Closing the Gap between TD Learning and Supervised Learning – A Generalisation Point of View

Raj Ghugare[1] **Matthieu Geist**[2] **Glen Berseth**[1*] **Benjamin Eysenbach**[3*]

[1]Mila, Université de Montréal [2]Google DeepMind [3]Princeton University

raj.ghugare@mila.quebec

Abstract

Some reinforcement learning (RL) algorithms have the capability of recombining together pieces of previously seen experience to solve a task never seen before during training. This oft-sought property is one of the few ways in which dynamic programming based RL algorithms are considered different from supervised learning (SL) based RL algorithms. Yet, recent RL methods based on off-the-shelf SL algorithms achieve excellent results without an explicit mechanism for stitching; it remains unclear whether those methods forgo this important stitching property. This paper studies this question in the setting of goal-reaching problems. We show that the desirable stitching property corresponds to a form of generalisation: after training on a distribution of (state, goal) pairs, one would like to evaluate on (state, goal) pairs not seen *together* in the training data. Our analysis shows that this sort of generalisation is different from *i.i.d.* generalisation. This connection between stitching and generalisation reveals why we should not expect existing RL methods based on SL to perform stitching, even in the limit of large datasets and models. We experimentally validate this result on carefully constructed datasets. This connection also suggests a simple remedy, the same remedy for improving generalisation in SL: data augmentation. We propose a naive *temporal* data augmentation approach and demonstrate that adding it to RL methods based on SL enables them to stitch together experience so that they succeed in navigating between states and goals unseen together during training.

1 Introduction

Recent methods that view RL as a purely SL problem of mapping input states and desired goals, to optimal actions [1, 2] have gained a lot of attention due to their simplicity, scalability [3]. These methods sample a goal g from the dataset, which was previously encountered after taking an action a from a state s, and then imitate a by treating it as an optimal label for reaching g from s. This simple recipe achieves excellent results on common benchmarks [4]. However, at a deeper and fundamental level, there are some important differences between RL and SL. This paper studies one of those differences: the capability of some RL algorithms to stitch together pieces of experience to solve a task never seen during training. This stitching property [5] is common among RL algorithms that perform dynamic programming (e.g., DQN [6], DDPG [7], TD3 [8], IQL [9]). While some papers have claimed that some SL approaches already have this stitching property [2], both our theoretical and empirical analyses suggest some important limitations of these prior claims.

Our primary contribution is to formally relate this stitching property to a form of generalisation. Viewing RL as a SL problem, the task of the agent is to generate optimal outputs (actions) given certain inputs (state-goal pairs). To test the stitching property, we will evaluate the performance on (start, goal) pairs that do not appear together (but do appear separately) during training. Our analysis proves that this problem occurs due to a difference in training and testing distributions and that this difference does not go away by increasing the amount of data. Rather, when data are collected from a mixture of policies, there can be (state, goal) pairs that are never visited in the same trajectory, despite being frequented in separate trajectories. This way of data collection reflects real world offline datasets, which are collected via different sources. Our empirical results support the theory: we demonstrate that prior RL methods based on SL (DT [2] and RvS [4]) fail to perform stitching, even

[*] Equal advising.

when trained on abundant quantities of data. This result does not contradict experiments from prior work, but underscores the importance of carefully constructing datasets when testing for stitching. We have open sourced our benchmark, data, and code for testing the stitching capabilities of RL algorithms.[1]

Taking a cue from SL, if generalisation is the problem, then data augmentation is likely an effective approach [10]. The second contribution of this paper is a way of applying data augmentation to these same SL methods such that they acquire this stitching property and succeed in navigating between unseen (start, goal) pairs. This form of data augmentation involves *time*, rather than random cropping or shifting colors, by augmenting the goals sampled during training with new goals from other trajectories. Our data augmentation scheme does require an explicit notion of distance between states to detect overlapping trajectories. We demonstrate that this data augmentation endows the prior RL methods based on SL with the *stitching generalisation* property. In our experiments, we use the standard L2 distance to solve tasks with state dimensions upto $\mathbb{R}^{29}$.

Overall, our work hints that SL approaches to RL may not obviate the algorithm designer – even when trained on vast quantities of data, these approaches do not perform stitching. We provide a framework for studying stitching as a form of generalisation and devise a simple data-augmentation strategy which improves the generalisation capabilities of SL approaches by up to Todo:XX. For the RL community, we hope that these results shift the narrative around generalisation and data augmentation – these are not just computer vision techniques tacked on top of RL methods, but can actually get to the core differences between supervised learning and reinforcement learning. We hope that our work shifts the narrative around generalisation and data augmentation in RL. In RL, we can ask questions about generalisation that do not seem to make sense in other areas of machine learning; for example, how can we use data from one task to solve an unseen task? And, in RL, data augmentation is not just a tool for learning perceptual invariances, but also can enable generalisation to structurally different tasks.

2 Related Work

Generalisation in RL is generally associated with making correct predictions for unseen but similar states and actions [11–13], ignoring irrelevant details [14–16], or robustness towards changes in the reward or transition dynamics [17–19]. A main thrust of RL research over the last few years seems to be to look at how large models [20, 21] can provide generalisation [22–24] with impressive results. We cast stitching as a distinct form of generalisation and show that such large models, trained only with simple SL objectives, cannot perform stitching [9]. We acknowledge that using a learned model of the world might facilitate stitching [25–27] because the model can be used to generate trajectories that navigate between state-goal pairs unseen together in the training data.

Data augmentation in RL has been proposed as a remedy to improve generalisation in RL [28–34], akin to SL [35]. Our proposed data augmentation requires augmenting the true goal, such that the optimal action remains the same even for the augmented goals. Perhaps the most similar prior work are the ones which use dynamic programming to augment existing trajectories to improve the performance properties of SL algorithms [36–38]. However, because these methods still require dynamic programming, they don't have the same simplicity that make SL algorithms appealing in the first place. Our method is conceptually similar, but draws an important connection between generalisation and stitching, and our empirical results show that good performance can be achieved without any dynamic programming.

Prior methods that do some form of explicit stitching. Previous work on stitching abilities of SL algorithms have conflicting claims. The DT paper [2] shows experiments where their method performs stitching, suggesting that transformer-based SL algorithms generalise on out-of-distribution goals. On the contrary, [39] provide a simple example where SL algorithms do not perform stitching, regardless of whether a transformer is used. RvS [4] shows that a simple SL algorithm with a fully connected architecture can match the performance of TD algorithms, especially on D4RL's antmaze environment, which supposedly requires stitching to achieve high performance. We provide a formal definition of stitching, and show (both empirically and theoretically) that SL-based RL algorithms do not exhibit such generalisation.

[1] See supplementary material for code and datasets.

3 Preliminaries

Controlled Markov processes. We will study the problem of goal-conditioned RL in a controlled Markov process with states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$. The dynamics are $p(s' \mid s, a)$, the initial state distribution is $p_0(s_0)$, the discount factor is γ. The policy $\pi(a, \mid s, g)$ is conditioned on a pair of state and goal $s, g \in \mathcal{S}$. For a policy π, define $p_t^\pi(s_t \mid s_0)$ as the distribution over states visited after exactly t steps. We can then define the discounted state occupancy distribution and its conditional counterpart as

$$p_+^\pi(s_{t+} = g) \triangleq \mathbb{E}_{s \sim p_0(s_0)}\left[p_+^\pi(s_{t+} = g \mid s_0 = s)\right], \tag{1}$$

$$p_+^\pi(s_{t+} = g \mid s_0 = s) \triangleq (1 - \gamma)\sum_{t=0}^{\infty} \gamma^t p_t^\pi(s_t = g \mid s_0 = s), \tag{2}$$

where s_{t+} is the variable that specifies a future state corresponding to the discounted state occupancy distribution. Given a state-goal pair $s, g \sim p_{\text{test}}(s, g)$ at test time, the task of the policy is to maximise the probability of reaching the goal g in the future

$$\max_\pi J(\pi; s, g), \quad \text{where } \; J(\pi; s, g) = \mathbb{E}_{s,g \sim p_{\text{test}}(s,g)}\left[p_+^\pi(s_{t+} = g \mid s_0 = s)\right]. \tag{3}$$

Data collection. Our work focuses on the offline RL setting where the agent has access to a fixed dataset of N trajectories $D = (\{s_0^i, a_0^i, ..\})_{i=1}^N$. Our theoretical analysis will assume that the dataset is collected by a set of policies $\{\beta(a \mid s, h)\}$, where h specifies some context. For example, h could reflect different goals, different language instructions, different users or even different start state distributions. Precisely, we assume that the data was collected by first sampling a context from a distribution $p(h)$, and then sampling a trajectory from the corresponding policy $\beta(a \mid s, h)$. We will use the shorthand notation $\beta_h(\cdot \mid \cdot) = \beta(\cdot \mid \cdot, h)$ to denote the data collecting policy conditioned on context h. Trajectories are assumed to be stored without h, hence the context denotes all hidden information that the true data collection policies used to collect the data. This construction covers real-world cases where data is collected from various sources depending on various hidden factors.

This setup of collecting data corresponds to a mixture of Markovian policies [2]. There is a classic result saying that, for every such *mixture* of Markovian policies, there exists a Markovian policy that has the same discounted state occupancy measure.

Lemma 3.1 (Rephrased from Theorem 2.8 of [40], Theorem 6.1 of [41])**.** *Let a set of context-conditioned policies $\{\beta_h(a \mid s)\}$ and distribution over contexts $p(h)$ be given. There exists a Markovian policy $\beta(a \mid s)$ such that it has the same discounted state occupancy measure as the mixture of policies:*

$$p_+^\beta(s_{t+}) = \mathbb{E}_{p(h)}\left[p_+^{\beta_h}(s_{t+})\right]. \tag{4}$$

The policy $\beta(a \mid s)$ is simple to construct mathematically as follows. For data collected from the mixture of context conditioned policies, let $p^\beta(h \mid s)$ be the distribution over the context given that the policy arrived in state s.

$$\beta(a \mid s) \triangleq \sum_h \beta_h(a \mid s) p^\beta(h \mid s). \tag{5}$$

Theorem 6.1 [41] proves the correctness of this construction. The policy $\beta(a \mid s)$ is also easy to construct empirically – simply perform behavioral cloning (BC) on data aggregated from the set of policies. We will hence call this policy the BC policy.

Outcome Conditional behavioral cloning (OCBC). While our theoretical analysis will consider generalisation abstracted away from any particular RL algorithm, we will present empirical results using a simple and popular class of goal-conditioned RL methods: Outcome conditional behavioral cloning [42]. These methods go by a number of names, including Decision Transformer (DT) [2], Upside down RL [1], RL via Supervised Learning (RvS) [4], Goal Conditioned Supervised Learning (GCSL) [43] and many others [44, 45]. These methods take as input a dataset of trajectories

[2] Note that the mixture is at the level of trajectories, not at the level of individual actions.

$\mathcal{D} = \{(s_0, a_0, s_1, a_1)\}$ and learn a goal-conditioned policy $\pi(a \mid s, g)$ using a maximum likelihood objective:

$$\max_{\pi(\cdot|\cdot,\cdot)} \mathbb{E}_{(s,a,g)\sim\mathcal{D}} \left[\log \pi(a \mid s, g)\right]. \tag{6}$$

The sampling above can be done by first sampling a trajectory from the dataset (uniformly at random), then sampling a (state, action) pair from that trajectory, and setting the goal to be a random state that occurred later in that same trajectory. If we incorporate our data collecting assumptions, then this sampling can be written as

$$\max_{\pi(\cdot|\cdot,\cdot)} \mathbb{E}_{h\sim p(h)} \Big[\mathbb{E}_{\substack{s,a\sim\beta_h(a|s),\, p_+^{\beta_h}(s) \\ s_{t+}\sim p_+^{\beta_h}(s_{t+}|s,a)}} \left[\log \pi(a \mid s, s_{t+})\right]\Big]. \tag{7}$$

4 "Stitching" as a form of generalisation

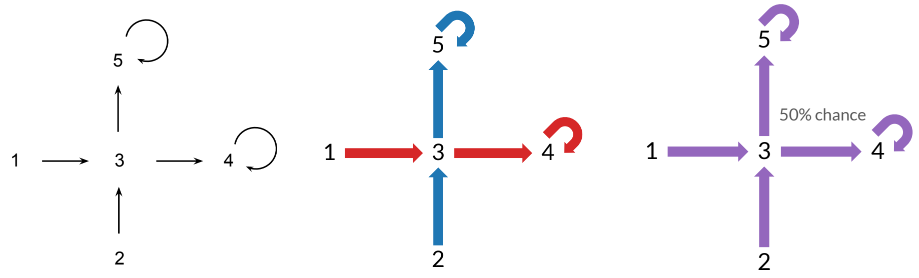
Before concretely defining stitching generalisation, we will describe three desirable properties that are colloquially associated with "stitching" and the learning dynamics of TD methods. *(P.1)* The ability to select infrequently seen paths that are more optimal than frequent ones. While a shorter trajectory between the state and the goal may occur infrequently in the dataset, TD methods can find more examples of this trajectory by recombining pieces of different trajectories, thanks to dynamic programming. This property is enjoyed by both SARSA (expectation over actions) and Q-learning (max over actions) methods, and is primarily associated with the sample efficiency of learning. *(P.2)* The ability to evaluate policies different than those which collected the data, and perform multiple steps of policy improvement. This property is unique to Q-learning. *(P.3)* Temporal difference methods (both Q-learning and SARSA) can also recombine trajectories to find paths between states never seen together during training. This property is different from the first property in that it is not a matter of data efficiency – temporal difference methods can find paths that will never be sampled from the data collecting policies, even if given infinite samples. All three of these properties are colloquially referred to as "stitching" in the literature. While these properties are not entirely orthogonal, they are distinct: certain algorithms may have just some of these properties. Obtaining all these properties in a simpler (than TD) framework is difficult, and it remains unclear whether OCBC methods possess any of them. Our work takes one step in this direction, by focusing on the third property. We formalize this property as a form of generalisation which we will refer to as stitching generalisation.

Defining stitching generalisation will allow us to analyze when and whether OCBC methods perform stitching, both theoretically (this section) and experimentally (Section 6). Intuitively, *stitching generalisation* looks at navigating between states and goals which are never seen together in the same trajectory, but can be navigated using the information present in different trajectories. It therefore tests a form of "stitching" [46, 47], akin to "compositional generalisation" [48–50]. To define this generalisation, we will specify a training distribution and testing distribution. The training distribution corresponds to sampling a context $h \sim p(h)$ and then sampling an (s, g) pair from the corresponding policy β_h. This is exactly how OCBC methods are trained in practice (Section 3). The testing distribution corresponds to sampling an (s, g) pair from the BC policy $\beta(a \mid s)$ defined in Equation (5). For each distribution, we will measure the performance $f^{\pi(\cdot|\cdot,g)}(s, g)$ of goal-conditioned policy $\pi(a \mid s, g)$.

Definition 1 (Stitching generalisation)**.** *Let a set of context-conditioned policies $\{\beta_h(a \mid s)\}$ be given, along with a prior over contexts $p(h)$. Let $\beta(a \mid s)$ be the policy constructed via Eq. (5). Let $\pi(a \mid s, g)$ be a policy for evaluation. The stitching generalisation of a policy $\pi(a \mid s, g)$ measures the differences in goal-reaching performance for goals sampled $g \sim p_+^{\beta}(s_{t+} \mid s)$ versus goals sampled from $g \sim \mathbb{E}_{p(h)}[p_+^{\beta_h}(s_{t+} \mid s)]$:*

$$\underbrace{\mathbb{E}_{\substack{s\sim p_+^{\beta}(s) \\ g\sim p_+^{\beta}(s_{t+}|s)}} \left[f^{\pi(\cdot|s,g)}(s,g)\right]}_{\textit{test performance}} - \underbrace{\mathbb{E}_{\substack{h\sim p(h),\, s\sim p_+^{\beta_h}(s) \\ g\sim p_+^{\beta_h}(s_{t+}|s)}} \left[f^{\pi(\cdot|s,g)}(s,g)\right]}_{\textit{train performance}}.$$

The precise way performance f is measured is not important for our analysis: "generalisation" simply means that the performance under one distribution is similar to the performance under another. In our



(a) MDP. (b) $\beta_{h=1}$, $\beta_{h=2}$. (c) β.

Figure 1: *(a)* The MDP has 5 states and two actions (up and right). Prior work [39, 51] also have similar counterexamples, though they have not related this to generalisation to the best of our knowledge.*(b)* Data is collected using two contexts conditioned policies shown in blue and red. Both policies collect an equal amount of data $p(h=1)=p(h=2)=0.5$. *(c)* The behavior cloned policy (equation 5) is shown in purple to indicate that it is obtained from combining data from both blue and red policies. The conditional occupancy distribution of the purple policy is used to test stitching. This distribution is different from the training distribution. During training *(b)*, state 2 and goal 4 will never be sampled together, but during testing *(c)*, state 2 and goal 4 have a non-zero probability of being sampled.

experiments, we will look at performance measured by the success rate at reaching the commanded goal.

On the surface, it may seem like stitching generalisation is the same as the standard i.i.d. generalisation studied in SL. It may seem that both the test and train distributions are the same. Lemma 3.1 about reducing mixtures of policies to a single Markovian policy seems to hint that this might be true. *However, stitching generalisation is not the same as i.i.d. generalisation, and is more akin to combinatorial generalisation* [50]. Indeed, this distinction has not been made before while analysing OCBC methods [39, 42]. This misconception is demonstrated by the following lemma:

Lemma 4.1. *There exist a collection of policies* $\{\beta_h\}$ *and context distribution* $p(h)$ *such that,* conditioned on a state*, the distribution of states and goals for the data collecting policies (training) is different from the distribution of states and goals (testing) for BC policy* β*.*

$$\mathbb{E}_{p(h)}\left[p_+^{\beta_h}(s_{t+}\mid s)p_+^{\beta_h}(s)\right] \neq p_+^{\beta}(s_{t+}\mid s)p_+^{\beta}(s) \quad \textit{for some states } s, s_{t+}. \tag{8}$$

Proof. We prove this Lemma by providing a simple counterexample. Consider the deterministic MDP shown in Figure 1 which has five states $[1,5]$ and two actions $\{\text{right}, \text{up}\}$. The states four and five are absorbing states; once the agent enters one of these states, it will stay there for eternity. There are two data collecting agents β_h with contexts $h=1$ and $h=2$, which navigate upward from state two to state five, and rightward from state one to state four respectively. Both policies collect equal amount of data $p(h=1)=p(h=2)=0.5$. We will prove that the training and testing distribution (LHS and RHS of Eq. (8)) for the state-goal pair $\{s=2, s_{t+}=4\}$ are not equal.

$$\begin{aligned}
&\mathbb{E}_{p(h)}\left[p_+^{\beta_h}(s_{t+}=4\mid s=2)p_+^{\beta_h}(s=2)\right]\\
&=\frac{p_+^{\beta_{h=1}}(4\mid 2)p_+^{\beta_{h=1}}(2)}{2}+\frac{p_+^{\beta_{h=2}}(4\mid 2)p_+^{\beta_{h=2}}(2)}{2}\\
&=\frac{p_+^{\beta_{h=1}}(4\mid 2)(1-\gamma)}{2}+\frac{0\times(1-\gamma)}{2}\\
&=\frac{(1-\gamma)^2}{2}\sum_{t=0}^{\infty}\gamma^t p_t^{\beta_{h=1}}(4\mid 2)\\
&=\frac{(1-\gamma)^2}{2}\times 0=0
\end{aligned}$$

$$\begin{aligned}
&p_+^{\beta}(s_{t+}=4\mid s=2)p_+^{\beta}(s=2)\\
&=\frac{p_+^{\beta}(s_{t+}=4\mid s=2)(1-\gamma)}{2}\\
&=\frac{(1-\gamma)^2}{2}\sum_{t=0}^{\infty}\gamma^t p_t^{\beta}(4\mid 2)\\
&=\frac{(1-\gamma)^2}{2}\left(\frac{\gamma^2}{2}+\frac{\gamma^3}{2}+\dots\right)\\
&=\frac{(1-\gamma)\gamma^2}{4}
\end{aligned}$$

The LHS and RHS are unequal for all values of $\gamma\in(0,1)$. □

Algorithm 1 OCBC + Temporal Data Augmentation

```
Input: Dataset : $D = (\{s_0, a_0, \dots\})$.
Initialize OCBC policy $\pi_\theta(a|s,g)$ with parameters $\theta$.
Set $\epsilon$ = augmentation probability, $m$ = mini-batch size.
$(\{d_0, d_1, \dots\})$ = CLUSTER($\{s_0, s_1, \dots\}$).            ▷ Group all states in the dataset.
while not converged do
    for $t = 1, \cdots, m$ do
        Sample $(s_t, a_t, g_{t+}) \sim D$.                           ▷ Equation (7)
        if $u \sim unif[0,1] \leq \epsilon$ then
            Get the group of the goal: $k = d_{t+}$.
            Sample waypoint states from the same group: $w \sim \{s_i; \forall i$ such that $d_i = k\}$ .
            Sample augmented goal $\tilde{g}$ from the future of $w$, from the same trajectory as $w$.
            Augment the goal $g_{t+} = \tilde{g}$.
        Collect the loss $\mathcal{L}_t(\theta) = -\log \pi_\theta(a_t \mid s_t, g_{t+})$.
    Update $\theta$ using gradient descent on the mini-batch loss $\frac{1}{m}\sum_{t=1}^{m} \mathcal{L}_t(\theta)$
    Return : $\pi_\theta(a|s,g)$
```

In summary, while the BC policy $\beta(a \mid s)$ will visit the same states as the mixture of data collecting policies *on average*, conditioned on some state, the BC policy $\beta(a \mid s)$ may visit a different distribution of future states than the mixture of policies. Even if infinite data is collected from the data collecting policies, there can be pairs of states that will never be visited by any one data collecting policy in a single trajectory. The important implication of this negative result is that stitching requires the OCBC algorithm to recover a distribution over state-goal pairs (β) which is different from the one it is trained on ($\beta_{h=1}, \beta_{h=2}$). In theory, the training distribution has enough information to recover the state-goal distribution of the BC policy and it is upto the algorithm to extract this. For example, TD-learning algorithms can recover this implicitly, without the knowledge of the contexts of the data collecting policies.

Many RL methods sidestep this negative result by doing dynamic programming (i.e., temporal difference learning). One way of viewing dynamic programming is that it considers all possible ways of stitching together trajectories, and selects the best among these stitched trajectories. But for OCBC algorithms based on SL (e.g., DT [2], RvS [4], GCSL [43], PCHID [44], URL [1]), it is not clear apriori why these methods should have the stitching generalisation property, leading to the following hypothesis:

Hypothesis 1. *Conditional imitation learning methods do not have the stitching generalisation property.*

We will test this hypothesis empirically in our experiments. In Appendix A, we discuss connections between stitching generalisation and spurious correlations.

5 Temporal Augmentation Facilitates generalisation

The previous section introduced forms of generalisation tailored to the RL setting. These definitions allow us to rethink the oft-sought "stitching" property as a form of generalisation, and measure that generalisation in the same way we measure generalisation in SL: by measuring a difference in performance under two objectives.

Casting stitching as a form of generalisation allows us to employ a standard tool from SL: data augmentation. When the computer vision expert wants a model that can generalize to random crops, they train their model on randomly-cropped images. Indeed, prior work has applied data augmentation to RL to achieve *various* notions of generalisation [28, 50]. However, we use a different type of data augmentation to facilitate stitching. In this section, we describe a data augmentation approach that allows OCBC methods to improve their stitching capabilities.

Recall that OCBC policies are trained on (s, a, g) triplets. To perform data augmentation, we will replace g with a different goal $\tilde{g}$. To sample these new goals $\tilde{g}$, we first take the original goal g and identify states from the offline dataset which are nearby to this goal (Section 5). Let w denote one of these nearby "waypoint" states. Looking at the trajectory that contains w, the new goal $\tilde{g}$ is a random state that occurs after w in this trajectory. We visualize this data augmentation in Fig. 2.

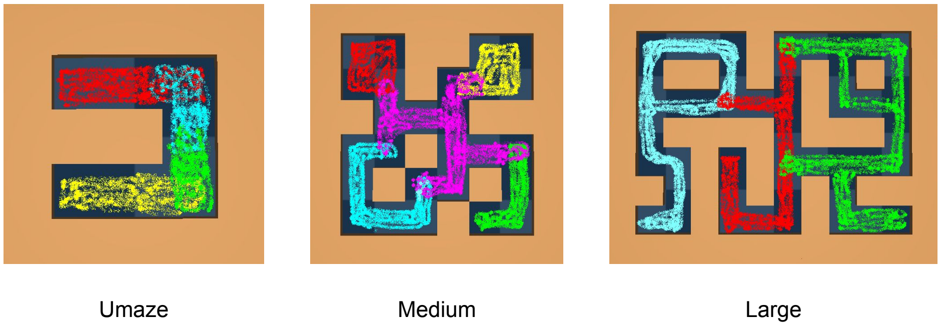


Figure 3: **Offline datasets that do require stitching.** Different colors represent the navigation regions of different data collecting policies. These visualisations are for the "point" mazes. The "ant" maze datasets are similar to these ones. Appendix Fig. 11 shows the "ant" maze datasets.

Identifying nearby states. The problem of sampling nearby states can be solved by clustering all the states from the offline dataset before training. This assumes a distance metric in the state space. Using this distance metric, every state can be assigned a discrete label from k different categories using a clustering algorithm [52, 53]. Finding a good distance metric is difficult, especially in high-dimensional settings [54].

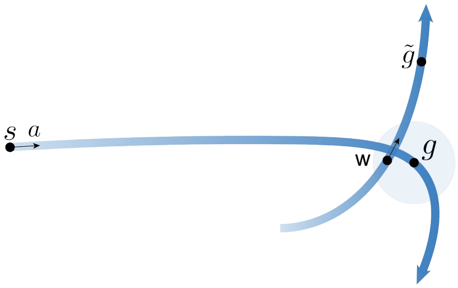


Figure 2: DATA AUGMENTATION FOR STITCHING: After sampling an initial training example (s, a, g) (Eq. (7)), we look for a *waypoint* state w in the light blue region around the original goal g, and then sample a new *augmented* goal $\tilde{g}$ from later in that trajectory. This is a simple approach to sample cross trajectory goals $\tilde{g}$ such that the action a is still an optimal action at state g.

Theoretical intuition on temporal data augmentation. While exact theoretical guarantees for temporal data augmentation will depend on how good the distance metric is, we can think of the data augmentation as trying to approximately sample goals from $p_+^\beta(s_{t+} = g \mid s)$. To understand this, we can dry run temporal data augmentation on the MDP in Fig. 1 and see how it enables the agent to traverse from state 2 to state 4. Suppose that the OCBC algorithm normally samples an $(s, a, g) = (2, \text{up}, 3)$ triplet from the replay buffer Eq. (6). The data augmentation will sample a waypoint state from the same group as g. Since we are in a tabular MDP, the waypoint would be $s = 3$ itself, but from other trajectories. Suppose the waypoint $s = 3$ is sampled from the red trajectory. Then the augmented goal $\tilde{g}$ will be a state in the red trajectory, randomly sampled from the future of the waypoint. Since, there is only one state in the future, the augmented goal will be $\tilde{g} = 4$, and the augmented training sample will be $(s, a, g) = (2, \text{up}, 4)$. This training sample is optimal for reaching state 4 from state 2. But without temporal augmentation, OCBC methods will never see this sample.

Method summary. Algorithm 1 summarizes how our data augmentation can be applied on top of existing OCBC algorithms. Given a method to group all the states in the offline dataset, we can add our data-augmentation to existing OCBC algorithms by adding about 5 lines of code (marked in blue). In our experiments, we use the k-means algorithm [52] to group states together. We use two types of inputs to the k-means to group states together (1) Entire state vector, and (2) Only the goal coordinates of the state vector. In all our experiments, goals are specified by the desired x, y coordinates. The first approach does not assume separate access to the goal coordinates and is therefore generally applicable. In our experiments, all OCBC policies do have separate access to the goal information as they are only conditioned on desired x, y coordinates. For generality, we include results corresponding to both types of data-augmentations.

Popular offline datasets do not evaluate stitching. While the maze datasets from D4RL [46] were originally motivated to test the stitching capabilities of RL algorithms, we find that most test state-goal pairs are in-training distribution, i.e, they are visited by a single data collecting policy. Thus, a good success rate on these datasets does not necessarily mean that a method performs stitching. This may explain why OCBC methods have achieved excellent results on the original maze tasks [4], despite the fact that our theory suggests that these methods do not perform stitching. In our experiments, we collect new offline datasets that explicitly test for stitching. To evaluate stitching generalisation, we need to test the policy on out of distribution state-goal pairs – states which occur in the test distribution, but not in the train distribution 1. These state-goal pairs satisfy two conditions: 1) No individual data collection policy (β_h) should navigate between the state-goal pairs; 2) The BC policy (β) should have some chance of navigating between the state-goal pairs. Formally this means that we should test using (s, g) pairs such that

$$p_+^{\beta}(s_{t+} = g \mid s) > 0 \quad \text{and} \quad p_+^{\beta_h}(s_{t+} = g \mid s) = 0 \; \forall \; h. \tag{9}$$

Figure 3 visualises the datasets that we collect for our experiments. Different data collecting policies (shown in different colors) have different regions of navigation. During data collection, these policies navigate between random state-goal pairs chosen from their region of navigation. In Appendix C.3, we contrast our datasets with the original D4RL datasets (see Fig. 12 for a similar visualisation of the original D4RL datasets) and discuss the differences that are necessary to test for stitching generalisation.

6 Experiments

The experiments aim *(1)* to verify our theoretical claim that OCBC methods do not always exhibit stitching generalisation; and *(2)* to evaluate how adding temporal augmentation to OCBC methods improves stitching.

OCBC methods. **RvS** [4] is an OCBC algorithm that uses a fully connected neural network policy and often achieves results comparable to TD-learning algorithms on various offline RL benchmarks [4]. **DT** [2] treats RL as a sequential SL problem and uses the transformer architecture as a policy. DT outputs an action, conditioning not only on the current state, but a history of states, actions and goals. We use all the original hyperparameters of the baselines. See Appendix C.2 for implementation details.

Tasks. We use the "point" and "ant" mazes (umaze, medium and large) from D4RL [46]. As discussed in Section 5, we carefully collect our new offline datasets to test for stitching (see Fig. 3 and Fig. 11 for visualisation).

Differences between original D4RL and our datasets. We made two main changes in the way our datasets (Fig. 3, Fig. 11) were collected compared to the original D4RL datasets (Fig. 12). *First*, we ensure that different data collecting policies have distinct navigation regions, with only a small overlapping region. This change helps to clearly distinguish between algorithms that can and cannot perform stitching generalisation. *Second*, the agent in the original D4RL datasets often moves in a direction that is largely dependent on its current location in the maze. For example in the topmost row of the umaze, the D4RL policy always moves towards the right. To reduce such spurious relations, we randomize the start-state and goal sampling, for the data collecting policies. That is, in the topmost row of our umaze datasets, the data collecting policy moves both towards the right or left, depending on its start-state and goal. Details about how such spurious relations can hamper stitching generalisation are discussed in Appendix A.

Testing for stitching. We command the agent to navigate between (state, goal) pairs previously unseen *together*, and measure the success rate. Each task chooses states and goals randomly from 2-6 different regions in the maze. In Fig. 4, we can see that both DT and RvS struggle to solve unseen tasks at test-time. However, applying temporal data-augmentation to RvS improves the goal-reaching success rate on $^5/_6$ tasks, likely because the augmentation results in sampling (state, goal) pairs otherwise *unseen together*. While temporal augmentation boosts the success rates, there remains room for other scalable methods to achieve even better performance.

Does more data remove the need for augmentation? Although our theory (Lemma 4.1) suggests that generalisation is required because of a change in distribution and is not a problem due to limited

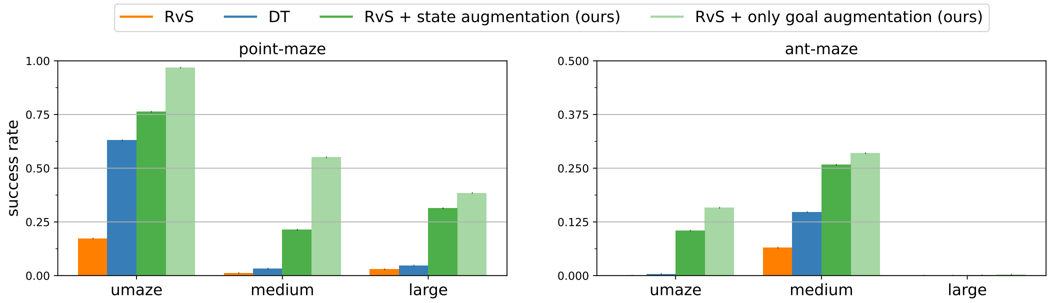
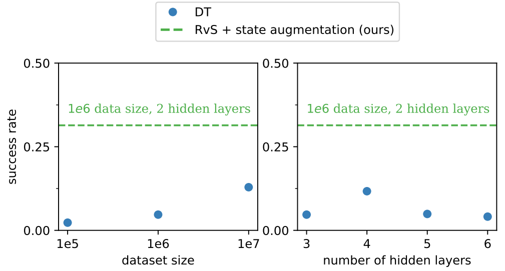


Figure 4: Adding data augmentation outperforms the OCBC baselines on most tasks. “Only goal augmentation” refers to an oracle version of our augmentation that uses priviledged information (x, y coordinates) when performing augmentation. The reason we do not add data augmentation with DT is that because DT [2] takes in a context of previous states and actions, the data augmentation method should cluster a history of states and actions, which was difficult to do computationally.



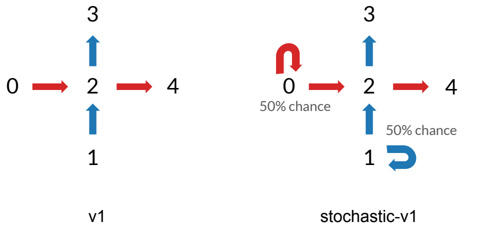


Figure 5: Comparison of DT trained on different sizes of offline dataset (left) and using a different number of hidden layers (right). We also add RvS + data augmentation on the complete state. Even with larger datasets or model sizes, the generalisation of DT is worse than RvS + data augmentation.

Figure 6: MDP with 5 states and 2 actions (up and right). All episodes end after taking two actions. Data is collected using two policies (red and blue). The only difference between v1 and v1-stochastic, is the data collecting policies are stochastic at states 0 and 1. The stitching task is to navigate from states 1 to 4.

data, conventional wisdom says that larger datasets generally result in better generalisation. To empirically test whether this is the case, we train DT on 10 million transitions (10 times more than Fig. 4) on the point-maze large task. In Fig. 5 (left), we see that even with more data, the stitching generalisation of DT does not improve much. Lastly, scaling the size of transformer models [55] is known to perform better in many SL problems. To understand whether this can have an effect on stitching capabilities, we increased the number of layers in the original DT model. In Fig. 5 (right), we can see that increasing the number of layers does not have an effect on DT’s stitching capabilities.

7 Discussion

In this work, we shined a light over an area that the community has been investigating recently, *can SL-based approaches perform stitching*. We find that theoretically and empirically, OCBC methods can not perform the type of stitching that temporal difference methods are known to have. We found that this limitation is tied to a type of generalisation we call stitching generalisation. We propose a type of temporal data augmentation to precisely perform the desired type of stitching and help bridge the gap between OCBC and temporal difference algorithms. Using this data augmentation, we are able to increase the stitching abilities of OCBC algorithms by a factor of 2× in “point" maze and 2.5× in “ant" maze.

Limitations and future work. While we gave an intuitive argument for why augmentation facilitates stitching, there is more work to be done to provide a formal proof that this augmentation always results in stitching. Our proposed augmentation also assumes an access to a good distance metric in the state space. Lifting these assumptions and developing scalable OCBC algorithms that generalise is a promising direction for future work.

References

[1] Juergen Schmidhuber. Reinforcement learning upside down: Don't predict rewards – just map them to actions, 2020.

[2] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[3] Kuang-Huei Lee, Ofir Nachum, Mengjiao Yang, Lisa Lee, Daniel Freeman, Winnie Xu, Sergio Guadarrama, Ian Fischer, Eric Jang, Henryk Michalewski, and Igor Mordatch. Multi-game decision transformers, 2022.

[4] Scott Emmons, Benjamin Eysenbach, Ilya Kostrikov, and Sergey Levine. Rvs: What is essential for offline rl via supervised learning? *arXiv preprint arXiv:2112.10751*, 2021.

[5] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[7] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[8] 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.

[9] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *arXiv preprint arXiv:2110.06169*, 2021.

[10] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning, 2017.

[11] Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning, 2018.

[12] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning, 2019.

[13] Kenny Young, Aditya Ramesh, Louis Kirsch, and Jürgen Schmidhuber. The benefits of model-based generalization in reinforcement learning, 2023.

[14] Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction, 2021.

[15] Homanga Bharadhwaj, Mohammad Babaeizadeh, Dumitru Erhan, and Sergey Levine. Information prioritization through empowerment in visual model-based rl, 2022.

[16] Maximilian Igl, Kamil Ciosek, Yingzhen Li, Sebastian Tschiatschek, Cheng Zhang, Sam Devlin, and Katja Hofmann. Generalization in reinforcement learning with selective noise injection and information bottleneck, 2019.

[17] Jun Morimoto and Kenji Doya. Robust reinforcement learning. In T. Leen, T. Dietterich, and V. Tresp, editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000.

[18] Chen Tessler, Yonathan Efroni, and Shie Mannor. Action robust reinforcement learning and applications in continuous control, 2019.

[19] Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. Robust predictable control, 2021.

[20] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

[21] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[22] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. Rt-1: Robotics transformer for real-world control at scale, 2023.

[23] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances, 2022.

[24] Homanga Bharadhwaj, Abhinav Gupta, Shubham Tulsiani, and Vikash Kumar. Zero-shot robot manipulation from passive human videos, 2023.

[25] Charles A. Hepburn and Giovanni Montana. Model-based trajectory stitching for improved offline reinforcement learning, 2022.

[26] E. Barnard. Temporal-difference methods and markov models. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(2):357–365, 1993.

[27] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels, 2019.

[28] Austin Stone, Oscar Ramirez, Kurt Konolige, and Rico Jonschkowski. The distracting control suite – a challenging benchmark for reinforcement learning from pixels, 2021.

[29] Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale, 2021.

[30] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning, 2020.

[31] Nicklas Hansen and Xiaolong Wang. Generalization in reinforcement learning by soft data augmentation, 2021.

[32] Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels, 2021.

[33] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning, 2021.

[34] Chaochao Lu, Biwei Huang, Ke Wang, José Miguel Hernández-Lobato, Kun Zhang, and Bernhard Schölkopf. Sample-efficient reinforcement learning via counterfactual-based data augmentation, 2020.

[35] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6:1–48, 2019.

[36] Taku Yamagata, Ahmed Khalil, and Raul Santos-Rodriguez. Q-learning decision transformer: Leveraging dynamic programming for conditional sequence modelling in offline RL, 2023.

[37] Keiran Paster, Silviu Pitis, Sheila A. McIlraith, and Jimmy Ba. Return augmentation gives supervised RL temporal compositionality, 2023.

[38] Ian Char, Viraj Mehta, Adam Villaflor, John M. Dolan, and Jeff Schneider. Bats: Best action trajectory stitching, 2022.

[39] David Brandfonbrener, Alberto Bietti, Jacob Buckman, Romain Laroche, and Joan Bruna. When does return-conditioned supervised learning work for offline reinforcement learning? In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[40] Brian D Ziebart. *Modeling purposeful adaptive behavior with the principle of maximum causal entropy*. Carnegie Mellon University, 2010.

[41] Eugene A Feinberg and Adam Shwartz. *Handbook of Markov decision processes: methods and applications*, volume 40. Springer Science & Business Media, 2012.

[42] Benjamin Eysenbach, Soumith Udatha, Sergey Levine, and Ruslan Salakhutdinov. Imitating past successes can be very suboptimal. *arXiv preprint arXiv:2206.03378*, 2022.

[43] Dibya Ghosh, Abhishek Gupta, Ashwin Reddy, Justin Fu, Coline Devin, Benjamin Eysenbach, and Sergey Levine. Learning to reach goals via iterated supervised learning. *arXiv preprint arXiv:1912.06088*, 2019.

[44] Hao Sun, Zhizhong Li, Xiaotong Liu, Bolei Zhou, and Dahua Lin. Policy continuation with hindsight inverse dynamics. *Advances in Neural Information Processing Systems*, 32, 2019.

[45] Aviral Kumar, Xue Bin Peng, and Sergey Levine. Reward-conditioned policies, 2019.

[46] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2021.

[47] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34:1273–1286, 2021.

[48] Ivan Vankov and Jeffrey Bowers. Training neural networks to encode symbols enables combinatorial generalization, 2019.

[49] Philippe Hansen-Estruch, Amy Zhang, Ashvin Nair, Patrick Yin, and Sergey Levine. Bisimulation makes analogies in goal-conditioned reinforcement learning, 2022.

[50] Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of zero-shot generalisation in deep reinforcement learning. *Journal of Artificial Intelligence Research*, 76:201–264, jan 2023.

[51] Keiran Paster, Sheila A. McIlraith, and Jimmy Ba. Planning from pixels using inverse dynamics models. In *International Conference on Learning Representations*, 2021.

[52] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[53] Alexander A. Sherstov and Peter Stone. Function approximation via tile coding: Automating parameter choice. In Jean-Daniel Zucker and Lorenza Saitta, editors, *Abstraction, Reformulation and Approximation*, pages 194–205, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[54] Charu Aggarwal, Alexander Hinneburg, and Daniel Keim. On the surprising behavior of distance metric in high-dimensional space. *First publ. in: Database theory, ICDT 200, 8th International Conference, London, UK, January 4 - 6, 2001 / Jan Van den Bussche ... (eds.). Berlin: Springer, 2001, pp. 420-434 (=Lecture notes in computer science ; 1973)*, 02 2002.

[55] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[56] Ishita Dasgupta, Erin Grant, and Thomas L. Griffiths. Distinguishing rule- and exemplar-based generalization in learning systems, 2022.

[57] Shiori Sagawa, Pang Wei Koh, Tatsunori B Hashimoto, and Percy Liang. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. *arXiv preprint arXiv:1911.08731*, 2019.

[58] Mark Towers, Jordan K. Terry, Ariel Kwiatkowski, John U. Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Andrew Tan Jin Shen, and Omar G. Younis. Gymnasium, March 2023.

[59] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

A Relationship with Spurious Correlations.

Handling stitching is somewhat akin to handling spurious correlations studied in the SL community. In the RL setting, we want to navigate from A to B given a dataset that contains some trajectories with A and some with B but none with both A and B. This is somewhat analogous to common settings in object detection in computer vision, where the object background is highly indicative of the object class. For example, the common waterbirds dataset [57] aims to classify images of birds into two classes, "land birds" and "water birds," but the image backgrounds are correlated with the class: water birds are usually depicted on top of a background with water. For evaluation, the classifier is shown an image of a "water bird" on top of a land background (and vice versa). Similar to the RL setting, SL evaluation is done using pairs of inputs that are rarely seen together during training. However, whereas the SL setting aims to learn a classifier that ignores certain aspects of the input, the RL setting is different because the aim is to learn a policy that can reason about both inputs.

There is another connection between the RL setting and spurious correlations, a connection that makes the RL setting look the opposite of the SL setting. For some goal-conditioned RL datasets, the current state is sufficient for predicting which action leads to the goal – the policy does not need to look at the goal. In other datasets, the goal is sufficient for predicting the correct action. However, for navigating between pairs of states unseen together in the dataset, a policy must look at both the state and the goal inputs.

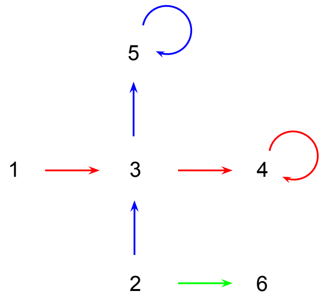


Figure 7: Spurious Correlations: To understand how spurious correlations are related to stitching, let's look at this simple determinisitc MDP in which three data collecting policies (red, blue and green) collect the offline dataset. Note that an OCBC algorithm which ignores the state, can also achieve a zero training loss Eq. (6) on this offline dataset. Whenever 4 is the desired goal in the dataset, action right is always optimal irrespective of the current state. Any SL algorithm that learns a minimal decision rule will in fact learn to ignore the state to reduce the training loss to zero in this case [56]. But during test time, starting at state 2 and conditioned on goal 4 such an SL algorithm will ignore the current state and move towards right which is clearly suboptimal.

B Additional Experiments.

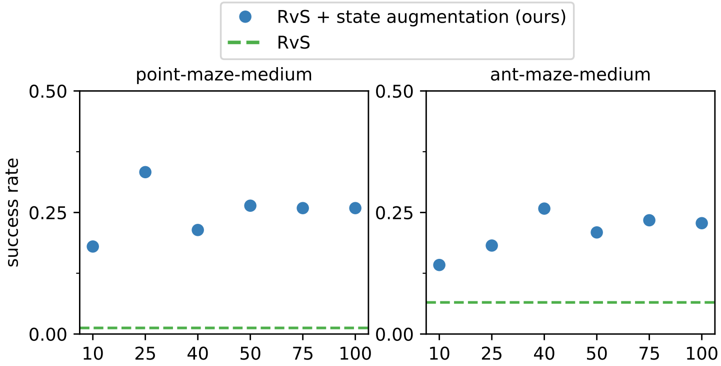


Figure 8: Comparison of our data augmentation trained with different number of centroids in the K-means algorithm on pointmaze-medium and antmaze-medium. All values outperform RvS across both tasks.

B.1 Ablating the number of centroids for K-means.

In Fig. 8, we ablate the choice of the number of centroids used in K-means on two environments – pointmaze-medium and antmaze-medium. All choices of centroids significantly outperform the RvS method on both tasks.

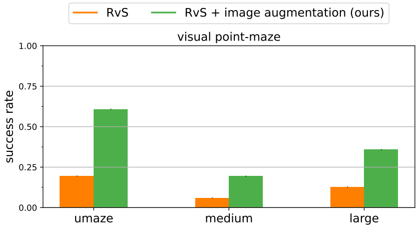


Figure 9: Comparison RvS + data augmentation with RvS on 3 image based pointmaze tasks. Applying K-means on high dimensional images enables generalisation, even when using a naive L2 distance. Note that our method makes no extra assumption and uses the same information as the baseline.

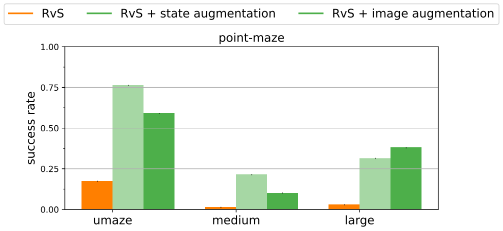


Figure 10: To estimate the difference in performance caused by clustering in high dimensions, we perform an experiment on the state based pointmaze task. We handicap the data augmentation method, by applying the clustering on images instead of the low level states, while keeping all other things the same. The clusters obtained using images do increase generalisation, even outperforming state based augmentation on the largest maze.

B.2 Image based experiments.

As mentioned in Section 5, it can be difficult to provide a good distance metric, especially high for dimensional problems. We test this empirically using two image based experiments across three pointmaze tasks.

Fig. 9. : We compare RvS and data augmentation on 3 image based pointmaze tasks (umaze, medium and large). The datasets are collected similar to our main experiments Fig. 1. The only difference is that instead of the x, y coordinates, the observations consist of a top down image of the maze (64×64). In Fig. 9, we can see that even a naive L2 distance based K-means achieves much better stitching generalisation than RvS.

Fig. 10. : To remove other factors like the difficulty of learning a policy from images, we repeat the same state-based experiment as Fig. 1, on pointmaze tasks. The only difference is that, we include a version of data augmentation that clusters states by applying K means on the corresponding images, instead of states. This experiment handicaps the data augmentation, while the policy learns from low level states. In Fig. 10, we can see that the image based clusters perform much better than RvS on all three tasks, even outperforming the state based clusters on the largest maze.

C Experimental Details

C.1 Environments

We use the "point" and "ant" mazes (umaze, medium and large) from D4RL [46]. As discussed in Section 5, we carefully collect our new offline datasets to test for stitching generalisation (see Fig. 3 for visualisation). In the "point" maze, the task is to navigate a ball with 2 degrees of freedom that is force-actuated in the cartesian directions x and y. In the "ant" maze task, the agent is a 3-d ant from Farama Foundation [58]. To collect data for the "point" maze, we use a PID controller. To collect data for the "ant" maze, we use the same pre-trained policy from D4RL [46]. In Fig. 11, we provide a visualisation of the offline dataset in all "ant" mazes.

C.2 Implementation Details

In this section we provide all the implementation details as well as hyper-parameters used for all the algorithms in our experiments – DT, RvS, and RvS + temporal data augmentation.

DT. We used the exact same hyper-parameters that the original DT paper [2] used for their mujoco experiments. The original DT paper [2] conditioned the transformer on future returns rather than future goals. For our experiments, we switch this conditioning to goals instead. At every time-step the transformer takes in as input the previous action, current state, and desired goal. The desired goal

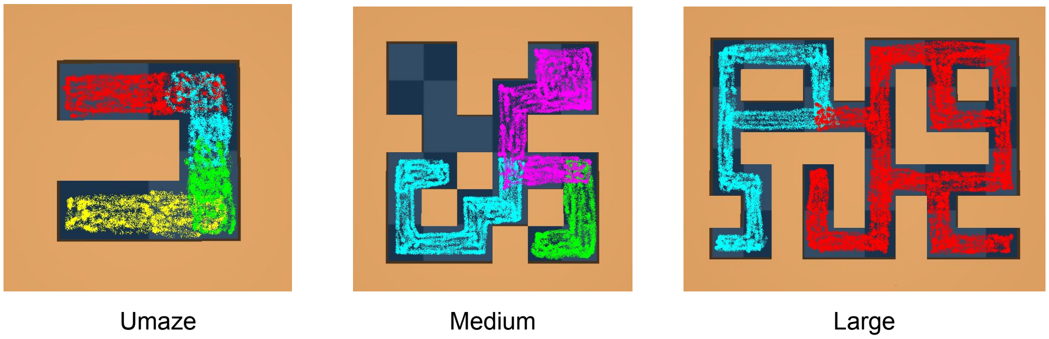


Figure 11: Offline datasets that we collect for the “ant" mazes. Different colors represent the navigation regions of different data collecting policies. See Fig. 3 for a similar visualisation of the “point" maze datasets.

remains constant throughout the episode, but is still fed to the transformer at every timestep. All hyperparameters used for DT are mentioned in Table 1.

Table 1: Hyperparameters for DT.

hyperparameter	value
training steps	1×10^5
batch size	256
context len	20
optimizer	AdamW
learning rate	1×10^{-3}
warmup steps	5000
weight decay	1×10^{-4}
dropout	0.1
hidden layers (self attention layers)	3
embedding dimension	128
number of attention heads	1

RvS. RvS is an OCBC algorithm which uses a fully connected neural network policy. We use the hyperparameters as prescribed by the original paper [4]. All hyperparameters used for RvS are mentioned in Table 2.

Table 2: Hyperparameters for RvS.

hyperparameter	value
training steps	1×10^6
batch size	256
optimizer	Adam
learning rate	1×10^{-3}
hidden layers	2
hidden layer dimension	1024

RvS + temporal data augmentation. As mentioned in Algorithm 1, given a method to cluster states together, it only requires 5 lines of code to add the temporal data augmentation on top of an OCBC method. We use the k-means algorithm from scikit-learn [59] with the default parameters to group states together. Adding data augmentation on top of RvS introduces 2 extra hyperparameters, which we mention in Table 3. We *do not* tune both of these hyperparameters in our paper. Nevertheless, we do ablate the choice of K in k-means.

Table 3: Hyperparameters for temporal data augmentation.

hyperparameter	value
K (number of clusters for k-means) :	
umaze	20
medium	40
large	80
ϵ (probability of augmenting a goal)	0.5

C.3 Differences between the original D4RL and our datasets.

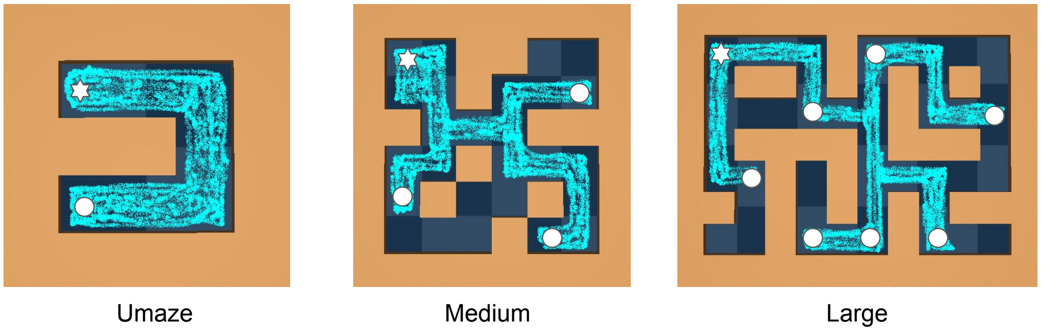


Figure 12: Similar to Fig. 3 and Fig. 11, we create a visualisation of the original d4rl dataset. This is not an exact visualisation of the actual trajectories that are present inside those datasets, but a visualisation of the data collecting policies that those datasets used [46]. During data collection, the policy starts from one starting region, which is marked by a white star. The policy navigates to a goal selected from one of the goal regions, which are marked by white circles. During test time, start-states and goals are selected from similar starting and goal regions, making it difficult to evaluate the stitching generalisation of offline RL algorithms.