

Deep Parameter Interpolation for Scalar Conditioning

Chicago Y. Park¹ Michael T. McCann² Cristina Garcia-Cardona²
 Brendt Wohlberg² Ulugbek S. Kamilov¹

¹University of Wisconsin–Madison ²Los Alamos National Laboratory

mccann@lanl.gov

Abstract

We propose deep parameter interpolation (DPI), a general-purpose method for transforming an existing deep neural network architecture into one that accepts an additional scalar input. Recent deep generative models, including diffusion models and flow matching, employ a single neural network to learn a time- or noise level-dependent vector field. Designing a network architecture to accurately represent this vector field is challenging because the network must integrate information from two different sources: a high-dimensional vector (usually an image) and a scalar. Common approaches either encode the scalar as an additional image input or combine scalar and vector information in specific network components, which restricts architecture choices. Instead, we propose to maintain two learnable parameter sets within a single network and to introduce the scalar dependency by dynamically interpolating between the parameter sets based on the scalar value during training and sampling. DPI is a simple, architecture-agnostic method for adding scalar dependence to a neural network. We demonstrate that our method improves denoising performance and enhances sample quality for both diffusion and flow matching models, while achieving computational efficiency comparable to standard scalar conditioning techniques. Code is available at github.com/uw-cig/parameter.interpolation.

1. Introduction

Generative modeling has made significant progress through the development of diffusion models [7, 23] and flow matching methods [1, 10, 11]. Both frameworks generate complex data by progressively transforming samples from tractable source distributions — such as Gaussian noise — into structured samples. Diffusion models achieve this through iterative denoising guided by learned score functions, typically

formulated as stochastic differential equations (SDEs). In contrast, flow matching models learn deterministic dynamics along probability paths, often formulated as ordinary differential equations (ODEs), to continuously transport samples from a source distribution toward the data distribution.

To model this progressive transformation, diffusion and flow matching frameworks typically employ a single neural network that operates across all sampling steps [7, 10, 17, 21, 23]. This network is trained to represent a step-dependent vector field that varies with a scalar variable—interpreted as time / noise level in diffusion models or as time in flow matching formulations. Because the network architecture itself is not inherently aware of this scalar, conditioning mechanisms are introduced to provide it with the corresponding value at each step. Existing approaches can be grouped into three categories: (1) *embedding-based conditioning*, which encodes the scalar variable into a learned embedding and injects it through dedicated conditioning modules, typically by adding it to the feature map [7, 23], or by modulating normalization layers [2, 3, 5, 15, 18]; (2) *input-level conditioning* [10, 26], which augments inputs with constant-valued maps indicating the current step; and (3) *external rescaling* [22], formulated for score-based diffusion models but not for flow matching, conditions the network by rescaling the predicted scores based on their known magnitude without modifying the network.

While each category provides a means for conditioning, they have distinct limitations. (1) *embedding-based conditioning* has shown strong performance but relies on specialized architectural modules, such as normalization-based conditioning blocks, which constrain network design and require careful alignment of feature dimensionalities across normalization layers. In contrast, (2) *input-level conditioning* and (3) *external rescaling* are architecture-agnostic, but our experimental results indicate that they offer reduced expressiveness relative to embedding-based conditioning in denoising and unconditional image synthesis tasks. Moreover, (3) is less straightforward to extend to flow matching

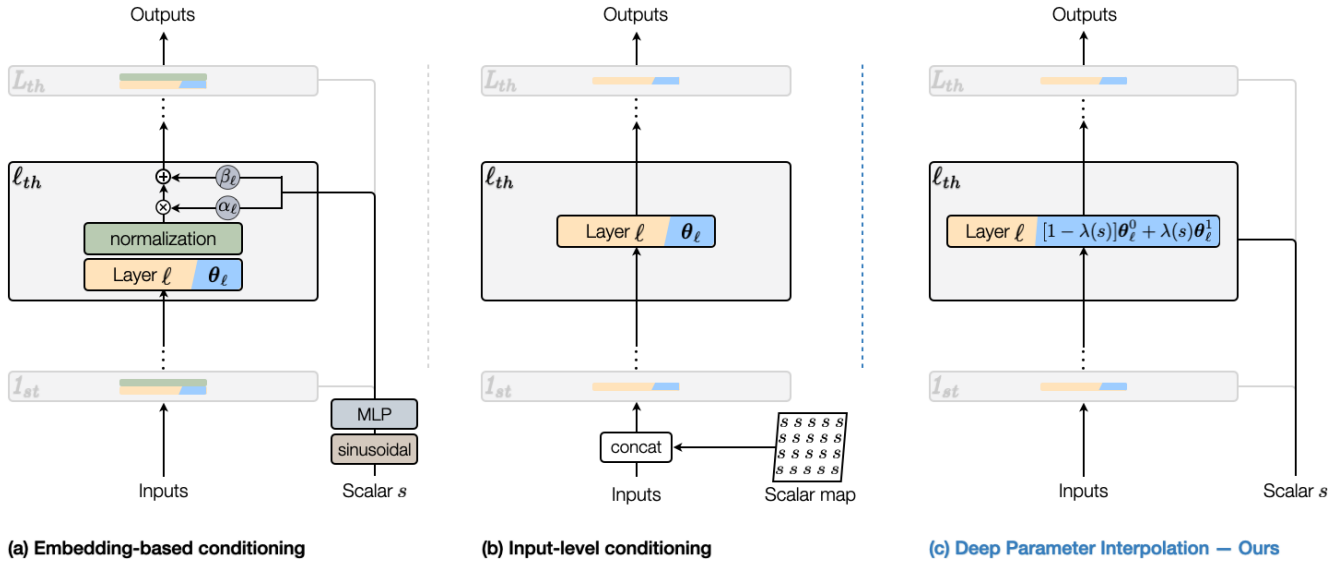


Figure 1. Comparison of scalar conditioning mechanisms for diffusion and flow matching models. Each method integrates a scalar variable s (e.g., time or noise level) into the network, where learnable modules at layer ℓ are parameterized by θ_ℓ . (a) **Embedding-based conditioning** injects a sinusoidal embedding of s through an MLP to modulate features; effective but *architecturally constrained*, as it requires conditioning modules (e.g., normalization layers). (b) **Input-level conditioning** concatenates a constant-valued scalar map with the input tensor; simple and architecture-agnostic but *less expressive*. (c) **Deep parameter interpolation — ours** maintains two learnable parameter sets, θ^0 and θ^1 , and introduces scalar dependency at the parameter level by interpolating between them based on the scalar value s , where a learnable monotonic function $\lambda(s) \in [0, 1]$ controls the interpolation to enable smooth adaptation across scalar values.

since it is designed specifically for score-based diffusion frameworks. These trade-offs highlight the need for a conditioning strategy that is both architecture-independent and expressive across generative modeling frameworks.

We introduce deep parameter interpolation (DPI), a general-purpose method for conditioning neural networks on scalar variables such as time or noise level. Instead of injecting the scalar through input channels or requiring a specific architectural requirement to accept a scalar variable, we maintain two learnable parameter sets within a single network and dynamically interpolate between them according to the scalar value, introducing scalar awareness at the parameter level without altering the architecture. Empirically, DPI achieves consistent improvements in denoising accuracy and sample quality over existing conditioning mechanisms, while achieving computational efficiency comparable to standard scalar conditioning techniques.

2. Background

Modern generative modeling frameworks rely on parameterized stochastic or deterministic processes that progressively transform noise into structured data. In this section, we introduce the score-based stochastic process and the flow-based deterministic process, along with the scalar conditioning methods used in both.

2.1. Score-Based Diffusion Models

Score-based diffusion models [7, 17, 21, 23] learn the gradient of the log-density (score function) using neural networks. Tweedie’s formula [4] connects the score function to the minimum mean squared error (MMSE) denoiser, enabling estimation of the score from noisy observations alone. Specifically, for noisy images $\mathbf{x}_t = \mathbf{x} + \sigma_t \mathbf{n}$, where \mathbf{x} is clean image, $\mathbf{n} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and σ_t denotes the noise level, the score can be approximated as

$$\nabla \log p_{\sigma_t}(\mathbf{x}_t) \approx \frac{D_\theta(\mathbf{x}_t; \sigma_t) - \mathbf{x}_t}{\sigma_t^2},$$

where D_θ is a denoising network trained to minimize the mean squared error (MSE):

$$\mathbb{E}_{\mathbf{x}, \mathbf{n}, t} \left[\|D_\theta(\mathbf{x} + \sigma_t \mathbf{n}; \sigma_t) - \mathbf{x}\|_2^2 \right].$$

The scalar σ_t (equivalently represented by the time variable t [16]) thus acts as a conditioning signal that determines the current noise level during both training and sampling.

This formulation enables the denoiser to estimate the score function at various scalar noise levels, which makes it applicable for reverse-time sampling using stochastic processes [13, 19, 24]. The sampling process follows a stochastic random walk [7, 17, 23], where each step involves moving the current state in the direction suggested by the estimated score function, combined with random noise.

2.2. Flow Matching Generative Models

Flow matching models [1, 10, 11] generate samples by learning continuous-time dynamics that transport samples from a source distribution toward the data distribution. Instead of estimating the score function, these models predict the velocity field $\mathbf{v}_\theta(\mathbf{x}_t; t)$ that describes how \mathbf{x}_t evolves along a predefined probability path:

$$\frac{d\mathbf{x}_t}{dt} = \mathbf{v}_\theta(\mathbf{x}_t; t).$$

A common setup adopts an affine probability path [10], where \mathbf{x}_t is constructed as

$$\mathbf{x}_t = \beta_t \mathbf{x}_1 + \gamma_t \mathbf{x}_0, \quad (1)$$

with \mathbf{x}_1 as a clean data sample, $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ as Gaussian noise, and β_t, γ_t controlling the interpolation over time. Differentiating this path gives the target velocity:

$$\mathbf{v}(\mathbf{x}_t; t) = \frac{d\beta_t}{dt} \mathbf{x}_1 + \frac{d\gamma_t}{dt} \mathbf{x}_0.$$

The network \mathbf{v}_θ is trained to predict the target velocity by minimizing the MSE:

$$\mathbb{E}_{\mathbf{x}_1, \mathbf{x}_0, t} \left[\|\mathbf{v}_\theta(\mathbf{x}_t; t) - \mathbf{v}(\mathbf{x}_t; t)\|_2^2 \right].$$

The scalar variable t thus defines both the interpolation along the probability path and the conditioning context under which the velocity field is predicted.

This formulation enables the flow matching model to estimate the velocity field along the probability path, making it applicable for generative sampling. The sampling process evolves the state by following the learned velocity direction over continuous time, typically implemented using numerical ODE solvers.

2.3. Scalar Conditioning in Generative Models

Both score-based diffusion and flow matching models employ a single neural network that operates across a range of scalar conditions — the noise level $\sigma_t \in [\sigma_1, \sigma_T]$ in diffusion models and the time step $t \in [0, 1]$ in flow matching. To make this possible, various conditioning mechanisms have been proposed to inform the network of the current step.

A common approach conditions the network by passing a sinusoidal embedding of the scalar variable s through a multi-layer perceptron (MLP) φ to produce modulation parameters

$$[a_\ell(s), b_\ell(s)] = \varphi_\ell(\text{sinusoidal}(s)),$$

where $a_\ell(s)$ and $b_\ell(s)$ are scale and shift coefficients applied at layer ℓ . These parameters modulate the normalized activations as

$$h_{\ell+1} = a_\ell(s) \odot \text{normalization}(h_\ell) + b_\ell(s),$$

where h_ℓ denotes the feature map at layer ℓ , and \odot represents element-wise multiplication. While effective and popular, this design assumes the presence of normalization layers and ties the conditioning MLP to the channel dimensionality of each feature map, making it nontrivial to adapt the method to arbitrary network architectures.

Less constrained methods avoid architectural modifications by applying conditioning externally rather than through dedicated embedding modules. A simple approach applies conditioning at the input level by concatenating a constant-valued map that encodes the conditioning scalar s as the input tensor, $\mathbf{x}' = \text{concatenate}(\mathbf{x}, s \cdot \mathbf{1}_{H \times W})$, where $\mathbf{x} \in \mathbb{R}^{C \times H \times W}$ denotes the original input, and the constant map $s \cdot \mathbf{1}_{H \times W}$ adds one channel encoding the scalar value. This formulation requires no architectural changes and is compatible with a wide range of backbones, though in practice we observe that its influence on feature representations tends to be relatively weak compared to embedding-based conditioning.

An alternative strategy, introduced in noise conditional score networks (NCSNv2) [22], removes explicit conditioning altogether by rescaling the predicted score $s_\theta(\mathbf{x}; \sigma)$ according to its magnitude with respect to the noise level σ . Empirically, the norm of the learned score satisfies $\|s_\theta(\mathbf{x}; \sigma)\|_2 \propto 1/\sigma$, which motivates the NCSNv2 parameterization

$$s_\theta(\mathbf{x}; \sigma) = \frac{1}{\sigma} \tilde{s}_\theta(\mathbf{x}),$$

where $\tilde{s}_\theta(\mathbf{x})$ denotes an unconditional network output. This formulation entirely removes architectural coupling; however, extending it to flow matching models remains nontrivial since they do not explicitly estimate scores, and our experiments further indicate that its conditioning capacity is limited.

3. DPI: Deep Parameter Interpolation

We propose *deep parameter interpolation (DPI)*, a simple yet effective strategy for conditioning neural networks on a scalar variable s — such as time or noise level — directly at the parameter level. Instead of injecting scalar information through embeddings or input augmentation, our approach makes the model’s learnable parameters themselves vary smoothly with s .

To achieve this, DPI maintains two sets of learnable parameters for each learnable module (e.g., convolutional or linear layer) within a single network. It introduces scalar dependency at the parameter level by interpolating between these two parameter sets according to the scalar value. We design the interpolation through a learnable monotonic function such that the interpolated parameter set is identical to the first parameter set at $s = s_{\min}$, identical to the second set at $s = s_{\max}$, and a linear combination of both for intermediate



Figure 2. Examples of FFHQ 64×64 generated samples by DRUNet [26] and ADM [3] using diffusion SDE and flow ODE solvers with our deep parameter interpolation (DPI). FID scores are reported in parentheses.

values $s \in (s_{\min}, s_{\max})$, enabling smooth network behavior across scalar values. This parameter-level conditioning preserves the original architecture while adding negligible computational overhead, making it broadly applicable across diffusion, flow matching, and other scalar-conditioned frameworks with architectural flexibility.

Parameter interpolation in neural networks has been explored in other contexts, e.g., in [25], which interpolates the parameters of separately trained networks for different objectives—one optimized for perceptual quality and another for distortion minimization. Our approach is distinct in that the interpolation occurs within a single model during training, enabling scalar-conditioned adaptation without relying on multiple pre-trained networks.

3.1. Interpolation of Learnable Modules for Scalar Conditioning

Our key idea is that the vector field we aim to approximate varies smoothly with the scalar variable s . To achieve this in the neural network, we enforce smoothness directly at the parameter level by letting parameters of each layer change smoothly with s instead of keeping them fixed.

Concretely, let the base (i.e., not scalar-conditioned) network be defined as a function $f : \mathbb{R}^p \times \mathbb{R}^m \rightarrow \mathbb{R}^n; (\theta, \mathbf{x}) \mapsto$

$f(\theta, \mathbf{x})$, where $\theta \in \mathbb{R}^p$ are the network parameters (e.g., weights and biases) and \mathbf{x} is the input (usually an image). The DPI version of f is a function $g : \mathbb{R}^p \times \mathbb{R}^p \times \mathbb{R}^m \times \mathbb{R} \rightarrow \mathbb{R}^n$ defined by

$$g(\theta^0, \theta^1, \mathbf{x}, s) = f([1 - \lambda(s)]\theta^0 + \lambda(s)\theta^1, \mathbf{x}), \quad (2)$$

where $\lambda : \mathbb{R} \rightarrow \mathbb{R}$ is a monotonically increasing function such that $\lambda(s_{\min}) = 0$ and $\lambda(s_{\max}) = 1$.

As a result of this design, $g(\theta^0, \theta^1, \mathbf{x}, s_{\min}) = f(\theta^0, \mathbf{x})$ and $g(\theta^0, \theta^1, \mathbf{x}, s_{\max}) = f(\theta^1, \mathbf{x})$; for intermediate values of s , the two parameter sets are smoothly blended by $\lambda(s)$. This parameter level interpolation allows the network to adapt continuously across scalar values, introducing scalar awareness without modifying the underlying architecture (see Figure 1).

Although DPI maintains two learnable sets within a single network, its GPU memory and computational overhead remain comparable to a standard single-set network. Interpolation involves only a lightweight element-wise combination of the two parameter sets, after which the model operates with a single set of interpolated weights. As a result, the method achieves higher efficiency in GPU memory usage and comparable computation to embedding-based conditioning strategy such as MLP-based conditioning. Further

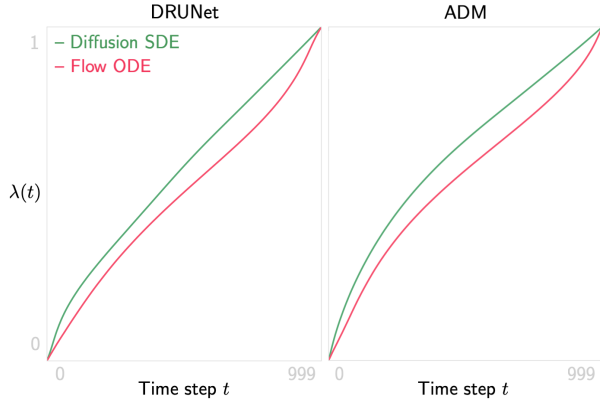


Figure 3. Learned interpolation functions λ for DRUNet and ADM under both diffusion and flow matching frameworks. The monotonic function λ in Section 3.2 defines how the model interpolates two parameter sets within a single network as the scalar variable (time or noise level) progresses. Differences in λ shapes indicate architecture- and framework-specific adaptation behavior.

analysis is provided in Section 4.4.

3.2. Learnable Monotonic Interpolation Function

The interpolation function $\lambda(s)$ determines how the network transitions as the scalar variable progresses from s_{\min} to s_{\max} . It is defined to be monotonic and to satisfy

$$\lambda(s_{\min}) = 0, \quad \lambda(s_{\max}) = 1,$$

ensuring that the model initially relies on the first parameter set at s_{\min} and gradually transitions to the s_{\max} as s increases.

We aim to make $\lambda(s)$ learnable while strictly enforcing its monotonicity across the scalar range. To achieve this, we design $\lambda(s)$ as a normalized cumulative distribution over a set of discrete scalar steps. This formulation allows flexible learning of the transition shape while guaranteeing that $\lambda(s)$ increases monotonically from 0 to 1.

Concretely, we introduce a learnable vector $\phi \in \mathbb{R}^S$, where S denotes the number of discrete scalar values (e.g., total number of timesteps or noise levels). We compute a softmax function over ϕ :

$$p_i = \frac{\exp(\phi_i)}{\sum_{j=1}^S \exp(\phi_j)},$$

where the softmax ensures that all p_i are positive and sum to one.

We then define the interpolation function as the cumulative sum:

$$\lambda_\phi(s_i) = \sum_{j=1}^i p_j.$$

This formulation guarantees $\lambda_\phi(s_i) \in [0, 1]$ and enforces strict monotonicity across the scalar range.

By learning this interpolation, the model continuously adapts its internal behavior across scalar values, effectively balancing denoising in diffusion models and velocity estimation in flow matching, without requiring explicit conditional inputs.

4. Numerical Evaluations

We evaluate the proposed parameter interpolation in both diffusion and flow matching generative frameworks. Our objectives are to (1) improve the denoising accuracy of diffusion models across a wide range of scalar conditions and (2) enhance unconditional image generation quality for both diffusion and flow matching models with comparable computational efficiency to standard scalar conditioning techniques. To this end, we conduct diffusion experiments that separately assess denoising and unconditional generation quality, and flow matching experiments focused on unconditional image generation.

4.1. Experimental Setup

Model Architectures. We employ two representative architectures to demonstrate both compatibility and generality. First, the deep residual UNet (DRUNet) [26], a widely used image denoiser not originally designed for generative modeling, is included to demonstrate that our proposed conditioning enables such non-generative architectures to function effectively as generative models. Second, the ablated diffusion model (ADM) U-Net [3] is selected for its open-source implementation of the sinusoidal embedding-based timestep conditioning that many later diffusion models build upon [2, 5, 15, 18]. While the main objective of our experiments is to compare different scalar conditioning methods within each architecture, we additionally set the number of parameters of both architectures to be approximately equal—by adding more residual blocks to DRUNet—to enable a fair comparison across architectures as well. The key architectural specifications for both models are summarized in Table 1.

Training Details. All models, including baselines, are trained from scratch on 69,000 FFHQ images (64×64 RGB) [8] for 500,000 iterations using a single NVIDIA

Table 1. Architectural details of DRUNet [26] and ADM [3] used in our generative modeling experiments.

	DRUNet [26]	ADM [3]
# Parameters	63.9 M	60.9 M
# Residual blocks	8	1
Base channel width	64	128
# Attention heads	N/A	4
# Head channels	N/A	64
Attention resolutions	N/A	[16]

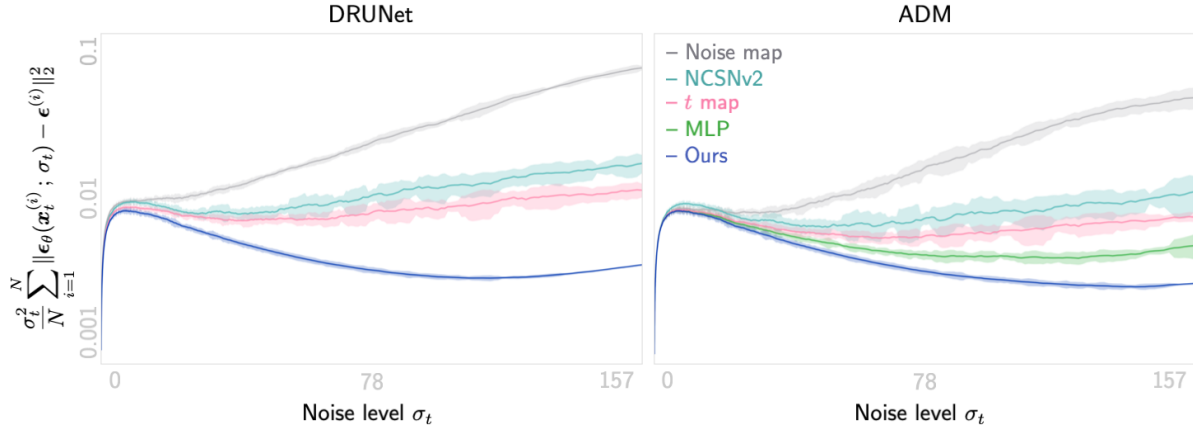


Figure 4. Denoising performance across diffusion noise levels for different scalar conditioning methods using DRUNet and ADM. The plot shows the noise-scaled mean squared error (MSE), $\frac{\sigma_t^2}{N} \sum_{i=1}^N \|\epsilon_\theta(\mathbf{x}_t^{(i)}; \sigma_t) - \epsilon^{(i)}\|_2^2$, where σ_t^2 denotes the noise variance at each diffusion step and N the number of test samples. Our method consistently achieves lower error across a wide range of noise levels.

Table 2. Comparison of scalar conditioning methods for image denoising. We report the mean squared noise prediction error, $\frac{1}{N} \sum_{i=1}^N \|\epsilon_\theta(\mathbf{x}_t^{(i)}; \sigma_t) - \epsilon^{(i)}\|_2^2$, where N denotes the number of test samples. Results are shown at representative timesteps $t \in \{0, 333, 666, 999\}$ with corresponding noise levels $\sigma_t \in \{0.01, 1.46, 9.49, 157\}$. The rightmost column shows the expected error over the full noise schedule. **Best** and **second-best** results are color-coded per noise level and per-architecture.

Sampling methods	Conditioning	$\frac{1}{N} \sum_{i=1}^N \ \epsilon_\theta(\mathbf{x}_t^{(i)}; \sigma_t) - \epsilon^{(i)}\ _2^2$				$\mathbb{E} \ \epsilon_\theta(\mathbf{x}_t; \sigma_t) - \epsilon\ _2^2$
		$t = 0 (\sigma_t = 0.01)$	$t = 333 (\sigma_t = 1.46)$	$t = 666 (\sigma_t = 9.49)$	$t = 999 (\sigma_t = 157)$	
DRUNet Diffusion	NCSNv2	5.282e-1	1.331e-2	1.510e-3	1.123e-4	2.038e-2
	σ map	5.562e-1	1.306e-2	1.522e-3	4.179e-4	2.077e-2
	t map	5.305e-1	1.249e-2	1.410e-3	7.854e-5	1.987e-2
	Ours	5.144e-1	1.180e-2	1.345e-3	2.556e-5	1.865e-2
ADM Diffusion	NCSNv2	4.914e-1	1.165e-2	1.453e-3	1.009e-4	1.795e-2
	σ map	4.970e-1	1.172e-2	1.390e-3	2.673e-4	1.819e-2
	t map	4.922e-1	1.164e-2	1.366e-3	5.346e-5	1.793e-2
	MLP	4.853e-1	1.171e-2	1.361e-3	4.369e-5	1.794e-2
	Ours	4.850e-1	1.161e-2	1.346e-3	1.917e-5	1.772e-2

RTX A6000 GPU. We use the AdamW optimizer [12] with learning rate 1×10^{-5} , batch size 256, dropout rate 0.1, and weight decay of 0.05. An exponential moving average (EMA) with decay rate 0.9999 is applied to stabilize optimization. For our proposed parameter interpolation, which includes a learnable monotonic interpolation function $\lambda(s)$, we assign a separate learning rate of 1×10^{-3} to the interpolation coefficients ϕ (introduced in Section 3.2). All other hyperparameters are kept identical across models to ensure a fair comparison.

Parameter Interpolation. We apply the proposed parameter interpolation mechanism (Section 3) to introduce scalar dependency at the parameter level. Specifically, we maintain two sets of learnable parameters and use a scalar-dependent, learnable interpolation function $\lambda(s)$ to linearly interpolate between them at each scalar step s , producing a single scalar-conditioned network configuration.

For DRUNet, the interpolated parameters include all

weights in 2D convolutional and transposed convolutional layers. For ADM, both convolutional weights and biases, as well as group-normalization parameters, are interpolated. The interpolation function $\lambda(s)$ is defined by a learnable vector ϕ of length $S = 1000$, matching the total timestep range used in both the diffusion and flow matching frameworks. As shown in Figure 3, the learned interpolation functions $\lambda(s)$ vary across architectures and generative frameworks, indicating that the model automatically adjusts its transition dynamics to match the scalar progression.

4.2. Diffusion Framework Evaluations

In diffusion-based frameworks, we evaluate (1) denoising accuracy across noise levels and (2) unconditional image generation quality. These experiments validate whether our parameter interpolation can improve both step-wise noise handling and overall sampling performance.

Training Setup. All diffusion models are trained un-

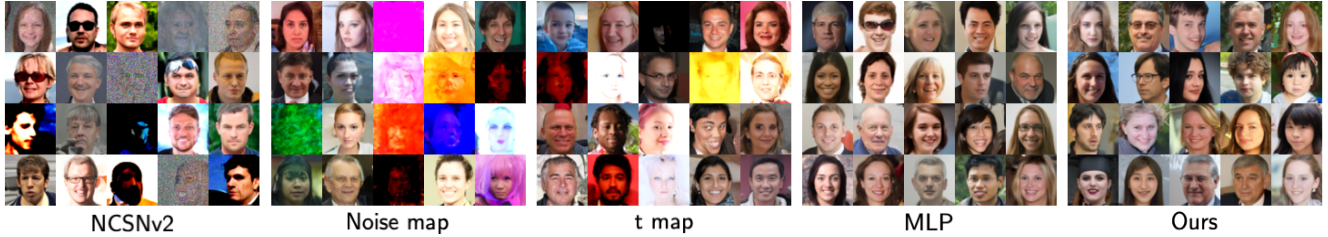


Figure 5. Examples of FFHQ 64×64 generated samples by ADM [3] architecture using DDIM sampler [20] with five different scalar conditioning methods.

Table 3. Comparison of image generation quality across conditioning methods on both diffusion-based and flow matching-based sampling frameworks. Our parameter interpolation achieves consistently superior performance in FID and sFID across all architectures, demonstrating effective and architecture-agnostic conditioning for generative modeling. **Best** and **second-best** results are color-coded per sampling methods and per-architecture.

Sampling methods	Conditioning	FID↓	sFID↓	Precision↑	Recall↑
DRUNet Diffusion	NCSNv2	111.98	68.13	0.243	0.265
	σ map	137.77	150.21	0.089	0.127
	t map	26.23	30.38	0.457	0.280
	Ours	12.23	17.11	0.633	0.307
DRUNet Flow	t map	13.73	16.60	0.627	0.302
	Ours	12.57	16.21	0.635	0.318
ADM Diffusion	NCSNv2	67.50	47.35	0.216	0.401
	σ map	96.71	58.77	0.283	0.341
	t map	13.14	21.90	0.581	0.376
	MLP	10.14	22.44	0.692	0.342
	Ours	8.32	17.45	0.667	0.377
ADM Flow	t map	8.51	16.41	0.685	0.373
	MLP	8.52	16.25	0.683	0.381
	Ours	7.96	16.20	0.687	0.389

der the variance-preserving (VP) formulation [3, 7], where clean data \mathbf{x}_0 are progressively corrupted by Gaussian noise according to

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}, \quad (3)$$

with $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ representing the injected noise. Here, $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$ is the cumulative product of per-step noise coefficients α_s , and α_t is scheduled linearly such that the variance increases from 0.0001 to 0.2 over training steps. This schedule ensures that $\mathbf{x}_{t=0}$ corresponds to the data distribution, while $\mathbf{x}_{t=T}$ approaches a standard Gaussian.

Each diffusion model learns to predict the added noise $\boldsymbol{\epsilon}$ from the noisy observation \mathbf{x}_t , parameterized as $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t; \sigma_t)$. The training objective minimizes the mean squared error between the predicted and true noise:

$$\mathcal{L}_{\text{MSE}} = \mathbb{E}_{t, \boldsymbol{\epsilon}} \left[\|\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, \sigma_t) - \boldsymbol{\epsilon}\|_2^2 \right].$$

All diffusion-based baselines follow this formulation, except NCSNv2 [22], which directly estimates the score function $\nabla \log p_{\sigma_t}(\mathbf{x}_t)$ instead of the noise.

Baseline Conditioning. For ADM architecture, we test four noise conditioning schemes: NCSNv2 [22], σ map [26], t map [10], and MLP-based conditioning that modulates normalization layers [3]. Among addition-based and normalization-based MLP conditioning variants, the normalization-based one is chosen, as it is adopted by many recent high-performing diffusion models [2, 5, 18]. For DRUNet architecture, we test NCSNv2, σ map, and t map, omitting MLP-based conditioning due to the absence of normalization layers required for injecting sinusoidal embeddings of the scalar variable.

For NCSNv2 conditioning, we follow the original loss formulation in [22], where the network directly predicts the score function, and apply a scaling term derived from the variance of the corresponding VP diffusion model. The original NCSNv2, however, incorporates additional sampling refinements—an extra denoising step after annealed Langevin dynamics and a variance-exploding (VE) formulation with tuned step-size selection—which we omit to ensure a fair comparison under a consistent diffusion probabilistic sampling framework.

For the σ map and t map methods, the scalar variable is encoded as a spatially constant, single-channel map appended to the input. In the σ map case, the conditioning value corresponds to the effective noise level associated with timestep t , defined as

$$\sigma_t = \sqrt{\frac{1 - \bar{\alpha}_t}{\bar{\alpha}_t}},$$

following [16, 17]. For the t map method, the conditioning value is given by the normalized timestep t/T , which provides a scalar representation of the current diffusion step. Finally, MLP-based conditioning injects sinusoidal embeddings of t into normalization layers via learned scale-shift parameters.

Denoising Evaluation. We assess denoising performance—a fundamental indicator of diffusion model quality—on 500 held-out images. Noisy image generation was repeated with 20 random seeds, and all conditioning methods share

the same noisy image realizations to ensure fair comparison. Figure 4 illustrates the mean squared error (MSE) of predicted noise $\epsilon_\theta(\mathbf{x}_t)$ against the ground truth ϵ across timesteps for DRUNet and ADM. Table 2 summarizes representative results, showing that our proposed scalar conditioning consistently achieves the lowest prediction errors across both architectures.

Sampling Evaluation. We generate 50,000 samples per model using 200 DDIM steps [20] and evaluate FID [6], sFID [14], precision, and recall [9] using the public implementation from [3], available at the following repository¹. As shown in Figure 5, scalar conditioning methods with minimal architectural constraints (e.g., NCSNv2, σ map, and t map) can generate plausible faces but often produce artifacts or unstable exposure. In contrast, our method achieves stable and coherent visual quality without relying on dedicated scalar embedding modules, comparable to MLP-based conditioning that explicitly uses such embeddings. Quantitative results in Table 3 further confirm these improvements across all evaluation metrics.

4.3. Flow Matching Framework Evaluations

Training Setup. Flow matching models are trained to predict velocity fields along the affine probability path [10],

$$\mathbf{x}_t = \beta_t \mathbf{x}_1 + \gamma_t \mathbf{x}_0, \quad \text{with} \quad \beta_t = t, \gamma_t = 1 - t,$$

where \mathbf{x}_1 is a data sample and $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. This formulation defines a linear interpolation between the data and Gaussian prior distributions, with the model learning the time-dependent velocity field that maps \mathbf{x}_0 to \mathbf{x}_1 .

We use the same architectures, parameter settings, and conditioning schemes as in Section 4.2, except for NCSNv2 [22], which is omitted because flow matching models directly predict velocity fields rather than score functions.

Sampling Evaluation. Consistent with diffusion framework evaluations, we generate 50,000 samples using the probability flow ODE with 200 steps. We evaluate the same perceptual metrics (FID, sFID, precision, and recall). As shown in Table 3, parameter interpolation again yields consistent quality improvements in FID and sFID across both ADM and DRUNet architectures, confirming that the proposed conditioning generalizes effectively beyond the diffusion formulation.

4.4. Computational Efficiency of Parameter Interpolation

We compare computational efficiency in terms of floating-point operations (FLOPs) and peak GPU memory under identical batch and image settings (see Table 4). Empirically, FLOPs increase by less than 0.72%, and peak GPU memory rises by approximately 5–9%, depending on the architecture.

¹<https://github.com/openai/guided-diffusion>

Table 4. Computational statistics of different conditioning methods. We report the number of parameters, peak GPU memory usage, and FLOPs for DRUNet and ADM under various scalar-conditioning schemes. Our parameter interpolation maintains efficiency comparable to baseline approaches despite doubling the parameter count.

Models	t conditioning	Params (M)	Peak GPU (GB)	FLOPs (G)
DRUNet	NCSNv2	63.90	17.64	34.86
	t map	63.90	17.65	34.86
	Ours	127.80	19.32	35.11
ADM	NCSNv2	60.87	37.24	49.54
	t map	60.87	37.25	49.54
	MLP	68.16	41.82	49.56
	Ours	121.74	38.84	49.77

In contrast, MLP-based conditioning introduces additional per-layer transformations that noticeably raise GPU memory requirements. Overall, parameter interpolation achieves efficient conditioning with minimal overhead, preserving computation and memory efficiency while maintaining strong generative performance and broad architectural compatibility.

5. Conclusion

We introduce *deep parameter interpolation (DPI)*, a simple and general-purpose approach for conditioning neural networks on scalar variables such as time or noise level in diffusion and flow matching frameworks. Unlike existing conditioning mechanisms that require input modifications or specialized embedding layers, our method introduces scalar dependence directly at the parameter level by interpolating between two learnable parameter sets within a single network according to a learnable monotonic function. This design maintains the original architecture, making it broadly applicable across diverse generative frameworks. Empirical evaluations on both diffusion and flow matching models demonstrate that DPI consistently improves denoising accuracy and sample quality across multiple architectures, including those not originally designed for generative modeling. Furthermore, the method achieves these gains with negligible computational overhead, offering a favorable balance between flexibility, efficiency, and performance.

6. Acknowledgement

This work was supported in part by the National Science Foundation under Grants No. 2622128 and No. 2043134 (CAREER), and in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Award DE-SC0025589 and Triad National Security, LLC (‘Triad’) contract 89233218CNA000001 [FWP: LANLE2A2].

References

- [1] Michael Samuel Albergo and Eric Vanden-Eijnden. Building normalizing flows with stochastic interpolants. In *International Conference on Learning Representations*, 2023. 1, 3
- [2] Shoufa Chen, Mengmeng Xu, Jiawei Ren, Yuren Cong, Sen He, Yanping Xie, Animesh Sinha, Ping Luo, Tao Xiang, and Juan-Manuel Perez-Rua. Gentron: Diffusion transformers for image and video generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6441–6451, 2024. 1, 5, 7
- [3] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat GANs on image synthesis. In *Advances in Neural Information Processing Systems*, pages 8780–8794, 2021. 1, 4, 5, 7, 8
- [4] Bradley Efron. Tweedie’s formula and selection bias. *Journal of the American Statistical Association*, 106(496):1602–1614, 2011. 2
- [5] 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, Alex Goodwin, Yannik Marek, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis. In *Proceedings of the 41st International Conference on Machine Learning (ICML 2024)*, 2024. 1, 5, 7
- [6] 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 *Advances in Neural Information Processing Systems*, 2017. 8
- [7] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems*, pages 6840–6851, 2020. 1, 2, 7
- [8] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019. 5
- [9] Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. In *Advances in Neural Information Processing Systems*, 2019. 8
- [10] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023. 1, 3, 7, 8
- [11] Xingchao Liu, Chengyu Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *International Conference on Learning Representations*, 2023. 1, 3
- [12] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *The Seventh International Conference on Learning Representations*, 2019. 6
- [13] Koichi Miyasawa. An empirical Bayes estimator of the mean of a normal population. *Bulletin of the International Statistical Institute*, 38:181–188, 1961. 2
- [14] Charlie Nash, Jacob Menick, Sander Dieleman, and Peter Battaglia. Generating images with sparse representations. In *Proceedings of the 38th International Conference on Machine Learning*, pages 7958–7968, 2021. 8
- [15] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021. 1, 5
- [16] Chicago Y. Park, Yuyang Hu, Michael T. McCann, Cristina Garcia-Cardona, Brendt Wohlberg, and Ulugbek S. Kamilov. Plug-and-play priors as a score-based method. In *IEEE International Conference on Image Processing*, Anchorage, Alaska, 2025. 2, 7
- [17] Chicago Y. Park, Michael T. McCann, Cristina Garcia-Cardona, Brendt Wohlberg, and Ulugbek S. Kamilov. Random walks with Tweedie: A unified view of score-based diffusion models [in the spotlight]. *IEEE Signal Processing Magazine*, 42(3):40–51, 2025. 1, 2, 7
- [18] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4195–4205, 2023. 1, 5, 7
- [19] Herbert Robbins. An empirical Bayes approach to statistics. In *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, pages 157–163. The Regents of the University of California, 1956. 2
- [20] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021. 7, 8
- [21] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *Advances in Neural Information Processing Systems*, 2019. 1, 2
- [22] Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. 1, 3, 7, 8
- [23] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations*, 2021. 1, 2
- [24] Pascal Vincent. A connection between score matching and denoising autoencoders. *Neural computation*, 23(7):1661–1674, 2011. 2
- [25] Xintao Wang, Ke Yu, Chao Dong, Xiaoou Tang, and Chen Change Loy. Deep network interpolation for continuous imagery effect transition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1692–1701, 2019. 4
- [26] Kai Zhang, Yawei Li, Wangmeng Zuo, Lei Zhang, Luc Van Gool, and Radu Timofte. Plug-and-play image restoration with deep denoiser prior. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):6360–6376, 2021. 1, 4, 5, 7