Algorithm 2 STAR (Semantic Transfer Accelerated RL) – High-level Policy Optimization

Inputs: H-step reward function $\tilde{r}(s_t, k_t, z_t)$, reward weight κ, discount η, target divergences $\delta_q, \delta_p, \delta_l$, learning rates $\lambda_\pi, \lambda_Q, \lambda_\alpha$, target update rate τ.
Load pre-trained & freeze: low-level policy $\pi^l(a_t|s_t, k_t, z_t)$, skill priors $p^{\text{demo}}(k_t|s_t), p^{\text{TA}}(k_t|s_t), p^{\text{TA}}(z_t|s_t, k_t)$, discriminator $D(s)$, progress predictor $P(s)$
Initialize: replay buffer $\mathcal{D}$, high-level policies $\pi_\sigma^{\text{sem}}(k_t|s_t), \pi_\theta^{\text{lat}}(z_t|s_t, k_t)$, critic $Q_\phi(s_t, k_t, z_t)$, target network $Q_{\bar{\phi}}(s_t, k_t, z_t)$

for each iteration **do**
  for every H environment steps **do**
    $k_t \sim \pi^{\text{sem}}(k_t|s_t)$ ▷ sample semantic skill from policy
    $z_t \sim \pi^{\text{lat}}(z_t|s_t, z_t)$ ▷ sample latent skill from policy
    $s_{t'} \sim p(s_{t+H}|s_t, \pi^l(a_t|s_t, k_t, z_t))$ ▷ execute skill in environment
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{s_t, k_t, z_t, R_\Sigma, s_{t'}\}$ ▷ store transition in replay buffer, R_Σ = H-step summed reward
  for each gradient step **do**
    $\tilde{r} = (1-\kappa) \cdot R_\Sigma + \kappa \cdot \big[\log D(s_t) - \log\big(1 - D(s_t)\big)\big]$ ▷ compute combined reward
    $\tilde{r} = \mathbb{1}(D(s) < 0.5) \cdot \tilde{r} + \mathbb{1}(D(s) \geq 0.5) \cdot \tilde{r} \cdot P(s)$ ▷ optionally shape reward with progress predictor
    $\bar{Q} = \tilde{r} + \eta\big[Q_{\bar{\phi}}(s_{t'}, \pi_\theta^{\text{lat}}(z_{t'}|s_{t'}, \pi_\sigma^{\text{sem}}(k_{t'}|s_{t'})))$
      $- \big[\alpha_q D_{\text{KL}}\big(\pi_\sigma^{\text{sem}}(k_{t'}|s_{t'}), p^{\text{demo}}(k_{t'}|s_{t'})\big) \cdot D(s_{t'})$
      $+ \alpha_p D_{\text{KL}}\big(\pi_\sigma^{\text{sem}}(k_{t'}|s_{t'}), p^{\text{TA}}(k_{t'}|s_{t'})\big) \cdot \big(1 - D(s_{t'})\big)$
      $+ \alpha_l D_{\text{KL}}\big(\pi_\theta^{\text{lat}}(z_{t'}|s_{t'}, k_{t'}), p^{\text{TA}}(z_{t'}|s_{t'}, k_{t'})\big)\big)\big]$ ▷ compute Q-target
    $(\sigma, \theta) \leftarrow (\sigma, \theta) - \lambda_\pi \nabla_{(\sigma,\theta)}\big[Q_\phi(s_t, \pi_\theta^{\text{lat}}(z_t|s_t, \pi_\sigma^{\text{sem}}(k_t|s_t)))$
      $- \big[\alpha_q D_{\text{KL}}\big(\pi_\sigma^{\text{sem}}(k_t|s_t), p^{\text{demo}}(k_t|s_t)\big) \cdot D(s_t)$
      $+ \alpha_p D_{\text{KL}}\big(\pi_\sigma^{\text{sem}}(k_t|s_t), p^{\text{TA}}(k_t|s_t)\big) \cdot \big(1 - D(s_t)\big)$
      $+ \alpha_l D_{\text{KL}}\big(\pi_\sigma^{\text{lat}}(z_t|s_t, k_t), p^{\text{TA}}(z_t|s_t, k_t)\big)\big)\big]$ ▷ update high-level policy weights
    $\phi \leftarrow \phi - \lambda_Q \nabla_\phi\big[\frac{1}{2}\big(Q_\phi(s_t, k_t, z_t) - \bar{Q}\big)^2\big]$ ▷ update critic weights
    $\alpha_q \leftarrow \alpha_q - \lambda_\alpha \nabla_{\alpha_q}\big[\alpha_q \cdot (D_{\text{KL}}(\pi_\sigma^{\text{sem}}(k_t|s_t), p^{\text{demo}}(k_t|s_t)) - \delta_q)\big]$ ▷ update alpha values
    $\alpha_p \leftarrow \alpha_p - \lambda_\alpha \nabla_{\alpha_p}\big[\alpha_p \cdot (D_{\text{KL}}(\pi_\sigma^{\text{sem}}(k_t|s_t), p^{\text{TA}}(k_t|s_t)) - \delta_p)\big]$
    $\alpha_l \leftarrow \alpha_l - \lambda_\alpha \nabla_{\alpha_l}\big[\alpha_l \cdot (D_{\text{KL}}(\pi_\theta^{\text{lat}}(z_t|s_t, k_t), p^{\text{TA}}(z_t|s_t, k_t)) - \delta_l)\big]$
    $\bar{\phi} \leftarrow \tau\phi + (1-\tau)\bar{\phi}$ ▷ update target network weights
return trained high-level policies $\pi_\sigma^{\text{sem}}(k_t|s_t), \pi_\theta^{\text{lat}}(z_t|s_t, k_t)$

A Full Algorithm

We present a detailed description of the downstream RL algorithm for our STAR approach in Algorithm 2. It builds on soft actor-critic [32, 34]. In contrast to the original SAC we operate in a hybrid action space with mixed discrete and continuous actions: $\pi^{\text{sem}}(k|s)$ outputs discrete semantic skill IDs and $\pi^{\text{lat}}(z|s, k)$ outputs continuous latent variables.

For all input hyperparameters we use the default values from Pertsch et al. [16, 14] and only adapt the regularization weights α_q, α_p and α_l for each task. They can either be set to a fixed value or automatically tuned via dual gradient descent in lines 24-26 by setting target parameters δ_q, δ_p and δ_l [34].

B Overview of Pertsch et al. [14]

While the goal of our work is to imitate semantic skills *across* domains, we build on ideas from Pertsch et al. [14], which use *in-domain* demonstrations. Pertsch et al. [14] study demonstration-guided RL using a two-layer hierarchical policy architecture: a high-level policy $\pi^h(z|s)$ outputs temporally extended actions, or *skills*, as learned latent representation z. The skill z gets decoded into actions a by a learned low-level policy $\pi^l(a|s, z)$. Here, z captures a skill's behavior in terms of its low-level actions instead of its semantics. Pertsch et al. [14] assume access to two datasets: demonstration trajectories $\mathcal{D}^{\text{demo}}$ which solve the task at hand and a task-agnostic dataset $\mathcal{D}^{\text{TA}}$ of state-action trajectories from a range of prior tasks. First, they pre-train the latent skill representation

z and the low-level policy $\pi^l(a|s,z)$ using $\mathcal{D}^{\text{TA}}$. Next, they use the demonstration dataset $\mathcal{D}^{\text{demo}}$ and the task-agnostic dataset $\mathcal{D}^{\text{TA}}$ to learn a demonstration prior $p^{\text{demo}}(z|s)$ and a task-agnostic prior $p^{\text{TA}}(z|s)$ over z. The former captures the distribution over skills in the demonstrations, while the latter represents the skills in the task-agnostic dataset. Additionally, they use both datasets to train a discriminator $D(s)$ to distinguish states sampled from the task-agnostic and demonstration data. Both pre-trained prior distributions are used to regularize the high-level policy $\pi^h(z|s)$ during RL: when $D(s)$ classifies a state as part of the demonstrations, the policy is regularized towards $p^{\text{demo}}(z|s)$, encouraging it to imitate the demonstrated skills. In states which $D(s)$ classifies as outside the demonstration support, the policy is regularized towards $p^{\text{TA}}(z|s)$, encouraging it to explore the environment to reach back onto the demonstration support. The optimization objective for $\pi^h(z|s)$ is:

$$\mathbb{E}_{\pi^h}\Big[R(s,a) \underbrace{-\alpha_q D_{\text{KL}}\big(\pi^h(z|s), p^{\text{demo}}(z|s)\big)\cdot D(s)}_{\text{demonstration prior regularization}} \underbrace{-\alpha_p D_{\text{KL}}\big(\pi^h(z|s), p^{\text{TA}}(z|s)\big)\cdot (1-D(s))}_{\text{task-agnostic prior regularization}}\Big]. \tag{6}$$

Crucially, the learned skills z do not represent semantic skills, but instead reflect the underlying sequences of low-level actions. Thus, Pertsch et al. [14]'s approach is unsuitable for cross-domain imitation, since a policy would imitate the demonstration's low-level actions instead of its semantics.

C Implementation Details

C.1 Skill Learning

We summarize the pre-training objectives of all model components in Table 1. We instantiate all components with deep neural networks. We use a single-layer LSTM with 128 hidden units for the inference network and 3-layer MLPs with 128 hidden units for the low-level policy. The skill-representation z is a 10-dimensional continuous latent variable. All skill priors are implemented as 5-layer MLPs with 128 hidden units. The semantic skill priors output logits of a categorical distribution over k, the non-semantic prior outputs mean and log-variance of a diagonal Gaussian distribution over z. We use batch normalization after every layer and leaky ReLU activations. We auto-tune the regularization weight β for training the low-level skill policy using dual gradient descent and set the target to $2\text{e}{-2}$ for the maze and $1\text{e}{-2}$ for all kitchen experiments.

When training on image-based human data we add a 6-layer CNN-encoder to the semantic skill prior $p^{\text{TA}}(k|s)$ trained on the source domain dataset $\mathcal{D}_S$. The encoder reduces image resolution by half and doubles the number of channels in each layer, starting with a resolution of 64x64 and 8 channels in the first layer. We use batch normalization and leaky ReLU activations for this encoder too.

The demonstration discriminator $D(s)$ is implemented as a 2-layer MLP with 32 hidden units and no batch normalization to avoid overfitting. We use a sigmoid activation in it's final layer to constrain its output in range $(0, 1)$.

For cross-domain state matching we use a symmetric temporal window with $\gamma^-, \gamma^+ = 0.99$. Only in the experiments with missing skills (see Section 5.3) we set $\gamma^-, \gamma^+ = 0$.

All networks are optimized using the RAdam optimizer [35] with parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$, batch size 128 and learning rate $1\text{e}{-3}$. The computational complexity of our approach is comparable to that of prior skill-based RL approaches like Pertsch et al. [16]. On a single NVIDIA V100 GPU we can train the low-level policy and all skill priors in approximately 10 hours and the demonstration discriminator in approximately 3 hours.

C.2 Semantic Imitation

The high-level policies $\pi^{\text{sem}}(k|s)$ and $\pi^{\text{lat}}(z|s,k)$ are implemented as 5-layer MLPs with batch normalization and ReLU activations. The former outputs the logits of a categorical distribution over k, the latter the mean and log-variance of a diagonal Gaussian distribution over z. We initialize the semantic high-level policy with the pre-trained demonstration skill prior $p^{\text{demo}}(k|s)$ and the non-semantic high-level policy with the pre-trained task-agnostic latent skill prior $p^{\text{TA}}(z|s,k)$. We implement the critic as a 5-layer MLP with 256 hidden units per layer that outputs a $|\mathcal{K}|$-dimensional vector of Q-values. The scalar Q-value is then computed as the expectation under the output distribution of π^{sem}.

Table 1: List of all pre-trained model components, their respective functionality and their pre-training objectives. We use $\lfloor \cdot \rfloor$ to indicate stopped gradients and τ^T to denote demonstration trajectories relabeled with matched target domain states from Section 4.3.

MODEL	SYMBOL	DESCRIPTION	TRAINING OBJECTIVE
Skill Policy	$\pi^l(a\|s,k,z)$	Executes a given skill, defined by semantic skill ID and low-level execution latent.	Equation (1)
Demonstration Semantic Skill Distribution	$p^{\text{demo}}(k\|s)$	Captures semantic skill distribution of demonstration sequences.	$\mathbb{E}_{s,a,k\sim\tau^T}\Big[-\sum_{i=1}^{K} k_i \cdot \log p^{\text{demo}}(k_i\|s)\Big]$
Task-Agnostic Semantic Skill Prior	$p^{\text{TA}}(k\|s)$	Captures semantic skill distribution of task-agnostic prior experience.	$\mathbb{E}_{s,a,k\sim\mathcal{D}_T}\Big[-\sum_{i=1}^{K} k_i \cdot \log p^{\text{TA}}(k_i\|s)\Big]$
Task-Agnostic Low-level Execution Prior	$p^{\text{TA}}(z\|s,k)$	Captures distribution over low-level execution latents from task-agnostic prior experience.	$\mathbb{E}_{s,a,k\sim\mathcal{D}_T}\Big[D_{\text{KL}}\big(\lfloor q(z\|s,a,k)\rfloor, p^{\text{TA}}(z\|s,k)\big)\Big]$
Demonstration Support Discriminator	$D(s)$	Determines whether a state is within the support of the demonstrations.	$-\frac{1}{2}\cdot\Big[\underbrace{\mathbb{E}_{s\sim\tau^T}\big[\log D(s)\big]}_{\text{demonstrations}} + \underbrace{\mathbb{E}_{s\sim\mathcal{D}_T}\big[\log\big(1-D(s)\big)\big]}_{\text{task-agnostic data}}\Big]$

We use batch size 256, replay buffer capacity of $1\text{e}6$ and discount factor $\gamma = 0.99$. We warm-start training by initializing the replay buffer with 2000 steps. We use the Adam optimizer [36] with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and learning rate $3\text{e}{-4}$ for updating policy and critic. Analogous to SAC, we train two separate critic networks and compute the Q-value as the minimum over both estimates to stabilize training. The target networks get updated at a rate of $\tau = 5\text{e}{-3}$. The latent high-level policy's actions are limited in the range $[-2, 2]$ by a $\tanh$ "squashing function" (see Haarnoja et al. [32], appendix C). We set all α parameters to fixed values of 10 in the maze navigation task and $5\text{e}{-2}$ in all kitchen tasks.

For reward computation we set the factor $\kappa = 0.9$, i.e., we blend environment and discriminator-based rewards. In practice, we find that we can improve convergence speed by using a *shaped* discriminator reward that increases towards the end of the demonstration. This is comparable to goal-proximity based rewards used in in-domain imitation, e.g., Lee et al. [37]. To compute the shaped reward, we pre-train a progress predictor $P(s)$ along with the discriminator $D(s)$. $P(s)$ estimates the time step of a state within a demonstration relative to the total length of the demonstration, thus its outputs are bound in the range $[0, 1]$. We implement the progress predictor as a simple 3-layer MLP with a sigmoid output activation. During RL training we can then compute the shaped reward as:

$$r(s,a) = \kappa \cdot R(s,a) + (1-\kappa) \cdot \begin{cases} P(s) \cdot R_D & \text{if } P(s) \geq 0.5 \\ R_D & \text{otherwise} \end{cases}$$
$$\text{with } R_D = \log D(s_t) - \log\big(1 - D(s_t)\big) \tag{7}$$

For all RL results we average the results of three independently seeded runs and display mean and standard deviation across seeds. The computation time for these experiments varies by environment and is mainly determined by the simulation time of the used environments. Across all environments we can typically finish downstream task training within <12h on a single NVIDIA V100 GPU.

C.3 Comparisons

SPiRL. We follow the approach of Pertsch et al. [16] which first trains a latent skill representation from task-agnostic data and then uses a pre-trained task-agnostic prior to regularize the policy during downstream learning. To allow for fair comparison, we adapt the SPiRL approach to work with our semantic skill model. In this way both SPiRL and STAR use the same set of learned semantic skills. During downstream task learning we regularize both high-level policies $\pi^{\text{sem}}(k|s)$

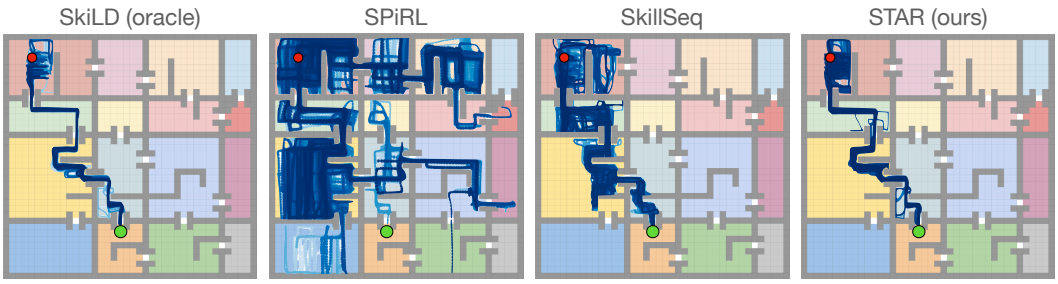


Figure 8: Qualitative maze results. We visualize the trajectories of the different policies during training. The **SkiLD** approach leverages in-domain demonstrations to quickly learn how to reach the goal. **SPiRL** leverages skills from the task-agnostic dataset to widely explore the maze, but fails to reach the goal. **SkillSeq** makes progress towards the goal, but can get stuck in intermediate rooms, leading to a substantially lower success rate than the oracle method. Our approach, **STAR**, is able to match the performance of the oracle baseline and quickly learn to reach the goal while following the sequence of demonstrated rooms.

and $\pi^{\text{lat}}(z|s,k)$ using the corresponding task-agnostic skill priors $p^{\text{TA}}(k|s)$ and $p^{\text{TA}}(z|s,k)$, analogous to the task-agnostic skill prior regularization in the original SPiRL work.

SkiLD. We similarly adapt SkiLD [14] to work with our learned semantic skill model. In contrast to the SPiRL comparison, we now regularize both high-level policies with skill distributions trained on the target domain demonstrations whenever $D(s)$ classifies a state as being part of the demonstration support (see Section B).

SkillSeq. We pre-train a skill-ID conditioned policy on the task-agnostic target domain dataset $\mathcal{D}_T$ using behavioral cloning. We split this policy into a 3-layer MLP encoder and a 3-layer MLP policy head that produces the output action. The policy has an additional 3-layer MLP output head that is trained to estimate whether the current skill terminates in the input state. We use the semantic skill labels k in the task-agnostic dataset to determine states in which a skill ends and train the termination predictor as a binary classifier. During downstream learning, we use a programmatic high-level policy that has access to the true sequence of semantic skills required to solve the downstream task and conditions the low-level policy on these skill IDs one-by-one. The skill ID is switched to the next skill when the pre-trained termination predictor infers the current state as a terminal state for the current skill. For fair comparison we use online RL for finetuning the skill-conditioned policy via soft actor-critic (SAC, Haarnoja et al. [32]).

BC+RL. We train a policy directly on the source domain demonstrations via behavioral cloning. We then use this pre-trained policy to initialize the policy during target task training in the target domain. We fine-tune this initialization using SAC with the rewards provided by the target environment. Similar to Rajeswaran et al. [7], Nair et al. [8] we regularize the policy towards the pre-trained BC policy during downstream learning.

C.4 Environments and Data Collection

Maze navigation. We generate two maze layouts with the same number of rooms. We indicate a room's semantic ID via its color in Figure 10. We ensure the same "room connectivity" between both layouts, i.e., corresponding semantic rooms have the same openings between each other. For example the yellow room connects to the blue room but not to the green room in both layouts. This ensures that we can follow the same sequence of semantic rooms in both environments. While we ensure that the semantic layout of the mazes is equivalent, their physical layout is substantially different: the mazes are rotated by 180 degrees, for example the red room is in the bottom right corner for the first maze but in the top left corner for the second. Additionally, the layout of individual rooms and the positions of obstacle walls change between the domains. As a result, simple imitation of the low-level planar velocity actions from one domain will not lead to successfully following the same sequence of semantic rooms in the other domain. We define a total of 48 semantic skills: one for each room-to-room traversal, e.g., "go from the red room to the green room", and one for reaching a

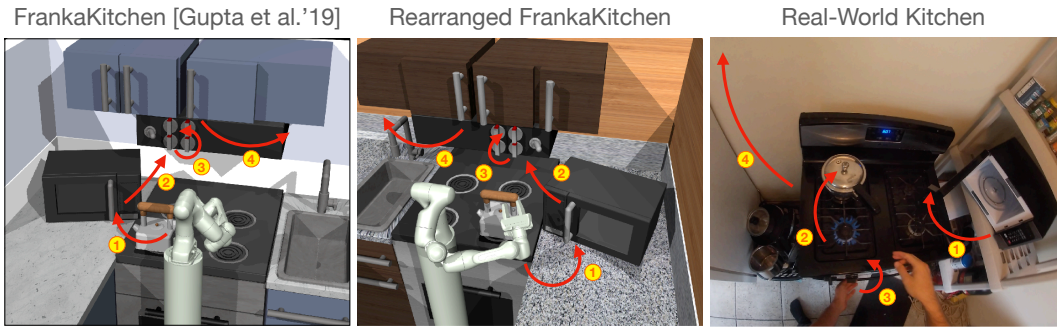


Figure 9: Three semantically equivalent kitchen environments. **Left**: FrankaKitchen environment [3], **middle**: rearranged FrankaKitchen environment, **right**: real-world kitchen. In all three environments we define the same set of seven semantic object manipulation skills like "open the microwave", "turn on the stove" etc. The two simulated kitchen environments require different robot joint actuations to perform the same semantic skills. The real-world kitchen has a different agent embodiment (robot vs. human), layout and observation domain (low-dimensional state vs image observations).

goal within each room, e.g., "reach a goal in the green room". Thus, the semantic description of a demonstrated trajectory could for example be: "Go from the red room to the green room, then from the green to the beige room, . . . , then from the blue to the orange room and then reach a goal in the orange room."

Simulated Kitchen. We use the FrankaKitchen environment of Gupta et al. [3] (see Figure 9, left) and define a set of seven semantic manipulation skills: opening the microwave, opening the slide and hinge cabinet, turning on bottom and top stove and flipping the light switch. We also create a rearranged version of the kitchen environment (Figure 9, middle) with different layout and visual appearance but the same set of semantic interaction options. In both environments we use the state-action definition of Gupta et al. [3]: (1) a 60-dimensional state representation consisting of the agent's joint state as well as object states like opening angles or object pose, (2) a 9-dimensional action space consisting of 7 robot joint velocities and two gripper finger positions.

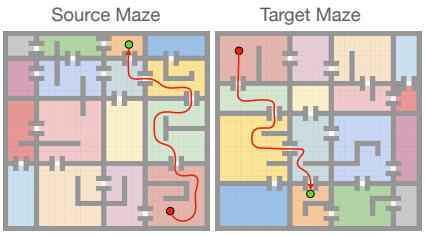


Figure 10: Source and target semantic maze navigation environments. A room's color indicates its semantic ID. The red trajectory shows the traversal of semantic rooms demonstrated in the source domain and the corresponding trajectory in the target domain. The low-level planar velocity commands required to follow the demonstration in the target domain is substantially different.

For the FrankaKitchen environment we can use the data provided by Gupta et al. [3]: 600 human teleoperated sequences each solving four different semantic tasks in sequence. In the newly created rearranged kitchen environment we collect a comparable dataset by controlling the robot via trajectory optimization on a dense reward function. We use the CEM implementation of Lowrey et al. [38]. For both datasets we label the semantic skills by programmatically detecting the end state of an object interaction using the low-dimensional state representation.

Real-World Kitchen. Data collection is performed by fixating a GoPro camera to the head of a human data collector which then performs a sequence of semantic skills. The camera is angled to widely capture the area in front of the human. During data collection and within each trajectory we vary the viewpoint, skill execution speed and hand used for skill execution. We collect 20 human demonstration sequences for the task sequence: open microwave, move kettle, turn on stove, open cabinet. We then automatically generate semantic skill predictions via zero-shot inference with a pre-trained action recognition model. Specifically, we use the publicly available SlowFast model trained on the EPIC Kitchens 100 dataset [33, 39]. The model takes in a window of 32 consecutive video images at a 256×256 px resolution and outputs a distribution over 97 verb and 300 object classes. Since our simulated FrankaKitchen target environment does not support the same set of

skills, we define a mapping from the output of the EPIC Kitchens model to the applicable skills in the Franka Kitchen environment, e.g., we will map outputs for the verb "open" and the noun "microwave" to the "open microwave" skill in FrankaKitchen. Note that some skill distinctions in the FrankaKitchen environment are not supported by EPIC Kitchens, like "turn on top burner" vs "turn on bottom burner". In such cases we map the outputs of the EPIC Kitchens model to a single skill in the target environment. With this skill mapping we finetune the EPIC Kitchens model for outputting the relevant classes. Note that this model finetuning is performed with the original EPIC Kitchens data, i.e., *no* additional, domain specific data is used in this step and no additional annotations need to be collected. This finetuning is performed such that the resulting model directly outputs a distribution over the relevant skills. Alternatively, the relevant skills could be extracted from the output of the original model and the distribution could be renormalized.

To generate the skill predictions for the human video demonstrations, we move a sliding window of 32 frames over the demonstrations and generate a prediction in each step using the EPIC Kitchens model. We pad the resulting skill distribution sequence with the first and last predicted skill distribution to obtain the same number of skill predictions as there are frames in the demonstration video. Then we use the sequence of skill distributions to perform cross-domain matching and semantic imitation as detailed in Section 4.2, without any changes to the algorithm.

D Imitation Learning Results

We evaluate our approach in the "pure" imitation learning setting in the kitchen environment. Here, we assume *no access* to environment rewards. Instead, we rely solely on the discriminator-based reward learned from the cross-domain demonstrations to guide learning (see Section 4.2). We present evaluations in the FrankaKitchen environment in Figure 11. Our approach STAR is able to learn the target task from demonstrations without any environment rewards, although learning is somewhat slower than in the demonstration-guided RL setting with environment reward access. In contrast, standard imitation learning approaches are unable to learn the task since they struggle with the large domain gap between source domain demonstrations and target domain execution. These results show that our approach STAR is applicable both, in the demonstration-guided RL setting *with* environment rewards, and in the imitation learning setting *without* environment rewards.

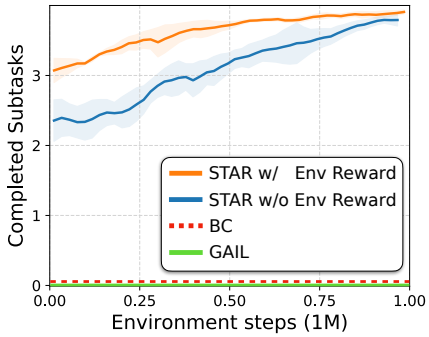


Figure 11: Imitation learning on the simulated FrankaKitchen task. Our approach STAR is able to learn the target task even without access to any environment rewards, while common imitation learning approaches fail to learn the task due to the large domain gap between source demonstrations and target task environment.

E Label Noise Robustness Analysis

An important aspect for the scalability of an approach is its ability to cope with noise in the training data. While prior work on skill-based RL has investigated the robustness of such approaches to suboptimal behavior in the training data [16], we will focus on an aspect of the training data that is specifically important for our cross-domain imitation approach: the semantic skill labels. In this section, we investigate the robustness of our approach to different forms of noise on the semantic skill labels. Such noise can either be introduced through inaccuracies in the manual labeling process or via an automated form of skill labeling, as performed with the EPIC kitchens models in Section 5.2. To cleanly investigate the robustness to *different forms* of skill label noise, we start from a noise-free set of labels, which we can easily obtain programmatically in the simulated FrankaKitchen environment. We then artificially perturb the labels to introduce artifacts that mimic realistic labeling errors. This allows us to (1) investigate different forms of noise independently and (2) vary the magnitude of the introduced noise in a controlled way.

Specifically, we introduce noise along three axis:

- **skill length noise**: artificially perturbs the length of a labeled skill within a range $[1 - l_n \dots 1 + l_n]$ of the true length of the skill, mimicking a labeler's uncertainty on when exactly a skill ends

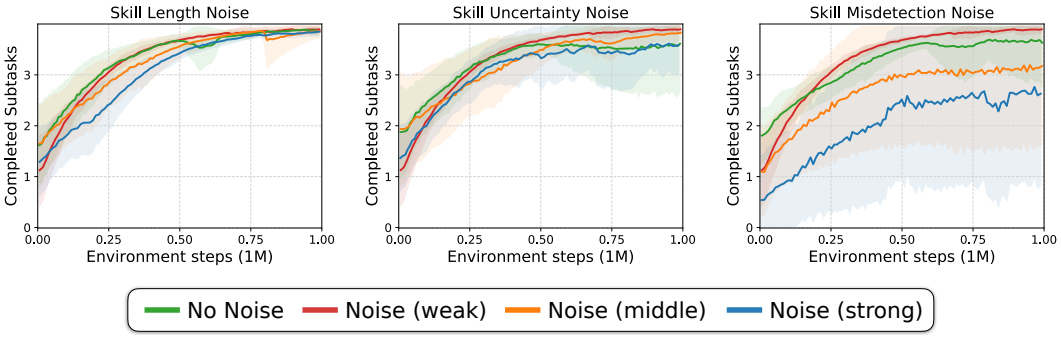


Figure 12: Robustness of our approach, STAR, to different forms of noise in the semantic skill labels. Our approach is robust to noise in the length of annotated skills and uncertainty between different skills. While STAR is also shows some robustness to completely incorrect skill labels, frequent and confident mis-detections / mis-labelings can lead to errors during the cross-domain matching and thus impair learning performance on the target task.

Table 2: Parametrization of the noise levels for the skill label robustness experiment.

	SKILL LENGTH NOISE	SKILL UNCERTAINTY NOISE	SKILL MISDETECTION NOISE
Varied Parameter	l_n (Percentual length noise window)	N_n (Number of uncertain segments)	N_m (number of misdetected segments)
Weak Noise	10 %	1	1
Middle Noise	20 %	2	2
Strong Noise	30 %	3	3

- **skill uncertainty noise**: perturbs the distribution over detected skills around N_n transition between skills by adding probability weight to erroneous skills produced via a random walk, mimicking the uncertainty e.g., produced by a pre-trained action recognition model
- **skill misdetection noise**: adds N_m incorrectly detected skill segments at randomly sampled points throughout the sequence of randomly sampled lengths, mimicking mis-labelings which can (rarely) occur in human data or (more frequently) in auto-labeled data

We show evaluations of our approach with different *levels* of noise along all three axis in Figure 12. We perform these evaluations in the simulated FrankaKitchen environment and average performance across 10 seeds to reduce the noise-induced variance in the results. The parameters of the different tested noise levels are detailed in Table 2.

The results in Figure 12 show that STAR is robust to a wider range of noise levels in the annotated skill length and uncertainty between the skills: the performance does not significantly change even with increased noise levels. However, we find that confident mis-predictions / mis-labelings of skills can have a negative impact on the performance. Particularly if mis-predictions happen frequently ("Noise (strong)"), states between the source and target domain can be mismatched, leading to worse target task performance. But we find that even in the case of mis-detections STAR is able to handle a moderate amount of such noise robustly, which is important for STAR's scalability to large and noisy real-world datasets.

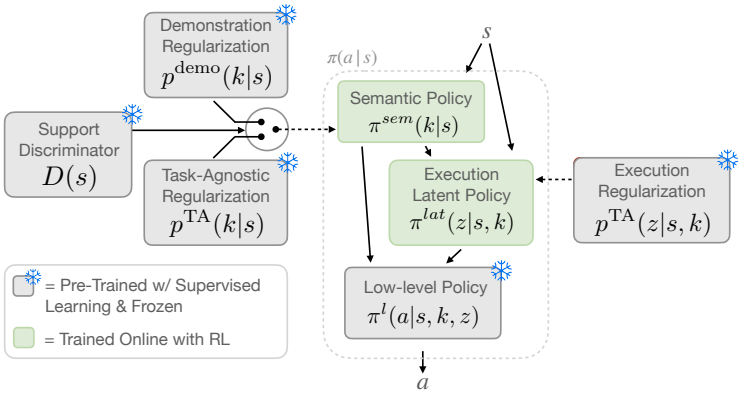


Figure 13: Visualization of all model components. The colors indicate the objective type used for training. Only the high-level policy is trained with online RL on the downstream task, all other components are pre-trained fully offline via supervised learning and frozen during downstream training.

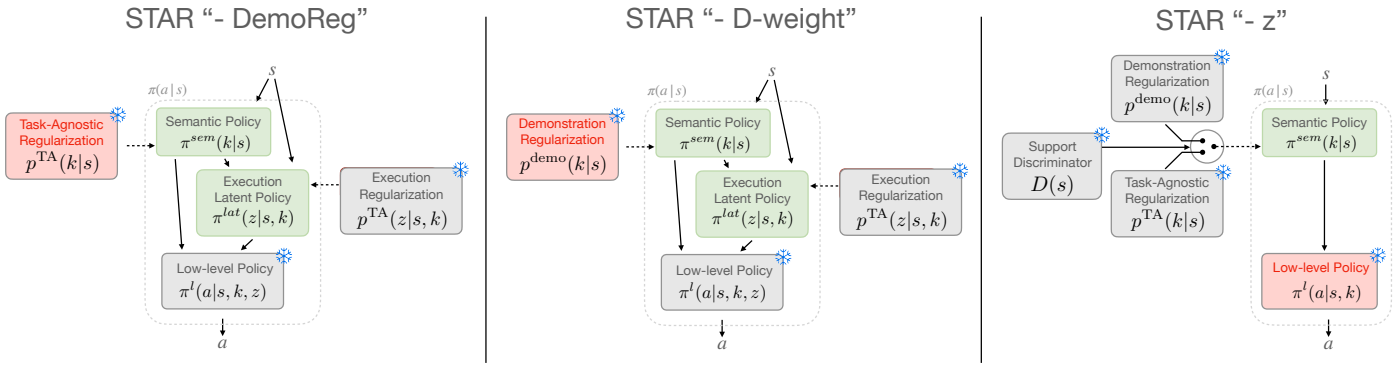


Figure 14: Overview of some of the performed ablations. **-DemoReg**: removes the demonstration regularization for the high-level semantic policy and uses only the task-agnostic prior for regularization, **-D-weight**: removes the discriminator-based weighting between demonstration and task-agnostic regularization and uses only the former for guiding the policy during downstream training, **-z**: removes the latent variable z from the low-level policy and instead uses a deterministic low-level policy and *no* execution latent policy.

F Detailed Ablation Description

We provide an overview of the components of our approach in Figure 13. The figure highlights that most components are trained offline with simple supervised objectives and then frozen during downstream task learning, making their training straightforward and reproducible. Only the high-level semantic and execution policy are trained via online RL on the downstream task.

We also provide a more detailed description of the performed ablation studies from Figure 5, right, below. These ablation studies demonstrate the importance of the different components of our model. Finally, we visualize the resulting models for multiple of our ablation studies in Figure 14.

STAR - D-reward. Ablates the discriminator-based dense reward (see Section 4.2). Instead trains the high-level policy only based on the environment-provided reward on the downstream task.

STAR - TempAgg. Ablates the temporal aggregation during cross-domain matching (see Section 4.3). Instead uses single state semantic skill distributions to find matching states.

STAR - DemoReg. Ablates the policy regularization with cross-domain skill distributions. Instead simply regularizes with task-agnostic skill priors derived from the target domain play data (see Figure 14, left).

STAR - D-weight. Ablates the discriminator-based weighting between demonstration and task-agnostic skill distributions. Instead always regularizes the high-level semantic policy towards the demonstration skill distribution (see Figure 14, middle).

STAR - z. Ablates the use of the latent execution variable z in the skill policy. Instead trains a simpler low-level policy *without* latent variable z and removes the execution latent policy (see Figure 14, right).

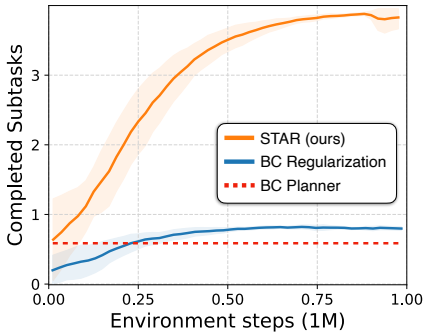
G Additional Ablation Experiments

We perform an additional ablation experiment to test whether replacing the high-level policy's weighted KL-regularization scheme from equation 3 with a simpler behavioral cloning regularization objective can lead to comparable performance. Concretely, we replace the policy's objective from equation 3 with:

$$\max_{\pi^h} \Big[Q(s,a) \underbrace{-\alpha \mathbb{E}_{k\sim\pi^{\text{sem}}(k|s)} p^{\text{demo}}(k|s)}_{\text{BC regularization}} \underbrace{-\alpha_l D_{\text{KL}}\big(\pi^{\text{lat}}(z|s,k), p^{\text{TA}}(z|s,k)\big)}_{\text{task-agnostic execution prior regularization}} \Big]. \tag{8}$$

We also experimented with removing the execution prior regularization term, i.e., setting $\alpha_l = 0$, but found it to be crucial for training since the initial policy rapidly degrades without it.

We report quantitative results on the human video demonstration to simulated kitchen manipulation task in the figure on the right. The *BC-Reg* objective in equation 8 obtains 75% lower performance than our full objective from equation 3. This is because the behavioral cloning regularization is also computed on states outside the demonstrations' support, leading to incorrect regularization. Instead, our approach uses the discriminator to only apply regularization *within* the support of the demonstrations.



We also add comparison to an even simpler baseline that clones the transferred semantic skill embeddings from the demonstrations, equivalent to a **semantic-level BC planner**. This approach does not perform well due to accumulating errors of the high-level planner (see figure on the right). Without online training, this approach cannot correct the shortcomings of the planner.