Why Does Action Chunking Improve Behavioral Cloning Performance in Robotic Control?

Filippo Lazzati
Politecnico di Milano

Kyle Stachowicz
UC Berkeley

William Chen
UC Berkeley

Alberto Maria Metelli
Politecnico di Milano

Andrew Wagenmaker
UC Berkeley

Sergey Levine
UC Berkeley

Abstract: Action chunking—predicting and executing multiple actions instead of a single action—has proven to be a critical component for learning effective robotic control policies. However, our precise understanding of *why* action chunking improves performance has remained limited. In this work we seek to close this gap. Through rigorous experimental evaluations in both simulated and real-world settings, we show that existing hypotheses for the success of action chunking—temporal consistency, horizon reduction, and representation learning—fail to explain the success of action chunking. Instead, we find that action chunking benefits from greater non-Markovian expressivity and reduced compounding error compared to Markovian policies, but, in many settings of interest, these effects can be fully captured by *delayed* policies, which at each step predict a single action based on the observation k steps in the past. We then show that there exists an additional benefit of action chunking that we refer to as *implicit ensembling*. In particular, by learning a diversity of temporal relationships (that is, $a_t|o_t$, $a_t|o_{t-1}, \ldots$), action-chunked policies exhibit behavior matching that of a model ensemble, increasing their robustness and generalization ability over policies that only learn a *single* temporal relationship. Building on these insights, we show that in simulated and real-world robotic control settings, we can match the performance of action chunking without action chunking—by deploying an action chunking policy as an ensemble of policies with randomized delays. Furthermore, we propose a policy class that amplifies the benefits of action chunking by explicitly instantiating an ensemble, and which we show significantly improves over the performance of action chunking in many domains.

Website: https://action-chunking.github.io.

1 Introduction

Action chunking—predicting and executing a *sequence* of actions rather than a *single* action—is an essential ingredient in modern approaches to behavioral cloning for robotic control [1, 2]. Virtually all state-of-the-art policies for robotic control—from generalist policies capable of solving a wide range of tasks [3, 4], to single-task policies able to perform dexterous, high-precision maneuvers [5]—use some form of action chunking to achieve high performance.

In behavioral cloning, action chunking simply requires training a policy to predict k demonstrator actions from a single observation, modeling $(a_t, a_{t+1}, \ldots, a_{t+k-1}) \mid o_t$. Here the prediction target $\boldsymbol{a}_{t:t+k} = (a_t, a_{t+1}, \ldots, a_{t+k-1})$ is referred to as the *action chunk*. At inference time, action chunking policies execute all or part of this action chunk in an *open-loop* fashion, effectively relying on an action prediction from a "stale" observation o_t for multiple timesteps, rather than computing a fresh

Correspondence to: Filippo Lazzati {filippo.lazzati@polimi.it} and Andrew Wagenmaker {ajwagen@berkeley.edu}.

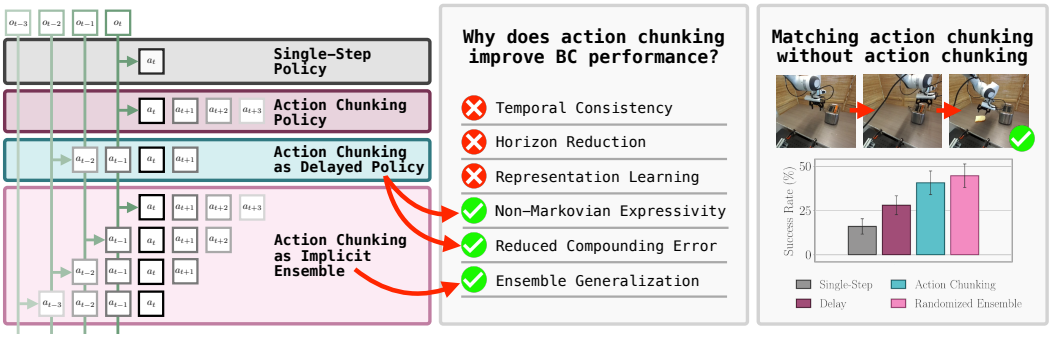


Figure 1: In this work we investigate why action chunking improves the performance of behavioral cloning in robotic control. We show that common hypothesis—temporal consistency, horizon reduction, and representation learning—are not able to explain the performance of action chunking. Instead, we demonstrate that a combination of three factors—expressing non-Markovian delays in human behavior, reducing compounding error by predicting actions based on past observations, and ensemble-like effects induced by learning a variety of temporal relationships—almost fully explains the success of action chunking. Motivated by our insights, we show that in both simulated and real-world robotic control, we can match the performance of action chunking without explicitly utilizing action chunking.

action at each timestep. Across a range of settings, executing action chunks in this manner achieves higher performance than recomputing and executing a single action at each timestep.

While empirically the necessity of action chunking is largely undisputed, our understanding of *why* action chunking is necessary remains rudimentary. Common hypothesis in the literature include:

1. **Temporal consistency**: Action-chunked policies can better represent the temporally correlated behaviors exhibited by human demonstrators [1, 2, 6].
2. **Horizon reduction**: Action chunks reduce the effective horizon of the environment, mitigating compounding error [1].
3. **Representation learning**: Action chunking serves as an auxiliary loss in policy training, improving representation learning and generalization [2, 7].

In this work we seek to develop a more complete understanding of the mechanisms enabling the success of action chunking in robotic control. Our first key observation is that, in many settings, the performance of action chunking can in fact be replicated by deploying a *delayed* policy $\pi(a_t \mid o_{t-k})$. Through theoretical analysis and controlled experiments on both simulated and real-world robotic manipulation settings, we show that, while humans do exhibit non-Markovian behaviors and action chunking does meaningfully reduce compounding error, predicting actions based on delayed observations is often sufficient to capture the necessary non-Markovian behaviors and achieve sufficient reductions in compounding error—the temporal consistency and horizon reduction provided by action chunking is not necessary for effective performance.

While non-Markovian expressivity and the reduction in compounding error provided by predicting actions based on past observations can partially explain the success of action chunking, we find settings where these effects are not fully explanatory. We identify an additional mechanism driving action chunking's improved performance, which we refer to as *implicit ensembling*: an action-chunked policy $\pi(\boldsymbol{a}_{t:t+k}|o_t)$ can be interpreted as an *ensemble* of k delayed policies $\{\pi(a_t|o_{t-i})\}_{0\leq i<k}$ and, by learning a diversity of temporal relationships, inherits many of the benefits of ensemble-based methods [8–10]. Across all simulated and real-world settings we consider, we show that we can match the performance of action chunking *without action chunking* by deploying the action-chunked policy as an ensemble of policies with random delay.

While the focus of this work is primarily analysis of existing approaches, our results also provide insight into how we can improve on the performance of action chunking. In particular, motivated by the implicit ensembling effect of action chunking, we show that by training an *explicit* ensemble of

delayed policies, we can exceed the performance of action chunking in many settings. Altogether, our results provide a much more complete picture for why action chunking enables improved performance in behavioral cloning for robotic control and show that, in many settings, the typical instantiation of action chunking is not necessary for effective policy performance.

Organization. The rest of this paper is organized as follows. In Section 2 we discuss related work, and in Section 3 outline our problem setting. Next, in Section 4 we investigate existing hypothesis for why action chunking improves performance—temporal consistency (Section 4.1), horizon reduction (Section 4.2), and representation learning (Section 4.3)—ultimately arguing that these are inadequate to explain the success of action chunking. In Section 5, we show that action chunking policies act as implicit ensembles, and that this effect is critical to their success. Section 6 then extends our simulated results to the real-world, demonstrating that our conclusions hold there as well. Motivated by our findings, in Section 7 we show that amplifying the implicit ensembling effects of action chunking with explicit ensembles can lead to even further performance improvements. Finally, in Section 8, we close by highlighting several interesting directions for future work.

2 Related Work

Behavioral cloning in robotics. Imitation learning via behavioral cloning (BC, [11])—where a policy is trained via supervised learning to mimic the demonstrator's actions—is widely used in robotics [12–17]. Recent work scales BC by collecting large-scale human demonstration datasets [18–20] and training expressive generative models on these datasets [1, 2, 5, 21–26], resulting in generalist vision-language-action (VLA) policies capable of performing a wide variety of tasks [27–34]. To the best of our knowledge, *nearly every large-scale robot policy training effort since ACT [1] predicts and executes chunked actions* [3, 4, 29, 35–39]. In contrast to these works, rather than simply demonstrating another example of applying BC to robotic control, the focus of this paper is in understanding why action chunking helps, and showing how we can move beyond action chunking. We remark as well that the ensembling approach we propose in Section 7 is somewhat related to the work of [40], although the RL finetuning objective considered there is tangential to our objective.

Action chunking. To the best of our knowledge, the first works to explicitly propose the use of action chunking in robotics in its current form are the concurrent works on ACT [1] and Diffusion Policy [2]. While many previous works also explored the idea of a policy *producing* multiple actions, typically these works learned receding-horizon policies [41], where only the first action is executed. Several works have sought to understand or improve on action chunking. In particular, several approaches have been proposed to adjust the policy's sampling process in order to maintain temporal consistency *across* action chunks [42–46]. Other works have sought to obtain the benefits of action chunking while ensuring policy reactivity by mixing action chunks of different length [47], training a policy via reinforcement learning to adaptively select the chunk length [48], or using a world model to adaptively switch between action chunks at test-time [49].

Perhaps most related to our work is the work of Simchowitz et al. [50] and Zhang et al. [51], which seek to obtain a theoretical understanding for why action chunking helps, largely from a control-theoretic perspective. This line of work primarily aims to understand the *minimal* conditions under which action chunking enables improved performance, showing that action chunking has benefits even if the demonstrator is deterministic and Markovian and the environment is fully observed. In particular, Zhang et al. [51] shows that, under certain control-theoretic notions of stability, action chunking can mitigate compounding error in continuous control domains. While this is related to our investigation of the compounding error hypothesis (see Section 4.2), we find that in practice a variety of other effects also contribute to the success of action chunking.

3 Preliminaries

In this work we study action chunking in the context of behavioral cloning. We consider interaction with a potentially non-Markovian, partially observed environment $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{O}, P, P_0, P_{\mathcal{O}}, r)$,

where $\mathcal{S}$, $\mathcal{A} := \mathbb{R}^{d_\mathcal{A}}$, and $\mathcal{O}$ are the state, action, and observation spaces, $P : \mathcal{H} \times \mathcal{A} \to \triangle_\mathcal{S}$ is the transition kernel for histories $\mathcal{H}$, $P_0 \in \triangle_\mathcal{S}$ is the initial state distribution, $P_\mathcal{O} : \mathcal{S} \to \triangle_\mathcal{O}$ is the observation distribution, and $r : \mathcal{O} \to [0, 1]$ the reward. An episode consists of a sequence of states, observations, and actions $(s_1, o_1, a_1, s_2, o_2, a_2, s_3, \dots)$ where $s_1 \sim P_0$, $s_{t+1} \sim P(\cdot \mid h_t, a_t)$, for h_t the history of states and actions up to t, and $o_t \sim P_\mathcal{O}(s_t)$. This continues for H steps or until a successful state is reached, i.e., $r(o_t) = 1$. We denote by $\mathcal{J}(\pi)$ the expected episode reward of policy π, where the expectation is over trajectories induced by π on $\mathcal{M}$ (for a 0-1 success reward, this corresponds to the probability that some successful state will be reached before step H). While we consider non-Markovian, partially observed environments for the sake of generality, the majority of our conclusions also hold in Markovian (i.e. environments where $P(\cdot \mid a_t, s_t, s_{t-1}, s_{t-2}, \dots) = P(\cdot \mid a_t, s_t)$) and fully observed settings.

We assume access to a dataset of demonstration trajectories $\mathfrak{D} = \{\tau_\mathtt{i}\}_{\mathtt{i}=1}^N$, with each trajectory $\tau_\mathtt{i} = (o_1^\mathtt{i}, a_1^\mathtt{i}, \dots, o_H^\mathtt{i}, a_H^\mathtt{i}, o_{H+1}^\mathtt{i})$ generated by some demonstrator π_demo on $\mathcal{M}$. The demonstrator may be non-Markovian, meaning that it may condition on the full history of observations. In general, we assume that $\mathfrak{D}$ consists of successful (although not necessarily optimal) behaviors, and are interested in learning policies that maximize success rate $\mathcal{J}(\pi)$. In the following, it will be convenient to split $\mathfrak{D}$ into train and validation sets, $\mathfrak{D}_\text{train}$ and $\mathfrak{D}_\text{val}$. Throughout this work, we denote $[k] := \{1, \dots, k\}$.

Behavioral cloning. Behavioral cloning is a standard approach to learning from demonstrations that trains a policy via supervised learning to mimic the actions present in $\mathfrak{D}_\text{train}$. In particular, in the simplest, Markovian case, behavioral cloning flattens the training dataset into $(o_t^\mathtt{i}, a_t^\mathtt{i})$ pairs and trains a policy $\widehat{\pi}(a_t \mid o_t)$ to predict the actions from the dataset:

$$\widehat{\pi} \leftarrow \arg\max_\pi \textstyle\sum_{(o_t, a_t) \in \mathfrak{D}_\text{train}} \log \pi(a_t \mid o_t). \tag{1}$$

At deployment, given observation o_t an action $a_t \sim \widehat{\pi}(\cdot \mid o_t)$ is sampled and executed in the environment. In practice, modern approaches to BC in robotics typically parameterize π with a generative model—usually either an autoregressive transformer [4, 30] or diffusion/flow model [2, 3]—and aim to model the full demonstrator distribution. While our results are independent of the exact instantiation of the BC policy, for all experiments we parameterize $\widehat{\pi}$ as a diffusion model [2].

Action chunking. Action-chunked policies are trained to predict the next k actions produced by the demonstrator instead of only the single next action. That is, action chunking models the conditional distribution of sequences of actions:

$$\widehat{\pi} \leftarrow \arg\min_\pi \textstyle\sum_{(o_t, \boldsymbol{a}_{t:t+k}) \in \mathfrak{D}_\text{train}} \log \pi(\boldsymbol{a}_{t:t+k} \mid o_t) \tag{2}$$

for $\boldsymbol{a}_{t:t+k} = (a_t, a_{t+1}, \dots, a_{t+k-1})$ the sequence of k actions produced from the demonstrator following each o_t in $\mathfrak{D}_\text{train}$ (the "action chunk"). At deployment, action-chunked policies typically sample $\boldsymbol{a}_{t:t+k} \sim \widehat{\pi}(\cdot \mid o_t)$ and execute either part or all of $\boldsymbol{a}_{t:t+k}$ open-loop before querying the policy for a new sequence of actions.

Policy notation. We let $\widehat{\pi}_k$ denote a policy that produces action chunks of length k. In practice, it is common to only execute a portion of an action chunk, e.g. the first $n < k$ steps of the action chunk, and then recompute a fresh action chunk. We denote a policy that is deployed in this fashion as $\widehat{\pi}_k^n$. We will also consider *single-step delayed policies*, where we compute a new action at each step, but condition on some observation d steps in the past. We denote such a policy as π_delay^d (so, for example, $a_t \sim \pi_\text{delay}^d(\cdot \mid o_{t-d})$). We can also execute an action-chunking policy $\widehat{\pi}_k$ as a delayed policy for delay $d < k$ by sampling $\boldsymbol{a}_{t-d:t-d+k} \sim \widehat{\pi}_k(\cdot \mid o_{t-d})$ and taking the $(d+1)$th action in $\boldsymbol{a}_{t-d:t-d+k}$. We denote this induced delayed policy as $\pi_\text{delay}^d[\widehat{\pi}_k]$. For brevity, we refer to single-step Markovian BC policies as "Markov" or "single-step", action chunking policies with chunk size k as AC(k), and delayed policies with delay d as Delay($d+1$).

Validation error. Throughout this work, we consider validation error—computed as mean-squared error (MSE) of *mean* actions sampled from the policy on $\mathfrak{D}_\text{val}$—to measure how well a policy fits the data. We do not use this as a proxy for task success, only for analysis purposes. Formally, for

an action-chunk k policy $\widehat{\pi}_k$, we define the validation error:

$$L_{\text{val}}(\widehat{\pi}_k) := \frac{1}{d_{\mathcal{A}} \cdot k} \cdot \frac{1}{|\mathfrak{D}_{\text{val}}|} \sum_{(o_t, \boldsymbol{a}_{t:t+k}) \in \mathfrak{D}_{\text{val}}} \|\mathbb{E}_{\boldsymbol{a} \sim \widehat{\pi}_k(\cdot|o_t)}[\boldsymbol{a}] - \boldsymbol{a}_{t:t+k}\|_2^2,$$

where, for action chunks, we take $\|\boldsymbol{a}\|_2^2 := \sum_i \|[\boldsymbol{a}]_i\|_2^2$ for $[\boldsymbol{a}]_i$ the ith element in the chunk. If $\widehat{\pi}$ is a delayed policy (i.e. π^d_{delay} or $\pi^d_{\text{delay}}[\widehat{\pi}_k]$) or an action-chunked policy where we only execute the first n actions in the chunk (i.e. $\widehat{\pi}^n_k$), we define $L_{\text{val}}(\widehat{\pi})$ analogously, but only compute the loss over the actions that the policy would execute ($a_{t+d} \mid o_t$ for a delayed policy, and $a_t, \ldots, a_{t+n-1} \mid o_t$ for $\widehat{\pi}^n_k$), in either case normalizing L_{val} by the number of actions the loss is computed over rather than $1/k$. We avoid the standard denoising validation loss due to ambiguity with ensembled policies (see Section A.1 for further discussion of this point). While this notion of error does not capture multi-modality, in practice diffusion policies rarely exhibit significant multi-modality [52] so this is not a major shortcoming (in particular, we found that the policies we trained in this work did not exhibit significant multi-modality, see e.g. Section B.1).

4 Do Existing Hypotheses Explain the Performance of Action Chunking?

In this section we seek to understand why BC policies trained to predict and execute action chunks outperform those trained to predict only single actions. We proceed by investigating each of the commonly cited hypotheses for the success of action chunking—**temporal consistency** (Section 4.1), **horizon reduction** (Section 4.2), and **representation learning** (Section 4.3).

To investigate these hypothesis, in this section we consider the `Libero` behavioral cloning benchmark [53] (in the following sections we also consider the `Robomimic` benchmark [54], as well as real-world robotic settings). `Libero` contains 130 total tasks, each provided with 50 successful human demonstrations—we primarily consider the `Libero-90` suite of 90 tasks. We train diffusion policies [2] on these demonstrations, using image-based observations. For each point in the following results, we average over at least three random seeds; error bars denote standard errors. Please see Section A for detailed experimental setup and Section E for results on individual tasks.

4.1 Hypothesis 1: Temporal Consistency

The temporal consistency hypothesis starts from the observation that human demonstrators may not be truly Markovian, and, in particular, often produce smoothly-varying, correlated action sequences that are better modeled by policies explicitly encoding temporal correlations than Markovian policies [1, 2, 4, 6]. Here we investigate this claim experimentally.

Experiment 1: Human demonstrations are better modeled by delayed policies. To test this hypothesis, we first consider whether non-Markovian policies are able to better model the demonstration data in `Libero-90` compared to single-step Markovian policies. In particular, we select $k = 20$ and train an action chunk 20 policy $\widehat{\pi}_{20}$ on $\mathfrak{D}_{\text{train}}$. We consider the action prediction MSE on $\mathfrak{D}_{\text{val}}$ for policy $\widehat{\pi}^n_{20}$ and $n \in [20]$, $L_{\text{val}}(\widehat{\pi}^n_{20})$—the average action prediction error over the first n steps in the action chunk. We also plot $L_{\text{val}}(\pi^n_{\text{delay}}[\widehat{\pi}_{20}])$—the action prediction error of the induced delayed policy. Conceptually, $L_{\text{val}}(\widehat{\pi}^n_{20})$ measures how effectively we can predict $(a_t, \ldots, a_{t+n-1}) \mid o_t$ while $L_{\text{val}}(\pi^n_{\text{delay}}[\widehat{\pi}_{20}])$ measures $a_{t+n} \mid o_t$ (equivalently, $a_t \mid o_{t-n}$). We compare both of these for different values of n to $L_{\text{val}}(\widehat{\pi}^1_k)$, the validation MSE of a Markovian policy.

Our results, aggregated across `Libero-90`, are given in Figure 2. We confirm that we can more easily predict a_t given o_{t-n} than given o_t, for all $n < 18$. Furthermore, we find that we can more easily predict an action based on a delayed observation than we can predict the full action chunk. Note that, if the demonstrator were purely Markovian, we would expect the mutual information between a_t and o_t to be at least as large as the mutual information between a_t and o_{t-n} (by the data processing inequality). However, we see the opposite: a_t is best predicted by conditioning on o_{t-10}. This suggests that human demonstrators exhibit significant non-Markovian behavior[1]. ♢

[1]Note that, in the case of a purely Markovian, fully-observed environment, if the demonstrator is non-Markovian, we can perfectly replicate their state distribution with a Markovian policy *so long as this policy*

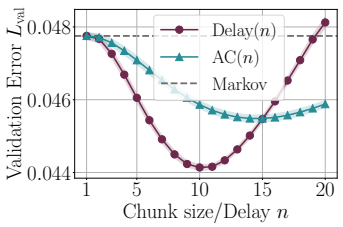


Figure 2: Action prediction validation error of delayed policies and action chunking policies, aggregated over `Libero-90`.

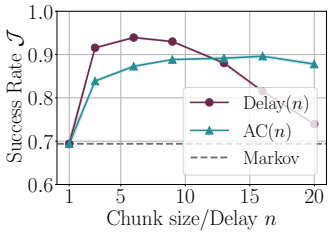


Figure 3: Average task success of delayed policies and action chunking policies, aggregated over `Libero-90`.

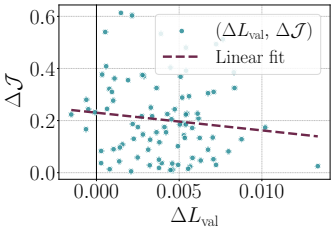


Figure 4: Success difference of AC(10) and Markovian policy, vs non-Markovian-ness, for each task in `Libero-90`.

Experiment 2: Delayed policies match the performance of action chunking. Notably, while Figure 2 shows that $a_t \mid o_{t-n}$ achieves lower validation error than $a_t \mid o_t$, it also shows that the validation error averaged over the entire prediction $\boldsymbol{a}_{t:t+n} \mid o_t$ is greater than $a_t \mid o_{t-n}$, for proper choice of n. To test how this impacts policy performance, we compare the success rate of $\widehat{\pi}^n_{20}$ to $\pi^n_{\text{delay}}[\widehat{\pi}_{20}]$ for various choices of n. We illustrate the performance of these approaches in Figure 3, averaging the success rate achieved by each policy over all `Libero-90` tasks. We see that simply executing a delayed policy matches or exceeds the performance achieved by an action-chunking policy. This suggests that, on `Libero-90`, delayed policies capture the non-Markovian demonstrator behaviors relevant for policy performance. Thus, while action chunking may model human non-Markovian behavior more effectively than Markovian policies, leading to more effective performance, the full non-Markovian expressivity of action-chunked policies is not required—it suffices to simply predict an action based on a delayed observation. Concretely, this suggests that in standard robotic manipulation settings (in particular in this case `Libero`), temporal consistency is not necessary to perform effectively, and the benefits of action chunking are not due to improving temporal consistency. In Section B.5, we show that this is true for VLAs as well—we find that the `Libero` finetune of $\pi_{0.5}$ [33] performs just as well when executed as a delayed policy as a standard action-chunked policy. ♢

Takeaway 1: Action chunking improves performance by capturing non-Markovianity in human demonstrators, but temporal consistency is not, in general, required

We conclude that humans do exhibit non-Markovian behavior, which can be more effectively modeled by an action-chunking policy than a Markovian policy. However, delayed policies model human demonstrators just as effectively, and perform equivalently to or better than action-chunked policies in terms of success rate, suggesting that the relevant non-Markovian behavior is captured by delayed policies, and temporal consistency is not required.

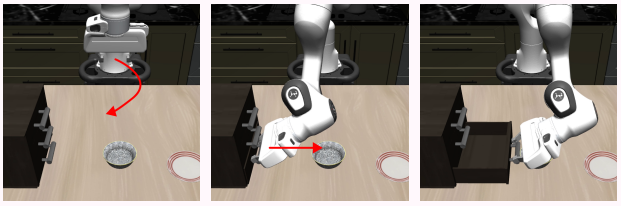
4.2 Hypothesis 2: Horizon Reduction

While humans exhibit non-Markovian behavior that action chunking is able to capture, our results suggest that delayed policies are able to capture the relevant non-Markovian behavior as well. Here we investigate if other effects also contribute to the performance of action chunking, and begin by investigating the explanatory power of non-Markovian expressivity.

Experiment 3: Non-Markovian expressivity only partially explains the success of action chunking. If non-Markovian expressivity fully explained the superior performance of action chunking, we would expect that in settings where action chunking does *not* provide additional expressivity benefits (for example, when the demonstrator is Markovian), Markovian policies would

is indexed on timestep t. However, if the policy is not indexed on timestep, as is usually the case in practice, then a Markovian policy does not suffice to model a non-Markovian demonstrator. In other words, even in the simplest settings, we require non-Markovian policies to model the behavior of a non-Markovian demonstrator. See Section C for further discussion of this point.

Example: When do non-Markovian behaviors arise in practice?



(a) $t = 0$. (b) $t = 80$. (c) $t = 110$.

Figure 5: Illustration of the task "open the bottom drawer of the cabinet." The panels show (a) the initial state, (b) the "decision boundary" at which the demonstrator pauses and changes direction, and (c) the nearly completed task.

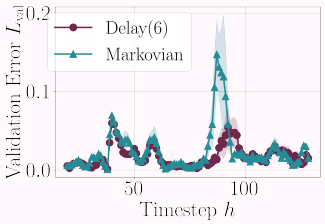


Figure 6: Action prediction validation error vs. episode timestep. The demonstrator pauses near timestep 80 while opening the drawer.

To illustrate where non-Markovian behavior can arise in practice, we consider Task 6 of `Libero-90`, "open the bottom drawer of the cabinet", which we illustrate in Figure 5. This task requires the robot to move down, grasp the bottom handle, and pull the drawer open. While this primarily involves straightforward free-space motion, at the point where the robot reaches the drawer it must both fasten onto the handle and change direction. We refer to this point as the *decision boundary*—in order to complete the task, the robot must "decide" at this point to transition from motion in one direction to motion in another direction.

We observe that the human demonstrations exhibit significant pauses at this decision boundary. Instead of immediately changing direction, the human demonstrator stops for several steps, perhaps to ensure the handle is effectively grasped, before proceeding. Such pauses are non-Markovian—while the state has not changed meaningfully, the demonstrator consistently waits for n steps. As such, a Markovian policy will not effectively fit this behavior—instead of predicting "wait" for n steps deterministically, it will predict "wait" with some probability and "move" with some probability (i.e. the marginal action distribution at the decision-boundary). In contrast, either an action-chunked or delayed policy *can* model this non-Markovian behavior, as long as the chunk size and delay are at least n. To illustrate this quantitatively, in Figure 6 we plot the action prediction error for a Markovian policy and a delayed policy along the held-out demonstration trajectory shown in Figure 5. We see that the Markovian and delayed policy have similar prediction error except around step 80—the precise point this "decision boundary" is encountered—when the Markovian policy has significantly higher prediction error.

In policy rollouts, we find that this behavior causes the Markovian policy to deviate from the demonstrator behavior at this decision boundary, leading to out-of-distribution states and poor performance, while delayed or action-chunked policies effectively model the human demonstrator at this state, enabling successful task completion. Such "decision boundaries" are very common across robotic manipulation tasks, and, as such, the non-Markovian expressivity of action chunk and delayed policies can enable significant improvements over Markovian policies by fully capturing these behaviors.

perform as well as action chunking policies. To investigate this, we evaluate the behavior of $\widehat{\pi}^{10}_{20}$ (executing action chunks of length 10) compared to $\widehat{\pi}^{1}_{20}$ (executing a Markovian policy) on each `Libero` task individually. We consider the metric $L_{\text{val}}(\widehat{\pi}^{1}_{20}) - \min_{n \geqslant 1} L_{\text{val}}(\pi^{n}_{\text{delay}}[\widehat{\pi}_{20}])$, which we denote ΔL_{val}, as a proxy for measuring how non-Markovian the demonstrator is, evaluated for each `Libero` task. $L_{\text{val}}(\widehat{\pi}^{1}_{20})$ corresponds to how well a Markovian policy can predict the demonstrator's behavior, while $\min_{n \geqslant 1} L_{\text{val}}(\pi^{n}_{\text{delay}}[\widehat{\pi}_{20}])$ corresponds to how well the *best* delayed policy can predict their behavior—we would therefore expect this value to be large only if non-Markovian policies are useful for predicting demonstrator behavior on a particular task. In Figure 4, we plot the success rate difference between $\widehat{\pi}^{10}_{20}$ and $\widehat{\pi}^{1}_{20}$ for each task in `Libero-90` (denoted as $\Delta\mathcal{J}$) against this measure of non-Markovian behavior.

If non-Markovian expressivity were predictive of the success of action chunking, we would expect there to be a strong correlation between ΔL_{val} and $\Delta \mathcal{J}$. Somewhat surprisingly, however, we find little correlation between how non-Markovian the demonstrator is for a given task and how much action chunking improves on a single-step policy. Indeed, in several tasks for which action chunking still improves significantly on the Markovian policy, there is little difference between L_{val} of a Markovian and non-Markovian policy—even though the demonstrator's behavior can be accurately modeled by a Markovian policy, non-Markovian policies can still lead to substantial performance improvements in terms of success rate[2]. $\Diamond$

These results suggest that non-Markovian expressivity only partially explains the success of action chunking, leading us to conclude that other effects must be present as well. A second hypothesis commonly proposed in the literature is that action chunking reduces the *effective horizon* of the environment, and as such reduces compounding error. Formally, the suboptimality of a BC policy $\widehat{\pi}$ typically scales with the horizon of the problem H as well as the supervised learning loss ϵ (i.e., the population validation error under the demonstrator's trajectory distribution) [55]: $\mathcal{J}(\pi_{\text{demo}}) - \mathcal{J}(\widehat{\pi}) \leqslant C(H) \cdot \epsilon$, where $C(H)$ denotes some measure of how quickly the error compounds in our environment (that is, how quickly supervised learning error leads to performance reduction). Though the precise scaling of $C(H)$ depends on environment properties, in general $C(H)$ increases with H. Thus, if we can decrease H—for example, to H/k—while, critically, keeping ϵ fixed, this should reduce the compounding error and total suboptimality. By only computing an action every k steps, action chunking *does* decrease the effective horizon in this fashion. Here we investigate whether this effect can explain the performance of action chunking.

As a starting point, we note that, as shown in Figure 3, delayed policies can match the performance of action chunking in aggregate across `Libero-90`. This immediately suggests that the performance of action chunking is not due, strictly speaking, to horizon reduction: delayed policies recompute actions at each step, so do not reduce the effective horizon as action chunking does, yet still perform comparably to action-chunked policies. However, for the tasks in Figure 4 where $\widehat{\pi}^{10}_{20}$ and $\widehat{\pi}^{1}_{20}$ achieve approximately the same validation loss (ϵ), several of these tasks exhibit significant gaps in performance between action-chunked and single-step policies. This suggests that, while horizon reduction may not be necessary, action chunking nonetheless mitigates compounding error: $C(H)$ is still lower for an action-chunked policy. To explain this, we investigate the effect of action chunked and delayed predictions theoretically.

Theory: Predicting demonstrator actions from past states provably reduces compounding error. Consider the setting with deterministic dynamics, so that $s_{t+1} = P(s_t, a_t)$, and P is 1-Lipschitz in both state and action. Assume that all policies considered and the reward r are 1-Lipschitz in the state (please see Section D for a precise statement of these assumptions). In this setting, we have the following lower bound on the performance of a Markovian policy.

Theorem 1. *There exists an environment satisfying the above assumptions, a Markov demonstrator π_{demo}, and a Markov policy $\widehat{\pi}$ satisfying $\max_t \mathbb{E}^{\pi_{\text{demo}}}[W_1(\pi_{\text{demo}}(s_t), \widehat{\pi}(s_t))] \leqslant \epsilon$, for $W_1(\cdot, \cdot)$ the Wasserstein-1 metric, such that $\mathcal{J}(\pi_{\text{demo}}) \geqslant \mathcal{J}(\widehat{\pi}) + \Omega(2^H \cdot \epsilon)$.*

Theorem 1 shows that, if $\widehat{\pi}$ satisfies a standard bound on supervised learning error, then this error can compound exponentially in horizon, as 2^H. Please see Section D for a proof of this result. We next show that this exponential dependence can be reduced by playing an action-chunked policy, but that it can *also* be reduced by simply playing a delayed policy.

Theorem 2. *Assume that for all $t \in [H]$ and some $n \geqslant 0$:*

$$\mathbb{E}^{\pi_{\text{demo}}}\big[\max_{i \in [n]} W_1(\pi_{\text{demo}}(s_{t+i-1}), [\widehat{\pi}_n(s_t)]_i)\big] \leqslant \epsilon. \tag{3}$$

[2]Note that, for fully expressing non-Markovian behaviors, a *history-conditioned* policy, e.g. modeling $a_t \mid o_t, o_{t-1}, o_{t-2}, \ldots$, may be necessary. In practice, however, the performance of history-conditioned policies is typically very poor, and we find that delayed policies modeling $a_t \mid o_{t-d}$ perform much better. We believe this is due to the increased sample complexity of history-conditioned policies: by conditioning on a full history, we increase the effective observation space *exponentially* with the length of the history, increasing the challenge of generalization in the low-data regime.

for $[\widehat{\pi}_n(s_t)]_i$ *the* i*th step in the action chunk. Then, for* $k < n$*:*

$$\mathcal{J}(\pi_{\text{demo}}) - \mathcal{J}(\widehat{\pi}_n^k) \leqslant \mathcal{O}((k+1)^{H/k} \cdot \epsilon) \quad \textit{and} \quad \mathcal{J}(\pi_{\text{demo}}) - \mathcal{J}(\pi_{\text{delay}}^k[\widehat{\pi}_n]) \leqslant \mathcal{O}((k+1)^{H/k} \cdot \epsilon).$$

Theorem 2 shows that, if the action-chunked policy is able to fit the demonstrator effectively over the entire chunk, then we can bound the compounding error by $\mathcal{O}((k+1)^{H/k})$[3]. In particular, if $k = c \cdot H$ for some constant fraction c, this scales polynomially in H rather than exponentially, offering an exponential improvement over a Markovian policy. Notably, however, the delayed policy achieves the exact same reduction in compounding error. Furthermore, in Section D we show that this is tight—action-chunked policies must incur compounding error that scales at least as $\Omega((k+1)^{H/k})$, showing that, in the setting of deterministic, smooth dynamics, action chunking does not exhibit any benefits over simply playing a delayed policy in terms of compounding error. ♢

These results show that, while action chunking can mitigate compounding error, the primary mechanism is simply because it conditions on previous observations, rather than because it reduces horizon. Intuitively, this arises because earlier states are likely to have compounded *less* error—they are more in-distribution—than later states, if the initial state distributions for train and test are identical. Thus, if we can effectively model the demonstrator's action at a future state given only a delayed observation, we would expect this prediction to be *more accurate* than if we attempt this inference conditioned on the current state. Notably, our experimental results in Figure 2 show that we *can* model the demonstrator's action at a future state effectively, suggesting that the key criteria of Theorem 2 hold in practice. While these results rely on the assumption that dynamics are smooth, we show in Section B.3 that this assumption holds in many robotic control settings of interest.

Takeaway 2: Action chunking reduces compounding error by predicting actions based on past observations

Action chunking reduces compounding error but, in many settings, this results not from horizon reduction, but from action-chunked policies predicting actions based on delayed observations. We find that, as a result, we can often replicate the success of action chunking simply by training a policy to predict actions based on a delayed observations.

4.3 Hypothesis 3: Representation Learning

The final hypothesis we evaluate is whether action chunking has representational benefits. A commonly-observed phenomenon is that training an action chunk k policy and only executing the first $n < k$ actions from the chunk still performs better than a Markovian policy, even for small n. This is commonly attributed to representational benefits of training with action chunking. Here we evaluate whether this effect can also contribute to the superior performance of action chunking.

Experiment 4: Delayed policies match the representational benefits of action chunking. To test this hypothesis, we compare the performance of $\widehat{\pi}_{20}^1$ to $\widehat{\pi}_1$—executing a single action at each step, but training on action chunks vs. single actions—and also compare this to $\pi_{\text{delay}}^5[\widehat{\pi}_{20}]$ and $\widehat{\pi}_{\text{delay}}^5$, where $\widehat{\pi}_{\text{delay}}^5$ denotes the policy trained directly to predict $a_t \mid o_{t-5}$ (while $\pi_{\text{delay}}^5[\widehat{\pi}_{20}]$ is the delay policy induced by $\widehat{\pi}_{20}$). We plot aggregate performance on `Libero-90` in Figure 7. We find that, while action chunking *does* provide representational benefits when compared to a single-step policy, it *does not* provide meaningful benefits when compared to training a delayed policy. ♢

Takeaway 3: Action chunking only gives representational benefits for short horizons

Action chunking provides representation-learning benefits only when predicting the first few steps in the action chunk, but not for predicting actions with more substantial delays. Thus, the benefits of action chunking can again be fully replicated with a delayed policy.

[3] We remark that, in the case of absolute position control where the dynamics can be approximately modeled by $s_{t+1} = P(s_t, a_t) \approx a_t$, rates on the order of $\Omega(H^2 \cdot \epsilon)$ and $\mathcal{O}(H^2/k \cdot \epsilon)$ can be obtained for, respectively, the Markovian learner and the action chunking/delayed learners, using similar proof techniques.

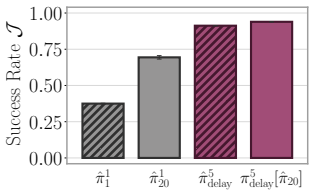


Figure 7: Success of Markovian and delayed policies trained with and without action chunks, aggregated over `Libero-90`.

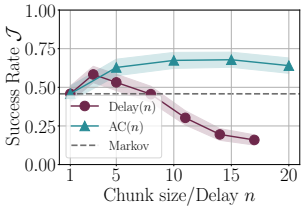


Figure 8: Average task success of delayed policies and action chunking policies, aggregated over all `Robomimic` tasks.

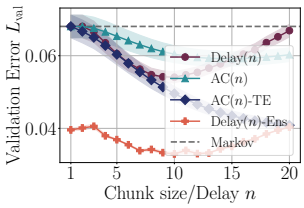


Figure 9: Validation error of Delay(n), AC(n), AC(n) temporal ensemble, Delay(n) ensemble, over `Robomimic` tasks.

5 Action-Chunked Policies as Implicit Ensembles

The preceding results suggest that non-Markovian expressivity and reduction in compounding error contribute to the success of action chunking but that, on `Libero`, these effects can be fully captured with only a delayed policy—action chunking is not required. We next seek to understand whether these results generalize beyond `Libero`, and consider an additional benchmark, the `Robomimic` benchmark [54]. In particular, we consider the four most challenging from `Robomimic`: `Can`, `Square`, `Transport`, and `Tool Hang`, and we report results training on the Proficient-Human demonstrations (please see Section B.4 for results on Multi-Human; our results hold identically there). In Figure 8, we compare the performance of action-chunked policies to that of delayed policies aggregated across these `Robomimic` tasks, and in columns 2 and 3 of Table 1 provide numerical success rates for the best-case action-chunked policy compared to the best-case delayed policy. We see that, while delayed policies do improve on single-step policies significantly in `Robomimic`, they fail to match the performance of action chunking in general, contrasting our results on `Libero`.

In this section we seek to offer an explanation for why this is the case. Our key insight is that, in addition to the aforementioned benefits of action chunking, action-chunked policies act as *implicit ensembles*. When training an action-chunked policy, for a given action a_t in our training data, we learn the relationships $a_t \mid o_t$, $a_t \mid o_{t-1}$, …, $a_t \mid o_{t-k+1}$. In other words, rather than just fitting a *single* $a_t \mid o_{t-d}$, we fit what, effectively, is an *ensemble* of predictors, each learning a different temporal relationship. By learning the relationships $a_t \mid o_t$, $a_t \mid o_{t-1}$, $a_t \mid o_{t-2}$, …, $a_t \mid o_{t-k+1}$, an action-chunked policy learns to predict demonstrator actions based on subsets of the true "feature" $(o_t, o_{t-1}, \ldots, o_1)$. Conceptually, this is often how ensemble-based methods such as random forests operate—aggregating predictors trained on different features—and it is known that such approaches can achieve lower generalization error than the average error across ensemble members, leading to improved performance [8–10]. We conjecture that, by aggregating action predictions over time through interaction with the environment, action chunking implicitly acts as an ensemble, inheriting such benefits. Here we investigate experimentally to what extent this holds in practice.

Experiment 5: Temporal ensembles enable more accurate action prediction than delayed policies. We first evaluate the validation error on `Robomimic`. We consider an identical setup to that considered in Figure 2, but add in two additional methods. First, we consider the temporal ensemble induced by $\hat{\pi}_k^n$. That is, for some sequence of observations $(o_1, \ldots, o_k)$, we compute $a^i \leftarrow [\hat{\pi}_k(o_{k-i+1})]_i$ for $i \in [n]$; thus, $\bar{a}^n \leftarrow \frac{1}{n}\sum_{i=1}^n a^i$, so that $\bar{a}^n$ is the prediction induced by combining the actions produced by the previous n predictions induced by $\hat{\pi}_k$, the *induced temporal ensemble* (which we refer to as AC(n)-TE)[4]. We also train an *actual ensemble*, training policies $\{\hat{\pi}_{20,i}\}_{i=1}^m$, where each $\hat{\pi}_{20,i}$ is trained on the same dataset $\mathfrak{D}_{\text{train}}$, but from different random initializations. To aggregate the ensemble predictions, we average the actions predicted by each ensemble

[4] We note that existing works such as ACT [1] propose similar temporal ensembles. The primary difference between our instantiation of a temporal ensemble and existing approaches is that we combine predictions linearly, while previous works apply an exponential weighting, significantly downweighting the contributions of more recent timesteps, and mitigating the ensembling effects.

Task	Markovian	AC(n)	Delay(n)	AC(n)-RDE	AC(n)-TE
`Libero-90`	68.9 ± 0.6	89.2 ± 0.2	**94.0** ± 0.1	93.6 ± 0.1	92.7 ± 0.1
`Libero-10`	19.8 ± 1.5	**88.7** ± 0.4	86.8 ± 0.4	**88.5** ± 0.4	86.0 ± 0.5
`Libero-Spatial`	58.1 ± 0.9	91.6 ± 0.4	**92.0** ± 0.5	**92.7** ± 0.3	91.1 ± 0.3
`Libero-Goal`	62.7 ± 1.0	**96.5** ± 0.2	**96.3** ± 0.3	**96.3** ± 0.3	**96.7** ± 0.2
`Libero-Object`	59.8 ± 2.3	**98.0** ± 0.4	**97.4** ± 0.3	**97.3** ± 0.5	**97.6** ± 0.4
`Robomimic Can PH`	83.7 ± 0.4	**97.2** ± 0.3	93.5 ± 0.4	96.7 ± 0.2	96.2 ± 0.3
`Robomimic Square PH`	69.0 ± 0.8	**85.4** ± 0.5	80.8 ± 0.6	82.4 ± 0.6	80.6 ± 0.5
`Robomimic Transport PH`	3.3 ± 0.3	**12.6** ± 0.5	7.9 ± 0.5	**12.1** ± 0.5	**12.2** ± 0.6
`Robomimic Tool Hang PH`	28.0 ± 0.8	**75.2** ± 0.5	51.6 ± 0.9	71.8 ± 0.8	42.2 ± 1.0

Table 1: Comparison of success rates across `Libero-90` and `Robomimic`. We see that action chunking significantly outperforms delayed policies in several settings, but that randomized delay ensembles perform comparably to action chunking in all cases.

member. In particular, here we only consider the delayed ensemble, that ensembles $\pi^n_{\text{delay}}[\widehat{\pi}_{20,i}]$ (which we denote as Delay$(n+1)$-Ens).

We plot our results in Figure 9. As can be seen, the temporal ensemble induced by the action-chunked policy achieves lower validation loss than that induced by the standard action-chunked policy, and nearly as low as that of the true ensemble. This suggests that the predictive properties of the ensemble induced by an action-chunked policy are indeed similar to that of a true ensemble, and significantly outperform a single delayed policy. We also note that, for the correct choice of delay, the delayed policy achieves lower validation loss than that of the action-chunked policy, suggesting that non-Markovian expressivity is not the key limitation of the delayed policies in `Robomimic`. Together, this suggests that the implicit ensemble induced by $\widehat{\pi}_{20}$ significantly improves on the predictive ability of any single delayed policy, and comes close to that of a true ensemble. $\Diamond$

Experiment 6: Randomized delay deployment of action chunk policies. If the cumulative benefits of action chunking are its ability to (a) express non-Markovian demonstrators (Section 4.1), (b) reduce compounding error by predicting on past observations (Section 4.2), and (c) instantiate an implicit ensemble, then the key shortcoming of $\pi^d_{\text{delay}}[\widehat{\pi}_{20}]$ is simply that it only leverages one of the temporal relationships learned by $\widehat{\pi}_{20}$ (in particular $a_t \mid o_{t-d}$), while typical deployment of action chunking leverages all learned relationships. Thus, if we could enable prediction based on past observations while leveraging all learned temporal relationships in a different manner than standard action chunking deployment, we would expect to achieve similar performance as action chunking.

To test this, we consider the following procedure. At each step t first sample $i \sim \text{unif}(\{0, 1, ..., n-1\})$, then sample $a_t \sim \pi^i_{\text{delay}}[\widehat{\pi}_k](o_{t-i})$. That is, we randomize over the delay of the possible delayed policies induced by $\widehat{\pi}^n_k$ at each step. We state the results for this approach in Table 1 (we denote this approach as the "randomized delay ensemble", or AC(n)-RDE). We see that this approach (nearly) matches or outperforms the performance of action chunking across all settings, *even when the delayed policies perform worse than the action-chunked policy*. Furthermore, the induced random delay ensemble also exceeds the performance of the temporal ensemble achieved by averaging the predictions (AC(n)-TE), suggesting that the ensemble randomization is critical to achieving effective performances. $\Diamond$

Takeaway 4: Action chunking implicitly instantiates an ensemble

Action chunking instantiates an implicit ensemble of delayed policies, learning all temporal relationships $a_t \mid o_{t-i+1}$ for $i \in [k]$. This provides significant "ensemble-like" benefits, improving action prediction ability and success rate. Combined with the other two hypotheses, this almost fully explains the benefits of action chunking across `Libero-90` and `Robomimic`.

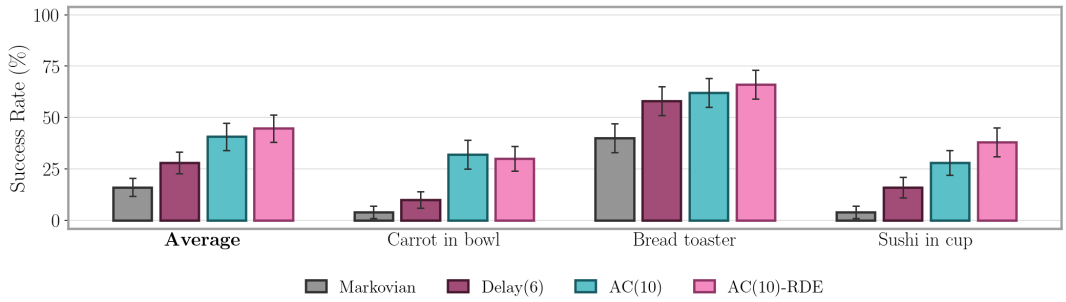
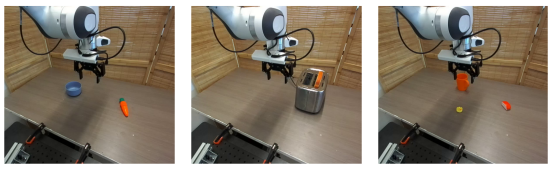


Figure 10: Comparison of success rates across the three real-world tasks considered. While action chunking significantly outperforms Markovian policies, delayed policies capture much of the performance of action chunking, and randomized delays (RDE) fully captures the performance of action chunking.

6 Randomized Delay Ensembles Match the Performance of Action Chunking in Real-World Robotic Control

We next test whether our conclusions hold in real-world robotic manipulation settings. We utilize the Franka Emika robot arm and consider three tasks, illustrated in Figures 11a-11c. Specifically, in the first task the goal is to pick up a carrot and put it in a bowl ("carrot in bowl"), in the second it is to remove the bread from the toaster and put it on the table ("bread toaster"), and in the final task we must pick up the sushi and put it in a cup ("sushi in cup"). We adopt delta joint position control and use control frequency 15Hz. See Appendix A.5 for more details on real-world experiments.

For each task, we collect 50 human demonstrations and train a diffusion policy with action chunk length $k = 20$ on the demonstrations, $\widehat{\pi}_{20}$. We consider several different sampling strategies. First, we deploy the policy as an action chunk 10, $\widehat{\pi}_{20}^{10}$, and action chunk 1 (Markovian) policy, $\widehat{\pi}_{20}^{1}$. Next, motivated by our previous results, we test the induced delay 5 policy, $\pi_{\text{delay}}^{5}[\widehat{\pi}_{20}]$ as well as the randomized delay ensemble policy introduced in the previous section, AC(10)-RDE. Unlike the simulated setting where pausing to compute the next action does not lead to changes in the environment, in the real world, pausing to compute the next action can lead to non-negligible changes in the state—in practice, the robot is not completely static even if no action is being commanded, so the state at the start and end of policy inference can be different, leading to poor policy performance. To mitigate the effect of this, we interleave policy inference and operation and start computing the action for the next step before the current action has finished executing. While this eliminates the need for pauses, it does introduce a small delay—in practice, then, all approaches here are run with an additional delay of 1 timestep.



(a) Carrot in bowl. (b) Bread toaster. (c) Sushi in cup.

Figure 11: Real-world evaluation tasks.

Our results are given in Figure 10, where each bar corresponds to the average success rate rolling out a policy 50 times. We see that the Markovian policies perform poorly and are significantly outperformed by action chunking policies. The delayed policy, while outperforming the Markovian policy, cannot, in general, match the performance of the action chunked policy. However, the randomized delay ensemble matches and even exceeds the performance of the action-chunking policy. These results perfectly replicate our simulated results. While simple delayed policies are able to capture non-Markovian behavior and mitigate compounding error in a way that enables significant improvements over Markovian policies, they do not capture the implicit ensembling effect of action chunking. However, by using the randomized delay ensemble induced by the action-chunked policy, we can match or exceed the performance of action chunking.

Task	Markovian	AC(n)	Delay(n)	AC(n)-Ens	Delay(n)-Ens	AC(n)-RDE-Ens
Libero-90	68.9 ± 0.6	89.2 ± 0.2	94.0 ± 0.1	94.1 ± 0.1	**95.0** ± 0.1	94.6 ± 0.0
Libero-10	19.8 ± 1.5	88.7 ± 0.4	86.8 ± 0.4	**90.6** ± 0.4	**90.2** ± 0.5	89.5 ± 0.5
Libero-Spatial	58.1 ± 0.9	91.6 ± 0.4	92.0 ± 0.5	**95.4** ± 0.4	**94.8** ± 0.4	**94.7** ± 0.6
Libero-Goal	62.7 ± 1.0	96.5 ± 0.2	96.3 ± 0.3	**98.8** ± 0.3	97.5 ± 0.5	97.4 ± 0.3
Libero-Object	59.8 ± 2.3	**98.0** ± 0.4	**97.4** ± 0.3	**98.8** ± 0.0	**98.2** ± 0.4	**98.3** ± 0.4
Robomimic Can PH	83.7 ± 0.4	97.2 ± 0.3	93.5 ± 0.4	**98.5** ± 0.1	97.7 ± 0.4	**98.7** ± 0.3
Robomimic Square PH	69.0 ± 0.8	85.4 ± 0.5	80.8 ± 0.6	**88.3** ± 0.3	**87.7** ± 1.1	85.7 ± 0.3
Robomimic Transport PH	3.3 ± 0.3	12.6 ± 0.5	7.9 ± 0.5	**41.5** ± 0.4	38.9 ± 2.1	38.4 ± 0.2
Robomimic Tool Hang PH	28.0 ± 0.8	75.2 ± 0.5	51.6 ± 0.9	**87.6** ± 1.6	**84.9** ± 1.2	**86.4** ± 0.7

Table 2: Comparison of success rates across Libero-90 and Robomimic for an explicit ensemble. We see that our explicit ensemble outperforms the implicitly-ensembled AC(n) policies.

Takeaway 5: Action chunking is not, in general, necessary for effective real-world robotic control

For certain real-world robotic control settings, even when Markovian policies are insufficient, we can match the performance of action chunking with randomized delay ensembles—playing action chunks is not necessary as long as we predict based on past observations, and replicate the implicit ensembling effect of action chunking.

7 Going Beyond Action Chunking with Ensembled BC Policies

Our previous results suggest that the key benefits of action chunking are (a) its ability to condition on delayed observations and (b) its implicit ensembling behavior. Here, we consider whether these insights motivate more effective approaches to BC for robotic control. While a full investigation of this is beyond the scope of this work (see the following section for discussion of other promising directions), here we make a first attempt at applying our insights to improve BC performance.

Our starting point is Figure 9, where we see that a true ensemble of delayed policies achieves even lower action prediction loss than the temporal ensemble induced by the action-chunked policy. Motivated by this, we hope to amplify these benefits of action chunking by deploying an actual ensemble—that is, a set of policies $\{\widehat{\pi}_{20,i}\}_{i=1}^{m}$ trained independently on $\mathfrak{D}_{\text{train}}$, as described above. If the benefits of action chunking are indeed what our analysis suggests, we would hope that by training an explicit ensemble (rather than the implicit ensemble induced by an action-chunked policy) we might achieve even better performance.

Here we consider the ensemble $\{\widehat{\pi}_{20,i}\}_{i=1}^{m}$ (AC(n)-Ens) as well as the induced ensemble $\{\pi^{n}_{\text{delay}}[\widehat{\pi}_{20,i}]\}_{i=1}^{m}$ (Delay($n+1$)-Ens) and the randomized delay ensemble considered in Experiment 6, but where we now also sample an ensemble member $i \in \text{unif}([m])$ at each step in addition to a delay (AC(n)-RDE-Ens). We consider two aggregation approaches for AC(n)-Ens and Delay($n+1$)-Ens: either predicting an action from each ensemble and averaging their predictions, or randomly sampling an ensemble index and simply computing the action from that ensemble member. We provide results for both types of ensembles in Table 2 (in all cases providing results for the best-case delay or action chunk length, and ensemble aggregation approach). We see that, *across all settings, explicit ensembles improve on the performance of action chunking*, and this is true whether we use the full action-chunked ensemble or the ensemble of induced delay policy. Notably, on Robomimic Transport, we see an approximately +30% boost in performance from using the explicit ensemble over the action-chunked policy. Not only do these results corroborate our findings on action chunking—we can match the performance of action chunking by using delayed ensembles—they also suggest a promising approach for improving the performance of BC policies.

8 Conclusion

In this work, we present an analysis of the mechanisms enabling the effective performance of action chunking. We examine in detail three mechanisms already postulated in existing litera-

ture [2, 50, 56]: the ability to express temporally consistent behaviors, a horizon-reduction effect, and more favorable representation learning. We show that existing hypothesis fail to fully capture the benefits of action chunking, and instead demonstrate that non-Markovian expressivity and reduction in compounding error—both captured by simple delayed policies—as well as the implicit ensembling effect of action chunking policies, is able to almost fully explain the success of action chunking. Our results hold on standard simulated benchmarks, as well as real-world robotic manipulation tasks. We further show that by taking an *explicit* ensemble of delayed policies, it is possible to further increase performance beyond that of action chunking. We believe this work opens up a variety of interesting directions for future work.

Going Beyond Action Chunking. How can we improve over the performance of action chunking? While our results in Section 7 are directly motivated by our observations that action chunking acts as an implicit ensemble, we believe our insights motivate other approaches that could lead to further improvements. As a concrete direction, we note that in Figure 2, delays of around 5-15 achieve the lowest validation error. While our results on implicit ensembling suggest that utilizing multiple delays is important, could we adaptively select the delays based on which achieve lowest validation error, and could this improve performance further?

Non-Markovian Behavior and History Conditioning. Our results on non-Markovian expressivity suggest that this expressivity is a primary benefit of action chunking. While our results show that a simple delayed policy captures much of the necessary non-Markovian behavior, to capture general non-Markovian behavior we may need a fully history conditioned policy, modeling $a_t \mid o_t, o_{t-1}, o_{t-2}, \ldots$. Progress has been made developing history-conditioned policies for robotic control [57, 58], yet fully incorporating history-conditioning has proved challenging. Interestingly, our results suggest that history conditioning may not be the full solution. While history conditioning would capture non-Markovian behavior, it is not clear that it captures the reduction in compounding error or implicit ensembling effects induced by action chunking. How can we ensure that we obtain the other benefits of action chunking while incorporating history-conditioning to fully express non-Markovian behavior?

The Role of Control Frequency. In all our experiments we utilized a control frequency in the range of 15-20 Hz (for simulated experiments, demonstrations were collected at 20 Hz). Experimentally, we found that when utilizing higher control frequencies, for example 50-60 Hz, delayed policies were not able to replicate the success of action chunking, but treating sub-chunks of length 5 as "actions" (so that the effective control frequency is in the range of 10-20Hz), performance was significantly improved. In other words, the temporal consistency of action chunking *does* matter, but only at high frequencies. We believe that this may be due to the frequency at which humans operate. There is evidence that human's visually guided behavior operates at a frequency in the 2-10Hz range [59, 60], and, as a result, at a control frequency of 50-60Hz, actions are recorded at a much higher frequency than the human demonstrator is updating their behavior, so there may exist significant temporal correlation between actions that is effectively modeled with action chunking. While this is consistent with our analysis—we can see this as simply another form of non-Markovian behavior—it suggests that there exists a complex relationship between the behaviors expressed by the demonstrator, the control frequency of the policy, and the role of action chunking. We believe that deeper investigation into this interaction could lead to further insights into the role of action chunking.

Practical Benefits of Action Chunking. As noted in Section 6, unlike in simulation where there is no "pause" in the environment necessary for policy inference, in the real world pauses for inference can affect policy performance. Fundamentally, such pauses introduce distribution shift between the deployed policy and the demonstrator—the demonstrator has no "inference pauses", and if the robot state changes at inference pauses in policy deployment, this could lead to greater compounding error. As such, mitigating pauses due to inference will likely lead to improved performance. Action chunking naturally reduces inference pauses, and recent work has sought to eliminate the need for any inference calls [61]. To what extent does the reduction in inference pauses contribute to the benefits of action chunking in practice?

Acknowledgments

We acknowledge ISCRA for awarding this project access to the LEONARDO supercomputer, owned by the EuroHPC Joint Undertaking, hosted by CINECA (Italy). In addition, we acknowledge support from ONR N00014-25-1-2060.

References

[1] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware, 2023. URL https://arxiv.org/abs/2304.13705.

[2] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.

[3] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, S. Jakubczak, T. Jones, L. Ke, S. Levine, A. Li-Bell, M. Mothukuri, S. Nair, K. Pertsch, L. X. Shi, J. Tanner, Q. Vuong, A. Walling, H. Wang, and U. Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2024. URL https://arxiv.org/abs/2410.24164.

[4] K. Pertsch, K. Stachowicz, B. Ichter, D. Driess, S. Nair, Q. Vuong, O. Mees, C. Finn, and S. Levine. Fast: Efficient action tokenization for vision-language-action models, 2025. URL https://arxiv.org/abs/2501.09747.

[5] T. Z. Zhao, J. Tompson, D. Driess, P. Florence, K. Ghasemipour, C. Finn, and A. Wahid. Aloha unleashed: A simple recipe for robot dexterity. *arXiv preprint arXiv:2410.13126*, 2024.

[6] Q. Li, Z. Zhou, and S. Levine. Reinforcement learning with action chunking. *arXiv preprint arXiv:2507.07969*, 2025.

[7] M. Torne, A. Tang, Y. Liu, and C. Finn. Learning long-context diffusion policies via past-token prediction. *arXiv preprint arXiv:2505.09561*, 2025.

[8] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. *Advances in neural information processing systems*, 7, 1994.

[9] T. K. Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.

[10] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[11] D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 1. Morgan-Kaufmann, 1988. URL https://proceedings.neurips.cc/paper_files/paper/1988/file/812b4ba287f5ee0bc9d43bbf5bbe87fb-Paper.pdf.

[12] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[13] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR. URL https://proceedings.mlr.press/v15/ross11a.html.

[14] M. Bojarski. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[15] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel. Deep imitation learning for complex manipulation tasks from virtual reality teleoperation. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 5628–5635. IEEE, 2018.

[16] R. Rahmatizadeh, P. Abolghasemi, L. Bölöni, and S. Levine. 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)*, pages 3758–3765. IEEE, 2018.

[17] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021.

[18] H. R. Walke, K. Black, T. Z. Zhao, Q. Vuong, C. Zheng, P. Hansen-Estruch, A. W. He, V. Myers, M. J. Kim, M. Du, et al. Bridgedata v2: A dataset for robot learning at scale. In *Conference on Robot Learning*, pages 1723–1736. PMLR, 2023.

[19] A. O'Neill, A. Rehman, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar, A. Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903. IEEE, 2024.

[20] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna, S. Dasari, S. Karamcheti, S. Nasiriany, M. K. Srirama, L. Y. Chen, K. Ellis, P. D. Fagan, J. Hejna, M. Itkina, M. Lepert, Y. J. Ma, P. T. Miller, J. Wu, S. Belkhale, S. Dass, H. Ha, A. Jain, A. Lee, Y. Lee, M. Memmel, S. Park, I. Radosavovic, K. Wang, A. Zhan, K. Black, C. Chi, K. B. Hatch, S. Lin, J. Lu, J. Mercat, A. Rehman, P. R. Sanketi, A. Sharma, C. Simpson, Q. Vuong, H. R. Walke, B. Wulfe, T. Xiao, J. H. Yang, A. Yavary, T. Z. Zhao, C. Agia, R. Baijal, M. G. Castro, D. Chen, Q. Chen, T. Chung, J. Drake, E. P. Foster, J. Gao, V. Guizilini, D. A. Herrera, M. Heo, K. Hsu, J. Hu, M. Z. Irshad, D. Jackson, C. Le, Y. Li, K. Lin, R. Lin, Z. Ma, A. Maddukuri, S. Mirchandani, D. Morton, T. Nguyen, A. O'Neill, R. Scalise, D. Seale, V. Son, S. Tian, E. Tran, A. E. Wang, Y. Wu, A. Xie, J. Yang, P. Yin, Y. Zhang, O. Bastani, G. Berseth, J. Bohg, K. Goldberg, A. Gupta, A. Gupta, D. Jayaraman, J. J. Lim, J. Malik, R. Martín-Martín, S. Ramamoorthy, D. Sadigh, S. Song, J. Wu, M. C. Yip, Y. Zhu, T. Kollar, S. Levine, and C. Finn. Droid: A large-scale in-the-wild robot manipulation dataset. 2024.

[21] N. M. Shafiullah, Z. Cui, A. A. Altanzaya, and L. Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35:22955–22968, 2022.

[22] Z. J. Cui, Y. Wang, N. M. M. Shafiullah, and L. Pinto. From play to policy: Conditional behavior generation from uncurated robot data. *arXiv preprint arXiv:2210.10047*, 2022.

[23] S. Dasari, O. Mees, S. Zhao, M. K. Srirama, and S. Levine. The ingredients for robotic diffusion transformers. *arXiv preprint arXiv:2410.10088*, 2024.

[24] L. Ankile, A. Simeonov, I. Shenfeld, and P. Agrawal. Juicer: Data-efficient imitation learning for robotic assembly. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5096–5103. IEEE, 2024.

[25] Y. Ze, G. Zhang, K. Zhang, C. Hu, M. Wang, and H. Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. *arXiv preprint arXiv:2403.03954*, 2024.

[26] A. Sridhar, D. Shah, C. Glossop, and S. Levine. Nomad: Goal masked diffusion policies for navigation and exploration. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 63–70. IEEE, 2024.

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

[28] J. Gu, S. Kirmani, P. Wohlhart, Y. Lu, M. G. Arenas, K. Rao, W. Yu, C. Fu, K. Gopalakrishnan, Z. Xu, et al. Rt-trajectory: Robotic task generalization via hindsight trajectory sketches. *arXiv preprint arXiv:2311.01977*, 2023.

[29] O. M. Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, T. Kreiman, C. Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

[30] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[31] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[32] J. Bjorck, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. Fan, Y. Fang, D. Fox, F. Hu, S. Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

[33] P. Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

[34] L. Zha, A. J. Hancock, M. Zhang, T. Yin, Y. Huang, D. Shah, A. Z. Ren, and A. Majumdar. Lap: Language-action pre-training enables zero-shot cross-embodiment transfer. *arXiv preprint arXiv:2602.10556*, 2026.

[35] H. Fang, J. Duan, D. Clay, S. Wang, S. Liu, W. Huang, X. Fan, W.-C. Tsai, S. Chen, Y. R. Wang, S. Xing, J. Cho, J. S. Park, A. Eftekhar, P. Sushko, K. Farley, A. Wadhwa, C. Harrison, W. Han, Y.-C. Lee, E. VanderBilt, R. Hendrix, S. Ellawela, L. Ngoo, J. Chai, Z. Ren, A. Farhadi, D. Fox, and R. Krishna. Molmoact2: Action reasoning models for real-world deployment. *arXiv preprint arXiv:2605.02881*, 2026. URL https://allenai.org/blog/molmoact2.

[36] Gemini Robotics Team. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025.

[37] TRI LBM Team. A careful examination of large behavior models for multitask dexterous manipulation. *arXiv preprint arXiv:2507.05331*, 2025. URL https://toyotaresearchinstitute.github.io/lbm1/.

[38] J. Pai, L. Achenbach, V. Montesinos, B. Forrai, O. Mees, and E. Nava. mimic-video: Video-action models for generalizable robot control beyond vlas. *arXiv preprint arXiv:2512.15692*, 2025. URL https://mimic-video.github.io/.

[39] T. Z. Zhao, J. Tompson, D. Driess, P. Florence, K. Ghasemipour, C. Finn, and A. Wahid. Aloha unleashed: A simple recipe for robot dexterity. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*. PMLR, 2025. URL https://proceedings.mlr.press/v270/zhao25b.html.

[40] A. Wagenmaker, P. Dong, R. Tsao, C. Finn, and S. Levine. Posterior behavioral cloning: Pretraining bc policies for efficient rl finetuning. *arXiv preprint arXiv:2512.16911*, 2025.

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

[42] Y. Liu, J. Hamid, A. Xie, Y. Lee, M. Du, and C. Finn. Bidirectional decoding: Improving action chunking via guided test-time sampling. In *International Conference on Learning Representations*, volume 2025, pages 4594–4627, 2025.

[43] R. Malhotra, Y. Liu, and C. Finn. Self-guided action diffusion. *arXiv preprint arXiv:2508.12189*, 2025.

[44] M. Park, K. Kim, J. Hyung, H. Jang, H. Jin, J. Yun, H. Lee, and J. Choo. Acg: Action coherence guidance for flow-based vla models. *arXiv preprint arXiv:2510.22201*, 2025.

[45] K. Black, M. Galliker, and S. Levine. Real-time execution of action chunking flow policies. *Advances in Neural Information Processing Systems*, 38:33383–33407, 2026.

[46] K. Black, A. Z. Ren, M. Equi, and S. Levine. Training-time action conditioning for efficient real-time chunking. *arXiv preprint arXiv:2512.05964*, 2025.

[47] D. Jing, G. Wang, J. Liu, W. Tang, Z. Sun, Y. Yao, Z. Wei, Y. Liu, Z. Lu, and M. Ding. Mixture of horizons in action chunking. *arXiv preprint arXiv:2511.19433*, 2025.

[48] Y. Weng, X. Zhang, Y. Mu, Y. Zhu, and Y. Li. Temporal action selection for action chunking. *arXiv preprint arXiv:2511.04421*, 2025.

[49] W. Chen, K. Zhang, C. Lin, Z. Zhang, Y. She, Y. Liu, R. A. Yeh, S. Mou, and Y. Gu. Dreamchunk: Reactive action chunking with latent world model. *arXiv preprint arXiv:2606.18589*, 2026.

[50] M. Simchowitz, D. Pfrommer, and A. Jadbabaie. The pitfalls of imitation learning when actions are continuous. *arXiv preprint arXiv:2503.09722*, 2025.

[51] T. T. Zhang, D. Pfrommer, N. Matni, and M. Simchowitz. Imitation learning in continuous action spaces: mitigating compounding error without interaction. *arXiv preprint arXiv*, 2507, 2025.

[52] C. Pan, G. Anantharaman, N.-C. Huang, C. Jin, D. Pfrommer, C. Yuan, F. Permenter, G. Qu, N. Boffi, G. Shi, et al. Much ado about noising: Dispelling the myths of generative robotic control. *arXiv preprint arXiv:2512.01809*, 2025.

[53] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, and P. Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *arXiv preprint arXiv:2306.03310*, 2023.

[54] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *arXiv preprint arXiv:2108.03298*, 2021.

[55] D. J. Foster, A. Block, and D. Misra. Is behavior cloning all you need? understanding horizon in imitation learning. *Advances in Neural Information Processing Systems*, 37:120602–120666, 2024.

[56] T. T. Zhang, D. Pfrommer, C. Pan, N. Matni, and M. Simchowitz. Action chunking and exploratory data collection yield exponential improvements in behavior cloning for continuous control, 2025. URL https://arxiv.org/abs/2507.09061.

[57] M. Torne, K. Pertsch, H. Walke, K. Vedder, S. Nair, B. Ichter, A. Z. Ren, H. Wang, J. Tang, K. Stachowicz, et al. Mem: Multi-scale embodied memory for vision language action models. *arXiv preprint arXiv:2603.03596*, 2026.

[58] M. S. Mark, J. Liang, M. Attarian, C. Fu, D. Dwibedi, D. Shah, and A. Kumar. Bpp: Long-context robot imitation learning by focusing on key history frames. *arXiv preprint arXiv:2602.15010*, 2026.

[59] B. Day and I. Lyon. Voluntary modification of automatic arm movements evoked by motion of a visual target. *Experimental Brain Research*, 130(2):159–168, 2000.

[60] D. Susilaradeya, W. Xu, T. M. Hall, F. Galan, K. Alter, and A. Jackson. Extrinsic and intrinsic dynamics in movement intermittency. *Elife*, 8:e40145, 2019.

[61] K. Black, M. Y. Galliker, and S. Levine. Real-time execution of action chunking flow policies, 2025. URL https://arxiv.org/abs/2506.07339.

[62] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, K. Lin, A. Maddukuri, S. Nasiriany, and Y. Zhu. robosuite: A modular simulation framework and benchmark for robot learning, 2025. URL https://arxiv.org/abs/2009.12293.

[63] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 2023.

[64] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6840–6851. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf.

[65] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

[66] D. Misra. Mish: A self regularized non-monotonic activation function, 2020. URL https://arxiv.org/abs/1908.08681.

[67] I. Loshchilov and F. Hutter. Decoupled weight decay regularization, 2019. URL https://arxiv.org/abs/1711.05101.

[68] I. Loshchilov and F. Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Skq89Scxx.

[69] R. Laroche and R. Tachet Des Combes. On the occupancy measure of non-Markovian policies in continuous MDPs. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 18548–18562. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/laroche23a.html.

A Additional Experimental Details

In the following, we provide additional experimental details for all the simulations conducted in the paper. We begin with a discussion on the environments and datasets considered (Appendix A.2), then describe the network architectures considered (Appendix A.3), and, finally, we provide additional details for each simulation in the main paper (Appendix A.4).

We mention that all simulations were carried out using one Nvidia H100 GPU and ten Nvidia A100 GPUs.

A.1 Validation Error vs Validation Loss

Here we expand on our choice of the validation error metric, $L_{\text{val}}(\pi)$, that evaluates the action prediction error. In particular, $L_{\text{val}}(\pi)$ measures how far the expected action predictions of policy π are from the actual actions in the validation set. In the setting where π is a diffusion policy, as we consider in this work, another natural choice of validation error would simply be the denoising diffusion loss on the validation set, that is:

$$\mathbb{E}_{(\boldsymbol{a},o)\sim\text{unif}(\mathfrak{D}_{\text{val}})}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(0,I)}\mathbb{E}_{t\sim\text{unif}([0,1])}[\|\boldsymbol{\epsilon}-\boldsymbol{\epsilon}_\theta(\sqrt{\alpha_t}\boldsymbol{a}+\sqrt{1-\alpha_t}\boldsymbol{\epsilon},t;o)\|_2^2]$$

where $\boldsymbol{\epsilon}_\theta$ is the policy's denoising network. The primary challenge with using this loss in our setting is that it does not allow us to isolate the contribution of each individual timestep. In particular, we want to evaluate the validation loss at step i in the action chunk, we could compute this as

$$\mathbb{E}_{(\boldsymbol{a},o)\sim\text{unif}(\mathfrak{D}_{\text{val}})}\mathbb{E}_{\boldsymbol{\epsilon}\sim\mathcal{N}(0,I)}\mathbb{E}_{t\sim\text{unif}([0,1])}[\|[\boldsymbol{\epsilon}-\boldsymbol{\epsilon}_\theta(\sqrt{\alpha_t}\boldsymbol{a}+\sqrt{1-\alpha_t}\boldsymbol{\epsilon},t;o)]_i\|_2^2],$$

but this loss allows information to bleed over from other timesteps in the chunk to step i—as the network observes all of $\sqrt{\alpha_t}\boldsymbol{a}+\sqrt{1-\alpha_t}\boldsymbol{\epsilon}$, it sees a noisy version of the action at steps $i+1$ and $i-1$, for example, so its prediction at step i may depend strongly on this. As such, it is not possible to isolate the prediction of step i in the chunk as we do, for example, in Figure 2.

The second challenge with this validation loss is that it does not extend to the ensembling validation loss we consider in Section 5. Since we consider ensembled predictions across timesteps in the chunk, it is not obvious we can directly ensemble in the diffusion noise space—in particular, $\boldsymbol{\epsilon}_\theta(\sqrt{\alpha_t}\boldsymbol{a}+\sqrt{1-\alpha_t}\boldsymbol{\epsilon},t;o_h)$ and $\boldsymbol{\epsilon}_\theta(\sqrt{\alpha_t}\boldsymbol{a}+\sqrt{1-\alpha_t}\boldsymbol{\epsilon},t;o_{h-d})$ cannot be combined directly, since, while they may partially overlap, they ultimately predict different action chunks, and it is not obvious how action chunks can be combined in noise space.

Given this, we opt instead to use the validation error that simply evaluates the actual action predictions. Note that this resolves the first issue since all action predictions begin from noise $\boldsymbol{\epsilon}\sim\mathcal{N}(0,I)$, so oracle information from one step in the chunk cannot affect the prediction at another step in the chunk (since this has access to no oracle information). In addition, it also resolves the second issue since we can easily combine a given step of the chunk in action space. While in principle the validation error we consider may not incorporate multimodality effectively, we found that in practice the diffusion policies we trained were not multimodal—for a given observation, o, the predictions tend to be unimodal—so this did not prove an issue in practice.

A.2 Description of Environments and Datasets

`Robomimic` [54] and `Libero` [53] are two widely adopted benchmarks built on top of the `Robosuite` project [62]. We consider tasks `square`, `can`, `transport` and `tool hang` for `Robomimic`, and all the 90 tasks in the `Libero 90` suite for `Libero`.

Robomimic. For each `Robomimic` task, we considered the PH (Proficient-Human) dataset, consisting of $N = 200$ successful trajectories collected by a "single, experienced teleoperator" [54], at a control frequency of 20Hz. Actions for all these tasks except `transport` are 7-dimensional vectors where "the first 3 coordinates are the desired translation from the current end effector position, the next 3 coordinates encoder the desired delta rotation from the current end effector rotation, and

Hyperparameter	Libero	Robomimic
denoising net architecture	MLP	MLP
denoising net size	$[4096, 4096, 4096, 4096, 4096]$	$[3000, 3000, 3000, 3000, 3000]$
conditioning net architecture	ViT	MLP
conditioning net size	4 heads, depth 1, spatial embedding 128	$[1024, 1024, 64]$
multi-task policy	yes	no

Table 3: High-level net architecture

the final coordinate controls the opening and closing of the gripper fingers." For `transport`, since there are two robot arms, then the action space is 14-dimensional. Regarding observations, we avoid using images, but consider the full low-dimensional object observations provided by the `Robomimic` suite, made of proprioception observations (9-dimensional per arm, "consisting of the end effector position (3-dim), quaternion (4-dim), and gripper finger positions (2-dim)"), and ground-truth object states (with varying dimensionality depending on the task, containing in general the absolute position of the object/s and their relative position wrt the robot end effector). See Appendix E of Mandlekar et al. [54] for more details.

Libero. For each `Libero 90` task, we are given 50 successful demonstrations from a human demonstrator. All 90 tasks share the same action space, which coincides with the action space of all the considered `Robomimic` tasks (i.e., a 7-dimensional vector encoding the delta translation/rotation of the end effector pose and the opening/closing of the gripper). Regarding the observation space, things are rather different from `Robomimic`, as we do not consider ground-truth object states, but image observations. So, our observation is made of proprioception observations (9-dimensional per arm, consisting of two components for the gripper finger positions, and seven components for the joint orientations of the arm), and image observations (that is, two (128,128,3)-dimensional images, one from the front-view camera and one from the wrist-mounted camera).

A.3 Network Architecture

For all the experiments, we trained diffusion policies [63] to fit the expert demonstrations, with some differences between `Robomimic` and `Libero` in order to handle the different types of observations. From a high-level perspective, we used conditional DDPMs [64] to fit (sequences of) actions given observations, where we use a multi-layer perceptron (MLP) for the denoising network, and either an MLP (`Robomimic`) or a Vision Transformer (ViT [65], `Libero`) to encode the observation features. For `Robomimic`, we always train single-task policies (i.e., different policies for each task), where the denoising net is made of 5 layers of 3000 neurons each, while the MLP to encode observations is made of three layers of 1024, 1024, and 64 neurons. Instead, for `Libero`, we train multi-task policies, i.e., a single network to predict actions for all the 90 tasks. We accomplish this by including into the observation a 90-dim one-hot encoding of the task. The denoising net for `Libero` is made of 5 layers of 4096 neurons each, while we encode observations using a ViT encoder trained along with the network using a patch size of 8, 4 heads, and a depth of 1. The output embedding dimension will be 128. The two images are then concatenated before being given in input to the (denoising net) MLP. Before passing the images in input to the ViT encoder, we perform some random shift (translation) data augmentation for images: we randomly move each image 4 pixels horizontally and vertically. See Table 3 for a summary of this description.

Hyperparameters. The hyperparameters we adopt for training/optimization and for the DDPM are the same for both `Robomimic` and `Libero`, and are reported in Table 4. More specifically, we use 20 denoising steps and adopt a 32-dimensional embedding for the time step (we use a sinusoidal positional embedding, as standard in DDPMs). The activation function we use is Mish [66], and we add residual connections in the denoising MLP nets. We train our DDPMs for 3000 epochs with AdamW [67] with learning rate 1e-4 and weight decay 1e-6. We adopt a cosine annealing scheduler for the learning rate [68] with 100 warmup steps and minimum and maximum learning rates of 1e-5

Hyperparameter	Value
activation function	Mish
use residual connections	True
batch size	256
use EMA	True
EMA decay	0.995
optimizer	AdamW
lr	1e-4
weight decay	1e-6
number of epochs	3000
time embedding	sinusoidal positional
time embedding size	32
denoising steps	20

Table 4: Hyperparameters

and 1e-4, with 3000 first cycle steps. We use a batch size of 256. We also use an empirical moving average (EMA) of the model weights, which starts at epoch 20, and is updated every 10 epochs with decay 0.995. It is worthy mentioning that, while for `Robomimic` one epoch corresponds to a number of batches that spans the whole training dataset, for `Libero`, where we have much more data as we train multi-task policies, one epoch corresponds to a randomly chosen subset of all the training samples, computed as $300/N$ (where N is the total number of expert trajectories). Simply put, we are merely rescaling the true number of gradient iterations to make it more comparable to `Robomimic`, where we have less expert trajectories.

Padding and normalization. We perform padding at the end of expert's trajectories every time that we train with chunk sizes $k > 1$ or pure delayed policies with delay $d > 0$, by replicating the last action for $k - 1$ times. Before training, we normalize the expert trajectories as it is best practice for DDPMs. In particular, in both `Robomimic` and `Libero`, we normalized the low-dimensional components of the state/observation, and the action with a min-max normalization to $[-1, +1]$: given the (300 or 50) expert's trajectories, we computed for each state and action component, the min and max values, and then we applied min-max normalization: $s_{\text{norm}} := 2[(s - s_{\min})/(s_{\max} - s_{\min} + 1e-6) - 0.5]$, $a_{\text{norm}} := 2[(a - a_{\min})/(a_{\max} - a_{\min} + 1e-6) - 0.5]$. Instead, `Libero` image observations, having discrete values in $\{0, 1, \dots, 255\}$, have been normalized through: $i_{\text{norm}} := i/255.0 - 0.5$.

A.4 Additional Details for each Simulation

Here we provide additional simulation-specific details for all the experiments described in the main paper, in particular for Figs. 2-9 and Tables 1-2. First, we describe the evaluation protocol adopted.

A.4.1 Evaluation Protocol

For every `Robomimic` task, the success rate of each policy seed is assessed based on 250 rollouts played for a maximum number of timesteps of 300 for `square`, 250 for `can`, 650 for both `transport` and `tool hang`, while for each `Libero` task we consider 40 rollouts played for a maximum number of timesteps of 400. Moreover, in `Robomimic` the mean action used for the computation of the validation loss is the average of 50 action samples, while in `Libero` we used 20 action samples. As a last note, we mention that throughout all the paper, for delayed policies induced by some chunked (e.g., $\pi^k_{\text{delay}}[\widehat{\pi}_{20}]$), since they are not well-defined in the first timesteps of a simulation, we initialize the simulation by playing a single chunk of $\widehat{\pi}^{k-1}_{20}$.

A.4.2 `Libero` Validation Loss (Figure 2)

For this simulation, we trained three $\widehat{\pi}_{20}$ policies on a subset of all the `Libero` data. Specifically, instead of using all the 50 expert trajectories for each of the 90 tasks (overall $50 * 90 = 4500$), we used for training only 25 expert trajectories for each task (overall $25 * 90 = 2250$), and kept the remaining data for validation. Note that both policies have been trained on the *same* data, but using different random seeds for training, which affect the weight initialization, the random batches selection, etc.. For validation, for every delay $k \in \{0\} \cup [19]$, for every timestep $h \geqslant 20$ and validation trajectory i,[5] we sampled 20 actions from $\pi^k_{\text{delay}}[\widehat{\pi}_{20}](\cdot|s^i_{h-k})$, averaged them obtaining $\bar{a}$, and then computed the squared error between $\bar{a}$ and the expert's action a^i_h, for all coordinates except the gripper component: $\|\bar{a} - a^i_h\|_2^2$ (note that both actions are *normalized* during this simulation). We repeated this procedure for each of the 90 tasks, for each of the 25 validation trajectories of each task, and for each timestep of each trajectory (whose length depends on the specific demonstration and task), and finally averaged over all these values (note that this procedure makes tasks with longer horizon more representative, compliant to the training procedure). After having obtained these 20 values for the first policy seed, we repeated the same procedure for the two other policy seeds. Finally, we compute the mean and standard error of the mean (SEM, standard deviation divided by $\sqrt{3}$ in this case) for these three 20-dim arrays, obtaining the Delay(n) plot of Fig. 2. Instead, the AC(n) plot has been obtained by replacing the three 20-dim arrays with their cumulative means, and then computing the mean and SEM similarly as for Delay(n).

A.4.3 `Libero` Success Rate (Figure 3)

We trained three $\widehat{\pi}_{20}$ policies on all the `Libero` data with different random seeds, rolled them out either as $\pi^k_{\text{delay}}[\widehat{\pi}_{20}]$ or as $\widehat{\pi}^{k'}_{20}$ for $k \in \{0, 2, 5, 8, 12, 15, 19\}$ and $k' \in \{3, 6, 9, 13, 16, 20\}$ (recall that $\pi^0_{\text{delay}}[\widehat{\pi}_{20}]$ coincides with $\widehat{\pi}^1_{20}$), and finally computed the mean/SEM over these three seeds (so neglecting the variability due to using a finite number of rollouts per task).

A.4.4 `Libero` Success Rate vs. Validation Loss (Figure 4)

For this simulation, we used the same validation values as for Fig. 2 (see Appendix A.4.2), obtained by averaging over the three policy seeds but not over the tasks. Instead, the success rates are obtained as those in Table 1 (see Appendix A.4.8), i.e., by training fifteen policies $\widehat{\pi}_{20}$ on all the `Libero` data with different random seeds, rolling them out either as $\widehat{\pi}^1_{20}$ or as $\widehat{\pi}^{10}_{20}$ (we do 40 rollouts per task), and finally averaging the success rate over the fifteen policy seeds. The line is computed as the least square solution over all the plotted points.

A.4.5 `Libero` With and Without Action Chunking (Figure 7)

For $\widehat{\pi}^1_{20}$ and $\pi^5_{\text{delay}}[\widehat{\pi}_{20}]$, we trained three $\widehat{\pi}_{20}$ policies on all the `Libero` data with three different random seeds. Instead, for $\widehat{\pi}^1_1$ we actually trained three $\widehat{\pi}_1$ policies, while for π^5_{delay} we also trained three different policies of this kind. Then evaluation has been carried out as usual, with 40 rollouts per task, then averaging the success rate among all tasks. The SEM reported in Fig. 7 corresponds to the three seeds only. As a last note on this experiment, we would like to mention that, for π^5_{delay}, differently from the induced one $\pi^5_{\text{delay}}[\widehat{\pi}_{20}]$, we cannot initialize it with a chunk; thus, what we did was just to play the "wrong" action for the first five timesteps, that is, for timestep $0 \leqslant h < 5$, we simply gave the first observation of the simulation o_0 in input to policy π^5_{delay}, and played the outputted action.

A.4.6 `Robomimic` Success Rate (Figure 8)

For each of the four `Robomimic` tasks considered, we trained ten $\widehat{\pi}_{20}$ using different random seeds, rolled them out in the corresponding environment either as $\pi^k_{\text{delay}}[\widehat{\pi}_{20}]$ or as $\widehat{\pi}^{k'}_{20}$ for

[5] Note that we skip the first 20 timesteps of each validation trajectory to make a fair comparison with higher delays. Indeed, while we can assess the validation error for $k = 0$, we are not able to do so for, e.g., $k = 15$ when $h < 15$.

Algorithm	$\mathcal{J}$	t succ
Markovian	69.0 ± 0.8	157 ± 1
AC(5)	83.3 ± 0.8	148 ± 0
AC(10)	85.4 ± 0.5	144 ± 0
Delay(5)	76.8 ± 0.8	152 ± 1
Delay(3)	80.8 ± 0.6	150 ± 0
AC(10)-RDE	82.4 ± 0.6	146 ± 0
AC(10)-TE	80.6 ± 0.5	147 ± 0
AC(5)-Ens mean	87.5 ± 0.5	145 ± 1
AC(5)-Ens random	88.0 ± 0.3	146 ± 1
AC(10)-Ens mean	88.3 ± 0.3	143 ± 1
AC(10)-Ens random	87.1 ± 0.2	143 ± 0
Delay(3)-Ens mean	87.5 ± 0.2	148 ± 1
Delay(3)-Ens random	87.7 ± 1.1	147 ± 1
Delay(5)-Ens mean	85.9 ± 0.7	147 ± 1
Delay(5)-Ens random	84.8 ± 1.3	148 ± 0

Table 5: Complete success rate results for `Robomimic square PH`.

$k \in \{0, 2, 4, 7, 10, 13, 16\}$ and $k' \in \{5, 10, 15, 20\}$, and then computed the success rates. The values reported in Fig. 8 correspond to the mean and SEM over these forty success rates.

A.4.7 `Robomimic` Validation Loss (Figure 9)

The calculation of the validation loss for AC(n) and Delay(n) is analogous to those in Appendix A.4.2, with the difference that we use ten policy seeds trained on a subset of 30 demonstration trajectories per task (the remaining 170 expert trajectories are used for validation) and compute means over 50 sample actions. For AC(n)-TE, calculations are analogous. Instead, for Delay(n)-Ens, we considered three subsets of five policy seeds each from this group of ten policy seeds; for each group of seeds, our action predictions are the average of mean actions over the considered five seeds. Then, calculation of mean and SEM follows analogously using these three values.

A.4.8 General Results (Tables 1-2)

We trained fifteen $\widehat{\pi}_{20}$ policies on all the `Libero` data (one multitask policy for each libero suite) with different random seeds, and then rolled out each policy seed as AC(n) for $n \in \{1, 10\}$, Delay(n) for $n \in \{6\}$, AC(n)-RDE for $n \in \{10\}$, and AC(n)-TE for $n \in \{10\}$ (to be precise, we run all the fifteen seeds only for `Libero-90`, while "only" ten seeds for other suites, as success rate was roughly already converged). For the ensembles AC(n)-Ens, Delay(n)-Ens and AC(n)-RDE-Ens, we grouped the fifteen policy seeds into three disjoint groups of five seeds each, and then rolled out each ensemble and averaged the results. We used $n = 10$ for AC(n)-Ens and AC(n)-RDE-Ens, and $n = 5$ for Delay(n)-Ens. We did so for both aggregation approaches described in Section 5.

Regarding `Robomimic`, things are similar: we trained fifteen $\widehat{\pi}_{20}$ policies on *each* `Robomimic` task using all the data, rolled them out, then average results. The difference is that we considered multiple values for n, specifically: for AC(n) we used $n \in \{1, 5, 10\}$, for D(n) we used $n \in \{3, 5\}$, for AC(n)-Ens we used $n \in \{5, 10\}$ for both aggregation methods, for Delay(n)-Ens we used $n \in \{3, 5\}$ for both aggregation methods, while for AC(n)-RDE-Ens we used only $n = 10$. Complete results are reported in Tables 5-8.

A.5 Additional Details for Real-World Experiments

We trained our diffusion policies using the same hyperparameters as for `Libero`, with the differences that the MLP has five layers with 2000 neurons each, images in input have dimension $120 \times 120 \times 3$, and we use 15 denoising steps instead of 20 (to speed up inference for real-time evaluation of the policies). Note that, as action space, we use delta control for training and playing our policies (where delta is referred to the current state, and not the initial state of the chunk). To achieve this, we convert

Algorithm	$\mathcal{J}$	t succ
Markovian	83.7 ± 0.4	116 ± 0
AC(5)	95.2 ± 0.4	112 ± 0
AC(10)	97.2 ± 0.3	110 ± 0
Delay(5)	93.1 ± 0.4	112 ± 0
Delay(3)	93.5 ± 0.4	112 ± 0
AC(10)-RDE	96.7 ± 0.2	110 ± 0
AC(10)-TE	96.2 ± 0.3	113 ± 0
AC(5)-Ens mean	97.7 ± 0.1	109 ± 0
AC(5)-Ens random	96.9 ± 0.2	111 ± 0
AC(10)-Ens mean	98.5 ± 0.1	109 ± 0
AC(10)-Ens random	97.6 ± 0.2	110 ± 0
Delay(5)-Ens random	96.9 ± 0.3	110 ± 0
Delay(5)-Ens mean	97.5 ± 0.1	109 ± 0
Delay(3)-Ens random	97.7 ± 0.4	110 ± 0
Delay(3)-Ens mean	97.5 ± 0.3	110 ± 0

Table 6: Complete success rate results for `Robomimic can PH`.

Algorithm	$\mathcal{J}$	t succ
Markovian	3.3 ± 0.3	512 ± 4
AC(5)	7.5 ± 0.5	483 ± 4
AC(10)	12.6 ± 0.5	472 ± 3
Delay(5)	7.9 ± 0.5	479 ± 4
Delay(3)	6.8 ± 0.5	493 ± 3
AC(10)-RDE	12.1 ± 0.5	477 ± 3
AC(10)-TE	12.2 ± 0.6	483 ± 2
AC(5)-Ens mean	39.1 ± 1.3	466 ± 3
AC(5)-Ens random	26.3 ± 1.0	467 ± 2
AC(10)-Ens mean	41.5 ± 0.4	461 ± 1
AC(10)-Ens random	22.8 ± 0.3	467 ± 3
Delay(5)-Ens random	35.6 ± 0.8	470 ± 3
Delay(5)-Ens mean	37.3 ± 0.2	469 ± 4
Delay(3)-Ens random	38.9 ± 2.1	468 ± 3
Delay(3)-Ens mean	37.6 ± 1.2	471 ± 1

Table 7: Complete success rate results for `Robomimic transport PH`.

Algorithm	$\mathcal{J}$	t succ
Markovian	28.0 ± 0.8	494 ± 3
AC(5)	65.2 ± 1.1	455 ± 2
AC(10)	75.2 ± 0.5	441 ± 2
Delay(5)	36.1 ± 0.7	477 ± 3
Delay(3)	51.6 ± 0.9	468 ± 2
AC(10)-RDE	71.8 ± 0.8	449 ± 1
AC(10)-TE	42.2 ± 1.0	462 ± 2
AC(5)-Ens mean	71.9 ± 2.3	441 ± 5
AC(5)-Ens random	87.6 ± 1.6	436 ± 1
AC(10)-Ens mean	74.9 ± 1.8	434 ± 1
AC(10)-Ens random	86.7 ± 1.1	439 ± 2
Delay(5)-Ens random	72.7 ± 1.2	447 ± 2
Delay(5)-Ens mean	46.8 ± 1.1	455 ± 4
Delay(3)-Ens random	84.9 ± 1.2	439 ± 1
Delay(3)-Ens mean	62.7 ± 1.7	447 ± 3

Table 8: Complete success rate results for `Robomimic tool hang PH`.

the actions from their original version as absolute joints, to delta joints (simply by computing the difference with the current joint absolute position), train on this space, and then at inference time

compute the sum between the predicted action and the current position, and give this new (absolute joint) action to the low-level controller. We mention that we use a Robotiq gripper.

For evaluation, for each considered method, we rolled out 50 trajectories all with the same initial states (modulo some accuracy differences in re-setting the objects to the initial positions). Results are reported in Figure 10. Note that, as mentioned in the main text, every policy is actually reported with a missing delay of 1 (which we denote with Delay(2)). So, e.g., AC(10) should be AC(10)-Delay(2), with which we mean the policy induced from $\widehat{\pi}_{20}$ which plays action chunks with size 10 using a delay of 1, i.e., which plays $a_{h:h+10}|s_{h-1}$. In a similar manner, AC(10)-Delay(2)-RDE represents the policy that, at each timestep h, samples at random an integer $i \in [10]$ (crucially, $\{0\} \notin [10]$), and playes $a_h|s_{h-i}$. Note that we decided to apply a delay of 1 to each of these policies in order to run them in real time: indeed, by avoiding to condition on the current state, the inference of the chunk can be executed asynchronous from at least one timestep in the past, giving the policy server enough time to make inference without having to wait for a chunk to be ready.

Crucially, these real-world results *replicate our simulation results*. Note that Delay(7) is better than Delay(2), but still not as good as AC(10)-Delay(2). Instead, the method which involves an implicit ensemble of delayed policies, i.e., AC(10)-Delay(2)-RDE, matches (or even slightly outperforms) AC(10)-Delay(2), as expected.

As a last note, we mention that no tuning of the chunk sizes nor the delay have been made. We just chosen a delay of 1 based on the inference time of our diffusion policies, and then committed in advance to the aforementioned chunk sizes (1 and 10) and delays (5). We then collected 50 rollouts per method and reported the results.

B Additional Simulations

In this appendix, we provide results for additional simulations we conducted. In particular, we begin with a simulation alternative to AC(n)-RDE to show that we do not need temporal consistency in `Robomimic PH` and `Libero-90` (Appendix B.1), then we show that action chunking and delayed policies are more sample efficient than Markovian policies in `Libero-90` (Appendix B.2). Next, we show that the dynamics of both `Robomimic PH` and `Libero` are smooth, supporting our assumption for Theorem 2 (Appendix B.3). Finally, we report additional results on `Robomimic MH` (Multi-Human datasets [54]) and other `Libero` suites (Appendix B.4), and results with using the OpenPi policy [3] on `Libero` suites other than 90 (Appendix B.5).

B.1 Action Chunking without Temporal Consistency

In Section 5, we showed that AC(n)-RDE matches the performance of AC(n), implying that, i.a., the temporal consistency property of action chunking (i.e., playing a *joint* distribution over a sequence of actions) is not required. Here, we provide an alternative simulation to show this. Specifically, we trained fifteen $\widehat{\pi}_{20}$ policies in both `Libero` and each individual `Robomimic` task, and then we compared the average performance of $\widehat{\pi}^{10}_{20}$ with that of the policy that plays in sequence $\pi^0_{\text{delay}}[\widehat{\pi}_{20}]$, $\pi^1_{\text{delay}}[\widehat{\pi}_{20}]$, $\dots$, $\pi^9_{\text{delay}}[\widehat{\pi}_{20}]$, and then restarts from $\pi^0_{\text{delay}}[\widehat{\pi}_{20}]$, $\pi^1_{\text{delay}}[\widehat{\pi}_{20}]$, $\dots$ (we call the latter policy AC(10)-Ordered). Intuitively, this latter policy plays delayed policies in the same order as the action chunking $\widehat{\pi}^{10}_{20}$ policy does; the only difference is that $\widehat{\pi}^{10}_{20}$ is temporally consistent, i.e., tries to match the joint action distribution exhibited by the expert, while AC(10)-Ordered plays each action in the chunk according to its marginal probability. As shown in Table 9, also this strategy matches the performance of AC(10) across all the considered simulation tasks.

B.2 Sample Efficiency in `Libero`

Here we aim to show explicitly that action chunking and delayed policies are more sample efficient than Markovian policies. To this aim, we focused on `Libero`, and chose three dataset sizes: 10, 25, 50. We then splitted the expert demonstration data for each task (overall 50), into two disjoint subsets of 10 trajectories, two disjoing subsets of 25 trajectories, and then all the 50 trajectories. We

Task	AC(10)	AC(10)-Ordered
`Libero 90`	89.2 ± 0.2	93.5 ± 0.1
`Robomimic square PH`	85.4 ± 0.5	85.2 ± 0.5
`Robomimic can PH`	97.2 ± 0.3	96.8 ± 0.3
`Robomimic transport PH`	12.6 ± 0.5	12.5 ± 0.5
`Robomimic tool hang PH`	75.2 ± 0.5	75.2 ± 0.6

Table 9: Another simulation about temporal consistency.

then trained two multitask $\widehat{\pi}_{20}$ policies on each of these five datasets, rolled them out for 100 rollouts in each task, computed the success rates and averaged these values per dataset size, obtaining the plot in Fig. 12. It is clear from such figure that the gap between AC(10) and AC(1) decreases as we increase the number of expert demonstrations, in particular moving from 0.39 for $N = 10$, to 0.31 for $N = 25$, up to 0.20 for $N = 50$. A similar pattern is observed also for Delay(6), further supporting the insights of our theoretical analysis.

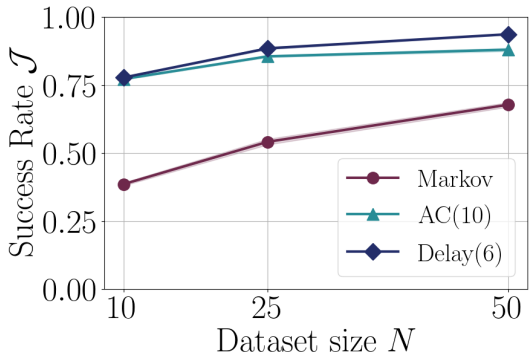


Figure 12: Comparison of Markovian, action chunking and delayed policies for increasing dataset sizes.

B.3 `Robomimic` Dynamics are Smooth

In Section 4.2, we made the theoretical assumption that the dynamics is smooth. This would definitely be true at least for the action translation components, if the underlying low-level controller was perfect, and could deploy the map $s_{h+1} = s_h + a_h$. In Figs. 13-15, we took the first expert trajectory in the non-normalized `Robomimic square` dataset, and plot the sequence of next state end effector position components along with the trivial predictions $s_h + a_h$, $s_{h-3} + a_{h-3}$ and $s_{h-6} + a_{h-6}$ as a function of the timestep. It is clear that, with rare exceptions mostly due to interaction with objects, the smooth map $s_{h-3} + a_{h-3}$ is a good predictor of this environment dynamics.

Intuitively, it makes sense that the true dynamics for delta control is something like $s_{h+1} = s_{h-3} + a_{h-3}$ instead of $s_{h+1} = s_h + a_h$. Indeed, note that the low-level controller in `Libero` and `Robomimic` operates at a 500Hz frequency, while our (high-level) policies takes actions with frequency 20Hz. Thus, the low-level controller has $500/20 = 25$ steps of 0.002 sec to try to get the robot to the desired/commanded position $s_{h-3} + a_{h-3}$ at time step $h - 3$; clearly, this amount of time might not be enough, and so the robot arm is able to get to position $s_{h-3} + a_{h-3}$ just some timesteps later, i.e., at s_{h+1} (of course as long as the subsequent actions a_{h-2}, a_{h-1}, a_h do not refer to completely different desired positions, which is not the case as humans cannot take completely uncorrelated actions at every h (i.e., every $1/20$Hz $= 0.05$sec), as they are not that fast).

B.4 Results on `Robomimic MH` and Other `Libero` Suites

For `Robomimic MH`, we trained 15 single task policies (and for ensemble average over three groups of 5 seeds). Results are shown in Table 10, and are analogous to Table 1.

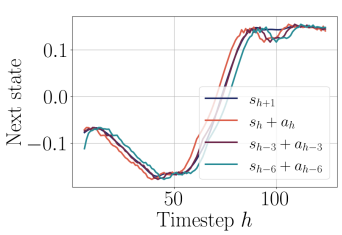


Figure 13: Action component for translation along the first axis.

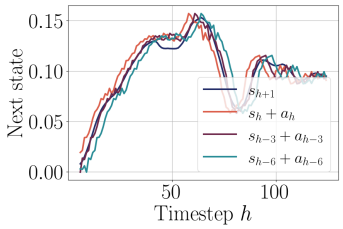


Figure 14: Action component for translation along the second axis.

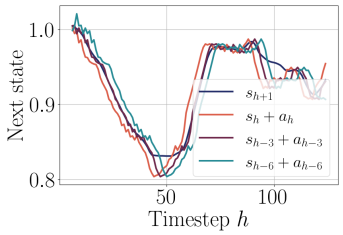


Figure 15: Action component for translation along the third axis.

Task	Markovian	AC(n)	Delay(n)	AC(n)-RDE	AC(n)-TE	AC(n)-Ens	Delay(n)-Ens
`Robomimic Can MH`	77.8 ± 0.7	93.9 ± 0.4	90.9 ± 0.5	94.3 ± 0.4	93.7 ± 0.4	**97.7** ± 0.8	**96.9** ± 0.1
`Robomimic Square MH`	43.4 ± 0.9	75.2 ± 0.7	63.4 ± 0.7	75.0 ± 0.6	67.7 ± 0.5	**81.1** ± 0.9	79.2 ± 0.6
`Robomimic Transport MH`	0.4 ± 0.1	4.7 ± 0.3	2.8 ± 0.2	4.9 ± 0.3	4.9 ± 0.3	**30.0** ± 0.9	24.5 ± 0.9

Table 10: Comparison of success rates across other `Robomimic MH`.

Algorithm	SR	t succ
Markovian	19.2 ± 1.5	350 ± 7
AC(5)	80.3 ± 1.1	288 ± 2
AC(10)	88.0 ± 0.4	271 ± 1
AC(15)	88.2 ± 0.8	264 ± 1
Delay(3)	78.1 ± 0.7	287 ± 2
Delay(6)	86.7 ± 1.6	271 ± 1
Delay(9)	84.1 ± 0.6	264 ± 1
Delay(13)	74.5 ± 1.3	264 ± 1
Delay(16)	64.1 ± 0.4	267 ± 0

Table 11: Comparison different chunk sizes and delays for `Libero-10`.

Algorithm	SR	t succ
Markovian	55.5 ± 1.4	126 ± 1
AC(5)	87.8 ± 1.5	112 ± 0
AC(10)	90.6 ± 0.3	109 ± 0
AC(15)	91.7 ± 0.7	107 ± 0
Delay(3)	86.9 ± 1.2	112 ± 0
Delay(6)	92.0 ± 0.9	109 ± 0
Delay(9)	88.6 ± 1.5	108 ± 0
Delay(13)	79.7 ± 0.3	109 ± 0
Delay(16)	72.7 ± 0.8	108 ± 0

Table 12: Comparison different chunk sizes and delays for `Libero-Spatial`.

Additionally, we also trained 3 multitask policies per `Libero` suite, and compared the different delays with different chunk sizes, just like we did in Figure 3 for `Libero-90`. See Tables 11-14.

B.5 Results with $\pi_{0.5}$

We used the open-source weights for the $\pi_{0.5}$ policy [33] fine-tuned on `Libero` available at `gs://openpi-assets/checkpoints/pi05_libero` (see repository `https://github.com/Physical-Intelligence/openpi` [31]). We ran 40 rollouts for each `Libero` task in every suite different than 90, as such policy has not been fine-tuned for this suite. Results are reported in Table 15. As you can see, they are compatible with those we obtained by training our own DDPM [64] policies. Note that this fine-tuned policy has an overall chunk size of 10. We also conjecture that the performance of the Markovian policy is much better here because, i.a., a pre-processing of the demonstration data removing pauses has been applied.

Algorithm	SR	t succ
Markovian	63.7 ± 2.1	137 ± 0
AC(5)	92.2 ± 0.8	119 ± 0
AC(10)	96.5 ± 0.3	113 ± 0
AC(15)	96.4 ± 0.9	112 ± 0
Delay(3)	92.5 ± 0.7	120 ± 1
Delay(6)	96.5 ± 0.2	114 ± 0
Delay(9)	93.9 ± 0.3	113 ± 0
Delay(13)	92.0 ± 0.3	112 ± 1
Delay(16)	86.9 ± 0.3	111 ± 0

Table 13: Comparison different chunk sizes and delays for `Libero-Goal`.

Algorithm	SR	t succ
Markovian	67.0 ± 0.9	160 ± 1
AC(5)	95.9 ± 0.5	142 ± 1
AC(10)	96.9 ± 0.6	137 ± 1
AC(15)	98.4 ± 0.3	138 ± 1
Delay(3)	94.7 ± 0.3	143 ± 1
Delay(6)	97.2 ± 0.3	138 ± 0
Delay(9)	96.6 ± 0.2	138 ± 0
Delay(13)	86.4 ± 0.6	142 ± 1
Delay(16)	74.6 ± 1.5	141 ± 1

Table 14: Comparison different chunk sizes and delays for `Libero-Object`.

Task	Markovian	AC(5)	AC(10)	Delay(3)	Delay(6)	Delay(9)	AC(10)-RDE	AC(10)-TE
`Libero-10`	84.0 ± 6.1	**91.2** ± 4.1	**93.8** ± 1.9	**90.5** ± 4.7	**91.0** ± 2.7	88.5 ± 1.9	**93.8** ± 2.5	**92.2** ± 1.4
`Libero-Spatial`	**96.7** ± 1.0	**96.7** ± 1.1	**97.8** ± 1.0	**96.8** ± 1.1	**98.0** ± 0.8	**95.0** ± 1.3	**97.0** ± 0.9	**98.0** ± 0.9
`Libero-Goal`	92.5 ± 2.6	**97.5** ± 1.1	97.5 ± 0.6	96.7 ± 1.2	**98.5** ± 0.4	97.8 ± 0.6	**99.0** ± 0.5	**98.5** ± 0.5
`Libero-Object`	91.2 ± 1.7	**98.5** ± 0.5	**99.2** ± 0.5	95.5 ± 1.1	97.8 ± 0.7	97.5 ± 0.9	**98.8** ± 0.6	**98.8** ± 0.6

Table 15: Comparison of success rates for $\pi_{0.5}$ [33] on `Libero`.

C Stationary Markov Policies are Not Expressive Enough for Non-Markov Experts

In this appendix, we aim to clarify that *stationary* Markovian learners are not expressive enough for perfectly replicating (potentially) non-Markovian human behavior, even in the limit of infinite data. We provide here a very simple example of environment to show that *stationary* Markovian policies are not expressive enough to achieve the same success rate of a *non-Markovian* expert, even if the environment is Markov.

To see this, take an MDP with n states labeled $s_1 \dots s_n$, where the action simply consists of moving either left or right ($s_n \to s_{n\pm1}$). Success is defined by reaching either extremal state 1 or n, while the initial state distribution is uniform over all states different from goal states. Assume the following simple non-Markovian demonstrator policy: select an action $\Delta = \pm 1$ uniformly at random at the first timestep and apply the same action for the rest of the episode. If we fit a Markov policy $\pi(a|s)$ to this dataset, we exhibit a random walk. While the demonstration trajectories all succeed in at most n steps, the Markovian behavior-cloning policy π succeeds with $p < 1$ (Fig. 16). The discrepancy with standard literature [69] comes from the assumption of a finite horizon for success.

Note that if we were to simply condition π on the step in the episode, we could perfectly match the state-action distribution of the demonstrator (though the trajectory distribution would differ) [69]—the key piece in this example is that we do not condition π on the timestep. While timestep conditioning would resolve the challenge here, in practice, it is non-standard to condition on timestep so we would not expect policies in practice to necessarily express non-Markovian demonstrators. Being more non-Markovian than a Markovian policy, action chunking policies can provide more

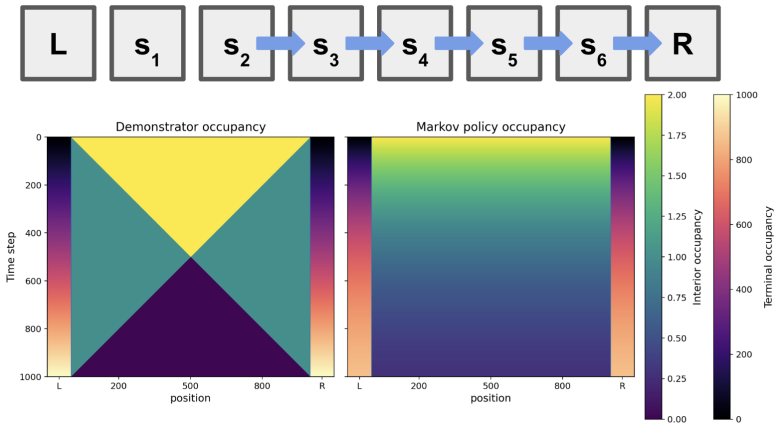


Figure 16: Illustrative example of random-walk behavior on a 1D line-world with $n = 1000$ states. While the (non-Markovian) demonstrator moves constantly left or constantly right and arrives at either the left or right terminating states with 100% probability before the chosen horizon, marginalizing this into a Markovian policy (i.e. via BC) results in significantly different state occupancy and in the limit of large N only achieves $1 - e^{-2}$ success rate at time $T = N$.

favorable rates here (intuitively, action chunking policies induce random walks with longer steps, and so are faster to reach the extrema of the interval).

D Predicting Actions Based on Delayed States Provably Mitigates Compounding Error

Throughout, for some metrics $d_{\mathcal{S}}$ and $d_{\mathcal{A}}$ in, respectively, the state $\mathcal{S}$ and action $\mathcal{A}$ spaces, we will make the following assumptions.

Assumption D.1 (Dynamics is smooth in the state)**.** *The dynamics P is 1-Lipschitz in the state. Formally:*

$$d_{\mathcal{S}}\big(P(s,a), P(s',a)\big) \leqslant d_{\mathcal{S}}\big(s,s'\big) \qquad \forall s,s' \in \mathcal{S}, \forall a \in \mathcal{A}.$$

Assumption D.2 (Dynamics is smooth in the action)**.** *The dynamics P is 1-Lipschitz in the action. Formally:*

$$d_{\mathcal{S}}\big(P(s,a), P(s,a')\big) \leqslant d_{\mathcal{A}}\big(a,a'\big) \qquad \forall s \in \mathcal{S}, \forall a,a' \in \mathcal{A}.$$

Assumption D.3 (Reward is smooth)**.** *The reward r is 1-Lipschitz in the state. Formally:*

$$\big|r(s) - r(s')\big| \leqslant d_{\mathcal{S}}\big(s,s'\big) \qquad \forall s,s' \in \mathcal{S}.$$

Assumption D.4 (Learned policies are smooth)**.** *The learned policy $\widehat{\pi}$ is 1-Lipschitz in the state. Formally, if $\widehat{\pi}$ is a Markovian or action chunked policy $\widehat{\pi}_k$, then:*

$$W_1\big([\widehat{\pi}_k(\cdot|s)]_i, [\widehat{\pi}_k(\cdot|s')]_i\big) \leqslant d_{\mathcal{S}}(s,s') \qquad \forall s,s' \in \mathcal{S}, \forall i \in [k],$$

where index i denotes the i-th action in the chunk outputted by $\widehat{\pi}_k$. If instead $\widehat{\pi}$ is a (potentially induced) delayed policy π^d_{delay}, then:

$$W_1\big(\pi^d_{\text{delay}}(\cdot|s), \pi^d_{\text{delay}}(\cdot|s')\big) \leqslant d_{\mathcal{S}}(s,s') \qquad \forall s,s' \in \mathcal{S}.$$

To be precise, we would like to mention the fact that, in general, it might not be possible to construct a "smooth" learner as prescribed by Assumption D.4, which also has small generalization error w.r.t. the expert's policy, as assumed by Theorem 2. Thus, to assume this, we are implicitly assuming that Assumption D.4 holds also for the expert's policy.

We would like to remark that, in the following proofs, we will *not* assume a binary success/failure reward, but we will consider a smooth reward as prescribed by Assumption D.3. As such, our episodes will not terminate when any non-zero reward is received, but they will always terminate after H timesteps.

D.1 Upper Bound for Delayed Policy

Throughout, we define $s_i := s_0$ for $i < 0$.

Proof of Theorem 2. We begin by proving the result for the delayed policy, that is, let our learner $\widehat{\pi}$ be a delayed policy π^k_{delay} for some $k \geqslant 0$ (note that, for $k = 0$, we are effectively studying a Markovian policy). Formally, as the behavior of policy π^k_{delay} is undefined for $t < k$, we assume that we are going to play the chunked policy there (i.e., we use some $\widehat{\pi}$ policy) and assume that the same upper bound to the generalization error ϵ holds for all the actions there.

Delayed policy.

By definition, for any policy π, we have that $\mathcal{J}(\pi) = \mathbb{E}^{\pi}[\sum_{t=0}^{H-1} r(s_t)]$. Therefore, by triangle inequality, we can write:

$$\mathcal{J}(\pi_{\text{demo}}) - \mathcal{J}(\widehat{\pi}) \leqslant \sum_{t=0}^{H-1} \left|\mathbb{E}^{\pi_{\text{demo}}}[r(s_t)] - \mathbb{E}^{\widehat{\pi}}[r(s_t)]\right|,$$

where we would like to recall that symbol $\mathbb{E}^{\pi}$ denotes the trajectory distribution of policy π, so that $\mathbb{E}^{\pi}[r(s_t)] = \sum_{\tau} \mathbb{P}^{\pi}(\tau) r(s_t)$, where τ denotes some trajectory of length h, $\mathbb{P}^{\pi}(\cdot)$ is the induced probability measure over trajectories by π, and s_t denotes the state at timestep t of trajectory τ.

Consider some $t \geqslant 1$. Let $s_t^{\pi_{\text{demo}}}$ denote the state generated in a trajectory induced by playing π_{demo} and $s_t^{\widehat{\pi}}$ the state generated by playing $\widehat{\pi}$. Similarly define $a_t^{\pi_{\text{demo}}}$ and $a_t^{\widehat{\pi}}$ the corresponding actions. Since we have assumed that r is 1-Lipschitz, by Assumption D.3 and Jensen's inequality, we have

$$\left|\mathbb{E}^{\pi_{\text{demo}}}[r(s_t)] - \mathbb{E}^{\widehat{\pi}}[r(s_t)]\right| = \left|\mathbb{E}[r(s_t^{\pi_{\text{demo}}}) - r(s_t^{\widehat{\pi}})]\right| \leqslant \mathbb{E}[d_{\mathcal{S}}(s_t^{\pi_{\text{demo}}}, s_t^{\widehat{\pi}})],$$

where $\mathbb{E}$ is the expectation w.r.t. some coupling between the trajectory distributions $\mathbb{P}^{\pi_{\text{demo}}}$ and $\mathbb{P}^{\widehat{\pi}}$ up to timestep t.

By the definition of the dynamics, we have $s_t^{\pi_{\text{demo}}} = P(s_{t-1}^{\pi_{\text{demo}}}, a_{t-1}^{\pi_{\text{demo}}})$ and $s_t^{\widehat{\pi}} = P(s_{t-1}^{\widehat{\pi}}, a_{t-1}^{\widehat{\pi}})$. Since we have assumed that P is 1-Lipschitz in both state and action, then by Assumptions D.1-D.2 and by the triangle inequality we have

$$\mathbb{E}[d_{\mathcal{S}}(s_t^{\pi_{\text{demo}}}, s_t^{\widehat{\pi}})] \leqslant \mathbb{E}[d_{\mathcal{S}}(s_{t-1}^{\pi_{\text{demo}}}, s_{t-1}^{\widehat{\pi}})] + \mathbb{E}[d_{\mathcal{A}}(a_{t-1}^{\pi_{\text{demo}}}, a_{t-1}^{\widehat{\pi}})].$$

By assumption, $a_{t-1}^{\pi_{\text{demo}}} \sim \pi_{\text{demo}}(\cdot \mid \boldsymbol{h}_{t-1}^{\pi_{\text{demo}}})$ (for $\boldsymbol{h}^{\pi_{\text{demo}}}$ the history generated under π_{demo}) and $a_{t-1}^{\widehat{\pi}} \sim \widehat{\pi}(\cdot \mid s_{t-k-1}^{\widehat{\pi}})$. By the triangle inequality, we have

$$\mathbb{E}[d_{\mathcal{A}}(a_{t-1}^{\pi_{\text{demo}}}, a_{t-1}^{\widehat{\pi}})] \leqslant \mathbb{E}[\mathbb{E}_{a\sim\widehat{\pi}(\cdot|s_{t-k-1}^{\pi_{\text{demo}}})}[d_{\mathcal{A}}(a_{t-1}^{\pi_{\text{demo}}}, a)]] + \mathbb{E}[\mathbb{E}_{a\sim\widehat{\pi}(\cdot|s_{t-k-1}^{\pi_{\text{demo}}})}[d_{\mathcal{A}}(a, a_{t-1}^{\widehat{\pi}})]]. \quad (4)$$

Considering the optimal coupling between $a_{t-1}^{\pi_{\text{demo}}} \mid \mathcal{F}_{t-1}$, for $\mathcal{F}_{t-1}$ the filtration up to step $t-1$, and $a \sim \widehat{\pi}(\cdot \mid s_{t-k-1}^{\pi_{\text{demo}}})$, we can bound

$$\mathbb{E}[\mathbb{E}_{a\sim\widehat{\pi}(\cdot|s_{t-k-1}^{\pi_{\text{demo}}})}[d_{\mathcal{A}}(a_{t-1}^{\pi_{\text{demo}}}, a)]] \leqslant \mathbb{E}[W_1(\pi_{\text{demo}}(\cdot \mid \boldsymbol{h}_{t-1}^{\pi_{\text{demo}}}), \widehat{\pi}(\cdot \mid s_{t-k-1}^{\pi_{\text{demo}}}))]] \leqslant \epsilon,$$

where the last inequality follows by assumption. Note that

$$\begin{aligned}
\mathbb{E}[\mathbb{E}_{a\sim\widehat{\pi}(\cdot|s_{t-k-1}^{\pi_{\text{demo}}})}[d_{\mathcal{A}}(a, a_{t-1}^{\widehat{\pi}})]] &= \mathbb{E}[\mathbb{E}_{a\sim\widehat{\pi}(\cdot|s_{t-k-1}^{\pi_{\text{demo}}}), a'\sim\widehat{\pi}(\cdot|s_{t-k-1}^{\widehat{\pi}})}[d_{\mathcal{A}}(a, a')]] \\
&\leqslant \mathbb{E}[W_1(\widehat{\pi}(\cdot \mid s_{t-k-1}^{\pi_{\text{demo}}}), \widehat{\pi}(\cdot \mid s_{t-k-1}^{\widehat{\pi}}))] \\
&\leqslant \mathbb{E}[d_{\mathcal{S}}(s_{t-k-1}^{\pi_{\text{demo}}}, s_{t-k-1}^{\widehat{\pi}})]
\end{aligned}$$

where the first inequality follows taking the optimal coupling between a and a', and the second inequality follows from Assumption D.4. Altogether then, we have shown that

$$\mathbb{E}[d_{\mathcal{S}}(s_t^{\pi_{\text{demo}}}, s_t^{\widehat{\pi}})] \leqslant \mathbb{E}[d_{\mathcal{S}}(s_{t-1}^{\pi_{\text{demo}}}, s_{t-1}^{\widehat{\pi}})] + \mathbb{E}[d_{\mathcal{S}}(s_{t-k-1}^{\pi_{\text{demo}}}, s_{t-k-1}^{\widehat{\pi}})] + \epsilon.$$

By Lemma D.1, we can bound

$$\mathbb{E}[d_{\mathcal{S}}(s_t^{\pi_{\text{demo}}}, s_t^{\widehat{\pi}})] \leqslant 2(k+1)^{\lceil t/k \rceil} \cdot \epsilon.$$

The result then follows by summing over t and using geometric sums:

$$\begin{aligned}\sum_{t=0}^{H-1}(k+1)^{\lceil t/k\rceil} &= 1+\sum_{m=1}^{k}(k+1)+\sum_{m=k+1}^{2k}(k+1)^2+\dots\\ &\leqslant 1+\sum_{m=1}^{\lceil (H-1)/k\rceil}k(k+1)^m\\ &\leqslant 1+k\sum_{m=0}^{\lceil (H-1)/k\rceil}(k+1)^m\\ &= 1+k\frac{(k+1)^{\lceil (H-1)/k\rceil+1}-1}{(k+1)-1}\\ &= (k+1)^{\lceil (H-1)/k\rceil+1}.\end{aligned}$$

Action chunking (sketch).

Note that the proof applied above can be easily modified for using any sequence of delayed policies with different delays. In particular, the key difference is just that, in Eq. (4), instead of using $\widehat{\pi}(\cdot \mid s^{\pi_{\text{demo}}}_{t-k-1})$, we should use the desired delay d, i.e., $\widehat{\pi}(\cdot \mid s^{\pi_{\text{demo}}}_{t-d})$. Then all other passages follow analogously. In case of action chunking, we play a sequence of delayed policies in ascending order by delay, up to reach the maximum delay $k-1$ (corresponding to chunk size k), before restarting. So, if we set our learner $\widehat{\pi}$ be a chunked policy $\widehat{\pi}_k$, then it is easy to observe that we end up to the following recursion, for any timestep $t = kx + r$, where $x := \lfloor t/k \rfloor$ and $r := t - k\lfloor t/k \rfloor$:

$$\mathbb{E}[d_{\mathcal{S}}(s^{\pi_{\text{demo}}}_t, s^{\widehat{\pi}}_t)] \leqslant \mathbb{E}[d_{\mathcal{S}}(s^{\pi_{\text{demo}}}_{kx+r-1}, s^{\widehat{\pi}}_{kx+r-1})] + \mathbb{E}[d_{\mathcal{S}}(s^{\pi_{\text{demo}}}_n, s^{\widehat{\pi}}_n)] + \epsilon,$$

where $n = kx$ if $r > 0$ and $n = k(x-1)$ otherwise.

Thanks to Lemma D.2, we obtain:

$$\mathbb{E}[d_{\mathcal{S}}(s^{\pi_{\text{demo}}}_t, s^{\widehat{\pi}}_t)] \leqslant (k+1)^{\lfloor t/k\rfloor+1}\cdot\epsilon.$$

The result then follows by summing over h and using geometric sums:

$$\begin{aligned}\sum_{t=0}^{H-1}(k+1)^{\lfloor t/k\rfloor+1} &= \sum_{m=0}^{k-1}(k+1)+\sum_{m=k}^{2k-1}(k+1)^2+\dots\\ &\leqslant \sum_{m=0}^{\lfloor (H-1)/k\rfloor+1}(k+1)^m\\ &= \frac{(k+1)^{\lfloor (H-1)/k\rfloor+2}-1}{(k+1)-1}\\ &\leqslant 2(k+1)^{\lfloor (H-1)/k\rfloor+1}.\end{aligned}$$

This concludes the proof.[6] □

Lemma D.1. *Consider a sequence $\{\delta_t\}_t$ with $\delta_t = 0$ for $t \leqslant 0$, and that satisfies*

$$\delta_{t+1} \leqslant \delta_t + \delta_{t-k} + \epsilon, \tag{5}$$

for some constant $\epsilon \geqslant 0$. Then we can bound $\delta_t \leqslant 2(k+1)^{\lceil t/k\rceil}\cdot\epsilon$.

[6] Note that the same rate would be obtained even if we randomized or used any other kind of order for playing delayed policies. Moreover, note that the upper bound for the delayed policies is not tight, and could probably obtain a bound where the contribution of the k factor is doubled, since a delayed policy with delay k roughly corresponds to an action chunking policy with chunk size $2k$.

Proof. Define $\{\bar{\delta}_t\}_t$ as the sequence which satisfies $\bar{\delta}_t = 0$ for $t \leqslant 0$ and $\bar{\delta}_{t+1} = \bar{\delta}_t + \bar{\delta}_{t-k} + \epsilon$. By construction, we then have that $\{\bar{\delta}_t\}_t$ satisfies (5), and $\bar{\delta}_t \geqslant \delta_t \ \forall t$ for any other sequence satisfying (5).[7] Note also that $\bar{\delta}_{t+1} \geqslant \bar{\delta}_t$.[8]

Recursing backwards, for any $k \geqslant 1$, we have

$$\begin{aligned}
\bar{\delta}_t &= \bar{\delta}_{t-1} + \bar{\delta}_{t-k-1} + \epsilon \\
&= \bar{\delta}_{t-2} + \bar{\delta}_{t-k-1} + \bar{\delta}_{t-k-2} + 2\epsilon \\
&\vdots \\
&= \bar{\delta}_{t-k} + \sum_{i=1}^{k} \bar{\delta}_{t-k-i} + k\epsilon \\
&= \sum_{i=0}^{k} \bar{\delta}_{t-k-i} + k\epsilon.
\end{aligned}$$

Since $\bar{\delta}_{t+1} \geqslant \bar{\delta}_t$ for any t, we can bound this as

$$\bar{\delta}_t \leqslant (k+1)\bar{\delta}_{t-k} + k\epsilon.$$

Recursing this backwards, we get

$$\begin{aligned}
\bar{\delta}_t &\leqslant (k+1)\bar{\delta}_{t-k} + (k+1)\epsilon \leqslant (k+1)^2\bar{\delta}_{t-2k} + (k+1)^2\epsilon + (k+1)\epsilon \\
&\leqslant \ldots \leqslant \sum_{i=1}^{\lceil t/k \rceil} (k+1)^i \epsilon,
\end{aligned}$$

where we note that $x = \lceil t/k \rceil$ is the minimum integer value guaranteeing $t - xk \leqslant 0$.

Lastly, we use the bound for geometric sums:

$$\begin{aligned}
\bar{\delta}_t &\leqslant \sum_{i=1}^{\lceil t/k \rceil} (k+1)^i \epsilon \leqslant \epsilon \sum_{i=0}^{\lceil t/k \rceil} (k+1)^i = \epsilon \frac{(k+1)^{\lceil t/k \rceil + 1} - 1}{k} \\
&\leqslant \epsilon \frac{k+1}{k} (k+1)^{\lceil t/k \rceil} \leqslant 2\epsilon (k+1)^{\lceil t/k \rceil}.
\end{aligned}$$

This completes the proof. □

Lemma D.2. *Consider a sequence $\{\delta_t\}_t$ with $\delta_t = 0$ for $t = 0$, and that satisfies, for any timestep $t = kx + r$, where $x := \lfloor t/k \rfloor$ and $r := t - k\lfloor t/k \rfloor$*

$$\delta_{kx+r} \leqslant \delta_{kx+r-1} + \delta_n + \epsilon, \tag{6}$$

where $n = kx$ if $r > 0$ and $n = k(x-1)$ otherwise, for some constant $\epsilon \geqslant 0$. Then we can bound $\delta_t \leqslant (k+1)^{\lfloor t/k \rfloor + 1} \cdot \epsilon$.

Proof. Define $\{\bar{\delta}_t\}_t$ as the sequence which satisfies $\bar{\delta}_t = 0$ for $t = 0$ and $\bar{\delta}_{kx+r} = \bar{\delta}_{kx+r-1} + \bar{\delta}_n + \epsilon$. By construction, we then have that $\{\bar{\delta}_t\}_t$ satisfies (6), and $\bar{\delta}_t \geqslant \delta_t \ \forall t$ for any other sequence satisfying (6).[9]

[7] Formally, this can be shown by induction. At $t \leqslant 0$, we have $\bar{\delta}_t = \delta_t = 0$ by definition. Then, let us make the inductive hypothesis that $\bar{\delta}_t \geqslant \delta_t \ \forall t \leqslant \bar{t}$. Then, at $t = \bar{t}$, we have that: $\bar{\delta}_{t+1} := \bar{\delta}_t + \bar{\delta}_{t-k} + \epsilon \geqslant \delta_t + \delta_{t-k} + \epsilon \geqslant \delta_{t+1}$, where we used first the definition of $\{\bar{\delta}_t\}_t$, then the inductive hypothesis, and finally the definition of $\{\delta_t\}_t$.

[8] This follows easily from the definition and the fact that all terms are non-negative: $\bar{\delta}_{t+1} = \bar{\delta}_t + \bar{\delta}_{t-k} + \epsilon \geqslant \bar{\delta}_t + 0 + 0 \geqslant \bar{\delta}_t$.

[9] This can be proved analogously to the proof of Lemma D.1.

Recursing backwards, for any $k \geqslant 1$, we have

$$\begin{aligned}\bar{\delta}_{kx+r} &\leqslant \bar{\delta}_{kx+k} \\ &= \bar{\delta}_{kx+(k-1)} + \bar{\delta}_{kx} + \epsilon \\ &= \bar{\delta}_{kx+(k-2)} + 2\bar{\delta}_{kx} + 2\epsilon \\ &\vdots \\ &= \bar{\delta}_{kx} + k\bar{\delta}_{kx} + k\epsilon \\ &= (k+1)\bar{\delta}_{kx} + k\epsilon.\end{aligned}$$

Recursing this backwards, we get, for $t = kx + r$:

$$\begin{aligned}\bar{\delta}_t &\leqslant (k+1)\bar{\delta}_{kx} + k\epsilon \leqslant (k+1)^2\bar{\delta}_{k(x-1)} + k(k+1)\epsilon + k\epsilon \\ &\leqslant \ldots \leqslant k\sum_{i=0}^{\lfloor t/k \rfloor}(k+1)^i\epsilon.\end{aligned}$$

Lastly, we use the bound for geometric sums:

$$\begin{aligned}\bar{\delta}_t &\leqslant k\epsilon\sum_{i=0}^{\lfloor t/k \rfloor}(k+1)^i = k\epsilon\frac{(k+1)^{\lfloor t/k \rfloor+1}-1}{k} \\ &\leqslant \epsilon(k+1)^{\lfloor t/k \rfloor+1}.\end{aligned}$$

This completes the proof. □

D.2 Lower Bound for Markovian Policies

In this appendix, we will use symbol $[x_1, x_2, \ldots, x_d]$ to denote a d-dimensional vector, and symbol $[x]_i$ to denote the ith component of vector x.

Proof of Theorem 1. Let the environment $\mathcal{M}$ be such that: $\mathcal{S} = \mathcal{A} = \mathbb{R}^2$, $P_0([0,0]) = 1$, and $P(s,a) = s + a$ for all s, a. Let $r(s) = 1 - |[s]_1|$. Let $d_{\mathcal{S}}$ and $d_{\mathcal{A}}$ be the metrics induced by any norm $\|\cdot\|$. Then, note that the Lipschitz assumption on P (that is, Assumptions D.1-D.2) holds since:

$$\begin{aligned}\|P(s,a) - P(s',a)\| &= \|(s+a) - (s'+a)\| = \|s - s'\|, \\ \|P(s,a) - P(s,a')\| &= \|(s+a) - (s+a')\| = \|a - a'\|.\end{aligned}$$

Furthermore, the Lipschitz assumption on r (that is, Assumption D.3) holds since

$$|r(s) - r(s')| = ||[s]_1| - |[s']_1|| \leqslant |[s]_1 - [s']_1| \leqslant \|s - s'\|_2.$$

Let the expert's policy $\pi_{\text{demo}} : \mathcal{S} \to \mathcal{A}$ be $\pi_{\text{demo}}(s) = [0,1]\ \forall s$. This induces a support: $\text{supp}(\pi_{\text{demo}}) = \{[0,k] \in \mathcal{S} : k \in \mathbb{N}\}$.

Let $\widehat{\pi} : \mathcal{S} \to \mathcal{A}$ be: $\widehat{\pi}([s^1, s^2]) = [\epsilon + \min_{s' \in \text{supp}(\pi_{\text{demo}})} \|s - s'\|_2, 1]$, and note that it is 1-Lipschitz (i.e., satisfies Assumption D.4) since:

$$\begin{aligned}\|\widehat{\pi}(s) - \widehat{\pi}(s'')\|_2 &= \Big|\min_{s' \in \text{supp}(\pi_{\text{demo}})}\|s - s'\|_2 - \min_{s' \in \text{supp}(\pi_{\text{demo}})}\|s'' - s'\|_2\Big| \\ &\leqslant \|s - s''\|_2,\end{aligned}$$

where we used that the map $d_{\mathcal{X}}(u) \coloneqq \min_{x \in \mathcal{X}} \|u - x\|_2$ is 1-Lipschitz for any $\mathcal{X} \subseteq \mathbb{R}^2$. Moreover, note that this $\widehat{\pi}$ satisfies also the (last) hypothesis of this theorem, which is that of having a bounded generalization error, since:

$$\mathbb{E}^{\pi_{\text{demo}}}[W_1(\pi_{\text{demo}}(s_t), \widehat{\pi}(s_t))] = \mathbb{E}^{\pi_{\text{demo}}}[\|\pi_{\text{demo}}(s_t) - \widehat{\pi}(s_t)\|_2] = \epsilon.$$

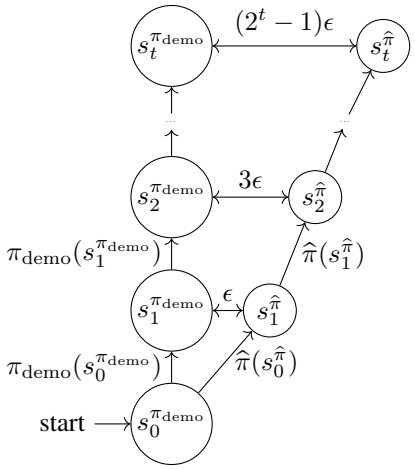


Figure 17: Construction for lower bound on performance of Markovian policy (Theorem 1)

As shown also in Figure 17, it is easy to see that the sequence of expert states is $s_0^{\pi_{\text{demo}}} = [0,0], s_1^{\pi_{\text{demo}}} = [0,1], s_2^{\pi_{\text{demo}}} = [0,2], \dots, s_t^{\pi_{\text{demo}}} = [0,t]$, while the sequence of states followed by $\widehat{\pi}$ starts at $s_0^{\widehat{\pi}} = [0,0]$ and grows based on $s_{t+1}^{\widehat{\pi}} = s_t^{\widehat{\pi}} + \widehat{\pi}(s_t^{\widehat{\pi}})$ as:

$$\begin{cases} s_0^{\widehat{\pi}} = [0,0], \\ s_1^{\widehat{\pi}} = [0,0] + [\epsilon + 0, 1] = [\epsilon, 1], \\ s_2^{\widehat{\pi}} = [\epsilon,1] + [\epsilon+\epsilon, 1] = [3\epsilon, 2], \\ s_3^{\widehat{\pi}} = [3\epsilon,2] + [\epsilon + 3\epsilon, 1] = [7\epsilon, 3], \\ \dots \end{cases}.$$

Thus:

$$[s_t^{\widehat{\pi}}]_1 = \epsilon + 2[s_{t-1}^{\widehat{\pi}}]_1 = \epsilon(2^t - 1).$$

Note that $[s_t^{\pi_{\text{demo}}}]_1 = 0$, so that $\mathcal{J}(\pi_{\text{demo}}) = H$. However,

$$\mathcal{J}(\widehat{\pi}) = \sum_{t=0}^{H-1} r(s_t^{\widehat{\pi}}) = \sum_{t=0}^{H-1} (1 - [s_t^{\widehat{\pi}}]_1) = H - \epsilon \cdot \sum_{t=0}^{H-1} (2^h - 1) = H - \epsilon \cdot (2^H - H - 1).$$

This completes the proof. □

D.3 Lower Bound for Action-Chunked Policies

Theorem 3. *There exists an environment satisfying the above assumptions, a Markov demonstrator π_{demo}, and an action chunking learner $\widehat{\pi}_k$ satisfying $\max_h \mathbb{E}^{\pi_{\text{demo}}}[\max_{i\in[k]} W_1(\pi_{\text{demo}}(s_{t+i-1}), [\widehat{\pi}_k(s_t)]_i)] \leqslant \epsilon$, for $W_1(\cdot,\cdot)$ the Wasserstein-1 metric, such that $\mathcal{J}(\pi_{\text{demo}}) \geqslant \mathcal{J}(\widehat{\pi}) + \Omega((k+1)^{k/H} \cdot \epsilon)$.*

Proof. We consider the same environment, reward, and expert as in Theorem 1.

Let $[\widehat{\pi}([s^1, s^2])]_i = [\epsilon + \min_{s' \in \text{supp}(\pi_{\text{demo}})} \|s - s'\|_2, 1]$, and note that it is 1-Lipschitz (i.e., satisfies Assumption D.4) since, for any action in the chunk $i \in [k]$:

$$\begin{aligned} \|[\widehat{\pi}(s)]_i - [\widehat{\pi}(s'')]_i\|_2 &= \Big| \min_{s' \in \text{supp}(\pi_{\text{demo}})} \|s - s'\|_2 - \min_{s' \in \text{supp}(\pi_{\text{demo}})} \|s'' - s'\|_2 \Big| \\ &\leqslant \|s - s''\|_2, \end{aligned}$$

where we used that the map $d_{\mathcal{X}}(u) := \min_{x \in \mathcal{X}} \|u - x\|_2$ is 1-Lipschitz for any $\mathcal{X} \subseteq \mathbb{R}^2$. Moreover, note that this $\widehat{\pi}$ satisfies the assumption of uniform bound on the generalization error along the chunk, since:

$$\mathbb{E}^{\pi_{\text{demo}}}[W_1(\pi_{\text{demo}}(s_t), \widehat{\pi}_i(s_{t-i}))] = \mathbb{E}^{\pi_{\text{demo}}}[\|\pi_{\text{demo}}(s_t) - \widehat{\pi}_i(s_{t-i})\|_2] = \epsilon.$$

With this choice of $\widehat{\pi}$, since we start at $s_0^{\pi_{\text{demo}}} = [0, 0]$, for the first k timesteps we play $\widehat{\pi}_i(s_0^{\pi_{\text{demo}}}) = [\epsilon, 1]$, so that $s_k^{\widehat{\pi}} = [k\epsilon, k]$. Then, we have $\widehat{\pi}(s_k^{\widehat{\pi}}) = [k\epsilon+\epsilon, 1]$, so that $\widehat{\pi}(s_{2k}^{\widehat{\pi}}) = [k\epsilon+k(k\epsilon+\epsilon), 2k]$. More generally, we see that

$$s_{ik}^{\widehat{\pi}} = [(k+1)^i \cdot \epsilon, ik]$$

and, more generally, $s_t^{\widehat{\pi}} = [(k+1)^{i_t} \cdot \epsilon + (t - i_t k) \cdot ((k+1)^{i_t} \cdot \epsilon + \epsilon), t]$, where $i_t := \lfloor t/k \rfloor$ denotes the closest action chunk boundary index. It follows that

$$\begin{aligned}
\mathcal{J}(\widehat{\pi}) &= H - \epsilon \cdot \sum_{i=0}^{H/k-1} \sum_{j=0}^{k} \left[(k+1)^i + j \cdot ((k+1)^i + 1)\right] \\
&= H - \epsilon \cdot \left[\frac{k+1}{k} \cdot (k+1)^{H/k} - \frac{k+1}{k} + H\right].
\end{aligned}$$

This proves the result. □

E Individualized Result Plots

In this section we report the plots of Figs. 2 (see Figs. 18-20), 3 (see Figs. 21-23), 9 (see Fig. 24), 8 (see Fig. 25), individually for each task instead of averaging over all tasks.

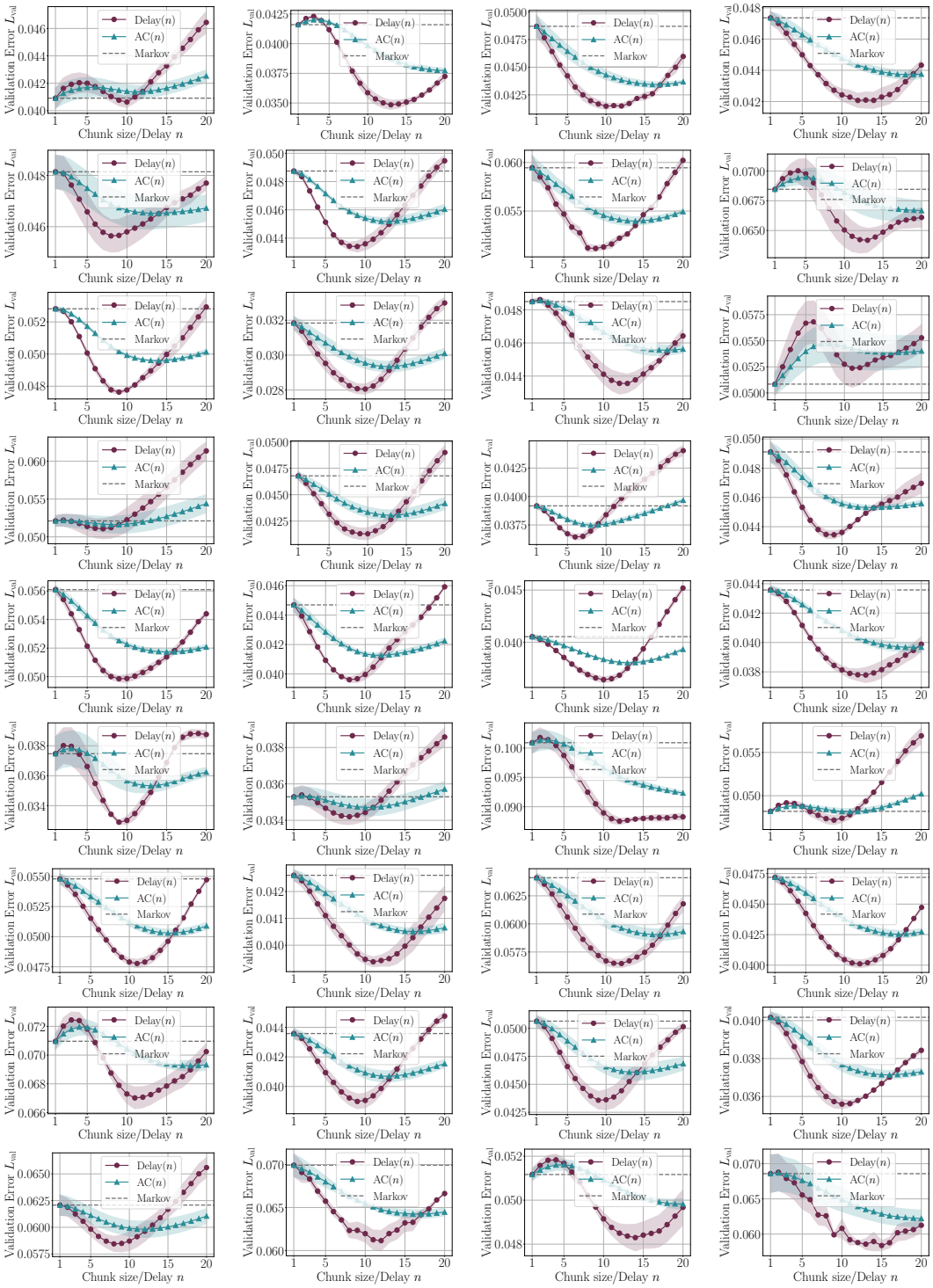


Figure 18: Validation loss for each `Libero` task from 0 to 35 (corresponding to Fig. 2), part 1.

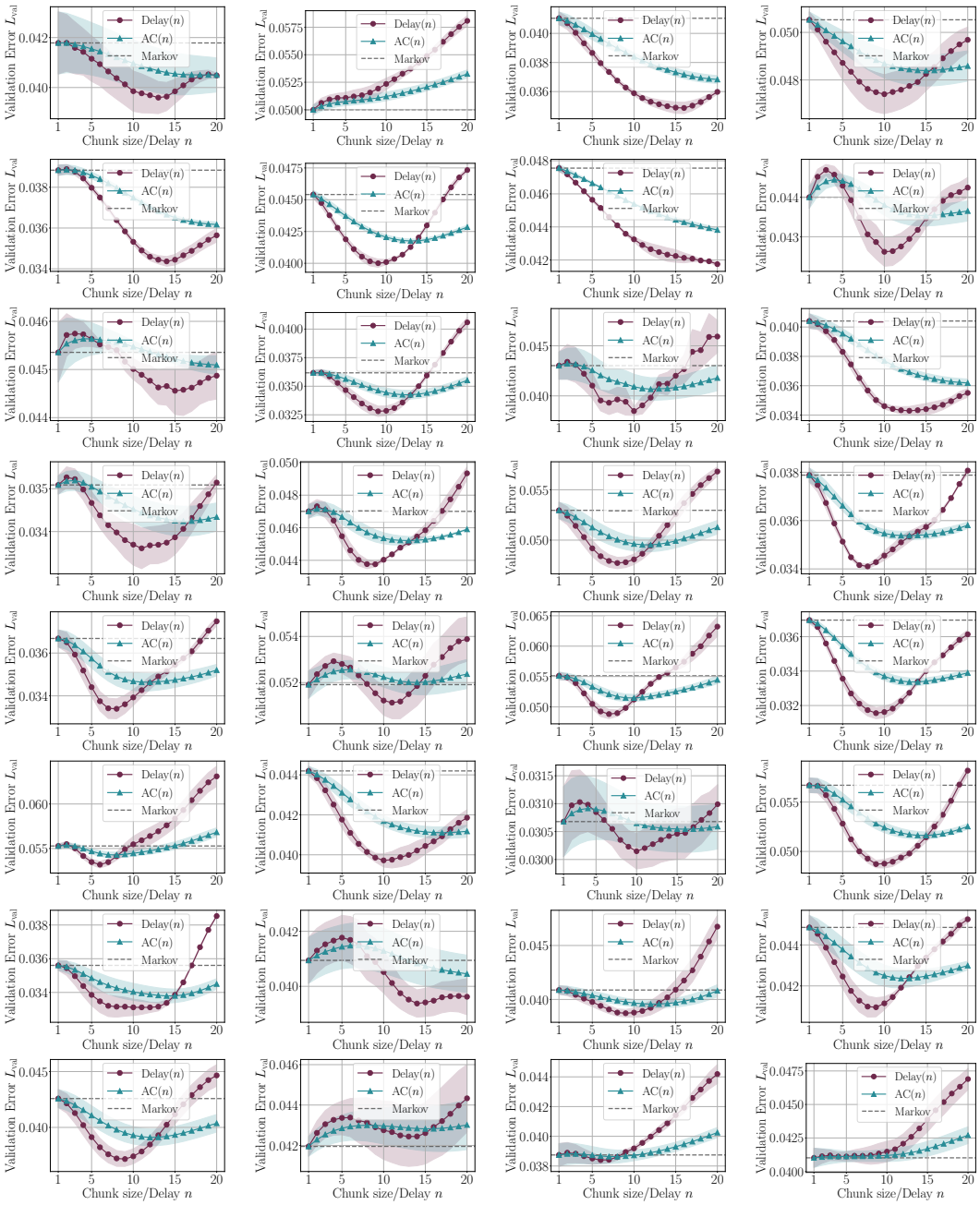


Figure 19: Validation loss for each `Libero` task from 36 to 67 (corresponding to Fig. 2), part 2.

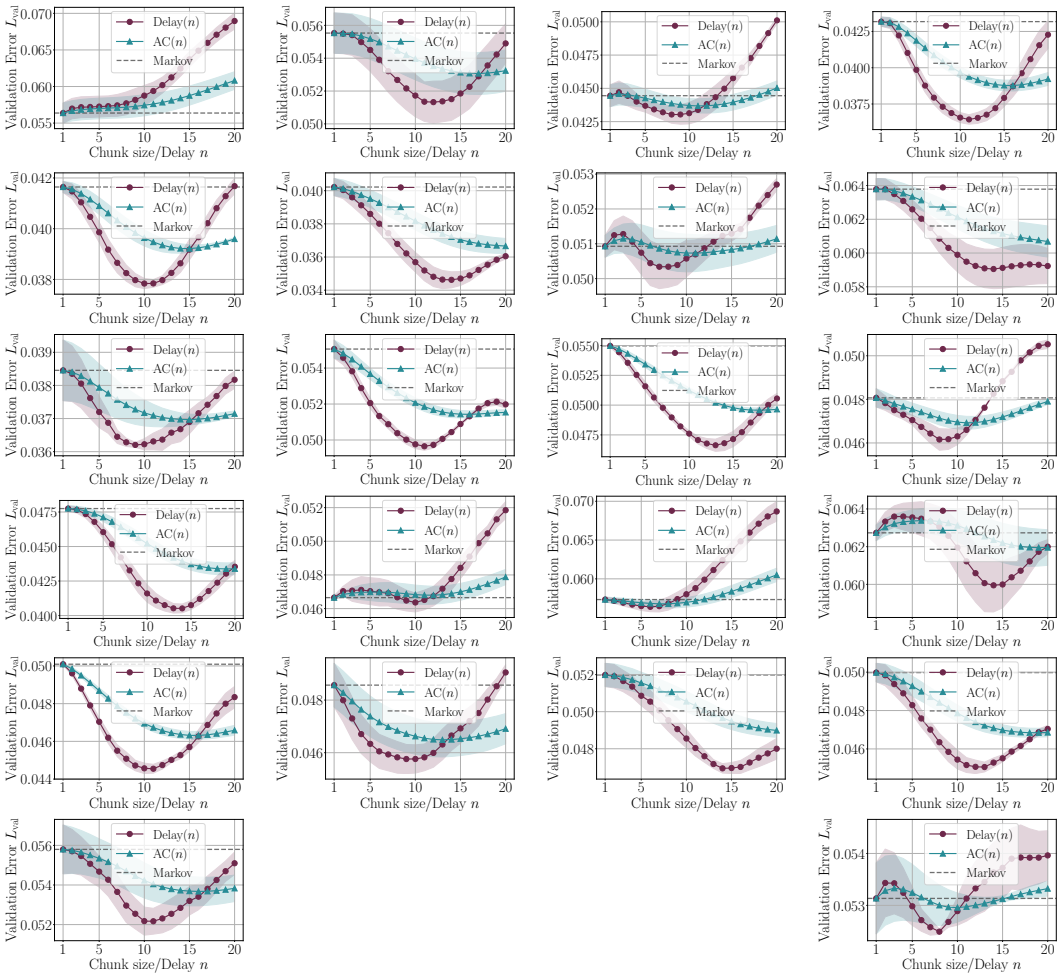


Figure 20: Validation loss for each `Libero` task from 68 to 89 (corresponding to Fig. 2), part 3.

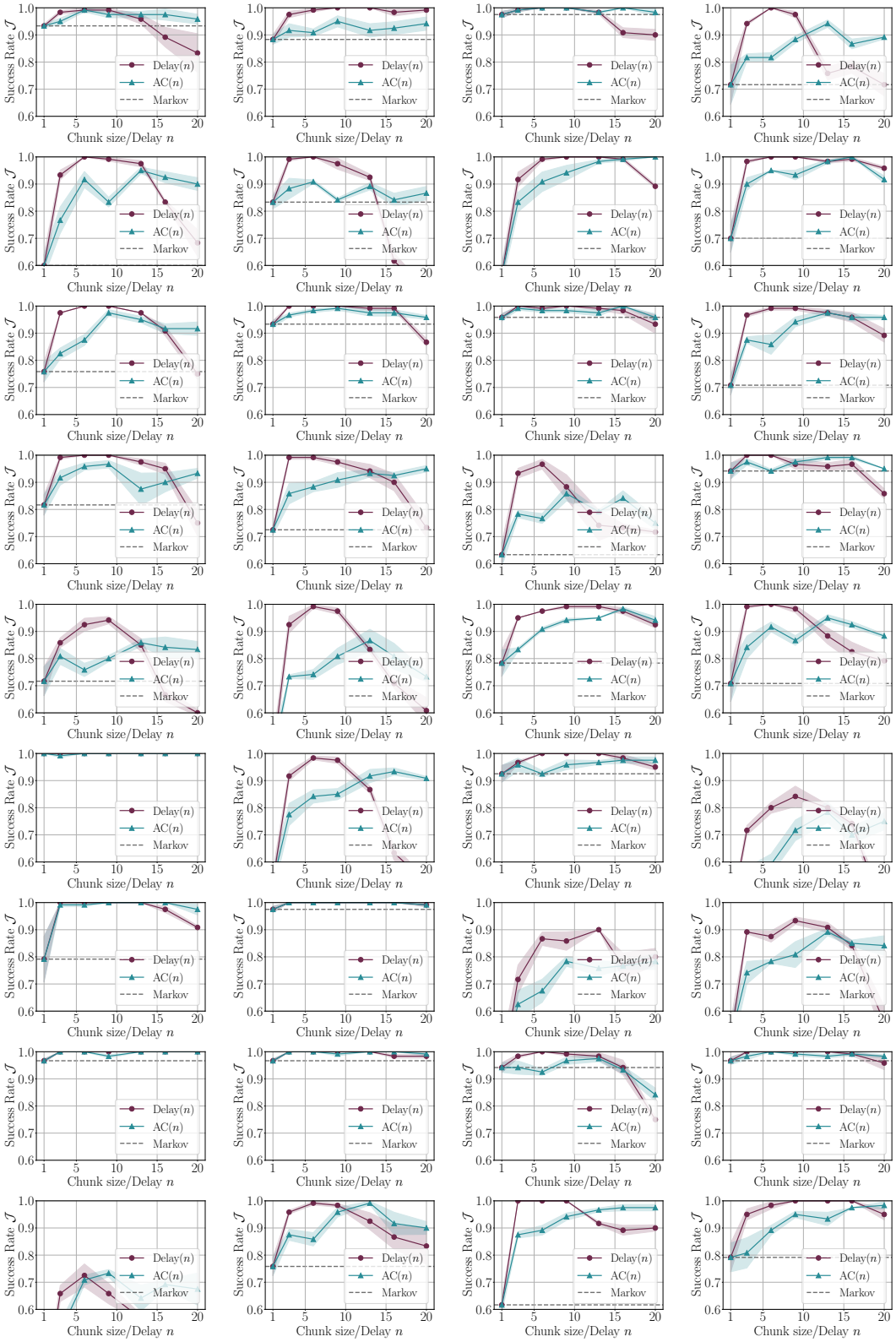


Figure 21: Success rate for each `Libero` task from 0 to 35 (corresponding to Fig. 3), part 1.

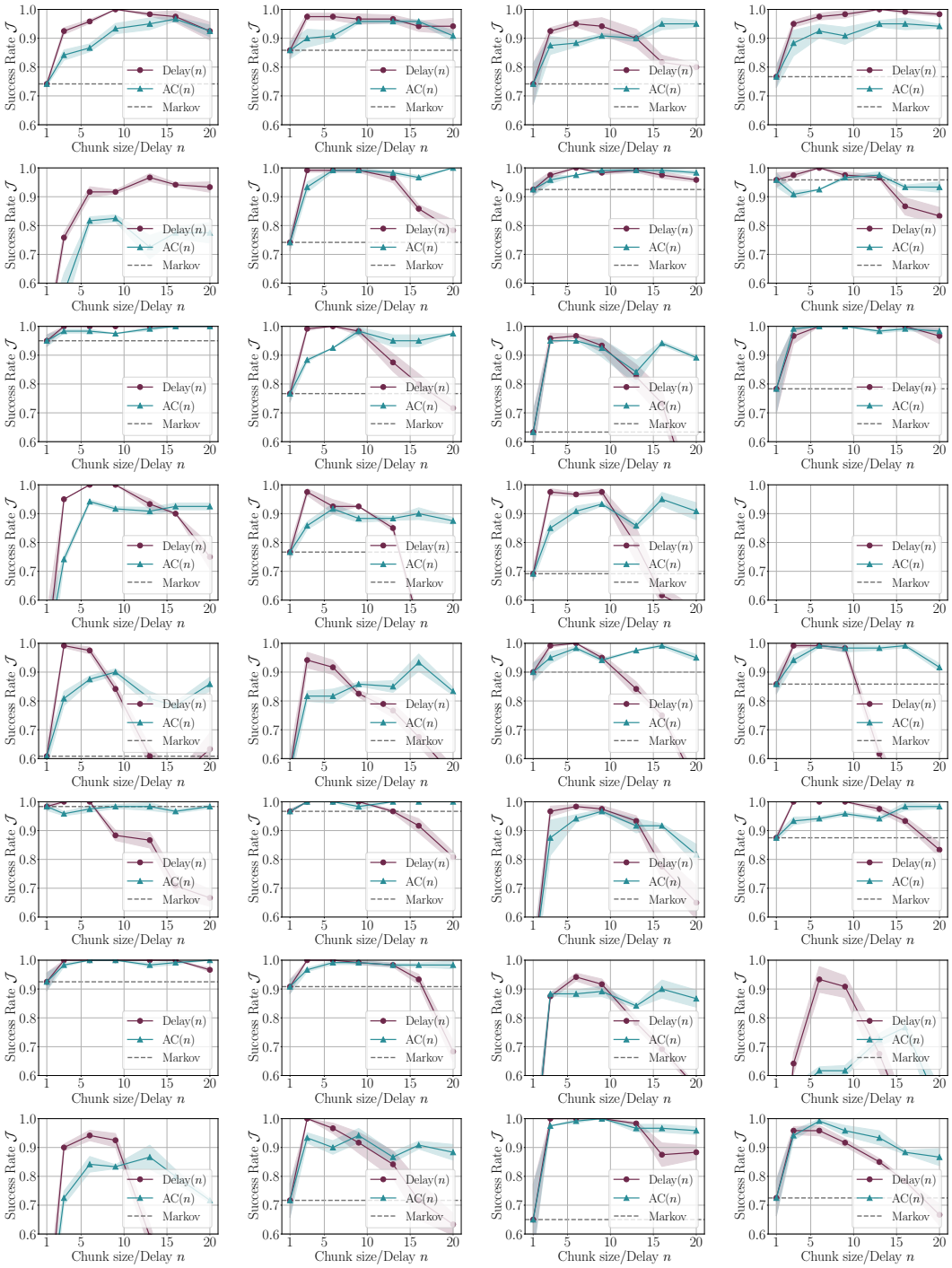


Figure 22: Success rate for each `Libero` task from 36 to 67 (corresponding to Fig. 3), part 2.

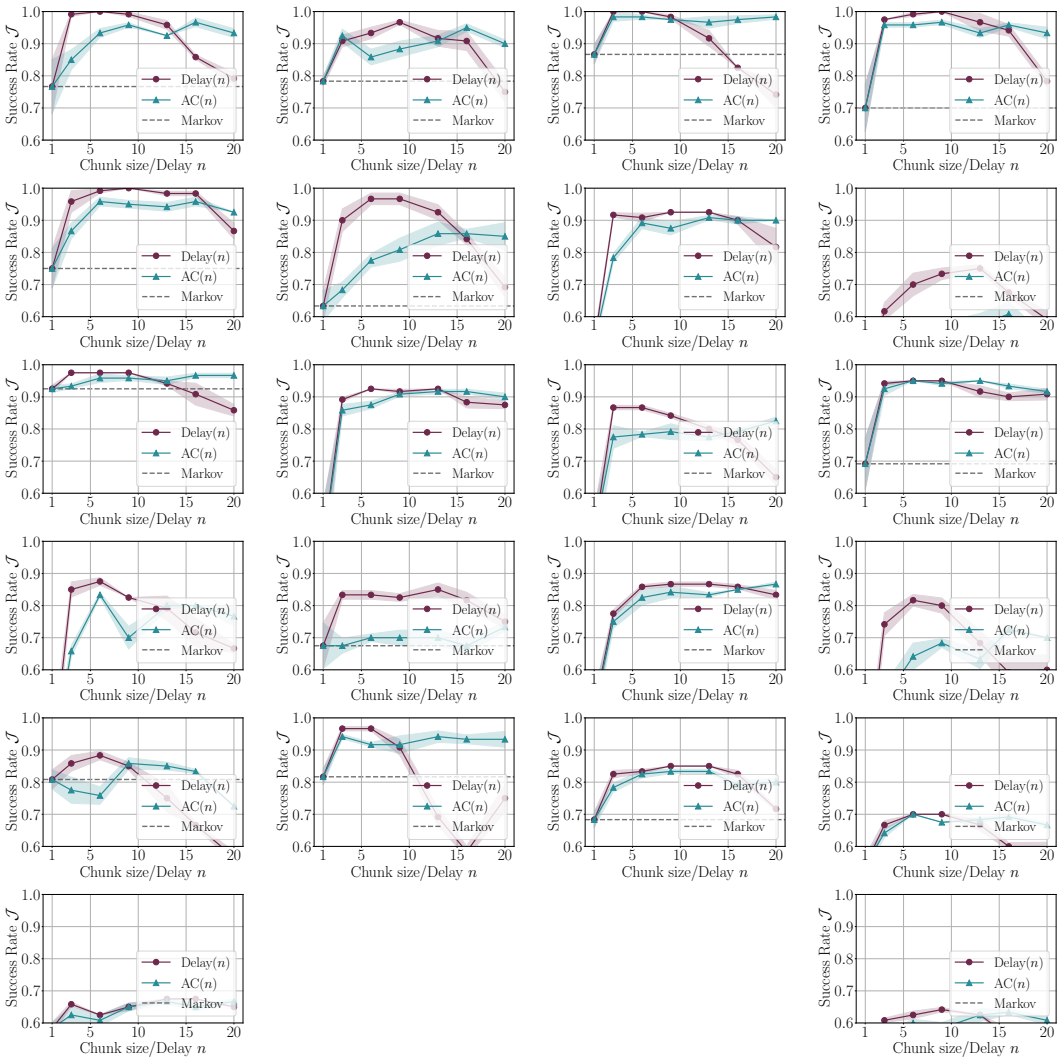


Figure 23: Success rate for each `Libero` task from 68 to 89 (corresponding to Fig. 3), part 3.

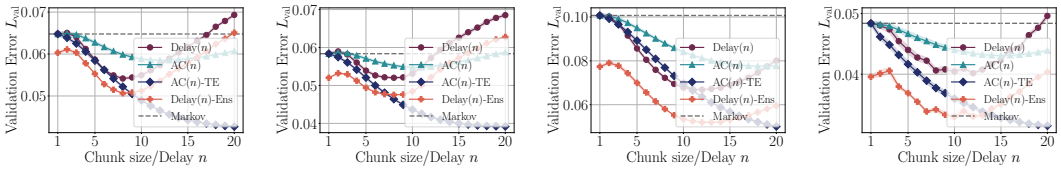


Figure 24: Val loss for each `Robomimic` task in this order from left to right: can, square, transport, tool hang (corresponding to Fig. 9), part 1.

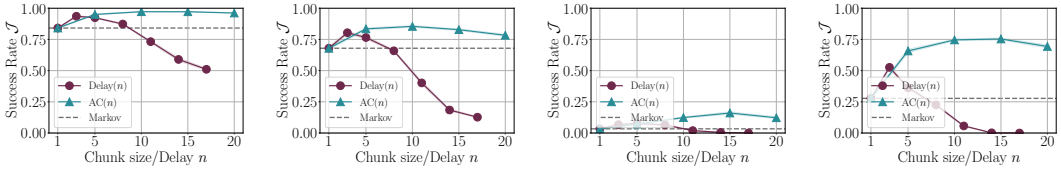


Figure 25: Success rate for each `Robomimic` task in this order from left to right: can, square, transport, tool hang (corresponding to Fig. 8), part 1.