What the fuck is reinforcement learning for LLMs, actually?

Reinforcement learning for Large Language Models is a training paradigm that optimizes language models to maximize rewards rather than simply predict next tokens. (Dipendra Misra +2) At its core, it transforms LLMs from pattern matchers into decision-makers that learn policies for generating text that achieves specific goals.

The fundamental difference from standard training: Instead of minimizing cross-entropy loss on predetermined text, RL for LLMs optimizes expected cumulative reward through trial-and-error generation and feedback. (Substack) (IBM) This enables models to learn behaviors that weren't explicitly demonstrated in training data—like reasoning strategies, tool use, or alignment with human values.

Traditional RL vs LLM RL: The Core Distinctions

The "Never Finished Training" Misconception

You're partially correct about traditional RL's continuous learning nature, but the distinction is more nuanced:

Traditional RL characteristics:

- **Continuous environment interaction**: Agent learns while actively exploring (online learning)
- **Multi-step episodes**: Sequences of state→action→reward→new state
- **Temporal credit assignment**: Rewards propagate backward through time via discount factors
- **Clear episode boundaries**: Games end, robots reach goals, episodes terminate

LLM RL characteristics:

- **Batch learning from generated data**: Model generates text, receives rewards, updates offline
- **Single "episode" per generation**: Each complete response is essentially one episode
- **Token-level sequential decisions**: Each token is an "action" in the RL framework
- **Sparse reward structure**: Typically only the final token receives the "real" reward (ArXiv +2)

Is It Actually Different from Gradient Descent?

This is where your confusion is justified—there's significant debate in the field:

Yes, it's fundamentally different:

- **Stochastic gradient estimation** vs exact gradient computation

- **Policy gradient theorem** enables optimization of non-differentiable reward functions
- **Exploration-exploitation tradeoff** through stochastic policies
- **Value functions** estimate future rewards, not present loss

But also, it's surprisingly similar:

- Direct Preference Optimization (DPO) shows RLHF objectives can be achieved through supervised learning (Stack Exchange +3)
- Both use gradient-based parameter updates
- The KL-constrained RLHF objective mathematically reduces to a specific supervised loss function (Analytics Vidhya)

The verdict: LLM RL uses different mathematical machinery (policy gradients) to solve a different problem (reward maximization), but in practice, some objectives can be achieved through either approach. (Iclr-blogposts)

How the Math Actually Works

RLHF: The Three-Phase Process

Phase 1: Supervised Fine-Tuning (SFT)

```
L_SFT = -E[log π_θ(y|x)]
```

Standard next-token prediction on high-quality demonstrations.

Phase 2: Reward Model Training

```
L_RM = -E[log σ(r_θ(x, y_preferred) - r_θ(x, y_dispreferred))]
```

Learn to predict human preferences using the Bradley-Terry model. (ArXiv +4)

Phase 3: RL Fine-tuning with PPO

```
J(θ) = E[r(x,y)] - β * KL(π_θ||π_ref)
```

Maximize reward while staying close to the reference model. (Amazon +2)

Reward Propagation: Not Just Backpropagation

Key difference: Reward propagation uses the policy gradient estimator: (Stack Exchange) (IBM)

```
∇_θJ(θ) = E[∇_θ log π_θ(a|s) · A(s,a)]
```

This is **fundamentally different** from backpropagation because:

1. **Sampling-based**: Estimates gradients through Monte Carlo sampling (IBM)
2. **Handles non-differentiability**: Works even when rewards aren't differentiable (Substack)
3. **High variance**: Requires variance reduction techniques (baselines, advantage estimation) (Dipendra Misra)
4. **Credit assignment problem**: Must distribute sparse rewards across many tokens (Baeldung +2)

PPO Adaptation for LLMs

The Proximal Policy Optimization algorithm is adapted by: (Johnwlambert)

- Treating each token as an action
- Using KL divergence to prevent "forgetting" pre-training
- Adding value heads to estimate future rewards
- Clipping probability ratios to ensure stable updates (ArXiv +3)

Thinking Models and Chain-of-Thought

Your understanding here is largely correct but incomplete:

What you got right:

- RL teaches models "how to think" by rewarding reasoning processes
- Rewards propagate back through intermediate reasoning tokens
- This is fundamentally about learning reasoning strategies, not memorizing answers

What's missing:

- **Process vs Outcome rewards**: The most effective approach uses step-level feedback (ArXiv +5)
- **Pure RL emergence**: DeepSeek-R1 shows reasoning can emerge from RL alone, without demonstrations (GitHub +2)
- **Efficient algorithms**: GRPO eliminates value networks while maintaining effectiveness (GitHub +3)

DeepSeek's Breakthrough

DeepSeek-R1 validated that sophisticated reasoning behaviors emerge purely from RL: (Sebastianraschka) (GitHub)

- Self-verification and reflection appear without being explicitly taught
- Long chains of thought develop naturally when rewarded for correct answers

- The model learns to "think out loud" as an instrumental goal for accuracy Sebastianraschka ArXiv

Agentic Systems: Beyond Text Generation

RL for agentic systems represents a different paradigm:

Action space transformation:

- From: Selecting next token from vocabulary
- To: Choosing tools, APIs, search queries, navigation actions

Episode structure:

- Variable-length episodes based on task completion
- Real environment interaction with external systems Apple ArXiv
- Sparse rewards only upon successful task completion

Key insight: This is much closer to traditional RL because agents interact with actual environments, not just generate text. Apple

Where Your Understanding Needs Adjustment

1. "RL for LLMs is just standard neural network training"

- Partially true: DPO shows some objectives achievable through supervised learning Stack Exchange +3
- But false in general: Policy gradient methods enable fundamentally different optimization

2. "Traditional RL never finishes training"

- This overstates the difference: Both can have distinct training/deployment phases
- The key difference is online (learning while acting) vs offline (learning from batches) Huggingface Iclr-blogposts

3. "Spreading rewards through intermediate steps"

- You correctly identify this as the credit assignment problem Baeldung +2
- But it's not just "fancy gradient descent"—it requires specialized algorithms for variance reduction

4. "The action-observation-reward loop"

- In LLM RL, this becomes: context→token→(implicit reward 0)→updated context
- Only at sequence end: final token→actual reward→parameter update Yuge Shi Substack

The Bottom Line

Reinforcement learning for LLMs is a genuine adaptation of RL principles to the language domain, with important modifications:

1. **It's real RL**: Uses policy gradients, value functions, and exploration-exploitation tradeoffs Dipendra Misra IBM
2. **But also unique**: Single-episode structure, sparse rewards, and token-level actions create distinct challenges
3. **Serves different purposes**: Alignment (RLHF), reasoning (process rewards), and agency (tool use) Chatgptguide ArXiv
4. **The debate continues**: Whether complex supervised learning can replace RL remains an open question Stack Exchange Iclr-blogposts

The field is rapidly evolving, with innovations like DPO challenging our understanding of when RL is necessary versus when clever supervised learning suffices. Maartengrootendorst +2 What's clear is that RL has unlocked capabilities in LLMs—from human alignment to complex reasoning—that weren't achievable through standard training alone. Aigloballab Deeplearning