A Task Details

Task	precise placing	multimodal placing	multi-step sequencing	unseen poses	unseen colors	unseen objects	language instruction
put-blocks-in-bowls-seen-colors*	✗	✓	✗	✓	✗	✗	goal
put-blocks-in-bowls-unseen-colors*	✗	✓	✗	✓	✓	✗	goal
assembling-kits-seq-seen-colors	✓	✓	✓	✓	✗	✓	step
assembling-kits-seq-unseen-colors	✓	✓	✓	✓	✓	✓	step
packing-unseen-shapes	✗	✓	✗	✓	✓	✓	goal
stack-block-pyramid-seq-seen-colors	✓	✓	✓	✓	✗	✗	step
stack-block-pyramid-seq-unseen-colors	✓	✓	✓	✓	✓	✗	step
towers-of-hanoi-seq-seen-colors	✓	✓	✓	✓	✗	✗	step
towers-of-hanoi-seq-unseen-colors	✓	✓	✓	✓	✓	✗	step
packing-box-pairs-seen-colors*§	✓	✓	✓	✓	✗	✓	goal
packing-box-pairs-unseen-colors*§	✓	✓	✓	✓	✓	✓	goal
packing-seen-google-objects-seq§	✗	✓	✓	✓	✗	✗	step
packing-unseen-google-objects-seq§	✗	✓	✓	✓	✓	✓	step
packing-seen-google-objects-group*§	✗	✓	✗	✓	✗	✗	goal
packing-unseen-google-objects-group*§	✗	✓	✗	✓	✓	✓	goal
align-rope*†	✓	✓	✓	✓	✗	✗	goal
separating-piles-seen-colors*†	✓	✓	✓	✓	✗	✗	goal
separating-piles-unseen-colors*†	✓	✓	✓	✓	✓	✗	goal

§ tasks that are commonly found in industry.
* tasks that have more than one correct sequence of actions.
† tasks that require manipulating deformable objects and granular media.

Table 3. Language-conditioned tasks in Ravens [2] with their associated challenges.

We extend the Ravens benchmark [2] to 10 language-conditioned. 8 out of 10 tasks have two evaluation variants, denoted by seen and unseen in their names. See Table A for an overview of the challenges associated with each task and split. Figure 5 presents the full list of attributes, shapes, and objects across seen and unseen splits. All tasks use hand-coded experts to generate expert demonstrations. These experts use privileged state information from the simulator along with pre-specified heuristics to complete the tasks. We refer the reader to the original Transporter paper [2] for details regarding these experts. The following is a description of each language-conditioned task:

A.1 Align Rope

Example: Figure 1(a).

Task: Manipulate a deformable rope to connect its end-points between two corners of a 3-sided square. There are four possible combinations for aligning the rope: "front left tip to front right tip". "front right tip to back right corner", "front left tip to back left corner", and "back right corner to back left corner". Here 'front' and 'back' refer to canonical positions on the 3-sided square. The poses of both the rope and 3-sided square are randomized for each task instance.

Objects: All align-rope instances contain a rope with 20 articulated beads and a 3-sided square.

Success Metric: The poses of all beads match the line segments between the two correct sides.

A.2 Packing Unseen Shapes

Example: Figure 1(b).

Task: Place a specified shape in the brown box. Each task instance contains 1 shape to be picked along with 4 distractor shapes. The shape colors are randomized but have no relevance to the task. This task does not require precise placements and is mostly a test of the agent's semantic understanding of arbitrary shapes.

Objects: packing-unseen-shapes is trained with seen shapes but evaluated on unseen shapes from Figure 5.

Success Metric: The correct shape is inside the bounds of the brown box.

A.3 Assembling Kits Seq

Example: Figure 1(c).

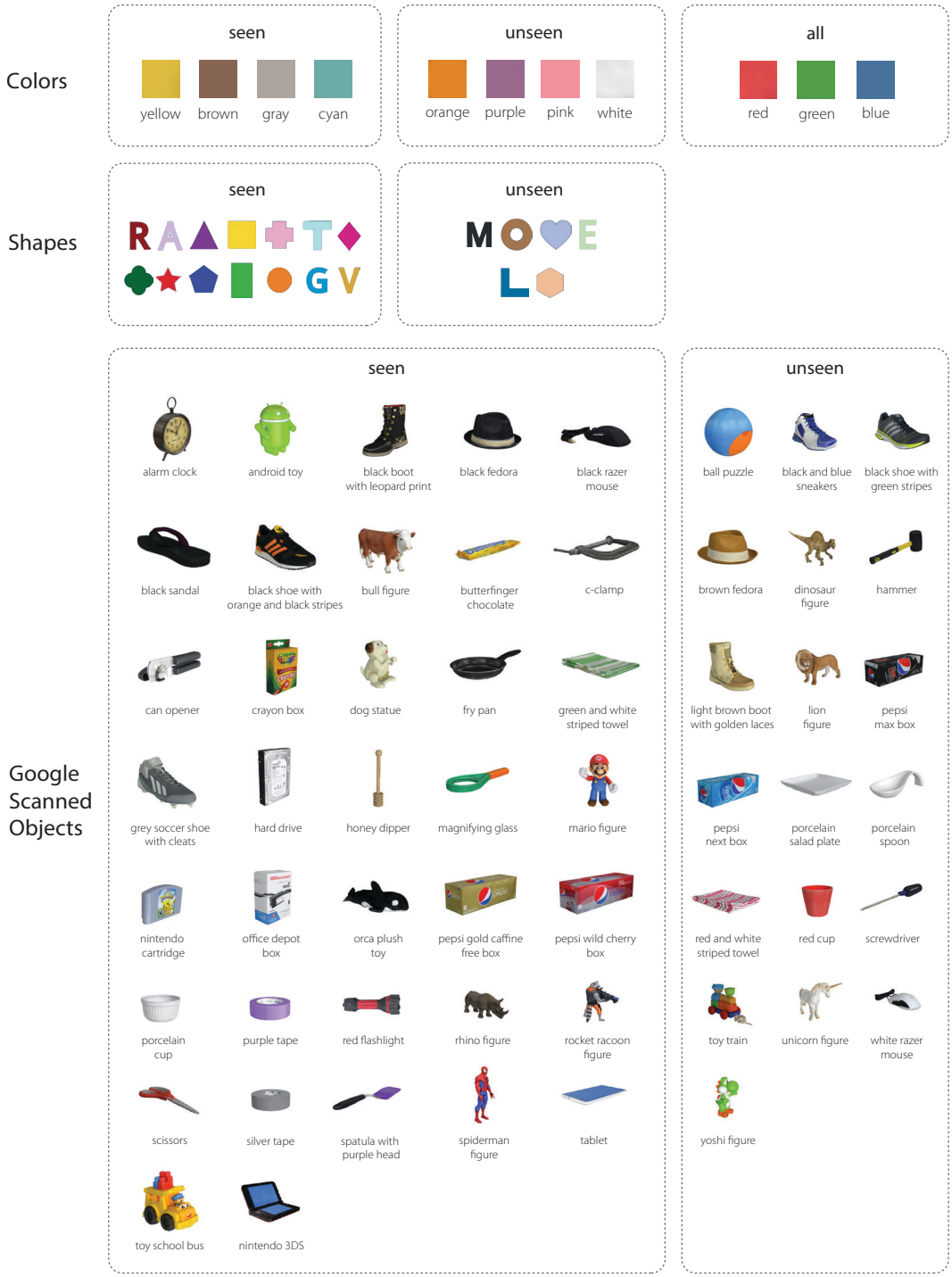

Figure 5. **Attributes and Objects**: Attributes and objects across seen and unseen splits. Shapes objects are from Transporter [2]. Other tabletop objects are from the Google Scanned Objects dataset [61]

Task: Precisely place each specified shape in the specified hole following the order prescribed in the language instruction at each timestep. This is one of the hardest tasks in the benchmark requiring precise placements of unseen shapes of unseen colors and grounding spatial relationships like "the middle square hole" or "the bottom letter R hole". Each task instance contains 5 shapes and a kit with randomized poses.

Objects: Both assembling-kits-seq-seen-colors and assembling-kits-seq-unseen-colors are trained on seen shapes but evaluated on unseen shapes from Figure 5. However for color randomization, assembling-kits-seq-seen-colors is trained and evaluated on seen colors, and assembling-kits-seq-unseen-colors is trained with seen colors but evaluated on unseen colors from Figure 5.

Success Metric: The pose of each shape matches the specified hole at the correct timestep. The final score is the total number of shapes that were placed in the correct pose at the correct timestep, divided by the total number of shapes in the scene (always 5).

A.4 Put Blocks in Bowl

Example: Figure 1(d).

Task: Place all blocks of a specified color in a bowl of specified color. Each bowl fits just one block and all scenes contain enough bowls achieve the goal. Each task instance contains several distractor blocks and bowls with randomized colors. The solutions to this task are multi-modal in that there could be several ways to place the blocks specified in the language goal. This task does not require precise placements and mostly tests an agent's ability to ground color attributes.

Objects: put-blocks-in-bowl-seen-colors is trained and evaluated on seen colors from Figure 5 for both blocks and bowls. put-blocks-in-bowl-unseen-colors is trained on seen colors but evaluated on unseen colors from Figure 5 for both blocks and bowls.

Success Metric: All blocks of the specified color are within the bounds a bowl of the specified color. The final score is the total number of correct blocks in the correct bowls, divided by the total number of relevant color blocks in the scene.

A.5 Packing Box Pairs

Example: Figure 1(e).

Task: Tightly pack all the boxes of two specified colors inside the brown box. All scenes contain the exact number of relevant color blocks to fill the box completely, but also contain some distractor boxes of irrelevant colors. The sizes of the boxes and the brown box are randomized. The distractor objects have equivalent sizes to the relevant objects to make the task more difficult. Sometimes the scene only contains one of the two specified specified colors and the agent has to actively ignore the missing color. Overall, this task requires both semantic understanding of colors and precise spatial reasoning for tightly packing boxes of unknown sizes.

Objects: Boxes with randomized widths and lengths and a brown box. packing-box-pairs-seen-colors is trained and evaluated on seen color boxes from Figure 5. packing-box-pairs-unseen-colors is trained on seen color boxes but evaluated on unseen color boxes from Figure 5.

Success Metric: All blocks of the two specified colors are tightly packed inside the bounds of the brown box. The final score is the total volume of the correct color blocks inside the box, divided by the total volume of the relevant color blocks in the scene.

A.6 Packing Google Objects Seq

Example: Figure 1(f).

Task: Place the specified objects in the brown box following the order prescribed in the language instruction at each timestep. This task does not require precise placements and mostly evaluates an agent's ability to ground semantic object descriptions. All objects in a scene are unique without any duplicates. The poses of the objects and the box are randomized for each scene.

Objects: packing-seen-google-objects-seq is trained and evaluated on all 56 objects in Figure 5. packing-unseen-google-objects-seq is trained on 37 seen objects but evaluated on 19 unseen objects in Figure 5.

Success Metric: Each specified object is within the bounds of the brown box at the correct timestep. The final score is the total volume of the correct objects placed inside the box at the correct timestep, divided by the total volume of the relevant objects.

A.7 Packing Google Objects Group

Example: Figure 1(g).

Task: Place all objects of the specified category in the brown box. This task does not require precise placements or following a specific action sequence. Each scene contains objects of multiple categories with each category containing at least 2 duplicates. The task cannot be solved by counting the number of objects since there are distractor objects, each with 2 or more duplicates.

Objects: packing-seen-google-objects-group is trained and evaluated on all 56 objects in Figure 5. packing-unseen-google-objects-group is trained on 37 seen objects but evaluated on 19 unseen objects in Figure 5.

Success Metric: All specified objects of a category are within the bounds of the brown box. The final score is the total volume of the correct objects in the box, divided by the total volume of the relevant objects of the specified category in the scene.

A.8 Stack Block Pyramid

Example: Figure 1(h).

Task: Build a pyramid of colored blocks in a color sequence specified through the step-by-step language instructions. Each task contains 6 blocks with randomized colors and 1 rectangular base, all initially placed at random poses.

Objects: 6 blocks and 1 rectangular base. stack-block-pyramid-seq-seen-colors is trained and evaluated on seen color blocks from Figure 5. stack-block-pyramid-seq-unseen-colors is trained on seen color blocks but evaluated on unseen color blocks from Figure 5.

Success Metric: The pose of each block at the corresponding timestep matches the specified location. The final score is the total number of blocks in the correct pose at the correct timestep, divided by the total number of blocks (always 6).

A.9 Separating Piles

Example: Figure 1(i).

Task: Sweep the pile of blocks into the specified zone. Each scene contains two square zones: one relevant to the task, another as a distractor. The pile and zones are placed at random poses on the table.

Objects: A pile of colored blocks and two squares. separating-piles-seen-colors is trained and evaluated on seen colors from Figure 5 for all blocks and squares. separating-piles-unseen-colors is trained on seen colors but evaluated on unseen colors from Figure 5 for all blocks and squares.

Success Metric: All blocks are inside the bounds of the specified zone. The final score is the total number of blocks inside the correct zone, divided by the total number of blocks in the scene.

A.10 Towers of Hanoi Seq

Example: Figure 1(j).

Task: Move the ring to the specified peg in the language instruction at each timestep. The sequence of ring placements is always the same, i.e. the perfect solution to three-ring Towers of Hanoi. This task can be solved without using colors by just observing the ring sizes. However, it tests the agent's ability to ignore irrelevant concepts to the task (color in this case). The task involves precise pick and place actions for moving the rings from peg to peg.

Objects: 1 peg base and 3 rings (small, medium, and big). towers-of-hanoi-seen-colors is trained and evaluated on seen ring colors from Figure 5. towers-of-hanoi-unseen-colors is trained on seen ring colors but evaluated on unseen ring colors from Figure 5.

Success Metric: The pose of each ring at the corresponding timestep matches the specified peg location. The final score is the total number of correct ring placements, divided by total steps in the perfect solution (7 for three-ring Towers of Hanoi).

B Evaluation Workflow and Validation Results

Method	packing-box-pairs seen-colors				packing-box-pairs unseen-colors				packing-seen-google objects-seq				packing-unseen-google objects-seq				packing-seen-google objects-group				packing-unseen-google objects-group			
	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
Transporter-only [2]	48.9	57.2	59.4	60.6	37.8	52.3	54.5	60.7	30.2	41.6	42.4	46.3	26.3	37.1	42.9	40.8	56.3	52.8	55.6	54.5	30.8	55.3	53.6	56.0
CLIP-only	37.1	72.3	87.4	90.9	36.1	61.8	67.2	62.9	30.5	76.5	89.1	**97.7**	37.8	48.9	55.2	58.9	53.3	66.1	90.6	94.6	46.7	63.3	76.7	78.1
RN50-BERT	40.0	64.4	94.7	90.5	42.1	58.7	62.4	72.2	29.7	49.8	90.4	94.6	39.9	41.8	57.5	57.2	48.5	56.9	83.1	93.6	44.8	55.3	71.7	77.9
CLIPort (single)	51.9	84.7	95.9	**98.0**	47.1	66.9	70.0	71.9	14.4	63.9	**95.3**	96.9	25.0	50.6	62.7	62.0	53.3	72.5	90.3	**95.6**	54.9	68.5	78.3	73.3
CLIPort (multi)	**68.6**	**90.0**	**96.0**	96.3	**55.9**	**70.3**	**76.6**	**72.9**	**45.7**	**78.4**	83.8	83.4	**50.8**	**60.8**	**65.1**	**68.8**	**69.4**	**86.2**	**92.2**	93.2	**66.9**	**73.4**	**82.0**	**81.7**
CLIPort (multi-attr)	–	–	–	–	*46.2*	*72.0*	*86.2*	*80.3*	–	–	–	–	*35.4*	*45.1*	*78.7*	*87.4*	–	–	–	–	*48.6*	*69.3*	*84.8*	*89.1*

Method	stack-block-pyramid seq-seen-colors				stack-block-pyramid seq-unseen-colors				separating-piles seen-colors				separating-piles unseen-colors				towers-of-hanoi seq-seen-colors				towers-of-hanoi seq-unseen-colors			
	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
Transporter-only [2]	4.8	4.0	6.8	5.7	4.8	5.3	5.0	5.0	42.8	52.9	54.7	55.6	47.8	53.4	52.6	54.8	25.1	74.4	**100**	**100**	25.6	46.4	77.0	81.7
CLIP-only	5.5	30.0	58.7	59.0	2.0	16.3	5.7	19.3	39.7	69.6	90.4	92.9	46.4	61.6	76.9	74.4	10.9	48.1	88.6	52.9	15.9	44.7	67.1	58.1
RN50-BERT	5.7	35.5	94.0	98.0	5.2	10.5	19.7	33.3	33.3	55.9	53.0	48.7	35.7	52.2	53.1	57.0	26.4	68.1	92.7	95.9	16.3	75.0	82.0	84.3
CLIPort (single)	29.0	68.8	95.0	**99.3**	15.8	29.0	32.7	**41.8**	45.1	58.6	**96.8**	**99.9**	50.7	56.5	**83.8**	**83.0**	55.3	**94.1**	99.9	**100**	**66.6**	**91.9**	**96.4**	**100**
CLIPort (multi)	**38.3**	**71.0**	**97.0**	97.3	**27.8**	**31.8**	**39.3**	33.3	**53.2**	**73.0**	92.7	89.2	**55.5**	**71.2**	79.5	76.7	**67.6**	94.0	99.1	**100**	55.6	68.6	79.1	67.0
CLIPort (multi-attr)	–	–	–	–	*17.2*	*45.2*	*65.3*	*81.5*	–	–	–	–	*49.9*	*51.8*	*48.2*	*59.8*	–	–	–	–	*56.7*	*78.0*	*88.3*	*96.9*

Method	align-rope				packing-unseen-shapes				assembling-kits-seq seen-colors				assembling-kits-seq unseen-colors				put-blocks-in-bowls seen-colors				put-blocks-in-bowls unseen-colors			
	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
Transporter-only [2]	6.3	24.7	39.8	48.2	28.0	34.0	27.0	32.0	6.8	15.2	30.8	32.6	9.4	15.6	30.4	30.0	18.8	45.2	63.2	69.0	12.2	16.8	20.5	21.7
CLIP-only	15.4	47.6	76.7	74.3	26.0	36.0	40.0	**43.0**	1.4	6.4	19.0	27.2	4.2	5.6	12.0	16.2	22.3	62.2	94.7	98.5	15.8	29.7	38.3	24.7
RN50-BERT	6.8	26.9	69.8	61.1	22.0	31.0	29.0	30.0	2.4	6.8	15.2	23.0	2.2	7.6	15.2	19.4	10.8	46.3	82.3	92.2	14.0	24.2	29.7	27.7
CLIPort (single)	14.8	**66.2**	**93.2**	**98.2**	22.0	**42.0**	35.0	40.0	11.0	28.8	**51.6**	**72.0**	**17.2**	23.2	33.0	**38.0**	21.7	73.0	98.2	**100**	17.2	32.5	40.2	**48.3**
CLIPort (multi)	**19.2**	52.4	80.2	72.2	**29.0**	**42.0**	**47.0**	41.0	**17.4**	**37.2**	48.2	57.6	12.2	**23.8**	**36.4**	29.0	**59.7**	**94.0**	**100**	**100**	**33.8**	**42.7**	**55.3**	43.3
CLIPort (multi-attr)	–	–	–	–	–	–	–	–	–	–	–	–	*9.0*	*18.4*	*41.6*	*39.8*	–	–	–	–	*23.0*	*41.8*	*66.5*	*75.7*

Table 4. **Validation Results.** Task success scores (mean %) from 100 evaluation instances vs. # of training demonstrations (1, 10, 100, or 1000). The challenges pertaining to each task can be found in Appendix A. CLIPort (single) models are trained on seen splits, and evaluated on both seen and unseen splits. CLIPort (multi) models are trained on seen splits of all 10 tasks with $1\mathbb{T}$, $10\mathbb{T}$, $100\mathbb{T}$, and $1000\mathbb{T}$ demonstrations where $\mathbb{T} = 10$. CLIPort (multi-attr) indicate CLIPort (multi) models trained on seen-and-unseen splits from all tasks *except* for that one particular heldout task, for which it is trained only the seen split. See Figure 6 for an overview with average scores.

Evaluation Workflow. All simulated experiments in Section 4.1 follow a four-phase workflow: (1) generate train, validation, and test sets, (2) train agents on the train set, (3) optimize on the validation set to find the best checkpoint, (4) evaluate the best checkpoint on the test set. Both validation and test sets consist of 100 evaluation instances each. We found that validation loss is a poor metric for determining the best checkpoint as actions are often multi-modal. In a task like "put the yellow blocks in the red bowl" where there are three possible yellow blocks to choose from, the validation loss is high if the agent chooses a different yellow block to the expert, but in fact choosing any yellow block would suffice in achieving the goal. This issue is addressed by determining the best checkpoint through task execution performance on the validation set.

Validation Performances. During validation, we evaluate a trained agent across fixed checkpoints between 1K-200K iterations for single-task settings and 1K-600K iterations for multi-task settings. We then choose the best-performing checkpoint for each task. Table 4 presents validation results for all tests in Section 4.1. Following Transporter [2], we use a learning rate of `1e-4` with no additional hyperparameter tuning. We note that better learning rate schedules and other hyperparameter optimizations could possibly improve the performance of agents, especially in multi-task settings.

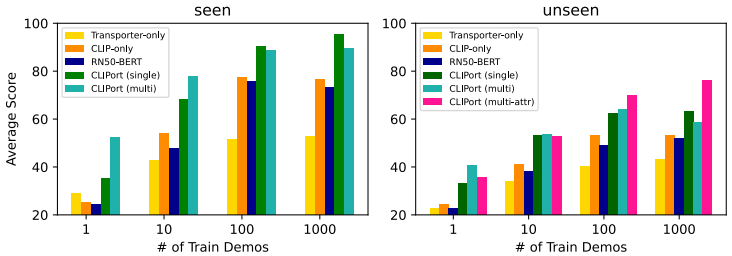

Figure 6. Average validation scores across seen and unseen splits for all tasks in Table 4.

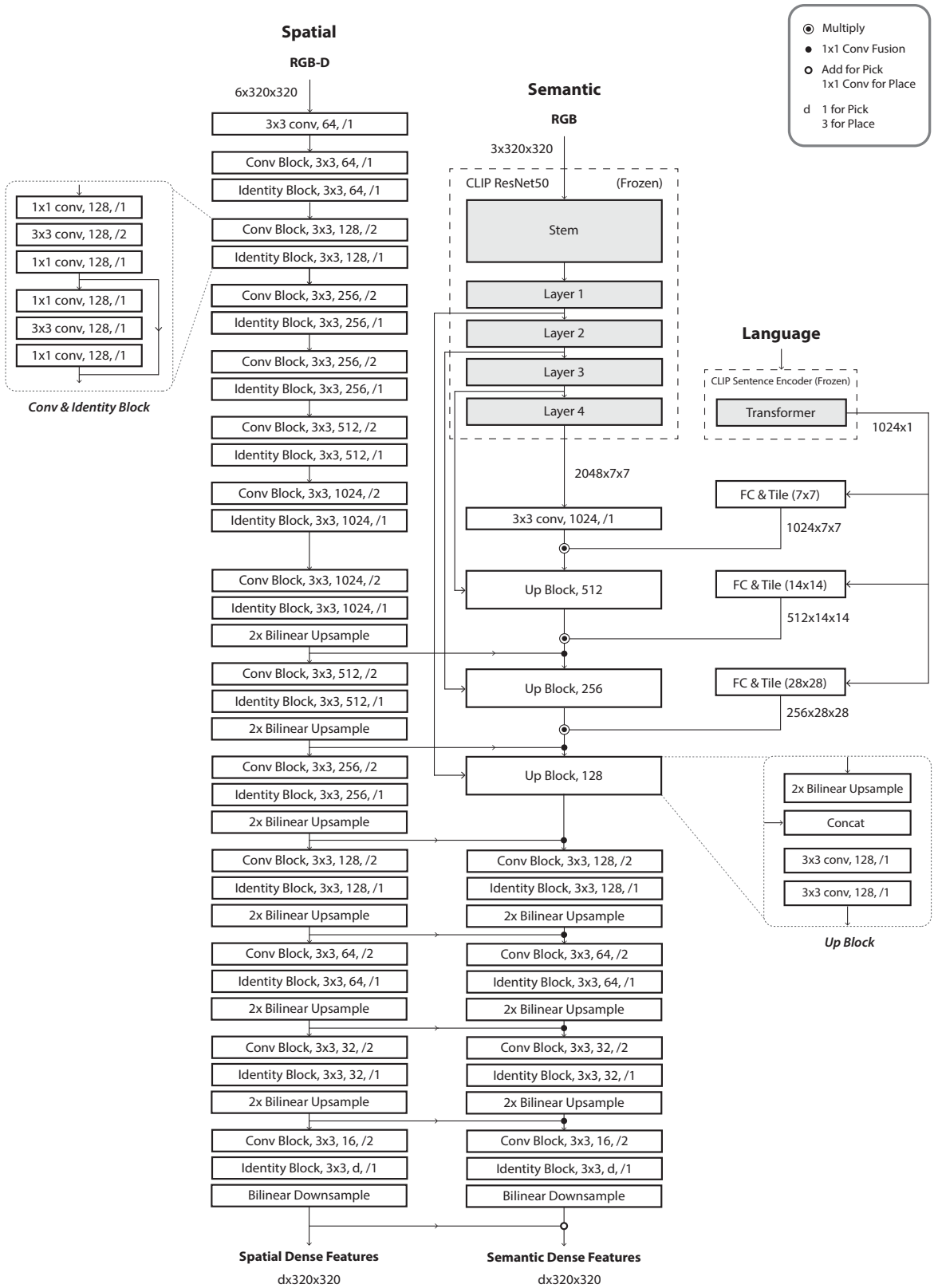

Figure 7. **CLIPORT Two-Stream Architecture**: A detailed architecture diagram of the semantic and spatial pathways.

C Two Stream Architecture Details

Figure 7 provides a detailed architecture diagram of CLIPORT's two-stream design. We use ReLU activations after each `conv` and `identity` blocks without any Batch Normalization. Note that we repeat the depth input to match the dimensions of the RGB image $\mathbb{R}^{H\times W\times 1} \rightarrow \mathbb{R}^{H\times W\times 3}$ following Transporter [2]. All models were implemented in PyTorch [64]. For CLIP, we use the implementation and pre-trained checkpoint released by the authors[3].

D Robot Setup

Hardware Setup. All real-robot experiments were conducted on a Franka Panda robot with a parallel-gripper. For perception, we use a Kinect-2 RGB-D camera mounted on a tripod, tilted down looking at the table. Although the Kinect-2 provides images at a resolution of 1280×720, we use downsampled 960×540 images for a faster user-interface. The extrinsic calibration between the camera and the robot base-frame is computed with an AR Marker through ARUCO ROS[4]. See Figure 8 for an overview of the setup.

Figure 8. Real-Robot Experimental Setup.

Demonstrations and Execution. For collecting demonstrations with the Franka Panda, we developed a 2D interactive tool that uses the top-down RGB view from the Kinect-2 to specify pick-and-place locations. The user first selects a 2D bounding box on the live RGB feed, and then picks a discrete rotation angle by clicking around the bounding box. For grasping, we use a simple heuristic to determine the height at which to close the fingers. First we segment the pointcloud encapsulated by the bounding box, then we vertically crop the pointcloud up to the height of the gripper fingers, and then compute a 3D centroid of the selected points by taking an average. This 3D centroid is used to plan a path for the end-effector with an RRT* motion-planner to execute a predefined sequence – go down, open/close the gripper, raise up. For executing a trained CLIPORT model, a similar grasping approach is used, but instead of the user-specified bounding box, we take 32×32 crops centered around the pick and place predictions (i.e. affordance argmax) to compute 3D centroids from the pointcloud. Only the sweeping and folding actions are different in that the end-effector does not raise up after grasping.

Pick Rotations for Parallel Grippers. The suction gripper used in simulation does not require a pick rotation since the grasps are specified as pin-point locations. However, with the Franka Panda, the parallel gripper requires a specific yaw rotation at which to grasp an object. To handle this, we separate the pick module $\mathcal{Q}_{\text{pick}}$ into two components: locator and rotator. The locator predicts a pixel location (u, v) given the full observation and language input. The rotator takes a 64×64 crop of the observation at (u, v) along with the language input and predicts a discrete rotation angle by selecting from one of k rotated crops. We use $k = 36$ in all our hardware experiments. While it's possible to predict both the location and rotation with a single module, this decoupled approach allows us to fit the model on a single GPU (NVIDIA P100) with reduced memory usage from cropped rotations.

E Data Augmentation

Following common practice and the original Transporter implementation [2], we augment the training samples by applying random $\mathbf{SE}(2)$ transformations. Augmentations where $\mathcal{T}_{\text{pick}}$ or $\mathcal{T}_{\text{place}}$ are out of frame after the transformation are discarded. These augmentations are particular important for learning spatially-equivariant representations with FCNs without overfitting to images from limited training demonstrations.

[3] https://github.com/openai/CLIP

[4] https://github.com/pal-robotics/aruco_ros

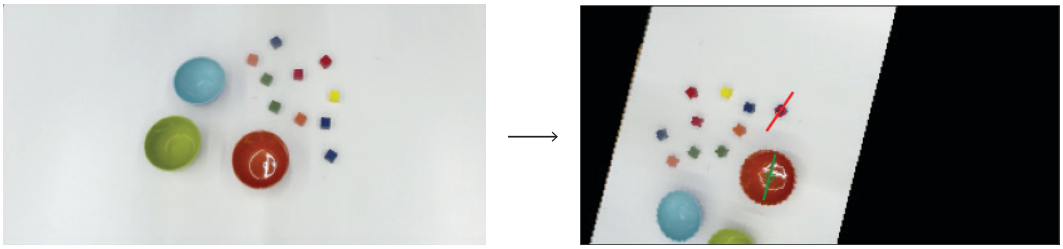

Figure 9. **Data Augmentation**: $\mathbf{SE}(2)$ transform applied to RGB-D input. The left image shows the original input, and the right image shows the transformed input along with expert $\mathcal{T}_{\text{pick}}$ (red) and $\mathcal{T}_{\text{place}}$ (green) actions.

F Ablations and Baselines

	stack-block-pyramid seq-seen-colors				stack-block-pyramid seq-unseen-colors				packing-seen-google object-seq				packing-unseen-google object-seq			
Method	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
One-Stream Transporter-only	4.5	2.3	5.2	4.5	3.0	4.0	2.3	5.8	26.2	39.7	45.4	46.3	19.9	29.8	28.7	37.3
One-Stream CLIP-only	6.3	28.7	55.7	54.8	2.0	12.2	18.3	19.5	52.5	62.0	**89.6**	92.7	43.4	**65.9**	**73.1**	70.0
One-Stream Language Transporter	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.2	0.1	0.1	0.0	0.0
One-Stream Image-Goal Transporter	1.8	1.3	7.0	6.8	2.5	4.7	4.2	4.8	**64.5**	**67.0**	81.8	85.4	**47.7**	62.8	71.0	**83.3**
Two-Stream CLIP-Transporter w/o skips	0.0	4.3	3.8	3.3	4.2	5.2	3.2	2.5	22.9	26.1	36.9	38.9	24.4	29.9	33.7	38.3
Two-Stream Untrained-Sem-Transporter	3.0	12.7	61.5	51.2	1.0	6.8	17.2	15.7	28.8	40.5	67.1	79.7	27.2	34.7	33.0	34.8
Two-Stream RN50-BERT-Transporter	5.3	35.0	89.0	97.5	6.2	12.2	21.5	30.7	32.9	48.4	87.9	94.0	29.3	48.5	48.3	56.1
Two-Stream CLIP-Transporter (ours)	**28.3**	**64.7**	**93.3**	**98.8**	**13.7**	**24.3**	**31.2**	**41.3**	14.8	59.5	86.8	**96.2**	27.2	50.0	65.5	71.9

Table 5. **Ablations and Baselines.** Evaluation scores (mean %) for stack-block-pyramid-seq and packing-google-objects-seq tasks from 100 evaluation runs. Stacking block pyramids involves both semantic and precise spatial reasoning, whereas packing objects mostly involves semantic grounding without requiring any precise placements.

Table 5 presents various baselines and ablations from our simulated experiments. The following is a description of each model:

One-Stream Transporter-only is the original Transporter [2] with RGB-D input, or equivalently, the **spatial** stream of CLIPORT. For all experiments, we implemented our own version of Transporter in PyTorch and did not use the modeling code provided with the original paper. Our Transporter models are also trained for 200K iterations instead of 40k iterations.

One-Stream CLIP-only is the **semantic** stream of CLIPORT with RGB and language input.

One-Stream Language Transporter is Transporter [2], but the bottleneck features are conditioned with CLIP language features in a similar fashion to the **semantic** stream in CLIPORT. This model performs very poorly because the high-level language features corrupt the low-level spatial features necessary for precise pick-and-place actions.

One-Stream Image-Goal Transporter is a goal-conditioned version of Transporter [6] which receives a goal-image as input. For sequential tasks with a specific order (indicated with seq in their name), we provide the goal-image from the next timestep, and for non-sequential tasks we provide the goal-image from the final timestep. The implementation follows the goal-conditioned Transporter proposed in [6], except we found that element-wise addition worked better than element-wise product for combining goal-image features with $\mathcal{Q}_{\text{place}}$ features.

Two-Stream CLIP-Transporter w/o skips is a variant of the CLIPORT model without skip connections from the CLIP-ResNet encoder to the decoder layers. The results in Table 5 show that these skip connections are particularly important for good performance. We hypothesize that utilizing different levels of semantic information from the visual encoder – patterns, shapes, parts, objects, and high-level concepts, is crucial for conditioning the **semantic** stream decoders.

Two-Stream RN50-BERT-Transporter is the same two-stream architecture as CLIPORT, except instead of the CLIP ResNet50, we use a standard ResNet50 [62] pre-trained on ImageNet classification. And instead of the CLIP sentence encoder, we use a pretrained DistilBERT model [65] to extract language embeddings. CLIP offers the benefit of multi-modal alignment between vision and language features while not being restricted to instance segmentation or bounding box detection pipelines.

Two-Stream Untrained-Sem-Transporter uses an untrained ResNet50 and Transformer language encoder for the **semantic** stream. Even without any pre-training, the random features from the **semantic** stream somewhat help in conditioning policies. However, the performances are substantially worse than models with pre-trained multimodal features.

G Performance on Demo-Conditioned Tasks

	block-insertion				place-red-in-green				towers-of-hanoi				align-box-corner				stack-block-pyramid			
Method	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
Transporter [2]	97.0	**100**	**100**	**100**	**100**	**100**	**100**	**100**	52.3	90.3	98.7	**100**	**69.0**	85.0	**100**	97.0	51.7	74.8	**96.8**	**99.3**
CLIPORT w/o Lang	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**100**	**88.7**	**99.0**	**99.7**	**100**	59.0	98.0	99.0	99.0	**71.0**	**92.0**	95.3	97.8
Transporter (multi) [2]	98.0	99.0	**100**	**100**	91.5	99.5	**100**	**100**	49.6	79.6	96.3	92.9	50.0	**99.0**	99.0	**100**	16.3	37.3	36.0	26.7
CLIPORT w/o Lang (multi)	0.0	99.0	**100**	**100**	0.0	94.7	**100**	92.5	0.0	57.6	85.9	75.3	0.0	86.0	98.0	**100**	0.0	66.0	80.8	77.7

	palletizing-boxes				assembling-kits				packing-boxes				manipulating-rope				sweeping-piles			
	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000	1	10	100	1000
Transporter [2]	**91.6**	**99.0**	**99.9**	**99.9**	33.2	67.4	**98.2**	**100**	88.6	96.0	98.2	**100**	62.7	78.5	93.7	97.8	98.8	100	99.9	99.8
CLIPORT w/o Lang	89.4	98.6	99.6	99.4	**52.8**	83.2	92.8	97.8	**96.9**	**99.5**	**100**	**100**	**69.4**	**93.6**	**97.9**	**100**	**99.2**	**100**	**100**	**100**
Transporter (multi) [2]	90.7	98.7	99.7	99.1	22.6	58.6	66.8	68.8	93.4	96.6	**100**	**100**	34.3	68.7	87.2	83.7	92.5	97.0	95.6	97.3
CLIPORT w/o Lang (multi)	0.0	61.1	94.9	86.4	0.0	**86.6**	95.2	89.0	0.4	98.8	99.3	**100**	0.4	90.0	85.2	93.2	6.5	99.8	**100**	**100**

Table 6. **Demo-Conditioned Tasks.** Validation task success scores (mean %) from 100 evaluation instances vs. # of demonstration episodes (1, 10, 100, or 1000) used in training.

To investigate if our framework can be applied to demo-conditioned tasks that do not require language instructions, we run evaluations on the original Transporter tasks [2]. Table 6 compares our two-stream architecture without language conditioning to Transporter. Our method outperforms Transporter in $30/40 = 75\%$ of the evaluations in Table 6, especially in low-data regimes with 100 demonstrations or less. Particularly for the assembling-kits and manipulating-rope tasks, the two-stream architecture shows significant performance gains. We hypothesize that this is because the CLIP-ResNet model provides a strong visual prior on object representations for learning generalizable policies.

H Affordance Prediction Examples

Figure 10 showcases more examples of affordance predictions from trained CLIPORT (multi) models. Traditional object-centric representations like pose and instance segmentation generally struggle to represent piles of beans or squares on a chessboard. In such cases, a single detector would have to be trained (with supervision data) to detect every bean and square on the chessboard, which is often infeasible, especially in multi-task settings.

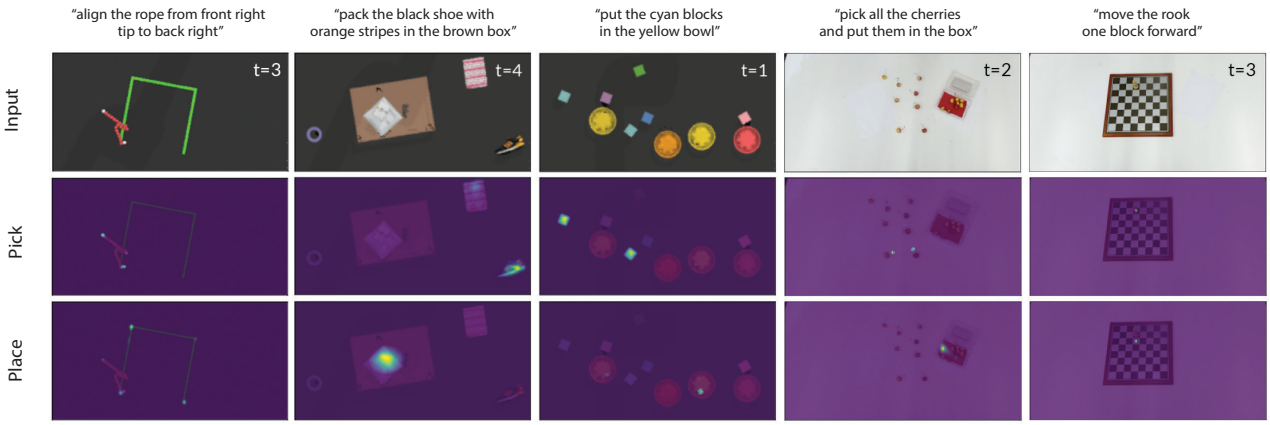

Figure 10. More examples of pick and place affordance predictions from CLIPORT (multi). The left three columns are from simulated tasks, and the right two columns are from real-world tasks.

I Limitations and Risks

While CLIPORT is highly capable, it is not without issues. In the following sections we discuss various limitations and risks of using CLIPORT for real-world manipulation.

Balanced Datasets. CLIPORT can learn generalizable policies from very few demonstrations, but it relies heavily on a balanced training dataset with a good converge of expected skills and invariances. As discussed in Section 4.3, the model will exploit any bias, e.g. always place "yellow blocks" inside 'blue bowls" if that is the only example of yellow blocks that it's provided with. Sometimes these biases can be hard to spot since everything (from perception to action) is trained end-to-end through demonstrations. During our real-world experiments we ended up iteratively refining some datasets after finding such biases during execution.

Hand-Eye Calibration and Closed-Loop Control. The execution of policies is sensitive to the accuracy of the hand-eye calibration. The action-space of CLIPORT is 2D pixels with yaw-rotations. Translating these pixel coordinates to end-effector poses relies on carefully calibrated extrinsics between the robot's base frame and the RGB-D camera. Further, while the framework takes closed-loop actions across discrete pick-and-place timesteps, the execution of each pick and place primitive itself is open-loop. This restricts usage to mostly quasi-static tasks and leads to issues if objects move while the robot is executing a pick or place primitive. Future works could incorporate a separate visuo-servoing mechanism for more robust grasping.

Dexterous Manipulation. Extending CLIPORT's action-space to 6-DOF or N-DOF control for dexterous non-quasi-static manipulation is non-trivial. The $\mathbf{SE}(2)$ action-space is one of the key factors that make Transporter and CLIPORT highly data efficient. Since the actual end-effector control is abstracted away, the model can easily reason about high-level affordances at discrete timesteps, but at the price of loosing dexterity. Similarly, extending $\mathbf{SE}(2)$ equivariance to $\mathbf{SE}(3)$ equivariance is also non-trivial. Cross-correlating in voxelized 3D spaces might be expensive and slow.

Grasping Novel Objects. CLIPORT has some limited capacity in grasping unseen instances of objects in one-shot or few-shot settings. While CLIP is a pure vision-language model with no understanding of affordances, actions, or physical properties, in CLIPORT we fine-tune CLIP's visual representations in the **semantic** decoder layers to produce visual affordance predictions – like grasping pliers by the handle. We illustrate this in Figure 11 with an example of one-shot learning. Despite having seen just a single training example with pliers, CLIPORT is able to correctly grasp the handles of 2/3 unseen pliers of different shapes, sizes, and colors. The model fails in Test 3 where the instance is significantly outside the training distribution. But even so, the model is able to correctly localize the pliers among the distractor objects, and with a few more training examples it might be able to correctly grasp the instance. In contrast, RN50-BERT struggles to identify pliers with just a single example since pliers are not part of the 1000 ImageNet classes [66]. Further, without the appropriate language goal to condition the policy, e.g. when provided with a nonsensical object name like "dax", the model falls back to the most familiar object seen during training.

Grounding Complex Object Relationships. In general, CLIPORT struggles with complex object-relationships that require reasoning about several objects. The model performs poorly on assembling-kits-seq tasks that involve grounding spatial relationships like "middle" with unseen shapes and language. The model's capacity to infer these relationships purely from dense global features might be limited. Also, CLIPORT cannot count objects since it does not maintain a history or belief across timesteps, thus limiting instructions to 'any' or 'all' quantifiers. Future works could explore neuro-symbolic [67] or attention-based [68] methods for better generalization to novel object-relationships.

Scope of Language Grounding. CLIPORT's understanding of verb-noun phrases is tightly grounded in the demonstrations and tasks seen during training. For instance, an user could have used "sort out all the Mars bars from the pile and put them in the yellow bin" while demonstrating a task. Here the model only understands 'sort' in the context of separating something from the pile and putting it in a bin, and not in the most generic sense that is applicable in any context, like sorting numbered blocks in descending order.

Task Completion. CLIPORT relies on an expert to indicate task-completion. For real-world tasks, this means the model keeps taking actions until an user stops the execution. Future works can address this issue by training a success classifier [2] to predict task completion from RGB-D observations.

Risks from Pre-Trained Models. CLIP was trained with massive amounts of "in-the-wild" image-caption pairs from the internet. This makes it prone to unchecked biases and associations [59, 69] that can be harmful to certain individuals and communities. The end-to-end framework is also vulnerable to adversarial attacks [59] that try to maliciously affect the model's behavior. These issues are further exacerbated by the fact that we use CLIP's representations to take actions with a physical robot. For safe deployment in the real-world, keeping humans in the loop – both during the training phase and while instructing the robot, might help in mitigating some of these issues and potential risks.

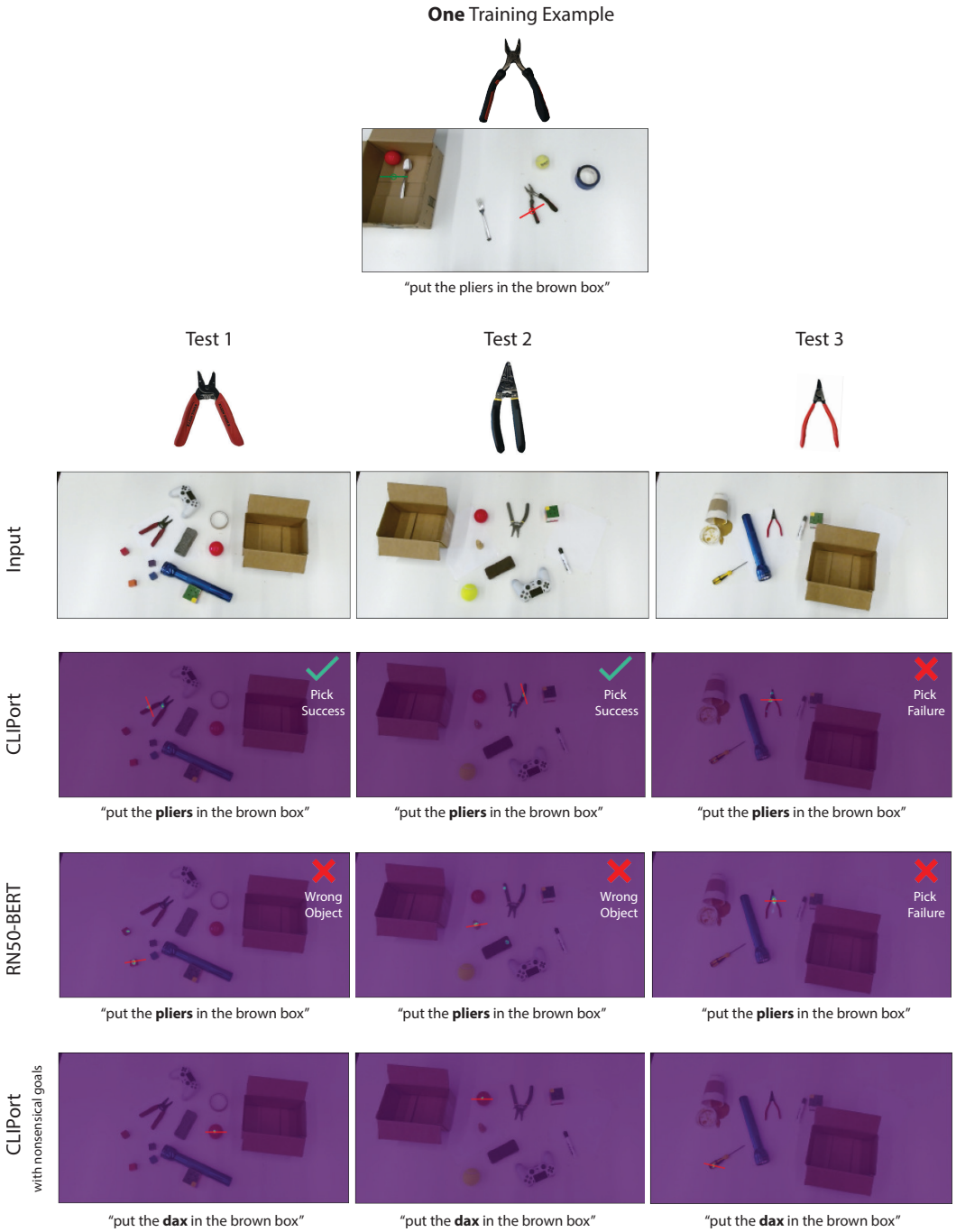

Figure 11. **One-Shot Learning**. Selected examples of grasping pliers with CLIPORT, RN50-BERT, and CLIPORT with nonsensical goals.