Algorithm 1 SkiLD (**Ski**ll-based **L**earning with **D**emonstrations)

```
Inputs: $H$-step reward function $\tilde{r}(s_t, z_t)$, reward weight $\gamma$, discount $\eta$, target divergences $\delta, \delta_q$,
    learning rates $\lambda_\pi, \lambda_Q, \lambda_\alpha$, target update rate $\tau$.
Initialize replay buffer $\mathcal{D}$, high-level policy $\pi_\theta(z_t|s_t)$, critic $Q_\phi(s_t, z_t)$, target network $Q_{\bar{\phi}}(s_t, z_t)$
for each iteration do
    for every $H$ environment steps do
        $z_t \sim \pi(z_t|s_t)$    ▷ sample skill from policy
        $s_{t'} \sim p(s_{t+H}|s_t, z_t)$    ▷ execute skill in environment
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{s_t, z_t, \tilde{r}(s_t, z_t), s_{t'}\}$    ▷ store transition in replay buffer
    for each gradient step do
        $r_\Sigma = (1-\gamma) \cdot \tilde{r}(s_t, z_t) + \gamma \cdot \big[\log D(s_t) - \log\big(1 - D(s_t)\big)\big]$    ▷ compute combined reward
        $\bar{Q} = r_\Sigma + \eta\big[Q_{\bar{\phi}}(s_{t'}, \pi_\theta(z_{t'}|s_{t'})) - \big[\alpha_q D_{\text{KL}}\big(\pi_\theta(z_{t'}|s_{t'}), q_\zeta(z_{t'}|s_{t'})\big) \cdot D(s_{t'})$
            $+ \alpha D_{\text{KL}}\big(\pi_\theta(z_{t'}|s_{t'}), p(z_{t'}|s_{t'})\big) \cdot \big(1 - D(s_{t'})\big)\big]$    ▷ compute Q-target
        $\theta \leftarrow \theta - \lambda_\pi \nabla_\theta \big[Q_\phi(s_t, \pi_\theta(z_t|s_t)) - \big[\alpha_q D_{\text{KL}}\big(\pi_\theta(z_t|s_t), q_\zeta(z_t|s_t)\big) \cdot D(s_t)$
            $+ \alpha D_{\text{KL}}\big(\pi_\theta(z_t|s_t), p(z_t|s_t)\big) \cdot \big(1 - D(s_t)\big)\big]$    ▷ update policy weights
        $\phi \leftarrow \phi - \lambda_Q \nabla_\phi \big[\frac{1}{2}\big(Q_\phi(s_t, z_t) - \bar{Q}\big)^2\big]$    ▷ update critic weights
        $\alpha \leftarrow \alpha - \lambda_\alpha \nabla_\alpha \big[\alpha \cdot \big(D_{\text{KL}}(\pi_\theta(z_t|s_t), p(z_t|s_t)) - \delta\big)\big]$    ▷ update alpha
        $\alpha_q \leftarrow \alpha_q - \lambda_\alpha \nabla_{\alpha_q} \big[\alpha_q \cdot \big(D_{\text{KL}}(\pi_\theta(z_t|s_t), q_\zeta(z_t|s_t)) - \delta_q\big)\big]$    ▷ update alpha-q
        $\bar{\phi} \leftarrow \tau\phi + (1-\tau)\bar{\phi}$    ▷ update target network weights
return trained policy $\pi_\theta(z_t|s_t)$
```

A Full Algorithm

We detail our full SkiLD algorithm for demonstration-guided RL with learned skills in Algorithm 1. It is based on the SPiRL algorithm for RL with learned skills [16] which in turn builds on Soft-Actor Critic [39], an off-policy model-free RL algorithm. We mark changes of our algorithm with respect to SPiRL and SAC in red in Algorithm 1.

The hyperparameters α and α_q can either be constant, or they can be automatically tuned using dual gradient descent [35, 16]. In the latter case, we need to define a set of *target divergences* δ, δ_q. The parameters α and α_q are then optimized to ensure that the expected divergence between policy and skill prior and posterior distributions is equal to the chosen target divergence (see Algorithm 1).

B Implementation and Experimental Details

B.1 Implementation Details: Pre-Training

We introduce our objective for learning the skill inference network $q_\omega(z|s, a)$ and low-level skill policy $\pi_\phi(a_t|s_t, z)$ in Section 3.2. In practice, we instantiate all model components with deep neural networks Q_ω, Π_ϕ respectively, and optimize the full model using back-propagation. We also jointly train our skill prior network P. We follow the common assumption of Gaussian, unit-variance output distributions for low-level policy actions, leading to the following network loss:

$$\mathcal{L} = \underbrace{\prod_{t=0}^{H-2} \big\|a_t - \Pi_\phi(s_t, z)\big\|^2 + \beta D_{\text{KL}}\big(Q_\omega(s_{0:H-1}, a_{0:H-2}) \,||\, \mathcal{N}(0, I)\big)}_{\text{skill representation training}}$$
$$+ \underbrace{D_{\text{KL}}\big(\lfloor Q_\omega(s_{0:H-1}, a_{0:H-2})\rfloor \,||\, P(s_0)\big)}_{\text{skill prior training}}.$$

Here $\lfloor\cdot\rfloor$ indicates that we stop gradient flow from the prior training objective into the skill inference network for improved training stability. After training the skill inference network with above objective, we train the skill posterior network Q_ζ by minimizing KL divergence to the skill inference network's output on trajectories sampled from the demonstration data. We minimize the following objective:

$$\mathcal{L}_{\text{post}} = D_{\text{KL}}\big(\lfloor Q_\omega(s_{0:H-1}, a_{0:H-2})\rfloor \,||\, Q_\zeta(s_0)\big)$$

We use a 1-layer LSTM with 128 hidden units for the inference network and 3-layer MLPs with 128 hidden units in each layer for the low-level policy. We encode skills of horizon 10 into 10-dimensional skill representations z. Skill prior and posterior networks are implemented as 5-layer MLPs with 128 hidden units per layer. They both parametrize mean and standard deviation of Gaussian output distributions. All networks use batch normalization after every layer and leaky ReLU activation functions. We tune the regularization weight β to be $1\mathrm{e}{-3}$ for the maze and $5\mathrm{e}{-4}$ for kitchen and office environment.

For the demonstration discriminator $D(s)$ we use a 3-layer MLP with only 32 hidden units per layer to avoid overfitting. It uses a sigmoid activation function on the final layer and leaky ReLU activations otherwise. We train the discriminator with binary cross-entropy loss on samples from task-agnostic and demonstration datasets:

$$\mathcal{L}_{\mathrm{D}} = -\frac{1}{N} \cdot \Big[\underbrace{\sum_{i=1}^{N/2} \log D(s_i^d)}_{\text{demonstrations}} + \underbrace{\sum_{j=1}^{N/2} \log \big(1 - D(s_j)\big)}_{\text{task-agnostic data}} \Big]$$

We optimize all networks using the RAdam optimizer [41] with parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$, batch size 128 and learning rate $1\mathrm{e}{-3}$. On a single NVIDIA Titan X GPU we can train the skill representation and skill prior in approximately 5 hours, the skill posterior in approximately 3 hours and the discriminator in approximately 3 hours.

B.2 Implementation Details: Downstream RL

The architecture of the policy mirrors the one of the skill prior and posterior networks. The critic is a simple 2-layer MLP with 256 hidden units per layer. The policy outputs the parameters of a Gaussian action distribution while the critic outputs a single Q-value estimate. We initialize the policy with the weights of the skill posterior network.

We use the hyperparameters of the standard SAC implementation [39] with batch size 256, replay buffer capacity of $1\mathrm{e}6$ and discount factor $\gamma = 0.99$. We collect 5000 warmup rollout steps to initialize the replay buffer before training. We use the Adam optimizer [42] with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and learning rate $3\mathrm{e}{-4}$ for updating policy, critic and temperatures α and α_q. Analogous to SAC, we train two separate critic networks and compute the Q-value as the minimum over both estimates to stabilize training. The corresponding target networks get updated at a rate of $\tau = 5\mathrm{e}{-3}$. The policy's actions are limited in the range $[-2, 2]$ by a $\tanh$ "squashing function" (see Haarnoja et al. [39], appendix C).

We use automatic tuning of α and α_q in the maze navigation task and set the target divergences to 1 and 10 respectively. In the kitchen and office environments we obtained best results by using constant values of $\alpha = \alpha_q = 1\mathrm{e}{-1}$. In all experiments we set $\kappa = 0.9$.

For all RL results we average the results of three independently seeded runs and display mean and standard deviation across seeds.

B.3 Implementation Details: Comparisons

BC+RL. This comparison is representative of demonstration-guided RL approaches that use BC objectives to initialize and regularize the policy during RL [6, 7]. We pre-train a BC policy on the demonstration dataset and use it to initialize the RL policy. We use SAC to train the policy on the target task. Similar to Nair et al. [7] we augment the policy update with a regularization term that minimizes the L2 loss between the predicted mean of the policy's output distribution and the output of the BC pre-trained policy[8].

Demo Replay. This comparison is representative of approaches that initialize the replay buffer of an off-policy RL agent with demonstration transitions [4, 5]. In practice we use SAC and initialize a second replay buffer with the demonstration transitions. Since the demonstrations do not come with

---

[8] We also tried sampling action targets directly from the demonstration replay buffer, but found using a BC policy as target more effective on the tested tasks.

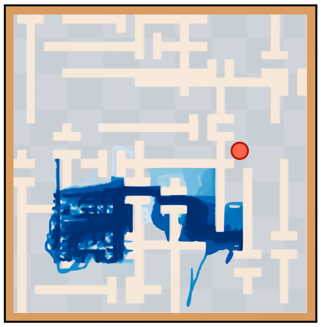

Figure 8: Qualitative results for GAIL+RL on maze navigation. Even though it makes progress towards the goal (**red**), it fails to ever obtain the sparse goal reaching reward.

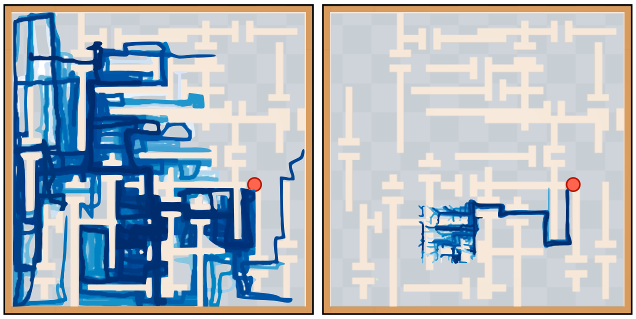

Figure 9: We compare the exploration behavior in the maze. We roll out skills sampled from SPiRL's task-agnostic skill prior (**left**) and our task-specific skill posterior (**right**) and find that the latter leads to more targeted exploration towards the goal (red).

reward, we heuristically set the reward of each demonstration trajectory to be a high value (100 for the maze, 4 for the robotic environments) on the final transition and zero everywhere else. During each SAC update, we sample half of the training mini-batch from the normal SAC replay buffer and half from the demonstration replay buffer. All other aspects of SAC remain unchanged.

B.4 Environment Details

Maze Navigation. We adapt the maze navigation task from Pertsch et al. [16] which extends the maze navigation tasks from the D4RL benchmark [43]. The starting position is sampled uniformly from a start region and the agent receives a one-time sparse reward of 100 when reaching the fixed goal position, which also ends the episode. The 4D observation space contains 2D position and velocity of the agent. The agent is controlled via 2D velocity commands.

Robot Kitchen Environment. We use the kitchen environment from Gupta et al. [24]. For solving the target task, the agent needs to execute a fixed sequence of four subtasks by controlling an Emika Franka Panda 7-DOF robot via joint velocity and continuous gripper actuation commands. The 30-dimensional state space contains the robot's joint angles as well as object-specific features that characterize the position of each of the manipulatable objects. We use 20 state-action sequences from the dataset of Gupta et al. [24] as demonstrations. Since the dataset does not have large variation *within* the demonstrations for one task, the support of those demonstration is very narrow. We collect a demonstration dataset with widened support by initializing the environment at states along the demonstrations and rolling out a random policy for 10 steps.

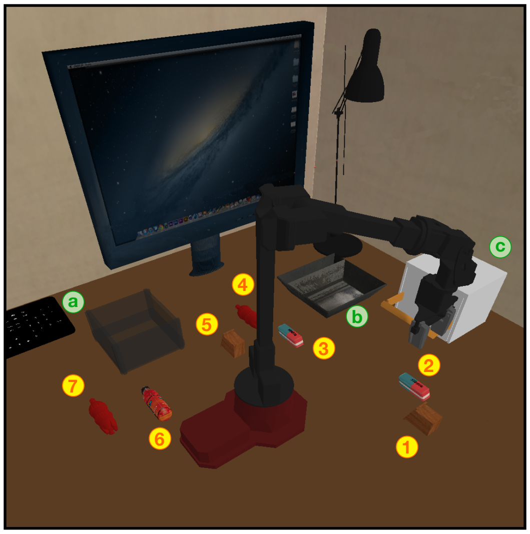

Figure 10: Office cleanup task. The robot agent needs to place three randomly sampled objects **(1-7)** inside randomly sampled containers **(a-c)**. During task-agnostic data collection we apply random noise to the initial position of the objects.

Robot Office Environment. We create a novel office cleanup task in which a 5-DOF WidowX robot needs to place a number of objects into designated containers, requiring the execution of a sequence of pick, place and drawer open and close subtasks (see Figure 10). The agent controls position and orientation of the end-effector and a continuous gripper actuation, resulting in a 7-dimensional action space. For simulating the environment we build on the Roboverse framework [21]. During collection of the task-agnostic data we randomly sample a subset of three of the seven objects as well as a random order of target containers and use scripted policies to execute the task. We only save successful executions. For the target task we fix object positions and require the agent to place three objects in fixed target containers. The 97-dimensional state space contains the agent's end-effector position and orientation as well as position and orientation of all objects and containers.

Differences to Pertsch et al. [16]. While both maze navigation and kitchen environment are based on the tasks in Pertsch et al. [16], we made multiple changes to increase task complexity, resulting in the lower absolute performance of the SPiRL baseline in Figure 4. For the maze navigation task we added randomness to the starting position and terminate the episode upon reaching the goal position, reducing the max. reward obtainable for successfully solving the task. We also switched to a low-dimensional state representation for simplicity. For the kitchen environment, the task originally used in Gupta et al. [24] as well as Pertsch et al. [16] was well aligned with the training data distribution and there were no demonstrations available for this task. In our evaluation we use a different downstream task (see section F) which is less well-aligned with the training data and therefore harder to learn. This also allows us to use sequences from the dataset of Gupta et al. [24] as demonstrations for this task.

C Skill Representation Comparison

In Section 3.2 we described our skill representation based on a closed-loop low-level policy as a more powerful alternative to the open-loop action decoder-based representation of Pertsch et al. [16]. To compare the performance of the two representations we perform rollouts with the learned skill prior: we sample a skill from the prior and rollout the low-level policy for H steps. We repeat this until the episode terminates and visualize the results for multiple episodes in maze and kitchen environment in Figure 11.

In Figure 11 (top) we see that both representations lead to effective exploration in the maze environment. Since the 2D maze navigation task does not require control in high-dimensional action spaces, both skill representations are sufficient to accurately reproduce behaviors observed in the task-agnostic training data.

In contrast, the results on the kitchen environment (Figure 11, bottom) show that the closed-loop skill representation is able to more accurately control the high-DOF robotic manipulator and reliably solve multiple subtasks per rollout episode.[9] We hypothesize that the closed-loop skill policy is able to learn more robust skills from the task-agnostic training data, particularly in high-dimensional control problems.

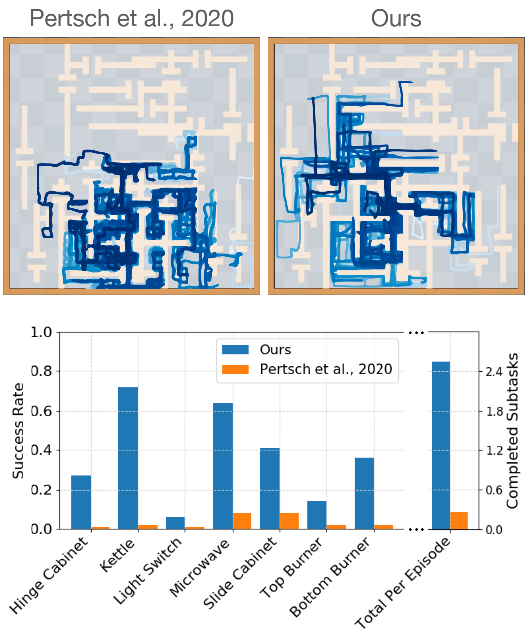

Figure 11: Comparison of our closed-loop skill representation with the open-loop representation of Pertsch et al. [16]. **Top**: Skill prior rollouts for 100 k steps in the maze environment. **Bottom**: Subtask success rates for prior rollouts in the kitchen environment.

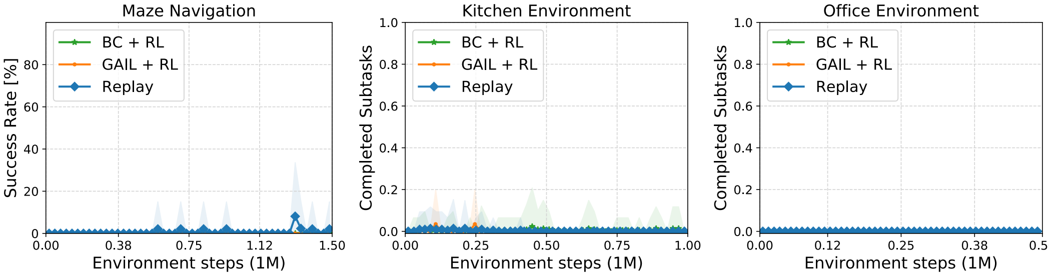

Figure 12: Downstream task performance for prior demonstration-guided RL approaches with combined task-agnostic and task-specific data. All prior approaches are unable to leverage the task-agnostic data, showing a performance decrease when attempting to use it.

D Demonstration-Guided RL Comparisons with Task-Agnostic Experience

In Section 4.2 we compared our approach to prior demonstration-guided RL approaches which are not designed to leverage task-agnostic datasets. We applied these prior works in the setting they were designed for: using only task-specific demonstrations of the target task. Here, we conduct experiments in which we run these prior works using the *combined* task-agnostic and task-specific datasets to give them access to the same data that our approach used.

From the results in Figure 12 we can see that none of the prior works is able to effectively leverage the additional task-agnostic data. In many cases the performance of the approaches is worse than when only using task-specific data (see Figure 4). Since prior approaches are not designed to leverage task-agnostic data, applying them in the combined-data setting can hurt learning on the target task. In contrast, our approach can effectively leverage the task-agnostic data for accelerating demonstration-guided RL.

E Skill-Based Imitation Learning

We ablate the influence of the environment reward feedback on the performance of our approach by setting the reward weight $\kappa = 1.0$, thus relying solely on the learned discriminator reward. Our goal is to test whether our approach SkiLD is able to leverage task-agnostic experience to improve the performance of pure *imitation learning*, i.e., learning to follow demonstrations without environment reward feedback.

We compare SkiLD to common approaches for imitation learning: behavioral cloning (BC, Pomerleau [11]) and generative adversarial imitation learning (GAIL, Ho and Ermon [13]). We also experiment with a version of our skill-based imitation learning approach that performs online finetuning of the pre-trained discriminator $D(s)$ using data collected during training of the imitation policy.

We summarize the results of the imitation learning experiments in Figure 13. Learning purely by imitating the demonstrations, without additional reward feedback, is generally slower than demonstration-guided RL on tasks that require more challenging control, like in the kitchen environment, where the pre-trained discriminator does not capture the desired trajectory distribution accurately. Yet, we find that our approach is able to leverage task-agnostic data to effectively imitate complex, long-horizon behaviors while prior imitation learning approaches struggle. Further, online finetuning of the learned discriminator improves imitation learning performance when the pre-trained discriminator is not accurate enough.

In the maze navigation task the pre-trained discriminator represents the distribution of solution trajectories well, so pure imitation performance is comparable to demonstration-guided RL. We find that finetuning the discriminator on the maze "sharpens" the decision boundary of the discriminator,

---

[9] See `https://sites.google.com/view/skill-demo-rl` for skill prior rollout videos with both skill representations in the kitchen environment.

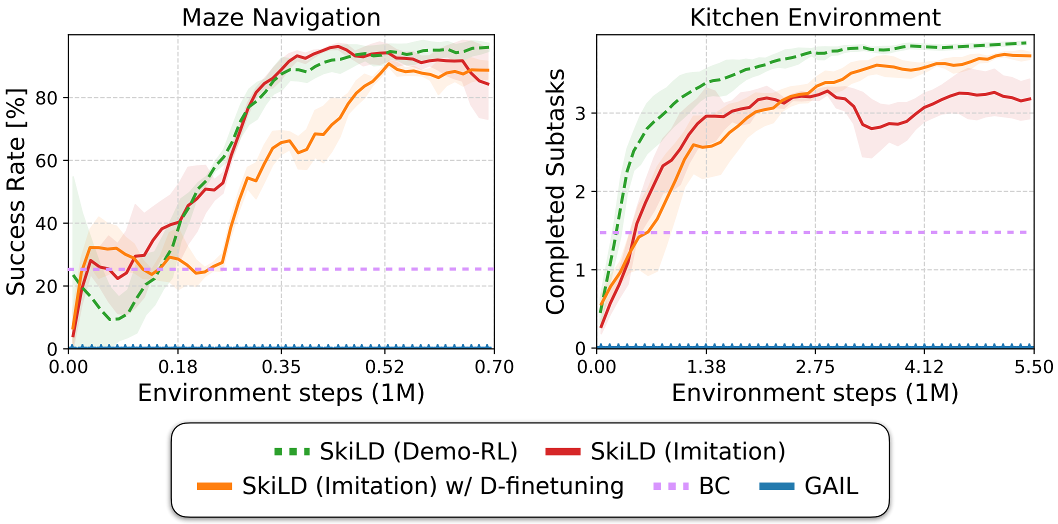

Figure 13: Imitation learning performance on maze navigation and kitchen tasks. Compared to prior imitation learning methods, SkiLD can leverage prior experience to enable the imitation of complex, long-horizon behaviors. Finetuning the pre-trained discriminator $D(s)$ further improves performance on more challenging control tasks like in the kitchen environment.

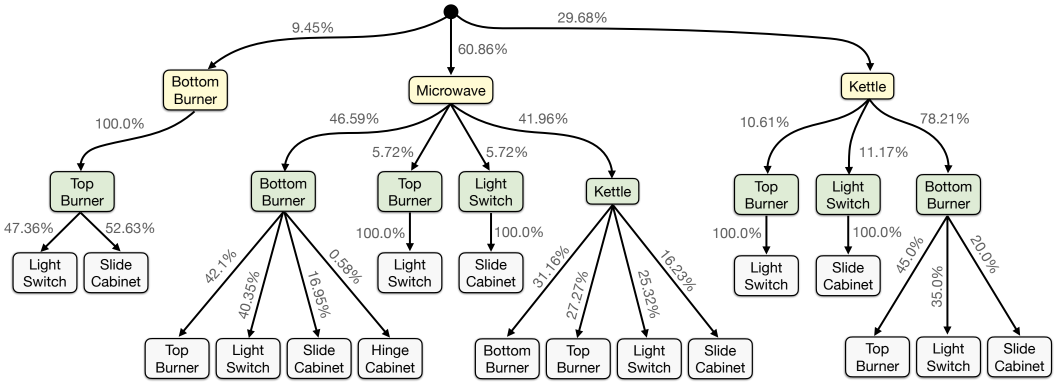

Figure 14: Subtask transition probabilities in the kitchen environment's task-agnostic training dataset from Gupta et al. [24]. Each dataset trajectory consists of four consecutive subtasks, of which we display three (**yellow**: first, **green**: second, **grey**: third subtask). The transition probability to the fourth subtask is always near $100\,\%$. In Section 4.5 we test our approach on a target task with good alignment to the task-agnostic data (*Microwave - Kettle - Light Switch - Hinge Cabinet*) and a target task which is mis-aligned to the data (*Microwave - Light Switch - Slide Cabinet - Hinge Cabinet*).

i.e., increases its confidence in correctly estimating the demonstration support. Yet, this does not lead to faster overall convergence since the pre-trained discriminator is already sufficiently accurate.

F Kitchen Data Analysis

For the kitchen manipulation experiments we use the dataset provided by Gupta et al. [24] as task-agnostic pre-training data. It consists of 603 teleoperated sequences, each of which shows the completion of four consecutive subtasks. In total there are seven possible subtasks: opening the microwave, moving the kettle, turning on top and bottom burner, flipping the light switch and opening a slide and a hinge cabinet.

In Figure 14 we analyze the transition probabilities between subtasks in the task-agnostic dataset. We can see that these transition probabilities are not uniformly distributed, but instead certain transitions are more likely than others, e.g., it is much more likely to sample a training trajectory in which the agent first opens the microwave than one in which it starts by turning on the bottom burner.

In Section 4.5 we test the effect this bias in transition probabilities has on the learning of target tasks. Concretely, we investigate two cases: good alignment between task-agnostic data and target task and mis-alignment between the two. In the former case we choose the target task *Kettle - Bottom Burner - Top Burner - Slide Cabinet*, since the required subtask transitions are likely under the training data distribution. For the mis-aligned case we choose *Microwave - Light Switch - Slide Cabinet - Hinge Cabinet* as target task, since particularly the transition from opening the microwave to flipping the light switch is very unlikely to be observed in the training data.