Post-training LLMs: From Algorithms to Training Infrastructures

Presenter: Renjie Zheng, Zheng Wu, Haibin Lin

ByteDance

NEURAL INFORMATION PROCESSING SYSTEMS

Agenda:

Post-training Algorithms

- **SFT, PPO, DPO, DQO**
- PRM for Code Generation

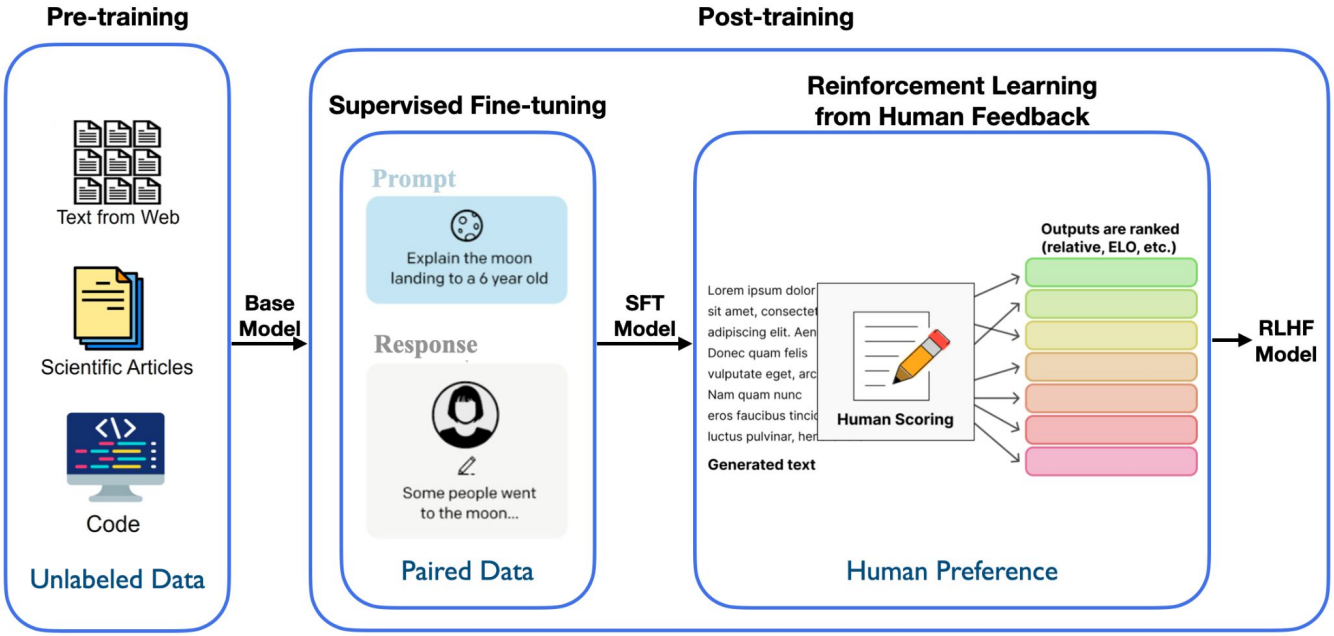
Infrastructure and frameworks

- Programming model, veRL
- Code demo

code repo and
talk materials

What is Post-training?

Current standard pipeline for training LLMs



Training language models to follow instructions with human feedback; Ouyang et al., 2022

code repo and
talk materials

Why do we need post-training?

Pre-trained Models:

- Primarily trained on vast web data with next-token prediction.
- Possess general-purpose language abilities but not easy to use.

Post-Training Enhances User-Friendliness:

- **Improves Instruction Following**: Tailors the model to better understand and execute user commands.
- **Reduces Harmful Outputs**: Refines the model to minimize the generation of biased, toxic, or unsafe content.
- **Aligns with User Intent and Preferences**: Personalizes responses to reflect user expectations, tone, and context.
- …

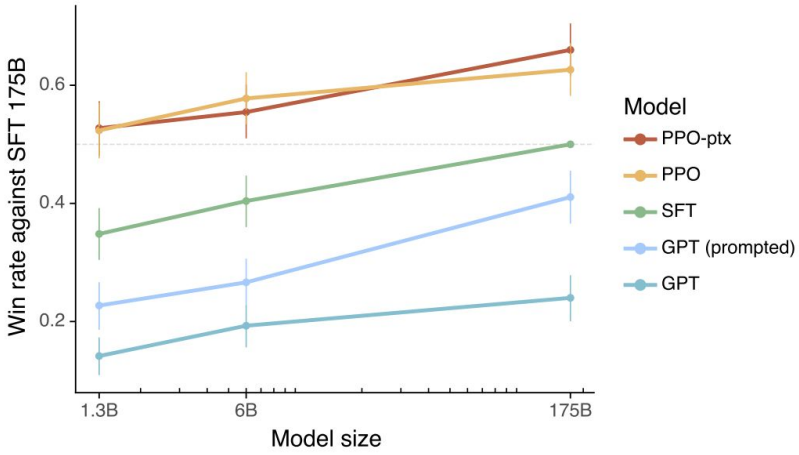
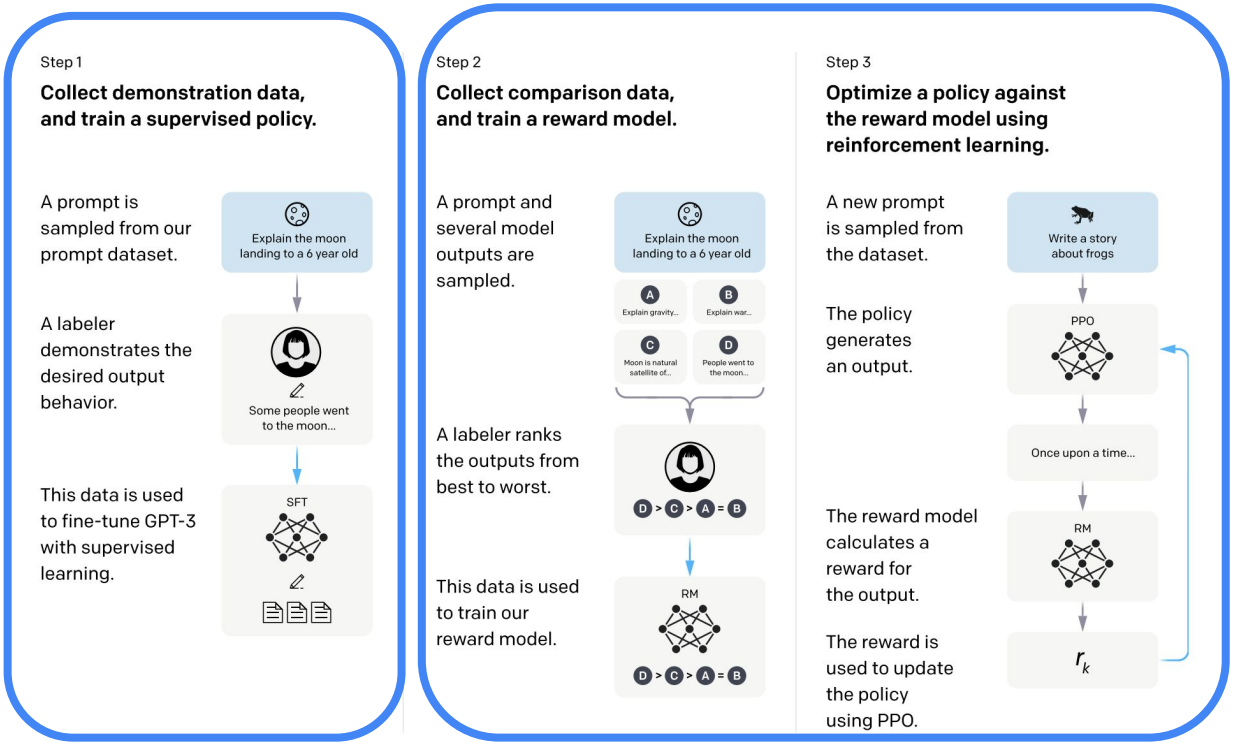


Figure from [Ouyang et al. 2022]

code repo and
talk materials

Post-training Algorithms

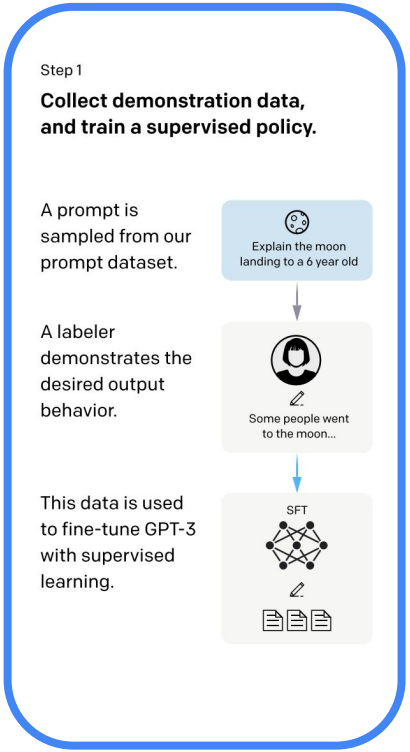


SFT RLHF

Figure from [Ouyang et al. 2022]

code repo and
talk materials

Supervised Fine-tuning (SFT)



SFT

Prompt	Here is some data about a restaurant: name = Aromi, eatType = coffee shop, food = English, customer rating = 5 out of 5, area = city centre. Write a sentence that includes the following data about a restaurant
Response	In the city centre there is a coffee shop with a customer rating of 5 out of 5 called Aromi which serves English food.

Data Example from FLAN

- **Data**
  - **Data size:** a small dataset is sufficient
  - **Data source**: human annotated or generated from models (reject sampling)
- **Training**
  - Gradient descent

code repo and
talk materials

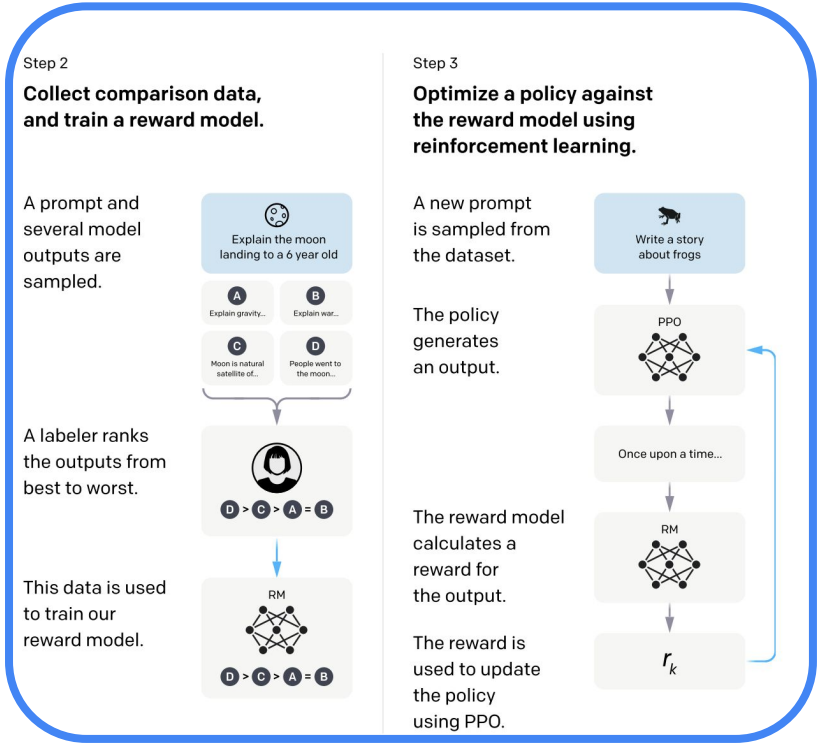
RLHF

Differences between SFT & RLHF

- **SFT**
  - Imitation Learning
- **RLHF**
  - Optimization

How RLHF works

- Reward Model data collection
- Reward Model training
- Reinforcement Learning training



RLHF

LIMA: Less Is More for Alignment; Zhou et al., 2023.
Scaling Laws for Reward Model Overoptimization; Gao et al., 2022

code repo and
talk materials

Why We Need Reinforcement Learning?

- RL can efficiently further improve the model performance after SFT
  - RM data collection is cheaper
  - Scaling RLHF is more efficient
  - ……

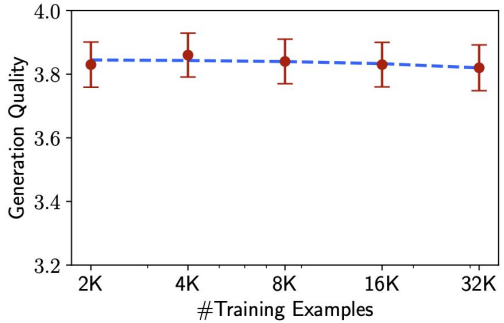
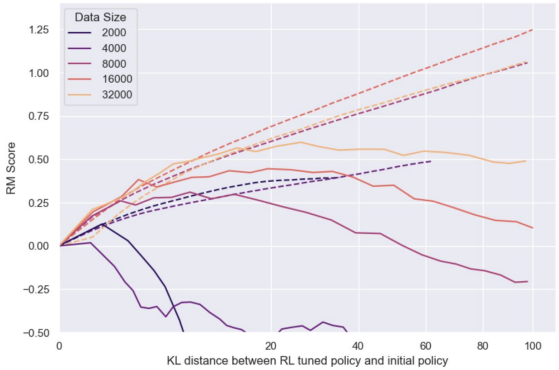


Figure 6: Performance of 7B models trained with exponentially increasing amounts of data, sampled from (quality-filtered) Stack Exchange. Despite an up to 16-fold increase in data size, performance as measured by ChatGPT plateaus.

[From Zhou et al. 2023]



(b) RL

[From Gao et al. 2023]

code repo and
talk materials

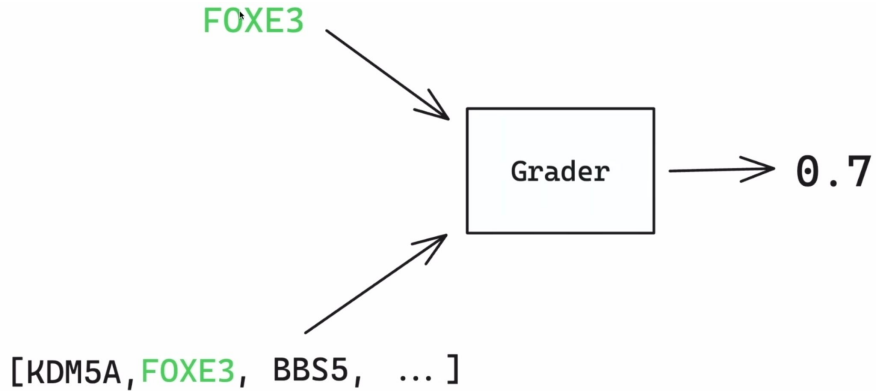
Reward Signals

- RL from **Human** Feedback
  - Use **human preference** data to train a reward model.
- RL from **Automated** Feedback
  - Coding: Results validated through unit tests.
  - Math: Solutions checked against ground-truth answers.
  - …

Test results

test	verdict	time	
#1	ACCEPTED	0.01 s	»
#2	ACCEPTED	0.01 s	»
#3	ACCEPTED	0.01 s	»
#4	ACCEPTED	0.01 s	»
#5	ACCEPTED	0.01 s	»
#6	ACCEPTED	0.08 s	»
#7	ACCEPTED	0.07 s	»
#8	ACCEPTED	0.07 s	»
#9	WRONG ANSWER	0.14 s	»
#10	ACCEPTED	0.13 s	»
#11	ACCEPTED	0.01 s	»

Unit Tests for Coding Task



Grader in OpenAI Reinforcement Fine-tuning

Proximal Policy Optimization Algorithms; Schulman et al., 2017.

code repo and
talk materials

Algorithm: Proximal Policy Optimization (PPO)

Policy Gradients
Maximize expected reward

$$\nabla_\theta E_{p_\theta}[R(z)] = E_{p_\theta}[\mathrm{R(z)}\nabla_\theta \log p_\theta\ (z)]$$

Trust Region Policy Optimization (TRPO)
Introduces constraints to ensure stable updates by limiting the policy's deviation from the previous policy.

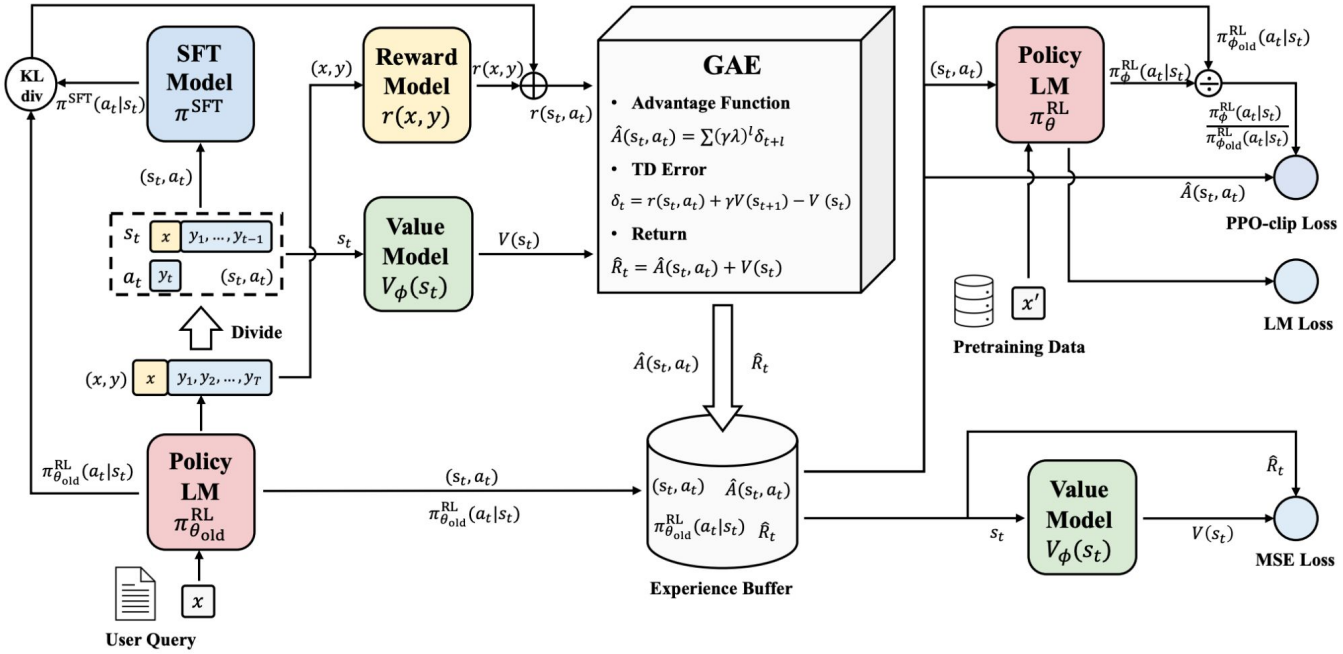
$$\begin{aligned} &\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] \\ &\text{subject to} \quad \hat{\mathbb{E}}_t[\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta. \end{aligned}$$

Proximal Policy Optimization (PPO)
Simplifies TRPO by introducing a clipping mechanism to constrain policy updates.

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a),\ \ \mathrm{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon\right) A^{\pi_{\theta_k}}(s, a)\right)$$

code repo and
talk materials

Algorithm: Proximal Policy Optimization (PPO)



[From Zheng et al. 2023]

Direct Preference Optimization: Your Language Model is Secretly a Reward Model; Rafailov et al., 2023.

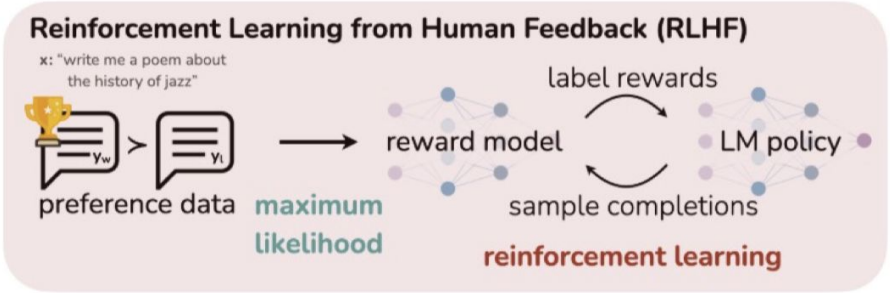
code repo and
talk materials

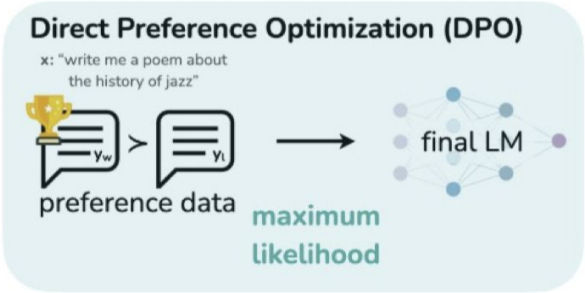
Algorithm: Direct Preference Optimization (DPO)

One limitation of PPO: **High computational requirements during training**

- rely on multiple models
- extensive online sampling

	PPO	DPO
Eliminates need for reward model	No	Yes
Free from online sampling	No	Yes





code repo and
talk materials

What Does DPO Update Do?

$$\nabla_\theta \mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) =$$
$$-\beta \mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\Big[\underbrace{\sigma(\hat{r}_\theta(x, y_l) - \hat{r}_\theta(x, y_w))}_{\text{higher weight when reward estimate is wrong}}\Big[\underbrace{\nabla_\theta \log \pi(y_w \mid x)}_{\text{increase likelihood of } y_w} - \underbrace{\nabla_\theta \log \pi(y_l \mid x)}_{\text{decrease likelihood of } y_l}\Big]\Big],$$

- Directly optimize the policy from preference data
- The loss function for DPO is derived to adjust the policy by:
  - Increasing the likelihood of preferred responses.
  - Decreasing the likelihood of less preferred responses.

code repo and
talk materials

Algorithm: Direct Q-Function Optimization (DQO)

Limitations of current post-training algorithms

- **High computational requirements:**
  - rely on multiple models and extensive online sampling for training (e.g., PPO)
- **Challenges with multi-step reasoning:**
  - framed as bandit problems (e.g., DPO, DRO) struggle with tasks requiring complex multi-step reasoning

	PPO	DPO	KTO	**DQO**
Eliminates need for reward model	No	Yes	Yes	**Yes**
Free from online sampling	No	Yes	Yes	**Yes**
Can learn from unbalanced samples	Yes	No	Yes	**Yes**
Supports process supervision	Yes	No	No	**Yes**

code repo and
talk materials

Loss Function of DQO

- DQO is formulated as a **Markov Decision Process (MDP)**
- DQO simultaneously trains a **policy model** and a **value model**
- The loss functions are adapted from the **Soft Actor-Critic (SAC)** framework
- **Policy Model Loss:**

$$L_\pi(\theta) = \mathbb{E}_{(s_h,a_h)\sim\mathcal{D}}\left[\left(\beta \log \frac{\pi_\theta(a_h|s_h)}{\pi_{\text{ref}}(a_h|s_h)} - \Big(r(s_h,a_h) + V_\phi(s_{h+1}) - V_\phi(s_h)\Big)\right)^2\right].$$

  - $r(s_h,a_h) + V_\phi(s_{h+1}) - V_\phi(s_h)$ is **advantage function**
  - Intuitively, **encourage** actions where **advantage > 0; discourage** actions where **advantage < 0**

- **Value Model Loss:**

$$L_V(\phi) = \mathbb{E}_{(s_h,a_h)\sim\mathcal{D}}\left[\left(\overline{r}(s_h,a_h) + V_\phi(s_{h+1}) + \beta\mathcal{H}(\pi_\theta(\cdot|s_h)) - V_\phi(s_h)\right)^2\right].$$

code repo and
talk materials

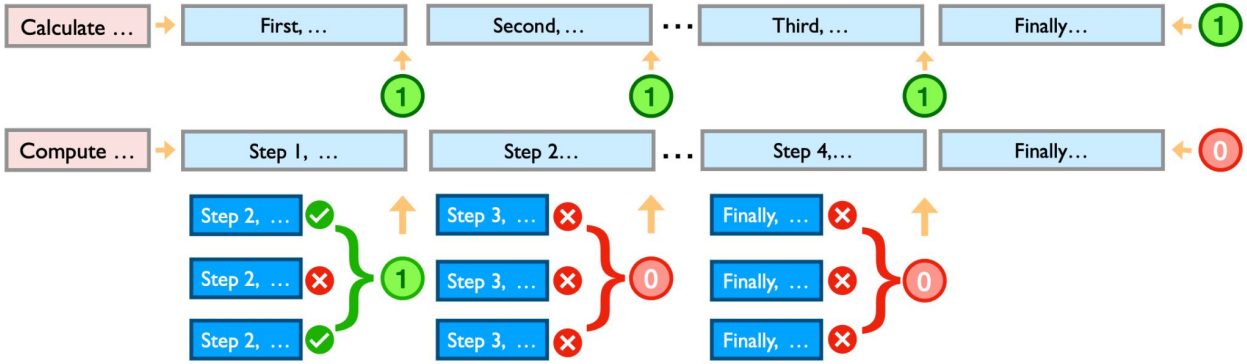
Experiment Results

Data Generation Method	GSM8K Greedy	GSM8K Sample	MATH Greedy	MATH Sample
Qwen2-7B-Instruct	59.06	53.30	37.44	35.42
SFT	85.06	84.15	44.56	41.34
Reject Sampling	84.15	83.47	49.82	48.02
DPO	**87.26**	84.23	51.66	49.00
KTO	86.35	83.55	50.32	46.30
DRO	86.73	83.39	51.18	48.40
DQO	**87.26**	**84.69**	**51.72**	**50.18**

Data Generation Method	GSM8K Greedy	GSM8K Sample	MATH Greedy	MATH Sample
Gemma-1.1-7B-it	39.65	38.89	17.04	16.06
SFT	53.45	45.46	21.64	18.00
Reject Sampling	53.60	53.37	21.74	20.42
DPO	57.85	59.41	22.62	22.66
KTO	50.49	50.19	18.56	18.56
DRO	60.58	60.05	24.56	23.70
DQO	**63.91**	**64.06**	**24.90**	**24.68**

code repo and
talk materials

DQO w/ Process Scores



Method Generation Method	GSM8K Greedy	 Sample	MATH Greedy	 Sample
Gemma-1.1-7b-it	39.65	38.89	17.04	16.06
DQO	63.91	64.06	24.90	24.68
DQO w/ process scores	**65.04**	**65.35**	**25.54**	**25.10**

Agenda:

Post-training Algorithms

- SFT, PPO, DPO, DQO
- **PRM for Code Generation**

Infrastructure and frameworks

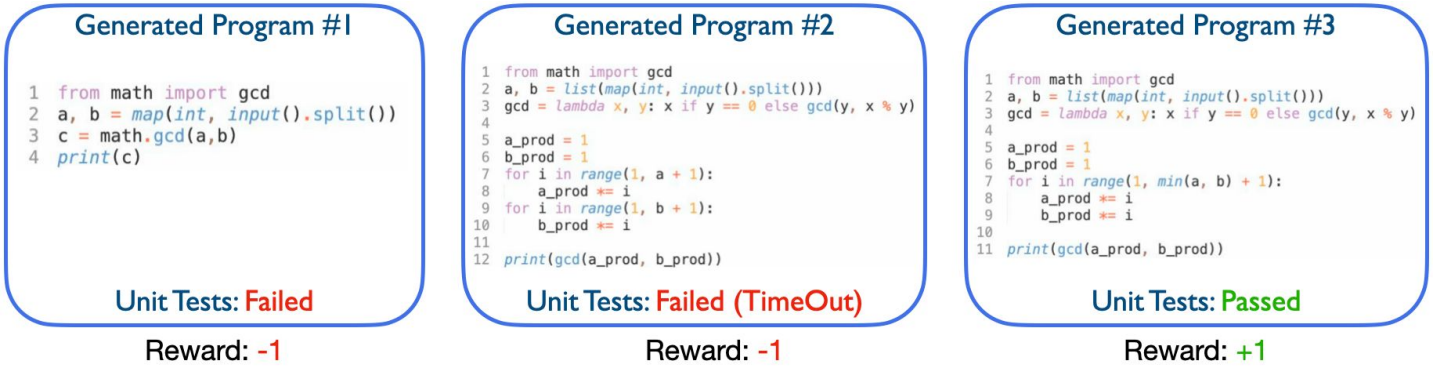
- Programming model, veRL
- Code demo

code repo and
talk materials

RL for Code Generation

Traditional code generation RL training pipelines for LLMs face limited learning because it only gets feedback at the end (pass/fail of unit tests).

- **Challenge**: Without feedback during code generation, the model struggles to correct errors or make gradual improvements.



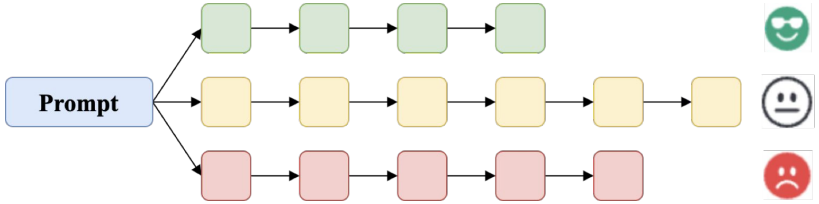
- **Human Insight**: Programmers refine code iteratively, correcting errors line-by-line, which AI typically does not do.

code repo and
talk materials

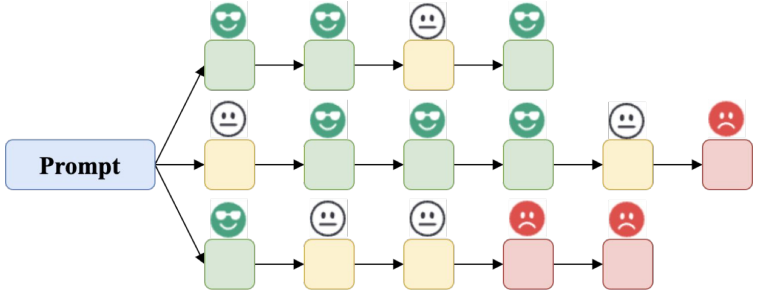
Process Supervision-Guided RL

By providing intermediate rewards that assess the correctness of partial code sequences, our approach guides the model more effectively toward generating correct programs

- **Traditional method**: Reinforcement Learning from Unit Test Feedback (RLTF)



- **Our method**: Process Supervision-Guided RLTF



code repo and
talk materials

Process Reward Model (PRM)

What is a PRM? A model that provides feedback for each line of code during generation, enabling step-by-step corrections and guidance.

- A PRM evaluates the correctness of each line of code by predicting a score between -1 (incorrect) and +1 (correct), providing feedback for the generated code snippet

```
-0.02 Here is a Python function to solve the problem:

-0.05 ```python
+0.95 def solve():
+0.98     S = input()
+0.95     char_count = {}
+0.96     for char in S:
+0.97         if char in char_count:
+0.96             char_count[char] += 1
+0.96         else:
+0.96             char_count[char] = 1
+0.96     distinct_char_count = 0
-0.41     for count in char_count.values():
-0.68         if count == 2:
-0.56             distinct_char_count += 1
-0.57     if distinct_char_count == 2:
-0.56         print("Yes")
-0.57     else:
-0.56         print("No")
```

code repo and
talk materials

Process Supervision-Guided Policy Optimization

In the RL training phase, the PRM is integrated to supply **dense rewards** (line-by-line guidance) and **value initialization** (starting points based on PRM predictions).

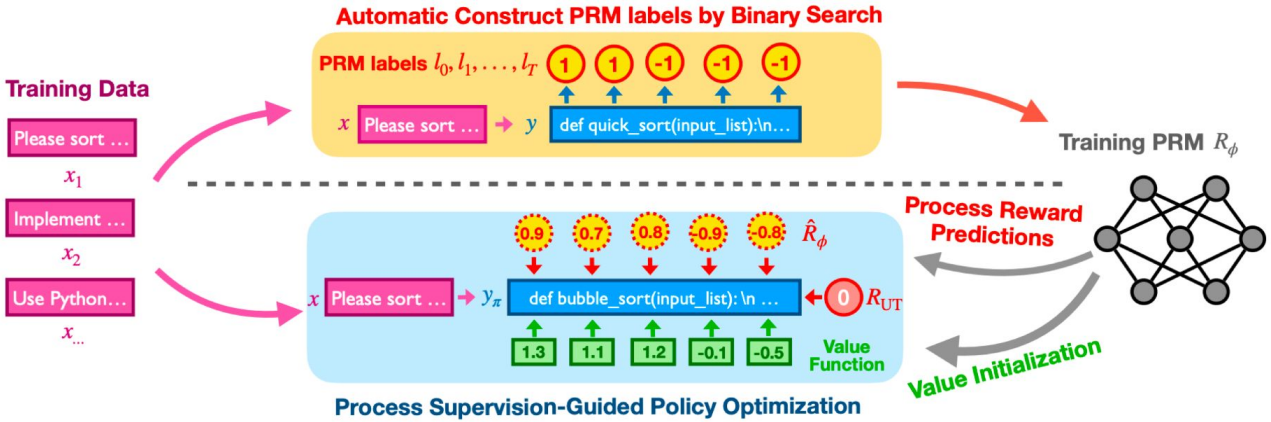


Figure 1: Overview of our method. The approach consists of two main components: (1) A binary search-based method to automate PRM training data labeling, which is used to train a code PRM; and (2) Integration of the PRM into RL training as both dense rewards and value function initialization.

code repo and
talk materials

PRM Data Collection Process

For each generated code sequence, a binary search identifies the first line where errors occur with a BoN completer

Step 1: Start with the whole code sequence

Step 2: Check if the prefix (up to a midpoint line) can be completed into a correct program that passes tests with the BoN completer

Step 3: Narrow down by adjusting the midpoint until finding the line where errors start

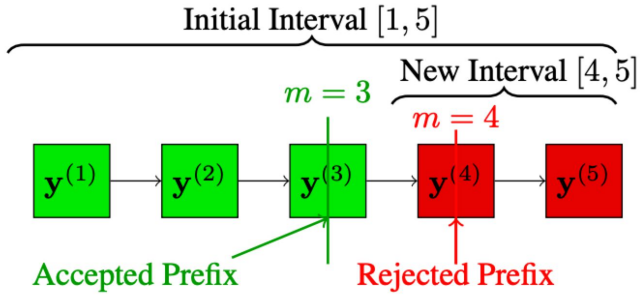


Figure 2: Binary search over code steps at line level to label prefixes. The first midpoint at $m = 3$ is accepted, so the search interval moves to $[4, 5]$. The next midpoint at $m = 4$ is rejected, indicating unrecoverable errors occur after step 3.

code repo and
talk materials

Overall Training Pipeline

To develop a PRM capable of providing meaningful process supervision throughout the RL training process, we designed the following training pipeline to create a robust PRM and integrate it into RL training:

1) RL Baseline Training: Fine-tune the SFT policy using PPO

2) PRM Data Collection:

- Sample multiple policy checkpoints in RL baseline training to cover the state space
- Collect training data for PRM using binary search to label actions for each checkpoint

3) PRM Training: Train the PRM using regression loss on collected data

4) Integrating PRM into RL:

- Start from the scratch, fine-tune the SFT policy using RL with PRM
- Use PRM as dense, step-wise rewards in PPO (**DenseReward**)
- Use PRM as the initialization of the critic in PPO (**ValueInit**)

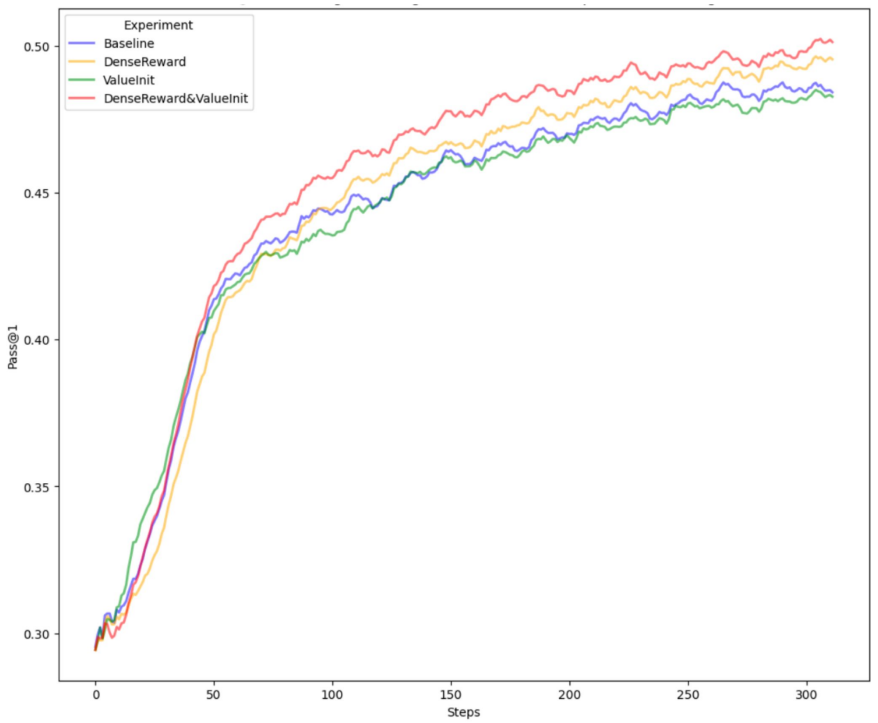
code repo and
talk materials

Comparing RL Training Curves

We compared the following settings:

- RL Baseline
- RL w/ DenseReward
- RL w/ ValueInit
- RL w/ DenseReward + ValueInit

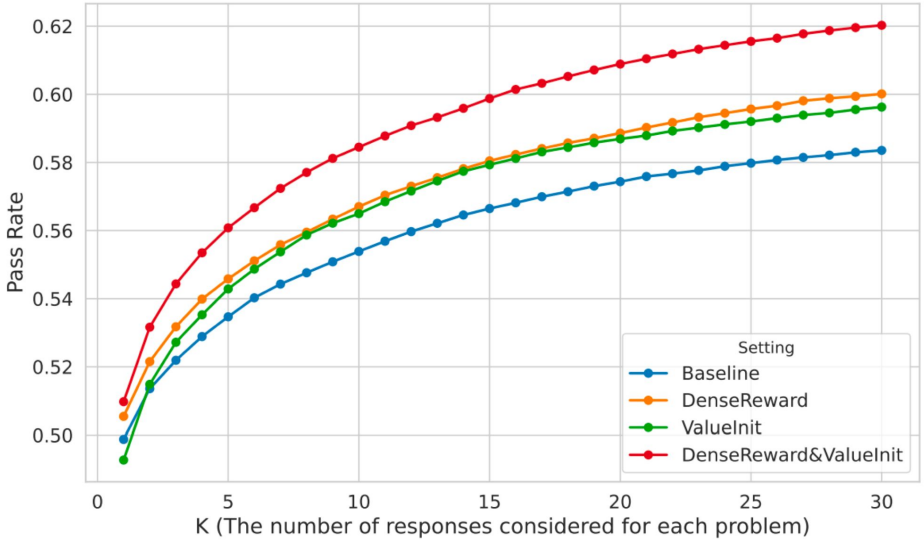
When PRM is applied as both
DenseReward and ValueInit,
it achieves the best performance



code repo and
talk materials

PRM Enhances Exploration in RL Training

We compare the Best-of-K performance of the policy learned across all four settings on the training set.
Both **DenseReward** and **ValueInit** independently enhance performance. Furthermore, when both are enabled, the model achieves the greatest improvement.



code repo and
talk materials

Evaluation Results

Model	Setting		Dataset						
			LiveCodeBench				InHouseBench		
	Dense Reward	Value Init.	Easy	Medium	Hard	Overall	Contest	NL2Alg	Overall
GPT-4o-mini	-	-	81.9	27.2	3.6	40.7	43.8	68.4	51.4
Qwen2-72B	-	-	65.0	21.3	2.8	32.2	14.8	51.3	26.1
Gemini-Flash-1.5	-	-	67.7	13.1	1.9	29.6	-	-	-
DeepseekCoder-33B	-	-	60.8	14.8	1.2	27.7	10.3	50.3	22.7
Ours-SFT	-	-	55.3	9.3	0.3	23.5	10.4	41.4	20.0
Ours-RL	×	×	**70.0**	7.2	1.7	28.2	24.4	48.7	31.8
	×	✓	67.9	8.9	1.9	28.2	25.0	45.4	31.4
	✓	×	68.5	9.9	**2.5**	28.9	25.2	48.1	32.3
	✓	✓	69.3	**12.0**	1.6	**29.8**	**27.9**	**53.5**	**35.8**

Table 1: Comparison of model performance (Pass@1) across LiveCodeBench and InHouseBench with PRM as DenseReward and ValueInit settings. The performance of our models (`InHouse-Lite` series) on both LiveCodeBench and InHouseBench are averaged over 10 independent runs. We also report the performance of other public models for comparative purpose. The performance of Gemini-Flash-1.5 on InHouseBench is omitted due to legal considerations.

code repo and
talk materials

PRM Improves Long-horizon Code Generation

We measure the improvement of the policy trained with PRM compared to the baseline policy and analyze its effect based on the length of the generated responses.

Our findings show that PRM provides greater benefits for generating longer responses.

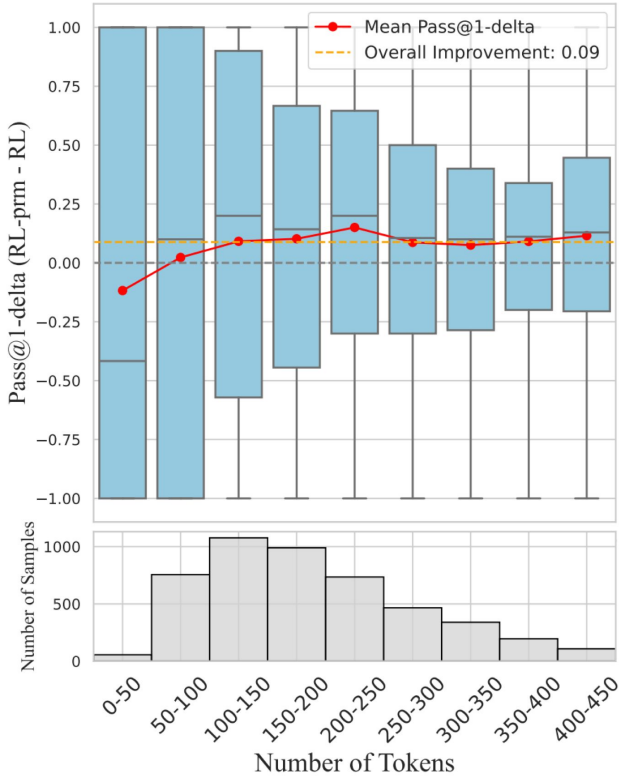


Figure 3: Pass@1 difference between policies trained with and without PRM across varying response lengths. Policies trained with PRM exhibit consistent improvements over those without PRM for longer-horizon responses (greater than 100 tokens). This demonstrates PRM's effectiveness in providing intermediate feedback, thereby enabling RL to do more explorations.

Agenda:

Post-training Algorithms

- SFT, PPO, DPO, DQO
- PRM for Code Generation

Infrastructure and frameworks

- **Programming model, veRL**
- **Code demo**

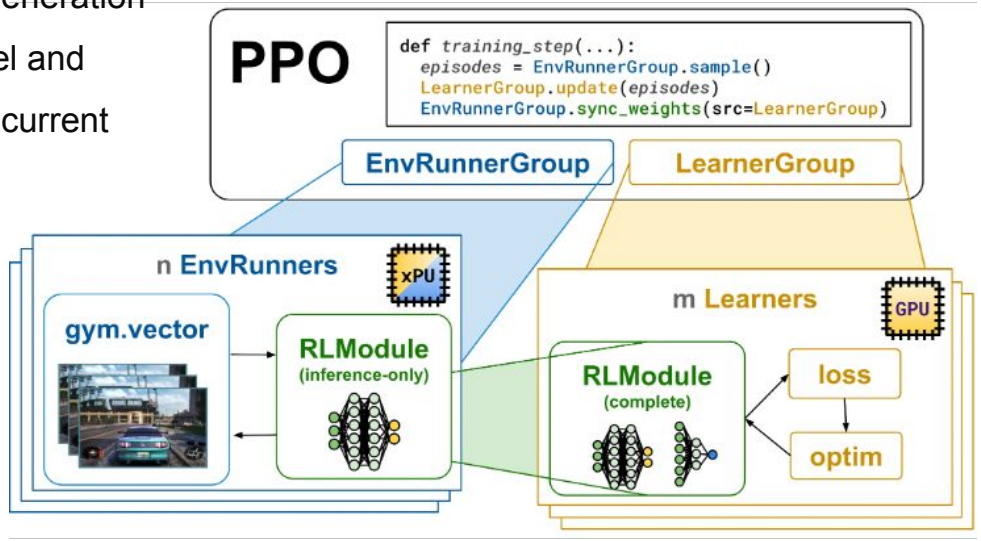
https://docs.ray.io/en/latest/rllib/package_ref/doc/ray.rllib.env.env_runner.EnvRunner.html

code repo and
talk materials

Can we Reuse Existing RL Infra for LLMs?

RLlib and EnvRunner

- gym.vector -> Autoregressive Generation
- Reward is computed via a model and the KL divergence between the current policy and the initial policy



Kwon et al, Efficient Memory Management for Large Language Model Serving with PagedAttention, 2021
Narayanan et al, Efficient Large-Scale Language Model Training on GPU Clusters Using Megatron-LM, 2021
Zhao et al, PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel, 2023

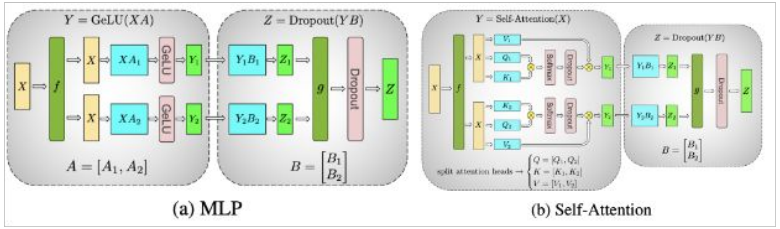
code repo and talk materials

Challenge 1: Model sizes scale

Distributed data parallelism is not feasible to train large models

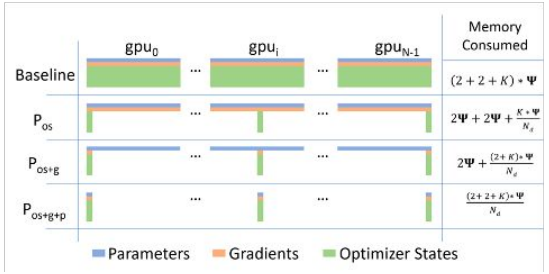
- Parallelism for model & activation sharding
- One 70B model requires 840GB memory for optimizer states, RL involves multiple models

Tensor Parallelism



Pipeline Parallelism

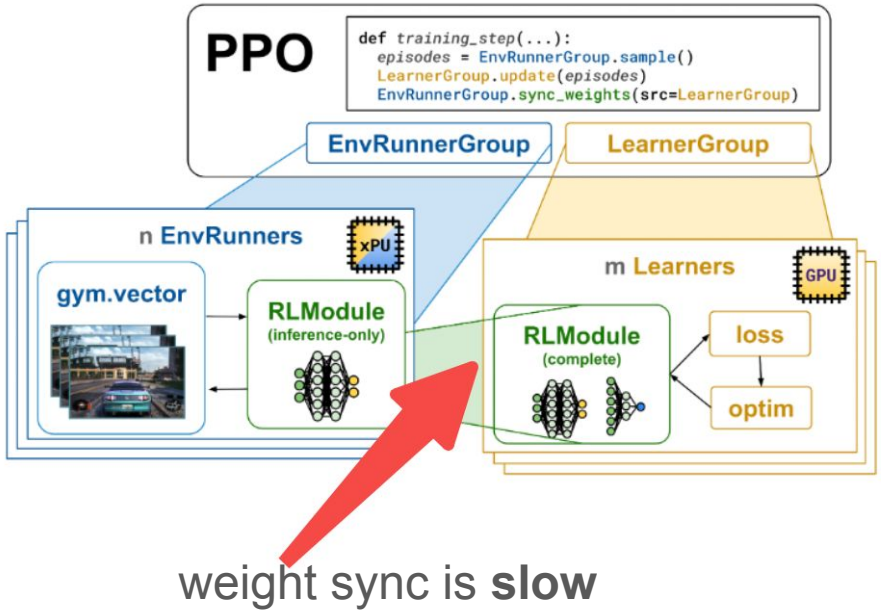
ZeRO/Fully Sharded Data Parallelism



code repo and
talk materials

Challenge 2: Weight sync is expensive

For instance, Llama 405B in bf16 requires transferring **810GB** of weight between train & rollout



code repo and
talk materials

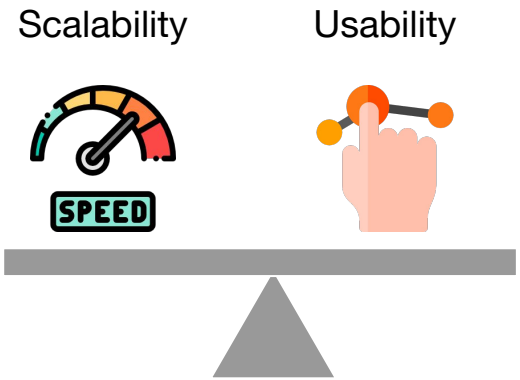
Challenge 3: Flexibility for Research

The nature of RL research

- Typically doesn't care about neural net architecture
- Care more about RL dataflow
  - Trajectory sample: on-policy or off-policy
  - Loss function: Q loss or policy loss

Flexibility means that

- New algorithms can be easily implemented
- Modules of the framework can be reused



Barham et al. Pathways: Asynchronous Distributed Dataflow for ML, 2022
https://docs.ray.io/en/latest/ray-core/ray-dag.html

code repo and
talk materials

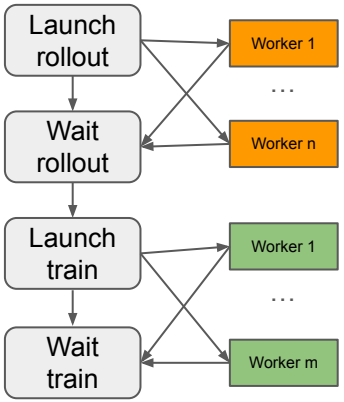
Single-Controller vs Multi-Controller

Single-controller

- A central control process with a list of workers
- The control process gives workers instructions and data to compute
- The control process gather results from workers

+ flexible. RPC semantics
- dispatch overhead

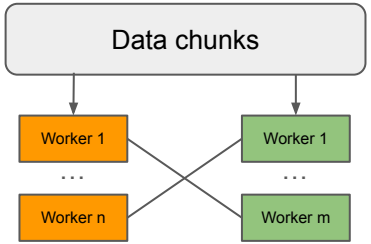
Example: deepseed-chat



Multi-controller

- A list of workers, no central control process
- Each worker executes the same program using different chunks of data in a batch (SPMD)

+ fast
- less flexible



Example: Megatron-LM, vLLM

code repo and
talk materials

Key Observations (Two-level Data Flow)

Dataflow of RL

- Control how data is computed and exchanged among different components (e.g., actor/critic/rollout)

Dataflow of LLM computation

- Most open source LLM computation workloads are implemented using multi-controller patterns: vLLM, Megatron-LM
- Primitives
  - Autoregressive generation
  - LLM inference
  - LLM training

code repo and
talk materials

Hybrid Controller

Main idea

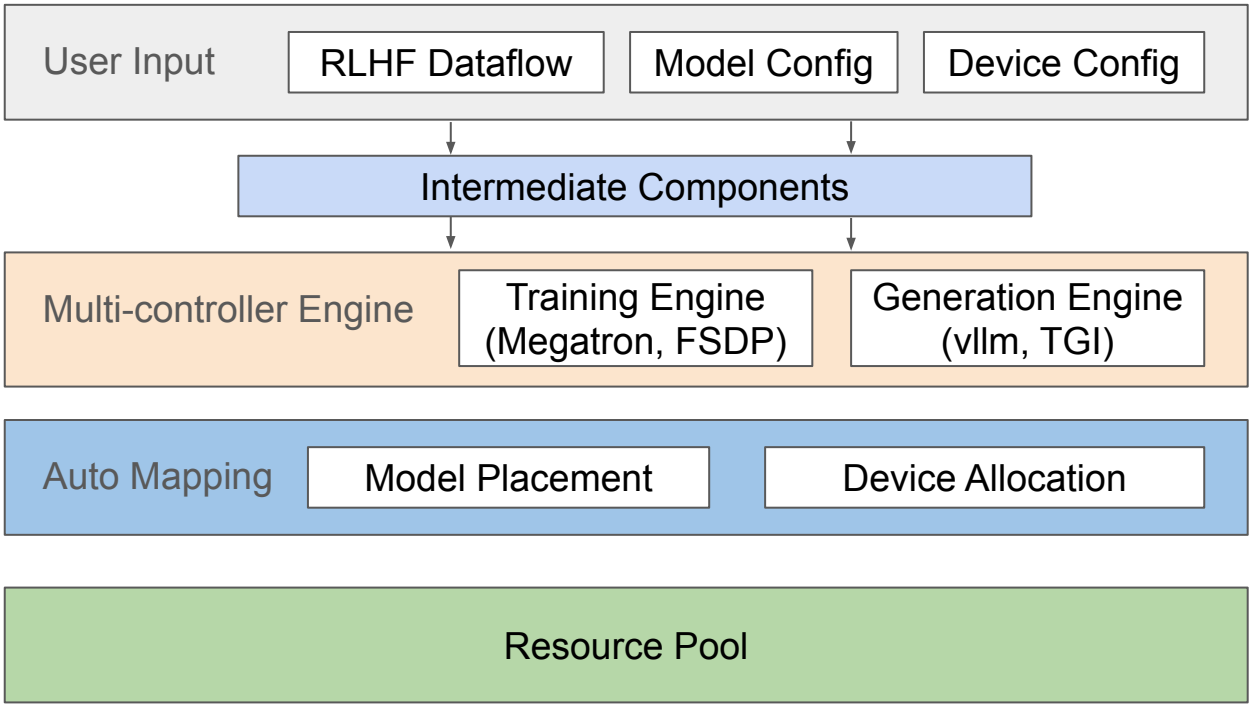
- Single-controller for RL dataflow
- Multi-controller for computational dataflow (LLM workload primitives)
- **The single-controller process can manipulate computations encapsulated by multi-controller as if it is a single process**

Key intermediate components

- A data structure (APIs) that can be invoked by the single process **externally** and can perform multi-controller-based computations **internally**.

code repo and
talk materials

veRL: distributed LLM RL with Hybrid Controller



code repo and
talk materials

RLHF Intermediate Components

Multi-controller engines

- Rollout Engine
- Reward model
- Ref policy model
- Actor model
- Critic model

Implementation

- Model initialization
- vLLM for generation
- Megatron-LM/FSDP for training and inference

```
# ppo actor/rollout
@ray.remote
class PPOActorRolloutHybridEngine(Worker):
    def __init__(self, config):
            pass

    @register(DP_COMPUTE_PROTO)
    def compute_log_prob(self, batch):
            pass

    @register(DP_COMPUTE_PROTO)
    def update_policy(self, batch):
            pass

    @register(DP_COMPUTE_PROTO)
    def generate_sequences(self, batch):
            pass

# ppo reference policy
@ray.remote
class PPOReferencePolicy:
    def __init__(self, config):
            pass

    @register(DP_COMPUTE_PROTO)
    def compute_log_prob(self, batch):
            pass
```

```
# ppo critic
@ray.remote
class PPOCritic(Worker):
    def __init__(self, config):
            pass

    @register(DP_COMPUTE_PROTO)
    def compute_values(self, batch):
            pass

    @register(DP_COMPUTE_PROTO)
    def update_critic(self, batch):
            pass


# reward model
@ray.remote
class PPORewardModel(Worker):
    def __init__(self, config):
            pass

    @register(DP_COMPUTE_PROTO)
    def compute_reward(self, batch):
            pass
```

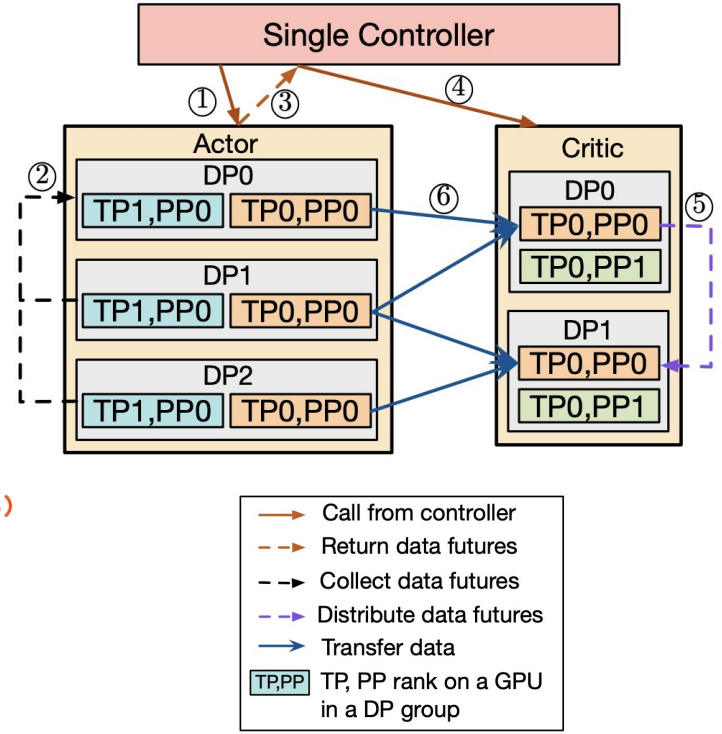
code repo and
talk materials

RLHF Single Process Controller

```
# intialize worker groups
actor_wg = …
critic_wg = …
ref_wg = …
reward_wg = …

# ppo actor/rollout
for prompt_batch in dataloader:
    batch = actor_wg.generate_sequences(prompt_batch)
    values = critic_wg.compute_values(batch)
    ref_log_prob = ref_wg.compute_log_prob(batch)
    score = reward_wg.compute_reward(batch)
    returns, advantage = compute_advantage(score, ref_log_prob, values)
    actor_wg.update(batch, advantage)
    critic_wg.update(batch, returns)
```

WorkerGroup handles how to split/replicate batch of data based on the underlying model/data parallelism strategies



code repo and
talk materials

veRL Example: new algorithms

Support PPO variants such as ReMax, GRPO and Safe-RLHF with a few line of changes

- ReMax/GRPO estimates advantages without using critic
- Safe-RLHF deploys multiple reward models

```
# Initialize cost model by reusing the RewardWorker
cost = RewardWorker(cost_config, resource_pool)
...  # omit other models initialization
algo_type = "Safe-RLHF" # specify different RLHF numerical computation.
# Examples of PPO and Safe-RLHF
for (prompts, pretrain_batch) in dataloader:
    # Stage 1: Generate responses
    batch = actor.generate_sequences(prompts)
    batch = actor.generate_sequences(prompts, do_sample=False)
    # Stage 2: Prepare experience
    batch = critic.compute_values(batch)
    batch = reference.compute_log_prob(batch
    batch = reward.compute_reward(batch)
    batch = cost.compute_cost(batch)
    batch = compute_advantages(batch, algo_type)
    # Stage 3: Actor and critic training
    critic_metrics = critic.update_critic(batch, loss_func=algo_type)
    pretrain_loss = actor.compute_loss(pretrain_batch)
    batch["pretrain_loss"] = pretrain_loss
    actor_metrics = actor.update_actor(batch, loss_func=algo_type)
```

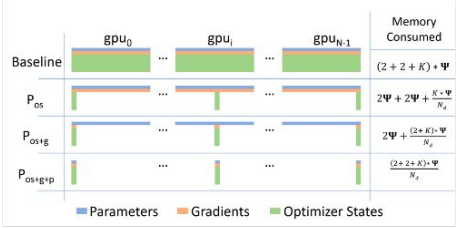
is added for ReMax
Not necessary in ReMax
is added for Safe-RLHF

code repo and
talk materials

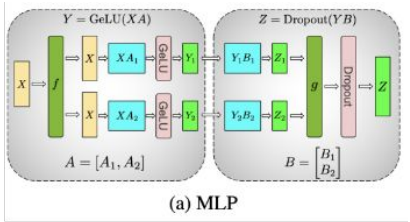
Generalized Hybrid Engine

Motivation

- Share the weights between rollout and training if possible to avoid weight synchronization
- Perform model parameter sharding redistribution (e.g. from FSDP in training to TP in rollout)



Training (zero3)



Rollout (tensor parallelism)

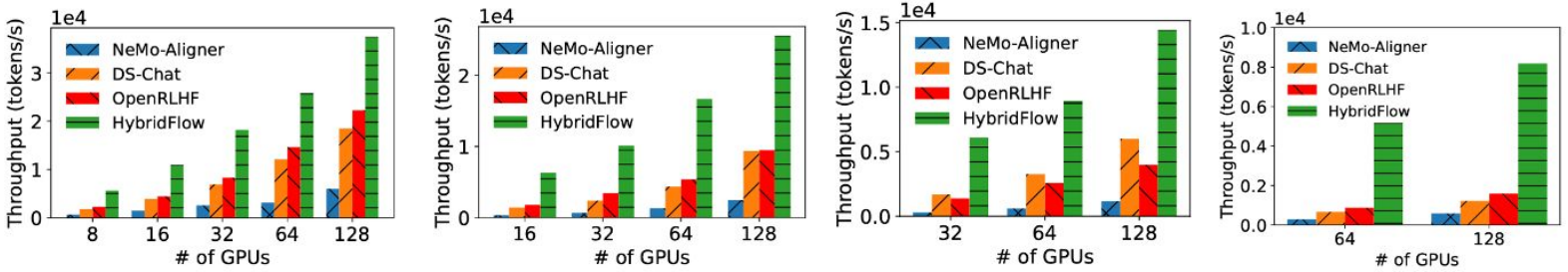
```python
@register(DP_COMPUTE_PROTO)
def generate_sequences(self, batch):
    # inside the context manager, the weight from the actor is bind to the rollout
    with self.sharding_manager:
        # transform the input data device mesh from FSDP to TP
        prompts = self.sharding_manager.preprocess_data(prompts)
        # generate sequences
        responses = rollout.generate_sequences(prompts)
        # transform the output data device mesh from TP to FSDP
        output = self.sharding_manager.postprocess_data(responses)
```

Under the hood, sharding manager performs weight resharding via Pytorch DTensor

code repo and
talk materials

Experimental Results

1.68x to 20.57x compared with state-of-the-art RLHF frameworks ranging from 7B to 70B models



(a) 7B (1.68×~8.63×) (b) 13B (2.70×~18.96×) (c) 34B (2.41×~20.57×) (d) 70B (5.17×~17.98×)

Notebook: PPO training with veRL

News

- [2024/12] The team will present Post-training LLMs: From Algorithms to Infrastructure at NeurIPS 2024.
  - Slides, notebooks, and video be available soon

code repo and
talk materials

What if building something sophisticated…

MCTS for generation

- Find tokens with high returns by using policy and reward models
- Interleave decoding and scoring

RL data flow

- Given the prefix, select a new leaf node
- Insert the new node to the tree if not exists
- Evaluate reward for the new node
- Update the reward to the tree

```
def decode_by_mcts():
    response = []
    while next_token != EOS:
        root = Node(next_token)
        # decode next token
        while root.count < budget:
            # mcts calculation
            new_token = select()
            new_node = expand(new_token)
            simulation(new_node, new_token)
            back_propagation()
        next_token = find_token_by_max_count()
        response.append(next_token)
    return response
```

code repo and
talk materials

Building MCTS on veRL

Worker Group

```
# ppo actor/rollout
@ray.remote
class MCTSRolloutEngine(Worker):
    def __init__(self, config):
      pass

    ...

    @register(DP_COMPUTE_PROTO)
    def generate_next_token(self, batch):
        sentences = []
        for sentence in batch:
            root = get_mcts_root(sentence)
            if root.count < budget:
                new_token = select()
                root.expand(new_token)
                sentence = sentence.append(new_token)
            else:
                new_token = root.find_next_token_by_max_count()
                sentence = root.sentence.append(new_token)
        sentences.append(sentence)
        return sentences

    @register(DP_COMPUTE_PROTO)
    def back_propagation(self, values):
        back_propagation(values)
```

Single Controller

```
for batch in dataloader:
    while any(batch.last_tokens != EOS):
        batch = actor_wg.generate_next_token(batch)
        values = critic_wg.compute_values(batch)
        actor_wg.back_propagation(values)
    values = critic_wg.compute_values(batch)
    ref_log_prob = ref_wg.compute_log_prob(batch)
    score = reward_wg.compute_reward(batch)
    returns, advantage = compute_advantage(score, ref_log_prob, values)
    actor_wg.update(batch, advantage)
    critic_wg.update(batch, returns)
```

Career Opportunities

Experienced

Research Scientist, Large Language Model

San Jose / Seattle

Experienced

Research Scientist in ML Systems

San Jose / Seattle

Campus

Research Scientist, Large Language Model - PhD 2025 Start

San Jose / Seattle

Campus

Research Scientist in ML Systems

San Jose / Seattle

Intern

Student Researcher, Large Language Model

San Jose / Seattle

Intern

Student Researcher (Doubao (Seed) - Foundation Model) - 2025 Start (PhD/MS/BS)

San Jose / Seattle

Talent Acquisition Team

Joy Miao - LLM
joy.miao@bytedance.com WeChat ID: **mjy427**

Ariel Ren - ML System
ariel.ren@bytedance.com WeChat ID: **aaar0415**

THANKS.

Single-controller Example: Special token

Regenerate sentences

- Special token
- Using existing APIs

RL data flow

- Generate sentences
- Compute scores and log probs
- Compute the returns and advantages
- Find the token with the smallest advantage
- Regenerate sentences

```
# intialize worker groups
actor_wg = …
critic_wg = …
ref_wg = …
reward_wg = …

# ppo actor/rollout
for prompt_batch in dataloader:
    batch = actor_wg.generate_sequences(prompt_batch)
    train_batch, train_returns, train_advantage = torch.empty(0, seq_len)
    while special token condition:
        values = critic_wg.compute_values(batch)
        ref_log_prob = ref_wg.compute_log_prob(batch)
        score = reward_wg.compute_reward(batch)
        returns, advantage = compute_advantage(score, ref_log_prob, values)
        train_batch = train_batch.concat(batch)
        train_returns = train_returns.concat(returns)
        train_advantage = train_advantage.concat(advantage)
        special_tokens_pos = find_special_token(batch, train_advantage)
        batch = batch[:, : special_tokens_pos]
    actor_wg.update(train_batch, train_advantage)
    critic_wg.update(train_batch, train_returns)
```