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

Kai Yan **Alexander G. Schwing** **Yu-Xiong Wang**
University of Illinois Urbana-Champaign
{kaiyan3, aschwing, yxw}@illinois.edu
https://github.com/289371298/CEIP

Abstract

Although reinforcement learning has found widespread use in dense reward settings, training autonomous agents with sparse rewards remains challenging. To address this difficulty, prior work has shown promising results when using not only task-specific demonstrations but also task-agnostic albeit somewhat related demonstrations. In most cases, the available demonstrations are distilled into an implicit prior, commonly represented via a single deep net. Explicit priors in the form of a database that can be queried have also been shown to lead to encouraging results. To better benefit from available demonstrations, we develop a method to Combine Explicit and Implicit Priors (CEIP). CEIP exploits multiple implicit priors in the form of normalizing flows in parallel to form a single complex prior. Moreover, CEIP uses an effective explicit retrieval and push-forward mechanism to condition the implicit priors. In three challenging environments, we find the proposed CEIP method to improve upon sophisticated state-of-the-art techniques.

1 Introduction

Reinforcement learning (RL) has found widespread use across domains from robotics [57] and game AI [44] to recommender systems [6]. Despite its success, reinforcement learning is also known to be sample inefficient. For instance, training a robot arm with sparse rewards to sort objects from scratch still requires many training steps if it is at all feasible [46].

To increase the sample efficiency of reinforcement learning, prior work aims to leverage demonstrations [4, 34, 40]. These demonstrations can be *task-specific* [4, 17], i.e., they directly correspond to and address the task of interest. More recently, the use of *task-agnostic* demonstrations has also been studied [14, 16, 34, 46], showing that demonstrations for loosely related tasks can enhance sample efficiency of reinforcement learning agents.

To benefit from either of these two types of demonstrations, most work distills the information within the demonstrations into an *implicit prior*, by encoding available demonstrations in a deep net. For example, SKiLD [34] and FIST [16] use a variational auto-encoder (VAE) to encode the "skills," i.e., action sequences, in a latent space, and train a prior conditioned on states based on demonstrations to use the skills. Differently, PARROT [46] adopts a state-conditional normalizing flow to encode a transformation from a latent space to the actual action space. However, the idea of using the available demonstrations as an *explicit prior* has not received a lot of attention. Explicit priors enable the agent to maintain a database of demonstrations, which can be used to retrieve state-action sequences given an agent's current state. This technique has been utilized in robotics [32, 47] and early attempts of reinforcement learning with demonstrations [4]. It was also implemented as a baseline in [14]. One notable recent exception is FIST [16], which queries a database of demonstrations using the current state to retrieve a likely next state. The use of an explicit prior was shown to greatly enhance the

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

performance. However, FIST uses pure imitation learning without any RL, hence losing the chance for trial and remedy if the imitation is not good enough.

Our key insight is to leverage demonstrations both explicitly *and* implicitly, thus benefiting from both worlds. To achieve this, we develop **CEIP**, a method which **c**ombines **e**xplicit and **i**mplicit **p**riors. **CEIP** leverages implicit demonstrations by learning a transformation from a latent space to the real action space via normalizing flows. More importantly, different from prior work, such as PARROT and FIST which combine all the information within a single deep net, **CEIP** selects the most useful prior by combining multiple flows *in parallel* to form a single large flow. To benefit from demonstrations explicitly, **CEIP** augments the input of the normalizing flow with a likely future state, which is retrieved via a lookup from a database of transitions. For an effective retrieval, we propose a push-forward technique which ensures the database to return future states that have not been referred to yet, encouraging the agent to complete the whole trajectory even if it fails on a single task.

We evaluate the proposed approach on three challenging environments: fetchreach [36], kitchen [11], and office [45]. In each environment, we study the use of both task-specific and task-agnostic demonstrations. We observe that integrating an explicit prior, especially with our proposed push-forward technique, greatly improves results. Notably, the proposed approach works well on sophisticated long-horizon robotics tasks with a few, or sometimes even one task-specific demonstration.

2 Preliminaries

Reinforcement Learning. Reinforcement learning (RL) aims to train an *agent* to make the 'best' decision towards completing a particular task in a given environment. The environment and the task are often described as a Markov Decision Process (MDP), which is defined by a tuple $(\mathcal{S}, \mathcal{A}, T, r, \gamma)$. In timestep t of the Markov process, the agent observes the current *state* $s_t \in \mathcal{S}$, and executes an *action* $a_t \in \mathcal{A}$ following some probability distribution, i.e., *policy* $\pi(a_t|s_t) \in \Delta(\mathcal{A})$, where $\Delta(\mathcal{A})$ denotes the probability simplex over elements in space $\mathcal{A}$. Upon executing action a_t, the state of the agent changes to s_{t+1} following the dynamics of the environment, which are governed by the *transition function* $T(s_t, a_t) : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$. Meanwhile, the agent receives a *reward* $r(s_t, a_t) \in \mathbb{R}$. The agent aims to maximize the cumulative reward $\sum_t \gamma^t r(s_t, a_t)$, where $\gamma \in [0, 1]$ is the discount factor. One complete run in an environment is called an *episode*, and the corresponding state-action pairs $\tau = \{(s_1, a_1), (s_2, a_2), \dots\}$ form a *trajectory* τ.

Normalizing Flows. A normalizing flow [24] is a generative model that transforms elements z_0 drawn from a simple distribution p_z, e.g., a Gaussian, to elements a_0 drawn from a more complex distribution p_a. For this transformation, a bijective function f is used, i.e., $a_0 = f(z_0)$. The use of a bijective function ensures that the log-likelihood of the more complex distribution at any point is tractable and that samples of such a distribution can be easily generated by taking samples from the simple distribution and pushing them through the flow. Formally, the core idea of a normalizing flow can be summarized via $p_a(a_0) = p_z(f^{-1}(a_0)) \left| \frac{\partial f^{-1}(a)}{\partial a}|_{a=a_0} \right|$, where $|\cdot|$ is the determinant (guaranteed positive by flow designs), a is a random variable with the desired more complex distribution, and z is a random variable governed by a simple distribution. To efficiently compute the determinant of the Jacobian matrix of f^{-1}, special constraints are imposed on the form of f. For example, coupling flows like RealNVP [8] and autoregressive flows [31] impose the Jacobian of f^{-1} to be triangular.

3 CEIP: Combining Explicit and Implicit Priors

3.1 Overview

As illustrated in Fig. 1, our goal is to train an autonomous agent to solve challenging tasks despite sparse rewards, such as controlling a robot arm to complete item manipulation tasks (like turning on a switch or opening a cabinet). For this we aim to benefit from available demonstrations. Formally, we consider a task-specific dataset $D_{\text{TS}} = \{\tau_1^{\text{TS}}, \tau_2^{\text{TS}}, \dots, \tau_m^{\text{TS}}\}$, where τ_i^{TS} is the i-th trajectory of the task-specific dataset, and a task-agnostic dataset $D_{\text{TA}} = \{\bigcup D_i | i \in \{1, 2, 3, \dots, n\}\}$, where $D_i = \{\tau_1^i, \tau_2^i, \dots, \tau_{m_i}^i\}$ subsumes the demonstration trajectories for the i-th task in the task-agnostic dataset. Each trajectory $\tau = \{(s_1, a_1), (s_2, a_2), \dots\}$ in the dataset is a state-action pair sequence

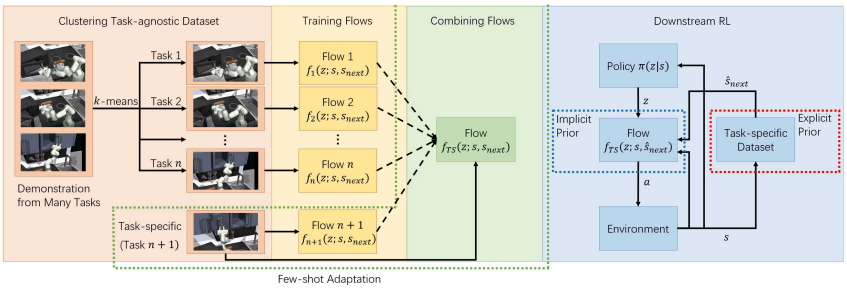


Figure 1: Overview of our proposed approach, CEIP. Our approach can be divided into three steps: a) cluster the task-agnostic dataset into different tasks, and then train one flow on each of the n tasks of the task-agnostic dataset; b) train a flow on the task-specific dataset, and then train the coefficients to combine the $n+1$ flows into one large flow f_{TS}, which is the implicit prior; c) conduct reinforcement learning on the target task; for each timestep, we perform a dataset lookup in the task-specific dataset to find the state most similar to current state s, and return the likely *next state* $\hat{s}_{\text{next}}$ in the trajectory, which is the explicit prior.

of a complete episode, where s is the state, and a is the action. We assume that the number of available task-specific trajectories is very small, i.e., $\sum_{i=1}^{n} m_i \gg m$, which is common in practice. For readability, we will also refer to D_{TS} using D_{n+1}.

Our approach leverages demonstrations implicitly by training a normalizing flow f_{TS}, which transforms the probability distribution represented by a policy $\pi(z|s)$ over a simple latent probability space $\mathcal{Z}$, i.e., $z \in \mathcal{Z}$, into a reasonable expert policy over the space of real-world actions $\mathcal{A}$. As before, s is the current environment state. Thus, the downstream RL agent only needs to learn a policy $\pi(z|s)$ that results in a probability distribution over latent space $\mathcal{Z}$, which is subsequently mapped via the flow f_{TS} to a real-world action $a \in \mathcal{A}$. Intuitively, the MDP in the latent space is governed by a less complex probability distribution, making it easier to train because the flow increases the exposure of more likely actions, while reducing the chance that a less-likely action is chosen. This is because the flow reduces the probability mass for less likely actions given the current state.

Task-agnostic demonstrations contain useful patterns that may be related to the task at hand. However, not all the task-agnostic data are always equally useful, as different task-agnostic data may require to expose different parts of the action space. Therefore, different from prior work where all data are fed into the same deep net model, we first partition the task-agnostic dataset into different groups according to task similarity so as to increase flexibility. For this we use a classical k-means algorithm. We then train different flows f_i on each of the groups, and finally combine the flows via learned coefficients into a single flow f_{TS}. Beneficially, this process permits to expose different parts of the action space as needed and according to perceived task similarity.

Lastly, our approach further leverages demonstrations explicitly, by conditioning the flow not only on the current state but also on a likely next state, to better inform the agent of the state it should try to achieve with its current action. In the following, we first discuss the implicit prior of **CEIP** in Sec. 3.2; afterward we discuss our explicit prior in Sec. 3.3, and the downstream reinforcement learning with both priors in Sec. 3.4.

3.2 Implicit Prior

To better benefit from demonstrations implicitly, we use a 1-layer normalizing flow as the backbone of our implicit prior. It essentially corresponds to a conditioned affine transformation of a Gaussian distribution. We choose a flow-based model instead of a VAE-based one for two reasons: 1) as the dimensionality before and after the transformation via a normalizing flow remains identical and since the flow is invertible, the agent is guaranteed to have control over the whole action space. This ensures that all parts of the action space are accessible, which is not guaranteed by VAE-based methods like SKiLD or FIST; 2) normalizing flows, especially coupling flows such as RealNVP [8], can be easily stacked *horizontally*, so that the combination of parallel flows is also a flow. Among feasible flow models, we found that the simplest 1-layer flow suffices to achieve good results, and is even more robust in training than a more complex RealNVP. Next, in Sec. 3.2.1 we first introduce details regarding the normalizing flow f_i, before we discuss in Sec. 3.2.2 how to combine the flows into one flow f_{TS} applicable to the task for which the task-specific dataset contains demonstrations.

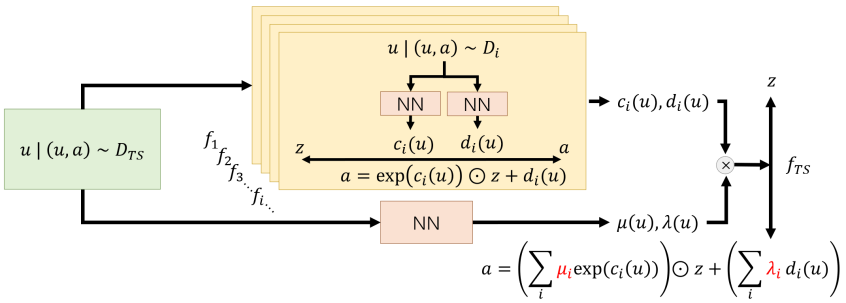


Figure 2: An illustration of how we combine different flows into one large flow for the task-specific dataset. Each red block of "NN" stands for a neural network. Note that $c_i(u)$ and $d_i(u)$ are vectors, while μ_i and λ_i are the i-th dimension of $\mu(u)$ and $\lambda(u)$.

3.2.1 Normalizing Flow Prior. For each task i in the task-agnostic dataset, i.e., for each D_i, we train a conditional 1-layer normalizing flow $f_i(z; u) = a$ which maps a latent space variable $z \in \mathbb{R}^q$ to an action $a \in \mathbb{R}^q$, where q is the number of dimensions of the real-valued action vector. We let u refer to a conditioning variable. In our case u is either the current environment state s (if no explicit prior is used) or a concatenation of the current and a likely next state $[s, s_{\text{next}}]$ (if an explicit prior is used). Concretely, the formulation of our 1-layer flow is

$$f_i(z; u) = a = \exp\{c_i(u)\} \odot z + d_i(u), \tag{1}$$

where $c_i(u) \in \mathbb{R}^q$, $d_i(u) \in \mathbb{R}^q$ are trainable deep nets, and $\odot$ refers to the Hadamard product. The $\exp$ function is applied elementwise. When training the flow, we sample state-action pairs (without explicit prior) or transitions (with explicit prior) (u, a) from the dataset D_i, and maximize the log-likelihood $\mathbb{E}_{(u,a)\sim D_i} \log p(a|u)$; refer to [24] for how to maximize this objective.

In the discussion above, we assume the decomposition of the task-agnostic dataset into tasks to be given. If such a decomposition is not provided (e.g., for the kitchen and office environments in our experiments), we perform a k-means clustering to divide the task-agnostic dataset into different parts. The clustering algorithm operates on the last state of a trajectory, which is used to represent the whole trajectory. The intuition is two-fold. First, for many real-world MDPs, achieving a particular terminal state is more important than the actions taken [12]. For example, when we control a robot to pick and place items, we want all target items to reach the right place eventually; however, we do not care too much about the actions taken to achieve this state. Second, among all the states, the final state is often the most informative about the task that the agent has completed. The number of clusters k in the k-means algorithm is a hyperparameter, which empirically should be larger than the number of dimensions of the action space. Though we assume the task-agnostic dataset is partitioned into labeled clusters, our experiments show that our approach is robust and good results are achieved even without a precise ground-truth decomposition.

In addition to the clusters in the task-agnostic dataset, we train a flow $f_{n+1}(z; u) = a$ on the task-specific dataset $D_{n+1} = D_{\text{TS}}$, using the same maximum log-likelihood loss, which is optional but always available. This is not necessary when the task is relatively simple and the episodes are short (e.g., the fetchreach environment in the experiment section), but becomes particularly helpful in scenarios where some subtasks of a task sequence only appear in the task-specific dataset (e.g., the kitchen environment).

3.2.2 Few-shot Adaptation. The flow models discussed in Sec. 3.2.1 learn which parts of the action space to be more strongly exposed from the latent space. However, not all the flows expose useful parts of the action space for the current state. For example, the target task needs the agent to move its gripper upwards at a particular location, but in the task-agnostic dataset, the robot more often moves the gripper downwards to finish another task. In order to select the most useful prior, we need to tune our set of flows learned on the task-agnostic datasets to the small number of trajectories available in the task-specific dataset. To ensure that this does not lead to overfitting as only a very small number of task-specific trajectories are available, we train a set of coefficients that selects the flow that works the best for the current task. Concretely, given all the trained flows, we train a set of coefficients to combine the flows f_1 to f_n trained on the task-agnostic data, and also the flow f_{n+1} trained on the task-specific data. The coefficients select from the set of available flows the most useful

one. To achieve this, we use the combination flow illustrated in Fig. 2 which is formally specified as follows:

$$f_{\text{TS}}(z;u) = \left(\sum_{i=1}^{n+1} \mu_i(u) \exp\{c_i(u)\}\right) \odot z + \left(\sum_{i=1}^{n+1} \lambda_i(u) d_i(u)\right). \tag{2}$$

Here, $\mu_i(u) \in \mathbb{R}$, $\lambda_i(u) \in \mathbb{R}$ are the i-th entry of the deep nets $\mu(u) \in \mathbb{R}^{n+1}$, $\lambda(u) \in \mathbb{R}^{n+1}$, respectively, which yield the coefficients while the deep nets c_i and d_i are frozen. As before, the $\exp$ function is applied elementwise. We use a softplus activation and an offset at the output of μ to force $\mu_i(u) \geq 10^{-4}$ for any i for numerical stability. Note that the combined flow f_{TS} consisting of multiple 1-layer flows is also a 1-layer normalizing flow. Hence, all the compelling properties over VAE-based architectures described at the beginning of Sec. 3.2 remain valid. To train the combined flow, we use the same log likelihood loss $\mathbb{E}_{(u,a)\sim D_{\text{TS}}} \log p(a|u)$ as that for training single flows. Here, we optimize the deep nets $\mu(u)$ and $\lambda(u)$ which parameterize f_{TS}.

Obviously, the employed combination of flows can be straightforwardly extended to a more complicated flow, e.g., a RealNVP [8] or Glow [22]. However, we found the discussed simple formulation to work remarkably well and to be robust.

3.3 Explicit Prior

Beyond distilling information from demonstrations into deep nets which are then used as implicit priors, we find explicit use of demonstrations to also be remarkably useful. To benefit, we encode future state information into the input of the flow. More specifically, instead of sampling (s, a)-pairs from a dataset D for training the flows, we consider sampling a *transition* (s, a, s_{next}) from D. During training, we concatenate s and s_{next} before feeding it into a flow, i.e., $u = [s, s_{\text{next}}]$ instead of $u = s$.

However, we do not know the future state s_{next} when deploying the policy. To obtain an estimate, we use task-specific demonstrations as explicit priors. More formally, we use the trajectories within the task-specific dataset D_{TS} as a database. This is manageable as we assume the task-specific dataset to be small. For each environment step of reinforcement learning with current state s, we perform a lookup, where s is the query, states s_{key} in the trajectories are the keys, and their corresponding next state s_{next} is the value. Concretely, we assume s_{next} belongs to trajectory τ in the task-specific dataset D_{TS}, and define $\hat{s}_{\text{next}}$ as the result of the database retrieval with respect to the given query s, i.e.,

$$\begin{aligned} \hat{s}_{\text{next}} &= \text{argmin}_{s_{\text{next}}|(s_{\text{key}},a,s_{\text{next}})\in D_{\text{TS}}} [(s_{\text{key}} - s)^2 + C \cdot \delta(s_{\text{next}})], \text{ where} \\ \delta(s_{\text{next}}) &= \begin{cases} 1 \text{ if } \exists s'_{\text{next}} \in \tau, \text{ s.t. } s'_{\text{next}} \text{ is no earlier than } s_{\text{next}} \text{ in } \tau \text{ and has been retrieved,} \\ 0 \text{ otherwise.} \end{cases} \end{aligned} \tag{3}$$

In Eq. (3), C is a constant and δ is the indicator function. We set $u = [s, \hat{s}_{\text{next}}]$ as the condition, feed it into the trained flow f_{TS}, and map the latent space element z obtained from the RL policy to the real-world action a. The penalty term δ is a push-forward technique, which aims to push the agent to move forward instead of staying put, imposing monotonicity on the retrieved $\hat{s}_{\text{next}}$. Consider an agent at a particular state s and a flow f_{TS}, conditioned on $u = [s, \hat{s}_{\text{next}}]$ which maps the chosen action z to a real-world action a that does not modify the environment. Without the penalty term, the agent will remain at the same state, retrieve the same likely next state, which again maps onto the action that does not change the environment. Intuitively, this term discourages 1) retrieving the same state twice, and 2) returning to earlier states in a given trajectory. In our experiments, we set $C = 1$.

3.4 Reinforcement Learning with Priors

Given the implicit and explicit priors, we use RL to train a policy $\pi(z|s)$ to accomplish the target task demonstrated in the task-specific dataset. As shown in Fig. 1, the RL agent receives a state s and provides a latent space element z. The conditioning variable of the flow is retrieved via the dataset lookup described in Sec. 3.3 and the real-world action a is then computed using the flow. Note, our approach is suitable for any RL method, i.e., the policy $\pi(z|s)$ can be trained using any RL algorithm such as proximal policy optimization (PPO) [43] or soft-actor-critic (SAC) [15].

4 Experiments

In this section, we evaluate our CEIP approach on three challenging environments: fetchreach (Sec. 4.1), kitchen (Sec. 4.2), and office (Sec. 4.3), which are all tasks that manipulate a robot arm. In each experiment, we study the following questions: 1) Can the algorithm make good use of the demonstrations compared to baselines? 2) Are our core design decisions (e.g., state augmentation with explicit prior and the push-forward technique) indeed helpful?

Baselines. We compare the proposed method to three baselines: PARROT [46], SKiLD [34], and FIST [16]. In all environments, we use reward as our criteria (higher is better). The results are averaged over 3 runs for SKiLD (much slower to train) and 9 runs for all other methods unless otherwise mentioned. To differentiate variants of the PARROT baseline and our method, we use suffixes. We use "EX" to refer to variants with explicit prior, and "forward" for variants with the push-forward technique. For our method, if we train a task-specific flow on $D_{\text{TS}} = D_{n+1}$, we append the abbreviation "TS." For PARROT, the use of the task-specific data is indicated with "TS" and the use of task-agnostic data is indicated with "TA."[1] See Table 3 and Table 4 for precise correspondence.

4.1 FetchReach Environment

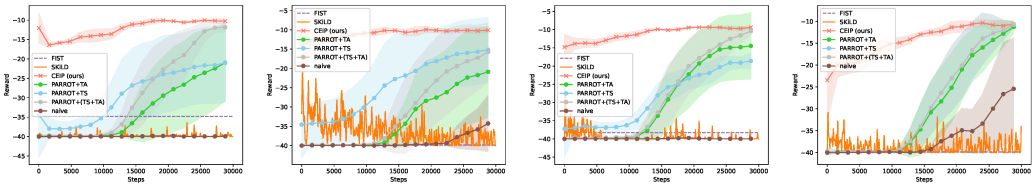
Environment Setup. The agent needs to control a robot arm to move its gripper to a goal location in 3D space, and remain there. During an episode of 40 steps, the agent receives a 10-dimensional state about its location and outputs a 4-dimensional action, which indicates the change of coordinates of the agent and the openness of the gripper. It will receive a reward of 0 if it arrives and stays in the vicinity of its target. Otherwise, it will receive a reward of -1. This environment is a harder version of the FetchReach-v1 robotics environment in gym [36], where we increase the average distance of the starting point to the goal, effectively increasing the training difficulty. Moreover, to test the robustness of the algorithm, we sample a random action from a normal distribution at the beginning of each episode, which the agent executes for x steps before the episode begins. We use $x \sim U[5, 20]$. For simplicity, we denote the goal generated with azimuth $\frac{\pi d}{4}$ as "direction d" (e.g., direction 4.5).

Dataset Setup. We use trajectories from directions $d \in \{0, 1, \ldots, 7\}$ as the task-agnostic data. Each task includes 40 trajectories, and each of the trajectories has 40 steps, i.e., 1600 environment steps in total. The task-specific datasets contain directions $4.5, 5.5, 6.5$, and 7.5. (The robot cannot reach the other four .5 directions due to physical limits.) For each task-specific dataset, we use 4 trajectories, for a total of 160 environment steps.

Experimental Setup. For fetchreach, we use a fully-connected deep net with one hidden layer of width 32 and ReLU [1] activation as a standard "block" of our algorithm (each block corresponds to a red "NN" rectangle in Fig. 2). We have a pair of blocks for $c_i(s)$ and $d_i(s)$ for each flow f_i. For flow training, we train 8 flows for 8 directions in the task-agnostic dataset without the explicit prior. We use a batch size of 40 and train for 1000 epochs for both each flow and the combination of flows, with gradient clipping at norm 10^{-4}, learning rate 0.001, and Adam optimizer [21]. We use the model that has the best performance on the validation dataset at the end of every epoch. For each dataset, we randomly draw 80% state-action pairs (or transitions in ablation) as the training set and 20% state-action pairs as the validation set. The combination of flow is also a block, which outputs both $\mu(s)$ and $\lambda(s)$. See the Appendix for the implementation details of SKiLD, FIST, and PARROT. For each method with RL training, we use a soft-actor-critic (SAC) [15] with 30K environment steps, a batch size of 256, and 1000 steps of initial random exploration. Unless otherwise noted, all other RL hyperparameters in all experiments use the default values of Stable-baselines3 [38].

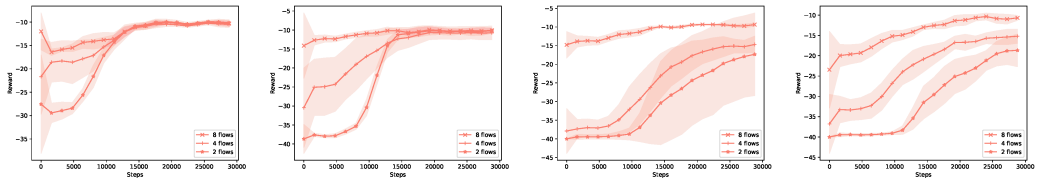
Main Results. Fig. 3 shows the results for different methods without explicit priors or task-specific single flow f_{n+1}. In all four tasks, our method significantly outperforms the other baselines. This indicates that the flow training indeed helps boost the exploration process. Naïve reinforcement learning from scratch fails in most cases, which underscores the necessity of utilizing demonstrations to aid RL exploration. As this is a simple task with only a few wildly varied trajectories, adding a flow for the task-specific dataset does not improve our method. Noteworthy, neither SKiLD nor FIST works on fetchreach. Their VAE-based architecture with each action sequence as the agent's output ("skill") can not be trained with the little amount of wildly varied data with short horizon.

[1] The original PARROT in [46] is essentially PARROT+TA. It is straightforward to use PARROT directly on the task-specific dataset. Hence, we tried PARROT+TS and PARROT+(TS+TA) as well.



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

Figure 3: Main performance results on the fetchreach environment for different directions, where the lines are the mean reward (higher is better) and shades are the standard deviation. FIST is represented by a dashed line as it does not require RL.



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

Figure 4: Ablation on the number of flows used in our method. We observe more flows to lead to better performance, likely because expressivity increases which helps in fitting the expert policy.

Flow-based models like ours and PARROT, which only consider the action of the current step instead of the action sequence, work better.

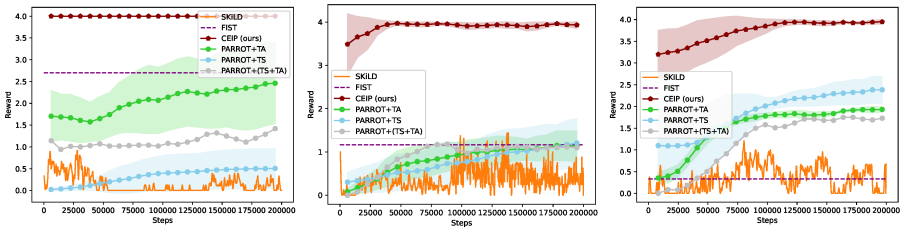
Are more flows helpful for CEIP? Fig. 4 shows the performance of our method using a different number of flows, which are trained on the data of the directions that are the closest to the task-specific direction (e.g., directions 5 and 6 for 2 flows with the target being direction 5.5). The result shows that within a reasonable range, increasing the number of flows improves the expressivity and consequently results of our model. See Appendix D for more ablation studies.

4.2 Kitchen Environment

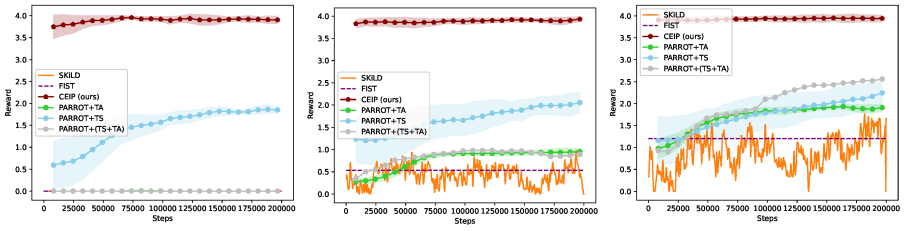
Environment Setup. We use the kitchen environment adopted from D4RL [11], which serves as a testbed for many reinforcement learning and imitation learning approaches, like SPiRL [33], SKiLD [34], relay policy learning [14], and FIST [16]. The agent needs to control a 7-DOF robot arm to complete a sequence of four tasks (e.g., move the kettle, or slide the cabinet) in the correct order. The agent will receive a $+1$ reward only upon finishing a task, and 0 otherwise. The action space is 9-dimensional and the state space is 60-dimensional. This environment is very challenging, as high-precision control of the robot arm is needed and also a long horizon of 280 timesteps is needed. Moreover, there is a small noise applied to each agent action, which requires the agent to be robust.

Dataset Setup. We use two dataset settings, which are adopted from SKiLD and FIST (denoted as *Kitchen-SKiLD* and *Kitchen-FIST* below). In Kitchen-SKiLD, we use 601 teleoperated sequences that perform a variety of task sequences as the task-agnostic dataset, and use *only one* trajectory for the task-specific dataset. In Kitchen-FIST, we use part of the task-agnostic dataset (about $200 - 300$ trajectories) that *does not* contain a particular task in the task-specific dataset, and use *only one* trajectory for the task-specific dataset. There are two different task-specific datasets in Kitchen-SKiLD, and four different task-specific datasets in Kitchen-FIST. The latter is significantly harder, as the agent must learn a new task from very little data. For simplicity, we denote them as "SKiLD-A/B" and "FIST-A/B/C/D" respectively. See Appendix C for details on each task.

Experimental Setup. We use k-means to partition the task-agnostic datasets into 24 different clusters, and train 24 flows accordingly. For each flow, we use a fully-connected network with 2 hidden layers of width 256 with ReLU activation as a "block" for our algorithm. For the combination of flows, we use a fully-connected network with 1 hidden layer of width 64 with ReLU activation. Each layer of the deep net (except the output layer) described above has a 1D batchnorm function. The blocks are used analogously to the fetchreach environment. We use a batch size of 256 for the task-agnostic



(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 5: Comparison on Kitchen-SKiLD and Kitchen-FIST environments.

dataset and a batch size of 128 for the task-specific dataset. Other training hyperparameters are identical to the fetchreach environment. For each RL training, we use proximal policy optimization (PPO) [43] for 200K environment steps, with update interval being 2048 (Kitchen-SKiLD) / 4096 (Kitchen-FIST), 60 epochs per update, and a batch size of 64.

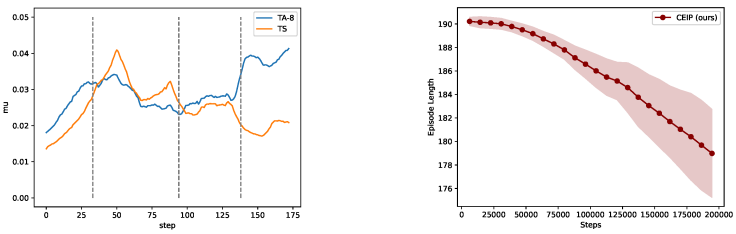
Main Results. Fig. 5 shows the main results on Kitchen-SKiLD and Kitchen-FIST. Our method outperforms all other baselines in all of the 6 settings of the task-agnostic and task-specific datasets. For our method, we use the task-specific single flow f_{n+1}, explicit prior, and the push-forward technique. We compare to the original PARROT formulation. See Appendix D for ablation studies of PARROT with explicit prior and our method without f_{n+1} or explicit prior.

Does CEIP overly rely on the task-specific flow if it is used? One concern for our method could be: does the task-specific single flow dominate the model? Theoretically, when all flows are perfect, a trivial combination of flows that minimizes the training objective is to set $\lambda_{n+1} = 1$, $\mu_{n+1} = 1$ for the task-specific single flow, and $\lambda_i \approx 0, \mu_i \approx 0$ for $i \neq n+1$. To study this concern, we plot the change of the coefficient μ during an episode in Fig. 6. We observe that the single flow on the task-specific dataset is not dominating the combination of the flow, despite being trained on the task-specific dataset. The blue curve with legend 'TA-8' in Fig. 6 shows the coefficient for the 8th flow trained in the task-agnostic dataset. It exhibits an increase of μ at the end of an episode, as the last subtask in the target task is more relevant to the prior encoded in the 8th flow. Intuitively, over-reliance in our design (Fig. 8 in the Appendix) is discouraged, because of the softplus function and the positive offset applied on μ. For over-reliance, all task-agnostic flows f_i with $i \in \{1, 2, \ldots, n\}$ should have a coefficient of $\mu_i = 0$, which is hard to approach due to the offset of μ and softplus. In fact, a degenerated CEIP is essentially PARROT+TS(+EX+forward), which is worse than our method but still a powerful baseline.

Is reinforcement learning useful in cases with a perfect initial reward? Fig. 6 shows the episode length of our method on Kitchen-SKiLD-A. Even if the reward is already perfect, the reinforcement learning process is still able to maximize discounted reward, which optimizes the path.

4.3 Office Environment

Environment and Dataset Setups. We follow SKiLD, where a robot with 8-dimensional action space and 97-dimensional state space needs to put three randomly selected items on a table into three containers in the correct, randomly generated order. The agent will receive a $+1$ reward when it completes a subtask (e.g., picking up an item, or dropping the item at the right place), and 0 otherwise. This environment is even harder than the kitchen environment, as the agent must manipulate freely movable objects and the number of possible subtasks in the task-agnostic dataset is much larger than that in the kitchen environment. We use the same task-agnostic dataset as SKiLD, which contains



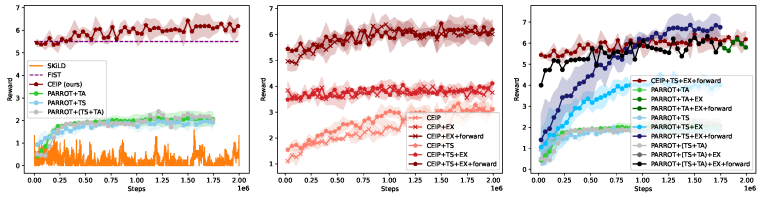
(a) Illustration of coefficient (b) Average episode length

Figure 6: a) Illustration of the coefficient change of a trained CEIP model during an episode of Kitchen-SKiLD-A. This CEIP model is trained with the task-specific single flow and without the explicit prior. 'TA-8' is the 8-th single flow for the task-agnostic dataset, and 'TS' is the single flow for the task-specific dataset. The grey dotted lines are the partition of different subtasks. b) Average episode length of our method on the Kitchen-SKiLD-A task. The episode ends immediately when all the tasks are completed; thus, shortening length means that RL helps to find policy with more efficient completion of tasks.

2400 trajectories with randomized subtasks sampled from a script policy. For the task-specific dataset, we use 5 trajectories for a particular combination of tasks.

Experimental Setup. Similar to the kitchen environment, we use k-means over the last state of each trajectory and partition the task-agnostic dataset into 24 clusters. The architecture and training paradigm of the flow model are identical to those used in the kitchen environment. For RL training, we use PPO for 2M environment steps, with update interval being 4096 environment steps, 60 epochs per update, and a batch size of 64. All other hyperparameters follow the kitchen environment setting. We run each method with 3 different seeds.

Main Results and Ablation. Fig. 7a shows the main result across different methods. Our method with explicit prior, push-forward technique, and task-specific flow outperforms all baselines. FIST works well in this environment, probably because of two reasons: 1) there are a sufficient number of task-specific trajectories for the VAE-architecture, and 2) the office environment is less noisy than the kitchen environment. However, as FIST does not contain a reinforcement learning stage, it has no chance to improve on a decent policy which could have been a good start for an RL agent. Fig. 7b shows the ablation of our method. While the task-specific single flow f_{n+1} does not help in this environment, the explicit prior greatly improves results. Also, as illustrated, the reward curve of the variants with the explicit prior but without the push-forward skill does not grow, which is due to the agent getting stuck as described at the end of Sec. 3.3. Fig. 7c shows the ablation result of PARROT, which also emphasizes that the explicit prior and push-forward skill greatly improve results.



(a) Main result (b) Ours ablation (c) PARROT ablation

Figure 7: Main result and ablation of our method and PARROT on the office environment.

5 Related Work

Reinforcement Learning with Demonstrations. Using demonstrations to improve the sample efficiency of RL is an established direction [13, 26, 41, 55]. Recently, the use of task-agnostic demonstrations has gained popularity, as task-specific data need to be sampled from a particular expert and can be expensive to acquire [33, 34, 46]. To utilize the prior, skill-based methods such as SPiRL [33], SKiLD [34], and SIMPL [30] extract action sequences from the dataset with a VAE-based model, while TRAIL [58] recovers transitions from a task-agnostic dataset with uniformly randomly sampled action. Our method considers the situation where both task-agnostic *and* task-specific data exist and significantly improves results over prior work with similar settings, e.g., SKiLD [34].

Action Priors. An action prior is a common way to utilize demonstrations for reinforcement learning [34] and imitation learning [16]. Most work uses an implicit prior, where a probability distribution of actions conditioned on a state is learned by a deep net and then used to rule out unlikely attempts [3], to form a hierarchical structure [34, 46], or to serve as a regularizer for RL training [34, 40], preventing the agent to stray too far from expert demonstrations. Explicit priors are less explored. They come in the form of nearest neighbors [2] (as in our work) or in the form of locally weighted regression [32]. They are utilized in robotics [2, 32, 42, 47] and early work of RL with demonstrations [4]. Another way to explicitly use demonstrations includes filling the buffer of offline RL algorithms with transitions sampled from an expert dataset to help exploration [17, 29, 54]. Different from all such work, we propose a novel way of using both implicit and explicit priors.

Normalizing Flow. Normalizing flows are a generative model that can be used for variational inference [23, 52] and density estimation [18, 31] and come in different forms: RealNVP [8], Glow [22], or autoregressive flow [31]. Many methods use normalizing flows in reinforcement learning [20, 27, 28, 48, 49, 56] and imitation learning [5]. However, most prior work uses normalizing flows as a strong density estimator to exploit a richer class of policies. Most closely related to our work is PARROT [46], which trains a single normalizing flow as an implicit prior. Different from our work, PARROT does not differentiate tasks among the task-agnostic dataset and does not use an explicit prior. More importantly, different from prior work, we develop a simple yet effective way to combine flows using learned coefficients. While there are some approaches that combine flows via variational mixtures [7, 35], they have not been shown to succeed on challenging RL tasks.

Few-shot Generalization. Few-shot generalization [50] is broadly related, as a model is first trained across different datasets, and then adapted to a new dataset with small sample size. For example, similar to our work, FLUTE [51], SUR [9], and URT [25] use models for multiple datasets, which are then combined via weights for few-shot adaptation. Other methods have shared parameters across different tasks and only used some components within the model for adaptation [10, 37, 39, 51, 59]. While most work focuses on classification tasks, we address more complex RL tasks. Also, different from existing work, we found training of independent 1-layer flows without shared layers to be more flexible, and free from negative transfer as also reported by [19].

6 Discussion and Conclusion

We developed **CEIP**, a method for reinforcement learning which combines explicit and implicit priors obtained from task-agnostic and task-specific demonstrations. For implicit priors we use normalizing flows. For explicit priors we use a database lookup with a push-forward retrieval. In three challenging environments, we show that **CEIP** improves upon baselines.

Limitations. Limitations of CEIP are as follows: 1) *Training time*. The use of demonstrations requires training a decent number of flows which can be time-consuming, albeit mitigated to some extent by parallel training. 2) *Reliance on optimality of expert demonstrations.* Similar to prior work like SKiLD [34] and FIST [16], our method assumes availability of optimal state-action trajectories for the target task. Accuracy of those demonstrations impacts results. Future work will focus on improving robustness and generality. 3) *Balance between the degree of freedom and generalization in fitting the flow mixture.* Fig. 4 reveals that more degrees of freedom in the flow mixture improve results of CEIP. Our current design uses a linear combination which offers $O(n)$ degrees of freedom (μ and λ), where n is the number of flows. In contrast, too many degrees of freedom will result in overfitting. It is interesting future work to study this tradeoff.

Societal impact. Our work helps to train RL agents more efficiently from demonstrations for the same and closely related tasks, particularly when the environment only provides sparse rewards. If successful, this expands the applicability of automation. However, increased automation may also cause job loss which negatively impacts society.

Acknowledgements. This work was supported in part by NSF under Grants 1718221, 2008387, 2045586, 2106825, MRI 1725729, NIFA award 2020-67021-32799, the Jump ARCHES endowment through the Health Care Engineering Systems Center, the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign through the NCSA Fellows program, and the IBM-Illinois Discovery Accelerator Institute. We thank NVIDIA for a GPU.

References

[1] A. F. Agarap. Deep learning using rectified linear units (ReLU). *ArXiv preprint: arXiv1803.08375*, 2018.

[2] S. P. Arunachalam, S. Silwal, B. Evans, and L. Pinto. Dexterous imitation made easy: A learning-based framework for efficient dexterous manipulation. *ArXiv preprint: arXiv2203.13251*, 2022.

[3] O. Biza, D. Wang, R. Platt, J.-W. van de Meent, and L. L. Wong. Action priors for large action spaces in robotics. In *AAMAS*, 2021.

[4] T. Brys, A. Harutyunyan, H. B. Suay, S. Chernova, M. E. Taylor, and A. Nowé. Reinforcement learning from demonstration through shaping. In *IJCAI*, 2015.

[5] W.-D. Chang, J. C. G. Higuera, S. Fujimoto, D. Meger, and G. Dudek. Il-flow: Imitation learning from observation using normalizing flows. In *NeurIPS 2021 Workshop on Robot Learning*, 2021.

[6] X. Chen, S. Li, H. Li, S. Jiang, Y. Qi, and L. Song. Generative adversarial user model for reinforcement learning based recommendation system. In *ICML*, 2019.

[7] S. Ciobanu. Mixtures of normalizing flows. In *CAINE*, 2021.

[8] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real NVP. In *ICLR*, 2017.

[9] N. Dvornik, C. Schmid, and J. Mairal. Selecting relevant features from a multi-domain representation for few-shot classification. In *ECCV*, 2020.

[10] S. Flennerhag, A. A. Rusu, R. Pascanu, H. Yin, and R. Hadsell. Meta-learning with warped gradient descent. In *ICLR*, 2020.

[11] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4RL: Datasets for deep data-driven reinforcement learning. *ArXiv preprint: arXiv2004.07219*, 2020.

[12] S. Ghasemipour, R. Zemel, and S. Gu. A divergence minimization perspective on imitation learning methods. In *CoRL*, 2019.

[13] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, and L. M. Gambardella. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 2016.

[14] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *CoRL*, 2019.

[15] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

[16] K. Hakhamaneshi, R. Zhao, A. Zhan, P. Abbeel, and M. Laskin. Hierarchical few-shot imitation with skill transition models. In *ICLR*, 2022.

[17] T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Piot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, G. Dulac-Arnold, J. Agapiou, J. Z. Leibo, and A. Gruslys. Deep q-learning from demonstrations. In *AAAI*, 2018.

[18] C.-W. Huang, D. Krueger, A. Lacoste, and A. Courville. Neural autoregressive flows. In *ICML*, 2018.

[19] A. Javaloy and I. Valera. Rotograd: Gradient homogenization in multitask learning. In *ICLR*, 2021.

[20] S. A. Khader, H. Yin, P. Falco, and D. Kragic. Learning stable normalizing-flow control for robotic manipulation. In *ICRA*, 2021.

[21] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[22] D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *NeurIPS*, 2018.

[23] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. Improved variational inference with inverse autoregressive flow. In *NIPS*, 2016.

[24] I. Kobyzev, S. J. Prince, and M. A. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[25] L. Liu, W. Hamilton, G. Long, J. Jiang, and H. Larochelle. A universal representation transformer layer for few-shot image classification. In *ICLR*, 2021.

[26] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet. Learning latent plans from play. In *CoRL*, 2019.

[27] X. Ma, J. K. Gupta, and M. J. Kochenderfer. Normalizing flow policies for multi-agent systems. In *GameSec*, 2020.

[28] B. Mazoure, T. Doan, A. Durand, R. D. Hjelm, and J. Pineau. Leveraging exploration in off-policy algorithms via normalizing flows. In *CoRL*, 2019.

[29] A. Nair, A. Gupta, M. Dalal, and S. Levine. AWAC: Accelerating online reinforcement learning with offline datasets. *ArXiv preprint: arXiv2006.09359*, 2020.

[30] T. Nam, S.-H. Sun, K. Pertsch, S. J. Hwang, and J. J. Lim. Skill-based meta-reinforcement learning. In *ICLR*, 2022.

[31] G. Papamakarios, T. Pavlakou, and I. Murray. Masked autoregressive flow for density estimation. In *NIPS*, 2017.

[32] J. Pari, N. M. Shafiullah, S. P. Arunachalam, and L. Pinto. The surprising effectiveness of representation learning for visual imitation. In *RSS*, 2022.

[33] K. Pertsch, Y. Lee, and J. J. Lim. Accelerating reinforcement learning with learned skill priors. In *CoRL*, 2020.

[34] K. Pertsch, Y. Lee, Y. Wu, and J. J. Lim. Demonstration-guided reinforcement learning with learned skills. In *CoRL*, 2021.

[35] G. G. P. F. Pires and M. A. T. Figueiredo. Variational mixture of normalizing flows. In *ESANN*, 2020.

[36] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, and W. Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research. *ArXiv preprint: arXiv1802.09464*, 2018.

[37] J. Puigcerver, C. Riquelme, B. Mustafa, C. Renggli, A. S. Pinto, S. Gelly, D. Keysers, and N. Houlsby. Scalable transfer learning with expert models. In *ICLR*, 2021.

[38] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 2021.

[39] Z. Ren, R. Yeh, and A. G. Schwing. Not all unlabeled data are equal: Learning to weight data in semi-supervised learning. In *NeurIPS*, 2020.

[40] D. Rengarajan, G. Vaidya, A. Sarvesh, D. Kalathil, and S. Shakkottai. Reinforcement learning with sparse rewards using guidance from offline demonstration. In *ICLR*, 2022.

[41] S. Schaal. Learning from demonstration. In *NIPS*, 1996.

[42] S. Schaal and C. Atkeson. Robot juggling: implementation of memory-based learning. *IEEE Control Systems Magazine*, 1994.

[43] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *ArXiv preprint arXiv:1707.06347*, 2017.

[44] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *Science*, 2018.

[45] A. Singh, A. Yu, J. Yang, J. Zhang, A. Kumar, and S. Levine. Cog: Connecting new skills to past experience with offline reinforcement learning. In *CoRL*, 2020.

[46] A. Singh, H. Liu, G. Zhou, A. Yu, N. Rhinehart, and S. Levine. Parrot: Data-driven behavioral priors for reinforcement learning. In *ICLR*, 2021.

[47] D. Singh Chaplot, R. Salakhutdinov, A. Gupta, and S. Gupta. Neural topological slam for visual navigation. In *CVPR*, 2020.

[48] Y. Tang and S. Agrawal. Boosting trust region policy optimization by normalizing flows policy. *ArXiv preprint: arXiv1809.10326*, 2018.

[49] A. Touati, H. Satija, J. Romoff, J. Pineau, and P. Vincent. Randomized value functions via multiplicative normalizing flows. In *UAI*, 2019.

[50] E. Triantafillou, T. Zhu, V. Dumoulin, P. Lamblin, U. Evci, K. Xu, R. Goroshin, C. Gelada, K. Swersky, P. Manzagol, and H. Larochelle. Meta-dataset: A dataset of datasets for learning to learn from few examples. In *ICLR*, 2020.

[51] E. Triantafillou, H. Larochelle, R. Zemel, and V. Dumoulin. Learning a universal template for few-shot dataset generalization. In *ICML*, 2021.

[52] R. van den Berg, L. Hasenclever, J. Tomczak, and M. Welling. Sylvester normalizing flows for variational inference. In *UAI*, 2018.

[53] A. van den Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *ArXiv preprint arXiv:1807.03748*, 2018.

[54] M. Vecerík, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. M. O. Heess, T. Rothörl, T. Lampe, and M. A. Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *ArXiv preprint: arXiv1707.08817*, 2017.

[55] Y. Wang, C. Xu, B. Du, and H. Lee. Learning to weight imperfect demonstrations. In *ICML*, 2021.

[56] P. N. Ward, A. Smofsky, and A. Bose. Improving exploration in soft-actor-critic with normalizing flows policies. In *Workshop on Invertible Neural Networks and Normalizing Flows in ICML*, 2019.

[57] F. Xia, C. Li, R. Martín-Martín, O. Litany, A. Toshev, and S. Savarese. ReLMoGen: Leveraging motion generation in reinforcement learning for mobile manipulation. *ArXiv preprint arXiv:2008.07792*, 2020.

[58] M. Yang, S. Levine, and O. Nachum. TRAIL: Near-optimal imitation learning with suboptimal data. In *NeurIPS*, 2021.

[59] L. M. Zintgraf, K. Shiarlis, V. Kurin, K. Hofmann, and S. Whiteson. Fast context adaptation via meta-learning. In *ICML*, 2019.

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

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 $f_1, \dots, f_{n+1}$, which is to maximize the (empirical) log-likelihood

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

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

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

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 ∈ {1, 2, ..., n+1} do                    // recall that we denote D_TS = D_{n+1}
    for j ∈ {1, 2, ..., M} do                  // for loop over epochs
        foreach (u, a) ~ D_i do                // for each data point
            z_0 ← (a − d_i(u)) / exp{c_i(u)}   // elementwise division
            L = log p_z(z_0) − c_i(u)^T 1      // z ~ N(0, I)
            c_i ← c_i + a × ∂L/∂c_i
            d_i ← d_i + a × ∂L/∂d_i
// Training the combination of flows
for j ∈ {1, 2, ..., M_2} do                    // for loop over epochs
    foreach (u, a) ~ D_TS do                   // for each data point
        μ_0 ← μ(u)
        λ_0 ← λ(u)
        c = Σ_{i=1}^{n+1} μ_{0,i} c_i(u)
        d = Σ_{i=1}^{n+1} λ_{0,i} d_i(u)
        z_0 ← (a − d) / c                      // elementwise division
        L ← log p_z(z_0) − c^T 1               // z ~ N(0, I)
        μ ← μ + a × ∂L/∂μ
        λ ← λ + a × ∂L/∂λ
```

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
      $r(\tau) \leftarrow -1$
1 **foreach** $\tau \in D_{TS}$ **do**
2   **foreach** $(s_{key}, a, s_{next}) \in \tau$ **do**
      // Assume this is the i-th step
3   **if** (s_0, j_0, τ_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
4        $s_0 \leftarrow s_{\text{key}}$
5        $j_0 \leftarrow i$
6        $\tau_0 \leftarrow \tau$
7 $r(\tau_0) \leftarrow j_0$ // update last reference for the chosen trajectory
8 $\mu_0 \leftarrow \mu(u)$
9 $\lambda_0 \leftarrow \lambda(u)$
10 $c \leftarrow \sum_{i=1}^{n+1} \mu_{0,i} c_i(u)$
11 $d \leftarrow \sum_{i=1}^{n+1} \lambda_{0,i} d_i(u)$ // get transformation from latent to action space
12 Sample z_0 from RL policy $\pi(z|s)$
13 $a \leftarrow c \odot z_0 + d$

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."

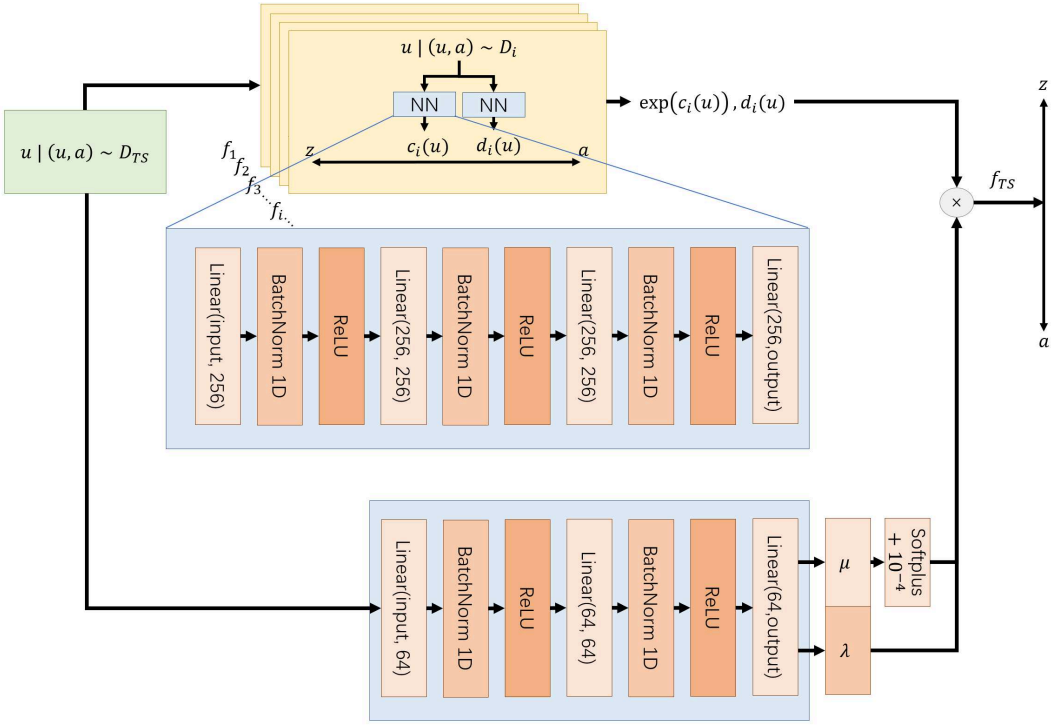


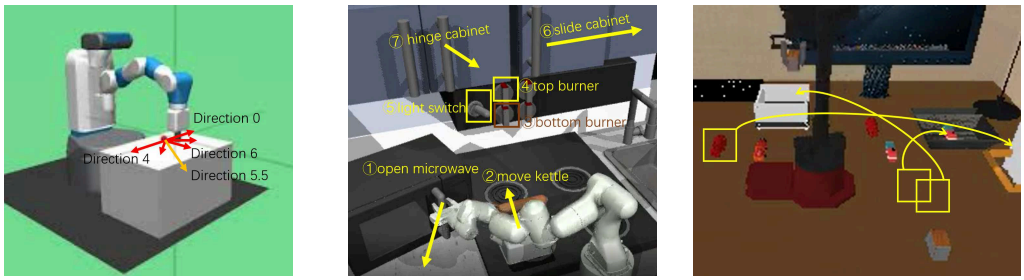
Figure 8: Illustration of our architecture used for kitchen and office environments.

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.

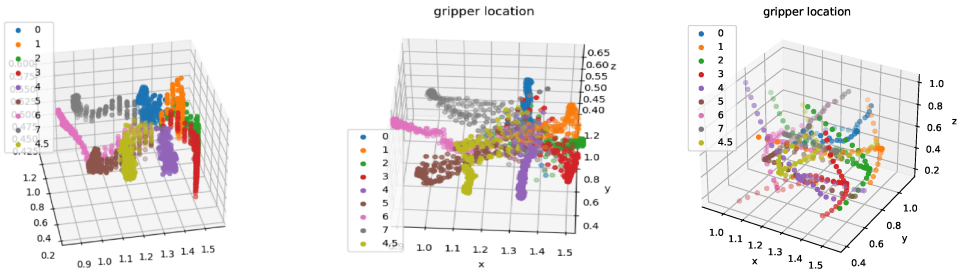
C.1 Fetchreach

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.

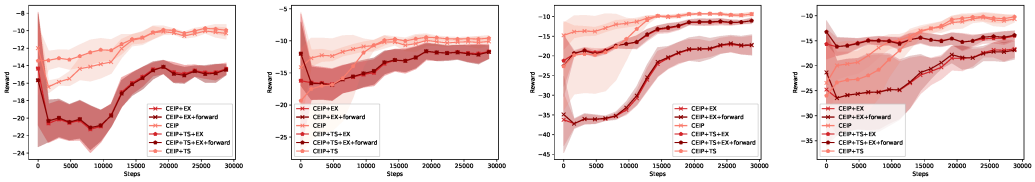
D.1 Abbreviations for Ablation Tests

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

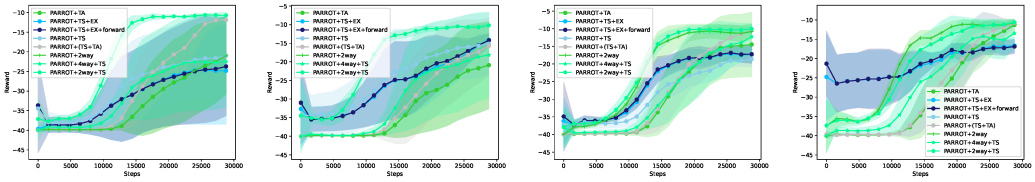
Ablation on components of our method. Fig. 11 shows the ablation study on different components of our method. Results for our method show that using an explicit prior and the push-forward technique slows down the reward growth during RL training, if applied on a relatively easy and short-horizon environment. 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.

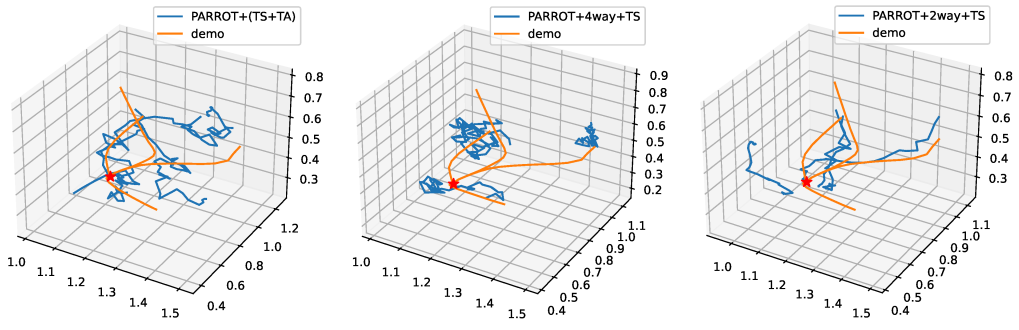
task-agnostic dataset with different levels of relevance to the downstream task affect results. We also test the effect of the explicit prior and the push-forward technique using task-specific data only. The results can be summarized as follows: 1) using the explicit prior and the push-forward technique 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.

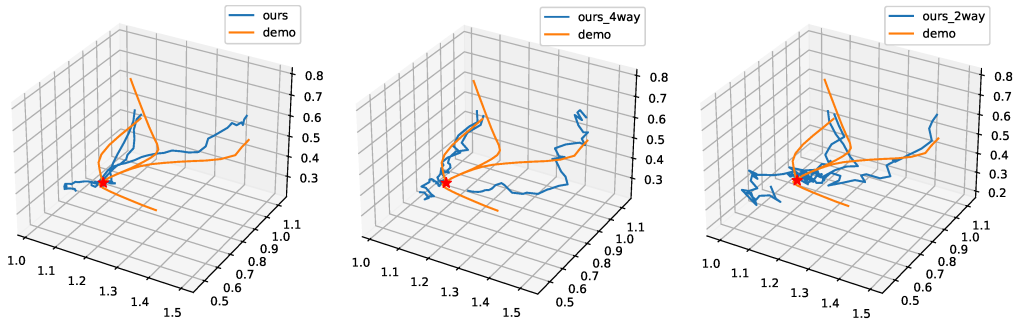
Fig. 14 illustrates the trajectories generated by each method after RL training. As shown in the figure, our method exhibits smoother trajectories after RL training, enabling the agent to reach its goal faster. FIST, SKiLD, and naïve RL fail to generate trajectories that steadily converge to the goal. 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.

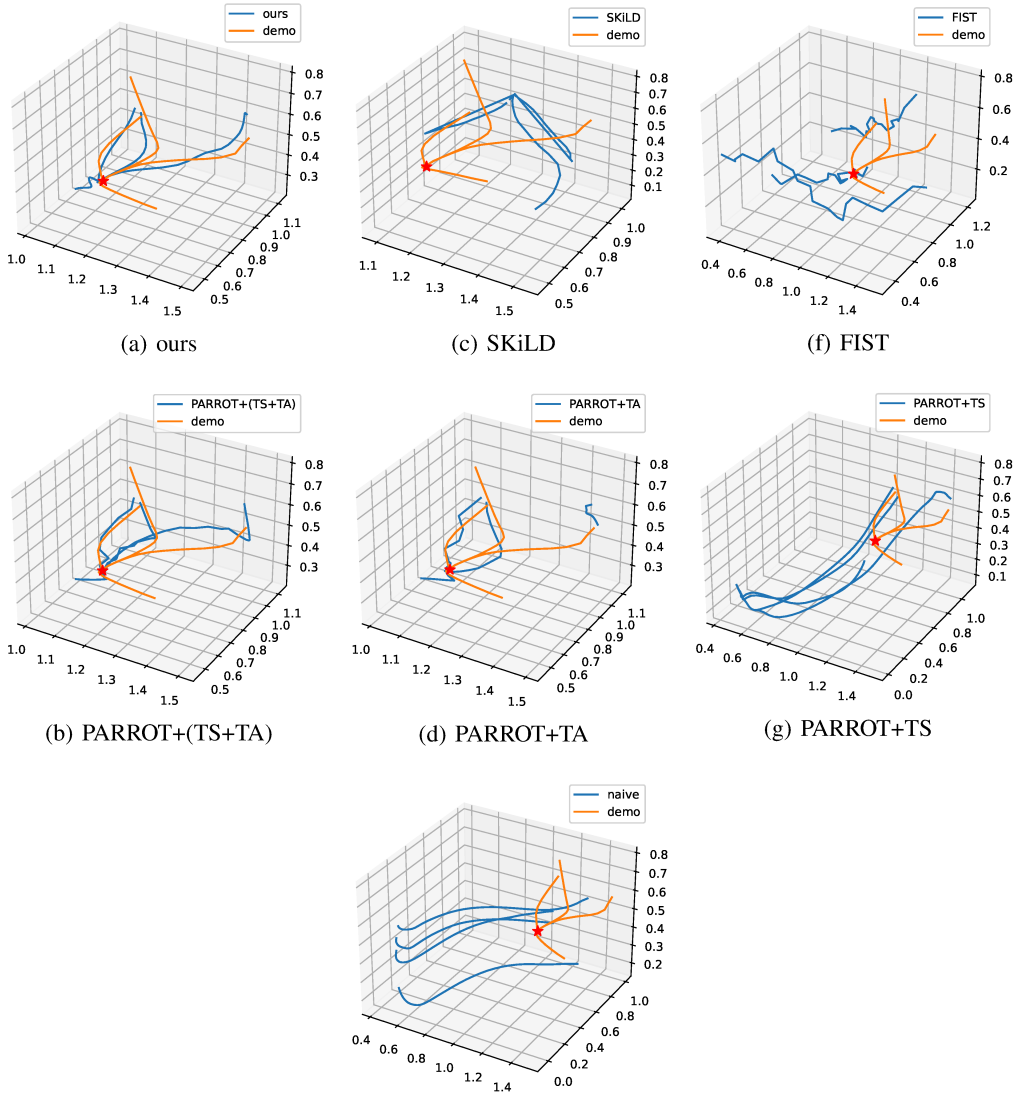
Ablation on components in PARROT. Fig. 16 shows the difference when different architectures are used. As one target sub-task is completely missing from the task-agnostic data, PARROT+TA fails as expected. Also note that the explicit prior boosts the results of PARROT, making it comparable to our method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

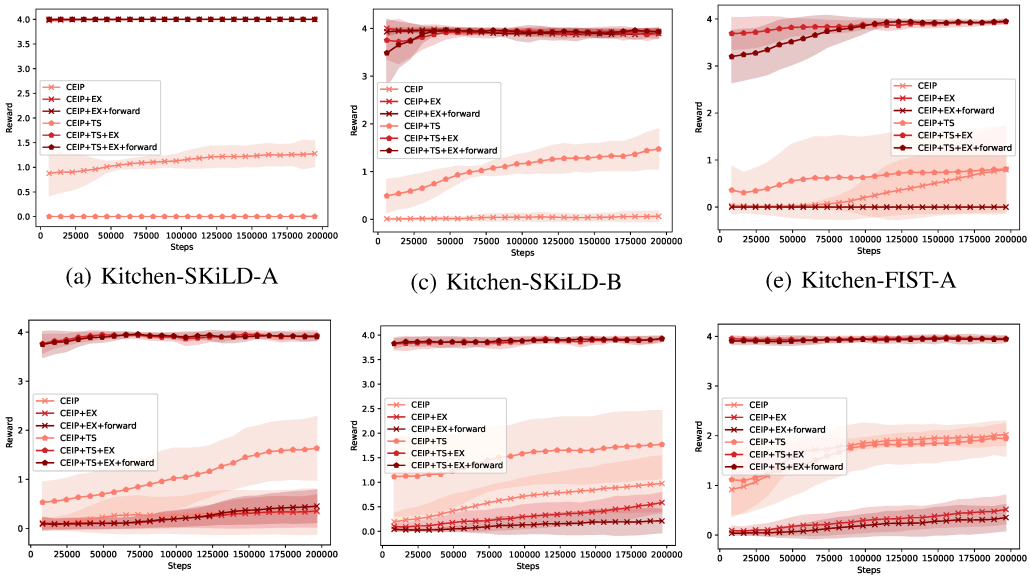
(b) PARROT+(TS+TA) (d) PARROT+TA (g) PARROT+TS

(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.

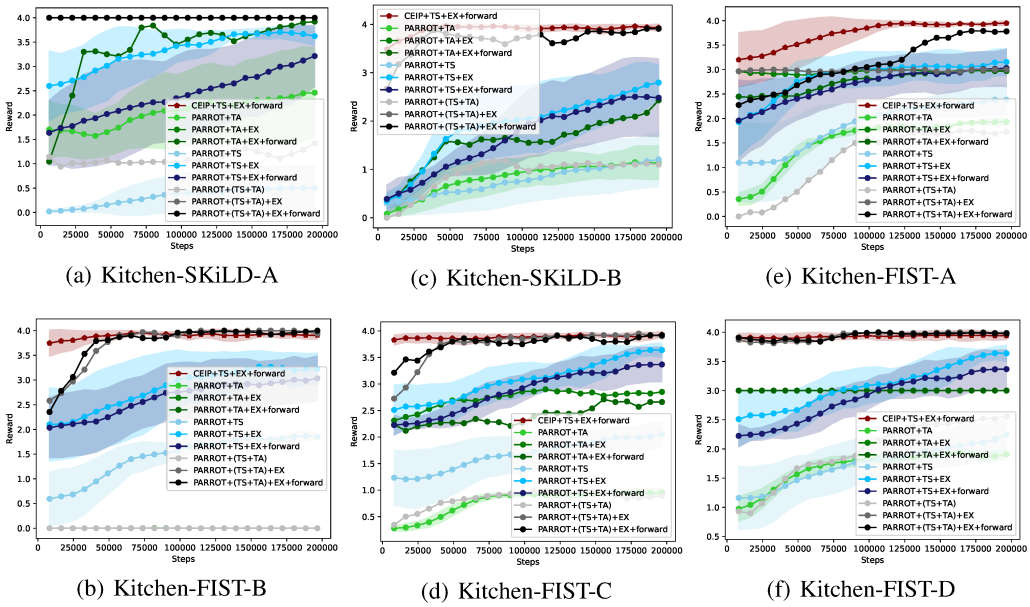
Label	Method	Kitchen-SKiLD-A	Kitchen-SKiLD-B	Kitchen-FIST-A	Kitchen-FIST-B	Kitchen-FIST-C	Kitchen-FIST-D
GT	CEIP+TS+EX	**4**	3.96	3.59	**4**	3.94	3.6
GT	CEIP+TS+EX+forward	**4**	3.95	3	**4**	3.81	3.4
GT24	CEIP+TS+EX	**4**	**4**	3.68	3.76	3.8	3.96
GT24	CEIP+TS+EX+forward	**4**	**4**	3.24	3.75	3.85	3.9
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±0.04	1.52±1.15	2.2±0.62	1.0±0.82	**4.0**±0.00
Kitchen-SKiLD-B	0.03±0.08	1.03±0.90	0.8±0.75	0.67±0.94	**3.93**±0.08
Kitchen-FIST-A	0.67±0.76	2.17±0.06	3.03±0.15	2.33±0.47	**3.95**±0.05
Kitchen-FIST-B	0.4±0.59	2.13±0.47	1.87±0.29	0.67±0.47	**3.89**±0.07
Kitchen-FIST-C	0.5±0.75	2.2±1.61	1.9±0.96	2.33±0.94	**3.92**±0.06
Kitchen-FIST-D	0.67±0.39	1.63±1.42	2.17±1.67	2.33±0.94	**3.94**±0.07
Office	0.62±0.59	0.53±0.42	1.83±0.49	4.67±0.83	**6.33**±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.