How Are Large Language Models Trained?

From next-token prediction to reasoning

Peng Wang

University of Virginia

Guest lecture for UVA CS 4774: Machine Learning (Instructor: Prof. Yanjun Qi)

October 15, 2026

Today: a quick look at how ChatGPT is trained

1. **Generative AI & LLMs** — what is an LLM?
2. **Language modeling** — predicting the next token
3. **Prompting & in-context learning** — learning by example
4. **Supervised fine-tuning** — following instructions
5. **RL basics** — learning from a score instead of a label
6. **Alignment** — learning from human preferences
7. **Verifiable rewards & GRPO** — learning to reason
8. **Wrap-up**

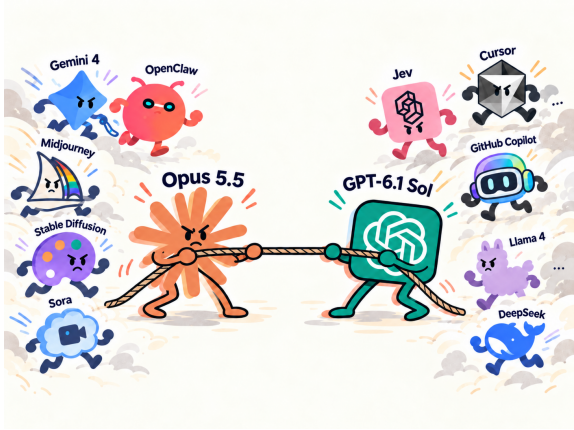
You already know (we will reuse it):
- logistic regression & softmax
- MLE & cross-entropy
- gradient descent / SGD
- regularization, neural nets

Coming later in CS 4774:
- attention & Transformers
- reinforcement learning in depth
- RAG, fine-tuning, agents

The big picture: every stage trains the *same* network by gradient descent on $\log p_\theta(\text{text})$. What changes from stage to stage is *which text* we use and *what weight* it gets.

Generative AI is everywhere (2026)

- **Text & chat:** ChatGPT, Claude, Gemini, Llama, Qwen, DeepSeek
- **Code:** GitHub Copilot, Cursor, Claude Code
- **Images / video:** Stable Diffusion, Midjourney, Sora, Veo
- **Multimodal:** one model reads images, audio, and text
- **Agents:** LLMs that browse, call tools, write and run code



A new frontier model every few months.

(illustration generated with ChatGPT)

Generative AI is everywhere (2026)

- **Text & chat:** ChatGPT, Claude, Gemini, Llama, Qwen, DeepSeek
- **Code:** GitHub Copilot, Cursor, Claude Code
- **Images / video:** Stable Diffusion, Midjourney, Sora, Veo
- **Multimodal:** one model reads images, audio, and text
- **Agents:** LLMs that browse, call tools, write and run code

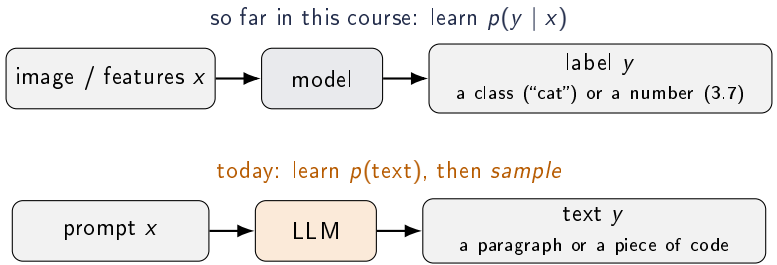
Behind all of it:

the **Transformer** (Vaswani et al., 2017)

+ **self-supervised pretraining**
  (on trillions of words of web text)
+ **scale**
+ **mid- and post-training**
  (fine-tuning, alignment, RL)

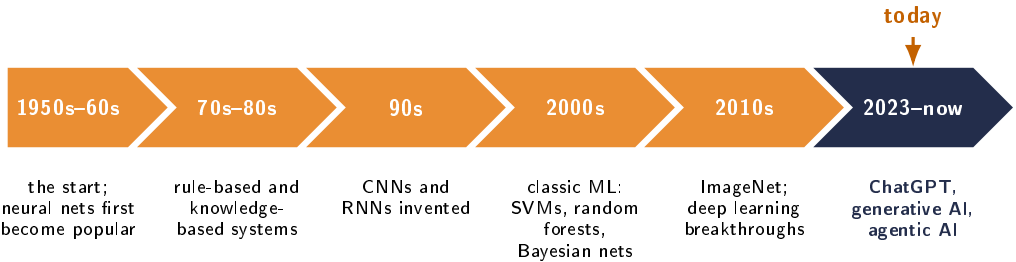
We will focus on the last one today.

Discriminative vs. generative: an LLM is both



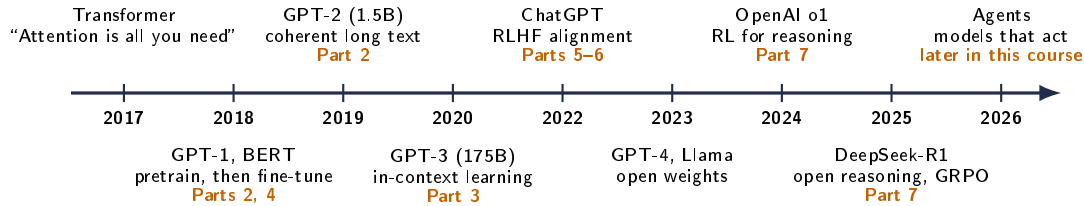
- A classifier picks one of K labels.
- A generative model puts probability on *whole pieces of data* and can produce new ones.
- An LLM is *both*: a classifier over the next word, run in a loop to generate text.

Zooming out: the machine learning timeline



The core has not changed. Today's models are still *neural networks trained by gradient descent on a loss*, exactly what you learned in the first half of this course. What changed is the *architecture*, the *data*, the *scale*, and the *training recipe*.

Zooming in: 2017 to 2026



- 2018–20: **bigger** models of the same kind, trained to predict the next word.
- 2022: the same network, **post-trained** to follow instructions and human preferences.
- 2024–25: trained with **reinforcement learning** to "think" before answering: $\sim$15% $\rightarrow$ $\sim$80% on olympiad math.
- 2026: the same models as **agents**: they call tools, browse, and write code over many steps.

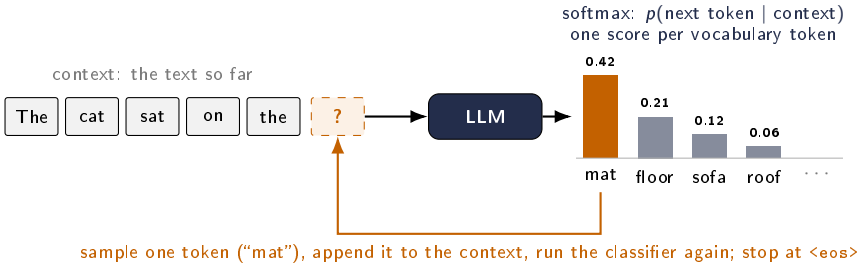
What is a large language model?

- a **Transformer decoder** that **generates text**
- **pretrained** on trillions of words from the web
- **billions** of parameters (GPT-3: 175B, Llama-3: 405B)
- a **general-purpose** solver: translation, coding, Q&A, math

At its core, an LLM is a classifier.

Input: the text so far.

Output: a distribution over the *next token*, one of $|V| \approx 100{,}000$ classes.



Base model vs. chat model: same network, different training

Prompt: Explain the moon landing to a 6 year old in a few sentences.

Base model (GPT-3, 2020)
after pre-training only: predicts the next word

Explain the theory of gravity to a 6 year old.
Explain the theory of relativity to a 6 year old in a few sentences.
Explain the big bang theory to a 6 year old.
Explain evolution to a 6 year old.

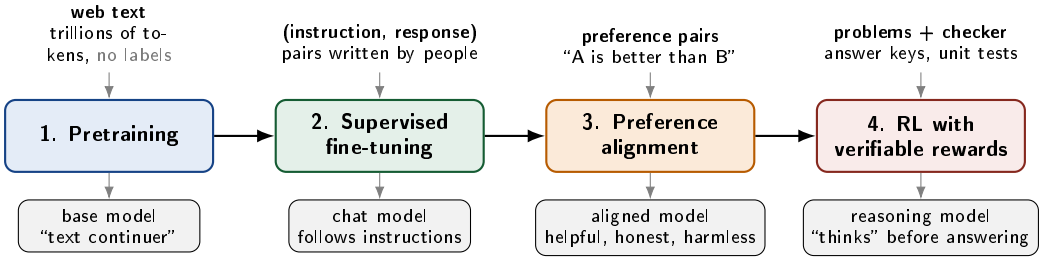
Chat model (InstructGPT, 2022)
the same network after post-training

People went to the moon, and they took pictures of what they saw, and sent them back to the earth so we could all see them.

Same network, two training stages. Pre-training on web text gives the base model, which continues the prompt the way web text would. **Post-training** on examples of helpful answers turns the *same network* into the chat model.

Example from OpenAI's InstructGPT announcement (2022); paper: Ouyang et al., 2022.

How a base model becomes a chat model: the training pipeline



- Not in the chain: **prompting / in-context learning**, which changes the *input*, not the weights.
- For each stage we ask the course's four questions: **Data? Loss? Optimization? What is learned?**

Part 2

Language modeling

predicting the next token

Generative AI & LLMs

▶ **Language modeling**

Prompting & in-context learning

Supervised fine-tuning

RL basics

Alignment

Verifiable rewards & GRPO

Wrap-up

A language model assigns probabilities to text

- A **language model (LM)** assigns a probability to any sequence of tokens $x_1, \ldots, x_T$.
- Chain rule of probability:

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_{<t})$$

- Example: $P(\text{I eat the apple}) = P(\text{I}) \cdot P(\text{eat} \mid \text{I}) \cdot P(\text{the} \mid \text{I eat}) \cdot P(\text{apple} \mid \text{I eat the})$

So an LM only ever has to answer **one** question, over and over:
given the text so far, what comes next? $P(x_t \mid x_{<t})$

Two ways to estimate $P(x_t \mid x_{<t})$

Old way: count (n-gram models)

$$P(\text{apple} \mid \text{I eat the}) = \frac{P(\text{I eat the apple})}{P(\text{I eat the})} \quad \text{conditional probability}$$

$$\approx \frac{\#(\text{I eat the apple})}{\#(\text{I eat the})} \quad \text{count in a corpus}$$

- each probability $\approx$ (count)/(corpus size); the corpus size cancels
- most 4-word sequences never occur, even in a huge corpus
- fixed window, no generalization

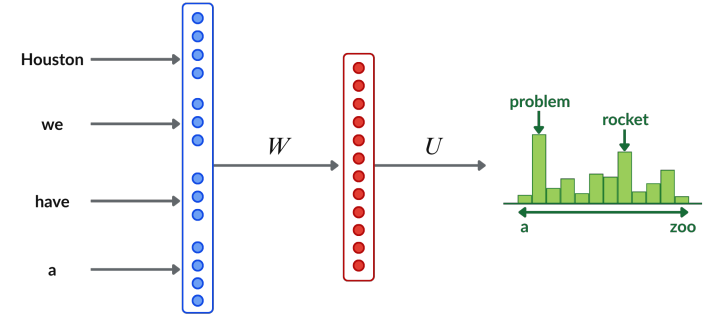
New way: a neural network

$$P(x_t \mid x_{<t}) \approx \mathrm{NN}_\theta(x_{<t})$$

- learns continuous representations; generalizes to unseen phrasings
- can condition on thousands of previous tokens

You've seen this (this course): A network with parameters θ, trained by gradient descent on a loss: the same as every model since Lecture 2.

The simplest neural language model



- Look up an **embedding** for each of the last few tokens and concatenate them; one **hidden layer**; a **softmax** over the whole vocabulary.
- This is the feed-forward network you already know, with a $|V|$-way output. Modern LLMs swap the middle for a Transformer and keep both ends.

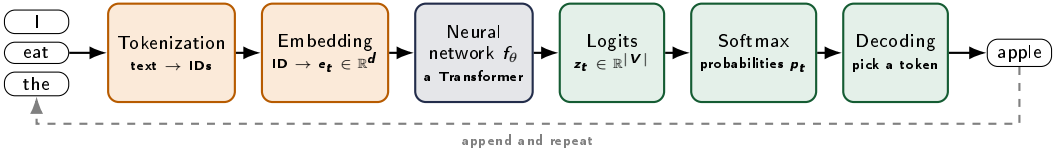
Step 0: tokenization, text becomes a sequence of integers

- A **token** is a common chunk of characters; the **vocabulary** V lists all tokens ($|V| \approx$ 50k–128k).
- **Tokenization** maps text $\rightarrow$ token IDs in $\{1, \ldots, |V|\}$.
- **Subword** tokens: frequent words are one token, rare words are split:
  `unhappiness` $\rightarrow$ `un` + `happi` + `ness`
- Foreshadowing: the model never sees letters, only tokens. That is one reason "how many r's in *strawberry*" is hard (we will see it in the prompting part).



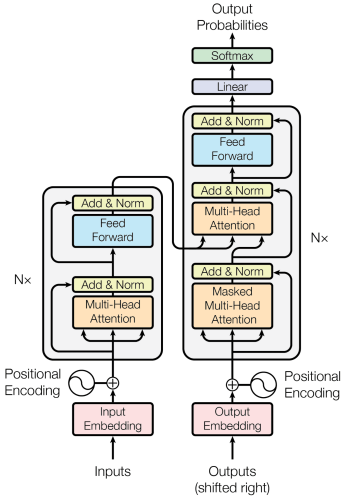
152 characters $\rightarrow$ 33 tokens.
https://platform.openai.com/tokenizer

The whole model, end to end



- **Embedding:** a learned lookup table $E \in \mathbb{R}^{|V| \times d}$; token ID i_t becomes the vector $e_t = E_{i_t}$.
- **Network:** a deep neural network (a Transformer) turns $e_1, \ldots, e_t$ into one vector $h_t \in \mathbb{R}^d$ that summarizes "the text so far".
- **Head:** a linear layer + softmax turns h_t into a distribution over the next token.

The network is a Transformer

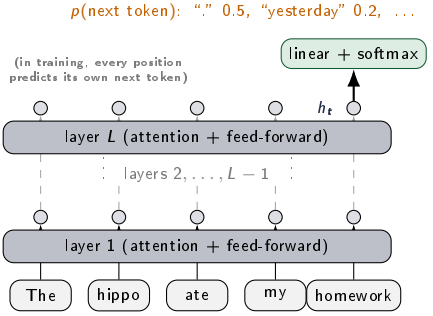


- The architecture from *Attention Is All You Need* (Vaswani et al., 2017). GPT-style LLMs keep only the right half, the **decoder**: N identical layers of attention + feed-forward, stacked.
- Look at the two ends: **embeddings** go in at the bottom; **Linear + Softmax** over the vocabulary comes out at the top. Those are the parts we care about today.

For today, the middle is a black box f_θ: a big differentiable function from the tokens so far to a vector h_t, trained by gradient descent. The course opens the box later (attention, Transformers).

Inside the network (just enough for today)

- A deep stack of layers: $h^{(\ell)} = f_\theta^{(\ell)}\big(h^{(\ell-1)}\big)$, $L = 32$ for Llama-3-8B.
- Each layer: attention (mix information *across positions*) + feed-forward (transform each position).
- Position t only sees positions $\leq t$: no cheating when predicting x_{t+1}.
- **Training:** all positions in parallel, in one forward pass; that is what makes trillions of tokens feasible.
- **Generation:** one token at a time, each conditioned on the ones before. This is called *autoregressive* generation.

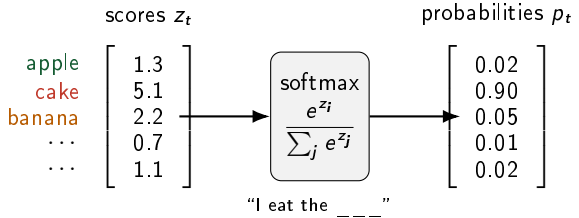


You've seen this (neural networks & deep learning): A Transformer is "just" another architecture: layers, nonlinearities, residuals, a loss at the top, gradients flowing down. Details later in this course.

The output layer is multi-class logistic regression

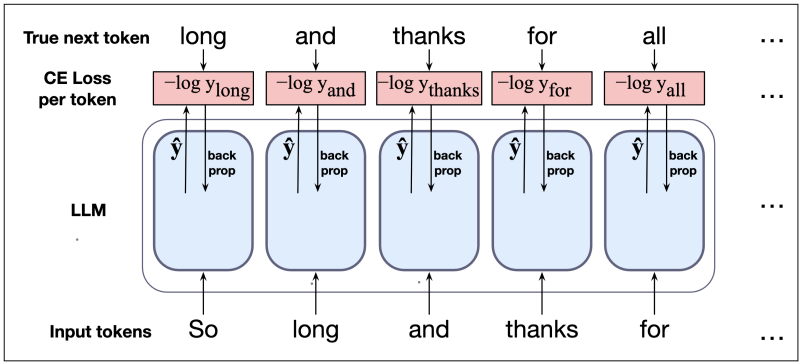
At position t the network gives $h_t \in \mathbb{R}^d$. Then

$$\underbrace{z_t = W_{\text{out}}\, h_t + b}_{\text{logits: one score per token}} \in \mathbb{R}^{|V|}, \qquad p_t(i) = \frac{\exp(z_{t,i})}{\sum_{j=1}^{|V|} \exp(z_{t,j})}$$



You've seen this (logistic regression): $p(y{=}1 \mid x) = \sigma(w^\top x + b)$ is the 2-class case; softmax is its K-class generalization. Here the features are h_t and $K = |V| \approx 100{,}000$. Nothing else is new.

Pretraining in one picture



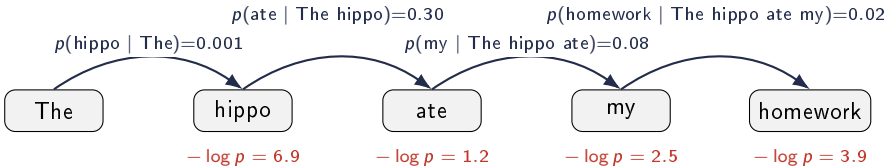
- Every position is a classification problem: the model outputs $\hat{y}$, a distribution over all possible next tokens.
- Loss at that position: the **negative log-likelihood (NLL)** of the *true* next token, $-\log \hat{y}_{\text{true}}$. Average over the batch, backpropagate, update every weight (embeddings included).

Figure: Jurafsky & Martin, *Speech and Language Processing*, 3rd ed.

Training = cross-entropy = maximum likelihood on the corpus

Pretraining loss (GPT-style, "causal" language modeling):

$$\mathcal{L}(\theta) = -\frac{1}{N}\sum_{t=1}^{N} \log p_\theta\left(x_t \mid x_{<t}\right)$$



- Each probability is conditioned on the *whole prefix* $x_{<t}$, not just the previous word. Target at each position = the **actual next token**; loss = its NLL; average over positions.
- **Self-supervised**: the labels are free, the next word is already in the text. Minimized by **SGD (Adam)** + backprop.

You've seen this (MLE): $\max \prod_t p_\theta(x_t \mid x_{<t}) \Leftrightarrow \min -\sum_t \log p_\theta(x_t \mid x_{<t})$. Pretraining *is* maximum likelihood.

Why does predicting the next word teach so much?

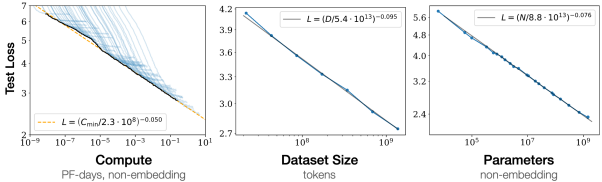
Minimizing one loss on trillions of words forces the model to pick up every regularity that helps predict the next token:

to fill the blank it must learn ...	**example**
grammar	In my free time, I like to {run, banana}
lexical semantics	At the zoo we saw giraffes, lions, and {zebras, spoon}
world knowledge	The capital of Denmark is {Copenhagen, London}
sentiment	Edge of my seat the whole time. The movie was {good, bad}
translation	The word for "pretty" in Spanish is {bonita, hola}
arithmetic	$3 + 8 + 4 = \{15, 11\}$

Pretraining is implicit multi-task learning. Nobody wrote a sentiment loss or a translation loss; the next-token loss already contains all of them.

Scale: more of everything, predictably better

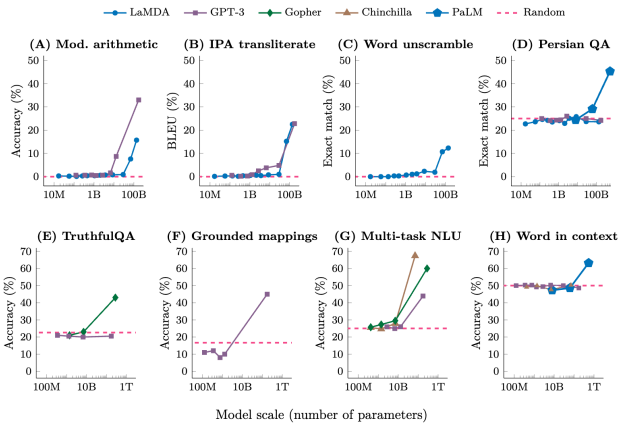
- **Data:** Llama-3 saw ∼15 trillion tokens.
- **Parameters:** 0.1B (2018) → 175B (2020) → 405B (2024).
- **Compute:** thousands of GPUs for months.
- **Scaling laws** (Kaplan et al., 2020): test loss falls as a **power law** in each factor: straight lines on log-log axes.



Test loss vs. training compute (in PF-days: 10^{15} floating-point ops per second, sustained for a day), dataset size (tokens), and parameters.

You've seen this (bias–variance): More capacity should eventually mean more variance (overfitting). Here test loss keeps falling with more parameters because **the data grows along with the model** and is seen only about once, so there is little to memorize. Scale data and parameters *together*.

Emergent abilities: some skills appear only at scale



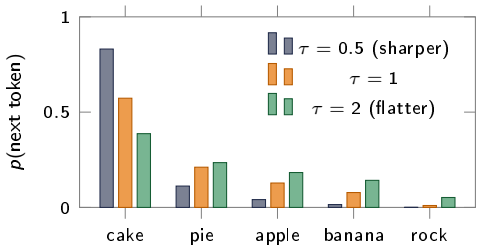
Few-shot accuracy vs. model size for five model families (Wei et al., 2022).

- Few-shot accuracy on arithmetic, word unscrambling, multi-task understanding, ...
- Stays at **chance** until a size threshold, then jumps; not predictable from smaller models.
- The same recipe at larger scale gave GPT-3 the ability we look at next: learning from examples in the prompt.

Decoding: from a probability distribution to text

Loop: compute p_t, pick x_t, append, repeat.

- **Greedy:** $x_t = \arg\max_i p_t(i)$. Deterministic; bland, repetitive.
- **Sampling:** $x_t \sim p_t$. Diverse; may pick nonsense from the tail.
- **Temperature τ:** divide the logits by τ before the softmax, $p_t^{(\tau)}(i) \propto \exp(z_{t,i}/\tau)$. $\tau \to 0$: greedy; $\tau = 1$: the model's own p_t; $\tau \to \infty$: uniform.
- **Top-k / top-p:** cut off the tail, then sample from what is left.



The same logits for "I eat the ___" at three temperatures.

This is why ChatGPT answers differently each time ($\tau > 0$), and why we can ask for *several different answers* to one question: the engine of reasoning training later today.

Part 3

Prompting & in-context learning

learning by example

Generative AI & LLMs
Language modeling
▶ **Prompting & in-context learning**
Supervised fine-tuning
RL basics
Alignment
Verifiable rewards & GRPO
Wrap-up

A pretrained LM continues text; prompting designs the input

- A **prompt** is the text we hand the model; its only job is to continue it.
- **Prompting** = write the prompt so that the *most likely continuation* is the answer we want.
- **Zero-shot:** a task description, no examples.
- No training, no gradient, no change to θ.

Sentiment analysis by continuation

Review: *The food was cold and the service painfully slow.*
Sentiment:

negative

Question answering by continuation

Q: Who wrote *Hamlet*?
A:

William Shakespeare

In-context learning: learning from examples with frozen weights

Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.

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

Few-shot prompting in the GPT-3 paper (Brown et al., 2020).

- **Few-shot / in-context learning (ICL):** put k input→output examples in the prompt; the model produces the output for a new input.
- Appears at scale (GPT-3): bigger models get much better at it.
- **No parameter update.** The "learning" happens inside one forward pass.

You've seen this (everything so far): Every algorithm in this course learns by changing weights with gradient steps. ICL adapts *without touching the weights*. How it does so is still debated: one line of work argues that attention implicitly performs a form of gradient descent on the examples inside the forward pass (von Oswald et al., 2023; Dai et al., 2023).

In-context learning, live: two examples change the answer

Demo 1: letter manipulation

Swap the second and the penultimate letter of the following word: `pothyn`

`pohtyn` ✗

Directly answer the last one (swapping the second and the penultimate letter):
`tarehd` → `thread`, `revir` → `river`, `pothyn` →

`python` ✓

Demo 2: counting letters

How many ‘r’ letters are there in the following word: “strawberry”?

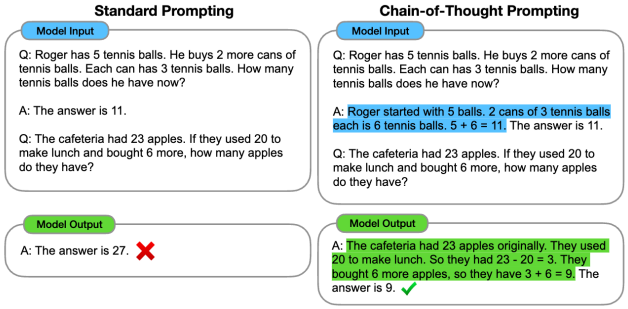
There are 2 ‘r’ letters in “strawberry”. ✗

Count how many ‘r’ letters are there in the following words:
“red”: 1, “roar”: 2, “strawberry”:

3 ✓

Why is this hard at all? The model sees *tokens*, not letters. The examples do more than fix the output format: they steer the model toward the right computation, working letter by letter. A first step toward chain-of-thought (next slide).

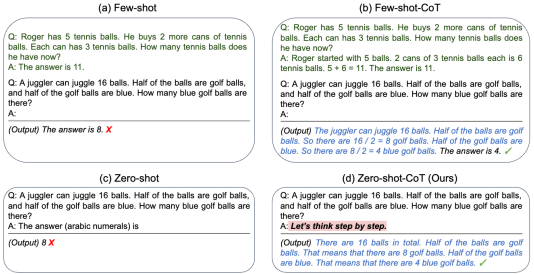
Chain-of-thought prompting: ask for the steps



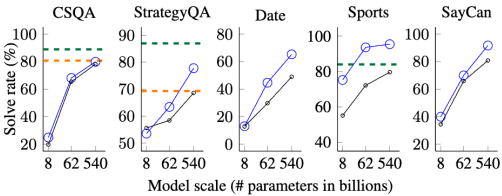
Few-shot chain-of-thought (Wei et al., 2022).

- Show (or just request) **intermediate reasoning steps** before the final answer.
- Large gains on math and logic problems.
- Why it helps: one forward pass does a **fixed amount of computation** per token. Writing the steps out puts each intermediate result back into the input, so the model gets a *scratchpad* and, in effect, more steps of computation.

Chain-of-thought works zero-shot, and mostly for large models



Just append *"Let's think step by step."* (Kojima et al., 2022).



CoT helps once models are large enough; small models produce fluent but wrong chains (Wei et al., 2022).

Foreshadowing the end of today: reasoning models (o1, R1) learn to produce long chains of thought *on their own*, via reinforcement learning.

The limits of prompting

- **Fragile:** results depend on wording and example choice.
- **Expensive at use time:** examples must be re-sent with every query.
- **A base model is not trying to help you:** it continues text (the moon-landing demo).
- **Nothing persists:** prompting cannot permanently add a skill, a style, or a safety rule.

To change what the model does *by default*, change θ.

Next: supervised fine-tuning: show it what good responses look like.

Then: reinforcement learning: tell it how good its own responses were.

Part 4

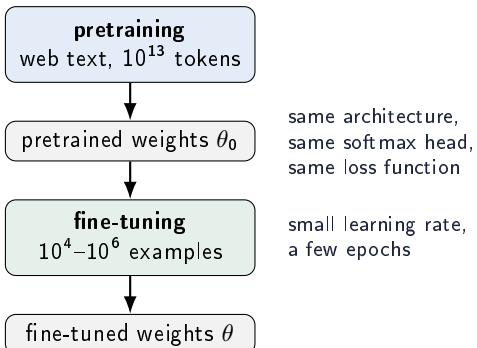
Supervised fine-tuning

following instructions

Generative AI & LLMs
Language modeling
Prompting & in-context learning
▶ **Supervised fine-tuning**
RL basics
Alignment
Verifiable rewards & GRPO
Wrap-up

Pretrain once, fine-tune many times

- **Pretraining** is done *once*, by a few labs: trillions of tokens, months of compute, tens of millions of dollars.
- **Fine-tuning** *continues training the same network* on a small, targeted dataset. Hours on one GPU.
- The pretrained weights θ_0 are the starting point. Fine-tuning moves them a little and keeps what pretraining learned.
- The fine-tuning we care about today is **instruction tuning**: turning the text continuer into an assistant.



Instruction tuning: the data

Instruction–response pairs written or curated by people:

Label the sentiment of this sentence: *The movie wasn't that great.*

negative

Translate to Spanish: *The office was built in three months.*

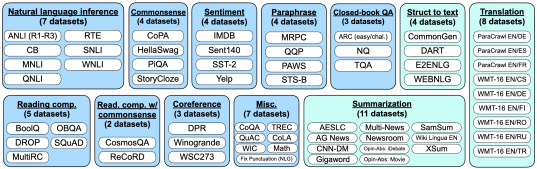
La oficina se construyó en tres meses.

Chat format: multi-turn, with roles, serialized with special tokens:

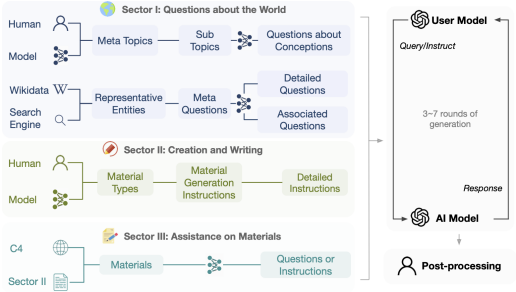
```
{"role": "user", "content": "Weather today?"}
{"role": "assistant", "content": "Sunny, high
of 75."}
{"role": "user", "content": "And tomorrow?"}
{"role": "assistant", "content": "Partly
cloudy, 72."}
```

- The model learns when to *start* (right after the <assistant> marker), when to *stop* (by emitting <eos>), and to follow the instruction.
- 10^4–10^6 examples, tiny next to 10^{13} pretraining tokens.

Where instruction data comes from



FLAN (Wei et al., 2022): 62 existing NLP datasets in 12 task clusters, each rephrased as natural-language instructions.



UltraChat (Ding et al., 2023): millions of multi-turn dialogues generated by having one LLM play the user and another the assistant.

- **Diversity** of tasks and phrasings matters more than the number of examples.
- Today much of the data is synthetic: strong models write the demonstrations, humans filter.

SFT loss: the same cross-entropy, on the response tokens

Supervised fine-tuning (SFT) objective on pairs (x, y) = (prompt, response):

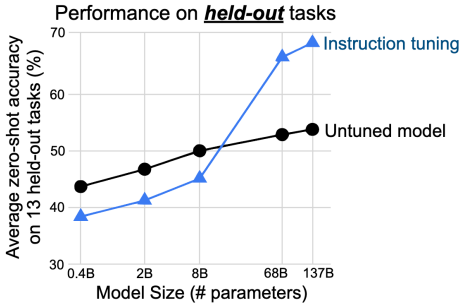
$$\mathcal{L}_{\text{SFT}}(\theta) = -\sum_{(x,y)\in\mathcal{D}} \sum_{t=1}^{|y|} \log p_\theta\big(y_t \mid x, y_{<t}\big)$$



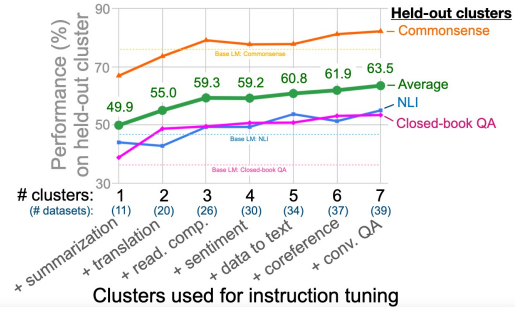
- Identical to the pretraining loss, except prompt tokens are **context only** ("loss masking").
- Same network, same softmax head, same optimizer. Maximizing the likelihood of the demonstrations = learning to **imitate** the demonstrated behavior ("behavior cloning" in RL terms).

Nothing new in the math, only new data. What it learns: the *format and behavior* of an assistant.

What instruction tuning gives you: zero-shot generalization



Zero-shot accuracy on *held-out* task types vs. model size (FLAN, Wei et al., 2022).



More task clusters in the fine-tuning mix ⇒ better on unseen clusters.

- Train on 60 tasks phrased as instructions ⇒ the model follows **new** instructions with no examples in the prompt.
- Chat-style data does the same for conversation: this is how a text continuer becomes a chatbot.

The limits of imitation

SFT = imitate human-written responses. Four problems:

1. **Costly supervision:** someone must *write* a full, high-quality response for every prompt.
2. **Every token counts the same:** the loss penalizes a typo exactly like a factual error.
3. **A ceiling:** the model can only copy the demonstrator, and for open-ended tasks there is no single "right" answer.
4. **Judging is easier than writing:** it is far easier to say *which of two answers is better*, or whether an answer is *correct*, than to write the best answer.

A new paradigm: learn from evaluation, not demonstration



Let the model *generate* its own responses, *score* them, and update toward the ones that scored well. That is **reinforcement learning**.

Two kinds of score:

- **Human preference:** "A is better than B" → a learned reward model (RLHF), or used directly (DPO).
- **Correctness:** "the answer is right" → a program that checks it (RLVR, GRPO).

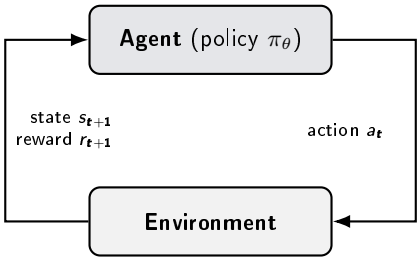
Part 5

RL basics

learning from a score instead of a label

Generative AI & LLMs
Language modeling
Prompting & in-context learning
Supervised fine-tuning
▶ RL basics
Alignment
Verifiable rewards & GRPO
Wrap-up

Reinforcement learning in one picture



The agent–environment loop (Sutton & Barto, *Reinforcement Learning: An Introduction*).

- **State** s_t**:** what the agent observes; **action** a_t**:** what it does.
- **Reward** r_t**:** a number from the environment. It is the *only* feedback the agent gets.
- **Policy** $\pi_\theta(a \mid s)$**:** a distribution over actions given the state. In other words, *a classifier again*.
- **Goal:** maximize the expected total reward $J(\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_t r_t\right]$.

Examples: Atari and Go (reward = score, win), a walking robot (distance), . . . and a chatbot?

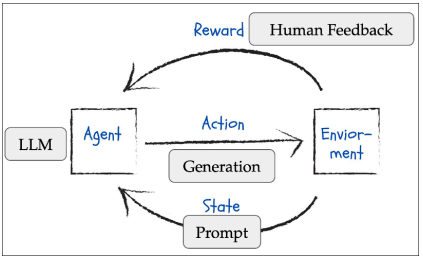
Supervised learning vs. reinforcement learning

	supervised learning (so far)	**reinforcement learning**
feedback	the *correct output* $y^\star$ for each input	only a *score* r for the output you produced
data	fixed dataset, i.i.d.	generated by *your own* policy, so it changes as you learn
granularity	per example, per token	often one number at the *end* of a long sequence
objective	minimize loss against labels	maximize expected reward
hard parts	(none of these)	**exploration**, **credit assignment**, **instability**, etc.

Why bother? For "is this helpful?" or "is this proof correct?", a **score** is far cheaper than a **demonstration**.

The LLM as a policy

- **State** s_t = (prompt x, tokens generated so far $y_{<t}$)
- **Action** a_t = the next token $y_t \in V$
- **Policy** $\pi_\theta(a_t \mid s_t) = p_\theta(y_t \mid x, y_{<t})$, the same softmax classifier as in language modeling
- **Episode** = one complete response $y = (y_1, \ldots, y_T)$
- **Reward** $R(x, y)$ = a score for the *whole* response, given once at the end



Prompt = state, generation = action, feedback = reward.

Objective: $J(\theta) = \mathbb{E}_{x\sim\mathcal{D}}\,\mathbb{E}_{y\sim\pi_\theta(\cdot|x)}\big[R(x,y)\big]$

Where does R come from? Either a **reward model** learned from human preferences (RLHF), or a **program** that checks the answer (RLVR).

Policy gradient: how to increase a reward you can only sample

Goal: maximize $J(\theta) = \sum_y \pi_\theta(y \mid x)\, R(x, y)$. We cannot differentiate R (a human judgment, a checker), but we can differentiate π_θ:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \sum_y \nabla_\theta \pi_\theta(y \mid x)\, R(x, y) && R \text{ does not depend on } \theta \\
&= \sum_y \pi_\theta(y \mid x)\, \nabla_\theta \log \pi_\theta(y \mid x)\, R(x, y) && \text{because } \nabla\pi = \pi\, \nabla \log \pi \\
&= \mathbb{E}_{y \sim \pi_\theta(\cdot \mid x)} \Big[R(x, y)\, \nabla_\theta \log \pi_\theta(y \mid x) \Big] && \text{an expectation: estimate it with samples}
\end{aligned}$$

REINFORCE (Williams, 1992): sample a response y, then $\theta \leftarrow \theta + \alpha\, R(x, y)\, \nabla_\theta \log \pi_\theta(y \mid x)$.

Why the log? The sum runs over all $|V|^T$ possible responses. The identity $\nabla\pi = \pi\, \nabla \log \pi$ turns it into an *average over samples* from the current policy.

And it splits per token: $\log \pi_\theta(y \mid x) = \sum_t \log p_\theta(y_t \mid x, y_{<t})$. Same form as an SFT step, but on the model's *own sample* y, not a label, and weighted by R.

Read it as: sample → score → SFT-shaped step on the sample

	gradient	where y comes from
SFT	$1 \cdot \nabla_\theta \log p_\theta(y^\star \mid x)$	a *human* wrote $y^\star$ (a label)
RL	$R(x, y) \cdot \nabla_\theta \log \pi_\theta(y \mid x)$	the *model* sampled y (not a label)

Policy gradient has the same form as an SFT step, but the sequence is the model's own sample and it is weighted by how good that sample was.

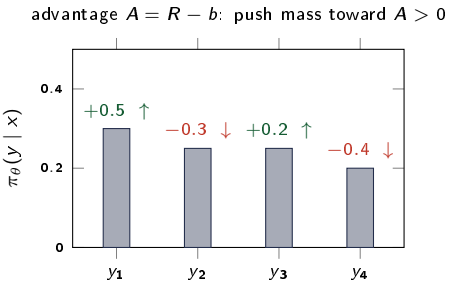
- Nothing changes in the network, the softmax head, or the optimizer.
- What changes: the data is the model's own output, and each sequence carries a *weight*.

Baselines and the advantage: good *relative to what*?

- If every response scores 0.6–0.9, *all* get pushed up. Only differences carry information.
- Fix: subtract a **baseline** $b(x)$ that does *not depend on the sampled* y:

$$\nabla_\theta J = \mathbb{E}\big[(R(x,y) - b(x))\,\nabla_\theta \log \pi_\theta(y \mid x)\big]$$

- Still unbiased, since $\mathbb{E}_y[\nabla_\theta \log \pi_\theta(y \mid x)] = 0$. A *well-chosen* b (near the expected reward) cuts the variance a lot; a poor one can even raise it.
- **Advantage** $A = R - b$: "how much better than expected".
- b: a learned value function ("critic") in classic RL; a *group of samples* in GRPO.



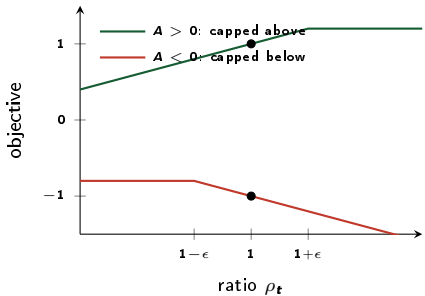
Four sampled responses to one prompt, rewards 0.9, 0.1, 0.6, 0.0, baseline $b = 0.4$.

Stability trick 1: don't move too far in one step (PPO clipping)

- The policy *generates its own training data*: one step too far → garbage samples → worse step. (In supervised learning the next batch corrects a bad step.)
- **PPO** (Schulman et al., 2017): per token, let $\rho_t = \dfrac{\pi_\theta(y_t \mid \cdot)}{\pi_{\text{old}}(y_t \mid \cdot)}$ and optimize

$$\min\big(\rho_t A,\ \text{clip}(\rho_t, 1-\epsilon, 1+\epsilon)\, A\big).$$

- Once a token's probability has moved by $\pm\epsilon$ (e.g. 20%), it gets no further push this round.



PPO's clipped objective for one token ($|A| = 1$).

Stability trick 2: stay close to where you started (KL penalty)

- Subtract a penalty for drifting away from a frozen **reference model** π_{ref} (usually the SFT model):

$$\max_{\theta} \; \mathbb{E}_{y \sim \pi_\theta}\big[R(x,y)\big] \; - \; \beta \, \mathrm{KL}\big(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\text{ref}}(\cdot \mid x)\big)$$

- Keeps the language fluent, prevents collapsing onto one answer, and limits exploiting flaws in the reward.
- β trades off "get more reward" against "stay a normal language model".

You've seen this (regularization): L_2 keeps the weights w near 0; the KL term keeps the *policy* π_θ near π_{ref}. Same idea, different space.

Part 6

Alignment

learning from human preferences

Generative AI & LLMs
Language modeling
Prompting & in-context learning
Supervised fine-tuning
RL basics
▶ **Alignment**
Verifiable rewards & GRPO
Wrap-up

Why alignment? Predicting text is not the same as helping

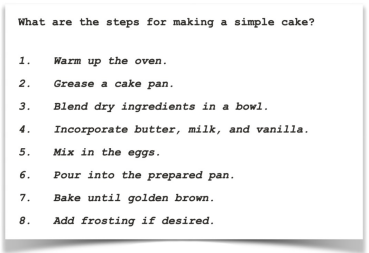
- **Objective mismatch:** "predict the next word" says nothing about being helpful, truthful, or safe.
- **Data bias:** web text contains toxic and misleading content; the LM does not distinguish good from bad behavior.
- GPT-3 (2020) was impressive but awkward to use; ChatGPT (2022) = the same kind of model + **alignment**.
- The target, "HHH": **helpful**, **honest**, **harmless**.

How do you write a loss function for ...

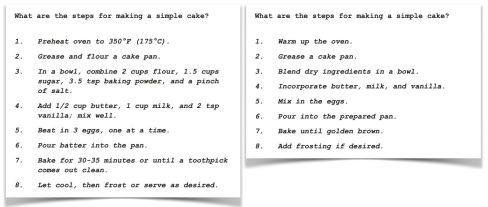
what is helpful? what is funny?
what is ethical? what is safe?

You don't. You **learn** it from human judgments, then optimize it with RL.

Step 1: collect comparisons, not scores



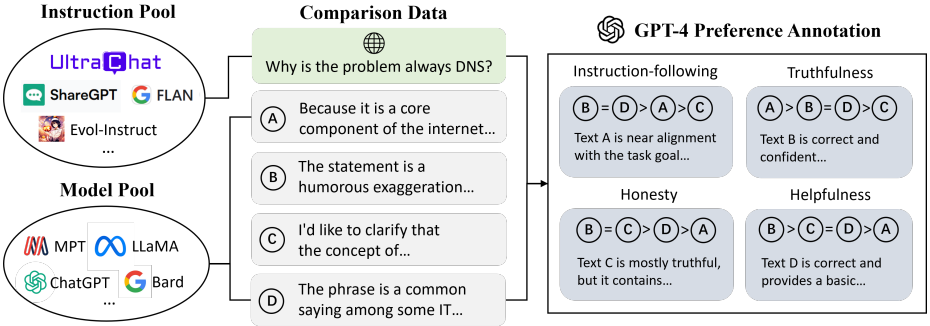
"Rate this answer 1–10": different people give very different numbers for the same text.



"Which of the two is better?": humans are much more consistent at relative judgments.

Preference data = triples $(x,\ y_w,\ y_l)$: prompt, **w**inning response, **l**osing response, both sampled from the model.

Preference data at scale: UltraFeedback



- Cui et al., 2023: 60k prompts from many instruction datasets, 4 responses per prompt from different LLMs.
- Each response rated by **GPT-4** on instruction-following, truthfulness, honesty, helpfulness ("RLAIF": AI feedback instead of human feedback).
- Highest-scoring response = y_w; a random other one = y_l. This is the dataset in Colab notebook 1.

Step 2: a reward model = logistic regression on pairs

Bradley–Terry model. Each response gets a hidden score $r_\phi(x, y) \in \mathbb{R}$; the chance that humans prefer y_w over y_l is

$$P(y_w \succ y_l \mid x) = \sigma\big(r_\phi(x, y_w) - r_\phi(x, y_l)\big), \qquad \sigma(z) = \tfrac{1}{1+e^{-z}}$$

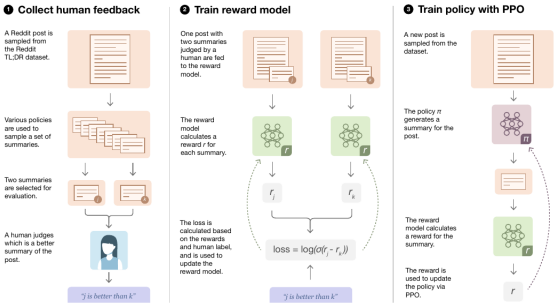
Training: maximize the likelihood of the observed comparisons:

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

- **Architecture:** the LLM backbone with the softmax head replaced by a linear layer that outputs *one number* (read at the last token).
- After training, r_ϕ is an **automatic judge** of any response: a cheap, differentiable proxy for human preference. Only a proxy, though: it can be fooled.

You've seen this (logistic regression): $P(\text{label} = 1) = \sigma(\text{score})$, trained by minimizing the log loss. Here the score is a *difference of two learned scores* and the label is "w beats l".

Step 3: RLHF, maximize the learned reward on a KL leash



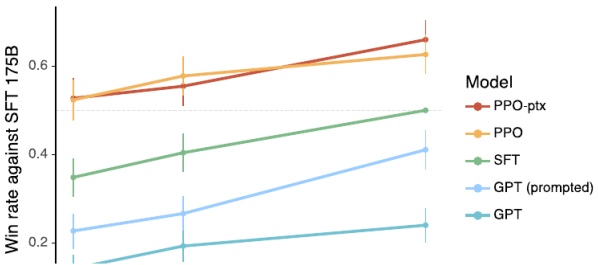
Collect comparisons → train the reward model → train the policy with PPO (Stiennon et al., 2020; Ouyang et al., 2022).

Objective: the policy-gradient recipe with $R = r_\phi$:

$$\max_\theta \; \mathbb{E}_{y\sim\pi_\theta}\big[r_\phi(x,y)\big] - \beta\,\mathrm{KL}\big(\pi_\theta \,\|\, \pi_{\mathrm{SFT}}\big)$$

- Optimized with policy gradient (PPO): sample, score with r_ϕ, weighted update on the sample, clip, repeat.
- **Reward hacking:** the policy finds outputs r_ϕ over-rates (long, confident, flattering); hence the KL leash.

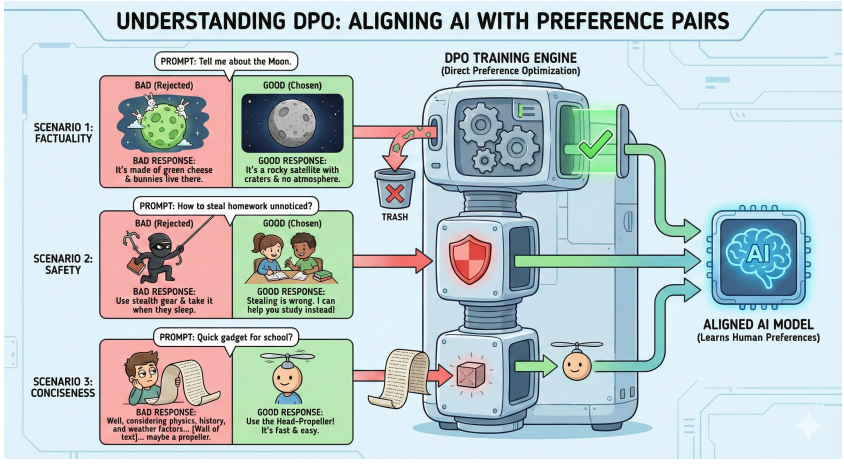
RLHF works: a 1.3B aligned model beats a 175B raw one



Human preference win rate against the 175B SFT model (InstructGPT, Ouyang et al., 2022).

- Humans preferred the **1.3B** RLHF model (PPO) over the **175B** GPT-3, a model 100× larger.
- Alignment is cheap compared with pretraining: tens of thousands of comparisons, not trillions of tokens.
- Every chat assistant since (ChatGPT, Claude, Llama-chat, ...) uses some version of this recipe.

Shortcut: DPO at a glance



- RLHF is heavy: a reward model, online sampling, PPO hyperparameters.
- **Direct Preference Optimization** (Rafailov et al., 2023): train the policy *directly* on the preference pairs. No reward model, no sampling.

DPO in three lines

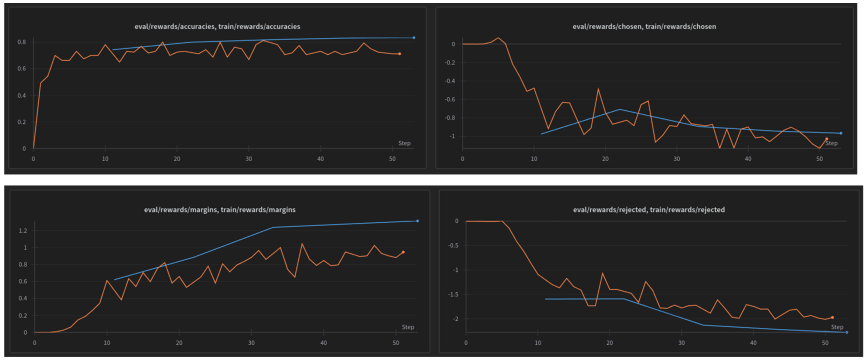
1. The KL-regularized objective has a **closed-form optimum**: $\pi^\star(y \mid x) \propto \pi_{\text{ref}}(y \mid x)\, e^{r(x,y)/\beta}$.
2. Invert it: $r(x,y) = \beta \log \dfrac{\pi^\star(y \mid x)}{\pi_{\text{ref}}(y \mid x)} + \text{const}$, a policy's **implicit reward**.
3. Plug it into the Bradley–Terry loss (Step 2); the constant cancels:

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

- **Gradient:** raise $\log \pi_\theta(y_w \mid x)$, lower $\log \pi_\theta(y_l \mid x)$, weighted by $\sigma(\hat{r}_\theta(y_l) - \hat{r}_\theta(y_w))$, so pairs the implicit reward gets *wrong* count most.

You've seen this (logistic regression): Still logistic regression on pairs, but a response's "score" is now how much more likely the policy makes it than the reference did. *The policy is the reward model.*

DPO in practice (Colab notebook 1)



Zephyr-7B (SFT) + UltraFeedback pairs, LoRA, one Colab GPU. Left: reward accuracy and margin. Right: implicit rewards $\hat{r}_\theta$ of the chosen and the rejected response.

- **Accuracy** ($\hat{r}_w > \hat{r}_l$) and the **margin** rise, then plateau: the policy learns the ranking.
- **Both** implicit rewards drift *down*: the loss only cares about the margin, and lowering $\pi_\theta(y_l)$ is the easiest way to widen it. One fix: add an SFT term on y_w.

RLHF vs. DPO: what you trade

	RLHF (PPO)	DPO
reward model	trained separately	implicit in the policy
sampling during training	online (model's own outputs)	none, fixed pairs (offline)
models in memory	4 (policy, reference, reward, value)	2 (policy, reference)
tuning	PPO hyperparameters	one knob: β
weakness	reward hacking	limited to the given pairs

Same preference data, same KL-regularized target. DPO = the cheap, stable default; RLHF = stronger when you can afford online sampling. Both are logistic-regression-flavored updates on $\log \pi_\theta$.

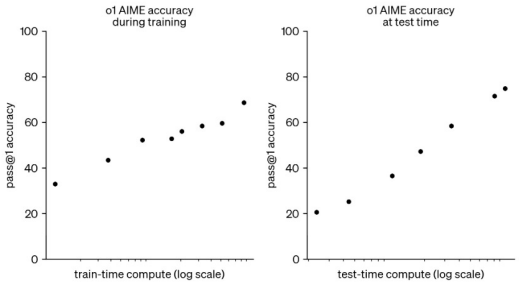
Part 7

Verifiable rewards & GRPO

learning to reason

Generative AI & LLMs
Language modeling
Prompting & in-context learning
Supervised fine-tuning
RL basics
Alignment
▶ **Verifiable rewards & GRPO**
Wrap-up

Reasoning models: think first, answer later



OpenAI o1 (2024): AIME accuracy grows with RL training compute (left) *and* with thinking time at test time (right).

- A **reasoning model** writes a long chain of thought *before* its final answer: chain-of-thought as a learned default.
- **o1** (Sept. 2024): closed recipe, clear message: RL training + "thinking time".
- **DeepSeek-R1** (Jan. 2025): open weights, open recipe: the algorithm is **GRPO**.

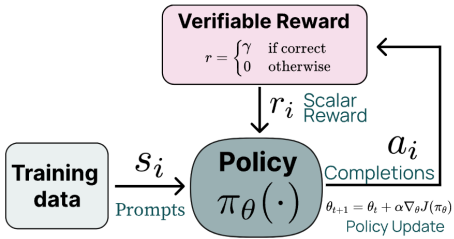
Nobody writes the chains of thought. Where does the training signal come from? From checking the *final answer*.

Verifiable rewards: let a program be the judge

RL with verifiable rewards (RLVR):

$$R(x, y) = \begin{cases} 1 & \text{answer in } y \text{ passes the checker} \\ 0 & \text{otherwise} \end{cases}$$

- **Math:** compare the boxed final answer with the answer key.
- **Code:** run the unit tests.
- **Logic, games, formal proofs:** a checker exists.
- Plus a small **format reward** (answer in `\boxed{}`) so the output can be parsed.



The RLVR loop (Tülu 3, Lambert et al., 2024): prompts in, completions out, a verifier scores them, policy gradient updates θ.

No human label per sample, no learned reward model to fool, unlimited re-checking.

Why RL, not just SFT on worked solutions?

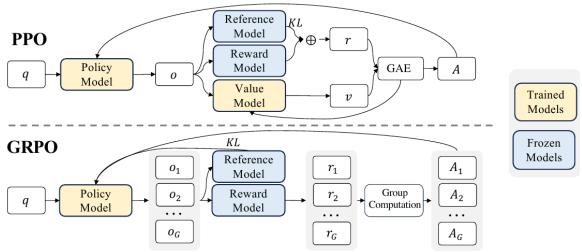
	SFT on human solutions	**RLVR**
data	a person must write every full solution	only the *answer key* is needed
what is optimized	imitate the human's steps (maybe clumsy)	the *outcome*; the model finds its own path
distribution	learns from human-written text	learns from *its own* samples, which is what it faces at inference
behaviors	only what was demonstrated	can *discover* re-checking, backtracking, trying another approach

RLHF makes models *sound* aligned (polite, helpful, safe) but does not make them reason better. For reasoning, the reward must be about *correctness*.

From PPO to GRPO: drop the critic, use the group

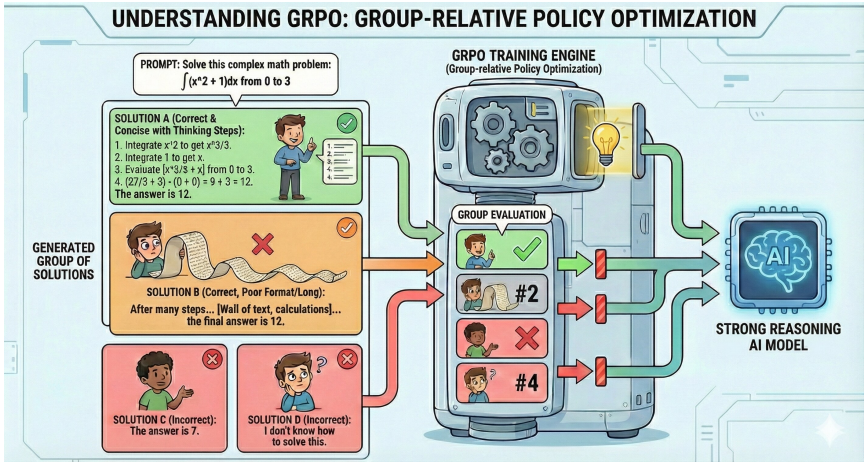
- PPO's baseline $b(x)$ comes from a learned **value network**, a second LLM-sized model trained alongside the policy.
- Expensive: doubles memory and compute.
- Fragile: with one reward at the *end* of a long chain, token-level value estimates are noisy.

GRPO (DeepSeekMath, Shao et al., 2024): we already sample several responses per question, so let them be *each other's baseline*. No critic.



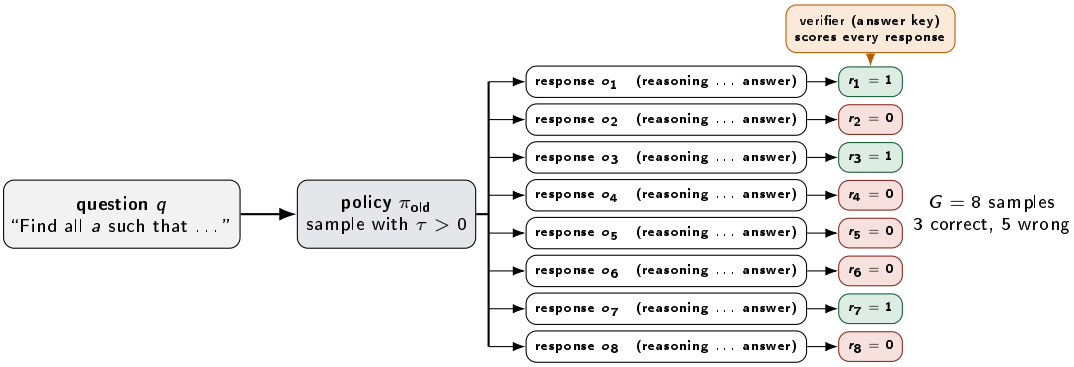
PPO keeps a value model (yellow = trained); GRPO replaces it by a "group computation" over G sampled outputs.

GRPO at a glance



Sample a **group** of solutions to one problem → check each one → reinforce the ones that did better than the group.

GRPO step 1: sample a group of answers to the same question



- For each question q: sample G responses from $\pi_{\text{old}}(\cdot \mid q)$ with temperature $\tau > 0$; the verifier scores each, $r_i \in \{0, 1\}$.
- Typical: $G = 8$–64 per question, hundreds of questions per batch. The model's *own attempts* are the data.

GRPO step 2: the advantage is relative to the group

$$\hat{A}_i = \frac{r_i - \text{mean}(r_1, \ldots, r_G)}{\text{std}(r_1, \ldots, r_G) + \varepsilon} \qquad \text{and the same } \hat{A}_i \text{ goes to } \textit{every token} \text{ of response } o_i.$$

- The **baseline is the group mean**, "how hard is this question for the current model", obtained for free.
- Dividing by the std puts easy and hard questions on one scale.
- All right or all wrong $\Rightarrow \hat{A}_i = 0$: no gradient. Learning happens where the model *sometimes* succeeds.

You've seen this (baselines and the advantage): advantage = reward − baseline; the baseline now comes from samples instead of a critic. Fine print: the group mean includes r_i itself, so it is not quite independent of sample i (a small bias; leaving r_i out, as in RLOO, removes it).

GRPO step 2: a worked example

Rewards of the $G = 8$ samples from step 1: 1, 0, 1, 0, 0, 0, 1, 0

$$\text{mean} = 3/8 = 0.375, \qquad \text{std} = \sqrt{0.375 \cdot 0.625} \approx 0.48$$

$$\hat{A}_{\text{correct}} = \frac{1 - 0.375}{0.48} \approx +1.29, \qquad \hat{A}_{\text{wrong}} = \frac{0 - 0.375}{0.48} \approx -0.77$$

- The 3 correct responses are pushed *up* with weight $+1.29$; the 5 wrong ones pushed *down* with -0.77.
- Had 7 of 8 been correct: $+0.38$ for the correct ones and -2.65 for the lone mistake: the rare failure is what gets corrected.
- Had 0 of 8 been correct: all advantages 0: this question teaches nothing *yet*.

GRPO step 3: the objective

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{q,\{o_i\}}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\underbrace{\min\Big(\rho_{i,t}\,\hat{A}_i,\ \text{clip}(\rho_{i,t}, 1-\epsilon, 1+\epsilon)\,\hat{A}_i\Big)}_{\text{PPO clipping}}\right] - \underbrace{\beta\,\text{KL}(\pi_\theta \,\|\, \pi_{\text{ref}})}_{\text{KL leash}}$$

$\rho_{i,t} = \pi_\theta(o_{i,t} \mid q, o_{i,<t}) \,/\, \pi_{\text{old}}(o_{i,t} \mid q, o_{i,<t})$ is the per-token probability ratio; $\hat{A}_i$ comes from step 2.

How to read it. Ignore the clip and KL for a moment ($\rho \approx 1$): the gradient is

$$\frac{1}{G}\sum_i \hat{A}_i \cdot \nabla_\theta \log \pi_\theta(o_i \mid q) \quad \text{(per token)}$$

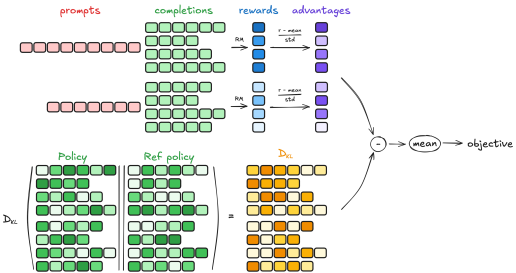
= an **SFT-shaped update on the model's own responses, weighted by advantage**: correct chains reinforced, wrong ones suppressed; clip and KL keep each step small.

GRPO step 3: the algorithm

In five lines:

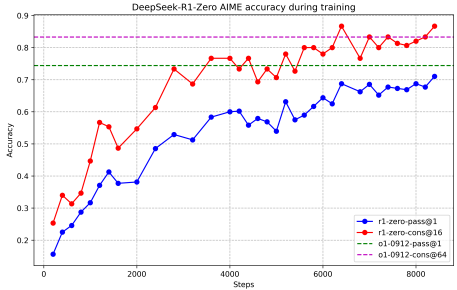
1. Take a batch of questions; set $\pi_{\text{old}} \leftarrow \pi_\theta$.
2. Sample G responses per question from π_{old}.
3. Score with the verifier; compute $\hat{A}_i$ per group.
4. A few gradient steps on J_{GRPO} (clipped, KL-regularized).
5. Repeat for thousands of batches.

No reward model, no value model: the policy, a frozen reference copy, and a checker.

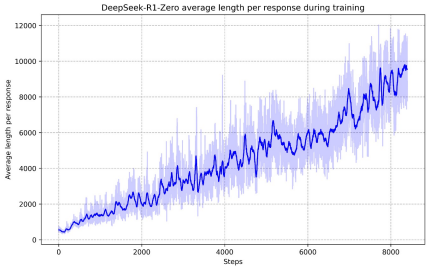


What a GRPO trainer computes (Hugging Face TRL): completions → rewards → group-normalized advantages; policy vs. reference log-probs → KL; both → the objective.

What happens during training: DeepSeek-R1-Zero



AIME 2024 accuracy during GRPO training (pure RL from the base model, no SFT).



Average response length during training: the model *learns to think longer*.

- AIME pass@1: **15.6%** → **71.0%** (86.7% with majority vote over 64 samples), on par with o1.
- The reward only says *correct / incorrect*; longer thinking, re-checking, and backtracking **emerged** because they raise the chance of a correct answer.

The "aha moment"

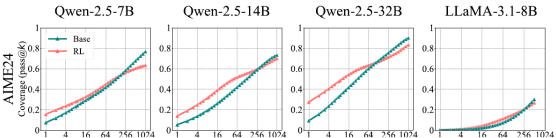
Question: If $a > 1$, then the sum of the real solutions of $\sqrt{a - \sqrt{a+x}} = x$ is equal to

Response: <think>
To solve the equation $\sqrt{a - \sqrt{a+x}} = x$, let's start by squaring both $\cdots$
$\left(\sqrt{a - \sqrt{a+x}}\right)^2 = x^2 \implies a - \sqrt{a+x} = x^2$.
Rearrange to isolate the inner square root term:
$(a - x^2)^2 = a + x \implies a^2 - 2ax^2 + (x^2)^2 = a + x \implies x^4 - 2ax^2 - x + (a^2 - a) = 0$
$\cdots$
Wait, wait. Wait. That's an aha moment I can flag here.
Let's reevaluate this step-by-step to identify if the correct sum can be $\cdots$
We started with the equation:
$\sqrt{a - \sqrt{a+x}} = x$
First, let's square both sides:
$a - \sqrt{a+x} = x^2 \implies \sqrt{a+x} = a - x^2$
Next, I could square both sides again, treating the equation: $\cdots$
$\cdots$

An intermediate R1-Zero checkpoint, mid-solution (DeepSeek-AI, 2025).

- The model stops, flags a problem with its own approach, and *re-evaluates*, in a tone nobody put in the training data.
- This is what "learning from evaluation" buys: behaviors that *work* get reinforced, whether or not a human would have written them.
- RL optimizes *the reward*, not readability or brevity; hence R1 adds SFT and RLHF stages after R1-Zero.

Open questions



pass@k (solved if *any* of k samples is correct) vs. k on AIME24: the RL model wins at small k, the *base* model catches up at large k (Yue et al., 2025).

- **New reasoning, or sharper sampling?** Much of RLVR's gain is sampling efficiency; RL can also narrow the paths the model explores ("diversity collapse").
- **Exploration and diversity** of reasoning trajectories: entropy bonuses, asymmetric clipping and dynamic sampling (DAPO), prolonged RL with KL resets (ProRL).
- **Sparse, outcome-only rewards** vs. step-level rewards; reward hacking even with verifiers.

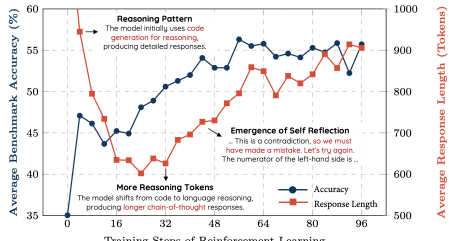
Hands-on: two Colab notebooks

Notebook 1: preference fine-tuning (DPO)

- SFT model (Zephyr-7B, QLoRA) + UltraFeedback preference pairs.
- Watch accuracy, margin, and the implicit rewards (the curves from the DPO slide).

Notebook 2: GRPO with a verifiable reward

- Math problems + answer keys; rule-based checker: $r \in \{0, 1\}$ + format reward.
- TRL `GRPOTrainer`: sample G, score, group-normalize, clipped update + KL.



What to expect while GRPO trains: accuracy and response length rise together; self-reflection appears (SimpleRL-Zoo, Zeng et al., 2025).

DPO: https://colab.research.google.com/drive/1i7aDdW-RHNNPgCCHW9Ky858riqYbrN1R
GRPO: https://colab.research.google.com/drive/1hydSFeV8hpTleZTp5EwpoDCjrqRp8JQ7
Tip: Colab Pro is free for students: `colab.research.google.com/signup` → "No cost for students and educators".

GRPO in fifteen lines of PyTorch

```
def grpo_loss(model, ref_model, batch, eps=0.2, beta=0.04):
    logits     = model(batch["input_ids"]).logits          # current policy
    ref_logits = ref_model(batch["input_ids"]).logits      # frozen reference
    # log-prob of each generated token: the same quantity as in the SFT loss
    logprobs     = token_logprobs(logits,     batch["labels"])
    ref_logprobs = token_logprobs(ref_logits, batch["labels"])
    kl = torch.exp(ref_logprobs - logprobs) - (ref_logprobs - logprobs) - 1
    # group-relative advantages: rewards has shape [num_prompts, G]
    r = batch["rewards"]
    advantages = (r - r.mean(dim=1, keepdim=True)) / (r.std(dim=1, keepdim=True) + 1e-8)
    advantages = advantages.reshape(-1, 1)                  # broadcast to every token
    # PPO-style clipped surrogate + KL penalty
    ratio = torch.exp(logprobs - batch["old_logprobs"])
    surr1 = ratio * advantages
    surr2 = torch.clamp(ratio, 1 - eps, 1 + eps) * advantages
    return -torch.min(surr1, surr2).mean() + beta * kl.mean()
```

- `logprobs`: the same $\log p_\theta(y_t \mid x, y_{<t})$ as in the SFT loss. `advantages`: the worked example of step 2.
- `surr1`/`surr2`: the PPO clip; `kl`: the KL leash. Simplified from TRL's `GRPOTrainer` (which also masks padding, handles several update epochs).

Part 8

Wrap-up

one gradient, many weightings

Generative AI & LLMs
Language modeling
Prompting & in-context learning
Supervised fine-tuning
RL basics
Alignment
Verifiable rewards & GRPO
▶ **Wrap-up**

The whole pipeline through this course's lens

Stage	Data	Loss	Optimization	What it learns
Pretraining	web text, $\sim 10^{13}$ tokens, no labels	cross-entropy on every next token (= MLE)	Adam, one pass, months on thousands of GPUs	language, knowledge, skills: a "text continuer"
Prompting / ICL	a few examples in the input	none	none (weights frozen)	task adaptation at inference time
SFT	10^4–10^6 (prompt, response) pairs	cross-entropy on response tokens	Adam, 1–3 epochs, small LR	assistant format and behavior
Reward model	preference pairs (x, y_w, y_l)	logistic loss on score differences (Bradley–Terry)	Adam (a classifier)	a proxy for human preference
RLHF / DPO	own samples (RLHF); preference pairs (DPO)	$-\mathbb{E}[r_\phi] + \beta\,\mathrm{KL}$; DPO: logistic loss on implicit rewards	PPO: clip + KL; DPO: plain Adam	helpfulness, tone, refusals
RLVR / GRPO	problems + checker; G own samples per problem	$-\mathbb{E}[\hat{A}_i \log \pi_\theta]$ + clip + KL, $r \in \{0, 1\}$	GRPO: group baseline, no critic	long chains of thought that end in correct answers

One gradient, many weightings

Every stage updates the same network with the same kind of gradient:

$$\theta \leftarrow \theta + \alpha \sum_{\text{sequences } y} w(y)\, \nabla_\theta \log p_\theta(y \mid x)$$

What changes is **where y comes from** and **what the weight w is**.

stage	where y comes from	weight $w(y)$	you know it as
pretraining	the web (x = preceding text)	1	MLE / cross-entropy
SFT	a human demonstration	1	MLE on response tokens
RLHF (policy gradient)	the model's own sample	$\hat{A} = r_\phi(x, y) - b(x)$	SFT-shaped, weighted
GRPO / RLVR	G samples per problem	$\hat{A}_i = (r_i - \text{mean})/\text{std}$	SFT-shaped, group baseline
DPO	a preference pair	$\pm\beta\, \sigma(\hat{r}_l - \hat{r}_w)$	logistic regression

Plus two stability tricks whenever y is self-generated: **clip** the per-token update, **KL-penalize** drift from the reference model.

Where this goes next in CS 4774

- 10/20: Deep learning on text: bag-of-words to **attention**
- 10/22: **Reinforcement learning** (value functions, Q-learning, Gym)
- 10/27: **LLM architecture and training** (inside today's black box)
- 10/29: Multimodal and vision-language models
- 11/05: **Prompting, RAG, and fine-tuning** (incl. LoRA, more on DPO)
- 11/10: Efficient LLMs and deployment
- 11/12, 11/17: **Agents**, tool use, multi-agent systems, evaluation, safety

Readings (all free online)

- Jurafsky & Martin, *Speech and Language Processing* (3rd ed.), Ch. 7–10: LMs, Transformers, fine-tuning, alignment.
- Brown et al. (2020), *Language Models are Few-Shot Learners*: GPT-3 and in-context learning.
- Wei et al. (2022), *Chain-of-Thought Prompting Elicits Reasoning in LLMs*.
- Ouyang et al. (2022), *Training language models to follow instructions with human feedback*: InstructGPT / RLHF.
- Rafailov et al. (2023), *Direct Preference Optimization*.
- Shao et al. (2024), *DeepSeekMath*: GRPO; DeepSeek-AI (2025), *DeepSeek-R1*.
- Lambert et al. (2024), *Tülu 3*: RL with verifiable rewards.
- Yue et al. (2025), *Does RL really incentivize reasoning capacity in LLMs beyond the base model?*

Acknowledgments and figure credits

These slides adapt material from:

- Prof. Yu Meng (UVA CS), NLP course lectures, Fall 2025: *LM pretraining, fine-tuning & decoding*; *In-context learning & scaling*; *LLM reasoning*; *RL for post-training LLMs*; *LLM alignment*.
- Prof. Jing Yang (UVA ECE), ECE 6501 *Reinforcement Learning and Its Application to LLMs*: RL fundamentals, PPO, LLM basics, RLHF/DPO, GRPO, RLVR.
- Prof. Yanjun Qi (UVA CS), CS 4774 Lecture 1 (the machine learning timeline); Peng Wang, CS 4774 practice sessions on DPO and GRPO (2025) and the two Colab notebooks.

Figures: Vaswani et al. 2017 (Transformer architecture); Jurafsky & Martin, SLP3 (feed-forward neural LM, redrawn; pretraining; LM head); Sutton & Barto (agent–environment loop); OpenAI tokenizer and *Learning to reason with LLMs* (o1 plot); Kaplan et al. 2020 (scaling laws); Wei et al. 2022 (emergent abilities; chain-of-thought; FLAN); Brown et al. 2020 (few-shot prompting); Kojima et al. 2022 (zero-shot CoT); lmarena.ai (ICL demos); Ding et al. 2023 (UltraChat); Cui et al. 2023 (UltraFeedback); lm-class.org (preference-data cartoons); Stiennon et al. 2020 and Ouyang et al. 2022 (RLHF pipeline and results); Hugging Face TRL docs (GRPO visual) and a TRL DPO tutorial (DPO training curves); Lambert et al. 2024 (RLVR loop); Shao et al. 2024 (PPO vs. GRPO); DeepSeek-AI 2025 (R1-Zero curves, "aha moment"); Yue et al. 2025 (pass@k); SimpleRL-Zoo, HKUST NLP (training curve); DPO/GRPO overview illustrations generated with Gemini; tug-of-war illustration generated with ChatGPT.

Thank you!

Three things to remember

1. An LLM is a next-token classifier (softmax, cross-entropy, SGD) trained on the whole web. Generation is that classifier in a loop.
2. Post-training reuses the *same* network and the *same* gradient: SFT changes the data; RL changes the *weights* on self-generated data.
3. Reasoning emerges when the reward is *correctness* and the model is free to search: sample a group, score, reinforce the winners (GRPO).

Questions?

Peng Wang
University of Virginia

Colab notebooks (DPO, GRPO): links on the "Hands-on" slide.