11-785: Vision-and-Language Manipulation

Team 53
Kevin Gmelin, Kwangkyun Kim, Sabrina Shen, Jaekyung Song

Abstract

Vision-and-Language Manipulation (VLM) describes a task where an agent that relies on vision to perceive the environment is required to follow language instructions to conduct robotic manipulation. These features allow manipulation agents to be commanded through intuitive language commands rather than requiring unique reward functions to be engineered for each task and eliminate the need for agents to be conditioned on harder-to-supply goals like image goals. For this project, we reimplemented a state-of-the-art VLM model, CLIPort, detailed in Shridhar et al. [2022]. In addition to the model implementation and training, we also evaluated two different VLM benchmarks and made modifications to reduce the computational load.

1 Introduction

Humans are often able to apply concepts learned from limited examples to a broader set of tasks. Furthermore, due to an intuitive understanding of language commands, people are generally able to accomplish these tasks without the need for explicit geometric or kinematic definitions of the environment and tasks. While developments in end-to-end, vision-based manipulation have allowed embodied AI to complete tasks involving precise spatial reasoning, they often fail to generalize these abilities when faced with new tasks as they gain no understanding of the semantics that inform the processes being learned. On the other hand, the generalized semantic representations of language have been significantly developed in isolation from the spatial reasoning required to inform robot dexterity. VLM combines vision-based manipulation with language as a goal input. The CLIPort algorithm from Shridhar et al. [2022] outlines an end-to-end model that has been able to demonstrate various tabletop tasks such as packing and folding from language and RGB-D inputs. For this project, we plan to reimplement this model. We will then evaluate our model in Ravens, a benchmark for VLM tasks from Zeng et al. [2020], which provides support for auto-generating training demonstrations labelled with language commands and tools to evaluate the success of VLM agents on various tasks. Additionally, our work included implementation of the training and validation pipeline, simulation setup for visualization and evaluation, and parameter tuning on our model.

2 Related Works

2.1 Transporter Networks for Vision-Based Manipulation

Zeng et al. [2020] proposed a very sample-efficient end-to-end imitation learning architecture known as a transporter network. In this network, the agent learns a policy for generating actions from images, where the action space consists of a pick pose and a place pose. First, a fully-convolutional network is used to generate a pick affordance map. The argmax of this affordance map is chosen as the pick. Then, a crop around the chosen pick is taken. Generating a pick-conditioned place is framed as a template matching problem where deep features from the pick crop are correlated with deep features from the overall image and the highest correlation is chosen for the place.

Preprint. Under review.

The work also introduces a new benchmark for testing vision-based manipulation models, Ravens. The Ravens benchmark allows for easy integration of a model into a physics simulation engine, which not only allows testing of how well a model can perform certain tasks, but also facilitates the generation of training data for various vision-based tasks. In this work, Ravens is used to generate various demonstrations for different tasks, which is used to train the CLIPort model.

2.2 Language-Conditioned Manipulation

In Jang et al. [2022], an imitation learning system is trained from raw demonstrations and human interventions collected using a VR system. It was shown that such a system could learn a policy conditioned on a language command and that pre-trained language embeddings can be effectively used for task conditioning without additional training of the sentence encoder. In Lynch and Sermanet [2020], the authors extended the Learning from Play algorithm of Lynch et al. [2020] to use multi-contextual imitation learning. They demonstrated that a language-conditioned agent could be trained with significantly less language-labelled demonstrations by training a single agent that could seamlessly switch between following image and language goals. In Shao et al. [2021], a language-conditioned agent was trained using a two stage process. In the first stage, a video-based action classifier was used to provide a reward signal to train reinforcement learning agents for specific tasks. In the second stage, the individual reinforcement learning agents are distilled into a single language-conditioned policy by imitating the single-task policies with language as a goal-condition. The authors found that the staged approach worked better than directly training a multi-task agent. The current state-of-the-art in vision-and-language manipulation is CLIPort Shridhar et al. [2022]. CLIPort combines the language-vision semantic understanding from CLIP (Radford et al. [2021]) with the spatial reasoning of transporter networks. CLIPort uses a pretrained CLIP network for embedding a vision observation and a language goal. It uses these embeddings with the transporter network to learn a language-conditioned, vision-based policy from demonstrations labelled with language commands.

2.3 VLM Benchmarks

The CLIPort algorithm from Shridhar et al. [2022] was originally tested in the Ravens benchmark, which was introduced by Zeng et al. [2020] as a vision-based manipulation benchmark. More recently, there have been various benchmarks released specific to vision-and-language manipulation, such as ALFRED from Shridhar et al. [2020] and CALVIN from Mees et al. [2022]. We investigated using the recently released VLMBench from Zheng et al. [2022]. This benchmark provides an environment for development, testing, and training of networks for VLM. It includes an Automatic Manipulation Solver (AMSolver) system to generate demonstrations for a diverse set of tasks which are coupled with language commands, as well as evaluation tools for evaluating the success of a VLM agent on various tasks. In Zheng et al. [2022], a 6D version of CLIPort, which adds additional heads to regress the z height, pitch angle, and roll angle for the pick and place, was also developed and evaluated on VLMBench to provide a baseline. After investigating several benchmarks, we decided to use the Ravens benchmark. VLMBench generates data and executes commands for 6 degrees of freedom, which was unnecessary for our 3 degree of freedom reimplementation of CLIPort. Ravens also has a simpler interface, which allowed faster development, integration, and testing. Lastly, the original CLIPort paper utilized Ravens, which made comparing our reimplementation to the original more straightforward.

3 Baseline Model

The model that we will be comparing ourselves to is the original CLIPort implementation described by Shridhar et al. [2022]. CLIPort is composed of several models, each with their own elaborate architectures, which we'll highlight below.

CLIPort was chosen as a baseline for several reasons. First, CLIPort is the current SOTA in language-conditioned imitation learning for manipulation. Furthermore, due to its transporter-based architecture, we believed that we could train a model to have reasonable performance in comparatively less time than a lot of other SOTA, complex models. Finally, there already exists published results on the CLIPort performance with the Ravens dataset, which can be used as a comparison for our own implementation of CLIPort.

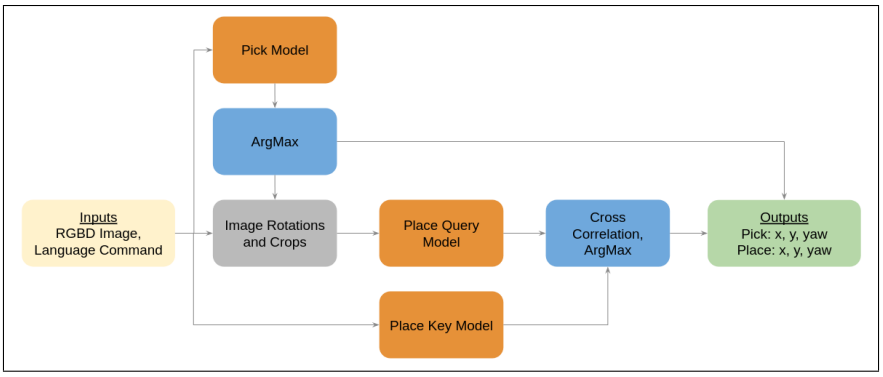


Figure 1: CLIPort Pick and Place: Inference

3.1 Model Overview

CLIPort consists of three models, each one consisting of a semantic stream and spatial stream. The first model is used to generate a pick affordance map, the second is used for generating the place features from the entire scene, and third is used for generating the place features for a small crop around a chosen pick location. The general flow of information between these three models at inference time is shown in Fig.1. The end-effector height is automatically handled by the Ravens environment, but in the real world, this would come from the heightmap and the height associated with the output pick and place locations. The roll and pitch for the pick and place are set such that the robot end-effector stays vertical, perpendicular to the table. Thus, the x, y, and yaw outputs from our model are sufficient for creating desired 6D end-effector poses for the pick and place.

Mathematically, from Shridhar et al. [2022], the model utilizes the orthogonal, top-down camera input o_t and a language command l_t to generate end-effector poses for picking and placing using the following policy π:

$$\pi(o_t, l_t) \rightarrow (\mathcal{T}_{pick}, \mathcal{T}_{place}) \in \mathbf{SE}(2) \times \mathbf{SE}(2)$$

More details on the policy generation process is listed in later sections.

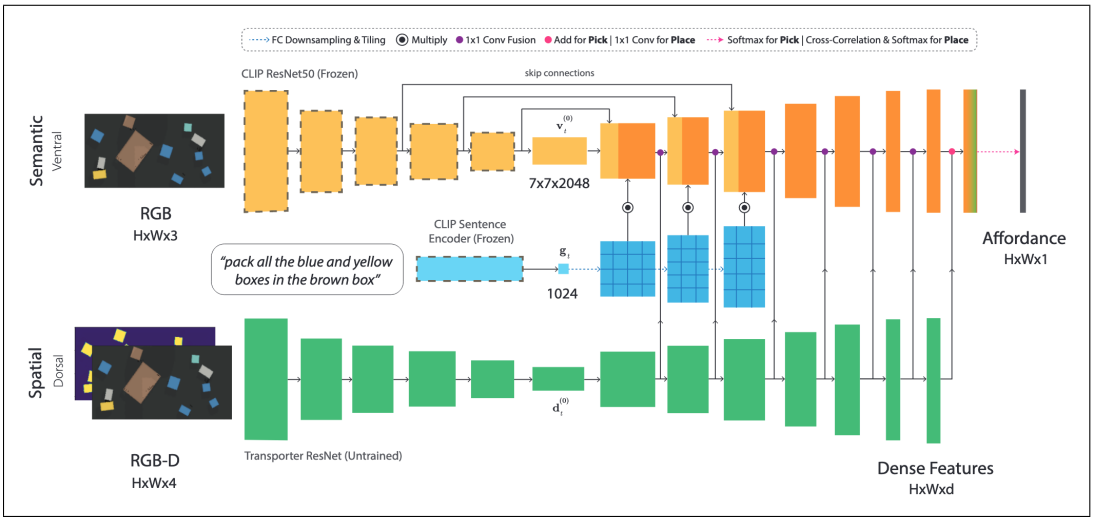


Figure 2: Architecture for the spatial and semantic streams used by the pick and place network in CLIPort. Diagram from Shridhar et al. [2022].

3.2 Spatial Stream

The spatial stream is responsible for imbuing the network with spatial awareness. Specifically, the stream allows the network to understand where objects of interest are. The architecture for the spatial stream is an untrained ResNet with a bottleneck, which encodes and decodes the input RGB-D image of 6 x 320 x 160. The architecture is designed like an autoencoder, where the width and height of the output are the same as the input (the number of output channels is 1 for pick and 3 for place). This structure allows intuitive understanding and visualization of the output, which is an affordance map. In the encoding section, the image size is reduced by a factor of 32, while the number of channels is increased; during the decoding section, the opposite occurs. As the data traverses the decoder, the embedded images are fed laterally into the semantic stream. It is notable that the spatial stream does not take in any inputs besides the initial image; its forward pass does not depend on the semantic stream in any way. A visualization of this process can be seen on the bottom half of Fig.2.

3.3 Semantic Stream

The semantic stream is responsible for generating language and visual features and then fusing these features with features from the spatial stream. The semantic stream uses a pre-trained CLIP transformer to embed the language command, and it uses a frozen pre-trained CLIP ResNet50 model to embed the RGB-D input. The image embedding is passed through multiple upsampling layers. The language embedding is downsampled to multiple different sizes and tiled such that it can be fused with the image embedding at different image upsampling layers. The upsampling of the image is also conditioned on lateral connections from the spatial stream. The output of the semantic stream is a 320 x 160 image of dense features, with 1 channel for the pick network and 3 channels for the place network. A visualization of this process can be seen on the top half of Fig.2.

3.4 Pick Network

The goal of the pick network is to take in the RGB-D image and language command and output a pick affordance map. This pick affordance map estimates the likelihood of a successful pick if the robot picks at a given location in the image. The pick network contains one spatial and one semantic stream. In addition to lateral connections between the spatial and semantic streams, the outputs of the two streams are added together to form the 1 x 320 x 160 affordance map. The argmax of this affordance map is used as the chosen pick location. If the affordance map computed from the inputs is the function $\mathcal{Q}_{pick}$, then the pick pose is:

$$\mathcal{T}_{pick} = \text{argmax}_{(u,v)} \mathcal{Q}_{pick}((u, v)|(o_t, l_t))$$

u, v are the coordinates for the end-effector position in pixel coordinates. Since o_t is the orthogonal projection of the environment, converting u and v to physical coordinates can be done using the camera projection matrix.

3.5 Place Network

The goal of the place network is to take in the RGB-D image and language command and output a place affordance map. The place network contains two sets of spatial and semantic streams. The first set of spatial and semantic streams are used to embed a small crop around the chosen pick into a set of dense features. The second set of spatial and semantic streams are used to embed the entire scene into another set of dense features. The first set of streams can be viewed as a query network, Φ_{query}, while the second set of streams can be viewed as a key network, Φ_{key}. Cross-correlation between the crop features and full image features is performed, with the output being the place affordance map. The argmax of the affordance map is used to select the best pose for end-effector placement, τ:

$$\mathcal{T}_{place} = \text{argmax}_{\tau}(\Phi_{query}(o_t[\mathcal{T}_{pick}], l_t) * \Phi_{key}(o_t, l_t))[\tau]$$

Note that $x[y]$ denotes cropping x about point y.

4 CLIPort Reimplementation Details

4.1 Model Implementation

First, we implemented the conv and identity blocks, which were used to build the spatial stream. Next, we implemented the up block, and made a wrapper for the frozen CLIP models. Finally, we implemented the fully connected and tiling blocks, and used these pieces to build the semantic stream, which incorporated the lateral outputs from the spatial stream. All of these components were implemented as hierarchical classes such that the two final stream architectures could be used to generate pick and place models.

The pick model uses the two stream architecture to generate a 1 channel affordance map while the place model has two variants: key and query. Both place models generate a 3 channel affordance map, but the query network uses cropped and rotated versions of the input RGB-D image, while the key network uses the full, unaltered input. The full CLIPort model incorporates the pick and two place models as follows: first, the pick model was used to determine a pick location by finding the maximum x, y coordinate in the pick affordance map. Then, the two place model outputs, Q_{query} and Q_{key}, are cross correlated to determine a place location. The output of the full model, then, is the x, y, and yaw coordinate for the pick and place operations.

4.2 Simplifying Assumptions

For simplification of model training, we assumed that the quality of picks is invariant to the yaw of the end-effector. For the pick and place operations, a discrete number of rotations are considered for each, which determines yaw. Assuming yaw invariance for the pick model reduces the number of rotations that have to be ran for the pick operation to just 1. This means the yaw angle for picks will be a constant, which we set to 0 degrees. This assumption is valid for our test environment because our simulated manipulator uses a suction cup end-effector. The yaw of the suction cup end-effector does not matter before it has picked up anything, as it is symmetric about its z-axis. If we used a different end-effector, such as a two-finger gripper, then the yaw would matter for a pick operation, and we would have to consider more than one rotation. For the place model, we still consider 12 rotations over 360 degrees of rotation, which gives a resolution of 30 degrees.

4.3 Integration

The model was thoroughly tested on every level before integration with the training pipeline. By unit testing each block, stream, and model we were able to verify that all input and output sizes were correct according to our reference paper. This process ensured smooth integration of the model with itself, as well as the rest of the system.

4.4 Loss Criterion

To train the initial CLIPort model, cross-entropy loss was used to train the x, y, and yaw classification of the pick and place networks, where the ground truth label was a one-hot pixel map. We had to flatten the network output and ground truth label before calculating the cross-entropy loss. This was done so that the class we were trying to predict would become a tuple of x, y, and yaw, which was found to be suitable for imitating pick and place operations.

5 Experimental Setup

In addition to the baseline model development, there were a few other experiments we tried to test the impacts on CLIPort reimplementation and reduce time in training.

5.1 Dataset and Dataloader

The first benchmark we looked at was VLMBench from Zheng et al. [2022]. In order to get the data for training and validation, we had to download the datasets from a public Google Drive link, and utilized the provided dataloaders. However, we found that the dataset included information not relevant to our use case, such as z, roll, and pitch. Additionally, we found the codebase difficult to understand and

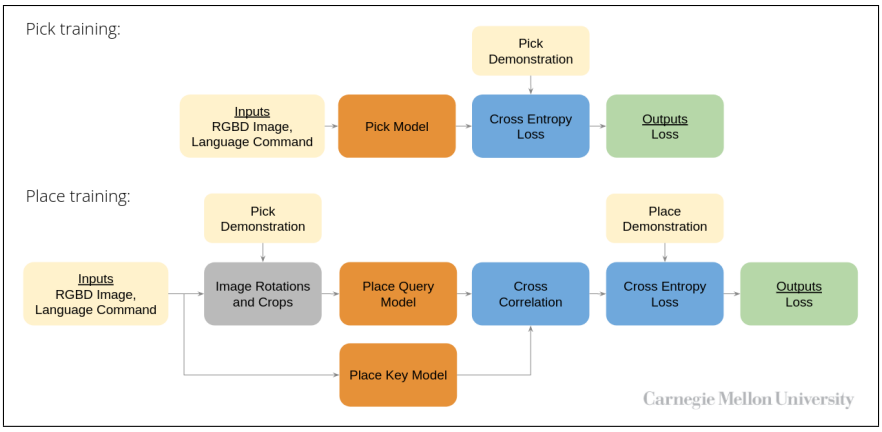


Figure 3: CLIPort Pick and Place: Training

work with. We then switched to using Ravens from Zeng et al. [2020]. This benchmark allowed us to generate our own data using simulation runs, which allowed greater flexibility in how we trained. We also found the dataloaders much easier to set up and run, allowing us to train and evaluate our reimplementation much more quickly. Importantly, the Ravens environment provided a gym interface, which made it very easy to set up evaluation runs. For the dataloader, we set the batch size to 1 as we found various parts of our algorithm did not support variable batch sizes, and we implemented a custom cache limit as the unbounded cache size being used by the original Ravens dataloaders resulted in us running out of RAM during training.

5.2 Training and Validation Pipeline

The Ravens benchmark was used to generate data for training and validation. By running provided scripts, and specifying what task we want to generate data for, we generated 1000 training demonstrations and 100 test demonstrations. We used the provided data augmentation utilities to increase the training data we got from the various simulation runs.

Once we had the data, we were able to start training our pick and place models, as shown in Fig 3. These models were trained independently and simultaneously, as they did not directly depend on each other. For the place model, which requires a pick coordinate, we used teacher forcing, where the ground truth pick location from the demonstration was used instead of our pick model's output. The separate training also made testing and evaluating performance simpler as the two could be trained in parallel. For both models, the pick or place coordinates from the demonstrations were provided as targets, and then inference was run for the models given the RGB-D image and language command. The ground truth and the model outputs were then used to compute a loss, which were used to update the model's parameters.

6 Results

The task we trained our model for was packing specific pairs of blocks into a box, eg. "pack all the gray and purple blocks into the brown box." This task was chosen because it required a thorough understanding of semantics (color, blocks, box), as well as spatial reasoning.

6.1 Performance Metrics and Evaluation

We used success rate as the performance metric, which is the average reward across 100 evaluation runs; for our task, the reward is the fraction of correct blocks in the box over the total number of blocks that should be in the box. The maximum reward and success rate is 1. Each evaluation run consists of up to 20 consecutive actions, with the same language command being used for every step.

We first trained our model for 3,000 training steps, then for 30,000 steps. For the former, we achieved a success rate of 0.38, and for the latter, a success rate of 0.59. The baseline we are comparing to, the

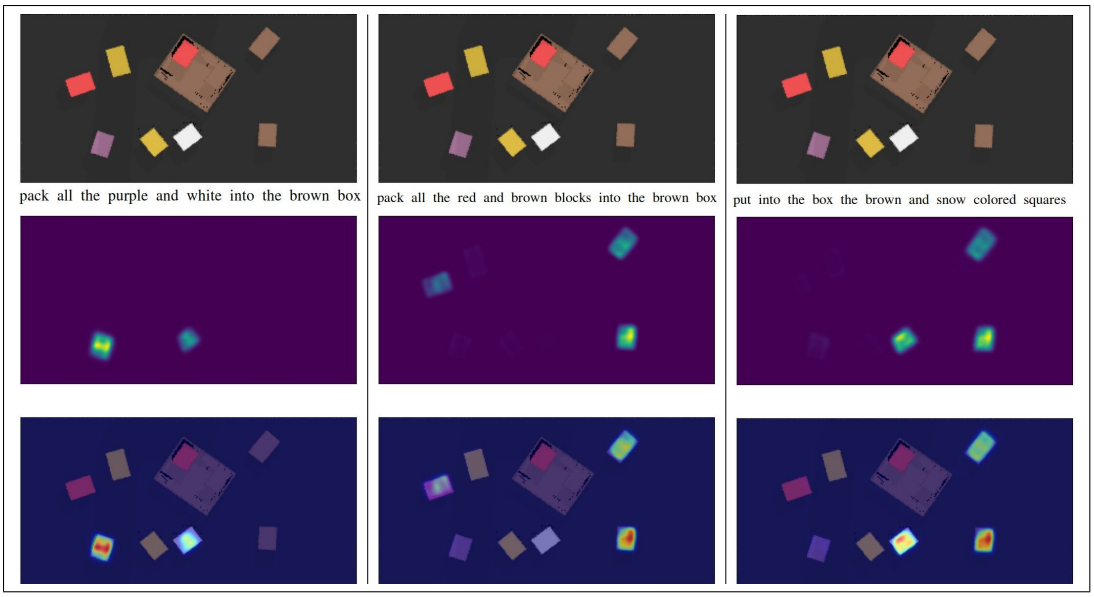


Figure 4: Example pick affordances for three different commands. Top is raw image, middle shows affordances, and bottom shows overlay of affordances onto raw image.

original CLIPort, had a success rate of 0.98 after 200,000 training steps. We were unable to match the training length of the original implementation due to limitations on time, as training for 200,000 steps would take about a week with the hardware available to us. While the original CLIPort paper provides ablations for how the performance varies with the number of training demonstrations, they did not report any results dependent on the number of training steps. Our results provide a glance into the training performance with varied number of training steps. For reference, training for 3,000 training steps took about 2 hours and training for 30,000 steps took about 18 hours. This is significantly faster than the estimated 5 days that it would have taken to train for 200,000 steps.

Another major difference between the original implementation and our implementation is that the original implementation performs a post-training optimization step which evaluates success rate for the checkpoints generated at each training step in order to select the best performing model. Meanwhile, we use the last checkpoint after a fixed number of training steps, which also approximately corresponded with the minimum validation loss. Validation loss is only a proxy for success rate, and so this post-training optimization is expected to have increased the relative success rate of the original implementation.

We believe the major difference in performance is due to a significant difference in time spent on training between our models. Another difference in our implementation is that the original had three copies of frozen CLIP models for its three two-stream architectures, while we passed pointers to a single frozen CLIP model to multiple places. However, the frozen CLIP model does not have any learnable parameters or recurrent layers, so sharing it should not have any adverse effects. Another difference is the original considered 36 rotations for place, but we only consider 12. This factor is unlikely to be the sole explanation for a discrepancy this large.

Qualitatively, we can see that our model is robust and able to handle complex scenarios, as shown in Fig 4. The figure shows three scenarios: left is a typical case of putting blocks in the box, and affordances shows that the correct blocks are being selected. The middle scenario shows the model understands to ignore correct blocks already in the box, as the red block in the box has low affordance. The scenario on the right shows the model can handle language commands that are very different from what it had seen during training, as the pick affordances are reasonable despite a change in phrasing and color description. This generalization can be owed to the use of the pretrained CLIP model. Fig 5 shows the affordance for placing the blocks. Here, we can see that the model is making an effort to place blocks in unoccupied sections of the box, avoiding stacking blocks if possible. Results at the end of three full episode are shown in Fig 6; please see the appendix and our GitHub for more results, images, and videos.

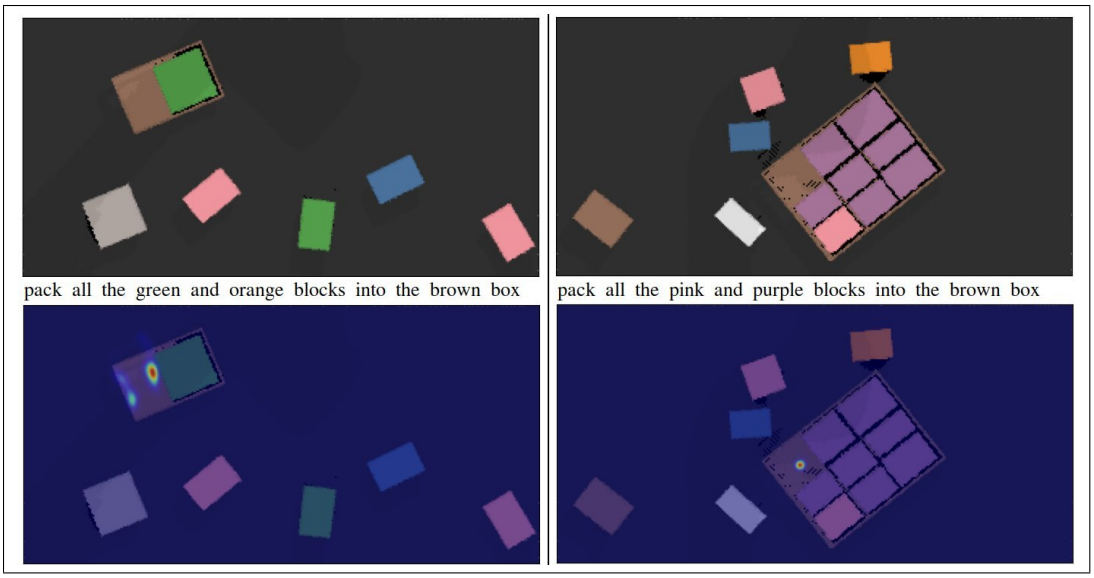
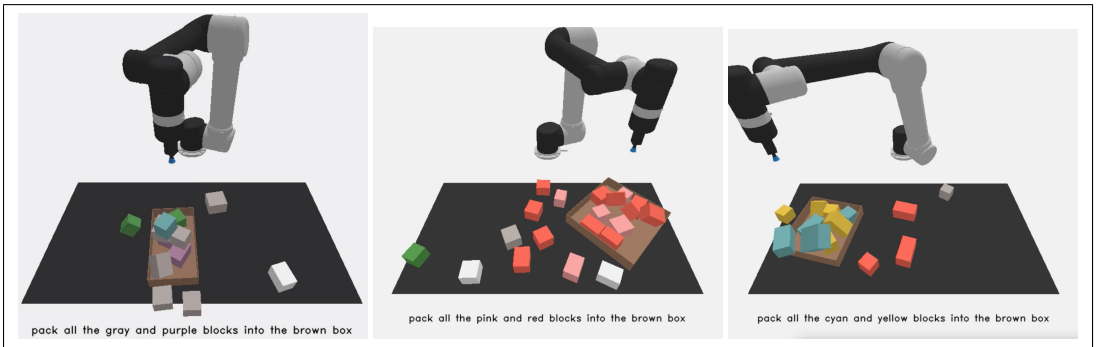


Figure 5: Example place affordances for two different commands. Top is raw image, and bottom shows overlay of affordances onto raw image.



(a) Performance at 3,000 training steps (b) Performance at 30,000 training steps (c) Performance at 30,000 training steps

Figure 6: Results of box packing tasks after full episode

7 Conclusion

We were able to create an end-to-end learning agent with non-trivial success in a comparatively short training time by reimplementing the CLIPort agent and showed that it could control the robot in a logical manner even after only 3000 training steps. Part of the reason why we were able to get reasonable results quickly can be attributed to heavy inductive biases from the transporter network, as well as the strong language and image representations from the pre-trained CLIP model. We also showed some qualitative results on the ability for the model to adapt to out-of-distribution language commands (relative to the CLIPort training distribution, not relative to the CLIP training set) due to the CLIP model. Furthermore, our result bodes well for the practical applications of VLM, especially if further training can result in the performance indicated by the original CLIPort paper.

Our Github repository can be found at `https://github.com/KevinGmelin/VLM`:

References

Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *Conference on Robot Learning*, pages 894–906. PMLR, 2022.

Andy Zeng, Pete Florence, Jonathan Tompson, Stefan Welker, Jonathan Chien, Maria Attarian, Travis Armstrong, Ivan Krasin, Dan Duong, Vikas Sindhwani, et al. Transporter networks: Rearranging the visual world for robotic manipulation. *arXiv preprint arXiv:2010.14406*, 2020.

Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pages 991–1002. PMLR, 2022.

Corey Lynch and Pierre Sermanet. Language conditioned imitation learning over unstructured data. *arXiv preprint arXiv:2005.07648*, 2020.

Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on robot learning*, pages 1113–1132. PMLR, 2020.

Lin Shao, Toki Migimatsu, Qiang Zhang, Karen Yang, and Jeannette Bohg. Concept2robot: Learning manipulation concepts from instructions and human demonstrations. *The International Journal of Robotics Research*, 40(12-14):1419–1434, 2021.

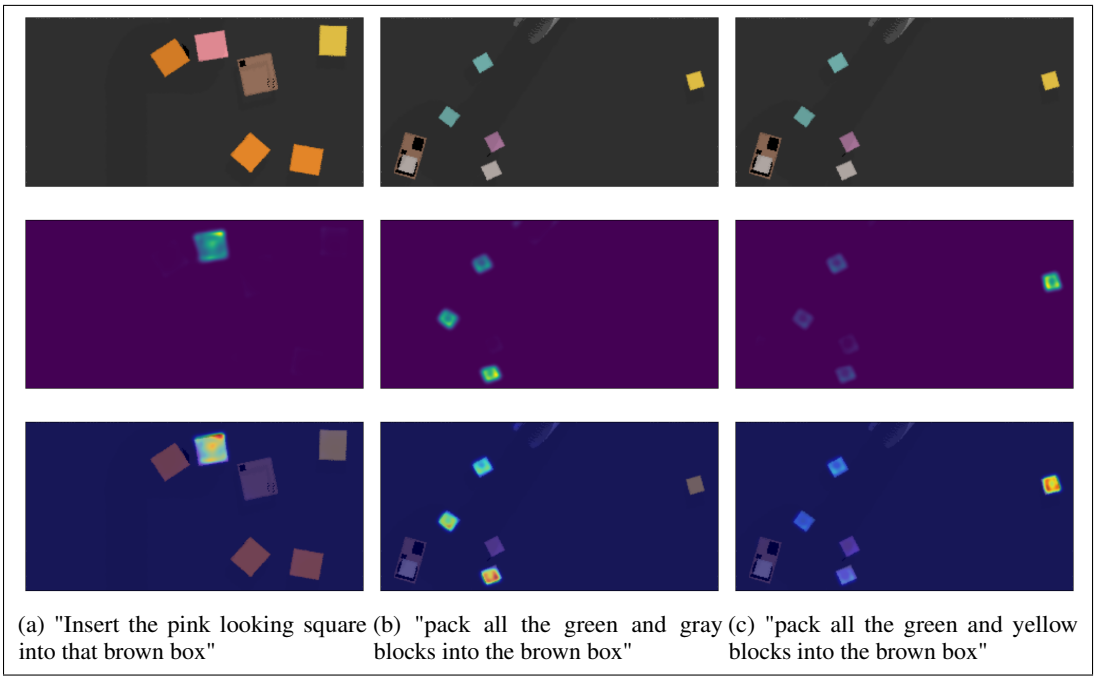
Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749, 2020.

Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 2022.

Kaizhi Zheng, Xiaotong Chen, Odest Chadwicke Jenkins, and Xin Eric Wang. Vlmbench: A compositional benchmark for vision-and-language manipulation. *arXiv preprint arXiv:2206.08522*, 2022.

Appendix

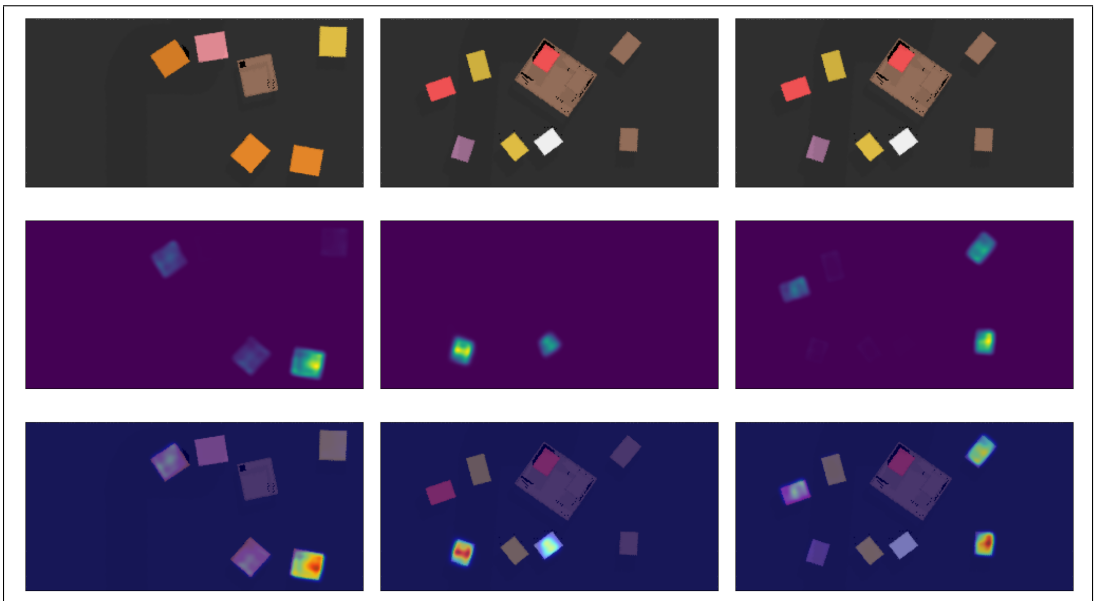


(a) "Insert the pink looking square into that brown box"

(b) "pack all the green and gray blocks into the brown box"

(c) "pack all the green and yellow blocks into the brown box"

Figure 7: Pick affordances for different scenarios and commands (1). Top: raw image, middle: affordances, bottom: overlay of affordances onto raw image

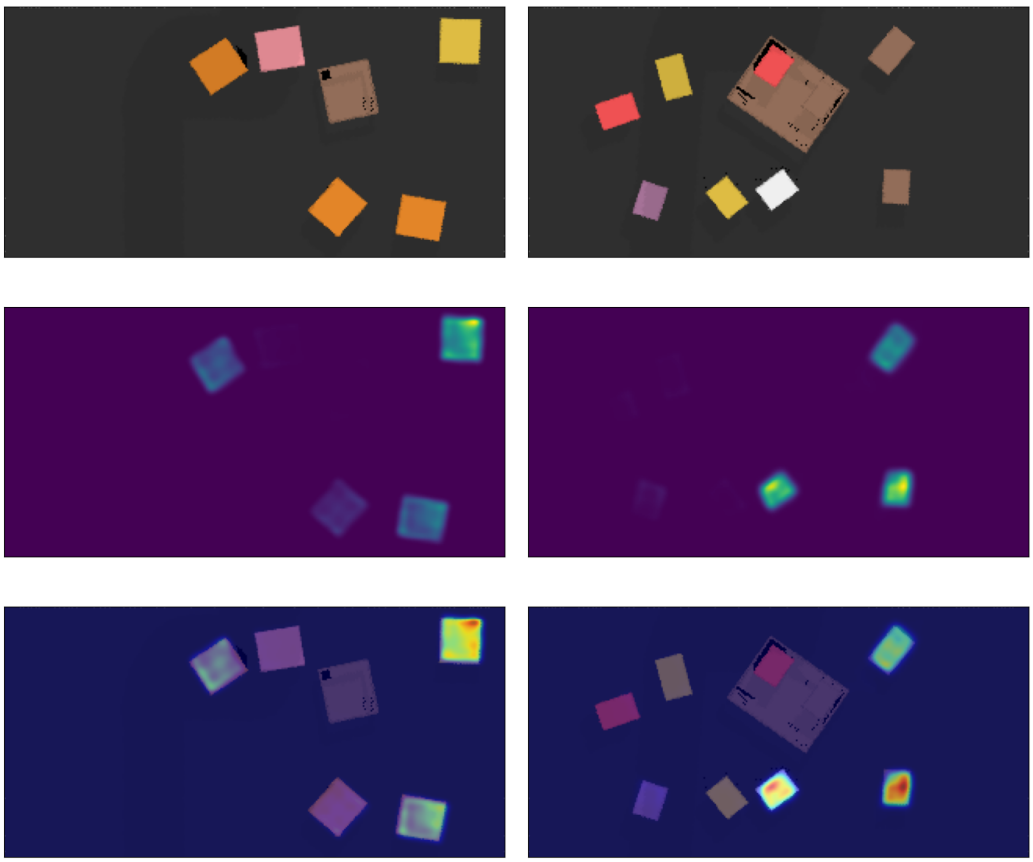


(a) "pack all the orange blocks into the brown box"

(b) "pack all the purple and white into the brown box"

(c) "pack all the red and brown blocks into the brown box"

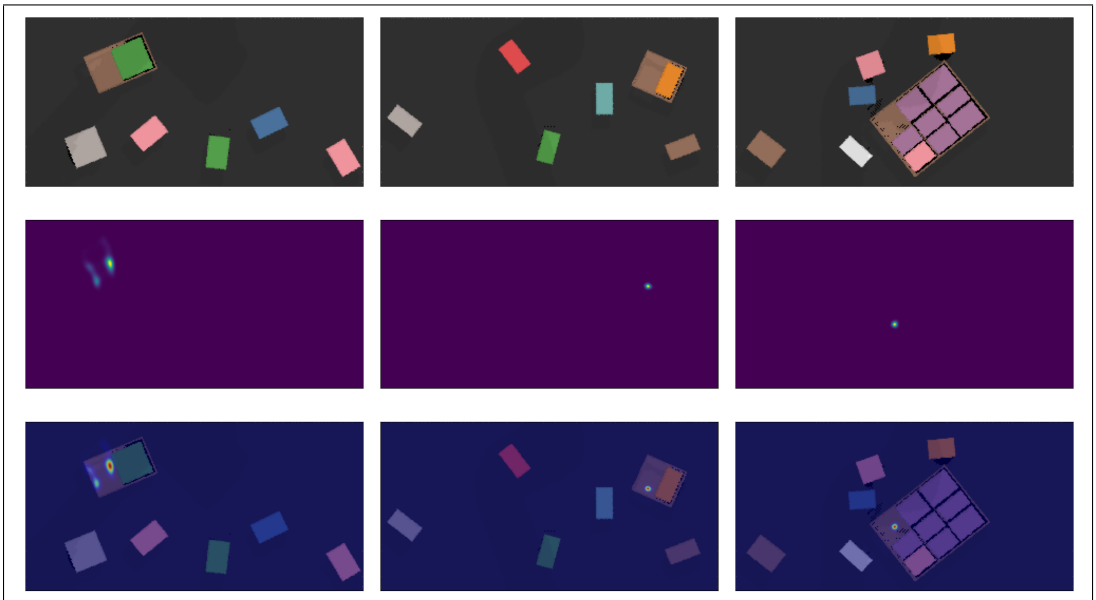
Figure 8: Pick affordances for different scenarios and commands (2). Top: raw image, middle: affordances, bottom: overlay of affordances onto raw image



(a) "pack all the yellow blocks into the brown box"

(b) "put into the box the brown and snow colored squares"

Figure 9: Pick affordances for different scenarios and commands (3). Top: raw image, middle: affordances, bottom: overlay of affordances onto raw image

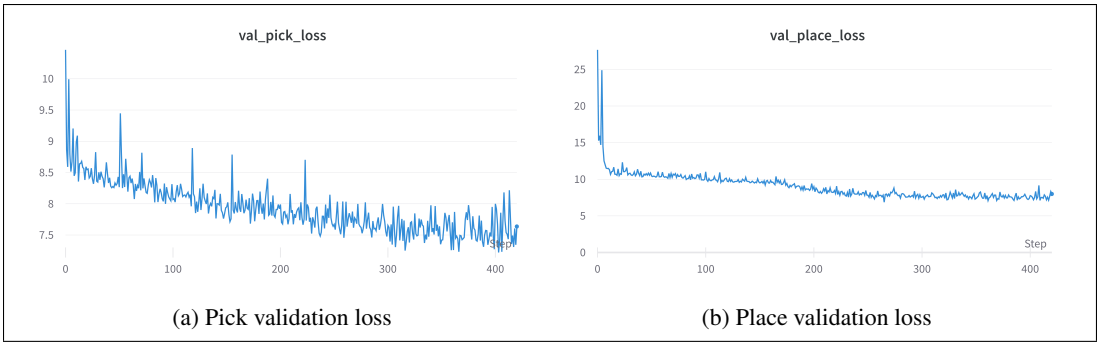


(a) "pack all the green and orange blocks into the brown box"

(b) "pack all the orange and brown blocks into the brown box"

(c) "pack all the pink and purple blocks into the brown box"

Figure 10: Place affordances for different scenarios and commands. Top: raw image, middle: affordances, bottom: overlay of affordances onto raw image



(a) Pick validation loss

(b) Place validation loss

Figure 11: Validation loss over training