Probabilistic Machine Learning

Alexandros Kyriakopoulos
vkb160@alumni.ku.dk

Catarina Carapinha
fqr728@alumni.ku.dk

Gabriela Kirejczyk
cnk494@alumni.ku.dk

January 17, 2025

Abstract

This report examines Denoising Diffusion Probabilistic Models (DDPMs) and Gaussian Processes (GPs) for generative modeling and constrained function fitting. We explore parameterization, guided diffusion, and continuous-time extensions in DDPMs, evaluating their effects on generative performance using MNIST. For GPs, we incorporate integral constraints, derive posterior distributions, and analyze their influence on function fitting and model behavior. Our code can be found at: https://github.com/AlexandrosKyr/Probabilistic-Machine-Learning-Diffusion-GaussianProcesses.

Keywords Denoising Diffusion Probabilistic Models (DDPMs) · Generative Models · Log-Likelihood · Gaussian Processes (GPs) · Variational Lower Bound (VLB)

1 Quantitative Analysis of Diffusion Models

Denoising diffusion probabilistic models (DDPMs) have emerged as a class of generative models capable of achieving high-quality image synthesis through a gradual reverse-noising process. In this project, we explore variations and extensions of the basic DDPM model, focusing on comparing noise prediction strategies, variance reduction techniques, and the benefits of transitioning to continuous-time formulations.

1.1 Denoising Diffusion Models with Different Parameterizations: ϵ, μ, and x_0

In this section, we analyze the impact of using different target parameterizations for the neural network's predictions in DDPMs, namely the noise level (ϵ), the mean of the reverse process (μ), and the original clean image distribution (x_0).

Implementation Details To support the implementation of different target parameterizations, we modified the provided NN template (based on Song et al. [2021]'s UNET-style prediction model) to include a post-processing step (Appendix A.1) that outputs the appropriate predictor, based on the predicted added noise ϵ_θ at each time step t (default output). We integrated the post-processing step in the forward pass of the network using the following diffusion equations (derived from Ho et al. [2020]):

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

$$\epsilon_\theta = \frac{x_t - \sqrt{\bar{\alpha}_t} \cdot x_{0,\theta}}{\sqrt{1-\bar{\alpha}_t}} \iff x_{0,\theta} = \frac{x_t - \sqrt{1-\bar{\alpha}_t} \cdot \epsilon_\theta}{\sqrt{\bar{\alpha}_t}}, \tag{2}$$

where β_t controls how much Gaussian noise is added to the data at each timestep during the forward diffusion process, $\alpha_t = 1 - \beta_t$ represents the noise scaling factor, and $\bar{\alpha}t$ is equal to the cumulative product of α_t values from timestep 1 to t (i.e. remaining signal after t timesteps). These metrics are of important in defining the reverse diffusion process, which aims to recover x_0 from x_t based on the equation $x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}$.

Along with the forward pass in the neural network, we also modified the reverse diffusion and ELBO calculator methods within the DDPM class to take the target parameterization as an argument, incorporating conditional logic to dynamically compute the appropriate predictions and respective mean-squared error (MSE) based on the specified parameterization when calling the network (detailed in Appendices A.2 and A.3). Finally, we instantiated separate versions of the

network, DDPM, optimizer, and scheduler for each parameterization, ensuring that all model variations were trained under identical conditions, thereby enabling fair and reliable comparisons.

Results and Comparison The results for the different target parameterizations align closely with the findings in Ho et al. [2020]; the *ϵ-based DDPM* achieved the best generative quality, with the lowest FID score and evidence ELBO. The *μ-based DDPM* performs moderately well, with a higher FID score and slightly lower ELBO, suggesting a decline in generative quality compared to the base model. Visual inspection also showcases that predicting the mean of the reverse process introduces limitations in capturing small generative details, despite maintaining reasonable overall performance (Appendix A.4).

1.2 Guided Diffusion Models

In this section, we explore two methods for achieving conditional generation with diffusion models: **Classifier Guidance** Nichol and Dhariwal [2021] and **Classifier-Free Guidance**Ho and Salimans [2022]. Both approaches aim to control the generation process to produce specific digits, but they achieve this through different mechanisms, as outlined in Weng [2021]. In our experiments, we tested both methods using a range of guidance scales to evaluate their impact on the quality of the generated samples. The optimal guidance scale was used based on a combination of visual inspection and quantitative analysis, the results of which can be found in Table 2.

1.2.1 Classifier Guidance

The classifier guidance method extends the denoising diffusion probabilistic model (DDPM) [Ho et al., 2020] by incorporating gradients from a pre-trained classifier. The reverse process, defined in Eq. 1, is modified with a gradient term $\nabla_{x_t} \log p(y|x_t)$, where $p(y|x_t)$ is the classifier's predicted probability for class y. This guides the generation toward the target class (digit) [Nichol and Dhariwal, 2021].

Implementation Details: Classifier Guidance We extended the Denoising Diffusion Probabilistic Model (DDPM) [Ho et al., 2020] with classifier guidance as described in Nichol and Dhariwal [2021] and outlined in Algorithm 4. A classifier was trained on noisy images with Gaussian noise proportional to timestep t, following Ho et al. [2020]. Gradients of the classifier's log-probability, $\nabla_{x_t} \log p(y|x_t)$, were scaled by a guidance factor `guidancescale` and integrated into the reverse diffusion step for class-conditional sampling. A new class, `GuidedDDPM`, was introduced to compute these gradients and modify the reverse process accordingly.

1.2.2 Classifier-Free Guidance

Classifier-free guidance, introduced by Ho and Salimans [2022], enables conditional generation without a pre-trained classifier. Unlike classifier guidance [**?**], it trains a single diffusion model for both conditional and unconditional generation by randomly dropping conditioning information with probability p_{uncond}. During sampling, the model combines conditional and unconditional scores using $\epsilon_\theta^{\text{guided}} = \epsilon_\theta^{\text{uncond}} + w \cdot (\epsilon_\theta^{\text{cond}} - \epsilon_\theta^{\text{uncond}})$, where w controls the trade-off between sample quality and diversity Ho and Salimans [2022].

Implementation Details: Classifier-Free Guidance As outlined in Algorithm 5, we implemented classifier-free guidance by introducing `ConditionalScoreNet` and `CFGDDPM`. `ConditionalScoreNet` extends the provided U-Net-based score network with class conditioning via an embedding layer, using a null token for unconditional generation without changing the network architecture. `CFGDDPM` incorporates the diffusion process and computes conditional and unconditional scores during sampling. Following Ho and Salimans [2022], conditioning information was randomly dropped with probability $p_{\text{uncond}} = 0.01$. The reverse diffusion step combines these scores to produce guided samples.

Results and Comparison A comparison of classifier-guided (CG) and classifier-free (CFG) diffusion models highlights distinct strengths consistent with Nichol and Dhariwal [2021] and Ho and Salimans [2022]. For MNIST, CG achieves optimal performance at $w = 5$ (FID $= 55.26 \pm 1.22$), while CFG performs best at $w = 7$ (FID $= 56.10 \pm 0.62$), reflecting the trade-off between quality and diversity. The classifier guided approach, is better when using moderate guidance strengths, whereas classifier-free guidance requires higher w to achieve comparable quality. Beyond $w = 7$, performance degrades for both models, indicating overfitting and loss of diversity at high guidance. ELBO metrics further underscore classifier-guided's superior balance of reconstruction and distribution preservation at $w = 5$, while Inception Scores remain stable across methods due to MNIST's simplicity. Ultimately classifer-guided diffusion produces noticeably better digits.

1.3 Continuous Version of Diffusion Models

Continuous-time denoising diffusion probabilistic models (DDPMs) are an extension of the common framework of discrete-time models. While discrete diffusion models add noise in a fixed number of steps, a continuous model and, especially, Stochastic Differential Equation (SDE) formulation allows us to treat this process as a continuous transformation of data to noise. The advantages of using a continuous noising processes could include new, more efficient sampling procedures such as predictor-corrector framework as well as ability to compute model likelihood, as described and developed in [Song et al., 2021].

1.3.1 Diffusion Models with Variance Exploding, Variance Preserving, and Sub Variance Preserving SDEs

Following [Song et al., 2021], we are investigating the effects of employing different noising processes (different SDEs):

Variance Preserving (VP) SDE maintains a stable variance of the data distribution during diffusion. Its forward process is formulated as: $dx_t = -\frac{1}{2}\beta(t)x_t\,dt + \sqrt{\beta(t)}\,dW_t$.

Variance Exploding (VE) SDE adds noise with increasing variance over time, meaning an exponential growth of noise over time. The forward process is defined as: $dx_t = \sqrt{2\beta(t)}\,dW_t$, where $\beta(t)$ is a time-dependent noise coefficient and W_t is a Wiener process.

Sub Variance Preserving (SUBVP) offers a trade-off between VE and VP, balancing noise control and flexibility. The forward process is given by: $dx_t = -\alpha(t)x_t\,dt + \sqrt{\beta(t)}\,dW_t$, where $\alpha(t)$ and $\beta(t)$ are time-varying parameters.

Implementation Details The implementation required modifying the original DDPM code by adding three different SDE formulations (VP, VE, and SUBVP) with their respective marginal probability and diffusion coefficient functions. The sampling process was changed to use either a Predictor-Corrector sampler or an ODE (used in the report as produced better results), and the loss function to handle the different SDE types with appropriate noise schedules.

Results and Comparison Looking at the results of continuous DDPM, the samples from the VE SDE seem to be the best among the three methods. Also, they seem to be more clear and of a better quality that the result of the base model, but with thinner digits. However, on the quantitative scores from Tab. 1 they perform worse. A reason of that could be that they are less similar to the real images. Regarding the comparison between the three SDE models, it agrees with what [Song et al., 2021] found, that when using a black-box ODE sampler, the samples from VE SDEs typically have a better quality that those from VP/SUBVP SDEs. Looking at both samples from Fig. 5 and the quantitative metrics, it is clear that the VE SDE samples are more readable. All the results show a great potential of the continuous model. Due to time constraints, we did not complete a full hyperparameter search, however, it was visible that some of the samples had more clear digits than others, but noisier background.

1.4 Quantitative Performance Comparison

Method	FID	Inception Score	Log-Likelihood (bits/dim)	Training Time (minutes)
Base DDPM	59.69 ± 0.91	2.09 ± 0.04	-23.7633 ± 0.3650 (*)	09:59
DDPM μ	73.86 ± 0.69	2.07 ± 0.02	-26.1582 ± 0.7096 (*)	10:26
DDPM x_0	117.58 ± 0.75	2.23 ± 0.05	-99.6665 ± 8.6140 (*)	10:02
VP SDE	201.54 ± 8.92	2.09 ± 0.02	4.86	08:27
VE SDE	164.53 ± 6.45	1.95 ± 0.03	4.66	08:25
Sub-VP SDE	329.71 ± 11.28	1.50 ± 0.01	7.11	08:32
Classifier Guided DDPM	55.26 ± 1.22	2.11 ± 0.02	-24.5517 ± 0.3285 (*)	09:58
Classifier-Free DDPM	56.10 ± 0.62	2.07 ± 0.03	-24.3048 ± 0.2923 (*)	11:58

Table 1: Performance comparison of various DDPM variants and extensions. Results are presented as mean ± standard deviation where applicable *(computed over 5 sampling runs)*

(*) *Using Evidence ELBO as a proxy for likelihood*

Likelihood as a metric shows mixed utility across different types of diffusion models. For Score-Based Generative Models (SBGM) using Stochastic Differential Equations (SDEs), it works exceptionally well Song et al. [2021]. These models allow exact likelihood computation through reverse-time probability flow ODEs, directly connecting likelihood to the model's ability to capture the true data distribution. However, for methods like Classifier Guidance and Classifier-Free Guidance, likelihood is a less meaningful metricHo and Salimans [2022] Nichol and Dhariwal [2021].

These approaches focus on balancing sample diversity and fidelity, often trading off a perfect fit to the distribution for improved quality. Nichol and Dhariwal [2021]. Additionally, it is intractable to compute the exact likelihood for discrete diffusion models, as it would require integrating over the whole latent space, which is computationally infeasible. As such, a common alternative is to use Evidence ELBO (Evidence Lower Bound) as a proxy for log-likelihood, by computing the forward and reverse transitions for each timestep t, and approximating the expectations using Monte Carlo sampling Jiang [2021]. While likelihood is a powerful and interpretable metric in SDE-based models, its relevance diminishes in approaches that prioritize trade-offs or visual quality over probabilistic alignment.

Results on all models The results demonstrate that while the base DDPM model performed reasonably well, the ϵ-based parameterization achieved superior generative quality with the lowest FID and ELBO. Classifier-guided and classifier-free approaches showed trade-offs, with classifier-guided diffusion achieving better reconstruction and quality at moderate guidance levels. Continuous DDPMs, particularly VE SDEs, produced visually clearer samples but scored worse quantitatively, likely due to reduced similarity to real images, highlighting the promise of continuous models despite some trade-offs.

2 Function Fitting with Gaussian Processes (GP)

In this section, we extend the Gaussian Process (GP) model to address two tasks: (1) fitting a standard GP to a target function $g(x)$ with a fixed noise term $\epsilon \sim \mathcal{N}(0, 0.01)$ by experimenting with MAP and NUTS for kernel hyperparameter tuning, and (2) extending the GP to include an integral constraint.

2.1 Fitting a Standard GP

Kernel The function $g(x) = -(\sin(6\pi x))^2 + 6x^2 - 5x^4 + \frac{3}{2}$ exhibits a sine term $-(\sin(6\pi x))^2$ which induces periodic oscillations, and a polynomial term $6x^2 - 5x^4 + \frac{3}{2}$ that introduces more complex nonlinear trends. To accurately model the behavior of the target function, we experimented with Gaussian Process (GP) regression using a combination of periodic and polynomial kernels, and a squared exponential kernel (RBF). We observed that the standard **RBF kernel** $k(x, x') = \sigma^2 \exp\left(-0.5 \times \frac{|x-x'|^2}{l^2}\right)$ was particularly effective due to its flexibility in capturing and modeling smooth fluctuations. It accurately represents both the periodic and nonlinear features of $g(x) + \epsilon$, outperforming the more specialized kernel combination, where we combined an RBF kernel with a polynomial one.

Parameters The RBF kernel introduces two variable hyperparameters to the GP model: **variance** (σ^2) and **lengthscale** (l^2), in addition to the fixed noise variance parameter (ϵ). We opted to model the kernel hyperparameters following a **log-normal** prior distribution, motivated by both theoretical and practical considerations. As they are inherently non-negative, they represent a suitable choice for these kernel parameters, ensuring that the positivity requirements of the GP model are satisfied while avoiding overly tight constraints. For variance, the prior is centered around the log of the observed variance in $g(x)$, aligning with the data's inherent variability. For lengthscale, the prior captures the scale of the function's oscillatory behavior, centered around the log of the computed periodicity of the target function.

$$\sigma^2 \sim \text{LogNormal}(\mu = -0.64, \sigma = 0.5)$$
$$l \sim \text{LogNormal}(\mu = -2.94, \sigma = 0.5)$$

We can formulate the chosen probabilistic model as:

$$p(\theta, \mathbf{y} \mid \mathbf{X}) \propto \mathcal{N}\left(\mathbf{y} \mid \mathbf{0}, \mathbf{K} + \sigma_n^2 \mathbf{I}\right) \times \text{LogNormal}\left(\sigma_k^2 \mid -0.64, 0.5^2\right) \times \text{LogNormal}\left(l \mid -2.94, 0.5^2\right),$$

$$\text{where } K_{ij} = \sigma_k^2 \exp\left(-\frac{1}{2}\frac{|x_i - x_j|^2}{l^2}\right).$$

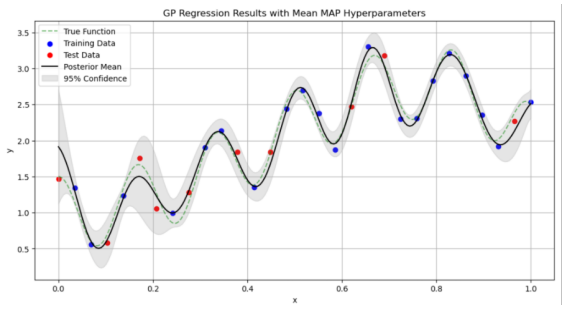
Approximate Posterior Likelihood The posterior predictive distribution captures both mean predictions and uncertainties through kernel-weighted combinations, with Cholesky decomposition ensuring numerical stability Rasmussen and Williams [2006] having:

$$p(y_* \mid x_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(y_* \mid \mu_*, \sigma_*^2), \quad \mu_* = \mathbf{k}_*^\top (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{y}, \quad \sigma_*^2 = k_{**} - \mathbf{k}_*^\top (\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{k}_*.$$

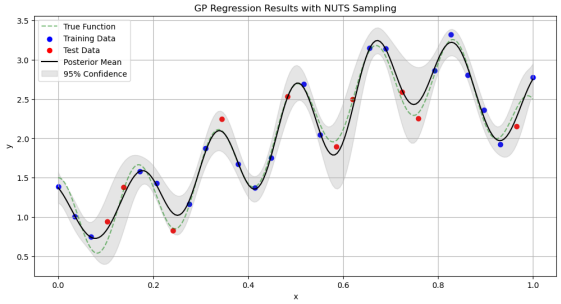
GP Regression with MAP For MAP we optimized the hyperparameters using gradient descent and backpropagation, minimizing the negative log marginal likelihood (NLL) as the loss function as per Team [n.d.] documentation.

GP Regression with NUTS For the NUTS approach, we used ArviZ diagnostics (Appendix B.12) to fine-tune the sampler parameters, selecting 500 warm-up steps and 2 parallel chains as an optimal balance between output quality and computational efficiency, since 1000 warm-up steps and 3 chains provided the same results. We then sampled 500 kernel hyperparameter values from the priors using the No-U-Turn Sampler, leveraging Hamiltonian Monte Carlo for effective posterior exploration.

Results Both MAP and NUTS produced aligned optimal kernel lengthscales (0.066), but differed in kernel variance, with NUTS estimating slightly higher values (1.842 ± 0.100) than MAP (1.641 ± 0.064). In Figure 1 we observe that both approaches demonstrate strong performance in fitting the target function based on the averaged optimal parameters obtained across 20 runs. Nonetheless, MAP achieved better average test log likelihood (0.394 ± 3.821) compared to NUTS (2.345 ± 4.907), though both exhibited high standard deviation. This suggests MAP offers better predictive performance, while NUTS provides broader posterior exploration, valuable for uncertainty quantification.



(a) GP Regression Results with the Mean of the MAP Hyperparameters after 20 Runs



(b) GP Regression Results with Mean NUTS Hyperparameters after 20 Runs

Figure 1: Comparison of GP Regression Results with Mean Hyperparameters for MAP and NUTS after 20 Runs.

2.2 Fitting a GP with Integral Constraints

Next, we introduce an integral constraint to the problem. We assume a GP prior $f \sim GP(0, k(\cdot,\cdot))$, with the kernel parameters estimated from the MAP approach from the first task. Since exact integration of an integral constraint is infeasible, we approximate it using the trapezoidal rule shown by the below:

$$q = \int_0^1 f(x)\,dx = 2, \quad q \approx \hat{q} = \sum_{i=1}^{\ell} w_i f(x_i), \quad w_i = \begin{cases} \frac{1}{2\ell-2}, & \text{for } i \in \{1,\ell\}, \\ \frac{1}{\ell-1}, & \text{otherwise.} \end{cases} \tag{3}$$

2.2.1 Proof that the random variable $(\hat{q}, f)|X$ is normally distributed

Recall that the function f follows a Gaussian Process $f \sim \mathcal{GP}(0, k(\cdot,\cdot))$. Given input points X, the function values $f|X$ follow a multivariate normal distribution $f|X \sim \mathcal{N}(0, K)$, where K is the kernel matrix with elements $K_{ij} = k(x_i, x_j)$. This follows from the definition of $\mathcal{GP}$ - any finite collection of points from a $\mathcal{GP}$ is multivariate normally distributed.

The trapezoidal rule approximates the integral constraint (eq. 9) as $\hat{q} = \sum_{i=1}^{\ell} w_i f(x_i)$, where the weights w_i are given by eq. 3.

We can, express $(\hat{q}, f)|X$ as a linear transformation of f, and write the transformation matrix A as an $(\ell+1) \times \ell$ matrix, both shown below:

$$\begin{bmatrix} \hat{q} \\ f \end{bmatrix} = A \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_\ell \end{bmatrix}, \quad A = \begin{bmatrix} w_1 & w_2 & \dots & w_\ell \\ 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}. \tag{4}$$

The first row of A contains the trapezoidal weights for $\hat{q}$, while the remaining ℓ rows form an identity matrix for f.

Since $f|X \sim \mathcal{N}(0, K)$, and $(\hat{q}, f)|X$ is a linear transformation of f, we apply the standard result for multivariate normal transformations with key property that if $Y = AX$ where $X \sim \mathcal{N}(\mu, \Sigma)$, then $Y \sim \mathcal{N}(A\mu, A\Sigma A^\top)$. This follows from two fundamental rules of normal distributions: linear combinations of normal variables are normal and covariances transform quadratically under linear maps.

Applying this to our case: $(\hat{q}, f)|X \sim \mathcal{N}(\mu, \Sigma)$ where: $\mu = A \cdot 0 = 0$ (zero vector of length $\ell + 1$) and $\Sigma = AKA^\top$. Thus, we have proven that $(\hat{q}, f)|X$ follows a normal distribution, as it is a linear transformation of a normally distributed random vector $f|X$.

2.3 Derivation of the Probability Distribution of $f|X, \hat{q}$

Since we have established that $(\hat{q}, f)|X \sim \mathcal{N}(0, \Sigma)$, we now derive the conditional distribution $f|X, \hat{q}$ using the standard result for conditioning multivariate normal distributions. We will use the property of if a joint Gaussian distribution is given by:

$$\begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \right),$$

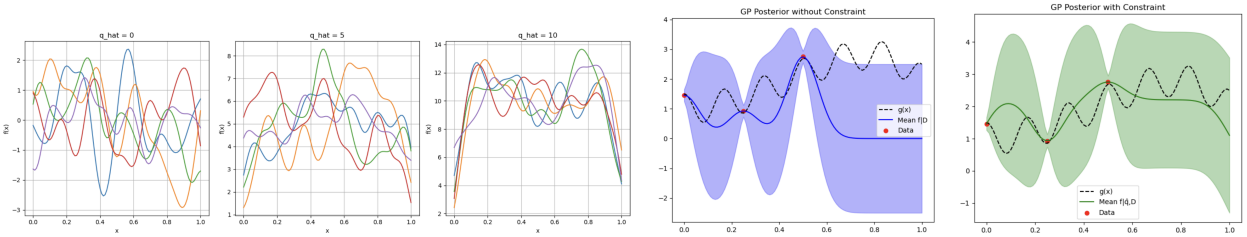
then the conditional distribution $Y_2|Y_1$ follows $Y_2|Y_1 \sim \mathcal{N}\left(\mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(Y_1 - \mu_1), \quad \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}\right)$.

In our case, we set $Y_1 = \hat{q}$, $Y_2 = f$, $\mu_1 = 0$, $\mu_2 = 0$, $\Sigma_{11} = w^\top K w$ (scalar), $\Sigma_{12} = w^\top K$ (row vector), $\Sigma_{21} = Kw$ (column vector), $\Sigma_{22} = K$ (full covariance matrix). Using the standard conditional normal formula, we obtain $\mathbb{E}[f|X, \hat{q}] = Kw(w^\top K w)^{-1}\hat{q}$ and $\text{Cov}(f|X, \hat{q}) = K - Kw(w^\top K w)^{-1}w^\top K$. Thus, the conditional distribution is $f|X, \hat{q} \sim \mathcal{N}\left(Kw(w^\top K w)^{-1}\hat{q}, \quad K - Kw(w^\top K w)^{-1}w^\top K\right)$.

2.4 Discussion on the Rank of Cov$(f|X, \hat{q})$

The covariance matrix $\text{Cov}(f|X, \hat{q})$ is obtained by subtracting a rank-1 update $Kw(w^\top K w)^{-1}w^\top K$ from K. Since K is positive semi-definite and we're subtracting an outer product, this tells us that. The rank drops by exactly 1 because we are forcing $\int f(x)dx = \hat{q}$. It creates a linear relationship between all the function values. From a mathematical perspective, we observe a loss of precisely one dimension in the space of possible functions. After fixing $\ell - 1$ function values and the integral constraint, the final value becomes fully determined. (The above is also verified by computing the rank using `torch.linalg.matrix_rank`. Thus, the covariance matrix $\text{Cov}(f|X, \hat{q})$ is rank-deficient by 1.

2.5 Bayesian Inference and Gaussian Process Posterior Computation



(a) Five samples from $f \mid X, \hat{q}$ for $\ell = 101$ and different choices for, respectively, $\hat{q} = \{0, 5, 10\}$

(b) The posterior $f \mid \mathcal{D}$ with $\ell = 101$ and $\hat{q} = 2$

(c) The posterior $f \mid \hat{q}, \mathcal{D}$ with $\ell = 101$ and $\hat{q} = 2$

Figure 2: Results showing influence of integral constraints.

Plot (a) shows how different integral constraints affect the sampled functions. As $\hat{q}$ increases, the overall magnitude of the functions increases, and the samples become more constrained and similar to each other, visible at $\hat{q} = 10$. This demonstrates how the integral constraint influences the behavior of the Gaussian process by restricting the space of possible functions to those that satisfy the required integral value. This constraining effect is further illustrated in the posterior plots, where incorporating the integral constraint ($\hat{q} = 2$) affects the GP posterior. The constrained version (c) shows reduced uncertainty compared to the unconstrained version (b) and appears to follow the true function $g(x)$ more accurately, suggesting that the integral constraint helps to regularize the GP predictions.

References

Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2021. doi:10.48550/arXiv.2011.13456. URL https://arxiv.org/abs/2011.13456.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in neural information processing systems*, 33:6840–6851, 2020. doi:10.48550/arXiv.2006.11239. URL https://arxiv.org/abs/2006.11239.

Alexander Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. *arXiv preprint arXiv:2102.09672*, 2021. doi:10.48550/arXiv.2102.09672. URL https://arxiv.org/abs/2102.09672.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022. doi:10.48550/arXiv.2207.12598. URL https://arxiv.org/abs/2207.12598.

Lilian Weng. What are diffusion models?, 2021. URL https://lilianweng.github.io/posts/2021-07-11-diffusion-models/.

Yunfan Jiang. Understanding elbo (evidence lower bound), 2021. URL https://yunfanj.com/blog/2021/01/11/ELBO.html. Accessed: 2025-01-14.

Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, Cambridge, MA, USA, 2006. ISBN 026218253X. URL http://www.GaussianProcess.org/gpml.

Pyro Team. Maximum likelihood and maximum a posteriori estimation, n.d. URL https://pyro.ai/examples/mle_map.html. Accessed: 2025-01-17.

Appendix A: Quantitative Analysis of Diffusion Models

A.1 Neural Network Forward Pass: Post-Processing Step for Different Target Parameterizations

Algorithm 1 Postprocessing in UNET for Target Parameterizations ϵ, μ or x_0

Require: x (noisy input at timestep t), t (current timestep)
Require: Optional: α_t, β_t, $\alpha_{\text{bar},t}$ (diffusion parameters)
Require: target: Prediction target *'eps'*, *'mu'*, or *'x0'* (default: *'eps'*)
Ensure: Predicted output based on the target parameterization
procedure POSTPROCESSING($x, t, \alpha_t, \beta_t, \alpha_{\text{bar},t}$, target)
  if *target* $=$ *'eps'* **then**
    Default: Predict noise level
    return h
  target $=$ 'mu' ▷ Predict μ using diffusion formulas if α_t, β_t, or $\alpha_{\text{bar},t}$ are not provided. **if** α_t, β_t, *or* $\alpha_{bar,t}$ *are not provided* **then**
    return ValueError("α_t, β_t, and $\alpha_{\text{bar},t}$ must be provided for target='mu'")
  Broadcast α_t, β_t, and $\alpha_{\text{bar},t}$ to match spatial dimensions
  $\alpha_t \leftarrow \alpha_t[:, :, :, :]$, $\beta_t \leftarrow \beta_t[:, :, :, :]$, $\alpha_{\text{bar},t} \leftarrow \alpha_{\text{bar},t}[:, :, :, :]$
  $\mu \leftarrow \frac{1}{\sqrt{\alpha_t}} \cdot \left(x - \frac{\beta_t}{\sqrt{1-\alpha_{\text{bar},t}}} \cdot h \right)$
  return μ
  target $=$ 'x0' ▷ Predict x_0 using diffusion formulas if $\alpha_{\text{bar},t}$ is not provided. **if** $\alpha_{bar,t}$ *is not provided* **then**
    return ValueError("$\alpha_{\text{bar},t}$ must be provided for target='x0'")
  Reshape $\alpha_{\text{bar},t}$ to [batch_size, 1, 1, 1]
  $x_0 \leftarrow \frac{x - \sqrt{1-\alpha_{\text{bar},t}} \cdot h}{\sqrt{\alpha_{\text{bar},t}}}$
  return x_0
end procedure

A.2 Reverse Diffusion Process for Different Target Parameterizations

Algorithm 2 Reverse Diffusion Step: $p(x_{t-1}|x_t)$ in DDPM for Target Parameterizations ϵ, μ or x_0

Require: x_t (input at timestep t), t (current timestep), ϵ (random noise)
Require: Optional: target (Prediction target: *'eps'*, *'mu'*, or *'x0'*; default: *'eps'*)
Ensure: Mean and standard deviation for reverse diffusion

procedure REVERSEDIFFUSION(x_t, t, ϵ, target)
  $t_{\text{int}} \leftarrow t$.squeeze().long()
  $\alpha_t \leftarrow \alpha[t_{\text{int}}]$.view$(-1, 1)$.to($x_t$.device)
  $\beta_t \leftarrow \beta[t_{\text{int}}]$.view$(-1, 1)$.to($x_t$.device)
  $\alpha_{\text{bar},t} \leftarrow \alpha_{\text{bar}}[t_{\text{int}}]$.view$(-1, 1)$.to($x_t$.device)
  if *target* $=$ *'eps'* **then**
  Predict noise epsilon (default)
  pred $\leftarrow$ network(x_t, t)
  target $=$ 'mu' ▷ Predict mean μ
  pred $\leftarrow$ _network$\big(x_t$.view$(-1, 1, 28, 28)$, $\frac{t}{T}, \alpha_t, \beta_t, \alpha_{\text{bar},t}$, 'mu'$\big)$.view$(-1, 28 \times 28)$
  target $=$ 'x0' ▷ Predict x_0
  pred $\leftarrow$ _network$\big(x_t$.view$(-1, 1, 28, 28)$, $\frac{t}{T}, \alpha_{\text{bar},t}$, 'x0'$\big)$.view$(-1, 28 \times 28)$
  else
  return *ValueError("Unsupported target parameterization: target")*
  mean $\leftarrow \frac{1}{\sqrt{\alpha_t}} \cdot \big(x_t - \frac{\beta_t}{\sqrt{1-\alpha_{\text{bar},t}}} \cdot \text{pred}\big)$
  std $\leftarrow$ torch.where$\big(t > 1, \sqrt{\frac{1-\alpha_{\text{bar},t}}{1-\alpha_{\text{bar},t}} \cdot \beta_t}$, torch.zeros_like$(\beta_t)\big)$
  return mean $+$ std $\cdot\, \epsilon$
end procedure

A.3 ELBO Calculator for Different Target Parameterizations

Algorithm 3 ELBO Training Objective in DDPM for Target Parameterizations ϵ, μ or x_0

Require: x_0 (input image), target (prediction target: *'eps'*, *'x0'*, *'mu'*)
Ensure: ELBO value

procedure ELBO(x_0, target)
  $t \leftarrow$ Random integer in $[1, T]$
  $\epsilon \leftarrow$ Random noise with same shape as x_0
  $x_t \leftarrow$ ForwardDiffusion(x_0, t, ϵ)
  if *target* $=$ *'eps'* **then**
  Predict noise ϵ
  $\epsilon_{\text{pred}} \leftarrow$ Network(x_t, t)
  return $-$MSELoss$(\epsilon, \epsilon_{\text{pred}})$
  target $=$ 'x0' ▷ Predict x_0
  $x_0^{\text{pred}} \leftarrow$ Network$(x_t, t, \alpha_{\text{bar},t}$, 'x0')
  return $-$MSELoss(x_0, x_0^{pred})
  target $=$ 'mu' ▷ Predict μ
  $\mu_{\text{pred}} \leftarrow$ Network$(x_t, t, \alpha_t, \beta_t, \alpha_{\text{bar},t}$, 'mu')
  $\mu_{\text{true}} \leftarrow \frac{1}{\sqrt{\alpha_t}} \cdot \big(x_t - \frac{\beta_t}{\sqrt{1-\alpha_{\text{bar},t}}} \cdot \epsilon\big)$
  return $-$MSELoss$(\mu_{\text{pred}}, \mu_{\text{true}})$
  else
  return *ValueError("Unsupported prediction type: target")*
end procedure

A.4 Visual Comparison of Samples from Different Target Parameterizations

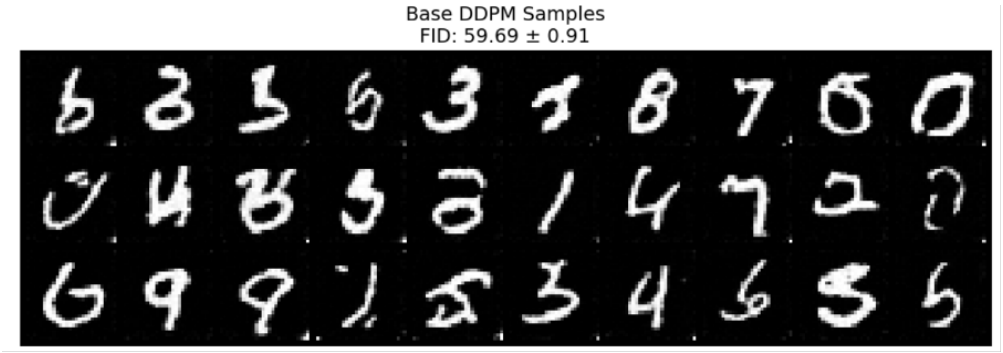
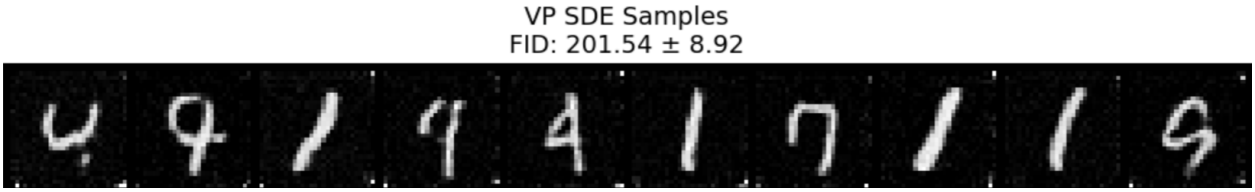
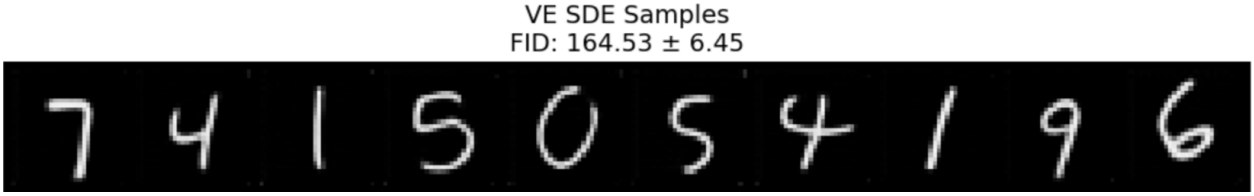


Figure 3: Base DDPM Samples (FID: 59.69 $\pm$ 0.91)

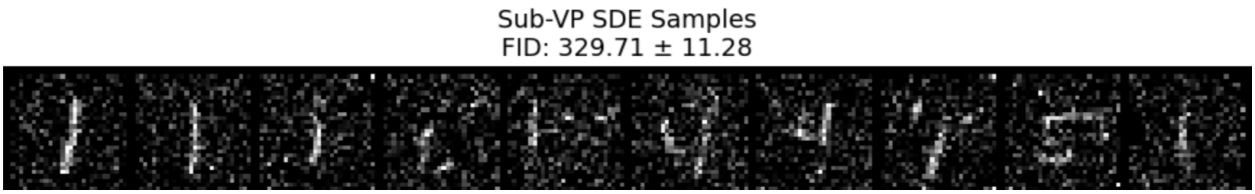
Figure 4: Samples Generated from DDPMs trained on μ and x_0 target parameterizations



(a)



(b)



(c)

Figure 5: Samples generated from Continuous DDPM using SDE, showing, respectively, samples from VP SDE, VE SPE and SUBVP SDE using an ODE sampler.

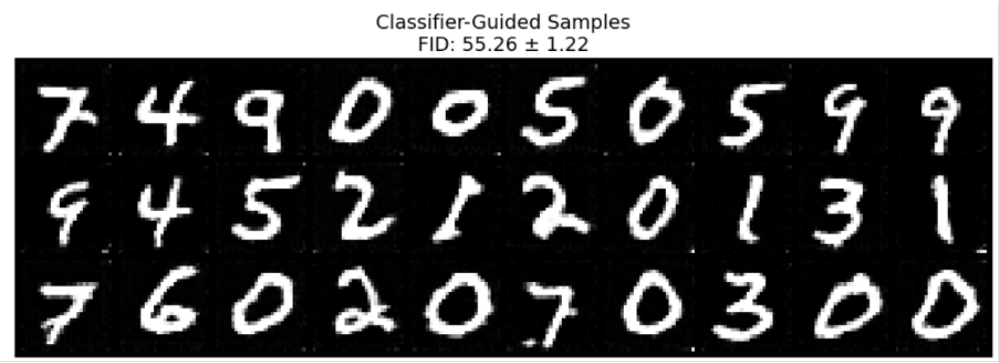


Figure 6: Classifier-Guided Samples (FID: 55.26 $\pm$ 1.22)

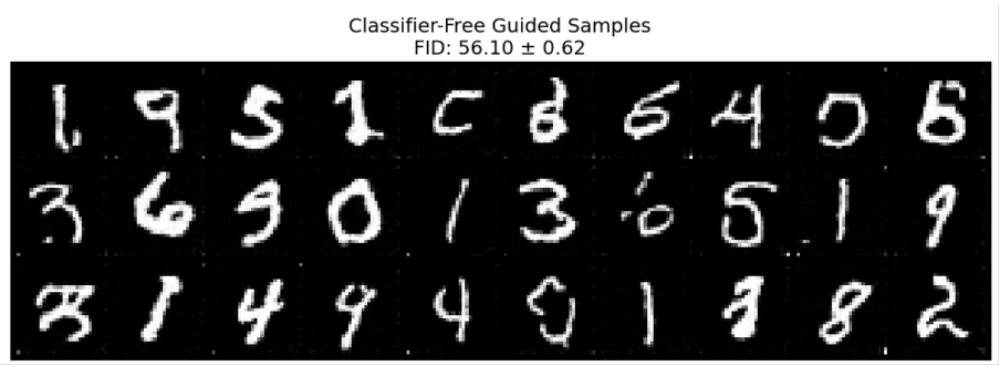


Figure 7: Classifier-Free Guided Samples (FID: 56.10 $\pm$ 0.62)

A.5 Guidance Free and Guided Diffusion

Algorithm 4 Classifier-Guided Diffusion with Improved Scaling

Require: x_t (noisy image), t (timestep), y (class label), γ (guidance scale)
Ensure: Next step prediction x_{t-1}

procedure GUIDEDREVERSESTEP(x_t, t, y, γ)
  $\epsilon_\theta \leftarrow \text{Network}(x_t, t)$
  $x_t^{\text{detach}} \leftarrow \text{StopGradient}(x_t)$
  logits $\leftarrow \text{Classifier}(x_t^{\text{detach}}, t)$
  log_probs $\leftarrow \text{LogSoftmax}(\text{logits})$
  $\nabla_{x_t} \log p(y|x_t) \leftarrow \nabla_{x_t} \text{log\_probs}[y]$
  scale $\leftarrow \gamma\sqrt{\alpha_{\text{bar},t}}$
  $\epsilon_{\text{guided}} \leftarrow \epsilon_\theta + \text{scale} \cdot \nabla_{x_t} \log p(y|x_t)$
  $\mu_t \leftarrow \frac{1}{\sqrt{\alpha_t}} \cdot (x_t - \frac{1-\alpha_t}{\sqrt{1-\alpha_{\text{bar},t}}} \cdot \epsilon_{\text{guided}})$ **if** $t > 1$ **then**
  $\sigma_t \leftarrow \sqrt{\frac{(1-\alpha_{\text{bar},t-1})}{(1-\alpha_{\text{bar},t})}\beta_t}$ **else**
  $\sigma_t \leftarrow 0$
  $\epsilon \leftarrow \mathcal{N}(0, I)$ if $t > 1$ else 0
  return $\mu_t + \sigma_t \cdot \epsilon$
end procedure

Algorithm 5 Classifier-Free Guided Diffusion with Joint Training

Require: x_t (noisy image), t (timestep), y (class label), w (guidance weight), p_{uncond} (dropout probability)
Ensure: Next step prediction x_{t-1}

procedure CFGFORWARD(x_t, t, y) **if** *training and Random*$(0,1) < p_{uncond}$ **then**
  $y \leftarrow$ None ▷ Random conditioning dropout

  return Network(x_t, t, y)

  procedure CFGREVERSESTEP(x_t, t, y, w)
    $\epsilon_{\theta,c} \leftarrow$ Network(x_t, t, y) ▷ Conditional
    $\epsilon_{\theta,u} \leftarrow$ Network$(x_t, t,$ None$)$ ▷ Unconditional
    $\epsilon_{\text{guided}} \leftarrow (1+w)\epsilon_{\theta,c} - w\epsilon_{\theta,u}$
    $\mu_t \leftarrow \frac{1}{\sqrt{\alpha_t}} \cdot (x_t - \frac{\beta_t}{\sqrt{1-\alpha_{\text{bar},t}}} \cdot \epsilon_{\text{guided}})$ **if** $t > 1$ **then**
    $\sigma_t \leftarrow \sqrt{\frac{(1-\alpha_{\text{bar},t-1})}{(1-\alpha_{\text{bar},t})}\beta_t}$ **else**
    $\sigma_t \leftarrow 0$
    $\epsilon \leftarrow \mathcal{N}(0, I)$ if $t > 1$ else 0
    return $\mu_t + \sigma_t \cdot \epsilon$
  end procedure=0

A.6 Training Variance-Preserving (VP) SDE

Algorithm 6 Train VP-SDE

Input: score_model, dataset, num_epochs, batch_size, optimizer
Output: Trained score model

Initialize DataLoader for dataset
for *epoch = 1* ***to*** *num_epochs* **do**
  for *batch* ***in*** *dataset* **do**
    Sample $t \sim \mathcal{U}(0,1)$
    Sample $\epsilon \sim \mathcal{N}(0, I)$
    $x_t \leftarrow \sqrt{\bar{\alpha}_t}x + \sqrt{1-\bar{\alpha}_t}\epsilon$
    $\epsilon_{\text{pred}} \leftarrow$ score_model(x_t, t)
    Loss $\leftarrow$ MSE$(\epsilon, \epsilon_{\text{pred}})$
    optimizer.step()
return score_model

A.7 Training Variance-Exploding (VE) SDE

Algorithm 7 Train VE-SDE

Input: score_model, dataset, num_epochs, batch_size, optimizer
Output: Trained score model
Initialize DataLoader for dataset
for *epoch = 1* ***to*** *num_epochs* **do**
  for *batch* ***in*** *dataset* **do**
    Sample $t \sim \mathcal{U}(0, 1)$
    Sample $\epsilon \sim \mathcal{N}(0, I)$
    $\sigma_t \leftarrow \sigma\sqrt{e^{2t} - 1}$
    $x_t \leftarrow x + \sigma_t \cdot \epsilon$
    $\epsilon_{\text{pred}} \leftarrow$ score_model(x_t, t)
    Loss $\leftarrow$ MSE$(\epsilon, \epsilon_{\text{pred}})$
    optimizer.step()
return score_model

A.8 Training Sub-VP SDE

Algorithm 8 Train SubVP-SDE

Input: score_model, dataset, num_epochs, batch_size, optimizer
Output: Trained score model
Initialize DataLoader for dataset
for *epoch = 1* ***to*** *num_epochs* **do**
  for *batch* ***in*** *dataset* **do**
    Sample $t \sim \mathcal{U}(0, 1)$
    Sample $\epsilon \sim \mathcal{N}(0, I)$
    $\beta_t \leftarrow \beta_{\text{min}} + (\beta_{\text{max}} - \beta_{\text{min}}) \cdot t$
    $\sigma_t \leftarrow \sqrt{1 - e^{-2\beta_t}}$
    $x_t \leftarrow \sqrt{\bar{\alpha}_t}x + \sqrt{1 - \bar{\alpha}_t}\epsilon$
    $\epsilon_{\text{pred}} \leftarrow$ score_model(x_t, t)
    Loss $\leftarrow$ MSE$(\epsilon, \epsilon_{\text{pred}})$
    optimizer.step()
return score_model

A.9 ODE Sampler

Algorithm 9 ODE Sampler

Input: score_model, x_T, num_steps, sde_type
Output: Sampled image x_0
$\Delta t \leftarrow 1/$num_steps
  $x \leftarrow x_T$ `// Initialize from Gaussian noise`
`// Define ODE function`
`ODE_func`(t, x): $\epsilon_{\text{pred}} \leftarrow$ score_model(x, t)
if *sde_type* == *'vp'* **then**
  return $-(1/\sqrt{\alpha_t}) \cdot (x - (\beta_t/\sqrt{1 - \bar{\alpha}_t}) \cdot \epsilon_{\text{pred}})$
else
  if *sde_type* == *'ve'* **then**
    return $-\beta_t \cdot \epsilon_{\text{pred}}$
  else
    return $-(1/\sqrt{\alpha_t}) \cdot (x - (\beta_t/\sqrt{1 - \bar{\alpha}_t}) \cdot \epsilon_{\text{pred}})$
`// Solve ODE using numerical integration (e.g., Runge-Kutta)`
Integrate `ODE_func` from $t = 1$ to $t = 0$
return x

A.10 PC (Predictor-Corrector) Sampler

Algorithm 10 PC Sampler

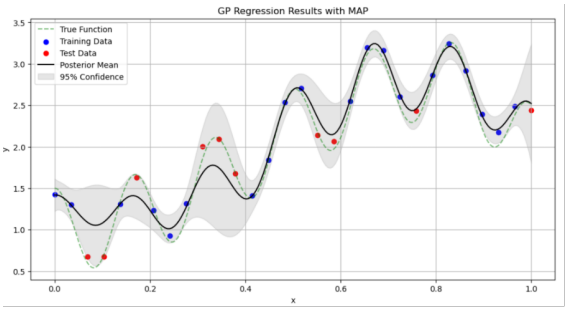
Input: score_model, x_T, num_steps, sde_type
Output: Sampled image x_0

$\Delta t \leftarrow 1/\text{num\_steps}$
$x \leftarrow x_T$ `// Initialize from Gaussian noise`
for $t \leftarrow$ *num_steps* ***down to*** *1* **do**
  `// Predictor Step (Euler-Maruyama)`
  $\epsilon_{\text{pred}} \leftarrow \text{score\_model}(x, t)$
  if *sde_type* == *'vp'* **then**
    $\mu_t \leftarrow (1/\sqrt{\alpha_t}) \cdot (x - (\beta_t/\sqrt{1-\bar{\alpha}_t}) \cdot \epsilon_{\text{pred}})$
  else
    if *sde_type* == *'ve'* **then**
      $\mu_t \leftarrow x + \beta_t \cdot \epsilon_{\text{pred}} \cdot \Delta t$
    else
      $\mu_t \leftarrow (1/\sqrt{\alpha_t}) \cdot (x - (\beta_t/\sqrt{1-\bar{\alpha}_t}) \cdot \epsilon_{\text{pred}})$
  `// Corrector Step (Langevin Dynamics)`
  for $i = 1$ ***to*** *num_corrector_steps* **do**
    SAMPLE $\epsilon \sim \mathcal{N}(0, I)$ $x \leftarrow x + \lambda \cdot \epsilon_{\text{pred}} + \sqrt{2\lambda} \cdot \epsilon$
  `// Final update`
  $x \leftarrow \mu_t$
return x

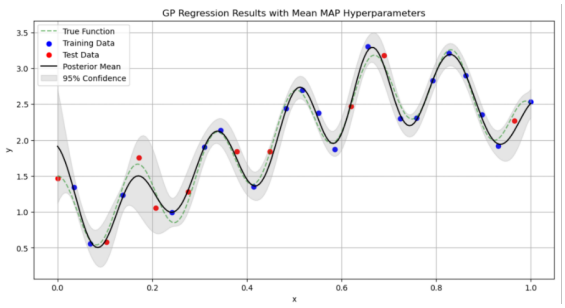
Metric/Method	w = 0.5	w = 1.0	w = 3.0	w = 5.0	w = 7.0
FID					
Base DDPM	58.69 ± 1.33	59.37 ± 0.55	58.63 ± 0.84	59.69 ± 0.91	59.80 ± 0.95
Classifier-Guided	51.67 ± 1.04	55.94 ± 1.14	60.54 ± 1.27	55.26 ± 1.22	53.16 ± 0.78
Classifier-Free	60.60 ± 0.60	59.46 ± 1.67	65.64 ± 1.57	55.75 ± 0.75	56.10 ± 0.62
Inception Score					
Base DDPM	2.12 ± 0.03	2.12 ± 0.02	2.09 ± 0.00	2.11 ± 0.04	2.12 ± 0.04
Classifier-Guided	2.11 ± 0.03	2.06 ± 0.05	1.99 ± 0.03	2.09 ± 0.06	2.06 ± 0.02
Classifier-Free	2.08 ± 0.03	2.06 ± 0.05	2.05 ± 0.04	2.10 ± 0.04	2.03 ± 0.04
Evidence ELBO (bits/dim)					
Base DDPM	-24.9530 ± 0.2280	-24.8798 ± 0.2675	-24.7989 ± 0.2763	-25.0754 ± 0.2566	-24.6546 ± 0.5010
Classifier-Guided	-24.7536 ± 0.3509	-24.6301 ± 0.4155	-24.3537 ± 0.3202	-24.5517 ± 0.3285	-24.4900 ± 0.3368
Classifier-Free	-25.0973 ± 0.5263	-25.4572 ± 0.1880	-25.0535 ± 0.4615	-24.8712 ± 0.3391	-25.2718 ± 0.2958
Reconstruction ELBO (bits/dim)					
Base DDPM	-23.9881 ± 0.2286	-23.9104 ± 0.2720	-23.8260 ± 0.2782	-24.1106 ± 0.2590	-23.6841 ± 0.5063
Classifier-Guided	-23.7888 ± 0.3559	-23.6621 ± 0.4150	-23.3858 ± 0.3247	-23.5818 ± 0.3270	-23.5190 ± 0.3347
Classifier-Free	-24.1303 ± 0.5304	-24.4912 ± 0.1896	-24.0843 ± 0.4608	-23.9035 ± 0.3381	-24.3048 ± 0.2923

Table 2: Comparison of FID, Inception Score, Evidence ELBO, and Reconstruction ELBO across Base DDPM, Classifier-Free Guidance, and Classifier-Guided Diffusion for various Guidance Scales (w) across five runs.

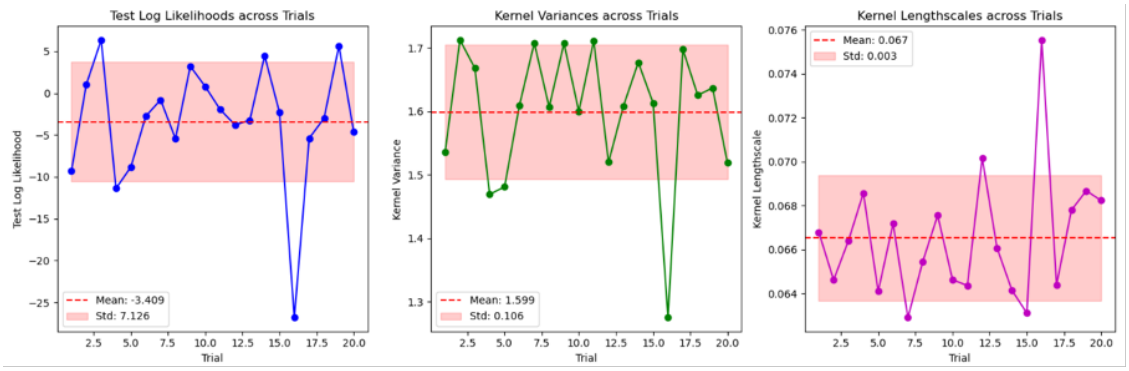
Appendix B: Function fitting with Gaussian Processes



(a) GP Regression Results with MAP after 1 Run



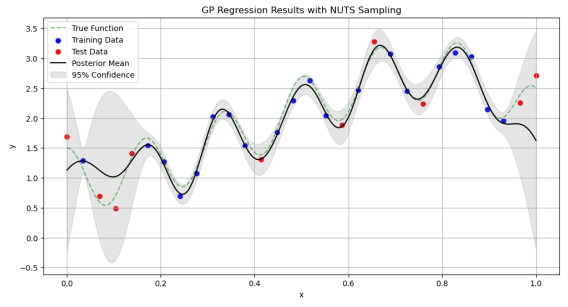
(b) GP Regression Results with the Mean of the MAP Hyperparameters after 20 Runs



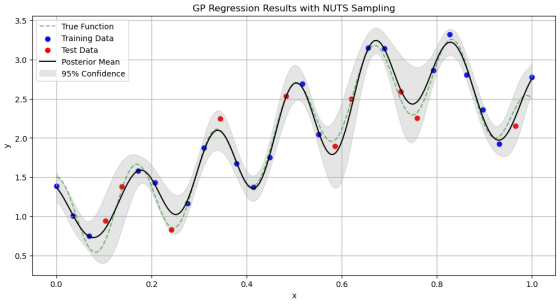
(c) Test Log Likelihoods and Kernel Hyperparameters Across 20 Trials

Figure 8: Overview of GP regression results and performance metrics. Top row: Comparison of single-run MAP results and averaged MAP hyperparameters. Bottom row: Test log likelihoods and kernel hyperparameter statistics across multiple trials.

B.11 Fitting a standard GP using NUTS



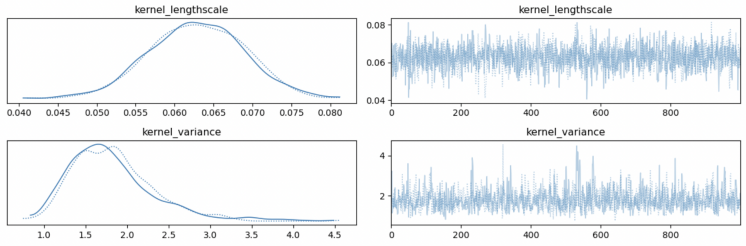
(a) GP Regression Results with NUTS after 1 run



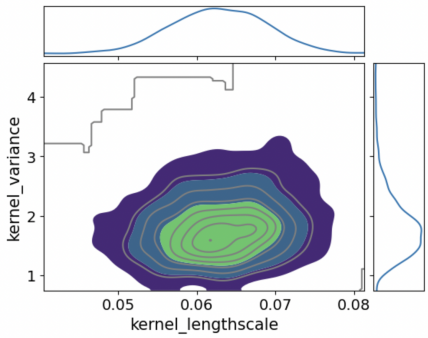
(b) GP Regression Results with Mean NUTS Hyperparameters after 20 runs

Figure 9: Comparison of single-run NUTS results and averaged NUTS hyperparameters

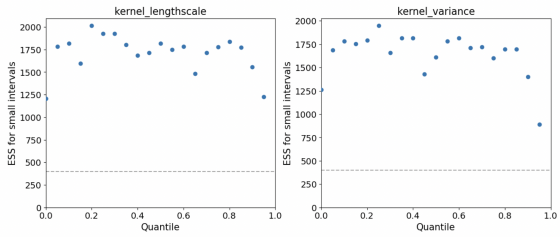
B.12 ArviZ Diagnostics for NUTS



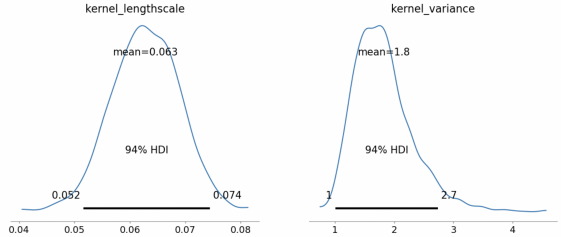
(a) ArviZ Trace and Posterior Plots for Kernel Hyperparameters



(b) ArviZ Joint Posterior Distribution of Kernel Hyperparameters



(c) ArviZ Effective Sample Size (ESS)



(d) ArviZ Posterior Distributions with High-Density Intervals (HDI)

Figure 10: ArviZ Diagnostics for fitting a standard GP using NUTS. Top Row: (a) Visualizes parameter convergence, uncertainty, and variability across iterations. (b) Highlights relationships between parameters and areas of high probability density. Bottom Row: (c) Assesses sampling efficiency and (d) provides credible intervals for robust parameter estimates