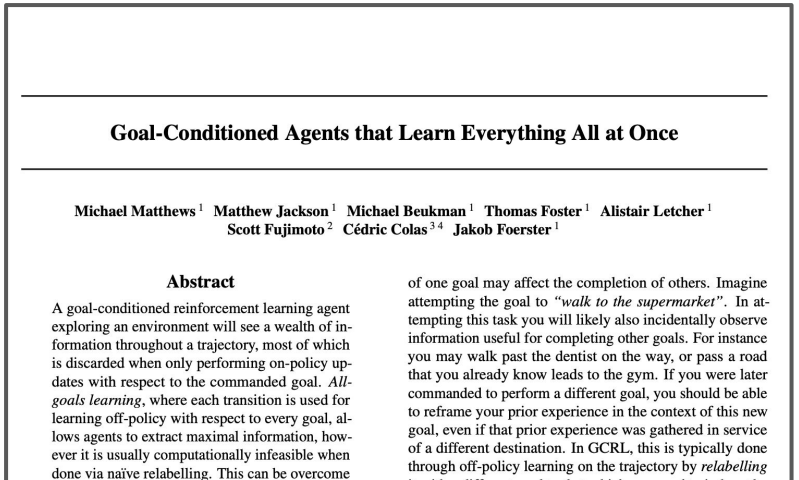
Calibration Slide

Goal-Conditioned Agents that Learn Everything All at Once

A2I / 16 July 2026



Goal-Conditioned Agents that Learn Everything All at Once

Michael Matthews [1] **Matthew Jackson** [1] **Michael Beukman** [1] **Thomas Foster** [1] **Alistair Letcher** [1]
Scott Fujimoto [2] **Cédric Colas** [3 4] **Jakob Foerster** [1]

Abstract

A goal-conditioned reinforcement learning agent exploring an environment will see a wealth of information throughout a trajectory, most of which is discarded when only performing on-policy updates with respect to the commanded goal. *All-goals learning*, where each transition is used for learning off-policy with respect to every goal, allows agents to extract maximal information, however it is usually computationally infeasible when done via naïve relabelling. This can be overcome

of one goal may affect the completion of others. Imagine attempting the goal to *"walk to the supermarket"*. In attempting this task you will likely also incidentally observe information useful for completing other goals. For instance you may walk past the dentist on the way, or pass a road that you already know leads to the gym. If you were later commanded to perform a different goal, you should be able to reframe your prior experience in the context of this new goal, even if that prior experience was gathered in service of a different destination. In GCRL, this is typically done through off-policy learning on the trajectory by *relabelling*

Short Summary

To improve learning in goal-conditioned RL, we often use Hindsight Experience Replay.

Can we sidestep that by learning with respect to all goals at once?

“The notion of goal conditioning lends itself well to efficient data reuse, by considering *how the* ***completion of one goal may affect the completion of others***.”

TESCO Express

“If you were later commanded to perform a different goal, you should be able to reframe your prior experience in the context of this new goal, *even if that prior experience was gathered in service of a different destination*”

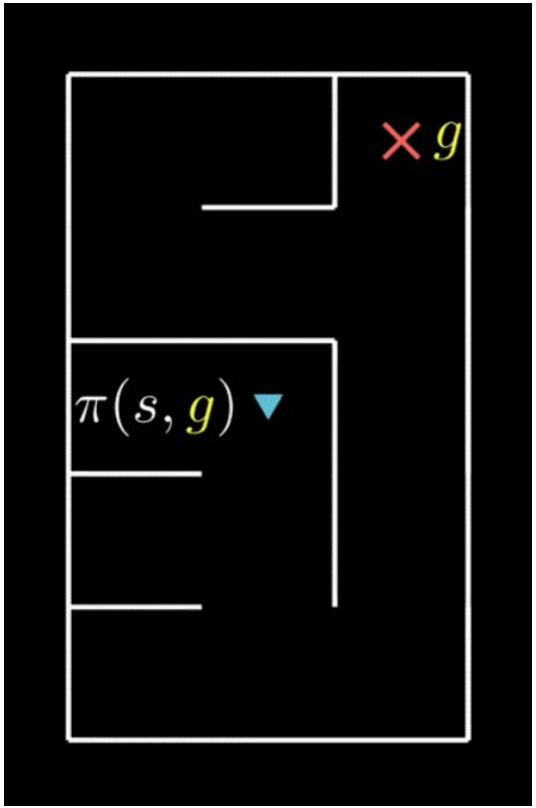
ORI
OXFORD ROBOTICS INSTITUTE

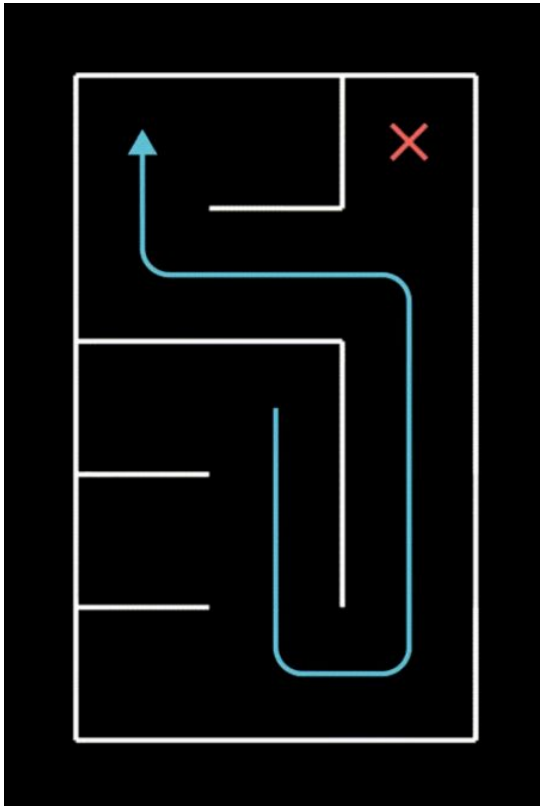
Intro to Goal-Conditioned Reinforcement Learning

We define a Goal-Conditioned Markov Decision Process as $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \mathcal{G}, \mathcal{Z}, p_0 \rangle$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the stochastic transition function and $p_0 : \Delta(\mathcal{S})$ is the initial state distribution. $\mathcal{G}$ represents the set of goals, $\mathcal{R} : \mathcal{S} \times \mathcal{G} \to \mathbb{R}$ is the goal-conditioned reward function and $\mathcal{Z} : \mathcal{S} \times \mathcal{G} \to \mathbb{B}$ is the goal-conditioned pseudo-termination function (Sutton et al., 2011).

A goal-conditioned policy $\pi : \mathcal{S} \times \mathcal{G} \to \Delta(\mathcal{A})$ is trained to maximise its expected discounted return under the uniform goal distribution $\mathbb{E}_{g \sim \mathbb{U}[\mathcal{G}]}[\sum_t \gamma^t \mathcal{R}(s_t, g)]$ where γ is the discount factor (Schaul et al., 2015). Trajectories are gathered by sampling an initial state $s_0 \sim p_0$ and goal $g \sim \mathbb{U}[\mathcal{G}]$ and then sequentially sampling actions $a \sim \pi(s_t, g)$ and states $s_{t+1} \sim \mathcal{T}(s_t, a_t)$. We refer to the notion of acting towards a specific goal g as *commanding* that goal.

How do we deal with suboptimal trajectories?

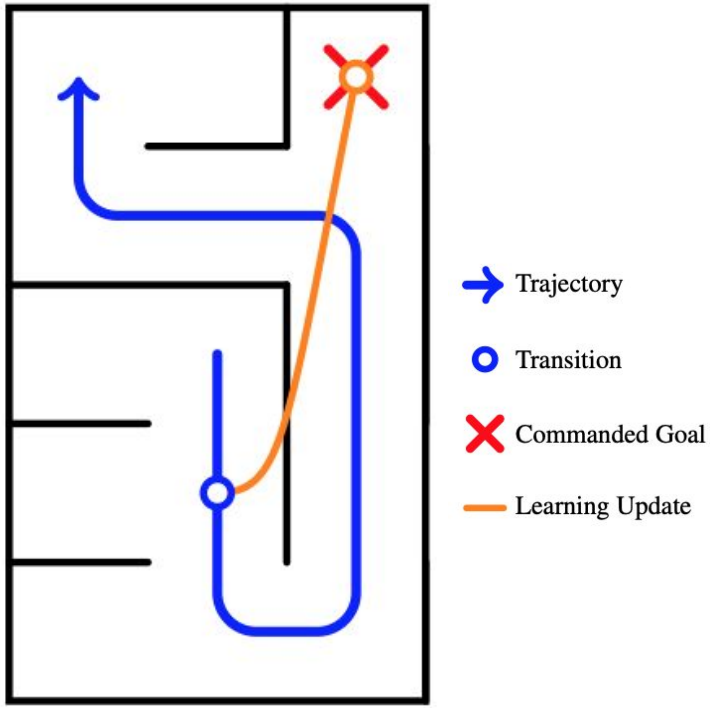




How do we deal with suboptimal trajectories?

Vanilla GCRL: the update is done with respect to the goal that was commanded, *even if this goal is never satisfied*.

- Simply treat this like any other RL environment and learn on the gathered trajectory, with respect to the commanded goal.
- Implicit assumption is that the goal space is just a component of the state space.



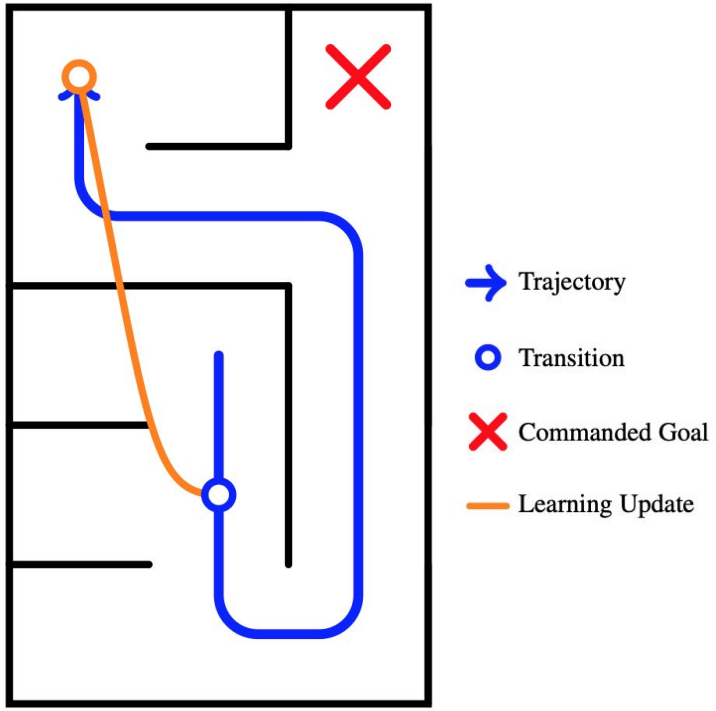
(a) Vanilla GCRL

How do we deal with suboptimal trajectories?

Goal Relabelling: take transitions that were gathered through commanding some goal *g* and use these to learn with respect to some different goal *g'*.

"What we are doing here is decomposing our decision problem into a true underlying environment and a goal conditioning layer, which can affect the reward but not the transition dynamics"

Hindsight Experience Replay: relabel with a goal *g'* that is achieved later in the episode.



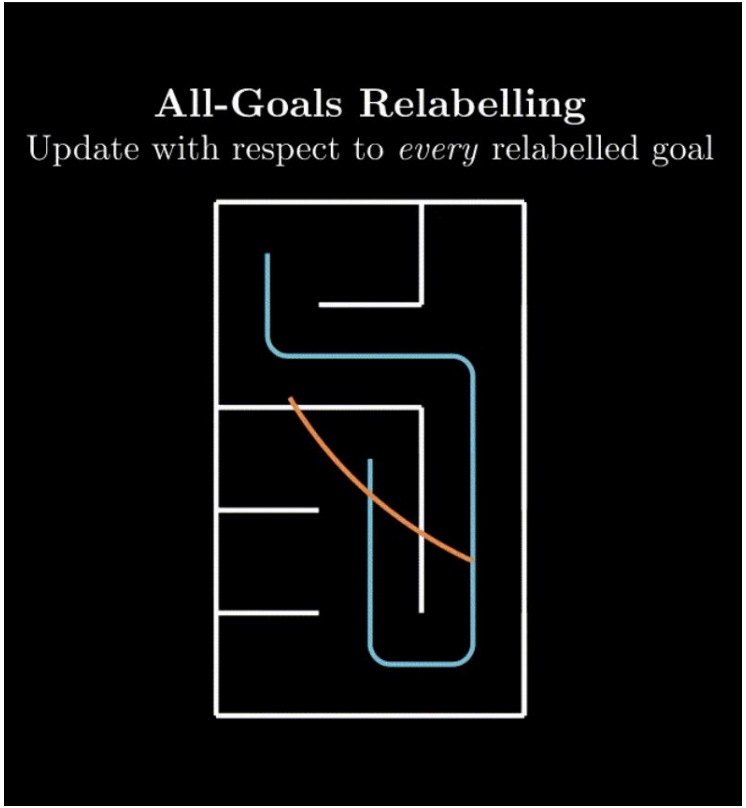
(b) HER

How do we deal with suboptimal trajectories?

We don’t even have to relabel with just a single goal – *couldn’t we relabel with many?*

We extract **maximal information** from every transition by considering how it relates to every goal in our goal set.

However, this is incredibly slow, as it blows up your training data by a factor of $|G|$. This is impractical for any non-trivial goal set.



Learning Everything All at Once (LEO)

Let's consider the problem of learning a goal-conditioned Q-network over a finite goal set *G* in the discrete action case (Universal Value Function Approximator - UVFA).

Naively, this could be done by learning a function that conditions on both the state and the goal and outputs an array of Q-values, one for each discrete action.

Given a tuple (s, a, s'), we could then perform the off-policy Q-learning update:

$$\mathcal{L}(g) = (\mathcal{R}(s', g) + \gamma \cdot \max_{a'} Q(a'|s', g) - Q(a|s, g))^2$$

Learning Everything All at Once (LEO)

We could relabel the trajectory as many times as we'd like to multiply the information content, but this becomes intractable for non-trivial goal sets.

The paper proposes learning an all-goals Q-function:

$$Q(s) : \mathcal{S} \rightarrow \mathbb{R}^{\mathcal{G} \times \mathcal{A}}$$

Rather than outputting a 1D array of Q-values (over actions) for each input state and goal, the all-goals Q-function outputs a 2D array of Q-values (over actions and goals) for each input state.

We can then vectorize over all goals:

$$\mathcal{L} = (\boldsymbol{\mathcal{R}}(s') + \gamma \cdot \max_{a'} \boldsymbol{Q}(a'|s') - \boldsymbol{Q}(a|s))^2$$

TLDR: Move goals from input axis to output axis.

Dual LEO

Consider using a LEO Q-network at inference time to take greedy actions. While we may know a priori which goal we wish to command, the network does not.

The network must produce outputs for every goal and assign equal weighting of its finite computation to each one, even though we will throw away the outputs for all but one goal goal.

To overcome this, they train both a LEO and UVFA network simultaneously.

- The LEO network can act as a “data sponge” that does not miss any crucial transitions with respect to any goal
- The UVFA network learns high fidelity estimates for the commanded goal.

Dual LEO

Consider using a LEO Q-network at inference time to take greedy actions. While we may know a priori which goal we wish to command, the network does not.

The network must produce outputs for every goal and assign equal weighting of its finite computation to each one, even though we will throw away the outputs for all but one goal goal.

To overcome this, they train both a LEO and UVFA network simultaneously.

- The LEO network can act as a “data sponge” that does not miss any crucial transitions with respect to any goal
- The UVFA network learns high fidelity estimates for the commanded goal.

Dual LEO (PQN)

Train two separate Q-networks on the same experience (data stream).

- Networks are updated independently using its own estimates for bootstrapping.

When acting toward commanded goal *g*, Dual LEO combines the two Q-estimates:

$$\alpha \cdot Q_{\text{LEO}}(s, a, g) + (1 - \alpha) \cdot Q_{\text{UVFA}}(s, a, g)$$

If there is difficult goal *g* sampled that has never yet been successfully achieved, the Q-value for it from UVFA will be ~0 but the Q-value from LEO > 0

This provides a commanded example of the goal being solved, giving signal to the UVFA network to start forming its own value estimates.

Use small alpha so that the linear combination will come close to just the high fidelity UVFA approximation as the magnitudes of its Q-estimates increase.

Dual LEO (PPO)

A PPO actor-critic network is trained as per normal, with actions taking by the PPO actor, while a LEO network trains off-policy on the same generated stream of data.

We then add losses to push the PPO policy $\pi(s, g)$ towards the greedy LEO policy $\text{argmax}_a Q_{\text{LEO}}(s, a, g)$ and the value network $V(s, g)$ towards the LEO value estimate $\max_a Q_{\text{LEO}}(s, a, g)$.

Unlike PQN, the action is not selected by linearly combining LEO and UVFA Q-values but by the PPO actor.

LEO for Continuous Control

You cannot output a Q-value for every possible continuous action. Therefore, LEO is extended to an **actor-critic** method.

Again, curry the goals from the observation space to the output space in the policy network to obtain an **all-goals policy**.

Then define the all-goals critic update (swap maximization in factor of actions selected by the all-goals policy):

$$\mathcal{L}_Q = (\boldsymbol{\mathcal{R}}(s') + \gamma \cdot \boldsymbol{Q}(s', \boldsymbol{\pi}(s')) - \boldsymbol{Q}(s, a))^2$$

The policy can still be updated with *(maximize the critic score)*:

$$\mathcal{L}_\pi = -\boldsymbol{Q}(s, \boldsymbol{\pi}(s))$$

But needs |*G*| backward passes :(

CraftaxGC

- We adapt Craftax into a goal- conditioned benchmark by defining a large set of semantic conditions: having different amounts of each item in inventory, being adjacent to specific blocks, items, or creatures, obtaining tools and weapons, reaching dungeon levels, and acquiring experience points or enchantments.

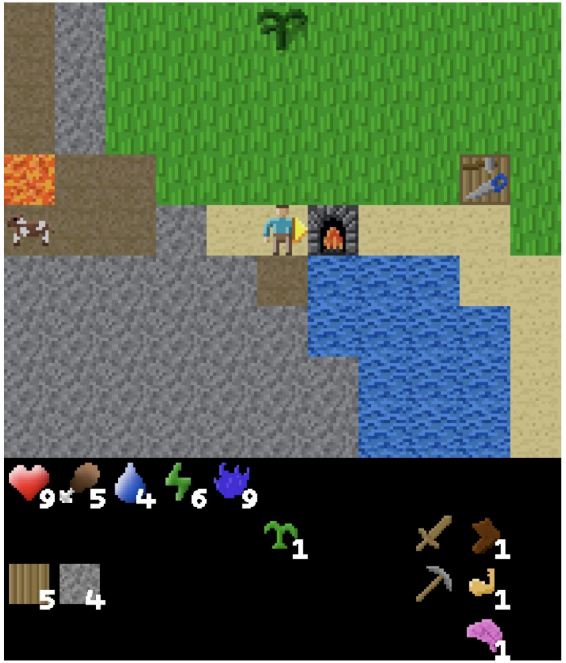


Figure 3. Example frame from CraftaxGC. Goals completed in this frame include `tools/stone_pickaxe`, `inventory/wood_5` and `block_map/furnace_right`.

CraftaxGC Results

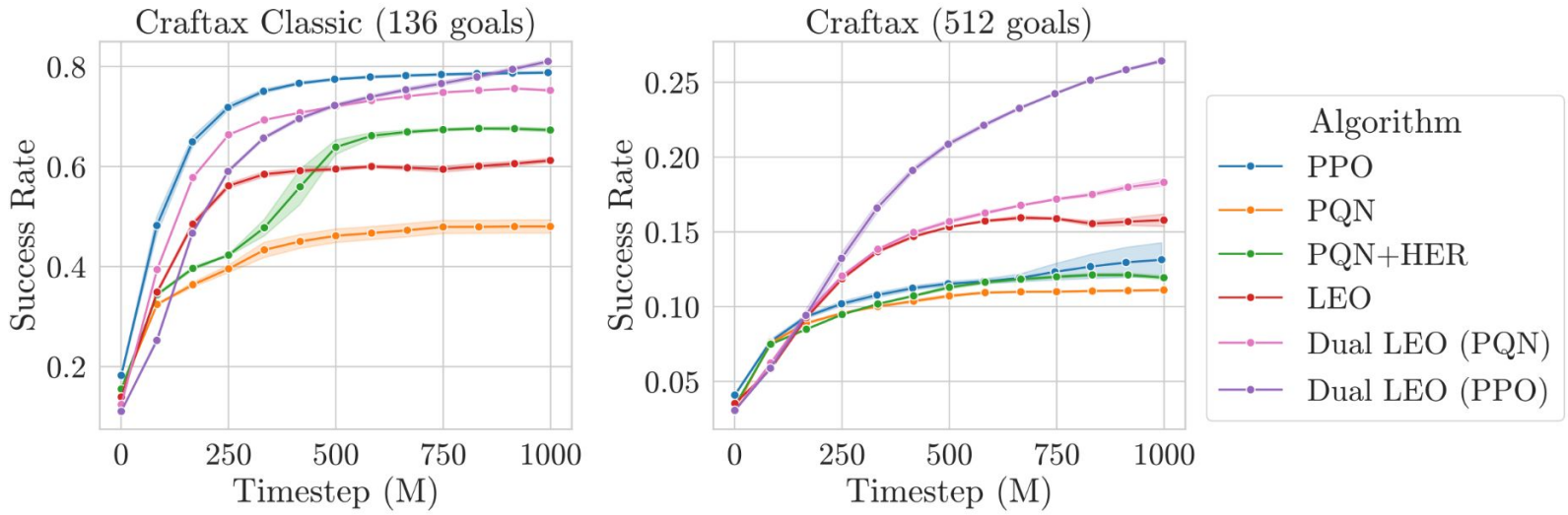


Figure 4. Mean success rate across all goals on CraftaxGC. The shaded area denotes 1 standard error over 5 seeds. We see LEO outperforming UVFA-style baselines on the larger goal set, but not on the smaller one. Dual LEO performs well in both cases, with the PPO variant achieving the best final performance in both settings.

Dual LEO Components

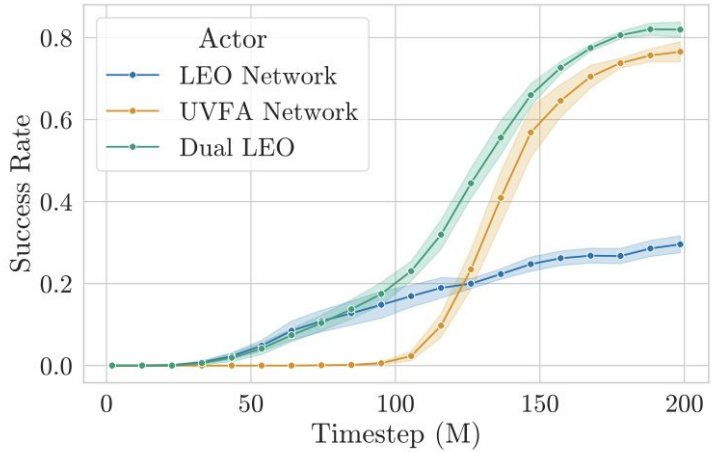


Figure 6. Mean success rate for the `inventory/coal-1` goal for Dual LEO (PQN), when acting greedily with respect to each of its components. The shaded area denotes 1 standard error over 5 seeds. Validating our hypothesis in Section 3.2, we see that the LEO network learns to achieve the goal early, providing positive examples of goal completion that allows the UVFA network to learn on.

Ant Maze Tasks

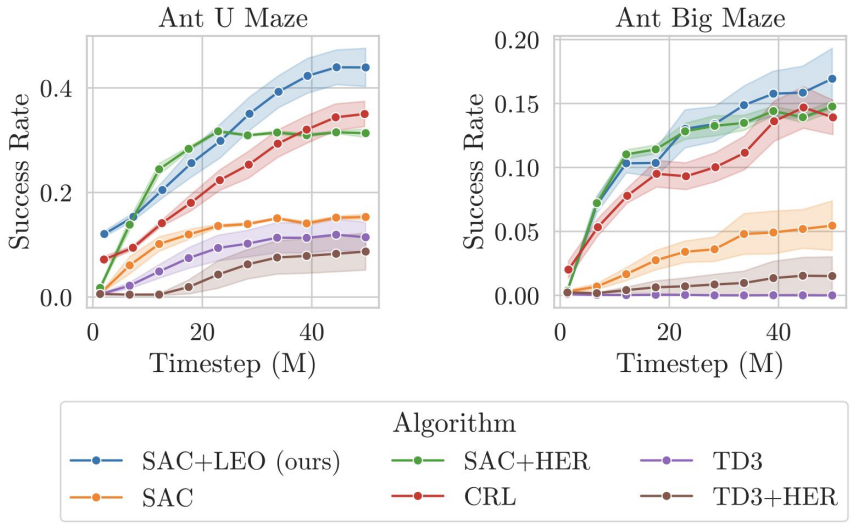


Figure 7. Mean success rate on JaxGCRL ant maze tasks for 50 million timesteps. The shaded area denotes 1 standard error over 5 seeds. Note that although LEO quantises the goal space internally, goal sampling for training and evaluation is done on the true underlying continuous goals just as with the other methods.

"On the larger maze the results are less clear, with LEO, SAC+HER and CRL all performing similarly".

This could be because the main difficulty in the ant maze tasks is learning the locomotive gait, rather than the differing goal positions. This is in contrast to Craftax where the different goals initiate very different and often opposing behaviours, with the low level locomotive task largely abstracted away.