A Pseudo-Code

Algorithm 1 CSRO Meta-training

Input: Offline Datasets $D = \{D_i\}_{i=1}^{N_{env}}$ of a set of training tasks $\{M_i\}_{i=1}^{N_{env}}$, initialize learned policy π_θ, Q-function Q_ω, context encoder q_ϕ, and CLUB encoder q_ψ, hyperparameter λ
Parameter: $\theta, \omega, \phi, \psi$

```
while not done do
    for step in training steps do
        Sample buffer D_i ~ D and context from buffer c = {(s_j, a_j, r_j, s'_j)} ~ D_i, history transitions h ~ D_i.
        Compute each transition embedding z = q_φ(z|(s, a, r, s')), z = q_ψ(z|(s, a)) and task representation z = q_φ(z|c)
        Compute L_VD(ψ)
        Update ψ to minimize L_VD(ψ)
        Compute L_encoder(φ) = L_maxMI(φ) + λ L_minMI(φ)
        Update φ to minimize L_encoder(φ)
        Use history transitions h to compute L_critic(ω), L_actor(θ)
        Update θ,ω to minimize L_critic(ω), L_actor(θ)
    end for
end while
```

Algorithm 2 CSRO Meta-testing

Input: A set of testing tasks $\{M_i\}_{i=1}^{N_{env}}$, learned policy π_θ, context encoder q_ϕ, random explore step t_r

```
for each task M_i do
    c = {}
    for t = 0, ..., T − 1 do
        if t < t_r then
            Agent samples a random action a_t to roll out (s_t, a_t, r_t, s'_t)
        else
            Compute posterior z = q_φ(z|c).
            Agent use π_θ(a|s, z) roll out (s_t, a_t, r_t, s'_t)
        end if
        c = c ∪ (s_t, a_t, r_t, s'_t)
    end for
    Compute posterior z = q_φ(z|c).
    Roll out policy π_θ(a|s, z) for evaluation
end for
```

B Environment Details

In this section, we show details about the environments of our experiment.

Point-Robot: A problem of control point robot navigation in 2D space. The start position is fixed to $(0, 0)$. The goal of each task is located on a unit semicircle centered on the start position. Each task needs to control the robot from the start position to the goal. The state space is $\mathbb{R}^2$, comprising the XY position of the robot. The action space is $[-1, -1]^2$, with each dimension corresponding to the moving distance in the XY direction. The reward function is defined as the negative distance from the goal.

Half-Cheetah-Vel: Control a Cheetah to move forward and achieve goal velocity. The target velocity is sampled from $[1, 3]$. The state space is $\mathbb{R}^{20}$, comprising the position and velocity of the cheetah; the angle and angular velocity of each joint. The action space is $[-1, 1]^6$, with each dimension corresponding to the torque of each joint. The reward function is the absolute difference between the agent's velocity and the target velocity plus the control cost.

Ant-Goal: The Ant-Goal task consists of controlling an "ant" robot to navigate. The goal of each task is located on a circle with radius 2 centered on $(0, 0)$. The state space is $\mathbb{R}^{29}$, comprising the position and velocity of the ant as well as the angle and angular velocity of 8 joints. The action space is $[-1, 1]^8$, with each dimension corresponding to the torque of each joint. The reward function is defined as the negative distance from the goal plus the control cost.

Humanoid-Dir: The Humanoid-Dir task consists of controlling a "humanoid" robot in the target direction. The target direction of each task is sampled from $[0, 2\pi]$. The state space is $\mathbb{R}^{376}$ and the action space is $[-1, 1]^{17}$. The reward function is the dot between the velocity of the robot and the target direction plus the staying alive bonus and control cost.

Hopper-Rand-Params: The Hopper-Rand-Params is control a one-legged robot to move forward. The source code is taken from the rand_param_envs repository.[1] The tasks are varied in body mass, body inertia, joint damping, and friction. Each parameter is the product of the default value and the coefficient sampled from $[1.5^{-3}, 1.5^3]$. The state space is $\mathbb{R}^{11}$ and the action space is $[-1, 1]^3$. The reward function is forward velocity plus the staying alive bonus and control cost.

Walker-Rand-Paras: The Walker-Rand-Params is control a bi-pedal robot to move forward, also from the rand_param_envs repository. Each parammeter is obtained in the same way as Hopper-Rand-Params and the reward function is the same as Hopper-Rand-Params. The state space is $\mathbb{R}^{17}$ and the action space is $[-1, 1]^6$.

C Offline Data Collections

For each task, we sample 40 environments from environment distribution. Out of these, 30 environments are designated as training environments, while the remaining 10 environments serve as test environments. We employ SAC [9] to train an agent on each training environment and save the policy at different training steps. To create offline datasets, we generate 50 trajectories using each policy from every environment. Table 2 presents the hyperparameters employed during the collection of offline datasets.

Table 2: Hyperparameters used in offline datasets collection.

Hyperparameters	Point-Robot	Half-Cheetah-Vel	Ant-Dir	Humanoid-Dir	Hopper-Rand-Params	Walker-Rand-Params
Training steps	5000	1e6	1e6	1e6	1e6	1e6
Initial steps	2e3	5e4	5e4	5e4	5e4	5e4
Eval frequency	200	5e4	5e4	5e4	5e4	5e4
Sampling episodes	50	50	50	50	50	50
Learning rate	1e-4	1e-4	1e-4	1e-4	1e-4	1e-4
Batch size	1024	1024	1024	1024	1024	1024

D Experimental Setting

For each task, We use offline datasets collected at different times to train. Details of using offline datasets in Table 3:

Table 3: Details of using offline datasets: The 'Checkpoints' column indicates the data collected by policies at different steps during the meta-training phase. The three numbers denote the starting steps, ending steps, and steps spacing.

Env	Checkpoints
Point-Robot	[2200, 4800, 200]
Half-Cheetah-Val	[100000, 950000, 50000]
Ant-Goal	[100000, 950000, 50000]
Humanoid-Dir	[50000, 950000, 50000]
Hopper-Rand-Params	[50000, 950000, 50000]
Walker-Rand-Params	[50000, 950000, 50000]

We list other hyperparameters in the offline meta-training phase in Table 4.

[1] https://github.com/dennisl88/rand_param_envs.

Table 4: Hyperparameters used in offline meta-training.

Hyperparameters	Point-Robot	Half-Cheetah-Vel	Ant-Dir	Humanoid-Dir	Hopper-Rand-Params	Walker-Rand-Params
Reward scale	100	5	5	5	5	5
Latent dimension	20	20	20	20	40	40
Use BRAC	False	True	True	True	True	True
Batch size	256	256	256	256	256	256
Meta batch size	16	16	10	16	16	16
Embedding batch size	1024	100	512	256	256	256
Actor Learning rate	3e-4	3e-4	3e-4	3e-4	3e-4	3e-4
Critic Learning rate	3e-4	3e-4	3e-4	3e-4	3e-4	3e-4
Encoder Learning rate	3e-4	3e-4	3e-4	3e-4	3e-4	3e-4
Maximum episode length	20	200	200	200	200	200
MinMI loss weight λ	25	10	50	50	25	25
behavior regularization	50	50	50	50	50	50
Discount factor	0.9	0.99	0.99	0.99	0.99	0.99

E Comparison Offline Test Results

Offline testing is an ideal evaluation method where the context used is sampled from the pre-collected offline data in the test task, thereby disregarding the context shift problem. To assess our performance in the offline test phase, we adopt the same approach as the training environment and collect offline datasets as context on the testing environment.

We compare CSRO with other methods across all six environments and plot the mean and standard deviation curves of returns based on 8 random seeds in Figure6. In most environments, CSRO demonstrates competitive performance compared to other baselines. Experimental results demonstrate the effectiveness of our algorithm, even in the absence of addressing the context shift problem.

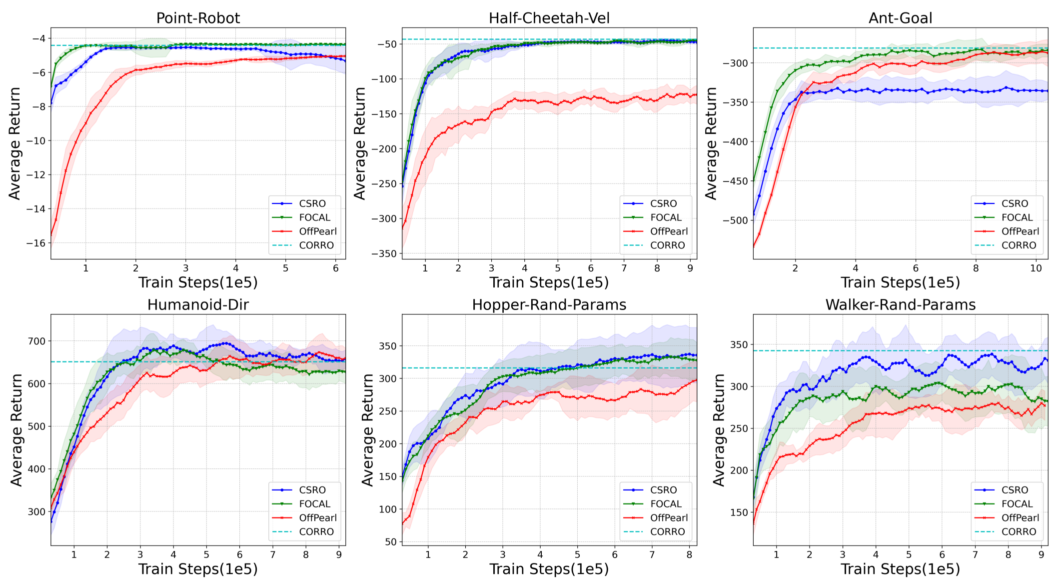

Figure 6: Compared to other OMRL methods, CSRO's offline testing averages returns on unseen testing tasks

F Ablation Offline and Online Test

Under the offline test scenario that ignores the context shift problem, the algorithm can achieve its highest performance. We compare the performance of the online testing method that uses the non-prior context collection strategy(Np) with offline testing. We conduct experiments on six environments and plot the mean and standard deviation curves of returns across 8 random seeds in Figure7.

In most environments, the performance of CSRO that uses Np is close to the offline test. There exists a gap between Point-Robot and Ant-Goal environments due to the particularly severe and challenging context shift problem in these two environments. However, our approach still outperforms previous methods. The experimental results highlight the efficacy of our approach in addressing the context shift issue, albeit with some remaining challenges in these specific environments.

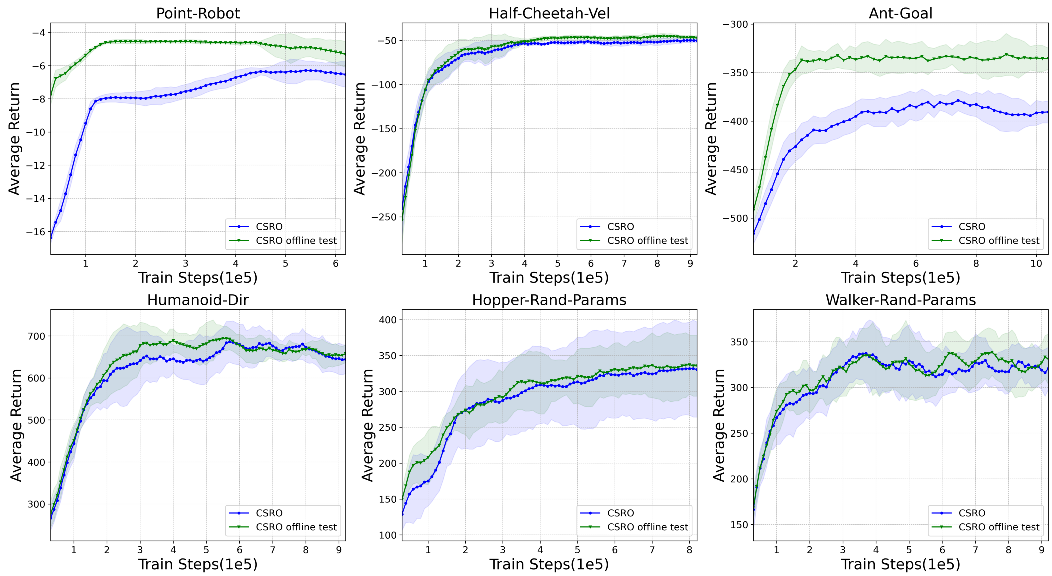

Figure 7: The average return of the offline test and the online test that uses the non-prior context collect strategy on unseen test tasks.

G Visualize Contexts and Trajectory of Online Test

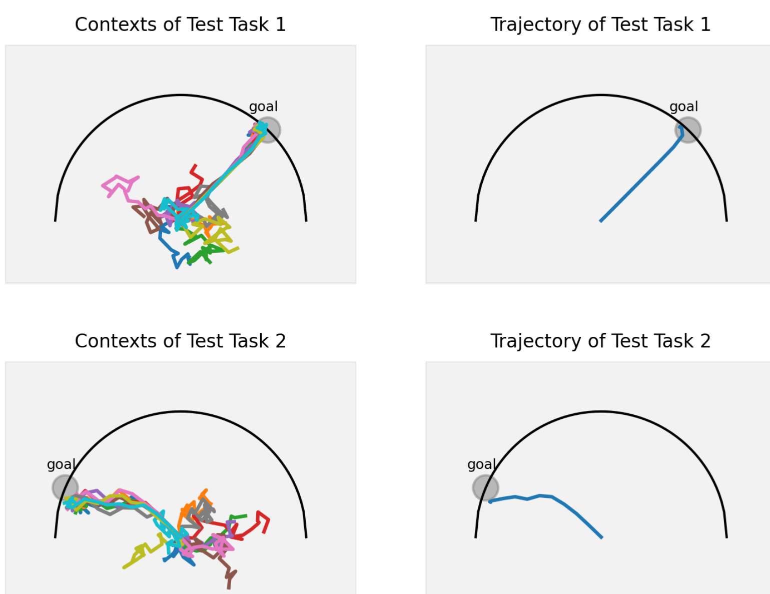

Figure 8: Visualization of contexts and trajectory after using the non-prior context collection strategy in the Point-Robot environment.

Lastly, we further study the non-prior context collection strategy. Figure 8 showcases two different tasks in the Point-Robot environment. We illustrate the context gathered through the non-prior context collection strategy and the corresponding trajectory navigation. Observing the visualizations, we notice that the agent's perception of the task improves after random exploration. Subsequent explorations enhance the agent's understanding of the environment, enabling it to accurately navigate toward the goal.