

Appendix A Mathematical Foundations

A.1 Stochastic Interpolant Formulation

Definition 1 (Stochastic Interpolant). Given densities $\rho_0, \rho_1 : \mathbb{R}^d \rightarrow \mathbb{R}_{\geq 0}$, a stochastic interpolant is defined as:

$$x_t = I(t, x_0, x_1) + \gamma(t)z, \quad t \in [0, 1]$$

where $I(0, x_0, x_1) = x_0$, $I(1, x_0, x_1) = x_1$, $\gamma(0) = \gamma(1) = 0$, $\gamma(t) > 0$ for $t \in (0, 1)$, $(x_0, x_1) \sim \nu$ marginalizing to ρ_0 and ρ_1 , and $z \sim \mathcal{N}(0, I_d)$ independent of (x_0, x_1) .

This formulation induces a time-dependent density ρ_t with ρ_0 and ρ_1 as boundary conditions. The velocity field transporting ρ_t is:

$$b(t, x) = \mathbb{E}[\dot{x}_t | x_t = x] = \mathbb{E}[\partial_t I(t, x_0, x_1) + \dot{\gamma}(t)z | x_t = x]$$

A.1.1 Extension to Observation-Conditioned Policies

For robotic control applications, we extend this framework to incorporate conditioning on observations, which is essential for generating situation-appropriate actions.

Definition 2 (Observation-Conditioned Flow Policy). Given robot observations \mathbf{o} , we sample the target action from the conditional distribution $\mathbf{x}_* \sim p(\mathbf{x} | \mathbf{o})$. The interpolant becomes:

$$\mathbf{x}_t = \alpha_t \mathbf{x}_* + \sigma_t \varepsilon, \quad \alpha_0 = 1, \alpha_1 = 0, \sigma_0 = 0, \sigma_1 = 1$$

where $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is standard Gaussian noise independent of \mathbf{x}_* and \mathbf{o} .

The conditional velocity field is then defined as:

$$\mathbf{v}(\mathbf{x}, t, \mathbf{o}) = \mathbb{E}[\dot{\mathbf{x}}_t | \mathbf{x}_t = \mathbf{x}, \mathbf{o}] = \dot{\alpha}_t \mathbb{E}[\mathbf{x}_* | \mathbf{x}_t = \mathbf{x}, \mathbf{o}] + \dot{\sigma}_t \mathbb{E}[\varepsilon | \mathbf{x}_t = \mathbf{x}, \mathbf{o}]$$

This conditioning mechanism allows our policy to generate actions specifically tailored to the current robot state, a critical capability for effective control in complex environments. By learning this conditional velocity field, the robot can map from observations to appropriate action distributions that account for the current context.

A.2 Linear-Gaussian Bridge Representation

For practical implementation, we employ the linear-Gaussian bridge formulation:

$$\mathbf{x}_t = \alpha_t \mathbf{x}_* + \sigma_t \varepsilon, \quad \alpha_0 = 1, \alpha_1 = 0, \sigma_0 = 0, \sigma_1 = 1 \quad (5)$$

where \mathbf{x}_* represents the target action and $\varepsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. This formulation provides a specific instantiation of the stochastic interpolant with coefficients α_t and σ_t that satisfy certain boundary conditions.

A.3 Conversion Relationships for Prediction Targets

The stochastic interpolant framework enables conversion between different prediction targets, which is essential for our flexible inference approach.

Theorem 1 (Score-Velocity Relation). For a stochastic interpolant with coefficients α_t and σ_t , the velocity field $\mathbf{v}(\mathbf{x}, t, \mathbf{o})$ and score function $s(\mathbf{x}, t, \mathbf{o}) = \nabla_{\mathbf{x}} \log p_t(\mathbf{x} | \mathbf{o})$ are related by:

$$s(\mathbf{x}, t, \mathbf{o}) = \sigma_t^{-1} \frac{\alpha_t \mathbf{v}(\mathbf{x}, t, \mathbf{o}) - \dot{\alpha}_t \mathbf{x}}{\dot{\alpha}_t \sigma_t - \alpha_t \dot{\sigma}_t} \quad (6)$$

Theorem 2 (Reverse-time SDE). The reverse SDE corresponding to the conditional density $p_t(\mathbf{x} | \mathbf{o})$ is given by:

$$d\mathbf{X}_t = \left[\mathbf{v}(\mathbf{X}_t, t, \mathbf{o}) - \frac{1}{2} w_t s(\mathbf{X}_t, t, \mathbf{o}) \right] dt + \sqrt{w_t} d\bar{\mathbf{W}}_t \quad (7)$$

with different diffusion schedule w_t and reverse Wiener process $\bar{\mathbf{W}}_t$.

These relationships provide exact, differentiable mappings among velocity, score, and noise targets, allowing the policy to switch between training objectives or inference solvers without retraining.

391 A.4 Interpolation Path Formulations

392 We consider several interpolation paths that connect the noise distribution (\mathbf{x}_0) to the data manifold
393 (\mathbf{x}_1):

Definition 3 (Linear Interpolant).

$$\alpha_t = 1 - t, \quad \sigma_t = t$$

394 A straight path suited to well-behaved trajectories.

Definition 4 (Variance-Preserving (VP)).

$$\alpha_t = \exp\left(-\frac{1}{2} \int_0^t \beta_s \, ds\right), \quad \sigma_t = \sqrt{1 - \exp\left(-\int_0^t \beta_s \, ds\right)}$$

395 A diffusion schedule β_t that controls the rate of noise addition while maintaining the variance-
396 preserving property.

Definition 5 (Generalized VP (GVP)).

$$\alpha_t = \cos\left(\frac{\pi}{2}t\right), \quad \sigma_t = \sin\left(\frac{\pi}{2}t\right)$$

397 Trigonometric path with $\alpha_t^2 + \sigma_t^2 = 1$ for all t , yielding smoother gradients and better stability.

398 A.5 Numerical Solution Methods

399 The following numerical methods allow sampling from our stochastic interpolant models:

400 **Definition 6** (Probability Flow ODE). The probability flow ODE corresponding to the stochastic
401 interpolant is:

$$\frac{d}{dt} \mathbf{X}_t = b(t, \mathbf{X}_t | \mathbf{o}) \tag{8}$$

402 which can be solved using numerical integrators such as Euler or Heun methods.

403 **Definition 7** (Forward SDE). The forward SDE corresponding to the stochastic interpolant can be
404 derived from the reverse-time SDE and written as:

$$d\mathbf{X}_t = \mathbf{f}(t, \mathbf{X}_t, \mathbf{o})dt + \sqrt{w_t}d\mathbf{W}_t \tag{9}$$

405 where \mathbf{f} is the drift term that depends on time, current state, and observations, w_t is the same diffusion
406 schedule used in the reverse-time SDE, and \mathbf{W}_t is a standard Wiener process. This formulation
407 introduces controlled stochasticity that can be beneficial for exploration in certain robotic tasks.

Appendix B Implementation Details

This appendix provides key implementation details of our DA-SIP framework.

B.1 Stochastic Interpolant Policy Implementation

Table 6: Stochastic interpolant policy hyperparameters

Parameter	Value	Parameter	Value
Optimizer	AdamW	Weight decay	1e-6
Learning rate	1e-4	Schedule	Cosine decay
Batch size	256	Gradient clipping	1.0
Training epochs	5,000	Checkpointing	Every 50 epochs
Loss function	MSE	EMA rate	0.9999
Prediction targets	Velocity/Score/Noise	LR Scheduler	Cosine decay
Interpolants	Linear/VP/GVP	warmup steps	500

B.2 Lightweight CNN Classifier

Our CNN classifier uses three convolutional blocks followed by fully connected layers:

Table 7: Lightweight CNN architecture and training parameters

Architecture		Training Parameters	
Input	RGB image $32 \times 32 \times 3$	Optimizer	Adam
Conv1	32 filters, 3×3 , BatchNorm, ReLU	Learning rate	1e-3
MaxPool1	2×2 stride 2	Batch size	64
Conv2	64 filters, 3×3 , BatchNorm, ReLU	Epochs	100
MaxPool2	2×2 stride 2	Early stopping	patience=15
Conv3	128 filters, 3×3 , BatchNorm, ReLU	Loss function	Categorical cross-entropy
MaxPool3	2×2 stride 2	Class weighting	Inverse frequency
Flatten	2048 units	Validation split	20%
FC1	256 units, ReLU, Dropout(0.5)	Augmentation	Flip, rotation, color jitter
FC2	6 units, Softmax		

We apply histogram equalization per color channel and use weighted random sampling to handle class imbalance with:

$$\text{weights} = \frac{n_{\text{samples}}}{n_{\text{classes}} \times \text{class_counts}} \quad (10)$$

B.3 Zero-shot VLM Classification

We use few-shot in-context learning with the following configuration:

- **Models evaluated:** Qwen-VL 2.5, LLaVA-Next, Gemma 3
- **Method:** Present reference images with labels alongside target image
- **Prompt:** "simply pick the label from the sample images you see in prompt and tell me which one is closest to what you see"
- **Category mapping:**

i	initialize	s	stochastic exploration
n	near	e	end state
g	grab	c	continuous pushing

Table 8: VLM fine-tuning configuration

Model Configuration	Value	Training Parameters	Value
Base model	Qwen2.5-VL-7B-Instruct	Learning rate	2e-5
Quantization	8-bit, NF4 type	Batch size	32
Language LoRA rank	16	Epochs	12
Vision LoRA rank	8	Optimizer	AdamW
LoRA alpha	32	Weight decay	0.01
Vision LoRA alpha	16	Warmup ratio	3%
LoRA dropout	0.05	Precision	FP16
Target modules	q/k/v/o/gate/up/down proj	Gradient checkpointing	Enabled
Image resolution	224×224	Max sequence length	200 tokens

Table 9: Performance on the “Lift” task comparing DDPM, DDIM, and Linear (SI Policy with linear interpolation) under varying inference steps. We used Euler ODE approximator for trained models for SIP. **Note that this is without distillation for SIP**

Inference Steps	Diffusion Policy (DDPM)	Diffusion Policy (DDIM)	SI Policy (ours)
100	1.00	1.00	1.00
50	1.00	0.99	1.00
10	0.04	0.03	1.00
5	0.03	0.02	1.00
1	0.00	0.03	1.00

Table 10: Success rates on the “Can” manipulation task: Comparing conventional diffusion methods (DDPM, DDIM) with our SI Policy using linear interpolation across different inference step counts. All models employ Euler ODE approximation without distillation.

Inference Steps	Diffusion Policy (DDPM)	Diffusion Policy (DDIM)	SI Policy (ours)
100	1.00	1.00	1.00
50	0.97	0.98	1.00
10	0.00	0.00	1.00
1	0.00	0.00	1.00

Table 11: Representative **PushT** results with **VP** interpolation, using either **Euler** or **Heun**, in **ODE** or **SDE** mode, and varying numbers of steps. Performance (success rate) improves with more sophisticated solvers, showing that each configuration must be carefully tuned for optimal results.

Interpolation	# Steps	ODE / SDE	Approximator	Final Step	Success Rate
VP	10	ODE	Euler	Euler	0.786
VP	25	ODE	Euler	Euler	0.897
VP	50	ODE	Euler	Euler	0.912
VP	100	ODE	Euler	Euler	0.918
VP	100	SDE	Euler	Linear	0.918
VP	10	ODE	Heun	Tweedie	0.881
VP	25	ODE	Heun	Tweedie	0.916
VP	50	ODE	Heun	Tweedie	0.920
VP	100	ODE	Heun	Tweedie	0.922
VP	100	SDE	Heun	Tweedie	0.926
Comparison of Diffusion Policy					
Inference Steps	DP (DDPM)	DP (DDIM)	SI Policy (ours)		
10	0.125	0.810	0.881		
25	0.214	0.813	0.916		
50	0.801	0.812	0.920		
100	0.807	0.810	0.926		

Table 12: **BlockPush** task success rate with GVP interpolation, comparing different approximators (Euler vs. Heun), ODE vs. SDE approaches, and final step methods. Results show that Heun with Tweedie correction achieves the highest success rates, while ODE performance scales with step count. The bottom section provides a direct comparison with standard Diffusion Policy.

Interpolation	# Steps	ODE / SDE	Approximator	Final Step	Success Rate
GVP	1	ODE	Euler	-	0.026
GVP	10	ODE	Euler	-	0.155
GVP	25	ODE	Euler	-	0.199
GVP	50	ODE	Euler	-	0.195
GVP	100	ODE	Euler	-	0.200
GVP	100	SDE	Euler	Euler	0.204
GVP	100	SDE	Euler	Tweedie	0.205
GVP	100	SDE	Heun	Euler	0.215
GVP	100	SDE	Heun	Tweedie	0.227
Comparison with Diffusion Policy					
Method		Configuration			Success Rate
Diffusion Policy (DDPM)		100 steps			0.210
SI Policy (ours)		GVP, Heun, Tweedie, 100 steps			0.227

Table 13: Performance on the **Transport** task with VP interpolation, comparing different numbers of steps, ODE vs. SDE approaches, and approximators. Results show that performance is highly dependent on sufficient step count, with SDE approaches achieving slightly better performance.

Interpolation	# Steps	ODE / SDE	Approximator	Final Step	Success Rate
VP	10	ODE	Euler	-	0.006
VP	25	ODE	Euler	-	0.758
VP	50	ODE	Euler	-	0.796
VP	100	ODE	Euler	-	0.784
VP	100	SDE	Euler	Euler	0.850
VP	100	SDE	Heun	Tweedie	0.850
Comparison with Diffusion Policy					
Diffusion Policy (DDPM)			SI Policy (ours)		
0.852			0.850		

Table 14: Performance on the **Square** task with VP interpolation, comparing different numbers of steps, ODE vs. SDE approaches, and approximators.

Interpolation	# Steps	ODE / SDE	Approximator	Final Step	Success Rate
VP	10	ODE	Euler	–	0.892
VP	25	ODE	Euler	–	0.950
VP	50	ODE	Euler	–	0.942
VP	100	ODE	Euler	–	0.926
VP	100	SDE	Euler	Euler	0.952
Comparison with Diffusion Policy					
Inference Steps		DP (DDPM)	DP (DDIM)	SI Policy (ours)	
10		0.000	0.928	0.892	
50		0.916	0.943	0.942	
100		0.944	0.962	0.952	

Table 15: Tool Hang performance with VP interpolation across different configurations (averaged across 3 seeds)

Interpolation	# Steps	ODE/SDE	Solver	Success Rate
VP	1	ODE	Euler	0.000
VP	10	ODE	Euler	0.115
VP	25	ODE	Euler	0.345
VP	50	ODE	Euler	0.381
VP	100	ODE	Euler	0.351
VP	10	ODE	Heun	0.342
VP	25	ODE	Heun	0.330
VP	50	ODE	Heun	0.371
VP	100	ODE	Heun	0.364
VP	100	SDE	Euler	0.357
VP	100	SDE	Heun	0.368
Comparison with Diffusion Policy				
Step Count	DP (DDPM)	DP (DDIM)	SI Policy (ours)	
1	0.000	0.482	0.000	
10	0.000	0.484	0.342	
25	0.000	0.486	0.371	
50	0.534	0.490	0.381	
100	0.484	0.480	0.368	

Note: Surprisingly, Diffusion Policy methods perform much better on Tool Hang than SI Policy consistently. DDPM achieves the highest overall success rate (0.534) at 50 steps, while DDIM shows remarkable consistency across all step counts, maintaining 0.48 success even at very low step counts.

Appendix D Data Collection and Annotation

This appendix details our methodology for collecting and annotating the dataset of 20,000 labeled robot states used for training and evaluating our difficulty classification models.

We developed a systematic approach to collect data of robot states across various manipulation tasks. Our data collection protocol was as follows:

D.1 Episode Recording and Frame Extraction

We recorded complete episodes across six simulation environments (Can, Lift, Push T, Square, Tool Hang, and Transport). We extracted frames at regular intervals (every 5th frame) with additional adaptive sampling to ensure representation of critical states (e.g., precise grasping moments). This approach yielded approximately 20,000 frames for annotation.

D.2 Annotation Process

Eight volunteers participated in the annotation process. Each annotator received tutorial videos for how to annotate each image for each task. A custom web-based annotation platform was used to help them annotate and each volunteer is given a distinct username.

D.3 Difficulty Categories

The following categories were used to classify robot states based on computational requirements:

1. **Initial (I)**: The robot is positioned away from targets and objects, performing gross positioning movements in free space. No precise control is needed.
2. **Near (N)**: The robot is approaching within approximately 10cm of a target object but has not yet initiated contact or grasping. Some care in motion planning is required.
3. **Grabbing (G)**: The robot is in the process of grasping an object, or has grasped an object and is moving it to a new location. Moderate precision is required.
4. **Stochastic (S)**: The robot is attempting a precise placement or alignment task that requires controlled variability (e.g., inserting a tool, threading a needle). High precision with some exploration is needed.
5. **Continuous (C)**: The robot is pushing or manipulating an object without grasping, requiring continuous fine control with millimeter precision (e.g., pushing a block along a specific path).
6. **End (E)**: Task objectives have been achieved, and the robot is in a terminal state or moving away from completed objectives.

D.4 Dataset Finalization

To ensure reliability, each state was labeled by multiple annotators. For the final dataset, we assigned category labels using majority voting. In cases of ties or significant disagreement, labels were adjudicated by a senior robotics researcher.

The most common boundary cases occurred between Near (N) and Grabbing (G) categories

These patterns reflect the continuous nature of difficulty transitions during task execution, with some states occupying boundary regions between categories.

The final dataset contains approximately 20,000 labeled states spanning all six difficulty categories across the different simulation environments, providing a comprehensive foundation for training our difficulty classifiers.