HybridFlow: A Flexible and Efficient RLHF Framework

Guangming Sheng
The University of Hong Kong
gmsheng@connect.hku.hk

Chi Zhang
ByteDance
zhangchi.usc1992@bytedance.com

Zilingfeng Ye
ByteDance
yezilingfeng@bytedance.com

Xibin Wu
ByteDance
wuxibin@bytedance.com

Wang Zhang
ByteDance
zhangwang.nozomi@bytedance.com

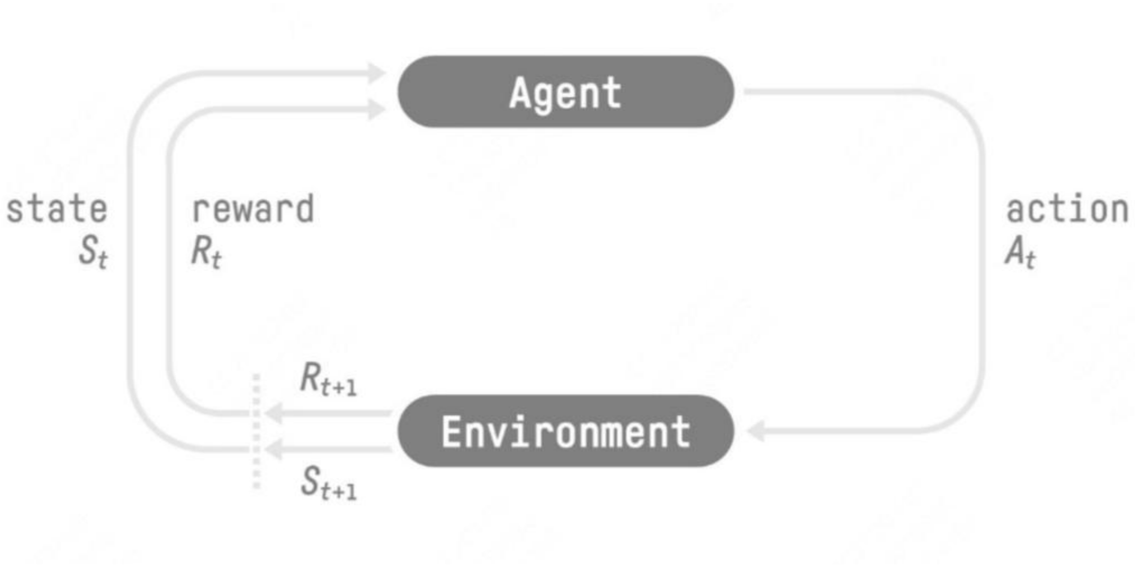
Ru Zhang
ByteDance
zhangru.1994@bytedance.com

Yanghua Peng
ByteDance
pengyanghua.yanghua@bytedance.com

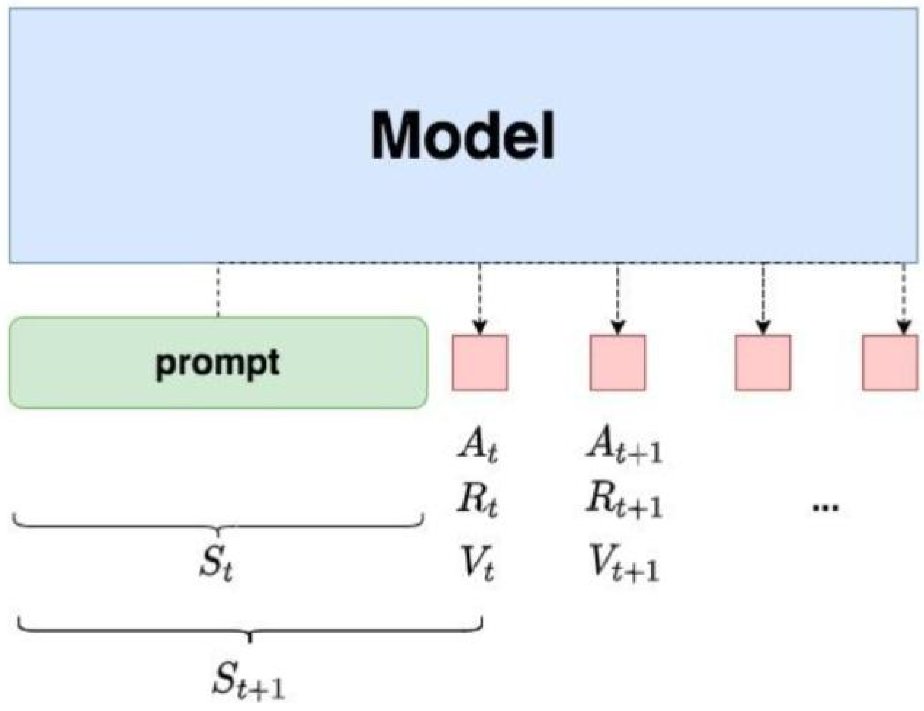
Haibin Lin
ByteDance
haibin.lin@bytedance.com

Chuan Wu
The University of Hong Kong
cwu@cs.hku.hk

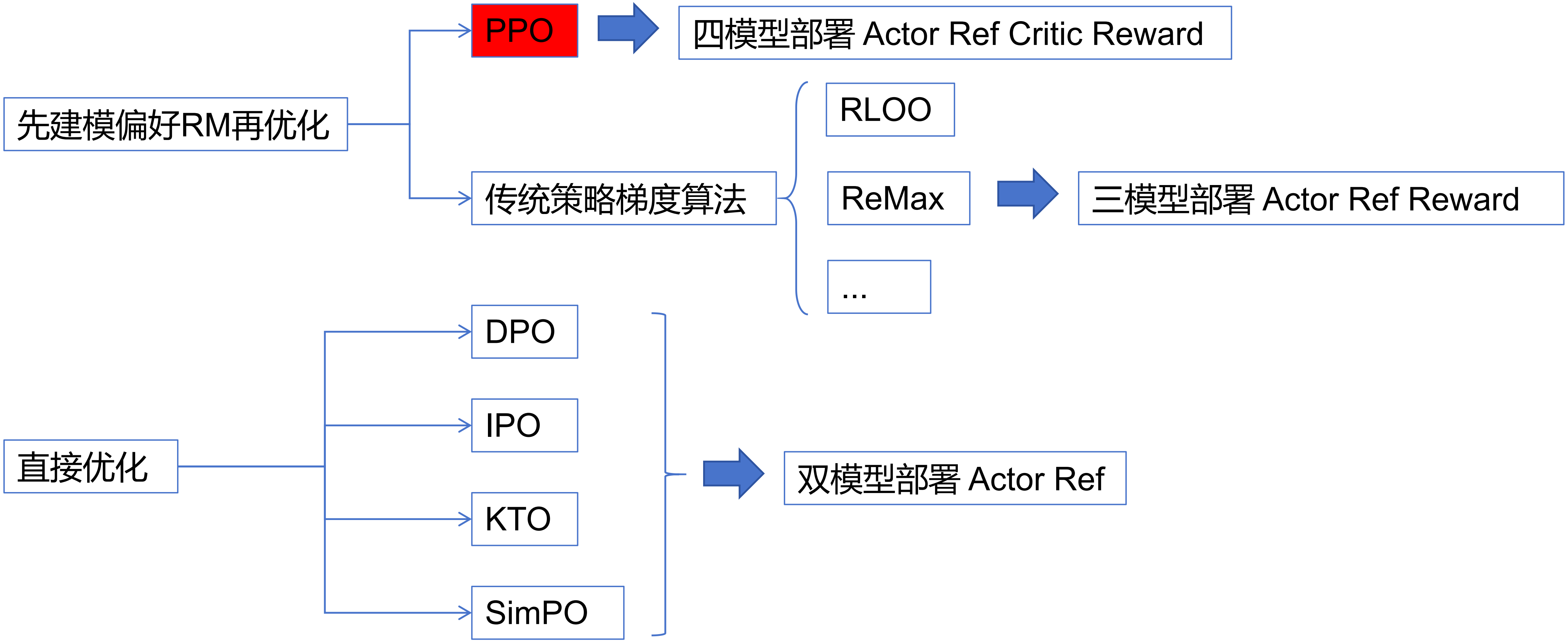
NLP中的强化学习



RL in NLP

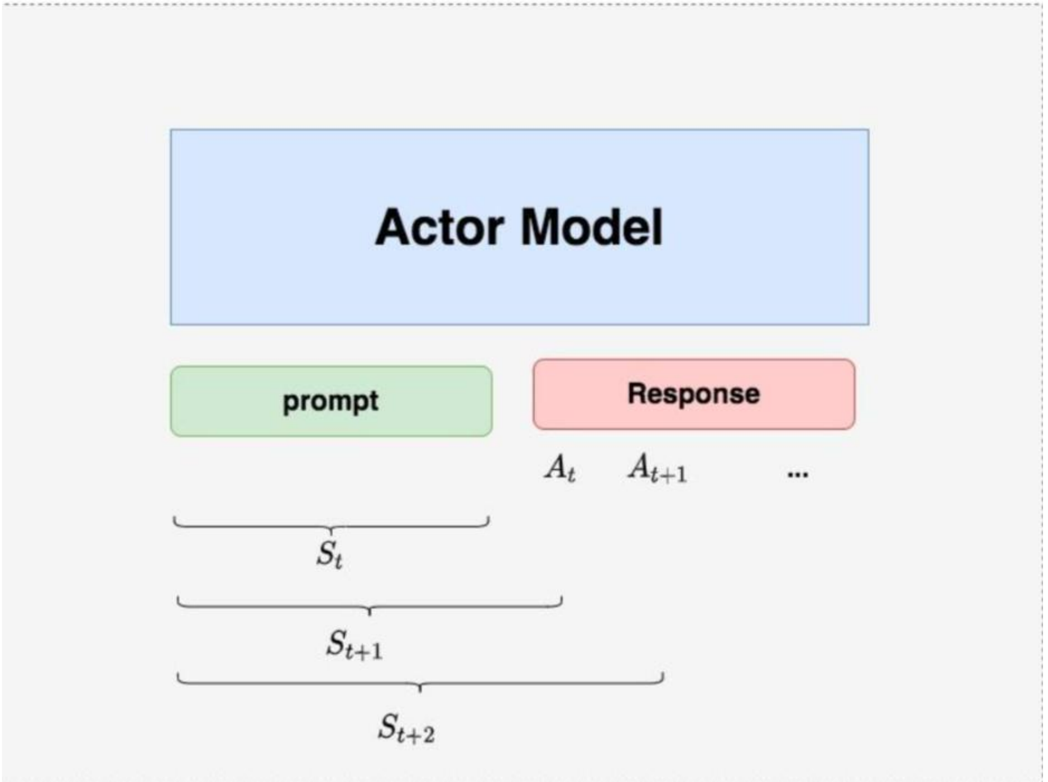


RL Training中的若干部署算法



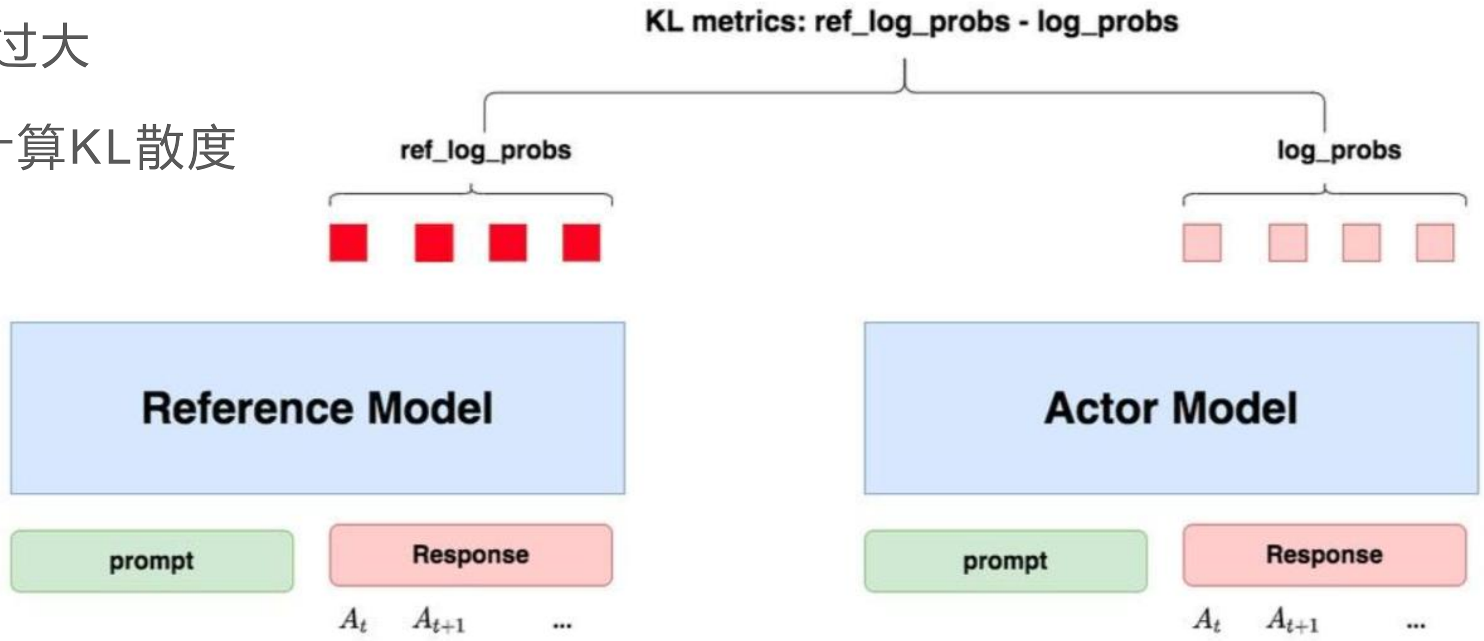
RLHF中的四个模型（PPO）- Actor Model

- Actor Model
- Actor就是我们想要训练的目标语言模型，一般用SFT阶段产出模型对其做初始化
- 将prompt输出给Actor Model
- 将prompt+response输出给后续模型计算得到loss



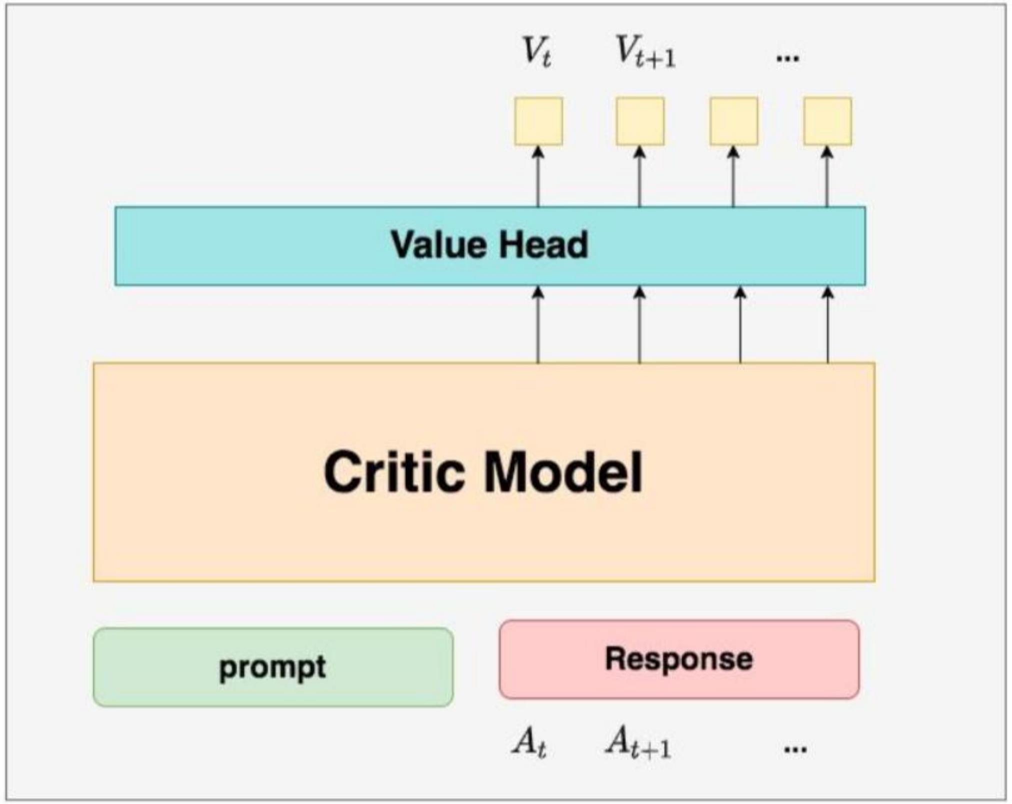
RLHF中的四个模型（PPO）- Reference Model

- Reference Model
- 一般也用SFT阶段得到的SFT模型做初始化
- 训练过程中参数是冻结的
- 主要作用是防止Actor与Ref差距过大
- 利用两个模型输出的log_probs计算KL散度



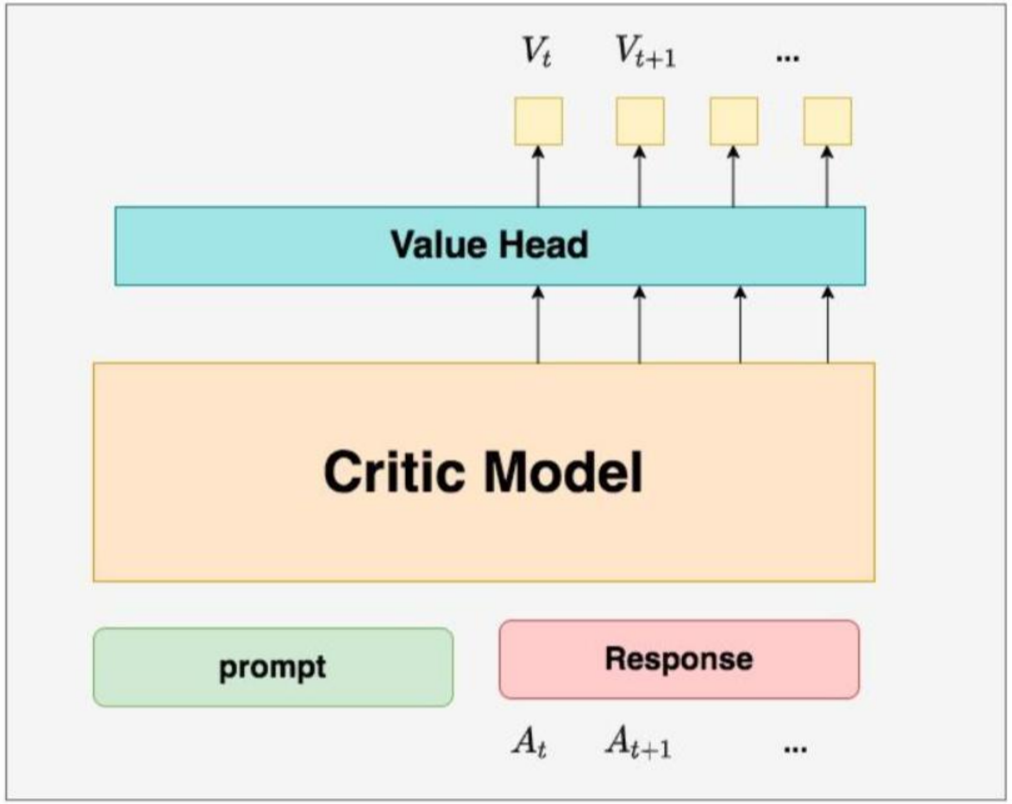
RLHF中的四个模型（PPO）- Critic Model

- Critic Model
- 用于预测期望总收益Vt
- 将输出结果映射成单一Vt值
- 需要做参数更新
- 设计和初始化方式有很多种，例如和Actor共享部分参数、从RW阶段的Reward Model初始化而来等



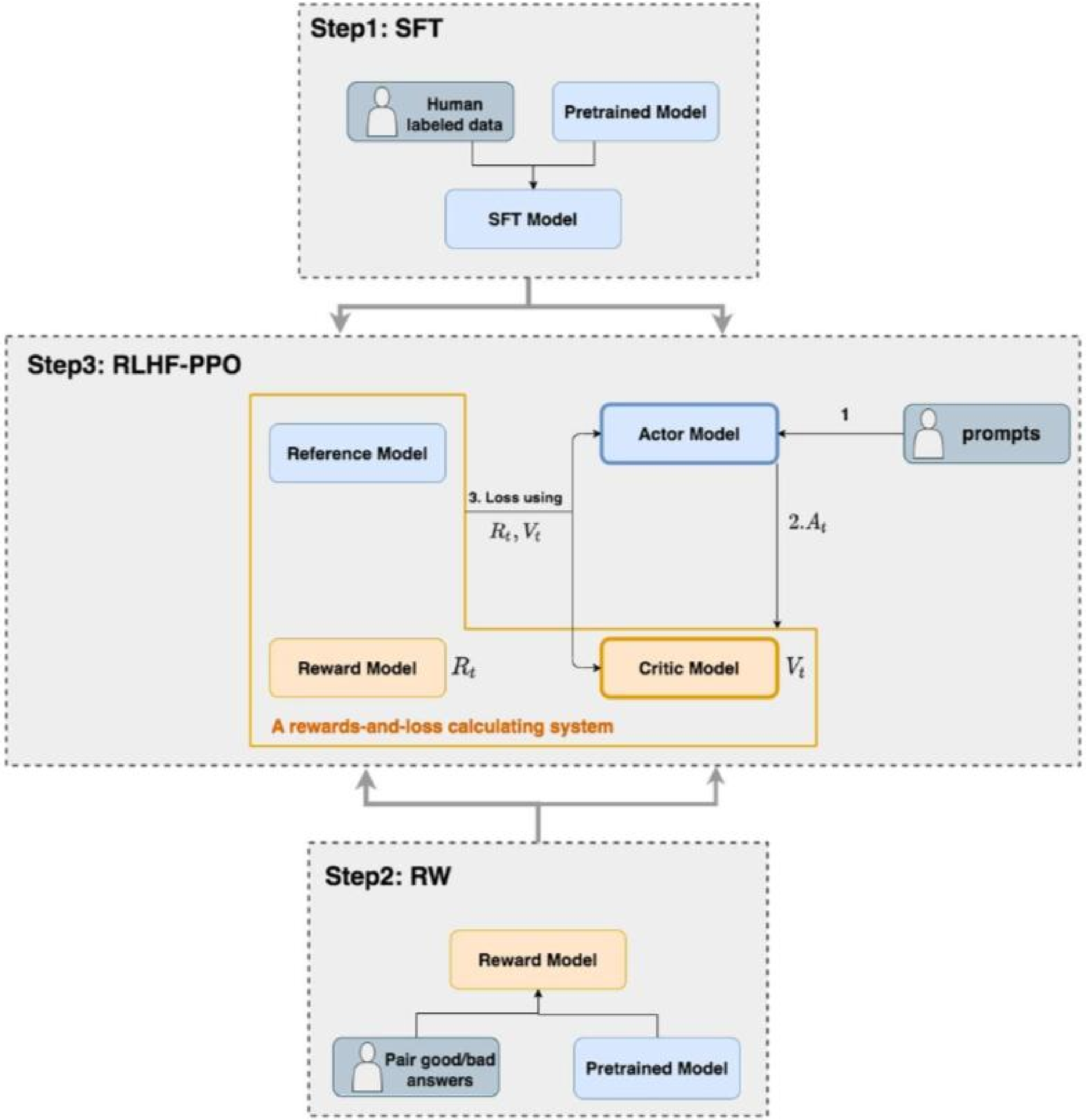
RLHF中的四个模型（PPO）- Reward Model

- Reward Model
- 用于计算生成token的即时收益Rt
- 结构与Critic Model是类似的
- 就是RW阶段所训练的奖励模型
- 在RLHF过程中，它的参数是冻结的

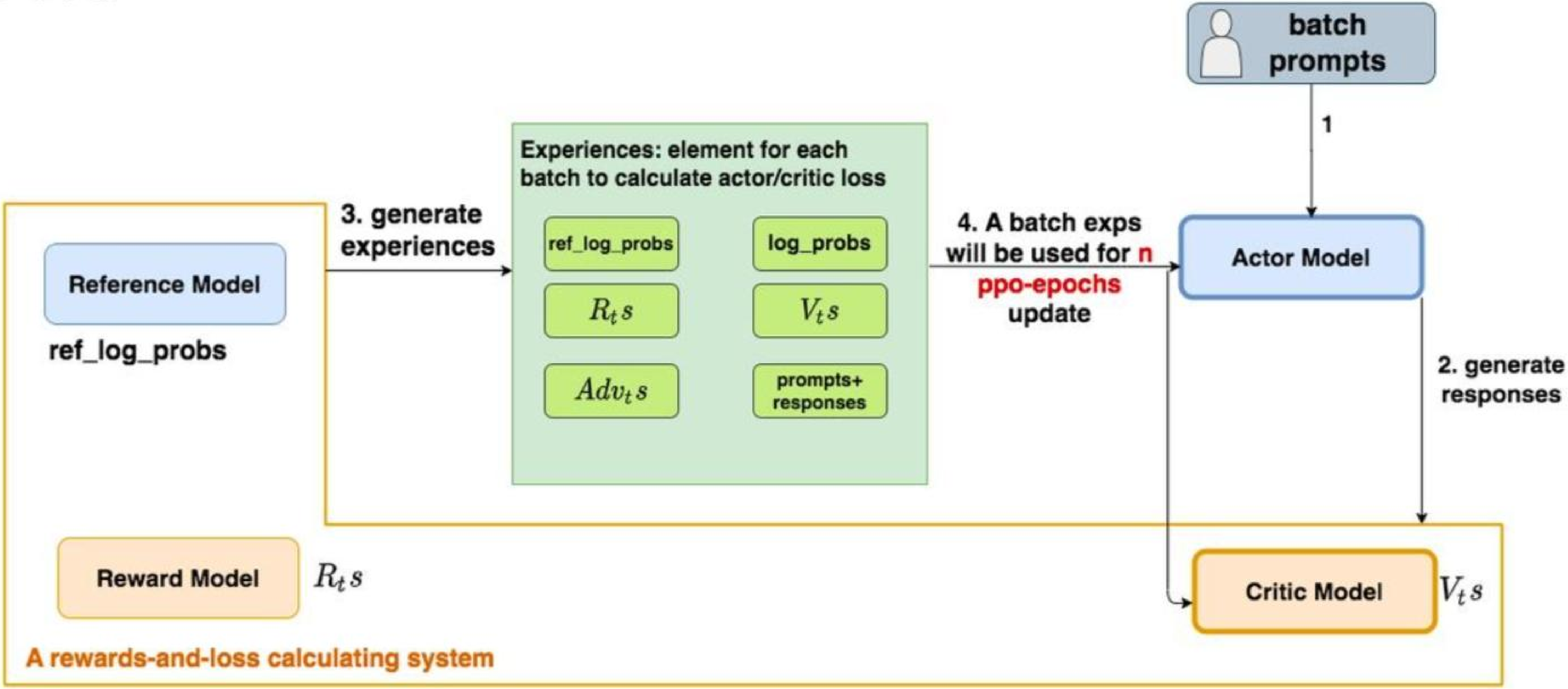


PPO workflow

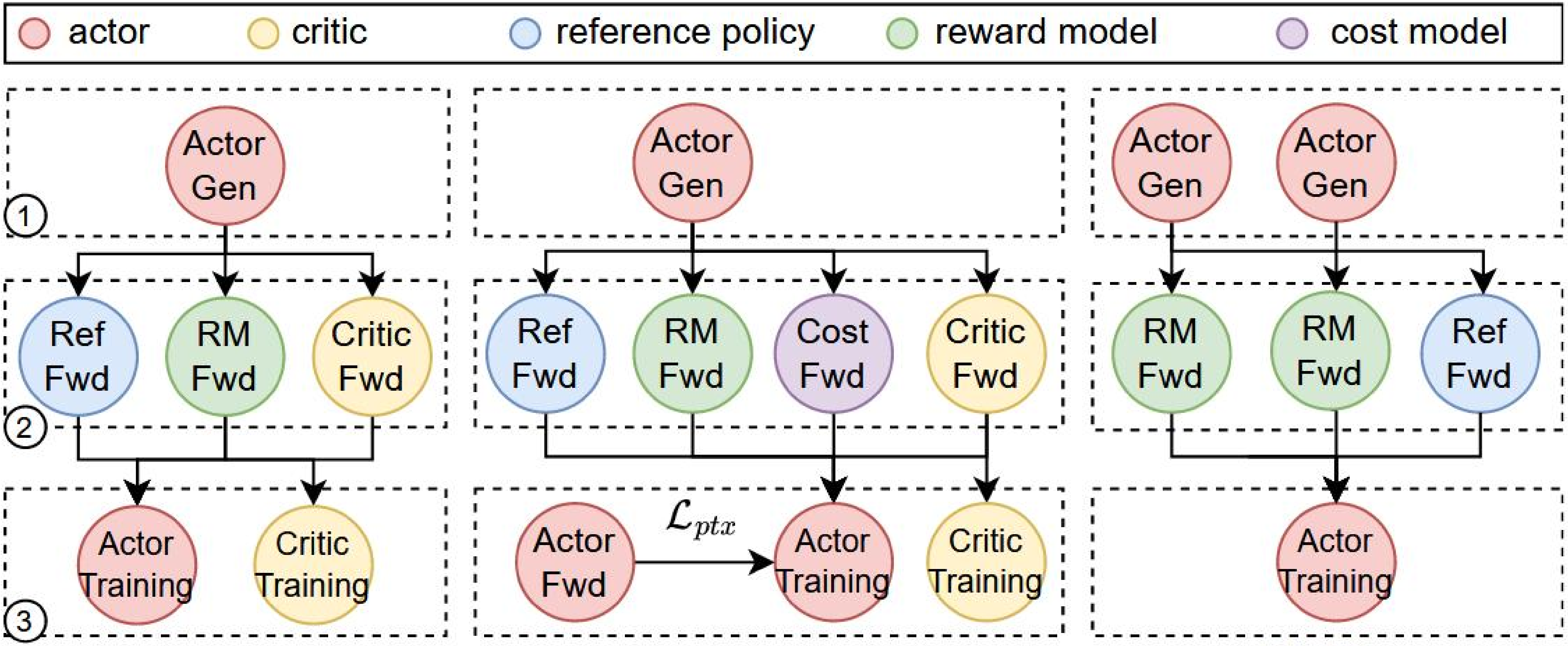
- SFT
- 训练奖励模型
- RLHF



PPO workflow



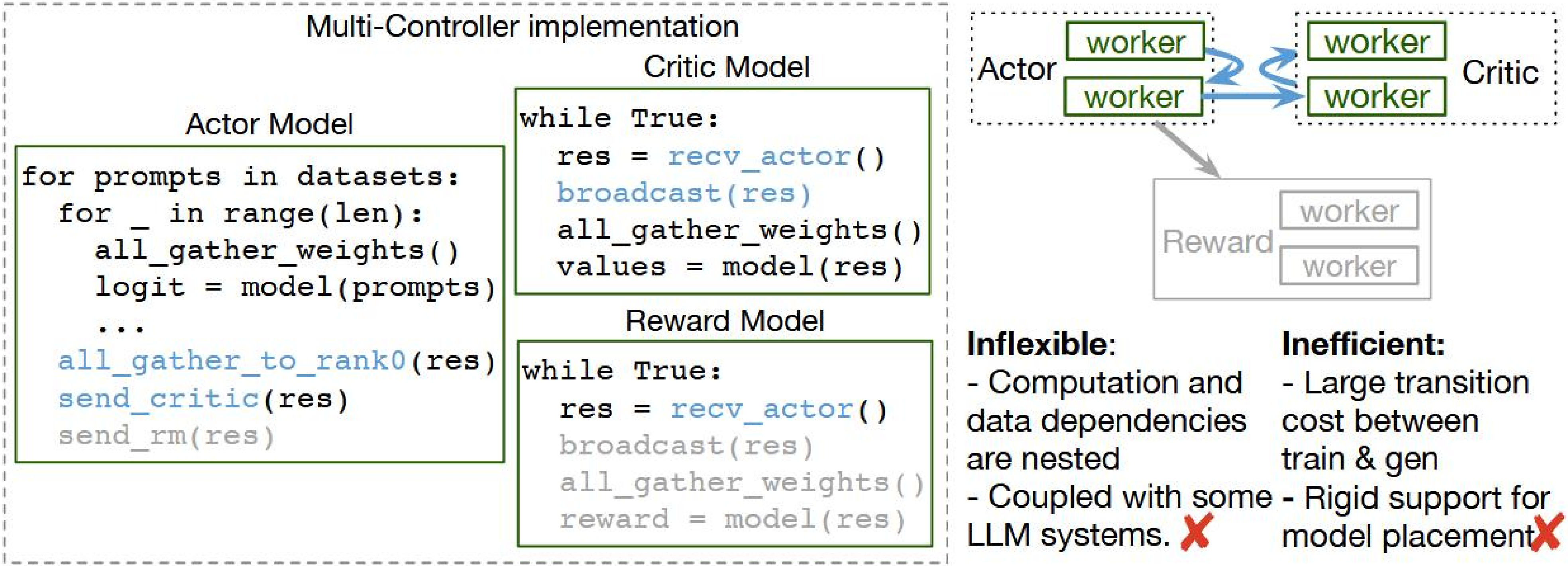
PPO Safe-RLHF ReMax



(a) PPO (b) Safe-RLHF (c) ReMax

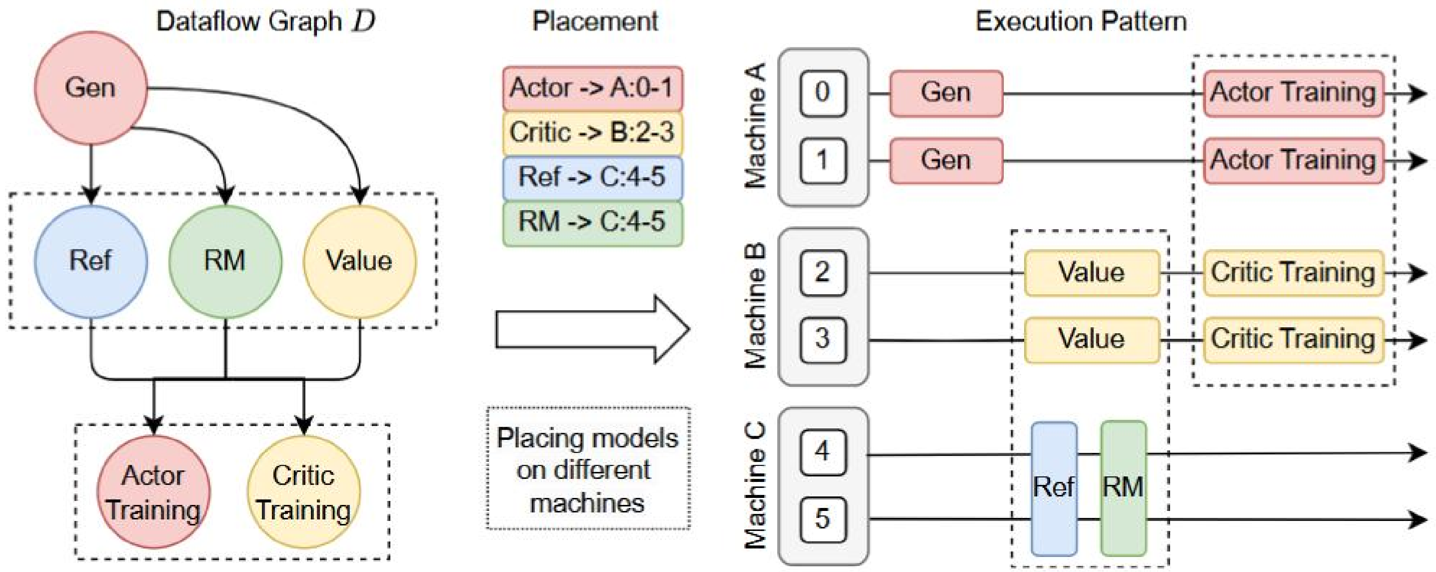
Motivation-Programming Model for Distributed ML

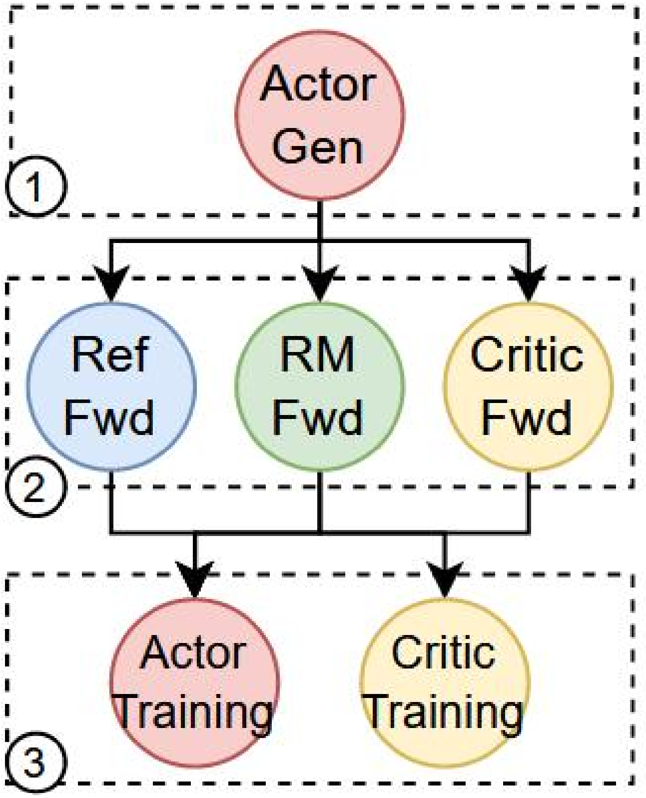
- 不断有传统强化学习算法整合到RLHF领域
- 需要灵活表示 RLHF 数据流图以适应不同的算法需求
- 两种数据流控制方式：单控制器和多控制器



Motivation-RLHF Characteristics

- 异构模型负载
  - 不同模型、不同阶段具有不同的内存占用的计算需求
- Actor Model训练和生成阶段的计算不均衡
  - actor gen 占60%
- 不同的模型放置需求



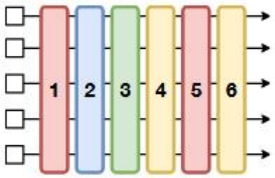
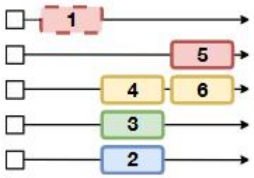
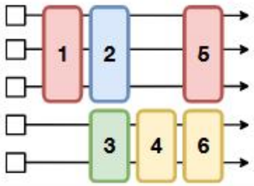


(a) PPO

Motivation-Limitations of existing RLHF systems

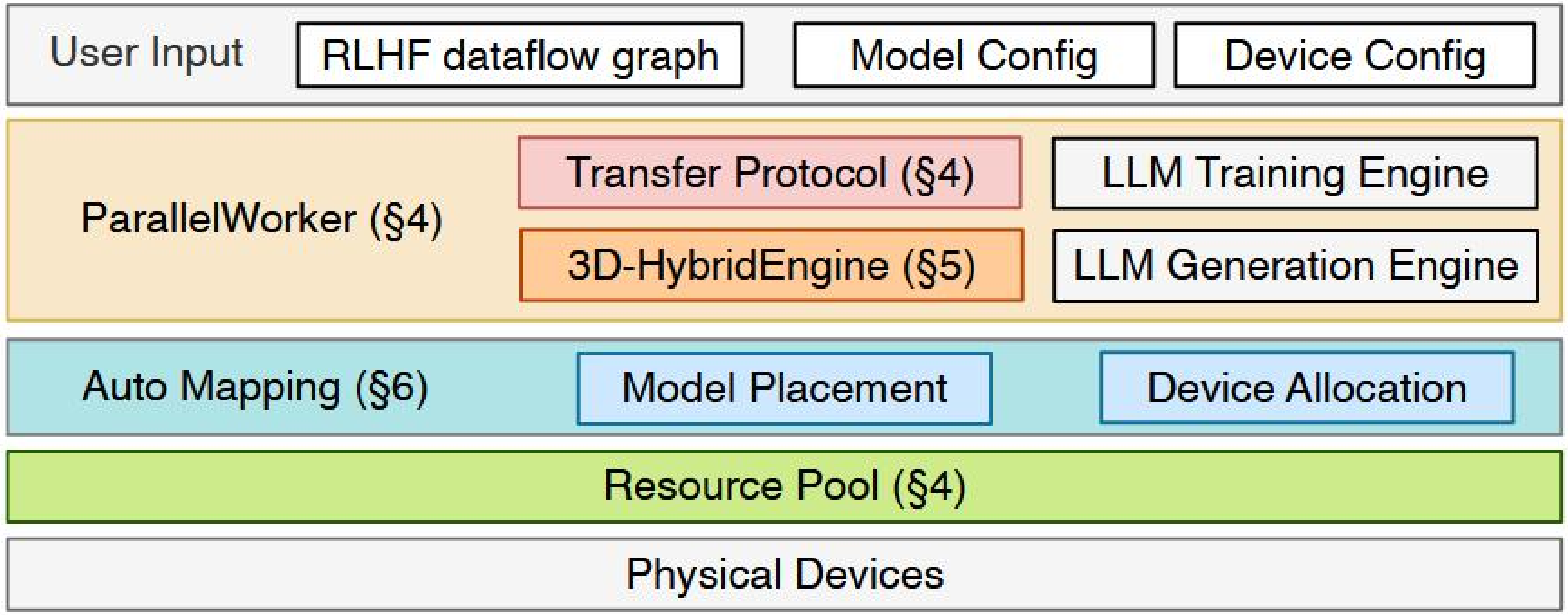
- 对各种 RLHF 数据流图的支持不够灵活
- RLHF执行效率低下

Table 1. Comparison of RLHF frameworks. Figures illustrate execution of one PPO iteration. Numbers 1-6 represent response generation, reward model inference, reference model inference, critic inference, actor training, and critic training, respectively.

RLHF system	DeepSpeed-Chat	OpenRLHF	NeMo-Aligner	**HybridFlow**
Parallelism	Training: ZeRO Generation:TP	Training: ZeRO Generation:TP	3D Parallelism for both training and generation	**Training: 3D, ZeRO, FSDP** **Generation: 3D Parallelism**
Actor weights in training & generation	Model resharding from ZeRO to TP	Using two copies of actor weights for the two stages	Using identical model partition in two stages (shared weights)	**Zero-redundancy** **model resharding**
Model Placement	Colocate all models on the same set of devices	Each model placed on separate devices	Actor/Ref colocated on some GPUs Critic/RM colocated on other GPUs	**Support various** **model placement**
Execution Pattern 				**Support various** **execution patterns**

HybridFlow Overview

- 提出了一个层次化的混合编程模型（单控制器+多控制器），便于构建 RLHF 数据流
- 设计了 3D-HybridEngine，高计算效率执行Actor Model的训练和生成，并在训练阶段和生成阶段之间实现零内存冗余转换
- 设计一种有效的映射算法，自动识别 RLHF 数据流中每个节点的优化 GPU 分配和放置



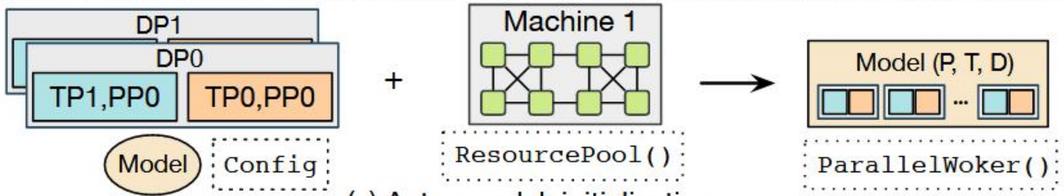
Hybrid Programming Model

- 节点内：封装分布式程序
- 节点间：统一模型间数据重分配实现
- 促进灵活的模型放置
- 异步数据流执行，不同RLHF算法的灵活实现

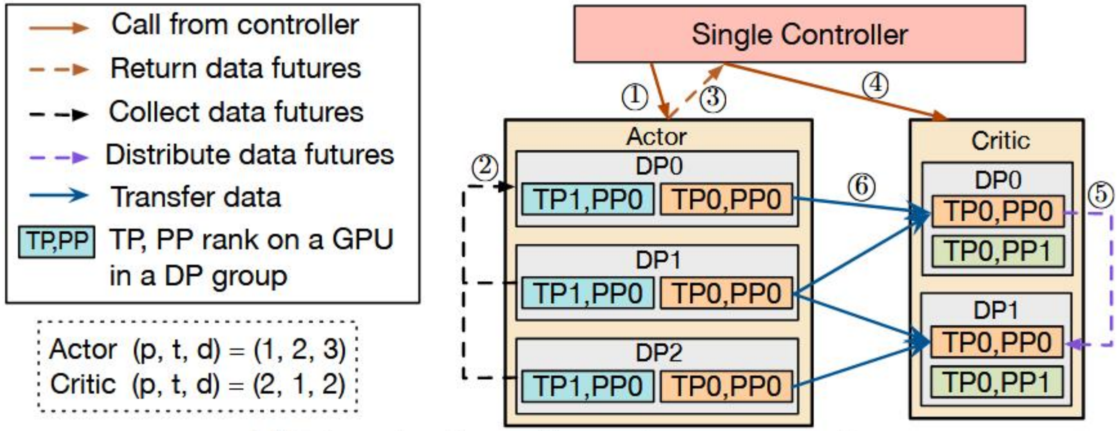
```
# Initialize cost model by reusing the RewardWorker
cost = RewardWorker(cost_config, resource_pool)
...  # omit other models initialization
algo_type = "Safe-RLHF" # specify different RLHF numerical computation.
# Examples of PPO and Safe-RLHF
for (prompts, pretrain_batch) in dataloader:
    # Stage 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

```
class ActorWorker(3DParallelWorker):
    # An example of distributed computation function
    @register(transfer_mode=3D_PROTO)
    def update_actor(self, prompts: DataProto):
        ...
# Allocate devices for a ResourcePool
resource_pool = ResourcePool([n_gpus_per_machine] * n_machines)
# Map the model to allocated devices and init model
actor_model = ActorWorker(actor_config, resource_pool)
```



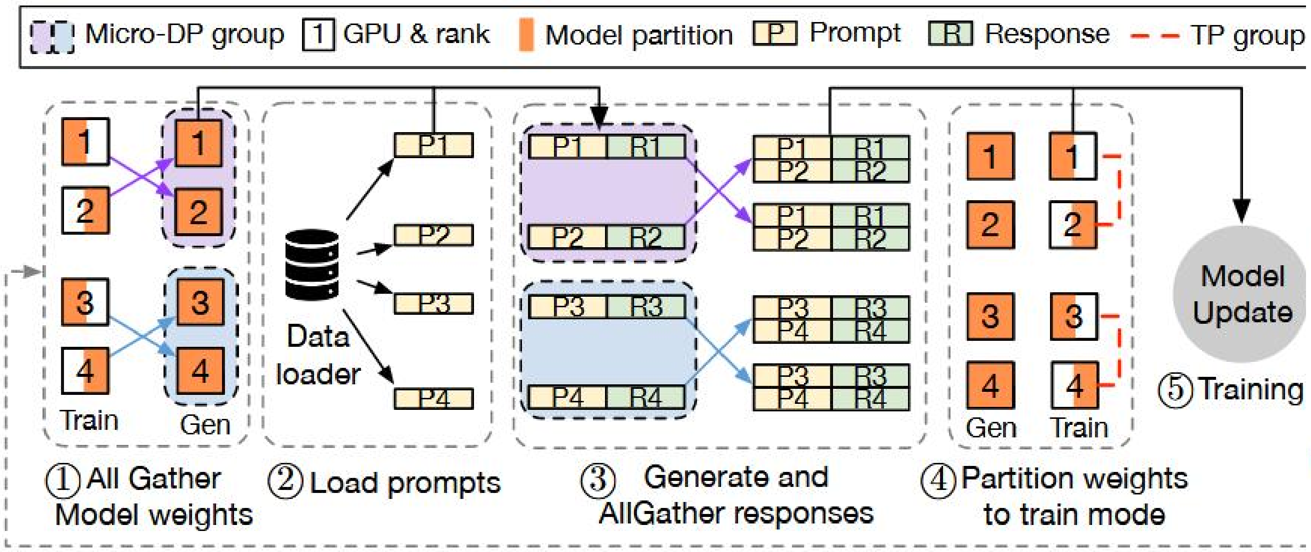
(a) Actor model initialization

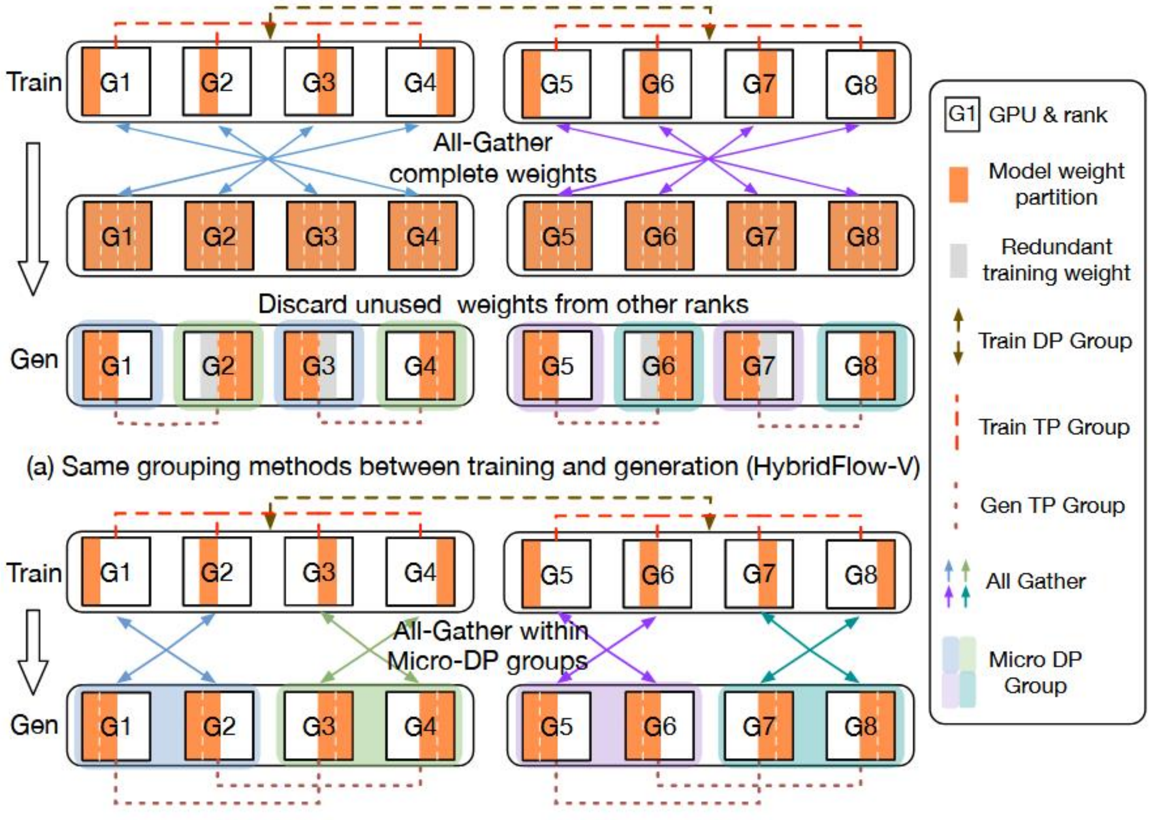


(b) Data resharding and asynchronous execution

3D-HybridEngine

- workflow
- Zero redundancy model resharding





(a) Same grouping methods between training and generation (HybridFlow-V)

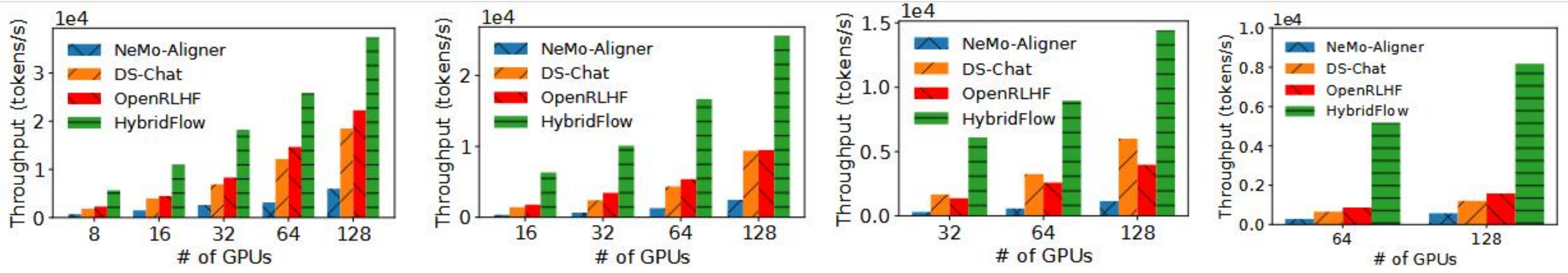
(b) Optimized parallel grouping methods (HybridFlow)

Auto Device Mapping

Algorithm 1 Device Mapping for an RLHF Dataflow

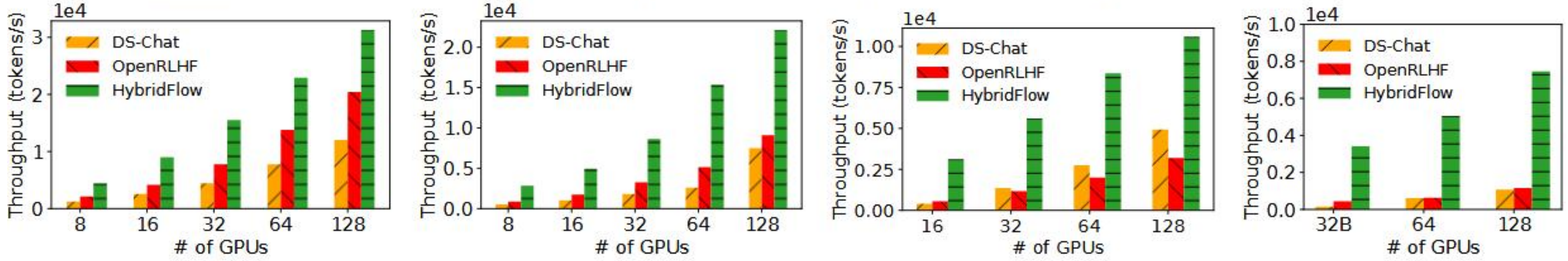
```
Input: RLHF dataflow graph D, LLMs in RLHF dataflow L=[l_1, l_2, ..., l_k], workload W of LLMs in RLHF dataflow, total # of GPUs N, memory capacity per GPU Q
Output: device mapping of models in RLHF dataflow
𝒫 ← get_placements(D, L, N)
C* ← ∞
best_mapping ← ∅
for all plm ∈ 𝒫 do
    C_plm ← ∞
    best_plm_alloc ← ∅
    A_min ← get_min_alloc(plm, Q, N)
    for all A ∈ enum_alloc(N, A_min) do
        L̂ ← []
        for all set ∈ plm do
            for all l ∈ set do
                l̂ ← auto_parallel(A, A_min, l, W)
                L̂.append(l̂)
        plm.update(L̂)
        C_alloc ← d_cost(D, plm, W)
        if C_alloc < C_plm then
            C_plm ← C_alloc
            best_plm_alloc ← (plm, A)
    if C_plm < C* then
        C* ← C_plm
        best_mapping ← best_plm_alloc
return best_mapping
Procedure d_cost(D, plm, W):
    s ← number of stages in D
    c ← [0] × s  // Initialize latency for each stage to 0
    for all set ∈ plm do
        c_g ← [0] × s
        for all i ∈ {0, ..., s − 1} do
            for all l̂ ∈ set do
                c_g[i] ← c_g[i] + simu(l̂, W[i])
            c[i] ← max{c[i], c_g[i]}
    return sum(c)
```

Evaluation-Throughput



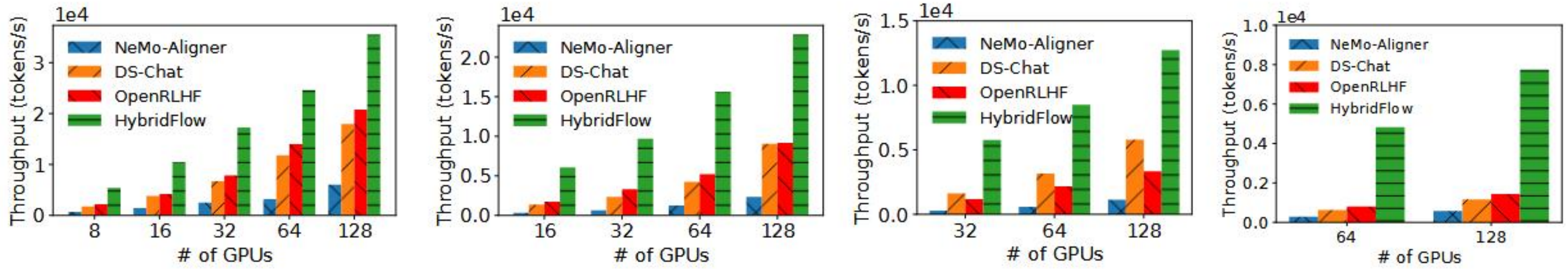
(a) 7B (1.68×~8.63×) (b) 13B (2.70×~18.96×) (c) 34B (2.41×~20.57×) (d) 70B (5.17×~17.98×)

Figure 9. PPO throughput. Numbers in parentheses are HybridFlow speedups compared with baselines.



(a) 7B (1.53×~2.56×) (b) 13B (2.49×~3.66×) (c) 34B (2.14×~4.80×) (d) 70B (6.46×~9.78×)

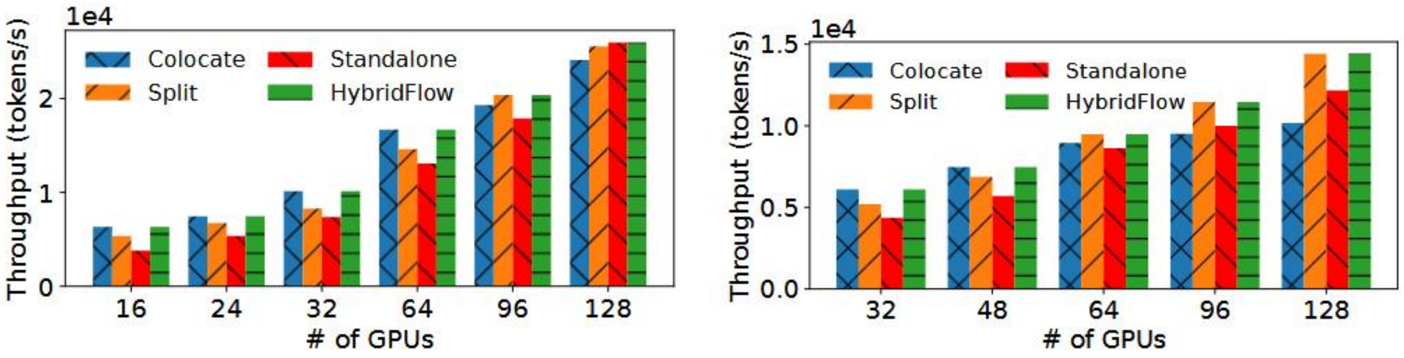
Figure 10. ReMax throughput. Numbers in parentheses are HybridFlow speedups compared with baselines



(a) 7B (1.71×~12.87×) (b) 13B (2.49×~18.47×) (c) 34B (2.20×~19.76×) (d) 70B (4.89×~16.86×)

Figure 11. Safe-RLHF throughput. Numbers in the parentheses are HybridFlow speedups compared with the baselines

Evaluation-Model Placement



(a) 13B (b) 34B

Figure 12. Throughput of HybridFlow under different placements

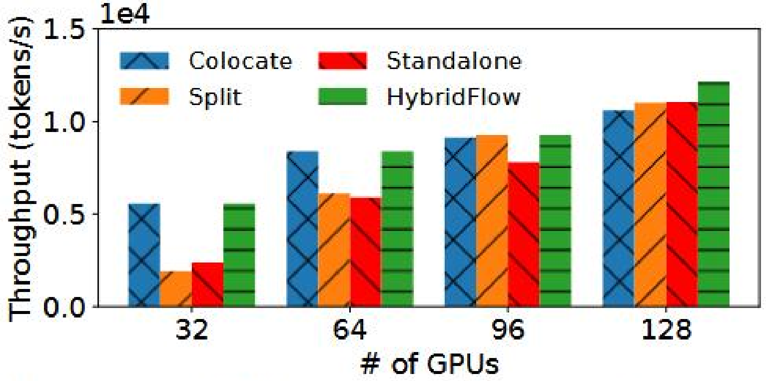
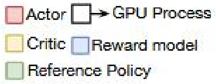
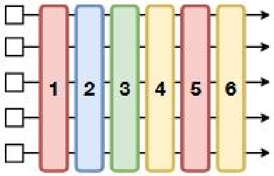
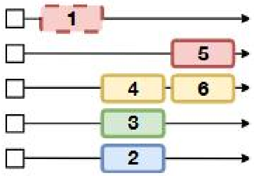
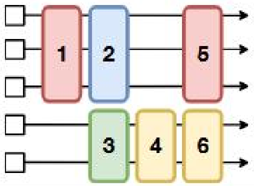
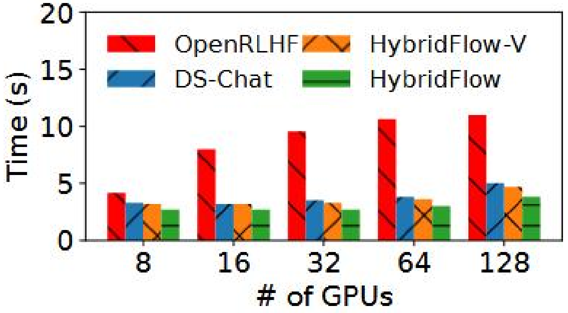


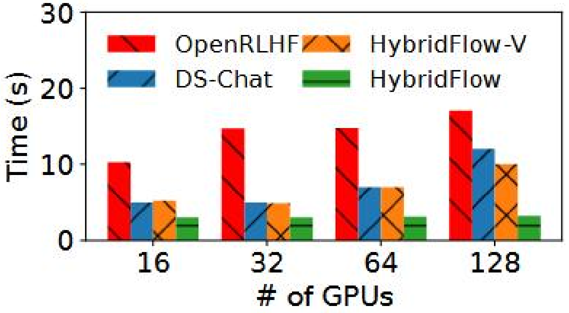
Figure 13. Placement comparison under 13B actor and reference policy & 70B critic and reward model.

RLHF system	DeepSpeed-Chat	OpenRLHF	NeMo-Aligner	**HybridFlow**
Execution Pattern 				**Support various execution patterns**

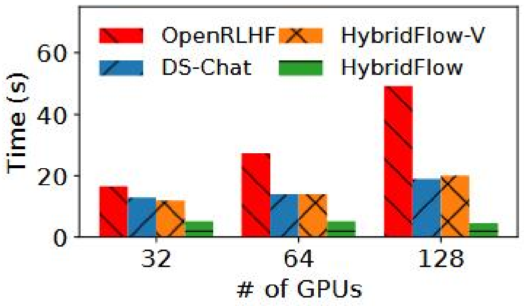
Evaluation-Transition Time



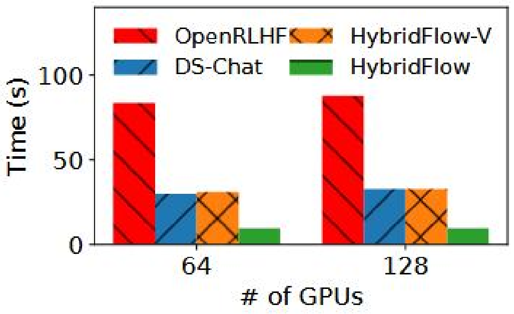
(a) 7B (T_g=2, P_g=1, T=8,P=1)



(b) 13B (T_g=4, P_g=1, T=8,P=1)



(c) 34B (T_g=8, P_g=1, T=8,P=4)



(d) 70B (T_g=8, P_g=1, T=8,P=8)

Evaluation-Algorithm Time

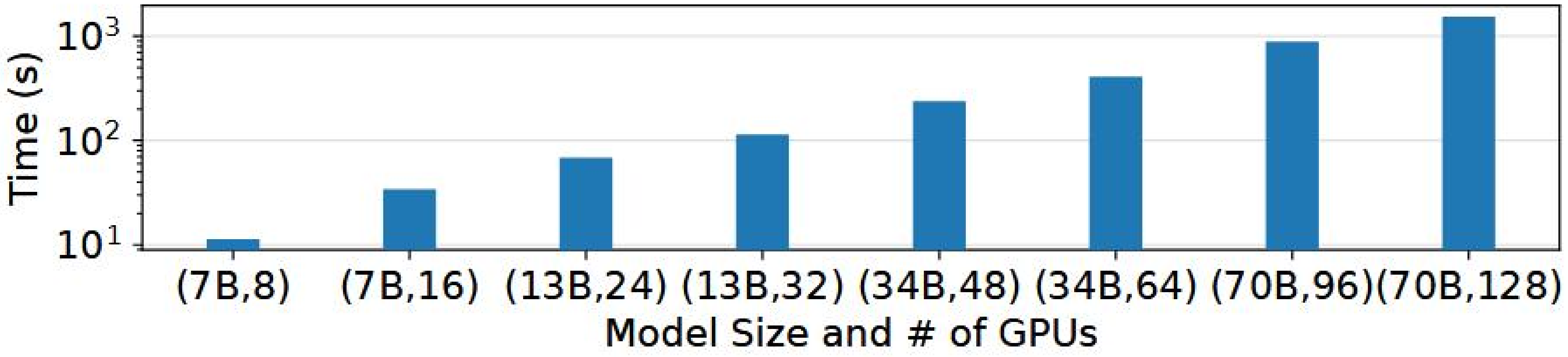


Figure 16. Runtime of device mapping algorithm. The model size and # of GPUs are simultaneously scaled.

Latest Frameworks

- Deepspeed-Chat
- OpenRLHF
- ReaLHF
- RLHFuse
- …