MoCoDA: Model-based Counterfactual Data Augmentation

Anonymous Authors[1]

Abstract

The number of states in a dynamic process is exponential in the number of objects, making reinforcement learning (RL) difficult in complex, multi-object domains. For agents to scale to the real world, they will need to react to and reason about unseen combinations of objects. We argue that the ability to recognize and use local factorization in transition dynamics is a key element in unlocking the power of multi-object reasoning. To this end, we show that (1) known local structure in the environment transitions is sufficient for an exponential reduction in the sample complexity of training a dynamics model, and (2) a locally factored dynamics model provably generalizes out-of-distribution to unseen states and actions. Knowing the local structure also allows us to predict *which* unseen states and actions this dynamics model will generalize to. We propose to leverage these observations in a novel Model-based Counterfactual Data Augmentation (MoCoDA) framework. MoCoDA applies a learned locally factored dynamics model to an augmented distribution of states and actions to generate counterfactual transitions for RL. MoCoDA works with a broader set of local structures than prior work and allows for direct control over the augmented training distribution. We show that MoCoDA enables RL agents to learn policies that generalize to unseen states and actions. We use MoCoDA to train an offline RL agent to solve an out-of-distribution robotics manipulation task on which standard offline RL algorithms fail.[1]

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

[1]Visualizations & code to be available at https://sites.google.com/view/mocoda-darl/

1 Introduction

Modern reinforcement learning (RL) algorithms have demonstrated remarkable success in several different domains such as games (Mnih et al., 2015; Silver et al., 2017) and robotic manipulation (Kalashnikov et al., 2018; Andrychowicz et al., 2020). By repeatedly attempting a single task through trial-and-error, these algorithms can learn to collect useful experience and eventually solve the task of interest. However, designing agents that can generalize in *off-task* and *multi-task* settings remains a open and challenging research question. This is especially true in the offline and zero-shot settings, in which the training data might be unrelated to the target task, and may lack sufficient coverage over possible states.

One way to enable generalization in such cases is through structured representations of states, transition dynamics or task spaces. These representations can be directly learned, sourced from known or learned abstractions over the state space, or derived from causal knowledge of the world. Symmetries present in such representations enable compositional generalization to new configurations of states or tasks, either by building the structure into the function approximator or algorithm (Kipf et al., 2020; Veerapaneni et al., 2020; Goyal et al., 2021; Nangue Tasse et al., 2020), or by using the structure for data augmentation (Andrychowicz et al., 2017; Laskin et al., 2020; Pitis et al., 2020b).

In this paper, we extend past work on structure-driven data augmentation by using a locally factored model of the transition dynamics to generate counterfactual training distributions. This enables agents to generalize beyond the support of their original training distribution, including to novel tasks where learning the optimal policy requires access to states never seen in the experience buffer. Our key insight is that a learned dynamics model that accurately captures local causal structure (a "locally factored" dynamics model) will predictably exhibit good generalization performance outside the empirical training distribution. We propose Model-based Counterfactual Data Augmentation (MoCoDA), which generates an augmented state-action distribution where its locally factored dynamics model is likely to perform well, then applies its dynamics model to generate new transition data. By training the agent's policy and value modules on this augmented dataset, they too learn to generalize well out-

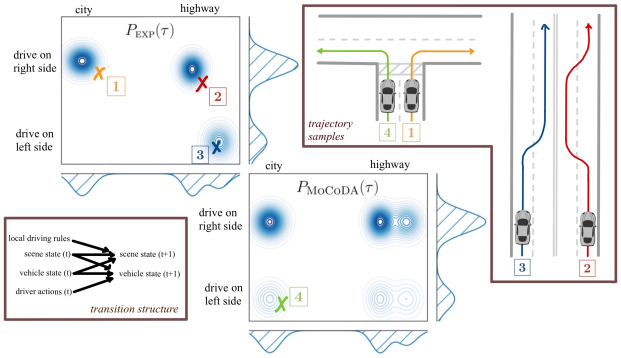


Figure 1: **Out-of-Distribution Generalization using MoCoDA**: A US driver can use MoCoDA to quickly adapt to driving in the left lane during a UK trip. Their prior experience $P_{\text{EMP}}(\tau)$ (**top left**) contains mostly right-driving experience (e.g. 1, 2) and a limited amount of left-driving experience after renting the car in the UK (e.g. 3). A locally factored model that captures the transition structure (**bottom left**) allows the agent to accurately sample counterfactual experience from $P_{\text{MoCoDA}}(\tau)$ (**bottom center**), including novel left-lane city driving maneuvers (e.g. 4). This enables fast adaptation when learning an optimal policy for the new task (UK driving). Our framework MoCoDA draws single-step transition samples from $P_{\text{MoCoDA}}(\tau)$ given $P_{\text{EMP}}(\tau)$ and knowledge of the causal structure; several realizations of this framework are described in Section 4.

of-distribution. To ground this in an example, we consider how a US driver might use MoCoDA to adapt to driving on the left side of the road while on vacation in the UK (Figure 1). Given knowledge of the target task, we can even focus the augmented distribution on relevant areas of the state-action space (e.g., states with the car on the left side of the road).

Our main contributions are:

A. Our proposed method, MoCoDA, leverages a masked dynamics model for data-augmentation in locally-factored settings, which relaxes strong assumptions made by prior work on factored MDPs and counterfactual data augmentation.

B. MoCoDA allows for direct control of the state-action distribution on which the agent trains; we show that controlling this distribution in a task relevant way can lead to improved performance.

C. We demonstrate "zero-shot" generalization of a policy trained with MoCoDA to states that the agent has never seen. With MoCoDA, we train an offline RL agent to solve an out-of-distribution robotics manipulation task on which standard offline RL algorithms fail.

2 Preliminaries

We model the environment as an infinite-horizon, reward-free Markov Decision Process (MDP), described by tuple $\langle \mathcal{S}, \mathcal{A}, P, \gamma \rangle$ consisting of the state space, action space, transition function, and discount factor, respectively (Puterman, 2014; Sutton and Barto, 2018). We use lowercase for generic instances and uppercase for variables (e.g., $s \in \text{range}(S) \subseteq \mathcal{S}$, though we also abuse notation and write $S \in \mathcal{S}$). A *task* is defined as a tuple $\langle r, P_0 \rangle$, where $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a reward function and P_0 is an initial distribution over S. The goal of the agent given a task is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes value $\mathbb{E}_{P,\pi} \sum_t \gamma^t r(s_t, a_t)$. Model-based RL (MBRL) is one approach to solving this problem, in which the agent learns a model P_θ of the transition dynamics P. The model is "rolled out" to generate "imagined" trajectories, which are used either for direct planning (De Boer et al., 2005; Chaslot et al., 2008), or as training data for the agent's policy and value functions (Sutton, 1991; Janner et al., 2019).

Factored MDPs. A factored MDP (FMDP) is a type of MDP that assumes a globally factored transition model, which can be used to exponentially improve the sample complexity of RL (Guestrin et al., 2003; Kearns and Koller, 1999; Osband and Van Roy, 2014). In an FMDP, states and actions are described by a set of variables $\{X^i\}$, so that $\mathcal{S} \times \mathcal{A} = \mathcal{X}^1 \times \mathcal{X}^2 \times \ldots \times \mathcal{X}^n$, and each state variable $X^i \in \mathcal{X}^i$ ($\mathcal{X}^i$ is a subspace of $\mathcal{S}$) is dependent on a subset of state-action variables (its "parents" $\text{Pa}(X^i)$) at the prior timestep, $X^i \sim P_i(\text{Pa}(X^i))$. We call a set $\{X^j\}$ of state-action variables a "parent set" if there exists a state variable X^i such that $\{X^j\} = \text{Pa}(X^i)$. We say that X^i is a "child" of its parent set $\text{Pa}(X^i)$. We refer to the tuple $\langle X^i, \text{Pa}(X^i), P_i(\cdot) \rangle$ as a "causal mechanism".

Local Causal Models. Because the strict global factorization assumed by FMDPs is rare, recent work on data augmentation for RL and object-oriented RL suggests that transition dynamics might be better understood in a local sense, where all objects may interact with each other over time, but in a locally sparse manner (Goyal et al., 2021; Kipf et al., 2020). Our work uses an abridged version of the Local Causal Model (LCM) framework (Pitis et al., 2020b), as follows: We assume the state-action space decomposes into a disjoint union of local neighborhoods: $\mathcal{S} \times \mathcal{A} = \mathcal{L}_1 \sqcup \mathcal{L}_2 \sqcup \cdots \sqcup \mathcal{L}_n$. A neighborhood $\mathcal{L}$ is associated with its own transition function $P^{\mathcal{L}}$, which is factored according to its graphical model $\mathcal{G}^{\mathcal{L}}$ (Koller and Friedman, 2009). We assume no two graphical models share the same structure (i.e., the structure of $\mathcal{G}^{\mathcal{L}}$ uniquely identifies $\mathcal{L}$). Then, analogously to FMDPs, if $(s_t, a_t) \in \mathcal{L}$, each state variable X^i_{t+1} at the next time step is dependent on its parents $\text{Pa}^{\mathcal{L}}(X^i_{t+1})$ at the prior timestep, $X^i_{t+1} \sim P^{\mathcal{L}}_i(\text{Pa}^{\mathcal{L}}(X^i_{t+1}))$. We define mask function $M : \mathcal{S} \times \mathcal{A} \to \{\mathcal{L}_i\}$ that maps $(s, a) \in \mathcal{L}$ to the adjacency matrix of $\mathcal{G}^{\mathcal{L}}$. This formalism is summarized in Figure 2, and differs from FMDPs in that each $\mathcal{L}$ has its own factorization.

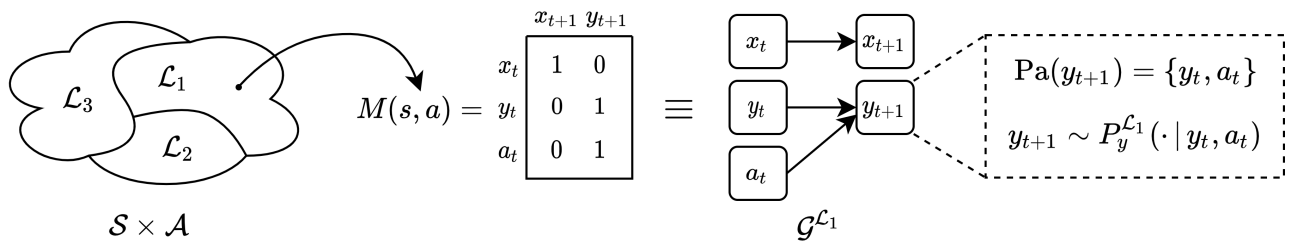


Figure 2: **Locally Factored Dynamics:** The state-action space $\mathcal{S} \times \mathcal{A}$ is divided into local subsets, $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$, which each have their own factored causal structure, $\mathcal{G}^{\mathcal{L}}$. The local transition model $P^{\mathcal{L}}$ is factored according to $\mathcal{G}^{\mathcal{L}}$; e.g., in the example shown, $P^{\mathcal{L}}(x_t, y_t, a_t) = [P_x(x_t), P_y(y_t, a_t)]$.

Given knowledge of M, the Counterfactual Data Augmentation (CoDA) framework (Pitis et al., 2020b) allowed model-free agents to stitch together disconnected components from two different transitions to create new, causally valid data. Our proposed framework (MoCoDA) *leverages a dynamics model to improve upon model-free CoDA* in several respects: (a) MoCoDA works with overlapping parent sets, (b) MoCoDA allows the agent to control the base data distribution, (c) MoCoDA demonstrates zero-shot generalization to new areas of the state space, allowing the agent to solve tasks that are entirely outside the original data distribution.

3 Generalization Properties of Locally Factored Models

3.1 Sample Complexity of Training a Locally Factored Dynamics Model

In this subsection, we provide an original adaptation of an elementary result from model-based RL to the *locally* factored setting, to show that factorization can exponentially improve sample complexity. We note that several theoretical works have shown that the FMDP structure can be exploited to obtain similarly strong sample complexity bounds in the FMDP setting. Our goal here is not to improve upon these results, but to adapt a small part (model-based generalization) to the significantly more general locally factored setting and show that local factorization is enough for (1) *exponential gains in sample complexity* and (2) *out-of-distribution generalization* with respect to the empirical joint, to a set of states and actions that may be exponentially larger than the empirical set. Note that the following discussion applies to tabular RL, but we apply our method to continuous domains.

Notation. We work with finite state ($|\mathcal{S}|$) and action ($|\mathcal{A}|$) spaces and assume that there are m local subspaces $\mathcal{L}$ of size $|\mathcal{L}|$, such that $m|\mathcal{L}| = |\mathcal{S}||\mathcal{A}|$. For each subspace $\mathcal{L}$, we assume transitions factor into k causal mechanisms $\{P_i\}$, each with the same number of possible children, $|c_i|$, and the same number of possible parents, $|\mathrm{Pa}_i|$. Note $m\Pi_i|c_i| = |\mathcal{S}|$ (child sets are mutually exclusive) but $m\Pi_i|\mathrm{Pa}_i| \geq |\mathcal{S}||\mathcal{A}|$ (parent sets may overlap).

Theorem 1. *Let n be the number of empirical samples used to train the model of each local causal mechanism, $P^{\mathcal{L}}_{i,\theta}$ at each configuration of parents $\mathrm{Pa}_i = x$. There exists constant c such that, if*

$$n \geq \frac{ck^2|c_i| \log(|\mathcal{S}||\mathcal{A}|/\delta)}{\epsilon^2},$$

then, with probability at least $1 - \delta$, we have:

$$\max_{(s,a)} \|P(s,a) - P_\theta(s,a)\|_1 \leq \epsilon.$$

Sketch of Proof. We apply a concentration inequality to bound the ℓ_1 error for fixed parents and extend this to a bound on the ℓ_1 error for a fixed (s,a) pair. The conclusion follows by a union bound across all states and actions. See Appendix A for details. □

To compare to full-state dynamics modeling, we can translate the sample complexity from the per-parent count n to a total count N. Recall $m\Pi_i|c_i| = |\mathcal{S}|$, so that $|c_i| = (|\mathcal{S}|/m)^{1/k}$, and $m\Pi_i|\mathrm{Pa}_i| \geq |\mathcal{S}||\mathcal{A}|$. We assume a small constant overlap factor $v \geq 1$, so that $|\mathrm{Pa}_i| = v(|\mathcal{S}||\mathcal{A}|/m)^{1/k}$. We need the total number of component visits to be $n|\mathrm{Pa}_i|km$, for a total of $nv(|\mathcal{S}||\mathcal{A}|/m)^{1/k}m$ state-action visits, assuming that parent set visits are allocated evenly, and noting that each state-action visit provides k parent set visits. This gives:

Corollary 1. *To bound the error as above, we need to have*

$$N \geq \frac{cmk^2(|\mathcal{S}|^2|\mathcal{A}|/m^2)^{1/k} \log(|\mathcal{S}||\mathcal{A}|/\delta)}{\epsilon^2},$$

total train samples, where we have absorbed the overlap factor v into constant c.

Comparing this to the analogous bound for full-state model learning (Agarwal et al. (2019), Prop. 2.1):

$$N \geq \frac{c|\mathcal{S}|^2|\mathcal{A}| \log(|\mathcal{S}||\mathcal{A}|/\delta)}{\epsilon^2},$$

we see that we have gone from super-linear $O(|\mathcal{S}|^2|\mathcal{A}| \log(|\mathcal{S}||\mathcal{A}|))$ sample complexity in

terms of $|S||A|$, to the exponentially smaller $O(mk^2(|\mathcal{S}|^2|\mathcal{A}|/m^2)^{1/k}\log(|\mathcal{S}||\mathcal{A}|))$.

This result implies that *for large enough $|\mathcal{S}||\mathcal{A}|$ our model must generalize to unseen states and actions*, since the number of samples needed (N) is exponentially smaller than the size of the state-action space ($|\mathcal{S}||\mathcal{A}|$). In contrast, if it did not, then sample complexity would be $\Omega(|\mathcal{S}||\mathcal{A}|)$.

Remark 3.1. *The global factorization property of FMDPs is a strict assumption that rarely holds in reality. Although local factorization is broadly applicable and significantly more realistic than the FMDP setting, it is not without cost. In FMDPs, we have a single subspace ($m = 1$). In the locally factored case, the number of subspaces m is likely to grow exponentially with the number of factors k, as there are exponentially many ways that k factors can interact. To be more precise, there are $k2^k$ possible bipartite graphs from k nodes to k nodes. Nevertheless, by comparing bases ($2 \ll |\mathcal{S}||\mathcal{A}|$), we see that we still obtain exponential gains in sample complexity from the locally factored approach.*

3.2 Training Value Functions and Policies for Out-of-Distribution Generalization

In the previous subsection, we saw that a locally factored dynamics model provably generalizes outside of the empirical joint distribution. A natural question is whether such *local factorization can be leveraged to obtain similar results for value functions and policies*?

We will show that the answer is *yes*, but perhaps counter-intuitively, it cannot be achieved by training on the empirical distribution, as was the case for dynamics models. The difference arises because learned value functions, and consequently learned policies, involve the long horizon prediction $\mathbb{E}_{P,\pi}\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)$, which can no longer take advantage of the sparsity of $\mathcal{G}^{\mathcal{L}}$. When compounded over time, sparse local structures can quickly produce an entangled long horizon structure (cf. the "butterfly effect"). Intuitively, even if several pool balls are far apart and locally disentangled, future collisions are central to planning and the optimal policy depends on the relative positions of all balls. This applies even if rewards are factored (e.g., rewards in most pool variants) (Sodhani et al., 2022).

We note that, although temporal entanglement may be exponential in the branching factor of the unrolled causal graph, it's possible for the long horizon structure to stay sparse (e.g., k independent factors that never interact). It's also possible that other regularities in the data will allow for good out-of-distribution generalization. Thus, we cannot claim that value functions and policies will never generalize well out-of-distribution (see Veerapaneni et al. (2020) for an example when they do). Nevertheless, we hypothesize that exponentially fast entanglement does occur in complex natural systems, making direct generalization of long horizon predictions difficult.

Out-distribution generalization of the policy and value function can be achieved, however, by leveraging the generalization properties of a locally factored dynamics model. We propose to do this by generating out-of-distribution states and actions (the "parent distribution"), and then applying our learned dynamics model to generate transitions that are used to train the policy and value function. We call this process Model-based Counterfactual Data Augmentation (MoCoDA).

4 Model-based Counterfactual Data Augmentation

In the previous section, we discussed how locally factored dynamics model can generalize beyond the empirical dataset to provide accurate predictions on an augmented state-action distribution we call the "parent distribution". We now seek to leverage this out-of-distribution generalization in the dynamics model to bootstrap the training of an RL agent. Our approach is to control the agent's training distribution $P(s, a, s')$ via the locally factored dynamics $P_\phi(s'|s, a)$ and the parent distribution $P_\theta(s, a)$ (both trained using experience data). This allows us to sample *augmented* transitions (perhaps unseen in the experience data) for consumption by a downstream RL agent. We call this framework MoCoDA, and summarize it using the following three-step process:

S1 Given known parent sets, generate appropriate parent distribution $P_\theta(s, a)$.

S2 Apply a learned dynamics model $P_\phi(s'|s, a)$ to parent distribution to generate "augmented dataset" of transitions (s, a, s').

S3 Use augmented dataset $s, a, s' \sim P_\theta P_\phi$ (alongside experience data, if desired) to train an off-policy RL agent on the (perhaps novel) target task.

Figure 3 illustrates this framework in a block diagram. An instance of MoCoDA is realized by specific choices at each step. For example, the original CoDA method (Pitis et al., 2020b) is an instance of MoCoDA, which (1) generates the parent distribution by uniformly swapping non-overlapping parent sets, and (2) uses subsamples of empirical transitions as a locally factored dynamics model. CoDA works when local graphs have non-overlapping parent sets, but it does not allow for control over the parent distribution and does not work in cases where parent sets overlap. MoCoDA generalizes CoDA, alleviating these restrictions and allowing for significantly more design choices, discussed next.

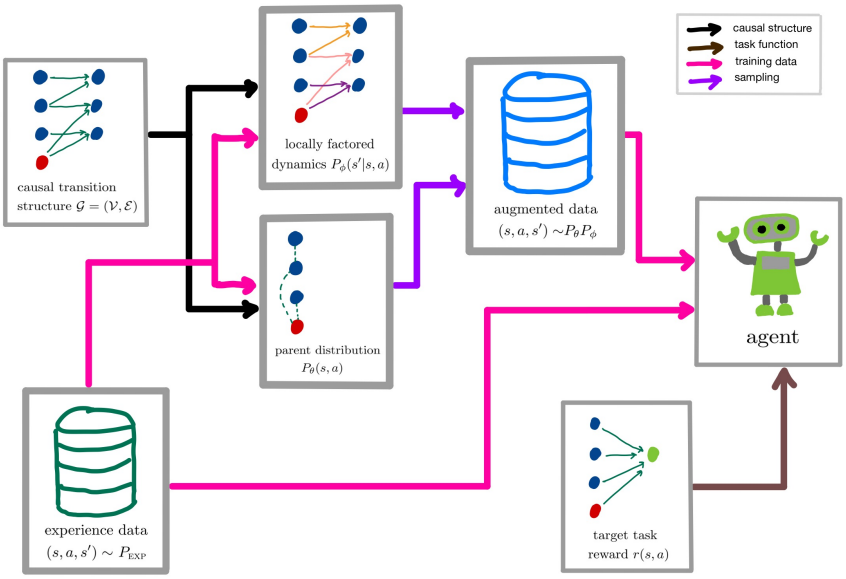


Figure 3: **Training an RL Agent with MoCoDA**: We use the empirical dataset to train parent distribution model, $P_\theta(s,a)$ and locally factored dynamics model $P_\phi(s' \mid s,a)$, both informed by the local structure. The dynamics model is applied to the parent distribution to produce augmented dataset $P_\theta P_\phi$. The augmented & empirical datasets are labeled with the target task reward, $r(s,a)$ and fed into the RL algorithm as training data.

4.1 Generating the Parent Distribution

What parent distribution (Step 1) should be used to generate the augmented dataset? We describe some options below, noting that our proposals (MoCoDA, MoCoDA-U, MoCoDA-P) rely on knowledge of parent sets—i.e., they require the state to be decomposed into objects.

Baseline Distributions. If we restrict ourselves to states and actions in the empirical dataset (**Emp**) or short-horizon rollouts that start in the empirical state-action distribution (**Dyna**), as is typical in Dyna-style approaches (Sutton and Barto, 2018; Janner et al., 2019), we limit ourselves to a small neighborhood of the empirical state-action distribution. This forgoes the opportunity to train our off-policy RL agent on out-of-distribution data that may be necessary for learning the target task.

Another option is to sample random state-actions from $\mathcal{S} \times \mathcal{A}$ (**Rand**). While this provides coverage of all state-actions relevant to the target task, there is no guarantee that our locally factorized model generalizes well in Rand. The proof of Theorem 1 shows that our model only generalizes well to a particular (s,a) if each component generalizes well on the configurations of each parent set in that (s,a). In context of Theorem 1, this occurs only if the empirical data used to train our model contained at least n samples for each set of parents in (s,a). This suggests focusing on data whose parent sets have sufficient support in the empirical dataset.

The MoCoDA distribution. We do this by constraining the marginal distribution of each parent set (within local neighborhood $\mathcal{L}$) in the augmented distribution to match the corresponding marginal in the empirical dataset. As there are many such distributions, in absence of additional information, it is sensible to choose the one with maximum entropy (Jaynes, 1957). We call this maximum entropy, marginal matching distribution the **MoCoDA** augmented distribution. Figure 1 provides an illustrative example of going from Emp (driving primarily on the right side) to MoCoDA (driving on both right and left). We propose an efficient way to generate the MoCoDA distribution using a set of Gaussian Mixture Models, one for each parent set distribution. We sample parent sets one at a time, conditioning on any previous partial samples due to overlap between parent sets. This process is detailed in Appendix B.

Weaknesses of the MoCoDA distribution. Although our locally factored dynamics model is likely to generalize well on MoCoDA, there are a few reasons why training our RL agent on MoCoDA in Step 3 may yield poor results. First, if there are empirical imbalances within parent sets (some parent configurations more common than others), these imbalances will appear in MoCoDA. Moreover, multiple such imbalances will compound exponentially, so that (s,a) tuples with rare parent combinations will be extremely rare in MoCoDA, even if the model generalizes well to them. Second, Support(MoCoDA) may be so large that it makes training the RL algorithm in Step 3 inefficient. Finally, the cost function used in RL algorithms is typically an expectation over the training distribution, and optimizing the agent in irrelevant areas of the state-action space may hurt performance. The above limitations suggest that rebalancing MoCoDA might improve results.

MOCODA-U and MOCODA-P. To mitigate the first weakness of MOCODA we might skew MOCODA toward the uniform distribution over its support, $\mathcal{U}(\text{Support}(\text{MOCODA}))$. Although this is possible to implement using rejection sampling when k is small, exponential imbalance makes it impractical when k is large. A more efficient implementation reweights the GMM components used in our MOCODA sampler. We call this approach (regardless of implementation) **MOCODA-U**. To mitigate the second and third weaknesses of MOCODA, we need additional knowledge about the target task—e.g., domain knowledge or expert trajectories. We can use such information to define a prioritized parent distribution **MOCODA-P** with support in Support(MOCODA), which can also be obtained via rejection sampling (perhaps on MOCODA-U to also relieve the initial imbalance).

4.2 The Choice of Dynamics Model and RL Algorithm

Once we have a parent distribution, $P_\theta(s, a)$, we generate our augmented dataset by applying dynamics model $P_\phi(s' \mid s, a)$. The natural choice in light of the discussion in Section 3 is a locally factored model. This requires knowledge of the local factorization, which is more involved than the parent set knowledge used to generate the MOCODA distribution and its reweighted variants. We note, however, that a locally factored model may not be strictly necessary for MOCODA, so long as the underlying dynamics are factored. Although unfactored models do not perform well in our experiments, we hypothesize that a good model with enough in-distribution data and the right regularization might learn to implicitly respect the local factorization. The choice of model architecture is not core to our work, and we leave exploration of this possibility to future work.

Masked Dynamics Model. In our experiments, we assume access to a mask function $M : \mathcal{S} \times \mathcal{A} \to \{0, 1\}^{(|\mathcal{S}|+|\mathcal{A}|)\times|\mathcal{S}|}$ (perhaps learned (Kipf et al., 2018; Pitis et al., 2020b)), which maps states and actions to the adjacency map of the local graph $\mathcal{G}^{\mathcal{L}}$. Given this mask function, we design a dynamics model P_ϕ that accepts $M(s, a)$ as an additional input and respects the causal relations in the mask (i.e., mutual information $I(X_t^i; X_{t+1}^j \mid (S_t, A_t) \setminus X_t^i) = 0$ if $M(s_t, a_t)_{ij} = 0$. There are many architectures that enforce this constraint. In our experiments we opt for a simple one, which first embeds each of the k parent sets: $f = [f_i(\text{Pa}_i)]_{i=1}^k$, and then computes the j-th child as a function of the sum of the masked embeddings, $g_j(M(s, a)_{\cdot,j} \cdot f)$. See Appendix B for further implementation details.

The RL Algorithm. After generating an augmented dataset by applying our dynamics model to the augmented distribution, we label the data with our target task reward and use the result to train an RL agent. MOCODA works with a wide range of algorithms, and the choice of algorithm will depend on the task setting. For example, our experiments are done in an offline setup, where the agent is given a buffer of empirical data, with no opportunity to explore. For this reason, it makes sense to use offline RL algorithms, as this setting has proven challenging for standard online algorithms (Levine et al., 2020).

Remark 4.1. *The rationales for (1) regularizing the policy toward the empirical distribution in offline RL algorithms, and (2) training on the* MOCODA *distribution, are compatible: in each case, we want to restrict ourselves to state-actions where our models generalize well. By using* MOCODA *we expand this set beyond the empirical distribution. Thus, when we apply offline RL algorithms in our experiments, we train their offline component (e.g., the action sampler in* BCQ *(Fujimoto et al., 2019) or the BC constraint in* TD3-BC *(Fujimoto and Gu, 2021)) on the expanded* MOCODA *training distribution.*

5 Experiments

Hypotheses Our experiments are aimed at finding support for two critical hypotheses:

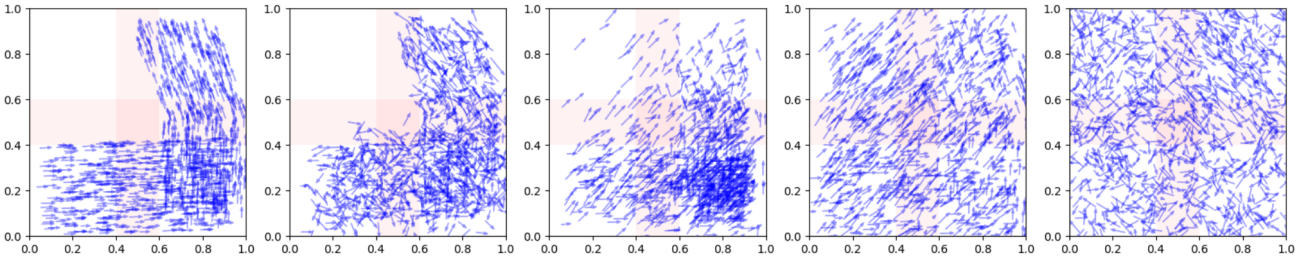
H1 Dynamics models, especially ones sensitive to the local factorization, are able to generalize well in the MOCODA distribution.

H2 This out-of-distribution generalization can be leveraged via data augmentation to train an RL agent to solve out-of-distribution tasks.

Note that support for **H2** provides implicit support for **H1**.

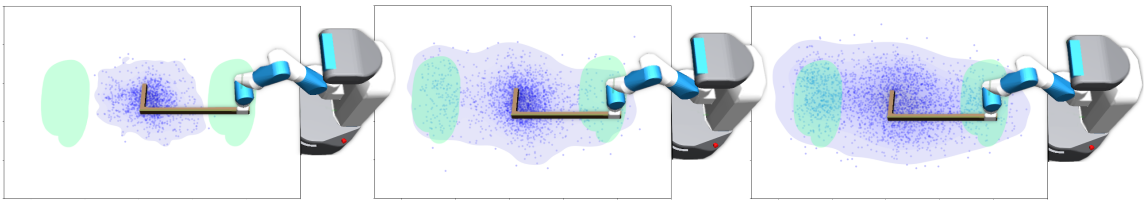
Domains We test MOCODA on two continuous control domains. First is a simple, but controlled, `2D Navigation` domain, where the agent must travel from one point in a square arena to another. States are 2D (x, y) coordinates and actions are 2D $(\Delta x, \Delta y)$ vectors. In most of the state space, the sub-actions Δx and Δy affect only their respective coordinate. In the top right quadrant, however, the Δx and Δy sub-actions each affect *both* x and y coordinates, so that the environment is locally factored. The agent has access to empirical training data consisting of left-to-right and bottom-to-top trajectories that are restricted to a ⌟ shape of the state space (see the EMP distribution in Figure 4). We consider a target task where the agent must move from the bottom left to the top right. In this task there is sufficient empirical data to solve the task by following the ⌟ shape of the data, but learning the optimal policy of going directly via the diagonal requires out-of-distribution generalization.

Second, we test MOCODA in a challenging `HookSweep2` robotics domain based on Hook-Sweep (Kurenkov et al., 2020), in which a Fetch robot must use a long hook to sweep two boxes to one side of the table (either toward or away from the agent). The boxes are initialized near the center



EMP DYNA MOCODA MOCODA-U RAND

Figure 4: **2D Navigation Visualization.** (Best viewed with 2x zoom) Blue arrows represent transition samples as a vector from (x_t, y_t) to (x_{t+1}, y_{t+1}). Shaded red areas mark the edges of the initial states of empirical trajectories and the center of the square. We see that 5-step rollouts (DYNA) do not fill in the center (needed for optimal policy), and fail to constrain actions to those that the model generalizes well on. For MOCODA, we see the effect of compounding dataset imbalance discussed in Subsection 4.1, which is resolved by MOCODA-U.



EMP MOCODA MOCODA-P

Figure 5: **HookSweep2 Visualization:** Stylized visualization of the distributions EMP (left), MOCODA (center), and MOCODA-P (right). Each figure can be understood as a top down view of the table, where a point is a plotted if the two blocks are close together on the table. The distribution EMP does not overlap with the green goal areas on the left and right, and so the agent is unable to learn. In the MOCODA distribution, the agent gets some success examples. In the MOCODA-P distribution, state-actions are reweighted so that the joint distribution of the two block positions is approximately uniform, leading to more evenly distributed coverage of the table.

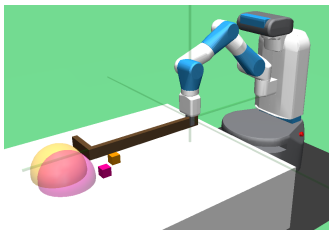


Figure 6: `HookSweep2` environment. At test time the agent must sweep both blocks to one side of the table, but empirical data contains only trajectories of a single block being sweeped.

of the table, and the empirical data contains trajectories of the agent sweeping exactly one box to one side of the table, leaving the other in the center. The target task requires the agent to generalize to states that it has never seen before (both boxes together on one side of the table). This is particularly challenging because the setup is entirely offline (no exploration), where poor out-of-distribution generalization typically requires special offline RL algorithms that constrain the agent's policy to the empirical distribution (Levine et al., 2020; Agarwal et al., 2020; Kumar et al., 2020; Fujimoto and Gu, 2021).

Directly comparing model generalization error. In the `2D Navigation` domain we have access to the ground truth dynamics, which allows us to directly compare generalization error on variety of distributions, visualized in Figure 4. We compare three different model architectures: unfactored, globally factored (assuming that the $(x, \Delta x)$ and $(y, \Delta y)$ causal mechanisms are independent everywhere, which is not true in the top right quadrant), and locally factored. The models are each trained on a empirical dataset of 35000 transitions for up to 600 epochs, which is early stopped using a validation set of 5000 transitions. The results are shown in Table 1. We find strong support for **H1**: even given the simple dynamics of `2d Navigation`, it is clear that the locally factored model is able to generalize better than a fully connected model, particularly on the MOCODA distribution, where performance degradation is minimal. We note that the DYNA distribution was formed by starting in EMP and doing 5-step rollouts with *random* actions. The random actions produce out-of-distribution data to which no model (not even the locally factored model) can generalize well to.

Solving out-of-distribution tasks. We apply the trained dynamics models to several base distributions and compare the performance of RL agents trained on each dataset. To ensure improvements are due to the augmented dataset and not agent architecture, we train several different algorithms, including: `SAC` (Haarnoja et al., 2018), `BCQ` (Fujimoto et al., 2019) (with DDPG (Lillicrap et al., 2016)), `CQL` (Kumar et al., 2020) and `TD3-BC` (Fujimoto and Gu, 2021).

The results on `2D Navigation` are shown in Table 2. We see that for all algorithms, the use of the MOCODA

Table 1: **2D Navigation Dynamics Modeling Results:** Mean squared error ± std. dev. over 5 seeds, scaled by 1e2 for clarity (best model boldfaced). The locally factored model experienced less performance degradation out-of-distribution, and performed better on all distributions, except for the empirical distribution (EMP) itself.

Model Architecture	**Generalization Error (MSE** $\times 1e2$**)** (lower is better)				
	EMP	DYNA	RAND	**MOCODA**	**MOCODA-U**
Not Factored	**0.14 ± 0.04**	2.41 ± 0.29	4.4 ± 0.31	0.95 ± 0.06	1.29 ± 0.15
Globally Factored	0.36 ± 0.01	2.09 ± 0.28	3.17 ± 0.3	0.41 ± 0.02	0.51 ± 0.02
Locally Factored	0.23 ± 0.1	**1.47 ± 0.27**	**2.03 ± 0.19**	**0.33 ± 0.11**	**0.46 ± 0.11**

Table 2: **2D Navigation Offline RL Results:** Average steps to completion ± std. dev. over 5 seeds for various RL algorithms (best distribution in each row boldfaced), where average steps was computed over the last 50 training epochs. Training on MOCODA and MOCODA-U improved performance in all cases. Interestingly, even using RAND improves performance, indicating the importance of training on out-of-distribution data. Note that this is an offline RL task, and so SAC (an algorithm designed for online RL) is not expected to perform well.

RL Algorithm	**Average Steps to Completion** (lower is better)			
	EMP	RAND	**MOCODA**	**MOCODA-U**
SAC (online RL)	53.1 ± 9.8	**27.6 ± 1.1**	38.8 ± 18.3	41.3 ± 17.7
BCQ	58.5 ± 10.1	31.7 ± 2.4	**22.8 ± 0.4**	24.8 ± 4.2
CQL	45.8 ± 4.0	27.6 ± 1.3	22.8 ± 0.2	**22.7 ± 0.3**
TD3-BC	40.0 ± 16.1	26.1 ± 0.8	21.0 ± 0.7	**20.7 ± 0.8**

Table 3: **HookSweep2 Offline RL Results:** Average success percentage (± std. dev. over 3 seeds), where the average was computed over the last 50 training epochs. SAC and CQL (omitted) were unsuccessful with all datasets. We see that MOCODA was necessary for learning, and that results improve drastically with MOCODA-P, which re-balances MOCODA toward a uniform distribution in the box coordinates (see Figure 5).

RL Algorithm	**Average Success Rate** (higher is better)		
	EMP	**MOCODA**	**MOCODA-P**
BCQ	2.0 ± 1.6	20.7 ± 4.1	**64.7 ± 4.1**
TD3-BC	0.7 ± 0.9	38.7 ± 7.5	**84.0 ± 2.8**

and MOCODA-U augmented datasets greatly improve the average step count, providing support for **H2** and suggesting that using these datasets allows the agents to learn to traverse the diagonal of the state space, even though it is out-of-distribution with respect to EMP. This is consistent with a qualitative assessment of the learned policies, which confirms that agents trained on the ⌟-shaped EMP distribution learn a ⌟-shaped policy, whereas agents trained on MOCODA and MOCODA-U learn the optimal (diagonal) policy.

The results on the more complex HookSweep2 environment, shown in Table 3, provide further support for **H2**. On this environment, only results for BCQ and TD3-BC are shown, as the other algorithms failed on all datasets. For HookSweep2 we used a prioritized MOCODA-P parent distribution, as follows: knowing that the target task involves placing two blocks, we applied rejection sampling to MOCODA to make the marginal distribution of the joint block positions approximately uniform over its support. The effect is to have good representation in all areas of the most important state features for the target task (the block positions). The visualization in Figure 5 makes clear why training on MOCODA or MOCODA-P was necessary in order to solve this task: the base EMP distribution simply does not have sufficient coverage of the goal space.

6 Conclusion

In this paper, we tackled the challenging yet common setting where the available empirical data provides insufficient coverage of critical parts of the state space. Starting with the insight that locally factored transition models are capable of generalizing outside of the empirical distribution, we proposed MOCODA, a framework for augmenting available data using a controllable "parent distribution" and locally factored dynamics model. We find that adding augmented samples from MOCODA allows RL agents to learn policies that traverse states and actions never before seen in the experience buffer. Although our data augmentation is "model-based", the transition samples it produces are compatible with any downstream RL algorithm that consumes single-step transitions.

Future work might (1) explore methods for learning lo-

cally factorized representations, especially in environments with high-dimensional inputs (e.g., pixels) (Jiang et al., 2019; Kipf et al., 2020), and consider how MoCoDA might integrate with latent representations, (2) combine the insights presented here with learned predictors of out-of-distribution generalization (e.g., uncertainty-based prediction) (Pan et al., 2020), (3) create benchmark environments for entity-based RL (Winter et al., 2022) so that object-oriented methods and models can be better evaluated, and (4) explore different approaches to re-balancing the training distribution for learning on downstream tasks. With regards to direction (1), we note that asserting (or not) certain independence relationships may have fairness implications for datasets (Park et al., 2018; Creager et al., 2020) that should be kept in mind or explored. This is relevant also in regards to direction 4, as dataset re-balancing may result in (or fix) biases in the data (Krasanakis et al., 2018). Re-balancing schemes should be sensitive to this.

References

Alekh Agarwal, Nan Jiang, Sham M Kakade, and Wen Sun. Reinforcement learning: Theory and algorithms. *CS Dept., UW Seattle, Seattle, WA, USA, Tech. Rep*, 2019.

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, pages 104–114. PMLR, 2020.

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in neural information processing systems*, pages 5048–5058, 2017.

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Christopher M Bishop. Mixture density networks. 1994.

Guillaume Chaslot, Sander Bakkes, Istvan Szita, and Pieter Spronck. Monte-carlo tree search: A new framework for game ai. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 4, pages 216–217, 2008.

Elliot Creager, David Madras, Toniann Pitassi, and Richard Zemel. Causal modeling for fairness in dynamical systems. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 2185–2195. PMLR, 13–18 Jul 2020.

Alexander D'Amour, Hansa Srinivasan, James Atwood, Pallavi Baljekar, David Sculley, and Yoni Halpern. Fairness is not static: deeper understanding of long term fairness via simulation studies. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 525–534, 2020.

Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062. PMLR, 2019.

Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. In *International Conference on Learning Representations*, 2021.

Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored mdps. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

Lily Hu and Issa Kohler-Hausmann. What's sex got to do with fair machine learning? In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 2020.

Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, pages 12498–12509, 2019.

Edwin T Jaynes. Information theory and statistical mechanics. *Physical review*, 106(4):620, 1957.

Jindong Jiang, Sepehr Janghorbani, Gerard De Melo, and Sungjin Ahn. Scalor: Generative world models with scalable object representations. In *International Conference on Learning Representations*, 2019.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673. PMLR, 2018.

Michael Kearns and Daphne Koller. Efficient reinforcement learning in factored mdps. In *IJCAI*, volume 16, pages 740–747, 1999.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Thomas Kipf, Ethan Fetaya, Kuan-Chieh Wang, Max Welling, and Richard Zemel. Neural relational inference for interacting systems. In *International Conference on Machine Learning*, pages 2688–2697. PMLR, 2018.

Thomas Kipf, Elise van der Pol, and Max Welling. Contrastive learning of structured world models. In *International Conference on Learning Representations*, 2020.

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Emmanouil Krasanakis, Eleftherios Spyromitros-Xioufis, Symeon Papadopoulos, and Yiannis Kompatsiaris. Adaptive sensitive reweighting to mitigate bias in fairness-aware classification. In *Proceedings of the 2018 world wide web conference*, pages 853–862, 2018.

Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33:1179–1191, 2020.

Andrey Kurenkov, Ajay Mandlekar, Roberto Martin-Martin, Silvio Savarese, and Animesh Garg. Ac-teach: A bayesian actor-critic method for policy learning with an ensemble of suboptimal teachers. In *Conference on Robot Learning*, pages 717–734. PMLR, 2020.

Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. In *Advances in Neural Information Processing Systems*, 2020.

Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Geraud Nangue Tasse, Steven James, and Benjamin Rosman. A boolean task algebra for reinforcement learning. *Advances in Neural Information Processing Systems*, 33:9497–9507, 2020.

Ian Osband and Benjamin Van Roy. Near-optimal reinforcement learning in factored MDPs. *Advances in Neural Information Processing Systems*, 27, 2014.

Feiyang Pan, Jia He, Dandan Tu, and Qing He. Trust the model when it is confident: Masked model-based actor-critic. *Advances in neural information processing systems*, 33:10537–10546, 2020.

Ji Ho Park, Jamin Shin, and Pascale Fung. Reducing gender bias in abusive language detection. In *Conference on Empirical Methods in Natural Language Processing*, 2018.

Silviu Pitis, Harris Chan, and Stephen Zhao. mrl: modular rl. https://github.com/spitis/mrl, 2020a.

Silviu Pitis, Elliot Creager, and Animesh Garg. Counterfactual data augmentation using locally factored dynamics. *Advances in Neural Information Processing Systems*, 33:3976–3990, 2020b.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676): 354–359, 2017.

Shagun Sodhani, Sergey Levine, and Amy Zhang. Improving generalization with approximate factored value functions. In *ICLR2022 Workshop on the Elements of Reasoning: Objects, Structure and Causality*, 2022. URL https://openreview.net/forum?id=B4exBrOUceq.

Alexander L Strehl. Model-based reinforcement learning in factored-state mdps. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 103–110. IEEE, 2007.

Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Rishi Veerapaneni, John D Co-Reyes, Michael Chang, Michael Janner, Chelsea Finn, Jiajun Wu, Joshua Tenenbaum, and Sergey Levine. Entity abstraction in visual model-based reinforcement learning. In *Conference on Robot Learning*, pages 1439–1456. PMLR, 2020.

Clemens Winter, Huang Costa, Bamford Chris, and Matricon Theo. Entity gym. https://github.com/entity-neural-network/entity-gym, 2022.

A Proof of Theorem 1

The dynamics model assumed is a maximum-likelihood, count-based model that has separate parameters for each causal mechanism, $P^{\mathcal{L}}_{i,\theta}$, in each local neighborhood. That is, for a given configuration of the parents $\text{Pa}_i = x$ in $P^{\mathcal{L}}_{i,\theta}$, we define count parameter θ_{ij} for the j-th possible child, c_{ij}, so that $P^{\mathcal{L}}_{i,\theta}(c_{ij} \mid x) = \theta_j / \sum_{k=1}^{|c_i|} \theta_k$.

We use the following two lemmas (see source material for proof):

Lemma 1 (Proposition A.8 of Agarwal et al. (2019))**.** *Let z be a discrete random variable that takes values in $\{1, \ldots, d\}$, distributed according to q. We write q as a vector where $\vec{q} = [Pr(z = j)]_{j=1}^{d}$. Assume we have n i.i.d. samples, and that our empirical estimate of $\vec{q}$ is $[\hat{\vec{q}}]_j = \sum_{i=1}^{n} \mathbf{1}[z_i = j]/n$.*
We have that $\forall \epsilon > 0$:

$$Pr(\|\hat{q} - \vec{q}\|_2 \geq 1/\sqrt{n} + \epsilon) \leq e^{-n\epsilon^2}$$

which implies that:

$$Pr(\|\hat{q} - \vec{q}\|_1 \geq \sqrt{d}(1/\sqrt{n} + \epsilon)) \leq e^{-n\epsilon^2}$$

Lemma 2 (Corollary 1 of Strehl (2007))**.** *If for all states and actions, each model $P_{i,\theta}$ of P_i is ϵ/k close to the ground truth in terms of the ℓ_1 norm: $\|P_i(s,a) - P_{i,\theta}(s,a)\|_1 < \epsilon/k$, then the aggregate transition model P_θ is ϵ close to the ground truth transition model: $\|P(s,a) - P_\theta(s,a)\|_1 < \epsilon$.*

Theorem 1. *Let n be the number of empirical samples used to train the model of each local causal mechanism $P^{\mathcal{L}}_{i,\theta}$ at each configuration of parents $\text{Pa}_i = x$. There exists positive constant c such that, if*

$$n \geq \frac{ck^2|c_i|\log(|\mathcal{S}||\mathcal{A}|/\delta)}{\epsilon^2},$$

then, with probability at least $1 - \delta$, we have:

$$\max_{(s,a)} \|P(s,a) - P_\theta(s,a)\|_1 \leq \epsilon.$$

Proof. Applying Lemma 1, we have that for fixed parents $\text{Pa}_i = x$, wp. at least $1 - \delta$,

$$\|P_i(x) - P_{i,\theta}(x)\|_1 \leq c\sqrt{\frac{|c_i|\log(1/\delta)}{n}},$$

where n is the number of independent samples used to train $P_{i,\theta}$ and c is a positive constant. Now consider a fixed (s,a), consisting of k parent sets. Applying Lemma 2 we have that, wp. at least $1-\delta$,

$$\|P(s,a) - P_\theta(s,a)\|_1 \leq ck\sqrt{\frac{|c_i|\log(1/\delta)}{n}}.$$

We apply the union bound across all states and actions to get that wp. at least $1 - \delta$,

$$\max_{(s,a)} \|P(s,a) - P_\theta(s,a)\|_1 \leq ck\sqrt{\frac{|c_i|\log(|S||A|/\delta)}{n}}.$$

The result follows by rearranging for n and relabeling c. □

To compare to full-state dynamics modeling, we can translate the sample complexity from the per-parent count n to a total count N. Recall $m\Pi_i|c_i| = |\mathcal{S}|$, so that $|c_i| = (|\mathcal{S}|/m)^{1/k}$, and $m\Pi_i|\text{Pa}_i| \geq |\mathcal{S}||\mathcal{A}|$. We assume a small constant overlap factor $v \geq 1$, so that $|\text{Pa}_i| = v(|\mathcal{S}||\mathcal{A}|/m)^{1/k}$. We need the total number of component visits to be $n|\text{Pa}_i|km$, for a total of $nv(|\mathcal{S}||\mathcal{A}|/m)^{1/k}m$ state-action visits, assuming that parent set visits are allocated evenly, and noting that each state-action visit provides k parent set visits. This gives:

Corollary 1. *To bound the error as above, we need to have*

$$N \geq \frac{cmk^2(|\mathcal{S}|^2|\mathcal{A}|/m^2)^{1/k}\log(|\mathcal{S}||\mathcal{A}|/\delta)}{\epsilon^2},$$

total train samples, where we have absorbed the overlap factor v into constant c.

To extend this and adapt other results to our setting, we could now apply the Simulation Lemma (Agarwal et al., 2019) to bound the value difference given the model error, or alternatively, develop the theory in the direction of (Strehl, 2007) and related work. However, we believe the core insights are already contained in Theorem 1 and Corollary 1.

B Implementation Details

Implementation code will be made available upon acceptance. There are numerous components involved that each have several different settings that were mostly just taken "as-is" or picked as reasonable defaults (e.g., using a layer size of 512 in most neural networks, or having 5 components in the MDN, or the specific implementation of rejection sampling for `Mocoda-U`). The best documentation for specific details is the code itself. As such, the implementation details below cover the broad strokes so that a reader might understand the general pipeline, and we refer the reader to the provided code for precise details.

B.1 Causal Transition Structure and Parent Set Definitions

We implement the local causal model as a mask function M that maps (state, action) tuples to an adjacency matrix of the causal structure. For example, in `2d Navigation`, the mask function was implemented as follows:

```
def Mask2dNavigation(input_tensor):
  """
  accepts B x num_sa_features, and returns B x num_parents x num_children
  """

  # base local mask
  mask = torch.tensor(
    [[1, 0],
     [0, 1],
     [1, 0],
     [0, 1]]).to(input_tensor.device)

  # change local mask in top right quadrant
  mask = mask[None].repeat((input_tensor.shape[0], 1, 1))
  mask[torch.logical_and(input_tensor[:,0] > 0.5, input_tensor[:, 1] > 0.5)] = 1

  return mask
```

As an example, the causal graph for the base local mask, which applies for most of the state space is shown in the figure 7. We used the base local graph to select the parent sets, in this case, $(x, \Delta x)$ and $(y, \Delta y)$.

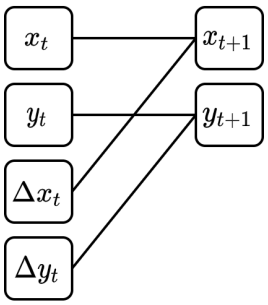


Figure 7: Causal graph for local mask

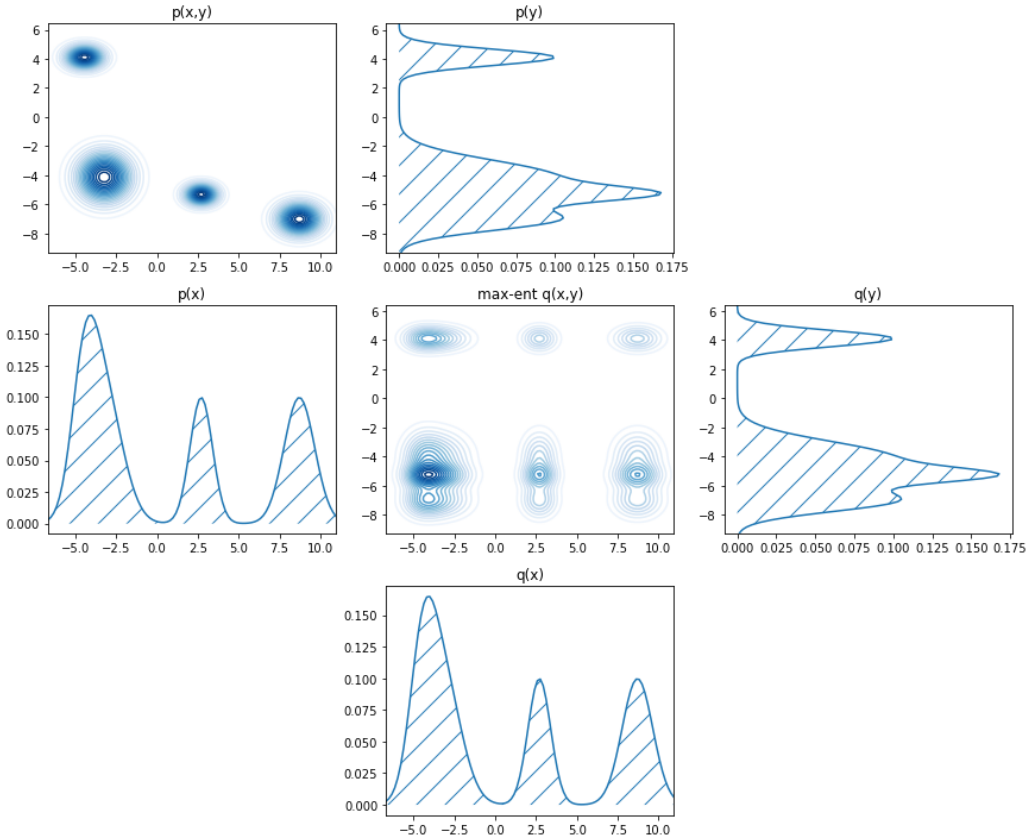


Figure 8: Hypothetical 2D illustration of the GMM-based parent set sampler. It is assumed that there are two non-overlapping parent sets $\{x\}$ and $\{y\}$, but that x and y exhibit dependence in the empirical data. We fit a GMM to each marginal $P(x)$ and $P(y)$ and sample from them independently to get $Q(x, y)$, which has the same marginal distributions (so that the components in a locally factored dynamics model will generalize), but eliminates the spurious dependence in the empirical data.

B.2 Parent Distribution

To sample the parent distribution in Step 1 of MoCoDA, we use the Gaussian Mixture Model (GMM) based approach described in the main text. The advantage of this approach is that we can easily do conditional sampling in case of overlapping parent sets. For a given local subset $\mathcal{L}$, we fit a separate GMM to the marginal of each parent set, as it appears in the empirical distribution for $\mathcal{L}$. To generate a new sample, we optionally shuffle the GMMs, and then sample from one GMM at a time, conditioning on any already generated features. This process eliminates any spurious correlations between features that are not part of the same parent set, and thus results in the maximum-entropy, marginal matching distribution.

In cases of multiple local neighborhoods, $\mathcal{L}_1, \mathcal{L}_2, \ldots$, one should respect the boundaries of the current local subset $\mathcal{L}$ during both training *and* generation. If a sample generated with the GMM for $\mathcal{L}$ falls outside of $\mathcal{L}$, that sample should be rejected, as the local causal structure is no longer valid, and the generalization guarantee for the locally factored model no longer holds.

As the local factorization in our experiments is quite simple, we did not stratify the GMM generator, and instead used a single GMM generator for the sparsest local causal structure. In the case of `2d Navigation` this did not generate any data that was out-of-distribution for the locally factored model components (as the agent's policy was consistent in all local neighborhoods). In the case of `HookSweep2`, there was a bit of locally out-of-distribution data in the local subspace in which there is a block collision; however, most of this data is unreachable as it involves overlapping blocks, and we obtained strong results even with this shortcut.

B.3 Dynamics Models

Our experiments used three different dynamics models. In each case, we used an ensemble of 5 base models, described below. The base models output a Gaussian mean and variance for each output variable and are trained independently via a

negative log likelihood loss. All models are trained using Adam Optimizer (Kingma and Ba, 2014).

A. **Unfactored:** The base model is a fully connected neural network with ReLU activations (MLP).

B. **Globally Factored:** The base model has one MLP for each causal mechanism in the sparsest local graph. For both `2d Navigation` and `HookSweep2` the sparest local graph has two components, so the base global model is composed of two MLPs.

C. **Locally factored:** The base model is designed as follows. For each child node, c_i, there is a separately parameterized single MLP that is preceded by a "Masked Composer" module. The Masked Composer applies a single layer MLP (linear transform followed by ReLU) to each root node, r_i (each parent set has several nodes), to obtain embeddings $\varepsilon_i(r_i)$. The i-th column of the mask is used to zero out the corresponding embeddings which are then summed, $\sum_i M_{ij}\varepsilon_i(r_i)$, and the result is passed as an input to the MLP.

   This architecture works (and enforces local factorization), but is likely poor, because it does not take advantage of potentially useful shared representations between parent nodes across children (since there is a separately parameterized Masked Composer for each child). A better architecture would likely use a single parameterization for a single, possibly deeper Masked Composer. As this is not the focus of our contribution, we stuck with simple model, as it "just worked" for purposes of our experiments.

B.4 Training Data for the RL Algorithm

This varied by experiment, and is described in the next Section. Notably, we divided the standard deviation returned by our dynamics models by a factor of three when generating data to avoid data that was too far out of distribution.

B.5 Reinforcement Learning Algorithms

We use Modular RL (Pitis et al., 2020a), adding three offline RL (Levine et al., 2020) algorithms: BCQ (Fujimoto et al., 2019), CQL (Kumar et al., 2020) & TD3-BC (Fujimoto and Gu, 2021).

The BCQ implementation uses DDPG (Lillicrap et al., 2016). For the generative model we use a Mixture Density Network (MDN) (Bishop, 1994) with 5 components, that produces 20 action samples at each call (both during test rollouts and when creating critic targets). The MDN was trained for 1000 batches with batch size of 2000. We did not use a perturbation model.

The CQL implementation uses SAC (Haarnoja et al., 2018). Rewards in our environments are sparse, and so value targets can be accurately clipped between two values (depends on the discount factor). CQL balances two losses: a penalty for Q-values of some non-behavioral distribution/policy (we use a random policy), and a bonus for the Q-values behavioral actions. We use an L1 penalty toward the lower end of the value target clipping range, and an L1 bonus toward the higher end of the value target clipping range. We then multiply that by a minimum Q coefficient, as in the original CQL implementation.

The TD3-BC implementation closely follows Fujimoto and Gu (2021).

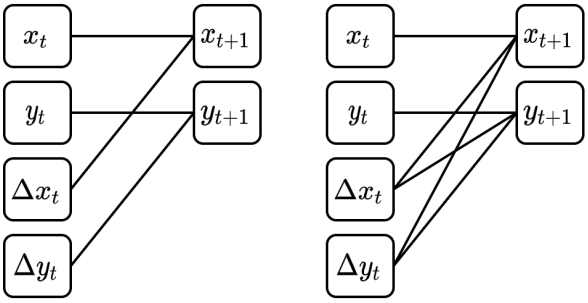
C Experimental Details

C.1 2D Navigation

In this environment, the agent must travel from one point in a square arena to another. States are 2D (x, y) coordinates and actions are 2D $(\Delta x, \Delta y)$ vectors.

```
observation_space = spaces.Box(np.zeros((2,)), np.ones((2,)), dtype=np.float32)
action_space = spaces.Box(-np.ones((2,)), np.ones((2,)), dtype=np.float32)
```

Episodes run for up to 70 steps. Rewards are sparse, with a -1 reward everywhere except the goal, where reward is 0. In most of the state space, the sub-actions Δx and Δy affect only their respective coordinate. In the top right quadrant, however, the Δx and Δy sub-actions each affect *both* x and y coordinates, so that the environment is locally factored. The two causal graphs are as follows:



The graph on the right applies only in the top-right quadrant; otherwise the graph on the left applies. The graph on the left has non-overlapping parent sets $(x, \Delta x)$ and $(y, \Delta y)$. The graph on the right has overlapping parent sets $(x, \Delta x, \Delta y)$ and $(y, \Delta x, \Delta y)$.

The agent has access to an empirical dataset consisting of left-to-right & bottom-to-top trajectories (20,000 transitions of each type):

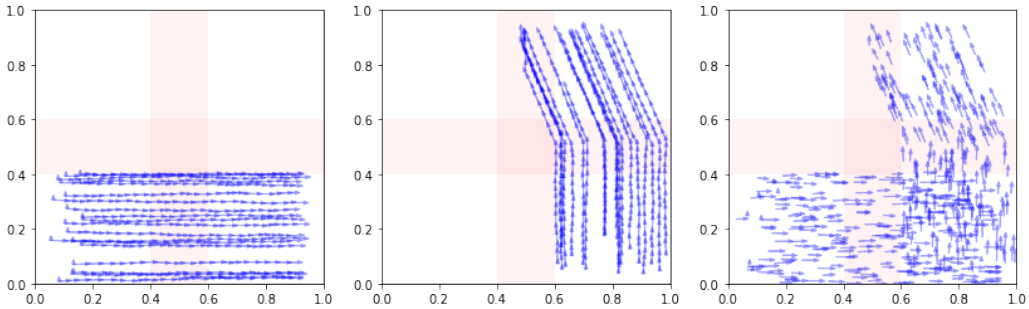


Figure 9: **Left/Middle:** Random samples of the two types of trajectories the agent has access to. **Right:** Random sample of transitions from this empirical dataset.

We consider a target task where the agent must move from the bottom left to the top right. In this task there is sufficient empirical data to solve the task by following the ⌟ shape of the data, but learning the optimal policy of going directly via the diagonal requires out-of-distribution generalization.

For `2d Navigation`, we generated the MOCODA distribution by fitting a GMM generator as described in the previous Section. Each GMM (one for each parent set) had 32 components, and was fit using expectation maximization. To obtain MOCODA-U, we implemented rejection sampling by using a KDE density estimator is as follows:

```
def prune_to_uniform(proposals, target_size=12000.):
  from sklearn.neighbors import KernelDensity
  sample = proposals[-10000:]

  fmap = lambda s: s[:, :2]
  K = KernelDensity(bandwidth=0.05)
  K.fit(fmap(sample))
  scores = K.score_samples(fmap(proposals))
  scores = np.maximum(scores, np.log(0.01))
  scores = (1. / np.exp(scores))
  scores = scores / scores.mean()  * (target_size / len(proposals))

  return proposals[np.random.uniform(size=scores.shape) < scores]
```

The dynamics models each had 2 layers of 256 neurons and were trained with a batch size of 512 and learning rate of 1e-4. Hyperparameters were not tuned once a working setting was found. Of the 40K empirical samples, 35K were used for training, and 5000 for validation. The models were trained for 600 epochs, with early stopping used in the last 50 epochs to find a locally optimal stopping point.

Augmented datasets of 200K samples were generated. In each case except EMP, 40K were the original empirical dataset

(thus 160K new samples were generated by applying the dynamics model to samples from the augmented distribution). In case of EMP, the 40K original samples were simply repeated 5 times to get a size 200K dataset. The locally factored network was used to generate the augmented datasets.

These augmented distributions were then used to train the downstream RL agents. The agent algorithms used a discount factor of 0.98, a target cutoff range of (-50, 0), batch size of 500, and used 2 layers of 512 neurons in both actor and critic networks. The agents were trained for 25K batches (for a total of 62.5 passes over the dataset).

For `2D Navigation` we ran 5 seeds, which all yielded similar results. For each seed we trained new parent set samplers and generated new augmented datasets.

C.2 HookSweep2

`HookSweep2` is a challenging robotics domain based on Hook-Sweep (Kurenkov et al., 2020), in which a Fetch robot must use a long hook to sweep two boxes to one side of the table (either toward or away from the agent). States, excluding the goal, are 16 dimensional continuous vectors. Goals are 6 dimensions. The agents all concatenate the goal to the state, and so operate on 22 dimensional states. The action space is a 4 dimensional continuous vector.

The environment contains two boxes that are initialized near the center of the table.

The empirical data contains 1M transitions from trajectories of an expert agent sweeping exactly one box to one side of the table, leaving the other in the center. The target task requires the agent to sweep *both* boxes together to one side of the table. This is particularly challenging because the setup is entirely offline (no exploration), where poor out-of-distribution generalization typically requires special offline RL algorithms that constrain the agent's policy to the empirical distribution (Levine et al., 2020; Agarwal et al., 2020; Kumar et al., 2020; Fujimoto and Gu, 2021).

Episodes run for 75 steps. Rewards are dense, but structured similarly to a sparse reward, with a base reward of -1 everywhere except the goal and a reward of 0 at the goal. Additional small rewards are given if the agent keeps the hook near the table (this was required to obtain natural movements from the trained expert agent).

In this environment, *we did not have the ground truth causal graph*, and so a heuristic was used. The heuristic (wrongly) assumes that the agent/hook *always* causes each of the next object position (hook and objects are always entangled), even though this is only true when the hook and the objects are touching. The heuristic considers the two boxes to be separate whenever they are further than 5cm from each other. Here is the implementation of the heuristic:

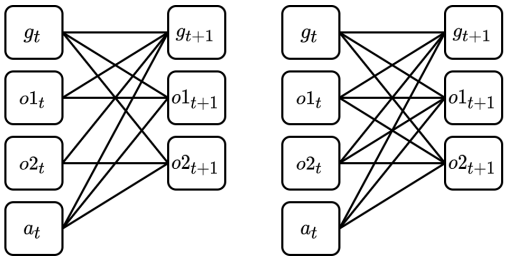
```
def MaskHookSweep2(input_tensor):

  # base local mask for when boxes are far apart
  mask = torch.tensor(
    [[1, 1, 1],
    [1, 1, 0],
    [1, 0, 1],
    [1, 1, 1]]
    ).to(input_tensor.device)
  mask = mask[None].repeat((input_tensor.shape[0], 1, 1))

  # change local mask when boxes are close to each other
  mask[torch.sum(torch.abs(input_tensor[:,O1X:O1X+2] -\
        input_tensor[:,O2X:O2X+2]), axis=1) < 0.05] = 1

  return mask
```

where the state-action components are (gripper, box1, box2, action). This heuristic returns the following two causal graphs (note that goals are not part of the dynamics, and are separately labeled using random goal samples from the environment):



The parent sets are (g, o1, a) and (g, o2, a) for the first graph, and (g, o1, o2, a) in the second graph.

For `HookSweep2`, the generation of the MoCoDA distribution is identical to how it was generated in `2d Navigation` (see previous subsection). To obtain MoCoDA-P, we implemented rejection sampling as follows:

```
def prune_to_uniform2(proposals, target_size=12000., smaller=True):
  proposals = proposals[np.linalg.norm(proposals[:,O1X:O1X+2] - proposals[:,O2X:O2X+2], axis=-1) < 0.3]
  sample = proposals[-5000:]

  fmap = lambda s: s[:,[O1X,O1X+1,O2X,O2X+1]]
  K = KernelDensity(bandwidth=0.05)
  K.fit(fmap(sample))
  scores = K.score_samples(fmap(proposals))
  scores = np.maximum(scores, np.log(0.05))
  scores = (1. / np.exp(scores))
  if np.minimum(scores, 1).sum() > 10000:
  while np.minimum(scores, 1).sum() > 10000:
    scores = scores * 0.99
  else:
  while np.minimum(scores, 1).sum() < 10000:
    scores = scores / 0.99

  return proposals[np.random.uniform(size=scores.shape) < scores]
```

The key difference to the `2d Navigation` is the definition of the `fmap` function, which defines the feature map under which the density is computed for rejection sampling.

The dynamics models for `HookSweep2` each had 2 layers of 512 neurons and were trained with a batch size of 512 and learning rate of 2e-4. Hyperparamters were not tuned once a working setting was found (learning rate was increased to make training slightly faster). Of 1M empirical samples, 5000 were used for validation. The models were trained for 4000 epochs, where each epoch involved 40K random samples, with early stopping used in the last 50 epochs to find a locally optimal stopping point.

Augmented datasets of 5M samples were generated. In each case except Emp, 1M were the original empirical dataset (thus 4M new samples were generated). In the case of Emp, the 1M original samplers were simply repeated 5 times to get the full augmented dataset. The locally factored network was used to generate the augmented datasets.

These augmented distributions were then used to train the downstream RL agents. The agent algorithms were the same as for `2d Navigation`, except that they used 3 layers of 512 neurons in both actor and critic networks. The agents were trained for 1M steps with batch size 500 (for a total of 100 passes over the dataset).

C.3 Licenses and Compute

All experiments were run on a modern desktop CPU and a NVIDIA GTX 1080 Ti GPU.

Code and assets are available under Apache and MIT licenses from Mujoco, OpenAI Gym, AC-Teach (Kurenkov et al., 2020), and Modular RL (Pitis et al., 2020a) repositories. The implementations used in this paper will be released upon acceptance under an open source license.

D Further Discussion of Broader Impacts

MoCoDA uses causally-motivated data augmentation to tackle sequential decision making problems where the available experience data may not be sufficient to find an optimal policy for the task at hand. While we have thusfar applied this approach to continuous control problems, there are a large body of problems that share this general motivation, where long-term fairness and robustness may be a central concern (D'Amour et al., 2020). In these cases, the causal assumptions used to implement MoCoDA deserve extra care and external scrutiny. For such problems, the structure of the state space may include sensitive and/or socially-ascribed attributes of groups and individuals (which cannot be directly intervened upon), so any graphical causal will involve normative assumptions about the environment in which the agent is embedded (Hu and Kohler-Hausmann, 2020).