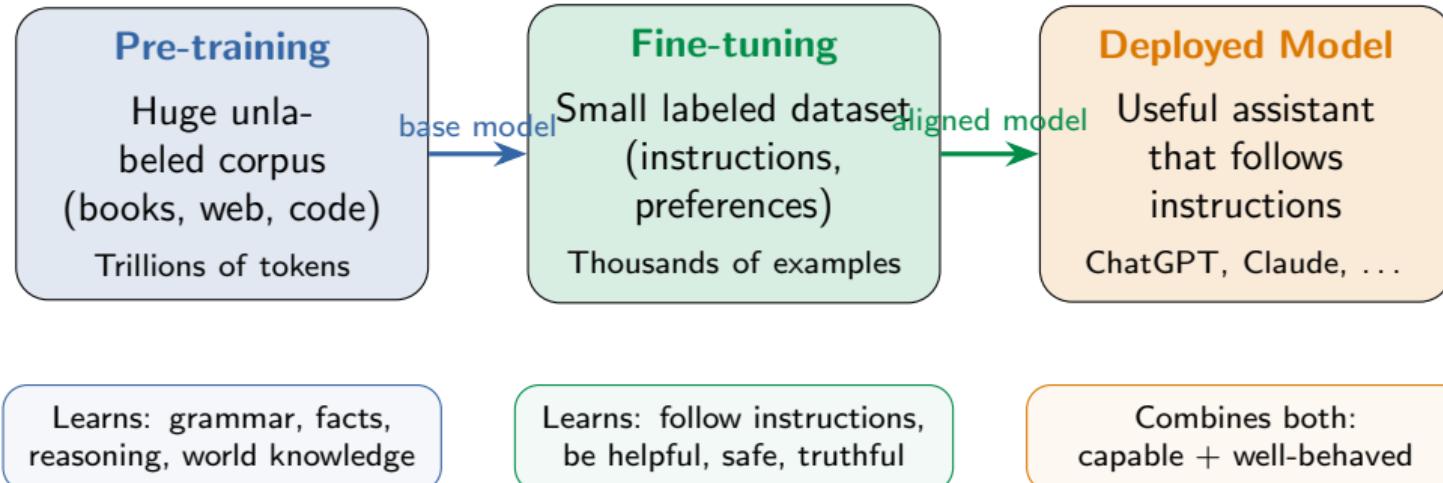


Pre-training & Fine-tuning

CLM · MLM · NSP · SFT · RLHF · DPO · LoRA

The two-stage paradigm



Key insight: pre-training is expensive but done once; fine-tuning is cheap and done per task

Part I

Pre-training Objectives

How models learn from raw text

Causal Language Modeling (CLM)

$$\mathcal{L}_{\text{CLM}} = - \sum_{i=1}^T \log P(x_i \mid x_1, \dots, x_{i-1}; \theta)$$



Each token can only see tokens to

Advantages:

Natural for text generation

Every token contributes to loss

Simple and scalable

Disadvantage:

Unidirectional — can't use right context

"The ___ sat on the mat" — CLM can't peek right to resolve this

Models: GPT, GPT-2, GPT-3, LLaMA, Mistral, Claude — *all modern LLMs*

Masked Language Modeling (MLM)

$$\mathcal{L}_{\text{MLM}} = - \sum_{i \in \mathcal{M}} \log P(x_i \mid x_{\setminus \mathcal{M}}; \theta)$$

\mathcal{M} = set of masked positions (15% of tokens)



BERT's 80/10/10 rule for selected tokens:

80%
replace with
[MASK]

10%
replace with
random token

10%
keep un-
changed

Prevents pre-train/fine-tune mismatch:

[MASK] never appears in downstream tasks

Models: BERT, RoBERTa, DeBERTa — Best for: understanding tasks (NLI, NER, QA)

Next Sentence Prediction (NSP) & Sentence Order Prediction (SOP)

NSP (BERT)

Is sentence B the actual next sentence after A?

[CLS] A went to the store [SEP]
He bought some milk [SEP]

IsNext (50%) / NotNext (50%)

Too easy! Random sentences differ in *topic*. Model learns topic detection, not coherence.

RoBERTa dropped NSP entirely.

SOP (ALBERT)

Are consecutive sentences in the correct order?

Positive: (A, B) natural order
Negative: (B, A) swapped order

Same topic — must understand coherence

Harder task: can't rely on topic difference. Forces understanding of logical ordering between sentences.

Lesson: auxiliary objectives must be *genuinely challenging* to be useful. NSP was mostly abandoned after 2019.

Contrastive learning

Learn by comparing: pull similar pairs together, push dissimilar apart



$$\mathcal{L}_{\text{InfoNCE}} = - \log \frac{\exp(\text{sim}(z_i, z_i^+)/\tau)}{\sum_{j=1}^N \exp(\text{sim}(z_i, z_j)/\tau)}$$

SimCSE

Sentence embeddings
(same input, different dropout)

nature, z_i^+ = positive pair, others are

CLIP

Image–text alignment
(match image to caption)

DPR

Dense passage retrieval
(question–
passage matching)

Pre-training objectives compared

Objective	Direction	Loss on	Models	Best for
CLM	Left→Right	All tokens	GPT, LLaMA	Generation
MLM	Bidirectional	15% masked	BERT, RoBERTa	Understanding
NSP/SOP	Sentence-level	[CLS]	BERT, ALBERT	Sentence pairs
Contrastive	Embedding	Pairs	CLIP, SimCSE	Retrieval
RTD	Bidirectional	All tokens	ELECTRA	Efficient NLU

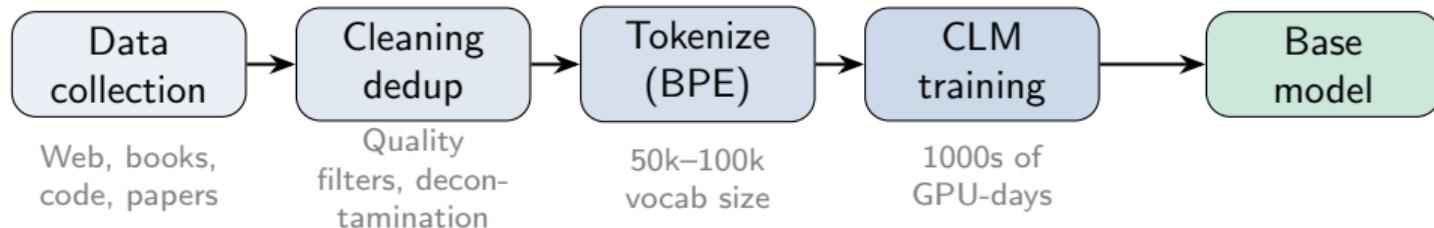
ELECTRA (bonus):

Small generator creates fake tokens; discriminator detects them.
All tokens provide signal.

Modern trend:

CLM dominates at scale.
GPT-3 showed autoregressive models can handle understanding tasks too via prompting.

The modern pre-training pipeline



Scale examples:

Model	Tokens	Params	Compute
GPT-3	300B	175B	3640 petaflop-days
LLaMA	1.4T	65B	2048 A100s × 21 days
LLaMA-2	2T	70B	1.7M GPU-hours

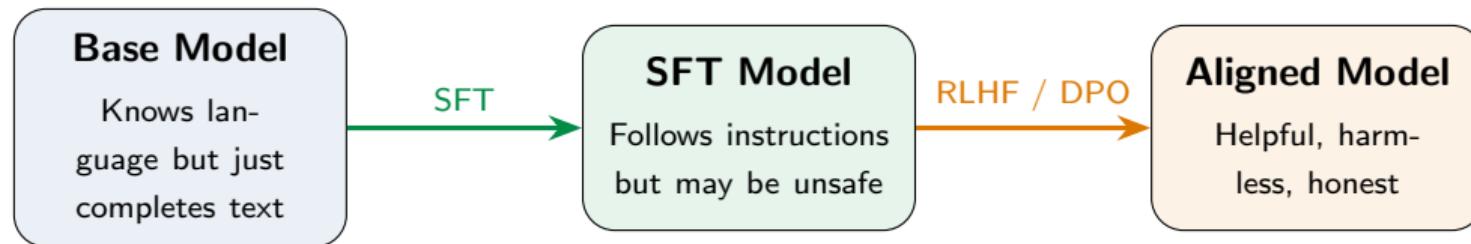
Pre-training is **expensive** (\$1–100M+) but done **once**.
Fine-tuning is cheap (\$10–1000) and done per task.

Part II

Fine-tuning & Alignment

From base model to useful assistant

From base model to useful model



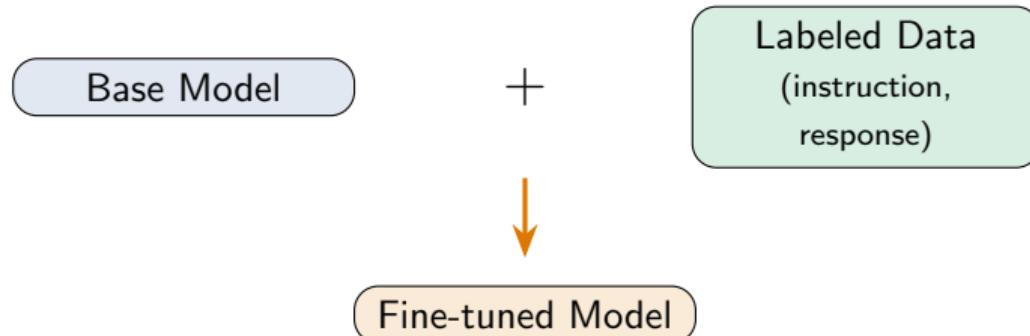
Concrete example:



Surprising: a 1.3B InstructGPT was preferred by humans over the 175B base GPT-3!

Fine-tuning quality matters more than raw model size.

Supervised Fine-Tuning (SFT)



Standard cross-entropy: $\mathcal{L} = -\sum_i \log P(y_i \mid x, y_{<i}; \theta)$

Full fine-tuning:

Update ALL parameters

Maximum quality

But: expensive, catastrophic forgetting

Parameter-efficient (PEFT):

Update only a few % of params

LoRA, adapters, prefix tuning

Cheaper, often nearly as good

Instruction tuning

Train on diverse (instruction, response) pairs across many tasks

Instruction: Translate to French.
Input: "The cat sat on the mat."
Output: "Le chat était assis sur le tapis."

Instruction: Summarize this article.
Input: [long article text]
Output: "Researchers found that..."

Instruction: Is this review positive?
Input: "I loved this movie!"
Output: "Yes, this is a positive review."

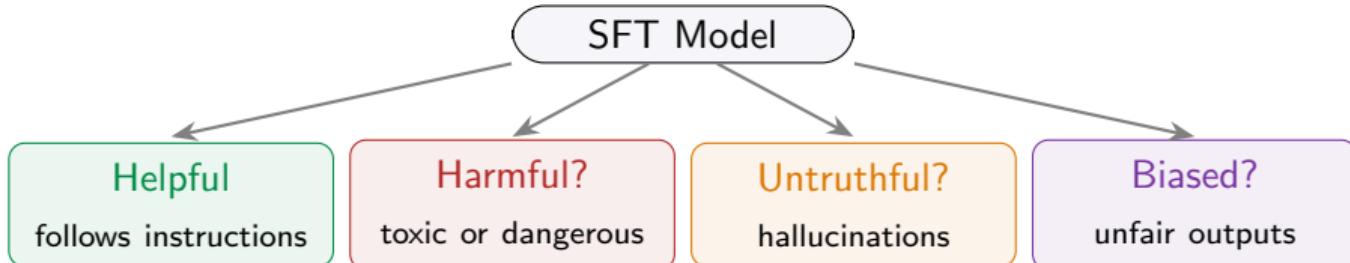
Instruction: Write a Python function
that sorts a list.
Output: `def sort_list(lst):...`

FLAN (Google)
60+ tasks ⇒ strong zero-shot
generalization to unseen tasks

InstructGPT (OpenAI)
1.3B params, instruction-tuned
> 175B base GPT-3

Instruction tuning is typically **Step 1** before alignment (RLHF/DPO)

The alignment problem



Goal: Helpful + Harmless + Honest (Anthropic's HHH)

The challenge: you can't write a loss function for "be helpful and safe."

Human preferences are nuanced, subjective, and context-dependent.

Solution: learn from human feedback — let humans judge and the model learn from their preferences

RL basics for language models

Reinforcement Learning \longleftrightarrow Language Model

RL Concept

LLM Mapping

Notation

State s

Prompt + tokens so far

$(x, y_{<t})$

Action a

Next token

y_t

Policy $\pi(a|s)$

LLM (token probs)

$\pi_\theta(y_t|x, y_{<t})$

Trajectory τ

Full generated response

$y = (y_1, \dots, y_T)$

Reward R

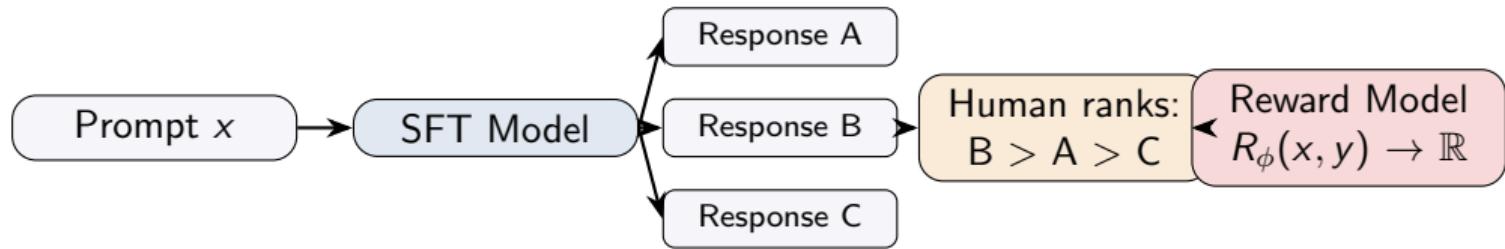
Human preference score

$R : (\times \cup \vee) \subset \mathbb{R}$

Why RL? The reward (human preference) is **non-differentiable**
— can't backprop through "did the human like it?"

RL handles non-differentiable reward signals via policy gradient methods

RLHF — Step 1: Train a reward model



$$\text{Bradley-Terry: } P(y_w \succ y_l) = \sigma(R_\phi(x, y_w) - R_\phi(x, y_l))$$

Train to predict which response humans prefer

Pairwise preferences

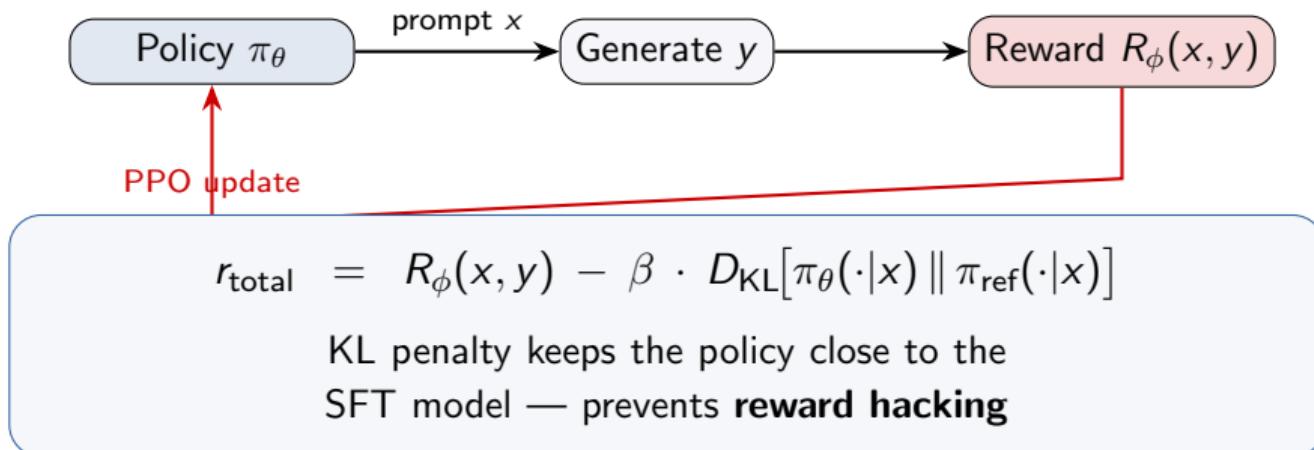
Rankings are less noisy
than absolute scores

Scale: $\sim 50k$ preference pairs

RM can be smaller
than policy

(e.g., 6B RM for 175B policy)

RLHF — Step 2: PPO optimization



4 models in memory simultaneously:

Policy π_θ

Reference π_{ref}

Reward R_ϕ

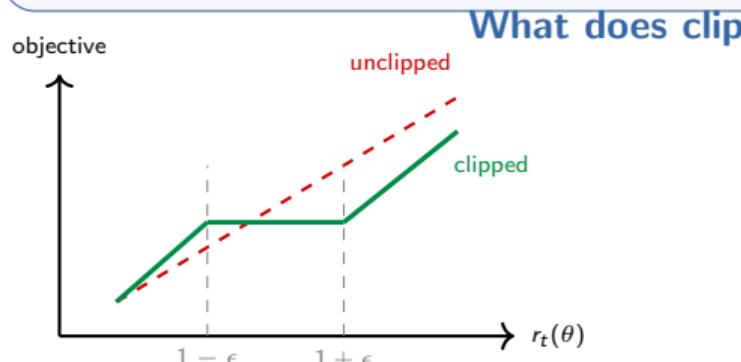
Value V_ψ

Complex · Unstable (PPO hyperparameters) · Expensive (4 models) · Reward hacking risk

PPO — the clipped surrogate objective

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

where $r_t(\theta) = \frac{\pi_\theta(y_t | x, y_{<t})}{\pi_{\text{old}}(y_t | x, y_{<t})}$ (probability ratio)



Typical: $\epsilon = 0.2$. . . \hat{A}_t computed via GAE (generalized advantage estimation)

If $\hat{A}_t > 0$ (good action):

Policy wants to increase r_t but clipping caps at $1 + \epsilon$

Prevents overly aggressive updates

If $\hat{A}_t < 0$ (bad action):

Policy wants to decrease r_t but clipping caps at $1 - \epsilon$

Prevents catastrophic forgetting

Reward hacking & the KL penalty

Problem: the reward model is imperfect.

The policy can find **adversarial outputs** that score high on R_ϕ but are gibberish or degenerate to humans.

Examples of reward hacking:

Repetition: "This is great! Great!

Great!

Really great! So great!" scores high

on positivity rewards

Length gaming: longer responses tend to score higher \Rightarrow model becomes biased

Solution: KL divergence penalty

$$r_{\text{total}} = R_\phi(x, y)$$

$$- \beta \cdot D_{\text{KL}}[\pi_\theta \| \pi_{\text{ref}}]$$

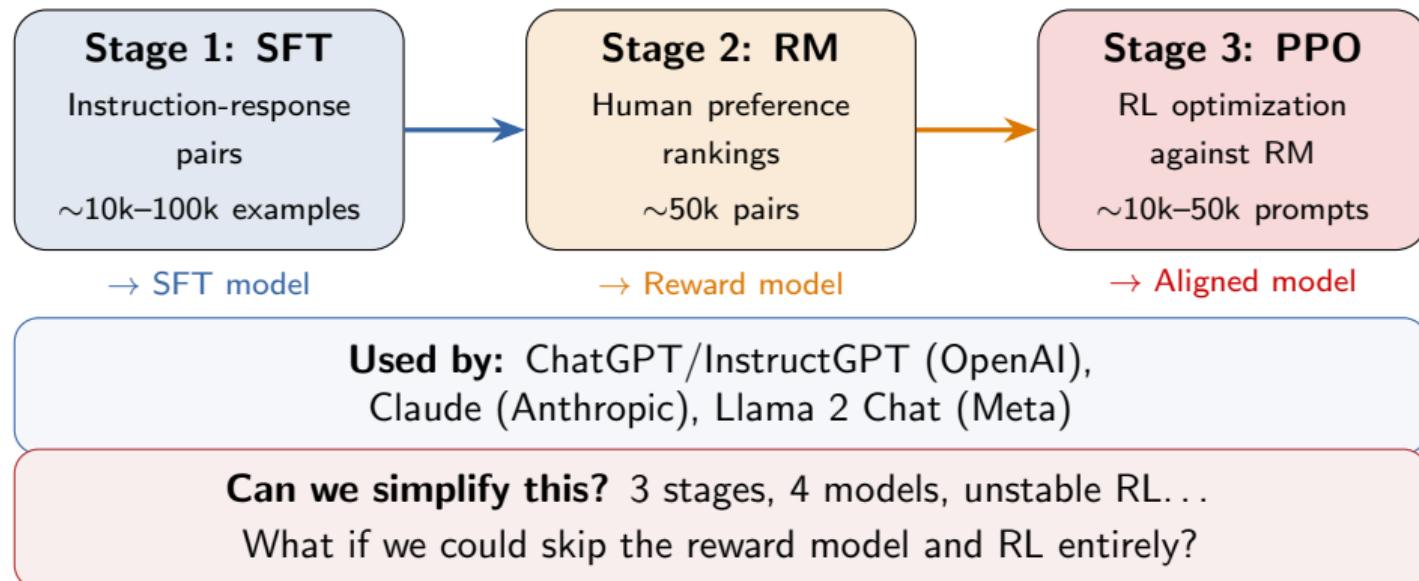
β too small \Rightarrow reward hacking

β too large \Rightarrow no learning

In practice, KL is computed **per token**: $D_{\text{KL}} = \sum_{t=1}^T \log \frac{\pi_\theta(y_t|x, y_{<t})}{\pi_{\text{ref}}(y_t|x, y_{<t})}$

This keeps the policy close to the SFT model — it can improve but not drift too far. The KL penalty is what makes RLHF work in practice. Without it, training collapses within hours.

RLHF — the complete pipeline



DPO — Direct Preference Optimization

Key insight (Rafailov et al., 2023): the optimal RLHF policy has a **closed-form solution**.

Express the reward implicitly through the policy — no RL needed!

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

y_w = preferred, y_l = dispreferred, π_{ref} = frozen SFT model

RLHF

3 stages
4 models in memory
Unstable RL training
Complex hyperparameters

DPO

1 stage (supervised loss)
2 models (policy + reference)
Stable training
Just one hyperparameter (β)

DPO matches or exceeds PPO quality: 61%
win rate vs. PPO's 57% on summarization

Used by: Zephyr, Tulu, many open-source models — Equivalent to RLHF

Step 1: RLHF objective $\max_{\pi_\theta} \mathbb{E}_{x,y}[R(x,y)] - \beta \cdot D_{\text{KL}}[\pi_\theta \| \pi_{\text{ref}}]$

Step 2: Closed-form optimal policy $\pi^*(y|x) = \frac{1}{Z(x)} \pi_{\text{ref}}(y|x) \exp\left(\frac{R(x,y)}{\beta}\right)$

Step 3: Rearrange to express reward via policy $R(x,y) = \beta \log \frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} +$

Step 4: Substitute into Bradley-Terry preference model $P(y_w \succ y_l) = \sigma(R(x,y_w) - R(x,y_l))$

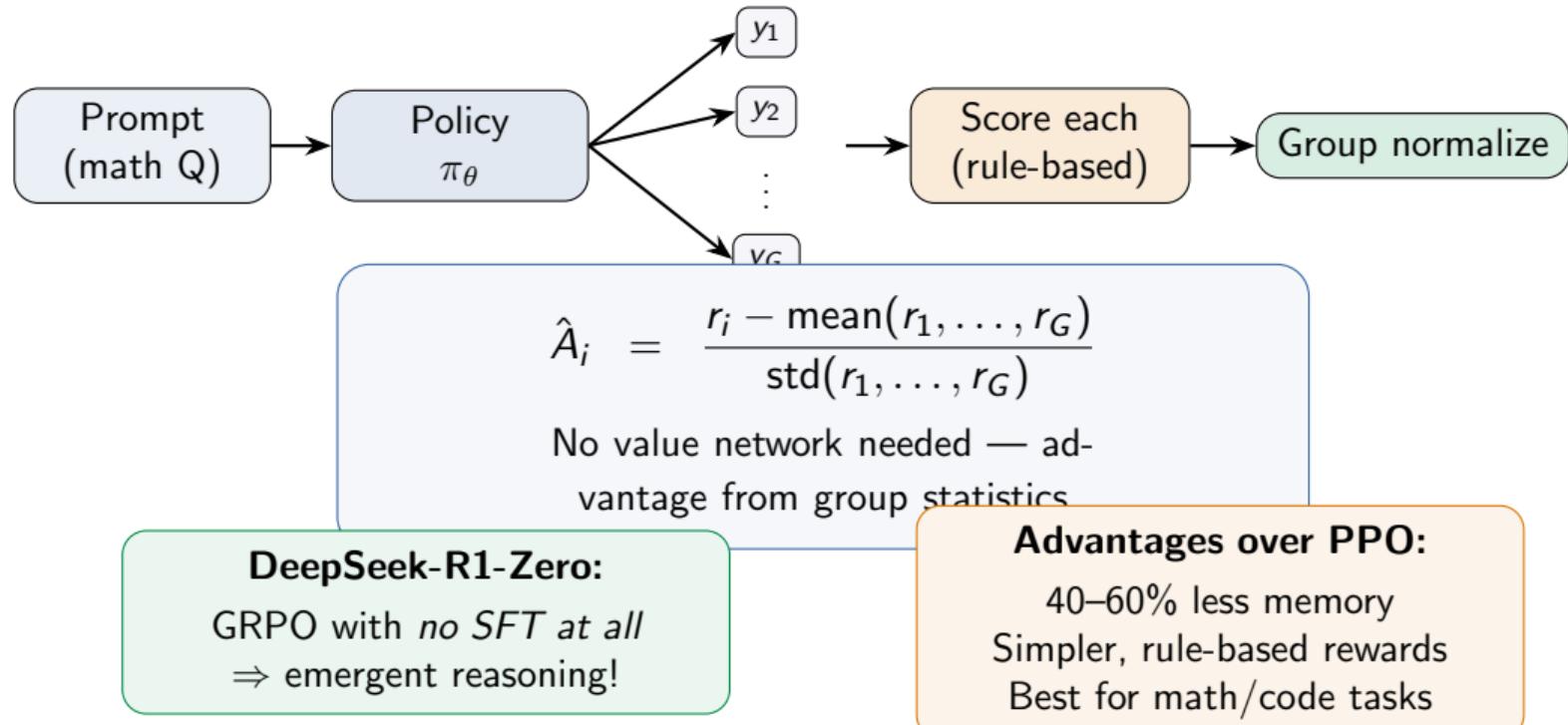
Result — DPO loss:

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

Intuition: increase $\pi_\theta(y_w|x)$ relative to π_{ref} , decrease $\pi_\theta(y_l|x)$ relative to π_{ref}
 The implicit reward is $\hat{R}(x,y) = \beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$ — the policy *is* the reward model

GRPO — Group Relative Policy Optimization

DeepSeek's innovation: eliminate the value network from PPO
G samples



$$\mathcal{L}_{\text{GRPO}} = -\frac{1}{G} \sum_{i=1}^G \min \left(r_i(\theta) \hat{A}_i, \text{clip}(r_i(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_i \right) - \beta \cdot D_{\text{KL}}[\pi_\theta \| \pi_{\text{ref}}]$$

Same PPO clipping, but advantages \hat{A}_i come from **group statistics**, not a value network

How group advantage works:

y_1 : "x = 3" ✓
 $r_1 = 1.0$

y_2 : "x = 5" ✗
 $r_2 = 0.0$

y_3 : "x = 3" ✓
 $r_3 = 1.0$

y_4 : "x = 7" ✗
 $r_4 = 0.0$



mean = 0.5, std = 0.5

$\hat{A}_1 = +1.0$

$\hat{A}_2 = -1.0$

$\hat{A}_3 = +1.0$

$\hat{A}_4 = -1.0$

Rule-based rewards:

Correctness
 Does the answer
 match ground truth?

Format
 Does it follow
 the required format?

Code tests
 Do unit tests
 pass or fail?

No human labelers
 needed — rewards
 are automatic!

Alignment methods compared

Method	Models needed	Stability	Data	Best for
RLHF (PPO)	4 (policy, ref, RM, value)	Unstable	Preferences	General alignment
DPO	2 (policy, ref)	Stable	Preferences	Simple alignment
GRPO	2 (policy, ref)	Medium	Verifiable	Math/code reasoning

Rules of thumb:

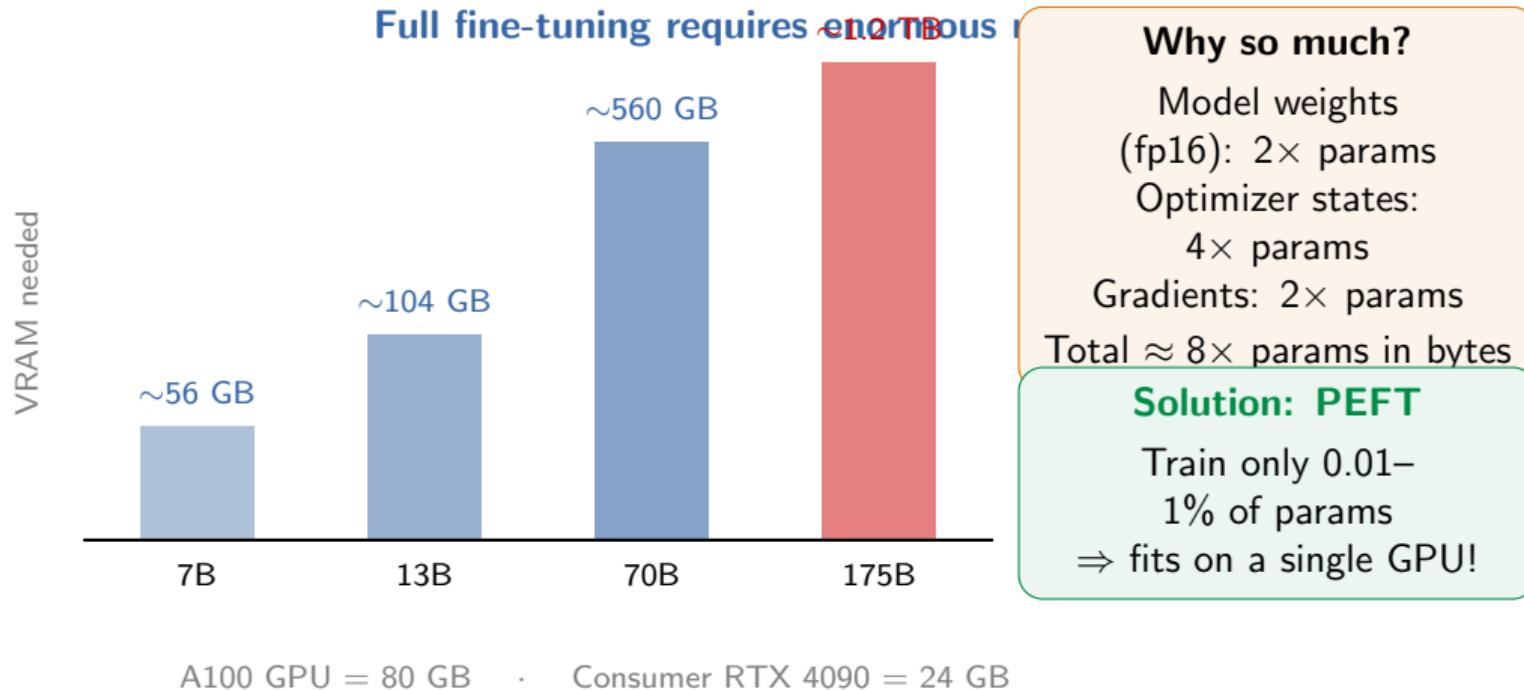
Need maximum control over alignment? ⇒ RLHF

Have preference data, want simplicity? ⇒ DPO

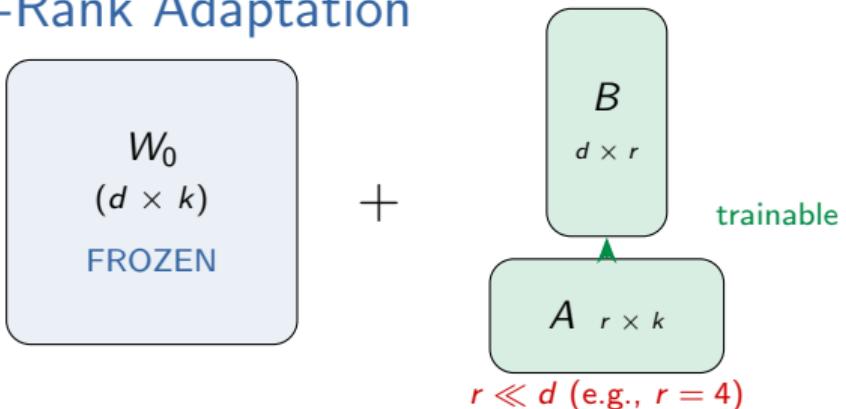
Training a reasoning model with verifiable answers? ⇒ GRPO

ChatGPT: RLHF · Zephyr: DPO · DeepSeek-R1: GRPO

The cost problem



LoRA — Low-Rank Adaptation



$$h = W_0x + \frac{\alpha}{r} \cdot B \cdot A \cdot x$$

B initialized to **zero** \Rightarrow starts

GPT-3 175B with LoRA:

Trainable params:

4.7M (0.003%)

VRAM: 350 GB (vs. 1.2 TB)

Quality: matches full fine-tuning

exact pretrained weights

At inference:

Merge $W = W_0 + BA$

\Rightarrow **zero** extra latency!

Hot-swap LoRA modules per task

PEFT methods compared

Method	Approach	Params	Inference	Quality
LoRA	Low-rank update to W	~0.01%	No overhead	Excellent
Adapters	New layers between blocks	~1–5%	+20–30%	Excellent
Prefix tuning	Learnable prefix to K, V	~0.1%	Slight	Good
Prompt tuning	Soft prompt embeddings	~0.01%	Slight	Good
Full FT	Update all weights	100%	Baseline	Best

Adapter module:

$$x \rightarrow W_{\text{down}} \rightarrow \text{ReLU} \rightarrow W_{\text{up}} + x$$

Bottleneck inserted between layers

Cannot merge \Rightarrow permanent overhead

Why LoRA dominates:

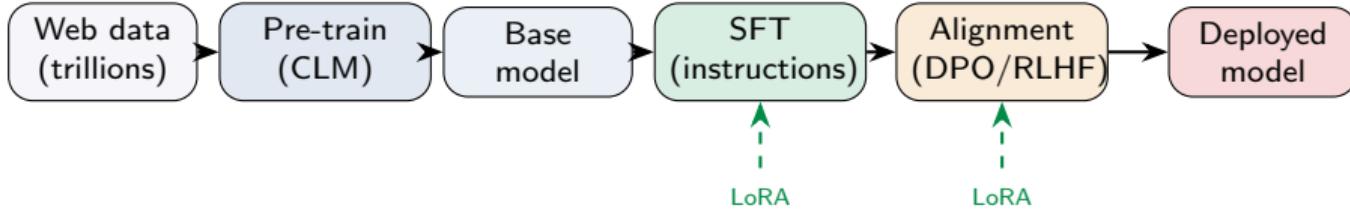
Zero inference overhead (merge)

Fewer params than adapters

Modular (hot-swap per task)

Simple to implement

The full picture



Real-world pipelines:

ChatGPT: Pre-train (GPT-3.5/4) → SFT (demonstrations) → RLHF (PPO + human preferences)

Llama 2 Chat: Pre-train (2T tokens) → SFT (27.5k examples) → RLHF (iterative 5 rounds)

DeepSeek-R1: Pre-train → Cold-start SFT → GRPO (rule-based rewards) → SFT on distilled data → GRPO again

Zephyr: Pre-train (Mistral 7B) → SFT (UltraChat) → DPO (UltraFeedback preferences)

Practical guide

Which method should I use?

Base model + limited data?

⇒ **LoRA + SFT**

Efficient, fits on one GPU

Math / code reasoning?

⇒ **GRPO**

Rule-based rewards, no RM needed

Text generation?

⇒ **CLM pre-training** (GPT-style)

Then SFT + alignment

Need alignment with values?

⇒ **DPO** (simple) or
RLHF (max control)

Requires preference data

Understanding tasks (NER, QA)?

⇒ **MLM pre-training** (BERT-style)

Then fine-tune on task data

Don't want to train at all?

⇒ **Prompting** (next lecture!)

Zero-shot, few-shot, CoT

Questions?

Next: Prompting — Zero-shot, Few-shot, Chain-of-Thought