A Appendix

A.1 Predicting Progress

We measure the performance of the learnt representations to encode progress by running a KNN classification. First, we collect 500 demonstrations on the full insertion task using manually defined way points and a PD controller. We vary the starting configuration of each robot arm by 1° for all its 7 joints. Then, we perform a CV split over the collected trajectories and obtain training and validation sets. We train both encoders using the training data. Once training is complete, we process all training trajectories with each encoder which results in two separate encoded data sets. We use those to train two separate 10-fold KNN classifiers - one for each type of encoding. Then, we process the never seen before validation set using each of the encoders and evaluate the accuracy of predicting the episode progress with the KNN classifiers. Table 9 shows the results. It can be seen that the VAE achieved much higher accuracy when evaluated on the training data as opposed to test, indicating it has overfitted to it. In contrast, the TCC was much better at predicting the episode progress.

Model	Train	Test
VAE	**88.6%**	63.6%
TCC	86.2%	**79.2%**

Figure 9: Predicting episode progress with 10-fold KNN.

A.2 Ablation of the online goal selection stage

Goal-conditioned (gc) policy learning takes in as input a desired goal state to solve for. However, in cases where the goal is as abstract as 'solve the task' there may be a number of different goals describing the same problem. Therefore, restricting the gc policy to a single target goal state may negatively impact the learning process. Additionally, using a small subset of goals, e.g. goals corresponding to the final states of all demonstrations as done in [31], may be insufficient too. In contrast, conditioning on a wider range of goal states that solve the same task can better capture the goal distribution describing the task at hand. As a result, we randomly sample a plausible target goal state for each roll-out during training (see Figure 2). We propose to continuously grow a target goal distribution as training evolves and the agent starts solving the training task. In this section, we compare the performance of our agent when using a single target goal to represent solving the task, using as target goals only the goal states from the provided near-optimal demonstrations, and continuously growing the goal database as learning progresses. Our findings reported in Figure 10 show that continuously growing the goal database allows for better and faster learning in the context of vaguely formulated goals such as the 3.5mm jack insertion considered in this work.

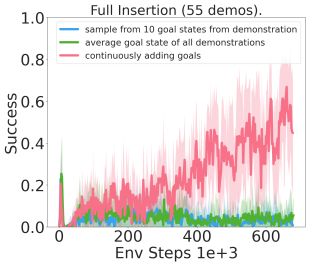
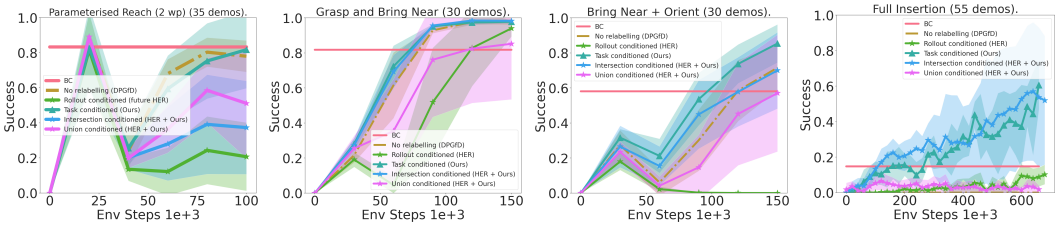


Figure 10: Different types of online goal selections.

A.3 Mixing the goal distribution support

Mixing goal candidates taken from the agent's rollout and the provided demonstrations can help for retroactive goal selection. A potential scenario is the one discussed in Appendix A.4. Having noisy, sub-optimal representations that do not preserve the notion of progress can be problematic. An



(a) (b) (c) (d)

Figure 11: Measuring per-step reward using the smallest number of demonstrations that resulted in learning.

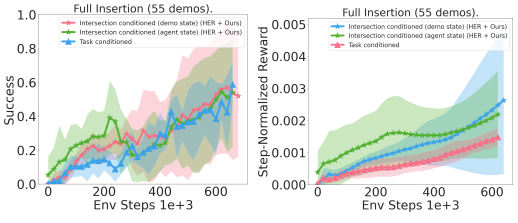
alternative scenario could involve a relatively narrow set of demonstrations. This is particularly evident in complex task settings where having a set of successful demonstrations can still be insufficient to solve the task as there are a vast number of failure modes, like in the full insertion task, for example. In this subsection, we focus on studying different candidate heuristics that can help relax these constraints. We build upon the formulation for retroactive goal selection introduced in Section 4.2 which allows us to fuse together the HER-style relabelling strategies and demo-driven ones too.

Using demonstration states as candidate goals: The simplest version of this is a union over both sets where the agent gets to sample a goal that comes from the rollout or the demonstration data. This can be expressed as $R_G = \{z_g \in \zeta \bigcup \mathscr{D} : p(z_g) > 0\}$. Such formulation can be useful, for example when the provided demonstrations are only partially useful to solving the task. Implicitly letting the agent to sample goals that are part of its own rollout can help model useful behaviours that over time could help get us closer to the provided demonstrations.

An alternative version is when the goal distribution $p(z_g)$ is composed of the intersection over the two types of goal distributions discussed in Section 4.2. In this case, the support of the goal distribution becomes $R_G = \{z_g \in \zeta \bigcap \mathscr{D} : p(z_g) > 0\}$. We find this intersection to be useful in cases where the adopted representation does not have an encoded notion of progress. Therefore, choosing goals that are $\varepsilon-$close to states produced by the agent's dynamics can hypothetically act as regularisation over the choice of goals we use but still ensure staying close to the target task. We can collect all qualifying goals for a time step t by iterating over the data set of successful trajectories obtained from demonstration and compare to z_t. Since z_t and all z_g are temporally consistent, we can use Eq. 3 to prune the data set and pick the closest z_g for each z_t. We used a task-conditioned sampler where we sample directly from a distribution implied from demonstrations. However, we can also compose distributions comprised of mixing goals from the agent's rollout and the demonstrations. Here we consider two different versions of this.

We report our overall results in Figure 11. Our results indicate that using the intersection over the rollout trajectory and the demonstrated goal results in broadly similar resutls as the task conditioned approach. However, the intersection based solution had much higher variance indicating that the agent is is less stable where some seeds achieved near perfect performance and others were closer to failure. We noticed that this type of goal conditioning can be useful when training with a VAE. That is, this sampling strategy can be useful in cases where the quality of the representations and also of the implied task distribution is poor, e.g. when they do not contain notion of progress or in low data regimes. We report these details in Appendix A.4.

Using relevant agent states as candidate goals: Utilising the demonstration states to collect candidate goal distributions can be a powerful tool as we demonstrate in this work. However, in complex tasks, such as the full insertion task (Figure 1), relabeling the goals with just demonstration states can sometimes fail to make the sparse-reward problem easier (as intended by HER) since the resulting goal distributions are still relatively narrow. This is particularly true in the beginning of the training process when there is a relatively large mismatch between the agent's Q function and the true underlying dynamics the provided demonstrations follow. Therefore, we consider an alternative method that can help speed up the training process. To this end, we can form a collection of candidate goals that is jointly conditioned on both the agent and the demonstrator's behaviours but is comprised of all z_t that fall under the ε threshold defined in Appendix A.8. In this setting, we focus on modelling the agent's behaviour directly by focusing only on relevant to the task states as opposed to strictly targeting actual demonstration states. Figure 12 summarises our findings. While using this type of joint conditioning can speed up training (plot on the right), it does not necessarily result in improved performance (plot on the left). We suspect that mixing the goal distribution support can be potentially very useful to using less demonstrations or partially useful demonstrations. We leave this study for future work.



(a) (b)

Figure 12: Comparing different demo-driven support distributions.

A.4 Noise sensitivity of the encoder: challenging the notion of progress

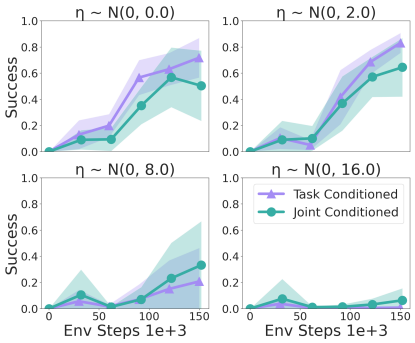


Figure 13: Injecting noise to the engineered encoder. Reach, Grasp, Lift, Orient Task.

We compare the performance of both the joint- and task-conditioned samplers on the Bring Near + Orient task and report the overall accuracy. We trained for a total of 150K environmental steps and report our findings in Figure 13. Relying on task-conditioned samples results in higher accuracy for the less noisy observations. This indicates that having a stronger representation is directly related to the agent's confidence in 'understanding' the actual task it has to solve. This study further confirms our conjecture that notion of progress is paramount to solving complex sequential tasks. Note that the relationship between the level of noise and performance depicted in Figure 13 does not affect the jointly conditioned relabelling strategies as much as it does for task-conditioned relabelling. In fact, a little bit of noise leads to improved performance for the former while it slows down training for the latter. This indicates that relying on alternative mechanisms for indicating progress, such as conditioning on the agent's own trajectory can be useful when progress is not successfully encoded in the representations used. This aligns with our motivation from Section 4 that task-conditioning works when we are confident in the quality of the task distribution. However, the proposed ablation in this section points towards an alternative mode that can potentially compensate for this. Namely, implicitly informing the agent for the notion of progress e.g. through building heuristics for hindsight goal selection that utilise both demonstrations and agent motion can be useful. We hypothesise that retroactive relabelling using only goals that are both similar to the demonstrations and aligned with the current agent's trajectory can be useful with respect to the agent's current understanding of the dynamics, represented through its current Q function. Next, we consider three different strategies for extracting candidate goals and discuss some of their benefits and limitations.

A.5 Quality of encoder: extended study

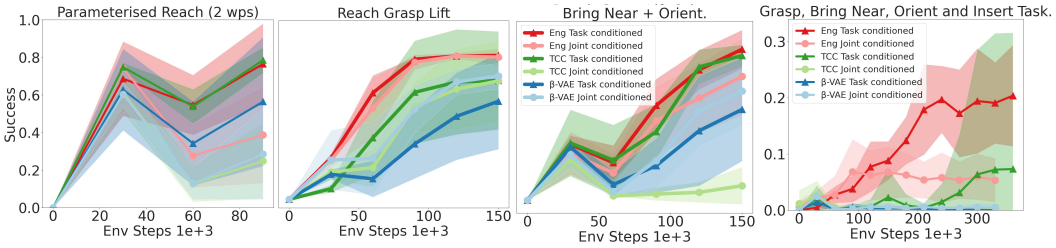


Figure 14: Learnt representations.

The previous section suggests that a joint-conditioned sampler can be more useful when the used representations do not encode notion of progress. In this section we compare using task-conditioned and joint-conditioned encoders using learnt representations instead. We compare using TCC and VAE and benchmark the results against a hand-engineered encoder. Even though TCC with a task-conditioned sampler achieved the closest results to the best hand-engineered solution across all tasks, we can see that a joint-conditioned encoder can work better for states that do not encode notion of progress.

Figure 14 illustrates the achieved results. We can see that using a β-VAE encoder worked best with goal sampling from a joint-conditioned goal distribution, $R_G = \{z_g \in \zeta \bigcap \mathscr{D} : p(z_g) > 0\}$. Note that the intersection between both distributions still results in a data set comprised of goal candidates that still belong to the implied from demonstrations task distribution. However, we only used the goals that were similar to the agent rollout. This result is connected to our observations from Appendix A.4 that a joint-conditioned sampler implicitly introduces a notion of progress via utilising the agent's own motion at the retroactive goal selection stage.

Another interesting observation is the final full insertion's task performance. Although TCC-based representation came closest to the engineered representation, it was still much lower. The full insertion task relies the most on the contact-rich manipulation to be completed when compared to the rest. The engineered representation contains information relevant to the manipulation which is why we suspect the gap between both learnt and engineered representation is much larger than the rest of the tasks. A potentially exciting future direction is attempting to extract representations that preserve the notion of contact as well as progress.

A.6 Per-step reward

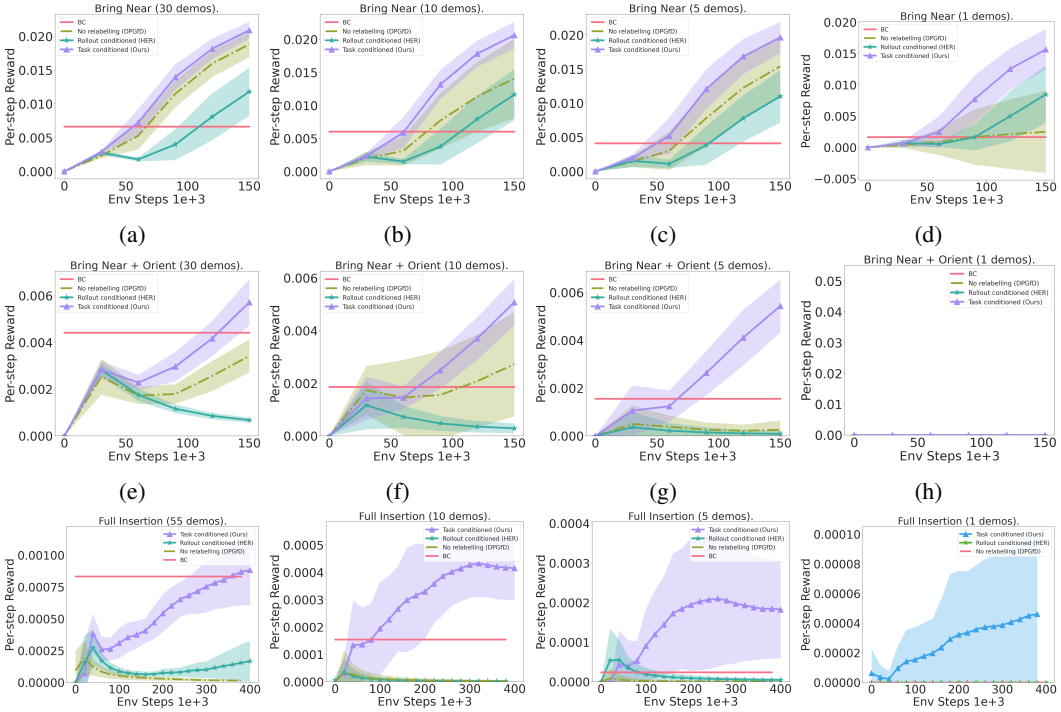


Figure 15: Measuring per-step reward.

A.7 Progress-based weighting

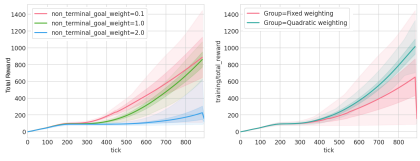


Figure 16: Measuring per-step reward using the smallest number of demonstrations that resulted in learning.

Weighting down the BC and actor-critic losses associated with intermediate states can have slight benefits to improving the speed of learning of goal-conditioned DPGfD. Although in practice there could be many different ways of reweighing loss values, we found two particular ones useful in our setting. One way of scaling such losses is by choosing a fixed weight ω that scales down all non-terminal states' losses during training by the same fixed weight. An alternative weighting can be defined by using a quadratically scaled weight using the episode progress. That is, for batch b, we get $\mathscr{L}_{BC}(b) = \lambda^p * \mathscr{L}_{BC}(b)$ and $\mathscr{L}_{TD} = \lambda^p * \mathscr{L}_{TD}$, where $*$ indicates element-wise multiplication and

$$\lambda^p = \begin{cases} 1.0, & \text{if } z_t = T \\ \omega, & \text{otherwise} \end{cases}, \; or \; \lambda^p = i^2, \; for \; i \in \{\frac{1}{T}, \ldots, \frac{T}{T}\}. \tag{4}$$

We used $\omega = 0.1$ in our experiments. Figure 16 illustrates an example of re-weighting on the Bring Near + Orient task. Down-weighting intermediate states can lead to slightly faster learning and a

higher variance performance. There is a difference between weighting states using a fixed value and assigning quadratic weighting proportional to the episode progress.

A.8 Computing the threshold

There are multiple ways to obtain an ε threshold for our goal conditioned reward. We used the provided demonstrations to compute the average distance $\varepsilon = \mu + k\sigma$, where μ and σ were extracted using a rolling distance between consecutive states from the encoded demonstrations, e.g. $||z_t^d - z_{t+m}^d||$, for a demonstration d with an m step gap in between the two states and k standard deviations. In our tasks, $m = 10$ for all tasks but the bring near and orient and the full insertion where we used $m = 5$. We use a rolling distance over a window of time steps because we did not want to let time step clusters often situated around the different "narrow phases" of a trajectory influence the average threshold. Broadly, we notice a relationship between the size of the rolling window and the precision and recall of the obtained goal-conditioned sparse reward function. We ablate the importance of a rolling window size on the bring near and orient task in Figure 17. There, it can be seen that too small of a rolling window size (such as 1) might have a noticeable negative impact on learning due to the clusters situated around the different phases. Effectively, too small of a window can affect the recall of our obtained reward which can be detrimental to learning. In contrast, too large of a rolling window can affect the speed of learning due to allowing for a more flexible threshold function. Using too large of a rolling window can reduce the precision of the obtained threshold by rewarding too many false positive states. In terms of learning, this can be detrimental to the speed of learning a successful policy and might result in converging to poorer performance too.

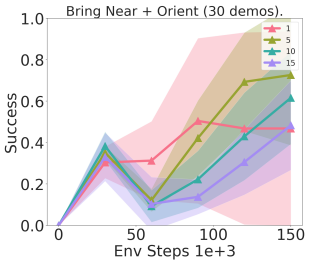


Figure 17: Ablating the rolling window.

A.9 Computing the engineered encoders

The engineered goal encoders vary between tasks, but in all cases the encoding captures some notion of progress of the agent through the set task.

Parameterized Reach: The engineered encoder for this task concatenates the robot arm pose and a mask of which waypoints have been visited so far.

Bring Near: The state encoder is the concatenation of the distance of the left and right grippers from their corresponding cables, the distance between the cable tips, and of whether the grippers have grasped their respective cables.

Bring Near and Orient: The encoder for this task is similar to Bring Near, but also adds the dot product between the z-axes of the left and right cable tips.

Bimanual Insertion: The encoder for this task is similar to Bring Near and Orient, but adds the distance of the cable tip from the socket bottom, along the z-axis of the socket.

A.10 Choosing number of hindsight goal samples

We followed the intuition that the complexity of the task guides the number of samples required to capture the overall task distribution. That is, we used 2 samples for the simplest task of Parameterised Reach, 4 for the Bring Near, 6 for Bring Near and Orient and 12 for the Bi-manual Insertion.