Automated Neural Theorem Proving

Adam Mawani, Chinmay Jindal, Hongyi Huang, Ryan Li, Thanosan Prathifkumar

Advised by Claire Zhao

May 10, 2025

Table of Contents

1. Autoformalization
2. Finding Proofs via Monte Carlo Tree Search
3. Reinforcement Learning with Lean Feedback
4. LLM Post-training for Math Reasoning

Autoformalization

Hongyi Huang

Challenges in Proof Verification

- How to ensure the complete rigour of a proposed mathematical proof?
- Modern mathematics is complex (often ≥ 100 pages/paper)
- Review takes time and effort
- Large scale collaboration on proofs?
- Need to ensure correctness of each contributor's results
- Effective blueprint of proofs

Interactive Proof Assistant: Lean

Theorem `compl_subset_compl_of_subset`: Suppose $A \subseteq B$. Then $B^c \subseteq A^c$.

```
theorem compl_subset_compl_of_subset {A B : Set U} (h1 : A ⊆ B) : Bᶜ ⊆ Aᶜ := by
```

```
intro x
```

Current Goal

Objects:

U : Type

A B : Set U

x : U

Assumptions:

h1 : $A \subseteq B$

Goal:

$x \in B^c \to x \in A^c$

Interactive Theorem Proving

Theorem `compl_subset_compl_of_subset`: Suppose $A \subseteq B$. Then $B^c \subseteq A^c$.

```
theorem compl_subset_compl_of_subset {A B : Set U} (h1 : A ⊆ B) : Bᶜ ⊆ Aᶜ := by
```

```
1 intro x
```

Current Goal

Objects:

U : Type

A B : Set U

x : U

Assumptions:

h1 : $A \subseteq B$

Goal:

$x \in B^c \to x \in A^c$

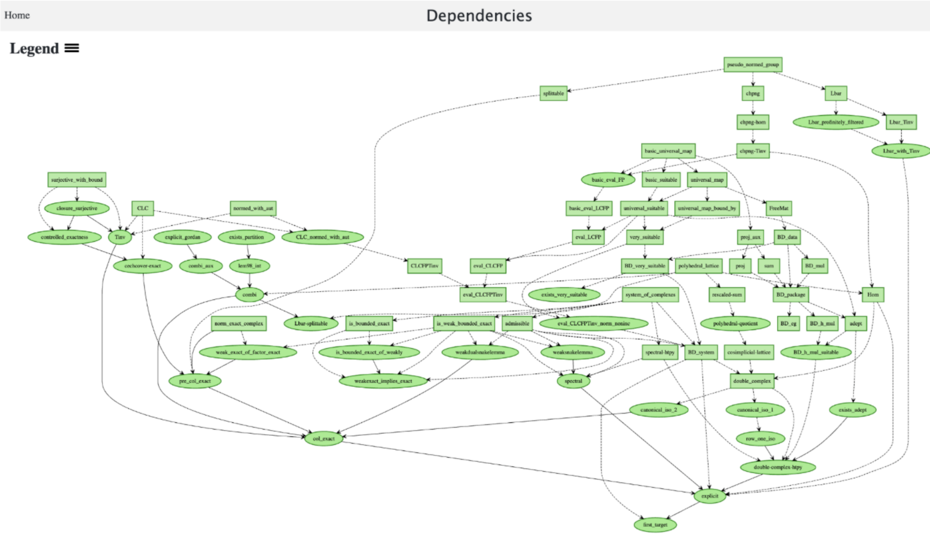
Interactive Theorem Proving

Theorem `compl_subset_compl_of_subset`: Suppose $A \subseteq B$. Then $B^c \subseteq A^c$.

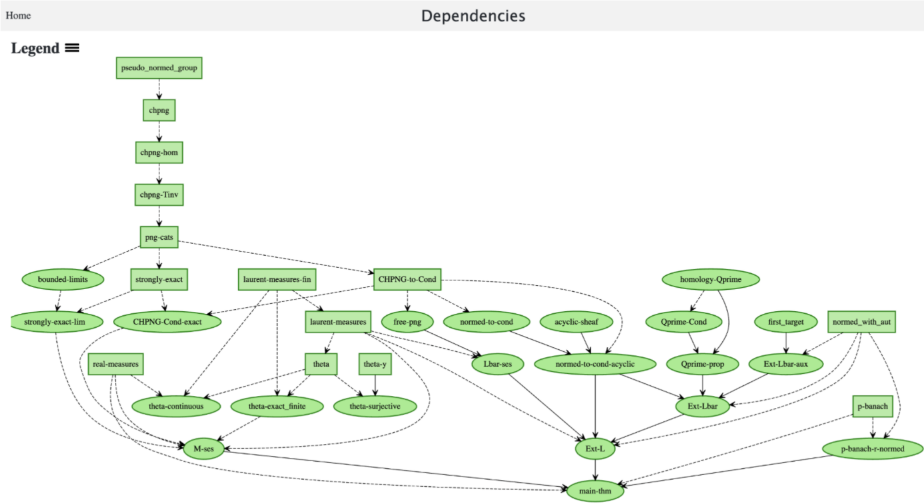
```
theorem compl_subset_compl_of_subset {A B : Set U} (h1 : A ⊆ B) : Bᶜ ⊆ Aᶜ := by
```

```
intro x
intro h2
rw [mem_compl_iff A x]
rw [mem_compl_iff] at h2
intro h3
have h4: x∈ B:= h1 h3
exact h2 h4
```

Peter Scholze's Liquid-Tensor Experiment



Peter Scholze's Liquid-Tensor Experiment



Peter Scholze's Liquid-Tensor Experiment

- Formal verification of Peter Scholze's proof in Lean4
  - Blueprint
  - Definition (Rectangles) + Lemmas (Bubbles)
  - Implication (Arrows)
- Took 1.5 years of Lean community effort
- Possibility for large scale cooperation on theorem proving

Lean is Hard.

Automated formalization

HyperTree Proof Search for Neural Theorem Proving (2022)

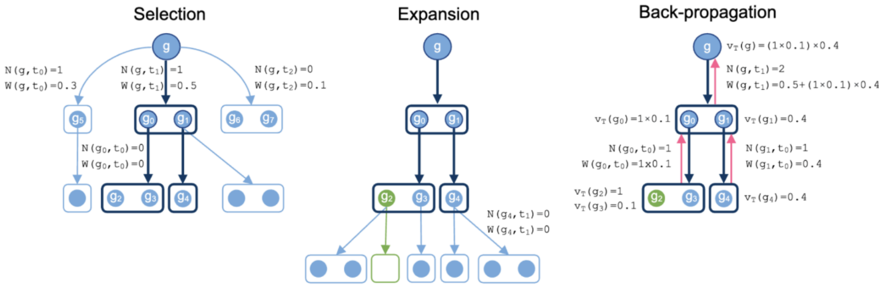
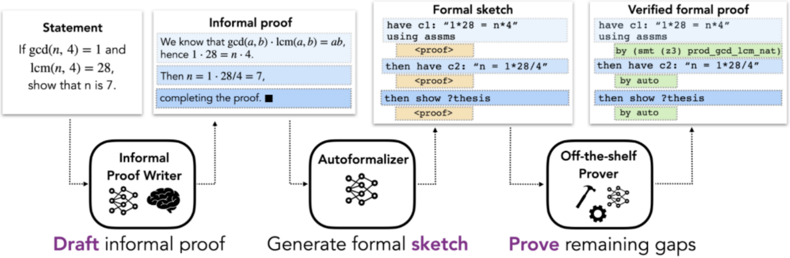


Figure: Monte-Carlo tree search (more from Adam later)

Toward Proof Automation

Draft, Sketch, and Prove: Guiding Formal Theorem Provers with Informal Proofs (2023)



Endowing LLM with Math Knowledge

LeanDojo: Theorem Proving with Retrieval-Augmented Language Models (2023)

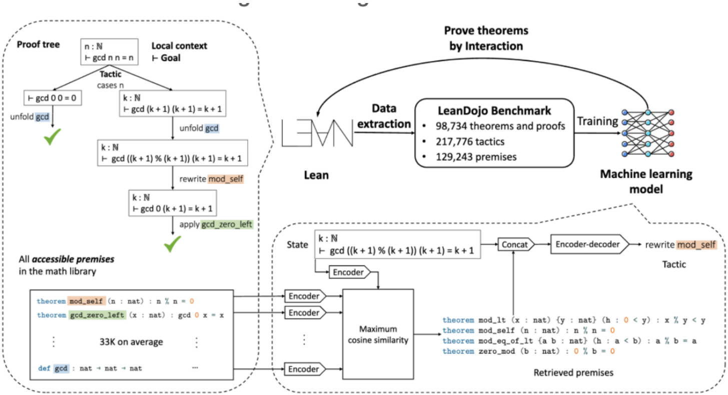


Figure: Retrieval augmented LLM proof generation (more from Thanosan later)

Latest research effort

DeepSeek Prover V2 (2025)

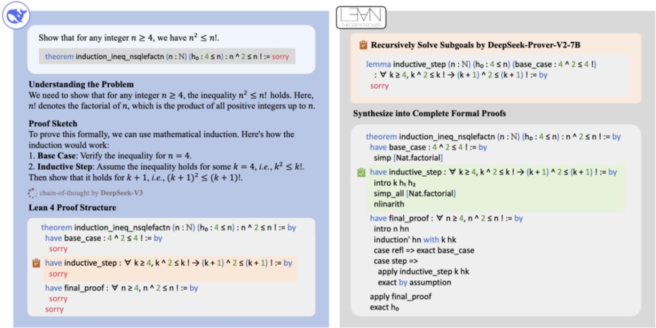


Figure: Chain of thought (LLM thinking), reinforcement learning (more from Ryan later)

State of the Art

Competition benchmarks to actual usefulness in research mathematics.

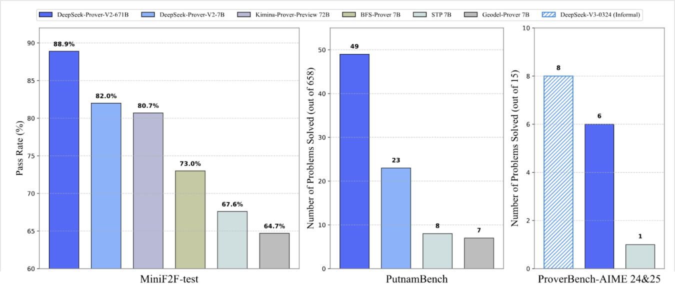


Figure: https://arxiv.org/pdf/2504.21801

Finding Proofs via Monte Carlo Tree Search

Adam Mawani

Challenge

- Constructing formal mathematical proofs is a search problem over a vast space of possible deductions

Challenge

- Constructing formal mathematical proofs is a search problem over a vast space of possible deductions
- LLMs show potential for generating informal reasoning steps, but lack reliability and structure

Challenge

- Constructing formal mathematical proofs is a search problem over a vast space of possible deductions
- LLMs show potential for generating informal reasoning steps, but lack reliability and structure
- Need a method to guide the proof search process toward correctness and completeness, while balancing exploration and exploitation

Challenge

- Constructing formal mathematical proofs is a search problem over a vast space of possible deductions
- LLMs show potential for generating informal reasoning steps, but lack reliability and structure
- Need a method to guide the proof search process toward correctness and completeness, while balancing exploration and exploitation
- Goal: Build a system that can automatically generate or complete proofs in Lean by efficiently navigating this search space

Challenge

- Constructing formal mathematical proofs is a search problem over a vast space of possible deductions
- LLMs show potential for generating informal reasoning steps, but lack reliability and structure
- Need a method to guide the proof search process toward correctness and completeness, while balancing exploration and exploitation
- Goal: Build a system that can automatically generate or complete proofs in Lean by efficiently navigating this search space

Existing Approaches

1. **Naive Search/Beam Search** Inefficient for deep or highly branching proof spaces and prone to local optima or premature convergence

Existing Approaches

1. **Naive Search/Beam Search** Inefficient for deep or highly branching proof spaces and prone to local optima or premature convergence
2. **Naive LLM Generation** Struggles with logical consistency and long-horizon planning, hallucinates steps, lacks built-in backtracking

Existing Approaches

1. **Naive Search/Beam Search** Inefficient for deep or highly branching proof spaces and prone to local optima or premature convergence
2. **Naive LLM Generation** Struggles with logical consistency and long-horizon planning, hallucinates steps, lacks built-in backtracking
3. **Reinforcement Learning** Promising scaling paradigm but but sample-inefficient without strong priors or structured guidance, sparse rewards

Monte Carlo Tree Search

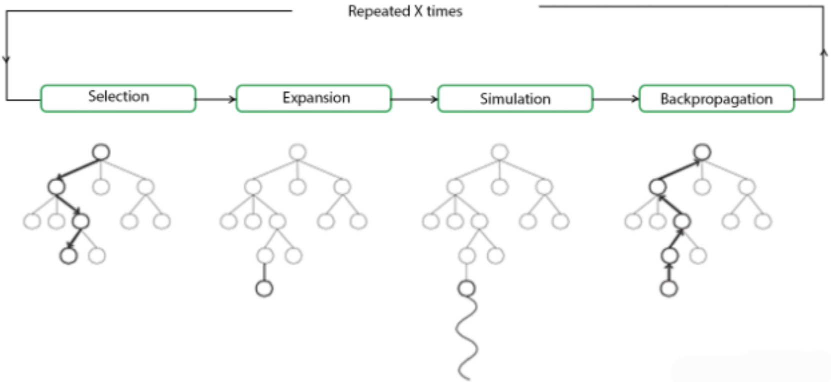
1. A heuristic search algorithm for decision processes that builds a search tree using random sampling of actions.
2. Success in related problems in solving and decision making
    1. AlphaGoZero (DeepMind, 2017)
    2. Planning problems through reinforcement learning (RL)

Monte Carlo Tree Search

Online Planning Algorithm

- **Selection** Traverse the tree by picking currently most optimal node to explore
- **Expansion** Add new nodes representing new/potential actions
- **Simulation** Run simulations from the new nodes to estimate quality of actions
- **Backpropagation** Update parent nodes with simulation results
- iterate until allocated compute exhuasted

Cartoon of MCTS



Node Selection Criteria

Exploration vs Exploitation

$$a_{\text{selection}} = \text{argmax}_{a \in A(s)} \left(\underbrace{Q(s,a)}_{exploit} + \underbrace{C\sqrt{\frac{\ln N(s)}{N(s,a)}}}_{explore} \right) \tag{1}$$

- $a_{\text{selection}}$ Action next
- $A(s)$ Set of actions available at state s.
- $Q(s,a)$ Sample average of reward of taking state-action pair.
- $N(s)$ Visitation frequency of state s
- $N(s,a)$ Visitation frequency of state-action pair
- C constant controlling explore-exploit tradeoff

Justifying MCTS for Proof Search

- Mathematical proofs form an enormous, branching tree of inference steps
- Mathematical proof construction can be viewed as a sequential decision-making process
- Each proof step is an action that transforms the current proof state into a new one
- Promising paradigm for inference time compute scaling to solve problems
- Coordinates exploration over valid proof steps using Lean or LLM evaluations
- Enables backtracking, self-correction, and goal-directed search

My Implementation

- Vertex: Partial proof states (e.g., current goal and context)
- Edges: Proof steps or tactics (e.g., applying a lemma)
- Leaves: Either a completed proof or an unprovable state
- The tree can be very wide (many tactics applicable at each step) and very deep (some proofs are long).

My Implementation

- State: Current proof goal + context (e.g., hypotheses, Lean environment)
- Actions: Possible next proof steps (theorems, tactics, lemmas)
- Transition Model: the LLM
- Reward Function: $+1$ if proof is correct as certified by Lean
- Sparse Reward Signal: Partial reward for subgoal reduction or valid deductions

Takeaways

1. Monte Carlo simulations - Allows low-cost exploration before full commitment
2. Statistical backpropagation - Focuses on the most promising proof paths over time
3. Anytime algorithm - Can return best-so-far proof paths even if stopped early
4. Integration-ready - Works well with LLM-based rollouts and formal validators

Challenges and Solutions

1. High branching factor - Prioritization via learned heuristics (LLM)
2. Simulation cost (in Lean) - Use fast approximate LLM simulations
3. Sparse rewards - Shaping rewards via subgoal closure
4. Large search space - Enhance tree search with multiagent systems

Reinforcement Learning with Lean Feedback

Ryan Li

Goal

Design a scalable reinforcement learning algorithm for training LLM for effective theorem proving.

Why Lean Alone is Not Enough?

1. Lean verifies proof correctness using a rich mathlib library, together with a useful set of tactics.
2. Human guidance is needed as automation plateaus at around 40%.
3. Lean suffers from search explosion due to branching factors, existing tactics works in the simpler cases.
4. Sparse rewards in Lean often lead to no learning until the end of MCTS.

Reinforcement Learning 101

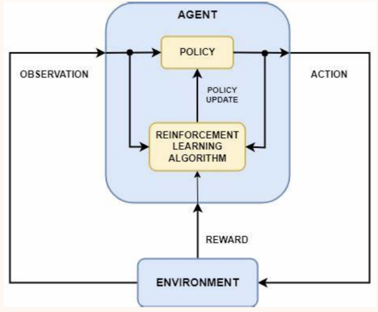


Figure: RL in a Nutshell

- Observe current state
- Select action through policy (LLM in our case)
- State transition based on action taken provides reward learning signal
- Reward updates policy, based on an optimization algorithm

The Mathematical Basis for RL

Definition (Markov Decision Process)

An MDP is defined as a tuple

$$\mathcal{M} = (S, A, T, d_0, r, \gamma)$$

where S is the set of states, A is the set actions, $T : S \times A \times S \to [0, 1]$ is a probablistic transitional kernel, $d_0 : S \to [0, 1]$ is the distribution of initial states, $r : S \times A \to \mathbb{R}$ is the reward function, and $0 < \gamma < 1$ is the discount factor.

RL as Optimization Problem

Fundamental Problem. Given a decision making policy π such as an LLM, we optimize for the expected discounted reward function

$$J(\pi) = \mathbb{E}_{\tau \sim \pi}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)\right]$$

Real-Life Constrained optimization, where constraint sets bounds to prevent large deviation of neural net parameter, to avoid reward hacking (learning to get to right answer with incorrect steps)!

Reward shaping

- Each Lean tactic $\approx$ help advance proof progress.
- If remaining Lean goals drops reward $+r$
- If lemma proven $+R$
- Full theorem R_{big}

RL Algorithms in LLM Training

- Reinforcement learning with human feedback (train neural net on human feedback to offer automated reward)
- Proximal policy optimization (PPO is a constrained optimization)
- Group Relative Policy Optimization (GRPO teach models to learn from the best among N possible solutions)

RL + MCTS

- Parallel tree search ≈ many graduate students working together
- Results update policy in LLM training
- Lean-Auto (rule-based): 40% MiniF2F success, no learning.
- ABEL (2024): 59.8%, AlphaZero-style RL + Aesop.
- DeepSeek-Prover-V1.5: 78%, MCTS + policy/value RL.
- DeepSeek-Prover-V2: 88.9%, huge LLM + synthetic sub-lemma RL.

Core Ideas

- LLM can absorbe lots of math knowledge and suggest what to do next in a math proof
- Conditioned on current proof state, it can generate tactics for next steps
- LLM can learn to prove theorems by training on human wrriten proofs but also by human-free RL
- Lean provides learning signals to train LLM policy through RL

LLM Post-training for Math Reasoning

Chinmay Jindal

Abstract

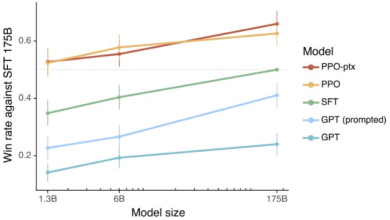
We

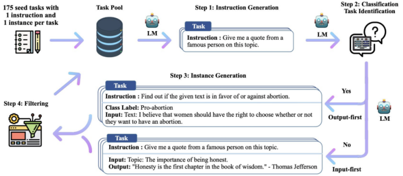
1. post-train a set of LLMs of difference sizes and architectures for theorem proving.
2. apply supervised finetuning (SFT), direct preference optimization (DPO), as well as group relevative policy optimization (GRPO) in the post-training pipeline.
3. leverage Lean for feedback and proof verification.

Supervised Finetuning

We SFT on small open source LLMs base models using a dataset $\mathcal{D}$ consisting of pairs (x, y) of problem and solution or theorem and proof. The optimization objective at this stage is

$$\mathcal{L}_{\mathsf{SFT}}(\theta) = -\mathbb{E}_{(x,y)\sim\mathcal{D}} \log p(y|x, \theta) \tag{2}$$





Reinforcement Learning with Human Feedback

Let $r(x, y)$ be the reward model. The Bradley-Terry preference model of human preferences stipulates that the probability in which outcome y_1 is preferred over y_2 (denoted $y_1 \succ y_2$) conditioned on x is

$$p(y_1 \succ y_2|x) = \frac{\exp r(x, y_1)}{\exp r(x, y_1) + \exp r(x, y_2)} \tag{3}$$

With a trained reward model, the policy model (LLM in our case) is aligned to human preferences by optimizing

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(y|x)}[r_\phi(x, y)] - \beta \mathsf{KL}[\pi_\theta(y|x)||\pi_{\mathsf{ref}}(y|x)] \tag{4}$$

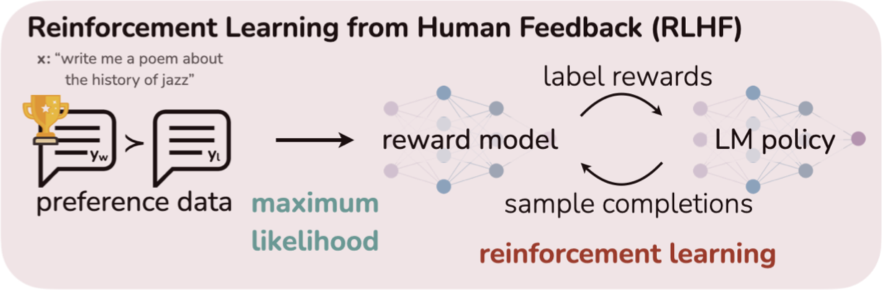


Figure: Illustrated preference learning through RLHF

Direct Preference Optimization

DPO is a simpler preference learning approach.

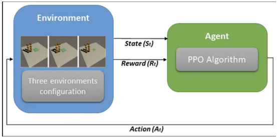
$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)} \right) \right]$$

$$\nabla_\theta \mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) =$$
$$-\beta \mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \bigg[\underbrace{\sigma(\hat{r}_\theta(x, y_l) - \hat{r}_\theta(x, y_w))}_{\text{higher weight when reward estimate is wrong}} \bigg[\underbrace{\nabla_\theta \log \pi(y_w \mid x)}_{\text{increase likelihood of } y_w} - \underbrace{\nabla_\theta \log \pi(y_l \mid x)}_{\text{decrease likelihood of } y_l} \bigg] \bigg]$$

Figure: Gradient based optimization of DPO objective.

Proximal Policy Optimization (PPO)

Maximising a surrogate objective while forcing each new policy to stay "proximal" (close) to the old one as measured by Kullback-Leibler divergence.



Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0, 1, 2, \ldots$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t),\ \ g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

end for

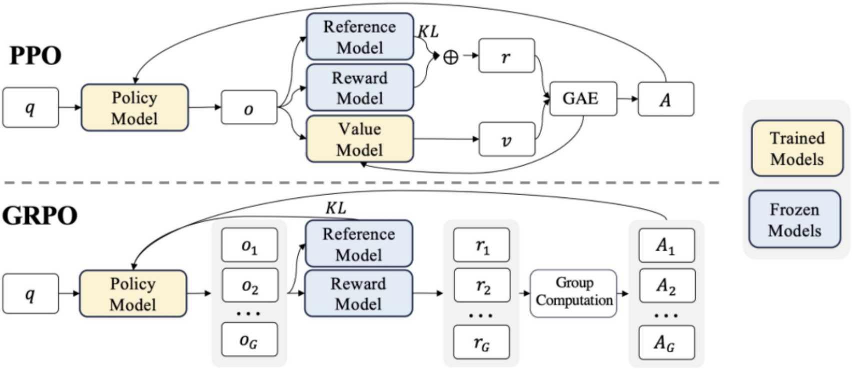
Group Relative Policy Optimization

- GRPO is a reinforcement learning post-training technique used in training DeepSeek-R1.
- Critic-free variant of PPO.
- In PPO a value function needs to be trained alongside the policy model.
- For each question q one samples a set of candidate ouputs $\{o_1, \ldots, o_G\}$ using the old policy. A reward model assigns scores $\mathbf{r} = \{r_1, \ldots, r_G\}$ to the outputs, and the normalized rewards are used to train the model $\hat{A}_i = \frac{r_i - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r})}$

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^{G} \sim \pi_{\theta_{old}}(O|q)]$$
$$\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\left\{\min\left[\frac{\pi_\theta(o_{i,t}|q,o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q,o_{i,<t})}\hat{A}_{i,t}, \text{clip}\left(\frac{\pi_\theta(o_{i,t}|q,o_{i,<t})}{\pi_{\theta_{old}}(o_{i,t}|q,o_{i,<t})}, 1-\varepsilon, 1+\varepsilon\right)\hat{A}_{i,t}\right] - \beta\mathbb{D}_{KL}\left[\pi_\theta \| \pi_{ref}\right]\right\}$$

Figure: The GRPO optimization objective

Comparison


PPO
q
Policy Model
o
Reference Model
Reward Model
Value Model
KL
⊕
r
v
GAE
A
GRPO
q
Policy Model
o_1
o_2
o_G
KL
Reference Model
Reward Model
r_1
r_2
r_G
Group Computation
A_1
A_2
A_G
Trained Models
Frozen Models

GRPO Illustration