Deep Reinforcement Learning

Lecture 13 - Advanced RL Applications

國立清華大學
NATIONAL TSING HUA UNIVERSITY

National Tsing Hua University
Department of Computer Science

Prof. Chun-Yi Lee

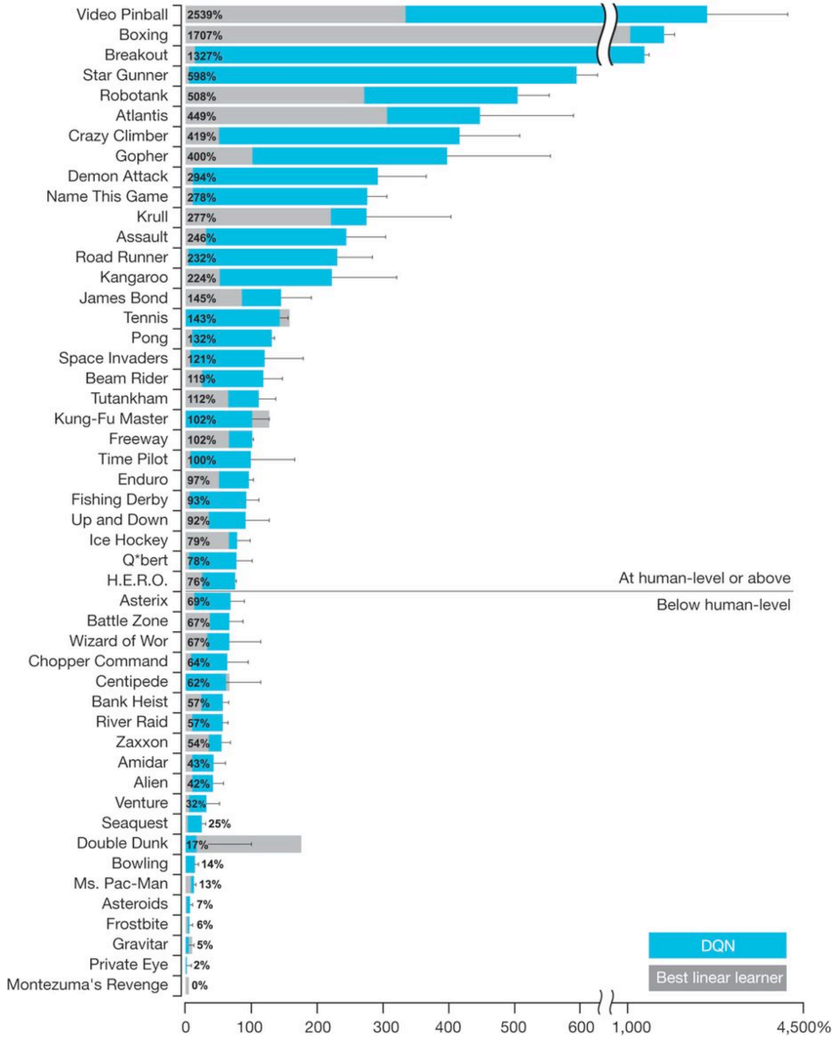
Outline

- Diversity-Driven Exploration
- Flow-Based Intrinsic Curiosity Module
- Adversarial Active Exploration for Inverse Dynamics Model Learning
- Macro Actions for Deep Reinforcement Learning
- Mixture of Step Returns in Bootstrapped DQN
- Efficient Inference Technique

DRL is Extremely Data-Inefficient

- For example, in the best case, DQN consumes **200M** frames to achieve the human-level performance!

- Moreover, DQN mostly fails to human even consuming **> 200M** frames.



If you are a Supervise Learning Practitioner

- You might wonder that why supervised learning (SL) can train DNN efficiently on several dataset (e.g. MNIST, CIFAR-10), but DRL cannot do that

It is because the ”training data”!

Bad training data contribute nothing on RL

Just like unclean training data undermines SL

Training Data for DRL

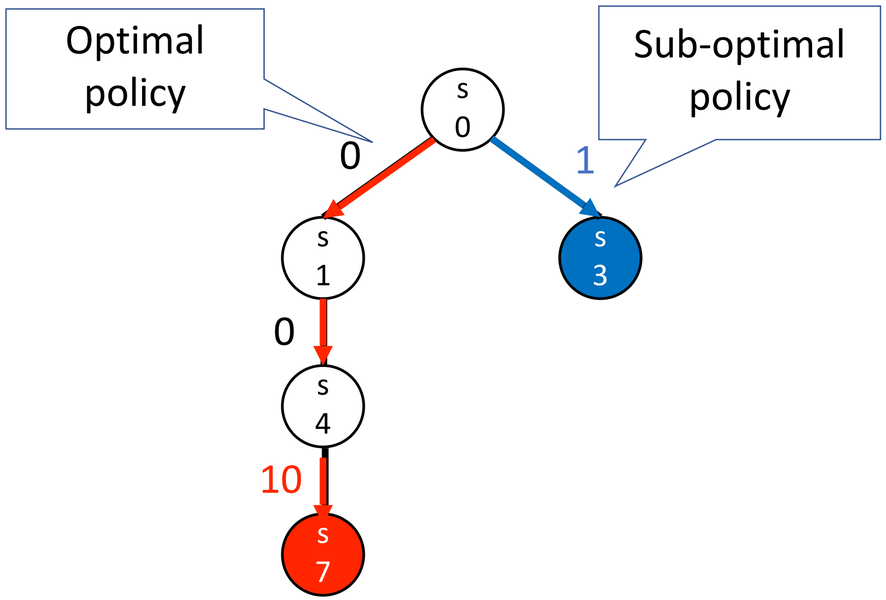
- Recap: the objective function of RL

Usually filled by the agent itself. Filling by human takes too much time.

$$J(.) = \mathbb{E}_{s_t, a_t, s_{t+1} \sim \mathcal{Z}}[.]$$

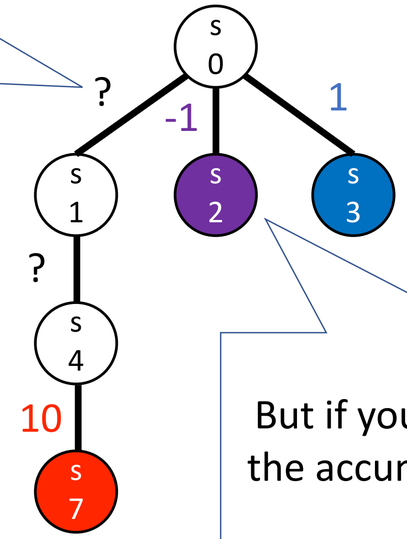
$$J(.) = \mathbb{E}_{s \sim d^{\pi}\theta, a \sim \pi_{\theta}}[.]$$

Sub-Optimal Policy



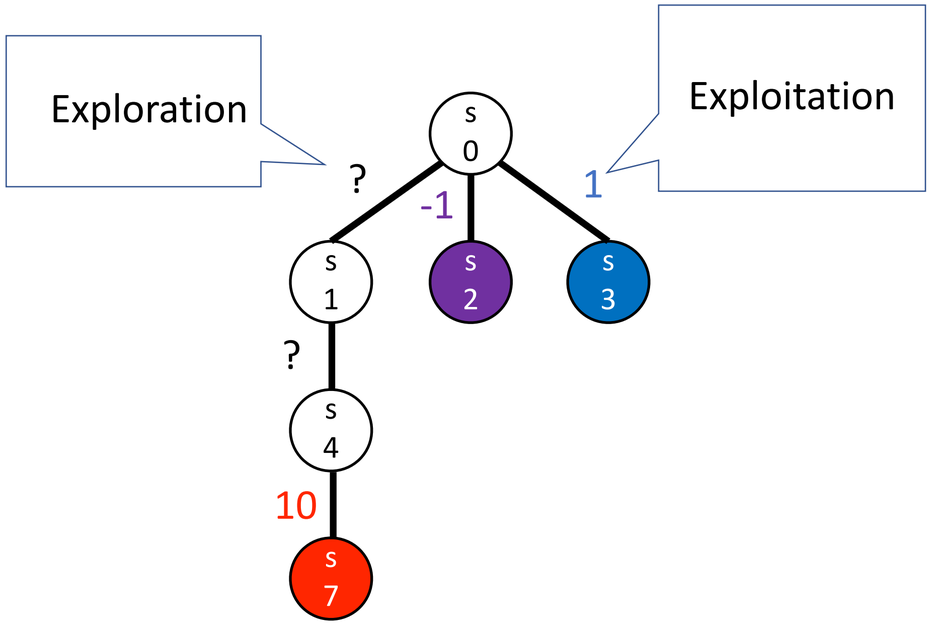
The Cause of Sub-Optimal Policy

If the agent don’t try the unfamiliar action, it will not find the higher reward



But if you keep trying all actions, the accumulated rewards will not be maximized

Exploitation and Exploration

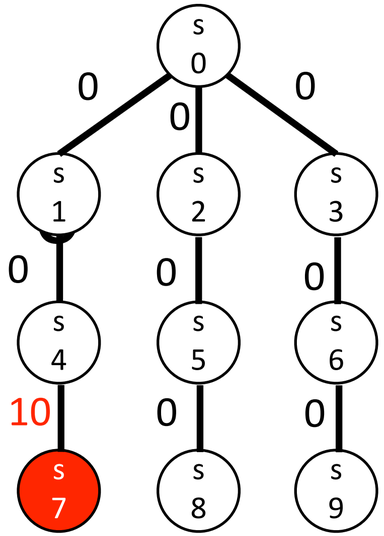


Balancing the exploitation and exploration is necessary for RL, otherwise no useful training data got

Sparse Rewards and Deceptive Rewards

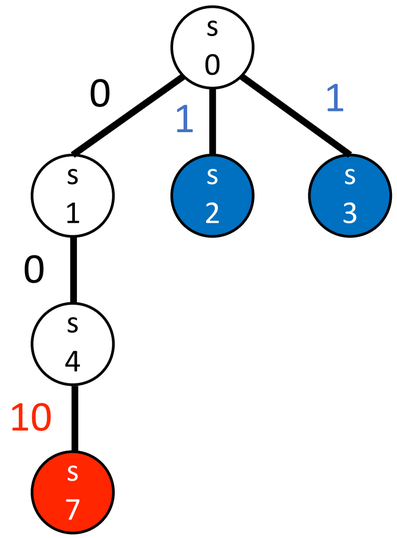
Sparse reward

Difficult to find the reward



Deceptive rewards

Prone to learn a sub-optimal policy



Diversity-Driven Exploration

The modified loss function encourages the agent to **act optimally while differentiating from prior policies**.

Optimality

$$L_D = L - \mathbb{E}_{\pi' \in \Pi}\left[\alpha D(\pi, \pi')\right]$$

"-" in the loss function indicates "maximization"

Diversity

Implementations on DRL

- We implemented the proposed method on three DRL algorithms:
  - DQN
  - DDPG
  - A3C

Implementation on DQN (Div-DQN)

- Recap: the vanilla DQN loss function

$$L(\theta) = \mathbb{E}_{\mathcal{Z}}\left[\left(r(s_t, a_t) + \gamma \max_a Q_{\theta^{Q-}}(s_{t+1}, a) - Q_{\theta^Q}(s_t, a_t)\right)^2\right] \qquad \mathbb{E}_{\mathcal{Z}}[.] = \mathbb{E}_{s_t, a_t, s_{t+1} \sim \mathcal{Z}}[.]$$

- We modify the loss function to:

We apply KL-divergence on soft-Q values

$$L_D(\theta) = \mathbb{E}_{\mathcal{Z}}\left[\left(r(s_t, a_t) + \gamma \max_a Q_{\theta^{Q-}}(s_{t+1}, a) - Q_{\theta^Q}(s_t, a_t)\right)^2 - \alpha D_{KL}(\pi_Q(.|s_t)||\pi_{\hat{Q}}(.|s_t))\right]$$

Diversity loss

$$\mathbb{E}_{\mathcal{Z}}[.] = \mathbb{E}_{s_t, a_t, s_{t+1}, \hat{Q}(s_t,.) \sim \mathcal{Z}}[.]$$

Store the Q-values at each timestep

$$\pi_Q(a|s) = \frac{\exp(Q(s, a))}{\sum_{a' \in \mathcal{A}} \exp(Q(s, a'))}$$

We transform Q-values to a Boltzmann distribution (so that we can use KL-divergence)

Implementation on DDPG (Div-DDPG)

- Recap: the vanilla actor loss function of DDPG

$$L(\theta^{\pi}) = \mathbb{E}_{\mathcal{Z}}\big[- Q_{\theta^Q}(s_t, \pi_{\theta^{\pi}}(s_t))\big] \qquad \mathbb{E}_{\mathcal{Z}}[.] = \mathbb{E}_{s_t \sim \mathcal{Z}}[.]$$

- We modify the loss function to

Compute the diversity with MSE

$$L_D(\theta^{\pi}) = \mathbb{E}_{\mathcal{Z}}\big[- Q_{\theta^Q}(s_t, \pi_{\theta^{\pi}}(s_t)) - \alpha D_{MSE}(\pi_{\theta^{\pi}}(s_t), a_t)\big] \qquad \mathbb{E}_{\mathcal{Z}}[.] = \mathbb{E}_{s_t, a_t \sim \mathcal{Z}}[.]$$

Diversity loss

Store the actions at each timestep

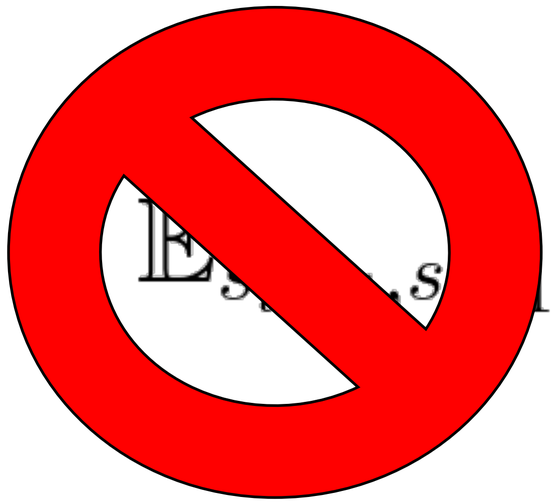
Implementation on A2C (Div-A2C)

- Recap: the vanilla actor loss function of A2C

$$L(\theta^{\pi}) = \mathbb{E}_{\pi_{\theta^{\pi}}}\left[- V_{\theta^{V}}(s_t)\right]$$

- How to compute the diversity loss term for Div-A2C? We cannot use **experience replay**!

$$\mathbb{E}_{\mathcal{Z}}\left[.\right] \quad \mathbb{E}_{s_t, \ldots, s_{t+1} \sim \mathcal{Z}}\left[.\right]$$



Implementation on A2C (Div-A2C)

- First, we store the past policies:

$$\Pi = \{\theta^{\pi i} : i \in \mathbb{N}, i \leq N_{prior}\}$$

We only store the recent policies parameters

- Then, we compute the diversity los

$$L_D(\theta^\pi) = \mathbb{E}_{\pi_{\theta^\pi}}\left[-V_{\theta^V}(s_t) - \mathbb{E}_{\theta^{\pi'} \in \Pi}\left[\alpha_{\pi'} D_{KL}\left(\pi_{\theta^\pi}(.|s_t) || \pi_{\theta^{\pi'}}(.|s_t)\right)\right]\right]$$

Diversity loss

Applying KL-divergence for A2C is straightforward as its policy is represented as a probability distribution

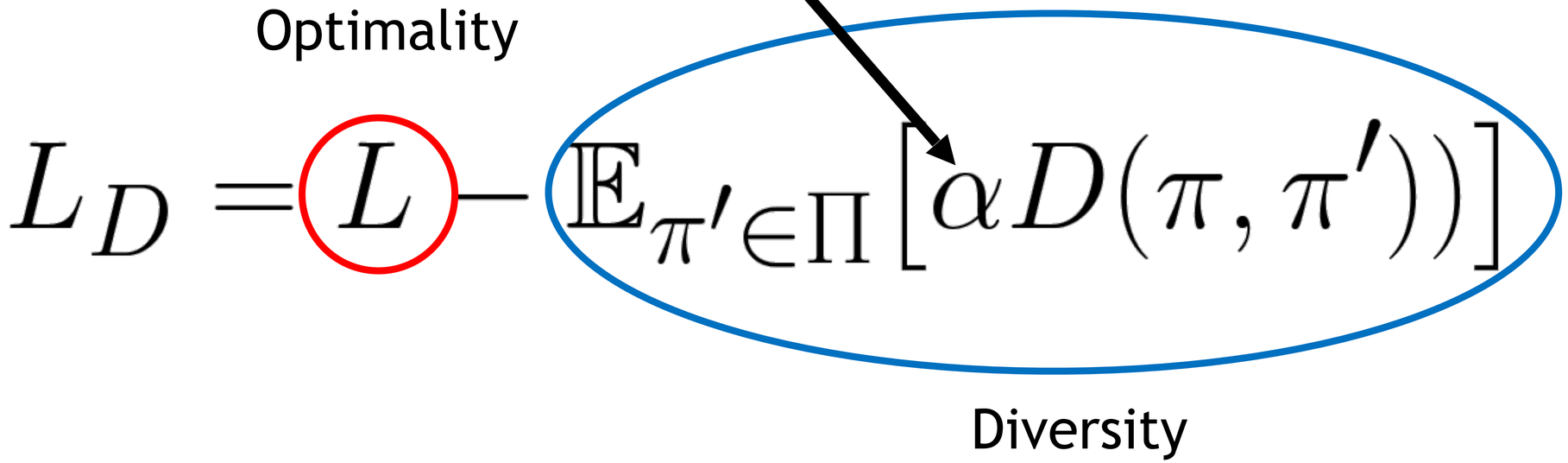
$$\mathbb{E}_{\pi_{\theta^\pi}}\left[.\right] = \mathbb{E}_{s \sim d^{\pi_{\theta^\pi}} a \sim \pi_{\theta^\pi}}\left[.\right]$$

The scaling factor varies different past policies, which will be discussed later.

Does it diverge?

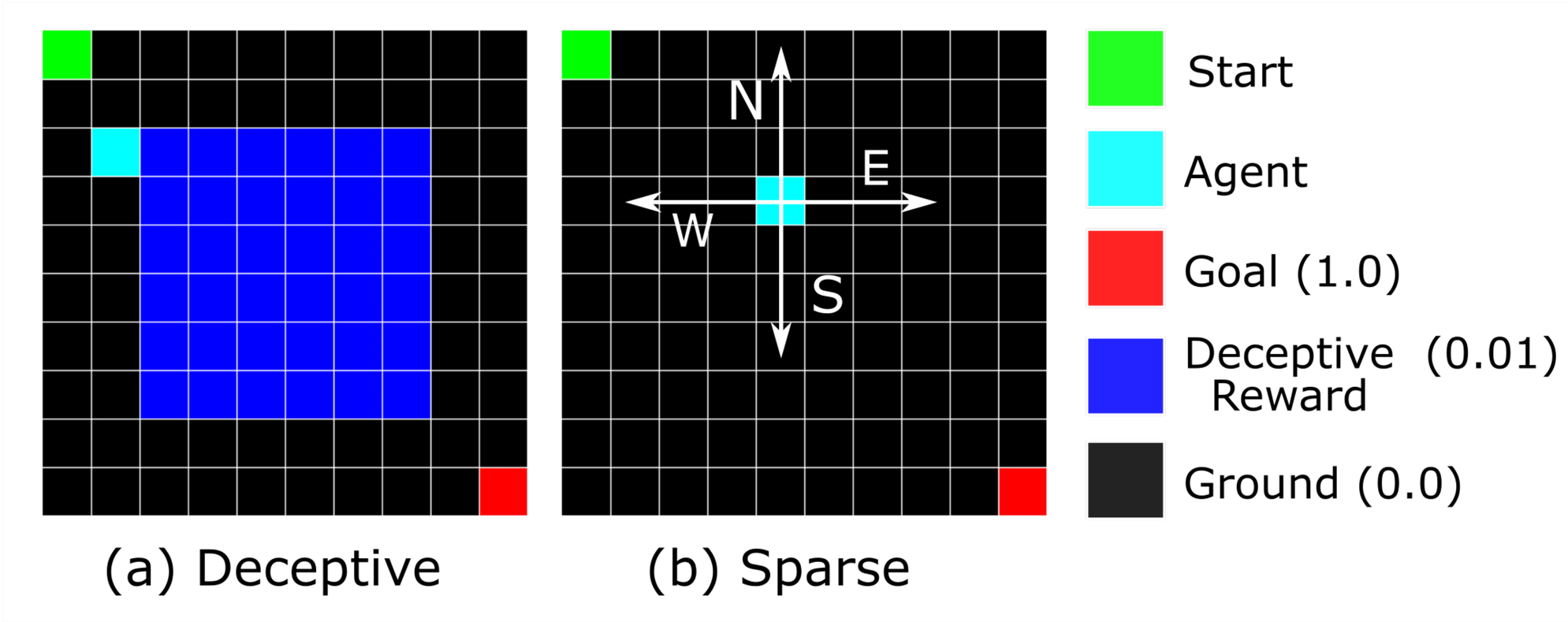
The scaling factor can prevent diverging

Adaptively adjust it!



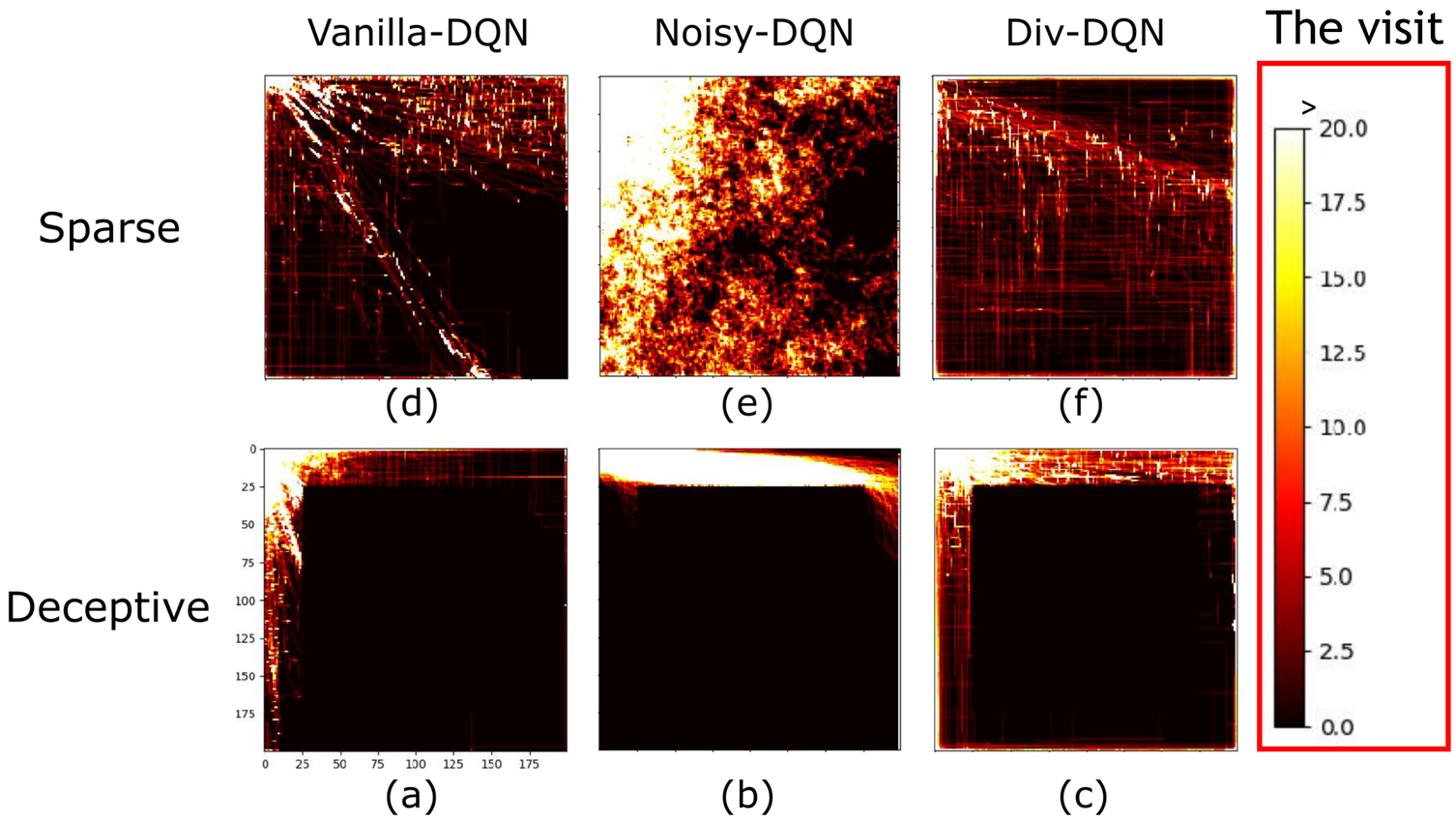
Huge gridworld

We demonstrate the effectiveness of our method in the environments with deceptive and sparse rewards

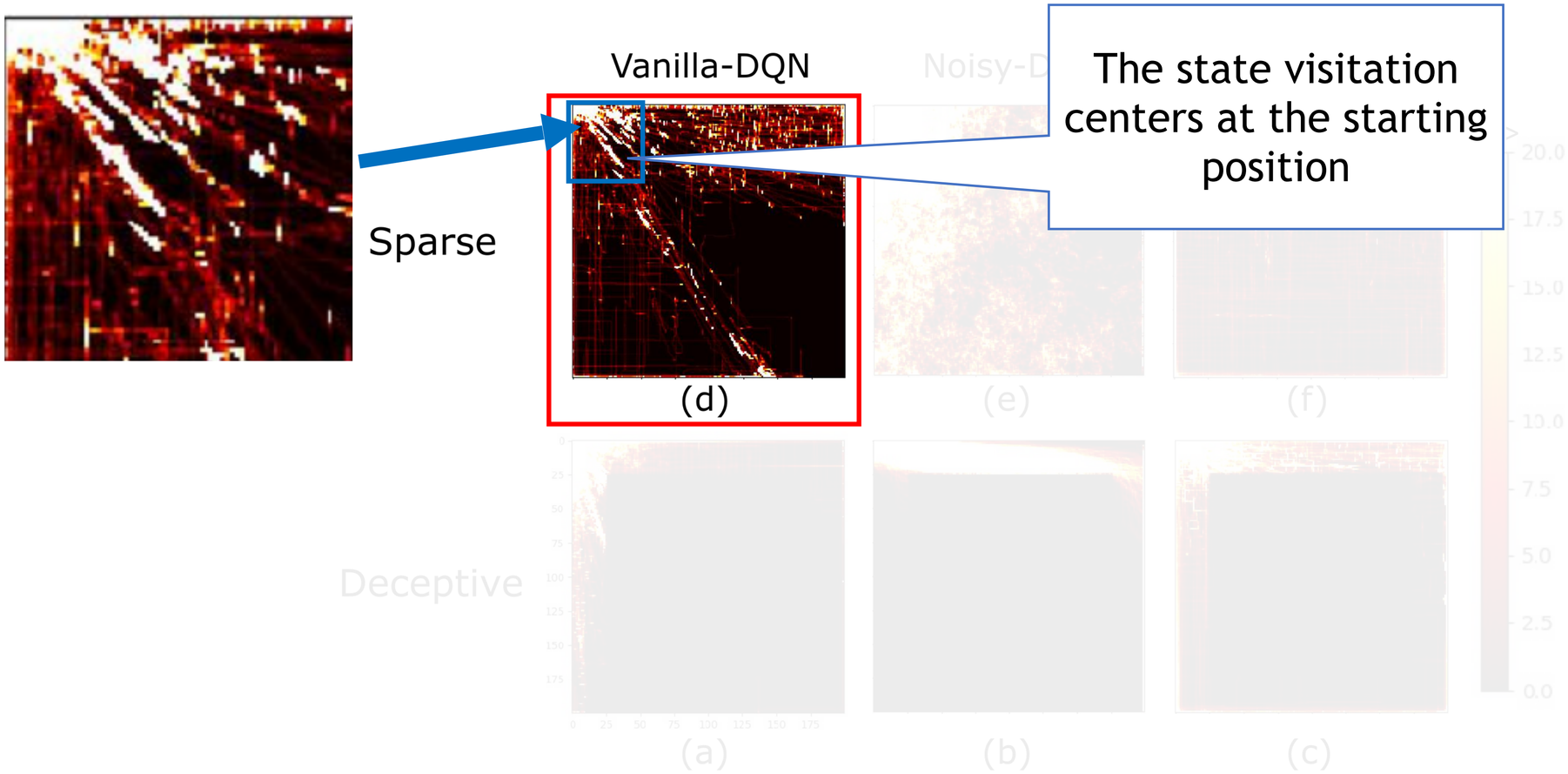


Qualitive results

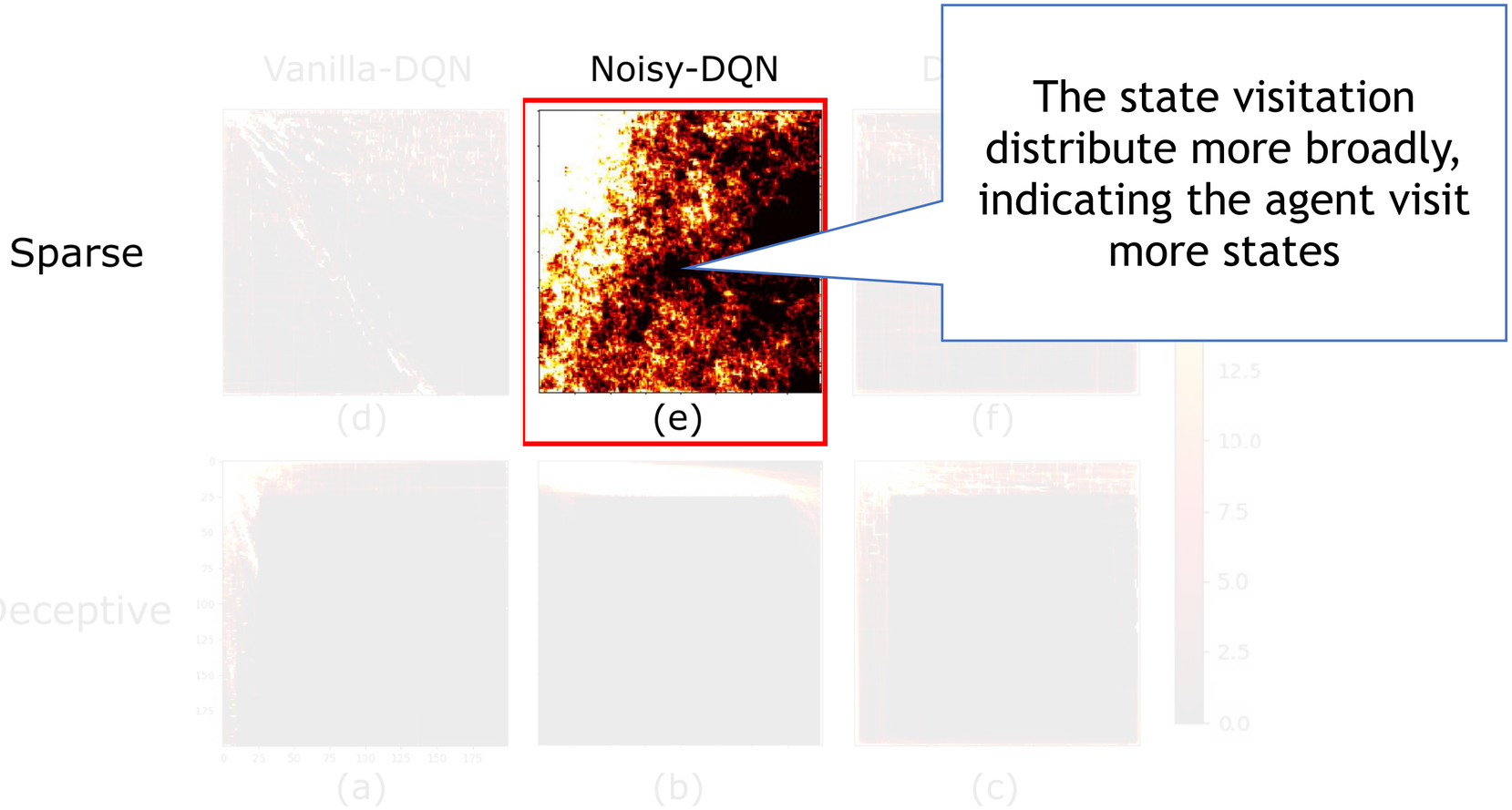
This figure is the state visitation of each algorithm in the environment of which size is 200x200



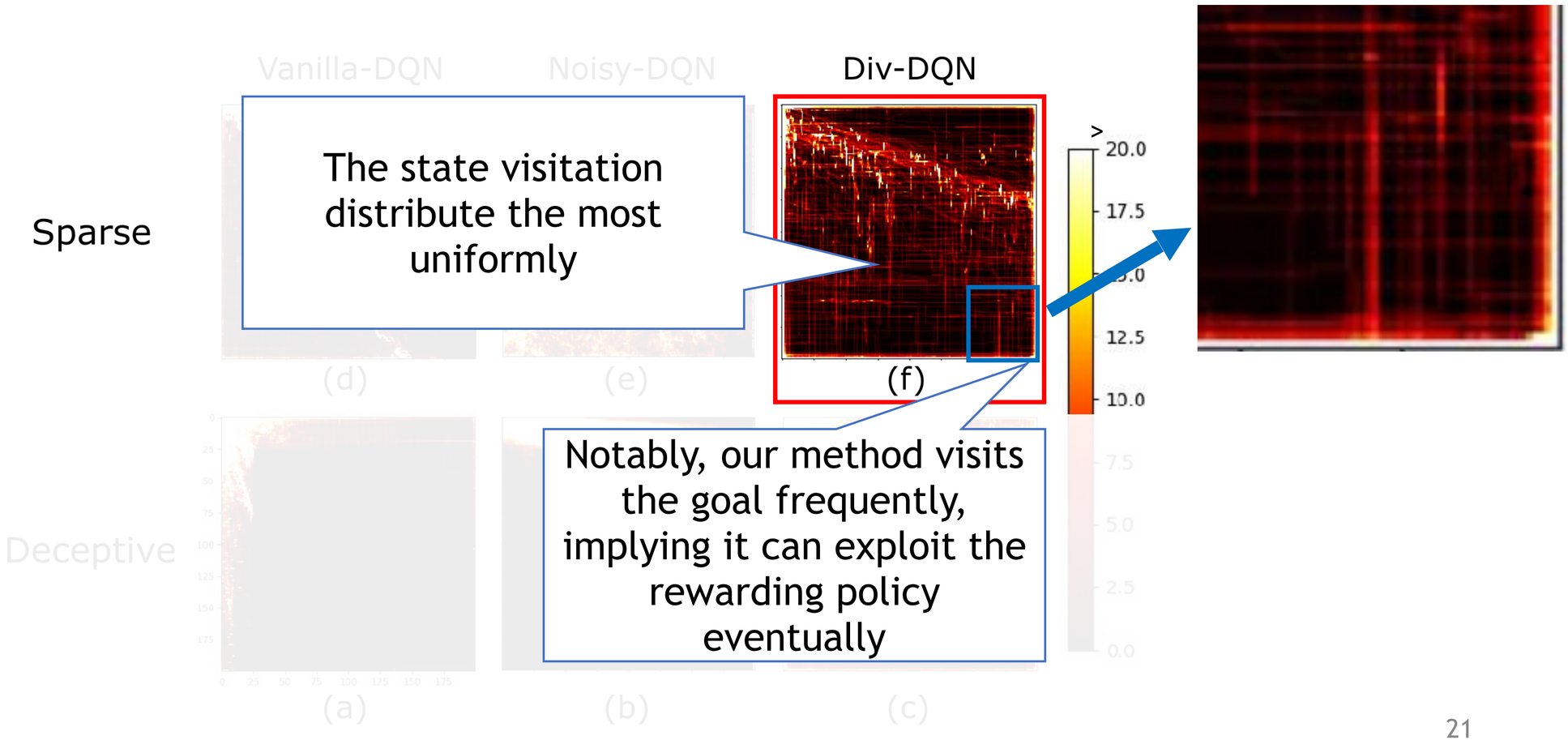
Qualitive results - Sparse



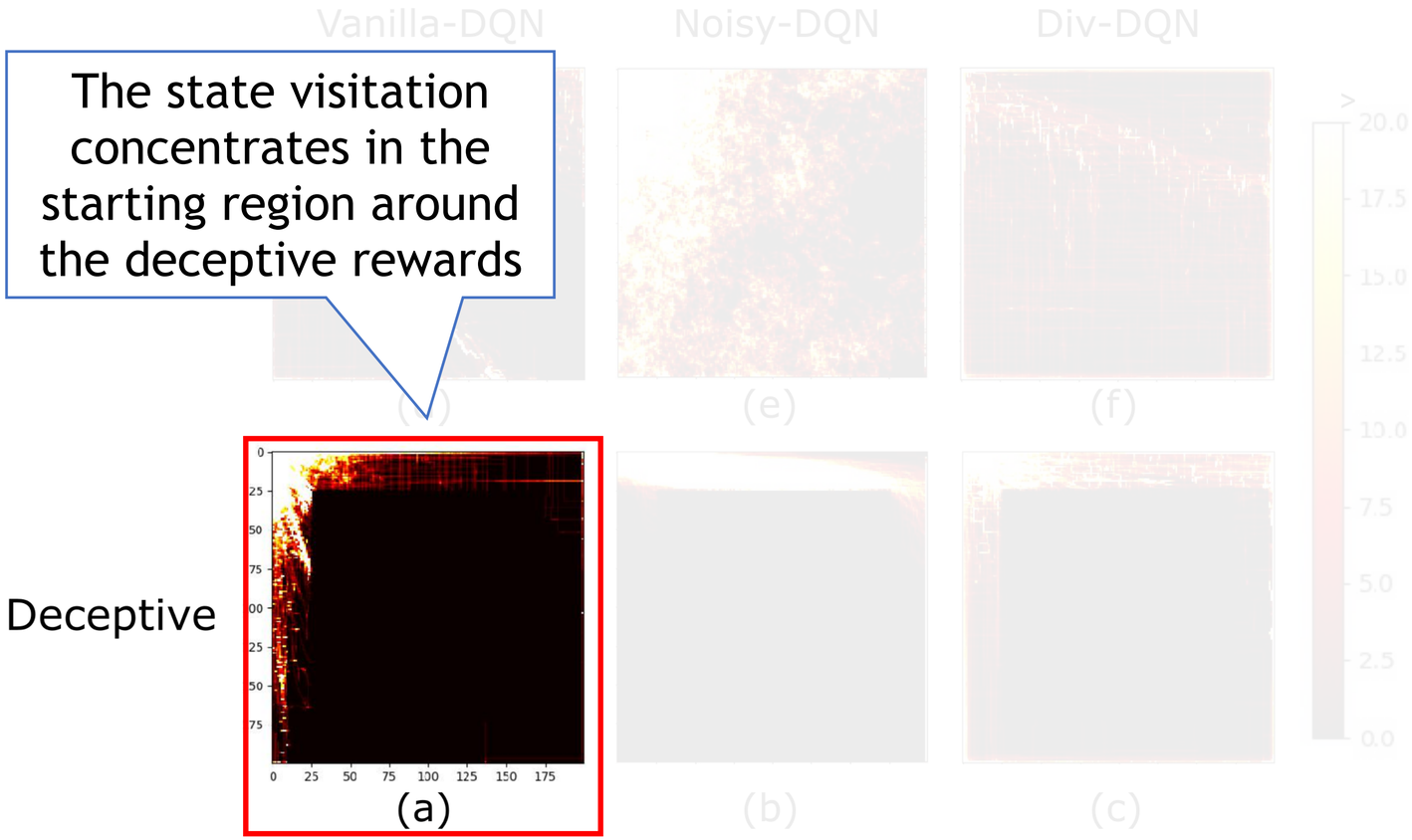
Qualitive results - Sparse



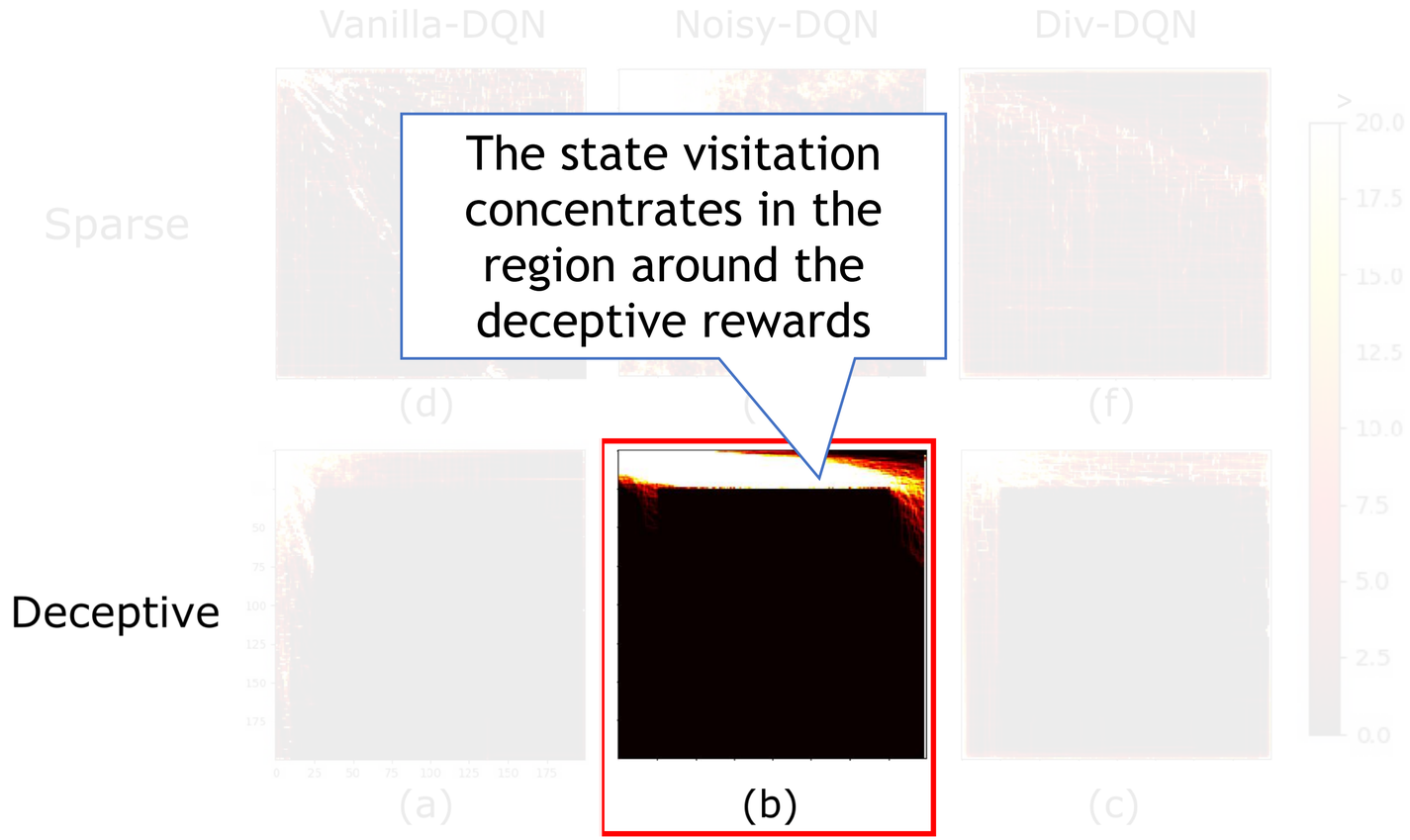
Qualitive results - Sparse



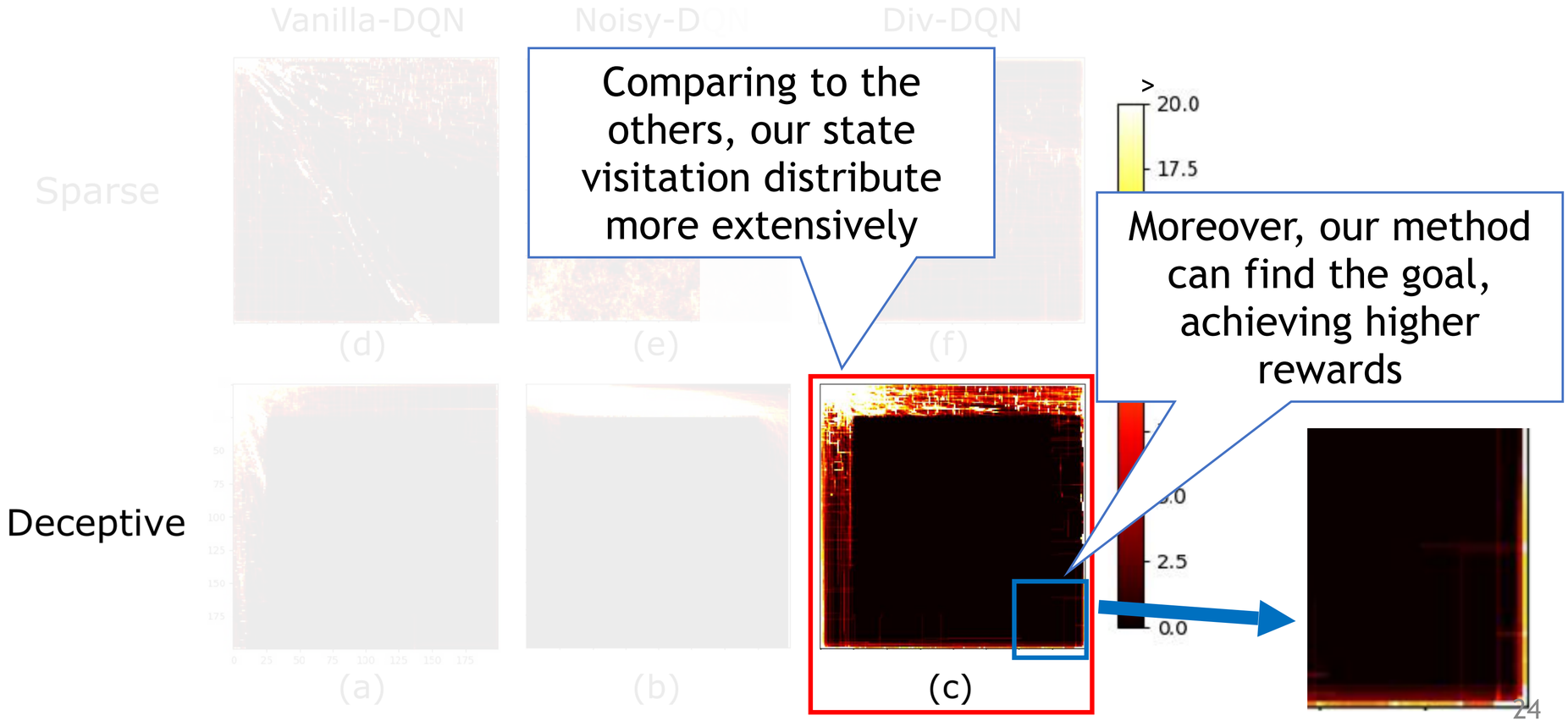
Qualitive results - Deceptive



Qualitive results - Deceptive



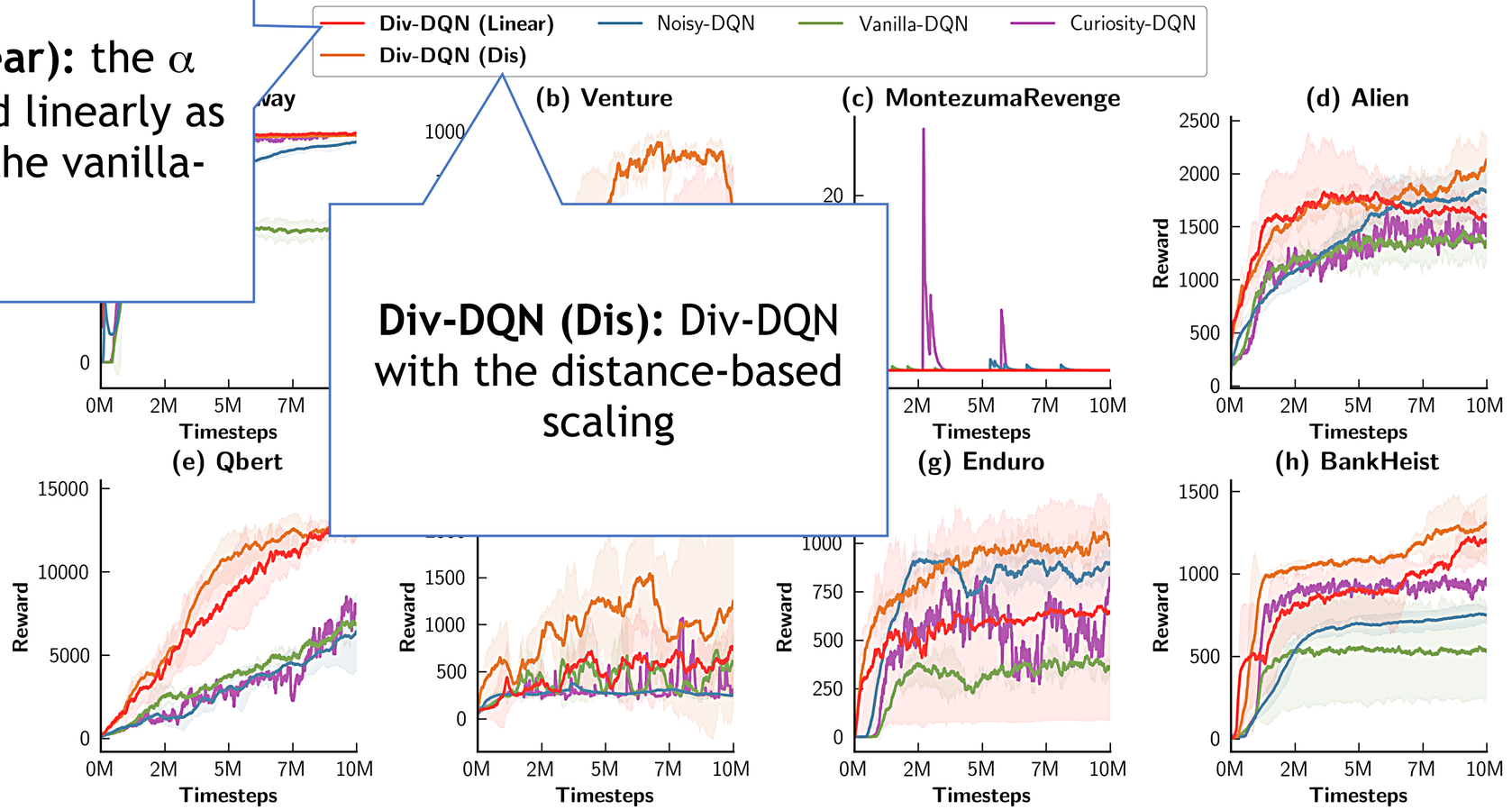
Qualitive results - Deceptive



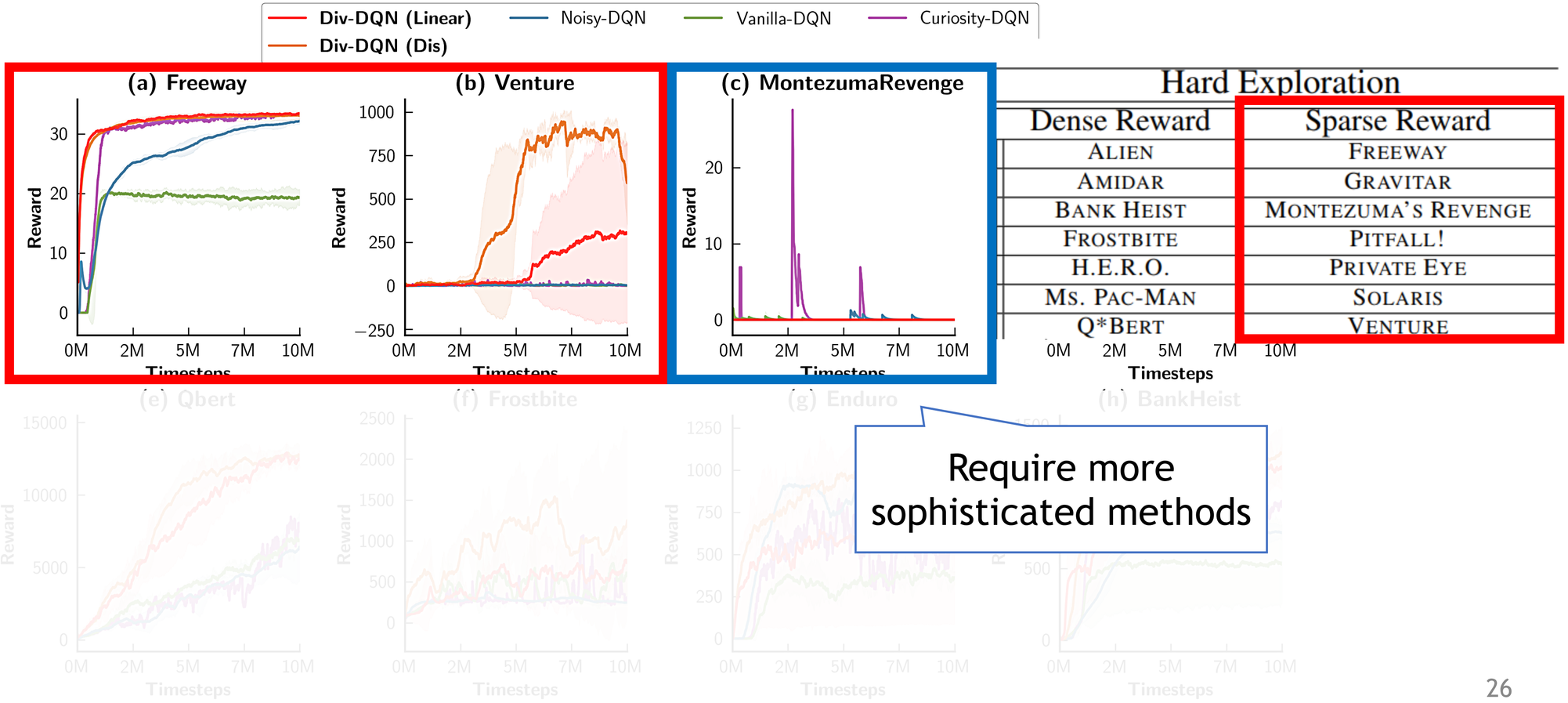
Quantitative results of Div-DQN

Div-DQN (Linear): the α value is decayed linearly as the ε value in the vanilla-DQN

Div-DQN (Dis): Div-DQN with the distance-based scaling



Quantitative results of Div-DQN – Sparse rewards



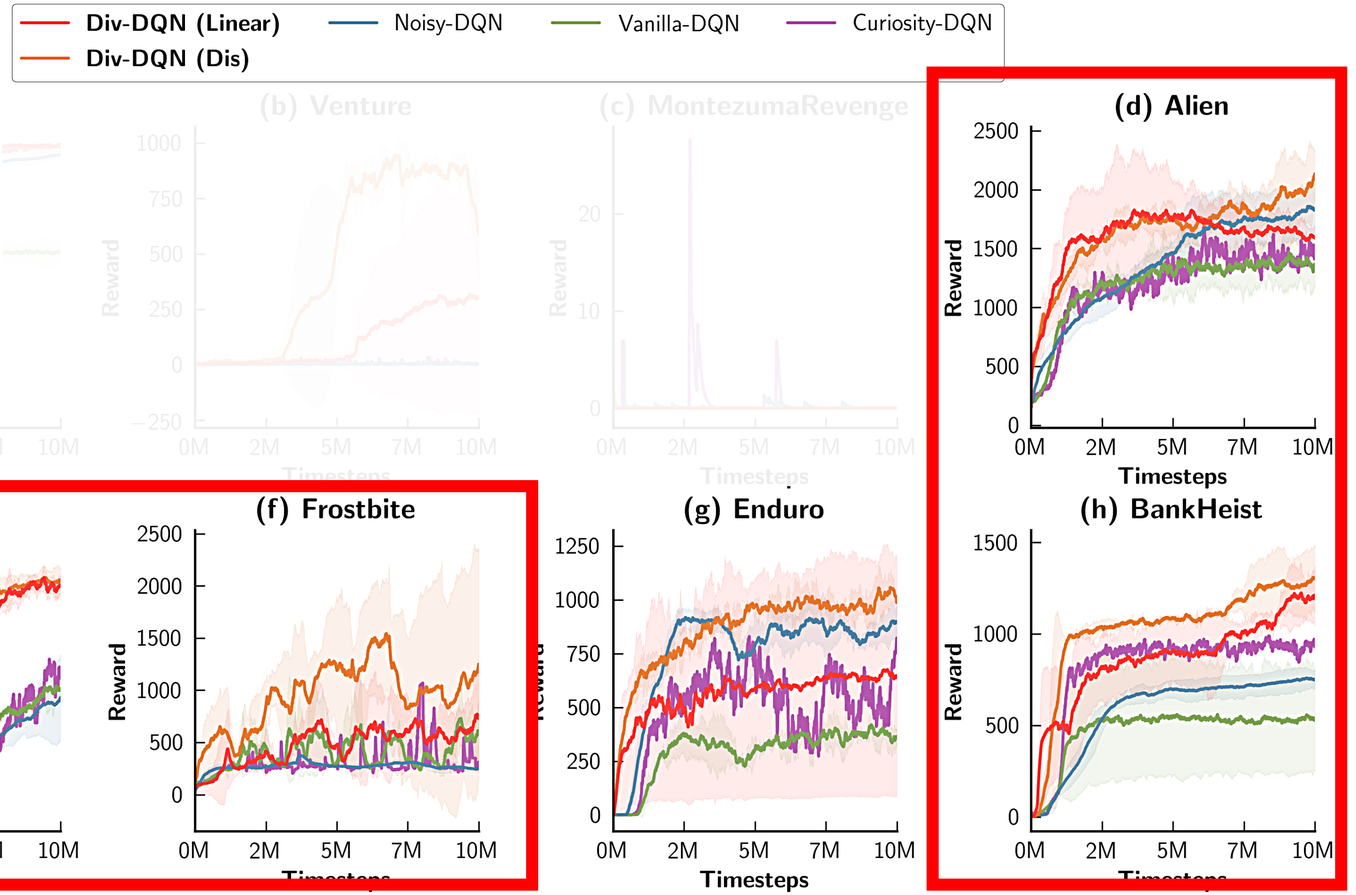
Hard Exploration	
Dense Reward	Sparse Reward
ALIEN	FREEWAY
AMIDAR	GRAVITAR
BANK HEIST	MONTEZUMA'S REVENGE
FROSTBITE	PITFALL!
H.E.R.O.	PRIVATE EYE
MS. PAC-MAN	SOLARIS
Q*BERT	VENTURE

Require more sophisticated methods

Quantitative results of Div-DQN – Deceptive rewards

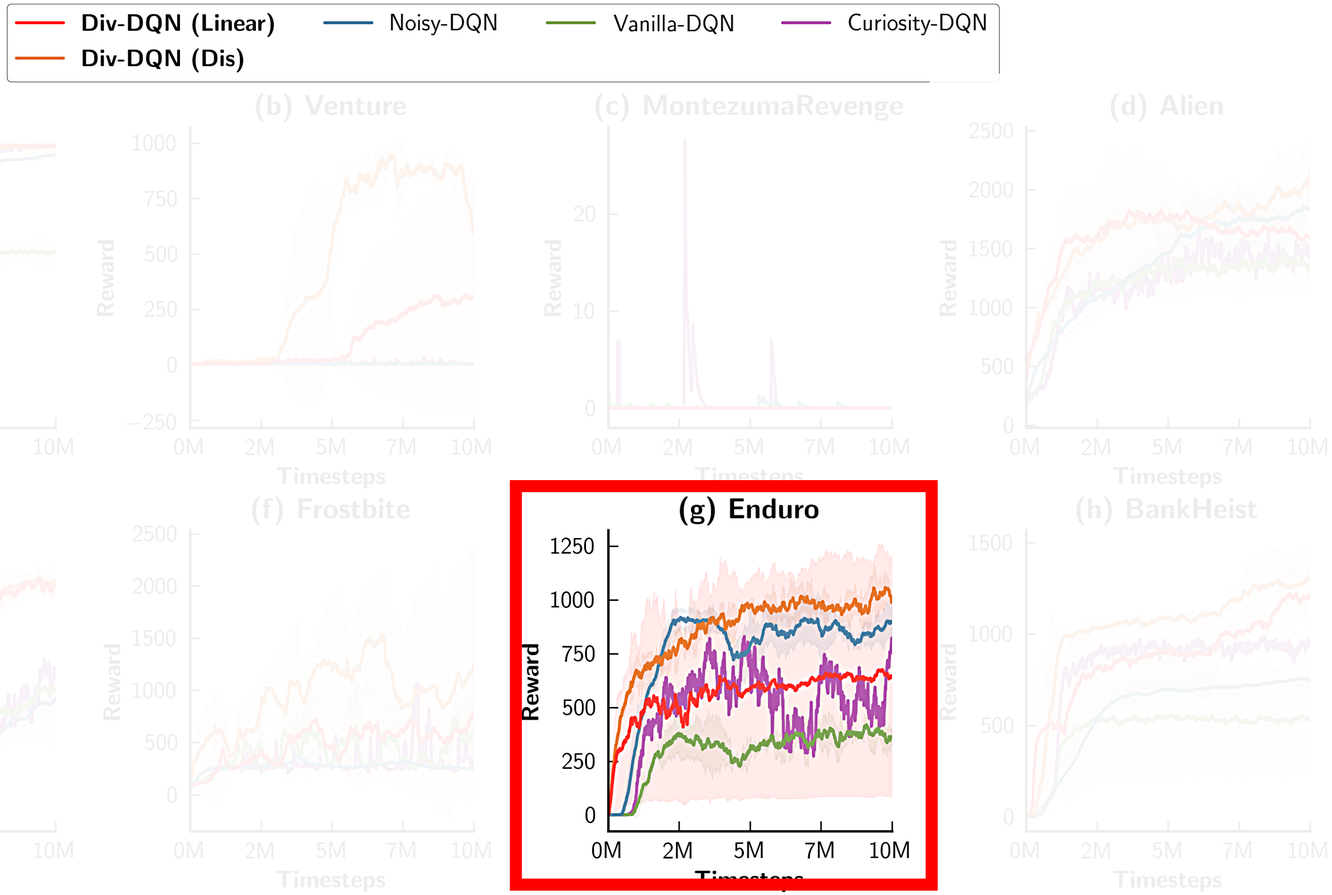
Deceptive rewards

Hard Exploration	
Dense Reward	Sparse Reward
ALIEN	FREEWAY
AMIDAR	GRAVITAR
BANK HEIST	MONTEZUMA'S REVENGE
FROSTBITE	PITFALL!
H.E.R.O.	PRIVATE EYE
MS. PAC-MAN	SOLARIS
Q*BERT	VENTURE

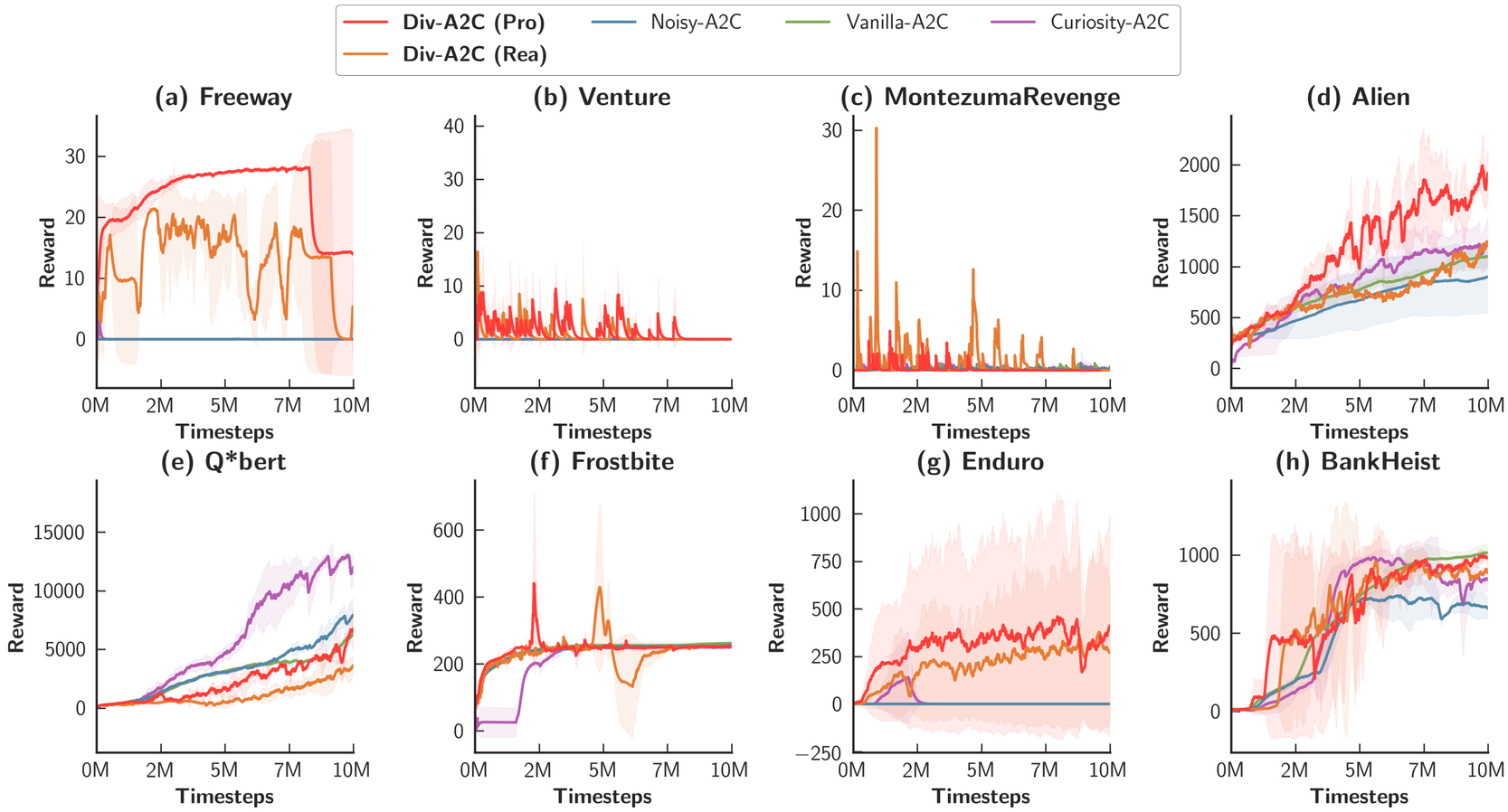


Quantitative results of Div-DQN

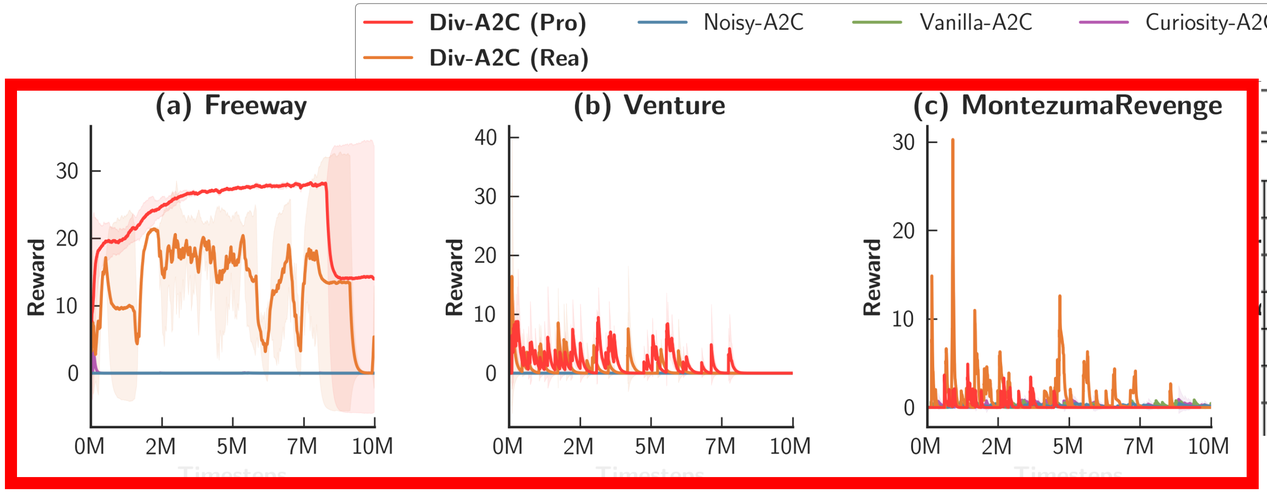
Easy Exploration	
Human-Optimal	
ASSAULT	ASTERIX
ASTEROIDS	ATLANTIS
BATTLE ZONE	BERZERK
BOWLING	BOXING
BREAKOUT	CENTIPEDE
CHOPPER CMD	CRAZY CLIMBER
DEFENDER	DEMON ATTACK
DOUBLE DUNK	ENDURO



Quantitative results of Div-A2C



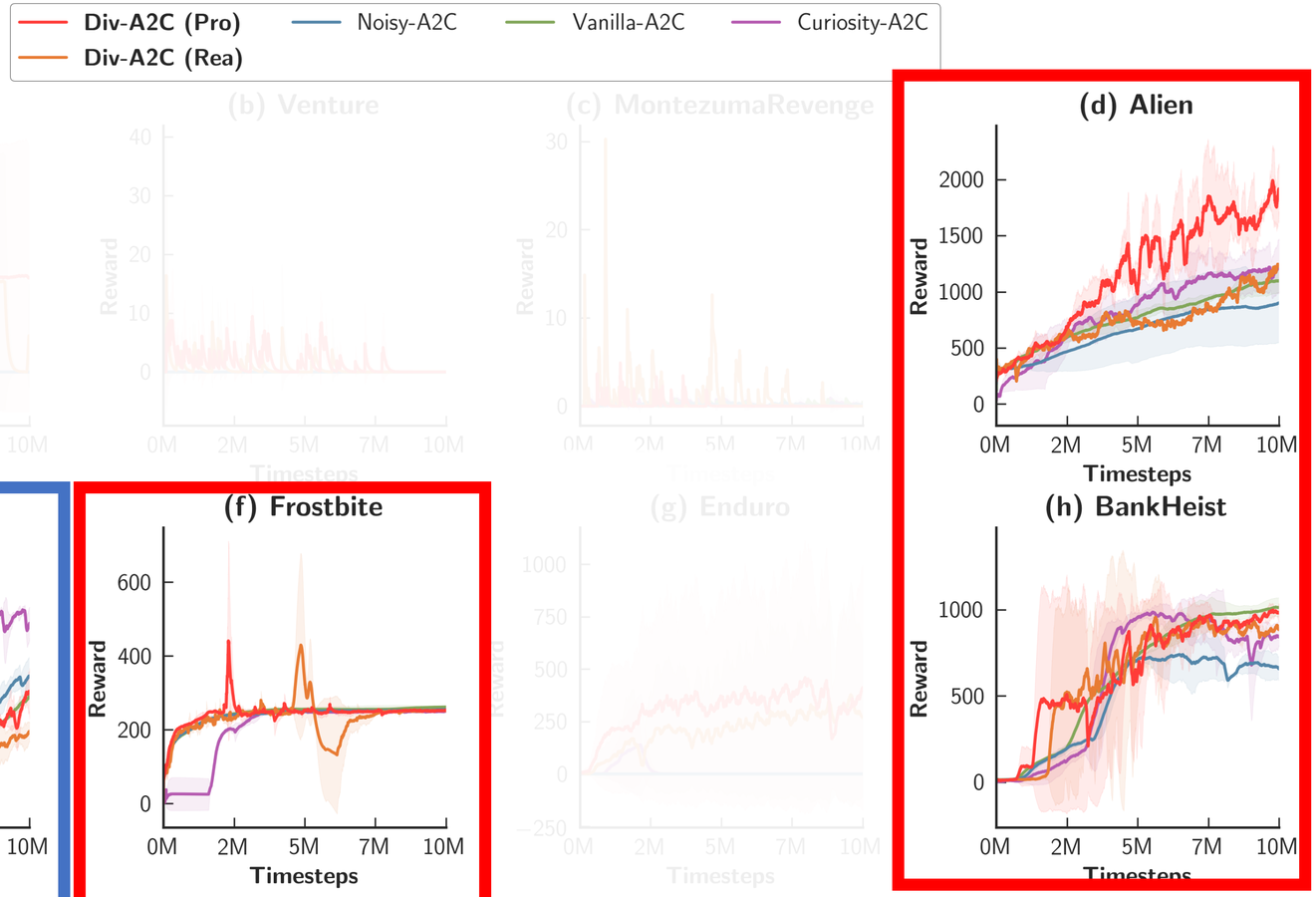
Quantitative results of Div-A2C



Hard Exploration	
Dense Reward	Sparse Reward
ALIEN	FREEWAY
AMIDAR	GRAVITAR
BANK HEIST	MONTEZUMA'S REVENGE
FROSTBITE	PITFALL!
H.E.R.O.	PRIVATE EYE
MS. PAC-MAN	SOLARIS
Q*BERT	VENTURE

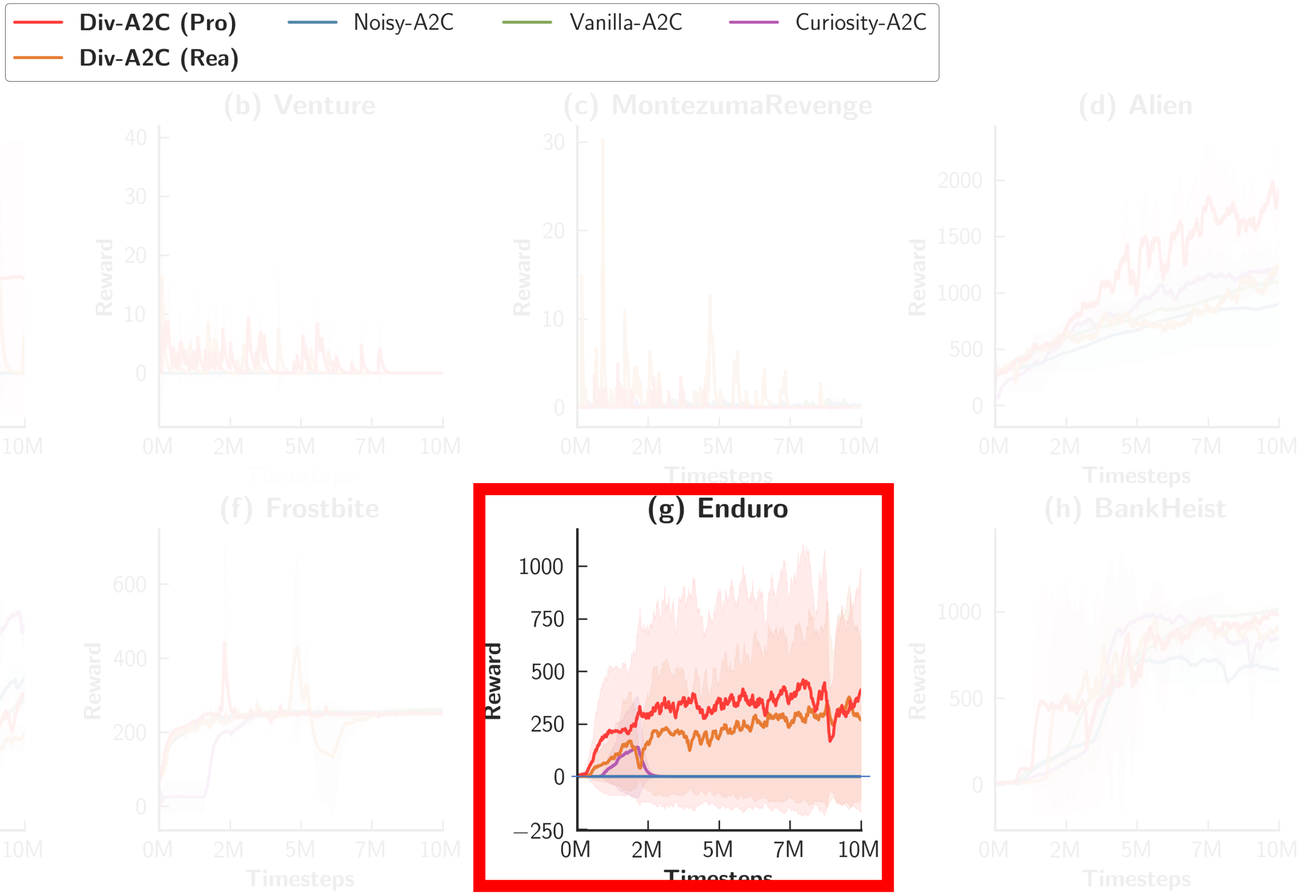
Quantitative results of Div-A2C

Hard Exploration	
Dense Reward	Sparse Reward
ALIEN	FREEWAY
AMIDAR	GRAVITAR
BANK HEIST	MONTEZUMA'S REVENGE
FROSTBITE	PITFALL!
H.E.R.O.	PRIVATE EYE
MS. PAC-MAN	SOLARIS
Q*BERT	VENTURE

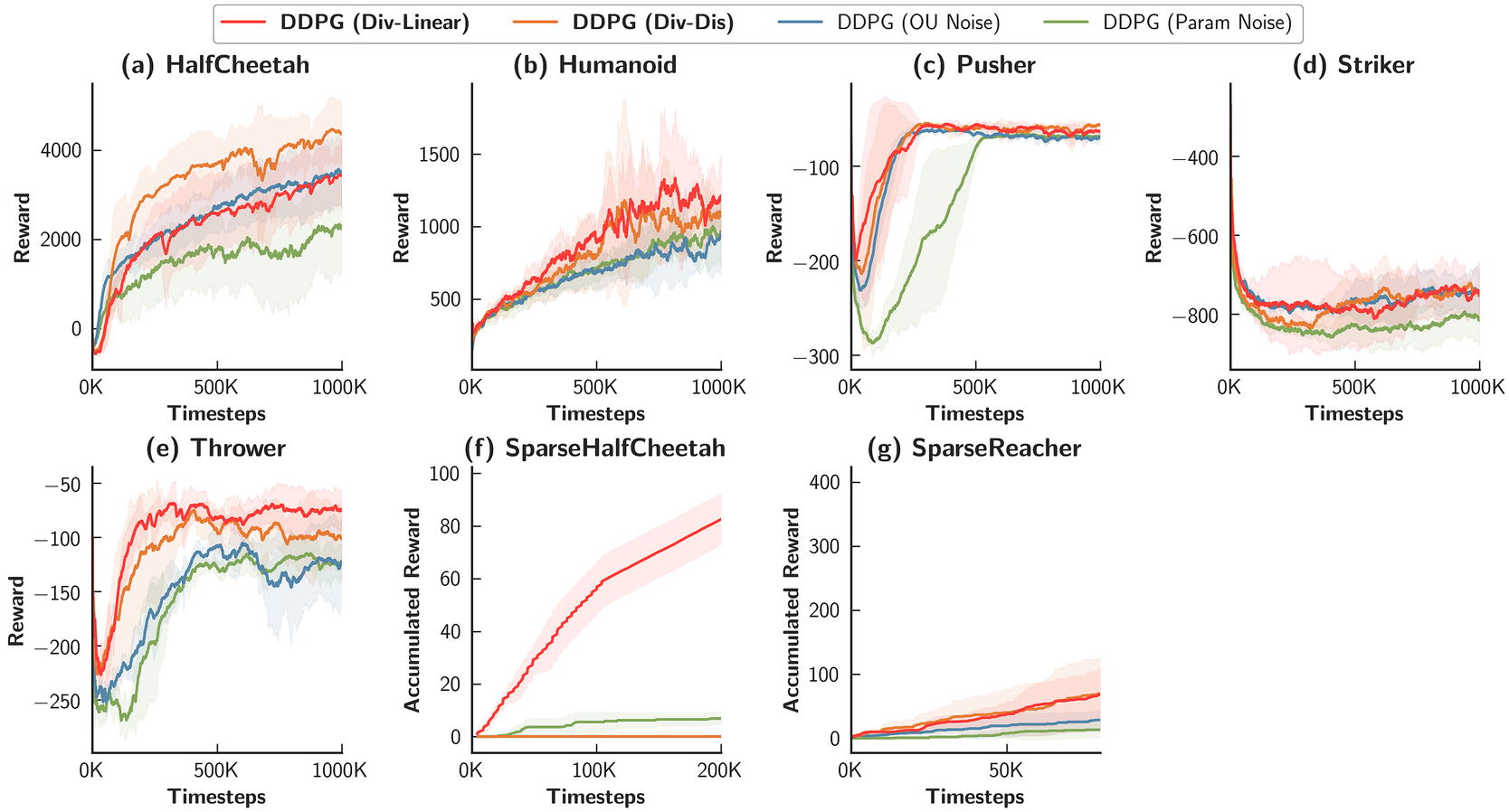


Quantitative results of Div-A2C

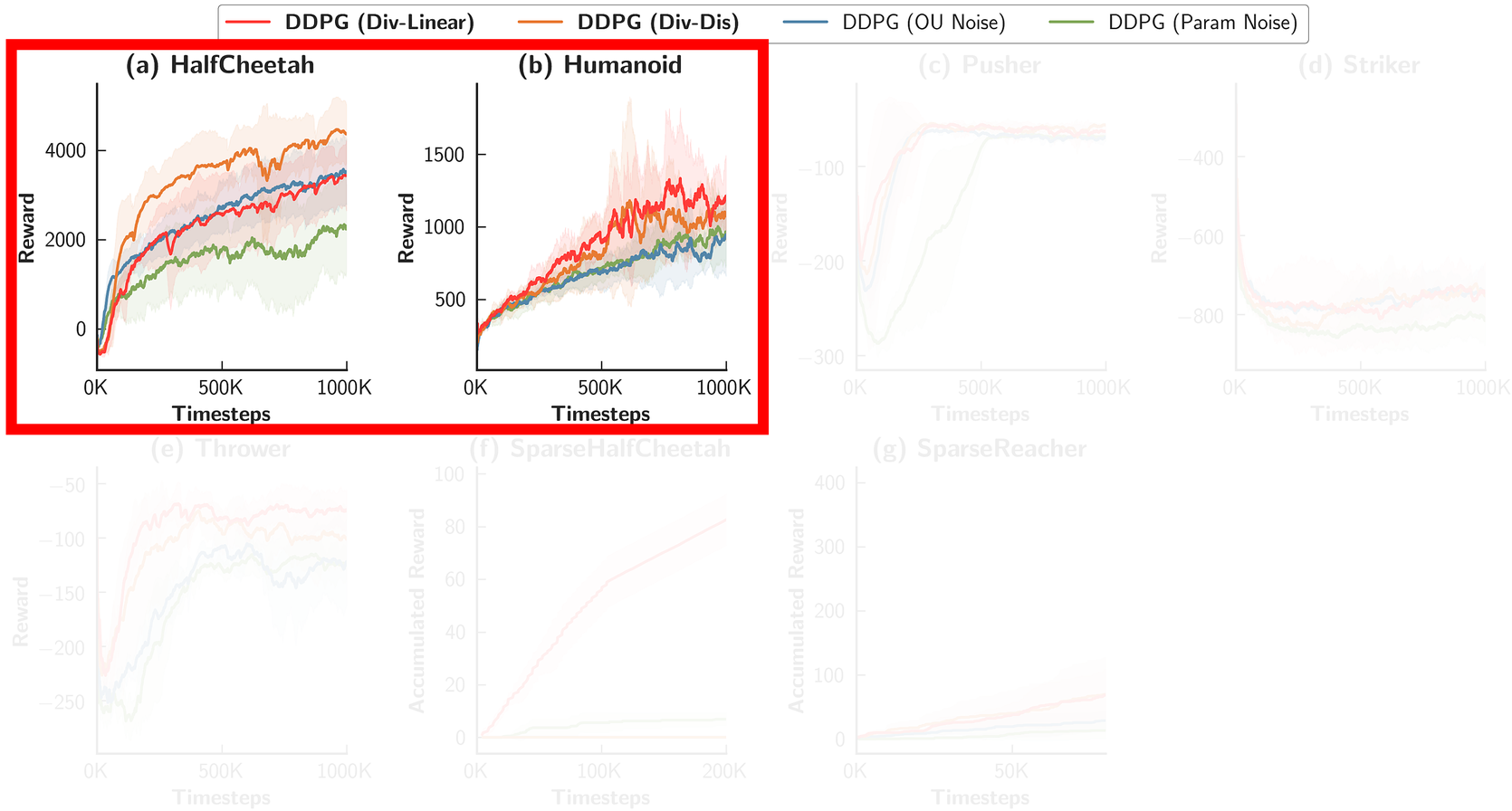
Easy Exploration	
Human-Optimal	
ASSAULT	ASTERIX
ASTEROIDS	ATLANTIS
BATTLE ZONE	BERZERK
BOWLING	BOXING
BREAKOUT	CENTIPEDE
CHOPPER CMD	CRAZY CLIMBER
DEFENDER	DEMON ATTACK
DOUBLE DUNK	ENDURO



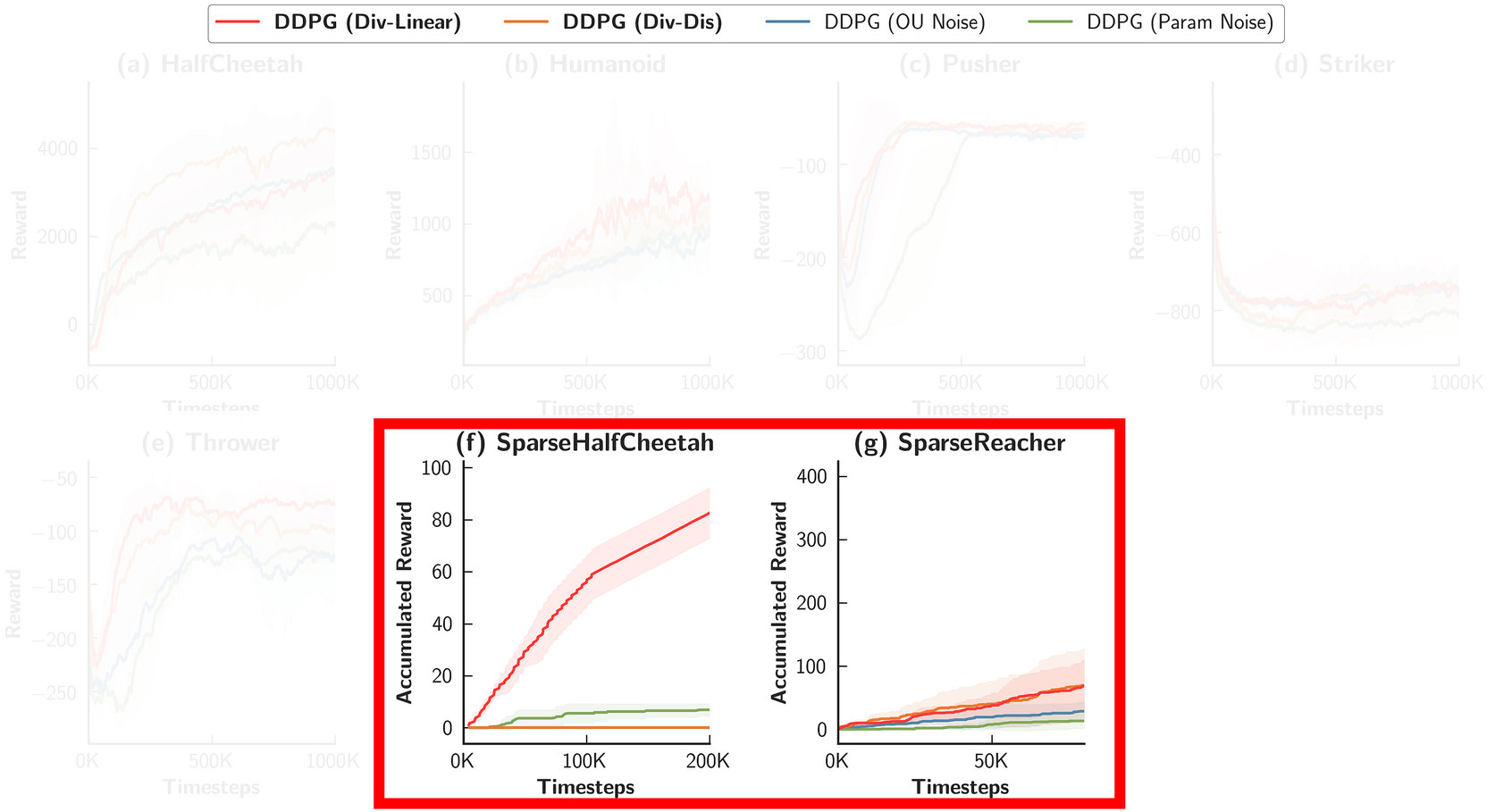
Quantitative results



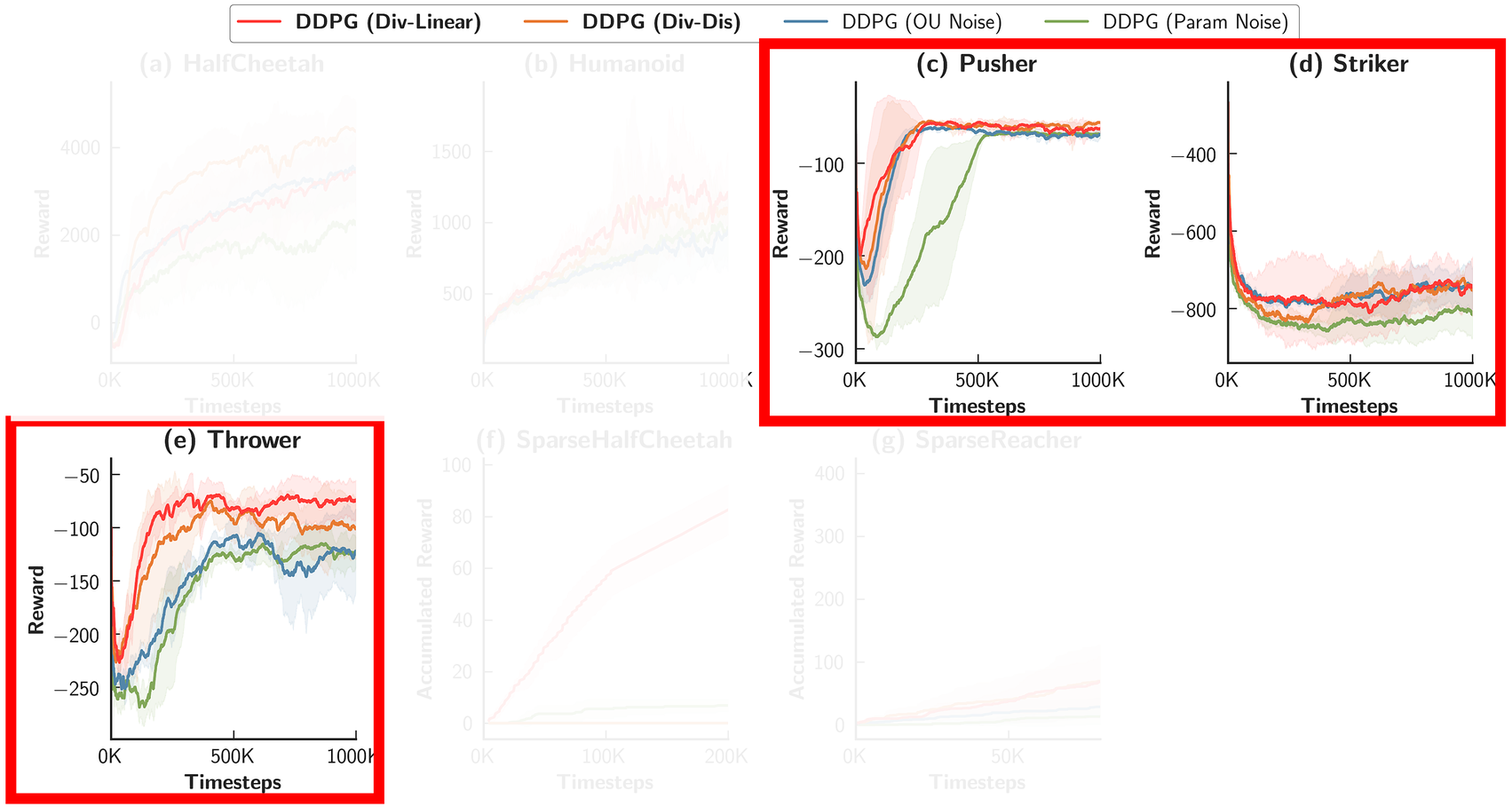
Quantitative results – Deceptive rewards



Quantitative results – Sparse rewards



Quantitative results – Large state space (Dense rewards)

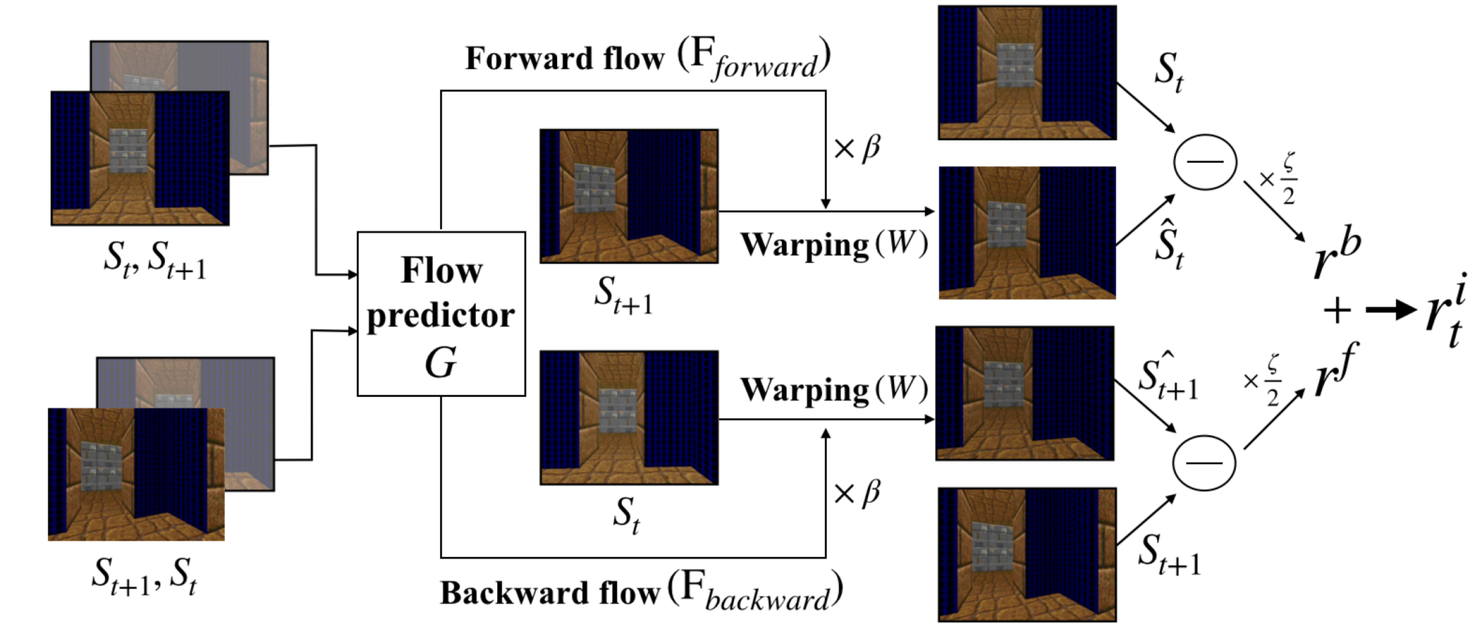


Outline

- Diversity-Driven Exploration
- Flow-Based Intrinsic Curiosity Module
- Adversarial Active Exploration for Inverse Dynamics Model Learning
- Macro Actions for Deep Reinforcement Learning
- Mixture of Step Returns in Bootstrapped DQN
- Efficient Inference Technique

Flow-Based Intrinsic Module

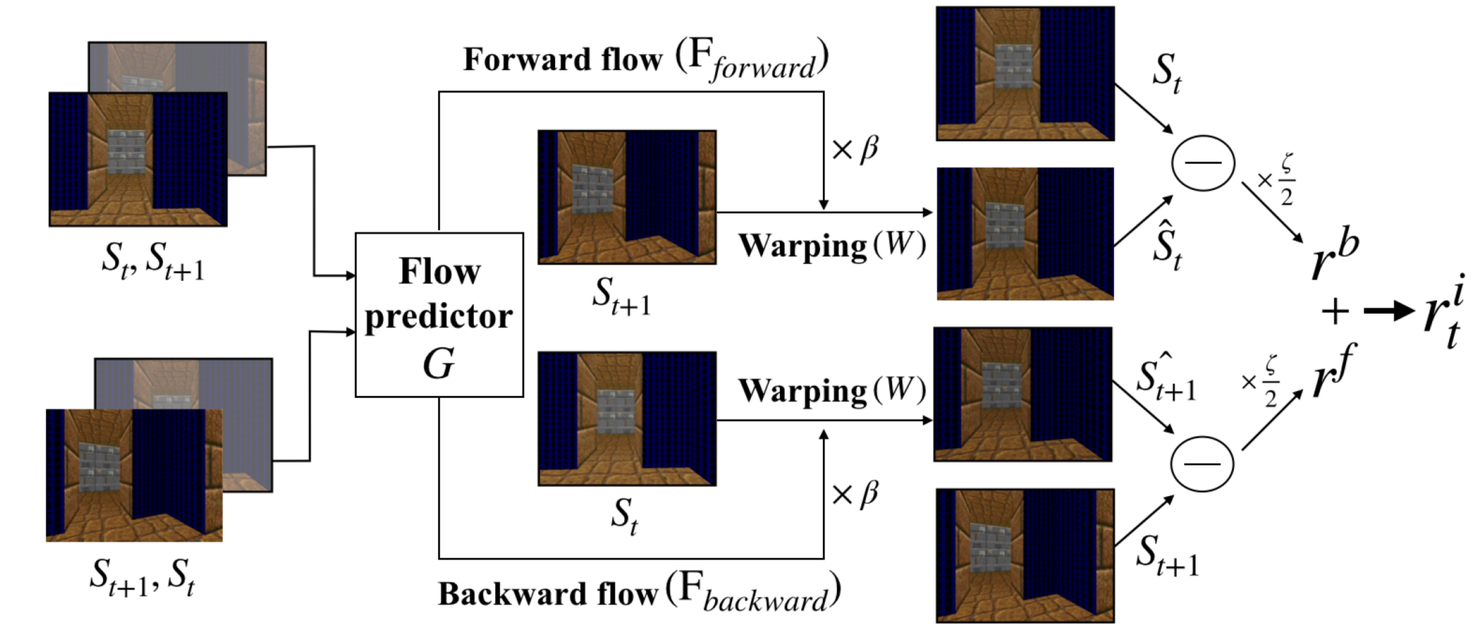
Architecture



- **We introduce flow-based intrinsic module (FICM) to replace ICM**
  - FICM uses optical flow prediction errors as the novelty of states
  - Optical flows are estimated in dual directions: Forward and backward
  - The differences between the warped frames and the actual frames serve as the intrinsic reward

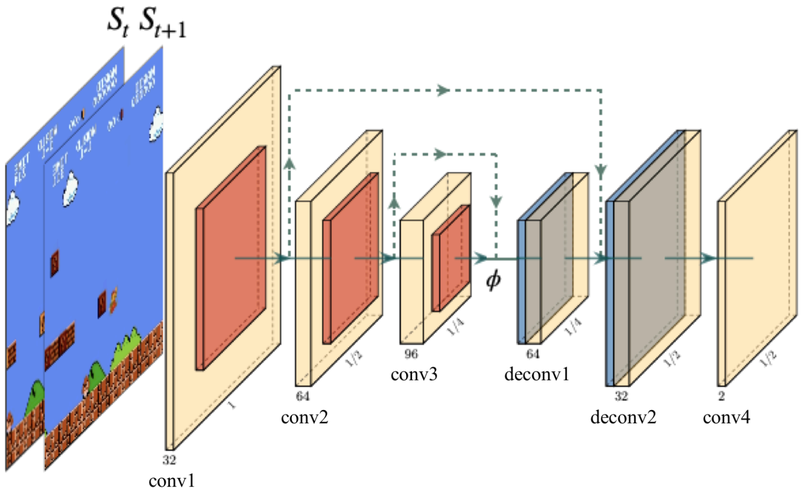
Flow-Based Intrinsic Module

Advantages

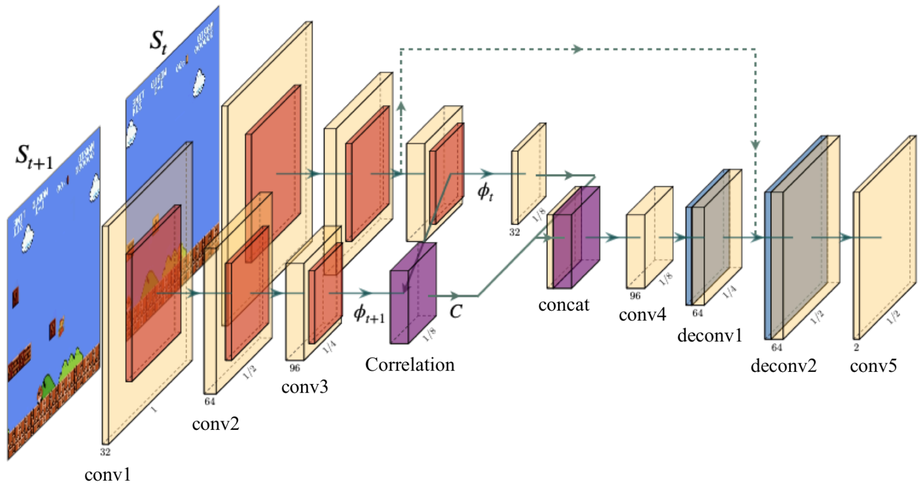


- **FICM offers the following advantages**
  - Suitable for complex and high-dimensional state space
  - Superior to ICM in environments with moving objects
  - Learns high-level features and is more general

Flow-Based Intrinsic Module Implementations



(a) The flow predictor of FICM-S

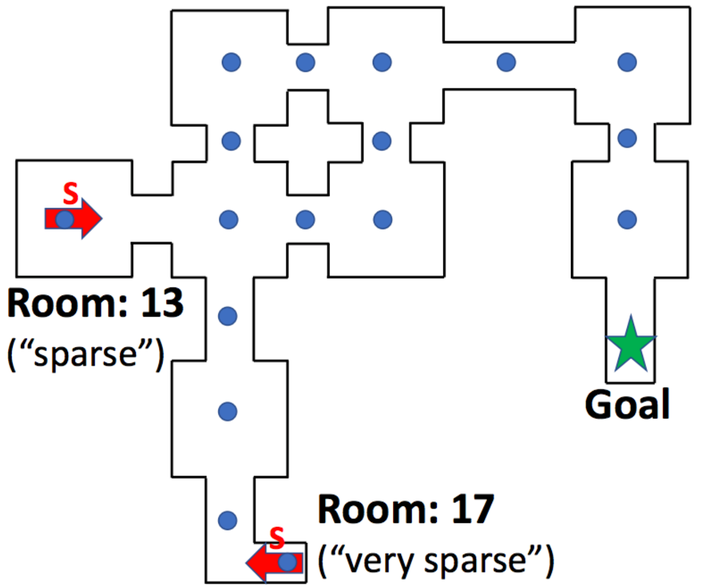


(b) The flow predictor of FICM-C

- **Two different implementations of flow predictors are provided**
  - Different implementations validate the generalizability of FICM
  - FICM only requires two states as its input, instead of eight as in ICM
  - The two implementations are based on FlowNet 2.0 modules

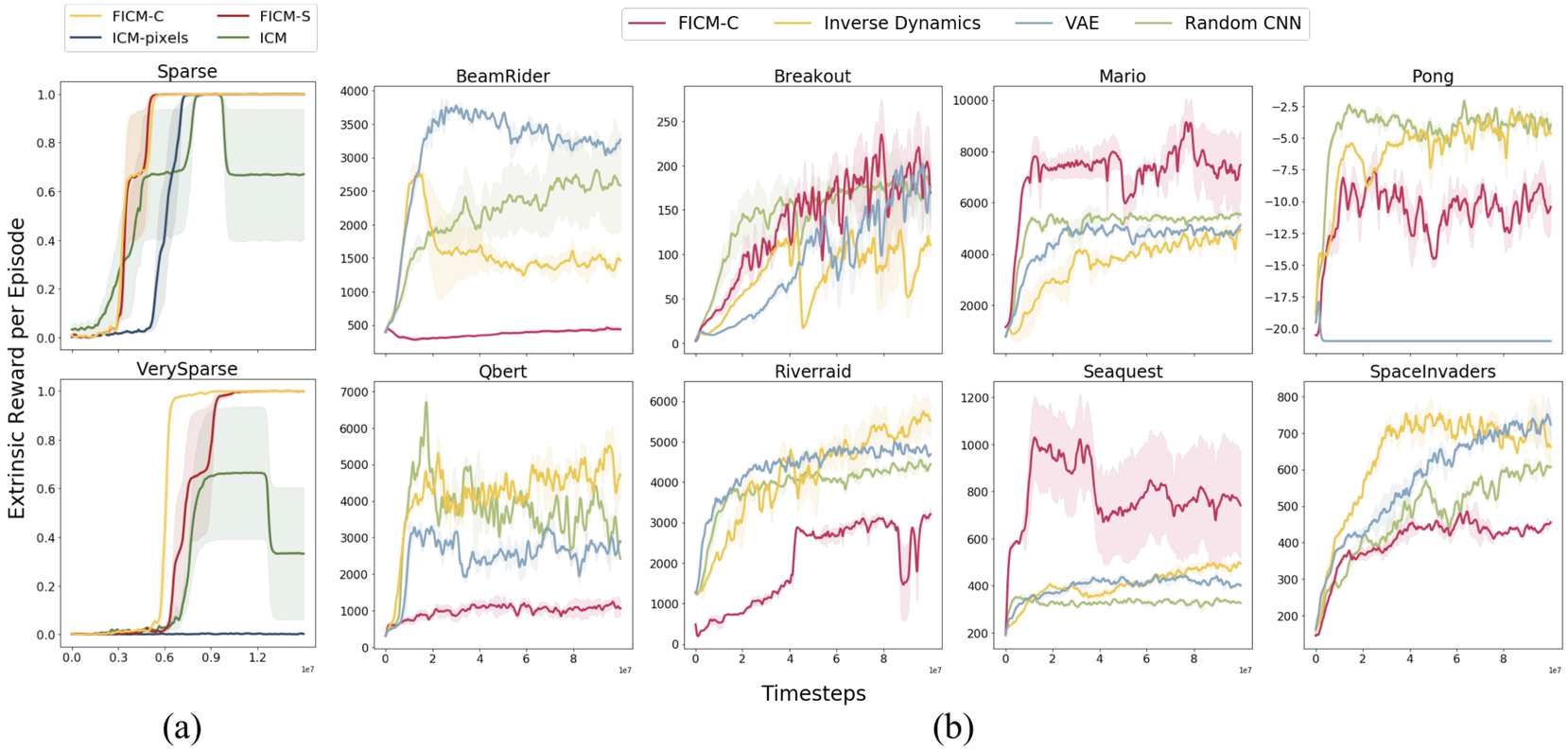
Experimental Environments

ViZDoom



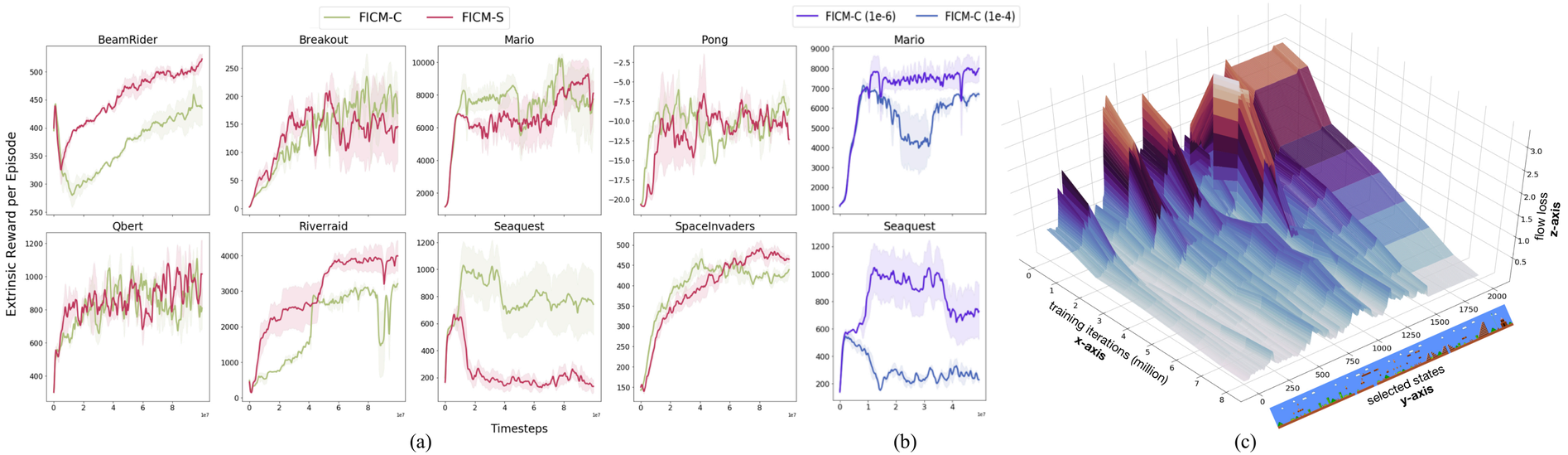
- **ViZDoom is a first-person shooting game environment**
  - The environment provides two modes: *Spare reward* and *very spare reward*
  - The agent spawn from different locations in the map
  - The goal is the only location that offers an extrinsic reward of "+1"

FICM Experimental Results



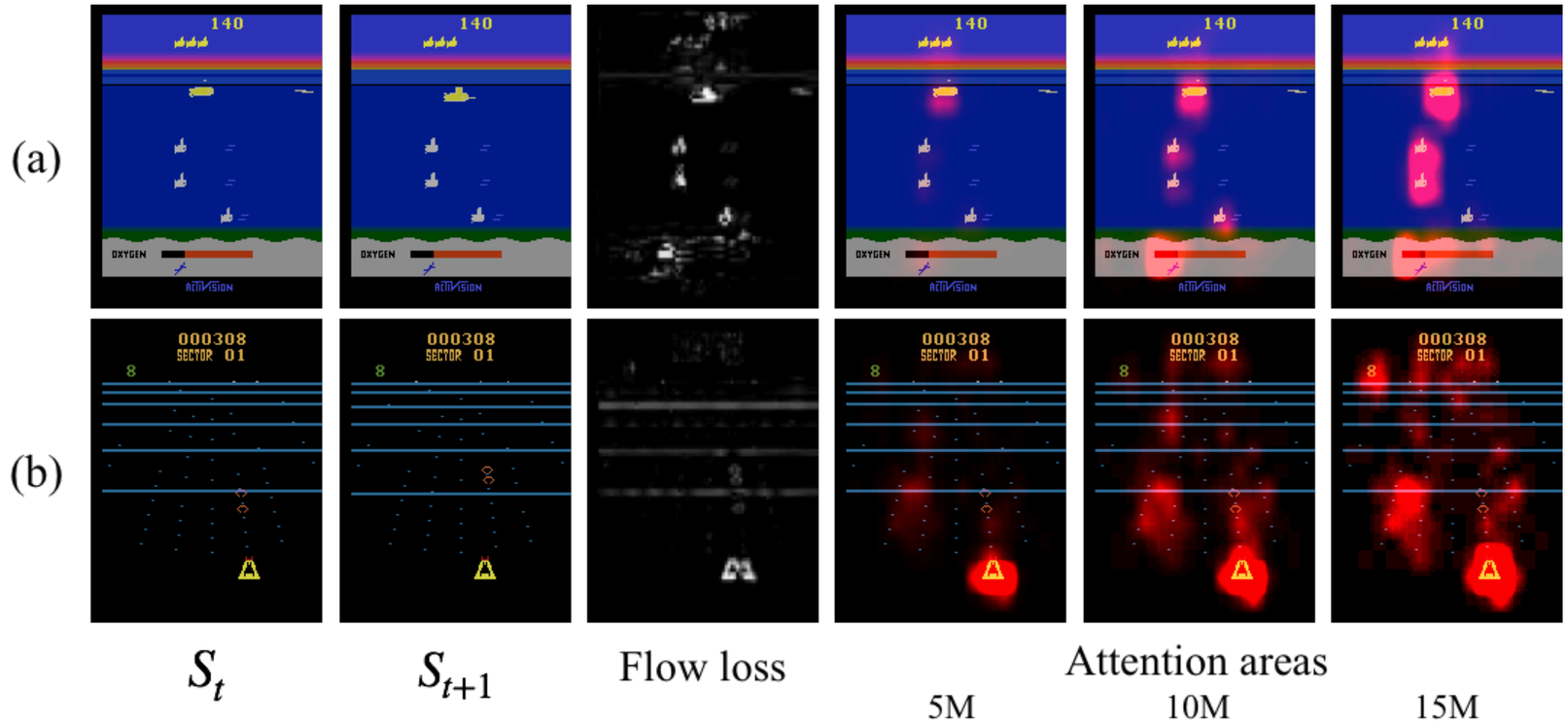
- **A number of experiments are conducted on Atari and ViZDoom**
  - We evaluated our methods with sparse extrinsic rewards in ViZDoom
  - We further evaluated the methods without extrinsic rewards in Atari games
  - We discovered that strengths of the agent and FICM models are required to be properly balanced

FICM Experimental Results



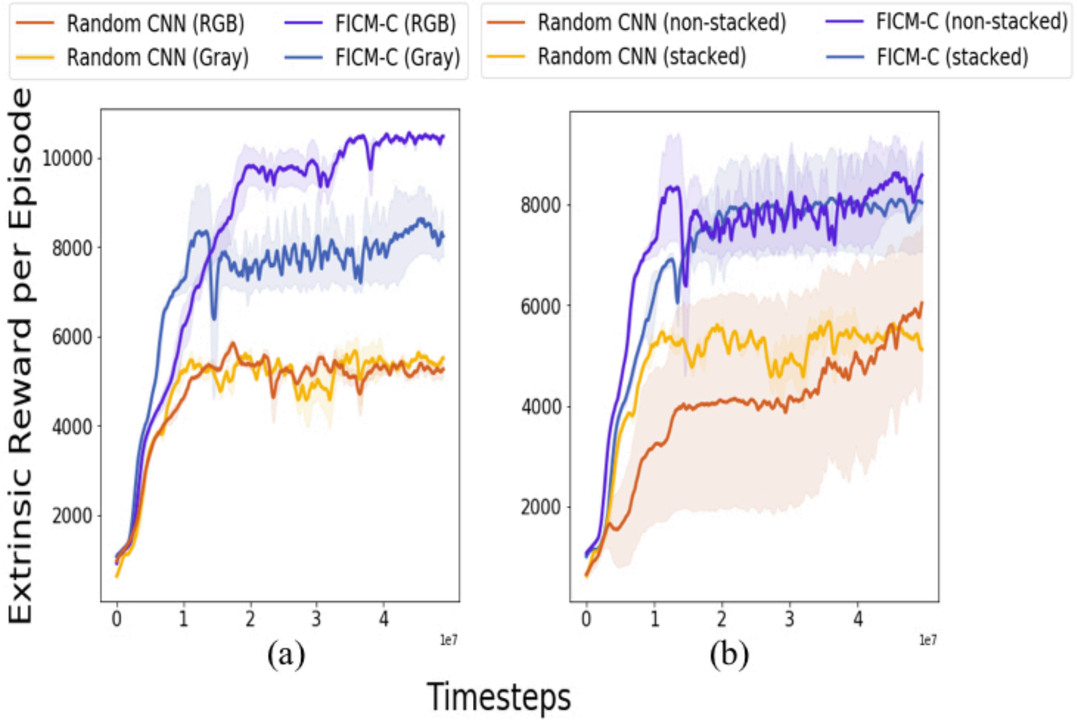
- **Ablation studies: FICM architectures, learning rates, and intrinsic reward trend**
  - The two proposed FICM architecture perform similar on most environments
  - The learning rate of the intrinsic module matters
  - The flow loss (i.e., the intrinsic rewards) decrease monotonously over time.

FICM Experimental Results



- **Visualization of flow loss and attention areas**
  - We further evaluate the proposed method on the Atari environments
  - We plot the flow loss on the third column, where the bright areas correspond to high flow loss
  - The attention areas of the agent align with those with high flow loss

FICM Experimental Results



- **Ablation studies: Comparison of different input dimensions**
  - We plot the results of **our method** versus **random CNN** for two different types of ablation analysis: number of input channels and number of stacked frames
  - FICM benefits from RGB inputs significantly on ***Super Mario Bros.***
  - Number of input frames do not introduce significant impact on performance.

Outline

- Diversity-Driven Exploration
- Flow-Based Intrinsic Curiosity Module
- **Adversarial Active Exploration for Inverse Dynamics Model Learning**
- **Macro Actions for Deep Reinforcement Learning**
- **Mixture of Step Returns in Bootstrapped DQN**
- **Efficient Inference Technique**

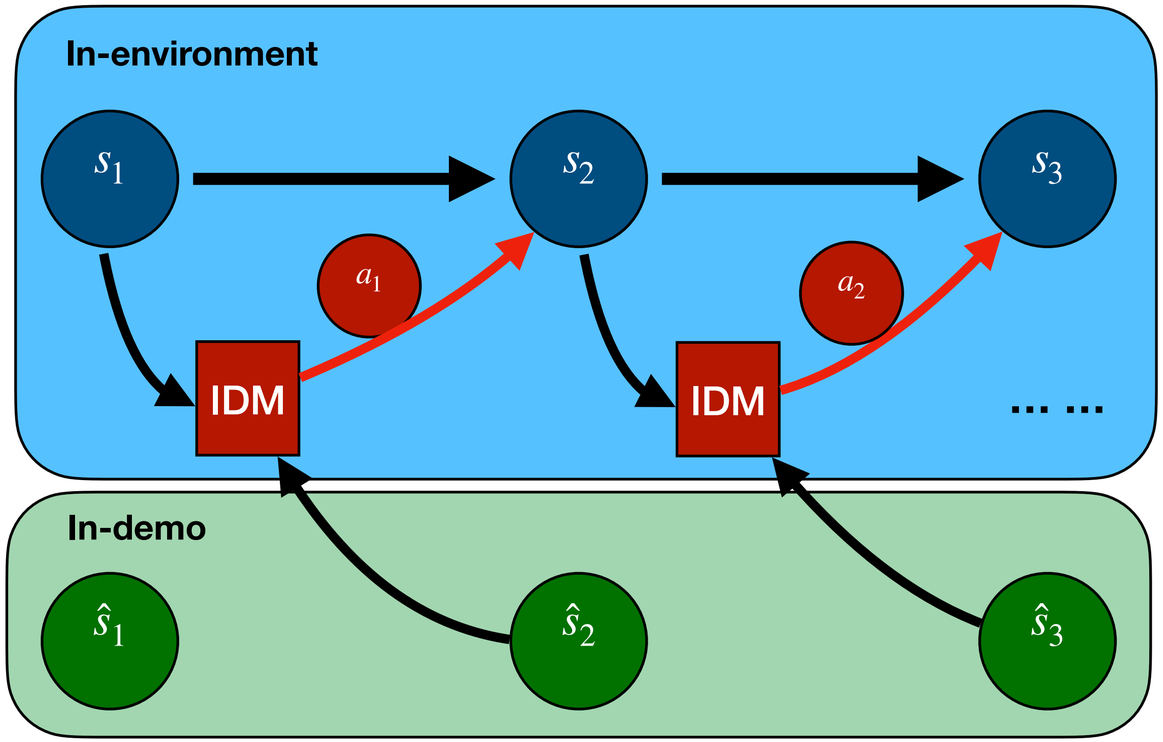
Contributions

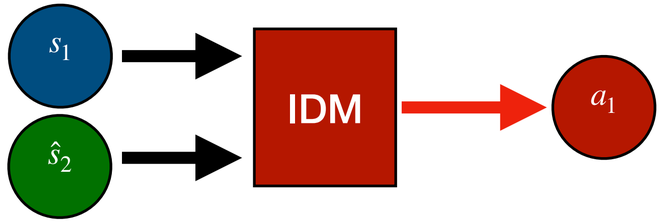
- An efficient autonomous training data acquisition strategy for learning an inverse dynamics model
- A reward shaping technique to stabilize the training process

Why are we interested in learning an inverse dynamics model (IDM)?

IDM is useful for robotic applications

Given a desired motion (i.e. a list of states, $\tau = [\hat{s}_1, \hat{s}_2, \cdots, \hat{s}_T]$), a robot can accomplish this motion by inferring the actions (i.e. a list of torques, $[a_1, a_2, \cdots, a_T]$) by IDM



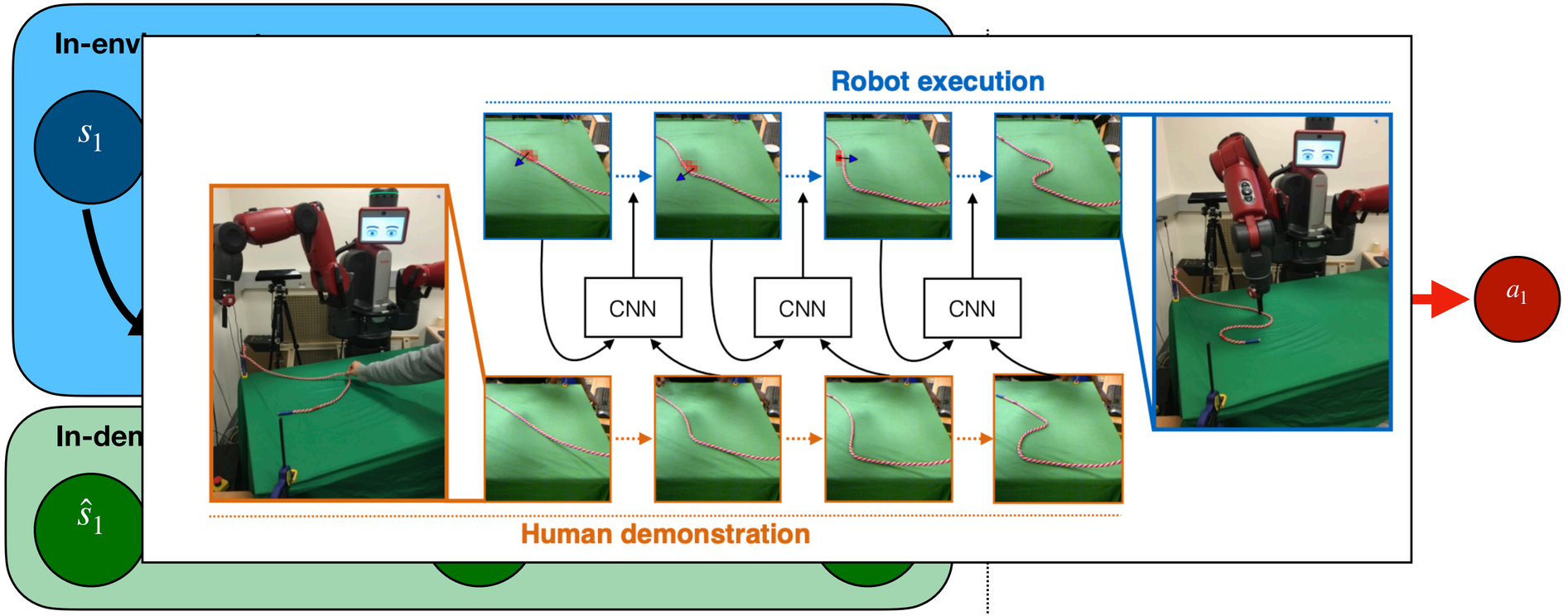


$$a_t = I(s_t, \hat{s}_{t+1})$$

Nair, Ashvin, et al. "Combining self-supervised learning and imitation for vision-based rope manipulation." *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017.
Agrawal, Pulkit, et al. "Learning to poke by poking: Experiential learning of intuitive physics." *Advances in Neural Information Processing Systems*. 2016.
Pathak, Deepak, et al. "Zero-Shot Visual Imitation." (2018) takes it as an image-based IDM

IDM is useful for robotic applications

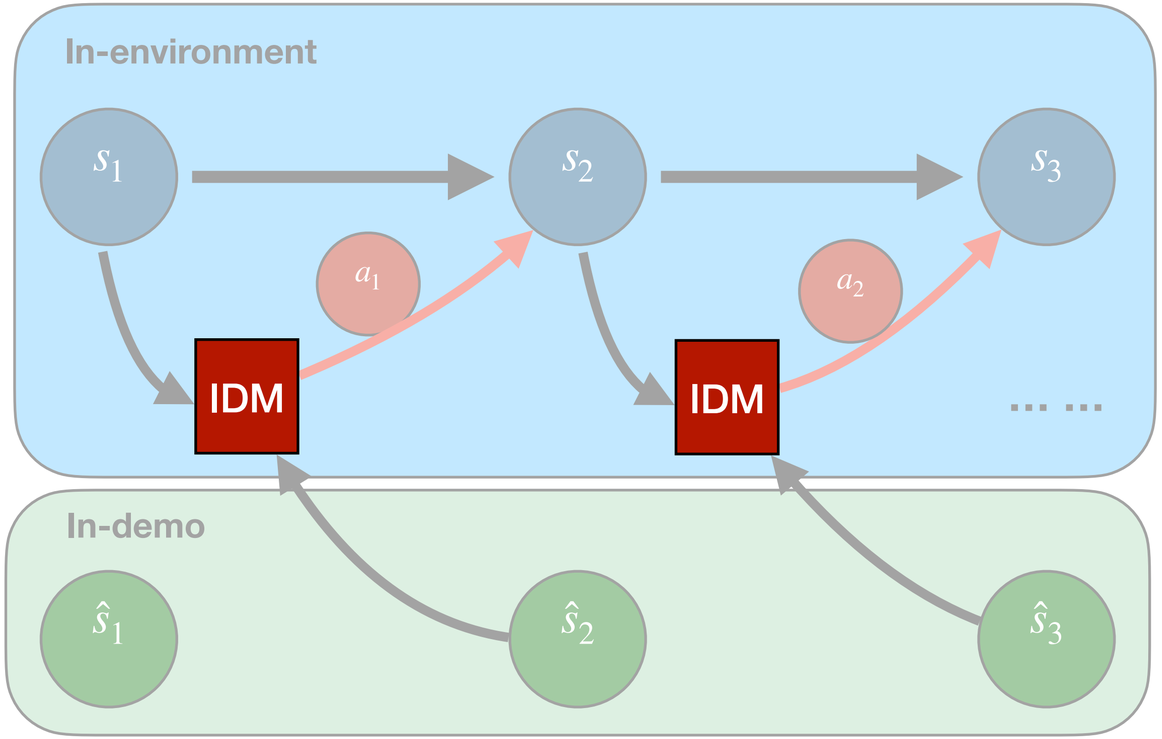
Given a desired motion (i.e. a list of states, $\tau = [\hat{s}_1, \hat{s}_2, \cdots, \hat{s}_T]$), a robot can accomplish this motion by inferring the actions (i.e. a list of torques, $[a_1, a_2, \cdots, a_T]$) by IDM

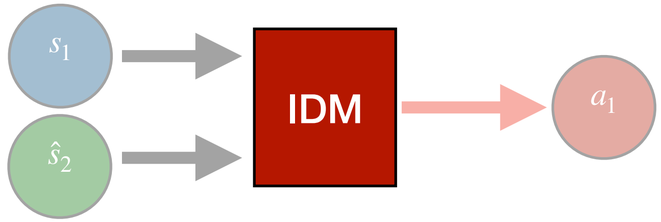


Nair, Ashvin, et al. "Combining self-supervised learning and imitation for vision-based rope manipulation." *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017.
Agrawal, Pulkit, et al. "Learning to poke by poking: Experiential learning of intuitive physics." *Advances in Neural Information Processing Systems*. 2016.
Pathak, Deepak, et al. "Zero-Shot Visual Imitation." (2018) takes it as an image-based IDM

How to obtain an IDM?

Data-driven modeling (e.g., neural network)





$$a_t = I(s_t, \hat{s}_{t+1})$$

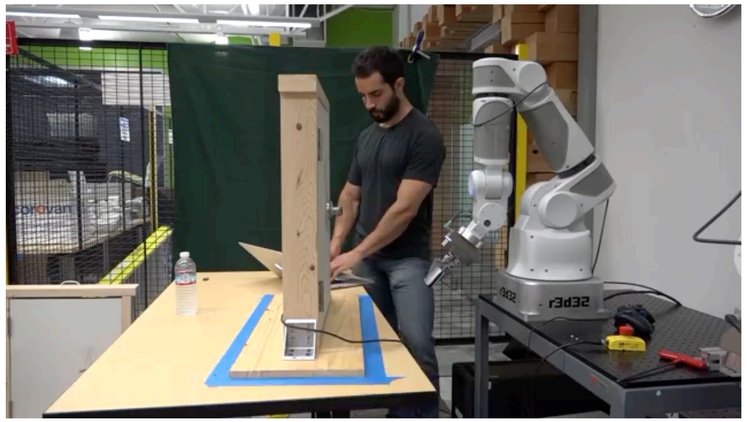
In this work, we take IDM as a Long Short Term Memory (LSTM)*

In this work, we take IDM as a Long Short Term Memory (LSTM)*

*Pathak, Deepak, et al. "Zero-Shot Visual Imitation." (2018) takes it as an image-based IDM

Training data collection

Human demonstration

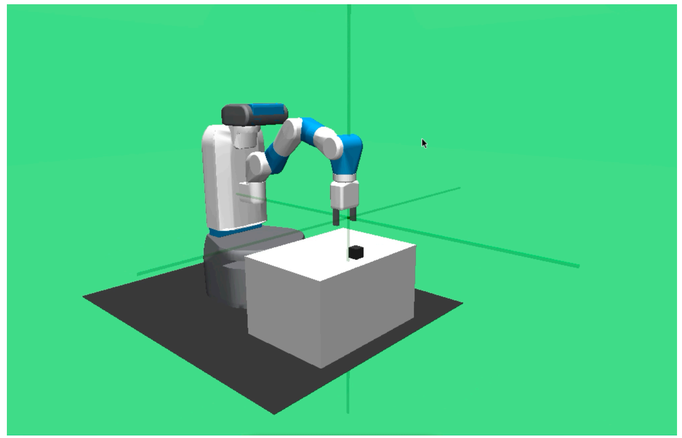


(from Google AI blog)

Pros: high-quality data (including complex motions)

Cons: too much human effort

Random exploration (by robots) [1, 2]



Pros: zero human effort

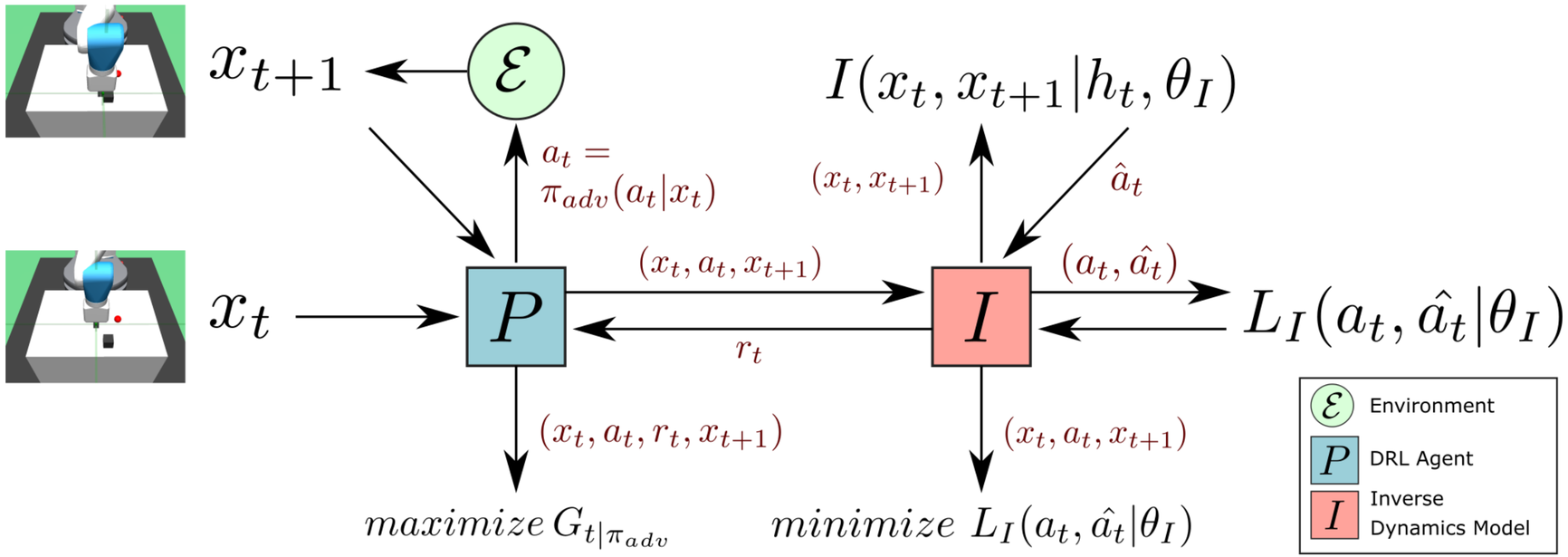
Cons: low-quality data (no complex motions)

[1] Nair, Ashvin, et al. "Combining self-supervised learning and imitation for vision-based rope manipulation." *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017.
[2] Agrawal, Pulkit, et al. "Learning to poke by poking: Experiential learning of intuitive physics." *Advances in Neural Information Processing Systems*. 2016.

Can we have an autonomous data collection strategy that has both advantages?

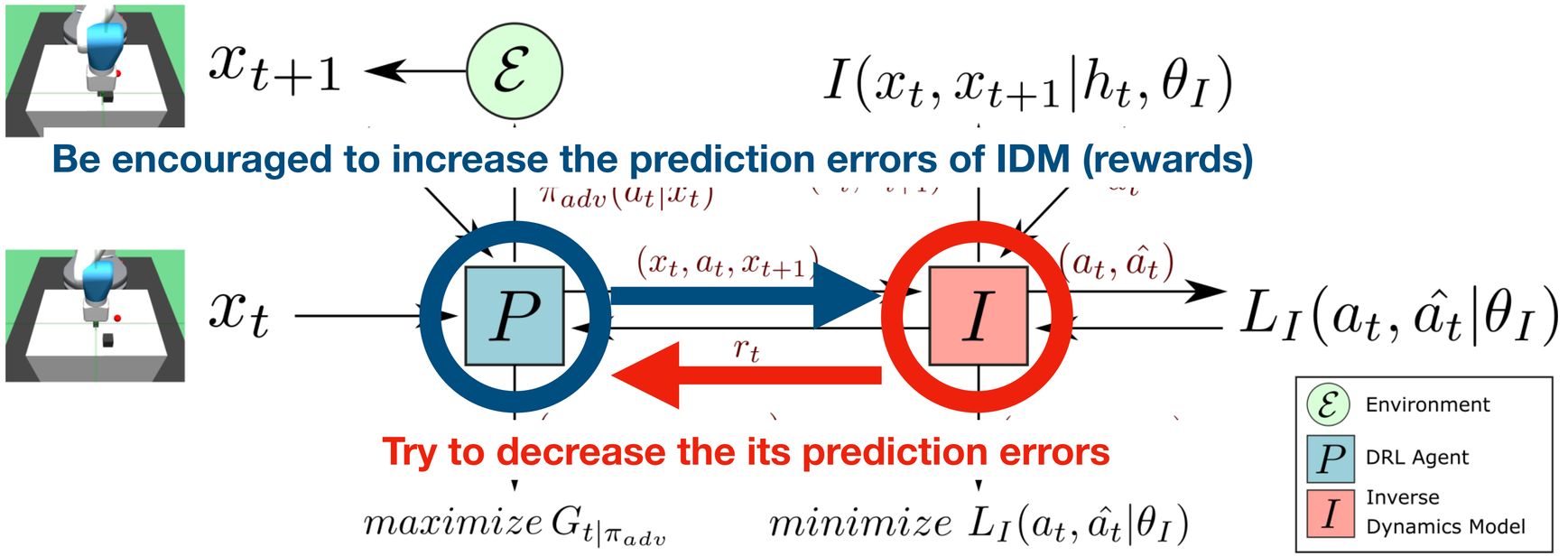
Adversarial Active Exploration

We train a **reinforcement learning (RL)*** agent
to collect **non-trivial** training data (i.e. complex motions) for IDM



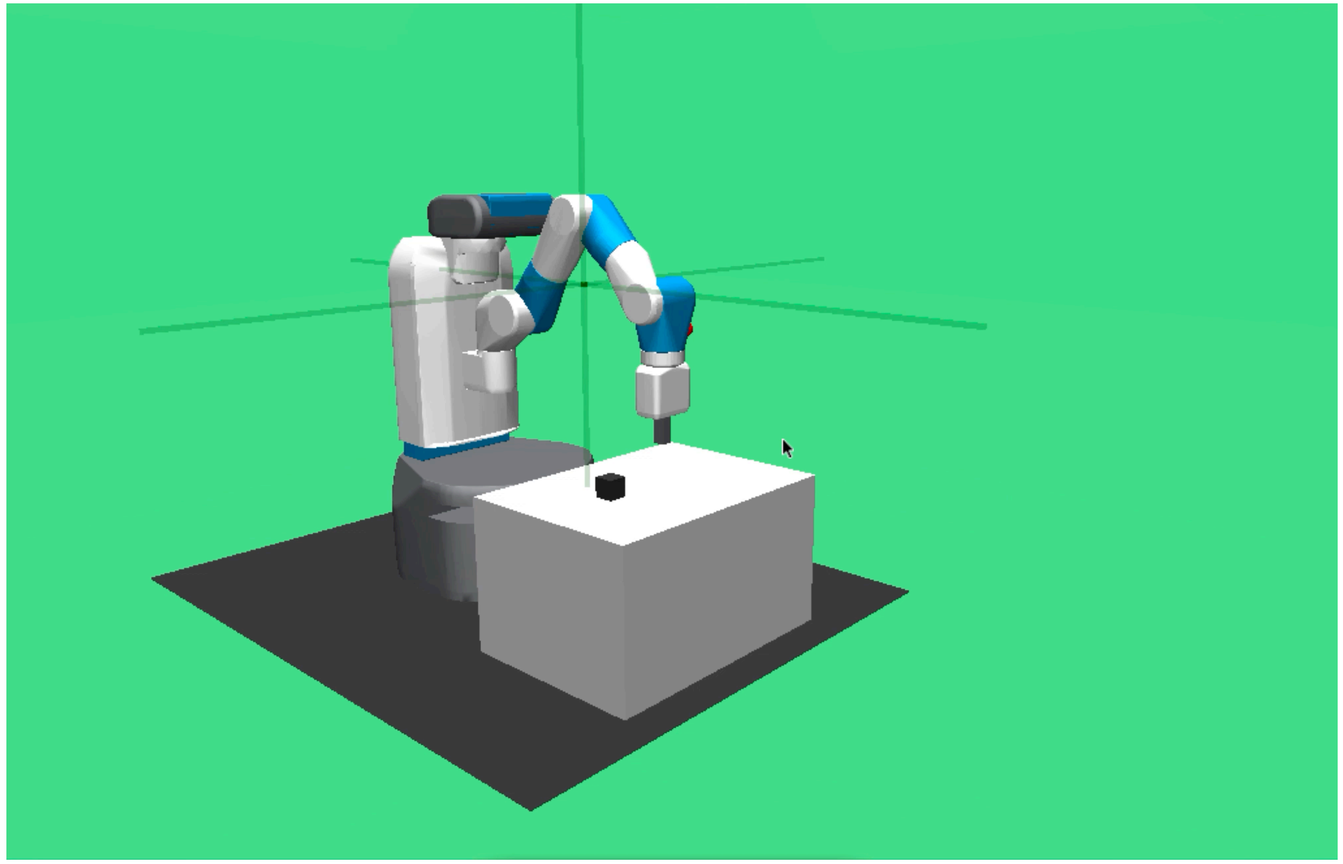
*Here we use Proximal Policy Optimization (PPO)

Adversarial Active Exploration



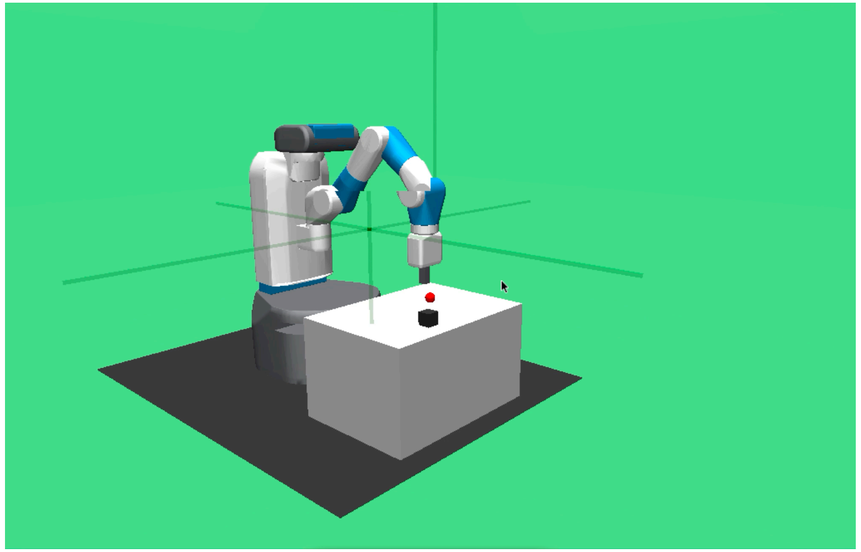
- The competitive relationship creates a **curriculum** to continually improve both sides
- As a result, there are a lot of **complex motions** in the collected dataset
- Also, our method **doesn't rely on human supervision**

Our method can collect non-trivial data



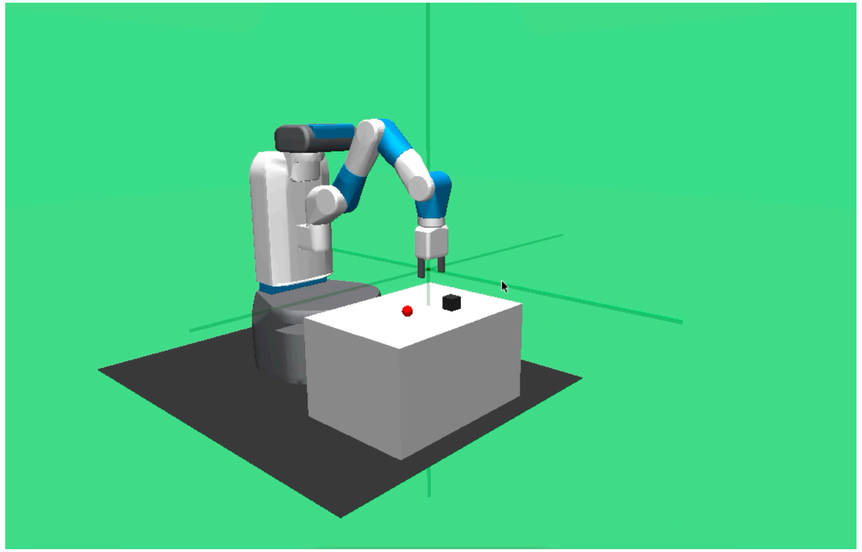
However, the collected data are too difficult sometimes

Strike out the object



Objects in projectile motions cause extraordinarily large training errors

Push the table

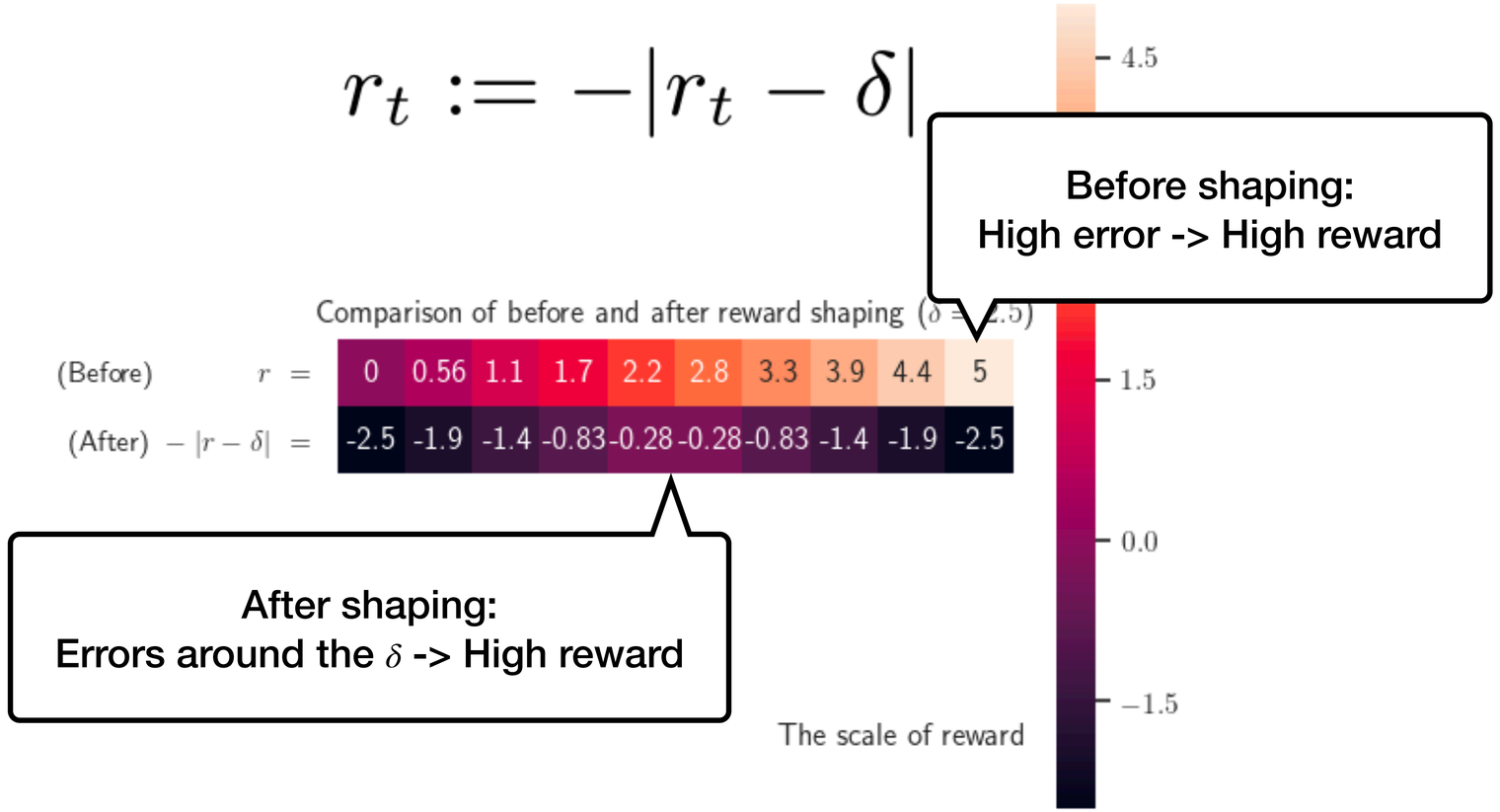


The table's position, velocity, and acceleration are not in the IDM's state space

Overly large errors lead to IDM failures

Solution: reward shaping

$$r_t := -|r_t - \delta|$$



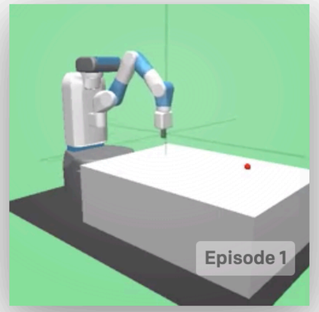
Encourage the RL agent to collect "**moderate**" samples

Benchmark platforms

FetchPickAndPlace-v1
Lift a block into the air.

FetchPush-v1
Push a block to a goal position.

FetchReach-v1
Move Fetch to a goal position.



FetchSlide-v1
Slide a puck to a goal position.

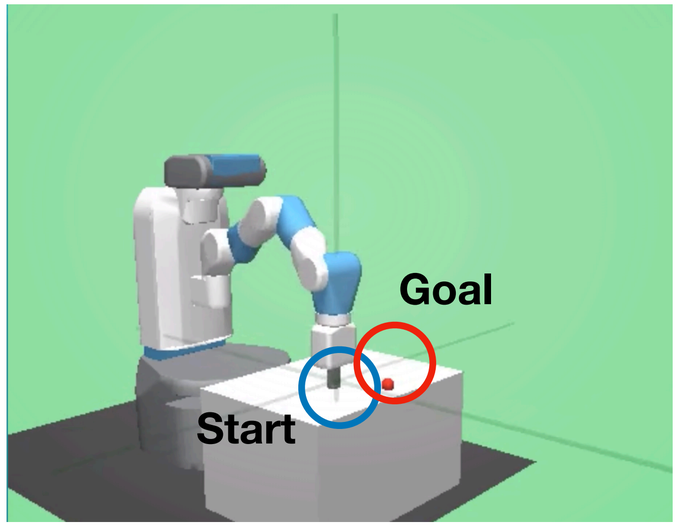
HandManipulateBlock-v0
Orient a block using a robot hand.

HandManipulateEgg-v0
Orient an egg using a robot hand.

See the detail in: gym.openai.com/envs/#robotics

Evaluation protocol

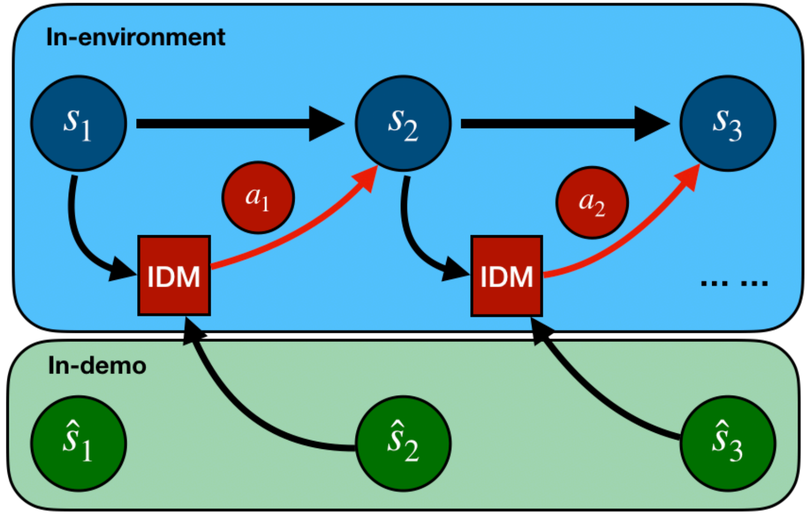
Instruct an expert to reach a randomly sampled goal (i.e. reach, push, pick, etc)



Record the trajectory

$$\tau = [\hat{s}_1, \hat{s}_2, \cdots, \hat{s}_T]$$

Instruct the robot to accomplish this trajectory τ by IDM



We report the success rate of goal achievement over multiple trajectories

Baselines

1. Random exploration (***Random***): collect training data by random actions

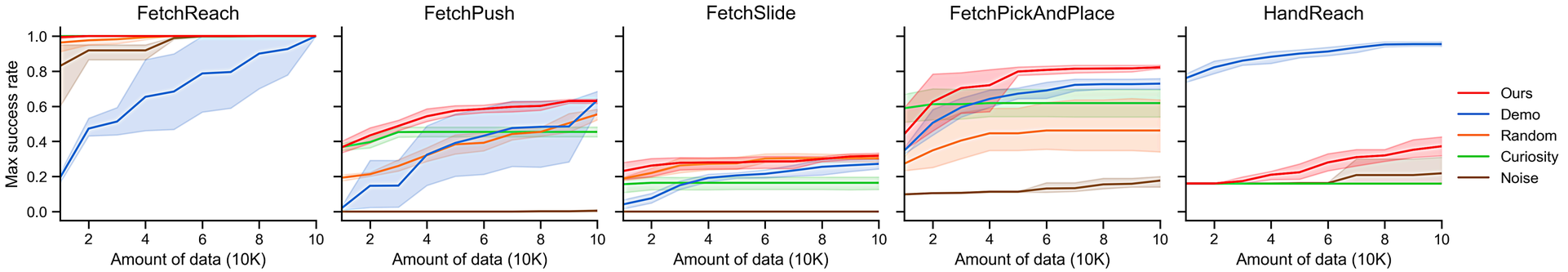
2. Demo (***Demo***): train IDM by experts' demonstration

3. Curiosity-driven exploration (***Curiosity***) [1]: maximize the **forward dynamics model's prediction errors**

4. Parameter noise (***Noise***) [2]: randomly sample the RL agents' policy parameters

[1] Pathak, Deepak, et al. "Curiosity-driven Exploration by Self-supervised Prediction." *International Conference on Machine Learning*. 2017.
[2] Plappert, Matthias, et al. "Parameter space noise for exploration." *International Conference on Representation Learning* (2017).

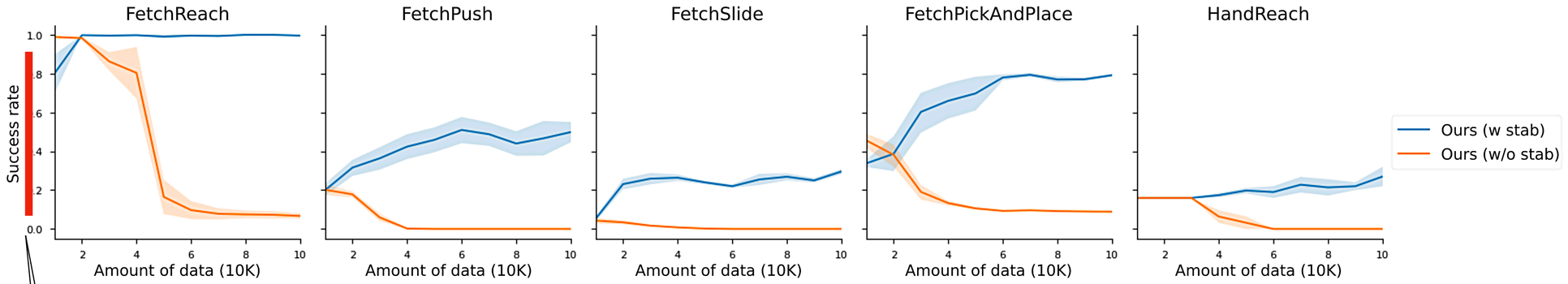
Experimental results



- Our method outperforms all the autonomous data collection strategies
- Also, the performance of our method is very close to *Demo* which uses experts' demonstration
- Being superior to *Curiosity* suggests that driving the data collection by inverse errors is more effective than by forward errors

Ablation studies: reward shaping

$$r_t := -|r_t - \delta|$$



Without shaping, we could learn very fast in the initial stage, but it collapse later

Note: we report the "**success rate**" not the "**max success rate**" in this experiment

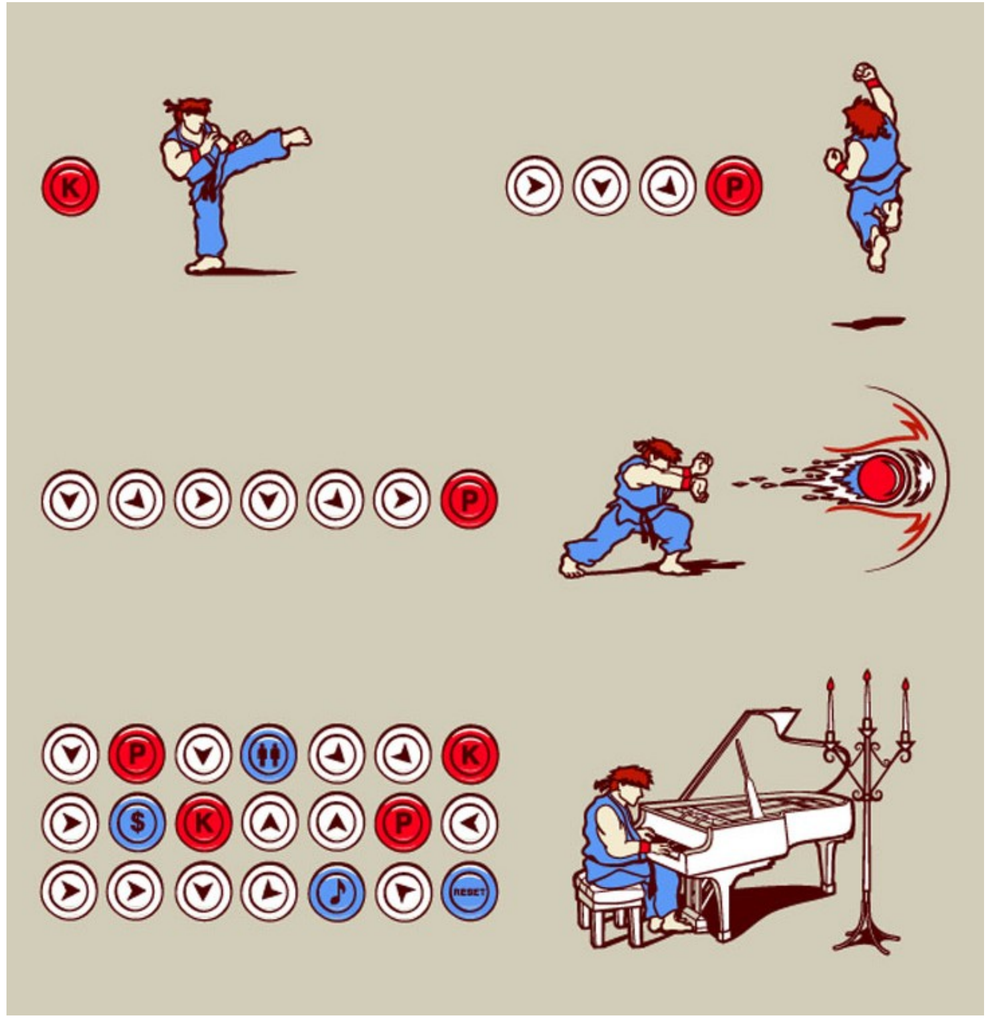
Conclusion

We introduce an efficient data collection strategy for IDM learning that requires **zero human effort** and can attain **superior performance**

Outline

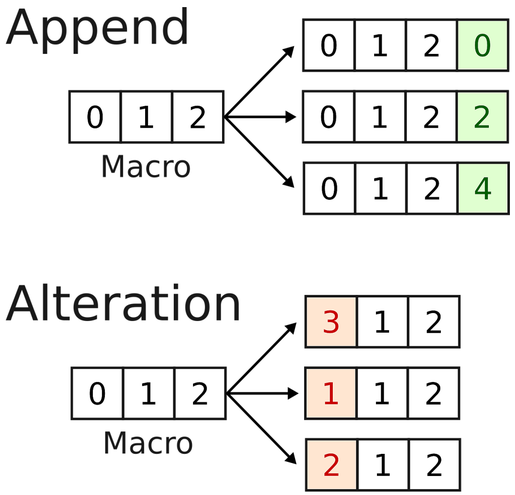
- Diversity-Driven Exploration
- Flow-Based Intrinsic Curiosity Module
- Adversarial Active Exploration for Inverse Dynamics Model Learning
- **Macro Actions for Deep Reinforcement Learning**
- **Mixture of Step Returns in Bootstrapped DQN**
- **Efficient Inference Technique**

Construction of Macro Actions for Deep Reinforcement Learning



- **Macro action is a new way for DRL agents to learn faster and more efficiently**
  - A macro action (or simply a macro) is defined as a sequence of “primitive actions” performed by the agent.
  - By following the macro action, an agent is not required to make decisions at every timestep.
  - Similar to “Combo moves” in video or basketball games.
- **Traditional approaches**
  - Repeated actions: train a neural network to learn when to stop a repeated action.
  - Extraction of mostly used action sequences from experience.

Construction of Macros via Genetic Algorithms (1/2)



Algorithm 1 Action space augmentation method

input: Primitive action space $\mathcal{A}$
input: Macro action m
output: Augmented action space $\mathcal{M}$
function AUGMENT($\mathcal{A}$, m)
    return $\mathcal{M} = \mathcal{A} \cup \{m\}$
end function

Algorithm 2 Macro evaluation method

input: Environment $\mathcal{E}$; DRL method $\mathcal{R}$; Augmented action space $\mathcal{M}$; Evaluation epochs N
output: Fitness score of $\mathcal{M}$
function EVALUATE($\mathcal{E}$, $\mathcal{R}$, $\mathcal{M}$, N)
    initialize: $E = [e_1, \ldots, e_N] \; \forall \, e \in E, e = 0$
    for i **in** 1 **to** N **do** ▷ N epoch
        Learn a policy over $\mathcal{M}$ in $\mathcal{E}$ using $\mathcal{R}$
        e_i = the average of all "last 100 episode mean rewards"
    end for
    return Average of E
end function

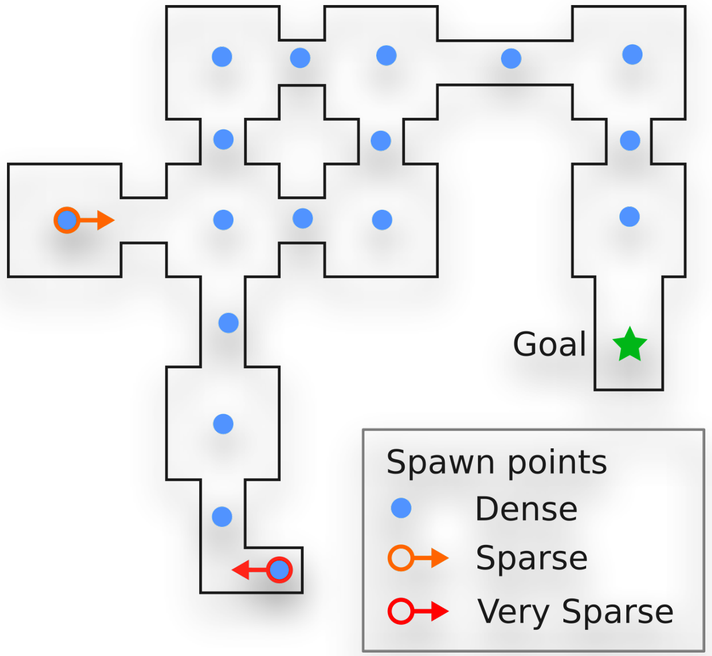
- **Macro action definition**
  - A macro can be any combination of primitive actions of arbitrary length.
- **Augment the original action space**
  - A macro is appended to the original action space.
  - The new action is called the "**augmented action space**"
- **Search for the best macro in the macro action space**
  - We define the macro action space as a space containing all possible macros.
  - We use **genetic algorithms** to search the best macros

Construction of Macros via Genetic Algorithms (2/2)

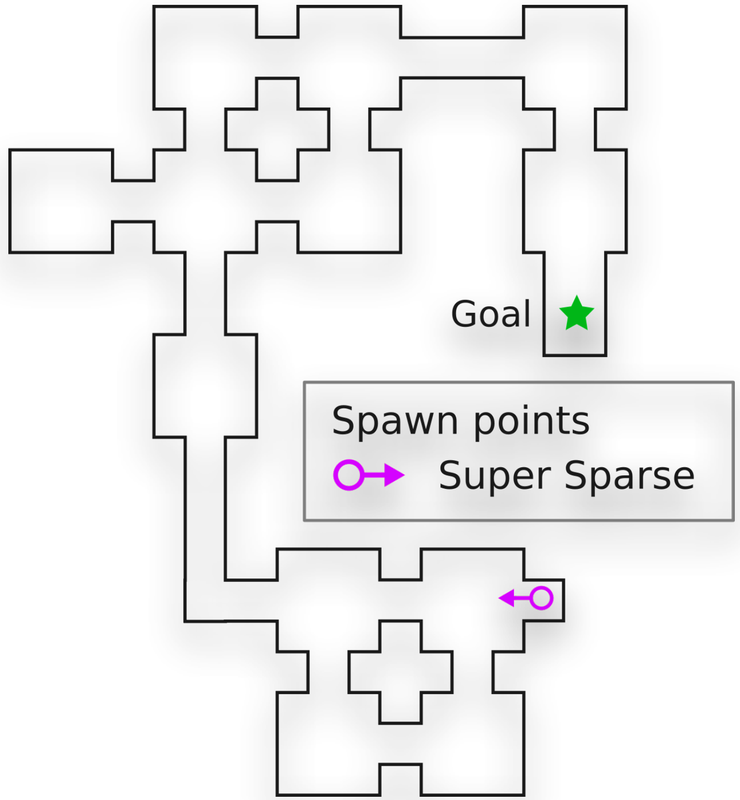
Algorithm 3 Macro construction algorithm based on GA

input: Environment $\mathcal{E}$; DRL algorithm $\mathcal{R}$; Total number of evaluated macros k
input: The sizes of sets Q, Q_+, Q_* = q, q_+, q_* respectively
output: List of the top q highest-performing macros evaluated Q
function CONSTRUCTION($\mathcal{E}$, $\mathcal{R}$, k, q, q_+, q_*)
    initialize: $\mathcal{A}$ = the primitive action space of $\mathcal{E}$
    initialize: $M = [m_1, \ldots, m_q]$, a list of q randomly generated macros such that $\forall\ m \in M, |m| = 2$
    initialize: $F = [f_1, \ldots, f_q], \forall\ f \in F, f = 0$, the fitness scores for all the macros in M
    initialize: $Q = \emptyset, Q_+ = \emptyset, Q_* = \emptyset, i = 0$
    while $i < k$ **do** ▷ Each iteration is one generation
        for j **in** 1 **to** q **do**
            $\mathcal{M} = \text{AUGMENT}(\mathcal{A}, m_j)$
            $f_j = \text{EVALUATE}(\mathcal{E}, \mathcal{R}, \mathcal{M})$
            $i = i + 1$
            if $i >= k$ **then break**
        end for
        $\bar{M} = [m_1, \ldots, m_q]$ the top q scoring macros in M according to F
        Q = the top q scoring macros in $Q \cup \bar{M}$
        for m **in** List of q_+ randomly selected macros in Q **do**
            $Q_+ = Q_+ \cup \{\text{APPEND}(\mathcal{A}, m)\}$
        end for
        for m **in** List of q_* randomly selected macros in Q **do**
            $Q_* = Q_* \cup \{\text{ALTER}(\mathcal{A}, m)\}$
        end for
        $M = Q_+ \cup Q_*$
    end while
    return Q
end function

ViZDoom Environments



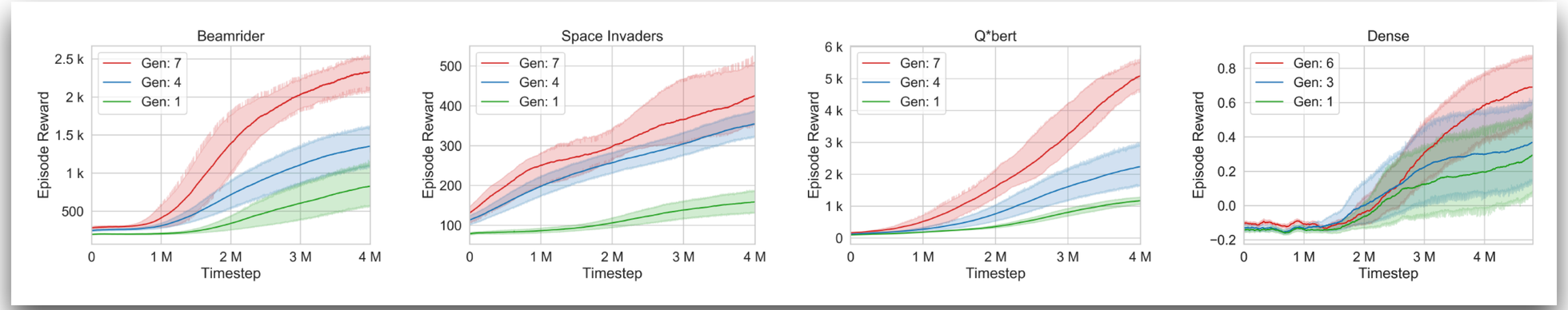
(a) Dense, Sparse, and Very Sparse



(a) Super Sparse

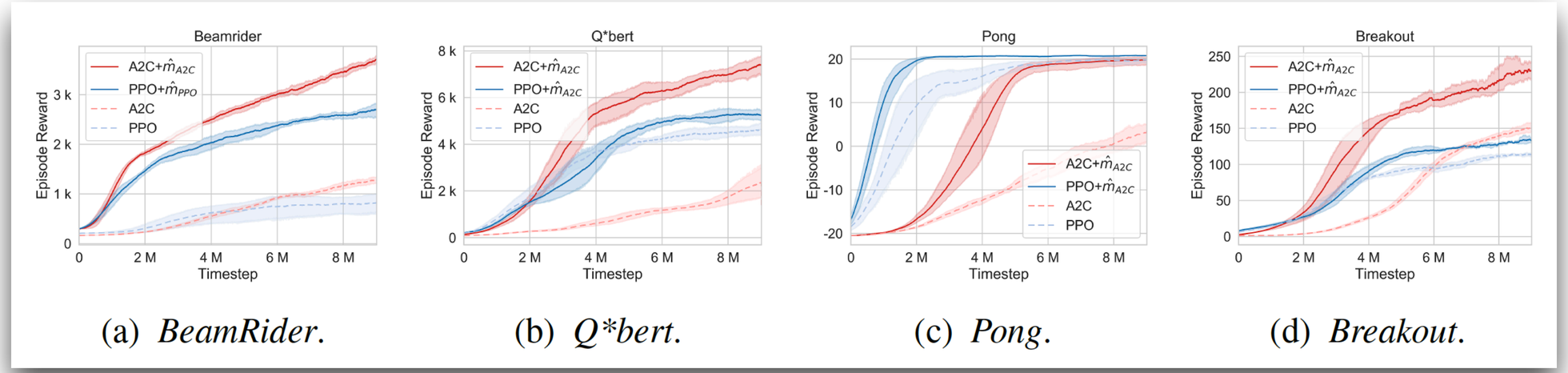
- **Four different environments are revised from the ViZDoom engine**
  - *Dense*, *Sparse*, *Very Sparse*, and *Super Sparse* correspond to different spawn points
  - An agent obtains "**+1**" reward when reach the **goal**
  - Different scarcity correspond to different difficult to reach the final **goal**

Evolution of Macros Over Generations



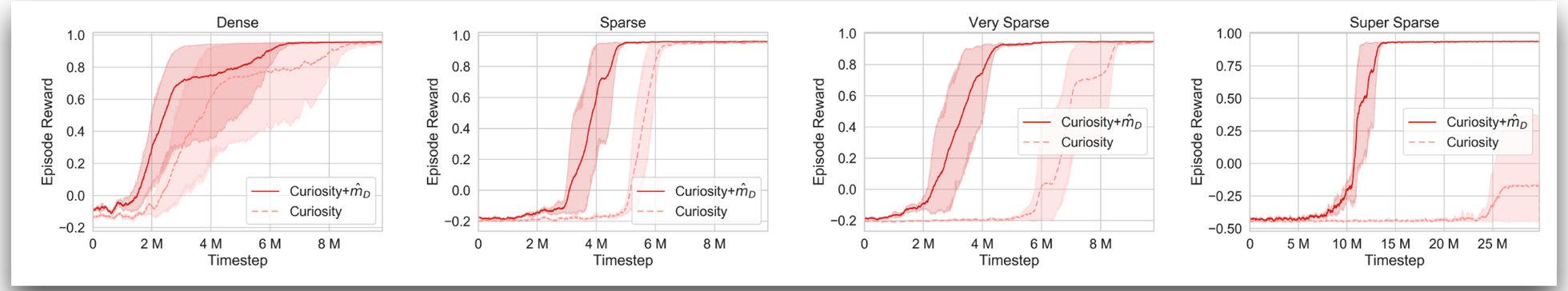
- **Learning curves of A2C trained using the augmented macro space**
  - Our methodology appends the "**A2C**" baseline with a macro $\mathbf{m_{A2C}}$
  - Our simulations are performed for Atari games using multiple generations of macros
  - "Gen" denotes generations
  - It is observed from the trends that the mean episode rewards obtained by the agents improve with generations
  - Later generations of the constructed macros may outperform earlier generations.

Transferability of Macros Among DRL Methods



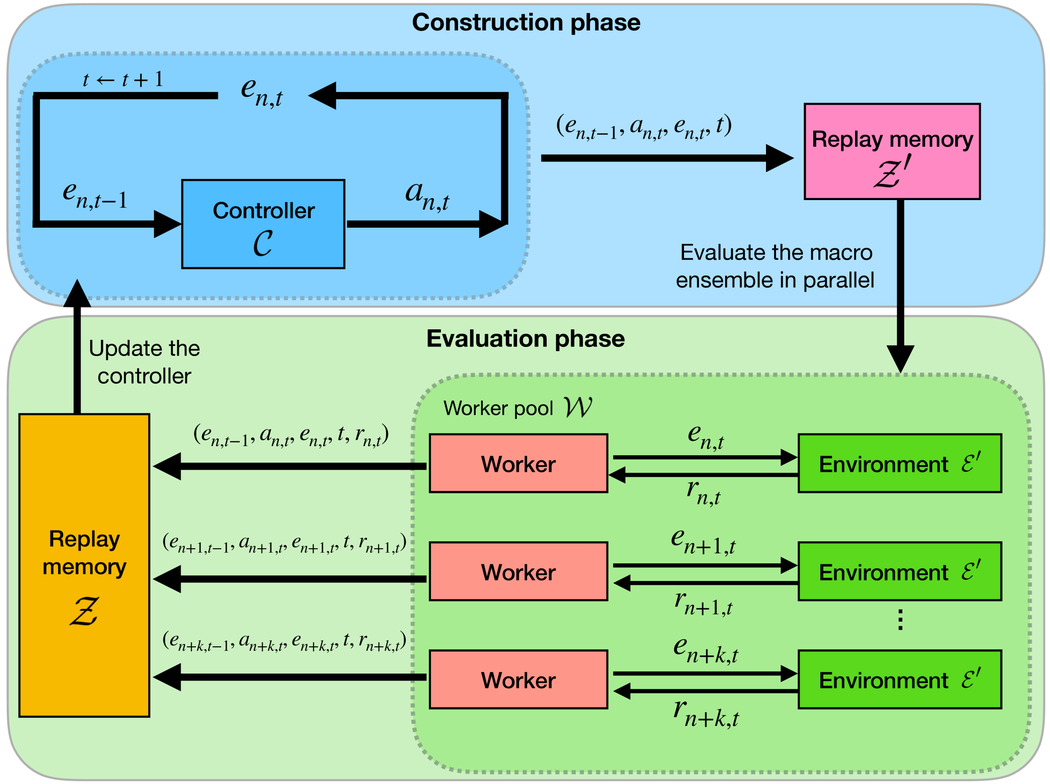
- **Comparison of our methodology versus the "A2C" and "PPO" baseline**
  - Our methodology appends the "**A2C**" and "**PPO**" baselines with a macro $\mathbf{m_{A2C}}$ and $\mathbf{m_{PPO}}$, respectively.
- **Examination of whether the best macros constructed by our methodology is complementary to the baseline DRL methods**
  - The learning curves reveal that the baseline DRL methods are both significantly benefited from the augmented action spaces.
- **Inspection of whether the macro constructed for one DRL baseline can be transferred to and utilized by another DRL baseline**
  - The results also show that PPO is able to use m^ A2C to enhance both the performance and the learning speed of the agents.

Transferability of Macros Among Environments: Results



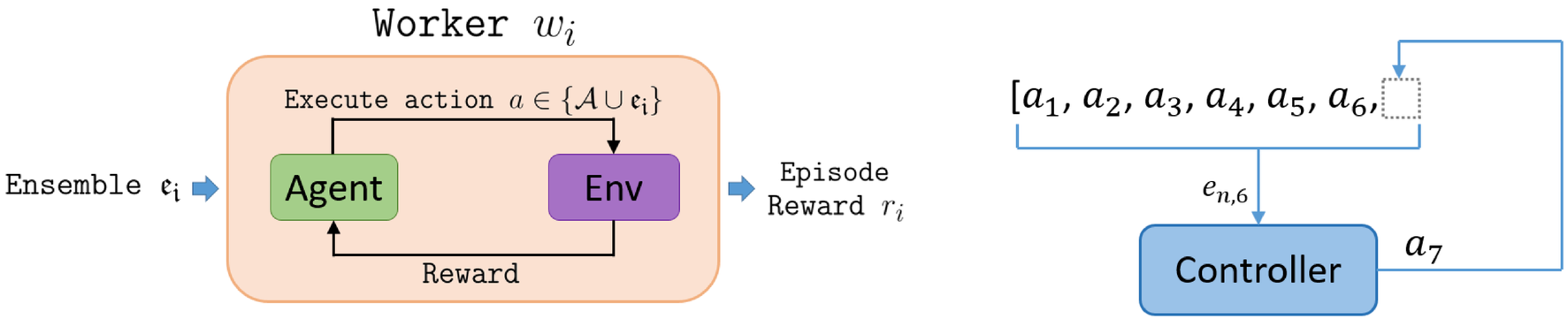
- **Comparison of our methodology versus the "Curiosity" baseline**
  - Our methodology appends the "**Curiosity**" baseline with a macro $\mathbf{m_D}$
  - Simulations are performed in four environmental settings:
    - *Dense*, *Sparse*, *Very Sparse*, and *Super Sparse*
  - In all of the four settings, our methodology allows agents to learn faster than the "**Curiosity**" baseline, which can not even learn to play in the *Super Sparse* setting
  - It is observed that the same macro $\mathbf{m_D}$ can be **transferred** among different environments.

Macro Action Ensemble Search



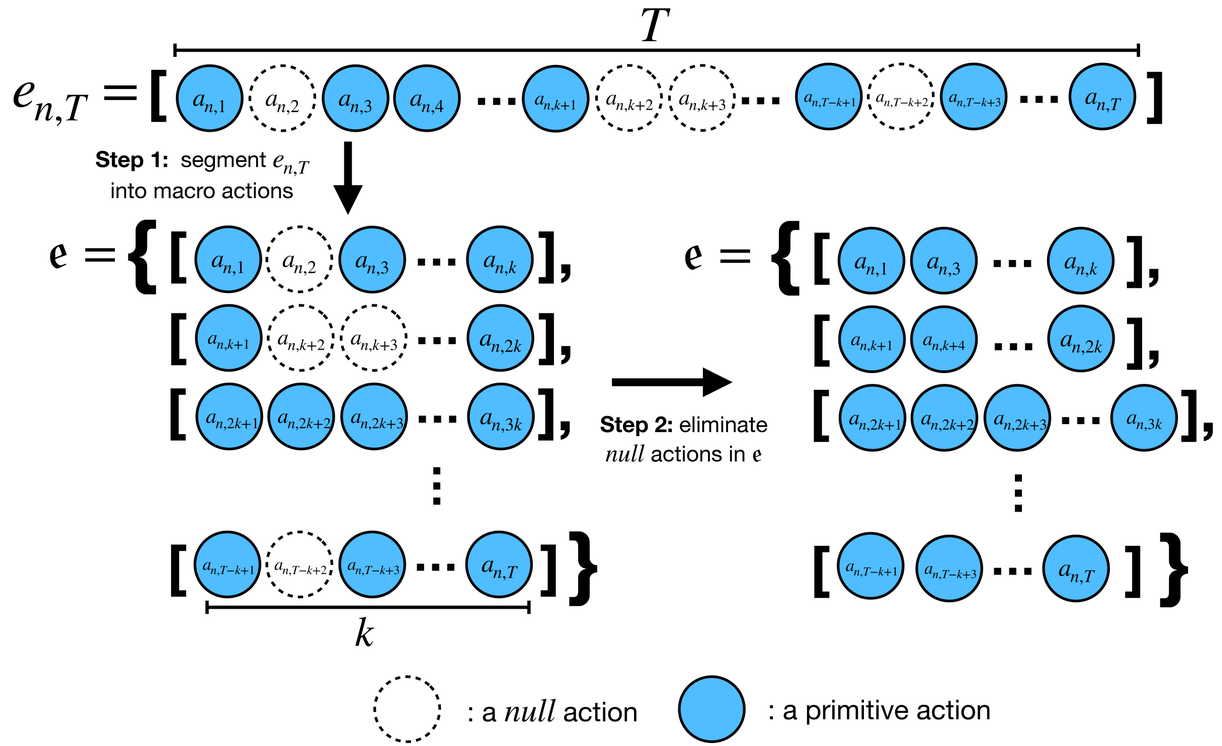
- **Overview of the proposed macro action ensemble construction framework**
  - The proposed methodology is composed of two phases: construction and evaluation
  - In the construction phase, an RL-based controller *C* constructs a macro action ensemble and sends them to a set of asynchronous parallel workers.
  - During the evaluation phase, a set of parallel workers implemented as agents evaluate the performance of the macro action ensembles in the target environment.

Ensemble Evaluation Phase



- **Overview of the evaluation phase**
  - The controller first distributes an ensemble *e* to an arbitrary worker from the worker pool W for evaluation.
  - The worker agent next uses M = A∪e to interact with the target environment, and learns a policy that is able to exploit M.
  - At the end of the evaluation phase, their performances measured as the received rewards are stored in a replay memory Z. The controller C then updates its policy using the data sampled from Z.

Ensemble Search via A Controller



- **Formulate the construction process of a macro action ensemble as an MDP**
  - A macro ensemble e is defined as a set of macros e = {m1, m2, . . . , mω }, where ω is some non-negative integer.
  - The set of ensembles form a macro action ensemble space.
  - We represent an ensemble e as a vector of primitive actions with length T, (i.e., e = [a_1, a_2, · · · , a_T].)
  - Each primitive action at in e stands for the decision of the controller C at episode step t in this MDP, where t ∈ {1, · · · ,T }.

Construction and Evaluation Algorithms

Algorithm 1 Macro action ensemble construction algorithm

input: The total number of ensemble searching episodes N
input: The fix length of an ensemble sequence T
input: Environment $\mathcal{E}$
input: Worker pool $\mathcal{W}$
output: Best constructed macro ensemble $\hat{\mathfrak{e}}$
Initialize controller $\mathcal{C}$ with random weights
Initialize empty replay memories $\mathcal{Z}$, $\mathcal{Z}'$
Launch PARALLEL EVALUATION($\mathcal{E}$, $\mathcal{W}$, $\mathcal{Z}$, $\mathcal{Z}'$)
for $n = 1, 2, ..., N$ **do** // the n^{th} ensemble
    Initialize ensemble sequence $e_{n,0}$ to a empty vector with size T
    for $t = 1, 2, ..., \mathrm{T}$ **do**
        $a_{n,t} \leftarrow$ { Select a random action a, // with probability ϵ ; Action predicted by $\mathcal{C}$ using $e_{n,t-1}$, // with probability $1-\epsilon$ }
        $e_{n,t} \leftarrow e_{n,t-1} \parallel a_{n,t}$
        $r_{n,t}$ is evaluated in PARALLEL EVALUATION
        Store $(e_{n,t-1}, a_{n,t}, e_{n,t}, t)$ into $\mathcal{Z}'$
    end for
    Optimize $\mathcal{C}$ by sampling mini-batches from $\mathcal{Z}$ using DQN method, where $\mathcal{Z}$ filled by PARALLEL EVALUATION
end for
i = $argmax_i(r_{i,T}), \forall r_{i,T} \in \mathcal{Z}$
Transform $e_{i,T}$ into macro ensemble $\hat{\mathfrak{e}}$

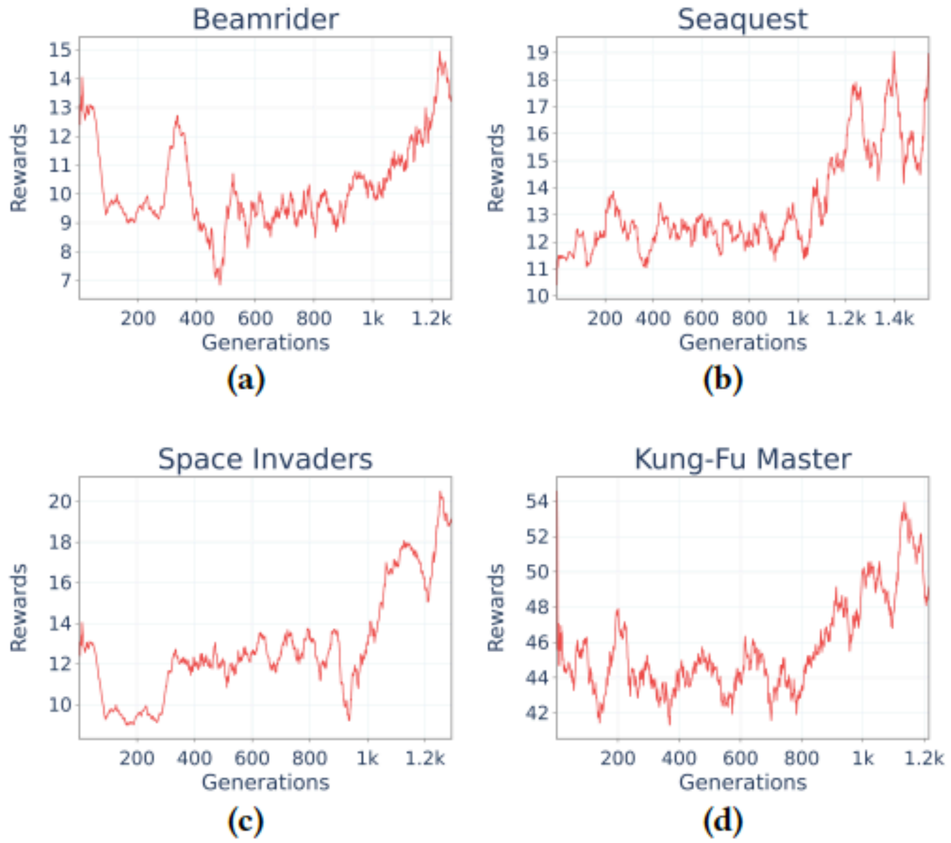
Algorithm 2 Parallel Evaluation

input: Environment $\mathcal{E}$;
input: Worker Pool $\mathcal{W}$;
input: Replay Memory $\mathcal{Z}$; $\mathcal{Z}'$;
function PARALLEL EVALUATION($\mathcal{E}$, $\mathcal{W}$, $\mathcal{Z}$, $\mathcal{Z}'$)
    while True **do**
        Pop $(e_{n,t-1}, a_{n,t}, e_{n,t}, t)$ from $\mathcal{Z}'$
        Run WORKER ROUTINE($\mathcal{E}$, $e_{n,t-1}, a_{n,t}, e_{n,t}, t$, $\mathcal{Z}$) asynchronously with an available worker selected from $\mathcal{W}$
    end while
end function

function WORKER ROUTINE($\mathcal{E}$, $e_{n,t-1}, a_{n,t}, e_{n,t}, t$, $\mathcal{Z}$)
    Duplicate $\mathcal{E}$ as $\mathcal{E}'$
    if $t == T$ **then**
        $r_{n,t} =$ ENSEMBLE EVALUATION($\mathcal{E}', e_{n,t}$)
    else
        $r_{n,t} = 0$
    end if
    Add $(e_{n,t-1}, a_{n,t}, e_{n,t}, t, r_{n,t})$ into $\mathcal{Z}$
end function

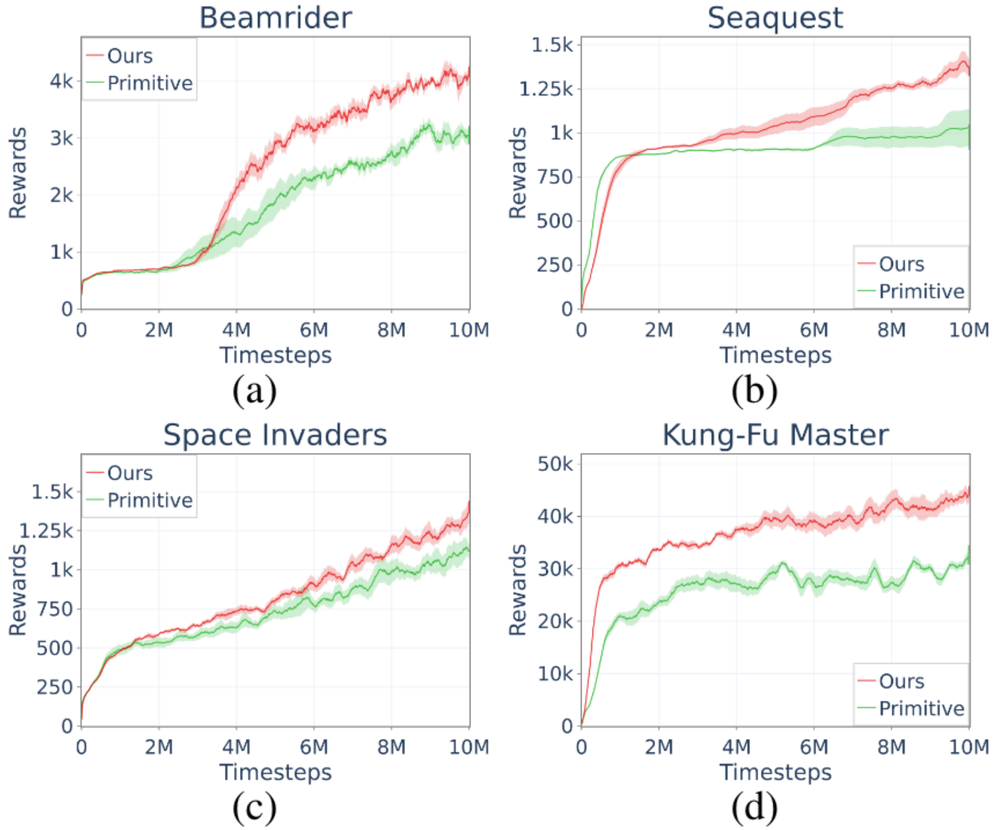
- **The construction and evaluation algorithms of a macro ensemble are summarized as Algorithms 1 & 2, respectively**
  - We train a controller based on DQN [20], with the goal to maximize the return in that MDP.
  - The controller C then iteratively predicts a primitive action a_t, where a_t can be a typical primitive action or a null action (denoted as "–1"), based on s_t and replaces the t^{th} element in the e_n with at until the termination time step T is reached.
  - After the agent is trained for a fixed number of timesteps, it is evaluated for additional 100 episodes.

Macro Ensemble Learning Curves



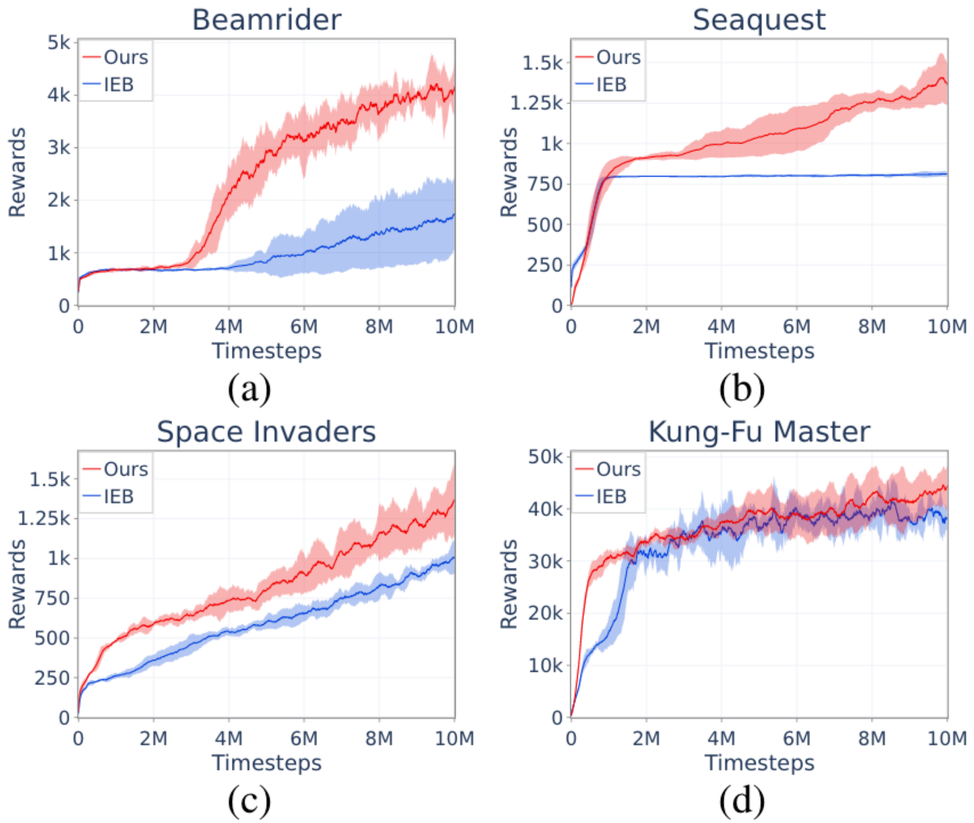
- **Learning curve of the controller**
  - In this plot, the x-axis represents the training episodes, which is also the number of the macro ensembles generated by the controller.
  - The y-axis represents the average performance of the agent training for 5M timesteps based on the constructed ensembles.
  - The rising curve indicates that the macro ensemble constructed by the controller is better than the randomly generated one.

Comparison against Primitive Actions



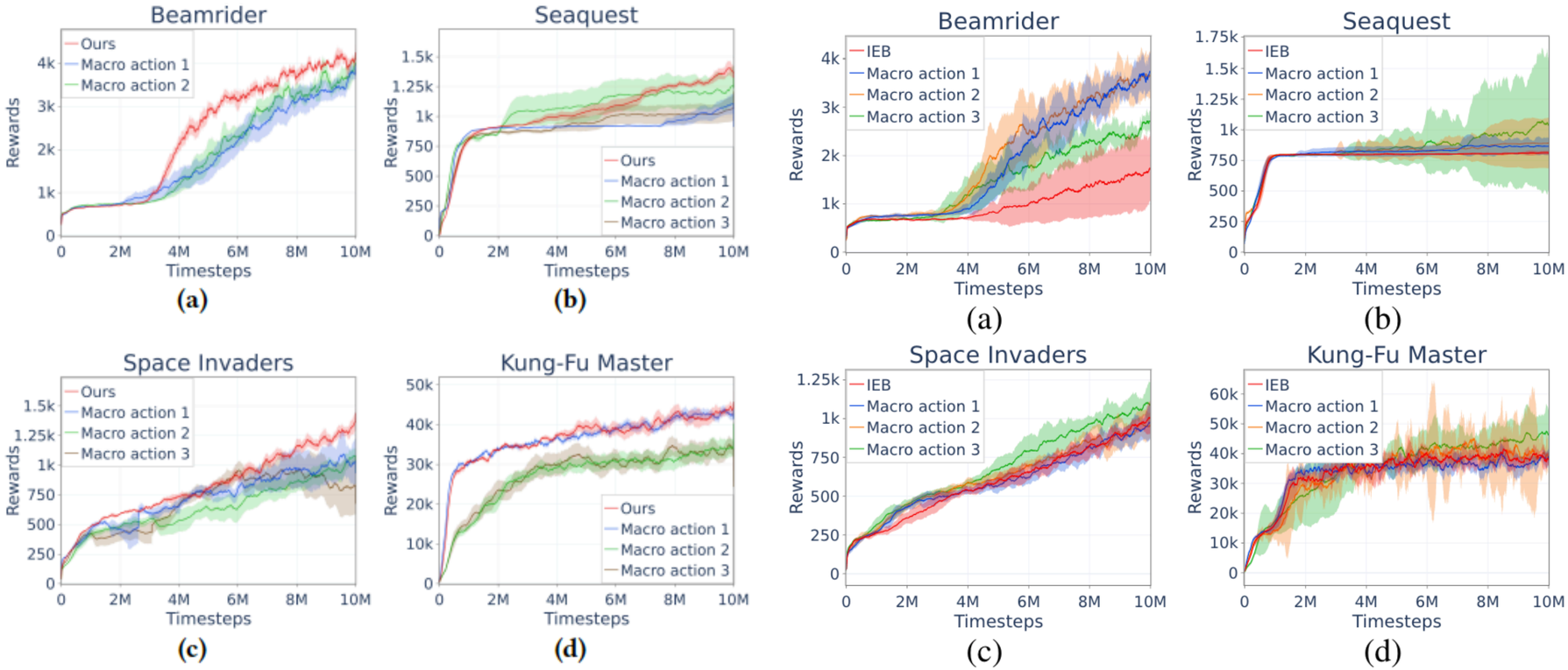
- **Comparison of our proposed methodology with the primitive action baseline**
  - Ours (red line) represents the performance of an agent trained with the macro ensemble constructed by our method.
  - Primitive action (green line) represents the performance of an agent trained with the basic primitive actions.
  - The figure shows that our proposed method is qualified for constructed superior macro ensembles.

Comparison against Online Macro Search



- **Comparison of our proposed methodology with macro ensembles searched by an online macro ensemble search algorithm called iterative experience based (IEB) method**
  - Ours (red line) represents the performance of an agent trained with the macro ensemble constructed by our method.
  - IEB (blue line) represents the performance of a an agent trained with the macro ensembles constructed by IEB.
  - The figure verify the advantage of a macro action ensemble over IEB.

Ensemble vs. Decoupled Macros



Our method **IEB**

- **Comparison of e discovered by our methodology with the decoupled macro actions.**
  - Ours (red line) represents the performance of an agent trained with the macro ensemble constructed by our method.
  - Macro 1 (blue) Macro 2 (green) Macro 3 (brown) in (b)(c)(d) represent the performance of each macro actions in the macro ensemble searched in Seaquest, SpaceInvaders and Kung-Fu Master respectively.
  - The higher performance of the ensemble compare to the relative lower performance of individual macro actions demonstrates the compatibility among the macro ensemble.

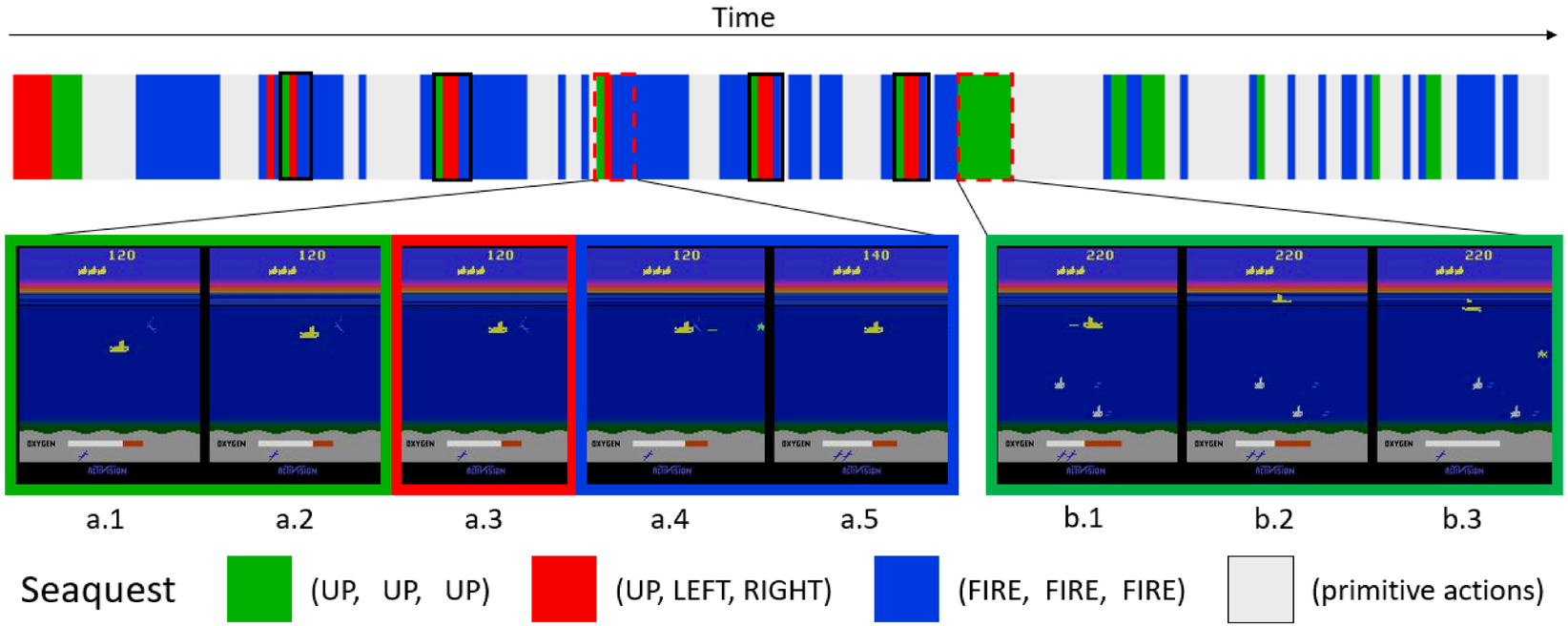
Evaluation Results

Atari 2600	Method	**Macro 1** m_1	**Macro 2** m_2	**Macro 3** m_3	$\max\{m_1, m_2, m_3\}$	**Ensemble Perf.**	**Improvement**
Asteroids	Ours	7105.89 (2432.10)	10180.60 (7857.42)	5589.20 (1049.35)	10180.60 (7857.42)	19685.48 (7421.24)	93.36%
	IEB	2187.89 (215.14)	2274.65 (133.90)	7620.44 (3946.70)	7620.44 (3946.70)	2349.21 (189.27)	-69.17%
Beamrider	Ours	3758.06 (405.62)	3996.31 (476.38)	N/A*	3996.31 (476.38)	4165.74 (372.66)	4.24%
	IEB	3739.30 (306.27)	3653.49 (405.27)	2699.69 (173.01)	3739.30 (306.27)	1722.42 (584.84)	-53.94%
Breakout	Ours	N/A†	N/A†	N/A†	N/A†	319.74 (42.36)	N/A†
	IEB	193.47 (36.36)	60.40 (15.38)	200.82 (44.37)	200.82 (44.37)	50.62 (3.92)	-74.79%
Crazy Climber	Ours	113472.61 (4190.81)	100104.19 (5012.19)	N/A*	113472.61 (4190.81)	111466.74 (3371.71)	-1.77%
	IEB	105351.48 (4350.37)	109049.90 (3399.09)	112424.02 (2533.33)	112424.02 (2533.33)	105851.57 (7493.50)	-5.85%
Frostbite	Ours	282.88 (11.95)	1594.20 (1488.34)	4010.67 (739.10)	4010.67 (739.10)	4298.52 (1151.88)	7.18%
	IEB	998.33 (1011.80)	308.29 (20.56)	283.63 (14.15)	998.33 (1011.80)	812.71 (747.94)	-18.59%
Gravitar	Ours	937.76 (103.74)	1011.37 (155.23)	N/A*	1011.37 (155.23)	1496.22 (254.70)	47.94%
	IEB	783.26 (50.80)	730.39 (119.08)	925.10 (68.26)	925.10 (68.26)	842.57 (138.18)	-8.92%
Kung-Fu Master	Ours	42485.00 (3921.55)	34240.31 (6655.82)	33829.44 (5635.99)	42485.00 (3921.55)	44453.20 (4955.30)	4.63%
	IEB	46255.84 (5966.10)	38848.04 (1184.16)	38914.63 (2347.86)	46255.84 (5966.10)	38236.28 (1803.30)	-17.34%
Seaquest	Ours	1105.22 (204.40)	1262.94 (399.28)	1072.03 (263.65)	1262.94 (399.28)	1362.75 (152.32)	7.90%
	IEB	865.28 (59.28)	891.97 (179.63)	1037.70 (461.58)	1037.70 (461.58)	811.42 (11.01)	-21.81%
Solaris	Ours	2880.61 (493.98)	2116.17 (274.59)	2366.28 (285.26)	2880.61 (493.98)	3236.40 (238.09)	12.35%
	IEB	2131.12 (366.63)	1873.44 (201.11)	2110.11 (251.30)	2131.12 (366.63)	1856.87 (213.70)	-12.87%
Space Invaders	Ours	713.95 (627.45)	1076.55 (131.65)	642.19 (521.62)	1076.55 (131.65)	1368.52 (159.05)	27.12%
	IEB	1086.10 (127.80)	952.24 (78.32)	974.43 (87.87)	1086.10 (127.80)	1006.45 (90.95)	-7.33%
Venture	Ours	0.00 (0.00)	0.00 (0.00)	N/A*	0.00 (0.00)	167.25 (334.51)	N/A‡
	IEB	0.00 (0.00)	0.00 (0.00)	12.57 (28.11)	12.57 (28.11)	0.00 (0.00)	-100.00%
Zaxxon	Ours	6688.42 (1173.89)	8746.27 (748.50)	8317.77 (551.52)	8746.27 (748.50)	8856.88 (817.74)	1.26%
	IEB	7249.23 (1037.50)	7975.56 (1629.89)	6840.86 (1574.92)	7975.56 (1629.89)	6657.54 (1686.32)	-16.53%

- **The above table summarizes the evaluation results of the RL agents**
  - The '*Average Perf.*' column represents the mean score of all the performances corresponding to the decoupled macros.
  - The *Synergism* exists among the macros if the score in the column of '*Ensemble Perf.*' is higher than the each score of decoupled macros in the same row.
  - It is observed that our method help construct macro ensembles that possess the *Synergism* property.

Qualitative Results

Timeline Analysis



- **History action sequences of Seaquest**
  - The top bar shows a trajectory in *Seaquest* with green representing three consecutive **UP** macro, red representing (**UP**, **LEFT**, **RIGHT**) macro, and blue representing three consecutive **FIRE** macro. The white segment represents the action sequences composed of primitive actions.
  - The macro actions and primitive actions are able to be jointly and interleave used by RL agents, which demonstrates the *Synergism* property empirically.

Outline

- Diversity-Driven Exploration
- Flow-Based Intrinsic Curiosity Module
- Adversarial Active Exploration for Inverse Dynamics Model Learning
- Macro Actions for Deep Reinforcement Learning
- **Mixture of Step Returns in Bootstrapped DQN**
- **Efficient Inference Technique**

Motivation

Motivation

Some problem

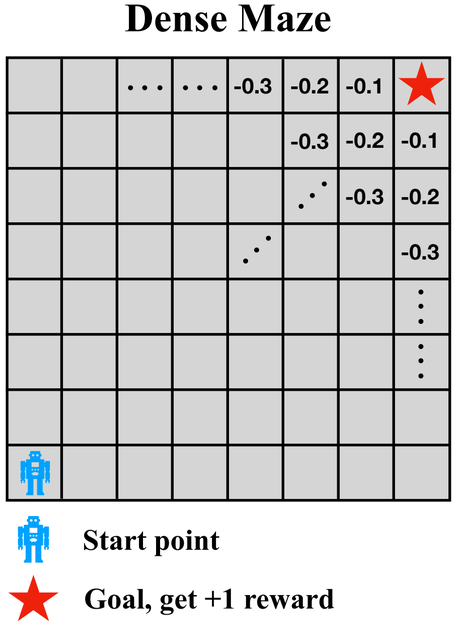
- Backup length n = ? is better.

$$y_{s,a} = R_t^n + \gamma^{n+1} \max_a Q(s_{t+1}, a, \theta^-)$$

Motivation

Behavior of different backup length — Maze environment

- In a simple maze environment

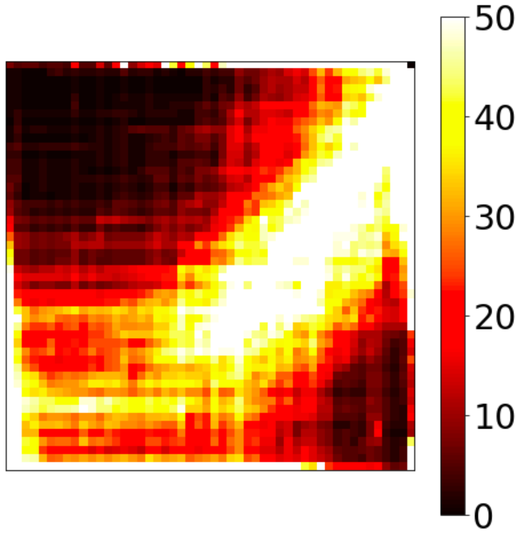


- Each episode, agent will start at left-down side

Motivation

Behavior of different backup length — Maze environment

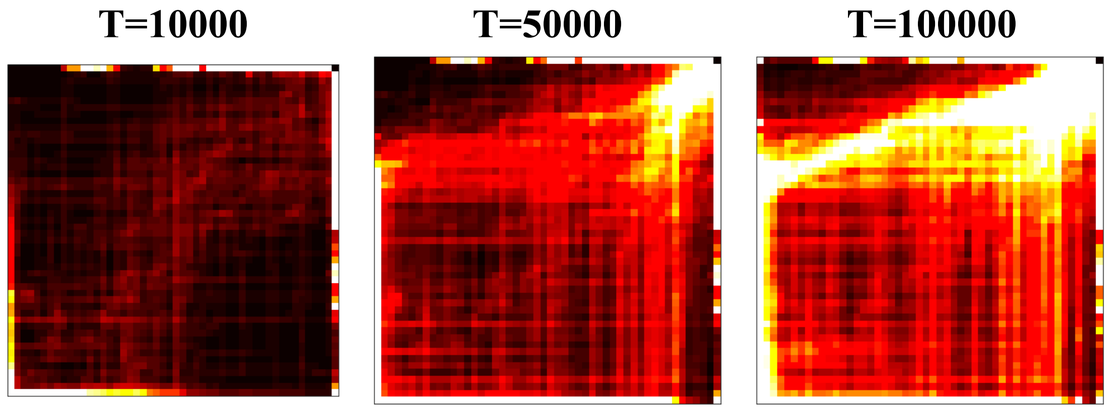
- DQN agent with 2 different step return as their target value
    - 1-step return
    - 5-step return
- Depict the states visited by the agents for 100k timesteps

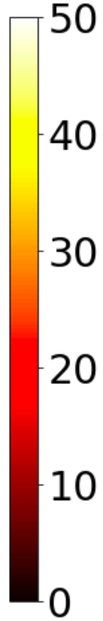


Motivation

Behavior of different backup length — Maze environment

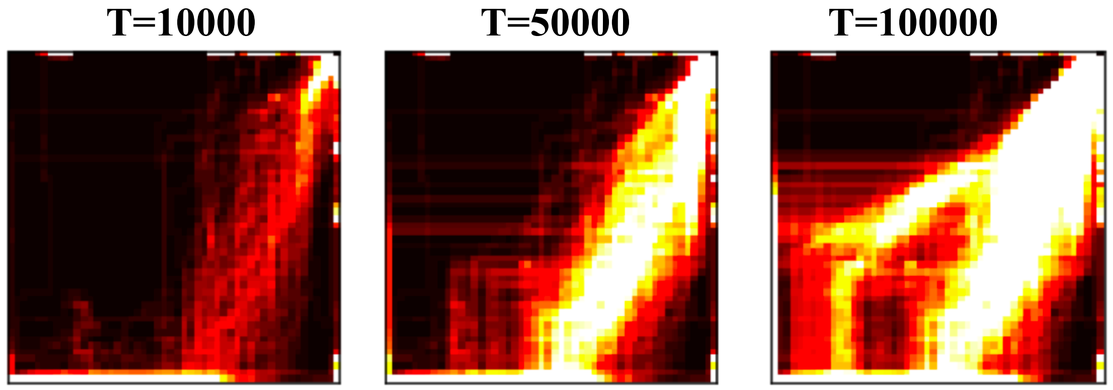
- 1-step





1. Converge slow
2. Visit more state

- 5-step



1. Fast learn a good policy
2. Visit less state

Motivation

Different backup length

- Agent behavior is very diverse with different target value

- Can we combine different n-step return?

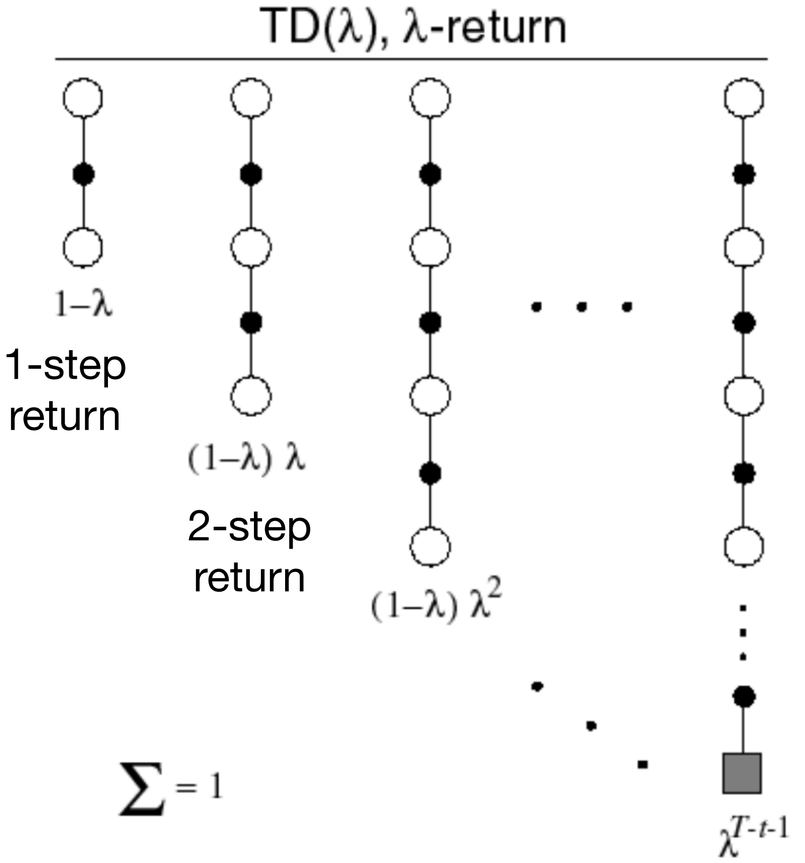
TD-lambda

TD-lambda

- An exponential average of all n-step return

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)}$$

- Use G_t^λ as the update target



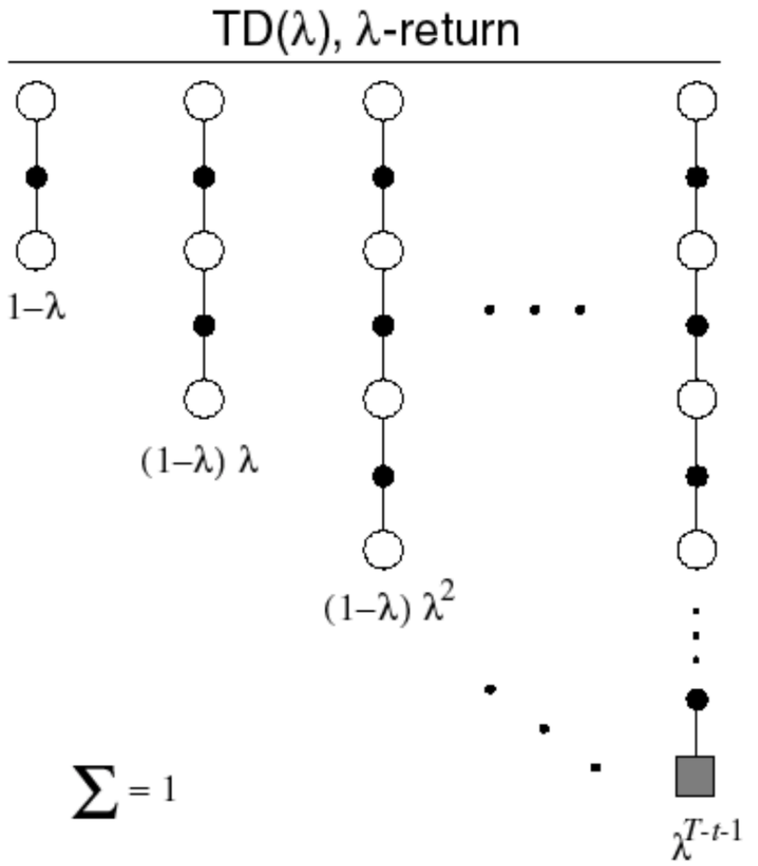
TD-lambda

TD-lambda

- Combine multiple distinct step returns,

  they still rely on a single target value

- Any method to combine?

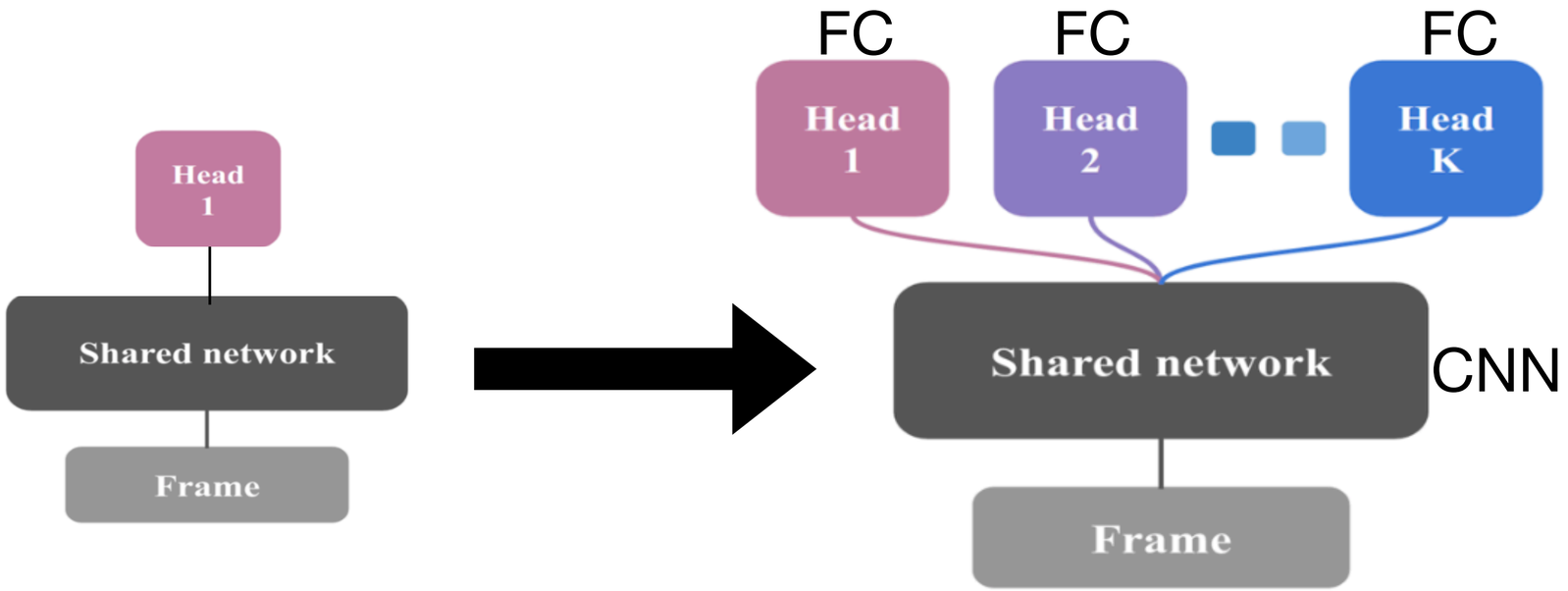


Methodology

Mixture Bootstrapped DQN

What is Bootstrapped DQN

- Bootstrapped DQN [1]

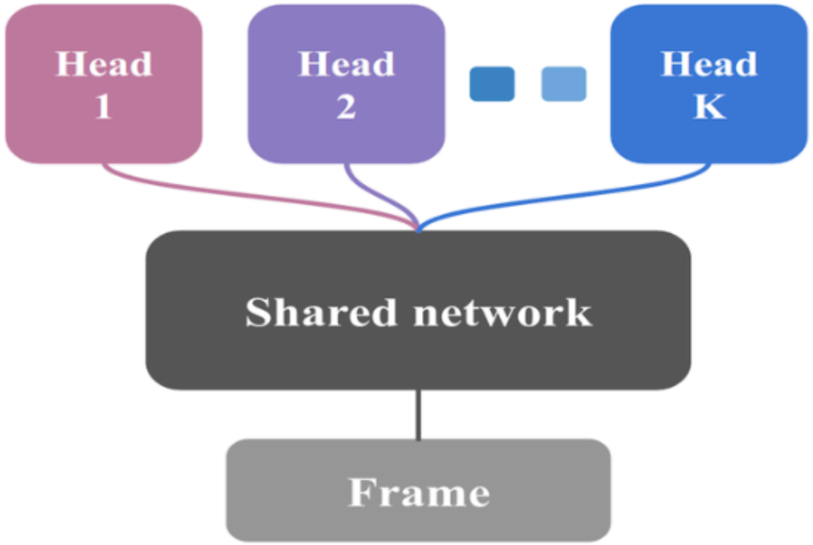


[1] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pages 4026–4034. 2016.

Mixture Bootstrapped DQN

Bootstrapped DQN

- K bootstrapped Q-value function heads
- Each episode re-samples a Q-value function head to use
- Collect data from different bootstrapped head makes agent explore well



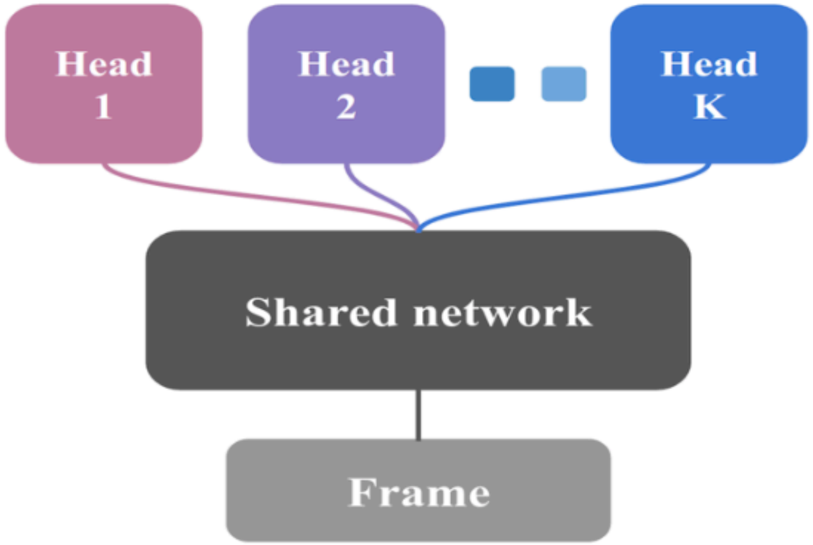
[1] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pages 4026–4034. 2016.

Mixture Bootstrapped DQN

Bootstrapped DQN problem

- Each head use same backup length
  (i.e. 1-step return)

- Might be lack in diversity and
  heterogeneity among the bootstrapped heads



[1] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, pages 4026–4034. 2016.

Mixture Bootstrapped DQN

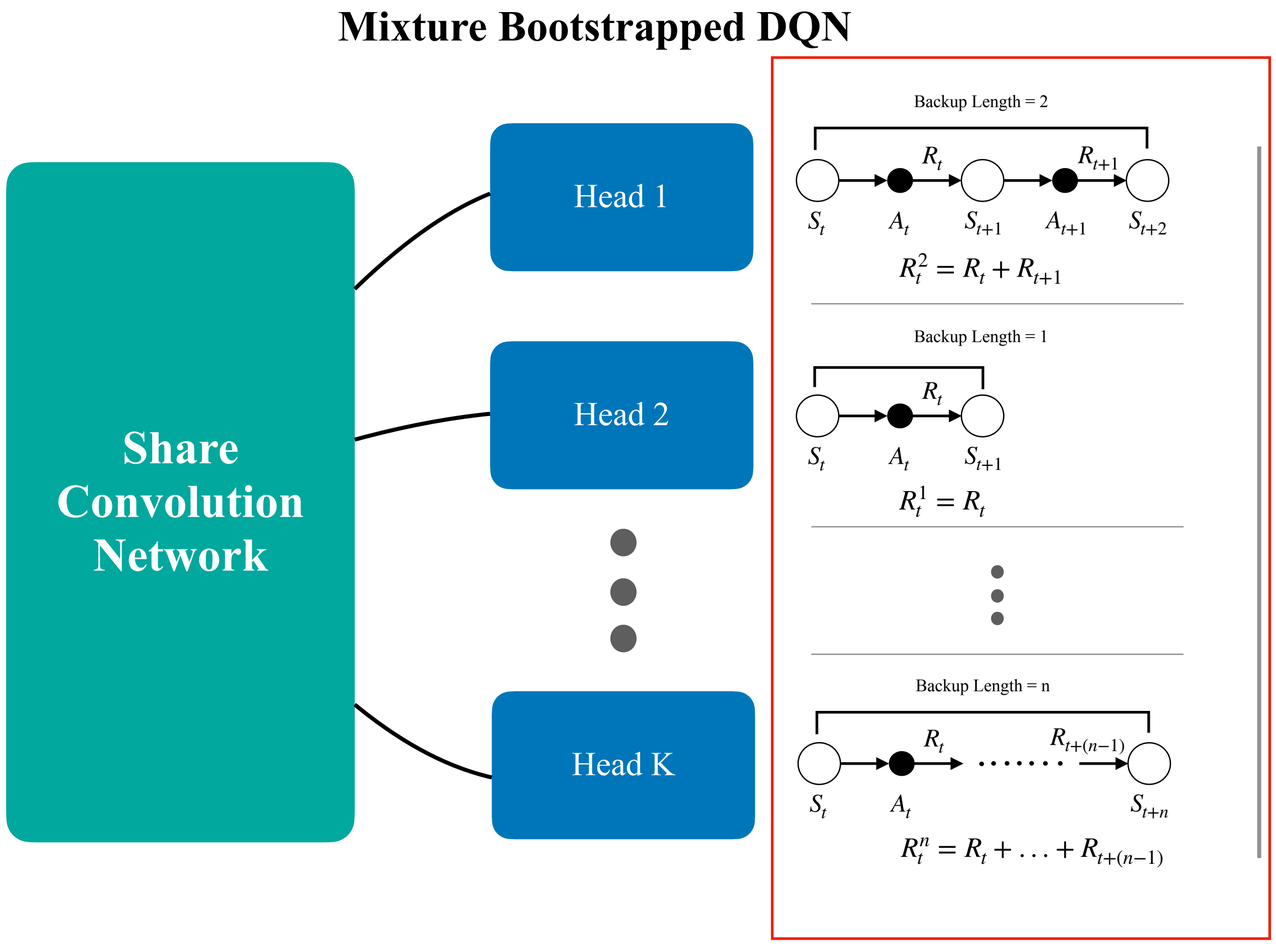
Mixture Bootstrapped DQN (MB-DQN)

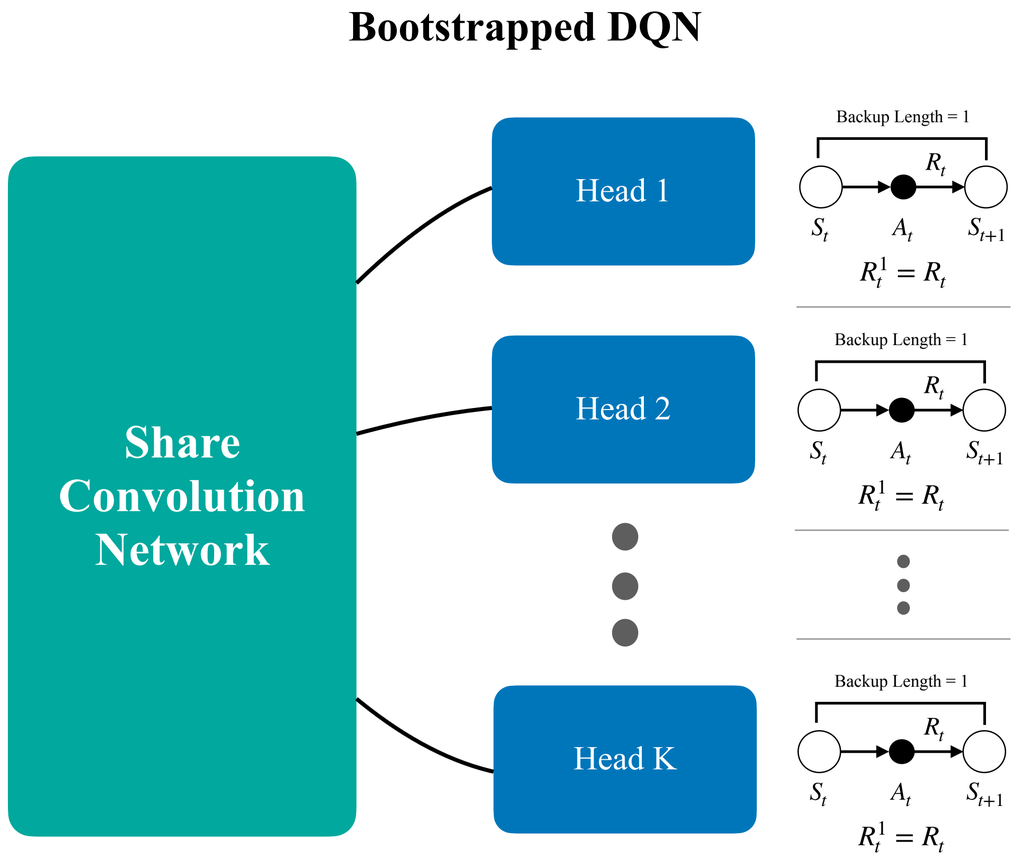
- Leverage the advantages of distinct Q-value function in Bootstrapped DQN

- Mixture different backup lengths for each bootstrapped different Q-value function heads

Mixture Bootstrapped DQN

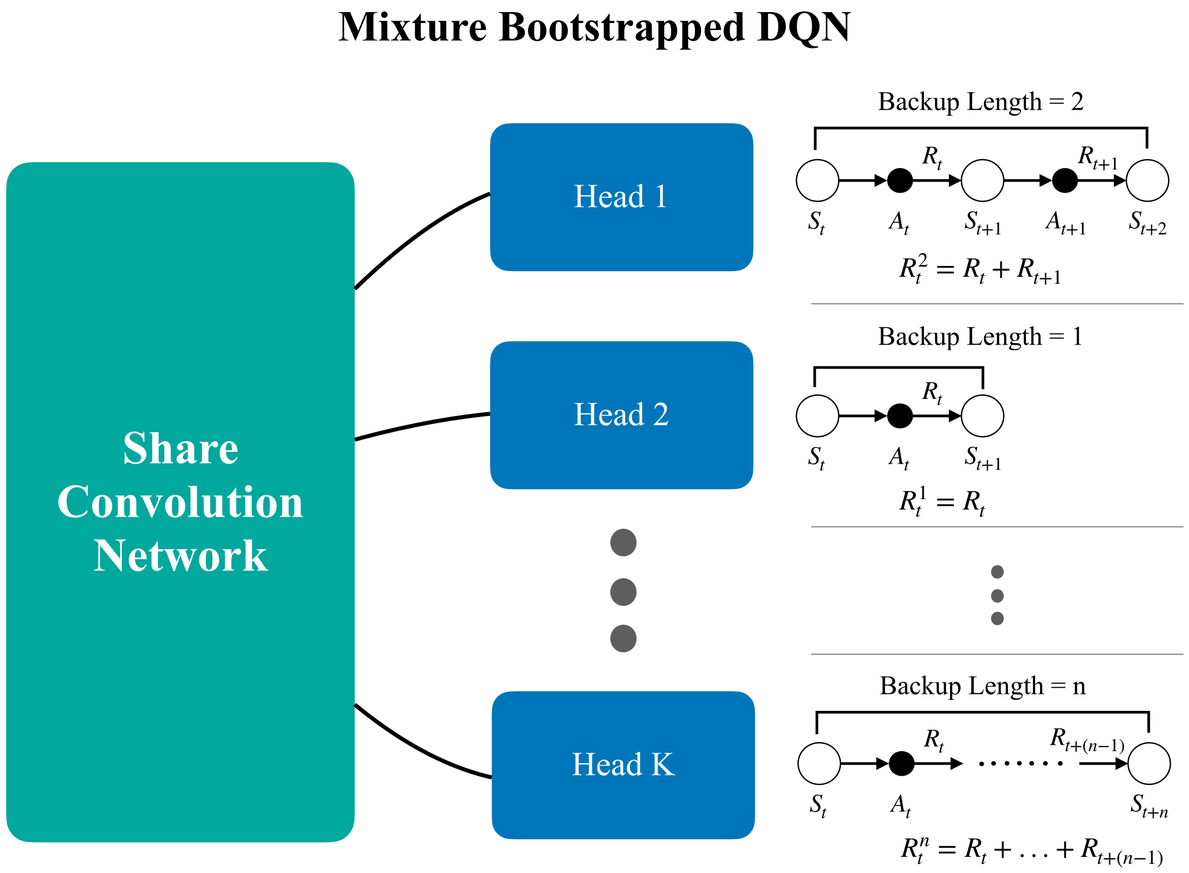
Mixture Bootstrapped DQN (MB-DQN)





Mixture Bootstrapped DQN

Mixture Bootstrapped DQN (MB-DQN)



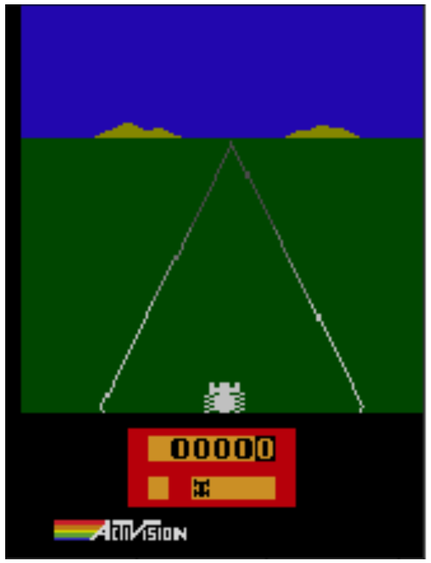
- Each head is trained with its own backup length n_k and target value
- Combine multiple distinct n-step returns
- Provide diversity and heterogeneity

Experimental Results

MB-DQN vs Bootstrapped DQN

Baselines and Environment

- Environment
  - 8 Atari games — *Breakout, Frostbite, Hero, Enduro, Qbert, Seaquest, CrazyClimber, and Freeway*



Enduro

Freeway

Seaquest

MB-DQN vs Bootstrapped DQN

Baselines and Environment

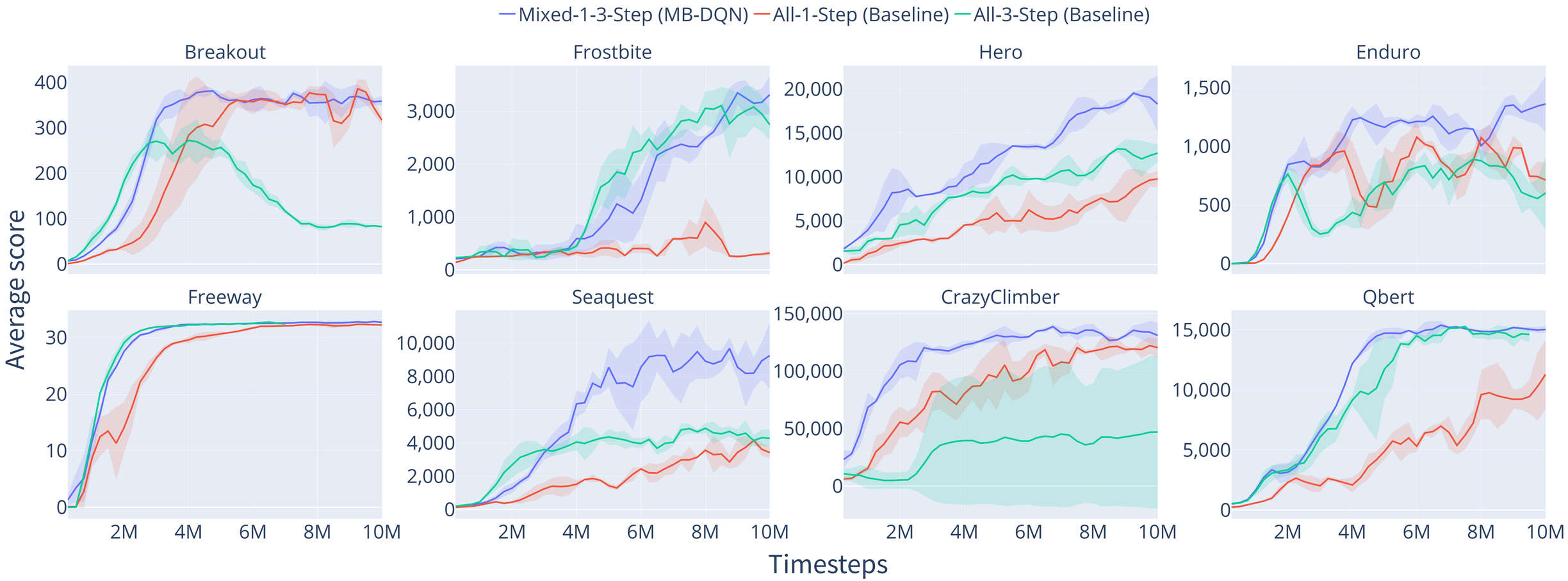
- Baseline
  - —Bootstrapped DQN with all 1-step return (All-1-step)
  - —Bootstrapped DQN with all 3-step return (All-3-step)

- Our Method
  - —MB-DQN (MB-DQN-1-3-step, 5x 1-step return, 5x 3-step return)

We use 10 bootstrapped heads in these method

MB-DQN vs Bootstrapped DQN

Evaluation Results



Outline

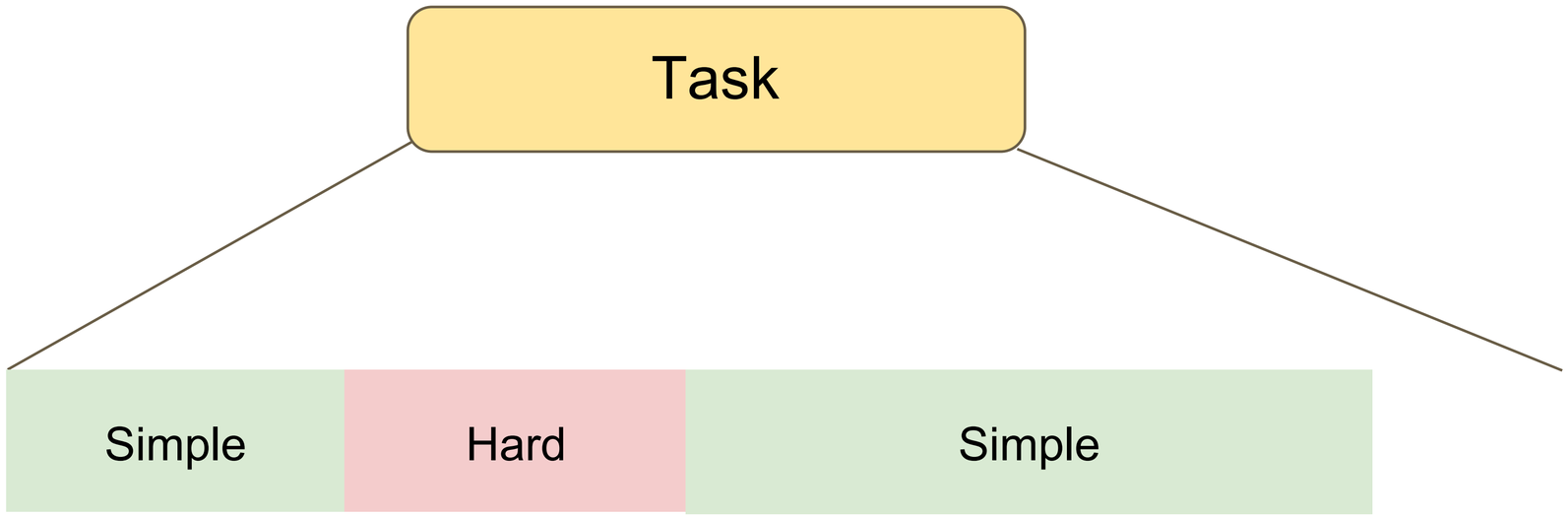
- Diversity-Driven Exploration
- Flow-Based Intrinsic Curiosity Module
- Adversarial Active Exploration for Inverse Dynamics Model Learning
- Macro Actions for Deep Reinforcement Learning
- Mixture of Step Returns in Bootstrapped DQN
- **Efficient Inference Technique**

Computational Cost-Aware Control Using Hierarchical Reinforcement Learning

- **Cost-aware reinforcement learning is necessary and important for autonomous systems**
  - Power consumption has long been an important issue due to the limitary battery
  - Even though low power embedded systems have been proposed, the amount of energy carried by each autonomous machine is limited
  - Previous deep learning works have been focused on low power perception modules (e.g., low power object detection or semantic segmentation)
  - Little attention has been paid to low power reinforcement learning (RL) methods

Motivation

- **A control task can usually be decomposed into segments of different levels of control complexity**
  - In the following figure, green parts correspond to simple segments, while red parts correspond to difficult segments



Task Segment Decomposition

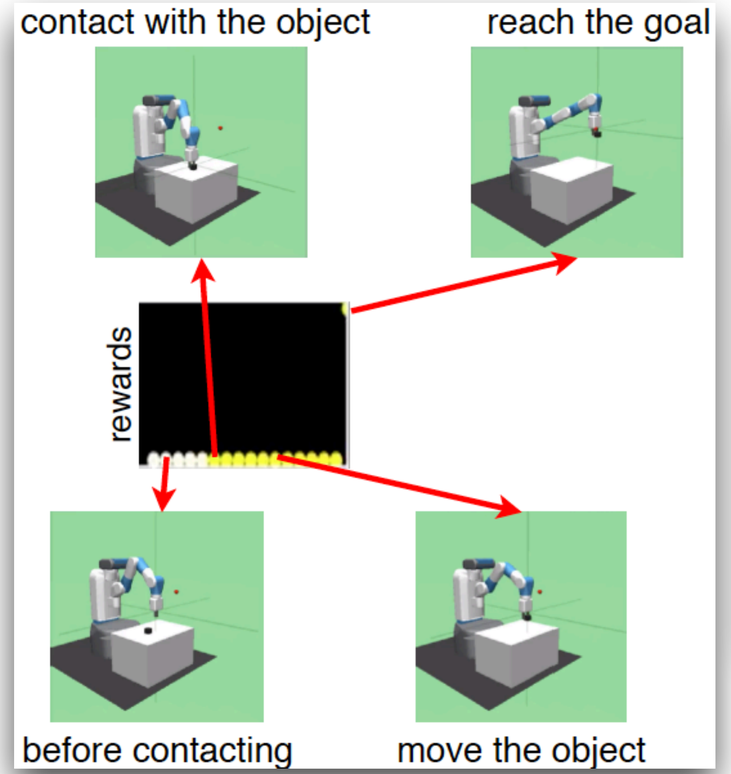
- **Motivation: different task segments have different task difficulties**
  - Assume that each task can be decomposed to multiple segments
  - Some segments do not require intensive computation to decide an action.
  - If these segments can be identified, different models with different complexity can be applied dynamically in different scenarios (i.e., task segments)

Driving on a straight road does not require a complex model

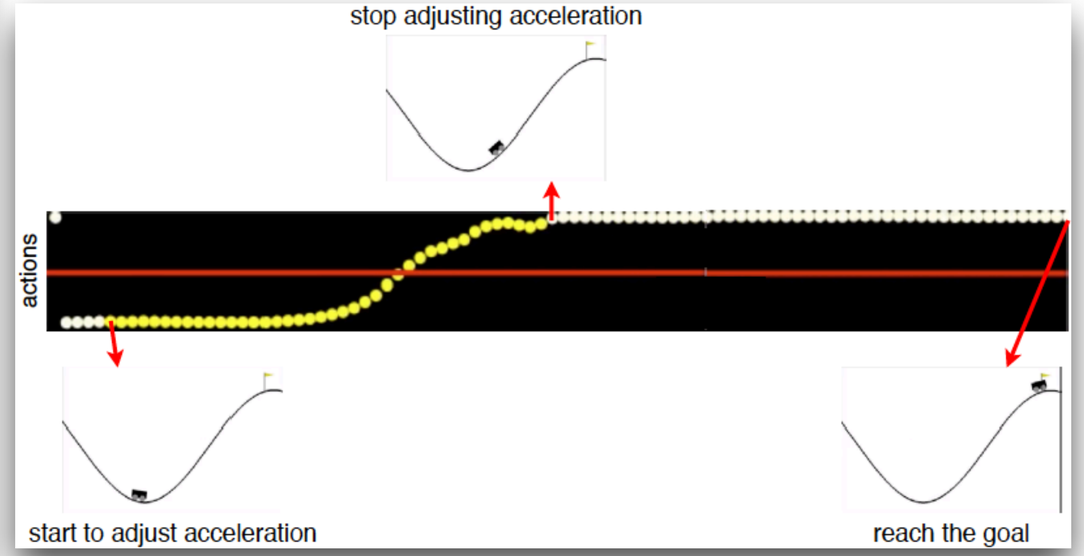
Drifting on a curvy road requires a very complex model

Examples of Task Segment Decomposition

- **Several examples are performed on the MuJoCo environments**
  - These examples demonstrate that robotic tasks can be decomposed to segments
  - The while ball denotes the simple model, while the yellow ball denotes the complex model



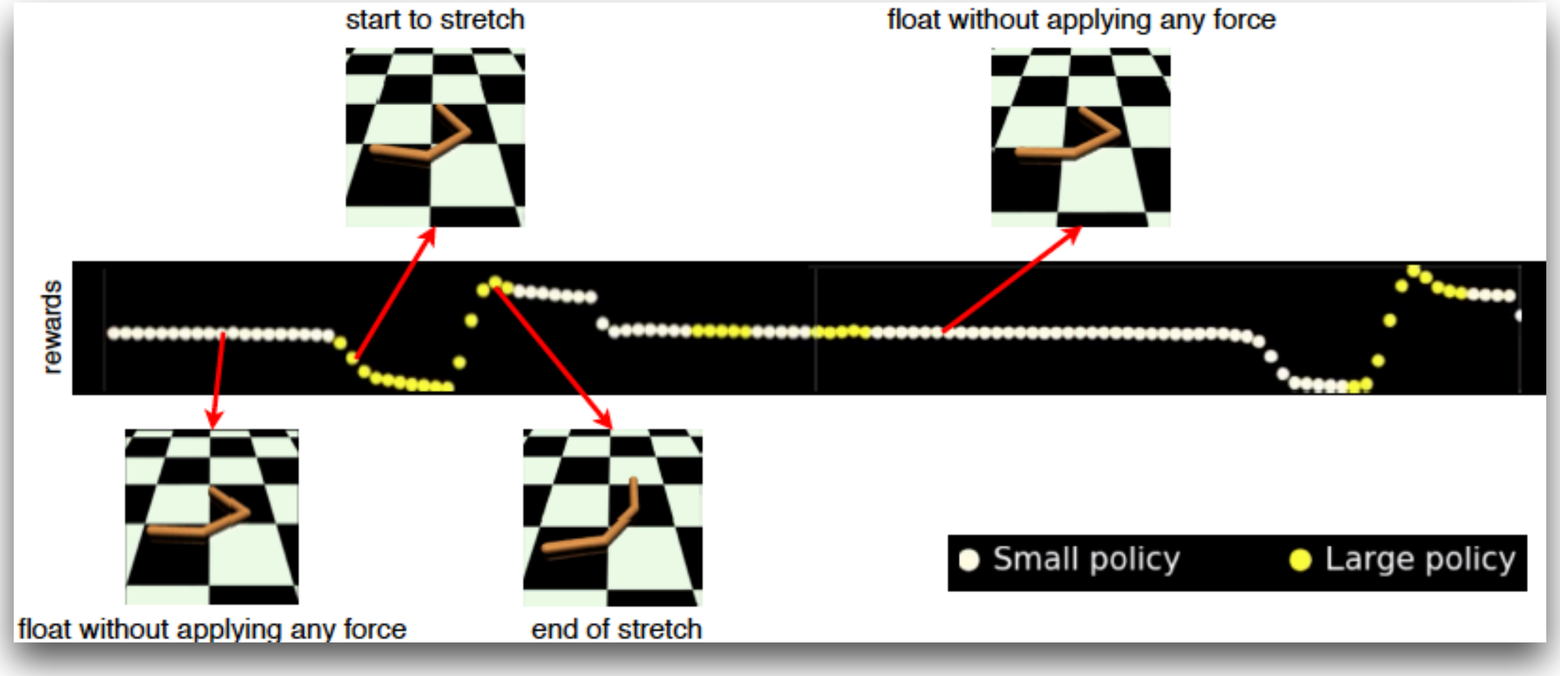
Robotic arm task



Mountain car task

Examples of Task Segment Decomposition

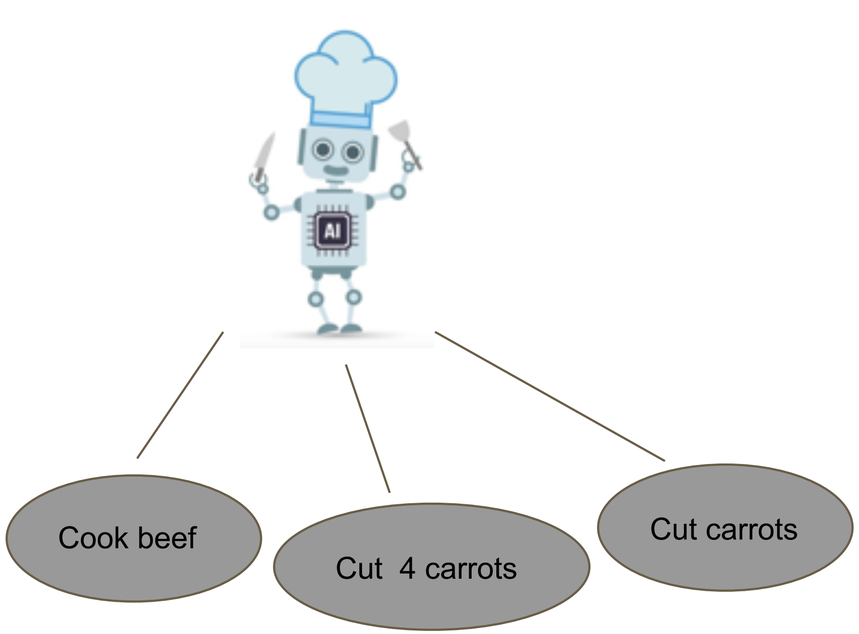
- **Several examples are performed on the MuJoCo environments**
  - These examples demonstrate that robotic tasks can be decomposed to segments
  - The while ball denotes the simple model, while the yellow ball denotes the complex model

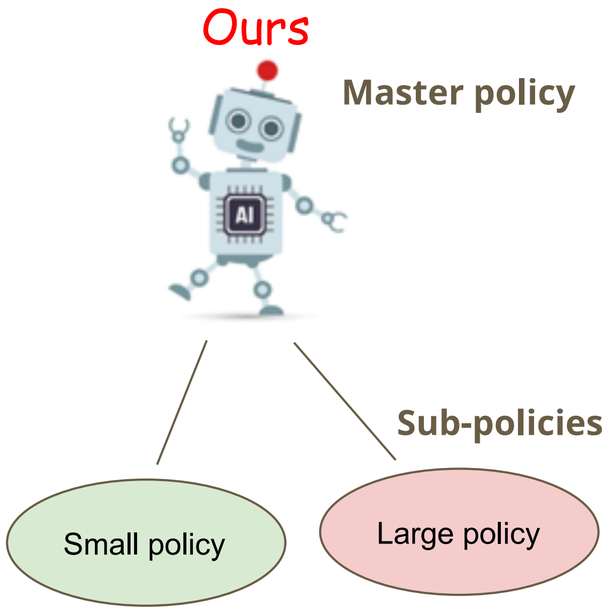


Swimmer task

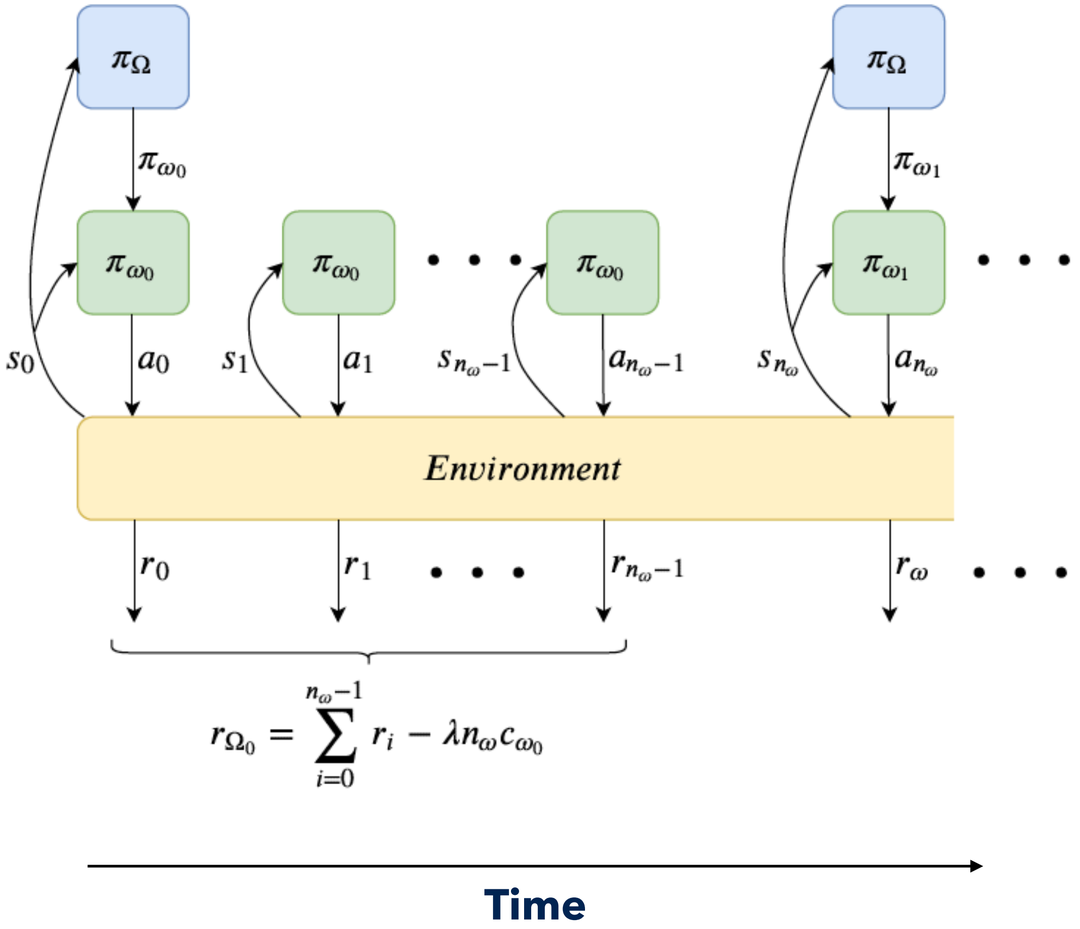
Background - Hierarchical RL

- **Hierarchical reinforcement learning (HRL) is a popular method in RL for handling complex control tasks**
  - HRL frameworks typically consists of a master policy and several slave policies





Model Architecture
Hierarchical Reinforcement Learning



- **Hierarchical reinforcement learning is adopted to implement the above idea**
  - The system consists of three policies
    - A master policy (π_Ω)
    - A small sub-policy ($\pi_{\omega\text{-small}}$)
    - A large sub-policy ($\pi_{\omega\text{-large}}$)
  - The master policy selects an sub-policy to use according to the observed state
  - The objective is to reduce FLOPs while maintaining performance

Pseudo Codes of the Proposed HRL

Algorithm 1: Computational cost-aware control using hierarchical reinforcement learning

input : total training steps T_{max}, environment E

Initialize master policy π_Ω and sub-policies $\pi_{\omega_{small}}, \pi_{\omega_{large}}$
Initialize global step counter and episode step counter: $T \leftarrow 0, t \leftarrow 0$
Initialize experience replay buffer for π_Ω and π_ω: $\mathcal{D}_\Omega \leftarrow \{\}, \mathcal{D}_\omega \leftarrow \{\}$
Get initial state s from environment E
while $T < T_{max}$ **do**
    if $t \bmod n_\omega == 0$ **then** ▷ select a sub-policy every n_ω timesteps
        choose ω according to $\pi_\Omega(\omega|s)$
        $s_\Omega \leftarrow s$
        $r_\Omega \leftarrow 0$
    choose a according to $\pi_\omega(a|s)$ ▷ act according to the chosen sub-policy
    take action a in s, observe state s', reward r, and terminal signal d
    $r_\Omega \leftarrow r_\Omega + r - \lambda c_\omega$ ▷ penalize master policy with weighted computational cost
    $t \leftarrow t + 1$

    if $t \bmod n_\omega == 0$ *or* $d == True$ **then** ▷ add transitions to replay buffer
        $\mathcal{D}_\Omega \leftarrow \mathcal{D}_\Omega \cup \{(s_\Omega, \omega, r_\Omega, s', d)\}$
    $\mathcal{D}_\omega \leftarrow \mathcal{D}_\omega \cup \{(s, a, r, s', d)\}$

    if d == *True* **then** ▷ reset the environment when an episode ends
        $t \leftarrow 0$
        reset environment and get a new state s

    if $\mathcal{D}_\Omega$ *and* $\mathcal{D}_\omega$ *have enough samples* **then** ▷ update master policy and sub-policies
        update π_Ω using sampled batches from $\mathcal{D}_\Omega$
        update both $\pi_{\omega_{small}}$ and $\pi_{\omega_{large}}$ using sampled batches from $\mathcal{D}_\omega$

    $T \leftarrow T + 1$

- **The system was trained such that the cost is taken care into consideration**
  - Separate replay buffers are used for the master policy and the sub-policies
  - Samples are stored into the replay buffer of the master policy every n_ω time steps
  - Samples are stored to the replay buffer of the sub-policies every time step
  - The master policy and the sub-policies are updated based on the batched sampled from the replay buffers

Experimental Results

- **Number of neurons per layer**
  - The small networks are adjusted such that their sizes are less than or equal to 1/4 of those of the large networks
  - Their performances are less than 30% of the criterion network

Environment	$\pi_{\omega_{small}}$		$\pi_{\omega_{large}}$		$\pi_{criterion}$	
	n_{units}	Scores	n_{units}	Scores	n_{units}	Scores
MountainCarContinuous-v0	8	-26.9 ± 13.7	64	87.8 ± 4.8	512	92.4 ± 5.1
BipedalWalker-v3	64	110.5 ± 146.8	256	297.5 ± 12.0	512	290.0 ± 14.9
HalfCheetah-v3	8	1139.8 ± 948.4	64	8126.0 ± 736.6	512	7412.0 ± 1442.7
Swimmer-v3	8	27.9 ± 5.9	256	66.4 ± 13.7	512	56.3 ± 6.9
Ant-v3	64	1246.8 ± 447.0	256	2687.4 ± 747.6	512	2962.0 ± 764.0
Walker2d-v3	64	2282.6 ± 290.6	256	4156.6 ± 148.0	512	4202.0 ± 343.8
FetchPush-v1	8	0.094 ± 0.009	64	0.970 ± 0.015	512	0.972 ± 0.011
FetchSlide-v1	64	0.100 ± 0.068	256	0.726 ± 0.053	512	0.666 ± 0.063
FetchPickAndPlace-v1	32	0.212 ± 0.149	128	0.948 ± 0.022	512	0.970 ± 0.010
walker-stand	8	289.0 ± 47.5	64	891.3 ± 143.2	512	915.7 ± 91.2
finger-spin	8	47.1 ± 75.8	64	928.4 ± 31.7	512	862.6 ± 88.1
cartpole-swingup	8	125.0 ± 74.1	64	819.8 ± 24.1	512	810.6 ± 50.5
ball_in_cup-catch	8	128.6 ± 52.5	64	970.0 ± 13.2	512	948.0 ± 29.2
hopper-stand	64	223.8 ± 165.9	256	611.6 ± 277.0	512	640.1 ± 230.5
fish-swim	8	74.6 ± 2.1	256	206.0 ± 36.5	512	193.1 ± 44.5
reacher-easy	8	166.5 ± 83.2	64	948.5 ± 18.8	512	954.0 ± 37.2

Experimental Results

- **Cost term**
  - The cost of the large networks are normalized to those of the small networks

Environment	$c_{\omega_{small}}$	$c_{\omega_{large}}$	λ
MountainCarContinuous-v0	1.0	44.7	1e−4
BipedalWalker-v3	1.0	12.0	1e−4
HalfCheetah-v3	1.0	20.0	8e−2
Swimmer-v3	1.0	424.8	1e−4
Ant-v3	1.0	8.0	1e−1
Walker2d-v3	1.0	12.3	3e−2
FetchPush-v1	1.0	17.5	8e−4
FetchSlide-v1	1.0	11.5	5e−5
FetchPickAndPlace-v1	1.0	9.4	2e−4
walker-stand	1.0	18.1	1e−2
finger-spin	1.0	29.1	1e−2
cartpole-swingup	1.0	37.4	3e−3
ball_in_cup-catch	1.0	30.1	1e−3
hopper-stand	1.0	12.8	8e−4
fish-swim	1.0	220.0	1e−4
reacher-easy	1.0	32.6	2e−3

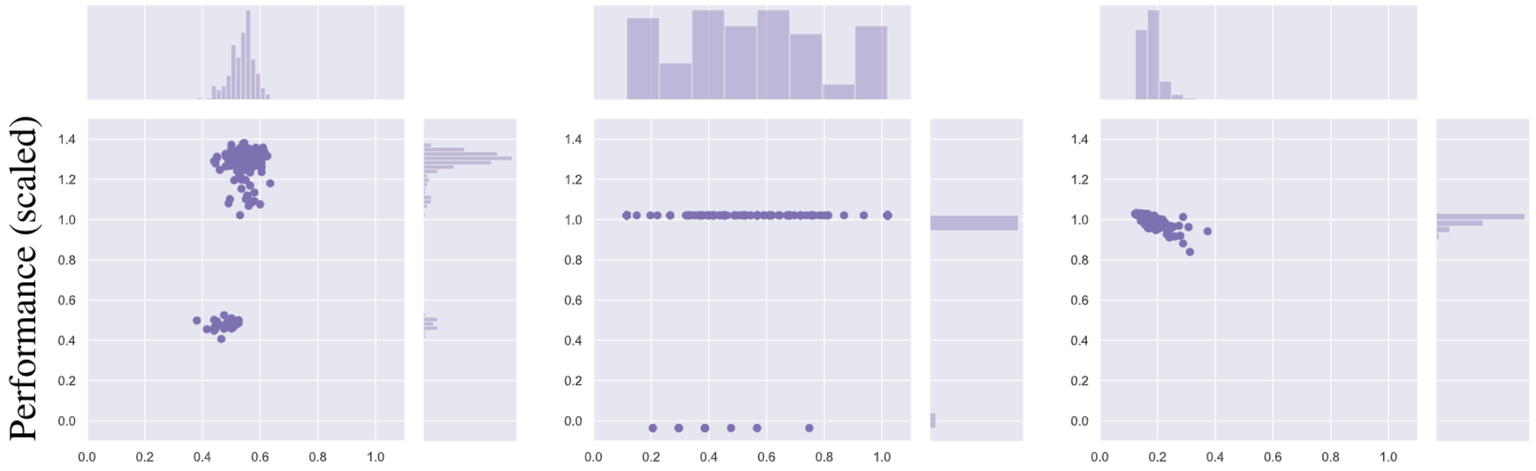
Experimental Results

- **Performance v.s. Cost**
  - Comparison of the best model selected from five training rounds with different random seeds
  - We show the performance comparison for the average scores and the best scores

Environment	π_{S-only}	π_{L-only}	*Ours (average)*	% using $\pi_{\omega_{large}}$	% Total FLOPs Reduction	*Ours (best)*	% using $\pi_{\omega_{large}}$	% Total FLOPs Reduction
BipedalWalker-v3	110.5 ± 146.8	297.5 ± 12.0	281.7 ± 24.6	$68.5\% \pm 5.1\%$	$28.3\% \pm 4.7\%$	309.1	67.6%	29.8%
HalfCheetah-v3	$1,139.8 \pm 948.4$	$8,126 \pm 736.6$	$7,483.4 \pm 1,170.1$	$99.4\% \pm 0.9\%$	$-6.0\% \pm 0.9\%$	8,388.2	100.0%	-6.6%
Walker2d-v3	$2,282.6 \pm 290.6$	$4,156.6 \pm 148.0$	$4,059.7 \pm 340.7$	$77.3\% \pm 17.3\%$	$20.3\% \pm 15.9\%$	4,255.1	57.0%	39.0%
FetchPush-v1	0.094 ± 0.009	0.970 ± 0.015	0.826 ± 0.119	$74.1\% \pm 3.0\%$	$17.5\% \pm 2.8\%$	0.915	72.8%	18.7%
FetchSlide-v1	0.100 ± 0.068	0.726 ± 0.053	0.501 ± 0.148	$57.6\% \pm 27.8\%$	$38.1\% \pm 25.4\%$	0.595	59.0%	36.8%
Cartpole-swingup	125.0 ± 74.1	819.8 ± 24.1	817.5 ± 44.7	$55.8\% \pm 42.9\%$	$37.5\% \pm 41.7\%$	848.5	17.3%	75.0%
Ball_in_cup-catch	128.6 ± 52.5	970.0 ± 13.2	967.5 ± 3.4	$3.0\% \pm 0.6\%$	$88.0\% \pm 0.6\%$	964.5	2.5%	88.5%
Hopper-stand	223.8 ± 165.9	611.6 ± 277.0	534.9 ± 322.2	$50.3\% \pm 19.5\%$	$45.3\% \pm 17.9\%$	876.7	45.0%	50.2%
Fish-swim	74.6 ± 2.1	206.0 ± 36.5	201.5 ± 59.6	$50.2\% \pm 5.5\%$	$51.0\% \pm 7.5\%$	268.6	56.4%	42.8%
Reacher-easy	165.5 ± 83.2	948.5 ± 18.8	689.7 ± 272.7	$50.4\% \pm 28.5\%$	$42.4\% \pm 27.4\%$	931.7	60.0%	33.1%

Experimental Results

- **Statistics comparison**
  - We compare the performance and costs of three environments for 200 independent test runs
  - Each dot in the figures correspond to a test run
  - It can be seen that our method is able to reduce costs while maintaining performances



(a) *Swimmer-v3* (b) *FetchPickAndPlace-v1* (c) *walker-stand*

Experimental Results

- **Comparison with baselines**
  - We compare our method with two representative baselines: behavior cloning (BC) and generative adversarial imitation learning (GAIL)
  - The network sizes of the two baselines are adjusted such that their FLOPs/Inference are roughly the same as ours
  - The results indicates that under the same FLOPs/inference, our method outperforms the baselines, with less amount of training data required

Environment	π_{L-only}	FLOPs/Inf	*Ours*	Avg-FLOPs/Inf	π_{fit}	GAIL [14]	BC [13]	FLOPs/Inf
MountainCarContinuous-v0	93.6	8,707	93.5	4,440	90.5	-99.9	93.3	4,603
Swimmer-v3	84.1	137,219	**108.8**	76,019	66.2	63.2	59.7	76,763
Ant-v3	3,927.2	196,099	**3,564.8**	119,032	2,553.0	-15.6	1,373.7	119,451
FetchPickAndPlace-v1	0.980	42,755	0.935	22,917	0.920	0.078	0.153	23,223
walker-stand	977.7	12,803	**967.2**	2,266	819.5	596.6	159.1	2,397
finger-spin	978.0	9,859	**871.2**	6,162	848.1	536.8	7.6	6,303

ありがとうございます