

Reasoning in AI - Second Part

Master 2 Natural Language Processing

- **Lecture 4: Test-Time Scaling & Advanced Inference Strategies**
- **Lecture 5: Emergence of Reasoning Capabilities of LLMs through Post-Training**
- **Lecture 6: Thinking Without Words: Latent Reasoning**

Gaspard Michel

Reasoning in AI - Second Part

Master 2 Natural Language Processing

- **Schedule**

- 26/01 – Lecture 4 (2h)
- 27/01 – Lab 4 (1h30) + MCQ (30 mins)
- **02/01 (Morning)** – Lecture 5 (2h)
- **02/01 (Afternoon)** – End of Lecture 5 + Lab 5
- **03/01 (Afternoon)** – MCQ (Lecture 5) + Lab 5 + Lecture 6
 - **No exam on Lecture 6!**

Gaspard Michel

Reasoning in AI - Second Part

Master 2 Natural Language Processing

- **Lecture 4: Test-Time Scaling & Advanced Inference Strategies**
- **Lecture 5: Emergence of Reasoning Capabilities of LLMs through Post-Training**
- **Lecture 6: Thinking Without Words: Latent Reasoning**

Gaspard Michel

Introduction

- In the previous Lecture, we saw how **test-time scaling** methods were able to elicit **reasoning steps** in LLM generations
- In this lecture, we will rather focus on **training methods that encourage LLMs to reason** when needed:

“Given a strong base LLM, how can we force the LLM to deliberately engage in a reasoning process when it thinks that it is necessary”

Introduction

“Given a strong base LLM, how can we force the LLM to deliberately engage in a reasoning process when it thinks that it is necessary”

- We focus here on situations where the base model is already well-trained and has a good language comprehension
- **Reasons:** prior research found that “reasoning behaviour” were **emerging solely from well-trained models** (linguistic and world knowledge)

Introduction

“Given a strong base LLM, how can we force the LLM to **deliberately engage in a reasoning process** when it thinks that it is necessary”

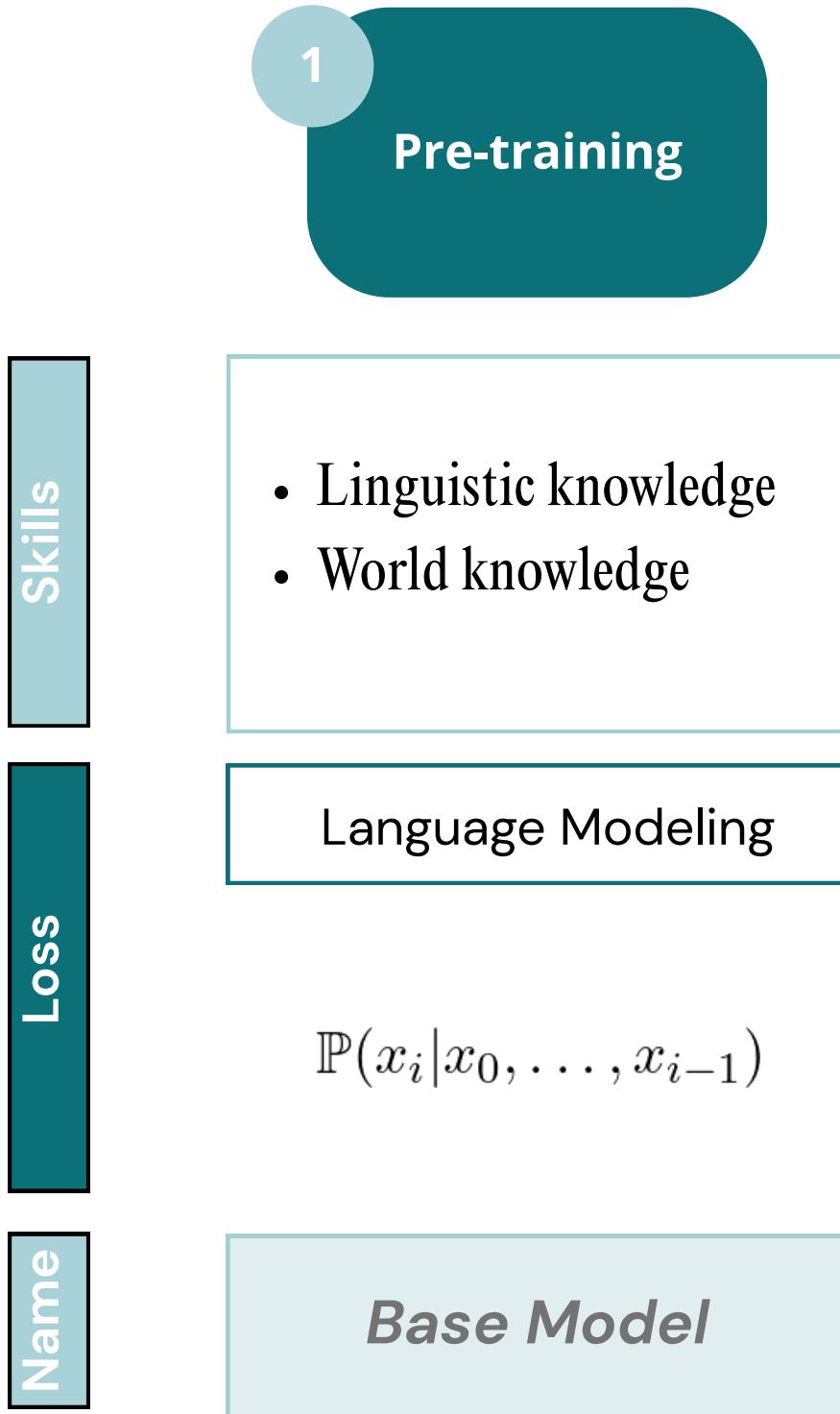
- Highlights the difference with CoT: we don’t “prompt” the model to reason, but it is rather a deliberate model choice
- But CoT is still used at training time to *force the model to reason*

Introduction

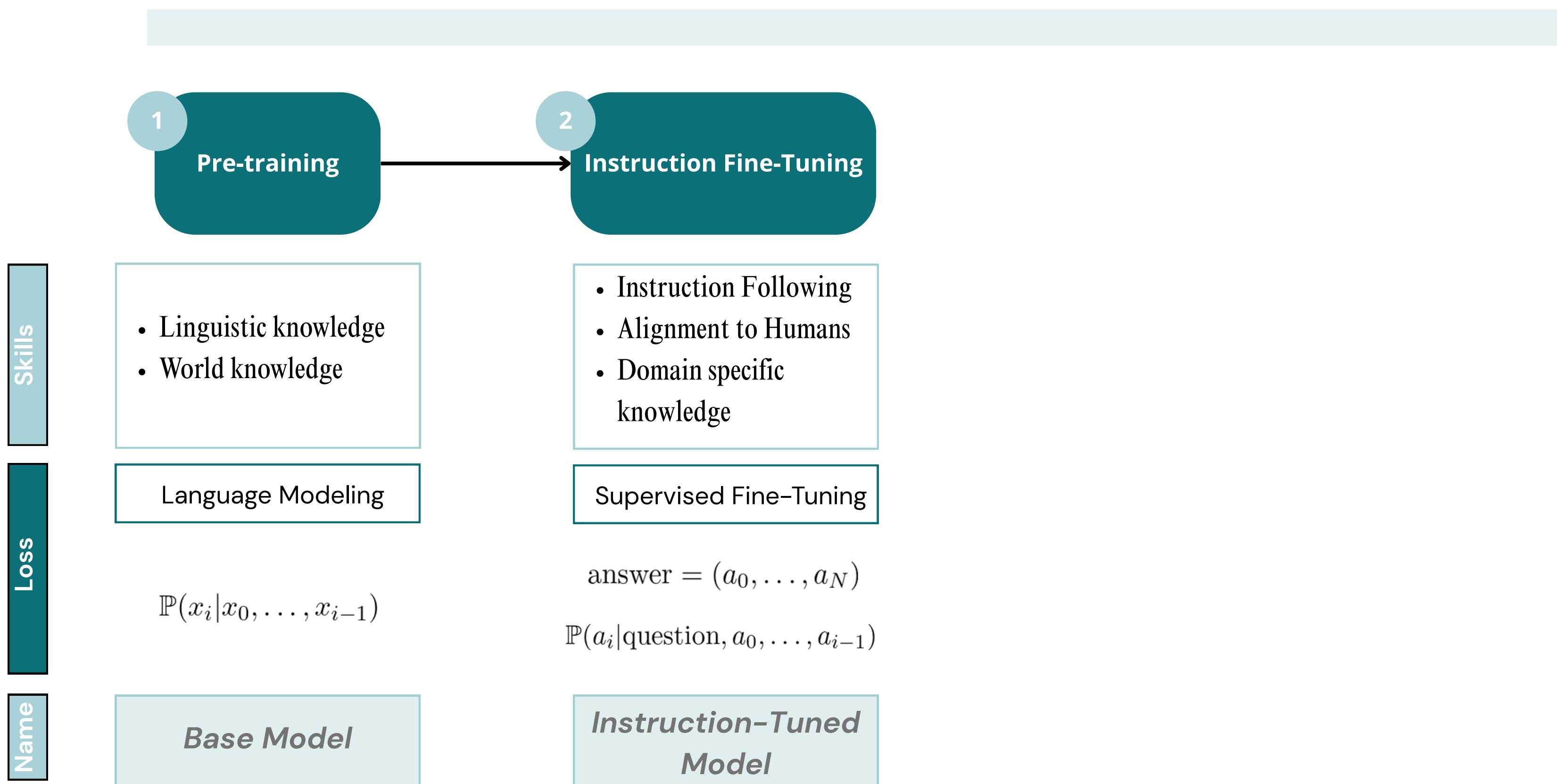
“Given a strong base LLM, how can we force the LLM to deliberately engage in a reasoning process **when it thinks that it is necessary**”

- Relates to the idea of “task difficulty”
- We don’t want the model to spend too much time “thinking” for simple questions
- We’ll see that this notion is **very challenging**

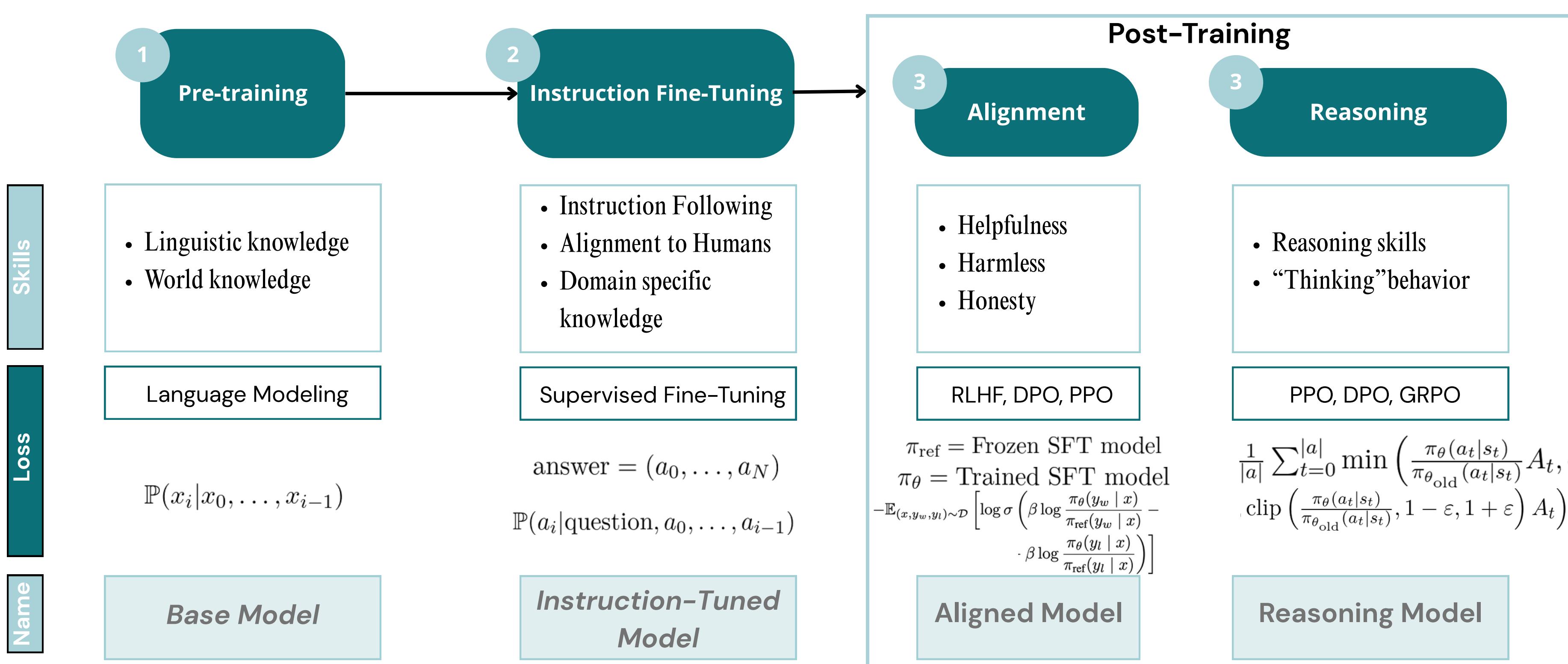
LLM Training Phases



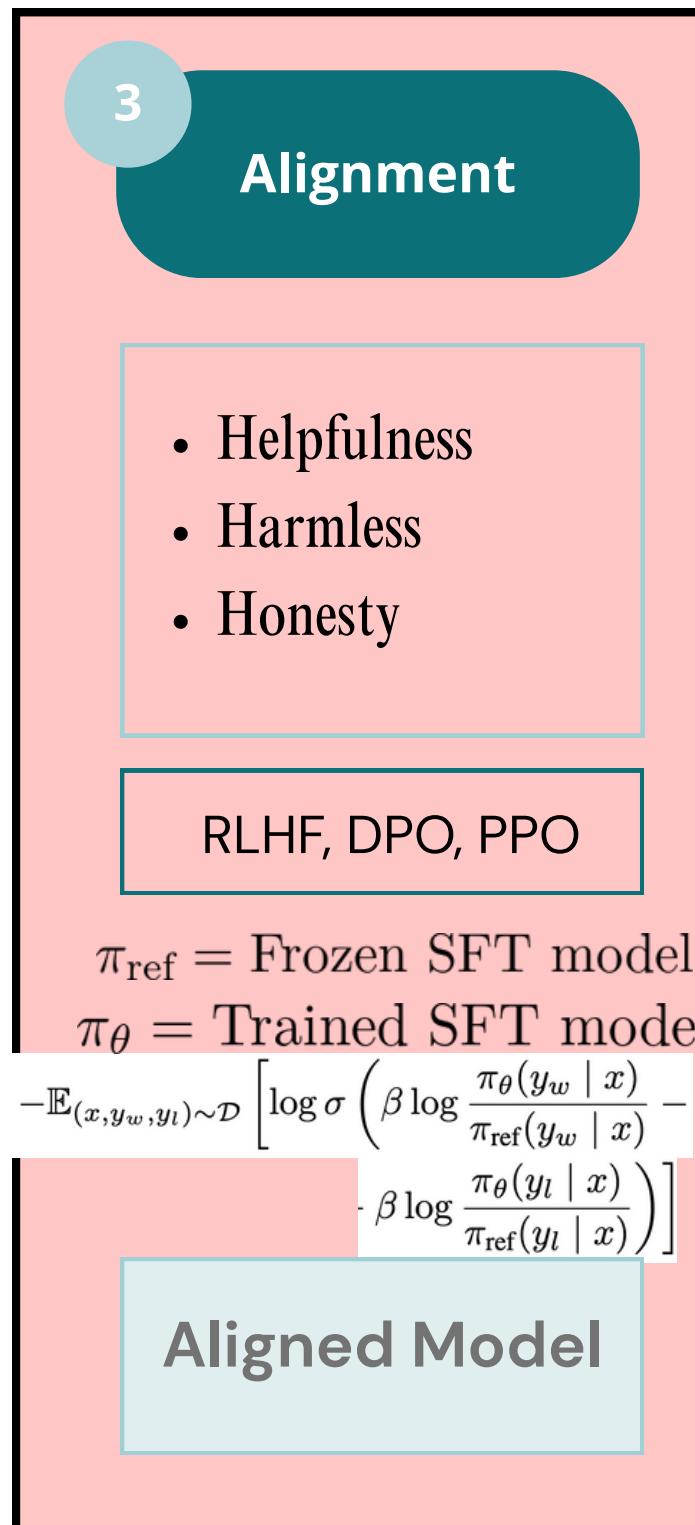
LLM Training Phases



LLM Training Phases



What we won't cover here



- Notes on the **Alignment stage**:
 - Based on **Reinforcement Learning with Human Feedback (RLHF)**
 - Human preference data (x, y_w, y_l)
 - Goal: Align LLM responses to human preferences**
 - Prohibitive cost of annotating preferences
 - Major investment from big companies [1]
 - Can use same RL algo as **Reasoning**

[1] <https://www.interconnects.ai/p/alignment-as-a-service>

Reasoning in AI - Second Part

Master 2 Natural Language Processing

- Lecture 5: Emergence of Reasoning Capabilities of LLMs through Post-Training
 - The Role of Supervised Fine Tuning and RL Intuitions in LLMs
 - Supervision Types in RL
 - RL Basics: Policy, Rewards, Baseline, Advantage
 - Policy Gradient Algorithms
 - Learning from Verifiable and Unverifiable Rewards

The Intuition of Reinforcement Learning

- Given a **state** $s \in \mathcal{S}$ (e.g. where a robot is in a room, a question) and **possible actions** $a \in \mathcal{A}$ (e.g. the robot moves, the LLM answers) find a **trajectory of states and actions** that maximizes **its future discounted rewards**
- In other words, learn a policy π that chooses actions a given a state s_t that maximizes the Reward of the next state $R(s_{t+1})$
- To learn the policy, the trained agent engages in **exploration & trial-and-error** to discover which **states and actions** are valuable

The Intuition of Reinforcement Learning – LLMs

- In Large Reasoning Models, actions $a \in \mathcal{A}$ are the act of generating tokens to answer a prompt $s \in \mathcal{S}$
- The policy π are the model weights
 - RL acts as a training paradigm to forces a base LLM to **reason**, i.e. to generate reasoning steps before answering a prompt, rewarding **usefull reasoning steps**

The Role of Supervised-Fine-Tuning

“Since we have access to human-annotated step-reasoning datasets such as GSM8K, can we just apply SFT on these reasoning traces?”

Math Word Problems (free response)

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.

StrategyQA

Q: Yes or no: Would a pear sink in water?

A: The density of a pear is about 0.6 g/cm³, which is less than water. Thus, a pear would float. So the answer is no.

The Limitations of SFT

- Reasoning datasets gives us access to **positive reasoning examples**
- But no supervision for **examples of negative reasoning**

Math Word Problems (free response)

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.

Wrong Reasoning

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. **3 cans of 2 tennis balls** each is 6 tennis balls. $5 + 6 = 11$.

The answer is 11

The Limitations of SFT

- SFT maximizes the following objective:

$$\max_{\theta} \mathbb{E}_{a_w \sim \pi^*} [\log \pi_{\theta}(a_w | q)]$$

where π^* is the **human distribution** (or policy in RL) stemming from data and π_{θ} is the LLM policy to be optimized

- The corresponding gradients used to update the parameters are:

$$\nabla_{\theta} := \mathbb{E}_{a_w \sim \pi^*} [\nabla \log \pi_{\theta}(a_w | q)]$$

→ maximizing **behaviour reproduction** of policy π^* , never sees other policies π^*

Math Word Problems (free response)

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.

Wrong Reasoning

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. 3 cans of 2 tennis balls each is 6 tennis balls. $5 + 6 = 11$.
The answer is 11

The Limitations of SFT

- In comparison, RL optimizes:

$$\max_{\theta} \mathbb{E}_{a \sim \pi_{\theta}} [R^*(q, a)]$$

For any Reward Functions $R^*(q, .) : \mathcal{T} \times \mathcal{T} \rightarrow \mathbb{R}$
where π_{θ} can be a mix of π and $\bar{\pi}$

- The corresponding (vanilla) updates are:

$$\nabla_{\theta} := \mathbb{E}_{a \sim \pi_{\theta}} [R^*(q, a) \nabla_{\theta} \log \pi_{\theta}(a | q)]$$

- Positive (π) and negative ($\bar{\pi}$) samples
gradient updates are **weighted by their rewards**
- Gradient updates can be more informative than SFT

Math Word Problems (free response)

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.

Wrong Reasoning

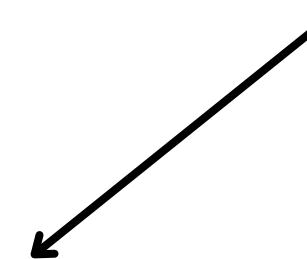
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

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

RL Algorithms

- As we will see later on, modern RL is **often regularized**
- Example with Proximal-Policy-Optimization (PPO):

$$\mathbb{E}_t \left[\min \left(\frac{\pi_\theta(a|q)}{\pi_{\theta_{\text{old}}}(a|q)} A, \text{clip} \left(\frac{\pi_\theta(a|q)}{\pi_{\theta_{\text{old}}}(a|q)}, 1 - \varepsilon, 1 + \varepsilon \right) A \right) - \beta \text{KL} (\pi_{\theta_{\text{ref}}}(\cdot | q), \pi_\theta(\cdot | q)) \right]$$



KL Diverge: measure the distance between two probability distributions

RL Algorithms

- As we will see later on, modern RL is **often regularized**
- Example with Proximal-Policy-Optimization (PPO):

Distance between 2 distributions

$$\mathbb{E}_t \left[\min \left(\frac{\pi_\theta(a|q)}{\pi_{\theta_{\text{old}}}(a|q)} A, \text{clip} \left(\frac{\pi_\theta(a|q)}{\pi_{\theta_{\text{old}}}(a|q)}, 1 - \varepsilon, 1 + \varepsilon \right) A \right) - \beta \text{KL} (\pi_{\theta_{\text{ref}}}(\cdot | q), \pi_\theta(\cdot | q)) \right]$$

$\pi_{\theta_{\text{ref}}}(\cdot | q)$

- Reference Model (LLM)
- Not updated
- Used to regularize policy updates (\uparrow training stability)

$\pi_\theta(\cdot | q)$

- Reasoning Model (LLM)
- Receives gradient updates
- Should not deviate too much from $\pi_{\theta_{\text{ref}}}(\cdot | q)$ (avoids forgetting)

The Role of SFT

$\pi_{\theta_{\text{ref}}}(\cdot \mid q)$

- Reference Model (LLM)
- Not updated
- Used to regularize policy updates (\uparrow training stability)

$\pi_{\theta}(\cdot \mid q)$

- Reasoning Model (LLM)
- Receives gradient updates
- Should not deviate too much from $\pi_{\theta_{\text{ref}}}(\cdot \mid q)$ (avoids forgetting)

- **Regularization** plays a central role in the success of RL training of LLMs
- But it implies that Reference Model $\pi_{\theta_{\text{ref}}}(\cdot \mid q)$ **should already be performant** since KL regularization \rightarrow no major deviations
 - This is why $\pi_{\theta_{\text{ref}}}(\cdot \mid q)$ is typically achieved through **SFT on high-quality data of interest** (e.g. GSM8K-Train) **before RL**

The Intuition of RL

- Recall the Vanilla Gradient Update of RL:

$$\nabla_{\theta} := \mathbb{E}_{a \sim \pi_{\theta}} [R^*(q, a) \nabla_{\theta} \log \pi_{\theta}(a|q)]$$

- In plain words, it means the **model generates answers**, which are **rewarded if they succeed** and **penalized if they fail**
- Thus **RL** can be seen as a **process of trial and error**, where the model is rewarded for **finding optimal strategies** that lead to sucessful reward signals
 - The **reward signals** play a crucial role in sucess of RL training

Recap

- SFT learn to mimic **in-distribution data**, and might struggle to generalize to unseen distribution
 - It only sees successful examples!
- In contrast, RL acts through **trial and error**, learning from **reward signals**
 - Gradient updates are usually more informative than SFT
 - Initial Reference Model needs to be good enough
- SFT acts as a “**warm-up**” for the reference policy, bridging the gap towards **powerful Reasoning Models**

Reasoning in AI - Second Part

Master 2 Natural Language Processing

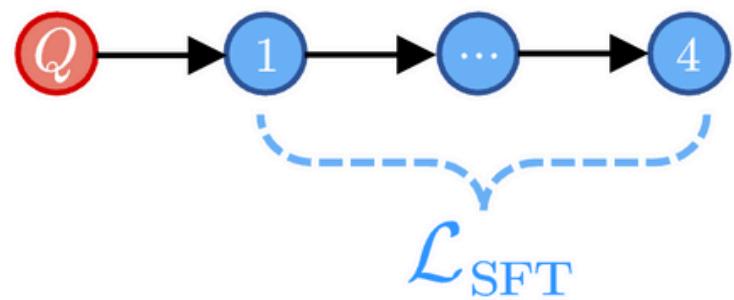
- Lecture 5: Emergence of Reasoning Capabilities of LLMs through Post-Training
 - The Role of Supervised Fine Tuning and RL Intuitions in LLMs
 - Supervision Types in RL
 - RL Basics: Policy, Rewards, Baseline, Advantage
 - Policy Gradient Algorithms
 - Learning from Verifiable and Unverifiable Rewards

Types of supervision for Reasoning Models

Given a model answer, how does gradient updates flow for each generated token?

Types of supervision for Reasoning Models

Supervised Fine-Tuning

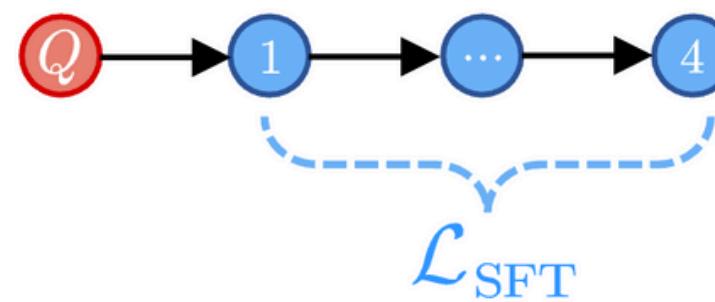


Typically the Language Modeling task

Token-level supervision
(predict next word)

Types of supervision for Reasoning Models

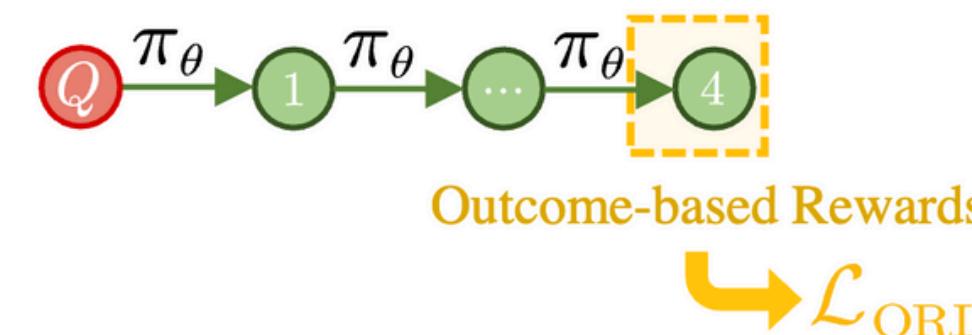
Supervised Fine-Tuning



Typically the Language Modeling task

Token-level supervision
(predict next word)

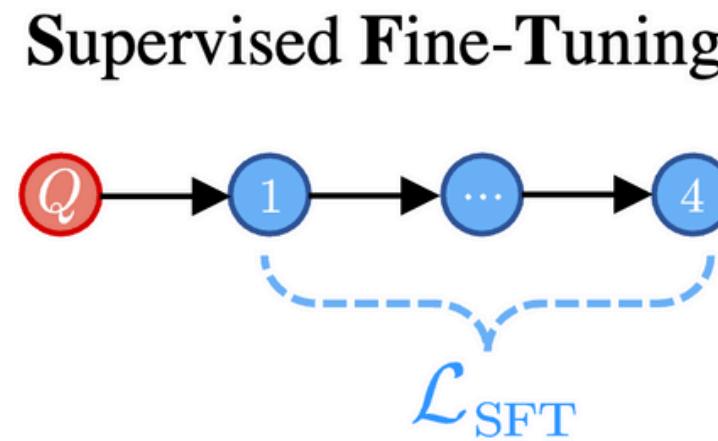
Outcome-Supervised RL



Each token gets a reward
based on the **answer**
reward

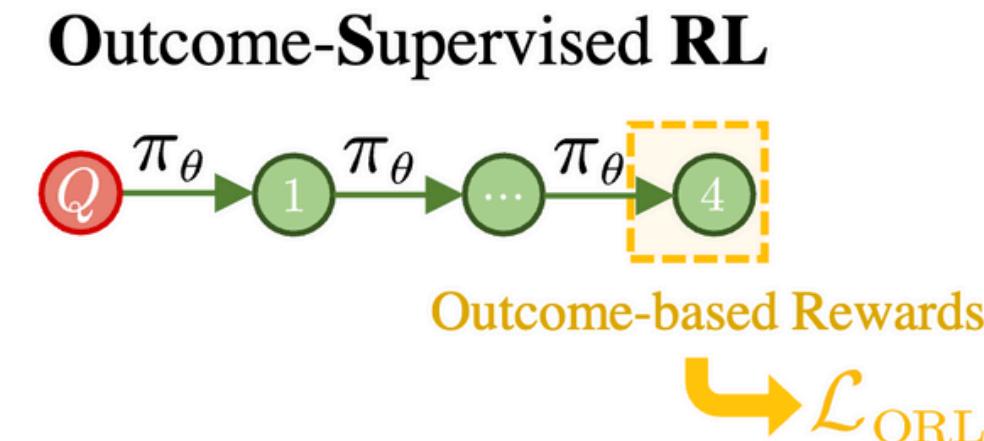
No rewards to
independent steps

Types of supervision for Reasoning Models



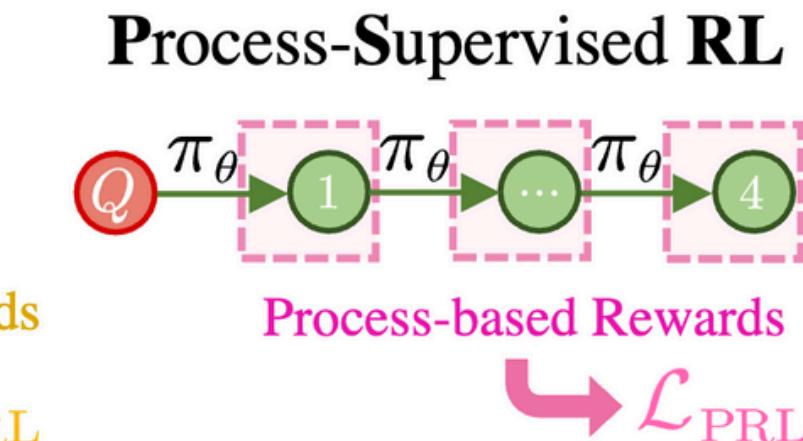
Typically the Language Modeling task

Token-level supervision
(predict next word)



Each token gets a reward
based on the answer
reward

No rewards to
independent steps

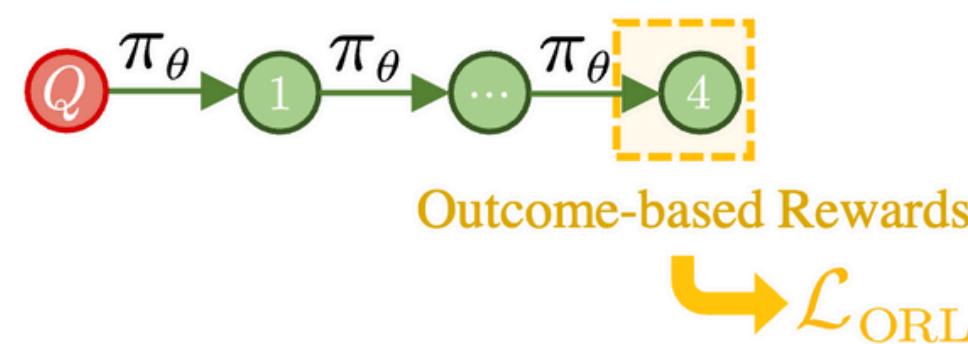


Scenarios where we can
evaluate the success of
each **reasoning step**

Token-level update based
on participation to **good
reasoning steps**

Types of supervision for Reasoning Models

Outcome-Supervised RL

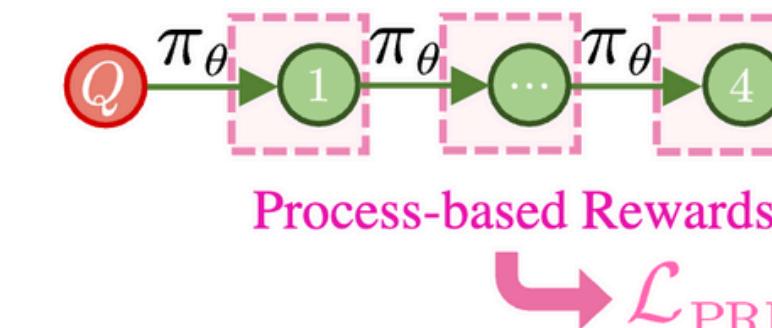


Easy to use: just verify the answer against ground-truth (math, code)

Might reward bad reasoning if it leads to correct answer

Does not rewards *individual successful reasoning steps*

Process-Supervised RL



Rewards *individual successful reasoning steps*

Theoretically more appealing, but requires extremely costly human annotation: is a step useful or not?

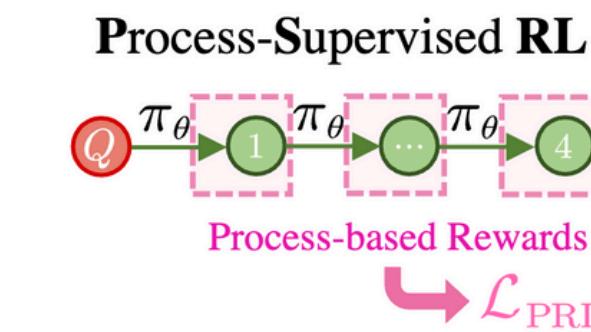
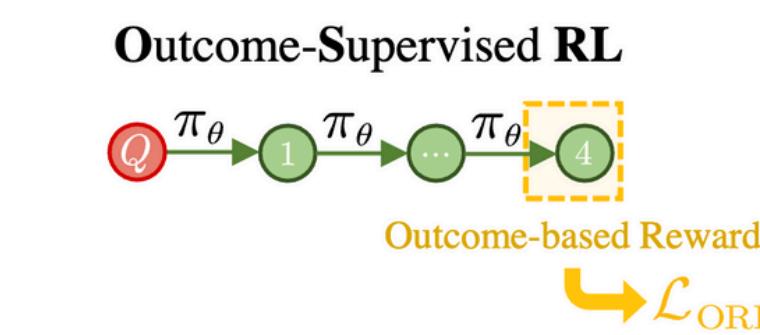
Types of supervision for Reasoning Models

- Given a model answer $a = (a_0, \dots, a_N)$, ORL and PRL influence what gradient update each individual token a_i will receive

Wrong Reasoning

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. **3 cans of 2 tennis balls** each is 6 tennis balls. $5 + 6 = 11$. The answer is 11



Tell me what you think
about gradient updates?

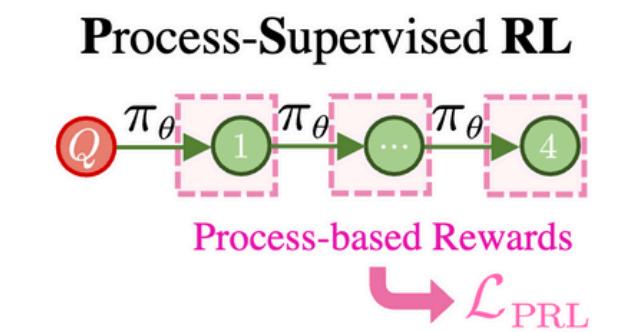
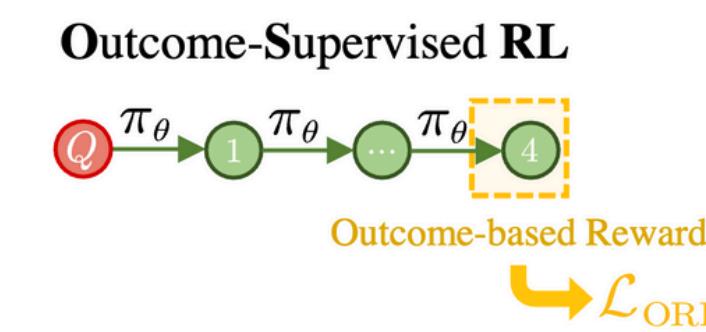
Types of supervision for Reasoning Models

- Given a model answer $a = (a_0, \dots, a_N)$, ORL and PRL influence what gradient update each individual token a_i will receive

Wrong Reasoning

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. 3 cans of 2 tennis balls each is 6 tennis balls. $5 + 6 = 11$.
The answer is 11



- Both green and red segments receive positive updates
- green receives positive update
- red receives negative update

Types of supervision for Reasoning Models

- The rest of this course will focus on **Outcome-based RL** because its conceptually simpler than **Process-based RL**
- But all algorithms presented next can be applied in both ORL and PRL
- The only difference is how gradient updates will flow towards specific token segments

Reasoning in AI - Second Part

Master 2 Natural Language Processing

- Lecture 5: Emergence of Reasoning Capabilities of LLMs through Post-Training
 - The Role of Supervised Fine Tuning and RL Intuitions in LLMs
 - Supervision Types in RL
 - RL Basics: Policy, Rewards, Baseline, Advantage
 - Policy Gradient Algorithms
 - Learning from Verifiable and Unverifiable Rewards

Vanilla Gradient Updates

- Recall from later the **Vanilla Gradient Update formula**:

$$\nabla_{\theta} := \mathbb{E}_{\mathbf{a} \sim \pi_{\theta}} [R^*(q, \mathbf{a}) \nabla_{\theta} \log \pi_{\theta}(\mathbf{a}|q)]$$

- This statistical object (expectation) can be re-written as:

$$\nabla_{\theta} = \int_a \nabla_{\theta} \pi_{\theta}(a|q) R(a, q) da$$

- This value is **intractable**: we can't compute its exact value as it would require "sampling all possible LLM answers"

Monte Carlo Estimation

- In practice, we can **estimate its value using its Monte Carlo estimation**
- This is quite simple:
 - Sample B “**rollouts**” (i.e. (question, answers) pairs) under current model π_θ
 - Calculate gradient for each rollout:

$$\nabla(q_i, a_i; \theta) = \nabla_\theta \log \pi_\theta(a_i | q_i) R(a_i, q_i)$$

- Average over rollouts

$$\nabla_\theta = \frac{1}{B} \sum_{i=1}^B \nabla(q_i, a_i; \theta)$$

Monte Carlo Estimation

- In practice, we can **estimate its value using its Monte Carlo estimation**
- This is quite simple:

Monte Carlo Estimation

- In practice, we can **estimate its value using its Monte Carlo estimation**
- This is quite simple:
 - Sample B “*rollouts*” (i.e. *(question, answers)* pairs) under current model π_θ

Monte Carlo Estimation

- In practice, we can **estimate its value using its Monte Carlo estimation**
- This is quite simple:
 - Sample B “**rollouts**” (i.e. (question, answers) pairs) under current model π_θ
 - Calculate gradient for each rollout:

$$\nabla(q_i, a_i; \theta) = \nabla_\theta \log \pi_\theta(a_i | q_i) R(a_i, q_i)$$

Monte Carlo Estimation

- In practice, we can **estimate its value using its Monte Carlo estimation**
- This is quite simple:
 - Sample B “**rollouts**” (i.e. (question, answers) pairs) under current model π_θ
 - Calculate gradient for each rollout:

$$\nabla(q_i, a_i; \theta) = \nabla_\theta \log \pi_\theta(a_i | q_i) R(a_i, q_i)$$

- Average over rollouts

$$\nabla_\theta = \frac{1}{B} \sum_{i=1}^B \nabla(q_i, a_i; \theta)$$

Monte Carlo Estimation – Instabilities

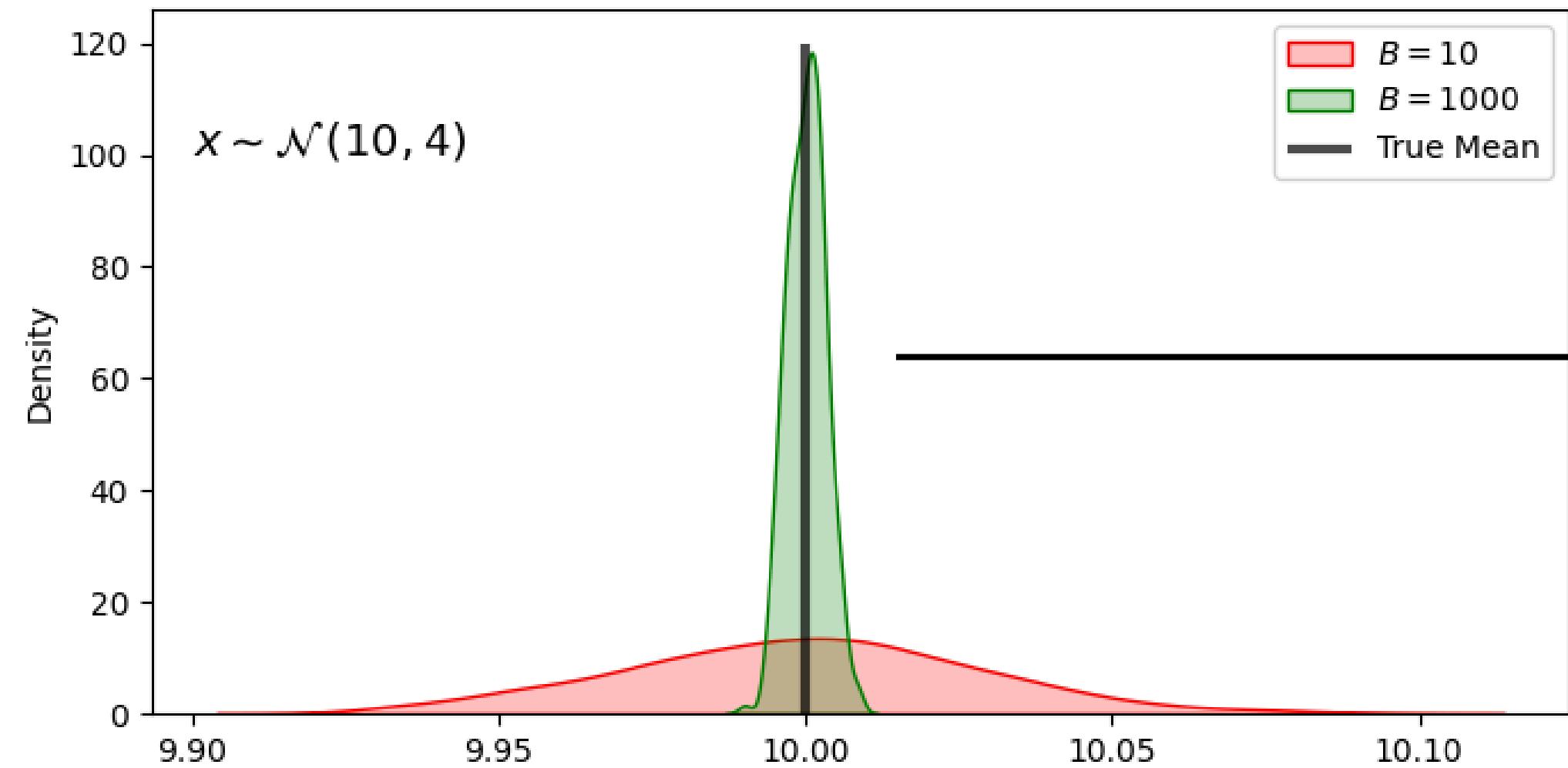
- In practice, we can **estimate its value using its Monte Carlo estimation**
- **Pros:** we can compute it easily
- **Cons:** the smaller the B the more **inaccurate** the estimation is

Monte Carlo Estimation – Instabilities

- In practice, we can **estimate its value using its Monte Carlo estimation**
- **Pros:** we can compute it easily
- **Cons:** the smaller the B the more **inaccurate** the estimation is
 - In statistical terms, it means the Monte Carlo estimator has a **large variance**
 - LLMs can't use a large B because of GPU memory constraints
 - Thus the gradient updates are often **noisy**, leading to **training instabilities**

Monte Carlo Estimation – Variance

- Example: trying to estimate $\mathbb{E}[x]$ for $x \sim \mathcal{N}(10, 4)$



That's what we want!

Monte Carlo Estimation – Variance Reduction

- A typical solution is to **normalize the Reward Formulation**
- We typically use a **Baseline to stabilize the gradient updates** $V(q_i)$
 - Measures the **Expected Reward/Return** of question q
- This baseline is then used to calculate **the Advantage**
$$A(q,a) = R(q,a) - V(q)$$
 - Measures the how much the answer a is better (in terms of rewards) than the expected reward for question q

"Was this token/answer better or worse than what we'd typically expect in this context?"

Monte Carlo Estimation – Variance Reduction

- The gradient updates becomes

$$\nabla_{\theta} := \mathbb{E}_{a \sim \pi_{\theta}} [A(q, a) \nabla_{\theta} \log \pi_{\theta}(a|q)]$$

with $A(q,a) = R(q,a) - V(q)$

Monte Carlo Estimation – Baselines

- Many ways of calculating the baseline $V(q_i)$
 - Most simple is the **Average Reward** over a batch of rewards

$$\forall i = 1, \dots, B \quad V(q_i) = \frac{1}{B} \sum_{j=1}^B R(q_j, a_j)$$

- Can also be learned as in PPO

RL Basics: Returns

- So far, we have seen:
 - **Rewards** $R(q,a)$ that rewards good answers a to questions q
 - **Baseline** $V(q)$ that measures the expected Reward for question q , serving as variance reduction for gradient estimation
 - **Advantage** $A(q,a) = R(q,a) - V(q)$ that measures how much an answer a is better than what we could have expected from question q
 - We often optimize the Advantage rather than the Reward directly

RL Basics: Returns

- There's an other mathematical object called **Returns**
- We focused before on (question, answer) pairs, but in the context of LLMs, answers (and questions) are **processed at the Token-Level**
- **Returns** tries to answer the following question:

*"The reward only scores finished responses ; but hundreds of token decisions were made along the way.
Which tokens actually deserved credit (or blame)?"*

RL Basics: Returns

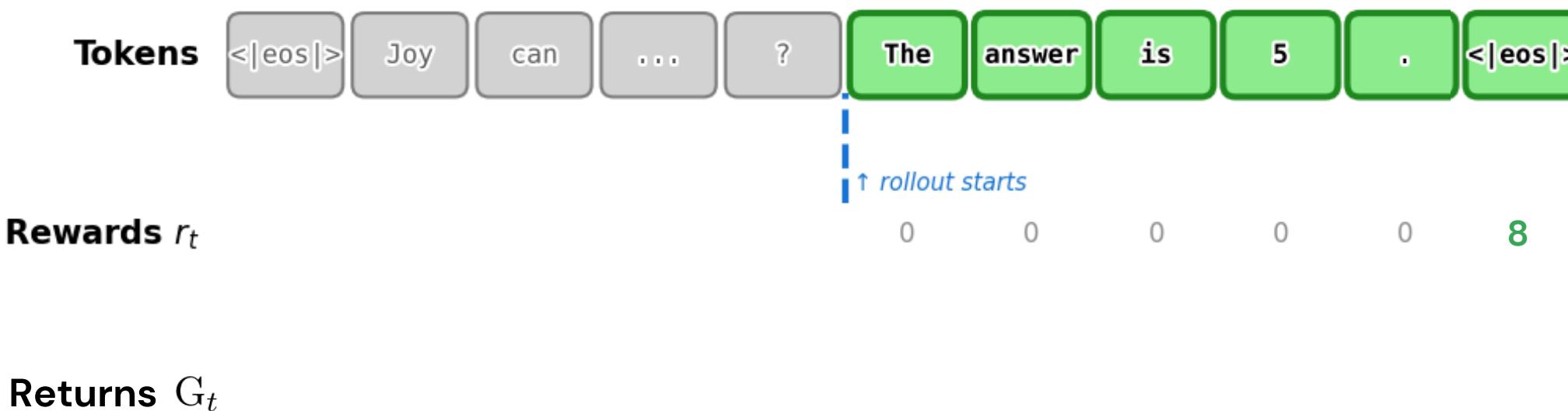
- Mathematically, let $a = (a_0, \dots, a_T)$ then the **Return** for token t is

$$G_t = \sum_{k=t}^T \gamma^{(k-t)} R_{k+1} = \gamma G_{t+1} + R_{t+1}$$

RL Basics: Returns

- Mathematically, let $a = (a_0, \dots, a_T)$ then the **Return** for token t is

$$G_t = \sum_{k=t}^T \gamma^{(k-t)} R_{k+1} = \gamma G_{t+1} + R_{t+1}$$

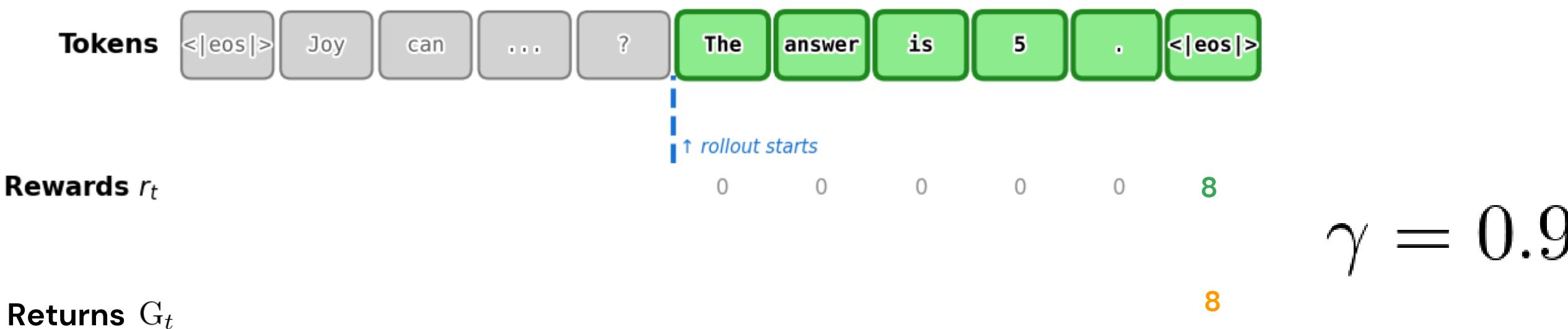


$$\gamma = 0.9$$

RL Basics: Returns

- Mathematically, let $a = (a_0, \dots, a_T)$ then the **Return** for token t is

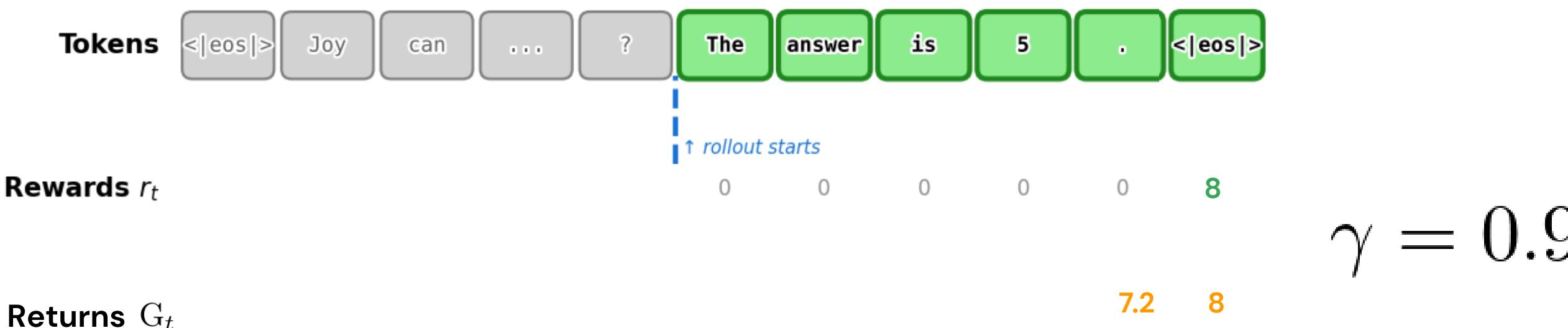
$$G_t = \sum_{k=t}^T \gamma^{(k-t)} R_{k+1} = \gamma G_{t+1} + R_{t+1}$$



RL Basics: Returns

- Mathematically, let $a = (a_0, \dots, a_T)$ then the **Return** for token t is

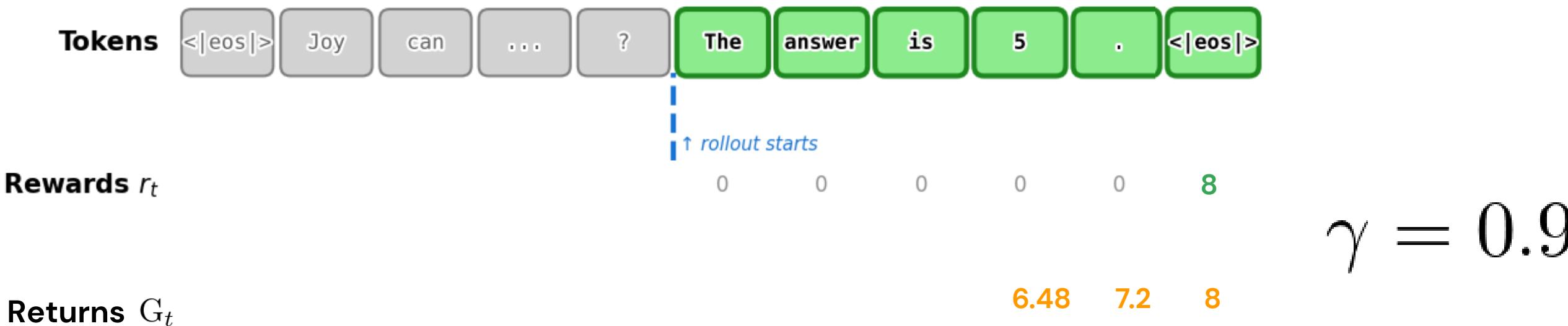
$$G_t = \sum_{k=t}^T \gamma^{(k-t)} R_{k+1} = \gamma G_{t+1} + R_{t+1}$$



RL Basics: Returns

- Mathematically, let $a = (a_0, \dots, a_T)$ then the **Return** for token t is

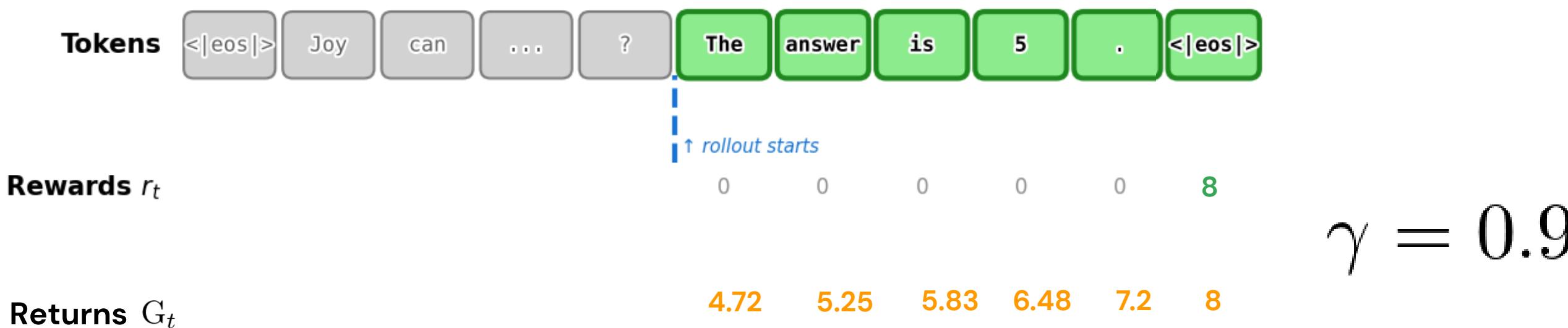
$$G_t = \sum_{k=t}^T \gamma^{(k-t)} R_{k+1} = \gamma G_{t+1} + R_{t+1}$$



RL Basics: Returns

- Mathematically, let $a = (a_0, \dots, a_T)$ then the **Return** for token t is

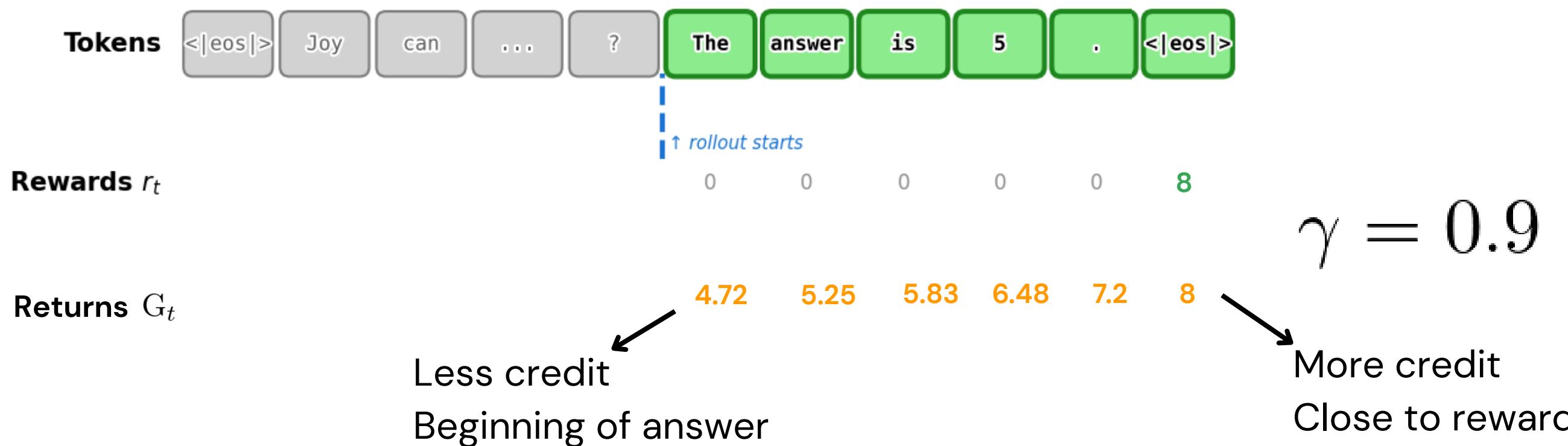
$$G_t = \sum_{k=t}^T \gamma^{(k-t)} R_{k+1} = \gamma G_{t+1} + R_{t+1}$$



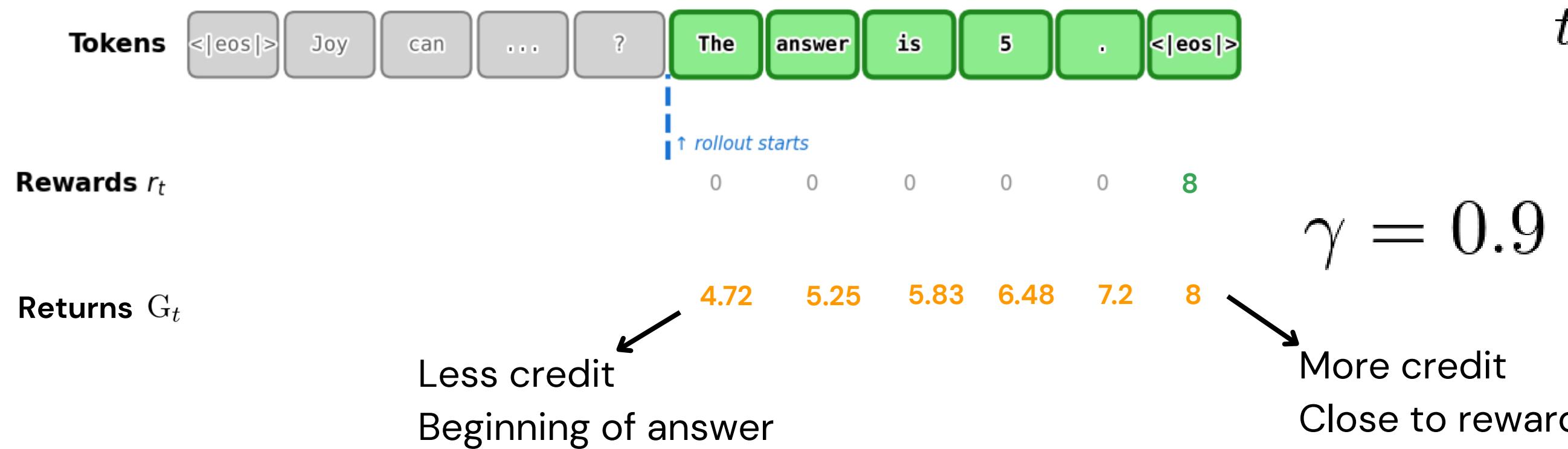
RL Basics: Returns

- Mathematically, let $a = (a_0, \dots, a_T)$ then the **Return** for token t is

$$G_t = \sum_{k=t}^T \gamma^{(k-t)} R_{k+1} = \gamma G_{t+1} + R_{t+1}$$



RL Basics: Returns



- **Returns** gives a way to assign credit to each individual tokens in the generation
- When training reasoning models , $\gamma \in [0.9, 1.0]$, often $\gamma = 1.0$
 - No discounts --> each token is assigned the same reward

RL Basics: Advantage with Returns

- We can now look at the **Full Vanilla Gradient Update**
 - **(Per-token) Advantage:** A_t (e.g. $A_t = G_t - b(a_t)$ but not limited to)
 - **Full policy update:** aggregates per-token Advantages weighted by their probability under the current policy

$$\nabla_{\theta} J(\theta) = \mathbb{E}_{a \sim \pi_{\theta}} \left[\sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | q) A_t \right]$$

Reasoning in AI - Second Part

Master 2 Natural Language Processing

- Lecture 5: Emergence of Reasoning Capabilities of LLMs through Post-Training
 - The Role of Supervised Fine Tuning and RL Intuitions in LLMs
 - Supervision Types in RL
 - RL Basics: Policy, Rewards, Baseline, Advantage
 - Policy Gradient Algorithms
 - Learning from Verifiable and Unverifiable Rewards

Policy Gradient Algorithms

- Variants in RL algorithms differ in:
 - **How they formulate the Per-Token Advantage** A_t
 - **How they weight the gradient updates** $\nabla_{\theta} J(\theta)$
- **General goal:** improve learning signal (flowing from the rewards) and training stability

Policy Gradient Algorithms

- We'll look into popular Policy Gradient Algorithms:
 - **REINFORCE**
 - **Proximal-Policy-Optimization (PPO)**
 - **Group-Relative Policy Optimization (GRPO)**

Policy Gradient Algorithms

- We follow this plan for each algorithm:

1. **Blueprint**
2. **Maths**
3. **Pseudo-Code**
4. **Comments**

Policy Gradient Algorithms

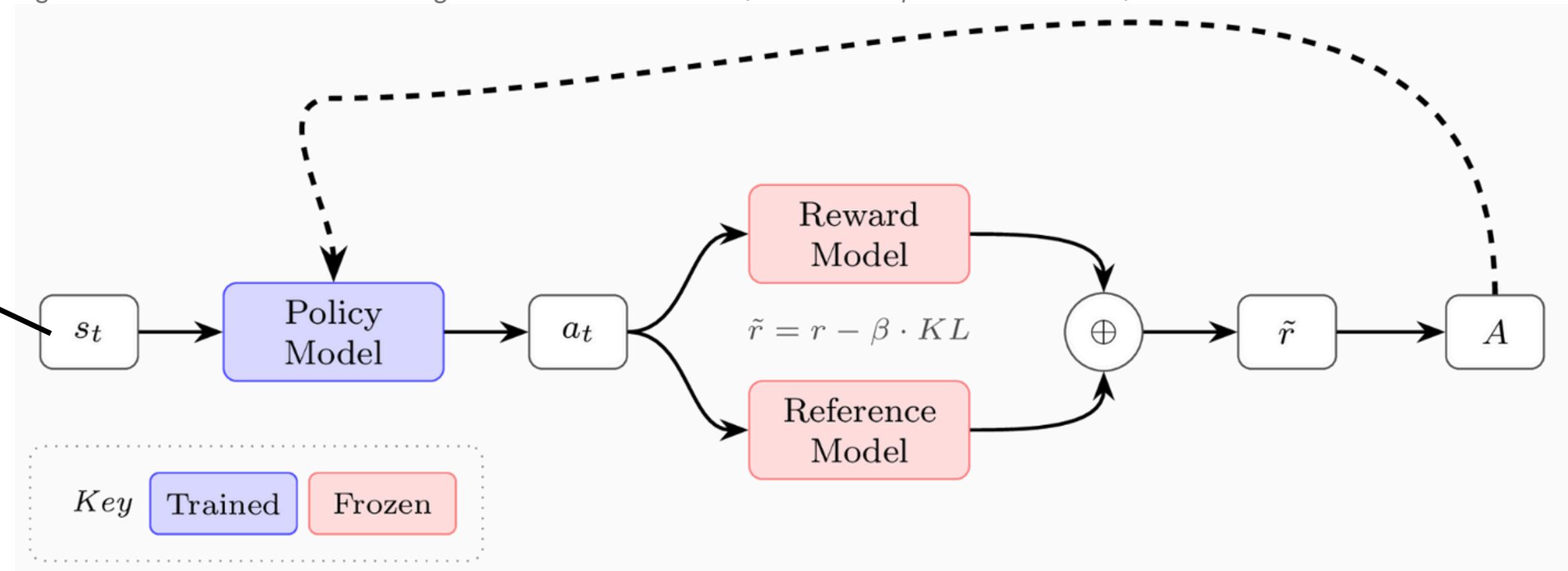
REINFORCE

Williams, R.J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach Learn* 8, 229–256 (1992). <https://doi.org/10.1007/BF00992696>

REINFORCE – Blueprint

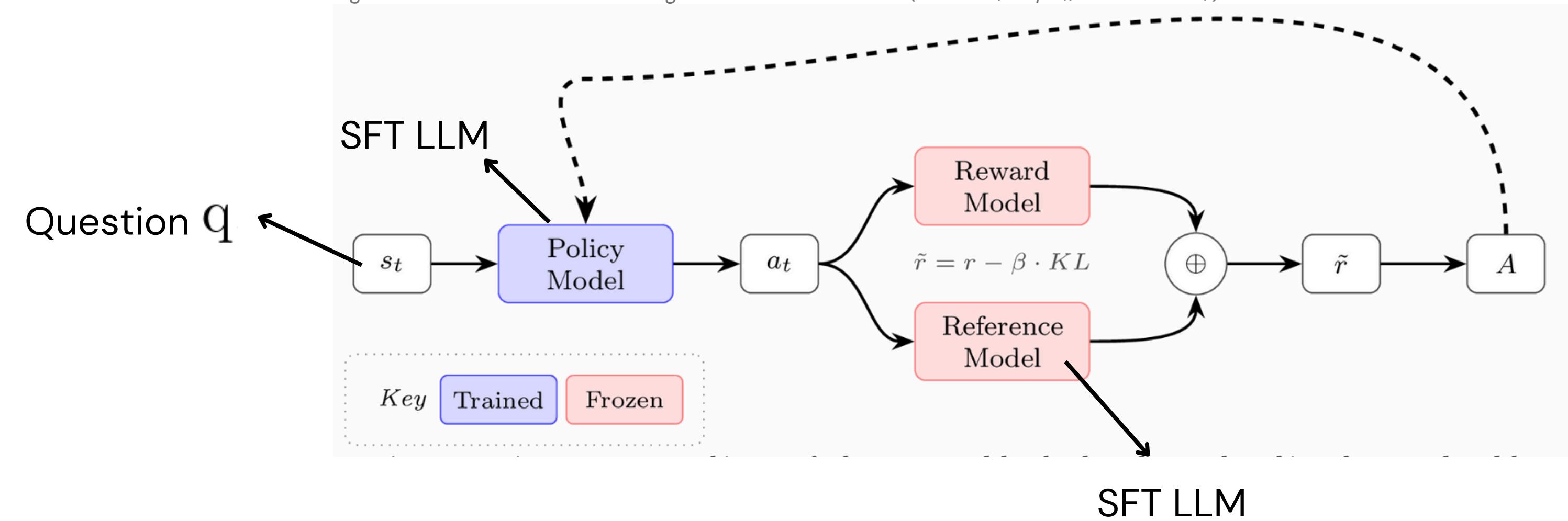
Figure from *Reinforcement Learning from Human Feedback* (Lambert, <https://rlhfbook.com/>)

Question q



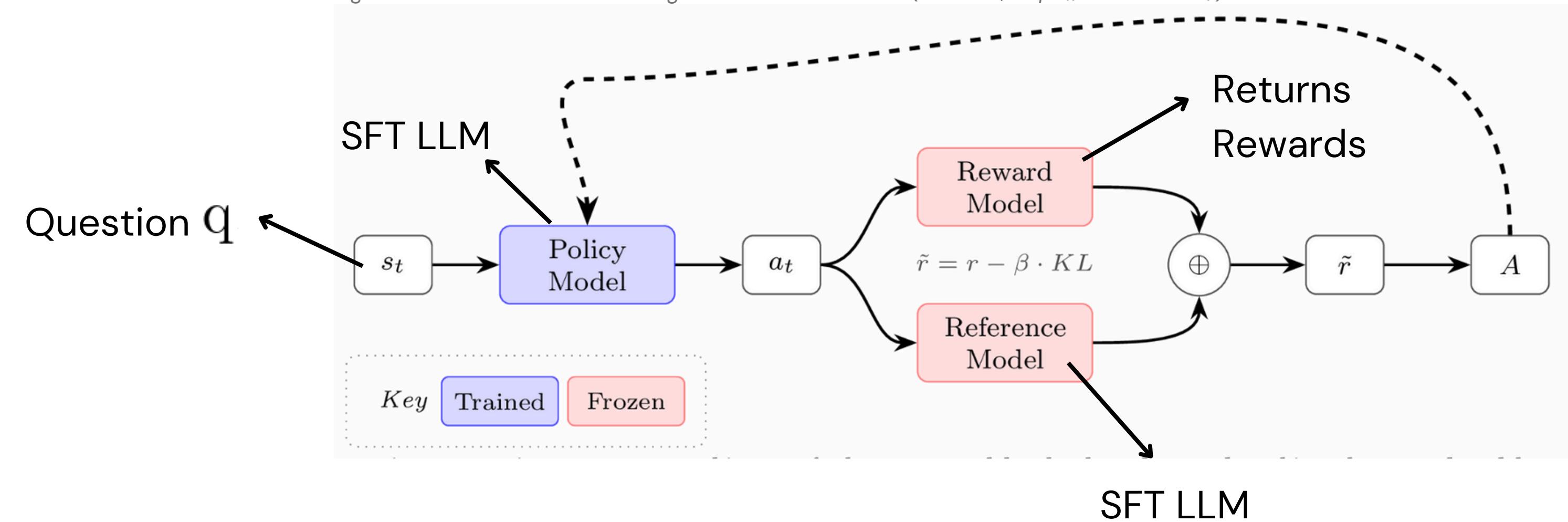
REINFORCE – Blueprint

Figure from *Reinforcement Learning from Human Feedback* (Lambert, <https://rlhfbook.com/>)



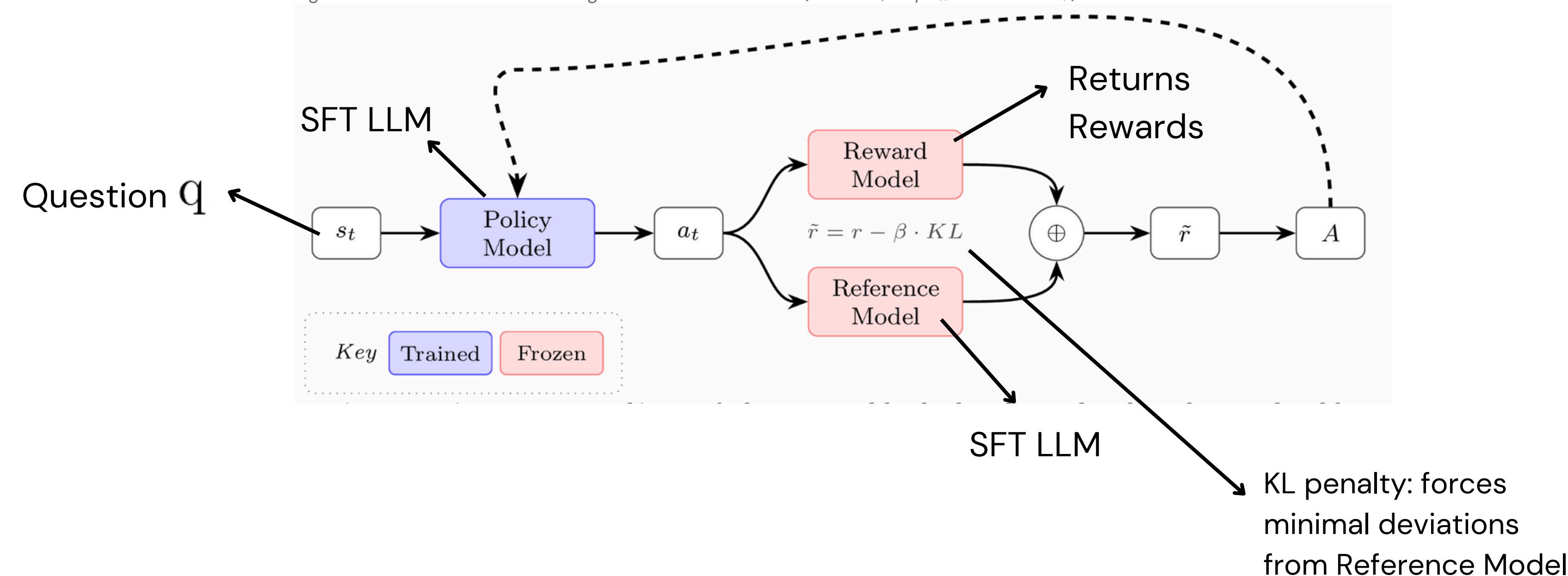
REINFORCE – Blueprint

Figure from *Reinforcement Learning from Human Feedback* (Lambert, <https://rlhfbook.com/>)



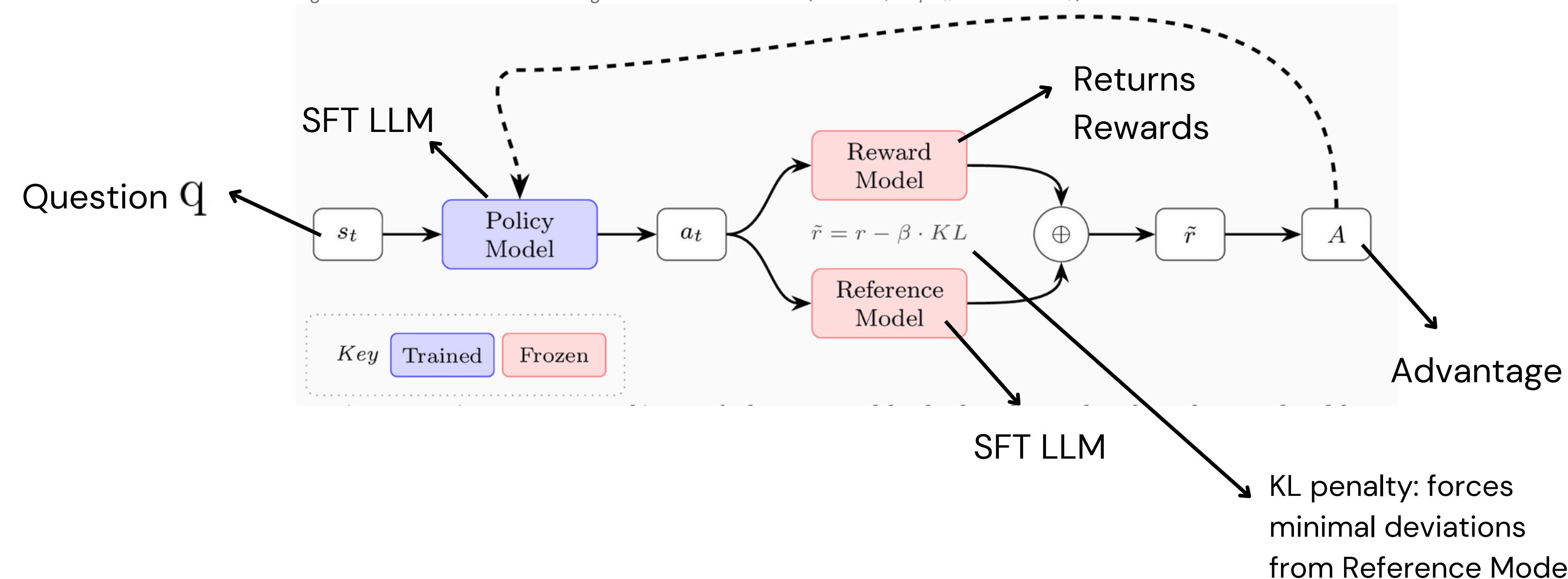
REINFORCE – Blueprint

Figure from *Reinforcement Learning from Human Feedback* (Lambert, <https://rlhfbook.com/>)



REINFORCE – Blueprint

Figure from *Reinforcement Learning from Human Feedback* (Lambert, <https://rlhfbook.com/>)



REINFORCE – Maths

- **Advantage:** $A_t = G_t - b(q)$ (Returns – Baseline)
- **Baseline:** $b(q) = \frac{1}{B} \sum_{j=1}^B R(q_j, a_j)$ (average reward in batch)
- **Gradient Update:** $\nabla_{\theta} J(\theta) = \mathbb{E}_{a \sim \pi_{\theta}} \left[\sum_{t=0}^T \nabla_{\theta} \log \pi_{\theta}(a_t | q) A_t \right]$
(vanilla full gradient update)

REINFORCE – Pseudo-Code

First

```
reference_model = LLM
policy_model = reference_model.clone()

for epoch in range(num_epochs):
    # 1. Sample a batch of prompts
    batch_prompts = random.sample(prompts, batch_size)

    # 2. Generate completions & collect trajectory data
    trajectories = [] # List of {prompt, completion, tokens, logprobs, reward}
    for prompt in batch_prompts:
        # Generate with policy model
        completion = policy_model.generate()

        # Gather log probabilities for both reference and policy models
        # This is  $P(a_t | q, a_{\{i < t\}})$ 
        logprobs = completion.logprobs
        ref_logprobs = reference_model(torch.cat((prompt, completion)).logprobs
        # Reference log probs of completion only
        ref_logprobs = ref_logprobs[:, prompt.size(1):]

        reward = REWARD_FN(prompt, completion)

        trajectories.append({
            'prompt': prompt,
            'completion': completion, # [T]
            'log_probs': logprobs, # [T]
            'ref_log_probs': ref_logprobs, # [T]
            'reward': reward # [T]
        })

    # 3. Compute baseline (variance reduction)
    rewards = [traj['reward'] for traj in trajectories]
    b = mean(rewards)
```

Second

```
# 4. Compute Returns  $G_t$ 
B,T = rewards.size()
returns = torch.zeros_like(rewards)
running = torch.zeros(B, device=rewards.device, dtype=rewards.dtype)

# done_mask: (B, T) 1.0 at terminal token (EOS or truncation if penalized), else 0.0
for t in reversed(range(T)):
    running = rewards[:, t] + gamma * (1.0 - done_mask[:, t]) * running
    returns[:, t] = running

# 4. Compute policy loss (REINFORCE objective)
total_loss = 0.0
per_token_loss_stacked = []
for traj in trajectories:
    log_probs = traj['log_probs'] # [T]
    R = traj['reward'] # [1]
    advantage = R - b # Scalar advantage

    # Policy gradient term:  $-\nabla \log \pi(a|s) * A(s,a)$ 
    per_token_loss = -log_probs * advantage # [T]
    per_token_loss_stacked.append(per_token_loss)

# [B, T]
ref_per_token_logs = torch.stack([traj['ref_log_probs'] for traj in trajectories])
per_token_logs = torch.stack([traj['log_probs'] for traj in trajectories])
per_token_loss_stacked = torch.stack(per_token_loss_stacked)

# KL regularization to reference model [B,T]
per_token_kl = (
    torch.exp(ref_per_token_logs - per_token_logs) - (ref_per_token_logs - per_token_logs) - 1
)

# Add KL penalty to token losses
per_token_loss = per_token_loss_stacked + beta * per_token_kl

total_loss = per_token_loss.sum(1) # [B] sum over tokens
total_loss = total_loss.mean() # [1] Average over batch

# 5. Update policy parameters
optimizer.zero_grad()
total_loss.backward()
optimizer.step()
```

REINFORCE – Comments

- REINFORCE is the default RL algorithm that applies the vanilla gradient update
- Never used in practice because other alternatives are just better in terms of stability



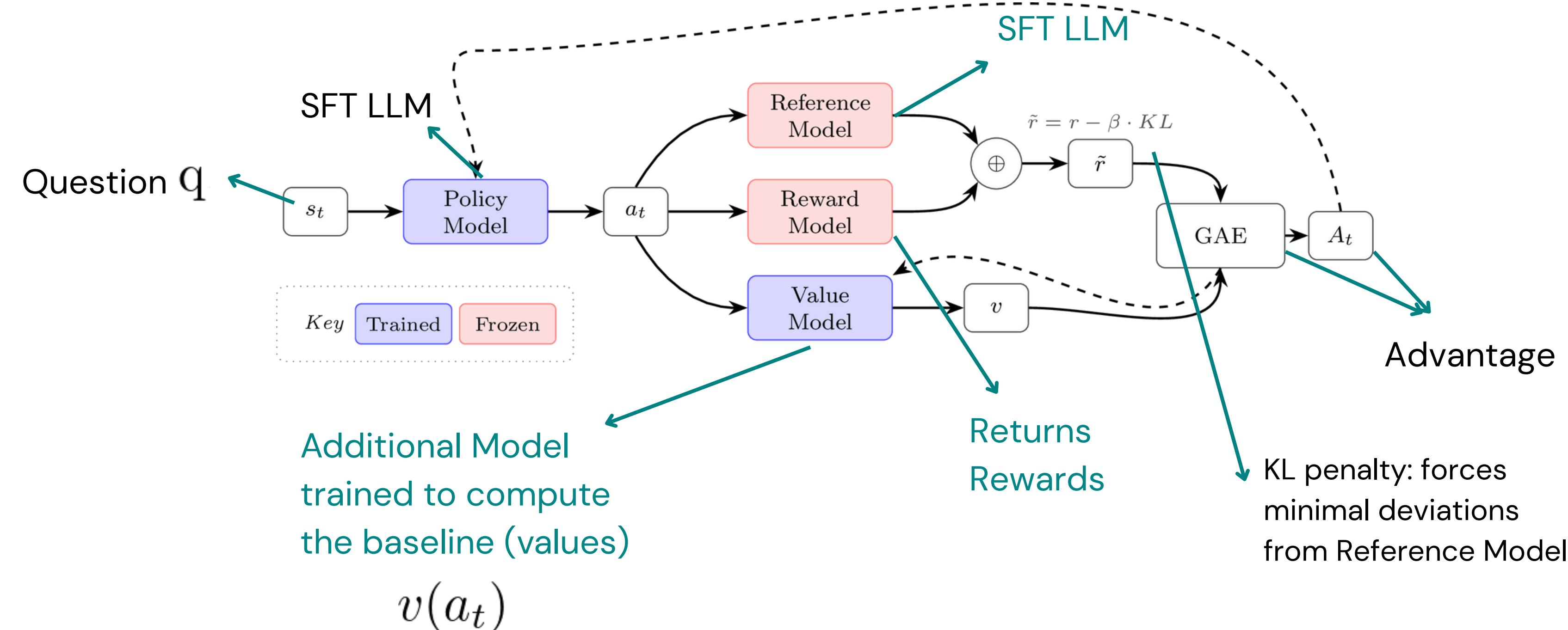
Policy Gradient Algorithms

Proximity-Policy-Optimization

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

PPO - Blueprint

Figure from *Reinforcement Learning from Human Feedback* (Lambert, <https://rlhfbook.com/>)



PPO – Maths

- **Baseline (value):** $v(a_t) = f_{\tilde{\theta}, \pi_\theta}(a_t, q)$
 - Separate network (typically policy model + Linear layers)
 - Trained to produce baseline values per-token
- **Advantage:** $A_t = G_t - v(a_t)$ (Returns – Predicted Baseline)
- **Objective:**

$$J(\theta) = \frac{1}{|a|} \sum_{t=0}^{|a|} \min \left(\frac{\pi_\theta(a_t|q)}{\pi_{\theta_{\text{old}}}(a_t|q)} A_t, \text{clip} \left(\frac{\pi_\theta(a_t|q)}{\pi_{\theta_{\text{old}}}(a_t|q)}, 1 - \varepsilon, 1 + \varepsilon \right) A_t \right)$$

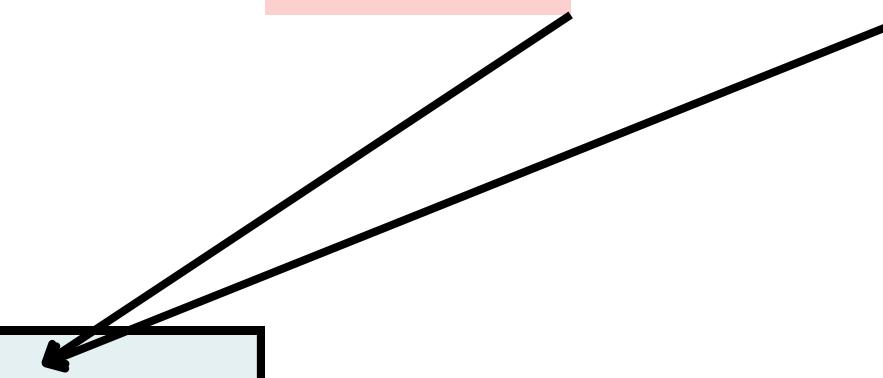
The PPO Objective

- **Objective:**

$$J(\theta) = \frac{1}{|a|} \sum_{t=0}^{|a|} \min \left(\frac{\pi_\theta(a_t|q)}{\pi_{\theta_{\text{old}}}(a_t|q)} A_t, \text{clip} \left(\frac{\pi_\theta(a_t|q)}{\pi_{\theta_{\text{old}}}(a_t|q)}, 1 - \varepsilon, 1 + \varepsilon \right) A_t \right)$$

The Importance Ratio

Measures how much probability of generating a_t has changed between the “old” and “current” policies



The PPO Objective: why it's clever

- **Objective:**

$$J(\theta) = \frac{1}{|a|} \sum_{t=0}^{|a|} \min \left(\frac{\pi_\theta(a_t|q)}{\pi_{\theta_{\text{old}}}(a_t|q)} A_t, \text{clip} \left(\frac{\pi_\theta(a_t|q)}{\pi_{\theta_{\text{old}}}(a_t|q)}, 1 - \varepsilon, 1 + \varepsilon \right) A_t \right)$$

The Importance Ratio

Measures how much probability of generating a_t has changed between the “old” and “current” policies

Typical PPO Iteration

```
answers = policy_model.generate(prompts)
old_logprobs = answers.logprobs()

for iteration in range(4) :
    # Get Importance Ratio
    new_logprobs = policy_model(answers).logprobs
    ratio = torch.exp(new_logprobs - old_logprobs)

    # calculate PPO objective
    loss = PPO_LOSS(ratio)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

The PPO Objective: why it's clever

- **Objective (to maximize):**

$$J(\theta) = \frac{1}{|a|} \sum_{t=0}^{|a|} \min \left(\frac{\pi_\theta(a_t|q)}{\pi_{\theta_{\text{old}}}(a_t|q)} A_t, \text{clip} \left(\frac{\pi_\theta(a_t|q)}{\pi_{\theta_{\text{old}}}(a_t|q)}, 1 - \varepsilon, 1 + \varepsilon \right) A_t \right)$$

```
# Typical PPO Iteration

answers = policy_model.generate(prompts)
old_logprobs = answers.logprobs()

for iteration in range(4) :
    # Get Importance Ratio
    new_logprobs = policy_model(answers).logprobs
    ratio = torch.exp(new_logprobs - old_logprobs) →

    # calculate PPO objective
    loss = PPO_LOSS(ratio)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Ratio starts at 1 for “iteration = 1”

Iteration > 1:

Ratio > 1 if gradient step increased likelihood of “useful tokens”

Ratio < 1 if gradient step decreased likelihood of “useful tokens”

The PPO Objective: why it's clever

- **Objective (to maximize):**

$$J(\theta) = \frac{1}{|a|} \sum_{t=0}^{|a|} \min \left(\frac{\pi_\theta(a_t|q)}{\pi_{\theta_{\text{old}}}(a_t|q)} A_t, \text{clip} \left(\frac{\pi_\theta(a_t|q)}{\pi_{\theta_{\text{old}}}(a_t|q)}, 1 - \varepsilon, 1 + \varepsilon \right) A_t \right)$$

```
# Typical PPO Iteration

answers = policy_model.generate(prompts)
old_logprobs = answers.logprobs()

for iteration in range(4) :
    # Get Importance Ratio
    new_logprobs = policy_model(answers).logprobs
    ratio = torch.exp(new_logprobs - old_logprobs) →

    # calculate PPO objective
    loss = PPO_LOSS(ratio)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Ratio > 1 if gradient step increased likelihood of “useful tokens”

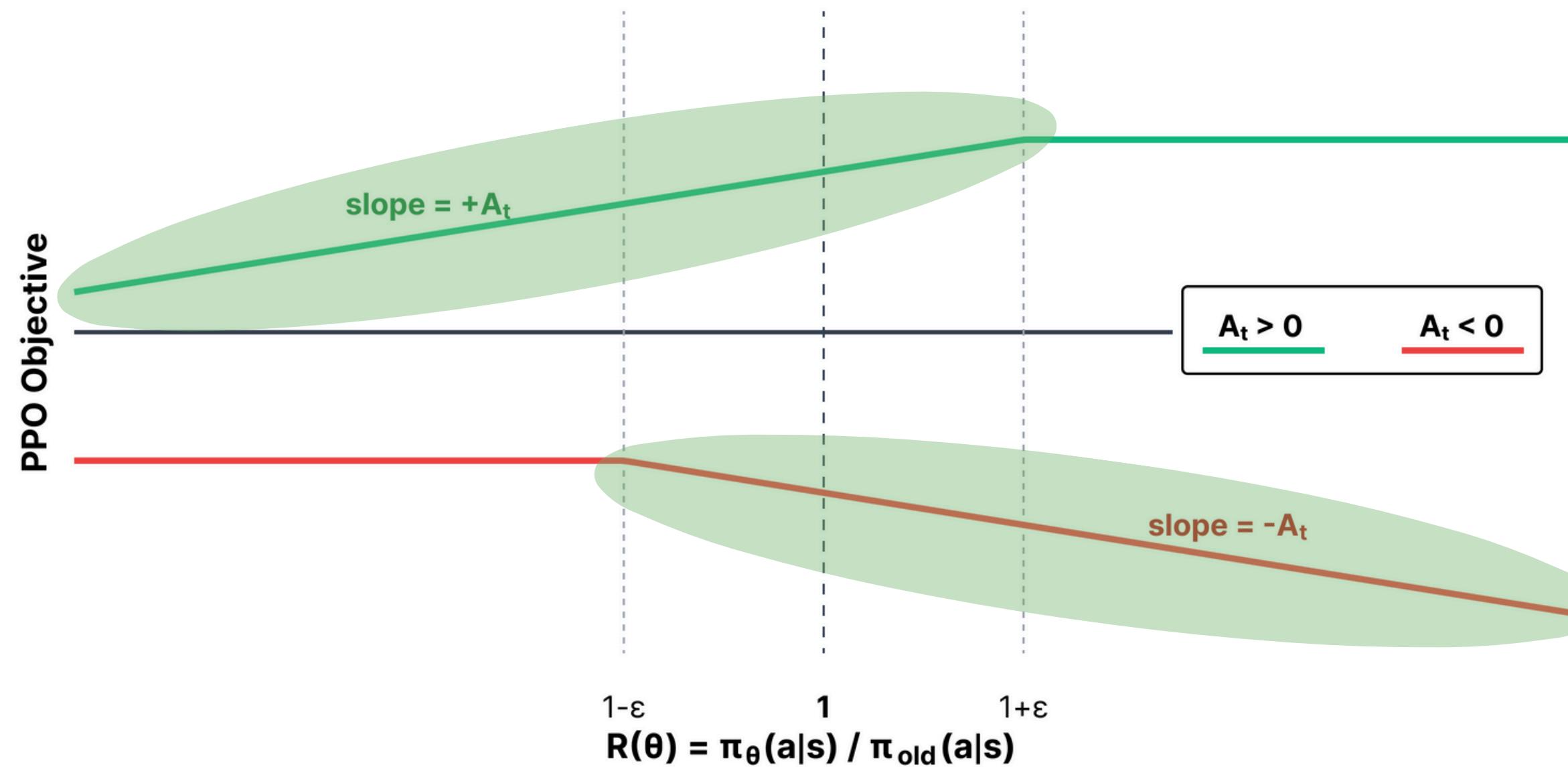
Ratio < 1 if gradient step decreased likelihood of “useful tokens”

Objective **Maximized** when:

- High Ratio for tokens with Positives Advantages
- Low Ratio with for tokens with Negative Advantages

The PPO Objective: why it's clever

- Gradient updates only in these situations:

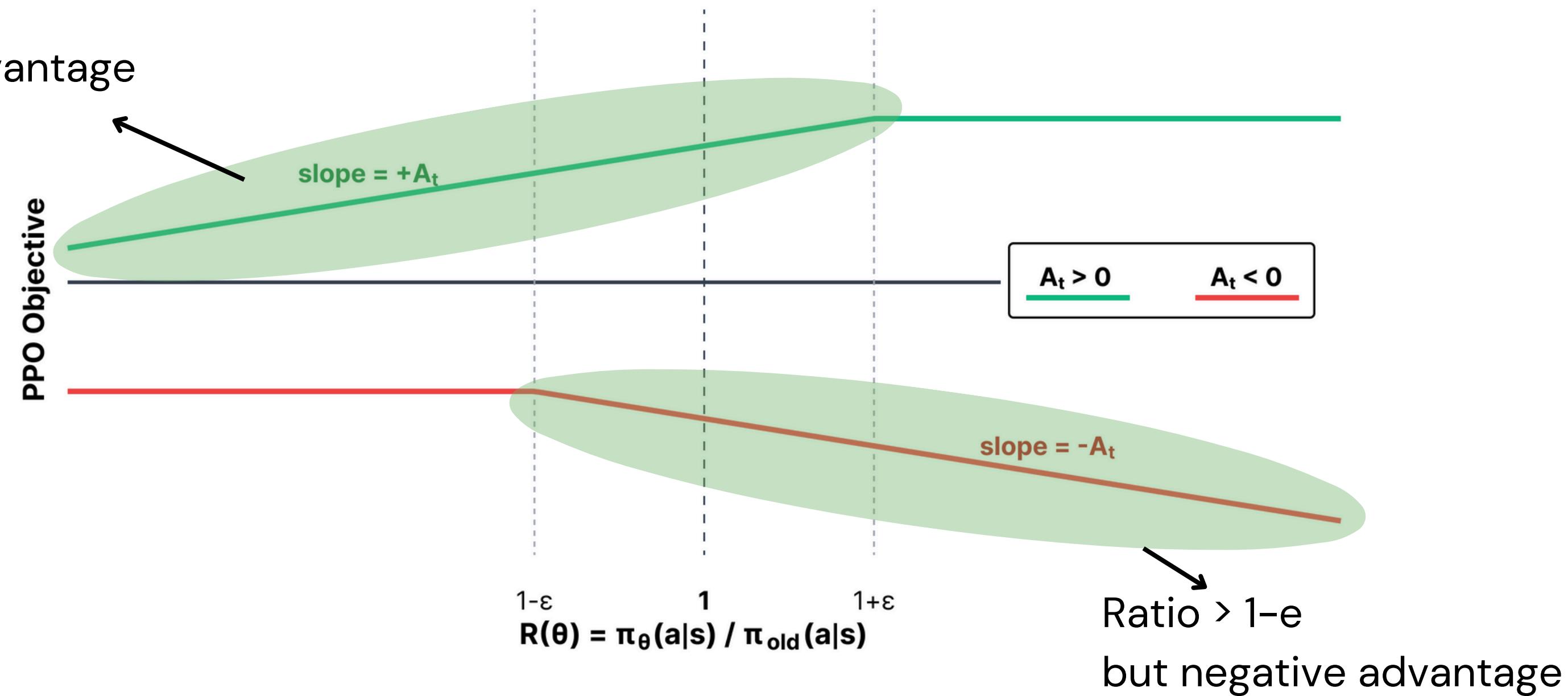


The PPO Objective: why it's clever

- Gradient updates only in these situations:

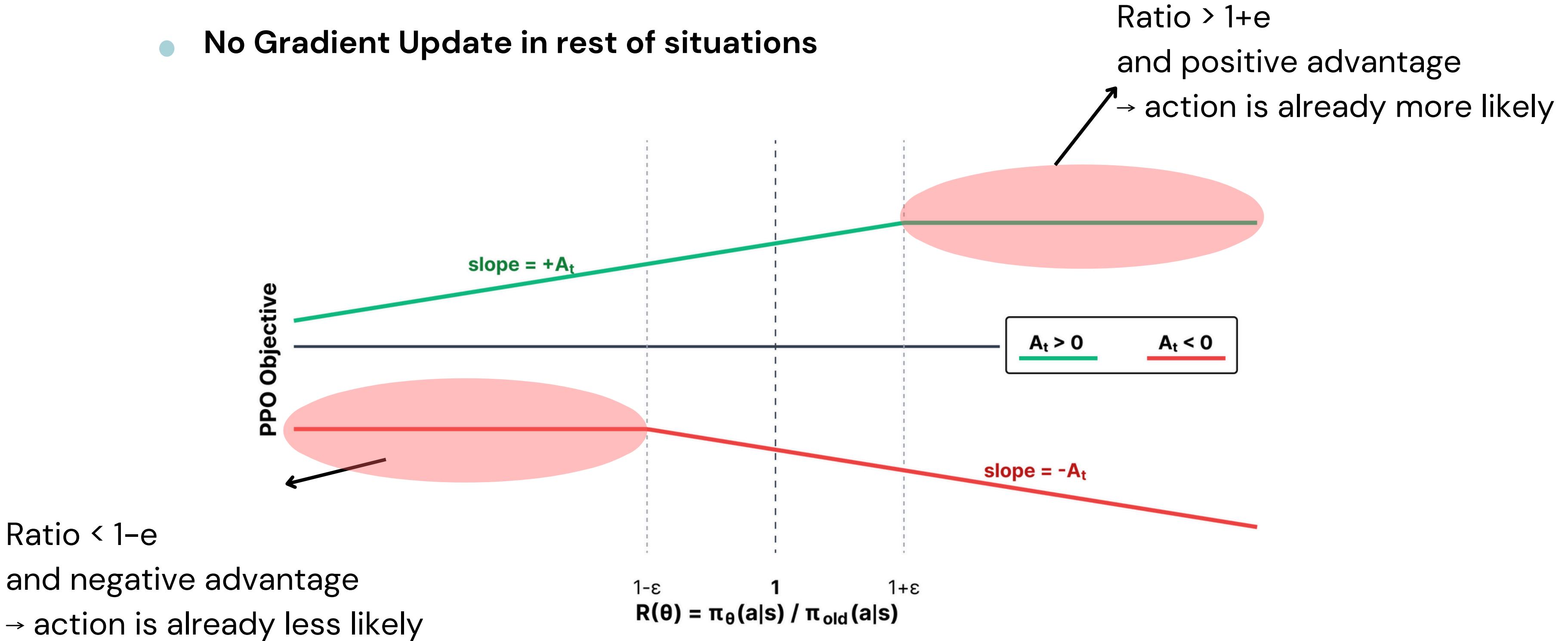
Ratio < 1+e

but positive advantage



The PPO Objective: why it's clever

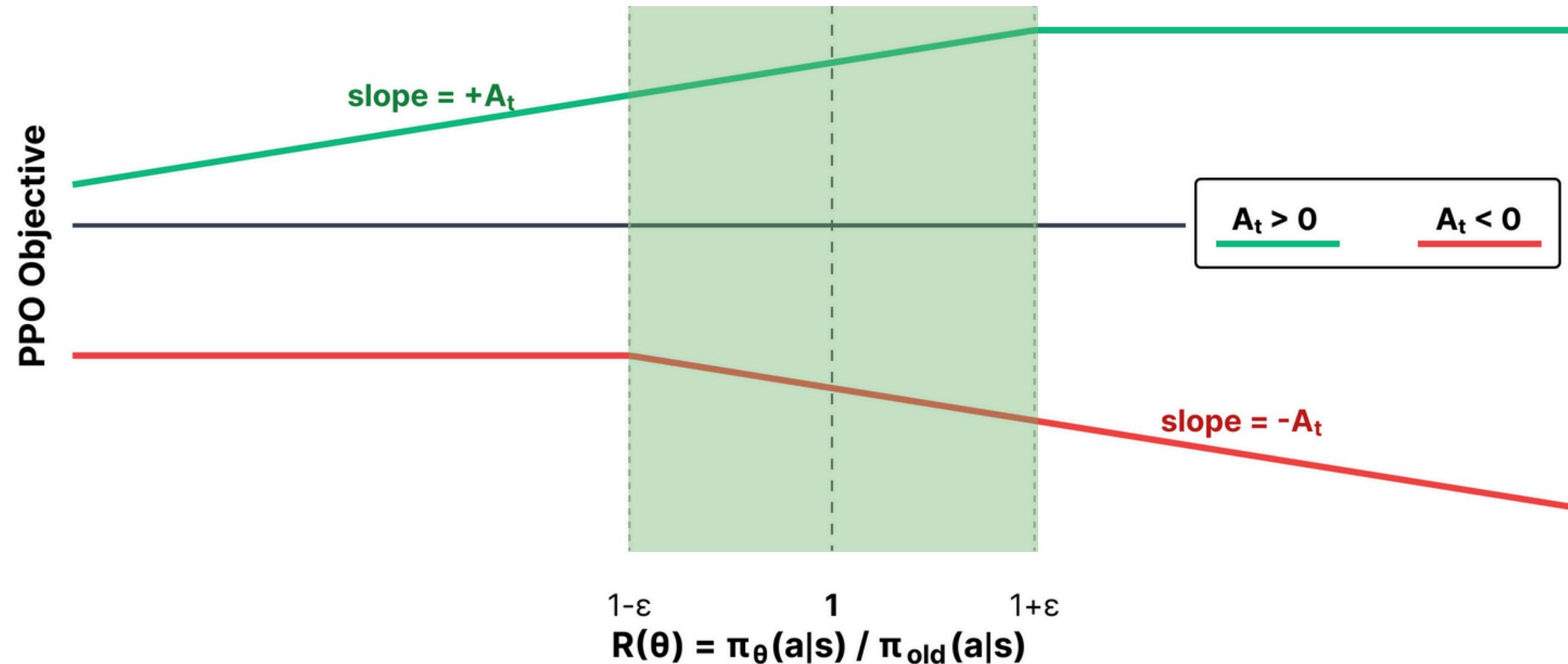
- No Gradient Update in rest of situations



The PPO Objective: why it's clever

- Standard Vanilla Gradient Update in Trust Region

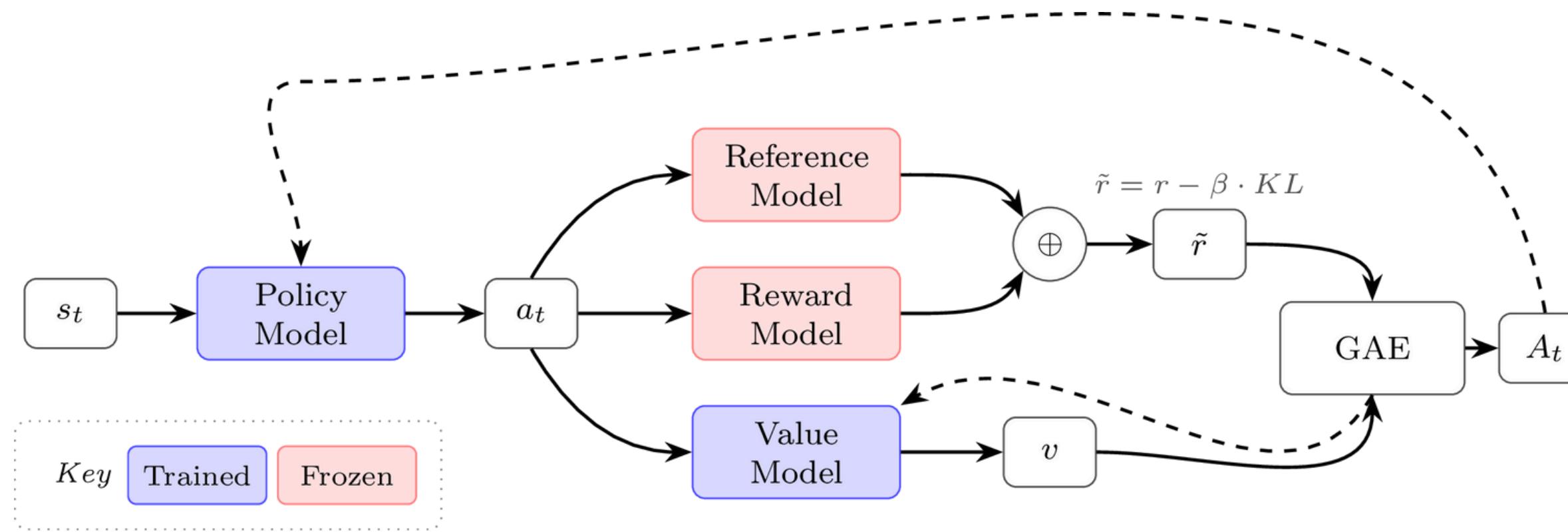
$$J(\theta) = \frac{1}{|a|} \sum_{t=0}^{|a|} \min \left(\frac{\pi_\theta(a_t|q)}{\pi_{\theta_{\text{old}}}(a_t|q)} A_t, \text{clip} \left(\frac{\pi_\theta(a_t|q)}{\pi_{\theta_{\text{old}}}(a_t|q)}, 1 - \varepsilon, 1 + \varepsilon \right) A_t \right)$$



The PPO Objective: why it's clever

- To summarize PPO uses a **modified objective** in order to :
 - Increase likelihood of generating useful tokens when it's necessary
 - Decrease likelihood of generating useless tokens when it's not necessary
- Thus, PPO encourages to explore new tokens (and thus new answers) more than standard Vanilla update

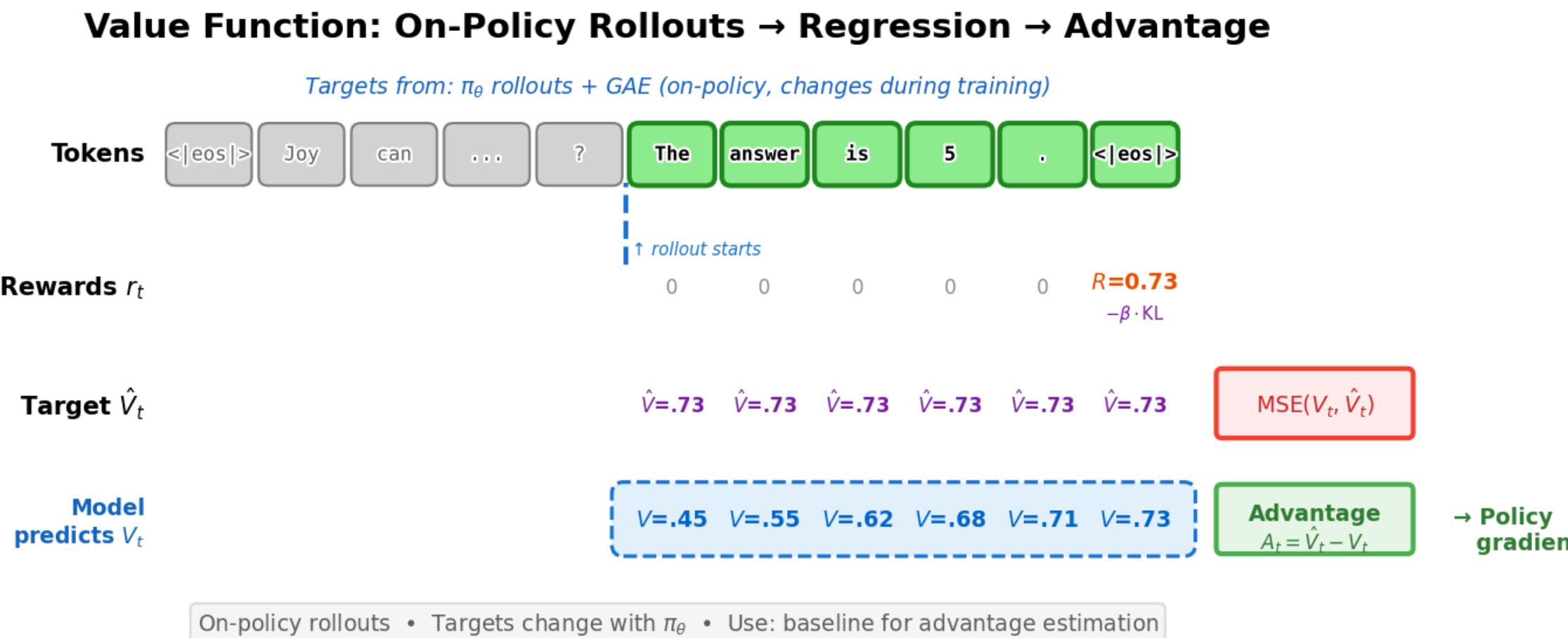
PPO - Value Function



- In PPO, each token is assigned a predicted **baseline value**
- A **separate** LLM instance is trained to predict those values

PPO - Value Function

- Recall that (per-token) baselines can be seen as the expected future return when using that token (with discounting)
- In PPO, the value network predicts the per-token returns, deducted from KL penalties



PPO - Pseudo-Code

First

```
reference_model = LLM
policy_model = reference_model.clone()
value_model = policy_model.clone()
# Head of the value model is just predicting a scalar value: the (discounted) per-token returns
value_model.value_head = torch.nn.Linear(policy_model.hidden_size, 1)

for epoch in range(num_epochs):
    # 1. Sample a batch of prompts
    batch_prompts = random.sample(prompts, batch_size)

    # 2. Generate completions & collect trajectory data
    trajectories = []
    # This step is often call "Rollout"
    for prompt in batch_prompts:
        with torch.no_grad():
            # Generate with policy model
            completion = policy_model.generate()

            # Gather log probabilities for both reference and policy models
            # This is P(a_t | q, a_{i<t})
            logprobs = completion.logprobs
            ref_logprobs = reference_model(torch.cat((prompt, completion)).logprobs
            # Reference log probs of completion only
            ref_logprobs = ref_logprobs[:, prompt.size(1):]

            reward = REWARD_FN(prompt, completion) # [T]
            per_token_kl = (
                torch.exp(ref_logprobs - logprobs) -
                (ref_logprobs - logprobs) - 1
            ) # [T]

            # !\ in PPO the KL penalty is directly added in the reward
            reward = reward - beta * per_token_kl

            values = value_model(completions)

            trajectories.append({
                'prompt': prompt,
                'completion': completion,      # [T]
                'log_probs': logprobs,         # [T]
                'ref_log_probs': ref_logprobs, # [T]
                'reward': reward,             # [T]
                'values': values,             # [T]
            })
```

Second

```
# 4. Compute Returns G_t and Advantages
rewards = torch.stack([traj['reward'] for traj in trajectories]) #[B,T]
values = torch.stack([traj['values'] for traj in trajectories]) #[B,T]

B,T = rewards.size()
returns = torch.zeros_like(rewards)
running = torch.zeros(B, device=rewards.device, dtype=rewards.dtype)

# done_mask: (B, T) 1.0 at terminal token (EOS or truncation if penalized), else 0.0
for t in reversed(range(T)):
    running = rewards[:, t] + gamma * (1.0 - done_mask[:, t]) * running
    returns[:, t] = running

# !\ Note how advantages are calculated from detached values
advantages = returns - values.detach()

# --- 3. MULTIPLE EPOCHS OVER MINIBATCHES ---
minibatch_size = B // 8 # build mini batches from rollouts for gradient updates
for epoch in range(4):
    idx = torch.randperm(batch_size)[:minibatch_size]

    # Slice minibatch data
    mb_completions = torch.stack([trajectories[i]['completions'] for i in idx]) # [B, T]
    mb_old_logprobs = torch.stack([trajectories[i]['log_probs'] for i in idx]) # [B, T]
    mb_ref_logprobs = torch.stack([trajectories[i]['ref_log_probs'] for i in idx]) # [B, T]
    mb_advantages = advantages[idx]
    mb_returns = returns[idx]
    mb_values = values[idx]

    # --- 4. POLICY LOSS (PPO-Clip) ---
    # Get current policy logprobs for same actions
    mb_logprobs = policy_model(mb_completions).logprobs
    # Compute ratio & clipped objective
    ratio = torch.exp(mb_new_logprobs - mb_old_logprobs)
    surr1 = ratio * mb_advantages
    surr2 = torch.clamp(ratio, 1 - config.clip_range, 1 + config.clip_range) * mb_advantages
    policy_loss = -(torch.min(surr1, surr2) * mask).sum() / mask.sum()

    # Value function loss: predict returns
    vf_loss = 0.5 * ((mb_returns - mb_values) ** 2) # Shape: (B*G, L)

    # Total loss is Policy Loss + Value Loss
    loss = policy_loss + vf_coef * vf_loss

    # Update parameters
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

PPO – Comments

- PPO is a fundamental Policy Gradient Update algorithms that **led to Deep RL success**, such as OpenAI's Five that learnt how to play DOTA 2 [1]
- Standard PPO implementations use **Generalized Advantage Estimation** (GAE) to compute the Advantage. We might look at it in the Lab
- It is still widely applied today, and often compared to GRPO in the literature
- Its **main limitation** is the necessity of a **separate value model**, which significantly increases the amount of necessary compute (imagine needing to run 3 instances of GPT-5.2 locally)

[1] Berner, Christopher, et al. "Dota 2 with large scale deep reinforcement learning." arXiv preprint arXiv:1912.06680 (2019).



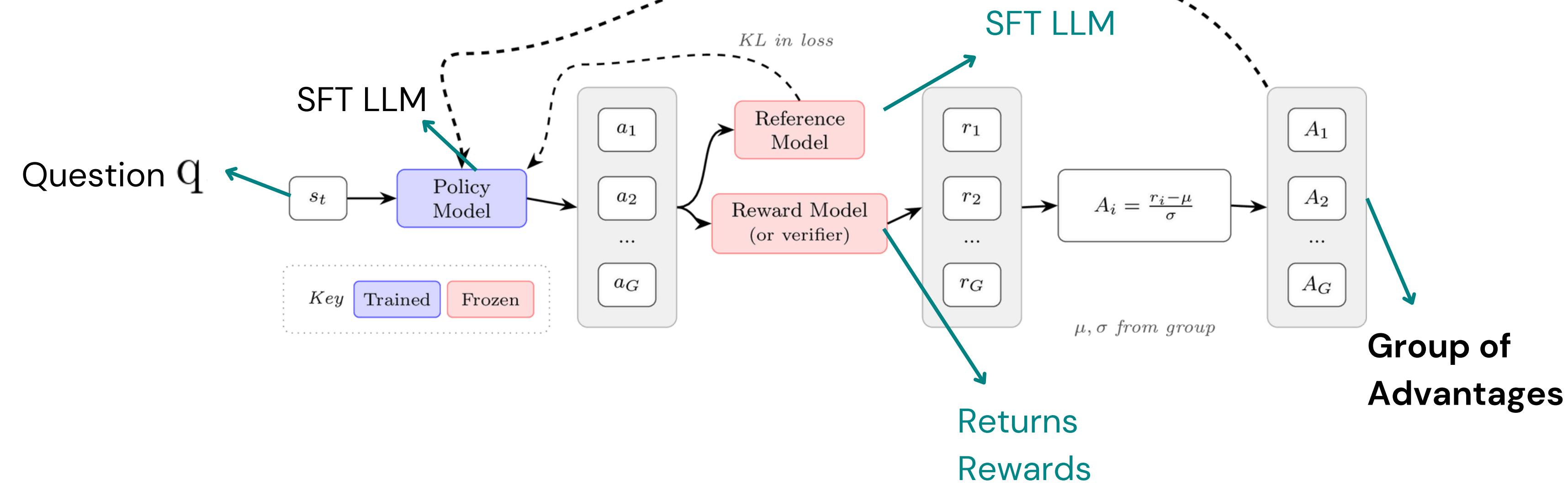
Policy Gradient Algorithms

Group-Relative-Policy-Optimization

Shao, Zhihong, et al. "Deepseekmath: Pushing the limits of mathematical reasoning in open language models." arXiv preprint arXiv:2402.03300 (2024).

GRPO – Blueprint

Figure from Reinforcement Learning from Human Feedback (Lambert, <https://rlhfbook.com/>)



KL penalty: forces
minimal deviations
from Reference Model

GRPO – Maths

- **No Value Model and no Baselines**
- **Multiple answers** (a_1, \dots, a_G) **for same prompt** (with sampling parameters)

- **Advantage:** $A_i = \frac{r_i - \text{mean}(r_1, r_2, \dots, r_G)}{\text{std}(r_1, r_2, \dots, r_G)}$

Natural regularization by *whitening* on the set of different answers for same prompt

- **Objective:**

$$J(\theta) = \frac{1}{G} \sum_{i=1}^G \frac{1}{|a_i|} \sum_{t=1}^{|a_i|} \left(\min \left(\frac{\pi_\theta(a_{i,t}|s_i)}{\pi_{\theta_{\text{old}}}(a_{i,t}|s_i)} A_{i,t}, \text{clip} \left(\frac{\pi_\theta(a_{i,t}|s_i)}{\pi_{\theta_{\text{old}}}(a_{i,t}|s_i)}, 1 - \varepsilon, 1 + \varepsilon \right) A_{i,t} \right) - \beta \mathcal{D}_{\text{KL}}(\pi_\theta(\cdot|s_i) || \pi_{\text{ref}}(\cdot|s_i)) \right)$$

GRPO – Maths

- **Objective:**

$$J(\theta) = \frac{1}{G} \sum_{i=1}^G \frac{1}{|a_i|} \sum_{t=1}^{|a_i|} \left(\min \left(\frac{\pi_\theta(a_{i,t}|s_i)}{\pi_{\theta_{\text{old}}}(a_{i,t}|s_i)} A_{i,t}, \text{clip} \left(\frac{\pi_\theta(a_{i,t}|s_i)}{\pi_{\theta_{\text{old}}}(a_{i,t}|s_i)}, 1 - \varepsilon, 1 + \varepsilon \right) A_{i,t} \right) - \beta \mathcal{D}_{\text{KL}}(\pi_\theta(\cdot|s_i) || \pi_{\text{ref}}(\cdot|s_i)) \right)$$

PPO Objective

GRPO – Maths

- **Objective:**

$$J(\theta) = \frac{1}{G} \sum_{i=1}^G \frac{1}{|a_i|} \sum_{t=1}^{|a_i|} \left(\min \left(\frac{\pi_\theta(a_{i,t}|s_i)}{\pi_{\theta_{\text{old}}}(a_{i,t}|s_i)} A_{i,t}, \text{clip} \left(\frac{\pi_\theta(a_{i,t}|s_i)}{\pi_{\theta_{\text{old}}}(a_{i,t}|s_i)}, 1 - \varepsilon, 1 + \varepsilon \right) A_{i,t} \right) - \beta \mathcal{D}_{\text{KL}}(\pi_\theta(\cdot|s_i) || \pi_{\text{ref}}(\cdot|s_i)) \right)$$

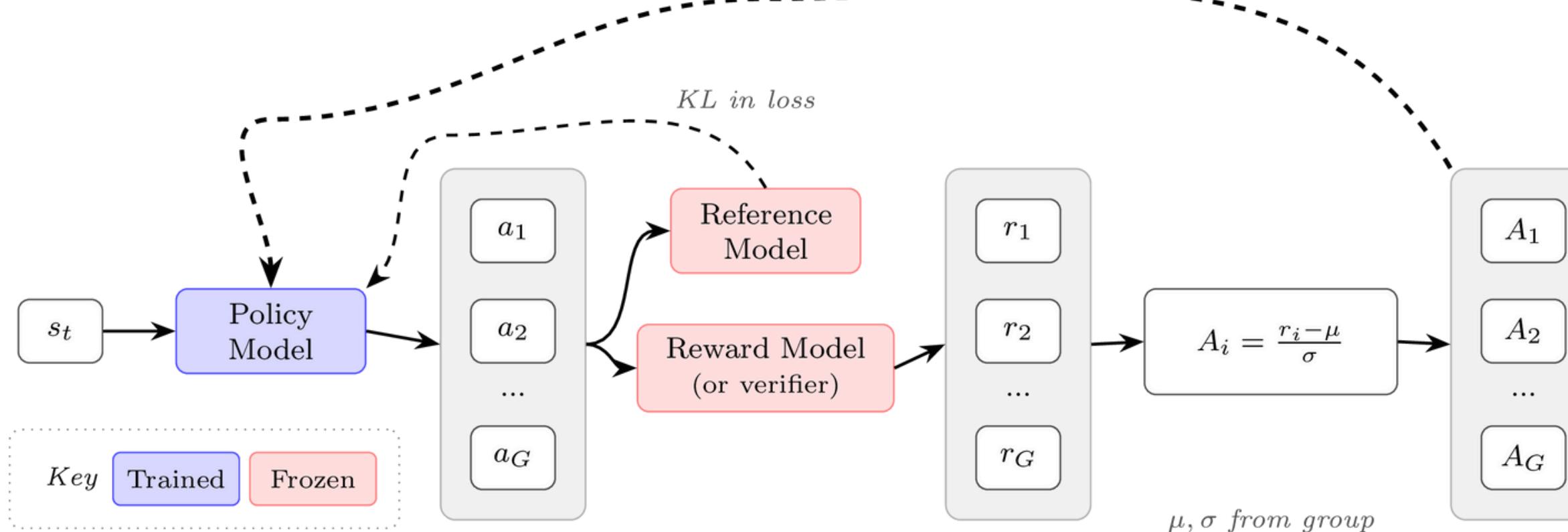
PPO Objective

KL Regularization in Loss

Different from PPO that applies regularization in per-token Rewards

GRPO – Intuition

Figure from *Reinforcement Learning from Human Feedback* (Lambert, <https://rlhfbook.com/>)



- Multiple answers per prompt allows the model to see “bad” and “good” answers in one gradient update
- Thus each weight update pushed the policy model to generate more “good” answers than “bad” ones

GRPO - Pseudo-Code

First

```
reference_model = LLM
policy_model = reference_model.clone()

for epoch in range(num_epochs):
    # 1. Sample a batch of prompts
    batch_prompts = random.sample(prompts, batch_size)

    with torch.no_grad() :
        # 2. Generate completions & collect trajectory data
        trajectories = []
        # This step is often call "Rollout"
        for prompt in batch_prompts:
            with torch.no_grad():
                # Generate with policy model
                completion = policy_model.generate(num_generations=G) # [B, G, T]

                # Gather log probabilities for both reference and policy models
                # This is P(a_t | q, a_{i<t})
                logprobs = completion.logprobs
                ref_logprobs = reference_model(torch.cat((prompt, completion)).logprobs # [B, G, T]
                # Reference log probs of completion only
                ref_logprobs = ref_logprobs[:, :, prompt.size(1):] # [B, G, T]

                reward = REWARD_FN(prompt, completion) # [G]

                trajectories.append({
                    'prompt': prompt,
                    'completion': completion,      # [G, T]
                    'log_probs': logprobs,         # [G, T]
                    'ref_log_probs': ref_logprobs, # [G, T]
                    'reward': reward,             # [G]
                })
```

Second

```
# 3. Compute Advantages
rewards = torch.stack([traj['reward'] for traj in trajectories]) #[B,G]
B,G = rewards.size()

rewards_mean = rewards.mean(1) # [B]
rewards_std = rewards.std(1) # [B]

rewards_std = rewards_mean.repeat_interleave(G, dim=0) # [B*G]
rewards_mean = rewards_std.repeat_interleave(G, dim=0) # [B*G]

advantages = (rewards.view(-1) - rewards_mean.unsqueeze(1)) / (rewards_std.unsqueeze(1) + 1e-6) # [B*G]

# Get reference Log Probs for KL
ref_per_token_logps =torch.stack([traj['ref_log_probs'] for traj in trajectories]) #[B,G,T]
ref_per_token_logps = ref_per_token_logps.view(-1, T)

old_logprobs = logprobs
# --- 5. OFF-Policy steps ---
for iteration in range(4) :

    # Get current policy logprobs for same actions
    logprobs = policy_model(torch.cat((prompt, completion))).logprobs #[B, G, T]

    # Get KL
    per_token_logps = logprobs.view(-1, T) # [B*G, T]

    per_token_kl = (
        torch.exp(ref_per_token_logps - per_token_logps) - (ref_per_token_logps - per_token_logps) - 1
    )

    # Compute ratio & clipped objective
    ratio = torch.exp(logprobs.view(-1, T) - old_logprobs.view(-1, T)) # [B*G, T]
    # All tokens get the same Advantage
    surr1 = ratio * advantages
    surr2 = torch.clamp(ratio, 1 - config.clip_range, 1 + config.clip_range) * advantages
    per_token_loss = -torch.min(surr1, surr2)

    # Incorporate KL in Loss
    loss = per_token_loss + beta * per_token_kl

    # Update parameters
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

GRPO – Comments

- GRPO is a very recent contribution (2024) from DeepSeek [1]
- It was specifically design for LLMs:
 - It does't use a Value Model, which saves compute ressources
 - Generating multiple answers to the same prompt makes sense in LLMs, but less sense when considering other applications (robotics)
- The core innovation thus lies in the **Advantage estimation**, relying on a really simple heuristic

[1] Shao, Zhihong, et al. "Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024." URL <https://arxiv.org/abs/2402.03300> 2.3 (2024): 5.



Policy Gradient Algorithms

Recap

Shao, Zhihong, et al. "Deepseekmath: Pushing the limits of mathematical reasoning in open language models." arXiv preprint arXiv:2402.03300 (2024).

Policy Gradient Algorithms – Recap

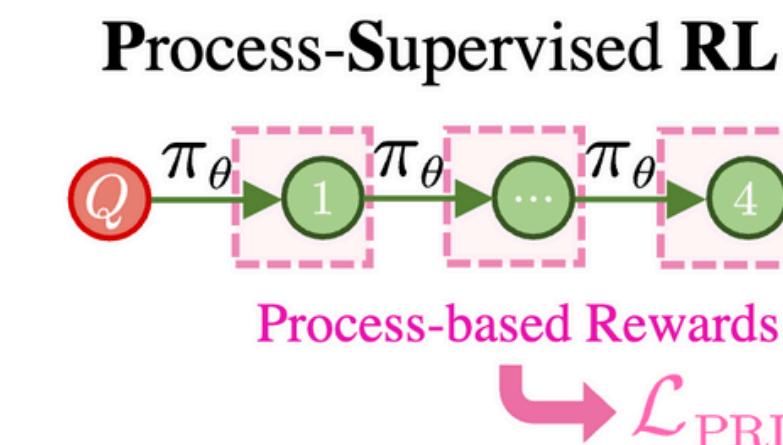
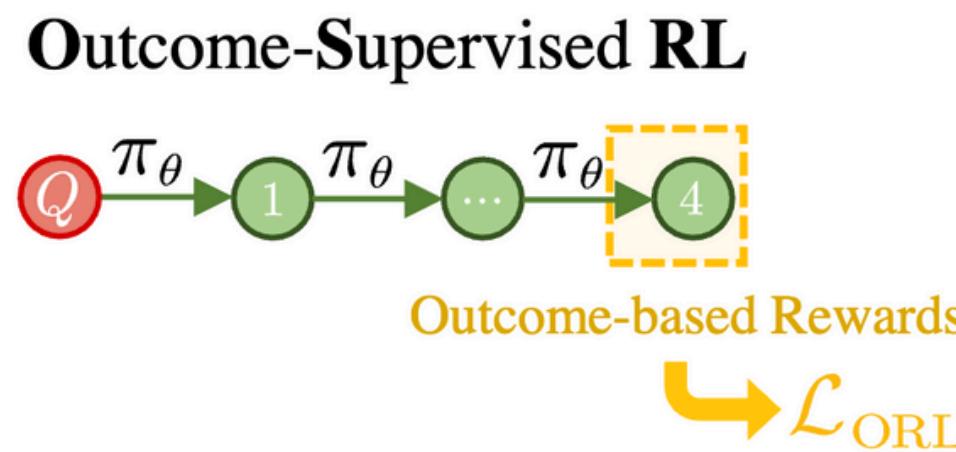
Method	Type	Reward Model	Value Model	Loss
REINFORCE	On-Policy	Yes	No	$-\frac{1}{T} \sum_{t=1}^T \log \pi_\theta(a_t s_t) (G_t - b(s_t))$
PPO	On-Policy	Yes	Yes	$-\frac{1}{T} \sum_{t=1}^T \min(\rho_t A_t, \text{clip}(\rho_t, 1 - \varepsilon, 1 + \varepsilon) A_t)$ $\rho_t = \frac{\pi_\theta(a_t s_t)}{\pi_{\theta_{\text{old}}}(a_t s_t)}$
GRPO	On-Policy	Yes	No	$-\frac{1}{G} \sum_{i=1}^G \min(\rho_i A_i, \text{clip}(\rho_i, 1 - \varepsilon, 1 + \varepsilon) A_i)$ $\rho_i = \frac{\pi_\theta(a_i s)}{\pi_{\theta_{\text{old}}}(a_i s)}, A_i = \frac{r_i - \text{mean}(r_{1:G})}{\text{std}(r_{1:G})}$

Policy Gradient Algorithms – Recap

- PPO and GRPO are central algorithms used to train **Large Reasoning Models**
- **Core innovation** lies in clever use of RL for **exploring** better “thoughts” that lead to increased performance in reasoning tasks
- Other algorithms are quite important such as **Direct-Preference-Optimization (DPO)**
 - Main algorithm used to align LLMs to human preferences
 - While we didn't cover it, this lecture should give you the tools to understand what happens in DPO

Policy Gradient Algorithms – Recap

- All the theory, pseudo-code etc.. we saw is directly applicable to the **Outcome-based RL**
- Application to **Process-based RL** is possible but requires slight modifications in terms of **Loss definition** and **implementation**



Reasoning in AI - Second Part

Master 2 Natural Language Processing

- Lecture 5: Emergence of Reasoning Capabilities of LLMs through Post-Training
 - The Role of Supervised Fine Tuning and RL Intuitions in LLMs
 - Supervision Types in RL
 - RL Basics: Policy, Rewards, Baseline, Advantage
 - Policy Gradient Algorithms
 - Learning from Verifiable and Unverifiable Rewards



Reinforcement Learning with Verifiable Rewards

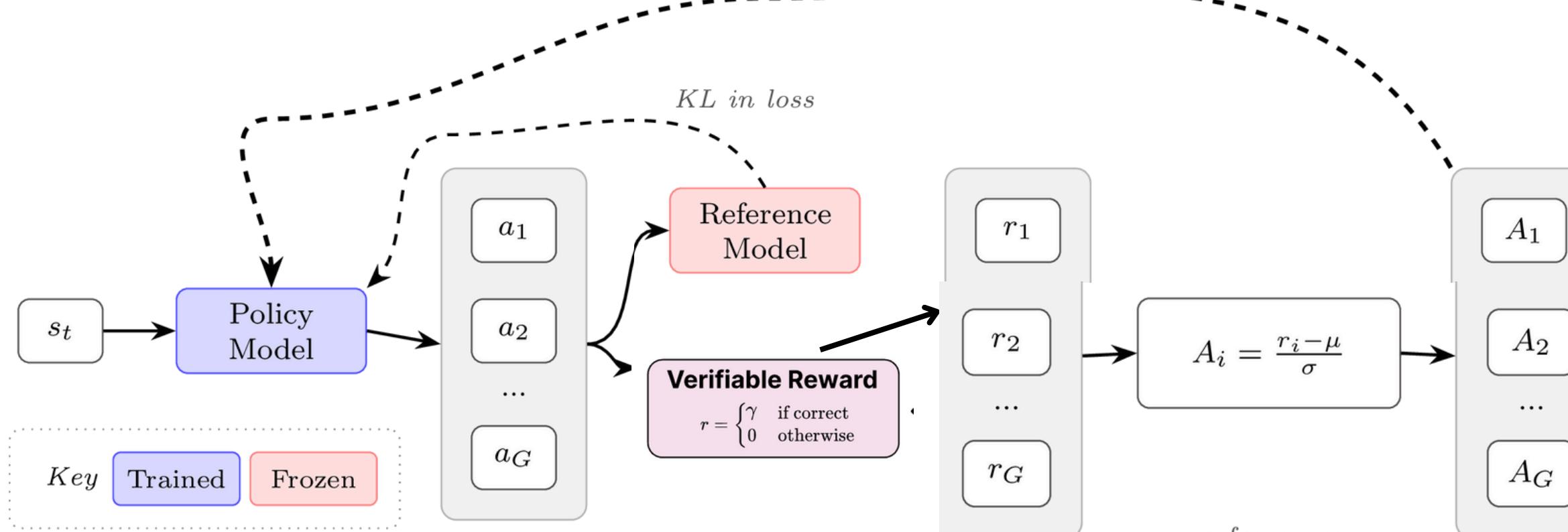
Lambert, Nathan, et al. "Tulu 3: Pushing frontiers in open language model post-training." arXiv preprint arXiv:2411.15124 (2024).

What is Verifiability

- Verifiability means we can **externally verify** if a generated answer matches a ground-truth answer

	Maths	Code	Instructions
Answer type	Equation	Functional Code Snippet	String
Generation Constraints	Must be the same as Answer	Must pass Unit-tests	Should satisfy many constraints
External Tool	Solver	Unit-Tests	String Match

How it is applied



- Recall that all Policy Gradient Algorithms use a **Reward Model**
- Learning from **Verifiable Rewards**, replaces the **Reward model** with a **verifiability step**



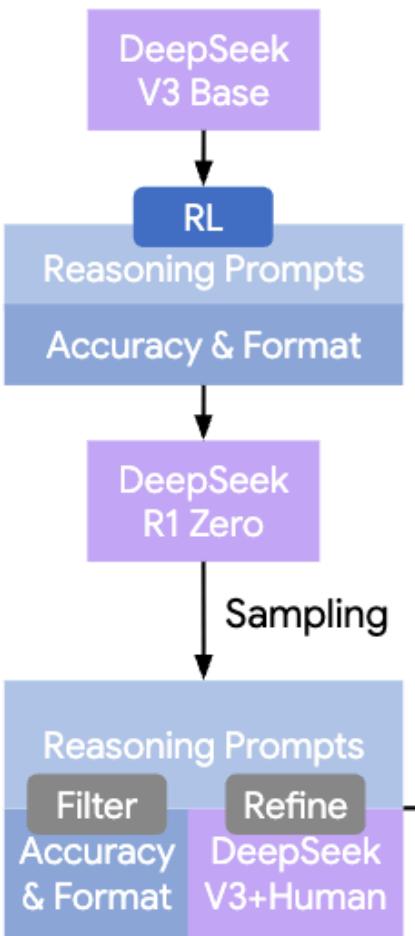
Reinforcement Learning with Verifiable Rewards

Case Study: DeepSeek-R1

Guo, Daya, et al. "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning." arXiv preprint arXiv:2501.12948 (2025).

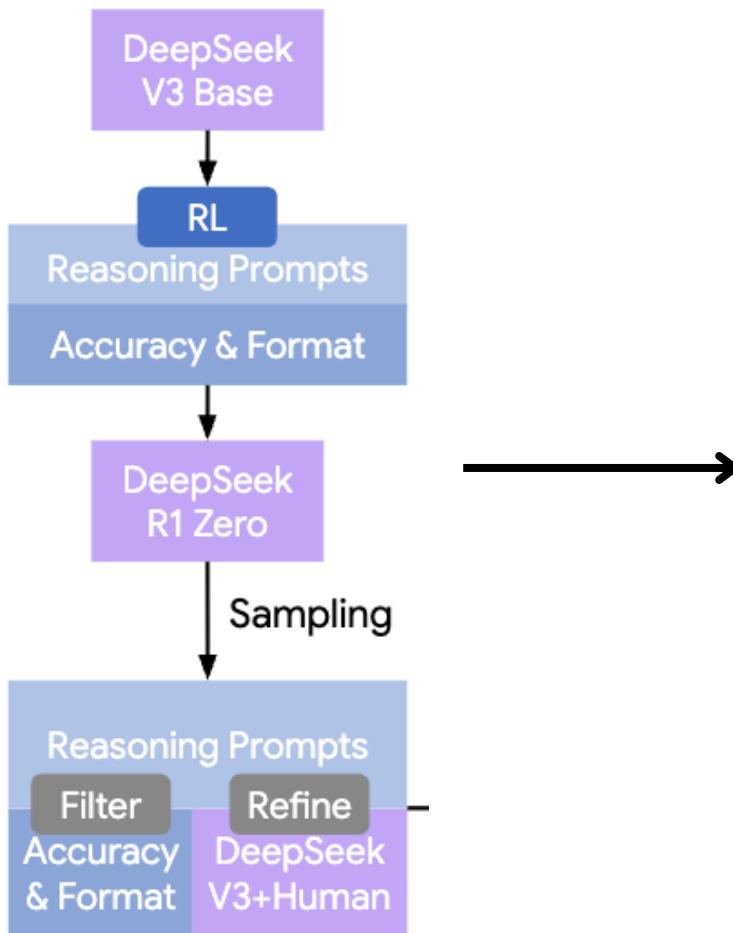
DeepSeek-R1

- **Stage 1: R1-Zero**



DeepSeek-R1

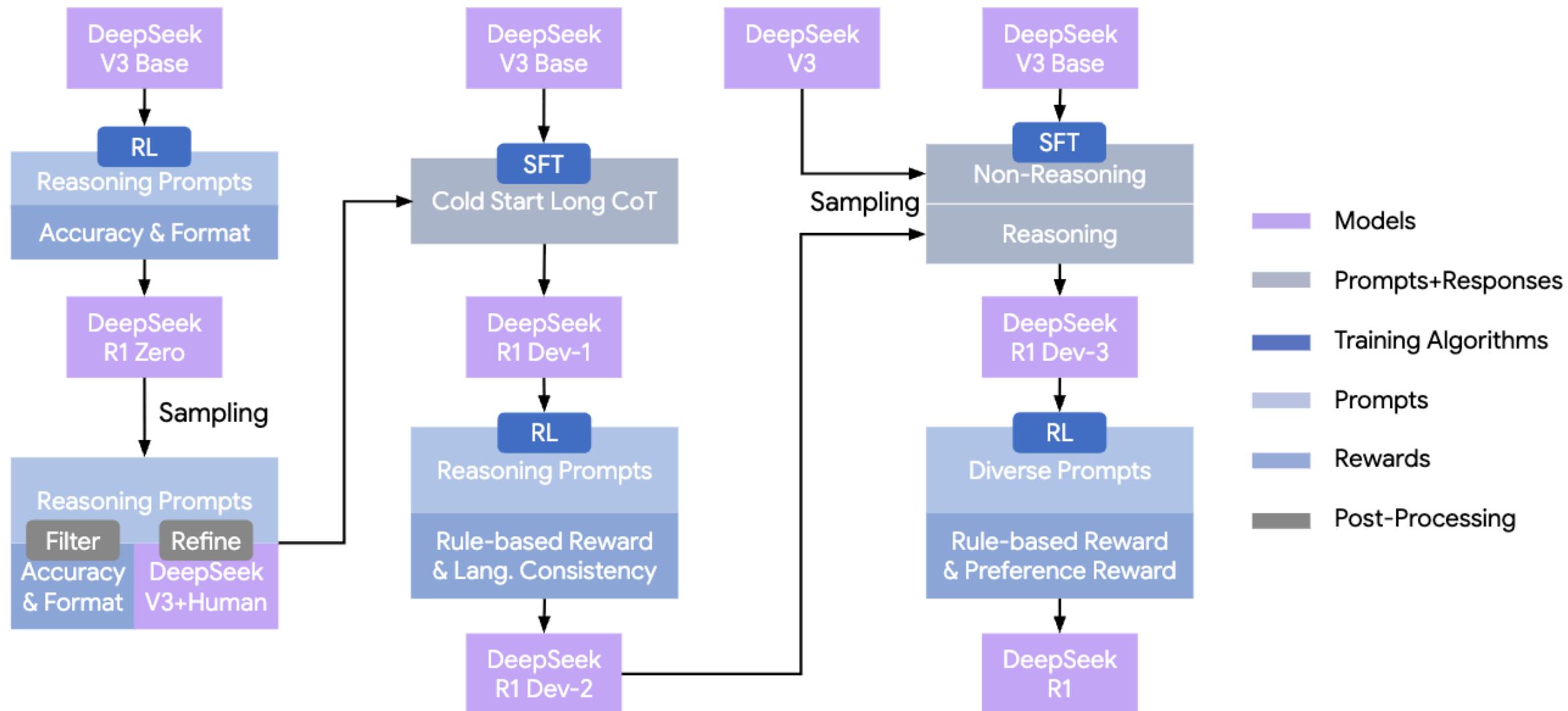
- **Stage 1: R1-Zero**



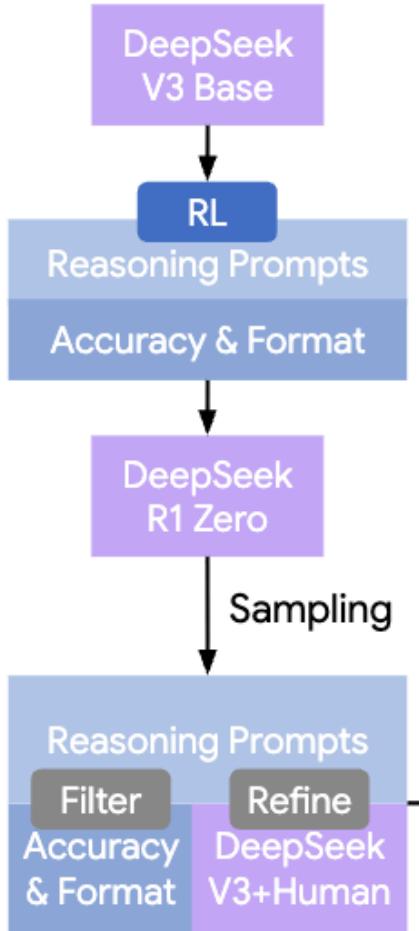
Reasoning Model
Trained with GRPO
Strong Reasoning but poor Readability

DeepSeek-R1

Stage 2: Full DeepSeek-R1



R1-Zero, training

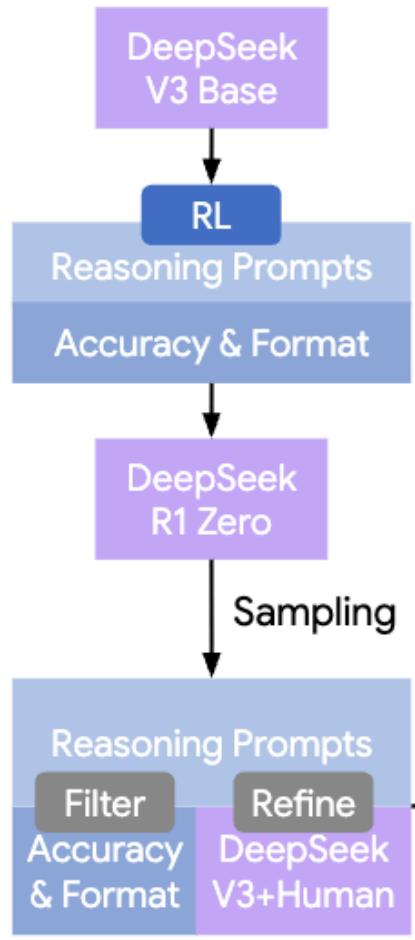


- Trained with GRPO
- Prompts follow this schema

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:

- Typically a CoT prompt to engage the model in reasoning

R1-Zero, Data

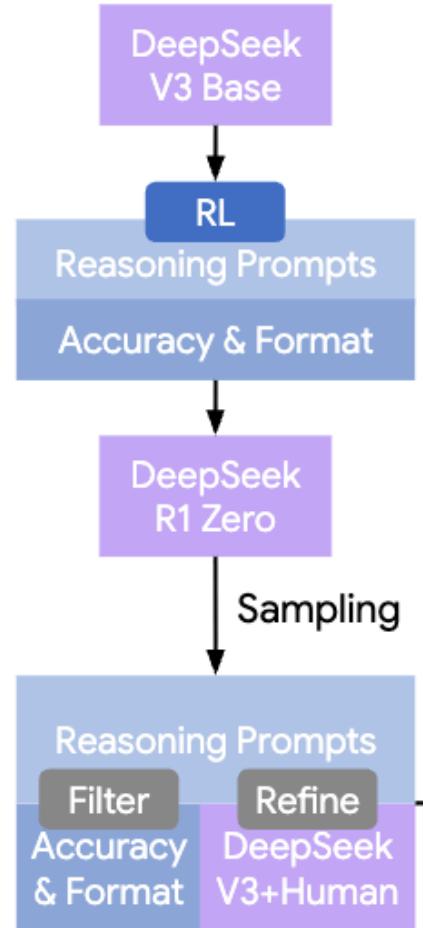


- Trained with GRPO
- 3 domains:
 - Mathematics: 26K quantitative reasoning questions
 - Coding: 17K algorithm competition question
 - STEM: 22K MCQ on physics, chemistry, biology



All are Verifiable!

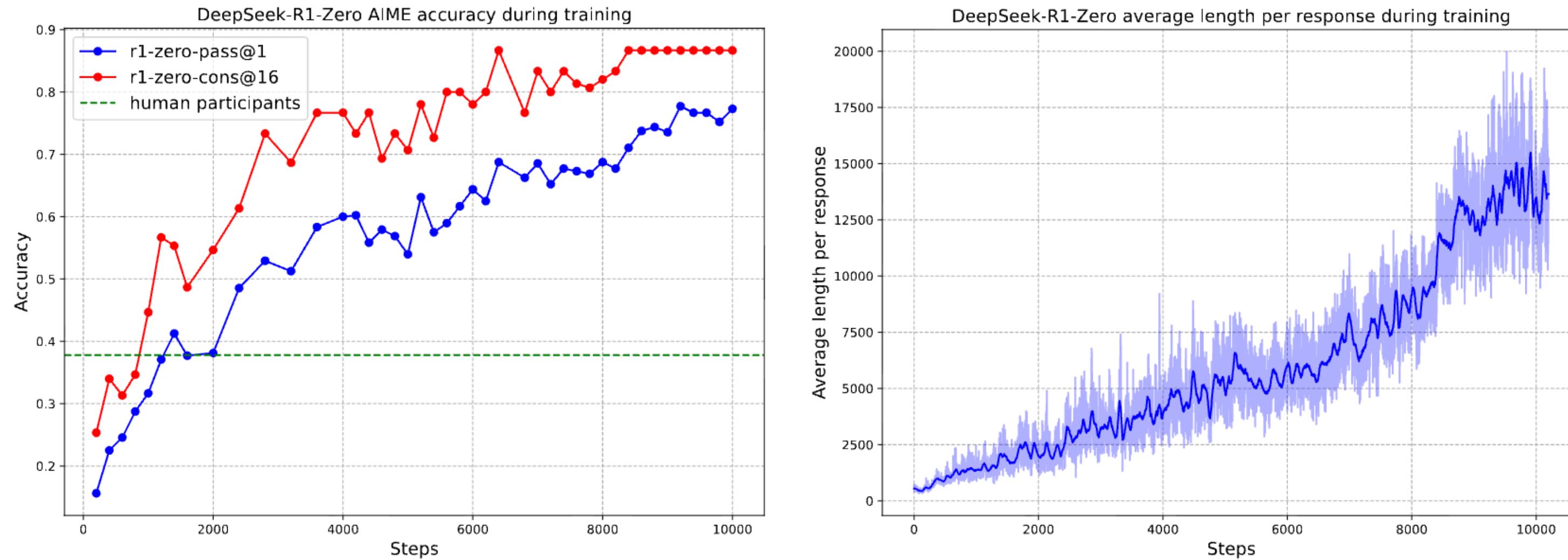
R1-Zero, Rewards



- **Rule-Based Rewards:**
- **Accuracy Reward**
 - Is the model answer the same as ground-truth?
- **Format Reward**
 - Rewards the model when it successfully encloses its thought process in `<think>` `</think>` tokens

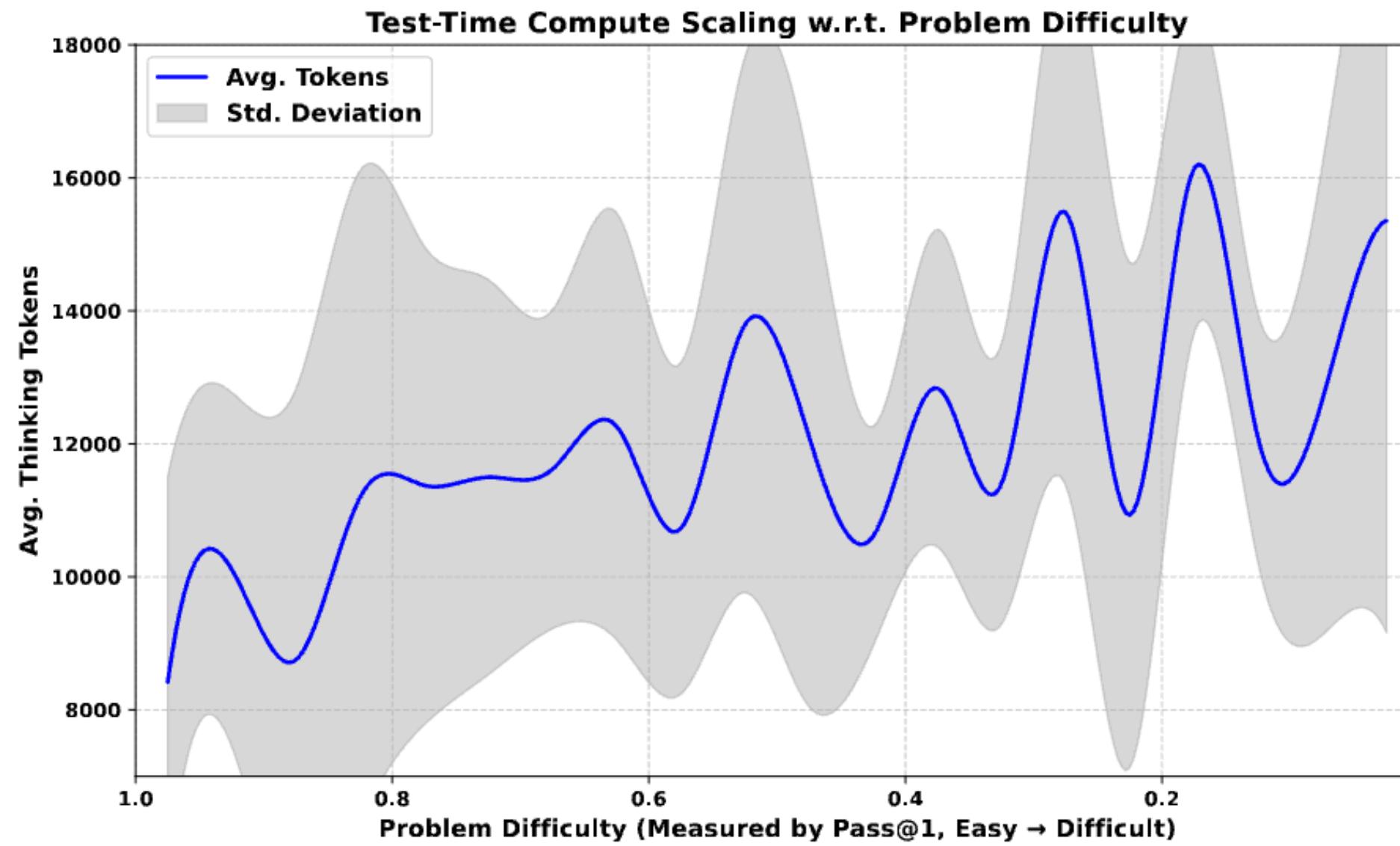
$$Reward_{rule} = Reward_{acc} + Reward_{format}$$

R1-Zero, Some Results



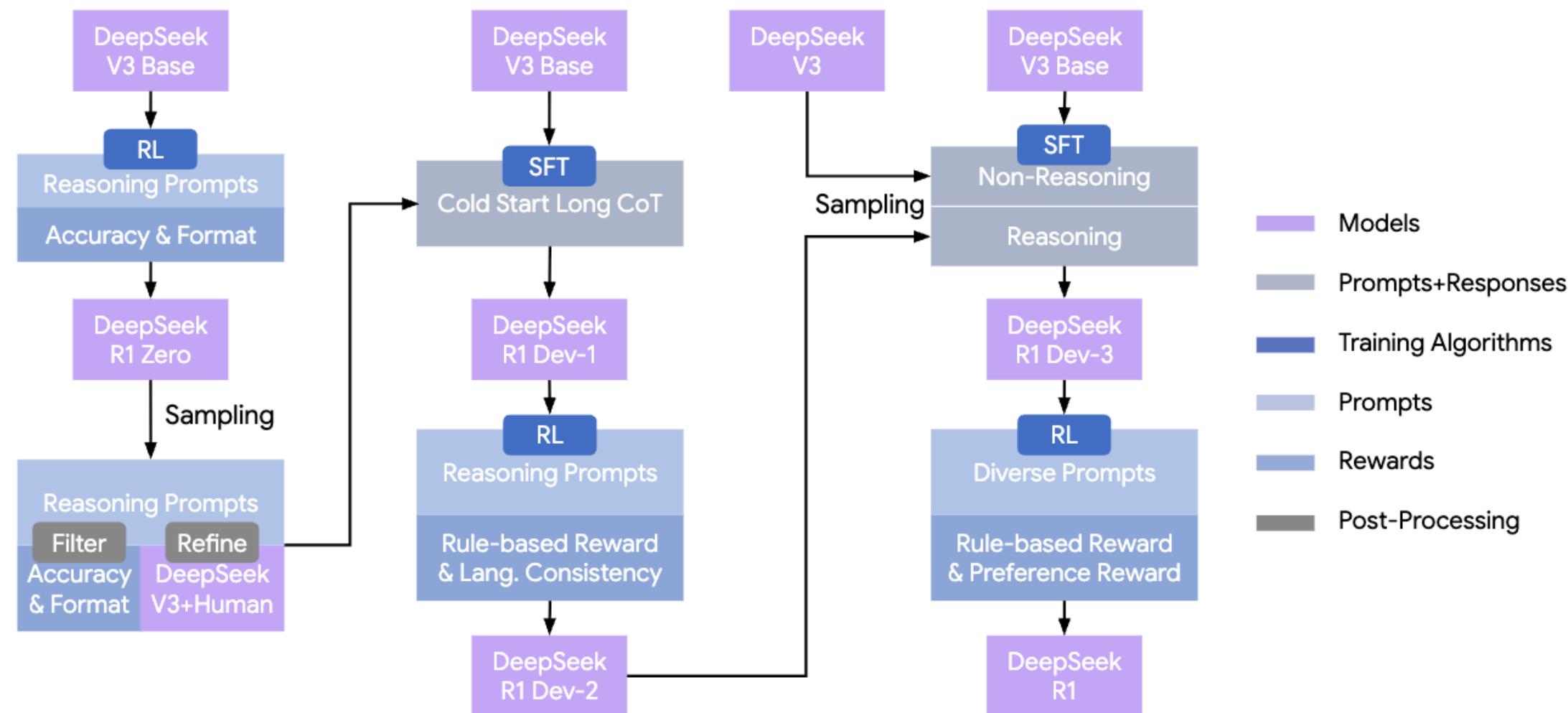
- Sharp increase of number of “Thinking Tokens”, correlated with increased performance on Math problems

R1-Zero, Some Results



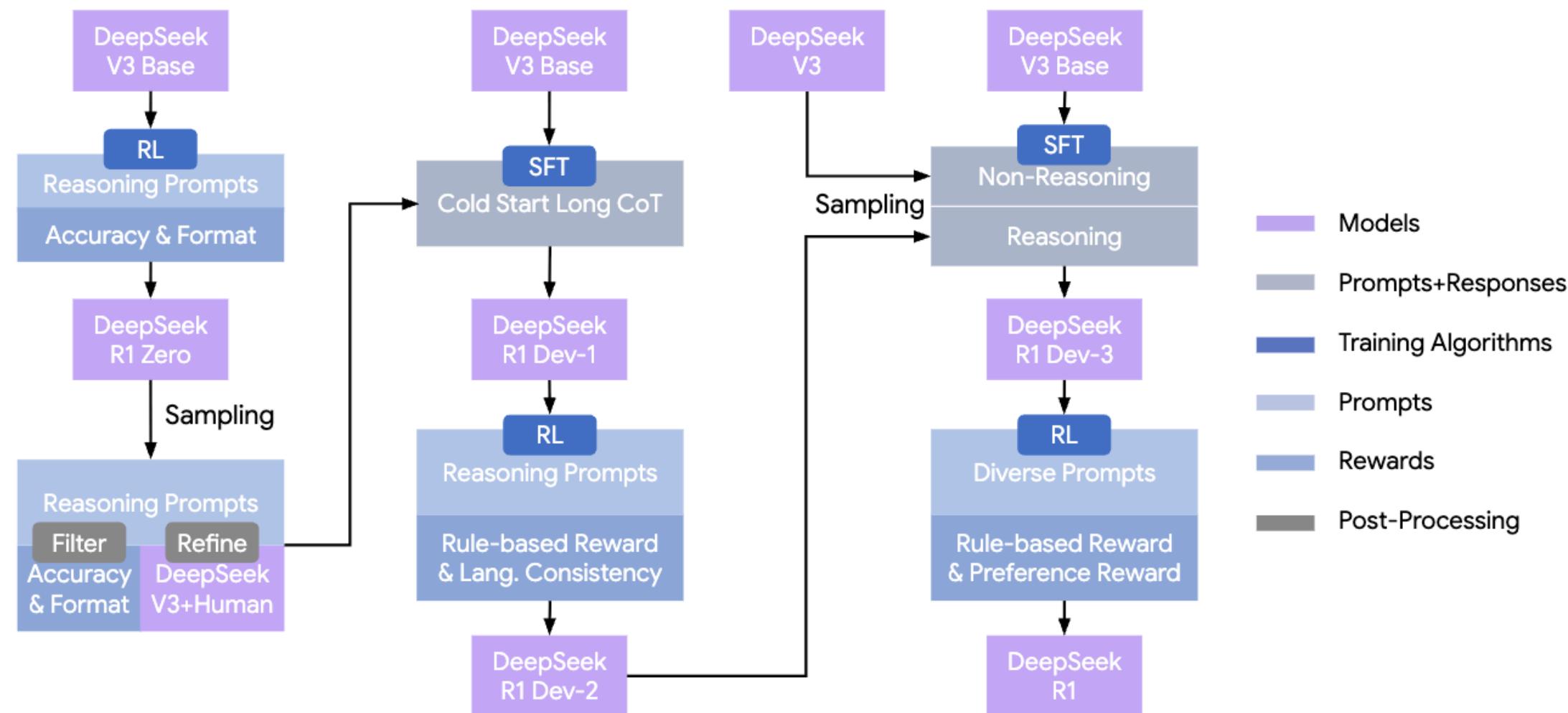
- Number of Thinking Tokens is also correlated with problem difficulty:
Relates to Test-Time Scaling: the more difficult a question, the more reasoning is necessary

DeepSeek-R1



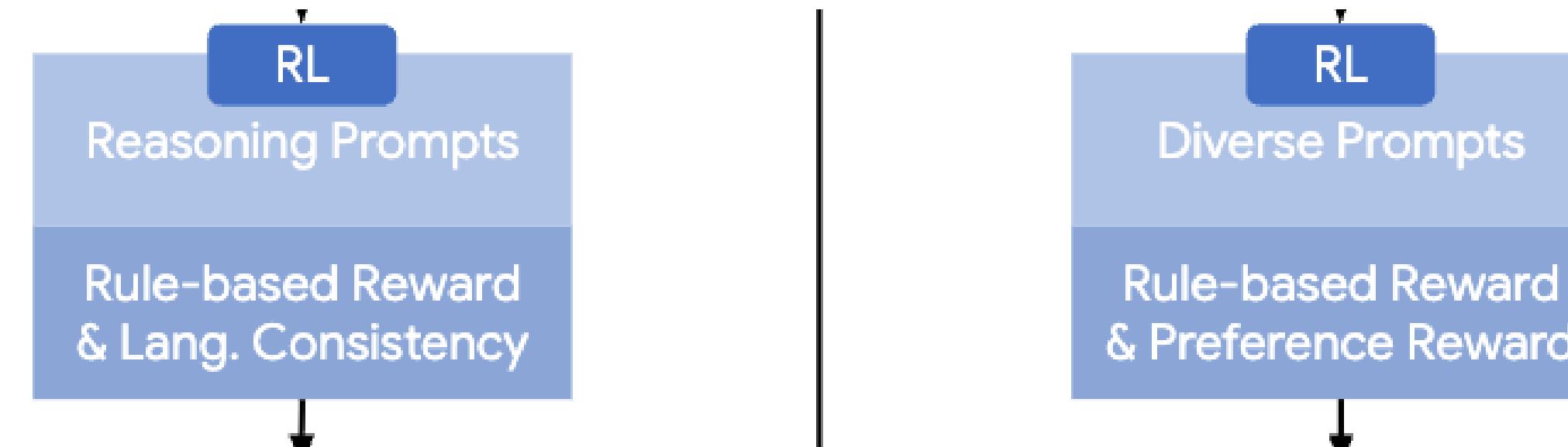
- Full R1 pipeline is a multi-stage approach, where each model checkpoint is used to create **new synthetic data used in next stages**

DeepSeek-R1



- Designed to mitigate R1-Zero issues of **language mixing** and **poor readability**

DeepSeek-R1



- Each RL stage use an additional **LLM-Based Reward Model** that will **score a model answer** on a particular dimension
 - Helpfulness
 - Safety
- Necessary step to have a “user-friendly product”

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:

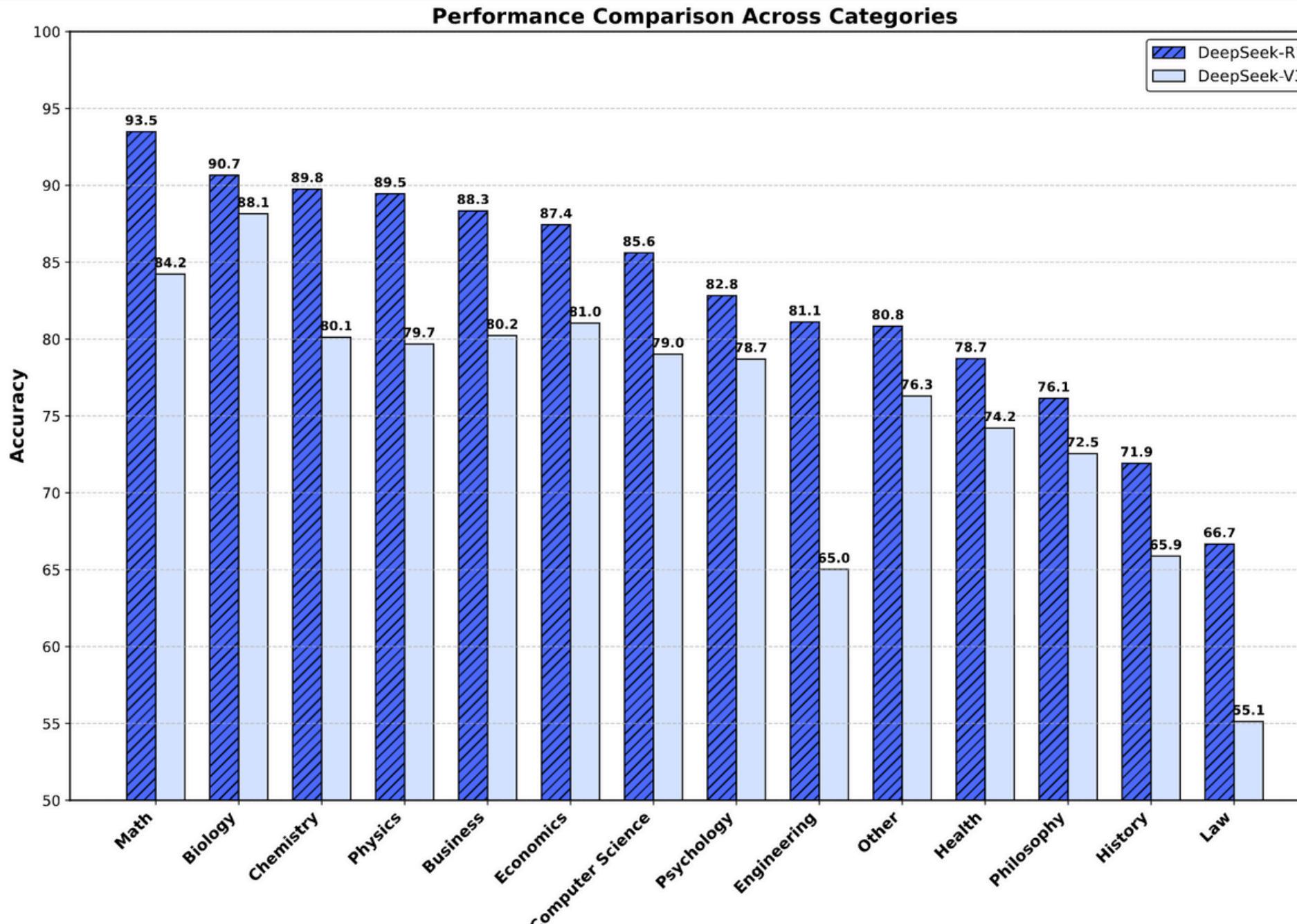
$$a - \sqrt{a + x} = x^2 \implies \sqrt{a + x} = a - x^2$$

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

...

- Emergence of **interesting reasoning patterns**: “the Aha moment”
- The model learns itself to engage in a different thinking process

DeepSeek-R1



MMLU Pro Question Types

- Performance improvement of R1 over the SFT model on all domains

Conclusion & Reward Hacking

- Training a Large Reasoning Model such as DeepSeek-R1 is very challenging
 - Complex pipeline where each step is designed to prevent any specific limitation (*language mixing, readability*)
- A typical issue in RL training is called **Reward Hacking**

“Phenomenon where the LLM exploits biases in the reward function to maximises it without aligning to human preferences”

Conclusion & Reward Hacking

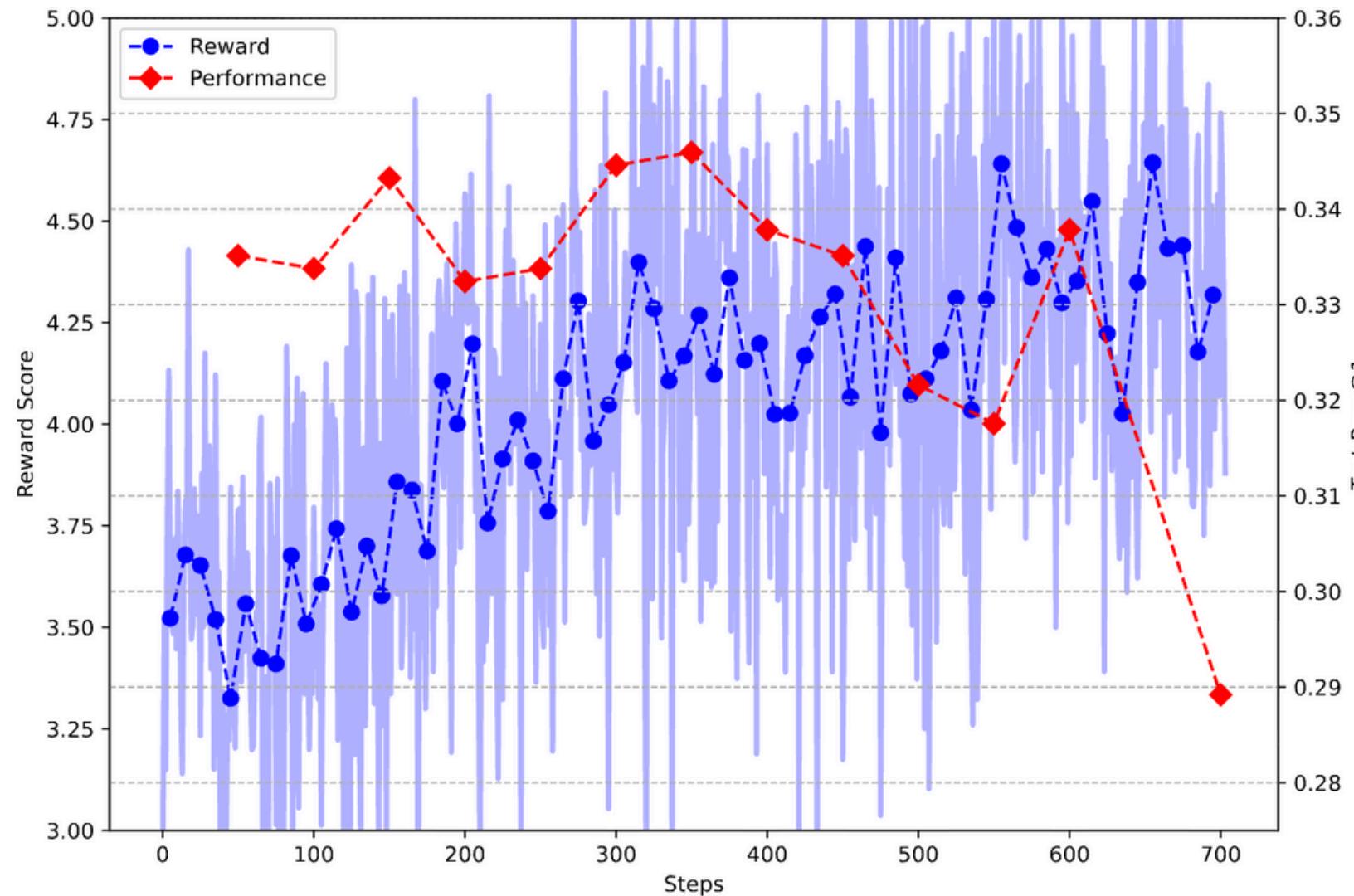


Figure 6 | Reward hacking: the reward exhibits an increasing trend as the performance on CodeForces decreases for training.

“Phenomenon where the LLM exploits biases in the reward function to maximises it without aligning to human preferences”

- Often happens when using a Learned (or LLM) Reward Model
- Using direct solvers on verifiable answers mitigate this issue

Conclusion & Reward Hacking

- Many efforts have been put to try and reproduce the results of DeepSeek-R1, but most have failed
- While DeepSeek weights are open source, the training code isn't

Notes on RL Infrastructure

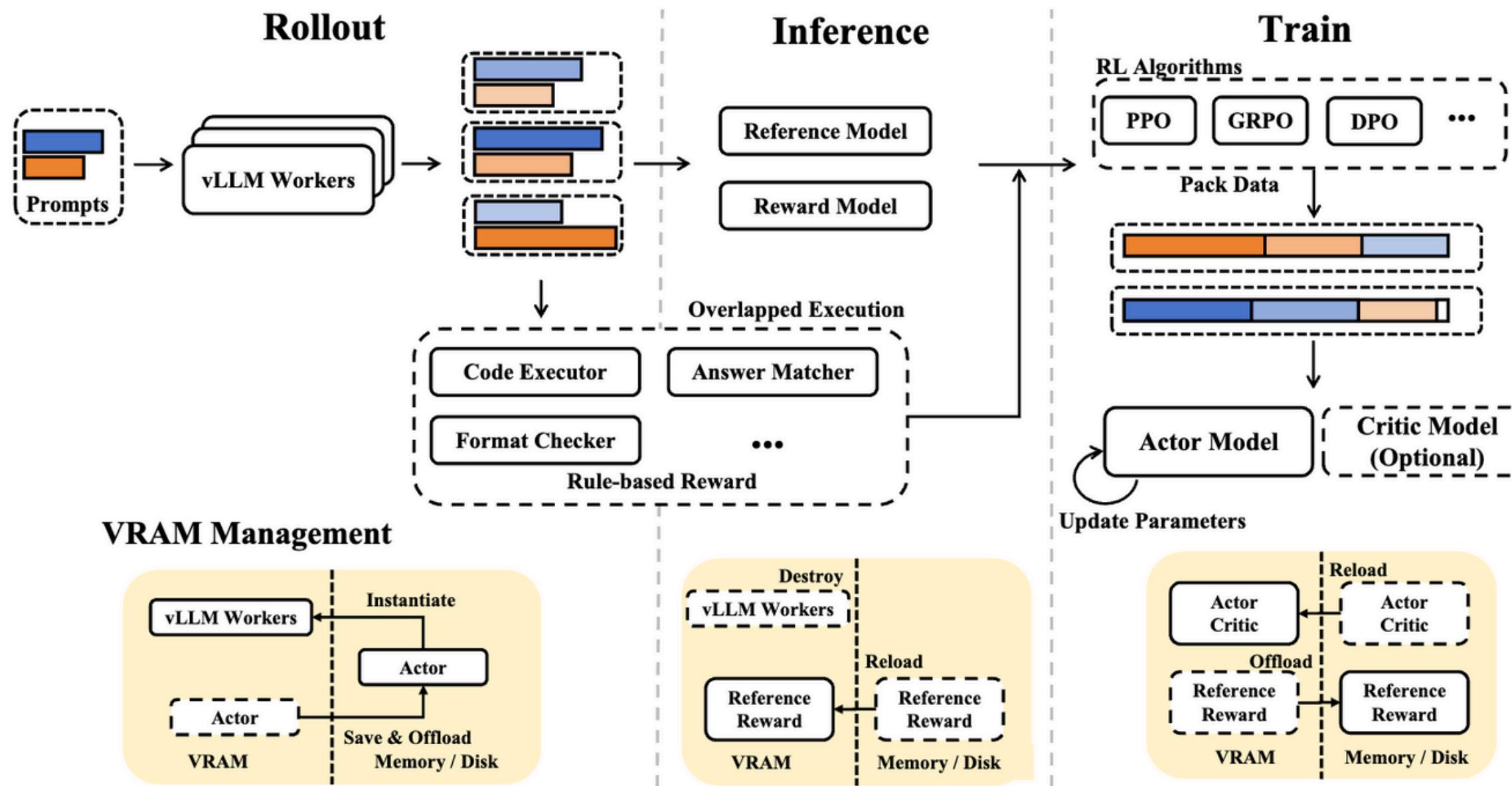
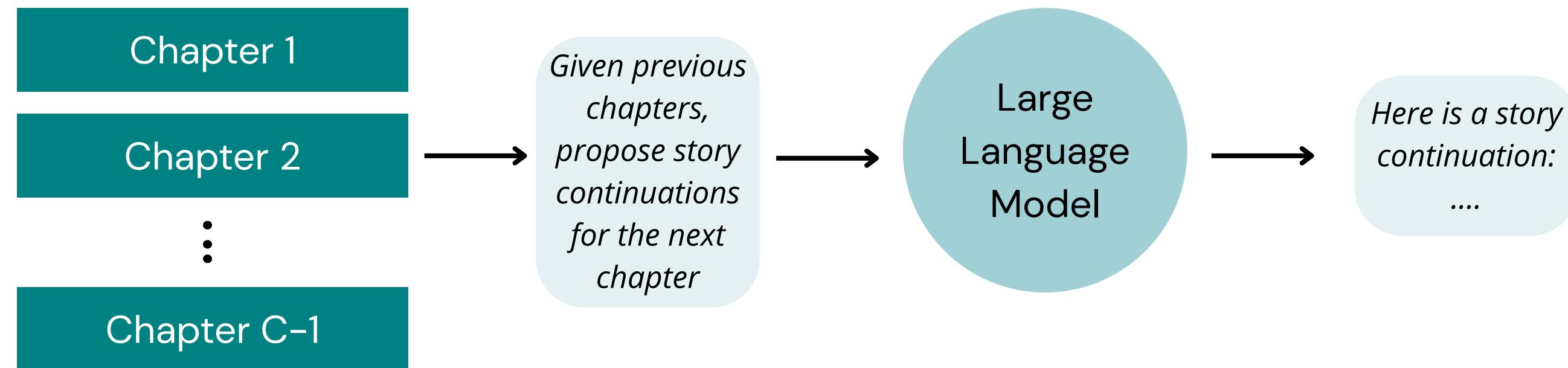


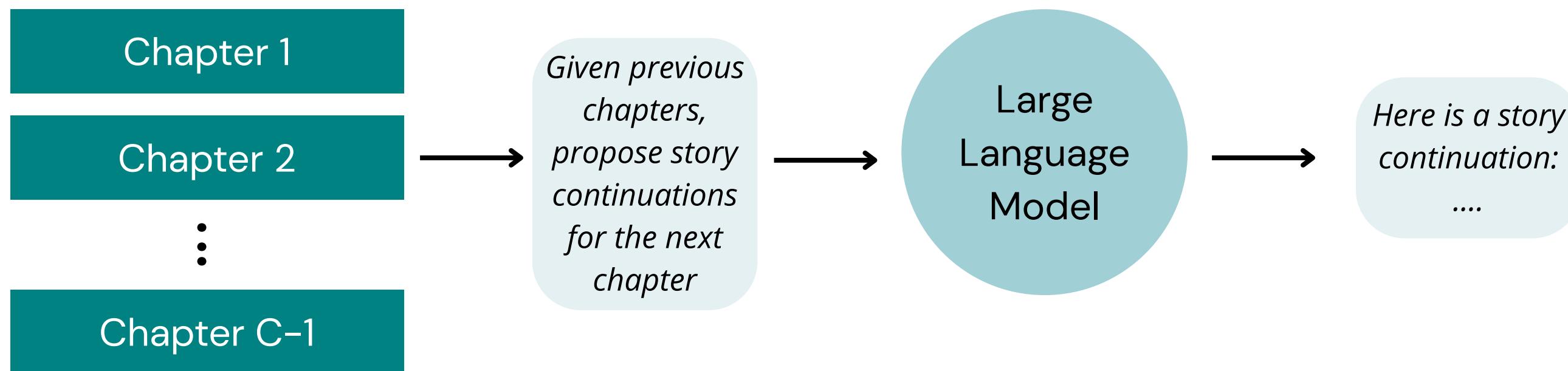
Figure 5 | Overview of our RL framework.

Unverifiable Rewards

- What if we can't **precisely judge** if an answer is good or not?
- We'll illustrate this with the domain of **Creative Writing**
- Suppose we have access to a book information until chapter C-1 and we'd like a model to propose **story continuations for chapter C**



Unverifiable Rewards



- What makes a **good story continuation?**
 - Depends on **genre, author style** and other more complex elements of **narrative theory**

Unverifiable Rewards

- How do you think we could reward good continuations?

Unverifiable Rewards

- How do you think we could reward good continuation?
1. We could use an external Reward Model who judges if snippet is good based on user-defined constraints (genre, style, etc..), but:

Unverifiable Rewards

- How do you think we could reward good continuation?
- 1. We could use an external Reward Model who judges if snippet is good based on user-defined constraints (genre, style, etc..), but:
 - Suggest to reward hacking: the trained policy might find *reward hacks*

Unverifiable Rewards

- How do you think we could reward good continuation?
- 1. We could use an external Reward Model who judges if snippet is good based on user-defined constraints (genre, style, etc..), but:
 - Suggest to reward hacking: the trained policy might find *reward hacks*
 - Reward Model biases will be inherently copied by the policy → can lead to overly-biased models (e.g. *favors white-male characters and defavors non-white female characters*)

Unverifiable Rewards

- How do you think we could reward good continuation?
- 1. We could use an external Reward Model who judges if snippet is good based on user-defined constraints (genre, style, etc..), but:
 - Suggest to reward hacking: the trained policy might find *reward hacks*
 - Reward Model biases will be inherently copied by the policy → can lead to overly-biased models (e.g. *favors white-male characters and defavors non-white female characters*)
 - Multiple answers to the same prompt are valid → the reward model might overfit on one possible “way of seeing the problem”

Unverifiable Rewards

- How do you think we could reward good continuation?

2. We could use user-defined heuristics:

Unverifiable Rewards

- How do you think we could reward good continuation?

2. We could use user-defined heuristics:

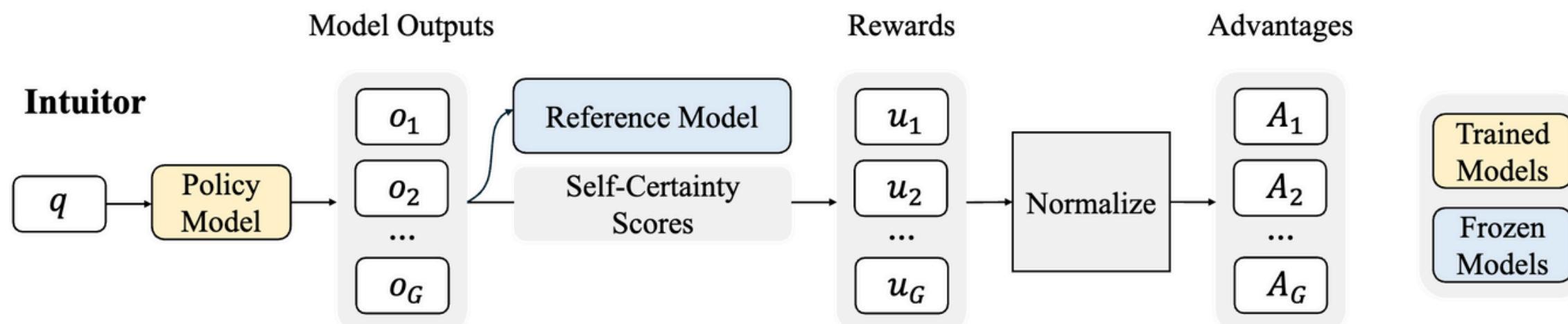
- Rewards if characters A, B and C are present in the snippet
- Rewards if characters A and B interact
- etc...
- But it's hard to find objective heuristics at training time
 - It's more like a "test-time" type of constraints

Unverifiable Rewards

- **Unverifiability is present in others domains:**
 - Ambiguous Question – there are multiple answers depending on what is the primary intent
 - Social Sciences – Often multiple valid opinions
 - Cultural Studies – Preferences are anchored in cultural values we have been raised with

Unverifiable Rewards

- A common solution to this problem is to use **Policy-Intrinsic rewards**
- This allows the model to receive **Internal Feedback** at each training iteration
- A typical internal feedback is a measure of a model *self-confidence*: how uncertain is the model when generating an answer
- Self-confidence is typically a **log-probability measure over the model distribution**





Reinforcement Learning with Unverifiable Rewards

Case Study: Story Generation

Gurung, Alexander, and Mirella Lapata. "Learning to reason for long-form story generation." arXiv preprint arXiv:2503.22828 (2025).

The Next Chapter Prediction Task

Prior Story Information

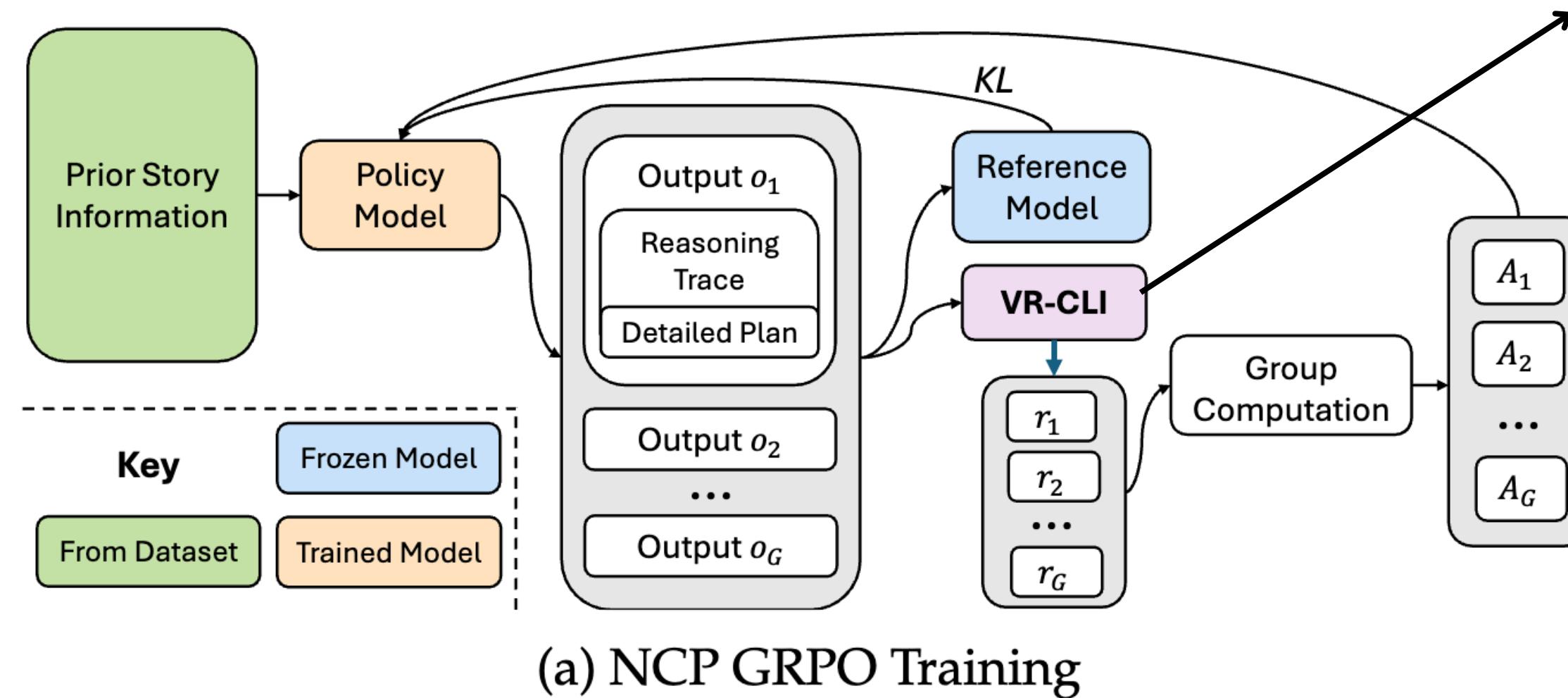
Story Information	Sample Sentence
Global Story Sketch	As the story progresses, it becomes clear that Barbara's family is complex and troubled.
Previous Story Summary	As the campers settle into their routine, Tracy becomes friends with Barbara, who arrives at the camp on foot (Chapter 7).
Character Sheets	She wants to be a good mother to Bear and seeks validation through her role as a mother.
Previous Chapter	The implication, Alice understood, was that to have more than one boy would complicate matters when it came to passing on the bank.
Next Chapter Synopsis	Peter tells Carl that his son is missing.

Desired output

Next Chapter	And when a voice came through the wires, it wasn't a member of the staff, but Peter Van Laar himself—to whom Carl nodded each time they crossed paths at the Preserve, but to whom he had actually spoken maybe twice in his life.
--------------	--

GRPO Setup

A self-confidence based reward

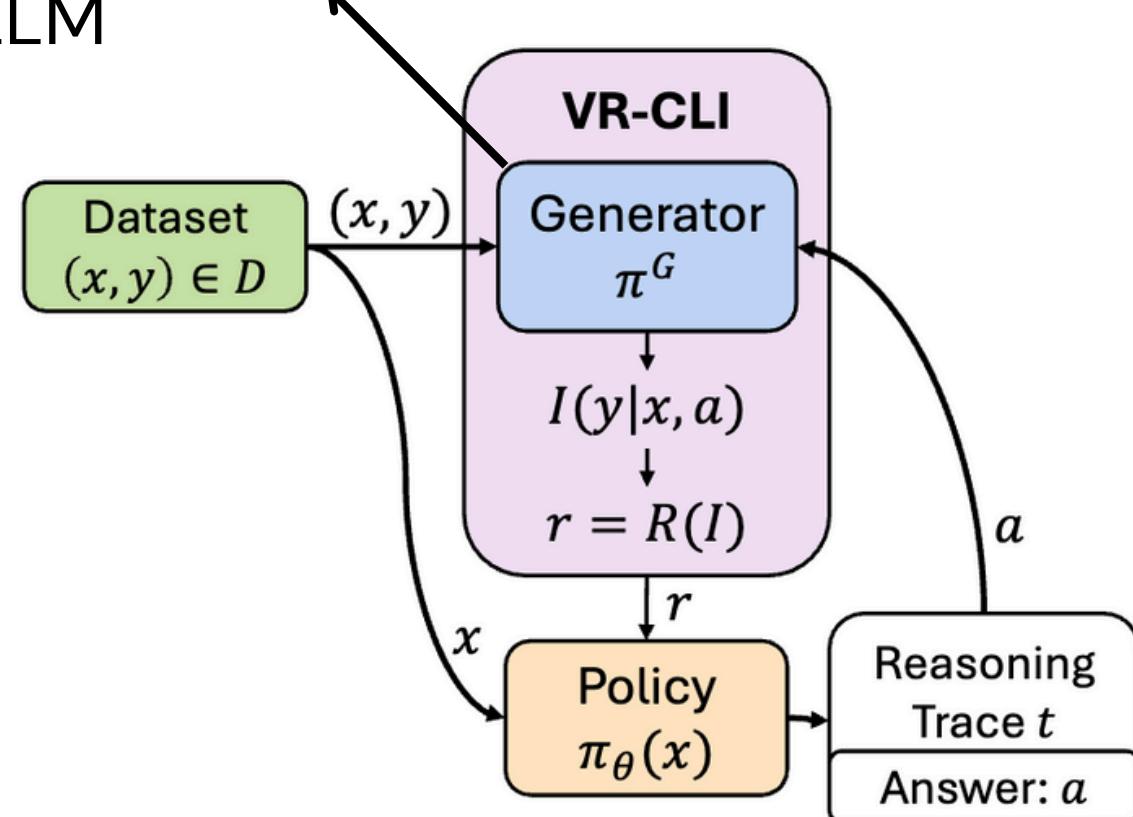


VR-CLI

- Verifiable Rewards via Completion Likelihood Improvement (**VR-CLI**)

Frozen SFT

LLM



- x is the story information
- y is the gold chapter text
- a is a generated **writing plan** (target output)
- PPL_{π^G} is the per-token Perplexity under Generator
- Then the perplexity improvement when using the writing plan is:

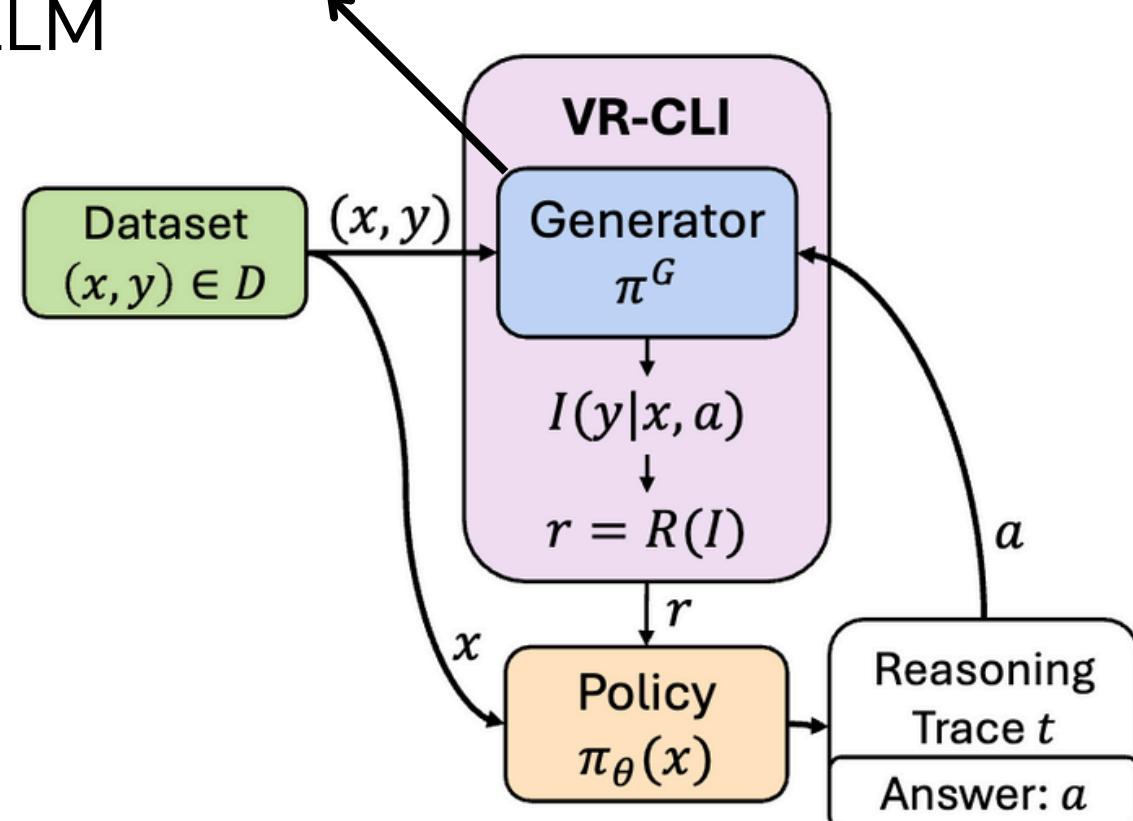
$$I(x, y, a) = \frac{PPL_{\pi^G}(y|x) - PPL_{\pi^G}(y|x, a)}{PPL_{\pi^G}(y|x)}$$

VR-CLI

- Verifiable Rewards via Completion Likelihood Improvement (**VR-CLI**)

Frozen SFT

LLM



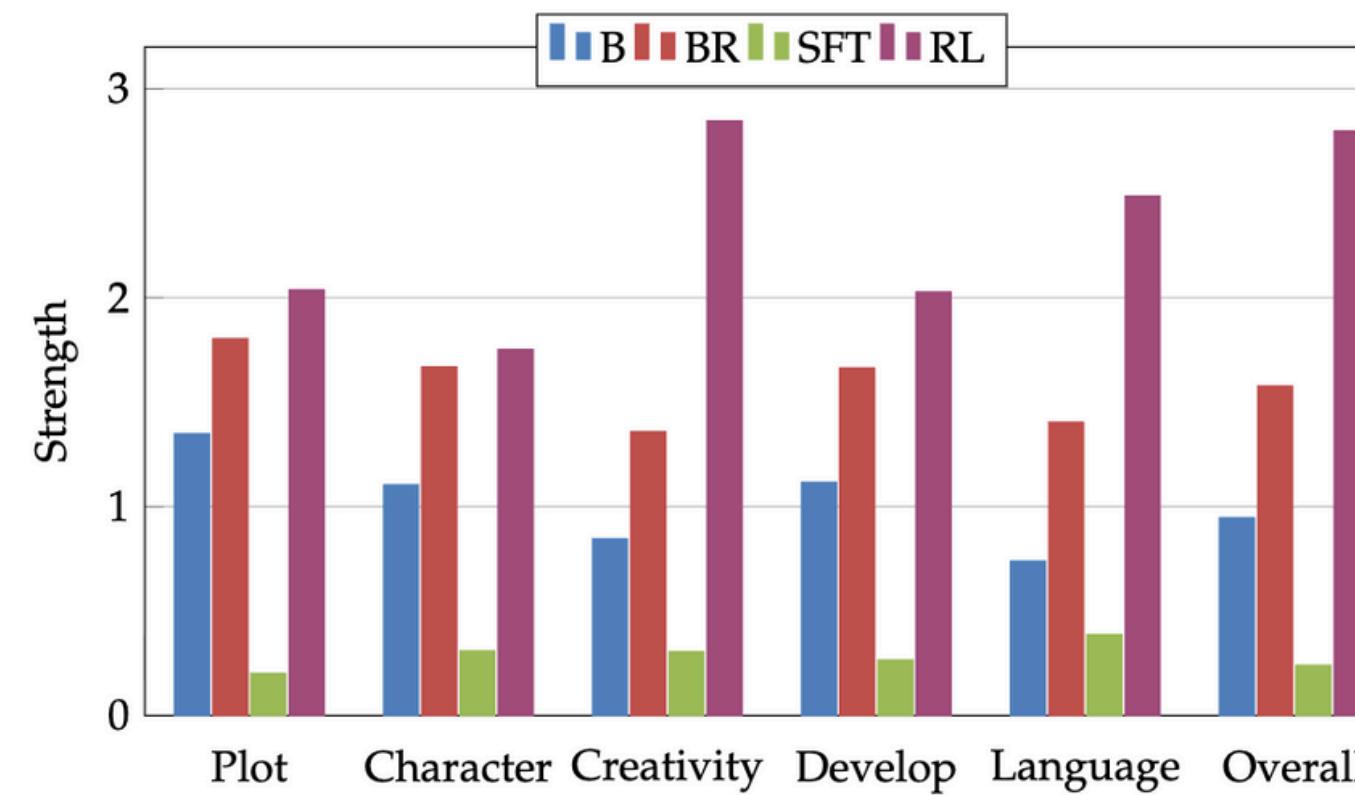
- Then the perplexity improvement when using the writing plan is:

$$I(x, y, a) = \frac{PPL_{\pi^G}(y|x) - PPL_{\pi^G}(y|x, a)}{PPL_{\pi^G}(y|x)}$$

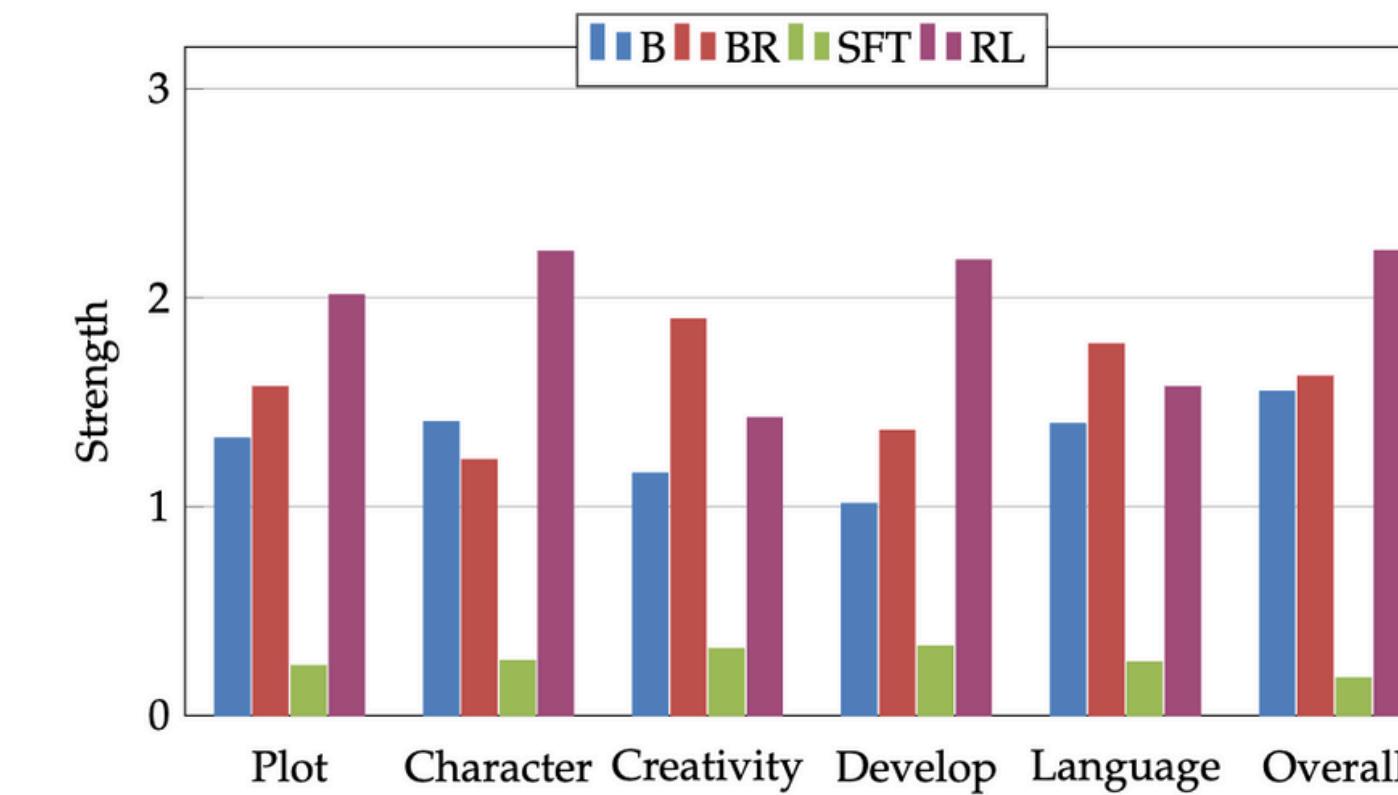
- The Reasoning model is trained to generate a **useful writing plan**, that helps the generator to improve the likelihood of generating the ground-truth chapter

Human Evaluation

- Humans are given 2 model completions and are asked to provide preferences along specific dimensions



(a) 7B Bradley Terry Relative Strength



(b) 3B Bradley Terry Relative Strength

Reasoning Trace Example

Reasoning Trace After Training, Qwen 2.5-7B-Instruct (2.58% Improvement)

Detailed Plan for Next Chapter###

1. **Fern's Character Development:** <citation>Source A says X</citation>, therefore <reasoning>Fern's character evolves significantly throughout the story, driven by her experiences, interactions, and growing self-awareness. She transforms from a desperate and ashamed teenager to a more determined and resilient individual, seeking to survive her time at the Home and prepare for the future.</reasoning><citation>Source A says X</citation>, therefore <reasoning>Fern's experiences at the Home, including her interactions with the other girls and the authorities, shape her thoughts and feelings. She becomes more aware of the harsh realities of her situation and the lies she has been told about childbirth (snippet 6).</reasoning>
2. **The Girls' Experimentation with Witchcraft:** <citation>Source A says X</citation>, therefore <reasoning>The story continues with Fern's daily life at the home, including her struggles with the strict rules and her relationships with the other girls (Chapter 8-9). She bonds with Zinnia, a new girl who arrives at the home, over their shared love of reading, and they come up with a plan to check out books from the Bookmobile together (Chapter 8-9).</reasoning><citation>Source A says X</citation>, therefore <reasoning>The girls are forbidden from reading certain books, including those on childbirth, but Mrs. Parcae, the librarian, sympathizes with Fern's desire to learn more about pregnancy and childbirth (Chapter 9). In a significant moment, Mrs. Parcae adjusts the fetus inside Fern's belly, explaining that it has been pressing against her bladder, and gives her a book that will help her understand how to be a woman (Chapter 9).</reasoning>
3. **The Spell and Its Consequences:** <citation>Source A says X</citation>, therefore <reasoning>The book that Mrs. Parcae gave Fern is titled How to Be a Groovy Witch. It contains spells for various things, all of which seem useful to Fern. Because Mrs. Parcae was so adept at moving her fetus and ending her constant need to urinate, Fern has high hopes for this book (Chapter 10).</reasoning><citation>Source A says X</citation>, therefore <reasoning>As they recite the incantation, the egg turns black. They hope that this means that the spell is working. When they transfer the egg and the other items into a witch bag and hide it in Dr. Vincent's office, he begins vomiting uncontrollably (Chapter 10).</reasoning>

In summary: In the next chapter, Fern's character development is portrayed through her experiences at the Home for unwed mothers. The chapter ends with the girls' experimentation with witchcraft, and the spell they cast to harm Dr. Vincent. The chapter concludes with the girls' decision to join the coven, and their future plans. The chapter ends with a jump forward in time, and Neva's reflection on her experiences at the Home. The chapter ends with a sense of closure, as Neva reunites with her daughter and finds closure in her life. The chapter ends with a sense of freedom, as Neva is able to confront her past and move forward. The chapter ends with a sense of satisfaction, as Neva's life is complete. The chapter ends with a sense of closure, as the story comes to an end. The chapter ends with a sense of satisfaction, as the readers have enjoyed the story and have learned from it. The chapter ends with a sense of closure, as the story has come to an end. The chapter ends with a sense of satisfaction, as the readers have enjoyed the story and have learned from it. The chapter ends with a sense of closure, as the story has come to an end.

Conclusion

- In Unverifiable setups, we need to frame **intelligent rewards** that are likely to help LLMs generate better answers
- In **VR-CLI**, the reward use an external generator model that will be further used to produce the final generations
 - The Policy Model is only trained to generate **plans/summaries** that are likely to help the generator
 - This design is **crucial**: training to generate answers directly is more prone to instabilities in unverifiable setups