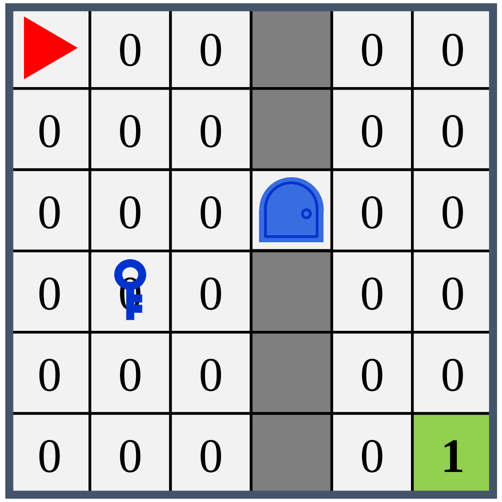
Highly Efficient Self-Adaptive Reward Shaping for Reinforcement Learning

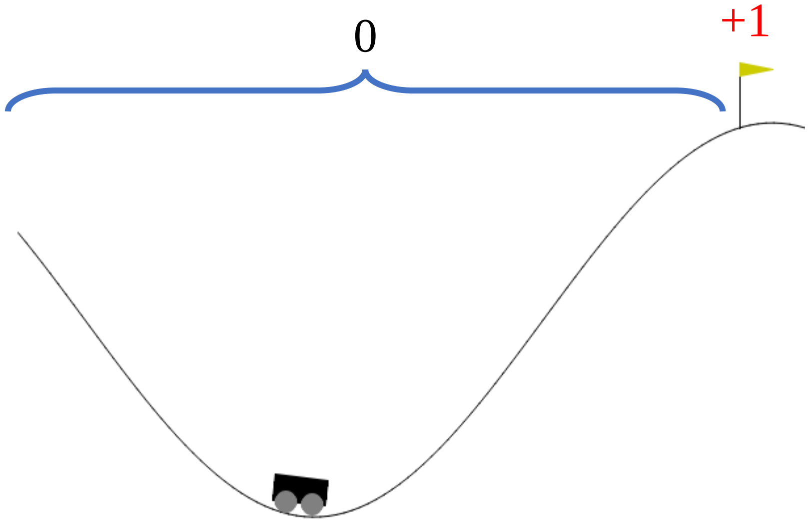
Haozhe Ma*[1], Zhengding Luo*[2], Thanh Vinh Vo[1], Kuankuan Sima[1], Tze-Yun Leong[1]

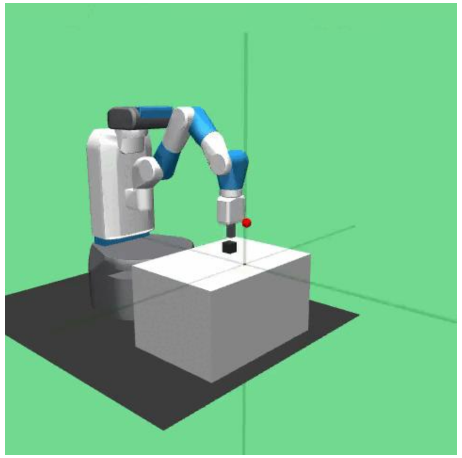
* Equal contribution, [1] National University of Singapore, [2] Nanyang Technological University
Corresponding: haozhe.ma@u.nus.edu

Sparse-Reward Challenge in Reinforcement Learning

- **Sparse and delayed** rewards: one of the main challenges in Reinforcement Learning.
  - Binary rewards: {0,1}.
  - Only indicating the completion of the overall task.
  - All 0s (no information) for all in-process states.
  - The agent cannot discriminate the values among different states, without getting any immediate feedback.
  - Common: usually don’t have any heuristics, prior knowledge or expertise to define an informative reward function.





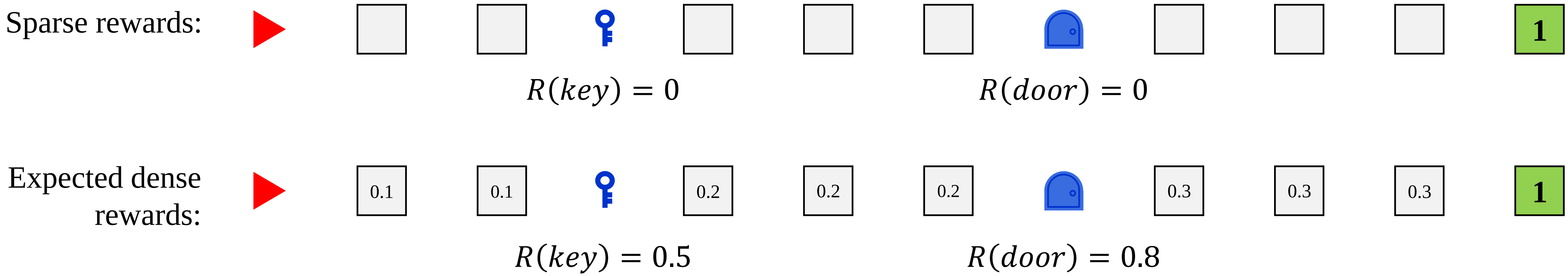


Reward Shaping

- **Reward Shaping (RS):** Reshape/rebuild the sparse environmental reward to more informative rewards.

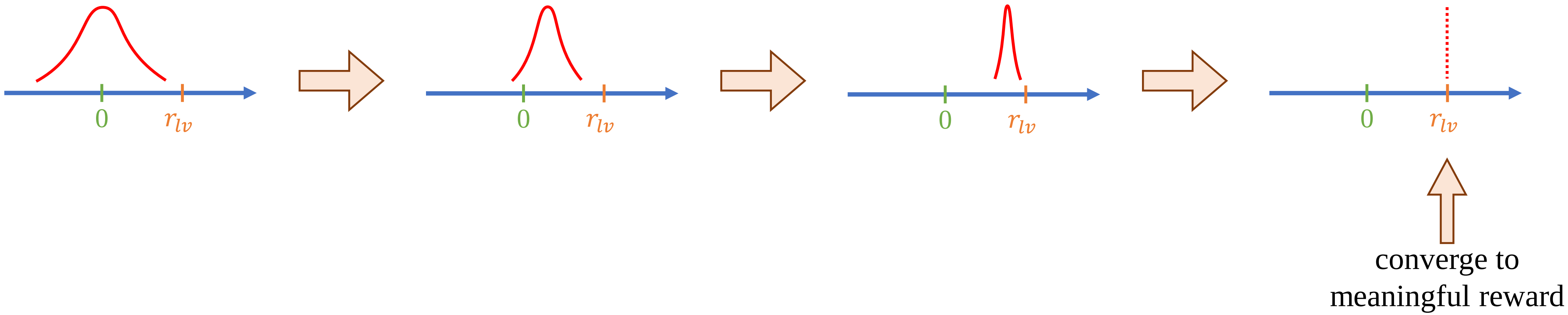
$$R^{new} = \alpha R^{env} + \beta \boldsymbol{R^{sha}}$$

- **Main problem**: how to find/learn/maintain a shaped reward $\boldsymbol{R^{sha}}$
  - Transfer/decompose/re-assign sparse rewards to dense rewards.



Inspiration from Previous Work

- Inspiration from a previous work *ReLara* [1]:
    - A mechanism to naturally balance exploration and exploitation from reward shaping perspective: designing shaped rewards evolving from stochastic to certain.
    - Converge to a meaningful reward.

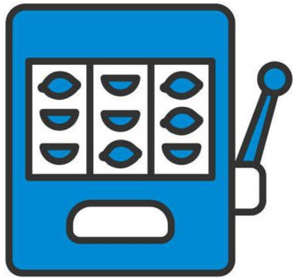


[1] Haozhe Ma, Kuankuan Sima, Thanh Vinh Vo, Di Fu, and Tze-Yun Leong. 2024. **Reward Shaping for Reinforcement Learning with An Assistant Reward Agent.** In Proceedings of the 41st International Conference on Machine Learning, PMLR, 33925–33939.

Thompson Sampling

- Beta distribution

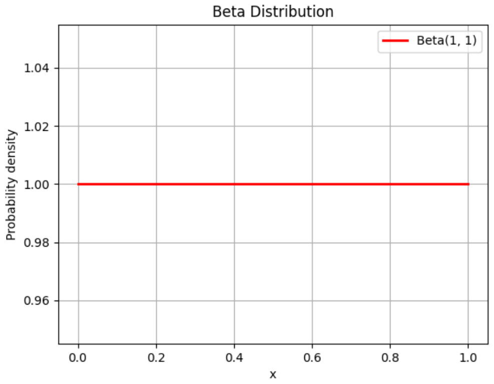
$P = 0.6$



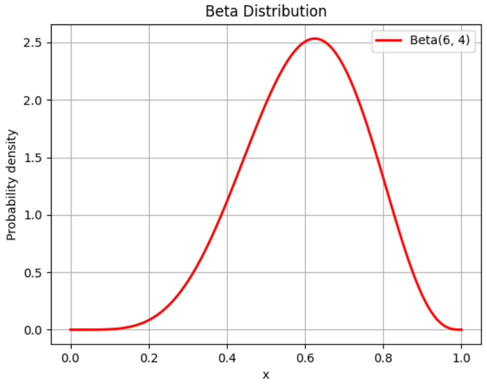
1. Tested 10 times, 6 of which succeed.
2. Tested 100 times, 60 of which succeed.
3. Tested 1000 times, 600 of which succeed.

The estimated probabilities of success are all 0.6, but the confidence is different.

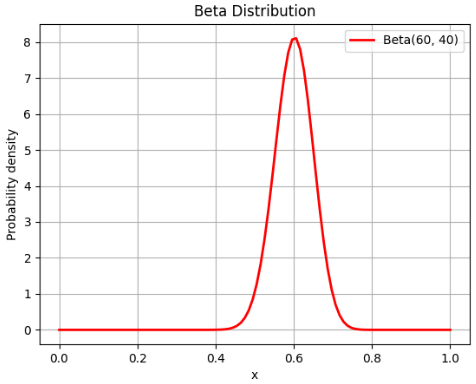
$$Beta(x;\alpha,\beta) = \frac{1}{B(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1}, \qquad \text{where } B(\alpha,\beta) = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$$



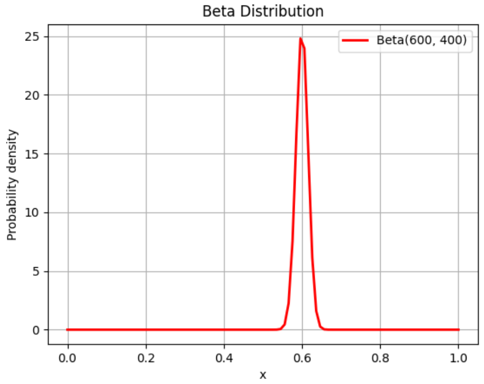
$Beta(0+1, 0+1)$



$Beta(6+1, 4+1)$



$Beta(60+1, 40+1)$

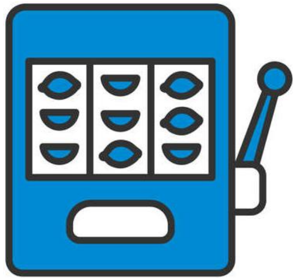


$Beta(600+1, 400+1)$

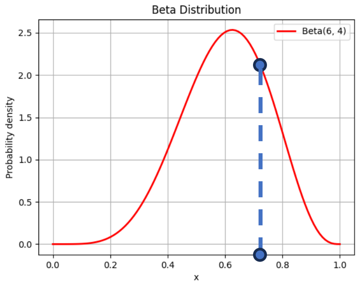
Thompson Sampling

- Thompson sampling is a commonly used algorithm in multi-armed bandit.
  - Balance the exploration-exploitation
  - Adaptively update the posterior distribution based on observations.

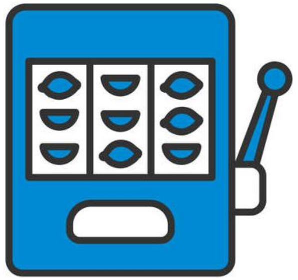
$P = 0.6$



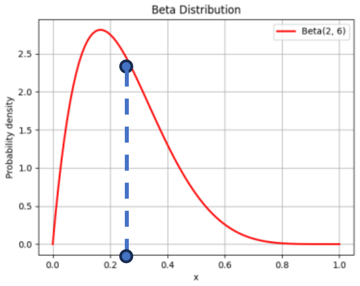
success	failure
6	4



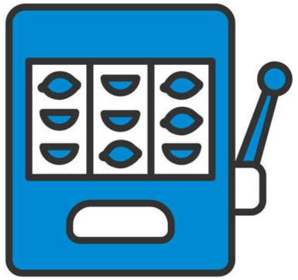
$P = 0.2$



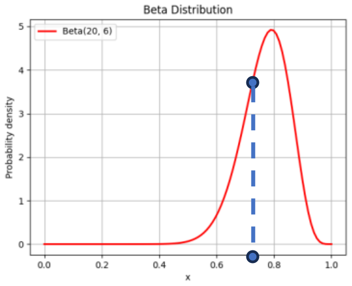
success	failure
2	6



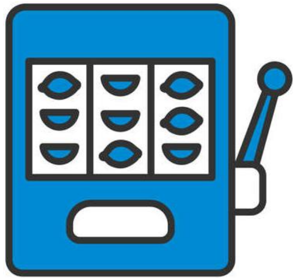
$P = 0.8$



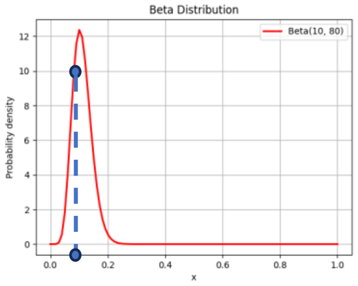
success	failure
20	6



$P = 0.1$

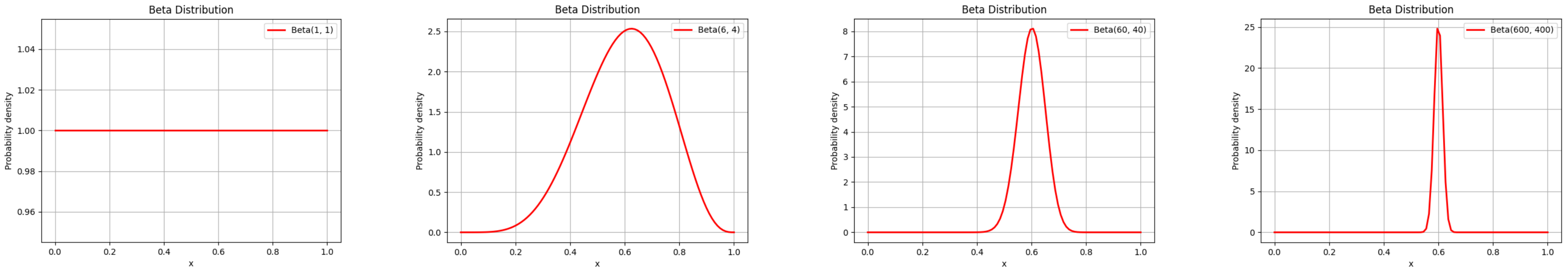


success	failure
10	80

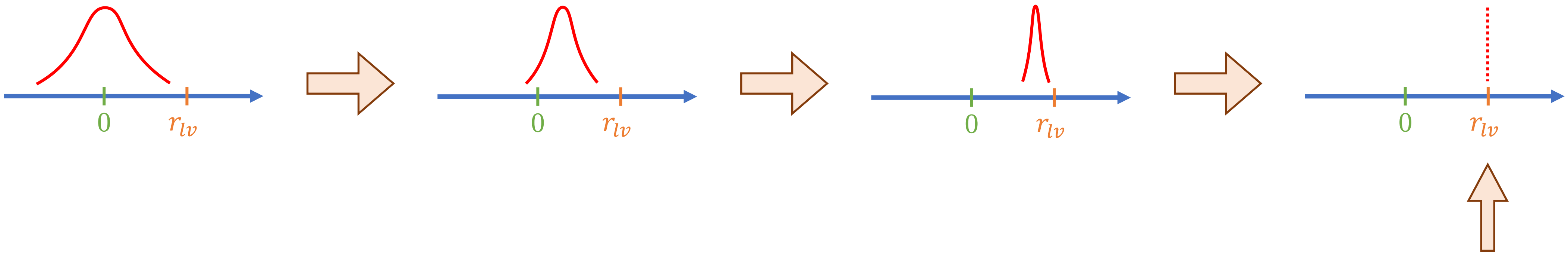


Thompson Sampling

- Beta distribution



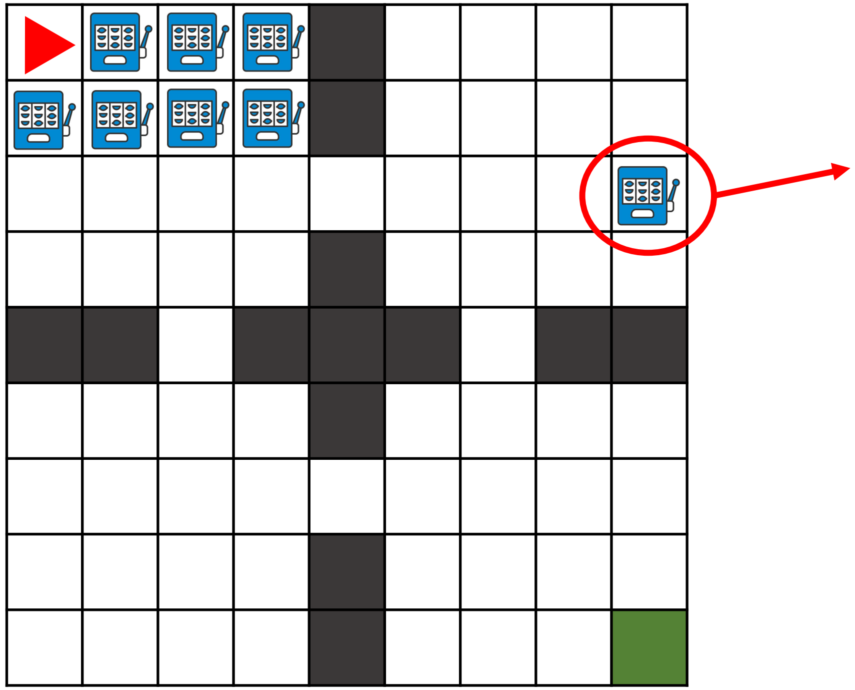
$Beta(0+1, 0+1)$ $\quad$ $Beta(6+1, 4+1)$ $\quad$ $Beta(60+1, 40+1)$ $\quad$ $Beta(600+1, 400+1)$



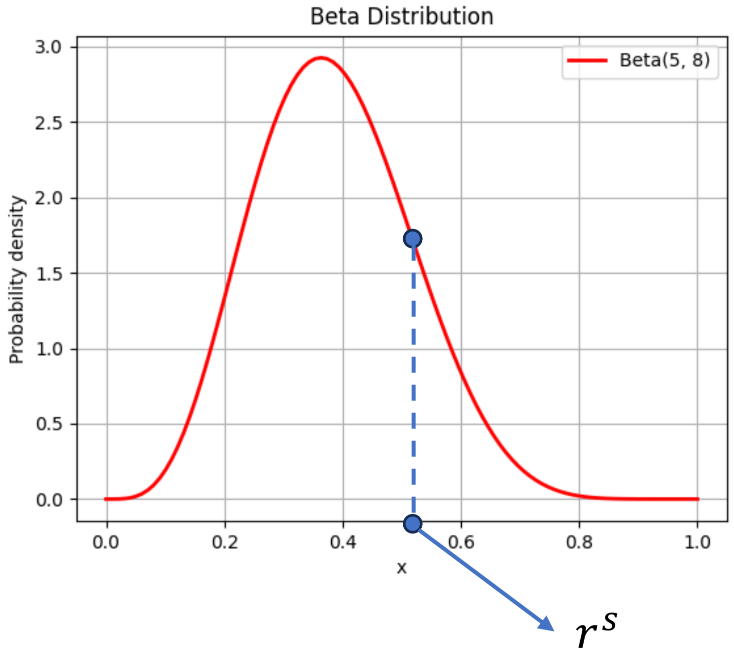
converge to meaningful reward

Self-Adaptive Success Rate (SASR) based Reward Shaping

- Assume there is a hidden bandit under each state, and the underlying success rate is the shaped reward.
- Success or Failure can be easily indicated by environmental sparse rewards.
- Proposed method: **S**elf-**A**daptive **S**uccess **R**ate (**SASR**) based reward shaping.

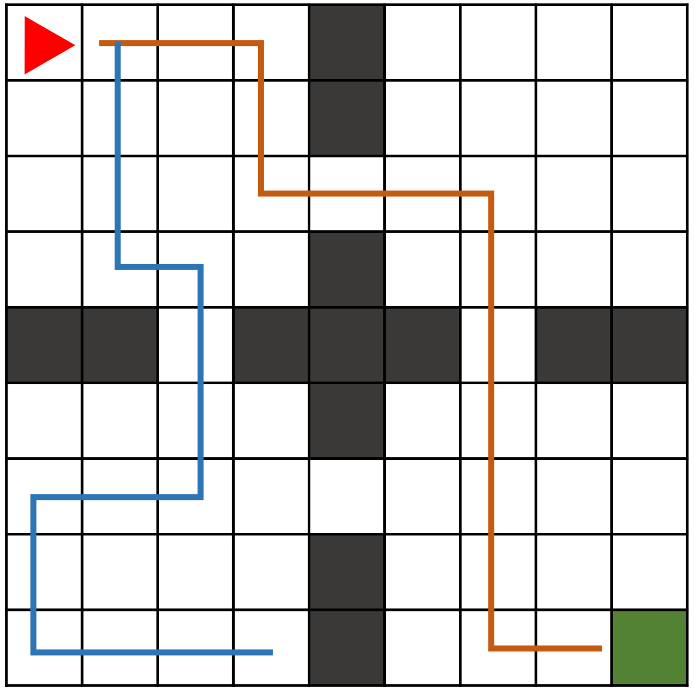


Success: 5
Failure: 8



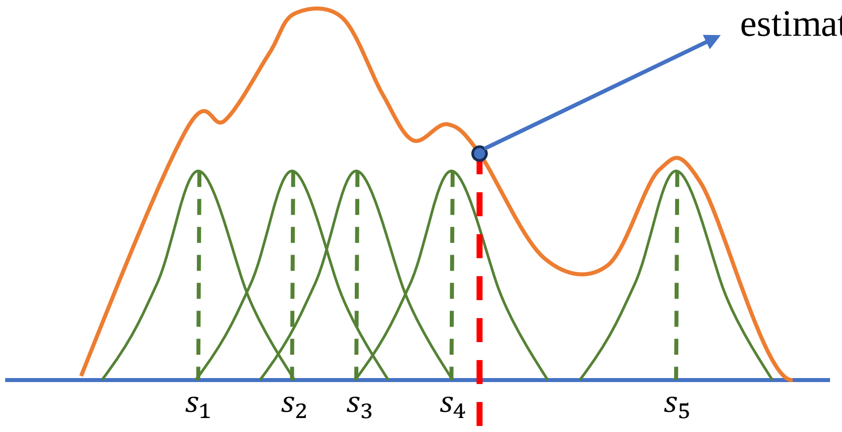
SASR in Tabular Environments

- If the state space is discrete with limited states:
  - Use a table to record the success and failure counts for each state
  - For each state, sample a shaped reward from a Beta distribution:
  $$r^s \sim Beta(S[s_i], F[s_i])$$
  - The counts can be updated after each trajectory



SASR in High-Dimensional Continuous State Space

- If the state space is high-dimensional, continuous with unlimited states.
- Idea: maintain a Beta distribution for each state:
  - However, we require the counts for success and failure for each state
  - We can't count and record by tabular approach.
- To estimate the counts, use Kernel Density Estimation (KDE)
  - Estimate the density given some data points.
  - Non-parametric approach.
  - An example of KDE using Gaussian kernel function:



estimated density f $\times$ total number N = estimated "count"

$$f(\hat{s}) = \frac{1}{Nh}\sum_{i=i}^{N} K\left(\frac{\hat{s} - s_i}{h}\right)$$

where h is the bandwidth, $K(\cdot)$ is the kernel function.

RFF for KDE

- Estimate counts by Kernel Density Estimation (KDE):
  - Non-parametric approach, avoiding learning models/networks.

$$f(\hat{s}) = \frac{1}{Nh}\sum_{i=i}^{N} K\left(\frac{\hat{s} - s_i}{h}\right)$$

  - Limitation:
    - For each new sample, need to compute the kernel function for all samples in the buffer.
    - For Gaussian kernel, there is nonlinear calculation (exponent).
    - The states are usually high-dimensional.
    - Although can use GPU for vectorized calculation, but still very time consuming.

- Using Random Fourier Feature (RFF) [2] to estimate the Gaussian kernel.

[2] Ali Rahimi and Benjamin Recht. 2007. Random features for large-scale kernel machines. Advances in neural information processing systems 20, (2007).

RFF for KDE

- Random Fourier Feature (RFF) [2]:
  - Approximates the kernel function as the inner product of two vectors.

$$K(\mathbf{x}, \mathbf{y}) \approx z(\mathbf{x})^T z(\mathbf{y})$$

where z: $\mathbb{R}^k \to \mathbb{R}^D$, and usually $D > k$

$$\mathbf{w} \sim p(\mathbf{w}), b \sim Uniform(0,2\pi)$$
$$z_{\mathbf{w}}(\mathbf{x}) = \sqrt{2}\cos(\mathbf{w}^T\mathbf{x} + b)$$

$$\begin{aligned}
k(\boldsymbol{x}-\boldsymbol{y}) &= \int_{\mathbb{R}^d} p(\boldsymbol{w}) e^{j\boldsymbol{w}^T(\boldsymbol{x}-\boldsymbol{y})} \mathrm{d}\boldsymbol{w} && (1) \\
&= \mathbf{E}_{\boldsymbol{w}}[e^{j\boldsymbol{w}^T(\boldsymbol{x}-\boldsymbol{y})}] && (2) \\
&= \mathbf{E}_{\boldsymbol{w}}[\cos(\boldsymbol{w}^T(\boldsymbol{x}-\boldsymbol{y}))] && (3) \\
&= \mathbf{E}_{\boldsymbol{w}}[\cos(\boldsymbol{w}^T(\boldsymbol{x}-\boldsymbol{y})+2b)] + \mathbf{E}_{\boldsymbol{w}}[\cos(\boldsymbol{w}^T(\boldsymbol{x}-\boldsymbol{y}))] && (4) \\
&= \mathbf{E}_{\boldsymbol{w}}[\sqrt{2}\cos(\boldsymbol{w}^T\boldsymbol{x}+b)\sqrt{2}\cos(\boldsymbol{w}^T\boldsymbol{y}+b)] && (5) \\
&= \mathbf{E}_{\boldsymbol{w}}[z_{\boldsymbol{w}}(\boldsymbol{x}) z_{\boldsymbol{w}}(\boldsymbol{y})] && (6) \\
&= \frac{1}{D}\sum_{m=1}^{D} z_{\boldsymbol{w}_m}(\boldsymbol{x}) z_{\boldsymbol{w}_m}(\boldsymbol{y}) && (7) \\
&= z(\boldsymbol{x})^T z(\boldsymbol{y}) && (8)
\end{aligned}$$

(1) Inverse Fourier Transform

(2) Bochner's Theorem

(3) Euler's Formula

(4) The expectation is 0

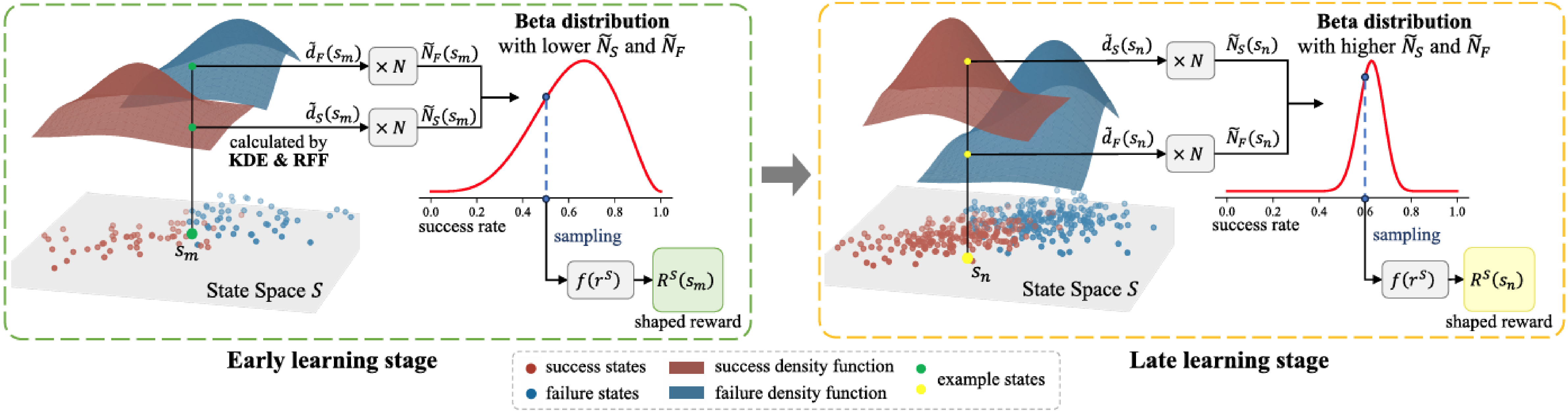
(5) Sum-to-Product Formulas

(6) Define the mapping z

(7) Monte Carlo Method

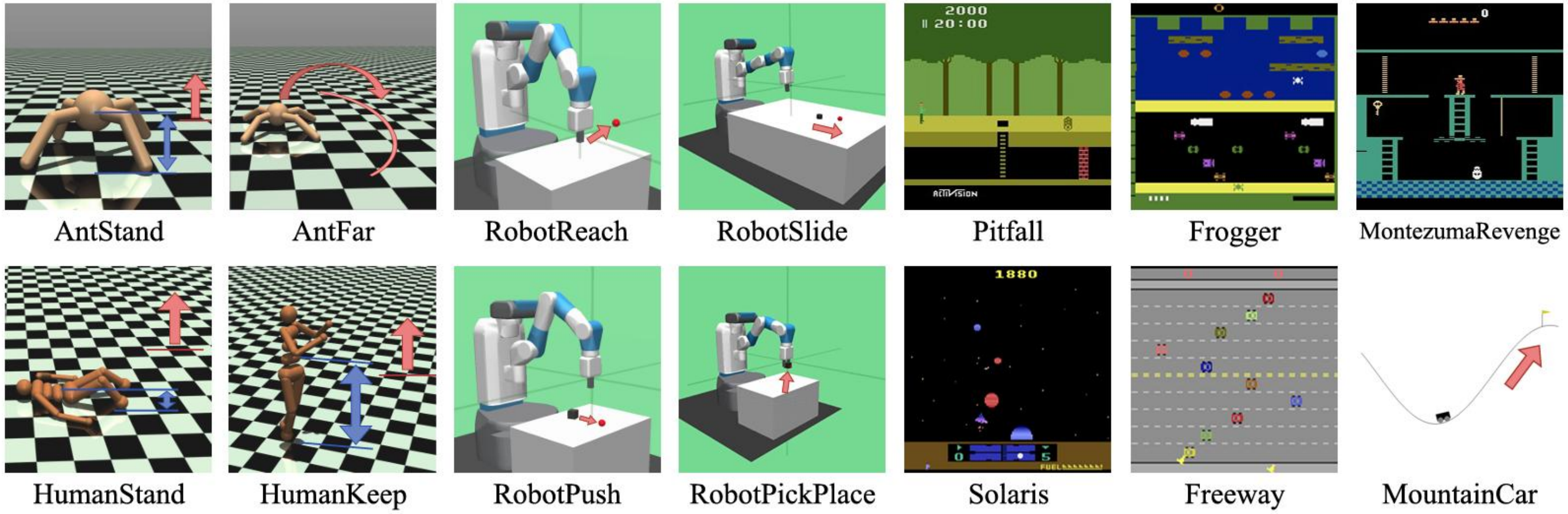
[2] Ali Rahimi and Benjamin Recht. 2007. Random features for large-scale kernel machines. Advances in neural information processing systems 20, (2007).

Self-Adaptive Reward Shaping (SASR)



Comparison with Baselines

- Environments



- Baselines
    - DRND (Yang et al., 2024)
    - ReLara (Ma et al., 2024)
    - GFA-RFE (Zhang et al., 2024)
    - ROSA (Mguni et al., 2023)
    - ExploRS (Devidze et al., 2022)
    - #Explo (Tang et al., 2017)
    - RND (Burda et al., 2018)
    - SAC (Haarnoja et al., 2018)
    - TD3 (Fujimoto et al., 2018)
    - PPO (Schulman et al., 2017)

Comparison with Baselines

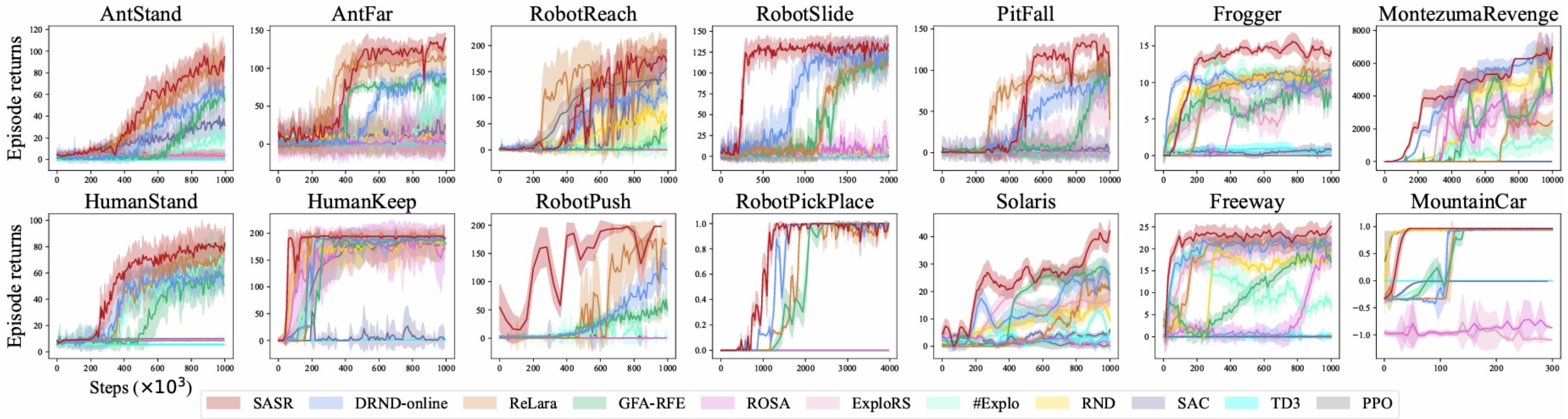


Table 1: The average episodic returns and standard errors of all models tested over 100 episodes.

Tasks	SASR	DRND-online	ReLara	GFA-RFE	ROSA	ExploRS	#Explo	RND	SAC	TD3	PPO
AntStand	**94.9±0.0**	67.3±0.0	90.5±1.7	54.2±0.0	3.8±0.4	5.1±0.4	17.9±0.0	4.0±0.2	31.6±0.0	0.0±0.0	4.9±0.1
AntFar	**139.8±0.0**	93.2±0.0	115.7±0.0	86.4±0.0	1.0±0.0	12.0±4.2	75.1±0.0	4.6±1.6	25.3±0.0	1.0±0.0	7.8±0.0
HumanStand	**79.8±2.0**	50.6±0.0	76.2±0.7	58.2±0.0	8.8±0.0	9.3±0.0	72.7±0.0	9.3±0.1	9.9±0.0	5.5±0.0	9.0±0.1
HumanKeep	**195.8±0.0**	154.5±0.0	194.9±0.0	141.5±0.0	169.7±0.0	182.8±0.0	195.0±0.0	180.7±0.0	2.5±0.0	1.0±0.0	138.1±0.0
RobotReach	170.2±0.0	99.8±0.0	**187.9±0.0**	42.1±0.0	0.1±0.0	0.7±0.0	4.6±0.0	69.3±0.0	156.5±0.0	0.0±0.0	79.5±0.0
RobotSlide	**132.3±1.3**	127.2±0.0	111.6±2.0	115.8±2.0	11.2±0.9	4.3±0.1	3.5±0.0	4.8±0.2	0.7±0.2	0.5±0.4	0.2±0.2
RobotPush	**167.1±0.0**	122.2±0.0	166.9±0.0	49.1±0.0	0.0±0.0	0.0±0.0	3.7±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
RobotPickPlace	**1.0±0.0**	**1.0±0.0**	1.0±0.0	0.5±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Pitfall	**93.0±0.0**	92.0±0.0	40.3±0.0	89.4±0.0	0.0±0.0	57.6±0.0	0.0±0.0	0.0±0.0	4.6±0.0	0.5±0.0	0.0±0.0
Frogger	**14.2±0.0**	11.7±0.0	11.6±0.0	7.9±0.0	9.8±0.0	8.3±0.0	11.9±0.0	10.5±0.0	0.8±0.0	0.7±0.0	0.0±0.0
Montezuma	6737.9±0.0	**6828.5±0.0**	2421.9±0.0	4755.3±0.0	4294.4±0.0	3971.5±0.0	1400.1±0.0	5494.3±0.0	0.0±0.0	0.0±0.0	0.0±0.0
Solaris	**42.1±0.0**	21.3±0.7	20.3±0.0	26.3±0.0	0.1±0.0	17.0±0.0	1.2±0.8	9.8±0.0	6.0±0.0	0.4±0.0	1.5±0.0
Freeway	**22.4±0.0**	19.8±0.0	21.5±0.0	10.1±0.0	18.0±0.0	17.5±0.0	6.9±0.0	13.0±0.0	0.1±0.0	0.2±0.0	0.0±0.0
MountainCar	**1.0±0.0**	**1.0±0.0**	**1.0±0.0**	**1.0±0.0**	-0.9±0.0	-1.0±0.0	**1.0±0.0**	**1.0±0.0**	-0.1±0.0	0.0±0.0	0.9±0.0

Self-Adaptive Exploration-Exploitation Balance

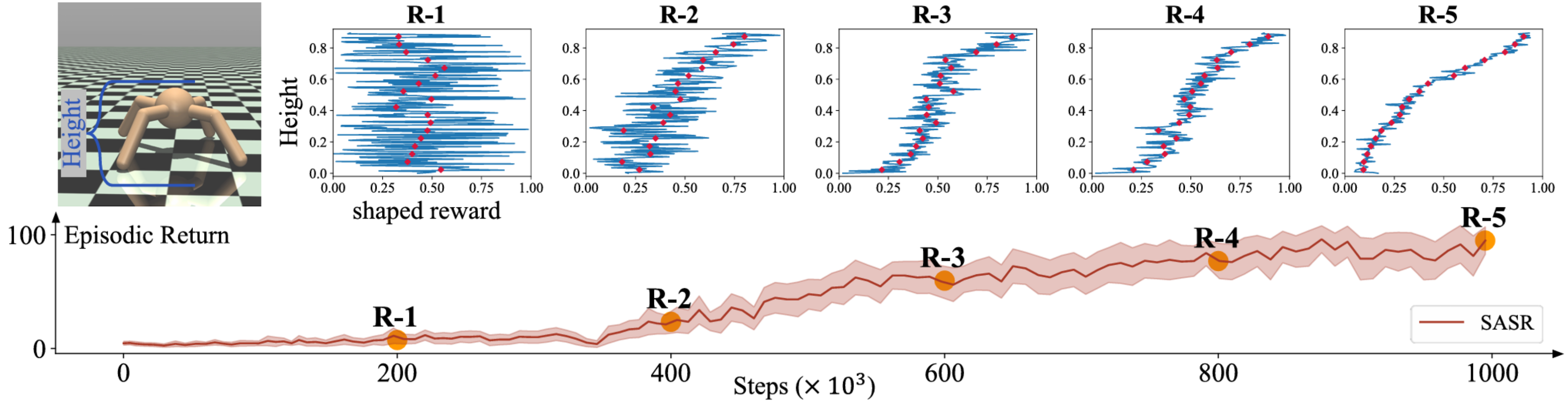
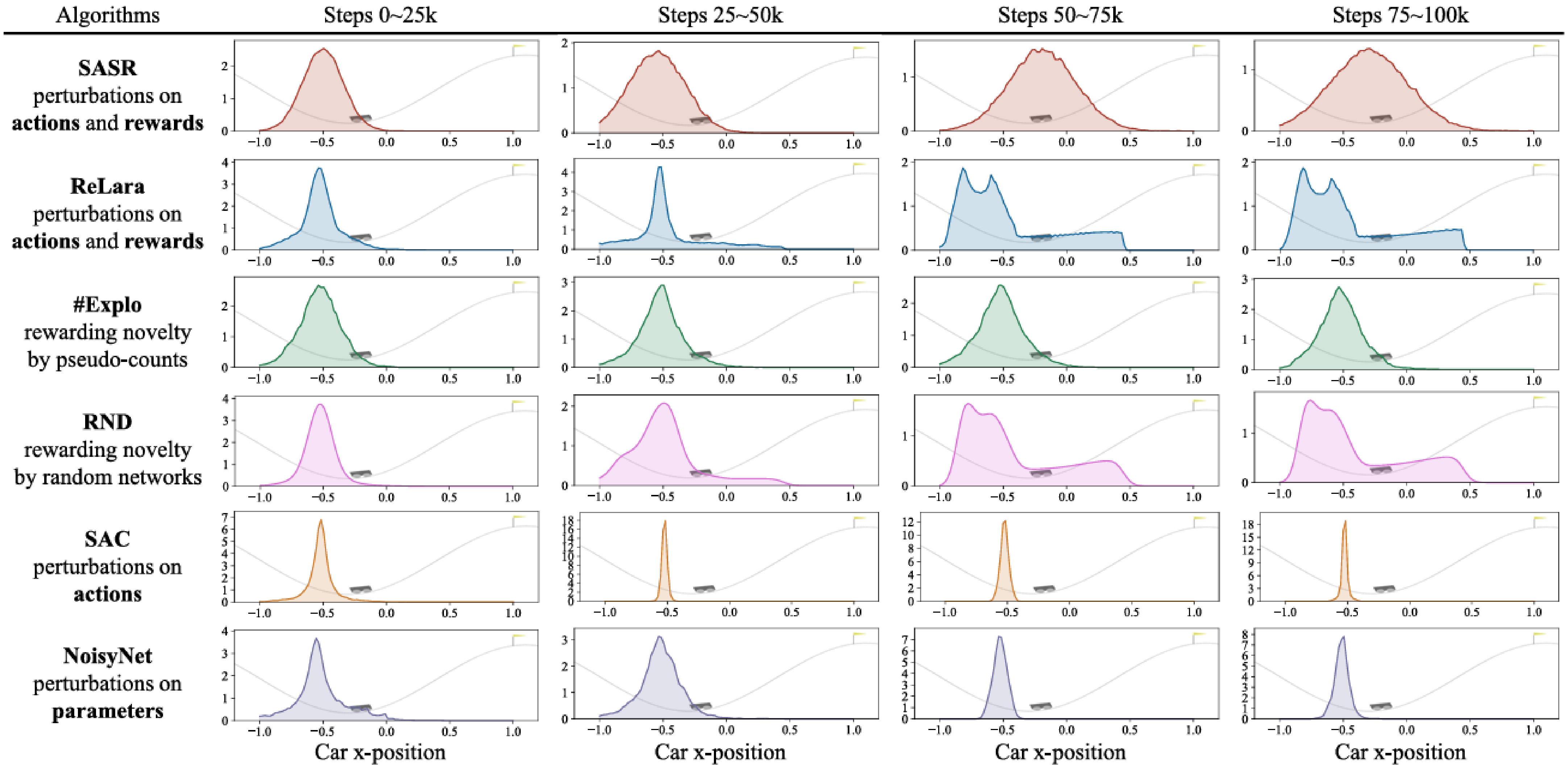


Figure 4: Distributions of the shaped rewards over the height of the ant robot in the *AntStand* task at different training stages. Red diamonds represent the estimated success rate, while the blue polylines show the actual shaped rewards sampled from the Beta distribution.

Self-Adaptive Exploration-Exploitation Balance



Thank You!

Paper

Codes