Chain of Thought Imitation with Procedure Cloning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Imitation learning aims to extract high-performance policies from logged demonstrations of expert behavior. It is common to frame imitation learning as a supervised learning problem in which one fits a function approximator to the input-output mapping exhibited by the logged demonstrations (input *observations* to output *actions*). While the framing of imitation learning as a supervised input-output learning problem allows for applicability in a wide variety of settings, it is also an overly simplistic view of the problem in situations where the expert demonstrations provide much richer insight into expert behavior. For example, applications such as path navigation, robot manipulation, and strategy games acquire expert demonstrations via planning, search, or some other multi-step algorithm, revealing not just the output action to be imitated but also the procedure for how to determine this action. While these intermediate computations may use tools not available to the agent during inference (e.g., environment simulators), they are nevertheless informative as a way to explain an expert's mapping of state to actions. To properly leverage expert procedure information without relying on the privileged tools the expert may have used to perform the procedure, we propose *procedure cloning*, which applies supervised sequence prediction to imitate the series of expert computations. This way, procedure cloning learns not only *what* to do (i.e., the output action), but *how* and *why* to do it (i.e., the procedure). Through empirical analysis on navigation, simulated robotic manipulation, and game-playing environments, we show that imitating the intermediate computations of an expert's behavior enables procedure cloning to learn policies exhibiting significant generalization to unseen environment configurations, including those configurations for which running the expert's procedure directly is infeasible.

1 Introduction

The idea of learning by imitation in autonomous agents closely resembles how humans (especially children) learn in real life — by watching and mimicking how someone else performs a certain task [1]. While humans are able to generalize exceedingly well from a small number of demonstrations, today's imitation-learned autonomous agents often struggle in situations that only slightly differ from the demonstrations, e.g., opening a door in a different shape or color [2]. One explanation for such a difference is that while humans imitate, we *understand* the task at a high level as opposed to only remembering a mapping from images to actions [3], and indeed, generalization failures can also occur in humans when there is a lack of understanding of the underlying reasons for the behavior, such as solving a complex math problem. As a result, students are told to "learn principles, not formulas" and "understand, do not memorize" to encourage better generalization — to be able to solve problems that are similar but different from the ones taught in lectures [4, 5, 6]. Just as students are taught the step-by-step derivations of a math problem during a lecture, is it possible to have an equivalent "chain of thought" supervision for training autonomous agents?

While solving complex math derivations may not be the predominant application of today's imitation-learned autonomous agents, the tasks they are commonly applied to — e.g., path navigation, robot

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

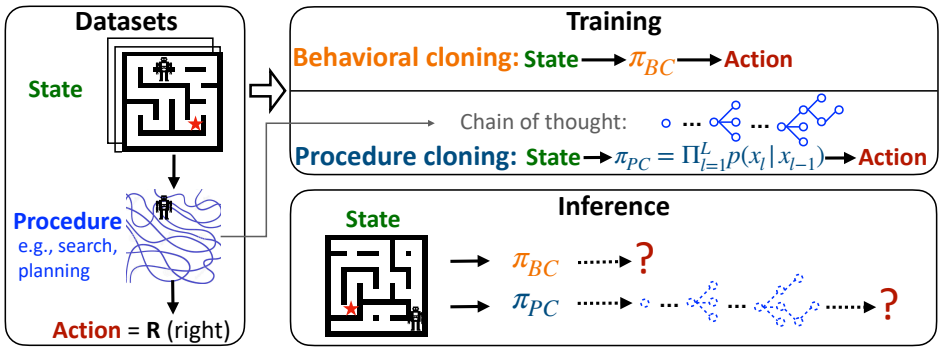

Figure 1: Visualization of the dataset collection, training, and inference of BC and PC on a maze navigation task. During dataset collection, the expert uses a search procedure to determine the optimal action to generate a path to the goal location (red star). During training, BC discards these intermediate search outputs and learns to map states to actions directly. In contrast, PC learns the complete sequence of intermediate computations (i.e., branches and backtracks) associated with the search procedure. During inference, PC generates a sequence of intermediate search outcomes emulating the search procedure on a new test map before outputting the final action.

manipulation, and strategy games — often *do* employ chain-of-thought reasoning procedures in the form of planning, search, or other multi-step algorithms when collecting expert data [7, 8, 9]. Since such multi-step algorithms are application-specific and therefore lack a common representation which can be systematically characterized, it is typical for imitation learning to assume access to only logged demonstrations of state-action pairs, leaving out the much richer insight into expert behavior provided by the algorithm's intermediate computations. In addition, the planning or search procedures used by the expert may rely on tools not available to the agent during inference (e.g., environment simulators), so how these intermediate computations should be used to facilitate imitation learning is not clear.

In this work, we formulate *chain of thought imitation* as an extension of traditional imitation learning. Different from the traditional imitation learning setup where an agent only has access to expert state-action pairs as demonstrations, chain of thought imitation also has access to the intermediate computations that generated the expert state-action pairs in the training data (Figure 1). We then propose *procedure cloning* (PC), an alternative to behavioral cloning (BC) [10], which applies supervised sequence prediction to imitate the complete series of of expert computations before outputting an expert action (Figure 1). Procedure cloning learns a policy by maximizing the likelihood of the joint distribution of procedure observations and expert actions, which can be modeled autoregressively using a transformer-like architecture [11]. During inference, a procedure cloned policy autoregressively generates procedure observations from a given input state, mimicking the computations of a search or planning algorithm before outputting the final action, thus avoiding any reliance on privileged tools or information used by the expert's procedure. From a modeling perspective, procedure cloning can employ a more expressive model (e.g., transformer) trained on more data (i.e., procedure observations), which leads to better generalization according to the new "scaling law" of large language models [12]. Intuitively, procedure cloning learns not only what to do (i.e., the output action), but how and why to do it (i.e., the procedure), which further resembles how humans learn complex tasks.

We demonstrate how to conduct procedure cloning and leverage the generalization ability of procedure cloned agents on a variety of path navigation, robotic manipulation, and strategy game tasks. For example, in path navigation an expert trajectory may be determined by using a BFS search algorithm on an annotated map (e.g., x, y coordinates of obstacles), which are expensive to obtain [13]. We show that a procedure cloning agent can successfully learn to imitate BFS on a previously unseen test map without requiring additional annotations, achieving 100% test accuracy while a BC agent completely fails to navigate (0% accuracy) when the maze layout changes. Similarly, in an image-based robotic manipulation task, we observe that BC quickly overfits to the set of training images, whereas procedure cloning learns to predict the intermediate computation outcomes of a scripting policy, thus generalizing much better and achieving a success metric of 83.9% compared to 78.2% from the previous state-of-the-art [14]. Finally, in strategy games expert trajectories are collected by running MCTS [15], which requires access to a simulator and can be extremely slow [16]. We show that procedure cloning can effectively learn from the path traversed by MCTS collected in

a deterministic environment to then successfully generalize in a zero-shot manner to stochastic environments and more difficult game settings, where running MCTS, even if given access to a simulator, performs poorly.

2 Related Work

Generalization in sequential decision making. Learning decision making policies with good generalization properties has a long-standing history in bandit [17, 18, 19], imitation learning [20], and reinforcement learning (RL) [21, 22, 23, 24] settings in the form of regret or Probably Approximately Correct (PAC) bound analysis. These studies are concerned with generalization in a continual learning setup without explicit separation of training and testing stages. We are more interested in an agent's ability to generalize to a separate "test" environment whose configuration is different from the training environment, which is often what happens when an agent is deployed to the real world after being trained in a simulator. With high-capacity neural network parametrized policies being the norm in training image-based agents, the risk of overfitting to the training environment is nontrivial [25, 26]. Various works have injected stochasticity into the training process including using stochastic policies [27], random starts [28, 29], sticky actions [30], and frame skipping [31] as a means to prevent overfitting to the specific environment dynamics experienced by the agent during training. A variety of image-based regularization techniques have also been applied to deep RL including dropout and l-2 regularization [32], data augmentation and batch normalization [33], domain randomization [34], and network randomization [35]. Instead of improving generalization through regularization or data augmentation like in existing work, we instead ask the question of whether learning the sequence of computations as opposed to only the final expert action during training can help an agent generalize better.

Access to additional task information Many previous works in imitation learning and RL have observed that access to additional task information can help an agent learn better policies. For instance, [36] relies on access to the simulator for collecting more expert trajectories to reduce the quadratic error in task horizon to linear. [37] assumes that expert demonstrations contain both high-level and low-level trajectories, and learns a hierarchical policy explicitly. While our procedure observations can appear to be high-level labels or sub-goals similar to [37, 38], we do not make assumptions about the structure of procedure observations, which can simply be scalar variable values computed during program execution. There is also a large body of literature that assumes access to a suboptimal offline dataset in addition to the expert demonstration data collected from the same environment, and conduct representation learning [39, 40, 41, 42, 43], hierarchical skill extraction [44, 45, 46], or dynamics model learning [47, 48, 49] on the suboptimal offline data followed by imitation learning from an expert. These works commonly assume good coverage in the offline data, which requires large amounts of state-action pairs being collected from running additional policies in the environment. We instead propose observing additional information from only running the expert policy, which requires many fewer state-action pairs being collected for training. Another line of existing work in the direction of multi-task learning suggests that learning an auxiliary task enhances imitation learning or RL [50, 51, 2, 52, 2, 53, 54]. For instance, [55] showed that predicting internal features of the emulator (e.g., enemy on the screen) is beneficial, [56] found predicting scene depth helps navigation, and [57] found predicting explanations helps relational tasks with causal structure. Chain of thought imitation differs from multi-task learning in that procedures directly influence the action outcomes through computations, in which case learning the procedure directly (as opposed to treating it as an auxiliary objective similar to existing work) is more beneficial. The graphical model in Figure 2 visualizes this distinction.

Chain of thought sequence modeling. The idea of decomposing multi-step problems into intermediate steps (the so-called chain of thought [58]) and learning the intermediate steps using a sequence model has been applied to *domain specific* problems such as program induction [59], learning to solve math problems [60], learning to execute [61], learning to reason [62, 63, 64, 65, 66, 67], and language model prompting [58]. The chain of thought imitation learning problem we formulate is *domain agnostic* and applicable to many sequential decision making task traditionally solved by imitation learning in a Markovian setting such as robot locomotion, navigation, manipulation, and strategy games. Unlike language-based tasks, problems in decision making have only recently started being explored by language models [68, 69, 70, 71, 72, 73], as the Markovian nature of these problems brings the value of sequence modeling into question. Our work contributes to bridging the gap between learning memoryless policies in Markovian environments and the intuition that large sequence models should help in reasoning-based decision making.

3 Preliminaries

In this section, we define relevant notations and introduce the problem of imitation learning. We also review behavioral cloning (BC), which our work aims to improve.

MDP notations. Consider a Markov decision process (MDP) [74] $\mathcal{M} := \langle S, A, \mathcal{R}, \mathcal{T}, \mu, \gamma \rangle$, consisting of a state space S, an action space A, a reward function $\mathcal{R} : S \times A \to \mathbb{R}$, a transition function $\mathcal{T} : S \times A \to \Delta(S)$[1], an initial state distribution $\mu \in \Delta(S)$, and a discount factor $\gamma \in [0, 1)$. A policy $\pi : S \to \Delta(A)$ interacts with the environment starting at an initial state $s_0 \sim \mu$. At each timestep $t \geq 0$, an action $a_t \sim \pi(s_t)$ is sampled and applied to the environment, and the environment transitions into the next state $s_{t+1} \sim \mathcal{T}(s_t, a_t)$ while producing a scalar reward $\mathcal{R}(s_t, a_t)$. The state visitation distribution $d^\pi(s)$ induced by a policy π is defined as $d^\pi(s) := (1-\gamma) \sum_{t=0}^{\infty} \gamma^t \cdot \Pr[s_t = s | \pi, \mathcal{M}]$. We use $(s, a) \sim d^\pi$ to denote $s \sim d^\pi, a \sim \pi(s)$.

Imitation learning. In the standard imitation learning setting, $\mathcal{R}$, $\mathcal{T}$, and μ are unknown to the agent. Learning solely takes place on a fixed dataset of expert state-action pairs $\mathcal{D}_* = \{(s^{(i)}, a^{(i)})\}_{i=1}^n$ collected by an expert policy π_* by sampling $s^{(i)} \sim d^{\pi_*}$ and $a^{(i)} \sim \pi_*(s^{(i)})$. The goal of imitation learning is to train a policy $\pi : S \to \Delta(A)$ on $\mathcal{D}_*$, such that π closely approximates π_* according to some metric such as the Kullback–Leibler (KL) divergence of their state visitation distributions $D_{\mathrm{KL}}(d^{\pi_*}(s) \| d^\pi(s))$. Regardless of the choice of this metric, it is computed over states including those outside of the training data $\mathcal{D}_*$, and requires an agent to generalize over unseen states to achieve good test performance.

Behavioral cloning (BC). The nature of learning from an expert dataset $\mathcal{D}_*$ leads to the common framing of imitation learning as a supervised problem of learning state to action mappings. Under this framing, behavioral cloning (BC) [10] proposes to solve the imitation learning problem by learning π that minimizes

$$J_{\mathrm{BC}}(\pi) := \mathbb{E}_{(s,a) \sim (d^{\pi_*}, \pi_*)}[-\log \pi(a|s)] \tag{1}$$

using samples from the training data $(s, a) \sim \mathcal{D}_*$. Equation 1 can also be viewed as the cross entropy loss for multi-class classification of state to action mappings. Agents trained with this vanilla BC objective (i.e., direct max-likelihood on the training dataset $\mathcal{D}_*$) can quickly fail to generalize to unseen states when the expert data size is small compared to the state-action space of the environment [36, 75], which is often the case when the states are based on images. Image-based inputs also pose the challenge of learning the right invariances for the agent to generalize from training to testing environments with different configurations (e.g., maze layouts and environment stochasticity).

4 Procedure cloning

In this section, we first observe that in many imitation learning situations, expert demonstrations can provide much richer insight into the desirable behavior than just the final optimal action. We formulate learning under these situations as *chain of thought imitation*, where an agent can learn from not just the optimal action but the "thought process" an expert goes through before arriving at the final decision. We then formalize such thought process as *procedures*, and propose *procedure cloning* for learning such thought process through supervised sequence prediction.

4.1 Chain of thought imitation

Depending on the form of the expert policy π_* used to generate the training data $\mathcal{D}_*$, an agent potentially has access to a rich set of information about a task which can facilitate learning. For instance, when π_* is some scripting policy following a fixed set of rules [76], the rules (e.g., "if close to enemy then fire") reveal causal information between firing and the enemy disappearing. In other settings when π_* is a search algorithm, the induced search tree exposes the path for finding the optimal action. When π_* is a multi-step hand-coded algorithm, the breakdown of the steps (e.g., "first move to objects then sweep") reveals important ordering information about the task. Even in the case where π_* is a human demonstrator, we can ask the human to explain the thought process that led to their decision. We refer to learning from the procedures (e.g., planning, search, multi-step algorithm) that generated the final action as *chain of thought imitation*.

[1] $\Delta(\mathcal{X})$ denotes the simplex over a set $\mathcal{X}$.

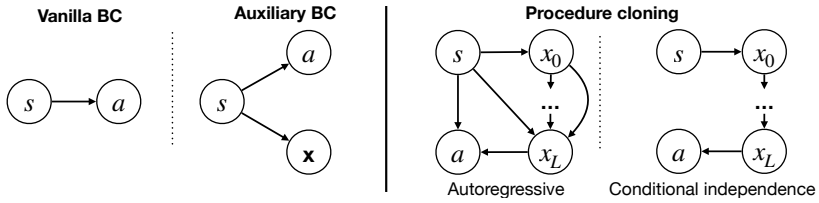

Figure 2: Graphical models of vanilla BC, auxiliary BC, and procedure cloning with autoregressive and conditionally independent factorization. Node s represents an input MDP state, a represents an expert action, and $\mathbf{x}$ represents the sequence of procedure observations $(x_0, ..., x_L)$.

4.2 Procedures and procedure observations

To formulate chain of thought imitation as a learning problem, we define a procedure $\mathbf{\Pi} : S \to \Delta(A)$ as a sequence of computations $(\Pi_0, \Pi_1, ..., \Pi_L, \Pi_{L+1})$ that first transform an input state $s \in S$ into some computation state (e.g., a variable value) using $\Pi_0 : S \to \mathbb{R}^d$, followed by repeatedly applying each subprocedure, $\Pi_\ell : \mathbb{R}^d \to \mathbb{R}^d, \forall \ell \in [1, L]$, to the computation state before mapping the last computation state back to the MDP action space using $\Pi_{L+1} : \mathbb{R}^d \to \Delta(A)$, in a sequential order. A procedure can be broken down to subprocedures at any user-defined granularity (e.g., a function or a for loop), depending on how frequently the computation state is retrieved for procedure learning. Each pair of training data, $(s, a) \sim \mathcal{D}_*$, is acquired through executing such a procedure, i.e., $\pi_* = \mathbf{\Pi}$.

We define *procedure observations* as the sequence of computation states captured from a procedure: $\mathbf{x} = (x_0, ...x_L) \in \mathbb{R}^{(L+1)\times d}$ where $x_0 = \Pi_0(s)$ and $x_\ell = \Pi_\ell \circ \Pi_{l-1} \circ ... \circ \Pi_1 \circ \Pi_0(s), \forall \ell \in [1, L]$. The training data with procedure observations is denoted as $\mathcal{D}_\Pi = \{(s^{(i)}, \mathbf{x}^{(i)}, a^{(i)})\}_{i=1}^n$. Note that we do not assume any procedure or procedure observation is available at test time, hence we still need to learn a policy π that only takes s as input.

4.3 Procedure cloning

With the procedure observations defined above, we are now ready for learning procedures through *procedure cloning* (PC). We model a PC policy by estimating the joint distribution of the procedure observations and the final action conditioned on the input state $p(a, \mathbf{x}|s)$, which we can factorize autoregressively as:

$$p(a, \mathbf{x}|s) = p(a|\mathbf{x}, s) \cdot \Pi_{l=1}^{L} p(x_\ell|\mathbf{x}_{<\ell}, s) \cdot p(x_0|s). \tag{2}$$

Under this factorization, estimating $p(a, \mathbf{x}|s)$ reduces to estimating each conditional factor, which can be parametrized using a transformer model [11]. The autoregressive factorization is highly flexible, but if the amount of expert demonstrations is small, and each procedure observation x_ℓ only depends on the previous procedure observation $x_{\ell-1}$ (i.e., the computation states are fully observed), and the final action only depends on the last computation state, a conditionally independent factorization can be more desirable:

$$p(a, \mathbf{x}|s) = p(a|x_L) \cdot \Pi_{l=1}^{L} p(x_\ell|x_{\ell-1}) \cdot p(x_0|s). \tag{3}$$

The graphical models of the two factorizations of PC policies are shown in Figure 2. To learn a PC policy, we maximize the empirical likelihood of the joint distribution in Equation 2 on given samples $\mathcal{D}_\Pi = \{(s^{(i)}, \mathbf{x}^{(i)}, a^{(i)})\}_{i=1}^n$:

$$\min_{\phi,\theta,\psi} J_{\mathrm{PC}}(\phi, \theta, \psi) = \hat{\mathbb{E}}_{(s,\mathbf{x},a)\sim\mathcal{D}_\Pi}[-\log p(a, \mathbf{x}|s)] \tag{4}$$

$$= \hat{\mathbb{E}}_{(s,\mathbf{x},a)\sim\mathcal{D}_\Pi}\left[-\log q_\psi(a|\mathbf{x}, s) - \sum_{\ell=1}^{L} \log p_\theta(x_\ell|\mathbf{x}_{<\ell}, s) - \log p_\phi(x_0|s)\right]. \tag{5}$$

Connection to BC with auxiliary tasks. The PC objective in Equation 5 can be reduced to the vanilla BC objective in Equation 1 by discarding the second and third term inside the expectation of Equation 5 and setting $q_\psi(a|\mathbf{x}, s) = \pi(a|s)$. We note that several previous works [51, 50, 2, 52, 55, 56] have used procedure information as auxiliary tasks to BC, which may be interpreted as learning $p(a, \mathbf{x}|s)$ under the assumption that a and $\mathbf{x}$ are independent conditioned on s: $p(a, \mathbf{x}|s) = \pi(a|s) \cdot p(\mathbf{x}|s)$. Procedure cloning instead focuses on situations where such a conditional independence assumption does not hold (i.e., a is directly computed by the procedure represented by $\mathbf{x}$), and in

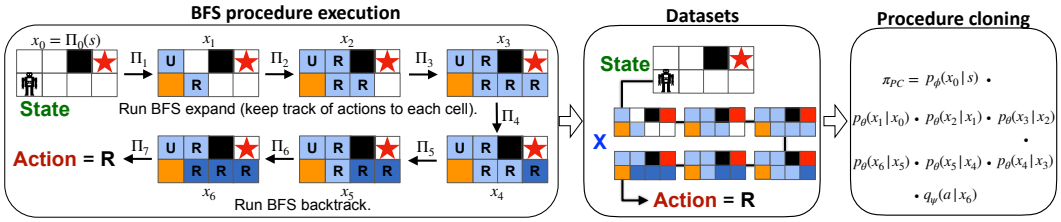

Figure 3: In a discrete maze, the expert employs BFS by first expanding a search perimeter until it encounters the goal cell, at which point it backtracks to find the optimal action at the starting state (cells in light blue are visited and dark blue are backtracked). We encode this algorithm as a sequence of procedure observations $(x_0, ..., x_6)$ of the intermediate computation states, with each x_i represented by a 2D array and each cell of the array containing BFS-relevant information (i.e., whether this cell is being expanded or backtracked and the action recorded when expanding to this cell). Procedure cloning is trained to predict the entire sequence of computations from input state to output action using a sequential model $p(a|x_L) \cdot \Pi_{l=1}^{L} p(x_\ell|x_{\ell-1}) \cdot p(x_0|s)$.

these situations, as we will show in our experiments, treating the procedure information as a precursor to a can perform better than using it as an auxiliary task. The graphical models of vanilla BC and BC with auxiliary task objective are shown in Figure 2.

5 Proof of concept: Synthetic maze navigation

In this section, we study a tabular maze navigation task with synthetically generated maze layouts (see Figure 3). We describe the task setup, followed by how to extract procedure data from a breadth-first search (BFS) path planning algorithm to train a procedure cloning agent, and empirically show that procedure cloning indeed generalizes much better to unseen maze layouts than other BC baselines. While this proof of concept illustration might seem domain-specific to BFS in mazes, we will see in Section 6.3 that procedure cloning can be applied to many types of search or multi-step algorithms.

Task description and evaluation protocol. We use a gridworld maze environment in which an agent seeks to navigate to a goal location in a maze from a random starting location using 4 discrete actions including up (`U`), down (`D`), left (`L`), and right (`R`). The input to the agent is a multi-channel "image" with the maze wall, goal location, and agent location encoded in separate channels. The maze layout is algorithmically generated with random internal walls that form a tunnel-shaped map (see example maze in Figure 4 and details in Appendix A.1). We generate a set of mazes $S_0 \subset S$ and split S_0 into disjoint training S_0^{train} and testing S_0^{test} sets. We then generate expert trajectories by running BFS on only the training set of mazes S_0^{train}. At test time, the agent is evaluated on S_0^{test} by their success rate of navigating to the goal. Figure 3 visualizes the data and training pipeline.

Procedure data collection. BFS is a common path planning algorithm for navigation [77, 78, 79, 80], which we use to generate expert trajectories for training an imitation learning agent. To compute the optimal action at each time step, BFS keeps track of a `visited` 2D array (colored cells in Figure 3) that marks whether each position (1) has been visited by the search, (2) if so, which action visited it, and (3) has a position been backtracked. We simply take a snapshot of the entire `visited` array as procedure observations x_ℓ every time BFS expands the search perimeter, resulting in a series of procedure data $\mathbf{x} = (x_1, ..., x_L)$ as shown in Figure 3. Π_0 is the identity map and $x_0 = s$. See pseudocode for collecting procedure observations in Appendix A.2.

Procedure learning. Since each of the `visited` 2D arrays x_ℓ only depends on the previous `visited` array $x_{\ell-1}$, and the final `visited` array after backtracking uniquely identifies the expert action on its own, we choose the conditionally independent factorization of $p(a, \mathbf{x}|s)$ (Equation 3) described in Section 4.3. Specifically, we parametrize $p_\theta(x_\ell|x_{\ell-1}), \forall \ell \in [1, L]$ using a deep convolutional neural network that takes in the current `visited` array $x_{\ell-1}$ as input and produces the next `visited` array x_ℓ as output. We optimize θ using the cross-entropy loss between the predicted and true next `visited` array. Since the procedure observations x_ℓ and the original input s are in the same image space, $\phi(x)$ shares the same parameters as $p_\theta(x_\ell|x_{\ell-1})$. During inference when a new test maze layout s is given, we apply $\hat{x}_0 = \phi(s)$ and $\hat{x}_\ell \sim p_\theta(\cdot|\hat{x}_{\ell-1})$ repeatedly until an array $\hat{x}_L$ is predicted for which the entry in $\hat{x}_L$ corresponding to the agent's current location is labelled as "backtracked", and we return the backtracked action as the final output action.

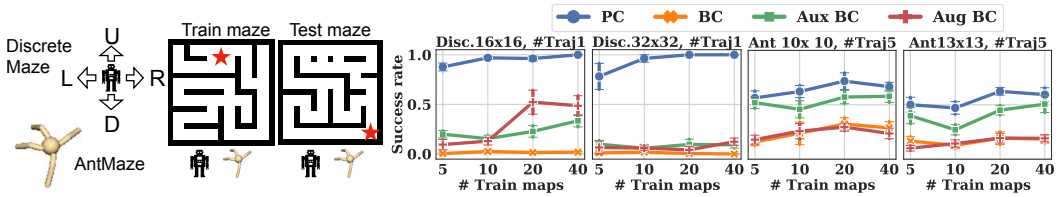

Figure 4: [Left] Visualization of the discrete maze (4 discrete actions) and AntMaze (8 continuous actions). [Right] Average success rate of PC and BC agents navigating to the goal from random start locations over 10 test mazes. Agents are trained on 5, 10, 20, 40 mazes of 1 and 5 expert trajectories on discrete maze and AntMaze, respectively. We find that procedure cloning leads to much better test maze generalization compared to alternative approaches.

Generalization results. We compare procedure cloning to applying data augmentation with random crop, translation, and zoom (Aug BC) or auxiliary objective of predicting the `visited` array (Aux BC) to vanilla BC. BC policies are parametrized with convolutional neural networks (CNN) and multi-layer perceptrons (MLPs) (see hyperparameters in Appendix A.5). Figure 4 (Disc.16 × 16 and Disc.32 × 32) shows the average success rate (over 5 trajectories) of reaching the goal from random start locations on 10 test maze layouts unseen during training. Procedure cloning successfully generalizes to test mazes, whereas vanilla BC completely fails to learn (0% sucess rate) in bigger maze 32 × 32. Aux BC and Aug BC help in the smaller maze but not in the bigger maze. The poor test performance of BC is due to generalization failure as opposed to insufficient model capacity as BC's success rate on the training mazes are close to 100% (see Figure 10 in Appendix B.2). We also include additional experiments using VIN [81] as a baseline in Appendix B.1, showing that even an approach specialized to the structure of gridworld tasks exhibits poor generalization.

6 Experiments

We now evaluate procedure cloning in larger scale settings on tasks including simulated robotic navigation [82] and manipulation [83, 14], and learning to play MinAtar [84] (a miniature version of Atari [85]). Procedure cloning exhibits significant generalization to previously unseen maze layouts, positions of objects being manipulated, and environment configurations such as transition stochasticity and game difficulty in each of the tasks, respectively. See Appendix B for more results.

6.1 Evaluating continuous robot navigation in AntMaze

Task description. Following the discrete maze navigation task in Section 5, we now consider a more realistic setting of mimicking real-world robotic navigation. We adopt the AntMaze environment from D4RL [82], where an 8-DoF "Ant" quadruped robot with continuous state and action spaces is placed in a 2D maze environment. The original task designed for offline RL only has one maze layout which is not revealed to the agent (the agent only sees its current position and joint-based state measurements). To adopt the task for evaluating generalization across maze layouts, we also pass the algorithmically generated maze layouts from Section 5 to the agent as inputs. During evaluation, the agent needs to navigate to the goal in previously unseen maze layouts.

PC implementation. The procedure that generated the expert training trajectories for goal-reaching in AntMaze involves a low-level PID controller that is good for navigating to local locations close to the robot and a high-level waypoint generator which uses BFS search to find the next local waypoint [82]. We apply procedure cloning to the high-level waypoint generator following the same steps as in discrete maze described in Section 5. The predicted waypoint is then passed to a Gaussian parametrized policy (optimized with max-likelihood) together with the agent's joint measurements. For BC and variants, we use a convolutional neural network to embed the maze layout into a fixed dimensional vector and concatenate the robot joint measurements together with the agent and goal locations to a Gaussian policy.

Generalization results. Figure 4 (Ant 10 × 10 and Ant 13 × 13) shows the average (over 5 trajectories) success rate of the robot reaching the goal from random starting locations on 10 test mazes. Procedure cloning consistently provides significant benefits over vanilla BC, whereas data augmentation on the maze layout and predicting the `visited` array as an auxiliary loss are less helpful. We know the poor test performance of BC (and other baseline methods) is more due to generalization failure as opposed to insufficient model capacity because the success rate on the training mazes between procedure cloning and auxiliary BC are similar (Figure 11 in Appendix B.3).

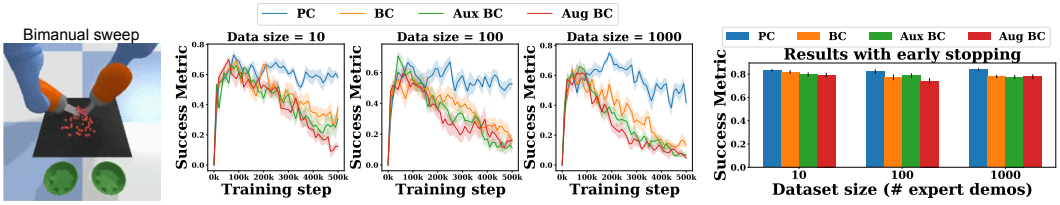

Figure 5: [Left] Visualization of the bimanual sweep task. [Middle] Average success metric (proportion of particles in bowls at the end of the episode) of PC and BC agents completing the bimanual sweeping task after learning on 10, 100, 1000 expert trajectories; each variant is an aggregate of 10 runs. All of our algorithm implementations use the implicit loss function described in [14] for this task. [Right] When using 1000 expert demonstrations with early stopping, PC achieves 83.9% compared to 78.2% success of the existing state-of-the-art achieved by implicit BC.

6.2 Evaluating image-based robot manipulation

Task description. The bimanual sweeping task [83, 14] requires two 7-DoF robot arms equipped with spatula-like end-effectors to sweep a pile of particles evenly into two bowls while avoiding dropping particles between the tips of the spatulas. The scripted oracle for collecting expert trajectories uses access to privileged information including object poses and contact points, which are not accessible at test time. Rather, only high-resolution (96×96) images in conjunction with end-effector positions and orientations are given to the imitation-learned agent during inference. Actions are end-effector positions for each robot arm (6 dimensions for each arm for a total of 12 dimensions for the action); for how to incorporate kinematic actions into this environment, we refer the reader to [83]. Random seeds determining the initial position of the particles are partitioned so that test-time configurations are not seen in training.

PC implementation and results. The scripted policy first scoops the particles by moving the spatulas to their computed geometric center and then moves the spatulas to a bowl before releasing the particles. At each state of a trajectory, we collect the Cartesian coordinates of one of these two goal points – geometric center of the particles or position of the bowl – depending on the expert's behavior mode at that state, and use these as procedure observations. We supervise the PC agent to first predict these coordinates, then predict the actions conditioned on the predicted coordinates and end-effector positions and orientations. For both BC and PC we parameterize log-likelihood using energy-based models, also known as an *implicit* loss, which is state-of-the-art for this task [14]. We compare PC to BC trained directly on image inputs and auxiliary BC which learns to predict oracle coordinates as an auxiliary objective. We see that the BC variants quickly overfit to the training set of observations, whereas PC generalizes much better (Figure 5, left). Despite early stopping (taking the maximum evaluation success rate over all training steps), PC still outperforms all of the BC variants (Figure 5, right), improving significantly over the state-of-the-art.

6.3 Evaluating strategy games in MinAtar

Task description. MinAtar is a minature version of the Atari Arcade Learning Environment [85] consisting of 5 games with simplified 10×10 multi-channel images as inputs. To generate the expert trajectories for training, we run a AlphaZero-style [15] Monte-Carlo tree search (MCTS) algorithm on the deterministic version of the environments to collect expert trajectories (see Appendix A.3 for details). During evaluation, we test the imitation-learned agents on a set of test environments with different seeds from the training environments where expert trajectories are collected. To further evaluate generalization, we apply sticky actions with probability 0.1 and game difficulty ramping to the test environments where such an option is available.

PC implementation. In contrast to running BFS in maze navigation where the `visited` array cleanly captures the entire state of the search, running MCTS in MinAtar is more convoluted, involving a number of MCTS simulation runs each with selection, expansion, rollouts, and backtrack steps and different tree structures, making capturing the full search state difficult. Fortunately, procedure cloning with autoregressive factorization (Equation 2) allows procedure data to be partial observations of the computation state, and so we elect to use only a subset of the MCTS computation states as supervision for PC. Namely, we record the optimal action sequence *after* the last MCTS simulation run (from the final search tree) and the goal image at the tree leaf as procedure observations (highlighted in blue in Figure 6); this is effectively the optimal future trajectory determined by MCTS. A PC policy is trained to use the input state to first predict the goal image using a CNN and then use

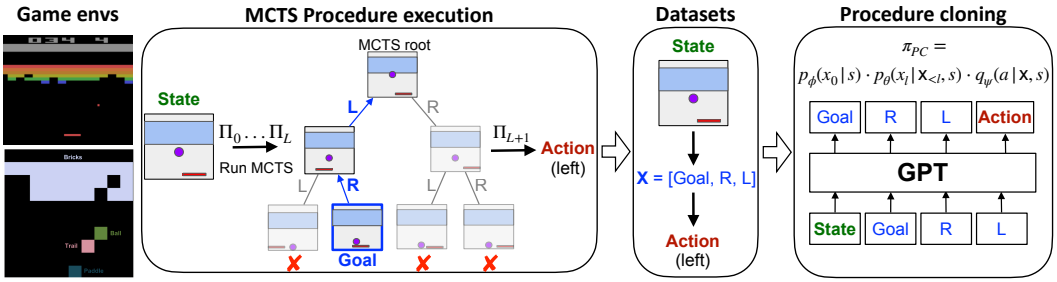

Figure 6: In the MinAtar game-playing environment, the expert uses MCTS $(\Pi_0, ..., \Pi_L)$ to find an optimal future trajectory [L, R, Goal]. We treat this future trajectory in reverse order [Goal, R, L] as procedure observations, so that procedure cloning is trained to first predict the goal image (MCTS leaf node) and then predict the optimal action sequence backwards from the goal using a GPT-like autoregressive model, ultimately predicting the expert's output action as its last prediction.

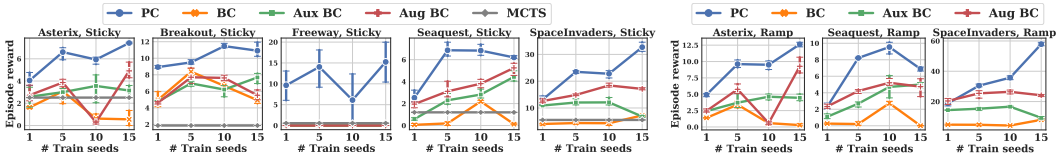

Figure 7: Average episode reward (over 50 episodes) of PC and BC agents playing MinAtar games over 3 test environments using sticky actions (left) and game difficulty ramping (right) not see in the training environments.

the goal as input to an autoregressive action sequence model $p(x_\ell|\mathbf{x}_{<\ell}, s)$ where x_ℓ is the optimal action that is ℓ-steps away from the goal (i.e., predicting the optimal action sequence *backwards* from the goal). Figure 6 illustrates the data and training pipeline of procedure cloning.

Generalization results. Figure 7 shows the average reward (over 50 episodes) collected by running PC and BC policies in 3 test environments with different environment seeds than the environments used to collect training trajectories. Figure 7 (left) evaluates generalization to stochastic environments, where we found sticky actions caused MCTS (gray) to struggle to search for good actions without extensive tuning. Figure 7 (right) evaluates generalization to more difficult game settings (available in 3 out of 5 MinAtar games). Autoregressive PC policy generalizes the best across all games and all settings.

7 Conclusion

We have identified a major gap between the imitation learning problem setup and how training data for imitation are often obtained in practice. Specifically, applications that use planning, search, or other multi-step algorithms reveal the procedure for determining an expert action in addition to outputting the expert action itself. This led us to formulate chain of thought imitation learning, where an agent is also given access to the intermediate computations that generated the expert state-action pairs. In order to generalize to test environments where intermediate computation is not available, we propose procedure cloning, which learns to predict the intermediate computation outcomes step-by-step using a sequence model, and emulates the intermediate computations during inference through autoregressive generation. Evaluation on a variety of navigation, manipulation, and game play tasks shows that procedure cloning generalizes much better to test environments of different configurations than alternative improvements over behavioral cloning. We hope the success of applying sequence modeling to imitation learning achieved by procedure cloning opens up new research directions in the intersection of large-scale sequence modeling and sequential decision making.

Limitations. One major limitation of procedure cloning compared to traditional BC is in the computational overhead, since PC needs to predict intermediate procedures. Furthermore, the choice of how to encode the expert's algorithm into a form amenable to PC is up to the practitioner. While we have presented ways to encode a variety of policies here (BFS, MCTS, scripted robotic policies), applying PC to other domains may require some amount of trial-and-error in designing the ideal computation sequence for PC.

References

[1] Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

[2] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, et al. Reinforcement and imitation learning for diverse visuomotor skills. *arXiv preprint arXiv:1802.09564*, 2018.

[3] Peter Jarvis. *Towards a comprehensive theory of human learning*. Routledge, 2012.

[4] George Katona. Organizing and memorizing: studies in the psychology of learning and teaching. 1940.

[5] Michael L Crawford. Teaching contextually. *Research, rationale, and techniques for improving student motivation and achievement in mathematics and science. Texas: Cord*, 2001.

[6] Heidi L Lujan and Stephen E DiCarlo. Too much teaching, not enough learning: what is the solution? *Advances in physiology education*, 30(1):17–22, 2006.

[7] James Bruce and Manuela M Veloso. Real-time randomized path planning for robot navigation. In *Robot soccer world cup*, pages 288–295. Springer, 2002.

[8] Richard M Murray, Zexiang Li, and S Shankar Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 2017.

[9] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[10] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA ARTIFICIAL INTELLIGENCE AND PSYCHOLOGY . . . , 1989.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[12] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[13] Charles Thorpe and Jay Gowdy. Annotated maps for autonomous land vehicles. In *1990 IEEE International Conference on Systems, Man, and Cybernetics Conference Proceedings*, pages 282–288. IEEE, 1990.

[14] Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Conference on Robot Learning*, pages 158–168. PMLR, 2022.

[15] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[16] Tristan Cazenave and Nicolas Jouandeau. A parallel monte-carlo tree search algorithm. In *International Conference on Computers and Games*, pages 72–80. Springer, 2008.

[17] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2):235–256, 2002.

[18] Sébastien Bubeck and Nicolo Cesa-Bianchi. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *arXiv preprint arXiv:1204.5721*, 2012.

[19] Alekh Agarwal, Daniel Hsu, Satyen Kale, John Langford, Lihong Li, and Robert Schapire. Taming the monster: A fast and simple algorithm for contextual bandits. In *International Conference on Machine Learning*, pages 1638–1646. PMLR, 2014.

[20] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668. JMLR Workshop and Conference Proceedings, 2010.

[21] Peter Auer, Thomas Jaksch, and Ronald Ortner. Near-optimal regret bounds for reinforcement learning. *Advances in neural information processing systems*, 21, 2008.

[22] Alexander L Strehl, Lihong Li, and Michael L Littman. Reinforcement learning in finite mdps: Pac analysis. *Journal of Machine Learning Research*, 10(11), 2009.

[23] Christoph Dann, Tor Lattimore, and Emma Brunskill. Unifying pac and regret: Uniform pac bounds for episodic reinforcement learning. *Advances in Neural Information Processing Systems*, 30, 2017.

[24] Mohammad Gheshlaghi Azar, Ian Osband, and Rémi Munos. Minimax regret bounds for reinforcement learning. In *International Conference on Machine Learning*, pages 263–272. PMLR, 2017.

[25] Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018.

[26] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

[27] Matthew Hausknecht and Peter Stone. The impact of determinism on learning atari 2600 games. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

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

[29] Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, et al. Massively parallel methods for deep reinforcement learning. *arXiv preprint arXiv:1507.04296*, 2015.

[30] Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[31] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[32] Jesse Farebrother, Marlos C Machado, and Michael Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.

[33] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pages 1282–1289. PMLR, 2019.

[34] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[35] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. *arXiv preprint arXiv:1910.05396*, 2019.

[36] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[37] Hoang Le, Nan Jiang, Alekh Agarwal, Miroslav Dudik, Yisong Yue, and Hal Daumé III. Hierarchical imitation and reinforcement learning. In *International conference on machine learning*, pages 2917–2926. PMLR, 2018.

[38] Michael H Lim, Andy Zeng, Brian Ichter, Maryam Bandari, Erwin Coumans, Claire Tomlin, Stefan Schaal, and Aleksandra Faust. Multi-task learning with sequence-conditioned transporter networks. *arXiv preprint arXiv:2109.07578*, 2021.

[39] Sanjeev Arora, Simon Du, Sham Kakade, Yuping Luo, and Nikunj Saunshi. Provable representation learning for imitation learning via bi-level optimization. In *International Conference on Machine Learning*, pages 367–376. PMLR, 2020.

[40] Amy Zhang, Rowan McAllister, Roberto Calandra, Yarin Gal, and Sergey Levine. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*, 2020.

[41] Mengjiao Yang and Ofir Nachum. Representation matters: Offline pretraining for sequential decision making. In *International Conference on Machine Learning*, pages 11784–11794. PMLR, 2021.

[42] Ofir Nachum and Mengjiao Yang. Provable representation learning for imitation with contrastive fourier features. *Advances in Neural Information Processing Systems*, 34, 2021.

[43] Mengjiao Yang, Sergey Levine, and Ofir Nachum. Trail: Near-optimal imitation learning with suboptimal data. *arXiv preprint arXiv:2110.14770*, 2021.

[44] Thomas Kipf, Yujia Li, Hanjun Dai, Vinicius Zambaldi, Alvaro Sanchez-Gonzalez, Edward Grefenstette, Pushmeet Kohli, and Peter Battaglia. Compile: Compositional imitation learning and execution. In *International Conference on Machine Learning*, pages 3418–3428. PMLR, 2019.

[45] Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. Opal: Offline primitive discovery for accelerating offline reinforcement learning. *arXiv preprint arXiv:2010.13611*, 2020.

[46] Kourosh Hakhamaneshi, Ruihan Zhao, Albert Zhan, Pieter Abbeel, and Michael Laskin. Hierarchical few-shot imitation with skill transition models. *arXiv preprint arXiv:2107.08981*, 2021.

[47] Arun Venkatraman, Martial Hebert, and J Andrew Bagnell. Improving multi-step prediction of learned time series models. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[48] Jonathan Chang, Masatoshi Uehara, Dhruv Sreenivas, Rahul Kidambi, and Wen Sun. Mitigating covariate shift in imitation learning via offline data with partial coverage. *Advances in Neural Information Processing Systems*, 34, 2021.

[49] Rafael Rafailov, Tianhe Yu, Aravind Rajeswaran, and Chelsea Finn. Visual adversarial imitation learning using variational models. *Advances in Neural Information Processing Systems*, 34, 2021.

[50] Long-Ji Lin and Tom M Mitchell. *Memory approaches to reinforcement learning in non-Markovian domains*. Citeseer, 1992.

[51] Xiujun Li, Lihong Li, Jianfeng Gao, Xiaodong He, Jianshu Chen, Li Deng, and Ji He. Recurrent reinforcement learning: a hybrid approach. *arXiv preprint arXiv:1509.03044*, 2015.

[52] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[53] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020.

[54] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, pages 66–75. PMLR, 2020.

[55] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[56] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

[57] Andrew K Lampinen, Nicholas A Roy, Ishita Dasgupta, Stephanie CY Chan, Allison C Tam, James L McClelland, Chen Yan, Adam Santoro, Neil C Rabinowitz, Jane X Wang, et al. Tell me why!–explanations support learning of relational and causal structure. *arXiv preprint arXiv:2112.03753*, 2021.

[58] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[59] Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*, 2017.

[60] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[61] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

[62] Peter Clark, Oyvind Tafjord, and Kyle Richardson. Transformers as soft reasoners over language. *arXiv preprint arXiv:2002.05867*, 2020.

[63] Mohammed Saeed, Naser Ahmadi, Preslav Nakov, and Paolo Papotti. Rulebert: Teaching soft rules to pre-trained language models. *arXiv preprint arXiv:2109.13006*, 2021.

[64] Nazneen Fatema Rajani, Bryan McCann, Caiming Xiong, and Richard Socher. Explain yourself! leveraging language models for commonsense reasoning. *arXiv preprint arXiv:1906.02361*, 2019.

[65] Alon Talmor, Oyvind Tafjord, Peter Clark, Yoav Goldberg, and Jonathan Berant. Teaching pre-trained models to systematically reason over implicit knowledge. 2020.

[66] Zhengzhong Liang, Steven Bethard, and Mihai Surdeanu. Explainable multi-hop verbal reasoning through internal monologue. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1225–1250, 2021.

[67] Eric Zelikman, Yuhuai Wu, and Noah D Goodman. Star: Bootstrapping reasoning with reasoning. *arXiv preprint arXiv:2203.14465*, 2022.

[68] Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. Stabilizing transformers for reinforcement learning. In *International Conference on Machine Learning*, pages 7487–7498. PMLR, 2020.

[69] 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, 2021.

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

[71] Hiroki Furuta, Yutaka Matsuo, and Shixiang Shane Gu. Generalized decision transformer for offline hindsight information matching. *arXiv preprint arXiv:2111.10364*, 2021.

[72] Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer. *arXiv preprint arXiv:2202.05607*, 2022.

[73] Machel Reid, Yutaro Yamada, and Shixiang Shane Gu. Can wikipedia help offline reinforcement learning? *arXiv preprint arXiv:2201.12122*, 2022.

[74] Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[75] Stephen Tu, Alexander Robey, and Nikolai Matni. Closing the closed-loop distribution shift in safe imitation learning. *arXiv preprint arXiv:2102.09161*, 2021.

[76] Steven Rabin. *Game AI pro: collected wisdom of game AI professionals*. CRC Press, 2013.

[77] Don Murray and James J Little. Using real-time stereo vision for mobile robot navigation. *autonomous robots*, 8(2):161–171, 2000.

[78] Wael Sabra, Matthew Khouzam, Arnaud Chanu, and Sylvain Martel. Use of 3d potential field and an enhanced breadth-first search algorithms for the path planning of microdevices propelled in the cardiovascular system. In *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, pages 3916–3920. IEEE, 2006.

[79] Soumabha Bhowmick, Abhishek Pant, Jayanta Mukherjee, and Alok Kanti Deb. A novel floor segmentation algorithm for mobile robot navigation. In *2015 Fifth National Conference on Computer Vision, Pattern Recognition, Image Processing and Graphics (NCVPRIPG)*, pages 1–4. IEEE, 2015.

[80] Hrudaya Kumar Tripathy, Sushruta Mishra, Hiren Kumar Thakkar, and Deepak Rai. Care: a collision-aware mobile robot navigation in grid environment using improved breadth first search. *Computers & Electrical Engineering*, 94:107327, 2021.

[81] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. *Advances in neural information processing systems*, 29, 2016.

[82] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[83] Aditya Ganapathi, Pete Florence, Jake Varley, Kaylee Burns, Ken Goldberg, and Andy Zeng. Implicit kinematic policies: Unifying joint and cartesian action spaces in end-to-end robot learning. *arXiv preprint arXiv:2203.01983*, 2022.

[84] Kenny Young and Tian Tian. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. *arXiv preprint arXiv:1903.03176*, 2019.

[85] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[86] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[87] Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, et al. Acme: A research framework for distributed reinforcement learning. *arXiv preprint arXiv:2006.00979*, 2020.

[88] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[89] Scott Reed and Nando De Freitas. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

[90] Danfei Xu, Suraj Nair, Yuke Zhu, Julian Gao, Animesh Garg, Li Fei-Fei, and Silvio Savarese. Neural task programming: Learning to generalize across hierarchical tasks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3795–3802. IEEE, 2018.

[91] Yiming Ding, Carlos Florensa, Pieter Abbeel, and Mariano Phielipp. Goal-conditioned imitation learning. *Advances in neural information processing systems*, 32, 2019.

[92] Ajay Mandlekar, Fabio Ramos, Byron Boots, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Dieter Fox. Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4414–4420. IEEE, 2020.

[93] Aravind Srinivas, Allan Jabri, Pieter Abbeel, Sergey Levine, and Chelsea Finn. Universal planning networks: Learning generalizable representations for visuomotor control. In *International Conference on Machine Learning*, pages 4732–4741. PMLR, 2018.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See last paragraph in Section 6.3.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Appendix A and supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix A.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See all plots in Section 6.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix A.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 6.

(b) Did you mention the license of the assets? [Yes] The licenses are Apache License 2.0 and GNU general public license.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] See supplementary material.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Experimental details

A.1 Maze generation details

We algorithmically generate maze layouts in a 2D gridworld environment. Each maze contains a randomly generated goal location and randomly generated internal walls that form a tunnel-like map as shown in Figure 8. The long corridors of the maze layouts make it difficult to learn a good navigation policy with little data, especially when the maze is large. The maze layout generator ensures that there is at least one valid path from any empty location to the goal.

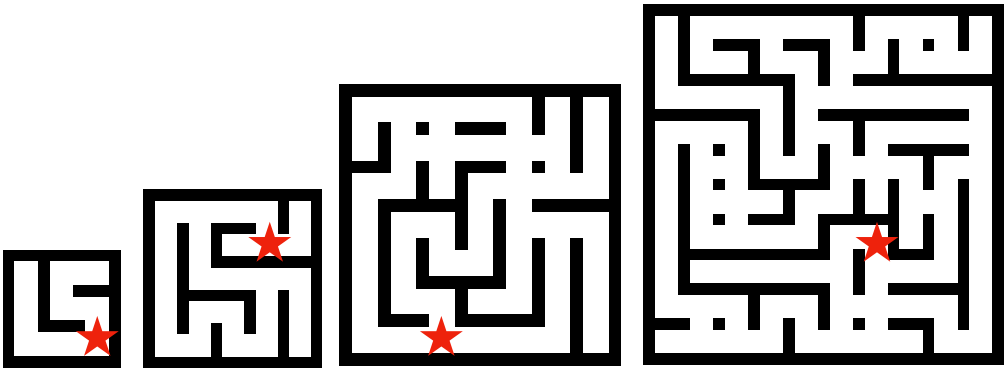

Figure 8: Example maze layouts with increasing maze size.

A.2 Pseudocode for BFS and MCTS procedure data extraction

Algorithm 1 and Algorithm 2 show the pseudocode of the BFS and MCTS procedures for computing the expert action used in our experiments. Procedure observations for training a PC agent is colored in blue (i.e., the `visited` array for BFS and `[goal, traj]` list for MCTS). Procedure observations are added to the training data $\mathcal{D}_{\Pi}$ as annotated. Algorithm 2 shows the UCT [86] action selection as a part of MCTS, but in practice we only record *greedy* action selection *after* running MCTS simulations as procedure observations.

Algorithm 1 BFS procedure annotation

Inputs start location s_0, queue = $[s_0]$
Initialize `visited` ▷ `Track the action that visits each cell.`
while queue **do** ▷ `Expand search.`
  s = queue.dequeue()
  for a, s' in actions, neighbors **do**
    if `visited`$[s']$ **then** continue
    `visited`$[s']$ = a
    queue.enqueue(s')
  end for ▷ ``visited'' $\rightarrow \mathcal{D}_\Pi$.
while not `visited`$[s_0]$ **do** ▷ `Backtrack.`
  `visited`[parent[s]] = `visited`$[s]$
  ▷ ``visited'' $\rightarrow \mathcal{D}_\Pi$.
  s = parent[s]
Return `visited`$[s_0]$

Algorithm 2 MCTS procedure annotation

Inputs root node n, prior p
Initialize `traj` $= [n]$
while node.children **do**
  $a = \text{UCT}(n)$ ▷ `Selection.`
  $n = n.\text{children}[a]$
  `traj` += $[n]$ ▷ `Add ``traj'' to` $\mathcal{D}_\Pi$.
$n = n.\text{expand}(p)$ ▷ `Expansion.`
v, `goal` $= \text{rollout}(n)$ ▷ `Simulation.`
▷ `Add ``goal'' to` $\mathcal{D}_\Pi$.
while `traj` **do**
  $n =$ `traj`.pop()
  update(n, v)
Return `argmax`(n.children.v)

A.3 Details of training the AlphaZero-style MCTS expert

For the expert MCTS policy on MinAtar, we modify the AlphaZero-style MCTS agent implementation from the Acme [87] library[2]. We use the MinAtar environment simulator for rollouts (as opposed to learning an environment model). During each MCTS simulation run, an action is selected similar to UCT [86] according to:

$$a_{\text{search}} = \arg\max_a \left[Q(s,a) + \beta \frac{\sqrt{N(s)}}{N(s,a)+1} \pi_{\text{prior}}(a \mid s) \right], \tag{6}$$

where $Q(s,a)$ is a learned value function using TD-learning, N is the visit-count of the tree node corresponding to s, β is a hyperparameter for controlling how greedily to search, and π_{prior} is a prior for guiding the MCTS search and is updated by minimizing $D_{\text{KL}}\left(\pi_{\text{prior}}(\cdot|s) \| \pi_{\text{MCTS}}(\cdot|s)\right)$, where $\pi_{\text{MCTS}}(\cdot|s)$ is the softmax of the node s's children visit-counts. The hyperparameters for training the MCTS expert is shown in Table 1.

Table 1: MCTS hyperparameters on MinAtar.

Hyperparameter	Value
Number of simulation runs	500
UCT β	1.0
TD discount γ	0.99
TD buffer size	100,000
Learning rate	0.0003
Batch size	64

A.4 Details for training PC policies

Predicting multiple goals on MinAtar. Since we do not limit the search depth of the MCTS agent explicitly, the resulting search tree after 500 simulation runs can have arbitrarily long branches. Therefore, for collecting procedure data, we set a fixed depth (i.e., 15) as the goal which represents the state 15 steps into the future, and the GPT-like [88] autoregressive prediction model is trained on sequence of length 15 actions. We further found that predicting more than a single goal along the optimal path and more than a single backward action sequence during each time step improve PC's performance.

Number of expert samples in $\mathcal{D}_\Pi$. The timeout for collecting expert trajectories and for evaluating imitation-learned agents are set to 100, 1000, 2500 for discrete maze, AntMaze, and MinAtar respectively. Because some MinAtar games are easier to learn than others, i.e., Freeway only requires

[2]Code of Acme: https://github.com/deepmind/acme

a single expert trajectory to learn a good BC policy, we take only the first 50 samples of a single trajectory as expert data for Freeway to increase the challenge of imitation learning. For the rest of the games, we take the full trajectories. Results in the main text (Figure 7) use 10 expert trajectories for each game. Results for other number of expert trajectories can be found in Appendix B.4.

PC timeout. During evaluation, PC needs to predict the stopping condition to output a final action after predicting a series of intermediate procedures. This can result in the intermidiate procedure being arbitrarily long. We therefore use the environment timeout steps as a hard stopping condition for PC, and output an action sampled uniformly at random if PC times out.

A.5 Architecture and hyperparameters

Architecture. The CNNs used in discrete maze, AntMaze, and MinAtar of our experiments have 5 layers of convolution with kernel size (3, 3, 3, 3, 3), stride (1, 1, 1, 1, 1), channel (128, 128, 256, 256, 256), `SAME` padding, and without any pooling. The MLPs in discrete maze, AntMaze, and MinAtar have 2 hidden layers of 256 units each interleaved with ReLU activation. We have experimented with larger CNN and MLP model capacity, but they do not improve the generalization performance of the agents. The transformer autoregressive model in the MinAtar experiment has 4 self-attention layers of 8 heads with 128 hidden units each, followed by 2 feedforward layers of 256 units interleaved with ReLU activation. The bimanual sweep experiment follows the same architecture as [14] – this original architecture already performs autoregressive modelling on actions, so it is straightforward to incorporate procedure observations into this model, treating them as additional predictions that appear before the output actions. For all data augmentation baseline, we apply random crop of the size of the image, random horizontal and vertical translation of [-10%, +10%], and random horizontal and vertical zoom of [-10%, +10%] sequentially.

Training. For discrete maze, AntMaze, and MinAtar, we use the Adam optimizer with learning rate 3e-4, train the models for 500k steps with batch size 32, and evaluate every 10k steps. The line plots we present show the average results of the the last 3 evaluations. Each model is trained on a single NVIDIA P100 GPU. For bimanual sweep, we follow the same training procedure as [14] using TPU v2.

B Additional experimental results

B.1 Results on value iteration network

We were curious about whether having a planning-like (procedure-like) module in the policy parametrization similar to Value Iteration Network (VIN) [81] can solve the generalization task in our maze setup. We therefore evaluate VIN on 5, 10, 20, 40 training mazes of size 8×8, 16×16, and 32×32 with 1 and 16 trajectories each and test if VIN can generalize to the 10 unseen test mazes. The original VIN paper used $k = 10$ (the number of VI blocks) and $k = 20$ for maze size 8×8, 16×16, which we follow, and use $k = 40$ for maze 32×32. Figure 9 shows that VIN does not generalize in our setting, likely due to VIN heavily relying on heavily on the grid structure of the task, where as our tunnel-shaped map is highly convoluted and requires a lot more training data (VIN was originally trained on 5000 training mazes with 7 trajectories each, which is way more data than our evaluation setting).

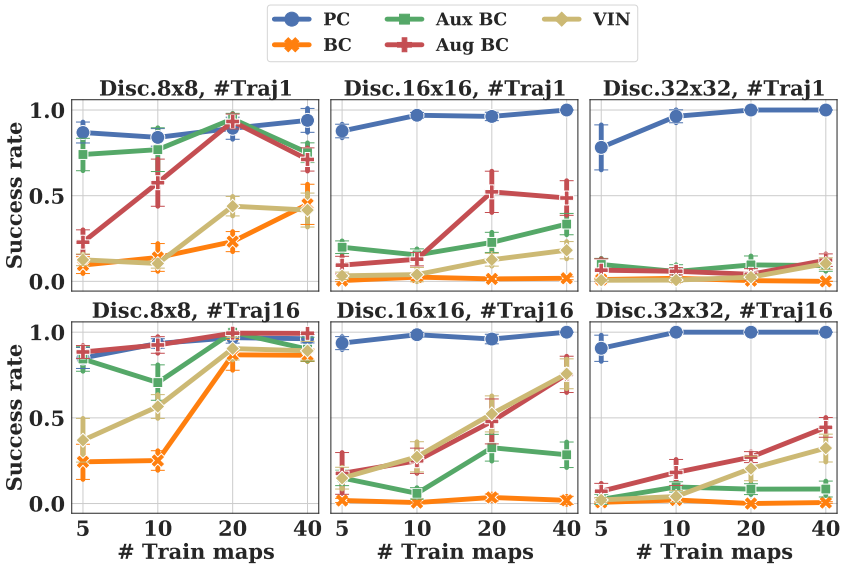

Figure 9: Average success rate of PC, BC (variants), and VIN navigating to the goal from random start locations over 10 test maze layouts in discrete maze.

B.2 Additional results on discrete maze

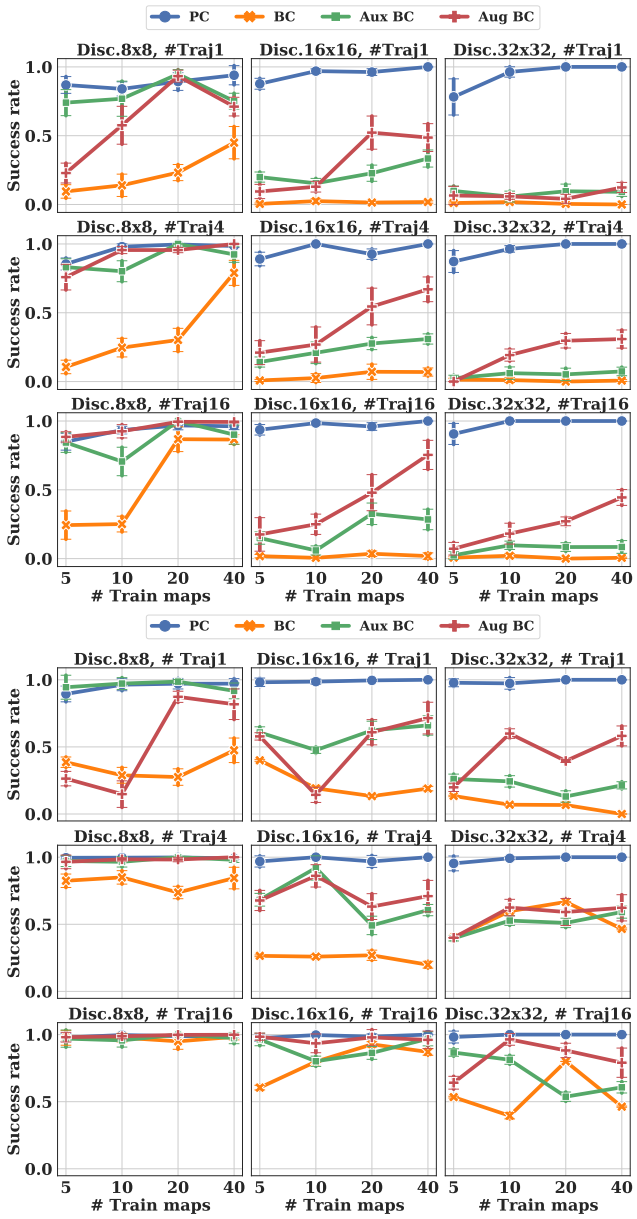

Figure 10: Average success rate of PC and BC agents navigating to the goal from random start locations over testing mazes (top) and training mazes (bottom) in discrete maze‘. While BC variants can achieve high training performance when the number of expert trajectories is large, they still exhibit poor generalization performance on test mazes.

B.3 Additional results on AntMaze

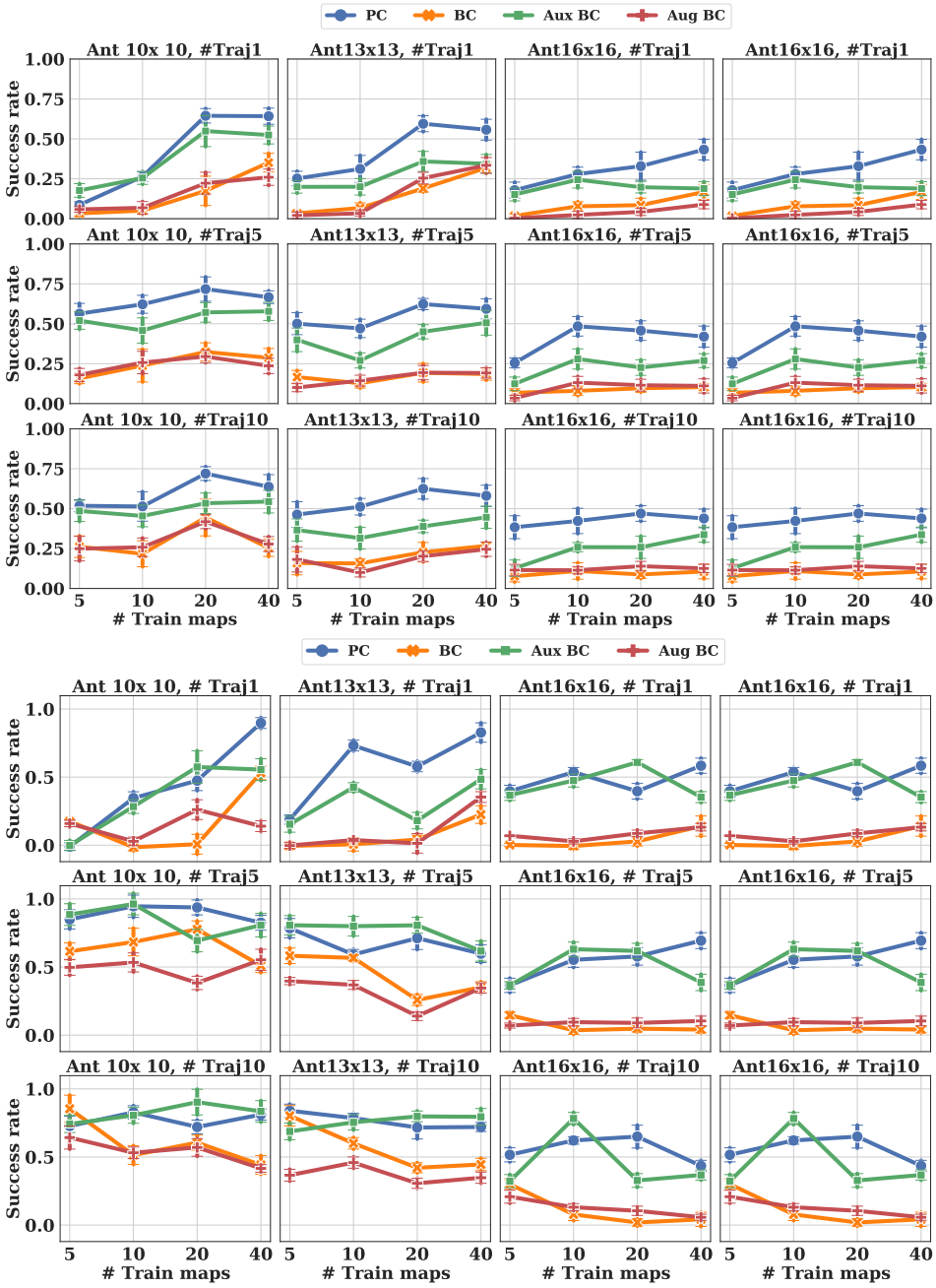

Figure 11: Average success rate of PC and BC agents navigating to the goal from random start locations over testing mazes (top) and training mazes (bottom) in AntMaze. While Aux BC can achieve similar training performance as PC, Aux BC still exhibits poorer generalization performance on test mazes.

B.4 Additional results on MinAtar

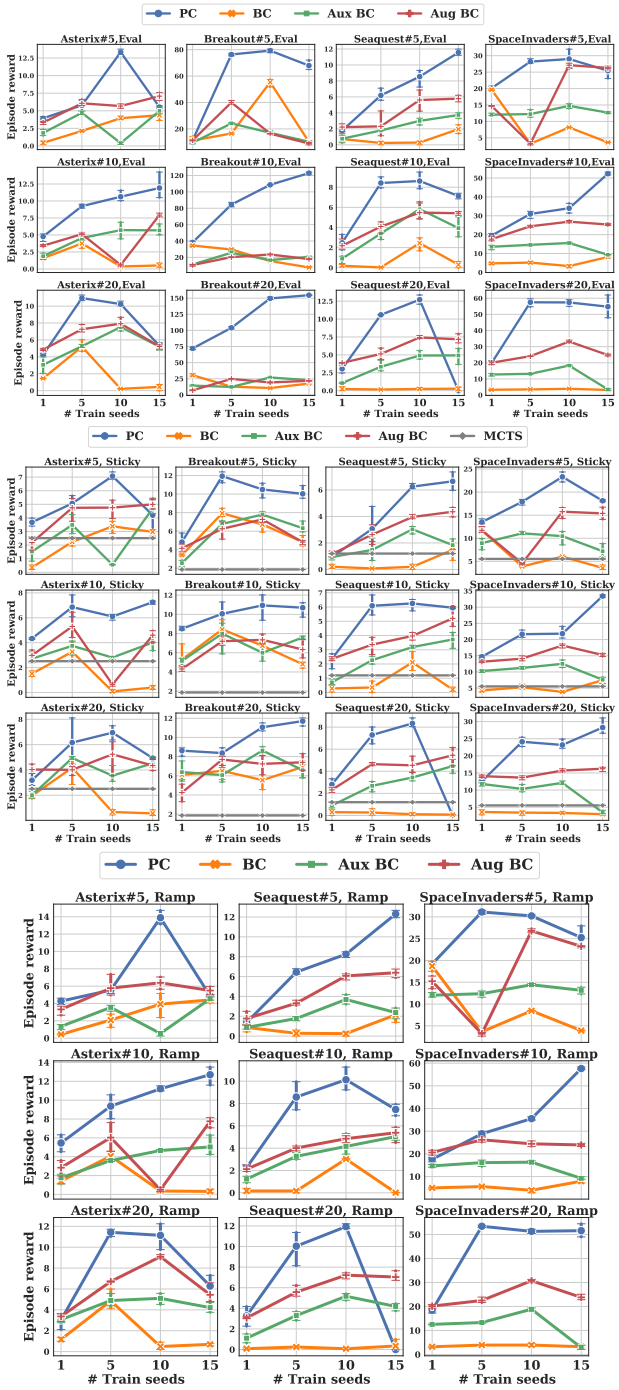

Figure 12: Average episode reward of PC and BC on MinAtar over 3 test environments with the same configuration as training (top), with sticky actions (middle), and with difficulty ramping (bottom). Rows have different number of expert trajectories (50, 10, 20). PC generally provides gains over BC in different evaluation settings.

C Further analysis of procedure cloning

C.1 Compounding error of PC during evaluation

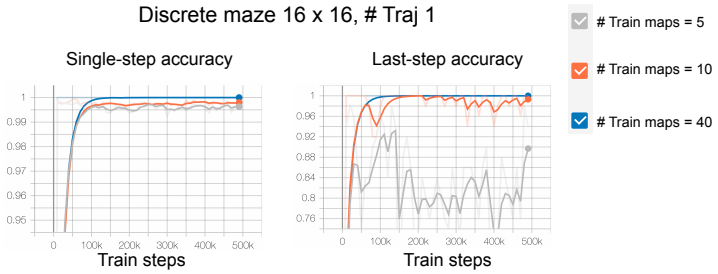

Figure 13: Per-step accuracy of predicting visitation map and final expert action prediction accuracy with compounding error evaluated on discrete maze of size 16x16 where a single trajectory from each maze layout is used for training. The compounding error quickly becomes negligible as the number of training mazes increases (i.e., 10 training mazes).

C.2 Visualization of learned PC procedure

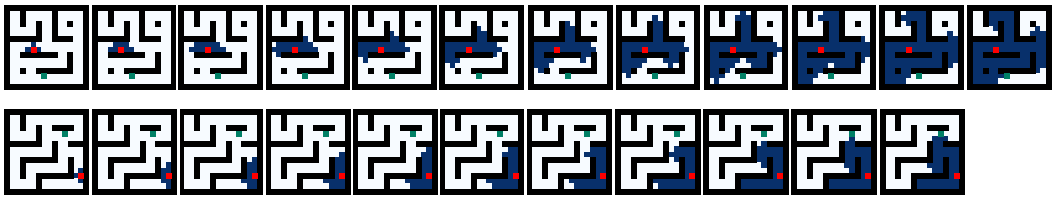

Figure 14: Example visualizations of PC's learned BFS procedure on discrete maze environment. PC learned to conduct BFS from the goal (red) backwards to the agent location (green). For the ease of visualization, the exact actions taken by the learned BFS traversal are eliminated from the blue cells.

C.3 Zero-shot generalization to greater distribution shift

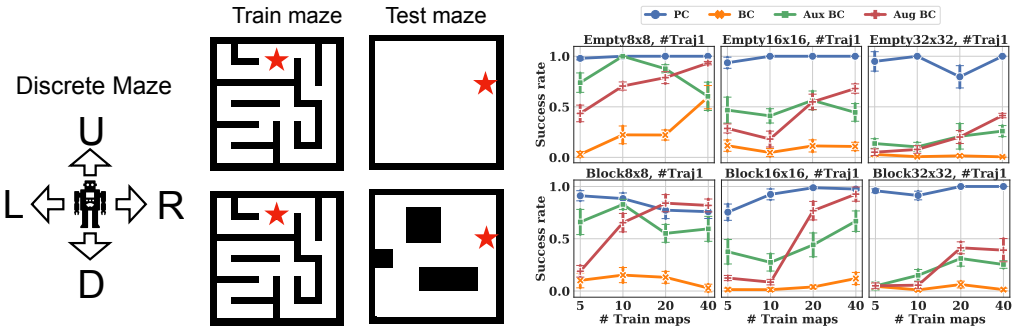

Figure 15: [Left] Visualization of training and test mazes. Training mazes have tunnel-shaped inner walls whereas test mazes have no inner walls or block-shaped inner walls. [Right] Average success rate of PC and BC agents navigating to the goal from random start locations over 10 test mazes. Agents are trained on 5, 10, 20, 40 mazes of 1 trajectory each. We find that procedure cloning leads to much better test maze generalization compared to alternative approaches when the test mazes exhibit drastic distribution shift from the training mazes.

D Additional related work

D.1 Neural program induction

While neural program induction (NPI) may be interpreted to apply to any "program" similar to PC, its demonstrated applications (both in [89] and [90]) are based on problems that exhibit hierarchical and modular solutions with shared and repeated subprograms (e.g., a "pick" primitive within a block-stacking task). Thus, NPI not only advocates to imitate the full program but also parameterize the agent in a modular way — like a program with function calls and return values — to take advantage of this modular and repeated structure. The main argument in these existing papers hinges on the modular decomposition of both task and agent, which allows for more efficient data sharing, especially in multiple-task settings. In contrast, our PC evaluations are on environments with much less modular structures, and we avoid using specialized agent parameterizations in favor of more generic architectures (e.g., transformers). Thus, we believe our work is complementary to NPI, showing that the paradigm of using procedural information – while first proposed by the NPI work to some extent – applies to much more general settings than initially suggested by NPI.

D.2 Goal-conditional imitation learning.

While some of our experiments may be interpreted as utilizing "goals" as intermediate computation, there is a key difference from goal-conditioned imitation learning [91, 92]. Namely, goal-conditioned imitation learning advocates for using portions of the observation (or learned functions of the observation) as goals, whereas PC uses information beyond what is available in the observation (e.g., coordinate positions of objects). Thus, the common argument in goal-conditioned imitation centers around getting more learning signal (goal-reaching) from the same data (s,a,s' tuples). In contrast, PC shows that having richer data in the demonstrations (procedure information) is useful, *even if that data is not available to the agent during inference*. We also note that another common argument made in goal-conditioned imitation learning is attributing their sample efficiency benefits to reduced temporal frequency induced by the hierarchical design, whereas PC shows benefits without any change in temporal frequency. Thus, we believe that PC presents a novel approach.

D.3 Learning-to-plan methods

Learning-to-plan methods (e.g., Universal Planning Networks [93], Value Iteration Networks [81]) are distinct from PC in that they train policies end-to-end using state-action tuples. No additional supervision of the intermediate computations of the expert is used; rather, these methods effectively propose a different policy parameterization leveraging inductive biases (i.e., with an embedded end-to-end differentiable planner). As shown in our evaluations of Implicit BC (Figure 5) and Value Iteration Networks (Appendix B.1), flexible policy parametrizations and architectures with desired inductive bias still fail to generalize without proper integration of procedure information during training, showing that PC provides benefits orthogonal to these existing works.