MUCH ADO ABOUT NOISING: DO FLOW MODELS ACTUALLY MAKE BETTER CONTROL POLICIES?

Anonymous authors
Paper under double-blind review

ABSTRACT

Generative models, like flows and diffusions, have recently emerged as popular and efficacious policy parameterizations in robotics. There has been much speculation as to the factors underlying their successes, ranging from capturing multi-modal action distribution to expressing more complex behaviors. In this work, we perform a comprehensive evaluation of popular generative control policies (GCPs) on common behavior cloning (BC) benchmarks. We find that GCPs *do not* owe their success to their ability to capture multi-modality or to express more complex observation-to-action mappings. Instead, we find that their advantage stems from *iterative computation*, as long as intermediate steps are supervised during training and this supervision is paired with a suitable level of *stochasticity*. As a validation of our findings, we show that a minimal iterative policy (**MIP**), a lightweight two-step regression-based policy, essentially matches the performance of flow GCPs. Our results suggest that the distribution-fitting component of GCPs is less salient than commonly believed, and point toward new design spaces focusing solely on control performance. Videos and supplementary materials are available at https://anonymous.4open.science/w/mip-anonymous/.

1 INTRODUCTION

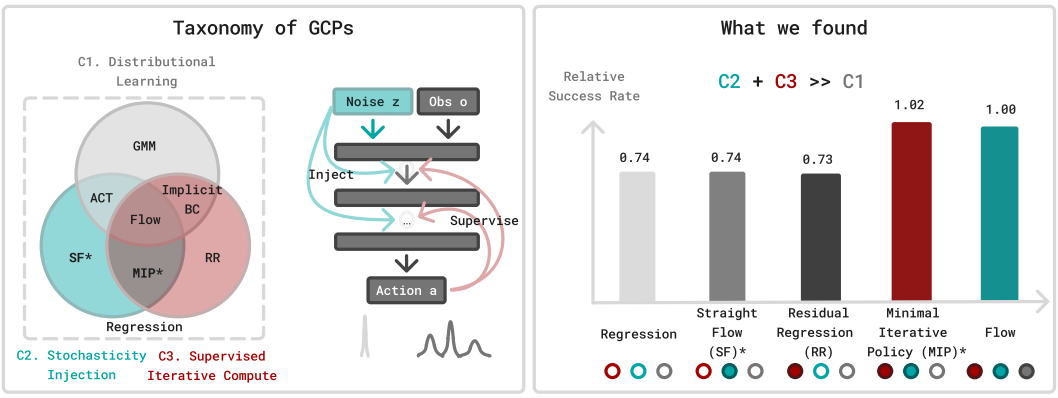

Figure 1: **The design space of GCPs.** This paper conducts a careful analysis of the design space of GCPs. After careful ablation on each component over 27 common behavior cloning benchmarks with both state and pixel-based observations (7 most challenging tasks' average relative success rate to flow is reported in the right plot), we find that the most important factor contributing to their success is the combination of *stochastic injection* (C2) and *supervised iterative computation* (C3). Surprisingly, distribution learning (C1) is the least important factor, due to the absence of learned multi-modality (Section 3.2) in single-task settings.

Long-horizon, dexterous manipulation tasks such as furniture assembly, food preparation, and manufacturing have been a holy grail in robotics. Recent large robot action models (Team et al., 2025; Black et al., 2024; Kim et al., 2024) have made substantial breakthroughs towards these goals by imitating expert demonstrations of diverse qualities. We provide a more comprehensive review of related work in Appendix A, but highlight here a key trend: while supervised learning from demonstration, also known as *behavior cloning* (BC), has been applied across domains for decades (Pomerleau,

1988), its recent success in robotic manipulation has coincided with the adoption of what we term **generative control policies** (GCPs): robotic control policies that use generative modeling architectures, such as diffusion models, flow models, and autoregressive transformers, as parameterizations of the mapping from observation to action. Given the seemingly transformative nature of GCPs for robot learning, there has been much speculation about the origin of their superior performance relative to policies trained with a regression loss, henceforth **regression control policies** (RCPs). GCPs, by modeling conditional distributions over actions, are uniquely suited to the multi-task pretraining paradigm popular in today's large robotic models. However, a number of hypotheses regarding the superiority of GCPs pertain even in the *single task* setting (Chi et al., 2023; Reuss et al., 2023):

H1. Better performance on pixel-based control

H2. Capturing multi-modality in the training data

H3. Greater expressivity due to iterative computation of the observation-to-action mapping

H4. Representation learning due to stochastic data augmentation

H5. Improved training stability and scalability

The gap between generative modeling and generative control. The objective for generative modeling in text and image domains is fundamentally different from the goal in a control task. In the former, one aims to generate high-quality and *diverse* samples from the original data distribution. In the latter, it suffices to select *any* action that leads to better downstream performance. Whereas much of the generative modeling literature has focused on the distribution of the *generated variable* (Lee et al., 2023), we aim to understand if it is necessary to reproduce the expert data distribution—for example by capturing any multi-modality—to attain strong control performance. If not, is most salient to capture about the *conditioning relationship* mapping $o \rightarrow a$?

Contributions. This paper adopts careful experimental methodology to rigorously test the key design components (Section 4) that contribute to the observed success of GCPs, and to account for the key mechanisms by which they contribute to improved performance (Section 5). We focus on a comprehensive study of the *single-task* setting in simulated environments, leaving evaluation in multi-task settings and on physical hardware to future study. Moreover, we restrict our study to flow-based GCPs trained via BC, given their popularity and adoption in industry (Black et al., 2024; Physical Intelligence et al., 2025; NVIDIA et al., 2025).

We begin by first identifying which factors *do not* contribute to the advantage of GCPs over RCPs.

Contribution 1 (**Neither multi-modality nor policy expressivity account for GCPs' success**, Section 3). Through careful benchmarking, we show that RCPs with appropriate architectures are highly competitive on both state- and image-based (H1) robot learning benchmarks (Section 3.1). Performance gaps only arise on certain tasks requiring high precision. However, we show that neither multi-modality (H2, Section 3.2) nor the ability to express more complex functions via multiple integration steps (H3, Section 3.3) satisfactorily accounts for this phenomenon.

Essential to this finding is controlling for architecture: to our knowledge, we are the first work to carefully benchmark expressive architectures popularized for Diffusion (Chi et al., 2023; Dasari et al., 2024) as regression policies. To determine what contributes to GCPs performance on these high-precision tasks (beyond architectural optimization), we parse the design space of generative control policies into three components, depicted in Figure 1 (left).

Contribution 2 (**Exposing the design space of GCPs**, Section 4). We introduce a novel taxonomy that parses the three essential design components of GCPs:

C1. *Distributional Learning*: matching a conditional distribution of actions given observations.

C2. *Stochasticity Injection*: injecting noise during training to improve the learning dynamics.

C3. *Supervised Iterative Computation*: generating output with multiple steps, each of which receives supervision during training.

With this taxonomy in hand, Section 4.1 introduces a family of algorithms, each of which lies along a spectrum between GCPs and RCPs by exhibiting different combinations of the above components. While we find that neither C2 nor C3 in isolation improve over regression, we find their combination yields a policy whose performance is competitive with flow, leading to our next contribution.

Contribution 3 (**`MIP`: the power of C2+C3**, Sections 4.1 and 4.2). As an algorithmic ablation that only combines C2+C3, we devise a *minimal iterative policy* (**`MIP`**), which invokes only two iterations, one-step of stochasticity during training, and deterministic inference. Despite its simplicity, **`MIP`** essentially matches the performance of flow-based GCPs across state-, pixel- and 3D point-cloud-based BC tasks, exposing that the combination of C2+C3 is responsible for the observed success of GCPs.

As described in Remark 4.1, **`MIP`** is substantively distinct from flow-map-based models (Boffi et al., 2025a;b), including consistency models (Song et al., 2023; Kim et al., 2023) and their extensions (Geng et al., 2025; Frans et al., 2024), in that the latter do satisfy C1, and require training over a continuum of noise levels.

Contribution 4 (**Attributing the benefits of C2+C3**, Section 5). We identify that a property we term *manifold adherence* captures the inductive bias of GCPs and **`MIP`** relative to RCPs, even in the absence of lower validation loss. We explain how this property is a useful proxy for closed-loop performance in control tasks. Finally, we expose how C3, through iterative computation, encourages manifold adherence, but only if stochasticity during training (C2) is present to mitigate compounding errors across iteration steps (as described in Section 5.2).

Manifold adherence in Section 5.1 measures the generated action's plausibility given out of distribution observations, where only off-manifold component is evaluated rather than the distance to the neighbors (Pari et al., 2021). Note that manifold adherence reflects a favorable inductive bias during learning, rather than brute expressivity of more complex behavior (H3). Moreover, C2 provides more of a supporting role to C3, rather than enhancing data-augmentation in its own right (H4). In addition, we find that C2+C3 also enhance scaling behavior (H5), likely due to better model utilization through decoupling across iterations. Finally, we identify that the subtle interplay between architecture choice, policy parameterization and task can affect performance by an even greater magnitude than the choice of policy parametrization (Section 5.3).

Takeaway. In robotic applications, our findings suggest that the distributional formulation of GCPs—sampling from a *distribution* of actions given observations—is the least important facet that contributes to their success. Rather, our work highlights that C2+C3 offer an exciting and under-explored sandbox for future algorithm design in continuous control and beyond.

2 Preliminaries

We consider a continuous control setting with observations $o \in O$ and actions $a \in A$ where O is the observation space and A is the action space. We learn a policy $\pi : O \to \Delta(A)$ from observations to (distributions over) actions to maximize the probability of success $J(\pi)$ on a given task, which we refer to as "performance." We consider the performance of policies learned via BC—that is, supervised learning from a distribution of (observation, actions pairs) drawn from a training distribution p_{train}. In applications, the actions a are often a short-open loop sequence of actions, or *action-chunks*, which have been shown to work more effectively for complex tasks with end-effector position commands (Zhao et al., 2023). See Appendix A for an unabridged related work.

Regression Control Policies (RCPs). A historically common policy choice for BC is regression control policies (RCPs) (Pomerleau, 1988; Bain & Sammut, 1995; Ross et al., 2011; Osa et al., 2018), given by a deterministic map $\pi : O \to A$. In applications, it is parameterized by a neural network π_θ and trained so as to minimize the L_2-loss on training data:

$$\pi_\theta \approx \arg\min_\theta \mathbb{E}\|\pi_\theta(o) - a\|^2, \quad (o, a) \sim p_{\text{train}}. \tag{2.1}$$

Generative Control Policies (GCPs). Generative control policies (GCPs) parameterize a *distribution* of actions a given an observation o. This is often accomplished in practice by representing the policy π_θ with a generative model such as a diffusion (Chi et al., 2023), flow (Zhang et al., 2024), or tokenized autoregressive transformer (Shafiullah et al., 2022). Given their popularity, we focus on flow-based GCPs (flow-GCPs). A flow-GCP learns a conditional flow field (Lipman et al., 2023; Chisari et al., 2024; Nguyen et al., 2025) $b : [0, 1] \times A \times O \to A$ by minimizing the objective

$$b_\theta \approx \arg\min_\theta \mathbb{E}\|b_t(I_t \mid o) - \dot{I}_t\|^2, \quad t \sim \text{Unif}([0, 1]), \;\; z \sim \text{N}(0, \mathbf{I}), \tag{2.2}$$

where again $(o, a) \sim p_{\text{train}}$, $I_t = ta + (1-t)z$ is the stochastic interpolant between the training action a and noise variable z, and where $\dot{I}_t = a - z$ is the time derivative of I_t. We note that this is a special case of the stochastic interpolant framework (Albergo & Vanden-Eijnden, 2022; Albergo et al., 2023; 2024), which permits a larger menu of design decisions. A flow model then predicts an action by integrating a flow. In the limit of infinite discretization steps, this amounts to sampling $a \sim \pi_\theta(\cdot \mid o)$ by sampling $z \sim \mathrm{N}(0, \mathbf{I})$, and then setting $a = a_1$, where $\{a_t\}_{t\in[0,1]}$ solves the ODE:

$$\tfrac{\mathrm{d}}{\mathrm{d}t} a_t = b_t(a_t \mid o) \qquad \text{with initial condition} \qquad a_0 = z. \tag{2.3}$$

In practical implementation, sampling is conducted via discretized Euler integration (see Appendix K.2 for details). This yields a policy $a = \pi_\theta(z, o)$ which is a deterministic function of the initial noise z and the observation o. All experiments, unless otherwise stated, perform 9 integration steps. We reiterate that other GCPs, e.g. based on diffusion models and autoregressive transformers, have been studied elsewhere. We choose to focus on flow models due to their state-of-the-art performance (Chi et al., 2023; Chisari et al., 2024; Zhang et al., 2024) and deployment in industry (Black et al., 2024; Physical Intelligence et al., 2025; NVIDIA et al., 2025).

Multi-Modality in Robot Learning. Past work has conjectured that for salient robotic control tasks, $p_{\text{train}}(a \mid o)$ exhibit *multi-modality*, i.e. the conditional distribution of a given o has multiple modes (Shafiullah et al., 2022; Zhao et al., 2023; Florence et al., 2022). This motivated the earliest use of GCPs (Chi et al., 2023) (H2). Section 3.2 calls into question the extent to which GCPs do in fact learn multi-modal distributions of $a \mid o$ on popular benchmarks.

3 Representational Capacity Does Not Explain GCPs Performance

This section demonstrates that neither advantages on pixel-based control (H1), nor multi-modality (H2), nor improved expressivity (H3) fully account for the GCPs performance relative to RCPs. Appendix H addresses other hypotheses, such as k-nearest neighbor approximation.

3.1 GCPs Mainly Outperform RCPs on a Few, High-Precision Tasks

We first isolate the tasks in which they exhibit stronger performance by comparing across 27 popular BC benchmarks (detailed in Appendix D.1), encompassing diverse data quality, modalities (**state**, **point clouds** and **image**), and domains (e.g., `MetaWorld`, `Robomimic`, `Adroit`, `D4RL`). Crucially, we implement RCPs using the *exact same* architectures as their corresponding flow models by simply setting the noise level and initial noise to zero: $z = 0$, $t = 0$, and study three widely-used architectures (`Chi-Transformer`, `Sudeep-DiT`, `Chi-UNet`; detailed in Appendix D.2). This architectural alignment enables RCPs to benefit from the sophisticated network designs typically reserved for GCPs, ensuring a truly fair comparison. Under this controlled comparison, we discover GCPs and RCPs achieve comparable performance on the vast numerical majority of state- and image-based imitation learning benchmarks, but performance gaps emerge on a few tasks that require particularly high precision. To account for architecture's substantial impact on final performance, we report best-case results with optimal architecture selection in Fig. 2 and the worst-case results using the poorest-performing architecture in Appendix D.3. Across both evaluations, GCPs only outperform RCPs by more than a 5% on a handful of tasks that mainly require high precision.

3.2 GCPs' Performance Does Not Arise From Multi-Modality

Earlier literature suggested that capturing multi-modality, as defined in Section 2, was precisely the root of the observed performance benefits of GCPs (Chi et al., 2023; Reuss et al., 2023). However, examining Fig. 2, we see that many tasks which have been understood to be multimodal (e.g., `Push-T`) do not show substantial performance gaps between RCPs and GCPs. On the other hand, RCPs and GCPs differ only on tasks that demand high precision (e.g. `Tool-Hang`, `Transport`). In this section, we provide additional evidence that **multimodality is not the main factor responsible for witnessed performance advantages of GCPs**.

Evidence A: GCPs exhibit unstructured action distributions. For fixed observations, we draw multiple action samples by denoising from different initial latents and visualize the resulting action set with their Q values $Q(a, o)$. We deliberately choose *symmetry-critical* or *high-ambiguity* states to

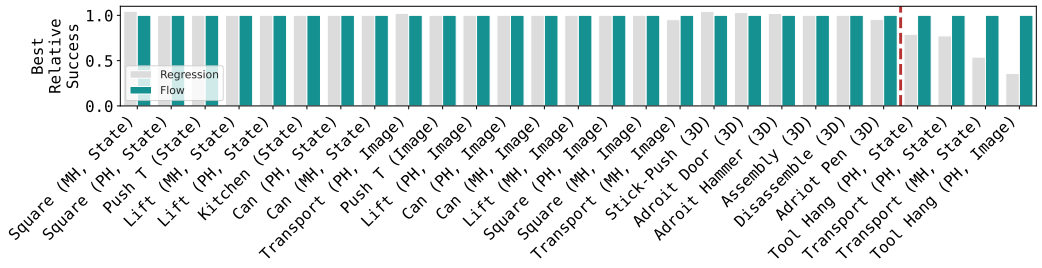

Figure 2: **Relative performance of RCPs compared to GCPs across common benchmarks.** For each task, we implement `Chi-Transformer`, `Sudeep-DiT` and `Chi-UNet`. For each architecture, we average performance of the best training checkpoint across three seeds. We then report the performance of the best-performing architecture, chosen individually for both RCPs and GCPs. For Flow, we always do 9 step Euler integrations, where its performance plateaued. For readability, RCPs success rates are plotted relative to flow, with flow normalized to performance of 1 per task. Tasks are grouped by observation modality, and ordered by relative RCPs performance. Red dashed line indicates threshold at which RCP attains $< 95\%$ success of GCPs.

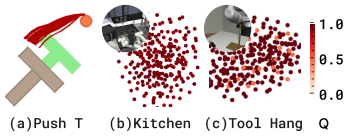

Figure 3: **A. Visualized action distribution with Q values.** Distinct modes are **not** observed in planned actions even at symmetric and ambiguous states. (`Kitchen` and `Tool-Hang`, t-SNE visualization.) In `Push-T`, we all trajectories goes to one side. For the rest, there is no clear clustering of actions or Q.

Task	$z \equiv 0$	$\mathrm{N}(0, \mathbf{I})$	Mean z
`Push-T`	0.97	0.97	0.95
`Kitchen`	0.99	0.99	0.97
`Tool-Hang`	0.78	0.80	0.76

Table 1: **B. Performance comparison of different sampling strategies**. We compare sampling $z = 0$, $z \sim \mathrm{N}(0, \mathbf{I})$, and mean over 64 $z^{(i)} \sim \mathrm{N}(0, \mathbf{I})$. Different sampling strategies show minor performance difference, indicating absence of distinct action modes.

Dataset	`Flow`	`Reg.`
Original	0.78	0.58
Deterministic	0.72	0.64

Table 2: **C. GCPs outperforms RCPs with deterministic experts.** Policy average success rate over 3 architectures, 3 seeds and 3 architectures given different dataset: one from original human demonstration and another collected by rolling out a flow policy in deterministic mode starting from zero noise.

maximize potential multi-modality: (a) `Push-T` at the symmetry axis of the T-shape, where taking the left or right path is equivalent, (b) `Kitchen` from an initial state with multiple first-subtask choices, and (c) `Tool-Hang` at the insertion pre-contact pose where human demonstrators pause for varying durations. In (a-c) we observe *single* clusters rather than distinct modes (high-dimensional actions visualized with t-SNE); see Fig. 3. Moreover, adherence to action cluster means do not correlate with performance: We color-code actions by Q-value, i.e. Monte-Carlo-estimated rewards-to-go (Appendix F.1). Highest returns are distributed evenly across samples.

Evidence B: Taking mean actions does not meaningfully degrade GCPs' performance. We evaluate flow policy's performance with three sampling strategies: zero noise $a = \pi(z = 0, o)$, stochastic sampling $a = \pi(z, o), z \sim \mathrm{N}(0, I)$, and *mean action* $a = \mathbb{E}_{z \sim \mathrm{N}(0,I)}[\pi(z, o)]$ (via Monte Carlo approximation). If the learned distribution were strongly multi-modal, or if their distributions lied on a manifold whose *curvature* was crucial to task success, the conditional mean would *collapse* modes and severely degrade performance. However, Table 1 shows that replacing stochastic sampling with the mean action only slightly affects performance, indicating absence of distinct action modes.

Evidence C: GCPs outperform RCPs on certain tasks even with deterministic experts. To fully remove any residual multi-modality, we recollect the dataset with trained flow policy evaluated in deterministic mode ($z = 0$) detailed in Appendix F.2. The new dataset is fully deterministic because action labels are provided by a deterministic policy evaluated in a deterministic environment. While the gap in performance between GCPs and RCPs shrinks somewhat, we still find that GCPs still outperforms RCPs, as in Table 2, suggesting that capturing some "hidden" stochasticity or multi-modality in the data does not suffice to explain the gap between the two.

Multi-modality and data coverage. The absence of observed multimodality is likely attributable to the large observation dimension of tasks relative to total number of demonstrations, such we rarely see two "conflicting" actions for nearby observation vectors (note: to grid a space of dimension d requires 2^d points). Some degree of "hidden" multi-modality may still be present, as indicated by

the slight narrowing of the performance gap in Table 2. Still, our central claim is that multi-modality is not *sufficient* to explain the full difference in performance.

3.3 Limitations of the expressivity of GCPs in the absence of multimodality

Multi-step generative models are believed to leverage iterative computation to express more complex probability distributions, both because more generation steps in outperform their few-step counterparts (Ho et al., 2020; Song et al., 2021a; Zhang & Chen, 2022; Nichol & Dhariwal, 2021), and via analogies between iterative computation in generative models and neural network depth (Chen et al., 2018). Yet for control, we need only capture the mapping from observation to a single effective action, rather than the distribution over all possible actions. Our findings thus have suggested that to represent such a mapping, we need not represent complex action distributions.

We now demonstrate that in the absence of multi-modality (as shown in Section 3.2), GCPs cannot express more complex mappings from the conditioning variable o to the generated variable a than RCPs can. We begin by considering a ground-truth conditional flow field $b_t^\star(o \mid a)$. Let $\pi_\theta^\star(z, o)$ represent the exactly integrated $b^\star$ from initial noise z to generated variable a. Given the absence of multi-modality (Section 3.2), we assume that the distribution of $a \mid o$ is κ-log-concave (Appendix I), satisfied by many classical unimodal distributions. We prove that the Lipschitz constant of $\pi_\theta^\star(z, o)$ with respect to o, a measure of the expressivity of the $o \to a$ mapping, is bounded by that of $b_t^\star$:

Theorem 1 (Informal). *Let $\|\cdot\|$ denote either the matrix operator or Frobenius norm, and suppose that the distribution of $a \mid o$ is κ-log-concave. Moreover, suppose that the flow field $b_t^\star(a \mid o)$ is L-Lipschitz: $\|\nabla_o b_t^\star(a \mid o)\| \leq L$. Then, with infinite integration steps, $\|\nabla_o \pi_\theta^\star(z, o)\| \leq L \cdot \sqrt{1 + \kappa^{-1}}$.*

See Appendix I for a formal statement and proof. A classical example of a log concave distribution is $a \mid o \sim \mathrm{N}(\mu(o), \frac{1}{\kappa})$; as long as the variance $1/\kappa$ is bounded *above* (even in the limit of a Dirac), there is at most a constant-multiplicative factor increase in the Lipschitz constant. When training a flow, $b_t^\star(a \mid o)$ is approximated by the neural network. Thus, in the prototypical unimodal example of κ-log-concave distributions, GCPs are not arbitrarily more expressive than RCPs. In fewer words: **more integration steps, even infinitely many, need not be equivalent to greater network depth**.

To verify our theoretical prediction, we quantify learned policies' Lipchitz constants with a zeroth-order proxy: starting from dataset states s_t with observation o_t, we inject small Gaussian perturbations in the executed action to reach a *feasible* nearby state $s_{t+1}^{(i)}$ with observation $o_{t+1}^{(i)}$, then measure input–output sensitivity via finite differences of the policy around the perturbed states (full algorithm and per-architecture results in Appendix E). This construction (i) avoids reliance on noisy higher-order gradients in complex architectures, and (ii) keeps evaluations on feasible observation to prevent conflating expressivity with model error on dynamically infeasible states. As predicted by our theory, GCPs are not strictly more expressive than RCPs as shown in Table 3. On the contrary, RCPs show increased Lipschitz constants off the manifold of training data, ruling out the assumption that GCPs win due to expressing policies with greater sensitivity to the input variable. We note that our methodology, which perturbs actions rather than states, is compatible pixel observations.

Method	Push-T		Kitchen	Tool-Hang	
	State	Image	State	State	Image
`Regression`	0.90	0.55	14.07	1.71	1.65
`Flow`	0.45	0.20	12.43	1.41	1.37

Table 3: **Policy Lipschitz constant comparison.** Lipschitz constant is averaged over 100 states.

4 What design components enable the success of GCPs?

Thus far, we have established that GCPs shine on high-precision, complex tasks, but that their performance is not directly attributable to multi-modality or more complex $o \to a$ mappings. To understand the actual factors contributing to GCP success, we elucidate three key algorithmic components (Fig. 1). Section 4.1 below proposes algorithmic variants which ablate these components. We find that minimal iterative policy (**`MIP`**, Components 2 and 3) is the reduced variant which matches the performance of flow (Section 4.2), whereas other variants match or perform worse than regression.

Component 1. **Distributional learning** denotes training a model to fit a conditional distribution $a \sim \pi_\theta(o)$ of actions given observations, as opposed to deterministic predictions (i.e., $a = \pi_\theta(o)$). [1]

Component 2. **Stochasticity injection** denotes the injection of additional stochastic inputs into the neural network during training time (e.g., the variable z in Eq. (2.2)).

Component 3. **Supervised Iterative Computation (SIC)** denotes the iterative refinement of predictions by feeding the previous outputs into the same network again during inference, and providing *supervision signals* at every step of the generation procedure at training time. For example, in flow GCPs, we integrate a supervised flow field $b_t(a_t \mid o)$ over time to get the final action a, and that b_t receives an independent supervisory signal for each t at training time (Eq. (2.2)).

4.1 Intermediates between RCPs and GCPs

We introduce a range of policies which lie along the spectrum between RCP and flow-based GCPs. Our findings in Section 3.2 suggest that Component 1, capturing general $a \mid o$ distributions, maybe the least salient for performance. Hence, all of the variants that follow abandon Component 1 and are *not consistent* for estimating a general conditional distributions of $a \mid o$. Instead, each variant exhibits some combination of Components 2 and 3. In following exposition, we make explicit the dependence of the network π_θ on t, understanding that the networks predict actions, not velocities.

We derive all variants by starting with a **two-step denoising** (**`TSD`**) policy. As discussed in Remark 4.1, this parametrization is superficially similar to, but substantively different than, popular flow-map/consistency/shortcut models (Boffi et al., 2025b). **`TSD`** performs two steps of denoising, one from zero, and a second from a fixed index $t_\star = .9$:

$$\pi_\theta^{\texttt{TSD}} \approx \arg\min_\theta \mathbb{E}\left(\|(\pi_\theta(o, I_0, t=0) - (t_\star)^{-1} I_{t_\star})\|^2 + \|(\pi_\theta(o, I_{t_\star}, t_\star) - a)\|^2\right). \tag{4.1}$$

where $(o, a) \sim p_{\text{train}}, z \sim \mathrm{N}(0, \mathbf{I})$, and $I_t = ta + (1-t)z$ is the same interpolant used in flow models, and where $t_\star = .9$ is fixed. The normalization by $t_\star$ in Eq. (4.1) comes from the identity $t_\star a = \mathbb{E}_z[I_{t_\star}]$. We then sample $\hat{a}_0^{\texttt{TSD}} \leftarrow \pi_\theta(o, z, 0)$ and $\hat{a}^{\texttt{TSD}} \leftarrow \pi_\theta(o, t_\star \hat{a}_0^{\texttt{TSD}} + (1-t_\star)z, t_\star)$. In practice, we find that $\pi^{\texttt{TSD}}$ is equivalent to a minimal policy which only adds training noise in the second step and has no stochasticity at inference time, which we call the minimal iterative policy.

Minimal Iterative Policy (`MIP`, ours). **`MIP`**, representing Components 2 and 3, is trained via

$$\pi_\theta^{\texttt{MIP}} \approx \arg\min_\theta \mathbb{E}(\|(\pi_\theta(o, I_0 = 0, t=0) - a)\|^2 + \|(\pi_\theta(o, I_{t_\star}, t_\star) - a)\|^2), \tag{4.2}$$

where $(o, a) \sim p_{\text{train}}, z \sim \mathrm{N}(0, \mathbf{I}), t_\star := .9$. At inference time, we compute:

$$\hat{a}_0^{\texttt{MIP}} \leftarrow \pi_\theta^{\texttt{MIP}}(o, 0, t=0), \quad \hat{a}^{\texttt{MIP}} \leftarrow \pi_\theta^{\texttt{MIP}}(o, t_\star \hat{a}_0^{\texttt{MIP}}, t_\star). \tag{4.3}$$

Minimal iterative policy provides a *minimal* implementation that still exhibits competitive performance with flow. Starting, with **`TSD`** and replace $(t_\star)^{-1} I_{t_\star}$ in the first term of the loss in Eq. (4.1) with its expectation $a = (t_\star)^{-1}\mathbb{E}[I_{t_\star}]$. We set the initial noise $I_0 = 0$ to be zero, so that z only contributes to the second training loss. Finally, we sample with $z = 0$ to isolate the effect of adding stochasticity at training time, without stochasticity at inference time (c.f. Table 1) . Since we provide supervision for both first step $\pi_\theta^{\texttt{MIP}}(o, I_0 = 0, t = 0)$ and second step $\pi_\theta^{\texttt{MIP}}(o, I_0 = I_{t_\star}, t = t_\star)$ with ground truth action a, **`MIP`** also exemplifies SIC in its simplest form.

Remark 4.1. (**`MIP`** v.s. Shortcut Models (Boffi et al., 2025a;b; Song et al., 2023; Geng et al., 2025)) While **`TSD`** and **`MIP`** share properties of a flow model while only conducting inference, these are fundamentally different than few-step shortcut/flow-map models. The latter integrate a flow field across a continuum of noise levels $t \in [0, 1]$, and therefore can correctly learn general distributions (i.e. they satisfy Component 1). On the other hand, **`TSD`** and **`MIP`** are trained to predict the conditional mean of the interpolant, which is not a valid objective for distribution fitting. The performance of **`MIP`** supports our overall theme that, in robotic control applications, faithfully capturing the full conditional distribution over actions is not needed for control performance.

[1] Note that Component 1 refers to *training* a model to fit a conditional distribution, not necessarily to the sampling. For example, training b_θ via flow model but conducting deterministic inference with $\Phi_{\theta,\text{eul}}(z = 0 \mid o)$ is still considered distributional learning.

Additional methods. Straight Flow (**SF**, ours), representing only Component 2, further simplifies **MIP** to a single stage by setting the interpolation index $t_\star = 1$ and removing the second term: $\pi_\theta^{\mathbf{SF}} \approx \mathbb{E}\|\pi_\theta(o, z, t = 0) - a\|^2$. Inference is performed in a single step, by setting $a = \pi_\theta^{\mathbf{SF}}(o, z, t = 0)$. Like RCPs, the optimal **SF** policy is the conditional mean of $a \mid o$. The only difference between the two is injection of stochastic input z during training. Our experiments with **SF** precisely isolate this effect—for example, determining if the additional stochasticity during training improves learning dynamics, or behaves like data augmentation. As with **MIP**, we set $z = 0$, as stochasticity at inference time has little effect on policy performance. Finally, we study residual regression (**RR**), which replaces $I_{t_\star}$ in Eq. (4.2) with its expectation over z: $\mathbb{E}[I_{t_\star}] = t_\star a$. This preserves SIC (Component 3) yet removes stochasticity injection. Full details are provided in Appendix C.

To summarize, minimal iterative policy (**MIP**), straight-flow (**SF**) and residual regression (**RR**) represent all combinations of Components 2 and 3 without exhibiting Component 1.

4.2 Components 2 and 3 drive performance: **MIP** matches **FLOW**

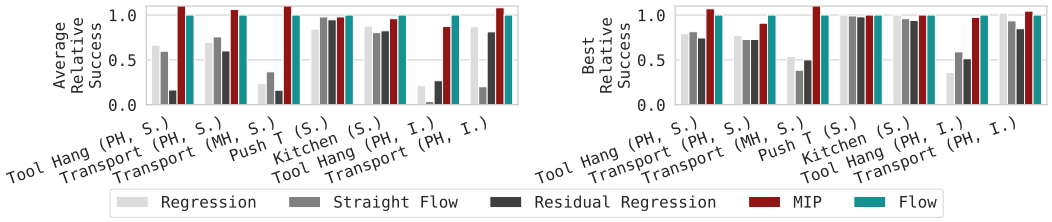

Figure 4: **Performance comparison between MIP and its variants.** Average relative success rate on worst architecture and the best relative success rate on optimal architecture are reported. "S": state; "I": image.

Based on the design space parsing in Section 4, we are able to systematically ablate different design components' contribution to the final performance in Fig. 4. Our evaluation shows that either stochasticity injection (Component 2, exhibit by **SF**) or supervised iterative computation (Component 3, exhibited by **RR**) in isolation do not match the success of GCPs. **MIP**, being the only method which combines *supervised* iterative computation and stochasticity injection, achieves success on par with flow. Thus we conclude: the performance of GCPs comes from combining stochastic injection and iterative computation. Distributional training appears to be the least important factor.

Remark 4.2. Appendix C.2 exhibits two further variants which preserve Components 2 and 3: one that does not supervise intermediate steps, and a second which does not condition a time step $t_\star$. The latter does not enable network to learn separate functions across time steps. Both perform even worse than regression, confirming the importance of supervision of intermediate steps and decoupling network behavior across time steps.

5 Explaining the Performance of **MIP**

5.1 Manifold ddherence, not reconstruction, drives performance

MIP, and the absence of multimodality, suggest a better ability to approximate the expert more accurately on training data. We test this by evaluating the L_2-error, i.e., reconstruction error, on validation set. Surprisingly, we find that **MIP**, **Flow**, and RCP exhibit the *same* validation loss; hence validation loss does predict their relative performance. Appendix G.1 reveals that validation loss doesn't correlate with performance across other axes of variation. Indeed, policy performance requires taking good actions on *o.o.d. states* under compounding error at deployment time (Simchowitz et al., 2025).

Metric	Regression	SF	RR	MIP	Flow
Off-manifold L_2	0.067	0.063	0.062	0.054	0.042
Validation L_2	0.290	0.234	0.224	0.195	0.217

Table 4: **Comparison of different methods on manifold adherence and reconstruction error.** Results are averaged across 3 different architectures and 32 states on state-based `Tool-Hang` with deterministic dataset.

Thus, we study a proxy which reflects performance in o.o.d. situations. We perturb expert trajectories in dataset as described in Appendix E.1, and evalute a novel metric that we call the *off-manifold norm*. Informally, this measures the projection error of a predicted action a onto the space spanned by expert actions at neighboring states; see Appendix G.2 a for formal definition. Our metric assesses the quality of actions under simulated compounding error. Table 4 reports both L_2 validation loss and off-manifold L_2 norm for different methods: while all methods achieve low validation loss, only **MIP** and **Flow** are able to achieve low off-manifold L_2 norm, indicating their better manifold adherence. As **SF** does not exhibit the same benefit, we conclude that supervised iterative computation facilitates projection onto the manifold of expert actions by refining the prediction across sequential steps. Appendix J provides additional confirmation of this hypothesis: GCPs are no better than RCP at fitting high frequency functions, but exhibit lower on-manifold error, suitably defined.

5.2 Stochasticity stabilizes iterative computation

SF matching regression, whilst **RR** underperforming regression, suggests that sequential action generation is highly brittle in the absence of stochasticity (Permenter & Yuan, 2024). Our findings support the hypothesis that stochasticity injection serves to provide "coverage" of the generative process. Specifically, we can think of learning to perform two-stage action generation as an "internal" behavior cloning problem (Ren et al., 2024) under the dynamics induced by the generative process. Injecting stochasticity amounts to enhancing coverage of the action $\hat{a}_0$ in the first step of **MIP**, thus enable iterative improvement with more NFEs (Appendix D.6). Its benefits are analogous to trajectory noising effective in other behavior cloning applications (Laskey et al., 2017; Block et al., 2023; 2024; Simchowitz et al., 2025). Similar benefits are found in the improved sensitivity analysis of diffusion relative to flows (Albergo et al., 2024).

5.3 Supervised iterative computation scales better with model capacity, if the architecture is right

Regression, enjoys stronger relative performance at the smallest model sizes but scales more poorly than flow and **MIP** with increased model capacity (Fig. 5). We conjecture that supervised iterative computation can better utilize larger models, both by introducing more supervision steps at training, and by providing more parameters to represent different computations at successive generation steps.

We conclude by emphasizing the role of *architecture design*. To showcase its importance, we ablate the performance of different method's average performance across both the 3 architectures above, and the more traditional `MLP` and `RNN` architectures, implemented with modern best practices including FiLM conditioning (Perez et al., 2018), and skip-connections (He et al., 2016)/LayerNorm (Ba et al., 2016) where appropriate (details in Appendix D.2). As demonstrated in Fig. 5, the combination of training method and architecture design has a strong yet somewhat erratic effect on both GCPs and RCP performance. In `Tool-Hang`, RCP achieves the best performance with an `MLP` architecture. In `Transport`, `MLP` with flow can even outperform more expressive architectures like `Chi-Transformer`. The coupling between training and architecture choice highlights the importance of controlling architecture design when comparing across methods.

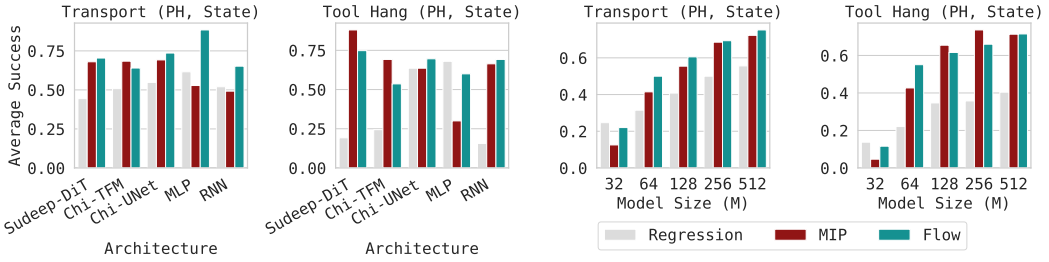

Figure 5: **Architecture and model size ablation.** Success rate are averaged across 3 seeds and 5 checkpoints on `Tool-Hang` and `Transport` tasks. Left 2 plots: architecture ablation. Right 2 plots: Model size ablation.

REFERENCES

Michael S. Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. *arXiv preprint arXiv:2209.15571*, 2022.

Michael S. Albergo, Nicholas M. Boffi, and Eric Vanden-Eijnden. Stochastic interpolants: A unifying framework for flows and diffusions. *arXiv preprint arXiv:2303.08797*, 2023.

Michael Samuel Albergo, Mark Goldstein, Nicholas Matthew Boffi, Rajesh Ranganath, and Eric Vanden-Eijnden. Stochastic interpolants with data-dependent couplings. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 2009.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Michael Bain and Claude Sammut. A framework for behavioural cloning. *Machine Intelligence*, 1995.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Adam Block, Dylan J Foster, Akshay Krishnamurthy, Max Simchowitz, and Cyril Zhang. Butterfly effects of sgd noise: Error amplification in behavior cloning and autoregression. *arXiv preprint arXiv:2310.11428*, 2023.

Adam Block, Ali Jadbabaie, Daniel Pfrommer, Max Simchowitz, and Russ Tedrake. Provable guarantees for generative behavior cloning: Bridging low-level stability and high-level behavior. *Advances in Neural Information Processing Systems*, 2024.

Nicholas M. Boffi, Michael S. Albergo, and Eric Vanden-Eijnden. Flow map matching with stochastic interpolants: A mathematical framework for consistency models. *arXiv preprint arXiv:2406.07507*, 2025a.

Nicholas M. Boffi, Michael S. Albergo, and Eric Vanden-Eijnden. How to build a consistency model: Learning flow maps via self-distillation. *arXiv preprint arXiv:2505.18825*, 2025b.

Andrew Brock, Jeff Donahue, and Karen Simonyan. Large Scale GAN Training for High Fidelity Natural Image Synthesis. *arXiv preprint arXiv:1809.11096*, 2019.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *arXiv preprint arXiv:2106.01345*, 2021.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 2018.

Sitan Chen, Sinho Chewi, Jerry Li, Yuanzhi Li, Adil Salim, and Anru R. Zhang. Sampling is as easy as learning the score: Theory for diffusion models with minimal data assumptions. *arXiv preprint arXiv:2209.11215*, 2023.

Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

Cheng Chi, Zhenjia Xu, Chuer Pan, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, Russ Tedrake, and Shuran Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. *arXiv preprint arXiv:2402.10329*, 2024.

Eugenio Chisari, Nick Heppert, Max Argus, Tim Welschehold, Thomas Brox, and Abhinav Valada. Learning robotic manipulation policies from point clouds with conditional flow matching. In *Conference on Robot Learning (CoRL)*, 2024.

Max Daniels. On the contractivity of stochastic interpolation flow. *arXiv preprint arXiv:2504.10653*, 2025.

Sudeep Dasari, Oier Mees, Sebastian Zhao, Mohan Kumar Srirama, and Sergey Levine. The ingredients for robotic diffusion transformers. *arXiv preprint arXiv:2410.10088*, 2024.

Zibin Dong, Yifu Yuan, Jianye Hao, Fei Ni, Yi Ma, Pengyi Li, and Yan Zheng. CleanDiffuser: An easy-to-use modularized library for diffusion models in decision making. *arXiv preprint arXiv:2406.09509*, 2024.

Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Conference on robot learning*. PMLR, 2022.

Kevin Frans, Danijar Hafner, Sergey Levine, and Pieter Abbeel. One step diffusion via shortcut models. *arXiv preprint arXiv:2410.12557*, 2024.

Zhengyang Geng, Mingyang Deng, Xingjian Bai, J. Zico Kolter, and Kaiming He. Mean flows for one-step generative modeling. *arXiv preprint arXiv:2505.13447*, 2025.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 2020.

Chengyang He, Xu Liu, Gadiel Sznaier Camps, Guillaume Sartoretti, and Mac Schwager. Demystifying diffusion policies: Action memorization and simple lookup table alternatives, 2025. URL https://arxiv.org/abs/2505.05787.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

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

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, 2020.

Xixi Hu, Qiang Liu, Xingchao Liu, and Bo Liu. Adaflow: Imitation learning with variance-adaptive flow-based policies. *Advances in Neural Information Processing Systems*, 2024.

Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in neural information processing systems*, 2022.

Tsung-Wei Ke, Nikolaos Gkanatsios, and Katerina Fragkiadaki. 3D diffuser actor: Policy diffusion with 3d scene representations. *arXiv preprint arXiv:2402.10885*, 2024.

Dongjun Kim, Chieh-Hsin Lai, Wei-Hsiang Liao, Naoki Murata, Yuhta Takida, Toshimitsu Uesaka, Yutong He, Yuki Mitsufuji, and Stefano Ermon. Consistency trajectory models: Learning probability flow ode trajectory of diffusion. *arXiv preprint arXiv:2310.02279*, 2023.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. OpenVLA: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

Michael Laskey, Jonathan Lee, Roy Fox, Anca Dragan, and Ken Goldberg. Dart: Noise injection for robust imitation learning. In *Conference on robot learning*. PMLR, 2017.

Holden Lee, Jianfeng Lu, and Yixin Tan. Convergence of score-based generative modeling for general data distributions. In *International Conference on Algorithmic Learning Theory*. PMLR, 2023.

Fanqi Lin, Yingdong Hu, Pingyue Sheng, Chuan Wen, Jiacheng You, and Yang Gao. Data scaling laws in imitation learning for robotic manipulation. *arXiv preprint arXiv:2410.18647*, 2024.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations (ICLR)*, 2023.

Songming Liu, Lingxuan Wu, Bangguo Li, Hengkai Tan, Huayu Chen, Zhengyi Wang, Ke Xu, Hang Su, and Jun Zhu. RDT-1B: A diffusion foundation model for bimanual manipulation. *arXiv preprint arXiv:2410.07864*, 2024.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.

Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. DPM-Solver++: Fast solver for guided sampling of diffusion probabilistic models. *Machine Intelligence Research*, 2025.

Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *Conference on Robot Learning (CoRL)*, 2021.

Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2023.

Khang Nguyen et al. Flowmp: Learning motion fields for robot planning with conditional flow matching. *arXiv preprint arXiv:2503.06135*, 2025.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International conference on machine learning*. PMLR, 2021.

NVIDIA, Johan Bjorck, Fernando Castaneda, 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.

Takayuki Osa, Joni Pajarinen, Gerhard Neumann, J Andrew Bagnell, Pieter Abbeel, Jan Peters, et al. An algorithmic perspective on imitation learning. *Foundations and Trends in Robotics*, 2018.

Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya 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)*. IEEE, 2024.

Jyothish Pari, Nur Muhammad Shafiullah, Sridhar Pandian Arunachalam, and Lerrel Pinto. The surprising effectiveness of representation learning for visual imitation. *arXiv preprint arXiv:2112.01511*, 2021.

Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, 2018.

Frank Permenter and Chenyang Yuan. Interpreting and improving diffusion models from an optimization perspective. *arXiv preprint arXiv:2306.04848*, 2024.

Karl Pertsch, Kyle Stachowicz, Brian Ichter, Danny Driess, Suraj Nair, Quan Vuong, Oier Mees, Chelsea Finn, and Sergey Levine. FAST: Efficient Action Tokenization for Vision-Language-Action Models. *arXiv preprint arXiv:2501.09747*, 2025.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, et al. $\pi_{0.5}$: A Vision-Language-Action Model with Open-World Generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Dean A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *Advances in Neural Information Processing Systems*, 1988.

Aaditya Prasad, Kevin Lin, Jimmy Wu, Linqi Zhou, and Jeannette Bohg. Consistency Policy: Accelerated Visuomotor Policies via Consistency Distillation. *arXiv preprint arXiv:2405.07503*, 2024.

Allen Z Ren, Justin Lidard, Lars L Ankile, Anthony Simeonov, Pulkit Agrawal, Anirudha Majumdar, Benjamin Burchfiel, Hongkai Dai, and Max Simchowitz. Diffusion policy policy optimization. *arXiv preprint arXiv:2409.00588*, 2024.

Moritz Reuss, Maximilian Xiling Li, Xiaogang Jia, and Rudolf Lioutikov. Goal-conditioned imitation learning using score-based diffusion policies. In *Proceedings of Robotics: Science and Systems (RSS)*, 2023.

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

Nur Muhammad Shafiullah, Zichen Cui, Ariuntuya Arty Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 2022.

Juyi Sheng, Ziyi Wang, Peiming Li, and Mengyuan Liu. MP1: Meanflow tames policy learning in 1-step for robotic manipulation. *arXiv preprint arXiv:2507.10543*, 2025.

Max Simchowitz, Daniel Pfrommer, and Ali Jadbabaie. The pitfalls of imitation learning when actions are continuous. *arXiv preprint arXiv:2503.09722*, 2025.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021a.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising Diffusion Implicit Models. *arXiv preprint arXiv:2010.02502*, 2022.

Yang Song and Prafulla Dhariwal. Improved Techniques for Training Consistency Models. *arXiv preprint arXiv:2310.14189*, 2023.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-Based Generative Modeling through Stochastic Differential Equations. *arXiv preprint arXiv:2011.13456*, 2021b.

Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. *arXiv preprint arXiv:2303.01469*, 2023.

Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Pannag Sanketi, Quan Vuong, Ted Xiao, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

TRI LBM Team, Jose Barreiros, Andrew Beaulieu, Aditya Bhat, Rick Cory, Eric Cousineau, Hongkai Dai, et al. A Careful Examination of Large Behavior Models for Multitask Dexterous Manipulation. *arXiv preprint arXiv:2507.05331*, 2025.

Jingyun Yang, Zi-ang Cao, Congyue Deng, Rika Antonova, Shuran Song, and Jeannette Bohg. EquiBot: Sim(3)-equivariant diffusion policy for generalizable and data efficient learning. *arXiv preprint arXiv:2407.01479*, 2024.

Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3D diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. *arXiv preprint arXiv:2403.03954*, 2024.

Qinglun Zhang, Zhen Liu, Haoqiang Fan, Guanghui Liu, Bing Zeng, and Shuaicheng Liu. Flowpolicy: Enabling fast and robust 3d flow-based policy via consistency flow matching for robot manipulation. *arXiv preprint arXiv:2412.04987*, 2024.

Qinsheng Zhang and Yongxin Chen. Fast sampling of diffusion models with exponential integrator. *arXiv preprint arXiv:2204.13902*, 2022.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

Zuyuan Zhu and Huosheng Hu. Robot learning from demonstration in robotic assembly: A survey. *Robotics*, 2018.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*. PMLR, 2023.

A Related Works

Robotic Behavior Cloning. Behavior cloning (BC), also known as learning from demonstrations (LfD), has become a popular paradigm to enable robots to conduct complex, diverse and long-horizon manipulation tasks by learning from expert demonstrations (Argall et al., 2009; Zhu & Hu, 2018; Zhao et al., 2023; Chi et al., 2024; Lin et al., 2024). In parallel, "robot foundation models" scale BC with internet-pretrained vision-language transformer-based backbones (Brohan et al., 2022; Zitkovich et al., 2023; O'Neill et al., 2024) and large-scale teleoperation datasets (Kim et al., 2024; Team et al., 2024). More recently, to better model continuous actions, generative models like diffusion and flow have been adopted to replace the tokenization method in transformers to achieve more expressive policies (NVIDIA et al., 2025; Black et al., 2024; Physical Intelligence et al., 2025; Liu et al., 2024). This work focuses on the generative modeling part of the behavior cloning pipeline, ablating the key design choices that lead to the success of generative control policies.

Generative Modeling. The recent success of behavior cloning policies is built upon a rapid evolution of generative modeling techniques, starting from tokenization methods (Brown et al., 2020; Chen et al., 2021; Pertsch et al., 2025) and adversarial methods (Brock et al., 2019; Goodfellow et al., 2020; Ho & Ermon, 2016). Later, probabilistic generative models with iterative computation like diffusion models (Ho et al., 2020; Song et al., 2021b; Lu et al., 2025; Song et al., 2022; Nichol & Dhariwal, 2021; Karras et al., 2022) became a popular choice for generative modeling thanks to their better training stability and sampling quality. Flow models (Lipman et al., 2023; Albergo & Vanden-Eijnden, 2022; Liu et al., 2022) and consistency/shortcut models (Song et al., 2023; Song & Dhariwal, 2023; Meng et al., 2023; Boffi et al., 2025a; Geng et al., 2025) were later developed to achieve faster sampling while maintaining the expressivity of diffusion models. Though there have been extensive studies on probabilistic generative modeling's effectiveness in image and text generation (Lee et al., 2023; Chen et al., 2023), its mechanism in control, especially the key design choices, are still opaque in decision making.

Generative Control Policies. To model diverse and complex behaviors, GCPs parameterize the relationship between observations and actions as a distribution rather than a deterministic function. Early works use transformers with tokenizers (Chen et al., 2021; Shafiullah et al., 2022), energy functions (Florence et al., 2022; Dasari et al., 2024) and VAEs (Zhao et al., 2023) to parameterize the distribution. Diffusion models (Reuss et al., 2023; Chi et al., 2023; Ke et al., 2024; Dong et al., 2024; Janner et al., 2022; Yang et al., 2024) were introduced for their better expressivity of complex and multi-modal behaviors, followed by flow-based (Zhang et al., 2024; Black et al., 2024; Physical Intelligence et al., 2025) and flow-map/consistency-model/shortcut-model-based acceleration methods (Hu et al., 2024; Prasad et al., 2024; Sheng et al., 2025).

Theoretical Literature on GCPs. Block et al. (2024) established that GCPs can imitate arbitrary expert distributions. Given our findings on the absence of multi-modality, a more closely related theoretical findings is that of Simchowitz et al. (2025), which elucidates how GCPs can circumvent certain worst-case compounding error phenomena in continuous-control imitation learning. Though the proposed mechanism is different, that finding is conceptually similar to our own: GCPs benefits arise from their favorable out-of-distribution properties, rather than raw expressivity of fitting in-distribution expert behavior.

B Previous Works' Connection with GCP's Taxonomy.

We classify GCPs into three components: distributional learning, stochasticity injection, and supervised iterative computation. Starting from regression, it has none of the three components. To model a more complex distribution, Gaussian Mixture Model (GMM) (Zhu & Hu, 2018) was used to parameterize the distribution, trained with cross entropy loss. To make the network be able to represent more complex distirbutions, prior to diffusion, non-parametric method like VAEs (Zhao et al., 2023) was used to parameterize the distribution, trained with reconstruction loss. During the training, a latent variables is predicted to predict the style the motion by mapping it from a noise z. Another line of work try to improve the policy expressivity by introducing iterative compute, like implicit behavior cloning (Florence et al., 2022; Dasari et al., 2024). The idea is to allow the network predict the energy function of the action rather the action itself. Compared to diffusion, the major difference is that they do not explicitly injecting noise during training. Lastly, flow-based

GCPs (Zhang et al., 2024; Black et al., 2024; Physical Intelligence et al., 2025), which holds all the three components and demonstrate state-of-the-art performance on popular benchmarks. In this paper, we look into a new combination that haven't been explored before, which is the combination of stochasticity injection and supervised iterative computation.

C Additional Policy Parametrizations

This section further elaborates the design space of **MIP** in stochasticity injection, iterative computation and intermediate supervision.

C.1 Full Abalation of MIP Variants

This section formally describes the training process of all **MIP** with different stochasticity injection and supervised iterative computation design.

Residual Regression (RR) removes all stochasticity in training and the training objective is:

$$\pi_\theta^{\texttt{RR}} \approx \arg\min_\theta \mathbb{E}_{(o,a)\sim p_{\text{train}}, z\sim \mathrm{N}(0,\mathbf{I})} \left(\|(\pi_\theta(o, I_0 = 0, t = 0) - t_\star a)\|^2 + \|(\pi_\theta(o, \mathrm{sg}(\pi_\theta(o, I_0 = 0, t = 0)), t_\star) - a)\|^2 \right).$$

Two-Step Denoising (TSD) The training objective is:

$$\pi_\theta^{\texttt{TSD}} \approx \arg\min_\theta \mathbb{E}_{(o,a)\sim p_{\text{train}}, z\sim \mathrm{N}(0,\mathbf{I})} \left(\|(\pi_\theta(o, I_0, t = 0) - t_\star a)\|^2 + \|(\pi_\theta(o, \mathrm{sg}(\pi_\theta(o, I_0, t = 0)) + (1 - t_\star)z, t_\star) - a)\|^2 \right).$$

where $I_0 = z$. Compared to **MIP**, **TSD** adds stochasticity to both first step training.

MIP with Data Augmentation (MIP-Dagger) To understand the importance of decoupling for enabling iterative computation, we propose an additional variant of **MIP** that lies between **MIP** and **RR**, where the two steps are partially coupled. Since the training method of second iteration is similar to data augmentation, we call this variant **MIP-Dagger**:

$$\pi_\theta^{\texttt{MIP-Dagger}} \approx \arg\min_\theta \mathop{\mathbb{E}}_{(o,a)\sim p_{\text{train}}, z\sim \mathrm{N}(0,\mathbf{I})} (\|(\pi_\theta(o, I_0 = 0, t = 0) - t_\star a)\|^2 + \|(\pi_\theta(o, t_\star \mathrm{sg}(\pi_\theta(o, I_0 = 0, t = 0)) + (1 - t_\star)z, t_\star) - a)\|^2),$$

where the major difference compared to **MIP** is the second step takes in the interpolant between first step output and noise rather than the action and noise.

MIP without intermediate supervision (MIP-NoSupervision) To understand the effect of intermediate supervision on iterative computation, we propose one variant of **MIP** that removes the supervision of intermediate computation steps while preserving stochasticity injection at training time, named **MIP-NoSupervision**:

$$\pi_\theta^{\texttt{MIP-NoSupervision}} \approx \arg\min_\theta \mathop{\mathbb{E}}_{(o,a)\sim p_{\text{train}}, z\sim \mathrm{N}(0,\mathbf{I})} (\|(\pi_\theta(o, t_\star \mathrm{sg}(\pi_\theta(o, I_0 = 0, t = 0)) + (1 - t_\star)z, t_\star) - a)\|^2),$$

where the first step's output is unsupervised.

MIP without t conditioning By removing t conditioning in **MIP**, it degenerates to **SF**. Here we present the multi-step integration process for straight flow when action distribution is Dirac delta. The integrator from s to t is:

$$a_t = \frac{t-s}{1-s}\pi_\theta(o, s \cdot a_s) + \frac{1-t}{1-s}a_s$$

C.2 Experiment Results

We benchmark all methods on the `Tool-Hang` task, given it is the one with the largest gap between RCP and GCPs. From Table 5, we can see that the important part is to add stochasticity injection between two iterations, and intermediate supervision is also important to realize the potential of iterative computation.

Method	NFEs	Success Rate
TSD	2	0.80
MIP	2	0.80
MIP-NoSupervision	2	0.42
MIP-Dagger	2	0.64
RR	2	0.54
SF	1	0.54
SF	3	0.55
SF	9	0.52

Table 5: Success rates across different **MIP** variants and **RR** on `Tool-Hang` task over 5 checkpoints across 3 architectures.

D Control Experiments

D.1 Task Settings

This section introduces all the tasks presented in the main paper. To reach a sound conclusion, use common benchmarks appears in previous works:

Robomimic Robomimic (Mandlekar et al., 2021) is a large-scale robotic manipulation benchmark designed to study imitation learning and offline reinforcement learning. It contains five manipulation tasks (`Lift`, `Can`, `Square`, `Transport`, `Tool-Hang`) with *proficient human (PH)* teleoperated demonstrations, and for four of them, additional *mixed proficient/non-proficient human (MH)* demonstration datasets are provided (9 variants in total). We report results on both *state-based* and *image-based* observations, since these two modalities pose distinct challenges. Among the tasks, `Tool-Hang` requires extremely precise end-effector positioning and fine-grained contact control, while `Transport` demands high-dimensional control and coordination over extended horizons.

Push-T `Push-T` (Florence et al., 2022) is adapted from the Implicit Behavior Cloning (IBC). The task involves pushing a T-shaped block to a fixed target location using a circular end-effector. Randomized initializations of both the block and the end-effector introduce significant variability. The task is contact-rich and requires modeling complex object dynamics for precise block placement. Two observation variants are considered: (*i*) raw RGB image observations and (*ii*) state-based observations containing object pose and end-effector position.

Kitchen The Franka `Kitchen` environment is designed to test the ability of IL and offline RL methods to perform long-horizon, multi-task manipulation. It includes 7 interactive objects, with human demonstration data consisting of 566 sequences, each completing 4 sub-tasks in arbitrary order (e.g., opening a cabinet, turning a knob). Success is measured by completing as many of the demonstrated sub-tasks as possible, regardless of order. This setup explicitly introduces both

short-horizon and long-horizon multimodality, requiring policies to generalize across compositional tasks.

`MetaWorld` `MetaWorld` is a large-scale suite of diverse manipulation tasks built in MuJoCo, where agents must perform challenging object interactions using a robotic gripper. We adopt the 3D observation setting using point cloud representations, ported from the DP3 framework (Ze et al., 2024), to better evaluate geometric reasoning and spatial generalization. Tasks in MetaWorld are categorized into different difficulty levels, with benchmarks testing few-shot adaptation and multi-task transfer learning.

`Adroit` `Adroit` is a suite of dexterous manipulation tasks featuring a 24-DoF anthropomorphic robotic hand. Tasks include pen rotation, door opening, and object relocation, all of which demand precise, coordinated multi-finger control. Following DP3 (Ze et al., 2024), we use point cloud observations to capture fine-grained 3D object-hand interactions. Policies are trained using VRL3, highlighting the challenges of high-dimensional control and sim-to-real transfer in dexterous manipulation.

D.2 Architecture Design

We study four policy backbones—`Chi-Transformer`, `Sudeep-DiT`, `Chi-UNet`, `RNN`, and `MLP` —under a common training recipe and data interface. Unless otherwise specified, *all models are capacity-matched to ∼20M parameters* to enable fair comparison.

`Chi-UNet` is adopted from Diffusion Policy (Chi et al., 2023) which built on top of 1D temporal U-Net (Janner et al., 2022) with FiLM conditioning (Perez et al., 2018) on observation o and flow time t. `Chi-UNet` has a strong inductive bias for the temporal structure of the action and tends to smooth out the action.

`Chi-Transformer` follows the time–series diffusion transformer from Diffusion Policy (Chi et al., 2023), where the noisy action tokens a_t form the input sequence and a *positional embedding* of the flow time t is prepended as the first token; observations o are mapped by a shared MLP into an observation-embedding sequence that conditions the decoder stack. Compared to `Chi-UNet`, `Chi-Transformer` uses token-wise self-attention over the whole action sequence, thus can model less-smooth and more complex actions.

`Sudeep-DiT` is a DiT-style (Diffusion Transformer) conditional noise network specialized for policies adopted from DiT-Policy (Dasari et al., 2024): observation o are first encoded into observation vectors; the flow time t is embedded via *positional embedding*; an encoder–decoder transformer then fuses these with initial noise z to predict next action. The key ingredient of `Sudeep-DiT` is replacing standard cross-attention with *adaLN-Zero* blocks—adaptive LayerNorm modulation using the mean encoder embedding and the time embedding, with zero-initialized output-scale projections—stabilizing diffusion training at scale. Compared to `Chi-Transformer`, `Sudeep-DiT` has adaLN-based conditioning (instead of vanilla cross-attention) and an explicit encoder-decoder split, yielding better training stability.

`RNN` The `RNN` backbone processes sequences with a stacked LSTM/GRU. For each action time step in the chunk, the input vector concatenates: the current noised action a_t, a time embedding for t, and a observation embedding for o. The RNN outputs are fed to a MLP head with LayerNorm+ApproxGELU+Dropout blocks before output the action with final linear head. All linear and recurrent weights use *orthogonal initialization* (biases zero), and RNN layer dropout is applied when depth>1.

`MLP` The `MLP` backbone flattens the action and observation, appending the time embedding. Each mlp block has LayerNorm, ApproxGELU and Dropout blocks with residual connection and *orthogonal* weight initialization throughout. Each block output is then modulated with FiLM conditioning.

`DP3` built on top of `Chi-UNet` with extra 3d perception encoder. We use the exact same architecture as 3D diffusion policy (Ze et al., 2024).

Model hyperparameters In the main experiments, we align the model capacity to 20M parameters for default if not specified, with detailed hyperparameters report in Table 6.

Backbone	Heads	Layers	Embedding dim	Dropout
`Sudeep-DiT`	8	8	256	0.1
`Chi-UNet`	–	–	256	–
`Chi-Transformer`	4	8	–	0.1
`RNN`	–	8	512	0.1
`MLP`	–	8	512	–

Table 6: Model hyperparameters.

D.3 Full Results for Flow and Regression Comparison

In the paper, we only present the aggregated results across 3 architectures. Figure 6 present the full results across all architectures with different training methods.

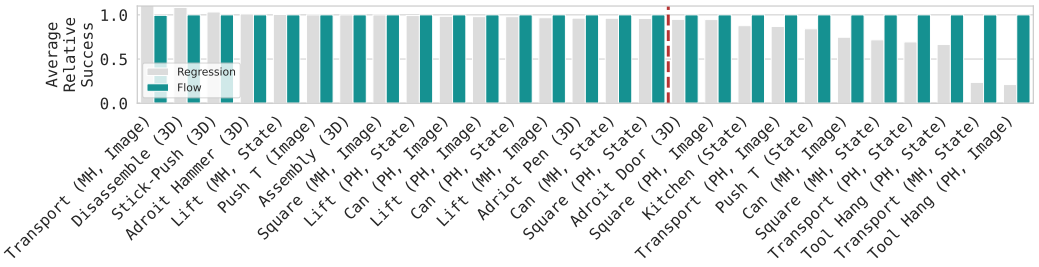

Figure 6: **Relative performance of RCP compared to GCP across common benchmarks (worst-case architecture).** For each task, we implement `Chi-Transformer`, `Sudeep-DiT` and `Chi-UNet`. For each architecture, we average performance of the last 5 training checkpoints across three seeds. We then report the performance of the worst-performing architecture, chosen individually for both RCP and GCP, to demonstrate method robustness. For **`Flow`**, we always do 9 step Euler integrations, where its performance plateaued. For readability, RCP success rates are plotted relative to flow, with flow normalized to performance of 1 per task. Tasks are grouped by observation modality, and ordered by relative RCP performance. Red dashed line indicates threshold at which RCP attains $< 95\%$ success of GCP.

D.4 Dataset Quality Ablation

GCPs are believed to handle data with diverse quality better. To test that assumption, we manually corrupt the expert dataset and inject stochactity and multi-modality in to the dataset. In Table 7, we compare 4 different datasets (3 of them collected by ourselves). In the collected dataset, we manually inject noise to the policy and add delay the policy from time to time to introduce multi-modality that is common in the real world.

D.5 Full Results for **`MIP`** and its variants

For `Kitchen`, the task has multiple stages. In the main results, we only report the performance of the last stage since it is the most challenging one. Table 11 shows the performance comparison across different design choices on `Kitchen` task.

D.6 Different Method's Performance with Different Number of Function Evaluations

We also provide detailed evaluation on different method's scaling behavior given different amount of online computation budgets. Table 12 highlights that only **`MIP`** and **`Flow`** benefit from iterative computate.

Architecture	Method	NFEs	Delayed & Noisy Policy (Worst Quality)	Delayed Policy (Mixed Quality)	Zero-Flow (Better Quality)	Proficient Human (Good Quality)
Chi-UNet	**Regression**	1	0.70/0.63	0.80/0.72	0.76/0.65	0.76/0.62
Chi-UNet	**SF**	1	0.70/0.62	0.82/0.76	0.84/0.77	0.62/0.38
Chi-UNet	**MIP**	2	0.80/0.72	0.82/0.61	0.74/0.64	0.80/0.68
Chi-UNet	**Flow**	9	0.76/0.68	0.74/0.50	0.76/0.54	0.84/0.70
Chi-Transformer	**Regression**	1	0.38/0.22	0.40/0.31	0.42/0.26	0.50/0.24
Chi-Transformer	**SF**	1	0.46/0.35	0.68/0.50	0.56/0.41	0.62/0.48
Chi-Transformer	**MIP**	2	0.56/0.49	0.70/0.54	0.64/0.56	0.72/0.68
Chi-Transformer	**Flow**	9	0.56/0.34	0.54/0.48	0.62/0.49	0.68/0.54
Sudeep-DiT	**Regression**	1	0.42/0.29	0.36/0.28	0.42/0.32	0.30/0.19
Sudeep-DiT	**SF**	1	0.66/0.41	0.60/0.54	0.72/0.57	0.68/0.50
Sudeep-DiT	**MIP**	2	0.66/0.56	0.74/0.58	0.70/0.61	0.86/0.78
Sudeep-DiT	**Flow**	9	0.56/0.45	0.66/0.58	0.72/0.65	0.78/0.68

Table 7: Performance comparison across different methods and data quality levels. We evaluate on the task `Tool-Hang` with state observations using 10M parameter networks. Success rates are reported as averages over 5 checkpoints across 3 seeds.

Architecture	Method	Lift		Can		Square		Transport		Tool-Hang	Push-T	Kitchen
		mh	ph	mh	ph	mh	ph	mh	ph			
Sudeep-DiT	**Flow**	**1.00**/0.99	**1.00/1.00**	**1.00**/0.94	**1.00**/1.00	0.88/0.75	**1.00**/0.94	0.40/0.27	0.80/0.70	0.86/0.75	**1.00/1.00**	0.98/0.96
Sudeep-DiT	**Regression**	**1.00**/0.99	**1.00/1.00**	0.92/0.90	**1.00**/0.98	0.72/0.53	0.94/0.86	0.12/0.06	0.50/0.44	0.52/0.39	**1.00/1.00**	0.98/0.92
Sudeep-DiT	**SF**	**1.00**/0.99	**1.00/1.00**	0.94/0.90	**1.00**/0.98	0.84/0.70	0.96/0.88	0.18/0.14	0.56/0.48	0.70/0.59	**1.00/1.00**	0.96/0.91
Sudeep-DiT	**MIP**	**1.00**/0.99	**1.00/1.00**	0.98/0.95	**1.00/1.00**	0.90/0.81	0.98/**0.94**	0.44/0.38	0.76/0.68	**0.92/0.88**	**1.00/1.00**	**1.00/0.97**
Chi-Transformer	**Flow**	**1.00/1.00**	**1.00/1.00**	**1.00**/0.93	**1.00**/0.98	0.78/0.74	0.96/0.89	0.44/0.34	**0.88**/0.64	0.68/0.54	**1.00/1.00**	**1.00**/0.96
Chi-Transformer	**Regression**	**1.00**/0.99	**1.00**/0.99	0.98/0.92	**1.00**/0.96	0.74/0.61	0.92/0.85	0.28/0.20	0.68/0.51	0.40/0.36	**1.00/1.00**	0.98/0.91
Chi-Transformer	**SF**	**1.00/1.00**	**1.00/1.00**	0.98/0.94	**1.00**/0.99	0.76/0.70	0.94/0.84	0.30/0.24	0.62/0.54	0.60/0.55	**1.00/1.00**	0.96/0.92
Chi-Transformer	**MIP**	**1.00/1.00**	**1.00/1.00**	0.96/0.95	**1.00**/1.00	0.86/0.73	0.96/0.89	0.42/0.37	0.80/0.68	0.76/0.69	**1.00/1.00**	0.98/0.96
Chi-UNet	**Flow**	**1.00/1.00**	**1.00/1.00**	**1.00**/0.98	**1.00/1.00**	0.90/0.78	0.98/0.94	0.52/0.40	0.80/**0.73**	0.84/0.70	**1.00/1.00**	**1.00**/0.97
Chi-UNet	**Regression**	**1.00/1.00**	**1.00/1.00**	**1.00**/0.96	**1.00**/0.99	**0.94/0.82**	**1.00**/0.91	0.22/0.16	0.64/0.55	0.68/0.64	**1.00/1.00**	0.92/0.88
Chi-UNet	**SF**	**1.00/1.00**	**1.00/1.00**	**1.00**/0.97	**1.00**/0.99	0.88/0.76	0.96/0.89	0.26/0.18	0.64/0.52	0.58/0.42	**1.00/1.00**	0.86/0.79
Chi-UNet	**MIP**	**1.00/1.00**	**1.00/1.00**	**1.00/0.98**	**1.00**/0.99	0.92/0.81	**1.00/0.94**	**0.62/0.46**	0.80/0.69	0.80/0.64	**1.00/1.00**	**1.00**/0.96

Table 8: Performance comparison of Flow and Regression methods across different **state-based** robotic manipulation tasks. For each task, we report the best checkpoint performance / averaged performance over last 5 checkpoints. Each experiment is run with 3 seeds and we report the average performance across all seeds.

E Lipschitz Constant Study Details

E.1 Lipschitz Evluation Method

We note that not all inputs o are dynamically feasible, and our dataset lies only on a narrow manifold of the observation space. Therefore, we must carefully evaluate the Lipschitz constant on the feasible observation space to avoid conflating model expressivity with errors arising from infeasible states. To ensure feasibility, instead of directly perturbing the state, we perturb the action and then roll it out in the environment. This guarantees that both the perturbed state and the resulting observation remain feasible.

In practice, we identify states that exhibit the highest ambiguity of actions in the dataset, referred to as *critical states*. For each critical state, we inject Gaussian noise $\eta \sim \mathcal{N}(0, \epsilon^2 I)$ into the normalized action, unnormalize it, and then roll it out. We select 100 critical states from the dataset. For each state, we perturb the corresponding expert action a with 64 independent Gaussian samples.

Let o denote the next nominal observation after applying the nominal action a. After rolling out the perturbed actions, we obtain perturbed observations $o^{(1)}, \ldots, o^{(N_{\text{perturb}})}$. The policy then predicts the perturbed actions $a^{(i)} = \pi(o^{(i)})$. To ensure comparability across different states and tasks, we evaluate the Lipschitz constant with respect to normalized observations $\bar{o} = \frac{o-\mu_o}{\sigma_o}$ and normalized actions $\bar{a} = \frac{a-\mu_a}{\sigma_a}$. Finally, the Lipschitz constant is estimated using a zeroth-order approximation:

$$L \approx \max_i \frac{\|\bar{a}^{(i)} - \bar{a}\|_2}{\|\eta\|_2}. \tag{E.1}$$

Full version of above process is stated in Algorithm 1.

Architecture	Method	Lift		Can		Square		Transport		Tool-Hang	Push-T
		mh	ph	mh	ph	mh	ph	mh	ph		
Sudeep-DiT	**Flow**	**1.00**/1.00	**1.00**/1.00	0.96/0.94	**1.00**/0.99	0.82/0.76	0.96/**0.94**	0.32/0.20	0.84/0.83	**0.78**/0.57	**1.00/1.00**
Sudeep-DiT	**Regression**	**1.00**/0.99	**1.00/1.00**	0.92/0.81	**1.00/1.00**	0.74/0.67	0.94/0.84	0.14/0.08	0.74/0.56	0.28/0.18	**1.00/1.00**
Sudeep-DiT	**SF**	**1.00**/0.99	**1.00/1.00**	0.94/0.90	**1.00**/0.98	0.84/0.70	0.96/0.88	0.14/0.10	0.76/0.63	0.46/0.40	**1.00/1.00**
Sudeep-DiT	**MIP**	**1.00**/0.99	**1.00/1.00**	0.98/0.95	**1.00/1.00**	0.90/0.81	0.98/**0.94**	0.34/0.28	0.90/0.84	0.76/**0.66**	**1.00/1.00**
Chi-Transformer	**Flow**	**1.00**/0.99	**1.00**/1.00	**1.00**/0.93	**1.00**/0.99	0.70/0.66	**0.98**/0.93	0.22/0.13	0.80/0.77	0.74/0.61	**1.00/1.00**
Chi-Transformer	**Regression**	**1.00**/0.98	**1.00**/0.98	**1.00**/0.94	**1.00**/0.96	**0.90**/0.72	**0.98**/0.90	0.40/0.27	0.94/0.87	0.44/0.36	**1.00/1.00**
Chi-Transformer	**SF**	**1.00**/0.99	**1.00/1.00**	0.98/0.94	**1.00**/0.99	0.76/0.70	0.94/0.84	0.30/0.24	0.86/0.70	0.34/0.31	**1.00/1.00**
Chi-Transformer	**MIP**	**1.00**/0.98	**1.00**/1.00	0.96/0.91	0.98/0.68	0.72/0.28	0.90/0.25	0.16/0.04	0.86/0.69	0.52/0.40	**1.00/1.00**
Chi-UNet	**Flow**	**1.00**/1.00	**1.00/1.00**	**1.00/0.97**	**1.00**/0.98	**0.90**/0.79	0.96/0.90	0.34/0.32	0.80/0.64	0.70/0.62	**1.00/1.00**
Chi-UNet	**Regression**	**1.00**/0.96	**1.00**/0.99	0.84/0.70	0.98/0.87	0.74/0.66	0.94/0.86	0.16/0.12	0.78/0.64	0.30/0.23	**1.00/1.00**
Chi-UNet	**SF**	**1.00/1.00**	**1.00/1.00**	**1.00**/0.97	**1.00**/0.99	0.78/0.66	0.96/0.89	0.26/0.18	0.46/0.13	0.06/0.02	**1.00/1.00**
Chi-UNet	**MIP**	**1.00/1.00**	**1.00/1.00**	**1.00**/0.95	**1.00**/0.98	0.86/**0.81**	0.96/0.92	**0.58/0.42**	**0.96/0.91**	0.56/0.50	**1.00/1.00**

Table 9: Performance comparison of Flow and Regression methods across different **image-based** robotic manipulation tasks. For each task, we report the best checkpoint performance / averaged performance over last 5 checkpoints. Each experiment is run with 3 seeds and we report the average performance across all seeds.

Architecture	Method	Adroit			MetaWorld		
		Hammer	Door	Pen	Stick-Push	Assembly	Disassemble
DP3	Flow	0.96 ± 0.02	$\mathbf{0.60 \pm 0.06}$	$\mathbf{0.54 \pm 0.11}$	0.92 ± 0.04	$\mathbf{0.98 \pm 0.03}$	0.72 ± 0.14
	Regression	$\mathbf{0.97 \pm 0.04}$	0.52 ± 0.16	0.47 ± 0.08	$\mathbf{0.95 \pm 0.06}$	$\mathbf{0.98 \pm 0.03}$	$\mathbf{0.78 \pm 0.08}$

Table 10: Performance comparison of Flow and Regression methods using DP3 architecture across different **point-cloud-based** robotic manipulation tasks. For each task, we report the best checkpoint performance / averaged performance over last 5 checkpoints. Each experiment is run with 3 seeds and we report the average performance across all seeds.

Architecture	**Method**	**P1**	**P2**	**P3**	**P4**
Chi-UNet	**Flow**	1.0	1.0	1.0	0.98
	MIP	1.0	1.0	1.0	0.94
	Regression	0.98	0.94	0.94	0.86
Chi-Transformer	**Flow**	1.0	1.0	1.0	1.0
	MIP	1.00	0.98	0.98	0.96
	Regression	1.0	1.0	0.98	0.94
Sudeep-DiT	Flow	1.0	1.0	1.0	0.98
	MIP	1.00	1.00	1.00	0.98
	Regression	1.0	0.98	0.96	0.88

Table 11: **Performance comparison across different design choices on kitchen task.** Kitchen task has multiple stages and we report the success rate of finishing n tasks in the table. For the performance reported in the main paper and previous tables, we report the success rate of finishing 4 tasks.

Algorithm 1 Lipschitz Constant Estimation via Action Perturbation

Require: Dataset $\mathcal{D}$, policy π, noise scale ϵ, number of critical states N_s=100, number of perturbations N_p=64
Ensure: Estimated Lipschitz constant L

$S \leftarrow$ identify N_s critical states from $\mathcal{D}$ ▷ Select states with highest action ambiguity
for all critical state $s \in S$ **do**
  $(a, o) \leftarrow$ expert action and nominal next observation for s ▷ Get ground truth action-observation pair
  $(\bar{a}, \bar{o}) \leftarrow$ normalize (a, o) using dataset statistics ▷ Ensure comparability across states/tasks
  for $i = 1$ to N_p **do**
    $\eta \sim \mathcal{N}(0, \epsilon^2 I)$ ▷ Sample Gaussian perturbation
    $a_{\text{pert}} \leftarrow \text{unnormalize}(\bar{a} + \eta)$ ▷ Create perturbed action in original scale
    $o^{(i)} \leftarrow \text{rollout}(a_{\text{pert}})$ in environment ▷ Execute perturbed action to get feasible state
    $\bar{o}^{(i)} \leftarrow \text{normalize}(o^{(i)})$ ▷ Normalize perturbed observation
    $a^{(i)} \leftarrow \pi(o^{(i)})$ ▷ Get policy prediction on perturbed state
    $\bar{a}^{(i)} \leftarrow \text{normalize}(a^{(i)})$ ▷ Normalize predicted action
    $r_i \leftarrow \frac{\|\bar{a}^{(i)} - \bar{a}\|_2}{\|\eta\|_2}$ ▷ Compute finite difference approximation
  $L_s \leftarrow \max_i r_i$ ▷ Local Lipschitz constant for state s
$L \leftarrow \frac{1}{N_s} \sum_{s=1}^{N_s} L_s$ ▷ Average across all critical states
return L

Method	Reg.	SF			RR		MIP		Flow		
NFEs	1	1	3	9	1	2	1	2	1	3	9
S.R.	0.46	0.54	0.55	0.52	0.31	0.33	0.50	0.74	0.32	0.55	0.66

Table 12: **Comparison of methods and their corresponding number of function evaluations (NFEs).** Evaluated on state-based `Tool-Hang` task over `Chi-UNet`. Average success rate is reported across 3 seeds and 5 checkpoints.

E.2 Full Lipschitz Evaluation Results

In the main text, we only report the average Lipschitz constant on critical states across 3 architectures. Here, we report the full Lipschitz constant evaluation reuslt in Table 13 with different architectures and tasks.

Task	Architecture	Method	Lipschitz Constant (Policy)
`Push-T` (State)	`Chi-UNet`	**Regression**	0.85 ± 0.58
		Flow	0.31 ± 0.01
	`Sudeep-DiT`	**Regression**	0.52 ± 0.11
		Flow	0.22 ± 0.02
	`Chi-Transformer`	**Regression**	1.33 ± 1.14
		Flow	0.82 ± 0.26
`Kitchen` (State)	`Chi-UNet`	**Regression**	13.47 ± 2.80
		Flow	13.31 ± 4.13
	`Sudeep-DiT`	**Regression**	15.37 ± 3.69
		Flow	12.54 ± 5.09
	`Chi-Transformer`	**Regression**	13.37 ± 4.00
		Flow	11.44 ± 4.10
`Tool-Hang` (PH, State)	`Chi-UNet`	**Regression**	1.63 ± 0.79
		Flow	1.53 ± 1.01
	`Sudeep-DiT`	**Regression**	1.86 ± 0.81
		Flow	1.34 ± 0.97
	`Chi-Transformer`	**Regression**	1.76 ± 1.02
		Flow	1.40 ± 0.99

Table 13: **Detailed: Per-architecture policy Lipschitz.**

F Multi-Modality Study Details

F.1 Q Function Estimation

To rule out the possibility of hidden multi-modality, we also plot Q functions for each action to see if there is any clear clustering pattern of Q w.r.t. different actions in t-SNE visualization. Since we only have access to expert actions rather than their policy, we estimate the Q function by Monte Carlo sampling with the learned flow policy. The detailed procedure is as follows:

Starting from one "critical state", we first sample N actions

$$a^{(i)} = \Phi(o, z^{(i)}, s = 0, t = 1), \quad i = 1, \ldots, N, \quad z^{(i)} \sim \mathrm{N}(0, \mathbf{I}).$$

For each sampled action $a^{(i)}$, we execute one environment step to obtain the next observation $o'^{(i)}$ and immediate reward $r(o, a^{(i)})$. Then, starting from $o'^{(i)}$, we rollout the learned policy for N_{MC} episodes until termination (horizon H), and average the cumulative returns to obtain an estimate of the continuation value. Thus, the Q-value for action $a^{(i)}$ is approximated as:

$$Q_\Phi(a^{(i)}, o) = r(o, a^{(i)}) + \frac{1}{N_{\text{MC}}} \sum_{j=1}^{N_{\text{MC}}} \sum_{t=1}^{H} r\big(o_t^{(j)}, a_t^{(j)}\big). \tag{F.1}$$

We set the discount factor $\gamma = 1.0$ since rewards are sparse and triggered only at task completion. The reward for `Tool-Hang` and `Kitchen` is defined by the *final* success signal (with `Kitchen`

's success requiring all 4 subtasks to be completed). The reward for `Push-T` is defined by *final* coverage.

Algorithm 2 Q Function Estimation via Monte Carlo Sampling

Require: Dataset $\mathcal{D}$, flow policy Φ, reward function r, number of critical states N_s=100, number of action samples N, Monte Carlo samples N_{MC}
Ensure: For each state o, pairs $\{(a^{(i)}, Q_\Phi(a^{(i)}, o))\}_{i=1}^{N}$

$S \leftarrow$ identify N_s critical states from $\mathcal{D}$ ▷ Select states with highest action ambiguity
for all critical state $s \in S$ **do**
  $o \leftarrow$ observation for state s
  for $i = 1$ to N **do** ▷ Sample actions and compute Q estimates
    $z^{(i)} \sim \mathrm{N}(0, \mathbf{I})$
    $a^{(i)} \leftarrow \Phi(o, z^{(i)}, s{=}0, t{=}1)$
    Execute $(o, a^{(i)})$ in env $\rightarrow$ obtain $o'^{(i)}$, $r^{(i)} = r(o, a^{(i)})$
    for $j = 1$ to N_{MC} **do** ▷ Monte Carlo rollouts from $o'^{(i)}$
      Rollout Φ from $o'^{(i)}$ until horizon H to get cumulative return $R_j^{(i)}$
    $Q_\Phi(a^{(i)}, o) \leftarrow r^{(i)} + \frac{1}{N_{\text{MC}}} \sum_{j=1}^{N_{\text{MC}}} R_j^{(i)}$
  Store $\{(a^{(i)}, Q_\Phi(a^{(i)}, o))\}_{i=1}^{N}$ for state s

The procedure above explicitly computes Q-values by rolling out trajectories separately for each sampled action.

F.2 Deterministic Dataset Generation

To generate a deterministic dataset that completely eliminates any potential multi-modality, we follow a systematic process:

First, we train a flow expert policy Φ on the original dataset. Then, we collect a new dataset by rolling out this expert policy from different initial states (using different random seeds than those used during testing). Crucially, during rollout, we always evaluate the flow policy deterministically by setting the initial noise to zero: $z = 0$. This ensures that the policy produces deterministic actions given any observation, completely removing any stochasticity from the action generation process.

During data collection, we discard all failed trajectories to maintain the same success rate as the original dataset. We continue collecting until we reach the target number of trajectories N_{traj}.

Algorithm 3 Deterministic Dataset Generation

Require: Trained flow policy Φ, target number of trajectories N_{traj}, maximum episode steps $T_{\max}$
Ensure: Deterministic dataset $\mathcal{D}_{\text{det}}$

$\mathcal{D}_{\text{det}} \leftarrow \emptyset$
$n_{\text{collected}} \leftarrow 0$
while $n_{\text{collected}} < N_{\text{traj}}$ **do**
  Reset environment with new random seed
  $o_0 \leftarrow$ initial observation
  $\tau \leftarrow [(o_0, \cdot)]$ ▷ Initialize trajectory
  for $t = 0$ to $T_{\max} - 1$ **do**
    $a_t \leftarrow \Phi(z = 0, o_t, s = 0, t = 1)$ ▷ Deterministic action
    $o_{t+1}, r_t, \text{done} \leftarrow \text{env.step}(a_t)$
    $\tau \leftarrow \tau \cup [(o_t, a_t)]$
    if done **then**
      break
  if trajectory τ is successful **then**
    $\mathcal{D}_{\text{det}} \leftarrow \mathcal{D}_{\text{det}} \cup \{\tau\}$
    $n_{\text{collected}} \leftarrow n_{\text{collected}} + 1$
return $\mathcal{D}_{\text{det}}$

G Manifold Adherence Study Details

G.1 Validation Loss Is Not a Good Proxy for Policy Performance

To investigate whether validation loss serves as a reliable proxy for policy performance, we examine its relationship with success rates on `Tool-Hang` across different architectures given different training methods. Evidence that validation loss is poorly correlated with success rate can be seen by comparing flow policies with varying numbers of function evaluations (NFEs) and their corresponding validation losses. Table 14 demonstrates that increasing NFEs does not reduce validation loss, yet policy performance consistently improves. We hypothesize that higher NFEs introduce stronger inductive bias and regularization, which projects actions back onto the data manifold, thereby enhancing generalization.

Architecture	Method	NFEs	Average Success Rate	L_2 Validation Loss
`Chi-UNet`	**`Regression`**	1	0.54	0.063
	`Flow`	1	0.36	0.053
	`Flow`	3	0.44	0.052
	`Flow`	9	0.58	0.053
`Chi-Transformer`	**`Regression`**	1	0.18	0.084
	`Flow`	1	0.06	0.093
	`Flow`	3	0.72	0.092
	`Flow`	9	0.68	0.089
`Sudeep-DiT`	**`Regression`**	1	0.20	0.063
	`Flow`	1	0.62	0.082
	`Flow`	3	0.76	0.080
	`Flow`	9	0.76	0.080

Table 14: Comparison of validation loss and success rate across different architectures and methods on state-based `Tool-Hang`. The results show that validation loss is not a reliable proxy for policy performance.

G.2 Manifold Adherence Evaluation Method

To evaluate the manifold adherence, we compute the projection error of a predicted action a onto the space spanned by expert actions at neighboring states. Concretely, given a state, we compute its ℓ_2 distance to all states in the training set. Then, we pick k nearest neighbor states and gather their corresponding actions $A = [a^{(0)}, a^{(1)}, \ldots, a^{(k)}]$. Lastly, we compute projection error by projecting a to the column space of A: $\|a - P_A(a)\|_2 = \min_c \|a - Ac\|_2$.

H Nearest Neighbor Hypothesis Study

Another popular hypothesis is that GCPs are learning a lookup table of observation-to-action mappings (Pari et al., 2021; He et al., 2025). This might be true for relatively simple tasks that do not require high precision and complex generalization, such as `Can`. However, for tasks that require higher precision and more contact, such as `Tool-Hang`, the nearest-neighbor/lookup-table assumption is insufficient to explain the success of GCPs. We evaluate the performance of a nearest-neighbor policy (VINN (Pari et al., 2021)) on state-based `Tool-Hang` and find that it achieves a success rate of only 12% as shown in Table 15. This is significantly lower than both flow and regression methods, indicating that the action manifold is not linearly spanned by the expert actions. Nevertheless, nearest-neighbor can still serve as a proxy for the expert action manifold, as it captures the general trend of actions—even though linear combinations of actions in the dataset cannot directly produce the correct action, the expert action manifold should not be too distant. Therefore, in this paper, we use nearest-neighbor as a proxy for the linearized expert action manifold rather than directly computing the distance between expert actions in the validation set and predicted actions.

Action Chunk Size	Success Rate (%)
1	0
8	4
16	12
32	2

Table 15: Performance of k-nearest neighbor policy on state-based `Tool-Hang` task. Using the same method as VINN with softmax over k=5 nearest neighbors.

I Theoretical analysis of GCP's expressivity

I.1 Formal statement of Theorem 1

In this section, we introduce the notation and definition required for the subsequent proofs and provide the formal statement of Theorem 1 from the main text. Throughout, let $\|\cdot\|_\circ$ denote any matrix norm satisfying the property $\|X_1X_2\|_\circ \le \|X_1\|_{\mathrm{op}}\|X_2\|_\circ$. In contrast to the notation used in the main text, we define $\Phi_{s,t}(a,o)$ as the solution at time t of the ODE:

$$\frac{\mathrm{d}}{\mathrm{d}t}a_t = b_t^\star(a_t \mid o), \quad \text{with initial condition } a_s = a. \tag{I.1}$$

Note that $\Phi_{0,1}(a_0 = z, o)$ coincides with the definition of $\pi_\theta^\star(z,o)$ in the main text. Next, we define the notion of κ-*log-concavity*.

Definition I.1 (κ-log-concavity). A distribution with density $\rho = e^{-V(x)}$ is said to be κ-*log-concave* if $V \in C^2(\mathbb{R}^d)$ and its Hessian satisfies $\nabla^2 V(x) \succcurlyeq \kappa \mathbf{I}$ for all $x \in \mathbb{R}^d$ and some $\kappa > 0$.

With this notation in place, we now state the formal version of Theorem 1.

Theorem 2. *Suppose that*

$$b_t^\star = \mathbb{E}[\dot{I}_t \mid I_t, o], \qquad \textit{where } I_t = (1-t)a_0 + ta_1, \quad a_0 \sim \mathrm{N}(0, \mathbf{I}), \quad a_1 \sim \rho_1, \tag{I.2}$$

where ρ_1 is κ-log-concave. Then, we have

$$\|\nabla_o \Phi_{0,t}(a_0, o)\|_\circ \le \int_0^t \sqrt{\frac{\kappa(1-t)^2 + t^2}{\kappa(1-s)^2 + s^2}} \cdot \|\nabla_o b_s^\star(a_s \mid o)\|_\circ \mathrm{d}s. \tag{I.3}$$

In particular, for $t = 1$ we obtain

$$\|\nabla_o \Phi_{0,1}(a_0, o)\|_\circ \le \sqrt{1 + \kappa^{-1}} \int_0^1 \|\nabla_o b_s^\star(a_s \mid o)\|_\circ \mathrm{d}s. \tag{I.4}$$

Remark I.1. Theorem 1 follows immediately from the fact that both the operator and the Frobenius norms satisfy $\|X_1X_2\|_\circ \le \|X_1\|_{\mathrm{op}}\|X_2\|_\circ$ together with the inequality Eq. (I.4).

I.2 Supporting lemmas

We state the supporting lemmas for proving Theorem 2 below and provide their proofs immediately for completeness. As a first step, we analyze the dynamical system satisfied by $\nabla_o \Phi_{s,t}(a,o)$.

Lemma I.1. *Define $a_t := \Phi_{0,t}(a_0, o)$ where a_0 is the initial condition, and define the matrices*

$$M_t := \nabla_o \Phi_{0,t}(a_0, o), \quad A_t := (\nabla_a b_t^\star)(a_t \mid o), \quad E_t := (\nabla_o b_t^\star)(a_t \mid o) \tag{I.5}$$

Then,

$$\frac{\mathrm{d}}{\mathrm{d}t}M_t = A_t M_t + E_t, \quad M_0 = 0 \tag{I.6}$$

Proof. Since $\Phi_{0,0}(a_0, o) = a_0$, $M_0 = 0$. Moreover,

$$\frac{\mathrm{d}}{\mathrm{d}t}\nabla_o \Phi_{0,t}(a_0, o) = \nabla_o \frac{\mathrm{d}}{\mathrm{d}t}\Phi_{0,t}(a_0, o) = \nabla_o (b_t^\star(\Phi_{0,t}(a_0, o) \mid o)) \tag{I.7}$$

$$= (\nabla_a b_t^\star)(a_t \mid o) \cdot \nabla_o \Phi_{0,t}(a_0, o) + (\nabla_o b_t^\star)(a_t \mid o) \tag{I.8}$$

□

Note that, from the previous lemma, we may introduce $\Lambda_{s,t}$ as the solution to the matrix ODE

$$\frac{\mathrm{d}}{\mathrm{d}t}\Lambda_{s,t} = A_t\Lambda_{s,t}, \quad \Lambda_{s,s} = \mathbf{I}. \tag{I.9}$$

Moreover, it follows that

$$\Lambda_{s,t} = \nabla_a \Phi_{s,t}(a_s, o). \tag{I.10}$$

We are now ready to state the relation between M_t and $\Lambda_{s,t}$.

Lemma I.2.

$$M_t = \int_0^t \Lambda_{s,t} E_s \mathrm{d}s. \tag{I.11}$$

Proof. Using $\frac{\mathrm{d}}{\mathrm{d}t}\Lambda_{0,t}^{-1} = -\Lambda_{0,t}^{-1}A_t$ and we consider the time derivative of $\Lambda_{0,t}^{-1}M_t$:

$$\frac{\mathrm{d}}{\mathrm{d}t}(\Lambda_{0,t}^{-1}M_t) = (\frac{\mathrm{d}}{\mathrm{d}t}\Lambda_{0,t}^{-1})M_t + \Lambda_{0,t}^{-1}(\frac{\mathrm{d}}{\mathrm{d}t}M_t) \tag{I.12}$$

$$= -\Lambda_{0,t}^{-1}A_tM_t + \Lambda_{0,t}^{-1}A_tM_t + \Lambda_{0,t}^{-1}E_t \tag{I.13}$$

$$= \Lambda_{0,t}^{-1}E_t. \tag{I.14}$$

Note that $\Lambda_{0,t}$ is invertible by uniqueness of the ODE solution in Eq. (I.9). Integrating both sides with respect to t gives

$$\Lambda_{0,t}^{-1}M_t = \int_0^t \Lambda_{0,s}^{-1}E_s \mathrm{d}s. \tag{I.15}$$

Hence, we have

$$M_t = \Lambda_{0,t}\int_0^t \Lambda_{0,s}^{-1}E_s \mathrm{d}s. \tag{I.16}$$

Note that $\Lambda_{0,s}^{-1} = \Lambda_{s,0}$ and $\Lambda_{0,t}\cdot\Lambda_{s,0} = \Lambda_{s,t}$, we obtain

$$M_t = \int_0^t \Lambda_{s,t}E_s \mathrm{d}s. \tag{I.17}$$

□

An immediate application of the triangle inequality and the property of $\|\cdot\|_\circ$ yields

$$\|M_t\|_\circ \le \int_0^t \|\Lambda_{s,t}\|_{\mathrm{op}}\|E_s\|_\circ \mathrm{d}s. \tag{I.18}$$

Moreover, $\|\Lambda_{s,t}\|_{\mathrm{op}}$ admits the bound:

Lemma I.3.

$$\|\Lambda_{s,t}\|_{\mathrm{op}} \le \exp\left(\int_s^t \|A_{s'}\|_{\mathrm{op}}\mathrm{d}s'\right). \tag{I.19}$$

Proof. Define $f_\omega(s,t) = \Lambda_{s,t}\omega$. We have

$$\frac{\mathrm{d}}{\mathrm{d}t}\|f_\omega(s,t)\|_2 = \frac{1}{\|f_\omega(s,t)\|_2}f_\omega(s,t)^\top\frac{\mathrm{d}}{\mathrm{d}t}f_\omega(s,t) \tag{I.20}$$

$$= \frac{1}{\|f_\omega(s,t)\|_2}\omega^\top\Lambda_{s,t}^\top A_t\Lambda_{s,t}\omega \tag{I.21}$$

$$\le \|A_t\|_{\mathrm{op}}\|f_\omega(s,t)\|_2. \tag{I.22}$$

By Gronwall's theorem and $\|f_\omega(s,s)\|_2 = \|\omega\|_2$, we obtain

$$\|f_\omega(s,t)\|_2 \le \|\omega\|_2\exp(\int_s^t \|A_{s'}\|_{\mathrm{op}}\mathrm{d}s'). \tag{I.23}$$

□

To bound $\exp\left(\int_s^t \|A_{s'}\|_{\mathrm{op}}\mathrm{d}s'\right)$, we use the following result from (Daniels, 2025), included here for completeness.

Theorem 3 (Restated; Theorem 6 in (Daniels, 2025)). *Suppose $\mu_0 \sim \mathrm{N}(0, \mathbf{I})$ and μ_1 is a κ-log-concave distribution with $\kappa > 0$. Define*

$$I_t = \alpha_t X_0 + \beta_t X_1, \qquad X_0 \sim \mu_0, \quad X_1 \sim \mu_1, \tag{I.24}$$

and let $v_t(x)$ denote the corresponding flow field. Then,

$$\nabla_x v_t(x) \preccurlyeq \frac{\kappa\alpha_t\dot{\alpha}_t + \beta_t\dot{\beta}_t}{\kappa\alpha_t^2 + \beta_t^2}\mathbf{I}. \tag{I.25}$$

With the result, we can bound $\exp\left(\int_s^t \|A_{s'}\|_{\mathrm{op}}\mathrm{d}s'\right)$ as follows.

Lemma I.4.

$$b_t^\star = \mathbb{E}[\dot{I}_t \mid I_t, o], \qquad \textit{where } I_t = (1-t)a_0 + ta_1, \quad a_0 \sim \mathrm{N}(0, \mathbf{I}), \quad a_1 \sim \rho_1, \tag{I.26}$$

where ρ_1 is κ-log-concave. Then, we have

$$\int_s^t \|\nabla_x b_{s'}^\star(a_{s'} \mid o)\|_{\mathrm{op}}\mathrm{d}s' \le \log\sqrt{\frac{\kappa(1-t)^2 + t^2}{\kappa(1-s)^2 + s^2}} \tag{I.27}$$

Proof. By leveraging Theorem 3 for each condition o, we have

$$\nabla_a b_{s'}^\star(a_{s'} \mid o) \preccurlyeq \frac{\kappa\alpha_{s'}\dot{\alpha_{s'}} + \beta_{s'}\dot{\beta_{s'}}}{\kappa\alpha_{s'}^2 + \beta_{s'}^2}\mathbf{I}, \tag{I.28}$$

then we have

$$\|\nabla_a b_{s'}^\star(a_{s'} \mid o)\|_{\mathrm{op}} \le \frac{\kappa\alpha_{s'}\dot{\alpha_{s'}} + \beta_{s'}\dot{\beta_{s'}}}{\kappa\alpha_{s'}^2 + \beta_{s'}^2}. \tag{I.29}$$

Integrating both sides, we obtain

$$\int_s^t \|\nabla_a b_{s'}^\star(a_{s'} \mid o)\|_{\mathrm{op}}\mathrm{d}s' \le \int_s^t \frac{\kappa\alpha_{s'}\dot{\alpha_{s'}} + \beta_{s'}\dot{\beta'_s}}{\kappa\alpha_{s'}^2 + \beta_{s'}^2}\mathrm{d}s' \tag{I.30}$$

$$= \frac{1}{2}\log(\kappa\alpha_{s'}^2 + \beta_{s'}^2)\Big|_s^t \tag{I.31}$$

$$= \log\sqrt{\frac{\kappa\alpha_t^2 + \beta_t^2}{\kappa\alpha_s^2 + \beta_s^2}}. \tag{I.32}$$

By substitute $\alpha_t = 1 - t$ and $\beta_t = t$, we have

$$\int_s^t \|\nabla_a b_{s'}^\star(a_{s'} \mid o)\|_{\mathrm{op}}\mathrm{d}s' \le \log\sqrt{\frac{\kappa(1-t)^2 + t^2}{\kappa(1-s)^2 + s^2}}. \tag{I.33}$$

□

With the preceding components in place, we now establish Theorem 2.

I.3 Proof of Theorem 2

By combining Eq. (I.18), Lemma I.3, and Lemma I.4, we have

$$\|\nabla_o \Phi_{0,t}(a_0, o)\|_\circ \le \int_0^t \sqrt{\frac{\kappa(1-t)^2 + t^2}{\kappa(1-s)^2 + s^2}} \cdot \|\nabla_o b_s^\star(a_s \mid o)\|_\circ \mathrm{d}s. \tag{I.34}$$

For $t = 1$, the function $s \mapsto \kappa(1-s)^2 + s^2$ attains its minimum at $s = \frac{\kappa}{\kappa+1}$. Applying Holder's inequality then yields

$$\|\nabla_o \Phi_{0,1}(a_0, o)\|_\circ \leq \int_0^1 \sqrt{\frac{1}{\kappa(1-s)^2 + s^2}} \cdot \|\nabla_o b_s^\star(a_s \mid o)\|_\circ \mathrm{d}s \tag{I.35}$$

$$\leq \max_{s\in[0,1]} \left(\sqrt{\frac{1}{\kappa(1-s)^2 + s^2}} \right) \cdot \int_0^1 \|\nabla_o b_s^\star(a_s \mid o)\|_\circ \mathrm{d}s \tag{I.36}$$

$$= \sqrt{1+\kappa^{-1}} \int_0^1 \|\nabla_o b_s^\star(a_s \mid o)\|_\circ \mathrm{d}s. \tag{I.37}$$

J Toy experiments: Testing the function approximation capabilities of regression and flow models

J.1 Experimental Setup: Regression Functions

In this section, we introduce a number of functions that determine the ability of our model to fit high-frequency data. In accordance with the standard regression nomenclature, we use x as input space and y as the output.

J.1.1 Mixture of Sines

Given weight, frequency and phase shift vectors $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m), \boldsymbol{\omega} = (\omega_1, \ldots, \omega_m), \boldsymbol{\phi} = (\phi_1, \ldots, \phi_m) \in \mathbb{R}^m$, we define the parameter $\boldsymbol{\psi} = (\boldsymbol{\alpha}, \omega, \boldsymbol{\phi})$.

$$g_\psi(x) := \sum_i \alpha_i \sin(\omega_i x + \phi_i) : \mathbb{R} \to \mathbb{R}. \tag{J.1}$$

We consider two regimes: In *fixed-weight regime*, we set $\alpha_i \equiv 1$ for all i. In the *fixed-Lipschitz* regime, we take $\alpha_i \equiv \frac{1}{\omega_i}$. The latter choice ensures that the Lipschitz constant of $g_\psi(x)$ is bounded by m, the number of distinct components. This allows us to test the fitting of high-frequency functions without an increase in Lipschitz constant, and captures the natural regime where we expect that high-frequency components contribute less to the overall functions.

For the `mixture-of-sines` experiments, we include both the fixed-Lipschitz and fixed-weight regimes. We $m = 10$ and sample $\boldsymbol{\omega}, \boldsymbol{\phi}$ from the set of primes. When sampling this way, we ensure that frequencies and phase shifts remain uncorrelated across the high-frequency components.

J.1.2 Projected Mixture of Sines

In our next experiment, we consider maps from $\mathbb{R} \to \mathbb{R}^d$, with $d = 8$ with a projection operator applied. Formally, for each $j = 1, \ldots, 10$, we let

$$P_j \in \mathbb{R}^{d\times d}, \quad \operatorname{rank}(P_j) = 3 \tag{J.2}$$

by a uniformly-at-random chosen rank 3 projection matrix. We then define our target function as

$$f(x) = P_{j(x)} \vec{g}(x) : \mathbb{R}^d \to \mathbb{R}. \tag{J.3}$$

where $\vec{g}(x) = (g_{\psi_1(x)}, \ldots, g_{\psi_8(x)})$ is a vector whose entries are functions of the form of that in Eq. (J.1), and where $j(x)$ is the index such that

$$x \in \left[\frac{j(x)-1}{10}, \frac{j(x)}{10} \right]. \tag{J.4}$$

These experiments fit a function along a x-dependent projection (see below for our evaluations that account for the projection discontinuity in a fair manner).

J.1.3 Projected Mixture of Sines under Nonlinear Transformation

Finally, we take the function $\vec{g}(x) = (g_{\psi_1}(x), \ldots, g_{\psi_d}(x))$, where now we take $d = 32$. Again, ψ_i are generated as Appendix J.1.1. Now we consider the function

$$h(x) = \phi^{-1}\left(P_{j(x)}\phi(\vec{g}(x))\right), \tag{J.5}$$

where, as above, $P_{j(x)}$ is an x-dependent projection, but we include composition with the function $\phi(u) = u^3$, broadcast entrywise.

J.2 Experimental Setup: Network architecture

We use MLP architecture with and without FiLM conditioning. We restrict ourselves to two- and three-layer MLPs with widths of 32 or 64, so that we are squarely in the low network expressivity regime.

J.3 Experimental Setup: Data Generation and Evaluation Metrics

Data generation. We intentionally constrain the dataset to be a low data regime. We typically choose no more data points than what is required for the reconstruction of these high frequency functions based on Nyquist sampling theorem. In our experiments this number is typically around 120, with $c \sim \mathrm{Unif}[0, 1]$. We train for 50000 epochs with batch size 32 and learning rate 0.001

L_2 Error. We evaluate the function approximation abilities of both regression and flow using the L_2 error on the test set. We use flow with NFEs $= 1$.

Projection Metric. To demonstrate this, we define a new projection metric. Let $c_j \in \{0.1, 0.2, \ldots, 0.9\}$ define the interval boundary points and let $\bar{I}_j = [c_j - 0.03, c_j + 0.03]$ be small neighborhoods around each boundary. We define P_j as the projection operator onto the subspace spanned by function evaluations in interval $[c_{j-1}, c_j]$, and P_{j+1} as the projection onto interval $[c_j, c_{j+1}]$. The combined projection $P_{j,j+1}$ projects onto the joint span of both adjacent subspaces. This is computed by performing SVD on the concatenated basis functions from both and retaining the top $2k$ singular vectors while accounting for numerical stability. Finally, we evaluate the normalized projection error onto the orthogonal complement of $P_{j,j+1}$:

$$\frac{\|(\mathbf{I} - \mathbf{P}_{j,j+1})(\hat{\mathbf{f}} - \mathbf{f}_{\text{true}})\|}{\|\hat{\mathbf{f}} - \mathbf{f}_{\text{true}}\|} \tag{J.6}$$

This metric quantifies how much of the prediction error lies outside the subspace spanned by the two adjacent intervals, providing insight into the model's ability to capture local structure at interval boundaries.

J.4 Findings

- When measured in the L_2 reconstruction error, we find that regression consistently outperforms flow.
- When measured in projection metric, in Projected Mixture of Sines and Projected Mixture of Sines under Nonlinear Transformation, we find that flow consistently outperforms regression.

Experiment	L_2 Error	Projection Metric
Mixture of Sines	**R**	NA
Projected Mixture of Sines	**R**	**F**
Projected Mixture of Sines under Nonlinear Transformation	**R**	**F**

Table 16: Summary of experimental findings. **R** = Regression outperforms, **F** = Flow outperforms.

J.5 Analysis of Findings: Manifold Adherence distinguishes **Flow** over Regression

- In Section 5 we conjecture that one of the factors that could explain superior performance of flow, is its superiority in approximating the expert more accurately in some measure of reconstruction error. However, the performance on L_2 reconstruction error for these to Mixture of Sines functions shows that flow is not any better at approximating functions as compared to regression.

- In Appendix G we study the projection error of error of a predicted action a onto the space spanned by expert actions at neighboring states. This was done to assess how well our policies can generate plausible actions under simulated compounding error.

- We designed our projection of mixture of sines experiments along the same lines, to further verify manifold adherence. Flow's consistently better performance on projection metric *especially under nonlinear transformation* as compared to regression further lends credence to the idea that manifold adherence is reason for flow's superior performance.

Mixture of Sines, $d = 1, m = 10$

Architecture	Regression	Flow
MLP 2 layer 32 wide	2.20	2.19
MLP 2 layer 64 wide	2.07	2.13
MLP 3 layer 32 wide	2.16	2.16
MLP 3 layer 64 wide	1.76	1.84

Table 17: No FiLM conditioning

Architecture	Regression	Flow
MLP 2 layer 32 wide	1.99	1.99
MLP 2 layer 64 wide	1.69	1.98
MLP 3 layer 32 wide	1.62	1.53
MLP 3 layer 64 wide	1.25	1.54

Table 18: With FiLM conditioning

Table 19: L_2 reconstruction error

Projected Mixture of Sines, $d = 32, k = 3, m = 10$

Architecture	Regression	Flow
MLP 2 layer 32 wide	0.59	0.60
MLP 2 layer 64 wide	0.55	0.58
MLP 3 layer 32 wide	0.55	0.59
MLP 3 layer 64 wide	0.50	0.56

Table 20: No FiLM conditioning

Architecture	Regression	Flow
MLP 2 layer 32 wide	0.56	0.59
MLP 2 layer 64 wide	0.55	0.52
MLP 3 layer 32 wide	0.50	0.57
MLP 3 layer 64 wide	0.44	0.55

Table 21: With FiLM conditioning

Table 22: L_2 reconstruction error

Architecture	Regression	Flow
MLP 2 layer 32 wide	0.909	0.904
MLP 2 layer 64 wide	0.911	0.908
MLP 3 layer 32 wide	0.910	0.903
MLP 3 layer 64 wide	0.914	0.904

Table 23: No FiLM conditioning

Architecture	Regression	Flow
MLP 2 layer 32 wide	0.910	0.903
MLP 2 layer 64 wide	0.915	0.904
MLP 3 layer 32 wide	0.914	0.902
MLP 3 layer 64 wide	0.918	0.913

Table 24: With FiLM conditioning

Table 25: Mean Projection Error - $\frac{\|(\mathbf{I}-\mathbf{P})(\hat{\mathbf{f}}-\mathbf{f}_{\text{true}})\|}{\|\hat{\mathbf{f}}-\mathbf{f}_{\text{true}}\|}$

Projected Mixture of Sines under Nonlinear Transformation - $d = 32, k = 3, m = 10, \phi(u) = u^3$

Architecture	Regression	Flow
MLP 2 layer 32 wide	1.67	1.76
MLP 2 layer 64 wide	1.63	1.70
MLP 3 layer 32 wide	1.58	1.72
MLP 3 layer 64 wide	1.45	1.67

Table 26: No FiLM conditioning

Architecture	Regression	Flow
MLP 2 layer 32 wide	1.60	1.75
MLP 2 layer 64 wide	1.39	1.65
MLP 3 layer 32 wide	1.54	1.70
MLP 3 layer 64 wide	1.17	1.55

Table 27: With FiLM conditioning

Table 28: L_2 reconstruction error

Architecture	Regression	Flow
MLP 2 layer 32 wide	0.90	0.87
MLP 2 layer 64 wide	0.89	0.85
MLP 3 layer 32 wide	0.88	0.82
MLP 3 layer 64 wide	0.87	0.83

Table 29: No FiLM conditioning

Architecture	Regression	Flow
MLP 2 layer 32 wide	0.89	0.84
MLP 2 layer 64 wide	0.86	0.82
MLP 3 layer 32 wide	0.88	0.81
MLP 3 layer 64 wide	0.86	0.79

Table 30: With FiLM conditioning

Table 31: Mean Projection Error - $\frac{\|(\mathbf{I}-\mathbf{P})(\hat{\mathbf{f}}-\mathbf{f}_{\text{true}})\|}{\|\hat{\mathbf{f}}-\mathbf{f}_{\text{true}}\|}$

K Appendix for Section 2

K.1 Markov Decision Processes Configuration

We consider a Markov Decision Process $\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, P, P_0)$[2] with the state space $\mathcal{S}$, the action space $\mathcal{A}$, the reward $R(s, a)$[3] obtained by taking action a in state s, the transition dynamics $P : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, and the initial-state distribution $P_0 \in \Delta(\mathcal{S})$. To formulate the success rate (i.e., performance) in this setting, we define the reward function as:

$$R(s, a) = \begin{cases} 1, & \text{if the task is successful under } (s, a), \\ 0, & \text{otherwise.} \end{cases} \tag{K.1}$$

Under this definition of rewards, the expected return of a policy π is $J(\pi) = \mathbb{E}[\sum_t R(s_t, a_t)]$, which reduces to $\mathbb{P}[\text{success under } \pi]$. Hence, $J(\pi)$ exactly equals the success rate of policy π.

K.2 Integrated Flow Prediction

For completeness, we provide the flow ODE as

$$\tfrac{\mathrm{d}}{\mathrm{d}t} a_t = b_t(a_t \mid o) \qquad \text{starting from} \qquad a_0 = z. \tag{K.2}$$

The associated integrated flow prediction is given by

$$\Phi_\theta(z \mid o) = z + \int_0^1 b_t(a_t \mid o)\mathrm{d}t. \tag{K.3}$$

In practice, to approximate the ODE solution for sampling, we employ the following discretized Euler integration.

Definition K.1 (Discretized Euler Integration). We discretize the time interval $[0, 1]$ to N steps with step size $h = 1/N$. The iterates are then updated according to

$$a_{k+1} = a_k + h\, b_{hk}(a_k \mid o), \quad k = 0, 1, \ldots, N-1. \tag{K.4}$$

[2]For simplicity, we consider the MDP case in this context by identifying the state with the observation defined in 2. More generally, one may consider a Partially Observable Markov Decision Process (POMDP), where the agent receives observation o emitted by an underlying latent state s.

[3]For ease of exposition, we use the same notation for rewards defined on random variables and their distributions.

The final iterate a_N serves as the Euler approximation $\Phi_{\theta,\mathrm{eul}}(z \mid o)$. We also refer to N as the *Number of Function Evaluations (NFEs)*.

L LLM USAGE

We used LLMs only for minor language polishing (grammar and wording) and to assist with literature search. All technical content, experiments, and conclusions were created and verified by the authors.