

Large Language Model Reasoning

From Informal to Formal

Bohan Lyu

lyubh.cn

Tsinghua University

July 22, 2025

Contents

1. Introduction of LLM Reasoning

2. Training Techniques

- 2.1 An Overview
- 2.2 Supervised Fine-Tuning (SFT) based Methods
- 2.3 Group Relative Policy Optimization (GRPO)
- 2.4 Direct Preference Optimization (DPO)

3. Inference-Time Techniques

4. Autonomous Theorem Proving with Formal Language

- 4.1 Introduction to Formal Language
- 4.2 The Potential of Formal Language Prover
- 4.3 Development of Open-Source Provers

5. Conclusion and Open Challenges

Contents

1. Introduction of LLM Reasoning

2. Training Techniques

- 2.1 An Overview
- 2.2 Supervised Fine-Tuning (SFT) based Methods
- 2.3 Group Relative Policy Optimization (GRPO)
- 2.4 Direct Preference Optimization (DPO)

3. Inference-Time Techniques

4. Autonomous Theorem Proving with Formal Language

- 4.1 Introduction to Formal Language
- 4.2 The Potential of Formal Language Prover
- 4.3 Development of Open-Source Provers

5. Conclusion and Open Challenges

Chain of Thought

CoT: Chain of Thought, “Let’s think step by step”.

(a) Few-shot

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) The answer is 8. X

(b) Few-shot-CoT

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are $16 / 2 = 8$ golf balls. Half of the golf balls are blue. So there are $8 / 2 = 4$ blue golf balls. The answer is 4. ✓

(c) Zero-shot

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A: The answer (arabic numerals) is

(Output) 8 X

(d) Zero-shot-CoT (Ours)

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?

A: **Let's think step by step.**

(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓

Figure: Prompting Chain of Thought.

Long CoT

OpenAI-O1, DeepSeek-R1.

Question: If $a > 1$, then the sum of the real solutions of $\sqrt{a - \sqrt{a+x}} = x$ is equal to

Response: <think>

To solve the equation $\sqrt{a - \sqrt{a+x}} = x$, let's start by squaring both ...

$$(\sqrt{a - \sqrt{a+x}})^2 = x^2 \implies a - \sqrt{a+x} = x^2.$$

Rearrange to isolate the inner square root term:

$$(a - x^2)^2 = a + x \implies a^2 - 2ax^2 + (x^2)^2 = a + x \implies x^4 - 2ax^2 - x + (a^2 - a) = 0$$

...

Wait, wait. Wait. That's an aha moment I can flag here.

Let's reevaluate this step-by-step to identify if the correct sum can be ...

We started with the equation:

$$\sqrt{a - \sqrt{a+x}} = x$$

First, let's square both sides:

$$a - \sqrt{a+x} = x^2 \implies \sqrt{a+x} = a - x^2$$

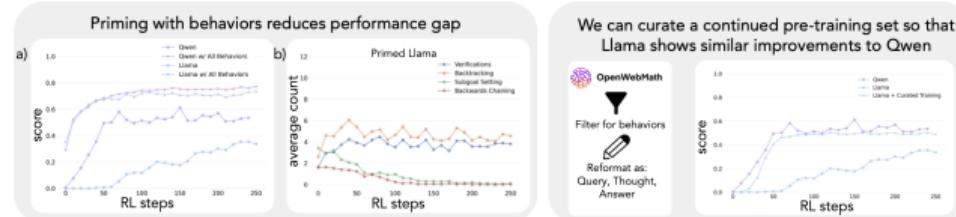
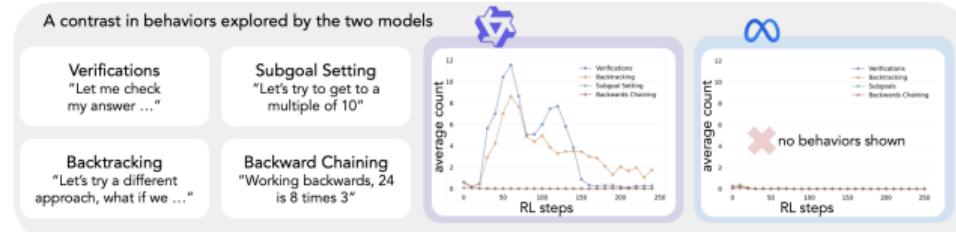
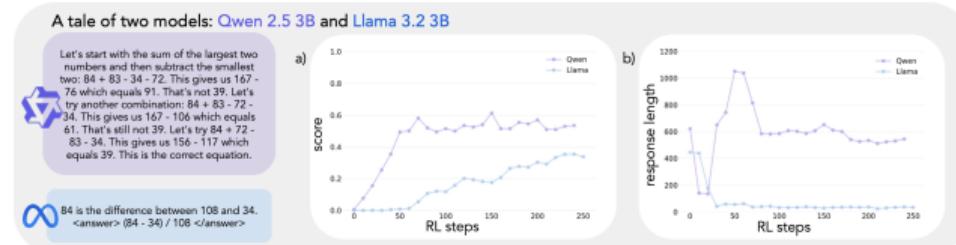
Next, I could square both sides again, treating the equation: ...

...

Figure: The “Aha Moment” observed by DeepSeek-R1.

Why can DeepSeek-R1 succeed? (1/2)

- Reinforcement learning based on the base model (with warm-up).
- Reflection behaviors in the base model.



Why can DeepSeek-R1 succeed? (2/2)

What else?

Why can DeepSeek-R1 succeed? (2/2)

What else?

Model	AIME 2024		MATH-500	GPQA Diamond	LiveCodeBench
	pass@1	cons@64	pass@1	pass@1	pass@1
QwQ-32B-Preview	50.0	60.0	90.6	54.5	41.9
DeepSeek-R1-Zero-Qwen-32B	47.0	60.0	91.6	55.0	40.2
DeepSeek-R1-Distill-Qwen-32B	72.6	83.3	94.3	62.1	57.2

Table: Comparison of distilled and RL Models on Reasoning-Related Benchmarks.

Why can DeepSeek-R1 succeed? (2/2)

What else?

Model	AIME 2024		MATH-500	GPQA Diamond	LiveCodeBench
	pass@1	cons@64	pass@1	pass@1	pass@1
QwQ-32B-Preview	50.0	60.0	90.6	54.5	41.9
DeepSeek-R1-Zero-Qwen-32B	47.0	60.0	91.6	55.0	40.2
DeepSeek-R1-Distill-Qwen-32B	72.6	83.3	94.3	62.1	57.2

Table: Comparison of distilled and RL Models on Reasoning-Related Benchmarks.

Rethink the position of academia in the development of LLM.
Rethink our position in academia.

Contents

1. Introduction of LLM Reasoning

2. Training Techniques

- 2.1 An Overview
- 2.2 Supervised Fine-Tuning (SFT) based Methods
- 2.3 Group Relative Policy Optimization (GRPO)
- 2.4 Direct Preference Optimization (DPO)

3. Inference-Time Techniques

4. Autonomous Theorem Proving with Formal Language

- 4.1 Introduction to Formal Language
- 4.2 The Potential of Formal Language Prover
- 4.3 Development of Open-Source Provers

5. Conclusion and Open Challenges

Contents

1. Introduction of LLM Reasoning

2. Training Techniques

2.1 An Overview

2.2 Supervised Fine-Tuning (SFT) based Methods

2.3 Group Relative Policy Optimization (GRPO)

2.4 Direct Preference Optimization (DPO)

3. Inference-Time Techniques

4. Autonomous Theorem Proving with Formal Language

4.1 Introduction to Formal Language

4.2 The Potential of Formal Language Prover

4.3 Development of Open-Source Provers

5. Conclusion and Open Challenges

An Overview

What are the techniques that can boost LLM Reasoning?

- Supervised Fine-Tuning (SFT) based methods: data is the key.
 - Human Annotation, LLM Distillation.
 - Self-Evolve: Reject Sampling Fine-tuning.
- Reinforcement Learning (RL) based methods:
 - Data: Filtering by Difficulty, Curriculum Learning.
 - Pipeline: Reinforcement Learning with Human Feedback (RLHF)&Reinforcement Learning with Verifiable Reward (RLVR), Process Reward Model (PRM)&Outcome Reward Model (ORM).
 - RL Algorithm: Proximal Policy Optimization (PPO), Group Relative Policy Optimization (GRPO), REINFORCE++.
- Direct Preference Optimization (DPO) and its variants, e.g., Iterative DPO, Step DPO.

Contents

1. Introduction of LLM Reasoning

2. Training Techniques

- 2.1 An Overview
- 2.2 Supervised Fine-Tuning (SFT) based Methods
- 2.3 Group Relative Policy Optimization (GRPO)
- 2.4 Direct Preference Optimization (DPO)

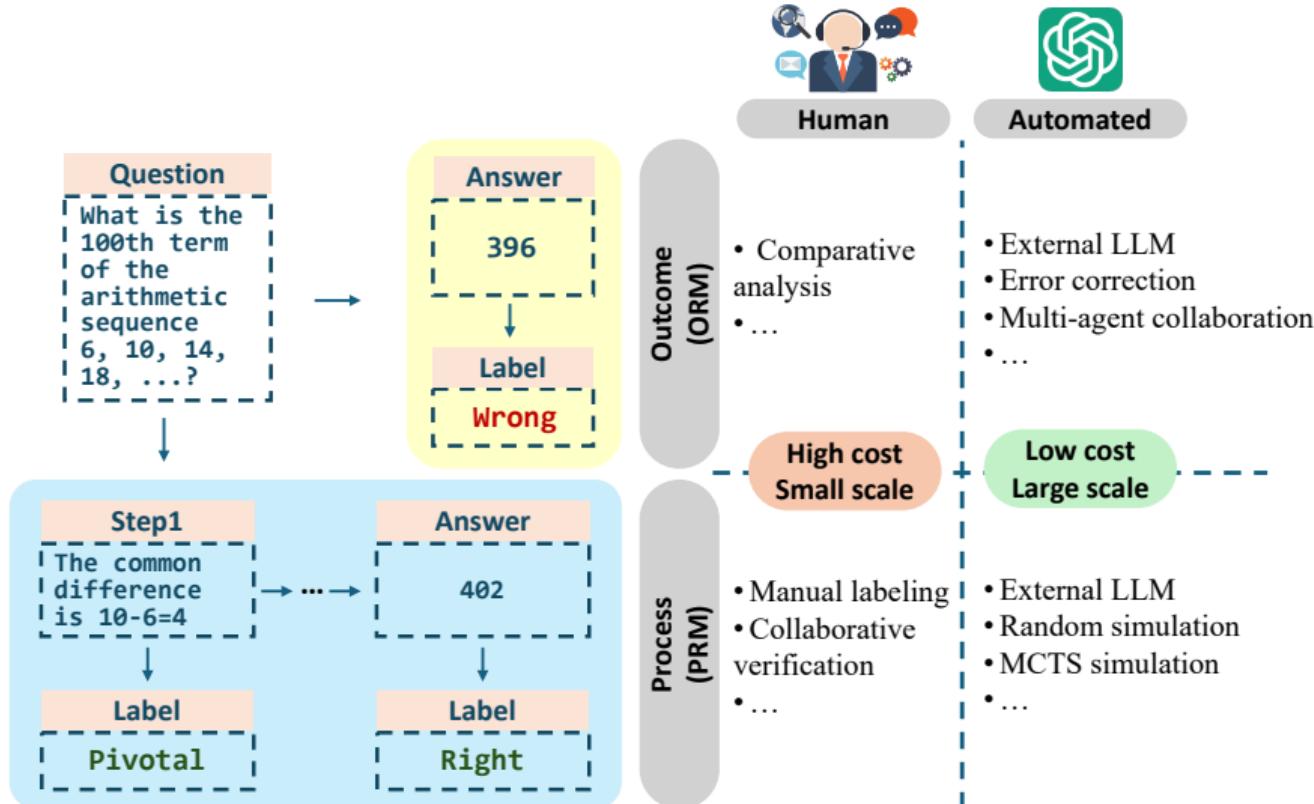
3. Inference-Time Techniques

4. Autonomous Theorem Proving with Formal Language

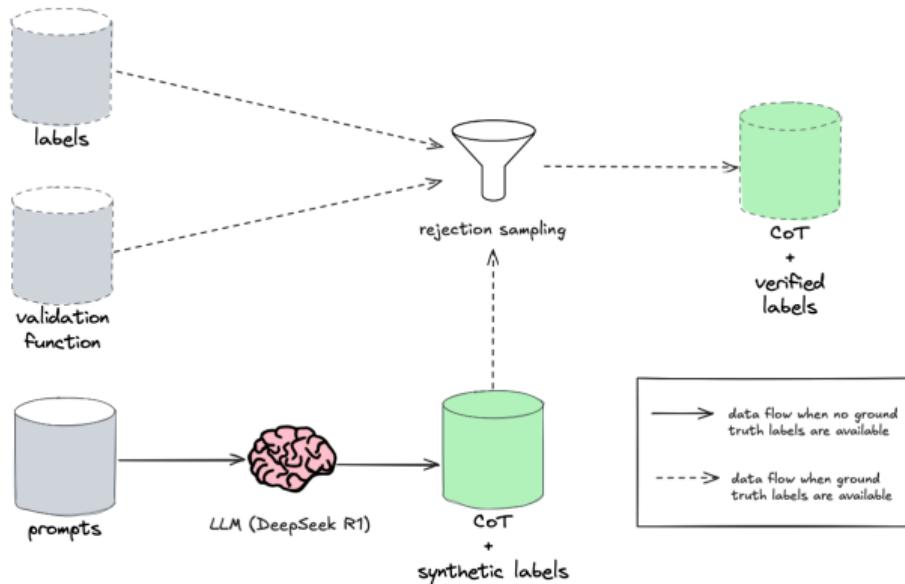
- 4.1 Introduction to Formal Language
- 4.2 The Potential of Formal Language Prover
- 4.3 Development of Open-Source Provers

5. Conclusion and Open Challenges

Human Annotation and LLM Automation



Reject Sampling Fine-tuning



Criteria:

- Unsupervised:
 - Semantic Entropy
 - Confidence
- Supervised:
 - Ground Truth (math)
 - Test Cases (code)
 - Compiler (formal math)
- Reward Model

$$\max_{\theta} J_{RFT}(\theta) = \mathbb{E}_{x \sim D} \sum_a \pi_{\theta_{\text{old}}}(a|x) \cdot \mathbf{1}[a \in H(x)] \cdot \log \pi_{\theta}(a|x) \quad (1)$$

Contents

1. Introduction of LLM Reasoning

2. Training Techniques

- 2.1 An Overview
- 2.2 Supervised Fine-Tuning (SFT) based Methods
- 2.3 Group Relative Policy Optimization (GRPO)
- 2.4 Direct Preference Optimization (DPO)

3. Inference-Time Techniques

4. Autonomous Theorem Proving with Formal Language

- 4.1 Introduction to Formal Language
- 4.2 The Potential of Formal Language Prover
- 4.3 Development of Open-Source Provers

5. Conclusion and Open Challenges

The GRPO Objective Function

For each question q , GRPO samples a group of outputs $\{o_1, o_2, \dots, o_G\}$ and maximizes the following objective:

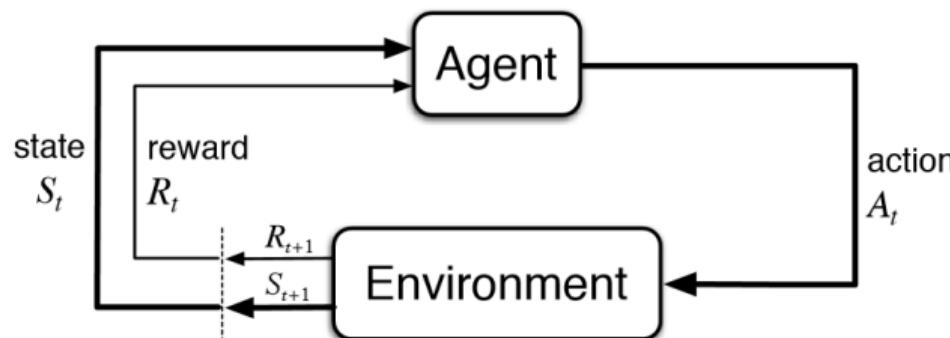
$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}[q \sim P(Q), \{o_i\}_{i=1}^G \sim \pi_{\theta_{old}}(O|q)]$$

$$\frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \left\{ \min \left[r_{i,t}(\theta) \hat{A}_{i,t}, \text{clip}(r_{i,t}(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_{i,t} \right] - \beta \mathbb{D}_{KL} [\pi_\theta || \pi_{ref}] \right\}$$

- $\hat{A}_i = R(o_i) - \frac{1}{G} \sum_{j=1}^G R(o_j)$ is the advantage calculated based on **relative rewards** within the group, not a learned value function.
- The KL-divergence penalty with a reference policy π_{ref} is added directly to the loss to stabilize training, rather than being part of the reward signal.

Basics of RL (1/4)

- **Markov Decision Process (MDP)**: Formal framework for RL defined by tuple (S, A, P, R, γ)
 - S : Set of states
 - A : Set of actions
 - $P(s'|s, a)$: State transition probability
 - $R(s, a, s')$: Reward function
 - $\gamma \in [0, 1]$: Discount factor
- **Trajectory τ** : Sequence of states and actions $\tau = (s_0, a_0, s_1, a_1, \dots)$



Basics of RL (2/4)

- **Reward Function** $r(s, a, s')$: Immediate feedback signal after taking action a in state s and transitioning to state s' .
- Often simplified as $r(s, a)$ when the next state is implicit.
- **Return** G_t : Cumulative discounted reward from time step t

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$$

- **Discount Factor** γ : Controls the importance of future rewards.
 - $\gamma \approx 0$: Myopic evaluation (immediate rewards)
 - $\gamma \approx 1$: Far-sighted evaluation (long-term rewards)

Basics of RL (3/4)

- **Policy** $\pi_\theta(a|s)$: A function, parameterized by θ , that maps a state s to a distribution over actions a . It defines the agent's behavior.
- **Objective Function** $J(\theta)$: The expected cumulative discounted reward obtained by following policy π_θ . Our goal is to find parameters θ that maximize this.

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

- **Optimal Policy** π^* : Policy that maximizes the expected return

$$\pi^* = \arg \max_{\pi} J(\pi)$$

Basics of RL (4/4)

- **Value Function** $V^\pi(s)$: The expected return starting from state s and then following policy π .

$$V^\pi(s) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=k}^{\infty} \gamma^{t-k} r(s_t, a_t) | s_k = s \right]$$

- **Action-Value Function** $Q^\pi(s, a)$: The expected return after taking action a in state s and then following policy π .

$$Q^\pi(s, a) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=k}^{\infty} \gamma^{t-k} r(s_t, a_t) | s_k = s, a_k = a \right]$$

- **Advantage Function** $A^\pi(s, a)$: How much better is action a compared to the average action in state s

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$$

Temporal difference (TD) advantage

We can expand $Q^\pi(s_t, a_t)$ using its definition:

$$\begin{aligned} Q^\pi(s_t, a_t) &= \mathbb{E}_{\tau \sim \pi} \left[\sum_{k=t}^{\infty} \gamma^{k-t} r(s_k, a_k) | s_t, a_t \right] \\ &= \mathbb{E}_{\tau \sim \pi} \left[r(s_t, a_t) + \gamma \sum_{k=t+1}^{\infty} \gamma^{k-(t+1)} r(s_k, a_k) | s_t, a_t \right] \\ &= r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim P(\cdot | s_t, a_t)} \left[\mathbb{E}_{\tau \sim \pi} \left[\sum_{k=t+1}^{\infty} \gamma^{k-(t+1)} r(s_k, a_k) | s_{t+1} \right] \right] \end{aligned}$$

The inner expectation is precisely the definition of $V^\pi(s_{t+1})$:

$$Q^\pi(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim P(\cdot | s_t, a_t)} [V^\pi(s_{t+1})]$$

Temporal difference (TD) advantage

Substituting back into the advantage function:

$$\begin{aligned} A^\pi(s_t, a_t) &= Q^\pi(s_t, a_t) - V^\pi(s_t) \\ &= r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim P(\cdot | s_t, a_t)} [V^\pi(s_{t+1})] - V^\pi(s_t) \end{aligned}$$

For a specific transition (s_t, a_t, s_{t+1}) , this gives us:

$$A^\pi(s_t, a_t) = r(s_t, a_t) + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$$

This is the one-step temporal difference (TD) advantage.

The TD-Advantage

A key identity connects the advantage function to the reward and value functions. This is often used for practical estimation.

$$A^\pi(s_t, a_t) = r(s_t, a_t) + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)$$

This is the one-step or TD-Advantage. It forms the basis for our proof.

The Performance Difference Lemma

Theorem (Performance Difference Lemma)

*The difference in performance between a new policy $\pi_{\theta'}$ and an old policy π_{θ} can be expressed as the expected advantage of the old policy, where the expectation is taken over the states and actions visited by the **new policy** $\pi_{\theta'}$.*

$$J(\theta') - J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta'}} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi_{\theta}}(s_t, a_t) \right]$$

We want to prove: $J(\theta') - J(\theta) = \mathbb{E}_{\tau \sim \pi_{\theta'}} \left[\sum_{t=0}^{\infty} \gamma^t A^{\pi_{\theta}}(s_t, a_t) \right]$

The core idea is to express the reward r_t in terms of the advantage function $A_t^{\pi_{\theta}}$ and the value function $V^{\pi_{\theta}}$. We will then substitute this into the definition of $J(\theta')$ and simplify.

Step 1 - Express Reward via Advantage

We start from the identity for the one-step advantage:

$$A^{\pi_\theta}(s_t, a_t) = r_t + \gamma V^{\pi_\theta}(s_{t+1}) - V^{\pi_\theta}(s_t)$$

By rearranging this equation, we can express the reward r_t as:

$$r_t = A^{\pi_\theta}(s_t, a_t) + V^{\pi_\theta}(s_t) - \gamma V^{\pi_\theta}(s_{t+1})$$

This allows us to introduce the advantage function into our objective.

Step 2 - Substitute into the Objective

Now, we take the definition of the objective for the new policy, $J(\theta')$:

$$J(\theta') = \mathbb{E}_{\tau \sim \pi_{\theta'}} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$$

And substitute our expression for r_t from the previous step:

$$J(\theta') = \mathbb{E}_{\tau \sim \pi_{\theta'}} \left[\sum_{t=0}^{\infty} \gamma^t (A^{\pi_\theta}(s_t, a_t) + V^{\pi_\theta}(s_t) - \gamma V^{\pi_\theta}(s_{t+1})) \right]$$

The next step is to simplify the sum.

Step 3 - The Telescoping Sum

Let's focus on the value function terms inside the expectation:

$$\sum_{t=0}^{\infty} \gamma^t V^{\pi_\theta}(s_t) - \sum_{t=0}^{\infty} \gamma^{t+1} V^{\pi_\theta}(s_{t+1})$$

Let's write out the first few terms:

- First sum: $(\gamma^0 V_0 + \gamma^1 V_1 + \gamma^2 V_2 + \dots)$
- Second sum: $(\gamma^1 V_1 + \gamma^2 V_2 + \gamma^3 V_3 + \dots)$

When we subtract the second sum from the first, all terms cancel out except for the very first one:

$$(\cancel{\gamma^0 V_0} + \cancel{\gamma^1 V_1} + \dots) - (\cancel{\gamma^1 V_1} + \dots) = \gamma^0 V^{\pi_\theta}(s_0) = V^{\pi_\theta}(s_0)$$

Step 4 - Finalizing the Proof

Substituting the result of the telescoping sum back into our expression for $J(\theta')$:

$$J(\theta') = \mathbb{E}_{\tau \sim \pi_{\theta'}} \left[\sum_{t=0}^{\infty} \gamma^t A_t^{\pi_\theta}(s_t, a_t) + V^{\pi_\theta}(s_0) \right]$$

We can split the expectation:

$$J(\theta') = \mathbb{E}_{\tau \sim \pi_{\theta'}} \left[\sum_{t=0}^{\infty} \gamma^t A_t^{\pi_\theta} \right] + \mathbb{E}_{\tau \sim \pi_{\theta'}} [V^{\pi_\theta}(s_0)]$$

Recognizing that $\mathbb{E}_{\tau \sim \pi_{\theta'}} [V^{\pi_\theta}(s_0)] = \mathbb{E}_{s_0} [V^{\pi_\theta}(s_0)] = J(\theta)$:

$$J(\theta') = \mathbb{E}_{\tau \sim \pi_{\theta'}} \left[\sum_{t=0}^{\infty} \gamma^t A_t^{\pi_\theta} \right] + J(\theta)$$

Rearranging gives the final lemma. Q.E.D.

From Lemma to Surrogate Objective

The performance difference lemma is great, but it has a major practical issue:

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

The expectation is over trajectories from the **new** policy $\pi_{\theta'}$, which we don't have yet. We cannot sample from it to compute the expectation.

Importance Sampling: If we want to compute:

$$\mathbb{E}_{x \sim p(x)}[f(x)] = \int f(x)p(x)dx$$

but we can only sample from another distribution $q(x)$ (for example, $q(x)$ is a distribution that is easier for us to sample from), we can write:

$$\mathbb{E}_{x \sim p(x)}[f(x)] = \int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx = \mathbb{E}_{x \sim q(x)} \left[f(x)\frac{p(x)}{q(x)} \right]$$

Objective Function

The final objective function that we optimize in practice is based on this importance sampling formulation. Let $r_t(\theta') = \frac{\pi_{\theta'}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ be the probability ratio. The objective is:

$$L(\theta') = \mathbb{E}_t \left[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A^{\pi_{\theta_{old}}}(s_t, a_t) \right]$$

where the expectation \mathbb{E}_t indicates an average over a finite batch of samples collected using the old policy $\pi_{\theta_{old}}$.

The PPO/TRPO Surrogate Objective - 2/2

Directly maximizing this objective is unstable. Note $r_t(\theta') = \frac{\pi_{\theta'}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$.

- **TRPO** formulates policy improvement as a constrained optimization problem:

$$\max_{\theta'} \mathbb{E}_t \left[\frac{\pi_{\theta'}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A^{\pi_{\theta_{old}}}(s_t, a_t) \right]$$

$$\text{s.t. } \mathbb{E}_t [KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_{\theta'}(\cdot|s_t)]] \leq \delta$$

This **KL-divergence constraint** ensures the new policy stays close to the old one.

- **PPO** has two variants, where $A_t = A^{\pi_{\theta_{old}}}(s_t, a_t)$ is the advantage estimate:
 - **Clipped objective** which is simpler to implement:

$$L^{CLIP}(\theta') = \mathbb{E}_t [\min(r_t(\theta')A_t, \text{clip}(r_t(\theta'), 1 - \epsilon, 1 + \epsilon)A_t)]$$

- **KL penalty variant** which adds a penalty term:

$$L^{KL}(\theta') = \mathbb{E}_t [r_t(\theta')A_t - \beta \cdot KL[\pi_{\theta_{old}}(\cdot|s_t), \pi_{\theta'}(\cdot|s_t)]]$$

Algorithm Implementation

What are the probabilities, states, actions, and rewards in LLMs (consider token-wise things)? How to implement them?

Example:

```
1 def compute_reward(r: Union[torch.Tensor, float], kl_coef: float, kl: torch.Tensor, action_mask: Optional[torch.Tensor] = None, ) -> torch.Tensor:
2     # Step 1: Calculate the KL penalty term for each step (corresponds to -beta * KL_t).
3     kl_reward = -kl_coef * kl
4     # Step 2: Construct the sparse final reward. The goal is to create a tensor where the total
        sequence reward 'r' is placed only at the position of the last valid token.
5     eos_indices = action_mask.size(1) - 1 - \
                    action_mask.long().flip(0).argmax(dim=1, keepdim=True)
6     last_reward = torch.zeros_like(kl)
7     last_reward.scatter_(dim=1, index=eos_indices, src=r.unsqueeze(1).to(kl.dtype))
8     # Step 3: Add the KL penalty and the final reward to get the final token-level reward r_t.
9     reward = last_reward + kl_reward
10    return reward
11
```

Please read the source code of OpenRLHF for more details.

Engineering Challenges

Packages: Ray, DeepSpeed, vllm ...

System Complexity

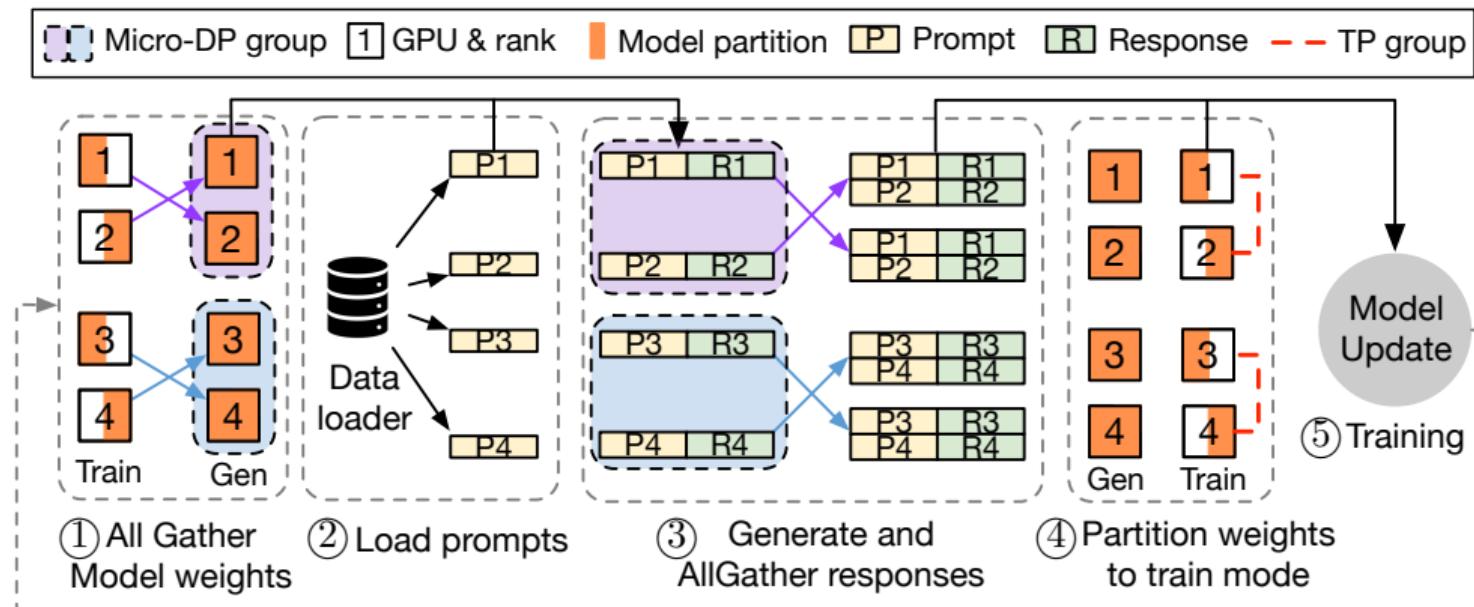


Figure: 3D-HybridEngine workflow in VeRL.

Contents

1. Introduction of LLM Reasoning

2. Training Techniques

- 2.1 An Overview
- 2.2 Supervised Fine-Tuning (SFT) based Methods
- 2.3 Group Relative Policy Optimization (GRPO)
- 2.4 Direct Preference Optimization (DPO)

3. Inference-Time Techniques

4. Autonomous Theorem Proving with Formal Language

- 4.1 Introduction to Formal Language
- 4.2 The Potential of Formal Language Prover
- 4.3 Development of Open-Source Provers

5. Conclusion and Open Challenges

Step 1: The RLHF Objective Function

We start with the standard RLHF objective, which is to maximize the reward while a KL-divergence penalty prevents the model from straying too far from a reference model.

$$\max_{\pi} \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi(y|x)} [r(x, y)] - \beta \mathbb{D}_{\text{KL}} [\pi(y | x) || \pi_{\text{ref}}(y | x)]$$

- $r(x, y)$ is a (usually learned) reward function that scores how good a completion y is for a given prompt x .
- π_{ref} is a fixed reference policy (usually the SFT model).
- π is the policy we want to optimize.
- β is a hyperparameter that controls the strength of the KL penalty.

This objective seeks a policy π that balances getting high rewards with staying close to π_{ref} .

Step 2: Deriving the Optimal Policy's Form - 1/2

Let's find the policy π that maximizes the objective. We can rewrite the objective for a single prompt x as:

$$\begin{aligned} & \max_{\pi} \sum_y \pi(y|x) r(x, y) - \beta \left(\sum_y \pi(y|x) \log \frac{\pi(y|x)}{\pi_{\text{ref}}(y|x)} \right) \\ &= \min_{\pi} \sum_y \pi(y|x) \left(\beta \log \frac{\pi(y|x)}{\pi_{\text{ref}}(y|x)} - r(x, y) \right) \\ &= \min_{\pi} \sum_y \pi(y|x) \left(\beta \log \frac{\pi(y|x)}{\pi_{\text{ref}}(y|x) \exp(r(x, y)/\beta)} \right) \end{aligned}$$

Step 2: Deriving the Optimal Policy's Form - 2/2

Let's define a new distribution $q(y|x) = \frac{1}{Z(x)}\pi_{\text{ref}}(y|x)\exp(r(x,y)/\beta)$, where $Z(x)$ is the normalizer. The objective becomes:

$$\begin{aligned} & \min_{\pi} \sum_y \pi(y|x) \left(\beta \log \frac{\pi(y|x)}{q(y|x)Z(x)} \right) \\ &= \min_{\pi} \beta \left(\sum_y \pi(y|x) \log \frac{\pi(y|x)}{q(y|x)} - \log Z(x) \right) \\ &= \min_{\pi} \beta (\mathbb{D}_{\text{KL}}(\pi||q) - \log Z(x)) \end{aligned}$$

Since $\log Z(x)$ is constant with respect to π , the minimum is achieved when $\mathbb{D}_{\text{KL}}(\pi||q) = 0$, which means $\pi = q$.

Step 3: The Closed-Form Optimal Policy

From the derivation on the previous slide, we found that the optimal policy π_r is equal to the distribution $q(y|x)$. This gives us the closed-form solution:

$$\pi_r(y | x) = \frac{1}{Z(x)} \pi_{\text{ref}}(y | x) \exp\left(\frac{1}{\beta} r(x, y)\right)$$

where $Z(x)$ is the partition function, defined as:

$$Z(x) = \sum_y \pi_{\text{ref}}(y | x) \exp\left(\frac{1}{\beta} r(x, y)\right)$$

The Problem

This solution is difficult to use in practice because computing $Z(x)$ requires summing over all possible sequences y in the entire vocabulary space, which is computationally intractable.

Step 4: The Key Trick - Reparameterizing the Reward with the Policy

DPO's core idea is to cleverly perform an algebraic transformation to express the reward function $r(x, y)$ in terms of the policies π_r and π_{ref} . We take the logarithm of both sides of the optimal policy equation and rearrange:

$$\log \pi_r(y | x) = -\log Z(x) + \log \pi_{\text{ref}}(y | x) + \frac{1}{\beta} r(x, y)$$

Solving for the reward function, we get:

$$r(x, y) = \beta \left(\log \frac{\pi_r(y | x)}{\pi_{\text{ref}}(y | x)} \right) + \beta \log Z(x)$$

This equation establishes a direct link between a reward function and its corresponding optimal policy.

Step 5: Introducing the Human Preference Model

RLHF typically uses the Bradley-Terry model to model human preferences. This model assumes that the probability p^* of a human preferring a response y_w (winner) over y_l (loser) can be expressed by a logistic function of the difference in their latent reward scores:

$$p^*(y_w \succ y_l | x) = \sigma(r(x, y_w) - r(x, y_l))$$

where σ is the sigmoid function.

Key Observation

This preference model depends only on the **difference in rewards** between the two candidates, not their absolute values.

Step 6: Eliminating the Intractable Partition Function

Now, we substitute the reward function expression from Step 4 into the preference model from Step 5. Let's compute the reward difference:

$$\begin{aligned} r(x, y_w) - r(x, y_l) &= \left(\beta \log \frac{\pi_r(y_w|x)}{\pi_{\text{ref}}(y_w|x)} + \beta \log Z(x) \right) \\ &\quad - \left(\beta \log \frac{\pi_r(y_l|x)}{\pi_{\text{ref}}(y_l|x)} + \beta \log Z(x) \right) \end{aligned}$$

As you can see, the intractable partition function term $\beta \log Z(x)$, which depends only on x , is **perfectly cancelled out** during the subtraction!

$$r(x, y_w) - r(x, y_l) = \beta \left(\log \frac{\pi_r(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \log \frac{\pi_r(y_l|x)}{\pi_{\text{ref}}(y_l|x)} \right)$$

Step 7: The Final DPO Loss Function

By eliminating $Z(x)$, we can express the preference probability directly in terms of the policy π_θ :

$$p(y_w \succ y_l | x) = \sigma \left(\beta \log \frac{\pi_\theta(y_w | x)}{\pi_{\text{ref}}(y_w | x)} - \beta \log \frac{\pi_\theta(y_l | x)}{\pi_{\text{ref}}(y_l | x)} \right)$$

Our goal is to find a policy π_θ that maximizes this probability over a given dataset of preferences $\mathcal{D} = \{(x, y_w, y_l)\}$. This is equivalent to minimizing the negative log-likelihood, which gives us the DPO loss function:

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

This is just a simple binary cross-entropy loss that can be optimized directly with gradient descent.

Contents

1. Introduction of LLM Reasoning

2. Training Techniques

- 2.1 An Overview
- 2.2 Supervised Fine-Tuning (SFT) based Methods
- 2.3 Group Relative Policy Optimization (GRPO)
- 2.4 Direct Preference Optimization (DPO)

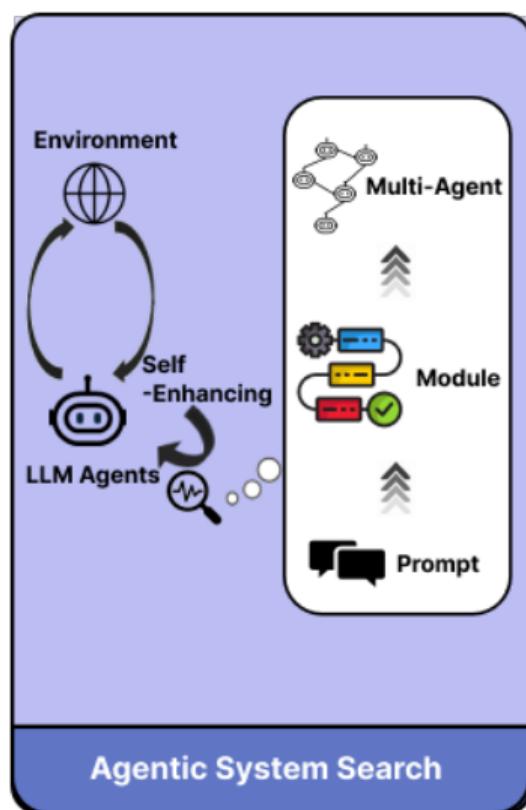
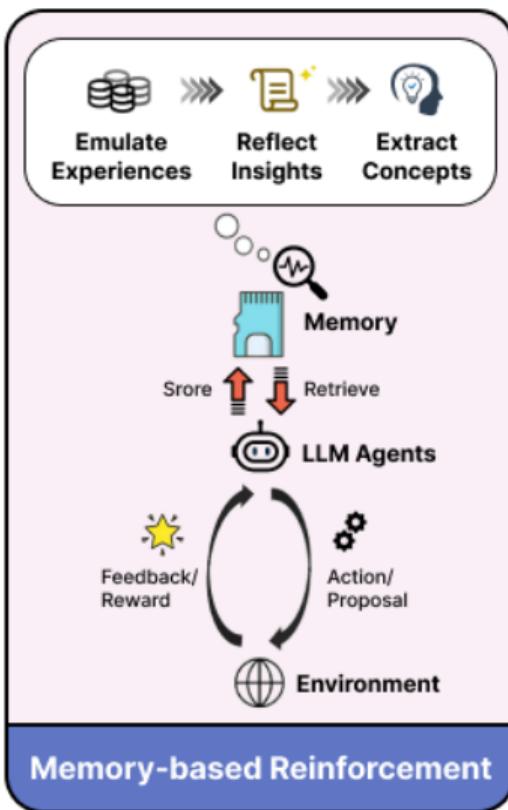
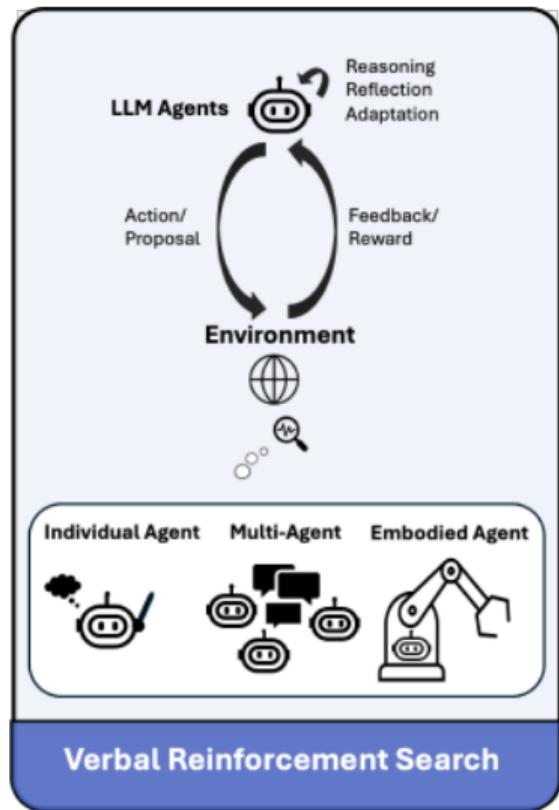
3. Inference-Time Techniques

4. Autonomous Theorem Proving with Formal Language

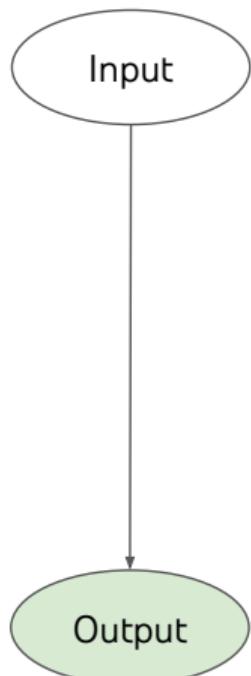
- 4.1 Introduction to Formal Language
- 4.2 The Potential of Formal Language Prover
- 4.3 Development of Open-Source Provers

5. Conclusion and Open Challenges

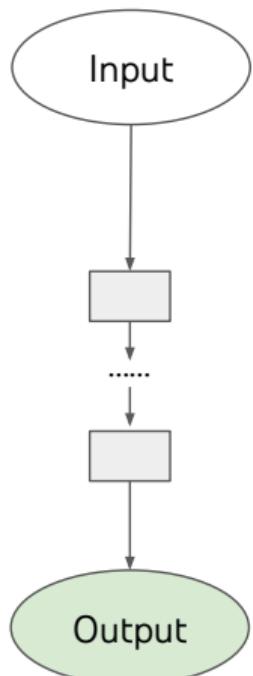
Agentic Methods



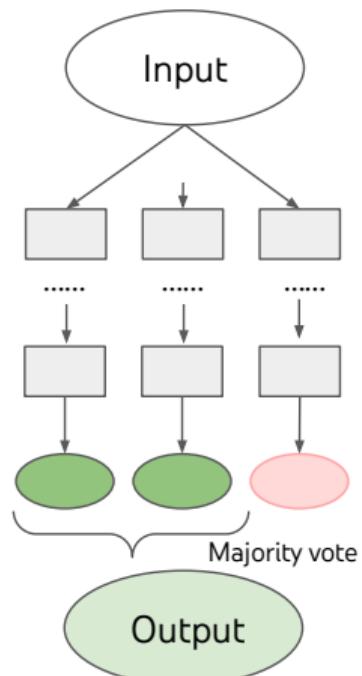
Search Methods: Tree of Thought



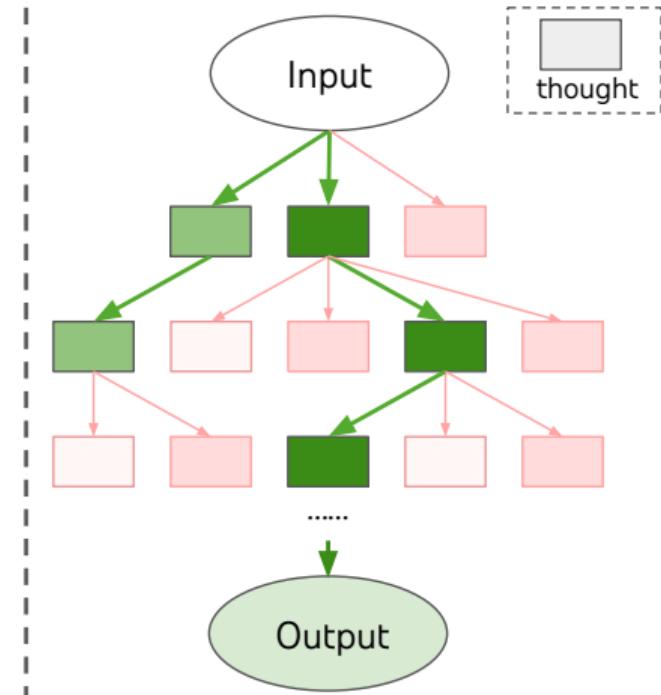
(a) Input-Output
Prompting (IO)



(c) Chain of Thought
Prompting (CoT)



(c) Self Consistency
with CoT (CoT-SC)



(d) Tree of Thoughts (ToT)

Search Methods: Reasoning via Planning

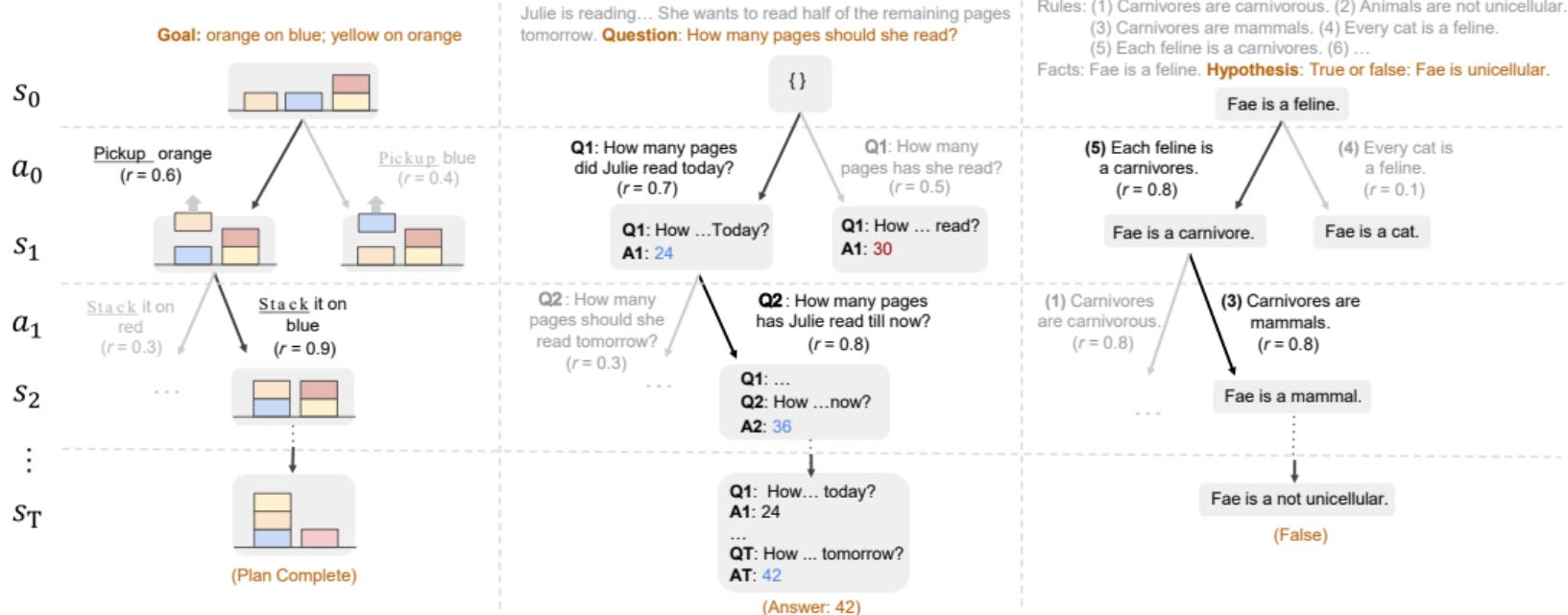


Figure: Reasoning with Language Model is Planning with World Model

Contents

1. Introduction of LLM Reasoning

2. Training Techniques

- 2.1 An Overview
- 2.2 Supervised Fine-Tuning (SFT) based Methods
- 2.3 Group Relative Policy Optimization (GRPO)
- 2.4 Direct Preference Optimization (DPO)

3. Inference-Time Techniques

4. Autonomous Theorem Proving with Formal Language

- 4.1 Introduction to Formal Language
- 4.2 The Potential of Formal Language Prover
- 4.3 Development of Open-Source Provers

5. Conclusion and Open Challenges

Contents

1. Introduction of LLM Reasoning

2. Training Techniques

- 2.1 An Overview
- 2.2 Supervised Fine-Tuning (SFT) based Methods
- 2.3 Group Relative Policy Optimization (GRPO)
- 2.4 Direct Preference Optimization (DPO)

3. Inference-Time Techniques

4. Autonomous Theorem Proving with Formal Language

- 4.1 Introduction to Formal Language
- 4.2 The Potential of Formal Language Prover
- 4.3 Development of Open-Source Provers

5. Conclusion and Open Challenges

AlphaGeometry: An Olympiad-level AI system for geometry

AlphaGeometry is a neuro-symbolic system made up of a neural language model and a symbolic deduction engine.

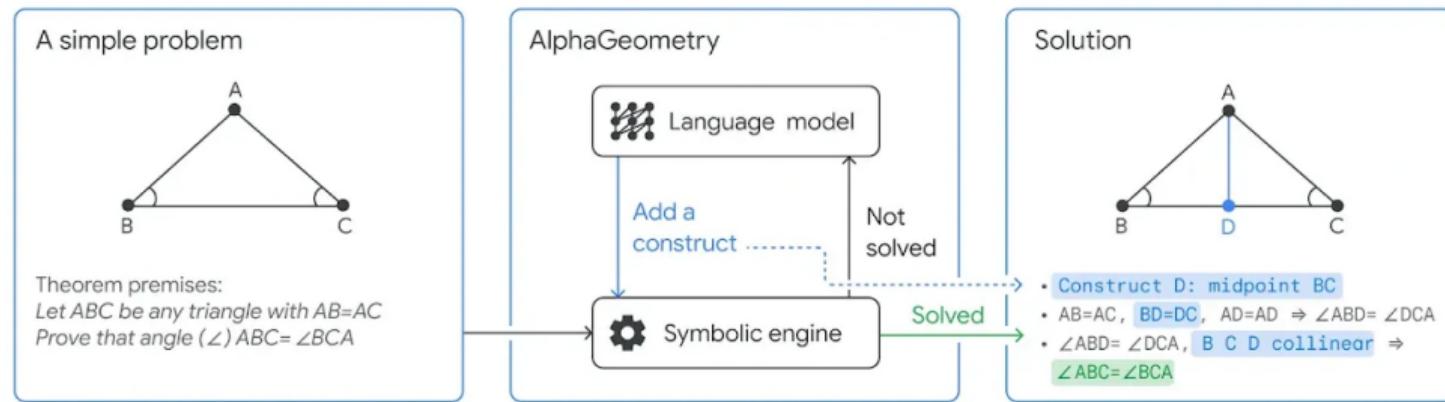


Figure: Given the problem diagram and its theorem premises (left), AlphaGeometry (middle) first uses its symbolic engine to deduce new statements about the diagram until the solution is found or new statements are exhausted. If no solution is found, AlphaGeometry's language model adds one potentially useful construct (blue), opening new paths of deduction for the symbolic engine. This loop continues until a solution is found (right). In this example, just one construct is required.

AlphaProof + AlphaGeometry 2: IMO 2024 silver madalist

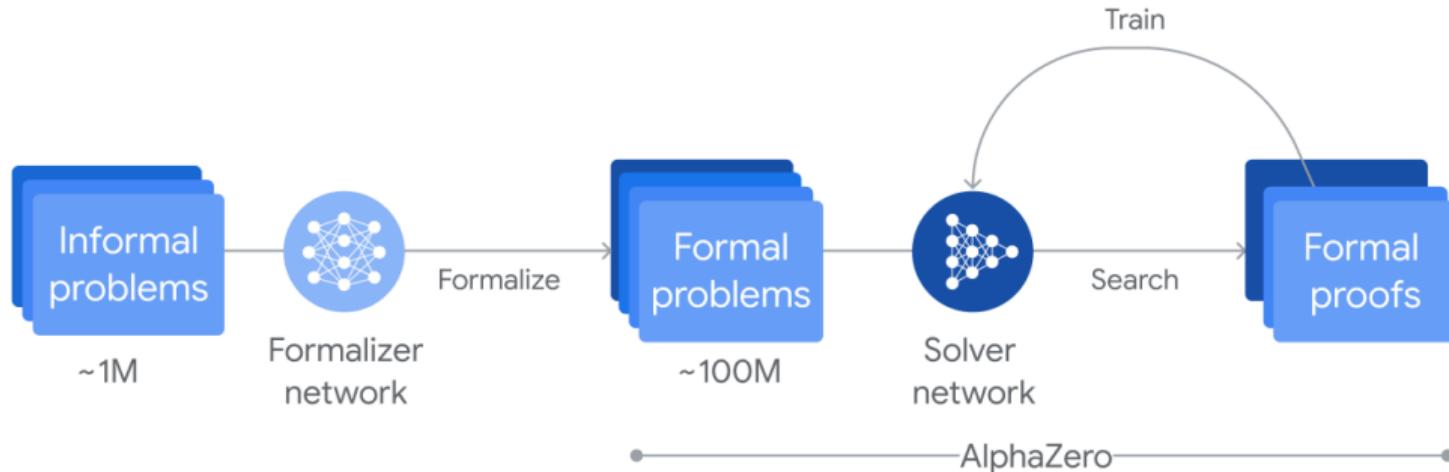


Figure: Process infographic of AlphaProof's reinforcement learning training loop: Around one million informal math problems are translated into a formal math language by a formalizer network. Then a solver network searches for proofs or disproofs of the problems, progressively training itself via the AlphaZero algorithm to solve more challenging problems.

AlphaProof is based on LEAN and Mathlib.

In 2025, ACM SIGPLAN Programming Languages Software Award was awarded to Gabriel Ebner, Soonho Kong, Leo de Moura and Sebastian Ullrich for Lean, cited for its “significant impact on mathematics, hardware and software verification, and AI”.



Figure: A dinner with Dr. Soonho Kong (left 1). Photographed by Dr. Zhiwen Qin.

LEAN Examples (1/3): Basic Proofs

Algebraic Identity

```
1 import Mathlib
2
3 theorem a_plus_b_squared (a b : ℚ)
4   (h1 : a - b = 4) (h2 : a *
5     b = 1) : (a + b) ^ 2 = 20 := 
6   by
7   calc
8     (a + b) ^ 2 = (a - b) ^ 2 + 4
9     * (a * b) := by ring
10    _ = 4 ^ 2 + 4 * 1 := by rw
11    [h1, h2]
12    _ = 20 := by norm_num
```

Inductive Proof

```
1 import Mathlib
2
3 theorem pow_two_ge_n_plus_one (n
4   : ℕ) : 2 ^ n ≥ n + 1 := by
5   induction n with
6   | zero => -- Base case: n = 0
7     norm_num -- Proves 1 ≥ 1
8   | succ k IH => -- Inductive step
9     calc
10      2 ^ (k + 1) = 2 * 2 ^ k :=
11      by ring
12      _ ≥ 2 * (k + 1) := by gcongr
13      _ = (k + 1 + 1) + k := by
14      ring
15      _ ≥ k + 1 + 1 := by linarith
```

LEAN Examples (2/3): A Curious Case in \mathbb{R}

```
1 import Mathlib
2
3 --- Disproof of the statement  $\forall x : \mathbb{R}, (x^{(1/3 : \mathbb{R})})^{(3 : \mathbb{R})} = x$ 
4 theorem cube_root_cubed_is_false :  $\neg (\forall x : \mathbb{R}, (x^{(1/3 : \mathbb{R})})^{(3 : \mathbb{R})} = x) := \text{by}$ 
5 push_neg
6 use  $\neg 1$ 
7 --- Goal:  $((-1 : \mathbb{R})^{(1 / 3 : \mathbb{R})})^{(3 : \mathbb{R})} \neq -1$ 
8
9 --- Calculate  $(-1)^{(1/3)}$ 
10 have hcbrt :  $(-1 : \mathbb{R})^{(1 / 3 : \mathbb{R})} = 1 / 2 := \text{by}$ 
11 have h_neg_one_lt_zero :  $(-1 : \mathbb{R}) < 0 := \text{by norm_num}$ 
12 rw [Real.rpow_def_of_neg h_neg_one_lt_zero]
13 rw [Real.log_neg_eq_log 1, Real.log_one, zero_mul, Real.exp_zero, one_mul]
14 rw [mul_comm, ←div_eq_mul_inv, Real.cos_pi_div_three]
15
16 rw [hcbrt]
17 norm_num --- Proves  $(1/2)^3 \neq -1$ 
```

The statement is false for real numbers. Lean finds a counterexample with $x = -1$. The definition of real number exponentiation for a negative base, ‘Real.rpow_def_of_neg’, involves the cosine of π times the exponent. This leads to $(-1)^{1/3} = \cos(\pi/3) = 1/2$. Since $(1/2)^3 \neq -1$, the general statement is disproven. This highlights the subtle pitfalls of informal reasoning.

LEAN Examples (3/3): Resolution in \mathbb{C}

```
1 import Mathlib
2
3 theorem complex_cube_root_cubed_eq_self (x : ℂ) : (x ^ ((3 : ℙ)⁻¹ : ℂ)) ^ 
4   (3 : ℙ) = x := by
5   have h3 : (3 : ℙ) ≠ 0 := by decide
6   exact Complex.cpow_nat_inv_pow x h3
```

In the complex plane, the identity holds true. The theorem ‘Complex.cpow_nat_inv_pow’ states that for any complex number x and any non-zero natural number n , taking the principal n -th root and then raising it to the power of n returns the original number x . The formal distinction between real and complex exponentiation is crucial, and Lean’s type system enforces this rigor, preventing such errors.

Contents

1. Introduction of LLM Reasoning

2. Training Techniques

- 2.1 An Overview
- 2.2 Supervised Fine-Tuning (SFT) based Methods
- 2.3 Group Relative Policy Optimization (GRPO)
- 2.4 Direct Preference Optimization (DPO)

3. Inference-Time Techniques

4. Autonomous Theorem Proving with Formal Language

- 4.1 Introduction to Formal Language
- 4.2 The Potential of Formal Language Prover
- 4.3 Development of Open-Source Provers

5. Conclusion and Open Challenges

Formal Language v.s. Informal Language

Given that $1 \leq x, y, z \leq 6$, how many cases are there in which the product of natural numbers x, y, z is divisible by 10?

Claude-3.5-Haiku (Poe) ...

Let's solve this systematically:

1. For a product to be divisible by 10, it must have factors 2 and 5

2. In the range 1 to 6, the numbers are:

- 1 = 1
- 2 = 2
- 3 = 3
- 4 = 2^2
- 5 = 5
- 6 = 2×3

Intermediate Steps ?

3. To be divisible by 10, the product must contain:

- At least one 2
- At least one 5

4. Possible combinations:

- One 5 and any other number(s)
- One 2 and one 5
- Multiple ways to choose these numbers from {1,2,3,4,5,6}

5. Counting method:

- Total ways to choose $x, y, z = 6^3 = 216$
- Ways divisible by 10 = Cases with 2 and 5

6. Calculating:

- If 5 is one of x, y, z : $(3 \text{ choose } 1) \times (5 \text{ choose } 1) \times (5 \text{ choose } 1) = 75$
- If 2 and 5 are different numbers: $(3 \text{ choose } 2) \times (2 \text{ choose } 1) \times (5 \text{ choose } 1) = 60$

7. Total cases: $75 + 60 = 135$

The answer is 135.

Final answer ✓

```
1 import Mathlib
2
3 def count_divisible_by_10 : ℕ :=
4   (finset.range 6).card * (finset.range 6).card * (finset.range 6).card
5
6 def is_divisible_by_10 (x y z : ℕ) : Prop :=
7   x * y * z % 10 = 0
8
9 theorem count_cases_divisible_by_10 :
10  let valid_numbers := (finset.range 1 7) in
11  valid_numbers.filter (λ x, valid_numbers.filter (λ y, valid_numbers.filter (λ z, is_divisible_by_10 x y z)).card).sum = 135 :=
12 begin
13  -- Count the valid cases
14  have h : (finset.range 1 7).prod (λ x, (finset.range 1 7).prod (λ y, (finset.range 1 7).filter (λ z,
15  is_divisible_by_10 x y z).card)) = 135,
16  { sorry }, -- Proof goes here
17  exact h,
```



Machine Checkable!
Whole proof ✓

Figure: Left: Informal Language; Right: Formal Language.

The Potential of Formal Language Prover



Terence Tao

@tao@mathstodon.xyz

I have just launched a "Lean companion" to my real analysis undergraduate textbook "Analysis I" at github.com/teorth/analysis. This gives a Lean translation (or paraphrasing) of the various definitions, theorems, and exercises in the textbook into Lean, thus allowing for an alternate way for students to work through the text. It is also designed to gradually transition into the standard Lean library Mathlib, thus also potentially serving as an introduction to that library as well.

Further discussion at terrytao.wordpress.com/2025/05...

Ya-Qin Zhang: AI can! Shing-tung Yau: AI can't!

张亚勤和丘成桐打了个赌：
5年内AI能证明
人类没有证明的猜想或者公式

搜狐首页

Contents

1. Introduction of LLM Reasoning

2. Training Techniques

- 2.1 An Overview
- 2.2 Supervised Fine-Tuning (SFT) based Methods
- 2.3 Group Relative Policy Optimization (GRPO)
- 2.4 Direct Preference Optimization (DPO)

3. Inference-Time Techniques

4. Autonomous Theorem Proving with Formal Language

- 4.1 Introduction to Formal Language
- 4.2 The Potential of Formal Language Prover
- 4.3 Development of Open-Source Provers

5. Conclusion and Open Challenges

The Development of Open-Source Provers in the Past Year

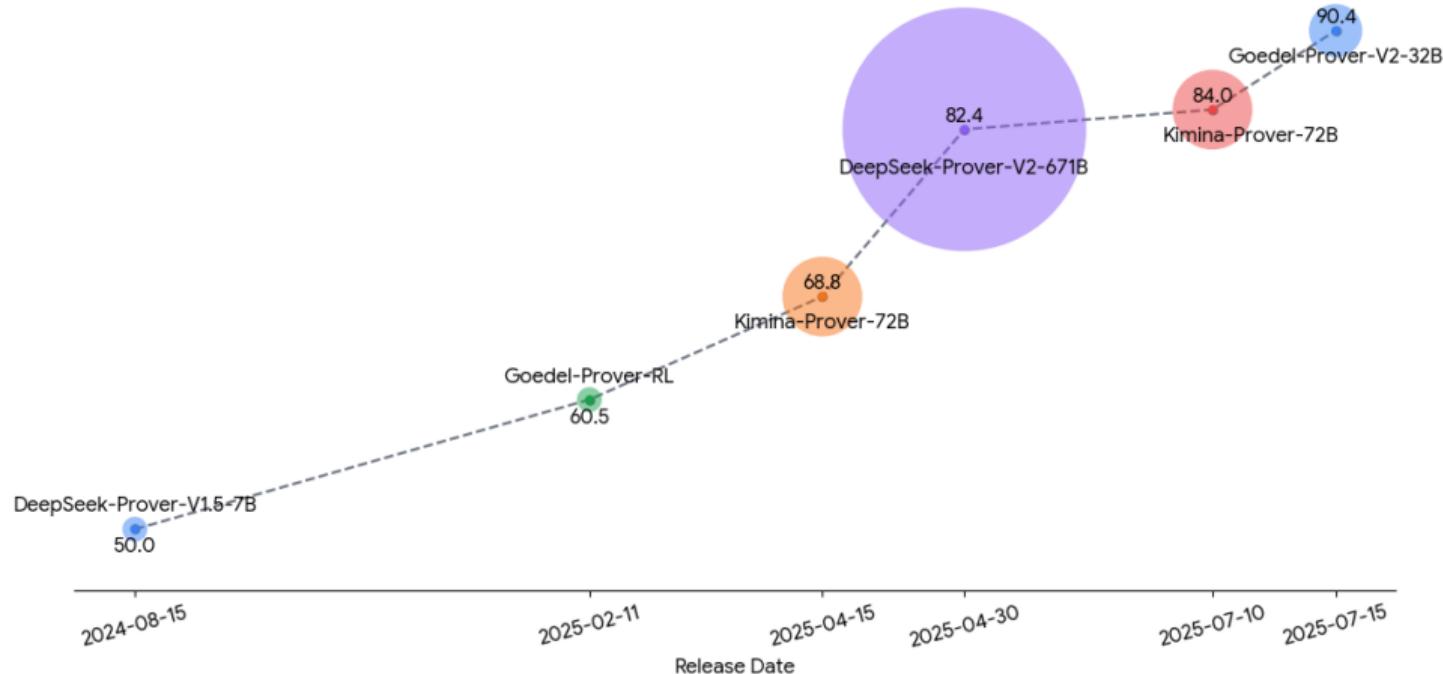


Figure: Performance of SOTA open-source provers over time (pass@32).

Achievements of Goedel-Prover-V2 - 1/2

NEW SOTA ON **MiniF2F**
+8.0% Pass@32

Our flagship 32B model significantly outperforms DeepSeek-Prover-V2-671B.

Small BUT Mighty
8B \gtrsim 671B

Our small 8B model matches the prior SOTA 671B model at Pass@32.

#1 ON PutnamBench
#1

Securing the top rank on the PutnamBench leaderboard.

Achievements of Goedel-Prover-V2 - 2/2

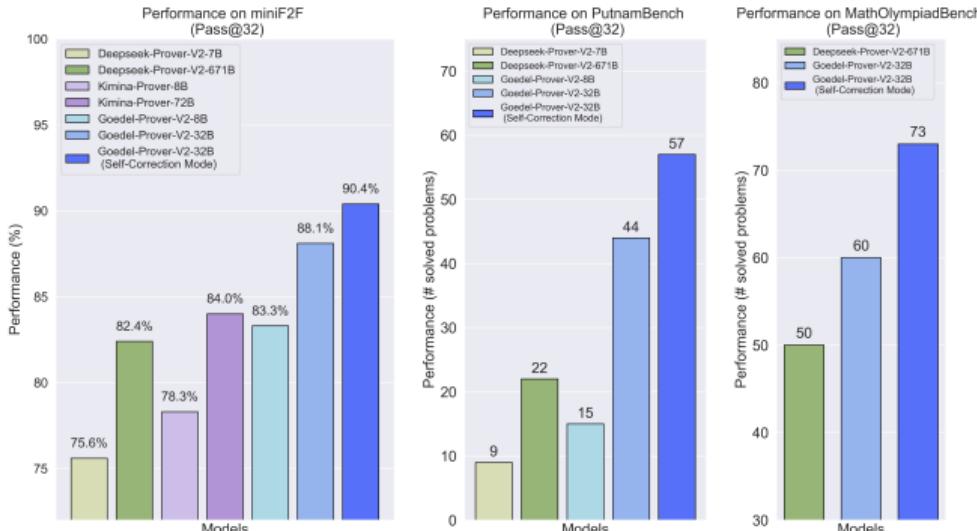


Figure: Pass@32 performance on MiniF2F, PutnamBench, and MathOlympiadBench.

Model	Solved	Pass
Goedel-Prover-V2-32B (self-correction mode)	64	Pass@64
Goedel-Prover-V2-32B (self-correction mode)	57	Pass@32
Goedel-Prover-V2-32B	44	Pass@32
DeepSeek-Prover-V2-671B	47	Pass@1024
DeepSeek-Prover-V2-671B	22	Pass@32
DSP+	23	Pass@128
Kimina-Prover-7B-Distill	10	Pass@192
Self-play Theorem Prover	8	Pass@3200
Goedel-Prover-V1	7	Pass@512

Table: PutnamBench leaderboard.
Goedel-Prover-V2-32B secures the top rank with significantly less compute.

Methods of Goedel-Prover-V2 - 1/4

Expert Iteration & RL: Formalizing problems, generating and verifying proofs, with supervised fine-tuning and reinforcement learning.

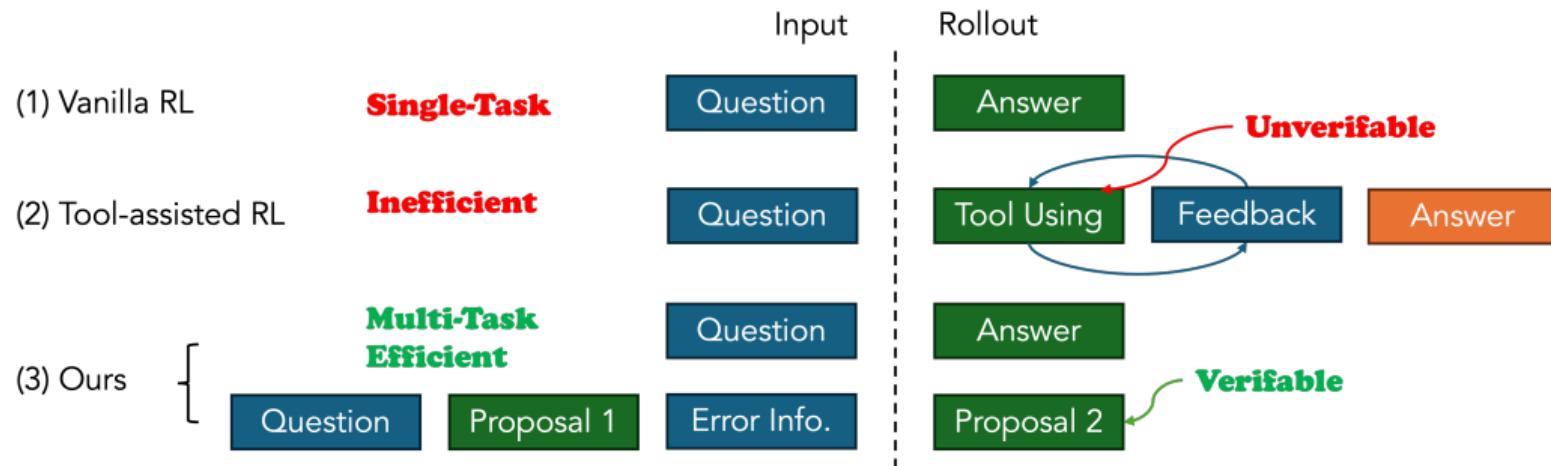


Figure: Our RL Pipeline, compared with vanilla and tool-assisted RL.

Scaffolded Data Synthesis: Implementing intermediate-difficulty problems that provide denser training signals.

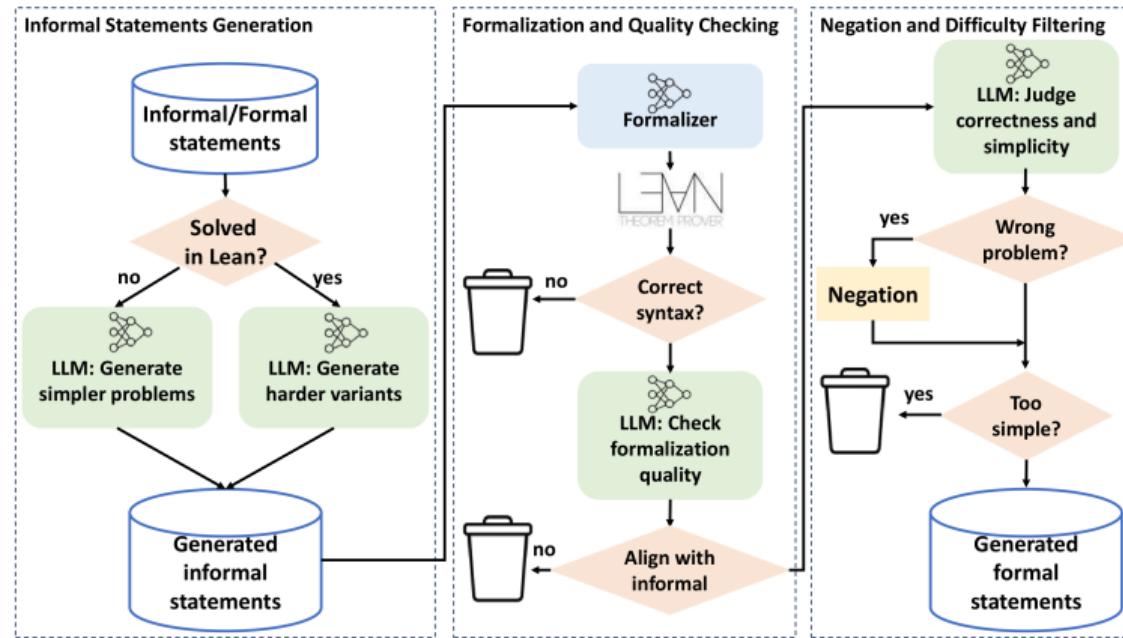


Figure: Scaffolded data synthesis pipeline.

Methods of Goedel-Prover-V2 - 3/4

Verifier-Guided Self-Correction: Our model uses Lean compilation feedback to iteratively correct its own proofs.

The screenshot shows the Lean 4 IDE interface. On the left, a code editor displays a Lean proof script. The script includes various tactic commands like `field_simp`, `norm_cast`, and `linarith` to prove a theorem about trigonometric identities. On the right, a terminal window shows the execution of the proof. A red box highlights the terminal output, which includes the goal `m : 88 * π / 180 = 88 * 0` and the feedback message `linarith failed to find a contradiction`. Below the terminal, a red banner reads "Lean Feedback".

```
1 field_simp [θ, mulComm, mulAssoc, Real.pi_ne_zero] at h
2 norm_cast at h
3 omega
4
5 have h6: cos (182 * θ) = -cos (2 * θ) := by
6 have hh: 182 * θ = 2 * θ + π:= by
7 field_simp [θ]
8 linarith
9 rw [hh]
10 simp
11
12 have h7: tan (88 * θ) = cos (2 * θ) / sin (2 * θ) := by
13 have hh: 88 * θ = π/2 - 2 * θ:= by
14 field_simp [θ]
15 linarith
16 rw [hh, Real.tan_pi_div_two_sub, Real.tan_eq_sin_div_cos]
17 field_simp
18
19 rw [h7]
20 linarith
21
22 field_simp [θ]
23
24
25
26 theorem sine_sum_equals_tan (m : ℚ) (h₀: 0 < m) (h₁ : (In k ∈ Finset.Icc (1 : N) 46, Real.sin (4 * k * Real.pi / 180)) = Real.tan (m * Real.pi / 180)) (h₂ : m < 90) : m.den + m.num = 89 := by
27 rw [sine_sum] at h₁
28
29 have hm: m=(88:Q) := by
30 have h₁: -(π/2) < 88*Real.pi/180 := by linarith [Real.pi_pos]
31 have h₂: 88*Real.pi/180 < (π/2) := by linarith [Real.pi_pos]
32 have h₃: -(π/2) < m*Real.pi/180 := by
33 have h₃₁: -(π/2) < 0 := by linarith
34 have h₃₂: 0 < m*Real.pi/180 := by positivity
35 linarith
```

Latest: Mathlib Examples Load ⌂

```
m : 88 * π / 180 = 88 * 0
↑
|- sin (4 * 0 * 0) = 0
```

Messages (1)

MathlibDemo.lean:70.2

linarith failed to find a contradiction

case h1.h

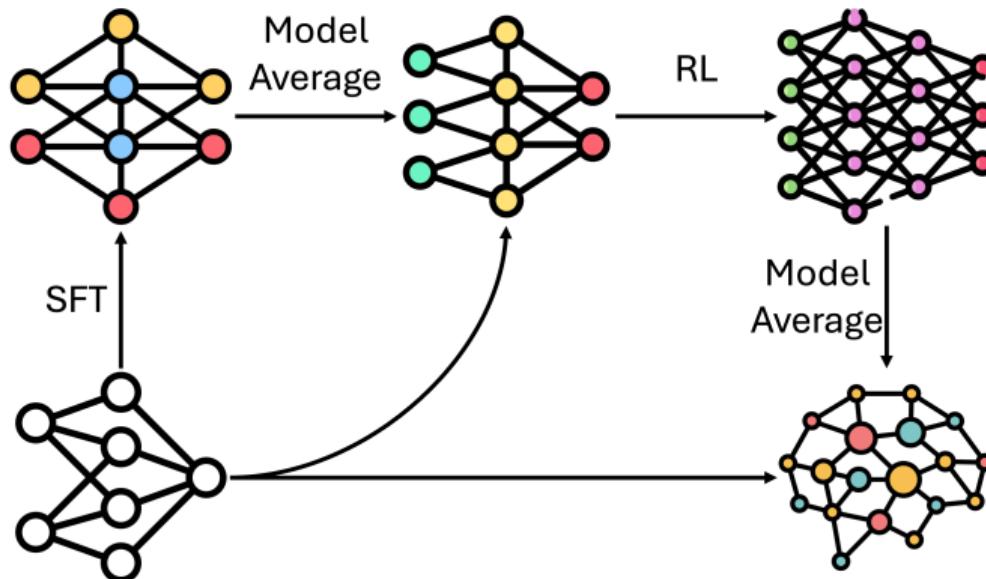
θ : ℝ := π / 180
f : ℕ → ℝ := fun k => sin (4 * 1k * θ)
h0 : ∀ (k : ℤ), 4 * k * π / 180 = 4 * k * θ
h1 : 88 * π / 180 = 88 * 0
h2 : ∀ (k : ℤ), 2 * sin (4 * k * θ) * sin (2 * θ) = sin (4 * k * θ) * (2 * sin (2 * θ))
S : ℝ := I_n k ∈ Finset.range 46, sin (4 * 1k * θ)
h3 : S * (2 * sin (2 * θ)) = cos (2 * θ) - cos (182 * θ)
hn : S = I_n k ∈ Finset.range 46, sin (4 * 1k * θ)
h5 : sin (2 * θ) ≠ 0
h6 : cos (182 * θ) = -cos (2 * θ)
h7 : tan (88 * θ) = cos (2 * θ) / sin (2 * θ)
ot : S < cos (2 * θ) / sin (2 * θ)
↑ False

failed Lean Feedback

All Messages (1)

Restart File

Model Averaging: Averaging trained models with the base model to restore diversity and boost Pass@K performance.



Goedel-Prover-V2 Team



Yong Lin
PLI Princeton



Shange Tang
PLI & NVIDIA



Bohan Lyu
Tsinghua



Ziran Yang
PLI Princeton



Jui-Hui Chung
PLI Princeton



Haoyu Zhao
PLI Princeton



Lai Jiang
SJTU



Yihan Geng
Peking University



Jiawei Ge
PLI Princeton



Jingruo Sun
Stanford



Jiayun Wu
Tsinghua



Jiri Gesi
Amazon



David Acuna
NVIDIA



Kaiyu Yang
Meta FAIR



Hongzhou Lin
Amazon



Yejin Choi
NVIDIA & Stanford



Danqi Chen
PLI Princeton



Sanjeev Arora
PLI Princeton



Chi Jin
PLI Princeton

Contents

1. Introduction of LLM Reasoning

2. Training Techniques

- 2.1 An Overview
- 2.2 Supervised Fine-Tuning (SFT) based Methods
- 2.3 Group Relative Policy Optimization (GRPO)
- 2.4 Direct Preference Optimization (DPO)

3. Inference-Time Techniques

4. Autonomous Theorem Proving with Formal Language

- 4.1 Introduction to Formal Language
- 4.2 The Potential of Formal Language Prover
- 4.3 Development of Open-Source Provers

5. Conclusion and Open Challenges

Conclusion

- **Rapid Technological Advancement:** Driven by advanced training techniques like Reinforcement Learning (e.g., PPO, GRPO) and Direct Preference Optimization (DPO), the reasoning capabilities of Large Language Models are quickly moving beyond simple heuristics.
- **Neuro-Symbolic Synergy:** The combination of neural language models and symbolic systems is unlocking new potential, especially in domains requiring rigorous logic, such as formal mathematics.
- **Bridging the Informal-to-Formal Gap:** With systems like AlphaGeometry and Goedel-Prover leading the charge, the research focus is shifting towards translating informal, natural language reasoning into verifiable, formal proofs.
- **The Power of Open Source:** The open-source community plays a crucial role in democratizing and accelerating progress in automated reasoning.

Open Challenges

- **Scalability Efficiency:** Reducing the immense computational cost of training and inference for state-of-the-art reasoning models is key to their widespread adoption.
- **The Informal-to-Formal Gap:** Accurately and automatically translating informal problem descriptions into a formal language remains a major technical bottleneck.
- **Broader Generalization:** Extending these powerful reasoning abilities beyond mathematics to other complex domains like law, medicine, and scientific discovery is a critical next step.
- **True Understanding:** Can models evolve from pattern matching and proof-searching to a deeper, more intuitive form of reasoning and problem-solving?
- **Reliability Trustworthiness:** In scenarios where formal verification is not possible, how can we fully trust the outputs of these models?

Thanks!

Bohan Lyu
lyubh.cn