Appendix

A Implementation Details

Codebase. Our major codebase is built upon the official implementation of RRL [25], which is publicly available on https://github.com/facebookresearch/RRL and includes the Adroit manipulation tasks [22]. The DexMV [21] tasks are from the official code https://github.com/yzqin/dexmv-sim. All the visual representations in our work are also available online, including RRL (pre-trained ResNet-50, provided in PyTorch officially), R3M (https://github.com/facebookresearch/r3m), MVP (https://github.com/ir413/mvp), VC-1 (https://github.com/facebookresearch/eai-vc), and FrankMocap (https://github.com/facebookresearch/frankmocap). This ensures the good reproducibility of our work. *We are also committed to releasing the code.*

Network architecture for H-InDex. The architecture employed by H-InDex is based on ResNet-50 [9], referred to as h_θ. In the initial stage (Stage 1), h_θ takes as input a 224×224 RGB image and processes it to generate a compact vector of size 2048. In Stage 2, we modify h_θ by removing the average pooling layer in the final layer, resulting in the image being decoded into a feature map with dimensions $7 \times 7 \times 2048$. Moving on to Stage 3, h_θ once again produces a compact vector of size 2048, while simultaneously updating the statistics within the BatchNorm layers using the exponential moving average operation.

Implementation details for Stage 2. Our implementation strictly follows the previous work that also uses the self-supervised keypoint detection as objective [10,13,14]. We give a PyTorch-style overview of the learning pipeline below and refer to [10] for more implementation details. Notably, the visual representation h_θ (24M) contains the majority of parameters, while all other modules in the pipeline maintain a parameter count ranging from 1M to 3M. We use 50 demonstration videos as training data for each task and train 100k iterations to ensure convergence with learning rate 1×10^{-4}. One of our core contributions is to only adapt the parameters in BatchNorm layers in h_θ, and we emphasize that the learning objective is not our contribution, as it has been well explored in [10, 13, 14].

```
for _ in range(num_iters):
    # sample data
    source_view, target_view = next(data_iter) # 3x224x224

    # self-supervised keypoint-based reconstruction
    # h_theta is our visual representation
    feature_map = h_theta(target_view) # -> 7x7x2048
    keypoint_feat = keypoint_encoder(feature_map) # -> 30x56x56
    keypoint_feat = up_sampler(keypoint_feature) # -> 256x28x28
    apperance_feat = apperance_encoder(source_view) # -> 256x28x28
    target_view_recon = image_decoder([keypoint_feat,apperance_feat]) # -> 3x224x224

    # compute loss
    loss = perceptual_loss(target_view, target_view_recon)

    # compute gradient and update model
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

B Task Descriptions

In this section, we briefly introduce our tasks. We use an Adroit dexterous hand for manipulation tasks. The task design follows Adroit [22] and DexMV [21]. Visualizations of task trajectories are available at h-index-rl.github.io.

Hammer (Adroit). It requires the robot hand to pick up the hammer on the table and use the hammer to hit the nail.

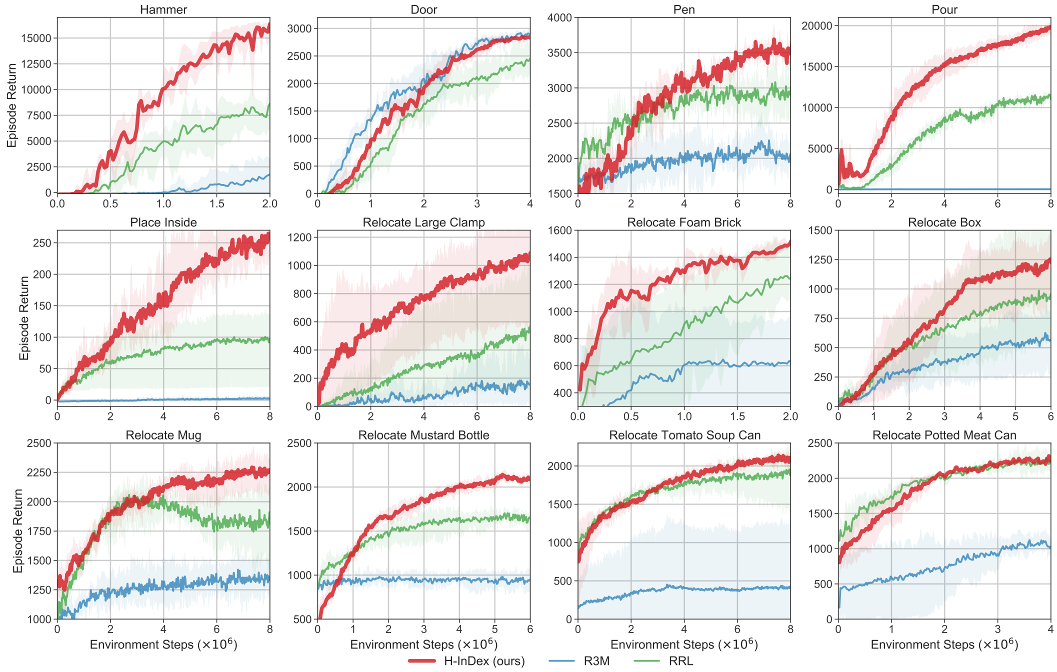

Figure 9: Episode return for **12** challenging dexterous manipulation tasks. Mean of 3 seeds with seed number $0, 1, 2$. Shaded area indicates 95% CIs.

Door (Adroit). It requires the robot hand to open the door on the table.

Pen (Adroit). It requires the robot hand to orient the pen to the target orientation.

Pour (DexMV). It requires the robot hand to reach the mug and pour the particles inside into a container.

Place inside (DexMV). It requires the robot hand to place the object on the table into the mug.

Relocate YCB objects [2] (DexMV). It requires the robot hand to pick up the object on the table to the target location. The objects in our tasks include *foam brick*, *box*, *mug*, *mustard bottle*, *tomato soup can*, and *potted meat can*.

C Main Experiments (ConvNets Only)

In our primary experimental analysis, we conduct a comprehensive comparison of five visual representations, with three of them being ConvNets, including our method. Figure 9 presents an isolated demonstration of the comparison among the ConvNets. Notably, our method H-InDex exhibits superior performance in comparison to the other ConvNets.

D Success Rates in Main Experiments

We present the success rates of our six task categories as in Table 1. Regarding the `hammer` task, it is evident that both H-InDex and VC-1 exhibit success rates near 100%. However, a notable disparity arises when considering episode returns, indicating the varying degrees of task execution proficiency even among successful agents.

Table 1: **Success rates for main experiments.** Highest success rates for each task are marked with **bold** fonts.

Task name / Method	RRL [25]	R3M [17]	MVP [29]	VC-1 [16]	H-InDex
Hammer	$89_{\pm 15}$	$24_{\pm 21}$	$83_{\pm 11}$	$97_{\pm 3}$	$\mathbf{100}_{\pm 0}$
Door	$92_{\pm 1}$	$99_{\pm 2}$	$\mathbf{100}_{\pm 0}$	$99_{\pm 2}$	$96_{\pm 5}$
Pen	$78_{\pm 4}$	$58_{\pm 6}$	$80_{\pm 4}$	$81_{\pm 2}$	$\mathbf{90}_{\pm 2}$
Pour	$38_{\pm 33}$	$0_{\pm 0}$	$23_{\pm 38}$	$67_{\pm 29}$	$\mathbf{99}_{\pm 2}$
Place inside	$68_{\pm 48}$	$2_{\pm 3}$	$97_{\pm 4}$	$\mathbf{99}_{\pm 1}$	$\mathbf{99}_{\pm 3}$
Relocate box	$85_{\pm 14}$	$45_{\pm 24}$	$48_{\pm 50}$	$49_{\pm 50}$	$\mathbf{94}_{\pm 5}$

436 # E Hyperparameters

We categorize hyperparameters into task-specific ones (Table 2) and task-agnostic ones (Table 3), Across all baselines, all the hyperparameters are shared except the momentum m, which is only used in our algorithm. All the hyperparameters for policy learning are the same as RRL [25]. This ensures the comparison between different representations is fair.

Our exploration of the momentum m in Table 2 has been limited to a specific set of values, namely $\{0, 0.1, 0.01, 0.001\}$, through the use of a grid search technique, due to the limitation on computation resources. It is observed that carefully tuning m could take more benefits.

Table 2: **Task-specific hyperparameters.**

Task name / Variable	Momentum m	Demonstrations	Training steps (M)	Episode length
Hammer	0.1	25	2	200
Door	0.0	25	4	200
Pen	0.0	25	6	100
Pour	0.0	50	8	200
Place inside	0.001	50	8	200
Relocate large clamp	0.01	50	8	100
Relocate foam brick	0.01	25	2	100
Relocate box	0.001	25	6	100
Relocate mug	0.0	25	8	100
Relocate mustard bottle	0.001	25	6	100
Relocate tomato soup can	0.01	25	8	100
Relocate potted meat can	0.0	25	4	100

Table 3: **Task-agnostic hyperparameters.**

Variable	Value
Dimension of image observations	$224 \times 224 \times 3$
Dimension of robot states	30
Dimension of actions	30
Hidden dimensions of policy π	$256, 256$
BC learning rate	0.001
BC epochs	5
BC batch size	32
RL learning rate	0.001
Number of trajectories for one step	100
VF batch size	64
VF epochs	2
RL step size	0.05
RL gamma	0.995
RL gae	0.97