Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Engineering

Causal Counterfactuals for Improving the Robustness of Reinforcement Learning

Tom He

April 19, 2022

A dissertation submitted in partial fulfilment
of the requirements for the degree of
MAI (Computer and Electronic Engineering)

Abstract

Intelligent robots are increasingly being considered for use alongside humans and in critical applications such as healthcare and manufacturing. Reinforcement Learning (RL) can enable agents or robots to learn tasks unsupervised. If RL is to be deployed for real-world robotic applications, the decisions or actions of the RL agent need to be trustworthy. The robustness and explainability of RL need to be improved for RL to be trustworthy in real-world deployments.

The robustness and explainability of RL can be improved by combining RL with a field called Causal Inference. Causal RL is a combination of Causal Inference and RL. Environments are represented as Structural Causal Models (SCM) in Causal RL. The RL agent learns about the underlying SCM through three different interactions: association (seeing), interventions (doing), and counterfactuals (imagining). All three interactions are needed to learn the complete underlying SCM. The causal information that is obtained through interacting with the environment is then stored as a causal representation. The causal representation captures all the invariant causal mechanisms across domains and tasks. The causal representation can be transferred to different tasks with similar causal structures and improves the robustness of RL.

However, currently, there is limited research in Causal RL as it is a recent field. Existing approaches have limitations such as assuming the SCM is known or partially known, the Causal RL solutions are usually not complete, and Causal RL has not been applied to complex tasks like robotics. This thesis proposes the first complete Causal RL solution that is applied to complex robotic tasks. The proposed solution is called Causal Counterfactual (CausalCF). CausalCF learns about the underlying SCM from scratch using the three different Causal RL interactions. The causal knowledge is stored as a causal representation, and the representation is scalable and transferable to different tasks. CausalCF is implemented and evaluated in a realistic robotic simulation environment called CausalWorld. CausalWorld provides a range of complex robotic control and object manipulation tasks.

CausalCF combines ideas from Causal Curiosity and CoPhy. Causal Curiosity provides an approach for using interventions and a causal representation to train an RL agent. CoPhy is a Deep Learning solution that can perform counterfactuals. CoPhy is adapted for use in RL. Each component of the CausalCF design was evaluated in CausalWorld. All the components of the CausalCF design improved the training performance and the robustness of the RL agent. The causal representation learnt from one task was directly transferred to train an RL agent in another task. The causal representations captured the causal mechanisms that remain invariant across tasks and improved the RL agent's training performance and robustness. CausalCF can be directly applied to the different tasks available in CausalWorld.

Acknowledgements

I would like to sincerely thank my supervisor, Prof. Ivana Dusparic, for her guidance and support. Ivana has been a fantastic supervisor, and her continuous guidance and support have been essential to completing this project. I received regular feedback about the progress of my research and enough freedom to explore different research topics and applications. I am truly grateful for being able to perform research in the fields that I am most passionate about, Robotics and AI (Reinforcement Learning).

I would also like to sincerely thank my PhD supervisor, Jasmina Gajcin, for her guidance and support throughout this project. Jasmina's PhD research project is also in the field of Causal RL and provided many helpful insights throughout this project.

Last but not least, I would like to thank my family for their love and support. Without the support of my family, I would not have been able to pursue and complete this degree.

Contents

List of Figures

List of Tables

Nomenclature

CausalCF -	Causal Counterfactual
CF_model -	Counterfactual model
Causal_rep -	Causal representation

Causal Inference:

SCM -	Structural Causal Model
PCH -	Pearl Causal Hierarchy
ICM -	Independent Causal Mechanisms principle
SMS -	Sparse Mechanism Shift hypothesis
POMIS -	Possibly-Optimal Minimal Intervention Set

Reinforcement Learning and Deep Learning:

FOMDP -	Fully Observable Markov Decision Process
POMDP -	Partially Observable Markov Decision Process
SAC -	Soft Actor Critic method

CausalWorld:

bs -	block size: meters (m)
bm -	block mass: kilograms (kg)
Space A and B -	Ranges of values for the CausalWorld variables.

1 Introduction

Reinforcement Learning (RL) uses unsupervised learning and suffers from the problem of dimensionality for large complex problems. The performance of Neural networks scales efficiently with large amounts of training data, and Neural networks can approximate any continuous function. Therefore, the state-of-the-art RL algorithms combine Deep Learning (DL) and RL into Deep Reinforcement Learning (DRL) to deal with the problem of dimensionality. However, Both DL and RL algorithms cannot generalize to environments that are out-of or a shift from the training distribution. DL architectures are often complex and are treated as black-box systems. For human users to trust the decisions made by the DRL systems, the robustness and explainability of DRL need to be improved.

Existing approaches [2] try to improve the explainability of DRL through different forms of simplification. [8] simplifies Neural networks into Linear Model U-Trees, which are more interpretable, but at the cost of "play performance." Another method called Hierarchical Policies [9] decomposes large complex tasks into simpler sub-tasks. However, the method requires humans to define the hierarchy and the different sub-tasks needed to complete a task. The existing approaches require many design decisions to transfer existing "knowledge" or "skills" to new tasks or domains, and it is a non-trivial task.

Simplifications cannot be the only approach for improving the explainability of DRL. The human brain is complex and is still not fully understood. As humans, we can trust one another despite how complex we are because we have the mechanisms that allow us to explain our actions and intentions.

Causal Inference [10, 11, 12] and RL are two fields that have been studied for some time. However, only recently have the two fields been combined to form Causal RL [13]. Causal RL is inspired by the way humans learn. Humans build causal models to represent the world around them. Humans learn these causal models through interacting with the world in different ways and observing the effects of these different interactions. Causal RL uses Structural Causal Models (SCM) to represent the world or the environment, and the SCM is stored as a causal representation. Causal RL learns a causal representation through three different interactions: association (seeing), intervention (doing), and counterfactual (imagining). The causal representations learnt capture causal mechanisms that remain

invariant across domains and, therefore, improve the robustness of DRL. The ability of the Causal RL agent to imagine different consequences allows the agent to explain the different choices of actions with how it aligns with the agent's intentions. A new benchmark called CausalWorld [6] aims to support research in Causal RL. CausalWorld provides a realistic simulation environment for training and testing Causal RL solutions in complex robotic tasks.

1.1 Challenges of Causal RL

Causal RL is a new field, and there is currently a limited amount of research on this field. However, Causal RL shows potential for improving the robustness and explainability of RL, and the field has been gaining interest in the most recent years. The biggest criticism about Causal RL is its feasibility for real-world problems because much of the existing research assumes the SCM is known or partially known [14, 15, 16]. Existing research usually investigates how to infer the causal effects from a known SCM, but not how to learn about the underlying SCM from scratch using the three different Causal RL interactions. Research that focuses on causal structure learning is usually not applied to RL [5, 16, 17]. Research on causal structure learning for RL usually does not provide the complete Causal RL solution [6, 18, 19], or it is not implemented for complex tasks that represent real-world tasks [20, 21].

1.2 Thesis Objective

The main objective of this thesis is to improve the robustness of RL through Causality and with the design for better explainability in mind. This thesis aims to design and implement a complete Causal RL solution called CausalCF (Causal Counterfactual). CausalCF will learn about the underlying SCM of an environment from scratch through observations, interventions and counterfactuals. The learnt causal knowledge is stored as an abstract causal representation that is scalable and transferable to different tasks. CausalCF will be trained and evaluated in the CausalWorld environment. CausalWorld provides a range of complex robotic control and object manipulation tasks where the RL agent has to coordinate three separate fingers to arrange blocks into their desired goal shapes.

1.3 Thesis Assumptions

A few assumptions are made in this thesis to be able to implement and evaluate the CausalCF solution. The assumptions are listed below:

- The causal variables of the environment used for training and testing the CausalCF

solution is accessible.

- CausalCF was designed to be trained and tested in CausalWorld. If CausalCF is applied to a different simulation environment or tasks different from CausalWorld, modifications to the design are needed.
- CausalWorld accurately models real-world physical dynamics and constraints. Therefore, the performance of CausalCF in CausalWorld corresponds to the feasibility of CausalCF for complex real-world tasks. CausalWorld provides techniques for reducing the Sim2Real gap (discussed in Chapter 2).

1.4 Thesis Contributions

This thesis identifies the need for Causal RL and provides the background information needed for understanding the proposed Causal RL solution (CausalCF). To our knowledge, this is the first attempt to design and implement a complete Causal RL solution for complex RL robotic tasks. CausalCF is the first Causal RL solution where the RL agent is provided with a mechanism to imagine future observations from different forms of interventions. CausalCF provides an approach for how interventions, counterfactuals and causal representations could be incorporated for various RL robotic applications. The contribution of the different components of the CausalCF design is evaluated. Evaluations will show if interventions and counterfactuals improve the RL agent's robustness. The causal representation learnt in one task will directly be transferred to train an RL agent in another task. The two tasks have similar causal structures. If the causal representation captures the invariant causal mechanisms, the RL agent will perform robustly in the new task. CausalCF will be evaluated in the challenging Stacking2 task provided by CausalWorld. CausalWorld [6] evaluated the state-of-the-art RL algorithms in the Stacking2 task, and all of them failed to converge to a good performance. This thesis also provides interesting future directions based on the work that has been done for CausalCF.

2 Background and Related Work

This chapter aims to identify the need for Causal RL and provide all the background information needed for understanding Causal RL and the proposed CausalCF solution. The most significant issues for current DRL systems are explainability and robustness. Explainability and robustness are especially important for RL because it is trained unsupervised. Explainable Reinforcement Learning (XRL) is an open area of research that may incorporate concepts from many different fields. The approach taken in this paper to help achieve XRL is by learning the underlying causal structures of tasks through a method called Causal RL. Causal RL would enable agents to generalize to other tasks (robustness) and provide more explainability. The chosen area of application for Causal RL is robotics. Intelligent robots are being considered for more critical roles like healthcare, manufacturing and search and rescue. It becomes essential that the decisions made by these intelligent robots can be trusted by their human partners. Section 2.1 and 2.2 will explain the concepts related to Reinforcement Learning (RL) and Deep Learning (DL) and how these two fields combine to become Deep RL (DRL). Section 2.3 introduces Explainable AI (XAI) and XRL, which aims to convert these black box systems to grey or white boxes that can be explained. Section 2.4 presents the concepts related to Causal Modeling and how it can be used to make RL more explainable and robust (Causal RL). Section 2.5 discusses how Causal RL can be applied to Robotics and make Robotic agents more robust and explainable.

2.1 Reinforcement Learning

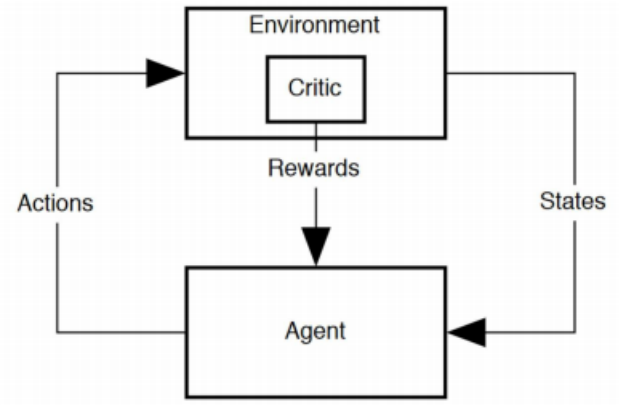


Figure 2.1: Adapted from [1], showing the interactions between the agent and the environment for Reinforcement Learning systems.

An RL agent receives information about its current state (S_t) and reward (R) from the environment. Based on the current state and the rewards the agent has received, the agent decides which action (A) it should take next, as shown in Figure 2.1. An RL agent may use one or more of these components to decide which action it should take next: value function, policy and model. The RL agent aims to find the optimum value function, policy, or model, to maximize cumulative rewards. The search for the optimum is an iterative process that involves exploration and exploitation.

All the states (S) in the RL environment are Markov, and they capture all the relevant information about the history [22]. Equation 1 describes the condition under which a state is Markov.

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ..., S_t] \tag{1}$$

The probability of the transitions between all the states s and its successor states s' is described by a State transition matrix $\mathcal{P}_{ss'}$.

The reward (R) is determined by the Critic depending on the current state of the environment. The agents aim to maximize the cumulative reward. Therefore, the agents want to consider both the states' long-term and immediate rewards, which is represented by the return (G_t).

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2a}$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \tag{2b}$$

$$G_t = R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ...) \tag{2c}$$

$$G_t = R_{t+1} + \gamma G_{t+1} \tag{2d}$$

The R_{t+1} represents the immediate reward and estimates of the future rewards $\gamma(R_{t+2} + \gamma R_{t+3} + ...)$ are discounted. The discount factor $\gamma \in [0, 1]$ is used because the uncertainty increases with the predictions about the rewards collected further into the future.

The value function v(s) describes how good or bad the states are; it predicts future rewards [22].

$$v(s) = E[G_t|S_t = s] \tag{3a}$$

$$v(s) = R_{t+1} + \gamma E[v(S_{t+1})|S_t = s] \tag{3b}$$

$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in S} \mathcal{P}_{ss'} v(s') \tag{3c}$$

Equation 3b is derived from Equation 3a by substituting in the result from Equation 2d. Equation 3b and 3c are representations of the Bellman equation, where $\mathcal{R}_s$ represents the immediate reward and $\sum_{s' \in S} \mathcal{P}_{ss'} v(s')$ is an estimate of the future rewards. Based on the value function, an action can then be chosen to maximize the cumulative rewards (value based). The value function can be used to update the policy (Actor Critic) or the policy can be updated directly from the states and rewards received from performing specific actions (policy based). The policy π is a mapping from state to action and is defined in Equation 4.

$$\pi(a|s) = P[A_t = a|S_t = s] \tag{4}$$

A value function can provide the expected return for following a policy from an initial state (state-value function $v_\pi(s)$) or an initial state and action (action-value function $q_\pi(s, a)$). Equations 5b and 6b are the Bellman equations for the state-value function and the action-value function respectively.

$$v_\pi(s) = E_\pi[G_t|S_t = s] \tag{5a}$$

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s] \tag{5b}$$

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a] \tag{6a}$$

$$q_\pi(s, a) = E_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \tag{6b}$$

The RL agent finds the optimal value functions $v_*(s)$ or $q_*(s, a)$ by solving the Bellman equations through trial and error, and the optimal policy π_* is obtained when the optimal value function is obtained [22].

$$v_*(s) = \max_\pi v_\pi(s) \tag{7}$$

$$q_*(s, a) = \max_{\pi} q_\pi(s, a) \tag{8}$$

An RL environment can be described formally using the Markov decision process [23]. The Markov decision process (MDP) is defined by the following parameters that have been discussed above: $\mathcal{S}$ is a finite set of states, $\mathcal{P}$ is the state transition probability matrix, $\mathcal{R}$ is a reward function, γ is the discount factor and $\mathcal{A}$ is a finite set of actions. In a fully observable MDP (FOMDP), the agent observes the complete state of the environment. However, in real-world applications such as robotics, the robots only observe the environment partially through a camera attached or some other form of sensor. Therefore, real-world scenarios are often partially observable MDPs (POMDP). POMDP has two extra parameters that define it, in addition to those of the FOMDP: $\mathcal{O}$ is a finite set of observations and $\mathcal{Z}$ is an observation function. In robotic applications, sampling real experiences directly from the environment can be costly. If the environment is a FOMDP, then a model can be learnt and simulated experiences can be sampled from the model to train the RL agent (model-based approach). A model is an agent's internal representation of the environment. The process of finding the optimal value function (and/or policy) from sampled simulated experiences is called Planning. A model-free approach is often used for a POMDP. In the model-free approach, the optimal value function (and/or policy) is found from sampled real experiences directly from the environment, and this process is known as learning. A hybrid model called Dyna-Q [22] captures the advantages of both the model-based and the model-free approaches. Dyna-Q learns a model from real experience, and then it plans and learns a value function (and/or policy) from real and simulated experiences.

2.1.1 Q-Learning

Q-Learning [24] is a commonly used model-free approach that is off-policy. Off-policy is an approach where the agent samples actions from an exploratory or behaviour policy μ and optimises the target policy π with respect to the value function from the real sampled experiences. Q-Learning converges to the optimal action-value function $q_*(s, a)$ if there are enough samples of all actions in all states through Q-value iteration. In many real-world tasks with a large state and action space and limited training resources, a sub-optimal solution is more feasible to aim for and is defined with a maximum number of iterations or the desired loss value. The Q-value iteration is described in Equation 9. The Q-value function $Q(s, a)$ provides the return from any state and action in set spaces S and A, respectively, action a is sampled from the behaviour policy μ, action a' is a subsequent action sampled from the target policy π, state s' is the successive state from state s and action a and α is the learning rate.

$$Q(s, a) = Q(s, a) + \alpha(R + \gamma \max_{a'} Q(s', a') - Q(s, a)) \tag{9}$$

Where $\alpha \in (0, 1]$ and $\gamma \in [0, 1]$. The target policy π acts greedy with respect to the Q-value function $Q(s, a)$, exploiting what it has learnt to maximize rewards. The behaviour policy μ acts ϵ-greedy with respect to $Q(s, a)$, exploring new actions and visiting new states. In ϵ-greedy exploration, the agent chooses a greedy action with probability 1-ϵ and a random action with probability ϵ.

2.2 Deep Learning and Deep RL

Many of the recent successes of Machine Learning is attributed to the success of Deep Learning. All Deep Learning methods are based on Neural Networks. The performance of Neural Networks scales efficiently with the amount of training data. Therefore, the increase in the availability of large amounts of freely accessible data and cheaper computing power has contributed to the quantum leaps of success in Deep Learning. The Deep Neural Networks was initially designed to model the brain with artificial neurons representing biological neurons. However, modern research on neural networks is based on many mathematical and engineering disciplines, and the aim is not to perfectly model the brain [25].

2.2.1 Feedforward Neural Network

Feedforward Neural Networks (also known as Multi-layer perceptrons (MLP) or Deep Feedforward Networks) serve as a basis for many of the most powerful and popular Deep Learning architectures used in commercial applications today [25]. Feedforward Neural Networks aim to learn an approximate function mapping from inputs x to outputs y. Feedforward Neural Networks are composed of an input layer, at least one hidden layer and an output layer. Information flows from the input to the intermediate hidden layers that define the mapping and to the output layer. The MLP's outputs have no feedback and, therefore, the term 'feedforward' in the name [25]. If feedback is added to the Feedforward Neural Network, it will become a Recurrent Neural Network (RNN). The term 'Neural' comes from the fact that MLP was inspired by neurosciences [25]. A hidden layer may consist of many hidden units defined by parameters θ, each unit represents a neuron in the brain. Feedforward Neural Networks typically have many different functions used at different layers. Therefore, it is a 'network' of different functions [25]. The depth of the Neural Network is determined by the number of hidden layers it has. The activation functions used at each layer are non-linear functions. ReLU and variants of it are the most frequently used activation functions.

A Feedforward Neural Network can approximate any continuous function with just one hidden layer to the desired accuracy, given that it has enough hidden units [26]. However, the number of units or parameters that the single layer will need to learn the mappings

$y = f(x, \theta)$ can become impractical. Deeper models require fewer parameters and generalize better than the single layer in many situations [27, 28, 29]. Most of the Neural Networks used today consist of many layers. The primary research revolves around the number of hidden units each layer should have and how these units should be connected. Every unit of a layer can be connected to every unit in an adjacent layer and this is called a fully connected or dense layer.

2.2.2 Backpropagation and Optimisation

In a Feedforward Neural Network, the information is propagated from the input through all the hidden layers and to the output. The error $e(f(x, \theta), y)$ between the computed output and the desired output is calculated using a loss function L(e). In order to obtain the desired mappings from input to output, the errors at the output need to be propagated backward through the layers to calculate the gradients. This is called back-propagation or backprop. The units' parameters or weights are updated using the calculated gradients to minimize the loss function using various forms of gradient descent.

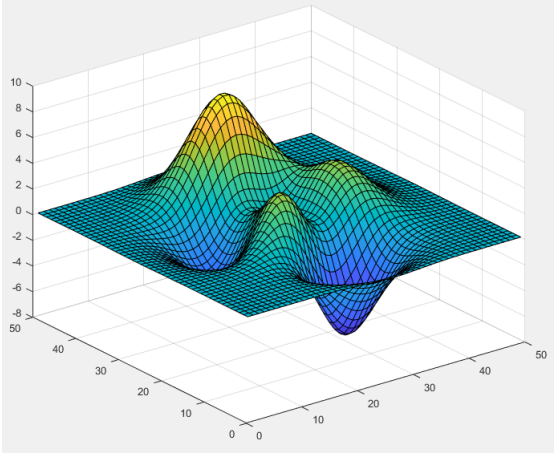


Figure 2.2: Plot created from Matlab that is a visual representation of a loss function.

The loss function usually consists of many local minima, as shown in Figure 2.2, and there is usually noise in the training data. The questions of when and how the gradient information should be used to update the parameters need to be considered for finding the global minimum or the desired minimum. The loss function is usually calculated as the average error over all n observations as shown in Equation 10.

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} e(f(x_i, \theta), y_i) \tag{10}$$

The gradient $\frac{\partial L}{\partial \theta}$ will be calculated once after all n observations in batch gradient descent. Batch gradient descent provides a more accurate estimate of the gradient. However, updates

are less frequent, and it might take longer to reach the desired minimum. Also, the memory usage scales with the size of the training data. In stochastic gradient descent (SGD), the gradient is calculated after every observation or example in the training data. SGD leads to high variances in the estimates of the gradients, and it will take more steps to reach the desired minimum. Mini-batch gradient descent is more commonly used, which calculates the gradient with more than one training example and less than all the examples. Optimisation algorithms determine how the gradient information should be used to update the parameters. Optimisation algorithms such as Adagrad [30], RMSProp [31] and Adam [32] provide more stable and efficient updates which leads to faster convergence.

In Adagrad, each parameter has a learning rate, and the algorithm adaptively updates the learning rates based on the past updates [30]. Adagrad helps with finding rare features or parameters that could be very informative [30]. The learning rate determines the size of the updates. The learning rate in Adagrad may shrink with the past updates, and it might become too small before reaching a convex structure [25]. RMSProp improves on the Adagrad by using a weighted moving average that solves the diminishing learning rates [25]. Adam uses moments and is like a combination of Adagrad and RMSProp. However, no combination of techniques will guarantee convergence to a global minimum or the desired minimum in every scenario.

2.2.3 Convolutional Neural Network

Convolutional Neural Networks (CNN) [33] are a kind of Neural Network that is specially used for processing images or data with grid-like topology. CNN uses a mathematical operation called convolution to take advantage of the spatial structure in images. CNN reduces the number of units or parameters needed over Neural Networks that only employ fully connected layers. It reduces the number of parameters by using sparse connectivity and parameter sharing. CNN convolves a kernel over an input array (like an image) and produces a new output array with a smaller dimension than the input. The size of the kernel is smaller than the input, and each unit in the output is connected to a patch of units on the input (sparse connections). The kernel uses the same weights or parameters to convolve over the different locations of the input (parameter sharing). CNN can deal with different input sizes and produce desired output sizes by using padding, strides, pooling, or changing the size of the kernel. Pooling is performed after the nonlinear activation function (ReLU) is applied to the output array. Pooling provides a statistical summary of the output's different locations and is commonly used in CNN. Many of the current architectures make use of convolutional layers. The layers decrease in dimensions with the convolutions, and the last layers are fully connected layers. Initial layers usually contain features like edge lines and curves, and high-level features like an eye are isolated towards the final layer, which is crucial for classification tasks.

2.2.4 Deep RL

The value function is crucial in RL algorithms, especially for the Q-Learning method. A lookup table is a possible option for representing the value function. However, a lookup table is infeasible for many tasks that have large state or action spaces. Usually, the value function is approximated. Neural Networks can approximate any function, and the performance scales efficiently with the amount of data. Therefore, Neural Networks are often used for representing the value function. Neural Networks could also represent other components of the RL algorithm like the model and the policy. Deep RL is a combination of Deep Learning and RL.

2.3 Explainable Reinforcement Learning (XRL)

AI systems and methods are becoming more powerful, but it is also becoming more complex. The output decisions or results of the AI systems are often taken as a fact and executed or accepted without any understanding of why the systems produced such outputs. In critical applications such as healthcare, law enforcement, and manufacturing, executing poorly understood or tested actions can lead to disastrous outcomes. There is a performance-transparency trade-off in AI systems [2]. AI systems are increasingly being considered to guide human decision-makers or operators who are experts in fields other than AI. Humans will not trust, use or work with systems they do not understand, especially in RL, where the agents are trained under no supervision. For AI systems to be trusted, it needs to be transparent and the decisions need to be justifiable, which enables the system to be fair and ethical [2]. In 2018, the EU's General Data Protection Regulation (GDPR) provided users the right to an explanation of decisions made by autonomous systems that may affect the users significantly [34]. In addition, the European Commission also produced the final assessment list of Ethical Guidelines for Trustworthy AI in 2020 [35]. Requirements such as technical robustness and safety, transparency, fairness and accountability were all included in the Ethical Guidelines [35]. Therefore, it is essential to derive methods that can explain the decisions of autonomous RL agents to humans that work with such systems.

Explainability methods for AI can be grouped using the following four categories: local versus global and intrinsic versus post-hoc [2]. Local and global refers to the scope of explanations that the methods provide. Intrinsic and post-hoc refers to when the methods generate or extract the explanations. The taxonomy of Explainable AI (XAI) methods is shown in Figure 2.3. The 4 XRL methods that will be briefly described are shown in Table 2.1.

In Deep RL (DRL), a neural network is used to represent policies, and it is not explainable. Programmatically Interpretable RL (PIRL) [36] improves on DRL by representing policies using "high-level, domain specific programming language" that is more interpretable and

verifiable. However, PIRL still requires a neural network policy (neural oracle) to be trained using DRL before using the Neurally Directed Program Search (NDPS) to find a programmatic policy that matches closely with the neural oracle.

Linear Model U-Trees (LMUT) [8] approximate the trained neural networks used in DRL that are used to represent the Q-value function. LMUTs are simpler representations of neural networks and are more interpretable. Due to the simplification, there is a loss of "play performance," but the LMUT generalizes better than the original neural networks. LMUTs are generated post-hoc, and it will not be easy to reuse the knowledge that the LMUT has learnt. One way that LMUT could be reused for new tasks is by stacking them in a hierarchical manner.

Hierarchical Policies [9] decompose complex tasks into simpler sub-tasks. Policies learnt from simpler sub-tasks can be reused for more complex tasks. New policies can be trained on top of existing ones if a task requires a new skill. However, currently, it requires human supervision to specify the skills that new tasks will need in the training stage.

Structural Causal Models (SCMs) provide the same advantages as Hierarchical Policies in their ability to reuse learnt skills or knowledge for new or more complex tasks. SCMs might be a solution to the current limitation of Hierarchical Policies where agents could learn the skills needed based on the similarity of the causal structures of the tasks. SCMs can provide global explanations by providing their objectives and the learnt causal graph. Local explanations can be generated in the form of causal chains that answer the 'why' and 'why not' questions. Causal graphs can serve as intrinsic representations of the environment for the RL agents [18], and they can also be generated post-hoc [15].

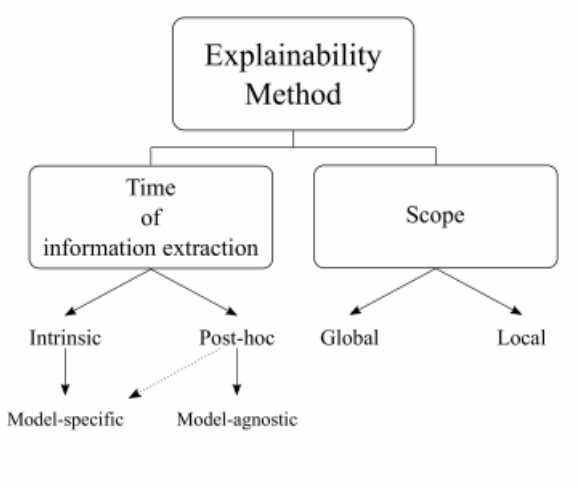


Figure 2.3: Taxonomy of XAI methods, taken from [2] which is adapted from [3].

Table 2.1: XRL methods categorized into scope and time, adapted from [2].

	Global	**Local**
Intrinsic	PIRL [36] and SCM	Hierarchical Policies [9] and SCM
Post-hoc	Linear Model U-Trees [8] and SCM	Linear Model U-Trees [8] and SCM [15]

2.4 Causal Inference and RL

The terms Causal Modeling, Causal Reasoning and Causal Inference all refer to the study of Causality [10, 11, 12]. Different papers use the terms in subtly different ways. In this paper, the term Causal Inference will refer to the problems of inferring the SCM from data and estimating the causal effects from a learnt representation of the SCM. Structural Causal Models (SCM) [10] are formally defined by the tuple (U, P(U), V, F). U is a set of exogenous variables and can represent the latent or unobserved variables in POMDP or FOMDP with large observation spaces in terms of RL. P(U) is the probability distribution over U, V is a set of endogenous variables and F is a set of functions that determines V. The *SCM M* can be used to represent a specific RL environment, and the agent learns a representation of the *SCM M* which could be in the form of a causal graph [4]. The aim is that the causal graph captures the invariances of the *SCM M* that could be transferred to different environments or tasks with similar underlying causal structures.

The agent can obtain different amounts of information about the *SCM M* through the three different types of interactions. The three types of interactions are observation or association (seeing), intervention (doing or experimenting) and counterfactual (imagining and introspection). The Pearl Causal Hierarchy (PCH) [4] is named after Judea Pearl for his discoveries presented in [37]. The PCH categorizes the three types of interactions into different levels as shown in Figure 2.4. In level 1, the agent observes an output value y for a given input value x. In level 2, the agent performs some action x in some context or state c and observes a resulting output value y. In level 3, the agent tries to imagine the outcome y given a particular intervention x, and the agent does this by using what it has learnt from other actions x' and outcomes y'. Interactions in higher levels provide more information about the *SCM M*. The Causal Hierarchy Theorem (CHT) [4] states that the three levels of causality cannot collapse, and they remain distinct. The CHT implies that agents who learn using lower-level interventions cannot make higher-level inferences like counterfactuals. The agent needs to use level 3 counterfactuals to learn the Structural Causal Model.

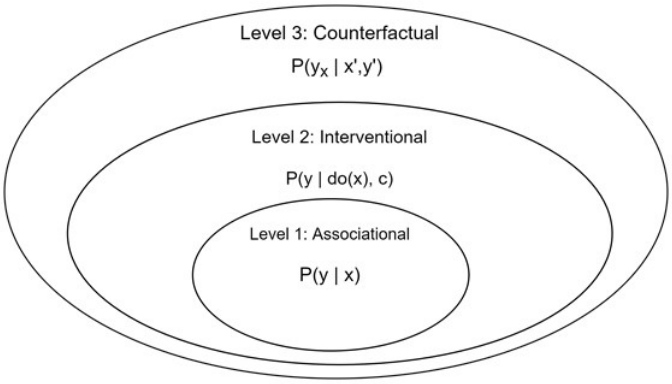


Figure 2.4: Interactions categorised by the PCH, adapted from [4]. Lower level interactions under determines higher level interactions.

Table 2.2 presents the taxonomy of models. Statistical models can only predict in an identically distributed data (i.i.d) setting, and it can be learnt from level 1 observational data alone. Many of the Deep Learning models are Statistical models, whether trained under supervision or not. Deep Learning models often fail to generalize when deployed in real-world environments. The real-world environment often shifts slightly from the training distribution, or there might be out-of-the-training distribution scenarios. Mechanistic models capture the physical mechanisms with equations [7]. Equations of physics can be used to infer the distance that a ball would fly when the initial angle is changed (intervened), and it can predict the behaviour of objects in outer space (counterfactuals). Structural Causal Models (SCMs) provide an abstraction of the physical models, and it is closer to the model that humans use to learn and understand the world around them. A child learns that the ball will move if they kick it without understanding Newton's third law of motion. Current Reinforcement Learning models are somewhere between the SCM and the Statistical models. The RL agents learn by performing different actions (interventions) in different states. When a robot picks up a block, the block is no longer affected by the friction between the block and the ground. However, the RL agents do not use causal models to capture the causal mechanisms, so they fail to generalize well.

Table 2.2: Taxonomy of models, adapted from [7]. The models capture the different amounts of information about an environment and can therefore make different levels of prediction.

Model	**Predict in i.i.d setting**	**Predict under distr.shift/ intervention**	**Answer counterfactual questions**
Mechanistic/ physical	yes	yes	yes
Structural Causal	yes	yes	yes
Statistical	yes	no	no

The Independent Causal Mechanism (ICM) principle [7, 38, 39] defines causal mechanisms to be "autonomous modules that do not inform or influence each other." The Sparse Mechanism Shift (SMS) hypothesis is based on the "influence" constraint of the ICM principle. The SMS hypothesis implies that interventions on causal variables "tend to manifest themselves in a sparse or local way" and "they should usually not affect all factors simultaneously." The ICM and SMS provide an idea of how RL agents could use interventions to discover causal mechanisms. [7] provides the conditions that explain the ICM, where PA_i are the causes or the parents of X_i.

1. Changing (or performing an intervention upon) one mechanism $P(X_i|PA_i)$ does not change any of the other mechanisms $P(X_j|PA_j)(i \neq j)$ [39].
2. Knowing some other mechanisms $P(X_i|PA_i)(i \neq j)$ does not give us information about a mechanism $P(X_j|PA_j)$ [40].

2.4.1 Causal RL

Causal RL [13] is a combination of Causal Inference [10, 11, 12] and RL. The RL agents mostly only partially observe real-world environments. Model-based RL provides the agents with the ability to reason and introspect. However, it is not always feasible to build a model that accurately captures all the dynamics of the environment to train from it. Model-free RL and a hybrid model (Dyna-Q) are usually used for environments that are POMDP. The hybrid model provides the advantages of both model-based RL and model-free RL. However, an additional training phase is still required for obtaining the model, and the model does not capture any causal mechanisms that could be transferable. SCMs provide similar advantages to a hybrid model, and it captures the transferable causal mechanisms that would make the RL agent more robust. The SCMs of partially observable environments can be seen as a jigsaw puzzle. The different levels of interactions (Figure 2.4) can provide different pieces of the whole puzzle.

Interventions are essential to help agents uncover the SCM. However, not all interventions provide the same benefits for the agents. [13, 41] provided two properties that describe where interventions should be performed to provide the most causal information and efficiency. The first property is Interventional Equivalence, and it defines a Minimal Intervention Set (MIS). Interventional Equivalence implies that no interventions and interventions on one or a small subset of variables are better than interventions on all the variables. Interventions on all variables with all the possible combinations are not MIS as there will be interventions that provide the same (equal) effect. Also, the number of combinations for all the variables will grow exponentially, making it inefficient. [6] also confirms this result with the agent trained under curriculum 2 (random interventions on all variables) "failed to learn any relevant skill" and "would need more data and optimization

steps." The second property is Partial-Orderedness, and it defines a Possibly-Optimal MIS (POMIS). Partial-Orderedness implies that interventions on a small subset of variables are better than no interventions. Interventions (doing) provide more information about the SCM than just observing (seeing). [6] confirms this result with agents trained under curriculum 1 (intervention on some variables) being more robust than agents trained under curriculum 0 (no interventions).

Level 3 counterfactuals are needed to learn the SCM. [13, 42] defines what level 3 counterfactuals are and a method for how observations, interventions and counterfactuals could be used to learn the SCM with unobserved confounders like in many real-world scenarios. Confounders could be the noise or some unobserved variable that affects the causal mechanisms and the output. Interventions can be written in the same notation as counterfactuals, in $E(Y_X = x)$. Interventions can be called level 2 counterfactuals. In order to understand what level 3 counterfactuals are, consider the following scenario where the agent intends to perform an action x_0 (following a policy) but realizes its intent and imagines the possibility of another action x_1. Interventions and counterfactuals can be distinguished by what the agent does at this step. There are three decisions that the agent can make:

- The agent does not interrupt and plays x_0 (intervention).
- The agent interrupts and plays x_1 (intervention). $\mathrm{X} = \mathrm{rand}() = x_1 \rightarrow P(y|do(x_1))$.
- The agent interrupts and plays x_0, but it also imagines the outcome if it had played x_1 (counterfactual). $\mathrm{X} = \mathrm{rand}() = x_1|x_0 \rightarrow P(Y_{x_1}|x_0)$.

The agents use previous interventions and counterfactuals that it has performed to imagine the outcome of the current counterfactual question. It shows that interventions might be necessary, but it is not sufficient for learning an SCM with unobserved confounders.

2.5 Causal RL in Robotics

RL agents use a large number of trials and errors to learn tasks. The training process of an RL agent will be expensive to perform with a real-world robot in a real-world environment. Physical hardware degrades over time, and the policy of the RL agent will need to be adjusted to produce the same controls. Engineers will need to create a dedicated environment for training the robot to ensure the safety of other humans and expensive pieces of equipment. Solutions that reduce the cost of implementing RL for robotics involve reducing the number of trials and errors needed to learn or reducing the trial and error cost.

Imitation learning can help reduce the number of trials and errors by providing robotic agents with prior expert knowledge. In Imitation learning, a human or another expert

teacher robot can demonstrate how to perform a specific task, and in some cases, the human might physically guide the robotic agent to perform a task. The visual and/or physical feedback guides the robotic agent closer to the optimal or near-optimal solution. Therefore, the robotic agent will require less trial and error to reach the desired optimum. However, this requires another human or robot to master a specific task first, and it limits the robots to solve tasks that are already solved.

Another solution for reducing the cost of implementing RL for robotics would be reducing trial and error costs. The solution is simulation-to-real (sim2real) transfer [43]. The cost of training an agent in a simulation is significantly lower than the cost of training the agent in a real-world environment. The only major cost for using a simulation is the initial cost of building the simulation. The simulation has to approximate the real-world deployment scenario to a certain degree for the simulation to be useful. If the agents receive different inputs (observations) and produce different outputs (control actions) in the simulation compared to when deployed, there will also be a high transfer cost for sim2real transfer. Traditional DL and RL methods often overfit the training environment and fail to generalize to real-world deployment scenarios. Existing methods [43] that help solve the sim2real transfer issues are zero-shot transfer, domain randomization [44] and adaptation [45], curriculum learning [29, 46] and incremental environment complexity [47]. The simulation environment and the Causal RL solutions discussed below incorporate some of the methods listed.

2.5.1 CausalWorld

Popular simulation environments used for robotics include Gazebo, PyBullet and Mujoco. Gazebo is the most popular simulation environment due to its integration with the Robot Operating System (ROS). ROS is used in the software programs present in many real robots. However, Gazebo is not integrated with DL and RL libraries. PyBullet and Mujoco are more suitable for DL and RL applications in robotics. However, performing interventions on different causal variables will be difficult in PyBullet and Mujoco as they were not created to support Causal Learning.

CausalWorld [6] is a benchmark created to propel research in Causal and Transfer learning for robotics. CausalWorld provides a simulation environment (powered by PyBullet) for robotic manipulation tasks based on the open-source TriFinger robot. Sim2real transfer is easy to perform as CausalWorld is a realistic simulation of TriFinger and real-world physics. Causal World reduces the cost of trials and errors.

Tasks available in CausalWorld

Different tasks are available for training and testing the RL agent. Tasks range in difficulty as shown in Figure 2.5. Pushing and Picking deals with one object (cube) by default, and there are no objects involved in Reaching. In Reaching, the RL agent has to reach a goal end-effector position with the three fingers of the TriFinger robot. The RL agent needs to push the block into the goal position and orientation for the Pushing task. The Picking task requires the RL agent to pick up the block to a goal height. The tasks then increase in difficulty with the Stacking2 and the Pick and Place task. Both of these tasks deal with two objects, and the RL agent has to use skills that it should have learnt in the three Easy tasks. The RL agent needs to pick the block over an obstacle and place it in a goal position and orientation for the Pick and Place task. Stacking2 requires the RL agent to stack two blocks on top of one another in a goal position, and the RL agent needs to start taking into consideration the stability of the stacked blocks.

There is a significant increase in difficulty from the Medium to the Hard tasks. The RL agent must now be able to generalize to different amounts of objects that could occur. A random number of blocks are dropped onto the stage in the General task. The resulting position and orientation of the blocks are set as the goal. The RL agent needs to orientate the blocks into the goal shape. Towers, Stacked Blocks and Creative Stacked Blocks are similar, and they all involve stacking blocks into a goal shape. All the blocks are the same size and shape in Towers, whereas in Stacked Blocks, the size and shape of the blocks can vary. Creative Stacked Blocks builds upon the Stacked Blocks task. In Creative Stacked Blocks, only a subset of the blocks needs to be used to build the goal shape. The RL agent needs to figure out which blocks to use and the stability of the resulting configuration. The user may also design custom tasks and objects to be used in CausalWorld.

The different tasks require some common skills and have common causal factors. The tasks in CausalWorld have similar causal structures, and the causal knowledge obtained from one task can be transferred to another. The RL agent can be trained using the Easy tasks and advance to more complex tasks depending on the agent's performance in testing. This method is called incremental environment complexity, and existing results [47] show that it reduces the gap in sim2real transfer and it improves the agent's performance in the real world.

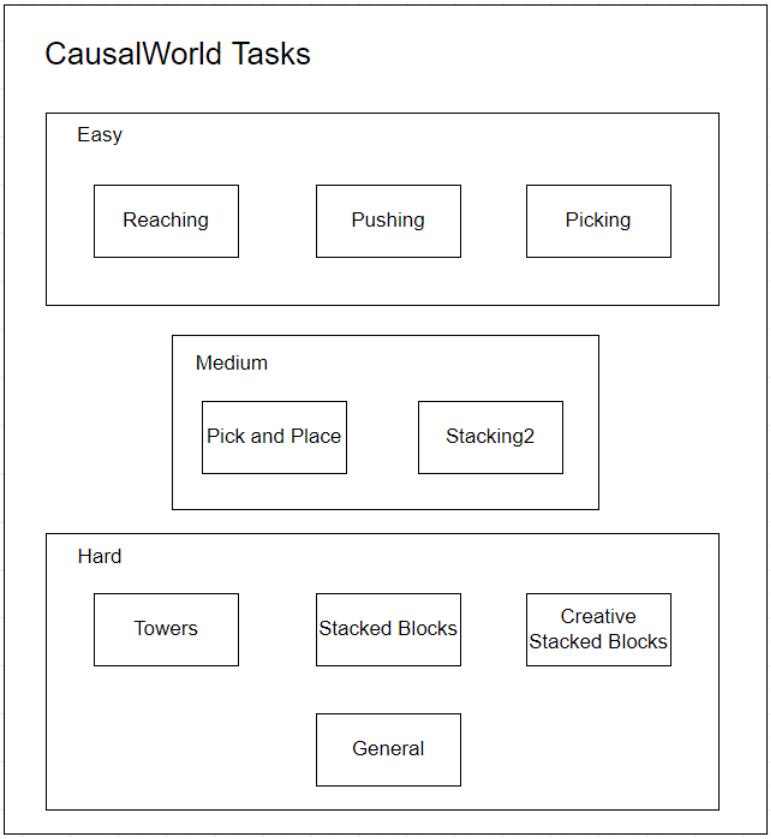


Figure 2.5: Classification of the tasks provided in CausalWorld by the level of difficulty.

Intervention on variables in CausalWorld

Domain randomization is another method for helping with sim2real transfer. Domain randomization creates perturbations to different variables of the RL environment and allows the RL agent to generalize better. Interventions are another way of performing domain randomization in Causal RL. The different variables that can be intervened on in CausalWorld are provided in Table 2.3. Space B extends the range of values that the variables can take with Space A. For example, the range of values for gravitational acceleration will be from -10 m/s^2 to -7 m/s^2 in Space A, but the range can be extended from -10 m/s^2 to -4 m/s^2 with the inclusion of Space B. Space A and B can be used to test the robustness of the RL agent. The RL agent could be trained in Space A and tested in Space B. Color is described using RGB values, and the three values can take the range from 0.0 to 1. Size and positions are described for the three axes: x,y and z. Size and mass values follow the SI units, m and kg, respectively.

Interventions on different variables can be easily performed in CausalWorld. Interventions can be defined precisely with intervention actors in CausalWorld and be used for Curriculum learning. Section 2.4 discusses how interventions can help the RL agent obtain more information about the underlying SCM and improve its performance and robustness.

Table 2.3: Table shows the variables that can be intervened in CausalWorld. Space A and B are the ranges of values for the different variables. Table is taken from [6].

Variable	Sub Variable	Space **A**	Space **B**
gravity[z]	-	$[-10, -7]$	$[-7, -4]$
floor friction	-	$[0.3, 0.6]$	$[0.6, 0.8]$
stage friction	-	$[0.3, 0.6]$	$[0.6, 0.8]$
stage color [rgb]	-	$[0.0, 0.5]^3$	$[0.5, 1]^3$
floor color [rgb]	-	$[0.0, 0.5]^3$	$[0.5, 1]^3$
joint positions	-	$[[-1.57, -1.2, -3.0]^3,$ $[-0.69, 0, 0]^3]$	$[[-0.69, 0, 0]^3,$ $[1.0, 1.57, 3.0]^3]$
block	size	$[0.055, 0.075]^3$	$[0.075, 0.095]^3$
block	color	$[0.0, 0.5]^3$	$[0.5, 1]^3$
block	mass	$[0.015, 0.045]$	$[0.045, 0.1]$
block	position (cylindrical)	$[0, -\pi, h/2],$ $[0.11, \pi, 0.15]]$	$[0.11, -\pi, h/2],$ $[0.15, \pi, 0.3]]$
goal cuboid	size	$[0.055, 0.075]^3$	$[0.075, 0.095]^3$
goal cuboid	color	$[0.0, 0.5]^3$	$[0.5, 1]^3$
link	color	$[0.0, 0.5]^3$	$[0.5, 1]^3$
link	mass	$[0.015, 0.045]$	$[0.045, 0.1]$

2.5.2 Methods for Causal RL

Previous work has been limited to applying Causal Inference to tasks where the underlying causal model is already known or partially known [14, 15, 16]. This requirement is not feasible for many real-world tasks. Work that involves learning the underlying causal model through the use of interventions and counterfactuals is either not applied in RL (CoPhy [5]) or they are not applied to a complex real-world task like robotics [20]. However, CoPhy provides an interesting mechanism that may enable RL agents to perform counterfactuals or imagine causal effects. [19] applies Causal RL to a real-world task, and the RL agent learns in a real-world environment instead of a simulation. However, it does not use any causal representation, and the interventions and counterfactual questions are human-guided and collected offline. Causal Curiosity [18] does not make use of any counterfactuals, but it does provide an exciting approach to Causal Learning and an interesting causal representation.

Causal Curiosity

The Causal Curiosity [18] solution has two main phases or stages as shown in Figure 2.6. Causal Curiosity was applied in CausalWorld. In the recursive training phase, the aim is to learn a CEM MPC (Cross-Entropy Method Model Predictive Controller) where K actions can be sampled, and each of these actions provides information about one causal factor. This goal is achieved through maximizing the Curiosity reward. The Curiosity reward is inspired by algorithmic information theory. The assumption is that the amount of information in a given observation is proportional to the number of causal factors that caused the observation. When the Curiosity reward is maximized, the amount of information in the observation is minimized to one causal factor.

There are K recursive layers to learn K actions in the training phase. A sequence of actions

is sampled from the CEM MPC and applied to a set of environments. The resulting observations are then passed through a k-means soft dynamic time warping clustering model (km_sdtw). The km_sdtw classifies the observation into two clusters, and the binary labels form the Causal representation (CausalRep). The binary clusters divide the environments, for example, 1 for environments where the object could be rolled (sphere) and 0 for where the object cannot be rolled (cube). The observations and the CausalRep are used to calculate the Curiosity reward, which is then used to optimize the MPC. The algorithm then recursively goes through the environments that were labeled 1 and 0. A trained CEM MPC and km_sdtw are obtained at the end of the training phase and used in the inference phase.

In the inference phase, K actions are sampled from the trained CEM MPC and applied to the RL training environment. The km_sdtw clusters the resulting observation and produces the CausalRep. The CausalRep is then concatenated to all the observations to train the RL agent. The RL agent maximizes the external reward.

The main drawback of the Causal Curiosity solution is that it does not make use of Counterfactuals. As it is stated in Section 2.4, Counterfactuals are required for the agent to learn the complete underlying SCM. CoPhy provides an interesting solution for performing Counterfactuals.

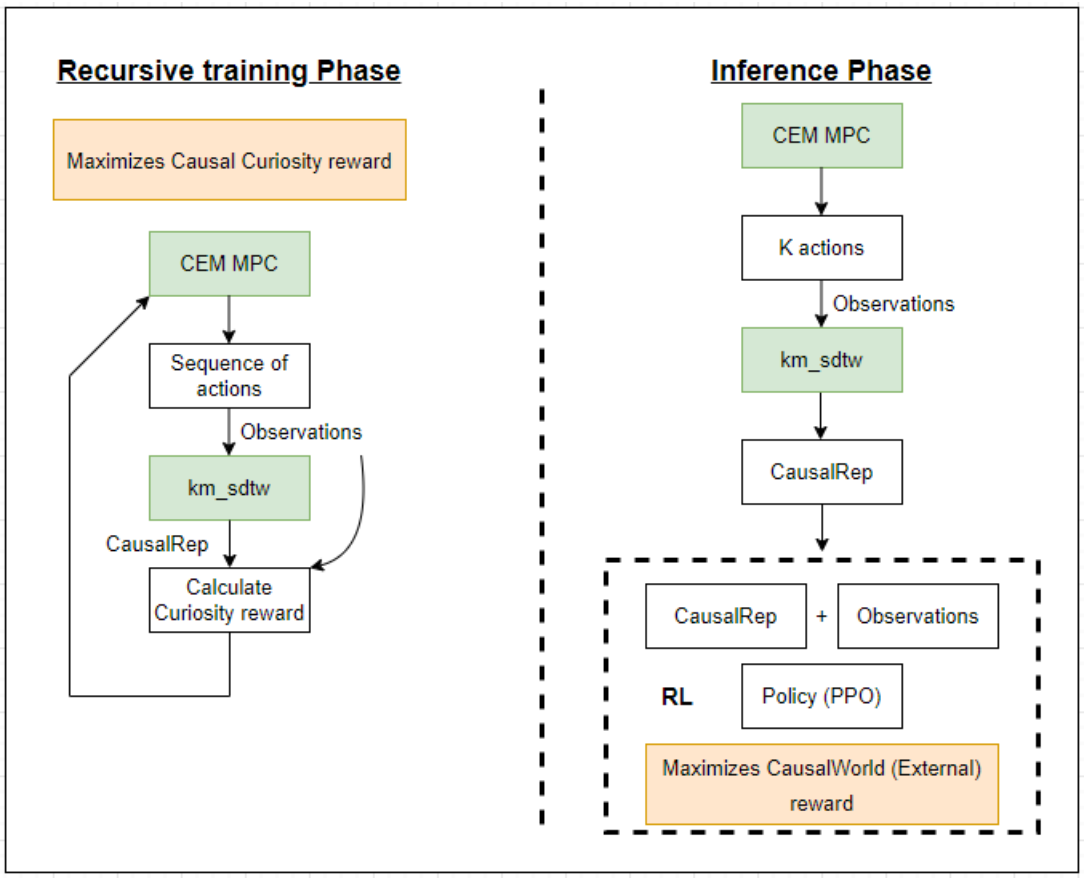


Figure 2.6: Causal Curiosity is divided into the recursive training phase and the inference phase. Recursive training phase maximizes the Curiosity reward and the Inference phase maximizes the external reward. The green blocks are the components that are learnt and transferred from the training phase to the inference phase.

CoPhy

Counterfactual learning of Physics dynamics (CoPhy) [5] is a DL solution applied to predict the position, orientation and stability of a given configuration of blocks for multiple time steps into the future. The overall architecture used is presented in Figure 2.7. Observations AB for τ time steps is used for training the entire model from end to end. The latent representation U or the CausalRep is the embedding in the last layer of the RNN. An intervention is made on observation A (the initial state or X_0), and the result is observation C. CausalRep is concatenated to all the observations at each time step and passed through the different blocks of the DL model to make the predictions. Predictions are made for $\tau - 1$ time steps as the initial observation C is given. The de-rendering blocks in Figure 2.7 are used to convert the pixel observations into latent representations. The graph convolutional network blocks (GCN) then add contextual information to the latent representations. Finally, the RNN temporally integrates the GCN blocks. CoPhy is a benchmark created for Causal Inference in DL.

The CoPhy solution could be adapted and be used as a mechanism that enables RL agents

to imagine or make Counterfactual predictions. RL agents could explain the choices of actions by providing the possible future outcomes of different states and actions. The predictions are based on the agent's understanding or abstract representation of the underlying SCM. An interesting idea is inspired by both Causal Curiosity and CoPhy. The idea is that causal representation does not necessarily have to be in the form of a causal graph. RL agents may learn an abstract representation of their understanding of the environment just as humans do. An interesting future direction might be investigating how RL agents could generate a causal graph to represent an underlying causal model through the learnt abstract causal representation. Causal graphs are a powerful representation that provides explainability to humans.

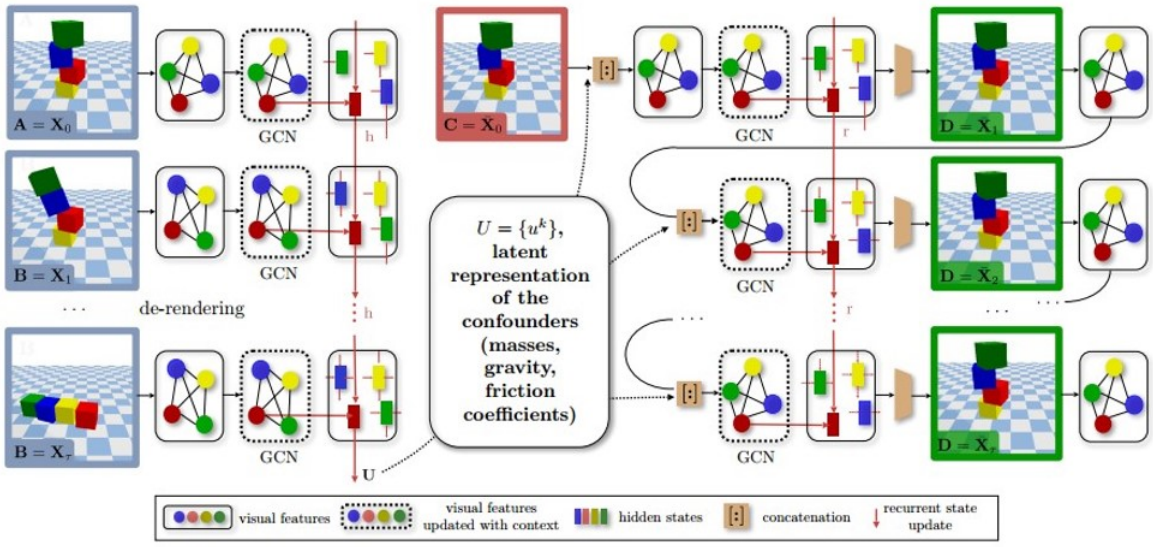


Figure 2.7: Overall architecture of the CoPhy solution. Taken from [5].

In Causal RL, agents learn causal factors that are invariant across domains. Causal RL allows agents to adapt their causal knowledge to new tasks and domains (domain adaptation). The transferability of causal knowledge also helps improve zero-shot transfer for RL agents. All these properties help solve sim2real transfer. This paper makes use of ideas from CoPhy [5] and Causal Curiosity [18] to create a new Causal RL solution that is applied to complex real-world robotic tasks in CausalWorld.

2.6 Summary

This chapter identified the need for Causal RL and provided the background needed for understanding the CausalCF solution. Reinforcement Learning (RL) and Deep Learning (DL) was introduced in Sections 2.1 and 2.2. Neural networks can be used as the value function, policy, or model for improving the scalability of an RL agent, and this combines the two fields to form DRL. The explainability and robustness of DRL solutions are essential for

people to use or trust the decisions made by the DRL models. Section 2.3 provides the different approaches that have been used to try to achieve Explainable AI (XAI) and also highlights the need for Explainable RL (XRL). A promising approach was combining Causal Inference and RL (Causal RL) to make RL more explainable and robust. Causal Inference is the study of Causality, and the theory is presented in Section 2.4. Section 2.5 discusses the limitations of previous work in applying Causal Inference for RL and the challenges of applying RL for robotics. This thesis proposes a new solution called CausalCF which will help overcome previous limitations and challenges of RL.

RL's limitations and challenges include the poor generalization capabilities of RL algorithms (robustness) and the cost of training and testing an RL solution for a robotic application. CausalCF will combine ideas from Causal Curiosity and CoPhy and use the tools and techniques provided by CausalWorld. Causal Curiosity uses interventions and provides an approach for training the RL agent with a causal representation. The causal representation captures invariant causal mechanisms with interventions that help improve the robustness of RL. However, as discussed in Section 2.4.1, counterfactuals are needed to learn about the complete underlying SCM. Counterfactuals allow the agent to capture more causal mechanisms and improve the robustness. CoPhy provides a DL solution that can perform counterfactuals. CausalCF will adapt CoPhy to work for RL. CausalWorld employs techniques like domain randomization, curriculum learning and incremental environment complexity for overcoming the challenges of applying RL for robotics. CausalWorld also provides an evaluation pipeline for testing the robustness of RL solutions. CausalCF is intended for a wide range of real-world robotic applications. CausalCF provides an approach for how interventions, counterfactuals and causal representations could be used for RL robotic applications in the real world. This thesis uses the CausalWorld benchmark to evaluate if CausalCF improves the robustness of RL, which corresponds to the feasibility of CausalCF for real-world applications. The next chapter will describe precisely how the Causal Curiosity and CoPhy solutions are combined into a new Causal RL solution (CausalCF) and adapted for training and testing in CausalWorld.

3 Design

This chapter describes the design of the different components of the proposed solution (CausalCF) and how they are integrated. CausalCF stands for Causal counterfactuals. CausalCF is the first Causal RL solution that performs level 3 counterfactuals (defined in Chapter 2 Section 2.4.1). CausalCF combines ideas from CoPhy [5] and Causal Curiosity [18]. Causal Curiosity provides an approach for using a causal representation and interventions to train RL agents. Interventions provide information about the underlying SCM and help the causal representation capture the causal mechanisms. The causal representation can then be transferred to different tasks with similar causal structures and improve the robustness of RL. However, Causal Curiosity does not use any counterfactuals. Level 3 counterfactuals can provide more information about the causal mechanisms and further improve RL's robustness. CoPhy provides a mechanism for performing counterfactuals in a DL application. CausalCF adapts CoPhy for RL and enables RL agents to perform counterfactuals. It will be the first time CoPhy is adapted for use in RL. Chapter 5 will evaluate if CoPhy still provides the same benefits after it is adapted for RL. CausalCF is designed to work for CausalWorld. CausalWorld is a realistic simulation environment, and it allows for useful interventions to be easily performed. The tasks provided in CausalWorld also have similar causal structures, which allows for further evaluation of the robustness of the Causal RL solution. Section 3.1 describes how the task of robotic control and object manipulation is modeled as an RL task in CausalWorld. Section 3.2 presents an overview of CausalCF and describes the ideas that are taken from CoPhy and Curiosity. Section 3.3 and 3.4 describes all the modifications that were made to the CoPhy solution and CausalWorld. Section 3.5 provides further details of the fully integrated solution.

3.1 RL for CausalWorld

CausalWorld [6] was explicitly designed for research in causal and transfer learning in RL. This section will describe the state space, action space and the design of the rewards used in CausalWorld.

3.1.1 Observation space

There are two types of observations that can be used. Pixel observation provides a total of six images that shows the current state of the environment at 60, 120 and 300 degrees and the corresponding view of the environment if the goals are achieved, as shown in Figure 3.1. CausalWorld provides another type of observation called Structured observation that presents the observation in the form of a vector, as shown in Figure 3.2. The R1, R2 and R3 parameters represent the robotic agent's current state. There is a scene feature for each object in the environment. The Object feature describes the current state of the block, and the Partial goal feature describes the desired end state of the block. The Object features are concatenated first, and the Partial goal features are concatenated in the same order at the end. The block's size, position, orientation, and velocity are defined for axes: x, y and z. Orientation has an additional angle parameter.

Structured observation is used to train the proposed Causal RL solution in this paper. Structured observation makes experimenting with the feasibility of the solution easier. Pixel observation is more challenging as the RL agent can only observe the environment partially, and an extra module is needed to convert the image into a meaningful representation for the agent to train. Future directions might involve using Pixel observations as it is more representative of real-world scenarios. However, the highest priority was to test the feasibility of the Causal RL solution in complex robotic tasks. Structured observation can directly be used as an input to the policy of the RL agent for deciding which actions to take next.

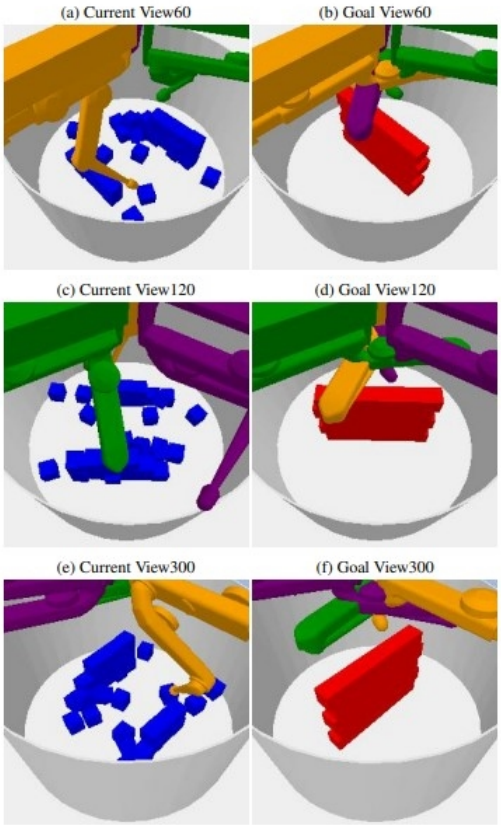


Figure 3.1: The figure provides an example of the six images that is provided with pixel observation. Taken from [6].

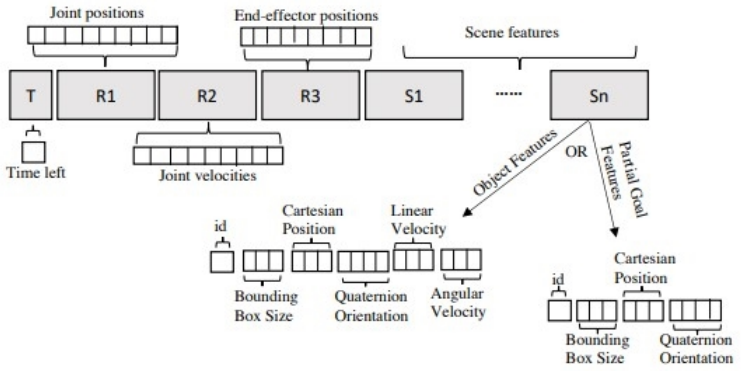


Figure 3.2: The diagram shows the different parameters or information that is provided in the Structured observation. Taken from [6].

3.1.2 Action space

The robot can be controlled through three different modes. The modes are joint positions, joint torques and end-effector positions as indicated with the R1, R2 and R3 parameters in Figure 3.2. The action is specified for each of the three fingers on the robot. CausalWorld converts the action specified by the vector into motor actions that control the robot. There

are constraints set on the values of the action parameters based on the physical properties of the robot. In this paper, the RL agent uses joint positions to control the robot. End-effector positions are non-deterministic for the current version of CausalWorld, and joint positions are more intuitive to use than joint torques.

3.1.3 Rewards

The consecutive actions of the Causal RL agent are chosen to maximize the cumulative rewards in CausalWorld. Rewards are defined differently for different tasks. Rewards are a sum of different functions. Each function aims to affect the agent's behavior in a particular way. The reward weightings of each function can be specified. RL agents would then prioritize to maximize the functions with larger weightings. Table 3.1 presents the dense rewards that were used in the different tasks for evaluating CausalCF. The weightings used for the tasks are the same as those that were used in the CausalWorld paper [6].

The weightings used for a task determine the agent's priorities in that task. The Pushing task uses weightings of [750, 250]. The RL agent prioritizes getting close to the blocks first and then aims to move the blocks into their goal positions. The Picking task has additional x,y and z subscripts as the agent have to now pick the block up to its desired height. The purpose of Δ_3^t is to make the agent move slower when the block is picked up and to make the actions smoother. Stacking2 deals with two objects and therefore has additional number subscripts. Stacking2 uses different reward mechanisms based on the current state of the environment. The agent first moves the bottom block into the desired goal position and orientation. For the block that is stacked on the top, the agent picks the block up to its goal height first and then places it in the goal position (x,y). Dense rewards are calculated at each time step. Dense rewards help with the convergence of RL algorithms as the agent can receive feedback from the environment for every action taken.

Table 3.1: Table presents the dense rewards that were used for the different tasks. Δ_1^t calculates how close the fingers are to the block. Δ_2^t calculates how close the block is to the goal. Δ_3^t calculates how much faster or slower the joints are moving. Δ_4^t calculates how much closer the block is getting to its goal. Adapted from [6].

Tasks	**Dense rewards**
Pushing	$-750\Delta_1^t - 250\Delta_2^t$
Picking	$-750\Delta_1^t - 250\Delta_{2,z}^t - 125\Delta_{2,x,y}^t - 0.005\Delta_3^t$
Stacking2	$1_{\delta_{1,1}^t > 0.02}(-750\Delta_{1,1}^t - 250\Delta_{2,1}^t) + 1_{\delta_{1,1}^t < 0.02}(-750\Delta_{1,2}^t - 250\Delta_{4,2,z}^t) - 1_{\delta_{2,2,z}^t > 0}(125\Delta_{2,2,x,y}^t - 0.005\Delta_3^t)$

3.1.4 Deep RL algorithms

Soft-Actor Critic (SAC) [48], Proximal Policy Optimisation (PPO) [49] and Twin-delayed deep deterministic policy gradient (TD3) [50] are the current state of the arts model-free RL algorithms. All these algorithms improved on the Q-Learning algorithm. The CausalWorld paper [6] evaluated the three state-of-the-art methods across four different tasks and three curriculums. The results were averaged across five random seeds, and the models' performance was stable. PPO required approximately ten times more time steps in training to converge than SAC and TD3. SAC converged to a better performance than TD3 in most tasks and generalized better than TD3.

The hyperparameter settings used for SAC will be the same as the ones used in [6]. The policy will be a two-layer MLP (256, 256). Table 3.2 presents the hyperparameters used for the SAC algorithm.

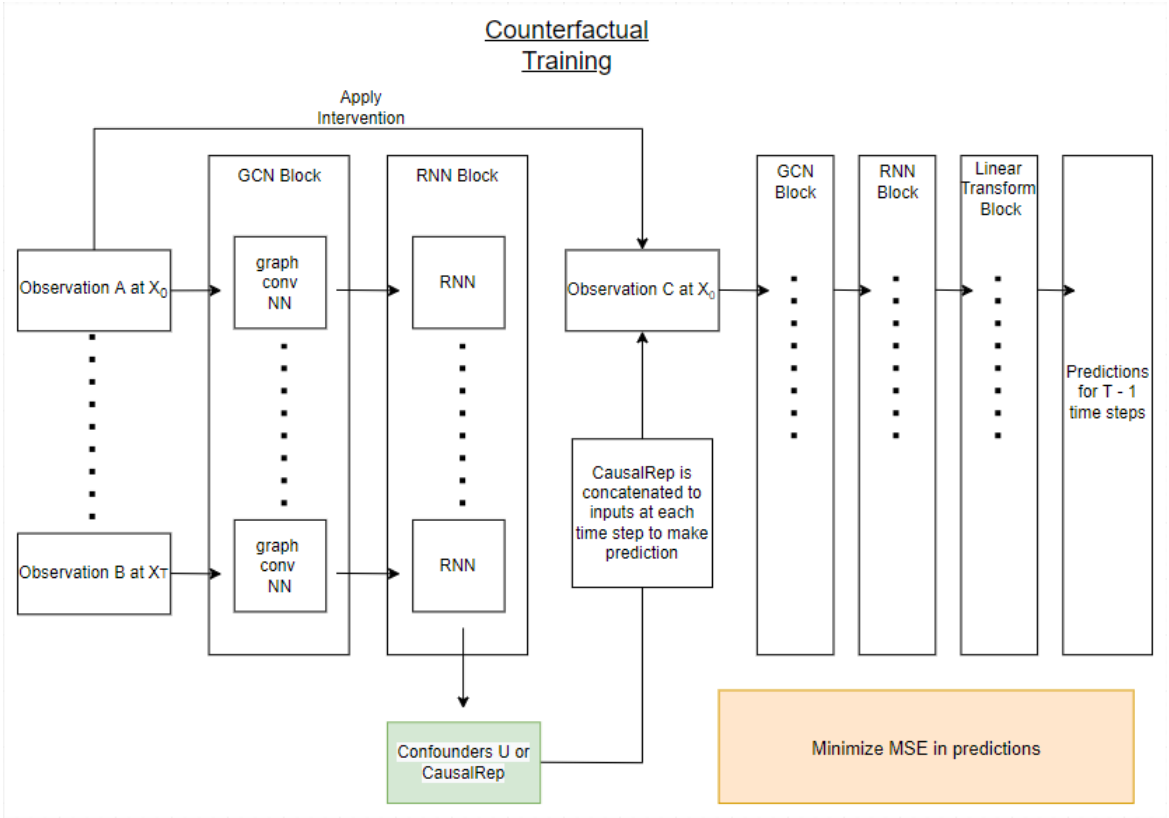
Table 3.2: Parameter configuration for the SAC algorithm.

Parameters	Value
gamma	0.95
tau	1e-3
ent_coef	1e-3
target_entropy	auto
learning_rate	1e-4
buffer_size	1 000 000
learning_starts	1000
batch_size	256

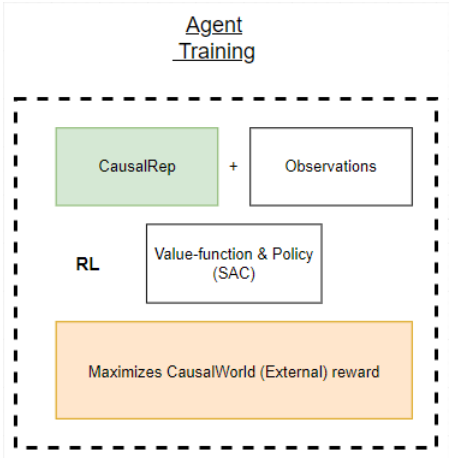
3.2 Overview Design of CausalCF Solution

The Causal RL solution proposed (CausalCF) combines CoPhy and Causal Curiosity. CausalCF has two main phases, Counterfactual training and Agent training, as shown in Figure 3.3. The aim of Counterfactual training (Figure 3.3a) is to obtain a useful causal representation that can be concatenated to the observations to train the RL agent. The RL agent then aims to maximize the CausalWorld reward in Agent training (Figure 3.3b). The CoPhy solution was adapted and trained from end to end in Counterfactual training. The idea of using two training phases and concatenating the causal representation to all the observations used for training the agent is inspired by Causal Curiosity. CausalCF has an additional component where it iterates between Counterfactual and Agent training. The iteration between the two training phases allows the agent to update the causal representation or its *causal knowledge* the better the agent gets. The extra iterations for

Counterfactual training will also allow the agent to capture more information about causal mechanisms.



(a) Counterfactual training. This modified version of CoPhy is called Counterfactual model (CF_model).



(b) Agent training

Figure 3.3: Overview design of the CausalCF solution. CausalCF has two phases of training like in Curiosity. The diagram in the Counterfactual training is adapted from the CoPhy DL model [5].

3.3 Modifying CoPhy for CausalWorld

Figure 3.4 provides an overview design of the old CoPhy solution. The RGB pixel images were 224x224. B is an extra dimension that was added when the tensor was processed. T is the number of time steps and K is the number of objects. The CoPhy solution is adaptable to varying amounts of objects and time steps. The final output of the solution was the x, y and z positions of each object for T-1 time steps and the stability (1 or 0) of each object at each time step.

The Derendering component can be removed for CausalCF as Structured observations are used. Structured observation only provides features of objects that are present in the environment. Therefore, CausalCF removes presence_ab and presence_c and modifies all the dependent processes. Counterfactual training also aims to train a model that can predict the Structured observations at each time step by minimizing the mean squared error (MSE) for the predictions (Figure 3.3a). Modifications had to be made throughout the entire architecture to achieve the goal of Counterfactual training.

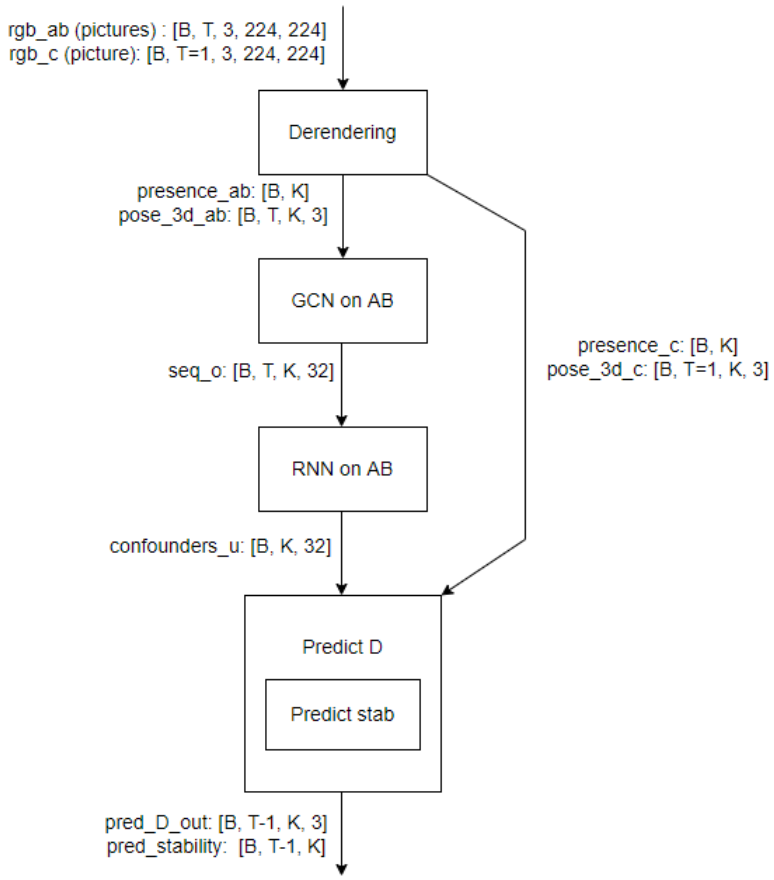


Figure 3.4: Overall flow of the old CoPhy solution showing the inputs and outputs of each component and the corresponding sizes.

The CoPhy solution had to be modified for use in CausalWorld, and Figure 3.5 provides an

overview design of the modified version of CoPhy (CF_model). Below presents a list of the variables used in Figure 3.5 and what they represent:

- n: number of objects the RL agent can manipulate
- m: number of objects that are fixed in place
- T: number of time steps
- K: total number of objects = n + m
- B: B is an extra redundant dimension added during the process of appending and stacking outputs of each time step or object.

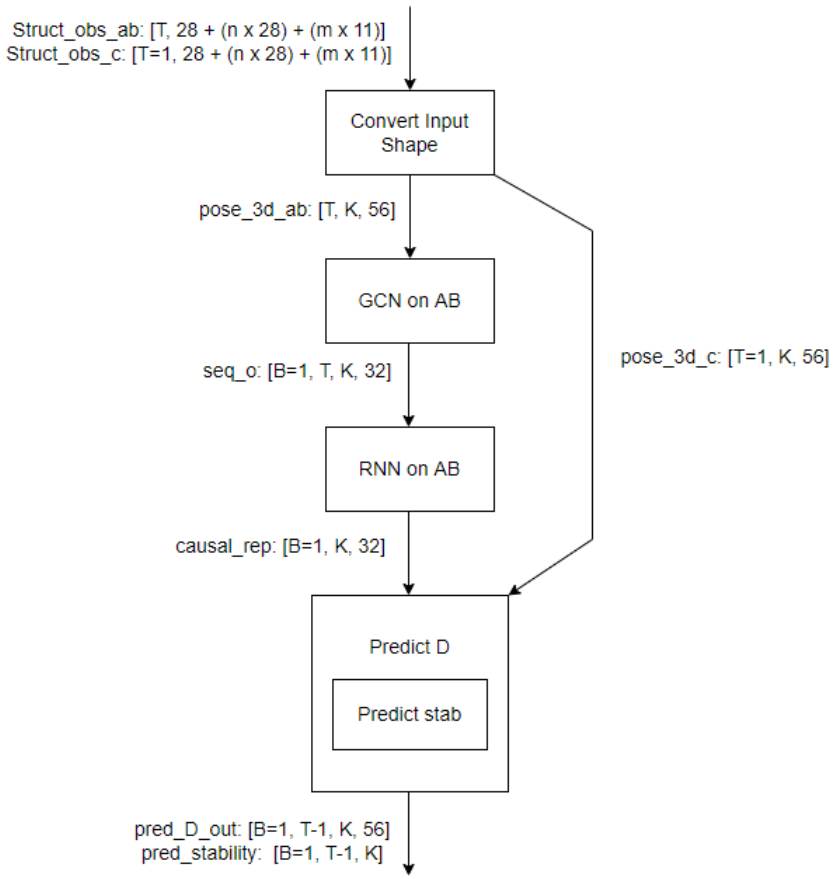


Figure 3.5: Overall flow of the modified CoPhy solution for CausalCF showing the inputs and outputs of each component and the corresponding sizes. The modified version of CoPhy will be referred to as CF_model.

Each of the CoPhy components has a set of NN sub-modules. The input sizes of all the NN sub-modules had to be modified to suit the new input shape. Table 3.3 presents the set of NN sub-modules that belong to each of the components in the CF_model. The mlp_inter type sub-modules are for computing the interactions between the objects. The Predict D component has "_delta" at the end of all the sub-module names because the sub-modules

predict the changes or Δ in the observations at each time step. The new observations at each time step are computed by adding the changes to the observation in the previous time step. Chapter 4 will provide more details on the definitions of the sub-modules.

Table 3.3: The components of the CF_model with the corresponding set of NN sub-modules that belong to each component.

Component	NN sub-modules used
GCN on AB	mlp_inter and mlp_out
RNN on AB	rnn
Predict stab	mlp_inter_stab and mlp_stab
Predict D	mlp_inter_delta, mlp_gcn_delta, rnn_delta and fc_delta

3.3.1 Convert Input Shape

The CoPhy solution was adaptable to varying amounts of objects and time steps, and the input shape contributes to this property. The goal of the component described in this section is to convert the observations into an input shape that the CF_model can process, and it is adaptable to varying amounts of objects and time steps. Figure 3.6 provides a representation of the desired input shape to be processed by the CF_model. The size of the last dimension of the input shape determines the input size of all the NN sub-modules. Therefore, the last dimension of the input shape is fixed. The last dimensions for pose_3d_ab and pose_3d_c in the CoPhy solution are fixed to 3 for each block's x, y and z positions. Structured observation (Figure 3.2) provides details about the time left, states of the robot and the states of all the objects. The feature dimensions of Structured observation are higher than that of the original CoPhy input. Obstacles or fixed blocks only have Partial goal features as the state will always remain the same for these objects. Converting Structured observation into the desired tensor shapes for pose_3d_ab and pose_3d_c is not straightforward.

A few assumptions and inferences need to be made to convert the Structured observations into the desired input shape. An action can be interpreted as being applied to all the objects. However, depending on the action and the current state of all the objects. The action will influence only a subset of the objects. CausalCF assumes that the CF_model can learn to determine if an action influences an object. Therefore, all the objects in the same time step share the same T, R1, R2 and R3 parameters.

The existence of immovable objects or obstacles in CausalWorld adds to the complexity. This thesis only trains and evaluates CausalCF in environments with no obstacles involved. However, the modules are designed so that they can be adapted to environments with obstacles. Obstacles only have a Partial goal feature. It might be falsely assumed that obstacles do not have to be included in pose_3d_ab and pose_3d_c because the actions

do not influence obstacles. However, obstacles may affect the influence of an action over other movable objects. Obstacles can block other objects from moving, or they could change the path that the object moves. Obstacles can be concatenated in the same way as all the other objects by specifying the Object features for the Obstacles. The id, Bounding box size, Cartesian position and Quaternion orientation fields of the Object features can be set to be the same as in Partial goal features for Obstacles (Figure 3.2). Obstacles are already in their "desired" end position. The Linear and Angular velocity can be set to zero as obstacles do not move. CF_model can compute the influence of obstacles over other objects through the mlp_inter type sub-modules.

Figure 3.7 shows the parameters of the Object feature in pose_3d_ab and pose_3d_c. Algorithm 1 describes how the stacked Structured observations are converted into the desired input shape. Currently, Algorithm 1 only deals with observations without obstacles, but it can easily be adapted to deal with obstacles. The output observations (desired_input_obs) from Algorithm 1 represents the pose_3d_ab and pose_3d_c that is processed by the CF_model.

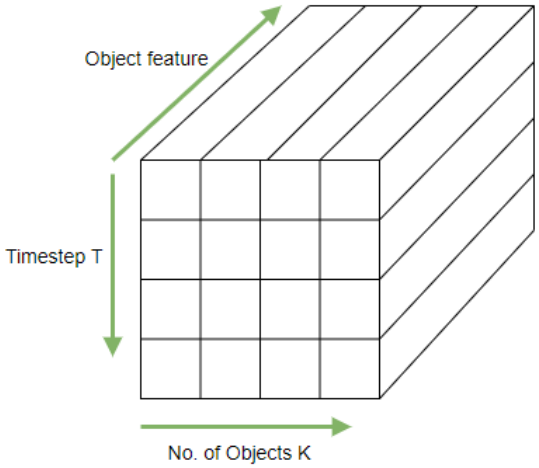


Figure 3.6: Representation of the desired tensor shape for inputs: pose_3d_ab and pose_3d_c.

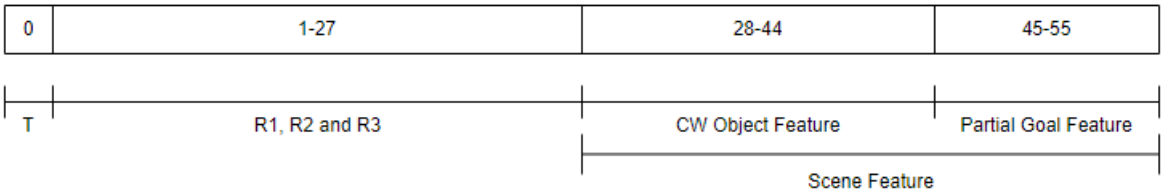


Figure 3.7: Parameters for the Object feature indicated in Figure 3.6. Names of the Parameters follow those of the Structured observation in Figure 3.2.

Algorithm 1: Algorithm for component Convert Input Shape

```
num_timesteps ← T;
num_objects ← K;
desired_input_obs ← [T, K, 56] ; // Empty array for storing new input
lens_obs ← stack_obs.shape[1] ; // Length of the Structured observation
 ; // stack_obs represent the Struct_obs_ab and Struct_obs_c in Figure
  3.5
for t = 0, num_timesteps do
    for k = 0, num_objects do
        desired_input_obs[t, k, : 28] ← stack_obs[t, : 28] ; // Store T, R1, R2 and
         R3
        obj_index_low ← 28 + (k ∗ 17);
        obj_index_up ← obj_index_low + 17;
        desired_input_obs[t, k, 28 : 45] ← stack_obs[t, obj_index_low :
         obj_index_up] ; // Store CW Object features
        part_goal_index_low ← (28 + (num_objects ∗ 17)) + (k ∗ 11);
        part_goal_index_up ← part_goal_index_low + 11;
        ; // Store Partial goal features
        if part_goal_index_up ≥ len_obs then
            desired_input_obs[t, k, 45 :] ← stack_obs[t, part_goal_index_low :];
        end
        else
            desired_input_obs[t, k, 45 :] ← stack_obs[t, part_goal_index_low :
             part_goal_index_up];
        end
    end
end
```

3.3.2 GCN on AB

The GCN on AB component first computes the interactions between the objects and then predicts the next position of the objects. There are processing steps before and after the tensors are passed to the NN sub-modules. These processing steps had to be modified to suit the new input shape. A change to be noted is the processing after the mlp_inter sub-module. The CoPhy solution had a function called aggreg_E that takes in the output from mlp_inter and the input presence_ab. The function computes the average effects of the objects and reduces the tensor dimension (mlp_inter output). CausalCF replaces this function with a simple function that reduces and calculates the mean along the same corresponding dimension in the original CoPhy solution. Algorithm 2 describes the processes

in the GCN on AB component.

Algorithm 2: Algorithm for component GCN on AB

```
; // Note: Accurate information about the tensor dimensions are not
  shown in the algorithm
num_timesteps ← T;
num_objects ← K;
for t = 0, num_timesteps do
    x_inter ← process1(pose_3d_ab[t]) ; // Process pose_3d_ab for computing
     interactions
    out_inter ← mlp_inter(x_inter);
    mean_obj_inter ← mean(out_inter, 1) ; // Replaces aggreg_E in original
     solution
    x_next_pos ← process2(pose_3d_ab[t], mean_obj_inter);
    out = mlp_out(x_next_pose);
    list_out.append(out);
end
seq_o ← stack(list_out, 1) ; // Converts list to a tensor
```

3.3.3 RNN on AB

RNN on AB takes the output from GCN on AB and temporally integrates the contextual information. RNN on AB outputs the causal representation (causal_rep) used to train the Causal RL agent. No changes had to be made to this component from the CoPhy solution. Algorithm 3 shows how contextual information is integrated temporally for each object, and the last layer of the RNN is taken as the causal_rep.

Algorithm 3: Algorithm for component RNN on AB

```
; // Note: Accurate information about the tensor dimensions are not
  shown in the algorithm
num_objects ← K;
for k = 0, num_objects do
    x_rnn ← process1(seq_o[k]) ;
    out_rnn ← rnn(x_rnn);
    list_out_rnn.append(out_rnn[−1]) ; // Appends last layer of rnn for
     each object
end
causal_rep ← stack(list_out_rnn, 1) ; // Converts list to a tensor
```

3.3.4 Predict D and Predict stab

The Predict D component predicts the output observations at each time step. Predict stab is a sub-component in Predict D that predicts the stability of all the objects. The stability of the objects affects the predicted observations and is included in the computations. Both Predict D and Predict stab in CoPhy made use of presence_c (Figure 3.4) and the aggreg_E function mentioned in Section 3.3.2. CausalCF does not make use of presence_c as shown in Figure 3.5. Appropriate processing steps that depended on presence_c were modified. Predict D is an integration of all the other components. Algorithm 4 and 5 describes the process for computing the predictions at each time step.

Algorithm 4: Algorithm for component Predict D

```
; // Note:  Accurate information about the tensor dimensions are not
 shown in the algorithm
num_timesteps ← T;
num_objects ← K;
curr_pose ← pose_3d_c;
for t = 0, num_timesteps do
    stability ← Predict_stab(causal_rep, curr_pose);
    list_stability.append(stability);
    x_inter_delta ← process1(curr_pose, causal_rep) ;
    out_inter_delta ← mlp_inter_delta(x_inter_delta);
    mean_obj_inter_delta ← mean(out_inter_delta, 1) ; // Replaces aggreg_E
    x_gcn_delta ← process2(curr_pose, mean_obj_inter_delta);
    out_gcn_delta = mlp_gcn_delta(x_gcn_delta);
    for k = 0, num_objects do
        out_rnn_delta ← rnn_delta(out_gcn_delta[k]);
        list_out_rnn_delta.append(out_rnn_delta[−1]);
    end
    delta_causal_rep ← stack(list_out_rnn_delta, 1);
    delta ← fc_delta(delta_causal_rep);
    delta ← compute(delta, stability);
    curr_pose ← curr_pose + delta;
    list_pose.append(curr_pose);
end
pred_D_out ← stack(list_pose, 1);
pred_stability ← stack(list_stability, 1);
```

Algorithm 5: Algorithm for component Predict stab

; // Note: Accurate information about the tensor dimensions are not shown in the algorithm

1 $x\_inter\_stab \leftarrow process1(causal\_rep, curr\_pose)$;

2 $out\_inter\_stab \leftarrow mlp\_inter\_stab(x\_inter\_stab)$;

3 $mean\_obj\_inter\_stab \leftarrow mean(out\_inter\_stab, 1)$; // Replaces aggreg_E

4 $x\_stab \leftarrow process2(curr\_pose, mean\_obj\_inter\_stab)$;

5 $stability \leftarrow mlp\_stab(x\_stab)$;

3.4 Using Causal Representations in CausalWorld

Modifications had to be made to the original CausalWorld library to concatenate the causal representation to all the observations for training the agent. The agent's decisions are influenced by the current observations and the causal knowledge that the agent has learned. The modifications provided more implementation challenges than design challenges. Implementation details are provided in Chapter 4. In terms of design, the goal was that the CausalWorld library must be able to function with or without a causal representation, and the user can specify it. An extra parameter called *mode* was introduced where *mode* can be set to 1 or 0, specifying if a causal representation exists or not. This design allows CausalCF to be evaluated against methods that do not use causal representations.

3.5 Integrated CausalCF Solution

CausalCF integrates all the modified components. CoPhy is a DL solution that makes use of supervised learning. Adapting a DL solution for use in RL was challenging. RL is usually always trained unsupervised. Supervised learning requires large amounts of labelled data that can be used to evaluate the predictions and train the DL models. RL agents do not use any labelled data, and it receives feedback in terms of rewards and states. DL models can often overfit classes with more training data, and caution must be taken to ensure that classes are relatively balanced. It is more difficult to achieve balanced samples for each scenario with an unsupervised agent.

The CF_model aims to predict the consequences of interactions in an environment. The Causal RL agent interacts with the environment to obtain information about the underlying SCM. The Causal RL agent can generate data to train the CF_model. Interventions are used to prevent the DL model from overfitting on particular samples, and in conjunction with Counterfactuals, it allows the causal representation (causal_rep) to capture more information about the SCM.

The RL agent acts randomly initially in training. Therefore, the data generated by the RL agent might not provide any useful causal information for training the CF_model initially. The current state-of-the-art RL algorithms take a few million time steps of training to converge even for some of the simplest tasks in CausalWorld [6]. Training the Causal RL agent and the CF_model from scratch in parallel will require massive amounts of training. Since the RL agent acts more randomly initially, the causal_rep might not be able to capture any useful causal information. This causal_rep is then concatenated to the observation to train the RL agent. The causal_rep will frequently vary in the initial stages of training, and if it does not capture any helpful information, the causal_rep might confuse the RL agent, and the performance might diverge.

There are several approaches to solve the problem of training the Causal RL agent and the CF_model from scratch. One approach is to concatenate a randomly initialized causal_rep to train the Causal RL agent until it starts to converge or converges on a particular task. The Causal RL agent can choose to stop training to generate data for training the CF_model, or the agent continues to train, and iteration happens between Counterfactual training and Agent training. If the agent stops training and generates data to train the CF_model, the causal_rep can be concatenated to train the agent further after the CF_model is trained. The new causal_rep might cause the performance to degrade initially, and the agent will require more iterations to converge. The agent can also continue to train, and iteration happens between Counterfactual training and Agent training, but the CF_model requires many iterations to train, resulting in an even longer training time.

If there is a pretrained agent, another approach exists that requires the least training time. CausalWorld provides pretrained agents for several of the benchmark tasks. The pretrained agents can be used to generate data and train the CF_model first. The causal_rep can then be concatenated to train the Causal RL agent. This approach allows us first to evaluate if the causal_rep helps the Causal RL agent in training. It is the first time that a Causal RL agent capable of performing Counterfactuals has been applied in CausalWorld. Iteration can then happen between Counterfactual training and Agent training, where the agent can further update its causal knowledge the better it gets. Figure 3.8 provides a summary of the approaches for training the Causal RL agent and the CF_model from scratch. This thesis uses approach 3 because it will require the least amount of training.

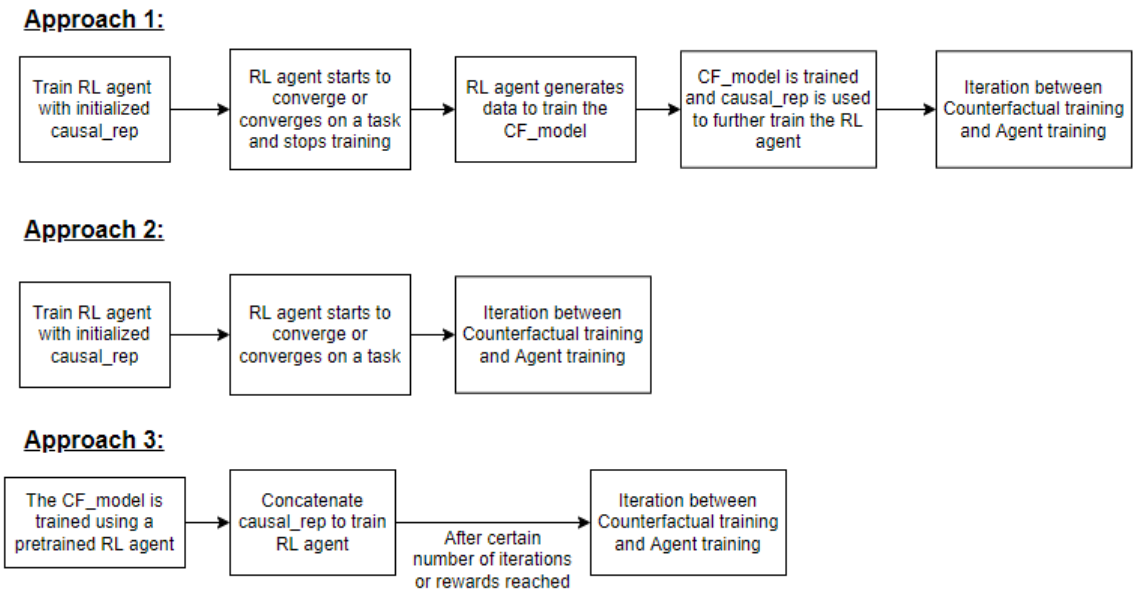


Figure 3.8: The figure provides 3 possible approaches to train the Causal RL agent and the CF_model from scratch.

3.5.1 Train CF_model with Pretrained agent

The pretrained agent is used to generate data for training the CF_model. CF_model is trained for 15 epochs, and for each epoch, the model is trained for 40 iterations. Figure 3.9 shows that the performance of the CF_model converges after 15 epochs. The pretrained agent generates observations for 30 time steps to train CF_model from end to end. In each epoch, an intervention is made on the environment, and the range of values is from Space A (Table 2.3). Algorithm 6 shows the type of interventions made at each epoch for training the CF_model, and it is inspired by Causal Curiosity [18]. Algorithm 7 describes how data is generated to train the CF_model.

Algorithm 6: Algorithm shows the interventions made at each epoch

```
num_epoch_mass ← 3;
num_epoch_goal ← 5;
for i = 0, num_epoch_mass do
    mass ← random_uniform_sample(space_a_range);
    env.do_intervention(mass);
    for j = 0, num_epoch_goal do
        goal ← sample_new_goal(space_a_range);
        env.do_intervention(goal);
        Train CF_model for 40 iterations
    end
end
```

Algorithm 7: Algorithm shows how data is generated to train CF_model

```
max_iter ← 40;
num_timesteps ← 30;
initial_obs ; // Stores initial Structured observation when environment
 resets
for curr_iter = 0, max_iter do
    action ← RL_agent.act(initial_obs);
    for t = 0, num_timesteps do
        next_obs ← env.step(action);
        action ← RL_agent.act(next_obs);
        stack_input_obs ← stack_input_obs.concatenate(next_obs);
    end
    desired_input_obs ← Convert_input_shape(stack_input_obs) ; // Calls
     Convert Input Shape component of CF_model
    pose_3d_ab ← process1(desired_input_obs);
    observation_c ← goal_intervention(env);
    desired_obs_c ← Convert_input_shape(observation_c);
    pose_3d_c ← process2(desired_obs_c);
    pred_D_out, pred_stability, causal_rep ←
     CF_model(pose_3d_ab, pose_3d_c) ; // Pass inputs through all the
     other components of CF_model
    action ← RL_agent.act(observation_c) ; // Generate ground-truth for
     CF_model predictions
    for t = 0, num_timesteps - 1 do
        next_obs ← env.step(action);
        action ← RL_agent.act(next_obs);
        stack_input_obs ← stack_input_obs.concatenate(next_obs);
    end
    desired_input_obs ← Convert_input_shape(stack_input_obs);
    actual_D_out ← process3(desired_input_obs);
    mse_3d ← Calc_loss(pred_D_out, actual_D_out);
    Update CF_model with loss calculation
    Reset environment
end
```

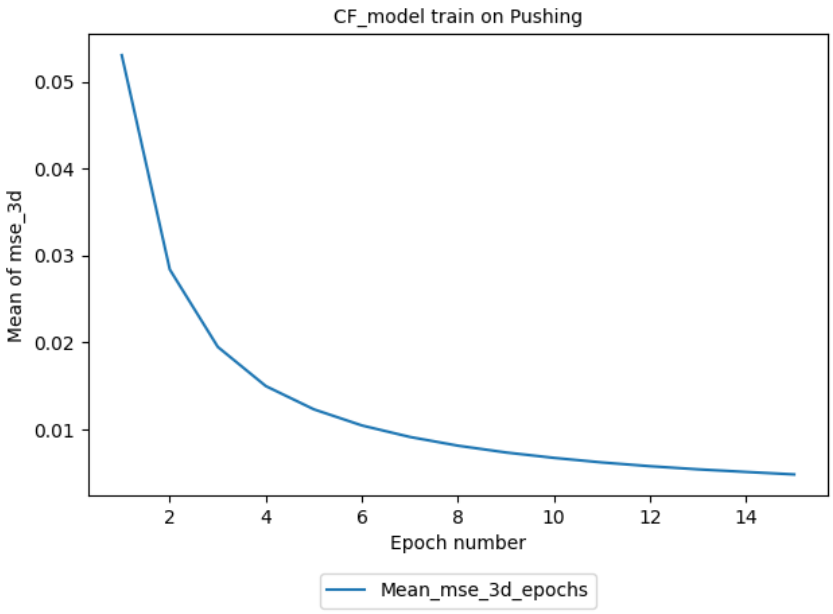


Figure 3.9: Plot of the training performance for the CF_model in the Pushing task.

3.5.2 Iteration between Counterfactual and Agent training

CausalCF follows approach 3 as described in Figure 3.8. CausalCF iterates between Counterfatual training and Agent training after the agent have been trained for 1.5 million time steps. Iteration between Counterfactual and Agent training happens every 500 000 time steps. Adding the iteration feature to CausalCF was more of an implementation challenge. The implementation challenge is a result of the design of the CausalWorld library. Implementation details will be provided in Chapter 4.

3.6 Summary

CausalCF successfully adapts and integrates ideas from CoPhy [5] and Curiosity [18]. CausalCF has two main training phases: Counterfactual training and Agent training. Counterfactual training aims to train the CF_model to perform counterfactuals and provide a useful causal representation that can be used to train a Causal RL agent. CF_model enables the RL agent to imagine the consequences of different actions or interventions and explain the agent's actions by relating the different consequences to the agent's intentions. In Agent training, the Causal RL agent aims to maximize the CausalWorld rewards. CausalCF iterates between the two training phases and allow the agent to update the causal representation further as the agent improves. CausalCF provides the complete Causal RL solution. CausalCF intervenes on the environment, observes the effects of interventions, and performs counterfactuals to learn about the underlying SCM from scratch. Causal factors are captured in the causal representations that CausalCF can use to train a Causal RL agent

and transfer it to new tasks and domains. CausalWorld is used to evaluate the feasibility of the CausalCF solution. Therefore, the design of CausalCF is tailored toward the tasks and the environment provided by CausalWorld. If CausalCF is adapted for use in a different benchmark or environment, modifications to the CausalCF design will be needed. Differences in the dimensions of the observations that come from different environments or benchmarks affect the design of CausalCF. The main design changes will happen in the CF_model because it depends on the dimensions of the observations. Future directions might involve investigating how CausalCF could be modified to be adaptable to observations with different dimensions. The next chapter will provide details about implementing CausalCF in CausalWorld.

4 Implementation

This chapter describes the implementation of CausalCF in CausalWorld. The implementation of CausalCF is available on github. Section 4.1 will provide details about the setup needed to install and run CausalCF. Section 4.2 presents the relationships between the different classes used in CausalCF. Section 4.3 introduces CausalWorld and describes the modifications that were needed to use a causal representation. Section 4.4 provides implementation details about the CF_model. Section 4.5 lists the different functions that were created to train and evaluate CausalCF.

4.1 Setup

CausalWorld is powered by Pybullet and makes use of OpenAI Gym environments. Some of the OpenAI Gym features that CausalWorld requires are only supported by Linux Operating Systems (OS). CausalCF is developed and implemented on a Windows machine. There are three methods for running a Linux OS on a Windows machine: Boot from disk, Virtual machine and Windows Subsystem for Linux (WSL). The user can install the full Linux OS with Boot from disk and Virtual machines. However, Boot from disk might run slow depending on the disk storage hardware and the interface to the computer. Configuring the Virtual machines to work nicely with the Windows machine can be complex and time-consuming. WSL (version 2) provides most of the features usually required of Linux OS, and the current version even allows for running Linux GUI apps. WSL uses minimal computing resources and is easy to set up with no extra configuration. File transfer between WSL and the Windows machine is also straightforward. WSL provides a command-line interface and is used for this project. Virtual environments are recommended to be used for installing the wide range of packages that are needed to run CausalCF. The latest versions of Python do not support some of the packages used in CausalCF. This project uses a Conda virtual environment to install and run CausalCF.

4.2 Overall Flow for CausalCF

Figure 4.1 and 4.2 provides an overview of the relationships between the classes that are used in training and evaluating CausalCF. Below describes the functions and purpose of the different classes used in training CausalCF (Figure 4.1):

- **CausalCF training** contains the code for Counterfactual and Agent training.
- **CF_model** contains all the components of CF_model besides from Convert Input Shape.
- **generate_task** is used for initializing the CausalWorld task and the function passes the list of arguments to the specified task.
- **Task Generators: Pushing, Picking and Stacking2** inherits from the **BaseTask** class. These classes contain all the properties that belong to a specific task and provide observations to CausalWorld.
- **CausalWorld** is used to initialize the CausalWorld environment. The class sets up the environment and coordinates all the other classes in the CausalWorld library. CausalWorld will call the relevant task and intervention classes for stepping through and intervening on the environment.
- **Actor Policies: Pushing and Stacking2** are the pretrained RL agents that CausalWorld provides for the Pushing and Stacking2 task.
- **GoalInterventionActorPolicy** is an intervention actor that can be used with a curriculum for training an RL agent. This actor policy intervenes on the goal pose of the object(s) at a frequency and a period that the user can specify.
- **CurriculumWrapper** allows a curriculum to be set for an environment. Advantages of using curriculum learning was discussed in Chapter 2 Section 2.5.
- **SAC** is the RL algorithm that CausalCF uses.
- **MlpPolicy** is used to define the MLP policy for the RL agent.
- **Monitor** is used record the training statistics of the RL agent.
- **CheckpointCallback** is used to save the training progress of RL agents at a specified frequency.

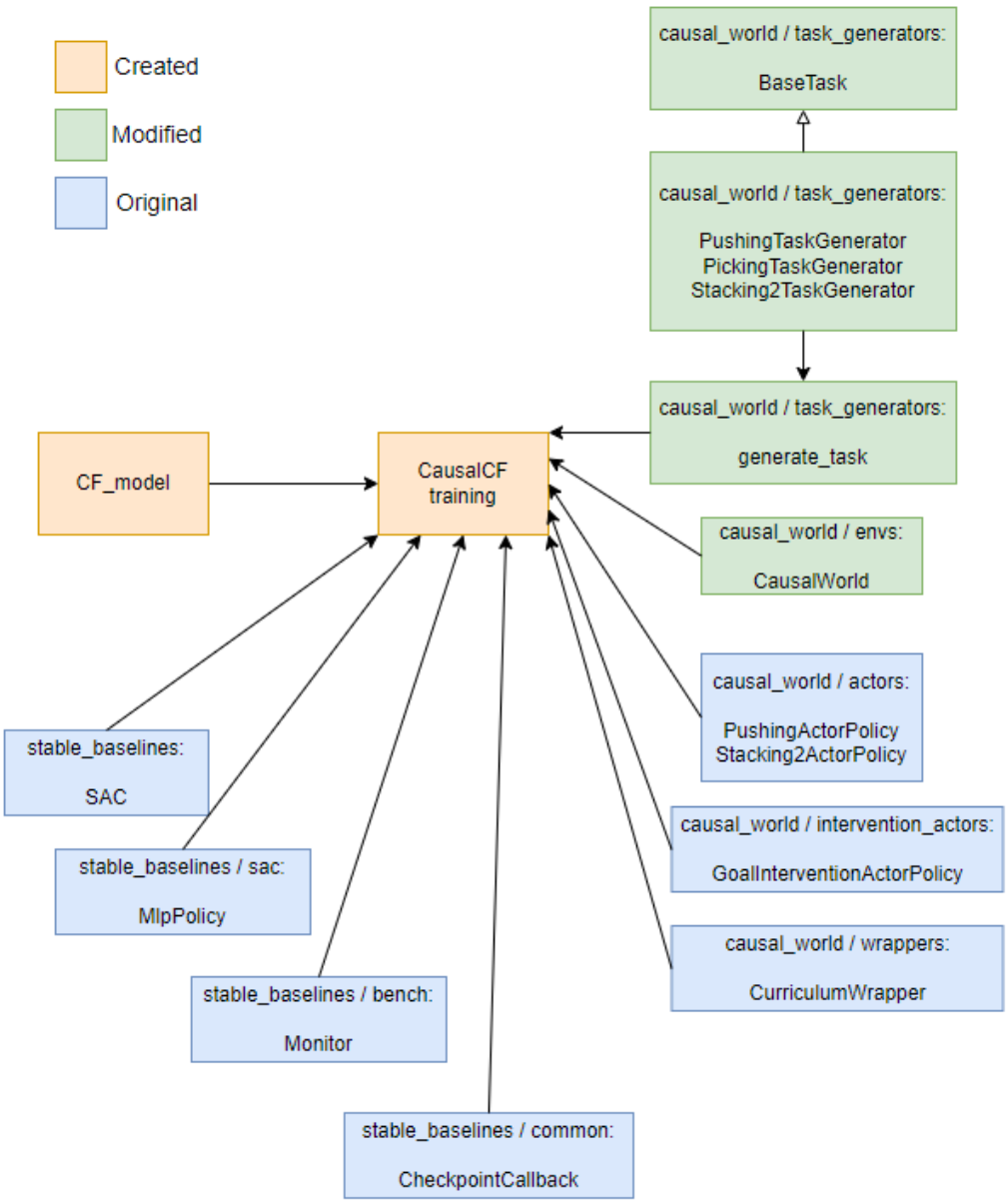


Figure 4.1: The diagram shows the classes that are used in training CausalCF. Classes are colour coded to show which is created and modified for CausalCF.

Below describes the functions and purpose of the different classes used in evaluating CausalCF (Figure 4.2):

- **generate_task**, **CausalWorld** and **SAC** have the same functions as in training.
- **Benchmarks: PUSHING and PICKING** are used to retrieve the different protocols that can be applied for evaluating a RL model on the pushing task.
- **EvaluationPipeline** evaluates the RL model by applying the different protocols.
- **visualiser** is used for automatically generating graph plots of the evaluation results.

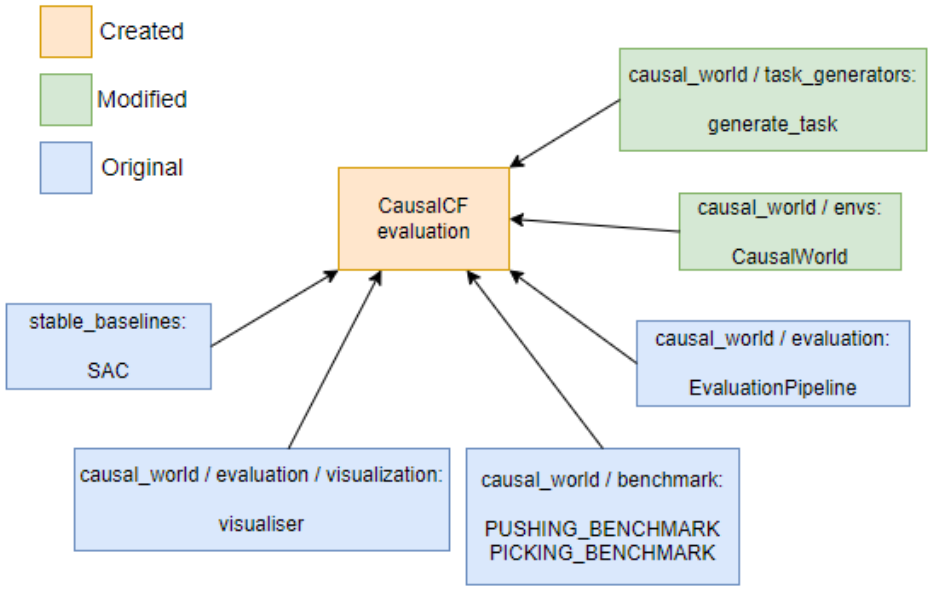


Figure 4.2: The diagram shows the classes that are used in evaluating CausalCF. Classes are colour coded to show which is created and modified for CausalCF.

4.3 CausalWorld and Causal Representations

CausalWorld is an open-source library available on GitHub and Python Package Index (PyPI). The CausalWorld paper [6] evaluated three state-of-the-art RL algorithms PPO, SAC, and TD3, that Stable-baselines3 provides. TensorFlow is used to define the policies for the RL algorithms. The Adam optimizer is used by default in training the RL policies. The CausalWorld library was modified to be able to use causal representations for training the RL agent. Modifications were challenging to implement due to the inter-dependencies of the different files in CausalWorld. Another challenge was navigating to where the CausalWorld library was installed. CausalWorld was installed using pip in a virtual environment that was created. Modifying the CausalWorld library required understanding of the Anaconda virtual environments and installed packages. Below presents a list of files and functions that had to be modified in the CausalWorld library for CausalCF:

- causal_world/envs/causalworld.py
  - **_reset_observations_space**(self)
  - **update_causal_rep**(self, new_causal_rep), this function is not in the original library.
- causal_world/task_generators/Individual task files (e.g. pushing.py)
  - **__init__**(self, variables_space, fractional_reward_weight, dense_reward_weights, activate_sparse_reward, tool_block_mass,

joint_positions, tool_block_position, tool_block_orientation, goal_block_position, goal_block_orientation)

- causal_world/task_generators/base_task.py
  - **__init__**(self, task_name, variables_space, fractional_reward_weight, dense_reward_weights, activate_sparse_reward)
  - **filter_structured_observations**(self)

The function **update_causal_rep** was created and added to the **CausalWorld** class so that the causal_rep can be updated in the iteration between Counterfactual and Agent training. In CausalWorld, the created tasks are wrapped in the environments created to train the RL agent. Therefore, task properties such as the causal_rep that is used to train the agent can only be accessed through the environment. Figure 4.3 shows the inter-dependencies of the files that were modified. The same functions have to be modified for each of the specific task files if it is to be used for CausalCF. Below presents the additional parameters that were added to the modified functions:

- causal_world/task_generators/Individual task files (e.g. pushing.py)
  - **__init__**(self, ..., mode, causal_rep, low_cRep, high_cRep, shape_cRep)
- causal_world/task_generators/base_task.py
  - **__init__**(self, ..., mode, causal_rep, low_cRep, high_cRep, shape_cRep)

The mode parameter specifies if a causal representation will be used. The causal representation is converted to an array and passed through as the argument causal_rep. The causal_rep is concatenated to all the observations in the function **filter_structured_observations**. The parameters low_cRep, high_cRep and shape_cRep is used for the function **_reset_observations_space** when the environment resets.

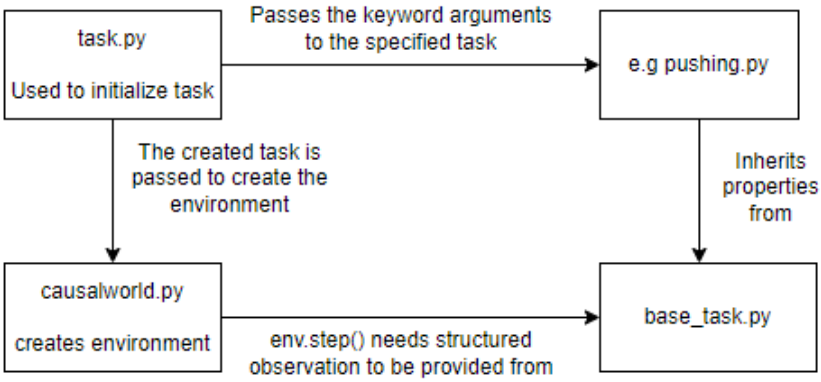


Figure 4.3: Shows the inter-dependencies between the task files. The file task.py is not modified as it simply passes the arguments as a list to the specific task.

4.4 CF_model

CF_model is an adapted version of CoPhy. The CoPhy benchmark is open-source, and all the code is available on GitHub. CoPhy used Pytorch to implement the NN module. The Adam optimizer is used with a learning rate of 1e-3. CUDA is enabled to speed up the training. Figure 4.4 shows the definitions of all the NN sub-modules that belong to each of the components of CF_model. Below lists out all the functions that CF_model uses:

- **__init__**(self, num_objects)
- **gcn_on_AB**(self, pose_3d_ab)
- **rnn_on_AB**(self, seq_o)
- **pred_stab**(self, causal_rep, curr_pose)
- **pred_D**(self, causal_rep, pose_3d_c, B, T)
- **forward**(self, pose_3d_ab, pose_3d_c)

The **forward** function passes the inputs and outputs to different components of the CF_model and returns the final output.

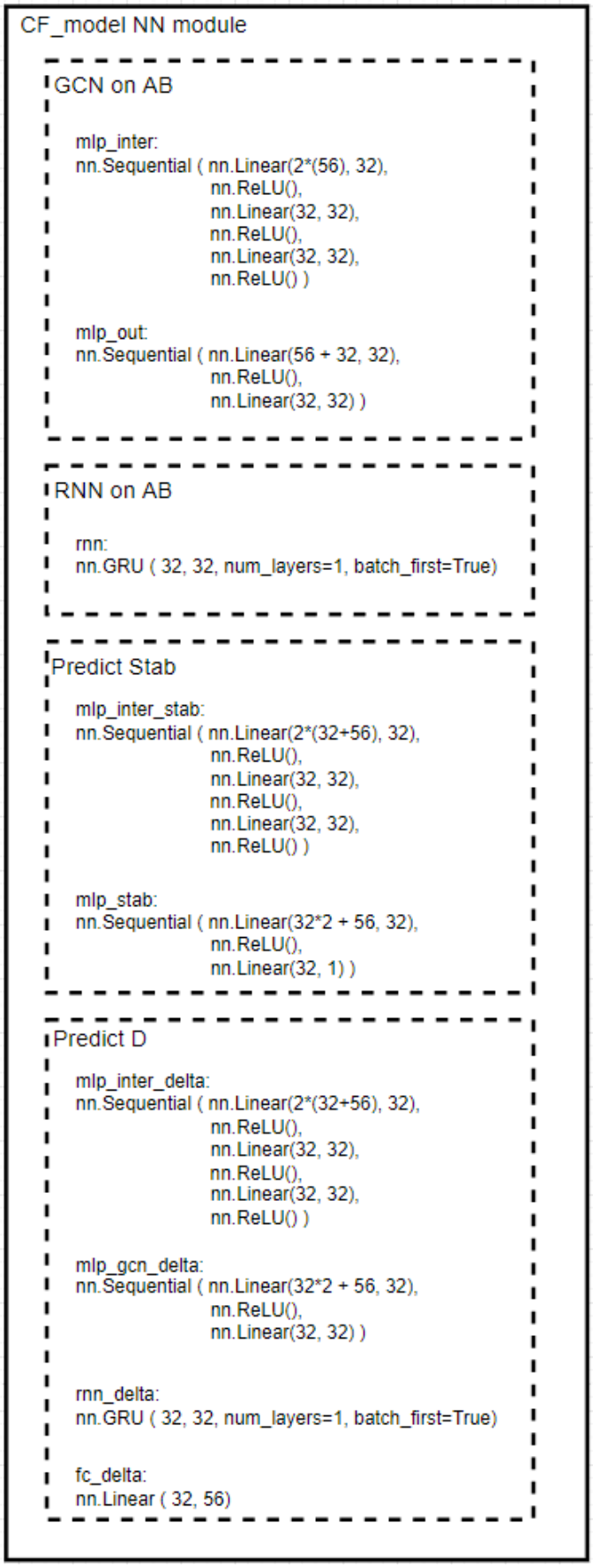


Figure 4.4: Shows the NN module of CF_model with all its components and NN sub-modules defined.

4.5 CausalCF

Training and evaluation are in separate files for CausalCF. The causal representation is detached from the gradients for training the CF_model, and it is saved as a numpy file. Different models were trained to be evaluated against each other. The models were trained for 7 million time steps and takes approximately 22 hours. Counterfactual training does not take long, and CausalCF took the same amount of time to train as the other models. The different models used in evaluation will be discussed in Chapter 5. The models were saved at every 500 000 time steps so that progress is not lost if there is a hardware or software failure. The functions used for training CausalCF are listed below:

- **Calc_num_objects**(env_obs)
- **Convert_input_shape**(stack_obs, timesteps, num_of_objects)
- **Calc_loss**(pred_obs_d, actual_obs_d)
- **Pretrain_CF_model**(device, env, RL_agent, initial_obs, num_of_objects, CF_model, print_freq, list_mse_3d, num_timesteps, max_iter)
- **train_CF_model**(device, env, RL_agent, initial_obs, num_of_objects, CF_model, print_freq, list_mse_3d, num_timesteps, max_iter)
- **main**()

Calc_num_objects takes in the structured observation and computes the number of objects that are in the observation. **Convert_input_shape** converts the stacked structured observations into the desired pose_3d_ab and pose_3d_c. **Pretrain_CF_model** trains the CF_model for 40 iterations. Pretrained agents provided by CausalWorld and the agents provided by Stable Baselines calls different functions for retrieving the next action from the policy. **train_CF_model** is defined almost the same as **Pretrain_CF_model**, but it is used by the SAC agent (Stable Baselines) to update the causal_rep when iteration happens between Counterfactual and Agent training. The **main** function contains the overall flow for Counterfactual training and Agent training. After the models are trained, they are evaluated in a separate file. Below lists the functions that are used for evaluating CausalCF:

- **CFRL_policy_fn**(obs)
- **main**()

The evaluation pipeline provided by CausalWorld passes the observations to a function (**CFRL_policy_fn**) that needs to be defined. **CFRL_policy_fn** concatenates the causal representations to the observations first. The policy then returns the action that is

mapped to that observation. The action is passed to the evaluation pipeline, and the process iterates until the evaluation is complete.

4.6 Summary

The setup used for implementing and running CausalCF was described in this Chapter. The causal representations are concatenated to the observations for training the Causal RL agent, and modifications had to be made to the CausalWorld library. The set of functions that had to be modified in the CausalWorld library was presented. Section 4.4 provides the functions and the definitions of all the NN sub-modules that the CF_model uses. The functions that were created for training and evaluating CausalCF were presented in Section 4.5. The next Chapter will evaluate the implemented designs of CausalCF in various scenarios.

5 Evaluation

CausalWorld consists of a wide range of non-trivial robotic control and object manipulation tasks. The tasks have similar causal structures and can be used to measure the robustness of CausalCF. CausalWorld bridges the gap in Sim2Real transfer by using techniques like domain randomization, curriculum learning and incremental environment complexity. CausalCF's performance in CausalWorld measures to a certain extent how feasible CausalCF is for real-world problems. CausalCF provides a complete Causal RL solution that can perform Interventions and Counterfactuals, and the causal representations learnt can be transferred to a different task. It was discussed in the previous chapters how CausalCF is designed and implemented to be tested in CausalWorld. CausalCF is the first complete Causal RL solution applied to non-trivial RL robotic control tasks. This chapter will evaluate the CausalCF solution in the CausalWorld environment. Section 5.1 presents the objectives that are set to be achieved in the evaluation. Section 5.2 discusses the different evaluations that are performed to achieve the objectives. Section 5.3 describes the metric that is used to measure the performance of CausalCF in CausalWorld. Section 5.4 presents and analyses the results obtained from the different evaluations performed.

5.1 Objectives

CausalCF aims to learn about the underlying SCM of an environment from scratch using Interventions and Counterfactuals and capture the causal structure with an abstract causal representation. The learnt causal representation is supposed to capture the invariant causal factors across domains. The causal representation can be transferred to new tasks or domains with a similar causal structure which improves the robustness of Causal RL agents. CausalCF is designed to work for different tasks available in CausalWorld. Therefore, there are three evaluation objectives:

1. Experimentally illustrate that CausalCF is more robust than methods that do not use Causal RL.
2. The causal representation learnt in CausalCF is transferable to new tasks.

3. CausalCF can be directly applied to different tasks in CausalWorld without any modifications to the design.

5.2 Evaluation Stages

There are three different types of evaluation performed on CausalCF. The three evaluation stages are component testing, causal_rep transfer and stacking challenge. Component testing contributes to objective 1 (Section 5.1). Causal_rep transfer contributes more towards objective 2 (Section 5.1). Objective 3 (Section 5.1) is illustrated with the stacking challenge.

Component testing compares four different RL solutions or models that are applied in the Pushing task (Chapter 2 Section 2.5.1). The four different RL models compared are CausalCF, CausalCF(without iterating between Counterfactual and Agent training), RL model (uses interventions but no counterfactuals), and an RL model that uses no interventions and no counterfactuals. The performance of the different RL models measures the contributions of the different components of CausalCF. The different RL models are passed through an evaluation pipeline that CausalWorld provides to measure generalizability.

Causal_rep transfer demonstrates the transferability of the causal representation learnt by CausalCF. The causal representation (causal_rep) learnt by CausalCF in the Pushing task is transferred to train a Causal RL agent in the Picking task (Chapter 2 Section 2.5.1). The performance of the Causal RL agent that used a causal_rep is compared against the one that did not use the causal_rep.

Stacking challenge directly applies CausalCF in the Stacking2 task (Chapter 2 Section 2.5.1) without any modifications and the training performance is measured. CausalWorld applied the state-of-the-art RL methods to Stacking2, and all of the methods failed to converge to a good performance.

5.3 Metrics

The rewards for the different tasks in CausalWorld were discussed in Chapter 3 Section 3.1.3. Rewards are commonly used in RL to measure performance because RL agents aim to maximize cumulative rewards. However, in the case of CausalWorld, there is a metric called fractional success that provides for more explainability than rewards. Equation 1 describes the formula used to calculate fractional success. Where n represents the number of objects, O_vol_i represents the current pose of the i-th object, and G_vol_i represents the desired

goal pose of the i-th object.

$$fractional\_success = \frac{\sum_{i=1}^{n} intersection(O\_vol_i, G\_vol_i)}{\sum_{i=1}^{n} G\_vol_i} \tag{1}$$

The goal of all the tasks in CausalWorld is to arrange the blocks into their desired goal shape. The function $intersection(O\_vol_i, G\_vol_i)$ calculates the overlapping volume between the current block pose and the objects' goal block pose. Figure 5.1 shows what is meant by overlapping volume. Fractional success measures how close the agent is to arrange all the objects into their desired goal shape. If fractional success is 0, there is no overlap between the current block pose and the desired block pose. When fractional success is 1, there is a perfect overlap between the current block pose and the desired block pose. Fractional success is used to evaluate the performance of the different RL solutions.

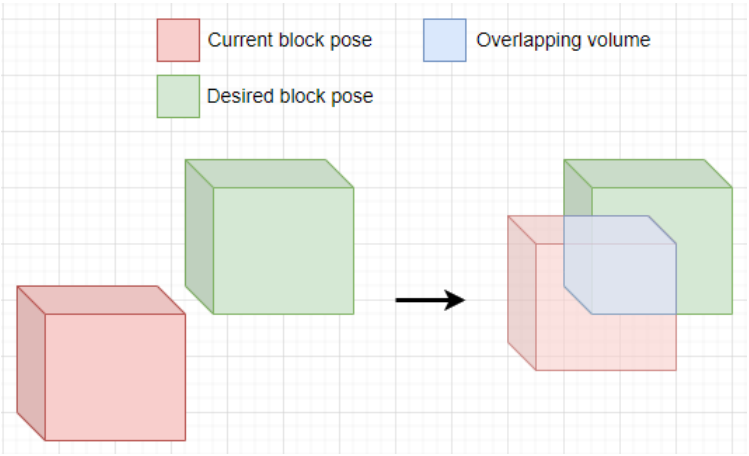


Figure 5.1: The diagram shows the overlapping volume between the current block pose and goal block pose for a single object.

5.4 Results and Analysis

This section presents the results of the different evaluations made on CausalCF in the CausalWorld environment. The different evaluations performed were briefly described in Section 5.2. Further details about each evaluation performed will be provided in this section. The results are analysed, and discussions are made about how the results achieve each of the objectives described in Section 5.1. Results for the Stacking2 task will be compared and referenced against the benchmark performances that are provided in the CausalWorld paper [6].

5.4.1 Training Details

[6] showed that SAC was able to converge for most tasks well before 10 million time steps. Therefore, all of the training was performed for 7 million time steps. Each of the training

sessions takes approximately 22 hours. Different tasks may have different episode lengths. Therefore, not all training would have been performed for the same number of episodes. Table 5.1 shows the training parameters used for the different tasks that CausalCF is evaluated in. All the RL solutions evaluated in this chapter uses the same SAC configuration described in Chapter 3 Section 3.1.4.

Table 5.1: Table presents the training parameters used for each task used to evaluate CausalCF.

	Pushing	**Picking**	**Stacking2**
Total time steps	7000000	7000000	7000000
Episode length	834	834	1667
Number of episodes	8393	8393	4199
Skipframe	3	3	3
Space	A	A	A
Checkpoint frequency	500000	500000	500000

5.4.2 Component Testing

Component Testing trains four different RL solutions: "CausalCF (iter)", "Counterfactual + Intervention" (CausalCF without iteration), "Intervene" (uses interventions and no counterfactuals) and "no_intervene" (uses no interventions and no counterfactuals). The trained RL solutions are further evaluated in an evaluation pipeline provided by CausalWorld to measure robustness.

The training performance of the different RL solutions in the Pushing task is provided in Figure 5.2. The x-axis is the number of time steps the agent was trained for, and the y-axis is the mean of the fractional successes for every 100 episodes. Some interventions might lead to unsolvable scenarios, and the agent will receive a low fractional success for these rare scenarios. These scenarios can happen in the later stages of training. The mean fractional success is used to produce a smoother plot of the training performance.

The "no_intervene" agent trains with more repetitive tasks as no interventions are used. Therefore, the environment for the "no_intervene" agent is simpler, and the agent was able to converge to a good training performance faster than all the other solutions. The "no_intervene" agent converged to a fractional success of higher than 0.9 in about 1.5 million time steps. In contrast to the training environment used for "no_intervene," all the environments used for the other RL solutions intervened on the goal shape at every episode. Intervention is made on one of the CausalWorld variables at every episode and this choice is

justified in Chapter 2 Section 2.4.1. Interventions make the task more challenging, and the "Intervene" solution struggled to achieve the same performance as "no_intervene." The "intervene" solution was only able to converge to an approximate fractional success of 0.8 in 7 million time steps, and the solution took longer to converge than the "no_intervene" solution, as shown in Figure 5.2. The "Counterfactual + Intervene" solution is CausalCF without the iteration between Counterfactual and Agent training. CausalCF outperformed "Intervene" in the same challenging training environment and converged to the same performance as the "no_intervene" solution. It took CausalCF longer to converge than "no_intervene" because the training environment for CausalCF is more challenging. CausalCF (iter) took longer to converge than "Counterfactual + Intervene" due to what was discussed in Chapter 3 Section 3.5. The variations to the causal_rep during the iteration require the RL agent to partially re-learn and converge more slowly. However, CausalCF converged to the same training performance as "no_intervene," and it should be more robust than all the other solutions trained. The robustness of the RL solutions is measured by evaluating the solutions in the evaluation pipeline provided by CausalWorld.

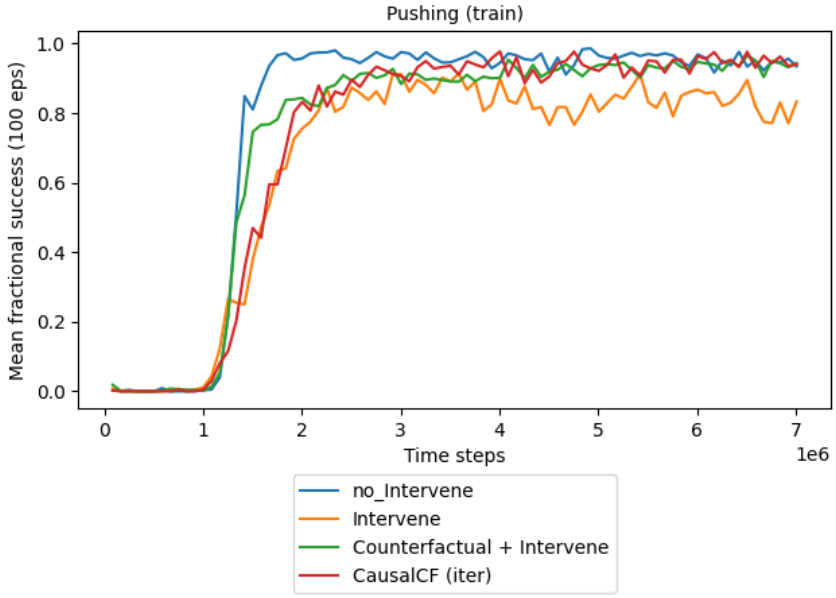


Figure 5.2: The training performance of the different RL solutions in the Pushing task.

The performance of the different RL solutions in the evaluation pipeline is provided in Figure 5.3. The naming and the corresponding color codes used in Figure 5.3 are kept consistent with Figure 5.2. In Figure 5.3, the x-axis shows the 12 protocols applied in the evaluation pipeline, and the y-axis is the full integrated fractional success. Each protocol evaluates the solution for 200 episodes. The full integrated fractional success takes the mean of the fractional success over the 200 episodes. Each of the 12 protocols applied in the evaluation pipeline modifies a specific subset of the CausalWorld variables. The range of values that

the CausalWorld variables can take is different for each protocol. Table 5.2 describes the 12 different protocols used. *Var* is the causal variables that are intervened in each protocol. Space A and B (Chapter 2 Table 2.3) define the ranges of values that these causal variables can take for each protocol. All the RL solutions performed better for protocols that used space A than B because they were all trained in space A. Interventions on different causal variables create different challenges. All the solutions performed worse on the protocols that intervened on the block pose. The change in block pose requires the agent to consider the initial grasp on the block, which the agent did not consider as much in training. "Intervene" generalized better than "no_intervene" as interventions on one or a subset of causal variables are better than no intervention (Chapter 2 Section 2.4.1). CausalCF was more robust than all the other solutions as the agents used counterfactuals to capture more causal information about the underlying SCM. CausalCF (iter) outperforms all the other solutions in most protocols and is the most robust (Objective 1). CausalCF (iter) allows the agent to update its causal_rep or causal knowledge further the better the agent gets. The extra iterations of Counterfactual training for CausalCF (iter) also allow the agent to capture more causal information.

Causal_rep captures the causal factors that remain invariant across domains, explaining why CausalCF is more robust than the other RL solutions. However, it cannot be determined at the moment if CausalCF was able to capture all of the causal mechanisms during training. An interesting future direction might be to apply ICM and SMS (Chapter 2 Section 2.4) to determine if all the causal mechanisms were captured. However, even if the causal_rep only captures the causal mechanisms partially. Causal_rep can be transferred to tasks with similar causal structures.

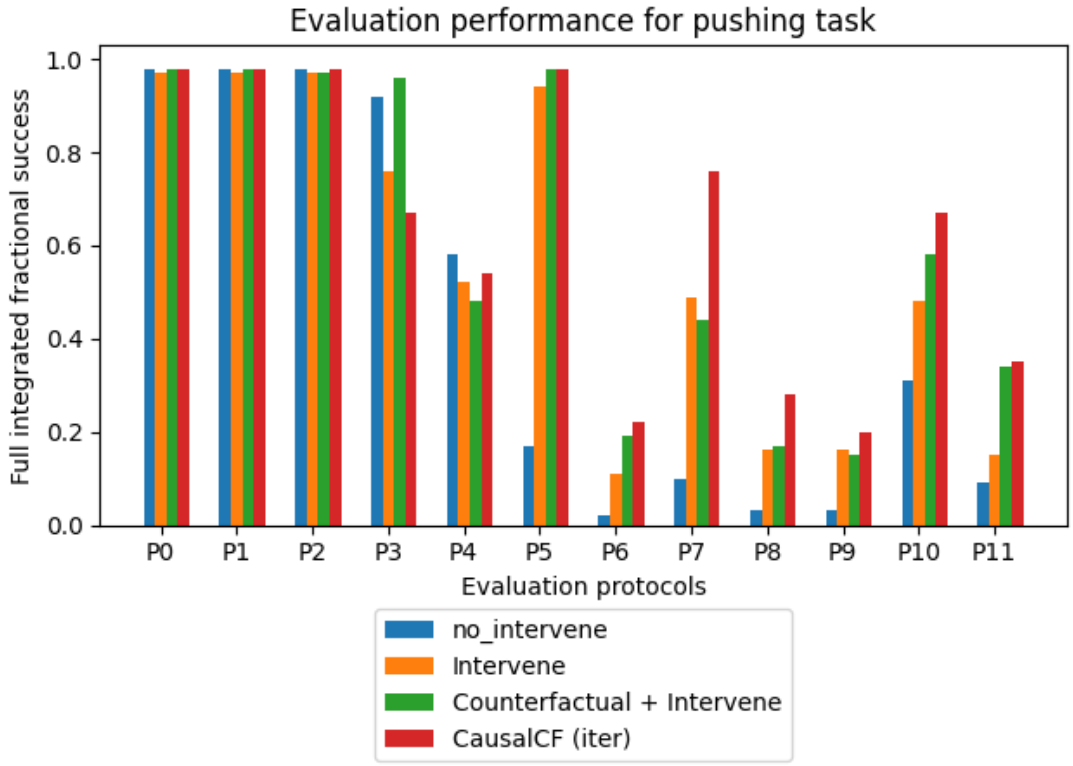


Figure 5.3: The graph plots the performance of the RL solutions for each evaluation protocol next to each other for easier comparison. There are 12 protocols applied to the evaluation pipeline.

Table 5.2: Table presents the modified causal variables and the values that the causal variables can take for each protocol. Adapted from CausalWorld [6]. Variables: bp - block pose, bm - block mass, bs - block size, gp - goal pose and ff - floor friction.

	P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
Space	A	A	B	A	A	A	B	A	B	B	A	B
Var	-	bm	bm	bs	bp	gp	bp, gp	bp, gp, bm	bp, gp, bm	bp, gp, bm, ff	all	all

5.4.3 Causal_rep transfer

The tasks in CausalWorld have similar causal structures where there are common causal mechanisms across the different tasks. Chapter 2 Section 2.5.1 discussed about how the skills learnt from one task is also used in another task. The learnt causal_rep of the CausalCF (iter) solution is transferred to train a new RL agent in the Picking task. The training and evaluation performance of the RL agent that used the causal_rep is compared

against an RL agent that trains only with interventions and no causal_rep.

The training performance of the two solutions is shown in Figure 5.4. The same Interventions were used for training both solutions. The goal shape of the block was intervened on at every episode. The "transfer Causal_rep + intervene" solution converged to a fractional success of higher than 0.9 in about 2.5 million time steps. The "Intervene" solution converged to a fractional success of about 0.8 in approximately 3 million time steps. The RL agent that used the transferred causal_rep converged faster and to a better training performance than the agent that only used interventions. The causal_rep provides additional prior causal knowledge for the RL agent. A good training performance does not mean that the solution is robust. The two RL solutions are passed through an evaluation pipeline for the Picking task.

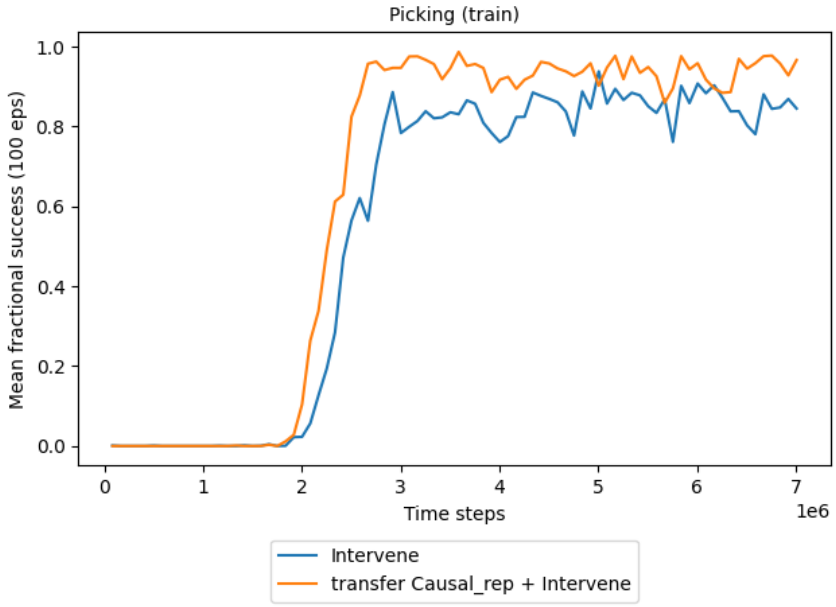


Figure 5.4: Training performance of the two different RL solutions compared. One uses a transferred causal_rep and interventions and the other only uses interventions.

Figure 5.5 shows the performance of the two RL solutions in the evaluation pipeline. The solution that used the causal_rep performed better than the one that did not in more of the evaluation protocols. The causal_rep learnt from the Pushing task helps the agent perform better and become more robust in the Picking task. This result confirms that the causal_rep is capturing causal mechanisms through Interventions and Counterfactuals. The agent's performance that used a causal_rep could have been improved if the agent was able to update the causal_rep with new experiences and causal information in the Picking task. There is one limitation with the transferability of the causal_rep for CausalCF. Currently, the dimensions of the causal_rep is influenced by the number of objects (Chapter 3 Figure 3.5). A causal_rep learnt in the Pushing task cannot be directly transferred to a task that

deals with more objects like Stacking2 and Towers. There will be a mismatch in dimensions if it is directly transferred. An approach to solve this transferability issue might be to assign the objects' latent representations with the learnt causal_rep, and if there are more objects in the new task, then the leftover objects are randomly initialized. The next section evaluates and discusses the adaptability and flexibility of CausalCF.

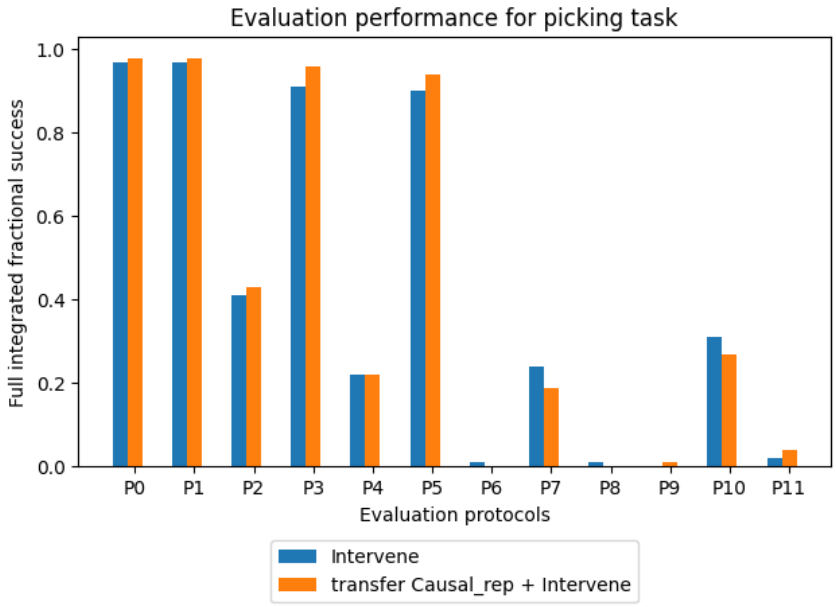


Figure 5.5: Two different RL solutions evaluated in the evaluation pipeline for the Picking task. The 12 protocols applied are the same as it is described in Table 5.2.

5.4.4 Stacking Challenge

CausalCF can be applied to any of the tasks in CausalWorld without further modifications to the design, except for tasks that involve obstacles and fixed blocks. Chapter 3 Section 3.3.1 discussed how CausalCF could be easily modified to accommodate tasks that have obstacles. CausalCF was directly applied to the Stacking2 task without any modifications to the design. Figure 5.6 shows the training performance of CausalCF in the Stacking2 task. The training performance was slightly below expectations. However, CausalCF does show the potential of performing better. The length of a Stacking2 episode is 1667, which means the agent could only train for 4199 episodes when the task is more challenging than Pushing and Picking. In the CausalWorld paper [6], SAC converged to a similar performance in 10 million time steps. PPO performed the best and reached a fractional success of approximately 0.5, but it required 100 million time steps. It can be observed in Figure 5.6 that CausalCF (iter) seems to be still improving, and the performance is not converging. The performance of CausalCF (iter) could have been improved by simply training it for more time steps. Also, counterfactuals were only performed for the first 30 time steps. The agent would not be able to capture much meaningful causal information in the first 30 time steps. If

counterfactuals were performed for the later stages of an episode, then more causal knowledge would have been captured, and the performance would have been improved. The agent has to consider aspects of stability and the interaction between the two blocks in the later stages of an episode. Stacking2 uses skills from Pushing, Picking, and Pick and Place. Another approach that would have improved the performance of CausalCF (iter) is to train it in other tasks first, like Pick and Place, and then transfer it to the Stacking2 task.

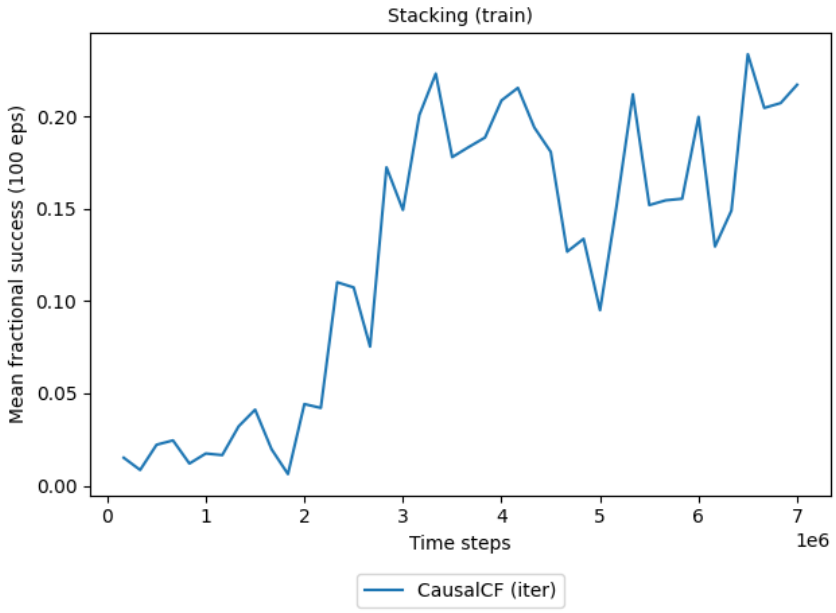


Figure 5.6: Training performance of CausalCF (iter) for the Stacking2 task.

5.5 Summary

This chapter evaluates the robustness of the CausalCF solution in a range of tasks provided by CausalWorld. The evaluation objectives and the evaluation stages needed for achieving the objectives were presented. A fractional success metric was introduced and used because it provided more explainability than using rewards.

CausalCF converged to excellent training performance and proved to be the most robust solution in Component testing. The evaluation results showed that none of the design components of CausalCF is redundant and that interventions and counterfactuals help improve the robustness of RL. In causal_rep transfer, the causal_rep learnt from the Pushing task was directly transferred to be used by a Causal RL agent in the Picking task. The agent that used the causal_rep was able to converge faster and perform better in training than the agent that did not use a causal_rep. The results from the evaluation pipeline showed that the agent that used the causal_rep was the most robust and proved that CausalCF could capture the causal mechanisms that remain invariant across domains. Limitations about the transferability of the causal_rep were highlighted, and corresponding

solutions were provided. CausalCF was directly applied to the Stacking2 task for the stacking challenge. The results show that CausalCF is a flexible and adaptable solution. CausalCF could not achieve a good training performance in 7 million time steps for the Stacking2 task. Reasons for the underperformance of CausalCF were discussed, and methods that could improve the performance of CausalCF in the Stacking2 task were provided.

6 Conclusion

CausalCF is the first complete Causal RL solution that successfully tackles complex RL robotic tasks. CausalCF improves the robustness of RL and provides mechanisms to help improve the explainability of RL. This chapter will summarize the thesis and provide interesting future directions based on the work done for CausalCF. This project does not contain any racial or gender dimensions. Negative social impacts that may arise from this work are the same as those that would arise from the general improvements of AI.

6.1 Thesis Contribution

Chapter 1 highlights the need for improving the robustness and explainability of RL and motivates the research done for this thesis. Limitations with existing research in this field were briefly discussed. Approaches that simplify Deep NN architectures suffer a loss in performance and have limitations with the transferability and robustness of the DRL solutions. Causal RL was introduced as an approach that will improve RL's robustness and explainability. Challenges for Causal RL were presented, and part of the thesis objectives is to overcome the challenges.

Chapter 2 identified the need for Causal RL and provided the background required for understanding CausalCF. Approaches that could help overcome the challenges of Causal RL were introduced. Causal Curiosity provided a method for using a causal representation and interventions to train an RL agent. CoPhy is a DL solution that can perform counterfactuals and be adapted for use in RL. CausalWorld allows interventions on different causal variables to be easily performed and provides techniques that reduce the cost of Sim2Real transfer. A range of complex RL robotic control tasks are available in CausalWorld to evaluate the robustness of CausalCF.

Chapter 3 described precisely how ideas from Causal Curiosity and CoPhy were combined and adapted to be trained and evaluated in CausalWorld. CausalCF uses two training phases and concatenates the causal representation to all the observations used for training the RL agent. This idea was inspired by Causal Curiosity. CoPhy was adapted for use in RL, and the adapted version is called CF_model. The two training phases of CausalCF are

Counterfactual and Agent training. Counterfactual training makes use of interventions and trains the CF_model to perform counterfactuals. The causal representations learnt during Counterfactual training capture useful causal mechanisms. In Agent training, the agent is trained with the causal representations concatenated to the observations. CausalCF provides an iteration feature where the agent can update the causal representation as the agent learns.

Chapter 4 provides details about how CausalCF is implemented to be trained and evaluated in CausalWorld. Modifications to the original CausalWorld library were needed to use causal representations for training RL agents. Modifying CausalWorld for CausalCF was challenging due to the interdependencies between the files. Detailed descriptions of each of the NN sub-modules of CF_model were presented.

Chapter 5 evaluated CausalCF in the CausalWorld benchmark with various scenarios and tasks. CausalCF achieved the best training performance and was the most robust in Component testing. The results show that each component of CausalCF contributes to improving the robustness of RL. The causal representation learnt in the Pushing task was transferred to train an agent in the Picking task. The agent that used the causal representation performed better in training and more robustly in the evaluation pipeline than the agent that did not use a causal representation. The causal representation captured causal mechanisms that remain invariant across the tasks. CausalCF is adaptable to varying amounts of objects and time steps. CausalCF was successfully applied to the Stacking2 task without any modifications needed. The performance of CausalCF in the Stacking2 task was below expectation. Discussions were made about how the performance could have been improved.

6.2 Future Work

Throughout the thesis, possible extensions of CausalCF and future directions were mentioned. This section compiles and groups all the future directions into three main categories: Extending CausalCF, Improving RL Transferability and Robustness, and Improving RL Explainability.

1. **Extending CausalCF**

    - Improve Transferability of Causal representation: Currently, the dimensions of the causal_rep are determined by the number of objects. The current design of causal_rep makes transferring the causal_rep to tasks with different amounts of objects troublesome. Causal_rep can be re-designed to have dimensions that are determined by the number of causal mechanisms. The number of causal mechanisms should remain the same across tasks and environments with similar

causal structures.

- Sim2Real Transfer for TriFinger: CausalCF is intended for real-world robotic applications. CausalWorld supports Sim2Real transfer, and it will be interesting to transfer CausalCF onto the real TriFinger robot and observe how CausalCF performs in the real world.
- Structured To Pixel Observations: Pixel observations are closer to real-world observations (images and videos) received by robots. Additional modules will need to be added to CausalCF to process the Pixel observations and extract the regions of interest (ROI). If all of the input observations provided to CausalCF are in the form of images. The dimensions of the different components of CausalCF could be fixed, and CausalCF could be applied to different benchmarks and environments that can provide image observations without any modifications to the design.
- Improve Adaptability of CausalCF: An additional feature that could be added to CausalCF is to allow the user to input the dimensions of the observations of the new benchmark or environment that CausalCF is applied in. The relevant design components of CausalCF are automatically set to make CausalCF suitable for use in the new benchmark or environment.

2. **Improving RL Transferability and Robustness**

- Multitask RL and Incremental Environment Complexity: Tasks with similar causal structures often have common skills that can be transferred from one task to another instead of training Causal RL solutions from scratch for a challenging task. The Causal RL solutions can be trained on multiple simpler tasks first. The Causal RL solution can tackle more challenging tasks once it learns to perform more straightforward tasks. For example, CausalCF could be trained with the Picking and Pushing task before it is trained with the Stacking2 task.
- Automatic Interventions: The RL agents perform interventions (actions) on the environment and receive states and rewards as feedback. Interventions on the causal factors of the environment are usually not made accessible to the agent. The developer is responsible for deciding which causal factors should be intervened during the training of the RL agent. An algorithm could be created to automatically compute the Possibly-Optimal MIS (Chapter 2 Section 2.4.1) from a list of causal factors, and the choice of which causal factors to intervene on is automated.
- Measuring Causal Structure Similarity: Causal RL agents could automatically choose the following tasks to learn based on the causal structure similarity of the

new task with tasks that the agent has already learnt. This research could be performed in conjunction with the Multitask RL and Incremental Environment Complexity direction.

- ICM and SMS for Capturing all Causal Mechanisms: ICM and SMS (Chapter 2 Section 2.4) provides a possible way of determining if the causal representation has captured all the causal mechanisms. The stopping criterion for the training of the Causal RL agent could include the number of causal mechanisms the agent has captured.

3. **Improving RL Explainability**

   - Generating Local Explanations from Counterfactuals and Rewards: CausalCF could provide local explanations for a particular action or decision by performing counterfactuals and imagining the future consequences of different actions based on the causal knowledge of the agent. Explanations would involve providing the different outcomes with the corresponding rewards as RL agents aim to maximize cumulative rewards. The probability of success [51] could be used instead of rewards because rewards vary from task to task. If the imagined outcomes and probability of success match the knowledge of the human expert, then the decisions or actions of the agent could be trusted.

   - Causal Graph from Causal Representation: Causal graphs can provide global and local explanations. Future work could investigate if causal graphs could be generated from abstract causal representations. Decoding the causal graph generated from the agent's causal knowledge into a causal graph relevant to humans will be challenging. Causal graphs and counterfactuals could serve as powerful tools for Causal RL agents to provide explainability to human coworkers.

CausalCF is a complete Causal RL solution that uses a causal representation and can perform the three different Causal RL interactions. The causal representation captures the invariant causal mechanisms across domains or tasks. The causal representation is transferable to new tasks and helps improve the RL agent's robustness. CausalCF can perform level 3 counterfactuals and imagine the consequences of interventions or actions. Therefore, CausalCF can provide local explanations by showing how the different consequences for different actions align with the agent's intentions or goals. CausalCF improves the robustness and explainability of RL and makes RL more trustworthy for real-world deployments.

Bibliography

[1] Andrew G Barto, Satinder Singh, Nuttapong Chentanez, et al. Intrinsically motivated learning of hierarchical collections of skills. In *Proceedings of the 3rd International Conference on Development and Learning*, pages 112–19. Piscataway, NJ, 2004.

[2] Erika Puiutta and Eric M. S. P. Veith. Explainable reinforcement learning: A survey. *ArXiv*, abs/2005.06247, 2020.

[3] Amina Adadi and Mohammed Berrada. Peeking inside the black-box: a survey on explainable artificial intelligence (xai). *IEEE access*, 6:52138–52160, 2018.

[4] E. Bareinboim, J. Correa, D. Ibeling, and T. Icard. On pearl's hierarchy and the foundations of causal inference. Technical Report R-60, Causal Artificial Intelligence Lab, Columbia University, Jul 2020. In: "Probabilistic and Causal Inference: The Works of Judea Pearl", ACM Books, in press.

[5] Fabien Baradel, Natalia Neverova, Julien Mille, Greg Mori, and Christian Wolf. Cophy: Counterfactual learning of physical dynamics. *ArXiv*, abs/1909.12000, 2020.

[6] Ossama Ahmed, Frederik Trauble, Anirudh Goyal, Alexander Neitz, Manuel Wuthrich, Yoshua Bengio, Bernhard Scholkopf, and Stefan Bauer. Causalworld: A robotic manipulation benchmark for causal structure and transfer learning. *ArXiv*, abs/2010.04296, 2021.

[7] Bernhard Scholkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Toward causal representation learning. *Proceedings of the IEEE*, 109:612–634, 2021.

[8] Guiliang Liu, Oliver Schulte, Wang Zhu, and Qingcan Li. Toward interpretable deep reinforcement learning with linear model u-trees. *ArXiv*, abs/1807.05887, 2018.

[9] Tianmin Shu, Caiming Xiong, and Richard Socher. Hierarchical and interpretable skill acquisition in multi-task reinforcement learning. *arXiv preprint arXiv:1712.07294*, 2017.

[10] Judea Pearl et al. Models, reasoning and inference. *Cambridge, UK: CambridgeUniversityPress*, 19, 2000.

[11] Joseph Y Halpern and Judea Pearl. Causes and explanations: A structural-model approach. part ii: Explanations. *The British journal for the philosophy of science*, 56 (4):889–911, 2005.

[12] Joseph Y Halpern and Judea Pearl. Causes and explanations: A structural-model approach. part i: Causes. *The British journal for the philosophy of science*, 2020.

[13] E Bareinboim. Towards causal reinforcement learning (crl). ONLINE, 2020. (https://crl.causalai.net/crl-icml20.pdf), Accessed 21 Jan. 2022.

[14] Richard Dazeley, Peter Vamplew, and Francisco Cruz. Explainable reinforcement learning for broad-xai: a conceptual framework and survey. *arXiv preprint arXiv:2108.09003*, 2021.

[15] Prashan Madumal, Tim Miller, Liz Sonenberg, and Frank Vetere. Explainable reinforcement learning through a causal lens. *ArXiv*, abs/1905.10958, 2020.

[16] Nan Rosemary Ke, Olexa Bilaniuk, Anirudh Goyal, Stefan Bauer, H. Larochelle, Chris Pal, and Yoshua Bengio. Learning neural causal models from unknown interventions. *ArXiv*, abs/1910.01075, 2019.

[17] Anirudh Goyal, Aniket Didolkar, Nan Rosemary Ke, Charles Blundell, Philippe Beaudoin, Nicolas Manfred Otto Heess, Michael C. Mozer, and Yoshua Bengio. Neural production systems. *ArXiv*, abs/2103.01937, 2021.

[18] Sumedh Anand Sontakke, Arash Mehrjou, Laurent Itti, and Bernhard Schölkopf. Causal curiosity: Rl agents discovering self-supervised experiments for causal representation learning. In *ICML*, 2021.

[19] Jun Jin, Daniel Graves, Cameron Haigh, Jun Luo, and Martin Jägersand. Offline learning of counterfactual perception as prediction for real-world robotic reinforcement learning. *ArXiv*, abs/2011.05857, 2020.

[20] Ishita Dasgupta, Jane X. Wang, Silvia Chiappa, Jovana Mitrovic, Pedro A. Ortega, David Raposo, Edward Hughes, Peter W. Battaglia, Matthew M. Botvinick, and Zeb Kurth-Nelson. Causal reasoning from meta-reinforcement learning. *ArXiv*, abs/1901.08162, 2019.

[21] Shengyu Zhu and Zhitang Chen. Causal discovery with reinforcement learning. *ArXiv*, abs/1906.04477, 2020.

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

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

[24] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

[25] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[26] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(89)90020-8. https://www.sciencedirect.com/science/article/pii/0893608089900208.

[27] Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay D. Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. *CoRR*, abs/1312.6082, 2014.

[28] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[29] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[30] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011.

[31] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Lecture 6a overview of mini–batch gradient descent. *Coursera Lecture slides https://class. coursera. org/neuralnets-2012-001/lecture,[Online*, 2012.

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

[33] Yann LeCun. Une procedure d'apprentissage ponr reseau a seuil asymetrique. *Proceedings of Cognitiva 85*, pages 599–604, 1985.

[34] Bryce Goodman and Seth Flaxman. European union regulations on algorithmic decision-making and a "right to explanation". *AI magazine*, 38(3):50–57, 2017.

[35] digital-strategy.ec.europa.eu. (n.d.). Assessment list for trustworthy artificial intelligence (altai) for self-assessment | shaping europe's digital future. ONLINE, 2020. (https://digital-strategy.ec.europa.eu/en/library/assessment-list-trustworthy-artificial-intelligence-altai-self-assessment), Accessed 11 Jan. 2022.

[36] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pages 5045–5054. PMLR, 2018.

[37] Judea Pearl and Dana Mackenzie. *The book of why: the new science of cause and effect*. Basic books, 2018.

[38] Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of causal inference: foundations and learning algorithms*. The MIT Press, 2017.

[39] Bernhard Schölkopf, Dominik Janzing, Jonas Peters, Eleni Sgouritsa, Kun Zhang, and Joris Mooij. On causal and anticausal learning. *arXiv preprint arXiv:1206.6471*, 2012.

[40] Dominik Janzing and Bernhard Schölkopf. Causal inference using the algorithmic markov condition. *IEEE Transactions on Information Theory*, 56(10):5168–5194, 2010.

[41] S. Lee and E. Bareinboim. Structural causal bandits: Where to intervene? In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 2568–2578, Montreal, Canada, 2018. Curran Associates, Inc.

[42] A. Forney, J. Pearl, and E. Bareinboim. Counterfactual data-fusion for online reinforcement learners. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1156–1164, International Convention Centre, Sydney, Australia, Aug 2017. PMLR.

[43] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744, 2020. doi: 10.1109/SSCI47803.2020.9308468.

[44] Joshua P Tobin. *Real-World Robotic Perception and Control Using Synthetic Data*. University of California, Berkeley, 2019.

[45] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, et al. Using simulation and domain adaptation to improve efficiency of deep robotic grasping.

In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 4243–4250. IEEE, 2018.

[46] Bangyu Qin, Yue Gao, and Yi Bai. Sim-to-real: Six-legged robot control with deep reinforcement learning and curriculum learning. In *2019 4th International Conference on Robotics and Automation Engineering (ICRAE)*, pages 1–5. IEEE, 2019.

[47] Thomas Chaffre, Julien Moras, Adrien Chan-Hon-Tong, and Julien Marzat. Sim-to-real transfer with incremental environment complexity for reinforcement learning of depth-based robot navigation. *arXiv preprint arXiv:2004.14684*, 2020.

[48] 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.

[49] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[50] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[51] Francisco Cruz, Richard Dazeley, Peter Vamplew, and Ithan Moreira. Explainable robotic systems: understanding goal-driven actions in a reinforcement learning scenario. *Neural Computing and Applications*, 2021.