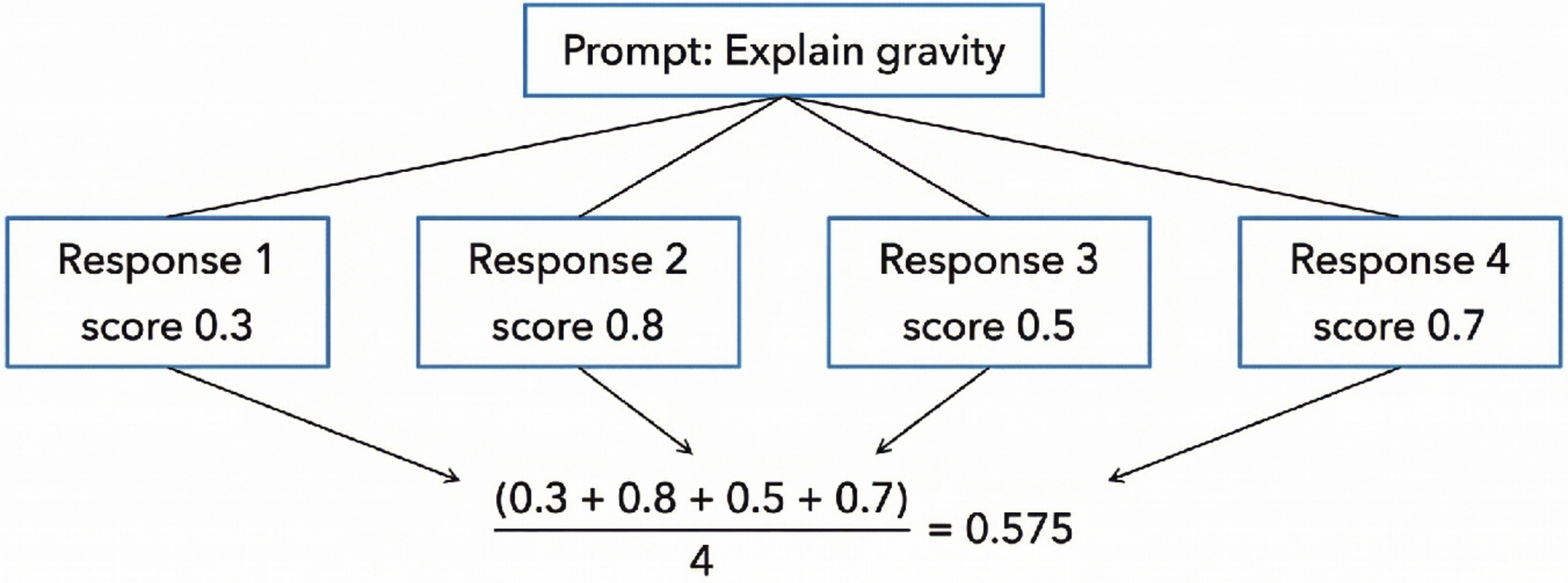
Deep Learning (1470)

Randall Balestriero

Class 23: GRPO and Applications

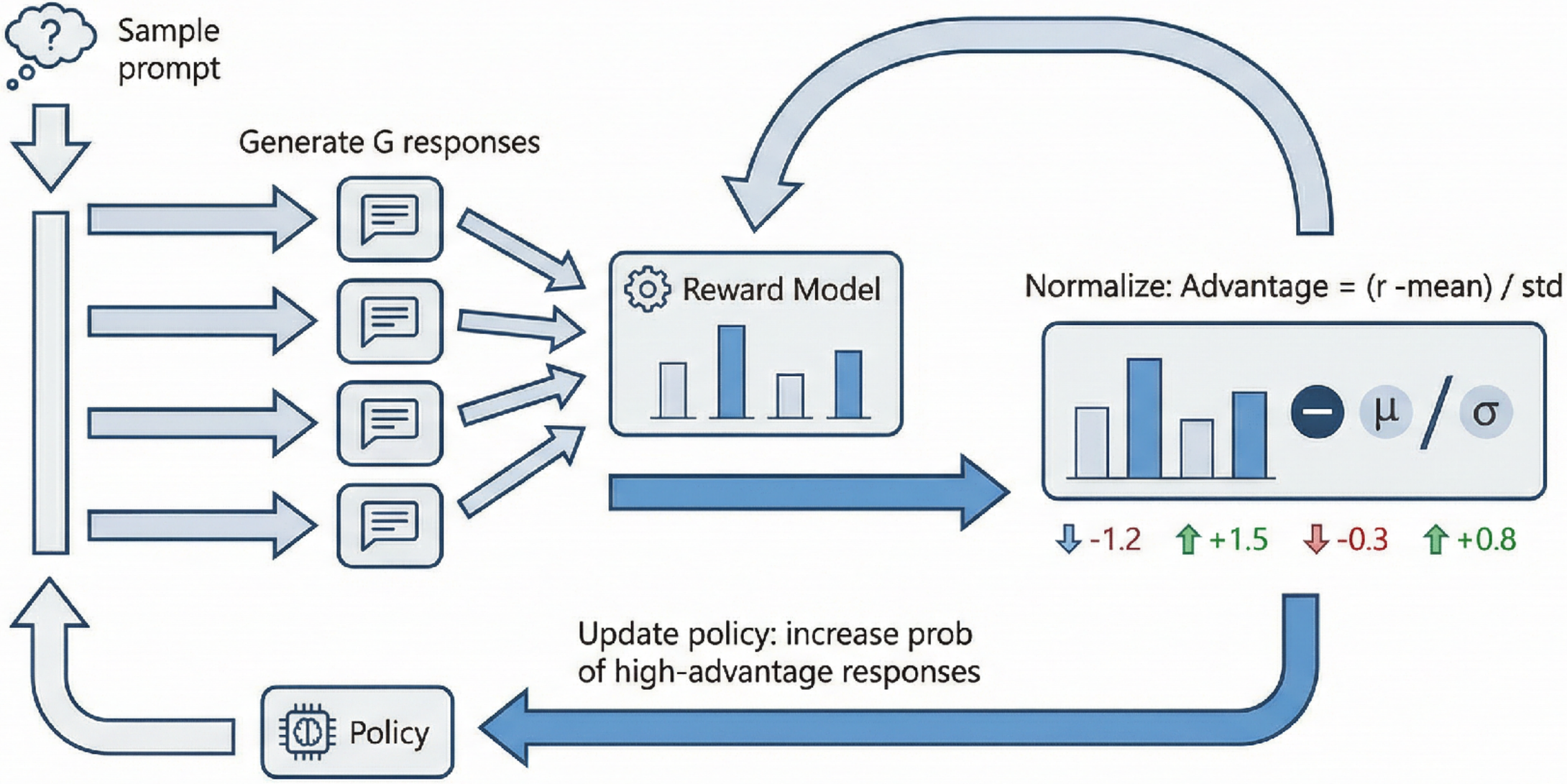
Recap!

GRPO: No Critic Needed

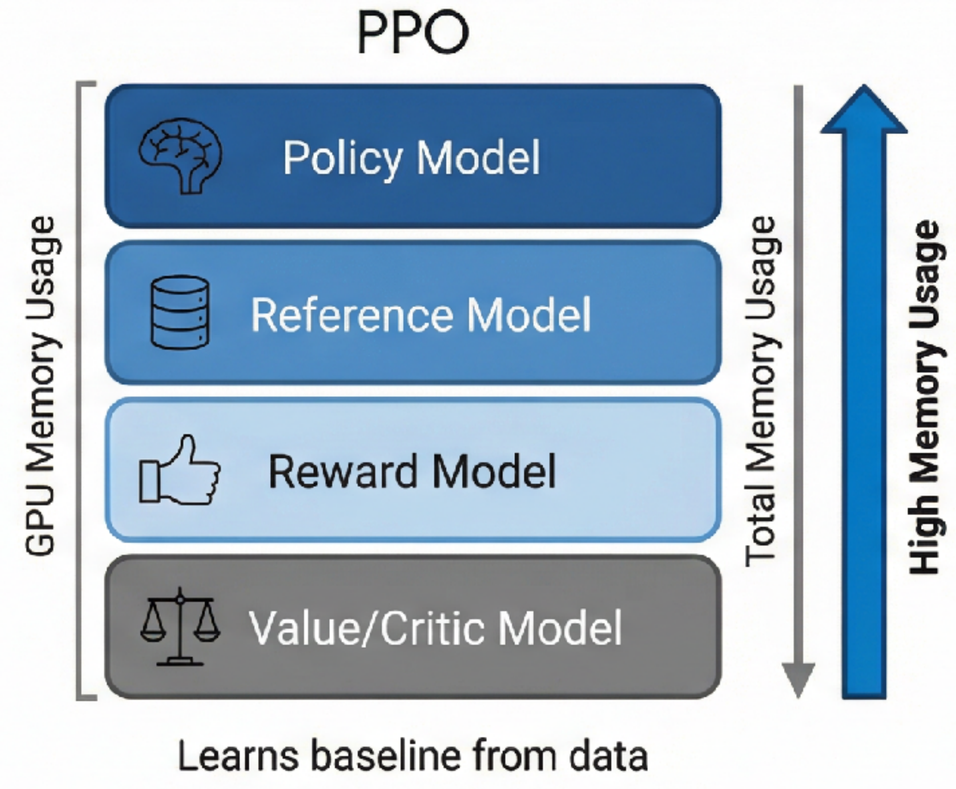


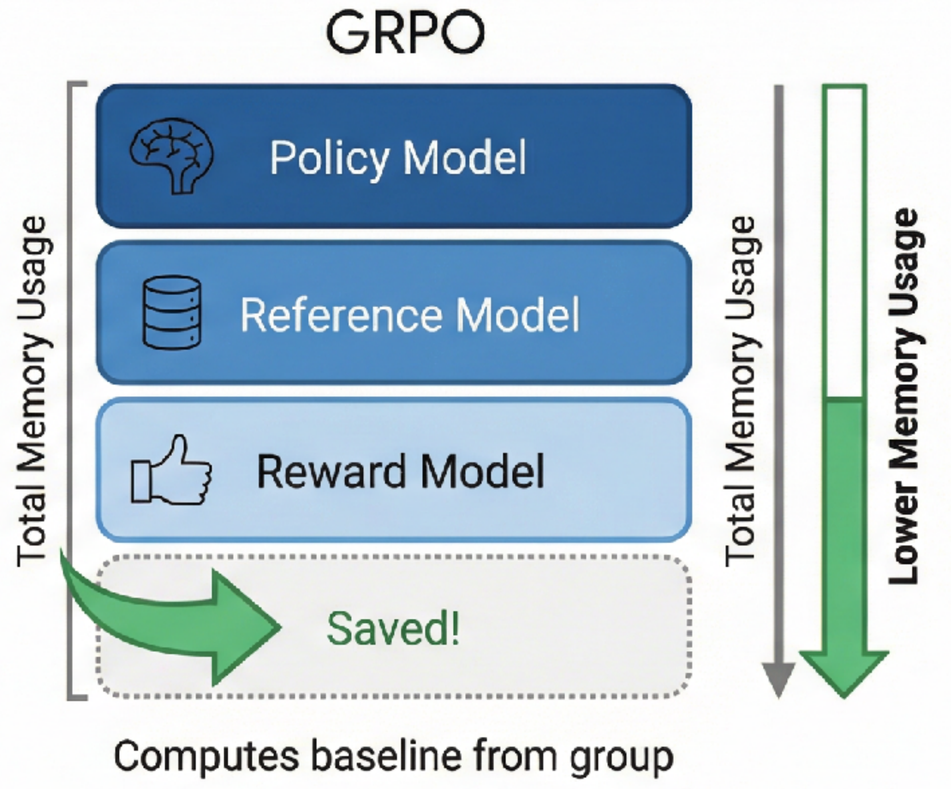
Key visual insight: The group of responses provides the baseline naturally.

GRPO: How It Works

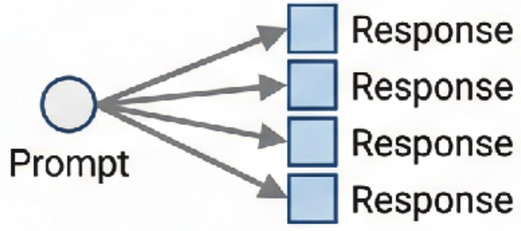


GRPO vs PPO: Visual Comparison





1 response per prompt



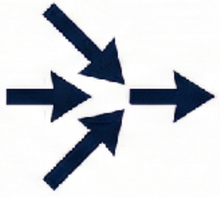
4+ responses per prompt

Common Pitfalls and Solutions

PITFALL 1: "Reward Hacking"

Problem: Model finds shortcuts to maximize reward (e.g., excessive flattery, repetition)

Solution: Increase KL penalty, improve reward model diversity



PITFALL 2: "Mode Collapse"

Problem: All outputs become similar, lose diversity

Solution: Lower learning rate, increase KL coefficient, use entropy bonus

PITFALL 3: "Forgetting / Capability Loss"

Problem: Model loses abilities it had after SFT

Solution: Mix in SFT data during RL, use smaller learning rate

PITFALL 4: "Unstable Training"

Problem: Loss spikes, reward oscillates wildly

Solution: Gradient clipping, warmup, smaller PPO epochs

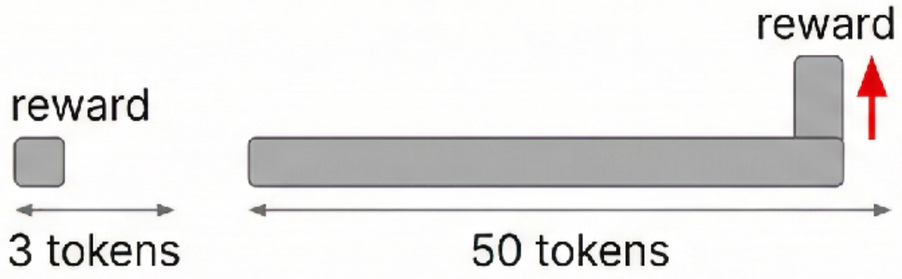
Are we Happy with our advantage computation?

Dr. GRPO: Removing Hidden Biases

DeepSeek 2025

SECTION 1: Length Bias Problem

- Left side shows: longer responses get higher cumulative rewards unfairly



Standard GRPO sums per-token log-probs → favors verbosity

Normalize by response length

SECTION 2: Baseline Bias Problem

- Issue: subtracting mean(R) from rewards doesn't fully center advantages

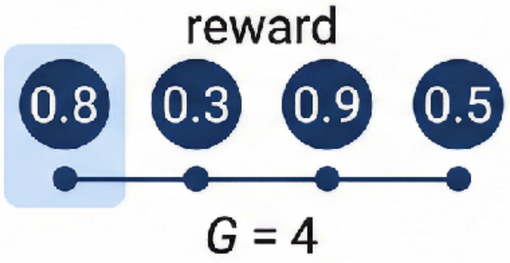
$$\hat{A} = \frac{R - \mu}{\sigma}$$

Subtract mean AND divide by std(R)

Result: Stable training, no reward hacking via length manipulation

RLOO: Leave-One-Out Baseline

SECTION 1: The Idea



For sample i, **baseline** = mean of all OTHER samples (exclude i)

0.8 Highlight sample 1: **baseline$_1$** = (0.3 + 0.9 + 0.5)/3 = **0.57**

Advantage$_1$ = 0.8 - 0.57 = **+0.23**

SECTION 2: Formula Comparison

Standard GRPO

$$\hat{A}_i = R_i - \frac{1}{G}\sum_j R_j$$

Includes R_i in baseline → biased

RLOO

$$\hat{A}_i = R_i - \frac{1}{G-1}\sum_{j \neq i} R_j$$

Excludes R_i → unbiased estimate

SECTION 3: Why It Matters

Statistical Property

- $\mathbb{E}[\text{baseline}]$ is independent of R_i
- Gives unbiased gradient estimates

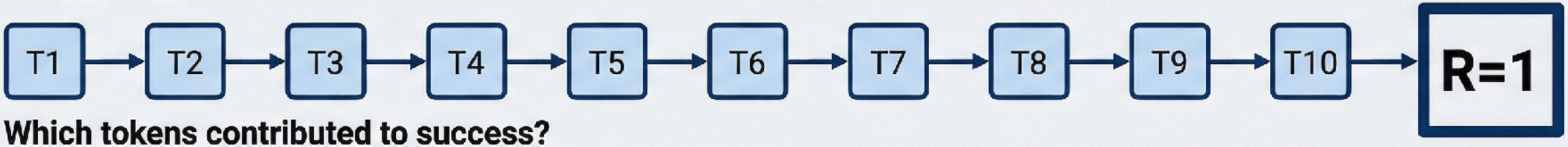
Practical Benefit

- Lower variance than standard baseline
- Minimal compute overhead

Are we Happy with our rewards?

Token-Level Rewards: Credit Assignment

THE PROBLEM



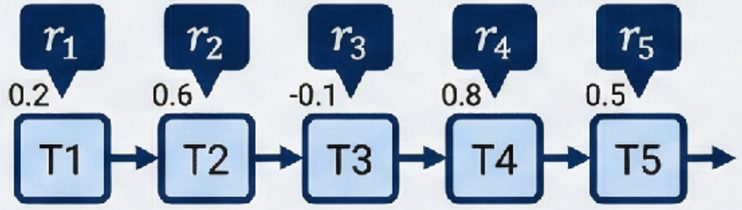
Which tokens contributed to success?

- Sparse outcome reward → no per-token signal

SOLUTION APPROACHES

METHOD A: Process Reward Models (PRMs)

Train separate model to score intermediate steps



Used in math reasoning (e.g., OpenAI PRM800K)

METHOD B: Reward Redistribution

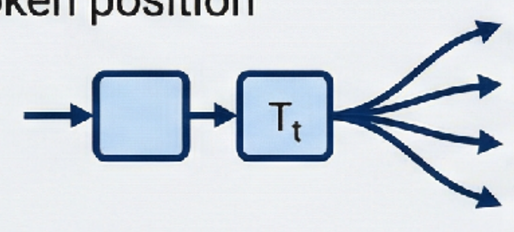
Distribute final R across tokens via heuristics

- uniform, length-weighted, attention-based

$$r_t = R \cdot \frac{w_t}{\sum w_t}$$

METHOD C: Monte Carlo Token Values

Sample continuations from each token position



Estimate $V(s_t) = \mathbb{E}[R \mid \text{prefix up to } t]$

Expensive but model-free

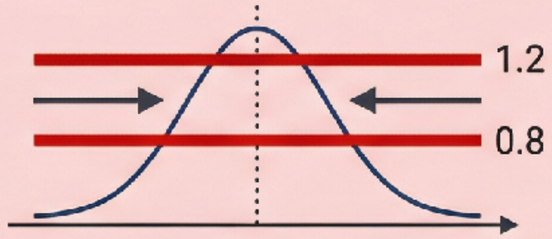
Key trade-off: PRM accuracy vs training cost of second model

DAPO: Intuition Behind the Fixes

SECTION 1 - "Clip-Higher: Let Good Actions Breathe"

BEFORE (Standard PPO)

Standard PPO clips ratio to [0.8, 1.2]



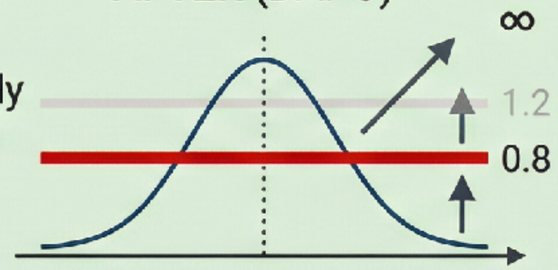
Policy can't increase prob of great actions beyond 1.2x

Result: Distribution narrows → entropy collapse → mode collapse

FIX

AFTER (DAPO)

DAPO clips only lower bound [0.8, ∞)



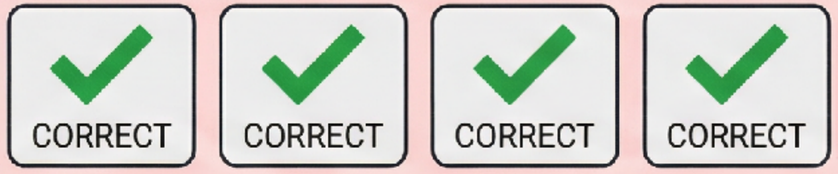
Great actions can grow without limit

Result: Maintains exploration, policy stays diverse

SECTION 2 - "Dynamic Sampling: No Wasted Batches"

BEFORE (Standard Batching)

Response



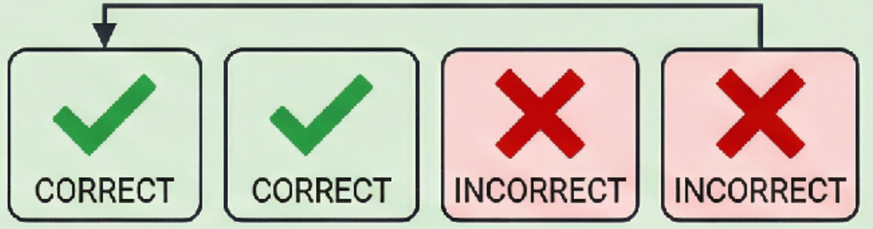
Advantage = R - mean(R) = 0 for everyone

Problem: Zero gradient signal, wasted compute

FIX

AFTER (DAPO Dynamic Sampling)

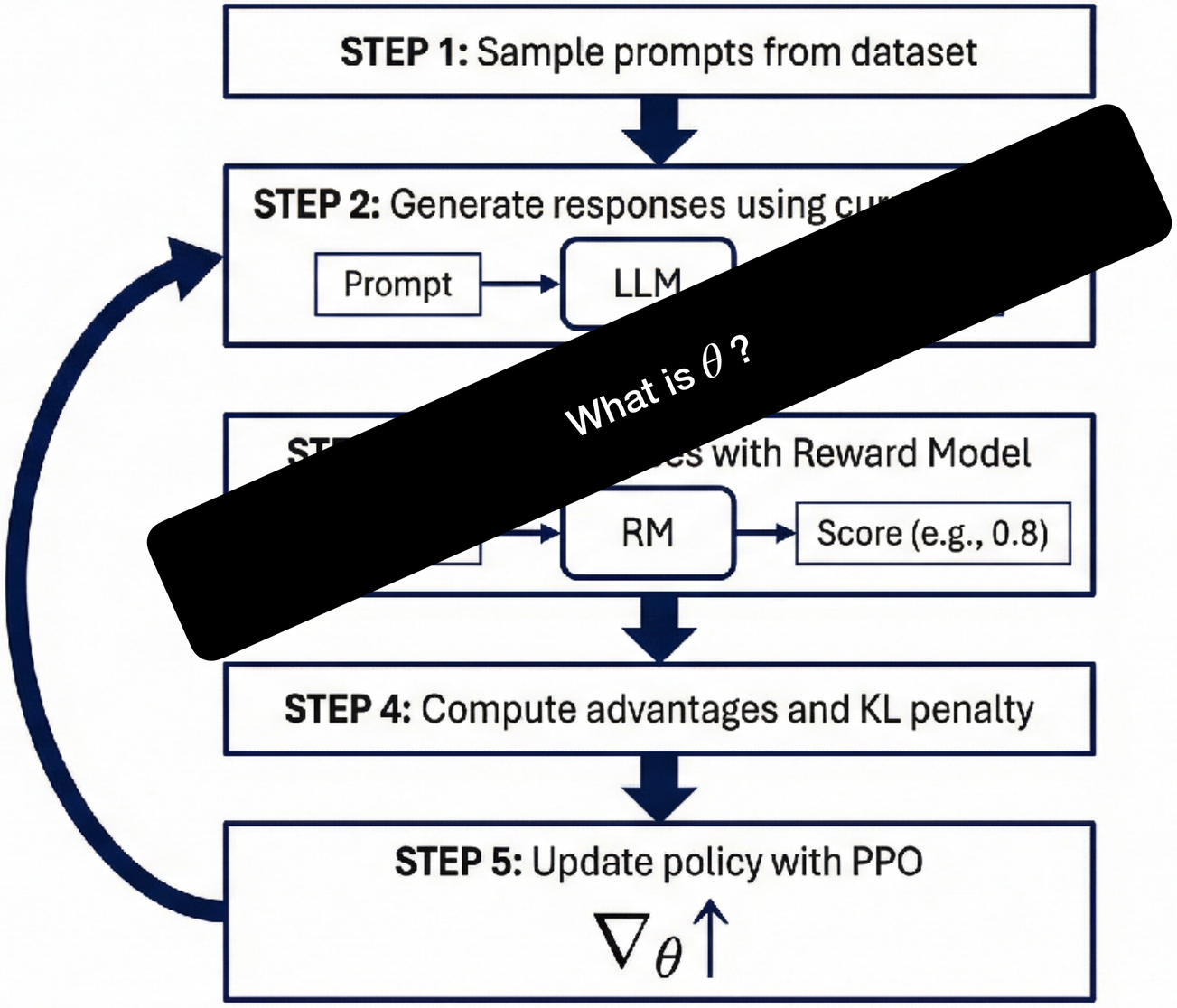
Keep sampling until mix exists



Solution: Now we have contrast → meaningful gradients

Core insight: Remove training bottlenecks that kill learning signal

RLHF Training Loop: Step by Step

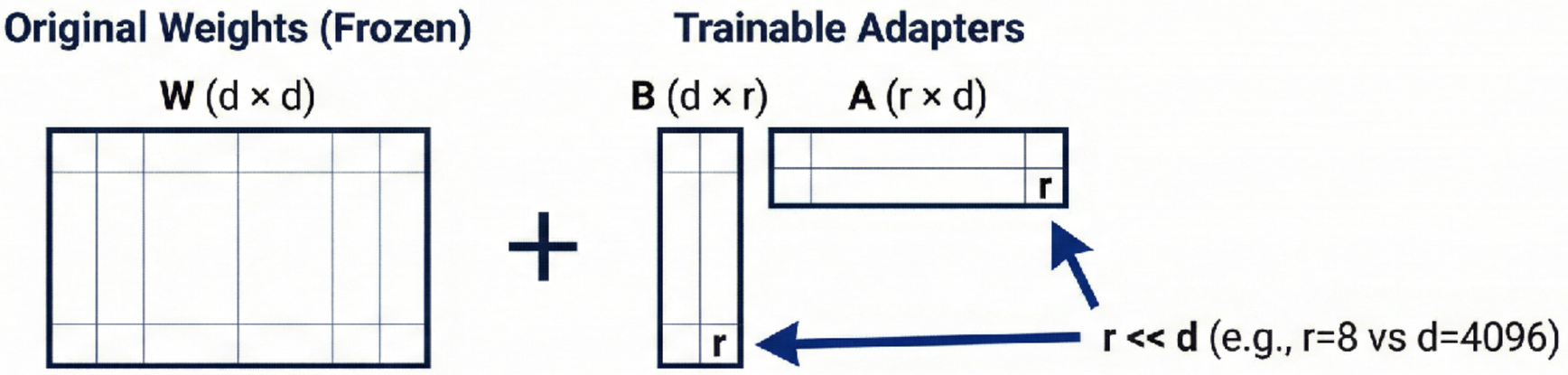


Batch Sizes:
- Batch of prompts: 32-128
- Responses per prompt: 1-4
- PPO epochs per batch: 1-4

LoRA: Low-Rank Adaptation

Hu et al., 2021

SECTION 1 - The Core Idea



$$W' = W + BA$$

SECTION 2 - What This Means

Freeze the giant model
Original weights stay fixed

Train tiny matrices
Only **BA** gets gradient updates

Merge at inference
W' = W + BA → no extra latency

Insight: Fine-tuning updates are empirically low-rank — we can exploit this structure

Why LoRA Works: Benefits & Theory

Practical Benefits

Memory Efficient

- GPT-3 175B: **350GB → 35MB adapters**
- Train on consumer GPUs

No Inference Cost

- Merge **BA into W**
- Same latency as original model

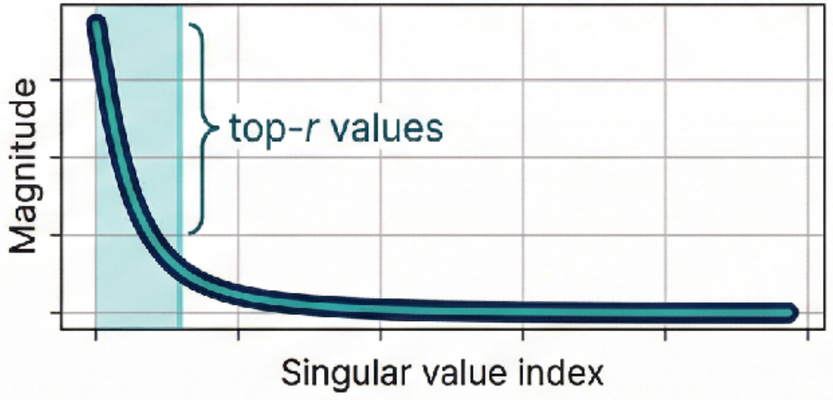
Modular & Swappable

- Different adapters for different tasks
- Hot-swap without reloading base model

Reduces Catastrophic Forgetting

- Original weights frozen
- Base capabilities preserved

Theoretical Justification



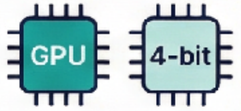
"Aghajanyan et al. 2020: "
Pre-trained models have low intrinsic dimensionality"

Weight updates **ΔW** have low intrinsic rank — most information in top singular values

LoRA Variants & Practical Guide

QLoRA

4-bit quantized base, train on single GPU

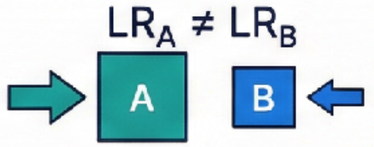


DoRA

magnitude and direction decomposition

LoRA+

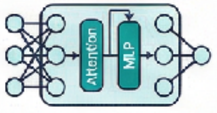
different learning rates for A and B matrices



rsLoRA

rank-stabilized scaling

Practical Guide

- **Where:** attention and MLP layers 
- **Rank: r=8** minimal  **r=32** balanced  **r=256** full 
- **When to skip:** pretraining, small models 

Model based RL a.k.a. World Models?

See you on Wednesday!