Residual Skill Policies: Learning an Adaptable Skill-based Action Space for Reinforcement Learning for Robotics

Anonymous Author(s)
Affiliation
Address
email

Abstract:
Skill-based reinforcement learning (RL) has emerged as a promising strategy to leverage prior knowledge for accelerated robot learning. Skills are typically extracted from expert demonstrations and are embedded in a latent space from which they can be sampled as actions by a high-level RL agent. However, this *skill-space* is expansive, and not all skills are relevant for a given robot state, making exploration difficult. Furthermore, the RL agent is limited to learning structurally similar tasks to those used to construct the skill-space. We firstly propose accelerating exploration in the skill-space using state-conditioned generative models to directly bias the high-level agent towards *sampling* skills relevant to a given state based on prior experience. Next, we propose a low-level residual policy for fine-grained *skill adaptation* to make skill-based RL generalise to task variations. Finally, we validate our approach across three challenging manipulation tasks that differ from those used to build the skill-space, demonstrating our ability to adapt to task variations while significantly accelerating exploration, outperforming prior works. Code and video are available on our project website: https://sites.google.com/view/reskill-rl

Keywords: Reinforcement Learning, Skill Learning, Transfer Learning

1 Introduction

Humans are remarkably efficient at learning new behaviours. We can attribute this to their ability to constantly draw on relevant prior experiences and adapt this knowledge to facilitate learning. Building on this observation, recent works have proposed various methods to incorporate the use of prior experience in deep reinforcement learning (RL) to accelerate robot learning.

One approach that has gained attention recently is the use of *skills* which are short sequences of atomic actions representing useful and task-agnostic behaviours extracted from datasets of expert demonstrations [1, 2, 3]. In the manipulation domain for example, such skills could include *move-left*, *grasp-object* and *lift-object*. These skills are typically embedded into a latent space, forming the action-space for a high-level RL policy. While yielding significant improvements over learning from *scratch*, there are still outstanding challenges for training RL agents using this latent *skill space*.

Firstly, naïve exploration by sampling uniformly over all skills can be extremely inefficient [3, 4, 2]. Typical skill spaces encode an expansive set of generic skills, with only a small fraction of skills relevant to be applied to a given robot state. Furthermore, the set of skills relevant to a given state do not typically cluster in the same neighbourhood in the skill space, i.e. the conditional density is *multi-modal*, making targeted sampling challenging. We address this issue of inefficient exploration by learning a *state-conditioned skill prior* over the skill space, which directly biases the agent's

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.

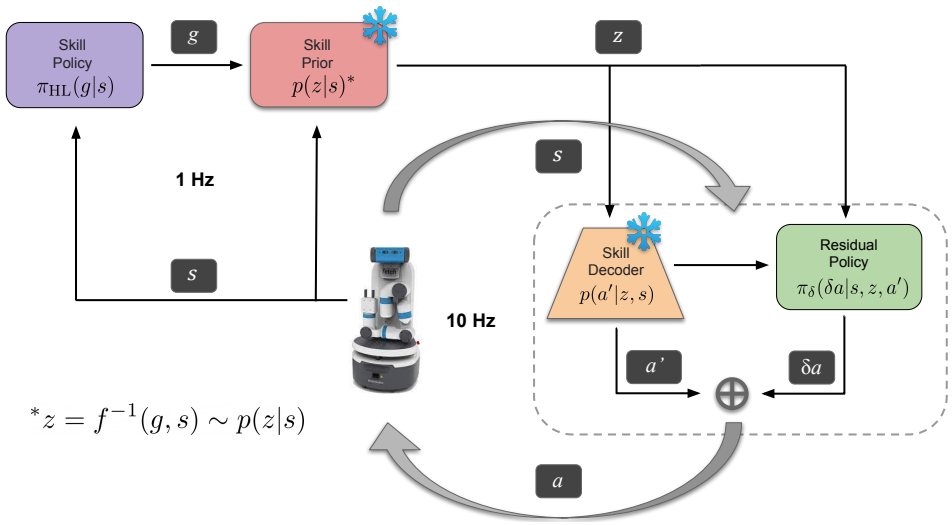


Figure 1: **Residual Skill Policies (ReSkill).** A skill-based learning framework for robotics. The skill prior transforms the agents action space to a state-conditioned skill space using normalising flows, where only the relevant skills for the current space are explored. The residual policy allows for fine-grained adaptation of the skills to environment variations and unseen tasks. The ❄ symbol signifies that the pre-trained weights for these skill modules are frozen during downstream task learning.

explorations towards sampling only those skills that are deemed relevant in its current state. Unlike previous works [3], our skill-prior captures the complex multi-modality of the skill space allowing us to sample directly from this density during exploration.

Secondly, current approaches to skill-based RL assume that the skills are optimal and that downstream tasks are drawn from the same distribution used to create the skill space. Consequently, skill-based learning methods have limited generality and adaptability to task variations. For example, if the available skills were extracted solely from demonstrations of block manipulation on an empty table, learning a downstream task with obstacles, object variations, or different friction coefficients may not be possible. In this case, using a non-exhaustive skill space as an action space cripples an RL agent's learning ability. We address this by introducing a low-level residual policy, enabling fine-grained *skill adaptation* of available skills to variations in the task not solvable by simply composing available skills. This relaxes the need for exhaustive and expert demonstration datasets to one that can be extracted from existing classical controllers, which are a cheap and readily available source of demonstration data for robotics.

We summarise the main contributions of our approach denoted as Residual Skill Policies (ReSkill) as follows: **(1)** a novel learning framework that can effectively re-purpose a handful of classical controllers as skills to solve a wide range of downstream RL tasks, **(2)** guiding exploration in a latent skill space by biasing the agent's exploration towards only those skills that are meaningful in a given state, and **(3)** a low-level residual policy that can adapt the skills to variations in the downstream task and environment. We evaluate ReSkill on 3 manipulation tasks that vary from those used for skill space construction and demonstrate the ability to both accelerate and attain higher performing final policies than all baselines.

2 Related Work

Skill-Based RL This has emerged as a promising approach to effectively leverage prior knowledge within the RL framework by embedding reusable skills into a continuous skill-space via stochastic latent variable models [3, 5, 6, 1, 2]. Skill-based latent variable models can provide temporal abstraction that allows for effective exploration of the search space, with the ability to draw on previously used behaviours to accelerate learning. However as described previously, the skill-space can be

difficult to explore. Pertsch et al. [3] proposed the use of a prior over skills to help better guide exploration. They learn a unimodal Gaussian prior that approximates the density of relevant skills for a given state, and regularise the downstream RL agent towards the prior during training to indirectly bias exploration. Singh et al. [4] proposed a more expressive strategy for learning these priors via the use of state-conditioned generative models. While this was conducted at the atomic action level, they show that we can directly bias exploration by sampling relevant actions from this density. In this work we extend this idea to skill-based action spaces.

Combining RL with Classical Control Our work is closely related to methods that seek to combine the strengths of classical control and RL for accelerated learning. In contrast to expert datasets or human demonstrators, classical control systems are cheap and readily available for most robotics tasks. Johannink et al. [7] proposed the residual reinforcement learning framework in which an RL agent learns an additive residual policy that modifies the behaviours of a predefined handcrafted controller to simplify exploration. Other works in this area have shown that combining the two modes of control allows for sample efficient initialisation [8, 9, 10], temporal abstraction for accelerated learning [11], and safe exploration [12]. While strongly motivating the re-use of existing handcrafted controllers, these methods are restricted to solving only one particular task for which the controller was designed, requiring a new controller for each new task. This limits the generality of these approaches to novel tasks. In this work, we decompose these controllers into task agnostic skills and explore how they can be repurposed for a wide range of tasks.

Hierarchical RL (HRL) These methods enable autonomous decomposition of challenging long-horizon decision-making tasks for efficient learning. Typical HRL approaches utilise a high-level policy that selects from a set of low-level temporally extended actions or options [13] that provide broader coverage of the exploration space when executed [14]. Prior work in hierarchical reinforcement learning can be divided into two main categories: learning low-level and high-level policies through active interaction with an environment [15, 16, 17], and learning options from demonstrations before re-composing them to perform long-horizon tasks through RL [18, 19]. Our work shares similarities with both strategies. We follow the latter approach using a pre-trained low-level skills module from demonstration datasets on which a high-level policy operates for skill composition while simultaneously learning a low-level residual policy capable of adapting these skills online.

3 Problem Formulation

We focus on skill-based RL which leverages *skills* as the actions for a high-level policy π_{HL}, where a skill $\boldsymbol{a}$ is defined as a sequence of atomic actions $\{a_t, ..., a_{t+H-1}\}$ over a fixed horizon H. Skills are typically embedded into a continuous latent space $\mathcal{Z}$ from which π_{HL} can sample during exploration [3, 2, 1]. Latent *skill-spaces* expressively capture vast amounts of task-agnostic prior experience for re-use in RL. However, they pose several challenges that can negatively impact downstream learning. Current methods rely on expert and exhaustive demonstration datasets that capture skills for a wide range of potential task variations. Such datasets can be tedious and expensive to obtain. As the skill space abstracts the RL agent's atomic action space, the generality of skill-based RL is greatly limited to tasks that closely match the distribution of those captured by the dataset. This poses a trade-off between generality and sample-efficiency in downstream learning [20]. Furthermore, latent skill spaces are expansive and can be arbitrarily structured, making random exploration difficult. Learning structure within this space can better inform exploration to accelerate downstream learning.

The downstream task is formulated as a Markov Decision Process (MDP) defined by a tuple (S, A, T, R, γ) consisting of states, actions, transition probabilities, rewards and discount factor. We learn the high-level policy $\pi_{\text{HL}}(z|s)$ to select the appropriate latent skills z in a given state s to maximise the discounted sum of rewards $J = \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} \gamma^t r_t\right]$, where T is the episode horizon. The latent skill z is executed in the environment for the skill horizon H before another skill is selected by π_{HL}.

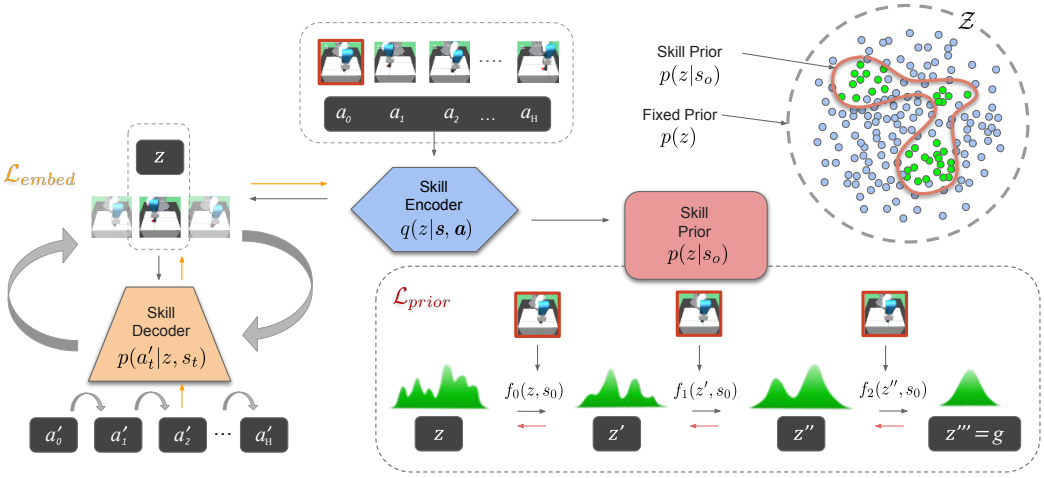


Figure 2: **Schematic for learning the state-conditioned skill space.** We train a VAE *embedding module* which encodes skills into a latent embedding space $\mathcal{Z}$. This module is comprised of an encoder and closed-loop skills decoder, where the decoder recovers atomic actions from a latent skill embedding z. The *state-conditioned skill prior* module is used to generate relevant skills for a given robot state which the downstream RL agent can directly sample from. This conditional density is multi-modal in the skill space and we estimate it using normalising flows. Both modules are trained jointly, with coloured arrows illustrating the gradient flow between them.

4 Approach

We aim to re-purpose existing classical controllers as skills to facilitate RL in solving new tasks. We decompose demonstration trajectories produced by these controllers into task-agnostic skills and embed them into a continuous space $\mathcal{Z}$ using stochastic latent variable models. To facilitate exploration for downstream RL in this skill-space, we propose a state-conditioned *skill prior* $p(z|s_0)$ over skills that directly biases the agent towards sampling skills relevant to a given state based on prior experience. Finally, to enable truly general-purpose learning with the skill-space, we propose a low-level residual policy $\pi_\delta(\delta a|\cdot)$ that can adapt the embedded skills to task variations not captured by the demonstration data. This gives the RL agent access to its atomic action space and relaxes the need for both expert and exhaustive datasets.

Our approach can be decomposed into three sub-problems: (1) skill extraction from existing controllers, (2) learning a skill embedding and skill prior, and (3) learning a hierarchical RL policy within this skill-space with a low-level, residual skill adaptation policy. We discuss each of these steps in more detail below.

4.1 Data Collection

We now describe how we build a dataset of task-agnostic skills for the high level RL policy to use. We collect robot trajectories acquired through performing demonstrations for a range of tasks using a handcrafted controller. In this work, we script 2 simple controllers for a 7-DoF robotic arm for *pushing* and *pick and place* manipulation tasks on an empty table. Each trajectory relates to a demonstration for a single task and does not necessarily have to be optimal. Atomic state-action pairs are recorded for the trajectory and a skill is obtained by slicing a length H segment from the trajectory. We utilise both the sequence of actions $\boldsymbol{a}$ as well as the corresponding states $\boldsymbol{s}$ for a given skill to learn the state-conditioned skill space described in the next section.

We collect 40K trajectories across all controllers, resulting in approximately 1.6M state-action pairs. The state consists of a 19D vector comprised of joint angles, joint velocities, gripper position and object position, while the actions are continuous 4D vectors representing end-effector and gripper velocities in Cartesian space. More details about data generation and the handcrafted controllers can be found in Appendix B.

4.2 Learning a State-Conditioned Skill Space for RL

Learning a state-conditioned skill space consists of two key steps: (1) embedding the extracted skills into a latent space; (2) learning a state-conditioned prior over the skills which we can sample from during exploration. Figure 2 provides a summary of our approach and we describe each component in detail below.

4.2.1 Embedding the skills

We follow the approach proposed in prior works [21, 1], where skills $\boldsymbol{a}$ are embedded into a latent space using a variational autoencoder (VAE) [22]. The VAE comprises of a probabilistic encoder q_ϕ and decoder p_θ with network weights ϕ and θ respectively, which maps the atomic representation of skills to the latent embedding space and vice versa. Our encoder $q_\phi(z \mid \boldsymbol{s}, \boldsymbol{a})$ jointly processes the full state-action sequence, while our decoder $p_\theta(a_t \mid z, s_t)$ reconstructs individual atomic actions conditioned on current state s_t and skill embedding z. The loss function for our VAE for a single training example is given by

$$\mathcal{L}_{embed} = \mathbb{E}_{q_\phi(z|\mathbf{s},\mathbf{a})}\left[\sum_{t=0}^{H-1} \log p_\theta\left(a_t \mid z, s_t\right)\right] - \beta D_{KL}(q_\phi\left(z \mid \boldsymbol{s}, \boldsymbol{a}\right) \parallel p(z)), \tag{1}$$

where $p(z) \sim \mathcal{N}(0, I)$ and $\beta \in [0, 1]$ is the weighting for the regularisation term [22]. Note that in contrast to prior works [3], we specify a *closed-loop* decoder which considers the current robot state s_t as well as skill z when recovering actions; we found this significantly improved performance in downstream learning.

4.2.2 Learning the state-conditioned skill prior

The nature of the VAE reconstruction loss described in Equation (1) allows us to sample skills from the latent space by using the prior $p(z) \sim \mathcal{N}(0, I)$. However, sampling directly from the prior can be inefficient, since the samples will range across the full, expansive set of available skills as well as regions that may not constitute any embeddings as shown in Figure 3. In reality, for a particular robot state, only a small subset of these skills are relevant for exploration. To address this inefficiency, we propose to learn a *state-conditioned skill prior* over the latent embedding space, which allows us to more frequently sample the *relevant skills* likely to be useful in a given state, thus accelerating exploration.

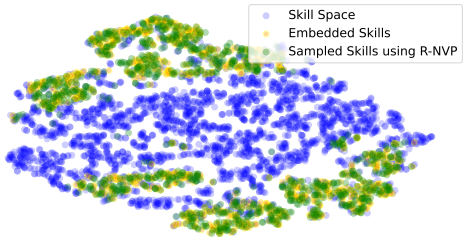
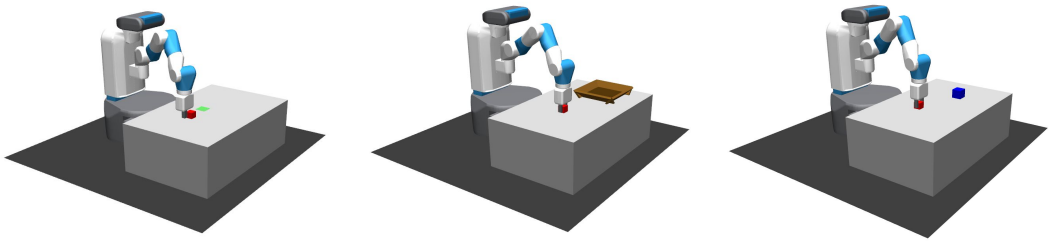


Figure 3: **t-SNE of the skill embedding space.** Blue denotes samples drawn from the VAE prior $p(z)$. Yellow denotes embedded skills drawn from the dataset. Green denotes samples using our trained skill prior. Note the multi-modality across the latent space. The skill prior samples from regions containing embedded and relevant skills, reducing the chances of sampling irrelevant regions (blue) not containing any meaningful skills.

Mathematically, we are looking to learn a conditional probability density over the latent skill-space $p(z|s_0)$ that we can draw samples from when training π_{HL}. This conditional density is typically highly multi-modal, i.e. for a given robot state, many skills can be relevant that may be far apart in the topology of the latent skill space. Prior works [3] propose a simple unimodal density (e.g. Gaussian) which does not sufficiently capture this multi-modality and hence is unsuitable to directly sample skills from.

One of our contributions is around gracefully handling this multi-modality by using the real-valued non-volume preserving transformations (real NVP) method by Dinh et al. [23] based on normalising flows [24]. Conditional real NVP [25] learns a mapping $f : \mathcal{Z} \times \mathcal{S} \rightarrow \mathcal{G}$ which we can use to generate samples from $p(z|s)$. Spaces $\mathcal{Z}$ and $\mathcal{G}$ are identical, i.e. $\mathcal{Z} \cong \mathcal{G} \cong \mathbb{R}^d$ and furthermore f is bijective



(a) Slippery Push (b) Table Cleanup (c) Pyramid Stack

Figure 4: **Downstream Tasks.** Robotic manipulation tasks involving a 7 DoF manipulator arm from Fetch Robotics. Each task exhibits a physical or dynamical variation to the environment that was not present in the skill extraction environment.

between $\mathcal{Z}$ and $\mathcal{G}$ for a fixed state s. Sampling is achieved as follows: first sample $g \sim p_{\mathcal{G}}(g)$ from a simple distribution (e.g. $p_{\mathcal{G}}(g) \sim \mathcal{N}(0, I)$) and then transform g into skill space $\mathcal{Z}$ using f^{-1}, i.e. $f^{-1}(g, s) \sim p(z|s)$. Since f fully defines $p(z|s)$, we henceforth refer to f as the skill prior.

We refer the reader to Dinh et al. [23] for more information about real NVP. We train our state-conditioned skill prior jointly with our VAE embedding module. Figure 3 illustrates the multi-modality present in the latent skill-space and how our skills prior captures this. Implementation details are provided in Appendix A.0.1.

4.3 Reinforcement Learning in a State-Conditioned Skill Space

Once trained, the decoder and skill prior weights are frozen and incorporated within the RL framework. Our high-level RL policy denoted $\pi_{\text{HL}}(g|s)$ is a neural network which maps a state to a vector $g \in G$ in the domain of the skill prior transformation f^{-1} defined in Section 4.2.2. The skill prior f^{-1} is then used to transform g to latent skill embedding z. Exploration for the high level policy therefore involves sampling $g \sim \mathcal{N}(0, I)$ during training. Our use of normalising flows for our skills prior is motivated by [4], who hypothesized that the bijective nature of f allows the RL agent to retain full control over the broader skill space $\mathcal{Z}$: for any given z, there exists a g that generates $z = f^{-1}(g; s)$ in the original MDP. This means that all skills from $\mathcal{Z}$ can be sampled by the skill prior with non-zero probability.

Once a latent skill z is attained by the high-level agent, the closed-loop decoder can reconstruct each action a' sequentially conditioned on the current state for the skill horizon H. To increase the diversity of downstream tasks that can be learned using the fixed skill space, we introduce a low-level residual policy π_δ that can adapt the decoded skills at the executable action level, providing the hierarchical agent with full control over the MDP. We condition this residual policy on the current state s, selected skill z, and the action proposal generated by the skill decoder a'. The produced action $\delta a \sim \pi_\delta(\delta a|s, z, a')$ is added to the decoded action a' before being executed on the robot. This low-level system is operated at a frequency of H Hz based on the skill horizon, while the outer loop operates at 1 Hz. Figure 1 summarises our Residual Skill Policy (ReSkill) architecture. Given the non-stationarity imposed by the low-level residual policy being trained jointly with the high-level policy, we utilise an on-policy RL algorithm for training the hierarchical agent. Implementation details are provided in Appendix A.0.2.

5 Experiments

We evaluate our approach in a robotic manipulation domain simulated in MuJoCO [26], involving a 7-DoF Fetch robotic arm interacting with blocks placed in front of it as shown in Figure 4. Evaluation is performed over three tasks, each one not solvable purely with the prior controllers described in Section 4.1. The downstream tasks are:

Slippery Push The agent is required to push a block to a given goal, however we lowered the friction of the table surface from that seen in the training push task.

Table Cleanup We introduce a rigid tray object in the scene, into which the agent must place the given block. The tray is not observed by the agent and the edges act as an obstacle.

Pyramid Stack Stacking task where the agent is rewarded for placing a smaller block onto a larger block. The prior controllers are unable to move the gripper above the height of the larger block.

We compare our approach against several state-of-the-art RL methods; these methods either operate directly on the atomic action space or train agents using a skills-based approach. The specific methods we compared against are as follows

1. **Scripted Controller:** The average performance of the handcrafted controllers used for data collection across the 3 environments.
2. **Behavioural Cloning (BC) + Fine-Tuning** Learns a policy by first using supervised learning on state-action pairs from the training set, followed by RL fine-tuning.
3. **SAC:** Trains an agent using Soft Actor Critic [27] with random weight initialization
4. **PPO:** Trains an agent using Proximal Policy Optimisation [28] with random weight initialization
5. **PaRRot:** Leverages flow-based density estimation to map the agent's action space to atomic actions that are likely to be useful in a given state [4].
6. **SPiRL:** A skill-based approach utilising KL regularisation of the RL policy towards a Gaussian skill prior [3]
7. **ReSkill (No Skill Prior:)** Ablation study where the high-level policy samples from $\mathcal{Z}$ directly with no skill prior.
8. **ReSkill (No Residual:)** Ablation study where we remove the low-level residual policy for adapting skills.

6 Results

The results across each task are summarised in Figure 5. Our approach, ReSkill, outperforms all the baselines, attaining the highest performing policy given its ability to adapt to task variations. ReSkill additionally converges with superior sample efficiency given the state-conditioned skill-prior. The baseline scripted controllers are sub-optimal across all three tasks, while training SAC and PPO from scratch fail to learn, highlighting the need for prior experience. Results are incrementally better using BC initialisation of policy weights; however, the agent still attains sub-optimal performance with high variance across the trials. A critical difference between PaRRot and ReSkill is using skills instead of atomic, single-step actions. Across all tasks, ReSkill significantly outperforms PaRRot, which we can attribute to the temporal abstraction of skills that accelerate search space coverage in sparse reward and long-horizon tasks. Further, we note that PaRRot struggles to learn in the *Slippery-Push* domain which we could attribute to its inability to deal with the variations in the downstream task. SPiRL accelerates learning, yielding stronger performance than atomic action methods on *Table-Cleanup* and comparable performance on *Slippery-Push*. However, it still fails to attain high performance as it only composes high-level skills without the ability to adapt them to task variations, as shown by ReSkill. To better understand the effectiveness of our sampling-based skill-prior over that used by SPiRL, we ablate ReSkill by removing the residual policy as shown in Figure 6. ReSkill (no-residual) outperforms SPiRL in sample efficiency across two tasks, with SPiRL unable to learn in the stacking task. We additionally ablate the residual and note that it plays a crucial role in fine-grained control for attaining higher rewards. This allows us to appropriately balance the trade-off between generality and sample-efficiency when leveraging prior knowledge in RL.

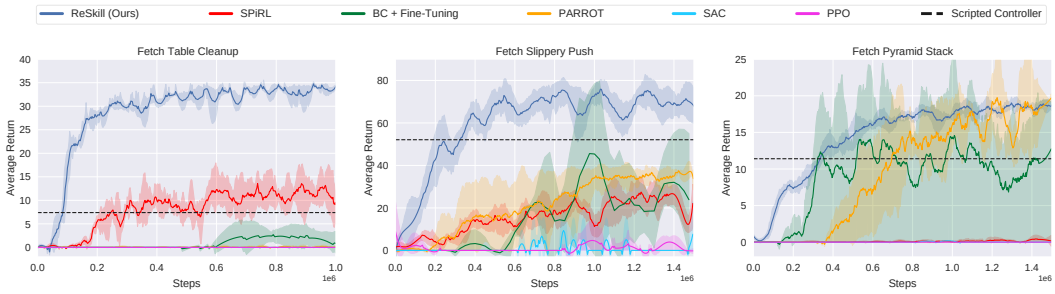


Figure 5: **Training Performance.** Average training performance across 3 random seeds for the different tasks. ReSkill outperforms all the baselines in both sample efficiency and convergence to the highest performing final policy.

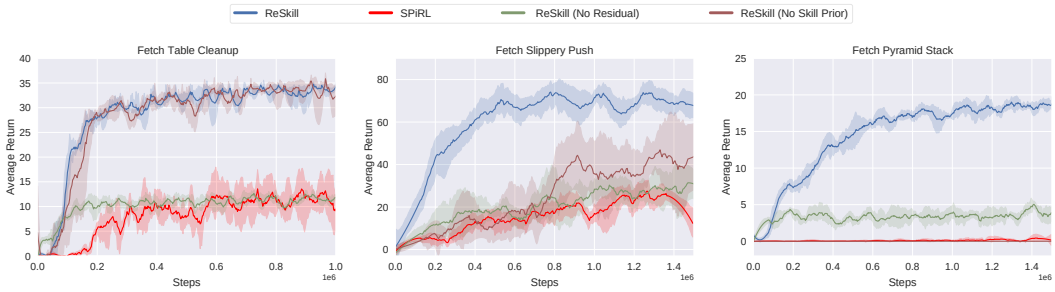


Figure 6: **Ablation Study.** Analysing the impact of the residual and skill prior on the average training performance across the 3 tasks. The skill prior plays an important role accelerating learning with the more difficult tasks, while the residual is important for attaining higher rewards by adapting the skills to the variations in the training environment.

6.1 Limitations

While we demonstrate significantly faster learning and better asymptotic performance than all the baseline methods, ReSkill still requires thousands of environment samples before converging to a high success rate. This limited our ability to demonstrate our method on a physical robot. An interesting direction to explore, however, would be to test the approach's applicability in the sim-to-real setting, where we train the skill-space on demonstration data collected in the real world and downstream RL agents are trained in simulation using this real-world skill-space. In addition, the non-stationarity of the MDP imposed by the low-level residual limits us to on-policy methods. Future works could explore the applicability of off-policy algorithms for further improvements to sample efficiency. Finally, learning the state-conditioned skill space requires two modules, specifically the VAE embedding module and flows-based skills prior module. Future work will explore using a single generative model such as a conditional VAE [29] to jointly handle tasks of both modules.

7 Conclusion

This work proposes Residual Skill Policies (ReSkill), a general-purpose skill-based RL framework that enables effective skill re-use for adaptable and efficient learning. We learn a high-level skill agent within a state-conditioned skill space for accelerated skill composition, as well as a low-level residual agent that allows for fine-grained skill adaptation to variations in the task and training environment. Additionally, we show that our framework allows us to re-purpose a handful of scripted controllers to solve a wide range of unseen tasks. We evaluate ReSkill across a range of manipulation tasks and demonstrate its ability to learn faster and achieve substantially higher final returns across all task variations than the baseline methods, succeeding where RL from scratch fails to learn.

References

[1] J. Merel, L. Hasenclever, A. Galashov, A. Ahuja, V. Pham, G. Wayne, Y. W. Teh, and N. M. O. Heess. Neural probabilistic motor primitives for humanoid control. *ArXiv*, abs/1811.11711, 2019.

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

[3] K. Pertsch, Y. Lee, and J. J. Lim. Accelerating reinforcement learning with learned skill priors. *arXiv preprint arXiv:2010.11944*, 2020.

[4] A. Singh, H. Liu, G. Zhou, A. Yu, N. Rhinehart, and S. Levine. Parrot: Data-driven behavioral priors for reinforcement learning. *ArXiv*, abs/2011.10024, 2021.

[5] K. Hausman, J. T. Springenberg, Z. Wang, N. Heess, and M. Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

[6] J. Merel, S. Tunyasuvunakool, A. Ahuja, Y. Tassa, L. Hasenclever, V. Pham, T. Erez, G. Wayne, and N. M. O. Heess. Catch & carry: Reusable neural controllers for vision-guided whole-body tasks. *arXiv: Artificial Intelligence*, 2019.

[7] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine. Residual reinforcement learning for robot control. *arXiv preprint arXiv:1812.03201*, 2018.

[8] K. Rana, V. Dasagi, B. Talbot, M. Milford, and N. Sünderhauf. Multiplicative controller fusion: Leveraging algorithmic priors for sample-efficient reinforcement learning and safe sim-to-real transfer. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6069–6076, 2020. doi:10.1109/IROS45743.2020.9341372.

[9] I. Uchendu, T. Xiao, Y. Lu, B. Zhu, M. Yan, J. Simon, M. Bennice, C. Fu, C. Ma, J. Jiao, et al. Jump-start reinforcement learning. *arXiv preprint arXiv:2204.02372*, 2022.

[10] L. Xie, S. Wang, S. Rosa, A. Markham, and N. Trigoni. Learning with training wheels: speeding up training with a simple controller for deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6276–6283. IEEE, 2018.

[11] 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.

[12] K. Rana, V. Dasagi, J. Haviland, B. Talbot, M. Milford, and N. Sünderhauf. Bayesian controller fusion: Leveraging control priors in deep reinforcement learning for robotics. *arXiv preprint arXiv:2107.09822*, 2021.

[13] R. S. Sutton, D. Precup, and S. Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[14] L. Steccanella, S. Totaro, D. Allonsius, and A. Jonsson. Hierarchical reinforcement learning for efficient exploration and transfer. *arXiv preprint arXiv:2011.06335*, 2020.

[15] A. G. Kupcsik, M. P. Deisenroth, J. Peters, and G. Neumann. Data-efficient generalization of robot skills with contextual policy search. In *AAAI*, 2013.

[16] N. M. O. Heess, G. Wayne, Y. Tassa, T. P. Lillicrap, M. A. Riedmiller, and D. Silver. Learning and transfer of modulated locomotor controllers. *ArXiv*, abs/1610.05182, 2016.

[17] T. Haarnoja, K. Hartikainen, P. Abbeel, and S. Levine. Latent space policies for hierarchical reinforcement learning. In *ICML*, 2018.

[18] T. Shankar, S. Tulsiani, L. Pinto, and A. Gupta. Discovering motor programs by recomposing demonstrations. In *International Conference on Learning Representations*, 2019.

[19] S. Krishnan, R. Fox, I. Stoica, and K. Goldberg. Ddco: Discovery of deep continuous options for robot learning from demonstrations. In *CoRL*, 2017.

[20] J. Gehring, G. Synnaeve, A. Krause, and N. Usunier. Hierarchical skills for efficient exploration. *Advances in Neural Information Processing Systems*, 34:11553–11564, 2021.

[21] T. Pearce, M. Zaki, A. Brintrup, and A. Neel. Uncertainty in neural networks: Bayesian ensembling. *arXiv preprint arXiv:1810.05546*, 2018.

[22] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[23] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[24] E. G. Tabak and C. V. Turner. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013.

[25] L. Ardizzone, J. Kruse, S. Wirkert, D. Rahner, E. W. Pellegrini, R. S. Klessen, L. Maier-Hein, C. Rother, and U. Köthe. Analyzing inverse problems with invertible neural networks. *arXiv preprint arXiv:1808.04730*, 2018.

[26] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[27] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint 1801.01290*, 2018.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*, Aug. 2017. URL http://arxiv.org/abs/1707.06347. arXiv: 1707.06347.

[29] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

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

[31] J. Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

[32] K. Perlin. An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3):287–296, 1985.

A Implementation Details

A.0.1 Skill Embedding and Skill Prior Learning

In our work, the conditioning information consists of a 19-dimensional state vector which describes the configuration of the robot and the pose of objects in the environment. We parameterise both the VAE skill embedding and normalising flows skill prior modules as deep neural networks.

VAE embedding module The VAE encoder first processes individual concatenated state-action pairs in sequence using a one-layer LSTM with 128 hidden units. The hidden layer after processing the final observation is then fed into an MLP block comprised of 3 linear layers with batch normalisation and ReLU activation units with two output heads over the last layer, yielding parameters μ_z, σ_z of the variational posterior $\log q_\phi(z|\mathbf{s}, \boldsymbol{a}) \sim \mathcal{N}(\mu_z, \sigma_z)$. We set our latent space to be 4-dimensional, i.e. $\mathcal{Z} \cong \mathbb{R}^4$. The decoder network mirrors the architecture as the MLP block used in the encoder, taking as input concatenated latent skill vector z and current state s_t. The final layer has a single head and a tanh layer to ensure actions are bounded between $(-1, 1)$. The full action sequence ***a'*** is decoded in sequence with observed states given to the encoder $\mathbf{s}$. The overall loss function for the embedding module is as provided in (1), noting the expectation is computed by drawing a single sample from posterior $q_\phi(z|\mathbf{s}, \boldsymbol{a})$. Furthermore, the VAE reconstruction loss term is the mean-squared error, i.e. $\log p_\theta\left(a_t \mid z, s_t\right) \propto (a_t - a_t')^2$.

State-conditioned skills prior sampling module The network parameterising the skill prior $f : \mathcal{Z} \times \mathcal{S} \rightarrow G$ is a conditional real NVP [23] which consists of four affine coupling layers, where each coupling layer takes as input the output of the previous coupling layer, and the robot state vector s_0. We use a standard Gaussian $p_\mathcal{G}(g) \sim \mathcal{N}(0, I)$ as our base distribution for our generative model. Our architecture for f is identical to the conditional Real NVP network used in Singh et al. [4], see Singh et al. [4] for an in-depth explanation. The loss function for a single example is given by

$$\mathcal{L}_{prior} = \log p_\mathcal{G}(f(z, s_0)) + \log \left| \det \frac{\partial f}{\partial z^\top} \right|. \tag{2}$$

During training we pass the initial state vector of the robot before applying the skill as conditioning information. We optimise our model using the Adam optimiser with a learning rate of 1e-4 and batch size of 128. The overall objective for training the skills model for a single training observation is given as

$$\mathcal{L}_{skills} = \mathcal{L}_{embed} + \mathcal{L}_{prior}, \tag{3}$$

noting that gradients of the skills prior loss w.r.t. z, i.e. $\partial \mathcal{L}_{prior} / \partial z$ are blocked. What this means is that the VAE embedding module is trained without being influenced by the skills prior objective, however the skills prior training is affected because the topology of latent space determined by the embedding module is evolving during training. We found this joint training yielded more effective skill priors compared to first training on $\mathcal{L}_{embed}$ and subsequently training the skills prior on $\mathcal{L}_{prior}$ after the embedding module has finished training.

A.0.2 Reinforcement Learning Setup

We utilise Proximal Policy Optimisation [30] as the underlying RL algorithm for both the high and low-level policies, using the standard hyper-parameters given in the SpinningUp implementation [31] with a clip ratio of 0.2, policy learning rate of 0.0003 and discount factor γ = 0.99. We found it important to train the high-level policy without the residual for an initial 20k steps before allowing the residual policy to modify the underlying skills. This allowed the high-level policy to experience the useful skills suggested by the skill-prior without being distorted by the random initial outputs of the residual policy.

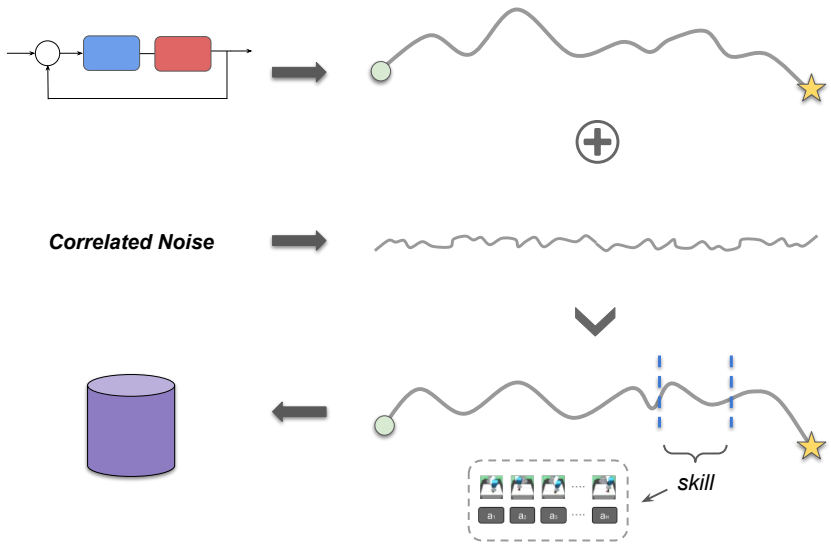


Figure 7: **Data collection.** Skill extraction from trajectories collected from a handcrafted controller. To obtain a diverse range of skills we add correlated Perlin noise to the trajectory rollouts before skill extraction.

B Data Collection

In order to re-purpose existing controllers for a wide range of tasks, we decompose their behaviours into task-agnostic skills. We firstly collect a dataset of demonstrations, consisting of state-action pairs, by randomly sampling a controller and executing it in the environment. In this work, we script 2 simple controllers for a 7-DoF robotic arm each capable of completing *pushing* and *pick and place* manipulation tasks on an empty table and are detailed below. To increase the diversity of skills collected, we add Perlin noise [32] to the controller outputs. As Perlin noise is correlated across the trajectory, it allows for smooth deviations in trajectory space. Before adding these trajectories to the dataset, we filter them based on a predefined rule: if it is longer than the skill horizon H we add it to our dataset. Skills can be extracted from a trajectory, by randomly slicing a H dimensional skill consisting of a sequence of actions $\boldsymbol{a} = \{a_t, ..., a_{t+H-1}\}$ and the corresponding states $\boldsymbol{s} = \{s_t, ..., s_{t+H-1}\}$ that these actions were executed in. Figure 7 shows a summary of the data collection process and the notion of a skill from a recorded trajectory.

B.1 Handcrafted Controllers

We describe each of the controllers derived to complete simple tasks on an empty table with a single object place in front of it as shown in Figure 4 (a).

B.1.1 Reactive Push Controller

This controller is designed for pushing an object to a target location. Although this policy performs well in the original Push task, its performance drops dramatically when the sliding friction on the block is reduced as in the *SlipperyPush* task.

Algorithm 1 Reactive Push

Given: threshold
Input: state
Output: action

if distance(targetObjectPose, objectPose) $<$ threshold **then**
  action $\leftarrow$ 0
else if gripper_at_pushloc(gripperPose, pushLoc) **then**
  action $\leftarrow$ push object to target
else if distance(gripperPose, pushLoc) $>$ threshold **then**
  action $\leftarrow$ move gripper to push location
end if
return *action*

B.1.2 Pick and Place Controller

This controller is designed to move to an object location, grasp the object and move it towards a target goal. The controller does not lift the block higher than 3cm above the surface of the table and therefore will fail if either the object has to be placed on a higher surface or alternatively, the target is on the other side of a barrier. These failure cases are present in both the *Pyramid-Stack* and *Table-Cleanup* tasks.

Algorithm 2 Pick and Place

Given: threshold
Input: state
Output: action

if distance(targetObjectPose, objectPose) $<$ threshold **then**
  action $\leftarrow$ 0
else if is_grasped(gripperPose, objectPose) **then**
  action $\leftarrow$ move to target
else if object_in_gripper(gripperPose, objectPose) **then**
  action $\leftarrow$ close gripper
else if gripper_above_object(gripperPose, objectPose) **then**
  action $\leftarrow$ move gripper down
else if distance(gripperPose, objectPose) $>$ threshold **then**
  action $\leftarrow$ move to object
end if
return *action*