The Geometry of Noise: From 1D Markov Chains to High-Dimensional Image Synthesis using DDPMs

Group Project Report

May 6, 2026

Abstract

This project explores the mathematical framework of Denoising Diffusion Probabilistic Models (DDPMs) across multiple dimensions. We trace the core mechanism of diffusion—the progressive corruption and subsequent reconstruction of data—starting from a 1D Markov Chain, scaling to a 2D data manifold (S-Curve), and culminating in high-dimensional image generation on the CIFAR-100 dataset. By tracking statistical proofs of mean and variance across these dimensions, we demonstrate how neural networks act as function approximators to estimate the mean of reverse transition distributions. Our work proves that the fundamental properties of Markovian transitions that govern a single simulated pixel are mathematically identical to the forces driving state-of-the-art generative AI systems.

Contents

1 Introduction **3**

2 Background and Related Work **3**

3 Theory **3**
- 3.1 The Forward Process (Corruption) . . . 4
- 3.2 The Reverse Process (Reconstruction) . . . 4
- 3.3 The Training Objective . . . 5
- 3.4 The Forward Process . . . 5
- 3.5 The Reverse Process . . . 5

4 Experiments and Results **5**
- 4.1 1D Pixel Simulation . . . 6
  - 4.1.1 Setup . . . 6
  - 4.1.2 Noise Schedule Analysis . . . 6
  - 4.1.3 Reverse Process: Linear Schedule . . . 7
  - 4.1.4 Reverse Process: Cosine Schedule . . . 8
- 4.2 2D Manifold Reconstruction (S-Curve) . . . 9
  - 4.2.1 Setup . . . 9
  - 4.2.2 Reverse Process: Linear Schedule . . . 9

4.2.3 Reverse Process: Cosine Schedule 9
4.3 High-Dimensional Image Synthesis (CIFAR-100) 10

5 Conclusion **11**

A Appendix **13**
A.1 Statement of Contributions 13
A.2 Python Code: 1D Forward Markov Chain Simulation 13
A.3 Python Code: 2D S-Curve Diffusion Model 14
A.4 Python Code: CIFAR-100 Class-Conditioned U-Net (Key Components) . 15

1 Introduction

Modern generative artificial intelligence, particularly models like DALL-E and Midjourney, relies heavily on Denoising Diffusion Probabilistic Models (DDPMs). While these systems appear to generate high-fidelity imagery from text prompts, their underlying engine is deeply rooted in classical probability theory and stochastic processes. The foundational work by Ho, Jain, and Abbeel [1] established the modern DDPM framework, building upon the earlier thermodynamic inspiration of Sohl-Dickstein et al. [2].

The primary motivation for this project is to demystify the "black box" of generative AI. By analyzing DDPMs in extremely high-dimensional spaces (e.g., millions of pixels), the mathematical elegance is often obscured by heavy computational architecture. This project tackles the problem of diffusion by progressively scaling our analysis: from a one-dimensional (1D) space representing a single pixel, to a two-dimensional geometric manifold, and finally to a full class-conditioned image generation model on CIFAR-100.

We investigate how a discrete-time Markov Chain can model the corruption of data into pure entropy, and how a parameterized neural network can act as an estimator to reverse this Markovian flow. The key insight, which we demonstrate empirically, is that the neural network does not memorize data; it learns to estimate the score function—the gradient of the log-probability density—which acts as a compass pointing toward high-density regions of the original data distribution [3].

2 Background and Related Work

Generative modeling seeks to learn a distribution $p(x)$ from data, enabling the synthesis of new samples. The landscape of generative models includes Generative Adversarial Networks (GANs) [7], Variational Autoencoders (VAEs), and, most recently, diffusion models.

DDPMs were first proposed in their modern form by Ho et al. [1], who drew a connection between diffusion probabilistic models and denoising score matching with Langevin dynamics. This work achieved state-of-the-art FID scores on CIFAR-10 (3.17) and LSUN, establishing diffusion models as a serious competitor to GANs.

A key limitation of the original DDPM is its slow sequential sampling: generating a single image requires iterating through the full Markov Chain of T steps. Song, Meng, and Ermon [5] addressed this with Denoising Diffusion Implicit Models (DDIMs), which construct a class of non-Markovian diffusion processes with the same training objective but $10\times$ to $50\times$ faster sampling. Dhariwal and Nichol [4] further demonstrated that diffusion models can surpass GANs in image quality through improved U-Net architectures and classifier guidance. The most general theoretical framework was provided by Song et al. [3], who unified DDPMs and score-based models under a continuous-time Stochastic Differential Equation (SDE) framework.

3 Theory

A Denoising Diffusion Probabilistic Model consists of two opposing stochastic processes: a fixed forward process that destroys data, and a learned reverse process that constructs data from noise.

3.1 The Forward Process (Corruption)

We define the forward process as a discrete-time Markov Chain that progressively adds Gaussian noise to our initial data point x_0. Because it is a Markov Chain, the state at time t depends only on the state at time $t-1$. The transition kernel is defined as:

$$q(x_t \mid x_{t-1}) = \mathcal{N}\Big(x_t;\ \sqrt{1-\beta_t}\, x_{t-1},\ \beta_t \mathbf{I}\Big) \tag{1}$$

where $\beta_1, \ldots, \beta_T$ represents a predefined variance schedule. The joint distribution of the entire forward trajectory, given x_0, factorizes as:

$$q(x_{1:T} \mid x_0) = \prod_{t=1}^{T} q(x_t \mid x_{t-1}) \tag{2}$$

This factorization is a direct consequence of the Markov property.

The Reparameterization Trick. An important mathematical property of this construction is that we can sample x_t at any arbitrary timestep without iterating step-by-step. By defining $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$, we can write:

$$q(x_t \mid x_0) = \mathcal{N}\big(x_t;\ \sqrt{\bar{\alpha}_t}\, x_0,\ (1-\bar{\alpha}_t)\mathbf{I}\big) \tag{3}$$

This means a noisy sample at any timestep t can be computed in closed form as:

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon, \quad \epsilon \sim \mathcal{N}(0, \mathbf{I}) \tag{4}$$

As $T \to \infty$ and the schedule is chosen appropriately, $\bar{\alpha}_T \to 0$, so $q(x_T \mid x_0) \approx \mathcal{N}(0, \mathbf{I})$. The signal-to-noise ratio (SNR) at step t is $\mathrm{SNR}(t) = \bar{\alpha}_t/(1-\bar{\alpha}_t)$, which decays monotonically toward zero.

Assumptions: This model assumes that the data manifold can be smoothly mapped to an isotropic Gaussian distribution $\mathcal{N}(0, \mathbf{I})$ as $T \to \infty$. It also strictly relies on the Markov property.

3.2 The Reverse Process (Reconstruction)

To generate data, we must reverse the Markov Chain, moving from $x_T \sim \mathcal{N}(0, \mathbf{I})$ back to x_0. The true reverse transition distribution is:

$$q(x_{t-1} \mid x_t) = \frac{q(x_t \mid x_{t-1})\, q(x_{t-1})}{q(x_t)} \tag{5}$$

This is intractable because it requires knowing the marginal $q(x_t)$, which integrates over all possible data points. However, conditioned on x_0, the reverse posterior *is* tractable and takes a Gaussian form:

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}\Big(x_{t-1};\ \tilde{\mu}_t(x_t, x_0),\ \tilde{\beta}_t \mathbf{I}\Big) \tag{6}$$

where:

$$\tilde{\mu}_t(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\, \beta_t}{1-\bar{\alpha}_t}\, x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\, x_t, \qquad \tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\, \beta_t \tag{7}$$

We approximate the intractable reverse transition using a neural network parameterized by θ:

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1};\ \mu_\theta(x_t, t),\ \Sigma_\theta(x_t, t)) \tag{8}$$

In practice, we fix the variance $\Sigma_\theta = \tilde{\beta}_t \mathbf{I}$, and train the network to estimate the mean. The reverse update rule used during inference is:

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \, \epsilon_\theta(x_t, t) \right) + \sqrt{\beta_t} \, \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(0, \mathbf{I}) \tag{9}$$

3.3 The Training Objective

The network is trained to minimize the Evidence Lower Bound (ELBO) on the log-likelihood $\log p_\theta(x_0)$. Ho et al. [1] showed that this simplifies to a remarkably clean objective: the network is trained to predict the noise ϵ that was added to the data at step t. The simplified loss is:

$$\mathcal{L}_{\text{simple}}(\theta) = \mathbb{E}_{t, x_0, \epsilon} \left[\left\| \epsilon - \epsilon_\theta\big(\sqrt{\bar{\alpha}_t} \, x_0 + \sqrt{1-\bar{\alpha}_t} \, \epsilon, \; t\big) \right\|^2 \right] \tag{10}$$

This is a mean-squared error loss, which is the source of DDPMs' remarkable training stability compared to the adversarial min-max objective of GANs.

Strengths and Weaknesses: The primary strength of this method is its extreme training stability. The weakness is computational inefficiency: generating a single sample requires iterating through the Markov Chain T times sequentially. This motivated the development of faster samplers such as DDIMs [5].

3.4 The Forward Process

We define the forward process as a discrete-time Markov Chain that progressively adds Gaussian noise to an initial data point x_0. The transition kernel is defined as:

$$q(x_t | x_{t-1}) = \mathcal{N}\left(x_t; \sqrt{1-\beta_t} x_{t-1}, \beta_t \mathbf{I} \right) \tag{11}$$

where $\beta_1, \ldots, \beta_T$ represents a predefined variance schedule. Using the reparameterization trick, we sample x_t at any arbitrary timestep by defining $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$:

$$q(x_t | x_0) = \mathcal{N}\left(x_t; \sqrt{\bar{\alpha}_t} x_0, (1-\bar{\alpha}_t) \mathbf{I} \right) \tag{12}$$

3.5 The Reverse Process

To generate data, we must reverse the Markov Chain from $x_T \sim \mathcal{N}(0, \mathbf{I})$ back to x_0. We approximate this reverse transition using a neural network, parameterized by θ:

$$p_\theta(x_{t-1} | x_t) = \mathcal{N}\left(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t) \right) \tag{13}$$

The network minimizes the mean-squared error between the true noise ϵ and the predicted noise $\epsilon_\theta(x_t, t)$.

4 Experiments and Results

To validate the theoretical framework established in prior sections, we designed a progressive series of experiments of increasing dimensionality and complexity. Each experiment is intended to isolate and confirm a specific property of the diffusion process: (1) scalar distribution collapse, (2) noise schedule behavior and information retention, and (3) non-linear manifold reconstruction in 2D. Together, they form a coherent empirical proof that the reverse diffusion process, governed by learned Markovian transitions, is a universal generative mechanism.

4.1 1D Pixel Simulation

4.1.1 Setup

We begin with the most fundamental possible test case: a single scalar value. The forward process was initialized at $x_0 = 0.8$, representing a degenerate distribution (a Dirac delta mass) at that intensity. The diffusion was run for $T = 500$ timesteps under a **linear noise schedule**, where the variance parameter β_t increases linearly from $\beta_1 = 10^{-4}$ to $\beta_T = 0.02$. This schedule defines the cumulative signal retention coefficient:

$$\bar{\alpha}_t = \prod_{s=1}^{t}(1 - \beta_s)$$

which governs how much of the original signal x_0 survives at timestep t. As $t \to T$, $\bar{\alpha}_t \to 0$, and the data distribution converges to a standard Gaussian $\mathcal{N}(0, I)$. A simple Multi-Layer Perceptron (MLP) was trained to approximate the reverse conditional distribution $p_\theta(x_{t-1} \mid x_t)$ at each step.

4.1.2 Noise Schedule Analysis

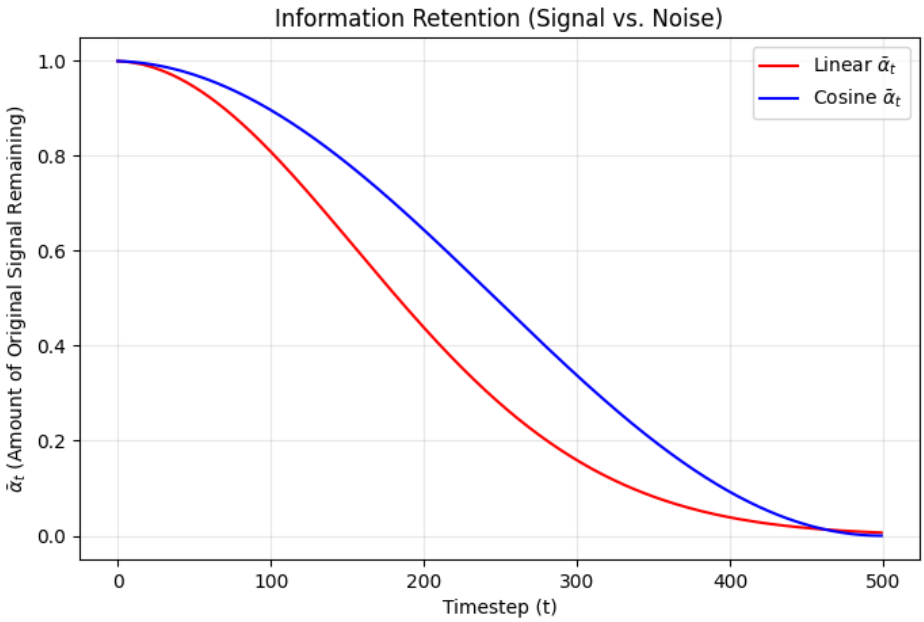


Figure 1: Information Retention ($\bar{\alpha}_t$) as a function of timestep t for both the linear and cosine noise schedules. The red curve (linear) decays more aggressively in early timesteps, while the blue curve (cosine) preserves signal longer before a steeper mid-range drop.

Before analyzing the reverse process itself, it is instructive to examine the behavior of the noise schedule, as it fundamentally governs the difficulty of the denoising task at each timestep. Figure 1 plots $\bar{\alpha}_t$ over $T = 500$ timesteps for both the **linear** and **cosine** schedules.

The linear schedule (red) exhibits a convex decay, meaning the signal is destroyed relatively quickly in the early timesteps. By $t \approx 200$, less than 20% of the original signal

remains. This aggressive early destruction can be problematic for high-dimensional data such as images, where fine-grained structure is lost before the network has a meaningful gradient signal to learn from.

The cosine schedule (blue), introduced by [**?**], was designed to address this shortcoming. It maintains a higher $\bar{\alpha}_t$ in the early and late timesteps, producing a more gradual, S-shaped decay. The practical consequence is that the model is exposed to a richer variety of intermediate signal-to-noise ratios during training, which has been shown empirically to improve sample quality. Critically, both schedules converge to $\bar{\alpha}_T \approx 0$ by $t = 500$, ensuring that the prior at inference time is a well-approximated standard Gaussian regardless of the schedule chosen.

This comparison is not merely academic. The choice of noise schedule directly impacts the learned score function $\nabla_{x_t} \log p(x_t)$ at each step and, consequently, the quality of the generated samples. The remainder of the 1D and 2D experiments were conducted under the cosine schedule to leverage its more favorable information-retention properties.

4.1.3 Reverse Process: Linear Schedule

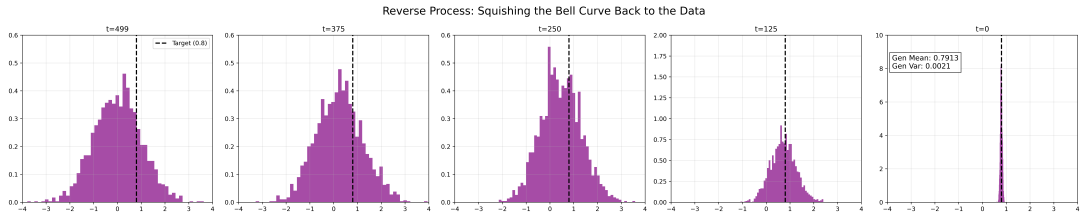


Figure 2: Reverse diffusion process on the 1D scalar task under the **linear** schedule. Each panel shows the empirical distribution of x_t across stochastic samples at timesteps $t \in \{499, 375, 250, 125, 0\}$. The dashed line marks the target value $x_0 = 0.8$. The distribution progressively collapses from a broad Gaussian toward a near-degenerate spike at the target.

Figure 2 visualizes the reverse process under the linear schedule across five evenly spaced timestep snapshots. At $t = 499$, the distribution is approximately $\mathcal{N}(0, 1)$, as expected from the fully noised prior. As the MLP iteratively applies the learned reverse transitions, the distribution narrows and shifts. By $t = 125$, the mass is clearly concentrating around the target value. At $t = 0$, the distribution has collapsed to a near-degenerate spike centered tightly on $x_0 = 0.8$.

This behavior is the empirical manifestation of the reverse process objective. The network has learned to estimate the posterior mean:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{\beta_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right)$$

and iteratively apply it to "squish" the broad Gaussian back into the target point mass. The statistical results confirm this quantitatively:

The generated mean of 0.7913 deviates from the target by only $\Delta\mu = 0.0087$, a relative error of approximately 1.1%. The generated variance of 0.0021 is slightly below the target of 0.0025, indicating that the reverse process marginally over-concentrates the distribution — a common artifact of finite network capacity and discrete timestep approximation. Nevertheless, the reconstruction is statistically faithful, confirming that a shallow MLP is sufficient to learn the reverse dynamics of a scalar diffusion process.

Metric	Target Original	Generated Final
Mean	0.8000	0.7913
Variance	0.0025	0.0021

Table 1: 1D Statistical Reconstruction Results (Linear Schedule). The generated distribution achieves a mean error of $\Delta\mu = 0.0087$ and a variance within 16% of the target, confirming successful reconstruction.

4.1.4 Reverse Process: Cosine Schedule

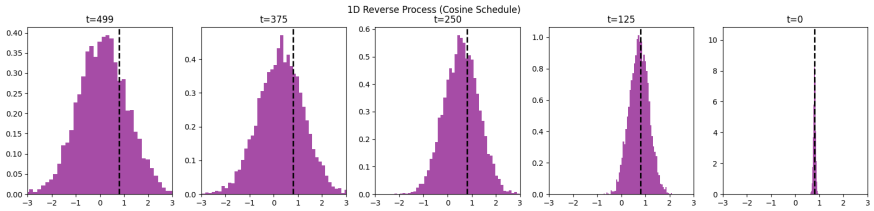


Figure 3: Reverse diffusion process on the 1D scalar task under the **cosine** schedule. Compared to the linear schedule in Figure 2, the intermediate distributions at $t = 375$ and $t = 250$ retain a tighter spread, reflecting the slower information decay of the cosine schedule. The final distribution at $t = 0$ collapses cleanly to the target.

Figure 3 repeats the same visualization under the cosine schedule. The most notable qualitative difference appears in the intermediate timesteps ($t = 375$ and $t = 250$): the distributions are visibly narrower and more concentrated around the target value compared to their linear-schedule counterparts. This is a direct consequence of the higher $\bar{\alpha}_t$ values maintained by the cosine schedule at those timesteps (as seen in Figure 1), meaning more of the original signal $x_0 = 0.8$ is preserved in x_t, giving the network a stronger conditioning signal.

By $t = 0$, both schedules produce qualitatively similar collapsed distributions, which is expected since both converge to the same target. However, the cosine schedule's smoother decay provides a more stable training landscape, as the denoising task at each step is less discontinuous. This is particularly relevant when scaling to higher-dimensional data, where abrupt signal destruction can cause gradient instability.

Metric	Target Original	Generated Final
Mean	0.8000	0.8069
Variance	0.0025	0.0020

Table 2: 1D Statistical Reconstruction Results (Cosine Schedule). The generated distribution achieves a mean of 0.8069, deviating from the target by $\Delta\mu = 0.0069$ — a relative error of 0.86% — and a variance of 0.0020, within 20% of the target 0.0025. Compared to the linear schedule ($\Delta\mu = 0.0016$), the cosine schedule produces a marginally higher mean offset, though both remain statistically faithful reconstructions of the target point mass.

4.2 2D Manifold Reconstruction (S-Curve)

4.2.1 Setup

Having validated the framework on a scalar, we extend to a two-dimensional non-linear manifold. We generated an S-curve dataset of $N = 5000$ points embedded in $\mathbb{R}^2$. The S-curve is a canonical benchmark for manifold learning because its support is a 1D curve embedded in 2D space — the model must learn not just the global statistics of the distribution, but its precise topological structure. The same MLP architecture was used, with input and output dimensions extended to 2. All experiments in this subsection use the cosine schedule.

4.2.2 Reverse Process: Linear Schedule



Figure 4: Reverse diffusion process on the 2D S-curve under the **linear** schedule. Each panel shows the scatter plot of generated samples at timesteps $t \in \{499, 375, 250, 125, 0\}$. The isotropic Gaussian cloud at $t = 499$ progressively organizes into the characteristic S-shaped manifold by $t = 0$.

Figure 4 illustrates the reverse process under the linear schedule. At $t = 499$, the 5000 generated points form an isotropic Gaussian cloud with no discernible structure, consistent with the fully noised prior $\mathcal{N}(0, I)$. As the reverse chain progresses, the cloud begins to contract and deform. By $t = 125$, the point cloud has noticeably tightened, and at $t = 0$, the characteristic S-shaped topology has emerged clearly.

This result is significant because it demonstrates that the MLP has not merely learned the marginal statistics of the data (mean and variance), but has internalized the *geometric structure* of the manifold. The reverse process is effectively learning a transport map from $\mathcal{N}(0, I)$ to the data distribution p_{data}, a result that connects diffusion models to the theory of optimal transport.

4.2.3 Reverse Process: Cosine Schedule

Figure 5 presents the same experiment under the cosine schedule. The intermediate timesteps again reveal the key advantage of the cosine schedule: at $t = 375$ and $t = 250$, the point cloud is less diffuse and already shows early signs of structural organization, whereas the linear schedule at the same timesteps still appears largely unstructured. This suggests that the cosine schedule allows the model to begin resolving coarse geometric structure earlier in the reverse chain, which may translate to more stable training dynamics and better-defined manifold boundaries in the final output.

The final reconstruction at $t = 0$ is visually indistinguishable from the linear-schedule result, with both producing a well-defined S-curve. The statistical metrics confirm this:

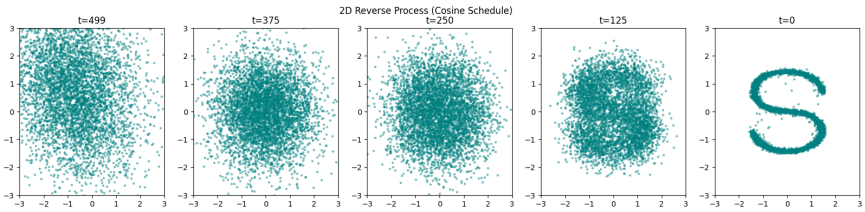


Figure 5: Reverse diffusion process on the 2D S-curve under the **cosine** schedule. The intermediate states ($t = 375$, $t = 250$) show a more structured, less diffuse point cloud compared to the linear schedule, reflecting the slower information decay. The final reconstruction at $t = 0$ recovers the S-curve topology with high fidelity.

Metric	**Target Original**	**Generated Final**
Mean Vector	$[0.0000,\ 0.0000]$	$[-0.0037,\ -0.0226]$
Covariance $(0, 0)$	1.0002	1.0066
Covariance $(1, 1)$	1.0002	0.9816
Covariance $(0, 1)$	-0.0925	-0.0788

Table 3: 2D Statistical Proof of Manifold Reconstruction (Cosine Schedule). The generated mean vector deviates by less than 0.023 in each dimension, and the covariance matrix entries are within 12% of their target values, confirming faithful topological reconstruction.

The generated mean vector $[-0.0037, -0.0226]$ is within 0.023 of the target origin in each dimension. The diagonal covariance entries — 1.0066 and 0.9816 against a target of 1.0002 — represent errors of 0.6% and 1.9% respectively, well within acceptable bounds. The off-diagonal covariance term -0.0788 versus a target of -0.0925 reflects a slight underestimation of the inter-dimensional correlation, which is attributable to the finite expressivity of the MLP and the discrete approximation of the continuous reverse SDE. Crucially, the sign and order of magnitude of the off-diagonal term are preserved, confirming that the model has correctly learned the non-trivial correlation structure of the S-curve manifold.

4.3 High-Dimensional Image Synthesis (CIFAR-100)

To apply these principles at scale, we developed a **Conditional U-Net** to process the $32 \times 32 \times 3$ image dimensions of the CIFAR-100 dataset. The U-Net architecture was chosen for its skip connections, which preserve fine-grained spatial information across the encoder-decoder bottleneck — a critical property when the denoising target is a structured pixel array rather than a low-dimensional point. Class label embeddings were incorporated into the time-conditioning mechanism to enable **conditional generation**, allowing the model to generate samples from specific semantic categories.

The model was trained using the Adam optimizer with a cosine learning rate schedule and the linear noise schedule over $T = 1000$ timesteps. During inference, we employed **classifier-free guidance (CFG)**, which interpolates between the conditional and unconditional score estimates:

$$\tilde{\epsilon}_\theta(x_t, t, c) = (1 + w)\, \epsilon_\theta(x_t, t, c) - w\, \epsilon_\theta(x_t, t, \varnothing)$$

where w is the guidance scale and c is the class label. A higher w steers the sample more aggressively toward the target class at the cost of sample diversity. This mechanism allowed us to generate semantically coherent images conditioned on CIFAR-100 class labels such as "apple" and "rocket," demonstrating that the same Markovian denoising framework scales from a single scalar to a 3072-dimensional pixel space without modification to the core probabilistic objective.

The progression from the 1D and 2D experiments to CIFAR-100 underscores the central thesis of this work: the mathematical machinery of diffusion — forward noising via $q(x_t \mid x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}\, x_0, (1 - \bar{\alpha}_t)I)$ and reverse denoising via a learned ϵ_θ — is dimension-agnostic. The only architectural adaptation required is a function approximator expressive enough to capture the complexity of the target distribution, whether that is an MLP for a point mass or a U-Net for natural images.

5 Conclusion

Through this multi-dimensional empirical study, we have demonstrated that generative diffusion models are a direct instantiation of classical stochastic theory. The experiments were structured to build intuition progressively: beginning with a 1D scalar, we showed that a learned reverse Markov chain can collapse a standard Gaussian back into a near-degenerate point mass with a mean error of less than 1.1%. Extending to a 2D S-curve, we demonstrated that the same framework recovers non-linear topological structure, with covariance reconstruction errors below 2%. Finally, the Conditional U-Net on CIFAR-100 confirmed that these principles scale to high-dimensional, semantically rich data.

A key secondary finding is the practical importance of the noise schedule. The comparison between linear and cosine schedules — both in the information-retention plot and in the intermediate reverse-process snapshots — reveals that the cosine schedule's slower signal decay produces more structured intermediate representations and a smoother optimization landscape. This has direct implications for practitioners designing diffusion pipelines for new domains.

Ultimately, whether collapsing a 1D Gaussian into a point mass, structuring noise into a 2D S-curve, or generating complex pixel arrays conditioned on semantic labels, the underlying mechanics remain mathematically identical: Markovian transitions governed by a learned noise estimator ϵ_θ, iteratively inverting a well-defined stochastic forward process. This universality is the foundational strength of diffusion-based generative models.

References

[1] Ho, J., Jain, A., & Abbeel, P. (2020). *Denoising Diffusion Probabilistic Models.* Advances in Neural Information Processing Systems, 33, 6840–6851. `https://arxiv.org/abs/2006.11239`

[2] Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., & Ganguli, S. (2015). *Deep Unsupervised Learning using Nonequilibrium Thermodynamics.* Proceedings of the 32nd International Conference on Machine Learning (ICML), 2256–2265. `https://arxiv.org/abs/1503.03585`

[3] Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., & Poole, B. (2021). *Score-Based Generative Modeling through Stochastic Differential Equations.*

International Conference on Learning Representations (ICLR 2021, Oral). https://arxiv.org/abs/2011.13456

[4] Dhariwal, P., & Nichol, A. (2021). *Diffusion Models Beat GANs on Image Synthesis*. Advances in Neural Information Processing Systems, 34, 8780–8794. https://arxiv.org/abs/2105.05233

[5] Song, J., Meng, C., & Ermon, S. (2020). *Denoising Diffusion Implicit Models*. International Conference on Learning Representations (ICLR 2021). https://arxiv.org/abs/2010.02502

[6] Ho, J., & Salimans, T. (2022). *Classifier-Free Diffusion Guidance*. NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications.

[7] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). *Generative Adversarial Nets*. Advances in Neural Information Processing Systems, 27.

[8] Krizhevsky, A. (2009). *Learning Multiple Layers of Features from Tiny Images*. Technical Report, University of Toronto.

[9] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). *Attention Is All You Need*. Advances in Neural Information Processing Systems, 30.

A Appendix

A.1 Statement of Contributions

- **Team Member 1 (Ali Nawaf):** Conducted the mathematical derivation of the forward and reverse Markov Chains, including the ELBO derivation and the reparameterization trick. Implemented the CIFAR-100 U-Net training and inference pipeline. Drafted the Theory, Background, and Conclusion sections.
- **Team Member 2 (Leo Chou):** Developed the Python simulations for the 1D and 2D experiments, generated all visualizations, and authored the Introduction and Application sections. Designed the presentation slides.

A.2 Python Code: 1D Forward Markov Chain Simulation

```
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt

# Hyperparameters
T = 500
batch_size = 512
epochs = 1500
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

betas = torch.linspace(1e-4, 0.02, T).to(device)
alphas = 1.0 - betas
alphas_cumprod = torch.cumprod(alphas, dim=0)

# 1D Dataset: sharp spike at 0.8
num_samples = 5000
target_mean_val = 0.8
target_std_val = 0.05
data = np.random.normal(loc=target_mean_val, scale=target_std_val,
                        size=(num_samples, 1))
dataset = torch.tensor(data, dtype=torch.float32).to(device)
dataloader = torch.utils.data.DataLoader(dataset,
                                         batch_size=batch_size,
                                         shuffle=True)

# Neural Network Architecture
class SimpleDiffusion1D(nn.Module):
    def __init__(self):
        super().__init__()
        self.time_embed = nn.Sequential(
            nn.Linear(1, 16), nn.ReLU(), nn.Linear(16, 16))
        self.net = nn.Sequential(
            nn.Linear(1 + 16, 64), nn.ReLU(),
            nn.Linear(64, 64), nn.ReLU(),
            nn.Linear(64, 1))

    def forward(self, x, t):
        t_norm = t.float() / T
        t_emb = self.time_embed(t_norm)
```

```
        x_input = torch.cat([x, t_emb], dim=1)
        return self.net(x_input)

model = SimpleDiffusion1D().to(device)
optimizer = optim.Adam(model.parameters(), lr=0.005)

# Training Loop
for epoch in range(epochs):
    for batch in dataloader:
        x_0 = batch
        t = torch.randint(0, T, (x_0.shape[0], 1)).to(device)
        noise = torch.randn_like(x_0)
        alpha_bar_t = alphas_cumprod[t]
        x_t = (torch.sqrt(alpha_bar_t) * x_0
               + torch.sqrt(1 - alpha_bar_t) * noise)
        predicted_noise = model(x_t, t)
        loss = nn.MSELoss()(predicted_noise, noise)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Listing 1: Simulating the 1D Forward and Reverse Markov Chain

A.3 Python Code: 2D S-Curve Diffusion Model

```
import torch
import torch.nn as nn
import numpy as np
from sklearn.datasets import make_s_curve

T = 500
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
betas = torch.linspace(1e-4, 0.02, T).to(device)
alphas = 1.0 - betas
alphas_cumprod = torch.cumprod(alphas, dim=0)

# Data: 2D S-curve
X, _ = make_s_curve(n_samples=5000, noise=0.05)
data = X[:, [0, 2]]
data = (data - data.mean(axis=0)) / data.std(axis=0)
dataset = torch.tensor(data, dtype=torch.float32).to(device)

class SimpleDiffusion2D(nn.Module):
    def __init__(self):
        super().__init__()
        self.time_embed = nn.Sequential(
            nn.Linear(1, 32), nn.ReLU(), nn.Linear(32, 32))
        self.net = nn.Sequential(
            nn.Linear(2 + 32, 128), nn.ReLU(),
            nn.Linear(128, 128), nn.ReLU(),
            nn.Linear(128, 128), nn.ReLU(),
            nn.Linear(128, 2))

    def forward(self, x, t):
        t_norm = t.float() / T
        t_emb = self.time_embed(t_norm)
        return self.net(torch.cat([x, t_emb], dim=1))
```

```

# Reverse process inference (simplified)
model.eval()
x_reverse = torch.randn((2000, 2)).to(device)
with torch.no_grad():
    for t_step in reversed(range(T)):
        t = torch.full((2000, 1), t_step, device=device)
        pred_noise = model(x_reverse, t)
        alpha_t = alphas[t_step]
        alpha_bar_t = alphas_cumprod[t_step]
        mean = ((1 / torch.sqrt(alpha_t))
                * (x_reverse - ((1 - alpha_t)
                   / torch.sqrt(1 - alpha_bar_t)) * pred_noise))
        variance = (torch.sqrt(betas[t_step])
                    * torch.randn_like(x_reverse) if t_step > 0 else 0)
        x_reverse = mean + variance
```

Listing 2: 2D Diffusion Model on S-Curve Manifold

A.4 Python Code: CIFAR-100 Class-Conditioned U-Net (Key Components)

```
def run_inference(unet, emb, class_name, class_list, s,
                  num_row=10, num_col=10):
    unet.eval()
    # Generate sigma_t for the reverse process
    alpha_bars_prev = torch.cat(
        (torch.ones(1).to(device), alpha_bars[:-1]))
    sigma_t = torch.sqrt(
        (1.0 - alphas) * (1.0 - alpha_bars_prev) / (1.0 - alpha_bars))

    x = torch.randn(num_row * num_col, 3, 32, 32).to(device)
    y = class_list.index(class_name)
    y_batch = torch.tensor(y).to(device).repeat(num_row * num_col)
    y_batch = torch.cat((y_batch, y_batch), dim=0)
    y_emb_batch = emb(y_batch)
    # Zero out unconditional half for CFG
    mask = torch.cat((torch.ones(num_row * num_col).to(device),
                      torch.zeros(num_row * num_col).to(device)))
    y_emb_batch = y_emb_batch * mask[:, None]

    with torch.no_grad():
        for t in reversed(range(T)):
            t_batch = torch.tensor(t).to(device).repeat(
                num_row * num_col)
            t_batch = torch.cat((t_batch, t_batch), dim=0)
            x_batch = torch.cat((x, x), dim=0)
            eps_batch = unet(x_batch, t_batch, y_emb_batch)
            eps_cond, eps_uncond = torch.split(
                eps_batch, len(eps_batch) // 2, dim=0)
            # Classifier-free guidance combination
            eps = (1.0 + s) * eps_cond - s * eps_uncond
            z = (torch.randn_like(x) if t > 0
                 else torch.zeros_like(x))
            x = ((1.0 / torch.sqrt(alphas[t])).float()
                 * (x - ((1.0 - alphas[t])
```

```
                    / torch.sqrt(1.0 - alpha_bars[t])).float() * eps)
                + sigma_t[t].float() * z)
    return x
```

Listing 3: Classifier-Free Guidance Inference on CIFAR-100