Post-Training Foundations

Lecture 3 Notes

CDSS 94: Building Thoughtful AI Systems (Spring 2026)

Contents

1 The Big Idea: Prediction is Compression

1.1 Motivation

The central thesis of this lecture is a simple but deep observation:

> *Prediction is compression. Compression requires understanding. When you train a model to predict the next token, you're training it to understand.*

An optimal predictor is an optimal compressor. Consider the sentence:

"The cat sat on the _____"

To predict the next word well, one needs to know about rhyme, children's literature, and feline behavior. To predict well, one must *model the world*.

1.2 Kolmogorov Complexity

Definition 1.1: Kolmogorov Complexity

The **Kolmogorov complexity** of a string x, denoted $K(x)$, is the length of the shortest program (in some fixed universal language) that produces x:

$$K(x) = \min\{|p| : U(p) = x\}$$

where U is a fixed universal Turing machine and $|p|$ denotes the length of program p.

This gives us the theoretical limit of compression. We can organize types of knowledge along a spectrum of compressibility:

Easy to Compress (Low K)	**Hard to Compress (High K)**
Math, Logic, Code	Emotional Nuance
Grammar, Syntax	Creative Taste
Factual Knowledge	Individual Voice
Rule-based, compressible	*Per-instance, incompressible*

Remark 1.2

Models learn the left column first. This is a direct consequence of the training objective: the next-token prediction loss rewards learning compressible patterns before incompressible ones. The right column is where post-training gets hard.

1.3 Consequences of the Compression View

Three key consequences follow from the prediction-as-compression framework:

(i) **Scale works.** More data + more parameters = more compression = more understanding. The model is not explicitly taught physics; it *discovers* physics because physics helps predict.

(ii) **Capabilities emerge in order.** The model learns the most compressible patterns first. Emergence looks sudden because compression machinery accumulates gradually, then passes a threshold.

(iii) **There is a ceiling.** If something is inherently incompressible (high Kolmogorov complexity per instance), no amount of scale helps. This wall may be where the most distinctly human capabilities live.

1.4 What is Post-Training?

Definition 1.3: Pre-training vs. Post-training

- **Pre-training:** Compress everything. Learn the structure of all text. Become a simulator of the internet.
- **Post-training:** Compress a specific subset. Carve out the behavioral region you actually want.

The post-training landscape includes:

Method	Description
SFT	Supervised Fine-Tuning
RLHF	PPO + variants
DPO	Direct Preference Optimization
CAI	Constitutional AI (RLAIF)
RL on Verifiable Rewards	Math/Code verification
GRPO	Group Relative Policy Optimization

Each method is a different answer to the question: *how do you compress human values into a neural network?*

2 Supervised Fine-Tuning (SFT)

2.1 The SFT Objective

SFT is the simplest form of post-training. We train on (prompt, ideal_response) pairs using standard next-token prediction, but on curated data.

Definition 2.1: SFT Loss

Given a prompt x and target response $y = (y_1, \ldots, y_T)$, the SFT loss is:

$$\mathcal{L}_{\text{SFT}} = -\mathbb{E}\left[\sum_{t=1}^{T} \log \pi_\theta(y_t \mid x, y_{<t})\right]$$

where the loss is computed **only over assistant tokens** (not the prompt). This is forward KL minimization from the data distribution to the model.

Remark 2.2

SFT does not teach new knowledge. It shifts the distribution over behaviors. The model already *can* be helpful; SFT tells it that it *should* be.

2.2 Quality $\gg$ Quantity

Typical SFT datasets contain 10K–100K examples, sometimes as few as 1K.

Example 2.3: LIMA

The LIMA paper showed that 1,000 high-quality examples could produce a strong chat model. The key insight is that **prompt distribution matters more than response quality**. The craft of SFT is the craft of dataset curation.

2.3 Common Failure Modes

(a) **Catastrophic forgetting.** Updates overwrite directions important for pretrained capabilities.

- Fix: low learning rate (10^{-5} to 5×10^{-6}), 1–2 epochs max, mix in pretrain data.

(b) **The 1-epoch phenomenon.** Training >1–2 epochs usually hurts. The model memorizes specific responses rather than learning the distribution.

(c) **Format overfitting.** The model learns the *shape* of good responses without the substance. It *looks* helpful rather than *being* helpful.

(d) **Sycophancy.** If the training data always agrees with the user, agreement becomes the default behavior.

3 KL Divergence

KL divergence is arguably *the most important quantity in post-training*. Every post-training method either explicitly or implicitly manages a KL budget.

3.1 Definition and Properties

Definition 3.1: KL Divergence

The **Kullback–Leibler divergence** from distribution P to distribution Q is:

$$\mathrm{KL}(P\|Q) = \sum_x P(x) \cdot \log \frac{P(x)}{Q(x)}$$

Key properties:

(i) $\mathrm{KL}(P\|Q) \geq 0$ always (Gibbs' inequality), with equality iff $P = Q$.

(ii) $\mathrm{KL}(P\|Q) \neq \mathrm{KL}(Q\|P)$ in general — KL divergence is **not symmetric**. This asymmetry matters enormously.

(iii) Interpretation: $\mathrm{KL}(P\|Q)$ is the *extra bits* needed when using distribution Q to encode data actually drawn from P.

3.2 KL is Everywhere

Method	**Where KL Appears**
SFT	Cross-entropy = forward KL from data to model
Distillation	KL from teacher to student
RLHF/PPO	KL penalty: keep policy near reference
DPO	KL constraint baked into derivation (β controls it)
Best-of-N	Effective KL cost $\approx \log(N)$

KL is the *currency* of post-training. Every method spends a KL budget.

3.3 Forward vs. Reverse KL

Definition 3.2: Forward KL

$$\mathrm{KL}(P_{\text{data}}\|\pi_{\text{model}})$$

This is what SFT minimizes. The model is penalized for **missing any mode** of the data. It is **mean-seeking** and conservative—it "covers everything." As a result, SFT models tend to sound generic and averaged.

Definition 3.3: Reverse KL

$$\mathrm{KL}(\pi_{\text{model}} \| P_{\text{target}})$$

This is what RLHF approximates. The model is penalized for **putting mass outside** the target. It is **mode-seeking** and sharp—it "picks the best." RL-trained models tend to sound decisive.

Remark 3.4

The standard pipeline uses both in sequence:

SFT (forward KL) $\longrightarrow$ broad coverage first $\longrightarrow$ RLHF (reverse KL) $\longrightarrow$ sharpen to the good stuff

3.4 The KL Budget

Think of the KL budget as: *how many bits of behavioral change can you afford?*

Theorem 3.5: Gao et al., 2022 — informal

As KL increases from the reference model, true quality first improves, then peaks, then decreases. Beyond the peak, the model overfits to the reward model—exploiting its compression artifacts.

Remark 3.6

Rules of thumb:

- Alarm bells: KL $>$ 10–20 nats/token $\Rightarrow$ model has drifted too far.
- The parameter β directly controls the exchange rate between reward and KL.

4 Distillation

Distillation is the process of compressing a large (teacher) model into a smaller (student) model.

4.1 The Distillation Loss

Definition 4.1: Distillation Objective

$$\mathcal{L} = \alpha \cdot \text{CE}(y_{\text{hard}},\ \text{student}) + (1-\alpha) \cdot \tau^2 \cdot \text{KL}\big(\text{teacher}_{\text{soft}} \| \text{student}_{\text{soft}}\big)$$

where τ is the temperature controlling the softness of the probability distribution. Higher τ reveals more structure in the teacher's uncertainty.

The key asymmetry in what transfers:

Transfers well (low K)	**Transfers poorly (high K)**
Style, tone, format	Multi-step reasoning
Simple behavioral rules	Edge cases and nuance
	Rare capabilities

Remark 4.2: The Imitation Gap

Distilled models *sound* good but fail on hard problems. They learn the *style* of intelligence before the *substance* of intelligence.

4.2 Chain-of-Thought Distillation

A powerful variant: don't just distill the answer—distill the *thinking*.
Step 1: Train a large model with RL to produce extended chain-of-thought (CoT).
Step 2: Distill the CoT traces into smaller models via SFT.
Step 3: The student learns the *reasoning pattern*, not just the final answer.

Example 4.3: R1-Distill

R1-Distill-Qwen-32B matches o1-mini on many benchmarks. This works because small models struggle to discover reasoning strategies via RL (sparse reward, vast action space). Distillation gives them the strategy for free. Analogy: *it is easier to learn a proof by reading it than to discover it yourself.*

5 Learning from Human Preferences

5.1 The Preference Problem

SFT teaches what a good response looks like, but what if you *can't write* the ideal response—you can only say which of two responses is better?

Remark 5.1

Three key observations motivate preference-based learning:

1. Humans are better at *comparing* than *generating*.
2. Many capabilities are easier to evaluate than demonstrate.
3. "I know it when I see it"—preferences exist even when explicit rules don't.

The pipeline is: pairwise judgments $\longrightarrow$ trained behavior.

5.2 Reward Modeling

Definition 5.2: Bradley–Terry Model

We model human preferences via the Bradley–Terry model:

$$P(y_w \succ y_l \mid x) = \sigma\big(r(x, y_w) - r(x, y_l)\big)$$

where σ is the sigmoid function and $r(x, y)$ is a scalar reward learned by a neural network.

Architecture: Take a pre-trained LLM, remove the language modeling head, add a linear projection to a scalar reward.

Definition 5.3: Reward Model Loss

$$\mathcal{L}_{\text{RM}} = -\mathbb{E}\left[\log \sigma\big(r(y_w) - r(y_l)\big)\right]$$

Remark 5.4

This is an extreme compression: millions of nuanced human judgments $\rightarrow$ a single scalar per response.

5.3 Reward Model Failures

The reward model is a *lossy compressor*. Common biases:

(a) **Length bias:** Longer responses score higher.
(b) **Format bias:** Markdown, bullets, headers $\rightarrow$ higher scores.
(c) **Sycophancy bias:** Responses that agree with the user score higher.
(d) **Confidence bias:** Confident-sounding but wrong beats hedged but correct.

> *"When a measure becomes a target, it ceases to be a good measure." — Goodhart's Law*

In the compression framework: $K(\text{true human preferences}) \gg K(\text{reward model})$. The gap between these is where **reward hacking** lives.

6 RLHF with PPO

6.1 The RLHF Objective

Definition 6.1: RLHF Objective

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

where:

- $r(x,y)$ pushes toward high reward ("be better"),
- $\beta \cdot \mathrm{KL}$ keeps the policy close to the reference ("don't break"),
- β is the exchange rate—the KL budget—the compression constraint.

6.2 The Optimal Solution

Theorem 6.2: Optimal RLHF Policy

The optimal policy under the KL-constrained objective is a Boltzmann distribution:

$$\pi^*(y \mid x) \propto \pi_{\text{ref}}(y \mid x) \cdot \exp\left(\frac{r(x,y)}{\beta}\right)$$

Remark 6.3

Limiting behavior:

- $\beta \to 0$: Pure reward maximization (dangerous—reward hacking).
- $\beta \to \infty$: Stay at reference (useless—no improvement).

We cannot compute π^* directly, so we use PPO to approximate it.

6.3 The PPO Loop

The PPO training loop proceeds as follows:

1. Sample prompts from dataset.
2. Generate completions from policy π_θ.
3. Score with reward model r_ϕ.
4. Compute advantages.
5. Update policy with clipped gradient.
6. Repeat.

The generation step is the bottleneck (autoregressive inference is slow). Four models must be held in memory simultaneously:

	Model	Status
1.	Policy π_θ	Being trained
2.	Reference π_{ref}	Frozen (SFT model)
3.	Reward model r_ϕ	Frozen
4.	Value function V_ψ	Being trained

For a 70B model: $4 \times 70\text{B} = 280\text{B}$ parameters in GPU memory.

Definition 6.4: PPO Clipped Objective

$$\mathcal{L} = \min\Big(\text{ratio} \cdot \hat{A},\ \text{clip}(\text{ratio},\ 1-\varepsilon,\ 1+\varepsilon) \cdot \hat{A}\Big)$$

where ratio $= \pi_\theta(y_t)/\pi_{\theta_{\text{old}}}(y_t)$ and $\varepsilon = 0.2$ is typical. Clipping prevents catastrophically large updates.

6.4 Why PPO is Hard

(a) **Hyperparameter sensitivity:** β (KL coefficient), learning rate (10^{-6} to 5×10^{-7}), PPO epochs, batch size, reward normalization...

(b) **Infrastructure cost:** 4 models in memory, generation bottleneck, training instability—can collapse suddenly.

These difficulties motivated the search for simpler alternatives.

7 Direct Preference Optimization (DPO)

7.1 The Key Insight

What if we could skip the reward model entirely? Starting from the optimal RLHF solution, we can express the reward as:

$$r(x, y) = \beta \cdot \log \frac{\pi^*(y \mid x)}{\pi_{\text{ref}}(y \mid x)} + \text{const}$$

Substituting into the Bradley–Terry preference model, the constant cancels:

Definition 7.1: DPO Loss

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

Remark 7.2

No reward model. No RL loop. No value function. DPO reduces preference optimization to a classification-like loss.

7.2 What DPO Does

The gradient of the DPO loss has the form:

$$\nabla \mathcal{L} \approx -w \cdot [\nabla \log \pi_\theta(y_w) - \nabla \log \pi_\theta(y_l)]$$

where the weight w is large when the model disagrees with the preference and small when it already agrees.

Key properties:

(i) **Contrastive:** Simultaneously pushes up preferred and pushes down dispreferred.
(ii) **Adaptive:** Focuses on hard examples automatically (implicit curriculum learning).
(iii) **KL-constrained:** The β parameter controls distance from reference.

The **implicit reward** learned by DPO is:

$$r_{\text{DPO}}(x, y) = \beta \cdot \log \frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)}$$

DPO doesn't *learn* a reward model—it *becomes* the reward model.

7.3 DPO Failure Modes

(a) **The offline problem:** As the policy drifts, the fixed preference pairs become less relevant. Fix: online/iterative DPO.
(b) **Mode collapse:** The model can satisfy preferences by making *both* responses unlikely, but y_l *more* unlikely.
(c) β **sensitivity:** Too low $\to$ collapse; too high $\to$ barely moves. Typical range: 0.1–0.5.
(d) **Quality gap:** Works best when y_w and y_l are close in quality.

7.4 The DPO Family

Method	Key Change
IPO	Squared loss—avoids overfitting to deterministic preferences
KTO	Works with single ratings (↑/↓)—much easier to collect
ORPO	No reference model—combines SFT and preference optimization
SimPO	Average log-prob as implicit reward—more robust to length bias
Online DPO	Regenerate pairs from current policy—fixes distribution shift

8 Constitutional AI (CAI)

8.1 The CAI Pipeline

The idea: write down principles and have the model apply them.

Remark 8.1

In the compression framework: $K(\text{constitution}) \ll K(\text{all individual preference judgments})$.

CAI has two phases:
Phase 1 (Supervised):

$$\text{Generate} \to \text{Critique against principle} \to \text{Revise} \to \text{Repeat} \to \text{SFT on final response}$$

Phase 2 (Reinforcement):

$$\text{Generate pairs} \to \text{Model judges which follows constitution} \to \text{Train RM} \to \text{RLHF}$$

8.2 When CAI Works and Breaks

- **Works when:** Human preferences are compressible by principles. "Don't help make weapons" is a simple rule (low K). Safety behaviors are generally rule-compressible.
- **Breaks when:** Preferences are irreducibly complex. "Write something beautiful" cannot be compressed into a principle. The constitution may have gaps, and self-reinforcing errors can arise if the model has systematic biases.

Remark 8.2: The Over-Refusal Problem

Models become too cautious. The constitution compresses "don't be harmful" well, but "still be maximally helpful" poorly.

8.3 Model as Judge

CAI opened the door to using LLMs as evaluators of LLMs.

- Agreement of GPT-4/Claude with expert humans: $\sim$80%.
- Human–human agreement: also $\sim$80%.
- Systematic biases: verbosity bias, position bias, self-preference bias, confidence bias.
- Mitigations: randomize order, use multiple judges, require chain-of-thought before judgment.

9 Best-of-N and Rejection Sampling

9.1 Best-of-N

The simplest form of "RL": generate N responses, pick the best one.

Definition 9.1: Best-of-N

$$y^* = \arg \max_{i \in \{1,\ldots,N\}} r(x, y_i)$$

The effective KL cost grows logarithmically:

N	KL cost (nats)
4	~ 0.64
16	~ 1.83
64	~ 3.18

Remark 9.2

Why Best-of-N is underrated: trivial to implement, no training instability, competitive with PPO at low KL budgets.

9.2 Iterated Rejection Sampling

1. **Round 1:** Generate $N \to$ select best $\to$ SFT on selected.
2. **Round 2:** Generate N from improved model $\to$ select best $\to$ SFT.
3. **Round 3:** ...

Each round, the base distribution improves, so Best-of-N selects from a better pool. This is connected to STaR (Self-Taught Reasoner) and ReST (Reinforcement from Self-Training).

Remark 9.3

Risk: each iteration compounds reward model errors. Works best with **verifiable rewards** where there is no proxy to hack.

9.3 RL on Verifiable Rewards

$$r(x, y) = \begin{cases} 1 & \text{if correct} \\ 0 & \text{if wrong} \end{cases}$$

No learned proxy. No Goodhart's Law. Just: *did you get it right?* The challenge is extremely sparse reward: 0 or 1 at the end of potentially 10,000 tokens.

10 The Reasoning Revolution

10.1 DeepSeek R1

The R1 training pipeline consists of three phases:

1. **Phase 1:** Cold-start SFT on long chain-of-thought examples.
2. **Phase 2:** Large-scale RL with GRPO on verifiable rewards.
3. **Phase 3:** Rejection sampling → SFT → another round of RL.

10.2 GRPO: Group Relative Policy Optimization

GRPO eliminates the need for a separate value function and learned reward model:

PPO requires	**GRPO requires**
Policy	Policy
Reference model	Reference model
Reward model	Verifiable reward (free)
Value function	(nothing—group baseline)
4 models	*2 models*

The advantage is estimated using the group of sampled responses as a baseline, rather than a learned value function.

10.3 Emergent Behaviors

The following behaviors were **not** explicitly trained—they emerged from reward maximization:

(i) **"Aha moments":** The model spontaneously learns to reconsider and backtrack.
(ii) **Self-verification:** Checking its own work before committing.
(iii) **Progressive lengthening:** Reasoning chains get longer as training proceeds.
(iv) **Strategic exploration:** Trying multiple approaches when the first fails.

10.4 Chain-of-Thought as Decompression

Remark 10.1

A direct answer like "42" is *compressed*—all reasoning is hidden in the model weights. A chain-of-thought answer like "First, I note that... then... so 42" is *decompressed*—each step is simple enough to execute autoregressively.
The model learns a **meta-compressor**: a program that generates variable-length decompression programs.

10.5 Process Reward Models

Instead of scoring only the final answer, score each step.

Definition 10.2: Process Reward Models

- **Human annotation:** Experts label each reasoning step as correct/incorrect.
- **Monte Carlo estimation:** From each step, generate K completions. The step score equals the fraction reaching the correct answer.
- **At training:** Weight policy gradients by step-level rewards.
- **At inference:** Beam search guided by step scores.

Remark 10.3

A good reasoning step *reduces the remaining Kolmogorov complexity* of the problem.

11 Multi-Turn and Agentic Post-Training

11.1 The Multi-Turn Problem

Everything discussed so far has been single-turn. Real deployment involves multi-turn conversations, tool use, and branching decisions.

Key challenges:

(i) **Delayed reward:** A bad question in turn 2 might only fail in turn 5.
(ii) **Credit assignment:** Across turns is extremely difficult.
(iii) **Massive action space:** Vocabulary $\times$ sequence length $\times$ turns.
(iv) **User simulation:** Required for training rollouts.

11.2 Tool Use Post-Training

The model is learning *policies*, not just responses. Consider a tool call:

```
<tool_call>search("quantum entanglement")</tool_call>
-> [tool returns result]
-> model continues conditioned on result
```

Remark 11.1

Key principles for tool-use training:

- Reward the *outcome*, not the tool call itself.
- Exploration challenge: if the model never tries a tool, it never gets rewarded.
- Bootstrap with SFT, then optimize with RL.

12 Evaluation

12.1 The Eval Crisis

Benchmark	Limitation
MT-Bench	GPT-4 as judge—ceiling effects
AlpacaEval	Biased toward GPT-4 preferences
Chatbot Arena	English tech questions overrepresented
GPQA, MATH	Verifiable but saturating

Once a benchmark becomes a target, it stops being a good measure. Goodhart's Law again.

12.2 Evaluating Subjective Capabilities

This is the hardest and most important frontier. Emotional intelligence, creativity, narrative ability, and taste all lack ground truth.

- Rubric-based human evaluation with detailed scoring.
- Pairwise comparison by domain experts.
- Long-form interaction: does quality hold over 50 turns?

Remark 12.1

Evaluating subjective capabilities is arguably the most important unsolved problem in post-training.

13 The Frontier

13.1 Scalable Oversight

How do you train a model that is better than its teacher?

(i) **Debate:** Two models argue; a human judges.
(ii) **Recursive reward modeling:** Use the model to help evaluate itself.
(iii) **Constitutional AI:** Principles generalize beyond the annotator's ability.
(iv) **Verification $\neq$ Generation:** You can check a proof without discovering it.

No one has solved this. It determines whether post-training scales indefinitely.

13.2 Synthetic Data Flywheels

When can a model improve itself? Three necessary conditions:

1. Verifiable reward (or very good proxy).
2. Sufficient exploration.
3. Remaining capacity for improvement.

Works for math and code (verifiable, bounded exploration). Collapses for open-ended tasks where verification itself is the hard part.

13.3 The Subjective Intelligence Gap

Current post-training optimizes for *consensus* preferences—a mean-field approximation.

But taste, emotional depth, and creative vision are *anti-consensus*:

- Current methods compress preferences into averages.
- Averages destroy what makes subjective capabilities valuable.
- You can't compress taste into rules (high K per instance).
- You can't average taste across people (that gives consensus, not taste).

Possible directions: personal models (compress one person's taste), creative RL loops (generate $\rightarrow$ react $\rightarrow$ adjust), richer feedback (structured critique instead of thumbs up/down).

13.4 The Scaling Question

Pre-training scales predictably. Does post-training?

Might scale	**Might not scale**
RL on verifiable rewards	Preference data beyond a threshold
Data quality at fixed quantity	Subjective capability training
Multi-turn RL	

The honest answer: we don't know yet.

14 Summary: The Compression Frame

Every post-training technique is a different way to compress human values into a neural network:

Method	**What It Compresses**
SFT	Demonstrated behaviors
Reward Modeling	Human preferences (lossy—Goodhart is the cost)
RLHF	Preferences via RL (powerful but unstable)
DPO	Preferences into policy (simpler, offline limits)
Constitutional AI	Preferences via principles (elegant, scalable, gaps in coverage)
RL on Verifiable	Correctness (lossless—limited domains)

> *"You can't compress what you don't understand."*
> *"You can't teach what you can't compress."*

The limits of post-training are the limits of our ability to compress human values into a training signal. Every advance is an advance in that compression. Every failure is a case where we couldn't compress well enough.

The most important capabilities may be the hardest to compress. And that's the work.