Eureka: Human-Level Reward Design via Coding Large Language Models

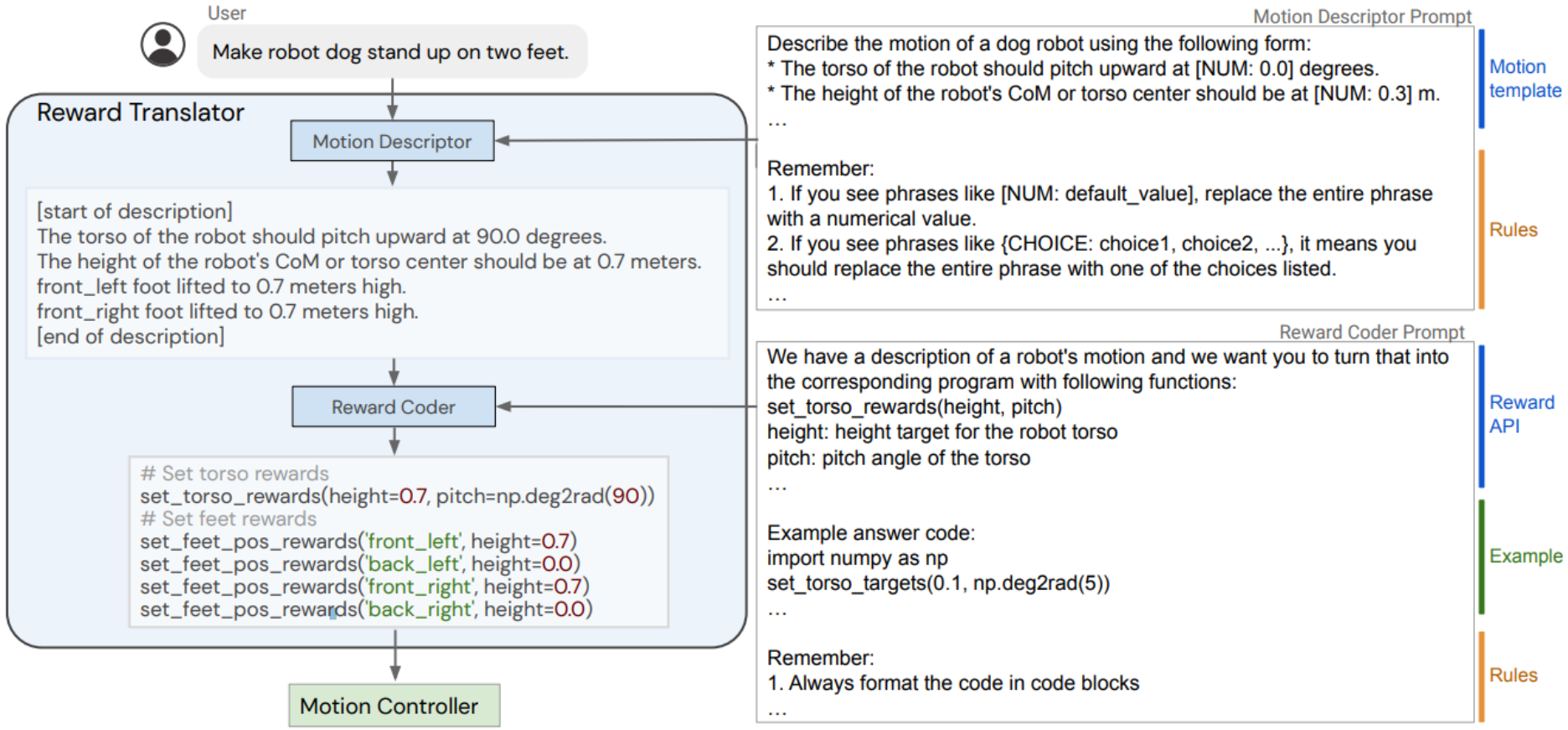
동아대학교
2373552
정 현 교

CONTENTS

Eureka: Human-Level Reward Design via Coding Large Language Models

1. 인간 수준의 Dexterity 달성을 위한 LLM의 문제점

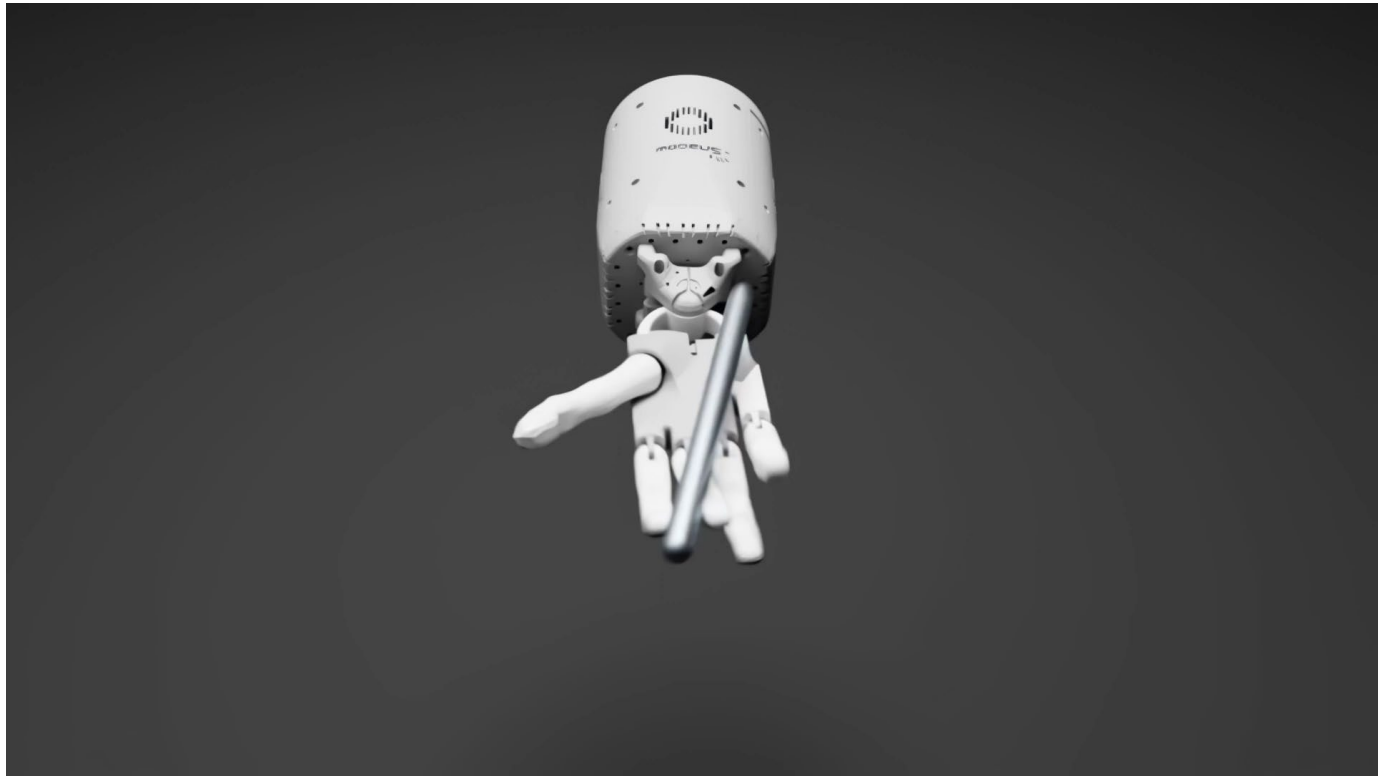
- LLM은 이미 로봇 과제에 대한 고수준(High-Level) 의미(Semantic) 계획자
- 펜 회전과 같은 **복잡한 저수준(Low-Level) 조작 작업을 학습하는 데 여전히 문제**
  → 기존의 경우, **작업 프롬프트 구성을 위해 전문적 지식 필요** 또는 **단순한 스킬만 학습**



2. 강화학습을 사용하기 어려운 이유

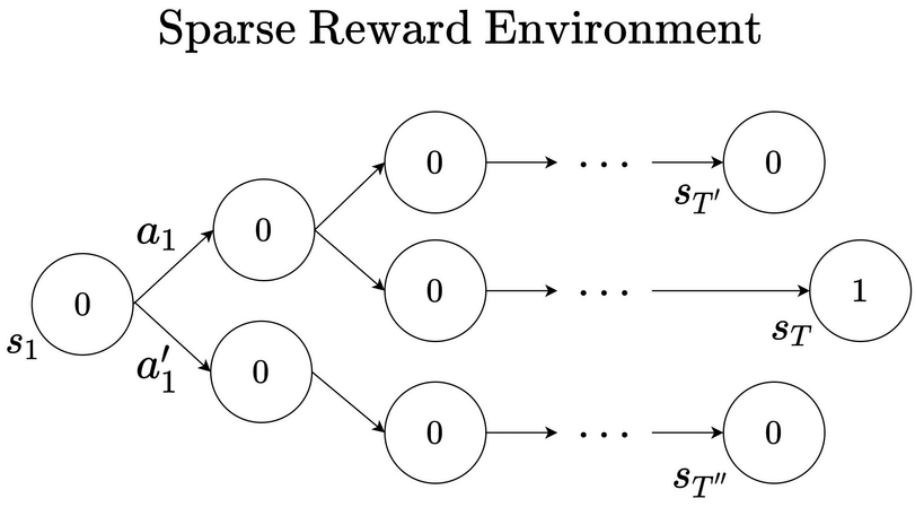
- 강화학습은 Dexterity뿐만 아니라 많은 다른 영역에서도 인상적인 결과가 존재
- 인간 디자이너가 원하는 행동에 대한 학습 신호를 정확하게 제공할 수 있도록 **주의 깊게 구성이 필요**
- 현실 세계의 강화 학습은 학습이 어려운 희소 보상(Sparse Reward)이 대부분으로, **원하는 행동에 대한 학습 신호를 제공하는 보상 성형이 필요**

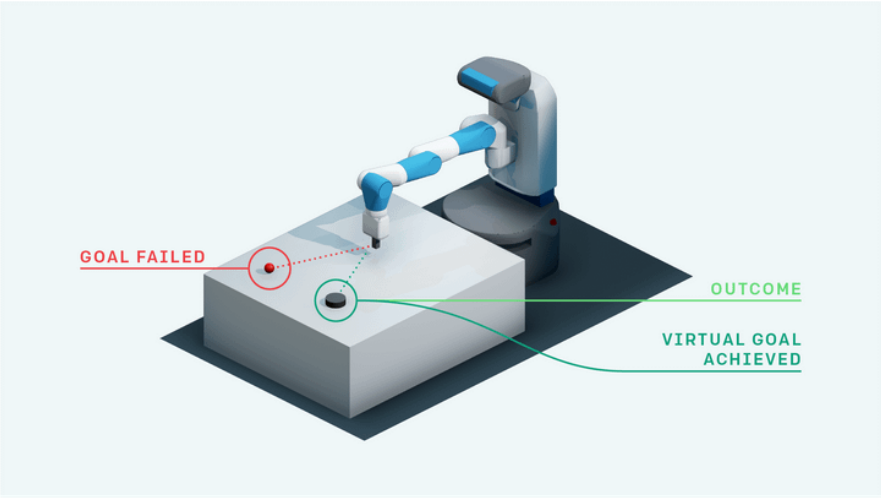
  → **보상 함수는 설계하기 어려워** 강화 학습 연구자 및 실무자의 92%가 수동 시행착오 보상 설계하고 89%가 최적이 아니라고 보고



희소 보상(Sparse Reward)

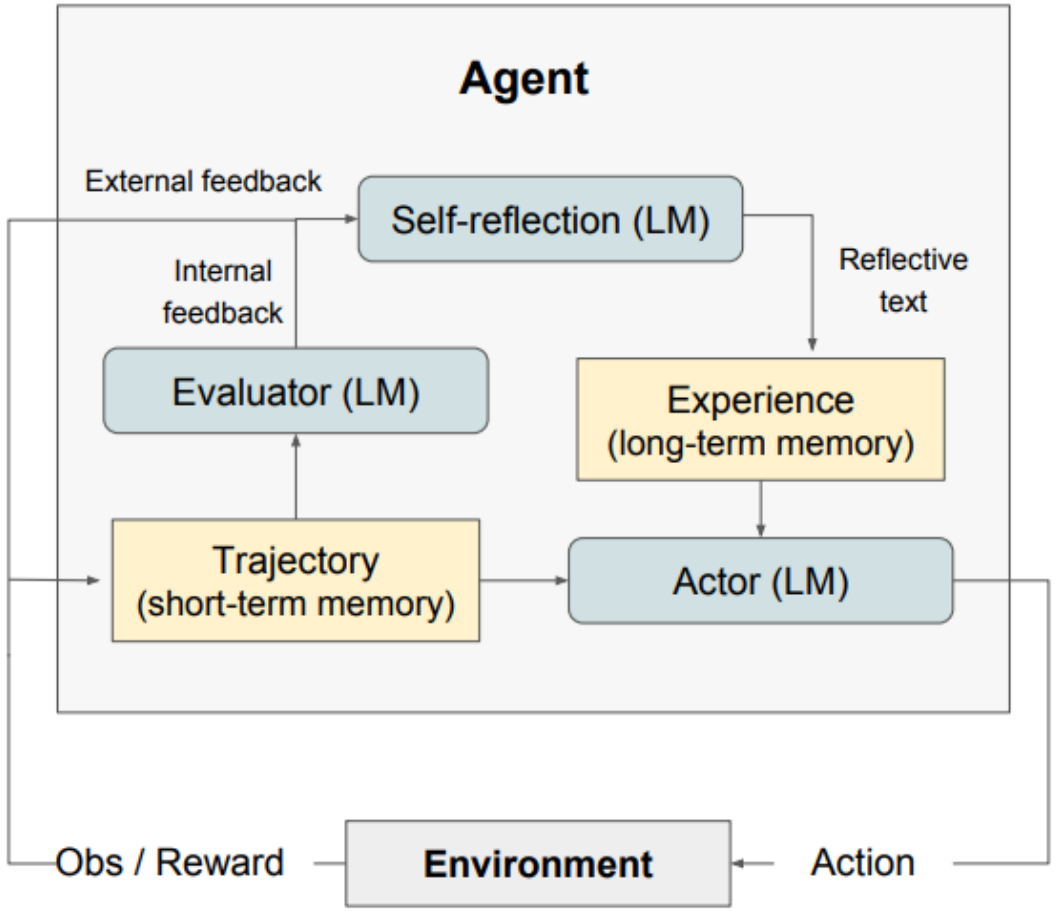
- **<u>에이전트가 목표를 달성할 때까지 보상을 거의 받지 못하는 상황을 의미</u>**
- Reward signal을 얻기 전까지 에이전트의 이용(Exploitation)은 의미 없는 행동이 되어 결국 탐험(Exploitation)만을 통해서 Reward을 얻는 것이 필요한, 탐험이 굉장히 어려운(Hard Exploration) 문제





3. LLM을 통한 보상 프로그래밍 알고리즘

- LLM은 코드 작성, Zero-shot(제로샷) 및 Context(맥락) 학습에 놀라운 능력을 보유
- 넓은 범위의 작업에 **인간 수준의 보상 생성 능력을 달성**
  → **인간 없이 번거로운 시행착오 절차를 자동화**하고 동시에 **안전 및 일치성을 보장하기 위해 인간과 호환**



EUREKA(**E**volution-driven **U**niversal **RE**ward **K**it for **A**gent)

1. **인간 수준의 성능을 달성**
   - 다양한 로봇 형태를 포함하여 과제별 프롬프팅(Prompting)이나 보상 템플릿 없이도 자동으로 보상을 생성
   - 83%에서 전문가의 보상을 능가하며, 평균 정규화 개선율이 52%를 달성

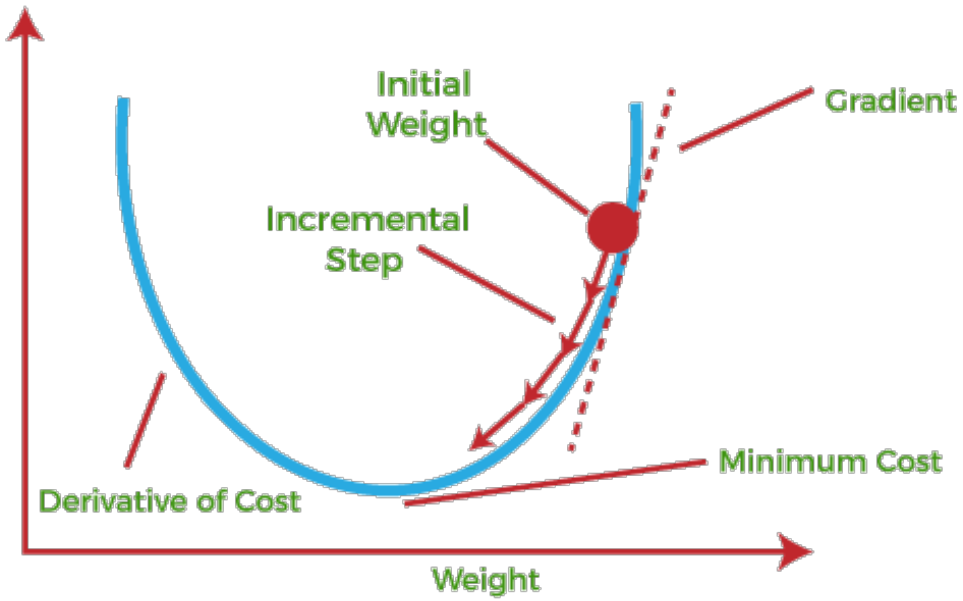
2. **수동 보상 엔지니어링으로 해결 불가능한 Dexterity 조작 과제 해결**
   - 특히 펜 스핀 과제(손이 미리 정의된 회전에 따라 펜을 최대한 많이 회전해야 하는 과제)를 고려

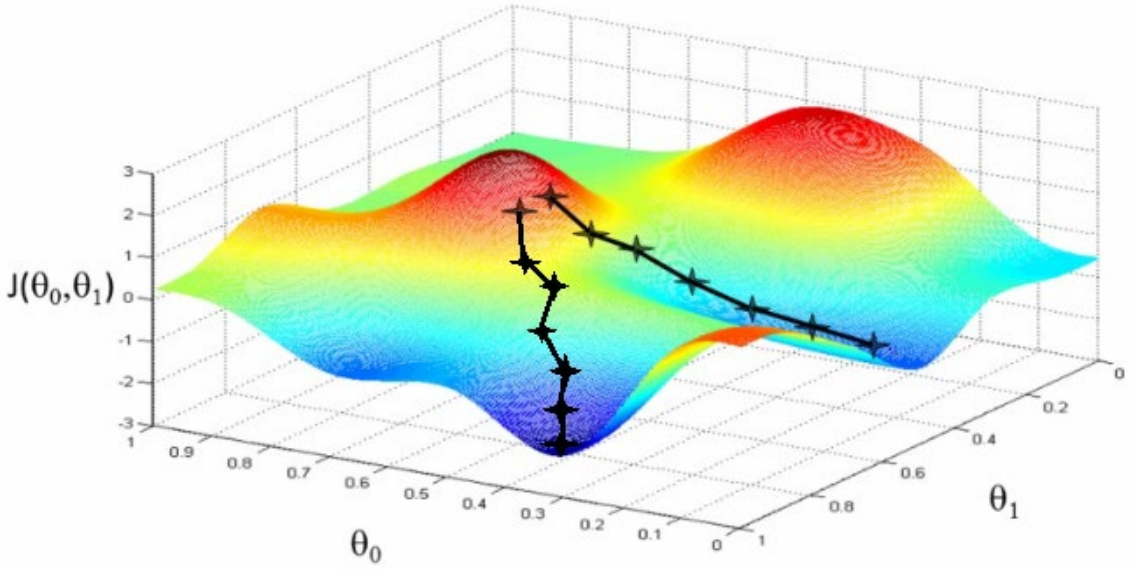
3. **인간 피드백을 기반으로 하는 새로운 Gradient-free In-Context Learning RLHF 활용**
   - 인간 피드백에서 파생된 성능이 우수하고 인간과 일치하는 보상 함수를 생성하는 새로운 Gradient-free In-Context Learning RLHF
   - EUREKA는 기존의 인간 보상 함수를 쉽게 활용하고 개선 가능
   - 인간의 피드백을 사용하여 에이전트가 미묘한 인간 선호를 포착하는 보상 함수 디자인을 협동적으로 수행하는 능력을 보유

경사 하강법(Gradient Descent)
함수의 기울기를 구하고 경사의 반대 방향으로 계속 이동시켜 극값에 이를 때까지 반복하는 최적화 알고리즘

$$\theta_{i+1} = \theta_i - \alpha \frac{\partial J}{\partial \theta}(\theta_i)$$

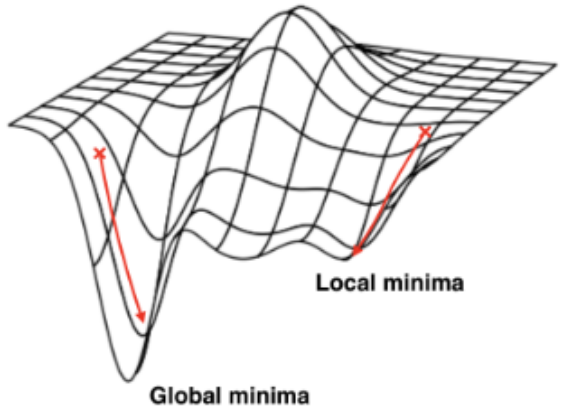
θ: Parameters (Weights)
α: Learning rate
J: Cost function

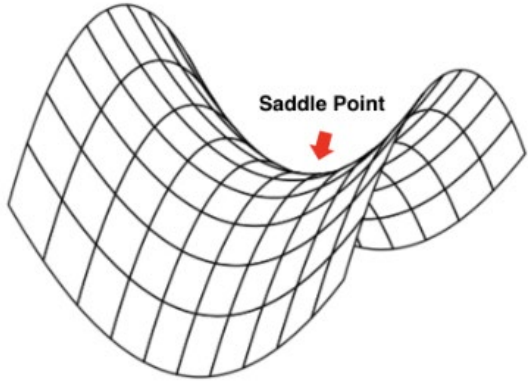


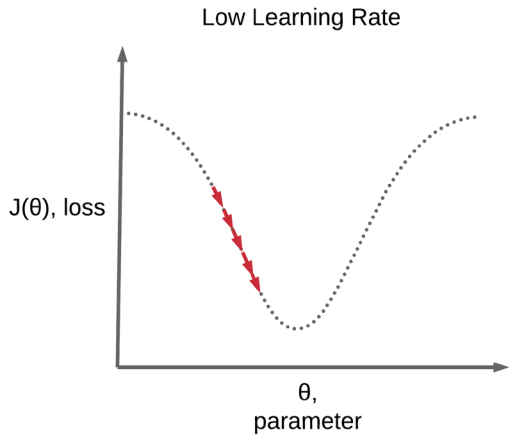


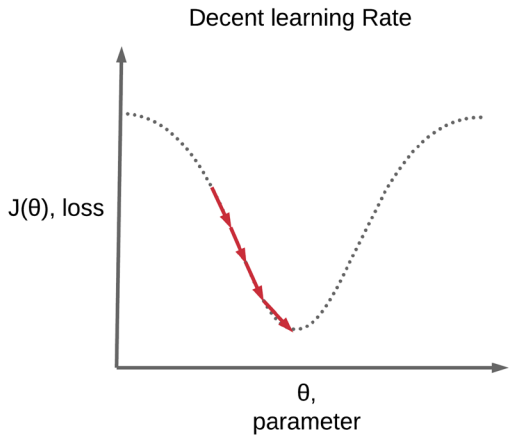
경사 하강법(Gradient Descent)의 문제점

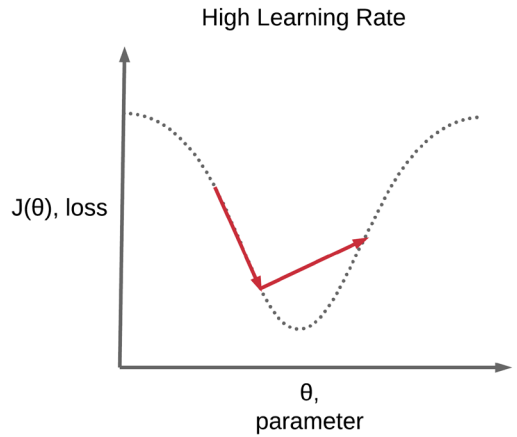
1. **Local minimum**: Global minimum 대신 Local minimum으로 수렴할 가능성 존재
2. **Saddle point**: Gradient는 0이나 최솟값 또는 최대값이 아닌 지점에 수렴할 가능성 존재
3. **Learning Rate**: Learning Rate에 민감하여 계산 시간이 증가할 가능성 존재







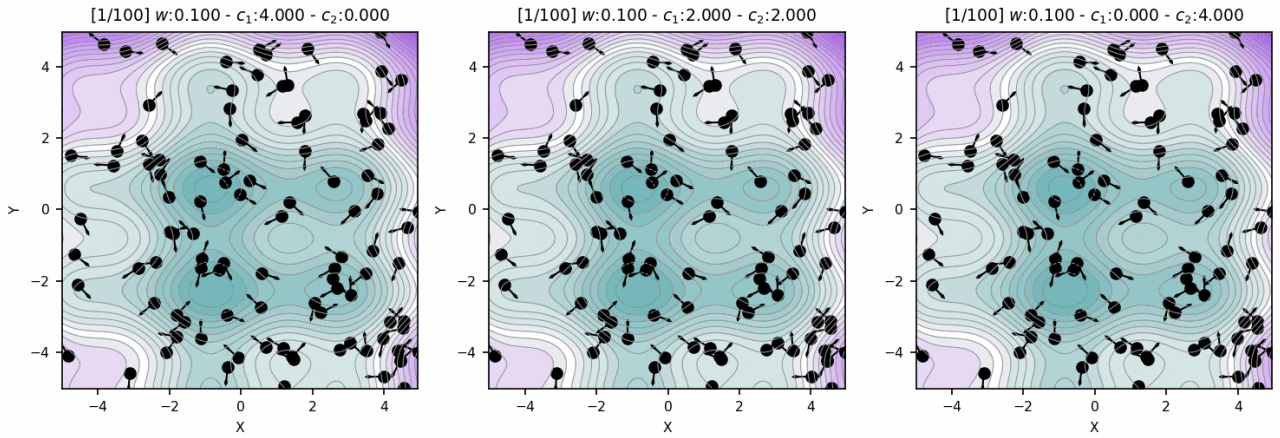


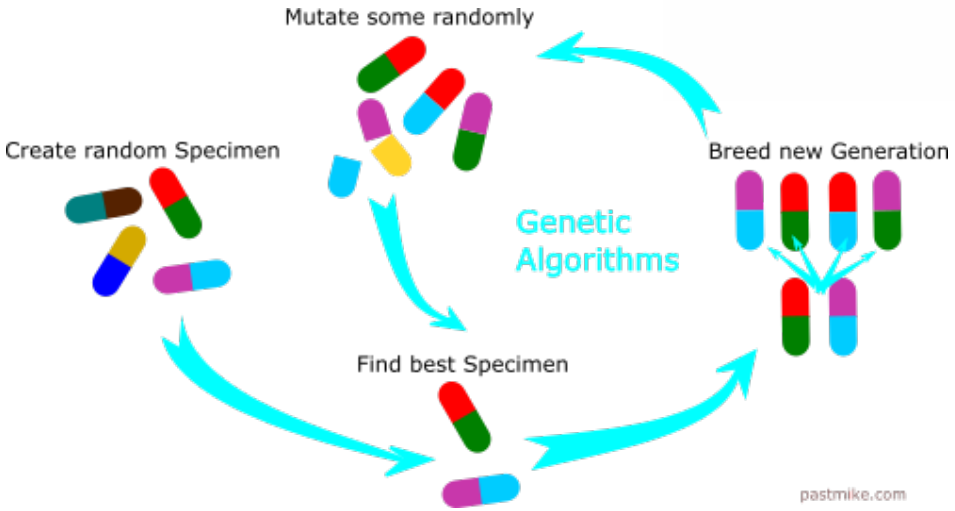


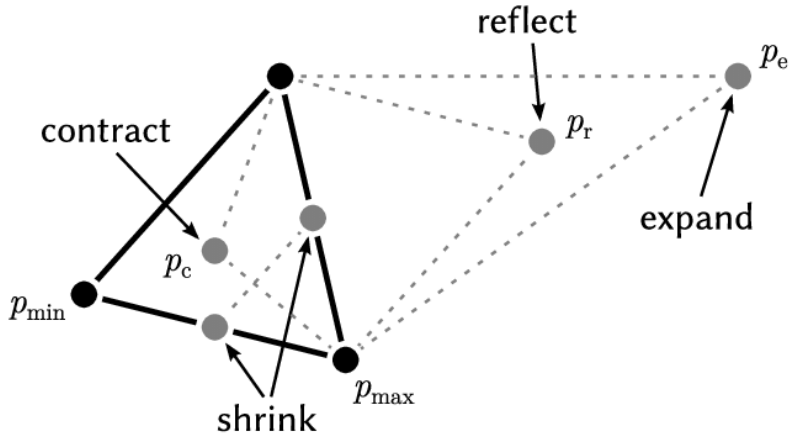
Derivative Free(Blackbox Optimization)
도함수를 사용하지 않는 최적화 알고리즘

- Particle swarm optimization
- Genetic algorithm
- Nelder–Mead method

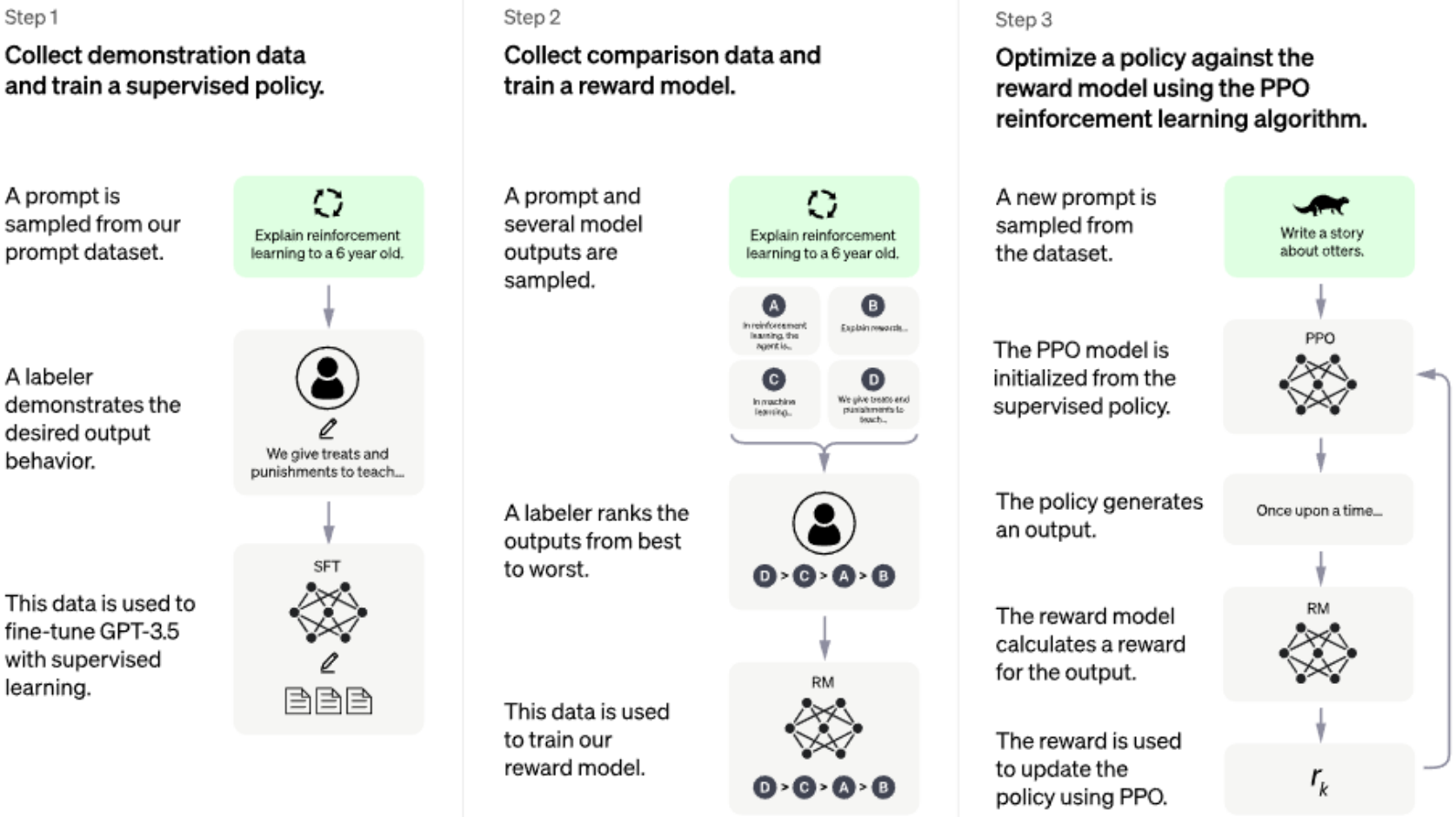
⋮





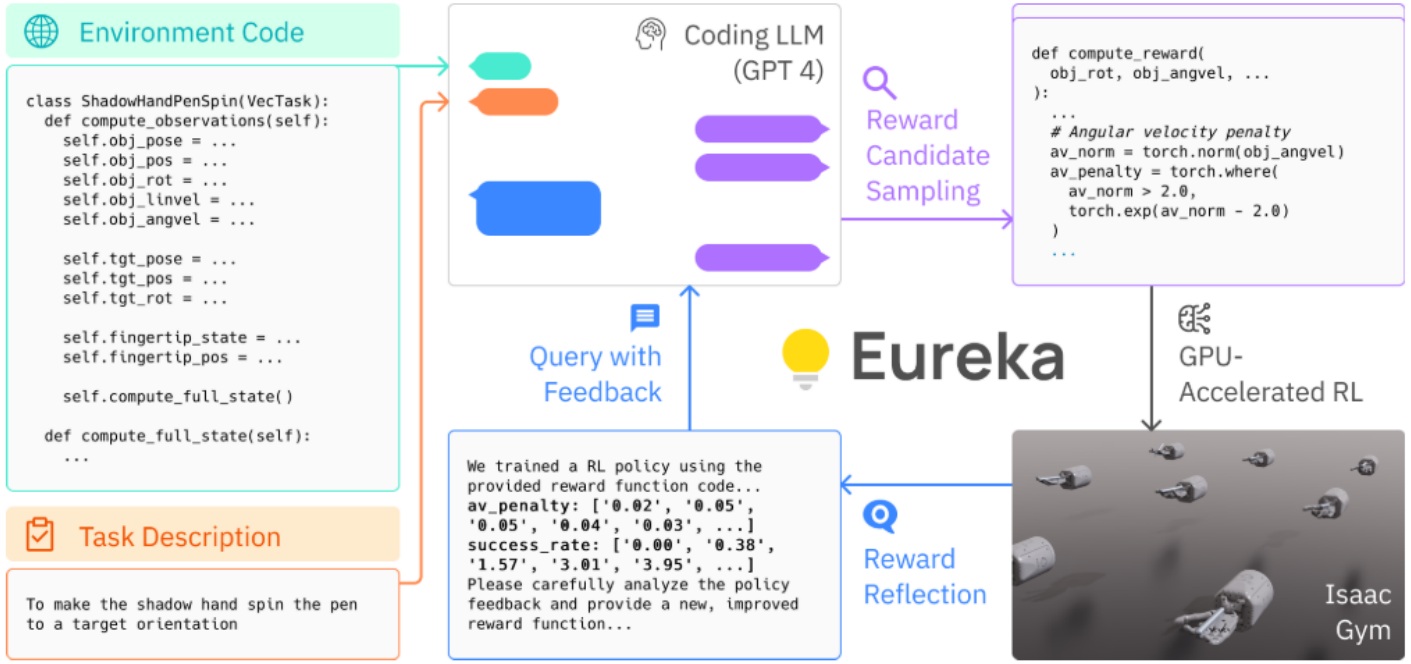


RLHF(**R**einforcement **L**earning from **H**uman **F**eedback)
기계 학습에서 인간의 피드백으로부터 직접 보상 모델을 학습시키고, 해당 모델을 보상 함수로 사용하여 최적화 알고리즘을 통해 강화 학습을 사용하여 에이전트의 정책을 최적화하는 기술



EUREKA의 핵심 알고리즘

1. **Environment as context**
   - 환경 소스 코드를 Context(맥락)으로 사용하여 EUREKA는 LLM(GPT-4)에서 **실행 가능한 보상 함수를 제로샷으로 생성 가능**
2. **Evolutionary search**
   - Evolutionary search를 통해 LLM context window에서 **보상 후보를 반복적으로 제안**
   - **가장 유망한 후보들을 개선함으로써** 보상의 품질을 크게 향상
3. **Reward reflection**
   - In-Context 개선은 Reward reflection을 통해 효과적으로 이루어지며, Policy가 학습된 통계를 기반으로 **보상에 대한 품질을 텍스트 요약으로 제공하여 자동화 및 타겟 지향적으로 보상 편집이 가능**
4. 그 외
   - Isaac Gym에서 GPU 가속 분산 강화 학습을 사용하여 평가



보상 디자인의 목표

- 직접 최적화하기 어려운 보상(예: 희소 보상)을 **Ground-truth** 형태를 갖춘 보상 함수로 반환하는 것
  - ✓ Ground-truth 보상 함수는 디자이너에 의한 쿼리를 통해서만 액세스 가능
  - ✓ Singh et al. (2010) 연구의 공식적 정의를 소개하고, 이를 Program synthesis setting에 맞게 조정하여 Reward Generation으로 명명

1. Reward Design Problem(RDP): $P=<M, R, \pi_M, F>$

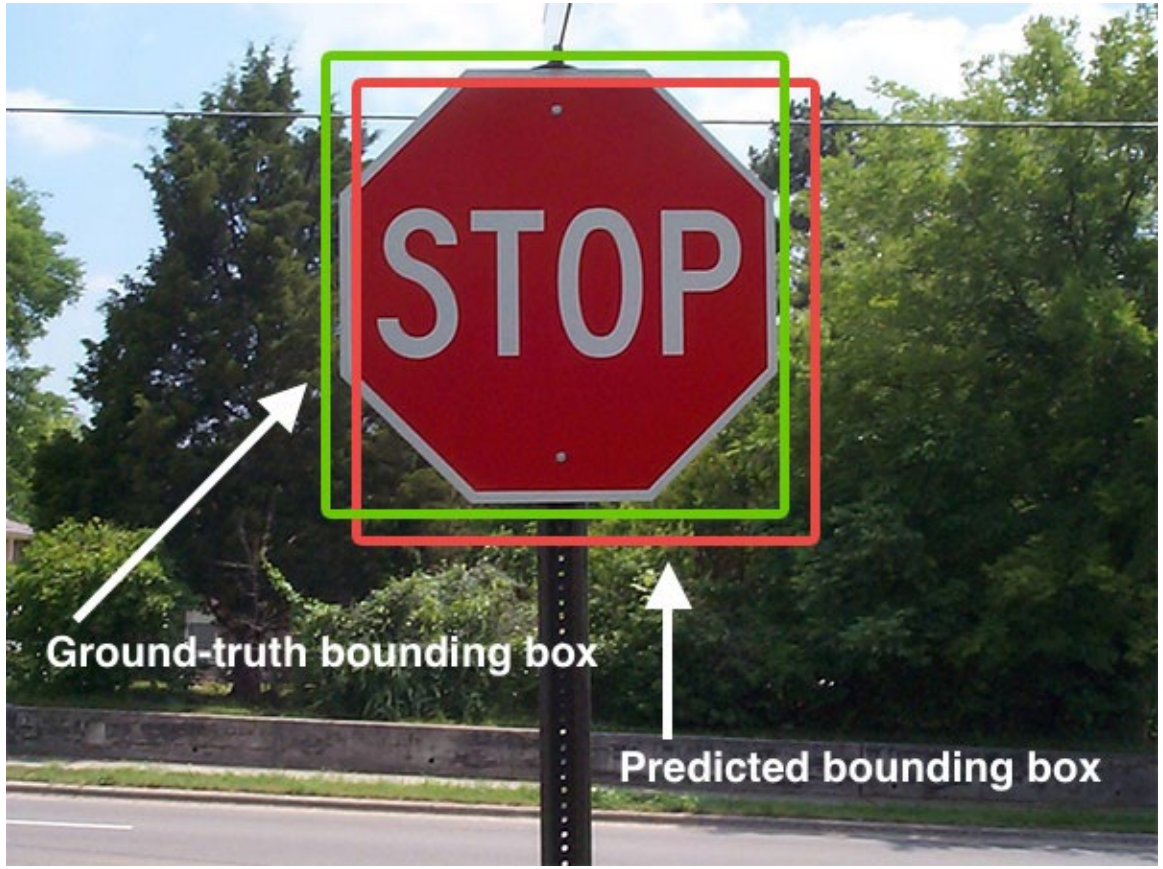
- RDP의 목표는 보상 **R**을 최적화하는 정책 $\pi := A_M(R)$가 가장 높은 적합성 점수 $F(\pi)$를 달성하도록 보상 함수를 출력하는 것
  - ✓ $M = (S, A, T)$: M은 World Model이며 3가지(상태 공간 S, 행동 공간 A, 전이 함수 T)로 구성
  - ✓ R: 보상 함수 공간
  - ✓ $A_M(\cdot) : R \rightarrow \Pi$: 정책 $\pi : S \rightarrow \Delta(A)$를 출력하는 학습 알고리즘으로 Markov Decision Process(MDP)와 $(M, \boldsymbol{R})$에서 보상 $\boldsymbol{R} \in R$을 최적화
  - ✓ $F : \Pi \rightarrow \mathbf{R}$: 정책 쿼리를 통해서만 액세스할 수 있는 모든 정책에 대하여 Scalar 평가를 생성하는 적합성 함수(즉, 적합성 함수를 사용하여 정책을 평가)

2. Reward Generation Problem

- 문제 설정에서 RDP 내의 각 구성 요소는 코드를 통해 지정
- 문자열 형태의 과제 I가 주어지면, Reward Generation Problem의 목표는 $F(A_M(\boldsymbol{R}))$을 최대화하는 보상 함수 코드 $\boldsymbol{R}$을 출력하는 것

Ground-truth

- Ground-truth는 학습하고자 하는 데이터의 원본 혹은 실제 값을 의미
- **참(True)이 아니라, 이상적이라고 여겨지는 결과**



ENVIRONMENT AS CONTEXT

1. **보상 코드를 제외하고 환경 소스 코드를 Context로 직접 제공할 필요가 존재**
   - 코드 LLM은 기존 프로그래밍 언어로 작성된 네이티브(Native) 코드에 대해 훈련되어 있으므로, **직접 훈련된 스타일과 구문으로 작성하게 하는 것이 코드 생성 능력이 강화**될 것으로 예상
   - 환경 소스 코드는 일반적으로 환경이 의미적으로 무엇을 포함하며 작업에 대한 보상 함수를 구성하는 데 **사용할 수 있는 변수가 무엇인지를 확인 가능**
2. EUREKA는 코딩 LLM에게 일반적인 보상 디자인 및 서식 지침만을 포함한 실행 가능한 Python 코드를 직접 반환하도록 지시
3. **생성된 보상이 항상 실행 가능하지 않을 수 있으며, 실행 가능하더라도 작업 적합성 F에 대해 최적화되지 않을 가능성 존재**
4. **과업별 서식 지정 및 보상 디자인 힌트로 프롬프트를 개선할 수 있지만, 새로운 과업에 확장되지 않고 전반적인 일반성을 저해**

```
def compute_reward(object_rot, goal_rot, object_angvel, object_pos, fingertip_pos):
    # Rotation reward
    rot_diff = torch.abs(torch.sum(object_rot * goal_rot, dim=1) - 1) / 2
-   rotation_reward_temp = 20.0
+   rotation_reward_temp = 30.0                                          Changing hyperparameter
    rotation_reward = torch.exp(-rotation_reward_temp * rot_diff)

    # Distance reward
+   min_distance_temp = 10.0
    min_distance = torch.min(torch.norm(fingertip_pos - object_pos[:, None], dim=2), dim=1).values
-   distance_reward = min_distance
+   uncapped_distance_reward = torch.exp(-min_distance_temp * min_distance)
+   distance_reward = torch.clamp(uncapped_distance_reward, 0.0, 1.0)    Changing functional form

-   total_reward = rotation_reward + distance_reward
+   # Angular velocity penalty                                           Adding new component
+   angvel_norm = torch.norm(object_angvel, dim=1)
+   angvel_threshold = 0.5
+   angvel_penalty_temp = 5.0
+   angular_velocity_penalty = torch.where(angvel_norm > angvel_threshold,
+       torch.exp(-angvel_penalty_temp * (angvel_norm - angvel_threshold)), torch.zeros_like(angvel_norm))
+
+   total_reward = 0.5 * rotation_reward + 0.3 * distance_reward - 0.2 * angular_velocity_penalty

    reward_components = {
        "rotation_reward": rotation_reward,
        "distance_reward": distance_reward,
+       "angular_velocity_penalty": angular_velocity_penalty,
    }

    return total_reward, reward_components
```

EVOLUTIONARY SEARCH

1. **EUREKA는 LLM에서 여러 독립적인 출력을 샘플링**
   - 반복별로 모든 보상 함수가 버그인 경우의 확률은 샘플 수가 증가함에 따라 지수적으로 감소
   - **소수의(16개) 출력만 샘플링해도** 첫 번째 반복에서 적어도 **하나의 실행 가능한 보상 코드가 포함된다는 것을 발견**
2. 이전 반복에서 실행 가능한 보상 함수를 제공하면 **EUREKA는 In-Context 보상 변이를 수행하고 텍스트 피드백을 기반으로 기존 보상 함수에서 새로운 개선된 보상 함수를 제안**
3. **변이를 통한 새로운 EUREKA의 반복은 이전 반복에서 가장 성능이 좋은 보상을 Context로 삼고 LLM에서 K개의 추가 독립 보상 출력을 생성**
4. 최적화 반복은 지정된 반복 횟수에 도달할 때까지 계속 진행
5. **더 나은 솔루션을 찾기 위해 무작위로 재시작**
   - 나쁜 초기 추정치를 극복하기 위한 글로벌 최적화의 전략
   - 모든 실험에서 EUREKA는 각 환경당 5회 독립 실행을 수행하며, 각 실행에 대해 16개의 샘플을 갖는 5개의 반복을 검색

Algorithm 1 EUREKA

Require: Task description l, environment code M, coding LLM `LLM`, fitness function F, initial prompt `prompt`
Hyperparameters: search iteration N, iteration batch size K
for N iterations **do**
  `// Sample` K `reward code from LLM`
  $\mathrm{R}_1, ..., \mathrm{R}_k \sim \mathtt{LLM}(l, M, \mathtt{prompt})$
  `// Evaluate reward candidates`
  $s_1 = F(\mathrm{R}_1), ..., s_K = F(\mathrm{R}_K)$
  `// Reward reflection`
  $\mathtt{prompt} := \mathtt{prompt} : \mathtt{Reflection}(R^n_{best}, s^n_{best})$, where $best = \arg\max_k s_1, ..., s_K$
  `// Update Eureka reward`
  $R_{\text{Eureka}}, s_{\text{Eureka}} = (R^n_{\text{best}}, s^n_{\text{best}}), \quad \text{if } s^n_{\text{best}} > s_{\text{Eureka}}$
Output: R_{Eureka}

REWARD REFLECTION

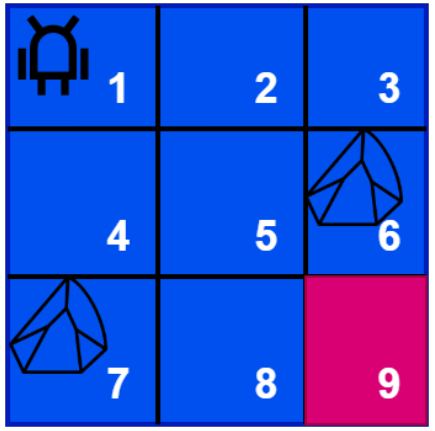
1. **컨텍스트 내 보상 돌연변이를 구체화하기 위해서는 생성된 보상의 품질을 말로 표현할 필요 존재**
   - 작업 적합성 함수는 전반적으로 Ground-truth 메트릭으로 기능하지만, **보상 함수가 작동하는 이유나 그렇지 않은 이유에 대한 유용한 정보를 제공하지 않아 Credit assignment 측면에서 부족**
2. **보상에 대한 더 정교하고 특정한 진단을 제공하기 위해, 정책 훈련을 요약하는 자동 피드백을 생성하기로 제안**
   - EUREKA 보상 함수가 개별 구성 요소를 드러내도록 요청될 경우, **교육 중간에 정책 체크포인트에서 모든 보상 구성 요소의 스칼라 값을 추적**
3. Reward Reflection 프로시저는 구성하기 간단하지만 보상 최적화의 알고리즘 종속성 때문에 중요
   - 보상 함수가 효과적인지 여부는 특정한 RL 알고리즘의 선택에 영향을 받으며, 하이퍼파라미터의 차이에 따라 동일한 최적화 도구에서도 동일한 보상이 매우 다르게 작동할 가능성 존재
4. **Reward Reflection을 통해 RL 알고리즘이 개별 보상 구성 요소를 얼마나 효과적으로 최적화하는지에 대한 자세한 내용을 제공함으로써, EUREKA는 더 정확한 보상 편집을 수행하고 고정된 RL 알고리즘과 더 잘 협력하는 보상 함수를 종합적으로 생성**

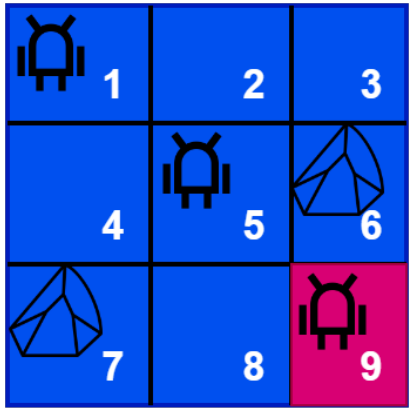
```
def compute_reward(object_rot, goal_rot, object_angvel, object_pos, fingertip_pos):
    # Rotation reward
    rot_diff = torch.abs(torch.sum(object_rot * goal_rot, dim=1) - 1) / 2
-   rotation_reward_temp = 20.0
+   rotation_reward_temp = 30.0                                   Changing hyperparameter
    rotation_reward = torch.exp(-rotation_reward_temp * rot_diff)

    # Distance reward
+   min_distance_temp = 10.0
    min_distance = torch.min(torch.norm(fingertip_pos - object_pos[:, None], dim=2), dim=1).values
-   distance_reward = min_distance
+   uncapped_distance_reward = torch.exp(-min_distance_temp * min_distance)
+   distance_reward = torch.clamp(uncapped_distance_reward, 0.0, 1.0)   Changing functional form

-   total_reward = rotation_reward + distance_reward
+   # Angular velocity penalty                                    Adding new component
+   angvel_norm = torch.norm(object_angvel, dim=1)
+   angvel_threshold = 0.5
+   angvel_penalty_temp = 5.0
+   angular_velocity_penalty = torch.where(angvel_norm > angvel_threshold,
+       torch.exp(-angvel_penalty_temp * (angvel_norm - angvel_threshold)), torch.zeros_like(angvel_norm))
+
+   total_reward = 0.5 * rotation_reward + 0.3 * distance_reward - 0.2 * angular_velocity_penalty

    reward_components = {
        "rotation_reward": rotation_reward,
        "distance_reward": distance_reward,
+       "angular_velocity_penalty": angular_velocity_penalty,
    }

    return total_reward, reward_components
```

```
reward_components = {
    "rotation_reward": rotation_reward,
    "distance_reward": distance_reward,
+   "angular_velocity_penalty": angular_velocity_penalty,
}

return total_reward, reward_components
```

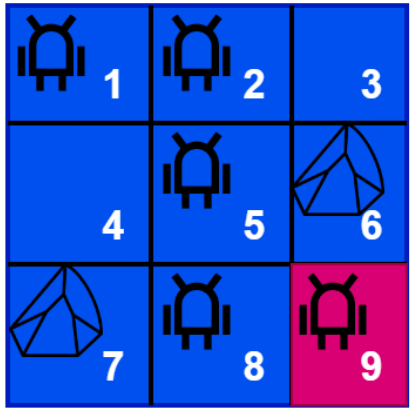
Credit Assignment Problem

- **에이전트가 받은 보상이 어떤 과거 행동(Action)들로부터 발생했는지 정확히 알아내는 것이 어려운 문제를 의미**
- 에이전트가 취한 행동이 미래 보상에 미치는 영향과 영향을 측정하는 문제

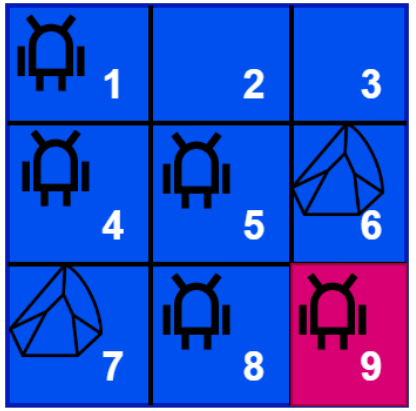




Path 1 (1-5-9)



Path 2 (1-2-5-8-9)

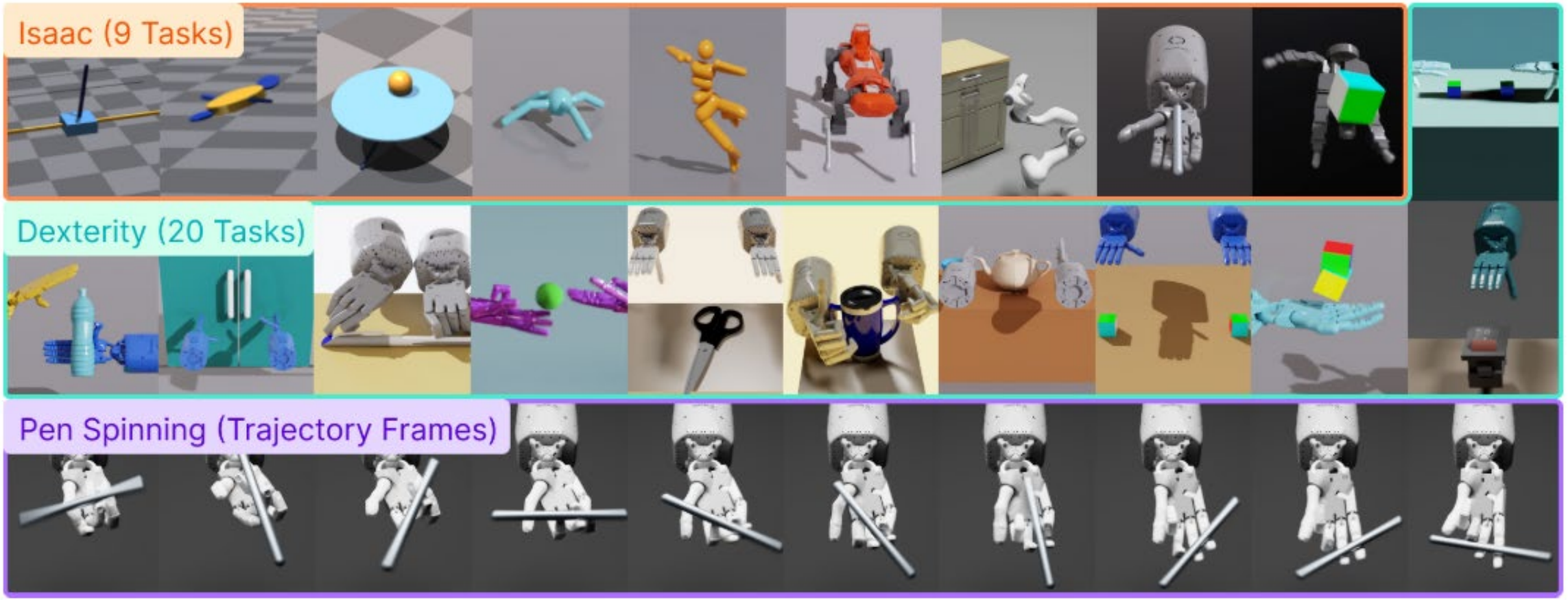


Path 3 (1-4-5-8-9)

EXPERIMENTS

Environments

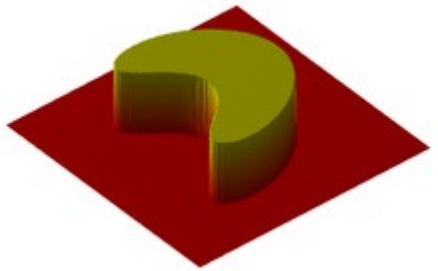
- GPT-4 사용
- Isaac Gym을 사용하여 구현된 10가지 다른 로봇과 29가지 작업으로 구성



EXPERIMENTS

BASELINES

1. L2R
   - **<u>템플릿화된 보상을 생성하기 위한 두 단계의 LLM-Prompting 솔루션을 제안</u>**
     - ✓ 자연어로 지정된 환경과 작업에 대해, 첫 번째 LLM은 에이전트의 동작을 설명하는 자연어 템플릿을 작성하도록 요청
     - ✓ 두 번째 LLM에게 "동작 설명"을 코드로 변환하도록 요청하여 수동으로 정의된 일련의 보상 API 기본 명령을 호출하는 코드로 변환
   - **<u>EUREKA와 일관성 있게 각 환경에 대해 5회 독립적인 L2R 실행을 수행하며 각 실행에 대해 16개의 보상 샘플을 생성</u>**
2. Human
   - 벤치마크 작업에서 제공된 원본 형성된 보상 함수
     - ✓ 보상 함수들은 작업을 설계한 강화 학습 연구자들에 의해 작성되었으며 **<u>전문가 수준의 인간이 작성한 보상 엔지니어링의 결과</u>**
3. Sparse
   - **<u>생성된 보상의 품질을 평가하는 데 사용되는 적합성 함수 F와 동일</u>**
   - Human과 마찬가지로 이들은 벤치마크에서 제공
   - **Dexterity 작업에서는** 작업 성공을 측정하는 균일한 **이진 지시 함수**(Indicator function)이며, **Isaac 작업에서는** 작업의 성격에 따라 **함수 형태가 다양**



TRAINING DETAILS

1. Policy Learning
   - **각 작업에 대해 모든 최종 보상 함수는 동일한 RL 알고리즘과 동일한 하이퍼파라미터 세트를 사용하여 최적화**
   - Isaac과 Dexterity는 튜닝된 PPO Implementation을 공유하며, 이 **Implementation과 작업별 PPO 하이퍼파라미터를 수정하지 않고 사용**
   - 작업 하이퍼파라미터는 공식적인 Human-Engineered 보상이 잘 작동하도록 조정
   - 각 보상에 대해 5개의 독립적인 PPO 훈련 실행을 수행하고, 정책 체크포인트에서 달성한 최대 작업 메트릭 값의 평균을 해당 보상의 성능으로 보고
2. Reward Evaluation Metrics
   - Isaac 작업의 경우, 각 작업의 작업 메트릭 F가 Semantic 의미와 척도에서 다양하므로 **EUREKA 및 L2R에 대한 Human normalized score를 보고**
   - Method−Sparse / |Human−Sparse|
     - ✓ EUREKA 보상이 인간 전문가 보상에 대한 Ground-truth 작업 메트릭과 어떻게 비교되는지에 대한 종합적인 측정을 제공
   - Dexterity의 경우 모든 작업이 **이진 성공 함수를 사용하여 평가되기 때문에 직접 성공률을 보고**

$$\frac{\text{Method}-\text{Sparse}}{|\text{Human}-\text{Sparse}|}$$

TRPO(**T**rust **R**egion **P**olicy **O**ptimization)
- Trust Region을 도입하여 정책을 안정적으로 업데이트
- 다른 강화학습 알고리즘에 비해 구현 및 이해가 다소 복잡

$$\rho_\pi(s) = P(s_0 = s) + \gamma P(s_1 = s) + \gamma^2 P(s_2 = s) + \dots,$$

Discounted visitation frequencies
- State가 s를 방문할 확률

$$\eta(\pi) = \mathbb{E}_{s_0, a_0, \dots}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)\right], \text{ where}$$

$$s_0 \sim \rho_0(s_0),\ a_t \sim \pi(a_t|s_t),\ s_{t+1} \sim P(s_{t+1}|s_t, a_t).$$

Expected cumulative reward
- 특정 정책을 따를 때, 기대할 수 있는 총 보상의 합= 특정 정책의 성능

$$Q_\pi(s_t, a_t) = \mathbb{E}_{s_{t+1}, a_{t+1}, \dots}\left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})\right],$$

$$V_\pi(s_t) = \mathbb{E}_{a_t, s_{t+1}, \dots}\left[\sum_{l=0}^{\infty} \gamma^l r(s_{t+l})\right],$$

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s), \text{ where}$$

$$a_t \sim \pi(a_t|s_t), s_{t+1} \sim P(s_{t+1}|s_t, a_t) \text{ for } t \geq 0.$$

Kakade & Langford (2002)

$$\eta(\tilde{\pi}) = \eta(\pi) + \mathbb{E}_{s_0, a_0, \cdots \sim \tilde{\pi}} \left[\sum_{t=0}^{\infty} \gamma^t A_\pi(s_t, a_t) \right]$$

Timestep의 관점 → States의 관점으로

$$\begin{aligned}
\eta(\tilde{\pi}) &= \eta(\pi) + \sum_{t=0}^{\infty} \sum_{s} P(s_t = s|\tilde{\pi}) \sum_{a} \tilde{\pi}(a|s) \gamma^t A_\pi(s, a) \\
&= \eta(\pi) + \sum_{s} \sum_{t=0}^{\infty} \gamma^t P(s_t = s|\tilde{\pi}) \sum_{a} \tilde{\pi}(a|s) A_\pi(s, a) \\
&= \eta(\pi) + \sum_{s} \rho_{\tilde{\pi}}(s) \sum_{a} \tilde{\pi}(a|s) A_\pi(s, a). \qquad (2)
\end{aligned}$$

π~ → π

$$L_\pi(\tilde{\pi}) = \eta(\pi) + \sum_{s} \rho_\pi(s) \sum_{a} \tilde{\pi}(a|s) A_\pi(s, a).$$

$$L_{\pi_{\theta_0}}(\pi_{\theta_0}) = \eta(\pi_{\theta_0}),$$
$$\nabla_\theta L_{\pi_{\theta_0}}(\pi_\theta)\big|_{\theta=\theta_0} = \nabla_\theta \eta(\pi_\theta)\big|_{\theta=\theta_0}.$$

충분히 작은 Step으로 정책을 업데이트
→ Lπ와 η는 first-order로 근사하여 동일
→ π_θ0 = π~
→ L_πθ0의 증가는 η의 증가와 동일
Step이 얼마나 작아야 같은지 확인 불가

Conservative policy iteration

- **Step을 설정을 위해 탄생한 방법**
- 이전의 정책과 새로운 정책을 일정 비율 혼합하여 사용
- 다만, 실질적인 사용은 무리(Mixture policy 한정으로 사용)이기에 General한 방법론 필요

$$\pi_{\text{new}}(a|s) = (1-\alpha)\pi_{\text{old}}(a|s) + \alpha\pi'(a|s).$$
$$\pi' = \arg\max_{\pi'} L_{\pi_{\text{old}}}(\pi')$$

Total variation divergence

$$\eta(\pi_{\text{new}}) \geq L_{\pi_{\text{old}}}(\pi_{\text{new}}) - \frac{4\epsilon\gamma}{(1-\gamma)^2}\alpha^2$$
$$where\ \epsilon = \max_{s,a} |A_\pi(s,a)|$$

KL divergence

- 쿨백-라이블러 발산은 두 확률분포의 차이를 계산하는 데에 사용하는 함수

$$\eta(\tilde{\pi}) \geq L_\pi(\tilde{\pi}) - CD_{\text{KL}}^{\max}(\pi, \tilde{\pi}),$$
$$\text{where } C = \frac{4\epsilon\gamma}{(1-\gamma)^2}.$$

Algorithm 1 Policy iteration algorithm guaranteeing non-decreasing expected return η

Initialize π_0.

for $i = 0, 1, 2, \ldots$ until convergence **do**

Compute all advantage values $A_{\pi_i}(s, a)$.

Solve the constrained optimization problem

$$\pi_{i+1} = \arg\max_{\pi} \left[L_{\pi_i}(\pi) - C D_{\mathrm{KL}}^{\max}(\pi_i, \pi) \right]$$

where $C = 4\epsilon\gamma/(1-\gamma)^2$

and $L_{\pi_i}(\pi) = \eta(\pi_i) + \sum_s \rho_{\pi_i}(s) \sum_a \pi(a|s) A_{\pi_i}(s, a)$

end for

Optimization of Parameterized Policies

- C가 클 경우, Penalty가 증가하여 Step size가 감소
- 이에, Constraint 형태로 변경
- 현실적으로 모든 State를 평가할 수 없기에 이를 Heuristics approximation으로 변경

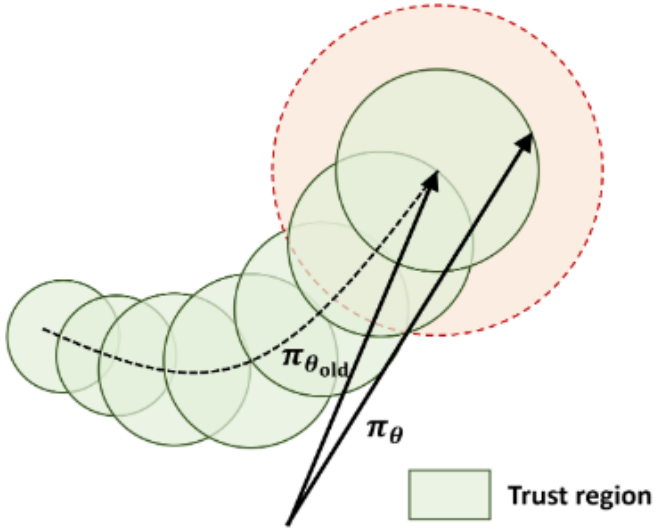
$$\underset{\theta}{\text{maximize}}\, L_{\theta_{\text{old}}}(\theta)$$
$$\text{subject to } D_{\text{KL}}^{\max}(\theta_{\text{old}}, \theta) \leq \delta.$$

$$\Rightarrow \quad \underset{\theta}{\text{maximize}} \sum_s \rho_{\theta_{\text{old}}}(s) \sum_a \pi_\theta(a|s) A_{\theta_{\text{old}}}(s,a)$$
$$\text{subject to } \overline{D}_{\text{KL}}^{\rho_{\theta_{\text{old}}}}(\theta_{\text{old}}, \theta) \leq \delta.$$

Importance sampling

- Sample 기반의 Monte-Carlo estimate로 대체

$$\underset{\theta}{\text{maximize}}\, \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}, a\sim q}\left[\frac{\pi_\theta(a|s)}{q(a|s)} Q_{\theta_{\text{old}}}(s,a)\right] \qquad (14)$$
$$\text{subject to } \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}\left[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(\cdot|s) \,\|\, \pi_\theta(\cdot|s))\right] \leq \delta.$$



PPO(**P**roximal **P**olicy **O**ptimization)

- TRPO의 경우, 복잡한 Second-order 방법으로 풀어야 하기에 매우 복잡
- **PPO는 First-order 방법으로 간단히 구현되며 좋은 성능 보유**

1. Clipping

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}, \text{ so } r(\theta_{\text{old}}) = 1.$$

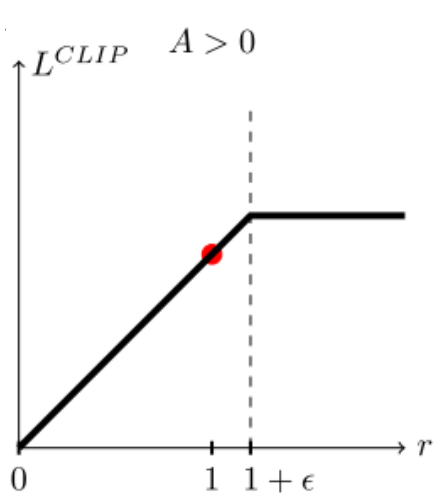
R은 Old 정책과 New 정책의 확률의 비율을 의미하며 정책이 업데이트

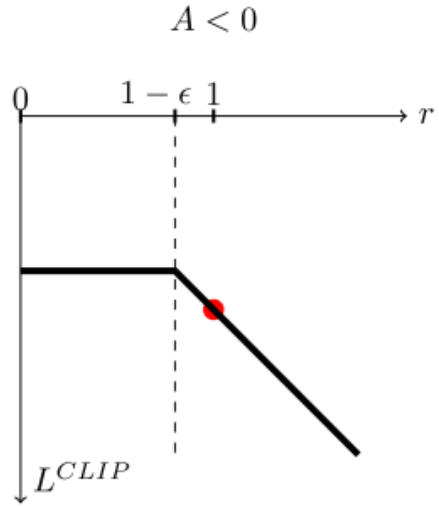
$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t\right] = \hat{\mathbb{E}}_t\left[r_t(\theta)\hat{A}_t\right].$$

Clipping은 r에 범위를 지정하여 급격한 증가나 감소를 억제

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

Clipped Loss





1) A > 0: 좋은 Action을 실행할 확률이 높기에 ε을 증가
2) A < 0: 나쁜 Action을 실행할 확률이 높기에 ε을 감소

Adaptive KL Penalty

- KL divergence에 범위를 지정하여 Step을 조정

$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t - \beta\,\mathrm{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]\right]$$

- If $d < d_{\text{targ}}/1.5$, $\beta \leftarrow \beta/2$
- If $d > d_{\text{targ}} \times 1.5$, $\beta \leftarrow \beta \times 2$

1) d < d_targ: Step이 적게 진행되어 β를 더 작게 주어 패널티를 감소해 지속적으로 업데이트를 증가
2) d > d_targ: Step이 많이 진행되어 β를 더 크게 주어 패널티를 증가해 지속적으로 업데이트를 감소

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

RESULTS

1. EUREKA outperforms human rewards
   - Dexterity
     - ✓ Dexterity의 20개 작업 중 15개에서 인간 수준을 뛰어넘거나 비슷한 성능
   - Isaac Sim
     - ✓ L2R은 저차원 작업 (예: CartPole, BallBalance)에서는 유사하나 고차원 작업은 성능이 저하
   - L2R은 일부 인간 보상 구성 요소에 액세스할 수 있지만 초기 반복 이후에도 여전히 EUREKA를 능가 불가
     - → **<u>L2R의 표현의 불충분성은 성능을 심각하게 제한</u>**
   - **EUREKA는 도메인 특정 지식 없이 처음부터 자유로운 형식의 보상을 생성하며 상당히 더 나은 성능을 발휘**

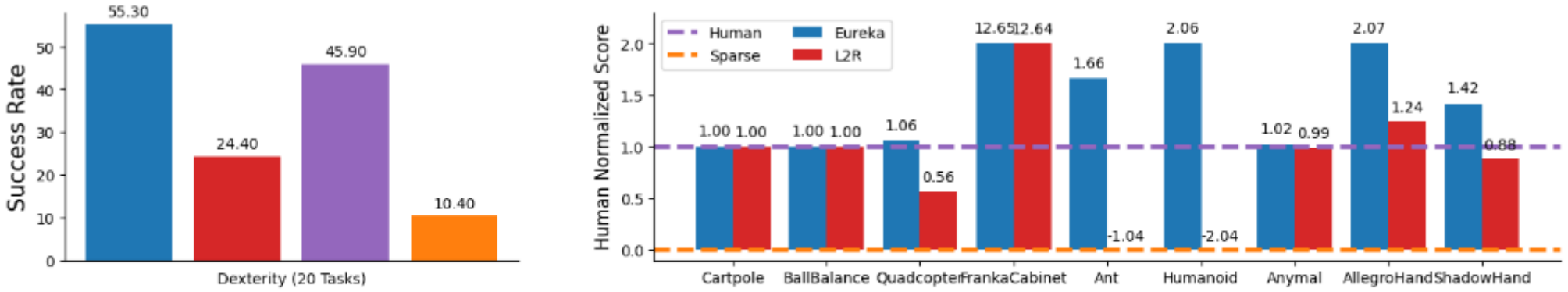


Figure 4: EUREKA outperforms Human and L2R across all tasks. In particular, EUREKA realizes much greater gains on high-dimensional dexterity environments.

RESULTS

2. EUREKA consistently improves over time
   - EUREKA without Evolution (32 Samples)
     - ✓ 초기 보상 생성 단계만 수행하며 기존의 EUREKA에서 두 번의 반복 동안에 동일한 수의 보상 함수를 샘플링
     - ✓ 이 실험은 고정된 보상 함수가 주어졌을 때 **EUREKA 진화를 수행하는 것과 단순히 반복적인 개선 없이 처음 시도하는 보상을 더 많이 샘플링하는 것 중 어떤 것이 유리한지를 연구하는 데 도움**
   - 두 벤치마크 모두에서 EUREKA 보상은 꾸준히 개선되어 최종적으로 초기 성능이 부족한 경우에도 인간 보상보다 성능이 우수
   - 일관적인 개선은 단순히 처음 반복에서 더 많이 샘플링하는 것으로 대체될 수 없으며 **Evolution 없이 수행한 실험의 성능이 양 벤치마크 모두에서 2번의 반복 후에도 EUREKA보다 낮음을 확인**
     - → **이러한 결과는 EUREKA의 혁신적인 Evolutionary optimization가 최종 성능에 불가결한 요소**

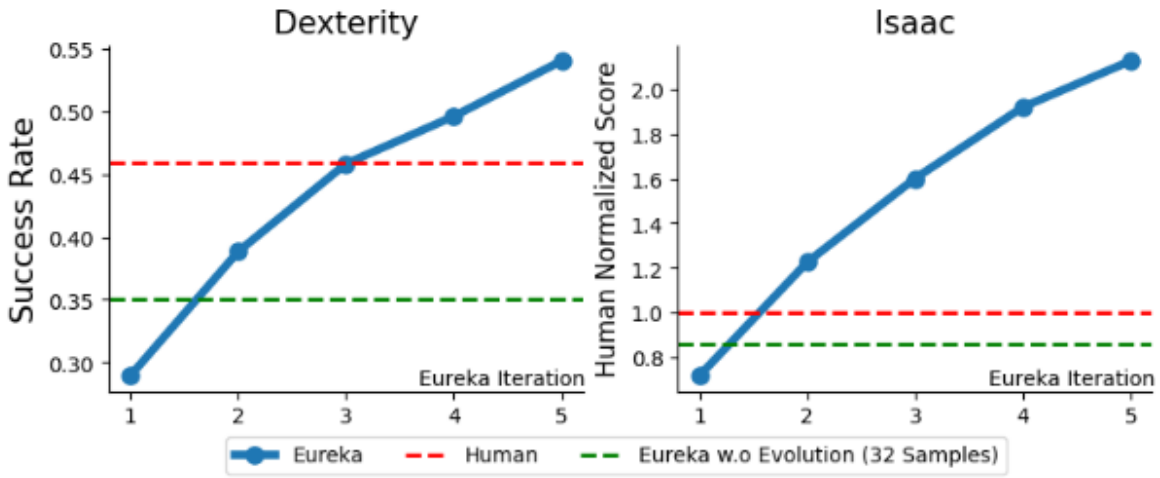


Figure 5: EUREKA progressively produces better rewards via in-context evolutionary reward search.

Algorithm 1 EUREKA

Require: Task description l, environment code M, coding LLM `LLM`, fitness function F, initial prompt `prompt`
Hyperparameters: search iteration N, iteration batch size K
for N iterations **do**
  `// Sample` K `reward code from LLM`
  $\mathrm{R}_1, \ldots, \mathrm{R}_k \sim \mathrm{LLM}(l, M, \mathrm{prompt})$
  `// Evaluate reward candidates`
  $s_1 = F(\mathrm{R}_1), \ldots, s_K = F(\mathrm{R}_K)$
  `// Reward reflection`
  $\mathrm{prompt} := \mathrm{prompt} : \mathrm{Reflection}(R^n_{best}, s^n_{best})$, where $best = \arg\max_k s_1, \ldots, s_K$
  `// Update Eureka reward`
  $R_{\text{Eureka}}, s_{\text{Eureka}} = (R^n_{\text{best}}, s^n_{\text{best}}), \quad \text{if } s^n_{\text{best}} > s_{\text{Eureka}}$
Output: R_{Eureka}

RESULTS

3. EUREKA generates novel rewards
   - EUREKA 보상의 Novelty을 평가하기 위해 **모든 Isaac 작업에서 EUREKA와 인간 보상 간의 Correlation 관계를 계산**
   - 각각의 점은 특정 하나의 작업에 대한 EUREKA 보상으로 상관 관계를 산점도로 Human Normalized Score에 대해 표현
   - **가설: 어려운 작업에 대해 인간 보상이 최적에 가까울 가능성이 낮아져 EUREKA 보상이 더 나을 여지가 증가**
   - 과제별 평균 Correlation 관계를 검토하여 어려운 작업일수록 EUREKA 보상이 Correlation이 저하
   - 몇 가지 경우에는 EUREKA 보상이 인간 보상과 심지어 음의 상관 관계를 가지지만 더 나은 성능을 발휘

     → **EUREKA가 인간 직관에 반하는 새로운 보상 설계 원칙을 발견할 수 있다는 것을 확인**

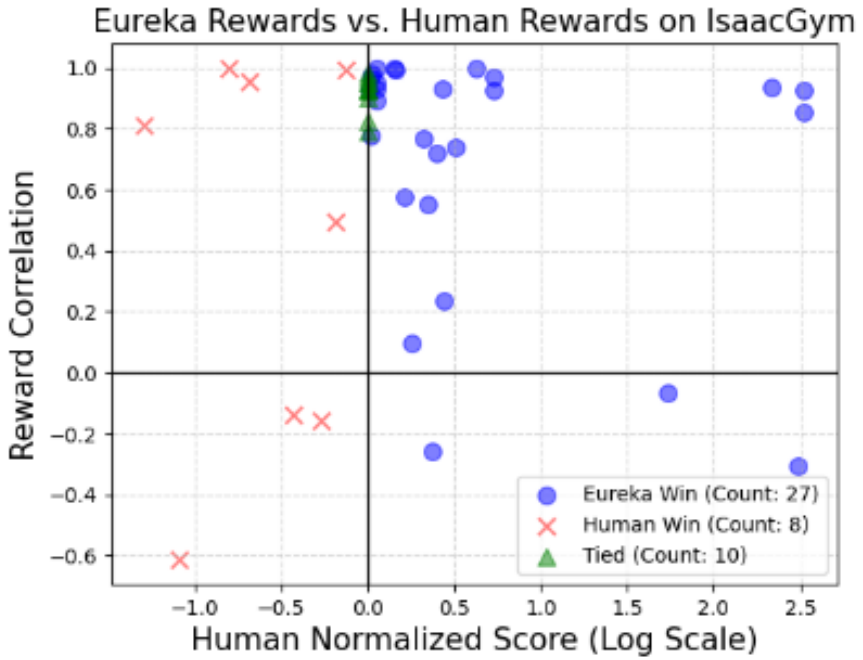


Figure 6: Eureka generates novel rewards.

RESULTS

4. Reward reflection enables targeted improvement

- **EUREKA (No Reward Reflection)**: 보상 피드백에 Reward reflection의 중요성을 평가하기 위해 보상 피드백 프롬프트를 작업 메트릭 F 만 포함하도록 줄인 실험
  - ✓ 모든 Isaac 작업을 기준으로 평균화하면 **Reward reflection이 없는 EUREKA는 평균 정규화 점수를 28.6% 감소**
  - ✓ **고차원 작업에서 성능이 훨씬 더 나빠지는 것을 관찰**

5. EUREKA with curriculum learning enables dexterous pen spinning

- **작업을 EUREKA에 의해 독립적으로 해결할 수 있는 관리 가능한 구성 요소로 분해할 수 있는지 조사**
  - ✓ 구체적으로 먼저 EUREKA에게 펜을 무작위 목표 구성으로 재조정하는 데 사용할 보상을 생성하도록 지시
  - ✓ 사전 훈련(Pre-Trained)된 정책으로 EUREKA 보상을 사용하여 목표에 도달하도록 세밀하게 조정(Fine-Tuned)
- **Fine-Tuning한 EUREKA는 성공적으로 회전하나 사전 훈련된 정책이나 처음부터 학습하는 정책은 실패**

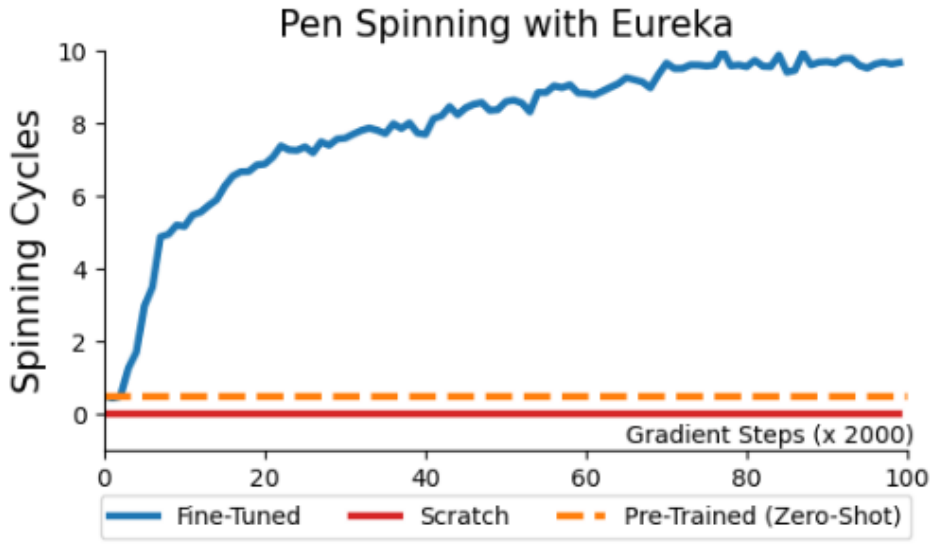


Figure 7: EUREKA can be flexibly combined with curriculum learning to acquire complex dexterous skills.

EUREKA FROM HUMAN FEEDBACK

1. EUREKA can improve and benefit from human reward functions
   - **Human reward function initialization으로 EUREKA 성능이 증가하며 EUREKA를 수정이 불필요**
     → **단순히 인간의 보상 함수를 첫 번째 EUREKA 반복의 출력으로 대체 가능**
     ✓ 모든 작업에서 EUREKA는 인간보다 일관되게 우수
       → **이는 EUREKA의 문맥에서의 보상 개선 능력이 기본 보상의 품질과 크게 독립적임을 시사**
     ✓ 보상 설계는 강화 학습에 대한 특수 지식과 경험이 필요하기 때문에 인간 디자이너도 보상을 설계하는 데는 능숙하지 않을 수 있음을 시사
       → **EUREKA의 보상 어시스턴트 기능을 보여주며, 유용한 상태 변수에 대한 인간 디자이너들의 지식을 완벽하게 보완하고 그들이 그것을 사용하여 보상을 설계하는 방법에 대해 능숙함을 보완**

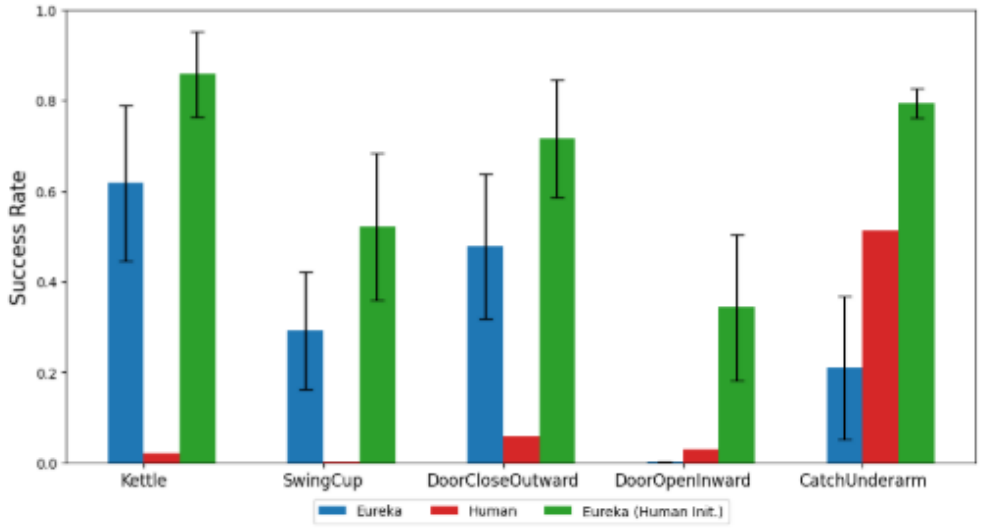


Figure 8: EUREKA effectively improves and benefits from human reward initialization.

EUREKA FROM HUMAN FEEDBACK

2. Reward reflection via human feedback induces aligned behavior

- **어려운 시나리오에서 인간이 나와서 원하는 행동과 수정에 대한 보상 반영을 단어로 표현하도록 EUREKA를 보강하기를 제안**
- 다음 20명의 익숙하지 않은 사용자에게 두 가지 정책 롤아웃 비디오 간의 기호를 나타내도록 하는 사용자 연구를 실시
  - ✓ 하나는 인간 보상 반영(EUREKA-HF)으로 훈련되었고, 다른 하나는 최상의 EUREKA 보상으로 훈련
  - ✓ **EUREKA-HF 에이전트는 사용자 대다수에게 선호되며 안정성을 위해 속도를 포기하도록 성공적으로 교환**
    - → **인간의 지시대로 EUREKA-HF 에이전트가 점차적으로 더 안전하고 안정된 보행을 습득**

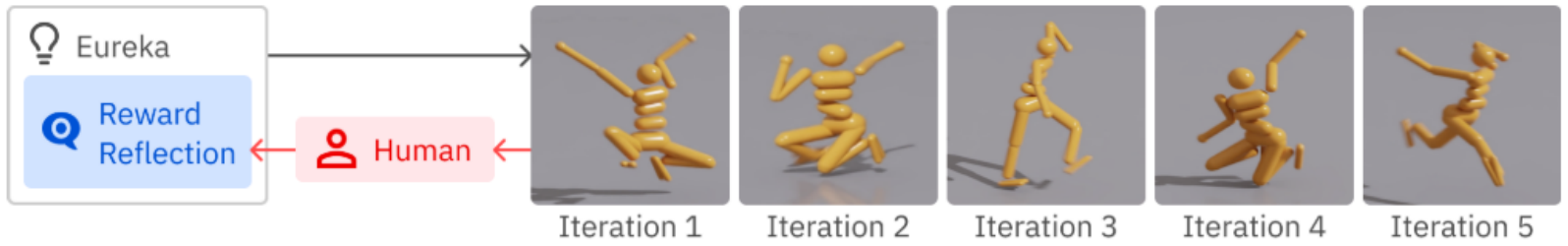


Figure 9: EUREKA can incorporate human feedback to modify rewards and induce more human-aligned policys.

Method	Forward Velocity	Human Preference
EUREKA	**7.53**	5/20
EUREKA-HF	5.58	**15/20**

Table 1: Human users prefer the Humanoid behavior learned via EUREKA rewards generated using human reward reflection.

CONCLUSION

1. **EUREKA는 코드 LLM과 In-Context Evolutionary Search을 기반으로 하는 범용적인 보상 디자인 알고리즘을 제시**
2. **특정 작업에 대한 프롬프트 엔지니어링이나 인간 개입 없이 EUREKA는 다양한 로봇과 작업에 대해 인간 수준의 보상 생성을 달성**
3. EUREKA는 특정 강점을 활용하여 Dexterous pen spinning 문제를 해결
4. **EUREKA는 인간 피드백으로부터 강화학습에 Gradient-free 접근을 가능하게 하며, Human reward initialization와 Text 피드백을 손쉽게 통합하여 더 나은 보상 생성을 유도**
5. EUREKA의 다재다능성과 상당한 성능 향상은 LLM과 Evolutionary 알고리즘을 결합하는 간단한 원칙이 보상 디자인에 대한 일반적이고 확장 가능한 접근 방식임을 시사하며, 이는 어려운 Open-ended 검색 문제에 일반적으로 적용 가능한 통찰력을 지님

Reference

1. https://ropiens.tistory.com/227
2. https://www.baeldung.com/cs/credit-assignment-problem
3. https://ai-com.tistory.com/entry/RL-%EA%B0%95%ED%99%94%ED%95%99%EC%8A%B5-%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98-5-PPO