

A Method Implementation Details

A.1 LOReL

We begin by describing the method implementation details for our method LOReL.

A.1.1 Reward Function

The reward function is trained as a binary classifier which takes as input the initial state s_0 , the current state s , and language instruction l , and outputs a scalar $[0,1]$ prediction.

Architecture. The network first concatenates the initial image (64,64,3) and goal image (64,64,3) channel wise. The resulting image is passed through a convolutional image encoder with ReLU activations with the following architecture [channels, kernel size, stride] (all with padding 1): [[32, 4, 2], [32, 4, 1], [64, 4, 2], [64, 4, 2], [128, 4, 2], [128, 4, 1], [256, 4, 2], [256, 4, 1]]. The output is flattened and passed through fully connected layers of size [512, 512, 512, 512, L] where L is a hyperparameter for the image embedding size. In our real robot experiments the observations have 4 camera views, each (64, 64, 3) so we concatenate each initial image and goal per view, then use the same convolutional architecture but with 4 groups.

Simultaneously, the language instruction is passed through a pretrained distilbert-base-uncased sentence encoder [72], available at <https://huggingface.co/distilbert-base-uncased>. The network performs sub-word tokenization using the distilber tokenizer, then passes the sentences through the 15 layer transformer network, outputting a real valued vector of size 768.

The resulting image encoding of size L and the sentence encoding of size 768 are concatenated, and fed through a fully connected network of size [L , L , L , 1] where the first three layers have ReLU activations and a 0.2 dropout, and the final scalar prediction goes through a Sigmoid activation.

Hyper-parameters. In our experiments we train the reward with $L = 128$, batchsize of 32, and Adam optimizer with learning rate 0.00001. In our simulated experiments we use $\alpha = 0$, that is, we only use the initial and final states of the episode for positives. In our real robot experiments we use $\alpha = 0.25$, selecting positives with initial state in the first quarter of the episode and final states from the last quarter of the episode.

Augmentation. To prevent the classifier from over-fitting, we use visual data augmentation in the form of color jitter (brightness=0.02, contrast=0.02, saturation=0.02, hue=0.02) and affine transformations of up to 20 pixels on the images (translate=(0.1, 0.1), scale=(0.9, 1.1)). We also add uniform random noise in $[-0.1, 0.1]^{768}$ to the instruction embeddings.

A.1.2 Visual Dynamics Model (Sim)

In our simulation experiments, we train an action-conditioned video prediction model using Stochastic Variational Video Prediction [73]. We train it on the full simulated dataset of 1M frames for 300000 iterations with all default hyperparameters using the codebase <https://github.com/tensorflow/tensor2tensor>.

A.1.3 Visual Dynamics Model (Real Robot)

On the real robot setup we leverage a pre-trained visual dynamics model on the robot desk setup using the GHVAE [74] architecture, that trains a VAE, encodes each of the 4 camera views, and learns a predictive model in the latent space.

A.1.4 MPC Planner

Finally for choosing actions with the learned reward and visual dynamics model we use model predictive control. Specifically, in simulation we optimize a single sequence of 20 actions which are stepped in the environment. We sample 200 action sequences of length 20, which are fed through the video prediction model and ranked according to the reward of their final state. The top 10% are used to refit the action sampling distribution, and the process is repeated 3 times before the best action sequence is stepped in the environment.

On the real robot, we optimize action sequences of length 5, using again 3 iterations of CEM, now with 48 sampled action sequences per iteration. After stepping each 5 step action sequence we step the agent in the environment, then repeat for the full 30 timestep episode. On the real robot, we use the same initial state $T=0$ as the initial state throughout the full episode.

A.2 Language Conditioned Imitation Learning

The language conditioned imitation learning agent takes in the current state s and language instruction l from that episode and is trained to minimize the mean squared error to the action taken in the data.

Architecture. This agent uses an identical image encoder, and identical distilbert sentence encoder as LOReL. The resulting embeddings are again concatenated and fed through a fully connected network of size $[L, L, L, A]$ where the first three layers have ReLU activations and a 0.2 dropout, and the final prediction outputs the A dimensional action. **Hyper-parameters.** This agent is trained with $L = 128$, batch-size of 32 and an Adam optimizer with learning rate 0.0001. **Augmentation.** The BC agent uses the same visual augmentation as LOReL. Unlike LOReL which struggles with over-fitting to language instructions, the BC agent struggles with under-fitting and learning different behavior for different instructions, hence we omit the uniform noise in language embedding space.

A.3 Language Conditioned Reinforcement Learning

The language-conditioned Q learning agent learns a language conditioned Q function which takes as input the initial state s_0 , current state s , action a , and language instruction l and aims to predict the discounted return for taking action a in state s . It is trained on balanced batches of positive transitions which are terminal states with reward 1, where the current state is the final state for the episode which was labeled with instruction l , and negative transitions where the current state is any state earlier in the episode and has reward 0. The Q function is then trained with the standard Bellman error, where target Q values which maximize s_{t+1} are compute by sampling M actions and taking the one with the highest Q value. During evaluation at each timestep the agent selects the action which maximizes the Q value at the current state via sampling M actions and taking the best one.

Architecture. The Q function uses an identical architecture to the LOReL classifier, with the exception that it also takes as input the action a which is concatenated at each of the final 3 fully connected layers. **Hyper-parameters.** We use $L = 128$, Adam optimizer with learning rate 0.0001, $M = 100$, and batch size 8. **Augmentation.** Like the BC agent, the Q-learning agent uses the same visual augmentation as LOReL but omits noise in the language embedding space to help the policy learn to differentiate between different instructions.

A.4 Oracle, Random, Pixel, LPIPS Baseline

The Oracle performance is computed using the exact same planner as LOReL, but uses a ground truth dynamics model and a ground truth cost computed on the environment state. The random policy simply samples random actions at each time steps. The Pixel baseline first receives a goal image in which the object has been moved to its desired goal position (which constitutes task success). It then uses the exact same visual dynamics model and planner, but with a cost function which minimizes the ℓ_2^2 pixel distance between the goal image and the final image in each trajectory. The LPIPS baseline uses the same planner, dynamics, and goal image, but uses LPIPS similarity instead of negative pixel distance.

B Environment/Data Details

B.1 Simulation Environment

The simulated environment is built off of Meta-World [56], and consists of a simulated sawyer arm interacting over a table with a drawer, a faucet, and two mugs. The agent is controlled using delta end effector control, and the episode horizon in the environment is 20 timesteps. The observation space is (64, 64, 3) RGB images.

B.2 Simulation Data and Annotation

We collect 50000 episodes in the simulated environment (each 20 timesteps) for a total of 1M frames. Data is collected by a random policy which samples uniformly in the action space. During data collection the arm and scene is reset randomly each episode. To annotate each episode, we procedurally look at the true environment state at the beginning of the episode and at the end of the episode and record the change in position of the objects (including the drawer, faucet, and mugs). For each object change in state we generate a phrase based on the measured change, such as “move the black mug left” or “turn the faucet right”. Then all the phrases for a single episode and shuffled and combined with an “and” to create the final annotation for that episode. If no object moves the episode is annotated with “Do nothing”. Doing so produces around 2300 unique annotations total.

B.3 Robot Environment

The robot environment consists of a real Franka Emika Panda robot operating over an Ikea desk which consists of a cabinet, 2 drawers, and numerous objects. The robot has 4 camera views, one one which is a side view of the table, one which is a birds eye view from the right side of the table, one which is a birds eye view from the left side of the table, and one wrist mounted camera. Each returns (64,64,3) RGB images. The robots action space includes delta end-effector control (up to 7cm) and grasping (however we do not use the grasping capability in any of our evaluation tasks).

B.4 Robot Data and Annotation

Our robot dataset consists of 3000 episodes, each 50 timesteps, taken directly without modification from concurrent work [78] which aims to learn a diverse set of skill using online RL. As a result, this data is from an autonomously trained RL agent, and contains a mix of meaningful and close to random behavior. Moreover, even when tasks are completed they may be completed in highly sub-optimal ways. Some of the tasks the agent is trained for include opening the drawers and cabinets, grasping and placing different objects, and inserting markers/plugs.

We then have each episode annotated using crowdsourcing, specifically Amazon Mechanical Turk (See Figure 8). We collect 2 annotations per episode for a total of 6000 annotations. During training, we filter out episodes where the annotators said the robot did nothing or they could not tell what the robot was doing, resulting in a final dataset of 1600 episodes. We visualize the top 25 annotations and their counts in the filtered dataset in Figure 9.

C Experimental Evaluation/Task Details

Next we describe the experimental setup in detail, specifically the tasks and their evaluation criteria.

C.1 Simulation Tasks.

The simulation tasks used for evaluation are (1) closing the drawer, (2) opening the drawer, (3) turning the faucet left, (4) turning the faucet right, (5) moving the black mug right, and (6) moving the white mug down. For language conditioned approaches these tasks are specified with “close drawer”, “open drawer”, “turn faucet left”, “turn faucet right”, “move black mug right”, and “move white mug down” respectively. For goal image comparisons, we generate goal images with the object moved to a position which satisfies the task, see Figure 10 for examples. For all drawer and mug tasks success is if the object moves in the correct direction by at least 2cm at any point in the episode. For the faucet it is if it rotates at least $\pi/10$ radians in the right direction at any point in the episode.

C.2 Experiment 1 Details

For experiment 1 we compare each of the methods on the 6 tasks described previously. Using the same 50000 episode dataset we train 3 seed each of our method, the imitation baseline, and the RL baseline. Our method and the imitation baseline are trained for 390000 iterations, while the RL agent is trained until the Bellman loss plateaus at around 240000 iterations. For each task and each seed we conduct 100 random trials (with random initialization) to compute a success rate out of 100. For the oracle, pixel, and random we directly run 300 random trials per task.

C.3 Experiment 2 Details

In experiment 2 we consider the same setup as experiment 1, but rephrase the instructions as described in the main text. The full set of rephrased instructions can be found in Figure 11. For the human provided unseen natural language, we sample one of the provided instructions at random for each episode.

C.4 Robot Tasks

On the real robot, we consider 5 tasks, (1) opening the left drawer, (2) opening the right drawer, (3) moving a stapler, (4) reach the markers in the left drawer, and (5) reaching the cabinet.

The success criteria for task (1) and (2) is to open the drawer at least 1 inch from its initial position at any point in the episode. For task (3) it is to translate the stapler on the table by any amount. For task (4) it is to reach the gripper tip within 1 inch of any marker in the left drawer. For task (5) it is making contact with the cabinet.

The initial state for task (1) is the robot above and to the right of the left drawer, and the drawer starts slightly open to make grasping easier. The initial state for task (2) is above and to the left of the right

Instructions: Given a video of a robot, write a sentence summarizing its behavior.

Write the sentence in the form of a command (i.e. "pick up the stapler" rather than "picking up the stapler").

Note: you can see the robot's behavior from 3 different camera viewpoints.


Some examples:

"Open the left drawer" **NOT** "Opening the left drawer",
 "Reach the marker" **NOT** "Reaching the marker",
 "Push the stapler up" **NOT** "Pushing the stapler up",
 "Insert the plug into the socket"
 "Open the cabinet"

Only write down an instruction that the robot successfully completed.
 For example, if the robot tries to pick up the marker and gets close but fails, you should put "Reach to the marker" or "Grasp next to the marker" rather than "pick up the marker".

If the robot does not successfully complete any task, write something like:
 "Do nothing"
 "Wave the arm in the air"

If you can't see the robot or can't tell what the robot is doing, write:
 "NA"



Write a command which describes the robot's behavior in the video...

Submit

Figure 8: **Annotation interface.** The annotators are shown a video of the episode, and are given the above instructions. We collect 6000 annotations, 2 for each of the 3000 episodes of data.

drawer, and again the drawer starts slightly open. For task (3) the robot starts near the middle/right side of the desk, and the stapler is initialized randomly on the front half of the desk. For task (4) and (5) the robot has the same initialization as in task (1), and in task (4) the markers are randomly moved around within the left drawer. For each episode of each task the initial position is randomized up to 5cm in any direction from the main initial position. All resets and successes are measured by human supervisor. See the website <https://sites.google.com/view/robotlore1> for videos of task completion.

C.5 Experiment 3 Details

In the real robot experiment we conduct 10 trials of each task for our method and for an ablation of our method trained without flipped initial/final negatives. The episode horizon for each task is 30 timesteps, and the agent plans 5 actions at a time as described in Section A.1.4. We also run 10 trials of tasks (1) and (3) with more sophisticated instructions.

Instruction	Count
open the left drawer	140
open left drawer	101
reach the stapler	93
reach for the marker	70
open the drawer	55
open the right drawer	51
reach the socket	48
insert the plug into the socket	46
reach to the marker	44
open right drawer	44
reach the green marker	41
reach the plastic hole with the green marker	41
reach for the stapler	34
reach the left drawer	34
reach to the stapler	32
open drawer	31
reach for marker	29
open and close the drawer	28
reach the plastic hole using the green marker	26
open and close the left drawer	25
grasp next to the marker	25
move the stapler around on the table	24
reach the black marker	23
open the cabinet	22
grasp next to the stapler	21

Figure 9: **Top Instructions.** We list the top 25 instructions and the number of times they appear in the data in the filtered robot dataset.

D Additional Results

D.1 Additional Ablations

First, we run an additional ablation in simulation which ablates the effect of both types of negative examples. When removing the randomly chosen negatives from different episodes, we see a dramatic drop in performance (average success rate 56% to 27%), as this is the main way the agent learns the effect of different language instructions. When removing the flipped negatives in simulation, we see limited impact (average success rate 56% to 55%). This is because the primary role of the flipped negatives are to prevent over-fitting to objects in the scene, and to capture temporal progress, and in simulation the scene is fixed, and has a large amount (50K) episodes of training data. This is in contrast to the real robot, with 1.6K episodes, and many different scenes, and in that setting removing the flipped negatives drops average success rate from 66% to 36%.

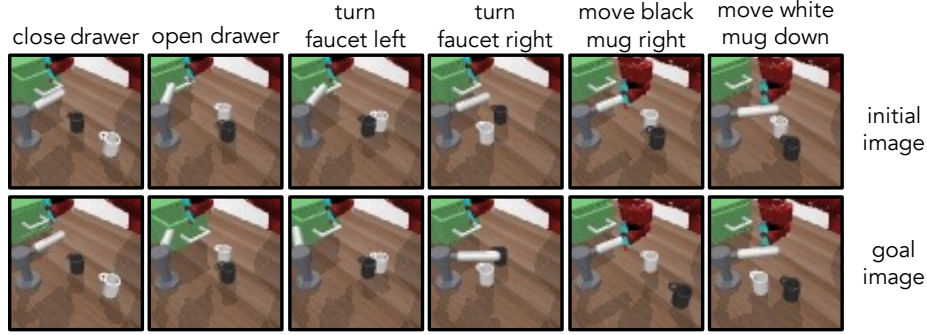


Figure 10: **Example Initial/Goal Images.** Examples of generated goal images for the Pixel baseline.

Second, In Table 3, we run two more ablations on the real robot, where we consider (1) removing the data cleaning/filtering step where we remove episodes where the annotators indicated the robot did nothing (**LOReL (- Filter)**) and (2) where we do not include noisy positives to generate more positive examples during training (**LOReL (- NP)**). First, we find removing filtering reduces performance considerably. In many cases annotators could not tell what the robot was doing, or wrote “do nothing” when they were unsure, making these labels particularly noisy. As a result, we filtered out these episodes primarily as a way of cleaning out noise in the data. While our simulation results suggest that our method can handle some behavior that completes no task ($\sim 20\%$ of the dataset), in the extreme case where almost half the data is labeled as not doing a task (as is the case in the real robot data), it can make learning an effective reward more difficult. Specifically, episodes of “do nothing” may dominate training batches and negative examples, making it harder to learn the difference between different language instructions, especially instructions that are infrequent in the data. In practice, post-hoc cleaning/filtering of the data requires little to no extra supervision, and can make training the reward considerably easier. Second, we find that generating more positive examples via noisy labeling is important in the robot domain, where there are only 1.6K episodes of data (after filtering/cleaning).

Task (10 Trials Each)	LOReL	LOReL (-Filter)	LOReL (-NP)
“Open the left drawer”	90%	70%	30%
“Open the right drawer”	40%	20%	30%
“Move the stapler”	50%	10%	70%
“Reach the marker”	70%	0%	10%
“Reach the cabinet”	80%	30%	20%
Average over tasks	66%	26%	32%

Table 3: **Additional Real Robot Ablations.** Our ablations suggest that both filtering out episodes which annotators labeled as “do nothing” and training with noisy positives are important for real robot performance.

D.2 LOReL Training Curves

In Figure 12 (left) we include the train/test accuracy curves for LOReL in simulation, as well as the ablations of LOReL in simulation (LOReL, LOReL without the randomly chosen negatives, and LOReL without the flipped negatives). As expected, removing random negatives makes fitting the data much easier, but leads to a worse reward and worse planning performance. In Figure 12 (right) we include the train/test accuracy curves for LOReL on real robot data, including LOReL, LOReL without filtering, LOReL without flipped negatives, and LOReL without noisy positives). We see that without filtering, fitting the data is much more difficult, and without the noisy positives, fitting the data is easier (but can also leads to a worse reward function for planning). For all ablations the last checkpoint before test accuracy starts decreasing is used in planning.

D.3 Unseen Instruction Generalization Results

In Figure 13 we include the per task success rates for our method when using the original command vs. unseen variations.

D.4 Qualitative Examples

We include qualitative examples of the ranked predicted trajectories under different language instructions on the real robot in Figures 14, 15, 16, 17, and 18.

Instruction	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6
Seen	Close drawer	Open drawer	Turn faucet left	Turn faucet right	Move black mug right	Move white mug down
Unseen Verb	shut drawer	pull drawer	rotate faucet left	rotate faucet right	push black mug right	push white mug down
Unseen Noun	close container	open container	turn tap left	turn tap right	move dark cup right	move light cup down
Unseen Verb+ Noun	shut container	pull container	rotate tap left	rotate tap right	push dark cup right	push light cup down
Human Provided	Push the drawer shut Push the drawer Shut the drawer shut drawer Slide the drawer closed Shut the drawer shut the dresser Shut the drawer. Shut the cupboard	Pull the drawer open Pull the handle Pull the drawer handle Pull the drawer open Pull open the drawer open the dresser Pull the drawer. Unclose the cabinet	Rotate the tap counterclockwise Turn faucet away from camera Rotate nozzle left faucet counterclockwise Rotate the faucet left Turn the faucet to the left rotate handle to the left Turn faucet counter clockwise. Spin nozzle left	Rotate tap clockwise Turn faucet towards camera Rotate nozzle right faucet clockwise Rotate the faucet right Turn the faucet to the right rotate handle rightward Turn faucet clockwise. Twirl valve right	Translate the black cup to the right Move black mug away from drawer Push black cup right black mug right Slide the black mug right Move the dark mug to the right push black cup right Move black mug right. Shift dark cup right	Translate the white cup down Move with mug closer to the faucet Bring white cup down white mug down Push the white mug down and left Move the lighter mug down shift white mug down Pull white mug to the front. Reposition white glass down

Figure 11: Example Instructions used in Experiment 2.



Figure 12: **LOReL Learning Curves** Learning Curves for LOReL and ablations in simulation (left) and on the real robot data (right).

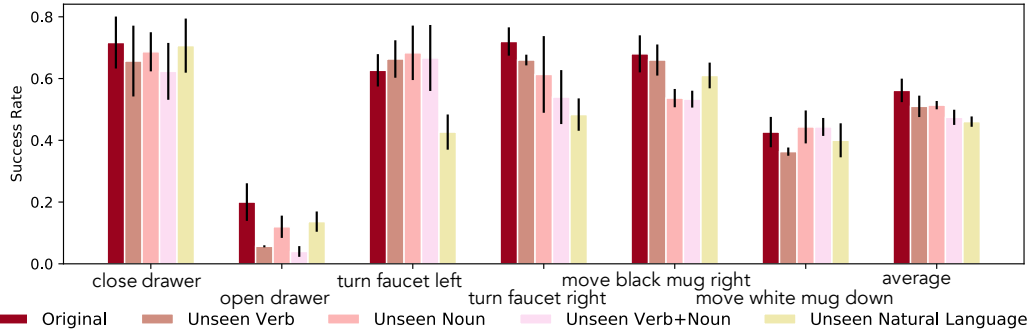


Figure 13: **Generalization to Unseen Commands.** We observe that by leveraging pre-trained language models, LOReL is able to generalize to unseen instructions with unseen verbs, nouns, and human generated instructions with a minimal ($\leq 10\%$) drop in performance. Success rates computed over 3 seeds of 100 trials.

“Open the left drawer”

Best

Worst

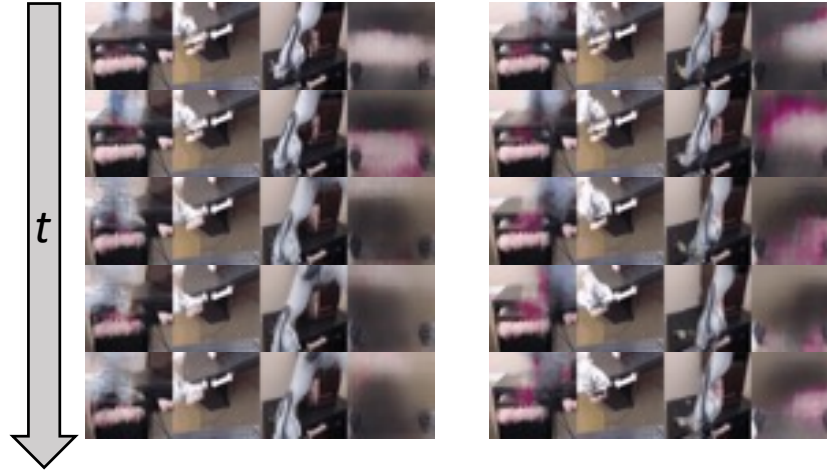


Figure 14: **LOReL Ranking**. Top and bottom ranked trajectories for “Open the left drawer”.

“Open the right drawer”

Best

Worst

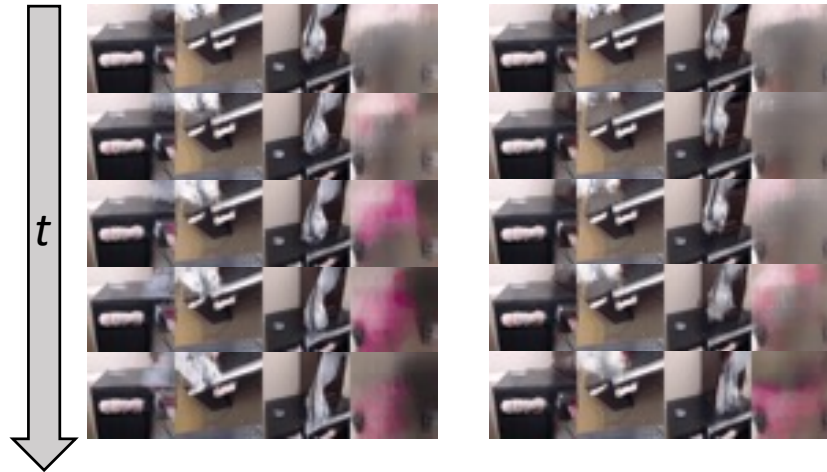


Figure 15: **LOReL Ranking**. Top and bottom ranked trajectories for “Open the right drawer”.



Figure 16: **LOReL Ranking**. Top and bottom ranked trajectories for “Move the stapler”.



Figure 17: **LOReL Ranking**. Top and bottom ranked trajectories for “Reach the marker”.



Figure 18: **LOReL Ranking**. Top and bottom ranked trajectories for "Reach the cabinet".