Chain-of-Thought Predictive Control with Behavior Cloning

Anonymous Authors [1]

Abstract

We study how to learn generalizable policies from demonstrations for complex continuous space tasks (e.g., low-level object manipulations). We aim to leverage the applicability & scalability of Behavior Cloning (BC) combined with the planning capabilities & generalizability of Model Predictive Control (MPC), and at the same time, overcome the challenges of BC with sub-optimal demos and enable planning-based control over a much longer horizon. Specifically, we utilize hierarchical structures in object manipulation tasks via key states that mark the boundary between sub-stages of a trajectory. We couple key state (the chain-of-thought) and action predictions during both training and evaluation stages, providing the model with a structured peek into the long-term future to dynamically adjust its plan. Our method resembles a closed-loop control design and we call it Chain-of-Thought Predictive Control (CoTPC). We empirically find key states governed by learnable patterns shared across demos, and thus CoTPC eases optimization of BC and produces policies much more generalizable than existing methods on four challenging object manipulation tasks in ManiSkill2.

1. Introduction

Behavior cloning (BC) remains a powerful approach to training autonomous agents for its simplicity, applicability, and scalability. Without requiring densely labelled rewards or a massive amount of on-policy interactions with the environments, BC casts policy learning as a supervised learning problem over a demonstration set. How well the learned policy generalizes to states unseen in the demos determines whether an agent can succeed, as the mistakes made by the model accumulate along the trajectories, known as the compounding errors (Ross & Bagnell, 2010). In particular, complex long-horizon tasks that operate in a low-level continuous action space expect BC agents to acquire generalizable and robust behaviors. For instance, inserting a peg into a hole requires both planning over a long trajectory and precise actuator controls.

Despite the recent progress relevant to BC (Chen et al., 2021; Florence et al., 2022; Shafiullah et al., 2022; Liu et al., 2022; Ajay et al., 2022), it remains extremely challenging to solve contact-rich object manipulation tasks by imitating the behaviors from demonstrations. Generated by human, RL agents, or planners (often sampling-based), expert demos for such tasks with a continuous action space often come with undesired properties such as inconsistency, high-entropy, and discontinuity, because of the underlying contact dynamics (Pfrommer et al., 2021) and/or how the demos are generated. Specifically, *inconsistency* reflects an intrinsic limitation of Markovian policies when dealing with conflicting sub-optimal demonstrations that lead to a negative synergy for vanilla BC. *High-entropy* refers to when the underlying demonstrator produces actions in a diverse manner that lead to high uncertainty and variance for the BC agents. *Discontinuity* describes the situation where very similar states from the demos give dissimilar actions, a sharp change of values that suggests discontinuity of the underlying mapping. These conditions create barriers to successful agents learned by behavior cloning from both the optimization and the generalization perspective, which will be discussed in detail in Sec. 4.1.

Another challenge that looms large for BC agents when dealing with object manipulation tasks is the *randomness* in the demos, i.e., the actual or apparent unpredictability of common planning algorithms when finding a successful trajectory. On one hand, luck matters more or less with the sampling nature of the demonstrators. For instance, in a continuous action space maze, a solution found by random search is a winning lottery ticket, whose pattern is unclear, hard to model and not very generalizable. On the other hand, with the internal mechanism (e.g., how BFS is performed) of the demonstrators hidden from the BC agents, even a deterministic demonstrator appears to admit randomness in its behaviors. In an effort to handle a similar issue, Procedure Cloning (Yang et al., 2022) was recently proposed to use intermediate computation outputs of demonstrators as additional supervision for improving the generalization of BC policies. However, it assumes knowledge of intermediate

[1]Anonymous Authors. Correspondence to: * <*>.

This is a preprint version under review.

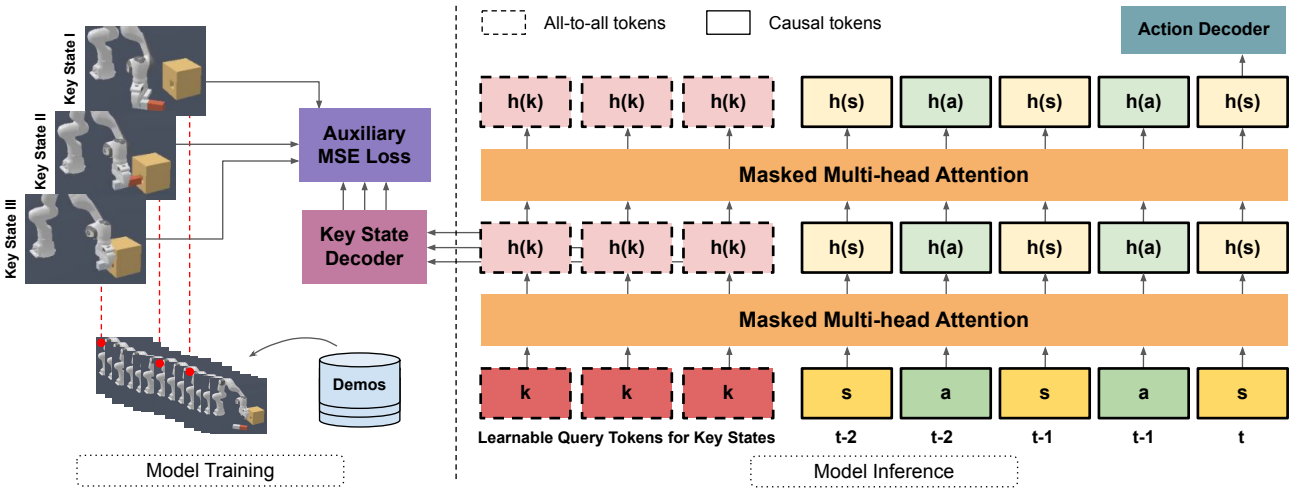

Figure 1. Illustration of our method. During training, we extract key states from demos and adopt an auxiliary MSE loss (BC loss omitted here). During inference, our model predicts key states (implicitly) and actions jointly at each time step, resembling a closed-loop control system. The key state tokens are all-to-all (can see any tokens up to the current step). The state and action tokens are causal (can only see previous tokens and the key state tokens). Only 2 attention layers and 3 inference steps are shown for better display.

computations, which might not be available and might be hardly learnable (e.g., a random search process) by neural networks.

To deal with these issues, we propose to leverage the hierarchical structures inherently presented in object manipulation tasks. We observe that certain key states in demos naturally mark the boundary of adjacent sub-tasks. They can be defined by simple rules (or conditions) shared across different demo trajectories. To complete a task, the agent has to reach states satisfying these conditions in a relatively fixed order and fashion. For example, to accomplish a peg insertion task, an agent has to first grasp the peg in a balanced manner, adjust its angle to make it aligned with the hole, and finally push it precisely. Accordingly, from each demo, three key states can be extracted: the first state when the peg is grasped, the first when it is aligned, and the finishing state with the peg inserted. In general, we can leverage human language to automatically generate rules, e.g., by asking ChatGPT "steps to insert a peg in a hole". Labeling the key states from the generated rules is trivial given the common access to privileged information in simulators.

More concretely, we propose to learn key state predictions as an auxiliary training loss for BC, which provides structured predictions of the long-term future. While the overall trajectories vary a lot (the aforementioned issues of demos), the key states (chain-of-thought) themselves admit fewer variations and share a common pattern useful for generalizable policy learning. From the optimization perspective, if the policy predicts actions conditioned on the chain-of-thought, the undesired properties of the demos become less concerning due to reduced variations of the inputs and the chain itself. At test time when no key states are given, our proposed model predicts key states, the chain-of-thought, implicitly, and generates reliable actions jointly with the dynamically updated chain at each time step. We integrate the key state predictor and action predictor gracefully in a shared Transformer backbone so that all predictions are conditioned on the trajectory history. From the generalization perspective, to deal with the randomness issue presented in demos, we leverage the impressive reasoning capabilities of Transformers (Brown et al., 2020) and labelled hierarchical information (i.e., the key states) to learn a policy with implicit long-horizon planning abilities that achieves generalizable decision-making. Note that, in contrast, Model Predictive Control is limited to short-term planning. Our overall design resembles a closed-loop control system and we call it **Chain-of-Thought Predictive Control (CoTPC)**. We compare CoTPC with strong baselines on four challenging object manipulation tasks from ManiSkill2 (Gu et al., 2023). CoTPC outperforms existing ones by a significant margin, especially for peg insertion where all baselines struggle.

2. Related Work

Learning from Demonstrations (LfD) Learning interactive agents from pre-collected demonstrations has been popular for policy learning due to its effectiveness and scalability. Roughly speaking, there are three categories: offline RL, online RL with auxiliary demos, and behavior cloning (BC). While offline RL approaches (Kumar et al., 2019; Fu et al., 2020; Levine et al., 2020; Kumar et al., 2020; Kostrikov et al., 2021; Chen et al., 2021; Wang et al., 2022) usually require demonstration with densely labelled rewards and methods that augment online RL with demos (Hester

et al., 2018; Kang et al., 2018; Ross et al., 2011; Nair et al., 2020; Rajeswaran et al., 2017; Ho & Ermon, 2016; Pertsch et al., 2021; Singh et al., 2020) rely on on-policy interactions, BC (Pomerleau, 1988) formulates fully supervised or self-supervised learning problems. With its simplicity, zero sample complexity, and better practicality, BC has been extensively used in real-world environments, especially in robotics (Zeng et al., 2021; Florence et al., 2022; Qin et al., 2022; Zhang et al., 2018; Brohan et al., 2022; Rahmatizadeh et al., 2018; Florence et al., 2019; Zeng et al., 2020).

Challenges in BC Despite the wide applications of BC, a well-known shortcoming is the compounding error (Ross et al., 2011). One source of this is the distribution shift (i.e., covariate shift) between the demo data and the ones unrolled at test time. Various methods were proposed to tackle it under different BC setups (Ross & Bagnell, 2010; Ross et al., 2011; Sun et al., 2017; Laskey et al., 2017; Tennenholtz et al., 2021; Brantley et al., 2019; Chang et al., 2021). Another related source is that there are certain properties of the collected demonstrations that make it challenging to clone a decent policy in the first place. Inconsistency, high-entropy, discontinuity (Florence et al., 2022), randomness and noisiness or sub-optimality (Sasaki & Yamashina, 2020; Wu et al., 2019) of either the demos or the underlying target policy lead to difficulties in policy representation and optimization by neural networks and results in great compounding errors during inference. See Sec. 4.1 for detailed discussions. In this paper, we tackle challenging object manipulation tasks whose demons suffer from these issues.

LfD as Sequence Modeling A recent research trend in CB, as well as in offline RL, is to relax the Markovian assumption of policies in the Markovian Decision Process (Howard, 1960) setup. With the widespread success of various sequence modeling models (Graves & Graves, 2012; Chung et al., 2014; Vaswani et al., 2017), model expressiveness and capacity are preferred over algorithmic sophistication. Among these, (Dasari & Gupta, 2021; Mandi et al., 2021) study one-shot imitation learning, (Lynch et al., 2020; Singh et al., 2020) explore behavior priors from demos, (Chen et al., 2021; Liu et al., 2022; Janner et al., 2021; Shafiullah et al., 2022; Ajay et al., 2022; Janner et al., 2022) examine different sequence modeling strategies for policy learning. In particular, methods based on Transformers (Vaswani et al., 2017; Brown et al., 2020) are extremely popular due to their simplicity and effectiveness. Through extensive comparison, we show the major advantages of our method compared to existing approaches.

Chain-of-Thought in Sequence Modeling Chain-of-Thought (Wei et al., 2022) refers to the general strategy of solving multi-step problems by decomposing them into a sequence of intermediate steps. It has recently been applied extensively in a variety of problems such as mathematical reasoning (Ling et al., 2017; Cobbe et al., 2021), program execution (Reed & De Freitas, 2015; Nye et al., 2021), commonsense or general reasoning (Rajani et al., 2019; Clark et al., 2020; Liang et al., 2021; Wei et al., 2022), and robotics (Xu et al., 2018; Zhang & Chai, 2021; Jia et al., 2022b; Gu et al., 2022; Yang et al., 2022). While inspired by the idea, we focus on BC for contact-rich object manipulation tasks (Mu et al., 2021; Gu et al., 2023). Our method leverages hierarchical structures inherent in robotic tasks and is generally applicable without requiring repeated action primitives (Zhang & Chai, 2021) or knowing the demonstrator's internal mechanism (Yang et al., 2022). Moreover, ours learns to predict the chain-of-thought along with actions dynamically.

Demonstrations for Robotics Tasks In practice, the optimality assumption of the demos is usually violated especially for robotics tasks. Demos for tasks involving low-level actuator actions primarily come in three forms: human demo captured via teleoperation (Kumar & Todorov, 2015; Vuong et al., 2021), expert demo generated by trained RL agents (Mu et al., 2021; Chen et al., 2022; Jia et al., 2022a), or trajectories found by task and motion planning (TAMP) algorithms (Gu et al., 2023; Qureshi et al., 2019) ((Fishman et al., 2022) learns from MP demos but focuses on short-horizon tasks without contact changes). These demos are in general sub-optimal due to either human bias, imperfect RL agents, or the nature of TAMP solvers. In our experiments, we use TAMP demos provided by ManiSkill2 (Gu et al., 2023). We illustrate several undesired properties of demonstrations in Sec. 4.1 that lead to difficulties in learning a generalizable policy via BC.

3. Preliminaries

MDP Formulation One of the most common ways to formulate a sequential decision-making problem is via a Markov Decision Process, or MDP (Howard, 1960), defined as a 6-tuple $\langle S, A, \mathcal{T}, \mathcal{R}, \rho_0, \gamma \rangle$, with a state space S, an action space A, a Markovian transition probability $\mathcal{T} : S \times A \to \Delta(S)$, a reward function $\mathcal{R} : S \times A \to \mathbb{R}$, an initial state distribution ρ_0, and a discount factor $\gamma \in [0, 1]$. An agent interacts with the environment characterized by $\mathcal{T}$ and $\mathcal{R}$ according to a policy $\pi : S \to \Delta(A)$. We denote a trajectory as τ_π as a sequence of $(s_0, a_0, s_1, a_0, ..., s_t, a_t)$ by taking actions according to a policy π. At each time step, the agent receives a reward signal $r_t \sim \mathcal{R}(s_t, a_t)$. The distribution of trajectories induced by π is denoted as $P(\tau_\pi)$. The goal is to find the optimal policy π^* that maximizes the expected return $\mathbb{E}_{\tau\sim\pi}[\sum_t \gamma^t r_t]$. Notice that, in robotics tasks and many real-world applications, the reward is at the best only sparsely given (e.g., a binary success signal) or given only after the trajectory ends.

Behavior Cloning BC assumes access to pre-collected demos $D = \{(s_t, a_t)\}_{t=1}^N$ generated by expert policies and learns the optimal policy with direct supervision by minimizing the BC loss $\mathbb{E}_{(s,a)\sim D}[-\log \pi(a|s)]$ w.r.t. a mapping π. It requires the learned policy to generalize to states unseen in the demos since the distribution $P(\tau_\pi)$ will be different from the demo one $P(\tau_D)$ at test time, a challenge known as distribution shift (Ross & Bagnell, 2010). Recently, several methods, particularly those based on Transformers, are proposed to relax the Markovian assumption. Instead of $\pi(a_t|s_t)$, the policy represents $\pi(a_t|s_{t-1}, s_{t-2}, ..., s_{t-T})$ or $\pi(a_t|s_{t-1}, a_{t-1}, ..., s_{t-T}, a_{t-T})$, i.e., considers the history up to a context size T. This change was empirically shown to be advantageous (also justified in Sec. 4.1).

4. Chain-of-Thought Predictive Control

We will illustrate the challenges in BC and introduce Chain-of-Thought Predictive Control (CoTPC) and its design choices.

4.1. Challenges of BC Illustrated with An Example

Given a set of demos $D = \{\tau_i = [(s_t, a_t)]_t\}_i$ for a task T, we learn a policy π without on-policy interactions to maximize the success rate of completing T by reducing compounding errors, a well-known issue for BC (Ross et al., 2011). For contact-rich object manipulations, it can be difficult to use neural networks to represent generalizable policies induced by the demos due to their undesired properties (high-entropy, discontinuity, etc.) or those of the underlying tasks (Pfrommer et al., 2021), especially with demos generated by planning algorithms (which are commonly used for their effectiveness and scalability). Here we use an example to illustrate the issues.

We consider a continuous maze illustrated in Fig. 2, where each node represents a small Euclidean ball and each edge a thin road connecting them. The agent starts at the leftmost ball (a grey star) to reach the rightmost ball (a black star). At each step, it moves between two balls directly connected by a (dashed) line. While not exactly a discrete graph, the structure here exemplifies how common task and motion planners (Garrett et al., 2021) find valid paths. The demo policy can be characterized by a distribution of all paths (i.e., the demos) connecting the start and the end within a maximum distance or number of steps.

Inconsistency - a Limitation of Markovian BC Due to the MDP formulation, the most commonly used policies for BC are Markovian ones, $\pi(a_t|s_t)$, that ignore the history. However, without the optimality assumption of the demos (which is commonly violated as in demos generated by planners), there is a fundamental limitation for BC with Markovian policies. While each path in the demos can be represented by a Markovian policy, the policy (also Markovian) linearly combined from them by imitating the combined demos can perform much worse than each individual one in the presence of conflicts across demos (negative synergy). We call this the inconsistency issue of demonstrations. In the left sub-graph of Fig. 2, two demos (marked in red and blue) travel clockwise and counter-clockwise, respectively. They share roads marked in mixed red-and-blue (with cyan markers) where opposite directions are given as guidance. A perfectly cloned Markovian policy, in this case, will perform random walks on the road, underperforming policies cloned from an individual demo. Instead, a non-Markovian policy can resolve inconsistency by including history as context to distinguish between different demos.

High-entropy - a Source of Compounding Errors Sometimes the history alone is not enough for the model to distinguish between different demos. In this case, the demo policy is said to have high entropy regarding its action distributions. For instance, in the middle sub-graph of Fig. 2, the roads in mixed red-and-blue constitute a shared history of four diverging (cyan arrows) demos marked by different colors. The high entropy from the demos, visualized around the cyan ball, leads to increased uncertainty and variance of cloned policies, making the BC agents more prone to compounding errors. Demos by planning algorithms, in particular, suffer from this issue. To the extreme, the action can be uniformly distributed. Note that multi-modality is a related but orthogonal issue, which (Shafiullah et al., 2022) focuses on, where a unimodal estimate of the (continuous) action distribution leads to significantly worse return.

Discontinuity - an Obstacle to Policy Learning For contact-rich manipulation tasks, expert policies often consist of sharp value changes or topology changes (e.g., due to contact changes). In the right sub-graph of Fig. 2, the demos come close to each other in the cyan circle while having distinct action outputs. Different from the inconsistency issue, here the two demos are not in conflict but with a sharp change of the underlying policy along the cyan line. Such discontinuity leads to difficulties in learning a robust and accurate model, thus harming generalizability. A recent method (Florence et al., 2022) deals with this by an energy-based implicit model in place of an explicit one. While theoretically sound, it is shown (Shafiullah et al., 2022) to be less practical for non-Markovian implicit models, which are outperformed by well-designed explicit ones.

Randomness - the Devil in Generalization In this continuous maze, the paths found by planning algorithms admit an apparent or actual lack of shared patterns, either due to the sampling nature or the fact that the internal mechanism is not revealed in the paths. Such a trait makes behavior cloning hard and less generalizable over unseen states. See

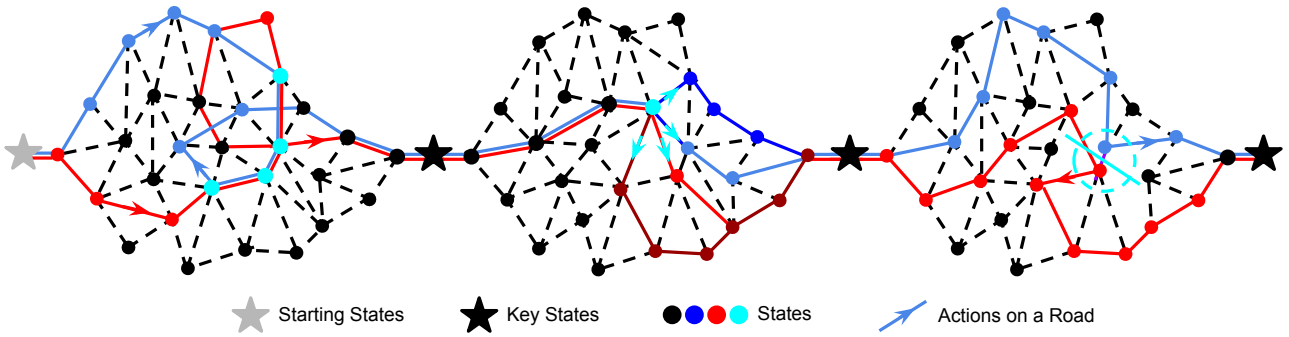

Figure 2. An example to illustrate the challenges of BC (inconsistency, high-entropy and discontinuity) for complex continuous space tasks (e.g., contact-rich object manipulations). It represents a continuous maze with paths (demonstrations) found by task and motion planners. The agent starts from the left and needs to reach the right. Each node (including stars) represents a Euclidean ball and each edge is a thin road (not a line). The sub-graphs resemble the hierarchical structures common in manipulation tasks. See details in Sec. 4.1.

a relevant argument in (Paster et al., 2022).

4.2. Our Approach: Dynamic Key State Prediction

To deal with the aforementioned issues of BC, we propose to leverage the structural guidance from a sequence of key states (the chain-of-thought) extracted from demonstrations. Many object manipulation tasks can be naturally decomposed into sub-tasks. The key states mark the boundary of adjacent sub-tasks (the black stars in Fig. 2), thus can give the model a peek into the succinct but long-term future to guide its behavior. Meanwhile, key states represent a global structure of a decision-making process and will benefit from the awareness of global policy progress. Thus, we propose to couple the learning of chain-of-thought as well as behavior cloning in a simple, unified and effective framework, which we will describe in detail as follows.

Model Architecture We use transformer (Brown et al., 2020) as the backbone and incorporate key state predictions by inserting K learnable query tokens at the beginning of the trajectory, where K is the number of key states (task-specific, e.g., 3 for peg insertion). Instead of using a causal attention mask, we design a *hybrid* masking regime in our model. Normal state or action tokens can attend to those in the past (standard causal mask) and so *always* see the key state tokens for action predictions (achieved with an action decoder). The key state query tokens, however, are all-to-all and can observe *all* action and state tokens occurring until the current time step t.

Training Objective The overall pipeline is illustrated in Fig. 1. The model is trained with behavior cloning loss as well as auxiliary key state prediction loss l_{ks}(with coefficient λ), which yields a simple objective: $l_{total} = l_{BC} + \lambda l_{ks}$.

Dynamic Key State Prediction At test time, our model *jointly generates actions and key states at each step* by conditioning on the trajectory history. The key states are *dynamically* updated during interactions, resembling a closed-loop control system. With this design, our model actively predicts and adjusts the key states and actions altogether for generalizable decision-making.

Coupled CoT and Behavior Modeling From an optimization perspective, we find that compared to the demos' overall high variance (and the aforementioned issues), the key states (chain-of-thought) admit fewer variations with a generalizable pattern shared across demos. Hence, action predictions conditioned on key states ease model optimization with reduced variations of inputs and consistent future cues. From a generalization perspective, the randomness issue in the demos is alleviated by both the impressive sequential reasoning capability of Transformers (Brown et al., 2020) and our design to jointly predict key states (chain-of-thought) and actions at each time step (reminiscent of Model Predictive Control). Intuitively speaking, the model adjusts its actions and its long-term forecast of the future given what has happened in the past, an implicit long-horizon planning capability that leads to consistent and generalizable decision-making patterns.

4.3. Key State Selection

As discussed previously, the underlying principle for choosing key states is to find a shared common pattern across demos that can improve the optimization and generalization of BC. Specifically, for each trajectory $\tau \in D$, we aim to find an ordered set of states $\{s_t | s_t \in \tau\}$ satisfying:

- The reaching of key states should indicate satisfying the necessary conditions for completing the task.
- The key states should admit fewer variations than the

average of other states.

- Functional correspondence (with a fixed order) exists between key states of different trajectories.

We find that locating key states with these conditions is easy for object manipulation tasks, as they are often inherently hierarchical with multiple stages. Roughly speaking, all trajectories for a contact-rich manipulation task share the same sequence of stages with a fixed order. And the boundaries of adjacent stages in a trajectory constitute the key states, which are critical for task completion and are reached in a very similar fashion among different demos. For example, in peg insertion, the robotic gripper goes through the reaching stage to grasp the peg (the first key state occurs when it first grasps the peg), then the adjusting stage that ends up aligning the peg with the hole (another key state), and eventually the pushing stage, the end of which (the last key state) marks task completion. We derive heuristic rules to automatically recognize such key states from demos. Most commonly used simulators (e.g., SAPIEN (Xiang et al., 2020)) provide access to privileged info. such as contact feedback and object poses, making implementing such rules rather trivial. Moreover, we further leverage human language to automatically generate the rules. For all four tasks in our experiments, we ask ChatGPT to produce structured answers. For instance, for peg-in-hole, we ask "what are the major steps to insert a peg into a hole?" and create rules based on the responses (see details in Appendix).

In our experiments and ablation studies, we empirically justify our principles of key states selection and compare the performance of (1) not using any key states, (2) using fewer states, (3) using more key states, and (4) using an alternative set of rules for selecting key states.

4.4. Key State Prediction

At test time, our model predicts key states (chain-of-thought) along with actions. Here we will discuss several relevant design options. Please also refer to the ablation studies.

Static vs. Dynamic Key State Prediction Static prediction refers to predicting the key states only once at the very beginning of each trajectory. Dynamic prediction, on the other hand, keeps updating its key state predictions along the way to task completion. The difference between the two is somewhat similar to open-loop vs closed-loop in motor controls, with dynamic predictions resembling the latter (also see Sec. 4.2). We find that the dynamic version generalizes significantly better as the model adjusts its prediction of both key states and actions on-the-fly.

Contextual Key State Prediction For a traditional Markovian policy, dynamic prediction means predicting the key states at each step by conditioning only the current state. For a non-Markovian policy that considers the context history (up to a maximum number of steps) for action prediction, we can utilize the context to predict key states as well. We call this contextual prediction where the history of states and actions is used. We find this strategy outperforms the alternative by a large margin. When combined with Transformer-based policies, contextual prediction and dynamic prediction can be achieved gracefully, as described in Sec. 4.2 and illustrated in Fig. 2.

Implicit vs. Explicit Key State Prediction We further observe that instead of predicting key states explicitly and predicting actions conditioned on them, it is better to do so implicitly. Specifically, an MLP (key state decoder) takes hidden-layer features from the Transformer to predict key states, which is trained with the auxiliary MSE loss (see Fig. 2). This encourages better representation that captures the hierarchical structure of the demos (so that it can predict key states). The alternative is to directly use the hidden features (or their linear projection) as the predictions.

4.5. Details for Training and Implementation

We adopt the minGPT implementation and use the same set of hyperparameters for all tasks (a feature embedding size of 128 and 4 masked multi-head attention layers, each with 8 attention heads, totalling around 1M learnable parameters). We use no positional embeddings for key state tokens and use learnable positional embedding for the state and action tokens similar to DT (Chen et al., 2021). The action decoder and key state decoder are both 2-layer MLPs. We use a coefficient $\lambda = 0.1$ for the auxiliary MSE loss. While any hidden layer can be utilized for key state predictions, we find the resulting performance varies (some can lead to training instabilities) and only use the first hidden layer for all tasks (making it task-specific does increase the performance, though). During training, for efficiency, we apply random masking to the action and state tokens so that the key states tokens attend to a history of varied length (from the first step to some t in the context history). At test time, while the key state decoder is unused, the action predictor attends to all tokens in the past (including the key states tokens) and the key state tokens (as is all-to-all) attend to all tokens up to the current time t. See more implementation details in Appendix.

5. Experiments

In this section, we first introduce the tasks and the metrics used to evaluate our method, discuss the baselines we compare with, and then present the main results. Furthermore, we show ablation studies that justify our design choices.

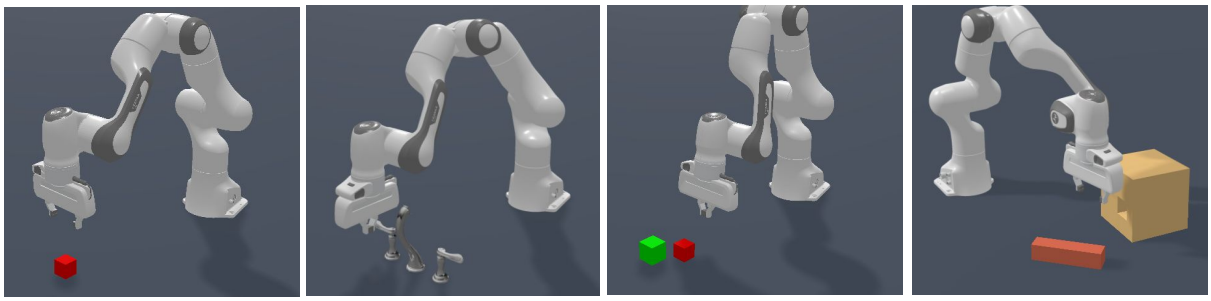

Figure 3. We use four challenging contact-rich object manipulation tasks from ManiSkill2 (Gu et al., 2023) to evaluate our method. Namely, `PickCube`, `TurnFaucet`, `StackCube` and `PegInsertionSide` (with a roughly increasing level of difficulty). We use state-space observations in our experiments while the images are rendered for illustration (goal location not visualized for `PickCube`).

Table 1. Numerical results for four tasks in ManiSkill2. We compare our method with four baselines for (or can be adapted to) BC, namely vanilla BC (Pomerleau, 1988), Decision Transformer (DT) (Chen et al., 2021), Behavior Transformer (BeT) (Shafiullah et al., 2022) and MaskDP (Liu et al., 2022). `SR` denotes the final success rate, while `SR*` denote intermediate ones. See the Appendix for a detailed description of these. All metrics are reported in percentage (%) with the best ones **bolded**.

METHOD	PICK CUBE		TURN FAUCET		STACK CUBE			PEG INSERTION			MEAN
	SR1	SR	SR1	SR	SR1	SR2	SR	SR1	SR2	SR	SR (%)
VANILLA BC	57.0	0.0	32.4	0.0	24.4	0.0	0.0	59.6	0.0	0.0	0.0
DECISION TRANSFORMER	81.2	53.6	58.2	25.4	**88.0**	18.8	8.2	73.8	11.4	0.2	21.9
BEHAVIOR TRANSFORMER	81.6	23.6	27.4	3.6	72.8	6.0	1.6	90.0	17.0	0.8	7.4
MASKEDDP + GT KEY STATES	80.2	54.7	53.1	28.2	78.2	15.7	7.8	62.6	5.8	0.0	22.7
COTPC (OURS)	**86.7**	**86.0**	**77.2**	**49.2**	86.8	**64.4**	**46.4**	**99.0**	**88.4**	**40.6**	**55.6**

Table 2. Results for ablation studies. See Sec. 5.3 for details.

	STACK CUBE			PEG INSERTION		
	SR1	SR2	SR	SR1	SR2	SR
PRED LAST STATE ONLY	53.0	20.6	12.0	96.8	65.6	9.6
PRED ALL STATES	80.4	43.4	30.0	92.4	70.2	18.0
PRED ALT KEY STATES	56.8	28.0	17.0	80.0	44.8	5.4
PRED W/ MLP	82.8	34.4	22.2	78.8	41.4	6.4
PRED EXPLICIT	65.6	38.6	21.6	97.4	51.6	11.2
PRED W/ ALT DECODER	72.4	49.2	33.0	96.0	65.8	12.8
COTPC	86.8	64.4	46.4	99.0	88.4	40.6

5.1. Tasks, Environments, Demonstration Data and Evaluation Protocols

We focus on low-level long-horizon object manipulation tasks, which are challenging for their high-dimensional continuous action space and their requirements of both multi-stage planning and high-precision action executions. We choose ManiSkill2 (Gu et al., 2023), an extension of ManiSkill (Mu et al., 2021) as the testbed, which features a variety of object manipulation tasks in environments with fully dynamic physical simulation. We choose four tasks with a (roughly) increasing level of difficulty (see Fig. 3). Namely, `PickCube` which asks the agent to pick up a cube and place it at a specified goal location; `TurnFaucet` for turning on faucets of various geometries and topology; `StackCube` for picking up a cube and placing it on top of another; and lastly, `PegInsertionSide` for inserting a cuboid-shaped peg into a hole. The `PegInsertionSide` is claimed to be particularly challenging with 3 millimeters of clearance and stable insertion required to complete the task, which is harder than that in other benchmarks.

To study how to improve BC, we choose to use state space observation (ranging from 43-d to 55-d for the four tasks) to avoid additional challenges of visual-based policy learning that might be confounding. We use an 8-d continuous action space for a delta joint position-based control, which is standard in many low-level object manipulation tasks. This action space is known to be challenging for learning behaviors with neural networks (Ajay et al., 2022; Florence et al., 2022). The initial object poses and target locations vary across different environment seeds for all four tasks. The geometries, sizes, or topology of objects also vary for `TurnFaucet` and `PegInsertionSide`. Each task has a maximum number of steps allowed (around 200). We use the provided demos by ManiSkill2, which are generated by TAMP solvers with the help of privileged information as sequences of state-action pairs without reward labels. We randomly sample 500 trajectories for each task.

For each task, we train agents using all methods being com-

pared on the same 500 demo trajectories. Each of the trajectories is generated with a different environment seed (variation). At test time, we evaluate all methods in the simulated environment using the same set of seeds used to generate the demos. Without allowing on-policy interactions during training, the learned policies have to generalize reasonably well over unseen states to have a decent test performance. We use success rate (SR) as the major metric. With the properties described in Sec. 4.1 of the demonstrations, we find that even achieving a non-zero `SR` is highly non-trivial for existing methods. We, therefore, also provide intermediate success rates (`SR1` and `SR2`) as additional metrics. See the Appendix for detailed descriptions of these metrics.

5.2. Baselines and Main Results

We first train a vanilla BC policy with a three-layer MLP and find that it performs poorly on all tasks due to the limitation of the Markovian assumption (see Sec. 4.1 for an explanation). We then compare our method with several popular non-Markovian baselines that achieve state-of-the-art performance. Namely, Decision Transformer (DT) (Chen et al., 2021), Behavior Transformer (BeT) (Shafiullah et al., 2022) and MaskDP (Liu et al., 2022). DT is originally proposed for offline RL that applies sequence modeling to demonstrations with densely labeled rewards. We adapt DT for the BC setup by ignoring the reward tokens. We implement our proposed CoTPC on top of DT by sharing the same configurations for the Transformer backbone. BeT is proposed to handle multi-modalities issues in BC and is claimed to be able to model arbitrary multi-modal state-action distributions via an action space discretization strategy. As multi-modalities is a relevant yet orthogonal issue, we do not find it to work well in the four tasks (more challenging than the ones used in their original paper), potentially because of the lack of high precision due to its discretization process. MaskDP, on the other hand, leverages masked auto-encoding (He et al., 2022), a bi-directional sequence modeling technique to improve the generalization of BC. It features action predictions conditioned on goal states (relevant to key states), with a limitation that it requires ground truth key states with their precise time steps provided to the model at test time. Ours, instead, predict actions and key states jointly. We find that MaskDP with GT key states outperformed by ours by a significant margin. Results are reported in Tab. 1 which shows the clear advantages of CoTPC over baseline methods. More details can be found in the Appendix.

5.3. Ablation Studies

We perform two sets of ablation studies to justify our design choices, with results summarized in Tab. 2.

Different Rules for Key States We perform ablation studies on the key state selection strategy. In the first variant, we only include the last key state (i.e., the very last state of each demo trajectory) in the auxiliary training loss, denoted as "Pred last state only". In the second variant, we include states other than our selected key states. In fact, we ask the model to predict all states given the previous state-action pairs and the current state, similar to learning the dynamic model (see implementation details in the Appendix). The extra states in this variant lead to noises during training as most states admit large variances; the resulting model generalizes worse since the guidance from key states is weakened. We denote this variant as "Pred all states". The last variant uses an alternative set of rules to select the key states, which produce states not at the boundaries between stages but rather in the middle of each stage, which are usually of high variations. We denote this as "Pred alt key states".

Different Key State Prediction Strategies We first justify the use of contextual and dynamic key state predictions. As discussed in Sec. 4.2, we find that with a Transformer architecture, the contextual (predicting conditioned on the history) and dynamic (predicting on-the-fly when carrying out the predicted actions) prediction strategies can be gracefully combined. Alternatively, we use an MLP to predict the key states given only the first state in the context history as inputs. This variant, denoted as "Pred w/ MLP", is semi-dynamic (and so semi-open loop) as it is unaware of the states and actions occurring after the first in the context buffer. In the next variant, we empirically show the advantage of implicit prediction (discussed in Sec. 4.4). We do not use the key state decoder and train the auxiliary MSE loss between hidden features (after a linear projection) corresponding to the key state tokens and the ground truth key states. In another variant, we use a three-layer MLP in place of a two-layer one for the key state decoder. We denote these two variants as "Pred explicit" and "Pred w/ alt decoder", respectively. See further implementation details for these ablation studies in the Appendix.

6. Conclusions and Future Work

In this work, we propose CoTPC for learning generalizable policies from demos by combining the benefits of BC and MPC while avoiding their shortcomings. CoTPC leverages hierarchical structures in object manipulation tasks and, at each step, predicts actions jointly with the key states that feature structured knowledge of the long-term future. With the guidance of generalizable patterns in the key states across different demos, CoTPC produces generalizable policies that significantly outperform existing methods on challenging contact-rich object manipulation tasks. We are currently performing evaluations on real robots to further justify our approach. We believe that CoTPC can be extended by scaling it up with multi-task demos, adapting it for visual inputs, and empowering it with online fine-tuning.

References

Ajay, A., Du, Y., Gupta, A., Tenenbaum, J., Jaakkola, T., and Agrawal, P. Is conditional generative modeling all you need for decision-making? *arXiv preprint arXiv:2211.15657*, 2022.

Brantley, K., Sun, W., and Henaff, M. Disagreement-regularized imitation learning. In *International Conference on Learning Representations*, 2019.

Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Dabis, J., Finn, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Hsu, J., et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chang, J., Uehara, M., Sreenivas, D., Kidambi, R., and Sun, W. Mitigating covariate shift in imitation learning via offline data with partial coverage. *Advances in Neural Information Processing Systems*, 34:965–979, 2021.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Chen, T., Xu, J., and Agrawal, P. A system for general in-hand object re-orientation. In *Conference on Robot Learning*, pp. 297–307. PMLR, 2022.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Clark, P., Tafjord, O., and Richardson, K. Transformers as soft reasoners over language. *arXiv preprint arXiv:2002.05867*, 2020.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Dasari, S. and Gupta, A. Transformers for one-shot visual imitation. In *Conference on Robot Learning*, pp. 2071–2084. PMLR, 2021.

Fishman, A., Murali, A., Eppner, C., Peele, B., Boots, B., and Fox, D. Motion policy networks. *arXiv preprint arXiv:2210.12209*, 2022.

Florence, P., Manuelli, L., and Tedrake, R. Self-supervised correspondence in visuomotor policy learning. *IEEE Robotics and Automation Letters*, 5(2):492–499, 2019.

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

Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

Garrett, C. R., Chitnis, R., Holladay, R., Kim, B., Silver, T., Kaelbling, L. P., and Lozano-Pérez, T. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

Graves, A. and Graves, A. Long short-term memory. *Supervised sequence labelling with recurrent neural networks*, pp. 37–45, 2012.

Gu, J., Chaplot, D. S., Su, H., and Malik, J. Multi-skill mobile manipulation for object rearrangement. *arXiv preprint arXiv:2209.02778*, 2022.

Gu, J., Xiang, F., Li, X., Ling, Z., Liu, X., Mu, T., Tang, Y., Tao, S., Wei, X., Yao, Y., Yuan, X., Xie, P., Huang, Z., Chen, R., and Su, H. Maniskill2: A unified benchmark for generalizable manipulation skills. In *International Conference on Learning Representations*, 2023.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16000–16009, 2022.

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., et al. Deep q-learning from demonstrations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Ho, J. and Ermon, S. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

Howard, R. A. Dynamic programming and markov processes. 1960.

Janner, M., Li, Q., and Levine, S. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34: 1273–1286, 2021.

Janner, M., Du, Y., Tenenbaum, J. B., and Levine, S. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

Jia, Z., Li, X., Ling, Z., Liu, S., Wu, Y., and Su, H. Improving policy optimization with generalist-specialist learning. In *International Conference on Machine Learning*, pp. 10104–10119. PMLR, 2022a.

Jia, Z., Lin, K., Zhao, Y., Gao, Q., Thattai, G., and Sukhatme, G. Learning to act with affordance-aware multimodal neural slam. *arXiv preprint arXiv:2201.09862*, 2022b.

Kang, B., Jie, Z., and Feng, J. Policy optimization with demonstrations. In *International conference on machine learning*, pp. 2469–2478. PMLR, 2018.

Kostrikov, I., Fergus, R., Tompson, J., and Nachum, O. Offline reinforcement learning with fisher divergence critic regularization. In *International Conference on Machine Learning*, pp. 5774–5783. PMLR, 2021.

Kumar, A., Fu, J., Soh, M., Tucker, G., and Levine, S. Stabilizing off-policy q-learning via bootstrapping error reduction. *Advances in Neural Information Processing Systems*, 32, 2019.

Kumar, A., Zhou, A., Tucker, G., and Levine, S. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33: 1179–1191, 2020.

Kumar, V. and Todorov, E. Mujoco haptix: A virtual reality system for hand manipulation. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pp. 657–663. IEEE, 2015.

Laskey, M., Lee, J., Fox, R., Dragan, A., and Goldberg, K. Dart: Noise injection for robust imitation learning. In *Conference on robot learning*, pp. 143–156. PMLR, 2017.

Levine, S., Kumar, A., Tucker, G., and Fu, J. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

Liang, Z., Bethard, S., and Surdeanu, M. 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*, 2021.

Ling, W., Yogatama, D., Dyer, C., and Blunsom, P. Program induction by rationale generation: Learning to solve and explain algebraic word problems. *arXiv preprint arXiv:1705.04146*, 2017.

Liu, F., Liu, H., Grover, A., and Abbeel, P. Masked autoencoding for scalable and generalizable decision making. *arXiv preprint arXiv:2211.12740*, 2022.

Lynch, C., Khansari, M., Xiao, T., Kumar, V., Tompson, J., Levine, S., and Sermanet, P. Learning latent plans from play. In *Conference on robot learning*, pp. 1113–1132. PMLR, 2020.

Mandi, Z., Liu, F., Lee, K., and Abbeel, P. Towards more generalizable one-shot visual imitation learning. *arXiv preprint arXiv:2110.13423*, 2021.

Mu, T., Ling, Z., Xiang, F., Yang, D., Li, X., Tao, S., Huang, Z., Jia, Z., and Su, H. Maniskill: Generalizable manipulation skill benchmark with large-scale demonstrations. *arXiv preprint arXiv:2107.14483*, 2021.

Nair, A., Dalal, M., Gupta, A., and Levine, S. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

Nye, M., Andreassen, A. J., Gur-Ari, G., Michalewski, H., Austin, J., Bieber, D., Dohan, D., Lewkowycz, A., Bosma, M., Luan, D., et al. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

Paster, K., McIlraith, S., and Ba, J. You can't count on luck: Why decision transformers fail in stochastic environments. *arXiv preprint arXiv:2205.15967*, 2022.

Pertsch, K., Lee, Y., and Lim, J. Accelerating reinforcement learning with learned skill priors. In *Conference on robot learning*, pp. 188–204. PMLR, 2021.

Pfrommer, S., Halm, M., and Posa, M. Contactnets: Learning discontinuous contact dynamics with smooth, implicit representations. In *Conference on Robot Learning*, pp. 2279–2291. PMLR, 2021.

Pomerleau, D. A. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

Qin, Y., Su, H., and Wang, X. From one hand to multiple hands: Imitation learning for dexterous manipulation from single-camera teleoperation. *arXiv preprint arXiv:2204.12490*, 2022.

Qureshi, A. H., Simeonov, A., Bency, M. J., and Yip, M. C. Motion planning networks. In *2019 International Conference on Robotics and Automation (ICRA)*, pp. 2118–2124. IEEE, 2019.

Rahmatizadeh, R., Abolghasemi, P., Bölöni, L., and Levine, S. Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 3758–3765. IEEE, 2018.

Rajani, N. F., McCann, B., Xiong, C., and Socher, R. Explain yourself! leveraging language models for commonsense reasoning. *arXiv preprint arXiv:1906.02361*, 2019.

Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

Reed, S. and De Freitas, N. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

Ross, S. and Bagnell, D. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 661–668. JMLR Workshop and Conference Proceedings, 2010.

Ross, S., Gordon, G., and Bagnell, D. 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*, pp. 627–635. JMLR Workshop and Conference Proceedings, 2011.

Sasaki, F. and Yamashina, R. Behavioral cloning from noisy demonstrations. In *International Conference on Learning Representations*, 2020.

Shafiullah, N. M. M., Cui, Z. J., Altanzaya, A., and Pinto, L. Behavior transformers: Cloning k modes with one stone. *arXiv preprint arXiv:2206.11251*, 2022.

Singh, A., Liu, H., Zhou, G., Yu, A., Rhinehart, N., and Levine, S. Parrot: Data-driven behavioral priors for reinforcement learning. *arXiv preprint arXiv:2011.10024*, 2020.

Sun, W., Venkatraman, A., Gordon, G. J., Boots, B., and Bagnell, J. A. Deeply aggrevated: Differentiable imitation learning for sequential prediction. In *International conference on machine learning*, pp. 3309–3318. PMLR, 2017.

Tennenholtz, G., Hallak, A., Dalal, G., Mannor, S., Chechik, G., and Shalit, U. On covariate shift of latent confounders in imitation and reinforcement learning. *arXiv preprint arXiv:2110.06539*, 2021.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Vuong, Q., Qin, Y., Guo, R., Wang, X., Su, H., and Christensen, H. Single rgb-d camera teleoperation for general robotic manipulation. *arXiv preprint arXiv:2106.14396*, 2021.

Wang, Z., Hunt, J. J., and Zhou, M. Diffusion policies as an expressive policy class for offline reinforcement learning. *arXiv preprint arXiv:2208.06193*, 2022.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Chi, E., Le, Q., and Zhou, D. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

Wu, Y.-H., Charoenphakdee, N., Bao, H., Tangkaratt, V., and Sugiyama, M. Imitation learning from imperfect demonstration. In *International Conference on Machine Learning*, pp. 6818–6827. PMLR, 2019.

Xiang, F., Qin, Y., Mo, K., Xia, Y., Zhu, H., Liu, F., Liu, M., Jiang, H., Yuan, Y., Wang, H., et al. Sapien: A simulated part-based interactive environment. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11097–11107, 2020.

Xu, D., Nair, S., Zhu, Y., Gao, J., Garg, A., Fei-Fei, L., and Savarese, S. Neural task programming: Learning to generalize across hierarchical tasks. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3795–3802. IEEE, 2018.

Yang, M., Schuurmans, D., Abbeel, P., and Nachum, O. Chain of thought imitation with procedure cloning. *arXiv preprint arXiv:2205.10816*, 2022.

Zeng, A., Song, S., Lee, J., Rodriguez, A., and Funkhouser, T. Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics*, 36(4): 1307–1319, 2020.

Zeng, A., Florence, P., Tompson, J., Welker, S., Chien, J., Attarian, M., Armstrong, T., Krasin, I., Duong, D., Sindhwani, V., et al. Transporter networks: Rearranging the visual world for robotic manipulation. In *Conference on Robot Learning*, pp. 726–747. PMLR, 2021.

Zhang, T., McCarthy, Z., Jow, O., Lee, D., Chen, X., Goldberg, K., and Abbeel, P. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5628–5635. IEEE, 2018.

Zhang, Y. and Chai, J. Hierarchical task learning from language instructions with unified transformers and self-monitoring. *arXiv preprint arXiv:2106.03427*, 2021.

A. Various Details for the Four Tasks Used In Our Experiments

Sampling details In our experiments, we use state space observation to avoid adding the additional challenges of visual-based policy learning that might be confounding to our studies of BC. The state observations for the four tasks have dimensions ranging from 43-d (`TurnFaucet`) to 55-d (`PegInserstionSide`). The original state observation dimension for `TurnFaucet` is 40. However, as there are different faucet models (with largely varied geometries) used across different environment variations, the original state space misses key information that can be used to distinguish between different faucet models. The visual observation space does not have this issue, though. We, therefore, append an extra 3-d vector for the pose of the faucet link by replaying the demos. As a result, the agent can learn to turn different faucet models jointly.

The demonstrations provided by ManiSkill2 (generated by TAMP solvers, see their original paper for details) contain 1000 trajectories for each of the four tasks except for `TurnFaucet`, where a varied number of trajectories are provided for different faucet models. We randomly choose 10 faucet models whose demos are generated by TAMP solvers. We then perform stratified sampling to choose 500 trajectories from `TurnFaucet` and 500 trajectories randomly sampled for the other three tasks. For both training and testing, we use the same set of demos (each trajectory has an average length of around 200). During testing, we set the maximum steps allowed as 150, 250, 200, 250 for `PickCube`, `TurnFaucet`, `StackCube` and `PegInsertionSide`, respectively.

To generate parsing rules to label key states for each demonstration, we ask ChatGPT "What are the major steps to XXX?", where XXX is the high-level description of the four tasks. We then write simple heuristics programs in the SAPIEN simulated environment (Xiang et al., 2020) (where ManiSkill2 is hosted) to automatically find the key states. Specifically, for `PickCube`, there are two key states: the first state when the cube is grasped by the robotic gripper and the finishing state (the last state) in a demo trajectory when the cube is placed in the target location. For `TurnFaucet`, there are two key states: the first state when the gripper has made a contact with the handle of the faucet and the finishing state (the last state) in a demo trajectory when the faucet is turned on. For `StackCube`, there are three key states: the first state the cube is grasped by the robotic gripper, the last state when the gripper is still grasping the cube while the cube is stacked on top of the other, and the finishing state (the last state) in a demo trajectory. Note that to complete the `StackCube` task, the gripper needs to leave both cubes aligned vertically and static. For `PegInsertionSide`, there are three key states: the first state when the peg is grasped, the first state the peg's angle is aligned with the hole, and the finishing state (the last state) in a demo trajectory where the peg is inserted (at least half of it) into the hole.

While the success conditions are well-defined by the authors of ManiSkill2 for the four tasks, we also derive intermediate success conditions to better evaluate the performance of different models. For `PickCube`, `SR1` is defined as whether at any time step, the cube has been grasped by the gripper successfully. For `TurnFaucet`, `SR1` is defined as whether at any time step, the faucet handle has been contacted by the gripper. For `StackCube`, `SR1` is defined as whether at any time step, the cube has been grasped by the gripper successfully, `SR2` is defined as whether at any time step, the cube has been stacked on top of the other cube. For `PegInsertionSide`, `SR1` is defined as whether at any time step, the peg has been grasped by the gripper successfully, `SR2` is defined as whether at any time step, the peg's angle has been adjusted to be aligned with the hole.

B. More Details for the Baselines and Our Proposed Method

For the MLP baseline (vanilla BC), it is a three-layer MLP with a hidden size of 256 and ReLU non-linearity. We train it with a constant learning rate of $1e-3$ with Adam optimizer with a batch size of 32 for 150K iterations (training longer leads to over-fitting even with tuned L2 regularization). For DT, we use the same In our proposed model, the action decoder and key state decoder are both 2-layer MLPs of one hidden layer of size 256 and with ReLU non-linearity. We train it with a constant learning rate of $5e-5$ (with a short warm-up period) with the Adam optimizer with a batch size of 256 for 1.5M iterations. We use a context size of 60 for all tasks. We find that our model, as well as DT that shares the same Transformer architecture and BC loss, can train much longer than others without worrying about overfitting. We use the exact same configurations for training and for the Transformer architecture (excluding the parts regarding key state tokens) in DT as for our method. For Behavior Transformer, we train it using the same configurations for architecture and for training as the one used for the Franka Kitchen task in the original paper. The Transformer backbone has approximately the same number of parameters (1M) as ours and DT. The changes are: we use 1024 bins for K-Means (we find that for our tasks, a smaller number of bins works worse); we use a context size of 10 (we find that for BeT, a longer context size leads to over-fitting, contrary to all other Transformer-based baselines). We train the model for around 50k iterations (we find that training longer leads to over-fitting easily for BeT, potentially because of its discretization strategy and the limited demos used for BC).

Some existing methods (Florence et al., 2022; Ajay et al., 2022; Yang et al., 2022) are not included because either they were compared unfavorably with the baselines or they assume access to information impractical in our tasks or their computation complexity is much higher. For each baseline (and our method), we report the evaluation results using the best checkpoint along the training (since we are evaluating the same set of environment variations used to generate the demo, this is essentially validation performance). For fairness, with each method, we save a model checkpoint only for every $T_{niter}//10$ iterations where T_{niter} is the total number of iterations for that model.

C. Details for the Ablation Studies

For the variant denoted as "Pred all states", we ask the model to predict all states given the previous state-action pairs and the current state, similar to learning the dynamic model. Specifically, in the Transformer backbone, we apply a next-state decoder (two-layer MLP similar to the action decoder) for each state token and train the model to predict the very next state (via an auxiliary MSE loss). In this variant, there are no query tokens for the key states as the state predictions are performed alternatively. In the variant denoted as "Pred w/ MLP", the key state decoder always takes the first state in the current context history as the only input and predicts the key states in a coupled manner. We use a three-layer MLP with a hidden size of 256. As the decoder is unaware of the states and actions occurring after the first in the context buffer (but the context buffer is changing as the predicted actions are carried out during inference), we consider this strategy as somewhere between a closed-loop and open-loop control system.

We will open-source our code soon.