A Appendix

We encourage the reader to refer to our website at the url: https://sites.google.com/view/lirf-corl-2022/ for additional video visualizations. Refer to https://github.com/penn-pal-lab/lirf_public for code release.

A.1 LIRF+Verify Approach Details

In Section 3.2, we proposed to use IRFs as verification mechanisms for introspective behaviors. Here we provide a detailed pseudocode for our LIRF+Verify approach.

Algorithm 2 LIRF+Verify Pseudocode

Require: LIRF policy π_T, IRF policy π_R, single-observation reward $D(o) \to [0, 1]$
1: $i = 0$
2: Rollout π_T in the task POMDP.
3: Keep the world state, switch to IRF POMDP and rollout π_R. The final observation is o^*.
4: $i{+}{+}$
5: **if** $D(o^*) > 0.5$ or $i \geq 10$ **then**
6: Terminate
7: **else**
8: Keep the world state, repeat from 1.
9: **end if**

A.2 Environment Details

In Section 5, we described the three tasks we use in our experiments. Here we provide supplementary minor details for each task. Refer to our code release for reproducibility.

Door Locking: During initialization, the orientation of the latch θ is randomized from $\theta \sim (90°, 180°)$, with $0°$ pointing outwards from the door, $90°$ pointing upwards. The door is firmly locked when $\theta \in (-40°, 40°)$. The latch can freely rotate without joint limits, so that it is possible for a policy to overturn the latch. An extra fixed handle is provided for the IRF policy so that it will not affect the state of the latch when trying to open to door. The control of the Adroit hand outputs 5-D actions (x, y, z, roll, and close fingers).

Weighted Block Stacking: In this environment, three visually identical blocks are presented to the Fetch robot, with one block significantly heavier than the other two. The task phase is executed by a Fetch robot with a two-fingered gripper, and the IRF phase is executed by a Fetch robot with an end-effector more suitable for horizontal poking actions.

Motion primitives: To expedite the RL training for the task policies and the IRF policies, we employ a "pick-and-place" motion primitive and a "poking" motion primitive respectively in the weighted block stacking task.

The task policies output a 4-dimensional action for each timestep, in which the first 2 dimensions define the x and y position in the world coordinate for where to pick up the block, and the last 2 dimensions correspond to the x and y position for where to place the block, assuming that a block is grasped. We predefine separate z heights for the "pick-up" actions and for the "place" actions. For visualizations of the "pick-and-place" primitive, refer to the supplemental videos.

To examine if a block tower is built in the correct order, i.e. the heavy block is put at the bottom, the robot pokes the tower to apply some disturbance and observes whether the block tower falls over. We modify the end-effector of Fetch to enable such poking actions as shown in Figure 7. The IRF policies output a 3-dimensional action (poking height, direction, magnitude) for each timestep. See supplemental slides for more poking actions.

Screwing: The 4-prong valve is connected to a motor underneath it. The valve can rotate freely without limits when the motor is not engaged, whereas it is locked in place or controllable through the motor when the motor is engaged.

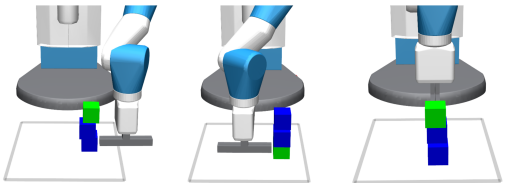

Figure 7: Poking primitives for π_R.

A.3 Metrics

In the main paper, we mainly use success rates to evaluate and compare all the policies. In this section, we describe how success rates are defined.

Success rates for the task policies are measured based on ground truth state. Successful door locking corresponds to the latch orientation $\theta \in (-40°, 40°)$, where it effectively locks the door. Successful block stacking corresponds to a standing tower of three blocks, with the heaviest block at the bottom. Successful screwing corresponds to the valve reaching the "tightened" state.

When an IRF policy is executed starting from an initial partially observed state σ, its performance is reported based on its ability to classify σ. Concretely, for a state σ, we first execute π_R to generate the new environment state s^*, which in turn generates a new observation o^*. Then we classify this observation using the single-observation reward classifier $D(o^*)$, and compare the result with the ground truth state-based success / failure label for σ.

A.4 Baselines

In this section, we provide additional details for the baselines described in Section 5.

A.4.1 Manual IRF Baseline Details

To demonstrate that the IRF policies are non-trivial to learn, we also compare our method against task policies trained against manual IRF policies, in which we manually engineer IRF policies to distinguish between goal states and failure states. Since it is difficult to engineer a motion primitive for the Adroit hand to open the door, we only evaluate the "Manual π_R" baseline on the weighted blocking task and the real-robot screwing task.

Screwing Here, we program a simple hard-coded movement of the hand to perturb the valve.

Weighted Block Stacking For this setting, we grid search for the best poking length, i.e. the one that best separates stable and unstable stacks, while using random poking heights and poking directions. The random π_R classification accuracy vs. poking length is shown in 8. We choose the poking length with the highest classification accuracy (2.5 cm) as our manual π_R.

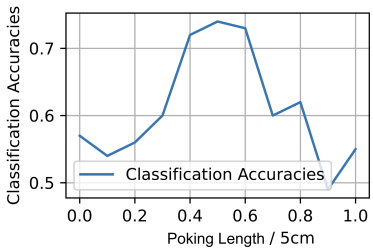

Figure 8: The performance of random π_R vs. π_R poking magnitude in weighted block stacking.

A.4.2 GAIfO Baseline Details

Where do expert demonstrations come from for the GAIfO imitation approach [46] used in our comparisons? For both door locking and real-robot screwing, we collect observation-only expert

trajectories as demonstrations from policies trained on GT-state observation (without aliasing) against GT reward.

For weighted block stacking, we hand-code an expert that always places the heavier block at the bottom, and places the other two blocks on top of the heavier block to build a block tower. The observations do not contain the block weights, to mimic a purely visual observation. However, for thoroughness, we also perform additional experiments when demonstration observations include block weight information for blocks that have been picked up within the demonstration. Naturally, GAIfO benefits from this, and its success rate improves from 0.275 to 0.780, still lower than our LIRF approaches.

A.4.3 GT State Reward Baseline Details

For all three tasks, the GT State Reward Baseline policies are trained with the same observation as LIRF polices against sparse rewards given at success.

A.5 LIRF Task Policy Training: Implementation Details

Here we provide additional implementation details for the LIRF task policy described in Section 5.

We use fully-connected MLPs with three hidden layers of 128 units and ReLU nonlinearities for all policies, critics and reward functions in the door locking experiment and real-robot screwing experiment.

Weighted Block Stacking To mimic visual observations of the identical blocks, where block identities would not be visible before interaction, we employ a permutation-invariant network (Deep Sets [47]) for both policies and VICE success classifiers, so that

$$\pi([\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2]) = \pi([\mathbf{x}_{perm(0)}, \mathbf{x}_{perm(1)}, \mathbf{x}_{perm(2)}]) \tag{3}$$

where $\mathbf{x}_i$ represents the 3D position and weight (if known) of block i, $perm(i)$ is a permutation function of i. Our neural networks for all policies, critics and reward functions contain the followings: two hidden layers of 128 units and ReLU nonlinearities, then one AvgPool1d along the object dimension, finally two layers of 128 units and ReLU nonlinearities. The weight for each block is initialized as -1. When a block is picked up, the weight for it becomes 1 if it is the heavier block, 0 otherwise.

A.6 Ablation Study on λ

We also carry out ablation studies on the hyper-parameter λ, which is the weight for the IRF reward, in the door locking setting. The results are shown in Table 3. As the IRF reward is only provided sparsely for the terminal state of the task policy, and the original single-observation reward $D(o_t)$ is provided for each step in the episode, we need λ to be large enough in order to have a substantial effect during LIRF training. Beyond that, the performance of LIRF is not very sensitive to λ.

LIRF Policy	Door Locking Success Rate
$\lambda = 10$	0.32
$\lambda = 100$	0.54
$\lambda = 1000$	0.64
$\lambda = 10000$	0.62

Table 3: Door locking success rates of LIRF policies trained with different λ.

A.7 IRF Policy Training Efficiency

In Section 5, we discussed the number of positive examples that is required for IRF policy training. Here we provide more details for IRF policy training efficiency.

To evaluate the efficiency of our example-based interactive reward function training, we record the number of positive examples that are required for training an accurate IRF policy. Figure 9 shows the training curve of π_R classification accuracy vs. number of positive examples given. The IRF policy

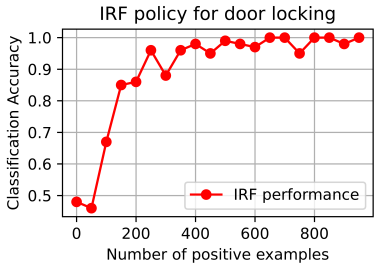

Figure 9: π_R classification accuracy vs. number of positive examples given in the door locking task.

is able to reach higher than 90% classification accuracy with fewer than 300 positive examples in the door locking task.

The performance of IRF converges after 250, 10k and 100 episodes with positive examples for door locking, weighted block stacking, and screwing tasks respectively. Note that since actionable positive examples are reusable across IRF training episodes unless they are destroyed, we observe that these three tasks use 110, 7k, and 1 actionable positive example respectively. Finally, we use a balanced sampling strategy, so the number of episodes with negative examples is roughly equal to the number of episodes with positive examples. Negative actionable examples for training the IRF policy are simply the outcomes of the current task policy, generated on-the-fly for each training episode.

A.8 Can the GT state reward baseline also benefit from multiple retries?

We showed in Section 5 Table 1 that LIRF verification policies can be used not just to train task policies but also to improve test-time task performance through identifying when the task policy has failed and should retry. The ground truth (GT) state reward baseline in the main paper doesn't directly permit retries since rewards in RL are typically assumed available only at training time, and the task policy for this baseline is assumed to only have access to observation histories. To understand how retries affect the performance of the GT state reward baseline policies, we evaluate two alternative approaches:

1. First, we run the GT state reward baseline with increased episode length so that its total duration matches the K retries the LIRF policy gets.
2. Second, we provide ground truth state reward access to check for task success instead of our IRF reward policy. This GT state reward policy with GT state reward checking would correspond to an "oracle" version of our method.

The results for GT state reward baselines with multiple retries are shown in Table 4. These experimental results align with our expectation that, without GT state success checking, the performance of GT state reward baselines does not improve with more retries since they tend to overturn the door latch. With GT state success checking, the GT state reward policies with multiple retries stop right after they successfully lock the door, achieving better performance, as they serve as an "oracle" version of our method.

Policy	Door Locking Success Rate
GT state reward without retries	0.71
GT state reward with 3 retries	0.71
GT state reward with 10 retries	0.60
GT state reward with 3 retries and GT state success checking	0.91
GT state reward with 10 retries and GT state success checking	0.98
LIRF (Ours)	0.64
LIRF+Verify (Ours)	0.96

Table 4: Door locking success rates of GT state reward baselines with multiple retries.

A.9 LIRF When State is Fully Observed: Door Closing Task

In the main paper, we evaluated LIRF in partially observed state-aliased settings where naive baselines like VICE fail. For example, for our door locking task, the visual appearance of the locked door is

indistinguishable from that of a closed but unlocked door. As such, VICE would learn policies that merely close the door.

To evaluate LIRF in a fully observed setting where VICE already achieves near-perfect performance, we select the door *closing* task. From Table 5, We can see that the VICE policy achieves near-perfect performance on the door closing task as it is fully-observed and relatively easy to learn, but the performance of LIRF does not degrade under such circumstances.

Policy	Door Closing Success Rate
VICE	1.0
LIRF	0.99

Table 5: Door closing success rates of VICE and LIRF.

A.10 Positive Examples and Scalability

In the limitations section, we pointed out that there are cases in which the number of actionable positive examples required by LIRF is large, and gathering and storing them is non-trivial. We now discuss scalability in more detail.

First, what causes the number of required actionable positive examples to be high or low? In the weighted block stacking task, the desired task outcome, a well-constructed stack of blocks, is not objectively a very stable object configuration (even if it is more stable than poorly constructed stacks) — in particular, relatively small perturbations can destroy it. As a result, we use around 7k positive examples, over 10k IRF training episodes (the remaining 3k episodes don't result in the destruction of a positive example so that we can reuse the examples provided). Further, it takes a long time for the interactive reward function (IRF) to learn the nuanced poking behavior that can accurately separate "good" block stacks from "bad", to then produce a good task policy. The IRF policy must apply enough force to the tower at the right locations to reveal the weights of the blocks, but be gentle enough to avoid toppling the tower. At the other extreme, in the screwing task, we required only one single positive example of a tightened screw, over 100 IRF training episodes with positive samples: here, the perturbation behavior to separate loose and tight screws is very straight-forward to learn, and the positive examples are not easily destroyed.

Second, how should we think about the cost of training LIRF policies in these terms? When the IRF policy does not destroy a positive example, it can be reused. For example, in the screwing task, technically only one single positive example of a tightened screw is required because the positive example physically cannot be destroyed (undoing a tight screw) by the robot during training, so that this one tightened screw can be reused for 100 times. Note that in the paper, we reported 250, 10k and 100 trials with positive samples for training IRF policies for door locking, weighted block stacking, and screwing tasks respectively. However, if we take reusability of positive examples into consideration, the IRF training only requires 110, 7k, and 1 positive examples.

Finally, how could LIRF be encouraged to learn from fewer positive examples? Here, we speculate that explicitly penalizing the destruction of positive examples and/or explicitly encouraging the verification policy to generate reversible behaviors will help. Further, we have used off-the-shelf policy learning approaches (SAC) without optimizing for sample efficiency, and there may be significant gains from using alternative, more sample-efficient approaches, such as from the model-based reinforcement learning literature.