Appendix

Table of Contents

A Additional Experiments

A.1 Real World: How much data is necessary?

Given SpawnNet's success in the real-world experiments with little data, we aim to see how well our method generalizes given differing amounts of demonstrations. To do so, we additionally evaluate SpawnNet+d with 30, 60, and all 90 demonstrations. For comparison, we additionally include LfS+aug+d with 90 demos; results are reported in Table 7.

SpawnNet+d performs surprisingly well with few demonstrations; it approaches the performance of the LfS+aug+d baseline with fewer than a third of the training demonstrations and exceeds it with two-thirds, demonstrating the effectiveness of dense features even with few demonstrations.

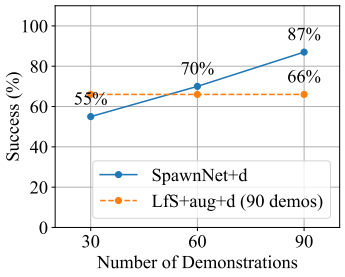

Figure 7: Comparing the number of training demos on *Place Bag*. With as few as 60 demos, SpawnNet exceeds LfS+aug+d with 90 demos.

B Architecture Design Choices

In this section, we describe specific choices in the architectures that we evaluate in our experiments.

Encoders. When comparing pretrained encoders, we control for similar parameter counts. MVP and DINO both use ViT-S (22M parameters), which has 12 layers. R3M uses ResNet-50 (23M parameters). For all encoders, we process each frame individually with the encoder, and concatenate representations across stacked frames and views before passing it into the MLP.

We further expand on our learned encoder architectures:

- **Learning-from-scratch architecture**: Our LfS architecture follows the deep convolutional encoder described in [33], with 128-channel 3x3 convolutions and 128-channel residual layers in each block. We detail this architecture in Figure 8.
- **SpawnNet**: For SpawnNet, we use ViT-S/8 with a stride of 8, resulting in spatial attention features of shape [384, 28, 28]. SpawnNet uses three adapters, taking pre-trained features from the 6^{th}, 9^{th}, and 12^{th} layers of the vision transformers respectively and mapping them to 64 channels (see Section 3.2) Each adapted 64-channel feature is then concatenated to the current learned 64-channel feature before a 128-channel residual block.

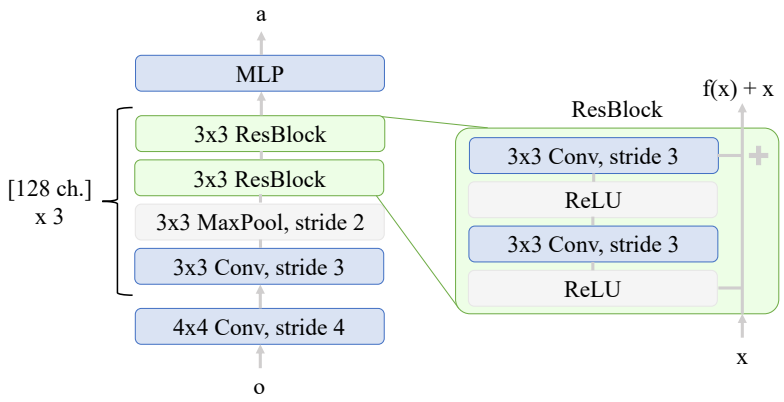

Figure 8: The convolutional encoder we consider for the LfS baseline. The initial 4x4 convolution transforms the input from its initial channel dimension to 128.

MLP. We parameterize the MLP for all encoders using the same architecture, with hidden layers of size [256, 128]. The feature vector extracted from the encoder is flattened first before being passed in.

Model Sizes and Inference Time. We report the number of trainable parameters and inference time for models trained on xArm tasks in Table 4. We note that *Inference* is the real-time inference speed; *Cached Inference* is the time taken for a forward pass with pre-calculated features (i.e. for training). Our LfS baseline has similar numbers of trainable parameters and cached inference speed as SpawnNet. Additionally, a SpawnNet backbone has approximately the same inference speed as a frozen pre-trained backbone.

Model	Trainable Params (M)	Inference (ms)	Cached Inference (ms)
DINO	0.84	57.27	0.27
Spawn-DINO	14.86	59.17	2.91
R3M	4.25	11.64	0.16
Spawn-R3M	15.02	14.15	2.83
LfS	15.11	2.47	2.47

Table 4: Inference times for different models. The increase in parameters between pre-trained and SpawnNet is from the use of dense spatial features instead of the CLS token.

Data Augmentations: Following Hansen et al. [14], we consider random shift and random color jitter data augmentations. For simulation tasks, we only apply data augmentation to LfS+aug with $p_{aug} = 0.5$. For real tasks, we apply data augmentation to all methods with $p_{aug} = 0.5$. We provide psuedocode for our implementations below:

```
import torchvision.transforms as T

sim_aug = T.Compose([ # random shift
    T.Pad(5, padding_mode='edge'),
    T.RandomResizedCrop(size=224, scale=(0.7, 1.0))])

real_aug = T.Compose([ # random shift (no pad) and color jitter
    T.RandomResizedCrop(size=224, scale=(0.7, 1.0)),
    T.ColorJitter(brightness=0.3)])
```

C Details on Real World Tasks

We provide more details about the real-world tasks, including the total number of demos, the breakdown of demos per instance, and further details about the experimental setup.

Place Bag: 102 total demonstrations, with 34 demonstrations split across a red, black, and brown bag. Bags are placed in front of the robot, with variations in the x-y position (within a 1'x1' box) and the rotation (within a 90 degree range). The stand is kept fixed on the table. The table itself translates up and down within a 3" range, adding height variation as well.

Hang Hat: 95 total demonstrations, with 30 demonstrations on a teal hat, 33 demonstrations on a black hat, and 32 on a navy hat. Demonstrations grab above the bill of the hat, and hats are placed on a fixed stand with varying rotation within a 90 degree range. The table height remains fixed.

Tidy Tools: 90 total demonstrations, with 15 per drawer. We define a drawer as a level on the shelf, and leave some "levels" as novel instances; this tests the policy's ability to generalize learned features spatially. We additionally vary the tool being manipulated between two different handled tools, and split these with 45 total demonstrations per tool across 6 different drawers. Tools are placed with a rotation within a 90 degree range inside of a 5"x5" box. Different drawers are placed with a rotation within a 45 degree range inside of a 5"x5" box. The table height remains fixed.

Evaluation. We place the novel instances within the training instances' pose variations as described above. Following [13], we additionally award partial credit for tasks which consist of multiple manipulation skills; for example, in the *Hang Hat* task, if the policy grasps the hat but is unable to hang it, we count the grasp as a success and the hang as a failure for a score of 0.5. The success rates are reported as the average success rate per instance. We perform 5 rollouts for each instance.

D Experimental Results

We produce numeric tables for all experiments presented.

D.1 Numerical Results for Simulation Experiments

The results in Table 5 correspond to the analysis presented in Section 4.1.

Method	**Open Door**		**Open Drawer**	
	Train	Val	Train	Val
SpawnNet	87.3 ± 1.6	58.3 ± 5.1	85.7 ± 2.8	61.1 ± 3.2
LfS+aug	90.5 ± 2.8	53.3 ± 2.4	81.0 ± 2.8	61.7 ± 2.4
R3M	92.1 ± 4.2	42.5 ± 5.1	82.5 ± 3.2	50.3 ± 1.8
MVP	68.3 ± 4.2	43.8 ± 4.7	58.7 ± 4.2	30.0 ± 1.6
PointCloudRL (expert)	89.5 ± 0.0	14.4 ± 0.0	95.2 ± 0.0	65.8 ± 0.0

Table 5: Numerical results for the simulation experiments.

D.2 Real Experiments

The results in Table 6 correspond to the analysis presented in Section 4.2.

Method	**Place Bag**		**Hang Hat**		**Tidy Tools**	
	Train	Val	Train	Val	Train	Val
SpawnNet+d	93.3 ± 3.8	86.7 ± 6.4	100 ± 0.0	80.0 ± 8.5	91.1 ± 3.6	88.6 ± 4.4
SpawnNet	83.3 ± 5.6	76.7 ± 10.8	93.3 ± 6.1	78.3 ± 6.4	90.0 ± 2.4	74.3 ± 6.3
LfS+aug+d	93.3 ± 4.2	66.7 ± 7.6	76.7 ± 15.2	60.0 ± 10.5	44.4 ± 9.4	33.3 ± 8.4
R3M	93.3 ± 3.8	20.0 ± 7.1	66.7 ± 13.9	20.0 ± 10.8	38.9 ± 7.4	15.2 ± 4.8

Table 6: Numerical performance on the real world tasks. We report the average of the total success rate for each instance. The bar denotes standard error.

E Pretrained Feature Visualization

Following the results presented in Figure 6, we present more examples of the learned features from the adapter layers. More visualizations can also be found on our project website.

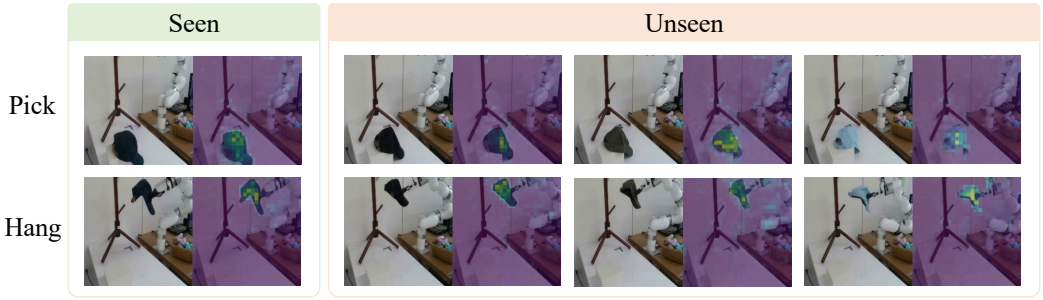

Figure 9: Visualized adapter features for the *Hang Hat* task. When grasping the hat, the adapter highlights relevant parts of the hat, such as the brim and front. When hanging the hat, the adapter highlights relevant parts of the hat, such as the back edge.

Figure 10: Visualized adapter features for the *Tidy Tools* task. When grasping the tool, the adapter highlights its handle, even with novel drawers in the background. When placing the tool in the drawer, the adapter highlights the drawer's front edge.