Policy Stitching: Learning Transferable Robot Policies

Anonymous Author(s)
Affiliation
email

Abstract: Training robots with reinforcement learning (RL) typically involves heavy interactions with the environment, and the acquired skills are often sensitive to changes in task environments and robot kinematics. Transfer RL aims to leverage previous knowledge to accelerate learning of new tasks or new body configurations. However, existing methods struggle to generalize to novel robot-task combinations and scale to realistic tasks due to complex architecture design or strong regularization that limits the capacity of the learned policy. We propose Policy Stitching, a novel framework that facilitates robot transfer learning for novel combinations of robots and tasks. Our key idea is to apply modular policy design and align the latent representations between the modular interfaces. Our method allows direct stitching of the robot and task modules trained separately to form a new policy for fast adaptation. Our simulated and real-world experiments on various 3D manipulation tasks demonstrate the superior zero-shot and few-shot transfer learning performances of our method.

Keywords: robot transfer learning, policy stitching

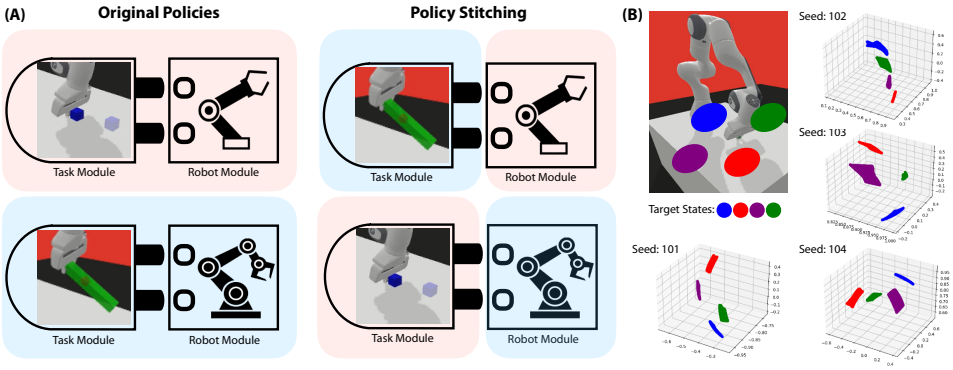


Fig. 1: **Policy Stitching.** (A) Our framework facilitates robot transfer learning among novel combinations of robots and tasks by decoupling and stitching robot and task modules. (B) Motivation example: A robot arm is trained to reach goals in four different target regions using the modular policy. Results from separate training runs with different random seeds (101-104) show misaligned latent representations.

1 Introduction

Robots are typically trained to excel at specific tasks such as relocating objects or navigating to predetermined locations. However, such robots need to be retrained from scratch when faced with new tasks or body changes. In contrast, humans demonstrate remarkable capabilities [1] to continuously acquire new skills by drawing on past experiences. Even under physical constraints imposed by injuries, we can still rapidly adapt to perform new tasks. Despite significant advancements in robotics and machine learning, robots still cannot generalize their experience across a wide range of tasks and body configurations.

Model-based robot learning, in particular self-modeling or self-identification learning [2, 3, 4, 5, 6, 7, 8, 9, 10], aims to learn a predictive model of the robot's kinematics and dynamics and then

Submitted to the 7th Conference on Robot Learning (CoRL 2023). Do not distribute.

employ this model for various downstream tasks through model predictive control. However, the learning process needs to be separated into two stages of robot-specific and task-specific learning. On the other hand, model-free reinforcement learning trains policies end to end, but it often has limited transfer learning performance [11]. Existing efforts regularize one large policy network for multi-task learning by learning routing connections to reuse part of the network weights [12] or assigning task-specific sub-networks [13]. However, the capacity of the large policy network grows exponentially as the number of tasks increases.

We introduce **Policy Stitching (PS)**, a model-free learning framework for knowledge transfer among novel robot and task combinations through modular policy design and transferable representation learning (Fig.1 (A)). We explicitly decouple robot-specific representation (e.g., robot kinematics and dynamics) and task-specific representation (e.g., object states) in our policy design to enable the reassembly of both modules for new robot-task combinations. For instance, given one policy trained for a 3-DoF manipulator to pick up a cube and another policy trained for a 5-DoF manipulator to pick up a stick, if we would like to have the 3-DoF manipulator pick up a stick now (i.e., +), our method can directly take the robot module in the first policy and stitch it with the task module in the second policy.

While the modular design allows direct stitching, the reassembled policy may not work at all. We find that this is because the output representation from one neural network module does not align with the desired input representation of another module, particularly when modules are trained on different tasks and robot body configurations. Past work in supervised learning [14, 15] has made similar observations. As a motivating example, in Fig.1(B), we show that, even under the same task and robot setup, latent representations from RL policies trained with different random seeds do not align with each other. More interestingly, they exhibit similar isometric transformations as shown in recent work on supervised learning [14, 15]. See Appendix B.1 for additional details of the isometric transformation phenomenon.

To this end, we further propose to generalize the latent representation alignment techniques from supervised learning to reinforcement learning for robot transfer learning. The key idea is to enforce transformation invariances by projecting the intermediate representations into the same latent coordinate system. Unlike supervised learning, RL does not come with human labels to help select anchor coordinates. We propose to use unsupervised clustering of target states to help resolve this gap. Our method produces aligned representations for effective policy stitching. In summary, our contributions are three-fold:

- **Policy Stitching**, a model-free reinforcement learning framework for robot transfer learning among novel robot and task combinations.
- **Modular Policy Design** for robot-specific and task-specific module reassembly. **Representation Alignment** to learn transferable latent space for direct module stitching.
- Demonstration of the clear advantages of our method in both zero-shot and few-shot transfer learning through **simulated and physical 3D manipulation tasks**.

2 Related Work

Robot Transfer Learning. Transferring robot policies across various environment dynamics or novel tasks is still an open challenge [16, 17, 18, 19]. Past efforts have proposed to transfer different components in reinforcement learning framework, such as value functions [20, 21, 22], rewards [23], experience samples [24], policies [25, 26, 27], parameters [28, 29, 30], and features [31]. In contrast, our method transfers structured network modules of a policy network across novel tasks or novel robot body configurations. Our modular design is similar to previous work [27], but we do not limit the capacity of the latent space to avoid overfitting. Instead, we enforce invariances among the learned latent representations of different modules through feature alignment. Moreover, our work surpasses previous accomplishments in 2D tasks by demonstrating 3D manipulation skills, both in simulation and the real world.

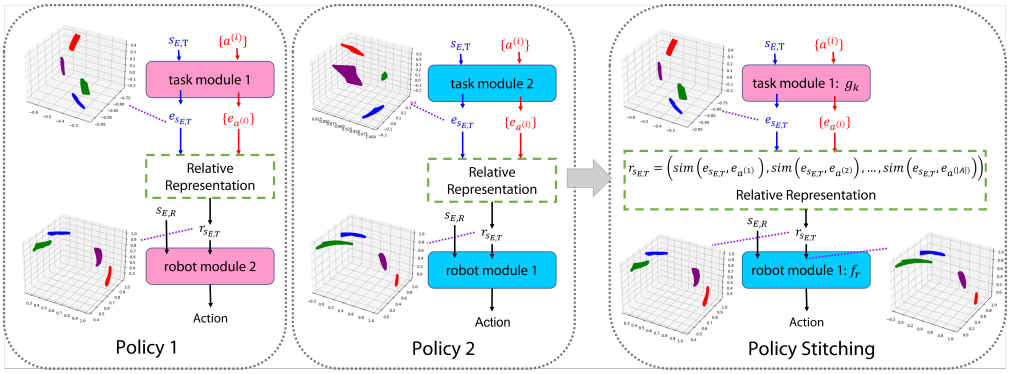


Fig. 2: **Method Overview.** Our modular policy design enables the task module to process task-specific states such as object states and environmental conditions and the robot module to process robot-specific states such as kinematics and dynamics. We generalize relative representations to model-free RL setup to align latent representations from separately trained policy networks. With both modular design and latent representation alignment, our method allows direct stitching of unseen combination of task and robot module for effective and efficient transfer.

Meta Learning. Meta-learning [32, 33, 34, 35, 36, 37] aims to achieve fast adaptation on a new task based on past learning experiences. Recent research has proposed generating better parameter initialization for learning new tasks [30, 38, 39, 40] or using memory-augmented neural networks [41, 42, 43] to assimilate new data swiftly without forgetting the previous knowledge. Another category of methods [44, 45, 46, 47, 48, 49] proposes to use a hypernetwork to generate the parameters of policy networks for different tasks. Our work also aims at fast adaptation to novel robot and task combinations, but our method reuses structured components in policy networks.

Compositional Reinforcement Learning. Functional compositional RL has been used in zero-shot transfer learning [27], multi-task learning [12], and lifelong learning [13]. Some approaches learn the structure of the modules [50, 51, 12, 13], but these trained modules can only be functional in a large modular system and cannot work when stitched with new modules. Another method [27] tries to make the network module reusable, but the module alignment problem prevents it from working on 3D tasks. Our work can directly stitch policy modules to improve zero-shot and few-shot learning performances in simulated and physical 3D manipulation tasks.

3 Method: Policy Stitching

Policy Stitching focuses on enabling effective knowledge transfer between different tasks or robot bodies. Our method consists of two main components: the modular policy design and transferable representations achieved through latent space alignment. Our framework is designed to be compatible with various model-free RL algorithms.

3.1 Modular Policy Design

We propose modular policy design to decompose the policy into two distinct modules focusing on robot specific and task specific information. Our design provides a straightforward ingredient for module reuse and stitching.

Consider an environment, E, with a robot r and a presented task k, we formulate the problem as a Markov Decision Process with an observed state s_E and action a_E. We denote the learning policy as $\pi_{E_{rk}}(a_E \mid s_E)$ which is parameterized by a function $\phi_{E_{rk}}(s_E)$. We utilize the model-free RL formulation in our experiments, specifically the Soft Actor-Critic (SAC) [52] algorithm.

We decompose the state s_E, the policy function $\phi_{E_{rk}}(s_E)$, and the Q-function into a **robot-specific** module and a **task-specific** module (Fig. 2). The state s_E is decomposed into a robot-specific state $s_{E,\mathcal{R}}$ and a task-specific state $s_{E,\mathcal{T}}$. The robot-specific state $s_{E,\mathcal{R}}$ only consists of the joint angles of the robot, while the task-specific state $s_{E,\mathcal{T}}$ includes task information at hand such as the current

position and orientation, linear and angular velocity, and goal position of the object. Similarly, the policy function $\phi_{E_{rk}}(s_E)$ is also decomposed into a task-specific module g_k to encode the input task states, and a robot-specific module f_r to implicitly capture robot kinematics and dynamics from input robot states. We follow the same design to decompose the Q-function into task-specific module q_k and robot-specific module h_r. Formally, our policy function and Q-function can be re-written as follows:

$$\phi_{E_{rk}}\left(s_E\right) = \phi_{E_{rk}}\left(s_{E,\mathcal{T}}, s_{E,\mathcal{R}}\right) = f_r\left(g_k\left(s_{E,\mathcal{T}}\right), s_{E,\mathcal{R}}\right). \tag{1}$$

$$Q_{E_{rk}}(s_E, a_E) = Q_{E_{rk}}\left(s_{E,\mathcal{T}}, s_{E,\mathcal{R}}, a_E\right) = h_r\left(q_k\left(s_{E,\mathcal{T}}\right), s_{E,\mathcal{R}}, a_E\right), \tag{2}$$

We name the modules after their main functionalities. Such modular design does not completely separate the information processing from each module. Since we train the entire policy end to end, the gradients flow in between the two modules during backpropagation. However, by explicitly enforcing their main functionalities through our modular design, we can still obtain effective improvements for robot transfer learning by reusing and stitching modules to leverage previously acquired knowledge. After the stitching, we can either use the new policy for zero-shot execution or perform few-shot learning for fast adaptation.

3.2 Transferable Representations through Latent Space Alignment

Our modular policy design allows for direct stitching, but a simple stitching approach may not yield optimal performance. Previous research [27] attributes the issue as overfitting to a particular robot and task, and has proposed to use dropout and very small bottleneck dimensions. However, the capacity of the policy is largely limited to simple 2D tasks and low-dimensional task states. Instead, we identify the fundamental issue as the lack of alignment enforcement of the latent embedding space at the output/input interface of two modules. Even under the same robot-task combination, the latent embedding vectors between the robot and task modules from different training seeds exhibit an almost isometric transformation relationship (Fig.1(B)). While similar observation has been made in supervised learning [15, 53], we have identified and addressed this issue in the context of RL and demonstrate the effectiveness in knowledge transfer across novel robot-task combinations.

We propose to generalize Relative Representations [15] in supervised learning to model-free RL. Unlike other techniques in supervised learning [54, 14, 55], Relative Representations does not introduce additional learnable parameters. The key idea is to project all latent representations at the interface between two modules to a shared coordinate system among multiple policy networks. Through this invariance enforcement of transformation, we can establish a consistent latent representation of the task-state that aligns with the desired latent input of the robot module.

However, the original Relative Representations relies on the ground-truth labels in the supervised dataset to provide the anchor points for building the latent coordinate system. Our setup in RL does not come with such labels. The anchor points are analogous to the concept of basis in a linear system, hence should be as dissimilar from each other as possible. To overcome this challenge, we select an anchor set $\mathbb{A}$ from the training set $\mathbb{S}$ by leveraging the unsupervised clustering algorithm, k-means [56], to identify the representative anchor points from the sampled task states. We select k anchor states that are closest to the centroids of the clusters. The number of clusters k depends on the dimension of the latent representation.

We want to represent every embedded task state $\boldsymbol{e}_{\boldsymbol{s}^{(i)}} = g_k\left(\boldsymbol{s}^{(i)}\right)$ with respect to the embedded anchor states $\boldsymbol{e}_{\boldsymbol{a}^{(j)}} = g_k\left(\boldsymbol{a}^{(i)}\right)$, where $\boldsymbol{a}^{(j)} \in \mathbb{A}$ and g_k is the task-specific module that embeds the task states. To this end, we capture the relationship between an embedded task state $\boldsymbol{e}_{\boldsymbol{s}^{(i)}}$ and an embedded anchor state $\boldsymbol{e}_{\boldsymbol{a}^{(j)}}$ using a similarity function $sim : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, which calculates a similarity score $r = \mathrm{sim}\left(\boldsymbol{e}_{\boldsymbol{s}^{(i)}}, \boldsymbol{e}_{\boldsymbol{a}^{(j)}}\right)$. Given the anchor set $a^{(1)}, \ldots, a^{(|\mathbb{A}|)}$, the transferable representation of an input task state $\boldsymbol{s}^{(i)} \in \mathbb{S}$ is calculated by:

$$\boldsymbol{r}_{\boldsymbol{s}^{(i)}} = \left(\mathrm{sim}\left(\boldsymbol{e}_{\boldsymbol{s}^{(i)}}, \boldsymbol{e}_{\boldsymbol{a}^{(1)}}\right), \mathrm{sim}\left(\boldsymbol{e}_{\boldsymbol{s}^{(i)}}, \boldsymbol{e}_{\boldsymbol{a}^{(2)}}\right), \ldots, \mathrm{sim}\left(\boldsymbol{e}_{\boldsymbol{s}^{(i)}}, \boldsymbol{e}_{\boldsymbol{a}^{(|A|)}}\right)\right), \tag{3}$$

which is a vector of length $|\mathbb{A}|$, and each element is the similarity score between the embedded task state and an embedded anchor state. We intentionally choose the cosine similarity as our similarity

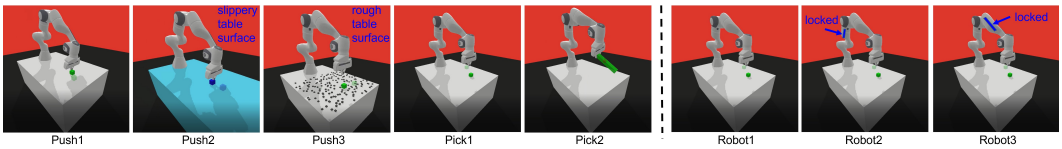


Fig. 3: **Simulation Experimental Setup**: From left to right, we show different task and robot configurations: standard table, slippery table, rough terrain with small rocks, cube object, stick object, 7-DoF Franka Emika Panda arm, third joint locked, and fifth joint locked.

measure because it is invariant to isometric transformations. Although it is not invariant to vector translation, we add a normalization layer before calculating cosine similarity to mitigate this issue.

3.3 Implementation Details

We challenge our methods in sparse reward setup and use hindsight experience replay [57] to encourage exploration. Both the task and robot modules are represented as MLPs [58] with four layers and 256 hidden dimensions. The last layer of the task module and the first layer of the robot module have a dimension of 128, thus the dimension of the latent representation and the selected anchor set are also 128 ($k = 128$). Detailed network structures of our method and other baselines can be found in Appendix A. Each of our policy is trained on 7 threads of the AMD EPYC 7513 CPU and 1 NVIDIA GeForce RTX 3090 GPU. Each training epoch consists of $35,000$ steps.

4 Experiments

We aim to evaluate the performance of Policy Stitching (PS) in both zero-shot and few-shot transfer setups. Furthermore, we generalize our experiments to a physical robot setup to demonstrate the practical applicability and real-world performance. Finally, we provide quantitative and qualitative analysis to understand the role of the learned transferable representations in effective policy stitching.

4.1 Simulation Experiment Setup and Baselines

We modify the panda-gym environment [59] to simulate 5 manipulation task environments and 3 distinct robots with varying kinematics (Fig.3). The tasks consist of 3 push scenarios, where the robot must push a cube to the goal position on surfaces with very different friction properties, and 2 pick tasks that involve pick and place task of objects with different shapes. In zero-shot evaluation, the stitched policy is directly tested without fine-tuning and each experiment consists of 200 test trajectories, which are repeated 5 times to report the mean and standard deviation. In few-shot evaluation, the stitched policy first interacts with the environment for additional 10 epochs to store data in the replay buffer and is then fine-tuned for a few epochs with SAC. We train with 3 random seeds to report the mean and standard deviation.

Robot-Task Combination. We evaluate PS on 8 robot-task combinations to cover policy stitching in both similar and dissimilar setup. We list all combinations as the titles of sub-figures in Fig.4. For instance, "t: Pi1-R1" represents a task module trained in the Pick1 task with Robot1, while "r: Pu1-R2" represents a robot module trained in the Push1 task with Robot2. These modules are then stitched together to form a new combination that has not been jointly trained together.

Baselines. We compare our method to the approach of Devin et al. [27], which uses a similar modular policy but with small bottleneck and dropout as regularization to alleviate the modules misalignment issue. In the few-shot test, we also include an additional comparison to increase the bottleneck dimension of the Devin et al. [27]'s method to the same dimension as our method, since the original bottleneck dimension is too small to capture necessary information for complex 3D tasks. Furthermore, we provide an ablation study with our modular policy design but no latent representation alignment to study its importance. We also build a Plain baseline which is an MLP that takes in all the states at the first layer and has no modular structure. When transferring to novel task-robot combination, we split the Plain MLP network into two parts (i.e.,top-half and bottom-half)

	Touching Rate (%)				*Success Rate (%)*			
	PS	***PS(Ablation)***	***Devin et al.***	***Plain***	***PS***	***PS(Ablation)***	***Devin et al.***	***Plain***
E1	73.0 ± 1.4	$\mathbf{75.8 \pm 4.4}$	44.8 ± 3.1	0.0 ± 0.0	$\mathbf{26.9 \pm 3.6}$	13.9 ± 3.5	11.7 ± 2.6	6.5 ± 2.5
E2	$\mathbf{99.9 \pm 0.2}$	93.5 ± 2.0	78.3 ± 5.4	17.9 ± 2.7	$\mathbf{24.4 \pm 1.5}$	6.8 ± 0.7	10.0 ± 0.9	9.3 ± 2.1
E3	90.8 ± 2.3	$\mathbf{95.6 \pm 0.8}$	82.6 ± 1.6	0.0 ± 0.0	$\mathbf{16.5 \pm 1.9}$	7.2 ± 1.5	9.8 ± 2.5	8.4 ± 3.7
E4	$\mathbf{95.4 \pm 1.9}$	88.7 ± 1.1	49.6 ± 1.1	1.6 ± 1.0	13.2 ± 1.1	$\mathbf{14.9 \pm 1.0}$	11.8 ± 2.4	9.0 ± 1.4
E5	$\mathbf{14.7 \pm 2.7}$	9.6 ± 1.6	8.6 ± 0.3	2.8 ± 0.9	2.1 ± 0.4	$\mathbf{4.0 \pm 1.9}$	2.9 ± 1.0	3.8 ± 1.6
E6	55.6 ± 2.4	$\mathbf{80.8 \pm 3.4}$	30.5 ± 4.2	0.1 ± 0.2	8.8 ± 1.9	$\mathbf{11.6 \pm 2.0}$	10.5 ± 2.1	9.0 ± 0.7
E7	$\mathbf{41.8 \pm 1.7}$	13.1 ± 2.9	9.8 ± 1.3	0.0 ± 0.0	$\mathbf{4.7 \pm 1.6}$	3.6 ± 0.8	3.0 ± 1.6	3.2 ± 1.4
E8	18.4 ± 1.3	$\mathbf{18.8 \pm 2.6}$	13.6 ± 1.1	0.0 ± 0.0	$\mathbf{5.3 \pm 1.7}$	3.0 ± 0.7	3.1 ± 0.5	3.0 ± 0.7

Tab. 1: **Success rates and touching rates of zero-shot transfer**

and perform the same reassembling operation as the modular networks. This operation that preserves a sub-network from the old task and adds a new sub-network for the new task has been widely used in other works [60, 61, 62, 63, 64]. See Appendix A for architecture details.

Metrics. We use success rate (task completion within a fixed number of steps) and touching rate (contact with the object during the task) as our evaluation metrics. In challenging transfer scenarios (e.g., a robot trained for a pushing task is required to perform a picking task), all methods may exhibit low success rates. Therefore, we use touching rate as a more informative metric to indicate whether the robot is engaging in meaningful interactions rather than arbitrary movements. Effective touching behavior can result in improved exploration for few-shot transfer learning, since touching serves as a preliminary behavior for picking or pushing.

4.1.1 Results: Zero-Shot Transfer in Simulation

E1-E3 are easier robot-task combinations than E4-E8 since the unseen setup is closer to their original combinations. As shown in Tab.1, for E1-E3, PS achieves significantly better or comparable results than Devin et al. [27] and the PS(Ablation) method, suggesting the latent representation alignment serves as the fundamental step to enable effective module stitching. Our method also outperforms Plain MLP policy, which indicates both modular policy design and latent representation alignment are beneficial.

For more challenging cases (E4-E8) where both task and robot configurations are drastically different, though all methods do not exhibit strong success rates, PS demonstrates notably higher touching rates. This implies more meaningful interactions that can potentially lead to a higher success rate and faster adaption in our few-shot experiments. Qualitatively, Plain baseline shows limited attempts to complete the task with frequent aimless arm swinging. Though PS(Ablation) and Devin et al. [27] methods show higher touching rates than the Plain baseline, they lack consistent attempts to move the target object to the goal after touching. In contrast, our PS method makes the robot attempt

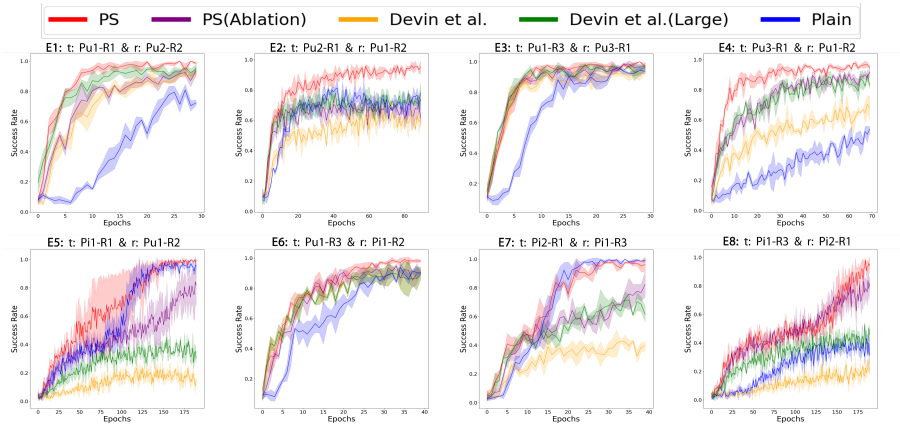


Fig. 4: **Success rates of the few-shot transfer learning in simulation.**

to push the object to the goal in most experiments. We show such qualitative comparisons in our supplementary video.

4.1.2 Results: Few-shot Transfer in Simulation

In Fig.4, PS achieves the highest or comparable success rates. Notably, in E2 and E4, while other methods tend to get stuck at local minima and reach a plateau at sub-optimal success rates, PS converges to significantly higher rates. In some experiments such as E1, E4, E5 and E7, PS transfers much faster and achieve higher success rates during the early phase of fine-tuning. This high transfer efficiency supports our hypothesis in the zero-shot experiment that the high touching rates achieved by PS can facilitate transfer learning through meaningful interactions. Even in challenging scenarios where all methods struggle to achieve a high success rate, PS can quickly adapt based on limited interactions. Furthermore, while some methods may achieve comparable results in a few instances, PS is the only method that demonstrates consistent satisfactory performance across all scenarios, suggesting that PS serves as a stable transfer learning framework.

4.2 Real World Experiment

First, we aim to evaluate the zero-shot transfer performance of PS in the real world by directly transferring the simulated policies to novel robot-task combinations. Moreover, we are interested in accessing the feasibility of continuously improving such policies given limited real-world interactions.

Setup. In our real-world experiment (Fig.5), we include a Robot1 as a 6-DoF UR5 arm, a Robot2 as the same UR5 arm but with the fifth joint being locked, a Task1 as pushing a cube to a goal position, and a Task2 as pushing a cylinder. The friction coefficient between the object and the table is 0.16 for Task 1 and 0.72 for Task 2. We first train Robot1-Task2 and Robot2-Task1 pairs in simulation. We then create Task1-Robot1 through policy stitching. For the few-shot experiments, we first fine-tune the policy for 50 epochs in simulation and test it in the real world. Following this, we then allow the policy to interact with the physical world for an additional 6 epochs (5 hours). We use an Intel RealSense D435i to detect the ArUco markers [65] attached to the object and the goal position on the table. Both objects are 3D printed. The cylinder has a sandpaper at its bottom to increase the friction coefficient. All evaluation results are based on 20 testing instances.

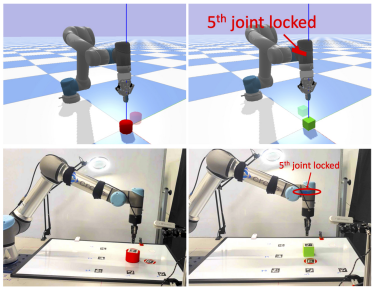


Fig. 5: **Real World Setup**.

Zero-Shot Results. PS achieves a success rate of **40%** and a touching rate of **100%** in zero-shot transfer. In comparison, both the PS(Ablation) method and the Plain baseline have 0% success rate. The PS(Ablation) method achieves a 75% touching rate, while the Plain baseline cannot even touch the object. The superior performances of PS indicate the great potential of policy stitching to generalize to novel robot-task combinations in real-world settings.

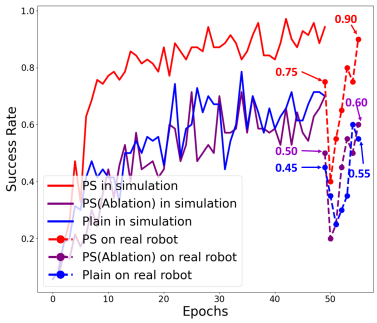


Fig. 6: **Few-Shot Results**. Solid curves shown the fine-tuning in simulation. Dotted curves show the further fine-tuning in the real world.

Few-Shot Results. As shown by the solid curves in Fig.6, after the first stage of fine-tuning in simulation, PS achieves the highest success rate of about 90% in simulation, surpassing both the PS(Ablation) and Plain methods (∼70%). When this policy is directly transferred to the real UR5 arm, PS still achieves the highest success rate at 75%, followed by the PS(Ablation) at 50% and the Plain method at 45%.

These results indicate that the modular design with relative representation achieves a promising success rate on a new task in both simulation and the real world after a few epochs of fine-tuning. Due to the sim2real gap, the success rates of all methods drop to some extent. However, after 6 epochs of fine-tuning on the real robot platform shown as the dotted curves, PS results in a final success

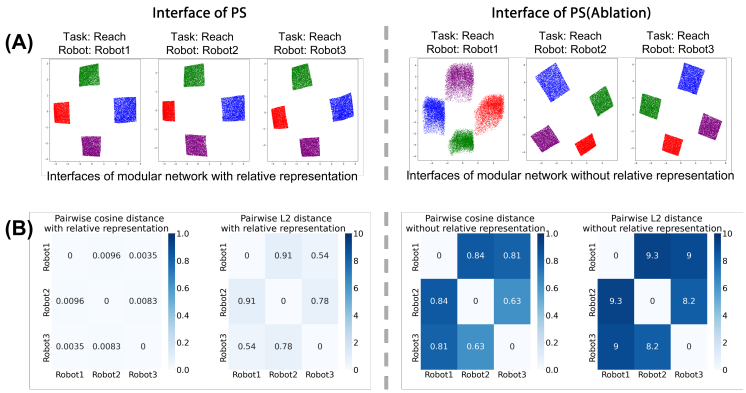


Fig. 7: **Latent Space Analysis.** We train robots with 3 different kinematics for the reaching task and visualize their latent space colorized by 4 groups of task states, similar to Fig.1. (A) Visualization of the 2D principal components of the latent representations. (B) The average pairwise cosine distances and L2 distances between each pair of the stitched latent states.

rate of 90%. Although the success rates of both baselines (i.e., 60% and 55%) are improved after the fine-tuning, there is still a significant gap with the performance of PS. Through both zero-shot and few-shot experiments, we ensure that our method performs effectively, efficiently, and reliably in physical scenarios.

4.3 Analysis: Latent Space at Module Interface

To understand how our latent space alignment benefits effective policy stitching by learning transferable representations, we provide further quantitative and qualitative analysis. Since the dimension of the latent embedding at the task-robot module interface is 128, we cannot directly visualize the latent representations. Therefore, we apply the Principal Component Analysis (PCA) [66] to reduce the dimension of the representation to 2. We train robots with 3 different kinematics on a reaching task with and without our latent space alignment. We colorize the principal components based on their task states and group them into four directions as in our motivation example in Fig.1(C). As shown in Fig.7(A), the latent spaces of all policies trained with PS remain consistent across various training environments, while the latent spaces of policies trained without latent space alignment still show approximate isometric transformations. The consistent latent space explains the effectiveness of our policy stitching method.

To account for potential information loss during PCA visualization, we also provide quantitative analysis. In Fig.7(B), we calculate the pairwise cosine distances and L2 distances (Appendix B.2) between the raw high-dimensional latent states of different stitched policies. The pairwise distances are significantly smaller when the latent representations are aligned. PS achieves an average cosine distance of 0.0055 and L2 distance of 0.240, while ablation has a much higher cosine distance of 0.865 and L2 distance of 1.275. This analysis indicates that the latent representations of normal modular networks differ greatly from each other, and the latent representation alignment can successfully reduce such differences and facilitate the learning of transferable representations for policy stitching.

5 Conclusion, Limitation and Future Work

In this work, we propose Policy Stitching, a novel framework to transfer robot learning policy to novel robot-task combinations. Through both simulated and real-world 3D robot manipulation experiments, we demonstrate the significant benefits of our modular policy design and latent space alignment in PS. PS paves a promising direction for effective and efficient robot policy transfer under the end-to-end model-free RL framework.

One limitation is that our current method of selecting anchor states based on clusters of test states may not generalize to scenarios with high-dimensional state representations, such as images. An exciting future direction is to study self-supervised methods to disentangle latent features of state representations to select the anchors. Another limitation is that our adapted relative representation method still cannot enforce strict invariance of the latent features, hence requires some level of fine-tuning. It will be interesting to explore alternative methods for aligning modules of networks without the need for anchor states, such as explicit communication protocols or constraints. Such methods may further simplify the approach and lead to stronger zero-shot performance.

References

[1] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[2] J. Bongard, V. Zykov, and H. Lipson. Resilient machines through continuous self-modeling. *Science*, 314(5802):1118–1121, 2006.

[3] A. S. Polydoros and L. Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

[4] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

[5] R. Kwiatkowski and H. Lipson. Task-agnostic self-modeling machines. *Science Robotics*, 4 (26):eaau9354, 2019.

[6] B. Chen, Y. Hu, L. Li, S. Cummings, and H. Lipson. Smile like you mean it: Driving animatronic robotic face with learned models. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2739–2746. IEEE, 2021.

[7] K. Hang, W. G. Bircher, A. S. Morgan, and A. M. Dollar. Manipulation for self-identification, and self-identification for better manipulation. *Science robotics*, 6(54):eabe1321, 2021.

[8] B. Chen, R. Kwiatkowski, C. Vondrick, and H. Lipson. Fully body visual self-modeling of robot morphologies. *Science Robotics*, 7(68):eabn1944, 2022.

[9] R. Kwiatkowski, Y. Hu, B. Chen, and H. Lipson. On the origins of self-modeling. *arXiv preprint arXiv:2209.02010*, 2022.

[10] Y. Hu, B. Chen, and H. Lipson. Egocentric visual self-modeling for legged robot locomotion. *arXiv preprint arXiv:2207.03386*, 2022.

[11] J. Hua, L. Zeng, G. Li, and Z. Ju. Learning for a robot: Deep reinforcement learning, imitation learning, transfer learning. *Sensors*, 21(4):1278, 2021.

[12] R. Yang, H. Xu, Y. Wu, and X. Wang. Multi-task reinforcement learning with soft modularization. *Advances in Neural Information Processing Systems*, 33:4767–4777, 2020.

[13] J. A. Mendez, H. van Seijen, and E. Eaton. Modular lifelong reinforcement learning via neural composition. *arXiv preprint arXiv:2207.00429*, 2022.

[14] M. Gygli, J. Uijlings, and V. Ferrari. Towards reusable network components by learning compatible representations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7620–7629, 2021.

[15] L. Moschella, V. Maiorca, M. Fumero, A. Norelli, F. Locatello, and E. Rodolà. Relative representations enable zero-shot latent space communication. *arXiv preprint arXiv:2209.15430*, 2022.

[16] M. E. Taylor and P. Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

[17] Z. Zhu, K. Lin, and J. Zhou. Transfer learning in deep reinforcement learning: A survey. *arXiv preprint arXiv:2009.07888*, 2020.

[18] S. Mohanty, J. Poonganam, A. Gaidon, A. Kolobov, B. Wulfe, D. Chakraborty, G. Šemetulskis, J. Schapke, J. Kubilius, J. Pašukonis, et al. Measuring sample efficiency and generalization in reinforcement learning benchmarks: Neurips 2020 procgen benchmark. *arXiv preprint arXiv:2103.15332*, 2021.

[19] P. Jian, C. Yang, D. Guo, H. Liu, and F. Sun. Adversarial skill learning for robust manipulation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2555–2561. IEEE, 2021.

[20] A. Tirinzoni, R. Rodriguez Sanchez, and M. Restelli. Transfer of value functions via variational methods. *Advances in Neural Information Processing Systems*, 31, 2018.

[21] Y. Zhang and M. M. Zavlanos. Transfer reinforcement learning under unobserved contextual information. In *2020 ACM/IEEE 11th International Conference on Cyber-Physical Systems (ICCPS)*, pages 75–86. IEEE, 2020.

[22] C. Liu, Y. Zhang, Y. Shen, and M. M. Zavlanos. Learning without knowing: Unobserved context in continuous transfer reinforcement learning. In *Learning for Dynamics and Control*, pages 791–802. PMLR, 2021.

[23] G. Konidaris and A. Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 489–496, 2006.

[24] A. Lazaric, M. Restelli, and A. Bonarini. Transfer of samples in batch reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 544–551, 2008.

[25] F. Fernández and M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 720–727, 2006.

[26] G. D. Konidaris and A. G. Barto. Building portable options: Skill transfer in reinforcement learning. In *Ijcai*, volume 7, pages 895–900, 2007.

[27] C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 2169–2176. IEEE, 2017.

[28] F. Doshi-Velez and G. Konidaris. Hidden parameter markov decision processes: A semiparametric regression approach for discovering latent task parametrizations. In *IJCAI: proceedings of the conference*, volume 2016, page 1432. NIH Public Access, 2016.

[29] T. W. Killian, S. Daulton, G. Konidaris, and F. Doshi-Velez. Robust and efficient transfer learning with hidden parameter markov decision processes. *Advances in neural information processing systems*, 30, 2017.

[30] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

[31] A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, H. P. van Hasselt, and D. Silver. Successor features for transfer in reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[32] J. Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

[33] J. Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

[34] J. Schmidhuber. A neural network that embeds its own meta-levels. In *IEEE International Conference on Neural Networks*, pages 407–412. IEEE, 1993.

[35] S. Thrun and L. Pratt. *Learning to learn*. Springer Science & Business Media, 2012.

[36] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. Building machines that learn and think like people. *Behavioral and brain sciences*, 40:e253, 2017.

[37] J. Vanschoren. Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*, 2018.

[38] C. Finn, K. Xu, and S. Levine. Probabilistic model-agnostic meta-learning. *Advances in neural information processing systems*, 31, 2018.

[39] C. Finn, A. Rajeswaran, S. Kakade, and S. Levine. Online meta-learning. In *International Conference on Machine Learning*, pages 1920–1930. PMLR, 2019.

[40] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018.

[41] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[42] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-learning with memory-augmented neural networks. In *International conference on machine learning*, pages 1842–1850. PMLR, 2016.

[43] T. Munkhdalai and H. Yu. Meta networks. In *International conference on machine learning*, pages 2554–2563. PMLR, 2017.

[44] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. Rl ^{2}: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[45] D. Zhao, J. von Oswald, S. Kobayashi, J. Sacramento, and B. F. Grewe. Meta-learning via hypernetworks. *4th Workshop on Meta-Learning at NeurIPS 2020, Vancouver, Canada*, 2020.

[46] D. Ha, A. Dai, and Q. V. Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

[47] J. Von Oswald, C. Henning, J. Sacramento, and B. F. Grewe. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*, 2019.

[48] J. Beck, M. T. Jackson, R. Vuorio, and S. Whiteson. Hypernetworks in meta-reinforcement learning. *arXiv preprint arXiv:2210.11348*, 2022.

[49] Z. Xian, S. Lal, H.-Y. Tung, E. A. Platanios, and K. Fragkiadaki. Hyperdynamics: Meta-learning object and agent dynamics with hypernetworks. *arXiv preprint arXiv:2103.09439*, 2021.

[50] A. Goyal, A. Lamb, J. Hoffmann, S. Sodhani, S. Levine, Y. Bengio, and B. Schölkopf. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*, 2019.

[51] S. Mittal, A. Lamb, A. Goyal, V. Voleti, M. Shanahan, G. Lajoie, M. Mozer, and Y. Bengio. Learning to combine top-down and bottom-up signals in recurrent neural networks with attention over modules. In *International Conference on Machine Learning*, pages 6972–6986. PMLR, 2020.

[52] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[53] C. Olah. Visualizing representations. http://colah.github.io/posts/2015-01-Visualizing-Representations/, 2015. Accessed: Month Day, Year.

[54] K. Lenc and A. Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 991–999, 2015.

[55] T. Hofmann, B. Schölkopf, and A. J. Smola. Kernel methods in machine learning. *The annals of statistics*, 36(3):1171–1220, 2008.

[56] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the royal statistical society. series c (applied statistics)*, 28(1):100–108, 1979.

[57] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[58] S. Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1998.

[59] Q. Gallouédec, N. Cazin, E. Dellandréa, and L. Chen. panda-gym: Open-Source Goal-Conditioned Environments for Robotic Learning. *4th Robot Learning Workshop: Self-Supervised and Lifelong Learning at NeurIPS*, 2021.

[60] J.-T. Huang, J. Li, D. Yu, L. Deng, and Y. Gong. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 7304–7308. IEEE, 2013.

[61] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1717–1724, 2014.

[62] M. Long, H. Zhu, J. Wang, and M. I. Jordan. Unsupervised domain adaptation with residual transfer networks. *Advances in neural information processing systems*, 29, 2016.

[63] F. Yang, C. Yang, H. Liu, and F. Sun. Evaluations of the gap between supervised and reinforcement lifelong learning on robotic manipulation tasks. In *Conference on Robot Learning*, pages 547–556. PMLR, 2022.

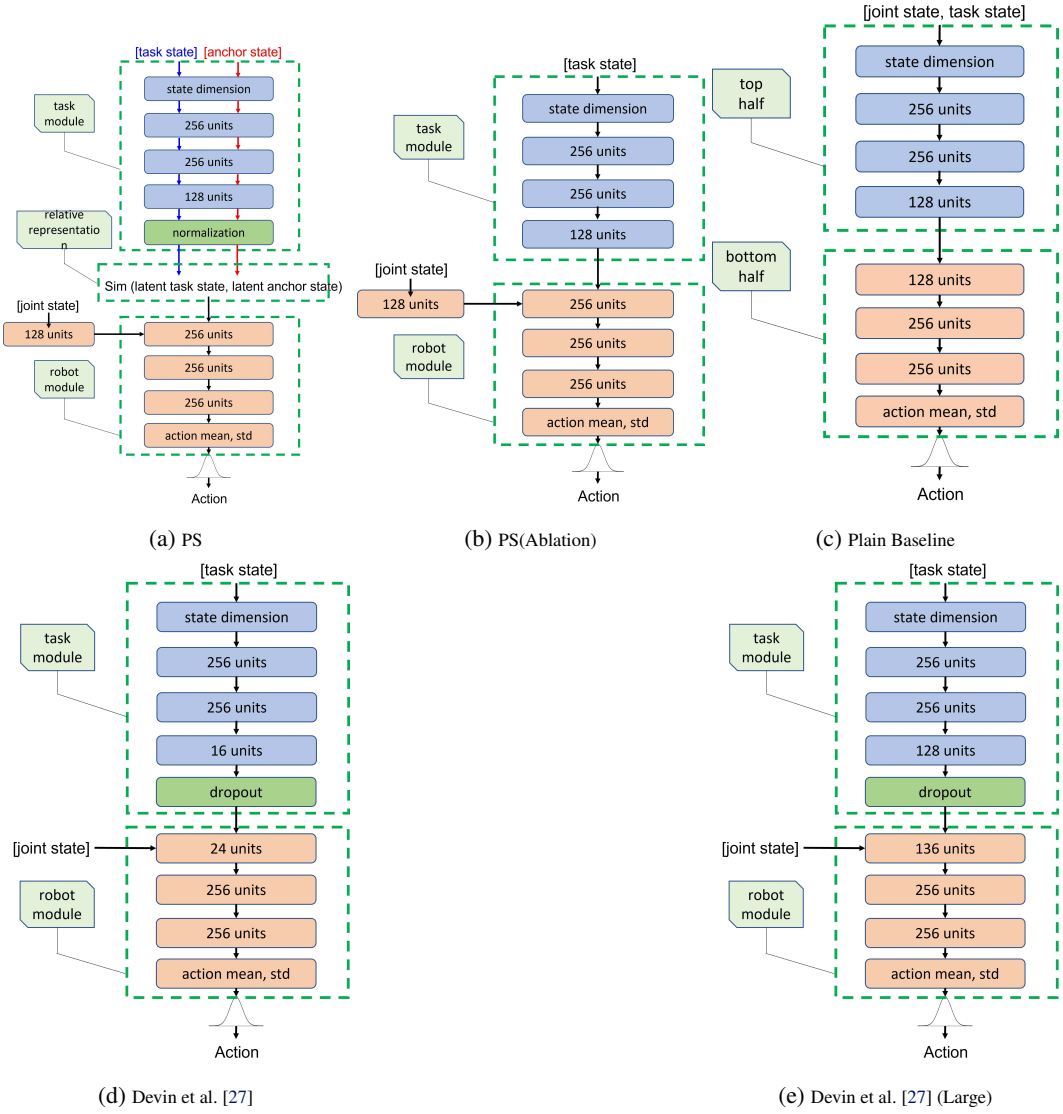
[64] J. Yoon, S. Kim, E. Yang, and S. J. Hwang. Scalable and order-robust continual learning with additive parameter decomposition. *arXiv preprint arXiv:1902.09432*, 2019.

[65] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. Marín-Jiménez. Aruco: a minimal library for augmented reality applications based on opencv. *Journal of Real-Time Image Processing*, 9(2):399–406, 2014. doi:10.1007/s11554-013-0320-4. URL https://doi.org/10.1007/s11554-013-0320-4.

[66] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(11):559–572, 1901. doi:10.1080/14786440109462720.

Appendix

A Network Structure



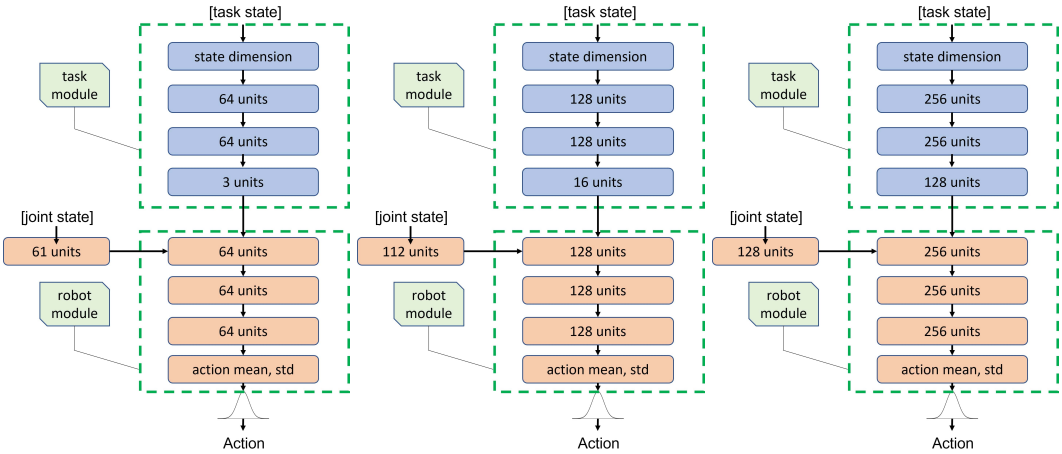
(a) PS (b) PS(Ablation) (c) Plain Baseline

(d) Devin et al. [27] (e) Devin et al. [27] (Large)

Fig. 8: Detailed network structure of PS, PS(Ablation), Devin et al., Devin et al.(Large) and Plain Baseline methods.

B Analysis of the Module Interface

We carry out additional analysis of the latent representations across different sizes of modular networks. We build three different sizes of modular networks, each interface dimension being 3D, 16D, and 128D as shown in Fig.9. The transferable representation is added to these networks as shown in Fig.8a. We construct six networks (3 different sizes, with and without relative representation) to perform the reaching task as described in Fig.1.



(a) Small modular network (b) Medium modular network (c) Large modular network

Fig. 9: The detailed architectures of the modular networks with three different interface dimensions.

B.1 Visualization of the latent representations at the modules interface

For the small networks with 3D interfaces, we plot the 3D latent representations directly. For the medium and large networks with 16D and 128D interfaces, we use PCA [66] to reduce the dimension to 2D. Fig.10 shows the visualization of the interface of the six networks trained with different random seeds. The isometric transformation relationship is shown for the PS(Ablation) method across all sizes of modular networks, and with the help of transferable presentation, PS achieves near invariance. Similarly, Fig.11 shows the interface of the networks trained with different robot types. The transferable representation achieves near invariance to isometric transformations across all types of robots.

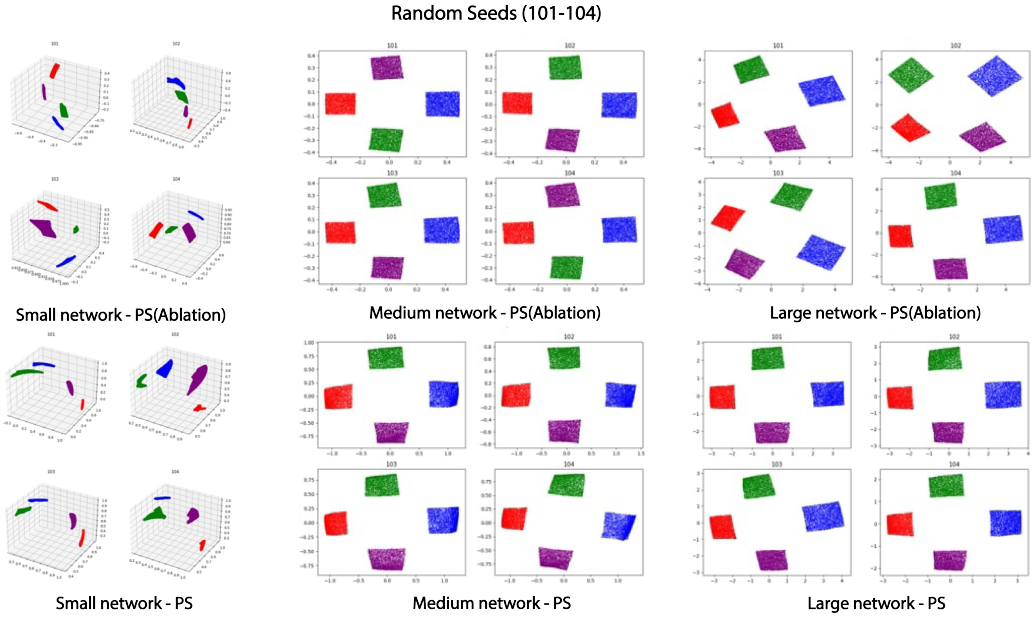


Fig. 10: **Latent Space Visualization** Train each policy network four times with four different random seeds (101-104). Without transferable representation, the latent representations at the interfaces have an approximate isometric transformation relationship. With relative representation, these latent representations are isometrically similar.

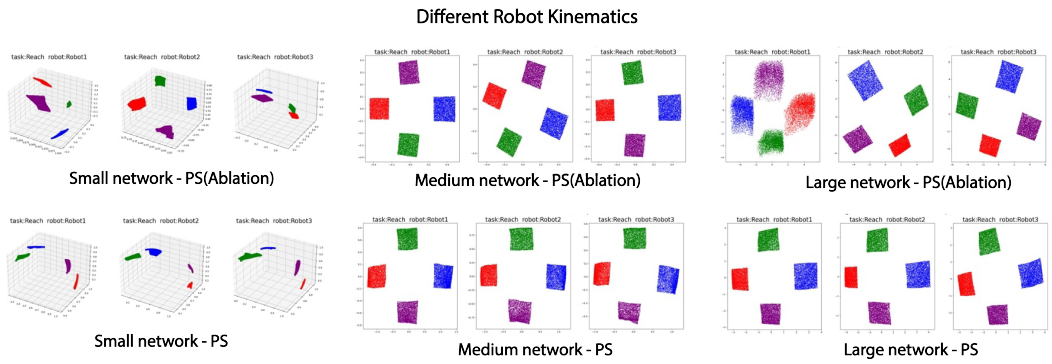


Fig. 11: **Latent Space Visualization** Train each policy network for the reaching task with three different types of robots as shown in Figure 3. With relative representation, the latent representations at the module interfaces are nearly the same across different environments. Without it, they have an approximately isometric transformation relationship.

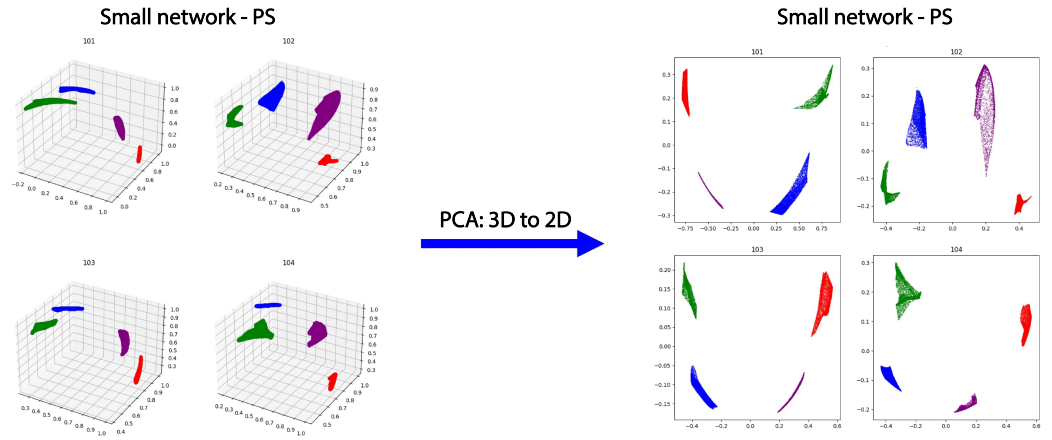
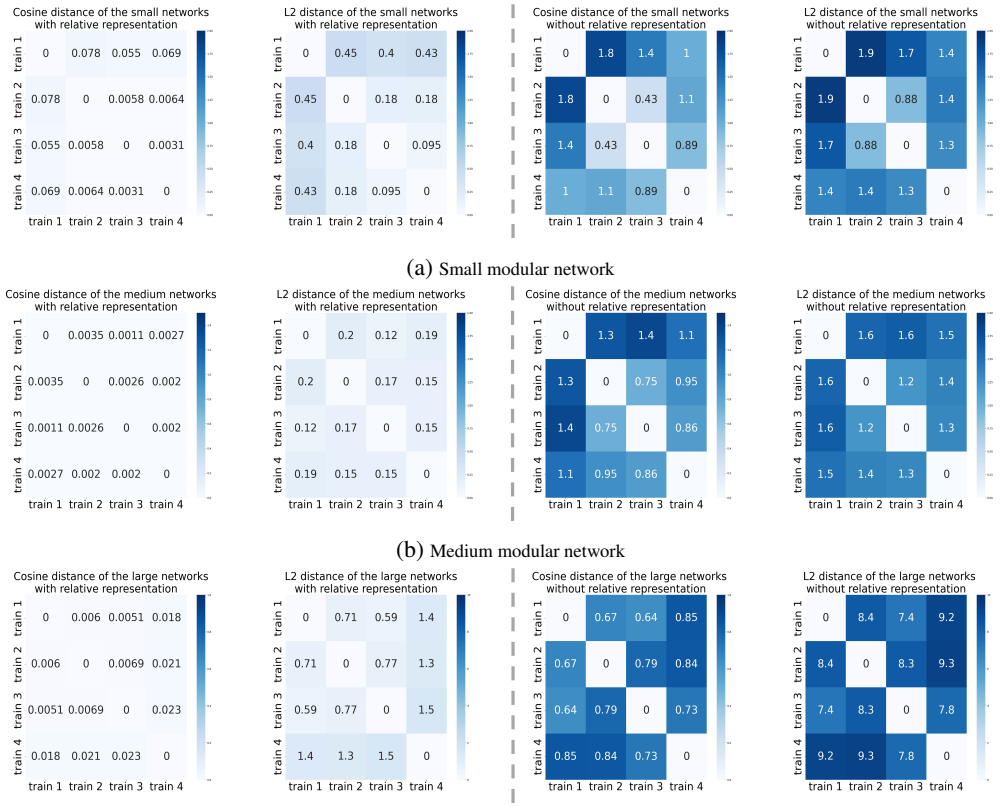


Fig. 12: **Limitation of visualization with PCA.** Raw latent distributions are very similar to each other in the 3D space, but after compression to 2D with PCA, the visualization results are quite different.

PCA is an information lossy compression process. The PCA method only guarantees identical output results when the input data sets are identical. When the input data sets are similar but not identical, the output results may vary considerably. In our experiments using PCA for visualization, we have observed that the PCA results of most interfaces with transferable representations are similar. However, in rare cases, we have noticed significant differences in the PCA results. As shown in Fig.12, the original 3D latent states have very similar distributions across the four different runs, but after the dimension reduction to 2D with PCA, the visualization results show isometric transformations. Moreover, in the case of small modular networks with 3D interfaces, achieving a high success rate of approximately 100% often requires a considerable amount of training time. Occasionally, the network may converge to a local minimum with a success rate of around 90%. When it converges to a local minimum, the latent representation at its interface typically differs from those that converge to the global minimum. PCA only provides an intuitive idea of the behavior at module interface, thus, we accompany these visualizations with quantitative analysis.

B.2 Quantitative analysis of the latent representations at the modules interface

To measure the similarity between two different latent representations, we use cosine and L2 pairwise distances. We compute the pairwise distance between two latent task states derived from the same input state. By considering a dataset of input states, we calculate the mean of the pairwise distances across all input states, obtaining the average pairwise distance between two modular networks.



(a) Small modular network

(b) Medium modular network

(c) Large modular network

Fig. 13: Cosine and L2 distances of different networks with and without transferable representation for different random seeds.

Given an input task state set $\mathbb{S}_{E,\mathcal{T}}$, the average pairwise cosine distance and L2 distance are defined as

$$\bar{d}_{cos} = \sum_{i=1}^{|\mathbb{S}_{E,\mathcal{T}}|} \left(1 - S_C\left(g_k^1\left(s_{E,\mathcal{T}}^i\right), g_k^2\left(s_{E,\mathcal{T}}^i\right)\right)\right) / \left|\mathbb{S}_{E,\mathcal{T}}\right|, \tag{4}$$

$$\bar{d}_{L2} = \sum_{i=1}^{|\mathbb{S}_{E,\mathcal{T}}|} d_{L2}\left(g_k^1\left(s_{E,\mathcal{T}}^i\right), g_k^2\left(s_{E,\mathcal{T}}^i\right)\right) / \left|\mathbb{S}_{E,\mathcal{T}}\right|, \tag{5}$$

where $S_C(\boldsymbol{a}, \boldsymbol{b}) = \frac{\boldsymbol{a}\boldsymbol{b}}{\|\boldsymbol{a}\|\|\boldsymbol{b}\|}$ is the cosine similarity and $d_{L2}(p, q) = \|p - q\|$ is the L2 distance.

Fig.13 shows the average pairwise distances of modular networks trained with four different random seeds. We calculate the distances for different sizes of networks shown in Figure 9 . We also calculate the mean and standard deviations of the data in Fig.9 and present them in Tab. 2. The results show that the transferable representation largely reduces the average pairwise distances of the latent spaces between different training runs.

We also train the modular networks in different environments and calculate the pairwise distances at the interfaces. Specifically, we train the policy networks on the reaching task with different robots shown in Figure 3. The average pairwise distances are shown in Figure 14 and we calculate the mean values and standard deviations in Tab.3. These quantitative results show that the relative representation makes the module interfaces much more similar to each other when trained in different environments.

	cosine distance	L2 distance
PS	$\mathbf{0.0363 \pm 0.0319}$	$\mathbf{0.289 \pm 0.141}$
PS(Ablation)	1.106 ± 0.434	1.426 ± 0.322

(a) Small modular network

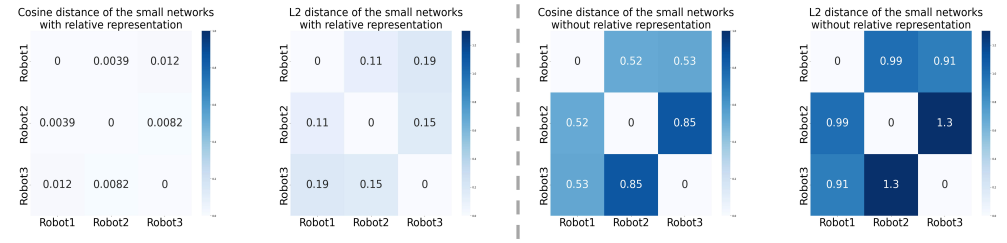
	cosine distance	L2 distance
PS	$\mathbf{0.00231 \pm 0.00073}$	$\mathbf{0.1633 \pm 0.0294}$
PS(Ablation)	1.054 ± 0.218	1.442 ± 0.151

(b) Meidum modular network

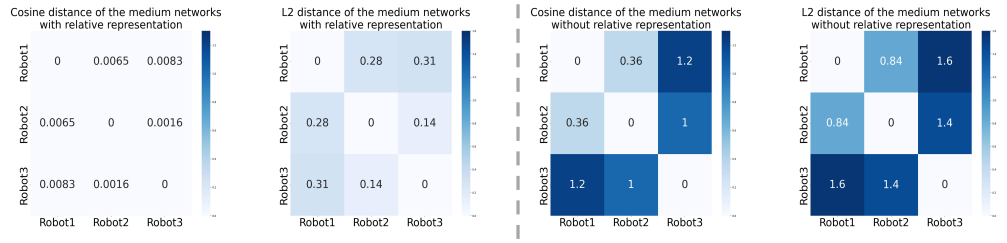
	cosine distance	L2 distance
PS	$\mathbf{0.013 \pm 0.007}$	$\mathbf{1.051 \pm 0.368}$
PS(Ablation)	0.753 ± 0.081	8.386 ± 0.687

(c) Large modular network

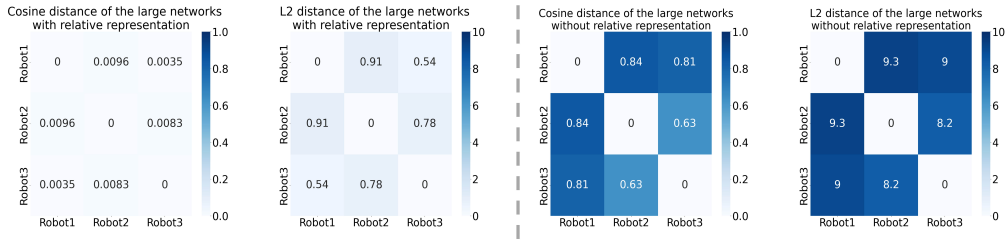
Tab. 2: Mean and standard deviation values of the average pairwise distances between trainings with four different random seeds (101-104)



(a) Small modular network



(b) Medium modular network



(c) Large modular network

Fig. 14: Cosine and L2 distances of different networks with and without transferable representation for different robot setup.

	cosine distance	L2 distance
PS	$\mathbf{0.0082 \pm 0.0035}$	$\mathbf{0.149 \pm 0.032}$
PS(Ablation)	0.633 ± 0.153	1.066 ± 0.165

(a) Small modular network

	cosine distance	L2 distance
PS	$\mathbf{0.0055 \pm 0.0029}$	$\mathbf{0.240 \pm 0.075}$
PS(Ablation)	0.865 ± 0.367	1.275 ± 0.311

(b) Medium modular network

	cosine distance	L2 distance
PS	$\mathbf{0.0071 \pm 0.0026}$	$\mathbf{0.743 \pm 0.152}$
PS(Ablation)	0.760 ± 0.094	8.822 ± 0.448

(c) Large modular network

Tab. 3: Mean and standard deviation values of the average pairwise distances between trainings with three different types of robot kinematics