Lecture 15: Mid-Training & Post-Training (SFT and RLHF) — From Next-Token Predictors to Controllable Assistants

Language Modeling from Scratch · lecture notes · rebuilt for NVIDIA Blackwell, late 2026

Section 1: From Base Language Models to Controllable Policies

Pretraining a language model on tens of trillions of tokens of web text and code produces an extraordinarily capable statistical predictor of the next token, yet it does not produce a usable assistant. Web documents are not conversations, user requests, or structured software engineering sessions. When given a prompt such as "Summarize the following article in three bullet points," a purely pretrained base model treats the prompt as the opening line of a document scraped from the internet. In the historical example of base **GPT-3 (2020)**, giving the model an instruction often caused it to continue writing more instructions, generate a quiz question, or drift into an unrelated web page rather than executing the task.

What is missing is controllability. Users and downstream applications need fine-grained control over output format, multi-step reasoning, code generation, safety boundaries, and structured tool calls. The goal of **mid-training** and **post-training** is to transform a pretrained base language model from an open-ended document completer into a reliable, steerable policy.

> **DEFINITION**
> **Post-training** is the collection of training stages applied after main pretraining—including supervised fine-tuning, preference optimization, reinforcement learning, and distillation—that shapes a base language model's next-token distribution into a controllable, instruction-following policy.

Three guiding questions structure our analysis of this transformation:

1. What does mid-training and post-training data actually look like, and how do its properties shape model behavior?
2. How do we train on demonstrations via **Supervised Fine-Tuning (SFT)** and on preferences via **Reinforcement Learning from Human Feedback (RLHF)** and **Direct Preference Optimization (DPO)**?
3. What breaks when we optimize too hard against learned rewards—and why did late-2026 frontier pipelines evolve away from the classic recipe?

Before examining individual stages, note an important practical asymmetry in the literature: pre-2023 papers such as Stiennon et al. (2020) and Bai et al. (2022) published full annotation interfaces, instructions, and data breakdowns, whereas modern frontier data recipes are closely guarded. We therefore draw on older public studies for fine-grained empirical mechanisms while anchoring our system architecture in late-2026 frontier technical reports.

How Post-Training Pipelines Evolved: 2022–2024 vs. 2026

To see where every technique in this lecture fits, compare two end-to-end pipeline architectures.

In the **2022–2024 classic recipe** (established by Ouyang et al. 2022 in InstructGPT and followed by historical open models such as Llama 2 and Llama 3), training proceeds through four linear boxes connected sequentially from left to right:

1. **Pretraining** (next-token prediction on bulk web text) →
2. **SFT** (supervised fine-tuning on human-written or distilled demonstrations) →
3. **Reward Model** (training a scalar scoring model on pairwise human comparisons) →
4. **PPO or DPO** (a single RLHF or offline preference-optimization stage producing the final model).

In the **2026 frontier pipeline**—represented by **Nemotron 3.5 Super** (Oct 2026), **Kimi K3** (July 2026), and **DeepSeek-V4.1-Flash** (Sept 2026)—the architecture expands into five distinct stages, replacing single-stage scalar RLHF and offline DPO with specialized online RL and multi-teacher distillation:

1. **Pretrain** (bulk web and code) →
2. **Mid-Training** (long context, dense code, reasoning, and early agentic data) →
3. **Cold-Start SFT** (multi-turn tool use and long packed trajectories) →
4. **Online RL Experts** (joint **Reinforcement Learning with Verifiable Rewards (RLVR)** and RLHF, using automated verifiers alongside generative reward models) →
5. **On-Policy Distillation** (**Multi-Teacher On-Policy Distillation, or MOPD**, consolidating many specialized expert teachers into a single unified model).

A striking structural fact separates these two eras: **none of the three 2026 frontier reports uses offline DPO in its pipeline.**

> **DEFINITION**
> **Multi-Teacher On-Policy Distillation (MOPD)** is a final post-training stage in which a single student model generates rollouts under its own policy and is trained to match the output distributions of multiple domain-specialized expert teacher models, unifying diverse skills without destructive interference.

Each of the three late-2026 frontier pipelines implements this general five-stage blueprint with distinct engineering choices:

Model (Release)	Post-Training Pipeline	Notable Design Choices
Nemotron 3.5 Super (Oct 2026)	SFT → RLVR → expert teacher models → MOPD	Runs RLVR and RLHF **jointly** in one RL stage, routed per environment; SFT packs trajectories up to **524,288 tokens**.
Kimi K3 (July 2026)	SFT cold start → RL into **9 domain/effort experts** → MOPD into one unified model	Uses an agentic generative judge with a prompt-specific rubric and scorepad, plus strict verbosity control against cold-start reference length.
DeepSeek-V4.1-Flash (Sept 2026)	SFT → large-scale asynchronous RL → full-vocabulary on-policy distillation	Distills from **more than 40** specialized teacher models into one 552B-parameter model (16B active parameters per token during decode).

Training specialized teachers once and consolidating them via on-policy distillation solves a chronic problem in multi-domain post-training: when a single RL run simultaneously optimizes coding, math, creative writing, and safety, conflicting reward signals interfere with one another. Separating specialist RL from final on-policy consolidation lets each expert maximize capability in its own domain before all skills are merged cleanly.

COMMON MISCONCEPTION

You might think that because 2026 frontier labs do not use offline DPO in their flagship pipelines, DPO is obsolete and not worth deriving carefully. That breaks for two reasons: first, DPO remains the standard accessible tool for open-weight and academic post-training when full online RL infrastructure is unavailable; second, the exact closed-form mathematics behind DPO—specifically the nonparametric optimum and implied-reward equivalence we derive in Section 8—is reused directly in modern online RL algorithms covered in Lecture 16.

WORKED EXAMPLE

Tracing a multi-domain model build through the 2022–2024 vs. 2026 pipelines

Setup: Suppose an engineering team wants a single assistant that solves competition math, writes multi-file software patches using shell tools, and holds helpful general-purpose conversations.

Under the 2022–2024 classic recipe:

1. Start from a web-pretrained base model (short context, minimal tool syntax).
2. Run one SFT stage mixing chat dialogues, math answers, and code snippets.
3. Train a single scalar reward model on pairwise human preferences across all three domains.
4. Run one PPO or DPO pass against that scalar reward model. Because one scalar RM scores both math proofs and casual prose, the policy often optimizes stylistic verbosity over factual correctness, and gains in one domain can degrade another.

Under the 2026 frontier pipeline (e.g., Kimi K3 or Nemotron 3.5 Super style):

1. Follow bulk web/code pretraining with **mid-training** to extend context length and raise code, math, and agentic workflow density before post-training begins.
2. Run **cold-start SFT** on long, multi-turn tool-call trajectories so the model reliably invokes shell/code tools and formats responses.
3. Run **online RL** into specialized domain/effort experts—routing math and software tasks to automated verifiers (unit tests, proof/answer checkers in RLVR) and open-ended conversation to an agentic generative judge with explicit length control (RLHF).
4. Run **MOPD** to distill the 9 to 40+ expert teachers into one unified production checkpoint.

Interpretation: Each added stage isolates a failure mode of the classic pipeline—mid-training prevents SFT forgetting, automated verifiers eliminate noisy preference labels on objective tasks, generative judges and length gates fix scalar-RM hacks, and MOPD prevents cross-domain interference.

KEY IDEA

- Pretraining produces a next-token document completer, not a controllable assistant; mid-training and post-training reshape that base distribution into a reliable, steerable policy.
- While the 2022–2024 recipe ended in a scalar reward model plus single-stage PPO or DPO, late-2026 frontier pipelines (Nemotron 3.5 Super, Kimi K3, DeepSeek-V4.1-Flash) rely on mid-training, cold-start SFT, joint online RLVR/RLHF experts, and multi-teacher on-policy distillation (MOPD).

This architectural map sets up the remainder of Lecture 15 (mid-training, SFT, preference data, generative judges, brief PPO, full DPO, and RLHF failure modes) and Lecture 16 (online RL via PPO, GRPO, RLVR, and why online RL plus MOPD replaced DPO at the frontier). To begin at the first bridge after bulk pretraining, we must address what happens when you try to jump straight from raw web pretraining into large-scale supervised fine-tuning.

✎ Check Your Understanding

1. **Yes or No:** Do any of the three late-2026 frontier technical reports (Nemotron 3.5 Super, Kimi K3, DeepSeek-V4.1-Flash) use offline DPO in their post-training pipelines? Briefly justify why DPO is still taught in this course.
   Answer: No; all three frontier pipelines replaced offline DPO with online RL (RLVR + RLHF) and on-policy distillation (MOPD). DPO is still essential because it remains the standard accessible method for open and academic post-training, and its closed-form implied-reward derivation is reused directly in modern online RL in Lecture 16.
2. **Fill in the blank:** In the 2026 frontier pipeline, training specialized RL teachers separately and merging them via _____ allows a single unified model to acquire many domain skills without cross-domain interference.
   Answer: Multi-Teacher On-Policy Distillation (MOPD).

Section 2: Mid-Training — Bridging Bulk Pretraining and SFT

Suppose you have finished main pretraining on tens of trillions of web and code tokens, and you now possess substantial compute along with billions of high-quality instruction, code, and reasoning tokens. Running naive, large-scale Supervised Fine-Tuning directly on top of the raw pretrained checkpoint fails badly: the sudden, sharp distribution shift from diverse web text to narrow instruction formats causes **catastrophic forgetting**, eroding the broad world knowledge and general capabilities acquired during pretraining.

Mid-training exists to solve this exact transition problem. Rather than jumping abruptly from bulk web pretraining to instruction fine-tuning, mid-training inserts an intermediate continued-pretraining stage that gradually steers the model's weights toward target capabilities while preserving general knowledge.

> **DEFINITION**
> **Mid-training** (also called continued pretraining or annealing) is a dedicated training stage run after main pretraining and before SFT that keeps the standard next-token prediction objective while shifting the data mixture toward curated high-quality text, long-context documents, code, math, and reasoning traces.

This produces a clean **three-stage view** of early model shaping:

1. **Main pretraining** on bulk web and code corpora.
2. **Mid-training** on a curated, long-context, reasoning and instruction-enriched mixture.
3. **Short, sharp SFT** to lock in assistant formatting, conversation structure, and tool interfaces.

By keeping the standard next-token prediction objective during mid-training, instruction and reasoning corpora are absorbed as pretraining data. This allows you to scale instruction and domain data by orders of magnitude without triggering catastrophic forgetting.

Two-Phase Training: Stable Stage vs. Decay Stage

A concrete public blueprint for how data mixtures shift between main pretraining and mid-training comes from the historical **MiniCPM / JetMoE** recipe, which splits pretraining into a **Stable Stage** (constant learning rate) and a **Decay Stage** (learning-rate cooldown / annealing).

Visualizing the data proportions across these two stages as stacked horizontal bars (summing to 100% each) reveals the exact mechanics of annealing:

- **Stable Stage (Main Pretraining):** Almost the entire mixture consists of bulk web and code (~97%), with ~3% academic/math sources (Open Web Math, Arxiv, peS2o) and **0% SFT data**. Specifically:
  - Code Pretrain: **25%**
  - CommonCrawl Chinese (`CC_Chn`): **25%**
  - Dolma (**24%**) + C4 (**15%**): **39%** combined
  - Pile / Wiki / Books / Other: **~11%**
  - SFT / Synthetic / QA: **0%**
- **Decay Stage (Mid-Training / Annealing):** During learning-rate decay, the bulk web and code shares are trimmed back—not eliminated—to make room for curated encyclopedic text and a **~21.5%** injection of SFT and synthetic instruction data:
  - Code Pretrain: **19.6%**
  - CommonCrawl (`CC_Chn`): **14.6%**
  - Dolma (**15.7%**) + C4 (**9.5%**): **25.2%** combined
  - Wiki / Pile / Books (higher-quality curated text): **~19.1%**
  - SFT & Synthetic: **~21.5%** mix, composed of `SFT_mixed` (4.8%), `Code_SFT` (3.8%), `Knowledge_SFT` (2.5%), `UltraChat` (2.0%), `Math_Synthetic` (1.9%), plus EvolInstruct, ShareGPT4, and related instruction corpora.

Notice the crucial design feature in that second bar: nearly 78.5% of the Decay Stage is still pretraining data (bulk web, code, Wiki, Pile, and Books). Keeping bulk pretraining data mixed alongside the newly introduced SFT and synthetic data during learning-rate decay acts as a replay anchor—preventing forgetting while steering the final weights toward target domains.

> **COMMON MISCONCEPTION**
> You might think that once main pretraining finishes, keeping bulk web data in the mid-training mixture wastes compute that would be better spent on 100% curated instruction and reasoning data. That breaks because dropping bulk pretraining data too abruptly causes severe distribution drift and catastrophic forgetting of broad factual and linguistic capabilities; preserving a substantial fraction of general pretraining text during annealing is what lets the model absorb billions of instruction tokens safely.

WORKED EXAMPLE

Allocating a 200-billion-token decay-stage mid-training budget

Setup: Suppose you run a 200B-token Decay Stage using the exact historical MiniCPM / JetMoE mixture proportions above. How many tokens are allocated to each category, and how does this compare to a naive 200B-token pure-SFT run?

Step 1 — Compute retained bulk and curated pretraining tokens:

- Code Pretrain: $0.196 \times 200\text{B} = 39.2\text{B}$ tokens
- CommonCrawl Chinese: $0.146 \times 200\text{B} = 29.2\text{B}$ tokens
- Dolma + C4: $(0.157 + 0.095) \times 200\text{B} = 0.252 \times 200\text{B} = 50.4\text{B}$ tokens
- Wiki / Pile / Books: $(1 - 0.196 - 0.146 - 0.252 - 0.215) \times 200\text{B} = 0.191 \times 200\text{B} = 38.2\text{B}$ tokens
- Subtotal of general/curated pretraining replay: $39.2 + 29.2 + 50.4 + 38.2 = 157.0\text{B}$ tokens (78.5%).

Step 2 — Compute injected SFT and synthetic tokens:

- Total SFT & Synthetic (21.5%): $0.215 \times 200\text{B} = 43.0\text{B}$ tokens.
- Breakdown of named subsets: `SFT_mixed` $= 9.6\text{B}$ (4.8%), `Code_SFT` $= 7.6\text{B}$ (3.8%), `Knowledge_SFT` $= 5.0\text{B}$ (2.5%), `UltraChat` $= 4.0\text{B}$ (2.0%), `Math_Synthetic` $= 3.8\text{B}$ (1.9%), with the remaining 13.0B (6.5%) from EvolInstruct, ShareGPT4, and similar corpora.

Step 3 — Check total and interpret: $157.0\text{B} + 43.0\text{B} = 200.0\text{B}$ tokens. The model absorbs 43B tokens of instruction, code-SFT, and synthetic math—orders of magnitude more than traditional post-training SFT—while 157B tokens of general pretraining replay continuously stabilize the weights during learning-rate decay. Running 200B tokens of pure SFT without that 157B replay anchor would overwrite general pretrained representations.

What Mid-Training Acquires in 2026

In late-2026 frontier models, mid-training has grown far beyond a learning-rate cooldown. It is now a dedicated capability-acquisition stage executed before post-training begins, targeting three core competencies:

1. **Long-context extension:** Scaling the model's usable context window far beyond main pretraining lengths—for example, extending to **1M tokens** in **DeepSeek-V4.1-Flash** and **Kimi K3**.
2. **Code and reasoning density:** Heavily upweighting full software repositories, formal mathematics, and multi-step derivations so deep reasoning patterns become native to the base distribution.
3. **Early agentic competence:** Exposing the base model to long-horizon workflows and interactive tool environments *prior* to SFT.

Why invest so heavily before SFT even starts? Because a stronger mid-trained checkpoint reduces how much SFT and RL must fight the base distribution. When the base distribution already assigns substantial probability mass to coherent 1M-token context use, multi-step derivations, and repository-scale code,

downstream SFT and RL can elicit and refine those behaviors rather than trying to force them from scratch. We will return to this concrete mechanism in Lecture 16, which walks through **Qwen3-Coder-Next** mid-training as a worked case study of long-context and agentic mid-training.

KEY IDEA

- Mid-training (continued pretraining) bridges bulk pretraining and SFT by shifting the data mixture toward curated text, long context, code, reasoning, and instruction data under the next-token prediction objective while retaining enough bulk pretraining replay to prevent catastrophic forgetting.
- In 2026 frontier models, mid-training is a dedicated capability stage that establishes long context (up to 1M tokens), high code/reasoning density, and early agentic competence so downstream SFT and RL do not have to fight the base distribution.

Even after strong mid-training, however, the model is still trained purely with next-token prediction over mixed documents—it does not yet reliably behave as a conversational assistant that follows user instructions, refuses harmful requests, or emits structured tool calls on cue. Shaping that exact interactive interface is the job of Supervised Fine-Tuning.

✎ Check Your Understanding

1. **Quick computation:** If a 100B-token decay-stage mid-training run follows the MiniCPM / JetMoE mixture (21.5% SFT & Synthetic, 78.5% bulk/curated pretraining), how many tokens of general pretraining replay does the model see during annealing, and how many tokens of `Code_SFT` (3.8%)?
   Answer: It sees 78.5B tokens of general/curated pretraining replay and 3.8B tokens of `Code_SFT`.
2. **What would happen if…** you removed all bulk pretraining data (CommonCrawl, Dolma, C4, Code Pretrain) during a large-scale mid-training run and trained exclusively on instruction and math-synthetic data?
   Answer: The sharp distribution shift would cause distribution drift and catastrophic forgetting, degrading the broad factual knowledge and general language capabilities learned during main pretraining.

Section 3: Supervised Fine-Tuning (SFT) — Objective, Data Evolution, and Knowledge Limits

Mid-training gives us a strong, long-context, reasoning-capable base model, yet it still models generic text continuations rather than a clean user–assistant turn interface. **Supervised Fine-Tuning (SFT)** resolves this by casting alignment as **imitation learning** (pure generative modeling): given a dataset $\mathcal{D}_{\text{SFT}}$ of

prompt–response pairs (x, y) produced by a reference policy $p^*(y \mid x)$—either human writers or a stronger teacher model—we train our model $\hat{p}_\theta(y \mid x)$ to imitate $p^*(y \mid x)$.

> **DEFINITION**
> **Supervised Fine-Tuning (SFT)** is an imitation-learning stage that trains a language model $\hat{p}_\theta(y \mid x)$ via maximum likelihood (next-token cross-entropy minimization) to match demonstration responses y drawn from a reference policy $p^*(y \mid x)$ given prompts x.

The SFT Objective

Formally, SFT seeks to fit $\hat{p}_\theta(y \mid x) \approx p^*(y \mid x)$ by solving:

$$\min_\theta \; -\mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{SFT}}}[\log \hat{p}_\theta(y \mid x)]$$

Let us break this objective apart piece by piece:

- θ denotes the trainable parameters of the language model.
- $(x, y) \sim \mathcal{D}_{\text{SFT}}$ denotes sampling a prompt x and its demonstration completion $y = (y_1, y_2, \ldots, y_T)$ from the curated SFT dataset.
- $\hat{p}_\theta(y \mid x)$ is the autoregressive conditional probability the model assigns to the entire response sequence y given the prompt x. By the chain rule of probability, this factorizes token by token:

  $$\hat{p}_\theta(y \mid x) = \prod_{t=1}^{T} \hat{p}_\theta(y_t \mid x, y_{<t})$$

  where $y_{<t} = (y_1, \ldots, y_{t-1})$ are the completion tokens generated before step t.
- Taking the logarithm converts the product into a sum of per-token log-probabilities:

  $$-\log \hat{p}_\theta(y \mid x) = -\sum_{t=1}^{T} \log \hat{p}_\theta(y_t \mid x, y_{<t})$$

  Each term $-\log \hat{p}_\theta(y_t \mid x, y_{<t})$ is the standard next-token cross-entropy loss at position t. Notice that the sum runs **only over response tokens** y_t, not over prompt tokens x—a property enforced in code by loss masking, which we examine in Section 4.
- Taking the expectation $\mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{SFT}}}[\cdot]$ averages this negative log-likelihood across the SFT dataset, and $\min_\theta$ adjusts the weights via gradient descent to push the predicted probability of every demonstration token as close to 1 as possible.

Taken as a single conceptual unit, the SFT objective minimizes the cross-entropy (equivalently, the forward KL divergence up to a constant entropy term) between the demonstration distribution $p^*(y \mid x)$ and the model $\hat{p}_\theta(y \mid x)$.

> **WORKED EXAMPLE**

Evaluating the SFT cross-entropy loss on a known vs. unfamiliar demonstration

Setup: Consider a prompt $x = \text{"What is 6 times 7?"}$ paired with a 3-token demonstration $y = (\text{"42"}, \text{"."}, \text{"<eos>"})$. Suppose our base model already knows basic arithmetic and assigns conditional probabilities $\hat{p}_\theta(\text{"42"} \mid x) = 0.50$, $\hat{p}_\theta(\text{"."} \mid x, \text{"42"}) = 0.80$, and $\hat{p}_\theta(\text{"<eos>"} \mid x, \text{"42"}, \text{"."}) = 0.90$.

Step 1 — Compute per-token negative log-likelihoods (in nats, natural log):

- Token 1 ("42"): $-\ln(0.50) = 0.6931$
- Token 2 ("."): $-\ln(0.80) = 0.2231$
- Token 3 (""): $-\ln(0.90) = 0.1054$

Step 2 — Sum over the response and compute per-token average:

- Sequence loss: $-\log \hat{p}_\theta(y \mid x) = 0.6931 + 0.2231 + 0.1054 = 1.0216$ nats.
- Mean per-token cross-entropy: $1.0216/3 = 0.3405$ nats/token.

Step 3 — Compare against a demonstration containing an unknown fact: Now suppose a second prompt asks for an obscure journal citation absent from the model's pretraining knowledge, so the model assigns $\hat{p}_\theta(y_1 \mid x) = 0.001$ to the author's surname. That single token alone generates a loss of $-\ln(0.001) = 6.9078$ nats—more than $6.7\times$ the loss of the entire arithmetic example.

Interpretation: SFT gently reinforces formatting and answers the model already half-knows, but produces massive loss spikes and aggressive gradient updates whenever a demonstration forces the model to emit specific facts it does not internally know. We will see shortly how this exact mechanism induces hallucination.

Two ingredients control SFT quality:

- **The data:** the prompt distribution x, the demonstration policy $p^*(y \mid x)$, and the dataset's style, knowledge coverage, safety, and tool-use traces.
- **The method:** standard gradient descent on next-token cross-entropy, combined with sequence packing and loss masking.

Evolution of Open SFT Data (2022–2026) and Four Archetypes

Because the optimization algorithm of SFT is simply cross-entropy gradient descent, the history of SFT progress is largely the history of **SFT data design**. Tracing the open-source progression from 2022 to 2026 forms a clear two-row developmental chain:

- **Row 1 (2022–2023 origins):**
  1. **FLAN / SuperNI** (academic NLP benchmarks wrapped in instruction templates; for instance, Natural Instructions v2 aggregated 372 datasets and 1,554 tasks) $\rightarrow$
  2. **Self-Instruct / Alpaca** (synthetic single-turn prompts and distilled outputs generated from a stronger model) $\rightarrow$
  3. **ShareGPT / Vicuna** (real user prompts paired with multi-turn chat logs) $\rightarrow$

4. **OpenAssistant** (crowdsourced conversation trees with detailed human-written responses).

- **Row 2 (2023–2026 maturation):** 5. **WizardLM (Evol-Instruct)** (automated in-depth and in-breadth prompt evolution plus filtering to raise complexity) → 6. **Tülu 3 (2024)** (curated multi-skill mixture covering math, code, safety, and persona-driven prompts) → 7. **Nemotron & 2026 SFT** (multi-turn tool-call and agent trajectories packed into very long contexts).

The overarching trend across these seven milestones is unmistakable: **terse NLP labels → helpful chat prose → long-horizon agentic tool trajectories.**

Comparing four representative archetypes illustrates how dramatically SFT datasets vary along four axes —style/length ("chattiness"), factual detail, scale, and safety/tool-use structure:

Dataset	Archetype	Representative Prompt	Typical Completion & Character
FLAN	NLP templates	"Ahold to Sell Spain Operations… What is this text about? OPTIONS: World, Sports, Business, Science/Tech"	`Business` — Valid supervision, but users do not want to chat with an NLP benchmark.
Alpaca	Synthetic 1-turn	"Give three tips for staying healthy."	"1. Eat a balanced diet… 2. Exercise regularly… 3. Get enough sleep…" — Short, clean numbered lists.
OpenAssistant	Human chat	"Write a short intro on 'monopsony' in labor markets and cite relevant research."	Detailed essay plus formal citation (*Bivens & Mishel (2013), J. Econ. Perspectives*) — High factual detail, which is both a pro and a con.
Nemotron-SFT-OpenCode	Agent / tool use	"Async/await vs .then() in JS, or integers vs floats for a calculator app?"	`{"role":"assistant","content":"…","tool_calls":[{"name":"todowrite","arguments":"…"}]}` — Structured multi-step actions.

Style, Length, and How SFT Mixtures Affect Benchmarks vs. Preferences

Quantitative statistics measured by **Wang et al. (2023)** across historical SFT corpora (with CoT and Flan V2 sampled to 100K examples each) show just how wide the structural gap between datasets is:

Dataset	Source	Instances	Avg. Rounds ($\bar{N}_{\text{rounds}}$)	Avg. Prompt Length ($\bar{L}_{\text{prompt}}$)	Avg. Completion Length ($\bar{L}_{\text{completion}}$)
SuperNI	NLP + human instructions	96,913	1.0	291.1	38.7
Flan V2	NLP + human instructions	100,000	1.0	355.7	**31.2**
CoT	NLP + human CoTs	100,000	1.0	266.0	53.2
Alpaca	Generated w/ Davinci-003	52,002	1.0	27.8	64.6
GPT4-Alpaca	Davinci-003 + GPT-4	52,002	1.0	28.0	161.8
OpenAssistant 1	Human-written from scratch	34,795	1.6	34.8	212.5
ShareGPT	User prompts + model outputs	168,864	**3.2**	71.0	**357.8**

Two structural facts stand out from this table:

1. Average completion length spans **more than** $10\times$, from just 31.2 tokens in Flan V2 up to 357.8 tokens in ShareGPT.
2. Multi-turn conversational structure ($\bar{N}_{\text{rounds}} > 1$) appears *only* in chat-sourced datasets (OpenAssistant 1 at 1.6, ShareGPT at 3.2) and later agentic corpora—not in NLP-derived or single-turn synthetic datasets.

Those stylistic differences create a sharp split between **preference evaluations** and **objective task benchmarks**. As **Dubois et al. (2023)** demonstrated, preference evaluators—both humans and GPT-4 judges—are highly style-sensitive: they prefer outputs formatted as **lists** in roughly **58–65%** of pairwise comparisons, and show a strong bias toward **longer outputs** ($\sim$**69%** for human evaluators and $\sim$**74%** for GPT-4). We will see this exact length bias return as a major failure mode when we study RLHF reward hacking in Section 6.

By contrast, objective capability benchmarks respond to the *skill mix* in the SFT data rather than chattiness. Training a historical **Vanilla LLaMA 13B** base model on different SFT subsets (Wang et al. 2023) yields the following breakdown:

SFT Data	MMLU	GSM (Math)	Codex (Code)	AlpacaEval (Pref.)	Average
Vanilla 13B	42.3	14.5	28.6	–	–
+Flan V2	**50.6**	20.0	*16.8* (regressed)	3.2	29.8
+CoT	44.2	40.0	*23.7* (regressed)	6.0	33.9
+OASST 1	43.3	15.0	31.9	58.1	36.9
+ShareGPT	49.3	27.0	34.1	**70.5**	42.0
+Human+GPT (Mixture)	49.3	**40.5**	35.9	56.5	**45.2**

Notice three critical lessons in this table:

- **Flan V2** boosts MMLU by $+8.3$ points ($42.3 \rightarrow 50.6$) because its multiple-choice templates match MMLU, yet it collapses to 3.2% on AlpacaEval (terse 31-token answers lose badly in chat preference evals) and *actively hurts* coding, dragging Codex down from 28.6 to 16.8.
- **CoT** drives a massive $+25.5$-point jump on GSM math ($14.5 \rightarrow 40.0$), yet also regresses Codex ($28.6 \rightarrow 23.7$) and scores only 6.0% on AlpacaEval.
- **ShareGPT** dominates AlpacaEval at 70.5% thanks to its long, multi-turn chatty responses, while the balanced **Human+GPT mixture** wins overall (45.2 average) by combining MMLU (49.3), GSM (40.5), Codex (35.9), and solid chat quality (56.5) without collapsing any single capability.

Single-source SFT routinely hurts orthogonal skills; well-curated mixtures win overall.

Knowledge Extraction and Hallucination in SFT

A deeper puzzle arises from the OpenAssistant archetype we saw earlier: why is high factual detail (such as an essay citing *Bivens & Mishel, 2013*) simultaneously a pro and a con?

The underlying mechanism was articulated by **Schulman (2023)**. A pretrained base language model internally stores a noisy knowledge graph with an implicit confidence attached to each factual edge. Suppose an SFT demonstration contains a **factually correct** answer—or a real academic citation—that happens to be *absent* from the base model's pretrained knowledge graph. When behavior cloning minimizes $-\log \hat{p}_\theta(y \mid x)$ on that demonstration, the model cannot magically acquire a verified external lookup mechanism from a single example. Instead, it learns a destructive meta-behavior: **when your internal confidence on a fact is low, do not hedge or abstain—guess confidently in the style of a real citation or authoritative answer.**

Gekhman et al. (2023) confirmed this empirically by separating SFT training examples into facts already *known* to the base model versus facts *unknown* to the base model, and tracking accuracy across 0 to 50 training epochs on two vertically aligned plots:

- **Top plot (Train Accuracy % from 0 to 100 vs. Epochs 0 to 50):** Two curves rise at drastically different rates. The **Train Known** curve shoots up almost immediately, climbing from near 0% to roughly 80–85% within the first 10 epochs and approaching ~98% by epoch 50. The **Train Unknown**

curve remains nearly flat near 5–15% through epoch 10, and only begins rising steeply *after* epoch 10 as the model slowly memorizes the unfamiliar facts, eventually reaching ~95% by epoch 50.

- **Bottom plot (Dev Accuracy % from 39 to 44 vs. Epochs 0 to 50):** On the held-out development set, **Dev Accuracy** climbs sharply from ~39.3% at epoch 0 to its peak of ~43.5% right at **epoch 10**. A vertical red dashed line drawn across both plots at **epoch 10** marks the exact point where **overfitting starts**: the precise epoch at which the model begins memorizing *Train Unknown* examples is the exact epoch at which *Dev Accuracy* peaks and then falls steadily back down toward ~39.5% by epoch 50.

This exposes an inherent trap in distillation: whenever you collect SFT demonstrations from a stronger teacher model that knows more tail facts than your student base model, you automatically inject unknown facts into $\mathcal{D}_{\text{SFT}}$, teaching the student to hallucinate with confidence.

COMMON MISCONCEPTION

You might think that adding more factually verified, high-detail SFT examples written by domain experts or a stronger frontier teacher will always make your fine-tuned model more truthful. That breaks because SFT extracts pretrained behaviors rather than reliably inserting new knowledge; when a demonstration contains facts absent from the student's base knowledge graph, cross-entropy imitation forces the student to emit confident completions from low-confidence internal states, actively increasing hallucination on held-out prompts.

Four core takeaways govern SFT data quality:

1. **SFT extracts pretrained behaviors rather than adding new knowledge.** Reserve new factual knowledge acquisition for pretraining and mid-training, not small-scale SFT.
2. **Even factually correct SFT data can hurt.** Whenever the target policy p^* knows facts that the student base model does not, behavior cloning rewards confident guessing.
3. **Small amounts of the right behavioral signal go a long way—with a long tail.** Style, formatting, and safety guidelines shift rapidly with modest data, though complex reasoning, coding, and agentic environments still benefit from scale and diversity.
4. **Correctness-based RL feedback** (which we introduce in Section 5 and expand in Lecture 16) is required to align generation with what the model actually knows or can verify, and to surpass the ceiling of demonstration writers.

KEY IDEA

- SFT minimizes next-token cross-entropy $-\mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{SFT}}}[\log \hat{p}_\theta(y \mid x)]$ to imitate a reference policy; because preference evaluators heavily favor length and list formatting while capability benchmarks track skill coverage, balanced multi-source mixtures outperform single-source SFT.
- SFT primarily elicits pretrained knowledge rather than teaching new facts: training on factually correct demonstrations that lie outside the base model's knowledge graph teaches confident guessing and triggers hallucination right when dev accuracy begins to fall.

Two critical domains of SFT demand specialized data pipelines and careful implementation mechanics beyond general chat prose: **safety alignment** (where small datasets must balance preventing harm against refusing benign requests) and **2026 agentic tool-use trajectories** (where long multi-turn logs can silently break naive cross-entropy training). We examine both next.

✎ Check Your Understanding

1. **Yes or No:** Can adding 100% factually accurate demonstrations from a stronger teacher model to an SFT dataset increase the student model's hallucination rate? Explain why in one to two sentences.
   Answer: Yes. When a teacher demonstration includes facts absent from the student's pretrained knowledge graph, SFT cross-entropy forces the student to imitate confident answers despite internal uncertainty, teaching it to guess confidently rather than hedge.
2. **Fill in the blank:** In the Gekhman et al. (2023) experiment, held-out development accuracy peaks and overfitting begins at the exact epoch when the model starts memorizing _____ facts in the SFT training set.
   Answer: unknown (facts not already known by the pretrained base model).

Section 4: Safety SFT and Agentic SFT Implementation

Once an instruction-tuned model follows arbitrary user commands reliably, two operational problems immediately emerge. First, an indiscriminately compliant model will assist with malicious misuse—automated disinformation, impersonation, spear-phishing, and scams (Goldstein et al. 2023; Kang et al. 2023). Second, in 2026 engineering applications, assistants do not merely emit single-turn prose: they execute multi-turn loops inside shells, codebases, browsers, and APIs, where raw environment outputs can be orders of magnitude longer than the model's own actions. Both problems require specialized SFT design.

Safety SFT: Balancing Harm and False Refusals

A naive safety fix—training on thousands of blunt refusals to any prompt containing sensitive keywords—creates a crippled assistant that refuses benign requests about history, cybersecurity defense, or medical terminology. Safety SFT is therefore a two-objective trade-off between minimizing **Violation Rate (VR)** (complying with genuinely harmful prompts) and minimizing **False Refusal Rate (FRR)** (refusing safe prompts).

DEFINITION

Violation Rate (VR) is the fraction of genuinely harmful or adversarial prompts to which a model produces unsafe compliance, while **False Refusal Rate (FRR)** is the fraction of benign prompts—especially sensitive or borderline ones—that the model erroneously refuses to answer.

Three empirical principles govern effective safety SFT:

1. **Quality beats quantity:** Raw human vendor annotations are often inconsistent on nuanced safety policies, so modern pipelines use AI-assisted quality assurance to enforce consistency.
2. **Include borderline prompts and diverse attacks:** Explicitly mixing in **borderline prompts** (benign requests phrased like adversarial ones) is essential to push down FRR. To cover diverse failure modes on the VR side, pipelines synthesize attacks through guided mutation and **Rainbow Teaming** (using MAP-Elites quality-diversity search to explore a grid of attack styles and risk categories).
3. **Model size matters:** Historical findings from **Llama 3 (July 2024)** showed that an 8B model requires a *higher proportion* of safety data than a 70B model to achieve the same Violation Rate. Larger models separate genuinely adversarial intent from benign borderline phrasing much more cleanly, yielding a strictly better VR–FRR Pareto frontier.

Historically, **Llama 2 (2023)** achieved its baseline safety alignment with roughly a few thousand safety SFT examples. Sample-efficiency curves explain why safety behavior shifts so rapidly. Consider the bar chart of **Mean Harm Score** (vertical axis from 0 to 3) versus **Added Alpaca-Style Safety Samples** (horizontal axis at $0, 100, 300, 500, 1000, 2000$ examples), reflecting a consistent trend across four benchmarks (`I-MaliciousInstructions` , `I-CoNa` hate speech, `I-Controversial` , and `Q-Harm`):

- At **0** safety samples, the mean harm score sits near **3.0**.
- At **100** samples, it drops sharply to roughly **2.05**.
- At **300** samples, it falls nearly in half again to roughly **1.2**.
- By **500** samples, the harm score drops to roughly **0.75**—capturing the vast majority of the initial safety gain.
- Past **1,000** samples (~0.5) and **2,000** samples (~0.3), the curve flattens into strong **diminishing returns**.

To handle real-world adversarial diversity beyond synthetic templates, open safety pipelines anchor on authentic user interactions:

- **WildChat:** 1 million in-the-wild ChatGPT interaction logs capturing realistic prompts.
- **Mine & Compose (WildTeaming):** mining real jailbreak tactics observed in the wild and composing them with vanilla harmful queries to synthesize challenging training attacks.
- **Non-compliance taxonomy:** separating distinct reasons a model should refuse or abstain—incomplete prompts, unsupported capabilities, indeterminate requests, and genuine safety-risk prompts.
- **Tülu 3 (2024) open safety mix:** combines three targeted datasets— `CoCoNot` (10,983 examples for contextual non-compliance), `WildJailbreak` (26,356 examples), and `WildGuardMix` (26,356 examples).

SFT in 2026: Agent & Tool-Use Trajectories and Loss Masking

In 2026, SFT datasets are no longer predominantly single-turn Q&A pairs. They consist largely of **multi-turn tool-call trajectories**. Rather than asking humans to hand-write long coding sessions, frontier pipelines run strong teacher models inside real execution environments—shells, codebases, browsers, and

APIs—recording complete action–observation rollouts and filtering them by task difficulty and automated verification (such as passing test suites).

These rollouts are dramatically longer than historical chat data: **Nemotron 3.5 Super (Oct 2026)** packs SFT trajectories up to **524,288 tokens** per training sequence. An individual assistant turn from `Nemotron-SFT-OpenCode-v1` combines natural-language reasoning with structured JSON tool invocations:

```
{"role": "assistant",
 "content": "I'll analyze the trade-offs based on AGENTS.md.",
 "tool_calls": [{"id": "call_b06182c2", "type": "function",
   "function": {"name": "todowrite",
     "arguments": "{\"todos\":[{\"id\":\"1\",\"status\":\"in_progress\",
       \"content\":\"Review AGENTS.md requirements\"}]}"}}]}
```

Across a full session, the conversation alternates between user messages, assistant reasoning plus JSON `tool_calls`, raw environment/tool outputs (terminal logs, file dumps, HTML pages, compiler errors), and subsequent assistant turns culminating in a final answer.

This interleaved structure makes **loss masking** non-negotiable. Consider a horizontal sequence layout of a single multi-turn agent trajectory inside a 524,288-token packed batch:

1. **User Prompt** → `mask = 0` (ignore in loss).
2. **Assistant +** `tool_calls` → `mask = 1` (compute cross-entropy loss).
3. **Tool / Environment Output (logs, HTML)** → `mask = 0` (**NEVER** train on this).
4. **Assistant Final Turn** → `mask = 1` (compute cross-entropy loss).

When multiple independent trajectories are concatenated ("packed") end-to-end into one 524,288-token buffer for training efficiency, a **block-diagonal attention mask** isolates each trajectory so tokens in Trajectory 2 cannot attend to tokens in Trajectory 1.

> **DEFINITION**
> **Loss masking** in SFT sets the target label to an ignored sentinel index (such as `-100` in PyTorch) on all non-assistant tokens—specifically user prompts and tool/environment outputs—so cross-entropy loss and gradients are computed exclusively over tokens the assistant itself is responsible for generating.

In implementation, the training loop is only five lines of code, yet two indexing details do all the heavy lifting:

```
for batch in train_dataloader:                    # labels = -100 on user & tool-output
        tokens
    logits = model(input_ids=batch["input_ids"],
        attention_mask=batch["attn_mask"]).logits
    loss = F.cross_entropy(logits[:, :-1].reshape(-1, V),
                           batch["labels"][:, 1:].reshape(-1), ignore_index=-100)
    loss.backward(); optimizer.step(); lr_scheduler.step(); optimizer.zero_grad()
```

Observe how every operation works:

- `batch["attn_mask"]` enforces causal masking *and* block-diagonal isolation between packed trajectories, allowing the assistant to read prior user prompts and tool outputs as context.
- `logits[:, :-1]` aligns the model's prediction at position t with the next token's ground-truth label `batch["labels"][:, 1:]` at position $t+1$, reshaped across the vocabulary dimension V.
- `ignore_index=-100` zeroes out the loss contribution of every user token and every tool-output token.

Why must we mask tool outputs and user turns? Tool outputs—stack traces, `grep` results, full source files, raw HTML—are routinely $10\times$ **to** $100\times$ **longer** than the assistant's tool calls. Training next-token loss on those tokens wastes model capacity predicting arbitrary external logs and actively teaches the assistant to **hallucinate fake environment responses** instead of waiting for real tool execution.

COMMON MISCONCEPTION

You might think that leaving loss unmasked on user prompts and tool outputs helps the model "understand" the environment logs and user requests better during SFT. That breaks for two reasons: the model already reads every user and tool token through self-attention during the forward pass (so masking the *loss* never hides context from the model), whereas computing cross-entropy on 10×–100× longer tool outputs drowns out the assistant's decision gradients and trains the model to fabricate tool outputs itself.

WORKED EXAMPLE

Quantifying gradient dilution with vs. without loss masking

Setup: Consider a single agentic debugging trajectory totaling $5{,}000$ tokens:

- User prompt: 100 tokens (`mask = 0`)
- Assistant turn 1 (reasoning + JSON tool call): 80 tokens (`mask = 1`)
- Tool / environment output (compiler and unit-test logs): $4{,}700$ tokens (`mask = 0`)
- Assistant final turn (bug fix + summary): 120 tokens (`mask = 1`)

Assume for comparison that every token has a similar average cross-entropy loss of 1.0 nat.

Case 1 — Without loss masking (naive next-token loss on all tokens):

- Total tokens contributing to loss: $100 + 80 + 4{,}700 + 120 = 5{,}000$.
- Fraction of loss coming from tool logs: $4{,}700/5{,}000 = 94.0\%$.
- Fraction of loss coming from the user prompt: $100/5{,}000 = 2.0\%$.

- Fraction of loss coming from actual assistant behavior: $(80 + 120)/5{,}000 = 200/5{,}000 = 4.0\%$.

Case 2 — With loss masking (`labels = -100` on user and tool tokens):

- Active loss tokens: $80 + 120 = 200$.
- Fraction of gradient coming from assistant reasoning, tool calls, and final response: $200/200 = 100.0\%$.
- Context still visible via causal self-attention: all $5{,}000$ tokens (the assistant final turn still attends to the 100-token prompt and all $4{,}700$ tokens of test logs).

Interpretation: Omitting loss masking redirects 96% of the optimization signal toward memorizing user phrasing and predicting compiler logs, starving the actual agent policy of gradient signal while teaching it to hallucinate tool outputs.

KEY IDEA

- Safety SFT is sample-efficient (roughly 500 targeted examples produce most of the initial harm reduction, with diminishing returns past 1,000), but requires borderline benign prompts and realistic adversarial data to balance Violation Rate against False Refusal Rate.
- Modern 2026 SFT trains on packed, long-context multi-turn tool-use trajectories (up to 524,288 tokens in Nemotron 3.5 Super) using block-diagonal attention masks and strict loss masking (`ignore_index=-100`) on user prompts and tool outputs.

Even with perfectly masked agentic SFT and clean safety mixtures, imitation learning hits a fundamental ceiling: $\hat{p}_\theta(y \mid x)$ can at best match the quality of the demonstration policy $p^*(y \mid x)$. It cannot systematically surpass its human or teacher writers, nor can it learn directly from verifying whether its own outputs succeed. Breaking past that imitation ceiling requires moving from generative imitation to reward optimization.

✎ Check Your Understanding

1. **Quick computation:** An agent trajectory has a 50-token user prompt, a 150-token assistant tool call, a 3,600-token HTML page returned by a browser tool, and a 200-token final assistant answer. Under proper SFT loss masking, on how many tokens is cross-entropy loss computed, and what percentage of the total sequence is masked out?
   Answer: Loss is computed on $150 + 200 = 350$ assistant tokens; the remaining $50 + 3{,}600 = 3{,}650$ tokens out of $4{,}000$ total (91.25%) are masked out with `ignore_index=-100`.
2. **What would happen if…** you packed two independent 100,000-token agent trajectories end-to-end into a single 200,000-token training sequence without using a block-diagonal attention mask?
   Answer: Tokens in the second trajectory would attend to unrelated prompts, tool calls, and logs from the first trajectory, corrupting the context distribution and teaching spurious cross-document dependencies.

Section 5: From Imitation to Optimization — Preference Data, Annotators, and AI Feedback

Supervised Fine-Tuning treats a language model as a density estimator fitting $\hat{p}_\theta(y \mid x) \approx p^*(y \mid x)$. That framing carries a hard ceiling: the trained model cannot reliably outperform the demonstration writers who generated y. **Optimization**—via Reinforcement Learning from Human Feedback (**RLHF**) or Reinforcement Learning with Verifiable Rewards (**RLVR**)—recasts the language model as a **policy** $p(y \mid x)$ that directly maximizes expected reward:

$$\max_p \ \mathbb{E}_{y \sim p(\cdot|x)}[R(y, x)]$$

Let us inspect the two formulations side by side:

- **Imitation (SFT):** Requires demonstrations sampled from a reference policy p^*; views the LM as a density estimator; quality is capped by the demonstration writers.
- **Optimization (RLHF / RLVR):** Views the LM as a decision-making policy; requires only a measurable scalar reward $R(y, x)$ (or a pairwise preference / automated verifier); can surpass the people who wrote the training prompts or demonstrations.

Why is optimization strictly more powerful than imitation? Because of the **Generation–Verification (G–V) gap**: verifying, grading, or ranking candidate outputs is often far easier than writing an ideal output from scratch.

DEFINITION

The **Generation–Verification (G–V) gap** is the asymmetry between producing an optimal solution from scratch and checking or comparing candidate solutions—enabling reward-based optimization to discover behaviors superior to human demonstrations whenever verification is easier than generation.

Two concrete domains expose the G–V gap sharply:

- **News summarization (Zhang et al. 2023):** Evaluators preferred outputs from Instruct Davinci over freelance human writers almost half the time (**49.6% vs. 50.4%**), with overall inter-rater agreement $\alpha = 0.07$—demonstrating that humans do not reliably *write* the summaries they themselves prefer when judging side by side.
- **Mathematical reasoning (e.g., DeepSeekMath-V2):** Self-verifiable mathematical proofs allow optimization to explore and verify solution paths that go well beyond human-written SFT solutions.

Collecting Pairwise Preference Data

For open-ended tasks where no automated unit test or mathematical verifier exists, how do we obtain the reward signal $R(y, x)$? The standard RLHF interface displays a prompt x alongside two (or K) candidate responses and asks a human annotator which response is better—often recording preference strength (such as "slightly better" versus "much better").

Because subjective quality judgments easily diverge across raters, annotation platforms write detailed rubrics to enforce consistency:

- **InstructGPT (Ouyang et al. 2022):** Annotators rate responses along three axes—**Helpful** (clear, answers the user's intended question, avoids rambling), **Truthful** (accurate, no hallucinated context), and **Harmless** (no abuse, bias, or real-world harm). The rubric specifies an explicit trade-off rule: **truthfulness and harmlessness outweigh helpfulness**, especially in high-stakes domains such as medical advice, legal questions, and financial loans.
- **Historical Bard guidelines:** Separated **Helpfulness** (on a 5-point scale) from **Presentation** (on a 3-point scale covering structure, neutral tone, and conciseness) to discourage raters from conflating polished formatting with substantive correctness.

Who Are the Annotators? Labor, Pitfalls, and Demographic Shift

Even with detailed rubrics, pairwise preference annotation is noisy, expensive, and deeply shaped by who performs the labeling. Modern preference pipelines sit across a wide labor spectrum spanning two orders of magnitude in compensation:

- **Platform workforce (Outlier / ScaleAI):** An Oxford Economics survey ($n \approx 911$–914, covering one platform) profiled the crowd/gig annotator pool:
  - *Education:* 44% Bachelor's, 32% Master's/professional, 9% Doctoral (PhD), 8% Associate's, 7% High school.
  - *Age:* 34% ages 35–44, 25% ages 25–34, 23% ages 45–54.

    - *Top task domains:* Language (21%), Creative Writing (18%), Generalist (13%), Biology (6%), Tech Writing (5%), Math (5%), Coding (4%).

- **Growth of paid expert annotation:** As frontier tasks outgrew generic chat filtering, labs shifted toward domain specialists. For example, **Handshake AI ("Project Stagecraft", Dec 2025)** employed roughly **3,000–4,000** freelancers paid at least **$50/hr** to construct realistic occupational tasks. Across the US market, specialist midpoint pay runs roughly **$35–$60/hr** for Expert Trainers and up to **$115–$120/hr** for Highly Skilled Experts in Medical, Engineering, and Legal domains.

Relying on crowdsourced labor introduces three serious operational, ethical, and statistical challenges:

1. **Operational pitfalls:** Under piece-rate incentives (paid per comparison completed), it is difficult to verify annotator domain expertise or enforce time-consuming fact-checking. Worse, **AI-assisted cheating**—workers secretly prompting external LMs to write or grade responses—silently corrupts the preference signal.
2. **Labor ethics at scale:** Early toxicity-labeling pipelines relied on contractors in Kenya paid less than **$2/hr** (documented in a TIME investigation), alongside broader concerns over precarious contractor conditions when screening traumatic content.
3. **Demographic shift (Santurkar et al. 2023):** The historical InstructGPT labeler pool had a specific demographic composition (52.6% Southeast Asian, 31.6% White; 22% Filipino, 22% Bangladeshi, 17% American; 52.6% undergraduate, 36.8% Master's). Comparing base `davinci` against RLHF-tuned `text-davinci-003` revealed measurable shifts in which demographic groups' opinions the model aligned with.

> **COMMON MISCONCEPTION**
> You might think that RLHF aligns a language model to universal, objective "human values" in the abstract. That breaks because a preference dataset reflects the specific demographic composition, economic incentives, rubric trade-offs, and cognitive blind spots of the particular annotator pool that labeled it—producing measurable shifts in opinion alignment and error tolerance.

Annotator Blind Spots: Assertiveness Hides Errors

Even honest, well-intentioned crowdworkers suffer from systematic perceptual blind spots. **Hosking, Blunsom, and Bartolo (2024)** measured the **Crowd vs. Expert Error Rate Gap**—how much *less* (negative percentages) or *more* (positive percentages) crowdworkers report errors compared to domain experts—as a function of the model's tone across three assertiveness levels: low assertiveness (`Assert --`), `Baseline`, and high assertiveness (`Assert ++`).

Reconstructing their 4×3 heatmap table makes the vulnerability unmistakable:

Error Type	`Assert --` (Hedged)	`Baseline`	`Assert ++` (Confident)
Factuality	-5.3%	-16.2%	-22.3%
Inconsistency	-7.9%	-10.6%	-16.9%
Repetition	-1.6%	-1.8%	-9.3%
Formatting	$+3.7\%$	$+3.1\%$	$+2.1\%$

Read each row carefully:

- On **Factuality**, crowdworkers under-report errors by -5.3% when the response is hedged (`Assert --`), by -16.2% at baseline, and by -22.3% when the response is assertive (`Assert ++`). Speaking in a confident tone masks more than $4\times$ **as many factual errors** from crowd raters.
- On **Inconsistency** (internal logical contradictions), the blind spot more than doubles from -7.9% to -16.9% under high assertiveness; even **Repetition** becomes harder to notice ($-1.6\% \to -9.3\%$).
- On **Formatting**, the numbers are slightly *positive* ($+2.1\%$ to $+3.7\%$): formatting flaws are visually effortless to spot, so crowdworkers actually flag them more often than experts do.

The engineering consequence is severe: naive RLHF trained on crowd preferences actively rewards **confident-sounding, well-formatted hallucinations** over careful, hedged answers.

> **WORKED EXAMPLE**
>
> **Quantifying how the assertiveness blind spot corrupts a preference dataset**
>
> *Setup:* Suppose an annotation batch contains $1{,}000$ model outputs that experts judge to have a 30.0% factual error rate (300 factually flawed responses). Compare how many of those factual errors crowdworkers detect under `Assert --` (hedged) versus `Assert ++` (confident) style, using the Hosking et al. (2024) error-rate gaps.
>
> *Step 1 — Hedged responses (`Assert --`, gap $= -5.3\%$):*
>
> - Crowdworker detection rate: $30.0\% - 5.3\% = 24.7\%$.
> - Factual errors caught: $0.247 \times 1{,}000 = 247$.
> - Factual errors missed: $300 - 247 = 53$ (17.7% of all factual errors slip through).
>
> *Step 2 — Confident responses (`Assert ++`, gap $= -22.3\%$):*
>
> - Crowdworker detection rate: $30.0\% - 22.3\% = 7.7\%$.
> - Factual errors caught: $0.077 \times 1{,}000 = 77$.
> - Factual errors missed: $300 - 77 = 223$ (74.3% of all factual errors slip through).
>
> *Interpretation:* Merely changing a wrong answer from cautious phrasing to assertive prose causes nearly three-quarters of its factual errors to go undetected by crowdworkers. Meanwhile, formatting flaws are caught at $+2.1\%$ to $+3.7\%$ above expert baseline. An optimizer maximizing crowd preference therefore receives a strong gradient pushing toward authoritative tone and clean formatting rather than factual accuracy.

Scaling Preference Data with AI Feedback (RLAIF) and Constitutional AI

Because high-quality human preference collection is slow, expensive, and noisy, researchers asked whether a strong language model could serve as the preference judge—an approach called **Reinforcement Learning from AI Feedback (RLAIF)**.

> **DEFINITION**
> **Reinforcement Learning from AI Feedback (RLAIF)** replaces or supplements human preference labelers with a strong language model judge that evaluates and ranks candidate responses—often guided by explicit rubrics or principles—to produce scalable preference data at a fraction of human annotation cost.

The historical breakthrough came from **Dubois et al. (2023) / AlpacaFarm**, which showed that using GPT-4 as a pairwise preference judge achieved:

- A **Spearman rank correlation of 0.98** ($R^2 = 0.87$) with human win-rates at the system level.
- **Human-mode agreement of ~0.65**, virtually matching human-to-human inter-annotator agreement (~0.66), at roughly $10\times$ **lower cost** (~$10–$20 versus >$100 per 1,000 examples).

Open-weight RLAIF pipelines rapidly adopted AI judges—including UltraFeedback, Zephyr 7B, OLMo, and **Tülu 3 (2024)**. In the Tülu 3 pipeline, for each prompt, the system sampled 4 candidate responses from a diverse **22-model pool** (mixing off-policy models with on-policy Tülu 3 SFT 8B and 70B models), scored every response along four axes (Helpfulness, Instruction Following, Truthfulness, and Honesty) using GPT-4o, and binarized the highest- and lowest-scoring responses into chosen/rejected pairs.

A principled variation of RLAIF is **Constitutional AI (Bai et al., Anthropic)**, which replaces thousands of human red-team preference labels with a small set of explicit written principles (a "constitution"). Its pipeline runs in two connected stages:

- **Stage 1 — Supervised Learning (SL-CAI):**
  1. Start from a **Helpful-Only RLHF Model** (which follows prompts without safety refusals).
  2. Feed it **Red-Teaming Prompts** designed to elicit harmful drafts.
  3. Pass each draft through a **Constitution Loop**: the model generates an initial *Response*, critiques its own response against a randomly drawn constitutional principle (*Critique*), and rewrites it to remove harm (*Revised Response*).
  4. Fine-tune the model on these self-revised responses to produce the **SL-CAI Model**.
- **Stage 2 — Reinforcement Learning from AI Feedback (RL-CAI):**
  1. Use the SL-CAI model to **Sample Pairs** of candidate responses on red-teaming prompts.
  2. Pass each pair to an **AI Constitution Judge**, which picks the response that best adheres to the constitutional principles.
  3. Train a **Preference Model (PM)** on these AI-labeled pairs (combining them with helpfulness preferences).
  4. Run **RLAIF** starting from the SL-CAI policy and optimizing against the Preference Model to obtain the **Final RL-CAI** model.

Human oversight thus shifts from labeling thousands of raw harmful outputs to authoring and auditing a compact, explicit set of principles. Yet RLAIF comes with a critical catch: **AI judges inherit and amplify style biases**—length bias, list-formatting bias, and self-preference—unless those biases are explicitly constrained.

KEY IDEA

- Moving from SFT to optimization leverages the Generation–Verification gap so models can surpass demonstration writers, but human pairwise preferences are costly, demographically skewed, and vulnerable to assertiveness blind spots that hide 4× more factual errors under confident tone.
- RLAIF and Constitutional AI scale preference labeling at ∼10× lower cost with near-human agreement by using AI judges and explicit principles, but unconstrained AI judges inherit and amplify style biases.

Those style biases are not harmless noise—once you fit a scalar reward model to preference data and optimize a policy against it, the optimizer exploits every spurious correlation in the judge. Next, we analyze how scalar reward models are formalized, how length hacking breaks them, and how 2026 frontier generative judges stop it.

✎ Check Your Understanding

1. **Fill in the blank:** The asymmetry whereby checking or ranking candidate answers is much easier than writing an optimal answer from scratch is called the _____ gap, and it explains why RL optimization can surpass SFT demonstration writers.
   Answer: Generation–Verification (G–V).
2. **Yes or No:** In Stage 1 of Constitutional AI (SL-CAI), do human annotators rewrite the harmful red-team outputs into safe demonstrations? Briefly explain how the safe SFT data is produced.
   Answer: No. The model itself runs a constitution loop—generating an initial response, critiquing it against an explicit constitutional principle, and writing a revised response—and the SL-CAI model is fine-tuned on those self-revisions.

Section 6: Reward Modeling — Scalar Models, Length Hacking, and 2026 Generative Judges

To turn pairwise preference data $\mathcal{D} = \{(x, y_w, y_l)\}$—where y_w is the winning (preferred) completion and y_l is the losing (rejected) completion for prompt x—into a continuous scoring function that an RL optimizer can maximize, the classic RLHF pipeline trains a **Bradley–Terry scalar reward model** (Stiennon et al. 2020).

The Bradley–Terry Scalar Reward Model

Architecturally, a scalar reward model $r_\theta(x, y) \in \mathbb{R}$ is initialized from the SFT language model by replacing its vocabulary unembedding head with a single scalar linear head that outputs one real number at the final token of the response. Under the Bradley–Terry preference model, the probability that y_w is preferred over y_l given prompt x is modeled as the logistic sigmoid of their reward difference:

$$P(y_w \succ y_l \mid x) = \sigma(r_\theta(x, y_w) - r_\theta(x, y_l)), \quad \text{where } \sigma(z) = \frac{1}{1 + e^{-z}}$$

Training r_θ by maximum likelihood over the preference dataset yields the scalar reward model loss:

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

Decomposing this equation from the innermost expression outward:

- $r_\theta(x, y_w) - r_\theta(x, y_l)$ is the **reward margin** separating the preferred response from the rejected response. Notice that adding any prompt-dependent constant $c(x)$ to both rewards leaves the difference unchanged: $(r_\theta(x, y_w) + c(x)) - (r_\theta(x, y_l) + c(x)) = r_\theta(x, y_w) - r_\theta(x, y_l)$. We will rely heavily on this exact shift-invariance property when deriving DPO in Section 8.
- $\sigma(\cdot)$ maps that real-valued margin into $(0, 1)$: a large positive margin maps near 1 (high confidence that $y_w \succ y_l$), a zero margin maps to 0.5 (indifference), and a negative margin maps below 0.5 (wrong ranking).
- $-\log \sigma(\cdot)$ is the binary cross-entropy loss for predicting label 1 ("y_w beats y_l"). As the margin grows positive, the loss decays toward 0; as the margin becomes negative, the loss grows almost linearly with the magnitude of the mistake.
- Taking the expectation over $(x, y_w, y_l) \sim \mathcal{D}$ averages this pairwise classification loss across all annotated comparisons.

As a single conceptual unit, $\mathcal{L}_{\text{RM}}$ trains a neural scorer whose real-valued difference between any two completions predicts the log-odds that a human (or AI) judge prefers the first over the second.

> **WORKED EXAMPLE**
>
> **Computing Bradley–Terry reward model loss and verifying shift invariance**
>
> *Setup:* Given prompt x and preference pair (y_w, y_l), evaluate $\mathcal{L}_{\text{RM}}$ in two situations and verify what happens when both rewards shift by a constant.
>
> *Case A — Correct ranking:* Suppose $r_\theta(x, y_w) = 2.0$ and $r_\theta(x, y_l) = 0.0$.
>
> - Margin: $\Delta = 2.0 - 0.0 = +2.0$.
> - Predicted preference probability: $\sigma(2.0) = \frac{1}{1+e^{-2}} = \frac{1}{1+0.1353} = 0.8808$.
> - Loss: $-\ln(0.8808) = 0.1269$ nats.
>
> *Case B — Inverted ranking:* Suppose $r_\theta(x, y_w) = 0.0$ and $r_\theta(x, y_l) = 2.0$.
>
> - Margin: $\Delta = 0.0 - 2.0 = -2.0$.
> - Predicted preference probability: $\sigma(-2.0) = \frac{1}{1+e^{2}} = \frac{1}{1+7.3891} = 0.1192$.

- Loss: $-\ln(0.1192) = 2.1269$ nats—nearly $17\times$ larger than Case A, driving a strong gradient push to raise $r_\theta(x, y_w)$ and lower $r_\theta(x, y_l)$.

Shift-invariance check: Add $+10.0$ to both scores in Case A, giving $r_\theta(x, y_w) = 12.0$ and $r_\theta(x, y_l) = 10.0$. The margin is still $12.0 - 10.0 = +2.0$, and the loss is identically 0.1269 nats.

Interpretation: A Bradley–Terry reward model is only identified up to an arbitrary prompt-dependent offset $c(x)$; its predictions depend solely on relative differences between completions for the same prompt.

Length Hacking: A Major Side Effect of RLHF

Remember from Section 3 that human evaluators (~69%) and GPT-4 (~74%) systematically prefer longer responses. When a scalar reward model $r_\theta(x, y)$ is trained on length-biased preference pairs, length becomes one of the easiest features for the scalar head to latch onto. Once an optimizer pushes the language model to maximize $r_\theta(x, y)$, **length hacking** follows immediately.

Singhal et al. (2024) plotted **Reward Model Score** (vertical axis from -2.0 to $+1.0$) against **Output Length in tokens** (horizontal axis from 50 to 250). The empirical distribution forms a prominent upward-sloping elliptical density cloud tilting from the lower-left (short, negative reward) to the upper-right (long, positive reward). Marking a concrete prompt—*"Why don't adults roll off the bed?"*—on this plot exposes the pathology:

- **SFT point (59 tokens, lower-left of the cloud):** Produces a concise, accurate explanation of muscle memory and sleep posture, yet receives a low reward score below 0.
- **RLHF point (243 tokens, upper-right of the cloud):** A dashed red trajectory jumps from the SFT point straight up the diagonal density ridge to more than $4\times$ the length. It conveys the exact same core answer, now padded with verbose filler about general discomfort and safety—and earns a high reward near $+0.7$.

Crucially, **Chen et al. (2024)** and **Singhal et al. (2024)** showed that this strong reward–length correlation persists across **PPO, DPO, and ReMax** alike.

COMMON MISCONCEPTION

You might think that length hacking is a quirk specific to PPO and can be eliminated simply by switching to a different preference optimizer such as DPO or ReMax. That breaks because length bias lives in the preference data and the resulting reward signal (explicit or implicit); any algorithm that faithfully maximizes a length-biased preference objective will discover that padding responses is an easy way to earn higher reward.

Reward Models in 2026: Generative Judges & Verbosity Control

In late-2026 frontier pipelines, the weaknesses of scalar Bradley–Terry heads—vulnerability to surface hacks like length, opacity of a single number, and inability to verify nuanced prompt constraints—drove two major architectural replacements:

1. For **verifiable domains** (mathematics, software engineering, structured tool execution), **automated verifiers** replace learned reward models directly (joint RLVR + RLHF).
2. For **non-verifiable domains** (writing, open-ended advice, synthesis), scalar Bradley–Terry heads have largely given way to **agentic generative reward models** paired with hard **verbosity gates**.

> **DEFINITION**
> An **agentic generative judge** (or generative reward model) is an LLM-based evaluator that reads candidate completions, authors a prompt-specific grading rubric, scores each candidate in a structured scorepad against that rubric, and outputs pairwise or group rankings rather than a black-box scalar logit.

Kimi K3 (July 2026) illustrates this 2026 architecture in a four-stage evaluation pipeline:

1. **Candidate Pair (y_a, y_b) from policy rollouts** enters the evaluator $\rightarrow$
2. **Verbosity Gate:** Compare each candidate's token length against $\sigma \cdot L_{\text{ref}}$, where L_{ref} is a reference length estimated from the cold-start model and σ is a strict length-ratio threshold. If $\text{len}(y) > \sigma \cdot L_{\text{ref}}$, that candidate **automatically loses** its comparison $\rightarrow$
3. **Agentic Generative Judge:** Surviving candidates are read by the judge, which writes a prompt-specific rubric and scores each candidate inside a structured **scorepad** $\rightarrow$
4. **Group Rewards:** Pairwise outcomes across a group of rollouts are aggregated via **binary tournament comparisons** to assign normalized group rewards for online RL.

The verbosity gate solves length hacking structurally rather than hoping a scalar head learns to ignore length: padding past $\sigma \cdot L_{\text{ref}}$ guarantees defeat before the judge even evaluates prose quality.

> **WORKED EXAMPLE**
> **Applying Kimi K3's verbosity gate and generative judge to candidate rollouts**
>
> *Setup:* For the prompt *"Why don't adults roll off the bed?"*, suppose the cold-start SFT reference model yields an estimated reference length $L_{\text{ref}} = 70$ tokens, and the verbosity gate uses multiplier $\sigma = 1.6$. The maximum allowed length is therefore:
>
> $$\sigma \cdot L_{\text{ref}} = 1.6 \times 70 = 112 \text{ tokens}$$
>
> Suppose the current policy generates three candidate rollouts for a tournament:
>
> - y_1 (95 tokens): explains proprioception, sleep-stage muscle atonia, and learned bed-edge awareness clearly.
> - y_2 (108 tokens): covers the same mechanisms plus a brief note on why toddlers roll out more frequently.

- y_3 (243 tokens): pads the explanation with generic paragraphs on mattress comfort and sleep hygiene.

Step 1 — Verbosity gate check:

- $\text{len}(y_1) = 95 \leq 112$ (passes).
- $\text{len}(y_2) = 108 \leq 112$ (passes).
- $\text{len}(y_3) = 243 > 112$ (**fails — automatic loss** against any passing candidate).

Step 2 — Agentic generative judge on surviving pair (y_1, y_2): The judge generates a rubric for the prompt (e.g., accuracy of physiological mechanism, clarity, conciseness), scores y_1 and y_2 on its scorepad, and picks y_2 for adding the helpful developmental contrast without exceeding the length budget.

Step 3 — Binary tournament group outcome: y_2 beats y_1 (via rubric) and y_3 (via verbosity gate); y_1 beats y_3 (via verbosity gate).

Interpretation: Unlike a scalar RM where the 243-token padded answer y_3 would score highest, the hard verbosity gate strips away all reward for padding, forcing the policy to improve substantive rubric quality within the natural length envelope.

KEY IDEA

- Scalar Bradley–Terry reward models train via $-\mathbb{E}[\log \sigma(r_\theta(x, y_w) - r_\theta(x, y_l))]$, making them shift-invariant per prompt but highly vulnerable to length hacking across PPO, DPO, and ReMax whenever preference data favors verbose prose.
- In 2026 frontier pipelines such as Kimi K3, automated verifiers handle objective domains while non-verifiable tasks replace scalar heads with rubric-writing agentic generative judges preceded by a hard verbosity gate ($\text{len} > \sigma \cdot L_{\text{ref}} \implies$ automatic loss).

Now that we understand how reward signals—both classic scalar RMs and modern judges—are constructed, we turn to the optimization problem itself: how do we maximize expected reward without letting the policy collapse or drift outside the region where the reward model is trustworthy?

✎ Check Your Understanding

1. **Quick computation:** If a Bradley–Terry reward model assigns $r_\theta(x, y_w) = 1.0$ and $r_\theta(x, y_l) = 1.0$ (a tie), what probability does it predict for $y_w \succ y_l$, and what is the loss $-\ln \sigma(r_\theta(x, y_w) - r_\theta(x, y_l))$?
   Answer: The margin is $1.0 - 1.0 = 0$, so $\sigma(0) = 0.5$ and the loss is $-\ln(0.5) = \ln 2 \approx 0.693$ nats.
2. **What would happen if…** you lowered the Kimi K3 verbosity-gate multiplier σ from 1.5 to 1.0?
   Answer: Any candidate even a single token longer than the cold-start reference length L_{ref} would automatically lose its comparisons, aggressively preventing verbosity—potentially at the cost of penalizing genuinely thorough answers that need slightly more length than the cold-start model.

Section 7: The KL-Regularized RLHF Objective and a Conceptual Preview of PPO

Suppose we have trained a reward model $r_\theta(x, y)$ and we now attempt to directly maximize $\mathbb{E}_{y \sim \pi_\phi^{\text{RL}}(\cdot|x)}[r_\theta(x, y)]$ with respect to the policy parameters ϕ. Unconstrained reward maximization fails in two ways almost immediately: first, the policy collapses onto a single high-reward string (losing language fluency and diversity); second, the policy drifts into out-of-distribution text regions where the reward model r_θ was never trained and assigns spuriously huge scores to nonsense.

The KL-Regularized RLHF Objective

To keep the RL policy anchored near the well-behaved SFT distribution π^{SFT}, **Ouyang et al. (2022, InstructGPT)** introduced the **KL-regularized RLHF objective**:

$$J(\phi) = \mathbb{E}_{(x,y)\sim\pi_\phi^{\text{RL}}}\left[r_\theta(x, y) - \beta \log \frac{\pi_\phi^{\text{RL}}(y \mid x)}{\pi^{\text{SFT}}(y \mid x)}\right] + \gamma\, \mathbb{E}_{x\sim D_{\text{pretrain}}}\left[\log \pi_\phi^{\text{RL}}(x)\right]$$

Let us unpack every symbol and subexpression hierarchically:

- $\pi_\phi^{\text{RL}}(y \mid x)$ is the trainable RL policy with parameters ϕ, initialized from π^{SFT}.
- $r_\theta(x, y)$ is the scalar reward assigned to completion y for prompt x (from the frozen reward model).
- $\log \frac{\pi_\phi^{\text{RL}}(y|x)}{\pi^{\text{SFT}}(y|x)}$ is the sequence-level log probability ratio between the current RL policy and the frozen SFT reference policy. Because both policies factorize autoregressively, this equals the sum of per-token log-ratios $\sum_{t=1}^{T} \log \frac{\pi_\phi^{\text{RL}}(y_t|x, y_{<t})}{\pi^{\text{SFT}}(y_t|x, y_{<t})}$.
- Taking the expectation of this log-ratio under $y \sim \pi_\phi^{\text{RL}}(\cdot \mid x)$ yields the exact **Kullback–Leibler (KL) divergence**:

$$\mathbb{E}_{y \sim \pi_\phi^{\mathrm{RL}}(\cdot|x)}\left[\log \frac{\pi_\phi^{\mathrm{RL}}(y \mid x)}{\pi^{\mathrm{SFT}}(y \mid x)}\right] = \mathbb{D}_{\mathrm{KL}}\left[\pi_\phi^{\mathrm{RL}}(\cdot \mid x) \,\|\, \pi^{\mathrm{SFT}}(\cdot \mid x)\right] \geq 0$$

- Multiplying by $-\beta$ (where $\beta > 0$ is the **KL penalty coefficient**) subtracts a penalty whenever π_ϕ^{RL} puts much higher probability on a completion y than π^{SFT} did. This per-token KL penalty prevents the policy from collapsing to a single mode or drifting outside the reward model's training distribution.
- The second expectation, $+\gamma\, \mathbb{E}_{x \sim D_{\text{pretrain}}}[\log \pi_\phi^{\mathrm{RL}}(x)]$, is the **pretraining mix ("PPO-ptx")**: it adds back a weighted ($\gamma \geq 0$) next-token log-likelihood gradient on general pretraining text D_{pretrain} to prevent regression on general NLP benchmarks. Setting $\gamma = 0$ recovers plain KL-regularized PPO.

Taken as a whole, $J(\phi)$ balances three forces simultaneously: climb toward higher reward r_θ, stay close to the SFT reference distribution via β, and preserve general pretrained capabilities via γ. We will reuse the exact first term of $J(\phi)$ (with $\gamma = 0$) as the starting equation for deriving DPO in Section 8.

WORKED EXAMPLE

How the KL penalty β blocks a reward-hacked response

Setup: Let $\beta = 0.1$ (and $\gamma = 0$). For a given prompt x, compare two candidate completions evaluated under the regularized reward $r_{\text{reg}}(x, y) = r_\theta(x, y) - \beta \log \frac{\pi_\phi^{\mathrm{RL}}(y|x)}{\pi^{\mathrm{SFT}}(y|x)}$:

- Response y_1 (faithful, natural): $r_\theta(x, y_1) = 1.5$, $\log \pi^{\mathrm{SFT}}(y_1 \mid x) = -20.0$, $\log \pi_\phi^{\mathrm{RL}}(y_1 \mid x) = -18.0$.
- Response y_2 (out-of-distribution reward hack): $r_\theta(x, y_2) = 2.1$, $\log \pi^{\mathrm{SFT}}(y_2 \mid x) = -35.0$ (very unlikely under SFT), $\log \pi_\phi^{\mathrm{RL}}(y_2 \mid x) = -25.0$ (heavily upweighted by the RL policy).

Step 1 — Compute log-ratios and penalized rewards at $\beta = 0.1$:

- For y_1: $\log \frac{\pi_\phi^{\mathrm{RL}}}{\pi^{\mathrm{SFT}}} = -18.0 - (-20.0) = +2.0$.

$$r_{\text{reg}}(x, y_1) = 1.5 - 0.1 \times 2.0 = 1.30$$

- For y_2: $\log \frac{\pi_\phi^{\mathrm{RL}}}{\pi^{\mathrm{SFT}}} = -25.0 - (-35.0) = +10.0$.

$$r_{\text{reg}}(x, y_2) = 2.1 - 0.1 \times 10.0 = 1.10$$

Step 2 — Check what happens if β is set too small ($\beta = 0.02$):

- $r_{\text{reg}}(x, y_1) = 1.5 - 0.02 \times 2.0 = 1.46$.
- $r_{\text{reg}}(x, y_2) = 2.1 - 0.02 \times 10.0 = 1.90$.

Interpretation: At $\beta = 0.1$, the KL penalty overrides the $+0.6$ raw reward advantage of y_2, keeping y_1 preferred ($1.30 > 1.10$). If β is weakened to 0.02, the regularization barrier disappears and the optimizer chases the out-of-distribution hack y_2 ($1.90 > 1.46$).

PPO at a Conceptual Level (Brief Preview)

Because sampling $y \sim \pi_\phi^{\mathrm{RL}}(\cdot \mid x)$ is a discrete, non-differentiable operation over tokens, we cannot backpropagate directly through the sampled text into ϕ. Classic online RLHF therefore uses policy-gradient optimization, culminating in **Proximal Policy Optimization (PPO)**. At a conceptual level, the progression from basic policy gradients to PPO unfolds in three steps (we will derive PPO, GRPO, and modern online RL in full mathematical detail in Lecture 16):

1. **Policy gradients (REINFORCE):** Using the log-derivative identity $\nabla_\theta p_\theta(z) = p_\theta(z)\nabla_\theta \log p_\theta(z)$, the gradient of expected reward is:

   $$\nabla_\theta \mathbb{E}_{p_\theta}[R(z)] = \mathbb{E}_{p_\theta}[R(z)\, \nabla_\theta \log p_\theta(z)]$$

   This estimator is unbiased, but its variance is far too high across long token sequences to train large language models stably.

2. **Trust Region Policy Optimization (TRPO):** To prevent destructive updates from noisy gradient steps, TRPO linearizes the objective around the current policy $\pi_{\theta_{\text{old}}}$ using an **advantage estimate** $\hat{A}_t$ (how much better action a_t is than average in state s_t) and imposes a hard second-order trust-region KL constraint:

   $$\max_\theta \ \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\,\hat{A}_t\right] \quad \text{s.t.} \quad \hat{\mathbb{E}}_t[\mathrm{KL}(\pi_{\theta_{\text{old}}}(\cdot \mid s_t) \,\|\, \pi_\theta(\cdot \mid s_t))] \le \delta$$

   Here the probability ratio $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ reweights actions sampled under $\pi_{\theta_{\text{old}}}$ so we can evaluate updates to π_θ, while δ caps how far the policy may move in one step. Enforcing the exact KL constraint requires expensive second-order optimization.

3. **Proximal Policy Optimization (PPO):** Replaces TRPO's expensive second-order KL constraint with a simple first-order **clipped probability ratio** surrogate:

   $$L^{\mathrm{CLIP}} = \min\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_k}(a \mid s)} A^{\pi_{\theta_k}}(s,a),\ \mathrm{clip}\left(\frac{\pi_\theta(a \mid s)}{\pi_{\theta_k}(a \mid s)},\, 1-\epsilon,\, 1+\epsilon\right) A^{\pi_{\theta_k}}(s,a)\right)$$

   - The ratio $r_t(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$ measures how much the updated policy shifts probability on action a relative to the policy π_{θ_k} that generated the rollout.
   - $\mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$ clamps that ratio to $[1-\epsilon, 1+\epsilon]$, removing any incentive to move the policy outside the trust interval.
   - Taking the $\min(\cdot,\cdot)$ creates a pessimistic lower bound: when $A > 0$, the ratio is capped at $1+\epsilon$ so the policy cannot over-increase probability on a good token in one step; when $A < 0$, the ratio is floored at $1-\epsilon$ so the policy cannot over-suppress a bad token in one step.

Running PPO in practice is systems-heavy: it requires maintaining **four** large models or components in concert—the trainable policy π_ϕ^{RL}, the frozen reference policy π^{SFT}, the frozen reward model r_θ, and a separate trainable **value network** (critic, often initialized from the RM) to estimate advantages A—all driven by an online rollout generation loop.

Can We Avoid On-Policy RL? Simple Offline Baselines

Because PPO has so many moving parts and sensitive hyperparameters, practitioners first explored four simpler alternatives that avoid an online RL loop:

Idea	How It Works & Trade-off
Control tokens	Prepend `[GOOD]` to chosen responses and `[BAD]` to rejected responses; run standard SFT on both; condition on `[GOOD]` at inference. Simple, but uses preference data weakly and relies on prompt conditioning.
SFT on preferred only	Discard all rejected completions y_l and run standard SFT on y_w. Easy to run, but throws away all negative contrast signal.
Rejection sampling / RFT	Train a reward model, sample multiple LM outputs per prompt, and run SFT on the top-ranked output. Stable, yet wastes lower-ranked samples and still trains via pure SFT.
Best-of-N (inference)	At serving time, sample e.g. $N = 1024$ completions and return the highest-RM output. Surprisingly strong quality baseline, but imposes prohibitive inference cost.

COMMON MISCONCEPTION

You might think that simply discarding rejected completions and running SFT on preferred responses y_w—or using Best-of-N sampling—gives the same alignment efficiency as KL-regularized RLHF without the complexity. That breaks because SFT on y_w throws away the negative signal telling the model what *not* to generate (and cannot enforce a margin between y_w and y_l), while Best-of-N shifts the optimization burden into massive inference-time compute (e.g., $1024\times$ generation cost per query) instead of amortizing it into the model weights.

KEY IDEA

- The KL-regularized RLHF objective $J(\phi)$ maximizes reward $r_\theta(x, y)$ minus a per-token KL penalty $\beta \log(\pi_\phi^{\text{RL}}/\pi^{\text{SFT}})$ that prevents mode collapse and reward-model drift, optionally plus a pretraining mix γ ("PPO-ptx").
- PPO optimizes this objective stably via a clipped probability ratio L^{CLIP}, but requires a policy, reference model, reward model, value network, and online rollout loop—motivating offline alternatives that avoid full RL machinery.

None of those four heuristic baselines actually optimizes the KL-regularized RLHF objective $J(\phi)$. This raises a clean mathematical question: can we keep the **exact** KL-regularized RLHF objective while eliminating both the explicit reward model and the online RL loop altogether? Answering that question

leads directly to Direct Preference Optimization.

✎ Check Your Understanding

1. **What would happen if…** you set $\beta = 0$ in the KL-regularized RLHF objective $J(\phi)$ while optimizing against a learned scalar reward model r_θ?
   Answer: Without the KL penalty anchoring π_ϕ^{RL} to π^{SFT}, the policy would quickly collapse to a narrow mode and drift outside the reward model's training distribution, exploiting spurious high-reward artifacts.
2. **Yes or No:** In PPO's clipped objective L^{CLIP}, if an action has positive advantage ($A > 0$) and the probability ratio $\pi_\theta(a \mid s)/\pi_{\theta_k}(a \mid s)$ already exceeds $1 + \epsilon$, does increasing $\pi_\theta(a \mid s)$ further increase L^{CLIP}? Explain briefly.
   Answer: No. Because L^{CLIP} takes the minimum of the unclipped and clipped terms, once the ratio exceeds $1 + \epsilon$ for $A > 0$, the objective plateaus at $(1 + \epsilon)A$, giving zero gradient for pushing the ratio higher.

Section 8: Direct Preference Optimization (DPO) — Derivation, Gradients, Variants, and 2026 Role

Comparing the workflow of classic RLHF against **Direct Preference Optimization (DPO)** clarifies the engineering simplification DPO achieves:

- **Classic RLHF (2 stages + rollouts):** Preference pairs $(x, y_w \succ y_l) \rightarrow$ fit an explicit scalar Reward Model r_θ via MLE $\rightarrow$ run an iterative PPO loop alternating between generating policy *rollouts* and receiving *rewards* from r_θ (plus a value network).
- **Direct Preference Optimization (DPO):** Preference pairs $(x, y_w \succ y_l) \rightarrow$ minimize a **closed-form MLE loss** directly on the policy parameters $\theta \rightarrow$ obtain the final LM π_θ in a single supervised pass—no reward model, no value network, and no online rollouts.

Intuitively, DPO takes positive gradient steps on the log-likelihood of preferred responses y_w and appropriately weighted negative steps on rejected responses y_l. Remarkably, this exact loss does not have to be guessed heuristically: it falls out **analytically** from the exact KL-regularized RLHF objective of Section 7.

DEFINITION

Direct Preference Optimization (DPO) is an offline preference-alignment algorithm that uses the closed-form solution of the KL-regularized RLHF objective to reparameterize the reward function in terms of the policy itself, reducing RLHF to a single supervised binary cross-entropy loss over pairwise preference data.

Full Derivation of the DPO Loss

We derive DPO in three rigorous steps, filling in every algebraic manipulation.

Step 1: Closed-Form Nonparametric Optimum of KL-Regularized RLHF

Start from the KL-regularized RLHF objective (Section 7, with $\gamma = 0$) for any fixed reward function $r(x, y)$ and frozen reference policy π_{ref} (typically π^{SFT}):

$$\max_{\pi} \; \mathbb{E}_{x \sim \mathcal{D},\, y \sim \pi(y|x)}[r(x, y)] - \beta \, \mathbb{D}_{\text{KL}}[\pi(y \mid x) \,\|\, \pi_{\text{ref}}(y \mid x)]$$

Make the **nonparametric assumption**: optimize over the space of *all* valid conditional probability distributions $\pi(\cdot \mid x)$ for each prompt x, ignoring neural network parameterization for the moment. For a fixed prompt x, expand the KL divergence and negate the objective to turn maximization into minimization:

$$\min_{\pi(\cdot|x)} \; \mathbb{E}_{y \sim \pi(\cdot|x)} \left[\beta \log \frac{\pi(y \mid x)}{\pi_{\text{ref}}(y \mid x)} - r(x, y) \right]$$

Divide through by the positive constant $\beta > 0$ (which does not change the minimizer):

$$\min_{\pi(\cdot|x)} \; \mathbb{E}_{y \sim \pi(\cdot|x)} \left[\log \frac{\pi(y \mid x)}{\pi_{\text{ref}}(y \mid x)} - \frac{1}{\beta} r(x, y) \right]$$

Since $\frac{1}{\beta} r(x, y) = \log \exp\left(\frac{1}{\beta} r(x, y)\right)$, combine the two terms inside a single logarithm:

$$\min_{\pi(\cdot|x)} \; \mathbb{E}_{y \sim \pi(\cdot|x)} \left[\log \frac{\pi(y \mid x)}{\pi_{\text{ref}}(y \mid x) \exp\left(\frac{1}{\beta} r(x, y)\right)} \right]$$

The denominator $\pi_{\text{ref}}(y \mid x) \exp\left(\frac{1}{\beta} r(x, y)\right)$ is strictly positive for all y in the support of π_{ref}, but it does not sum to 1. Define its normalizing constant (the **partition function**):

$$Z(x) = \sum_{y} \pi_{\text{ref}}(y \mid x) \exp\left(\frac{1}{\beta} r(x, y)\right)$$

and define the properly normalized conditional distribution:

$$\pi_r(y \mid x) = \frac{1}{Z(x)} \, \pi_{\text{ref}}(y \mid x) \, \exp\left(\frac{1}{\beta} r(x, y)\right)$$

Now multiply and divide the denominator inside our logarithm by $Z(x)$, so that $\pi_{\text{ref}}(y \mid x) \exp(\frac{1}{\beta} r(x, y)) = Z(x) \, \pi_r(y \mid x)$:

$$\begin{aligned}\mathbb{E}_{y\sim\pi(\cdot|x)}\left[\log\frac{\pi(y\mid x)}{Z(x)\,\pi_r(y\mid x)}\right] &= \mathbb{E}_{y\sim\pi(\cdot|x)}\left[\log\frac{\pi(y\mid x)}{\pi_r(y\mid x)}\right] - \log Z(x)\\ &= \mathbb{D}_{\mathrm{KL}}[\pi(y\mid x)\,\|\,\pi_r(y\mid x)] - \log Z(x)\end{aligned}$$

Look at that final expression: $\log Z(x)$ does not depend on π at all, and by Gibbs' inequality, $\mathbb{D}_{\mathrm{KL}}[\pi\|\pi_r] \geq 0$ with equality if and only if $\pi(y\mid x) = \pi_r(y\mid x)$ everywhere. Therefore, the unique closed-form maximizer of the KL-regularized RLHF objective is:

$$\pi_r(y\mid x) = \frac{1}{Z(x)}\,\pi_{\mathrm{ref}}(y\mid x)\,\exp\!\left(\frac{1}{\beta}r(x,y)\right)$$

Step 2: Solve for the Implied Reward

In RLHF, we normally know $r(x,y)$ and struggle to find π_r because computing $Z(x)$ requires summing over all exponentially many token sequences y. DPO flips the algebra around: solve the closed-form equation for $r(x,y)$ in terms of its optimal policy π_r.

Divide both sides by $\pi_{\mathrm{ref}}(y\mid x)$, multiply by $Z(x)$, take logarithms, and multiply by β:

$$\frac{\pi_r(y\mid x)}{\pi_{\mathrm{ref}}(y\mid x)} = \frac{1}{Z(x)}\exp\!\left(\frac{1}{\beta}r(x,y)\right) \implies \log\frac{\pi_r(y\mid x)}{\pi_{\mathrm{ref}}(y\mid x)} = \frac{1}{\beta}r(x,y) - \log Z(x)$$

Rearranging gives the exact reward–policy equivalence:

$$r(x,y) = \beta\log\frac{\pi_r(y\mid x)}{\pi_{\mathrm{ref}}(y\mid x)} + \beta\log Z(x)$$

Note this result carefully: we will reuse this exact nonparametric optimum and implied-reward equivalence directly in Lecture 16 (and it appears in modern reasoning frameworks such as Kimi-think).

Step 3: Cancel $Z(x)$ Inside the Bradley–Terry Pairwise Loss

Even after Step 2, the intractable partition function $\beta\log Z(x)$ still appears in $r(x,y)$. However, notice that $Z(x)$ depends **only on the prompt** x, not on the completion y!

Recall from Section 6 that the Stiennon / Bradley–Terry pairwise reward loss depends only on the *difference* $r(x,y_w) - r(x,y_l)$ for the same prompt x:

$$\mathcal{L}_{\mathrm{RM}}(r) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}[\log\sigma(r(x,y_w) - r(x,y_l))]$$

Substitute our Step 2 expression for $r(x,y)$ with our parameterized policy π_θ taking the place of π_r:

$$r(x,y_w) - r(x,y_l) = \left(\beta\log\frac{\pi_\theta(y_w\mid x)}{\pi_{\mathrm{ref}}(y_w\mid x)} + \cancel{\beta\log Z(x)}\right) - \left(\beta\log\frac{\pi_\theta(y_l\mid x)}{\pi_{\mathrm{ref}}(y_l\mid x)} + \cancel{\beta\log Z(x)}\right)$$

The intractable partition function $\beta\log Z(x)$ cancels identically! Plugging the difference into the Bradley–Terry loss yields the **DPO objective**:

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

In three steps: (1) the nonparametric closed form links $\pi_\theta \leftrightarrow r$; (2) we reparameterize r via π_θ; (3) substituting into pairwise preferences cancels $Z(x)$ and turns RLHF into maximum likelihood directly over the policy parameters θ.

What Does the DPO Gradient Actually Do?

Define the policy's **implied reward** (dropping the canceled $Z(x)$ offset) as:

$$\hat{r}_\theta(x, y) = \beta \log \frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)}$$

> **DEFINITION**
> The **implied reward** $\hat{r}_\theta(x, y) = \beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$ is the reward function (up to an additive prompt-dependent constant) for which the current policy π_θ would be the exact KL-regularized RLHF optimum relative to π_{ref}.

Let $u = \hat{r}_\theta(x, y_w) - \hat{r}_\theta(x, y_l)$. To differentiate $-\log \sigma(u)$ with respect to θ, recall two elementary calculus facts about the sigmoid $\sigma(u) = (1 + e^{-u})^{-1}$:

1. $\sigma'(u) = \sigma(u)(1 - \sigma(u))$.
2. $1 - \sigma(u) = \sigma(-u)$.

By the chain rule:

$$\nabla_\theta[-\log \sigma(u)] = -\frac{\sigma'(u)}{\sigma(u)} \nabla_\theta u = -(1 - \sigma(u)) \nabla_\theta u = -\sigma(-u) \nabla_\theta u$$

Now compute $\nabla_\theta u$. Because π_{ref} is frozen, $\nabla_\theta \log \pi_{\text{ref}}(y \mid x) = 0$, leaving:

$$\nabla_\theta u = \beta(\nabla_\theta \log \pi_\theta(y_w \mid x) - \nabla_\theta \log \pi_\theta(y_l \mid x))$$

Substituting $\nabla_\theta u$ and $-u = \hat{r}_\theta(x, y_l) - \hat{r}_\theta(x, y_w)$ produces the DPO gradient:

$$\nabla_\theta \mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\beta\, \mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\Big[\underbrace{\sigma(\hat{r}_\theta(x, y_l) - \hat{r}_\theta(x, y_w))}_{\text{higher weight when implied RM is wrong}} \cdot \Big(\underbrace{\nabla_\theta \log \pi_\theta(y_w \mid x)}_{\text{increase } y_w} - \underbrace{\nabla_\theta}$$

Every component of this gradient has a sharp operational role:

- **Update direction:** Since gradient *descent* subtracts $\nabla_\theta \mathcal{L}_{\text{DPO}}$, the leading minus sign cancels and the update moves in the direction $+\nabla_\theta \log \pi_\theta(y_w \mid x) - \nabla_\theta \log \pi_\theta(y_l \mid x)$, pushing probability mass **up** on preferred responses y_w and **down** on rejected responses y_l.
- **Adaptive weighting:** The scalar multiplier $\sigma(\hat{r}_\theta(x, y_l) - \hat{r}_\theta(x, y_w)) \in (0, 1)$ inspects how badly the policy currently misranks the pair. When the policy already ranks y_w far above y_l ($\hat{r}_w \gg \hat{r}_l$),

$\hat{r}_l - \hat{r}_w$ is strongly negative and the weight $\sigma(\hat{r}_l - \hat{r}_w) \to 0$. When the policy gets the pair backwards ($\hat{r}_l > \hat{r}_w$), the weight approaches 1.

- **Why the weight matters:** Naive unweighted positive/negative gradient updates keep pushing down $\log \pi_\theta(y_l \mid x)$ even after the model already avoids y_l, rapidly destabilizing the language model and collapsing generation fluency. The adaptive sigmoid weight derived from KL regularization automatically brakes updates once the margin is satisfied.

WORKED EXAMPLE

Numerical evaluation of DPO loss and adaptive gradient weights

Setup: Let $\beta = 0.1$. Evaluate $\hat{r}_\theta$, $\mathcal{L}_{\text{DPO}}$, and the gradient weight $\sigma(\hat{r}_l - \hat{r}_w)$ on a preference pair (x, y_w, y_l) in two regimes.

Case 1 — Policy currently misranks the pair: Suppose $\log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} = -5.0$ and $\log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)} = +15.0$.

- Implied rewards:

$$\hat{r}_w = 0.1 \times (-5.0) = -0.5, \qquad \hat{r}_l = 0.1 \times (+15.0) = +1.5$$

- Margin: $\hat{r}_w - \hat{r}_l = -0.5 - 1.5 = -2.0$.
- DPO loss: $-\ln \sigma(-2.0) = -\ln(0.1192) = 2.127$ nats.
- Adaptive gradient weight: $\sigma(\hat{r}_l - \hat{r}_w) = \sigma(+2.0) = \frac{1}{1+e^{-2}} = 0.881$.

Case 2 — Policy already ranks the pair correctly with a clear margin: Suppose $\log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} = +20.0$ and $\log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)} = -10.0$.

- Implied rewards: $\hat{r}_w = +2.0$, $\hat{r}_l = -1.0$.
- Margin: $\hat{r}_w - \hat{r}_l = +3.0$.
- DPO loss: $-\ln \sigma(+3.0) = -\ln(0.9526) = 0.0486$ nats.
- Adaptive gradient weight: $\sigma(\hat{r}_l - \hat{r}_w) = \sigma(-3.0) = \frac{1}{1+e^{3}} = 0.0474$.

Interpretation: When the policy gets the preference backwards (Case 1), DPO applies 88.1% of full gradient strength to swap the ranking. Once the policy prefers y_w by a $+3.0$ implied-reward margin (Case 2), the gradient weight automatically drops nearly $19\times$ to 4.7%, protecting the language model from over-pushing probabilities and collapsing fluency.

COMMON MISCONCEPTION

You might think that because DPO eliminates the explicit reward model, it escapes the assumptions of Bradley–Terry reward modeling and KL-regularized RLHF. That breaks because DPO is mathematically derived *directly* from both assumptions—it still assumes pairwise preferences follow a Bradley–Terry difference model and optimizes the exact same KL-regularized objective; it simply folds the reward model into the policy's own log-probability ratio $\hat{r}_\theta$.

DPO Variants & Iterative DPO

Two widely studied modifications address length bias and memory overhead in vanilla DPO:

1. **Length-Normalized DPO** (highlighted in **Tülu 3, 2024**): Because sequence log-probability $\log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$ sums over $|y|$ tokens, longer completions naturally accumulate larger magnitudes, reinforcing length bias. Dividing each log-ratio by completion length $|y|$ normalizes the implied reward to a per-token average:

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

2. **SimPO (Simple Preference Optimization):** Removes the frozen reference model π_{ref} altogether (cutting memory and forward-pass cost) while retaining length normalization and introducing an explicit target reward margin $\gamma > 0$:

$$-\mathbb{E}\left[\log \sigma\left(\frac{\beta}{|y_w|}\log\pi_\theta(y_w \mid x) - \frac{\beta}{|y_l|}\log\pi_\theta(y_l \mid x) - \gamma\right)\right]$$

   Subtracting γ requires the preferred response's length-normalized log-probability to exceed the rejected response's by at least γ/β before the loss approaches zero.

WORKED EXAMPLE

Why unnormalized DPO favors longer responses and how Length-Normalized DPO corrects it

Setup: Suppose $\beta = 0.1$. Compare a long preferred response ($|y_w| = 200$ tokens) whose average per-token log-ratio is $+0.05$ against a concise rejected response ($|y_l| = 40$ tokens) whose average per-token log-ratio is $+0.10$.

Step 1 — Standard (unnormalized) DPO implied rewards:

- Summed log-ratio for y_w: $200 \times 0.05 = +10.0 \implies \hat{r}_w = 0.1 \times 10.0 = +1.00$.
- Summed log-ratio for y_l: $40 \times 0.10 = +4.0 \implies \hat{r}_l = 0.1 \times 4.0 = +0.40$.
- Unnormalized margin: $\hat{r}_w - \hat{r}_l = +0.60 > 0$. Standard DPO believes y_w already beats y_l purely because y_w has $5\times$ more tokens to sum over.

Step 2 — Length-Normalized DPO implied rewards:

- Normalized implied reward for y_w: $\frac{0.1}{200}(10.0) = 0.005$.
- Normalized implied reward for y_l: $\frac{0.1}{40}(4.0) = 0.010$.
- Normalized margin: $0.005 - 0.010 = -0.005 < 0$.

Interpretation: Length normalization strips out the $5\times$ sequence-length multiplier so the model cannot satisfy the preference margin merely by inflating response length.

A deeper limitation of vanilla DPO is that it is **off-policy**: it trains on a static dataset of pairs (y_w, y_l) generated before optimization began. As π_θ changes, its own generated completions drift away from the static pairs. To bridge this gap, historical open recipes such as **Llama 3 (July 2024)** used **Iterative DPO** (a form of expert iteration), arranged as a repeating four-step loop:

1. **Sample K outputs per prompt** from the current policy $\rightarrow$
2. **RM Rejection Sampling** to identify the highest-quality completions $\rightarrow$
3. **SFT on Best + Capability Data** to anchor the policy on the winning samples without losing skills $\rightarrow$
4. **DPO on Fresh Preference Pairs** drawn from the updated policy $\rightarrow$ loop back to Step 1 for the next round.

Refreshing the preference pairs each round brings off-policy DPO closer to on-policy RL, though at the cost of reintroducing a reward model and repeated sampling passes.

DPO vs. PPO — and Where DPO Fits in 2026

Empirical comparisons between DPO, its variants, and PPO show that headline algorithm rankings are **highly contingent** on data quality, reward-model scale, and hyperparameter tuning (especially β):

Study & Setup	Aggregate Score
Ivison et al. (2024): Initial SFT	56.8
$\rightarrow$ DPO (weak preference data $\rightarrow$ better preference data)	$58.1 \rightarrow 61.0$
$\rightarrow$ PPO $\rightarrow$ PPO + bigger RM	$62.2 \rightarrow \mathbf{62.8}$
Tülu 3 (2024): SFT Base	55.7
$\rightarrow$ SimPO / DPO / PPO	52.9 / 55.2 / 55.5
$\rightarrow$ DPO-norm (highly β-sensitive: 46.8–$\mathbf{57.3}$)	**57.3**

Look closely at the Tülu 3 row: changing β alone swings Length-Normalized DPO from 46.8 (nearly 9 points *worse* than the SFT base) up to 57.3 (the best score in the table), and in Ivison et al., upgrading preference data (+2.9 points for DPO) and scaling the RM matter more than the choice of optimizer alone.

That brings us to the **2026 frontier verdict**:

- **None of the three late-2026 frontier reports** (**Nemotron 3.5 Super**, **Kimi K3**, **DeepSeek-V4.1-Flash**) uses offline DPO. Frontier pipelines replaced offline pairwise preference tuning with **online RL (joint RLVR + RLHF) plus on-policy distillation (MOPD)**.
- **DPO remains the standard accessible method** for open and academic post-training—and the right first preference-optimization algorithm to master—because it requires no separate reward model, value network, or rollout infrastructure.

We examine *why* online RL plus MOPD decisively beat offline DPO at the frontier in Lecture 16.

KEY IDEA

- DPO derives a closed-form supervised loss directly from KL-regularized RLHF by solving for the nonparametric optimum π_r, inverting for the implied reward $r(x, y) = \beta \log(\pi_r / \pi_{\text{ref}}) + \beta \log Z(x)$, and canceling $Z(x)$ inside the Bradley–Terry pairwise difference.
- Its gradient increases $\log \pi_\theta(y_w \mid x)$ and decreases $\log \pi_\theta(y_l \mid x)$ weighted by $\sigma(\hat{r}_l - \hat{r}_w)$; variants like Length-Normalized DPO, SimPO, and Iterative DPO address length bias, memory cost, and off-policy drift, though empirical performance remains highly sensitive to data quality and β.

Whether you optimize via PPO, DPO, Best-of-N, or modern online RL, pushing optimization too hard against any imperfect reward signal triggers two deep, fundamental failure modes that affect every alignment pipeline. We analyze both in the final section.

✎ Check Your Understanding

1. **Quick computation:** In DPO with $\beta = 0.2$, suppose for a preference pair (x, y_w, y_l) the log-ratios are $\log(\pi_\theta(y_w \mid x)/\pi_{\text{ref}}(y_w \mid x)) = 5$ and $\log(\pi_\theta(y_l \mid x)/\pi_{\text{ref}}(y_l \mid x)) = 5$. What are the implied rewards $\hat{r}_w$ and $\hat{r}_l$, and what is the adaptive gradient weight $\sigma(\hat{r}_l - \hat{r}_w)$?
   Answer: $\hat{r}_w = 0.2 \times 5 = 1.0$ and $\hat{r}_l = 0.2 \times 5 = 1.0$, so $\hat{r}_l - \hat{r}_w = 0$ and the adaptive gradient weight is $\sigma(0) = 0.5$.
2. **Fill in the blank:** In the DPO derivation, the intractable partition function $\beta \log Z(x)$ vanishes when substituted into the Bradley–Terry loss because $Z(x)$ depends only on _____, not on the completion y.
   Answer: the prompt x.

Section 9: Fundamental Failure Modes of RLHF and the Unified 2026 Stack

Optimizing a language model policy against preferences works remarkably well up to a point—and then breaks in two predictable, mathematically grounded ways: **reward overoptimization** (degrading true quality while proxy reward keeps climbing) and **mode collapse / lost calibration** (destroying probability calibration and output diversity).

Failure Mode 1: Reward Overoptimization (Goodhart's Law)

Any learned reward model—whether a scalar Bradley–Terry head or a noisy AI judge—is an imperfect proxy for true human judgment ("gold reward"). **Goodhart's Law** states that when a measure becomes a target, it ceases to be a good measure. In RLHF, pushing the RL policy too far in KL distance from the

initial SFT policy lets the optimizer discover blind spots in the proxy reward model, raising proxy scores while true quality collapses.

> **DEFINITION**
> **Reward overoptimization** is the RLHF failure mode (an instance of Goodhart's Law) in which increasing optimization pressure—measured by KL divergence from the initial policy—continues to increase the learned proxy reward model's score while true ("gold") output quality peaks and then degrades.

Gao et al. quantified this phenomenon cleanly in a classic plot of **Reward Score** (vertical axis from 0.0 to 1.4) against **KL Distance between the RL and Initial Policy** (horizontal axis from 0 to 100). Four curves trace out the exact trade-off:

- **Proxy RM (dashed grey curve):** Starts at 0.0 at $\mathrm{KL} = 0$ and climbs monotonically without bound, surpassing 1.4 well before $\mathrm{KL} = 100$.
- **Gold reward under a 3M-parameter proxy RM (red curve):** Rises alongside the proxy briefly, peaks early around a gold score of ~ 0.45 near $\mathrm{KL} \approx 25$–30, and then collapses steeply all the way back to 0.0 by $\mathrm{KL} = 100$.
- **Gold reward under an 85M-parameter proxy RM (blue curve):** Climbs higher, peaks around ~ 0.65 near $\mathrm{KL} \approx 35$–40, and then declines steadily down toward ~ 0.30 by $\mathrm{KL} = 100$.
- **Gold reward under a 3B-parameter proxy RM (green curve):** Achieves the highest peak (~ 0.90 near $\mathrm{KL} \approx 60$–70) and degrades most slowly, yet *still* bends downward past its peak.

Two structural conclusions emerge from these curves:

1. **Scaling the reward model helps, but never solves the problem:** Going from a 3M to a 3B proxy RM raises the peak gold reward and delays collapse to a larger KL budget, yet the downward turn always arrives eventually.
2. **Overoptimization is algorithm-agnostic (Dubois et al. 2023):** The exact same inverted-U gold-reward collapse appears across **PPO, Best-of-n, and Expert Iteration**, under both human preferences and noisy LM judges. It disappears *only* when optimizing against a noiseless oracle judge (which is precisely why verifiable automated checks in RLVR and strict rubric + verbosity gates in 2026 generative judges are so valuable).

Failure Mode 2: Mode Collapse & Lost Calibration

Pretraining via maximum likelihood over diverse human text produces a well-calibrated probability distribution: if a pretrained model assigns 70% probability to answer choice A on a multiple-choice question, answer A is right roughly 70% of the time. Reward maximization breaks that property. Optimizing expected reward does not reward matching the spread of plausible answers—it rewards placing **100% of the probability mass** on whichever single response scores highest on the reward model.

Consequently, RLHF turns a smooth probabilistic density estimator into a **peaked policy** exhibiting two linked pathologies:

- **Lost calibration:** In the **GPT-4 Technical Report (2023)** on MMLU, the pretrained base model achieved an **Expected Calibration Error (ECE) of 0.007** (nearly perfect calibration). After PPO, ECE degraded more than $10\times$ to **0.074**.
- **Entropy collapse (mode collapse):** RLHF-tuned models (such as historical `text-davinci-003`) exhibit token-level output entropy spiking sharply near zero compared to pretrained base models—repeating stereotyped phrasing, stock openings, and narrow reasoning paths.

> **DEFINITION**
> **Expected Calibration Error (ECE)** measures the weighted average absolute gap between a model's predicted confidence and its actual observed accuracy across probability bins; **mode collapse** in RLHF is the sharp concentration of a policy's probability mass onto a tiny subset of high-reward outputs, driving output entropy toward zero.

The calibration curve for a 52B model on 5-shot MMLU—plotting **Observed Accuracy** (vertical axis, 0.0 to 1.0) against **Predicted Probability** (horizontal axis, 0.0 to 1.0)—makes the distortion visible:

- A **dashed diagonal line** from $(0, 0)$ to $(1, 1)$ represents **perfect calibration** (observed accuracy equals predicted probability everywhere).
- **RLHF at temperature $T = 1$ (blue curve):** Forms a heavily bowed, overconfident S-curve sagging far *below* the diagonal across the middle and upper range. For instance, when the model predicts roughly 0.80 probability, observed accuracy sits near only 0.55—the policy is systematically overconfident because reward optimization sharpened its logits.
- **RLHF at temperature $T = 2.5$ (red curve):** Dividing the model's logits by $T = 2.5$ before softmax flattens the peaked distribution so the red curve tracks right along the dashed diagonal from $(0, 0)$ to $(1, 1)$.

Temperature scaling thus partially restores scalar probability calibration—yet it does **not** undo mode collapse: scaling logits by a uniform scalar preserves token ordering and cannot resurrect creative phrasing or alternative solution modes whose relative logits were crushed during RLHF.

> **COMMON MISCONCEPTION**
> You might think that training a much larger reward model eliminates reward overoptimization, or that applying post-hoc temperature scaling ($T = 2.5$) after RLHF fully restores the output diversity lost to mode collapse. Both break: Gao et al. proved that even a 3B reward model still exhibits gold-reward collapse once KL divergence grows large enough, and temperature scaling only re-calibrates existing softmax probabilities without recovering probability mass on diverse modes that RLHF suppressed.

> **WORKED EXAMPLE**
> **Computing Expected Calibration Error (ECE) before RLHF, after PPO sharpening, and after temperature scaling**

Setup: Consider a 300-question binary evaluation split evenly into two confidence bins ($n_1 = n_2 = 150$ questions each, so each bin has weight $150/300 = 0.5$):

- Bin 1 true accuracy: $\text{acc}_1 = 0.60$.
- Bin 2 true accuracy: $\text{acc}_2 = 0.89$.

Recall that $\text{ECE} = \sum_b \frac{n_b}{N} |\bar{p}_b - \text{acc}_b|$.

Step 1 — Pretrained base model (well-calibrated logits): Suppose the base model produces average logit gaps $\Delta z_1 = \ln(0.60/0.40) = 0.405$ in Bin 1 and $\Delta z_2 = \ln(0.89/0.11) = 2.091$ in Bin 2, yielding $\bar{p}_1 = \sigma(0.405) = 0.600$ and $\bar{p}_2 = \sigma(2.091) = 0.890$ (or say 0.900 with slight finite-sample noise).

- With $\bar{p}_1 = 0.600$ and $\bar{p}_2 = 0.900$:

$$\text{ECE}_{\text{base}} = 0.5\,|0.600 - 0.600| + 0.5\,|0.900 - 0.890| = 0.005$$

Step 2 — After PPO sharpening ($T = 1$, logits amplified by $2.5\times$ without changing which answer is ranked first):

- Bin 1: $\Delta z_1^{\text{RL}} = 2.5 \times 0.405 = 1.014 \implies \bar{p}_1 = \sigma(1.014) = 0.734$, while $\text{acc}_1 = 0.600$.
- Bin 2: $\Delta z_2^{\text{RL}} = 2.5 \times 2.091 = 5.227 \implies \bar{p}_2 = \sigma(5.227) = 0.995$, while $\text{acc}_2 = 0.890$.
- Compute ECE:

  $$\text{ECE}_{\text{RL},T=1} = 0.5\,|0.734 - 0.600| + 0.5\,|0.995 - 0.890| = 0.5(0.134) + 0.5(0.105) =$$

  Calibration error jumped more than $20\times$ purely from overconfident logit sharpening—mirroring GPT-4's $0.007 \to 0.074$ MMLU degradation.

Step 3 — Post-hoc temperature scaling ($T = 2.5$): Divide the RLHF logits by $T = 2.5$:

- Bin 1: $\sigma(1.014/2.5) = \sigma(0.405) = 0.600$.
- Bin 2: $\sigma(5.227/2.5) = \sigma(2.091) = 0.890$.
- $\text{ECE}_{\text{RL},T=2.5} = 0.5|0.600 - 0.600| + 0.5|0.890 - 0.890| = 0.000$.

Interpretation: Temperature scaling accurately repairs scalar confidence calibration on fixed choices, yet dividing by a constant T leaves every rank order identical and cannot restore generation diversity lost to mode collapse across open-ended sequences.

Putting the Stages Together: The Late-2026 Post-Training Map

Every stage we covered in this lecture exists specifically to fix the bottleneck or risk of the stage before it. Stacking them deliberately gives the complete late-2026 map:

Stage	Data / Signal	What It Buys You	Primary Risk
Mid-Training	Curated web, code, math, long-context, and synthetic mix	Long context (up to 1M tokens), coding/reasoning density, early agentic skill	Distribution drift if bulk pretraining mix drops too fast
SFT	Multi-turn demonstrations & tool trajectories (with loss masking)	Format, chat/agent interface, safety baselines, RL cold start	Hallucination on unknown facts; capped by teacher ceiling
Offline Pref (DPO / SimPO)	Static pairwise preferences (x, y_w, y_l)	Simple, accessible open/academic alignment without RM or rollouts	Off-policy distribution shift; length bias; β sensitivity
Online RL + MOPD *(2026 frontier)*	Verifiers (RLVR), generative rubric judges + verbosity gate	Surpasses human demos; merges 9–40+ domain experts cleanly	Reward hacking, overoptimization, entropy / mode collapse

KEY IDEA

- Reward overoptimization (Goodhart's Law) causes true gold quality to peak and collapse as KL divergence grows across PPO, Best-of-n, and Expert Iteration—even with large reward models—unless backed by noiseless verifiers or constrained judges.
- Reward maximization also causes mode collapse and $>10\times$ worse probability calibration (e.g., GPT-4 MMLU ECE jumping from 0.007 to 0.074); temperature scaling partially repairs calibration, while modern 2026 stacks pair mid-training, masked SFT, verifiable/rubric-gated online RL, and MOPD to balance every stage's failure mode.

With the conceptual foundations of mid-training, SFT, preference data, generative judges, DPO, and RLHF failure modes established, one major question remains open: how do online RL algorithms—specifically **PPO**, **GRPO**, and **RLVR**—actually operate in detail, how does a concrete agentic mid-training recipe like **Qwen3-Coder-Next** work end to end, and exactly *why* did online RL plus multi-teacher on-policy distillation replace offline DPO at the frontier? We answer those questions in Lecture 16.

✎ Check Your Understanding

1. **Yes or No:** Does replacing PPO with Best-of-n sampling or Expert Iteration eliminate reward overoptimization when using a learned scalar reward model? Briefly explain why.
   Answer: No. Dubois et al. (2023) showed that reward overoptimization is algorithm-agnostic—it occurs across PPO, Best-of-n, and Expert Iteration whenever optimization pushes hard against an imperfect proxy reward model, disappearing only with a noiseless oracle judge.
2. **What would happen if…** you applied temperature scaling ($T = 2.5$) to an RLHF-trained model's logits on MMLU? State one metric that improves and one limitation that remains.
   Answer: Probability calibration improves markedly (lowering ECE so predicted probabilities align closely with observed accuracy), but output diversity remains reduced because uniform temperature scaling cannot undo the mode collapse induced during RLHF.