Appendix A: Experiment Details

A.1 Task Setup

Here, we describe the inputs and outputs of our environment. Actions are executed at 20Hz and consist of a 10-dimensional command $[a_{base}, a_{arm}]$, where $a_{base} = [v_{lin}, v_{ang}] \in (-1, 1)^2$ is the differential drive velocity commands for the base; and $a_{arm} = [dx, dy, dz, da_x, da_y, da_z, grasp, reset] \in (-1, 1)^8$, where $[dx, dy, dz, da_x, da_y, da_z]$ is the 6DOF normalized delta commands in Cartesian space passed to an IK controller, $grasp > 0$ commands the gripper to close, and $reset > 0$ overrides the 6DOF delta command and instead moves the arm towards a pre-defined default pose. Observations consist of the head camera and wrist camera RGB-D frames (90 x 120), and the base laser scanner's point cloud (1 x 100).

A.2 Error Detection Architecture

Our error detector $\Phi(s, s_g)$ consists of a conditional variational auto-encoder (cVAE): $E_\phi(s_g, s), D_\phi(z, s)$, where encoder E_ϕ maps the current state s, and the goal state s_g that is H steps into the future, into a gaussian mixture model (GMM) latent space distribution z GMM$(\mu_1, ..., \mu_n, \sigma_1, ..., \sigma_n)$. The decoder D_ϕ then tries to reconstruct goal state s_g from the sampled latent embedding z conditioned on current state s_t.

There are two reasons why we choose this specific architecture. Reconstructing future goals conditioned on the current state enables our error detector to leverage temporal information to better inform its error prediction by implicitly coupling the likelihood of the future to the observations in the present. Secondly, using a GMM as the encoder's prior can be useful in MM and other long-horizon settings where demonstrations can be diverse and result in more complex state distributions that need to be modeled.

The cVAE is trained using a weighted combination of KL Divergence loss to regularize the learned GMM distribution and reconstruction error ϵ to encourage salient encoded distributions. With sufficient training data and capacity, the cVAE Φ should learn to reconstruct similar states to those observed from the demonstrations with low ϵ. We leverage ϵ for distinguishing between in- and out-of-distribution states: abnormally high ϵ likely correspond to unseen states and can be interpreted as failure modes during rollouts.

Algorithm 1: Error Detection

```
Result: Success Rate $S$
Input: $\Phi$, $\psi$, $K$, $H$, $\pi$, $T$, $O$, $\rho_0$, $\xi_0$
Initialize $t, k, S \leftarrow 0$, $s \sim \rho_0$, $o \sim \xi_0$,
  $\beta \leftarrow \emptyset$
while True do
    if success then
        $S \leftarrow 1$;
        terminate();
    end
    append($\beta, o$);
    if $t > H$ then
        $\epsilon = \Phi(\beta[t - H], \beta[t])$;
        if $\epsilon > \psi$ then
            $k += 1$;
            if $k == K$ then
                terminate();
            else
                $\epsilon =$ recover();
                if $\epsilon > \psi$ then
                    terminate();
                end
            end
        end
    end
    $a \sim \pi(o)$;
    $s \sim T(\cdot|s, a)$;
    $o \sim O(s, a)$;
end
```

While we have immediate access to all states at train time, we cannot directly apply this method during rollouts because we do not know a priori the future goal states s_g. Instead, we apply our error detector in the backwards direction: at timestep $t > H$, having recorded the prior H states in replay buffer β, we run the forward pass through our cVAE's encoder E_ϕ and decoder D_ϕ with $s = s_{t-H}$ and $s_g = s_t$. In this way, the model peers retroactively into the past and considers the plausibility of the current state as a future goal from the perspective of the past.

A.3 Hyperparameters

In Table A.1, we present the hyperparameters used for training our policy and error detector, selected from a small-scale hyperparameter sweep on a single task. Note that these values were then used uniformly across all tasks.

Table A.1: **Algorithm Hyperparameters:** We show selected hyperparameters used for our models during experiments. BC-TieredRNN and BC-RNN parameters were chosen such that the total parameter count was roughly the same. ReLMoGen parameters were chosen to match those presented in [45].

Algorithm	Hyperparameter	Value
BC-TieredRNN	LR	$1e-4$
	Sequence Length	50
	GMM Prior	True
	N GMM Modes	5
	RNN Horizons	$[50, 5]$
	RNN Strides	$[1, 10]$
	RNN Tier Hidden Dims	$[1000, 400]$
	RNN Tier z_i Dim	32
	Actor MLP Dims	$[300, 400]$
BC-RNN	LR	$1e-4$
	Sequence Length	50
	GMM Prior	True
	GMM Num Modes	5
	RNN Horizons	50
	RNN Tier Hidden Dims	1200
	Actor MLP Dims	$[300, 400]$

Algorithm	Hyperparameter	Value
ReLMoGen	Actor LR	$1e-4$
	Sequence Length	50
	GMM Prior	True
	N GMM Modes	5
	RNN Horizons	$[50, 5]$
	RNN Strides	$[1, 10]$
	RNN Tier Hidden Dims	$[1000, 400]$
	RNN Tier z_i Dim	32
	Actor MLP Dims	$[300, 400]$
Error Detector	LR	$1e-3$
	Sequence Length	10
	KL Weight	$1e-5$
	VAE GMM Prior	True
	VAE GMM Latent Dim	16
	VAE GMM Num Modes	10
	VAE Encoder MLP Dims	$[300, 400]$
	VAE Decoder MLP Dims	$[300, 400]$
	VAE Prior MLP Dims	$[128, 128]$
	ψ	0.05

A.4 MoMaRT Interface

In this section, we describe the specific details of our MoMaRT interface. A visual overview of our interface is shown in Fig. A.1. By default, To allow flexibility while minimizing operator overload, we leveragea multi-view interface. By default, operators are presented with the manipulation view (Fig. A.1 *left*), allowing them to toggle grasping or reset the arm configuration by pressing the large, easy-to-tap buttons. To navigate, the user can hold down the virtual joystick, which will transform the view into navigation mode (Fig. A.1 *right*), allowing them to control the robot's base. As with the original RoboTurk interface, the robot's end-effector is controlled by simply moving the smartphone; this motion is tracked and converted into corresponding 6DOF commands for the end-effector.

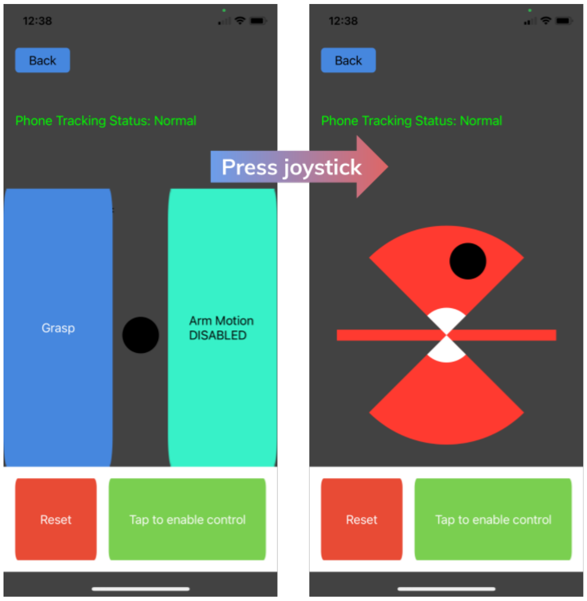

Figure A.1: MoMaRT interface showcasing the user interface. Users by default are presented with the manipulation view (*left*), allowing them to reset the arm or toggle grasping. By holding down the virtual joystick, the navigation view (*right*) immediately appears, allowing the user to drag the virtual joystick to navigate the robot base accordingly. The hourglass section allows for a combination of linear and rotational velocity, and the horizontal section allows for pure rotation.

By using this multi-view interface, users have the ability to navigate and control the arm at the same time without being overloading by the amount of content on the screen at a given time. Furthermore, an hourglass joystick interface was chosen instead of a full 360 degree joystick so that the mobile manipulator can also be allowed to turn in place at variable speed, enabling flexibility for the user to either slowly turn in tight corners or quickly rotate. Because our virtual joystick lacks haptic feedback, we create a dead-zone, shown in white (Fig. A.1 *right*), that prevents any acciden-

Statement	Disagree (1) to Agree (5)
The interface was easy to understand and control	3.5 ± 1.1
Except for the occasional stutters, movement between the robot and my arm felt natural	3.3 ± 1.1
I felt in control of the robot as I was teleoperating	3.4 ± 1.2
By the end of the session, I felt confident in my ability to solve the task	3.3 ± 1.5
The automatic reset arm functionality was helpful to use during demonstrations	4.8 ± 0.4

Table A.2: **MoMaRT Study:** After 30 minutes of playtesting, new users found that the interface was comfortable to use, and marginally agreed about the system's ease of usage and controlling the robot. Common feedback cited multiple limitations of teleoperating mobile manipulators, including limited field of view and lack of perceptual feedback for detecting collisions. However, users unanimously agreed that the ability to reset the arm to a stable configuration was helpful, validating our crucial design decision to include this functionality.

tal presses or transitions from triggering the base movement and significantly reduces the noise in navigation.

A.5 MoMaRT User Study

To better understand the qualitative performance of our system, we conducted a small-scale user study conducted over the course of 1 week with the goal of evaluating our platform's efficacy. 10 users who had never used the MoMaRT platform before were given 30 minutes to try the system. Users were placed in a "warmup" kitchen environment to become familiarized with the controls for a minimum of 5 minutes. Once the users felt comfortable with teleoperating, the users were placed into the *Table Setup from Dishwasher* task and asked to solve the task for the remainder of the session. Like all of the 5 core kitchen tasks, this task requires leveraging the full capabilities of the MoMaRT system to manipulate objects and interact with large articulated furniture. Both during and after the session, users were asked to provide both qualitative and quantitative feedback on specific aspects of their experience. The summary of the quantitative results can be seen in Table A.5.

We find that generally, users find the interface easy to learn, but difficult to master. When asked "How long did it take to feel comfortable with the controls?", 70% of users responded with 5 - 15 minutes, 10% with 15 - 30 minutes, and 20% with less than 5 minutes. Users also had generally favorable reviews of the overall usage of the system, and marginally agreed about the system's ease of usage. This suggests that our platform can be quick to learn and easily adapted to by different users.

However, users often had issues controlling the robot. Common feedback cited multiple limitations of teleoperating mobile manipulators, including limited field of view and lack of perceptual feedback for detecting collisions. These comments reflect the difficulty of the MM domain, which inherit these challenges and require additional design decisions that may be unnecessary in SM or navigation. Crucially, nearly all users unanimously agreed that the automatic arm reset functionality included was beneficial to them during task demonstration, validating this key design decision and highlighting the benefits of being able to recover from poor arm configurations during long-horizon execution.

Interestingly, while users generally shared the same feedback, their task performance varied widely. 50% of users were unable to generate a task success within the 30-minute session, while the other 50% were able to generate an average of 3.2 successes, with a corresponding 56% average success rate. We note that most users only used about 15 minutes of their 30-minute session to try to solve the task, and those who were unable to solve the full task were still able to partially solve it. Indeed, many of these users were close to solving the full task by the end, and we would expect them to reach their first success soon if given additional playtest time. This highlights the complex nature of our tasks, which are both long-horizon and require multiple successful interactions that can require careful precision and rotation-heavy movement not often explored in other data-driven works.

While the learning curve can be more steep compared to tabletop manipulation [35], it is promising that 50% of new users with no prior experience with MoMaRT can immediately start generating task successes after less than 30 minutes of task interaction. As a first-of-its kind interface, we hope that future work can build upon this platform to increase its intuitiveness for people. Moreover, despite the challenges faced with this platform, the merits are clear in the ultimate scale of data

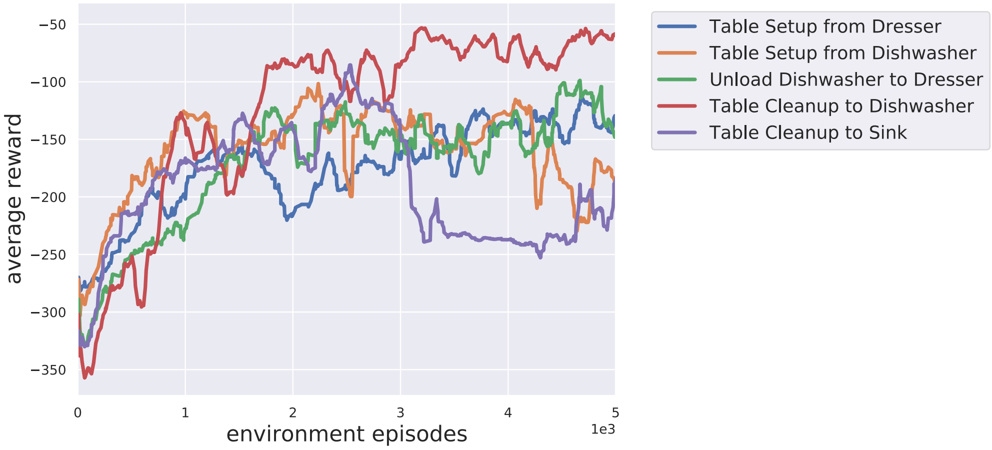

Figure A.2: Training progress for ReLMoGen. For all tasks, the agent is able to learn to get nearer the bowl but not the subsequent manipulation actions needed to complete the task.

and learning results we have been able to produce, and motivate additional research in the area of large-scale datasets for mobile manipulation.

A.6 ReLMoGen Results

We trained the SAC variant of ReLMoGen [45] baseline with a slightly modified version of the original implementation. Since the original action space of ReLMoGen would make it impossible to finish our tasks, given that they require the ability to grasp and to move the end effector in free space, our only modification was to expand the action space. Instead of only choosing between using motion planning for navigation or manipulation, and third scalar output was added to the action space that would allow the agent to use the pose and gripper control space that is used by the imitation learning agents. This makes it possible for the trained agents to accomplish the tasks, while still using the motion-planning based actions from the original model. The need to use this expanded action space is why we used the SAC variant of ReLMoGen instead of the DQN variant, since the latter outputs Q value maps instead of an action vector that is incompatible with the expanded action space.

All tasks were kept the same as they were for imitation learning, except that a reward function was added to enable RL. All tasks have a mix of shaped and sparse reward: there is a negative l2 reward based on the distance of the end effector to the bowl, since all tasks involve grasping the bowl as their initial sub-task, and each further sub-task (grasping the bowl, possibly removing trash from the bowl, and placing the bowl in an appropriate location) results in an addition of 1 per step for the rest of the episode. This reward was chosen since there is no easy way to provide a shaped reward for the subtasks of grasping the bowl, emptying it, or placing it anywhere on a piece of furniture. In all cases, the agent never got beyond the first phase of getting nearer the bowl, indicating the difficulty of learning to grasp or manipulate objects with a sparse reward.

A.7 TieredRNN Details

Here, we provide some further context justifying our choice of TieredRNN over RNN as our final evaluation model that might not be as readily apparent from our core results. We found during preliminary hyperparameter sweeping that the TieredRNN scales better than the RNN model given similar total parameter counts; that is, further increasing the RNN model's parameter count via the hidden dimension size generally resulted in policy degradation. Moreover, we found the TieredRNN model to also provide performance gains on previous iterations of tasks that were not included in this work. These empirical observations led us to deem the TieredRNN valuable enough to include as our final evaluation model, despite providing marginal benefits in this specific setting. We hope to continue iterating on this model in future work.

Task	Metric	Reconstruction Error (ours)	KL Divergence Error	VAE Encoder Variance Mean	VAE Encoder Variance Max	Policy Log Probability
Table Cleanup to Dishwasher	Precision	**96.4 ± 2.6**	86.7 ± 9.4	55.6 ± 19.2	61.1 ± 20.8	85.1 ± 8.2
	Recall	**100.0 ± 0.0**	94.9 ± 3.6	20.6 ± 6.0	40.3 ± 15.0	84.0 ± 6.4
Table Cleanup to Sink	Precision	**97.2 ± 3.9**	63.2 ± 5.6	53.3 ± 5.4	46.7 ± 11.9	64.5 ± 2.6
	Recall	**86.5 ± 10.3**	86.7 ± 9.6	100.0 ± 0.0	100.0 ± 0.0	93.0 ± 9.9
Table Setup from Dresser	Precision	**100.0 ± 0.0**	88.9 ± 15.7	26.7 ± 4.7	26.7 ± 4.7	47.9 ± 23.2
	Recall	**85.9 ± 10.0**	45.6 ± 21.4	100.0 ± 0.0	100.0 ± 0.0	47.2 ± 12.3
Table Setup from Dishwasher	Precision	**88.2 ± 10.2**	93.5 ± 4.9	47.8 ± 11.3	42.2 ± 5.7	55.4 ± 3.7
	Recall	**100.0 ± 0.0**	92.3 ± 10.9	100.0 ± 0.0	100.0 ± 0.0	93.3 ± 6.6
Unload Dishwasher to Dresser	Precision	**85.8 ± 5.3**	93.9 ± 8.6	50.9 ± 6.5	43.5 ± 10.0	69.2 ± 9.5
	Recall	**100.0 ± 0.0**	56.1 ± 11.9	58.3 ± 12.0	26.1 ± 11.6	85.0 ± 2.2

Table A.3: **Error Detector Comparisons:** We consider multiple alternatives to error detection metrics, and evaluate them using our Expert demonstration dataset on all tasks. The best model (bolded) is determined by averaging the error detector's precision and recall for a given task. We find that our method clearly outperforms all other alternatives, and is able to distinguish true errors with high accuracy, while other methods tend to be much more noisy and overly conservative (low precision).

Task	Train via Finetuning (ours)	Train from Scratch
Table Cleanup to Dishwasher	**14.1 ± 3.2**	4.8 ± 1.4
Table Cleanup to Sink	**11.5 ± 6.4**	7.8 ± 4.0
Table Setup from Dresser	**14.4 ± 4.7**	8.1 ± 2.9
Table Setup from Dishwasher	**26.3 ± 1.4**	15.6 ± 4.0
Unload Dishwasher to Dresser	**13.3 ± 2.4**	8.9 ± 2.7

Table A.4: **Few-Shot Generalization Ablation Study:** In the domain shift setting, pretraining our IL models on our original expert data and then finetuning using few-shot demonstrations shows better policy performance across all tasks compared to solely training from scratch using the few-shot demonstrations.

A.8 Error Detector Comparison

To better understand the relative performance of our error detection method, we evaluate multiple error metric alternatives. In addition to our main reconstruction loss metric, we consider another loss metric (KL Divergence loss for the VAE model), latent metrics (VAE Encoder Variance Mean / Max values), and a policy uncertainty metric (policy action log probability). For fair comparison, we quickly tune these baselines by viewing rollouts from a single seed on a single task, and heuristically choose the best threshold error value, and set $\psi_{KL} = 35.0$, $\psi_{enc\_mean} = 0.0011$, $\psi_{enc\_max} = 0.0017$, and $\psi_{\pi\_lp} = 17.0$. Note that for the policy log probability metric, we assume low probabilities correspond to error states; moreover, because we utilize a GMM policy, the log probabilites can span multiple orders of magnitude. For these reasons, we consider the negative log of the log probability. We test these methods utilizing the same procedure as our core experiments, evaluating three seeds for each error detector trained using the expert demonstration dataset on each task, and recording the error detector precision and recall statistics from 30 rollouts for each seed. Our results can be seen in Table A.3.

We find that across all tasks, our method easily outperforms all baselines, and achieves consistently high precision and recall rates. This further validates prior work [53, 54, 55, 56] that has found reconstruction error to be a viable metric for detecting errors, and also highlights the value of our method's ability to be easily tuned and robust across multiple tasks, which is a property not apparent in the other baselines. Indeed, we found that for the latent and log probability methods, the error signal was very noisy and did not necessarily transfer well between tasks given the same threshold ψ value.

The best baseline is the KL Divergence method, which is the only method to achieve near- or even marginally better precision over our method in certain tasks. This suggests that loss-based metrics can be a promising method for neural network-driven error detection methods, and motivates future research in this area.

A.9 Few-Shot Generalization Baseline

To better contextualize the challenges of few-shot generalization learning, we evaluate a baseline model that is trained from scratch exclusively on the few-shot demonstrations in the shifted domain setting. As in the case with our core results, we train each model for 30 epochs and record the average top three best evaluation success rates from 30 episodes aggregated over 3 seeds. The results can be seen in Table A.4.

We find that across all tasks, pretraining using the core dataset and finetuning on the few-shot demonstrations result in better policy performance compared to solely training on few-shot demonstrations from scratch. This validates the potential for direct transfer learning methods to provide tangible benefits in the MM setting, and suggests that information may be potentially shared across diverse task initializations.

A.10 Sim2real Potential

Due to the pandemic, we were unable to deploy our method on a real Fetch robot. However, we believe our method can transfer to the real world, because of the minimal assumptions we make. Both the teleoperator and agent are limited to on-board sensors, and are not provided any privileged information such as low-level object states or a global camera view: the imitation learning policy uses as input **only** RGB-D images and laser scan point clouds. The visual observations provided by the simulator, iGibson [62] are high-quality, with natural textures and materials rendered with a physics-based renderer, background and lighting probes obtained from real world. Moreover, the model used for our simulated robot closely aligns with the real Fetch robot, with identical sensor specifications, kinematics, and control schemes.

We do recognize there are some key limitations: the well-known gap between real and simulated physics dynamics, caused by approximated contact models, friction and misalignment in articulated joints, may be a source of some divergence in the real world. While more demonstration data may compensate for some of these discrepancies, it is unclear how much of an impact this can cause on downstream policy performance. We expect to be able to transfer part of the performance obtained with simulated data to real world, but if this demonstrates to be unfeasible, our immediate next step after the pandemic is to use MoMART to control a real mobile manipulator and collect real world data, as it has been done with similar smartphone-based systems for real-world stationary arms [57]. Because our system (teleoperation, imitation learning, and error detection) is not using any privileged information, we have high expectations that it will perform similarly as in simulation.