When Does Transfer Help? Data-Dependent Transfer in Robotic Manipulation

Abivishaq Balasubramanian
Georgia Institute of Technology

Nadira Mahamane
Georgia Institute of Technology

Sreeranj Jayadevan
Georgia Institute of Technology

Abstract

This research investigates the effectiveness of transfer learning in robotic manipulation tasks, specifically focusing on transferring skills between pick and push cube operations. We explored two distinct approaches: Reinforcement Learning using Proximal Policy Optimization (PPO) and Diffusion Policy with varying amounts of demonstration data. While our PPO implementation demonstrated the feasibility of skill transfer, our primary findings through the Diffusion Policy approach revealed that transfer learning shows promising results when sufficient training data is available. However, the benefits diminished with smaller datasets, accompanied by increased variance and lower success rates. We observed modest improvements in learning speed with larger datasets during transfer learning. Our research provides insights about the relationship between dataset size and transfer learning effectiveness in robotic manipulation. These findings contribute to our understanding of how pre-trained policies can be leveraged to enhance learning efficiency in related manipulation tasks. The highlight of this research is to show that transfer learning can enable faster learning of skills thus computationally and timely efficient.

1. Introduction

In this project, we aim to explore transfer learning for robot manipulation policies. For different robot manipulation tasks like pick cube and push cube, can we use the policy learned for one task to help learn a policy for another task? Essentially, there is a set of datasets $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2, ...\}$ that contains demonstrations for a set of tasks $T = \{T_1, T_2, ..\}$. Dataset $\mathcal{D}_i$ corresponds to demonstrations for task T_i. A set of policies $\pi = \{\pi_1, \pi_2, ...\}$ are used to accomplish each of the tasks. The objective is to effectively create a policy set using the set of datasets that can work for a (ensure success) different task in the list. Additionally, coming up with a policy that accomplishes that task most effectively will be using the ManiSkill [3] dataset, which is a robot manipulation dataset with human demonstrations. The policy inputs are joint states and camera images. The policy outputs a 7-sized vector that represents the end effector position, orientation, and command. This action is then achieved by an inverse kinematic Proportional derivative(PD) End-effector(EE) delta pose controller. We hypothesize that using transfer learning would be computationally more efficient than training a model from scratch for each skill. The second hypothesis is that the policy learned using transfer learning will outperform the policy trained from scratch.

2. Related Work

Robot manipulation is a fundamental aspect of robotic systems. Better policies for robot manipulation are essential for more capable robotics systems. Traditionally, a policy was trained for a specific task, resulting in different policies for different tasks. Some works focus on learning a single policy capable of handling multiple tasks. One research field, continual learning [7], aims to develop policies that continuously learn new skills over time. However, such policies often suffer from catastrophic forgetting, where performance on older tasks degrades, and previous skills are forgotten. Additionally, inspired by the success of large models, there has been a push for a single policy that can learn multiple tasks by leveraging large datasets [8]. Since robotic data is expensive, the field still lacks the required amount of data. Therefore, learning specific policies for specific tasks could still be beneficial.

Deep neural networks have been shown to learn transferable features that can be used across different computer vision tasks [10]. Models like AlexNet [6], VGG [9], and ResNet [4] have been used as feature extractors. These transferable features enable learning on smaller datasets, faster training times, and better performance compared

to training from randomly initialized weights. We aim to explore whether such benefits can also be seen in the domain of imitation learning for robot manipulation.

Furthermore, prior work suggested that combining behavioral cloning and GAIL shows stability and efficiency in various environments. Jena et al [5] in their paper Augmenting GAIL with BC for sample efficient imitation learning compare different algorithms' performances and highlight the advantages of the augment GAIL method. It appears such a technique extends well to real-world robots, outperforms baselines in different simulators, and demonstrates stability and efficiency. Moreover, the paper highlights how the method requires fewer environments and can be extended to an off-policy setting with minimal medications.

Diffusion Policy introduces action diffusion to improve visuomotor policy learning, enhancing action accuracy, inference speed, and real-time control by modifying training loss and excluding observation features from denoising [2]. Unlike LSTM-GMM and IBC, it effectively handles multimodal action distributions, achieving a 46.9% performance improvement in benchmarks.

Building on methods like augmented GAIL and Behavioral Cloning, our work uses Diffusion Policy to explore multimodal action learning for robot manipulation tasks, specifically transferring policies from cube-picking to cube-pushing. This complements prior approaches by leveraging Diffusion Policy's efficiency and robustness in multimodal environments.

3. Methods/Technical Approach

The primary goal of this work is to evaluate the effectiveness of transfer learning for policies across different tasks within an imitation learning framework. Transfer learning for policies is challenging, potentially due to limitations in dataset size. Small imitation learning datasets can lead to policies that fail to generalize effectively. To explore this, we used expert demonstrations collected from ManiSkill 2/3 to train both an RL PPO policy and a diffusion policy.

3.1. Baseline Approach

We implemented a baseline by training a policy for the tasks from scratch, using randomly initialized weights. Different hyperparameter tuning is done and the one with the highest success rate is maintained. It might also be insightful to keep the quickly trained one with a decent success rate. Currently, within 5% success rate difference from the top success rate is considered decent. The policy trained from random weights is called the random policy. For the baselines, we used results from RL plus PPO for the PickCube task.

3.2. Proposed Approach

For each method, we use the random policy trained on one task as the initial policy to train the other task. This policy is termed a transfer policy since it is supposed to benefit from the features learned for the first task. For instance, a policy is trained on the "Pick cube" task and then test whether it transfers well to the "push cube" task.

3.3. Evaluation and Metrics

To evaluate these policies, we measure the saturation success rate, which is the highest success rate achieved after extensive hyperparameter tuning. In preliminary experiments, imitation learning did not reach a 100% success rate; however, with reinforcement learning, the same architecture did achieve perfect success, indicating that the architecture itself can represent an ideal policy. This gap suggests that insufficient data may be limiting performance in the imitation learning setup. In cases where the policy achieves 100% success with imitation learning, we will instead compare the time to reach full success, observing whether transfer learning accelerates the learning process.

3.4. Dataset and simulation

We are using ManiSkill2/3 and it contains a heterogeneous collection of out-of-the-box task families for 20 manipulation skills. It uses the Franka Emika - Panda Manipulator which has 7 degrees of freedom. Stable Baselines 3 package was used in the PickCube-v0 environment with state observations. For our imitation learning approach, we started with the open-source model from [3], which enabled us to generate baseline results. This involved adapting the environment, tuning hyperparameters, aligning the number of observations to match the task, and leveraging available expert demonstrations without the challenges typically associated with data collection. Similarly, with the diffusion policy model, we use the open source documentation from [2] to train our transfer learning policy. Furthermore, the PPO model was derived from the paper [1] and was used to train on Maniskill3 demonstrations of pick and pushing cubes.

4. Experiments and Results

In order to study the evaluation metrics for transfer learning, firstly we need to generate required baselines to compare the results. For this project, we generated the baseline for the PickCube-v0 task using an RL PPO and diffusion model.

For the RL PPO policy, we started with a trained model on the PickCube task (source task) and used its learned weights to initialize training on the PushCube task (target task). Transfer was implemented by loading checkpoints from the source task, transferring all the learned parameters

(weights and biases) to initialize the target task. Initially, we attempted full transfer (copying all weights from the source to the target), but this approach was unsuccessful. Subsequently, we tried partial transfer, where only the weights of the first layer were copied and frozen. The frozen layers retained their transferred knowledge, keeping it fixed during training on the target task.

With the diffusion policy model, we varied the number of demonstrations to assess whether using fewer training samples could still yield effective results, making the training process more time-efficient in the context of transfer learning.

Also, for the transfer learning experiments, the PickCube task served as the source task, where the policy was trained to pick up an object using a robotic arm. The resulting model checkpoint was then used to initialize training for the PushCube task, the target task, which involved pushing an object. Since the two tasks involve different numbers of observations, we padded the input for the pushing task to match the observation size of the picking task.

4.1. Reinforcement Learning (PPO)

In state-based RL, observations are all flat/dense vectors. State-based is generally faster as generating visual observations is slow. For this report, tasks were treated as continuous with an infinite horizon. This means ”done” is always False until a time limit is hit. The model implements a Proximal Policy Optimization (PPO) algorithm with a Multi-Layer Perceptron (MLP) policy network. The architecture consists of two hidden layers for the MLP.

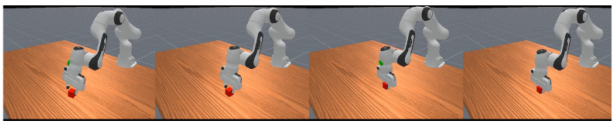
```
ActorCriticPolicy(
  (features_extractor): FlattenExtractor(
    (flatten): Flatten(start_dim=1, end_dim=-1)
  )
  (pi_features_extractor): FlattenExtractor(
    (flatten): Flatten(start_dim=1, end_dim=-1)
  )
  (vf_features_extractor): FlattenExtractor(
    (flatten): Flatten(start_dim=1, end_dim=-1)
  )
  (mlp_extractor): MlpExtractor(
    (policy_net): Sequential(
      (0): Linear(in_features=42, out_features=256, bias=True)
      (1): Tanh()
      (2): Linear(in_features=256, out_features=256, bias=True)
      (3): Tanh()
    )
    (value_net): Sequential(
      (0): Linear(in_features=42, out_features=256, bias=True)
      (1): Tanh()
      (2): Linear(in_features=256, out_features=256, bias=True)
      (3): Tanh()
    )
  )
  (action_net): Linear(in_features=256, out_features=7, bias=True)
  (value_net): Linear(in_features=256, out_features=1, bias=True)
)
```

Figure 1. Architecture used for generating RL baseline

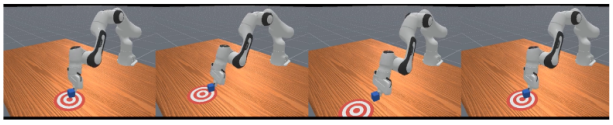
4.1.1 Experiments

Since PushCube task was fairly easy to learn, a success of 100 was achieved within 600000 rollout steps. PickCube training was done using 10000000 steps and success was achieved after about 3000000 steps and for Push with transfer of skills from Pick 2000000 steps was simulated. For transfer learning, we started with a trained model on PickCube (source task). Used its learned weights/knowledge to initialize training on PushCube (target task). Transfer was done through loading checkpoints from the source task. The steps followed were: 1.Load source checkpoint, 2.Transfer weights according to chosen mode, 3.Optionally freeze layers, 4.Continue training on target task. We initially implemented transfer learning using PPO for two tasks to see if there is any improvement. The source task for this was PickCube - training an agent to pick up a cube. The target task was PushCube - transferring the learned skills to pushing a cube and the method used was PPO with network weight transfer between similar tasks

We also had to handle different observation spaces for different tasks. This was solved using padding. Source task (PickCube): 42-dimensional observations. Target task (PushCube): 35-dimensional observations. So we padded the smaller observation space to match the larger one.



Picking Object Learned using RL PPO



Pushing Object using Partial Transfer Learning (RL PPO)

Figure 2. Visualization of a successful task executions using transfer learning from an initial model checkpoint pick cube to target task push cube.

The advantages we expected was faster learning. We expected transfer learning results to show faster initial learning compared to training PushCube from scratch. However this was not the case and we feel this has to do with the tasks inherently being not similar. This issue was however resolved using imitation learning techniques. The results from PPO was used to understand how transfer learning can be used across different tasks. Additionally, it could also be used to generate demonstrations for imitation learning for different tasks. Eventhough we did not get faster learning, there definitely was knowledge transfer. The agent was able to leverage basic skills (like cube interaction) learned from PickCube.

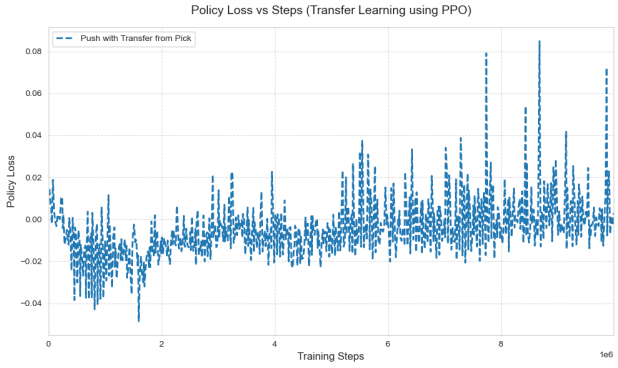


Figure 3. Policy Loss for Pick to Push Transfer

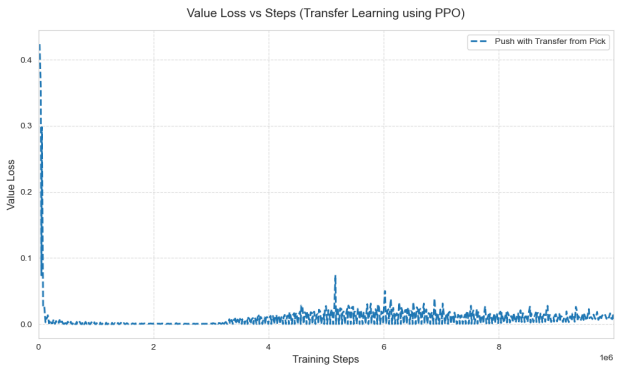


Figure 4. Value loss for Pick to Push Transfer

4.1.2 Hyperparameters and Transfer Learning

This PPO training setup is designed for the PushCube-v1 environment and leverages transfer learning using a pretrained checkpoint from the PickCube task (final ckpt.pt). The experiment uses 1024 parallel environments to collect data efficiently, allowing for rapid learning. Each environment runs for 20 steps per rollout, leading to frequent policy updates. This value (20) was higher for just PickCube task as this is more complicated and requires longer duration in each rollout to learn the task. The training process spans a total of 20 million timesteps, with evaluations occurring every 8 update steps to monitor progress.

During optimization, the data collected is divided into 32 minibatches, and each batch is iterated over for 8 epochs, ensuring thorough learning from the rollout data. The transfer learning setup was done in two modes: full transfer mode and partial transfer mode (freezing initial couple of layers). In both cases, both the policy and value function from the source checkpoint was imported. A smaller learning rate was used initially (e.g., -learning rate=1e-4) since we are starting from pre-trained weights Also total time steps was reduce from 30 million to 20 million timesteps since we are starting from a pre-trained model

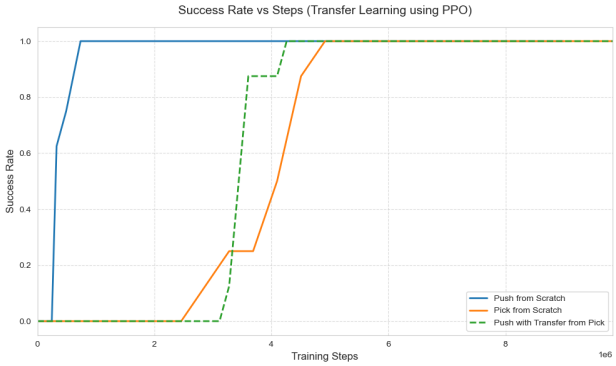


Figure 5. Success comparison for transfer learning and learning from scratch

4.2. Imitation Learning (Diffusion)

Diffusion policy performance was evaluated for the push task under two approaches: training from scratch and transfer learning using weights from the pick task. The objective was to assess the impact of transfer learning on policy performance and to understand its behavior with varying dataset sizes.

4.2.1 Experiments

The diffusion policy was trained for 30,000 iterations in both approaches. In the transfer learning scenario, a policy trained on the pick task was used to initialize the model for the push task. Due to differences in observation spaces between the tasks, the pick task included an additional dimension representing the target location in 3D space. To align the observation space for the push task, zeros were appended to account for the missing dimensions. Training trajectories were set to 16 steps, while execution during evaluation occurred over 8 steps.

To study the effect of dataset size, experiments were conducted in two settings. In the first setting, the dataset sizes for both tasks were varied, with 100, 50, and 25 demonstrations used for training. Each experiment was repeated across five random seeds to ensure statistical robustness. Results indicated that transfer learning facilitated slightly faster learning for the push task when 100 demonstrations were available, as shown in Figure 6. For smaller datasets, with 50 or 25 demonstrations, no significant differences in performance were observed between training approaches. Additionally, the saturation success rate decreased as dataset size was reduced, and variance across seeds increased, emphasizing the importance of data availability in both success rates and consistency. The results

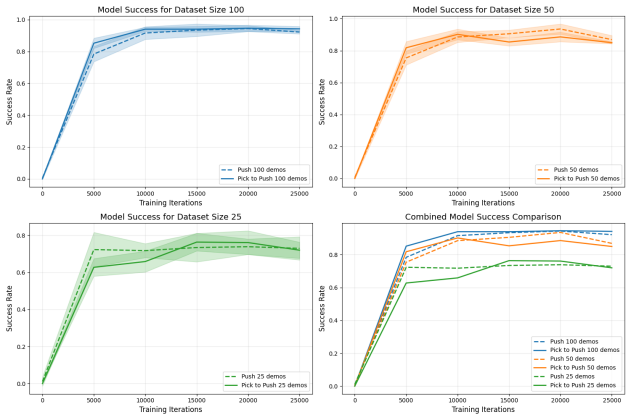


Figure 6. Impact of Dataset Size on Transfer Learning Performance

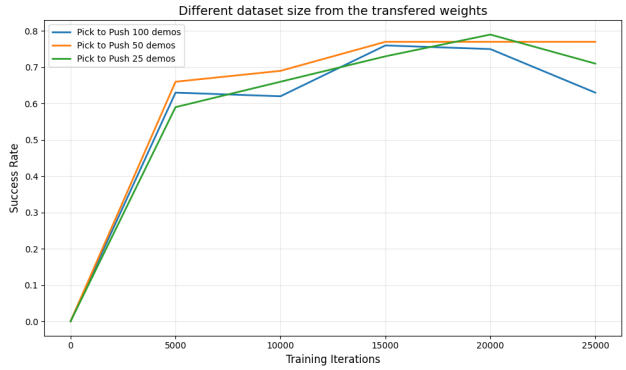


Figure 7. Impact of Initial Task Dataset Size on Transfer Learning Performance

could provide some encouragement to test out the trend for larger datasets.

In the second setting, the dataset size for the pick task was varied (100, 50, and 25 demonstrations), while the push task was consistently trained with 25 demonstrations. The policies trained on the pick task under these conditions were used to initialize training for the push task. The success rates across iterations for this case, depicted in Figure 7, show no discernible trends. Only a single seed was run for this experiment.

4.2.2 Model and Parameters

A conditional U-Net was employed as the noise prediction network. The network processes actions represented as sequences of 16 timesteps, where each timestep includes a 4-dimensional action vector comprising the 3D pose of the robotic end-effector and the gripper opening. The U-Net architecture featured dimensions of [64, 128, 256] for its down-sampling layers. The denoising process used a

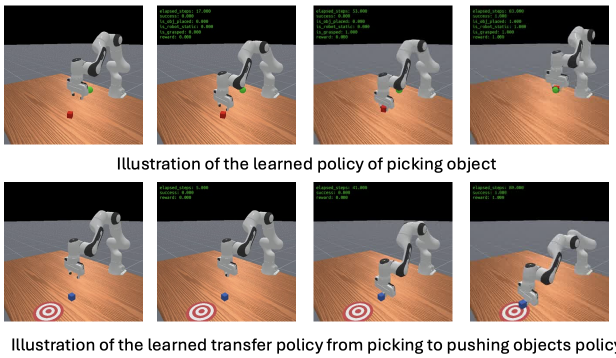


Figure 8. Visualization of a successful task executions using transfer learning from an initial model checkpoint pick cube to target task push cube.

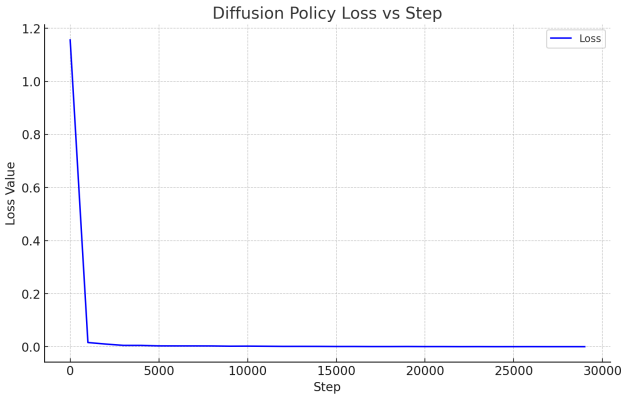


Figure 9. Diffusion Loss

DDPMScheduler, which performed noise removal iteratively over 100 timesteps during training, gradually refining the predicted actions. Padding was applied to align observation and action sequences, ensuring consistent input-output dimensions, especially at the start or end of trajectories. The loss function was the Mean Squared Error (MSE) between the predicted noise and the actual noise added to the actions during the diffusion process. Training optimization was carried out using the AdamW optimizer, with a learning rate of $10^{-4}$and a cosine learning rate scheduler. Additionally, an Exponential Moving Average (EMA) was employed to stabilize training by maintaining a smoothed version of model weights. The unet architecture used is shown in Table 1. One of the loss graphs is shown in Figure 9. The losses were stable throughout different runs.

5. Conclusion

We investigated transfer learning approaches in robotic manipulation using two distinct methods: Reinforcement Learning (PPO) and Diffusion Policy. Our primary focus was on transferring skills between pick and push cube

tasks. For PPO, we implemented transfer learning by initializing the push task with weights from a trained pick cube model, fixing observation space differences through zero-padding. While this demonstrated the feasibility of skill transfer in RL, our primary analysis centered on the Diffusion Policy approach.

The Diffusion Policy experiments explored the relationship between dataset size and transfer learning effectiveness. Our first set of experiments varied the dataset size (100, 50, and 25 demonstrations) for both tasks across five random seeds. Results showed modest improvements in learning speed with transfer learning when using 100 demonstrations, while smaller datasets showed no significant benefits. A second experiment series kept the push task dataset fixed at 25 demonstrations while varying the pick task dataset size, though no clear trends emerged from this single-seed investigation.

Our research provides valuable insights into the dynamics of transfer learning in robotic manipulation tasks. The success of the Diffusion Policy approach with larger datasets (100 demonstrations including the source task demos) suggests that transfer learning can be effective when sufficient training data is available. However, the diminishing returns with smaller datasets highlight the critical role of data quantity in transfer learning performance. The increased variance and lower success rates observed with smaller datasets shows the importance of availability of enough data for both performance and consistency. While our PPO implementation demonstrated the feasibility of transfer learning in RL, the Diffusion Policy results provide a better understanding of data requirements for effective skill transfer. These findings opens up promising avenues for future research, particularly in exploring how larger datasets might further enhance transfer learning efficiency and reduce training time. Our work suggests that scaling up demonstration data could be important for robust and efficient transfer learning in robotic manipulation tasks.

References

[1] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009. 2

[2] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023. 2

[3] Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, Xiaodi Yuan, Pengwei Xie, Zhiao Huang, Rui Chen, and Hao Su. Maniskill2: A unified benchmark for generalizable manipulation skills. In *International Conference on Learning Representations*, 2023. 1, 2

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1

[5] Rohit Jena, Changliu Liu, and Katia Sycara. Augmenting gail with bc for sample efficient imitation learning. In *Conference on Robot Learning*, pages 80–90. PMLR, 2021. 2

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012. 1

[7] Timothée Lesort, Vincenzo Lomonaco, Andrei Stoian, Davide Maltoni, David Filliat, and Natalia Díaz-Rodríguez. Continual learning for robotics: Definition, framework, learning strategies, opportunities and challenges. *Information fusion*, 58:52–68, 2020. 1

[8] Abhishek Padalkar, Acorn Pooley, Ajinkya Jain, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anikait Singh, Anthony Brohan, et al. Open x-embodiment: Robotic learning datasets and rt-x models. *arXiv preprint arXiv:2310.08864*, 2023. 1

[9] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1

[10] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in neural information processing systems*, 27, 2014. 1

6. Appendix

6.1. Reinforcement Learning for PickCube Task

3200 rollout steps were used with 32 batch size. In PPO KL divergence between old and new policies is monitored to ensure that policy updates are not too aggressive, preventing large deviations. The success rate with the pretrained weights from the Maniskill 2 tutorial was **100%**.The success rate for the trained model was **80%**. This is evident from the loss curve as well. It increases, plateaus and decreases slightly suggesting that policy started learning but did not converge to a more stable performance. More training steps or fine tuning is required to reach 100% success rate. The model that was trained was evaluated using 10 episodes or scenarios and the output was :
Success Rate: 0.8
Episode Lengths: [200, 47, 43, 45, 44, 87, 99, 90, 200, 103]

Table 1. 1D U-Net Architecture

Layer
Input
Downsample Module 1 (Conv + Residual)
Downsample Module 2 (Conv + Residual)
Downsample Module 3 (Conv + Residual)
Bottleneck (Residual Blocks)
Upsample Module 1 (ConvTranspose + Residual)
Upsample Module 2 (ConvTranspose + Residual)
Upsample Module 3 (ConvTranspose + Residual)
Condition Encoder
Final Convolution
Output

6.2. Generative Adversarial Imitation Learning

We tried two variants of Generative Adversarial Imitation Learning (GAIL). The first variant is similar to Generative Adversarial Networks (GANs). In this version, each state-action pair is treated as a separate data point, rather than considering the entire trajectory. The generator's input is a state, and its output is an action. The discriminator takes both the state and action as input and outputs a single value indicating whether the data was generated or real. The generator architecture has five layers, and the discriminator has three layers, with both having hidden layers of dimension 128. The generator's input dimension is 42, and its output dimension is 7. For the discriminator, the input dimension is $42 + 7 = 49$, and the output dimension is 1. The loss for the first variant is shown below.

The second variant uses the imitation library. The generator or policy uses the PPO object from the stable-baselines library with an MLP policy. We primarily set up the ManiSkill environment to work with this library.

Despite trying different hyperparameters, we did not get any viable policy. All policies ended up with zero success rate.

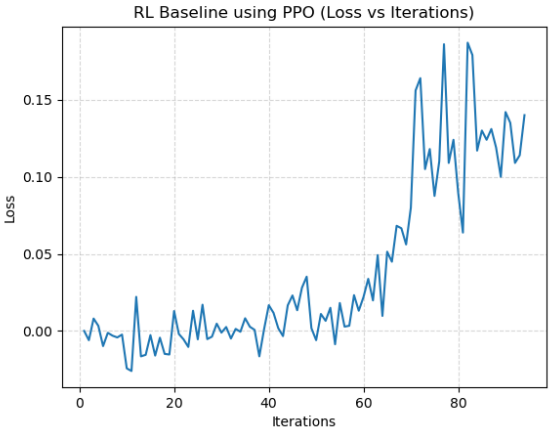


Figure 10. Loss value VS epochs

6.3. Behavior Cloning - MLP

We used trained an behaviour cloning method with the Maniskill2 data to learn the task of picking a cube to transfer it later to the tasks of pushing cubes. The model used is structured as a multilayer perceptron with the goal of mapping observation to actions. the network is structured as follow:

- a fully connected layer with input and output features. After each layer the input is updated to be the output size of the current layer.
- an activation function to introduce non-linearity
- After going through the hidden layers, a final layer maps the current dimension with a tanh activation. The activation constraints the output to a range of [-1,1] which is useful to control tasks within a fixed interval.

For this report, only state-based observations were considered. We trained our policy using 100K iterations and saved the model with the best accuracy and lowest loss value. It appears the maximum accuracy we could get with 572 batches and a learning rate of 1e-3 is **40%**. Figure 4 illustrates the evolution of the loss function as the number of iterations increases. We noticed that the loss decreases as the number of iterations increases and becomes constant around 80k. Figure 5 illustrates the depiction of what a successfully learned task looks like. We can see that the robot arm successfully gripped the cube and picked it up.

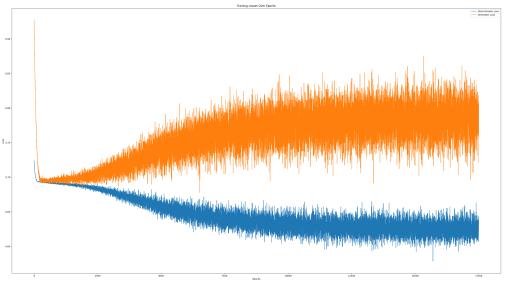


Figure 11. GAIL training loss. The orange line represents the generator loss, and the blue line represents the discriminator loss.

MLP seems to struggle to learn an effective policy with limited demonstrations. Initially, we were planning to collect more data using a policy trained on RL acting as an expert demonstrator. Although diffusion policy showed more promising results with a limited dataset hence we focused on that instead.

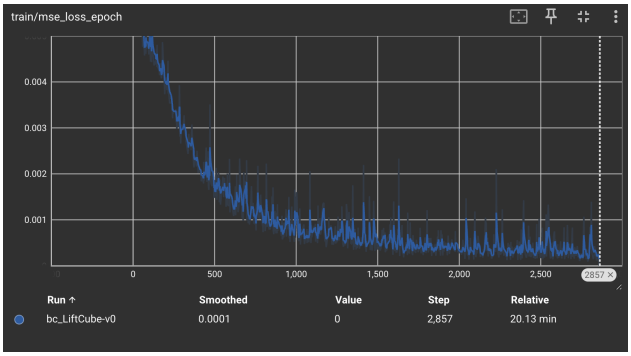


Figure 12. Loss value in function of the number of iterations

7. Team Contributions

Team Member	Contribution Summary
Sreeranj Jayadevan	Maniskill 2 and 3 environment setup. Generating baselines using RL PPO for different manipulation tasks. Training and evaluating success rates LiftCube, PickCube, PushCube and PickCube tasks. Training and evaluating transfer learning across tasks using RL PPO
Nadira Mahamane	Maniskill 2 and 3 environment setup. Configuring the code to execute on a local GPU cluster. Training and evaluating success rates of LiftCube/PushCube using Imitation Learning and generating baselines for the same. Diffusion policy training and evaluation.
Abivishaq Balasubramanian	Maniskill 2 and 3 environment setup. Configuring the code to execute on a local GPU cluster. Training and evaluating success rates of PickCube/PushCube using Diffusion Policy and generating baselines. Generating demos and studying impact of dataset size on transfer learning performance.

Table 2. Summary of team contributions.