Reward-Machine-Guided, Self-Paced Reinforcement Learning (Supplementary Material)

Cevahir Koprulu[1] **Ufuk Topcu**[1]

[1]University of Texas at Austin, USA

A THEORY

Theorem 2. *Under Assumption 1, $\mathcal{G}_1$ and $\mathcal{G}_2$ are sets of sufficient context parameters on (q, s, a, s') if and only if $\mathcal{G}_1 \cap \mathcal{G}_2$ is a set of sufficient context parameters on (q, s, a, s').*

Proof. Sufficiency (backward statement): $\mathcal{G}_1 \cap \mathcal{G}_2 \subseteq \mathcal{G}_1$ and $\mathcal{G}_1 \cap \mathcal{G}_2 \subseteq \mathcal{G}_2$, then by Lemma 1, $\mathcal{G}_1 \cap \mathcal{G}_2$ is a set of sufficient context parameters on (q, s, a, s').

Necessity (forward statement): Let $c_{\mathcal{G}} = [c[i]]_{i \in \mathcal{G}}$ for any $\mathcal{G} \subseteq 2^D$. Then, for any $c, c' \in \mathcal{C}$ that satisfy $c_{\mathcal{G}_1 \cap \mathcal{G}_2} = c'_{\mathcal{G}_1 \cap \mathcal{G}_2}$ we want to show that $\delta_{\mathsf{q}}(\mathsf{q}, L_c(s, a, s')) = \delta_{\mathsf{q}}(\mathsf{q}, L_{c'}(s, a, s'))$. Under Assumption 1, there exists $c'' \in \mathcal{C}$ such that

$$\begin{aligned} c_{\mathcal{G}_1 \setminus \mathcal{G}_2} &= c''_{\mathcal{G}_1 \setminus \mathcal{G}_2}, \\ c'_{\mathcal{G}_2 \setminus \mathcal{G}_1} &= c''_{\mathcal{G}_2 \setminus \mathcal{G}_1}, \\ c_{\mathcal{G}_1 \cap \mathcal{G}_2} &= c''_{\mathcal{G}_1 \cap \mathcal{G}_2} = c'_{\mathcal{G}_1 \cap \mathcal{G}_2}. \end{aligned}$$

Such a choice is always possible since $\mathcal{G}_1 \cap \mathcal{G}_2$, $\mathcal{G}_1 \setminus \mathcal{G}_2$ and $\mathcal{G}_2 \setminus \mathcal{G}_1$ are disjoint sets. Then,

$$\begin{aligned} c''_{\mathcal{G}_1} &= c_{\mathcal{G}_1}, \\ c''_{\mathcal{G}_2} &= c'_{\mathcal{G}_2}. \end{aligned}$$

Therefore, $\delta_{\mathsf{q}}(\mathsf{q}, L_c(s, a, s')) = \delta_{\mathsf{q}}(\mathsf{q}, L_{c''}(s, a, s')) = \delta_{\mathsf{q}}(\mathsf{q}, L_{c'}(s, a, s'))$. □

B EXPERIMENTAL DETAILS

We provide details about the algorithms and the environments studied in the paper. These details include hyperparameters of the algorithms, how we select them, and the structure of the environments.

B.1 ALGORITHMS

We compare two baseline algorithms and four automated curriculum generation methods:

1. **Default**: We evaluate learning without a curriculum, i.e. sampling contexts from the target context distribution and training the agent in these contexts as a baseline.
2. **Default***: We extend *Default* by running it on a product contextual MDP, thus we observe the effect of capturing temporal abstractions without learning a curriculum.

Accepted for the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023).

Algorithm 1 Intermediate Self-Paced RL

Input: Product MDP $\bar{\mathcal{M}}^L_{\mathcal{R}}$, target context distribution φ, initial context distribution $\varrho(\cdot|\nu_0)$,
Parameter: KL penalty proportion ζ, relative entropy bound ϵ, KL penalty offset offset K_α, number K of iterations, number N of rollouts
Output: Final policy π_{ω_K}

1: Initialize policy π_{ω_0}.
2: **for** $k = 1$ to K **do**
3: $c_i \sim \varrho(c|\nu_{k-1}), i \in [N]$, ▷ *sample contexts*
4: $\mathcal{D}_k \leftarrow \{(c_i, \bar{\tau}_i)|\bar{\tau}_i = (\bar{s}_{i,0}, a_{i,0}, \bar{r}_{i,1}, \bar{s}_{i,1}), \cdots, (\bar{s}_{i,T_i-1}, a_{i,T_i-1}, \bar{r}_{i,T_i}, \bar{s}_{i,T_i}), i \in [N]\}$, ▷ *collect trajectories*
5: $\pi_{\omega_k} \leftarrow \Psi(\mathcal{D}_k, \pi_{\omega_{k-1}})$ ▷ *update policy with RL algorithm* Ψ
6: Compute next context distribution parameter ν_k by solving

$$\begin{aligned} \max_{\nu_k} \quad & \frac{1}{N}\sum_{i=1}^{N}\sum_{t=0}^{T_i-1}\gamma^t \frac{\varrho(c_i|\nu_k)}{\varrho(c_i|\nu_{k-1})}\bar{r}_{i,t+1} - \alpha_k D_{\mathrm{KL}}(\varrho(c|\nu_k) \,||\, \varphi(c)) \\ \text{s.t.} \quad & D_{\mathrm{KL}}(\varrho(c|\nu_k) \,||\, \varrho(c|\nu_{k-1})) \le \epsilon, \end{aligned} \tag{1}$$

where $\alpha_k = \begin{cases} 0 & \text{if } k \le K_\alpha; \\ \mathsf{B}(\nu_{k-1}, \mathcal{D}_k) & \text{otherwise,} \end{cases}$, and $\mathsf{B}(\nu_{k-1}, \mathcal{D}_k) = \zeta \frac{\max(0, \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T_i}\gamma^t \bar{r}_{i,t})}{D_{\mathrm{KL}}(\varrho(c|\nu_{k-1})||\varphi(c))}$.

7: **end for**
8: **return** π_{ω_K}

3. **GoalGAN**: Florensa et al. [2018] develop *Goal Generative Adversarial Network* for the goal-conditioned setting, where a goal discriminator to determine whether a goal is at the intermediate difficulty for the current policy, and a goal generator which generates goals that are at such level of difficulty. Although GoalGAN takes an initial context distribution, similar to self-paced RL approaches, it does not allow for a target context distribution, instead, it generates a curriculum as if the target context distribution is a uniform distribution over the context space $\mathcal{C}$.
4. **ALP-GMM**: Florensa et al. [2018] propose *Absolute Learning Progress with Gaussian Mixture Models*, which generates a Gaussian mixture model over the absolute learning progress of task parameters, e.g. contexts in our setting. ALP-GMM uses a bandit scheme to choose a Gaussian as an arm whose utility is the absolute learning progress. The chosen Gaussian distribution is used to draw the next task parameter.
5. **SPDL**: Klink et al. [2020] propose *Self-paced Deep RL* by exploiting deep learning methods under the self-paced RL framework. We build Intermediate SPRL and RM-guided SPRL on top of this algorithm.
6. **Intermediate SPRL**: We present an intermediate self-paced RL algorithm, which runs on the product contextual MDP. Therefore, we assess how self-paced RL performs when the agent can capture the temporal abstractions. We provide the pseudocode in Algorithm 1.
7. **RM-guided SPRL**: We develop a reward-machine-guided, self-paced RL algorithm, which uses reward machines to update the policy and value functions of an RL agent, as well as to direct the curriculum generation.

In our experiments, all methods use the soft actor-critic algorithm Haarnoja et al. [2018] as the RL method of choice.

Table 1 shows the hyperparameters of RM-guided SPRL, Intermediate SPRL, and SPDL algorithms in three case studies:

- **Case-1:** Two-door environment & 2D context space with wide target distribution,
- **Case-2:** Customized Swimmer-v3 environment & 2D context space with narrow target distribution,
- **Case-3:** Customized HalfCheetah-v3 environment & 3D context space with narrow target distribution.

There are four parameters in Table 1 that we do not provide in Algorithm 1: K_{OFFSET}, n_{STEP}, σ_{LB}, and $D_{\text{KL}_{LB}}$. Klink et al. [2020] introduce these parameters as a part of SPDL. K_{OFFSET} is the number of context distribution updates before they enable a self-paced RL algorithm to update the initial context distribution. This parameter allows learning a meaningful value function that estimates the expectation of the value of the initial states, which the objective function of the self-paced RL problem takes into account. n_{STEP} is the number of environment interactions between two context distribution updates, and it replaces N, which is the number of rollouts, i.e., trajectories between two context distribution updates. σ_{LB} is the lower bound for the standard deviation of a context distribution, which is used to stabilize learning, particularly for narrow

	Algorithm	ϵ	K_{OFFSET}	ζ	K_α	n_{STEP}	σ_{LB}	$D_{\text{KL}_{LB}}$
Case-1	RM-guided SPRL	0.05	70	0.96	10	16384	$(4 \cdot 10^{-3}, 4 \cdot 10^{-3})$	8000
	Intermediate SPRL	0.05	70	1.2	10	16384	$(4 \cdot 10^{-3}, 4 \cdot 10^{-3})$	8000
	SPDL	0.05	70	1.2	10	16384	$(4 \cdot 10^{-3}, 4 \cdot 10^{-3})$	8000
Case-2	RM-guided SPRL	0.1	10	1.0	5	16384	$(4 \cdot 10^{-3}, 4 \cdot 10^{-3})$	8000
	Intermediate SPRL	0.1	10	4.0	5	16384	$(4 \cdot 10^{-3}, 4 \cdot 10^{-3})$	8000
	SPDL	0.1	10	4.0	5	16384	$(4 \cdot 10^{-3}, 4 \cdot 10^{-3})$	8000
Case-3	RM-guided SPRL	0.05	80	1.0	0	16384	$(4 \cdot 10^{-3}, 4 \cdot 10^{-3}, 4 \cdot 10^{-3})$	8000
	Intermediate SPRL	0.05	80	4.0	0	16384	$(4 \cdot 10^{-3}, 4 \cdot 10^{-3}, 4 \cdot 10^{-3})$	8000
	SPDL	0.05	80	4.0	0	16384	$(4 \cdot 10^{-3}, 4 \cdot 10^{-3}, 4 \cdot 10^{-3})$	8000

Table 1: Hyper-parameters for self-paced RL algorithms

Case	μ_{INIT}	Σ_{INIT}	μ_{TARGET}	Σ_{TARGET}
1	(0, 0)	$diag((0.25, 0.25))$	(2, 2)	$diag((1, 1))$
2	(0, 1)	$diag((0.1, 0.1))$	(-0.6, 1.6)	$diag((1.6 \cdot 10^{-7}, 1.6 \cdot 10^{-7}))$
3	(1, 4, 7)	$diag((0.25, 0.25, 0.25))$	(4, 7, 10)	$diag((1.6 \cdot 10^{-7}, 1.6 \cdot 10^{-7}, 1.6 \cdot 10^{-7}))$

Table 2: Initial and target context distributions

target distributions. $D_{\text{KL}_{LB}}$ is the threshold for KL divergence to the target context distribution and it determines when the algorithm stop clipping the standard deviation of a context distribution using σ_{LB}. In all case studies, we run a grid search over

$$
\begin{aligned}
K_\alpha &\in \{0, 5, 10\}, \\
n_{\text{STEP}} &\in \{8192, 16384\}, \\
D_{\text{KL}_{LB}} &\in \{8000, 10000\}, \\
\epsilon &\in \{0.05, 0.1\}.
\end{aligned}
$$

We set σ_{LB} with respect to the standard deviation of the narrow target context distribution (see Table 2). For K_{OFFSET}, the grid search is over the sets $\{60, 70\}$, $\{5, 10\}$, and $\{70, 80\}$ for Case-1, Case-2 and Case-3, respectively. For ζ, the parameter value search in Case-1 is over $\{0.96, 0.98, 1.0, 1.2\}$, whereas the search in Case-2 and Case-3 is over $\{1.0, 2.0, 3.0, 4.0\}$. We use the parameters that yield the fastest convergence to the optimal expected discounted return via Intermediate SPRL for SPDL (see Table 1).

We compare GoalGAN with self-paced RL algorithms in our case studies since it is a state-of-the-art automated curriculum generation method that can handle sparse rewards. We tune the random noise δ_{NOISE} that is on every sample, the number $n^{\text{GG}}_{\text{ROLLOUT}}$ of policy rollouts between context distribution updates, and the percentage p_{SUCCESS} of samples drawn from the success buffer. The tuning is done for every case via a grid-search over

$$
\begin{aligned}
\delta_{\text{NOISE}} &\in \{0.05, 0.1\}, \\
n^{\text{GG}}_{\text{ROLLOUT}} &\in \{50, 100\}, \\
p_{\text{SUCCESS}} &\in \{0.2, 0.3\}.
\end{aligned}
$$

We also evaluate ALP-GMM as it is a state-of-the-art automated curriculum generation method that is competitive with SPDL [Klink et al., 2021]. We tune the percentage of random context samples p_{RAND}, the number $n^{\text{AG}}_{\text{ROLLOUT}}$ of policy rollouts between context distribution updates, and the size of the buffer of past trajectories s_{BUFFER}. The tuning is done for every case via a grid-search over

$$
\begin{aligned}
p_{\text{RAND}} &\in \{0.2, 0.3\}, \\
n^{\text{AG}}_{\text{ROLLOUT}} &\in \{100, 200\}, \\
s_{\text{BUFFER}} &\in \{1000, 2000\}.
\end{aligned}
$$

Due to failing to accomplish the task in every experiment with all combinations of these available parameter values, we use a combination that we consider locally better than the other combinations in terms of the expected discounted return obtained in the evaluation runs. The hyperparameters of GoalGAN and ALP-GMM that we use are in Table 3.

Case	δ_{NOISE}	$n^{\text{GG}}_{\text{ROLLOUT}}$	p_{SUCCESS}	p_{RAND}	$n^{\text{AG}}_{\text{ROLLOUT}}$	s_{BUFFER}
1	0.1	100	0.3	0.2	200	1000
2	0.1	100	0.3	0.3	200	1000
3	0.1	100	0.3	0.3	200	1000

Table 3: Hyper-parameters for GoalGAN and ALP-GMM

In Case-1, we set the discount rate γ to 0.98, whereas Case-2 and Case-3 use $\gamma = 0.99$. Every curriculum learning method uses the stable-baselines3 Raffin et al. [2021] implementation of the soft actor-critic (SAC) RL algorithm Haarnoja et al. [2018]. In Case-1, we provide a replay buffer of size 150,000 and a batch size of 64, start the learning after 500 environment interactions, update the policy every 5 interactions via the soft Q-updates, and use a multi-layer perceptron policy with 2 layers of 64 neurons and tanh activation function. In Case-2 and Case-3, SAC uses a replay buffer size of 500,000, and batch size of 256, updates the policy every 8 steps and the learning starts after 10,000 interactions with the environment. SAC also uses a multi-layer perceptron policy with 2 layers of 256 neurons and ReLU activation function. In addition, we set the learning rate to 0.001. We keep the rest of the hyperparameters as what the implementation provides in its default setting.

We conduct all experiments on a laptop with an 11^{th} Gen Intel Core i7-11800H processor and an Nvidia GeForce RTX 3060 graphics card and 16GB of RAM.

B.2 ENVIRONMENTS

B.2.1 Two-Door Environment

Case-1 is based on a two-door environment (see Fig. 1). The two-door environment is a 40-by-40 grid world, where the initial state is at the coordinates (20, 35), which correspond to the positions along horizontal and vertical axes, respectively. Box is a 5-by-5 square, and its top-left corner is at the coordinates (15, 20). The goal is positioned at (20, 5). We set the vertical position of the doors to 30 and 10 for the first and second doors, respectively. The width of the doors has a width of 5. In short, the labeled contextual MDP $\bar{\mathcal{M}}^L$ of the two-door environment has a state space $S = \{1, 2, \cdots, 40\}^2$ and action space $A = \{U, D, L, R\}$, which correspond to the coordinates and the four cardinal directions, i.e., up, down, left, right, respectively. The transitions in the two-door environment are deterministic. Unless the agent completes the task or dies by moving onto a wall or the second door before getting the key, we allow the agent to take at most 4800 steps.

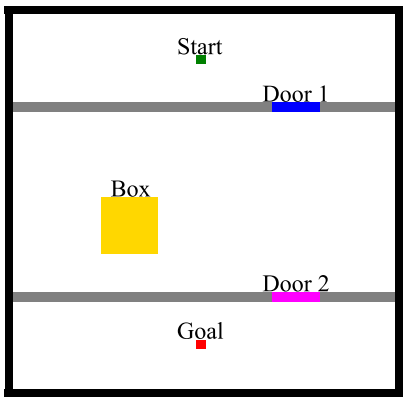


Figure 1: Two-door Environment

Case study 1 has a context space $\mathcal{C}_1 = [-4, 4]^2$, where the context parameters are the horizontal positions of the first and

second doors, respectively. We design the reward-machine-context mapping F_1 for case study 1 as

$$\begin{aligned}
&\mathsf{F}_1(\mathsf{q}_0, \mathsf{q}_0) =, \mathsf{F}_1(\mathsf{q}_0, \mathsf{q}_1) = \{1\}, \mathsf{F}_1(\mathsf{q}_0, \mathsf{q}_5) = \{1\},\\
&\mathsf{F}_1(\mathsf{q}_1, \mathsf{q}_1) = \{1\}, \mathsf{F}_1(\mathsf{q}_1, \mathsf{q}_2) =, \mathsf{F}_1(\mathsf{q}_1, \mathsf{q}_5) = \{1\},\\
&\mathsf{F}_1(\mathsf{q}_2, \mathsf{q}_2) = \{1\}, \mathsf{F}_1(\mathsf{q}_2, \mathsf{q}_3) = \{2\}, \mathsf{F}_1(\mathsf{q}_2, \mathsf{q}_5) = \{1, 2\},\\
&\mathsf{F}_1(\mathsf{q}_3, \mathsf{q}_3) = \{1, 2\}, \mathsf{F}_1(\mathsf{q}_3, \mathsf{q}_4) =, \mathsf{F}_1(\mathsf{q}_3, \mathsf{q}_5) = \{1, 2\}.
\end{aligned}$$

An expert designs such a mapping by asking questions about the task structure. For instance, for the transition $\mathsf{q}_1, \mathsf{q}_5$ in the reward machine in Figure 3, the expert should ask: Is there a transition s, a, s' in the labeled contextual MDP $\bar{\mathcal{M}}^L$ such that it causes the agent to hit the wall, i.e., $(\mathsf{q}_1, \mathsf{q}_5)$, for some context c but lets the agent pass through the door, i.e., $(\mathsf{q}_1, \mathsf{q}_2)$, for a different context c'? The idea is to find the context parameters $i \in \{1, \ldots, dim(\mathcal{C})\}$ for which a change of value, e.g. $c[i] \neq c'[i]$, prevents a transition $(\mathsf{q}, \mathsf{q}')$ in the reward machine from happening. For $(\mathsf{q}_1, \mathsf{q}_5)$, the mapping outputs the first context parameter, $\mathsf{F}(\mathsf{q}_1, \mathsf{q}_5) = \{1\}$, as the identifier, since it determines the position of the first door. In other words, when the agent is in the second room and can move into the first door/wall with an up action, then the position of the first door determines whether it moves into the door, or the wall. However, the position of the second door does not identify which transition will happen.

Figure 2 demonstrates the progression of the rate of successful task completion in contexts drawn from the target context distribution.

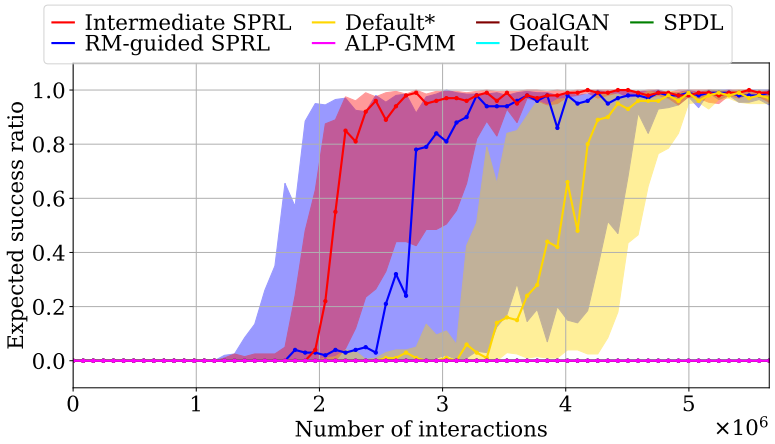


Figure 2: Two-door Environment: Progression of the successful episodes ratio in contexts drawn from the target context distribution over curriculum updates.

B.2.2 Custom Swimmer-v3 Environment

Case study 3 is based on a variation of the Swimmer-v3 environment from OpenAI gym Brockman et al. [2016]. The original environment consists of a robot that moves like a worm by applying force on 2 joints. The objective is to move towards the right of the initial position as fast as possible. The state and action spaces are 8 and 2-dimensional continuous spaces, respectively. We set the maximum number of steps in an episode to 10000. We design the reward-machine-context mapping F_3 for case study 3 as:

$$\mathsf{F}_2(\mathsf{q}_0, \mathsf{q}_0) = \{2\}, \mathsf{F}_2(\mathsf{q}_0, \mathsf{q}_1) = \{2\}, \mathsf{F}_2(\mathsf{q}_1, \mathsf{q}_1) = \{1\}, \mathsf{F}_2(\mathsf{q}_1, \mathsf{q}_2) = \{1\}, \mathsf{F}_2(\mathsf{q}_2, \mathsf{q}_2) = \{\}.$$

Figure 3 demonstrates the progression of the expected discounted return with respect to the target context distribution. Figure 5a illustrates how self-paced RL algorithms update context distribution parameters, i.e., mean and variance of normal distributions, during the training.

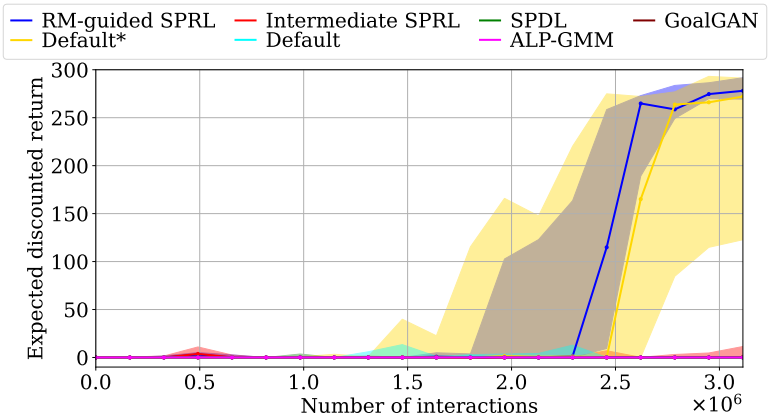


Figure 3: Customized-Swimmer Environment: Progression of the expected discounted return with respect to the target context distribution.

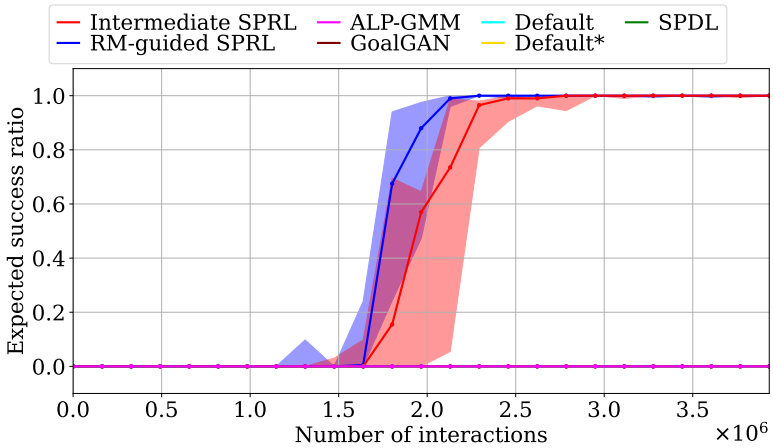


Figure 4: Customized HalfCheetah-v3 Environment: Progression of the successful episodes ratio in contexts drawn from the target context distribution over curriculum updates.

B.2.3 Custom HalfCheetah-v3 Environment

Case study 3 is based on a variation of the HalfCheetah-v3 environment from OpenAI gym Brockman et al. [2016]. The original environment consists of a 2-dimensional robot, shaped like a cheetah. The objective is to make the cheetah run forward, which corresponds to the right of the scene, as fast as possible by applying torque on 6 joints. The state and action spaces are 18 and 6-dimensional continuous spaces, respectively. We set the maximum number of steps in an episode to 2000. We design the reward-machine-context mapping F_3 for case study 3 as:

$$\mathsf{F}_3(q_0, q_0) = \{1\}, \mathsf{F}_3(q_0, q_1) = \{1\}, \mathsf{F}_3(q_1, q_1) = \{2\}, \mathsf{F}_3(q_1, q_2) = \{2\},$$
$$\mathsf{F}_3(q_2, q_2) = \{1\}, \mathsf{F}_3(q_2, q_3) = \{1\}, \mathsf{F}_3(q_3, q_3) = \{3\}, \mathsf{F}_3(q_3, q_4) = \{3\}.$$

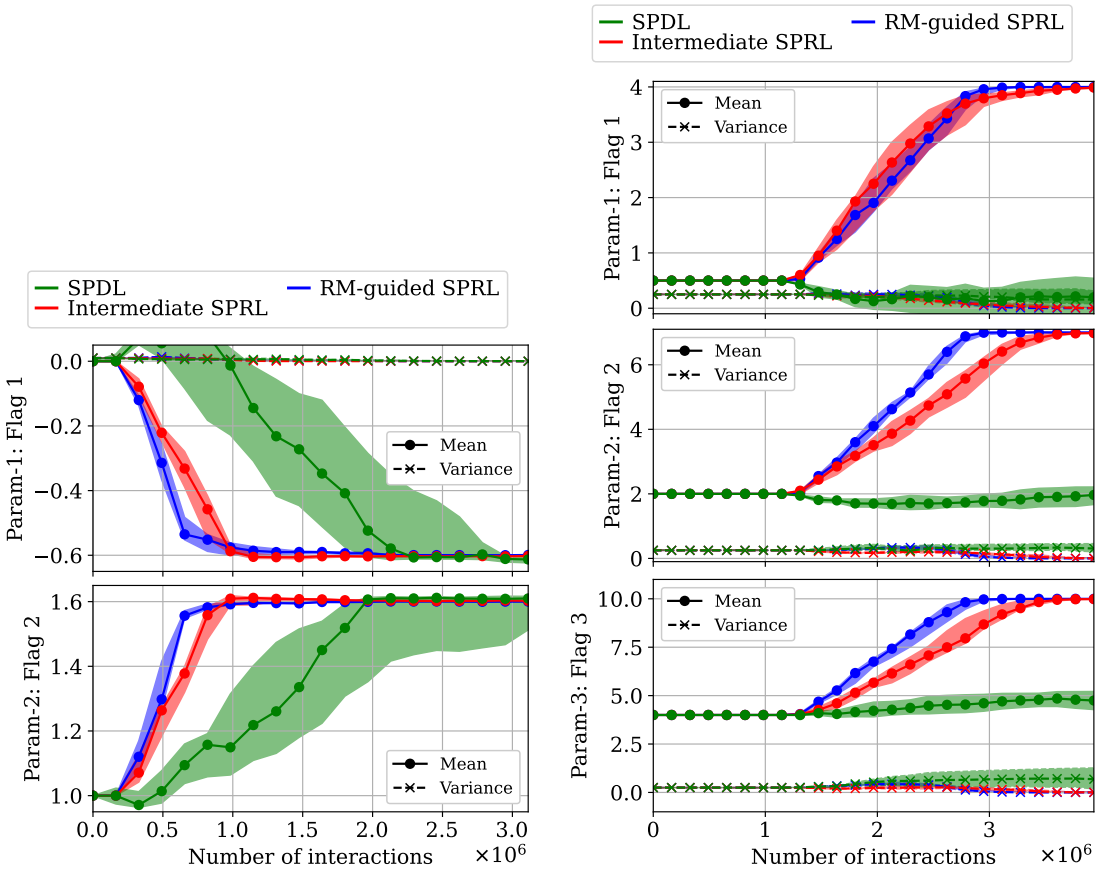
To support the discussion of results in customized HalfCheetah-v3, we provide two tables. Table 4 shows the expected discounted return and expected success rate achieved by the final policies in every training run of RM-guided SPRL and Intermediate SPRL. Table 5 provides the means of Gaussian context distributions generated at the final iteration in every training run of RM-guided SPRL and Intermediate SPRL. In the last training run of Intermediate SPRL, the trained agent fails to learn a policy that can complete the task and the curriculum does not converge to the target context distribution. Figure 4 demonstrates the progression of the rate of successful task completion in contexts drawn from the target context distribution. Figure 5b illustrates how self-paced RL algorithms update context distribution parameters, i.e., mean and variance of normal distributions, during the training.

Table 4: Customized HalfCheetah-v3: Expected discounted returns and success rates achieved by policies from the final iteration of every training run.

Algorithm	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Seed 6	Seed 7	Seed 8	Seed 9	Seed 10
RM-guided SPRL	455.37 100%	472.31 100%	479.13 100%	506.75 100%	446.76 99%	472.85 100%	501.31 100%	467.63 100%	507.10 100%	499.94 100%
Intermediate SPRL	475.57 100%	488.74 100%	467.48 100%	475.58 100%	492.25 100%	509.89 100%	485.06 100%	493.68 100%	492.13 100%	**-0.47** **0%**

Table 5: Customized HalfCheetah-v3: Means of Gaussian context distributions generated at the final iteration of every training run. Note that the mean of the target context distribution is $(4, 7, 10)$.

Algorithm	Seed 1	Seed 2	Seed 3	Seed 4	Seed 5	Seed 6	Seed 7	Seed 8	Seed 9	Seed 10
RM-guided SPRL	3.99 7.00 9.99	3.99 6.99 9.99	4.00 6.99 9.99	4.00 7.00 9.99	4.00 6.99 9.99	4.00 7.00 9.99	4.00 7.00 9.99	3.99 6.99 9.997	3.99 6.99 10.00	3.99 7.00 9.99
Intermediate SPRL	3.98 6.98 9.98	3.99 6.98 9.99	3.96 6.97 9.94	3.95 6.96 9.96	3.99 6.99 9.99	3.98 6.97 9.98	3.98 6.99 9.99	3.98 6.97 9.97	3.98 6.98 9.99	**0.11** **1.44** **3.74**



(a) Customized Swimmer-v3 environment.

(b) Customized HalfCheetah-v3 environment.

Figure 5: Progression of the statistics (mean and variance) of context distributions generated in the curriculum.

References

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *ICML*, pages 1515–1528, 2018.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, pages 1861–1870, 2018.

Pascal Klink, Carlo D' Eramo, Jan R Peters, and Joni Pajarinen. Self-paced deep reinforcement learning. In *NeurIPS*, pages 9216–9227, 2020.

Pascal Klink, Hany Abdulsamad, Boris Belousov, Carlo D'Eramo, Jan Peters, and Joni Pajarinen. A probabilistic interpretation of self-paced learning with applications to reinforcement learning. *JMLR*, 22:182:1–182:52, 2021.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *JMLR*, pages 1–8, 2021.