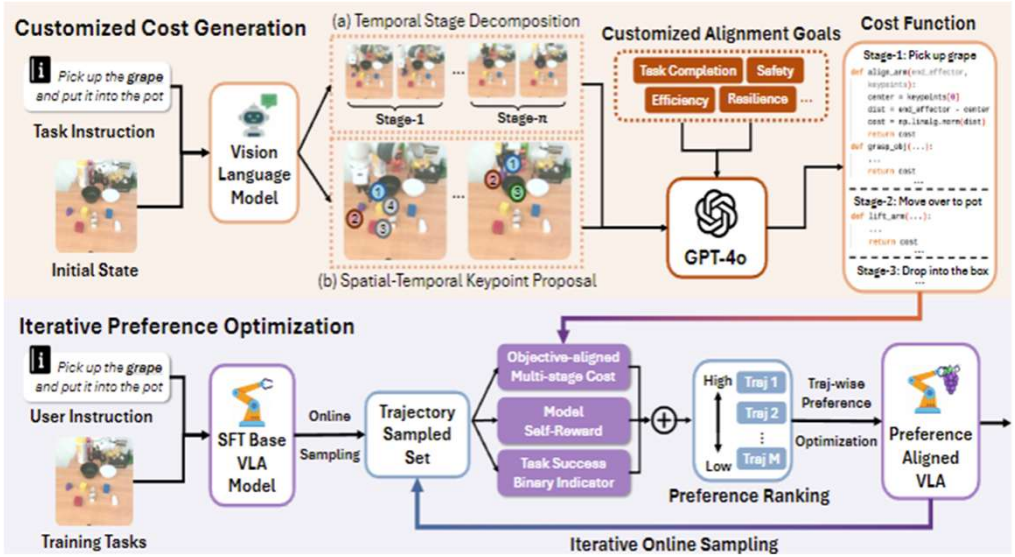
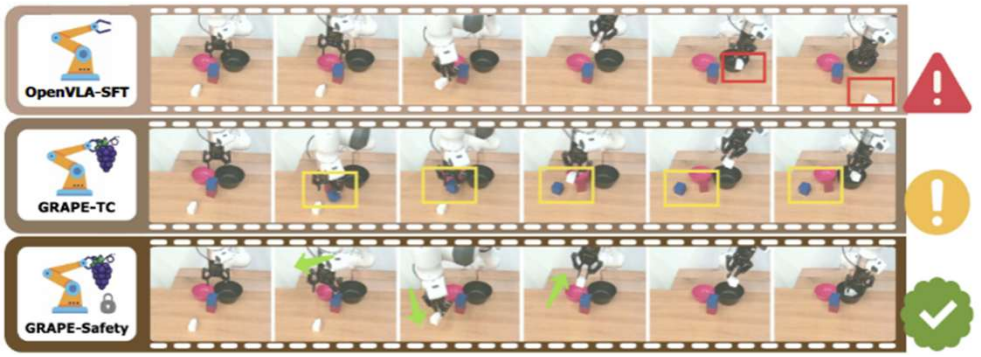
GRAPE: Generalizing Robot Policy via Preference Alignment





- Motivate:
  使用Trajectory-wised Preference Optimization和隐式奖励建模, 提高泛化性,
  支持自定义目标

GRAPE: Generalizing Robot Policy via Preference Alignment

- external reward:
  - 使用VLM对task进行分解, 得到不同阶段
  - 根据要求(安全性, …)使用LLM为每个阶段生成不同的cost function
  - $$R_{\text{ext}}(\zeta) = \prod_{i=1}^{\mathbf{S}} e^{-C^{S_i}(\{\kappa_{S_i}\})}$$
- self reward:
  - policy的对数似然, 用于提高指定任务下出现正确action的概率
  - $$R_{\text{self}}(\zeta) = \log(\pi(\zeta, q)) = \log(\prod_{i=1}^{T} \pi(a_i \mid (o_i, q)))$$
- GCPG reward:
  - 将上面的两个reward和一个指示是否完成的指示变量加权求和, 作为最终的reward
  - 这个reward 用于后续的TPO优化
  - $$R_{\text{GCPG}}(\zeta) = \lambda_1 R_{\text{self}}(\zeta) + \lambda_2 R_{\text{ext}}(\zeta) + \lambda_3 I_{\text{success}}(\zeta)$$

GRAPE: Generalizing Robot Policy via Preference Alignment

- 使用Trajectory-wised Preference Optimization, 定义偏好:

$$r(\zeta, q) = \beta \log \frac{\pi_\theta(\zeta \mid q)}{\pi_{\text{ref}}(\zeta \mid q)} + \beta \log Z(\zeta).$$

$$P(\zeta_w \succ \zeta_l) = \frac{\exp(r(\zeta_w), q)}{\exp(r(\zeta_w), q) + \exp(r(\zeta_l), q)}.$$

- 定义TPO的loss

$$\mathcal{L}_{\text{TPO}} = -\mathbb{E}_{(\zeta_w, \zeta_l) \sim \mathcal{D}} \left[\log \sigma \left(\beta \left(\log \frac{\pi_\theta(\zeta_w)}{\pi_{\text{ref}}(\zeta_w)} - \log \frac{\pi_\theta(\zeta_l)}{\pi_{\text{ref}}(\zeta_l)} \right) \right) \right]$$

- 优点:
  - 通过step-wise的human preference, 在trajectory-wise的层面对齐policy
  - 使用梯度下降反向传播, 稳定policy, 引导向最终目标
  - 提高泛化能力

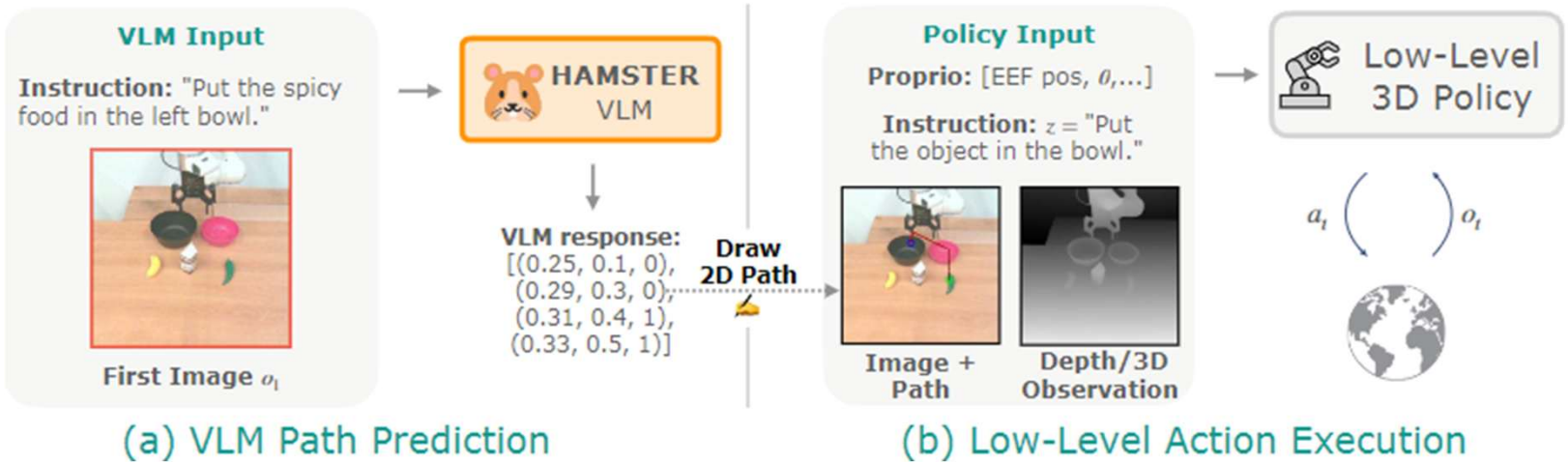
Algorithm 1 GRAPE Iterative Preference Optimization

Require: Base VLA policy π_θ, a collection of task instructions $Q = \{q_i\}$, stage decomposer $\mathcal{M}_D$, max iterations K, reward weights $\{\lambda_1, \lambda_2, \lambda_3\}$, stage-wise keypoints $\{\kappa_{S_i}\}$ cost functions $\{C_j^{S_i}\}$ and thresholds $\{\tau_j^{S_i}\}$

Ensure: policy π^* aligned towards customized objective

for $k = 1, \ldots, K$ **do**
  Sample trajectories $\mathcal{D}^k = \{\zeta_i\}_{i=1}^M$ using π_θ with Q
  for trajectory $\zeta \in \mathcal{D}^k$ **do**
    Decompose ζ into multiple stages S {Eq. (7)}
    Compute the cost for each stage C_{S_i}
    Calculate external reward $R_{\text{ext}}(\zeta)$ {Eq. (8)}
    Compute policy self-reward $R_{\text{self}}(\zeta)$ {Eq. (10)}
    Examine task success $I_{\text{success}}(\zeta)$ {Eq. (11)}
    Aggregate GCPG reward $R_{\text{GCPG}}(\zeta)$ {Eq. (9)}
  end for
  Rank $\mathcal{D}^k$ by their $R_{\text{GCPG}}(\zeta)$ rewards
  Pair $\{\zeta_w, \zeta_l\}$ from top-m and bottom-m trajectories
  Update π_θ using TPO loss {Eq. (5)}
end for

HAMSTER: Hierarchical Action Models for Open-World Robot Manipulation

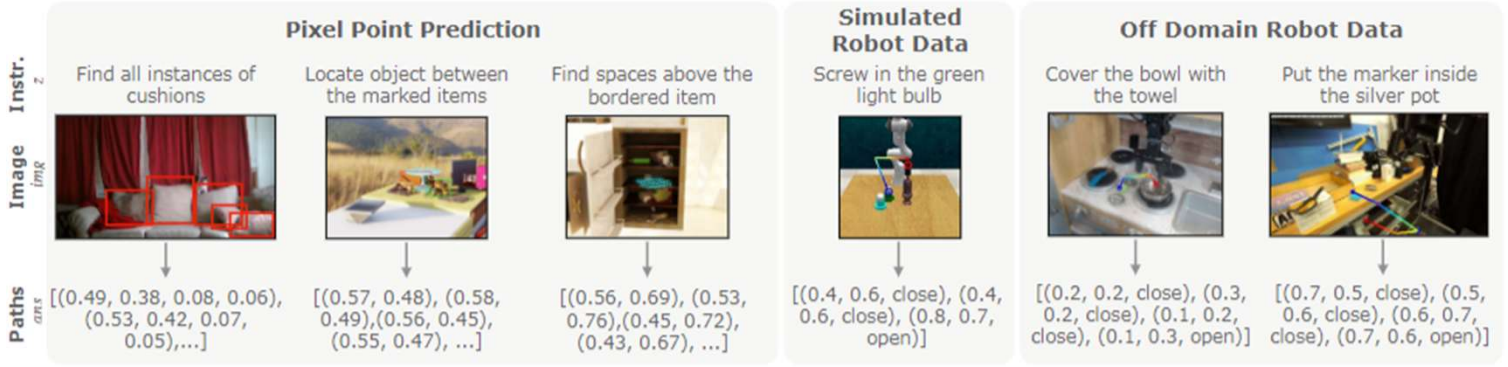


- Motivate: 结合大模型的泛化性和小模型的效率与灵活性

HAMSTER: Hierarchical Action Models for Open-World Robot Manipulation

- High-level VLM: 输入camera image 和 instruction, 预测end-effector在图片上的2D轨迹
- 防止2D path过长, 使用Ramer-Douglas-Peucker algorithm进行简化

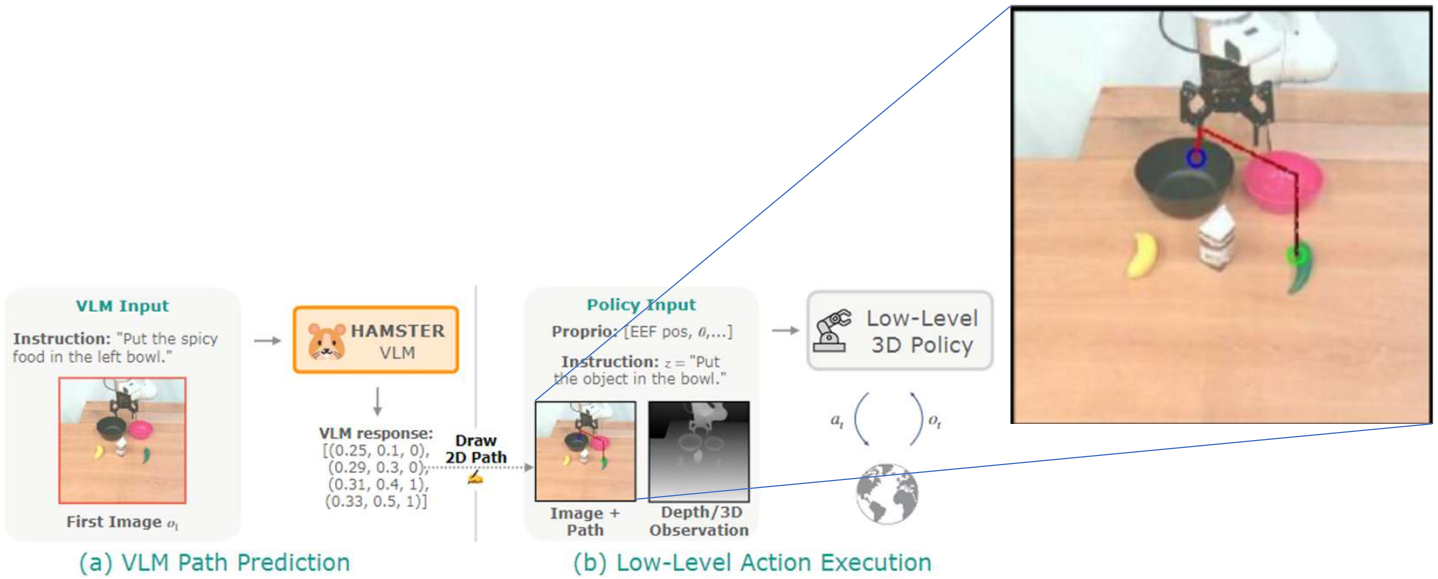
- 数据集:
  - **Pixel Prediction**: 使用RoboPoint数据集, 输入图片, 输出point的数组
  - **Simulate Robot Data:** 使用RLBench, 生成数据, 然后转换成第一帧图片上的轨迹
  - **Real Robot Data:** 使用Bridge, Open X-Embodiment, Droid, 使用RLBench转换成类似数据



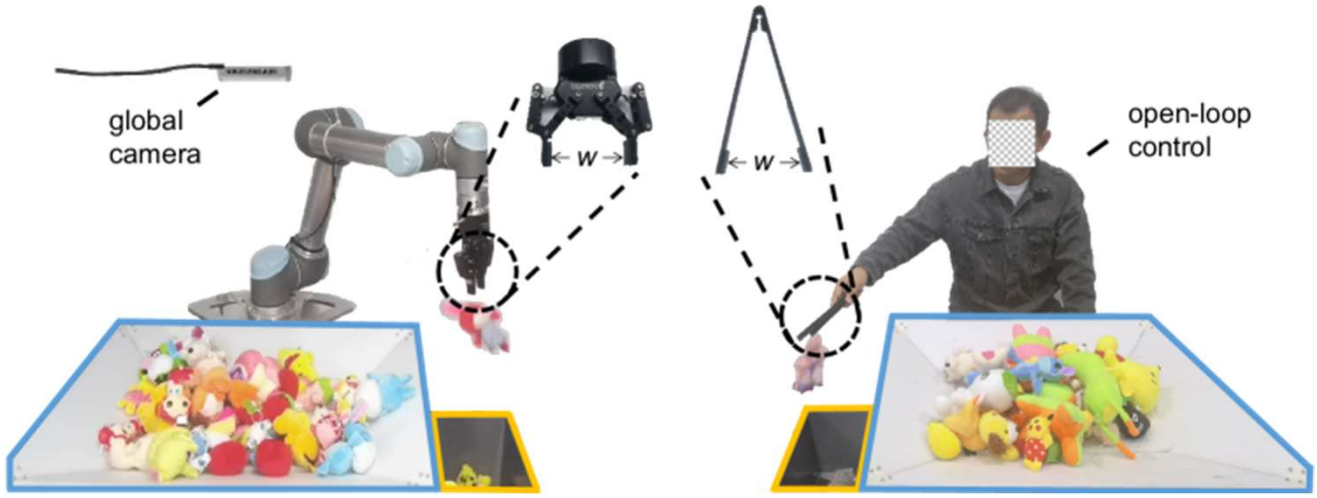
HAMSTER: Hierarchical Action Models for Open-World Robot Manipulation

- Low-level Policy Module: 输入当前状态, 3D感知信息, instruction 和 2D path, 生成robot actions

- 数据集:
  - 正常的数据集包含 state, observation, instruction, actions
  - 使用上面类似 **Simulation Robot Data** 的方法, 添加2D path



AnyGrasp: Robust and Efficient Grasp Perception in Spatial and Temporal Domains

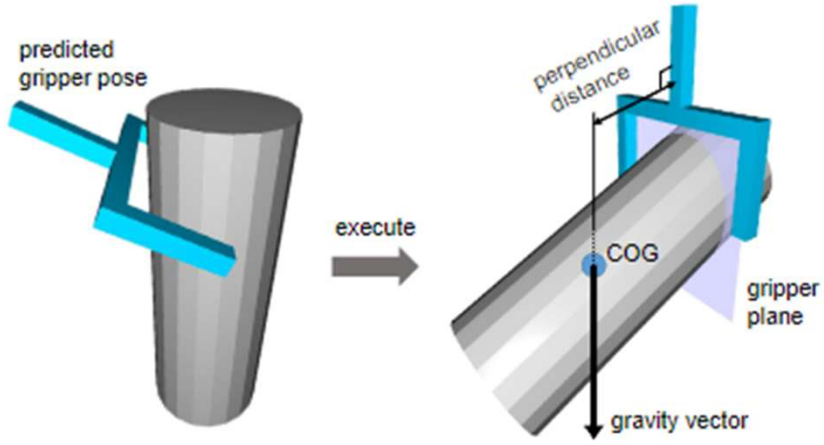


- Motivate: 结合物体重心感知弥合机器人与人类对于抓取的感知的差距

AnyGrasp: Robust and Efficient Grasp Perception in Spatial and Temporal Domains

- 定义Grasp Pose: $\mathcal{G} = [\mathbf{R}\ \mathbf{t}\ w]$
- 目标:

$$\{\mathcal{G}_1^\star, \mathcal{G}_2^\star, \cdots, \mathcal{G}_n^\star\}^{(t)} = \arg\max_{|\mathbf{G}^t|=n} \sum_{\mathcal{G}_i \in \mathbf{G}^t} \text{Prob}(s=1|\mathcal{E}^t, \mathcal{P}^t, \mathcal{G}_i),$$
$$\text{s.t. } dist(\mathcal{G}_k^t, \mathcal{G}_k^{t-1}|\mathcal{E}^t, \mathcal{E}^{t-1}) \leq \delta, \forall\ \mathcal{G}_k^{t-1} \in \mathbf{G}^{t-1},$$

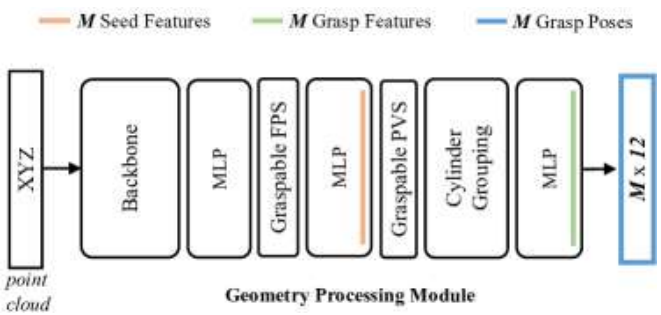


- 定义两个pose的距离:

$$\Delta\mathbf{R} = \arccos \frac{\text{trace}(\mathbf{R}_1^\top \mathbf{R}_2) - 1}{2},$$
$$\Delta\mathbf{t} = ||\mathbf{t}_1 - \mathbf{t}_2||,$$
$$d(\mathcal{G}_1, \mathcal{G}_2) = \frac{\Delta\mathbf{t}}{w_{\max}} + \gamma\frac{\Delta\mathbf{R}}{\pi},$$

AnyGrasp: Robust and Efficient Grasp Perception in Spatial and Temporal Domains

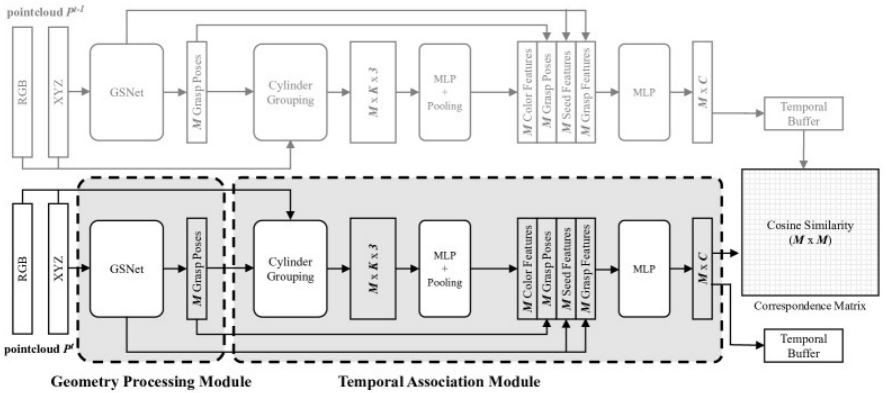
- 1. 输入点云P, 使用3D-Conv提取features
- 2. 使用MLP生成object mask和指示可抓取概率的heatmap
- 3. 执行Graspable FPS(可抓取最远端采样, Graspable Farthest Point Sampling): 根据object mask和heatmap, 从场景中采样M个seed point
- 4. 使用MLP生成300个view score, 给Graspable PVS(可抓取概率视图, Graspable Probabilistic View Selection), 选择抓取的view
- 5. Cylinder Grouping圆柱体分组模块沿该view对object的局部几何特征进行分组
- 6. 使用MLP对每个分组预测48个grasp pose的grasp score, 这些pose由12个in-plane的旋转和4个approach depth和48个grasp width组成



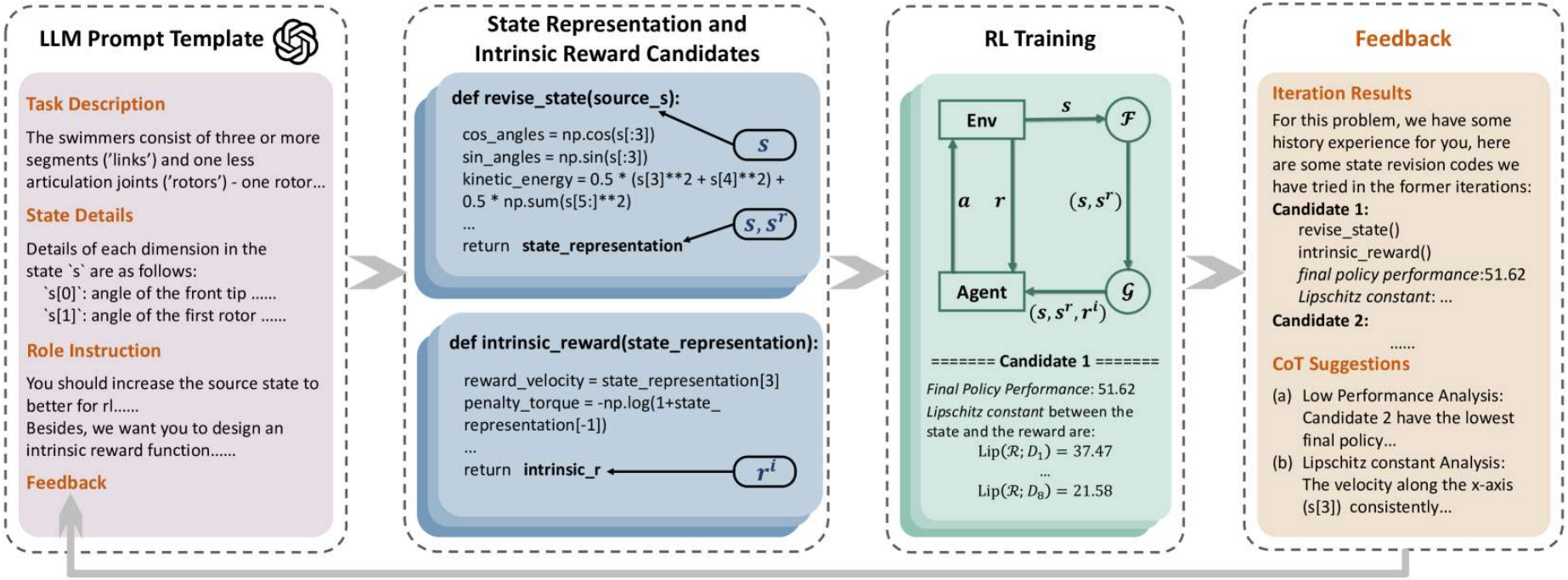
AnyGrasp: Robust and Efficient Grasp Perception in Spatial and Temporal Domains

- 1. 给定输入点云, Geometry processing module返回seed point和features
- 2. 使用cylinder grouping的方法对RGB信息分组, 然后使用MLP forward+Pooling, 得到color feature
- 3. 将color feature(刚刚获得的), seed feature, pose feature(GSNet中生成的), pose(GSNet最终结果)拼接, 给MLP, 生成特征向量
- Loss: $$L = \sum_{\mathcal{G}_i^1 \in \mathbf{G}_1} \frac{-1}{|\mathbf{P}(i)|} \sum_{\mathcal{G}_k^2 \in P(i)} \log \frac{\exp(s_{\text{corres}}(\mathcal{G}_i^1, \mathcal{G}_k^2)/\tau)}{\sum_{\mathcal{G}_j^2 \in \mathbf{G}_2} \exp(s_{\text{corres}}(\mathcal{G}_i^1, \mathcal{G}_j^2)/\tau)},$$

- 当前的feature vector和buffer中上一步的feature vectors计算, 找到top-n的feature vectors, 对应的pose就是所需要的pose



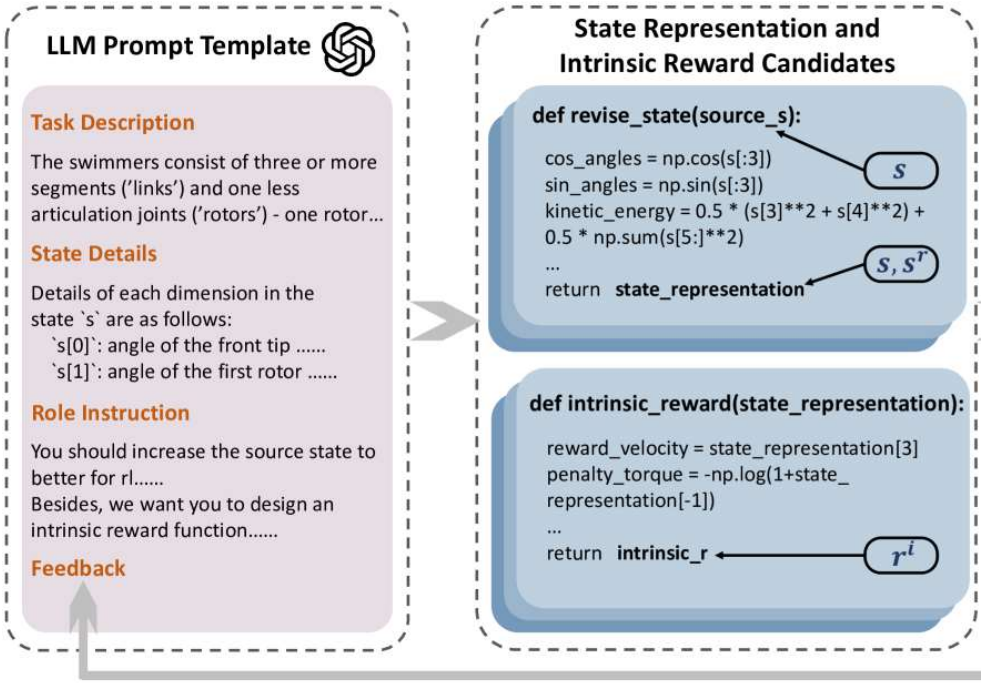
LLM-Empowered State Representation for Reinforcement Learning



- Motivate: 使用LLM增强state的表达, 获取内在隐藏的表达, 增强value network从state到reward的准确性

LLM-Empowered State Representation for Reinforcement Learning

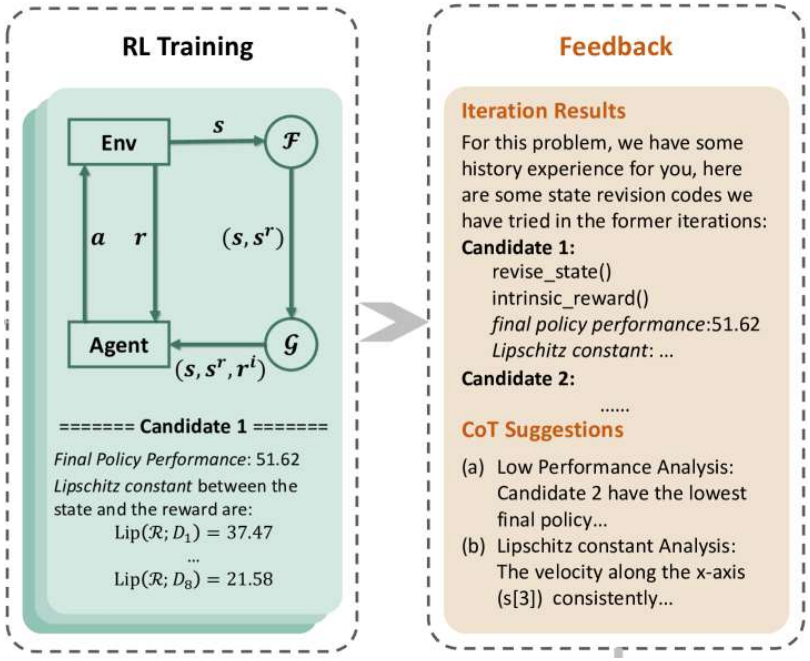
- 基于LLM嵌入的广泛的知识和先验信息, 使用LLM生成state repr.
- prompt输入分成4个部分:
  - 1. Task Description: 当前任务的描述
  - 2. State Details: 原始state的每一个维度所代表的含义
  - 3. Role Instruction: 要求LLM生成任务相关的状态表达和intrinsic reward代码
  - 4. Feedback: 历史信息
- 目标是通过LLM生成一个python函数, 将原始空间的state映射到LLM-Empowered state representation space中.
- 然后, 使用LLM基于state生成一个reward function



LLM-Empowered State Representation for Reinforcement Learning

- Lipschitz Constant: $\mathrm{Lip}(u;\mathcal{X}_0) = \sup_{x_1,x_2\in\mathcal{X}_0} \frac{\|u(x_1)-u(x_2)\|_2}{\|x_1-x_2\|_2}$.
- 使用Spectral Norm去估计Lipschitz Constant

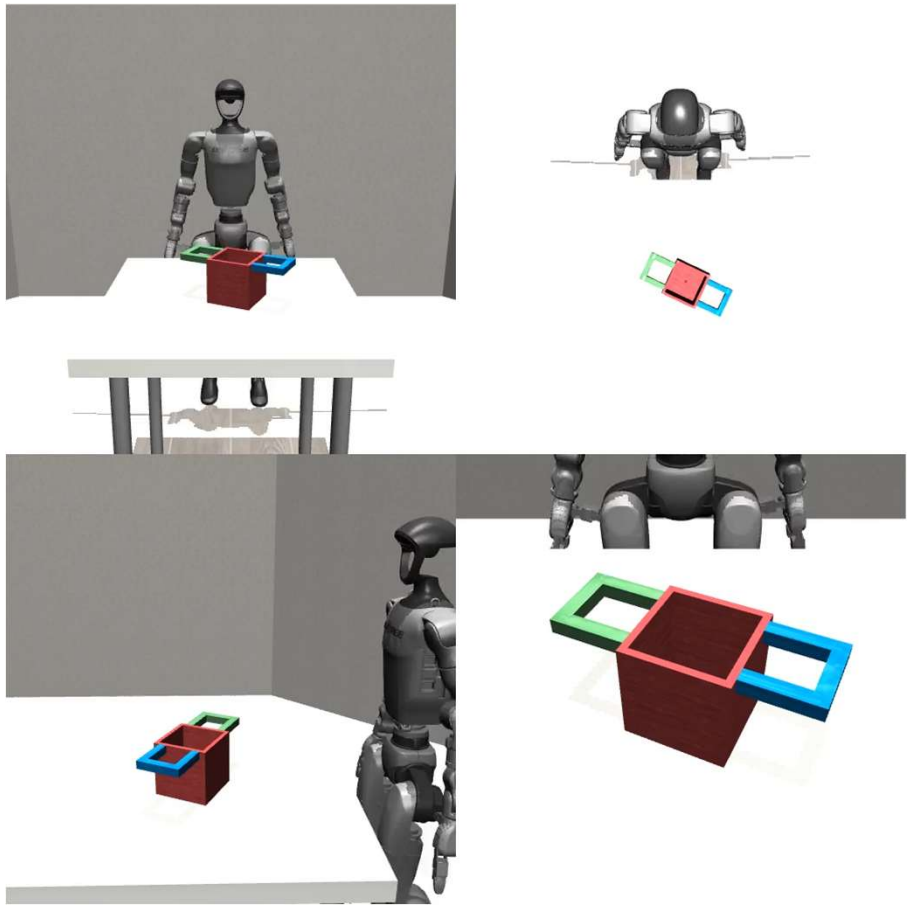
- 在每一个training iteration结束时, 将Lipschitz Constant数组和policy preformance作为Feedback提供给LLM, 根据Feedback调整生成的函数



Robosuite + G1

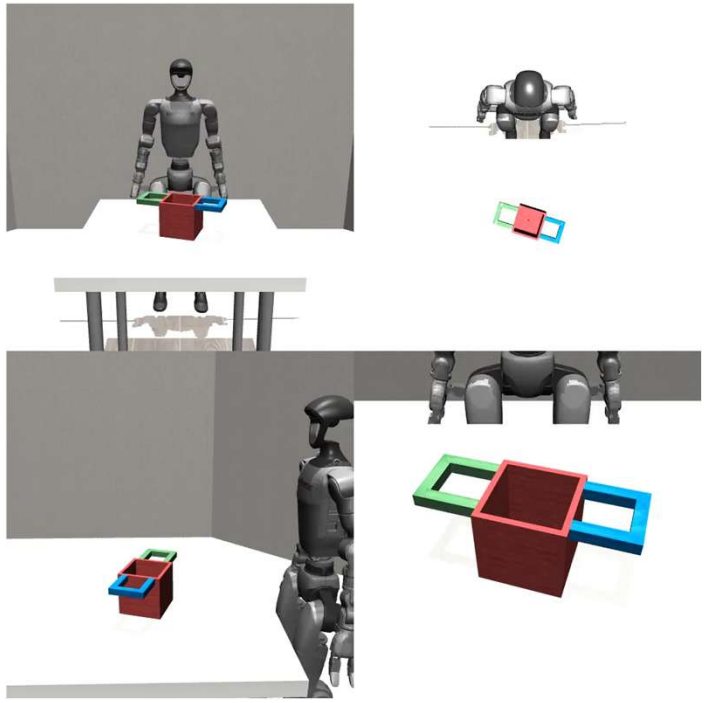
Robosuite

- 集成G1到robosuite中
- 成功运行random action



PPO Training

- 完成一轮训练
- 效果不佳: 无法保持直立状态



rollout/	
ep_len_mean	1e+03
ep_rew_mean	0.000889
time/	
fps	98
iterations	730
time_elapsed	15162
total_timesteps	1495040
train/	
approx_kl	0.038631216
clip_fraction	0.224
clip_range	0.2
entropy_loss	-207
explained_variance	-20.8
learning_rate	0.0005
loss	-20.8
n_updates	7290
policy_gradient_loss	-0.0825
std	37.8
value_loss	0.000812

```
Mean reward over 5 episodes: 0.00 +/- 0.00

Running demonstration episodes...
[robosuite WARNING] The config has defined for the controller "head", but the robot does not have this component. Skipping, but make sure this is intended.Removing the controller config for head from self.part_contr
oller_config. (robot.py:151)
[robosuite WARNING] The config has defined for the controller "base", but the robot does not have this component. Skipping, but make sure this is intended.Removing the controller config for base from self.part_contr
oller_config. (robot.py:151)
Demo episode 1 finished with reward: 0.00 in 1000 steps
[robosuite WARNING] The config has defined for the controller "head", but the robot does not have this component. Skipping, but make sure this is intended.Removing the controller config for head from self.part_contr
oller_config. (robot.py:151)
[robosuite WARNING] The config has defined for the controller "base", but the robot does not have this component. Skipping, but make sure this is intended.Removing the controller config for base from self.part_contr
oller_config. (robot.py:151)
Demo episode 2 finished with reward: 0.00 in 1000 steps
```