Pre-training & Fine-tuning

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

The two-stage paradigm

Pre-training

Huge unlabeled corpus (books, web, code)

Trillions of tokens

→ base model →

Fine-tuning

Small labeled dataset (instructions, preferences)

Thousands of examples

→ aligned model →

Deployed Model

Useful assistant that follows instructions

ChatGPT, Claude, ...

Learns: grammar, facts, reasoning, world knowledge

Learns: follow instructions, be helpful, safe, truthful

Combines both: capable + well-behaved

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)

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

The → cat → sat → on → **the** → **???**

Each token can only see tokens to

Disadvantage:

Unidirectional — can't use right context

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

Advantages:

Natural for text generation

Every token contributes to loss

Simple and scalable

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)

The [MASK] sat on the [MASK]

Bidirectional context — sees both left AND right

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

80%	**10%**	**10%**
replace with [MASK]	replace with random token	keep unchanged

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

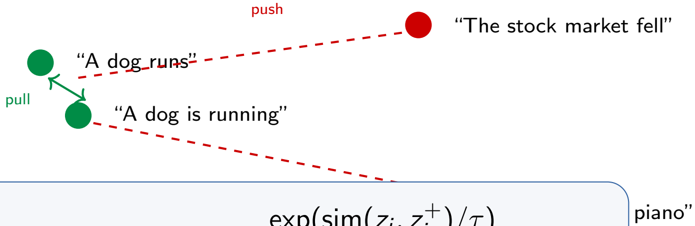
Same topic — must understand coherence

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



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

rature, $z_i^+ =$ positive pair, others a

SimCSE
Sentence embeddings (same input, different dropout)

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

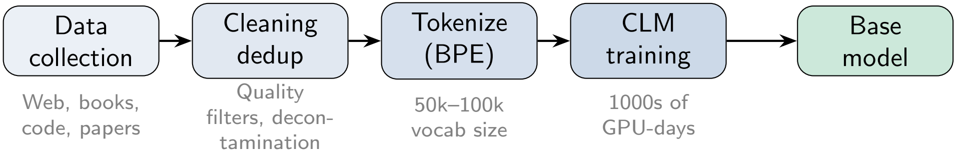
ELECTRA (bonus):

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 $\times$ 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

Base Model
Knows language but just completes text

$\xrightarrow{\text{SFT}}$

SFT Model
Follows instructions but may be unsafe

$\xrightarrow{\text{RLHF / DPO}}$

Aligned Model
Helpful, harmless, honest

Concrete example:

GPT-3
autocomplete

$\rightarrow$

InstructGPT
follows instructions

$\rightarrow$

ChatGPT
aligned assistant

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)

Base Model + Labeled Data (instruction, response)

↓

Fine-tuned Model

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)

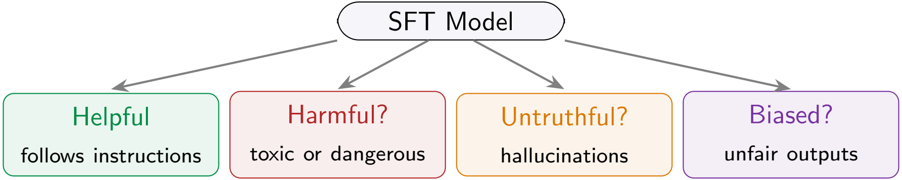
60+ tasks ⇒ strong zero-shot generalization to unseen tasks

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

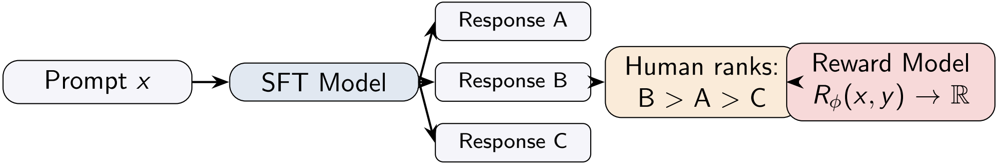
Reinforcement Learning $\longleftrightarrow$ Language Model

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	$y = (y_1, \ldots, y_T)$
Reward R	Human preference score	$R(x, y) \in \mathbb{R}$

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



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

Train to predict which response humans prefer

Pairwise preferences

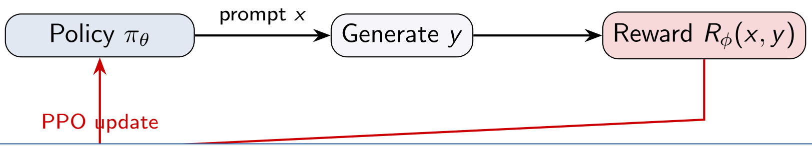
Rankings are less noisy than absolute scores

Scale: $\sim$50k preference pairs

RM can be smaller than policy

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

RLHF — Step 2: PPO optimization



$$r_{\text{total}} \;=\; R_\phi(x, y) \;-\; \beta \cdot D_{\text{KL}}\big[\pi_\theta(\cdot|x) \,\|\, \pi_{\text{ref}}(\cdot|x)\big]$$

KL penalty keeps the policy close to the SFT model — prevents **reward hacking**

4 models in memory simultaneously:

Policy π_θ | Reference π_{ref} | Reward R_ϕ | Value V_ψ

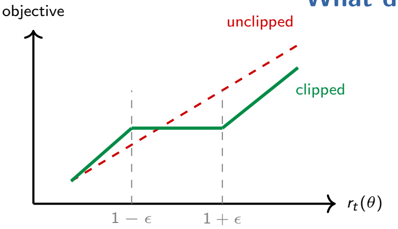
Complex · Unstable (PPO hyperparameters) · Expensive (4 models) · Reward hacking risk

PPO — the clipped surrogate objective

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

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

What does clip



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

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

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 becom

Solution: KL divergence penalty

$$r_{\text{total}} = R_\phi(x, y) - \beta \cdot D_{\text{KL}}\left[\pi_\theta \| \pi_{\text{ref}}\right]$$

β too small $\Rightarrow$ reward hacking
β too large $\Rightarrow$ no learning

e

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

Stage 1: SFT
Instruction-response pairs
∼10k–100k examples

→ SFT model

Stage 2: RM
Human preference rankings
∼50k pairs

→ Reward model

Stage 3: PPO
RL optimization against RM
∼10k–50k prompts

→ Aligned model

Used by: ChatGPT/InstructGPT (OpenAI),
Claude (Anthropic), Llama 2 Chat (Meta)

Can we simplify this? 3 stages, 4 models, unstable RL. . .
What if we could skip the reward model and RL entirely?

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**

DPO — the derivation

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

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

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

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

Result — DPO loss:

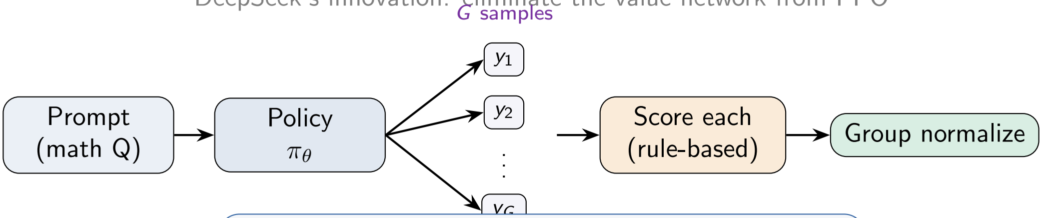
$$\mathcal{L}_{\mathrm{DPO}} = -\mathbb{E}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{\mathrm{ref}}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{\mathrm{ref}}(y_l|x)}\right)\right]$$

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 $\hat{R}(x,y) = \beta \log \frac{\pi_\theta(y|x)}{\pi_{\mathrm{ref}}(y|x)}$ — the policy *is* the reward model

GRPO — Group Relative Policy Optimization

DeepSeek's innovation: eliminate the value network from PPO



$$\hat{A}_i = \frac{r_i - \text{mean}(r_1, \ldots, r_G)}{\text{std}(r_1, \ldots, r_G)}$$

No value network needed — advantage from group statistics

DeepSeek-R1-Zero:

GRPO with *no SFT at all*
$\Rightarrow$ emergent reasoning!

Advantages over PPO:

40–60% less memory
Simpler, rule-based rewards
Best for math/code tasks

GRPO

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

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: "$x = 7$" × $r_4 = 0.0$

$\downarrow$ 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:

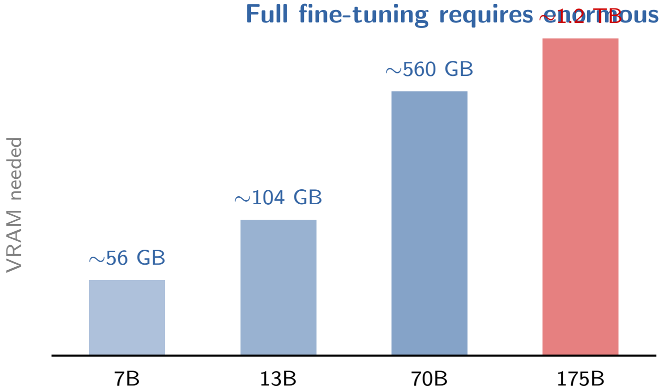
Need maximum control over alignment? $\Rightarrow$ **RLHF**

Have preference data, want simplicity? $\Rightarrow$ **DPO**

Training a reasoning model with verifiable answers? $\Rightarrow$ **GRPO**

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

The cost problem



Why so much?

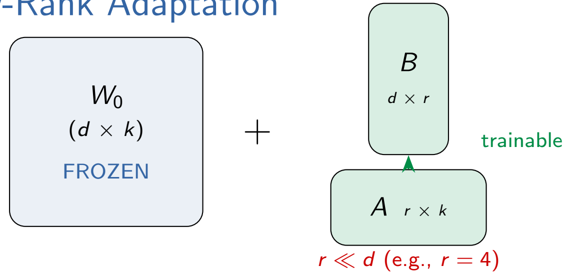
Model weights (fp16): $2\times$ params
Optimizer states: $4\times$ params
Gradients: $2\times$ params
Total $\approx 8\times$ params in bytes

Solution: PEFT

Train only 0.01–1% of params
$\Rightarrow$ fits on a single GPU!

A100 GPU = 80 GB · Consumer RTX 4090 = 24 GB

LoRA — Low-Rank Adaptation



$$h = W_0 x + \frac{\alpha}{r} \cdot B \cdot A \cdot x$$

B initialized to **zero** $\Rightarrow$ starts
ct pretrained weights

GPT-3 175B with LoRA:

Trainable params:
4.7M (0.003%)
VRAM: 350 GB (vs. 1.2 TB)
Quality: matches full fine-tuning

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	$\sim$0.01%	No overhead	Excellent
Adapters	New layers between blocks	$\sim$1–5%	+20–30%	Excellent
Prefix tuning	Learnable prefix to K, V	$\sim$0.1%	Slight	Good
Prompt tuning	Soft prompt embeddings	$\sim$0.01%	Slight	Good
Full FT	Update all weights	100%	Baseline	Best

Adapter module:

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

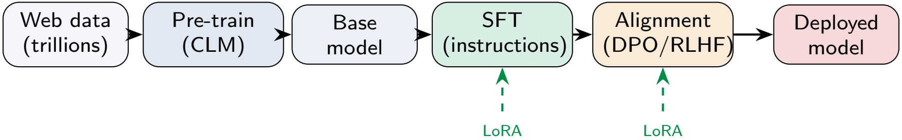
Bottleneck inserted between layers

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?

$\Rightarrow$ **LoRA + SFT**

Efficient, fits on one GPU

Math / code reasoning?

$\Rightarrow$ **GRPO**

Rule-based rewards, no RM needed

Text generation?

$\Rightarrow$ **CLM pre-training** (GPT-style)

Then SFT + alignment

Need alignment with values?

$\Rightarrow$ **DPO** (simple) or **RLHF** (max control)

Requires preference data

Understanding tasks (NER, QA)?

$\Rightarrow$ **MLM pre-training** (BERT-style)

Then fine-tune on task data

Don't want to train at all?

$\Rightarrow$ **Prompting** (next lecture!)

Zero-shot, few-shot, CoT

Further reading

Pre-training

- Devlin et al. (2019), "BERT" — masked language modeling
- Radford et al. (2019), "GPT-2" — causal language modeling

RLHF & Alignment

- Ouyang et al. (2022), "Training language models to follow instructions with human feedback" (InstructGPT)
- Rafailov et al. (2023), "Direct Preference Optimization" (DPO)
- Shao et al. (2024), "DeepSeekMath: Pushing the Limits of Mathematical Reasoning" (GRPO)

Parameter-Efficient Fine-Tuning

- Hu et al. (2022), "LoRA: Low-Rank Adaptation of Large Language Models"
- Houlsby et al. (2019), "Parameter-Efficient Transfer Learning for NLP" (Adapters)

Questions?

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