Explicitly capturing demonstrator multi-modality for distillation into diffusion policies

Shao Yuan Chew Chia[1], **Rebecca H. Jiang**[1], **Abhinav Agarwal**[1]
Andreas Hofmann[2], **Leslie Pack Kaelbling**[1], **Russ Tedrake**[1]
[1]Computer Science and Artificial Intelligence Laboratory (CSAIL),
Massachusetts Institute of Technology, Cambridge, MA, USA
[2]Pickle Robot Company, Charlestown, MA, USA
shaoyuan@mit.edu

Abstract: The *diversity* of a distilled diffusion policy's outputs matters beyond how it impacts the policy's success rate. In particular, downstream applications like test-time search and policy improvement require exploration. Thus, we want to distill multi-modal demonstrators into diffusion policies which reproduce their conditional action distributions. Standard data collection for behavior cloning applies a sliding window to demonstration episodes, which pairs *unique observation histories* with actions. These datasets can be fit by a policy that is either multi-modal, or uni-modal with high Lipschitz constant to capture a diversity of actions seen at *nearby* observation samples. Prior work finds that diffusion policies trained on sparse data learn the latter, memorizing one action per observation. We adapt *vine sampling* to behavior cloning data collection to resolve this ambiguity: at some states the demonstrator visits, query it k times from *exactly* the same observation. On a synthetic task, we study how the number of actions per observation, k, affects the learned policy, and find that larger k yields a policy that better matches the demonstrator's conditional distribution, including its multi-modality and diversity. On a 2D truck unloading task, vine sampling likewise leads to a policy closer to the demonstrator and substantially improves both best-of-N and expert iteration performance.

Keywords: behavior cloning, data collection, multi-modal action distributions

1 Introduction

Diffusion and flow policies trained with behavior cloning (BC) have proven effective on a range of challenging manipulation tasks [1, 2]. In BC, a policy is trained to imitate an *expert demonstrator*, such as a human teleoperator, a motion planner, or a scripted controller. In many applications, however, success rate is not the only metric of interest. The diversity of a policy's action distribution, and in particular its *multi-modality* – that is, its capacity to take different actions in response to the same observation – is important when the policy is used for exploration by some downstream procedure. For instance, test-time search draws many candidate actions from a policy and selects the best with a

Pretrain to Adapt Workshop at the 10th Conference on Robot Learning (CoRL 2026), Austin, Texas, USA.

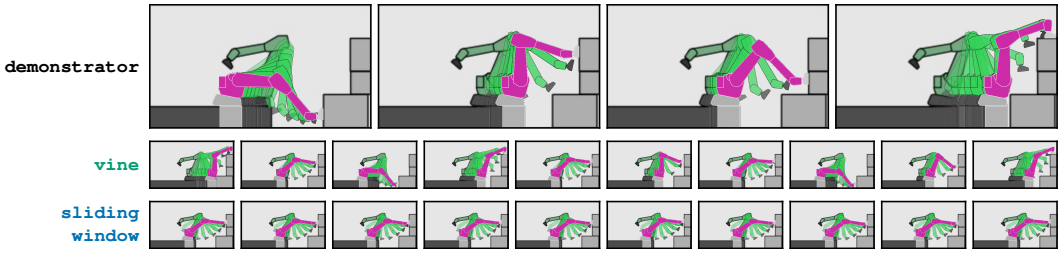


Fig. 1: Actions from the truck unloading demonstrator of app. C.1 (top), the `vine` policy (middle), and the `sliding window` policy (bottom), queried at the same state of a held-out problem instance. Each panel shows the observation image with future arm poses overlaid; poses with suction on are shown in pink. Of 10 samples each, the `vine` policy's reach three of the four grasps, while the `sliding window` policy's all approach the same grasp.

verifier or value function [3, 4], and is limited when the policy's samples are insufficiently diverse [5]. Expert iteration [6] and reinforcement learning (RL) post-training [7] similarly shift a policy toward higher-reward actions that it samples, directly or through a search it guides. Diversity also matters when a policy is used as a skill inside a task-and-motion-planning framework [8], where constraints may only become known at test time and a unimodal policy's single approach may prove inadmissible.

For RL post-training, this need has been made precise as a coverage condition: to reach the demonstrator's performance, the pretrained policy must, in some cases, place sufficient probability mass on the demonstrator's actions [7]. In robotics, BC policies often violate this coverage condition [7, 9]. Even when the demonstrator is stochastic and multi-modal, as in the top row of fig. 1, the trained policies often concentrate on a single mode per observation, as in its bottom row.

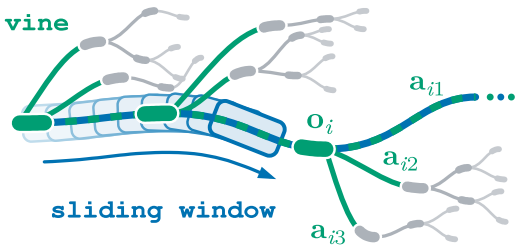


Fig. 2: The choices that a demonstrator has the capacity to take branch along an episode. Sliding window datasets record one action per observation history, while vine datasets record several counterfactual actions at a random subset of observation histories. Gray actions are recorded in neither dataset.

These low-diversity policies follow from the way training datasets are constructed. Standard practice collects full episodes of the demonstrator solving the task, then applies a *sliding window* that, at each timestep, pairs an observation history with the demonstrator's actions over the prediction horizon [10]. Each episode records a particular sequence of actions taken by the demonstrator. At each decision point, the demonstrator could have chosen differently, but the resulting dataset does not reflect these counterfactual actions (fig. 2). Because observations are often high dimensional and continuous, and reached states stochastic, other episodes never repeat exact observation histories.[1] The demonstrator's multi-modality is therefore not explicitly represented in the data [11]. At the same time, diffusion models at these dataset scales, given sufficiently expressive architectures and enough training, operate in or near the memorization regime (sec. 2). The standard pipeline therefore memorizes a dataset in which each observation is paired with a single mode; low policy diversity is an expected outcome. Some architectures may be better than others at nonetheless learning to generalize based on nearby observations paired with different action modes. Yet, these datasets themselves remain fundamentally ambiguous: the underlying demonstrator could be interpreted as multi-modal or unimodal with high Lipschitz constant. In this paper, we focus on disambiguating the dataset.

The natural solution is to record multiple samples per condition, and some methods already do so [12, 13]. In these works, however, it is one component of a larger method; its effect has not been isolated. To our knowledge, no work in BC has varied the number of recorded actions per observation, fixing dataset size and training budget, and studied the relationship between the conditional distributions of the demonstrator, dataset, and trained policy. We study this relationship in a controlled synthetic setting (sec. 4) and in a simulated 2D truck unloading problem that captures challenges relevant to real robot tasks. Furthermore, we study how this intervention affects performance on our ultimate goal of downstream policy self-improvement.

We consider *re-queryable settings*, in which the demonstrator can be queried multiple times from the same world state. Such settings arise in simulation, where the environment can be reset between queries, with closed-loop demonstrators such as a human teleoperator or a feedback controller. They also arise, in simulation or the real world, with demonstrators that compute their actions without changing the state of the world, such as an open-loop motion planner.

This paper makes two **contributions**.

Analysis. In a controlled synthetic setting, we study how the number of *samples* (recorded actions) per *condition* (observation) k affects how well the policy learns the demonstrator's conditional distribution

[1] When the observation space is discrete and small, the same observation history may repeat across episodes, and the sliding-window dataset may contain explicit multi-modality.

at matched total dataset size (sec. 4). With more samples per condition, the model still memorizes, but the memorized samples better capture the diversity and multi-modality of the demonstrator.

Application to BC for robotics. Motivated by this analysis, we adapt the vine procedure of Schulman et al. [14] into a BC data-collection protocol that explicitly captures the demonstrator's multi-modality: at each step, with some probability, re-query the stochastic demonstrator k times from the current state (sec. 5). On a simulated 2D truck unloading task, at matched dataset size, the resulting policy better matches the demonstrator's action distribution and diversity, and substantially improves both best-of-N search and expert iteration (sec. 6).

2 Related Work

Memorization in conditional diffusion models. The exact minimizer of the denoising objective is the score of the Gaussian-smoothed empirical distribution of the training set, so a sufficiently expressive and fully trained diffusion model memorizes, generating only its training samples [15, 16]. In the conditional setting, this holds at each training condition: the model generates the uniform distribution over the samples stored there [15]. Finer conditioning granularity, with unique per-sample labels as the extreme case, accelerates memorization [17]. A sliding-window dataset, in which every condition is a unique observation history, is an analogous case for BC, and diffusion policies in the sparse-data regime indeed behave like retrieval over their training set [18].

Increasing policy diversity downstream of data collection. Low sample diversity in behavior-cloned generative policies hampers self-improvement [19], reranking-based search [5], and RL fine-tuning [7, 9]. Wagenmaker et al. [7] and Hong et al. [9] synthesize increased diversity at training time, by noising actions and conditioning respectively, and Jin et al. [19, 20] at inference time, with perturbations to conditioning or in a learned latent space. Pan et al. [11] find that generative policies trained on standard BC benchmarks show little multi-modality and do not owe their single-rollout success to it; consistent with our analysis, they attribute its absence to data sparsity, as the same policies fit both modes when the data are dense, but they do not measure downstream search or improvement. Belkhale et al. [21] likewise note that individual states are visited only a few times, so a policy cannot match a high-entropy expert there, and curate demonstrations toward lower expert entropy to improve single-rollout success. In contrast, we retain the expert's entropy for downstream search by recording more samples per condition, which in our setting increases single-rollout success.

Multiple samples per condition. Training on multiple samples per condition is common where a condition has more than one correct answer: language models are fine-tuned on multiple responses per prompt, with gains that grow with the number of responses [22, 23], and segmentation models are trained on labels from multiple annotators [24]. In robotics, grasp planners are trained on multiple grasps per object [25, 26], and motion planners are trained on several trajectories per planning problem to learn multi-modal priors [27, 12]. Most closely related to our work, Dong et al. [13] distill a multi-step flow policy into a one-step student. They sample the teacher several times per observation and train the student with a set-level Implicit Maximum Likelihood Estimation objective that preserves the teacher's modes, reporting that success rate rises with the number of samples per observation. We study this core idea for diffusion models trained with the standard denoising objective to assess its achievement of two goals: reproducing the demonstrator's conditional action distribution, and facilitating downstream search and self-improvement.

3 Problem Formulation

3.1 Setting

The policy conditions on an observation history $\mathbf{o} \in \mathbb{R}^{T_o \times d_o}$ and outputs actions $\mathbf{a} \in \mathbb{R}^{T_a \times d_a}$, where T_o is the length of the history, T_a the prediction horizon (number of timesteps the predicted actions span), and d_o and d_a are the dimensions of a single timestep's observation and action, respectively. Square brackets index within an object: $\mathbf{a}[h, c]$ is the c-th action coordinate, h timesteps into the future.

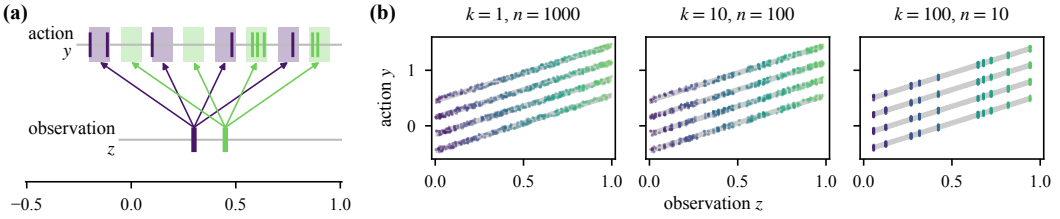


Fig. 3: (a) Coordinate y of the synthetic task at two observations, with $k = 5$ actions sampled at each; arrows run from each observation to its four valid modes. (b) Visualization of synthetic datasets with fixed total dataset size $n \cdot k = 1000$, but varied numbers of observations n, and actions per observations k; gray diagonals show the valid modes of the task. At fixed dataset size, more actions per condition comes at the cost of sparsity in z.

We model the expert demonstrator as a conditional distribution $\pi_E(\cdot \mid \mathbf{o})$ over actions. BC trains a policy π_θ on a dataset of *training samples*: observation–action pairs $(\mathbf{o}, \mathbf{a})$ with $\mathbf{a} \sim \pi_E(\cdot \mid \mathbf{o})$. In our setting, multiple actions may share the same observation history; we write the dataset grouped by it, $\mathcal{D} = \left\{ (\mathbf{o}_i,\ \mathbf{a}_{i1}, \ldots, \mathbf{a}_{ik}) \right\}_{i=1}^{n}$, with n distinct conditions and k actions per condition, each drawn independently from $\pi_E(\cdot \mid \mathbf{o}_i)$. This is equivalent to a dataset of $n \cdot k$ pairs $\{(\mathbf{o}_i, \mathbf{a}_{ij})\}$. Standard practice takes $k = 1$; we analyze the effect of increasing k, holding the dataset size $n \cdot k$ fixed (fig. 3b). We assume the policy class is expressive enough to represent multi-modal action distributions for the task.

3.2 Study of the trained policy

We are interested in whether $\pi_\theta(\cdot \mid \mathbf{o})$ reproduces $\pi_E(\cdot \mid \mathbf{o})$ as a distribution, rather than in the success rate of a single rollout of π_θ. We evaluate this via two metrics. The policy's conditional should be close to the demonstrator's as a distribution, which we capture via the 1-Wasserstein distance W_1 in (1). It should also place enough mass on each of the demonstrator's modes that N draws, as in best-of-N search, reach them. W_1 does not capture this, so we measure it as mode recall@N (sec. 4.2).

At a training condition $\mathbf{o}_i$, we compare three distributions over actions: the demonstrator's conditional $\pi_E(\cdot \mid \mathbf{o}_i)$ is the object we want the policy to reproduce; the empirical conditional $\hat{P}_k(\mathbf{o}_i) = \frac{1}{k} \sum_{j=1}^{k} \delta_{\mathbf{a}_{ij}}$ is the uniform distribution over the k training actions that the collection protocol stored at that condition; and the trained policy's conditional is $\pi_\theta(\cdot \mid \mathbf{o}_i)$. Baptista et al. [15] show that the minimizer of the conditional denoising objective at $\mathbf{o}_i$ is the empirical score, the score of $\hat{P}_k(\mathbf{o}_i)$ under the forward noising process, and that sampling with it reproduces $\hat{P}_k(\mathbf{o}_i)$ exactly. Because W_1 is a metric, the triangle inequality with $\hat{P}_k(\mathbf{o}_i)$ as a waypoint gives

$$\underbrace{W_1(\pi_\theta,\, \pi_E)}_{\text{(LHS) policy error}} \quad \leq \quad \underbrace{W_1(\pi_\theta,\, \hat{P}_k)}_{\text{(A) memorization error}} \quad + \quad \underbrace{W_1(\hat{P}_k,\, \pi_E)}_{\text{(B) data-collection sampling error}}, \tag{1}$$

where all three distributions are conditioned on $\mathbf{o}_i$. Term (A) is a property of the model: how far its conditional is from the training actions at that observation. Term (B) is a property of data collection: how well k draws represent the distribution they came from. Taking π_θ as the waypoint instead gives $W_1(\hat{P}_k, \pi_E) \leq W_1(\pi_\theta, \hat{P}_k) + W_1(\pi_\theta, \pi_E)$, which bounds policy error from below by (B) − (A). The policy error therefore lies within ± (A) of (B); when (A) is small, as it is near the minimizer described above, small (B) is necessary and sufficient for small policy error. Thus, a policy that memorizes its training actions can reduce its policy error only through a smaller (B), which on average requires a larger k. This motivates studying the isolated effects of increasing k on policy error and performance.

4 Analysis

4.1 Synthetic task

The synthetic task is defined on observations $\mathbf{o} \in \mathbb{R}^{1\times 16}$ and actions $\mathbf{a} \in \mathbb{R}^{1\times 16}$: 16-dimensional observations and actions, with no notion of time. The observation has one informative dimension, denoted $z := \mathbf{o}[1, 1]$, and 15 irrelevant "nuisance" dimensions. The observation marginal is $\mu = \mathrm{Unif}[0, 1]^{16}$, so $z \sim \mathrm{Unif}[0, 1]$. The nuisance dimensions enlarge the observation space, distancing the 16D observations from one another, to mimic the sparse-observation property seen in diffusion

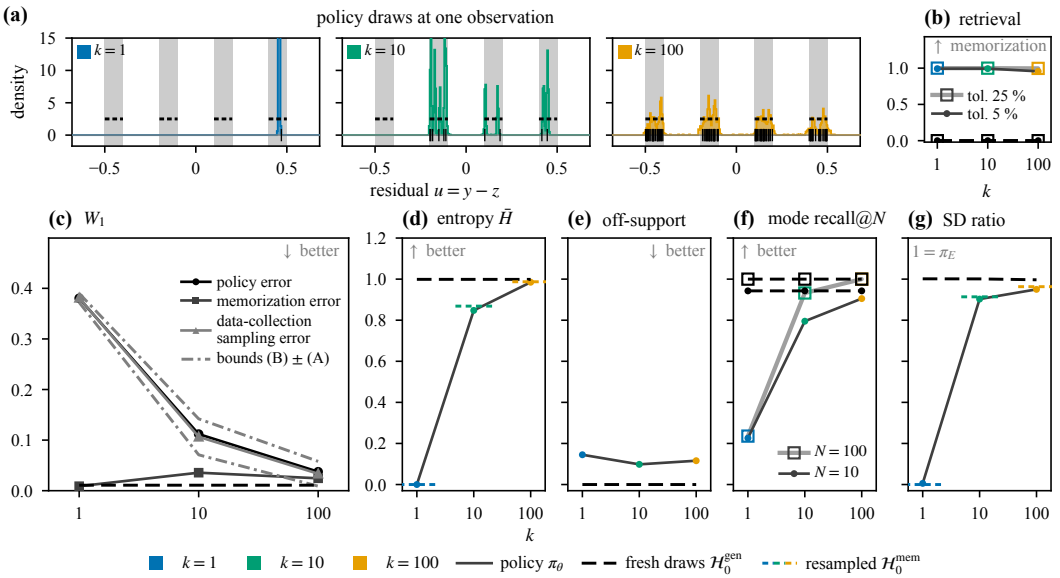


Fig. 4: Policies trained on the three datasets of fig. 3b, queried at their training observations. (a) Each policy's draws at a representative training observation. Dashed segments and gray bands are π_E and its support. Short black lines mark the training actions at each observation. (b)–(g) The metrics of sec. 4.2, averaged over training observations; (c) shows the terms of (1) on the primary coordinate y. Reference lines are fresh draws from the demonstrator (black dashed, $\mathcal{H}_0^{\text{gen}}$) and resampled draws from the training actions (colored dashed, $\mathcal{H}_0^{\text{mem}}$).

policies with high-dimensional image observation histories. As in these real observations, only a small subset of the high-dimensional input is decision-relevant. The *primary* action coordinate $y := \mathbf{a}[1, 1]$ determines whether an action is valid: it must lie in one of four intervals shifted by the observation z (fig. 3a). The remaining 15 coordinates can take any value, analogous to redundant degrees of freedom, as when inverse kinematics admits multiple joint configurations for one end-effector pose.

The demonstrator's conditional distribution $\pi_E(\cdot \mid \mathbf{o})$ is given by

$$y = z + u, \quad u \sim p_R, \qquad \mathbf{a}[1, c] \;=\; \varepsilon_c \quad (c = 2, \ldots, 16), \quad \varepsilon_c \overset{\text{i.i.d.}}{\sim} \mathcal{N}(0, 0.3^2), \tag{2}$$

where the residual distribution p_R is a mixture of the four valid modes, equally weighted and uniform (fig. 3a). The other 15 action dimensions are unconditional Gaussians. Because they are random, a policy draw can lie close to a training action in all 16 dimensions only if the policy reproduces that action, which our retrieval metric relies on.

4.2 Findings

We run a sweep of $k \in \{1, 10, 100\}$ at matched budget $n \cdot k = 1000$, so the number of conditions is $n \in \{1000, 100, 10\}$ (fig. 3b). Every metric is computed from 1000 policy draws per condition, averaged over conditions. To give a qualitative sense of what the model learned, we plot a histogram of the residual $u = y - z$ of the policy's primary-task action when the policy is queried repeatedly at a particular training condition (fig. 4a). Each policy's draws closely match the distribution of training actions seen at each condition. For $k = 1$, since there is only a single action in the dataset for this condition, the other three valid modes are not represented.

To test whether a policy memorizes its training data, we compare each of its draws with the nearest training action, as in nearest-neighbour tests of memorization for image diffusion models [28, 17] and diffusion policies [18]. *Retrieval* is the fraction of draws whose 16D distance to that action is at most 5% or 25% of the mean distance between two demonstrator draws at the same condition (fig. 4b). All three policies show high retrieval at both the 5% and 25% levels, whereas essentially none of the fresh draws from the demonstrator fall within these tolerances of the training actions.

Plotting the W_1 distances between π_E, π_θ, and $\hat{P}_k$ on the primary coordinate y (fig. 4c), the data-collection sampling error (B) decreases as k increases, i.e., the dataset better represents the demon-

strator's conditional distribution. The memorization error (A) is small at every k, consistent with the retrieval results. Since the policy error lies within (A) of (B), it closely tracks (B) and falls with it.

To understand the policy's multi-modal behavior, we estimate two quantities. The first is conditional behavior entropy $\bar{H}(\mathbf{o})$ [29], the entropy of the policy's distribution over the demonstrator's modes at a condition, normalized to $[0, 1]$, which measures how evenly probability mass is distributed among those modes. The second is off-support mass: the proportion of draws that land outside the demonstrator's support. The entropy $\bar{H}(\mathbf{o})$ is normalized over the mass in the demonstrator's support, so off-support mass complements it. The entropy $\bar{H}$ increases with k, from 0 at $k = 1$ (all on-support mass lies in one mode) to 1 at $k = 100$ (mass evenly distributed across the modes) (fig. 4d), while the off-support mass is similar across k values (fig. 4e).

Mode recall@N is the expected proportion of demonstrator mass lying in modes that at least one of N independent policy draws lands in. Mode recall@N is also substantially higher at larger k (fig. 4f). As expected, at $k = 1$, increasing N from 10 to 100 barely moves recall; more sampling at inference time does not recover the demonstrator's multi-modality. The primary-task standard-deviation (SD) ratio is the SD of the policy's primary-task draws divided by the demonstrator's; it is 1 in expectation for the demonstrator and 0 for a collapsed policy. On this measure of diversity, the $k = 10$ and $k = 100$ policies also match the demonstrator much more closely than the $k = 1$ policy does (fig. 4g).

At held-out conditions, compared to the $k = 1$ policy, the $k = 10$ and $k = 100$ policies continue to have favorable policy error, entropy $\bar{H}$, mode recall and SD ratio at every distance d from the query to its nearest training observation (fig. 6). The $k = 1$ policy remains a point mass at every distance, while the $k = 10$ and $k = 100$ policies gradually lose their four-mode structure, so the gaps narrow with distance. Retrieval falls for all policies, fastest for $k = 1$.

With enough data, the $k = 1$ policy also reaches a low policy error. At $n \cdot k = 10^5$, it matches the $k = 10$ and $k = 100$ policies in policy error, entropy $\bar{H}$, off-support mass and SD ratio (fig. 7). Its retrieval there is near zero (fig. 7b), and its policy error is far below the sampling error (B) of a $k = 1$ dataset (fig. 4c), whose mean does not depend on n. By the lower bound of sec. 3.2, it can reach that error only by moving away from its training actions, which it does by generalizing across conditions. For this task and model architecture, the advantage of larger k is therefore limited to the sparse-data regime, where a model with enough capacity and training memorizes. Diffusion policies at typical BC dataset scales operate in or near this regime (sec. 2), which motivates collecting several actions per condition (sec. 5).

5 Data collection method: vine sampling

Motivated by the preceding analysis, we investigate the effects of extending $k > 1$ data collection to robot BC. To facilitate this, we adapt *vine sampling*,[2] (alg. 1 in app. B). The demonstrator π_E visits states as usual: replanning periodically and enacting the stored plan in between. At each timestep, with probability p, we query the expert k times from the current state, resetting the environment between queries if the state is modified by the action decision process (e.g., simulation for closed-loop control). Only these states enter the dataset, each carrying exactly k actions. Because the expert is multi-modal, the k actions at a labeled state demonstrate up to k different ways of continuing from it; they capture the demonstrator's conditional distribution in a way the sliding window approach does not (sec. 1 and fig. 2). App. B discusses other sampling strategies, and how to apply vine sampling when the demonstrator cannot be queried from some visited states.

6 Truck unloading experiments

In this section, we compare policies trained with our proposed vine sampling data-collection approach and standard sliding window on a 2D truck unloading task (pictured in fig. 1). In this task, a planar

[2]We adapt the vine procedure of Schulman et al. [14] (also used by Kazemnejad et al. [30] and Hamid et al. [31]), which branches policy rollouts to estimate advantages for policy-gradient updates; we instead use the demonstrator's branched actions as BC training data.

robot with a suction gripper removes four boxes from a truck by placing each box on a conveyor belt that moves with its base. Success is defined as removing all boxes before the timeout at 80s. We also measure throughput, in pieces per hour. The task captures several challenges of real-world robotics, including meaningful multi-modality: the robot has to make both discrete (which box to grasp) and continuous (how to grasp it) decisions. At each decision, the number of modes grows with the number of boxes; over an episode the number of possible pick orders grows combinatorially. We describe the task and the model-based demonstrator we implement in more detail in app. C.1.

Each method collects demonstrations on the same 2,000 problem instances. For the $k = 1$ (`sliding window`) dataset we apply the standard sliding window to the episodes. The $k = 10$ (`vine`) dataset is collected with alg. 1 at $p = 0.1$. Since $p \cdot k = 1$, the two methods produce the same expected number of training samples per episode. Both datasets are truncated to 370,000 training samples, so the collection budget $n \cdot k$ is matched. Like in the synthetic experiment (sec. 4), at matched dataset size, the vine dataset captures explicit multi-modality at the cost of being sparser in conditioning space. We then train one policy on each dataset for 100,000 steps with the diffusion policy architecture of Chi et al. [10]. We compare against four baselines that use the sliding-window dataset and intervene downstream of data collection, at training or inference time: `early stopping`, `more search`, `SIME` and `PostBC`, each described with its results below.

We evaluate the trained policies on two downstream use cases: test-time best-of-N search, and expert iteration [6] policy improvement, where we use best-of-N in place of Monte-Carlo Tree Search. Their implementations are described in app. C.5. We report results for $N \in \{10, 100\}$ and the raw policy, which executes predicted actions without search. We initialize expert iteration with the `sliding window`, `vine` and `SIME` policies.

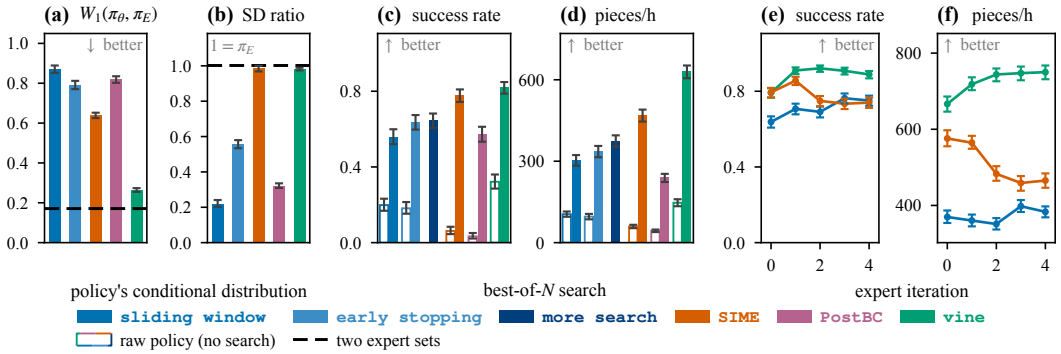


Fig. 5: Truck unloading experiment. (a), (b) Policy error $W_1(\pi_\theta, \pi_E)$ and SD ratio per method at held-out conditions. (c), (d) Success rate and throughput, in best-of-N search; $N = 10$ for solid bars except `more search` where $N = 100$. (e), (f) The same two metrics over expert iteration for the `sliding window`, `SIME`, and `vine` methods. Plotted values are means and error bars are 95% intervals.

`vine` and `sliding window`. As anticipated, `vine` shows superior policy error and SD ratio, suggesting that it captures the demonstrator's underlying distribution better than alternatives. It also demonstrates the intended impacts on search, yielding high success rates and throughput. Furthermore, these metrics improve via expert iteration, with success rate falling slightly at later iterations, which we attribute to imperfect distillation between iterations combined with diminishing diversity at later iterations, as only the base policy is trained via vine sampling. `sliding window`'s success rate rises, but its successful episodes take longer on average, so throughput changes little. We attribute this to its limited diversity: with similar candidate actions, best-of-N search has little better behavior to select and distill. In contrast, `vine` completes episodes faster over the course of expert iteration.

`early stopping`. SD ratio falls as training progresses and the policy memorizes (fig. 9b). We use an earlier checkpoint as a simple attempt to maintain diversity, at the cost of reduced training. We choose the 20k checkpoint, whose raw policy success rate and throughput are similar to those of the 100k `sliding window` checkpoint. Indeed, this yields a higher SD ratio, and thereby improves best-of-N search. However, while behaviorally diverse, it does not mimic the demonstrator's choices as well as `vine` (higher policy error), leading to worse downstream search than `vine`.

more search. We evaluate best-of-N with standard sliding window and $N = 100$ instead of the $N = 10$ used with all other methods. This increases success rate and throughput, but performance is still far from that of vine. If the policy has a collapsed distribution, sampling repeatedly from it is ineffective at recovering the demonstrator's coverage – we see this effect clearly in sec. 4.2, fig. 4f.

SIME. At inference time, SIME [19] adds noise to the output of the observation encoder, annealed across the denoising timesteps of the diffusion policy. We choose the SD of the added noise such that SIME matches the demonstrator's SD (fig. 8b). In search, SIME approaches the success rate of vine, but falls short on throughput, suggesting that the form of injected randomness adds enough options for the search to find an admissible, albeit inefficient, solution. As a raw policy, SIME produces more failures relative to sliding window, which shares its checkpoint. We attribute this to the noise injected into the observation encoding, which reduces the precision of the policy's actions, such as those needed to avoid contact with other objects. Across expert iteration, SIME's success rate and throughput fall in later iterations and its motions become increasingly jerky (figs. 5e and 5f), presumably because these imprecise actions are distilled into the policy at each iteration.

PostBC. PostBC [7] adds Gaussian noise to each training action, with the covariance of an ensemble's predictions of the mean action at that observation. This widens the policy's distribution, but does not add the demonstrator's counterfactual actions, which vine captures. Accordingly, PostBC raises SD ratio and lowers policy error only slightly relative to sliding window, and its raw policy succeeds far less often.

We conclude that, on this task, adding diversity to a policy trained on a sliding-window dataset, at either training or inference time, is a poor substitute for recording the demonstrator's multi-modality in the dataset itself. vine, whose dataset records that multi-modality explicitly, performs best in both search and policy self-improvement.

7 Conclusions and limitations

In this work, we study recording several demonstrator actions per observation as a way to distill a multi-modal demonstrator into a diffusion policy. In a controlled synthetic setting at a matched dataset size, the policy memorizes its training data at every k, but with larger k the memorized data contain more of the demonstrator's diversity and multi-modality. Motivated by this finding, we apply vine sampling to BC data collection, which queries the demonstrator k times at a random subset of the states the demonstrator visits. On a 2D truck unloading task, the vine policy reproduces the demonstrator's conditional distribution more closely than the sliding-window policy, and substantially improves both best-of-N search and expert iteration.

Vine sampling can require more demonstrator queries than the sliding window for the same number of training samples: the sliding window turns every timestep of an executed episode into a training sample, while vine sampling spends the same queries to complete the episode, and one additional query per training sample. For our truck unloading demonstrator, which plans once per pick and place, collecting the vine dataset costs about 30 times the compute of the sliding-window dataset at matched size. For conciseness, we have not investigated optimal values for the re-query probability p and the query count k. Furthermore, the best k for best-of-N search may not be best for rollouts of the raw policy. Finally, we study vine sampling only in simulation, where every method can be evaluated on the same problem instances and at a scale that would be costly on hardware. Our implementations of best-of-N search and expert iteration also use the simulator to evaluate candidate actions. When the demonstrator does not change the state of the world when queried, vine sampling does not require environment reset and can be run on hardware, but the re-queried plans are then never executed; their quality is not verified. Our synthetic analysis suggests that the advantage of larger k is limited to the sparse-data regime (sec. 4.2). Future work could study whether large robot foundation models [1, 2], at their data scales, model sizes and training recipes, operate in this regime, and hence whether vine sampling would improve the policies that test-time search and policy improvement build on.

Acknowledgments

This work was supported by Pickle Robot Company and The Charles Stark Draper Laboratory, Inc., where Rebecca H. Jiang is a Draper Scholar. Part of this work was done while Shao Yuan Chew Chia was an intern at Pickle Robot Company.

References

[1] 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. In *Proceedings of Robotics: Science and Systems (RSS)*, 2025. arXiv:2410.24164.

[2] TRI LBM Team, J. Barreiros, A. Beaulieu, A. Bhat, R. Cory, E. Cousineau, H. Dai, C.-H. Fang, K. Hashimoto, M. Z. Irshad, M. Itkina, N. Kuppuswamy, K.-H. Lee, K. Liu, D. McConachie, I. McMahon, H. Nishimura, C. Phillips-Grafflin, C. Richter, P. Shah, K. Srinivasan, B. Wulfe, C. Xu, M. Zhang, A. Alspach, M. Angeles, K. Arora, V. C. Guizilini, A. Castro, D. Chen, T.-S. Chu, S. Creasey, S. Curtis, R. Denitto, E. Dixon, E. Dusel, M. Ferreira, A. Goncalves, G. Gould, D. Guoy, S. Gupta, X. Han, K. Hatch, B. Hathaway, A. Henry, H. Hochsztein, P. Horgan, S. Iwase, D. Jackson, S. Karamcheti, S. Keh, J. Masterjohn, M. Masuda, J. Mercat, P. Miller, P. Mitiguy, T. Nguyen, J. Nimmer, Y. Noguchi, R. Ong, A. Onol, O. Pfannenstiehl, R. Poyner, L. P. M. Rocha, G. Richardson, C. Rodriguez, D. Seale, M. Sherman, M. Smith-Jones, D. Tago, P. Tokmakov, M. Tran, B. V. Hoorick, I. Vasiljevic, S. Zakharov, M. Zolotas, R. Ambrus, K. Fetzer-Borelli, B. Burchfiel, H. Kress-Gazit, S. Feng, S. Ford, and R. Tedrake. A careful examination of large behavior models for multitask dexterous manipulation. *Science Robotics*, 11(113):eaea6201, 2026. doi:10.1126/scirobotics.aea6201.

[3] H. Qi, H. Yin, A. Zhu, Y. Du, and H. Yang. Inference-Time Enhancement of Generative Robot Policies via Predictive World Modeling. *IEEE Robotics and Automation Letters (RAL)*, 11(5): 5534–5541, 2026. doi:10.1109/LRA.2026.3673995.

[4] J. Kwok, C. Agia, R. Sinha, M. Foutter, S. Li, I. Stoica, A. Mirhoseini, and M. Pavone. RoboMonkey: Scaling test-time sampling and verification for vision-language-action models. In *Proceedings of the 9th Conference on Robot Learning (CoRL)*, volume 305 of *Proceedings of Machine Learning Research*, 2025.

[5] L. Luo, X. Bu, R. Liu, Q. Huang, C. Tang, X. Hao, H. Wang, and W. Li. Data-asymmetric latent imagination and reranking for 3D robotic imitation learning. arXiv preprint arXiv:2605.10166, 2026.

[6] T. Anthony, Z. Tian, and D. Barber. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pages 5360–5370, 2017.

[7] 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.

[8] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:265–293, 2021. doi:10.1146/annurev-control-091420-084139.

[9] M. M. Hong, J. Zhang, A. Nagabandi, and A. Gupta. TMRL: Diffusion timestep-modulated pretraining enables exploration for efficient policy finetuning. In *Proceedings of Robotics: Science and Systems (RSS)*, 2026. arXiv:2605.12236.

[10] 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*, 44(10–11):1684–1704, 2025. doi:10.1177/02783649241273668.

[11] C. Pan, G. Anantharaman, N.-C. Huang, C. Jin, D. Pfrommer, C. Yuan, F. Permenter, G. Qu, N. Boffi, G. Shi, and M. Simchowitz. Much ado about noising: Dispelling the myths of generative robotic control. In *International Conference on Learning Representations (ICLR)*, 2026.

[12] J. Carvalho, A. T. Le, M. Baierl, D. Koert, and J. Peters. Motion planning diffusion: Learning and planning of robot motions with diffusion models. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1916–1923, 2023. doi:10.1109/IROS55552.2023.10342382.

[13] J. Dong, L. Zhang, L. Zhang, Y. Fu, K. Bai, Z.-C. Márton, Z. Bing, Z. Chen, A. C. Knoll, and J. Zhang. From flow to one step: Real-time multi-modal trajectory policies via implicit maximum likelihood estimation-based distribution distillation. arXiv preprint arXiv:2603.09415, 2026.

[14] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, pages 1889–1897, 2015.

[15] R. Baptista, A. Dasgupta, N. B. Kovachki, A. Oberai, and A. M. Stuart. Memorization and regularization in generative diffusion models. arXiv preprint arXiv:2501.15785, 2025.

[16] Z. Ye, Q. Zhu, M. Tao, and M. Chen. Provable separations between memorization and generalization in diffusion models. arXiv preprint arXiv:2511.03202, 2025.

[17] X. Gu, C. Du, T. Pang, C. Li, M. Lin, and Y. Wang. On memorization in diffusion models. *Transactions on Machine Learning Research (TMLR)*, 2025. arXiv:2310.02664.

[18] C. He, X. Liu, G. S. Camps, J. Bruno, G. Sartoretti, and M. Schwager. Demystifying robot diffusion policies: Action memorization and a simple lookup table alternative. In *International Conference on Learning Representations (ICLR)*, 2026. URL https://openreview.net/forum?id=PL0tJOfm7I. arXiv:2505.05787.

[19] Y. Jin, J. Lv, W. Yu, H. Fang, Y.-L. Li, and C. Lu. SIME: Enhancing policy self-improvement with modal-level exploration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2025. arXiv:2505.01396.

[20] Y. Jin, J. Lv, H. Xue, W. Chen, C. Wen, and C. Lu. SOE: Sample-efficient robot policy self-improvement via on-manifold exploration. arXiv preprint arXiv:2509.19292, 2025.

[21] S. Belkhale, Y. Cui, and D. Sadigh. Data quality in imitation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, 2023.

[22] Z. Yuan, H. Yuan, C. Li, G. Dong, K. Lu, C. Tan, C. Zhou, and J. Zhou. Scaling relationship on learning mathematical reasoning with large language models. arXiv preprint arXiv:2308.01825, 2023.

[23] H. Amin, K. Ahrabian, M. Yin, and R. Khanna. Escaping the mode lottery: Multi-response training improves language model generalization. arXiv preprint arXiv:2606.00544, 2026.

[24] S. A. A. Kohl, B. Romera-Paredes, C. Meyer, J. D. Fauw, J. R. Ledsam, K. H. Maier-Hein, S. M. A. Eslami, D. J. Rezende, and O. Ronneberger. A probabilistic u-net for segmentation of ambiguous images. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. arXiv:1806.05034.

[25] A. Mousavian, C. Eppner, and D. Fox. 6-DOF GraspNet: Variational grasp generation for object manipulation. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2901–2910, 2019. doi:10.1109/ICCV.2019.00299.

[26] J. Urain, N. Funk, J. Peters, and G. Chalvatzaki. SE(3)-DiffusionFields: Learning smooth cost functions for joint grasp and motion optimization through diffusion. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 5923–5930, 2023. doi: 10.1109/ICRA48891.2023.10161569.

[27] J. Urain, A. T. Le, A. Lambert, G. Chalvatzaki, B. Boots, and J. Peters. Learning implicit priors for motion optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7672–7679, 2022. doi:10.1109/IROS47612.2022.9981264.

[28] N. Carlini, J. Hayes, M. Nasr, M. Jagielski, V. Sehwag, F. Tramèr, B. Balle, D. Ippolito, and E. Wallace. Extracting training data from diffusion models. In *Proceedings of the 32nd USENIX Security Symposium*, 2023. arXiv:2301.13188.

[29] X. Jia, D. Blessing, X. Jiang, M. Reuss, A. Donat, R. Lioutikov, and G. Neumann. Towards diverse behaviors: A benchmark for imitation learning with human demonstrations. In *International Conference on Learning Representations (ICLR)*, 2024.

[30] A. Kazemnejad, M. Aghajohari, E. Portelance, A. Sordoni, S. Reddy, A. Courville, and N. L. Roux. VinePPO: Refining credit assignment in RL training of LLMs. In *International Conference on Machine Learning (ICML)*, volume 267 of *Proceedings of Machine Learning Research*, pages 29557–29590, 2025.

[31] J. I. Hamid, I. H. Orney, E. Xu, C. Finn, and D. Sadigh. Polychromic objectives for reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2026.

[32] R. Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics. `https://drake.mit.edu`, 2019.

[33] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report TR 98-11, Computer Science Department, Iowa State University, 1998.

[34] J. J. Kuffner and S. M. LaValle. RRT-Connect: An efficient approach to single-query path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 995–1001, 2000. doi:10.1109/ROBOT.2000.844730.

A Synthetic Analysis

This appendix gives the details of the synthetic analysis of sec. 4 and its results at held-out conditions and other dataset sizes.

A.1 Task, datasets and training

Each of the four modes of the residual distribution p_R of eq. (2) is an interval of width 0.1, and neighbouring modes are 0.2 apart, so the modes are centered at ± 0.15 and ± 0.45 and the support is $[-0.5, 0.5]$.

We collect twelve datasets for this synthetic task, one for each $k \in \{1, 10, 100\}$ at each total dataset size $n \cdot k \in \{10^2, 10^3, 10^4, 10^5\}$, and train a policy on each dataset. The three datasets with $n \cdot k = 10^3$ are plotted in fig. 3b, and their policies are evaluated on training conditions in fig. 4, and on held-out conditions in fig. 6. All twelve use the architecture and training recipe in table 1.

A.2 Metrics

For evaluations of the synthetic task policies, the W_1 terms against π_E in eq. (1) are computed exactly from the cumulative distribution function of p_R, without demonstrator samples, while the distribution of π_θ is approximated with 1000 samples, and $\hat{P}_k$ is exact from the training datasets. For retrieval, the nearest training action is searched over all $n \cdot k$ training actions, not only those stored at the query's

Implementation	LeRobot `DiffusionPolicy`
Architecture	1D convolutional U-Net, one level of width 256
Parameters	3.1M
Diffusion steps	100 for training and sampling (DDPM)
Noise schedule	cosine
Prediction target	ϵ
Optimizer	AdamW, learning rate 10^{-4}, weight decay 10^{-6}
Learning rate schedule	500-step warmup, then constant
Batch size	256
Training steps	5×10^6
Hardware	NVIDIA RTX 4090, 13.5 h per policy

Table 1: Architecture and training recipe of the twelve synthetic policies.

condition. Conditional behavior entropy is undefined when no draws land in the demonstrator's support, and is noisy when few do. Thus, when fewer than 15 of the 1000 land in the support, we set it to 0, its value for a collapsed policy, and keep the condition in the average. Dropping those conditions would mean each policy's entropy is averaged over a different set of conditions. At $k = 1$ and $n \cdot k = 10^3$, this happens at 7% of training conditions and 58% of held-out queries. At $k = 10$ and $k = 100$ this does not happen in any training conditions and occurs in under 1% of held-out queries. The reference lines $\mathcal{H}_0^{\text{gen}}$ and $\mathcal{H}_0^{\text{mem}}$ use the same number of draws as the policies (1000 draws). Plotted values are means over up to 128 training conditions per policy, evenly spaced in the informative observation coordinate z.

A.3 Held-out conditions

Held-out queries evaluated in fig. 6 are taken along straight lines that start at training observations and run in random directions in the 16-dimensional observation space. For each query, d is the distance to its nearest training observation, in units of the mean distance between two training observations computed on the $k = 1$, $n = 1000$ dataset.

We split d into 24 bins from 0 to 1.18 and draw 96 queries per bin for each policy, at most one from each line originating from a training observation. Plotted values at held-out conditions are means over the queries in each bin.

A.4 Results against total dataset size

Fig. 7 shows the metrics of fig. 4 for all twelve policies.

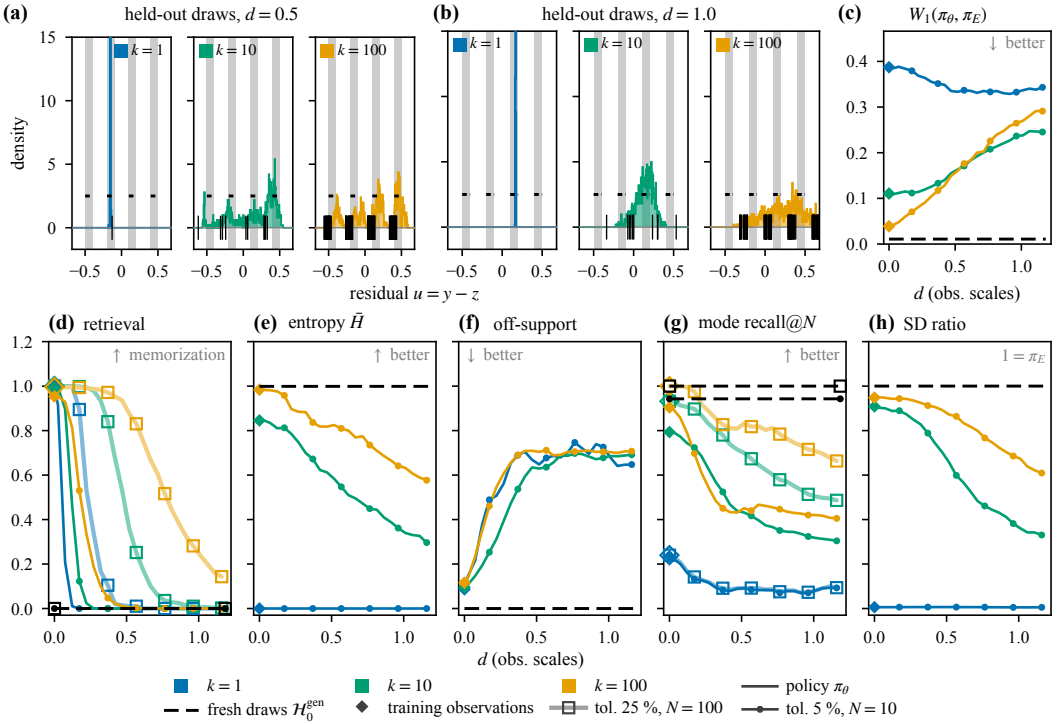


Fig. 6: The policies of fig. 4, queried at held-out conditions at distance d from the nearest training observation, in units of the mean distance between two training observations. (a), (b) Each policy's draws at a representative held-out condition at $d = 0.5$ and $d = 1.0$. Dashed segments and gray bands are π_E and its support. Short black lines mark the nearest training observation's actions, shifted into the query's residual frame. (c)–(h) The metrics of fig. 4, averaged within bins of d; diamonds at $d = 0$ are the values at training observations. Reference lines are fresh draws from the demonstrator (black dashed, $\mathcal{H}_0^{\text{gen}}$).

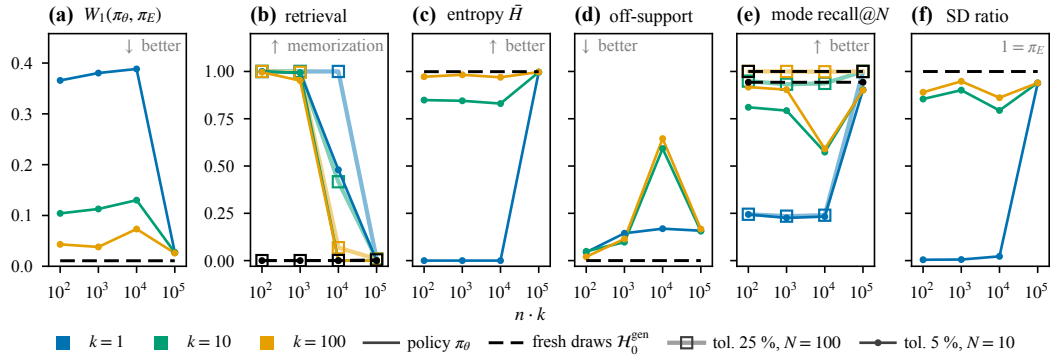


Fig. 7: The metrics of fig. 4 against the total dataset size $n \cdot k$, one curve per k, averaged over training observations. Reference lines are fresh draws from the demonstrator (black dashed, $\mathcal{H}_0^{\text{gen}}$).

B Vine Sampling

This appendix gives the vine sampling procedure of sec. 5 in full (alg. 1).

Algorithm 1: Vine sampling

input : expert demonstrator π_E, environment, re-query probability p, query multiplicity k

1 $\mathcal{D} \leftarrow \emptyset$
2 **for** *each episode* **do**
3 **while** *episode not done* **do**
4 form the observation history $\mathbf{o}$
5 record the state s
6 **if** $\xi \sim \mathrm{Unif}[0, 1] < p$ **then**
7 **for** $j = 1, \ldots, k$ **do**
8 $\mathbf{a}_j \sim \pi_E(\cdot \mid \mathbf{o})$
9 **if** *producing* $\mathbf{a}_j$ *modified* s **then** reset the environment to s
10 $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{o}, \mathbf{a}_1, \ldots, \mathbf{a}_k)\}$
11 **if** *replan required* **then**
12 $\tau \sim \pi_E(\cdot \mid \mathbf{o})$
13 pop the leading action of τ, and advance
14 **return** *dataset* $\mathcal{D}$

Besides vine sampling, other sampling strategies are possible; for example, the episode could continue branched along each of the k re-queried actions rather than only one. We forgo this, as it would shift the state distribution along the time axis, over-representing states later in the episode. Vine sampling labels a random fraction p of the states along a single trajectory, so it has p times as many conditions as the sliding window, distributed along the episode in the same way. It is simple to implement and can be carried out in real-world data collection when the demonstrator does not require resets, since it is only queried along states that are actually visited.

The algorithm, as written, requires that the demonstrator be queryable from any visited state s. In cases where this does not hold, e.g., systems limited by partial observability, alg. 1 can be modified to only re-query from states that the demonstrator can plan from, simply by adding a condition at line 6.

C Truck Unloading

This appendix gives the details of the truck unloading experiments of sec. 6 and their baselines.

C.1 Task and demonstrator

The task is simulated in Drake [32]. Task failure occurs upon object inter-penetration beyond a threshold, or if the low level controller cannot track the commanded joint positions within a tolerance. These failures are signs of poor policy performance, the former a simulator-based detection of in-collision motions. A failed episode's throughput is calculated using the 80s timeout as the duration. Each timestep incurs a step reward of -0.1. A reward of -100 is incurred on failure or timeout.

Each observation is an image and 6D state: the four joint positions, suction command (on/off), and a flag for whether any box is currently grasped. The policy takes in an observation history of $T_o = 2$ steps and outputs an action spanning $T_a = 32$ timesteps. Each action consists of four joint positions and a suction command. Each problem instance randomizes box sizes and configurations. By keeping the task in 2D and using a side view of the truck, we remove partial observability as a confounding factor.

The demonstrator generates plans by internally generating an ordered list of candidate grasps, attempting to plan motions to them, continuing down the list until an admissible motion is found. It randomly shuffles the order of the candidate grasps, over boxes and their faces. The demonstrator plans motions using bidirectional rapidly-exploring random trees (BiRRT) [33, 34] to generate initial motions which are subsequently refined via trajectory optimization. Two queries from one state may select different boxes, different faces of a box, different arm configurations to reach a selected grasp,

or different continuous motions to reach the grasp pose (fig. 1, top row). The expert can be re-queried at any visited state, necessary for alg. 1.

C.2 Problem instances and datasets

The policies are trained on problem instances 0 to 1,999, and expert iteration collects demonstrations on instances 2,000 to 6,999. All other evaluations use the same 600 test instances, separate from both. On the 600 test instances, the demonstrator succeeds in 95.8% of episodes with a throughput of 766 pieces per hour.

Episodes in which the demonstrator failed are kept in both datasets. It succeeded in 1,899 of the 2,000 episodes of the sliding-window dataset and 1,878 of the vine dataset's. The vine dataset stores the plans returned by the re-queries, while the sliding-window dataset stores the executed trajectory, which differs from the plan by the controller's tracking error. The expert re-queries used to measure policy error (app. C.4) are also plans, so they are the same kind of action as the vine policy's training actions. Per episode, collecting the vine dataset took 55 seconds of a 96-core machine and the sliding-window dataset took 1.8 seconds, the roughly 30-fold cost noted in sec. 7.

C.3 Policy and value networks

The `sliding window` and `vine` policies use the settings in table 2 and differ only in their dataset. `PostBC` differs only in its training targets (app. C.6).

Table 2: Diffusion policy settings.

Architecture	CNN-based U-Net of Chi et al. [10], channels $(256, 512, 1024)$
Image encoder	ResNet-18 trained from scratch, GroupNorm, spatial softmax
Diffusion steps	100 for training and sampling (DDPM)
Noise schedule	cosine
Optimizer	AdamW, learning rate 3×10^{-4} (constant), weight decay 10^{-6}
Batch size	2048
Training steps	100,000
EMA of weights	yes
Hardware	two NVIDIA H100 GPUs, about 8 h per policy

Table 3 gives the value network's settings.

Table 3: Value network settings.

Input	one image and the 6D state
Image encoder	ResNet-18 trained from scratch
State encoder	MLP
Head	MLP with a $\tanh$ output
Optimizer	Adam, learning rate 2×10^{-3}
Batch size	1024
Training steps	100,000
Hardware	one NVIDIA H100 GPU, 4.9 h

C.4 Metrics

Policy error and SD ratio (figs. 5a and 5b) compare a policy's actions with the demonstrator's at the same states. We choose these states along demonstrator episodes on the 600 test problem instances, taking the current state with probability 0.005 at each timestep, as in alg. 1. At each of the resulting 576 states we query the demonstrator 200 times. Some queries fail to return a plan, so we keep only the 507 states where all 200 succeed, thus every evaluated state has the same number of demonstrator actions.

We split the 200 demonstrator actions at each state into two sets of 100, which we call the estimate set and the reference set. The estimate set is our estimate of π_E at that state. The reference set shows what a policy that matched the demonstrator exactly would score with 100 draws, and gives the

reference lines in the figures. We draw 100 actions from each policy at each state to compute each of the metrics.

Each action spans 32 timesteps of four joint positions and a suction command, which have different units. To compute the distance between two actions, we first divide each action dimension c by the demonstrator's standard deviation ς_c in that dimension, computed over every timestep of all demonstrator actions at the 507 states, and take the root-mean-square difference over timesteps and dimensions:

$$d(\mathbf{a}, \mathbf{a}') = \sqrt{\frac{1}{T_a d_a} \sum_{h=1}^{T_a} \sum_{c=1}^{d_a} \left(\frac{\mathbf{a}[h,c] - \mathbf{a}'[h,c]}{\varsigma_c} \right)^2}. \tag{3}$$

Policy error at a state is W_1 under this distance between the 100 policy draws and the estimate set. For the SD ratio, we measure the spread of a set of actions as their root-mean-square distance from their mean, and divide the spread of the 100 policy draws by the spread of the estimate set, to get a ratio. Both metrics are averaged over the 507 states. Their reference lines are computed the same way, with the reference set in place of the policy draws.

All intervals in fig. 5 are 95% intervals. For policy error, SD ratio and throughput, they are bootstrap confidence intervals. We resample problem instances with replacement 1000 times, adding all states in each drawn instance to the mean, since states from the same problem instance may not be independent. For success rate, they are equal-tailed Bayesian credible intervals of the $\mathrm{Beta}(1 + \text{successes}, 1 + \text{failures})$ posterior.

C.5 Best-of-N search and expert iteration

Our implementation of best-of-N search draws N actions from the policy, simulates the leading 20 timesteps of each, discards candidates whose simulations fail, ranks the rest by the simulated reward plus a value network's estimate at the terminal state, and executes the best. The value network is trained on the same 2,000 demonstrator episodes used for policy training, to predict the reward-to-go given the current observation, and is shared across all methods. If every candidate fails in simulation, the episode fails. Planning time does not advance the simulation clock, so a slow search does not lower throughput.

In each iteration of expert iteration, a policy and a value network drive a best-of-100 search to collect demonstrations from 1,000 fresh problem instances. The search can be viewed as a policy improvement operator that generates demonstrations that are stronger than the raw policy's. Both the policy and value networks are then fine-tuned on this data, enabling the expert in the subsequent iteration to learn from these new best-of-N-improved demonstrations. In `SIME`'s run, SIME's noise is added in every iteration, with its standard deviation fixed at its value for the base policy. Within the expert iteration process, we apply the standard sliding-window collection for all three runs, so they otherwise differ only in the base policy that they were initialized with. While each method's expert iteration is initialized with a shared value function, the value function is fine-tuned during expert iteration, diverging across the methods. All three runs collect demonstrations on the same instances. In each iteration, the policy and the value network are fine-tuned for 10,000 steps, and data from j iterations earlier are replayed with weight 0.8^j. We do not evaluate the policies separately during expert iteration. Figs. 5e and 5f plot each policy's results on the 1,000 instances it collected, so iteration 0 is the base policy under best-of-100 on new instances.

C.6 SIME and PostBC

At inference-time, `SIME` [19] adds noise to the output of the observation encoder, annealed across the denoising timesteps of the diffusion policy. We apply it to the 100,000-step `sliding window` policy without retraining. The noise has standard deviation σs_{img}, where s_{img} is the root-mean-square standard deviation of the observation encoder's image output over the 576 states. It is scaled by a factor that falls linearly from one at the first denoising step to zero at the last, and each action draw

keeps one noise sample across all steps. This follows the SIME paper. We use $\sigma = 4$, where the SD ratio is closest to one and policy error is lowest (fig. 8).

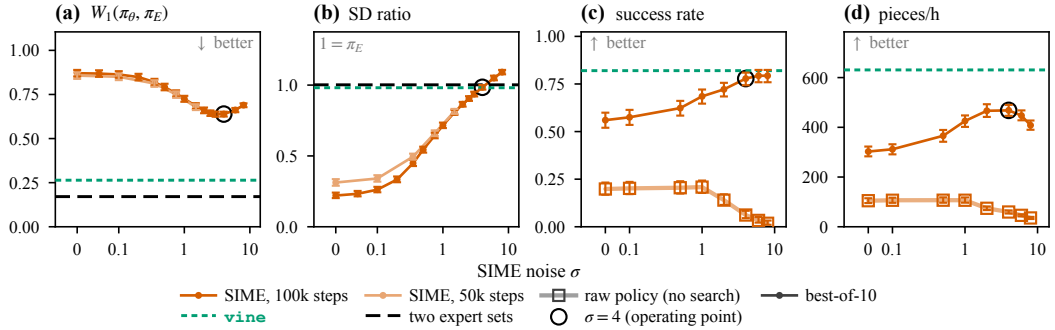


Fig. 8: The SIME baseline's noise sweep on the sliding window policy, against the noise scale σ; $\sigma = 0$ is the unmodified policy. (a) Policy error and (b) SD ratio at the states of fig. 5, for the 100,000-step checkpoint (dark) and the 50,000-step one (light). (c) Success rate and (d) throughput on the 600 test instances at the 100,000-step checkpoint, under best-of-10 (filled) and for the raw policy (open). Reference lines are two independent sets of expert re-queries (black dashed) and the vine policy at 100,000 steps, under best-of-10 in (c) and (d) (green). The circled point is $\sigma = 4$, the value used in fig. 5. Means, with 95% intervals as in fig. 5.

Where BC fits the empirical distribution of the dataset, PostBC [7] fits a posterior predictive estimate of the demonstrator's conditional distribution π_E. This distribution is wider than the empirical one where the dataset is sparse in conditioning space, and approaches it as data scales. In practice, PostBC trains an ensemble of regressors on bootstrap resamples of the demonstrations, then adds Gaussian noise to each training action, with the covariance of the ensemble's predictions at that observation. Each regressor predicts the conditional mean action, so the noise represents uncertainty about the mean. The ensemble has 10 regressors. Each is a copy of the policy's observation encoder followed by an MLP with three hidden layers of 512 units, trained on a bootstrap resample of the 2,000 episodes for 6,000 steps at batch size 256. The noise covariance spans the whole 32×5 action and is not rescaled ($\alpha = 1$ in the notation of Wagenmaker et al. [7]), and new noise is drawn each time a training sample is used. We use their default ensemble size and α without tuning, but train the ensemble for fewer epochs than they do since our 2,000 episodes are far larger than their 5–300-trajectory datasets.

C.7 Results against training step

Fig. 9 shows the sliding window and vine policies at every 10,000-step checkpoint.

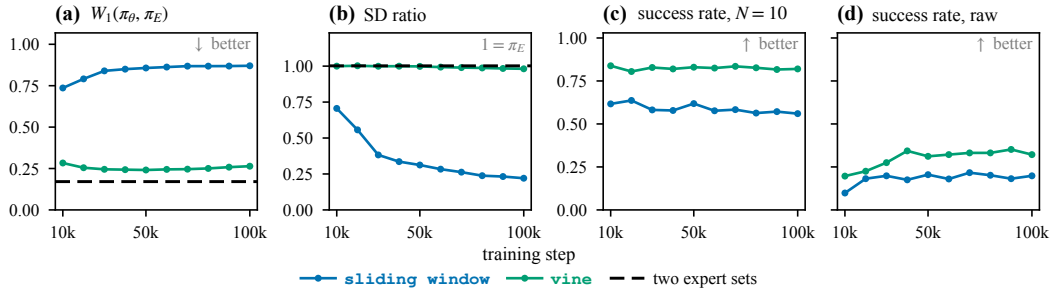


Fig. 9: The sliding window and vine policies against training step. (a) Policy error and (b) SD ratio at the states of fig. 5. (c) Success rate under best-of-10 and (d) of the raw policy, on the 600 test instances. Reference lines in (a) and (b) are two independent sets of expert re-queries (black dashed).