RL Research Group 20250626

FastTD3: Simple, Fast, and Capable Reinforcement Learning for Humanoid Control

(ArXiv 2025)

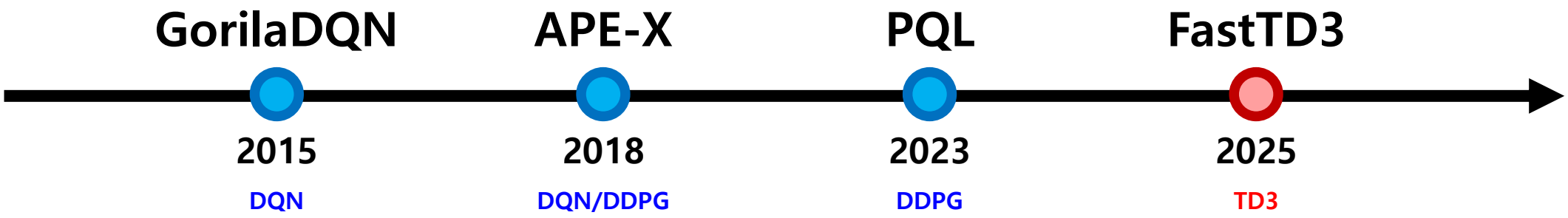
김재훈

Introduction

❖ Distributed vs. Distributional

- Distributed RL (분산 강화학습) → 많은 연산 자원을 활용해 병렬 환경을 구축하여 시간 효율적인 강화학습을 수행 (e.g., Ape-X)
- Distributional RL (분포 기반 강화학습) → 보상 분포를 학습하는 것을 목표로 하는 강화학습 (e.g., C51)

→ 이름은 비슷하지만 내용은 완전히 다르다!



GorilaDQN

❖ **Massively Parallel Methods for Deep Reinforcement Learning (ICML 2015)**

- 기존의 강화학습은 단일 연산 자원(ex. CPU x 1ea)에서만 진행이 되어 학습 시간이 오래 걸림
- 따라서 다수의 연산 자원을 활용한 병렬 처리 방식으로 학습 속도를 향상시키고자 함

Training on
a single machine

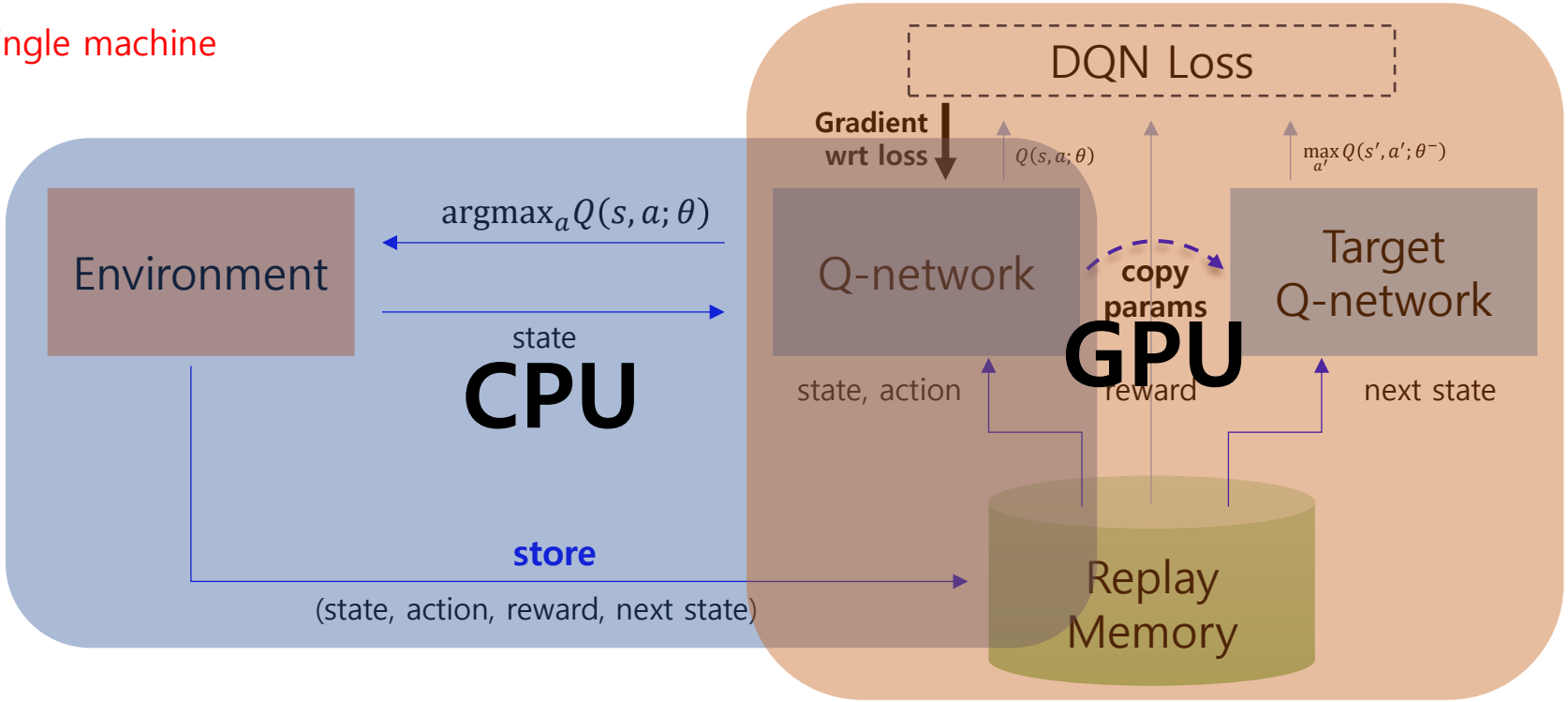
Training on
multiple machines

GorilaDQN

❖ **Massively Parallel Methods for Deep Reinforcement Learning (ICML 2015)**

- 기존의 강화학습은 단일 연산 자원(ex. CPU x 1ea)에서만 진행이 되어 학습 시간이 오래 걸림
- 따라서 다수의 연산 자원을 활용한 병렬 처리 방식으로 학습 속도를 향상시키고자 함

** Running DQN on a single machine



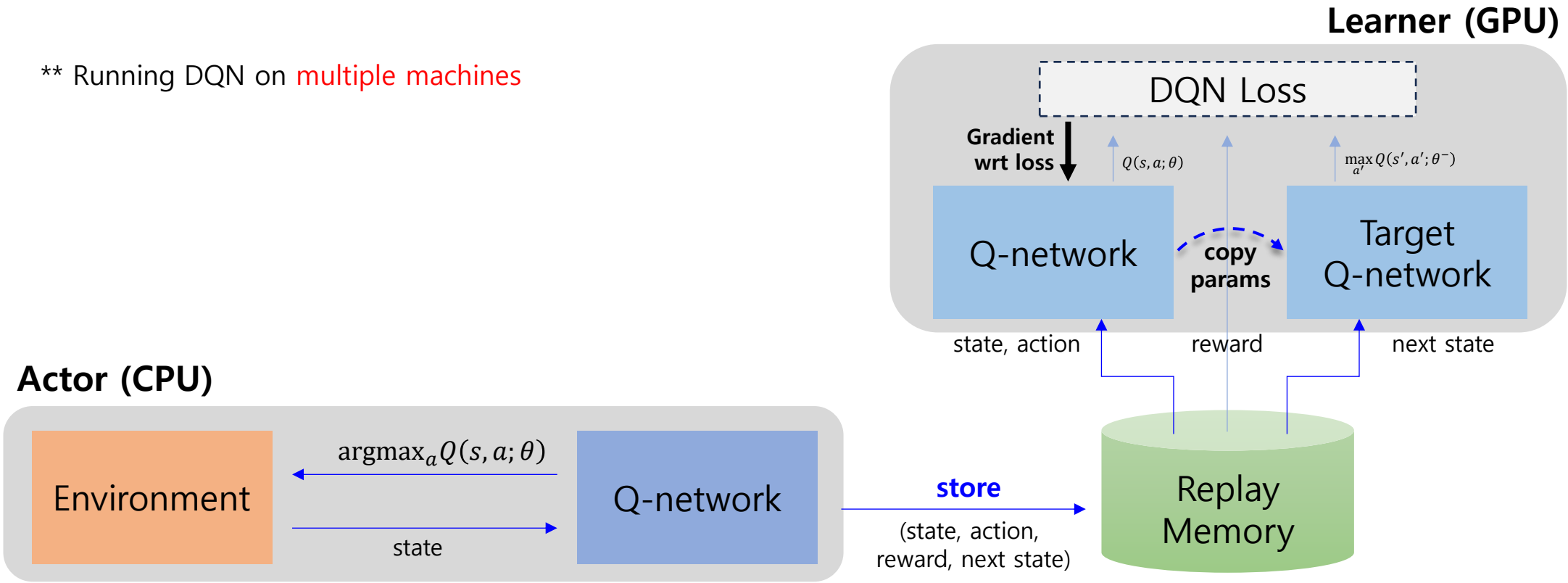
Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., De Maria, A., ... & Silver, D. (2015). Massively parallel methods for deep reinforcement learning. arXiv preprint arXiv:1507.04296.

GorilaDQN

❖ **Massively Parallel Methods for Deep Reinforcement Learning (ICML 2015)**

- 역할을 actor와 learner로 분리하여 학습을 수행 (이때 actor는 CPU, learner는 GPU로 구동)
- Actor는 환경에서 경험을 수집하고 learner는 수집한 경험으로 손실함수를 구함

** Running DQN on multiple machines

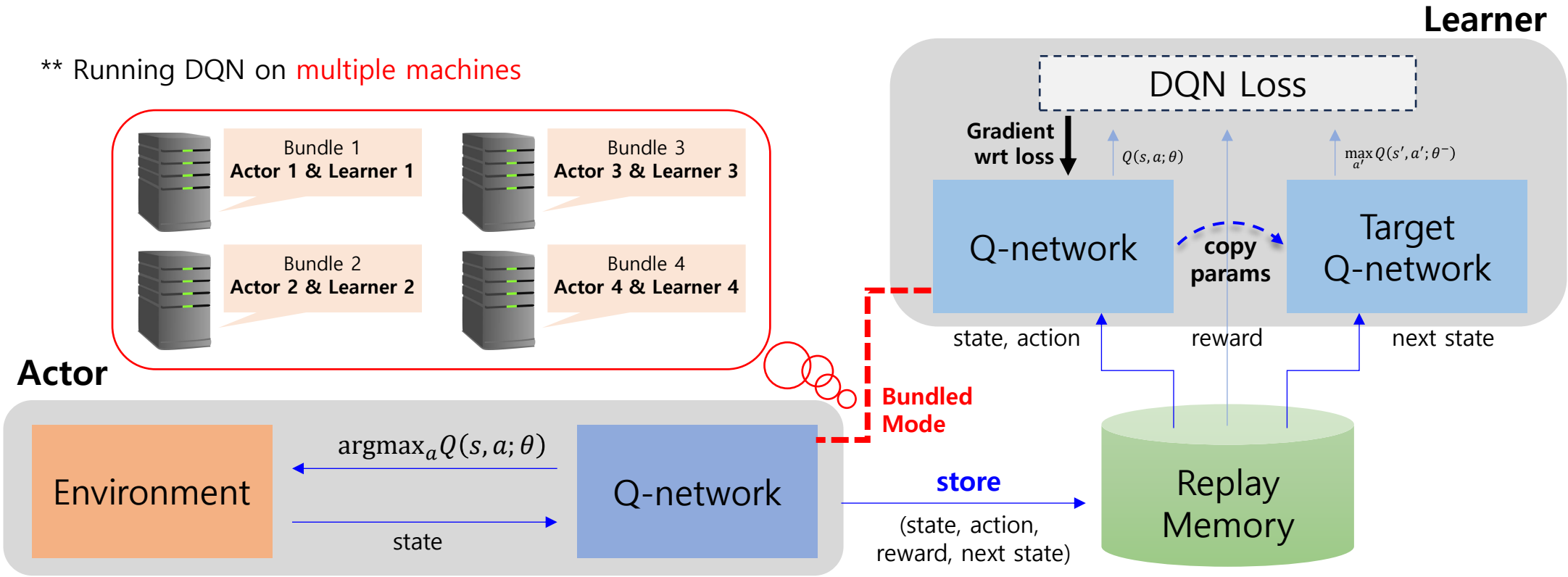


Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., De Maria, A., ... & Silver, D. (2015). Massively parallel methods for deep reinforcement learning. arXiv preprint arXiv:1507.04296.

GorilaDQN

❖ **Massively Parallel Methods for Deep Reinforcement Learning (ICML 2015)**

- Actor와 learner의 Q-network는 동일한 파라미터를 가짐
- Bundled mode를 쓰면 각 연산 자원마다 고유의 actor, learner, replay memory를 가짐

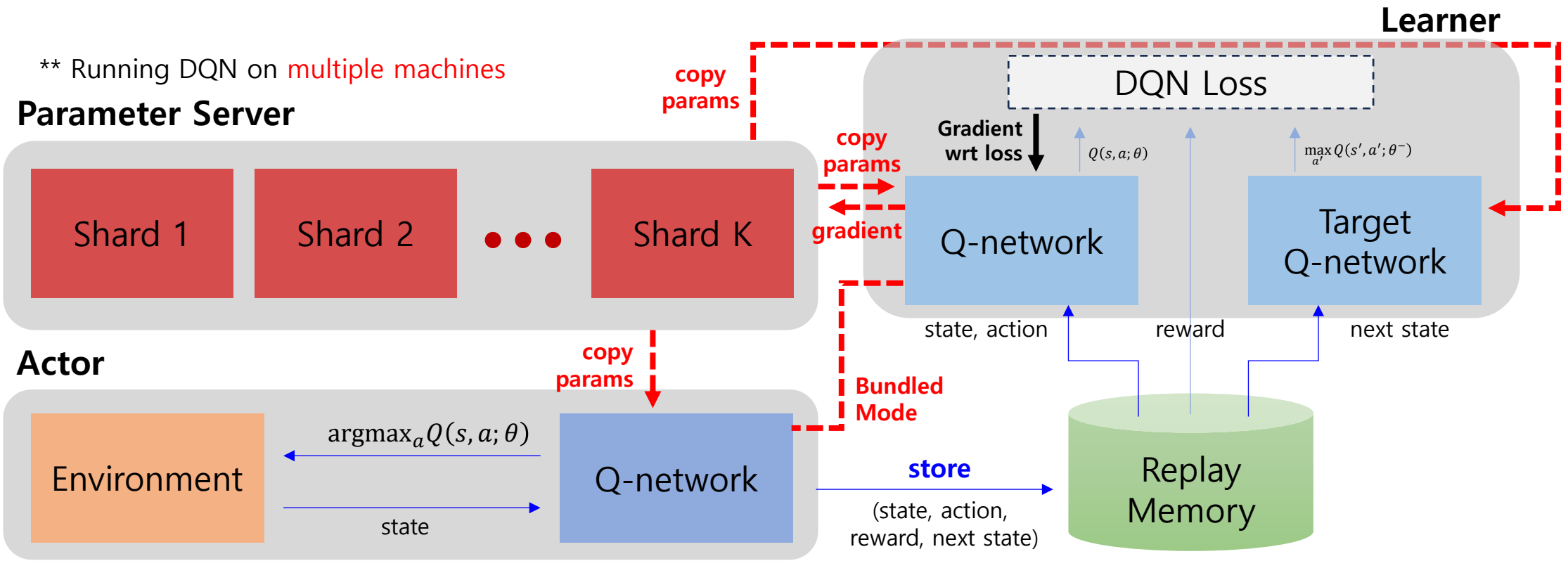


Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., De Maria, A., ... & Silver, D. (2015). Massively parallel methods for deep reinforcement learning. arXiv preprint arXiv:1507.04296.

GorilaDQN

❖ **Massively Parallel Methods for Deep Reinforcement Learning (ICML 2015)**

- Parameter server는 각 bundle로부터 연산된 기울기를 수집하고 업데이트한 Q-network 파라미터를 각 bundle에 배포함



Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., De Maria, A., ... & Silver, D. (2015). Massively parallel methods for deep reinforcement learning. arXiv preprint arXiv:1507.04296.

GorilaDQN

❖ Experiment (performance comparison)

- Single GPU DQN은 최대 14일 동안 게임 49개를 학습
- GorilaDQN은 4일만에 Single DQN보다 41개 게임에서 더 좋은 성능을 달성
- GorilaDQN으로 사람보다 좋은 성능을 49개 게임 중 24개에서 달성

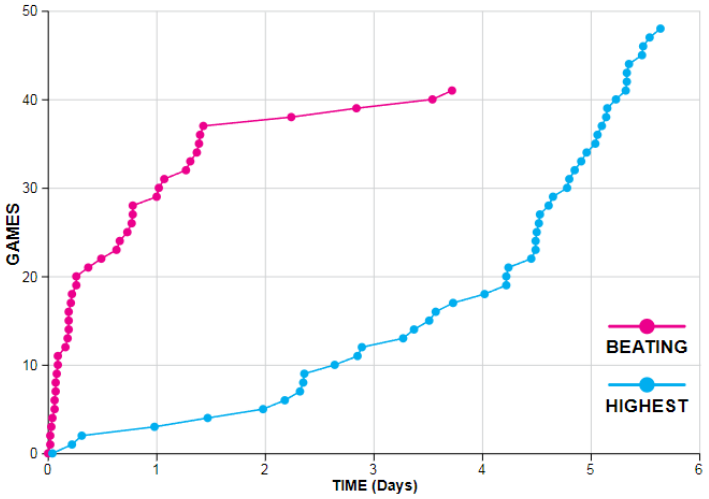


Figure 5. The time required by Gorila DQN to surpass single DQN performance (red curve) and to reach its peak performance (blue curve).

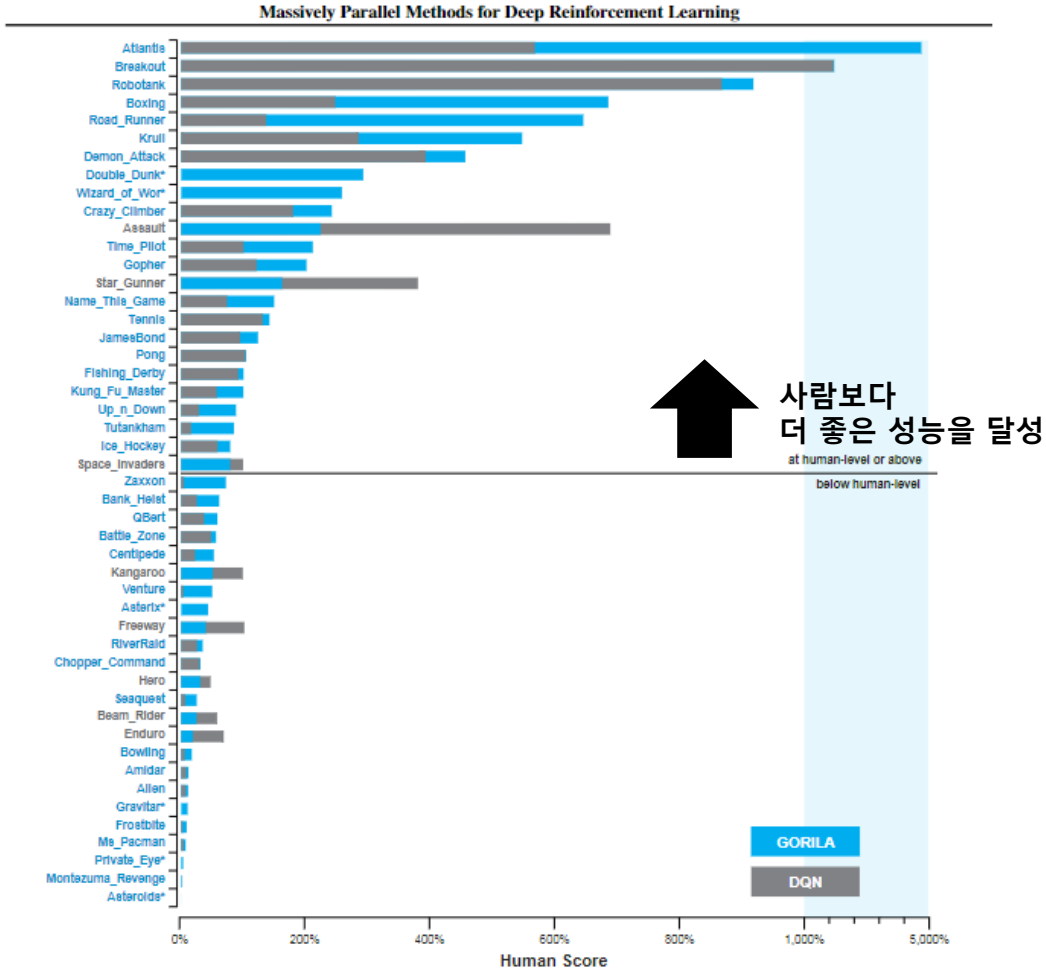


Figure 3. Performance of the Gorila agent on 49 Atari games with human starts evaluation compared with DQN (Mnih et al., 2015) performance with scores normalized to expert human performance. Font color indicates which method has the higher score. *Not showing DQN scores for Asterix, Asteroids, Double Dunk, Private Eye, Wizard Of Wor and Gravitar because the DQN human starts scores are less than the random agent baselines. Also not showing Video Pinball because the human expert scores are less than the random agent scores.

Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., De Maria, A., ... & Silver, D. (2015). Massively parallel methods for deep reinforcement learning. arXiv preprint arXiv:1507.04296.

Ape-X

❖ **Distributed Prioritized Experience Replay (ICLR 2018)**

- 기존 연구에서는 분산 학습을 통해 기울기 연산을 병렬로 진행하는 것에 주로 초점이 맞춰져 있었음
- 이번 연구에서는 분산 학습을 통해 경험을 수집하고 저장하는 부분을 개선하는데 초점을 맞춤

** 기존의 분산 학습 방식과의 비교

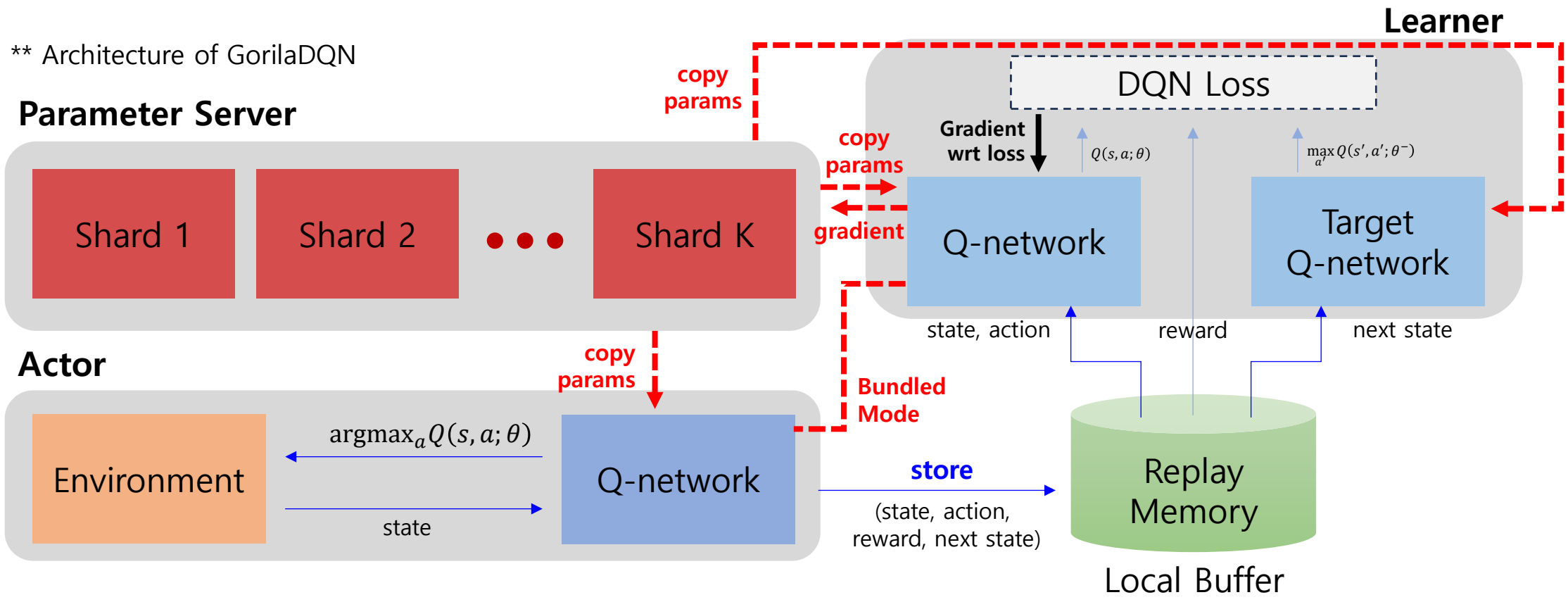
	Gorila (Nair. et al., 2015)	**Ape-X (Horgan. et al., 2018)**
Architecture for distributed learning	Yes (same number of actors & learners)	No (single learner & multiple actors)
Assign different ϵ to each actors to collect diverse experience	No	Yes
Calculating gradients	Parallel	Non-parallel
Experience replay strategy	Local buffer & uniform sampling	Global buffer & prioritized sampling

Ape-X

❖ **Distributed Prioritized Experience Replay (ICLR 2018)**

- 기존의 Gorila는 여러 묶음의 actor와 learner를 사용하였으며 메인 모델 파라미터를 별도의 서버에서 업데이트하는 방식을 사용
- Ape-X는 parameter server, bundled mode를 사용하지 않음 → Single machine에서도 사용 가능

** Architecture of GorilaDQN

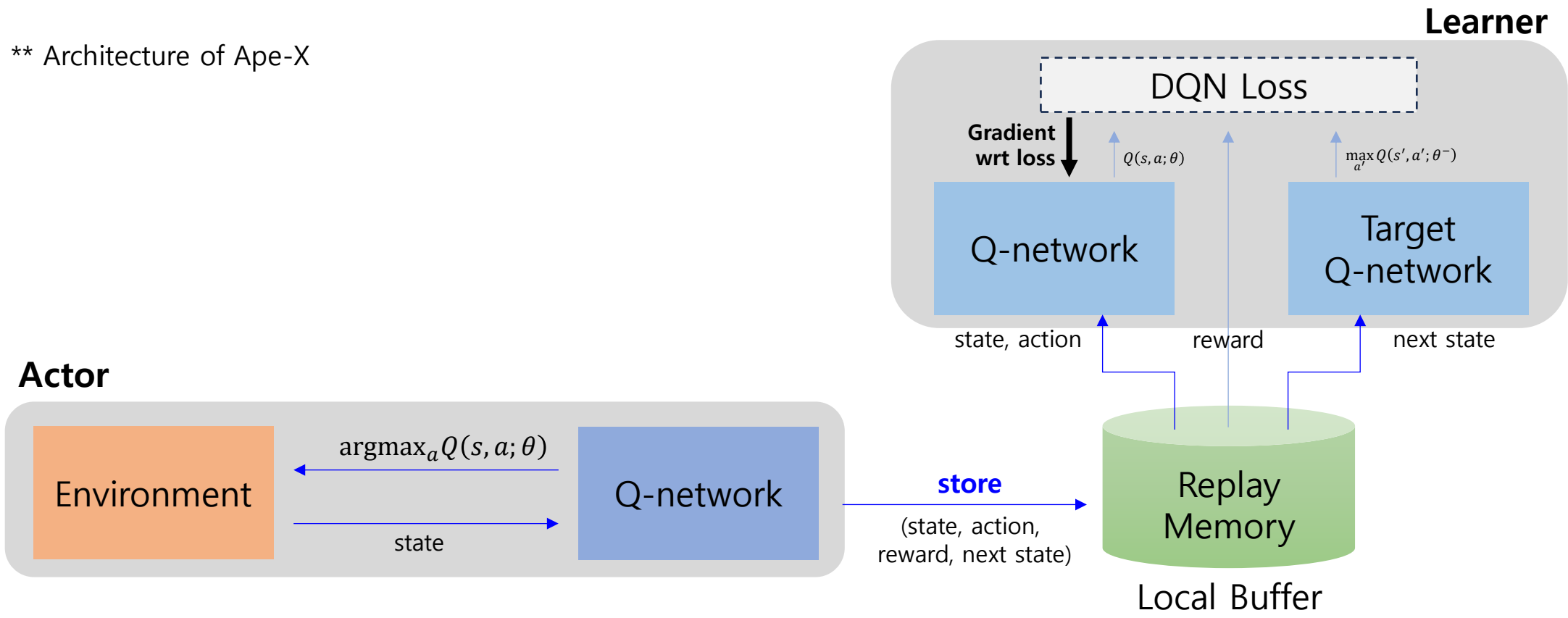


Ape-X

❖ **Distributed Prioritized Experience Replay (ICLR 2018)**

- 기존의 Gorila는 여러 묶음의 actor와 learner를 사용하였으며 메인 모델 파라미터를 별도의 서버에서 업데이트하는 방식을 사용
- Ape-X는 parameter server, bundled mode를 사용하지 않음 → Single machine에서도 사용 가능

** Architecture of Ape-X

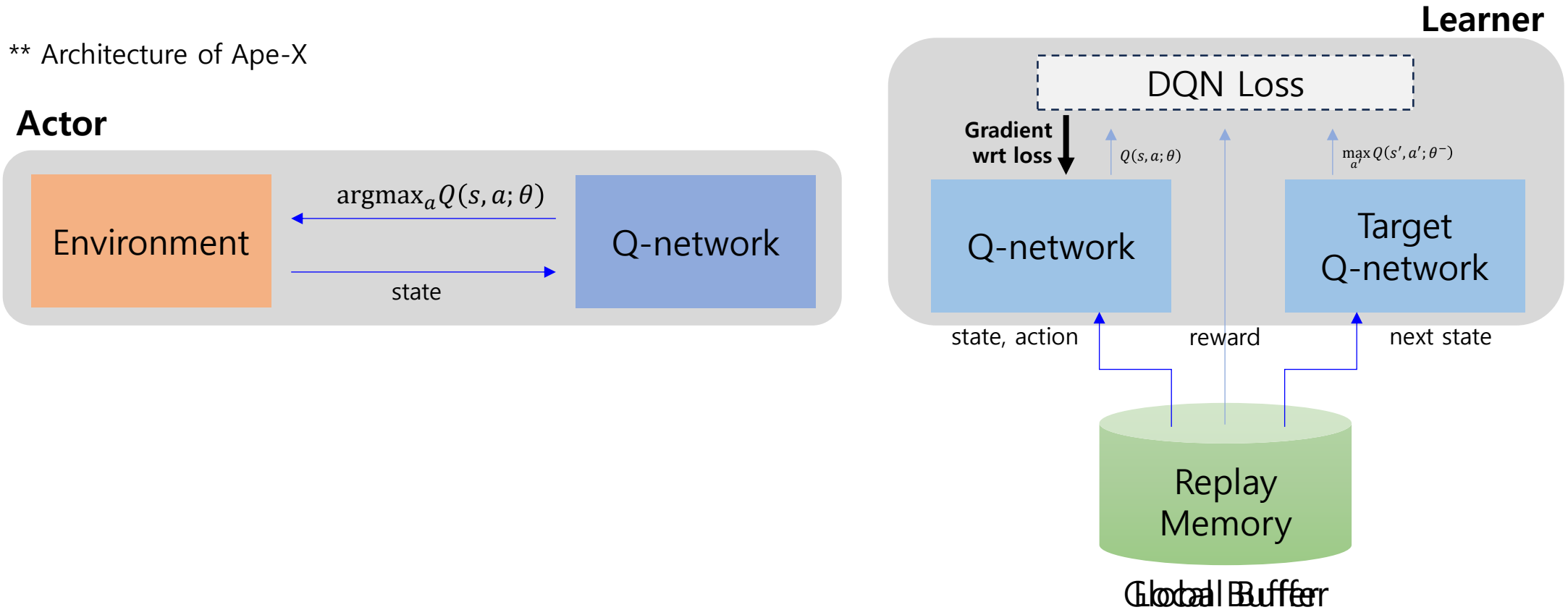


Ape-X

❖ **Distributed Prioritized Experience Replay (ICLR 2018)**

- Local / global buffer를 모두 사용하며 actor가 수집한 정보는 우선 local buffer에 저장

** Architecture of Ape-X

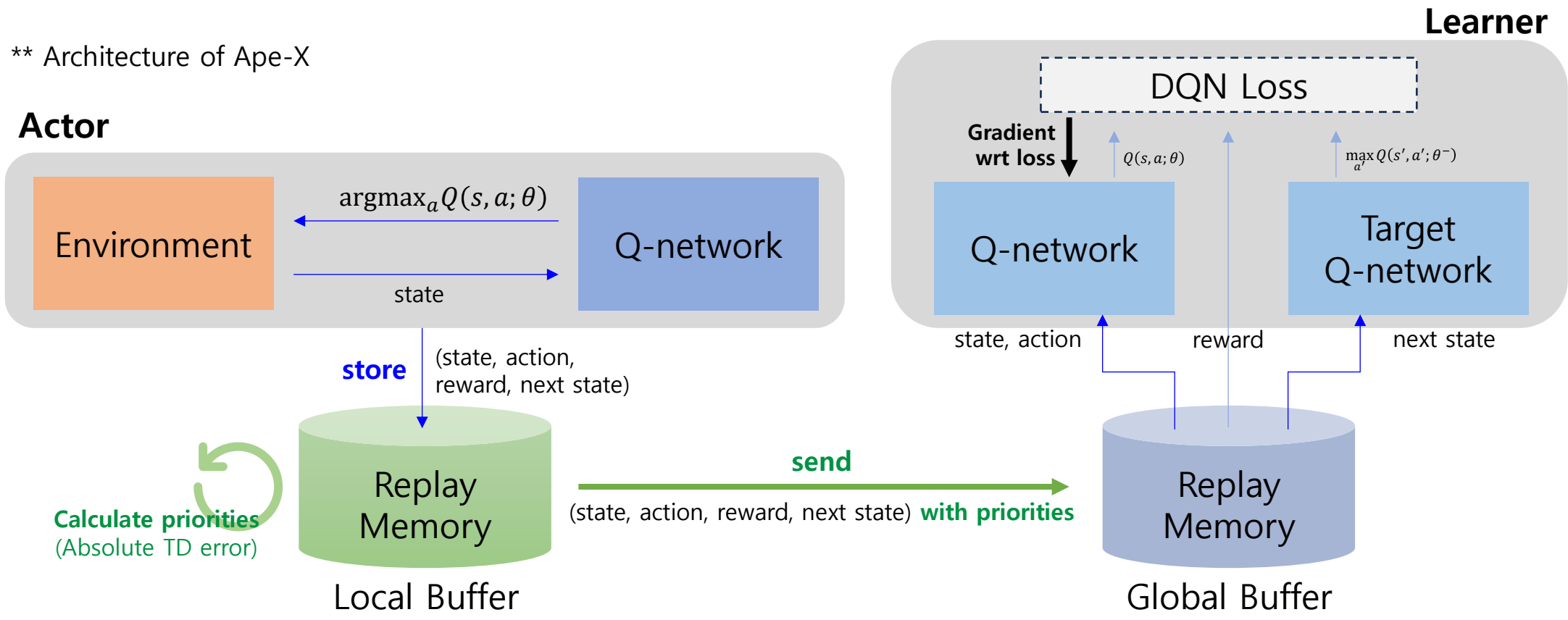


Ape-X

❖ **Distributed Prioritized Experience Replay (ICLR 2018)**

- Local / global buffer를 모두 사용하며 actor가 수집한 정보는 우선 local buffer에 저장
- Local buffer에서 priority 정보를 추가하여 global buffer에 이전

** Architecture of Ape-X

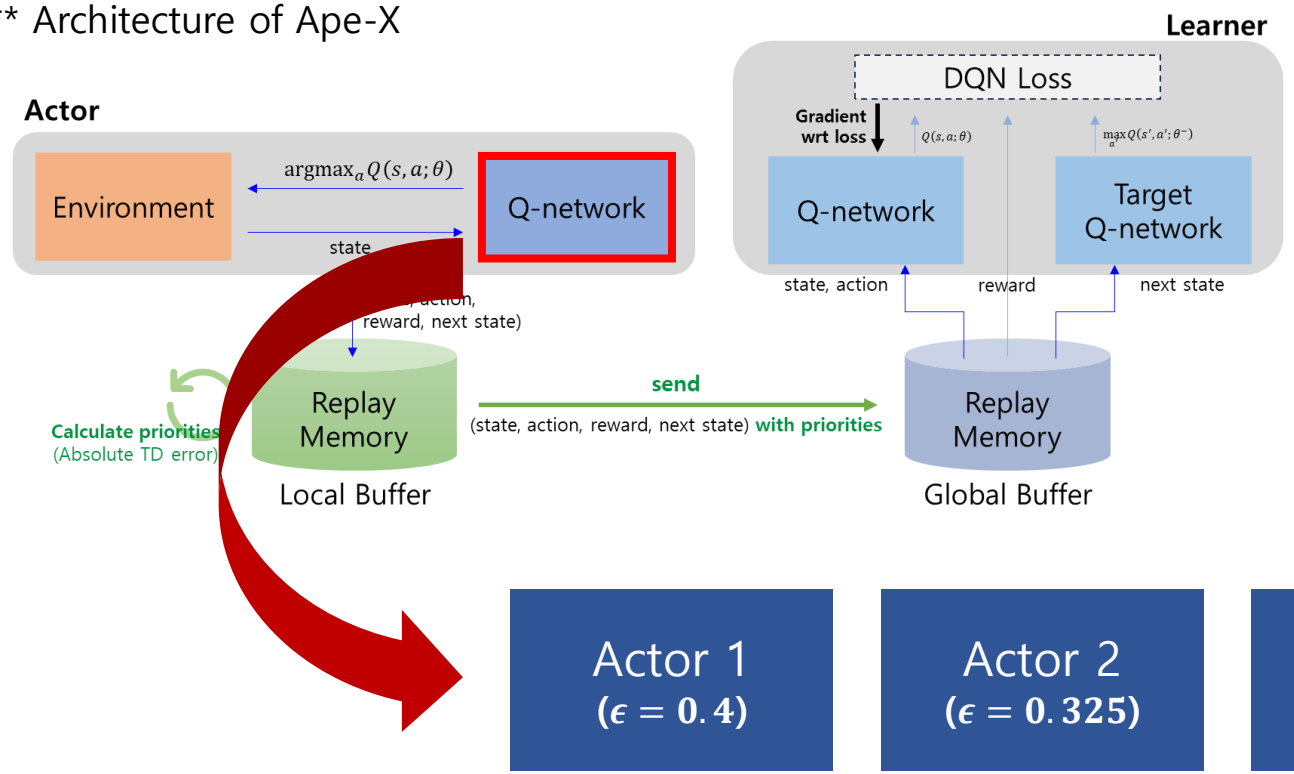


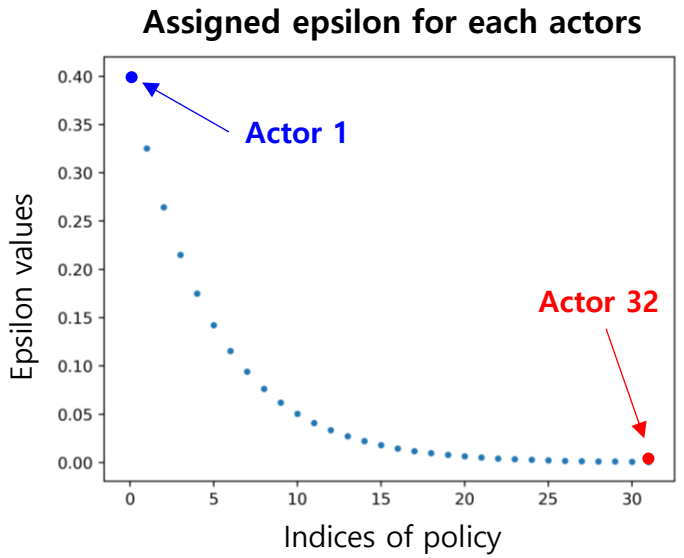
Ape-X

❖ Distributed Prioritized Experience Replay (ICLR 2018)

- Actor마다 탐험 정도(ϵ)를 다양하게 부여하여 더 다양한 경험을 수집하고자 함

** Architecture of Ape-X





Actor 1	Actor 2	Actor 3	...	Actor 32
$(\epsilon = 0.4)$	$(\epsilon = 0.325)$	$(\epsilon = 0.264)$		$(\epsilon = 0.0007)$

Ape-X

❖ Experiment (performance comparison)

- 학습 속도와 성능 모두 뛰어나게 향상된 것을 확인
- 한정된 시간 내에서 actor 수가 늘어남에 따라 더 높은 점수를 빠르게 달성

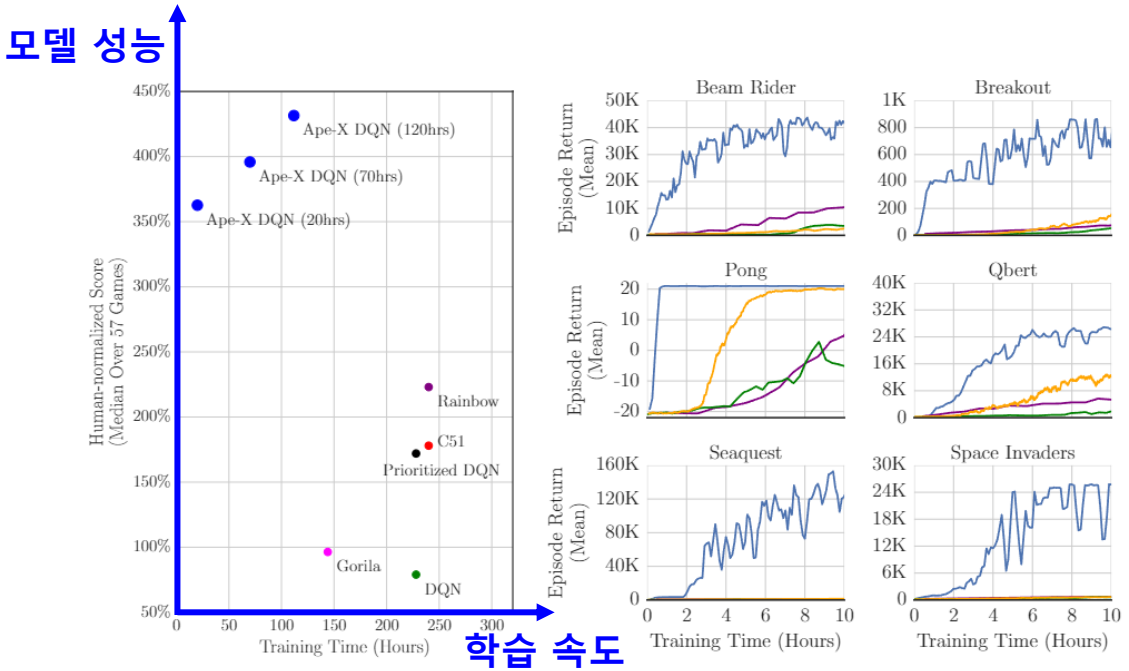


Figure 2: Left: Atari results aggregated across 57 games, evaluated from random no-op starts. Right: Atari training curves for selected games, against baselines. Blue: Ape-X DQN with 360 actors; Orange: A3C; Purple: Rainbow; Green: DQN. See appendix for longer runs over all games.

다른 방법론과의 성능 비교

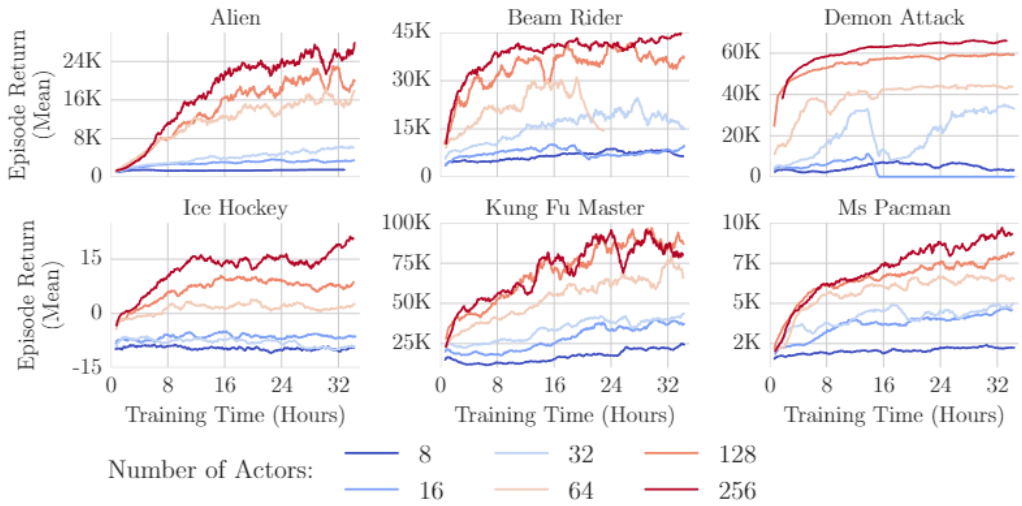


Figure 4: Scaling the number of actors. Performance consistently improves as we scale the number of actors from 8 to 256, note that the number of learning updates performed does not depend on the number of actors.

Actor 수에 따른 성능 비교

Ape-X

❖ **Experiment (performance comparison detail)**

- Ape-X에서 actor는 360 core를 사용 (1 core = 1 actor), learner는 1 GPU + 16 core를 사용
- 다른 단일 학습 그리고 분산 학습 방법론에 비해서 더 시간 효율적으로 높은 성능을 달성하는 것을 확인

Algorithm	Training Time	Environment Frames	Resources (per game)	Median (no-op starts)	Median (human starts)
Ape-X DQN	5 days	22800M	376 cores, 1 GPU [a]	**434%**	**358%**
Rainbow	10 days	200M	1 GPU	223%	153%
Distributional (C51)	10 days	200M	1 GPU	178%	125%
A3C	4 days	—	16 cores	—	117%
Prioritized Dueling	9.5 days	200M	1 GPU	172%	115%
DQN	9.5 days	200M	1 GPU	79%	68%
Gorila DQN [c]	∼4 days	—	unknown [b]	96%	78%
UNREAL [d]	—	250M	16 cores	331% [d]	250% [d]

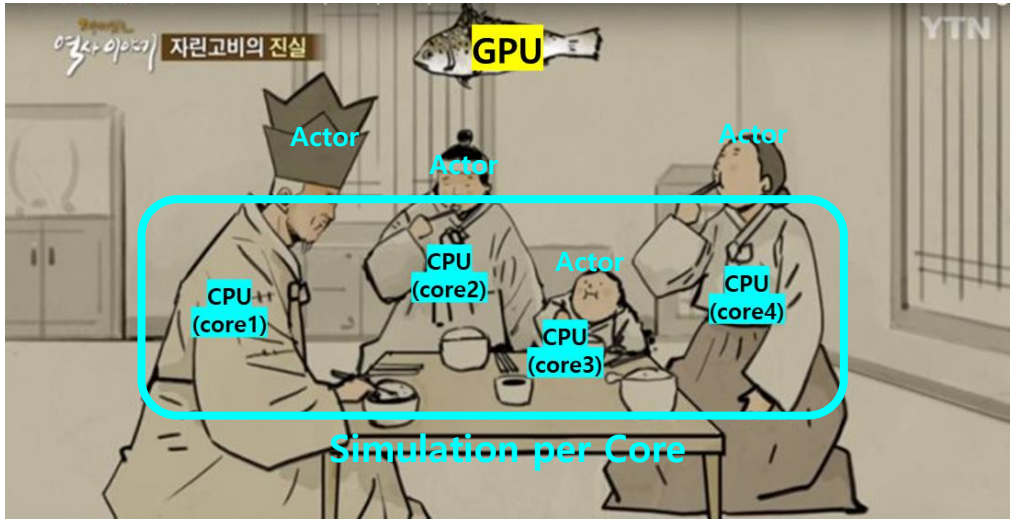
Table 1: Median normalized scores across 57 Atari games. [a] Tesla P100. [b] >100 CPUs, with a mixed number of cores per CPU machine. [c] Only evaluated on 49 games. [d] Hyper-parameters were tuned per game.

PQL

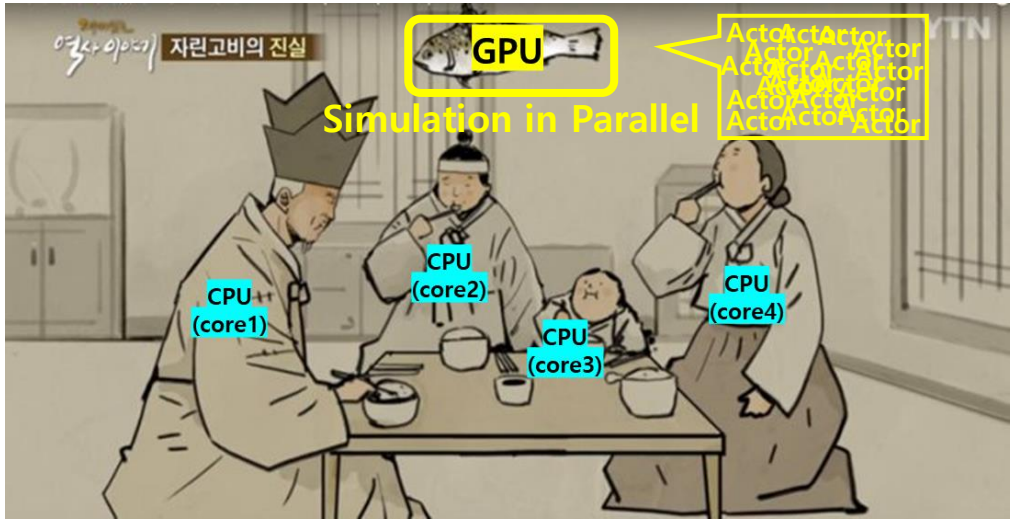
❖ **Parallel Q-Learning: Scaling Off-policy Reinforcement Learning under Massively Parallel Simulation (ICML 2023)**

- 기존에는 각 CPU 코어마다 시뮬레이션을 수행하는 방식을 진행 → 코어 수는 한정적이므로 병렬 학습의 scaling에 한계 (Ape-X <= 256)
- GPU 기반의 시뮬레이션(e.g., Issac Gym)이 가능해지면서 병렬 학습의 massive scaling이 가능해짐 (PQL >> 1,000)

  → 이에 적합한 Q-learning 알고리즘을 제안 (**Parallel Q-Learning**)



Ape-X



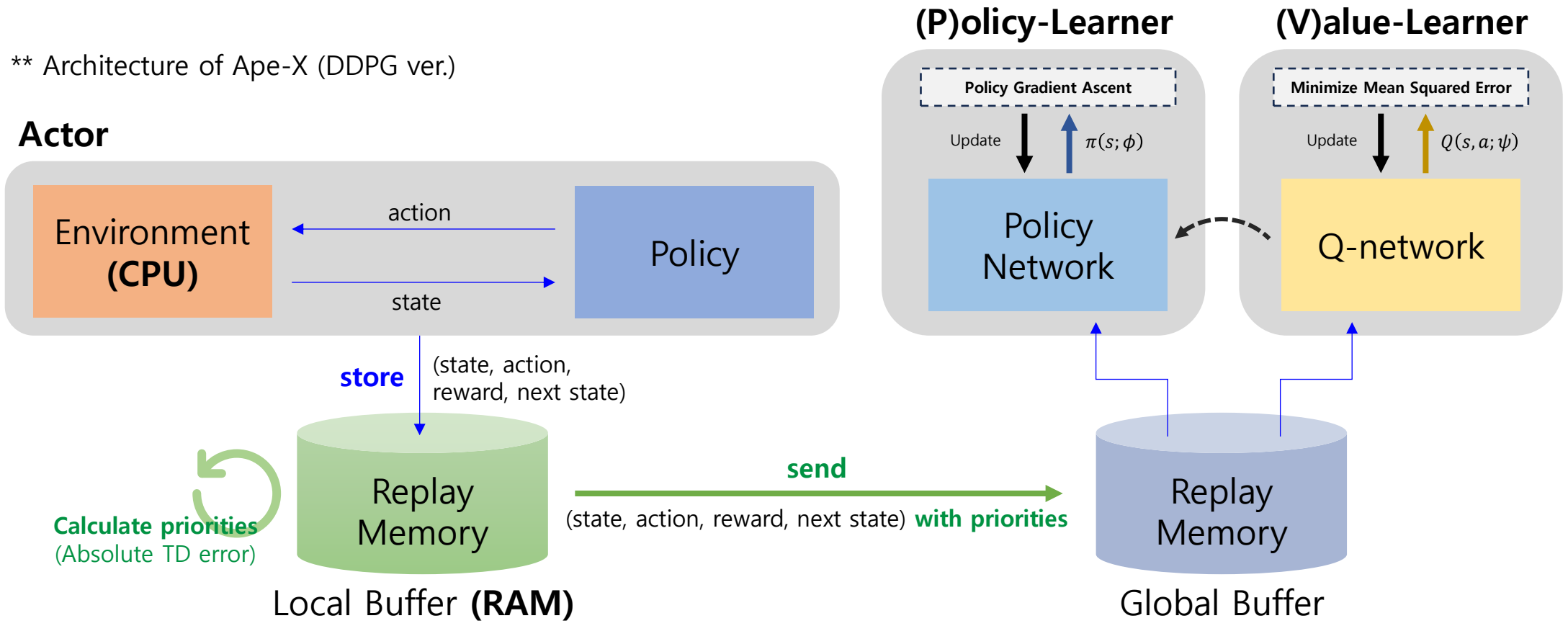
PQL

PQL

❖ Architecture

- Ape-X + DDPG를 기반으로 PQL을 구성
- Ape-X는 Actor로 데이터 수집 → Value-Learner → Policy-Learner **순차적**으로 수행

** Architecture of Ape-X (DDPG ver.)

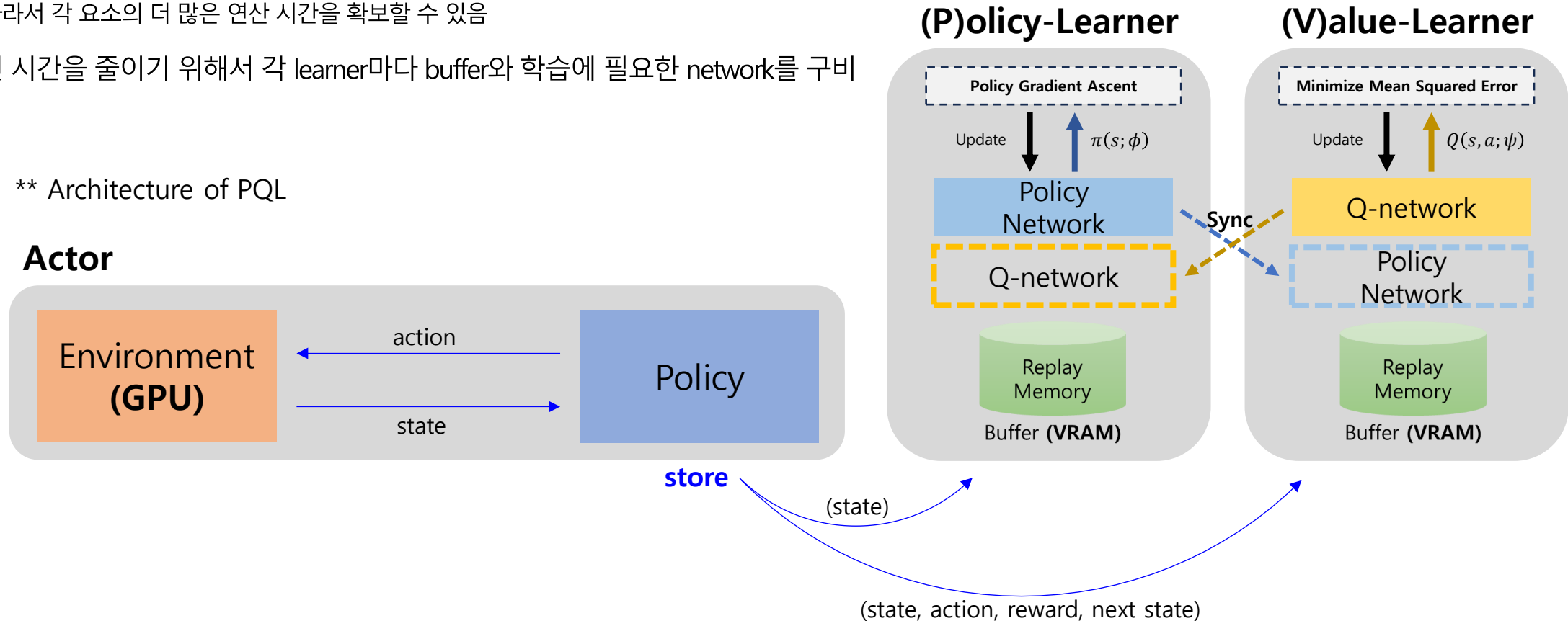


PQL

❖ Architecture

- Ape-X + DDPG를 기반으로 PQL을 구성
- PQL은 Actor로 데이터수집 & Policy-Learner & Value-Learner를 **동시에** 수행
  → 따라서 각 요소의 더 많은 연산 시간을 확보할 수 있음
- 통신 시간을 줄이기 위해서 각 learner마다 buffer와 학습에 필요한 network를 구비

** Architecture of PQL



PQL

❖ Update Frequency Ratio (β)

- f_a : The number of rollout steps per environment in Actor per unit time
- f_v : The number of Q function updates in V-learner per unit time
- f_p : The number of policy updates in P-learner per unit time
- $\boldsymbol{\beta_{a:v}} = \frac{f_a}{f_v}$: Determines how many steps Actor rolls out the policy with N environments when one Q function update is performed in the V-learner
- $\boldsymbol{\beta_{p:v}} = \frac{f_p}{f_v}$: Determines how many Q functions updates are performed in V-leaner when P-learner updates the policy once.

→ $\boldsymbol{\beta_{a:v}}$ = 1 : 8이면 Actor가 8 step을 수행할 때마다 V-Learner를 1번 업데이트

→ $\boldsymbol{\beta_{p:v}}$ = 1 : 2면 V-Learner를 2번 업데이트할 때 P-Learner는 1번 업데이트

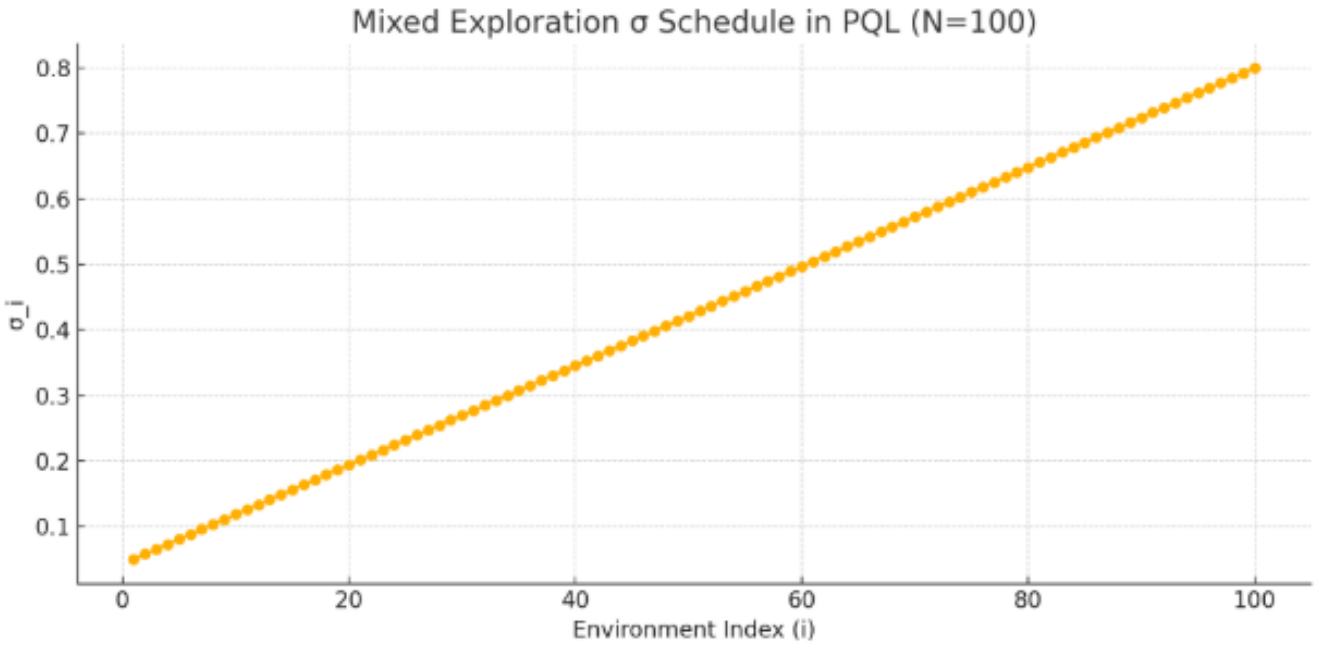
이 방식의 장점

1. 각 Learner와 Actor의 균형 있는 자원 사용
2. 자주 업데이트하면 학습이 불안정하므로 학습 속도를 조정 (안정성과 수렴 속도 확보)
3. Actor의 policy가 일정 주기로 동기화되는 구조라서 별도의 target policy network를 두지 않아도 동일한 역할을 수행

PQL

❖ Mixed Exploration

- Ape-X의 경우 각 actor 마다 exploration의 강도를 조금씩 다르게 부여하여 다양한 경험을 수집
- PQL도 이를 반영하였으며 고유의 노이즈 스케쥴 방식을 제안



2016). In our work, we uniformly generate the noise levels in the range of $[\sigma_{\min}, \sigma_{\max}]$. For the i^{th} environment out of N environments, $\sigma_i = \sigma_{\min} + \frac{i-1}{N-1}(\sigma_{\max} - \sigma_{\min})$ where $i \in \{1, 2, ..., N\}$. We use $\sigma_{\min} = 0.05, \sigma_{\max} = 0.8$ for all the tasks in our experiments.

** 3.3 Mixed Exploration

PQL

❖ Experiment

Batch Size & Buffer Size & # of GPU

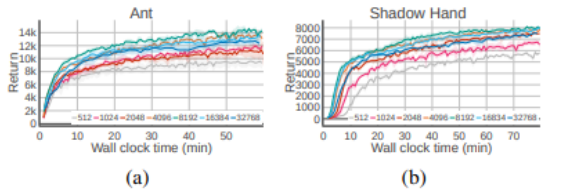


Figure 8. Effect of different batch sizes. Small batch size usually leads to slower learning. If the batch size is too big, the policy learning can slow down because GPUs have limited cores and it takes more time to process a very big batch of data.

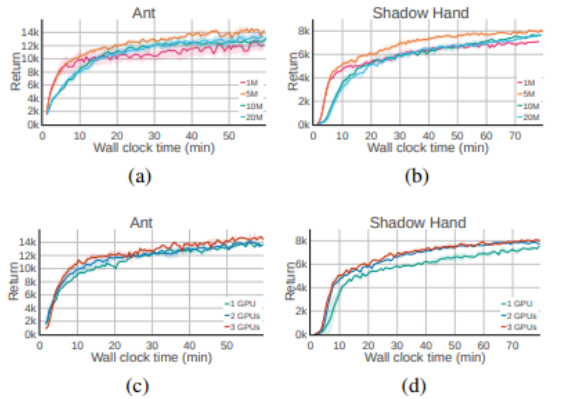


Figure 9. **(a)** and **(b)**: effect of different replay buffer size. **(c)** and **(d)**: effect of number of GPUs used for running PQL. PQL can be deployed on a flexible number of GPUs. In complex tasks such as *Shadow Hand*, it is beneficial to have at least 2 GPUs where the **Actor** runs on a separate GPU as the simulation itself consumes more GPU compute as the task complexity increases.

of Envs + Update Frequency

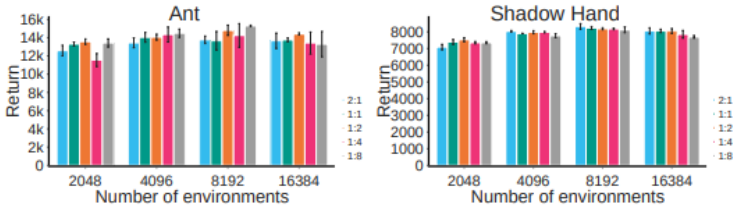


Figure 6. We show the averaged returns in evaluation after a fixed amount of training time ΔT. Across the set of different numbers of environments we experimented with $(2048, 4096, 8192, 16384)$, we found that setting $\beta_{p:v} = 1:2$ generally works well. $\Delta T = 60$ mins for *Ant*, and $\Delta T = 80$ mins for *Shadow Hand*. The complete learning curves are in Figure C.6.

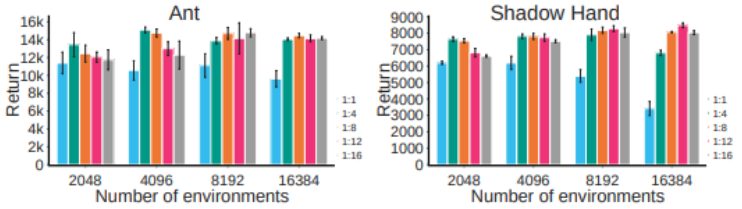


Figure 7. Given different values of N, we show the effect of different $\beta_{a:v}$. An overall trend we observe is that as N gets bigger, it's more beneficial to update the critic more frequently. We also found that $\beta_{a:v} = 1:8$ generally works well given different N values. So one can set $\beta_{a:v} = 1:8$ as a good initial value, and tune it if necessary on new tasks.

Mixed Exploration

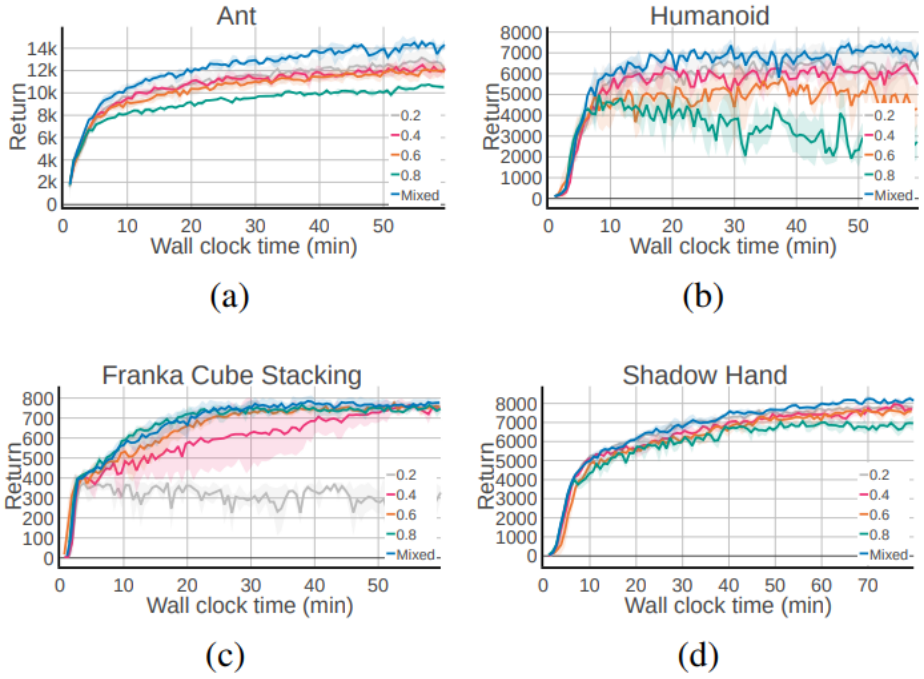


Figure 4. We compared our proposed mixed exploration scheme by applying different constant maximum noise values. We can see that the mixed exploration scheme either outperforms or is on par with other schemes, which can save the tuning effort on the noise level.

FastTD3

❖ **FastTD3: Simple, Fast, and Capable Reinforcement Learning for Humanoid Control (ArXiv 2025)**

- FastTD3는 PQL + TD3를 기반으로 구성됨 → PQL을 개선한 연구
- PQL의 단점으로는 asynchronous parallel process로 인한 활용의 복잡함이 있음

  → Asynchronous parallel process 요소로 인해 방법론이 확산되는데 제약 (PQL의 현재 인용 수는 15회)

  → FastTD3는 이를 **synchronous parallel process로 변경**하고 **최적화된 세팅**을 소개
- 주요 환경을 HumanoidBenchmark로 정한 이유는 아직 강화학습 방법론 중에 이를 48시간 내로 학습에 성공한 케이스가 없어서인듯...? (추측)

FastTD3

❖ PQL vs. FastTD3

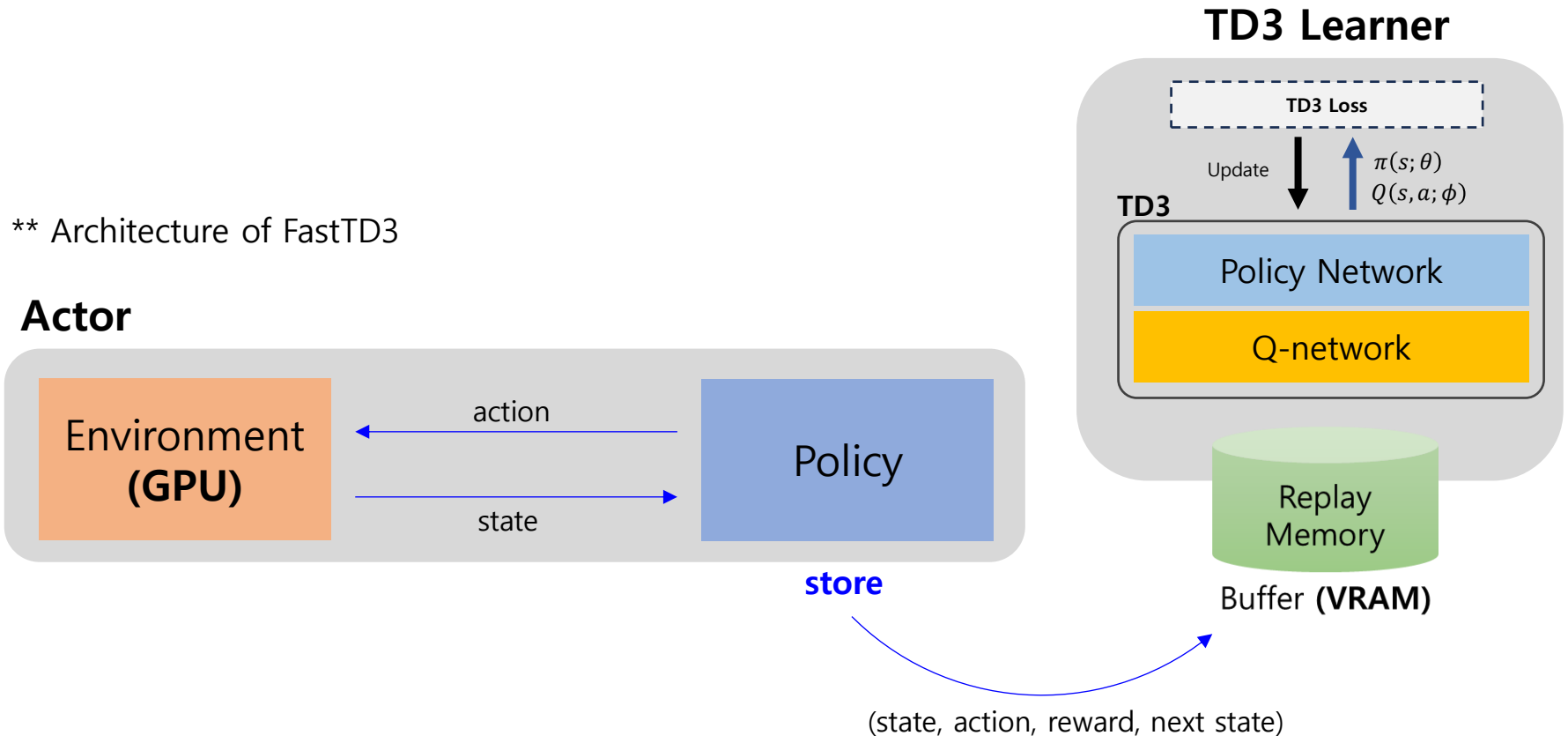
	PQL	FastTD3
Parallel Process	Asynchronous → 각 P/V-Learner가 동시에 업데이트	Synchronous → 하나의 Learner에서 policy/Q-network 순차적 업데이트
Replay Buffer	각 P/V-Learner에 Local Buffer → 1 million이 가장 좋은 성능 (1M, 5M, 10M, 20M 중)	Global Buffer (Size: N x envs) → N이 크면 클수록 성능이 향상
Large Batch Size	32,768일 때의 성능이 가장 좋음	32,768일 때의 성능이 가장 좋음
Exploration Noise	Mixed Exploration ($\sigma_{\min}$: 0.2 / σ_{max}: 0.8)	Mixed Exploration ($\sigma_{\min}$: 0.2 / σ_{max}: 0.4)
Distributional Critic	Optional (PQL-D)	Default
Asymmetric Actor-Critic	X	O
Update Frequency	V-Learner의 업데이트 1회에 따른 Actor ($\beta_{a:v}$) 및 P-Learner ($\beta_{p:v}$)의 업데이트 횟수를 설정	1. Update-to-Data (UTD) ratio 2. Delayed update strategy in TD3
Advanced Usage of PyTorch	No	1. torch.compile( ): 최대 35% 속도 개선 2. AMP(bfloat16): 최대 40% 속도 개선 } 최대 70% 속도 개선
Reward Tuning In Mujoco Playground	No	TD3에 알맞도록 reward 함수를 수정함 (기존에는 PPO에 최적인 reward 함수였음)

https://github.com/younggyoseo/FastTD3/blob/main/fast_td3/training_notebook.ipynb

FastTD3

❖ **Architecture**

** Architecture of FastTD3



https://github.com/younggyoseo/FastTD3/blob/main/fast_td3/training_notebook.ipynb

FastTD3

❖ Experiment

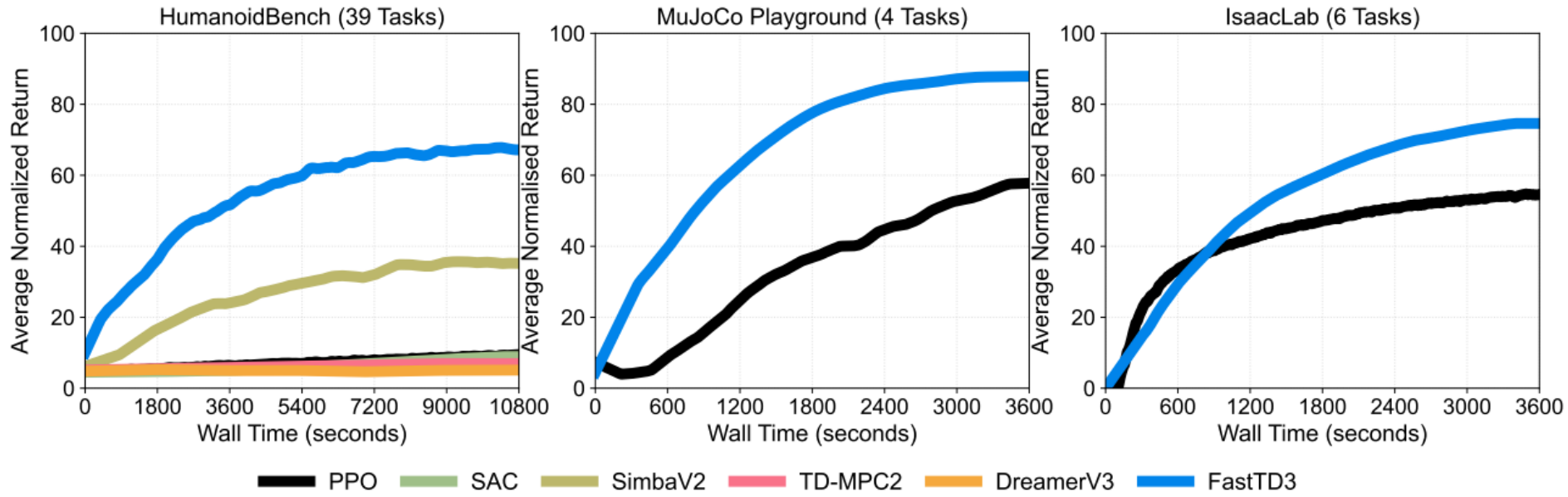
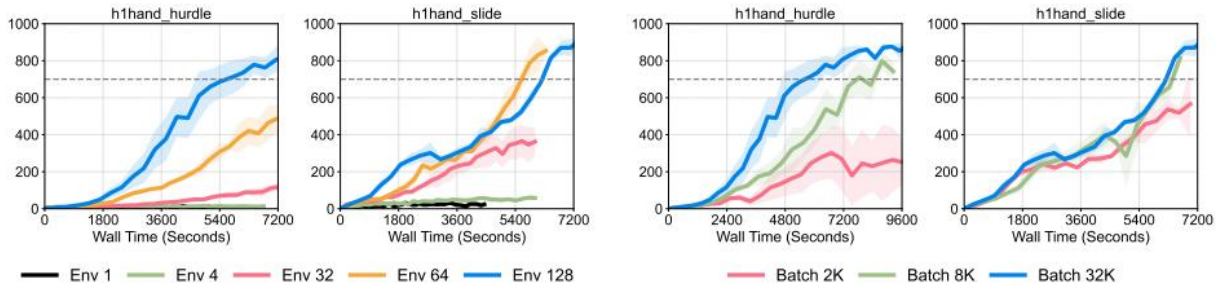


Figure 3: **Summary of results.** FastTD3 is a simple, fast, and capable RL algorithm that significantly speeds up training for humanoid robots on tasks from popular suites such as HumanoidBench (Sferrazza et al., 2024), IsaacLab (Mittal et al., 2023), and MuJoCo Playground (Zakka et al., 2025). To accelerate RL research in robotics, we provide an easy-to-use open-source implementation of FastTD3, enabling users to easily reproduce these results or build upon our work.

FastTD3

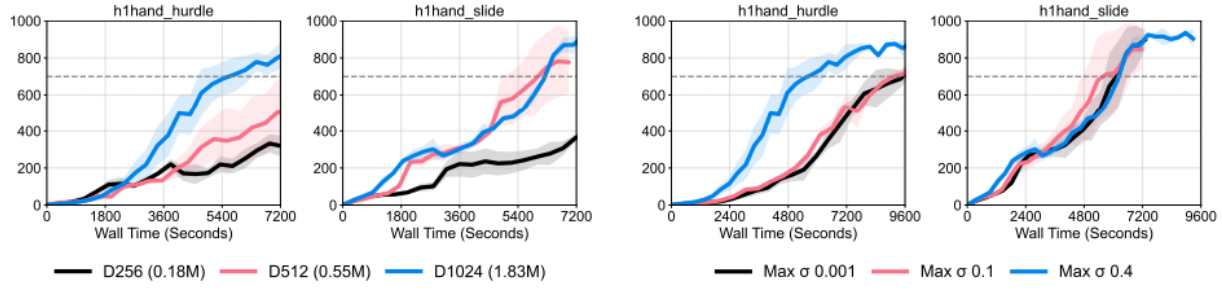
❖ Experiment



(a) Effect of parallel environments (b) Effect of batch size

(c) Effect of distributional RL (d) Effect of Clipped Double Q-learning

Figure 5: **Effect of design choices (1 / 2).** We investigate the effect of (a) parallel environments, (b) batch size, (c) distributional RL, and (d) Clipped Double Q-learning. The solid line and shaded regions represent the mean and standard deviation across three runs.



(a) Effect of model size (b) Effect of noise scales

(c) Effect of update-to-data ratio (d) Effect of replay buffer size

Figure 6: **Effect of design choices (2 / 2).** We investigate the effect of (a) model size, (b) noise scales, (c) update-to-data ratio, and (d) replay buffer size. The solid line and shaded regions represent the mean and standard deviation across three runs.

UTD 크기와 연산 시간 간의 trade-off 존재
→ 학습 시간 및 성능에 영향

FastTD3

❖ Experiment

```
if use_tuned_reward and is_humanoid_task:
    # NOTE: Tuned reward for G1. Used for producing Figure 7 in the paper.
    # Somehow it works reasonably for T1 as well.
    # However, see `sim2real.md` for sim-to-real RL with Booster T1
    train_env_cfg.reward_config.scales.energy = -5e-5
    train_env_cfg.reward_config.scales.action_rate = -1e-1
    train_env_cfg.reward_config.scales.torques = -1e-3
    train_env_cfg.reward_config.scales.pose = -1.0
    train_env_cfg.reward_config.scales.tracking_ang_vel = 1.25
    train_env_cfg.reward_config.scales.tracking_lin_vel = 1.25
    train_env_cfg.reward_config.scales.feet_phase = 1.0
    train_env_cfg.reward_config.scales.ang_vel_xy = -0.3
    train_env_cfg.reward_config.scales.orientation = -5.0
```

FastTD3에 알맞도록 일부 설정값을 수정

main · mujoco_playground / mujoco_playground / _src / locomotion / g1 / joystick.py

** Mujoco Playground 환경의 baseline이 PPO

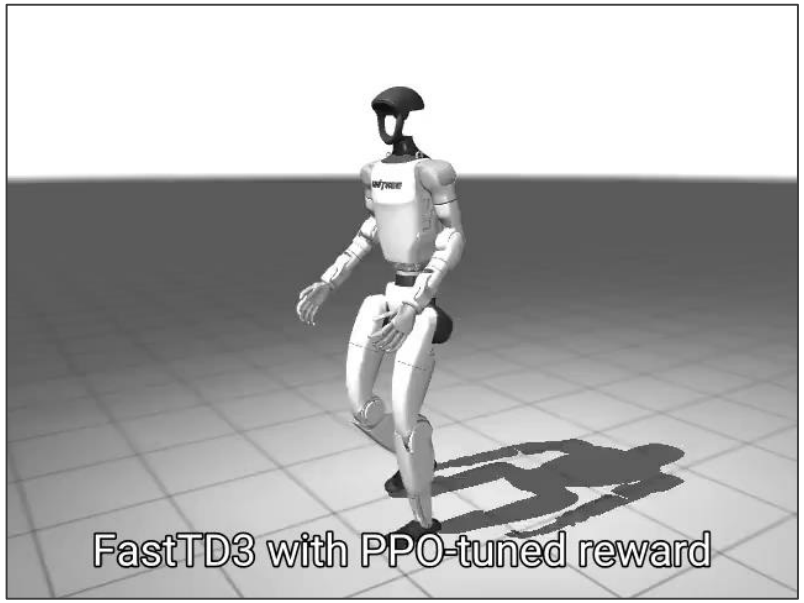
Code | Blame | 792 lines (710 loc) · 26.1 KB

```
34    def default_config() -> config_dict.ConfigDict:
53          ),
54          reward_config=config_dict.create(
55              scales=config_dict.create(
56                  # Tracking related rewards.
57                  tracking_lin_vel=1.0,
58                  tracking_ang_vel=0.75,
59                  # Base related rewards.
60                  lin_vel_z=0.0,
61                  ang_vel_xy=-0.15,
62                  orientation=-2.0,
63                  base_height=0.0,
64                  # Energy related rewards.
65                  torques=0.0,
66                  action_rate=0.0,
67                  energy=0.0,
68                  dof_acc=0.0,
69                  # Feet related rewards.
70                  feet_clearance=0.0,
71                  feet_air_time=2.0,
72                  feet_slip=-0.25,
73                  feet_height=0.0,
74                  feet_phase=1.0,
75                  # Other rewards.
76                  alive=0.0,
77                  stand_still=-1.0,
78                  termination=-100.0,
79                  collision=-0.1,
80                  contact_force=-0.01,
81                  # Pose related rewards.
82                  joint_deviation_knee=-0.1,
83                  joint_deviation_hip=-0.25,
84                  dof_pos_limits=-1.0,
85                  pose=-0.1,
86              ),
87              tracking_sigma=0.25,
88              max_foot_height=0.15,
89              base_height_target=0.5,
90              max_contact_force=500.0,
```

https://github.com/younggyoseo/FastTD3/blob/main/fast_td3/environments/mujoco_playground_env.py
https://github.com/google-deepmind/mujoco_playground/blob/main/mujoco_playground/_src/locomotion/g1/joystick.py

FastTD3

❖ **Experiment**



https://younggyo.me/fast_td3/