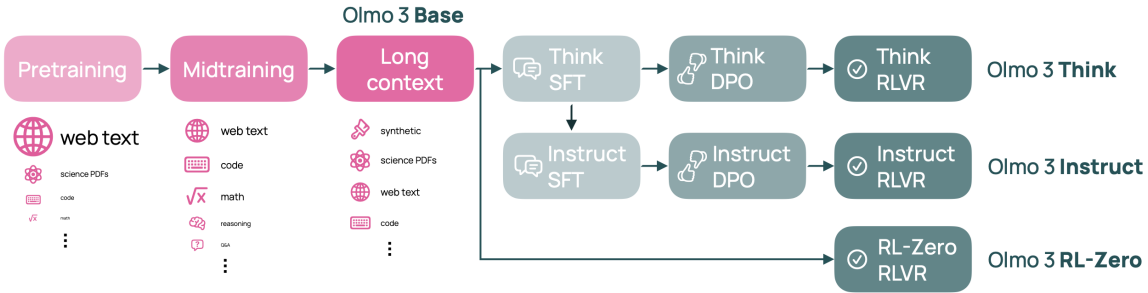


Figure 2 Depiction of model flow for Olmo 3. Development is divided into major **base model training (left)** and **post-training (right)** stages, each further divided into sub-stages with their own recipes (i.e., training data and method).

LLM lifecycle – the recent shift to post-training

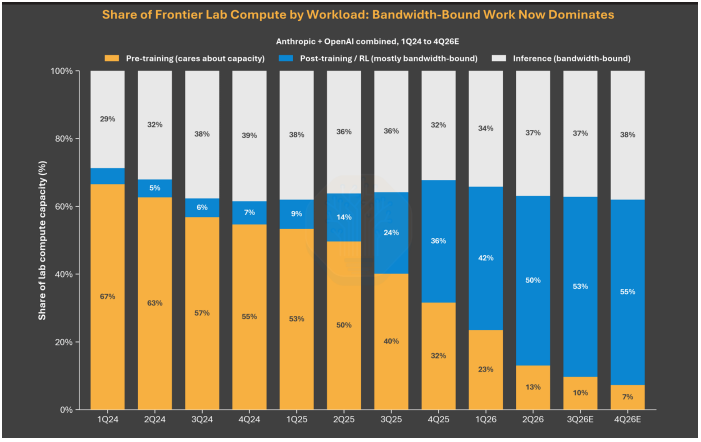


Figure: SemiAnalysis estimates and forecasts of compute allocation at OpenAI and Anthropic, source: `https://newsletter.semianalysis.com/p/long-live-the-short-king-why-4-hi`

Post-training of language models

Alignment

- LMs are pretrained on enormous amounts of texts $\rightarrow$ they can imitate all kinds of characters and domains: article by a nuclear physicist, blogpost by a conspiracy theorist, or a mean comment by an internet troll
- A big part of prompting is persuading LLMs to play the correct role $\rightarrow$ we need to align their behavior with the user intent

Post-training of language models

Alignment

- LMs are pretrained on enormous amounts of texts → they can imitate all kinds of characters and domains: article by a nuclear physicist, blogpost by a conspiracy theorist, or a mean comment by an internet troll
- A big part of prompting is persuading LLMs to play the correct role → we need to align their behavior with the user intent
- Or we can use post-training to get this behavior reliably

'Alignment' is a loaded term; in this lecture we use it in a narrow, technical sense: make the responses match what users prefer (formalized on the next slide).

Post-training of language models

From text completion to assistance

- A base model can complete any prompt $\boldsymbol{x}$ by sampling from $p_\theta(\cdot \mid \boldsymbol{x})$ – but the completion rarely aligns with what the user actually wants:

Prompt: *What is the capital of Norway?*
Base model: *What is the capital of Sweden? What is the capital of Finland? Test your knowledge with these 10 geography questions and . . .*

Post-training of language models

From text completion to assistance

- A base model can complete any prompt $\boldsymbol{x}$ by sampling from $p_\theta(\cdot \mid \boldsymbol{x})$ – but the completion rarely aligns with what the user actually wants:

Prompt: *What is the capital of Norway?*
Base model: *What is the capital of Sweden? What is the capital of Finland? Test your knowledge with these 10 geography questions and . . .*

- Post-training shifts the objective away from maximum likelihood: we want responses $\boldsymbol{y}$ that score well under a reward function r – we maximize the expected reward:

$$\underset{\theta}{\operatorname{argmax}} \underset{\boldsymbol{x}\sim\mathcal{X},\ \boldsymbol{y}\sim p_\theta(\cdot\mid\boldsymbol{x})}{\mathbb{E}} \Big[r(\boldsymbol{x},\boldsymbol{y}) \Big] \tag{2}$$

  $\mathcal{X}$: distribution of user queries; $p_\theta(\cdot \mid \boldsymbol{x})$: our model, sampling a response to $\boldsymbol{x}$; $r(\boldsymbol{x},\boldsymbol{y}) \in \mathbb{R}$: how well $\boldsymbol{y}$ answers the query $\boldsymbol{x}$

- For now, let's imagine the reward function is a perfect oracle

Post-training of language models

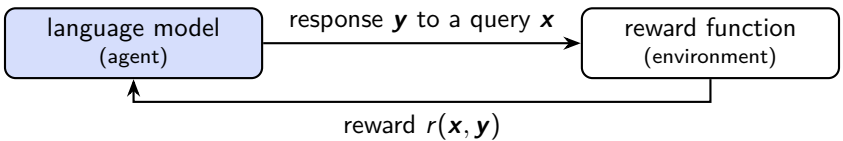
Two ways to increase the expected reward

Offline: supervised finetuning (SFT)

- somebody else writes the responses: queries paired with good answers (humans, or a stronger model)
- the model learns to imitate them – ordinary supervised learning
- the reward enters only implicitly, through which responses make it into the dataset

Online: reinforcement learning (RL)

- the model writes its own responses – nobody shows it the right answer
- all it gets back is a scalar reward $r(\boldsymbol{x}, \boldsymbol{y})$ for what it produced
- it is updated to get a higher reward next time – learning by trial and error



Contents

1 Introduction

2 Supervised finetuning
- Supervised finetuning
- Chat templates

3 Reinforcement learning

4 Where does the reward come from?

5 Direct preference optimization

6 Reasoning and tool use

7 Summary

Supervised finetuning

Offline supervised finetuning (SFT)

- Maximizing the expected reward directly is hard; a simple alternative is to create an instruction-tuning dataset $\mathcal{D}_{\text{SFT}} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$ of user queries and high-quality responses, ideally $\boldsymbol{y}_i \approx \operatorname{argmax}_{\boldsymbol{y}} r(\boldsymbol{x}_i, \boldsymbol{y})$
- Then we continue with a conditional version of the pretraining objective:

$$\operatorname*{argmax}_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y}) \sim \mathcal{D}_{\text{SFT}}} \Big[\log\, p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})\Big]. \tag{3}$$

Supervised finetuning

Offline supervised finetuning (SFT)

- Maximizing the expected reward directly is hard; a simple alternative is to create an instruction-tuning dataset $\mathcal{D}_{\text{SFT}} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}$ of user queries and high-quality responses, ideally $\boldsymbol{y}_i \approx \operatorname{argmax}_{\boldsymbol{y}} r(\boldsymbol{x}_i, \boldsymbol{y})$
- Then we continue with a conditional version of the pretraining objective:

$$\operatorname*{argmax}_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{(\boldsymbol{x},\boldsymbol{y}) \sim \mathcal{D}_{\text{SFT}}} \Big[\log\, p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})\Big]. \tag{3}$$

- In other words, we train the model to repeat the predefined responses word by word – offline training (the loss is computed on the response tokens $\boldsymbol{y}$ only; the query $\boldsymbol{x}$ is just conditioned on)
- Two kinds of training signals:
  - **Low-level**: exact sequence of words (should we care?)
  - **High-level**: helpfulness / honesty / harmfulness of responses

Supervised finetuning

What does SFT teach – and what not?

- Teaches the format and style of helpful responses by demonstration; the model picks up the preferences implicitly encoded in the curated responses
- Shifts the model from broad text completion to responding – a base model becomes a (not-yet-well-aligned) assistant

Supervised finetuning

What does SFT teach – and what not?

- Teaches the format and style of helpful responses by demonstration; the model picks up the preferences implicitly encoded in the curated responses
- Shifts the model from broad text completion to responding – a base model becomes a (not-yet-well-aligned) assistant
- **Limitations**
  - only a proxy: imitating the dataset raises the expected reward only as far as the dataset responses are actually good
  - the model cannot explore its own response strategies, and it is trained on responses it would never produce itself – distribution shift
  - finetuning on facts the model does not know can teach it to make things up (Gekhman et al., 2024)

Chat templates

What is a chat template?

- An LLM sees a flat token stream – it needs some markup: who is speaking, where the user turn ends, how to end its response
- Solution: special tokens delimiting roles and turns
- The model has to learn to finish its turn with the end-of-turn token – that is where generation stops
- The SFT loss is applied only to the assistant turns
- The new special tokens have random embeddings → SFT has to train them

```
<|im_start|>system
You are a helpful assistant.<|im_end|>
<|im_start|>user
What is the capital of Norway?<|im_end|>
<|im_start|>assistant
The capital of Norway is Oslo.<|im_end|>
<|im_start|>user
And of Sweden?<|im_end|>
<|im_start|>assistant
```

Contents

1 Introduction

2 Supervised finetuning

3 Reinforcement learning
- The RL objective
- Policy gradient methods
- The RL training loop
- From REINFORCE to PPO

4 Where does the reward come from?

5 Direct preference optimization

6 Reasoning and tool use

7 Summary

The RL objective

Online training

- Instead of imitating, we maximize the expected reward directly:

$$\operatorname*{argmax}_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{X},\, \boldsymbol{y}\sim p_{\boldsymbol{\theta}}(\cdot\,|\,\boldsymbol{x})} \Big[r(\boldsymbol{x}, \boldsymbol{y}) \Big].$$

- RL terminology: the LM is the policy, the prompt plus the tokens generated so far is the state, each generated token is an action, the whole response is a trajectory, and the reward usually arrives only at the end

The RL objective

Online training

- Instead of imitating, we maximize the expected reward directly:

$$\operatorname*{argmax}_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{X},\, \boldsymbol{y}\sim p_{\boldsymbol{\theta}}(\cdot\,|\,\boldsymbol{x})} \Big[r(\boldsymbol{x}, \boldsymbol{y}) \Big].$$

- RL terminology: the LM is the policy, the prompt plus the tokens generated so far is the state, each generated token is an action, the whole response is a trajectory, and the reward usually arrives only at the end
- Usually, we also want to regularize the policy to stay close to the initial SFT model:

$$\operatorname*{argmax}_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{X},\, \boldsymbol{y}\sim p_{\boldsymbol{\theta}}(\cdot\,|\,\boldsymbol{x})} \Big[r(\boldsymbol{x}, \boldsymbol{y}) - \beta\, D_{\mathrm{KL}}\Big(p_{\boldsymbol{\theta}}(\boldsymbol{y}\,|\,\boldsymbol{x}) \parallel p_{\mathsf{sft}}(\boldsymbol{y}\,|\,\boldsymbol{x}) \Big) \Big]$$

Side-note: Kullback–Leibler divergence

Definition (Kullback and Leibler, 1951)

$$D_{\mathrm{KL}}(P \parallel Q) := \mathop{\mathbb{E}}_{\boldsymbol{y}\sim P}\left[\log \frac{P(\boldsymbol{y})}{Q(\boldsymbol{y})}\right]$$
$$= \sum_{\boldsymbol{y}} P(\boldsymbol{y}) \log \frac{P(\boldsymbol{y})}{Q(\boldsymbol{y})}$$

- the extra surprise when samples drawn from P are expected to come from Q
- $D_{\mathrm{KL}}(P \parallel Q) \geq 0$, with equality iff $P = Q$
- not symmetric: $D_{\mathrm{KL}}(P \parallel Q) \neq D_{\mathrm{KL}}(Q \parallel P)$ in general, so it is not a distance

Policy gradient methods

How do we differentiate through sampling?

- Fix one query $\boldsymbol{x}$. We want $\nabla_{\boldsymbol{\theta}}\, \mathbb{E}_{\boldsymbol{y}\sim p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})}[r(\boldsymbol{x},\boldsymbol{y})]$, but $\boldsymbol{\theta}$ only enters through the sampling of $\boldsymbol{y}$, and r is a black box – nothing to backpropagate through

$$\nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{y}\sim p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})}\Big[r(\boldsymbol{x},\boldsymbol{y})\Big] = \nabla_{\boldsymbol{\theta}} \sum_{\boldsymbol{y}} p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})\, r(\boldsymbol{x},\boldsymbol{y}) \qquad \text{expectation = sum over all responses}$$

$$= \sum_{\boldsymbol{y}} r(\boldsymbol{x},\boldsymbol{y})\, \nabla_{\boldsymbol{\theta}}\, p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}) \qquad \text{gradient of a sum; } r \text{ is independent of } \boldsymbol{\theta}$$

$$= \sum_{\boldsymbol{y}} r(\boldsymbol{x},\boldsymbol{y})\, p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})\, \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}) \qquad \text{log-derivative trick: } \frac{d\log f}{d\theta} = \frac{1}{f}\frac{df}{d\theta} \Rightarrow \frac{df}{d\theta} = f\frac{d\log f}{d\theta}$$

$$= \mathop{\mathbb{E}}_{\boldsymbol{y}\sim p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})}\Big[\, r(\boldsymbol{x},\boldsymbol{y})\, \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}\,|\,\boldsymbol{x})\Big] \qquad \text{weighted by } p_{\boldsymbol{\theta}}\text{: an expectation again}$$

Policy gradient methods

REINFORCE

- Monte-Carlo estimate: sample a query $\boldsymbol{x}$ and a response $\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{x})$, get its reward, and step along $\widehat{\nabla}_{\boldsymbol{\theta}} = r(\boldsymbol{x}, \boldsymbol{y})\, \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})$
- This method is called REINFORCE (Williams, 1992)
- In code, this is an ordinary loss with the reward as a constant weight (no gradient flows through r):

$$\mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{i=1}^{N} r(\boldsymbol{x}_i, \boldsymbol{y}_i)\, \log p_{\boldsymbol{\theta}}(\boldsymbol{y}_i \mid \boldsymbol{x}_i), \qquad \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) = -\frac{1}{N} \sum_{i=1}^{N} r(\boldsymbol{x}_i, \boldsymbol{y}_i)\, \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}_i \mid \boldsymbol{x}_i)$$

REINFORCE with baseline

- Plain REINFORCE works in theory, but the gradient estimate has high variance $\rightarrow$ slow, unstable training
- Example: three responses a query get rewards $7, 8, 9$. REINFORCE pushes *all three* up, just by different amounts

Policy gradient methods

REINFORCE with baseline

- Plain REINFORCE works in theory, but the gradient estimate has high variance → slow, unstable training
- Example: three responses a query get rewards $7, 8, 9$. REINFORCE pushes *all three* up, just by different amounts
- Fix: subtract a baseline $b(\boldsymbol{x})$ that does not depend on $\boldsymbol{y}$:

$$\nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{X},\, \boldsymbol{y}\sim p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})} \big[r(\boldsymbol{x},\boldsymbol{y})\big] = \mathop{\mathbb{E}}_{\boldsymbol{x}\sim\mathcal{X},\, \boldsymbol{y}\sim p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})} \Big[\big(r(\boldsymbol{x},\boldsymbol{y}) - b(\boldsymbol{x})\big)\, \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}) \Big]$$

- Still unbiased: $\mathbb{E}_{\boldsymbol{y}\sim p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})}\big[b(\boldsymbol{x}) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x})\big] = b(\boldsymbol{x})\, \nabla_{\boldsymbol{\theta}} \sum_{\boldsymbol{y}} p_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}) = b(\boldsymbol{x})\, \nabla_{\boldsymbol{\theta}} 1 = 0$
- Natural choice: the expected reward of the current policy, $b(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{y}\sim p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})}[r(\boldsymbol{x},\boldsymbol{y})]$
- A response is pushed up when it is better than expected and down when worse
- $r(\boldsymbol{x},\boldsymbol{y}) - b(\boldsymbol{x})$ is called the advantage

Policy gradient methods

Where does the baseline come from?

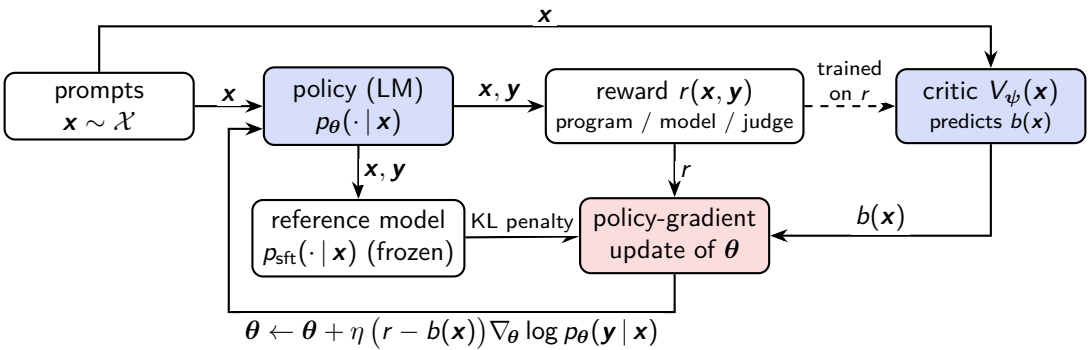
Learn it: actor–critic methods

- a second network, the critic V_ψ, is trained alongside the policy (the actor) to predict the expected reward, typically per token (Konda and Tsitsiklis, 2000)
- used by PPO (Schulman et al., 2017) – the algorithm behind InstructGPT and ChatGPT (Ouyang et al., 2022)
- cost: an extra LLM to train and run

Sample it: group baselines

- generate a group of G responses $\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(G)}$ to the same query, with rewards $r_1, \ldots, r_G$
- RLOO (Kool et al., 2019): leave-one-out mean $b_i = \frac{1}{G-1} \sum_{j \neq i} r_j$ (independent of $\boldsymbol{y}^{(i)} \rightarrow$ unbiased)
- GRPO (Shao et al., 2024): normalized advantage $A_i = \frac{r_i - \mathrm{mean}(r_{1..G})}{\mathrm{std}(r_{1..G})}$, used for DeepSeek-R1 (Guo et al., 2025)
- cost: G generations per query

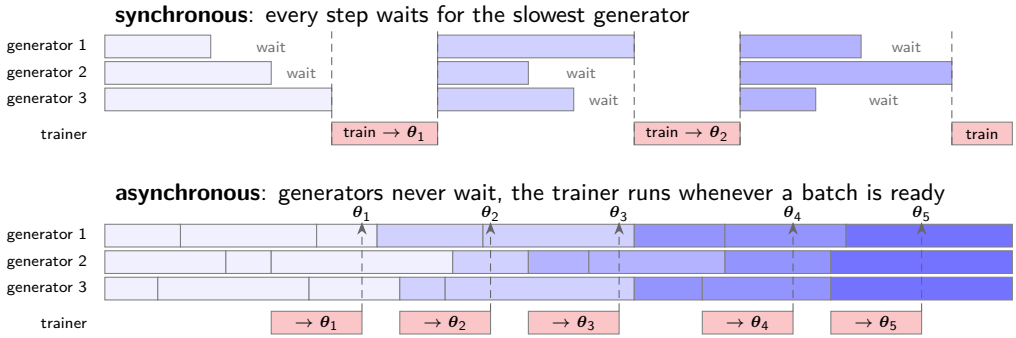
The RL training loop



- One training step: sample queries, generate responses, score them, run the reference model (KL) and the critic (baseline), update the policy & and the critic

The RL training loop

Running the loop: synchronous vs. asynchronous RL



From REINFORCE to PPO

The problem: our samples are slightly off-policy

- REINFORCE assumes that the responses were sampled from the current policy – the expectation and the gradient refer to the same $\boldsymbol{\theta}$:

$$\nabla_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})} \big[r(\boldsymbol{x}, \boldsymbol{y}) \big] = \mathop{\mathbb{E}}_{\boldsymbol{y} \sim p_{\boldsymbol{\theta}}(\cdot|\boldsymbol{x})} \Big[r(\boldsymbol{x}, \boldsymbol{y}) \, \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{y} \,|\, \boldsymbol{x}) \Big]$$

- In practice they are not. The responses come from an older version $\boldsymbol{\theta}_{\text{old}}$ of the policy because of asynchronous generation:
- Two fixes: (1) correct the expectation – importance sampling; (2) keep $\boldsymbol{\theta}$ close to $\boldsymbol{\theta}_{\text{old}}$ – clipping

From REINFORCE to PPO

Step 1: importance sampling

- An expectation under p_θ can be computed from samples of any other distribution, here $p_{\theta_{\text{old}}}$, by re-weighting:

$$\mathop{\mathbb{E}}_{\boldsymbol{y}\sim p_\theta}\left[f(\boldsymbol{y})\right] = \sum_{\boldsymbol{y}} p_\theta(\boldsymbol{y}|\boldsymbol{x})\, f(\boldsymbol{y}) = \sum_{\boldsymbol{y}} p_{\theta_{\text{old}}}(\boldsymbol{y}|\boldsymbol{x})\, \frac{p_\theta(\boldsymbol{y}|\boldsymbol{x})}{p_{\theta_{\text{old}}}(\boldsymbol{y}|\boldsymbol{x})}\, f(\boldsymbol{y}) = \mathop{\mathbb{E}}_{\boldsymbol{y}\sim p_{\theta_{\text{old}}}}\Big[\underbrace{\frac{p_\theta(\boldsymbol{y}|\boldsymbol{x})}{p_{\theta_{\text{old}}}(\boldsymbol{y}|\boldsymbol{x})}}_{\rho(\boldsymbol{x},\boldsymbol{y})}\, f(\boldsymbol{y})\Big]$$

From REINFORCE to PPO

Step 1: importance sampling

- An expectation under p_θ can be computed from samples of any other distribution, here $p_{\theta_{\text{old}}}$, by re-weighting:

$$\mathop{\mathbb{E}}_{\boldsymbol{y}\sim p_\theta}\left[f(\boldsymbol{y})\right] = \sum_{\boldsymbol{y}} p_\theta(\boldsymbol{y}|\boldsymbol{x})\, f(\boldsymbol{y}) = \sum_{\boldsymbol{y}} p_{\theta_{\text{old}}}(\boldsymbol{y}|\boldsymbol{x})\, \frac{p_\theta(\boldsymbol{y}|\boldsymbol{x})}{p_{\theta_{\text{old}}}(\boldsymbol{y}|\boldsymbol{x})}\, f(\boldsymbol{y}) = \mathop{\mathbb{E}}_{\boldsymbol{y}\sim p_{\theta_{\text{old}}}}\left[\underbrace{\frac{p_\theta(\boldsymbol{y}|\boldsymbol{x})}{p_{\theta_{\text{old}}}(\boldsymbol{y}|\boldsymbol{x})}}_{\rho(\boldsymbol{x},\boldsymbol{y})}\, f(\boldsymbol{y})\right]$$

- Applied to the policy gradient, with the importance ratio ρ ($= 1$ when $\boldsymbol{\theta} = \boldsymbol{\theta}_{\text{old}}$):

$$\nabla_\theta \mathop{\mathbb{E}}_{\boldsymbol{y}\sim p_\theta}\left[r(\boldsymbol{x},\boldsymbol{y})\right] = \mathop{\mathbb{E}}_{\boldsymbol{y}\sim p_{\theta_{\text{old}}}}\Big[\rho(\boldsymbol{x},\boldsymbol{y})\, r(\boldsymbol{x},\boldsymbol{y})\, \nabla_\theta \log p_\theta(\boldsymbol{y}\,|\,\boldsymbol{x})\Big]$$

Step 2: clipping – proximal policy optimization (PPO) (Schulman et al., 2017)

- Trust the correction only for small changes
- Once a sample has moved ρ past $1 \pm \varepsilon$ (typically $\varepsilon \approx 0.2$) in the direction its reward pushes, drop it from the update:

$$\text{gradient from sample } i = \begin{cases} 0 & \text{if } (r_i - b_i) > 0 \text{ and } \rho_i > 1 + \varepsilon \\ 0 & \text{if } (r_i - b_i) < 0 \text{ and } \rho_i < 1 - \varepsilon \\ \rho_i \, (r_i - b_i) \, \nabla_\theta \log p_\theta(\boldsymbol{y}_i \mid \boldsymbol{x}_i) & \text{otherwise} \end{cases}$$

Papers write this as a one-liner: $\min\big(\rho_i(r_i - b_i),\ \mathrm{clip}(\rho_i, 1-\varepsilon, 1+\varepsilon)\,(r_i - b_i)\big)$

Step 2: clipping – proximal policy optimization (PPO) (Schulman et al., 2017)

- Trust the correction only for small changes
- Once a sample has moved ρ past $1 \pm \varepsilon$ (typically $\varepsilon \approx 0.2$) in the direction its reward pushes, drop it from the update:

$$\text{gradient from sample } i = \begin{cases} 0 & \text{if } (r_i - b_i) > 0 \text{ and } \rho_i > 1 + \varepsilon \\ 0 & \text{if } (r_i - b_i) < 0 \text{ and } \rho_i < 1 - \varepsilon \\ \rho_i \, (r_i - b_i) \, \nabla_\theta \log p_\theta(\boldsymbol{y}_i \mid \boldsymbol{x}_i) & \text{otherwise} \end{cases}$$

Papers write this as a one-liner: $\min\big(\rho_i(r_i - b_i),\ \mathrm{clip}(\rho_i, 1-\varepsilon, 1+\varepsilon)\,(r_i - b_i)\big)$

- The policy stays proximal to the one that generated the data, so stale samples and several steps per batch are safe

Contents

1. Introduction
2. Supervised finetuning
3. Reinforcement learning
4. Where does the reward come from?
   - Three sources of reward
   - Verifiable rewards (RLVR)
   - Human feedback (RLHF)
   - AI feedback (RLAIF)
5. Direct preference optimization
6. Reasoning and tool use
7. Summary

Three sources of reward

RLVR: verifiable rewards

- the reward is computed by a program: final answer of a math problem, unit tests for code, ...
- only for tasks with checkable answers

RLHF: human feedback

- humans compare responses
- a reward model is trained on the comparisons and scores the policy's samples

RLAIF: AI feedback

- a strong LLM compares (or scores) the responses instead of humans
- cheap, scalable, available for any language the judge understands

- The training loop from the previous section is the same for all three – only the reward box changes

Verifiable rewards (RLVR)

RLVR: the reward is a program

- For some tasks the response can simply be checked: Does the final answer of a math problem match the reference? Does the code pass the unit tests? Does the response obey a constraint ("answer in exactly three bullet points"?)
- The reward is then 1 or 0 from a verifier program
- Two examples: Tülu 3 (Lambert et al., 2025) and DeepSeek-R1-Zero (Guo et al., 2025)

Human feedback (RLHF)

RLHF: learning the reward from human preferences

- The goal is a model $r_\phi(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}$ that can score any query-response pair according to human preferences
- Humans are bad at absolute scores but decent at comparisons $\rightarrow$ create a preference dataset $\mathcal{D}_{\text{PO}}$ of triples $(\boldsymbol{x}, \boldsymbol{y}_w, \boldsymbol{y}_\ell)$ with a preferred (*chosen*) and a *rejected* responses

Human feedback (RLHF)

RLHF: learning the reward from human preferences

- The goal is a model $r_\phi(\boldsymbol{x}, \boldsymbol{y}) \in \mathbb{R}$ that can score any query-response pair according to human preferences
- Humans are bad at absolute scores but decent at comparisons $\rightarrow$ create a preference dataset $\mathcal{D}_{\text{PO}}$ of triples $(\boldsymbol{x}, \boldsymbol{y}_w, \boldsymbol{y}_\ell)$ with a preferred (*chosen*) and a *rejected* responses
- Bradley–Terry model (Bradley and Terry, 1952): $P(\boldsymbol{y}_w \succ \boldsymbol{y}_\ell \,|\, \boldsymbol{x}) = \sigma\big(r_\phi(\boldsymbol{x}, \boldsymbol{y}_w) - r_\phi(\boldsymbol{x}, \boldsymbol{y}_\ell)\big)$, fitted by maximum likelihood:

$$\mathcal{L}_{\text{RM}}(\phi) = - \mathop{\mathbb{E}}_{(\boldsymbol{x}, \boldsymbol{y}_w, \boldsymbol{y}_\ell) \sim \mathcal{D}_{\text{PO}}} \Big[\log \sigma\big(r_\phi(\boldsymbol{x}, \boldsymbol{y}_w) - r_\phi(\boldsymbol{x}, \boldsymbol{y}_\ell) \big) \Big], \qquad \sigma(z) = \frac{1}{1 + e^{-z}}$$

- Architecture: LLM with a randomly initialized linear head that outputs a scalar; only score differences matter

AI feedback (RLAIF)

- The preference dataset $\mathcal{D}_{\text{PO}}$ is another data bottleneck: expensive and non-existent for most languages
- Observation: sufficiently capable LLMs give preference judgments that agree with humans about as often as humans agree with each other (Zheng et al., 2023)
- We can thus use existing LLMs to provide rewards – RLAIF (Bai et al., 2022)

AI feedback (RLAIF)

- The preference dataset $\mathcal{D}_{\text{PO}}$ is another data bottleneck: expensive and non-existent for most languages
- Observation: sufficiently capable LLMs give preference judgments that agree with humans about as often as humans agree with each other (Zheng et al., 2023)
- We can thus use existing LLMs to provide rewards – RLAIF (Bai et al., 2022)
- Two flavours of RLAIF:
  - **distilled**: the LLM labels the pairs $(\boldsymbol{y}_w, \boldsymbol{y}_\ell)$, then a reward model is trained as before
  - **direct (d-RLAIF)** (Lee et al., 2024): the judge is prompted for a score inside the training loop

Contents

1. Introduction
2. Supervised finetuning
3. Reinforcement learning
4. Where does the reward come from?
5. Direct preference optimization
   - Direct preference optimization
6. Reasoning and tool use
7. Summary

Direct preference optimization

Skipping the reward model and the RL loop (Rafailov et al., 2023)

- With some clever math, we can show that any reward can be written in terms of its optimal policy (for a KL-regularized RL objective):

$$r(\boldsymbol{x}, \boldsymbol{y}) = \beta \log \frac{p^*(\boldsymbol{y}|\boldsymbol{x})}{p_{\text{sft}}(\boldsymbol{y}|\boldsymbol{x})} + \beta \log Z(\boldsymbol{x})$$

Direct preference optimization

Skipping the reward model and the RL loop (Rafailov et al., 2023)

- With some clever math, we can show that any reward can be written in terms of its optimal policy (for a KL-regularized RL objective):

$$r(\boldsymbol{x}, \boldsymbol{y}) = \beta \log \frac{p^*(\boldsymbol{y}|\boldsymbol{x})}{p_{\text{sft}}(\boldsymbol{y}|\boldsymbol{x})} + \beta \log Z(\boldsymbol{x})$$

- Plug this into the Bradley–Terry loss and we can fit the policy itself directly on the preference data:

$$\mathcal{L}_{\text{DPO}}(\boldsymbol{\theta}) = - \mathop{\mathbb{E}}_{(\boldsymbol{x}, \boldsymbol{y}_w, \boldsymbol{y}_\ell) \sim \mathcal{D}_{\text{PO}}} \left[\log \sigma \left(\beta \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{y}_w|\boldsymbol{x})}{p_{\text{sft}}(\boldsymbol{y}_w|\boldsymbol{x})} - \beta \log \frac{p_{\boldsymbol{\theta}}(\boldsymbol{y}_\ell|\boldsymbol{x})}{p_{\text{sft}}(\boldsymbol{y}_\ell|\boldsymbol{x})} \right) \right]$$

- "Your language model is secretly a reward model": the log-ratios act as an implicit reward; β controls how far $p_{\boldsymbol{\theta}}$ may move from p_{sft}
- Just a supervised loss on $(\boldsymbol{x}, \boldsymbol{y}_w, \boldsymbol{y}_\ell)$: no reward model, no sampling during training

Direct preference optimization

Trade-offs

Pros

- simple, stable, cheap: no reward model, no generation loop
- widely used as a stage: Llama 3 (SFT + rejection sampling + DPO), Tülu 3 and Olmo 2/3 (SFT $\rightarrow$ DPO $\rightarrow$ RLVR) (Grattafiori et al., 2024; Lambert et al., 2025; Team Olmo et al., 2025)

Cons

- offline: fits fixed pairs, often written by another model; cannot discover new response strategies
- tends to generalize worse than on-policy RL (Xu et al., 2024)
- known failure mode: the probability of *both* $\boldsymbol{y}_w$ and $\boldsymbol{y}_\ell$ can decrease, as long as the margin grows

Trade-offs

Pros

- simple, stable, cheap: no reward model, no generation loop
- widely used as a stage: Llama 3 (SFT + rejection sampling + DPO), Tülu 3 and Olmo 2/3 (SFT → DPO → RLVR) (Grattafiori et al., 2024; Lambert et al., 2025; Team Olmo et al., 2025)

Cons

- offline: fits fixed pairs, often written by another model; cannot discover new response strategies
- tends to generalize worse than on-policy RL (Xu et al., 2024)
- known failure mode: the probability of *both* $\boldsymbol{y}_w$ and $\boldsymbol{y}_\ell$ can decrease, as long as the margin grows

- Much of the gap closes with on-policy preference data: sample pairs from the current model and let a reward model or a judge pick the winner (Ivison et al., 2024) – but then, why not just do full RL?

Contents

1. Introduction
2. Supervised finetuning
3. Reinforcement learning
4. Where does the reward come from?
5. Direct preference optimization
6. Reasoning and tool use
   - Reasoning and test-time scaling
   - Tool calling
7. Summary

Thinking before answering

- A transformer spends the same compute on every token – one forward pass per generated token. For a hard problem, the few tokens of the final answer give the model no room to compute
- Chain of thought / reasoning: let the model write out intermediate steps before the answer – the generated text works as a scratchpad (Wei et al., 2022)

Reasoning and test-time scaling

Thinking before answering

- A transformer spends the same compute on every token – one forward pass per generated token. For a hard problem, the few tokens of the final answer give the model no room to compute
- Chain of thought / reasoning: let the model write out intermediate steps before the answer – the generated text works as a scratchpad (Wei et al., 2022)
- In the chat template, the thinking lives in a `<think>...</think>` block that is separated from the response itself
- How does a model learn to think? Rewarded only according to the final response (not the intermediate thinking), the models discover reasoning strategies on their own (Guo et al., 2025)

Reasoning and test-time scaling

Test-time scaling

- Accuracy keeps improving as the model spends more tokens thinking – a new axis of scaling next to parameters and data (OpenAI, 2024)
  - **sequential**: longer reasoning
  - **parallel**: sample several answers and take a majority vote (Wang et al., 2023), or let a reward model pick the best one

How long should the model think?

- A reward on the final answer alone says nothing about cost: we don't want the model to spend an excessive number of tokens on simple problems
- In practice, the reward gets a second, length-based term: a penalty that grows with the number of thinking tokens

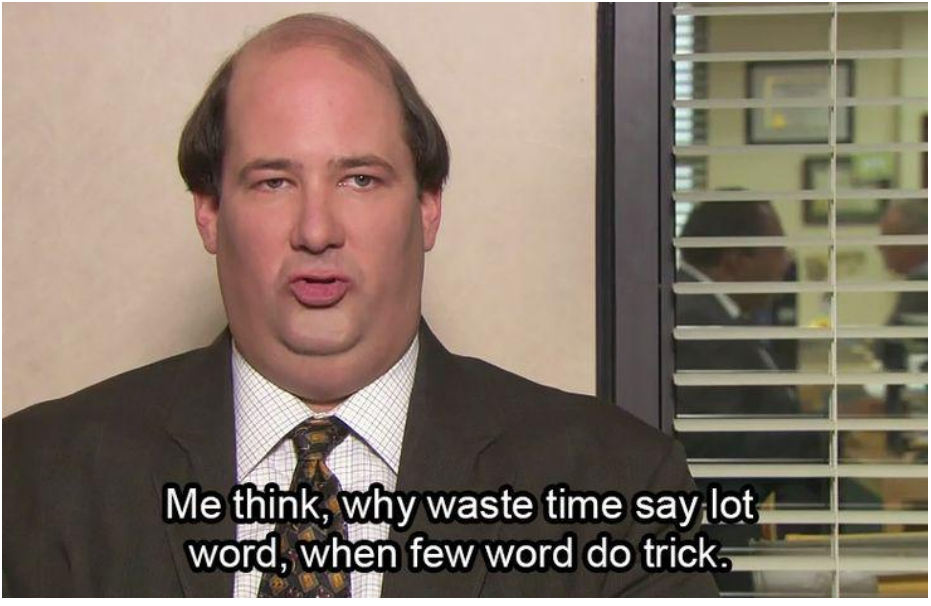
How long should the model think?

- A reward on the final answer alone says nothing about cost: we don't want the model to spend an excessive number of tokens on simple problems
- In practice, the reward gets a second, length-based term: a penalty that grows with the number of thinking tokens
- Users can ask for a reasoning budget ("effort" low/medium/high/max)
  - **train for it**: put the target length in the prompt and reward not spending more tokens (Aggarwal and Welleck, 2025)
  - **enforce it at inference**: when the budget is used up, close the thinking block (insert `</think>`) and let the model answer from what it has
- Hybrid reasoning: one model that answers with or without thinking, switched per request (Yang et al., 2025). The non-thinking mode is just an enforced empty thinking block

What does the thinking actually look like?

A raw reasoning trace of ChatGPT in 2026

Need add tests for invalid position. Where? tests/lsp-tool.test.mjs. Add test that line 0 rejected before server? Maybe check codeAction? Should catch error and return error result. Could also add service-level tests for direct service.hover with line 0? But tool enough? We need ensure no LSP starts? Could check status active clients 0 if invalid? Since lspService.hover validates before file read / server start. If path exists. Test invalid line 0 returns isError and text "requires 1-based line and column" maybe no root absolute. Need start. Also test decimal column, legacy character invalid, codeAction missing/zero. Maybe not too many.


Me think, why waste time say lot
word, when few word do trick.

Tool calling

Letting the model act

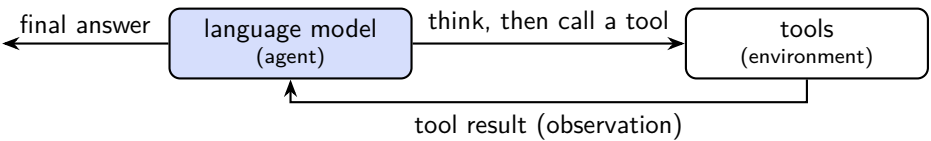
- So far the model only produces text. A tool call is a response in a fixed, machine-readable format that gets executed externally: web search, calculator, code execution, databases, any API
- The result comes back as a new turn, and the model continues
- The model has to be trained to (a) decide when a tool is needed, (b) produce exactly the right format and arguments, (c) read the result, and (d) know when it is done

```
<|im_start|>system
You may call these tools:
{"name": "get_weather",
 "parameters": {"city": "string"}}<|im_end|>
<|im_start|>user
Do I need an umbrella in Oslo today?<|im_end|>
<|im_start|>assistant
<tool_call>
{"name": "get_weather",
 "arguments": {"city": "Oslo"}}
</tool_call><|im_end|>
<|im_start|>tool
{"temp_c": 7, "condition": "rain"}<|im_end|>
<|im_start|>assistant
Yes -- it is 7 degrees and raining
in Oslo right now.<|im_end|>
```

Simplified ChatML-style example; the exact tags differ between model families.

Tool calling

Agents: tool calls in a loop



- Think, call a tool, read the result, repeat until done: an agent (Yao et al., 2023)
- Training-wise: same as before (reward calculated on the final response), the main bottleneck is providing diverse environments, prompts are not enough as an input

Contents

1 Introduction

2 Supervised finetuning

3 Reinforcement learning

4 Where does the reward come from?

5 Direct preference optimization

6 Reasoning and tool use

7 Summary
- Summary

Summary

	SFT	**DPO**	**RLHF / RLAIF**	**RLVR**
inputs	$(\boldsymbol{x}, \boldsymbol{y})$ demonstrations	$(\boldsymbol{x}, \boldsymbol{y}_w, \boldsymbol{y}_\ell)$ pairs	$\boldsymbol{x}$ & a reward model	$\boldsymbol{x}$ & a verifier
outputs from	dataset	dataset	the model	the model
reward	implicit	implicit, pairwise	learned or judged scalar	programmatic
cost	low	low	high (generation + reward model)	high (generation)
typical role	learning format, chat template	cheap preference optimization	helpfulness, style, safety	reasoning, math, code

- All of them aim at the same thing, a high expected reward of the model's own responses – SFT and DPO indirectly, RL directly
- Further reading: the RLHF book (Lambert, 2025)

References I

Pranjal Aggarwal and Sean Welleck. **L1: Controlling how long a reasoning model thinks with reinforcement learning**. In *Proceedings of the Second Conference on Language Modeling (COLM)*, 2025.

Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Ben Mann, Nova DasSarma, et al. **A general language assistant as a laboratory for alignment**, 2021.

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**, 2022.

Ralph Allan Bradley and Milton E. Terry. **Rank analysis of incomplete block designs: I. the method of paired comparisons**. *Biometrika*, 39(3/4):324–345, 1952.

Zorik Gekhman, Gal Yona, Roee Aharoni, Matan Eyal, Amir Feder, Roi Reichart, and Jonathan Herzig. **Does fine-tuning LLMs on new knowledge encourage hallucinations?** In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7765–7784, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

References II

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. **The Llama 3 herd of models**, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. **DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning**. *Nature*, 645(8081):633–638, 2025.

Hamish Ivison, Yizhong Wang, Jiacheng Liu, Zeqiu Wu, Valentina Pyatkin, Nathan Lambert, Noah A. Smith, Yejin Choi, and Hannaneh Hajishirzi. **Unpacking DPO and PPO: Disentangling best practices for learning from preference feedback**. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Vijay R. Konda and John N. Tsitsiklis. **Actor-critic algorithms**. In *Advances in Neural Information Processing Systems*, volume 12, pages 1008–1014. MIT Press, 2000.

Wouter Kool, Herke van Hoof, and Max Welling. **Buy 4 REINFORCE samples, get a baseline for free!** In *ICLR 2019 Workshop on Deep Reinforcement Learning Meets Structured Prediction*, 2019.

Solomon Kullback and Richard A. Leibler. **On information and sufficiency**. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

Nathan Lambert. **Reinforcement learning from human feedback**, 2025. Online book: rlhfbook.com.

References III

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. **Tülu 3: Pushing frontiers in open language model post-training**. In *Proceedings of the Second Conference on Language Modeling (COLM)*, 2025.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, and Sushant Prakash. **RLAIF vs. RLHF: Scaling reinforcement learning from human feedback with AI feedback**. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 26874–26901, 2024.

OpenAI. **Learning to reason with LLMs**. Blog post, 2024.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Gray, 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**. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

References IV

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. **Direct preference optimization: Your language model is secretly a reward model**. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. **Proximal policy optimization algorithms**, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. **DeepSeekMath: Pushing the limits of mathematical reasoning in open language models**, 2024.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li,

References V

Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. **Olmo 3**, 2025.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. **Self-consistency improves chain of thought reasoning in language models**. In *International Conference on Learning Representations*, 2023.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. **Chain-of-thought prompting elicits reasoning in large language models**. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Ronald J. Williams. **Simple statistical gradient-following algorithms for connectionist reinforcement learning**. *Machine Learning*, 8(3–4):229–256, May 1992. ISSN 0885-6125.

Shusheng Xu, Wei Fu, Jiaxuan Gao, Wenjie Ye, Weilin Liu, Zhiyu Mei, Guangju Wang, Chao Yu, and Yi Wu. **Is DPO superior to PPO for LLM alignment? a comprehensive study**. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 54983–54998, 2024.

References VI

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, et al. **Qwen3 technical report**, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. **ReAct: Synergizing reasoning and acting in language models**. In *International Conference on Learning Representations*, 2023.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. **Judging LLM-as-a-judge with MT-Bench and Chatbot Arena**. In *Advances in Neural Information Processing Systems 36, Datasets and Benchmarks Track*, 2023.