

# LLM: fine-tuning 2

Natural Language Processing

Ника Зыкова, 2025/12/01

# Недостатки SFT

- ❖ Оптимизирует неправильную целевую функцию: NTP не задает явно стиль, полезность и “человечность” ответов;
- ❖ Иногда задачу невозможно сформулировать так, чтобы удобно дообучать через SFT;
- ❖ Не учитывает предпочтения пользователей: нет разницы между двумя одинаково (не) правильными ответами;
- ❖ Переобучение на стиль: чаще всего в датасетах для sft сложно сохранить разнообразие;
- ❖ Модель видит только идеальные ответы;
- ❖ Нет механизма наказания за плохие ответы.

## Instruction

Проанализируй список расходов и вычисли общую сумму:

Хлеб: 55

Молоко: 80

Фрукты: 230

Ответ верни в виде: "Итого: X рублей".

## Output

Итого: 365 рублей.

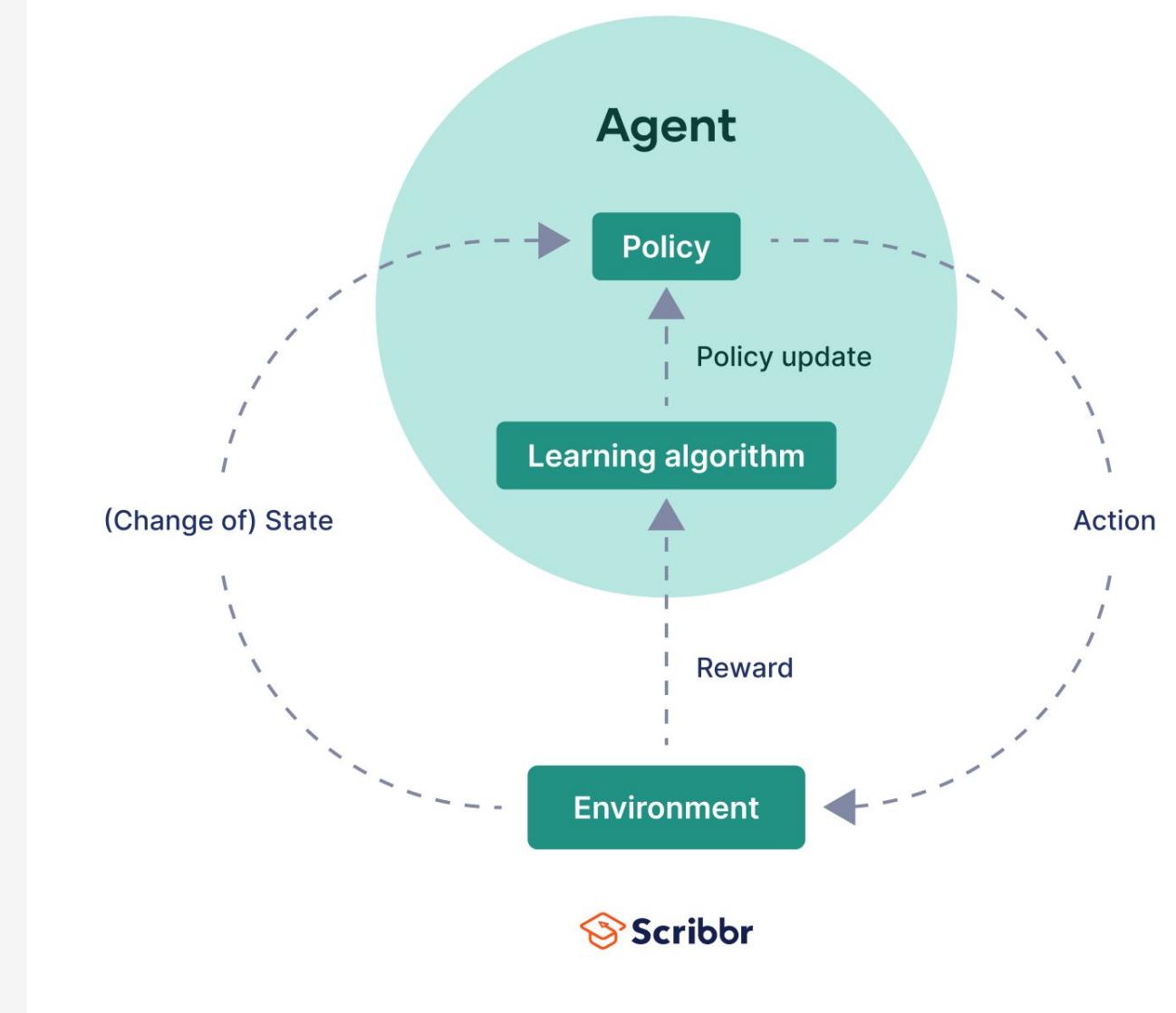
# Что еще?

Сейчас для дообучение очень активно применяется RL. При этом классический RL для LLM применять очень дорого, поэтому используются разные специальные алгоритмы:

- RJ (Rejection Sampling)
- PPO (Proximal Policy Optimization)
- DPO (Direct Preference Optimization)
- GRPO (Grouped Reinforcement Policy Optimization)

И многие другие, часто исправляющие конкретные недостатки перечисленных алгоритмов.

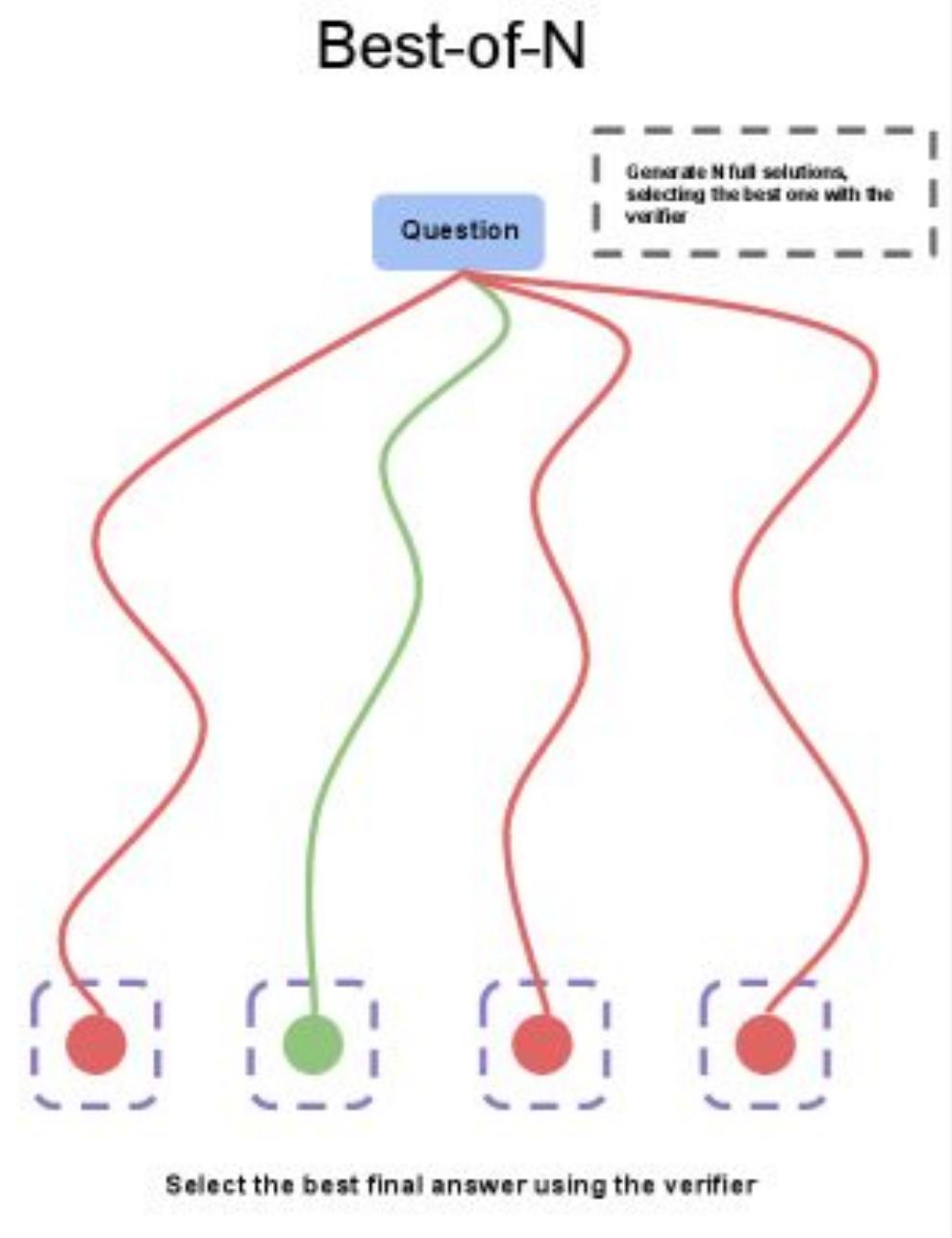
The general framework of reinforcement learning



# Rejection Sampling

1. Обучаем модель вознаграждения на размеченных парах (лучше/хуже);
2. Цикл обучения для одного примера:
  - Генерируем несколько вариантов ответа;
  - Оцениваем каждый с помощью модели вознаграждения;
  - Выбираем ответ с максимальной наградой, дообучаем модель на его генерацию.

Часто в лосс добавляют KL-дивергенцию относительно исходного распределения вероятностей, чтобы модель не развалилась в процессе обучения.



# Rejection Sampling

## Плюсы

- Технически простое решение, в итоговом виде даже не RL;
- Вычислительно быстрее большинства RL подходов (фактически, это SFT);
- Можно брать любые модели в качестве моделей вознаграждения, даже проприетарные LLM (тогда оценка делается через промпting).

## Минусы

- Надо генерировать несколько ответов на один вход;
- Если модель вознаграждения простая, то модель может переобучиться на предпочитаемые ей паттерны;
- Reward Hacking: способ оценки не обновляется в процессе обучения.

# PPO

$$r = \pi_t(a|s)/\pi_0(a|s), 1 - \varepsilon \leq r \leq 1 + \varepsilon$$

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

Генерируем несколько выходов, оцениваем для них средний ревард и то, насколько его прирост соотносится с увеличением вероятности токенов.

$\pi_0$  — старая политика (SFT модель)

$\pi^{\square}$  — политика, которую мы тренируем

а — сгенерированный токен/ответ

А — advantage (насколько ответ лучше среднего, среднее оценивает отдельная модель, которая обучается в процессе)

Плюс здесь также добавляется KL-регуляризация.

# RPO

## Плюсы

- Стабильность: clip позволяет избегать взрывов градиента и политики;
- Можно использовать любой ревард;
- Проще классических RL-подходов;
- Хорошо масштабируется.

## Минусы

- Очень долго: много генераций, модель-оценщик, несколько проходов RPO на каждом шаге;
- Сильно зависит от оценщика;
- Все еще недостаточно устойчиво;
- Reward Hacking: способ оценки не обновляется в процессе обучения;
- Требует сложной инженерии данных;
- Не подходит для обучения ризонингу и агентским навыкам.

# DPO

1. Обучаем модель вознаграждения на размеченных парах (лучше/хуже);
2. Обучаем модель на парах в стандартном supervised подходе, но со специфическим лоссом:

$$\mathcal{L}_R(r_\phi, \mathcal{D}) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} [\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))]$$

В чем идея: давайте вместо использования оценщика будем учить модель напрямую увеличивать вероятность более хороших ответов и уменьшать - более плохих. Здесь особенно важно, чтобы модель не развалилась в процессе, так что тоже используем KL:

$$D_{KL}(p||q) = \int_x p(x) \log \frac{p(x)}{q(x)} dx$$

# DPO

## Плюсы

- Не нужна модель-оценщик;
- Стабильность: не RL, поэтому нет специфических для него проблем;
- Простота реализации: просто supervised дообучение, надо всего лишь правильно записать лосс.

## Минусы

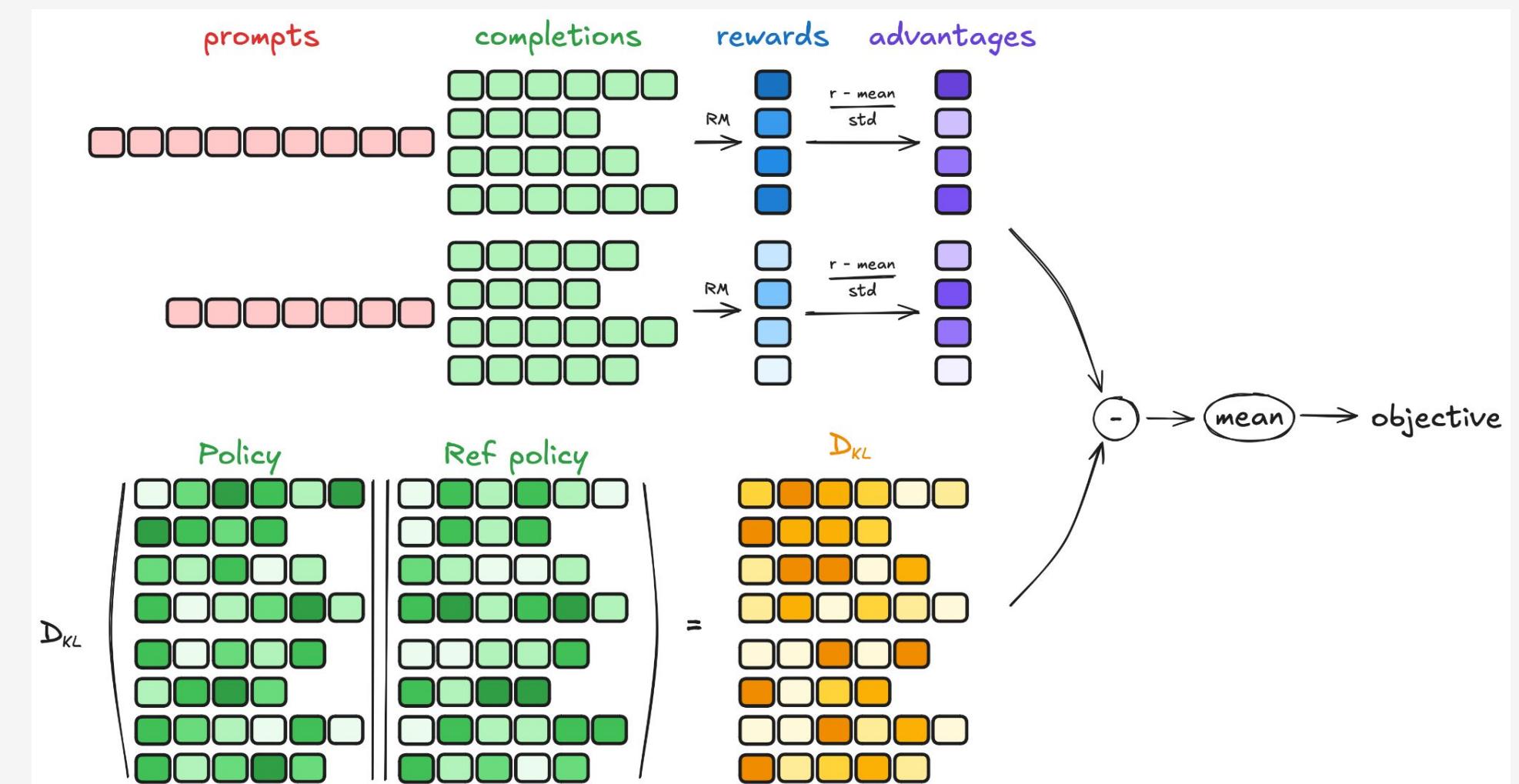
- Ограниченнная гибкость: при большом кол-ве итерация нужна новая разметка предпочтений;
- Резко дестабилизируется при длинном обучении на маленьких данных (когда много эпох);
- Модель склонна переобучаться.

# GRPO

Похоже на PPO, но вместо оценщика для преимущества используем разницу со

$$\mathcal{L}_{\text{GRPO}}(\theta) = -\frac{1}{\sum_{i=1}^G |o_i|} \sum_{i=1}^G \sum_{t=1}^{|o_i|} \left[ \frac{\pi_\theta(o_{i,t} | q, o_{i,<t})}{[\pi_\theta(o_{i,t} | q, o_{i,<t})]_{\text{no grad}}} \hat{A}_{i,t} - \beta \mathbb{D}_{\text{KL}} [\pi_\theta \| \pi_{\text{ref}}] \right]$$

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



# GRPO

## Плюсы

- Не нужна вторая модель-оценщик;
- Работает с автоматической оценкой задачи (код, решение математики);
- Нет взлома реварда при правильном использовании;
- Сильно дешевле и устойчивее PPO;
- Хорошо работает в задачах с рассуждениями.

## Минусы

- Подходит только для проверяемых задач: с лм/человеческими оценками дорого + модель легко взламывает ревард;
- Требует много генераций (обычно 4-32 ответов на промпт);
- Нужна хорошая оценка: правильные метки + хорошая инфраструктура;
- Переобучается на решения и забывает про стиль и нюансы;
- Не улучшает "soft qualities" модели (дружелюбие, стиль и т.д).

# Сравнение

Метод	Требования к данным	Вычислительная стоимость	Сложность	Стабильность	Где лучше всего подходит
SFT	Готовые хорошие ответы	+	+	++++	Базовый уровень, стиль
Rejection Sampling (RS)	RM или critic, много промптов	++++	++	++++	Reasoning, coding, math
PPO (RLHF)	RM + critic + большой датасет	+++++	+++++	++	Helpfulness, safety, сложные предпочтения
DPO	Пары good/bad	++	+++	++++	Глобальные предпочтения, стиль
GRPO	RM, группы ответов (много генерации)	++++	+++	+++	Математика, код, рассуждения