Post-Training Methods for LLMs: SFT, Online RL, and DPO

LLM Lab

December 2025

Contents

Post-Training Methods for LLMs

This tutorial provides a comprehensive comparison of the three main post-training approaches for Large Language Models: **Supervised Fine-Tuning (SFT)**, **Online Reinforcement Learning (RL)**, and

Direct Preference Optimization (DPO).

---

1. Overview and Comparison

1.1 The Three Approaches

Method	Principle	Training Signal
SFT	Imitate example responses	(x, y) pairs
Online RL	Maximize reward	Reward function $R(x, y)$
DPO	Prefer good over bad	Preference pairs (x, y_w, y_l)

1.2 Pros and Cons Summary

Method	Pros	Cons
SFT	Simple implementation; great for jump-starting new behavior	May degrade performance on tasks not in training data
Online RL	Better at improving capabilities without degrading unseen tasks	Complex implementation; requires good reward design
DPO	Contrastive training; good at fixing wrong behaviors	May overfit; complexity between SFT and RL

1.3 Current Usage (2024-2025)

All three methods are actively used, often in combination:

Typical Pipeline:

$$\text{Base Model} \xrightarrow{\text{SFT}} \text{Instruct Model} \xrightarrow{\text{DPO or RL}} \text{Aligned Model}$$

Examples:

- DeepSeek-R1: Base → SFT → RLVR (GRPO) → RLHF
- Llama 3: Base → SFT → DPO + PPO
- InstructGPT: Base → SFT → RLHF (PPO)

---

2. Supervised Fine-Tuning (SFT)

2.1 Principle

SFT trains the model to imitate high-quality example responses by maximizing the log-likelihood:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\log \pi_\theta(y|x)\right]$$

where:

- x is the input prompt
- y is the target response
- $\pi_\theta(y|x)$ is the model's probability of generating y given x
- $\mathcal{D}$ is the dataset of (prompt, response) pairs

2.2 Token-Level Formulation

Since $y = (y_1, y_2, \dots, y_T)$ is a sequence:

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

2.3 Use Cases

Use Case	Description
Instruction following	Teach model to follow user instructions
Format learning	JSON output, markdown, specific templates
Domain adaptation	Medical, legal, scientific domains
Distillation	Transfer knowledge from larger model
Cold start	Initial alignment before RL

2.4 Limitations

- **Exposure bias**: Model only sees correct examples during training
- **Catastrophic forgetting**: May lose capabilities on other tasks
- **Data quality ceiling**: Cannot exceed quality of training data

---

3. Online Reinforcement Learning

3.1 Principle

Online RL trains the model to maximize expected reward:

$$\mathcal{J}_{\text{RL}}(\theta) = \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)}\left[R(x, y)\right]$$

where $R(x, y)$ is the reward function (learned or rule-based).

3.2 The RL Pipeline

```
1. Sample prompt x from dataset
2. Generate response y ~  _ (·|x)
3. Compute reward r = R(x, y)
4. Update policy using policy gradient
```

3.3 Policy Gradient with KL Penalty

To prevent the policy from diverging too far from a reference model:

$$\mathcal{J}(\theta) = \mathbb{E}_{x,y\sim\pi_\theta}\left[R(x, y) - \beta \cdot D_{\text{KL}}(\pi_\theta \| \pi_{\text{ref}})\right]$$

where:

- π_{ref} is the reference policy (usually the SFT model)
- β is the KL penalty coefficient
- D_{KL} is the Kullback-Leibler divergence

3.4 How Rewards Update Model Weights

The reward does **not** directly update weights. Instead:

Step 1: Compute Advantage

For GRPO, sample G responses per prompt and compute group-relative advantage:

$$A_i = \frac{r_i - \bar{r}}{\sigma_r}$$

where $\bar{r} = \frac{1}{G}\sum_{j=1}^{G} r_j$ and $\sigma_r = \sqrt{\frac{1}{G}\sum_{j=1}^{G}(r_j - \bar{r})^2}$

Step 2: Policy Gradient

$$\nabla_\theta \mathcal{J} = \mathbb{E}\left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(y_t|x, y_{<t}) \cdot A\right]$$

Step 3: Clipped Update (PPO/GRPO)

$$\mathcal{L}(\theta) = \mathbb{E}\left[\min\left(r_t(\theta)A_t,\ \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)A_t\right)\right]$$

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

Intuition:

- Positive advantage ($A > 0$): Increase probability of those tokens
- Negative advantage ($A < 0$): Decrease probability of those tokens
- Clipping prevents too-large updates

3.5 Types of Reward Functions

Type	Example	Pros	Cons
Learned (RLHF)	Neural reward model	Flexible, captures preferences	Reward hacking, expensive
Verifiable (RLVR)	Math correctness, code tests	Reliable, cheap	Limited domains
Rule-based	Format checking, length	Simple, deterministic	Limited expressiveness

3.6 Use Cases

Use Case	Why RL?
Reasoning (math, code)	Verifiable rewards provide clear signal
Safety alignment	Can optimize for multiple objectives
Agentic behavior	Multi-turn, tool use with environment feedback
Capability improvement	Explore beyond training data

---

4. Direct Preference Optimization (DPO)

4.1 Principle

DPO directly optimizes the policy on preference pairs without training a separate reward model.

Given preference pairs (x, y_w, y_l) where y_w is preferred over y_l:

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

4.2 Notation Table

Symbol	Meaning
x	Input prompt
y_w	Preferred (winning) response
y_l	Rejected (losing) response
π_θ	Policy being trained
π_{ref}	Reference policy (frozen)
β	Temperature parameter
σ	Sigmoid function

4.3 Connection to Reward Modeling

DPO is derived from the RLHF objective. The implicit reward is:

$$r(x, y) = \beta\log\frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)} + \beta\log Z(x)$$

where $Z(x)$ is a partition function (constant for optimization).

Key insight: DPO shows that the optimal policy under Bradley-Terry preferences has a closed-form solution, eliminating the need for:

1. Training a separate reward model
2. Sampling from the policy during training
3. Complex RL optimization

4.4 How DPO Updates Weights

The gradient of the DPO loss:

$$\nabla_\theta\mathcal{L}_{\text{DPO}} = -\beta\mathbb{E}\left[\underbrace{\sigma(\hat{r}_l - \hat{r}_w)}_{\text{weight}}(\nabla_\theta\log\pi_\theta(y_w|x) - \nabla_\theta\log\pi_\theta(y_l|x))\right]$$

where $\hat{r} = \beta\log\frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}$

Intuition:

- Increase probability of preferred response y_w
- Decrease probability of rejected response y_l
- Weight by how “wrong” the model currently is

4.5 Use Cases

Use Case	Why DPO?
Fixing specific behaviors	Targeted preference pairs
Style alignment	Prefer certain tone/format
Safety fine-tuning	Prefer safe over unsafe responses
Quick iteration	Simpler than full RL pipeline

4.6 Limitations

- **Overfitting**: Can overfit to preference dataset
- **No exploration**: Unlike RL, doesn't generate new responses
- **Reference dependence**: Quality depends on reference model

---

5. Reward Function Design

5.1 Why It's Tricky

The slide notes: *"requires good design of reward functions"*

This is challenging because:

$$\text{Reward Model} \neq \text{True Human Preference}$$

The reward model is a **proxy** that can be exploited.

5.2 Examples of Reward Functions

Example 1: Math (Verifiable)

```python
def math_reward(response, ground_truth):
    answer = extract_boxed(response)
    return 1.0 if answer == ground_truth else 0.0
```

- **Pros**: Deterministic, no hacking possible
- **Cons**: Binary signal, no partial credit

Example 2: Code (Verifiable)

```python
def code_reward(response, test_cases):
    code = extract_code(response)
    results = [run_test(code, tc) for tc in test_cases]
    return sum(results) / len(test_cases)
```

- **Pros**: Objective correctness
- **Cons**: Tests may not cover all cases; model may overfit to tests

Example 3: Helpfulness (Learned)

```python
def helpfulness_reward(prompt, response):
    return reward_model.score(prompt, response)
```

- **Pros**: Captures nuanced preferences
- **Cons**: Reward hacking, distribution shift

Example 4: Multi-Objective

```python
def combined_reward(prompt, response):
    helpful = helpfulness_model(prompt, response)
    safe = safety_model(prompt, response)
    concise = -len(response) / 1000  # Penalize length
    return 0.5 * helpful + 0.3 * safe + 0.2 * concise
```

- **Pros**: Balances multiple goals
- **Cons**: Weight tuning is arbitrary; objectives may conflict

5.3 Common Reward Hacking Failures

Failure Mode	Description	Example
Verbosity	Longer = higher reward	Model pads responses
Sycophancy	Agreement = higher reward	Model agrees even when wrong
Format gaming	Structure = higher reward	Excessive bullet points
Keyword stuffing	Certain words score high	Repeating "helpful"
Specification gaming	Achieves metric, not intent	Hardcoded test outputs

5.4 Mitigation Strategies

Strategy	Description
KL penalty	Keep policy close to reference
Reward ensembles	Average multiple reward models
Verifiable rewards	Use when possible (math, code)
Length normalization	Divide reward by response length
Adversarial training	Train reward model on edge cases
Constitutional AI	Use principles, not just rewards

---

6. Summary

6.1 Method Comparison

Aspect	SFT	Online RL	DPO
Training signal	Examples	Rewards	Preferences
Exploration	None	Yes	None
Complexity	Low	High	Medium
Data needed	(x, y)	x + reward	(x, y_w, y_l)
Reward model	No	Yes (or verifiable)	No (implicit)
Best for	Initial alignment	Capability improvement	Behavior correction

6.2 Key Equations

Method	Loss Function
SFT	$-\mathbb{E}[\log \pi_\theta(y \| x)]$
RL (GRPO)	$\mathbb{E}[\min(r_t A_t, \text{clip}(r_t, 1 \pm \varepsilon) A_t)]$

Method	Loss Function
DPO	$-\mathbb{E}[\log \sigma(\beta \log \frac{\pi_\theta(y_w \| x)}{\pi_{\text{ref}}(y_w \| x)} - \beta \log \frac{\pi_\theta(y_l \| x)}{\pi_{\text{ref}}(y_l \| x)})]$

6.3 When to Use What

Scenario	Recommended Method
Starting from base model	SFT first
Have preference pairs	DPO
Verifiable tasks (math, code)	Online RL with RLVR
Need exploration/generalization	Online RL
Quick behavior fixes	DPO
Complex multi-objective alignment	Online RL

---

7. References

1. Ouyang et al. (2022). "Training language models to follow instructions with human feedback" (InstructGPT)
2. Rafailov et al. (2023). "Direct Preference Optimization: Your Language Model is Secretly a Reward Model"
3. Schulman et al. (2017). "Proximal Policy Optimization Algorithms"
4. DeepSeek-AI (2024). "DeepSeekMath: Pushing the Limits of Mathematical Reasoning"
5. DeepSeek-AI (2025). "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning"
6. Christiano et al. (2017). "Deep Reinforcement Learning from Human Preferences"