RLHF Training and Evaluation Methodology

LLM Lab

Contents

1 RLHF Training and Evaluation Methodology

This document explains how Reinforcement Learning from Human Feedback (RLHF) and RLAIF differ fundamentally from supervised learning in their training and evaluation approaches.

Key insight: RLHF does *not* use traditional train/validation/test splits like supervised learning. Instead, it uses procedural reward-based evaluation.

1.1 1. How RL Training and Testing Work in Modern LLM Post-Training

1.1.1 1.1 RL Data Pipeline

Unlike supervised learning with fixed labels, RLHF uses dynamic reward signals:

Training (RL-Train):

`{inputs} → {input, output, reward from RM + verifiers}`

Evaluation (RL-Test):

`{inputs} → {input, output, reward from new RM + verifiers}`

The key difference: rewards are **computed on the fly**, not pre-labeled.

1.1.2 1.2 RL Does Not Use Fixed Labels

RLHF does *not* have ground-truth targets. Instead, the **reward model** (RM) and other verifiers produce a reward signal dynamically:

$$\text{Reward} = \text{RM}(\text{input}, \text{model\_output}) + \text{verifiers}$$

where verifiers check for safety, style, factuality, etc.

Implications:

- RL training & evaluation are **procedural**, not label-based
- You evaluate the model by **recomputing reward**, not comparing to a label
- The reward signal can change based on the reward model used

1.1.3 1.3 RL "Testing" is Off-Policy Evaluation

Instead of a traditional validation split, RL evaluation uses:

1. **A fresh reward model** (not the one used during training)
2. **Held-out prompts** (different input distribution)
3. **External verifiers** (safety, factuality checkers)
4. **Automated or human evaluations**

This is why RL-Test uses:

> reward from **new RM + verifiers**

Goal: Check if the policy improved **under an unbiased evaluator**, not whether it memorized the training reward model.

1.2 2. Why RLHF Doesn't Use Traditional Validation Splits

RLHF doesn't optimize supervised loss, so it doesn't need a standard validation set. But there's a deeper reason:

1.2.1 2.1 The Reward Overfitting Problem

Modern post-training avoids validation splits for RL because of **reward overfitting**.

If you used a fixed validation set, the model could:

- **Overfit the reward model** - Learn to maximize the specific RM's quirks
- **Hack the reward function** - Find exploits in the reward computation

- **Exploit the evaluator** - Game the evaluation metrics
- **Mode collapse** - Collapse into reward-maximizing behaviors that don't generalize

Reusing the same validation prompts would cause the model to optimize *for the validation set specifically*, defeating the purpose.

1.2.2 2.2 The Solution: Separate Distributions

Strategy: Pick a fixed prompt distribution for RL training, but evaluate on a **separate distribution** AND recompute reward with a **fresh RM**.

This is why best practices emphasize:

> **"Keep splits apart from each other."**

Specifically, avoid leakage between:

1. **SFT data** - Supervised fine-tuning examples
2. **Reward model data** - Preference pairs for training the RM
3. **RL prompt data** - Prompts used during RL training
4. **Eval benchmarks** - Final evaluation datasets

Rationale: Each stage should use independent data to prevent the model from exploiting patterns across stages.

1.3 3. Hyperparameter Tuning Without Validation Sets

1.3.1 3.1 The Paradox

RLHF algorithms have many hyperparameters:

- Learning rate
- Clipping parameters (e.g., PPO ϵ)
- KL penalty strength β
- Reward scaling
- Rollout length
- Batch size
- Entropy bonus

Question: If there's no validation set, how are these tuned?

1.3.2 3.2 How Modern RLHF Hyperparameters Are Actually Tuned

1.3.2.1 Method 1: Short-Run Experiments Train for a few steps (minutes), then examine:

- **Reward curves** - Is reward increasing?
- **KL divergence** - Is the model staying close to the reference policy?
- **Entropy** - Is the model maintaining diversity?
- **Mode collapse signals** - Are outputs becoming repetitive?
- **Preference win rate** - How often does the model beat the reference?

This is extremely fast and provides immediate feedback.

1.3.2.2 Method 2: Cross-Check with SFT Validation Set The SFT validation split is sometimes reused *only for sanity checking degradation.*

Important: This is NOT used for RL optimization, only to ensure the model hasn't catastrophically regressed on basic capabilities.

1.3.2.3 Method 3: Human or Automated Spot Checks

When RLHF is unstable, human evaluators do a tiny batch of checks (20–50 prompts) to assess quality.

This is qualitative feedback, not a formal validation metric.

1.3.2.4 Method 4: Use Known Stable Defaults

RLHF hyperparameters are surprisingly **stable across models**. Labs reuse:

Hyperparameter	Typical Range
PPO KL penalty	0.01–0.05
Reward scaling	0.1–1.0
Batch size	1k–16k tokens
Rollout length	512–2048

These defaults work well across different model sizes and tasks.

1.3.2.5 Method 5: Use Reward Models for Hyperparameter Selection

Common trick:

1. Train model under several hyperparameter settings
2. Evaluate using **held-out reward model**
3. Pick the best configuration

Note: Still *no validation set in the classical sense.* The held-out RM serves as a proxy for human preferences.

1.4 4. Summary: Why RLHF is Different

1.4.1 4.1 Key Differences from Supervised Learning

Aspect	Supervised Learning	RLHF
Labels	Fixed ground-truth labels	Dynamic reward signals
Validation	Fixed validation set	Off-policy evaluation with fresh RM
Evaluation	Compare to labels	Recompute rewards
Overfitting Risk	Overfit to training labels	Overfit to reward model
Hyperparameter Tuning	Grid search on validation	Short runs + stable defaults

1.4.2 4.2 Why RLHF Doesn't Use Traditional Validation Splits

Five reasons:

1. **No ground-truth target** to validate against
2. **Reward model is learned** - validation would leak information
3. **RL easily overfits** the reward model
4. **Evaluation uses fresh RMs** and external evaluators
5. **Labs use other signals** (reward curves, KL, human evals) for tuning

1.4.3 4.3 Design Philosophy

This design is **intentional** to avoid:

- **Reward hacking** - Gaming the reward function
- **Data leakage** - Information flowing between training stages
- **Overfitting** - Memorizing validation patterns
- **Mode collapse** - Converging to narrow behaviors

The goal is to train models that generalize to human preferences, not just maximize a specific reward function.

1.5 5. Practical Implications

1.5.1 5.1 For Practitioners

When implementing RLHF:

1. **Keep data splits separate** - Don't reuse data across SFT, RM training, and RL
2. **Use multiple reward models** - Train several RMs and evaluate on held-out ones
3. **Monitor KL divergence** - Ensure the model doesn't drift too far from the reference
4. **Start with stable defaults** - Use known hyperparameter ranges
5. **Do short-run experiments** - Quick iterations to test stability

1.5.2 5.2 For Researchers

Open questions:

- How to better detect reward hacking?
- Can we develop validation metrics that don't leak?
- How to balance exploration vs. exploitation in RL training?
- What's the optimal KL penalty schedule?

1.6 6. Comparison Table: SFT vs RM Training vs RL Training

Stage	Data Type	Objective	Evaluation	Validation
SFT	(input, output) pairs	Maximize $\log p(y \mid x)$	Perplexity on held-out set	Traditional validation split
RM Training	(input, output , output , preference)	Learn preference function	Accuracy on held-out preferences	Traditional validation split
RL Training	(input) $\rightarrow$ generate $\rightarrow$ reward	Maximize expected reward	Off-policy with fresh RM	No traditional validation

Key insight: Only RL training breaks from the traditional train/val/test paradigm.

1.7 7. The RLHF Pipeline

1.7.1 7.1 Stage 1: Supervised Fine-Tuning (SFT)

Data: High-quality (input, output) pairs

Objective:

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

Validation: Traditional held-out set, measure perplexity

1.7.2 7.2 Stage 2: Reward Model Training

Data: Preference pairs (x, y_w, y_l) where y_w is preferred over y_l

Objective (Bradley-Terry model):

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

Validation: Accuracy on held-out preference pairs

1.7.3 7.3 Stage 3: RL Fine-Tuning (RLHF)

Data: Prompts $x \sim \mathcal{D}_{\text{prompts}}$

Objective (PPO):

$$\mathcal{L}_{\text{PPO}} = \mathbb{E}_{x,y} \left[\min\left(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right] - \beta \text{KL}(\pi_\theta \| \pi_{\text{ref}})$$

where: - $r_t(\theta) = \frac{\pi_\theta(y|x)}{\pi_{\text{old}}(y|x)}$ is the probability ratio - A_t is the advantage (reward - baseline) - β is the KL penalty coefficient

Evaluation: Generate on held-out prompts, evaluate with fresh RM + human evals

---

1.8 8. Future Directions

1.8.1 8.1 Alternative Approaches

Direct Preference Optimization (DPO): - Bypasses explicit reward model - Directly optimizes policy from preferences - Simpler pipeline, potentially more stable

Group Relative Policy Optimization (GRPO): - Uses group-based comparisons - More sample-efficient than PPO

Constitutional AI: - Self-critique and revision - Reduces need for human feedback

1.8.2 8.2 Open Research Questions

1. How to detect and prevent reward hacking automatically?
2. Can we develop better off-policy evaluation metrics?
3. What's the optimal balance between SFT and RL?
4. How to scale RLHF to multi-turn conversations?
5. Can we make reward models more robust and generalizable?

1.9 9. Conclusion

RLHF fundamentally differs from supervised learning:

- **No fixed labels** - rewards are computed dynamically
- **No traditional validation** - uses off-policy evaluation with fresh RMs
- **Procedural evaluation** - recompute rewards, don't compare to ground truth
- **Intentional design** - prevents reward hacking and overfitting

Understanding these differences is crucial for: - Implementing RLHF correctly - Debugging training issues - Designing better evaluation protocols - Advancing the field

The key takeaway: **RLHF is not supervised learning with a fancy loss function. It's a fundamentally different paradigm that requires different thinking about data, evaluation, and validation.**

1.10 References

1.10.1 Foundational Papers

1. Christiano, P. F., et al. (2017). Deep reinforcement learning from human feedback. NeurIPS.
2. Stiennon, N., et al. (2020). Learning to summarize from human feedback. NeurIPS.
3. Ouyang, L., et al. (2022). Training language models to follow instructions with human feedback. NeurIPS.

1.10.2 Modern Approaches

4. Rafailov, R., et al. (2023). Direct Preference Optimization: Your Language Model is Secretly a Reward Model. arXiv:2305.18290.
5. Bai, Y., et al. (2022). Constitutional AI: Harmlessness from AI Feedback. arXiv:2212.08073.
6. Schulman, J., et al. (2017). Proximal Policy Optimization Algorithms. arXiv:1707.06347.

1.10.3 Practical Guides

7. OpenAI. (2023). GPT-4 Technical Report. arXiv:2303.08774.
8. Anthropic. (2023). Claude 2 Technical Report.
9. Touvron, H., et al. (2023). Llama 2: Open Foundation and Fine-Tuned Chat Models. arXiv:2307.09288.

Last updated: November 2025