7 Appendix

7.1 Finetuning CLIP on in-domain Data

In our experiments, the dot products between the embeddings of possible captions and of an RGB observation from our environment $y = \phi_I(o_t) \cdot \phi_T(l_i)$ were often uninformative: correct and wrong pairs obtained very similar scores, and varied too little in range. Our goal is to set a threshold γ to recognise correct and wrong descriptions given an image: therefore we need a larger difference in score. To tackle this, we collect a dataset of image observations with various configurations of the objects and the corresponding language descriptions using an automated annotator based on the MuJoCo state of the simulation to finetune CLIP with in-domain data. The plot on the right provides an analysis of our findings: precision and recall tend to increase logarithmically with the dataset size. The key takeaway message is that, although CLIP is trained on around 10^8 images, just 10^3 in-domain pairs are enough to improve its performance on our tasks.

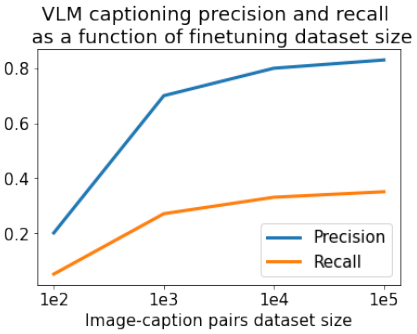

Figure 10: Captioning precision and recall of finetuned CLIP as a function of the dataset size. The logarithmic trend suggests that around 10^3 image-caption pairs unlock sufficient performance. Values obtained with $\gamma = 0.8$.

In our case, a high precision is more desirable than high recall: the former indicates that positive rewards are not noisy, while the opposite may disrupt the learning process. A lower recall indicates that the model may not be able to correctly identify all successful trajectories, but this simply translate in the need for more episodes to learn, and does not disrupt the learning process. We found a value of $\gamma = 0.8$ to be the best performing choice after finetuning.

7.2 Current Limitations and Future Work

1) In our current implementation, we use a simplified input and output space for the policies, namely the *state space* of the MDP - i.e. the positions of the objects and the end-effector as provided by the MuJoCo simulator - and a pick and place *action space*, as described in Sec. 3, where the policy can output a x, y position for the robot to either pick and place. This choice was adopted to have faster experiments iteration and therefore be able to focus our search on the main contribution of the paper: the interplay with the LLM and the VLM. Nevertheless, the recent literature has demonstrated that a wide range of robotics tasks can be executed through this action space formulation ([48]; [39]).

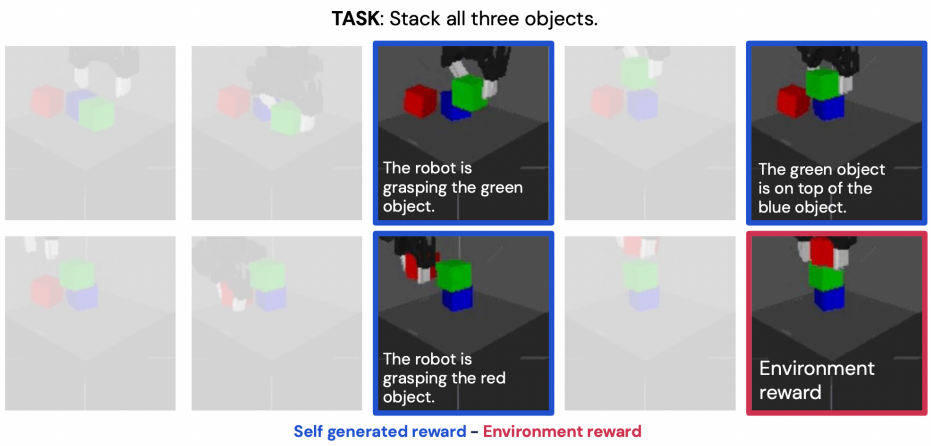

Figure 11: Autonomously identifying sub-goals and corresponding rewards becomes especially important when tasks become prohibitively sparse, like *Triple Stack*.

Imagine you are a robot arm interacting with an environment. In front of you there are a red object, a blue object, and a green object. When receiving a TASK, describe a set of SUBGOALS that would make you solve the task.

TASK: Stack the red object on top of the green object.
SUBGOALS: 1) The robot is grasping the red object.
2) The red object is on top of the green object.

TASK: Grasp the red object.
SUBGOALS: 1) The robot is grasping the red object.

TASK: Stack the blue object on top of the red object.
SUBGOALS: **1) The robot is grasping the blue object.**
2) The blue object is on top of the red object.

TASK: Stack all three objects.
SUBGOALS: **1) The robot is grasping the red object.**
2) The red object is on top of the green object.
3) The robot is grasping the blue object.
4) The blue object is on top of the red object.

Figure 12: An example of the prompt we used to condition the LLM, and its outputs. Normal text: user inserted text, **bold text:** LLM outputs.

Many works from the current literature ([25; 40; 5; 14]) demonstrate that, in order for the policy to scale to *image observations* as input and *end-effector velocities* as output, the model only needs more data, and therefore interaction time. As our goal was demonstrating the *relative performance improvements* brought by our method, our choice of MDP design does not reduce the generality of our findings. Our results will most likely translate also to models that use images as inputs, albeit with the need for more data.

2) We finetune CLIP on in-domain data, using the same objects we then use for the tasks. In future work, we plan to perform a larger scale finetuning of CLIP on more objects, possibly leaving out the object we actually use for the tasks, therefore also investigating the VLM capabilities to generalise to inter-class objects. At the moment, this was out of the scope of this work, as it would have required a considerable additional amount of computation and time.

3) We train and test our environment only in simulation: we plan to test the framework also on real-world environments, as our results suggest that 1) we can finetune CLIP with data from simulation and it generalises to real images (Sec. 5.4), therefore we can avoid expensive human annotations 2) the framework allows for efficient learning of even sparse tasks *from scratch* (Sec. 5.1), suggesting the applicability of our method to the real-world, where collecting robot experience is substantially more time expensive.

7.3 Prompts and outputs of the LLM

In Fig. 12 we show the prompt we used to allow in-context learning of the behaviour we expect from the LLM ([33]). With just two examples and a general description of the setting and its task, the LLM can generalise to novel combinations of objects and even novel, less well-defined tasks, like "*Stack all three objects*", outputting coherent sub-goals.