# Reinforcement Learning for Large Language Models

Instructor: Tong Zhang

Part 3: Reinforcement Learning with Verifiable Rewards (RLVR)

# Outline

- **From RLHF to RLVR**
- **Policy Gradient based Policy Learning** (High-level Overview)
- **GRPO**: Shao et al. (2024). *DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models.*
- **DeepSeek-R1: reasoning model training recipe**: Guo et al. (2025). "Deepseek-R1: Incentivizing reasoning capability in llms via reinforcement learning". *arXiv preprint arXiv:2501.12948*

# From RLHF to RLVR

**RLHF** focuses on alignment (helpfulness/safety), typically via a learned reward model from human preferences.

**RLVR** focuses on reasoning with verifiable outcomes, replacing subjective preference labels with programmatic verification when possible.

Examples:

- math: final answer correctness
- code: unit tests / execution outcomes
- structured tasks: deterministic constraints

# When RLVR Is Preferable to RLHF

RLHF is useful but noisy:

- preference labels are often subjective
- reward-model errors (reward hacking) can be hard to debug
- human preference data are costly to construct

RLVR is attractive when a verifier exists:

$$r(x, a) \in \{0, 1\} \quad \text{or} \quad r(x, a) \in \mathbb{R} \text{ from verifier score.}$$

No need to learn an extra reward model when a reliable verifier exists.

Often the verifier exists only for a subset of prompts with known solutions or desired behavior.

# RLVR Example: Math Verifier

- Prompt: "Solve $2x + 5 = 17$."
- Correct trace: $2x = 12 \to x = 6$; output "$x = 6$" (correct).
- Incorrect trace: $2x = 17 - 5 = 12 \to x = 7$; output "$x = 7$" (incorrect).
- Verifier checks final answer: $r(x, a) = 1$ for correct, $0$ for incorrect.

Remark: we often require reasoning tokens to output a fixed format, such as

`<think>...</think><answer>...</answer>`,

with a reward enforcing such format. However the main reward signal still comes from final-answer correctness.

# RLVR Example: Coding Unit Tests

- Prompt: “Write a function to reverse a string s.”
- Correct trace: “Need reversed order; Python slicing from end is concise.” $\rightarrow$ code uses ‘s[::-1]‘; passes unit tests.
- Incorrect trace: “String is already fine; just return input.” $\rightarrow$ code returns ‘s‘ unchanged; fails unit test ‘reverse(”abc”)==”cba”‘.
- Verifier runs unit tests (optionally in sandbox); reward is pass/fail or pass rate.
- Practical setup: use test-based reward as primary signal and optionally add a small format reward for reasoning/output tags.
- Full reasoning-quality checker can be added, but is a more advanced setup.

# RLHF vs RLVR

| | **RLHF** | **RLVR** |
|---|---|---|
| Primary goal | alignment (helpfulness/safety) | reasoning ability |
| Reward source | learned from human preferences | computed by verifier (no learning) |
| Reward verifiability | often subjective/non-checkable | explicitly checkable by rules/tests |
| Reward model? | usually yes | not necessary |
| Typical domains | general helpfulness/safety | math/code/structured QA |
| Main risk | reward model misspecification and reward hacking | verifier incompleteness (only outcome, not intermediate steps) |

# RLVR Objective

The optimization form is still KL-regularized policy learning:

$$\max_{\pi} \; \mathbb{E}_x \mathbb{E}_{a\sim\pi(\cdot|x)} \left[ r(x,a) - \beta \log \frac{\pi(a|x)}{\pi_{\text{ref}}(a|x)} \right].$$

In practice, we may define the RLVR reward as

$$r(x,a) = r_{\text{verifier}}(x,a) + \eta \, r_{\text{format}}(a),$$

where $a$ contains both the reasoning trace and final answer.

- $r_{\text{verifier}}(x,a)$: correctness/tests (primary signal)
- $r_{\text{format}}(a)$: check parseable structure, e.g. `<think>...</think><answer>...</answer>`
- Example: $r_{\text{format}} = 1$ if tags exist and parse succeeds, else $0$.

# Notation Used in This Lecture

We will use the following notation throughout the RLVR and GRPO slides:

- $x$: prompt or problem instance.
- $a = (a_1, \ldots, a_T)$: generated response tokens, including reasoning and final answer.
- $\pi_\theta(a \mid x)$: trainable policy; $\pi_{\mathrm{ref}}(a \mid x)$: reference policy.
- $r(x, a)$: verifier reward, often binary for math/code tasks.
- $\beta$: coefficient controlling the KL penalty to the reference policy.

The same symbols appear at the sequence level and token level:

$$\pi_\theta(a \mid x) = \prod_{t=1}^{T} \pi_\theta(a_t \mid x, a_{<t}).$$

# RL Terminology for RLVR

We can write RLVR as a standard RL problem with known reward:

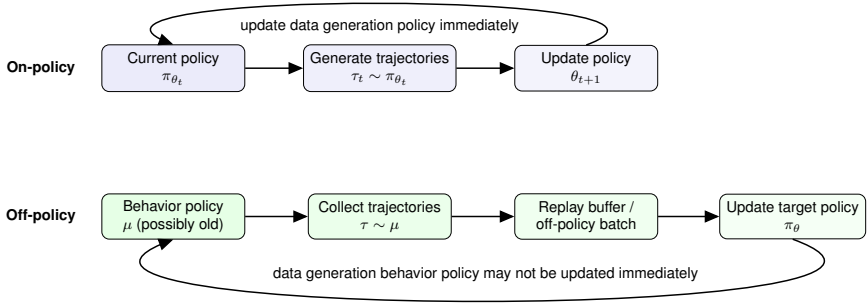
- state: prompt $x$
- action: response $a$ containing both thinking tokens and final answer
- policy $\pi_\theta(a \mid x)$: language model distribution
- trajectory $\tau$: full generated sequence (prompt $x$, reasoning and final answer $a$)
- entropy regularized reward:

$$\tilde{r}_\theta(x, a) = r(x, a) - \beta \ln \frac{\pi_\theta(a|x)}{\pi_{\text{ref}}(a|x)},$$

where $r(x, a)$ is the known verifier reward, plus optionally a format term.

Training goal: optimize policy parameters $\theta$ so expected verifier reward increases, subject to KL regularization to $\pi_{\text{ref}}$.

# RL Training Pipelines




- On-policy: training data must be generated by the current policy.
- Off-policy: training data can come from behavior/older policies.
- In RLVR, reward is known from a verifier; training focuses on optimizing policy $\pi_\theta$.

# Equivalent RL Formulations: Advantage

Given reward function $r(x, a)$, the objective of RL is to find policy $\pi_\theta$ to maximize

$$J(\theta) = \mathbb{E}_x \mathbb{E}_{a\sim\pi_\theta(\cdot|x)} \left[ r(x, a) - \beta \ln \frac{\pi_\theta(a|x)}{\pi_{\text{ref}}(a|x)} \right].$$

Consider any advantage function $A(x, a)$ is of the form

$$A(x, a) = r(x, a) - V(x),$$

where $V(x)$ can be an arbitrary function of $x$ (but independent of $a$).

Let

$$J_A(\theta) = \mathbb{E}_x \mathbb{E}_{a\sim\pi_\theta(\cdot|x)} \left[ A(x, a) - \beta \ln \frac{\pi_\theta(a|x)}{\pi_{\text{ref}}(a|x)} \right],$$

then optimizing $J(\theta)$ is equivalent to optimizing $J_A(\theta)$.

A common choice is $V(x) = \mathbb{E}_{a\sim\pi_{\theta_{\text{old}}}(\cdot|x)}[r(x, a)]$ for some $\theta_{\text{old}}$ close to current $\theta$, which reduces variance of $A(x, a)$.

# Equivalence Formulations with Behavior Policy

Our goal is to optimize

$$J_A(\theta) = \mathbb{E}_x \mathbb{E}_{a\sim\pi_\theta(\cdot|x)} \underbrace{\left[ A(x,a) - \beta \ln \frac{\pi_\theta(a|x)}{\pi_{\text{ref}}(a|x)} \right]}_{\text{data generated by current policy}}$$

Consider using behavior policy $\mu$ to generate trajectory $(x, a)$. Then we can rewrite $J_A(\theta)$ equivalently using behavior policy as follows

$$J_A(\theta) = \mathbb{E}_x \mathbb{E}_{a\sim\mu(\cdot|x)} \underbrace{\left[ \rho_\theta(x,a) \left( A(x,a) - \beta \ln \frac{\pi_\theta(a|x)}{\pi_{\text{ref}}(a|x)} \right) \right]}_{\text{data generated by behavior policy}}, \quad \rho_\theta(x,a) = \frac{\pi_\theta(a|x)}{\mu(a|x)}.$$

This reformulation uses importance weighting $\rho_\theta(x, a)$ to simulate policy $\pi_\theta$ using $\mu$.

# Policy Gradient Methods

- **PG** (Policy Gradient): optimize the policy by increasing probability of high-advantage responses.
- **Off-policy PG**: reuse responses from a behavior policy $\mu$, but correct with importance ratios.
- **PPO**: stabilize this update by clipping those ratios.

# Policy Gradient (PG)

RL Problem:

$$\max_{\theta} J_A(\theta), \quad J_A(\theta) = \mathbb{E}_x \mathbb{E}_{a\sim\mu(\cdot|x)} \left[ \rho_\theta(x,a) \left( A(x,a) - \beta \ln \frac{\pi_\theta(a|x)}{\pi_{\text{ref}}(a|x)} \right) \right], \rho_\theta(x,a) = \frac{\pi_\theta(a|x)}{\mu(a|x)}.$$

We have: $$\nabla_\theta J_A(\theta) = \mathbb{E}_x \mathbb{E}_{a\sim\mu(\cdot|x)} \left[ \underbrace{\left( A(x,a) - \beta \ln \frac{\pi_\theta(a|x)}{\pi_{\text{ref}}(a|x)} \right) \nabla_\theta \rho_\theta(x,a)}_{\text{PG}_\theta(x,a)} \right].$$

**Off-Policy PG**

Iterate:

- Sample prompt $x$ and generate trajectory: $a \sim \mu(\cdot|x)$ (behavior policy $\mu$ is often $\pi_{\theta_{\text{old}}}$).
- Update policy (with arbitrary prompt dependent function $V(x)$):

$$\theta_{\text{new}} \leftarrow \theta_{\text{old}} + \eta \, \text{PG}_{\theta_{\text{old}}}(x,a)$$

# Key Issues for Off-Policy PG

Off-policy PG Advantages:

- it allows data generation policy to decouple from policy learning.
- multiple PG steps can be performed before updating generation policy.
- trajectories using old policy can be reused.

Drawbacks:

- $\pi_\theta$ can drift away from $\mu$
- $\rho$ can be very large when $\pi_\theta$ and $\mu$ differ.
- This causes high-variance gradients and unstable optimization.

Mitigations:

- constrain how far the updated policy can move from the behavior policy.
- PPO does this with a simple clipped importance ratio.

# PPO Clipping Intuition

Suppose trajectories are generated by a behavior policy $\mu = \pi_{\theta_{\text{old}}}$, but we update the new policy $\pi_\theta$. PPO tracks the policy change using the importance ratio

$$\rho_\theta(x, a) = \frac{\pi_\theta(a|x)}{\mu(a|x)}.$$

A very large $\rho_\theta$ means the new policy assigns much higher probability to this response than the old policy did. This can make the gradient unstable.

PPO clips the ratio:

$$\rho_\theta(x, a) \mapsto \text{clip}(\rho_\theta(x, a), 1 - \epsilon, 1 + \epsilon),$$

so a single sampled response cannot push the policy update arbitrarily far.

# Proximal Policy Optimization (PPO)

PPO stabilizes policy-gradient learning by clipping the policy ratio.

Iterate

$$\theta_{\text{new}} = \arg\max_{\theta} \; J_{\text{PPO}}(\theta)$$

$$J_{\text{PPO}}(\theta) = \mathbb{E}_x \mathbb{E}_{a\sim\mu(\cdot|x)} \left[ \min \left( \underbrace{\rho_\theta(x,a) A(x,a)}_{\text{RL objective}}, \; \underbrace{\text{clip}(\rho_\theta(x,a), 1-\epsilon, 1+\epsilon) A(x,a)}_{\text{clipped RL objective}} \right) - \beta \rho(\theta) \ln \frac{\pi_\theta(a|x)}{\pi_{\text{ref}}(a|x)} \right], \qquad \rho_\theta(x,a) = \frac{\pi_\theta(a|x)}{\mu(a|x)}, \quad \mu(a|x) = \pi_{\theta_{\text{old}}}(a|x).$$

- First-order optimization using ordinary mini-batch training.
- Clipping discourages overly large policy updates.

Schulman et al. (2017). "Proximal policy optimization algorithms". *arXiv preprint arXiv:1707.06347*

# PPO for LLM: Practical Implementation (without KL term)

In practice for LLMs, PPO is implemented at token level. For sequence $x, a$ and token position $t$ in $a = [a_1, \ldots, a_t, \ldots]$, define state $s_t = (x, a_{<t})$ and action $a_t$.

$$J_{\text{PPO}}(\theta) = \mathbb{E}_{x,a,t}\Big[\min\Big(\rho_t(\theta)\hat{A}_t,\ \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\Big)\Big], \quad \rho_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}.$$

In practice, advantage is estimated using GAE (generalized advantage estimation):

$$\hat{A}_t \leftarrow \text{GAE}(r_t, V_\psi; \gamma, \lambda),$$

with hyperparameters $\gamma$ and $\lambda$.

- $r_t$: token reward obtained from terminal task reward and KL to a reference policy.
- $V_\psi$: a learned value function.

We skip implementation details here; see Schulman et al. (2017). "Proximal policy optimization algorithms". *arXiv preprint arXiv:1707.06347* and Hugging Face TRL (`https://github.com/huggingface/trl`).

# From PPO to GRPO

In RLVR, we can use verifiable rewards directly (no learned value model required).

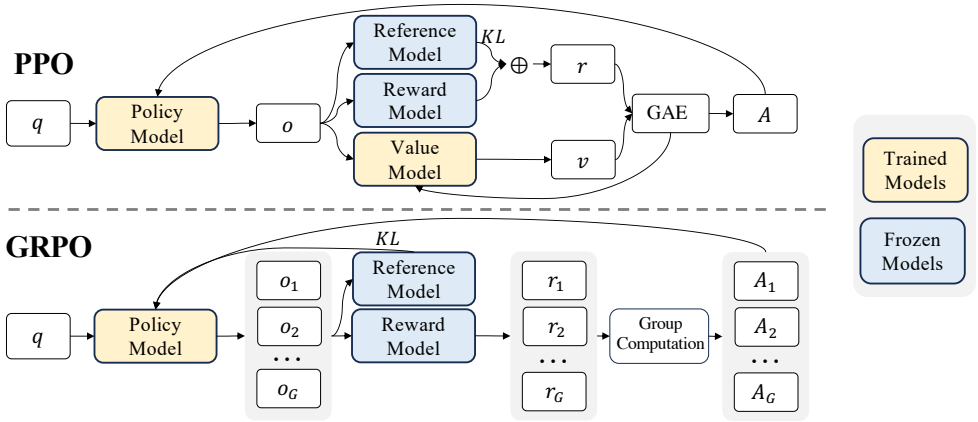
Instead of learning $V_\psi$ as in PPO, GRPO forms group-relative advantages without value function:

- sample a group of responses for one prompt
- score each response with verifier reward
- normalize rewards within the group to get $\hat{A}_i$

Then apply PPO-style clipping, with updates weighted at the token level.

Shao et al. (2024). *DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models.*

# PPO versus GRPO




GRPO replaces the value-model branch and GAE by group-relative advantages.

Source: Shao et al. (2024). *DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models.*

# GRPO Rollout Notation

For each prompt $x$, GRPO samples a group of $G$ responses from the behavior policy:

$$a_i = (a_{i,1}, \ldots, a_{i,T_i}) \sim \mu(\cdot \mid x), \qquad i = 1, \ldots, G, \qquad \mu = \pi_{\theta_{\text{old}}}.$$

A verifier assigns one outcome reward per response:

$$r_i = r(x, a_i).$$

After group normalization, the same response-level advantage $\hat{A}_i$ weights every token $a_{i,t}$ in that response.

This is why GRPO is critic-free but still uses token-level PPO-style ratios.

# GRPO Training Objective without KL Regularization

For prompt $x$, sample responses $\{a_i\}_{i=1}^{G}$ and verifier rewards $\{r_i\}$. Define group-normalized advantage

$$\hat{A}_i = \frac{r_i - \text{mean}(R)}{\text{std}(R)}, \qquad R = \{r_1, \ldots, r_G\}.$$

GRPO solves the following problem iteratively

$$\theta_{\text{new}} = \arg\max_{\theta} \left[ J^0_{\text{GRPO}}(\theta) \right],$$

$$J^0_{\text{GRPO}}(\theta) = \mathbb{E}_{x,\{a_i\}\sim\mu} \frac{1}{G} \sum_{i=1}^{G} \frac{1}{|a_i|} \sum_{t=1}^{|a_i|} \min\Big( \rho_{i,t}(\theta)\hat{A}_i,\ \text{clip}(\rho_{i,t}(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_i \Big),$$

$$\rho_{i,t}(\theta) = \frac{\pi_\theta(a_{i,t}|x, a_{i,<t})}{\mu(a_{i,t}|x, a_{i,<t})}, \qquad \mu = \pi_{\theta_{\text{old}}}.$$

Each token gets PPO-style weighting, while $\hat{A}_i$ comes from final answer verifier.

# RL Formulation

Given reward function $r(x, a)$, the objective of RL is to find policy $\pi_\theta$ to maximize

$$J_A(\theta) = \mathbb{E}_x \mathbb{E}_{a\sim\pi_\theta(\cdot|x)} \left[ A(x, a) - \beta \ln \frac{\pi_\theta(a|x)}{\pi_{\text{ref}}(a|x)} \right].$$

Following GRPO, we replace KL part by a nonnegative estimator

$$D_\theta(a|x) = \ln \frac{\pi_\theta(a|x)}{\pi_{\text{ref}}(a|x)} + \frac{\pi_{\text{ref}}(a|x)}{\pi_\theta(a|x)} - 1 \geq 0.$$

Because

$$\mathbb{E}_x \mathbb{E}_{a\sim\pi_\theta(\cdot|x)} \left[ \ln \frac{\pi_\theta(a|x)}{\pi_{\text{ref}}(a|x)} \right] = \mathbb{E}_x \mathbb{E}_{a\sim\pi_\theta(\cdot|x)} \left[ D_\theta(a|x) \right],$$

we know that $J_A(\theta)$ can be rewritten equivalently as:

$$J_A(\theta) = \mathbb{E}_x \mathbb{E}_{a\sim\pi_\theta(\cdot|x)} \left[ A(x, a) - \beta D_\theta(a|x) \right].$$

# GRPO with per Token KL Regularization

We can introduce explicit KL penalty into GRPO training objective:

$$J_{\text{GRPO}}(\theta) = J^0_{\text{GRPO}}(\theta) - \beta\, \mathbb{E}_{x,a_i}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|a_i|}\sum_{t=1}^{|a_i|}\rho_{i,t}(\theta) D_\theta(a_{i,t}|x, a_{i,<t})\right],$$

where penalty $D_\theta$ is always non-negative:

$$D_\theta(a_{i,t}|x, a_{i,<t}) = \frac{\pi_{\text{ref}}(a_{i,t} \mid x, a_{i,<t})}{\pi_\theta(a_{i,t} \mid x, a_{i,<t})} - \ln\frac{\pi_{\text{ref}}(a_{i,t} \mid x, a_{i,<t})}{\pi_\theta(a_{i,t} \mid x, a_{i,<t})} - 1.$$

The term $D_\theta$ is equivalent to KL penalty:

$$\mathbb{E}_{x,\{a_i\}\sim\mu}\left[\rho_{i,t}(\theta) D_\theta(a_{i,t}|x, a_{i,<t})\right] = \mathbb{E}_{x,a_i}\left[\rho_{i,t}(\theta)\ln\frac{\pi_\theta(a_{i,t} \mid x, a_{i,<t})}{\pi_{\text{ref}}(a_{i,t} \mid x, a_{i,<t})}\right].$$

# DeepSeekMath: GRPO Results

Reported gains from DeepSeekMath-Instruct 7B to DeepSeekMath-RL 7B using GRPO (Top-1):

| | | |
|---|---|---|
| GSM8K: | $82.9\% \rightarrow 88.2\%$ | $(+5.3)$ |
| MATH: | $46.8\% \rightarrow 51.7\%$ | $(+4.9)$ |
| MGSM-zh: | $73.2\% \rightarrow 79.6\%$ | $(+6.4)$ |
| CMATH: | $84.6\% \rightarrow 88.8\%$ | $(+4.2)$ |

Remark: GRPO improved both English and Chinese math benchmarks without training a separate value model.

Source: Shao et al. (2024). *DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models*.

# Benefits of GRPO over PPO for RLVR

GRPO becomes a key algorithm for training large-scale reasoning LLMs:

- no need to learn value function: reduces memory and tuning burden
- group-relative advantage: easy to compute and stabilizes updates
- naturally matches LLM applications with many sampled candidates
- work well with verifier: allow sparse reward at single-sample level

We will next discuss DeepSeek-R1: the first strong open source reasoning model with a multi-stage training recipe.

# DeepSeek-R1: Multi-Stage Reasoning Training

**DeepSeek-R1-Zero** starts from a base model and uses pure RL (GRPO + rule-based rewards), without an initial SFT stage.

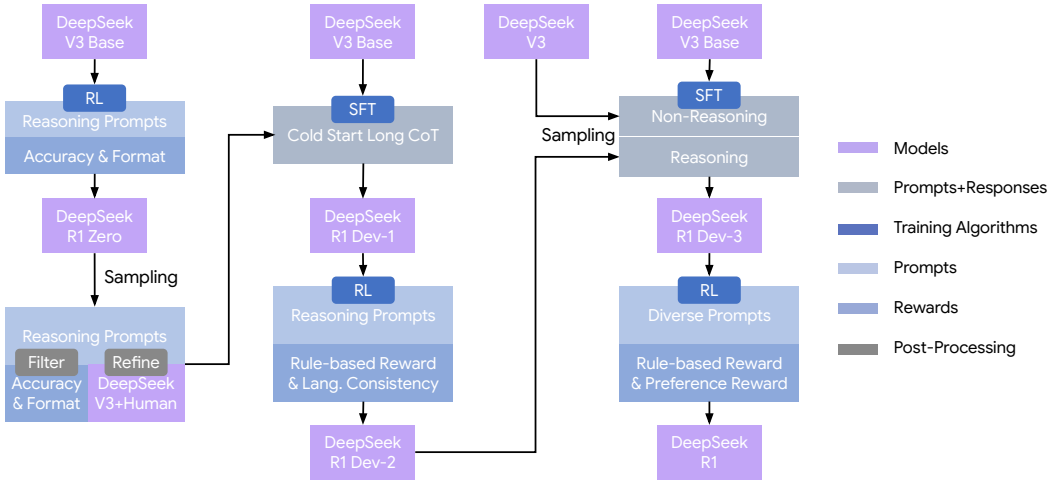
Reward design in R1-Zero:

- **accuracy reward**: answer correctness (math/code/verifiable tasks)
- **format reward**: enforce `<think>...</think><answer>...</answer>`

Reported effect: strong emergent reasoning behavior, but raw R1-Zero has readability/language-mixing issues.

DeepSeek-R1 then adds staged alignment to keep reasoning strength while improving usability.

# DeepSeek-R1 Multi-Stage Training Pipeline




Source: Guo et al. (2025). "Deepseek-R1: Incentivizing reasoning capability in llms via reinforcement learning". *arXiv preprint arXiv:2501.12948*

# Effect of Different Stages

Following the pipeline checkpoints (R1-Zero $\rightarrow$ Dev1 $\rightarrow$ Dev2 $\rightarrow$ Dev3 $\rightarrow$ R1):

- **Cold-start SFT (R1-Zero $\rightarrow$ Dev1):** improves readability/alignment, but can temporarily reduce pure reasoning scores.
- **Reasoning RL (Dev1 $\rightarrow$ Dev2):** recovers and strengthens math/code reasoning via rule-verifiable rewards.
- **Rejection sampling + SFT (Dev2 $\rightarrow$ Dev3):** mixes reasoning and non-reasoning data, improving instruction-following and general usability.
- **Second RL stage (Dev3 $\rightarrow$ R1):** combines reasoning rewards with preference-style signals; final gains are mainly in alignment/general quality, with reasoning kept strong.

Outcome: AIME-2024 Pass@1 rises from R1-Zero (77.9) to final R1 (79.8), while later stages especially improve non-reasoning benchmarks.

# DeepSeek-R1: Key Lessons

- RL with GRPO on verifiable rewards can produce large reasoning gains.
- Pure RL exploration is powerful, but reward/format design strongly affects behavior quality.
- Mixing reasoning and general-alignment data in later stages improves usability.
- Distillation is key for deployment: smaller distilled models inherit much of the reasoning gain.

# Future-KL Regularized GRPO

**Classical GRPO.** A local KL penalty controls per-token drift, but it does not assign KL credit to earlier tokens whose choices change the later contexts seen by the model.

Mathematically, a token at time $t$ changes every later context $(x, a_{i,<j})$ for $j > t$, so reverse-KL regularization gives a future return-to-go:

$$\hat{A}_{i,t} = \hat{A}_i^{\mathrm{GRPO}} - \beta \sum_{j=t}^{T_i} \log \frac{\pi_{\theta_{\mathrm{old}}}(a_{i,j} \mid x, a_{i,<j})}{\pi_{\mathrm{ref}}(a_{i,j} \mid x, a_{i,<j})}.$$

- Start with the usual group-normalized verifier advantage $\hat{A}_i^{\mathrm{GRPO}}$.
- Add a reverse cumulative sum of token log-ratios to propagate KL credit to earlier decisions.
- This gives Future-KL Policy Optimization (FKPO).

Source: Yao et al. (2026). "Future-KL Regularized GRPO: Process-Level Credit Assignment from $f$-Divergence Regularization". *arXiv preprint arXiv:2601.10201*

# Future-KL GRPO: Empirical Motivation


- Outcome-verifier RL is sensitive to how KL regularization is applied.
- Local loss-side KL can control drift but misses future-token credit assignment.
- Future-KL style corrections aim to preserve exploration and improve pass@k.

Source: Yao et al. (2026). "Future-KL Regularized GRPO: Process-Level Credit Assignment from $f$-Divergence Regularization". *arXiv preprint arXiv:2601.10201* , figure `math500_pass16.pdf`.

# References


- **PPO**: Schulman et al. (2017). "Proximal policy optimization algorithms". *arXiv preprint arXiv:1707.06347*
- **GRPO**: Shao et al. (2024). *DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models*.
- **DeepSeek-R1**: Guo et al. (2025). "Deepseek-R1: Incentivizing reasoning capability in llms via reinforcement learning". *arXiv preprint arXiv:2501.12948*
- **Future-KL GRPO**: Yao et al. (2026). "Future-KL Regularized GRPO: Process-Level Credit Assignment from $f$-Divergence Regularization". *arXiv preprint arXiv:2601.10201*