CMPT 413/713: Natural Language Processing

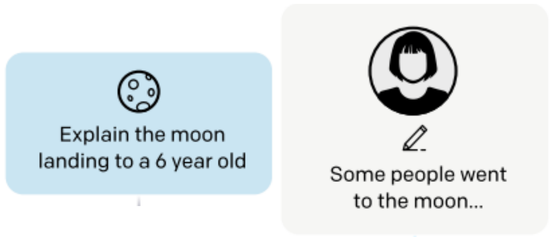
Post-training: Reasoning

Spring 2026

2026-03-04

Slides adapted from Anoop Sarkar

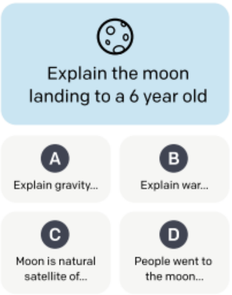
Training stages



Piles of unlabeled text!

Text with "instructions" and "responses"

Human preference data
Data about what people prefer



Self-supervised training LM objective

$p(w_t \mid w_{<t})$

Pre-training
LM training on large, large amount of data

Pre-training can be broken into stages (mid-training)

Supervised training LM objective

$p(w_t \mid w_{<t}; \text{prompt})$

Instruction-tuning
Supervised fine-tuning for instructions

Reinforcement learning

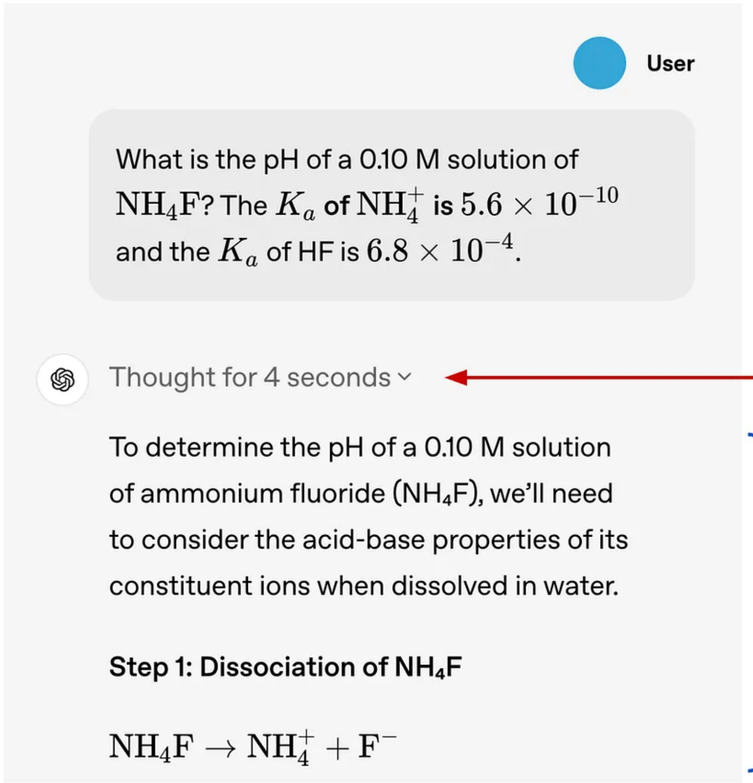
Preference optimization
Align to human preferences

Post-training
(Can have more iterations)

Training for reasoning

Reasoning models

- Think before responding with the answer
- Reasoning problems requires decomposing the problem into multiple steps (think step-by-step)
- Typically math, coding problems



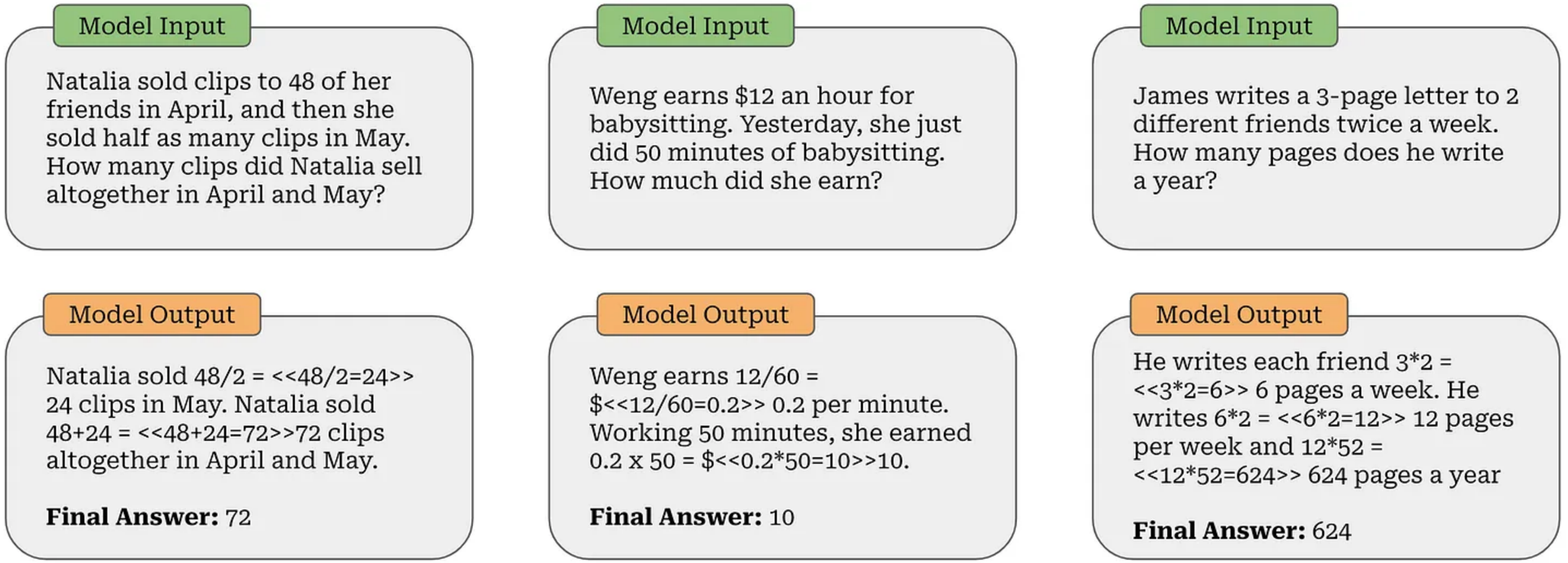
Full long CoT is hidden from the User

A summary of the long CoT is provided instead

GSM8K

Reasoning problems

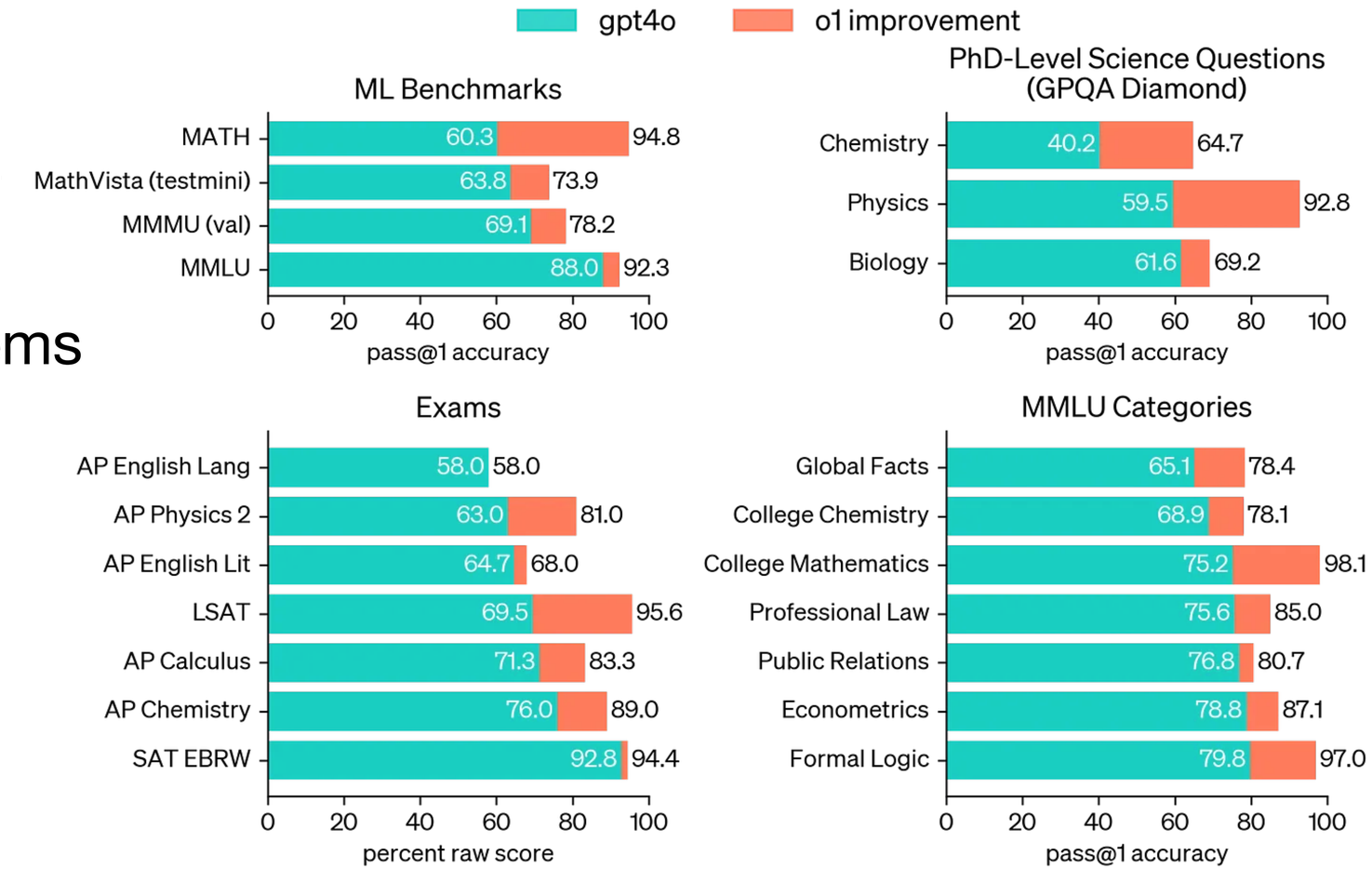
- Problem that requires step-by-step



GSM8K

Reasoning models

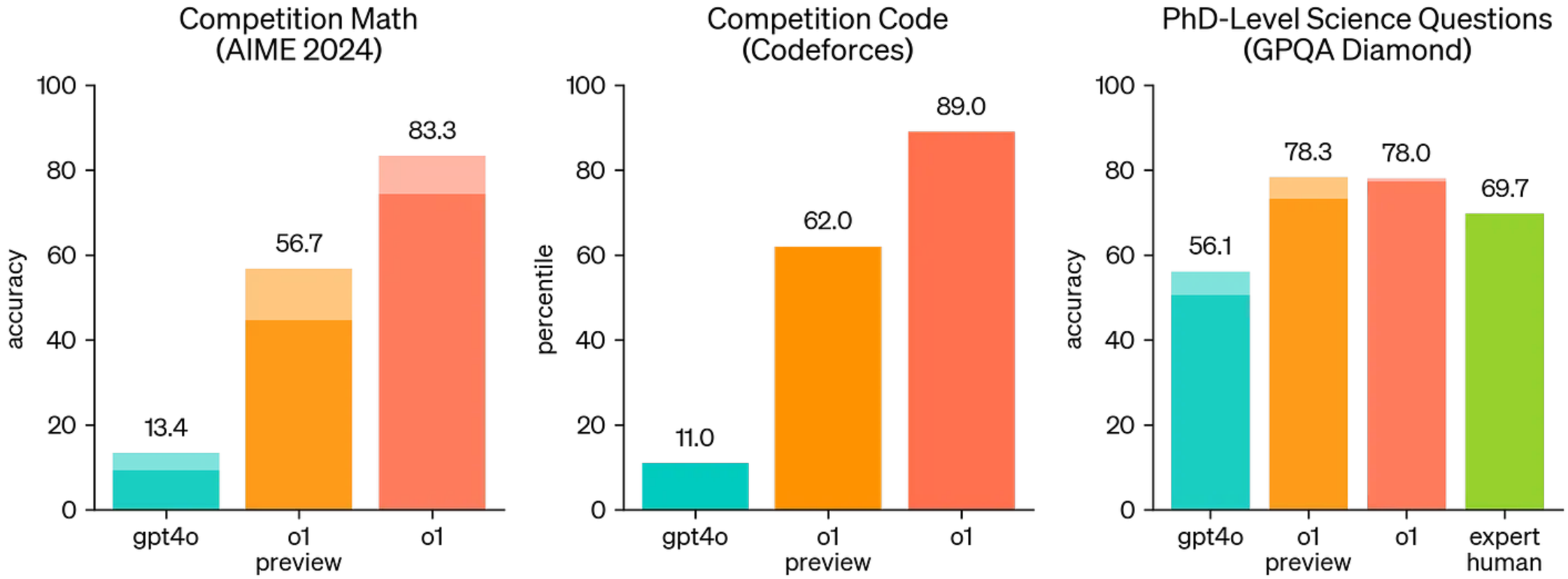
- Think through each part of a complex problem
- Decompose complex problems into smaller, solvable parts
- Critique its own (partial) solutions and find errors
- Explore many alternative solutions



https://openai.com/index/learning-to-reason-with-llms/

https://cameronrwolfe.substack.com/p/demystifying-reasoning-models

Reasoning models



https://openai.com/index/learning-to-reason-with-llms/

Training LLMs to think step-by-step

- Supervised training with reasoning data and reasoning traces
- Reinforcement learning with verifiable rewards (RLVR)

<|im_start|>user
Give me a short introduction to large language model.<|im_end|>

Prompt / Instruction

<|im_start|>assistant

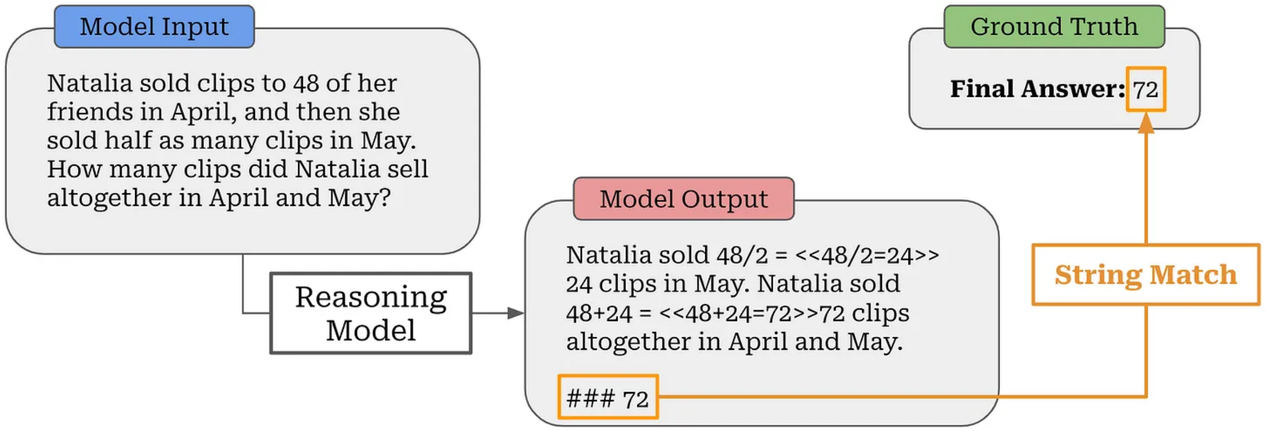
Reasoning Model Output

<think>
Okay, the user wants a short introduction to large language models. Let me start by defining what they are. They're AI systems trained on massive text data, right? I should mention their ability to understand and generate human-like text. Maybe include examples like GPT or BERT. Also, highlight their applications in tasks like answering questions, writing, and coding. Keep it concise but cover the key points. Make sure to note that they're based on deep learning and neural networks. Avoid technical jargon to keep it accessible. Let me check if I'm missing anything important. Oh, maybe mention their scale in terms of parameters. Yeah, that's a common point. Alright, time to put it all together in a clear, brief paragraph.
</think>

Reasoning Trace

Large language models (LLMs) are advanced artificial intelligence systems trained on vast amounts of text data to understand and generate human-like language. They use deep learning techniques, such as transformer architectures, to process and analyze text at scale, enabling them to perform tasks like answering questions, writing stories, coding, and translating languages. LLMs are characterized by their massive parameter counts, which allow them to capture complex patterns in language, making them highly versatile for a wide range of applications in natural language processing and beyond.
<|im_end|>

Final Response



https://cameronrwolfe.substack.com/p/grpo-tricks

Training LLMs to think step-by-step

- Supervised training with reasoning data and reasoning traces
- Reinforcement learning with verifiable rewards (RLVR)

- Alternative to training LLMs for reasoning is to scale up test-time compute
  - Reasoning during inference

Olmo training phases

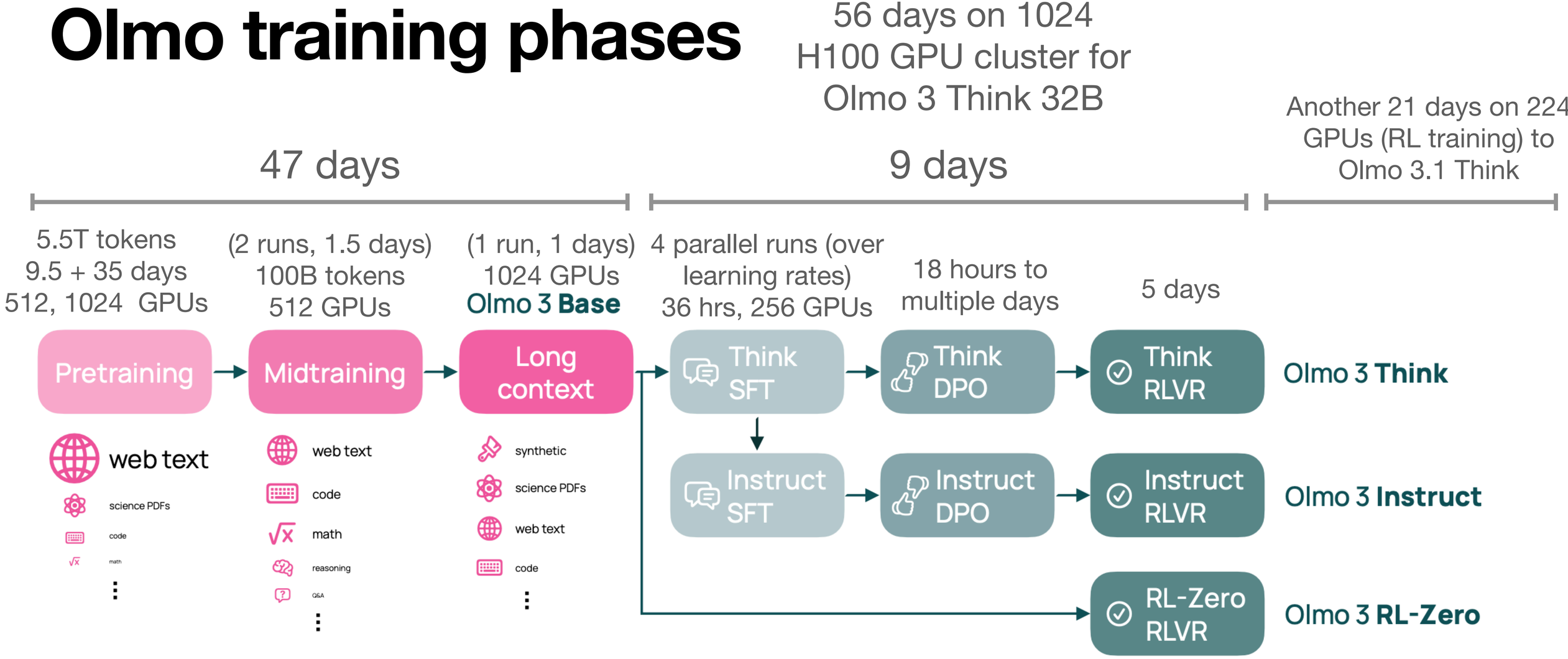


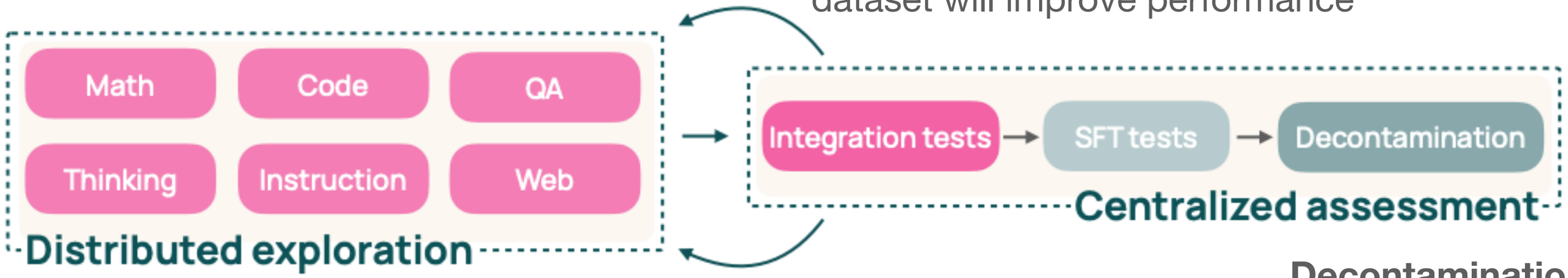
Figure 2 Depiction of model flow for Olmo 3. Development is divided into major **base model training (left)** and **post-training (right)** stages, each further divided into sub-stages with their own recipes (i.e., training data and method).

Figure from https://arxiv.org/pdf/2512.13961 [Olmo 3, 2025]

Mid-training

Training on cleaned, specialized data

Integration tests (on 100B tokens)
to verify whether including the
dataset will improve performance



Explore different mixes with small
amounts (5B to 10B tokens)
of sampled data.

Decontamination:
Run detection to try to avoid
benchmark contamination (i.e.
training on evaluation data)

Figure from https://arxiv.org/pdf/2512.13961 [Olmo 3, 2025]

Olmo Post-training

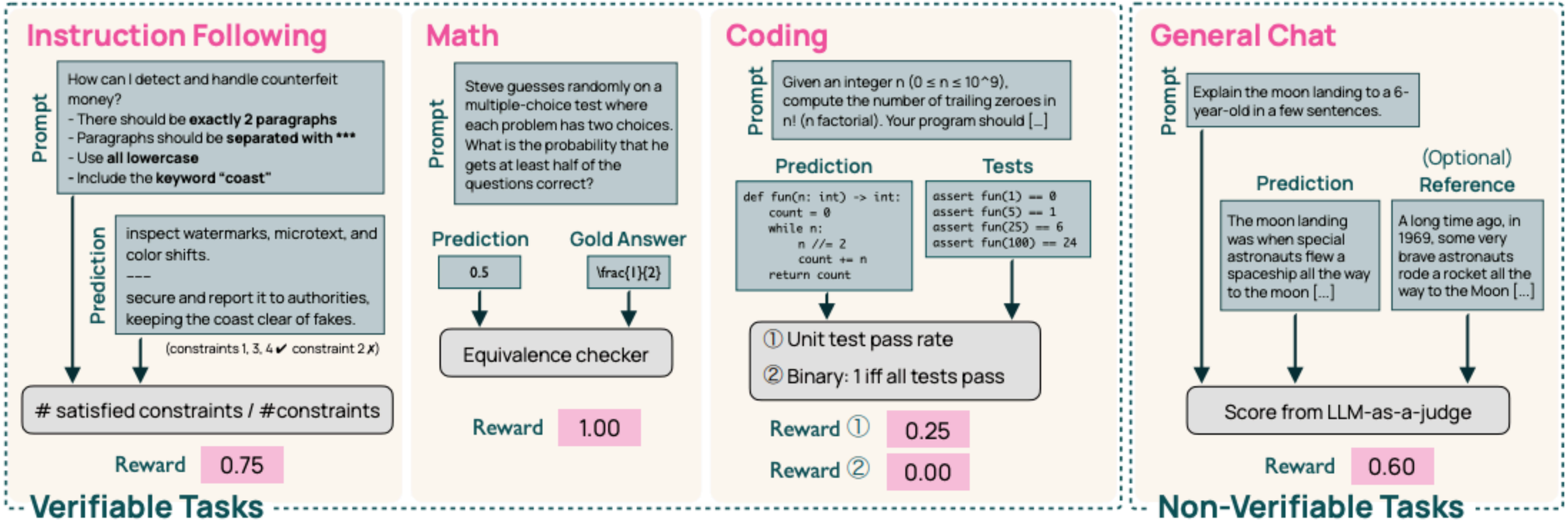
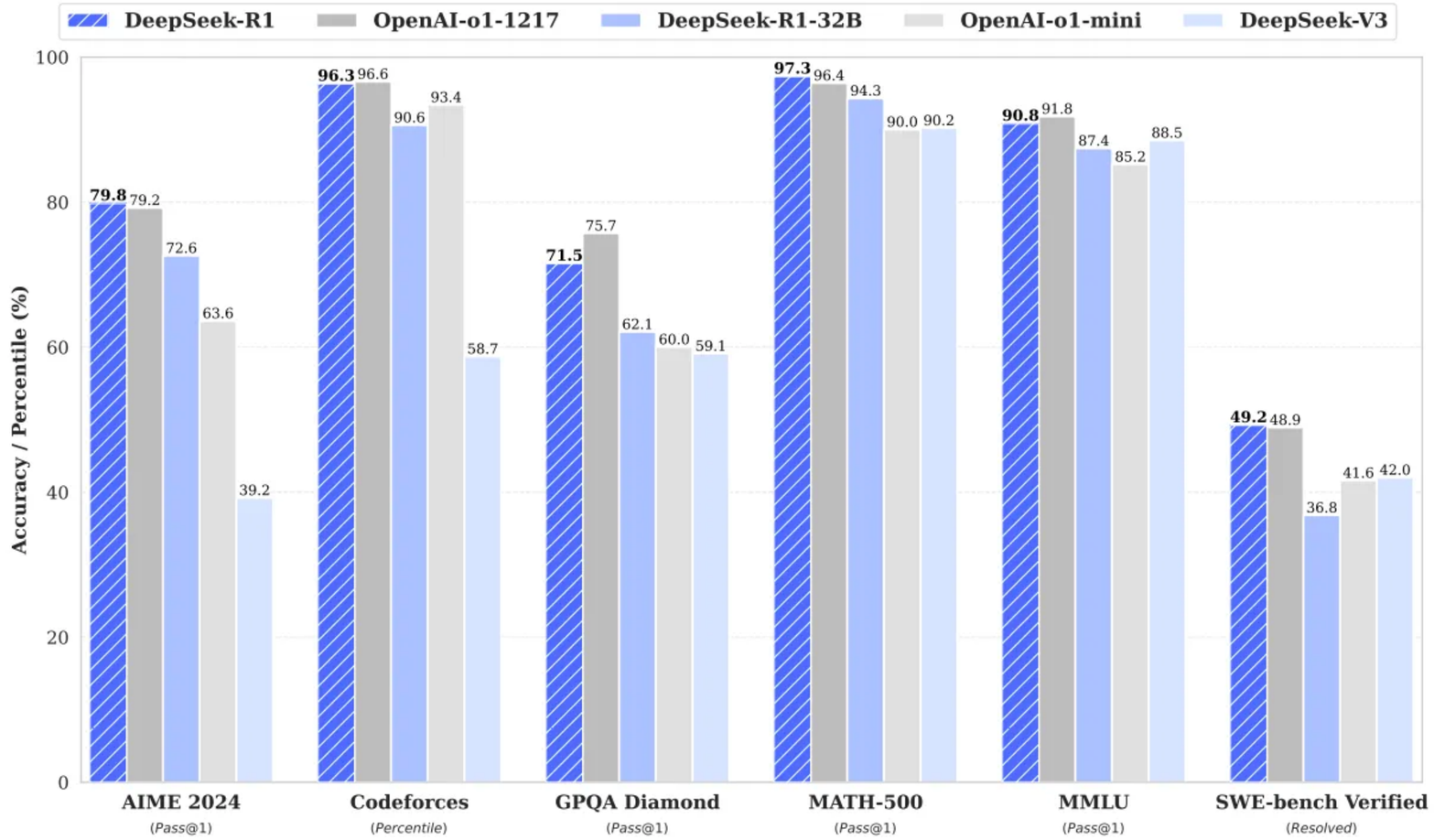


Figure 16 Verifiers and reward design for verifiable and non-verifiable tasks.

Figure from https://arxiv.org/pdf/2512.13961 [Olmo 3, 2025]

Deepseek R1

Competitive open-source model



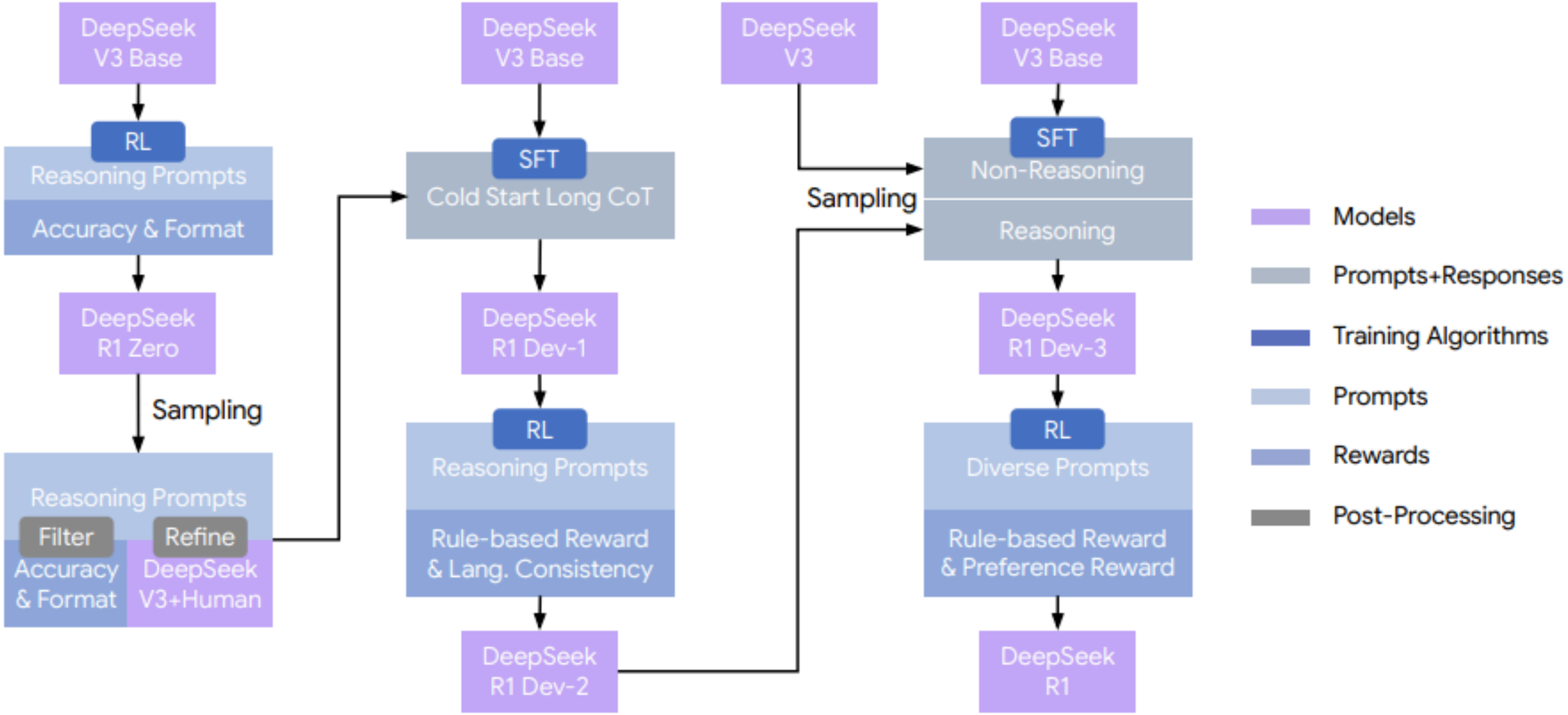
DeepSeek-R1 [DeepSeek 2025] - https://arxiv.org/pdf/2501.12948

Deepseek R1

- DeepSeek-V3 - MoE model trained with multi-token prediction
- DeepSeek-R1-Zero - Use RL to improve base model (DeepSeek-V3) for reasoning (e.g. generate “thought” process)
- DeepSeek-R1 - Improved version with cold-start, supervised fine-tuning on generated / filtered data.

DeepSeek-R1 [DeepSeek 2025] - https://arxiv.org/pdf/2501.12948

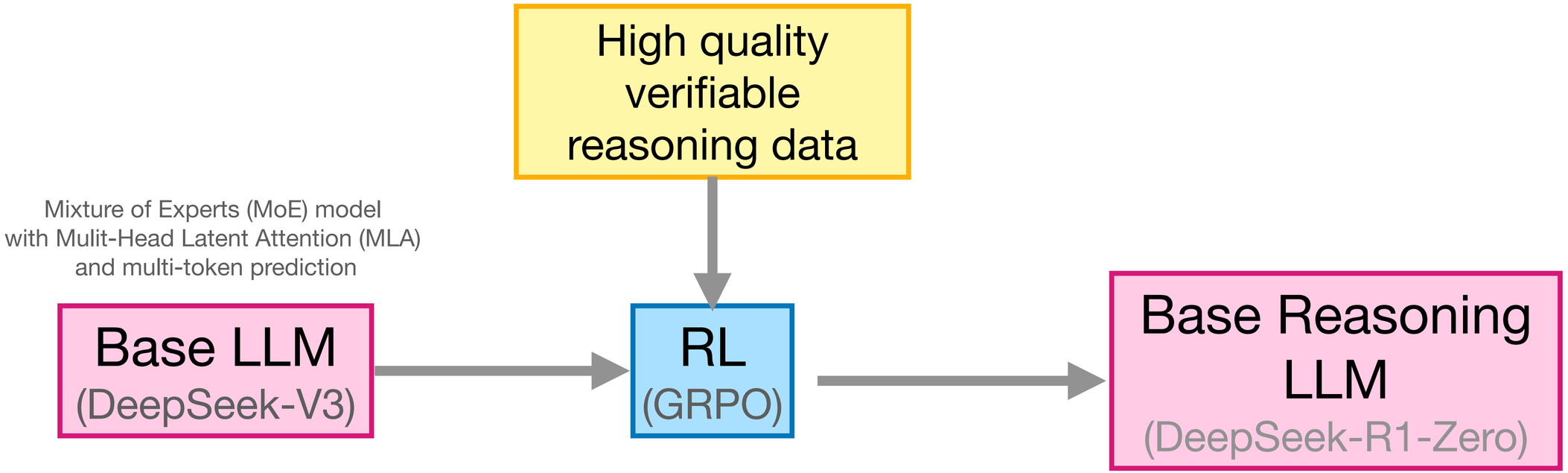
Deepseek R1



DeepSeek-R1 [DeepSeek 2025] - https://arxiv.org/pdf/2501.12948

Deepseek R1-Zero

Training recipe



DeepSeek-R1 [DeepSeek 2025] - https://arxiv.org/pdf/2501.12948

DeepSeek R1-Zero

Training recipe

- Train LLM to explicitly generate reasoning process

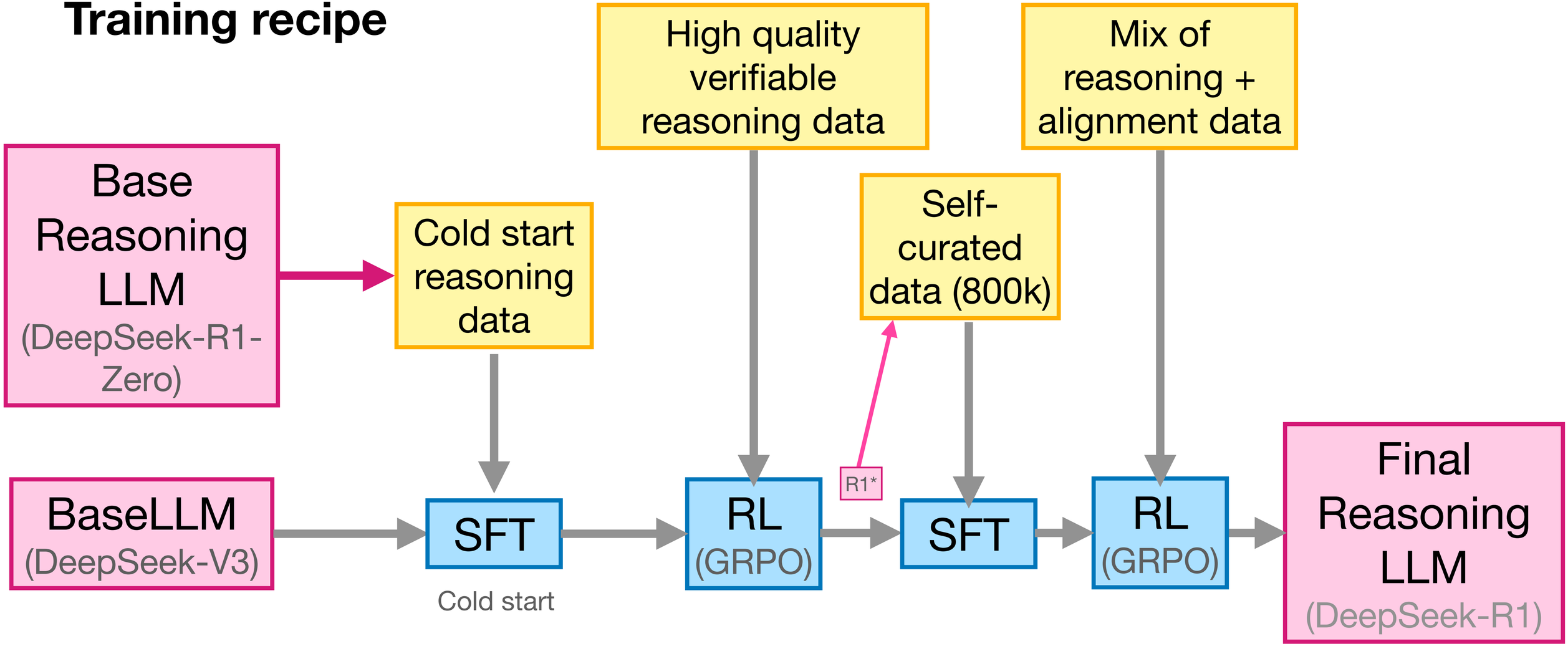
A conversation between User and Assistant. The user asks a question, and the Assistant solves it. The assistant first thinks about the reasoning process in the mind and then provides the user with the answer. The reasoning process and answer are enclosed within <think> </think> and <answer> </answer> tags, respectively, i.e., <think> reasoning process here </think> <answer> answer here </answer>. User: prompt. Assistant:

- **Accuracy reward** for whether response is correct and **format reward** to provide details of thinking process
- Additional **language consistency reward** is added when training DeepSeek-R1

DeepSeek-R1 [DeepSeek 2025] - https://arxiv.org/pdf/2501.12948

DeepSeek R1

Training recipe



DeepSeek-R1 [DeepSeek 2025] - https://arxiv.org/pdf/2501.12948

Reinforcement Learning Review

Some slides adapted from Graham Neubig and others

What is reinforcement learning?

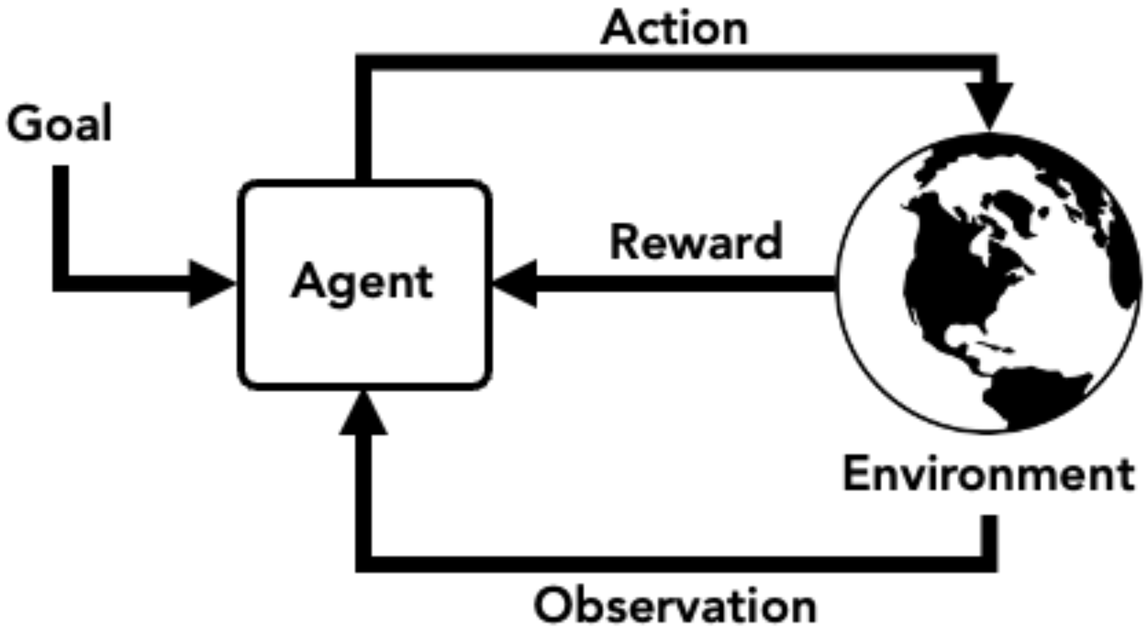
Learning where we have an

- Environment X
- Ability to make actions A
- Get delayed reward R

For text generation, we use RL when there is a **sequence-level evaluation metric** that is difficult to optimize without first generating the whole sentence.

Can also be used for longer dialog where reward is at the end (e.g. customer achieved their goal).

- Environment provides feedback
- No examples of optimal policy



Reinforcement learning

Determine policy to **maximize expected accumulated reward**.

Typically modelled as POMDP (sequence of states with partial observations)

- **Actions**: What token to output?
- **Policy**: What action(s) to take given sequence of observations and actions?
  - Policy models the probability of action given state
  - For text generation, what sequence of tokens y to generate given input x tokens: $\pi(a, s) = P(y \mid x)$
- **Reward**:
  RLHF: Provided by reward model trained on human preference
  RLVR: Provided by verifiable reward / answer

Reward in RL

- For an action at time step t, state s_t and action a_t, get a reward $r_t = R(s_t, a_t)$
- Goal is to maximize **total reward** over time $\sum_{t=0}^{\infty} r_t$
- But maybe better to get immediate reward rather than wait. So maximize **discounted reward** $\sum_{t=0}^{\infty} \gamma^t r_t$
- Everything is stochastic
  - **expected discounted reward** for a policy π $\mathbb{E}_{\pi}\left[\sum_{t}^{\infty} \gamma^t r_t\right]$

Reward in RL

- For a given trajectory $\tau = ((s_0, a_0), (s_1, a_1), \ldots)$, which is a sequence of states and actions in the world, we can have the following rewards.

Also known as **return**

- **Total reward:** $R_{\text{total}}(\tau) = \sum_{t=0}^{\infty} r_t$

- **Discounted reward:** $R_{\text{discounted}}(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$

Algorithms that optimize parameters of the policy directly are called **policy-based methods**

- Find a policy π to maximize **expected discounted reward**:

$$J(\pi) = \mathbb{E}_{\tau \sim \pi} = \int_{\tau} P(\tau \,|\, \pi) R(\tau) = \left[\sum_{t}^{\infty} \gamma^t r_t \right]$$

RL for text generation

- **State:** input text ($X = x_1, x_2, \ldots$) and text that is generated so far $(y_1, y_2, \ldots, y_{t-1})$
- **Action**: next token to generate y_t
- **Trajectory of actions** (generated text)**:** $Y = y1, \ldots, y_n$
- **Objective**: Maximize $J_{\mathrm{RL}}(X) = R(Y) \log P(Y|X)$

- **Loss**:

$$\ell_{\mathrm{RL}}(X) = -R(Y) \log P(Y|X)$$

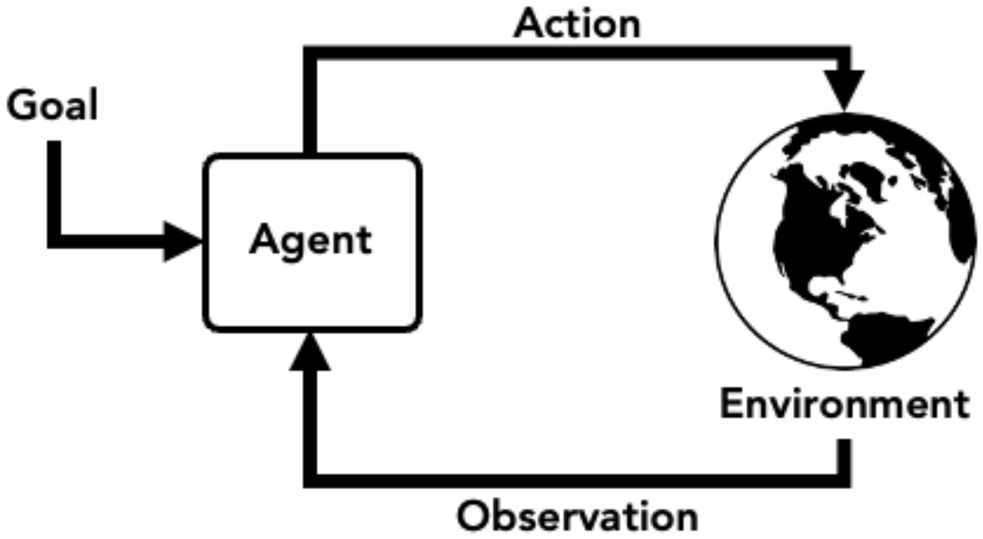
How is this related to supervised learning?

Supervised learning

In normal supervised learning, we do maximum likelihood estimation (MLE)

In the RL literature, this is also known as **imitation learning** where there is an "expert" demonstrating what actions to take at each time step

- Have expert demonstrations (possibly interactive)



$$\ell_{\mathrm{MLE}}(Y|X) = -\log P(Y|X)$$

Training set with
Input text X and output text Y

Examples of imitation learning algorithms:
- Behavior cloning - collect data (expert demonstrations once) and then train
- DAgger - iteratively collect data (as it encounters new states)

Self-training

- **Self-training**: If we start with a trained model, we can sample (or take argmax) to get data points (e.g. “demonstrations”) for training.

$$\hat{Y} \sim P(Y|X) \quad \text{or} \quad \hat{Y} = \arg\max_{Y} P(Y|X)$$

$$\ell_{\text{self}}(X) = -\log P(\hat{Y}|X)$$

Policy gradient / REINFORCE

- **Self-training**: If we start with a trained model, we can sample (or take argmax) to get data points (e.g. “demonstrations”) for training.

$$\hat{Y} \sim P(Y|X) \quad \text{or} \quad \hat{Y} = \arg\max_Y P(Y|X)$$

$$\ell_{\text{self}}(X) = -\log P(\hat{Y}|X)$$

- **RL**: Add term to scale loss by the reward

RL: Maximize reward

$$J_{\text{REINFORCE}}(X) = R(\hat{Y}) \log P(\hat{Y}|X)$$

$$\ell_{\text{REINFORCE}}(X) = -R(\hat{Y}) \log P(\hat{Y}|X)$$

Optimize policy using stochastic gradient descent

- **But some issues:**
  - step too small, training too slow,
  - too large, training is unstable (too much variability)

Regularization to existing model

- **KL-regularization**:

Improve reward

Keep model similar

$$J_{\text{KL-reg}} = \rho(\hat{Y}, X)R(\hat{Y}) - \beta \text{KL}[P(\cdot|X;\theta_{\text{old}}), P(\cdot|X;\theta)]$$

$$\rho(Y, X) = \frac{P(Y|X;\theta)}{P(Y|X;\theta_{\text{old}})}$$

Ratio function

  - **Add regularization term to keep model similar to initial model**
  - **Tricky to implement**

Regularization to existing model

- **KL-regularization**:

Improve reward

Keep model similar

$$J_{\text{KL-reg}} = \rho(\hat{Y}, X)R(\hat{Y}) - \beta\text{KL}[P(\cdot|X;\theta_{\text{old}}), P(\cdot|X;\theta)]$$

$$\rho(Y, X) = \frac{P(Y|X;\theta)}{P(Y|X;\theta_{\text{old}})}$$ Ratio function

- **Proximal policy optimization (PPO)**

Lower bound of unclipped objective

Prevent jumps

$$J_{\text{PPO}} = \min\left(\rho(\hat{Y}, X)R(\hat{Y}), \text{clip}(\rho(\hat{Y}, X), 1+\epsilon, 1-\epsilon)R(\hat{Y})\right)$$

Unclipped objective

Clipped objective

Avoid policy ratio between current / old from becoming too large or too small - keeps token probabilities similar

Going beyond reward

Hard to compute, so use neural networks to estimate these functions (critic training)

- State **Value** Function
  - How good is the state s_t?

$$V_\pi(s_t) = \mathbb{E}_\pi \left[\sum_{i=t}^{\infty} \gamma^{i-1} r_i \right]$$

- State-Action Value Function (**Q function)**
  - How good is a particular action a at state s_t?

$$Q_\pi(s_t, a) = \mathbb{E}_{s_{t+1}} \left[R(s_t, a, s_{t+1}) + \gamma \left[V_\pi(s_t) \right] \right]$$

- **Advantage**
  - How much better is taking action a than the average?

$$A_\pi(s_t, a) = Q_\pi(s_t, a) - V_\pi(s_t)$$

Use as return function in optimization:

$$J(X) = R(\hat{Y}) \log P(\hat{Y}|X) \rightarrow J(X) = A(\hat{Y}) \log P(\hat{Y}|X)$$

PPO with advantage

- **Proximal policy optimization (PPO)**

$$J_{\mathrm{PPO}} = \min\left(\rho(\hat{Y}, X)\hat{A}(\hat{Y}), \mathrm{clip}(\rho(\hat{Y}, X), 1+\epsilon, 1-\epsilon)\hat{A}(\hat{Y})\right)$$

$$\rho(Y, X) = \frac{P(Y|X;\theta)}{P(Y|X;\theta_{\mathrm{old}})}$$ Ratio function

On-policy, actor-critic algorithm

Actor: Policy network - determines what action to take

Critic: Value network - evaluates action by estimating the value function (so we can estimate the advantage function) $A(\hat{Y})$

RLHF with general policy gradient method

- Initialize the policy π_ϕ^{RL} to π^{SFT}, the policy output from SFT
- Loop for many steps
  - Initialize a new empty dataset $D_{\pi_\phi^{\text{RL}}}$
  - Loop for many steps
    - Sample a random prompt x from D_{RL}
    - Generate a response y from the policy π_ϕ^{RL}
    - Calculate the reward signal $r_\theta(x, y)$ from r_θ
    - Add the triple $(x, y, r_\theta(x, y))$ to $D_{\pi_\phi^{\text{RL}}}$
  - Use policy gradient method to increase objective

$$\text{objective}(\phi) = E_{(x,y)\sim D_{\pi_\phi^{\text{RL}}}}\left[r_\theta(x,y) - \beta \log\left(\frac{\pi_\phi^{\text{RL}}(y|x)}{\pi^{\text{SFT}}(y|x)}\right)\right]$$

RLHF with PPO

Algorithm 1 PPO

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0.
2: **for** $n = 0, 1, 2, \ldots$ **do**
3: Collect a set of trajectories $\mathcal{D}_n = \{\tau_i\}$ by executing policy $\pi(\theta_n)$ within the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates, $\hat{A}_t$ (using any advantage estimation method) based on the current value function V_{ϕ_n}.
6: Update the policy by maximizing the PPO-penalty/clip/ptx objective:

$$\theta_{n+1} = \arg \max_{\theta} \mathcal{L}_{\text{ppo-clip}}(\theta_n).$$

7: Update the value function by regression on mean-squared error:

$$\phi_{n+1} = \arg \min_{\phi} \mathcal{L}_{\text{critic}}(\phi_n).$$

8: **end for**

https://arxiv.org/pdf/2307.04964

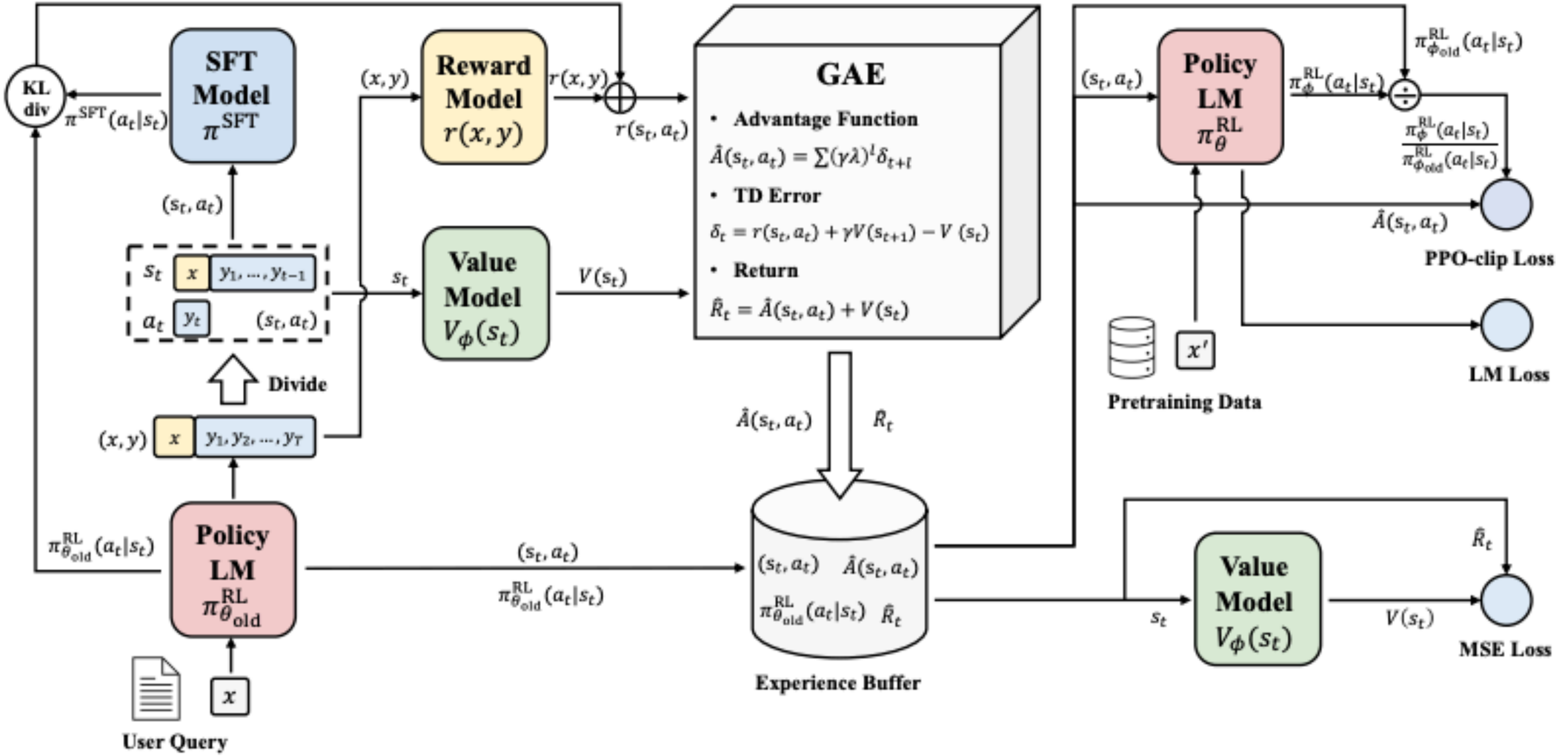


Figure 1: PPO workflow, depicting the sequential steps in the algorithm's execution. The process begins with sampling from the environment, followed by the application of GAE for improved advantage approximation. The diagram then illustrates the computation of various loss functions employed in PPO, signifying the iterative nature of the learning process and the policy updates derived from these losses.

https://arxiv.org/pdf/2307.04964

Group Relative Policy Optimization (GRPO)

- Replace advantage function estimated with neural reward model with advantage function that is computed from rule-based reward model
- Compute advantage as how much newer policy improves over older policy

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

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

$$\mathbb{D}_{KL}\left(\pi_\theta||\pi_{ref}\right) = \frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - \log\frac{\pi_{ref}(o_i|q)}{\pi_\theta(o_i|q)} - 1,$$

https://arxiv.org/pdf/2402.03300

Group Relative Policy Optimization (GRPO)

- Outcome supervision

$$\hat{A}_{i,t} = \tilde{r}_i = \frac{r_i - \text{mean}(\mathbf{r})}{\text{std}(\mathbf{r})} \qquad \mathbf{r} = \{r_1, \cdots, r_G\}$$

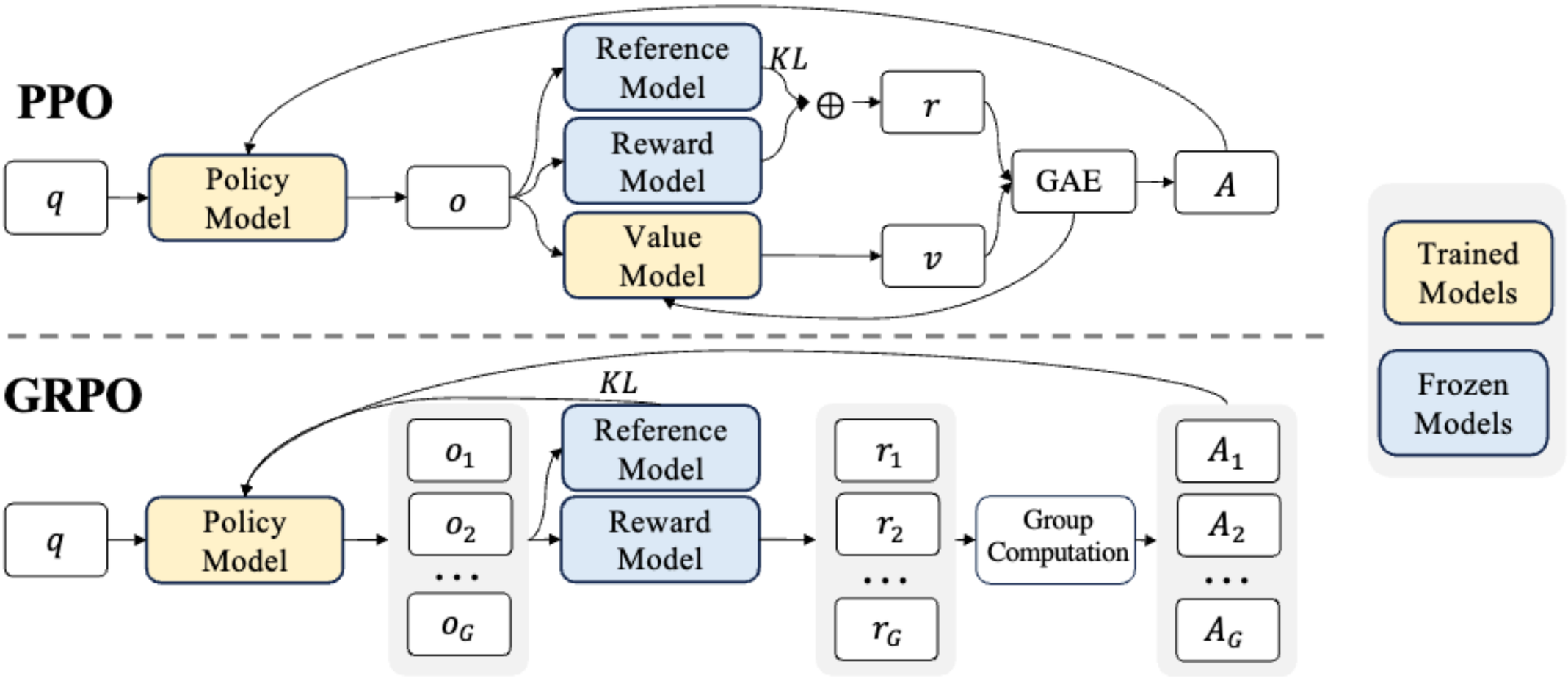
- Process supervision

$$\hat{A}_{i,t} = \sum_{index(j) \geq t} \tilde{r}_i^{index(j)} \qquad \tilde{r}_i^{index(j)} = \frac{r_i^{index(j)} - \text{mean}(\mathbf{R})}{\text{std}(\mathbf{R})}$$

$$\mathbf{r} = \{\{r_1^{index(1)}, \cdots, r_1^{index(K_1)}\}, \ldots, \{r_G^{index(1)}, \cdots, r_G^{index(K_G)}\}\}$$

https://arxiv.org/pdf/2402.03300

Group Relative Policy Optimization (GRPO)



Eliminate value model

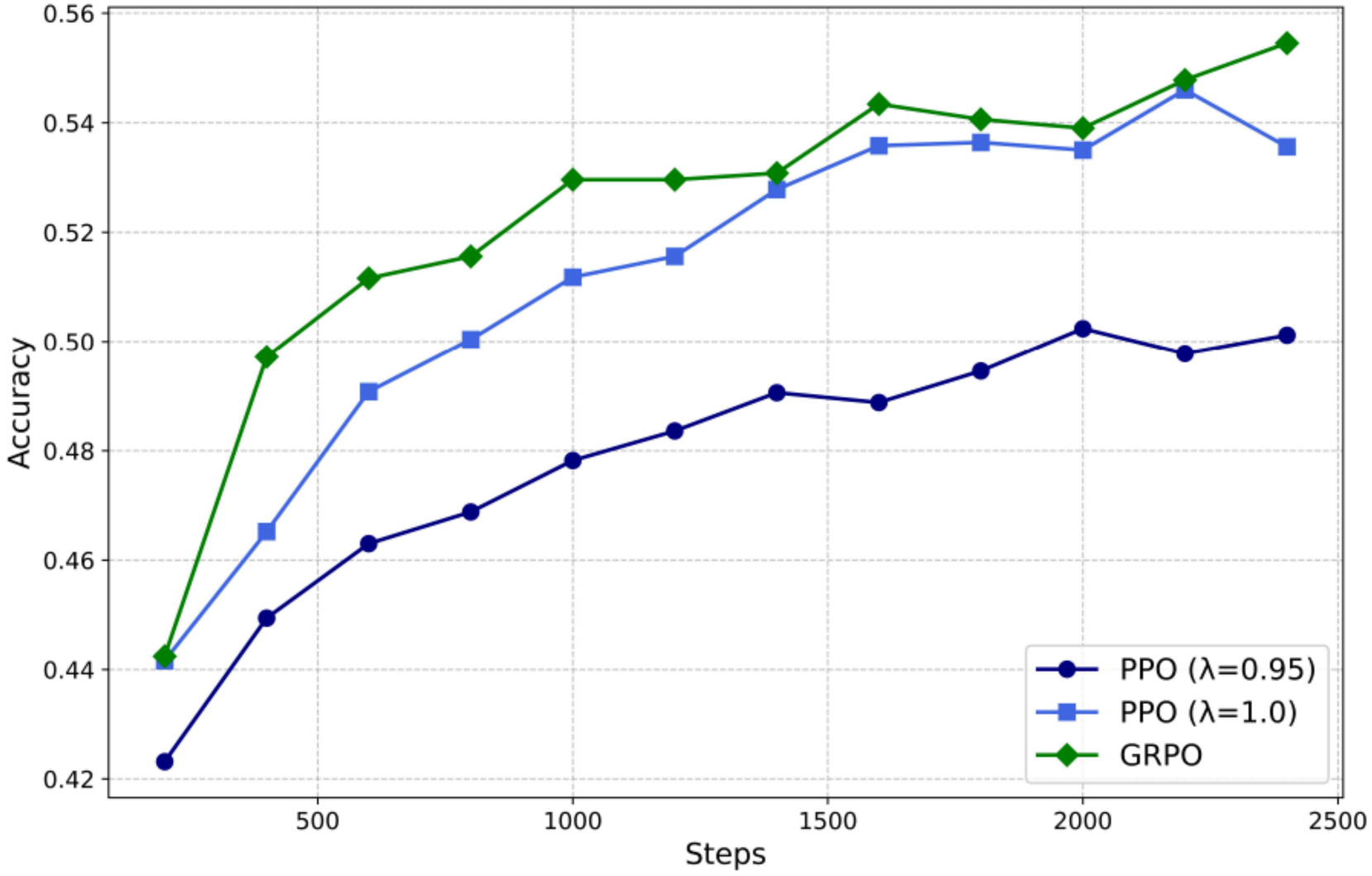
https://arxiv.org/pdf/2402.03300

References for PPO and GRPO

- PPO for LLMs: https://cameronrwolfe.substack.com/p/ppo-llm
- GRPO: https://cameronrwolfe.substack.com/p/grpo
- GRPO Tricks: https://cameronrwolfe.substack.com/p/grpo-tricks
- Online vs Offlice: https://cameronrwolfe.substack.com/p/online-rl

Comparison of PPO and GRPO

MATH task



DeepSeek-R1 [DeepSeek 2025] - https://arxiv.org/pdf/2501.12948

DeepSeek-R1 performance

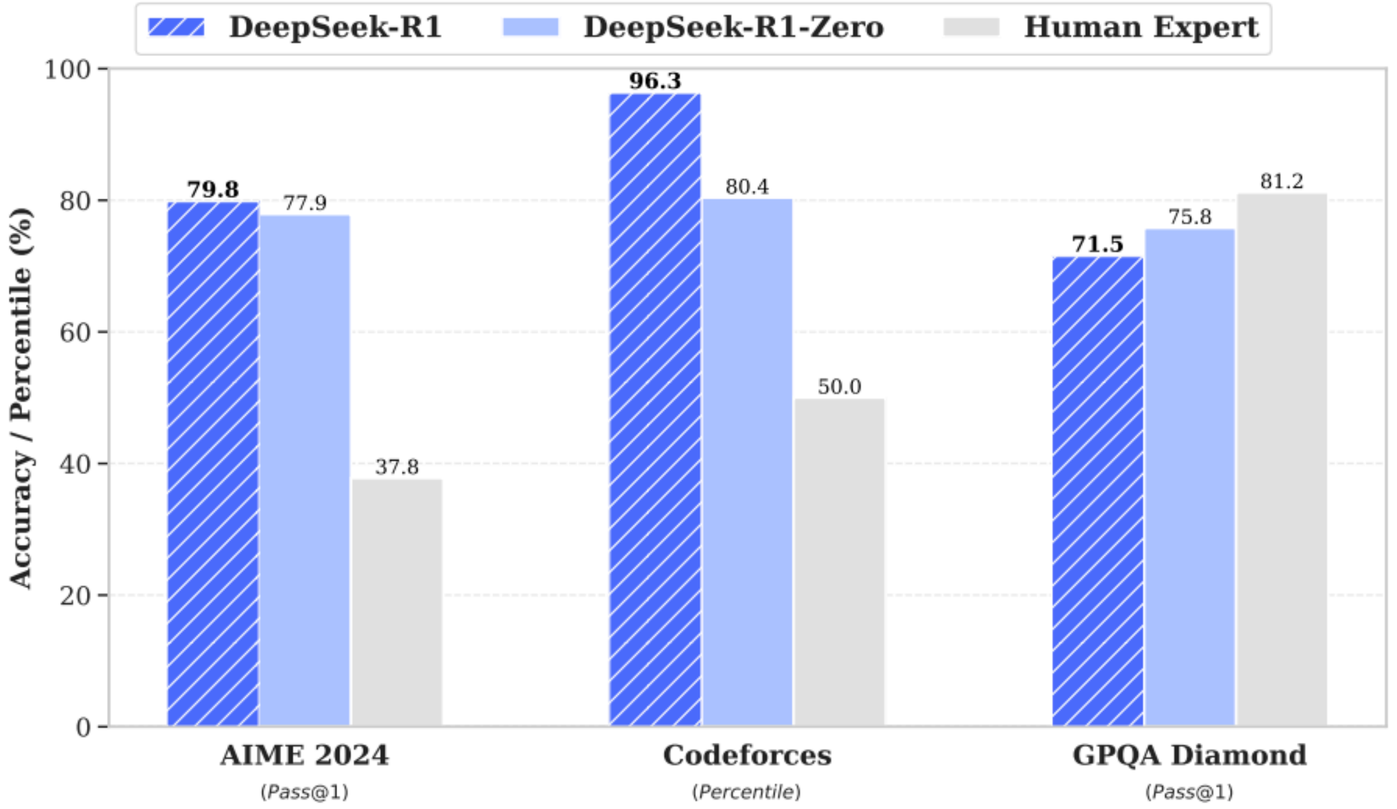
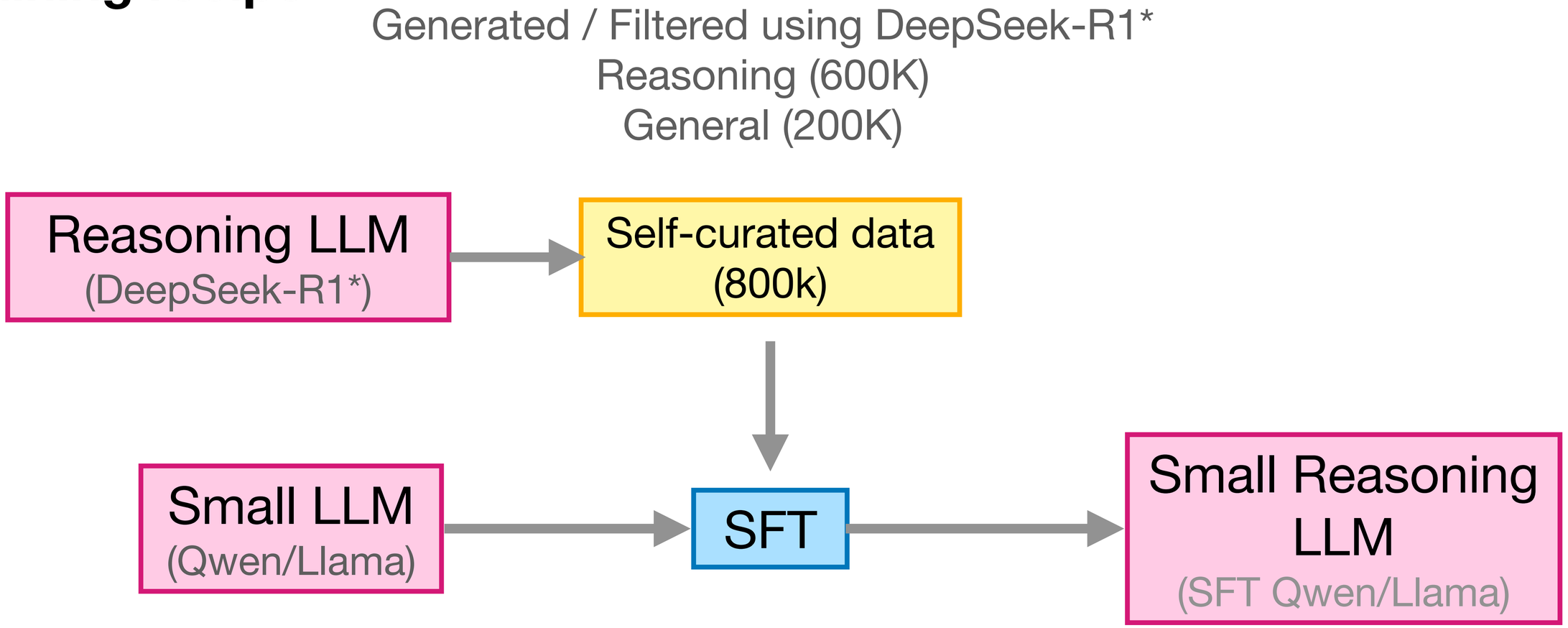


Figure 10 | The benchmark performance of DeepSeek-R1 and DeepSeek-R1-Zero is compared with human scores across different datasets. For AIME and Codeforces, the human scores represent the average performance of all human competitors. In the case of GPQA, the human score corresponds to Ph.D.-level individuals who had access to the web for answering the questions.

DeepSeek-R1 [DeepSeek 2025] - https://arxiv.org/pdf/2501.12948

Distillation of DeepSeek-R1

Training recipe



DeepSeek-R1 [DeepSeek 2025] - https://arxiv.org/pdf/2501.12948

DeepSeek-R1 performance

Distilled models

Table 15 | Comparison of DeepSeek-R1 distilled models and other comparable models on reasoning-related benchmarks. Numbers in bold denote the performance is statistically significant (t−test with $p < 0.01$).

Model	AIME 2024		MATH	GPQA Diamond	LiveCode Bench	CodeForces
	pass@1	cons@64	pass@1	pass@1	pass@1	rating
GPT-4o-0513	9.3	13.4	74.6	49.9	32.9	759
Claude-3.5-Sonnet-1022	16.0	26.7	78.3	65.0	38.9	717
DeepSeek-R1-Distill-Qwen-1.5B	28.9	52.7	83.9	33.8	16.9	954
DeepSeek-R1-Distill-Qwen-7B	55.5	83.3	92.8	49.1	37.6	1189
DeepSeek-R1-Distill-Qwen-14B	69.7	80.0	93.9	59.1	53.1	1481
DeepSeek-R1-Distill-Qwen-32B	**72.6**	83.3	94.3	62.1	57.2	1691
DeepSeek-R1-Distill-Llama-8B	50.4	80.0	89.1	49.0	39.6	1205
DeepSeek-R1-Distill-Llama-70B	70.0	**86.7**	**94.5**	**65.2**	**57.5**	**1633**

DeepSeek-R1 [DeepSeek 2025] - https://arxiv.org/pdf/2501.12948

Does RL increase the reasoning capacity?

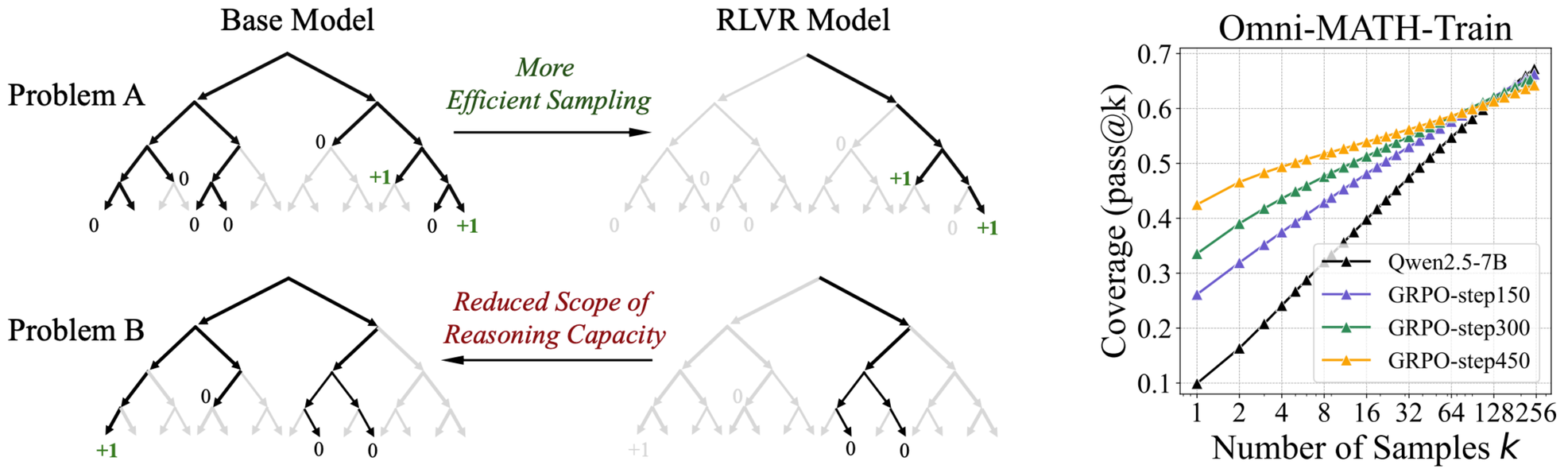


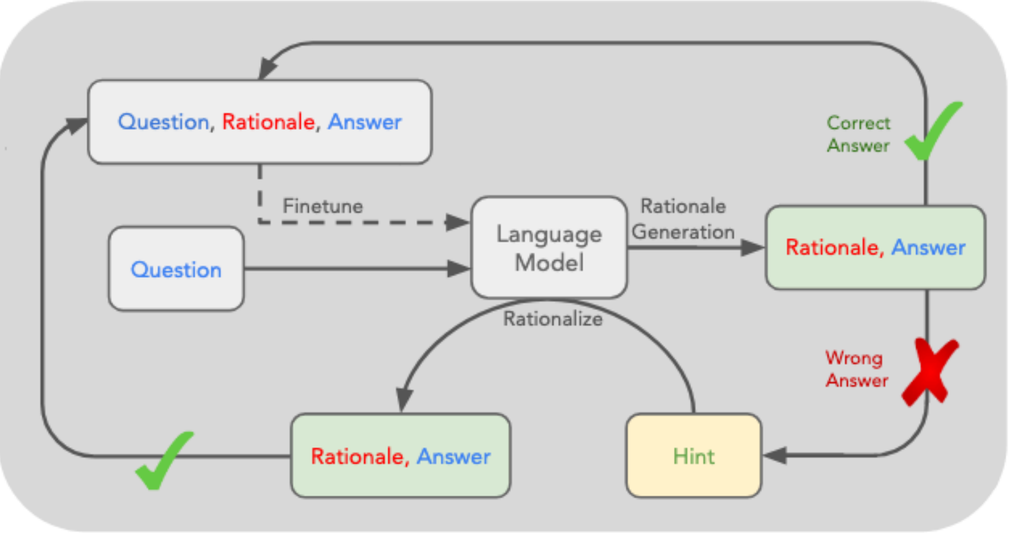
Figure 1: **(Left)** The effect of current RLVR on LLM's reasoning ability. Search trees are generated by repeated sampling from the base and RLVR-trained models for a given problem. Grey indicates paths that are unlikely to be sampled by the model, while **black** indicates paths that are likely to be sampled. Green indicates correct paths, which has positive rewards. Our key finding is that all reasoning paths in the RLVR model are already present in the base model. For certain problems like Problem A, RLVR training biases the distribution toward rewarded paths, improving sampling efficiency. However, this comes at the cost of reduced scope of reasoning capacity: For other problems like Problem B, the base model contains the correct path, whereas that of the RLVR model does not. **(Right)** As RLVR training progresses, the average performance (*i.e.*, pass@1) improves, but the coverage of solvable problems (*i.e.*, pass@256) decreases, indicating a reduction in LLM's reasoning boundary.

Does Reinforcement Learning Really Incentivize Reasoning Capacity in LLMs Beyond the Base Model? [Yue et al. 2025]

Self-training

STaR: Self-Taught Reasoner

- Start with a pretrained LLM
- Assumes dataset with questions and answers.
- Generate rationales
- Fine-tune LLM on correctly generated answer with rationale
- If it cannot generate the correct answer, provide hint of the correct answer

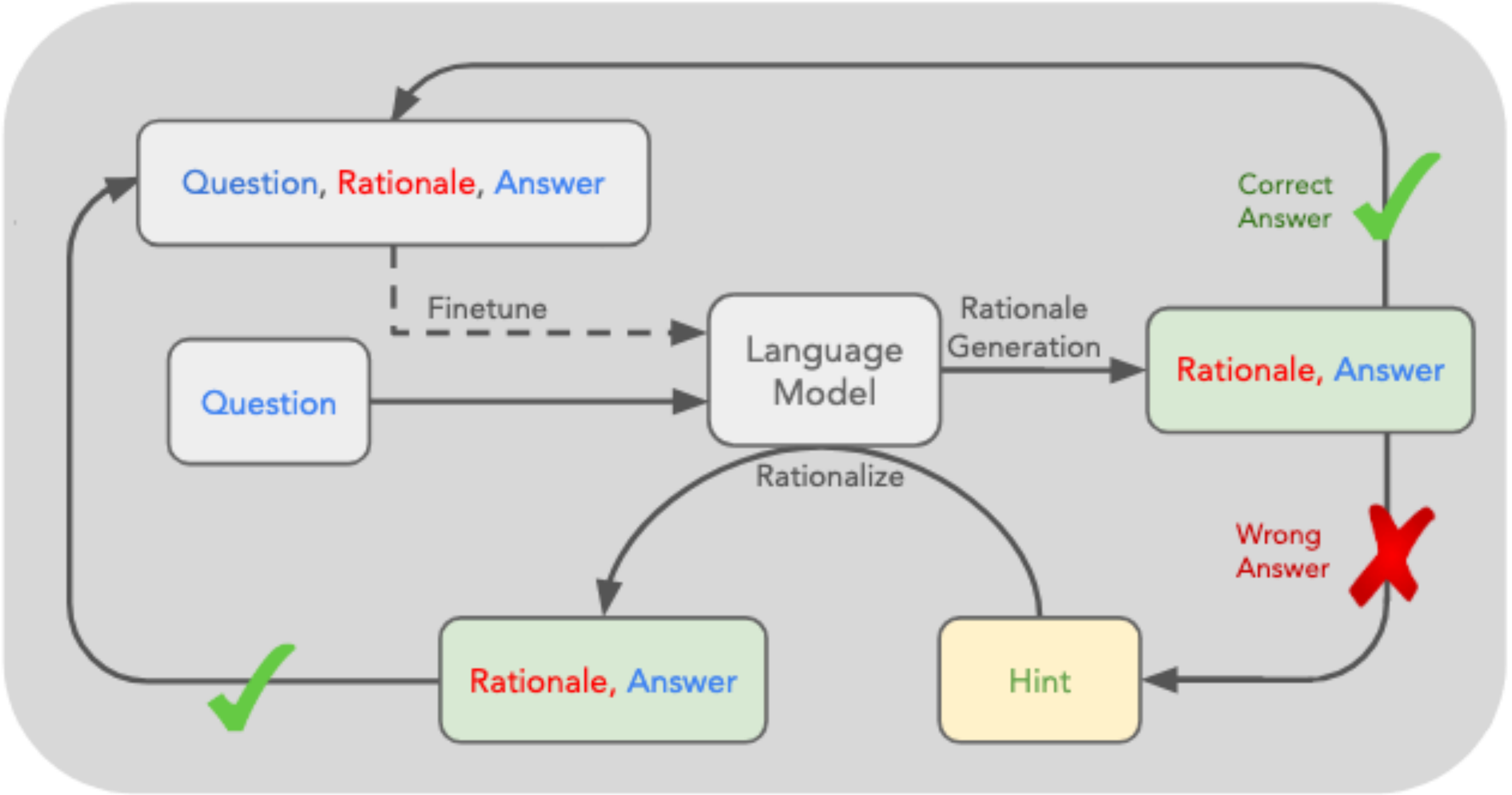



```
Q: What can be used
to carry a small dog?
Answer Choices:
(a) swimming pool
(b) basket
(c) dog show
(d) backyard
(e) own home
A: The answer must be
something that can be
used to carry a small
dog. Baskets are
designed to hold things.
Therefore, the answer
is basket (b).
```

```
Q: Where do you put your grapes just
before checking out?
Answer Choices:
(a) mouth
(b) grocery cart (CORRECT)
(c) super market
(d) fruit basket
(e) fruit market
```

STaR: Self-Taught Reasoner Bootstrapping Reasoning With Reasoning [Zelikman 2022] - https://arxiv.org/pdf/2203.14465

STaR: Self-Taught Reasoner




```
Q: What can be used
to carry a small dog?
Answer Choices:
(a) swimming pool
(b) basket
(c) dog show
(d) backyard
(e) own home
A: The answer must be
something that can be
used to carry a small
dog. Baskets are
designed to hold things.
Therefore, the answer
is basket (b).
```

STaR: Self-Taught Reasoner Bootstrapping Reasoning With Reasoning [Zelikman 2022] - https://arxiv.org/pdf/2203.14465

STaR: Self-Taught Reasoner

Algorithm 1 STaR

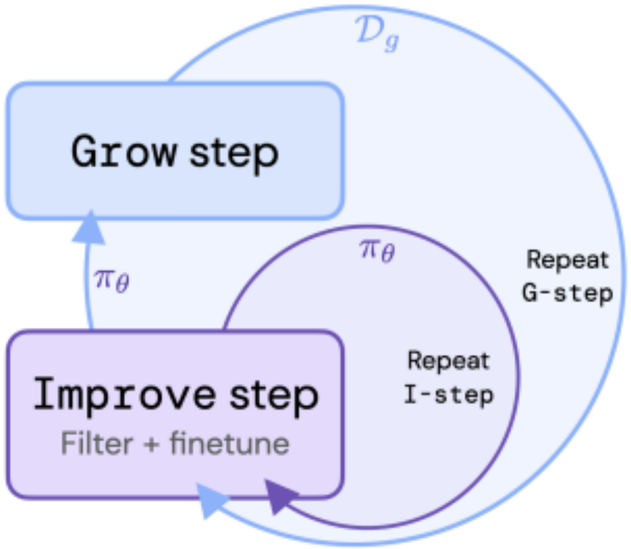
Input M: a pretrained LLM; dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{D}$ (w/ few-shot prompts)

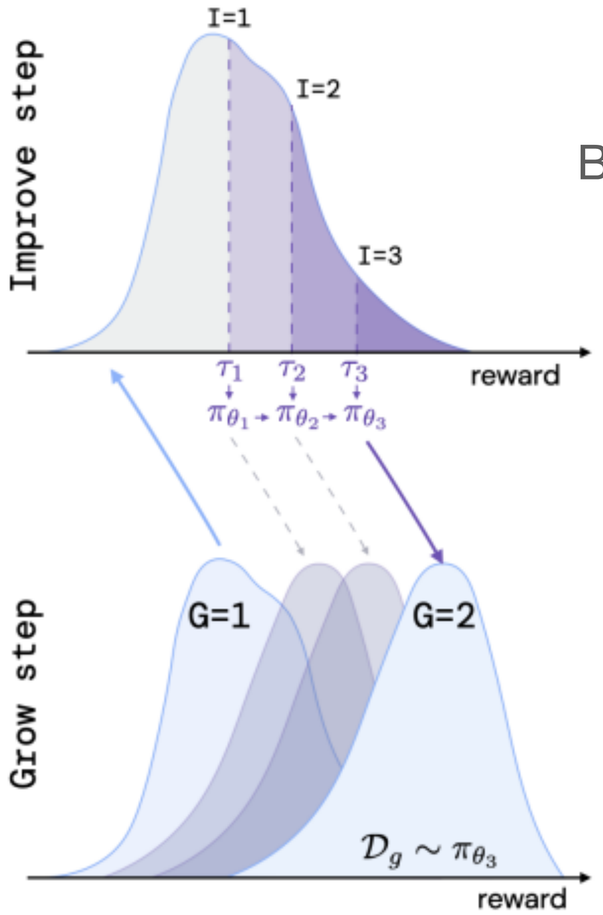
$M_0 \leftarrow M$ # Copy the original model
for n **in** $1...N$ **do** # Outer loop
  $(\hat{r}_i, \hat{y}_i) \leftarrow M_{n-1}(x_i) \quad \forall i \in [1, D]$ # Perform rationale generation
  $(\hat{r}_i^{\text{rat}}, \hat{y}_i^{\text{rat}}) \leftarrow M_{n-1}(\text{add\_hint}(x_i, y_i)) \quad \forall i \in [1, D]$ # Perform rationalization
  $\mathcal{D}_n \leftarrow \{(x_i, \hat{r}_i, y_i) \mid i \in [1, D] \wedge \hat{y}_i = y_i\}$ # Filter rationales using ground truth answers
  $\mathcal{D}_n^{\text{rat}} \leftarrow \{(x_i, \hat{r}_i^{\text{rat}}, y_i) \mid i \in [1, D] \wedge \hat{y}_i \neq y_i \wedge \hat{y}_i^{\text{rat}} = y_i\}$ # Filter rationalized rationales
  $M_n \leftarrow \text{train}(M, \mathcal{D}_n \cup \mathcal{D}_n^{\text{rat}})$ # Finetune the original model on correct solutions - inner loop
end for

STaR: Self-Taught Reasoner Bootstrapping Reasoning With Reasoning [Zelikman 2022] - https://arxiv.org/pdf/2203.14465

ReST: Reinforced Self-Training

- **Grow** - generate dataset for training
- **Improve** - use generated dataset to fine-tune the policy





Better policy is trained on better datasets

Dataset is iteratively filtered to be better (with higher reward thresholds)

Reinforced self-training (ReST) for language modeling [Gulcehre et al 2023] - https://arxiv.org/pdf/2308.08998

ReST: Reinforced Self-Training

Algorithm 1: *ReST* algorithm. *ReST* is a growing-batch RL algorithm. Given an initial policy of reasonable quality (for example, pre-trained using BC) iteratively applies `Grow` and `Improve` steps to update the policy. Here F is a filtering function, and $\mathcal{L}$ is an loss function.

Input: $\mathcal{D}$: Dataset, $\mathcal{D}_{eval}$: Evaluation dataset, $\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}; \theta)$: loss, $R(\boldsymbol{x}, \boldsymbol{y})$: reward model, G: number of grow steps, I: number of improve steps, N: number of samples per context

Train π_θ on $\mathcal{D}$ using loss $\mathcal{L}$.

for $g = 1$ *to* G **do**

  `// Grow`

  Generate dataset $\mathcal{D}_g$ by sampling: $\mathcal{D}_g = \{ (\boldsymbol{x}^i, \boldsymbol{y}^i)|_{i=1}^{N_g} \text{ s.t. } \boldsymbol{x}^i \sim \mathcal{D},\ \boldsymbol{y}^i \sim \pi_\theta(\boldsymbol{y}|\boldsymbol{x}^i) \} \cup \mathcal{D}$.

  Annotate $\mathcal{D}_g$ with the reward model $R(\boldsymbol{x}, \boldsymbol{y})$.

  for $i = 1$ *to* I **do**

    `// Improve`

    Choose threshold s.t. $\tau_1 > V_{\pi_\theta}$ for $V_{\pi_\theta} = \mathbb{E}_{\mathcal{D}_g}[R(\boldsymbol{x}, \boldsymbol{y})]$ and $\tau_{i+1} > \tau_i$.

    while *reward improves on* $\mathcal{D}_{eval}$ **do**

      Optimise θ on objective: $J(\theta) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}_g}\left[F(\boldsymbol{x}, \boldsymbol{y}; \tau_i)\ \mathcal{L}(\boldsymbol{x}, \boldsymbol{y}; \theta)\right]$

    end

  end

end

Output: Policy π_θ

Reinforced self-training (ReST) for language modeling [Gulcehre et al 2023] - https://arxiv.org/pdf/2308.08998

ReST: Reinforced Self-Training

Improved performance on MT with multiple improve steps

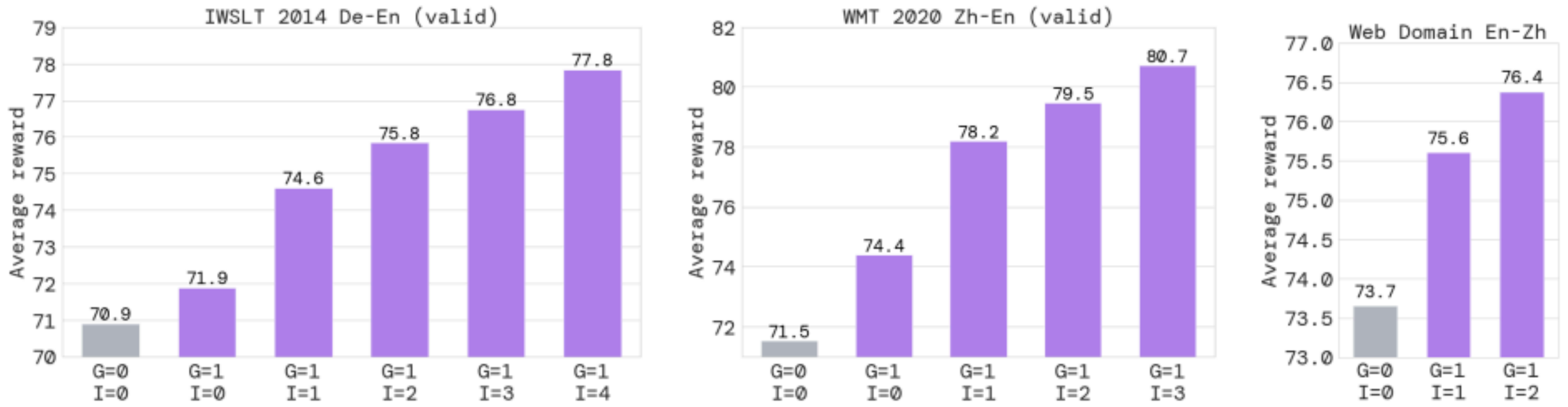


Figure 3 | **ReST with multiple `Improve` steps.** Average reward model scores on IWSLT 2014 De-En, WMT 2020 Zh-En, and Web Domain En-Zh validation sets. On each dataset, we report results with BC ($G = 0,\ I = 0$) and ReST with a single `Grow` step and several `Improve` steps with an increasing reward threshold. Each `Improve` step increases the reward model score in all three validation datasets. We found the suitable number of `Improve` steps to be a dataset-dependent hyperparameter.

Reinforced self-training (ReST) for language modeling [Gulcehre et al 2023] - https://arxiv.org/pdf/2308.08998

ReST: Reinforced Self-Training

Improved performance on MT with two grow steps

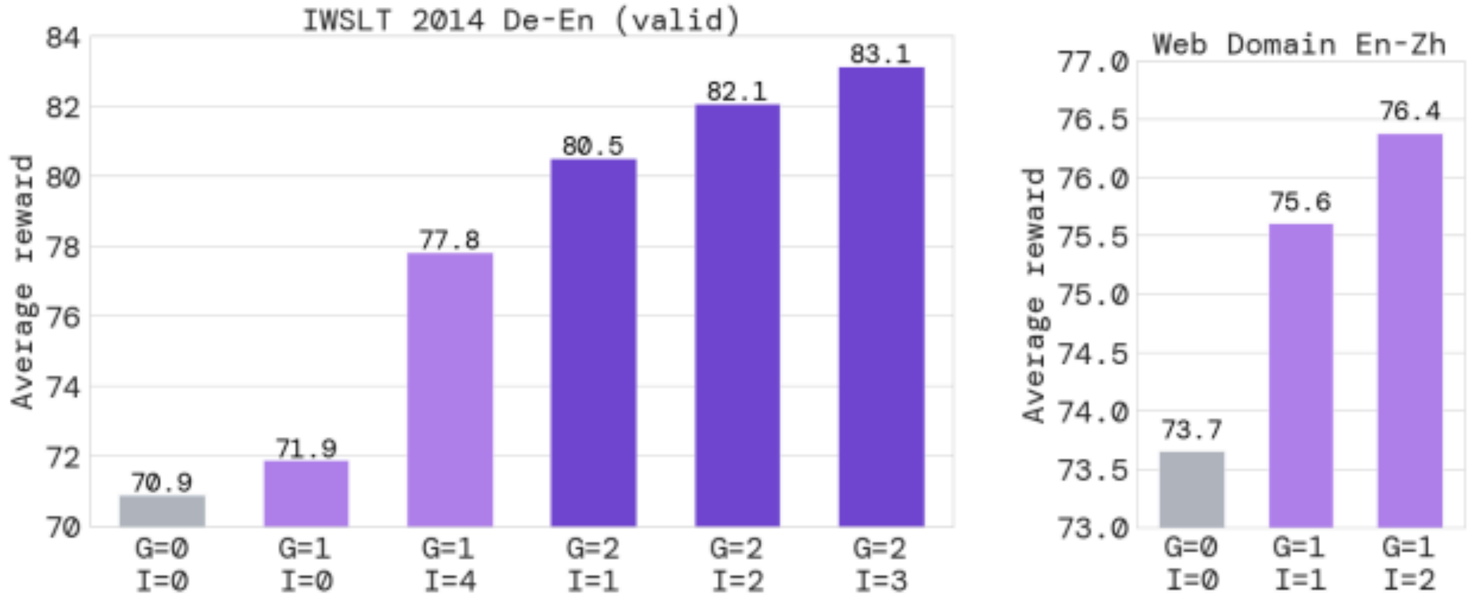


Figure 4 | **ReST with two `Grow` steps.** The second `Grow` step with subsequent `Improve` steps improves the performance by 5.3 points on IWSLT 2014 De-En and 0.8 points on Web Domain En-Zh task over the first `Grow` step.

Reinforced self-training (ReST) for language modeling [Gulcehre et al 2023] - https://arxiv.org/pdf/2308.08998

ReST^{EM}: EM for Reinforced Self-Training

- Expectation maximization for reinforced self-training - simpler version of ReST
  - Generate (E-step) - Generate data by sampling + annotate with reward
  - Improve (M-step) - Optimize parameters with generated data
- Differences from ReST
  - Focus on problems with binary rewards (0/1) vs bounded real-value rewards

Beyond Human Data: Scaling Self-Training for Problem-Solving with Language Models [Singh et al 2023] - https://arxiv.org/pdf/2312.06585

ReST^{EM}: EM for Reinforced Self-Training

Algorithm 1: ReST (Expectation-Maximization). Given a initial policy (e.g., pre-trained LM), ReST^{EM} iteratively applies `Generate` and `Improve` steps to update the policy.

Input: $\mathcal{D}$: Training dataset, $\mathcal{D}_{val}$: Validation dataset, $\mathcal{L}(\boldsymbol{x}, \boldsymbol{y}; \theta)$: loss, $r(\boldsymbol{x}, \boldsymbol{y})$: Non-negative reward function, I: number of iterations, N: number of samples per context

for $i = 1$ *to* I **do**

  `// Generate (E-step)`

  Generate dataset $\mathcal{D}_i$ by sampling: $\mathcal{D}_i = \{ (\boldsymbol{x}^j, \boldsymbol{y}^j)|_{j=1}^{N} \text{ s.t. } \boldsymbol{x}^j \sim \mathcal{D}, \ \boldsymbol{y}^j \sim p_\theta(\boldsymbol{y}|\boldsymbol{x}^j) \}$

  Annotate $\mathcal{D}_i$ with the reward $r(\boldsymbol{x}, \boldsymbol{y})$.

  `// Improve (M-step)`

  while *reward improves on* $\mathcal{D}_{val}$ **do**

    Optimise θ to maximize objective: $J(\theta) = \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}_i} [r(\boldsymbol{x}, \boldsymbol{y}) \ \log p_\theta(\boldsymbol{y}|\boldsymbol{x})]$

  end

end

Output: Policy p_θ

Beyond Human Data: Scaling Self-Training for Problem-Solving with Language Models [Singh et al 2023] - https://arxiv.org/pdf/2312.06585

ReST^{EM}: EM for Reinforced Self-Training

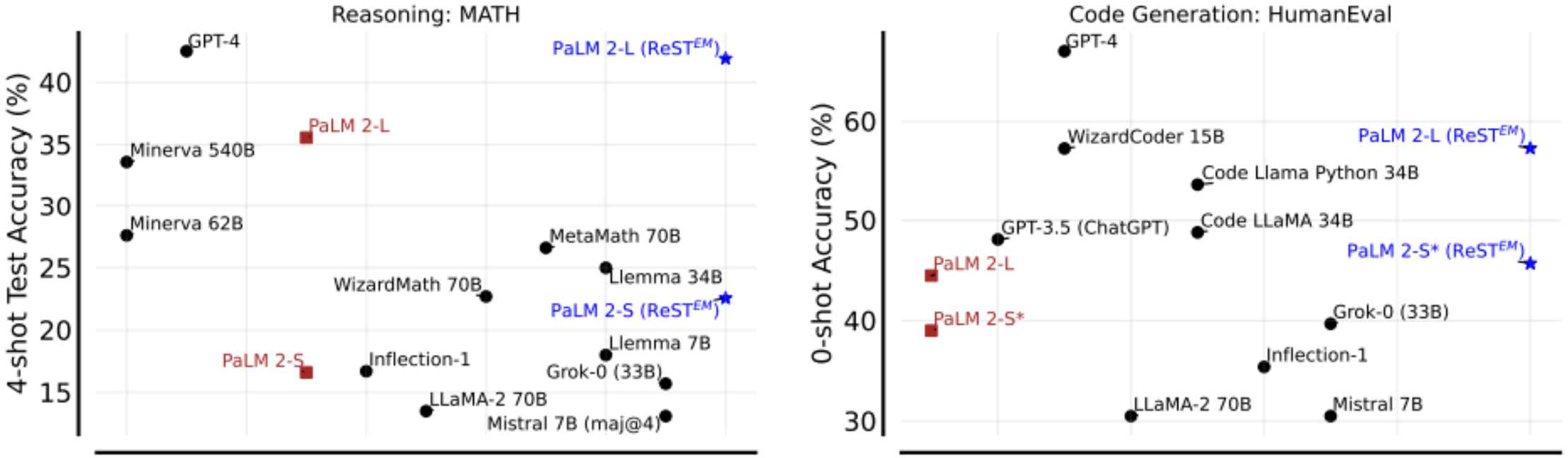


Figure 1 | Self-training with ReST^{EM} substantially improves test performance of PaLM 2 models on two challenging benchmarks: MATH and HumanEval. Results for other models are shown for general progress on these tasks and are typically not comparable due to difference in model scales. GPT-4 results are taken from Bubeck et al. (2023). The x-axis approximately denotes release time (not to scale).

Beyond Human Data: Scaling Self-Training for Problem-Solving with Language Models [Singh et al 2023] - https://arxiv.org/pdf/2312.06585

ReST-MCTS*: LLM Self-Training via Process Reward Guided Tree Search

- Includes training of process reward model
- Iteratively
  - Generate new traces / filter for high reward traces
  - Improve policy model
  - Collect process ward
  - Train process reward model

ReST-MCTS*: LLM Self-Training via Process Reward Guided Tree Search [Zhang et al. 2024] - https://arxiv.org/pdf/2406.03816

ReST-MCTS∗: LLM Self-Training via Process Reward Guided Tree Search

Algorithm 1: Mutual self-training ReST-MCTS* for value model and policy model.

Input: base LLM π, original dataset for policy model D_{S_0}, original dataset for value model D_0, new problem set D_G, number of solutions N, j-th solution A_j, correct solution a^*, value model V_θ, weighted value function w, quality value function v, number of iterations T.

$\pi_{S_0} \leftarrow \text{SFT}(\pi, D_{S_0})$ // fine-tune generator
$D_{V_0} \leftarrow \text{generate\_value\_data}(D_0, w, v)$ // initialize train set for value model
$V_0 \leftarrow \text{train\_value\_model}(V_\theta, D_{V_0})$ // initialize value model
for $i = 1$ to T **do**
  $D_{G_i} \leftarrow \text{generate\_policy\_data}(\pi_{S_{i-1}}, V_{i-1}$ guided MCTS*, $D_G, N)$ // generate synthetic data for policy model
  for $j = 1$ to N **do**
    $D_{G_i(A_j=a^*)} \leftarrow \text{label\_correctness}(D_{G_i})$ // match and select correct solutions
  end for
  $\pi_{S_i} \leftarrow \text{SFT}(\pi_{S_{i-1}}, D_{G_i(A_j=a^*)|_{j=1}^N})$ // self-training policy model
  $D_{V_i} \leftarrow \text{extract\_value\_data}(D_{G_i})$ // collect process reward and extract value data
  $V_i \leftarrow \text{train\_value\_model}(V_{i-1}, D_{V_i})$ // self-training value model
end for

Output: π_{S_T}, V_T

ReST-MCTS∗: LLM Self-Training via Process Reward Guided Tree Search [Zhang et al. 2024] - https://arxiv.org/pdf/2406.03816

ReST-MCTS∗: LLM Self-Training via Process Reward Guided Tree Search

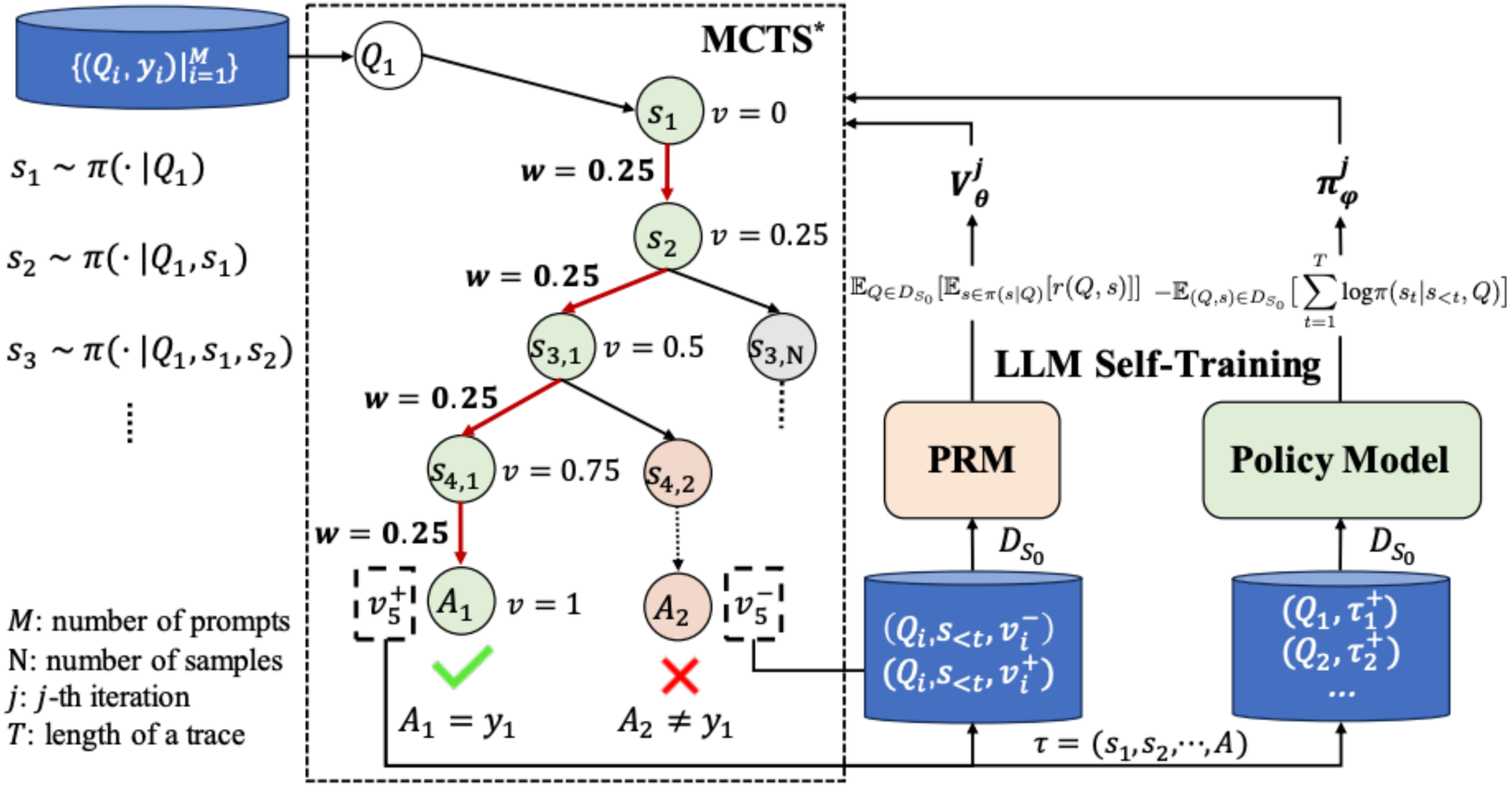


Figure 1: The left part presents the process of inferring process rewards and how we conduct process reward guide tree-search. The right part denotes the self-training of both the process reward model and the policy model.

ReST-MCTS∗: LLM Self-Training via Process Reward Guided Tree Search [Zhang et al. 2024] - https://arxiv.org/pdf/2406.03816

ReST-MCTS∗: LLM Self-Training via Process Reward Guided Tree Search

Table 1: Key differences between existing self-improvement methods and our approach. Train refers to whether to train a reward model.

Method	Reasoning Policy	Reward Guidance	
	Method	Value Label	Train
STaR [4]	CoT+Reflexion	**Final outcome reward** annotated by ground-truth answer	✗
ReST^{EM} [6] / RFT [5] / RPO [11]	CoT		✗
Verify Step-by-Step [1]	Best-of-N	**Per-step process reward** annotated by human	✓
MATH-SHEPHERD [12] / pDPO [13]	Best-of-N	**Per-step process reward** inferred from random rollout	✓
TS-LLM [14]	MCTS	**Per-step process reward** inferred from TD-λ [15]	✓
V-STaR [7]	CoT	**Final outcome reward** generated by multi-iteration LLMs	✓
Self-Rewarding [16]	CoT	**Final outcome reward** generated and judged by LLMs	✗
ReST-MCTS* (Ours)	MCTS*	**Per-step process reward** inferred from tree search (MCTS*)	Multi-Iter

ReST-MCTS∗: LLM Self-Training via Process Reward Guided Tree Search [Zhang et al. 2024] - https://arxiv.org/pdf/2406.03816

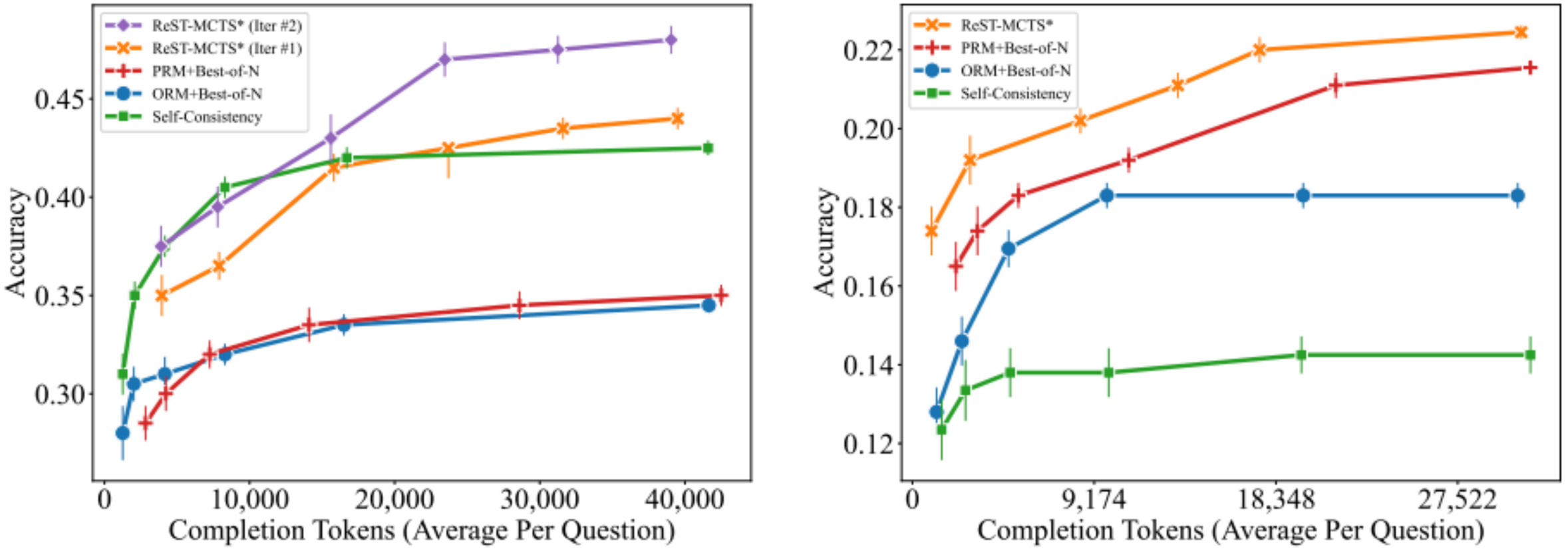
ReST-MCTS∗: LLM Self-Training via Process Reward Guided Tree Search

Table 2: Primary results by training both policy and value model for multiple iterations. For each backbone, different self-training approaches are conducted separately. This means each approach has its own generated train data and corresponding reward (value) model. Our evaluation is zero-shot only, the few-shot baseline only serves as a comparison.

Model	Self-Training Methods	MATH	$\text{GPQA}_{\text{Diamond}}$	CEval-Hard	Ave.
LLaMA-3-8B-Instruct	0th iteration (zero-shot)	20.76	27.27	26.32	24.78
	0th iteration (few-shot)	30.00	31.31	25.66	28.99
	(Below are fine-tuned from model of previous iteration with self-generated traces)				
	w/ ReST^{EM} (1st iteration)	30.84	26.77	21.05	17.22
	w/ Self-Rewarding (1st iteration)	30.34	26.26	25.66	27.42
	w/ **ReST-MCTS∗ (1st iteration)**	31.42	24.24	26.97	27.55
	w/ ReST^{EM} (2nd iteration)	33.52	25.25	21.71	26.83
	w/ Self-Rewarding (2nd iteration)	33.89	26.26	23.03	27.73
	w/ **ReST-MCTS∗ (2nd iteration)**	34.28	27.78	25.00	**29.02**

ReST-MCTS∗: LLM Self-Training via Process Reward Guided Tree Search [Zhang et al. 2024] - https://arxiv.org/pdf/2406.03816

ReST-MCTS∗: LLM Self-Training via Process Reward Guided Tree Search



(a) Self-training of value model on MATH. (b) Comparison of value model on SciBench.

Figure 2: Accuracy of different searches on MATH and SciBench with varied sampling budget.

ReST-MCTS∗: LLM Self-Training via Process Reward Guided Tree Search [Zhang et al. 2024] - https://arxiv.org/pdf/2406.03816