In *Proceedings of the 38th Conference on Artificial Intelligence* (AAAI 2024),
Vancouver, BC, Canada February 2024

Building Minimal and Reusable Causal State Abstractions for Reinforcement Learning

Zizhao Wang[*1], **Caroline Wang**[*1], **Xuesu Xiao**[2], **Yuke Zhu**[1], **Peter Stone**[1, 3]

[1]The University of Texas at Austin, [2]George Mason University, [3]Sony AI
zizhao.wang@utexas.edu, caroline.l.wang@utexas.edu, xiao@gmu.edu, yukez@cs.utexas.edu, pstone@cs.utexas.edu

Abstract

Two desiderata of reinforcement learning (RL) algorithms are the ability to learn from relatively little experience and the ability to learn policies that generalize to a range of problem specifications. In factored state spaces, one approach towards achieving both goals is to learn state abstractions, which only keep the necessary variables for learning the tasks at hand. This paper introduces *Causal Bisimulation Modeling* (CBM), a method that learns the causal relationships in the dynamics and reward functions for each task to derive a minimal, task-specific abstraction. CBM leverages and improves implicit modeling to train a high-fidelity causal dynamics model that can be reused for all tasks in the same environment. Empirical validation on manipulation environments and Deepmind Control Suite reveals that CBM's learned implicit dynamics models identify the underlying causal relationships and state abstractions more accurately than explicit ones. Furthermore, the derived state abstractions allow a task learner to achieve near-oracle levels of sample efficiency and outperform baselines on all tasks.

Introduction

Reinforcement learning (RL) is a general paradigm that enables autonomous decision-making in unknown environments. A common deficiency of deep RL algorithms is their sample inefficiency and lack of generalization to unseen states, thus limiting their applicability in data-expensive or safety-critical tasks. One way to improve sample efficiency and generalization is to learn a *state abstraction* which reduces the task learning space and eliminates unnecessary information that may lead to spurious correlations. Determining the optimal abstraction necessitates understanding which state variables affect the reward and how those variables are influenced by others during state transitions.

Prior methods obtain the desired abstractions by searching for the smallest subset of state variables that can predict the reward accurately while ensuring the subset is self-predictable in dynamics. Yet, their *dense* dynamics models are specific to the subset and thus have to be learned from scratch for each new task (Fu et al. 2021; Wang et al. 2022a; Zhang et al. 2020b), as shown in Fig. 1. Such approaches overlook a key characteristic of realistic problems: we often wish to build agents that solve multiple instances of tasks in the same environment, e.g. different cooking skills in a kitchen. To learn multi-task dynamics models, recent works seek to learn *causal* dynamical dependencies between state variables, from

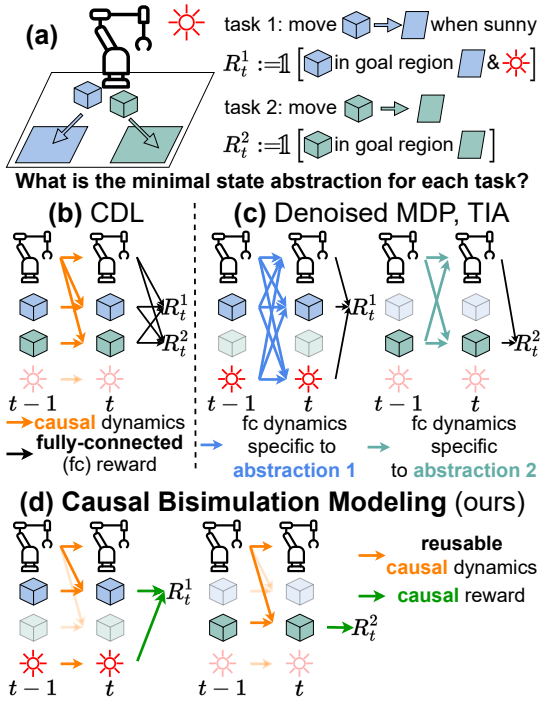

Figure 1: **(a)** Two tasks are defined by rewards R^1, R^2, and consist respectively of moving the blue and green blocks to their goal regions. Task 1 additionally requires moving the block only when it is sunny. Variables that are ignored by a state abstraction are semi-transparent. **(b)** CDL (Wang et al. 2022b) learns causal dependencies in the dynamics, but its derived state abstraction keeps *all* controllable state variables and ignores action-irrelevant ones, and thus the abstraction is non-minimal for task 2 and cannot learn task 1 due to its omission of the sun. **(c)** TIA and Denoised MDP (Fu et al. 2021; Wang et al. 2022a) can learn more concise abstractions (minimal in this example), but they require training fully-connected dynamics from scratch for each task. **(d)** In addition to the implicit *causal dynamics* that can be *reused* for all tasks, CBM identifies which variables affect the reward and derives a minimal state abstraction from the *causal reward* models.

[*]These authors contributed equally.

Copyright © 2024, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

which they derive a task-*independent* abstraction (Ding et al. 2022; Wang et al. 2021, 2022b). A notable example is Causal Dynamics Learning (CDL) (Wang et al. 2022b) which identifies all state variables that can be affected by the action and retains them in the abstraction.

While dynamics-based state abstractions only need to be learned once for the environment and can then be applied to any downstream task, we observe three weaknesses. First, the abstraction may not be minimal for a significant number of downstream tasks, since many tasks require manipulating only a subset of controllable variables. In such cases, CDL's abstraction could be further reduced to improve sample efficiency and generalization, as shown in Fig. 1(b). Second, ignoring state variables that are action-irrelevant limits CDL's application to many tasks. For example, for an autonomous vehicle, CDL's abstraction will ignore the traffic light, as the traffic light cannot be affected by the vehicle. In such cases, CDL's abstraction will cause the vehicle to be unable to follow traffic rules. Third, CDL employs explicit modeling of dynamics, directly predicting the next state as $\hat{s}_{t+1} = f(s_t, a_t)$ where f is a generic function parameterized by neural networks. However, prior works have shown *implicit modeling* ($\hat{s}_{t+1} = \arg\max_{s_{t+1} \in \mathcal{S}} g(s^i_{t+1}; s_t, a_t)$ where g is a critic function, see Sec.) generally achieves higher accuracy in model learning, particularly for non-smooth dynamics in real-world physical systems (Florence et al. 2022; Song and Kingma 2021). For instance, in robot manipulation, the object cannot be moved by the robot until they are in contact. In such environments, we show that inaccuracies of explicit modeling will lead to incorrect dynamical dependencies and thus non-minimal or incorrect state abstractions.

To address these weaknesses of CDL, we introduce Causal Bisimulation Modeling (CBM), a method that (1) learns **shared** task-agnostic dynamics between tasks while recovering a minimal, **task-specific** state abstraction, and (2) models causal dynamics dependencies with implicit models. Regarding the first contribution, in addition to dynamical relationships, CBM further infers which state variables affect the reward function with a causal reward model. In this way, CBM identifies state variables relevant to each task to further refine the state abstractions. The resultant causal abstraction is equivalent to bisimulation, a minimal abstraction that preserves the optimal value (Ferns, Panangaden, and Precup 2011). Regarding the second contribution, to the best of our knowledge, CBM is the first work that recovers causal dependencies with implicit models. To this end, CBM identifies and addresses two key problems of estimating conditional mutual information (CMI) with implicit models, allowing them to surpass explicit ones in both predictive accuracy and the identification of causal dependencies.

We validate CBM in robotic manipulation and Deepmind Control Suite, showing that (1) implicit models learn dynamical relationships and state abstractions more accurately compared to the explicit ones, and (2) CBM's task-specific state abstractions significantly improve sample efficiency and generalization compared to task-independent ones.

Related Work

Model-based State Abstractions for Decision Making Learned dynamics and reward models can be used in various ways for downstream task learning. Some methods directly use the learned models for planning (Williams et al. 2017; Chua et al. 2018; Nagabandi et al. 2018) or generate synthetic rollouts for reinforcement learning (Kurutach et al. 2018; Janner et al. 2019). Others use learned models to improve Q-value estimates (Feinberg et al. 2018; Amos et al. 2021), or generate state abstractions (Li, Walsh, and Littman 2006; Fu et al. 2021; Wang et al. 2022a, 2021; Zhang et al. 2019). This work belongs to the last class of methods.

The work closest to our CBM is Wang et al. (2022b) (CDL), which also learns a causal dynamics model and then derives a state abstraction for downstream task learning. As discussed in the introduction, CDL's abstraction is not minimal because it does not consider task information. In contrast, CBM considers causal reward relationships to derive a theoretically minimal, task-specific state abstraction.

Among model-based methods that learn task-specific state abstractions, the most closely related works are TIA (Fu et al. 2021), denoised MDP (Wang et al. 2022a), and ASR (Huang et al. 2022). Those methods learn dense, non-causal dynamics models from scratch for each task, which fails to take advantage of shared structures between the tasks. In contrast, CBM learns underlying causal dynamics that are shared between tasks in the same environment and applies the same dynamics model to all downstream tasks.

Implicit Models for Dynamics Learning Implicit models (Teh et al. 2003; Welling and Hinton 2002) have been widely used in many areas of machine learning, including image generation (Du and Mordatch 2019), natural language processing (Bakhtin et al. 2021; He et al. 2021), and density estimation (Saremi et al. 2018; Song et al. 2019). This is largely due to its ability to generalize probabilistic and deterministic approaches to classification, regression, and estimation (LeCun et al. 2006; Song and Kingma 2021).

Implicit modeling approaches have also been applied to reinforcement learning and the closely related problem of imitation learning, for modeling policies (Florence et al. 2022), value functions (Haarnoja et al. 2017, 2018), and dynamics (Pfrommer, Halm, and Posa 2020; Wang, Lu, and Zhao 2020). Multiple works have noted that implicit approaches are better able to model discontinuous surfaces, which is particularly advantageous for modeling the discontinuous contact dynamics common in robotics (Pfrommer, Halm, and Posa 2020; Florence et al. 2022). For such dynamics, though explicit models theoretically can capture such discontinuities through activation functions, empirically, they linearly interpolate between discontinuity boundaries when training data are finite, as found by Florence et al. (2022).

Background

We formulate our problem with Markov Decision Processes and adopt key concepts from CDL (Wang et al. 2022b).

Factored Markov Decision Processes We model K tasks in the same environment as a set of factored Markov decision processes (MDPs), $\mathcal{M}^k = (\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}^k)$. These MDPs have

the same (1) finite (bounded) state space, consisting of $d_{\mathcal{S}}$ state variables (factors) denoted as $\mathcal{S} = \mathcal{S}^1 \times \cdots \times \mathcal{S}^{d_{\mathcal{S}}}$, where each variable $\mathcal{S}^i$ is a scalar, (2) $d_{\mathcal{A}}$-dimensional action space, denoted as $\mathcal{A} \subseteq \mathbb{R}^{d_{\mathcal{A}}}$, and (3) transition probability $\mathcal{T}(s_{t+1}|s_t, a_t)$ (i.e., dynamics). However, each MDP has its own reward function $\mathcal{R}^k : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

Similar to prior model-based state abstraction learning methods, the goal of CBM is to learn the dynamics and reward functions from data, and to derive a task-specific abstraction for task learning. Following CDL, CBM assumes that the transitions of each state variable S^i are independent, i.e., $\mathcal{T}$ can be decomposed as $\mathcal{T}(s_{t+1}|s_t, a_t) = \prod_{i=1}^{d_{\mathcal{S}}} p(s^i_{t+1}|s_t, a_t)$. Though the assumption does not hold for all variables—for instance, quaternion variables representing object rotations in the manipulation environments—in practice, we find that our method can still learn dynamics accurately in such environments. For simplicity, we use x_t to denote all state variables and the action at t, i.e., $\boldsymbol{x_t} = \{\boldsymbol{s^1_t}, \cdots, \boldsymbol{s^{d_{\mathcal{S}}}_t}, \boldsymbol{a_t}\}$ and $x_t^{\neg i}$ to denote all those variables except for s^i_t, i.e., $\boldsymbol{x_t^{\neg i}} = \boldsymbol{x_t} \setminus \{\boldsymbol{s^i_t}\}$.

Causal Dynamics Learning (CDL) Instead of using a dense model, CDL models the dynamics as a causal graphical model (Pearl 2009) and recovers the necessary dependencies between each state variable pair $(\mathcal{S}^i_t, \mathcal{S}^j_{t+1})$ as well as $(\mathcal{A}_t, \mathcal{S}^j_{t+1})$ using causal discovery methods (Mastakouri, Schölkopf, and Janzing 2021). Then, aiming at improving sample efficiency during task learning, CDL derives a task-independent state abstraction by keeping (1) **controllable** state variables, i.e., those that can be changed by the action directly or indirectly, and (2) **action-relevant** state variables, i.e., those that cannot be changed by the action but can affect the action's influence on controllable variables. However, this abstraction keeps all controllable variables, while many tasks only need the agent to control one or a few of them, suggesting some variables in CDL's abstraction may be redundant.

Causal Bisimulation Modeling (CBM)

This section describes two main contributions of CBM: obtaining a task-specific state abstraction by augmenting causal dynamics models with causal reward modeling, and recovering accurate causal dynamics when using implicit models.

Causal Reward Model for Task-Specific Abstraction

Though CDL recovers the causal relationships in dynamics, it still uses a dense model for reward learning. Consequently, without knowing which state variables are causal parents of the reward (i.e., reward-relevant), there is no direct way to remove irrelevant variables to improve sample efficiency.

To resolve these issues, CBM learns a causal reward model following a similar strategy to what CDL uses to learn dynamics. Assuming that all state variables affect the reward for task $\mathcal{R}^k_t$ independently (for notational simplicity, we omit the environment index k for the reminder of the paper) and there are no dependencies between state variables at the same timestep, CBM examines the causal relationship between the state variable $\mathcal{S}^j_t$ and the reward $\mathcal{R}$ by learning two predictive models for the reward: (1) $p(r_t|x_t)$, which uses all variables for prediction, and (2) $p(r_t|x_t^{\neg j})$, which ignores s^j_t when predicting. Intuitively, if the prediction performance when ignoring s^j_t is significantly lower than when including it, then the causal dependency $\mathcal{S}^j_t \to \mathcal{R}$ exists. More precisely, CBM evaluates whether the Conditional Mutual Information (CMI) is larger than a predefined threshold, i.e., $\text{CMI}^{jk} := \mathop{\mathbb{E}}_{s_t, a_t, r_t}\left[\log \frac{p(r_t|x_t)}{p(r_t|x_t^{\neg j})}\right] \geq \epsilon$.

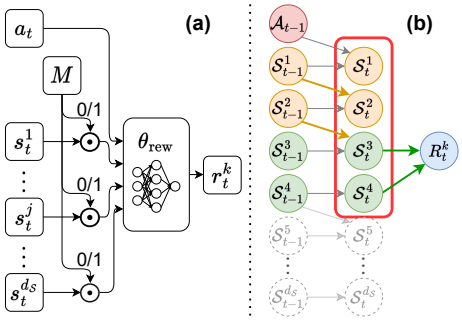

Figure 2: **(a)** The reward predictor architecture which can represent $p(r^k_t|a_t, M \odot s_t)$ conditioning any subsets of inputs by setting the binary mask M. **(b)** After CBM identifies the causal dependencies in dynamics and reward, its minimal state abstraction (marked by the red box) consists of (1) green variables that affect the reward, and (2) orange variables that influence green ones via dynamics. The semi-transparent state variables are ignored by the abstraction.

As shown in Fig. 2 (a), to make the method scalable, rather than training $p(r_t|x_t^{\neg j})$ for each $j \in \{1, \ldots, d_{\mathcal{S}}\}$, CBM combines all predictive models into one network $p_{\theta_{\text{rew}}}(r_t|M \odot x_t)$; where θ_{rew} is the network parameters and M is a manually defined binary mask used to ignore some input variables when predicting r_t. During training, CBM maximizes the prediction likelihood of both $p_{\theta_{\text{rew}}}(r_t|x_t)$ where M uses all inputs (i.e., all entries are set to 1), and $p_{\theta_{\text{rew}}}(r_t|x_t^{\neg j})$ where the entry for s^j_t is set to zero in M.

After recovering causal parents of $\mathcal{R}$, CBM derives a bisimulation by combining the causal reward model with the causal dynamics model, following the theorem below (see the Appendix for an explanation of how our setting satisfies the theorem's assumptions):

Theorem 1 (Connecting Bisimulation to Causal Feature Set (Thm 1 in Zhang et al. (2020a))). Consider an MDP $\mathcal{M}$ that satisfies Assumptions 1-3 in Zhang et al. (2020a). Let $\mathbf{P}_{\mathcal{R}} \subseteq 1, ..., d_{\mathcal{S}}$ be the set of variables such that the reward $\mathcal{R}(s, a)$ is a function only of $s^{\mathbf{P}_{\mathcal{R}}}$ (s restricted to the indices in $\mathbf{P}_{\mathcal{R}}$). Let $\mathbf{A}_{\mathcal{R}}$ denote the ancestors of $\mathbf{P}_{\mathcal{R}}$ in the causal graph corresponding to the transition dynamics of $\mathcal{M}$. Then the state abstraction $\phi(s) = s^{\mathbf{A}_{\mathcal{R}}}$ is a bisimulation abstraction for reward $\mathcal{R}$.

As illustrated in Fig. 2 (b), CBM's abstraction is selected as the union of (1) all $\mathcal{S}^j$ that $\mathcal{R}$ depends on, i.e., $\mathcal{S}^{\mathbf{P}_{\mathcal{R}}}$, and (2) all other state variables that can affect $\mathcal{S}^{\mathbf{P}_{\mathcal{R}}}$ via dynamics and not already included in $\mathcal{S}^{\mathbf{P}_{\mathcal{R}}}$. In other words, this union corresponds to $\mathcal{R}$'s causal ancestors (i.e., $\mathcal{S}^{\mathbf{A}_{\mathcal{R}}}$) in the learned causal dynamics and reward graph, and thus being equivalent to bisimulation — the *minimal state abstraction* that preserves the optimal values (Dean and Givan 1997; Ferns, Panangaden, and Precup 2011).

Causal Discovery with Implicit Dynamics Models

Implicit models have been shown to learn dynamics more accurately than explicit models (Florence et al. 2022). Motivated by this finding, the goal of CBM's dynamics learning module is to recover causal relationships between states and action variables via an *implicit* modeling approach. As in CBM's reward-learning approach, causal dependencies will also be detected by measuring the conditional mutual information CMI^{ij} between s_t^j and s_{t+1}^i for each i, j pair. In this section, first, we introduce the implicit dynamics model. Second, we describe how CBM estimates CMI from the implicit dynamics with a prior method by Sordoni et al. (2021). Third, we discuss two CMI overestimation issues of the prior method and how CBM solves them.

Implicit Dynamics Models For the transition of each state variable in $\mathcal{S}^i$, we would like an implicit model $g^i(s_{t+1}^i; x_t)$ such that $g^i : \mathcal{S}^i \times \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ to assign a high score to the label s_{t+1}^i from the ground truth distribution, and low values to other labels. During dynamics prediction, the model selects the s_{t+1} that maximizes the total score over all state variables: $\hat{s}_{t+1}^i = \arg\max_{s_{t+1}^i \in \mathcal{S}^i} \sum_{i=1}^{d_\mathcal{S}} g^i(s_{t+1}^i; x_t)$. For notational simplicity, in the remainder of Sec. , we omit the state variable index i on g as it is clear from the input variable s_{t+1}^i. An implicit dynamics model can be trained by minimizing the contrastive InfoNCE loss,

$$\mathcal{L}_{\text{NCE}}(g) = -\log \frac{e^{g(s_{t+1}^i; x_t)}}{e^{g(s_{t+1}^i; x_t)} + \sum_{n=1}^{N} e^{g(\tilde{s}_{t+1}^{i,n}; x_t)}}. \quad (1)$$

Minimizing the InfoNCE loss requires the ground truth label s_{t+1}^i, as well as N negative examples $\{\tilde{s}_{t+1}^{i,n}\}_{n=1}^N \sim p(s_{t+1}^i)$ sampled the value range S_{t+1}^i. This loss encourages g to distinguish the label s_{t+1}^i from negative samples, i.e., extract information about s_{t+1}^i from x_t. The remainder of the paper uses $\mathcal{L}_{\text{NCE}}(F(x; y))$ to denote the InfoNCE loss that encourages a generic model F to extract information about x from y.

CMI Estimation with Implicit Dynamics Models

Observation 1 Oord, Li, and Vinyals (2018) show that, for a generic model $F(x, y)$ that minimizes $\mathcal{L}_{\text{NCE}}$, the minimized $\mathcal{L}_{\text{NCE}}$ approximates the mutual information between x and y, i.e., $\mathbb{E}\left[\log(N+1) - \mathcal{L}_{\text{NCE}}(F(x; y))\right] \approx \mathbb{E}\left[\frac{p(y|x)}{p(y)}\right] = I(x; y)$. Inspired by this, Sordoni et al. (2021) propose to estimate CMI^{ij} using a conditional implicit model:

$$\text{CMI}^{ij} = \mathbb{E}\left[\log \frac{(N+1)e^{\phi(s_{t+1}^i; s_t^j | x_t^{\neg j})}}{e^{\phi(s_{t+1}^i; s_t^j | x_t^{\neg j})} + \sum_{n=1}^{N} e^{\phi(\tilde{s}_{t+1}^{i,n}; s_t^j | x_t^{\neg j})}}\right],$$
$$\text{where } \tilde{s}_{t+1}^{i,n} \sim p(s_{t+1}^i | x_t^{\neg j}). \quad (2)$$

Since CMI^{ij} measures how using s_t^j could additionally contribute to predicting s_{t+1}^i given the other state and action variables, $x_t^{\neg j}$, ϕ is a **conditioned** model trained to capture the additional information about s_{t+1}^i in s_t^j that is not present in $x_t^{\neg j}$. However, training ϕ and estimating CMI^{ij} with Eq. (2) both require negative samples from $p(s_{t+1}^i | x_t^{\neg j})$, which are not readily accessible, as the data can only be collected from the full state transition distribution $\mathcal{T}(s_{t+1} | x_t)$.

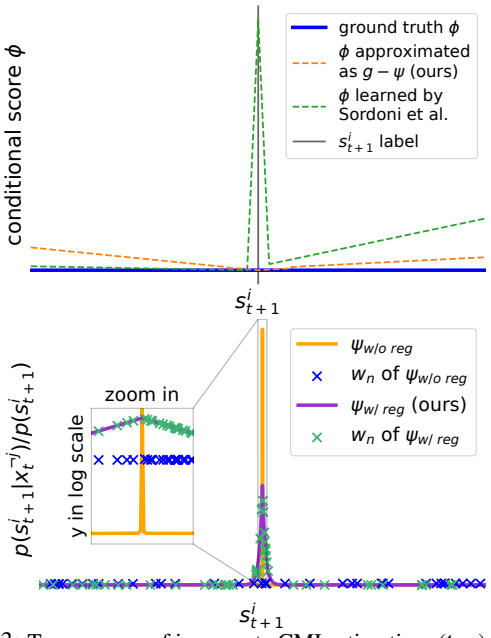

Figure 3: Two sources of inaccurate CMI estimation: (**top**) Overfitted conditional model: learned ϕ overestimate conditional information while the $g - \psi$ approximation is closer to the ground truth (0 for all s_{t+1}^i values), especially when close to the ground truth label $s_{t+1}^i = 0$. (**bottom**) Inaccurate Importance Sampling: without regularization, the likelihood ratio of $p(s_{t+1}^i | x_t^{\neg j}) / p(s_{t+1}^i)$ computed by ψ can be peaked at the label $s_{t+1}^i = 0$ and is therefore challenging to approximate by self-normalized importance sampling. In comparison, regularized ψ has a flatter likelihood ratio landscape and is easier to approximate with samples.

Observation 2 To tackle this issue, CBM uses the importance sampling approximation proposed by Sordoni et al. (2021) to compute $p(s_{t+1}^i | x_t^{\neg j})$ with samples from the marginal distribution $p(s_{t+1}^i)$. Thus, CMI^{ij} may be approximated as,

$$\mathbb{E}\left[\log \frac{(N+1)e^{\phi(s_{t+1}^i; s_t^j | x_t^{\neg j})}}{e^{\phi(s_{t+1}^i; s_t^j | x_t^{\neg j})} + N\sum_{n=1}^{N} w_n e^{\phi(\tilde{s}_{t+1}^{i,n}; s_t^j | x_t^{\neg j})}}\right],$$
$$w_n = \frac{e^{\psi(\tilde{s}_{t+1}^{i,n}; x_t^{\neg j})}}{\sum_{k=1}^{N} e^{\psi(\tilde{s}_{t+1}^{i,k}; x_t^{\neg j})}} \approx \frac{p(s_{t+1}^i | x_t^{\neg j})}{p(s_{t+1}^i)}. \quad (3)$$

where $\tilde{s}_{t+1}^{i,n} \sim p(s_{t+1}^i)$, and ψ is trained to extract information about s_{t+1}^i from $x_t^{\neg j}$ by minimizing $\mathcal{L}_{\text{NCE}}(\psi(s_{t+1}^i; x_t^{\neg j}))$ and is used to compute importance weights w_n in a self-normalized manner.

After learning ψ^*, one only needs ϕ to estimate CMI with Eq. (3). To this end, Sordoni et al. (2021) train ϕ by minimizing $\mathcal{L}_{\text{NCE}}(\phi(s_{t+1}^i; s_t^j | x_t^{\neg j}) + \psi^*(s_{t+1}^i; x_t^{\neg j}))$ while keeping ψ^* frozen, so ϕ learns to capture the *additional* information about s_{t+1}^i from s_t^j that is not present in $x_t^{\neg j}$ (i.e., absent in ψ^*). See pseudo-code in the Appendix for details.

Inaccurate CMI Estimation and Solutions In practice, the method proposed by Sordoni et al. (2021) often yields inaccurate CMI estimations and thus leads to incorrect causal dependencies. We discovered two reasons for the inaccuracy and proposed corresponding solutions.

Reason 1 – Overfitted Conditional Models In theory, when ϕ is trained as described above, it should condition on ψ and capture the *additional* information only. However, in practice, such trained ϕ still uses $x_t^{\neg j}$ to predict s_{t+1}^i directly rather than conditioning on it to estimate the additional contribution of s_t^j. Fig. 3 top shows an example where knowing s_t^j does not provide any additional information about s_{t+1}^i. In such a case, the ground truth conditional model should output the same score for all s_{t+1}^i values. In contrast, the scores output by ϕ are still peaked at the label of s_{t+1}^i, and using such an overfitted model in Eq. (3) would overestimate CMI.

To solve this issue, rather than using a learned ϕ, we use the approximation $\phi = g - \psi$. The motivation is as follows: as g is trained to use x_t to estimate the score of s_{t+1}^i and ψ estimates with all variables except for s_t^j, the difference of their estimated scores should reflect the additional information from s_t^j. In practice, as shown in Fig. 3 top, the conditional score estimated by this approximation is closer to the ground truth than the score of ϕ learned by Sordoni et al. (2021), especially in the neighbor of the ground truth label where the accuracy of conditional scores significantly influences CMI.

Reason 2 – Inaccurate Importance Sampling Meanwhile, when s_{t+1}^i has an almost deterministic transition (which is common in many environments, e.g., objects will not move unless manipulated by the robot), the importance sampling approximation in Eq. (3) could be inaccurate.

In detail, as shown in Fig. 3 bottom, for such transitions, the score estimated by ψ tends to have extremely sharp maxima — it is high only when s_{t+1}^i is very close to the ground truth labels. As a result, even with many negative samples from $p(s_{t+1}^i)$, it is likely that none of them are similar enough to samples from $p(s_{t+1}^i|x_t^{\neg j})$. Then, since the importance weight w_n in Eq. (3) is self-normalized among all negative samples, samples that are not from $p(s_{t+1}^i|x_t^{\neg j})$ still have large weights (rather than near-zero weights), thus leading to inaccurate CMI estimation.

To mitigate this issue, when training g and ψ for the dynamics of $\mathcal{S}_{t+1}^i$, beyond the InfoNCE loss, we regularize them to have flatter score landscapes with L2 penalty on their computed scores and the partial derivative of scores as follows,

$$\mathcal{L}_{\text{dyn}} = \mathcal{L}_{\text{reg}}(g) + \mathcal{L}_{\text{reg}}(\psi) \text{ where for generic } f, \tag{4}$$

$$\mathcal{L}_{\text{reg}}(f) = \mathcal{L}_{\text{NCE}}(f(s_{t+1}^i;\cdot)) + \sum_{\tilde{s}_{t+1}^i} \left(\lambda_1 \left\| f(\tilde{s}_{t+1}^i;\cdot) \right\|_2^2 + \lambda_2 \left\| \frac{\partial f(\tilde{s}_{t+1}^i;\cdot)}{\partial \tilde{s}_{t+1}^i} \right\|_2^2 \right).$$

The regularization is applied to both the label and all negative samples (i.e., $\tilde{s}_{t+1}^i \in \{s_{t+1}^i, \tilde{s}_{t+1}^{i,n}\}$), and λ_1, λ_2 are the weights of the regularization terms. As shown in Fig. 3 bottom, with regularization, ψ is flatter. As a result, with the same number of samples, the importance weights w_n approximate the likelihood ratio computed by ψ much better, compared to approximating sharp ψ without regularization.

To make the dynamics model scalable, we use the same masking technique as in Sec. to combine g and ψ for each j into one network. We use θ_{dyn} to denote the parameters of $d_{\mathcal{S}}$ such networks, each modeling the dynamics of state variable $\mathcal{S}_{t+1}^i$.

CBM for Task Learning

Algorithm 1 Causal Bisimulation Modeling (CBM)

1: Initialize the dynamics model θ_{dyn}.
2: (Optional) Pretrain θ_{dyn} (Eq. 4) with offline data.
3: **for** K tasks **do**
4: Initialize the reward model θ_{rew} and the policy π.
5: **for** T training steps **do**
6: Collect (s_t, a_t, r_t, s_{t+1}) with $a_t \sim \pi$.
7: Update θ_{dyn} (Eq. 4, optional) and θ_{rew} (Sec.).
8: Evaluate dynamical and reward dependencies (Eq. 3); Update the state abstraction for π (Fig. 2 (b)).
9: Update π SAC losses.

As shown in Alg. 1, for tasks with the same dynamics, CBM's dynamics model is shared across tasks. The dynamics model can either learn from offline data (line 2), or from transitions collected during task learning (line 7), or both.

When solving each task, CBM interweaves reward learning (line 7) with policy learning. The policy is trained via Soft Actor Critic (SAC, Haarnoja et al. (2018)), an off-policy reinforcement learning algorithm. The reward model is combined with the pre-trained dynamics to generate the task-specific abstraction (line 8). CBM applies the state abstraction to the policy π as a binary mask that zeros out ignored variables. During task learning, as the policy explores and learns, we expect it to gradually expose causal relationships that are necessary to solve the task, and, in return, the updated state abstractions reduce the learning space of the policy, making its learning sample efficient.

Experiments

We examine the following hypotheses. First, implicit models recover dynamical dependencies more accurately than explicit models (Sec.). Second, compared to CDL's task-independent state abstraction and prior task-specific abstraction works, CBM learns a more concise abstraction and improves sample efficiency and generalization of task learning over baselines (Sec.).

Environments To test CBM, we use two manipulation environments implemented with Robosuite (Zhu et al. 2020), shown in Fig. 5 left, and two tasks from the DeepMind Control Suite (DMC, Tunyasuvunakool et al. (2020)). In the block environment (b), there are multiple movable and unmovable blocks. The tasks in this environment include Pick and Stack. In the tool-use environment, we consider a challenging long-horizon task Series: the agent needs to use an L-shaped tool

	block			tool-use	
	causal graph	pick	stack	causal graph	series
explicit	87.5 ± 0.1	53.2 ± 4.6	59.6 ± 4.6	82.6 ± 0.2	80.0 ± 1.5
implicit (ours)	**90.5** ± 0.4	**95.7** ± 6.0	**95.7** ± 6.0	**85.5** ± 0.1	**98.8** ± 1.3

Table 1: Mean $\pm$ std. error of accuracy ($\uparrow$) for learned dynamics causal graphs and task abstractions.

to move a faraway block within reach, pick it up, and place it within the box. In the DMC, we consider the Cheetah and Walker tasks, two high-dimensional continuous control tasks. In all environments, as controllable distractors (cd), 20 variables whose values are random projections of the action (i.e., $W^T a_t$ where $W \subseteq \mathbb{R}^{d_\mathcal{A}}$ are randomly sampled) are added to the state space. We also add 20 uncontrollable distractors (ud) whose values are uniformly sampled from $[-1, 1]$. The distractors have no interaction with other state variables.

Baselines For the dynamics learning experiments, we compare the implicit dynamics model against the explicit model. All methods are trained and evaluated with 3 random seeds. For the state abstraction and task learning experiments, we compare CBM against the closely related methods of CDL, which uses a task-independent abstraction; TIA (Fu et al. 2021) and Denoised MDP (Wang et al. 2022a), which both learn task-specific state abstractions. To contextualize these methods, we also compare against an Oracle that learns with the ground-truth minimal abstractions, and reinforcement learning over the Full state space (no state abstraction). All methods are trained and evaluated with 5 random seeds. Further details are in the Appendix.

State Abstractions for Task Learning To fairly compare the effects of the various state abstractions for task learning, all methods use **implicit** dynamics models, including CDL, which originally used the inferior explicit models. All methods learn tasks via Soft Actor Critic (Haarnoja et al. 2018). The dynamics model is pretrained in Pick and Stack tasks, and it is learned jointly with the policy *from scratch* in all other tasks. Further details are in the Appendix.

Dynamics and Causal Graph Learning

This section compares implicit and explicit dynamics models, learned from the same offline data, in terms of how accurately they recover dynamics causal graphs and task abstractions. Experiments are conducted on the Robosuite environments[1], and results are shown in Table 1. For causal graphs, we compare the learned dynamics dependencies with the ground truth and measure the *causal graph accuracy* as the # correctly learned edges / total edges in the graph. The accuracy of implicit models is 3% higher than that of explicit ones, which corresponds to causal relationships between 67 pairs of state variables in the block environment, and 48 pairs in tool-use. As a result, when being used to derive a *task-specific* abstraction (we use the ground truth reward dependencies in this experiment only to avoid the interference from reward learning), implicit models derive more accurate state abstractions ($\geq 20\%$ difference in accuracy on all tasks) than explicit models. Note that the *state abstraction accuracy* is measured as the # of correctly classified state variables / total state variables. Implicit models also learn more generalizable dynamics (see Appendix).

[1]DMC tasks are not suitable for these experiments, as the ground truth causal graphs are not clear.

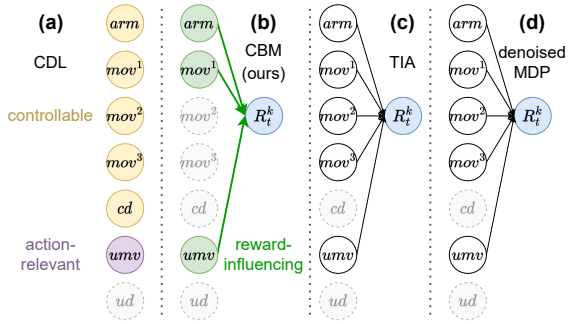

Figure 4: Object-level state abstractions for Stack learned by each method; semitransparent variables are excluded. **(a)** Though mov^2, mov^3, and controllable distractos cd are unnecessary, CDL still keeps them as they are controllable. **(b)** Compared to CDL, CBM (ours) successfully learns the minimal abstraction by further reasoning which state variables influence the reward. **(c)** TIA and **(d)** Denoised MDP fail to learn meaningful abstractions when their assumptions on the dynamics do not hold.

Task Learning with State Abstractions

This section compares policy learning with different state abstractions on various tasks in the DMC, the block environment (b), and the tool-use environment (t).

State Abstraction Accuracy Fig. 4 shows the abstraction learned by each method for the Stack task. For simplicity, abstractions are shown on the object level; state variable-level abstractions are in the Appendix. Among all methods, only CBM learns minimal abstraction. CDL keeps all controllable variables and thus uses a non-minimal abstraction. Meanwhile, TIA and Denoised MDP assume that state variables can be segregated into several dynamically independent components, and their abstractions keep only the task-relevant component. However, though mov^2 and mov^3 are task-irrelevant, their dynamics still depend on the task-relevant part (the end-effector and gripper). As a result, when their assumptions do not hold, TIA and Denoised MDP learn that (almost) all variables belong to the same component. Then, depending on their definitions of task relevance, all such variables are either included (TIA) or ignored (Denoised MDP) by the abstractions.

CBM is Sample-Efficient The performance metric for Pick and Stack tasks is the mean success rate at accomplishing the task, and the metric for Series and DMC tasks is the mean episode reward, evaluated over 50 test episodes and

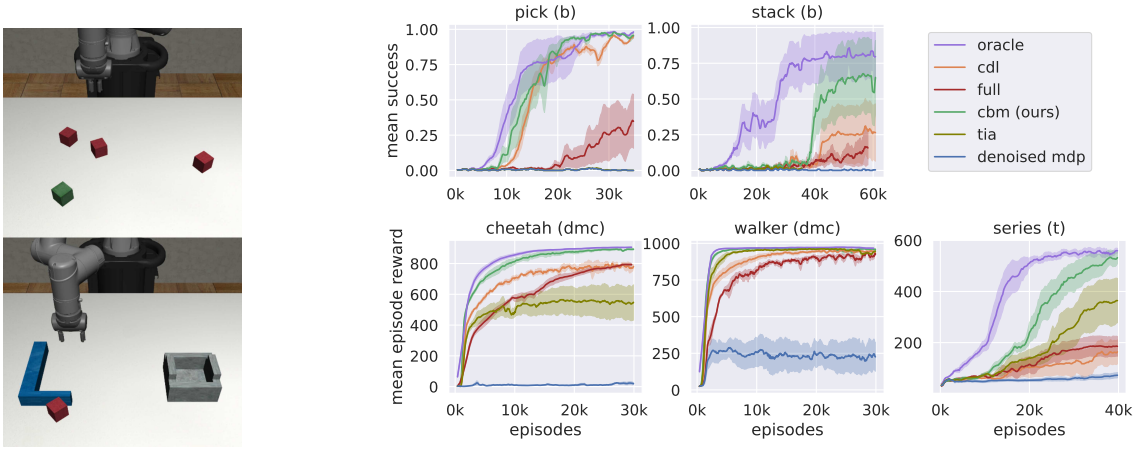

Figure 5: (**left**) top: Block environment with three *movable* blocks ($mov^{1\sim3}$), and an *unmovable* (unm) block fixed on the table. bottom: Tool-Use environment with the block, the L-shaped tool, and the box. (**right**) Learning curves of CBM (ours) compared to baseline methods and RL with the Oracle abstraction in five tasks. Each learning curve is generated from independent runs using 5 different random seeds, with mean and std. error computed across 50 test episodes per point on the learning curve. CBM is among the most sample-efficient methods, even approaching the efficiency of the Oracle on Pick, Cheetah, and Walker.

plotted with respect to the number of episodes.[2] The learning curves for all tasks are shown in Fig. 5 right. Recall that Oracle learns with the ground-truth state abstraction, whereas Full learns with no state abstraction. For all tasks, Oracle learns the fastest, demonstrating the possible gain in sample efficiency with an ideal state abstraction.

Overall, we find that CBM matches or improves in sample efficiency over the CDL, TIA, and Denoised MDP baselines in all settings. In all tasks, CBM is among the closest to the Oracle in sample efficiency, showing the benefit of the learned, near-minimal state abstraction. We observe that the higher the difficulty level of the task, the greater the benefit of learning a small task abstraction. For instance, Walker is a simple task, where almost all methods converge rapidly to an episode reward of 1000 by 20k episodes. The only exception is Denoised MDP, where the learned abstraction masks out some key variables among robot joint angles and angular velocities. On the other hand, the Series task is much more challenging, requiring a sequence of successful behaviors (reach tool, use tool to move block closer, pick and place block). Learning without any abstraction (Full) only learns to grasp the tool and achieves an episode reward of 200 over 40k training episodes, while CBM learns with much greater sample efficiency. We observe a similarly large gap between CBM and other methods on Stack (B), another complex manipulation task. An ablation of CBM with explicit dynamics models can be found in the Appendix; we find that the ablation has much worse sample efficiency than CBM, demonstrating the benefit of using implicit dynamics models.

CBM is Generalizable As shown in Fig. 6, we further measure each method's generalizability to unseen states on Pick and Stack (TIA and Denoised MDP fail to learn the tasks and thus are not evaluated). In addition to in-distribution (ID) states, we also evaluate the learned policies

[2]As each episode is a fixed number of steps, episodes have a linear relationship with training steps.

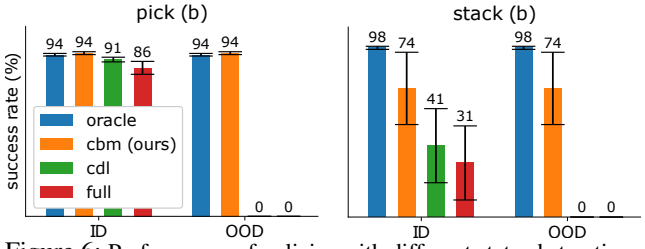

Figure 6: Performance of policies with different state abstractions on both ID and OOD states, in terms of mean and std. error of success rates (↑) in the block environment.

on out-of-distribution (OOD) states where the values of all task-irrelevant state variables are set to noise sampled from $\mathcal{N}(0, 1)$. For both tasks, only Oracle and CBM keep similar performance across ID and OOD states, as their minimal task-specific abstractions eliminate the influence of OOD variables. In contrast, though CDL can be robust against variables ignored by its abstraction, it still fails to generalize when redundant variables in its non-minimal abstractions have unseen values.

Conclusion

This paper studies how to generate task-specific minimal state abstractions for task learning. It introduces Causal Bisimulation Modeling (CBM), an algorithm that (1) learns a minimal state abstraction via causal reward learning, and (2) learns an implicit causal dynamics model. The experiments demonstrate that CBM learns more accurate and concise state abstractions, which lead to improved sample efficiency on downstream tasks compared to related methods. Further, the implicit dynamics model introduced by CBM improves over explicit dynamics models in terms of prediction error and causal graph accuracy. Promising directions for future work include relaxing the assumption of having a pre-defined factored state space to extend CBM to high-dimensional state spaces, such as images.

Acknowledgements

This work has taken place in the Learning Agents Research Group (LARG) at UT Austin. LARG research is supported in part by NSF (FAIN-2019844, NRT-2125858), ONR (N00014-18-2243), ARO (E2061621), Bosch, Lockheed Martin, and UT Austin's Good Systems grand challenge. Peter Stone serves as the Executive Director of Sony AI America and receives financial compensation for this work. The terms of this arrangement have been reviewed and approved by the University of Texas at Austin in accordance with its policy on objectivity in research.

References

Amos, B.; Stanton, S.; Yarats, D.; and Wilson, A. G. 2021. On the model-based stochastic value gradient for continuous reinforcement learning. In *Learning for Dynamics and Control*, 6–20. PMLR.

Bakhtin, A.; Deng, Y.; Gross, S.; Ott, M.; Ranzato, M.; and Szlam, A. 2021. Residual Energy-Based Models for Text. *Journal of Machine Learning Research*, 22(40): 1–41.

Chua, K.; Calandra, R.; McAllister, R.; and Levine, S. 2018. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *arXiv preprint arXiv:1805.12114*.

Dean, T.; and Givan, R. 1997. Model minimization in Markov decision processes. In *AAAI/IAAI*, 106–111.

Ding, W.; Lin, H.; Li, B.; and Zhao, D. 2022. Generalizing Goal-Conditioned Reinforcement Learning with Variational Causal Reasoning. *arXiv preprint arXiv:2207.09081*.

Du, Y.; and Mordatch, I. 2019. Implicit Generation and Modeling with Energy Based Models. In Wallach, H.; Larochelle, H.; Beygelzimer, A.; d'Alché-Buc, F.; Fox, E.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Feinberg, V.; Wan, A.; Stoica, I.; Jordan, M. I.; Gonzalez, J. E.; and Levine, S. 2018. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*.

Ferns, N.; Panangaden, P.; and Precup, D. 2011. Bisimulation metrics for continuous Markov decision processes. *SIAM Journal on Computing*, 40(6): 1662–1714.

Florence, P.; Lynch, C.; Zeng, A.; Ramirez, O. A.; Wahid, A.; Downs, L.; Wong, A.; Lee, J.; Mordatch, I.; and Tompson, J. 2022. Implicit behavioral cloning. In *Conference on Robot Learning*, 158–168. PMLR.

Fu, X.; Yang, G.; Agrawal, P.; and Jaakkola, T. 2021. Learning task informed abstractions. In *International Conference on Machine Learning*, 3480–3491. PMLR.

Haarnoja, T.; Tang, H.; Abbeel, P.; and Levine, S. 2017. Reinforcement Learning with Deep Energy-Based Policies. In *International Conference on Machine Learning*.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *International Conference on Machine Learning*.

He, T.; McCann, B.; Xiong, C.; and Hosseini-Asl, E. 2021. Joint Energy-based Model Training for Better Calibrated Natural Language Understanding Models. *ArXiv*, abs/2101.06829.

Huang, B.; Lu, C.; Leqi, L.; Hernández-Lobato, J. M.; Glymour, C.; Schölkopf, B.; and Zhang, K. 2022. Action-sufficient state representation learning for control with structural constraints. In *International Conference on Machine Learning*, 9260–9279. PMLR.

Jang, E.; Gu, S.; and Poole, B. 2016. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*.

Janner, M.; Fu, J.; Zhang, M.; and Levine, S. 2019. When to Trust Your Model: Model-Based Policy Optimization. In *Advances in Neural Information Processing Systems*.

Kurutach, T.; Clavera, I.; Duan, Y.; Tamar, A.; and Abbeel, P. 2018. Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*.

LeCun, Y.; Chopra, S.; Hadsell, R.; Ranzato, A.; and Huang, F. J. 2006. A Tutorial on Energy-Based Learning. *Predicting structured data*, 1.

Li, L.; Walsh, T. J.; and Littman, M. L. 2006. Towards a unified theory of state abstraction for MDPs. In *AI&M*.

Mastakouri, A. A.; Schölkopf, B.; and Janzing, D. 2021. Necessary and sufficient conditions for causal feature selection in time series with latent common causes. In *International Conference on Machine Learning*, 7502–7511. PMLR.

Nagabandi, A.; Kahn, G.; Fearing, R. S.; and Levine, S. 2018. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 7559–7566. IEEE.

Nikishin, E.; Schwarzer, M.; D'Oro, P.; Bacon, P.-L.; and Courville, A. 2022. The Primacy Bias in Deep Reinforcement Learning. In Chaudhuri, K.; Jegelka, S.; Song, L.; Szepesvari, C.; Niu, G.; and Sabato, S., eds., *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, 16828–16847. PMLR.

Oord, A. v. d.; Li, Y.; and Vinyals, O. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*.

Pearl, J. 2009. *Causality*. Cambridge university press.

Pfrommer, S.; Halm, M.; and Posa, M. 2020. ContactNets: Learning of Discontinuous Contact Dynamics with Smooth, Implicit Representations. In *Conference on Robot Learning*.

Saremi, S.; Mehrjou, A.; Schölkopf, B.; and Hyvärinen, A. 2018. Deep Energy Estimator Networks. *ArXiv*, abs/1805.08306.

Song, Y.; Garg, S.; Shi, J.; and Ermon, S. 2019. Sliced Score Matching: A Scalable Approach to Density and Score Estimation. In *Conference on Uncertainty in Artificial Intelligence*.

Song, Y.; and Kingma, D. P. 2021. How to Train Your Energy-Based Models. *ArXiv*, abs/2101.03288.

Sordoni, A.; Dziri, N.; Schulz, H.; Gordon, G.; Bachman, P.; and Des Combes, R. T. 2021. Decomposed mutual information estimation for contrastive representation learning. In *International Conference on Machine Learning*, 9859–9869. PMLR.

Teh, Y. W.; Welling, M.; Osindero, S.; and Hinton, G. E. 2003. Energy-Based Models for Sparse Overcomplete Representations. *Journal of Machine Learning Research*, 4: 1235–1260.

Tunyasuvunakool, S.; Muldal, A.; Doron, Y.; Liu, S.; Bohez, S.; Merel, J.; Erez, T.; Lillicrap, T.; Heess, N.; and Tassa, Y. 2020. dm control: Software and tasks for continuous control. *Software Impacts*, 6: 100022.

Wang, J.; Lu, Y.; and Zhao, H. 2020. CLOUD: Contrastive Learning of Unsupervised Dynamics. In *CoRL*.

Wang, T.; Du, S.; Torralba, A.; Isola, P.; Zhang, A.; and Tian, Y. 2022a. Denoised MDPs: Learning World Models Better Than the World Itself. In Chaudhuri, K.; Jegelka, S.; Song, L.; Szepesvari, C.; Niu, G.; and Sabato, S., eds., *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, 22591–22612. PMLR.

Wang, Z.; Xiao, X.; Zhu, Y.; and Stone, P. 2021. Task-Independent Causal State Abstraction. In *Proceedings of the 35th International Conference on Neural Information Processing Systems, Robot Learning workshop*.

Wang, Z.; Xiao, X.; Zhu, Y.; and Stone, P. 2022b. Causal Dynamics Learning for Task-Independent State Abstraction. In *Proceedings of the 39th International Conference on Machine Learning*.

Welling, M.; and Hinton, G. E. 2002. A New Learning Algorithm for Mean Field Boltzmann Machines. In *International Conference on Artificial Neural Networks*.

Weng, J.; Chen, H.; Yan, D.; You, K.; Duburcq, A.; Zhang, M.; Su, Y.; Su, H.; and Zhu, J. 2022. Tianshou: A Highly Modularized Deep Reinforcement Learning Library. *Journal of Machine Learning Research*, 23(267): 1–6.

Williams, G.; Wagener, N.; Goldfain, B.; Drews, P.; Rehg, J. M.; Boots, B.; and Theodorou, E. A. 2017. Information theoretic MPC for model-based reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 1714–1721. IEEE.

Zhang, A.; Lipton, Z. C.; Pineda, L.; Azizzadenesheli, K.; Anandkumar, A.; Itti, L.; Pineau, J.; and Furlanello, T. 2019. Learning causal state representations of partially observable environments. *arXiv preprint arXiv:1906.10437*.

Zhang, A.; Lyle, C.; Sodhani, S.; Filos, A.; Kwiatkowska, M.; Pineau, J.; Gal, Y.; and Precup, D. 2020a. Invariant causal prediction for block mdps. In *International Conference on Machine Learning*, 11214–11224. PMLR.

Zhang, A.; McAllister, R.; Calandra, R.; Gal, Y.; and Levine, S. 2020b. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*.

Zhu, Y.; Wong, J.; Mandlekar, A.; and Martín-Martín, R. 2020. robosuite: A Modular Simulation Framework and Benchmark for Robot Learning. In *arXiv preprint arXiv:2009.12293*.

Appendix

Pseudo-code for Sec

In this section, we provide pseudo-code for how Sordoni et al. (2021) and CBM learn the conditional implicit model ϕ, respectively, in Alg 2 and Alg 3. For easier reference, first, we reproduce key equations and notations as follows:

InfoNCE Loss For a generic model $f : \mathcal{Y} \times \mathcal{Z} \to \mathbb{R}$ that tries to extract information about a generic variable $y \in \mathcal{Y}$ from another variable $z \in \mathcal{Z}$. The corresponding InfoNCE loss of f is

$$\mathcal{L}_{\text{NCE}}(f(y;z)) = -\log \frac{e^{f(y;z)}}{e^{f(y;z)} + \sum_{n=1}^{N} e^{f(\tilde{y}^n;z)}}, \text{ where negative samples are drawn from } \tilde{y}^n \sim \mathcal{Y}.$$

Involved Implicit Models In Sec , when Sordoni et al. (2021) and CBM estimate conditional mutual information CMI^{ij} between s_t^j and s_{t+1}^i respectively, the following three implicit models are involved:

- $g(s_{t+1}^i; x_t) : \mathcal{S}^i \times \mathcal{X} \to \mathbb{R}$, extracting information about s_{t+1}^i from $x_t = (s_t, a_t)$, where $\mathcal{X} = \mathcal{S} \times \mathcal{A}$.
- $\psi(s_{t+1}^i; x_t^{\neg j}) : \mathcal{S}^i \times \mathcal{X}^{\neg j} \to \mathbb{R}$, extracting information about s_{t+1}^i from $x_t^{\neg j} = (s_t^1, \ldots, s_t^{j-1}, s_t^{j+1}, \ldots, s_t^{d_{\mathcal{S}}}, a_t)$, where $\mathcal{X}^{\neg j} = \mathcal{S}^1 \times \cdots \times \mathcal{S}_t^{j-1} \times \mathcal{S}_t^{j+1} \times \cdots \times \mathcal{S}_t^{d_{\mathcal{S}}} \times \mathcal{A}$.
- $\phi(s_{t+1}^i; s_t^j | x_t^{\neg j}) : \mathcal{S}^i \times \mathcal{S}^j \times \mathcal{X}^{\neg j} \to \mathbb{R}$, extracting the **additional** information about s_{t+1}^i in s_t^j that is not present in $x_t^{\neg j}$.

Algorithm 2 ϕ learned by Sordoni et al. (2021)

Initialize the ψ and ϕ.
repeat
  Sample s_{t+1}^i and $x_t^{\neg j}$ from data; Sample $\{\tilde{s}_{t+1}^{i,n}\}_{n=1}^N$ from $\mathcal{S}^i$.
  Optimize ψ with $\mathcal{L}_{\text{NCE}}(\psi(s_{t+1}^i; x_t^{\neg j}))$.
until ψ converges to ψ^*
repeat
  Sample s_{t+1}^i, s_t^j, $x_t^{\neg j}$ from data; Sample $\{\tilde{s}_{t+1}^{i,n}\}_{n=1}^N$ from $\mathcal{S}^i$.
  Optimize ϕ with the following loss while **keeping** ψ^* **frozen**

$$\mathcal{L}_{\text{NCE}}(\phi + \psi^*) = -\log \frac{e^{\phi(s_{t+1}^i; s_t^j | x_t^{\neg j}) + \psi^*(s_{t+1}^i; x_t^{\neg j})}}{e^{\phi(s_{t+1}^i; s_t^j | x_t^{\neg j}) + \psi^*(s_{t+1}^i; x_t^{\neg j})} + \sum_{n=1}^{N} e^{\phi(\tilde{s}_{t+1}^{i,n}; s_t^j | x_t^{\neg j}) + \psi^*(\tilde{s}_{t+1}^{i,n}; x_t^{\neg j})}}. \tag{5}$$

until ϕ converges

Algorithm 3 ϕ learned by CBM

Initialize the g and ψ.
repeat
  Sample s_{t+1}^i and x_t from data; Sample $\{\tilde{s}_{t+1}^{i,n}\}_{n=1}^N$ from $\mathcal{S}^i$.
  Optimize g with

$$\mathcal{L}_{\textbf{dyn}}(g(s_{t+1}^i; x_t)) = \mathcal{L}_{\text{NCE}}(g) + \sum_{\tilde{s}_{t+1}^i \in \{s_{t+1}^i\} \cup \{\tilde{s}_{t+1}^{i,n}\}_{n=1}^N} \left(\lambda_1 \left\| g(\tilde{s}_{t+1}^i; x_t) \right\|_2^2 + \lambda_2 \left\| \frac{\partial g(\tilde{s}_{t+1}^i; x_t)}{\partial \tilde{s}_{t+1}^i} \right\|_2^2 \right).$$

until g converges
repeat
  Sample s_{t+1}^i and $x_t^{\neg j}$ from data; Sample $\{\tilde{s}_{t+1}^{i,n}\}_{n=1}^N$ from $\mathcal{S}^i$.
  Optimize ψ with

$$\mathcal{L}_{\textbf{dyn}}(\psi(s_{t+1}^i; x_t^{\neg j})) = \mathcal{L}_{\text{NCE}}(\psi) + \sum_{\tilde{s}_{t+1}^i \in \{s_{t+1}^i\} \cup \{\tilde{s}_{t+1}^{i,n}\}_{n=1}^N} \left(\lambda_1 \left\| \psi(\tilde{s}_{t+1}^i; x_t^{\neg j}) \right\|_2^2 + \lambda_2 \left\| \frac{\partial \psi(\tilde{s}_{t+1}^i; x_t^{\neg j})}{\partial \tilde{s}_{t+1}^i} \right\|_2^2 \right).$$

until ψ converges
return $\phi(s_{t+1}^i; s_t^j | x_t^{\neg j}) = g(s_{t+1}^i; x_t) - \psi(s_{t+1}^i; x_t^{\neg j})$

After learning ϕ, CBM computes CMI^{ij} as in Alg. 4, following Sordoni et al. (2021).

Algorithm 4 CMI^{ij} computation using learned ψ and ϕ

1: Sample $\{\tilde{s}_{t+1}^{i,n}\}_{n=1}^{N}$ from $\mathcal{S}^i$.
2: **for** $n \in N$ **do**
3: Compute the self-normalized importance weight for each negative sample $\tilde{s}_{t+1}^{i,n}$ as

$$w_n = \frac{e^{\psi(\tilde{s}_{t+1}^{i,n}; x_t^{\neg j})}}{\sum_{m=1}^{N} e^{\psi(\tilde{s}_{t+1}^{i,m}; x_t^{\neg j})}} \approx \frac{p(s_{t+1}^i | x_t^{\neg j})}{p(s_{t+1}^i)}.$$

4: Compute the conditional mutual information as

$$\text{CMI}^{ij} = \mathbb{E}\left[\log \frac{(N+1)e^{\phi(s_{t+1}^i; s_t^j | x_t^{\neg j})}}{e^{\phi(s_{t+1}^i; s_t^j | x_t^{\neg j})} + N\sum_{n=1}^{N} w_n e^{\phi(\tilde{s}_{t+1}^{i,n}; s_t^j | x_t^{\neg j})}}\right].$$

Additional Comparison with Related Works

Regarding the assumptions of the method and the quality of learned abstractions, below we provide a table comparing CBM against ICP (Zhang et al. 2020a), CDL, TIA, and denoised MDPs in the following aspects:

- whether the method can learn in a single environment (SE)
- whether the method can learn minimal state abstractions (MSA)
- whether the learned dynamics can be shared across multiple tasks in the same environment (GD)
- whether the method can learn from high-dimensional image observations (IO), instead of assuming that the state space is factored

	SE	MSA	GD	IO
CBM (ours)	✓	✓	✓	×
ICP	×	×	×	✓
CDL	✓	×	✓	×
TIA and denoised MDP	✓	×	×	✓

Overall, as described above, our method can learn minimal state abstractions, does not require multiple environments for training, and can share the dynamics across different tasks, but at the cost of assuming a factored state space.

Theorem 1 Assumptions

Our MDP also follows the three assumptions of Theorem 1 in Zhang et al., (2020a). Specifically, (1) For Assumption 1 that each observation corresponds to a unique state, we assume the observation space is the state space and is fully observable. (2) For Assumption 2 that each observation component (i.e., each state variable in our setting) at $t+1$ is independent given observation at t, we have the same assumption in Sec 3.1. (3) For Assumption 3 about the difference between environments in the same family, we only focus on a single environment and thus there is no need for this assumption.

Dynamics Learning Implementation and Further Results

Implicit Dynamics Modelling Details

We compute the energy $E/\psi(s_{t+1}^i; M \odot [s_t, a_t])$ as $g(M \odot [s_t, a_t])^T h(s_{t+1}^i)$, where both g and h are multilayer perceptrons with three hidden layers of 128 units and outputs a size 128 vector. During training, we use regularization coefficients $\lambda_1 = \lambda_2 = 10^{-6}$. λ_1 and λ_2 are decided using grid search among $\{10^{-3}, 10^{-4}, \cdots, 10^{-7}\}$, by trading off prediction accuracy and causal graph accuracy. The CMI threshold used to infer causal relationships is $\epsilon = 0.02$. During inference, to predict the next step value of each state variable $\hat{s}_{t+1}^i$ as $\arg\max_{s_{t+1}^i} f(s_t, a_t, s_{t+1}^i)$, we uniformly sample 8,192 samples from $\mathcal{S}^i$ and select the one with the lowest energy. We test the effector of the sample number in Table. 3.

The architecture and hyperparameters of the implicit dynamics model are listed in Table 2. We tested networks with 64/128/256 neurons in each layer and chose 128 as it achieves the same prediction performance as 256 with a shorter computation time. Similarly, we choose the number of negative samples as 512 from $\{256, 512, 1024\}$. Other hyperparameters are not tuned. In our experiments, instead of predicting the energy based on $(M \odot [s_t, a_t])$ and s_{t+1}^i, we use $(M \odot [s_t, a_t])$ and Δs_t^i where Δs_t^i is the change of the variable at t. Then, for negative samples, we sample them from $[\Delta s_{\min}^i, \Delta s_{\max}^i]$.

Table 2: Hyperparameters for the implicit dynamics model.

Name	Value
feature architecture	[128, 128]
energy architecture	[128]
activation functions	ReLU
number of training transitions	2M
training step	3M
number of negative samples	512
learning rate	3e-4
batch size	32
prediction step during training, H	3

Dynamics Data Collection

CBM proposes a novel method to evaluate the causal dependencies with implicit models, recovering more accurate dependencies than explicit models. However, in addition to the causal discovery method, the quality of the recovered causal relationship also depends on the collected data. In this section, we explain how we ensure the data is collected by diverse policies to enable correct inference of causal dependencies.

If the dynamics model is trained on offline data only, the data should be collected by a diverse set of policies to break the spurious correlations. We may not have such offline data. For example, offline data are collected by a single policy only. In this case, we can augment the data with online data collected by the set $\{\pi_n^k\}_{k=1,\dots,K;t=1,\dots,T}$, where k in the task index and t is the training step. This set of policies will naturally be diverse, because:

- Considering k, each π_t^k learns from a different reward and behaves uniquely.
- Considering t, as π_t^k explores and gets updated, its behavior also changes and thus the data is being collected by different policies.
- Though Alg. 1 shows that we learn tasks sequentially for simplicity of presentation, in practice, we can learn all tasks simultaneously so that the dynamics model can use data collected by all policies for learning.

Learned Dynamics Causal Graph

In addition to the causal graph accuracy discussed in Table. 1. We also show the dynamics causal graphs learned by implicit and explicit dynamics for the block environment, in Fig. 7 and Fig. 8, respectively. The strength of the causal dependencies are measured in CMI and numbered in each cell. Meanwhile, the missing dependencies are marked in red while spurious ones are marked in green. We notice that implicit dynamics models tend to miss dependencies that are necessary but happen infrequently, while explicit models tend to depend on more spurious correlations and thus generalize badly in out-of-distribution states.

Dynamics Prediction

We also evaluate CBM using implicit dynamics and CDL in terms of prediction accuracy. Specifically, given s_t and $a_{t:t+19}$, we use them to generate 20-step predictions, i.e., $s_{t+1:t+20}$, on both in-distribution (ID) and out-of-distribution (OOD) s_t for the block and tool-use environments. For OOD states, distractor values are replaced with random values sampled from $\mathcal{N}(0, 100)$. The results are again measured on 10K transitions for each method.

As shown in Fig. 9, when measuring the prediction error of all state variables, CBM has lower prediction error than CDL on both ID and OOD states. Especially on OOD states, as CBM learns fewer spurious correlations than CDL, it keeps similar performance while CDL's errors increase significantly compared to ID states.

Dynamics Computation Cost

When predicting each single state variable, though our method draw 8192 samples and compute their energies, we want to point out that the computation cost is still comparable to explicit dynamics models and does not prevent the implicit dynamics from scaling to environments with large state spaces.

Requiring a large number of samples is an inherent weakness of implicit models. Nevertheless, this weakness has not prevented applying a similar model to robotic tasks (Florence et al., 2022) and to higher-dimensional states such as images (Chen et al., 2020b).

Meanwhile, the computational complexity is $O(d_S^2)$ for explicit dynamics and $O(d_S \times (d_S + N))$ for implicit dynamics (where d_S is the number of state variables and N is the number of samples). So for higher d_S, the complexity ratio of implicit over explicit dynamics ($= 1 + \frac{N}{d_S}$) is actually lower.

Moreover, for small d_S where the complexity ratio is high, one can trade off between computation and prediction accuracy. We show the mean and standard deviation of prediction performance (measured as one-step prediction error on all state variables)

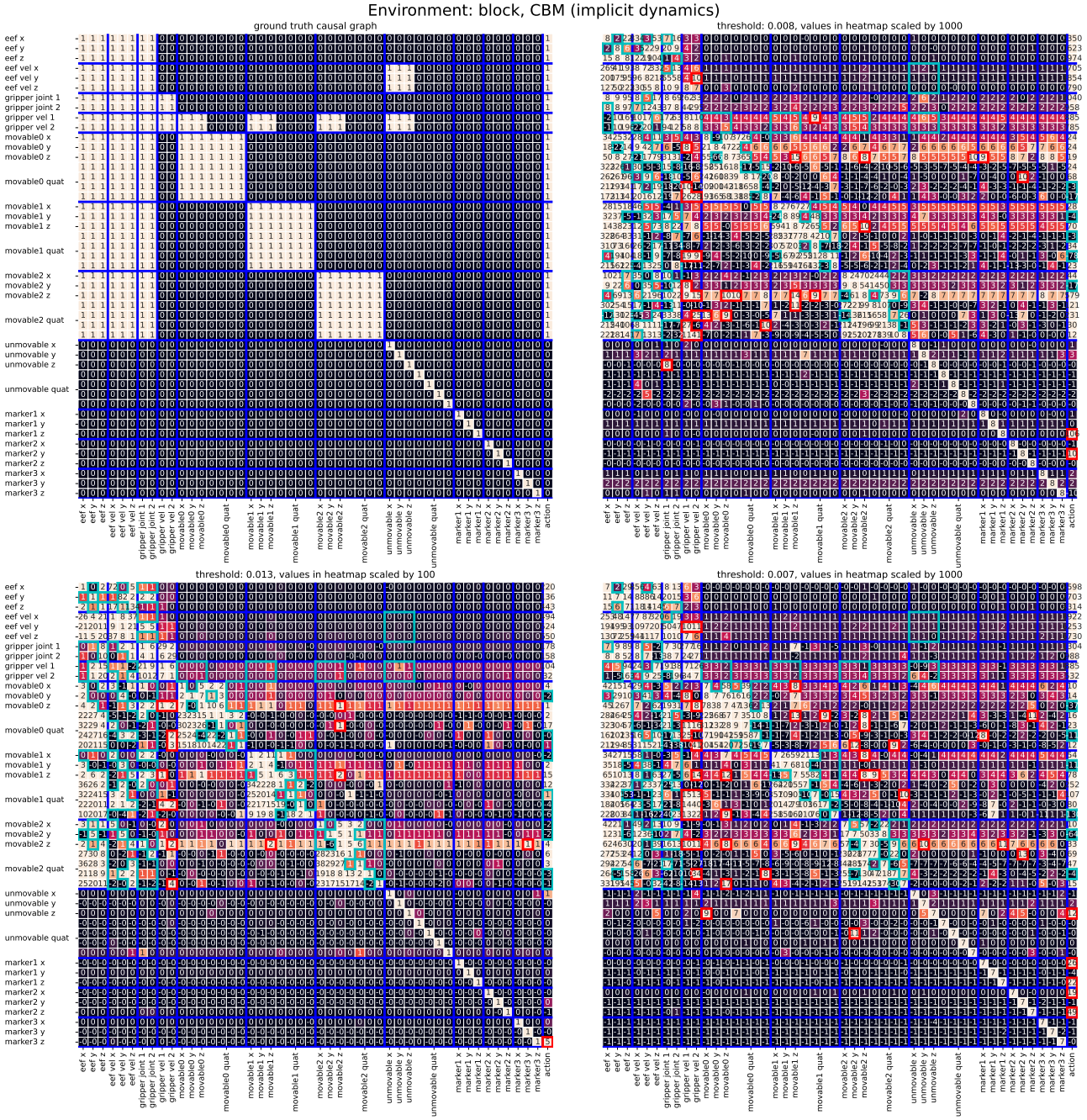

Figure 7: The ground truth dynamics causal graph and causal graphs learned by CBM in the block environment across 3 seeds.

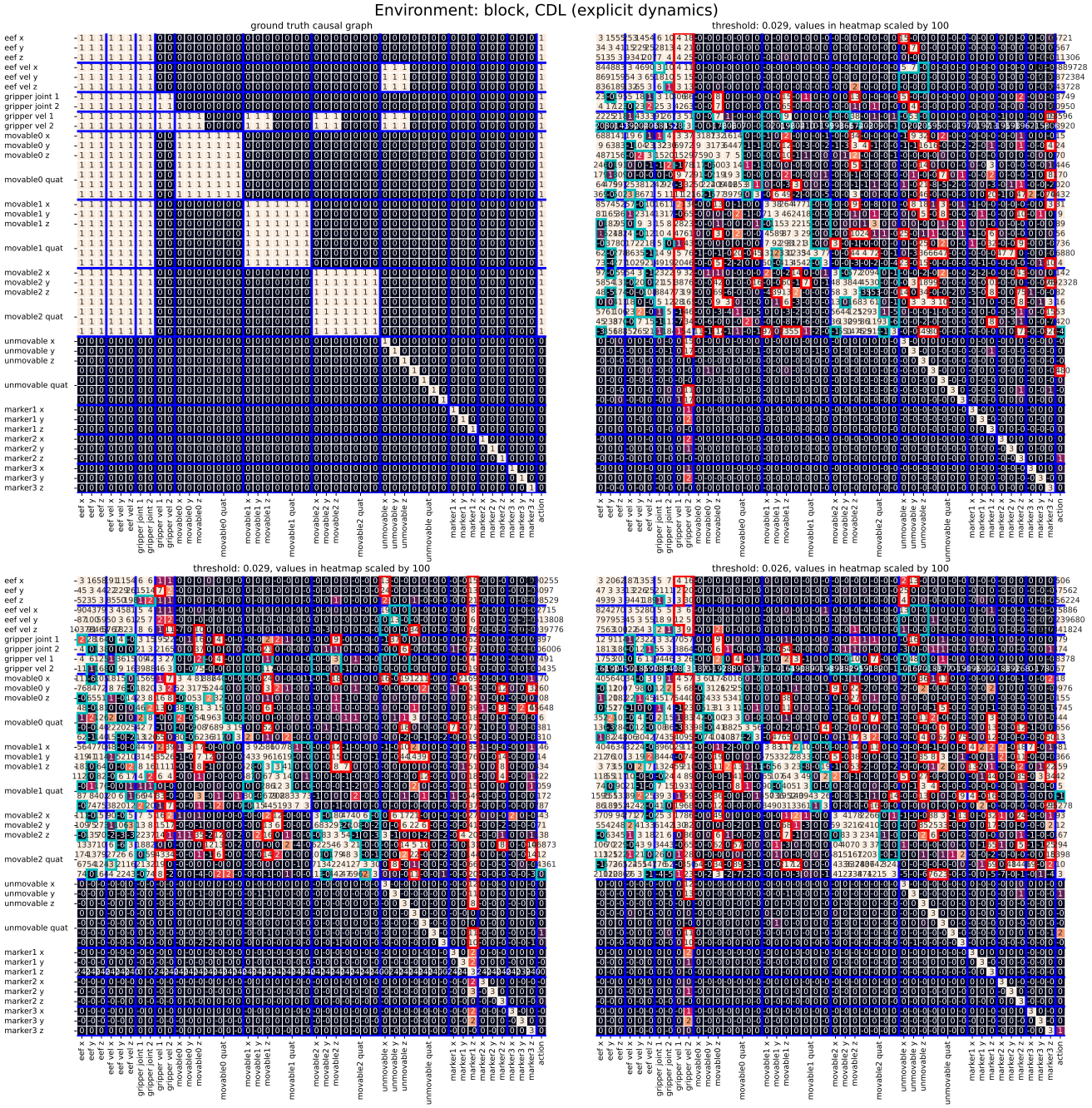

Figure 8: The ground truth dynamics causal graph and causal graphs learned by CDL in the block environment across 3 seeds.

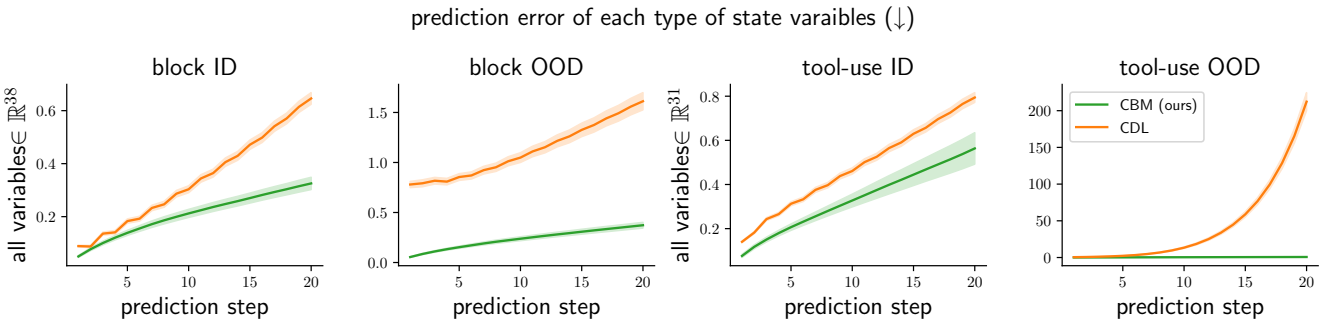

Figure 9: Dynamics prediction error in the block and tool-use environments.

Table 3: Prediction error and wall time for explicit dynamics and implicit dynamics with different number of samples.

	explicit	implcit (ours)					
		256	512	10	2048	4096	8192
prediction error	0.09 ± 0.05	0.17 ± 0.01	0.10 ± 0.01	0.07 ± 0.00	0.06 ± 0.00	$\mathbf{0.05} \pm 0.00$	$\mathbf{0.05} \pm 0.00$
wall time (s)	$\mathbf{5.79} \pm 0.21$	11.33 ± 0.06	16.47 ± 0.40	25.69 ± 0.37	47.86 ± 0.19	83.47 ± 2.44	163.35 ± 5.35

and wall time (in seconds) when choosing different numbers of samples and compare them with explicit models in the block environment ($d_S = 47$) on 25K transitions.

As shown in Table 3, one can use fewer samples (e.g. 1024 or 2048) to achieve a prediction error similar to using 8192 samples. Notice that implicit dynamics with those smaller sample sizes take only about 5x or 8x of the computation time for explicit dynamics, much lower than the theoretical ratios (i.e., $1 + \frac{N}{d_S}$ which are around 23x or 45x respectively), as we use a smaller network to extract feature from samples $s_{t+1}^{i,n}$ than from the current state variable s_t^j.

Task Learning Implementation and Further Results

In this section, we give more details on the RoboSuite environments (block and tool-use) used in the main paper, and methods for the sample efficiency experiments.

Task Rewards

For DMC tasks, we use the reward function in the official implementation code. For Robosuite tasks, let $eef_t \in \mathbb{R}^3$ be the current end-effector position and $g \in \mathbb{R}^3$ is the target position in this episode. The reward functions for the tasks used in the experiments of Sec. , are defined as follows:

Pick (B): raise the block mov to the target position g,

$$\begin{aligned} r_t = & \, 0.2(1 - \tanh\left(2.0||eef_t - mov_t||_2\right)) \\ & + \mathbb{1}\left[mov \text{ is grasped}\right] \left(0.4 + 0.5(1 - \tanh\left(5.0||mov_t - g||_2\right)\right) \\ & + \mathbb{1}\left[||mov_t - g||_2 < 0.05\right]. \end{aligned}$$

Stack (B): stack the movable object mov on the top of the unmovable object unm.

$$\begin{aligned} r_t = & \, 0.2 \cdot (1 - \tanh\left(2.0 \| eef_t - mov_t \|_2\right)) \\ & + 0.4 \cdot \mathbb{1}\left[mov \text{ is grasped}\right] \\ & + 0.5(1 - \tanh\left(5 || mov_{x,y} - unm_{x,y} ||_1\right)) \cdot \mathbb{1}\left[mov0 \text{ is lifted}\right] \\ & + 2.0 \cdot \mathbb{1}\left[success\right], \end{aligned}$$

where the notation $mov_{x,y}$ refers to the x and y coordinates of mov, and similarly for unm.

Table 4: Parameters of the reward predictor and SAC. Parameters shared if not specified.

Method	Name	Tasks				
		Pick (B)	Stack (B)	Series (T)	Cheetah (T)	Walker (DMC)
Reward Predictor	feature architecture	[128, 128]				
	predictor architecture	[128, 128]				
	activation functions	ReLU				
	training step	50K				
	learning rate	3e-4				
	batch size	64				
SAC	horizon	250		400		1000
	actor architecture	[256, 256]				
	critic architecture	[256, 256]				
	actor activation functions	[Relu, Relu]				
	critic activation functions	[Relu, Relu]				
	TD steps	1				
	batch size	256				
	grad clip norm	10				
	actor/critic learning rate	1e-4				
	tau	5e-3				
	gamma	0.99				
	buffer size	5e6				
	alpha start	0.9	0.9	0.9		0.5
	alpha finish	0.1	0.05	0.1		0.1
	alpha decay	0.666	3.333	5		1

Series (T): use the L-shaped tool to move the faraway block closer to the robot, then pick up the block and place it in the pot.

$$
\begin{aligned}
r_t = & \, 0.2 \cdot (1 - \tanh(2.0 \| eef_t - tool_t \|_2)) \\
& + 0.2 \cdot \mathbb{1}[tool \text{ is grasped}] \\
& + 0.2 \cdot (1 - \tanh(5 \| tool_t - block_t \|_2)) \cdot \mathbb{1}[tool \text{ is grasped}] \\
& + 0.2 \cdot (1 - \tanh(5 \max(-block_x, 0)) \cdot \mathbb{1}[tool \text{ is grasped}] \\
& + 0.4 \cdot \mathbb{1}[tool \text{ is not grasped}] \cdot \mathbb{1}[block_x < 0] \\
& + 0.4 \cdot (1 - \tanh(5 \| eef_t - tool_t \|_2) \cdot \mathbb{1}[block_x < 0] \\
& + 0.6 \cdot \mathbb{1}[block \text{ is grasped}] \\
& + 0.5 \cdot (1 - \tanh(5 \| block_{x,y} - pot_{x,y} \|_2)) \cdot \mathbb{1}[block \text{ is grasped}] \\
& + 2.0 \cdot \mathbb{1}[success] \, .
\end{aligned}
$$

State Abstraction Learning Methods

For Pick and Stack, the dynamics model is pretrained for all methods and frozen during task learning, and only the reward predictor is learned. For Series and DMC tasks, the dynamics model is learned from scratch during task learning, jointly with the policy and the reward predictor.

CDL For CDL's task-independent abstraction, the dynamics model is trained offline on 2M pre-collected transitions using scripted policies and then used to derive the abstraction, which remains fixed during task learning.

TIA and Denoised MDP They are originally implemented for representation learning of image state spaces. To adapt to factored state spaces, we replace their encoders and decoders with a binary mask that partitions state variables into task-relevant or irrelevant parts. The mask is jointly optimized with the dynamics and reward models using the Gumbel reparameterization trick (Jang, Gu, and Poole 2016). While the original implementations of TIA and Denoised MDP both learn dynamics from scratch during task learning, in Pick and Stack tasks, CDL and CBM use a task-independent dynamics model that is already trained by the task-learning phase. Thus, for a fair comparison, TIA and Denoised MDP are also initialized with pretrained dynamics models, where only the final layer of the model is allowed to vary during task learning. We also tried training the dynamics model from scratch, but this led to worse performance. For TIA and Denoised MDP, we conducted a search on the following hyperparameters:

- Gumbel temperature scheduling, among the final temperature of $\{0.05, 0.1, 0.2, 0.5, 1.0, 2.0, 5.0, 10.0\}$.

- Regularization coefficient on size of the abstraction, i.e., the number of the reward-relevant state variables for TIA, and the number of controllable and reward-relevant variables for Denoised MDP, among $\{10^{-3}, 10^{-4}, \cdots, 10^{-7}\}$.

We hypothesize that the reason why TIA and Denoised MDP perform worse compared to CBM (Sec) is that both methods rely on assumptions that do not hold in general MDPs, leading to inaccurately learned abstractions and inefficient task learning. Specifically, TIA assumes there is no dynamical dependency between two segments in the representation space (i.e., state variable space in our setting) — reward-relevant one and reward-irrelevant one. Meanwhile, Denoised MDP makes a similar assumption about dynamical independence between different segments. However, in general MDPs, reward-irrelevant state variables could be dynamically dependent on reward-relevant variables. For example, when the task is to pick block A with the robot arm, though block B is reward-irrelevant, block B still can be manipulated by the arm. As a result, block B will also be included in the same segment to minimize its prediction error. Hence, TIA and Denoised MDP often include all controllable variables in their abstractions. In contrast, CBM doesn't have this "dynamic independence" assumption and identifies dynamical causal relationships between state variables, thus it can remove controllable variables that are task-irrelevant from its abstraction.

SAC Policy We adopt the implementation of SAC by Tianshou (Weng et al. 2022). For most SAC parameters, tuned hyperparameter values published by prior literature worked well across all tasks. The exception is α, the entropy regularization coefficient, which controls the relative weights of the return and entropy terms in SAC. We found that automatic entropy scheduling often led to early convergence and task failure, so for each task, we defined a manual entropy decay schedule, which we tuned separately for each task. The entropy schedule is a function of both the current and total training steps (a constant). It is defined below and has the following parameters: $\alpha_{start}, \alpha_{finish}, \alpha_{decay}$.

$$\alpha(t)_{t_{total}} = (\alpha_{start} - \alpha_{finish}) \exp\left(\frac{-\alpha_{decay} \cdot t}{t_{total}}\right) + \alpha_{finish}.$$

The architecture and hyperparameters of SAC are listed in Table 4. SAC hyperparameters are shared across all methods for each task. We select between [0.5, 1.0] for α_{start}, [0.0, 0.2] for α_{finish}, and [0.5, 5] for α_{decay} for best SAC performance.

CBM For CBM, the reward predictor is jointly trained with the policy, the learned reward causal graph may change during training and thus change the derived bisimulation abstraction. When the abstraction changes, CBM resets π and relearns the policy from existing data in the replay buffer. Policy resetting is a technique proposed by Nikishin et al. (2022), who showed that periodically resetting the policy and retraining from the replay buffer may improve both sample efficiency and asymptotic performance for deep RL agents. For a fair comparison, policy resetting is applied to TIA and Denoised MDP as well. The architecture and hyperparameters of the reward predictor are listed in Table 4.

Learned Task-Specific Abstraction

For the Stack task, the learned state abstraction by each method is shown in Fig. 10. Again, CBM keeps all reward-influencing variables and their causal ancestors. Despite our efforts to hyperparameter tune, TIA and Denoised fail to learn meaningful abstractions. We found that both methods were highly sensitive to their regularization coefficients. Note that the Stack task also violates their assumptions of each component having independent dynamics, which may explain their failure to learn good abstractions.

Task Learning Ablation

CBM learns implicit dynamics and a causal reward function. In Sec. , we show that implicit dynamics models surpass explicit models in terms of causal graph accuracy and state abstraction accuracy. Fig. 11 shows an ablation of CBM which instead uses explicit dynamics. We observe that on the Pick task, the difference from CBM is not significant as Pick is relatively easy. On the more challenging Stack task where accurate abstraction plays a more important role, the performance of explicit dynamics is much worse than CBM which uses implicit dynamics.

Compute Architecture

The code is implemented with pyTorch. The 5 seeds selected are 0 - 4, and the seed can be specified in the configuration file. The experiments were conducted on machines of the following configurations:

- Nvidia Titan V GPU; Intel(R) Xeon(R) CPU E5-2630 v4 @2.20GHz
- Nvidia V100-SXM2 GPU; Intel(R) Xeon(R) CPU E5-2698 v4 @2.20GHz
- Nvidia A40 GPU; Intel(R) Xeon(R) Gold 6342 CPU @2.80GHz
- Nvidia A100 GPU; Intel(R) Xeon(R) Gold 6342 CPU @2.80GHz

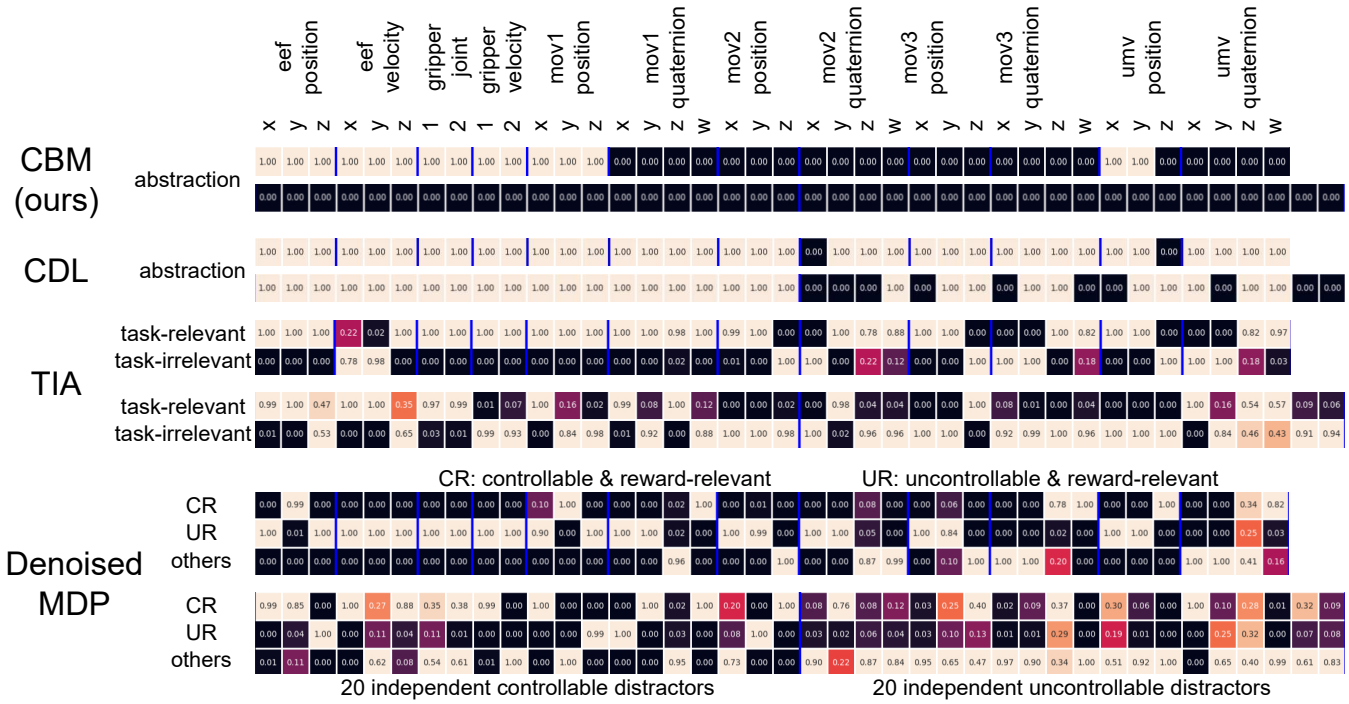

Figure 10: State abstractions learned by CBM, TIA, and Denoised MDP for the Stack task.

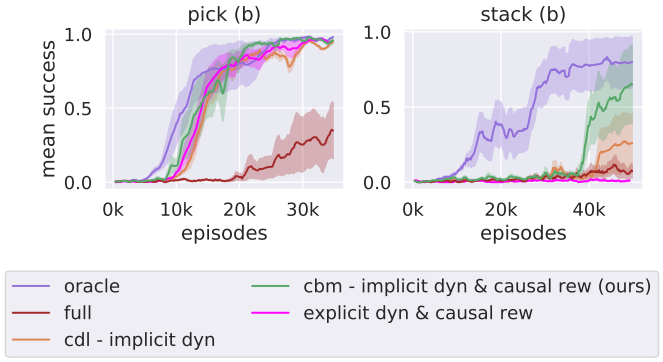

Figure 11: Learning curves of CBM and CDL (which both use implicit dynamics in the main paper), and an ablation of CBM that uses explicit dynamics on Pick and Stack tasks. We observe that the ablation has much worse sample efficiency on Stack.