

Breaking Down R1

Reasoning Via Reinforcement Learning

Zach Liu

CAISH

Feb 2025

Contents

1 Preliminaries

- Transformers
- Reinforcement Learning

2 R1-Zero

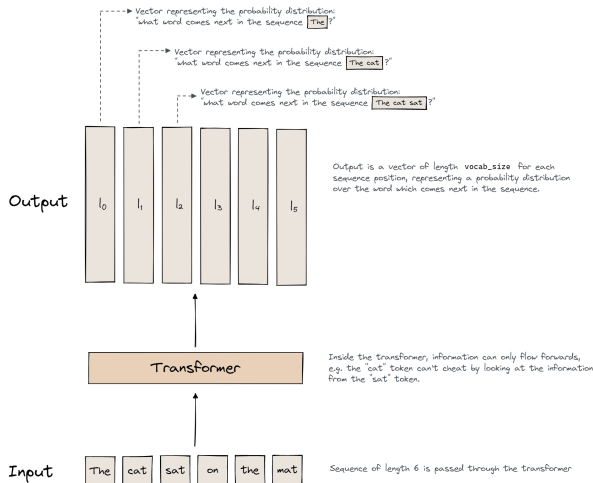
3 R1 Training Pipeline

- Phase 1: Cold Start
- Phase 2: Reasoning Reinforcement Learning
- Phase 3: Rejection Sampling and Supervised Fine-Tuning
- Phase 4: Diverse Reinforcement Learning Phase

4 R1 Engineering Unlocks

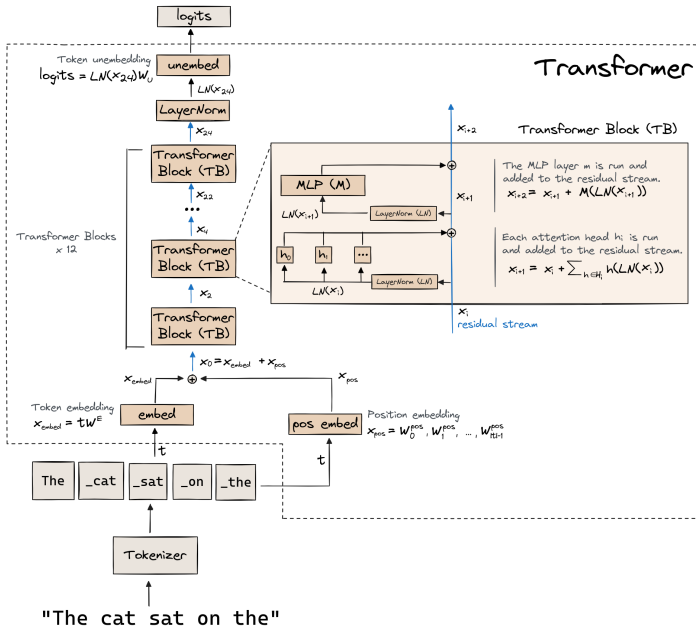
Preliminaries

Autoregressive Models



$$p(x_1, \dots, x_N) = \prod_{n=1}^N p(x_n | x_1, \dots, x_{n-1}) \quad (1)$$

Transformer Architecture



Reinforcement Learning Setup

- The agent in **state** s issues an **action** a to the environment, and the environment replies with **state, reward** pairs (s', r) . We can define a trajectory τ as a sequence of states and actions: $s_0, a_0, r_1, a_1, r_2, s_2, a_2, r_3 \dots$
- The agent chooses actions using a policy π which can either be a deterministic function $a = \pi(s)$ from states to actions, or more generally actions are sampled $a \sim \pi(\cdot|s)$
- The **return** at time t is the sum of rewards obtained *after* time t until the time T when the terminal state is reached, discounted by how far into the future:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} \dots \quad (2)$$

Value Functions and Advantages

- The **state value function** $V_\pi(s)$ is the expected return, if you start in state s and always act according to policy π :

$$V_\pi(s) = \mathbb{E}_\pi [G_t | s_t = s] \quad (3)$$

- The **action value function** is the expected return if you start in state s , take an arbitrary action a (which may not have come from the policy), and then afterwards forever act according to policy π :

$$Q_\pi(s, a) = \mathbb{E}_\pi [G_t | s_t = s, a_t = a] \quad (4)$$

- The **advantage function** describes how much better it is to take a specific action a in state s , over randomly selecting an action according to $\pi(\cdot|s)$, assuming you act according to π forever after:

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s) \quad (5)$$

Trust Region Policy Optimization (TRPO)

Let π_θ denote a policy with parameters θ . We aim to maximize the expected return J of policy π_θ , where we take expectation over all possible trajectories τ generated by following policy π_θ

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} [G(\tau)]$$

To achieve the optimum π_θ that maximizes J , we can iteratively update π_{θ_k} according to:

$$\theta_{k+1} = \arg \max_{\theta} \mathcal{L}(\theta_k, \theta) = \arg \max_{\theta} \mathbb{E}_{s, a \sim \pi_{\theta_k}} [L(s, a, \theta_k, \theta)], \quad \text{s.t. } \bar{D}_{KL}(\theta \| \theta_k) \leq \delta$$

where $\mathcal{L}(\theta_k, \theta)$ is the *surrogate advantage*, a measure of how policy π_θ performs relative to the old policy π_{θ_k} using data from the old policy:

$$L(s, a, \theta_k, \theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A_{\pi_{\theta_k}}(s, a)$$

TRPO updates policies by taking the largest step possible to improve performance, while satisfying a special constraint on how close the new and old policies are allowed to be.

Proximal Policy Optimization (PPO)

Proximal Policy Optimization updates policies via:

$$\theta_{k+1} = \arg \max_{\theta} \mathcal{L}(\theta_k, \theta) = \arg \max_{\theta} \mathbb{E}_{s, a \sim \pi_{\theta_k}} [L(s, a, \theta_k, \theta)],$$

typically taking multiple steps of SGD to maximize the objective. Here L is given by:

$$L(s, a, \theta_k, \theta) = \min \left(\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)} A_{\pi_{\theta_k}}(s, a), \text{clip} \left(\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)}, 1 - \epsilon, 1 + \epsilon \right) A_{\pi_{\theta_k}}(s, a) \right) \quad (6)$$

PPO-Clip doesn't have a KL-divergence term in the objective and doesn't have a constraint at all. Instead relies on specialized clipping in the objective function to remove incentives for the new policy to get far from the old policy.

Advantage Estimation in PPO

Generalized Advantage Estimation (GAE) is used in PPO to estimate the advantage function:

$$A_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \quad (7)$$

where:

- $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ (TD error)
 - ▶ the reward r_t is usually given by a trained (neural) reward model
- γ is the discount factor
- λ is the GAE parameter (tradeoff between bias and variance)
- $V(s)$ is the value function estimate, from value network
 - ▶ Often a value network V_ψ needs to be trained alongside the policy model π_θ to estimate the value, often as big as the policy model itself

Advantage Calculation in PPO:

The truncated form for practical implementation:

$$A_t = \delta_t + (\gamma\lambda)\delta_{t+1} + (\gamma\lambda)^2\delta_{t+2} + \dots \quad (8)$$

General RL Training Setup

Objective:

$$J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^T \gamma^t r_t \right],$$

where $\tau = \{s_0, a_0, r_1, s_1, a_1, \dots, s_T\}$ is a trajectory. For LLMs, *actions* a_t here are *tokens*!

Policy Gradient:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s,a \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a)],$$

or equivalently, using the advantage function $A^{\pi_\theta}(s, a)$:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s,a \sim \pi_\theta} [\nabla_\theta \log \pi_\theta(a|s) A^{\pi_\theta}(s, a)].$$

Trajectory Collection: LLM interacts with the environment (reward model) to sample trajectories. For each timestep:

$$s_t \xrightarrow{\pi_\theta} a_t \rightarrow r_{t+1}, s_{t+1}.$$

Return and Advantage Calculation:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \quad A(s_t, a_t) = Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t).$$

Generalized Advantage Estimation (GAE) are used to compute $A(s_t, a_t)$ practically

Policy Update:

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_\theta),$$

Using the computed advantages, update the policy parameters and value network by maximizing a surrogate objective (e.g., the PPO objective) to favor actions with higher expected returns. Repeat until agent's performance converges.

R1-Zero

Group Relative Policy Optimization (GRPO)

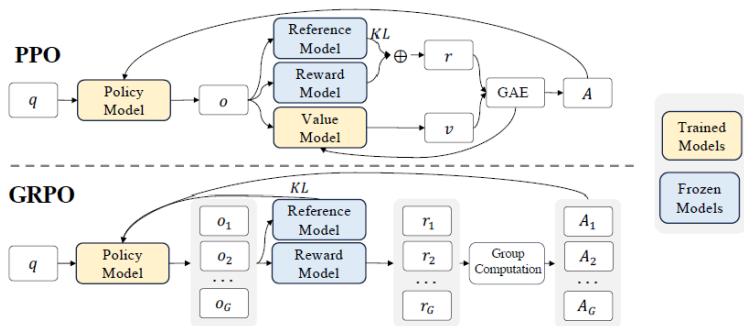


Figure 4 | Demonstration of PPO and our GRPO. GRPO foregoes the value model, instead estimating the baseline from group scores, significantly reducing training resources.

DeepSeek uses GRPO, created in-house, which foregoes the value network and estimates the advantages directly from group rewards instead.

For each question q , GRPO samples a group of outputs $\{o_i\}_{i=1}^G$ from the old policy $\pi_{\theta_{\text{old}}}$ and then optimizes the policy model π_{θ} by maximizing:

$$\begin{aligned} \mathcal{J}_{\text{GRPO}}(\theta) = & \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(Q|q)] \\ & \cdot \frac{1}{G} \sum_{i=1}^G \left(\min \left(\frac{\pi_{\theta}(o_i|q)}{\pi_{\theta_{\text{old}}}(o_i|q)} A_i, \text{clip} \left(\frac{\pi_{\theta}(o_i|q)}{\pi_{\theta_{\text{old}}}(o_i|q)}, 1 - \epsilon, 1 + \epsilon \right) A_i \right) - \beta \mathbb{D}_{\text{KL}}(\pi_{\theta} \parallel \pi_{\text{ref}}) \right), \end{aligned} \quad (1)$$

$$\mathbb{D}_{\text{KL}}(\pi_{\theta} \parallel \pi_{\text{ref}}) = \frac{\pi_{\text{ref}}(o_i|q)}{\pi_{\theta}(o_i|q)} - \log \frac{\pi_{\text{ref}}(o_i|q)}{\pi_{\theta}(o_i|q)} - 1, \quad (2)$$

where ϵ and β are hyper-parameters, and A_i is the advantage ¹

¹Instead of adding KL penalty in the reward like in PPO, GRPO regularizes by directly adding the KL divergence between the trained policy and the reference policy to the loss, avoiding complicating the calculation of advantage.

GRPO Advantages

Advantages A_i are computed using group rewards $\{r_1, r_2, \dots, r_G\}$:

$$A_i = \frac{r_i - \text{mean}(\{r_1, r_2, \dots, r_G\})}{\text{std}(\{r_1, r_2, \dots, r_G\})} \quad (3)$$

As the value network is typically another model of comparable size as the policy model, it brings a substantial memory and computational burden. GRPO eliminates value network completely!

Reward Modelling

R1-Zero uses a rule-based reward system consisting of:

- **Accuracy rewards:** The accuracy reward model evaluates whether the response is correct.
 - ▶ For math problems with deterministic results, can easily check if right or wrong. Provider higher reward if right.
 - ▶ For LeetCode problems, a compiler can be used to generate feedback based on predefined test cases
- **Format rewards:** Enforces the model to put its thinking process between `<think>` and `</think>` tags.

Usually for LLMs, a neural reward model (neural network used to estimate rewards) is used to provide rewards, as human preferences or requirements are difficult to define in a rules-based way. However, R1 does not use any neural reward model:

- Neural reward model may suffer from reward hacking
- Retraining the reward model needs additional training resources and it complicates the whole training pipeline

Performance

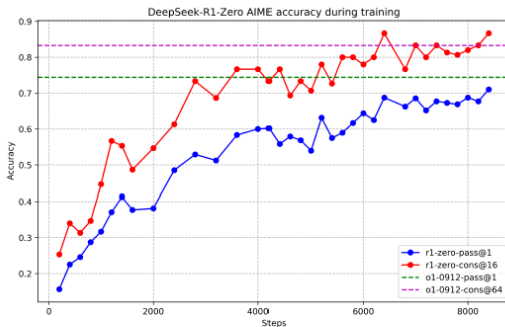


Figure 2 | AIME accuracy of DeepSeek-R1-Zero during training. For each question, we sample 16 responses and calculate the overall average accuracy to ensure a stable evaluation.

- No Supervised-Fine-Tuning involved in R1-Zero
- Majority voting bring performance on AIME benchmark from 71% to 86%

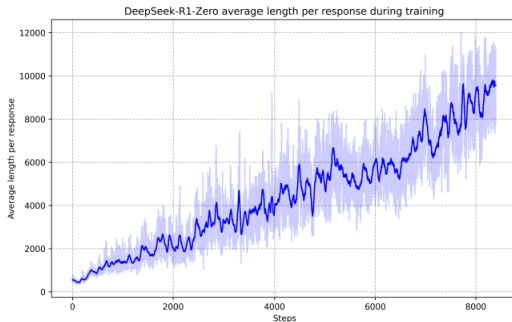


Figure 3 | The average response length of DeepSeek-R1-Zero on the training set during the RL process. DeepSeek-R1-Zero naturally learns to solve reasoning tasks with more thinking time.

- R1-Zero naturally acquires the ability to solve increasingly complex reasoning tasks by leveraging extended test-time computation (increased Chain-Of-Thought)
- Behaviours like backtracking and reflection emerge

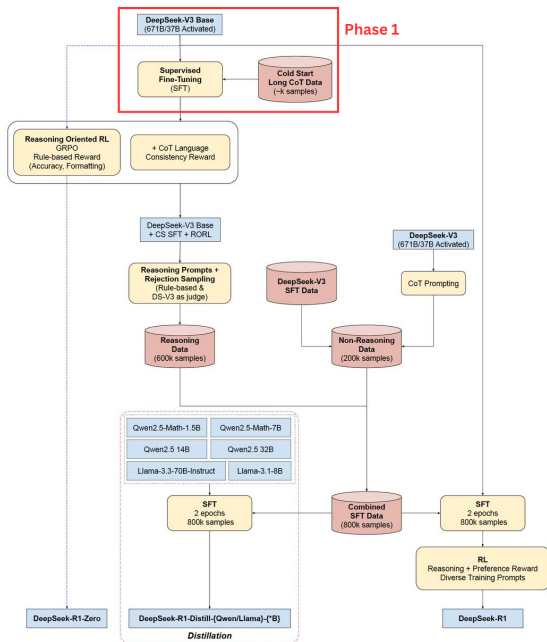
Problems

- Poor Readability of Chain-Of-Thought
- Language Mixing

Probably some form of reward over-optimization, more efficient to reason in its own language and achieve higher reward

R1 Training Pipeline

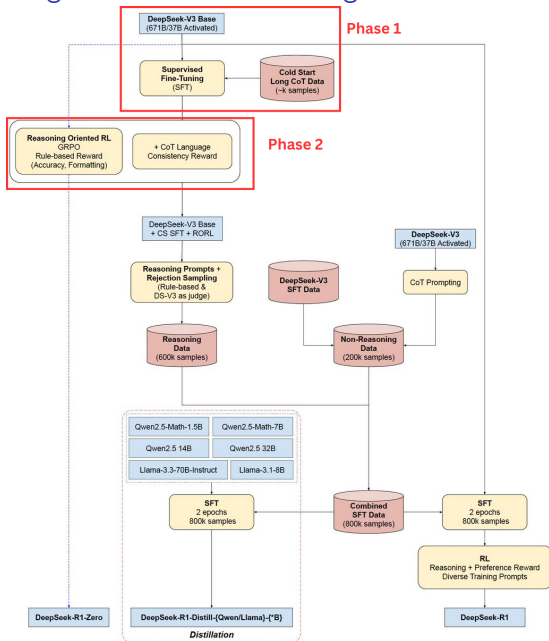
Phase 1: Cold Start



Phase 1: Cold Start

- Construct and collect small amounts (thousands) of long CoT data to fine-tune V3
 - ▶ Using few-shot prompting on V3 with long CoT examples
 - ▶ Direct prompting on V3 for detailed answers with reflection
 - ▶ Gathering DeepSeek-R1-Zero outputs with human post-processing
- Key advantages:
 - ▶ **Readability:** R1-Zero has unreadable responses. We define output format as `|special_token|<reasoning_process>|special_token|<summary>` so reasoning process (CoT) and summary of reasoning provided
 - ▶ **Potential:** Carefully designed patterns with human priors show better performance vs R1-Zero

Phase 2: Reasoning Reinforcement Learning



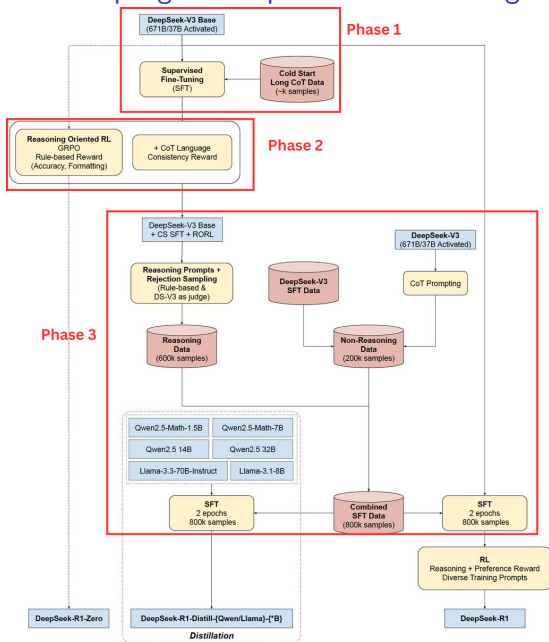
Phase 2: Reasoning Reinforcement Learning

After cold start fine-tuning, apply large-scale RL training similar to DeepSeek-R1-Zero:

- Focus on reasoning-intensive tasks:
 - ▶ Coding, mathematics, science, logic reasoning
 - ▶ Well-defined problems with clear solutions
- Language Consistency:
 - ▶ CoT exhibits language mixing in multi-language prompts
 - ▶ Introduce language consistency reward, measured as proportion of target language words in CoT

Final reward function combines reasoning task accuracy, language consistency reward, and formatting reward to train until convergence.

Phase 3: Rejection Sampling and Supervised Fine-Tuning



Phase 3: Rejection Sampling and Supervised Fine-Tuning

After RL convergence, collect SFT data from checkpoint for further training. Include

Reasoning Data (600k samples)

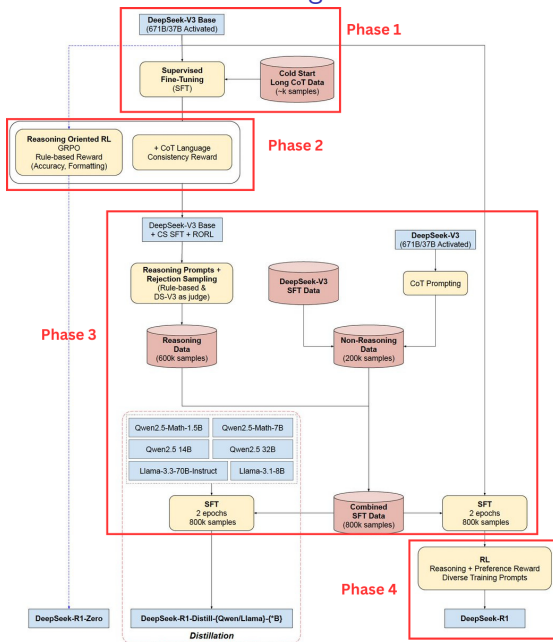
- Rejection sampling from RL checkpoint
- Expanded dataset includes:
 - ▶ Rule-based rewards
 - ▶ Generative rewards via DeepSeek-V3 as LLM-Judge
- Quality filters:
 - ▶ Remove mixed languages
 - ▶ Filter long paragraphs
 - ▶ Remove chaotic code blocks

Non-Reasoning Data (200k samples)

- Types:
 - ▶ Writing
 - ▶ Factual QA
 - ▶ Self-cognition
 - ▶ Translation
- Reuse SFT data for DeepSeek-V3

Final fine-tuning on **DeepSeek-V3-Base**: 2 epochs on combined 800k samples

Phase 4: Diverse Reinforcement Learning Phase



Phase 4: Diverse Reinforcement Learning Phase

Secondary RL stage to align with human preferences while improving, helpfulness, harmlessness, and reasoning:

Training Approach

- Reasoning data:
 - ▶ Same approach as DeepSeek-R1-Zero
 - ▶ Rule-based rewards on Math, code, logical reasoning
- General data:
 - ▶ Reward model as in DeepSeek V3
 - ▶ Use preference pairs

HHH Criteria

- Helpfulness:
 - ▶ Focus on final summary in summary tag is useful and relevance
 - ▶ Preserves reasoning process in the reasoning tag
- Harmlessness:
 - ▶ Evaluate full response for potential biases or harmful content

R1 Engineering Unlocks

Engineering Unlocks

Insane software and hardware optimizations

- Multi-Head Latent Attention (for efficient inference)
 - ▶ Low-rank join compression for $\mathbf{K}, \mathbf{Q}, \mathbf{V}$ matrices to reduce key-value cache memory usage
- DeepSeekMoE with Auxiliary-Loss-Free Load Balancing (efficient training)
 - ▶ 671B parameters, but only which 37B are activated for each token
- Multi-Token Prediction Objective