Appendix: CEIP: Combining Explicit and Implicit Priors for Reinforcement Learning with Demonstrations

This Appendix is organized as follows. First, we reiterate and highlight our key observations. In Sec. A, we then provide the pseudocode for training the implicit prior and the downstream reinforcement learning. Afterwards, we provide additional implementation details of the proposed method and major baselines in Sec. B, and additional details of experimental settings in Sec. C. In Sec. D, we provide additional experimental results and ablation studies. In Sec. E, we describe the computational resources consumed by and the training time of each method. Finally, in Sec. F, we describe the licenses of assets which we used to develop our code.

The key findings of our work include the following:

- **Is a task-specific flow necessary?** In environments where the episode length is relatively short and the dynamics are relatively simple, CEIP works better without the task-specific flow, explicit prior, and push-forward technique as the training complexity is unnecessarily increased. This is shown in Sec. D.2.
- **When is a task-specific flow helpful?** In environments where some tasks of the task-specific dataset are not part of the task-agnostic dataset, a flow trained on the task-specific dataset improves performance. This is shown in Sec. D.3.
- **How related should the tasks in the task-agnostic dataset be to the task at hand?** For both PARROT and CEIP, more related data in the task-agnostic dataset are beneficial. However, CEIP can automatically discover and compose related flows; in contrast, PARROT works better only when the dataset fed into the normalizing flow is manually picked to be more relevant to the target task. This is shown in Sec. D.2.
- **Will ground-truth labels help the performance of CEIP?** Ground-truth labels will sometimes improve the performance of CEIP; however, this is not always the case. This is shown in Sec. D.3.
- **How will simple baselines, e.g., behavior cloning and replaying demonstrations do?** We find those simple baselines to not work very well, which indicates the non-trivial nature of our testbed. However, introducing an explicit prior will significantly improve the performance of behavior cloning. This is shown in Sec. D.3.
- **How robust is CEIP with respect to the precision of task-specific demonstrations?** Similar to prior work such as FIST, imprecise task-specific demonstrations will affect performance. Nevertheless, we find CEIP to be more robust than prior work. This is shown in Sec. D.3.
- **What is the impact of using an explicit prior in PARROT?** PARROT results improve when an explicit prior is used, which further supports the design of CEIP. See ablation studies in Sec. D.2 and Sec. D.3.

To easily compare CEIP to baselines, we summarize all results achieved at the end of the training process for the proposed method and baselines on all testbeds in Table 1. To better understand the behavior of each method, please also see the code and videos of trajectories which are part of this Appendix.

A Algorithm Details

Alg. 1 provides the pseudocode for training the implicit prior. Alg. 2 illustrates how we use the policy $\pi(z|s)$ and the flows to compute the real-world action a, when an explicit prior is available (i.e., condition $u = [s, s_{\text{next}}]$) and when using the push-forward technique.

B Additional Implementation Details

We provide our code in the github repository `https://github.com/289371298/CEIP` for reference.

Environment	CEIP (ours)	PARROT+TA	PARROT+TS	FIST	SKiLD
Fetchreach-4.5	$-\mathbf{10.03}^{\dagger}\pm 0.64$	-19.33 ± 9.59	-20.30 ± 10.62	-34.80 ± 8.33	-39.91 ± 0.14
Fetchreach-5.5	$-\mathbf{9.76}^{\dagger}\pm 0.47$	-20.49 ± 11.51	-14.32 ± 7.53	-39.86 ± 0.50	-38.38 ± 2.81
Fetchreach-6.5	$-\mathbf{9.08}^{\dagger}\pm 0.36$	-14.52 ± 9.44	-18.52 ± 2.34	-38.30 ± 5.28	-40.00 ± 0.00
Fetchreach-7.5	$-\mathbf{10.29}^{\dagger}\pm 0.67$	$-\mathbf{10.34}\pm 0.79$	$-\mathbf{10.24}\pm 0.69$	-39.87 ± 0.72	-38.45 ± 2.67
Kitchen-SKiLD-A	$\mathbf{4.00}\pm 0.00$	2.52 ± 0.96	0.51 ± 0.46	2.70 ± 1.23	0.06 ± 0.10
Kitchen-SKiLD-B	$\mathbf{3.93}\pm 0.08$	1.13 ± 0.35	1.25 ± 0.60	1.17 ± 0.93	0.48 ± 0.48
Kitchen-FIST-A	$\mathbf{3.95}\pm 0.05$	1.94 ± 0.07	2.40 ± 0.31	0.33 ± 0.70	0.67 ± 1.15
Kitchen-FIST-B	$\mathbf{3.89}\pm 0.07$	0.00 ± 0.00	1.85 ± 0.05	1.20 ± 0.54	0.00 ± 0.00
Kitchen-FIST-C	$\mathbf{3.92}\pm 0.06$	0.96 ± 0.06	2.07 ± 0.23	0.00 ± 0.00	0.33 ± 0.57
Kitchen-FIST-D	$\mathbf{3.94}\pm 0.07$	1.92 ± 0.06	2.27 ± 0.24	0.53 ± 0.50	1.67 ± 0.58
Office	$\mathbf{6.33}\pm 0.30$	2.05 ± 0.31	1.97 ± 0.22	5.50 ± 1.12	0.50 ± 0.50

Table 1: Summary of the results of each method on all environments at the end of training (higher is better). For CEIP (our method), we are not using the explicit prior, task-specific single flow, and push-forward technique for fetchreach (which is denoted by '†'). We use all of them for the other experiments. For PARROT, we are not using the explicit prior, task-specific single flow, and push-forward technique, as all of them are our contributions. However, as shown in ablation study in Sec. D, these components are general and can be used to improve the performance of PARROT.

B.1 CEIP

B.1.1 Architecture. We use slightly different architectures for fetchreach and kitchen/office, because the number of dimensions of the states and actions in fetchreach is much smaller than that in the other two experiments. Moreover, the size of fetchreach is much smaller too. Hence, a smaller network is used for fetchreach to prevent overfitting.

Fetchreach. For each single flow, we use a pair of simple Multi-Layer Perceptron (MLP), one for $c_i(u)$ and the other one for $d_i(u)$. Each network has two hidden layers of width 32 for each single flow. The number of dimensions for the feature is 20 (with explicit prior) or 10 (without explicit prior). For the combination of flows, we use one fully-connected neural net with two hidden layers of width 32, which outputs both μ and λ. μ has an additional softplus activation and a 10^{-4} offset. If not otherwise specified, all activation functions in this section are ReLU.

Kitchen and Office. The architecture for the kitchen and office environments is roughly the same as that for the fetchreach environment. The difference is that we use three hidden layers of width 256 for c_i and d_i of each single flow, and that we use two hidden layers of width 64 for μ and λ. Also, we use a batchnorm function before each ReLU activation. See Fig. 8 for an illustration.

B.1.2 Flow Training. We use the standard flow training method [24] for training the task-agnostic and task-specific single flows $f_1, \ldots, f_{n+1}$, which is to maximize the (empirical) log-likelihood

$$\max_{f_i} \mathbb{E}_{(u,a)\sim D_i} \log p_a(a|u),$$

where $\log p_a(a|u) = \log p_z(f_i^{-1}(a;u)) + \log\left|\frac{\partial f_i^{-1}(a;u)}{\partial a}\right| = \log p_z(f_i^{-1}(a;u)) - c_i(u)^T\mathbf{1}$, and

$$f_i^{-1}(a;u) = z = \frac{a - d_i(u)}{\exp\{c_i(u)\}}. \tag{4}$$

Here, $c_i(u) \in \mathbb{R}^q$, $d_i(u) \in \mathbb{R}^q$ are trainable deep nets. The exp function and division are applied elementwise. We use a standard normal distribution over the latent space, i.e., $p_z = N(0, I)$. Moreover, we use maximization w.r.t. f_i to denote maximization w.r.t. the parameters of the deep nets c_i, d_i. To train the combined flow, we use a similar loss function to Eq. (4), i.e.,

$$\max_{f_{\text{TS}}} E_{(u,a)\in D_{\text{TS}}} \log p_a(a|u), \text{ where } \log p_a(a|u) = \log p_z(f_{\text{TS}}^{-1}(a;u)) + \log\left|\frac{\partial f_{\text{TS}}^{-1}(a;u)}{\partial a}\right|. \tag{5}$$

Again, p_z is a standard normal distribution. Here, maximization w.r.t. f_{TS} denotes maximization w.r.t. the parameters of the deep nets μ and λ as shown in Fig. 8.

Algorithm 1: Training of Implicit Prior

```
Input  : dataset $D_1, D_2, ..., D_n, D_{\text{TS}}$
Input  : training epoch for single flow $M$, for combination $M_2$
Input  : learning rate $a$
Output : normalizing flow $f_{TS}$, parameterized by $\mu(u)$, $\lambda(u)$, $c_i(u)$, and $d_i(u)$ where
         $i \in \{1, 2, \ldots, n+1\}$
begin
    // Training single flows
    for $i \in \{1, 2, \ldots, n+1\}$ do                  // recall that we denote $D_{\text{TS}} = D_{n+1}$
        for $j \in \{1, 2, \ldots, M\}$ do                // for loop over epochs
            foreach $(u, a) \sim D_i$ do                  // for each data point
                $z_0 \leftarrow \frac{a - d_i(u)}{\exp\{c_i(u)\}}$    // elementwise division
                $L = \log p_z(z_0) - c_i(u)^T \mathbf{1}$    // $z \sim N(0, I)$
                $c_i \leftarrow c_i + a \times \frac{\partial L}{\partial c_i}$
                $d_i \leftarrow d_i + a \times \frac{\partial L}{\partial d_i}$
    // Training the combination of flows
    for $j \in \{1, 2, \ldots, M_2\}$ do                 // for loop over epochs
        foreach $(u, a) \sim D_{TS}$ do                  // for each data point
            $\mu_0 \leftarrow \mu(u)$
            $\lambda_0 \leftarrow \lambda(u)$
            $c = \sum_{i=1}^{n+1} \mu_{0,i} c_i(u)$
            $d = \sum_{i=1}^{n+1} \lambda_{0,i} d_i(u)$
            $z_0 \leftarrow \frac{a - d}{c}$                 // elementwise division
            $L \leftarrow \log p_z(z_0) - c^T \mathbf{1}$    // $z \sim N(0, I)$
            $\mu \leftarrow \mu + a \times \frac{\partial L}{\partial \mu}$
            $\lambda \leftarrow \lambda + a \times \frac{\partial L}{\partial \lambda}$
```

Training Hyperparameters. To train each single flow, we use 1000 epochs on each cluster of the task-agnostic dataset $D_1, D_2, \ldots, D_n$ and task-specific D_{n+1} with a batchsize of 256. We use the Adam [21] optimizer with a learning rate of 0.001 and a gradient clipping at norm 10^{-4}. For each dataset, we randomly draw 80% of the state-action pairs / transitions (regardless of which trajectory they are in) as the training set and use the rest for validation. We use an early stopping that triggers when the current number of batches fed into the network is greater than 1000 (fetchreach) or 4000 (kitchen/office) and the validation loss does not improving during the last 20% of the batches. The model with the lowest loss on the validation set is stored and utilized. Each flow is trained separately and parameters are not shared. Note, we did not optimize the implementation for efficiency, but this can be accelerated via parallelization.

B.1.3 Reinforcement Learning. We use a well-established reliable implementation of RL algorithms, stable-baselines3[2], to carry out reinforcement learning. As stable-baselines3 needs a bounded action space, we set the latent (action) space $\mathcal{Z}$ of the RL agent $\pi(z|s)$ to be $[-3, 3]$ on each dimension.

B.2 PARROT

PARROT can be seen as a special case of CEIP, where the number of single flows is 1 and $\mu = 1, \lambda = 1$. This single flow is trained on all task-agnostic data. The original PARROT does not use an explicit prior or a push-forward technique, which are our contribution in this work. But these components can be added to PARROT in the same way as they are used in our method. For a fair comparison, PARROT uses exactly the same architecture and training paradigm of a single flow as CEIP.

---

[2] https://stable-baselines3.readthedocs.io/en/master/

Algorithm 2: Step Function of Reinforcement Learning

```
Input  : current state $s$, RL policy $\pi(z|s)$
Output : action in actual action space $a$
begin
    // $r$ is the last step referred to in the trajectory
    if A new episode begins then
        foreach $\tau \in D_{TS}$ do
                                    // reset last reference in each trajectory
            $r(\tau) \leftarrow -1$
    foreach $\tau \in D_{TS}$ do
        foreach $(s_{key}, a, s_{next}) \in \tau$ do
                                    // Assume this is the $i$-th step
            if $(s_0, j_0, \tau_0)$ undefined or $(s_{key} - s)^2 + [i \leq r(\tau)] < (s_0 - s)^2 + [j_0 \leq r(\tau_0)]$ then
                                    // The second term is an indicator function
                $s_0 \leftarrow s_{\text{key}}$
                $j_0 \leftarrow i$
                $\tau_0 \leftarrow \tau$
    $r(\tau_0) \leftarrow j_0$          // update last reference for the chosen trajectory
    $\mu_0 \leftarrow \mu(u)$
    $\lambda_0 \leftarrow \lambda(u)$
    $c \leftarrow \sum_{i=1}^{n+1} \mu_{0,i} c_i(u)$
    $d \leftarrow \sum_{i=1}^{n+1} \lambda_{0,i} d_i(u)$  // get transformation from latent to action space
    Sample $z_0$ from RL policy $\pi(z|s)$
    $a \leftarrow c \odot z_0 + d$
```

B.3 SKiLD

As CEIP, SKiLD also uses an implicit prior. However, different from CEIP which is flow-based, SKiLD uses a VAE-based architecture where the latent space is for an action sequence called "skill," and the decoder of the VAE maps actions from latent space to actual action sequences. In addition, SKiLD uses two implicit priors that take the current state as input and mimic the state-action sequence encoder, one for the entire task-agnostic dataset and the other for the task-specific dataset. To utilize both priors, a discriminator that takes the current state as input is trained. This discriminator approximates the confidence of the task-specific prior. A reward shaping in the downstream RL stage is then used to drive the agent back to states similar to those in the task-specific dataset, where the discriminator reports higher confidence for the task-specific prior. The reward shaping also encourages the RL agent to form a policy similar to the task-agnostic prior when the confidence is low, and a policy similar to the task-specific prior when the confidence is high. SKiLD does not use an explicit prior or the push-forward technique. However, in a similar spirit, the reward-shaping mechanism encourages the agent to visit states similar to those in the task-specific dataset. We follow the settings described by SKiLD [34], except for some minor modifications to better adapt SKiLD to the environments. These modifications are discussed next.

We change the configuration mostly for the fetchreach environment, because skills with 10 steps are too long for the fetchreach environment with 40 steps in an episode, and because the number of dimensions of the data and the number of datapoints are much smaller than they are in other environments. Therefore, we shorten a skill from 10 to 3 steps, and reduce the size of the skill prior and posterior, which are now 3-layer MLPs with width 32 instead of the original 5-layer MLP with width 256. Also, as the dataset size decreases, we change the number of epochs. For the skill prior, we use a batchsize of 20, and train for 7500 cycles over the task-agnostic dataset (for each cycle, one sub-trajectory of length 3 is sampled for each trajectory).[3] For the posterior, we use 30K

---

[3] See "RepeatedDataLoader" in SKiLD's official repository `https://github.com/clvrai/spirl/blob/5cd34db7c5e48137550801bf5ac3f8c452590e2c/spirl/utils/pytorch_utils.py` and `https://github.com/clvrai/spirl/blob/5cd34db7c5e48137550801bf5ac3f8c452590e2c/spirl/train.py` for the meaning of "cycles."

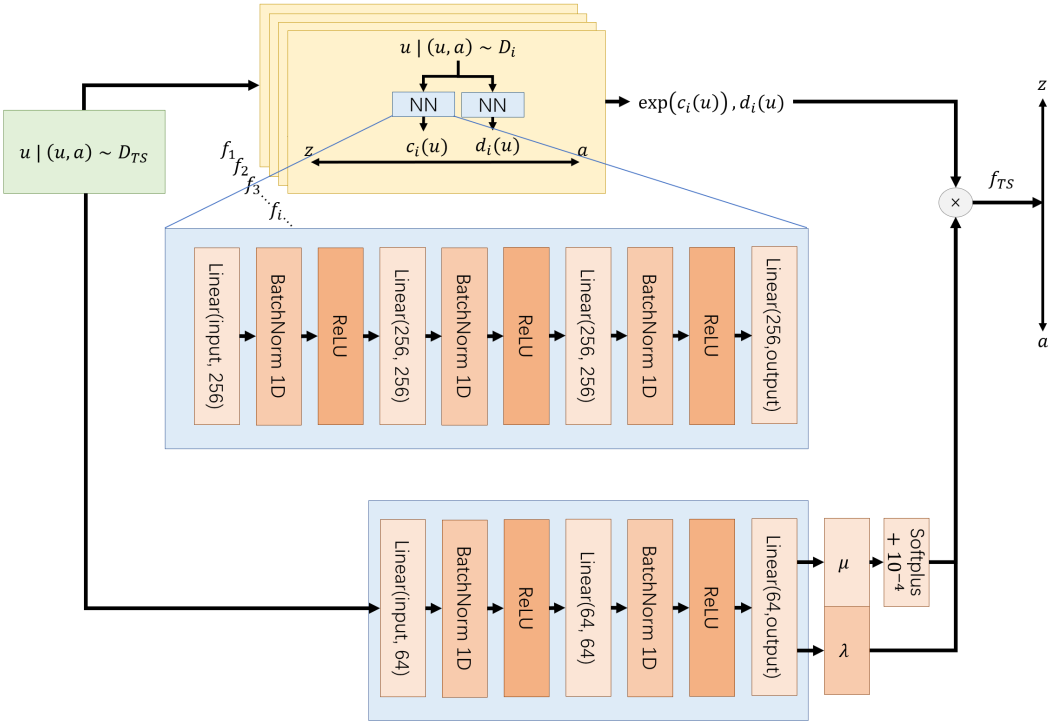


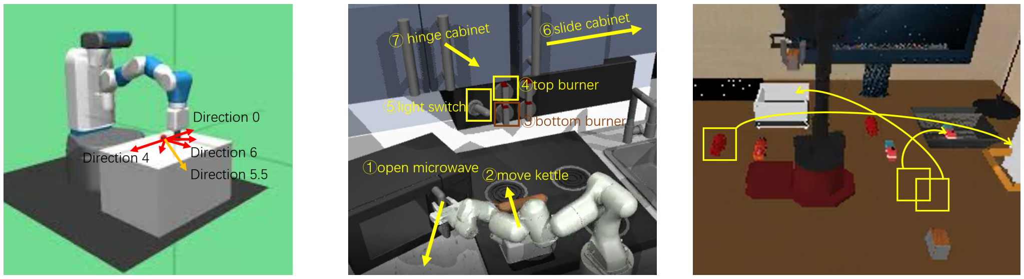
Figure 8: Illustration of our architecture used for kitchen and office environments.

training cycles over the task-specific dataset. The discriminator is trained for 300 epochs, sampling both task-agnostic and task-specific datasets. For RL, we use the settings employed for the kitchen environment in the original paper of SKiLD, where the hyperparameter $\alpha = 5$ is fixed. For the kitchen and office environments, we follow the original paper and use the same architecture: a 5-layer MLP with width 128 for the skill prior and posterior, a linear layer and long-short term memory (LSTM) with width 128 for the encoder, and a 3-layer MLP with width 32 for the discriminator. The training paradigm is almost the same as the one in the original paper, except that the cycles over the task-specific dataset are increased due to a decreased dataset size. We also use exactly the same RL settings as the original paper.

B.4 FIST

Conceptually, FIST can be viewed as SKiLD combined with an explicit prior. However, FIST uses pure imitation learning, while SKiLD includes a reinforcement learning phase. Also, different from SKiLD, FIST only uses one prior, which is first trained on the task-agnostic dataset and then fine-tuned on the task-specific dataset. To decide which key is the "closest" to the query in dataset retrieval, FIST conducts contrastive learning for the distance metric between states using the InfoNCE loss [53], where the positive sample is the future state (exactly H steps later, where H is the length of a skill) of a state in a dataset, and the negative samples are the future states of other states in the same dataset. This metric is trained on the combined task-agnostic and task-specific data. However, in our experiment we found that using Euclidean distance as the metric suffices to achieve good result.

For FIST, we mostly follow the settings described in the original paper [16], with the exception of some minor modifications. Similar to SKiLD, on fetchreach we use 3 steps for a skill, and a lighter architecture for the skill prior and posterior network with 2 hidden layers of width 32 instead of 5 hidden layers of width 128 for the other experiments. We use the settings for the kitchen environment in the original paper for all other experiments. Moreover, the original FIST is occasionally unstable at the beginning of skill prior training in the office environment, due to an initial loss being too large. To remove this instability, we add gradient clipping at norm 10^{-3} during the first 100 steps.



(a) Fetchreach (b) Kitchen (c) Office

Figure 9: Illustration of each environment. For fetchreach, the task-agnostic dataset consists of demonstrations which move the gripper in the directions of the red arrows, and the task-specific dataset contains demonstrations which move the gripper in the directions of the yellow arrows. For the kitchen environment, the agent needs to complete four out of seven tasks mapped on the picture in the correct order. For the office environment, the agent needs to put items in the container as illustrated in the figure, using the correct order.

C Additional Details of Experimental Settings

In this section, we introduce additional details related to the environment settings and dataset settings for each environment.

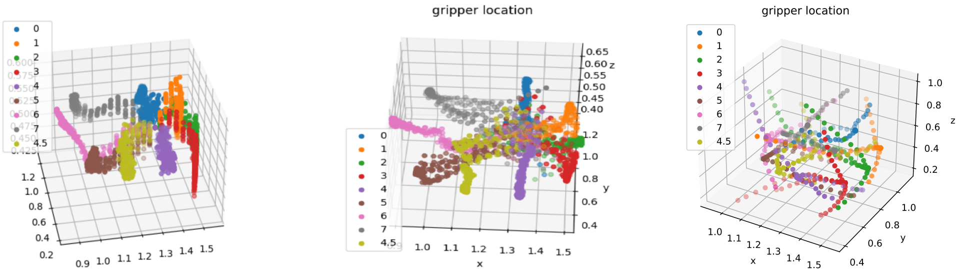
C.1 Fetchreach

Environment Settings. In our version of fetchreach (illustrated in Fig. 9(a)), we need to train a robot arm to move its gripper to a given but unknown location as quickly as possible, and stay there once the goal is reached. The state is 10-dimensional, with the first three dimensions describing the current location of the gripper. The other dimensions are the openness of the gripper and the current velocity. For each of the 40 steps, the agent needs to output a 4-dimensional action $a \in [-1, 1]^4$, where the first three dimensions are the direction which the gripper is moving to and the fourth is the openness of the gripper (unused in this experiment). The agent receives a reward of 0 if the Euclidean distance between the gripper and the target is smaller than 0.05, and -1 otherwise. A perfect agent should achieve a reward of around -10. The goal denoted as direction d (e.g., direction 4.5) is generated by first assigning a direction $d \in [0, 8)$, then selecting the goal with the Euclidean distance being 0.3 away and the azimuth being $\frac{d\pi}{4}$, and finally applying a uniform noise of $U[-0.015, 0.015]$ on each of the three dimensions. In order to test the robustness of the algorithms and increase difficulty, before each episode begins, we first sample a random action from a normal distribution, and then let the agent execute the action for x steps, where $x \sim U[5, 20]$. This greatly increases the variety of the trajectories, as shown in Fig. 10.

Dataset Settings. The dataset is acquired by first training an RL agent with soft actor critic (SAC) which receives the negative current Euclidean distance as a reward until convergence, and then sampling trajectories on the trained RL agent. For each direction in $\{0, 1, 2, \ldots, 7\}$, 40 trajectories (1600 steps) are sampled. For each direction in $\{4.5, 5.5, 6.5, 7.5\}$, 4 trajectories (160 steps) are sampled.

C.2 Kitchen-SKiLD

Environment Settings. We adopt the same setting as SKiLD [34] and FIST [16], where an agent needs to finish four out of seven tasks in the correct order. The tasks are: open the microwave, move the kettle, turn on the light switch, turn on the bottom burner, turn on the top burner, slide the right cabinet, and hinge the left cabinet. The agent needs to complete all four tasks within 280 timesteps, and a $+1$ reward is given when one task is completed. The state is 60-dimensional, where the first 9 dimensions describe the current location of the robot, the next 21 dimensions describe the current object location (the unrelated objects will be zeroed out), and the rest are constant and describe the initial location of each object. The action a is 9-dimensional where $a \in [-1, 1]^9$, which controls the



(a) No randomization (b) Random action at each step (c) Our randomization

Figure 10: Illustration of expert trajectories with no start randomization (left), sampling a randomized action for 10 steps (middle), and our way of randomization (right). Note that our randomization greatly increases the variation of the trajectories.

Task	Subtask Missing	Target Task	Dataset Size
A	Top Burner	Microwave, Kettle, Top Burner, Light Switch	66823 / 210
B	Microwave	Microwave, Bottom Burner, Light Switch, Slide Cabinet	52898 / 200
C	Kettle	Microwave, Kettle, Slide Cabinet, Hinge Cabinet	53576 / 246
D	Slide Cabinet	Microwave, Kettle, Slide Cabinet, Hinge Cabinet	45267 / 246

Table 2: List of four different settings in Kitchen-FIST. The dataset size format is "task-agnostic / task-specific." The dataset size is counted in state-action pairs.

joints of the arm. A uniform noise of $[-0.1, 0.1]$ is applied to the observation of the robot in every step.[4]

Dataset Settings. We use exactly the same task-agnostic dataset as SKiLD, which includes 33 different task sequences with a total of 136950 state-action pairs generated by 'relay policy learning' [14]. We choose the first trajectory from the task-specific dataset of SKiLD as our task-specific dataset, which includes 214 state-action pairs for task A and 262 state-action pairs for task B. Task A is "move the kettle, turn on the bottom burner, turn on the top burner, and slide the right cabinet;" task B is "open the microwave, turn on the light switch, slide the right cabinet, and hinge the left cabinet."

C.3 Kitchen-FIST

Dataset Settings. We use exactly the same task-agnostic dataset as FIST [16]. There are four pairs of task-agnostic and task-specific datasets, which are illustrated in Table 2.

C.4 Office

Environment Settings. We adopt the settings from SKiLD, where we need to train a robot arm to put three out of seven items into three containers (illustrated in Fig. 9(c)), which are the two trays and one drawer. The robot arm receives a 97-dimensional state and has 8 dimensions of actions which control the position and angle of the gripper, as well as the continuous gripper actuation. There are 8 subtasks in the experiments: pick up the first/second item, drop the first/second item in the right place, open the drawer, pick up the third item, drop the third item correctly, and close the drawer. The agent will receive a $+1$ reward upon completion of each subtask. The episode length is at most 350 steps, and will finish as soon as all the tasks are finished.

Dataset Settings. We use the same task-agnostic dataset and a subset of the task-specific dataset. We use a subset of the task-specific dataset because SKiLD, CEIP, PARROT+TS, and FIST can all solve the problem very well with the whole task-specific dataset, making it hard to compare. The task-agnostic dataset contains 2417 trajectories from randomly-generated tasks, which has $7 \times 6 \times 5 = 210$ possibilities and contains 456033 state-action pairs. The task-specific dataset contains 5 trajectories with 1155 state-action pairs.

---

[4] See SKiLD's official repository https://github.com/kpertsch/d4rl/blob/master/d4rl/kitchen/adept_envs/franka/robot/franka_robot.py for details.

Method	Task-specific flow	Explicit prior	Push-forward
CEIP			
CEIP+EX		✓	
CEIP+EX+forward		✓	✓
CEIP+TS	✓		
CEIP+TS+EX	✓	✓	
CEIP+TS+EX+forward	✓	✓	✓

Table 3: Abbreviations for variants of CEIP. See Fig. 4 for difference between CEIP, CEIP+2way, and CEIP+4way; the latter two only appear in the fetchreach ablation.

Method	Use task-agnostic data	Use task-specific data	Explicit prior	Push-forward
PARROT+TA	✓			
PARROT+TS		✓		
PARROT+(TS+TA)	✓	✓		
PARROT+TA+EX	✓		✓	
PARROT+TS+EX		✓	✓	
PARROT+(TS+TA)+EX	✓	✓	✓	
PARROT+TA+EX+forward	✓		✓	✓
PARROT+TS+EX+forward		✓	✓	✓
PARROT+(TS+TA)+EX+forward	✓	✓	✓	✓
PARROT+2way+TS	two most related directions	✓		
PARROT+4way+TS	four most related directions	✓		
PARROT+2way	two most related directions			

Table 4: Abbreviations for variants of PARROT. All variants of PARROT only use a single flow for all data, which is the key difference between CEIP and PARROT with explicit prior. “2way” and “4way” only appear in the fetchreach environment where there are 8 directions in the task-agnostic dataset.

D Additional Experimental Results

This section includes additional experimental results, which are ablation studies and auxiliary metrics that help to better understand the properties of different methods. See the beginning of the Appendix for a summary of the key findings. Please also see our supplementary material for sample videos of each trained method on the kitchen and office environments.

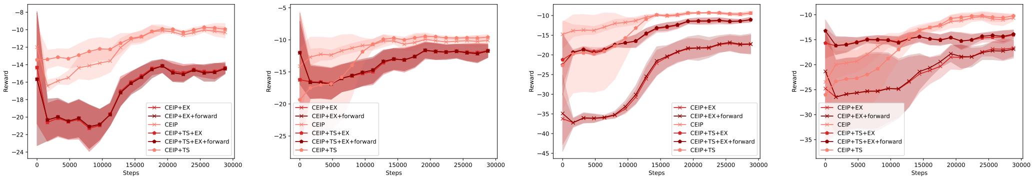
D.1 Abbreviations for Ablation Tests

In our experiments, we test multiple variants of CEIP and PARROT for a more thorough ablation. To more easily differentiate the variants of both methods with different components, we will use abbreviations as listed in Table 3 and Table 4.

D.2 Fetchreach

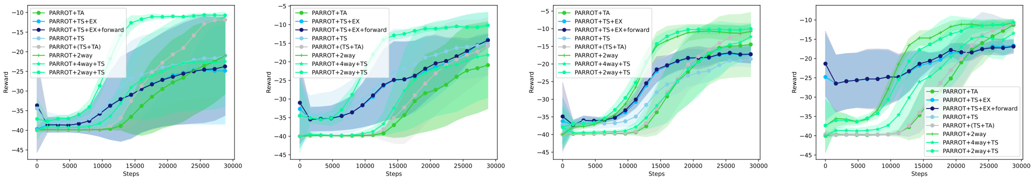
Ablation on components of our method. Fig. 11 shows the ablation study on different components of our method. Results for our method show that using an explicit prior and the push-forward technique slows down the reward growth during RL training, if applied on a relatively easy and short-horizon environment. This is likely because we unnecessarily add training complexity to an environment with a relatively easy task, short horizon, and well-clustered task-agnostic datasets. However, our method with those components still works better than many baselines and performs well given more steps.

Ablation on components and data relevance in PARROT. To better understand the properties of PARROT, we ablate the data used when training PARROT (see Fig. 12). We select a subset of the task-agnostic data that is more relevant to the task-specific dataset, and study how data in the



(a) Direction 4.5 (b) Direction 5.5 (c) Direction 6.5 (d) Direction 7.5

Figure 11: Ablation on architecture and components of our method. We observe that the reward actually grows slower when using the task-specific single flow, explicit prior, and push-forward technique, likely because the training complexity is unnecessarily increased.



(a) Direction 4.5 (b) Direction 5.5 (c) Direction 6.5 (d) Direction 7.5

Figure 12: Ablation on the data used for PARROT. “2way” and “4way” mean that we feed the two/four directions in the task-agnostic dataset that are closest to the target direction (e.g., if the target direction is 4.5, we refer to data from directions 4 and 5 as “2way,” and data from directions $3, 4, 5, 6$ as “4way”). Note, PARROT with the task-specific data and “2way” data is significantly better than other variants of PARROT. However, PARROT is only improved when such data are selected manually, while our method can automatically combine the flows and select useful priors. Also, PARROT with “2way” data from the task-agnostic dataset but without task-specific dataset works well, but it is unstable, which emphasizes the importance of the task-specific dataset even if it is small.

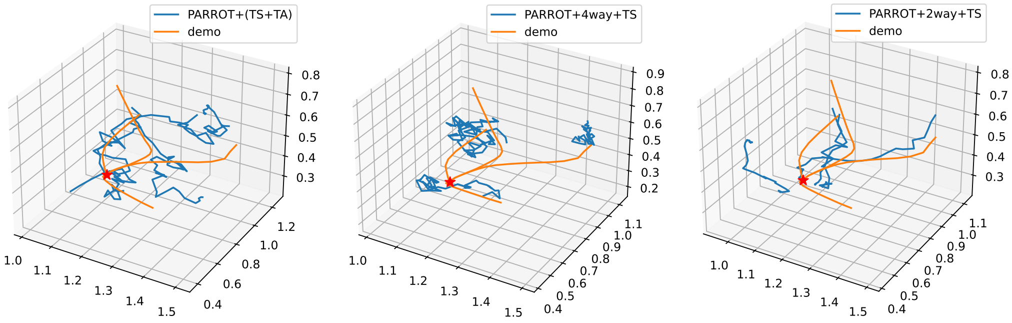
task-agnostic dataset with different levels of relevance to the downstream task affect results. We also test the effect of the explicit prior and the push-forward technique using task-specific data only. The results can be summarized as follows: 1) using the explicit prior and the push-forward technique slows down the reward growth during RL training, if applied on a relatively easy and short-horizon environment; 2) selecting more relevant data for PARROT is an effective way to improve PARROT, which supports our motivation to combine the flows to select the most useful prior.

Illustration of trajectories. To validate the effect of the implicit prior, Fig. 13 shows the trajectories generated by our method and PARROT without any RL training. We clearly observe that the trajectories generated by PARROT become more accurate when the data are more related (from left to right), which is achieved by manual selection but can be done automatically in CEIP. Our method improves when more flows are used (from right to left), as more flows increase expressivity.

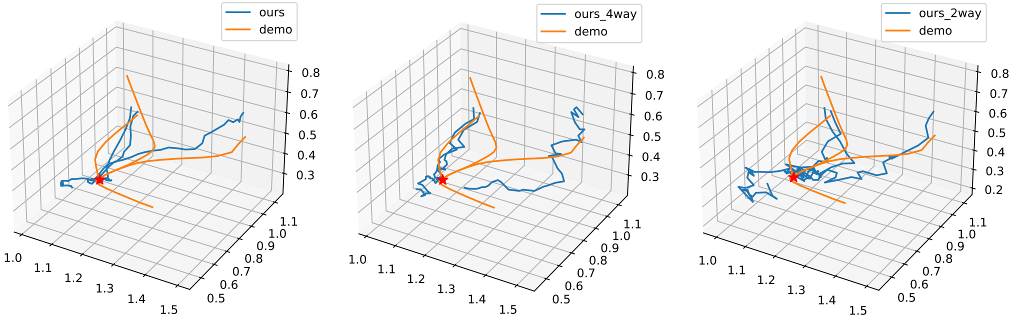
Fig. 14 illustrates the trajectories generated by each method after RL training. As shown in the figure, our method exhibits smoother trajectories after RL training, enabling the agent to reach its goal faster. FIST, SKiLD, and naïve RL fail to generate trajectories that steadily converge to the goal. Although PARROT+(TS+TA) (PARROT with both task-agnostic and task-specific datasets) struggles at the beginning of RL, the prior enables the agent to reach the goal occasionally. Because of this, it learns to rule out other infeasible directions. PARROT+TA fails to reach the goal when the starting location is too far, as it has no idea about how to reach the goal.

D.3 Kitchen and Office

Ablation on components of CEIP. Fig. 15 shows the difference of performance using different architectures for our method. We observe that the explicit prior plays a crucial role in both Kitchen-SKiLD and Kitchen-FIST. Also, for Kitchen-FIST, where one of the target sub-tasks is only part of the task-specific data, the presence of the task-specific single flow f_{n+1} is also crucial for success. We do not find the push-forward technique to help much in this setting.



(a) PARROT+(TS+TA) (c) PARROT+4way+TS (e) PARROT+2way+TS



(b) ours (d) ours with 4 flows (f) ours with 2 flows

Figure 13: Illustration of trajectories generated by our method and PARROT under the direction 4.5 setting in the fetchreach environment **without any RL training**. Both methods do not use the explicit prior or the push-forward technique. Our method does not use the task-specific single flow. For PARROT, 2/4way means the two/four most related directions in the task-agnostic dataset (i.e., directions 4, 5 / 3, 4, 5, 6). For our method, 2/4 flows are trained on the two/four most related directions in the task-agnostic dataset. The orange line is the task-specific dataset for reference. All orange lines converge at the red star, which is the goal.

Ablation on components in PARROT. Fig. 16 shows the difference when different architectures are used. As one target sub-task is completely missing from the task-agnostic data, PARROT+TA fails as expected. Also note that the explicit prior boosts the results of PARROT, making it comparable to our method if given enough training time.

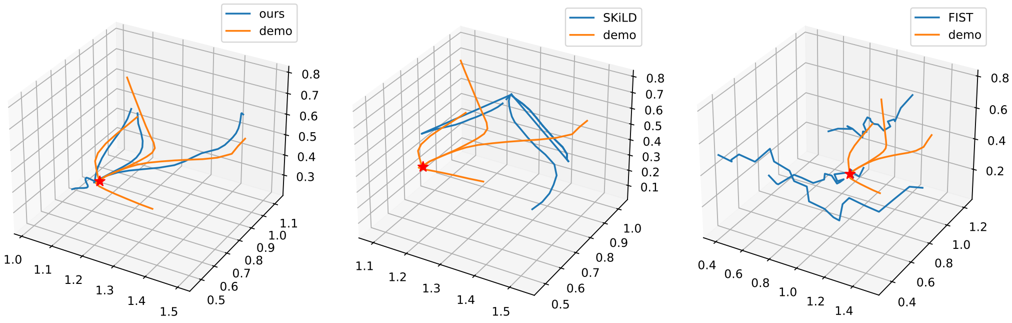
Ablation on the effect of using ground-truth labels. Table 5 and Table 6 show the performance comparison between using ground-truth labels and labels acquired by k-means in the kitchen environment.[5] As we are using 24 labels in the main paper, but not all the task-agnostic datasets have 24 ground-truth labels, we also show the result using ground-truth pruned to 24 labels for a fair comparison. For Kitchen-SKiLD where the number of ground-truth labels is 33, there are exactly 9 labels that have no more than 3 demonstrations. We merge each of them into the label that is next to them in the dictionary order of concatenated task names. For kitchen-FIST where the number of ground-truth labels is x, $x < 24$, we select the $24 - x$ labels with the most demonstrations, and divide them evenly into two halves; each half is a new label. Note, no task information is taken into account when merging.

For readability, we use some suffixes in Table 5 and Table 6 to differentiate variants of CEIP in the "label" column. The meaning of the suffixes are as follows:

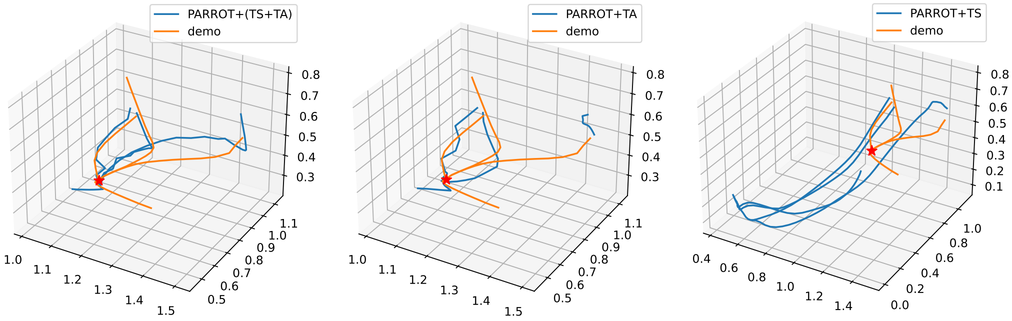
- **GT24**: Ground-truth labels, but merged or split to form 24 labels;
- **GT**: Ground-truth labels; the number of subtasks differs;
- **KM**: K-means labels.

---

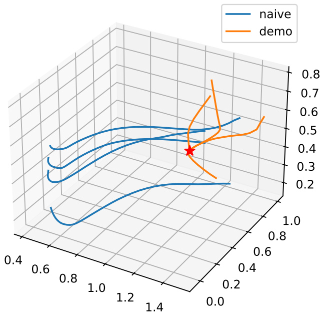
[5] The office environment has 210 ground-truth labels, which is hard to train.



(a) ours (c) SKiLD (f) FIST



(b) PARROT+(TS+TA) (d) PARROT+TA (g) PARROT+TS

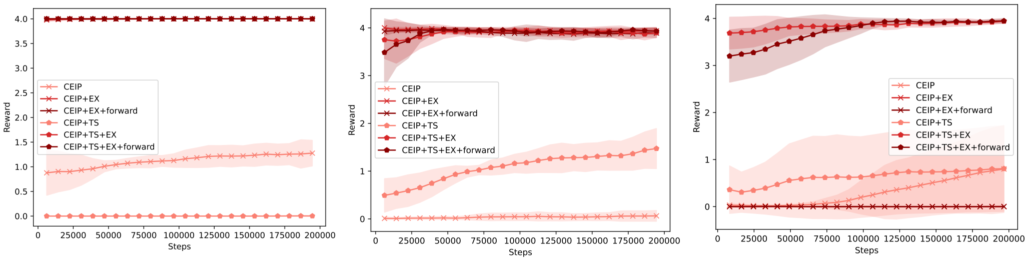


(e) naive

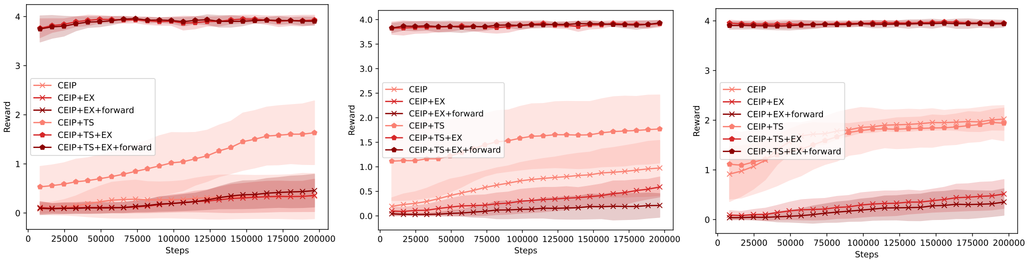
Figure 14: Illustration of trajectories generated by all methods **after RL training**; similar to Fig. 13, the blue curves are the trajectories, the orange curves are the demonstrations, and the red star is the goal. Our method and PARROT do not use the explicit prior or the push-forward technique. Our method does not use the task-specific single flow.

The result suggests that for Kitchen-SKiLD, ground truth (both 24 flows and 33 flows) helps as CEIP with ground-truth labels works better than CEIP with k-means label (Table 5 shows higher reward). For Kitchen-FIST, the reward is similar before and after RL training, and the precise label does not matter.

Performance of behavior cloning and replaying demonstrations. We test behavior cloning and replaying demonstrations (which is duplicating actions regardless of current state) on the kitchen and office environment to see if the task-specific dataset already provides the optimal solution for our testbeds. Table 7 shows the result of vanilla behavior cloning (BC), behavior cloning with explicit prior (BC+EX), with explicit prior and push-forward (BC+EX+forward), and replaying demonstrations (replay). The result shows that: 1) behavior cloning and replay are very brittle, and cannot directly solve our testbed; 2) an explicit prior significantly improves the performance of behavior cloning, which proves the validity of our design.



(a) Kitchen-SKiLD-A (c) Kitchen-SKiLD-B (e) Kitchen-FIST-A



(b) Kitchen-FIST-B (d) Kitchen-FIST-C (f) Kitchen-FIST-D

Figure 15: Ablation on the components of our method in the kitchen environment. For both environments, the presence of an explicit prior greatly enhances the results; for kitchen-FIST where part of the target task disappears from the task-agnostic dataset, a task-specific flow is also very important.

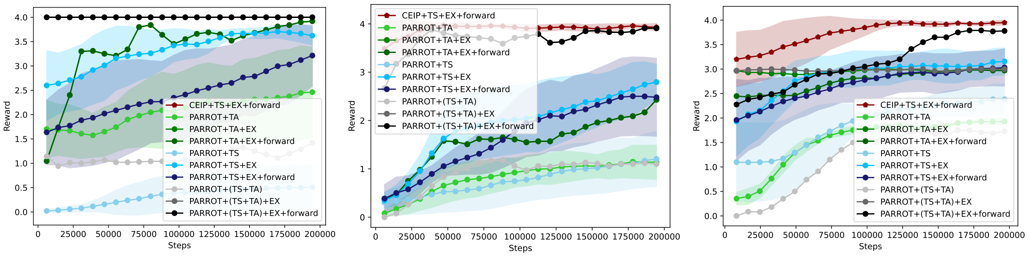
Label	Method	Kitchen-SKiLD-A	Kitchen-SKiLD-B	Kitchen-FIST-A	Kitchen-FIST-B	Kitchen-FIST-C	Kitchen-FIST-D
GT	CEIP+TS+EX	**4**	3.96	3.59	**4**	3.94	3.6
GT	CEIP+TS+EX+forward	**4**	3.95	3	**4**	3.81	3.4
GT24	CEIP+TS+EX	**4**	**4**	3.68	3.76	3.8	3.96
GT24	CEIP+TS+EX+forward	**4**	**4**	3.24	3.75	3.85	3.9
KM	CEIP+TS+EX	**4**	3.81	3.44	3.8	**4**	3.75
KM	CEIP+TS+EX+forward	**4**	3.32	3.41	**4**	3.94	3.76

Table 5: Ground-truth label and k-means label impact for CEIP+TS+EX and CEIP+TS+EX+forward before RL.

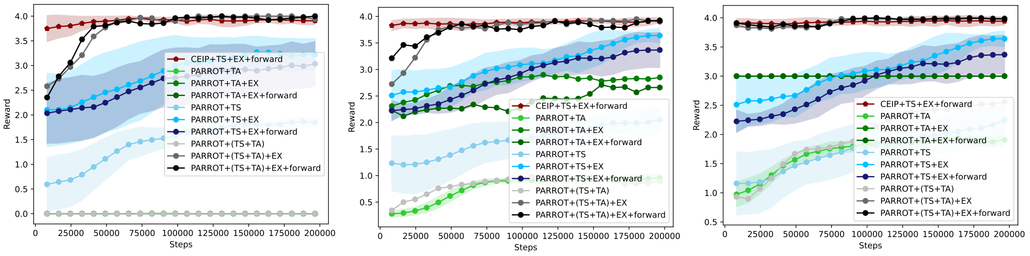
Robustness of CEIP with respect to the precision of task-specific demonstrations. We test the robustness of CEIP and FIST with imprecise task-specific demonstrations in the office environment. The original office environment uses a $[-0.01, 0.01]$ uniformly random noise for the starting position of each dimension for each item in the environment. We increase this noise at test time (which the agent never sees in imitation learning) and summarize the result in Table 8. The result shows that albeit an improvement upon FIST, CEIP is still not robust to imprecise demonstrations, which is a limitation that we discussed in the limitation section.

E Computational Resource Usage

All experiments are conducted on an Ubuntu 18.04 server with 72 Intel Xeon Gold 6254 CPUs @ 3.10GHz, with a single NVIDIA RTX 2080Ti GPU. Under such settings, our method and PARROT+TA require around $1.5-3.5$ hours for training the implicit prior in the kitchen and office environments, depending on early stopping. FIST requires around 40 minutes for prior training and 5 minutes for the other parts. SKiLD requires around 9 hours for prior training, $6-7$ hours for posterior training, and $6-7$ hours for discriminator training. PARROT+TS only needs a few minutes. As for reinforcement learning / deployment, our method on kitchen needs on average 10 minutes for each run on fetchreach, and less than 2 hours for the kitchen environment. For the office environment, we reach a speed of 12 steps per second (including updates); SKiLD and PARROT can reach a speed of 20 steps per second; FIST can reach a speed of $25-30$ steps per second as it has no RL updates.



(a) Kitchen-SKiLD-A (c) Kitchen-SKiLD-B (e) Kitchen-FIST-A



(b) Kitchen-FIST-B (d) Kitchen-FIST-C (f) Kitchen-FIST-D

Figure 16: Ablation results for PARROT in the kitchen environment. For convenience, we also list CEIP+TS+EX+forward for reference. Note, CEIP+TS+EX+forward outperforms all variants of PARROT. Similar to CEIP, PARROT can be improved by using an explicit prior. Note, in Kitchen-FIST-B, PARROT+TA cannot learn anything, because the very first subtask in the target task sequence is missing in the task-agnostic dataset. It can only learn all subtasks before the missing subtask.

Label	Method	Kitchen-SKiLD-A	Kitchen-SKiLD-B	Kitchen-FIST-A	Kitchen-FIST-B	Kitchen-FIST-C	Kitchen-FIST-D
GT	CEIP+TS+EX	**4**	3.87	3.93	3.8	3.94	3.71
GT	CEIP+TS+EX+forward	**4**	3.87	3.9	3.74	3.96	3.93
GT24	CEIP+TS+EX	**4**	**4**	3.92	3.97	3.99	3.87
GT24	CEIP+TS+EX+forward	**4**	**4**	3.99	3.88	3.95	3.96
KM	CEIP+TS+EX	**4**	**4**	3.94	3.92	3.93	3.95
KM	CEIP+TS+EX+forward	**4**	**4**	3.95	3.89	3.92	3.94

Table 6: Ground-truth label and k-means label impact for CEIP+TS+EX and CEIP+TS+EX+forward after RL.

F Dataset and Algorithm Licenses

We developed our code on the basis of multiple environment testbeds and algorithm repositories.

Environment testbeds. We adopt fetchreach using the gym package from OpenAI, which has an MIT license. For the kitchen environment, we are using a forked version of the d4rl package which

Environment	BC	BC+EX	BC+EX+forward	Replay	CEIP+TS+EX+forward
Kitchen-SKiLD-A	$0.02_{\pm 0.04}$	$1.52_{\pm 1.15}$	$2.2_{\pm 0.62}$	$1.0_{\pm 0.82}$	$\mathbf{4.0}_{\pm 0.00}$
Kitchen-SKiLD-B	$0.03_{\pm 0.08}$	$1.03_{\pm 0.90}$	$0.8_{\pm 0.75}$	$0.67_{\pm 0.94}$	$\mathbf{3.93}_{\pm 0.08}$
Kitchen-FIST-A	$0.67_{\pm 0.76}$	$2.17_{\pm 0.06}$	$3.03_{\pm 0.15}$	$2.33_{\pm 0.47}$	$\mathbf{3.95}_{\pm 0.05}$
Kitchen-FIST-B	$0.4_{\pm 0.59}$	$2.13_{\pm 0.47}$	$1.87_{\pm 0.29}$	$0.67_{\pm 0.47}$	$\mathbf{3.89}_{\pm 0.07}$
Kitchen-FIST-C	$0.5_{\pm 0.75}$	$2.2_{\pm 1.61}$	$1.9_{\pm 0.96}$	$2.33_{\pm 0.94}$	$\mathbf{3.92}_{\pm 0.06}$
Kitchen-FIST-D	$0.67_{\pm 0.39}$	$1.63_{\pm 1.42}$	$2.17_{\pm 1.67}$	$2.33_{\pm 0.94}$	$\mathbf{3.94}_{\pm 0.07}$
Office	$0.62_{\pm 0.59}$	$0.53_{\pm 0.42}$	$1.83_{\pm 0.49}$	$4.67_{\pm 0.83}$	$\mathbf{6.33}_{\pm 0.30}$

Table 7: Performance of behavior cloning and replaying demonstrations. For convenience, we also list CEIP+TS+EX+forward for reference.

Noise level	CEIP+TS+EX	CEIP+TS+EX+forward	FIST
0.01 (original)	4.17	**6.33**	5.6
0.02	**4.20**	4.17	3.8
0.05	0.57	**0.83**	0.6
0.1	0.05	**0.1**	0
0.2	0.01	**0.02**	0

Table 8: Comparison of the reward for CEIP and FIST when noise increases.

has an Apache-2.0 license. For the office environment, we are using a forked version of the roboverse, which has an MIT license.

Algorithm repositories. We implement PARROT from scratch as PARROT is not open-sourced. For SKiLD and FIST, we use their official github repositories. SKiLD has no license, but we have informed the authors and got their consent for using code academically. FIST has a BSD-3-clause license.