Using Gradient-based Optimization for Planning with Deep Q-Networks in Parametrized Action Spaces

Jonas Ehrhardt, Johannes Schmidt, René Heesch, Oliver Niggemann

HSU-AI Institute for Artificial Intelligence
Helmut-Schmidt-University, Hamburg, Germany
firstname.lastname@hsu-hh.de

Abstract

Many real-world planning problems feature parametrized action spaces, where each action is augmented by continuous parameters. Though deep Reinforcement Learning has achieved remarkable results in solving control and planning problems, it falls short at two central challenges of real-world planning problems with parametrized action spaces: (i) There is an infinite number of action-parameter candidates in every step of solving a planning problem, (ii) interacting with the planning domain is typically prohibitively expensive and available recordings from the planning domain are sparse. To counter these challenges, we introduce our novel Goal-Conditioned Model-Augmented Deep Q-Networks algorithm (GCM-DQN). The intuition behind GCM-DQN is to use gradient-based optimization on the surface of the Q-Function, instead of blunt estimators, to estimate the optimal parameters of an action in a state. In combination with a goal-conditioning of the DQN, and a state transition model, this allows us to find plans for planning problems in planning domains with parametrized action spaces. Our algorithm outperforms state-of-the-art Reinforcement Learning algorithms for planning in parametrized action spaces.

1 Introduction

Planning, the combinatorial problem of finding a sequence of actions that transitions an initial state into a goal state, is a fundamental problem in many real-world applications and AI (Ghallab, Nau, and Traverso 2016; Sutton and Barto 2018). Conventional planning and Reinforcement Learning methods typically feature either purely discrete action spaces (i.e. a finite set of actions, like moving up, down, left, or right in a grid world) *or* purely continuous action spaces (i.e. an infinite set of actions, like controlling the acceleration of a cart on a slope) (Sutton and Barto 2018; Masson, Ranchod, and Konidaris 2016). However, many real-world problems feature *parametrized action spaces*. In a parametrized action space, a finite set of actions is augmented by real-valued parameters, which influence the effects of the actions (Masson, Ranchod, and Konidaris 2016; Hausknecht and Stone 2016; Heesch, Ehrhardt, and Niggemann 2024). During planning in parametrized action spaces, a planner hence must not only select from the finite action set, but also real-valued parameters, to reach its goal (Masson, Ranchod, and Konidaris 2016). For example, consider injection molding, where there is a finite set of actions (e.g. close mold, inject, hold, cool, eject), which are each augmented by real-valued parameters (e.g. heating/cooling energy, velocity, pressure, etc.). Both the combinatorial aspect of finite action selection, e.g., injecting material before closing the mold would lead to a mess, as well as the parametrization aspect, e.g., injecting too cold material leads to poor surface characteristics of the molded product majorly, have a major influence on the molded product. Getting both aspects right is the task of planning in parametrized action spaces. Besides this simplified example, many other real-world problems, from robotics to factory planning, feature parametrized action spaces (Hausknecht and Stone 2016; Masson, Ranchod, and Konidaris 2016; Xiong et al. 2018; Ehrhardt, Heesch, and Niggemann 2024; Heesch et al. 2024).

There are two central challenges in solving planning problems in real-world parametrized action spaces: *(i)* Due to the continuous nature of the parameter space, there is an infinite number of action-parameter tuples a planner has to choose from in every state. This *infinite branching* of action-parameter tuples in every state poses a challenge for selecting the optimal action-parameter tuple (Wu, Say, and Sanner 2020). Typically, infinite branching is either countered by parameter estimators (Lillicrap et al. 2016), which have the risk of being imprecise, or search (Ma et al. 2023), which has the risk of being computationally expensive. *(ii)* Often there is no sufficient model of the planning domain available, interaction with the domain is prohibitively expensive or unsafe, and recorded data is scarce (Levine et al. 2020). Hence, solving planning problems typically, either requires a manually crafted, expensive, and error-prone planning domain model (Grand, Pellier, and Fiorino 2022; Heesch, Ehrhardt, and Niggemann 2024), or requires advanced Reinforcement Learning algorithms which can be trained offline, meaning without interaction with the planning domain, but strongly rely on the assumption that the distribution of the recorded data does not shift strongly from the application cases (Levine et al. 2020).

In this paper, we tackle the challenges of infinite branching and training data scarcity in real-world parametrized action spaces. Therefore, we propose to extend the well known Deep Q-Network (DQN) algorithm (Mnih et al. 2015). DQN uses a Neural Network to approximate the action value function, which returns the expected cumulative return of taking an action in a state. In combination with a greedy policy, DQN can solve even complex planning and control problems (Mnih

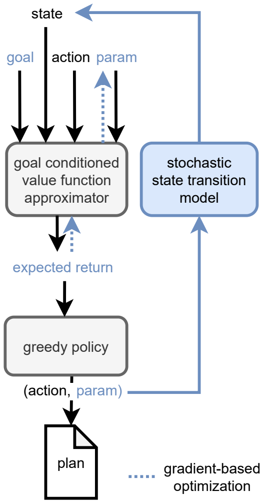
state
goal
action
param
goal conditioned value function approximator
stochastic state transition model
expected return
greedy policy
(action, param)
plan
gradient-based optimization

Figure 1: We propose the Goal-Conditioned Model-Augmented DQN (GCM-DQN) algorithm, an offline extension to DQN that allows for solving planning problems in domains with parametrized action spaces (novel extension are marked in blue). GCM-DQN takes an initial state and iteratively computes the optimal action to select via a goal-conditioned value function approximator. For estimating the optimal parameters, it uses a gradient-based optimization over the DQNs input. By greedily selecting the optimal action-parameter tuple, the next state can be computed with a stochastic state-transition model. The process stops, once a resulting state matches the goal-state.

et al. 2015). We propose to transfer DQN into a novel, offline and model-augmented Reinforcement Learning setup, which allows us to use it for solving planning problems in planning domains with parametrized action spaces (Masson, Ranchod, and Konidaris 2016) (cf. Figure 1). More precisely, we propose three extension to the DQN algorithm: *(a)* To tackle infinite branching, we introduce $paramOpt$, a novel gradient-based optimization algorithm, to efficiently find optimal parameters for a given action in a given state. *(b)* To make our algorithm applicable to unseen planning problems, we integrate a goal-conditioning to the DQN (Schaul et al. 2015). *(c)* To allow using the DQN for planning without interacting with the environment, we propose a novel state-transition model, which is trained along the DQN and allows for planning in deterministic and probabilistic domains. We reduce the amount of training data to fit the models, by employing Hindsight Experience Replay (Andrychowicz et al. 2017) and Conservative Q-Learning (Kumar et al. 2020).

As a result, we present our Goal-Conditioned Model Augmented DQN algorithm (GCM-DQN). GCM-DQN is can be trained on a sparse dataset of recorded plans from a planning domain. It returns a DQN which can either be used as a policy in probabilistic scenarios, or in combination with the parallelly trained state transition model as planner for deterministic domains. In contrast to estimator or search-based algorithms for planning in parametrized action spaces, GCM-DQN converges quickly to optimal parameters due to the gradient-based parameter optimization. The main contributions of our paper are:

- $paramOpt$ novel gradient-based optimization algorithm to efficiently counter infinite branching in planning domains with parametrized action spaces.
- A novel integration of $paramOpt$, goal-conditioning, and a novel state-transition model into DQN to allow harnessing it for planning.
- A systematic and comprehensive evaluation of our approach against state-of-the-art Reinforcement Learning paradigms for parametrized action spaces.

2 Related Work

In Deep Reinforcement Learning, there are two directions when handling parameterized action spaces: Using Neural Networks as estimators that suggest parameters for actions, and using search or optimization to find optimal parameters for an action. Typically, policy network approaches are grounded in the Deep Deterministic Policy Gradient (DDPG) paradigm (Lillicrap et al. 2016). DDPG is an Actor-Critic approach, in which the actor is a deep policy network that, given a state, suggests actions and the critic is a deep Q-network that calculates the cumulative expected return of the suggested action and state. Using backpropagation over both networks allows for adapting their weights to converge to an optimal policy- and Q-network. To solve planning problems in parametrized action spaces, Hausknecht and Stone (2016) extendeded the DDPG paradigm by expanding the deep policy network with an additional non-binary output for suggesting parameters values, resulting in the P-DDPG algorithm. Fan et al. (2019) propose a similar approach. They use individual separate heads for selecting an action from the finite action set, and individual separate heads for estimating its numerical parameters (Fan et al. 2019). However, both approaches neglect that there is a dependency between an action and the numerical parameters (Li et al. 2021). Hence, Li et al. (2021) proposed to encode the finite set of actions and numerical parameters into a joint latent representation space on which the policy operates, and from which discrete and continuous components are decoded for interaction with the environment. While the introduced approaches can handle parametrized action spaces, they remain restricted to online settings, which require the agent to interact directly with the environment, and are not well suited to an offline scenario with only little available training data.

Optimization or search-based approaches typically follow a value-based paradigm, in which a greedy policy selects the action-parameter tuple with the highest expected return. While methods like (Tavakoli, Pardo, and Kormushev 2018) use a divide-and-conquer approach for complex actions-parameter tuples that operates on a joint latent representation, Xiong et al. (2018) uses a separate parameter estimation network which feeds into a DQN, forming a parametrized DQN or P-DQN. Thereby, they can select a discrete action directly using a greedy policy and do not rely on a continuous relaxation of the discrete action components (as, e.g., Hausknecht and Stone (2016)) (Xiong et al. 2018).

Finally, Ma et al. (2023) uses an evolutionary optimization algorithm for estimating an optimal action from a continuous action space. While such approaches can also be adapted to parametrized action spaces, they are computationally expensive due to the uninformed optimization paradigm.

In contrast to typical Reinforcement Learning tasks, e.g., like control, the reward structure in planning problems sparse. Typically, the reward for solving a planning problem is formalized by a single reward signal upon reaching the goal state. This sparse reward signal hence is exclusively dependent on the goal state, and changes for planning problems with diverging goal states. To make Reinforcement Learning agents applicable to altering reward functions, Schaul et al. (2015) introduced Universal Value Function Approximators. Universal Value Function Approximators condition the value function approximator on an embedding of the goal state, hence making it generalizable across altering planning problems within the same domain (Schaul et al. 2015). Other methods for countering sparsity of reward signals, especially in offline settings, include data augmentation, such as Hindsight Experience Replay (Andrychowicz et al. 2017), or regularization in training by additional loss terms, such as Conservative Q-Learning (Kumar et al. 2020).

3 Formalization

Reinforcement Learning follows the assumption that there is an underlying MDP within all planning domains. As we focus on planning problems in parametrized action spaces, we consider Parametrized Action Markov Decision Processes (PAMDP) (Masson, Ranchod, and Konidaris 2016).

3.1 Parametrized Action Markov Decision Processes

PAMDPs extend continuous Markov Decision Processes by introducing a hybrid, so-called, parametrized action space. They can be formalized as a tuple

$$\langle \mathcal{S}, A, \Psi, \mathcal{T}, \mathcal{R}, \gamma \rangle, \tag{1}$$

where $\mathcal{S} \subseteq \mathbb{R}^n$ is the continuous state space, $A = \{a_0, ..., a_k, ..., a_K\}, K \in \mathbb{N}$ is a finite set of actions, in which each action a_k is extended by a continuous parameter space $\Psi_k \in \mathbb{R}$ and the union of all parameter spaces is given as $\Psi = \bigcup_{k=1}^{K} \Psi_k$. Together they form the parametrized action space

$$\mathcal{A} = \bigcup_{a_k \in A} \{(a_k, \psi_k) | \psi_k \in \Psi_k\}. \tag{2}$$

$\mathcal{T}$ is the transition function $\mathcal{T} = P(s_{t+1}|s_t, a_t, \psi_t)$ that describes the probability of transitioning into state $s_{t+1} \in \mathcal{S}$ given state $s_t \in \mathcal{S}$, action $a_t \in A$ and a parameter $\psi_t \in \Psi$ at time t. $\mathcal{R}$ is the reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ that returns the scalar reward r when transitioning from s_t into s_{t+1} using an action a_t, and $\gamma \in \mathbb{R}$ a discount factor. We will further refer to $\mathcal{T}$ as the dynamics of the MDP.

As the transition dynamics in real-world PAMDPs can grow very complex, large models and large datasets are needed to properly capture them. Leveraging on the parametrized action spaces, we propose to manage the complexity of real-world dynamics by a modular factorization of the parametrized action space. Therefore, we split $\mathcal{T}$ into a finite set $\mathcal{T}_d$ of K transition functions $\mathcal{T}_{a_k}$, which each are related to one individual action a_k each:

$$\begin{aligned} \mathcal{T}_d = \{\mathcal{T}_{a_k} | \mathcal{T}_{a_k} = P_{a_k}(s_{t+1}|s_t, \psi_k), \\ \psi_k \in \Psi_k, \;\; k = 1, ..., K\} \end{aligned} \tag{3}$$

This allows us to model the transition dynamics for each action in one individual model $f_{a_k} \approx \mathcal{T}_{a_k}$, reducing the complexity of the modeling problem, while overall not affecting the PAMDP dynamics. We can denote the collection of all f_{a_k} as $\mathcal{F} = \{f_{a_k}\}_{k=1}^{K}$. During planning, we can infer state transitions by sampling from the transition models

$$s_{t+1} \sim f_{a_k}(s_t, \psi_t). \tag{4}$$

In deterministic scenarios, the transition probabilities of $\mathcal{T}_{a_k}$ collapse to a Dirac delta distribution, which effectively turns f_{a_k} into a deterministic function

$$f_{a_k}(s_t, \psi_t) = s_{t+1}. \tag{5}$$

3.2 Describing Planning Problems with PAMDPs

Planning describes the task of finding a sequence $\tau = \{(a_t, \psi_t)\}_{t=0}^{T-1}$ of T action-parameter tuples, that transition an initial state s_0 into a goal state $g \in G \subset \mathcal{S}$. Hence, a planning problem in a PAMDP can be denoted as

$$\langle \mathcal{S}, A, \Psi, \mathcal{F}, \mathcal{R}_G, \gamma, s_0, G \rangle, \tag{6}$$

where $\mathcal{R}_G$ is a goal conditioned, sparse reward function

$$\mathcal{R}_G(s) = \begin{cases} r, & \text{if } s \in G \\ 0, & \text{else} \end{cases} \tag{7}$$

, with the numerical reward value $r \in \mathbb{R}$.

Reinforcement Learning typically solves planning problems by iteratively applying a policy π on the planning problem. Hence, a plan can be seen as a trajectory-level instantiation of a policy. A policy in a PAMDP is a mapping from the current state s_t and goal state g to an action-parameter tuple. For deterministic planning domains, the mapping is a function $\pi_{\text{(det)}}(s_t, g) = (a, \psi)$, For probabilistic planning domains, the mapping is a conditional distribution $\pi((a, \psi)|s_t, g)$, where $s_t \in \mathcal{S}, \quad g \in G, \quad a, \psi \in \mathcal{A}$.

For deterministic domains, the solution of a planning problem is a plan τ, which, when executed from s_0, reaches a $g \in G$. For probabilistic domains, the solution of a planning problem is a proper policy π. A proper policy optimizes the discounted return of the planning problem and results in a goal state $g \in G$. The sequence of actions-parameter tuples selected by the policy during execution forms a plan τ.

4 Solution

In this section, we introduce our GCM-DQN algorithm. GCM-DQN tackles the challenges of infinite branching, prohibitively expensive domain interactions, and data scarcity in real world planning domains with parametrized action spaces. The intuition of GCM-DQN is to leverage on the differentiability of a DQN (Mnih et al. 2015) during planning for finding the optimal parameters and actions via gradient-based optimization, instead of using estimators or search. Therefore, we add three extensions to the DQN algorithm (Mnih et al. 2015): *(a)* To tackle the problem of infinite branching, we introduce the $paramOpt$ algorithm, a gradient-based optimization algorithm inspired by (Wu, Say, and Sanner 2017; Heesch et al. 2024), for finding an (leastwise locally) optimal action-parameter tuple during planning (cf. Section 4.3). *(b)* To make GCM-DQN applicable to any planning problem within the planning domain, we introduce a goal-conditioning to the DQN, as proposed in (Schaul et al. 2015). We tackle data scarcity in training the goal-conditioned DQN, by using Hindsight Experience Replay (Andrychowicz et al. 2017) and Conservative Q-Learning (Kumar et al. 2020) (cf. Section 4.2). *(c)* Finally, to counter prohibitively expensive domain interaction, we propose a novel state transition model which is parallelly trained to the DQN on the same dataset (cf. Section 4.4), allowing to simulate state transitions without any interaction with the planning domain.

By combining the three proposed extensions, we result in our novel GCM-DQN algorithm (cf. Section 4.1). GCM-DQN can operate in planning domains with parametrized action spaces. It can either be used as a policy for probabilistic planning domains or, when using the state transition model, as a planner for deterministic planning domains (cf. Figure 2).

4.1 Planning with Goal-Conditioned Model-augmented Deep Q-Networks

In this Section we provide an overview on the GCM-DQN algorithm (cf. Algorithm 1 and Figure 2). In its essence GCM-DQN is a goal-conditioned greedy policy, which is trained in an offline setting. Hence, the first step includes training the DQN Q_θ and the state transition models $\mathcal{F} = \{f_{a_k} | k = 1, ..., K\}$ using a dataset of recorded plans $\mathcal{D}$. During planning, GCM-DQN uses the $paramOpt$ algorithm (cf. Algorithm 2) on Q_θ to calculate the optimal parameter $\tilde{\psi}_k^*$ for every action. To guide the selection of optimal action-parameter tuples, we calculate a decision value δ_k for each action. δ_k includes the Q value, the weighted variance of the succeeding state var_k (cf. Equation 18), and a potential based shaping factor ω (Ng, Harada, and Russell 1999):

$$\delta_k = Q_\theta(s_t, a_{k_t}, \tilde{\psi}_{k_t}^*, g) + \lambda_1 \text{var}_{k_t} + \lambda_2 \omega, \quad \lambda_1, \lambda_2 \in \mathbb{R}. \tag{8}$$

Using δ_k instead of the pure Q values counters, the selection of actions which would lead into non-permissible states, e.g., colliding with boundaries. A greedy policy π_{greedy} then picks the highest δ_k and adds the corresponding action-parameter $(a_k, \tilde{\psi}_k)$ tuple to the plan. By sampling from the associated state transition model f_{a_k} the next state s_{t+1} can be inferred and passed to the next iteration. The iterations stop, when s_{t+1} becomes a state within G (or $G \pm \varepsilon$, where ε is an error margin). In cases, in which there is no solution to the planning problem, a stopping criterion ζ can be introduced to bound the maximum number of iterations. The complete GCM-DQN algorithm is outlined in Algorithm 1. The following section introduce the extensions of GCM-DQN in detail.

Algorithm 1: GCM-DQN during planning

```
Require : $\mathcal{D}$          // recorded plans
          $s_0$          // starting state
          $G$          // goal state(s)
          $\varepsilon, \zeta$     // tolerance, max steps
$Q_\theta \leftarrow$ TRAINGCMDQN($\mathcal{D}$)   // cf. Section 4.2
$\mathcal{F} \leftarrow$ TRAINSTM($\mathcal{D}$)    // cf. Section 4.4
$\tau \leftarrow \emptyset$
$s \leftarrow s_0$
for $t \leftarrow 0$ to $\zeta - 1$ while $s \notin G \pm \varepsilon$ do
    $(a^*, \psi^*) \leftarrow \arg\max_{a \in A}$ DECISIONVALUE($Q_\theta(s, a,$ PARAMOPT$), s$)
    // cf. Section 4.3
    append $(a^*, \psi^*)$ to $\tau$
    $s \leftarrow s_{t+1} \sim f_{a^*}(s, \psi^*)$
return $\tau$                    // trajectory $((a_0, \psi_0), (a_1, \psi_1), \dots)$
```

4.2 Goal-Conditioned DQN for Parametrized Action Spaces

In this section, we describe our adaptions to DQN to allow using it for planning in planning domains with parametrized action spaces. We achieve this by including the goal state into the input of the DQN, thereby conditioning it on the goal state, and handling continuous per-action parameters via gradient-based optimization.

The original DQN uses a Neural Network to approximate the action value function $Q(s, a)$ of a domain (Mnih et al. 2015), which describes the expected discounted return for taking action a in state s, and satisfies the Bellman equation in the optimal case

$$Q(s_t, a_t) = \mathop{\mathbb{E}}_{s_{t+1} \sim P(\cdot | s_t, a_t)} [\mathcal{R}(s_t) + \gamma \max_{a_{t+1} \in A} Q(s_{t+1}, a_{t+1})]. \tag{9}$$

For our application, we expand the classical Q-function's input with the goal state of the planning problem (Schaul et al. 2015) and the parametrized actions, so that

$$Q(s, a) \rightsquigarrow Q(s, a_k, \psi_k, g), \tag{10}$$

where $a_k \in A$ is an action from the finite action set, ψ_k is an associated continuous parameter, and g is the goal state of the planning problem. For our updated Q-function, the Bellman equation becomes

$$\begin{aligned} Q(s_t, a_{k_t}, \psi_{k_t}, g) = &\mathop{\mathbb{E}}_{s_{t+1} \sim P(\cdot | s_t, a_{k_t}, \psi_{k_t})} [\mathcal{R}_g(s_t) \\ &+ \gamma \max_{k_{t+1} \in K} \mathop{\arg\max}_{\psi_{k_{t+1}} \in \Psi_k} Q(s_{t+1}, a_{k_{t+1}}, \psi_{k_{t+1}}, g)]. \end{aligned} \tag{11}$$

As the inner maximization over $\psi_{k_{t+1}}$ is non-convex when Q is approximated by a Neural Network, solving it is intractable. Hence, we propose to leverage on global optimization algorithms for finding leastwise local optima for ψ and solve Equation 11 in two steps. In the first step, we find optimal action-parameter tuples for each action in the current

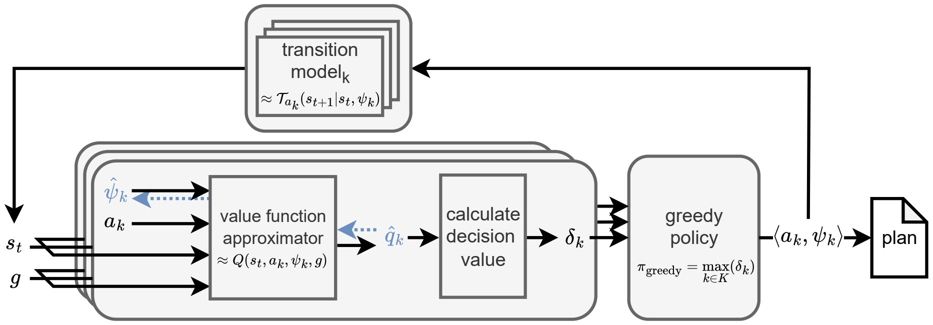
transition model$_k$
$\approx \mathcal{T}_{a_k}(s_{t+1}|s_t, \psi_k)$
$\hat{\psi}_k$
a_k
s_t
g
value function approximator
$\approx Q(s_t, a_k, \psi_k, g)$
$\hat{q}_k$
calculate decision value
δ_k
greedy policy
$\pi_{\text{greedy}} = \max_{k \in K}(\delta_k)$
$\langle a_k, \psi_k \rangle$
plan

Figure 2: We introduce the GCM-DQN algorithm. A goal-conditioned and model-based DQN approach for solving planning problems in parametrized action spaces. GCM-DQN leverages on gradient-based optimization during execution (marked in blue) to find (leastwise locally) optimal action-parameter tuples and uses a modular state transition model to sample successor states.

state,

$$\psi_k^* = \arg\max_{\psi_k \in \Psi_k} Q(s, a_k, \psi_k, g) \quad \forall\, k \in K, \tag{12}$$

using projected gradient ascent (cf. Section 4.3). As we cannot guarantee a global optimum, we denote the resulting parameters with $\tilde{\psi}_{k_{t+1}}^*$. This first step allows us to reformulate Equation 11 as

$$\begin{aligned} Q(s_t, a_{k_t}, \psi_{k_t}, g) = \mathop{\mathbb{E}}_{s_{t+1} \sim P(\cdot|s_t, a_{k_t}, \psi_{k_t})} [\mathcal{R}_g(s_t) \\ + \gamma \max_{k_{t+1} \in K} Q(s_{t+1}, a_{k_{t+1}}, \tilde{\psi}_{k_{t+1}}^*, g)], \end{aligned} \tag{13}$$

which resembles the Bellman equation with a goal conditioning and an approximate inner maximization.

We *train* our goal-conditioned DQN Q_θ, with parameters $\theta \in \mathbb{R}$, for parametrized action spaces in an offline Reinforcement Learning setup, to cater the restrictions on of prohibitively expensive domain interactions in real-world planning domains. Therefore, we assume a training dataset of recorded plans $\mathcal{D} = \{\tau_j\}_{j=0}^J$. A major problem in offline Reinforcement Learning is the distributional shift between training data and the application domain (Levine et al. 2020). We counter this problem by augmenting $\mathcal{D}$ with Hindsight Experience Replay (Andrychowicz et al. 2017), and Conservative Action Sampling (Chebotar et al. 2021). Hindsight Experience Replay augments the available dataset by sampling sub-traces from the recorded plans, relabeling the final state as the goal state (Andrychowicz et al. 2017). Conservative Action Sampling also samples sub-trances from the recorded plans, however, labeling their final state as miss, therefore artificially creating negative samples for the dataset (Chebotar et al. 2021). Using both augmentation techniques, results in the datasets $\tilde{\mathcal{D}}$ (Andrychowicz et al. 2017) and $\bar{\tilde{\mathcal{D}}}$ (Chebotar et al. 2021).

Following (Mnih et al. 2015) we use an off-policy training setup, using an online network Q_θ and a target network Q_{θ^-}. During training, only the weights of Q_θ are updated via gradient descent, whereas the weights of Q_{θ^-} are copied from Q_θ every η steps. We use a composite loss function

$$\mathcal{L}_{CQL} = \mathcal{L}_Q + \mathcal{L}_P \tag{14}$$

consisting of the squared TD-loss $\mathcal{L}_Q$ (Mnih et al. 2015) and a conservative penalty term $\mathcal{L}_P$ (Kumar et al. 2020). The conservative penalty term $\mathcal{L}_P$ helps to regularize Q_θ to overestimate Q-values of unseen or underrepresented actions (Kumar et al. 2020). We denote the squared TD-loss as

$$\begin{aligned} \mathcal{L}_Q = \mathop{\mathbb{E}}_{(s_t, a_{k_t}, \psi_{k_t}, r_t, s_{t+1}) \sim \bar{\tilde{\mathcal{D}}}} [r_t \\ + \gamma(1 - d_t) \max_{k_{t+1} \in K} Q_{\theta^-}(s_{t+1}, a_{k_{t+1}}, \tilde{\psi}_{k_{t+1}}^*, g) \\ - Q_\theta(s_t, a_{k_t}, \psi_{k_t}, g)]^2, \end{aligned} \tag{15}$$

where $d_t \in \{0, 1\}$ indicates whether the plan at time t, so that $d_t = 1,$ if $s_{t+1} \in G$. Following (Kumar et al. 2020), we formulate the conservative penalty term as

$$\begin{aligned} \mathcal{L}_P = \alpha[\log(\sum_{k \in K} \frac{1}{M} \sum_{m=1}^{M} \exp(Q_\theta(s_t, a_{k_t}, \psi_{k_t}^{(m)}, g))) \\ - Q_\theta(s_t, a_{k_t}, \tilde{\psi}_{k_t}^*, g)]. \end{aligned} \tag{16}$$

where α is the trade-off factor between Bellman-fit and conservatism, $K = |A|$ is the number of discrete actions, and M is the number of parameter samples per action used in the log-sum-exp penalty. For our offline training, we draw M samples $\psi_{k_t}^{(m)}$ uniformly from the empirical pool of parameters for action a_k to approximate $\int_\psi e^{Q_\theta(s_t, a_{k_t}, \psi_{k_t}, g)} d\psi$.

Regarding $\mathcal{D}$, three edge cases must be considered: *(i)* $\mathcal{D}$ including no data, *(ii)* $\mathcal{D}$ including little data, and *(iii)* $\mathcal{D}$ including infinite data. In case *(i)*, where no data is available, Q_θ cannot be trained. Hence, data must be collected by random exploration or through sampling state transitions from the domain. Case *(ii)* describes the normal operation of GCM-DQN. We note that the higher the variance in the dataset, the better the approximation of Q_θ to the real Q. Case *(iii)* describes a special case, where all data are available. Given a large enough θ, this allows Q_θ to fit Q exactly.

4.3 Gradient-based Parameter Estimation

For finding the optimal parameters for an action in a given state, we propose to leverage on the differentiability of the DQN and use gradient ascent in a nested optimization loop for finding optimal parameters for a given action (cf. Equation 12). Therefore, we introduce the *paramOpt* algorithm, which draws inspiration from (Kingma et al. 2014) and its applications in (Wu, Say, and Sanner 2020; Heesch et al. 2024).

The idea of *paramOpt* is to use the same algorithm, which is used to adapt the weights of Q_θ during training, for finding the optimal action-parameter tuples during execution. However, instead of optimizing the weights of the Q_θ, we optimize the parameter component ψ of its input. Therefore, we initialize the parameter component ψ with a guess $\hat{\psi}$, e.g., random numbers, zeros, or values from $\mathcal{D}$. After calculating $Q_\theta(s, a, \hat{\psi}, g)$, we use backpropagation to derive the gradient with respect to $\hat{\psi}$, allowing us to use gradient ascent with a learning rate β to update $\hat{\psi}$ in a direction which increases the Q-value. The optimization stops after the updates of the Q-value, Δ_Q, fall below a threshold ξ, returning the last update of $\hat{\psi}$ as $\tilde{\psi}^*$. Algorithm 2 summarizes our parameter estimation loop through input optimization.

Algorithm 2: PARAMOPT Gradient-Based Parameter Optimization

```
Require: $s, a, g$          // state, action, goal
         $Q_\theta$         // goal-conditioned DQN
         $\beta$                  // learning rate
         $\xi$               // stopping threshold
$\hat{\psi} \leftarrow$ init()               // initial guess
$\Delta_Q \leftarrow +\infty$
$Q^{(\text{prev})} \leftarrow -\infty$
while $\Delta_Q > \xi$ do
    $g_\psi \leftarrow \nabla_\psi Q_\theta(s, a, \hat{\psi}, g)$   // backprop wrt. parameters
    $\hat{\psi} \leftarrow \text{clip}_{[\psi_{\min}, \psi_{\max}]}(\hat{\psi} + \beta\, g_\psi)$
        // projected gradient ascent
    $Q^{(\text{val})} \leftarrow Q_\theta(s, a, \hat{\psi}, g)$        // caclulate action value
    $\Delta_Q \leftarrow Q^{(\text{val})} - Q^{(\text{prev})}$
    $Q^{(\text{prev})} \leftarrow Q^{(\text{val})}$
return $\tilde{\psi}^* \leftarrow \hat{\psi}$     // optimized parameter
```

As we are using gradient ascent as optimization algorithm over the DQN, we cannot guarantee to find the true global optimum ψ^*. This is due to the non-convex shape of Q_θ. The result of the optimization hence can be strongly dependent on the initialization of $\hat{\psi}$ and the learning rate β. As there are different options for initialization, e.g., zeros, ones, or random numbers, we suggest incorporating prior knowledge from the dataset, in the form of estimators like the mean over observed parameter settings as starting guesses.

Additionally, parameters are typically bound to value ranges, e.g., a temperature cannot fall below 0 Kelvin. To incorporate this, we use projected gradient ascent (Calamai and Moré 1987) during optimization, effectively clipping values that exceed the bounds. As one naïve solution for retrieving the bounds, we suggest iterating through the dataset $\mathcal{D}$ and collecting minima and maxima of each parameter.

4.4 Learning State Transition Dynamics

In real-world planning problems, directly interacting with the planning domain to predict action effects is rarely possible or prohibitively expensive (Levine et al. 2020). Hence, planning requires a model of the state transition dynamics (Ghallab, Nau, and Traverso 2016) which maps a current state s_t and parameters ψ_t to a successor state. In deterministic domains this is a function $f(s_t, \psi_t) = s_{t+1}$ (cf. Eq. 5); in probabilistic domains it is a conditional distribution $p(s_{t+1} \mid s_t, \psi_t)$ from which s_{t+1} is sampled (cf. Eq. 4).

Following the modular per–action factorization of PAMDP dynamics (cf. Eq. 3), we learn one transition model action, $\mathcal{F} = \{f_{a_k}\}_{k=1}^{K}$, each predicting the next state for action a_k given (s_t, ψ_t). Thereby, we use the same dataset $\mathcal{D}$, which is also used for training the DQN.

We propose to capture the stochasticity of probabilistic planning domains with a novel conditional latent-variable state transition model, inspired by (Sohn, Lee, and Yan 2015). Thereby, each per-action model comprises an encoder e_k and a decoder d_k part.

During training , the encoder processes the input s_t, ψ_t, and s_{t+1} into the parameters μ and σ of a latent posterior $q_e(z|s_t, s_{t+1}, \psi_t)$. Using the reparametrization trick, it samples $z = \mu + \sigma \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$. The decoder d_k reconstructs s_{t+1} from s_t, ψ_t, and z under a standard normal prior $p_d(z) = \mathcal{N}(0, I)$. As training criterion, we minimize the negative Evidence Lower Bound,

$$\begin{aligned} \mathcal{L} = -\underset{q_e(z|s_t, s_{t+1}, \psi_t)}{\mathbb{E}}&[\log p_d(s_{t+1}|s_t, \psi_t, z)] \\ &+ D_{\text{KL}}\big(q_e(z|s_t, s_{t+1}, \psi_t) | \, p_d(z)\big), \end{aligned} \tag{17}$$

where D_{KL} denotes the Kullback–Leibler divergence.

During planning , the encoder is discarded and only d_k is further used. Given the current state s_t and parameters $\tilde{\psi}_t^*$ (estimated with *paramOpt*), we draw $z \sim \mathcal{N}(0, I)$ and decode samples $\hat{s}_{t+1} = d_k(s_t, \tilde{\psi}_t^*, z)$. Boundaries and non-permissible states can be detected by analyzing the scalar variance var of $\hat{s}_{t+1}^{(n)}$ when sampling the z vector n times:

$$\begin{aligned} \text{var} &= \frac{1}{n-1} \sum_{i=1}^{n} ||\hat{s}_{t+1}^{(i)} - \bar{s}_{t+1}||^2, \\ \bar{s}_{t+1} &= \frac{1}{n} \sum_{i=1}^{n} \hat{s}_{t+1}^{(i)}, \end{aligned} \tag{18}$$

A high variance indicates a high predictive uncertainty in $\hat{s}_{t+1}$, which indicates boundaries or non-permissible states, like obstacles.

For deterministic domains, the stochastic latent z can be omitted and d_k reduces to a standard Multilayer Perceptron.

5 Evaluation

We evaluate our GCM-DQN algorithm empirically against offline versions of state-of-the-art Reinforcement Learning baselines for solving planning problems in parametrized action spaces (Hausknecht and Stone 2016; Xiong et al. 2018). Therefore, we used domains with navigation problems and domains from the international planning competition's (IPC) reinforcement learning track (Taitler et al. 2024). As performance metrics, we use the rate of successfully solved planning problems from a set of unseen planning problems, and a success-weighted distance of the found trajectories to optimal trajectories. We hypothesize that $(H1)$ GCM-DQN shows a higher performance than the baselines, when trained on the same limited dataset of plans $\mathcal{D}$, and $(H2)$ GCM-DQN longer maintains a higher performance than the baselines, when systematically reducing the number of samples in $\mathcal{D}$. Additionally, we perform an ablation study on the three extensions of GCM-DQN: the goal-conditioning, $paramOpt$, and the state-transition models (cf. Appendix C).

For setting up our experiments, we followed the experimental design guidelines for empirical Machine Learning research by (Vranješ et al. 2024). We generated samples for the datasets $\mathcal{D}$ by running either an A^* search or JaxPlan (Gimelfarb, Taitler, and Sanner 2024) for randomly initialized planning problems of the chosen planning domains. We used Optuna (Akiba et al. 2019) for hyperparameter optimization of GCM-DQN and the baselines to allow for a fair comparison. We repeated all experiments on eight different seeds to rule out lucky initializations. All code and datasets for replicating the experiments can be found under https://anonymous.4open.science/r/gcmdqn-7CA6. The full experimental setup with domain descriptions and metrics can be found in Appendix A.

5.1 Evaluating GCM-DQN's Performance

For evaluating the performance of GCM-DQN in comparison to the baselines, we created a training dataset $\mathcal{D}$ of 128 solved planning problems and a test dataset of 100 solved problems per domain. The results are summarized in Table 1. For the navigation domains, our results indicate that GCM-DQN shows a higher mean planning success rate over the eight different seeds than the baselines, when trained on a limited dataset of 128 plans. For the IPC domains, either P-DDPG or GCM-DQN show the highest performance, with only narrow differences. As the IPC domains have a stronger emphasis on the parametrization than on the combinatorial action selection it is expectable that the Actor-Critic approach performs well in the IPC domains, while underperforming in the navigation domains (and vice-versa for P-DQN). Overall, all algorithms show declining performance with increasing complexity of the planning domains. Additionally, we observed a lower variance due to different seeds for GCM-DQN. Full results, inlcuding the success-weighted planning distance, can be found in Appendix B.

5.2 Evaluating GCM-DQN's Performance with Succeedingly Scarce Data

The application scenario for GCM-DQN is solving planning problems under circumstances where only little data is available and interactions with the environment are not possible. For evaluating the behavior of GCM-DQN on scarce data, we trained the GCM-DQN and the baselines on succeedingly less samples in $\mathcal{D}$. Therefore, we created subsets of $\mathcal{D}$ containing $\{64, 32, 16, 8, 4, 2\}$ samples and trained GCM-DQN and the baselines on the hyperparameter settings from above. For each algorithm and dataset, we repeated the procedure on eight different seeds and evaluated the algorithms again on a test set of 100 unseen planning problems. Figure 3 shows the results for the navigation and IPC domains.

We hypothesized that GCM-DQN maintains a higher performance under progressive sample reduction compared to the baseline methods. For the navigation domains, this holds true in the most cases. GCM-DQN shows a continuous increase in planning success rates, when increasing the number of plans in the training dataset. In the circle domain P-DQN shows better performance between 2 and 16 samples. However, P-DQN shows a higher variance across the different seeds in comparison to GCM-DQN.

For the IPC domains, GCM-DQN shows a similar performance as the P-DDPG algorithm. This indicates that for combinatorial, as well as parametrization problems, GCM-DQN performs well, however being outperformed in the HVAC and PowerGen domains by P-DDPG (by a slight margin). Yet, it is noteworthy that P-DDPG shows a high variance across the different seeds, while GCM-DQN remains com-

Table 1: The table shows the mean success rate $\rho\pm$ standard deviation across eight seeds for our GCM-DQN algorithm and the baselines over different navigation and IPC planning domains.

domain name	P-DQN (Xiong et al. 2018)	P-DDPG (Hausknecht and Stone 2016)	GCM-DQN (ours)
navigation – bars	0.7852 ± 0.2063	0.1952 ± 0.0049	**0.7922** ± **0.1141**
navigation – circle	0.8466 ± 0.1579	0.0653 ± 0.0053	**0.9375** ± **0.0250**
navigation – squeeze	0.5351 ± 0.1624	0.1326 ± 0.1429	**0.9405** ± **0.0308**
navigation – cross	0.7405 ± 0.1203	0.0680 ± 0.0165	**0.8497** ± **0.1171**
IPC – HVAC – instance0	0.1484 ± 0.2209	0.6045 ± 0.2212	**0.6669** ± **0.1687**
IPC – HVAC – instance1	0.5029 ± 0.0058	**0.5410** ± **0.0354**	0.5273 ± 0.0239
IPC – HVAC – instance2	0.4472 ± 0.0055	**0.4492** ± **0.0072**	0.4492 ± 0.0055
IPC – HVAC – instance3	0.1171 ± 0.0011	0.1221 ± 0.0058	**0.1299** ± **0.0102**
IPC – PowerGen – instance1	0.0000 ± 0.0000	**0.3691** ± **0.1268**	0.2910 ± 0.0147
IPC – PowerGen – instance2	0.0000 ± 0.0000	0.0000 ± 0.0000	**0.1171** ± **0.0263**
IPC – Reservoir – instance1	0.0009 ± 0.0027	0.5918 ± 0.1398	**0.6796** ± **0.1048**

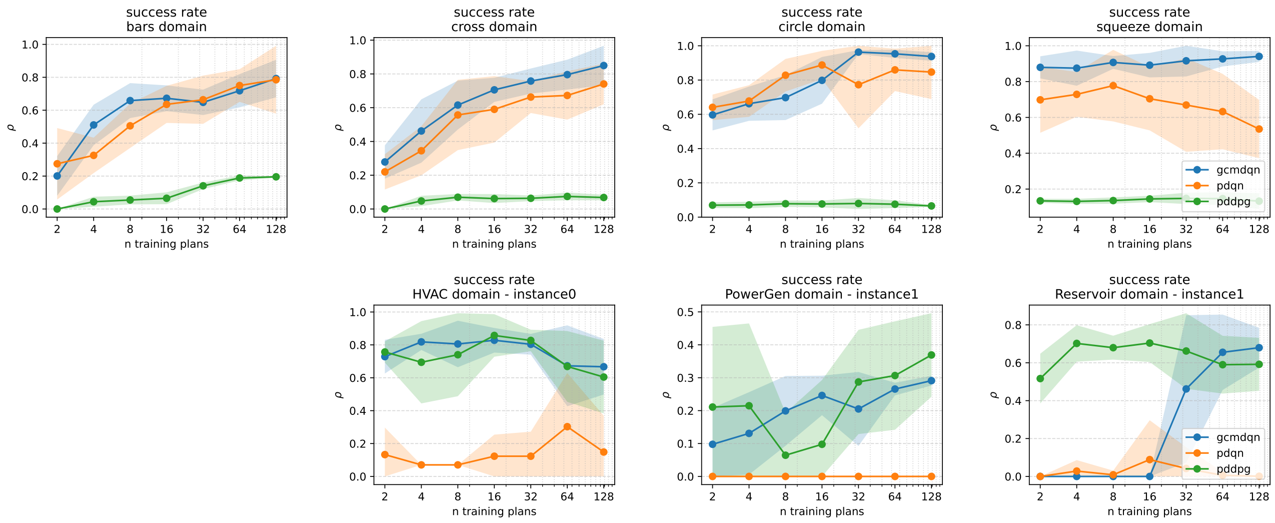

Figure 3: The figure shows the success rate for the navigation and IPC domains, when altering the number of plans in the training dataset. We repeated the experiments on eight different seeds. The shaded areas show the variance of the differently seeded runs.

paratively stable. In the Reservoir domain, we observed a collapse of GCM-DQN when having only little available samples, however surpassing even P-DDPGs performance when 64 or more traninig plans are in $\mathcal{D}$.

6 Discussion

Architectural Limitations of GCM-DQN Given the architecture we chose for our GCM-DQN algorithm, there are inherent limitations. Our gradient-based $paramOpt$ function for estimating the parameters for actions can converge to local optima in the Q-function. Especially in complex, non-convex Q-functions, this poses a serious problem. Mitigation strategies could include ensemble approaches with differently seeded optimizers, multi-start optimization with different initial guesses, or a combination of both. Additionally, in essence, our GCM-DQN algorithm is one-step greedy (though implicitly operating on the expected returns of the DQN). Especially for domains in which long plans are necessary to reach a goal, the sparse reward signal of training data might lead to wrong results. Using the transition model for look-ahead methods, like Monte Carlo Tree Search, might result in better performance of the planner. Alternatively, a hierarchical perspective where GCM-DQN plans between intermediate goals might lead to increased performance with longer plans. As some hyperparameters, e.g., the α weight of Conservative Q-Learning or the number of Conservative Actions Samples, have a strong impact on the performance and stability of the planner, including them as parameters in the training loop to dynamically adapt the conservatism or data augmentation level of the model during training, might be a future improvement.

Distributional Shift in Offline Reinforcement Learning One of the core challenges in Offline Reinforcement Learning is the distributional shift between the training data and application scenarios (Levine et al. 2020). Especially in the context of planning, the planner is likely to encounter state, action, parameter combinations that lie outside the support of the training data, which can lead to extrapolation errors. We mitigated this risk, using three mechanisms from the Offline Reinforcement Learning literature: Using Hindsight Experience Replay (Andrychowicz et al. 2017), Conservative Action Sampling (Chebotar et al. 2021), and Conservative Q-Learning (Kumar et al. 2020). Our results indicate that all measures improved training stability and planning performance.

7 Conclusion & Outlook

In this paper, we introduced the Goal-conditioned Model-augmented DQN algorithm (GCM-DQN), a model-augmented Offline Reinforcement Learning algorithm for planning in parametrized action spaces, where no model of the planning domain and only a limited dataset of recorded plans are available. GCM-DQN tackles three central challenges of planning with Reinforcement Learning in parametrized action spaces: *(i)* infinite branching of action-parameter tuples, *(ii)* goal-dependent reward functions, and *(iii)* substituting domain interactions with a model during planning time. To address the challenges, we introduce $paramOpt$, a novel gradient-based optimization algorithm over the DQN for finding the optimal parameters for an action in a state, a goal-conditioning of the DQN that allows for planning with changing and sparse reward functions, and a novel state transition model that allows to capture the inherent uncertainty in stochastic of probabilistic planning domains. We evaluate GCM-DQN against offline versions of two closely related algorithms. GCM-DQN shows higher performance than the baselines, especially in data scarce scenarios. Future work will include the refinement of GCM-DQNs architecture and its application on real-world industrial planning scenarios.

Acknowledgement

This research as part of the project LaiLa and EKI is funded by dtec.bw – Digitalization and Technology Research Center of the Bundeswehr, which we gratefully acknowledge. dtec.bw is funded by the European Union – NextGenerationEU. Any use of generative AI in this manuscript adheres to ethical guidelines of the IEEE for the use and acknowledgement of generative AI. Each author has made a substantial contribution to the work, using LLMs exclusively for language refinement, formatting purposes, and for non-substantial coding, e.g., for creating plots.

References

Akiba, T.; Sano, S.; Yanase, T.; Ohta, T.; and Koyama, M. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

Andrychowicz, M.; Wolski, F.; Ray, A.; Schneider, J.; Fong, R.; Welinder, P.; McGrew, B.; Tobin, J.; Abbeel, P.; and Zaremba, W. 2017. Hindsight Experience Replay. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Calamai, P. H.; and Moré, J. J. 1987. Projected gradient methods for linearly constrained problems. *Mathematical Programming*, 39(1): 93–116.

Chebotar, Y.; Hausman, K.; Lu, Y.; Xiao, T.; Kalashnikov, D.; Varley, J.; Irpan, A.; Eysenbach, B.; Julian, R. C.; Finn, C.; and Levine, S. 2021. Actionable Models: Unsupervised Offline Reinforcement Learning of Robotic Skills. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, 1518–1528. PMLR.

Ehrhardt, J.; Heesch, R.; and Niggemann, O. 2024. Learning Process Steps as Dynamical Systems for a Sub-Symbolic Approach of Process Planning in Cyber-Physical Production Systems. In *Artificial Intelligence. ECAI 2023 International Workshops*, 332–345. Cham: Springer Nature Switzerland. ISBN 978-3-031-50485-3.

Fan, Z.; Su, R.; Zhang, W.; and Yu, Y. 2019. Hybrid Actor-Critic Reinforcement Learning in Parameterized Action Space. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, 2279–2285. International Joint Conferences on Artificial Intelligence Organization.

Ghallab, M.; Nau, D.; and Traverso, P. 2016. *Automated Planning and Acting*. Cambridge University Press.

Gimelfarb, M.; Taitler, A.; and Sanner, S. 2024. JaxPlan and GurobiPlan: Optimization Baselines for Replanning in Discrete and Mixed Discrete-Continuous Probabilistic Domains. *Proceedings of the International Conference on Automated Planning and Scheduling*, 34: 230–238.

Grand, M.; Pellier, D.; and Fiorino, H. 2022. TempAMLSI: Temporal Action Model Learning Based on STRIPS Translation. *Proceedings of the International Conference on Automated Planning and Scheduling*, 32: 597–605.

Hausknecht, M.; and Stone, P. 2016. Deep Reinforcement Learning in Parameterized Action Space.

Heesch, R.; Cimatti, A.; Ehrhardt, J.; Diedrich, A.; and Niggemann, O. 2024. A Lazy Approach to Neural Numerical Planning with Control Parameters. In *European Conference on Artificial Intelligence (ECAI)*.

Heesch, R.; Ehrhardt, J.; and Niggemann, O. 2024. Integrating Machine Learning into an SMT-Based Planning Approach for Production Planning in Cyber-Physical Production Systems. In *Artificial Intelligence. ECAI 2023 International Workshops*, 318–331. Cham: Springer Nature Switzerland. ISBN 978-3-031-50485-3.

Ilharco, G.; Jain, V.; Ku, A.; Ie, E.; and Baldridge, J. 2019. General Evaluation for Instruction Conditioned Navigation using Dynamic Time Warping.

Kingma, D. P.; Mohamed, S.; Rezende, D. J.; and Welling, M. 2014. Semi-supervised Learning with Deep Generative Models. In Ghahramani, Z.; Welling, M.; Cortes, C.; Lawrence, N.; and Weinberger, K., eds., *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Kumar, A.; Zhou, A.; Tucker, G.; and Levine, S. 2020. Conservative Q-Learning for Offline Reinforcement Learning. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 1179–1191. Curran Associates, Inc.

Levine, S.; Kumar, A.; Tucker, G.; and Fu, J. 2020. Offline Reinforcement Learning: Tutorial, Review, and Perspectives on Open Problems.

Li, B.; Tang, H.; Zheng, Y.; Hao, J.; Li, P.; Wang, Z.; Meng, Z.; and Wang, L. 2021. HyAR: Addressing Discrete-Continuous Action Reinforcement Learning via Hybrid Action Representation.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2016. Continuous control with deep reinforcement learning.

Ma, Y.; Liu, T.; Wei, B.; Liu, Y.; Xu, K.; and Li, W. 2023. *Evolutionary Action Selection for Gradient-Based Policy Learning*, 579–590. Springer International Publishing. ISBN 9783031301117.

Masson, W.; Ranchod, P.; and Konidaris, G. 2016. Reinforcement Learning with Parameterized Actions. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1).

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533.

Ng, A. Y.; Harada, D.; and Russell, S. J. 1999. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, 278–287. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. ISBN 1558606122.

Schaul, T.; Horgan, D.; Gregor, K.; and Silver, D. 2015. Universal Value Function Approximators. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, 1312–1320. Lille, France: PMLR.

Sohn, K.; Lee, H.; and Yan, X. 2015. Learning Structured Output Representation using Deep Conditional Generative Models. In Cortes, C.; Lawrence, N.; Lee, D.; Sugiyama, M.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement learning: An introduction*. MIT press.

Taitler, A.; Alford, R.; Espasa, J.; Behnke, G.; Fišer, D.; Gimelfarb, M.; Pommerening, F.; Sanner, S.; Scala, E.; Schreiber, D.; Segovia-Aguas, J.; and Seipp, J. 2024. The 2023 International Planning Competition. *AI Magazine*, 45(2): 280–296.

Tavakoli, A.; Pardo, F.; and Kormushev, P. 2018. Action Branching Architectures for Deep Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1).

Vintsyuk, T. K. 1972. Speech discrimination by dynamic programming. *Cybernetics*, 4(1): 52–57.

Vranješ, D.; Ehrhardt, J.; Heesch, R.; Moddemann, L.; Steude, H. S.; and Niggemann, O. 2024. Design Principles for Falsifiable, Replicable and Reproducible Empirical Machine Learning Research. In *35th International Conference on Principles of Diagnosis and Resilient Systems (DX 2024)*, volume 125. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Wu, G.; Say, B.; and Sanner, S. 2017. Scalable Planning with Tensorflow for Hybrid Nonlinear Domains. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Wu, G.; Say, B.; and Sanner, S. 2020. Scalable Planning with Deep Neural Network Learned Transition Models. *Journal of Artificial Intelligence Research*, 68: 571–606.

Xiong, J.; Wang, Q.; Yang, Z.; Sun, P.; Han, L.; Zheng, Y.; Fu, H.; Zhang, T.; Liu, J.; and Liu, H. 2018. Parametrized Deep Q-Networks Learning: Reinforcement Learning with Discrete-Continuous Hybrid Action Space.

Appendix

A Experimental Setup

We used the following planning domains for our evaluation:

Navigation Domains The navigation domains feature two-dimensional path finding problems in a continuous space with obstacles (cf. Figure 4). The goal is to find a sequence of actions that lead from the start state to the goal state. There is a set of four actions - up, down, left, right - in which each action can be augmented with a plus minus ten-degree tilt. The step-width is fixed and collisions with the obstacles are forbidden. The planning problems are non-trivial, as the reward function is sparse and planners need to deal with linear and non-linear obstacles.

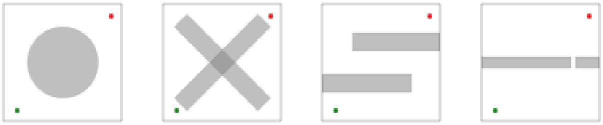

Figure 4: Evaluation domains from the **Navigation Domains**. From left to right, circle-domain, cross-domain, bars-domain, squeeze-domain. Exemplary start states are green and exemplary goal states are red.

IPC Domains The IPC domains feature domains from the International Planning Competition's Probabilistic and Reinforcement Learning Track from 2023 (Taitler et al. 2024). We picked the Reservoir, PowerGen and HVAC domains (cf. Figure 5). The challenge in the *Reservoir* domain is to control the continuous flow of water in a series of interconnected reservoirs. The problem is difficult due to its stochastic transitions and high state and action dimensions (Gimelfarb, Taitler, and Sanner 2024). The challenge in the *PowerGen* domain is to control a power distribution network, consisting of different types of power generation units with different cost characteristic. The demand is coupled to a temperature variable. The challenge lies in the stochastic nature of the temperature variable and power units that are expensive to start and cheap to run. (Gimelfarb, Taitler, and Sanner 2024). The challenge in the *HVAC* domain is to control a heating system with continuous actions in a building with multiple interconnected rooms to maintain a specific temperature. Occupancy of the rooms is a Boolean stochastic variable which influences the heating costs. (Gimelfarb, Taitler, and Sanner 2024). While the focus of the navigation domains lies more on the combinatorial aspects of planning problem, the IPC domains focus more on finding the correct parameters to solve the planning problem.

Metrics As it is unrealistic that a learning algorithm which is trained on a scarce dataset $\mathcal{D}$ could match the typical planning evaluation metrics, like soundness, completeness, efficiency, and optimality[1], we turned to comparative metrics describing the performance of GCM-DQN and the baselines in comparison to a gold-standard algorithm which matches the traditional evaluation metrics under the cost of heavy computation.

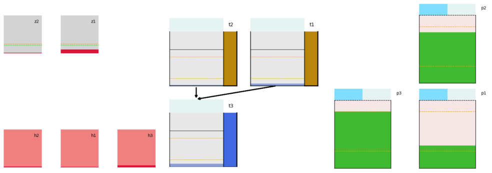

Figure 5: Evaluation domains from the **IPC Domains**. From left to right, HVAC domain, Reservoir domain, PowerGen domain. The images are taken from https://github.com/pyrddlgym-project.

As metrics we chose the *success rate* ρ and a *success-weighted distance metric* of a successful rollout and gold-standard trajectory. The planning success rate describes the number of successfully solved planning problems from a set of unseen test planning problems in the same planning domain. As the planning success rate alone does not take the optimality of the trajectories into account, we additionally considered the Dynamic Time Warping distance (DTW) between a trajectory and a gold-standard trajectory (Vintsyuk 1972). The DTW has the advantage that it can capture the similarity of two trajectories, even if they have different lengths. However, when considering multiple experimental trials over different seeds and domains, it gets confounded by the rate of successful trials. A policy that solved only one trial perfectly will achieve a better mean DTW than a policy that solved all planning problems with the cost of higher variance in some DTWs. We hence did not turn to pure DTW as distance metric but included a success-weighting, to rule out planners with low success rate, but potentially low DTWs, following Ilharco et al. (2019).

For each planning problem i of the test set, we compute

$$\mathrm{sDTW}_i = S_i \cdot \exp\Big[-\frac{\mathrm{DTW}_i}{\alpha\,|\tau_i^*|}\Big], \tag{19}$$

where α is a constant scale factor, $|\tau_i^*|$ is the length of the gold standard trajectory, and $S_i \in \{0, 1\}$ indicates whether the planner solved the planning problem. We caclulate the average over all sDTW_i in the test set

$$\mathrm{sDTW} = \frac{1}{N}\sum_{i=1}^{N} \mathrm{sDTW}_i \quad . \tag{20}$$

Thus, with the resulting sDTW metric, 0 corresponds to solving nothing, and 1 corresponds to solving everything with plans that are identical to the gold standard planner's solutions.

Baselines As baselines we used P-DQN (Xiong et al. 2018) and P-DDPG (Hausknecht and Stone 2016) from literature, as, to our knowledge, there are no offline Reinforcement Learning algorithms for solving planning problems

[1] As our algorithm is grounded in the Bellman Equation, its solutions will converge to optimal, sound, and complete results with an infinitely large dataset $\mathcal{D}$. However, this is not its operational scenario. We hence do not consider very large datasets for evaluation.

in PAMDPs. While P-DDPG is a policy based approach which is trained in an actor-critic setup (Hausknecht and Stone 2016), P-DQN is closer related to our approach using a DQN for evaluating different action-parameter tuples. However, instead of finding optimal parameter values via gradient-based search, it uses a Neural Network as heuristic for suggesting parameter values (Xiong et al. 2018). We transferred both baselines in an offline and model-based setting, using Conservative Q-learning, Hindsight Experience Replay, and potential-based shaping as for our algorithm. We gave each algorithm double the budget of steps needed to solve the planning problem, as the gold trajectories needed. We restricted the gradient-based optimization of $paramOpt$ to 100 updates.

B Full Results

In this section we report the full results, including the success-weighted DTW distance (sDTW) for our experiments on the navigation and IPC domains. The full results considering the performance in solving the planning problems are reported in Table 2. The results for succeedingly scarce training data are reported in Figures 6 for the navigation domains and 7 for the IPC domains.

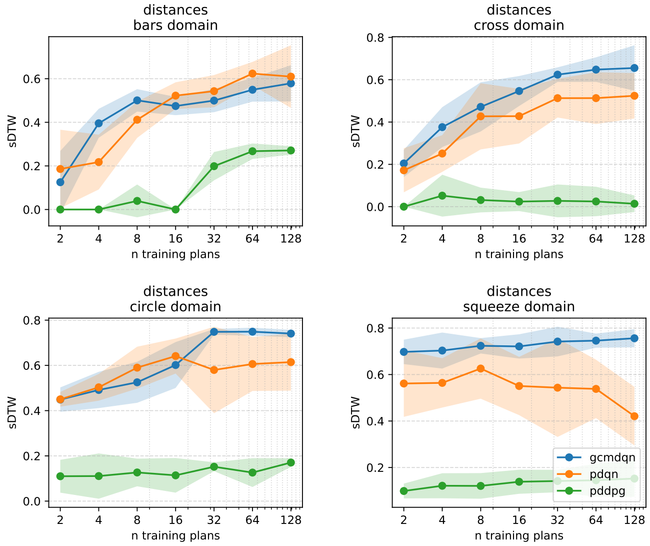

Figure 6: The figure shows the success weighted DTW distance of GCM-DQN and the P-DQN (Xiong et al. 2018) and P-DDPG (Hausknecht and Stone 2016) baselines on the navigation domains, under the constraint of succeedingly scarce training data.

C Ablation Studies

For each extension of our GCM-DQN algorithm, we performed an ablation study. We evaluate the ablation variant of our algorithm against the full version, using the same performance metrics as introduced in Section 5.

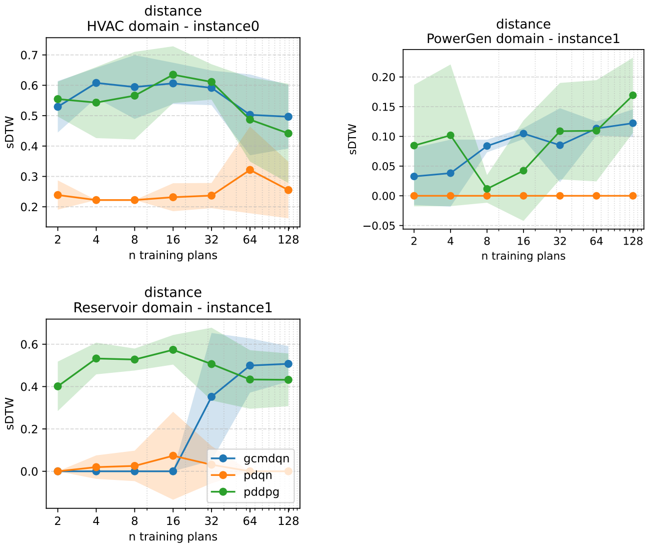

Figure 7: The figure shows the success weighted DTW distance of GCM-DQN and the P-DQN (Xiong et al. 2018) and P-DDPG (Hausknecht and Stone 2016) baselines on the IPC domains, under the constraint of succeedingly scarce training data.

C.1 Ablation Study Goal-Conditioning

For evaluating the impact of the goal conditioning in GCM-DQN, we replaced the conditioning during training and planning by a vector of zeros, carrying no information for reaching the actual goal state. We compared the DTW distance of the ablated version with the version from the paper, hypothesizing that the goal-conditioned GCM-DQN would show a lower DTW distance, meaning a better fit, than the ablated version. The results are shown in Figure 8. We could show that goal-conditioned version of GCM-DQN shows a higher DTW distance than the ablated version, with an exemption for the low sample regions of the bars, and circle domains.

C.2 Ablation Study $paramOpt$

For evaluating the impact of the goal $paramOpt$ algorithm during planning, we compared the performance of the full GCM-DQN algorithm against a variant, in which we replaced $paramOpt$ with a random draw of parameters from the observed parameter range in the training data. We compared the DTW distance of the ablated version with the paper version of GCM-DQN hypothesizing that the version which uses $paramOpt$ would show a lower DTW distance, meaning a better fit, than the ablated version. The results are shown in Figure 9. We could show that $paramOpt$ reduces the DTW distance in all cases, however, its impact being dependent on the planning domain.

C.3 Ablation Study Conservative Q-Learning

For evaluating the impact of Conservative Q-Leanring on our GCM-DQN algorithm, we compared the version from the paper against a variant in which we tuned down the α

Table 2: The table shows the mean success weighted DTW distance ± standard deviation across eight seeds for our GCM-DQN algorithm and the baselines over different navigation and IPC planning domains.

domain name	P-DQN (Xiong et al. 2018)	P-DDPG (Hausknecht and Stone 2016)	GCM-DQN (ours)
navigation – bars	**0.6096 ± 0.1437**	0.2709 ± 0.0049	0.5788 ± 0.0831
navigation – circle	0.6144 ± 0.1261	0.1705 ± 0.0003	**0.7407 ± 0.0163**
navigation – squeeze	0.4207 ± 0.1260	0.1524 ± 0.0053	**0.7561 ± 0.0390**
navigation – cross	0.5241 ± 0.1072	0.0138 ± 0.0013	**0.6553 ± 0.1073**
IPC – HVAC – instance0	0.2551 ± 0.0930	0.4415 ± 0.0233	**0.4965 ± 0.1045**
IPC – HVAC – instance1	0.5066 ± 0.0125	0.5432 ± 0.0289	**0.5249 ± 0.0194**
IPC – HVAC – instance2	0.4407 ± 0.0061	0.4475 ± 0.0166	**0.4520 ± 0.0190**
IPC – HVAC – instance3	0.0666 ± 0.0001	0.0949 ± 0.0302	**0.1168 ± 0.0361**
IPC – PowerGen – instance1	0.0000 ± 0.0000	**0.1691 ± 0.0632**	0.1222 ± 0.0233
IPC – PowerGen – instance2	0.0000 ± 0.0000	0.0000 ± 0.0000	**0.0085 ± 0.0170**
IPC – Reservoir – instance1	0.0000 ± 0.0000	0.4319 ± 0.1246	**0.5073 ± 0.0835**

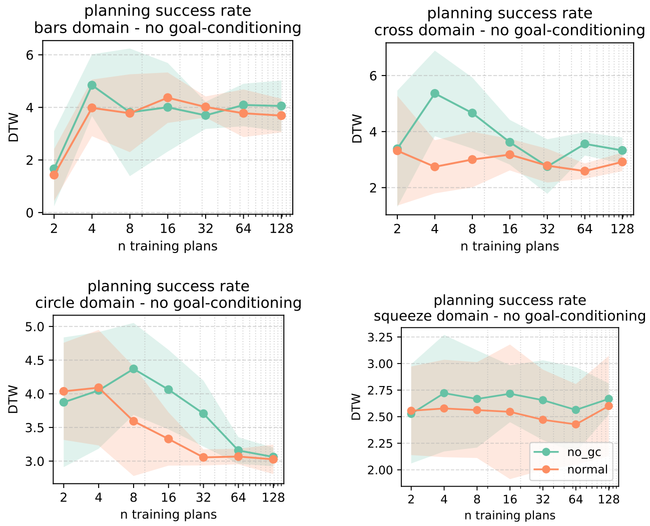

Figure 8: The figure shows the DTW distance of GCM-DQN with goal conditioning (normal) and without goal-conditioning (no gc) in the geometric and IPC domains.

parameter to 0, effectively ruling out the conservative term of the loss function in GCM-DQN. We used the DTW distance for evaluating the planning performance of the algorithms. The results are shown in Figure 10. We could show that the version using Conservative Q-Learning shows lower planning distances than the ablated version, with an exemption in the lower sample number region of the circle domain, and the middle sample number of the bars domain.

C.4 Ablation Study Hindsight Experience Replay and Conservative Action Sampling

To evaluate the impact of our data augmentation techniques, we evaluated GCM-DQN being trained with differently augmented datasets. These are the augmentations, we applied to $\mathcal{D}$:

- (no-her) this variant includes an augmentation only with Conservative Action Sampling (cf. Figure 11).
- (no-cas) this variant includes an augmentation only with Hindsight Experience Replay (cf. Figure 12).
- (no-cas-no-her) this variant includes no data augmentation (cf. Figure 13).

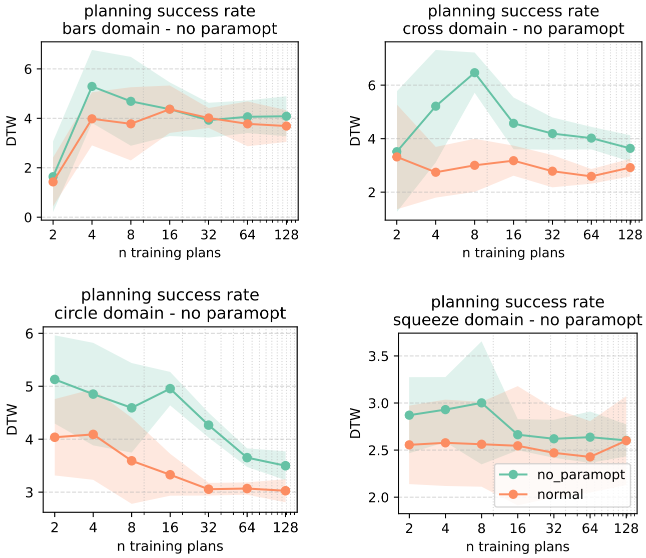

Figure 9: The figure shows the DTW distance of GCM-DQN (normal) against the performance of a GCM-DQN variant that does not use the *paramOpt* algorithm for finding optimal parameters during planning, but randomly draws from the range of observed parameters (no paramopt).

We evaluated the planning success rate of the different augmentation techniques, hypothesizing that augmenting the $\mathcal{D}$ will have a positive impact on the planning success rate. The results are shown in Figures 11, 12, and 13. We could show that for most of the tested domains an augmentation of the dataset has a positive effect on the planning success rate. While pure HER does not show a big effect, CAS tends to contribute more to a better planning success rate of GCM-

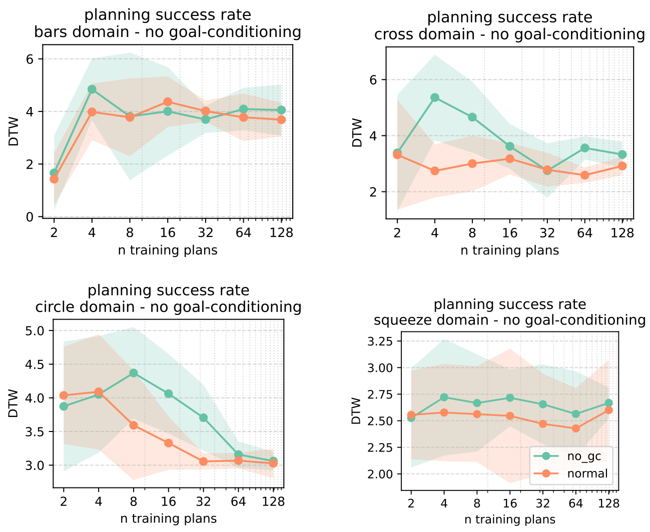

Figure 10: The figure shows the DTW distance of GCM-DQN (normal) and an ablated variant using no Conservative Q-Learning (no cql), as described in (Kumar et al. 2020).

DQN.

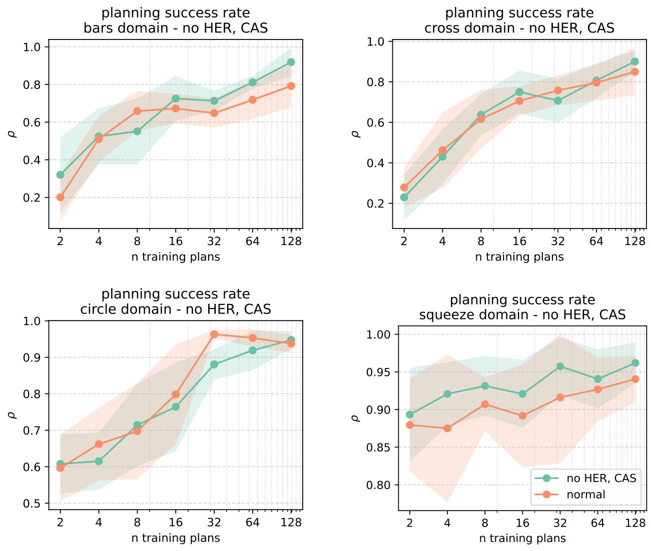

Figure 11: The figure shows the planning success rate of GCM-DQN when only augmenting the dataset $\mathcal{D}$ with Conservative Action Sampling (no HER, CAS), in comparison to the paper version (normal).

C.5 Ablation Study State Transition Models

In Equation 3 we introduced the possibility of modeling the transition dynamics in PAMDPs with an ensemble of individual transition models that approximate the effects of one individual action each. In this section, we compare the performance of such an ensemble-based approach (normal)

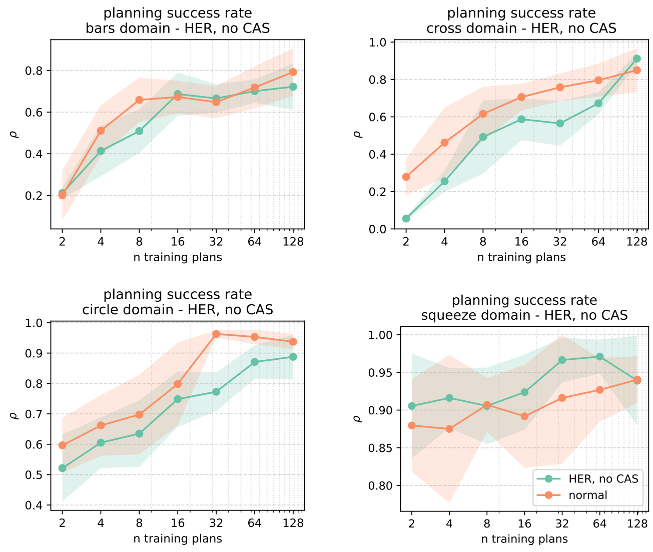

Figure 12: The figure shows the planning success rate of GCM-DQN when only augmenting the dataset $\mathcal{D}$ with Hindsight Experience Replay (HER, no CAS), in comparison to the version that we employed in the paper (normal).

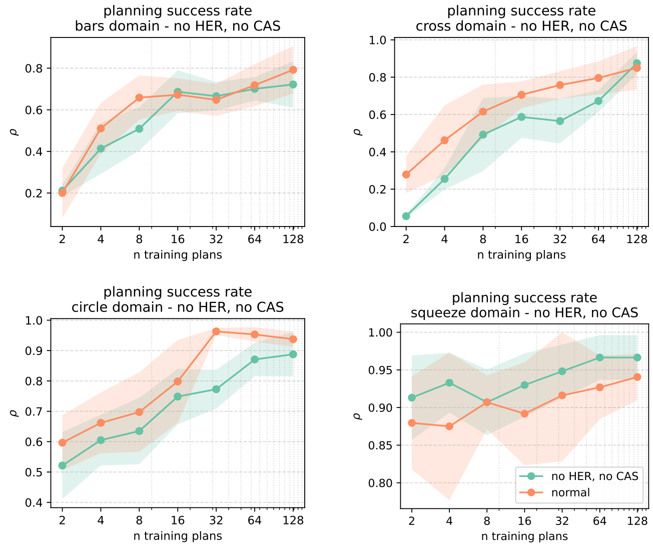

Figure 13: The figure shows the planning success rate of GCM-DQN when using no augmentation on the dataset $\mathcal{D}$, in comparison to the version that we employed in the paper (no HER, no CAS).

against using a single stm for approximating all action effects (single stm). Therefore, we ran an individual finetuning for each state transition model variant (cf. Appendix A) and compared the planning success rate and planning distance of the two variants. In the tested domains, we could not identify a large distance between the two types of state transition models (cf. Figure 14), with an exception in the squeeze domain, where the ensemble-based approach showed a higher planning success rate than a single transition model.

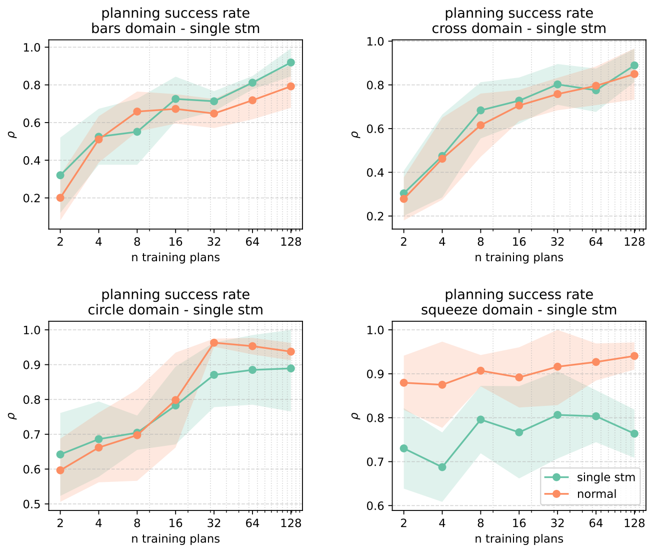

Figure 14: The figure compares the planning success rate of GCM-DQN using either a single state transition model for all actions (single stm) or an ensemble of state transition models (normal).

D Network Architectures

The following section shows the employed network architectures. The parametrizations can be found in the online repository https://anonymous.4open.science/r/gcmdqn-7CA6.

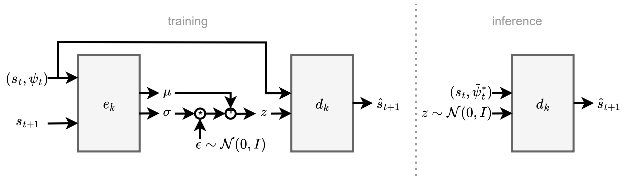

Figure 15: To capture the stochasticity in PAMDPs, we propose to use conditional stochastic state transition models. During training *(left)*, encoder e_ϕ and decoder d_ρ are trained to reconstruct s_{t+1} conditioned on s_t and ψ_t. During inference *(right)*, only the decoder d_ρ is used to generate s_{t+1} from the conditions s_t and ψ_t and a random sampled z from a normal distribution.

E Additional Discussion

Data Quantity and Diversity The quantity and diversity of the training data in the training dataset $\mathcal{D}$ had a significant impact on the performance of the tested algorithms. Our results support the intuition that more and diverse data improves the approximation of the true Q-function and true transition dynamics. The planning success rate of our GCM-DQN algorithm continuously improved as the number of plans in $\mathcal{D}$ increased. All methods struggled in scenarios where only few samples in the training dataset were available. Interestingly P-DQN had a slight advantage in domains with only 2 to 4 plans in $\mathcal{D}$, as soon as the number of plans in $\mathcal{D}$ goes beyond 16 samples, GCM-DQN surpassed it. We deliberately focused on scarce data scenarios in our evaluation, as they reflect the real-world application of planners, where collecting more data is expensive and an interaction with the environment is not possible. In this context, including Conservative Q-Learning an and Hindsight Experience Replay as mitigations for scarce data was important. Even though Hindsight Experience replay did not raise the mean planning success rates, it reduced the outcome variability and thus improved the reliability of GCM-DQN on small data. This implies that when working with scarce data and the performance is insufficient, adding additional data to $\mathcal{D}$ may be more effective than tweaking the algorithms in isolation.

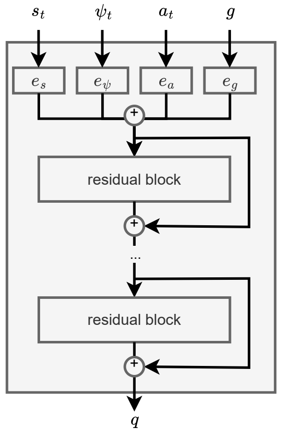

Figure 16: The figure shows the architecture that we used for the DQN in our GCM-DQN algorithm. We process each input to the network in a separate head and use residual blocks to smoothen the q surface. This allows our $paramOpt$ algorithm (cf. Algorithm 2) to converge to optimal parameters, easier.

Aleatoric Uncertainty from Latent Factors in the Planning Domain Real-world application scenarios for planners, e.g., industrial processes often show hidden factors and randomness that offline training cannot fully predict. I.e., in a manufacturing domain, tool wear out can alter a system's dynamics, introducing aleatoric uncertainty. Though our GCM-DQN approach attempts to accommodate stochasticity in its state transition models, systematic latent factor shifts over time will lead to mis-predictions of future transitions as the underlying transition dynamics changed. This limitation, however, is not unique to our approach but shared by all offline learning methods. Mitigating it could involve a periodic re-training with ”fresh” data or designing the model to model these factors explicitly or in latent variables.

Evaluation Fairness of Offline Baselines Finally, we discuss the evaluation fairness of the employed baselines. The employed baselines P-DDPG (Hausknecht and Stone 2016) and P-DQN (Xiong et al. 2018) were originally designed for online Reinforcement learning, where extensive interactions with the environment shapes the policy and DQNs. Conversely, we evaluated them in an offline, model-based setting. However, to ensure a fair evaluation with our GCM-DQN algorithm, we adapted both baselines to the offline setup, by incorporating the same techniques that we used in GCM-DQN to improve the training performance of the models. Namely, we used the same state transition models, Conservative Q-Learning, Hindsight Experience Replay, and Conservative Action sampling, creating a common and fair ground for evaluation.