A Environment Details

In this section we provide further details on the different environments used in our experiments.

A.1 MT-Coarse Manipulation

For coarse manipulation tasks we focus on a variety of objects including blocks, mugs, cups, and shoes (both men and women shoes). As noted in the main paper, for these set of objects we focus on pick-and-place skills. However, we note that we did experiment with more complex contact-rich skills (e.g. pushing, stacking). However, we found the physics to be unstable with more complex objects (e.g. cups). For instance, pushing cups would almost always topple them and roll over. For future work, we hope to make our skills more robust.

Specifically, we use fixed size blocks with different semantic colors, 4 mugs, 4 cups and 4 shoes. We use google scanned objects [64] to collect non-block objects and use mujoco [65] to simulate our environment. We use the latest mujoco environments to import meshes into the simulator. Each environment in this set of tasks is created by first selecting a target object-type and then selecting a target object from the set of objects. We then select 3-5 distractor objects to fill the scene. These objects are uniformly selected from the remaining objects.

A.2 MT-Precise Manipulation

As noted in the main paper for precise manipulation tasks we use the spatial precision set of tasks from RLBench [57]. Overall, we use 4 tasks (see Figure 3 (Left)) – square block insertion, pick up small objects, shape sorting, and unplug usb from computer. We avoid using the motion-planner augmented approach for solving these tasks and instead opt for learning reactive closed-loop control policies. We use the delta end-effector actions for our tasks. Additionally, we use standard front and wrist mounted camera. along with proprioceptive and force-torque feedback as policy input.

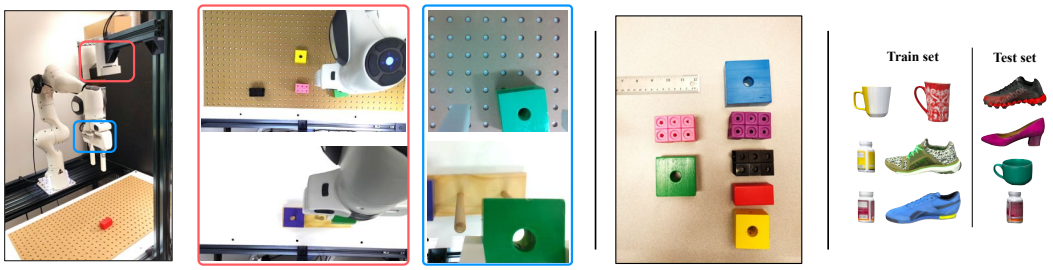
However, directly using end-effector actions increases the policy horizon significantly. Moreover, naively using the original input distribution for each task also requires learning full 6-DOF policies. Both of these can significantly increase the data requirements to learn the manipulation policy. To avoid this we restrict the starting distributions for each task such that the objects are spawned in a slightly narrow region infront of the robot. We further make other task-specific changes, detailed below, such that the robot can perform each task without changing hand orientations.

Insert Onto Square Peg: For this task we restrict the orientations of the square ring (blue object) and the peg on which to insert. This allows the robot to perform the task without changing gripper orientations. Further, we use a region of $40cm \times 30cm$ infront of the robot to spawn both the base and ring. Finally, the default task configuration provides 20 different peg colors, of which we use the first 10 colors for training and remaining 10 colors for robustness experiments.

Pick and Lift Small: For this task, we again use a region of $40cm \times 30cm$ infront of the robot to spawn both all objects. We also restrict the orientation of each object such that it can be grasped directly without requireing gripper orientation changes.

Shape-Sorting: The default configuration for the shape-sorting task considers 4 different shaped objects (see Figure 3 Bottom-Left) – square, cylinder, triangle, star, moon. In the default RLBench configuration most objects directly stick to the robot finger and are simply dropped into the hole for task completion. However, with closed loop control we find that non-symmetric objects (star, triangle, and moon) can have significant post-grasp displacement such that it is impossible to insert these objects without changing gripper orientation. Hence, we exclude these two objects from evaluation and only use symmetric square and cylinder objects.

Take USB Out: This task requires the robot to unplug a USB inserted into the computer. However, the default configuration for this task requires 6-dof control. To avoid this, we create smaller computer and USB assets and mount them vertically on the table such that the USB can be unplugged without changing hand orientation. See Figure 3 (Bottom-Right) for visualization.



(A) Real-world setup for pickup and insertion tasks.

(B) Examples objects for real-world pickup task

(C) Example objects for coarse manipulation task

Figure 6: *Left:* Real World env setup with third-person (red) and first-person (blue) camera views. *Middle:* Example objects set used for real-world pickup task. *Right:* Example objects used for MT-coarse.

A.3 MT-Dynamic Manipulation

This task involves using the CMU Ballbot in simulation (PyBullet [66]) to perform a dynamic pick up task. The task involves picking up a block that is placed on a table in front of the ballbot. We use two blocks (red and blue) in this task and use language instructions to specify which object to pick up. The initial conditions are set such that the table and objects are always out of the reach of the ballbot arms and the ballbot has to roll forward to pick up the objects. We use a statically mounted camera looking at the table and the ballbot as the third-person camera and the camera located on the turret of the ballbot as the first-person camera. The turret tilt is adjusted such that the objects on the table are initially out of the view of the turret camera and only when the ballbot starts moving towards the table, the objects come into view. The third person camera is always able to view both the objects and the ballbot. We use task space control to control the ballbot end-effector while a center of mass balancing controller is always running in a high-frequency feedback loop to balance the ballbot.

B Architecture Details

Section 3 discusses the overall architecture used in our work. To recall, our proposed architecture uses a multi-resolution approach with multiple-sensors, each with different fidelity. We process each sensor with a separate network which is conditionally initialized using a pre-trained vision-language model. The output of each vision model is flattened to create a set of patches. For DETR [50, 51] based model we use a ResNet-101 backbone and flatten the output layer into 49 patches and add positional embedding to it. For CLIP [4] we use a ViT-B model and use hierarchical features from the 5'th, 8'th and 11'th layer. Since MDETR already does vision-language fusion using a transformer we directly use its output. However, since CLIP only weakly associates vision and language at the last layer, we additionally use FiLM layers to condition the output. Our use of FiLM is similar to previous models [67]. For each camera modality we use a small transformer with multi-head attention. Each transformer uses an embedding size of 256 and 8 heads. We use post layer-norm in each transformer layer. Further, in each transformer layer we use cross-attention with the other camera. Overall we use 3 transformer layers for each camera modality. Our force-torque and proprioceptive input is concatenated together and mapped into 256 dimensions using a linear layer. We concatenate the readout tokens from each camera transformer and the force-torque embedding. This 256×3 size embedding is then processed by 2 linear layers of size 512 which output the robot action.

Input: For each of our camera sensor we use an image of size 224×224. For proprioceptive input we use the end-effector position of the arm. While for force-torque input we use the 6 dimensional force-torque data. We use cropping augmentation for both camera sensors. Specifically, we first resize the image to 226 and then do random crop with shift $= 8$. For, more aggressive pixel level

Key	Value
batch size	16
proprio and force torque embedding	256
camera-transformer embedding Dim.	256
camera-transformer feedForward Dim.	768
Number of transformer layers	3
learning rate	0.0001
warmup epochs	5
total epochs	60
optimizer	AdamW
weight decay	0.01
scheduler	cosine

Table 6: Hyperparameters used for our architecture and model training.

augmentations we stochastically apply grayscale and use color jitter with brightness $\in (0.4, 0.8)$, contrast $\in (0.4, 0.8)$, saturation $\in (0.4, 0.6)$ and hue $\in (0.0, 0.5)$. These augmentations significantly change the underlying visual semantics of the task.

B.1 Training Details

In this section we provide details on the demonstrations (for each environment type) used to train our approach. Further, we also provide details on the train and heldout configurations used for robustness evaluation.

MT-Coarse: As noted above in Appendix A.1, we use multiple different objects to train and evaluate our policy. Each environment is created by first sampling a target object and then a set of distractor objects. For each environment and skill combination we collect 20 demonstrations. Overall, this gives us ≈ 1000 demonstrations across all tasks. We then learn one policy across all tasks.

MT-Precise: For spatial precision tasks from tasks from RLBench [57] we use 4 different tasks. As discussed in Section A.2, each task has it's own set of variations. For training our multi-task policy we use try to balance the number of demonstrations from each task. For square peg insertion (*insert onto square peg*) task we use first 10 variations for training and gather 25 trajectories per variation. Each other task has less than 4 variations hence for each task we use 100 demonstrations each for training. To test visual-semantic robustness for these tasks Section 5.2 we use the insert-onto-square-peg task since only this task has any semantic variations. We use the remaining 10 peg colors (i.e. 10 heldout variations) to test each approach.

MT-Dynamic: To collect expert demonstrations, we sample the locations of the objects on the table in a 70cm*20cm region and sample the initial ballbot location in a 50cm*50cm region. We collect 50 demonstrations for each task setting (each block). As noted earlier, the third-person camera is used at a frequency of 5Hz, the turret camera is used at 20Hz and proprioception and force-torque feedback is used at 75Hz.

Real-World: For real-world tasks we collect data using teleoperation with a leap-motion device which can track hand movements upto a 100Hz. We map these movements to robot movements and collect proprioceptive and force-torque data at 75Hz, while both cameras are recorded at 30Hz. To collect data for pickup tasks we use two blocks with different shapes and different colors. The green and pink blocks in Figure 6 (Right) were used to collect all training data. While evaluation happened on 8 other blocks, each with a different shape and color. For training our policies we collect 60 demonstrations for each pickup variation and 50 demos for the insertion task. We note that the initial state distribution for insertion was narrower than pickup and hence it required fewer demonstrations.

Metrics: We use task success as the evaluation metric. Since we use a multi-task setting we report mean success over all tasks. During training, we evaluate the policy every 4 epochs on all train tasks. We report the average over *top-5* mean success rates across all evaluation epochs. For task generalization results (**Q3**) we use the trained policy and evaluate it on novel visual-semantic tasks

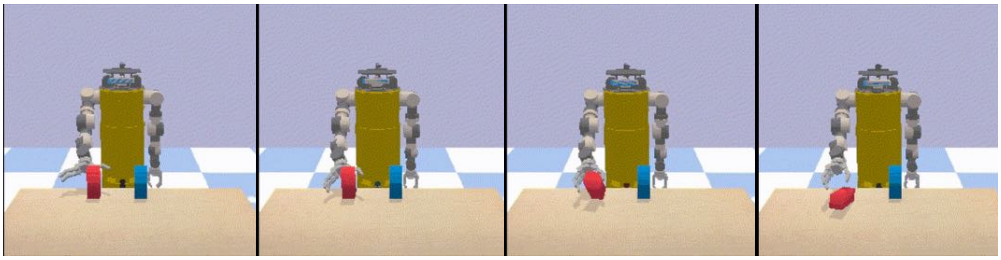


Figure 7: Example failure case for MT-Dynamic (Ballbot) task. As can be seen in the figure, if the robot approaches the object but does not react fast enough to the object contact, the block can topple resulting, in task failure.

which were never seen during training. Hence, for Q3 we report task success on novel unseen tasks. For all evaluations we use 20 rollouts per task. Further training details are provided in Appendix B.1.

C Additional Ablations

We further ablation on the different components of our proposed approach. For these set of results instead of using all 3 environment suites for evaluation, we choose the most appropriate environment suite for each component of our approach and evaluate on it.

Pixel-Level Augmentations: We evaluate the effect of pixel-level augmentations (color jitter, gray-scale) on the training and generalization of our MT-policies on MT-Coarse. Figure 5 reports results on both training and heldout (novel) evaluation configurations. We see that while there is very little difference in training performance, extensive pixel-level augmentations helps generalization by close to $\approx 15\%$. While pixel-level augmentations change the semantics of the task, our multi-modal approach is still able to complete the task because of visual-language grounded provided from pretraining.

Multi-Modal Fusion using Cross-Attention: We compare use of early fusion using cross-attention with late fusion using concatenation. Figure 5 shows that using cross-attention improves the performance by around $\approx 8\%$ on both train and heldout configuration. Thus, using cross-attention for multi-modal fusion is more effective than concatenation. However, we note that cross-attention requires more parameters and has slower inference.

Effect of Pretrained-VLMs: We also evaluate the effects of using pretrained-VLMs. Figure 5 shows the training and heldout performance using ImageNet initialization which only has visual pretraining and no vision-language pretraining. We see that while training performance matches our approach the heldout performance decreases tremendously. This large decrease is due to missing visual-language grounding since we use *separately* trained visual and language models.