Supplementary Material

A Count Models

Our approach makes use of Locality Sensitive Hashing (LSH) to map high-dimensional, continuous inputs to discrete hash codes. The idea behind LHS is to hash states according to a certain similarity metrics. Our LHS algorithm of choice is a variation of SimHash Charikar (2002), which uses angular distance between states as a similarity metrics. The mapping function first projects the input (dim: n) to a lower dimensional space (dim: k):

$$\phi(s) = \left\lfloor \frac{Mg(s) + v}{\sigma} \right\rceil$$

where $g(s)$ is an optional preprocessing function, which in our case is equal to the identity mapping, $M \in \mathbb{R}^{k,n}$ is a matrix with entries drawn from an i.i.d. standard Gaussian $\mathcal{N}(0,1)$ and $v \in \mathbb{R}^k$ is a random vector with uniform entries in the interval $[0, \sigma)$, where σ is a task-dependent constant. SimHash takes the sign of the random projection, while we round it up to the next integer as in (Datar et al., 2004). This better reflects similarity under the Euclidean distance rather than angular distance and the probability of a collision (two inputs having the same hash) depends on their Euclidean distance.

Once we have computed the mapping $\phi(\cdot)$, we use it inside the Count-Min Sketch algorithm. Count-Min Sketch is designed to support memory-efficient counting without introducing too many over-counts. It maintains a separate count n_j for each hash function ϕ_j defined as $\phi_j(s) = \phi(s) \,\%\, p_j$, where p_j is a large prime number. Our implementation follows the one of Tang et al. (2017) in the static hashing variant[1]. As in their paper, we consider $j = 6$ 'buckets' equal to $[999931, 999953, 999959, 999961, 999979, 999983]$, which we keep fixed for all the experiments. The number of counts is then $\min_{j=[1,6]} n_j(\phi_j(s))$.

Algorithm 2: Count models update

Input : $\mathcal{D}_{\text{last}}$: replay buffer of most recently collected rollouts; n: input dimension, k: hashing dimension; σ: random vector interval range; I: training iterations
Output : single-state count model $N(s,a)$, state-pair count model $N(s,a,s')$

```
1  Initialize random matrix $M \in \mathbb{R}^{n,k}$ with entries from the standard Gaussian distribution $\mathcal{N}(0,1)$
2  Initialize random vector $v \in \mathbb{R}^k$ with entries from uniform distribution on the interval $[0,\sigma)$
3  for $i$ = 0 to $I-1$                                   // loop over training iterations
4  do
5      for (s,a) in $\mathcal{D}_{\text{last}}$                        // update $N(s,a)$
6      do
7          increase counts for $(s,a)$
8      for $\tau$ in $\mathcal{D}_{\text{last}}$                          // update $N(s,a,s')$
9      do
10         increase counts for all possible triples $(s_i, a_i, s_j) \in \tau$, with $j > i$   // $O(\text{len}(\tau)^2)$ triples
```

A.1 Granularity

In fig. S7 we show the effect of the dimension k on the granularity of the state space counting. The count model $N(s,a)$ was previously updated with the (s,a) transitions from a set of 2000 trajectories collected with *Ours+HER* policy, which amounts to 200k transitions.

[1] https://github.com/openai/EPG/blob/master/epg/exploration.py

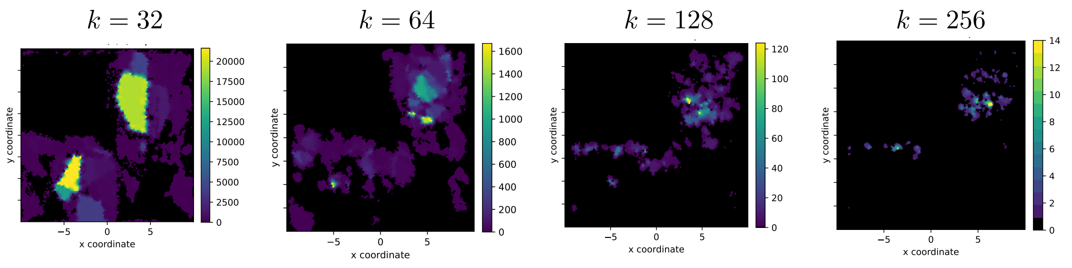
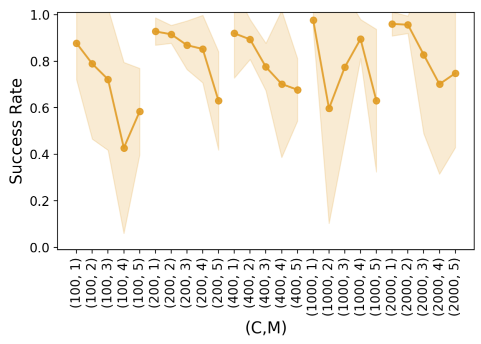


Figure S7: Granularity of the single-state count model $N(s, a)$ for the POINT MAZE environment depending on the dimension of the hash key k. Every plot represents the number of estimated counts of any input (s, a) where $s = (x, y, v_x = 0, v_y = 0)$ and $a = (0, 0)$, given that the count model previously visited a dataset of 200k points (trajectories obtained with *Ours + HER*).

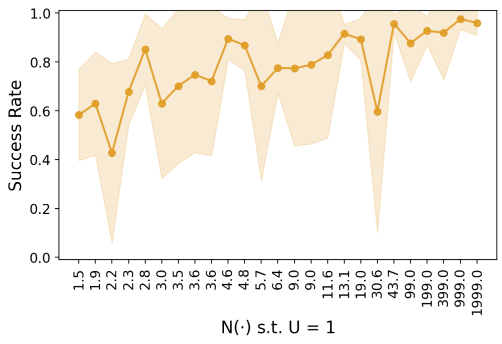
A.2 SENSITIVITY ANALYSIS

As we can see from Figure S8 (a), the performance on the Point Maze drops as we increase the scale at which the uncertainty decays (exponent M). At iteration 0, when no point has been visited, the count term is zero and the corresponding uncertainty is equal to its maximum value, which is the value of C. Then, for a fixed C, the higher the exponent M is, the faster the uncertainty decays with the counts and, consequently, the performance. This is because in the Point Maze environment we observe the "wormhole" phenomenon, also documented in (Eysenbach et al., 2019), where the distance estimates are overly optimistic and do not take into account the presence of obstacles. As a result the agent thinks that it can go through them in order to reach a goal on the other side.

In this case, having a stronger uncertainty for the distance estimate is better, as confirmed in plot S9 (b), where we plot the success rate vs. the amount of counts necessary to have 1 unit of uncertainty. The plot shows how the performance increases with the number of counts, suggesting that a very pessimistic count model is a better choice for the Point Maze. The values we used for all of our experiments are C = 400 and M = 2.



(a) Grouped by the numerator C.



(b) Ordered by the count needed to have uncertainty = 1 timestep

Figure S8: Sensitivity analysis of the parameters regulating the decay of the count-based uncertainty $U[\cdot] = \frac{C}{(1+N(\cdot))^M}$.

B FETCH PICK AND PLACE

In fig. S9 (b) we report the FETCH PICK AND PLACE performance without the additional goal for the end effector which we used in the final experiments (fig. 3). The reported value is in line with the results presented in the original HER paper (Andrychowicz et al., 2017). Augmenting the goal space with an extra goal position for the end effector, equal to the one for the box, increases the sample efficiency of a $\sim 10x$ factor without introducing any explicit reward shaping.

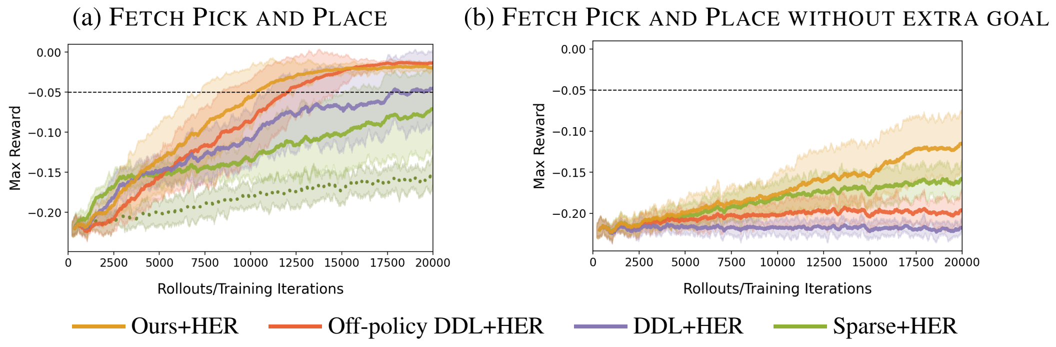


Figure S9: Performance comparison on the FETCH PICK AND PLACE task with and without additional goal for the end effector. The dark green dotted line of the left plot is the same as the solid green line of the right plot.

B.1 ADDITIONAL LOCAL MINIMUM: FETCH PICK AND PLACE WITH WALL

In this section we present the rates at which the hypotenuse gets penalized (merged into the catheti sum) or regularized (decreased to a lower catheti sum), for the Fetch Pick and Place with Wall (fig. S11). We also show two illustrative frames from Sparse+HER and Ours+HER, that show the local minima where the baselines tend to get stuck (fig. S10).

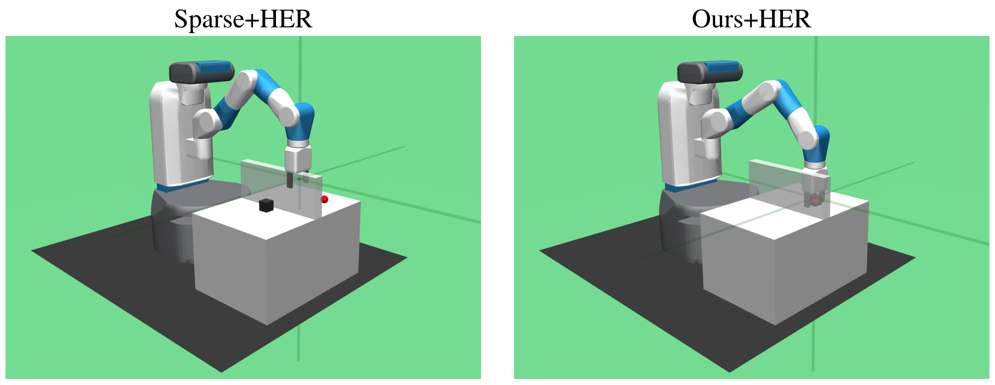


Figure S10: Frames of trajectories produced by a goal-conditioned policy learned from Sparse+HER vs Ours+HER. The additional wall introduces further exploration difficulties in the FETCH PICK AND PLACE task.

C IMPLEMENTATION DETAILS

C.1 HYPERPARAMETERS

All the networks (Q-functions, policies, distances) use the Adam optimizer. All the task horizons are equal to 50 time steps, apart from the 100 time steps used for the POINT MAZE task. Table S1 contains the parameters used for DDPG and SAC, together with the best k value for HER relabeling. Differently from the other methods, the best k for Ours+HER is lower; in fact, only 20% of the data ($k = 0.25$) gets relabeled with the achieved goal. We believe that this is due to the effect of the triangular loss: the distance to the desired goal better reflects the true shortest path and it is more informative than the distance to the achieved goal. In table S2 we report the architecture and training parameters for the distance network, and the chosen hashing key k for every environment.

Parameter	**Value**
Episode Length	50
Batch Size	256
Updates per Episode	100
Replay Buffer Size	$5e10^5$
Learning Rate	0.001
Discount Factor γ	0.98
Polyak Averaging	0.95
Action Noise (DDPG)	0.2
Action L_2 penalty (DDPG)	1
Random ϵ-Exploration	0.3
Q-Target Clipping (Sparse+HER)	$[-50, 0]$
Q-Target Clipping (others)	$[-1275, 0]$
Policy Network	3×256
Q-Function Network	3×256
Activation Function	ReLU
Weight Initialization	Xavier Uniform
Normalize Input	Yes
HER Replay Strategy	Future
HER Replay-k	see right table

Environment	**Method**	**HER-k**
POINT MAZE	Ours+HER	0.25
	DDL+HER	4
	Off-Policy DDL+HER	4
	Sparse+HER	None
FETCH REACH	Ours+HER	4
	DDL+HER	4
	Off-Policy DDL+HER	4
	Sparse+HER	4
FETCH REACH WITH WALL	Ours+HER	4
	DDL+HER	4
	Off-Policy DDL+HER	4
	Sparse+HER	4
FETCH PICK AND PLACE	Ours+HER	4
	DDL+HER	4
	Off-Policy DDL+HER	4
	Sparse+HER	4
FETCH PICK AND PLACE WITH WALL	Ours+HER	4
	DDL+HER	8
	Off-Policy DDL+HER	8
	Sparse+HER	4
CLAW	Ours+HER	0
	DDL+HER	4
	Off-Policy DDL+HER	4
	Sparse+HER	4

Table S1: SAC & DDPG hyperparameters and best-k (HER) for each environment (grid search over $\{0, 0.25, 1, 4, 8\}$).

Parameter	**Value**
Distance Network	3×256
Distance Network (DDL)	2×256
Distance Network (DDL)	3×256 (Fetch Envs)
Batch Size	256
Updates per Episode	100
Replay Buffer Size	$1e10^6$
Replay Buffer Size (DDL)	$1e10^5$
Learning Rate	0.0003

Environment	k for for $N(s,a)$	k for for $N(s,a,s')$	σ
POINT MAZE	64	32	20
FETCH REACH (WITH WALL)	128	128	0.3
FETCH PICK AND PLACE	32	64	0.3
CLAW	32	64	0.3

Table S2: (left) Parameters for the distance network used in Ours+Her, DDL+HER, Off-Policy DDL + HER. (right) Chosen dimension of the hash state key for the presented environments. Left column is for the single state count model used in the temporal loss, while the right column is for the double state model used in the triangular loss.

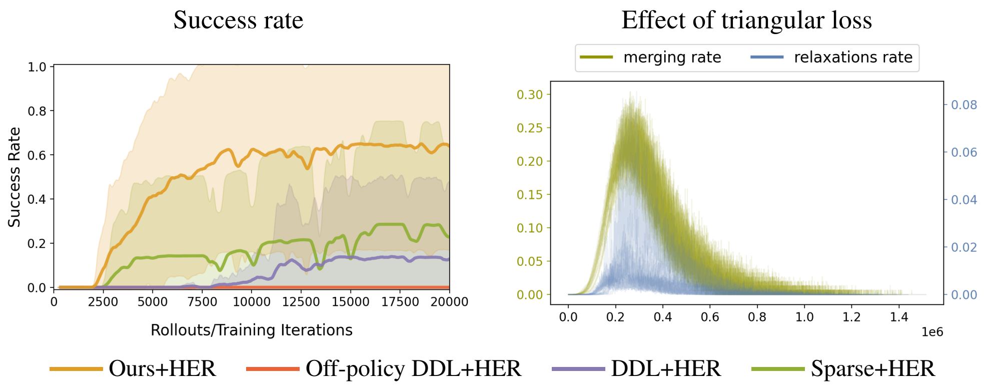


Figure S11: Fetch Pick and Place With Wall task. Our learned distance helps the optimizer to avoid the local minima created by the wall. On the right, we can see how the *merging rate* of the hypotenuse into the catheti sum is the same as in the case without wall (fig. 5 (b)), namely, at most 30% of the training batch gets penalized. However, the rate at which the hypotenuse gets corrected - the *relaxations rate* - is one order of magnitude higher.

C.2 Distance Learning

The temporal loss in eq. 4a is trained on states belonging to the same trajectory. The rollouts are chosen randomly from the buffer, then we use a very basic procedure to sample the tuples (s_i, a_i, s_j). The index i is sampled uniformly from 0 to $T-1$, while the index j is sampled uniformly from $i+1$ to $T-1$.