
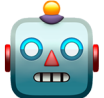






Reinforcement Learning for Language Model Alignment

LM post-training methods: RLHF, DPO, GRPO

Motivation

-  Language models trained on more and more data
-  Language models getting larger and better
-  Language models as world models?
-  Language models as multitask assistants?
-  Language models following human instructions?
-  Language models incorporating human values?

Outline

1. Language model training

2. Reinforcement Learning

3. Reinforcement learning from human preferences (RLHF)

4. Group Relative Policy Optimization (GRPO)

5. Direct Preference Optimization (DPO)

Language model

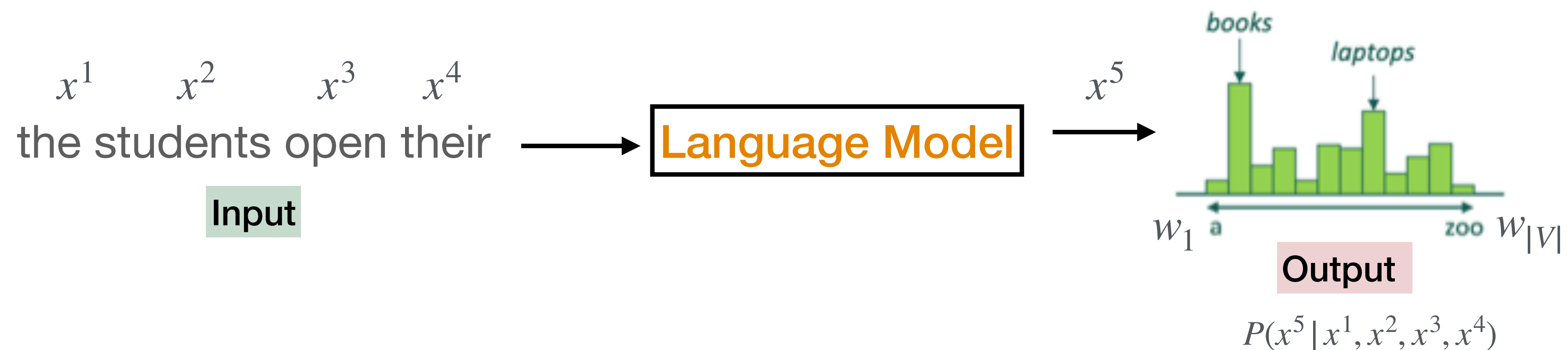
LM (language model): predicts next word given input

- Input: text
- Output: predict the next word by sampling

Formal definition: compute the probability distribution of next word given a sequence of previous words, x^1, x^2, \dots, x^t

$$P(x^{(t+1)} | x^1, \dots, x^t)$$

where $x^{(t+1)}$ can be a word in the dictionary $V = \{w_1, \dots, w_{|V|}\}$



LM training

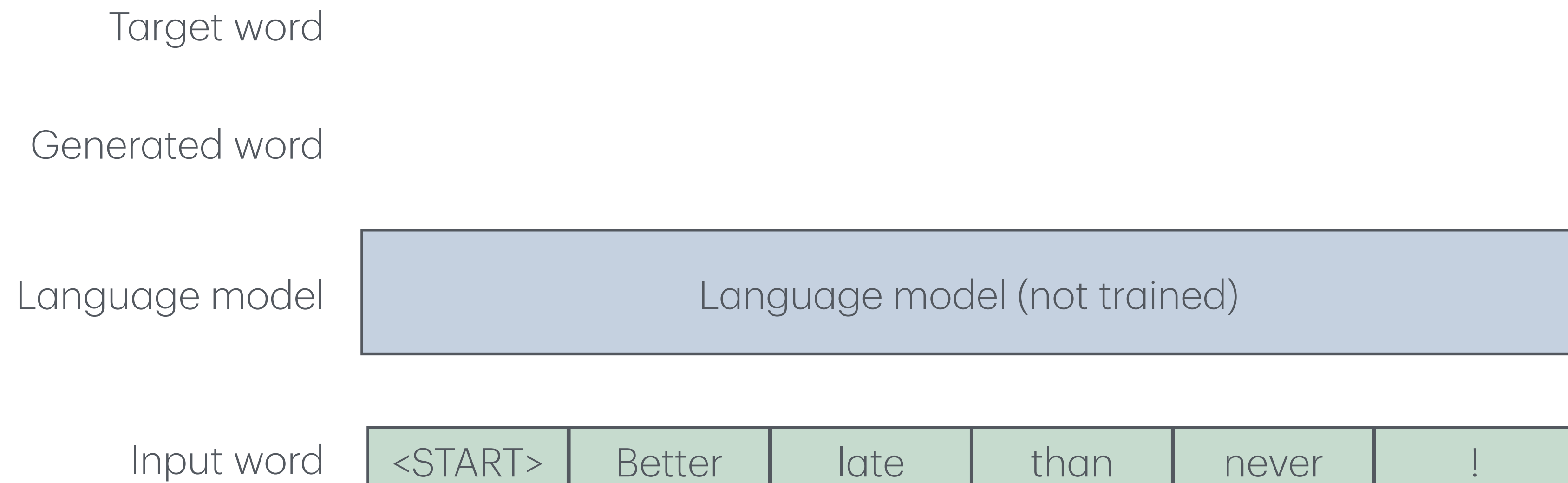
- Pre-training: model with random weights trained with large but unlabeled data
- Post-training:
 - Supervised fine-tuning (SFT) (or Instruction fine-tuning):
 - Trained with labeled data (question/answer pair)
 - RL training
 - Trained with preference data to optimize reward

Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's *self-supervised* training because there are no explicit labels, but input text is used as both input and target label.

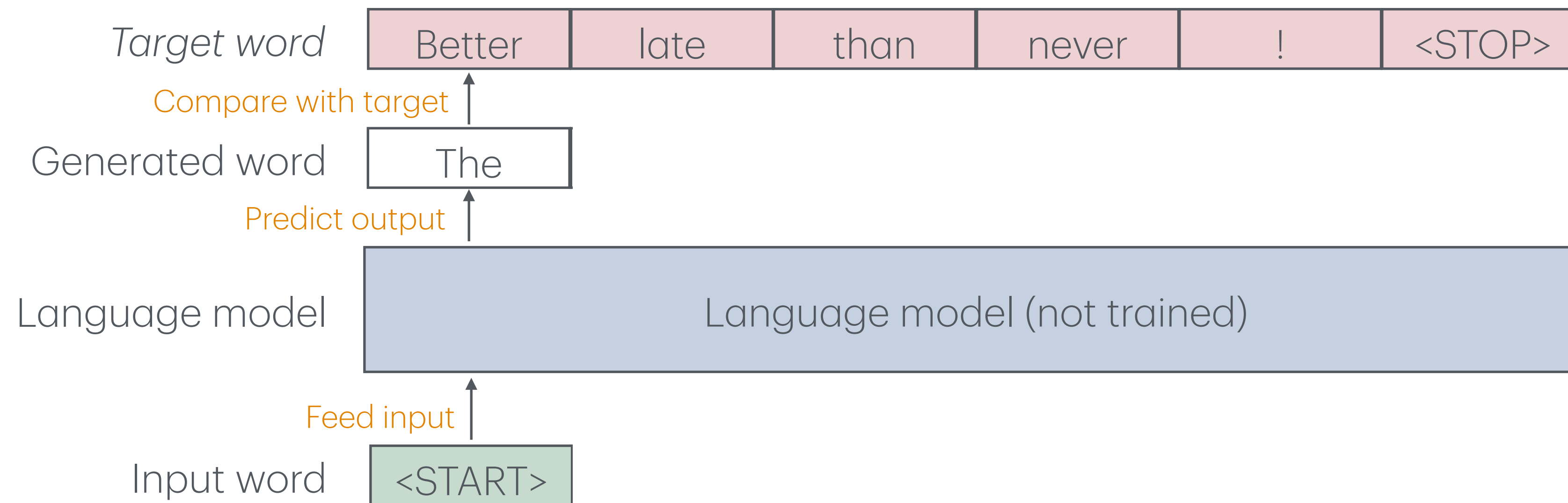


Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.

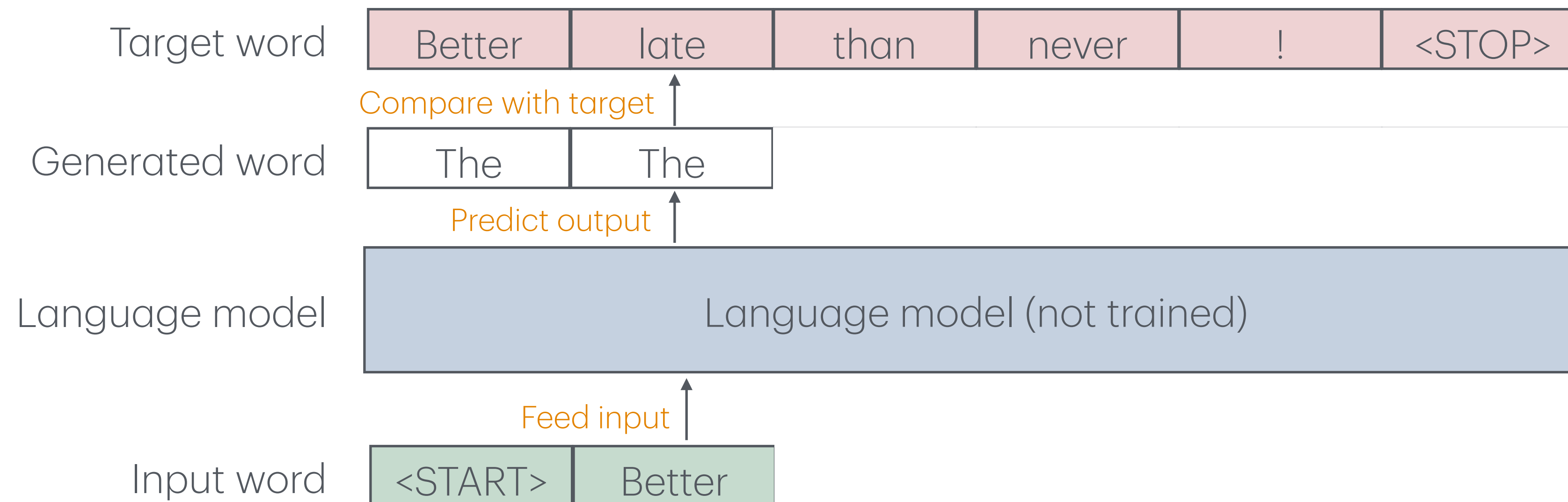


Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.

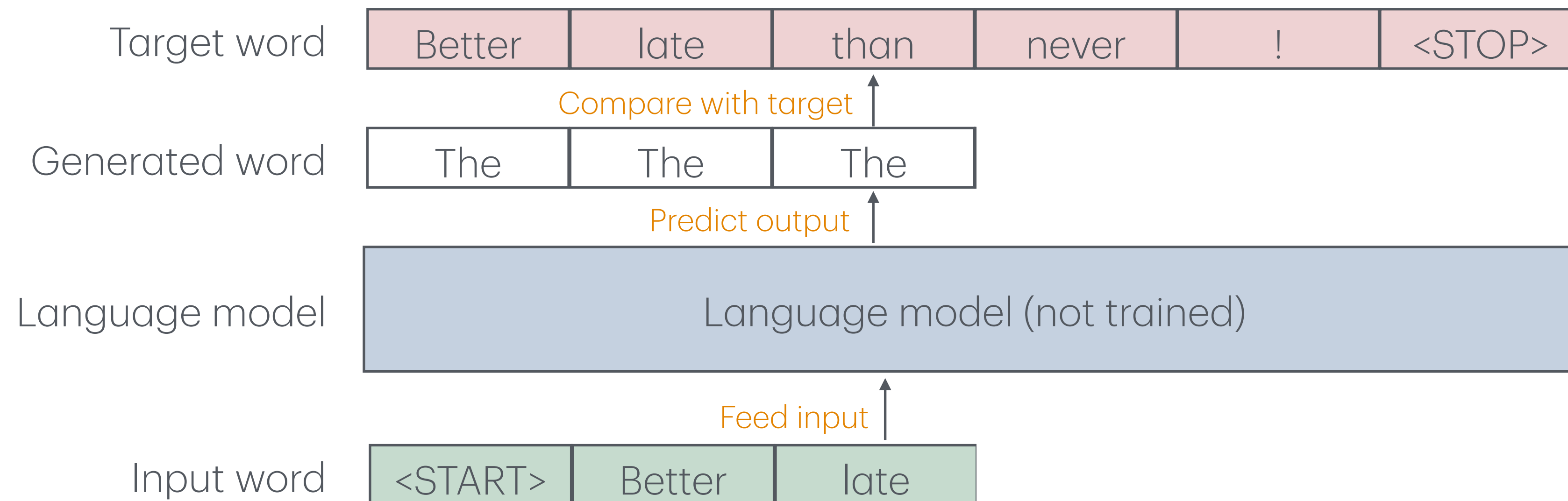


Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.

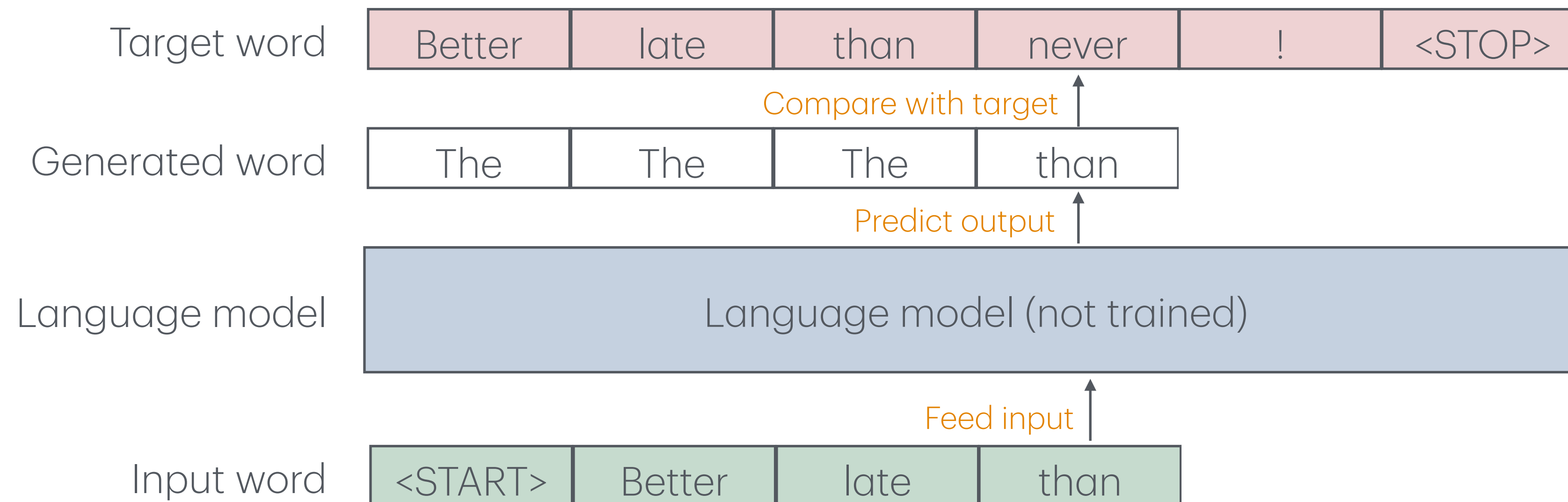


Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.

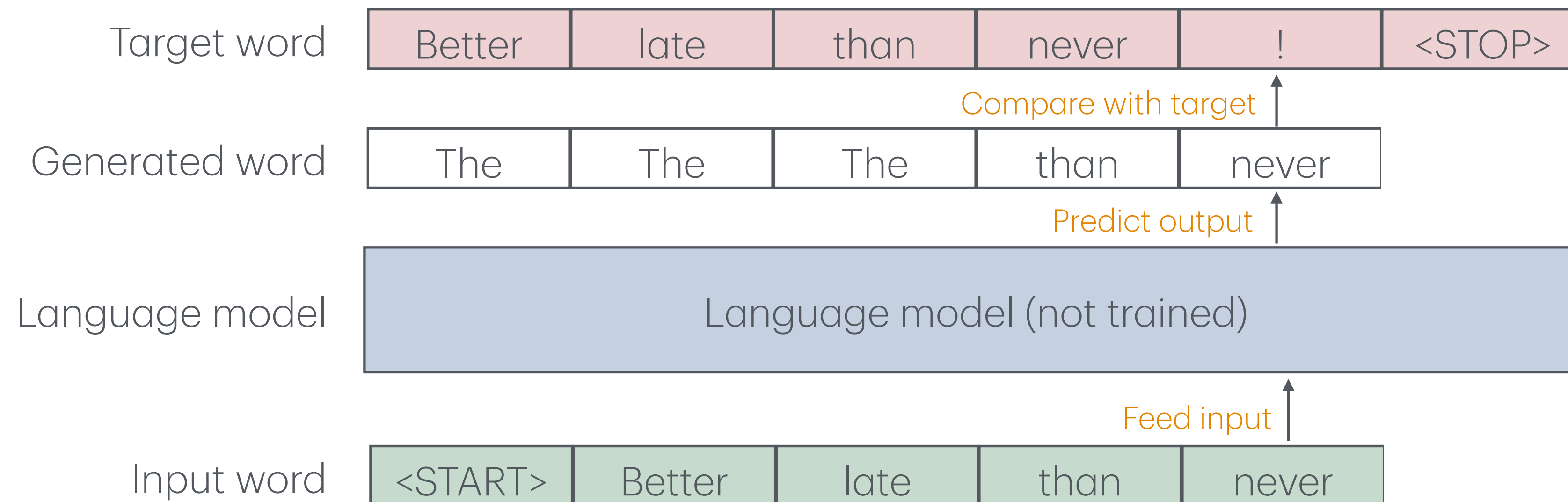


Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.

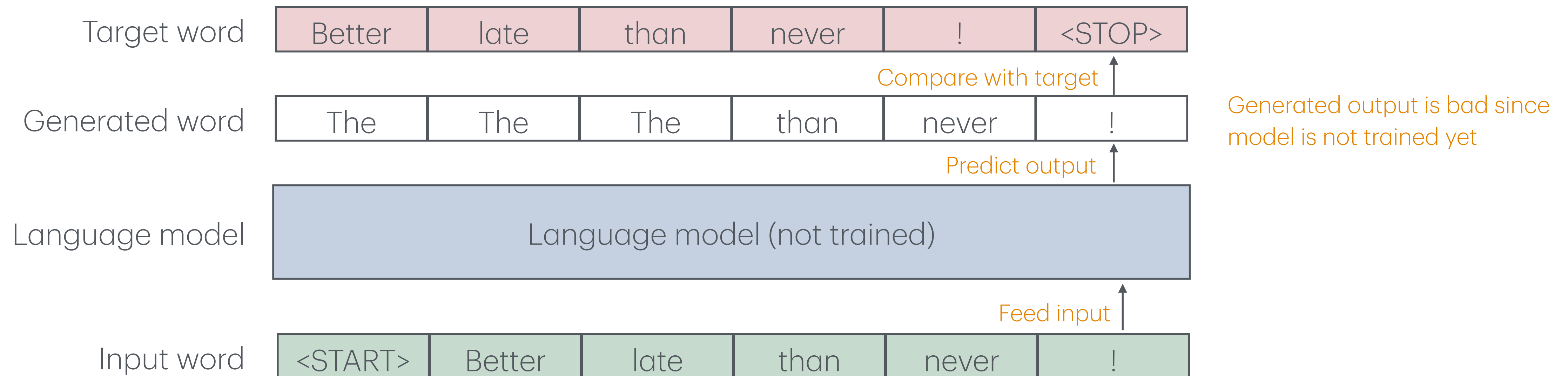


Pre-training LM using Self-supervision

Language models are trained to *predict the next word one at a time* using large text corpus.

Input sentences are fed to the model one by one, and the next word is the target label for the training.

It's self-supervised training because there are no explicit labels, but input text is used as both input and target label.

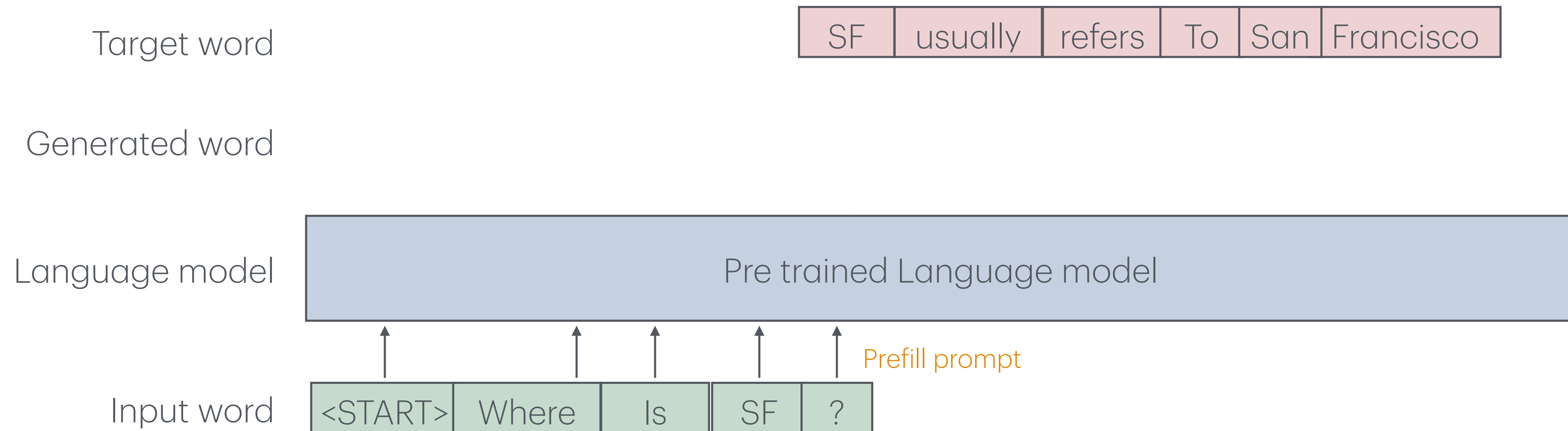


Post-training LM using SFT

Supervised fine-tuning (or instruction fine-tuning) training trains the model to generate output that matches the ground true answer.

Q: where is SF?

A: SF usually refers to San Francisco, a city in California, USA

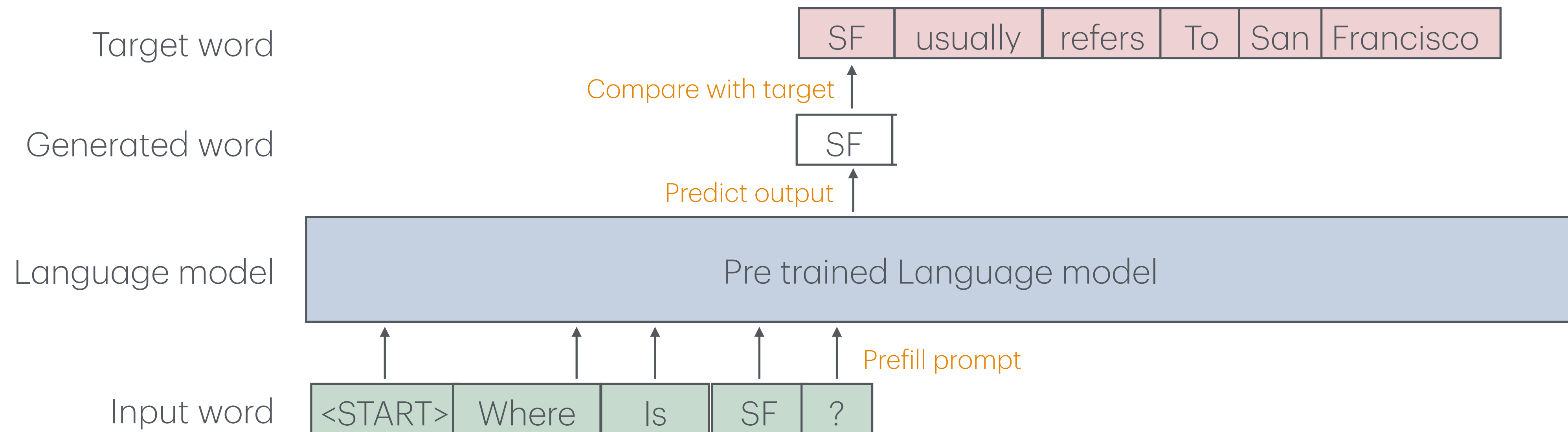


Post-training LM using SFT

Supervised fine-tuning (or instruction fine-tuning) training trains the model to generate output that matches the ground true answer.

Q: where is SF?

A: SF usually refers to San Francisco, a city in California, USA

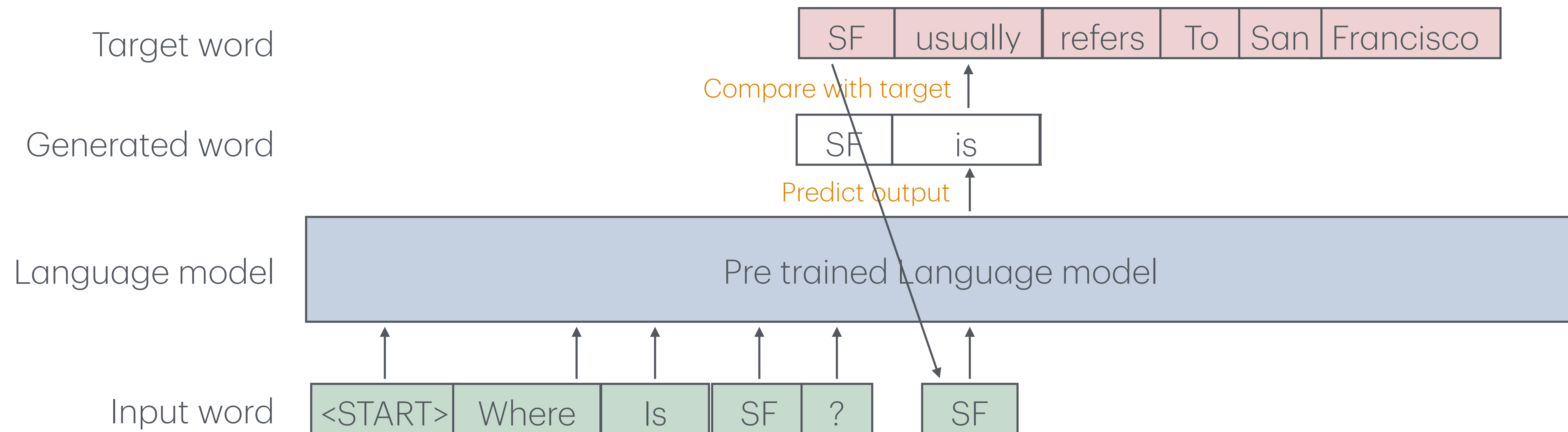


Post-training LM using SFT

Supervised fine-tuning (or instruction fine-tuning) training trains the model to generate output that matches the ground true answer.

Q: where is SF?

A: SF usually refers to San Francisco, a city in California, USA

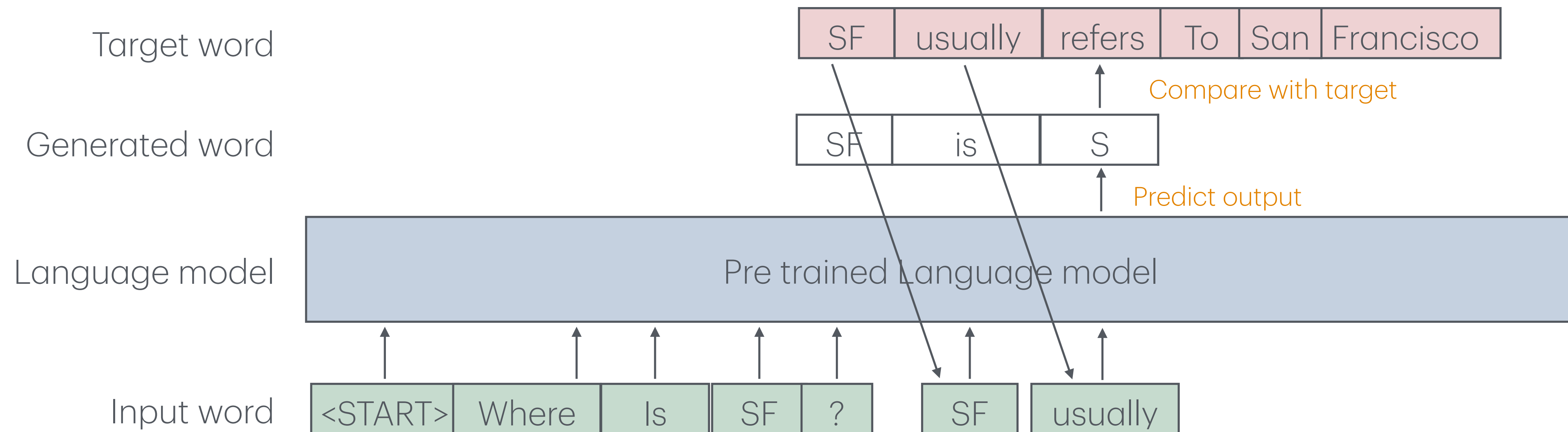


Post-training LM using SFT

Supervised fine-tuning (or instruction fine-tuning) training trains the model to generate output that matches the ground true answer.

Q: where is SF?

A: SF usually refers to San Francisco, a city in California, USA

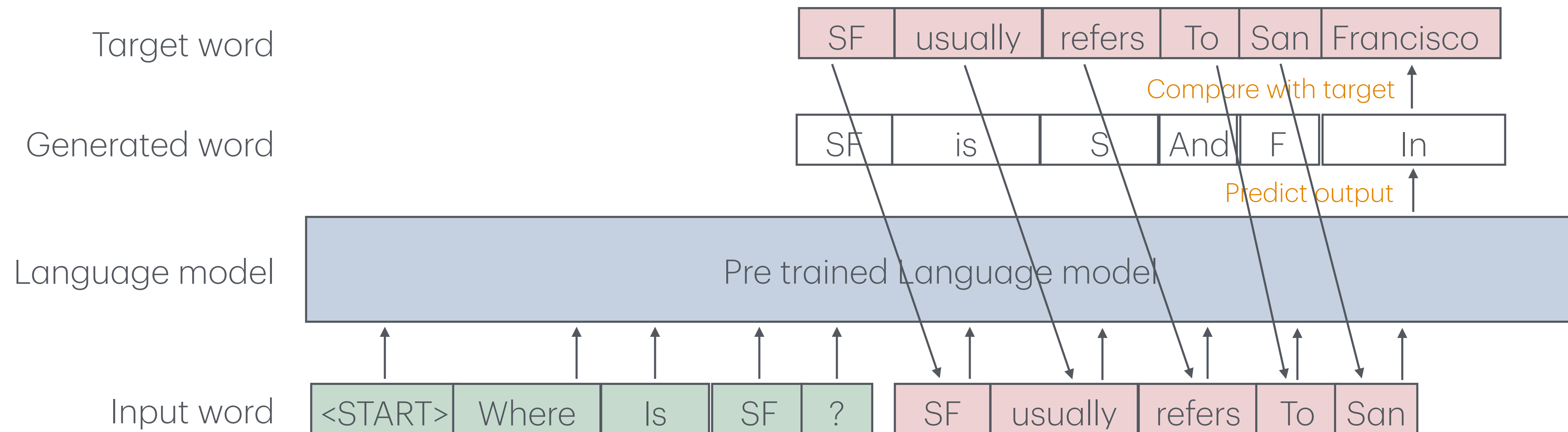


Post training LM using SFT

Supervised fine-tuning (or instruction fine-tuning) training trains the model to generate output that matches the ground true answer.

Q: where is SF?

A: SF usually refers to San Francisco, a city in California, USA



Summary: Post-training LM using SFT

- Improves performance of language model's instruction following ability
- Works best when using pre-trained knowledge, not adding new behavior/knowledge
- Limitations:
 - Model may not be good at open-ended tasks. Since model is trained with *next token prediction*, it's generality is limited by dataset used
 - Model may *hallucinate*. Due to a mismatch between its knowledge and the annotated data

Post training LM using RL

More details will be described in the followings slides

RL raining trains the model to maximize the objective based on reward 🏆.

For example, **A1 answer** is better than **A2 answer**.

- Q: where is SF?

- A1 (👍) : SF usually refers to San Francisco, a city in California, USA

- A2 (👎): SF is in CA, USA

Instead of next token based objective, RL nudges the model to generate answers with higher reward 🏆

Generated word

SF	is	n	CA	,	USA
----	----	---	----	---	-----

← Provide 👎reward

Language model

Pre trained Language model

Input word

<START>	Where	Is	SF	?
---------	-------	----	----	---

Prefill prompt

Post training LM using RL

More details will be described in the followings slides

RL raining trains the model to maximize the objective based on reward 🏆.

For example, **A1 answer** is better than **A2 answer**.

- Q: where is SF?
- A1 (👍) : SF usually refers to San Francisco, a city in California, USA
- A2 (👎): SF is in CA, USA

Instead of next token based objective, RL nudges the model to generate answers with higher reward 🏆

Generated word

SF	usually	means	San	Fran	sisco
----	---------	-------	-----	------	-------

← Provide 👍 **reward**

Language model

Pre trained Language model

Input word

<START>	Where	Is	SF	?
---------	-------	----	----	---

↑ Prefill prompt

Outline

1. Language model training

2. Reinforcement Learning

3. Reinforcement learning from human preferences (RLHF)

4. Group Relative Policy Optimization (GRPO)

5. Direct Preference Optimization (DPO)

Reinforcement Learning: Abridged History

- The field of reinforcement learning (RL) has studied these (and related) problems for many years now [Williams, 1992; Sutton and Barto, 1998]
- Circa 2013: resurgence of interest in RL applied to deep learning, game-playing [Mnih et al., 2013]
- But there is a renewed interest in applying RL. Why?
 - RL w/ LMs has commonly been viewed as very hard to get right (still is!)
 - We have found successful RL variants that work for language (e.g., PPO; [Schulman et al., 2017])

RL: problem description

- An agent: interacts with an environment by taking actions, a_t
- State, s_t : at any step t , the agent exists in state s_t
- Policy, p_θ : the agent uses a policy function (p_θ) to choose an action (a_t) at a given state
- Reward, r_t : the environment returns a reward (r_t) for the action (a_t) and takes the agent to the new state s_{t+1}
- To solve RL, we need i) reward function, and ii) policy function



s_t : state

r_t : reward

a_t : actions

p_θ : policy

LM as RL problem

- State, s_t : context/prompt
- Action, a_t : completion to prompt
- Policy, p_θ : Language model predicts next token conditioned on states
- Reward, r_t : the environment returns a reward (r_t) for the action (a_t) and takes the agent to the new state s_{t+1}
- To solve RL, we need i) reward function, and ii) policy function



s_t : state (context)

r_t : reward

a_t : actions (next sentence)

p_θ : policy (LM)

Outline

1. Language model training
2. Reinforcement Learning
- 3. Reinforcement learning from human preferences (RLHF)**
4. Group Relative Policy Optimization (GRPO)
5. Direct Preference Optimization (DPO)

RLHF

- A reward function, $R(a_t, s_t) \in \mathbb{R}$ for any output a_t to a prompt, s_t
- The reward is higher when human prefer the output
- Good generation is equivalent to the output that maximizes expected reward

We need:

1. Find best LM, p_θ that maximizes expected reward (optimization problem)
2. Reward function: $R(s_t, a_t)$

Expected reward when outputs a are drawn from the distribution, p_θ

$$\mathbb{E}_{\hat{a} \sim p_\theta} [R(\hat{a}, s)]$$

\mathbb{E} : empirical expectation (i.e., average)
 \sim : sampling from a given distribution

Policy p_θ is LM parameterized θ and assigns probabilities to text a .

Optimizing policy function

- How do we change the LM parameter θ to maximize $\mathbb{E}_{\hat{a} \sim p_\theta} [R(\hat{a}, s)]$?
- Gradient ascent:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \mathbb{E}_{\hat{a} \sim p_\theta} [R(\hat{a}, s)]$$

How do we estimate
the gradient of this

R is not depends on θ , so we
cannot compute its grad wrt θ

- Policy gradient: an approach for estimating an optimizing this objective

Policy gradient/REINFORCE

- Compute the gradient

$$\nabla_{\theta} \mathbb{E}_{\hat{a} \sim p_{\theta}}[R(a, s)] = \nabla_{\theta} \sum_a p_{\theta}(a) R(a, s) = \sum_a R(a, s) \cdot \nabla_{\theta} p_{\theta}(a)$$

Def. Of expectation

Gradient distributes over sum

- Use log-derivative trick $\nabla_{\theta} p_{\theta}(a) = p_{\theta}(a) \cdot \nabla_{\theta} \log p_{\theta}(a)$

Log-derivative trick

$$\nabla_{\theta} \mathbb{E}_{\hat{a} \sim p_{\theta}}[R(a, s)] = \sum_a R(a, s) p_{\theta}(a) \nabla_{\theta} \log p_{\theta}(a) = \mathbb{E}_{a \sim p_{\theta}}[R(a, s) \nabla_{\theta} \log p_{\theta}(a)]$$

- We can approximate this expectation with Monte Carlo samples

$$\nabla_{\theta} \mathbb{E}_{a \sim p_{\theta}}[R(a, s)] \approx \frac{1}{n} \sum_{i=1}^n R(a, s) \nabla_{\theta} \log p_{\theta}(a)$$

Policy gradient/REINFORCE

- This is the update rule

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} \mathbb{E}_{\hat{a} \sim p_{\theta}} [R(\hat{a}, s)]$$

$$\nabla_{\theta} \mathbb{E}_{a \sim p_{\theta}} [R(a, s)] \approx \frac{1}{n} \sum_{i=1}^n R(a, s) \nabla_{\theta} \log p_{\theta}(a)$$

- If $R(a, s)$ is **large**, take **large** steps to maximize $p_{\theta}(a)$
- If $R(a, s)$ is **small**, take **small** steps to maximize $p_{\theta}(a)$
- We reinforce good actions, increasing the chance they happen again.

Reward with human preferences

- Human annotation is expensive 💰
- We can use a reward model to learn human preferences [[Knox and Stone, 2009](#)]
- ❌ Approach 1: ask humans to provide scores for each output
 - ⚠️ it will be noisy and hard to calibrate
- ✅ Approach 2: pairwise comparisons. This can be more reliable [[Phelps et al., 2015](#); [Clark et al., 2018](#)]

Winning output should score higher than losing output

$$J_{RM}(\phi) = - \mathbb{E}_{(a^w, a^l) \sim D} \left[\log \sigma \left(RM_{\phi}(a^w) - RM_{\phi}(a^l) \right) \right]$$

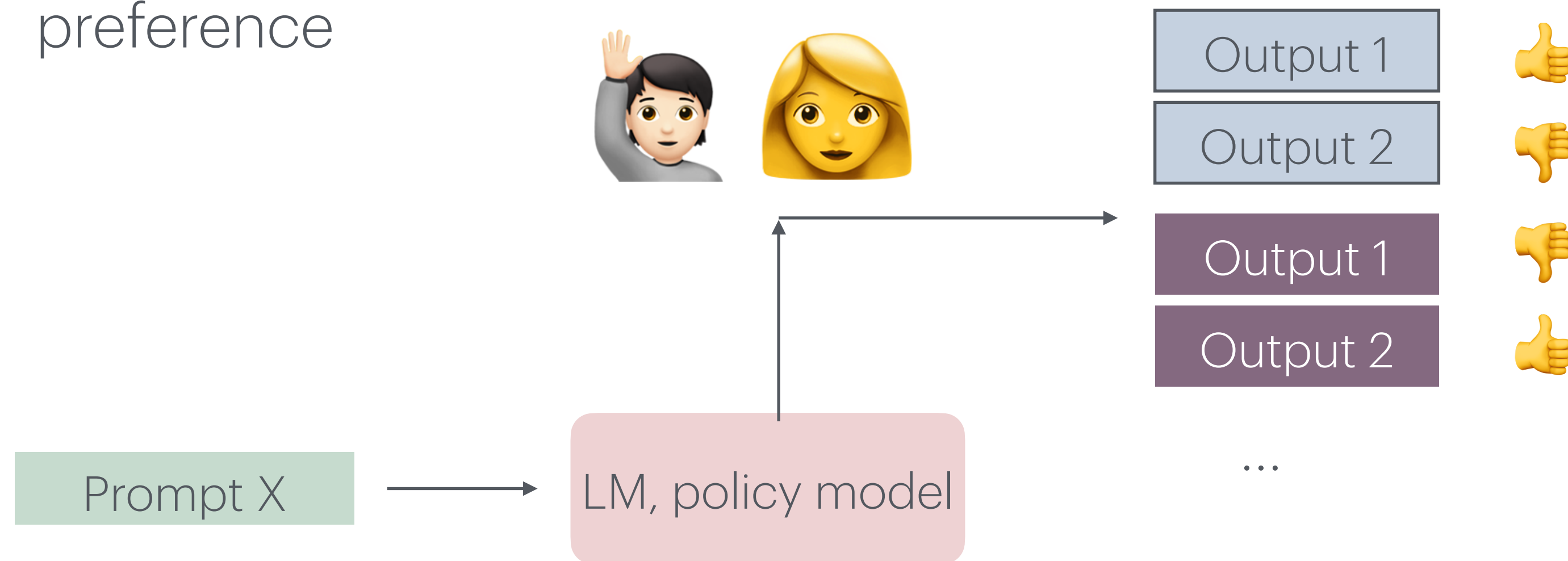
Bradley-Terry [1952] paired comparison model

Winning output

Losing output

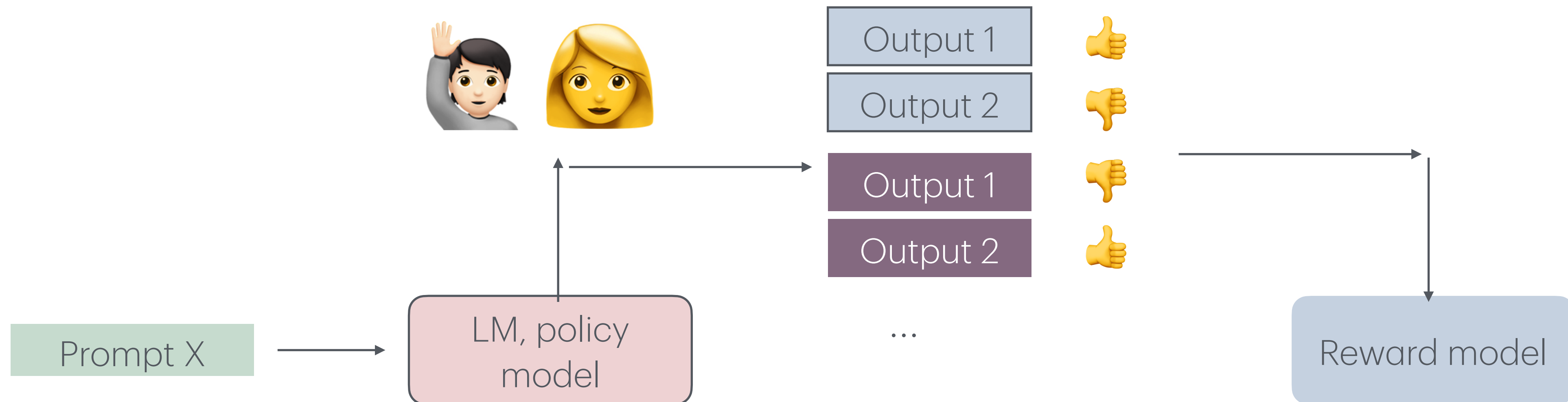
Putting it together

- Collect a dataset with human preferences
- Present multiple outputs to human annotators and ask them to rank the output based on preference



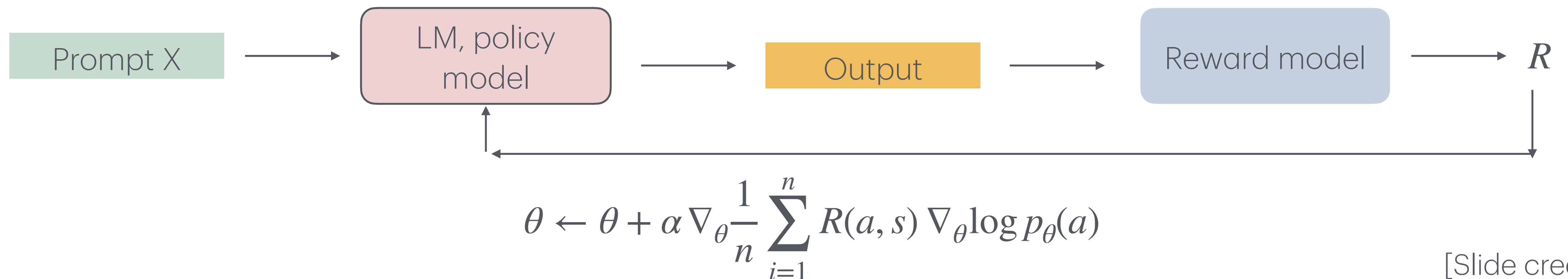
Putting it together

- Train the reward model using the preference data
 - The reward model returns a scalar reward to represent human preference



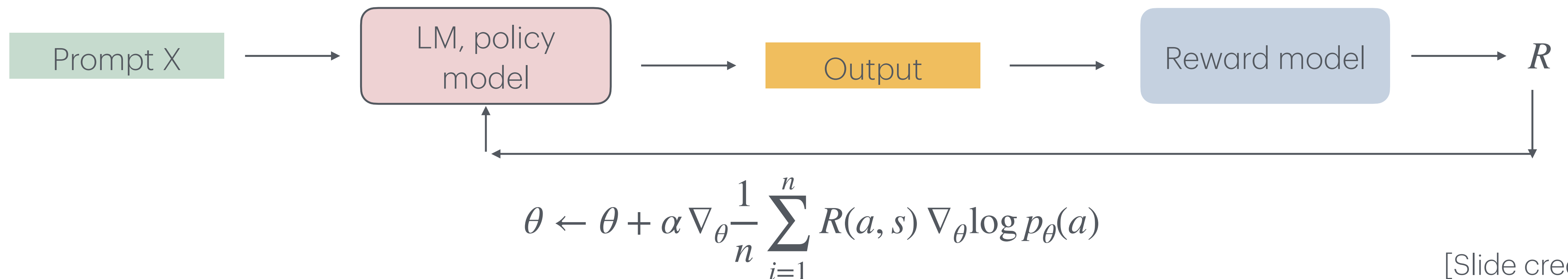
Putting it together

- Learn a policy (LM) that optimizes against the reward model



One missing ingredient

- The policy gradient optimizer seeks to maximize its reward, it will learn to cheat. (i.e. reward hacking)
- It learns to gain high reward while producing gibberish
- The reward model is trained on natural language inputs and might fail to assign low scores to unnatural input.



Regularizing with pre-trained model

- modify the reward with a regularization term that penalizes outputs that deviate from natural language.
 - i.e add penalty term that penalizes too much deviations from the distribution of the pre-trained LM

$$\hat{R}(a, s) := R(a, s) - \beta \log \left(\frac{p^{RL}(a)}{p^{PT}(a)} \right)$$

- This prevents the policy model from diverting too far from the pre-trained model
 - $p^{RL}(a) \gg p^{PT}(a)$: pay an explicit price
 - $p^{RL}(a) \ll p^{Ref}(a)$: sampling a becomes unlikely

Proximal Policy Optimization (PPO)

- Currently, most popular approach, [[Schulman et al., 2017](#)]
- It is designed to be conservative in its updates and makes training more stable
- advantage actor-critic method
 - Actor-critic: the learning objective includes an estimated value function to “critique” the policy (actor) actions.
 - Advantage: instead of optimizing directly using rewards like REINFORCE, updates rely on “advantage”.

Outline

1. Language model training
2. Reinforcement Learning
3. Reinforcement learning from human preferences (RLHF)
- 4. Group Relative Policy Optimization (GRPO)**
5. Direct Preference Optimization (DPO)

Group Relative Policy Optimization (GRPO)

- PPO uses large memory for training
 - Uses 4 LMs: reward, value, policy, and reference policy
- GRPO skips the value model, reduces memory usage
- Value function used for estimate advantage in PPO
 - GRPO proposed alternate way to estimate advantage
 - Use different samples from the single prompt to estimate the advantage

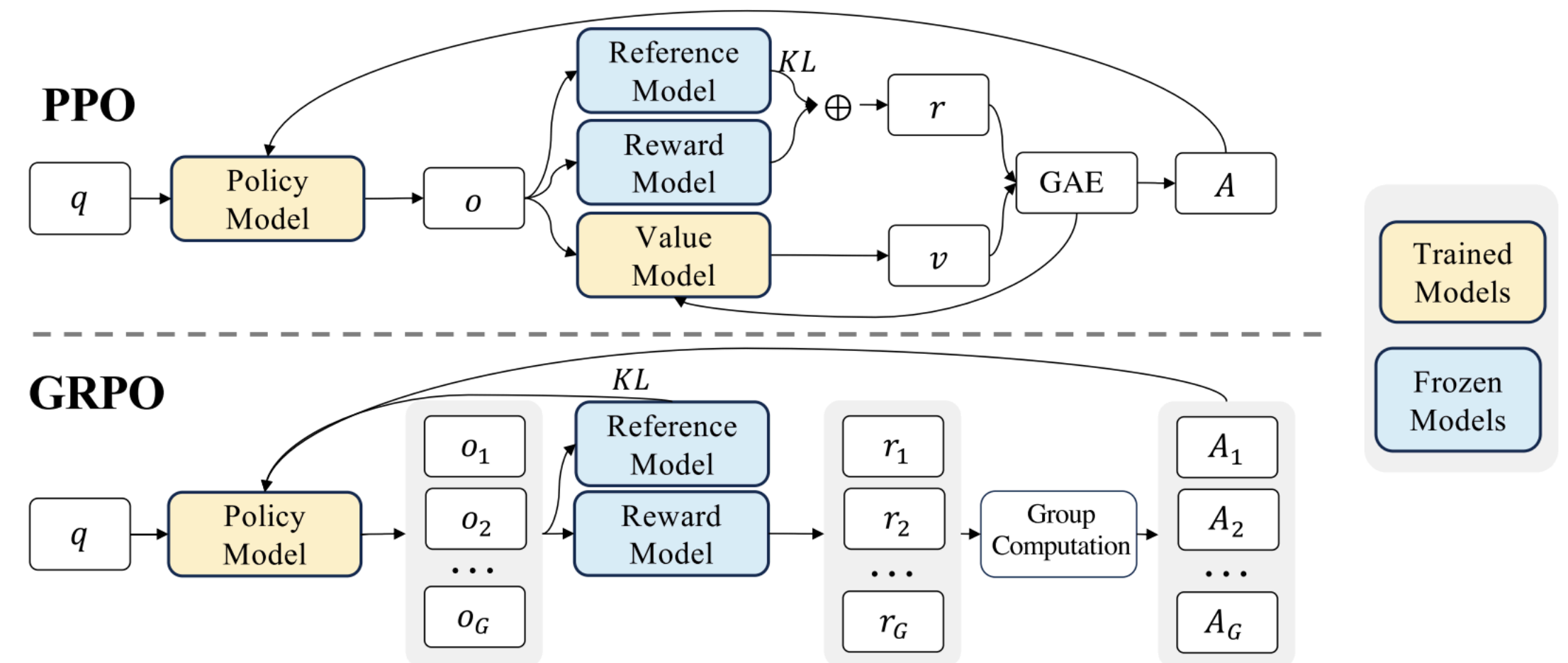


Figure 4 | Demonstration of PPO and our GRPO. GRPO foregoes the value model, instead estimating the baseline from group scores, significantly reducing training resources.

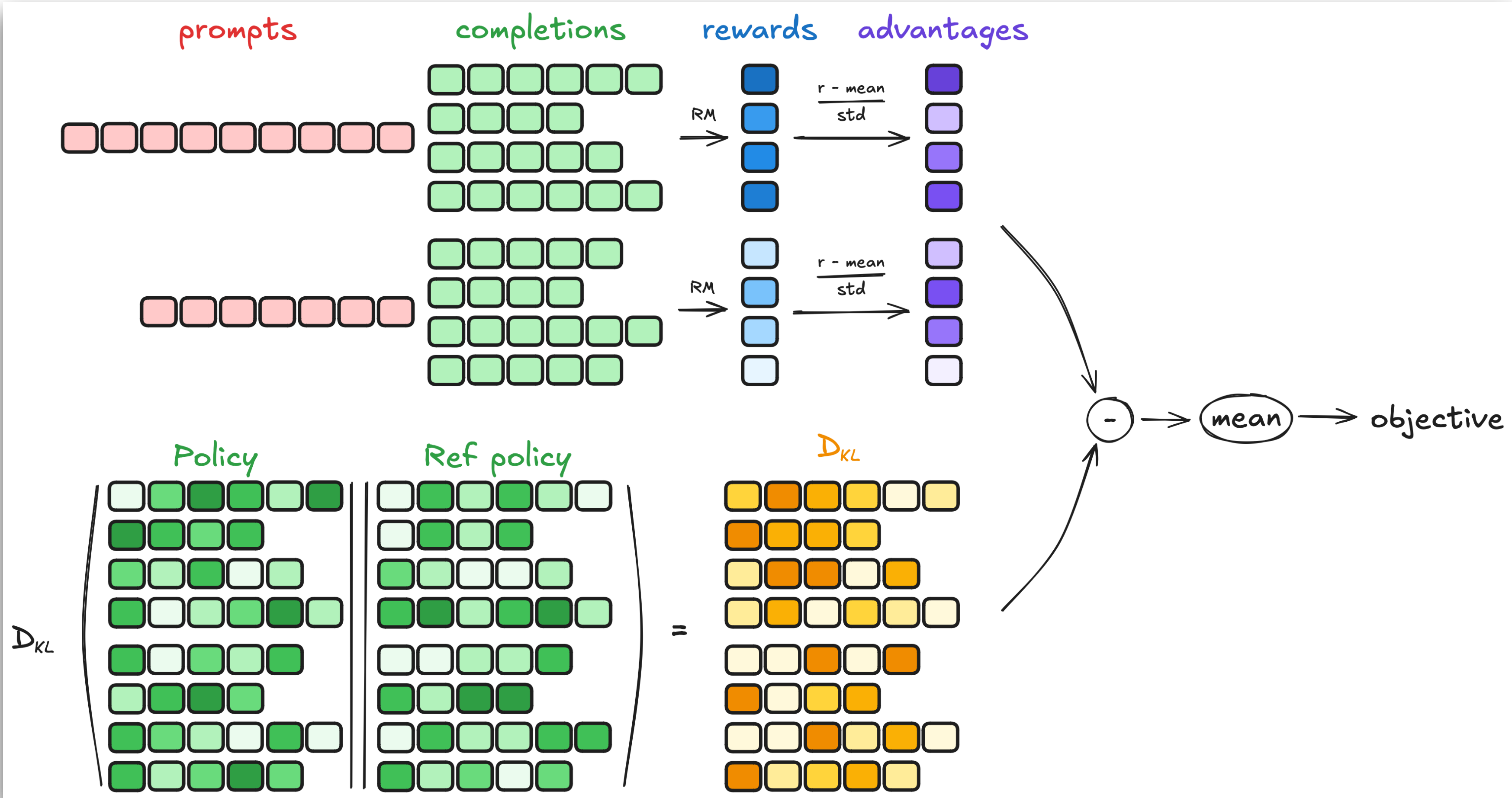
GRPO

- DeepSeek uses rule-based reward for math and coding tasks.
- Execute multiple rollouts
- From these rollouts, estimate advantage function based on relative goodness of these responses

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

- Advantage of each rollout is gap between its reward wrt the mean reward of the response divided by std.

GRPO



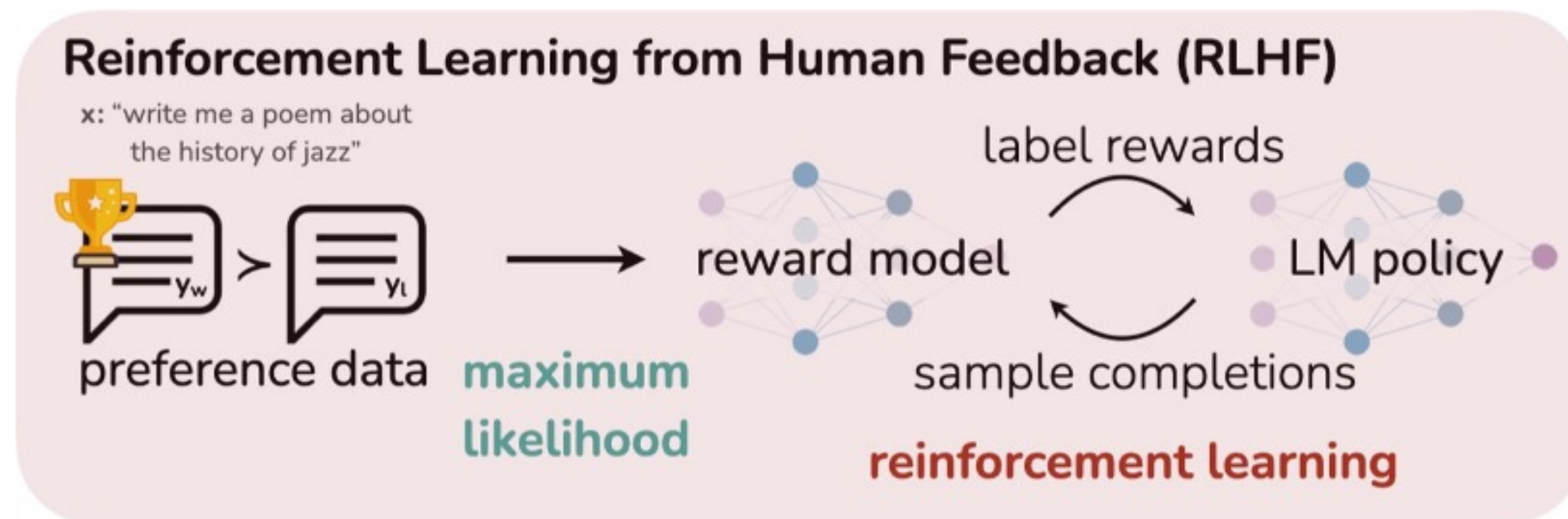
[Slide credit: ref. 2, https://huggingface.co/docs/trl/main/en/grpo_trainer]

Outline

1. Language model training
2. Reinforcement Learning
3. Reinforcement learning from human preferences (RLHF)
4. Group Relative Policy Optimization (GRPO)
- 5. Direct Preference Optimization (DPO)**

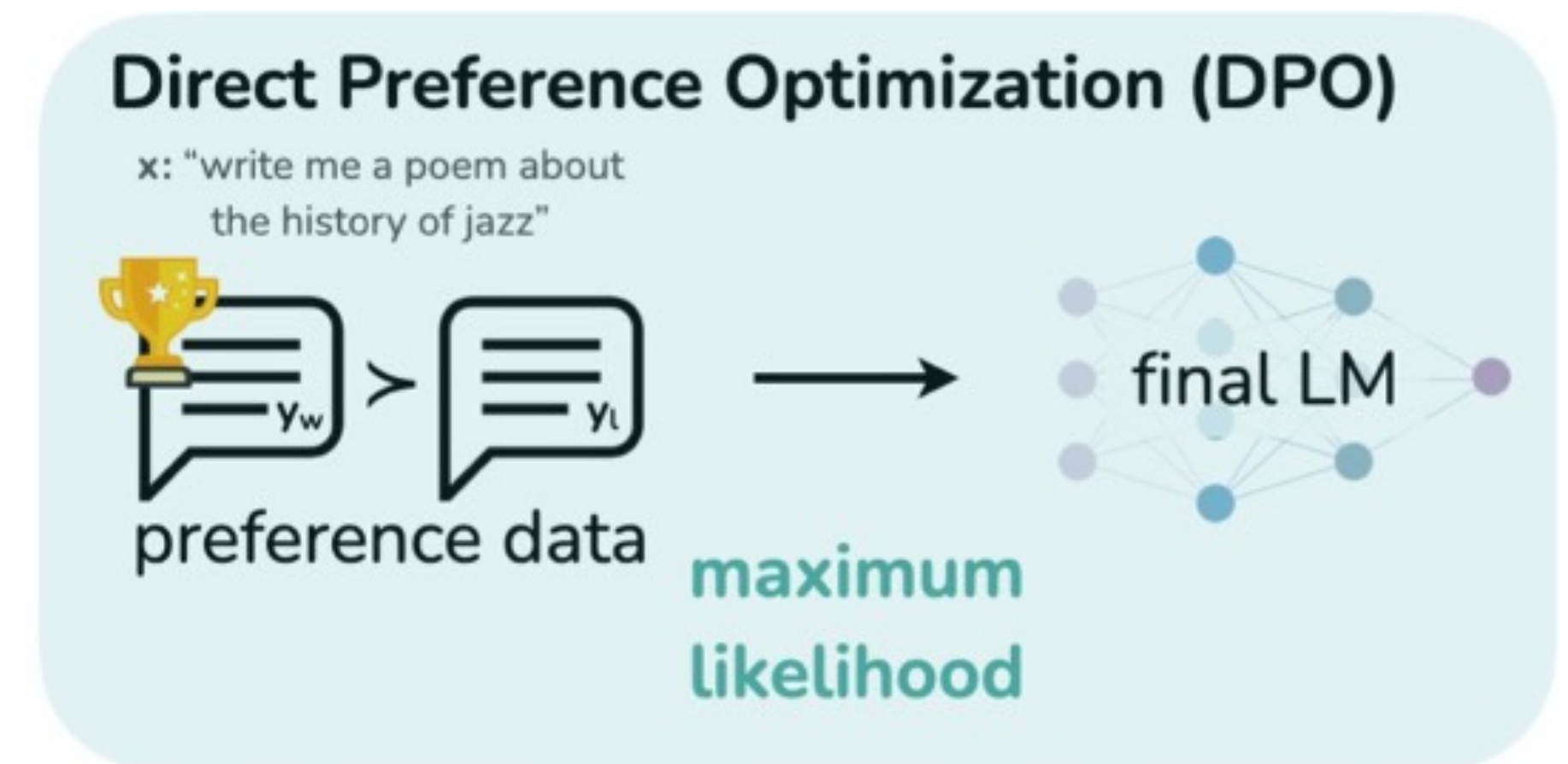
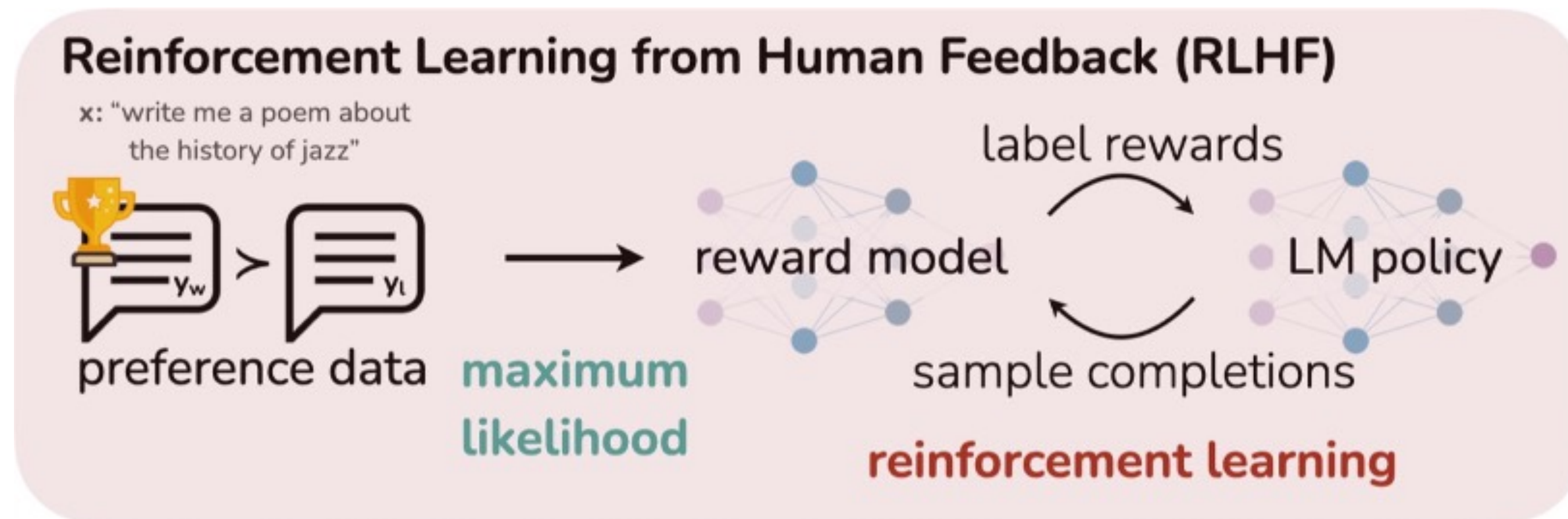
DPO: motivation

- The RLHF pipeline is considerably more complex than supervised learning
 - Involves training multiple LMs and sampling from the LM policy in the loop of training
- Is there a way to simplify this pipeline?
 - For example, by using a single language model



DPO

- DPO directly optimizes for human preferences
 - Removing RL and fitting a separate reward model
- One can use mathematical derivations to simplify the RLHF objective to an equivalent objective that is simpler to optimize.



DPO

RLHF objectives

1. Reward objective: $\mathcal{L}_R(r_\phi, \mathcal{D}) = - \mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[\log \sigma \left(r_\phi(x, y_w) - r_\phi(x, y_l) \right) \right]$

2. Policy objective: $\max_{\pi_\theta} \mathbb{E}_{x \sim \mathcal{D}, y \sim \pi_\theta(y|x)} \left[r_\phi(x, y) \right] - \beta \mathbb{D}_{\text{KL}} \left[\pi_\theta(y | x) \parallel \pi_{\text{ref}}(y | x) \right]$

Maximizing the reward of the generated output

Minimizing the deviation from the base policy

DPO objectives

1) maximizing reward of the winning response vs that of losing one; 2)
minimizing deviations from the base policy

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

y_w : winning response, y_l : losing response

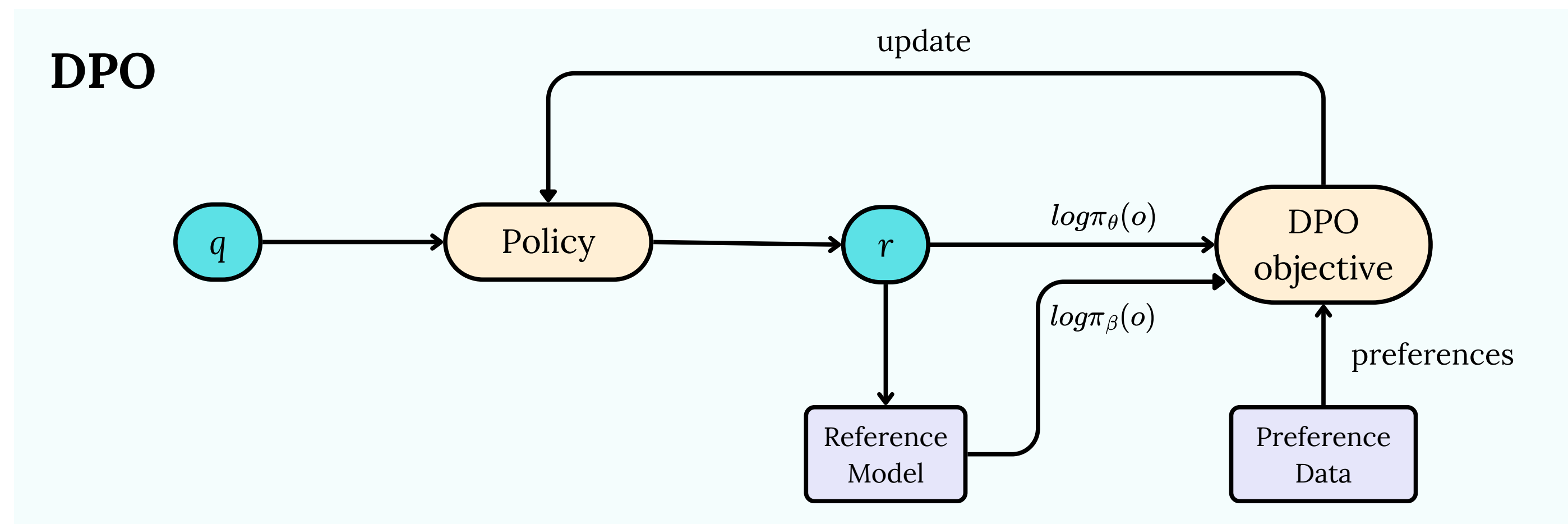
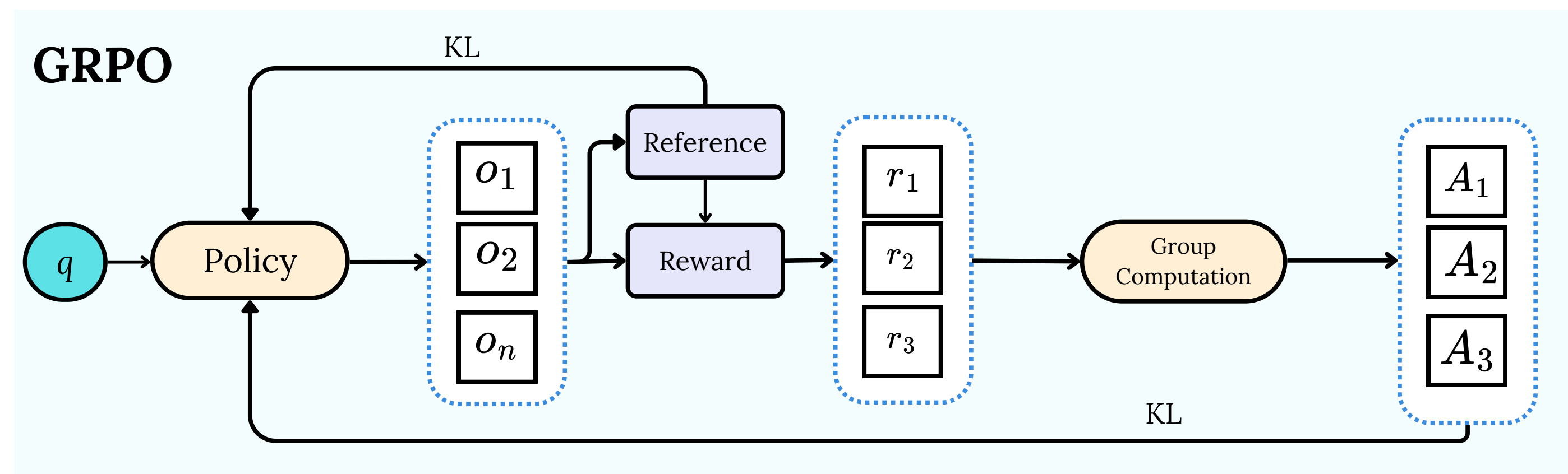
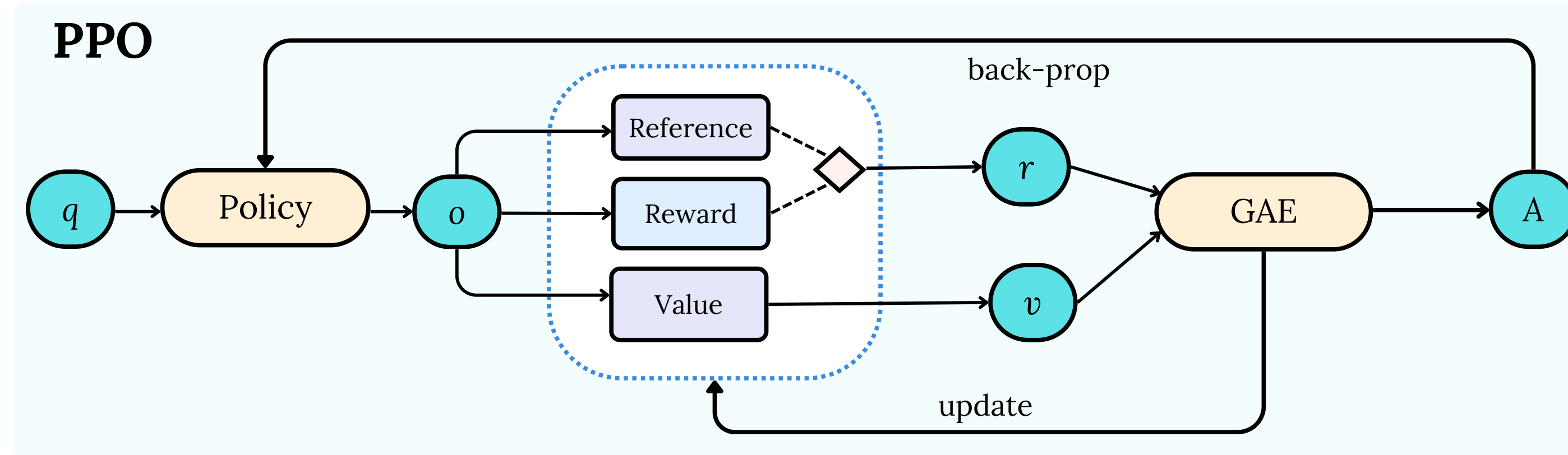
Summary: DPO

- Optimize the reward using supervised losses (which in turn, optimizes the policy)
- RLHF with out RL part
- We need preference dataset with x and y_w (winning response), and y_l (losing response)
- Widely used
- Recent example from model for GUI agent, “UI-TARS: Pioneering Automated GUI Interaction with Native Agents”

Agent DPO During online bootstrapping, a large number of erroneous steps (negative examples) are naturally generated. However, SFT only utilizes the corrected steps (i.e., “positive” examples), while ignoring the negative samples, which limits its ability to explicitly guide the agent away from suboptimal actions. To address this limitation, we turn to Direct Preference Optimization (DPO) (Rafailov et al., 2023), which leverages both the corrected and erroneous actions by introducing a reference-based objective. This approach optimizes UI-TARS by directly encoding a preference for corrected actions over erroneous ones, thereby making better use of the available data.

Summary

- Motivation for RL training is to overcome some of the issues with SFT
- RLHF uses reward model to capture human preferences
- For some domains (math and coding), rule-based reward can be used



References

1. [Instruction Finetuning, and RLHF, originally by Jesse Mu](#)
2. [Alignment of LMs, Daniel Khashabi](#)
3. [Aligning LLMs to preferences, Vivek Srikumar](#)
4. [Reinforcement Learning from Human Feedback, Nathan Lambert](#)

Backup

Example of RL trained model

- OpenAI deep research: searches requested topics and provide through analysis and summary
- Give model to access browser and code execution environment
- Create RL datasets that browsing and code execution

