

Beyond Vision-Language-Action Models: Adapting, Steering, and Accelerating Generalist Robot Policies

Sungjae Park

CMU-RI-TR-26-73

July 2026

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania

Thesis Committee

Shubham Tulsiani, *Chair*

Katerina Fragkiadaki

Andrew Wang

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Robotics.

To everyone who filled this journey with love and joy.

Abstract

Generalist robot policies, vision-language-action models that pair a large pretrained vision-language backbone with a diffusion or flow-matching action head, are increasingly capable, yet hard to deploy in the real world. Three gaps separate such a policy from a deployable one: a *data* gap (adapting to a new task still demands task-specific teleoperation data), an *inference* gap (the policy samples its action distribution with no control over how conservative or diverse the result is), and an *architecture* gap (a large policy is too slow to replan often, so it must run its action chunks open-loop and cannot react mid-motion). This thesis argues that these gaps can be closed not by training larger models on more robot data, but by changing how a pretrained policy generates its actions, with little or no additional training.

We develop three methods. **DemoDiffusion** (data) imitates a single human demonstration instead of collecting teleoperation data: it retargets the human hand motion into a coarse robot trajectory, then uses a *frozen* generalist diffusion policy to refine it into plausible robot actions. It needs no task-specific or paired human-robot data, and succeeds even where the base policy fails outright. **Temporal Score Rescaling** (inference) rescales the learned score/flow at inference to draw from a sharper or broader distribution than the model was trained on; it is training-free, works with any diffusion or flow model, and improves image generation, depth, pose, and protein design alongside robot policies. $\pi\mathbf{R}^2$ (architecture, inference) builds on diffusion forcing to split conditioning into a fast proprioceptive channel and a slow, asynchronously updated vision-language channel, so the policy reacts to fresh proprioception while tolerating stale vision. A latency-adaptive schedule lets a single model handle varying inference latency and emit actions in a single denoising step, making a large policy reactive and real-time, several times faster than the original.

Together, these methods adapt, steer, and accelerate a pretrained policy, taking vision-language-action models beyond what they can do as trained and toward real-world deployment.

Keywords: vision-language-action models, generalist robot policies, diffusion policy,

flow matching, imitation learning, real-time control, temperature sampling

Acknowledgments

I am deeply grateful to my advisor, Shubham Tulsiani, for his guidance, encouragement, and insight throughout this work. I often explored many different directions, and he was always patient in providing thoughtful and technically grounded feedback throughout my studies. Beyond research, I also learned many valuable things from him, such as how to make effective slides and how to communicate technical ideas clearly through writing. I also thank the members of my thesis committee, Katerina Fragkiadaki and Andrew Wang, for their time and valuable feedback.

The research presented in this thesis was a collaborative effort, and I am grateful to my co-authors on the DemoDiffusion, $\pi\mathbf{R}^2$, and Temporal Score Rescaling projects for their contributions, discussions, and help with the real-robot experiments. I am especially thankful to Homanga Bharadhwaj, Yanbo Xu, Lucas Wu, and Yuxuan Kuang for creating a collaborative research environment.

Throughout my studies, I was fortunate to make many good friends across Smith Hall, Newell-Simon Hall, and the Gates and Hillman Centers. At Smith Hall, I enjoyed the many meals and foosball matches with Qitao Zhao, Yanbo Xu, Zihan Wang, Lucas Wu, and Yuxuan Kuang, as well as the everyday interactions with my labmates Yehonathan, Himangi, Soumoujit, Clara, Hanzhe, and Minsik. At Newell-Simon Hall, I would like to thank Kallol Saha, Yishu Li, and Kyutae Sim for organizing many fun activities outside of research, such as FC26, ping-pong, pool, and a trip. I also want to thank Sriram Krishna for the many enjoyable late-night conversations. At the Gates and Hillman Centers, I thank Andrew Wang, Tony Tao, Hengkai Pan, Jason Liu, and Jiahui Yang for making my summer memorable and for bringing excitement throughout my research journey.

Finally, I am deeply thankful to my family for their unconditional love, support, and encouragement. I could not have made it through this challenging period without them, and this work would not have been possible without their support.

Contents

Abstract	ii
Acknowledgments	iv
1 Introduction	1
1.1 Thesis Statement	2
1.2 Contributions	2
1.3 Thesis Organization	3
2 Background	4
2.1 Imitation Learning and Generalist Manipulation Policies	4
2.2 Diffusion Models and Flow Matching	5
2.3 Action Chunking and Real-Time Execution	6
3 Related Work	7
3.1 Generalist Manipulation Policies	7
3.2 Learning from Human and Non-Robot Data	8
3.3 Real-Time and Reactive Policy Execution	9
3.4 Controllable Generation in Diffusion and Flow Models	10
4 One-Shot Imitation from Human Demonstrations	12
4.1 Method	14
4.1.1 Overview	14
4.1.2 Kinematic Retargeting of Human Hand Trajectories to Robot End-Effector Poses	15
4.1.3 Closed-Loop Denoising of Robot Actions with a Pre-Trained Diffusion Policy	16
4.2 Experiments	17
4.2.1 Dexterous Grasping in Simulation	18

4.2.2	Real-World Manipulation	19
4.3	Limitations	23
4.4	Summary	24
5	Steering Sampling Diversity	25
5.1	Formulation	27
5.1.1	Preliminaries	27
5.1.2	Temporal Score Rescaling	28
5.1.3	Steering Inference in Diffusion and Flow Matching	29
5.2	Analysis	31
5.2.1	Validation on Toy Distributions	31
5.2.2	Interpreting Rescaling Hyperparameters	32
5.3	Applications	32
5.3.1	Text-to-Image Generation	33
5.3.2	Protein Generation	35
5.3.3	Depth Estimation	35
5.3.4	Pose Prediction	36
5.3.5	Robotic Manipulation	37
5.3.6	Parameter Selection Guidelines	39
5.4	Limitations	39
5.5	Summary	40
6	Reactive Real-Time Flow Policies	41
6.1	Method	43
6.1.1	Preliminaries	44
6.1.2	Proprioception-Reactive Diffusion Forcing	45
6.1.3	Latency-Adaptive Flow Schedule	46
6.2	Experiments	48
6.2.1	Simulation Experiments	48
6.2.2	Real-World Experiments	50
6.3	Limitations	52
6.4	Summary	52
7	Discussion	53
7.1	Adapting, Steering, and Accelerating: A Unified View	53
7.2	Limitations	54
7.3	Future Work	54

8 Conclusion	55
A DemoDiffusion: Additional Details	57
A.1 Real-World Manipulation	57
A.1.1 Hardware Setup	57
A.1.2 Task Setup Description	57
A.1.3 Text Prompts	59
A.1.4 Inference Details	59
A.1.5 Kinematic Retargeting Implementation Details	59
A.1.6 Hyperparameters	60
A.1.7 Additional Results	60
A.2 Dexterous Grasping in Simulation	62
A.2.1 Task Setup Description	62
A.2.2 Kinematic Retargeting Implementation Details	63
A.2.3 Robot Policy Implementation Details	63
A.2.4 Hyperparameters	64
B Temporal Score Rescaling: Additional Details	66
B.1 Additional Results	66
B.1.1 Image Generation	66
B.1.2 Pose Prediction	67
B.1.3 Depth Estimation	69
B.1.4 Quantifying Mode Collapse	71
B.1.5 Empirical Analysis on Score Approximation	71
B.1.6 Additional Comparison on 2D Toy Distribution	73
B.2 Proof for TSR for Mixture of Well-Separated Gaussians	74
B.2.1 Error in TSR Score Approximation	76
B.2.2 Upper Bound of the Error	76
B.2.3 Proof of Theorem	77
B.2.4 Proof of Lemma	79
B.3 Constant Noise Scaling	82
C $\pi\mathbf{R}^2$: Additional Details	85
C.1 Implementation Details	85
C.1.1 Simulation	85
C.1.2 Real-World	88
C.2 Training and Inference Algorithms	91

List of Figures

4.1	Overview of <i>DemoDiffusion</i>	13
4.2	Retargeted trajectory to closed-loop robot actions	15
4.3	Dexterous grasping results in simulation	18
4.4	Real-world manipulation tasks	20
4.5	Workspace with five cameras	21
4.6	Qualitative comparisons for real-robot manipulation	22
5.1	Temporal Score Rescaling overview	26
5.2	Qualitative examples for varying k	30
5.3	Conditional generation on a mixture of 1D Gaussians	32
5.4	Toy 2D distributions and rescaling factor	33
5.5	Image generation FID vs. CLIP	34
5.6	Protein generation	36
5.7	Qualitative depth estimation comparisons	38
5.8	Predicted poses on SYMSOL	39
6.1	Overview of $\pi\mathbf{R}^2$	42
6.2	$\pi\mathbf{R}^2$ inference cycle	46
6.3	Simulation results	48
6.4	Real-world tasks and qualitative results	50
A.1	Real-world manipulation tasks at test time	58
A.2	Zero-shot generalization to new objects	62
B.1	Ablations over (k, σ) on SD3	67
B.2	Ablations over CFG scale and samplers	67
B.3	Additional qualitative examples on SD3	68
B.4	Ablations over (k, σ) on pose prediction	69
B.5	More predicted poses on SYMSOL	70

B.6	Effects of (k, σ) on depth estimation	71
B.7	More depth prediction comparison	72
B.8	MNIST samples: DDPM, CNS, and TSR	73
B.9	MNIST class distribution	73
B.10	Empirical score approximation error	74
B.11	Comparison with CSS and MCMC baselines	75
C.1	Leap Cube Reorientation goals	86
C.2	Real-world workspace	88

List of Tables

4.1	Simulated dexterous grasping results	19
4.2	Quantitative real-world manipulation results	20
4.3	Robustness of <i>DemoDiffusion</i>	23
5.1	Text-to-image generation across models	35
5.2	Quantitative evaluation of depth estimation	37
5.3	Pose prediction	37
5.4	Results for robotic manipulation	38
6.1	Real-world results	50
A.1	Text prompts for real-world tasks	59
A.2	Hyperparameters in real-world experiments	60
A.3	Additional quantitative real-world results	61
A.4	Hyperparameters in simulation	65
B.1	Results on Stable Diffusion 2	69
C.1	Simulation hyperparameters	88
C.2	Low-level driver settings	89
C.3	Real-world task prompts	90
C.4	Real-world hyperparameters	92

Chapter 1

Introduction

A central goal of robot learning is a single *generalist* policy that can be told, in natural language, to perform a wide range of manipulation tasks and simply do them. The last few years have brought this goal remarkably close. Vision-language-action (VLA) models, generalist policies built by attaching an action head to a large pretrained vision-language backbone and training on hundreds of thousands of teleoperated demonstrations, now follow language instructions, generalize across scenes, and produce fluent multi-step behavior. These models are typically *generative*: they represent the distribution over actions with a diffusion or flow-matching head and sample action chunks from it, which lets a single policy capture the many valid ways of completing a task.

Yet a capable pretrained VLA is not the same as a *deployable* one. When one tries to put such a policy to work on a real robot and a new task, three practical gaps appear again and again. First, a **data** gap: adapting the policy to a new task or embodiment still tends to require task-specific teleoperation, which is slow to collect and does not exist for the long tail of things we might want a robot to do. Second, an **inference** gap: even a well-trained generative policy samples from its learned distribution as-is, with no principled control over whether it draws the single most likely action or explores the full breadth of behaviors, even though different situations call for different amounts of diversity. Third, an **architecture** gap: the very size that makes these policies capable also makes them too slow to replan often, so they must execute their action chunks open-loop and cannot react to what they see mid-motion, which is disqualifying for dynamic, contact-rich, closed-loop control.

This thesis argues that these gaps are not reasons to abandon generalist generative policies but places to improve them, and that all three can be addressed by changing how these policies generate their actions, rather than by training larger models with

more robot data. It develops three methods that together span the three gaps, data, inference, and architecture, each requiring little or no additional training of the underlying policy.

1.1 Thesis Statement

*Pretrained generalist generative policies are powerful but hard to deploy: they demand task-specific data, sample without control over their own distribution, and run too slowly to react. Rather than train ever-larger models, we can close these data, inference, and architecture gaps by changing how a pretrained policy generates its actions, with little or no additional training. Concretely, we **adapt** it to a new task from a single human demonstration, **steer** its sampling distribution, and **accelerate** it into a reactive, real-time controller.*

The three methods share one idea: a modern generalist policy is a score/flow-based generative model, and the sequence of denoising steps that turns noise into an action is a rich surface for adding capability with little or no additional training. Each method changes a different part of that process to close a different gap.

1.2 Contributions

The contributions of this thesis are as follows:

- **One-shot imitation from a single human demonstration (*DemoDiffusion*, Chapter 4).**¹ Closing the *data* gap, we show that a frozen, pretrained generalist diffusion policy can be turned into a one-shot imitator by treating a kinematically retargeted human demonstration as a partially noised trajectory and denoising it back onto the manifold of plausible robot actions, in the spirit of SDEdit. The method needs no task-specific training and no paired human-robot data, and it succeeds even on tasks where the underlying generalist policy fails outright.
- **Training-free control of sampling diversity (TSR, Chapter 5).**² Closing the *inference* gap, we introduce a mechanism to steer the sampling sharpness

¹DemoDiffusion was accepted to ICRA 2026 [85].

²Temporal Score Rescaling was accepted to ICML 2026 [116].

of any pretrained diffusion or flow model by rescaling its learned score/flow, controlling a local sampling temperature without fine-tuning, without extra model forward passes, and compatibly with both deterministic and stochastic samplers. We characterize its behavior analytically for simple distributions and validate it across five disparate domains, from robot manipulation to image generation and protein design.

- **Reactive real-time flow policies ($\pi\mathbf{R}^2$, Chapter 6).**³ Closing the *architecture* gap, and reaching into inference as well, we make large action-chunking flow policies reactive and real-time through two ideas built on diffusion forcing: a split of conditioning into a fast proprioceptive channel and an asynchronously updated slow vision-language channel, and a latency-adaptive schedule that emits actions in a single denoising step per call. Finetuned from a pretrained policy on a real arm-hand platform, it closes the loop several times faster than the original.
- **A unifying view of generalist-policy deployment.** We frame these methods as complementary ways of changing how a pretrained policy generates its actions, and discuss (Chapter 7) what they reveal about adapting, steering, and accelerating such policies with little or no additional training.

1.3 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 reviews the technical background: imitation learning and generalist manipulation policies, diffusion models and flow matching, action-chunking and real-time execution. Chapter 3 reviews the related work in four areas: generalist manipulation policies, learning from human and non-robot data, real-time and reactive execution, and controllable generation in diffusion and flow models. Chapter 4 presents *DemoDiffusion*, one-shot imitation from a human demonstration. Chapter 5 presents TSR, temporal score rescaling for controllable sampling. Chapter 6 presents $\pi\mathbf{R}^2$, reactive real-time flow policies. Chapter 7 draws out the connections among the three methods, their shared limitations, and directions for future work, and Chapter 8 concludes.

³ $\pi\mathbf{R}^2$ is under review for CoRL 2026.

Chapter 2

Background

This chapter introduces the technical concepts that recur throughout the thesis. All three methods act on *generalist generative policies*: policies that imitate demonstrations (Section 2.1), represent actions with diffusion or flow-matching models (Section 2.2), and predict actions in chunks that must eventually be executed on real hardware (Section 2.3).

2.1 Imitation Learning and Generalist Manipulation Policies

We study manipulation in the imitation-learning setting. A policy $\pi_{\theta}(a \mid o)$ is trained on a dataset of demonstrations, each a sequence of observation-action pairs, to reproduce the demonstrated behavior by maximizing the likelihood of demonstrated actions under the policy. Observations o combine one or more camera views, proprioception, and, increasingly, a natural-language instruction; actions a are typically end-effector or joint targets.

A *generalist* manipulation policy pushes this recipe to scale. Rather than training a separate policy per task, one trains a single language-conditioned policy on a large, diverse corpus of demonstrations spanning many tasks, scenes, and, in some cases, embodiments. Modern instances, often called vision-language-action (VLA) models [56, 57, 133], initialize from a large pretrained vision-language backbone and attach an action head, inheriting broad visual and semantic priors and then learning to act. The policies used in this thesis, such as π_0 [11, 45] and GR00T-family models [10, 80], follow this pattern. Because the space of valid behaviors for a given instruction is genuinely multi-modal, these policies are almost always *generative*: they

model a distribution over actions rather than a single deterministic mapping, which is what motivates the diffusion and flow-matching machinery reviewed next.

2.2 Diffusion Models and Flow Matching

Diffusion and flow-matching models are the generative backbone of every method in this thesis, so we review them in a common language. Both define a process that gradually transforms clean data $x_0 \sim p_{\text{data}}$ into noise and learn to reverse it.

Score-based diffusion. A forward process progressively adds Gaussian noise, producing a family of noised marginals p_t that interpolate between the data distribution at $t=0$ and an approximately Gaussian distribution at $t=1$. Sampling runs the process in reverse, and the only quantity needed to do so is the *score* of each noised marginal, $\nabla_x \log p_t(x)$. A network is trained by denoising score matching to approximate this score [42, 104] (equivalently, to predict the noise ϵ added to a sample), and new samples are drawn by numerically integrating the reverse-time dynamics with either a deterministic (probability-flow ODE) or a stochastic (reverse SDE) sampler. This score-function view is central to Chapter 5, which intervenes directly on the predicted score at inference time.

Flow matching. Flow matching [62] learns a time-dependent velocity field that transports a simple base distribution to the data distribution along (often straight-line) probability paths, trained by regressing the velocity onto a conditional target. At inference, one integrates an ODE from noise to data. Flow matching and diffusion are closely related, the learned velocity and the score are affine reparameterizations of one another [3], so techniques derived for one transfer to the other. Flow-matching policies are the substrate of Chapter 6.

Partial noising, then denoising (SDEdit). A key primitive for Chapter 4, introduced by SDEdit [75], is that one need not start sampling from pure noise. Given an approximate or corrupted sample, one can add a controlled *partial* amount of noise, up to an intermediate time s , and then denoise all the way from s back to 0. The result stays close to the input while being pulled onto the model’s data manifold; the choice of s trades off fidelity to the input against realism under the model. This is the mechanism by which a coarse trajectory can be refined by a frozen policy without any retraining.

2.3 Action Chunking and Real-Time Execution

Manipulation policies rarely predict a single action. Instead they predict an *action chunk* [130], a short horizon of H future actions, and execute some or all of it before replanning. Chunking improves temporal consistency and reduces compounding error, and it is the standard output format of diffusion and flow policies. But it introduces a tension central to Chapter 6: a chunk is generated from a single past observation and then executed *open-loop*, so the policy cannot respond to new observations that arrive mid-chunk. Replanning more frequently would restore reactivity, but running a large backbone plus multiple denoising steps is slow, and this inference *latency* caps how often a policy can replan, leaving committed actions stale. Diffusion forcing [16], which assigns each position in the chunk its own noise level, provides a flexible way to reconcile chunked prediction with frequent, partial replanning, and is the foundation $\pi\mathbf{R}^2$ builds on.

Chapter 3

Related Work

This chapter reviews the four areas the thesis builds on: generalist robot policies, learning manipulation from human and non-robot data, reactive execution of action-chunking policies, and controllable generation in diffusion and flow models. Method-specific baselines appear in the respective chapters.

3.1 Generalist Manipulation Policies

There has been a growing trend in developing ‘*generalist*’ robotic policies that can perform multiple tasks based on a specified goal in the form of an image or a language instruction. Collecting large-scale robot interaction datasets [32, 110] combined with behavior cloning and vision-language pre-training [9, 11, 14, 46, 48, 57, 133] is the predominant recipe for training such multi-task policies. However, since collecting large-scale robot interaction datasets in natural settings like homes and offices is challenging due to operational constraints, current best models still struggle to perform manipulation tasks zero-shot and thus are not yet deployable in-the-wild.

Architecturally, predicting a *chunk* of future actions rather than a single action, introduced by ACT [130], improves the temporal coherence of imitation-learned policies and has since become standard, including for mobile and contact-rich manipulation [33, 131]. This design has been widely adopted by expressive multi-modal policies based on diffusion [19] and flow matching [117, 127]. Foundation robotics models also couple chunked flow-matching action heads with large vision-language backbones, yielding vision-language-action (VLA) models that scale imitation learning across diverse tasks and embodiments [20, 55, 56, 133]. Notable examples include $\pi_0/\pi_{0.5}$ and GR00T N series, whose vision-language module feeds a flow-matching action head conditioned on robot proprioception [11, 45, 80].

Precisely to address this deployment gap, *DemoDiffusion* (Chapter 4) enables using such *generalist* diffusion policies to follow a human demonstration, letting them perform tasks that the pre-trained policy struggles to accomplish zero-shot.

3.2 Learning from Human and Non-Robot Data

Instead of requiring large-scale robot interaction datasets, a growing body of works have begun utilizing human videos and large-scale web videos for robotics. These approaches have been enabled by recent advances in computer vision for representation learning [27, 38, 132], predicting tracks [26, 51], and reconstructing hand-object interactions from monocular videos [88, 94, 118, 119]. Earlier works had already learned self-supervised visual representations from such non-robotic datasets [21, 35, 36], which can serve as the visual backbone of robotic policies [52, 72, 73, 78, 83, 114]. Recent works in this paradigm predict manipulation-relevant cues from web videos in the form of motion trajectories [8, 98, 107] and object affordances [5, 74, 99, 105] and combine these predictive models with a limited amount of robot interaction data for training policies via conditional behavior cloning. *DemoDiffusion* (Chapter 4) is orthogonal to these approaches in that instead of using human videos for training, it requires a single human demonstration during deployment for guiding a pre-trained generalist policy to perform a new task.

One-shot imitation from robot demonstrations. One-shot imitation aims to enable robots to perform a new task with guidance from a single demonstration. Prior works have explored this in the context of a robot demonstration provided at test time, via training demonstration-conditioned policies [22]. Recent works have developed learning-based visual servoing systems [109] and algorithms for identifying object invariances [129] to replicate the robot end-effector actions from the demonstration onto novel configurations of objects. However, requiring a robot demonstration for imitation is restrictive for ubiquitous deployment as end-users might find it challenging to tele-operate the robot to collect demonstrations in natural human environments.

One-shot imitation from human demonstrations. To imitate a human demonstration for robot manipulation, a straightforward approach is kinematic retargeting of the human hand pose to the robot end-effector pose per time-step, as demonstrated by some recent works [60, 61, 82]. Although simple to implement, the human-robot embodiment mismatch typically introduces errors in the retargeting and the open-loop

execution is brittle to object pose variations during deployment. To remedy this, another line of works seek to learn this retargeting implicitly by training a closed-loop policy from paired human-robot datasets [47, 84] enabling more flexible deployments. Since such paired datasets are difficult to collect, other works [4, 18, 37, 70] perform reinforcement learning with reward functions derived from video comparison. However, such methods require online interaction and resets, limiting their practicality. *DemoDiffusion* (Chapter 4) proposes an alternate paradigm: given the human demonstration, it refines retargeted human behavior using a pre-trained robot policy, avoiding brittle replays and enabling closed-loop control. This formulation avoids both expensive paired human-robot data collection and cumbersome test-time fine-tuning via interaction.

3.3 Real-Time and Reactive Policy Execution

For dynamic tasks, committing a (sub-)chunk open-loop is suboptimal, as it cannot react to sensory input arriving mid-execution. The high inference latency of modern VLAs has motivated methods for executing action chunks smoothly while the next prediction is computed [115]. RTC maintains continuity across asynchronous chunks via inference-time inpainting on a frozen action prefix, while Training-Time RTC instead folds this prefix conditioning into training, avoiding the extra inference-time cost [12, 13]. Streaming Diffusion Policy and Streaming Flow Policy emit actions incrementally from a diffusion-forcing-style variable-noise buffer maintained across observations [43, 49]. None, however, address the latency of repeatedly conditioning on a large semantic backbone; Streaming Diffusion Policy, for instance, uses compact visuomotor policies and runs synchronously. Concurrent to $\pi\mathbf{R}^2$ (Chapter 6), FASTER [69] similarly reduces reaction latency in flow-based VLAs via a diffusion-forcing-inspired horizon-aware schedule with action-prefix conditioning, but forwards the backbone once per chunk and conditions all actions on a single fixed observation, improving latency and smoothness without improving reactivity to sensory input arriving during execution.

Coupling a large backbone with multi-step flow-matching denoising only exacerbates the reactivity problem: the combined per-call latency limits how often the policy can replan. Crucially, this backbone latency would persist even under a diffusion-forcing schedule, since each emitted action still requires a full forward pass through the slow semantic backbone. $\pi\mathbf{R}^2$ (Chapter 6) instead exploits an asymmetry these models already expose (proprioception is far cheaper to process than vision and language), refreshing proprioceptive conditioning at every control tick while letting the slow

vision-language features update asynchronously, making this same class of models reactive.

3.4 Controllable Generation in Diffusion and Flow Models

A widely adopted mechanism for steering sampling in diffusion and flow models is to leverage Classifier-Free Guidance (CFG) [41]. While this allows one to trade off likelihood and diversity by controlling the effect of the conditioning on the drawn samples, it is fundamentally different from temperature scaling. Moreover, CFG cannot be applied to unconditional models and even for conditional ones, requires training with condition dropout. An alternative to CFG by Karras et al. [53] is to use a ‘bad version’ of the diffusion model for guidance, but its probabilistic interpretation is unclear and it also requires intermediate checkpoints which are not widely available even for open-weight models. In comparison, TSR (Chapter 5) serves as a plug-and-play technique compatible with any diffusion and flow matching model without any requirement on training. Moreover, as demonstrated for image generation, it is orthogonal to CFG and can be applied together for further improvement in quality.

We are not the first to consider temperature sampling in the context of diffusion models. In particular, Shih et al. [100] presented a technique to finetune diffusion (and autoregressive) models for temperature scaled inference. Their approach assigned an importance weight to each training sample based on its likelihood approximated by computing its ELBO with respect to a pretrained diffusion model on the same data. However, this approach is not training-free, making it difficult to leverage for large models and impossible in scenarios where training data is unavailable. An alternative training-free approach is to modify the reverse sampling by applying stochastic MCMC corrector steps [28, 104], but can incur 5 times additional computation. Recent work [101] avoid excessive corrector steps by re-weighting on batches of samples, which is inefficient when only one sample is desired (e.g., image generation, robot manipulation). In contrast, TSR is a training-free approach that has *zero inference overhead*.

Perhaps the closest to TSR in terms of being efficient and training-free is the technique of ‘Constant Noise Scaling’ (CNS) where one scales the stochastic noise at each sampling step by a constant. While CNS is the de facto approach to control sample variance in several domains [34, 120], as Shih et al. [100] point out, it is only a

‘*pseudo temperature*’ sampling method. Intuitively, the noise-to-score ratio controls the strength of exploration versus converging to distribution modes during sampling. By scaling down this ratio by a constant, CNS over-suppresses exploration at high noise levels and under-suppresses it at low noise levels, leading to inadequate exploration of the data space when the model should recover global structure. CNS behaves differently from temperature scaling and drops modes even for simple distributions. Moreover, CNS only applies to stochastic samplers and struggles with modern flow-matching models. In contrast, TSR (Chapter 5) proposes a time-dependent score scaling schedule that preserves the global structure of the sampled distribution and is compatible with both deterministic and stochastic samplers.

Chapter 4

Bridging the Embodiment Gap: One-Shot Imitation from Human Demonstrations

Building on the manipulation background of Chapter 2, this chapter presents *DemoDiffusion*, which closes the *data* gap identified in Chapter 1: adapting a pretrained generalist policy to a new task without collecting task-specific teleoperation data.¹

How do we build robot manipulation systems that can be readily deployed in unstructured human environments? One possible answer is to learn ‘generalist’ policies that are capable of accomplishing any generic task (specified via some description, *e.g.* language or an image goal) in any environment. Indeed, there is a general optimism about this paradigm, reflected in several ongoing efforts to collect large-scale demonstration datasets to train such policies. While these efforts have produced unified policies capable of diverse tasks [11, 14, 48, 57, 133], these policies still struggle to perform meaningfully when deployed zero-shot to novel environments or asked to perform unseen tasks. Deployment thus often requires additional fine-tuning using task-and-scene-specific robot demonstrations [11], but this is a non-trivial overhead, as collecting robot demonstrations in the real world can be time-consuming and beyond the expertise of an average user. We instead pursue an alternate deployment mechanism: leveraging a pre-trained generalist policy to perform a task by imitating a single *human* demonstration.

We are of course not the first to consider this goal of allowing robots to imitate any given human demonstration. One common approach [17, 61, 82, 111] is to instantiate

¹DemoDiffusion was accepted to ICRA 2026 [85].

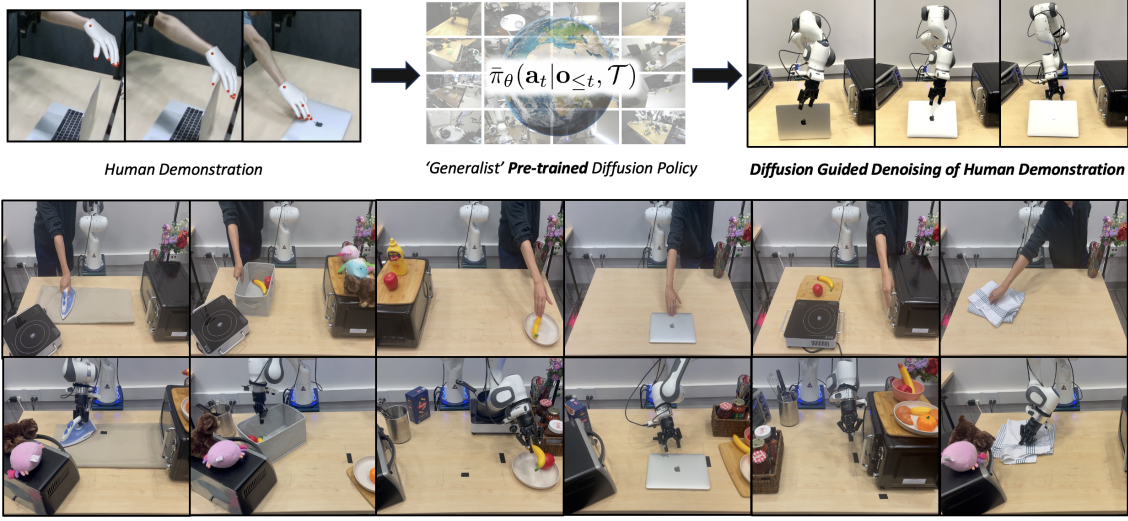


Figure 4.1: **Overview of *DemoDiffusion*.** We show how *generalist* pre-trained diffusion policies can be used to follow a generic human demonstration of a manipulation task during deployment. Our real-world manipulation results encompass a wide diversity of tasks involving everyday objects.

this as a kinematic retargeting task and compute open-loop robot actions that maximize a manually defined similarity between the achieved robot end-effector configuration and the observed human hand poses (*e.g.* matching locations of fingertips). However, the human-robot embodiment mismatch makes it difficult for the retargeted actions to achieve precisely the same effects as the human ones, and the open-loop execution further makes the approach brittle to noise and scene variations. Another line of work for human imitation attempts to learn robot policies through online reinforcement learning [4, 18, 37, 67, 68], where the human demonstration helps define reward functions. While this can overcome the embodiment gap, such test-time online RL requires hours of online interaction and resets, making it difficult to adopt for generic real-world manipulation tasks, especially in safety-critical scenarios. We seek an approach that, like kinematic retargeting, can be deployed one-shot (*i.e.* without test-time training), but still benefits from learned priors for precise closed-loop interaction.

Our approach builds on the insight that pre-trained diffusion policies can act as priors for robot action. Inspired by prior work on leveraging pre-trained diffusion models for image editing [75], we present *DemoDiffusion*, a formulation that utilizes a diffusion policy trained on robot interaction data to synthesize coherent robot actions from a human demonstration. Specifically, we first perform kinematic retargeting by extracting human hand poses from the demonstration and obtain an open-loop robot action trajectory. While this trajectory is typically suboptimal due to embodiment

differences and a lack of closed-loop feedback, it serves as an effective initialization that can be improved via a diffusion policy. We do so by injecting Gaussian noise and applying the pre-trained diffusion policy to iteratively denoise the trajectory conditioned on robot observations, yielding a refined, executable sequence of robot actions. *DemoDiffusion* thus enables the robot to use the pre-trained policy as a prior and adapt the human-derived trajectory to its own embodiment and environment in a closed-loop manner.

In summary, we propose *DemoDiffusion*: a framework for robotic manipulation that allows a robot to perform generic tasks in natural environments by following a human demonstration with guidance from a pre-trained diffusion policy. Importantly, our framework does not require any robot demonstration of the target task within the target environment, online interaction, or fine-tuning. Our experiments across simulation and real-world environments show that *DemoDiffusion* surpasses both the base policy and the retargeted trajectory, achieving 83.8% success across eight real-world tasks, even those where the pre-trained *generalist* policy fails entirely.

4.1 Method

We target the problem of one-shot visual imitation. Given a human demonstration $\mathbf{D}$ depicting a manipulation task with description $\mathcal{T}$, we want a robot manipulator to perform the same task. Unlike prior work that relies on test-time RL training [4] or paired human-robot datasets [47], we wish to enable such imitation ‘one-shot’, assuming access to a closed-loop diffusion policy $\bar{\pi}_{\theta}(\mathbf{a}_t \mid \mathbf{o}_{\leq t}, \mathcal{T})$ pre-trained on some broad robot interaction dataset $\mathcal{D}_{robot}$. Our assumption on the human demonstration $\mathbf{D}$ is also very minimal: it can either be an RGBD video or a multi-view video of a human doing a task, such that 3D hand poses per time-step can be reliably extracted from it.

4.1.1 Overview

Our approach is based on two insights. The first is that the trajectory of the hand pose in the human demonstration $\mathbf{D}$ provides useful information about the approximate trajectory the robot end-effector should follow, and we can perform *kinematic retargeting* of the hand trajectory $\{\mathbf{h}_t\}_{t=0}^T$ to an open-loop robot end-effector trajectory $\{\hat{\mathbf{a}}_t\}_{t=0}^T$. The second insight is that the kinematically retargeted robot trajectory has the correct form of motion, but these actions may not be very precise within the distribution of

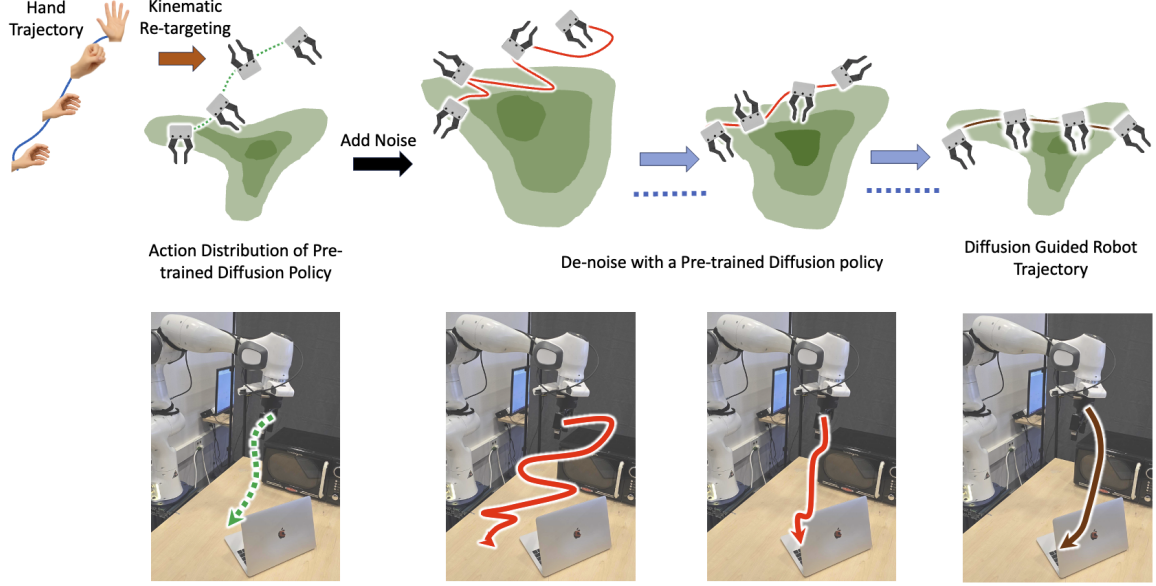


Figure 4.2: **Retargeted human hand trajectory to a closed-loop robot action sequence, for the task $\mathcal{T}$: “shut down the laptop”.** The dotted line shows the trajectory of robot end-effector poses after kinematic retargeting. The olive contour plot depicts the distribution of trajectories from a pre-trained diffusion policy. Given a kinematic retargeting, we first perturb it with Gaussian noise and progressively remove the noise by simulating the reverse SDE with the diffusion policy. This process gradually projects a potentially infeasible but approximately correct retargeting onto the manifold of plausible robot actions that can perform real-world manipulation, in this case closing the laptop without missing the edge.

plausible robot actions given the current observation. A diffusion policy models this likelihood, and we can use a pre-trained ‘*generalist*’ diffusion policy $\bar{\pi}_{\theta}(\mathbf{a}_t \mid \mathbf{o}_{\leq t}, \mathcal{T})$ to refine the retargeted robot actions in a closed-loop manner, thus inferring actions that, while still similar to the human demonstration, are more likely under the policy and better aligned with the robot embodiment (see Figure 4.2). We describe both of these steps in detail below.

4.1.2 Kinematic Retargeting of Human Hand Trajectories to Robot End-Effector Poses

Given a human demonstration $\mathbf{D}$ of a manipulation task $\mathcal{T}$, our first step is to extract the 3D hand pose trajectory $\{\mathbf{h}_t\}_{t=0}^T$. Each hand pose $\mathbf{h}_t \in \mathbb{R}^{3 \times J}$ corresponds to the 3D locations of J keypoints (*e.g.* wrist and fingertips), which we estimate using a pre-trained monocular hand pose estimator applied to each video frame [87]. These keypoints encode the motion of the human performing the task and form the basis for

retargeting to a robotic end-effector.

To translate this human motion into a robot-executable trajectory, we define a simple geometric mapping function $f_{\text{retarget}} : \mathbb{R}^{3 \times J} \rightarrow \mathbb{R}^6$ that converts the human hand pose $\mathbf{h}_t$ into a robot configuration $\hat{\mathbf{a}}_t = f_{\text{retarget}}(\mathbf{h}_t)$. The mapping aligns the wrist pose of the human to the robot end-effector pose. For a two-finger gripper, we use the distance between the thumb and the remaining fingers of the hand mesh to infer a binary robot grasp, and for a dexterous robotic hand we match the robot hand’s fingertip positions to those of a human hand using inverse kinematics. We set the initial configuration of the robot as the kinematically retargeted initial configuration from the human demonstration.

This kinematic retargeting procedure yields a full trajectory $\{\hat{\mathbf{a}}_t\}_{t=0}^T$ in the robot’s configuration space, which can be executed as an open-loop policy. However, due to differences in morphology and embodiment between humans and robots, the absence of environment feedback, and the inaccuracy of the hand estimation module, this trajectory often leads to suboptimal or unstable behavior during open-loop execution. We thus treat this as the basis for searching for a plausible robot action trajectory via denoising with a (pre-trained) closed-loop diffusion policy.

4.1.3 Closed-Loop Denoising of Robot Actions with a Pre-Trained Diffusion Policy

A diffusion policy $\bar{\pi}_\theta(\mathbf{a}_t \mid \mathbf{o}_{\leq t}, \mathcal{T})$ [19, 91] pre-trained on diverse offline robot interaction data reliably models the distribution of plausible robot actions $\mathbf{a}_t$ given previous observations $\mathbf{o}_{\leq t}$. To predict actions $\mathbf{a}_t$, diffusion policies start with Gaussian noise $\tilde{\mathbf{a}}_t^{(S)} \sim \mathbf{N}(0, I)$ and iteratively denoise it at different diffusion steps s :

$$\tilde{\mathbf{a}}_t^{(s-1)} = \bar{\pi}_\theta(\tilde{\mathbf{a}}_t^{(s)}, \mathbf{o}_{\leq t}), \quad s = S, S-1, \dots, 0. \quad (4.1)$$

We use this to modify the kinematically retargeted trajectory $\{\hat{\mathbf{a}}_t\}_{t=0}^T$ to obtain robot actions that follow the high-level motion in the human trajectory and still lie within the distribution of plausible actions under $\bar{\pi}_\theta$. To do this, we modify this typical reverse diffusion process so that, instead of starting with pure noise $\tilde{\mathbf{a}}_t^{(S)}$ at step S , we start from an intermediate diffusion step s^* such that $0 < s^* < S$. We define $\{\tilde{\mathbf{a}}_t^{(s^*)}\}_{t=0}^T$ to be a noisy version of the kinematically retargeted trajectory as follows:

$$\tilde{\mathbf{a}}_t^{(s^*)} = \sqrt{\alpha_{s^*}} \hat{\mathbf{a}}_t + \sqrt{1 - \alpha_{s^*}} \epsilon_t, \quad \epsilon_t \sim \mathbf{N}(0, I). \quad (4.2)$$

Here α corresponds to the diffusion schedule of the pre-trained policy. This procedure, inspired by SDEdit [75] which adopted a similar approach for image editing, relies on the assumption that $\{\hat{\mathbf{a}}_t\}_{t=0}^T$ is an approximate version of the ideal action trajectory that should be executed by the robot, and it potentially lies outside the distribution of feasible actions under the pre-trained diffusion policy $\bar{\pi}_\theta$. Intuitively, adding moderate noise to the retargeted trajectory and then denoising preserves the high-level structure of the human demonstration while allowing the policy to correct low-level infeasibilities from the embodiment gap. The diffusion policy $\bar{\pi}_\theta$ then performs iterative denoising steps, conditioned on the robot’s observations $\mathbf{o}_{\leq t}$, to refine this noisy trajectory into feasible robot actions, following Equation 4.1.

After s^* denoising steps, the final output $\mathbf{a}_t = \tilde{\mathbf{a}}_t^{(0)}$ is deployed on the robot. Importantly, this process is carried out in a closed-loop manner: the policy uses real-time observations from the cameras in the scene to iteratively improve its predictions, thereby compensating for embodiment mismatch and external perturbations (*e.g.* object slippage or occlusion). The key hyperparameter in this process for *DemoDiffusion* is the diffusion step s^* , which trades off between faithfulness to the demonstration and likelihood under the robot policy: in the limit $s^* = S$ we recover a rollout from the base policy $\bar{\pi}_\theta$, and in the limit $s^* = 0$ we recover the kinematically retargeted trajectory $\{\hat{\mathbf{a}}_t\}_{t=0}^T$. While derived in the context of diffusion, note that this iterative refinement perspective also applies to flow-matching [62], which we demonstrate in our real-world experiments.

4.2 Experiments

We evaluate our approach on dexterous grasping in simulation and across diverse real-world tasks comprising prehensile and non-prehensile table-top manipulation. Through our experiments, we aim to answer the following research questions:

- Can *DemoDiffusion* outperform pure kinematic retargeting from the human demonstration?
- Do the human demonstrations allow *DemoDiffusion* to perform new tasks where the pre-trained diffusion policy fails?
- How can we effectively trade off between faithfulness to the human demonstration and performing the task reliably, with a varying noise level s^*/S ?

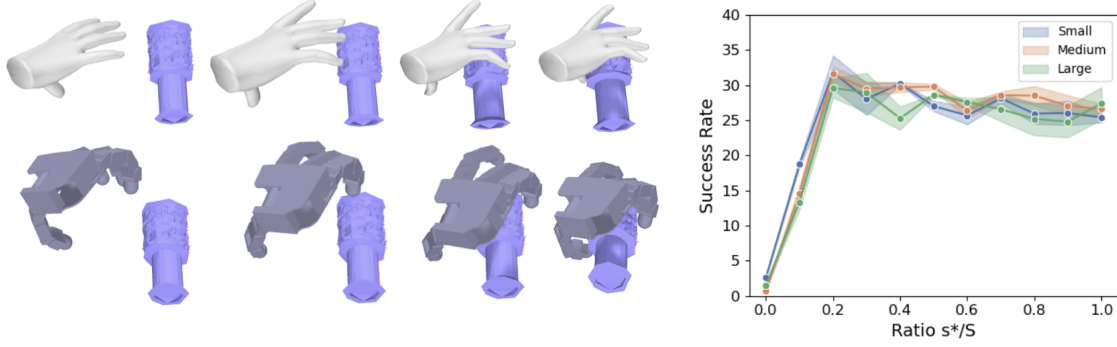


Figure 4.3: **Dexterous grasping results in simulation.** (Left) A human demonstration and *DemoDiffusion* rollout for dexterous grasping. (Right) Success rates (mean $\pm$ std over 3 seeds) as a function of the diffusion step s^* . Here, $s^*/S = 0$ corresponds to kinematic retargeting and $s^*/S = 1$ corresponds to the robot policy.

- How robust is *DemoDiffusion* to noise in the human demonstration, or to a different kinematic retargeting?

4.2.1 Dexterous Grasping in Simulation

To verify our intuition, we first consider a simulation environment where the target task is restricted to picking up a generic object with a 16-DOF four-fingered Allegro hand. Specifically, we train a dexterous grasping policy across a small set of generic objects in simulation, and test our method on human hand grasping trajectories on a different set of objects. This serves as the pre-trained diffusion model for this experiment.

For training the robot policy, we collect a total of 985 grasping trajectories of the Allegro hand over 58 training objects [125] (randomly sampled 26 and 32 objects from ShapeNet [15] and PartNet [76], respectively), resulting in a 22.2% success rate. We use a variant of 3D Diffusion Policy [123], which takes 3D point tracks of an object instead of point clouds. At test time, we provide human grasping trajectories on a subset of the Objaverse dataset [23], which are unseen during training. The test set contains 1220 objects, with one human grasping trajectory per object, provided by GraspXL [125]. Note that there is no error from hand estimation or the initial configuration of the object in this setup. We use ground-truth human hand 3D keypoints, and the object is located at the same location as in the human demonstration.

Results. As in GraspXL [125], we group the objects based on their size, resulting in small, medium, and large objects. We compare *DemoDiffusion* with two baselines.

Table 4.1: **Simulated dexterous grasping results.** The numbers show the average of success rates over 3 seeds. We use $s^*/S = 0.2$, where the robot policy uses $S = 1000$. *DemoDiffusion* consistently outperforms both baselines across all groups.

Group	Robot Policy	Kinematic	<i>DemoDiffusion</i>
<i>Small</i>	25.4	2.6	31.8
<i>Medium</i>	26.6	0.7	31.6
<i>Large</i>	27.4	1.5	29.6
Average	26.5	1.6	31.0

The first is open-loop kinematic retargeting obtained from the human demonstration. The second is directly deploying the robot policy we trained. Table 4.1 shows the main results. Our method outperforms the baselines, with the largest gains on small objects. We hypothesize this is because larger objects are easier to grasp, supported by the base robot policy’s performance improving for larger objects. This experiment shows that even in a highly controlled setup where the robot environment is initialized identically to that of the human demonstration, kinematic retargeting cannot solve the task, while slightly refining with a generalist grasping policy significantly improves performance. Using kinematic retargeting to initiate the denoising process also helps the denoising process itself, outperforming the robot policy. Additionally, *DemoDiffusion* achieves higher inference speed, by a factor of S/s^* .

Ablation studies. We analyze the influence of s^* on *DemoDiffusion* in Figure 4.3. Overall, we observe a consistent trend of performance gain from $s^*/S = 1$ to $s^*/S = 0.2$, showing that *DemoDiffusion* is robust to the choice of this hyperparameter.

4.2.2 Real-World Manipulation

For the real-world experiments, we use a Franka Emika Panda arm equipped with a two-finger gripper from Robotiq. We use the publicly available pre-trained flow-matching policy $\bar{\pi}_\theta(\mathbf{a}_t \mid \mathbf{o}_{\leq t}, \mathcal{T})$ called Pi-0, released by Physical Intelligence [11], trained on the DROID dataset [55]. This policy takes as input a language task specification $\mathcal{T}$ and observations from two cameras in the scene, and outputs per-timestep joint velocities. This policy was trained on a large offline dataset of tele-operated robot demonstrations, and is a *generalist* policy that can perform a broad set of tasks. We do not fine-tune or adapt this policy in any way and directly use it for *DemoDiffusion*. We use an off-the-shelf monocular hand reconstruction model, Hamer [87], to obtain



Figure 4.4: **Real-world manipulation tasks.** Human demonstrations for the eight evaluation tasks, shown as two frames per task. Tasks span prehensile and non-prehensile manipulation, including grasping, pushing, closing, wiping, and placing.

Table 4.2: **Quantitative results for real-world manipulation.** Success rates (%) are over 10 trials per task.

Task	Pi-0 Kinematic		<i>DemoDiffusion</i>
Shut Down Laptop	20	10	60
Close Microwave	10	80	90
Drag Basket	10	80	80
Wipe Table	50	0	100
Iron Curtain	20	70	90
Pick up Bear	0	40	60
Pick Place Bowl	0	70	100
Pick Place Banana	0	70	90
Average	13.8	52.5	83.8

per-timestep hand mesh reconstructions from each of the external cameras given a human demonstration in the scene. Our experiments follow the protocol of a human first demonstrating an object manipulation in the scene, followed by robot execution after re-setting the scene.

Workspace. Figure 4.5 describes our real-robot manipulation setup. The scene has 5 cameras, including 4 Realsense external cameras and 1 Zed-Mini wrist camera mounted on the robot gripper. We calibrate the four external cameras and use them to triangulate the 3D hand pose so that we can obtain the position and orientation of a human hand in world coordinates, with the origin at the base of the robot. The pre-trained policy Pi-0 requires just two cameras (marked in purple), and the policy does not use any calibration information. While Pi-0 uses a Zed 2 camera for the external view, we use a Realsense camera due to hardware availability, which



Figure 4.5: **Workspace with five cameras.** We use the four external cameras for triangulation to obtain the global pose of the hand mesh from a human demonstration. The pre-trained policy uses the two cameras marked in purple.

introduces additional difficulty.

Tasks. We perform evaluations for 8 different manipulation tasks, whose language descriptions and human demonstrations are described in Figure 4.4. Each of these tasks requires a different manipulation strategy involving either prehensile or non-prehensile manipulation. We collect just a single human demonstration per task, but for robot execution we make organic variations in the location of objects in the scene. Additionally, we add other objects to the scene as visual distractors, to see whether *DemoDiffusion* can generalize under such settings. All variations are consistent across the baselines and our method.

Baselines. For the real-world experiments, we compare *DemoDiffusion* with two baselines. The first is open-loop kinematic retargeting obtained from the human demonstration. The second is directly deploying the Pi-0 model with the language instruction for the respective task. Since Pi-0 operates in the joint velocity space, for *DemoDiffusion* we first convert the kinematically retargeted trajectory to the joint velocity space with IK and then add noise to the trajectory, followed by denoising with Pi-0. We deploy *DemoDiffusion* with a heuristically selected noise level s^*/S



Figure 4.6: **Qualitative comparisons for real-robot manipulation.** Rollout progressions (start, intermediate, final frame) for two tasks. Kinematic retargeting (top) produces plausible motion but loses contact before completing the task. Pi-0 (middle) performs general reaching but fails to manipulate the correct object. *DemoDiffusion* (bottom) reaches the object and maintains contact through task completion. See the project page for full videos.

depending on the task – preferring the kinematic trajectory with $s^*/S = 0.2$ when Pi-0 has a zero success rate, and $s^*/S = 0.4$ otherwise. Note that in the limit of 0 noise added, we recover the first baseline, kinematic retargeting, and in the limit of maximum noise 1 we obtain the second baseline, Pi-0.

Qualitative visualizations. In Figure 4.6, we show rollouts for two tasks across all the methods. We observe that Pi-0 can only perform general reaching behaviors: it either fails to follow the manipulation trajectory needed for the task or confuses the target object with a distractor. On the other hand, we observe that (open-loop) kinematic retargeting from the human demonstration has a plausible motion of the robot end-effector, but it completely misses grasping the iron and loses contact with the laptop before fully closing it. Compared to these, *DemoDiffusion* reaches the respective object and also follows the approximate motion of the retargeted trajectory while remaining in contact with the object to solve the tasks.

Quantitative comparisons. In Table 4.2, we compare *DemoDiffusion* and the baselines across all the real-world manipulation tasks. *DemoDiffusion* matches or outperforms both baselines across all 8 tasks, with the largest gains on tasks requiring precise contact – *e.g.* 60% vs. 10–20% on shutting the laptop and 100% vs. 0–50% on

Table 4.3: **Robustness of *DemoDiffusion*.** We ablate *DemoDiffusion* under noisy 3D keypoints (2nd row) and with a different kinematic retargeting (3rd row). For noisy 3D keypoints, we randomly shift the tracked hand 3D keypoints by 5 cm. For the different kinematic retargeting, we use the distance between the thumb tip and the index fingertip instead of all fingers.

	Shut Laptop	Pick Place Banana
<i>DemoDiffusion</i>	60	90
+5cm noise	40	70
with Thumb+Index	30	80
Thumb+Index	0	30

wiping the table. Notably, even for tasks where both baselines fail, *DemoDiffusion* succeeds by operating at an intermediate noise level s^* that combines the trajectory structure from the human demonstration with the policy’s closed-loop refinement.

Noisy human demonstrations. To evaluate the robustness of *DemoDiffusion* to imperfect inputs, we injected noise into the tracked human trajectories and evaluated *DemoDiffusion*. Specifically, we place the object at the same location as in the human demonstration, and randomly shift the tracked hand 3D keypoints by 5 cm. The results in Table 4.3 indicate that *DemoDiffusion* maintains high-quality performance despite the presence of tracking noise, thereby reducing the reliance on highly accurate 3D annotations in the human demonstration.

Different kinematic retargeting. Lastly, we further assessed the sensitivity of our method to the choice of retargeting by evaluating an alternative approach (‘Thumb+Index’). Specifically, we use the distance between the thumb and index finger [60], instead of all fingers, to infer the grasp. Table 4.3 shows the results. While this alternative retargeting itself shows degraded performance compared to the previous retargeting, *DemoDiffusion* consistently improves task performance despite the suboptimal retargeting.

4.3 Limitations

Our approach has several limitations. First, it assumes that the robot should act similarly to the human to successfully complete the task, which may not hold in scenarios requiring different strategies due to embodiment or environmental constraints.

Second, while effective for one-shot imitation (not requiring training or test-time fine-tuning), the method does not produce a reusable task policy that can generalize across arbitrary variations of the task. Third, the quality of the retargeted trajectory is crucial: accurate 3D human motion capture is challenging, and errors in retargeting can impact downstream performance, though our method shows a degree of robustness to such noise. Additionally, the method implicitly assumes that the timing and speed of human and robot actions are aligned; extending the approach to allow for temporal alignment at test time is a promising direction of future work.

4.4 Summary

We presented *DemoDiffusion*, a simple approach that leverages *generalist* diffusion models for imitating a human demonstration. By retargeting hand motion from the human demonstration into an approximate robot end-effector trajectory and modifying it through denoising with a pre-trained diffusion policy, *DemoDiffusion* enables the robot to produce actions that are both aligned with the demonstrated intent and contextually plausible. Our method consistently outperforms both the base diffusion policy and the raw retargeted trajectory across diverse tasks in simulation and the real world, without requiring additional training, paired demonstrations, or reward functions. We envision that this can serve as a starting point for future efforts at human imitation across generic scenarios, and perhaps also yield an improved exploration strategy for methods pursuing policy adaptation with online RL.

Chapter 5

Steering Sampling Diversity via Temporal Score Rescaling

This chapter presents TSR (Temporal Score Rescaling), a mechanism to steer the sampling diversity of diffusion and flow models, letting a user sample from a sharper or broader distribution than the one the model was trained on.¹ Whereas the preceding chapter adapted a frozen generalist robot policy to a new task (Chapter 4), TSR closes the *inference* gap and is the most general contribution of this thesis: the same mechanism applies not only to robot policies but to any off-the-shelf diffusion or flow model, from image generation to protein design.

Score-based generative models, such as denoising diffusion [42] and flow matching [63, 66], have become ubiquitous across AI applications. Given training data $\{\mathbf{x}^n\}$, they can model the underlying data distribution $p(\mathbf{x})$ (or $p(\mathbf{x}|\mathbf{c})$ for conditional settings) and at inference, they allow drawing samples $\mathbf{x} \sim p(\mathbf{x})$ *e.g.* to generate novel images. However, in certain applications, we may not want to truly sample the original data distribution. For example, when predicting depth from RGB input, we may want the more likely estimate(s) as output. In contrast, an artist exploring design choices may want the trained image generative model to yield more diverse samples, even if they may be somewhat less likely in the data. In this work, we ask whether we can steer the sampling process of diffusion or flow matching models to output more likely (or conversely, more diverse) samples than the original training data.

This process of trading off sample likelihood and diversity at inference is commonly referred to as *temperature sampling* [40] (a higher temperature leads to diverse samples, and a lower temperature leads to more likely ones). While prior methods have investigated temperature sampling for score-based generative models like denoising diffusion,

¹Temporal Score Rescaling was accepted to ICML 2026 [116].

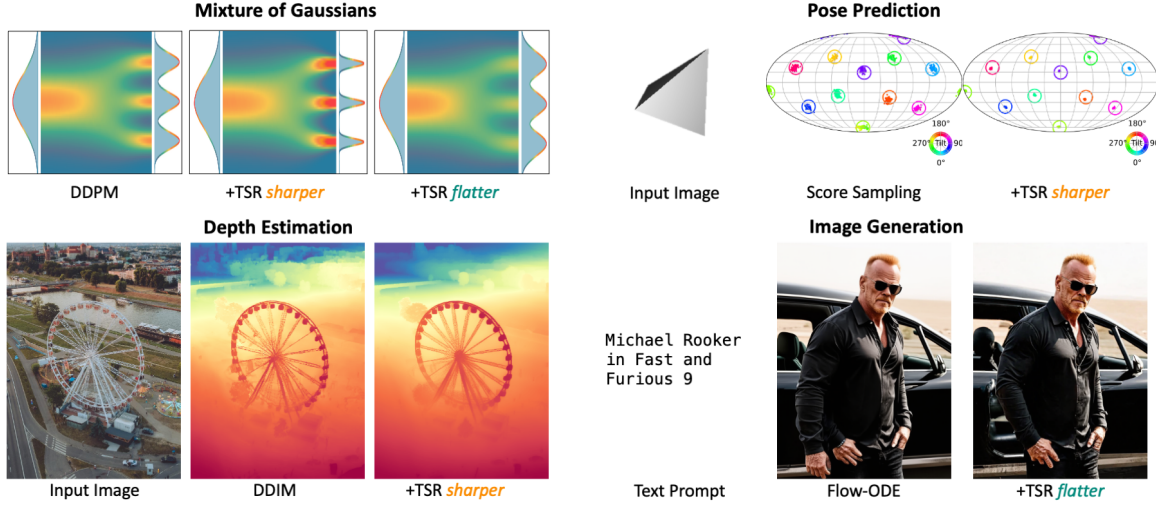


Figure 5.1: **Temporal Score Rescaling (TSR)**. TSR provides a mechanism to steer the sampling diversity of diffusion and flow models at inference. *Top-left:* Probability density evolution when sampling a 1D Gaussian mixture with DDPM, and the effects of TSR, which can control the sampling process to yield sharper or flatter distributions. *Top-right, bottom:* TSR can be applied to any pre-trained diffusion or flow model, improving performance across diverse domains such as pose prediction, depth estimation, and image generation.

developing an efficient temperature sampling method for pre-trained diffusion/flow models remains an open challenge. For example, commonly leveraged techniques like classifier-free guidance [41] or variance-reduced sampling [34, 120] can trade off sampling diversity and likelihood, but as we show later, these are not probabilistically interpretable as temperature scaling the data distribution. Conversely, methods such as likelihood-weighted finetuning [100] or Langevin correction [28, 104] can indeed allow temperature sampling, but at the cost of additional training or significantly increased inference-time computation. In this work, we instead seek to develop a (local) temperature sampling method that is: a) *training free i.e.* does not require fine-tuning or distilling a pre-trained model, b) compatible with deterministic samplers *e.g.* DDIM [102], c) efficient *i.e.* does not increase the number of score evaluations at inference, and d) provably correct for some simple distributions.

Towards developing such an approach, we note that denoising diffusion and flow matching models define a forward process to induce noisy data distributions $p(\mathbf{x}_t)$ and train neural networks to approximate the corresponding score functions $\nabla \log p(\mathbf{x}_t)$. We ask whether one can analytically relate these to the score of the (hypothetical) distributions $\bar{p}(\mathbf{x}_t)$ that would be induced by the forward process if the original data distribution were temperature scaled. We study the case of mixture of isotropic

Gaussians, and derive a simple (time-dependent) rescaling function. As the reverse sampling process for sampling flow/diffusion models relies only on the learned score functions, our derived rescaling thus allows a training-free approach by simply scaling the inferred score at each inference step. While the analytical derivation is restricted to a simple setting, we show that our approach can be generally interpreted as a ‘local’ temperature sampling method, where it does not alter the overall distribution of global modes, but controls the local variance of samples around it.

We perform experiments to highlight the broad applicability of TSR. We show that it can efficiently allow local temperature sampling for denoising diffusion and flow matching models and is compatible with generic stochastic and deterministic samplers. We study diverse applications like image generation, depth estimation, pose prediction, robot manipulation, and protein generation. Across these applications, we show that TSR can provide a plug-and-play solution to control the sampling diversity of pre-trained models and leads to consistent performance gains *e.g.* allowing more precise depth and pose inference, or enabling image generation to better match real data distribution.

5.1 Formulation

5.1.1 Preliminaries

Both diffusion and flow matching models fall under the family of stochastic interpolants [3], which convert samples from data distribution $\mathbf{x}_0 \sim p_0(\mathbf{x})$ to Gaussian noise $\epsilon \sim \mathcal{N}(0, \mathbf{I})$. The interpolant process can be defined as:

$$\mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \epsilon \quad (5.1)$$

Different noise schedules α_t, σ_t correspond to different formulations of stochastic interpolants. For example, for flow matching models, it is common to set $\alpha_t = 1 - t$, $\sigma_t = t$, while for variance-preserving diffusion models, they are defined such that $\alpha_t^2 + \sigma_t^2 = 1$.

We can sample from the data distribution by training a model $\mathbf{s}_\theta(\mathbf{x}, t) = \nabla \log p_t(\mathbf{x})$ that estimates the score of the noisy distribution. Starting from $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$, the sampling process usually solves either a reverse-time SDE or a probability flow ODE. In practice, the learned model could predict various equivalent parameterizations of the score, such as noise $\epsilon_\theta(\mathbf{x}_t, t)$ (common in denoising diffusion) or the probability flow velocity $\mathbf{v}_\theta(\mathbf{x}_t, t)$ (common in flow matching), which can all be expressed as linear

combinations of score and $\mathbf{x}_t$ (see Section 5.1.3).

5.1.2 Temporal Score Rescaling

Given a pre-trained score function $\mathbf{s}_\theta$, we are interested in designing a temperature sampling process that does not require training or additional computation at inference. In particular, we propose a mechanism that achieves *local temperature scaling*, which steers the variance of the sampled distribution while preserving the global distribution structure (*e.g.* without mode dropping). More formally, we define local temperature scaling as the task that takes in a data distribution $p_0(\mathbf{x})$ modeled as a mixture of (*an unknown set of*) Gaussians and generates the corresponding ‘sharper’ or ‘flatter’ distributions $\tilde{p}_0^k(\mathbf{x})$ (parameterized by k):

$$\begin{aligned} p_0(\mathbf{x}) &\equiv \sum_m w_m \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m) \\ \Rightarrow \tilde{p}_0^k(\mathbf{x}) &\equiv \sum_m w_m \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_m, \frac{1}{k} \boldsymbol{\Sigma}_m) \end{aligned}$$

Intuitively, $\tilde{p}_0^k(\mathbf{x})$ represents a distribution where the variance near each local mode in the data distribution is scaled by $\frac{1}{k}$, while preserving all the means and weights. Such a local scaling effect is different from the traditional temperature scaling that would change the weights of modes and alter the distribution structure. We now formulate our problem statement as: How can we alter the pretrained score function $\mathbf{s}_\theta$ so that a diffusion or flow sampler yields $\tilde{p}_0^k(\mathbf{x})$ instead of $p_0(\mathbf{x})$?

Isotropic Gaussian Data. To instantiate this, we start with a simple scenario where the data are drawn from a single isotropic Gaussian distribution $\mathbf{x}_0 \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2 \mathbf{I})$. The target is to sample from the locally scaled distribution $\tilde{p}_0^k(\mathbf{x}) \equiv \mathcal{N}(\boldsymbol{\mu}, \frac{\sigma^2}{k} \mathbf{I})$. Under the stochastic interpolant process (Equation 5.1), we define $p_t(\mathbf{x})$, $\tilde{p}_t^k(\mathbf{x})$ as the noisy distributions at time t for the original and scaled data distribution, respectively. Since both the original and scaled data distributions are Gaussian, their corresponding noisy distribution can also be shown to be Gaussian:

$$\begin{aligned} p_t(\mathbf{x}) &= \mathcal{N}(\alpha_t \boldsymbol{\mu}, (\alpha_t^2 \sigma^2 + \sigma_t^2) \mathbf{I}) \\ \tilde{p}_t^k(\mathbf{x}) &= \mathcal{N}(\alpha_t \boldsymbol{\mu}, (\alpha_t^2 \frac{\sigma^2}{k} + \sigma_t^2) \mathbf{I}) \end{aligned} \tag{5.2}$$

Then, we can derive the corresponding score functions for the above distributions:

$$\begin{aligned}\nabla \log p_t(\mathbf{x}) &= -\frac{\mathbf{x} - \alpha_t \boldsymbol{\mu}}{\alpha_t^2 \sigma^2 + \sigma_t^2}, \\ \nabla \log \tilde{p}_t^k(\mathbf{x}) &= -\frac{\mathbf{x} - \alpha_t \boldsymbol{\mu}}{\alpha_t^2 \frac{\sigma^2}{k} + \sigma_t^2}\end{aligned}\tag{5.3}$$

Comparing the two score functions above, we observe that the score for the scaled distribution and the score for the original distribution follow a time-dependent linear relationship:

$$\nabla \log \tilde{p}_t^k(\mathbf{x}) = \frac{\eta_t \sigma^2 + 1}{\eta_t \frac{\sigma^2}{k} + 1} \nabla \log p_t(\mathbf{x})\tag{5.4}$$

where $\eta_t = \alpha_t^2 / \sigma_t^2$ is the signal-to-noise ratio. Note that $k = 1.0$ recovers the original score. Given a score estimator $\mathbf{s}_\theta(\mathbf{x}, t) = \nabla \log p_t(\mathbf{x})$, we can compute the score of $\tilde{p}_t^k$ with the above score rescaling equation and thus sample from $\tilde{p}_0^k$ from the same sampling process.

Mixture of Gaussians. We can show that the score ratio relationship (Equation 5.4) is also a valid approximation if the data distribution is a mixture of *well-separated* isotropic Gaussians. In Section B.2, we prove that the expected error between the score computed by Equation 5.4 and the real score are bounded at all timesteps t . On the high level, we derive an exponential bound for small t , where the modes are well-separated and only one Gaussian component dominates. For large t , we derive a polynomial bound based on the intuition that the distributions are similar to pure noise $\mathcal{N}(0, \mathbf{I})$. The error vanishes at both ends when t converges to 0 or 1. The maximum error at any intermediate t also converges to zero as the modes become more separated. We also empirically verify these results in Section B.1.5.

5.1.3 Steering Inference in Diffusion and Flow Matching

While the above analytical derivation for a score rescaling function focused on simple distributions, we empirically find that it can be applied across generic distributions and we operationalize Equation 5.4 to define TSR sampling, a simple algorithm for steering sampling in diffusion and flow models:

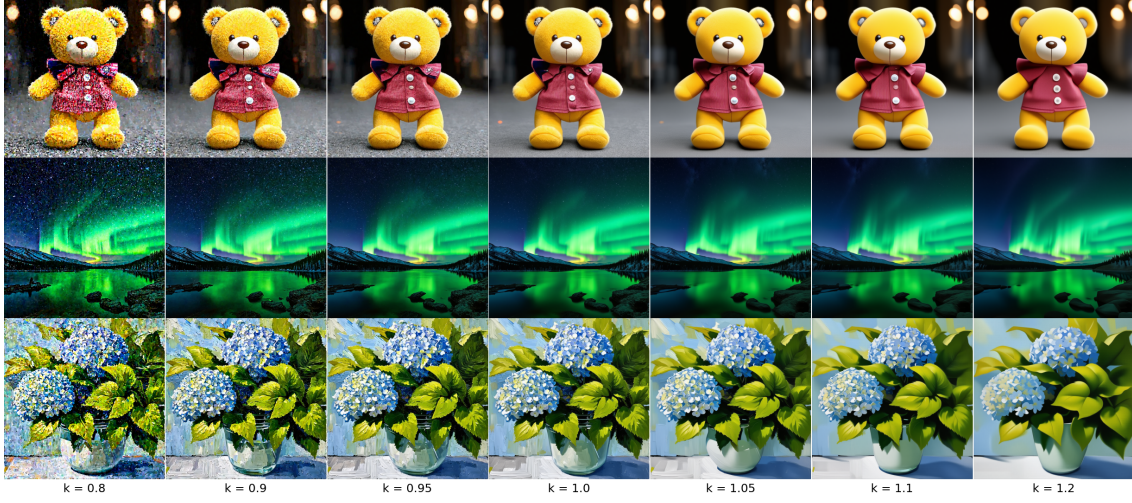


Figure 5.2: **Qualitative Examples for Varying k .** TSR allows for tuning the generated outputs to be more diverse and detailed (lower k) or more smooth and likely (higher k). While neither extreme is desirable, we notice a k slightly smaller than 1 gives pleasing images with enhanced details.

Sampling with Temporal Score Rescaling TSR (k, σ)

Given a pre-trained score model $\mathbf{s}_\theta$, TSR sampling substitutes its score prediction with:

$$\begin{aligned} \tilde{\mathbf{s}}_\theta(\mathbf{x}, t) &= r_t(k, \sigma) \mathbf{s}_\theta(\mathbf{x}, t), \\ r_t(k, \sigma) &:= \frac{\eta_t \sigma^2 + 1}{\eta_t \frac{\sigma^2}{k} + 1} \end{aligned} \tag{5.5}$$

where k, σ are user-defined parameters, and η_t is the signal-to-noise ratio of the forward process.

This makes TSR a plug-and-play method compatible with any parameterization of $\mathbf{s}_\theta$ and sampling algorithm, since conversions between score and model predictions are always linear and invertible.

Denoising Diffusion. These models are typically instantiated via neural networks ϵ_θ that learn to predict the noise added. We can infer the predicted score from this noise via a simple linear relation $\mathbf{s}_\theta(\mathbf{x}, t) = -\sigma_t^{-1} \epsilon_\theta(\mathbf{x}, t)$. We can thus perform TSR sampling in denoising diffusion models by simply using a rescaled noise prediction $\tilde{\epsilon}_\theta(\mathbf{x}, t)$ in any diffusion sampler (*e.g.* DDPM, DDIM):

$$\tilde{\epsilon}_\theta(\mathbf{x}, t) = r_t(k, \sigma) \epsilon_\theta(\mathbf{x}, t) \tag{5.6}$$

Flow Matching. For flow matching models predicting the probability flow velocity $\mathbf{v}_\theta(\mathbf{x}, t)$, the corresponding score function can be computed by [71]:

$$\mathbf{s}_\theta(\mathbf{x}, t) = -\frac{\alpha_t \mathbf{v}_\theta(\mathbf{x}, t) - \dot{\alpha}_t \mathbf{x}}{\sigma_t(\dot{\alpha}_t \sigma_t - \alpha_t \dot{\sigma}_t)} \quad (5.7)$$

Combining Equation 5.5 and Equation 5.7, we can derive the corresponding flow velocity $\tilde{\mathbf{v}}_\theta$ for the scaled distribution, such that $\tilde{\mathbf{s}}_\theta$ is a proper scaled version of the original score:

$$\tilde{\mathbf{v}}_\theta(\mathbf{x}, t) = \alpha_t^{-1}(r_t(k, \sigma)(\alpha_t \mathbf{v}_\theta(\mathbf{x}, t) - \dot{\alpha}_t \mathbf{x}) + \dot{\alpha}_t \mathbf{x}) \quad (5.8)$$

Applying this scaled velocity $\tilde{\mathbf{v}}_\theta$ in the flow samplers yields desired samples from the scaled distribution. Similar conversion can also be derived for other parameterizations of diffusion models like $\mathbf{x}_0$ -prediction and $\mathbf{v}$ -prediction.

5.2 Analysis

To understand the behavior of TSR, we first empirically validate it on toy data and show it is more effective in scaling the variance of samples while preserving each local mode compared to existing approaches. Then, we analyze how the input parameters (k, σ) control TSR and interpret their meanings in general settings.

5.2.1 Validation on Toy Distributions

Mixture of 1D Gaussians. We begin with a simple conditional generation task using a uniform mixture of 1D isotropic Gaussians in Figure 5.3, where the left three and right three modes correspond to two different classes. We apply classifier-free guidance (CFG) with guidance scale 10, constant noise scaling (CNS) and TSR with $k = 10$ individually to scale the conditional distribution and evaluate whether each method preserves all modes under scaling. As shown in Figure 5.3, CFG produces imbalanced samples, often favoring outer modes, while CNS shifts mass toward central modes at the expense of others. By contrast, TSR samples evenly across all modes while reducing intra-mode variance, demonstrating that it preserves the multimodal structure even under conditioning.

General 2D Distributions. We also apply TSR to unconditional generation on two complex 2D distributions: checkerboard and swiss roll. We train a small-scale diffusion model for each distribution and compare the scaled distribution sampled

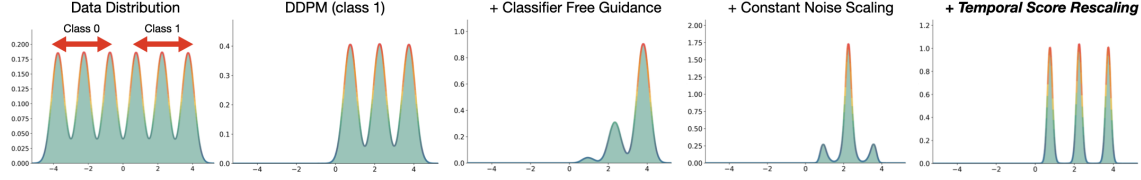


Figure 5.3: **Conditional generation on a mixture of 1D Gaussians.** A uniform mixture of 1D isotropic Gaussians, where the left three and right three modes form two classes, is scaled with CFG, CNS, and TSR. CFG produces imbalanced samples that favor outer modes and CNS shifts mass toward central modes, whereas TSR samples evenly across all modes while reducing intra-mode variance.

by CNS and TSR in Figure 5.4. We observe that CNS consistently biases samples toward the central modes, resulting in mode collapse and poor coverage of peripheral regions. This supports the intuition that reducing noise too aggressively restricts exploration during the sampling process. In contrast, TSR maintains coverage of the global distribution while reducing local variance around each mode, producing samples aligning with the true distribution. These results show that, although derived for isotropic Gaussian data, TSR generalizes to more complex scenarios and provides consistent improvements in both conditional and unconditional generation.

5.2.2 Interpreting Rescaling Hyperparameters

In the derivation of TSR, k referred to the factor of variance reduction and σ referred to the variance of the modes in the data distribution. However, in real-world scenarios with more complex distributions, the variance of the data distribution is unknown. We provide an intuitive explanation of the role of k and σ on the rescaling factor r_t to democratize the practical use of TSR in various scenarios. Specifically, we show how the rescaling factor r_t changes over sampling time with different k and σ values in Figure 5.4. Intuitively, k indicates the max/min of the rescaling factor r_t . As $t \rightarrow 0$, signal-to-noise ratio $\eta_t \rightarrow \infty$, and $r_t \rightarrow k$. Meanwhile, σ indicates how early we want to steer the sampling process. The larger σ , the earlier the sampling is steered. A very small σ lets us use the original diffusion sampling ($r_t \approx 1.0$) and only steer the last few denoising steps.

5.3 Applications

We demonstrate the broad applicability and effectiveness of TSR by applying it to a diverse set of real-world applications, spanning image generation (Section 5.3.1),

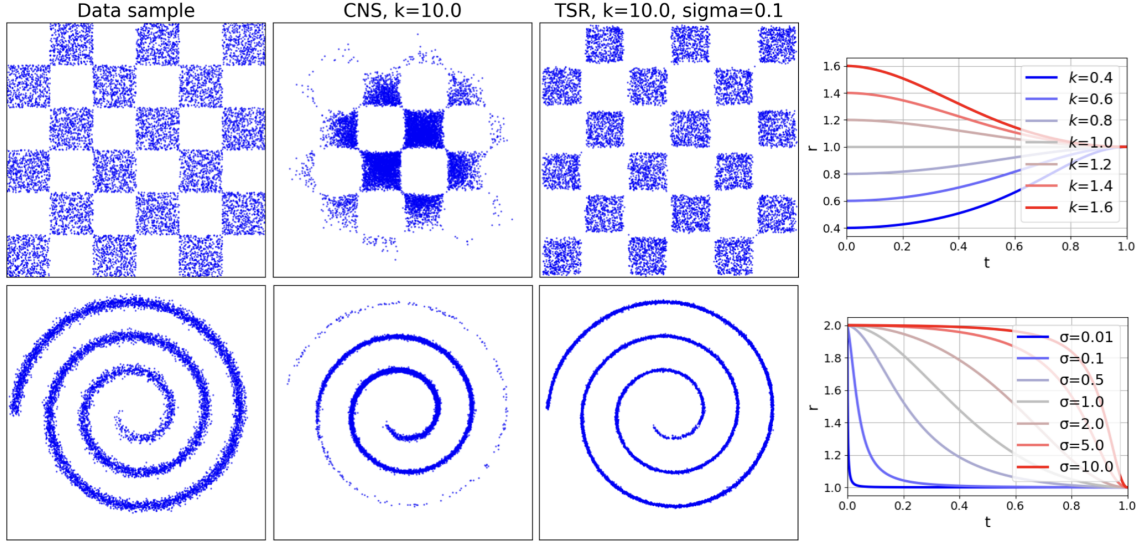


Figure 5.4: **Left: Comparison on 2D Checkerboard and Swiss Roll Distributions.** We compare samples from CNS and TSR. While CNS biases sampling towards the central modes and drops peripheral ones, TSR preserves all modes while reducing variance without generating divergent samples. **Right: Effect of Hyperparameters on the Rescaling Factor.** In the rightmost column, we plot the TSR rescaling factor r_t on the y-axis against diffusion time t . With $\sigma = 1.0$, varying k controls the asymptotic value of r_t (top); with $k = 2.0$, varying σ determines how early rescaling takes effect during sampling (bottom).

protein design (Section 5.3.2), depth estimation (Section 5.3.3), pose prediction (Section 5.3.4), and robot manipulation (Section 5.3.5). For image generation, we find that a smaller k enhances details and improves performance, while for other tasks, a larger k yields higher accuracy of model predictions. To ease the parameter selection process, we include a general guideline in Section 5.3.6.

5.3.1 Text-to-Image Generation

We examine the effect of steering the sampling distribution for diversity versus likelihood with TSR on Stable Diffusion 3 [30], a leading flow matching text-to-image model. As a creative task, image generation benefits from sampling a flatter distribution, which helps to recover more pleasing images with more high frequency details. We evaluate FID [39, 86] and CLIP [90] scores against a 5k image subset from LAION Aesthetics [97] across different CFG guidance scale w_{cfg} , TSR parameter k and σ . We fix the number of sampling steps to 30. In Figure 5.5, we see adjusting w_{cfg} makes a trade-off between text-alignment and image fidelity—higher w_{cfg} increases CLIP score at the cost of worse FID. Meanwhile, TSR allows for additional improvement

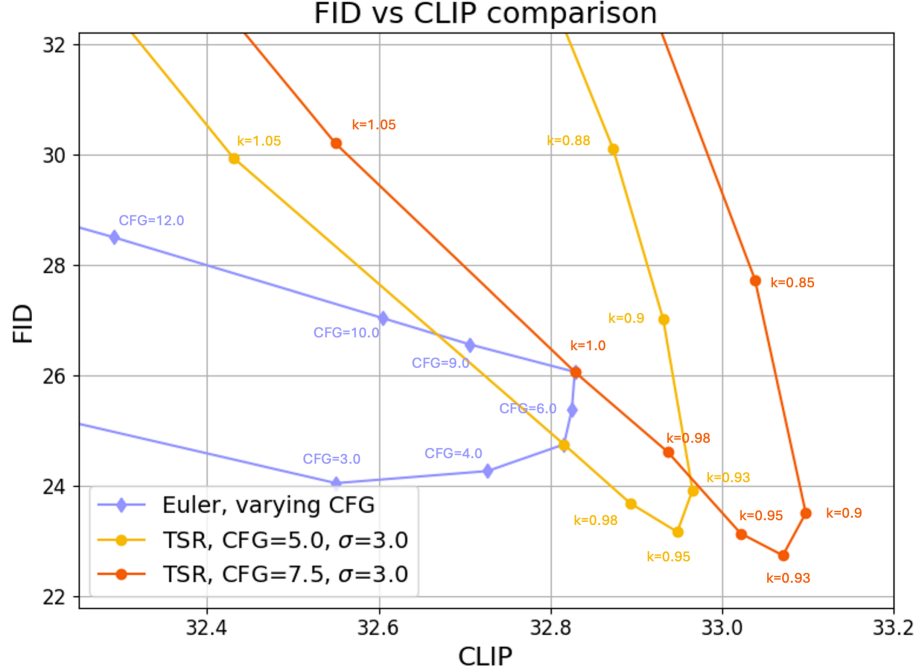


Figure 5.5: **Image Generation.** TSR achieves better text-alignment (CLIP) and image fidelity (FID), improving upon the Pareto frontier of CFG, which trades off between FID and CLIP.

beyond the Pareto frontier of CFG. Compared to the regular Euler ODE sampling, TSR reduces *FID score* from $24.77 (\pm 0.10)$ to $22.81 (\pm 0.13)$ and increases *CLIP score* from $32.82 (\pm 0.014)$ to $33.05 (\pm 0.018)$. These results are averaged over 5 random seeds. TSR achieves the optimal performance with $k = 0.93, \sigma = 3.0$. To verify the transferability of these parameters, we apply the same (k, σ) to Flux.1 dev [58] and Stable Diffusion 2 [92] and report the results in Table 5.1. The optimal (k, σ) found on SD3 consistently improve performance on other models as well, suggesting the robustness of the choice for (k, σ) across models.

Notably, while it is possible to perform stochastic sampling with flow models like SD3, we found that it performs significantly worse than ODE sampling with the same compute budget (see Section B.1.1), making CNS impractical. We also show in Section B.1.1 that TSR achieves superior performance with denoising diffusion model (SD2, 92) compared to CNS and other common samplers. Qualitatively, we observe in Figure 5.2 that lower k leads to images with more high-frequency detail (in the extreme case more noise), and higher k leads to smoother images. We infer that using a smaller k flattens the modeled distribution and allows better coverage of the desirable image space. Overall, our results highlight that the control over the

Table 5.1: **Evaluation of Text-to-Image Generation across Models.** TSR consistently improves image quality across Stable Diffusion 3 [30], Stable Diffusion 2 [92], and Flux.1 dev [58]. The optimal (k, σ) found on SD3 generalize effectively to other models. For SD3 and Flux.1 dev, the default scheduler is Euler-ODE. For SD2 the default scheduler is DDPM.

	SD3		SD2		Flux.1 dev	
	FID ↓	CLIP ↑	FID ↓	CLIP ↑	FID ↓	CLIP ↑
Default Scheduler	24.77	32.82	22.81	33.66	53.99	31.97
+ TSR	22.81	33.05	19.75	33.75	51.79	32.14

likelihood-diversity trade-off enabled by TSR is beneficial in image generation.

5.3.2 Protein Generation

Generative models have emerged as a powerful paradigm in AI for Science. For example, protein discovery [1, 50, 113] is an application where such models have seen widespread adoption. However, not all generated proteins are valid in the real world. Thus, improving the designability of generated proteins is an important goal. CNS has previously been used to enhance sampling quality [34, 120].

We conduct experiments with FoldingDiff [113], a diffusion-based protein generation method, and compare TSR with CNS. Evaluation uses two complementary metrics: designability score [113], measuring structural quality and real-world feasibility, and protein FID [31], capturing distributional similarity and thus diversity. Ideally, a method should achieve a high designability score and a low FID. As shown in Figure 5.6, samples from TSR lie in the bottom right regions, which shows TSR maintains protein diversity better than CNS, while improving the designability.

5.3.3 Depth Estimation

The task of monocular depth estimation is inherently challenging due to its uncertainty—an object may appear large but distant, or small but close. Recent methods [29, 54, 95] address this with diffusion models, where different samples correspond to plausible variations or interpolations of the underlying depth structure. We adopt Marigold [54], which fine-tunes a pre-trained text-to-image diffusion model for depth estimation and achieves strong results. However, individual samples can be suboptimal due to both the sampling stochasticity and the ambiguity of depth estimation [54]. To mitigate

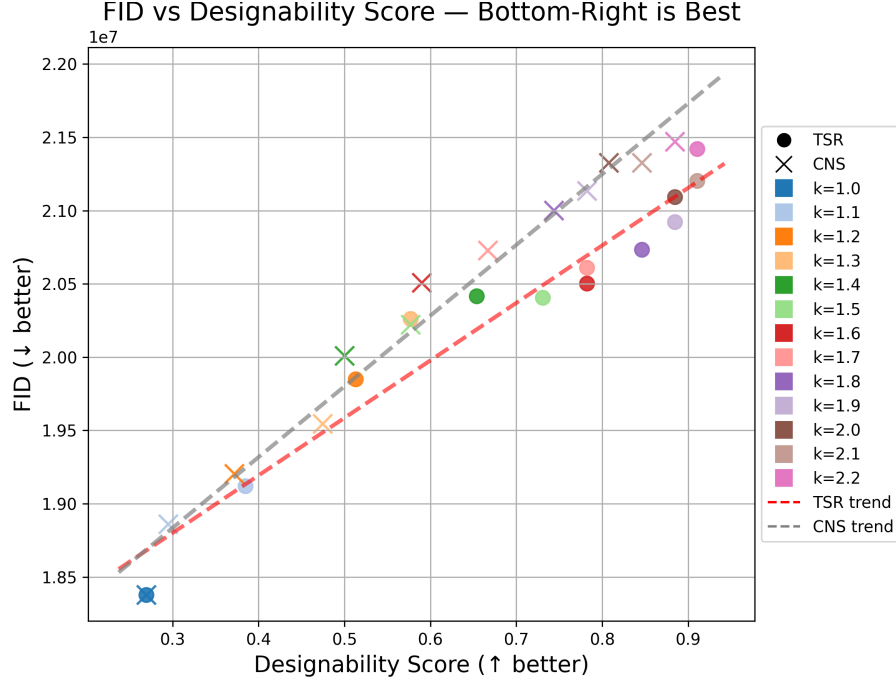


Figure 5.6: **Protein Generation.** TSR improves the designability score while preserving the diversity (FID) better compared to CNS. The original sampling has a designability score of 0.22.

this issue, it is desirable to increase the likelihood of each sampled estimate—i.e., to encourage samples to concentrate around the dominant modes of the learned distribution. Doing so reduces sampling variability and suppresses uncertain or noisy depth predictions.

We evaluate on the ETH3D [96] and NYUv2 [79] datasets. As shown in Table 5.2, TSR outperforms the default DDIM and CNS on prediction accuracy. By sampling from a sharper distribution, TSR yields more probable outputs given the input image. Qualitative comparisons in Figure 5.7 further show that TSR produces cleaner depth maps than DDIM, particularly in high-uncertainty regions.

5.3.4 Pose Prediction

Previous work [44, 59, 112, 126] has shown that diffusion models can effectively predict object and camera poses in the $SO(3)$ space. We demonstrate TSR can improve such models’ accuracy by sampling from a sharper distribution. Our evaluations are based on the $SO(3)$ diffusion models proposed by Hsiao et al. [44], where we apply TSR and evaluate on the SYMSOL dataset [77], which contains geometric shapes with a high order of symmetries. We visualize the effect of TSR in Figure 5.8

Table 5.2: **Quantitative Evaluation of Depth Estimation.** TSR improves depth estimation and outperforms the naive baseline.

	ETH3D		NYUv2	
	AbsRel ↓	δ_1 ↑	AbsRel ↓	δ_1 ↑
DDIM	7.1	90.4	6.0	95.9
+ CNS	6.82	95.6	5.85	96.0
+ TSR	6.68	95.7	5.84	96.0

Table 5.3: **Pose Prediction.** Mean error (deg) and accuracy within thresholds 0.2, 0.5, 1. $(k, \sigma) = (7.0, 0.5)$ for TSR, $k = 1600$ for CNS.

	Error (deg) ↓	Acc@ (deg) ↑		
		0.2	0.5	1.0
Score Sampling	0.444	9.4	68.3	97.9
+ CNS (1600)	0.350	20.0	84.9	99.1
+ TSR (7.0, 0.5)	0.356	18.5	84.0	99.0

where we show the sampled poses on an example image from SYMSOL. TSR samples poses more concentrated around ground truth modes (the circle centers) than the baseline score matching sampling used in Hsiao et al. [44]. In quantitative evaluation (Table 5.3), TSR predictions have lower average error and higher accuracy under a range of accuracy thresholds compared to score matching sampling, highlighting the benefits of predicting samples close to modes. We find that CNS also reduces pose error, achieving a performance slightly better than TSR on SYMSOL. However, we note that TSR remains robust and applicable over many tasks and sampling methods where constant noise scaling is not possible.

5.3.5 Robotic Manipulation

Lastly, we examine the applicability of TSR on predicting robot actions, with a focus on robotic manipulation. One notable difference of this domain compared to others is that the policy only models a distribution of actions for a short horizon, as it is a sequential decision-making problem.

We chose Pi-0 [11], a generalist robotic flow-matching policy released by Physical

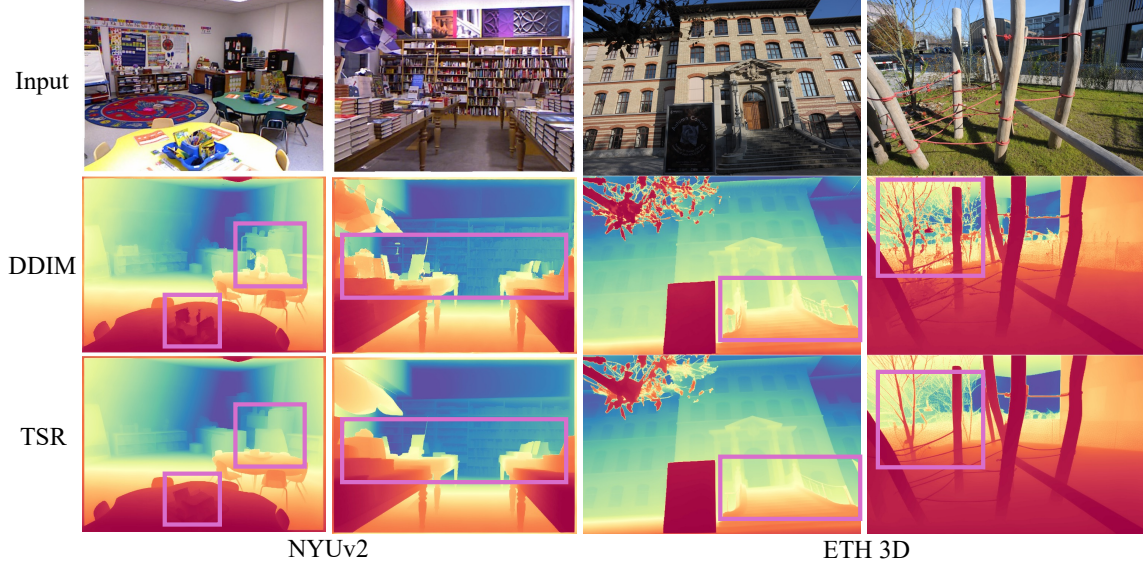


Figure 5.7: **Qualitative Depth Estimation Comparisons.** Compared to DDIM, TSR with $k > 1$ predicts cleaner depth in the regions with high uncertainty (highlighted by pink boxes).

Table 5.4: **Results for Robotic Manipulation.** Success rates are computed across 3 seeds, 10 tasks, with 50 rollouts per task. The results are computed with the best (k, σ) for TSR.

Model / Task ID	0	1	2	3	4	5	6	7	8	9	Average
Pi-0	86.0	97.3	80.7	96.0	84.7	93.3	81.3	94.7	23.3	80.0	81.7
+ TSR (1.25,0.25)	86.7	97.3	77.3	97.3	84.7	96.0	82.7	96.0	21.3	88.7	82.8

Intelligence, finetuned on LIBERO [64], a simulation benchmark for robotic manipulation. Specifically, we evaluate the policy over 10 tasks in the LIBERO-10 benchmark with shared (k, σ) values. The results are in Table 5.4. Without any further training, TSR improves the performance of 6 tasks and maintains performance for 2 tasks. One notable point is that the two tasks (Task ID 2 and 8) where TSR shows worse performance are precisely those in which the base Pi-0 policy itself exhibits low success rates. This suggests that the suboptimal performance may be due to a common ‘sharpening’ ($k > 1$) hyper-parameter across tasks as this may be suboptimal when the policy is not correct, and that tuning TSR’s k for each task may yield further gains.

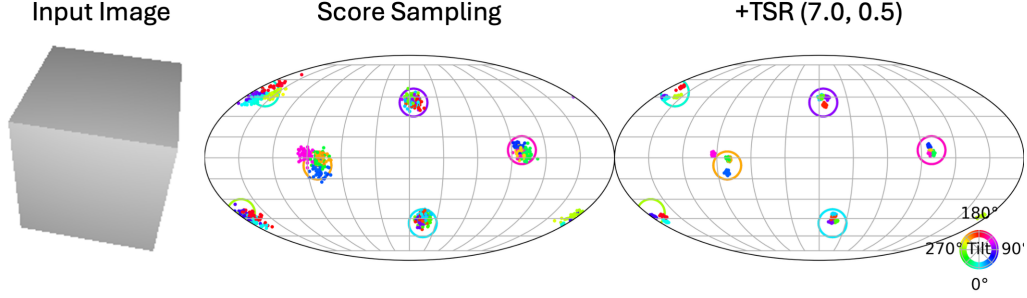


Figure 5.8: **Predicted poses on SYMSOL.** TSR reduces pose prediction error: each dot marks a sample’s first canonical axis (colored by rotation), while circles denote ground-truth poses.

5.3.6 Parameter Selection Guidelines

Similar to CFG [41], the hyperparameters σ and ϵ in TSR only need to be tuned once per model. Moreover, as shown in Table 5.1, the same set of hyperparameters can generalize across different models within the same task. To further simplify hyperparameter selection, we provide the following practical guidelines.

As shown in the appendix across multiple tasks (Figure B.1, Figure B.4, and Figure B.6), the effect of σ and ϵ on generation quality follows a predictable parabola-like trend when varying one parameter while fixing the other. Based on this observation, we first fix σ (*e.g.* $\sigma = 1.0$) and identify the optimal ϵ^* via binary search. We then fix ϵ^* and tune σ in the same manner, optionally repeating the process once more if needed. This procedure is usually substantially more efficient than exhaustive grid search.

5.4 Limitations

We presented TSR, an approach to alter the sampling distribution for pre-trained diffusion and flow models. While we demonstrated its efficacy across several (toy and real) tasks, there are fundamental limitations worth highlighting. First, unlike temperature scaling, TSR can only alter the ‘local’ sampling and there might exist applications where a global temperature scaling is more desirable *e.g.* TSR does not change the weights of the components in a Gaussian mixture, only the variance. Moreover, while TSR does empirically steer the sampling diversity in generic scenarios, the theoretical guarantees are limited to simpler settings and one may be able to derive a better algorithm for different distributions. Nevertheless, as TSR can be

readily applied to any off-the-shelf denoising diffusion and flow matching model, we believe it would be a generally useful technique for the community to explore. In particular, the alternative strategy of ‘constant noise scaling’ is already adopted across applications [34, 120], and our work offers an alternative that is empirically better and more widely applicable (*e.g.* in deterministic sampling).

5.5 Summary

This chapter introduced TSR, a training-free mechanism that steers the sampling diversity of any pre-trained diffusion or flow matching model by rescaling its predicted score with a single time-dependent factor $r_t(k, \sigma)$. We showed that this factor arises analytically as the ratio between the scores of the original and a locally variance-scaled data distribution for isotropic Gaussians, proved that the approximation remains valid for well-separated mixtures of Gaussians, and interpreted the effect as a ‘local’ temperature that reshapes samples around each mode without disturbing the global mode structure – distinguishing it from constant noise scaling, which we showed cannot realize true temperature scaling. Because the score-to-prediction conversions are linear and invertible, TSR drops into existing deterministic and stochastic samplers with zero inference overhead. Empirically, a single knob delivered consistent gains across five disparate domains – image generation, protein design, depth estimation, pose prediction, and robot manipulation – with a smaller k helping creative image synthesis and a larger k sharpening perception and control outputs.

Chapter 6

Reactive Real-Time Flow Policies

This chapter presents $\pi\mathbf{R}^2$, which closes the *architecture* gap identified in Chapter 1.¹ The preceding chapters adapted a frozen generalist policy to a new task from a single human demonstration (Chapter 4) and steered its sampling distribution (Chapter 5), both leaving the policy’s speed untouched; this chapter instead makes such large foundation-model policies fast enough to *react*.

We have witnessed remarkable progress in training ‘robotics foundation models’ [10, 11, 14, 45, 57, 133], and three design choices have become ubiquitous across recent efforts. First, **large pre-trained backbones** such as vision-language models (VLMs) [2, 6, 7, 24, 65, 108] are leveraged to process visual (and language) input and extract rich, generalizable representations for downstream action prediction. Second, **expressive policy architectures** such as diffusion [19] and flow matching [81, 121] allow effectively capturing the multi-modal action distributions inherent in diverse human demonstrations. Finally, **action chunking** [128, 130] enables these models to jointly predict multiple actions, providing a richer training signal and improving the temporal consistency of executed actions.

On the one hand, these choices have been crucial in unlocking generalizable and scalable imitation learning. However, they result in *limited reactivity* and *increased latency*, making these foundation models ill-suited for dynamic manipulation tasks that demand fast, smooth, and reactive control. Specifically, the execution of a ‘chunk’ of predicted actions is performed open-loop, without the ability to react to the sensory input streaming in. While the reactivity can in theory be improved by only executing a small ‘sub-chunk’ (in limit just one action) and re-predicting with updated sensory input, naively doing so is infeasible due to the significant computation required in the ‘perception-to-action’ prediction, a large visual (and language) processing backbone

¹ $\pi\mathbf{R}^2$ is under review for CoRL 2026.

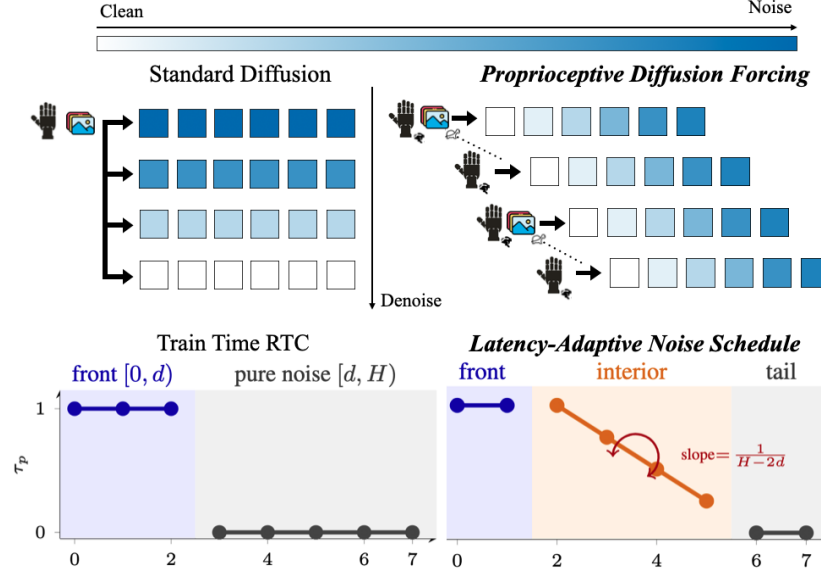


Figure 6.1: Overview of $\pi\mathbf{R}^2$. **Top.** While standard diffusion/flow matching relies on stale observations to predict actions via iterative denoising, $\pi\mathbf{R}^2$ disentangles the observation into a fast channel (proprioception) and a slow channel (image encoding, VLM embedding, etc.), and uses up-to-date observations for each denoising step. **Bottom.** To incorporate latency and smooth execution, we adopt an adaptive noise schedule, where the action chunk is divided into three regions: clean actions to be taken during inference, actions with increasing noise level following Diffusion-Forcing, and pure noise with the same length as clean actions. Compared to Train-Time RTC, this enables faster inference and smoother actions.

followed by multiple denoising iterations for flow matching policy inference. This computational bottleneck coupled with action chunking flow matching thus results in two (related) fundamental limitations: a) reduced reactivity, as a large (sub-)chunk of actions must be executed open-loop while the next chunk is predicted, and b) increased latency, as the predicted actions are a function of ‘stale’ sensory input.

$\pi\mathbf{R}^2$ overcomes these limitations, enabling reactive real-time policy inference *without* compromising on the design principles of large pre-trained backbones, expressive multi-modal policies, and multi-action prediction. We note that instead of flow (or diffusion) policy inference, which maps uniformly random noise to action chunks over multiple denoising iterations, leveraging the flexible schedule in ‘diffusion forcing’ [16] offers several benefits: a) the immediate actions to be executed can be at a lower noise level and predicted in fewer iterations, and b) different denoising iterations involved in predicting an action can rely on updated sensory input as available. Although prior work has explored diffusion forcing for policy learning [43], we address the key issues that prevent its widespread adoption for robotics foundation models.

First, while the (immediate) action denoising requires less computation, the backbone (*e.g.* VLM) feature processing becomes the bottleneck for ‘perception-to-action prediction’ latency. Our key insight that allows circumventing this bottleneck is that not all input modalities need to be processed at the same rate: proprioceptive signals (joint positions, velocities, torques, and contact forces) can be retrieved and processed orders of magnitude faster than images or text. Moreover, for dynamic tasks, proprioception carries sufficient information for *local* reactive corrections (*e.g.*, responding to an unexpected contact or compensating for object slip), while vision and language provide the *global* context needed to guide coarse motion plans. We thus disentangle the conditioning for action denoising into asynchronously updated ‘slow’ and always up-to-date ‘fast’ features, effectively allowing high-frequency proprioceptive control within an action chunk. Second, despite our asynchronous diffusion forcing, practical constraints (communication delays, GPU bandwidth) often prevent zero latency. We develop a delay-adaptive noise schedule for diffusion forcing that enables seamless real-time execution despite latency, allowing the model to continue executing the ‘previous’ actions while ensuring smooth temporally-coherent outputs from subsequent denoising iterations.

Together, these design choices allow $\pi\mathbf{R}^2$ to achieve closed-loop *replanning* at 25 Hz using a large-scale VLA (GR00T N1.7 [10]) on A5000 GPUs, roughly $\sim 4\times$ faster than the original model’s replanning rate. Crucially, this means the policy predicts each action using a fresh observation every 40 ms, rather than committing to a long action chunk and replanning only sparsely (*e.g.* at ~ 7 Hz). $\pi\mathbf{R}^2$ thus matches the reactivity of simple feedforward policies while retaining the expressivity of flow matching, the training benefits of chunk-based inference, and the semantic grounding of pretrained backbones. We first validate $\pi\mathbf{R}^2$ in simulation where we highlight the benefits of increased reactivity and reduced latency compared to the typical action chunking flow policies. We also show how, instead of training from scratch, we can finetune pre-trained VLAs with a diffusion forcing schedule and demonstrate the efficacy of $\pi\mathbf{R}^2$ for complex real-world manipulation tasks, where we find that it improves over the baselines by up to 11% in simulation and 20% in the real world over the strongest baseline, corresponding to relative gains of 52.3% and 50%, respectively.

6.1 Method

Our goal is to modify existing VLA architectures for higher reactivity and lower inference latency. We first review the building blocks – flow matching and diffusion

forcing – and then introduce two modifications that together yield a real-time, closed-loop flow policy compatible with any existing VLAs.

6.1.1 Preliminaries

Flow Matching and Action Chunking Policy

We build on flow matching [62, 124] with action chunking [128, 130]. Flow matching learns a velocity field $v_\theta(\mathbf{x}_t, t)$ that transports samples from noise $p_0 = \mathcal{N}(\mathbf{0}, \mathbf{I})$ to data $p_1 = p_{\text{data}}$ along the conditional interpolation $\mathbf{x}_t = (1-t)\boldsymbol{\epsilon} + t\mathbf{x}_1$, $t \in [0, 1]$, trained against the conditional velocity $u_t(\mathbf{x}_t \mid \mathbf{x}_1) = \mathbf{x}_1 - \boldsymbol{\epsilon}$:

$$\mathcal{L}_{\text{FM}} = \mathbb{E}_{t, \mathbf{x}_1, \boldsymbol{\epsilon}} \left[\|v_\theta(\mathbf{x}_t, t) - (\mathbf{x}_1 - \boldsymbol{\epsilon})\|^2 \right]. \quad (6.1)$$

In the action-chunking policy setting, $\mathbf{x}_1 = (\mathbf{a}_t, \dots, \mathbf{a}_{t+H})$ is a chunk of H future actions conditioned on observation $\mathbf{o}_t$, generated jointly by K denoising steps; the robot executes the first $h \leq H$ actions before re-planning. **Crucially, all H positions share a single noise level t and the chunk is committed open-loop: the h executed actions never account for new observations during their execution window**, limiting the reactivity that manipulation tasks demand.

Diffusion Forcing and Streaming Diffusion

Diffusion forcing [16, 103] generalizes flow matching by assigning each chunk position $p \in \{0, \dots, H-1\}$ an independent noise level $\tau_p \in [0, 1]$:

$$\mathbf{x}_{\tau, p} = (1-\tau_p)\boldsymbol{\epsilon}_p + \tau_p \mathbf{a}_p, \quad (6.2)$$

and the model $v_\theta(\mathbf{x}_\tau, \boldsymbol{\tau}, \mathbf{o})$ predicts a per-position velocity. The flexibility of position-dependent noise levels enables *closed-loop* action prediction by conditioning successive denoising steps on progressively more recent observations within a single chunk. *Streaming diffusion* [43] is one such instantiation: it imposes a linearly increasing noise schedule across the chunk so that the leading positions are fully denoised within a few steps, and each denoising step incorporates a fresh observation.

In this work, we also adopt diffusion forcing, focusing on an underexplored regime: enabling large-scale VLAs to be more reactive and closed-loop, where large latency naturally arises from its model size and components. The following two subsections present the architectural and scheduling modifications that make diffusion forcing a

practical real-time policy on top of any VLA action head.

6.1.2 Proprioception-Reactive Diffusion Forcing

A VLA processes the observation $\mathbf{o}_t = (\mathbf{s}_t, \mathbf{I}_t, \mathbf{T}_t)$ – proprioception $\mathbf{s}_t$ (joint positions, end-effector pose), image $\mathbf{I}_t$, and language $\mathbf{T}_t$ – to predict an action chunk $(\mathbf{a}_t, \dots, \mathbf{a}_{t+H})$. Internally, the inputs are preprocessed (dominated by image processing) and passed through the VLM backbone, producing language-aligned features; proprioception is processed by a small MLP into a state embedding; the DiT action head then conditions on the concatenated representation and runs K denoising steps over the chunk. In practice, these components differ greatly in cost. On RTX A6000 with GR00T-N1.7, image preprocessing + VLM (~ 60 ms) plus DiT denoising ($K=4$ steps, ~ 80 ms) sum to ~ 140 ms per call – ~ 7 control ticks at 50 Hz. This motivates disentangling the slow VLM path from the fast action prediction loop.

We propose to disentangle the DiT’s conditioning (Figure 6.1, left) into a *slow channel* (vision and language features from the VLM and image/text encoders) and a *fast channel* (proprioception, fresh every tick). This split mirrors the structure of manipulation itself: vision and language provide coarse spatial and task guidance, while fresh proprioception drives the fine motor refinement that precision demands. Diffusion forcing additionally reduces the per-call denoising work: because the front of the chunk is kept near $\tau = 1$, a single denoising step per call suffices to produce clean actions at the front (one or more, depending on the schedule; see Section 6.1.3), versus K steps in standard chunked denoising. Combining the two, the DiT action head runs one denoising step against fresh proprioception and a *cached* slow feature; vision and language are processed asynchronously in a background thread (image preprocessing + VLM forward) and refresh the cache when complete, so the action head conditions on a slow feature of age d_{vlm} ticks (the vision/text wall-clock latency). Per-call cost of the action head is therefore one NFE only (~ 20 ms), independent of backbone size.

To absorb the resulting slow-channel staleness, we delay the slow channel at training time by $d_{\text{vlm}} \sim \text{Uniform}\{0, \dots, d_{\text{vlm}}^{\max}\}$ and supply the delay value through a learned scalar embedding added to the slow representation; at deploy time the measured d_{vlm} uses the same embedding. The resulting policy is *proprioception-reactive*: it reacts to fresh joint state every denoising step while tolerating bounded staleness on vision and language, recovering closed-loop control even behind a large VLA backbone. We refer to this architecture variant as *asynchronous vision/text processing*.

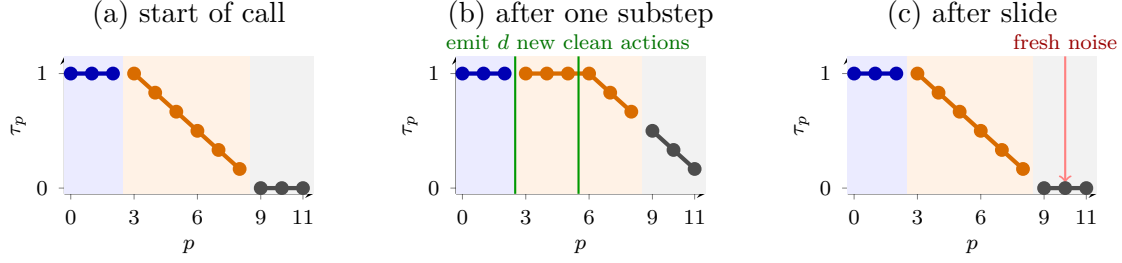


Figure 6.2: **Inference cycle, one call.** (a) Buffer at $\tau^{*,d}$ at the start of the call. (b) One Euler substep (one NFE) shifts the schedule right by d slots: positions $[d, 2d)$ reach $\tau=1$ and are released, and the ramp extends through the back. (c) The buffer slides d positions: just-emitted actions become the new front conditioning, d fresh-noise slots ($\tau=0$) are appended – the schedule reproduces exactly.

6.1.3 Latency-Adaptive Flow Schedule

By combining diffusion forcing with asynchronous vision/text processing and single-step DiT inference, the per-call delay d of the action prediction loop reduces to one denoising step of the action head. However, in reality, d varies: it depends on each component size and inference GPU, includes network latency when policy and robot run on separate machines (e.g., Ethernet round-trips in remote-inference setups), and fluctuates with computational load and per-call jitter. The policy must therefore handle a range of inference delays $d \in [1, d_{\max}]$.

A naive linearly increasing noise schedule, as in standard streaming diffusion [43], does not address this regime: it implicitly assumes $d = 0$ (instantaneous inference, no in-flight actions), so when applied at $d > 0$ the returning chunk can jump at the chunk boundary [13, 106]. We address both issues with a per-position noise schedule parameterized by the inference delay d : a clamped-clean front carries the d in-flight actions as inpaint conditioning, a ramped interior emits d clean actions per call at the cost of one denoising step, and randomizing d at training lets a single model adapt to whatever value is measured at each call.

Staircase schedule (Figure 6.1, right). For a target inference delay d , $\pi\mathbf{R}^2$ uses the three-region staircase $\tau^{*,d} \in [0, 1]^H$

$$\tau_p^{*,d} = \begin{cases} 1 & 0 \leq p < d \\ 1 - \frac{p-d}{H-2d} & d \leq p < H-d \\ 0 & H-d \leq p \leq H-1 \end{cases} \quad (6.3)$$

with interior slope $s = 1/(H-2d)$. Three roles by position: **front** $[0, d)$, the in-flight actions, clamped clean as inpaint conditioning; **interior** $[d, H-d)$, a linear ramp from clean to noise; **tail** $[H-d, H)$, the d pure-noise slots appended at the back per cycle. This staircase can be viewed as a diffusion-forcing generalization of training-time RTC [12]: both clamp a clean front of d in-flight actions for inpaint conditioning, but RTC applies a single shared noise level over the remaining $H-d$ positions, whereas the staircase replaces it with a ramped interior plus a pure-noise tail to enable single-step emission under diffusion forcing.

Training. At each batch we sample $d \sim \text{Uniform}\{1, \dots, d_{\max}\}$ and build $\tau^{*,d}$. The front d slots are filled by the ground-truth actions and excluded from the loss via $m_p = \mathbf{1}[p \geq d]$ (mirroring inference-time inpaint conditioning); the interior and tail are noised at the staircase levels and contribute to the per-position MSE. We additionally apply symmetric jitter $\tau_p \leftarrow \text{clip}(\tau_p + \delta_p, 0, 1)$ with $\delta_p \sim \text{Uniform}[-j, j]$ to absorb small deviations from the central staircase that arise from per-call d variation. With probability 0.2, we instead train on a standard flow schedule (single $\tau \sim \text{Uniform}[0, 1]$ shared across positions, no mask), so the same network can also denoise a full chunk from pure noise, used at inference to warm-start the buffer at episode start.

Compatibility with existing VLA action heads. The procedure above plugs into any DiT-style VLA action head with a one-line architectural change: the AdaLN conditioning (which modulates each DiT block by the noise level τ) becomes per-position, with one (γ_p, β_p) pair per chunk position rather than one shared across positions. All other components (attention, MLP, VLM backbone, slow/fast feature paths) remain unchanged, so existing pretrained VLAs can be fine-tuned with $\pi\mathbf{R}^2$ without touching the backbone.

Inference cycle (Figure 6.2). At episode start we warm up the buffer with standard-flow inference (denoising all H positions from pure noise) and re-noise to match $\tau^{*,d}$ for the initial measured d . From then on, each call applies one denoising step that updates each position by

$$\mathbf{x}_p \leftarrow \mathbf{x}_p + \Delta\tau_p \cdot v_\theta(\mathbf{x}, \boldsymbol{\tau}, \mathbf{o})_p, \quad \tau_p \leftarrow \tau_p + \Delta\tau_p, \quad (6.4)$$

with region-dependent per-position advances $\Delta\tau_p$ chosen so the schedule shifts right by d slots: positions $[d, 2d)$ reach $\tau=1$ and are emitted, and the rest of the buffer rotates forward (Figure 6.2b). The buffer then slides by d positions, with d fresh-noise

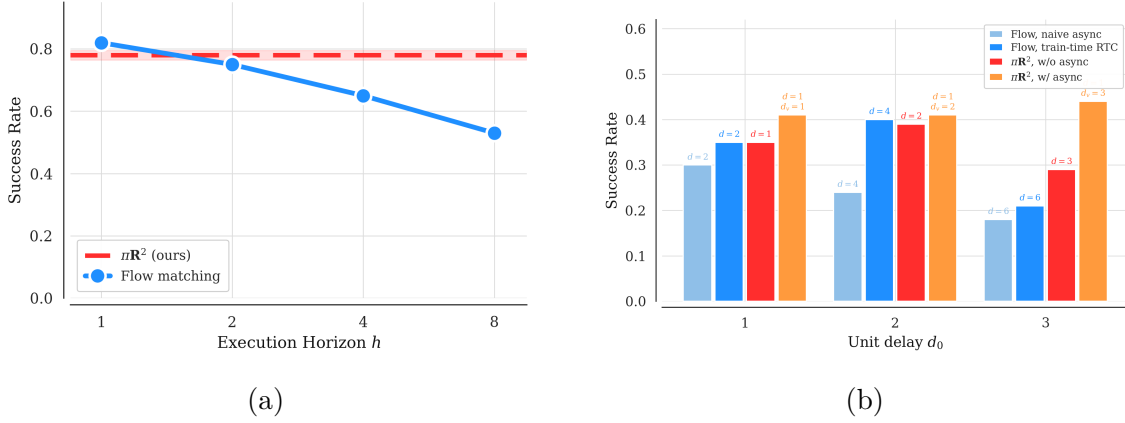


Figure 6.3: Simulation results. **Left (a).** Without any inference delay, executing a smaller number of actions within the chunk benefits the performance for a flow matching policy. $\pi\mathbf{R}^2$ also achieves the same performance via replanning every timestep, while only the last denoising step is conditioned on up-to-date state. **Right (b).** When there is an inference delay for each component, $\pi\mathbf{R}^2$ reduces the effective delay d by reducing the number of denoising steps and asynchronously processing the visual features. For each datapoint, d indicates the effective delay of proprioception, while d_v is visual delay.

slots ($\tau=0$) appended at the back (Figure 6.2c). The schedule reproduces exactly when d holds steady; when d changes between calls the $\Delta\tau_p$ pattern adapts to the new measured d , and the buffer is pulled toward the new $\tau^{*,d}$ over a few calls.

6.2 Experiments

6.2.1 Simulation Experiments

Setup. We first test $\pi\mathbf{R}^2$ on the Leap Cube Reorientation task in MuJoCo Playground [122], a common task that requires reactivity to keep the cube from falling and to achieve the target cube pose. Demonstrations are collected at 50 Hz from 4 RL experts (different seeds), yielding 200 trajectories per task. All methods are trained as state-based flow policies, with prediction chunk length $H = 16$. We run two studies: an execution-horizon h sweep under zero inference delay (Section 6.2.1), and a deployment study under a fixed latency budget (Section 6.2.1). Full implementation details are in Appendix C.

Execution-horizon (h) sweep without inference delay

We first study the setting where inference delay $d = 0$, to isolate two things: (i) the tradeoff between task performance and execution horizon h (executing more actions open-loop misses the recent state), and (ii) whether $\pi\mathbf{R}^2$'s amortized denoising, where each action's final denoising step conditions on the most recent observation, matches the reactivity of $h=1$ flow at a fraction of the per-call cost. At evaluation, standard flow matching uses 16 denoising steps to predict the whole chunk, whereas $\pi\mathbf{R}^2$ uses one denoising step and emits one action at a time. We sweep $h \in \{1, 2, 4, 8\}$ for standard flow; $\pi\mathbf{R}^2$ emits one action per call ($h = 1$, 1 NFE per call). Results in Figure 6.3 (left) confirm the h tradeoff for standard flow (performance degrades as h grows, with $h \in \{1, 2\}$ best on the reactivity-sensitive tasks), and show that $\pi\mathbf{R}^2$ at $h=1$ matches the flow configuration of $h \in \{1, 2\}$. This supports that amortizing the denoising budget across calls does not sacrifice quality, even though each call costs only one NFE. We next introduce realistic latency and test each deployment regime under that constraint.

Deployment under VLA-level inference delay

To consider a realistic latency, we take into account GR00T-N1.7 computation cost (Section 6.1.2: vision/text ≈ 60 ms, $K=4$ NFE denoising ≈ 80 ms) and define unit delay d_0 as the $K = 4$ denoising cost; vision/text $\approx 0.75 d_0$ and one NFE $= d_0/4$. The effective per-call delays are then $1.75 d_0$ (end-to-end inference), $1.0 d_0$ ($\pi\mathbf{R}^2$ without asynchronous processing), and $0.25 d_0$ ($\pi\mathbf{R}^2$ with asynchronous processing). Setting $d_0 \in \{1, 2, 3\}$ control ticks and ceil-rounding gives effective proprioception delay $d \in \{2, 4, 6\}/\{1, 2, 3\}/\{1, 1, 1\}$, respectively. For $\pi\mathbf{R}^2$ with async processing, we additionally delay the object pose by $d_{vis} \in \{1, 2, 3\}$. For all methods, we set execution horizon $h = d$ across all methods.

Results. We compare four methods: (i) *naive-async*, (ii) *train-time RTC* [12], (iii) $\pi\mathbf{R}^2$ without async processing (Section 6.1.3), and (iv) $\pi\mathbf{R}^2$ with async processing (Section 6.1.2). Figure 6.3 (right) shows the results. $\pi\mathbf{R}^2$ with async wins at every d_0 (0.41, 0.41, 0.44), beating naive-async (0.30, 0.24, 0.18) and RTC (0.35, 0.40, 0.21) with margins that widen as delay grows. The reason is effective delay: baselines pay $d = \lceil 1.75 d_0 \rceil$ per call while $\pi\mathbf{R}^2$ pays $d = d_0$ (without async) or $d = 1$ (with async). Async processing lets us hold the action delay at 1 while only the visual delay d_{vis} grows, so performance degrades gracefully with d_{vis} – supporting the view that vision

Table 6.1: **Real World Results.** (xArm6 + XHand, fine-tuned GR00T N1.7). d is the measured wall-clock inference delay in 25-Hz control ticks (1 tick $\approx$ 40 ms), which is the same as execution horizon h for all methods other than synchronous inference. For each task we report success rate (SR) over N trials and a partial-progress score (Prog) capturing the fraction of subgoals completed per episode.

Setting	d	Don't Spill		Tidy up Book		Insert Box	
		SR	Prog	SR	Prog	SR	Prog
Flow, Synchronous ($h = 10$)	4 $\sim$ 5	4/20	16/80	4/20	9/40	11/20	56/80
Flow, Naive Async, dense, TE [130]	4 $\sim$ 5	7/20	30/80	7/20	15/40	12/20	61/80
Flow, Train-Time RTC [12]	5	9/20	45/80	8/20	18/40	10/20	53/80
$\pi\mathbf{R}^2$	1–2	10/20	55/80	12/20	24/40	16/20	68/80

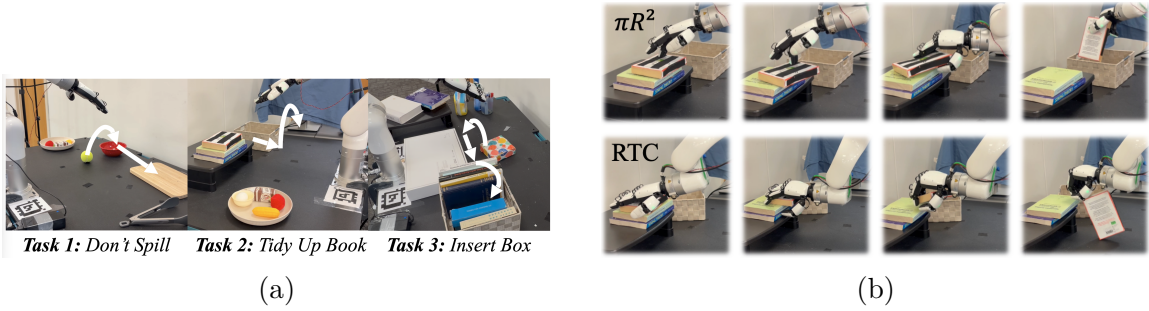


Figure 6.4: **Real-world tasks and qualitative results.** **Left (a).** Real-world manipulation tasks: *Don't Spill* (left), *Tidy Up Book* (middle), *Insert Box*. Each image shows how the task should be solved. **Right (b).** Rollout comparison on *Tidy Up Book*: $\pi\mathbf{R}^2$ tracks the contact dynamics smoothly, while Train-time RTC jitters and fails to pick up the book due to inaccuracy.

provides coarse guidance while up-to-date proprioception drives manipulation.

6.2.2 Real-World Experiments

We now mirror the deployment study above on a real xArm6 + XHand setup by fine-tuning the GR00T-N1.7 model [80] for dexterous manipulation. Demos are collected at 25 Hz (capped by the teleoperation recording pipeline), so we report d in 25-Hz ticks (1 tick $\approx$ 40 ms); We use RTX A5000 for both training and inference, resulting in $d = 4 \sim 5$ for baselines and $d = 1 \sim 2$ for $\pi\mathbf{R}^2$. We include arm and hand joint angles, along with hand per-joint torque and fingertip forces in the proprioceptive state.

Tasks. We evaluate our method on three contact-rich dexterous manipulation tasks. *Don't Spill* requires placing a ball into a bowl and moving the bowl onto a cutting board without dropping the ball. *Tidy Up Book* requires extracting a book from a pile, grasping it, and placing it into a basket. *Insert Box* requires pushing a box against a wall to stand it upright before inserting it between a row of books.

All three tasks demand highly reactive behavior. Picking up the ball requires dynamic grasping, moving the bowl without tilting it relies on proprioceptive feedback, and both extracting a book from a pile and pushing a box upright require precise contact-rich interactions and reactive control. We collect 200, 300, and 300 teleoperated demonstrations for *Don't Spill*, *Tidy Up Book*, and *Insert Box*, respectively.

Baselines. We compare $\pi\mathbf{R}^2$ against three baselines, all fine-tuned from GR00T-N1.7. (1) *Flow, Synchronous* stalls the robot for the full $d=4 \sim 5$ ticks while inference runs, then executes the predicted chunk ($h = 10$). (2) *Flow, Naive Async (dense, TE)* [131] runs the policy asynchronously without conditioning on in-flight actions: *dense* re-issues a query every d ticks (whenever the previous call returns), and *TE* (temporal ensembling) [130] for overlapping action predictions for the same timestep. (3) *Flow, Train-Time RTC* [12] additionally trains the policy to inpaint a clean d -action front, anchoring the new chunk to the executed history at the chunk boundary.

Results. Table 6.1 summarizes the experimental results. $\pi\mathbf{R}^2$ with asynchronous vision-language inference operates near the per-control-tick limit ($d=1$ at 25 Hz, with occasional increases to $d=2$ under network delays), whereas all flow-based baselines incur the full GR00T pipeline latency of $d \in \{4, 5\}$. Synchronous inference produces jittery motion at chunk boundaries because the robot pauses while waiting for the next action chunk, leading to failures such as the ball rolling off the table. Naive asynchronous inference combined with temporal ensembling yields smoother motion but sacrifices execution precision. Train-Time RTC [12] is the strongest baseline; however, $\pi\mathbf{R}^2$ outperforms it across all metrics, with the largest gains on reactivity-critical tasks such as *Tidy Up Book* and *Insert Box*, achieving approximately 50% relative improvement. We observe that RTC also struggles to recover from failures because previously generated in-flight actions bias the policy toward continuing its recent motion, which reduces its ability to react promptly, particularly in *Insert Box*. Qualitatively, $\pi\mathbf{R}^2$ produces smoother and more accurate motions, even during failure recovery. These results demonstrate the value of real-time proprioceptive feedback enabled by $\pi\mathbf{R}^2$.

6.3 Limitations

Our approach has several limitations. First, we do not address sources of latency external to the model itself, such as communication delays between the inference server and the robot client. Second, we keep the base policy architecture unchanged; designing it to emphasize proprioceptive features more strongly (e.g., dedicated attention heads for proprioceptive tokens) could further amplify reactivity, and we leave this direction to future work.

6.4 Summary

We presented $\pi\mathbf{R}^2$, a principled modification to existing VLA architectures that yields real-time, closed-loop policies through two orthogonal contributions: asynchronous vision/text processing that decouples the slow VLM backbone from the action loop, and a latency-adaptive flow schedule that absorbs variable inference delay via a per-position noise schedule parameterized by d . Together, they reduce per-call delay by up to $4\times$ while remaining compatible with any VLA action head. This enables $\pi\mathbf{R}^2$ to handle highly reactive behaviors – non-prehensile, contact-rich manipulation – where baselines struggle, both in simulation and on real hardware. Additional implementation details, training and inference algorithms, and hyperparameters are provided in Appendix C.

Chapter 7

Discussion

The three methods in this thesis target different bottlenecks in deploying generalist policies (data, inference, and architecture) but they are more alike than their applications suggest. This chapter draws out what they share, how they might compose, and the limitations they have in common.

7.1 Adapting, Steering, and Accelerating: A Unified View

A generative policy turns noise into an action through a sequence of denoising steps guided by a learned score/flow. Each method in this thesis intervenes on that process, adding capability with little or no additional training. *DemoDiffusion* (Chapter 4) intervenes on the *starting point* and *extent* of denoising: instead of sampling from pure noise, it seeds the process with a retargeted human trajectory and denoises only partway, so a frozen policy adapts to a new task from one demonstration. TSR (Chapter 5) intervenes on the *score/flow itself*: rescaling it during sampling draws from a sharper or broader distribution than the model was trained on, again with no additional training. $\pi\mathbf{R}^2$ (Chapter 6) intervenes on the *conditioning and schedule* of denoising: with a lightweight finetune, it decouples fast and slow conditioning channels and reshapes the per-position noise schedule so that actions can be emitted in a single step and the loop closed in real time. Viewed this way, the thesis studies how much can be gained by treating the generative process of a pretrained policy, rather than a fresh model trained from scratch, as the object of design.

The interventions are largely orthogonal and could in principle compose. A deployed generalist policy might be adapted to a new task from a human demonstration with

DemoDiffusion, have its sampling sharpened by TSR to pick more reliable actions, and be made reactive with $\pi\mathbf{R}^2$, the first two needing no additional training and the third only a lightweight finetune. TSR in particular is not robot-specific: because it acts on the score of any diffusion or flow model, the same mechanism that sharpens a manipulation policy also improves image generation, depth estimation, pose prediction, and protein design. This suggests that lessons about controllable inference transfer across modern generative models, not just robot policies.

7.2 Limitations

Several limitations are common to the three methods. First, all inherit the capabilities and failure modes of the pretrained model they build on: *DemoDiffusion* cannot refine a trajectory into a behavior the base policy has no notion of, $\pi\mathbf{R}^2$ does not change what the backbone represents, and TSR can only reweight the distribution a model has already learned, not add missing modes. Second, each exposes inference-time hyperparameters (the denoising strength for *DemoDiffusion*, the schedule and channel timing for $\pi\mathbf{R}^2$, and the rescaling magnitude and onset for TSR) that currently require task-specific tuning, though each chapter offers guidelines. Finally, the methods are demonstrated on today’s policies; whether the same interventions remain the best levers as backbones and training recipes evolve is an open question.

7.3 Future Work

These observations suggest two main directions for future work. The first concerns *data for dexterity*: all three methods build on a pretrained policy that itself requires real robot data, and none of them magically extends a policy to more dexterous tasks that this data does not already cover. Enabling dexterous manipulation, especially with multi-fingered robotic hands, will therefore depend on better ways to collect dexterous real-robot data, for example through large-scale training in simulation or assistive teleoperation that lowers the cost and improves the quality of data collection. The second is specific to $\pi\mathbf{R}^2$: it feeds fresh proprioceptive and tactile signals to the policy more frequently, but does not change *how* those signals are conditioned or fused into the policy. Studying better ways to condition proprioception and touch, so that a policy exploits these fast, contact-rich signals rather than merely receiving them more often, is a promising direction for more reactive and contact-rich manipulation.

Chapter 8

Conclusion

This thesis started from a practical tension. Pretrained generalist policies, vision-language-action models with expressive diffusion and flow-matching heads, are remarkably capable, yet turning one into a working robot exposes stubborn gaps: adapting it to a new task still needs task-specific data, sampling from it offers no control over the distribution it draws from, and running it fast enough to react strains a large backbone. The unifying claim is that these gaps can be closed not by training larger models, but by changing how a pretrained policy generates its actions, with little or no additional training.

We developed this claim through three methods, each working on the same generative process at a different point. *DemoDiffusion* (Chapter 4) works on *data*: it turns a frozen generalist policy into a one-shot imitator by treating a kinematically retargeted human demonstration as a partially noised trajectory and denoising it back onto the robot’s action manifold. It succeeds even where the base policy fails and needs no task-specific or paired data. TSR (Chapter 5) works on *inference* in full generality: rescaling the learned score during sampling acts as a local temperature that sharpens or broadens the output distribution of any diffusion or flow model, improving robot policies alongside image generation, depth, pose, and protein design, with no fine-tuning. $\pi\mathbf{R}^2$ (Chapter 6) works on *architecture and inference*: with a lightweight finetune, it splits conditioning into fast proprioceptive and slow vision-language channels and adopts a latency-adaptive, single-step flow schedule, making a large action-chunking policy reactive and real-time on real hardware.

Seen together, adapting, steering, and accelerating are three ways of cashing out the same idea: the process that turns a generative policy into behavior is itself a design surface, and working with it, rather than only scaling the model behind it, is a powerful and inexpensive way to make capable models deployable. We hope this

perspective encourages more work that treats the generative process, not only the weights, as a first-class part of building generalist robots.

Appendix A

DemoDiffusion: Additional Details

This appendix provides additional experimental details, implementation details, and results for the real-world and simulation experiments of *DemoDiffusion* (Chapter 4).

A.1 Real-World Manipulation

A.1.1 Hardware Setup

We replicate the DROID [55] setup for the hardware. Due to hardware availability, we use a RealSense D455 instead of a ZED 2 for the rear-view camera.

A.1.2 Task Setup Description

The overview of the scenes for each task at test time is shown in Figure A.1. During data collection, we mark the target object location with a tape. During evaluation, we randomize the target object location within the tape, which is consistent across all methods. Overall, kinematic retargeting may fail due to small misalignment of object placement, inaccurate 3D hand keypoints, or camera extrinsics, etc. Pi-0 may fail if it cannot identify the object to interact with and manipulates other objects in the scene. It may also reach the target object but fail to manipulate it in the desired way. Note that previous approaches [17, 61, 82, 111] use kinematic retargeting to transfer human demonstrations to the robot.

Pick and Place the Banana. The task is to pick up the toy banana and place it on the plate on the right side of the table. The episode is considered successful if the banana is successfully placed on the plate.

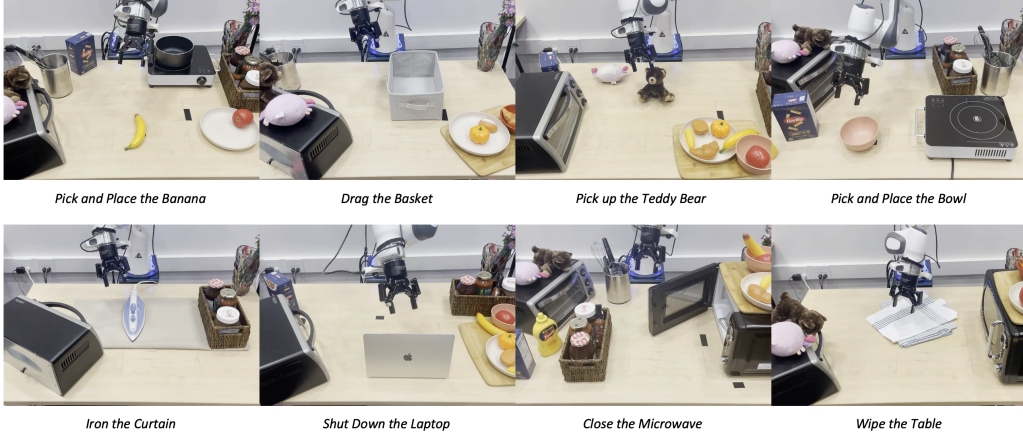


Figure A.1: **Real-world manipulation tasks at test time.** Each image corresponds to an example of the scene for each task at test time. To test our method in a natural setup, we add other objects to the scene as visual distractors. For each rollout, we use the same set of distractors across methods.

Drag the Basket. The task is to drag the basket to the right side of the table. The human demonstration grasps the edge of the basket and pulls it to the right. The episode is considered successful if the robot grasps the basket / puts the gripper inside the basket and moves it to any side of the table.

Pick Up the Teddy Bear. The task is to pick up the teddy bear located at the center of the table. The human demonstration grasps the neck of the bear and picks it up. The episode is considered successful if the robot picks up the bear by the end of the episode.

Pick and Place the Bowl. The task is to pick up the bowl and place it on the stove on the right side of the table. The episode is considered successful if the bowl is successfully placed on the stove.

Iron Curtain. The task is to grasp the ironing machine located at the center of the table and slide it to the right side of the curtain. The episode is considered successful if the robot grasps/pushes the ironing machine to any part of the curtain.

Shut Down the Laptop. The task is to close the laptop located in front of the robot. The episode is considered successful if the angle of the laptop is smaller than 10 degrees at the end of the episode.

Close the Microwave. The task is to close the microwave located at the corner of the table. The episode is considered successful if the handle part touches the body part of the microwave at the end of the episode.

Wipe the Table. The task is to pinch/poke the given tissue and wipe the table. The episode is considered successful if the robot pinches/pokes the tissue and performs one stroke to any side of the table without missing it.

A.1.3 Text Prompts

We use the following text conditions for Pi-0 [11], which are also used for *DemoDiffusion*.

Table A.1: Text prompts used for real-world manipulation tasks.

Task	Prompt
Pick and Place the Banana	<i>pick and place the banana on the plate</i>
Drag the Basket	<i>drag the basket to the right</i>
Pick Up the Teddy Bear	<i>pick up the bear doll</i>
Pick and Place the Bowl	<i>pick and place the bowl on the stove</i>
Iron the Curtain	<i>move the ironing machine to the right</i>
Wipe the Table	<i>wipe the table</i>
Close the Microwave	<i>close the microwave</i>
Shut Down the Laptop	<i>close the laptop</i>

A.1.4 Inference Details

At each timestep t , we index the human demonstration at the same timestep t and compute the kinematically retargeted actions. As Pi-0 predicts a chunk of actions with length $H = 10$, we also use the action chunk computed from the human demonstration from timestep t to $t + H - 1$. The episode length is set to be the same as the length (number of timesteps) of the human demonstrations across all methods.

A.1.5 Kinematic Retargeting Implementation Details

A human demonstration $\mathbf{D}$ of a manipulation task $\mathcal{T}$ consists of the 3D hand pose trajectory $\{\mathbf{h}_t\}_{t=0}^T$. Each hand pose $\mathbf{h}_t \in \mathbb{R}^{3 \times J}$ corresponds to the 3D locations of J

Table A.2: Hyperparameters in the real-world experiments.

Hyperparameter	Value
Open Loop Horizon	8
Predict Action Horizon	10
Denoising Steps at Inference (Pi-0)	20

keypoints (*e.g.* wrist and fingertips). To kinematically retarget these 3D keypoints to a robotic end-effector, we use the human’s wrist position A , thumb tip position B , and the average of the other fingertips C . Specifically, $\vec{AB}$, along with the cross product of $\vec{AB}$ and $\vec{AC}$, defines the orientation of the end-effector, and the human’s wrist position is used as the position of the end-effector. As Pi-0 uses joint velocity as its action space, the end-effector pose is further converted into joint velocity using inverse kinematics. For the gripper, we use the norm of $\vec{BC}$ as its width. To stabilize grasping, we set the gripper action to be fully closed if the norm of $\vec{BC}$ is smaller than 80% of the gripper’s maximum width, similar to Lepert et al. [60].

A.1.6 Hyperparameters

To show our method’s applicability across diverse setups, we do not change the speed of the given human demonstration when performing kinematic retargeting. At inference, we use the following hyperparameters for Pi-0 and *DemoDiffusion*, with the only difference being the number of denoising steps, as *DemoDiffusion* starts denoising from intermediate steps. Specifically, we use the hyperparameters in Table A.2. Depending on the noise level s^*/S , *DemoDiffusion* uses $20 \times s^*/S$ denoising steps.

A.1.7 Additional Results

Results with different hyperparameters. We present the results of *DemoDiffusion* using two different hyperparameter settings, $s^*/S = 0.2$ and $s^*/S = 0.4$. The results are summarized in Table A.3. Overall, we observe a consistent trend based on the performance of the base diffusion policy (*Pi-0*):

- When the base policy *Pi-0* achieves a non-zero success rate, $s^*/S = 0.4$ generally yields better performance. This is observed in tasks such as *shut laptop*, *close microwave*, *drag basket*, *wipe table*, and *iron curtain*.

Table A.3: **Additional quantitative results for real-world manipulation.** Success rates (%) for *DemoDiffusion* at two noise levels compared with the baselines.

Task	Pi-0	Kinematic Retargeting	<i>DemoDiffusion</i> (0.2)	<i>DemoDiffusion</i> (0.4)
Shut Laptop	20	10	40	60
Close Microwave	10	80	80	90
Drag Basket	10	80	70	80
Wipe Table	50	0	60	100
Iron Curtain	20	70	70	90
Pick Up Bear	0	40	60	40
Pick Place Banana	0	70	90	20
Pick Place Bowl	0	70	100	0
Average	13.8	52.5	71.3	60.0

- Conversely, when *Pi-0* has a zero success rate, $s^*/S = 0.2$ tends to perform better. This is the case for tasks including *pick up bear*, *pick place banana*, and *pick place bowl*. We observe that while $s^*/S = 0.4$ shows adaptive end-effector motion, it tends to avoid activating the gripper at the correct timestep.

These findings suggest that a more effective base policy enables *DemoDiffusion* to benefit from a higher noise level, as it can leverage the base policy’s strengths more effectively. In contrast, when the base policy is less reliable, a lower noise level tends to show better performance. We believe this can serve as a guideline for choosing the noise level s^*/S for *DemoDiffusion*.

Zero-shot generalization to new objects. Lastly, we investigate whether *DemoDiffusion* can achieve zero-shot generalization to new objects within the same category. We use the same human demonstration and text prompt we previously used while evaluating on a new object within the same category. Results are shown in Figure A.2. Although we only provide a human demonstration on a different object, we observe that *DemoDiffusion* can adapt its behavior to a new object.



Figure A.2: **Zero-shot generalization to new objects.** We evaluate *DemoDiffusion*’s generalization capabilities to new objects on 3 tasks – drag the basket, wipe the table, and pick and place the bowl. While the objects have different colors and shapes from those of the objects in the human demonstration, *DemoDiffusion* adjusts its behavior to these new objects. Please refer to the website for detailed visualizations.

A.2 Dexterous Grasping in Simulation

A.2.1 Task Setup Description

In our simulation experiment, the task is to pick up the object. The rollout is considered a success when the object’s z position is higher than 0.10 m at the end of the episode. Similar to the real-world experiments, at each timestep of inference, we compute a kinematically retargeted action chunk from the human demonstration at the same timestep.

Training data. As we do not have a *generalist* pre-trained diffusion policy in simulation, we first train a diffusion policy while constraining the target task to picking up generic objects. Specifically, we collected a total of 985 grasping trajectories of the Allegro hand over 58 objects in the RaiSim simulator for training. The trajectories were generated by rolling out the expert Allegro RL policy with random grasping directions. The 58 objects are randomly sampled from ShapeNet [15] (26 objects) and PartNet [76] (32 objects). The training objects and expert Allegro RL policy are provided by GraspXL [125].

Test data. At test time, we provide human grasping trajectories for a different set of objects. As it is challenging to provide human demonstrations in simulation, we instead use a MANO [93] hand RL policy to collect human demonstrations. The test set contains a total of 1220 objects, which is a subset of the Objaverse dataset [23]. The objects are divided into three groups based on their size: small scale $s \in [3, 5]$ cm,

medium scale $m \in [5, 7]$ cm, and large scale $l \in [7, 9]$ cm. As the object meshes do not contain material information, we assume fixed density, resulting in diverse mass. We additionally use the same friction coefficient for all objects. Again, the test objects and expert MANO [93] RL policy are provided by GraspXL [125]. Additionally, we do not change the speed of the given human demonstration when performing kinematic retargeting.

A.2.2 Kinematic Retargeting Implementation Details

Following Wang et al. [111], we use inverse kinematics to match the fingertip positions between the human hand and the robot hand, ignoring the pinky finger. The pose of the robot hand base is initialized to be the same as that of the human hand.

A.2.3 Robot Policy Implementation Details

We train a hierarchical policy, where the high-level planner predicts the robot’s future goal state, and the low-level controller predicts the action given the desired future robot state. Both policies were trained using a variant of 3D Diffusion Policy [123], where we modify the point cloud encoder to take point tracks, for a richer representation of the object. Specifically, while the original point encoder takes object points at each timestep, the modified encoder takes a sequence of object points (object tracks) as input.

The high-level planner and low-level controller use a subset of the following as input and output. All rotations are represented in Euler angles.

Robot state. Contains its wrist position in the world frame, rotation in the world frame, and absolute hand joint angles.

Object state. Contains 3D points of its surface. For a history of robot state, we assume 3D point tracks to be given (*i.e.* each 3D point across timesteps corresponds to the same point of the object).

Robot goal state. Contains its wrist position in the local frame (wrist frame at the previous timestep), rotation in the local frame, and relative hand joint angles to the previous timestep.

Action. Contains the desired robot’s wrist position in the local frame (previous wrist frame), desired rotation in the local frame, and desired relative hand joint angles to the previous timestep. Note that the action and the robot’s goal state are not the same. For instance, the desired hand joint angles can penetrate the object surface, while the robot’s state cannot. Such a difference creates contact force, enabling a stable grasp.

High-level planner. The high-level planner takes in a history of robot states and object states, and predicts a sequence of robot goal states. At test time, we use the kinematic retargeting result as an initial estimate for the robot’s goal state. We then apply *DemoDiffusion* to refine the robot goal state, and feed it to the low-level controller.

Low-level controller. The low-level controller takes in a history of robot states and a sequence of robot goal states, and predicts a sequence of robot actions. The goal state encoder has the same architecture as the robot state encoder. Once it outputs the action, the action is used as a position target for a PD controller provided in the RaiSim simulator.

A.2.4 Hyperparameters

For policy training, we use the default hyperparameters provided in the 3D Diffusion Policy [123] with minimal modifications. For the simulation environment, we use the default hyperparameters provided by GraspXL [125]. Specifically, we use the same parameters (Table A.4) for both the high-level planner and low-level controller.

Table A.4: Hyperparameters in the simulation experiments.

Hyperparameter	Value
H	4
N_{obs}	2
N_{act}	1
N_{goal}	2
$N_{latency}$	0
Number of Object Points	1024
Train Timesteps	1000
Inference Timesteps	10
Episode Length	155
Wrist Position Pgain	100.0
Wrist Position Dgain	0.1
Wrist Rotation Pgain	100.0
Wrist Rotation Dgain	0.2
Finger Rotation Pgain	100.0
Finger Rotation Dgain	0.2

Appendix B

Temporal Score Rescaling: Additional Details

This appendix supplements Chapter 5. Section B.1 provides additional qualitative and quantitative results across the application domains, together with mode-collapse, empirical score-error, and 2D toy-distribution studies. Section B.2 gives the full proof that TSR yields a valid score approximation for mixtures of well-separated Gaussians, and Section B.3 proves that Constant Noise Scaling cannot realize true temperature scaling.

B.1 Additional Results

B.1.1 Image Generation

We show additional qualitative examples generated by Stable Diffusion 3 with TSR in Figure B.3. We show TSR with varying k and highlight the control over the expression of high-frequency details. To better demonstrate the effect of user-input parameters (k, σ) on image quality, we plot FID and CLIP scores ablating over different (k, σ) in Figure B.1. The trends in Figure B.1 clearly demonstrate that TSR with a k slightly smaller than 1 and various σ values improves both metrics, and the optimal performance is achieved at $k = 0.93, \sigma = 3.0$. We compare the regular Euler-ODE sampling and TSR on different CFG guidance scales in Figure B.2, highlighting that TSR is orthogonal to CFG and improves model performance at various CFG settings. In Figure B.2, we also present the performance of CNS, which has to be applied on a stochastic sampler (Euler-SDE). As Stable Diffusion 3 is a flow matching model, stochastic samplers perform significantly worse than ODE samplers, especially when

the inference steps are less than 100. These results align with the findings in Ma et al. [71]. Therefore, CNS does not practically apply to flow matching models like SD3. In Table B.1, we additionally present quantitative results on Stable Diffusion 2, which is a denoising diffusion-based model. While CNS can improve over DDPM sampling, it is outperformed by TSR.

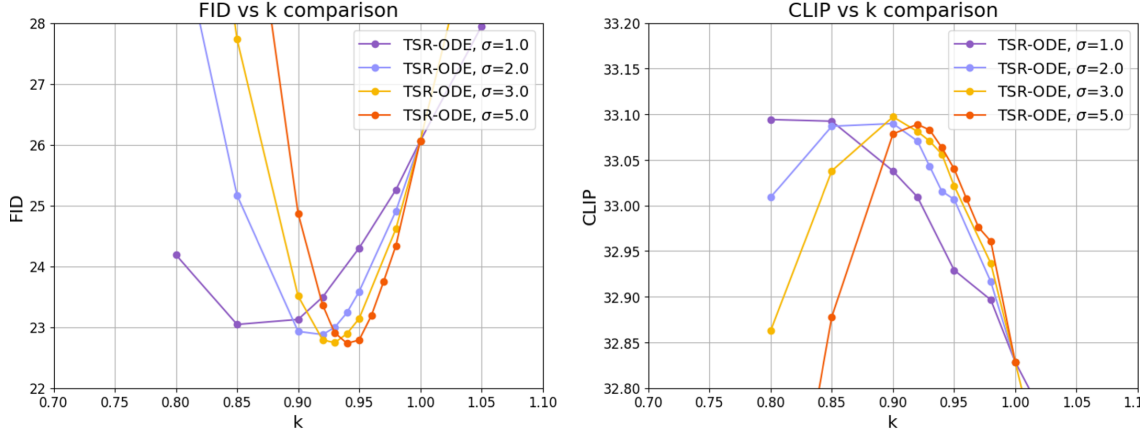


Figure B.1: Ablations over the TSR parameters (k, σ) on Stable Diffusion 3.

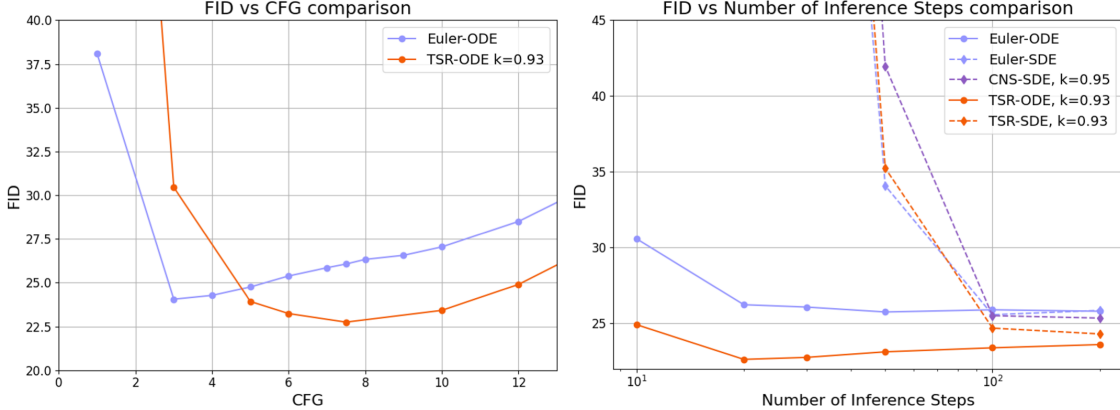


Figure B.2: Ablations over CFG scale and samplers. *Left*: Comparing regular sampling with TSR with various CFG guidance scale on Stable Diffusion 3. *Right*: Comparing deterministic and stochastic samplers with TSR or CNS. Stochastic sampling is much worse on flow models, making CNS impractical.

B.1.2 Pose Prediction

We show more pose prediction results in Figure B.5. TSR predicts tighter samples around the ground truth mode, which can be observed by the low spread of sampled

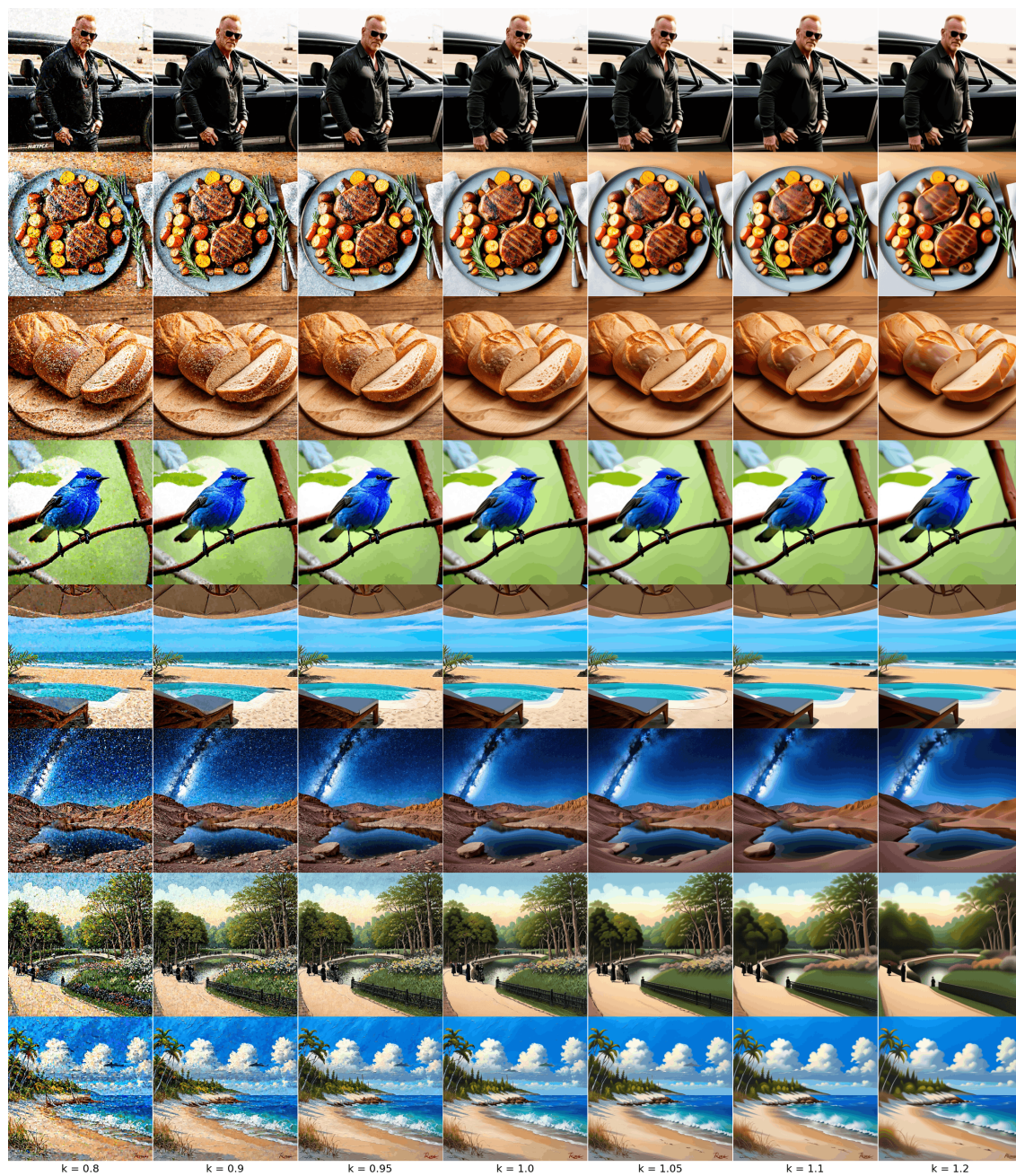


Figure B.3: Additional qualitative examples of TSR on Stable Diffusion 3.

Table B.1: **Results on Stable Diffusion 2.** TSR improves FID and CLIP on various samplers, outperforming CNS.

	FID ↓	CLIP ↑
DDIM	21.28	33.54
+ TSR (0.95, 1.0)	20.05	33.61
DDPM	22.81	33.66
+ Constant Noise Scaling (0.96)	19.87	33.68
+ TSR (0.9, 1.0)	19.57	33.77
EulerDiscrete	22.11	33.54
+ TSR (0.95, 1.0)	19.95	33.61

poses compared to score sampling. We also include ablations over parameters (k, σ) in Figure B.4, showing that TSR consistently improves accuracy by increasing k , across different σ .

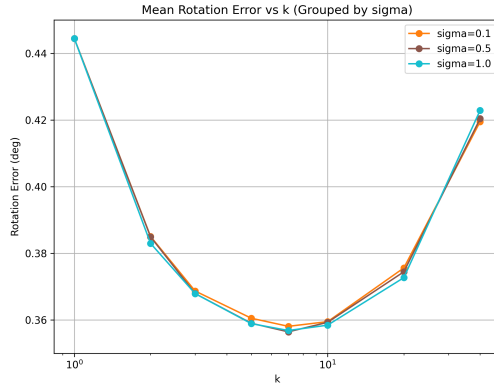


Figure B.4: **Ablations over (k, σ) on pose prediction.** TSR with various (k, σ) configurations effectively outperforms the baseline sampling method $k = 1$. While TSR is not sensitive to σ in pose estimation, TSR reaches optimal performance with $k \approx 7$.

B.1.3 Depth Estimation

We also show the effect of σ and k on the AbsRel metric in Figure B.6. Compared with the DDIM sample ($k = 1$), TSR demonstrates consistent performance gain in various (k, σ) configurations. We also include more depth samples and comparisons in Figure B.7. A consistent improvement of TSR result can be observed, compared to the DDIM samples.

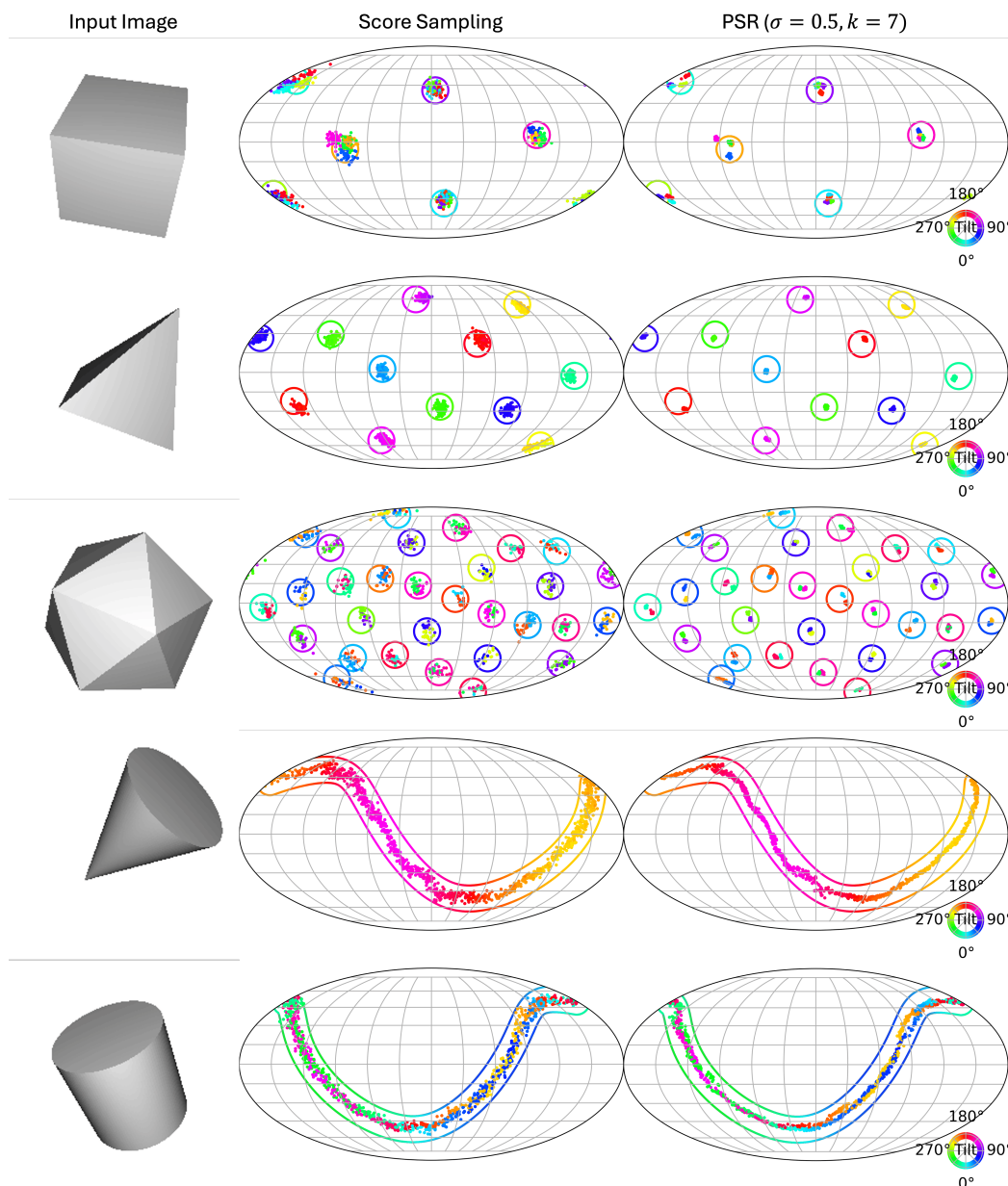


Figure B.5: **More predicted poses on SYMSOL.** We show all 5 classes of shapes in SYMSOL. We use $\sigma = 1, k = 7$ for these visualizations. TSR consistently reduces prediction error across all classes compared to score sampling. We modify the location of samples to exaggerate error by a factor of 15 to show the visual difference given plotting constraints.

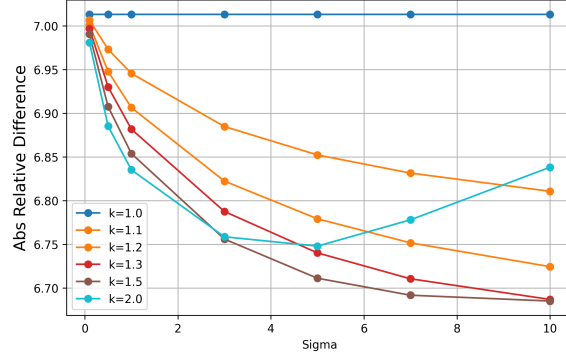


Figure B.6: **Effects of (k, σ) on depth estimation.** Comparing with DDIM sample ($k = 1$), TSR demonstrates consistent performance gains in various (k, σ) configurations.

B.1.4 Quantifying Mode Collapse

To systematically evaluate the mode-collapse behavior of temperature-scaling approaches (Constant Noise Scaling and TSR), we train an unconditional DDPM on the MNIST dataset [25] and apply each sampling method. We additionally train a classifier to label generated samples and assess whether Constant Noise Scaling or TSR exhibits mode drop, i.e., produces an imbalanced distribution of digits.

Figures B.8 and B.9 summarize the results, using $k = 5.0$ for Constant Noise Scaling (CNS) and $(k, \sigma) = (5.0, 1.0)$ for TSR. CNS disproportionately generates digits ‘1’ (40.4%) and ‘9’ (28.7%), likely because their straight or curved components appear frequently across other digits, making them easier to synthesize under decreased noise. In contrast, TSR produces a distribution of digits that closely matches that of DDPM, indicating that it preserves all modes. Furthermore, TSR generates noticeably clearer samples than DDPM, demonstrating the benefit of tempered sampling. In summary, TSR maintains mode coverage on MNIST while improving sample quality.

B.1.5 Empirical Analysis on Score Approximation

To empirically analyze our proposed approximation, we conduct an experiment using a 2D mixture of four Gaussian distributions, which is visualized in Figure B.10. We denote the distance between neighboring modes as Δ and the variance of each mode as σ . Setting the scaling parameter $k = 2$, we systematically vary Δ and σ to study the behavior of the approximation error. We quantify the deviation by computing the expected absolute relative difference (Abs. Rel.) between the score estimated by TSR and the ground-truth score.

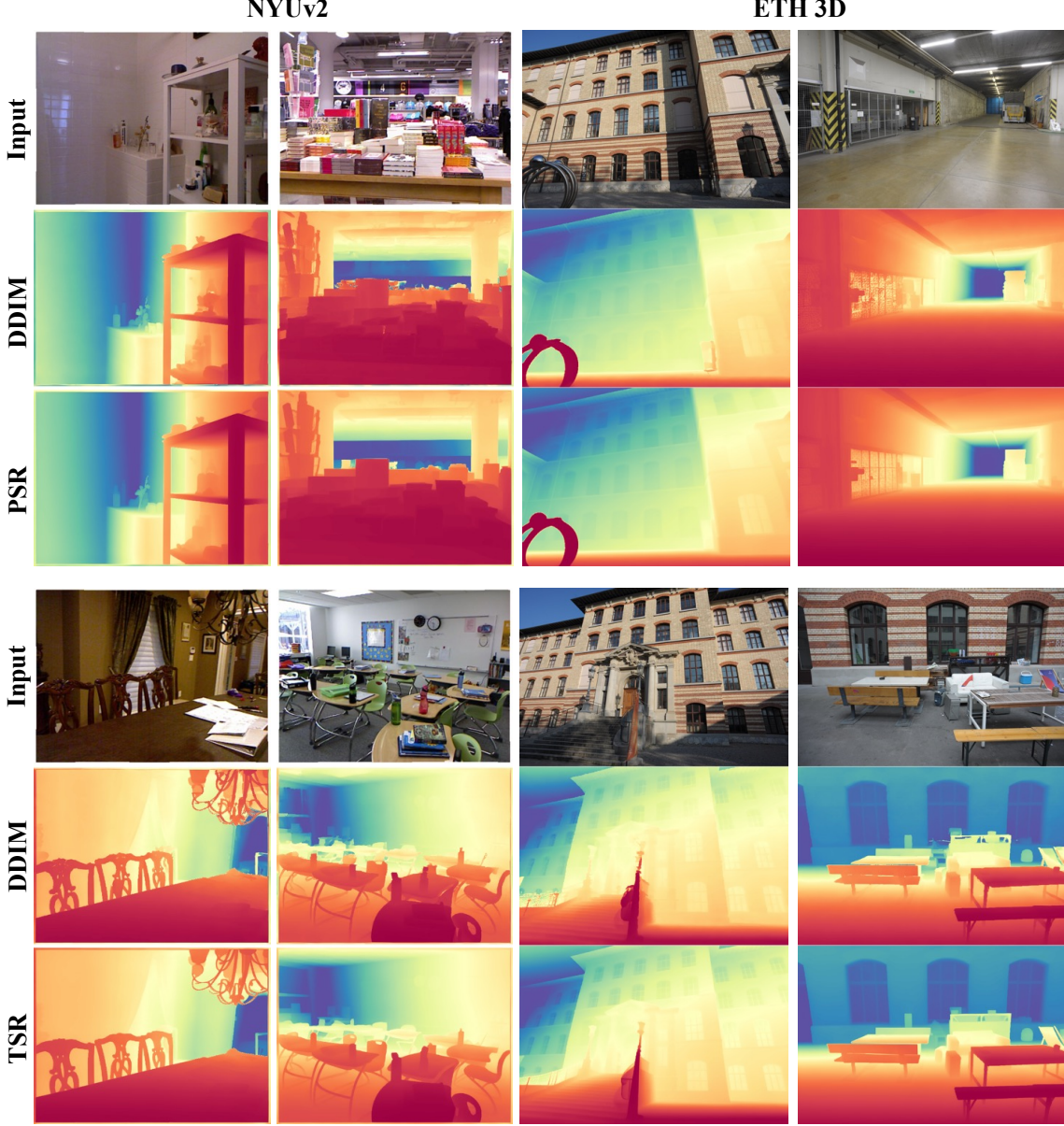


Figure B.7: **More Depth Prediction Comparison.** We include more samples from NYUv2 and ETH3D. TSR demonstrates consistent improvement compared to the DDIM samples.

As illustrated in Figure B.10, the error vanishes at both ends in the range of timestep t and peaks at intermediate t . Furthermore, we analyze the maximum error occurring across all timesteps with respect to σ and Δ . The results demonstrate that the maximum error vanishes as the mode variance σ decreases or the mode separation Δ increases, verifying that the approximation becomes exact as the modes are more separated. These results empirically confirm the theoretical bound we proved in

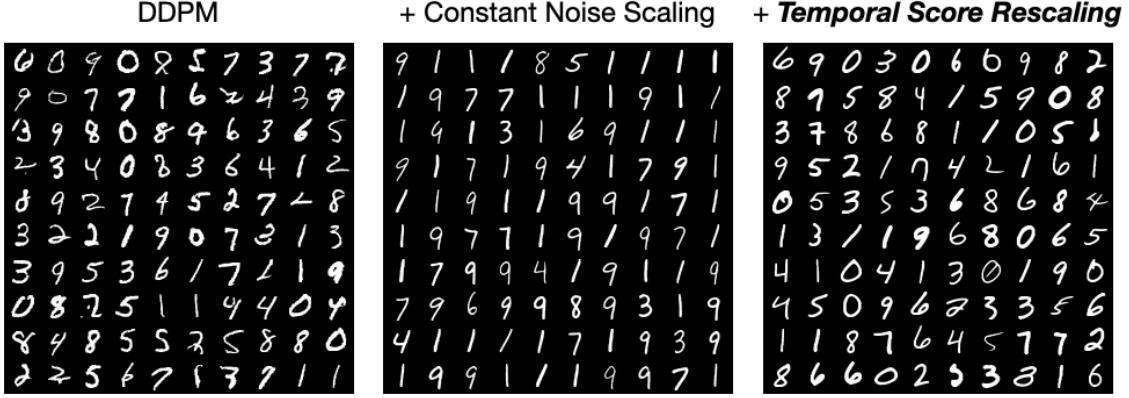


Figure B.8: Samples generated on MNIST using DDPM, Constant Noise Scaling (CNS), and TSR. CNS tends to favor generating 1 and 9, while TSR produces clearer digits while preserving diversity across modes.

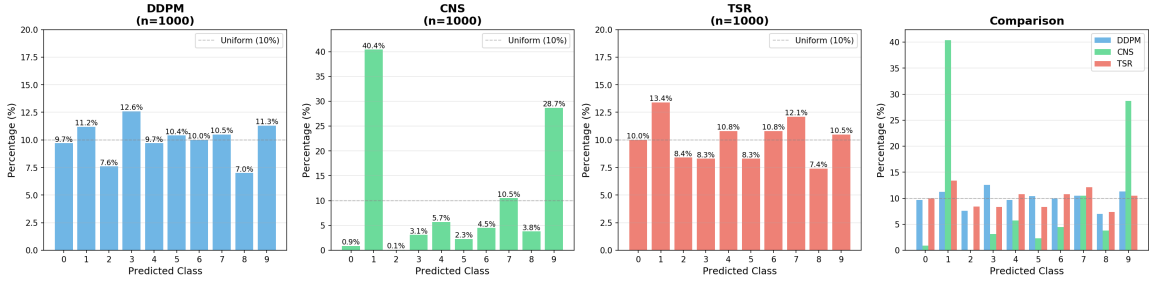


Figure B.9: Class distribution of generated MNIST samples under DDPM, CNS, and TSR (CNS: $k = 5.0$; TSR: $(k, \sigma) = (5.0, 1.0)$). CNS exhibits mode imbalance, whereas TSR maintains a balanced distribution consistent with the dataset.

Section B.2.

B.1.6 Additional Comparison on 2D Toy Distribution

We compare against two additional existing methods that can have similar effect as CNS in temperature sampling. One is constant score scaling (CSS), adopted by Skreta et al. [101]. Instead of scaling down the noise term like CNS in Equation B.5, CSS constantly scales the score prediction at each diffusion step, which is equivalent to solving the following reverse SDE:

$$d\mathbf{x} = [f(\mathbf{x}, t) - kg(t)^2 \nabla \log p_t(\mathbf{x})]dt + g(t)d\bar{\mathbf{w}} \quad (\text{B.1})$$

The other method is the MCMC corrector studied in [28, 104]. At each CNS sampling step, we apply additional n stochastic MCMC corrector steps with the score prediction

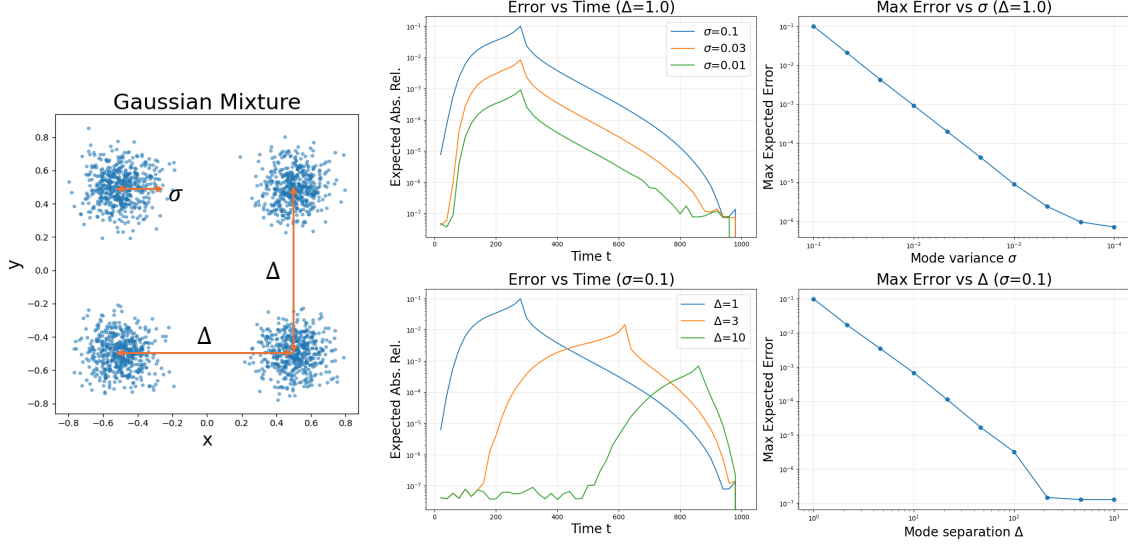


Figure B.10: **Empirical Score Approximation Error.** For the mixture of Gaussians depicted on the left, with mode distance Δ and mode variance σ , we compute the expected error of TSR approximation at $k = 2$. The maximum error is bounded and decreases as σ decreases or Δ increases (right column).

scaled by k , which approach the sampled distribution toward the target tempered distribution $p(x)^k$.

We compare TSR against the above two methods, along with CNS and DDPM sampling, on the same checkerboard setting shown in Figure 5.4. We present the qualitative comparison in Figure B.11. Constant score scaling (CSS) demonstrates similar mode-dropping behavior as CNS. We apply the MCMC corrector with CNS with 2x, 5x, and 10x more score evaluation compared to regular sampling. While the MCMC corrector alleviates mode-dropping, it still prefers central modes. TSR is the only method that preserves the uniform weights across modes in this setting.

B.2 Proof for TSR for Mixture of Well-Separated Gaussians

We show that for a mixture of well-separated Gaussians, the score approximation in TSR is valid, with the approximation error vanishing asymptotically.

We begin by introducing the notation and defining the estimation error in Section B.2.1. Our main result is stated in Section B.2.2. The proof of this result is given in Section B.2.3, supported by several lemmas whose proofs are provided in Section B.2.4.

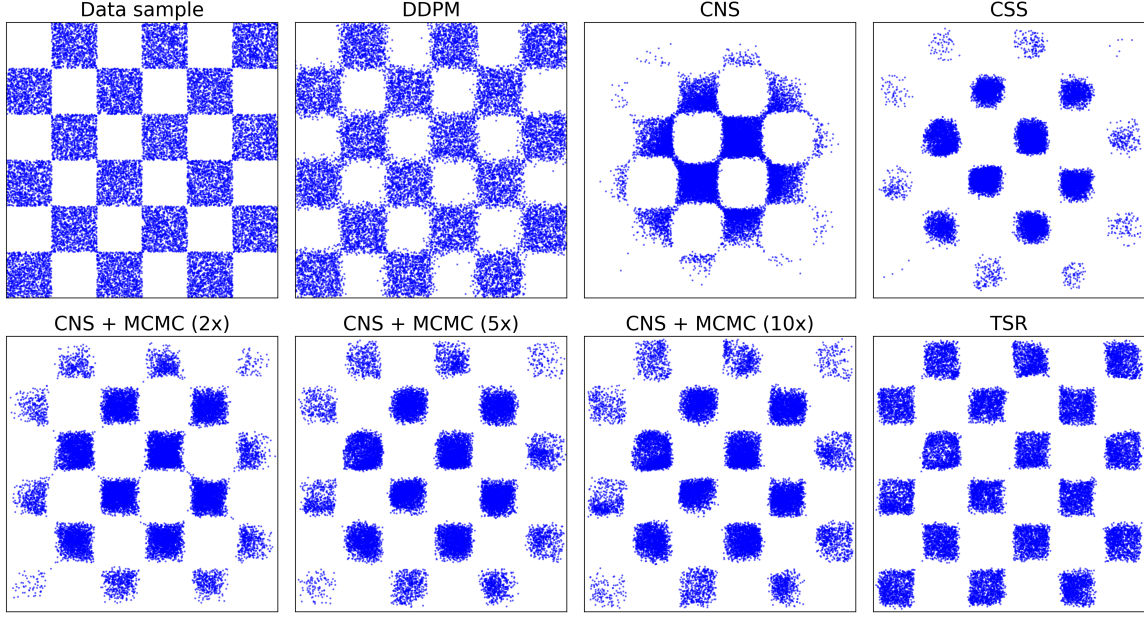


Figure B.11: **Comparison with the CSS and MCMC baseline.** On the 2D checkerboard distribution, both CNS and CSS demonstrate mode-dropping behavior. We apply the MCMC corrector to CNS with 2x, 5x, and 10x more score evaluations. While the MCMC corrector alleviates mode-dropping, it still prefers central modes.

Notations:

- α_t, σ_t, k : Diffusion/flow schedule coefficients and sharpening factor.
- $p_t^k(\mathbf{x})$: Induced distribution at time t given the data distribution sharpened by k .
- $\Delta \gg \delta$: Distance between two mixture means at $t = 0$. Define $\Delta_t = \alpha_t \Delta$.
- σ : Variance of each Gaussian in the mixture at $t = 0$.
- $\sigma_{t,k}^2 \equiv \frac{\alpha_t^2 \sigma^2}{k} + \sigma_t^2$: Variance of each Gaussian at time t with sharpening factor k .
- $\delta_{t,n}(\mathbf{x}) \equiv \mathbf{x} - \alpha_t \boldsymbol{\mu}_n$: Offset vector from $\mathbf{x}$ to the center of the n -th Gaussian at diffusion time t .
- $p_{t,n}^k(\mathbf{x}) \propto \exp\left(-\frac{\|\delta_{t,n}(\mathbf{x})\|^2}{\sigma_{t,k}^2}\right)$: Unnormalized density of $\mathbf{x}$ under the n -th Gaussian.
- $w_{t,n}^k(\mathbf{x}) \equiv \frac{p_{t,n}^k(\mathbf{x})}{\sum_m p_{t,m}^k(\mathbf{x})}$: Responsibility of the n -th Gaussian for $\mathbf{x}$.
- N : Number of Gaussians in the mixture. Dependent on the dataset only.
- d : Dimensionality of the data, i.e. $d = 2$ for 2D Gaussian Mixture.

- $\Delta_{\max} = \max_{i,j} |\boldsymbol{\mu}_i - \boldsymbol{\mu}_j|$: Maximum pairwise distance between Gaussian means in the mixture. For a general dataset, this term is bounded by $(N-1)\Delta$.

B.2.1 Error in TSR Score Approximation

Score. The score of the original data is given by:

$$\nabla \log p_t(\mathbf{x}) = -\frac{1}{(\alpha_t^2 \sigma^2 + \sigma_t^2)} \sum_n w_{t,n}^1(\mathbf{x}) \boldsymbol{\delta}_{t,n}(\mathbf{x})$$

For the target distribution $p^k(\mathbf{x}_0) = \sum_i \mathcal{N}(x; \boldsymbol{\mu}_i, \frac{\sigma^2}{k} \mathcal{I})$ the corresponding noisy distribution $p^k(\mathbf{x}_t) = \sum_i \mathcal{N}(x; \alpha_t \boldsymbol{\mu}_i, (\frac{\alpha_t^2 \sigma^2}{k} + \sigma_t^2) \mathcal{I})$, we have:

$$\nabla \log p_t^k(\mathbf{x}) = -\frac{1}{(\alpha_t^2 \sigma^2 / k + \sigma_t^2)} \sum_n w_{t,n}^k(\mathbf{x}) \boldsymbol{\delta}_{t,n}(\mathbf{x})$$

In TSR, we approximate the score of $p^k(\mathbf{x}_t)$ by:

$$\begin{aligned} \nabla \log \tilde{p}_t^k(\mathbf{x}) &\approx \frac{\alpha_t^2 \sigma^2 + \sigma_t^2}{\alpha_t^2 \sigma^2 / k + \sigma_t^2} \nabla \log p_t(\mathbf{x}) = \frac{\sigma_{t,1}^2}{\sigma_{t,k}^2} \left(-\frac{1}{\sigma_{t,1}^2} \sum_n w_{t,n}^1 \boldsymbol{\delta}_{t,n}(\mathbf{x}) \right) \\ &= -\frac{1}{\sigma_{t,k}^2} \sum_n w_{t,n}^1 \boldsymbol{\delta}_{t,n}(\mathbf{x}) \end{aligned}$$

Definition B.1 (Error in TSR Score Approximation). Define the amount of error in the score approximation as the expected difference between the scores:

$$Error(t) = \mathbb{E}_{\mathbf{x} \sim p_t^k} \left\| \frac{1}{\sigma_{t,k}^2} \sum_n (w_{t,n}^1(\mathbf{x}) - w_{t,n}^k(\mathbf{x})) \boldsymbol{\delta}_{t,n}(\mathbf{x}) \right\|$$

B.2.2 Upper Bound of the Error

The objective of this proof is to establish a bound on the error term $Error(t)$. Our main results are as follows:

Theorem B.2 (Upper Bound of the Error). *For $Error(t)$, there exist two upper bounds:*

$$\begin{aligned} Error(t) &\leq B_{exp} = 6 \cdot \frac{\alpha_t \Delta_{\max}}{\sigma_{t,k}^2} \cdot \exp \left(-\frac{\alpha_t^2 \Delta^2}{8\sigma_{t,1}^2} \right) \\ Error(t) &\leq B_{poly} = \frac{\alpha_t \Delta_{\max}}{4\sigma_{t,k}^2} \left(\frac{1}{\sigma_{t,k}^2} - \frac{1}{\sigma_{t,1}^2} \right) N(d\sigma_{t,k}^2 + \alpha_t^2 \Delta_{\max}^2) \end{aligned}$$

Theorem B.3 (Vanishing Behavior of Error). *Assuming $\sigma = \epsilon\Delta$, when $1 - \alpha_t^2 > \sqrt{\epsilon}$, we have:*

$$B_{\text{poly}} \sim O(\sqrt{\epsilon})$$

When $1 - \alpha_t^2 \leq \sqrt{\epsilon}$ (i.e. $\alpha_t \approx 1$) we have:

$$B_{\text{exp}} \sim O\left(\frac{1}{\sqrt{\epsilon}\Delta} \exp\left(-\frac{1}{\sqrt{\epsilon}}\right)\right)$$

Conclusion. Combining Theorem B.2 and Theorem B.3, when $\epsilon \rightarrow 0$, we have $\text{Error}(t) \rightarrow 0$. Therefore, when the Gaussians are well-separated ($\epsilon \rightarrow 0$), the approximation error vanishes to 0.

B.2.3 Proof of Theorem

Before proving the theorems, we first state several lemmas that are useful to the proof, whose proof will be given in the next section.

Lemma B.4. *The TSR approximation error $\text{Error}(t)$ is bounded as follows:*

$$\text{Error}(t) \leq \frac{\alpha_t \Delta_{\max}}{\sigma_{t,k}^2} \mathbb{E}_{\mathbf{x} \sim p_t^k} \|\text{dist}(w_t^1(\mathbf{x}), w_t^k(\mathbf{x}))\| \quad (\text{B.2})$$

where $\text{dist}(w_t^1(\mathbf{x}), w_t^k(\mathbf{x})) = \sum_n \|w_{t,n}^1(\mathbf{x}) - w_{t,n}^k(\mathbf{x})\|$.

Lemma B.5. *There exists a polynomial bound for $\mathbb{E}_{\mathbf{x} \sim p_t^k} \|\text{dist}(w_t^1(\mathbf{x}), w_t^k(\mathbf{x}))\|$:*

$$\mathbb{E}_{\mathbf{x} \sim p_t^k} \|\text{dist}(w_t^1(\mathbf{x}), w_t^k(\mathbf{x}))\| \leq 6 \cdot \exp\left(-\frac{\alpha_t^2 \Delta^2}{8\sigma_{t,1}^2}\right)$$

Lemma B.6. *There exists an exponential bound for $\mathbb{E}_{\mathbf{x} \sim p_t^k} \|\text{dist}(w_t^1(\mathbf{x}), w_t^k(\mathbf{x}))\|$:*

$$\mathbb{E}_{\mathbf{x} \sim p_t^k} \|\text{dist}(w_t^1(\mathbf{x}), w_t^k(\mathbf{x}))\| \leq \frac{1}{4} \left(\frac{1}{\sigma_{t,k}^2} - \frac{1}{\sigma_{t,1}^2} \right) N\left(d\sigma_{t,k}^2 + \alpha_t^2 \Delta_{\max}^2\right)$$

Proof of Theorem B.2. Combining Lemma B.4 and Lemma B.5, we obtain the polynomial bound for $\text{Error}(t)$.

Similarly, Lemma B.4 and Lemma B.6 will give us the exponential bound for $\text{Error}(t)$. $\square$

Proof of Theorem B.3. For simplicity, we assume diffusion scheduling, that is, $\sigma_t^2 = 1 - \alpha_t^2$ in this part. We also assume $\sigma = \epsilon\Delta$. As the dataset is fixed, we can rewrite $\Delta_{\max} = c\Delta$, where c is a constant that only depends on the dataset.

Vanishing of Polynomial Bound. Following the polynomial bound from Theorem B.2, we have:

$$\begin{aligned} B_{\text{poly}} &= N \frac{\alpha_t \Delta_{\max}}{\sigma_{t,k}^2} \left(\frac{(1 - 1/k) \sigma^2 \alpha_t^2}{\sigma_{t,1}^2 \sigma_{t,k}^2} \right) (d \sigma_{t,1}^2 + \alpha_t^2 \Delta_{\max}^2) \\ &= N (1 - 1/k) \frac{\alpha_t^3 \Delta_{\max} \sigma^2}{\sigma_{t,k}^4} \left(d + \frac{\alpha_t^2 \Delta_{\max}^2}{\sigma_{t,1}^2} \right) \end{aligned}$$

Consider $1 - \alpha_t^2 > \sqrt{\epsilon} \Delta^2$, we have: $\sigma_{t,k}^2 = \alpha_t^2 \sigma^2 / k + (1 - \alpha_t^2) > (1 - \alpha_t^2) > \sqrt{\epsilon} \Delta^2$. Therefore, we have:

$$\frac{\alpha_t^3 \Delta_{\max} \sigma^2}{\sigma_{t,k}^4} d \leq \frac{c \alpha_t^3 \epsilon^2 \Delta^3}{\epsilon \Delta^4} d = \alpha_t^3 c d \frac{\epsilon}{\Delta}$$

Since $\alpha_t \leq 1$ and c and d are constant given a dataset, we can absorb them into a constant. Therefore, $\frac{\alpha_t^3 c \Delta_{\max} \sigma^2}{\sigma_{t,k}^4} d \leq C_1 \frac{\epsilon}{\Delta}$, for some $C_1 = O(cd)$.

Similarly to previously proved, for the second term, $\frac{\alpha_t^3 \Delta_{\max} \sigma^2}{\sigma_{t,k}^4} \cdot \frac{\alpha_t^2 \Delta_{\max}^2}{\sigma_{t,1}^2}$, we have:

$$\frac{\alpha_t^5 \sigma^2 \Delta_{\max}^3}{\sigma_{t,k}^4 \sigma_{t,1}^2} \leq \frac{\alpha_t^5 c^3 \Delta^3 (\epsilon^2 \Delta^2)}{\epsilon \Delta^4 \cdot \sqrt{\epsilon} \Delta^2} \leq C_2 \frac{\sqrt{\epsilon}}{\Delta}$$

where C_2 is a constant term based on the dataset (and α_t).

Therefore, we have the following.

$$B_{\text{poly}} \leq C_1 \frac{\epsilon}{\Delta} + C_2 \frac{\sqrt{\epsilon}}{\Delta} \leq C \frac{\sqrt{\epsilon}}{\Delta}$$

We can see that the polynomial bound is $O(\sqrt{\epsilon})$ for such α_t , which goes to 0 as $\epsilon \rightarrow 0$.

Vanishing of Exponential Bound. Assuming the diffusion schedule, and consider α_t such that $1 - \alpha_t^2 < \sqrt{\epsilon} \Delta^2$, we have:

$$B_{\text{exp}} = 6 \frac{\alpha_t \Delta_{\max}}{\alpha_t^2 \sigma^2 / k + 1 - \alpha_t^2} \exp\left(-\frac{\alpha_t^2 \Delta^2}{8(\alpha_t^2 \sigma^2 + 1 - \alpha_t^2)}\right)$$

With our assumption of $\sigma = \epsilon \Delta$, for a small ϵ :

$$\begin{aligned} \alpha_{t,1}^2 &= \alpha_t^2 \sigma^2 + 1 - \alpha_t^2 = \alpha_t^2 \epsilon^2 \Delta^2 + (1 - \alpha_t^2) \\ &\leq 2(1 - \alpha_t^2) \leq 2\sqrt{\epsilon} \Delta^2 \\ -\frac{\alpha_t^2 \Delta^2}{8\alpha_{t,1}^2} &\leq -\frac{\alpha_t^2 \Delta^2}{8 \cdot 2\sqrt{\epsilon} \Delta^2} = -\frac{\alpha_t^2}{16\sqrt{\epsilon}} \end{aligned}$$

Therefore,

$$\exp\left(-\frac{\alpha_t^2 \Delta^2}{8\alpha_{t,1}^2}\right) \leq \exp\left(-\frac{\alpha_t^2}{16\sqrt{\epsilon}}\right)$$

As $\alpha_t^2 \sigma^2 / k + 1 - \alpha_t^2$ is dominated by $1 - \alpha_t^2$, we have $\alpha_t^2 \sigma^2 / k + 1 - \alpha_t^2 \approx 1 - \alpha_t^2$.

Therefore, we have:

$$B_{\text{exp}} \leq 6 \frac{\alpha_t \Delta_{\max}}{\alpha_t^2 \sigma^2 / k + 1 - \alpha_t^2} \exp\left(-\frac{\alpha_t^2}{16\sqrt{\epsilon}}\right) \approx \frac{6c\alpha_t}{\sqrt{\epsilon}\Delta} \exp\left(-\frac{\alpha_t^2}{16\sqrt{\epsilon}}\right)$$

As we consider α_t such that $1 - \alpha_t^2 < \sqrt{\epsilon}\Delta^2$, then we can write the exponential bound as $O(\frac{1}{\sqrt{\epsilon}\Delta} \exp(-\frac{1}{\sqrt{\epsilon}}))$, which also vanishes as $\epsilon \rightarrow 0$.

Conclusion. In both cases, at least one bound is vanishingly small as $\epsilon \rightarrow 0$. $\square$

B.2.4 Proof of Lemma

Proof of Lemma B.4. Upper bound of the error. Using the triangle inequality and the fact that $\sum_n w_{t,n}^1(\mathbf{x}) = 1$ and $\sum_n w_{t,n}^k(\mathbf{x}) = 1$, we have the following result:

$$\begin{aligned} \text{Error}(t) &= \mathbb{E}_{\mathbf{x} \sim p_t^k} \frac{1}{\sigma_{t,k}^2} \left\| \sum_n (w_{t,n}^1(\mathbf{x}) - w_{t,n}^k(\mathbf{x})) \boldsymbol{\delta}_{t,n}(\mathbf{x}) \right\| \\ &\leq \frac{1}{\sigma_{t,k}^2} \mathbb{E}_{\mathbf{x} \sim p_t^k} \left\| \sum_n (w_{t,n}^1(\mathbf{x}) - w_{t,n}^k(\mathbf{x})) \right\| \|\boldsymbol{\delta}_{t,n}(\mathbf{x})\| \\ &\leq \frac{1}{\sigma_{t,k}^2} \mathbb{E}_{\mathbf{x} \sim p_t^k} \left\| \sum_n ((w_{t,n}^1(\mathbf{x}) - w_{t,n}^k(\mathbf{x})) \alpha_t \boldsymbol{\delta}_{\max}) \right\| \\ &\leq \frac{\alpha_t \boldsymbol{\delta}_{\max}}{\sigma_{t,k}^2} \mathbb{E}_{\mathbf{x} \sim p_t^k} \sum_n \|w_{t,n}^1(\mathbf{x}) - w_{t,n}^k(\mathbf{x})\| \end{aligned}$$

Therefore, the approximation error is bounded as follows:

$$\text{Error}(t) \leq \frac{\alpha_t \Delta_{\max}}{\sigma_{t,k}^2} \mathbb{E}_{\mathbf{x} \sim p_t^k} \|\text{dist}(w_t^1(\mathbf{x}), w_t^k(\mathbf{x}))\| \quad (\text{B.3})$$

where $\text{dist}(w_t^1(\mathbf{x}), w_t^k(\mathbf{x})) = \sum_n \|w_{t,n}^1(\mathbf{x}) - w_{t,n}^k(\mathbf{x})\|$. $\square$

Proof of Lemma B.5. Exponential Bound. Following our problem setting, we have:

$$p_t(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^N \mathcal{N}(x; \alpha_t \boldsymbol{\mu}_i, (\alpha_t^2 \sigma^2 + \sigma_t^2) I).$$

and

$$q_t(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^N \mathcal{N}(x; \alpha_t \boldsymbol{\mu}_i, (\frac{\alpha_t^2 \sigma^2}{k} + \sigma_t^2) I).$$

where $p_t(\mathbf{x})$ is the original distribution, and $q_t(\mathbf{x})$ is the desired distribution with altered variance.

For each x , the responsibility vector under a mixture is defined as:

$$r^{(p)}(\mathbf{x}) = (r_1^{(p)}(\mathbf{x}), \dots, r_N^{(p)}(\mathbf{x}))$$

where $r_i^{(p)}(\mathbf{x}) = \frac{\mathcal{N}(x; \alpha_t \boldsymbol{\mu}_i, \alpha_t^2 \sigma^2 + \sigma_t^2)}{\sum_{j=1}^N \mathcal{N}(x; \alpha_t \boldsymbol{\mu}_j, \alpha_t^2 \sigma^2 + \sigma_t^2)}$. $r^{(q)}(\mathbf{x})$ is defined analogously as $r_i^{(q)}(\mathbf{x}) = \frac{\mathcal{N}(x; \alpha_t \boldsymbol{\mu}_i, \alpha_t^2 \sigma^2 / k + \sigma_t^2)}{\sum_{j=1}^N \mathcal{N}(x; \alpha_t \boldsymbol{\mu}_j, \alpha_t^2 \sigma^2 / k + \sigma_t^2)}$.

Now we have $\mathbb{E}_{\mathbf{x} \sim p_t^k} \|\text{dist}(w_t^1(\mathbf{x}), w_t^k(\mathbf{x}))\| = \mathbb{E}_{\mathbf{x} \sim p_t^k} [D(\mathbf{x})]$, where $D(\mathbf{x}) := \|r^{(p)}(\mathbf{x}) - r^{(q)}(\mathbf{x})\|_1$.

Define $i(\mathbf{x}) = \max_i r_i$, and e_i as the one-hot vector where the i th entry is one. Using the triangle inequality, we have:

$$D(\mathbf{x}) = \|r^{(p)} - r^{(q)}\|_1 \leq \|r^{(p)} - e_{i_p(\mathbf{x})}\|_1 + \|e_{i_p(\mathbf{x})} - e_{i_q(\mathbf{x})}\|_1 + \|e_{i_q(\mathbf{x})} - r^{(q)}\|_1.$$

and that

$$\begin{aligned} \|r^{(p)} - e_{i_p(\mathbf{x})}\|_1 &= 2(1 - r_{i_p(\mathbf{x})}^{(p)}(\mathbf{x})) \\ \|e_{i_p} - e_{i_q}\|_1 &= 2 * \mathbf{1}\{i_p \neq i_q\} \\ \|r^{(q)} - e_{i_q(\mathbf{x})}\|_1 &= 2(1 - r_{i_q(\mathbf{x})}^{(q)}(\mathbf{x})) \end{aligned}$$

Concentration of responsibilities for the true component. Let:

$$\epsilon := \max_{i \neq j} \mathbb{P}_{x \sim \mathcal{N}(\boldsymbol{\mu}_i, \sigma^2)} [\|x - \boldsymbol{\mu}_j\| < \|x - \boldsymbol{\mu}_i\|]$$

That is, the probability that a sample from component i is closer to another component j . Then:

$$\mathbb{E}_{x \sim p} \left[1 - \max_j r_j^{(p)}(\mathbf{x}) \right] \leq \epsilon \quad \Rightarrow \quad \mathbb{E}_{x \sim p} [D(\mathbf{x})] \approx 2\epsilon$$

Recall $\Delta := \min_{i \neq j} \|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|$ to be the minimum pairwise distance between the means. Using Gaussian tail bounds, we can approximate:

$$\epsilon \approx \exp\left(-\frac{\Delta^2}{8\sigma^2}\right)$$

Hence, we have:

$$\begin{aligned} E_{x \sim p_t^k} \left(2(1 - r_{i_p(\mathbf{x})}^p(\mathbf{x})) \right) &\leq 2 \cdot \exp \left(-\frac{\alpha_t^2 \Delta^2}{8\sigma_{t,1}^2} \right) \\ E_{x \sim p_t^k} \left(2(1 - r_{i_q(\mathbf{x})}^q(\mathbf{x})) \right) &\leq 2 \cdot \exp \left(-\frac{\alpha_t^2 \Delta^2}{8\sigma_{t,k}^2} \right) \end{aligned}$$

Bounding $\Pr(i_p \neq i_q)$. As $p_t(\mathbf{x})$ and $q_t(\mathbf{x})$ share the same modes, we have $\Pr(i_p \neq i_q) \leq \sum_i \Pr(i_p \neq i_q \mid x \sim \text{component } i) \Pr(x \text{ from } i)$, which can also be bounded using Gaussian tail bounds as above.

Therefore, we have:

$$\begin{aligned} E_{x \sim p_t^k}(D(\mathbf{x})) &\leq E_x(\|r^{(p)} - e_{i_p(\mathbf{x})}\|_1) + E_x(\|e_{i_p(\mathbf{x})} - e_{i_q(\mathbf{x})}\|_1) + E_x(\|e_{i_q(\mathbf{x})} - r^{(q)}\|_1) \\ &= E_x\left(2(1 - r_{i_p(\mathbf{x})}^p(\mathbf{x}))\right) + E_x\left(2 * \mathbf{1}\{i_p \neq i_q\}\right) + E_x\left(2(1 - r_{i_q(\mathbf{x})}^q(\mathbf{x}))\right) \\ &\leq 2 \cdot \exp\left(-\frac{\alpha_t^2 \Delta^2}{8\sigma_{t,1}^2}\right) + \left(\exp\left(-\frac{\alpha_t^2 \Delta^2}{8\sigma_{t,1}^2}\right) + \exp\left(-\frac{\alpha_t^2 \Delta^2}{8\sigma_{t,k}^2}\right)\right) + 2 \cdot \exp\left(-\frac{\alpha_t^2 \Delta^2}{8\sigma_{t,k}^2}\right) \\ &\leq 6 \cdot \exp\left(-\frac{\alpha_t^2 \Delta^2}{8\sigma_{t,1}^2}\right) \end{aligned}$$

Finally:

$$E_{x \sim p_t^k}(D(\mathbf{x})) \leq 6 \cdot \exp\left(-\frac{\alpha_t^2 \Delta^2}{8\sigma_{t,1}^2}\right)$$

□

Proof of Lemma B.6. Polynomial Bound. We consider the softmax representation of the responsibilities:

$$w_t^k(\mathbf{x}) = \text{softmax}(z_t^k(\mathbf{x})), \quad \text{where} \quad z_{t,n}^k(\mathbf{x}) := -\frac{\|\mathbf{x} - \alpha_t \boldsymbol{\mu}_n\|^2}{2\sigma_{t,k}^2}.$$

Using the Softmax Lipschitz bound that $\|\text{softmax}(z) - \text{softmax}(z')\|_1 \leq 1/2 \|z - z'\|_1$, we have:

$$\|w_t^k(\mathbf{x}) - w_t^1(\mathbf{x})\|_1 \leq \frac{1}{2} \|z_t^k(\mathbf{x}) - z_t^1(\mathbf{x})\|_1.$$

Compute the logits difference coordinatewise:

$$\begin{aligned} z_{t,n}^k(\mathbf{x}) - z_{t,n}^1(\mathbf{x}) &= -\frac{\|\boldsymbol{\delta}_{t,n}(\mathbf{x})\|^2}{2\sigma_{t,k}^2} + \frac{\|\boldsymbol{\delta}_{t,n}(\mathbf{x})\|^2}{2\sigma_{t,1}^2} \\ &= \frac{1}{2} \left(\frac{1}{\sigma_{t,1}^2} - \frac{1}{\sigma_{t,k}^2} \right) \|\boldsymbol{\delta}_{t,n}(\mathbf{x})\|^2. \end{aligned}$$

Adding absolute values,

$$\|z_t^k(\mathbf{x}) - z_t^1(\mathbf{x})\|_1 = \frac{1}{4} \left(\frac{1}{\sigma_{t,k}^2} - \frac{1}{\sigma_{t,1}^2} \right) \sum_{n=1}^N \|\boldsymbol{\delta}_{t,n}(\mathbf{x})\|^2$$

Bounding $\mathbb{E}_x \left[\sum_{n=1}^N \|\boldsymbol{\delta}_{t,n}(\mathbf{x})\|^2 \right]$. Let $x \sim p_t^k$ be drawn from the mixture with means $\{\alpha_t \boldsymbol{\mu}_i\}$ and variance $\sigma_{t,k}^2$. Write the expectation as a mixture-average:

$$\mathbb{E}_x \left[\sum_{n=1}^N \|\boldsymbol{\delta}_{t,n}(\mathbf{x})\|^2 \right] = \frac{1}{N} \sum_{i=1}^N \mathbb{E}_{x \sim \mathcal{N}(\alpha_t \boldsymbol{\mu}_i, \sigma_{t,k}^2 I)} \left[\sum_{n=1}^N \|x - \alpha_t \boldsymbol{\mu}_n\|^2 \right].$$

When the sample was generated from component i , for any other n , we have

$$\mathbb{E} \|x - \alpha_t \boldsymbol{\mu}_n\|^2 = \mathbb{E} \left[\|x - \alpha_t \boldsymbol{\mu}_i + \alpha_t \boldsymbol{\mu}_i - \alpha_t \boldsymbol{\mu}_n\|^2 \right] = \mathbb{E} \|x - \alpha_t \boldsymbol{\mu}_i\|^2 + \|\alpha_t \boldsymbol{\mu}_i - \alpha_t \boldsymbol{\mu}_n\|^2$$

because the cross-term has zero mean.

Since the first term equals the trace of the covariance $= d\sigma_{t,1}^2$, we have:

$$\mathbb{E} \|x - \alpha_t \boldsymbol{\mu}_n\|^2 = d\sigma_{t,1}^2 + \|\alpha_t(\boldsymbol{\mu}_i - \boldsymbol{\mu}_n)\|^2$$

Summing over all N (including $n = i$, for which the pairwise term is zero) gives $\mathbb{E}_{x \sim \mathcal{N}(\alpha_t \boldsymbol{\mu}_i, \sigma_{t,1}^2 I)} \left[\sum_{n=1}^N \|\boldsymbol{\delta}_{t,n}(\mathbf{x})\|^2 \right] = Nd\sigma_{t,1}^2 + \sum_{n=1}^N \|\alpha_t(\boldsymbol{\mu}_i - \boldsymbol{\mu}_n)\|^2$.

Now, bound the pairwise squared distances by the diameter squared: $\|\alpha_t(\boldsymbol{\mu}_{t,i} - \boldsymbol{\mu}_{t,n})\|^2 \leq \alpha_t^2 \Delta_{\max}^2$.

Therefore, we have: $\mathbb{E}_x \left[\sum_{n=1}^N \|\boldsymbol{\delta}_{t,n}(\mathbf{x})\|^2 \right] \leq N(d\sigma_{t,1}^2 + \alpha_t^2 \Delta_{\max}^2)$.

We then have the polynomial bound as:

$$\mathbb{E}_{x \sim p} [D(\mathbf{x})] \leq \frac{1}{4} \left(\frac{1}{\sigma_{t,k}^2} - \frac{1}{\sigma_{t,1}^2} \right) N(d\sigma_{t,k}^2 + \alpha_t^2 \Delta_{\max}^2)$$

□

B.3 Constant Noise Scaling

In this section, we provide a more detailed analysis of Constant Noise Scaling. As discussed in Section 3.4, CNS has been adopted as a practical technique to control the trade-off between sample variance and diversity. We intuitively explain and empirically verify that CNS does not correspond to true temperature scaling. We now provide a

more rigorous proof that CNS cannot produce the temperature-scaled distribution. Following Song et al. [104], a regular score-based model $\mathbf{s}_\theta(\mathbf{x}, t) = \nabla \log p_t(\mathbf{x})$ trained on data distribution $p_0(\mathbf{x})$ can sample by solving the reverse time diffusion SDE:

$$d\mathbf{x} = [f(t)\mathbf{x} - g(t)^2 \mathbf{s}_\theta(\mathbf{x}, t)]dt + g(t)d\bar{\mathbf{w}} \quad (\text{B.4})$$

where $f(t), g(t)$ are the time-dependent drift and diffusion coefficients, $d\bar{\mathbf{w}}$ is the standard Wiener process. CNS solves the following SDE instead:

$$d\mathbf{x} = [f(t)\mathbf{x} - (\frac{g(t)}{\sqrt{k}})^2 (k\mathbf{s}_\theta(\mathbf{x}, t))]dt + \frac{g(t)}{\sqrt{k}}d\bar{\mathbf{w}} \quad (\text{B.5})$$

Practically, CNS scales the stochastic noise added at each sampling step by $1/\sqrt{k}$. When $k > 1$, less noise is added and the process generates samples with reduced variance, and vice versa. To analyze the relationship between CNS and temperature scaling, we denote the temperature-scaled data distribution $q_0(\mathbf{x})$, such that $q_0(\mathbf{x}) \propto p_0(\mathbf{x})^k$.

Theorem B.7. *For general data distribution $p_0(\mathbf{x})$, there is no prior distribution $q'_T(\mathbf{x})$, such that Equation B.5 starts from $q'_T(\mathbf{x})$ and generates the temperature scaled distribution $q_0(\mathbf{x}) \propto p_0(\mathbf{x})^k$.*

Proof. We start by considering the following forward SDE:

$$d\mathbf{x} = f(t)\mathbf{x}dt + \frac{g(t)}{\sqrt{k}}d\mathbf{w} \quad (\text{B.6})$$

Let the initial distribution at $t = 0$ be $q_0(\mathbf{x})$, we define the time-dependent distribution generated by this forward SDE as $q_t(\mathbf{x})$. Then, one corresponding reverse SDE that can sample $q_0(\mathbf{x})$ takes the form of

$$d\mathbf{x} = [f(t)\mathbf{x} - (\frac{g(t)}{\sqrt{k}})^2 (\nabla \log q_t(\mathbf{x}))]dt + \frac{g(t)}{\sqrt{k}}d\bar{\mathbf{w}} \quad (\text{B.7})$$

Comparing Equation B.5 and Equation B.7, we can infer the following Lemma:

Lemma B.8. *The CNS reverse-time SDE Equation B.5 and the SDE Equation B.7 are equivalent if and only if $\nabla \log q_t(\mathbf{x}) = k\mathbf{s}_\theta(\mathbf{x}, t)$ for all time t .*

By construction, Equation B.7 evolves from $q_T(\mathbf{x})$ to $q_0(\mathbf{x})$. Now we assume CNS (Equation B.5) starts from the same prior distribution $q_T(\mathbf{x}) = \mathcal{N}(0, \frac{1}{k}\mathbf{I})$; by Lemma B.8, CNS correctly performs temperature scaling and samples from $q_0(\mathbf{x})$ if

and only if $\nabla \log q_t(\mathbf{x}) = k\mathbf{s}_\theta(\mathbf{x}, t)$. Now we show that this condition is not true in general.

Left Side. To compute $q_t(\mathbf{x})$, we need to solve the SDE in Equation B.6. For an initial condition $x = X_0$, the solution $X(t)$ is given by the following stochastic interpolant:

$$X(t) = \alpha_q(t)X_0 + \sigma_q(t)\epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I}) \quad (\text{B.8})$$

$$\begin{aligned} \alpha_q(t) &= \int_0^t f(s)ds = \alpha_t \\ \sigma_q(t) &= \int_0^t \frac{g(s)^2}{k} \exp(-2 \int_0^s f(u)du)ds = \frac{\sigma_t}{k} \end{aligned}$$

Therefore, we can compute the $q_t(\mathbf{x})$ by

$$q_t(\mathbf{x}) = \int q_0(\mathbf{y}) \mathcal{N}(\mathbf{x}; \alpha_t \mathbf{y}, \frac{\sigma_t^2}{k} \mathbf{I}) d\mathbf{y} \quad (\text{B.9})$$

Right Side. For the original diffusion process without scaling, we can compute the noisy distribution $p_t(\mathbf{x})$ at time t as

$$p_t(\mathbf{x}) = \int p_0(\mathbf{y}) \mathcal{N}(\mathbf{x}; \alpha_t \mathbf{y}, \sigma_t^2 \mathbf{I}) d\mathbf{y} \quad (\text{B.10})$$

Comparing Equation B.9 and Equation B.10, we can infer that $\nabla \log q_t(\mathbf{x}) \neq k\mathbf{s}_\theta(\mathbf{x}, t)$ for general distribution. One simple counterexample is where $p_0(\mathbf{x})$ is a mixture of Gaussians. By previous reasoning, CNS cannot generate $q_0(\mathbf{x})$ if the prior distribution is $q_T(\mathbf{x})$.

What if we allow initial samples drawn from distributions other than $q_T(\mathbf{x})$? We consider the special case where $p_0(\mathbf{x}) = \mathcal{N}(0, \mathbf{I})$, then $p_t(\mathbf{x}) = p_0(\mathbf{x})$, $q_t(\mathbf{x}) = q_0(\mathbf{x})$. The condition $\nabla \log q_t(\mathbf{x}) = k\mathbf{s}_\theta(\mathbf{x}, t)$ trivially holds true. By Lemma B.8, CNS (Equation B.5) and Equation B.7 are equivalent. Therefore, CNS can generate $q_0(\mathbf{x})$ if and only if the prior distribution at time T is the same as $q_T(\mathbf{x})$. For any other prior distribution, CNS would not be able to generate $q_0(\mathbf{x})$.

In conclusion, there does not exist a prior distribution $q'_T(\mathbf{x})$, from which CNS can always generate the temperature scaled distribution $q_0(\mathbf{x})$. $\square$

Appendix C

$\pi\mathbf{R}^2$: Additional Details

This appendix provides additional details for $\pi\mathbf{R}^2$ (Chapter 6): full implementation details for the simulation and real-world experiments, the per-batch training step and deployment-time inference loop as pseudocode, and the complete hyperparameter settings.

C.1 Implementation Details

C.1.1 Simulation

Task. The Leap Cube Reorientation task in MuJoCo Playground [122] requires a 16-DoF Leap Hand to orient a cube without dropping it from the palm. We modify the original environment in two ways: we raise the control rate from 20 Hz to 50 Hz to enable more reactive control, and we restrict the goal distribution to four *blue side up* yaw poses $\{0, \pi/2, \pi, 3\pi/2\}$ rather than uniformly random goals. An episode succeeds if the cube reaches any of the four goals within 0.2 rad in 600 steps; we report multi-success rate (MSR) over 100 episodes per condition.

Dataset. Four PPO experts (different seeds) generate 50 trajectories each, 200 total at 50 Hz. The simulator injects per-step sensing noise into the observation, sampled independently for each component: uniform noise in $[-0.05, 0.05]$ rad on joint angles, uniform noise in $[-0.02, 0.02]$ m on cube position, and Gaussian noise with scale 0.1 on the cube orientation quaternion (the quaternion is re-normalized after perturbation). The recorded demonstrations preserve this noise, so the training distribution already includes realistic sensing noise on both the proprioceptive and vision-derived components.

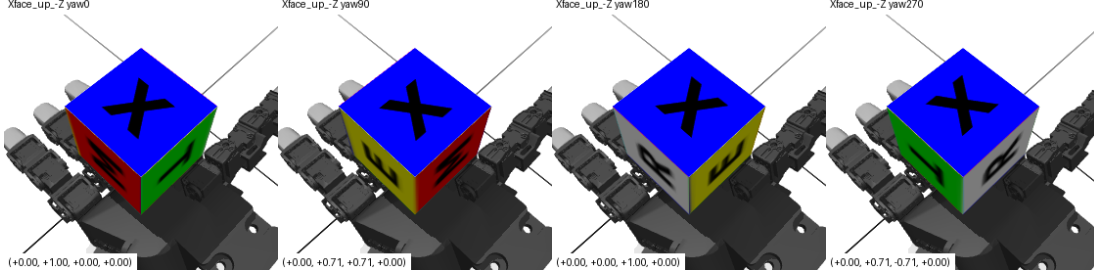


Figure C.1: **Leap Cube Reorientation goals.** The Leap Hand grasps a cube and must rotate it until the blue face points up at one of four target yaw poses ($0^\circ, 90^\circ, 180^\circ, 270^\circ$). The bottom caption shows the corresponding goal quaternions (w, x, y, z) .

Observation space. Each observation $\mathbf{o}_t$ concatenates:

- **Proprioception (32 dim):** noisy joint angles (16) and per-joint tracking error – noisy joint angle minus the most recent motor target (16).
- **Vision-derived cube pose (9 dim):** palm-to-cube position error (3) and absolute cube orientation in the world frame (6).

The 9-dim vision-derived subset stands in for VLM features per Section 6.1.2; asynchronous-vision/text evaluations apply staleness only to this subset.

Action space. 16-dim relative joint commands at 50 Hz. Each action $\mathbf{a}_t \in [-1, 1]^{16}$ is scaled by 0.5 and added to the current motor target.

Architecture. All variants share a Conditional U-Net 1D action head with chunk length $H=16$ and $n_{\text{obs}}=2$. The U-Net uses FiLM [89] modulation by the noise level τ at every residual block; $\pi\mathbf{R}^2$ replaces the shared FiLM projection with a per-position one – one (γ_p, β_p) pair per chunk position – analogous to the per-position AdaLN change described in Section 6.1.3 for DiT-based heads. The rest of the architecture is identical across all variants.

Training schedule. At each batch we sample $d \sim \text{Uniform}\{1, \dots, d_{\text{max}}\}$ with $d_{\text{max}}=5$, build $\boldsymbol{\tau}^{*,d}$, fill the front- d slots with ground-truth actions, and mask them out of the loss via $m_p = \mathbf{1}[p \geq d]$. With probability 0.2 we instead train on a standard flow schedule (single $\tau \sim \text{Uniform}[0, 1]$ shared across positions, no mask), so the same network can also denoise a full chunk from pure noise. The standard-flow branch supplies the inference pass that initializes the buffer at episode start (Section 6.1.3,

warm-up). For the zero-delay sync curve in Section 6.2.1, we additionally train a $d=0$ -only variant ($d_{\max}=0$).

Slow-subset delay in simulation. The simulation policies are purely state-based: no image or language input. The 9-dim vision-derived subset of the state vector (palm-to-cube position error and absolute cube orientation) plays the slow-channel role from Section 6.1.2. In this setting we find that neither delayed-visual-state sampling at training nor the learned delay embedding is necessary: the env already injects sensor noise on this subset at every step, so the policy is already tolerant to perturbed slow inputs out of the box.

For all rows of Section 6.2.1, a single $\pi\mathbf{R}^2$ checkpoint is used regardless of d_{vis} : the synchronous slow-channel rows ($d_{\text{vis}}=0$) and the asynchronous-vision/text rows ($d_{\text{vis}}>0$) share the same network, and evaluation simply feeds a d_{vis} -step-old version of the 9-dim subset at test time.

Evaluation protocol. At inference, $\pi\mathbf{R}^2$ performs one denoising step per policy call (each call outputs n_{act} actions). Train-time RTC and standard flow runs $K=15$ denoising steps to predict the whole chunk. Each (method, d) cell in Section 6.2.1 reports the mean multi-success rate over 100 episodes. Full sim hyperparameters are in Table C.1.

Baselines. All three variants share the dataset, optimizer, and training budget. $\pi\mathbf{R}^2$ and train-time RTC additionally share the per-batch delay-sampling protocol: each batch draws $d \in \{0, \dots, d_{\max}\}$ with $d_{\max}=5$ from an exponentially-decaying distribution $p(d=k) \propto e^{-\alpha_d k}$ with $\alpha_d=1$ (smaller d more likely following 12), and the front- d positions are clamped to ground-truth actions and masked out of the loss. The three variants then differ only in:

- **Flow:** no d sampling and no front clamp; a single shared noise level $\tau \sim \text{Uniform}[0, 1]$ is applied to the entire chunk. This is standard chunked flow matching, evaluated at NFE=15.
- **Train-time RTC** [12]: same per-batch d sampling as $\pi\mathbf{R}^2$, but the noise schedule applies a single shared noise level $\tau \sim \text{Uniform}[0, 1]$ to the back $H-d$ positions (no per-position structure). No standard-flow warm-up mixing.
- $\pi\mathbf{R}^2$: same per-batch d sampling as train-time RTC, plus the $\alpha=0.2$ standard-flow warm-up branch (with probability 0.2 the batch instead uses a single $\tau \sim$

Table C.1: **Simulation training and inference hyperparameters** for the Leap Cube Reorientation Conditional U-Net 1D policies. Values shown are $\pi\mathbf{R}^2$ defaults; the right column lists deviations used by baselines. Empty = baseline same as $\pi\mathbf{R}^2$.

Parameter	$\pi\mathbf{R}^2$	Baseline deviations
<i>Shared across all variants</i>		
Chunk length H	16	
Observation history n_{obs}	2	
Control rate	50 Hz	
Optimizer	AdamW ($\beta=[0.95, 0.999]$)	
Learning rate (peak)	1×10^{-4}	
LR schedule	cosine, 500-step warm-up	
Weight decay	1×10^{-6}	
Gradient-norm clip	1.0	
Batch size (global)	256	
GPUs	$1 \times \text{A5000}$	
Training budget	800 epochs	
<i>Method-specific (noise schedule + inference)</i>		
Train-time d_{max}	5	5 (Train-time RTC); n/a (Flow)
Standard-flow warm-up prob. α	0.2	n/a
Inference budget per call	1	15 (Train-time RTC, Flow)

Uniform $[0, 1]$ on all positions and no front clamp, so the same network can also initialize a buffer from pure noise at episode start). On the remaining 0.8 of batches we apply the three-region staircase $\tau^{*,d}$ of Section 6.1.3, and the U-Net uses per-position FiLM heads $\{(\gamma_p, \beta_p)\}_{p=0}^{H-1}$.

C.1.2 Real-World

Hardware and Workspace. A 6-DoF xArm6 manipulator carries a 12-DoF XHand at its wrist; a single overhead 640×480 RGB camera looks down on the workspace from above (Figure C.2). We fine-tune on $8 \times$ NVIDIA RTX A5000/A6000 (24/48 GB VRAM). Deployment uses a single RTX A5000 for the baselines and the $\pi\mathbf{R}^2$ without asynchronous slow-channel encoding; with asynchronous-vision/text variant $\pi\mathbf{R}^2$ uses $2 \times$ RTX A5000 so the slow VLM and the fast DiT action head do not share compute or memory bandwidth. Sharing one GPU between the two workers measurably inflates DiT latency and would conflate

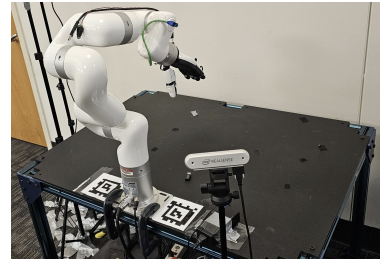


Figure C.2: **Real-world workspace.** xArm6 + XHand with a single overhead RGB camera.

the measured d .

Control rate. Predicted actions run closed-loop at 25 Hz at deployment ($T_{\text{ctrl}} \approx 40$ ms per tick). Teleoperation captures demonstrations at the same rate, bottlenecked by the camera and the XHand–workstation USB link.

Base model. We fine-tune NVIDIA GR00T-N1.7 [80]. The Qwen-VL backbone and Eagle-2 vision encoder remain *frozen*; only the action head – state/action projectors, the position embedding, and the DiT – is trained. The only architectural change to the action head replaces the DiT’s shared AdaLN modulation with the per-position (γ_p, β_p) heads of Section 6.1.3.

Observation space. Each call to the policy receives:

- **Proprioception** (45/51 dim): 6 xArm6 joint angles, 12 XHand joint angles (the actively-controlled finger DoFs), 12 corresponding per-joint torques, and 15 fingertip-force values (3-axis force from each of 5 sensorized fingertips). For *Insert Box*, we additionally provide xArm6 joint torques (6 dim) as input.
- **Vision and language:** one 640×480 RGB image from the overhead camera, encoded by Eagle-2 into vision tokens, concatenated with the task prompt (Table C.3) at the VLM backbone input.

Action space and low-level drivers. Each policy call emits a chunk of $H=50$ absolute joint-position targets (6 xArm6 + 12 XHand) at 25 Hz, spanning 2 s of motion. We send one target per control tick to each robot’s driver in position-control mode. The arms’ onboard firmware interpolates between consecutive commands at their internal servo rates (Table C.2).

Table C.2: **Low-level driver settings.** Both robots run in position-only mode at 25 Hz outbound from the policy; the onboard firmware interpolates between targets at its internal rate.

	xArm6	XHand
Driver mode	<code>mode=1</code> (streaming servo)	RS485, position mode
Command call	<code>set_servo_angle_j</code>	finger <code>set_target</code>
Bus / link	TCP	RS485
Onboard control	API (PID + interp.)	per-joint PID, $k_p=100$, $k_i=0$, $k_d=0$
Safety limits	collision sensitivity = 2	torque limit = 300

Data collection. A single operator teleoperates the robot to collect 200 demonstrations for *Don't Spill* and 300 for *Tidy Up Book*, randomizing the object's initial pose.

Table C.3: **Real-world task prompts.** Exact language input passed to GR00T-N1.7 at both training and deployment.

Task	Language prompt
Don't Spill	put the ball in the bowl and put them on the cutting board
Tidy Up Book	put the books in the basket
Insert Box	put the box in the basket

Training the asynchronous vision/text mechanism. The real-world $\pi\mathbf{R}^2$ training explicitly exercises the asynchronous-vision/text mechanism of Section 6.1.2. Each batch samples a vision delay $d_{\text{vis}} \sim \text{Uniform}\{0, \dots, d_{\text{vis}}^{\text{max}}\}$ with $d_{\text{vis}}^{\text{max}}=5$ ticks (200 ms at 25 Hz); the policy then receives the image recorded d_{vis} ticks earlier (i.e., the corresponding past frame in the demonstration), while proprioception stays current. For each batch, the integer $d_{\text{vis}} \in \{0, \dots, d_{\text{vis}}^{\text{max}}\}$ indexes a learned embedding $e(d_{\text{vis}})$ from a $(d_{\text{vis}}^{\text{max}}+1)$ -entry lookup table. The DiT adds $e(d_{\text{vis}})$ to its action-token features (broadcast across the H chunk positions), so the action head conditions on both the cached visual feature and its age. We zero-initialize the lookup table so that the untrained checkpoint reproduces the no-delay variant exactly. At deployment, the measured wall-clock vision latency passes through the same embedding before each DiT call.

Training hyperparameters. We fine-tune the GR00T-N1.7 action head (projectors + DiT) and keep the VLM backbone and vision encoder frozen. Full optimizer settings, batch size, and step budgets are in Table C.4. The per-task budgets (40,000 steps for *Don't Spill*, 28,000 for *Tidy Up Book*, 49,000 for *Insert Box*) correspond to approximately 200/100/100 epochs over each dataset. The per-position AdaLN parameters $\{(\gamma_p, \beta_p)\}_{p=0}^{H-1}$ initialize from the pretrained shared pair, so each head starts from a position-uniform schedule and gradually specializes during fine-tuning. All four rows in Table 6.1 fine-tune from the same GR00T-N1.7 checkpoint with identical data and budget; they differ only in the noise schedule and (for $\pi\mathbf{R}^2$) the AdaLN modification. The train-time RTC baseline samples its per-batch delay from $d \in \{0, \dots, 10\}$ (i.e., $d_{\text{max}}=10$) rather than the $d_{\text{max}}=5$ used in $\pi\mathbf{R}^2$, as $\pi\mathbf{R}^2$ is separating

the delay into two parts while RTC treats it as a whole. Full hyperparameters are in Table C.4.

Evaluation protocol. Each (method, task) cell runs $N=20$ trials with randomized initial object placement. Success rate (SR) counts whole-task completion within the episode time limit (30 seconds). The progress score (Prog) decomposes each task into sub-goals and reports the per-trial fraction completed:

- *Don't Spill* – 4 sub-goals: pick up ball, place ball in bowl, lift bowl off the table, place bowl on the cutting board.
- *Tidy Up Book* – 2 sub-goals: pull a book free from the pile, place it inside the basket.
- *Insert Box* – 4 sub-goals: push the box towards the wall, make it stand, grasp, and insert between books.

Deployment query modes. The four rows of Table 6.1 differ in how the action-head worker queries the policy:

- *Flow, Synchronous*: the main loop blocks until each call returns; the robot freezes during inference.
- *Flow, Naive Async, dense, ensemble*: the action worker queries back-to-back and combines overlapping chunks with the temporal-ensembling weights of Zhao et al. [130].
- *Train-time RTC* and $\pi\mathbf{R}^2$: the action worker queries back-to-back; the new chunk replaces the active one at the next swap boundary. The actions executed during policy call is given as input.
- $\pi\mathbf{R}^2$ w/ **async vision/text**: additionally spawns a VLM-cache worker on a separate GPU that refreshes the cached visual feature in the background.

C.2 Training and Inference Algorithms

We give pseudocode for the per-batch training step (Algorithm 1) and the deployment-time inference loop (Algorithm 2). The inference loop is the loop implemented in our deployment script; the wall-clock measurement that auto-derives d uses a rolling window of the most recent query times.

Table C.4: **Real-world training and inference hyperparameters** for the GR00T-N1.7 fine-tunes. Values shown are $\pi\mathbf{R}^2$ defaults; the right column lists deviations used by baselines. Empty = baseline same as $\pi\mathbf{R}^2$.

Parameter	$\pi\mathbf{R}^2$	Baseline deviations
<i>Shared across all variants</i>		
Chunk length H	50	
Observation history n_{obs}	1	
Control rate	25 Hz	
Optimizer	fused AdamW	
Learning rate (peak)	1×10^{-4}	
LR schedule	cosine, 5% warm-up	
Weight decay	1×10^{-5}	
Gradient-norm clip	1.0	
Batch size	512	
Precision	bf16 + tf32	
GPUs	8× A5000/A6000	
Training budget	40k / 28k / 49k steps (<i>Don't Spill / Tidy Up Book / Insert Box</i>)	
<i>Method-specific (noise schedule + delay + inference)</i>		
Train-time d_{max}	5	10 (Train-time RTC); n/a (Flow)
Train-time $d_{\text{vis}}^{\text{max}}$	5	n/a
Delay-embedding lookup size	6 entries $\rightarrow$ DiT hidden dim	n/a
Standard-flow warm-up prob. α	0.2	0 (Flow, Train-time RTC)
NFE per call	1 (one sub-chunk)	4 (Flow, Train-time RTC; full chunk)

Algorithm 1 $\pi\mathbf{R}^2$ training step. Each batch samples a delay d , builds the staircase $\boldsymbol{\tau}^{*,d}$, masks the front, applies jitter, and (for real-world training) additionally delays the slow channel.

Require: Action chunk $\mathbf{a}_{0:H-1}$, observation $(\mathbf{o}_t^{\text{fast}}, \mathbf{o}_{t:t-T_{\text{slow}}+1}^{\text{slow}})$, model v_θ

Require: Hyperparams $d_{\text{max}}, d_{\text{vis}}^{\text{max}}, j, \alpha$

```

1:  $u \sim \text{Uniform}[0, 1]$ 
2: if  $u < \alpha$  then                                      $\triangleright$  warm-up branch: standard flow
3:    $t \sim \text{Uniform}[0, 1]; \tau_p \leftarrow t, \forall p$ 
4:    $\mathbf{x}_{\tau,p} \leftarrow (1 - \tau_p)\boldsymbol{\epsilon}_p + \tau_p\mathbf{a}_p$ 
5:    $m_p \leftarrow 1, \forall p$                                 $\triangleright$  all positions contribute to loss
6: else                                                      $\triangleright$  staircase branch
7:    $d \sim \text{Uniform}\{1, \dots, d_{\text{max}}\}$ 
8:   Build  $\boldsymbol{\tau}^{*,d}$  as in Section 6.1.3                      $\triangleright$  three-region staircase
9:    $\delta_p \sim \text{Uniform}[-j, j]; \tau_p \leftarrow \text{clip}(\tau_p^{*,d} + \delta_p, 0, 1)$ 
10:   $\mathbf{x}_{\tau,p} \leftarrow (1 - \tau_p)\boldsymbol{\epsilon}_p + \tau_p\mathbf{a}_p$ 
11:   $\mathbf{x}_{\tau,p} \leftarrow \mathbf{a}_p$  if  $p < d$                       $\triangleright$  front  $d$  slots are ground-truth (inpaint conditioning)
12:   $m_p \leftarrow \mathbf{1}[p \geq d]$ 
13: end if
14:  $d_{\text{vis}} \sim \text{Uniform}\{0, \dots, d_{\text{vis}}^{\text{max}}\}$             $\triangleright$  slow-channel staleness (real-world only)
15:  $\mathbf{o}^{\text{slow}} \leftarrow \mathbf{o}_{t-d_{\text{vis}}}^{\text{slow}}; \text{append } e(d_{\text{vis}}) \text{ to slow representation}$ 
16:  $\hat{\mathbf{v}}_p \leftarrow v_\theta(\mathbf{x}_\tau, \boldsymbol{\tau}, \mathbf{o}_t^{\text{fast}}, \mathbf{o}^{\text{slow}})_p$ 
17:  $\mathcal{L} \leftarrow \sum_{p=0}^{H-1} m_p \|\hat{\mathbf{v}}_p - (\mathbf{a}_p - \boldsymbol{\epsilon}_p)\|^2$ 
18: Update  $\theta$  on  $\mathcal{L}$ 

```

Algorithm 2 $\pi\mathbf{R}^2$ inference loop on the robot. The action head runs continuously with fresh proprioception and a cached slow feature; the VLM runs asynchronously in a background thread; the per-call delay d is auto-derived from a rolling measurement of action-head wall-clock latency.

Require: Policy v_θ , observation source, robot controller, control period T_{ctrl}

Require: Rolling window W of recent query times (e.g. $W=20$)

```

1:  $\mathbf{C} \leftarrow \text{WarmStart}()$   $\triangleright$  Section 6.1.3: full standard-flow inference, denoise all  $H$ 
   positions
2:  $\mathbf{C} \leftarrow$  re-noise  $\mathbf{C}$  to match  $\boldsymbol{\tau}^{*,d_0}$  for initial estimate  $d_0$ 
3:  $i \leftarrow 0$   $\triangleright$  chunk index
4: queue  $Q \leftarrow \emptyset$   $\triangleright$  rolling latency window
5: Spawn VLM_Worker thread (continuously updates cached slow feature)
6: Spawn Action_Worker thread (continuously queries action head)
7: loop  $\triangleright$  one iteration per control tick (25 Hz)
8:    $t_0 \leftarrow$  current wall-clock time
9:   Read fresh proprioception  $\mathbf{o}_t^{\text{fast}}$  from robot sensors
10:  Publish  $\mathbf{o}_t^{\text{fast}}$  to shared state for workers
11:  if new chunk  $\mathbf{C}_{\text{new}}$  is available from Action_Worker then
12:     $\mathbf{C} \leftarrow \mathbf{C}_{\text{new}}$ ;  $i \leftarrow d$   $\triangleright$  swap and skip past in-flight front actions
13:  end if
14:  Send action  $\mathbf{C}_i$  to robot;  $i \leftarrow i + 1$ 
15:  Sleep until  $t_0 + T_{\text{ctrl}}$ 
16: end loop

17: procedure ACTION_WORKER  $\triangleright$  background thread; one denoising step per call
18:   loop
19:     Snapshot  $(\mathbf{o}_t^{\text{fast}}, t_{\text{state}})$  and the cached  $(\mathbf{o}_{\text{cached}}^{\text{slow}}, t_{\text{image}})$ 
20:      $d_{\text{vis}} \leftarrow \text{round}((t_{\text{state}} - t_{\text{image}})/T_{\text{ctrl}})$   $\triangleright$  age of cached slow feature in control
       ticks
21:      $d \leftarrow \max(1, \text{round}(\text{mean}(Q)/T_{\text{ctrl}}))$   $\triangleright$  auto-derived from rolling window
22:     Snapshot front- $d$  inpaint:  $\mathbf{a}_{0:d}^{\text{inflight}} \leftarrow \mathbf{C}_{i:i+d}$ 
23:      $q_0 \leftarrow$  wall-clock
24:      $\mathbf{C}_{\text{new}} \leftarrow v_\theta.\text{euler\_step}(\mathbf{a}_{0:d}^{\text{inflight}}, \mathbf{o}_t^{\text{fast}}, \mathbf{o}_{\text{cached}}^{\text{slow}}, e(d_{\text{vis}}))$   $\triangleright$  one NFE;
       per-position  $\Delta\tau_p$  as in Equation 6.4
25:     Push wall-clock  $-q_0$  into  $Q$ 
26:     Hand off  $\mathbf{C}_{\text{new}}$  to main loop
27:   end loop
28: end procedure

29: procedure VLM_WORKER  $\triangleright$  background thread; runs vision/text forward
   asynchronously
30:   loop
31:     Read fresh image and language;  $t_{\text{image}} \leftarrow$  image capture time
32:      $\mathbf{o}_{\text{cached}}^{\text{slow}} \leftarrow$  VLM forward (image, language)
33:     Atomically update cache with  $(\mathbf{o}_{\text{cached}}^{\text{slow}}, t_{\text{image}})$ 
34:   end loop
35: end procedure

```

Bibliography

- [1] Josh Abramson, Jonas Adler, Jack Dunger, Richard Evans, Tim Green, Alexander Pritzel, Olaf Ronneberger, Lindsay Willmore, Andrew J Ballard, Joshua Bambrick, et al. Accurate structure prediction of biomolecular interactions with alphafold 3. In *Nature*, 2024.
- [2] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736, 2022.
- [3] 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.
- [4] Shikhar Bahl, Abhinav Gupta, and Deepak Pathak. Human-to-robot imitation in the wild. In *RSS*, 2022.
- [5] Shikhar Bahl, Russell Mendonca, Lili Chen, Unnat Jain, and Deepak Pathak. Affordances from human videos as a versatile representation for robotics. In *CVPR*, 2023.
- [6] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025.
- [7] Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, Thomas Unterthiner, Daniel Keysers, Skanda Koppula, Fangyu Liu, Adam Grycner, Alexey Gritsenko, Neil Houlsby, Manoj Kumar, Keran Rong, Julian Eisenschlos, Rishabh Kabra, Matthias Bauer, Matko Bošnjak, Xi Chen, Matthias Minderer, Paul Voigtlaender, Ioana Bica, Ivana

- Balazevic, Joan Puigcerver, Pinelopi Papalampidi, Olivier Henaff, Xi Xiong, Radu Soricut, Jeremiah Harmsen, and Xiaohua Zhai. Paligemma: A versatile 3b vlm for transfer, 2024. URL <https://arxiv.org/abs/2407.07726>.
- [8] Homanga Bharadhwaj, Roozbeh Mottaghi, Abhinav Gupta, and Shubham Tulsiani. Track2act: Predicting point tracks from internet videos enables generalizable robot manipulation. In *ECCV*, 2024.
- [9] Homanga Bharadhwaj, Jay Vakil, Mohit Sharma, Abhinav Gupta, Shubham Tulsiani, and Vikash Kumar. Roboagent: Generalization and efficiency in robot manipulation via semantic augmentations and action chunking. In *ICRA*, 2024.
- [10] Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.
- [11] 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. In *RSS*, 2025.
- [12] Kevin Black, Allen Z Ren, Michael Equi, and Sergey Levine. Training-time action conditioning for efficient real-time chunking. *arXiv preprint arXiv:2512.05964*, 2025.
- [13] Kevin Black, Manuel Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. *Advances in Neural Information Processing Systems*, 38: 33383–33407, 2026.
- [14] 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. In *RSS*, 2023.
- [15] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

- [16] Boyuan Chen, Diego Martí Monsó, Yilun Du, Max Simchowitz, Russ Tedrake, and Vincent Sitzmann. Diffusion forcing: Next-token prediction meets full-sequence diffusion. *Advances in Neural Information Processing Systems*, 37: 24081–24125, 2024.
- [17] Sirui Chen, Chen Wang, Kaden Nguyen, Li Fei-Fei, and C Karen Liu. Arcap: Collecting high-quality human demonstrations for robot learning with augmented reality feedback. In *ICRA*, 2025.
- [18] Yuanpei Chen, Chen Wang, Yaodong Yang, and C Karen Liu. Object-centric dexterous manipulation from human motion data. In *CoRL*, 2024.
- [19] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. In *IJRR*, 2023.
- [20] Open X-Embodiment Collaboration, Abby O’Neill, Abdul Rehman, Abhinav Gupta, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, Albert Tung, Alex Bewley, Alex Herzog, Alex Irpan, Alexander Khazatsky, Anant Rai, Anchit Gupta, Andrew Wang, Andrey Kolobov, Anikait Singh, Animesh Garg, Aniruddha Kembhavi, Annie Xie, Anthony Brohan, Antonin Raffin, Archit Sharma, Arefeh Yavary, Arhan Jain, Ashwin Balakrishna, Ayzaan Wahid, Ben Burgess-Limerick, Beomjoon Kim, Bernhard Schölkopf, Blake Wulfe, Brian Ichter, Cewu Lu, Charles Xu, Charlotte Le, Chelsea Finn, Chen Wang, Chenfeng Xu, Cheng Chi, Chenguang Huang, Christine Chan, Christopher Agia, Chuer Pan, Chuyuan Fu, Coline Devin, Danfei Xu, Daniel Morton, Danny Driess, Daphne Chen, Deepak Pathak, Dhruv Shah, Dieter Buehler, Dinesh Jayaraman, Dmitry Kalashnikov, Dorsa Sadigh, Edward Johns, Ethan Foster, Fangchen Liu, Federico Ceola, Fei Xia, Feiyu Zhao, Felipe Vieira Frujeri, Freek Stulp, Gaoyue Zhou, Gaurav S. Sukhatme, Gautam Salhotra, Ge Yan, Gilbert Feng, Giulio Schiavi, Glen Berseth, Gregory Kahn, Guangwen Yang, Guanzhi Wang, Hao Su, Hao-Shu Fang, Haochen Shi, Henghui Bao, Heni Ben Amor, Henrik I Christensen, Hiroki Furuta, Homanga Bharadhwaj, Homer Walke, Hongjie Fang, Huy Ha, Igor Mordatch, Ilija Radosavovic, Isabel Leal, Jacky Liang, Jad Abou-Chakra, Jaehyung Kim, Jaimyn Drake, Jan Peters, Jan Schneider, Jasmine Hsu, Jay Vakil, Jeannette Bohg, Jeffrey Bingham, Jeffrey Wu, Jensen Gao, Jiaheng Hu, Jiajun Wu, Jialin Wu, Jiankai Sun, Jianlan Luo, Jiayuan Gu, Jie

Tan, Jihoon Oh, Jimmy Wu, Jingpei Lu, Jingyun Yang, Jitendra Malik, João Silvério, Joey Hejna, Jonathan Booher, Jonathan Tompson, Jonathan Yang, Jordi Salvador, Joseph J. Lim, Junhyek Han, Kaiyuan Wang, Kanishka Rao, Karl Pertsch, Karol Hausman, Keegan Go, Keerthana Gopalakrishnan, Ken Goldberg, Kendra Byrne, Kenneth Oslund, Kento Kawaharazuka, Kevin Black, Kevin Lin, Kevin Zhang, Kiana Ehsani, Kiran Lekkala, Kirsty Ellis, Krishan Rana, Krishnan Srinivasan, Kuan Fang, Kunal Pratap Singh, Kuo-Hao Zeng, Kyle Hatch, Kyle Hsu, Laurent Itti, Lawrence Yunliang Chen, Lerrel Pinto, Li Fei-Fei, Liam Tan, Linxi "Jim" Fan, Lionel Ott, Lisa Lee, Luca Weihs, Magnum Chen, Marion Lepert, Marius Memmel, Masayoshi Tomizuka, Masha Itkina, Mateo Guaman Castro, Max Spero, Maximilian Du, Michael Ahn, Michael C. Yip, Mingtong Zhang, Mingyu Ding, Minh Ho, Mohan Kumar Srirama, Mohit Sharma, Moo Jin Kim, Muhammad Zubair Irshad, Naoaki Kanazawa, Nicklas Hansen, Nicolas Heess, Nikhil J Joshi, Niko Suenderhauf, Ning Liu, Norman Di Palo, Nur Muhammad Mahi Shafiullah, Oier Mees, Oliver Kroemer, Osbert Bastani, Pannag R Sanketi, Patrick "Tree" Miller, Patrick Yin, Paul Wohlhart, Peng Xu, Peter David Fagan, Peter Mitrano, Pierre Sermanet, Pieter Abbeel, Priya Sundareshan, Qiuyu Chen, Quan Vuong, Rafael Rafailov, Ran Tian, Ria Doshi, Roberto Mart'in-Mart'in, Rohan Bajjal, Rosario Scalise, Rose Hendrix, Roy Lin, Runjia Qian, Ruohan Zhang, Russell Mendonca, Rutav Shah, Ryan Hoque, Ryan Julian, Samuel Bustamante, Sean Kirmani, Sergey Levine, Shan Lin, Sherry Moore, Shikhar Bahl, Shivin Dass, Shubham Sonawani, Shubham Tulsiani, Shuran Song, Sichun Xu, Siddhant Halder, Siddharth Karamcheti, Simeon Adebola, Simon Guist, Soroush Nasiriany, Stefan Schaal, Stefan Welker, Stephen Tian, Subramanian Ramamoorthy, Sudeep Dasari, Suneel Belkhale, Sungjae Park, Suraj Nair, Suvir Mirchandani, Takayuki Osa, Tanmay Gupta, Tatsuya Harada, Tatsuya Matsushima, Ted Xiao, Thomas Kollar, Tianhe Yu, Tianli Ding, Todor Davchev, Tony Z. Zhao, Travis Armstrong, Trevor Darrell, Trinity Chung, Vidhi Jain, Vikash Kumar, Vincent Vanhoucke, Vitor Guizilini, Wei Zhan, Wenxuan Zhou, Wolfram Burgard, Xi Chen, Xiangyu Chen, Xiaolong Wang, Xinghao Zhu, Xinyang Geng, Xiyuan Liu, Xu Liangwei, Xuanlin Li, Yansong Pang, Yao Lu, Yecheng Jason Ma, Yejin Kim, Yevgen Chebotar, Yifan Zhou, Yifeng Zhu, Yilin Wu, Ying Xu, Yixuan Wang, Yonatan Bisk, Yongqiang Dou, Yoonyoung Cho, Youngwoon Lee, Yuchen Cui, Yue Cao, Yueh-Hua Wu, Yujin Tang, Yuke Zhu, Yunchu Zhang, Yunfan Jiang, Yunshuang Li, Yunzhu Li, Yusuke Iwasawa, Yutaka Matsuo, Zehan Ma, Zhuo Xu, Zichen Jeff Cui, Zichen

- Zhang, Zipeng Fu, and Zipeng Lin. Open X-Embodiment: Robotic learning datasets and RT-X models. <https://arxiv.org/abs/2310.08864>, 2023.
- [21] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Sanja Fidler, Antonino Furnari, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, et al. Scaling egocentric vision: The epic-kitchens dataset. In *ECCV*, 2018.
- [22] Sudeep Dasari and Abhinav Gupta. Transformers for one-shot visual imitation. In *CoRL*, 2020.
- [23] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *CVPR*, 2023.
- [24] Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, Jiasen Lu, Taira Anderson, Erin Bransom, Kiana Ehsani, Huong Ngo, YenSung Chen, Ajay Patel, Mark Yatskar, Chris Callison-Burch, Andrew Head, Rose Hendrix, Favyen Bastani, Eli VanderBilt, Nathan Lambert, Yvonne Chou, Arnavi Chheda, Jenna Sparks, Sam Skjonsberg, Michael Schmitz, Aaron Sarnat, Byron Bischoff, Pete Walsh, Chris Newell, Piper Wolters, Tanmay Gupta, Kuo-Hao Zeng, Jon Borchardt, Dirk Groeneveld, Crystal Nam, Sophie Lebrecht, Caitlin Wittliff, Carissa Schoenick, Oscar Michel, Ranjay Krishna, Luca Weihs, Noah A. Smith, Hannaneh Hajishirzi, Ross Girshick, Ali Farhadi, and Aniruddha Kembhavi. Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models, 2024. URL <https://arxiv.org/abs/2409.17146>.
- [25] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. In *IEEE signal processing magazine*, 2012.
- [26] Carl Doersch, Ankush Gupta, Larisa Markeeva, Adrià Recasens, Lucas Smaira, Yusuf Aytar, João Carreira, Andrew Zisserman, and Yi Yang. Tap-vid: A benchmark for tracking any point in a video. In *NeurIPS*, 2022.
- [27] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

- [28] Yilun Du, Conor Durkan, Robin Strudel, Joshua B. Tenenbaum, Sander Dieleman, Rob Fergus, Jascha Sohl-Dickstein, Arnaud Doucet, and Will Grathwohl. Reduce, Reuse, Recycle: Compositional Generation with Energy-Based Diffusion Models and MCMC. In *ICML*, 2023.
- [29] Yiquan Duan, Xianda Guo, and Zheng Zhu. Diffusiondepth: Diffusion denoising approach for monocular depth estimation. In *ECCV*, 2024.
- [30] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, Kyle Lacey, Alex Goodwin, Yannik Marek, and Robin Rombach. Scaling Rectified Flow Transformers for High-Resolution Image Synthesis. In *ICML*, 2024.
- [31] Felix Faltings, Hannes Stark, Tommi Jaakkola, and Regina Barzilay. Protein fid: Improved evaluation of protein structure generative models. *arXiv preprint arXiv:2505.08041*, 2025.
- [32] Hao-Shu Fang, Hongjie Fang, Zhenyu Tang, Jirong Liu, Junbo Wang, Haoyi Zhu, and Cewu Lu. Rh20t: A robotic dataset for learning diverse skills in one-shot. In *ICRA*, 2024.
- [33] Zipeng Fu, Tony Z. Zhao, and Chelsea Finn. Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation, 2024. URL <https://arxiv.org/abs/2401.02117>.
- [34] Tomas Geffner, Kieran Didi, Zuobai Zhang, Danny Reidenbach, Zhonglin Cao, Jason Yim, Mario Geiger, Christian Dallago, Emine Kucukbenli, Arash Vahdat, et al. Proteina: Scaling flow-based protein structure generative models. In *ICLR*, 2025.
- [35] Kristen Grauman, Andrew Westbury, Eugene Byrne, Zachary Chavis, Antonino Furnari, Rohit Girdhar, Jackson Hamburger, Hao Jiang, Miao Liu, Xingyu Liu, et al. Ego4d: Around the world in 3,000 hours of egocentric video. In *CVPR*, 2022.
- [36] Kristen Grauman, Andrew Westbury, Lorenzo Torresani, Kris Kitani, Jitendra Malik, Triantafyllos Afouras, Kumar Ashutosh, Vijay Baiyya, Siddhant Bansal, Bikram Boote, et al. Ego-exo4d: Understanding skilled human activity from first-and third-person perspectives. In *CVPR*, 2024.

- [37] Irmak Guzey, Yinlong Dai, Georgy Savva, Raunaq Bhirangi, and Lerrel Pinto. Bridging the human to robot dexterity gap through object-oriented rewards. In *ICRA*, 2025.
- [38] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, 2022.
- [39] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, 2017.
- [40] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *stat*, 2015.
- [41] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS*, 2022.
- [42] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.
- [43] Sigmund H Høeg, Yilun Du, and Olav Egeland. Streaming diffusion policy: Fast policy synthesis with variable noise diffusion models. *arXiv preprint arXiv:2406.04806*, 2024.
- [44] Tsu-Ching Hsiao, Hao-Wei Chen, Hsuan-Kung Yang, and Chun-Yi Lee. Confronting ambiguity in 6d object pose estimation via score-based diffusion on se(3). In *CVPR*, 2024.
- [45] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. URL <https://arxiv.org/abs/2504.16054>.
- [46] Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo

- Fusai, et al. $\pi 0.5$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.
- [47] Vidhi Jain, Maria Attarian, Nikhil J Joshi, Ayzaan Wahid, Danny Driess, Quan Vuong, Pannag R Sanketi, Pierre Sermanet, Stefan Welker, Christine Chan, et al. Vid2robot: End-to-end video-conditioned policy learning with cross-attention transformers. In *RSS*, 2024.
- [48] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *CoRL*, 2022.
- [49] Sunshine Jiang, Xiaolin Fang, Nicholas Roy, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Siddharth Ancha. Streaming flow policy: Simplifying diffusion/flow-matching policies by treating action trajectories as flow trajectories, 2025. URL <https://arxiv.org/abs/2505.21851>.
- [50] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. In *Nature*, 2021.
- [51] Nikita Karaev, Ignacio Rocco, Benjamin Graham, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. Cotracker: It is better to track together. In *ECCV*, 2024.
- [52] Siddharth Karamcheti, Suraj Nair, Annie S Chen, Thomas Kollar, Chelsea Finn, Dorsa Sadigh, and Percy Liang. Language-driven representation learning for robotics. In *RSS*, 2023.
- [53] Tero Karras, Miika Aittala, Tuomas Kynkäänniemi, Jaakko Lehtinen, Timo Aila, and Samuli Laine. Guiding a diffusion model with a bad version of itself. *NeurIPS*, 2024.
- [54] Bingxin Ke, Anton Obukhov, Shengyu Huang, Nando Metzger, Rodrigo Caye Daudt, and Konrad Schindler. Repurposing diffusion-based image generators for monocular depth estimation. In *CVPR*, 2024.
- [55] Alexander Khazatsky, Karl Pertsch, Suraj Nair, Ashwin Balakrishna, Sudeep Dasari, Siddharth Karamcheti, Soroush Nasiriany, Mohan Kumar Srirama,

- Lawrence Yunliang Chen, Kirsty Ellis, et al. Droid: A large-scale in-the-wild robot manipulation dataset. In *RSS*, 2024.
- [56] 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.
- [57] 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. In *CoRL*, 2024.
- [58] Black Forest Labs. Flux. <https://github.com/black-forest-labs/flux>, 2024.
- [59] Adam Leach, Sebastian M Schmon, Matteo T. Degiacomi, and Chris G. Willcocks. Denoising diffusion probabilistic models on $SO(3)$ for rotational alignment. In *ICLR Workshops*, 2022.
- [60] Marion Lepert, Jiaying Fang, and Jeannette Bohg. Phantom: Training robots without robots using only human videos, 2025. URL <https://arxiv.org/abs/2503.00779>.
- [61] Jinhan Li, Yifeng Zhu, Yuqi Xie, Zhenyu Jiang, Mingyo Seo, Georgios Pavlakos, and Yuke Zhu. Okami: Teaching humanoid robots manipulation skills through single video imitation. In *CoRL*, 2024.
- [62] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.
- [63] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow Matching for Generative Modeling. In *ICLR*, 2023.
- [64] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. In *NeurIPS*, 2023.
- [65] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916, 2023.

- [66] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow. In *ICLR*, 2023.
- [67] Xueyi Liu, Kangbo Lyu, Jieqiong Zhang, Tao Du, and Li Yi. Quasisim: Parameterized quasi-physical simulators for dexterous manipulations transfer. In *ECCV*, 2024.
- [68] Xueyi Liu, Jianibieke Adalibieke, Qianwei Han, Yuzhe Qin, and Li Yi. Dextrack: Towards generalizable neural tracking control for dexterous manipulation from human references. In *ICLR*, 2025.
- [69] Yuxiang Lu, Zhe Liu, Xianzhe Fan, Zhenya Yang, Jinghua Hou, Junyi Li, Kaixin Ding, and Hengshuang Zhao. Faster: Rethinking real-time flow vlas, 2026. URL <https://arxiv.org/abs/2603.19199>.
- [70] Tyler Ga Wei Lum, Olivia Y Lee, C Karen Liu, and Jeannette Bohg. Crossing the human-robot embodiment gap with sim-to-real rl using one human demonstration. *arXiv preprint arXiv:2504.12609*, 2025.
- [71] Nanye Ma, Mark Goldstein, Michael S Albergo, Nicholas M Boffi, Eric Vanden-Eijnden, and Saining Xie. Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers. In *ECCV*, 2024.
- [72] Yecheng Jason Ma, Vikash Kumar, Amy Zhang, Osbert Bastani, and Dinesh Jayaraman. Liv: Language-image representations and rewards for robotic control. In *ICML*, 2023.
- [73] Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. Vip: Towards universal visual reward and representation via value-implicit pre-training. In *ICLR*, 2023.
- [74] Russell Mendonca, Shikhar Bahl, and Deepak Pathak. Structured world models from human videos. In *RSS*, 2023.
- [75] Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. Sedit: Guided image synthesis and editing with stochastic differential equations. In *ICLR*, 2022.
- [76] Kaichun Mo, Shilin Zhu, Angel X Chang, Li Yi, Subarna Tripathi, Leonidas J Guibas, and Hao Su. Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding. In *CVPR*, 2019.

- [77] Kieran A Murphy, Carlos Esteves, Varun Jampani, Srikumar Ramalingam, and Ameesh Makadia. Implicit-pdf: Non-parametric representation of probability distributions on the rotation manifold. In *ICML*, 2021.
- [78] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3m: A universal visual representation for robot manipulation. In *CoRL*, 2022.
- [79] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgb-d images. In *ECCV*, 2012.
- [80] NVIDIA, :, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llopart, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzheng Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. Gr00t n1: An open foundation model for generalist humanoid robots, 2025. URL <https://arxiv.org/abs/2503.14734>.
- [81] Chaoyi Pan, Giri Anantharaman, Nai-Chieh Huang, Claire Jin, Daniel Pfrommer, Chenyang Yuan, Frank Permenter, Guannan Qu, Nicholas Boffi, Guanya Shi, et al. Much ado about noising: Dispelling the myths of generative robotic control. *arXiv preprint arXiv:2512.01809*, 2025.
- [82] Georgios Papagiannis, Norman Di Palo, Pietro Vitiello, and Edward Johns. R+x: Retrieval and execution from everyday human videos. In *ICRA*, 2025.
- [83] Simone Parisi, Aravind Rajeswaran, Senthil Purushwalkam, and Abhinav Gupta. The unsurprising effectiveness of pre-trained vision models for control. In *ICML*, 2022.
- [84] Sungjae Park, Seungho Lee, Mingi Choi, Jiye Lee, Jeonghwan Kim, Jisoo Kim, and Hanbyul Joo. Learning to transfer human hand skills for robot manipulations. *arXiv preprint arXiv:2501.04169*, 2025.
- [85] Sungjae Park, Homanga Bharadhwaj, and Shubham Tulsiani. Demodiffusion: One-shot human imitation using pre-trained diffusion policy. In *International Conference on Robotics and Automation (ICRA)*, 2026.

- [86] Gaurav Parmar, Richard Zhang, and Jun-Yan Zhu. On aliased resizing and surprising subtleties in gan evaluation. In *CVPR*, 2022.
- [87] Georgios Pavlakos, Dandan Shan, Ilija Radosavovic, Angjoo Kanazawa, David Fouhey, and Jitendra Malik. Reconstructing hands in 3d with transformers. In *CVPR*, 2024.
- [88] Georgios Pavlakos, Dandan Shan, Ilija Radosavovic, Angjoo Kanazawa, David Fouhey, and Jitendra Malik. Reconstructing hands in 3d with transformers. In *CVPR*, 2024.
- [89] 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*, volume 32, 2018.
- [90] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.
- [91] Moritz Reuss, Maximilian Li, Xiaogang Jia, and Rudolf Lioutikov. Goal conditioned imitation learning using score-based diffusion policies. In *RSS*, 2023.
- [92] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.
- [93] Javier Romero, Dimitrios Tzionas, and Michael J Black. Embodied hands: Modeling and capturing hands and bodies together. In *Transactions on Graphics*, 2017.
- [94] Yu Rong, Takaaki Shiratori, and Hanbyul Joo. Frankmocap: A monocular 3d whole-body pose estimation system via regression and integration. In *CVPR*, 2021.
- [95] Saurabh Saxena, Abhishek Kar, Mohammad Norouzi, and David J Fleet. Monocular depth estimation using diffusion models. *arXiv preprint arXiv:2302.14816*, 2023.
- [96] Thomas Schops, Johannes L Schonberger, Silvano Galliani, Torsten Sattler, Konrad Schindler, Marc Pollefeys, and Andreas Geiger. A multi-view stereo

- benchmark with high-resolution images and multi-camera videos. In *CVPR*, 2017.
- [97] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. Laion-5b: An open large-scale dataset for training next generation image-text models. In *NeurIPS*, 2022.
- [98] Kenneth Shaw, Shikhar Bahl, and Deepak Pathak. Videodex: Learning dexterity from internet videos. In *CoRL*, 2023.
- [99] Junyao Shi, Zhuolun Zhao, Tianyou Wang, Ian Pedroza, Amy Luo, Jie Wang, Jason Ma, and Dinesh Jayaraman. Zeromimic: Distilling robotic manipulation skills from web videos. In *ICRA*, 2025.
- [100] Andy Shih, Dorsa Sadigh, and Stefano Ermon. Long horizon temperature scaling. In *ICML*, 2023.
- [101] Marta Skreta, Tara Akhound-Sadegh, Viktor Ohanesian, Roberto Bondesan, Alan Aspuru-Guzik, Arnaud Doucet, Rob Brekelmans, Alexander Tong, and Kirill Neklyudov. Feynman-kac correctors in diffusion: Annealing, guidance, and product of experts. In *Forty-second International Conference on Machine Learning*, 2025.
- [102] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *ICLR*, 2021.
- [103] Kiwhan Song, Boyuan Chen, Max Simchowitz, Yilun Du, Russ Tedrake, and Vincent Sitzmann. History-guided video diffusion. *arXiv preprint arXiv:2502.06764*, 2025.
- [104] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-Based Generative Modeling through Stochastic Differential Equations. In *ICLR*, 2021.
- [105] Mohan Kumar Srirama, Sudeep Dasari, Shikhar Bahl, and Abhinav Gupta. Hrp: Human affordances for robotic pre-training. In *RSS*, 2024.
- [106] Jiaming Tang, Yufei Sun, Yilong Zhao, Shang Yang, Yujun Lin, Zhuoyang Zhang, James Hou, Yao Lu, Zhijian Liu, and Song Han. Vlash: Real-time vlas

- via future-state-aware asynchronous inference. *arXiv preprint arXiv:2512.01031*, 2025.
- [107] Tony Tao, Mohan Kumar Srirama, Jason Jingzhou Liu, Kenneth Shaw, and Deepak Pathak. Dexwild: Dexterous human interactions for in-the-wild robot policies. *arXiv preprint arXiv:2505.07813*, 2025.
- [108] Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.
- [109] Eugene Valassakis, Georgios Papagiannis, Norman Di Palo, and Edward Johns. Demonstrate once, imitate immediately (dome): Learning visual servoing for one-shot imitation learning. In *IROS*, 2022.
- [110] Homer Rich Walke, Kevin Black, Tony Z Zhao, Quan Vuong, Chongyi Zheng, Philippe Hansen-Estruch, Andre Wang He, Vivek Myers, Moo Jin Kim, Max Du, et al. Bridgedata v2: A dataset for robot learning at scale. In *CoRL*, 2023.
- [111] Chen Wang, Haochen Shi, Weizhuo Wang, Ruohan Zhang, Li Fei-Fei, and C Karen Liu. Dexcap: Scalable and portable mocap data collection system for dexterous manipulation. In *RSS*, 2024.
- [112] Jianyuan Wang, Christian Rupprecht, and David Novotny. PoseDiffusion: Solving pose estimation via diffusion-aided bundle adjustment. In *ICCV*, 2023.
- [113] Kevin E Wu, Kevin K Yang, Rianne van den Berg, Sarah Alamdari, James Y Zou, Alex X Lu, and Ava P Amini. Protein structure generation via folding diffusion. In *Nature*, 2024.
- [114] Tete Xiao, Ilija Radosavovic, Trevor Darrell, and Jitendra Malik. Masked visual pre-training for motor control. *arXiv preprint arXiv:2203.06173*, 2022.
- [115] Haozhe Xie, Beichen Wen, Jiarui Zheng, Zhaoxi Chen, Fangzhou Hong, Haiwen Diao, and Ziwei Liu. Dynamicvla: A vision-language-action model for dynamic object manipulation. *arXiv preprint arXiv:2601.22153*, 2026.
- [116] Yanbo Xu, Yu Wu, Sungjae Park, Zhizhou Zhou, and Shubham Tulsiani. Temporal score rescaling for temperature sampling in diffusion and flow models. In *International Conference on Machine Learning (ICML)*, 2026.

- [117] Ge Yan, Jiyue Zhu, Yuquan Deng, Shiqi Yang, Ri-Zhao Qiu, Xuxin Cheng, Marius Memmel, Ranjay Krishna, Ankit Goyal, Xiaolong Wang, and Dieter Fox. Manifold: A general robot manipulation policy via consistency flow training, 2025. URL <https://arxiv.org/abs/2509.01819>.
- [118] Yufei Ye, Abhinav Gupta, and Shubham Tulsiani. What’s in your hands? 3d reconstruction of generic objects in hands. In *CVPR*, 2022.
- [119] Yufei Ye, Poorvi Hebbar, Abhinav Gupta, and Shubham Tulsiani. Diffusion-guided reconstruction of everyday hand-object interaction clips. In *ICCV*, 2023.
- [120] Jason Yim, Brian L Trippe, Valentin De Bortoli, Emile Mathieu, Arnaud Doucet, Regina Barzilay, and Tommi Jaakkola. Se (3) diffusion model with application to protein backbone generation. In *ICML*, 2023.
- [121] Chengbo Yuan, Chuan Wen, Tong Zhang, and Yang Gao. General flow as foundation affordance for scalable robot learning. *arXiv preprint arXiv:2401.11439*, 2024.
- [122] Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A Kahrs, et al. Mujoco playground. *arXiv preprint arXiv:2502.08844*, 2025.
- [123] Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. In *RSS*, 2024.
- [124] Fan Zhang and Michael Gienger. Affordance-based robot manipulation with flow matching. *arXiv preprint arXiv:2409.01083*, 2024.
- [125] Hui Zhang, Sammy Christen, Zicong Fan, Otmar Hilliges, and Jie Song. Grasppl: Generating grasping motions for diverse objects at scale. In *ECCV*, 2024.
- [126] Jason Y Zhang, Amy Lin, Moneish Kumar, Tzu-Hsuan Yang, Deva Ramanan, and Shubham Tulsiani. Cameras as rays: Pose estimation via ray diffusion. In *ICLR*, 2024.
- [127] 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, 2024. URL <https://arxiv.org/abs/2412.04987>.

- [128] Thomas T Zhang, Daniel Pfrommer, Chaoyi Pan, Nikolai Matni, and Max Simchowitz. Action chunking and exploratory data collection yield exponential improvements in behavior cloning for continuous control. *arXiv preprint arXiv:2507.09061*, 2025.
- [129] Xinyu Zhang and Abdeslam Boularias. One-shot imitation learning with invariance matching for robotic manipulation. In *RSS*, 2024.
- [130] 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.
- [131] Tony Z Zhao, Jonathan Tompson, Danny Driess, Pete Florence, Kamyar Ghasemipour, Chelsea Finn, and Ayzaan Wahid. Aloha unleashed: A simple recipe for robot dexterity. *arXiv preprint arXiv:2410.13126*, 2024.
- [132] Lingting Zhu, Guying Lin, Jinnan Chen, Xinjie Zhang, Zhenchao Jin, Zhao Wang, and Lequan Yu. Large images are gaussians: High-quality large image representation with levels of 2d gaussian splatting. In *AAAI*, 2025.
- [133] 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 *CoRL*, 2023.