Foundations of Diffusion Models

Piotr Grzybowski

January 10, 2026

Introduction

This work is written for those who want to understand the fundamentals of diffusion models from first principles. Rather than presenting final formulas as facts, we uncover the reasoning that leads to them, walking through the intuition, assumptions, and algebraic steps that often remain hidden in academic papers. If you have wondered and tried to understand *"where did this equation come from?"* while reading diffusion models literature, this work may provide the answers.

In our experiments, we refer to the metrics available in the original papers, but we do not attempt to reproduce or surpass their results. Instead, we take a deliberate and exploratory approach, imagining how the authors of advanced techniques might have ideated them. You won't need a data center to follow along, every experiment presented here can be replicated on a single GPU. The goal is to observe empirically and visually, how the theory transforms and evolves into practical challenges.

It is important to note that this work is not a programming guide. Although it is accompanied by a code repository that mirrors the described mathematical framework, this exists to support an understanding of the implementation details and to enable the reproduction of the presented experiments. While we will occasionally reference code snippets to ground the equations in practice, the primary focus remains on conceptual clarity and mathematical insight.

To fully benefit from this journey, you should be comfortable with basic algebra, probability, and statistics. In essence, think of this work as a guided reconstruction of diffusion models to practical experiments designed for the curious mind that values *why* as much as *how*.

Chapter 1

Diffusion Model Framework

This chapter establishes the complete mathematical and conceptual foundation of diffusion models. We begin by introducing the broader context of generative modeling and explain why diffusion models represent a significant advancement in the field. We then systematically develop the mathematical framework through four key components:

- **Generative AI and Diffusion Models**: Historical context comparing VAEs and GANs, motivation for iterative generation, and core concepts behind diffusion-based approaches
- **Forward Process**: How clean data is systematically corrupted with noise over T timesteps, including the joint distribution $q(x_{1:T} \mid x_0)$ and the closed-form solution that enables efficient training
- **Noise Scheduling**: Controlling the noise addition to prevent variance explosion and ensure convergence to standard Gaussian noise
- **Reverse Process**: The generative model $p_\theta(x_{0:T})$ that learns to invert the forward process, along with the intractability challenges that motivate variational inference

By the end of this chapter, you will have a complete understanding of how diffusion models work conceptually and mathematically, providing the foundation for the variational inference techniques and training objectives developed in subsequent chapters.

1.1 Generative AI and Diffusion Models

1.1.1 Generative AI

Generative AI, often abbreviated as GenAI, refers to a class of machine learning models that learn to generate new data that resembles a given dataset. If trained on images of human faces, a generative model should be able to produce entirely new, synthetic faces that appear realistic to humans yet do not belong to any real individual. At its core, a generative model aims to learn the underlying data distribution, typically denoted as $p(x)$, where x represents a data sample. Once this distribution is learned, we can then draw new samples $\tilde{x} \sim p(x)$ from it which share the same characteristics and patterns as the training data. This idea is central to a wide range of applications: synthetic data generation, text-to-image systems, inpainting, super-resolution, music composition, and beyond.

In the case of images, the goal is to generate new images that are similar in terms of structure, texture, semantics and context. Over the years, several families of algorithms have been proposed to tackle this challenge, each differing in how they represent the data distribution and how they are trained. Some models generate images in a single pass transforming random noise directly into a full image, while others rely on a more gradual, multi-step refinement process. The earliest deep generative models approached image generation as a single-step procedure. These models learn to map a source of randomness or compressed information often referred to as a latent variable into a complete, high dimensional image in just one forward pass. This approach is both efficient and conceptually straightforward: sample an input from a known distribution, feed it through a neural network and collect a high quality image. Two foundational model families exemplify this idea: Variational Autoencoders (VAEs) [**kingma2013auto**] and Generative Adversarial Networks (GANs) [**goodfellow2014generative**], which we highly recommend to get familiar with.

While they marked significant milestones in deep generative modeling, both exhibit notable limitations. VAEs, grounded in variational inference, offer a probabilistic framework with structured latent spaces but often produce oversmoothed, low fidelity outputs. This is largely due to their reliance on pixel-wise reconstruction losses, which tend to average over plausible outcomes in multimodal distributions. GANs, in contrast, excel at generating sharp and realistic samples but suffer from unstable training dynamics and mode collapse, owing to the adversarial min-max optimization between the generator and discriminator. Moreover, GANs lack an explicit likelihood, precluding uncertainty quantification and limiting their interpretability.

These complementary weaknesses motivated the search for alternative generative paradigms. *Diffusion models* emerged as a promising solution, introducing a fundamentally different generative approach. Instead of mapping latent variables directly to images in a single step, diffusion models solve this task through iterative denoising, transforming random noise into coherent images through a series of gradual refinements. This multi-step approach offers several key advantages: stable training dynamics, high-quality sample generation, and explicit likelihood estimation.

1.1.2 Diffusion Models: Core Concept

The original idea behind diffusion was deceptively simple: create a model that could gradually generate images by refining them step by step from random noise, rather than in a single pass. This iterative generation approach allows us to concentrate on smaller, manageable tasks that slowly improve the image quality. The model is designed to obtain a slightly better version of the image after each diffusion step. See the visualization in Figure ??.

1.1.3 Trade-offs and Advantages

A key trade-off in diffusion models is their inference pace. Since image generation may unfold over hundreds or even thousands of denoising steps, it can be orders of magnitude slower than one-shot generation approaches. However, this slower generation is often acceptable, as in practice image synthesis is rarely a real-time constraint. In many applications, generating

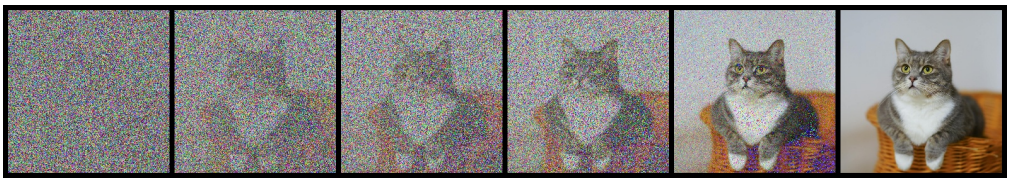


Figure 1.1: Denoising process: iterative transformation from Gaussian noise to a clean image through gradual refinement steps.

a high-quality image is more valuable than generating it instantly. Moreover, modern generative systems typically run asynchronously, where latency is a secondary concern compared to fidelity and control.

1.1.4 Generation Process

During generation, a sequence of progressively refined images $(x_T, x_{T-1}, \ldots, x_0)$ is observed over a predefined number of steps T. After each step, a fraction of noise is removed from x_t, resulting in x_{t-1} and enhancing the image quality. The final element in the sequence, x_0, should be a high-quality image that ideally should be indistinguishable from samples drawn from the true data distribution $p_{\text{data}}(x)$.

1.1.5 Training Data Requirements

To train a diffusion model, we need a collection of high-quality images as a starting point. However, clean images alone are insufficient for the diffusion approach. Since the generation process has been divided into smaller subtasks, we must prepare a dataset that reflects the denoising steps. If the objective of the neural network is to predict x_{t-1} given x_t as input, the training dataset should contain such pairs (x_t, x_{t-1}) for various timesteps t. See Figure **??** for an illustration of this image preparetion procedure.

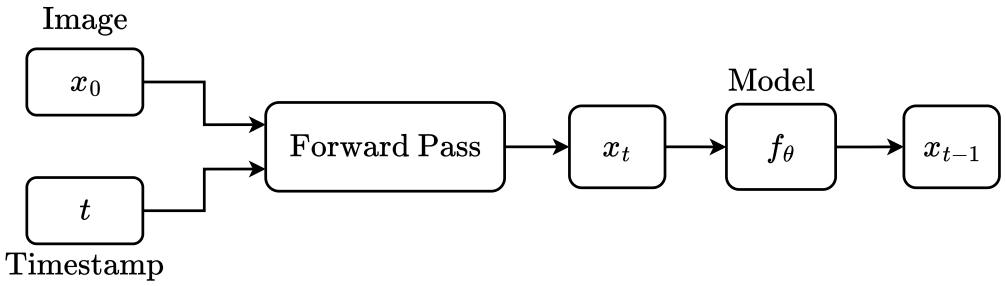


Figure 1.2: Training data generation: forward process from clean image x_0, through noisy image x_t, to create training pairs for predicting less noisy x_{t-1}.

1.1.6 Roadmap

In the remainder of this chapter, we will systematically explore the theoretical foundations of:

- **Forward Process**: How clean images are gradually corrupted with noise to create training data
- **Noise Scheduling**: Techniques for controlling the variance of noise added at each timestep to prevent variance explosion
- **Reverse Process**: The learned denoising procedure that generates images from noise

1.2 Forward Process

1.2.1 Naive Forward Process

The forward diffusion process enables adding noise to an original image x_0 over a specified number of steps t. The parametrization of the timestep is crucial because the neural network needs to understand what type of transformation is required at each denoising stage. Early steps may focus more on shaping the general structure, while final steps involve only minor polishing and sharpening. To obtain x_1, we sample and add noise ϵ to the clean image x_0. More generally, for a given timestep t, we obtain x_t by adding noise $\epsilon_t \sim \mathcal{N}(0, \mathrm{I})$ to its predecessor as shown in the Equation **??**.

$$x_t = x_{t-1} + \epsilon_t \tag{1.1}$$

It is important to notice the probabilistic nature of this process. x_t is not a deterministic value but a random variable, since noise is always sampled. As a result, the forward step defines a distribution over x_t conditioned on x_{t-1}. We use q to denote this distribution, from which we sample x_t as shown in Equation **??**.

$$x_t \sim q(x_t \mid x_{t-1}) \tag{1.2}$$

1.2.2 Joint Forward Distribution

While a single transition $q(x_t \mid x_{t-1})$ describes one diffusion step, the complete forward process over T steps defines a joint distribution over all intermediate noisy images. Since each step depends only on its immediate predecessor (the Markov property), the joint distribution factorizes as a product of individual transitions:

$$q(x_{1:T} \mid x_0) = \prod_{t=1}^{T} q(x_t \mid x_{t-1}) \tag{1.3}$$

This factorized form has important implications. First, it means we can efficiently sample a complete forward trajectory by sequentially applying single-step transitions. Second, and crucially for later chapters, this distribution $q(x_{1:T} \mid x_0)$ will serve as the proposal distribution in importance sampling when deriving the training objective. By conditioning on the clean image x_0, it provides a tractable way to generate noisy samples at all timesteps, which becomes essential for constructing the Evidence Lower Bound (ELBO) in Chapter 2.

1.2.3 Convergence Analysis

While a single step $x_t \sim q(x_t \mid x_{t-1})$ is easy to interpret, we need to analyze its behavior over T steps to fully characterize how an image is gradually transformed into noise. A key question is how the distribution of x_T will look like after applying T steps. The importance of this analysis stems from our need to establish a common, well-defined starting point for the reverse process. Standard Gaussian noise provides an ideal reference distribution that is both theoretically convenient and practically easy to sample from, ensuring our diffusion model has a reliable foundation for generation.

Hypothesis 1 *The forward diffusion process as defined in Equation* **??** *produces a structured degradation of the original image x_0 such that after T timesteps, the resulting distribution converges to a standard Gaussian distribution $\mathcal{N}(0, \mathrm{I})$.*

To verify Hypothesis **??** we can express the first few steps explicitly

$$\begin{aligned} x_1 &= x_0 + \epsilon_1 \\ x_2 &= x_1 + \epsilon_2 = x_0 + \epsilon_1 + \epsilon_2 \\ x_3 &= x_2 + \epsilon_3 = x_0 + \epsilon_1 + \epsilon_2 + \epsilon_3 \end{aligned}$$

By continuing this expansion, a clear pattern emerges. Each intermediate image x_t is formed by starting with the original image x_0 and adding all noise terms introduced at each previous step. Therefore, we can express this pattern generally as shown in Equation **??**.

$$x_t = x_0 + \sum_{i=1}^{t} \epsilon_i \tag{1.4}$$

In order to understand the noise characteristics introduced after t iterations, we can utilize the additive properties of Gaussian distributions.

Property 1 *Let X and Y be independent Gaussian random variables. Then their sum $Z = X + Y$ follows a Gaussian distribution:*

$$Z \sim \mathcal{N}(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2) \tag{1.5}$$

Equivalently, the mean and variance satisfy:

$$\begin{aligned} \mathbb{E}[X + Y] &= \mathbb{E}[X] + \mathbb{E}[Y] \\ \mathrm{Var}(X + Y) &= \mathrm{Var}(X) + \mathrm{Var}(Y) \end{aligned} \tag{1.6}$$

Since each $\epsilon_t \sim \mathcal{N}(0, \mathrm{I})$ is independently sampled Gaussian noise with unit variance, the total noise variance becomes $t\mathrm{I}$ as a sum of t independent steps. This property allows us to replace the cumulative sum in Equation **??** with a single Gaussian noise addition having variance $t\mathrm{I}$, as shown in Equation **??**:

$$x_t = x_0 + \mathcal{N}(0, t\mathrm{I}) \tag{1.7}$$

1.2.4 Variance Explosion Problem

While the naive forward process seems reasonable at first glance, it suffers from a fundamental flaw that makes it unsuitable for diffusion models. To understand this issue, let's analyze the statistical properties of x_t from Equation **??**. The mean remains stable across all timesteps:

$$\mathbb{E}[x_t] = \mathbb{E}[x_0 + \mathcal{N}(0, t\mathrm{I})] = \mathbb{E}[x_0] + \mathbb{E}[\mathcal{N}(0, t\mathrm{I})] = x_0 \tag{1.8}$$

However, the variance grows linearly with time:

$$\mathrm{Var}[x_t] = \mathrm{Var}[x_0] + \mathrm{Var}[\mathcal{N}(0, t\mathrm{I})] = \mathrm{Var}[x_0] + t\mathrm{I} \tag{1.9}$$

This leads to **variance explosion** as $t \to \infty$, we have $\mathrm{Var}[x_t] \to \infty$. Instead of converging to a well-behaved standard Gaussian distribution, the process becomes increasingly unstable.

$$\lim_{t\to\infty} x_t = x_0 + \mathcal{N}(0, \infty) \quad \text{(undefined)} \tag{1.10}$$

This variance explosion fundamentally contradicts our goal of reaching a tractable reference distribution. Rather than producing structured degradation toward standard Gaussian noise, the naive process yields an uncontrolled explosion that makes denoising impossible. The solution requires a more sophisticated forward process that balances signal decay with noise addition, ensuring convergence to a well-defined target distribution.

To address the challenge of uncontrolled noise accumulation and facilitate an effective starting point for the reverse process, a mechanism known as *Noise Scheduling* was introduced. This method carefully regulates the variance of the noise added at each step, ensuring a consistent and manageable noise level throughout the process. The next chapter will explore this noise scheduling mechanism in detail, showing how it resolves the variance explosion problem while maintaining the desired properties of the forward diffusion process.

1.3 Noise Scheduling

1.3.1 Motivation and Problem Statement

As demonstrated in the previous section, the naive forward process suffers from uncontrolled variance explosion, making it unsuitable for practical diffusion models. Noise scheduling differs from the naive approach as it controls the noise-to-signal ratio at each step. Instead of adding noise directly to the previous image, it generates x_t as a weighted sum of x_{t-1} and ϵ. See the formulation of the noise scheduling in Equation **??**.

$$x_t = \sqrt{\alpha_t} x_{t-1} + \sqrt{\beta_t}\epsilon_t \tag{1.11}$$

We observe two hyperparameters α_t and β_t. Both of them are defined over $(0, 1)$ domain, and are strictly connected to each other as shown in Equation **??**.

$$\alpha_t = 1 - \beta_t \tag{1.12}$$

They control the retention of information from the previous image x_{t-1} and the noise ϵ_t at step t. This methodology enforces a cap on noise addition, ensures convergence to pure

Gaussian noise $\mathcal{N}(0, \mathrm{I})$ as $t \to \infty$, and crucially mitigates the variance explosion problem identified in the previous section.

1.3.2 Mathematical Formulation

Hypothesis 2 *Forward process with noise scheduling defined in* **??** *ensures that* x_t *converges to a standard Gaussian distribution* $\mathcal{N}(0, \mathrm{I})$ *as* $t \to \infty$ *while maintaining bounded variance at each timestep.*

To verify Hypothesis **??** we can express the first few steps explicitly

$$\begin{aligned} x_1 &= \sqrt{\alpha_1} x_0 + \sqrt{1-\alpha_1}\epsilon_1 \\ x_2 &= \sqrt{\alpha_2} x_1 + \sqrt{1-\alpha_2}\epsilon_2 \\ x_3 &= \sqrt{\alpha_3} x_2 + \sqrt{1-\alpha_3}\epsilon_3 \end{aligned} \tag{1.13}$$

We start by substituting x_1 into the x_2 formula and using the properties of multiplication and square roots.

$$\begin{aligned} x_2 &= \sqrt{\alpha_2}(\sqrt{\alpha_1} x_0 + \sqrt{1-\alpha_1}\epsilon_1) + \sqrt{1-\alpha_2}\epsilon_2 \\ x_2 &= \sqrt{\alpha_1\alpha_2} x_0 + \sqrt{\alpha_2 - \alpha_1\alpha_2}\epsilon_1 + \sqrt{1-\alpha_2}\epsilon_2 \end{aligned} \tag{1.14}$$

Following the same procedure by substituting x_2 into the x_3 formula

$$\begin{aligned} x_3 &= \sqrt{\alpha_3}(\sqrt{\alpha_1\alpha_2} x_0 + \sqrt{\alpha_2 - \alpha_1\alpha_2}\epsilon_1 + \sqrt{1-\alpha_2}\epsilon_2) + \sqrt{1-\alpha_3}\epsilon_3 \\ x_3 &= \sqrt{\alpha_1\alpha_2\alpha_3} x_0 + \sqrt{\alpha_2\alpha_3 - \alpha_1\alpha_2\alpha_3}\epsilon_1 + \sqrt{\alpha_3 - \alpha_2\alpha_3}\epsilon_2 + \sqrt{1-\alpha_3}\epsilon_3 \end{aligned} \tag{1.15}$$

Additional steps reveal a clear pattern. With each iterative expansion, another α_t term multiplies the coefficient of x_0, while scaled noise terms ϵ_t accumulate with appropriate weights.

$$\begin{aligned} x_1 &= \sqrt{\alpha_1} x_0 + \sqrt{1-\alpha_1}\epsilon_1 \\ x_2 &= \sqrt{\alpha_1\alpha_2} x_0 + \sqrt{\alpha_2 - \alpha_1\alpha_2}\epsilon_1 + \sqrt{1-\alpha_2}\epsilon_2 \\ x_3 &= \sqrt{\alpha_1\alpha_2\alpha_3} x_0 + \sqrt{\alpha_2\alpha_3 - \alpha_1\alpha_2\alpha_3}\epsilon_1 + \sqrt{\alpha_3 - \alpha_2\alpha_3}\epsilon_2 + \sqrt{1-\alpha_3}\epsilon_3 \end{aligned} \tag{1.16}$$

The first term of each x_t indicates how much of the original x_0 is preserved at a given timestep. With each consecutive step, an additional α_t appears under the square root, gradually reducing the weight of x_0. This scaling effect ensures that as t increases, the image x_t retains progressively less of the original structure. For any arbitrary step x_t, this coefficient becomes $\sqrt{\alpha_1 \cdot \alpha_2 \cdots \alpha_t}$, representing the cumulative product of all α_t factors up to step t. To simplify notation, we introduce new variable, γ_t, defined as the cumulative product of α_t as shown in Equation **??**.

$$\gamma_t = \prod_{i=1}^{t} \alpha_i \tag{1.17}$$

The general expression for x_t can be described as a weighted combination of x_0 and t distinct noise components, each with its own weight, collectively forming the overall noise term as outlined in Equation **??**.

$$x_t = \sqrt{\gamma_t} x_0 + \text{Noise}_t \tag{1.18}$$

Each term of Noise_t is a Gaussian noise, which ensures that the cumulative noise is also Gaussian. To express the cumulative noise in closed form, we need to determine its total variance. We represent Noise_t in its generalized form by introducing products with appropriate indexing to account for the contributions of each noise component as shown in Equation **??**.

$$\text{Noise}_t = \sqrt{\prod_{i=2}^{t} \alpha_i - \prod_{i=1}^{t} \alpha_i}\epsilon_1 + \sqrt{\prod_{i=3}^{t} \alpha_i - \prod_{i=2}^{t} \alpha_i}\epsilon_2 + \sqrt{\prod_{i=4}^{t} \alpha_i - \prod_{i=3}^{t} \alpha_i}\epsilon_3 + \ldots + \sqrt{1-\alpha_t}\epsilon_t \tag{1.19}$$

We utilize Property **??** to evaluate the total Noise_t variance.

Property 2 (Variance Scaling) *For any random variable X and constant c, the variance scales quadratically:*

$$\text{Var}[cX] = c^2 \text{Var}[X] \tag{1.20}$$

We square each coefficient to compute the variance, noting how terms cancel:

$$\text{Var}[\text{N}_t] = I\left[\left(\prod_{i=2}^{t}\alpha_i - \prod_{i=1}^{t}\alpha_i\right) + \left(\prod_{i=3}^{t}\alpha_i - \prod_{i=2}^{t}\alpha_i\right) + \left(\prod_{i=4}^{t}\alpha_i - \prod_{i=3}^{t}\alpha_i\right) + \ldots + (1-\alpha_t)\right]$$

The highlighted coefficients progressively cancel out when summed, greatly simplifying the overall expression. Additionally, we use the definition of γ_t to arrive at the final form of Noise_t variance as shown in Equation **??**.

$$\text{Var}[\text{Noise}_t] = I\left(1 - \prod_{i=1}^{t} \alpha_i\right) = I(1-\gamma_t) \tag{1.21}$$

1.3.3 Closed-form Solution

It allows us to express x_t in closed form as we know that $\text{Noise}_t \sim \mathcal{N}(0, (1-\gamma_t)\text{I})$. We can use a single, cumulative noise addition instead of an iterative approach over t timesteps, as shown in Equation **??**.

$$x_t = \sqrt{\gamma_t} x_0 + \sqrt{1-\gamma_t}\epsilon_t \tag{1.22}$$

Note that as t increases, γ_t gradually shrinks, causing the influence of x_0 to fade, while the effect of the noise $\sqrt{1-\gamma_t}$ starts dominating. By retaining most of the information early,

the model can understand and make precise adjustments that have a significant impact on the final quality of the image. At higher noise levels, the model prioritizes reconstructing the image's overall structure, capturing coarse semantic patterns that serve as a scaffold for later refinement of finer details.

To verify Hypothesis **??**, we examine the limiting behavior of our forward process as $t \to \infty$:

$$\begin{aligned} \lim_{t\to\infty} \sqrt{\gamma_t} &= \lim_{t\to\infty} \sqrt{\prod_{i=1}^{t} \alpha_i} = 0 \\ \lim_{t\to\infty} \sqrt{1-\gamma_t} &= \lim_{t\to\infty} \sqrt{1 - \prod_{i=1}^{t} \alpha_i} = 1 \\ \lim_{t\to\infty} x_t &= 0 \cdot x_0 + 1 \cdot \epsilon = \epsilon_t \sim \mathcal{N}(0, \mathrm{I}) \end{aligned} \tag{1.23}$$

Hypothesis **??** holds and noise scheduling guarantees that, regardless of the starting data it converges to a standard Gaussian distribution at the limit.

1.3.4 Practical Implementation

Having established the theoretical foundations of noise scheduling, it is important to address its practical realization. While the idealized formulation of the diffusion process considers an infinite number of steps to asymptotically transform data into pure noise, such an approach is computationally infeasible. In practice, most implementations settle on a finite number of steps T, commonly in the range of 1000 to several thousand. This provides a sufficient temporal resolution to gradually destroy and reconstruct the data, while maintaining tractable training and sampling runtimes.

A natural question then arises: how should one choose the values of α_t and β_t? Since β_t governs the noise added at each step and directly influences $\alpha_t = 1 - \beta_t$, the most straightforward approach is to define β_t as a linear interpolation between two values a and b, spread evenly across the T steps as in Equation **??**.

$$\beta_t = \text{linspace}(a, b, T) \tag{1.24}$$

It defines a linear scheduler, which is not only simple to implement but also exhibits the desired behavior when paired with the computation of γ_t. The next step is to choose suitable values for a and b. These should ensure that $\gamma_0 \approx 1$ (preserving the original data at $t = 0$) and that $\gamma_T \approx 0$ (guaranteeing that the signal is almost entirely replaced by noise at the final step). In [**ho2020denoising**], the authors use $T = 1000$ steps and define $\beta_t \in [10^{-4}, 0.02]$. These values were chosen empirically to provide smooth noise injection that balances preservation of structure in early steps with complete destruction in later steps. This results in a smooth and progressive noise injection where γ_t decays steadily toward zero. This configuration has become a standard reference point for many subsequent works in diffusion-based generative modeling. See the visualization of the considered noise schedule in Figure **??**.

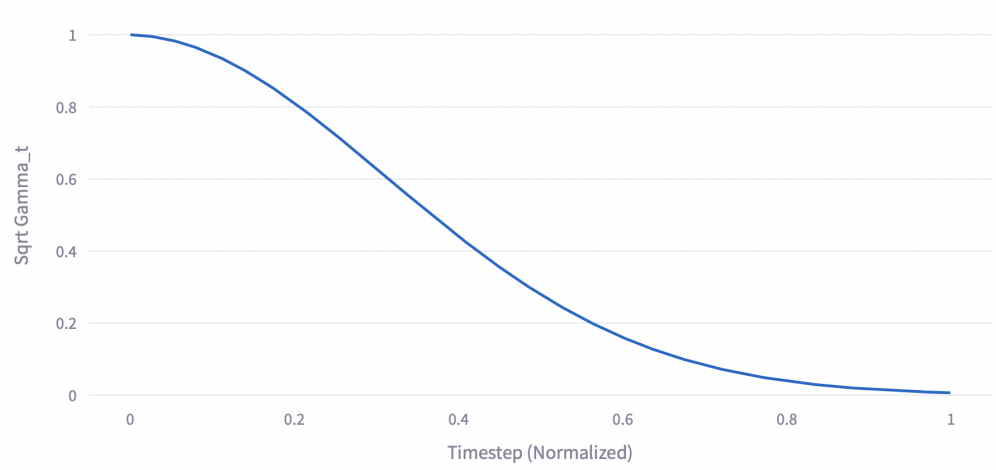


Figure 1.3: $\sqrt{\gamma_t}$ for a linear scheduler with $\beta_t \in [10^{-4}, 0.02]$ and $T = 1000$

Importantly, once a set of parameters (a, b, T) has been selected and validated for the desired noise profile, the same characteristics can be preserved when increasing the number of steps to a new value $T' > T$. This is achieved by scaling the original range proportionally as shown in Equation **??**.

$$\beta_t^{(T')} = \text{linspace}(a', b', T'), \text{ where } a' = a \cdot \frac{T}{T'} \text{ and } b' = b \cdot \frac{T}{T'} \tag{1.25}$$

This trick ensures that the cumulative noise distribution γ_t follows a similar decay profile, just spread across more steps. The intuition is that each step now adds a smaller amount of noise, but in aggregate, the diffusion process converges toward the same limit as before.

We now have a principled and configurable mechanism for progressively degrading clean data into noise through a finite, well-defined forward process. It enables us not only to visualize and understand how noise accumulates over time but also to generate training pairs (x_0, x_t) where we can compute the added noise directly. These pairs form the foundation for training the reverse denoising process that we explore in the next section.

1.4 Reverse Process

1.4.1 Motivation and Overview

Having established both the forward process and noise scheduling in previous sections we now turn to the core generative challenge: learning to reverse the diffusion process. Recall that our forward process systematically transforms clean data x_0 into pure Gaussian noise x_T through a sequence of controlled noise additions. The reverse process must learn to invert this transformation, starting from random noise $x_T \sim \mathcal{N}(0, \mathrm{I})$ and progressively denoising to generate realistic samples x_0. The ultimate aim of the model is to be able to sample new data points x_0 that are indistinguishable from those drawn from the true data distribution $p_{\text{data}}(x_0)$. In probabilistic terms, this means defining a parameterized distribution $p_\theta(x_0)$ such that $p_\theta(x_0) \approx p_{\text{data}}(x_0)$ in terms of distributional distance. If we succeed, then sampling from $p_\theta(x_0)$ produces realistic images or other data modalities.

During the denoising, the variables $\{x_T, x_{T-1}, \ldots, x_0\}$ are treated as random variables linked through a sequence of transformations. Each reverse step depends only on the immediate state from the next timestep, which makes the individual transitions conditionally independent given their predecessor. This conditional independence ensures that knowing x_t renders all earlier variables irrelevant for predicting x_{t-1} - a property captured formally by a Markov assumption. Under this assumption, the reverse process is described by a factorized joint distribution as shown in Equation **??**.

$$p_\theta(x_{0:T}) = p_\theta(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \tag{1.26}$$

Since the observable data x_0 is generated alongside intermediate latent variables $x_{1:T}$, its marginal distribution is obtained by integrating over all possible latent trajectories: In other words, we marginalize out the latent variables $x_{1:T}$.

$$p_\theta(x_0) = \int p_\theta(x_0, x_1, \ldots, x_T) dx_1 dx_2 \ldots dx_T \tag{1.27}$$

Conceptually, this aggregates contributions from every possible sequence of noisy intermediate images that could lead to x_0 through the reverse process.

1.4.2 Likelihood Objective

The goal of the training is to have an ability to sample high quality images from pure noise, which are similar to the true samples. Formally, model training translates into likelihood maximization as shown in Equation **??**.

$$\theta^* = \underset{\theta}{\operatorname{argmax}}\ p_\theta(x_0) \tag{1.28}$$

However, directly evaluating $p_\theta(x_0)$ is computationally intractable. The marginalization requires integrating over a $T \cdot D$ dimensional space (where D is data dimensionality), with exponentially many possible trajectories and neural network nonlinearity preventing closed-form solutions. This necessitates alternative, tractable training objectives.

1.4.3 Importance Sampling Transformation

To make progress, we introduce an auxiliary distribution $q(x_{1:T} \mid x_0)$, which we choose to be the joint forward process distribution defined in Equation **??**. Recall that this distribution factorizes as $q(x_{1:T} \mid x_0) = \prod_{t=1}^{T} q(x_t \mid x_{t-1})$ due to the Markov property of the forward diffusion. This distribution has two critical advantages: it is tractable to evaluate for any $(x_0, x_{1:T})$ pair, and it is straightforward to sample from. The key observation is that we can multiply and divide the marginal likelihood by this known distribution without changing its value as shown in Equation **??**.

$$p_\theta(x_0) = \int q(x_{1:T}|x_0) \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)} dx_{1:T} \tag{1.29}$$

This algebraic manipulation transforms the problem fundamentally. The original integral requires summing over all possible latent trajectories under the intractable reverse distribution p_θ. By introducing q, we shift the integration to be performed under the known forward distribution instead. The expectation is now taken with respect to $q(x_{1:T} \mid x_0)$, which we can easily sample from by running the forward diffusion process. This technique is known as importance sampling, where q serves as the proposal distribution, as formalized in Equation **??**.

$$p_\theta(x_0) = E_{q(x_{1:T}|x_0)}\left[\frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}\right] \tag{1.30}$$

While this importance sampling formulation expresses the marginal likelihood as an expectation over a tractable distribution, it introduces a new challenge: variance explosion in Monte Carlo estimation. When p_θ and q are poorly aligned, which is typical during early training, the importance weights can vary dramatically across samples. Some trajectories may have tiny q probabilities but relatively large p_θ, leading to enormous importance weights that dominate the gradient estimates and cause training instability.

This variance problem makes direct Monte Carlo estimation of the importance sampling objective impractical. Fortunately, the Evidence Lower Bound (ELBO) provides a solution. By working with the logarithm of the likelihood and applying Jensen's inequality, the ELBO transforms the intractable expectation into a tractable optimization objective. This yields both computational stability through reduced variance and a principled variational inference framework that will be fully developed in Chapter 2.

1.5 Summary

We have now established the theoretical framework from the forward process through noise scheduling to the reverse process and its challenges. The key insights from this chapter are:

- **Forward process with noise scheduling**: The formulation $x_t = \sqrt{\alpha_t}x_{t-1} + \sqrt{1-\alpha_t}\epsilon_t$ prevents variance explosion and ensures convergence to standard Gaussian noise. The joint distribution $q(x_{1:T} \mid x_0) = \prod_{t=1}^{T} q(x_t \mid x_{t-1})$ factorizes due to the Markov property, and the closed-form solution $x_t = \sqrt{\gamma_t}x_0 + \sqrt{1-\gamma_t}\epsilon$ enables efficient training
- **Reverse process formulation**: The generative model follows a Markov chain structure $p_\theta(x_{0:T}) = p_\theta(x_T)\prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$, where each denoising step depends only on the current state
- **Intractable likelihood**: Direct maximum likelihood estimation $\max_\theta p_\theta(x_0)$ requires intractable high-dimensional marginalization over all possible latent trajectories
- **Importance sampling foundation**: By introducing the forward process $q(x_{1:T}|x_0)$ as a proposal distribution, we transform the problem into a tractable expectation, though variance explosion challenges motivate the need for the ELBO framework

Chapter 2 will develop the mathematical machinery needed to train these models in practice, focusing on the ELBO derivation and variational inference techniques that make optimization both tractable and stable.

Chapter 2

Variational Inference

We concluded Chapter 1 with a critical insight: while importance sampling provides a mathematical pathway to tractable likelihood estimation, it suffers from severe variance problems that make direct optimization unstable. The importance weights can vary dramatically, leading to unreliable gradient estimates and poor training dynamics.

This chapter addresses these challenges through variational inference, a powerful framework from probabilistic machine learning that transforms intractable optimization problems into manageable approximations. The key insight is to work with the logarithm of the likelihood rather than the raw likelihood itself, which dramatically improves numerical stability. Our journey through this chapter follows:

1. **Evidence Lower Bound (ELBO) Derivation**: We apply Jensen's inequality to derive a tractable bound on the negative log-likelihood, converting the unstable importance sampling objective into a stable variational inference problem
2. **ELBO Expansion**: We manipulate the ELBO algebraically to introduce the posterior distribution $q(x_{t-1} \mid x_t, x_0)$, which conditions on clean data and enables tractable training
3. **Explicit Posterior Formulation**: We derive the closed-form Gaussian parameterization of $q(x_{t-1} \mid x_t, x_0)$, providing concrete mean and variance formulas needed for practical implementation

By the end of this chapter, we will have transformed the theoretical framework from Chapter 1 into practical mathematical tools that can be implemented and optimized efficiently.

2.1 Evidence Lower Bound Derivation

We established in Chapter 1 an importance sampling formulation for the intractable likelihood, but this approach suffers from severe variance problems that make direct optimization unstable. The solution lies in variational inference: instead of working with the raw likelihood, we derive a tractable bound on the negative log-likelihood.

2.1.1 Derivation

We begin with transforming the training objective into the negative log-likelihood. In practice, we minimize the negative log-likelihood (cross-entropy loss) rather than maximize the likelihood directly, as shown in Equation **??**.

$$L_{\text{CE}} = -\log p_\theta(x_0) \tag{2.1}$$

Substituting our importance sampling formulation from Chapter 1 we have:

$$L_{\text{CE}} = -\log \mathbb{E}_{q(x_{1:T}|x_0)} \left[\frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)} \right] \tag{2.2}$$

This formulation still suffers from the same variance problems. Simply taking the logarithm alone doesn't help because the expectation remains outside, leaving the high-variance estimator unchanged. The crucial step is to move the logarithm **inside** the expectation using Jensen's inequality. Using the logarithm property $-\log a = \log a^{-1}$, we can rewrite **??** as **??**.

$$L_{\text{CE}} = \log \mathbb{E}_{q(x_{1:T}|x_0)} \left[\frac{q(x_{1:T} \mid x_0)}{p_\theta(x_{0:T})} \right] \tag{2.3}$$

For a concave function like log, Jensen's inequality states:

$$\mathbb{E}[\log f(x)] \leq \log \mathbb{E}[f(x)] \tag{2.4}$$

Rearranging Jensen's inequality for our case and applying it to move the logarithm inside the expectation gives us Equation **??**.

$$L_{\text{CE}} = \log \mathbb{E}_{q(x_{1:T}|x_0)} \left[\frac{q(x_{1:T} \mid x_0)}{p_\theta(x_{0:T})} \right] \geq \mathbb{E}_{q(x_{1:T}|x_0)} \left[\log \frac{q(x_{1:T} \mid x_0)}{p_\theta(x_{0:T})} \right] = L_{\text{VLB}} \tag{2.5}$$

Jensen's inequality provides a bound rather than an equality: while moving the log inside the expectation dramatically reduces variance, we obtain an upper bound rather than the exact value. The resulting objective L_{VLB} is called the variational lower bound (or upper bound on the loss). This bound is beneficial because:

- The variance reduction enables stable gradient computation
- The bias is systematic rather than random, making optimization predictable
- Minimizing an upper bound on the loss still improves the true objective

The quantity L_{VLB} is the *variational lower bound* (VLB) on the negative log-likelihood, or equivalently, an upper bound on the cross-entropy loss. By minimizing L_{VLB}, we minimize an upper bound on L_{CE}, which indirectly maximizes the log-likelihood $\log p_\theta(x_0)$. Since the bound is tight when p_θ matches the true posterior, optimizing the bound drives the model parameters toward the correct distribution.

2.1.2 Initial ELBO Expansion

To proceed further, we need to express the ELBO in a more explicit and usable form. First, we expand the condensed version to include product operators.

$$L_{VLB} = \mathbb{E}_{q(x_{1:T}|x_0)} \left[\log \frac{\prod_{t=1}^{T} q(x_t|x_{t-1})}{p_\theta(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)} \right] \tag{2.6}$$

We leverage the logarithm properties to rewrite the fraction with subtraction, then products with sums.

$$\begin{aligned} L_{VLB} &= \mathbb{E}_{q(x_{1:T}|x_0)} \left[\log \left(\prod_{t=1}^{T} q(x_t|x_{t-1}) \right) - \log \left(p_\theta(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) \right) \right] \\ &= \mathbb{E}_{q(x_{1:T}|x_0)} \left[\sum_{t=1}^{T} \log q(x_t|x_{t-1}) - \log p_\theta(x_T) - \sum_{t=1}^{T} \log p_\theta(x_{t-1}|x_t) \right] \end{aligned} \tag{2.7}$$

Notice that both sums are defined over the same range $t \in (1, T)$, we can combine them into a single sum:

$$L_{VLB} = \mathbb{E}_{q(x_{1:T}|x_0)} \left[\sum_{t=1}^{T} \Big(\log q(x_t|x_{t-1}) - \log p_\theta(x_{t-1}|x_t) \Big) - \log p_\theta(x_T) \right] \tag{2.8}$$

The logarithm property allows us to replace subtraction with division, resulting in a ratio of forward and reverse steps.

$$L_{VLB} = \mathbb{E}_{q(x_{1:T}|x_0)} \left[\sum_{t=1}^{T} \log \frac{q(x_t|x_{t-1})}{p_\theta(x_{t-1}|x_t)} - \log p_\theta(x_T) \right] \tag{2.9}$$

This form reveals a fundamental challenge: the terms $q(x_t \mid x_{t-1})$ and $p_\theta(x_{t-1} \mid x_t)$ describe opposite directions of the diffusion process, making direct optimization difficult. The next section will show how conditioning on the clean data x_0 resolves this challenge and leads to a tractable training objective.

2.2 ELBO Expansion

The current ELBO formulation involves comparing $q(x_t \mid x_{t-1})$ and $p_\theta(x_{t-1} \mid x_t)$ transitions in opposite directions, which creates optimization challenges. We would prefer to work with $q(x_{t-1} \mid x_t)$ to match directions, but this requires unknown marginals via Bayes' theorem:

$$q(x_{t-1} \mid x_t) = \frac{q(x_t \mid x_{t-1}) q(x_{t-1})}{q(x_t)} \tag{2.10}$$

While we know the forward transition $q(x_t \mid x_{t-1})$, the marginal distributions $q(x_{t-1})$ and $q(x_t)$ are unknown as they emerge implicitly from the forward process across all possible x_0 values.

The key insight is that this obstacle becomes our solution: instead of working with unknown marginals that average over all possible x_0, we condition explicitly on the known training data x_0, forming $q(x_{t-1} \mid x_t, x_0)$. This conditioning leverages our access to clean training data and transforms the problem into working with tractable conditional distributions.

Hypothesis 3 *Given the forward process defined from noise scheduling, the conditional distribution $q(x_{t-1} \mid x_t, x_0)$ is tractable and admits a closed-form Gaussian parameterization.*

To verify Hypothesis **??**, we begin by using the chain rule on joint distribution $q(x_0, x_{t-1}, x_t)$ as shown in Equation **??**.

$$\begin{aligned} q(x_0, x_{t-1}, x_t) &= q(x_t|x_{t-1}, x_0) \cdot q(x_{t-1}|x_0) \cdot q(x_0) \\ q(x_0, x_{t-1}, x_t) &= q(x_{t-1}|x_t, x_0) \cdot q(x_t|x_0) \cdot q(x_0) \end{aligned} \tag{2.11}$$

As both right-hand sides are equal to each other, we can compare them and further simplify

$$\begin{aligned} q(x_{t-1}|x_t, x_0) \cdot q(x_t|x_0) \cdot q(x_0) &= q(x_t|x_{t-1}, x_0) \cdot q(x_{t-1}|x_0) \cdot q(x_0) \\ q(x_{t-1}|x_t, x_0) \cdot q(x_t|x_0) &= q(x_t|x_{t-1}, x_0) \cdot q(x_{t-1}|x_0) \end{aligned} \tag{2.12}$$

Resulting in the Equation **??**.

$$q(x_{t-1}|x_t, x_0) = \frac{q(x_t|x_{t-1}, x_0) \cdot q(x_{t-1}|x_0)}{q(x_t|x_0)} \tag{2.13}$$

Consider following insights into the tractability of **??**:

- $q(x_t \mid x_{t-1}, x_0)$: Represents the forward diffusion step with additional conditioning on x_0. Since the forward process is Markovian, x_0 provides no extra information for this intermediate step. Thus, we can simplify it to $q(x_t \mid x_{t-1})$, which is a tractable forward step from the noise scheduling framework.
- $q(x_t \mid x_0)$: This expression has a known and tractable closed form from our noise scheduling analysis.
- $q(x_{t-1} \mid x_0)$: Similarly, this term also has a tractable closed form, as it only differs in timestep.

Based on these observations, we state that $q(x_{t-1}|x_t, x_0)$ is tractable as all its factors are tractable which prove hypothesis **??**. To leverage this in the definition of L_{VLB}, we need to rearrange it to represent $q(x_t \mid x_{t-1})$.

$$q(x_t|x_{t-1}) = \frac{q(x_{t-1}|x_t, x_0)q(x_t|x_0)}{q(x_{t-1}|x_0)} \tag{2.14}$$

Which we now use in the formulation of L_{VLB}.

2.2.1 Final ELBO Form

In order to handle some edge cases let us extract the first term $t = 1$ of the sum in **??** from it as a standalone term as shown in **??**.

$$L_{VLB} = \mathbb{E}_{q(x_{1:T}|x_0)} \left[\sum_{t=2}^{T} \log \frac{q(x_t|x_{t-1})}{p_\theta(x_{t-1}|x_t)} + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)} - \log p_\theta(x_T) \right] \tag{2.15}$$

Now we use previously derived form of $q(x_t \mid x_{t-1})$

$$L_{VLB} = \mathbb{E}_{q(x_{1:T}|x_0)} \left[\sum_{t=2}^{T} \log \frac{\frac{q(x_{t-1}|x_t,x_0)q(x_t|x_0)}{q(x_{t-1}|x_0)}}{p_\theta(x_{t-1}|x_t)} + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)} - \log p_\theta(x_T) \right] \tag{2.16}$$

We simplify the fractions and leverage logarithm property to express the sum of products as two distincts sums:

$$\begin{aligned} L_{VLB} &= \mathbb{E}_{q(x_{1:T}|x_0)} \left[\sum_{t=2}^{T} \log \left(\frac{q(x_{t-1}|x_t, x_0)}{p_\theta(x_{t-1}|x_t)} \frac{q(x_t|x_0)}{q(x_{t-1}|x_0)} \right) + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)} - \log p_\theta(x_T) \right] \\ &= \mathbb{E}_{q(x_{1:T}|x_0)} \left[\sum_{t=2}^{T} \log \frac{q(x_{t-1}|x_t, x_0)}{p_\theta(x_{t-1}|x_t)} + \sum_{t=2}^{T} \log \frac{q(x_t|x_0)}{q(x_{t-1}|x_0)} + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)} - \log p_\theta(x_T) \right] \end{aligned}$$

We can simplify significantly the sum term marked in cyan. First, we transform the fraction into a subtraction as shown in Equation **??**.

$$\sum_{t=2}^{T} \log \frac{q(x_t|x_0)}{q(x_{t-1}|x_0)} = \sum_{t=2}^{T} \big(\log q(x_t|x_0) - \log q(x_{t-1}|x_0) \big) \tag{2.17}$$

We examine a few explicit terms of this summation to identify the telescoping pattern:

$$\begin{aligned} t = 2 &: \log q(x_2|x_0) - \log q(x_1|x_0) \\ t = 3 &: \log q(x_3|x_0) - \log q(x_2|x_0) \\ t = 4 &: \log q(x_4|x_0) - \log q(x_3|x_0) \\ t = T &: \log q(x_T|x_0) - \log q(x_{T-1}|x_0) \end{aligned} \tag{2.18}$$

The consecutive terms cancel each other out, leaving only two remaining terms.

$$\sum_{t=2}^{T} \log \frac{q(x_t|x_0)}{q(x_{t-1}|x_0)} = \log q(x_T|x_0) - \log q(x_1|x_0) \tag{2.19}$$

We substitute them in place of the sum. As a result, we can simplify the highlighted terms as they are equal with opposite signs.

$$\begin{aligned} L_{VLB} &= \mathbb{E}_{q(x_{1:T}|x_0)} \left[\sum_{t=2}^{T} \log \frac{q(x_{t-1}|x_t, x_0)}{p_\theta(x_{t-1}|x_t)} + \log q(x_T|x_0) - \log q(x_1|x_0) + \log \frac{q(x_1|x_0)}{p_\theta(x_0|x_1)} - \log p_\theta(x_T) \right] \\ &= \mathbb{E}_{q(x_{1:T}|x_0)} \left[\sum_{t=2}^{T} \log \frac{q(x_{t-1}|x_t, x_0)}{p_\theta(x_{t-1}|x_t)} + \log q(x_T|x_0) - \log p_\theta(x_0|x_1) - \log p_\theta(x_T) \right] \end{aligned} \tag{2.20}$$

We group terms conditioned on x_T with a single logarithm.

$$L_{VLB} = \mathbb{E}_{q(x_{1:T}|x_0)} \left[\sum_{t=2}^{T} \log \frac{q(x_{t-1}|x_t, x_0)}{p_\theta(x_{t-1}|x_t)} + \log \frac{q(x_T|x_0)}{p_\theta(x_T)} - \log p_\theta(x_0|x_1) \right] \tag{2.21}$$

To conclude, we reformulate the loss as a sum of three distinct expectations. This approach allows us to explicitly represent each term, providing a clearer understanding of their individual contributions.

$$L_{VLB} = \underbrace{\mathbb{E}_{q(x_T|x_0)} \left[\log \frac{q(x_T|x_0)}{p_\theta(x_T)} \right]}_{L_T} + \underbrace{\mathbb{E}_{q(x_{2:T}|x_0)} \left[\sum_{t=2}^{T} \log \frac{q(x_{t-1}|x_t, x_0)}{p_\theta(x_{t-1}|x_t)} \right]}_{L_t} - \underbrace{\log p_\theta(x_0|x_1)}_{L_0} \tag{2.22}$$

The reconstruction term $L_0 = -\log p_\theta(x_0|x_1)$ appears without an explicit expectation operator. Formally, this term should be written as $L_0 = \mathbb{E}_{q(x_1|x_0)}[-\log p_\theta(x_0|x_1)]$, representing an expectation over the forward diffusion distribution. However, since $q(x_1|x_0)$ is a simple Gaussian and L_0 depends only on the single latent variable x_1, this expectation is straightforward to approximate with a single Monte Carlo sample during training. Therefore, following standard practice in the diffusion literature, we omit the expectation symbol with the implicit understanding that x_1 is sampled from $q(x_1|x_0)$. This notational simplification does not change the underlying expectation being estimated.

2.2.2 Kullback-Leibler Interpretation

We can reformulate L_{VLB} to reveal the underlying objective of the derivation process. During training, our goal is to optimize $p_\theta(x_{t-1} \mid x_t)$ to closely approximate $q(x_{t-1} \mid x_t, x_0)$. Notably, the expectation of the logarithm in this context corresponds directly to the definition of the Kullback-Leibler (KL) divergence shown in Equation **??**.

$$D_{KL}(P||Q) = \int P(x) \log \frac{P(x)}{Q(x)} dx \tag{2.23}$$

The KL divergence quantifies the difference between two probability distributions. By applying its definition to the expectations in L_{VLB}, we arrive at the following decomposition:

$$L_{VLB} = \underbrace{D_{KL}\Big(q(x_T|x_0) \;||\; p_\theta(x_T)\Big)}_{L_T} + \underbrace{\sum_{t=2}^{T} D_{KL}\Big(q(x_{t-1}|x_t, x_0) \;||\; p_\theta(x_{t-1}|x_t)\Big)}_{L_t} - \underbrace{\log p_\theta(x_0|x_1)}_{L_0} \tag{2.24}$$

This decomposition reveals that training a diffusion model is fundamentally about matching distributions: making our parameterized reverse process p_θ as close as possible to the true posterior process $q(x_{t-1} \mid x_t, x_0)$.

2.3 Explicit Posterior Formulation

In the previous section, we established that the posterior distribution $q(x_{t-1} \mid x_t, x_0)$ can be expressed using Bayes' theorem, but we have not yet determined its explicit mathematical form. This posterior distribution is crucial for diffusion model training because it provides the ground truth target that our neural network must learn to approximate. While we know the structural relationship between terms, we need the concrete Gaussian parameterization specifically the mean $\hat{\mu}_t$ and variance $\hat{\sigma}_t^2$ to make this distribution practically useful.

The derivation involves three key steps: (1) express each Bayes' theorem component as Gaussian densities from the forward process, (2) simplify by focusing on exponent terms dependent on x_{t-1}, and (3) apply complete square technique to extract mean $\hat{\mu}_t$ and variance $\hat{\sigma}_t^2$. By the end of this derivation, we will have transformed the abstract probabilistic statement of $q(x_{t-1} \mid x_t, x_0)$ into a fully explicit Gaussian parameterization. This result is of central importance, as it provides a tractable, closed-form distribution for the reverse process conditioned on the clean data x_0 a key step towards designing and implementing effective denoising diffusion models.

2.3.1 Proportionality

We recall the probabilistic form of $q(x_{t-1} \mid x_t, x_0)$ from Equation **??**:

$$q(x_{t-1} \mid x_t, x_0) = \frac{q(x_t \mid x_{t-1}) \cdot q(x_{t-1} \mid x_0)}{q(x_t \mid x_0)}.$$

Each term in this Bayes' expression is Gaussian due to our forward process definition. Under the noise scheduling framework, these distributions are:

$$\begin{aligned} q(x_t \mid x_0) &\sim \mathcal{N}(x_t \mid \sqrt{\gamma_t} x_0, (1-\gamma_t)\mathrm{I}) \\ q(x_{t-1} \mid x_0) &\sim \mathcal{N}(x_{t-1} \mid \sqrt{\gamma_{t-1}} x_0, (1-\gamma_{t-1})\mathrm{I}) \\ q(x_t \mid x_{t-1}) &\sim \mathcal{N}(x_t \mid \sqrt{\alpha_t} x_{t-1}, (1-\alpha_t)\mathrm{I}) \end{aligned} \tag{2.25}$$

The denominator in the Bayes expression acts purely as a normalization factor independent of x_{t-1}. This means that $q(x_{t-1} \mid x_t, x_0)$ is proportional to the product of two Gaussian distributions. The product of Gaussians is itself Gaussian, which we assume is parameterized by a mean $\hat{\mu}_t$ and a variance $\hat{\sigma}_t^2$. These insights lead to the formulation shown in Equation **??**. This insight allows us to write:

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}(x_{t-1} \mid \hat{\mu}_t, \hat{\sigma}_t^2), \tag{2.26}$$

In order to identify $\hat{\mu}_t$ and $\hat{\sigma}_t^2$ we begin with explicit form of each term of **??** using definition of Gaussian density function.

$$\begin{aligned}
\mathcal{N}(x \mid \mu, \sigma^2) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\Big) \\
q(x_t \mid x_0) &= \frac{1}{\sqrt{2\pi(1-\gamma_t)}} \exp\Big(-\frac{1}{2}\frac{(x_t-\sqrt{\gamma_t}x_0)^2}{1-\gamma_t}\Big) \\
q(x_{t-1} \mid x_0) &= \frac{1}{\sqrt{2\pi(1-\gamma_{t-1})}} \exp\Big(-\frac{1}{2}\frac{(x_{t-1}-\sqrt{\gamma_{t-1}}x_0)^2}{1-\gamma_{t-1}}\Big) \\
q(x_t \mid x_{t-1}) &= \frac{1}{\sqrt{2\pi(1-\alpha_t)}} \exp\Big(-\frac{1}{2}\frac{(x_t-\sqrt{\alpha_t}x_{t-1})^2}{1-\alpha_t}\Big)
\end{aligned} \tag{2.27}$$

Note that each term preceding the exponent is merely a constant. When expressing values up to proportionality, these constants can be safely omitted. This simplification allows us to concentrate on the critical terms within the exponent without altering the proportional relationship. Using the property of exponents, $\exp(a)\exp(b) = \exp(a+b)$, we can rewrite it as follows:

$$q(x_{t-1}|x_t, x_0) \propto \exp\Big[-\frac{1}{2}\Big(\frac{(x_t-\sqrt{\alpha_t}x_{t-1})^2}{(1-\alpha_t)} + \frac{(x_{t-1}-\sqrt{\gamma_{t-1}}x_0)^2}{(1-\gamma_{t-1})} - \frac{(x_t-\sqrt{\gamma_t}x_0)^2}{(1-\gamma_t)}\Big)\Big] \tag{2.28}$$

Note that the third term carries a negative sign, as it originally appeared in the denominator of $q(x_{t-1}|x_t, x_0)$. Notice that the last term is also independent of variable x_{t-1}. This is because $\frac{(x_t-\sqrt{\gamma_t}x_0)^2}{(1-\gamma_t)}$ involves only x_t and x_0, neither of which depend on x_{t-1}. The independence implies that this term serves as a constant with respect to the variable we are deriving the distribution for. With this in mind, we replace this term with some constant function $C(x_t, x_0)$.

$$q(x_{t-1}|x_t, x_0) \propto \exp\Big[-\frac{1}{2}\Big(\frac{(x_t-\sqrt{\alpha_t}x_{t-1})^2}{(1-\alpha_t)} + \frac{(x_{t-1}-\sqrt{\gamma_{t-1}}x_0)^2}{(1-\gamma_{t-1})} - C(x_t, x_0)\Big)\Big] \tag{2.29}$$

We rewrite the first two fractions in their resolved explicit forms:

$$q \propto \exp\Big[-\frac{1}{2}\Big(\frac{x_t^2 - 2\sqrt{\alpha_t}x_t x_{t-1} + \alpha_t x_{t-1}^2}{1-\alpha_t} + \frac{x_{t-1}^2 - 2\sqrt{\gamma_{t-1}}x_{t-1}x_0 + \gamma_{t-1}x_0^2}{1-\gamma_{t-1}} - C(x_t, x_0)\Big)\Big] \tag{2.30}$$

Expanding the numerators into separate fractions, we highlight the terms that depend on x_{t-1} as shown in **??**.

$$q \propto \exp\Big[-\frac{1}{2}\Big(\frac{x_t^2}{1-\alpha_t} - \frac{2\sqrt{\alpha_t}x_t x_{t-1}}{1-\alpha_t} + \frac{\alpha_t x_{t-1}^2}{1-\alpha_t} + \frac{x_{t-1}^2}{1-\gamma_{t-1}} - \frac{2\sqrt{\gamma_{t-1}}x_{t-1}x_0}{1-\gamma_{t-1}} + \frac{\gamma_{t-1}x_0^2}{1-\gamma_{t-1}} - C(x_t, x_0)\Big)\Big] \tag{2.31}$$

2.3.2 Quadratic Form of $q(x_{t-1}|x_t, x_0)$

We collect all terms involving the variable x_{t-1} to form a standard quadratic expression. Terms independent of x_{t-1} can be absorbed into the constant $C(x_t, x_0)$ as shown in **??**.

$$q \propto \exp\Big[-\frac{1}{2}\Big(\Big(\frac{\alpha_t}{1-\alpha_t}+\frac{1}{1-\gamma_{t-1}}\Big)x_{t-1}^2-\Big(\frac{2\sqrt{\alpha_t}x_t}{1-\alpha_t}+\frac{2\sqrt{\gamma_{t-1}}x_0}{1-\gamma_{t-1}}\Big)x_{t-1}+C(x_t,x_0)\Big)\Big] \tag{2.32}$$

This quadratic form reveals the underlying structure we need. The exponent of a Gaussian distribution takes the form of a scaled quadratic expression $(x-\mu)^2/\sigma^2$, which corresponds to the vertex form of a quadratic equation as shown in Equation **??**.

$$f(x) = g(x-h)^2 + k \tag{2.33}$$

The parameter h corresponds to the mean μ, the coefficient g is the inverse of σ^2, and k is set to zero, as it does not appear in the Gaussian formula. The next step is to rewrite the exponent of **??** into a vertex form. By identifying g and h, we can directly infer the values of $\hat{\mu}$ and $\hat{\sigma}^2$.

2.3.3 Complete Square Technique

One effective method for transforming a quadratic expression into vertex form is the Complete Square Technique shown in **??**.

$$ax^2+bx+c = a(x+\frac{b}{2a})^2+(c-\frac{b}{4a}) \tag{2.34}$$

Applying this technique to our quadratic expression allows us to extract the Gaussian parameters directly:

$$\begin{aligned} q(x_{t-1} \mid x_t, x_0) &\propto \exp\left(-\frac{1}{2}\left(ax_{t-1}^2 - bx_{t-1} + C(x_t,x_0)\right)\right) \\ &\propto \exp\left(-\frac{1}{2}\,a\left(x_{t-1}+\frac{b}{2a}\right)^2 + C(x_t,x_0)\right) \\ &\propto \exp\left(-\frac{1}{2}\frac{(x_{t-1}-\hat{\mu}_t)^2}{\hat{\sigma}_t^2}\right) \end{aligned} \tag{2.35}$$

By comparing this completed square form with the standard Gaussian exponent, we can directly identify the parameters. In the vertex form of a quadratic, the shift term represents the mean, while the coefficient scaling the squared term corresponds to the inverse variance.

From the completed square form, we can directly read off the Gaussian parameters. The mean $\hat{\mu}_t$ is determined by:

$$\hat{\mu}_t = -\frac{b}{2a} \tag{2.36}$$

where b is the linear coefficient and a is the quadratic coefficient. Similarly, the variance is the reciprocal of the quadratic coefficient:

$$\hat{\sigma}_t^2 = \frac{1}{a} \tag{2.37}$$

2.3.4 Evaluation of variance $\hat{\sigma}_t^2$

To determine the variance $\hat{\sigma}_t^2$ we start by transforming a into a common denominator form.

$$a = \frac{\alpha_t}{1-\alpha_t} + \frac{1}{1-\gamma_{t-1}} = \frac{\alpha_t(1-\gamma_{t-1}) + 1 - \alpha_t}{(1-\alpha_t)(1-\gamma_{t-1})} = \frac{\alpha_t - \alpha_t\gamma_{t-1} + 1 - \alpha_t}{(1-\alpha_t)(1-\gamma_{t-1})} = \frac{1-\alpha_t\gamma_{t-1}}{(1-\alpha_t)(1-\gamma_{t-1})} \tag{2.38}$$

Recall that $\gamma_t = \prod_{i=1}^{t} \alpha_i$ is the cumulative product of all α values up to timestep t, which means $\gamma_t = \alpha_t \cdot \gamma_{t-1}$. Having that in mind, we end up with the final posterior variance after taking the inverse of a as shown in Equation **??**

$$\hat{\sigma}_t^2 = \frac{(1-\alpha_t)(1-\gamma_{t-1})}{1-\gamma_t} \tag{2.39}$$

2.3.5 Evaluation of mean $\hat{\mu}_t$

To compute the mean, we substitute our expressions for a and b into the formula $\hat{\mu}_t = -\frac{b}{2a}$. The calculation involves careful algebraic manipulation:

$$\hat{\mu}_t = -\frac{b}{2a} = -\frac{-\left(\frac{2\sqrt{\alpha_t}x_t}{1-\alpha_t} + \frac{2\sqrt{\gamma_{t-1}}x_0}{1-\gamma_{t-1}}\right)}{2\frac{1-\gamma_t}{(1-\alpha_t)(1-\gamma_{t-1})}} = \frac{\frac{\sqrt{\alpha_t}}{1-\alpha_t}x_t + \frac{\sqrt{\gamma_{t-1}}}{1-\gamma_{t-1}}x_0}{\frac{1-\gamma_t}{(1-\alpha_t)(1-\gamma_{t-1})}} \tag{2.40}$$

After canceling common terms $(1-\alpha_t)$ and $(1-\gamma_{t-1})$ from numerator and denominator, we obtain the final expression for the posterior mean:

$$\hat{\mu}_t = \frac{\sqrt{\alpha_t}(1-\gamma_{t-1})}{1-\gamma_t}x_t + \frac{\sqrt{\gamma_{t-1}}(1-\alpha_t)}{1-\gamma_t}x_0 \tag{2.41}$$

2.4 Summary

We have successfully derived the exact Gaussian form of the posterior distribution $q(x_{t-1} \mid x_t, x_0)$. This result is remarkable because it provides a closed-form expression that depends only on the forward process parameters α_t and γ_t. The posterior mean $\hat{\mu}_t$ is a weighted combination of the current noisy observation x_t and the clean data x_0, while the variance $\hat{\sigma}_t^2$ depends purely on the noise schedule. This tractable posterior serves as the ground truth target for training neural networks in diffusion models. With these explicit formulas, we can now proceed to design practical neural network parameterizations.

Chapter 3

Training Design

Having developed the complete mathematical framework for diffusion models in the previous chapters, we now turn to the practical challenge of implementation. Chapter 1 established the conceptual foundation of how diffusion models work, while Chapter 2 provided the theoretical machinery through variational inference. This chapter bridges theory and practice by addressing the crucial question: *How do we actually train a diffusion model?*

The journey from mathematical elegance to working code involves several key decisions and challenges:

- **Model Parametrization**: How should we represent the denoising function $p_\theta(x_{t-1}|x_t)$ with neural networks? What should the network predict?
- **Loss Function Design**: How do we convert the theoretical ELBO into practical, computable loss functions that can be optimized with gradient descent?
- **Training Stability**: What are the practical challenges that arise when we attempt to train these models, and how do we address them?

This chapter presents the first, most natural approaches to these questions. As we'll discover, the obvious solutions often reveal important limitations that guide us toward more sophisticated techniques. This represents the beginning of our practical journey with diffusion models - establishing a baseline implementation that works, while identifying areas for future improvement.

By the end of this chapter, you will have a complete, trainable diffusion model implementation, along with an understanding of its strengths, limitations, and opportunities for enhancement. Chapters 4 and 5 will build upon this baseline, introducing advanced techniques that significantly improve training efficiency and generation quality.

3.1 Model Parameterization

Having derived the exact Gaussian form of the posterior $q(x_{t-1} \mid x_t, x_0)$ with parameters $\hat{\mu}_t$ and $\hat{\sigma}_t^2$, we now face the practical question: how do we design neural networks to approximate this distribution?

The key insight is that since the true posterior is Gaussian, our neural network approximation $p_\theta(x_{t-1} \mid x_t)$ should also be Gaussian. This leads us to parameterize the reverse process

with learnable mean $\mu_\theta(x_t, t)$ and covariance $\Sigma_\theta(x_t, t)$ that approximate the true posterior parameters. This approach preserves the probabilistic nature of the process while allowing us to predict distributions rather than point estimates. We parameterize the reverse step as shown in Equation **??**.

$$p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(x_{t-1} \mid \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)) \tag{3.1}$$

The most direct approach is to design a neural network that outputs both the mean and covariance parameters directly, as illustrated in Figure **??**.

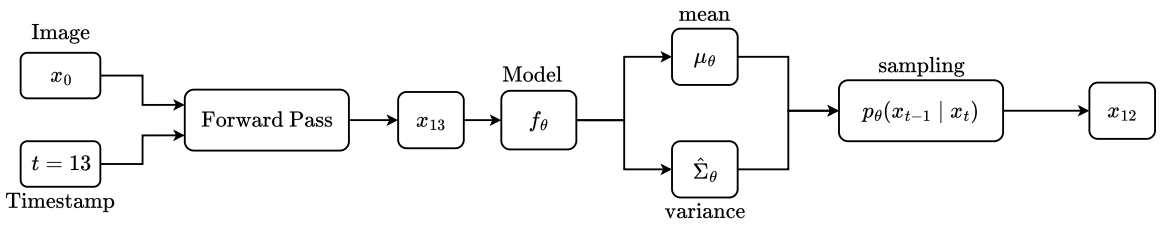


Figure 3.1: Neural network parameterization of the reverse process, predicting mean and variance from noisy input.

For μ_θ, this is straightforward and can be achieved by outputting a tensor of the same shape as the input image. Consider RGB images with shape $(H, W, 3)$: the output tensor becomes $(H, W, 3)$, modeling mean value per pixel per channel.

Variance modeling presents a significant challenge due to the sheer size of the covariance matrix. Take a standard image with dimensions $(256, 256, 3)$ which contains 196,608 pixels. A full covariance matrix would have shape $(196,608 \times 196,608)$, resulting in roughly 3.87×10^{10} entries. Storing, computing, and differentiating through such a structure is completely infeasible in practice. This raises an important question: do we really need to model the full covariance matrix?

Fortunately, full covariance modeling is unnecessary. The workaround lies in assuming that the covariance matrix is diagonal. This means we assume that each pixel (and color channel) in the image has independent noise, and there are no off-diagonal correlations between them. In other words, we represent the covariance as a simple vector of per-pixel variances, rather than a full matrix of pixel-pixel interactions.

3.1.1 Diagonal Covariance Matrix

Under the diagonal matrix assumption, instead of needing a $(196,608 \times 196,608)$ matrix, we only need to predict a variance value for each of the 196,608 pixels. This yields a dramatic reduction from tens of billions of parameters to just hundreds of thousands, making the problem computationally tractable.

The diagonal assumption is justified because spatial dependencies (edges, textures, shapes, and other visual structures) are already captured by the mean prediction $\mu_\theta(x_t, t)$. Since convolutional neural networks process the entire image context when predicting the mean, they naturally capture complex spatial relationships. The variance's role is different: it describes

the uncertainty in each pixel's value during sampling, which tends to be more local and independent across pixels. For illustration, consider a $(4, 4)$ image with the following diagonal covariance matrix:

$$\Sigma = \begin{bmatrix} \sigma_0^2 & 0 & 0 & 0 \\ 0 & \sigma_1^2 & 0 & 0 \\ 0 & 0 & \sigma_2^2 & 0 \\ 0 & 0 & 0 & \sigma_3^2 \end{bmatrix}$$

With this diagonal assumption, we can express the covariance matrix as:

$$\Sigma_\theta(x_t, t) = \text{diag}(\sigma_\theta^2(x_t, t)) \tag{3.2}$$

where $\sigma_\theta^2(x_t, t)$ is a tensor of per-pixel variances with the same spatial dimensions as the input. This means both the predicted mean μ_θ and variance σ_θ^2 can be represented as tensors of identical shape. In practice, we can stack them along the channel dimension, producing an output of shape $(H, W, 2C)$ for an input of shape (H, W, C).

3.1.2 Training Procedure

With all components defined, we can now consider the complete training procedure. For each training sample, we:

1. Sample a random timestep t and corrupt the clean image x_0 to x_t using the forward process
2. Compute the true posterior parameters $\hat{\mu}_t$ and $\hat{\sigma}_t^2$ from Chapter 2
3. Run the neural network to predict μ_θ and σ_θ^2
4. Evaluate the L_{VLB} loss and compute gradients

The complete procedure is outlined in Algorithm **??**.

Algorithm 1 Training Step Procedure

$x_0 \sim p(x_0)$
$t = \text{Uniform}(1, T)$
$x_t = \text{sample}(x_0, t)$
$\hat{\mu}_t = \text{posterior-mean}(x_0, t)$
$\hat{\sigma}_t^2 = \text{posterior-variance}(t)$
$\mu_\theta, \Sigma_\theta = \text{model}(x_t, t)$
$\text{loss} = L_{VLB}(\mu_\theta, \Sigma_\theta, \hat{\mu}_t, \hat{\sigma}_t^2)$
loss.backward()

Having established how to parameterize the model and defined the training procedure, we will complete the formulation of L_{VLB} in the next section.

3.2 Variational Lower Bound Loss Function

We now convert the variational lower bound from Chapter 2 into concrete, implementable loss functions. Recall that our ELBO decomposes into three distinct terms:

$$L_{VLB} = L_T + L_t + L_0$$

Each term presents unique computational and theoretical challenges that require careful treatment to create a practical training objective. In this section, we will:

1. Revisit the individual components of L_{VLB}, starting with L_T, and explain their relative contributions to the total loss.
2. Derive the closed-form KL divergence for the L_t term under the diagonal covariance assumption, simplifying it to an elementwise form.
3. Handle the L_0 term for discrete data, introducing a discretized Gaussian decoder and its efficient implementation.
4. Present the complete training and sampling procedures.

By the end of this section, we will have a fully specified and implementable L_{VLB} formulation, ready to be integrated into both the training and sampling stages of a denoising diffusion model.

3.2.1 L_T component

The first term, $L_T = D_{KL}(q(x_T \mid x_0) \mid\mid p_\theta(x_T))$, represents the divergence between the noisy data at the final timestep x_T and our chosen prior $p_\theta(x_T)$. Recall that as $t \to T$, both x_T and $p_\theta(x_T)$ converge to standard Gaussian $\mathcal{N}(0, \mathrm{I})$. Therefore D_{KL} becomes very small and contributes negligibly to the overall VLB loss. Additionally, the L_T term does not depend on any model parameters: the forward process is fixed and the prior distribution $p_\theta(x_T)$ is chosen rather than trained. This term can therefore often be ignored without impact on optimization.

3.2.2 L_t component

The middle term L_t represents the core of diffusion model training. This is where the neural network learns to approximate the true posterior at each timestep. For two Gaussian distributions, the KL divergence has a closed-form expression:

$$D_{KL}(q(x_{t-1} \mid x_t, x_0) \mid\mid p_\theta(x_{t-1} \mid x_t)) = \frac{1}{2}\Big[\log\frac{|\Sigma_p|}{|\Sigma_q|} - n + Tr(\Sigma_p^{-1}\Sigma_q) + (\mu_p - \mu_q)^T\Sigma_p^{-1}(\mu_p - \mu_q)\Big] \tag{3.3}$$

For practical implementation, we leverage our diagonal covariance assumption. Both the true posterior $\hat{\sigma}_t^2$ and predicted covariance σ_θ^2 have diagonal structure, allowing us to compute the KL divergence elementwise. This simplifies the multivariate KL formula to a scalar form applied to each pixel independently:

$$D_{KL}(q(x_{t-1}|x_t, x_0) \mid\mid p(x_{t-1} \mid x_t)) = \frac{1}{2}\Big[\log\frac{\sigma_\theta^2}{\hat{\sigma}^2} - 1 + \frac{\hat{\sigma}^2}{\sigma_\theta^2} + (\hat{\mu} - \mu_\theta)^2\frac{1}{\sigma_\theta^2}\Big] \tag{3.4}$$

This can be evaluated elementwise for each pixel and then averaged across the image. The term n describes the dimensionality of the distribution; when we deal with the elementwise case, it becomes 1.

3.2.3 Practical Implementation Details

For numerical stability, we compute the KL divergence using log variance rather than raw variance. This avoids undefined operations like log(0) and ensures stable gradients during training. To reflect this assumption, we first apply a standard logarithmic identity to express the log of a fraction as the difference of logs, as shown in Equation **??**:

$$\log \frac{\hat{\sigma}^2}{\sigma_\theta^2} = \log \hat{\sigma}^2 - \log \sigma_\theta^2 \tag{3.5}$$

The next transformation is less obvious but follows from the relationship between exponential and logarithmic functions. We rewrite the ratio of variances $\frac{\sigma_\theta^2}{\hat{\sigma}^2}$ using this property, as shown in Equation **??**.

$$\frac{a}{b} = \exp\left(\log \frac{a}{b}\right) = \exp\left(\log a - \log b\right) \tag{3.6}$$

Using the above property we rewrite **??** into the **??**.

$$D_{KL} = \frac{1}{2}\left[\log \hat{\sigma}^2 - \log \sigma_\theta^2 - 1 + \exp(\log \sigma_\theta^2 - \log \hat{\sigma}^2) + (\hat{\mu} - \mu_\theta)^2 \frac{1}{\exp(\log \hat{\sigma}^2)}\right] \tag{3.7}$$

3.2.4 L_0 and Discrete Decoder

The final term $L_0 = -\log p_\theta(x_0 \mid x_1)$ requires special treatment because it bridges the gap between our continuous Gaussian model and discrete pixel values. For images, x_0 contains discrete pixel values in the range $[0, 255]$. This creates a fundamental mismatch: our neural network outputs a continuous Gaussian distribution, but the target x_0 consists of discrete values. We cannot directly apply KL divergence between continuous and discrete distributions.

The solution is to discretize the continuous output distribution by integrating over bins centered around each pixel value.

We divide the continuous domain into bins, each centered on a discrete pixel value. Interior bins have width 1 (spanning ± 0.5 around the center), while edge bins are narrower since they extend only to the domain boundary. Given the definition, a value sampled from output distribution will be discretized into the value of which bin it falls. For example, output value 97.52 it will be assigned into 98. So ultimately, the task becomes maximizing the likelihood that a value sampled from the output distribution falls into the right bin. To visualize the approach, consider the trained distribution for a single individual pixel value.

To identify this probability we can use the Cumulative Distribution Function (CDF). Assume target pixel value $x_0^i = 127$. Based on bin definition we set it to $\text{bin}(x_0^i) = [126.5, 127.5]$. The probability is shown in Equation **??** based on CDF definition.

$$P(126.5 < x_0^i \leq 127.5 \mid x_1^i) = \Phi(127.5) - \Phi(126.5) = \int_{126.5}^{127.5} \mathcal{N}(x_0^i \mid \mu, \sigma^2) dx_0^i \tag{3.8}$$

Visually it is shown in the Figure **??**.

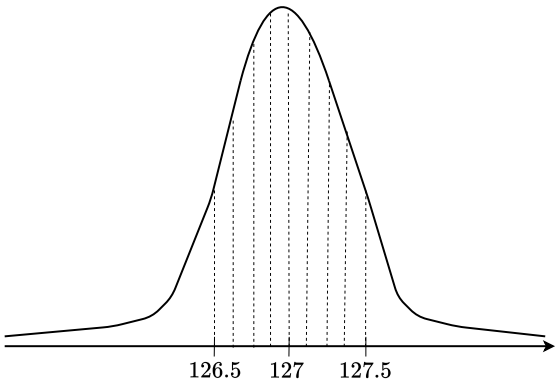


Figure 3.2: Example of discretized Gaussian likelihood for a single pixel value. The shaded area represents the probability mass within the bin boundaries.

Computing the integral for each pixel can be computationally expensive; however, the CDF can be efficiently approximated using the method proposed by Page (1995), as presented in Equation **??**.

$$\Phi(x) = \frac{1}{2}\left(1 + \tanh\left(\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right)\right) \tag{3.9}$$

This approximation is originally defined for the standard Gaussian distribution $\mathcal{N}(0, 1)$. To apply it in our setting, we first standardize the distribution to center it around zero using Z-Score formula.

$$Z = \frac{X - \mu}{\sigma}$$

Given that we have two bin boundaries, standardization must be performed twice, for as shown in Equation **??**.

$$z^i_{0\pm} = \frac{(x^i_0 \pm 0.5) - \mu}{\sigma} \tag{3.10}$$

Since pixels are assumed to be independent, the likelihood of the entire image can be expressed as the product of individual pixel distributions.

$$p_\theta(x_0 \mid x_1) = \prod_{i=1}^{D} p_\theta(x^i_0 \mid x^i_1)$$

However in the L_{VLB}, we are dealing with the logarithm of the discrete likelihood. Application of logarithm transforms product into a summation, as shown in Equation **??**.

$$\log p_\theta(x_0 \mid x_1) = \sum_{i=1}^{D} \log p_\theta(x^i_0 \mid x^i_1) \tag{3.11}$$

Discretized Decoder Implementation Details

For the purpose of explaining the L_0 component, we previously introduced a slight oversimplification. Specifically, working with pixel values in the range $(0, 255)$ is impractical in

deep learning. In practice, normalized value ranges such as $(0, 1)$ or $(-1, 1)$ are commonly used. In our implementation, we use the range $(-1, 1)$ centered at zero. Due to this change in value range, the bin width must also be adjusted accordingly, becoming $\frac{2}{255}$.

The boundary conditions at ± 0.999 handle edge cases where pixel values approach the extremes of the valid range. For pixels near the lower bound ($x < -0.999$), we compute the CDF from negative infinity to the upper boundary, representing the total probability mass for all values below the quantization threshold. Similarly, for pixels near the upper bound ($x > 0.999$), we compute the CDF from the lower boundary to positive infinity. This treatment prevents numerical instabilities that would arise from evaluating the Gaussian CDF at the precise boundaries of the quantization interval. It also ensures that the total probability mass is properly conserved across the discrete pixel space.

Algorithm 2 Discretized Gaussian Log Likelihood

Require: x, μ, σ^2
Ensure: log likelihood of discretized Gaussian distribution
1: step $= 1/255$
2: $x = x - \mu$
3: upper $= (x + \text{step})/\sigma$
4: lower $= (x - \text{step})/\sigma$
5: **if** $x_i < -0.999$ **then**
6: $\quad L_0(x_i) = \Phi(\text{upper})$
7: **else if** $x_i > 0.999$ **then**
8: $\quad L_0(x_i) = 1 - \Phi(\text{lower})$
9: **else**
10: $\quad L_0(x_i) = \Phi(\text{upper}) - \Phi(\text{lower})$
11: **return** $\log(\text{clamp}(L_0, 10^{-12}))$

This completes the derivation of the exact formula and the implementation of L_{VLB}. At this stage it is ready to be used in training procedure. We have now derived the complete, implementable form of the variational lower bound loss function for diffusion models. The key achievements of this chapter:

- **Tractable loss decomposition**: Converting the theoretical ELBO into three practical components (L_T, L_t, L_0)
- **Closed-form KL divergences**: Leveraging the diagonal covariance assumption for efficient computation
- **Discrete data handling**: Bridging continuous models and discrete pixels through bin integration
- **Complete procedures**: Providing algorithmic specifications for both training and sampling

This framework provides the foundation for implementing and training diffusion models, though practical considerations like architectural choices, hyperparameter tuning, and computational optimizations will further influence real-world performance.

Chapter 4

First Attempts and Failures

Having established the complete theoretical framework in previous chapters, we now transition from mathematical elegance to practical experiments that will expose lack of practical feasibility. This chapter documents first series of training experiments, emphasizing the iterative nature of research and the importance of understanding why certain approaches fail.

4.1 Experimental Setup and Limitations

Before presenting our results, it is important to establish realistic expectations regarding computational resources and experimental scope. The original DDPM paper [**ho2020denoising**] reports training their CIFAR-10 model with 35.7 million parameters on TPU v3-8 (equivalent to 8 V100 GPUs) for 800,000 steps, achieving 21 steps per second at batch size 128, requiring approximately 10.6 hours for completion. Such computational infrastructure represents a significant investment not accessible to most indyvidual researchers or practitioners.

Our experimental setup operates under considerably more modest constraints: dual GTX 1080 Ti GPUs with training sessions limited to approximately 2.5 hours. While our model architecture maintains a similar parameter count to the original work, the reduced computational budget and training duration prevent direct quantitative comparison with published state-of-the-art results. However, these limitations do not diminish the value of our investigation.

Unless otherwise stated, all experiments share the following standardized configuration:

- **Architecture:** UNet-based denoiser 37.5M parameters
- **Optimizer:** AdamW with learning rate 2×10^{-4}
- **Datasets:** CIFAR-10
- **Metrics:** Training Loss, Gradient Norm, Sampling NLL, Mean MSE.

4.2 Experiment 1: Direct Mean and Direct Variance

The most natural starting point follows directly from our theoretical derivation: train a neural network to predict directly both the mean and variance of the $p_\theta(x_{t-1}|x_t)$ distribution as presented in Equation **??**.

$$(\mu_\theta, \sigma_\theta^2) = f_\theta(x_t, t) \tag{4.1}$$

The network outputs $2C$ channels, where the first C channels represent the mean and the last C channels represent the variance as shown in Figure **??**.

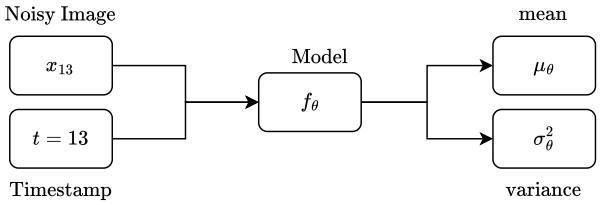


Figure 4.1: Experiment 1: direct-direct-vlb parameterization.

We let this experiment run for 30 epochs and present the training characteristics in Figure **??**. Weobserve severe gradient spikes to very large values, the mean MSE remains at levels indicative of poor approximation quality and appears to be stuck in a local minimum. Most critically, the negative log-likelihood values remained orders of magnitude higher than expected for well-behaved generative models, suggesting fundamental issues with the model's ability to capture the underlying data distribution.

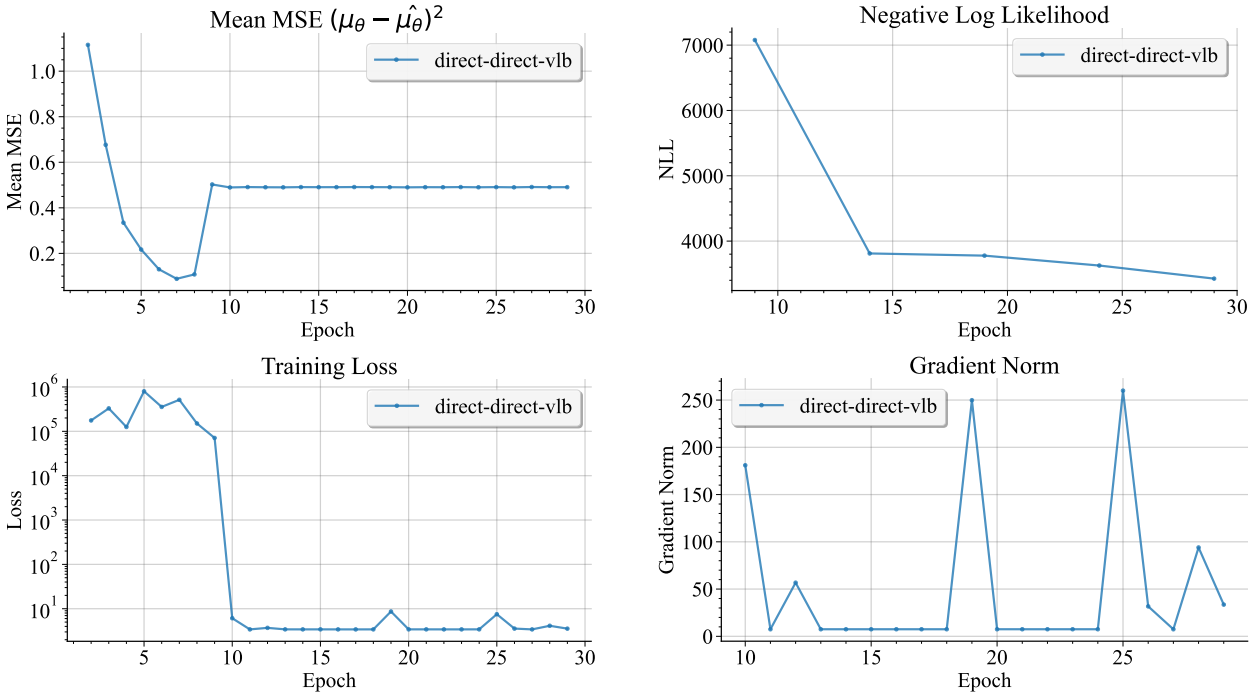


Figure 4.2: Training characteristics of direct-direct-vlb parameterization.

After running sampling procedure, the output images did not differ much from random noise as presented in Figure **??**.

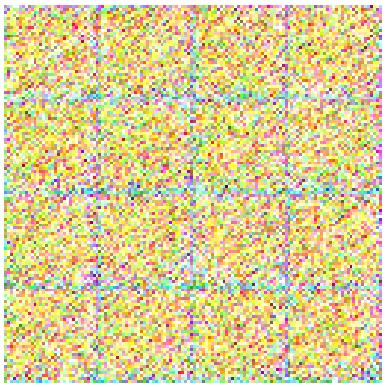


Figure 4.3: Sample images in Experiment 1

In order to understand the root cause of the instabilities and very high values during the training we need examine the gradient behavior of the loss function. To do so, we evaluate the derivative of $L_t(\mu_\theta, \sigma_\theta^2)$ with respect to σ_θ^2 as shown in Equation **??**.

$$\frac{\partial L_t(\mu_\theta, \sigma_\theta^2)}{\partial \sigma_\theta^2} = \frac{\partial}{\partial \sigma_\theta^2}\frac{1}{2}\Big[\log\frac{\sigma_\theta^2}{\hat{\sigma}^2} - 1 + \frac{\hat{\sigma}^2}{\sigma_\theta^2} + \frac{(\hat{\mu}-\mu_\theta)^2}{\sigma_\theta^2}\Big] = \frac{1}{2}\left[\frac{1}{\sigma_\theta^2} - \frac{\hat{\sigma}^2}{(\sigma_\theta^2)^2} - \frac{(\hat{\mu}-\mu_\theta)^2}{(\sigma_\theta^2)^2}\right] \tag{4.2}$$

Recall that the target range of posterior variance is $\hat{\sigma}^2 \in [0, 0.02]$. With that in mind the denominators $(\sigma_\theta^2)^2$ may be dangerous and cause extremely large scaling due to the square of values close to zero. For the purpose of illustration suppose that we select some timestep t at which $\hat{\sigma}^2 = 0.01$, mean error of $(\hat{\mu} - \mu_\theta)^2 = 0.05$. Consider underestimation scenario where the network outputs $\sigma_\theta^2 = 0.001$ as shown below.

$$\frac{\partial L_t}{\partial \sigma_\theta^2} = \frac{1}{2}\left[\frac{1}{0.0001} - \frac{0.01}{(0.0001)^2} - \frac{0.05}{(0.0001)^2}\right] = -2.995 \times 10^6.$$

This example demonstrates how small variance predictions can lead to catastrophic gradient magnitudes. It implies large gradient fluctuations. Even if gradient clipping prevents divergence, repeatedly hitting the clipping threshold is a sign that the parameterization yields poorly scaled updates.

In the next experiment we will investigate simple trick which may make the optimization more stable.

4.3 Experiment 2: Predicting Log Variance

In order to deal with numerical instabilities, log function is used very often. Instead of predicting σ_θ^2 directly, we can parametrize model such as it predicts logarithm of variance $\log \sigma_\theta^2$ as shown in Equation **??**.

$$(\mu_\theta, \log \sigma_\theta^2) = f_\theta(x_t, t) \tag{4.3}$$

Such reparameterization does not require any architectural modifications, we simply compute $\sigma_\theta^2 = \exp(\log \sigma_\theta^2)$ when needed. In order to understand the difference in gradient flow we

examine the derivative of L_t with respect to $s = \log \sigma_\theta^2$ as shown in Equation **??**.

$$\begin{aligned}\frac{\partial L_t}{\partial s} &= \frac{\partial}{\partial s}\frac{1}{2}\Big[\log\frac{\exp(s)}{\hat{\sigma}^2} - 1 + \frac{\hat{\sigma}^2}{\exp(s)} + \frac{(\hat{\mu}-\mu_\theta)^2}{\exp(s)}\Big] \\ &= \frac{1}{2}\frac{\partial}{\partial s}\Big[s - \log(\hat{\sigma}^2) - 1 + \frac{\hat{\sigma}^2}{\exp(s)} + \frac{(\hat{\mu}-\mu_\theta)^2}{\exp(s)}\Big] \\ &= \frac{1}{2}\Big[1 - \frac{\hat{\sigma}^2}{\exp(s)} - \frac{(\hat{\mu}-\mu_\theta)^2}{\exp(s)}\Big] = \frac{1}{2}\Big[1 - \frac{\hat{\sigma}^2}{\sigma_\theta^2} - \frac{(\hat{\mu}-\mu_\theta)^2}{\sigma_\theta^2}\Big]\end{aligned} \tag{4.4}$$

The improvement is apparent as the problematic denominators grow lineary rather than with square. This reduces the gradient blow-up by an entire order of magnitude, which should stabilize the training dynamics. To quantify this improvement, we apply the same numerical example from Experiment 1

$$\frac{\partial L_t}{\partial \sigma_\theta^2} = \frac{1}{2}\left[1 - \frac{0.01}{0.0001} - \frac{0.05}{0.0001}\right] = -299.5.$$

The gradient magnitudes are orders of magnitude smaller than in the direct variance case, effectively eliminating the catastrophic spikes that plagued the previous approach which gives hopes that it will stabilize the training. See the training characteristics of Experiment 2 in Figure **??**.

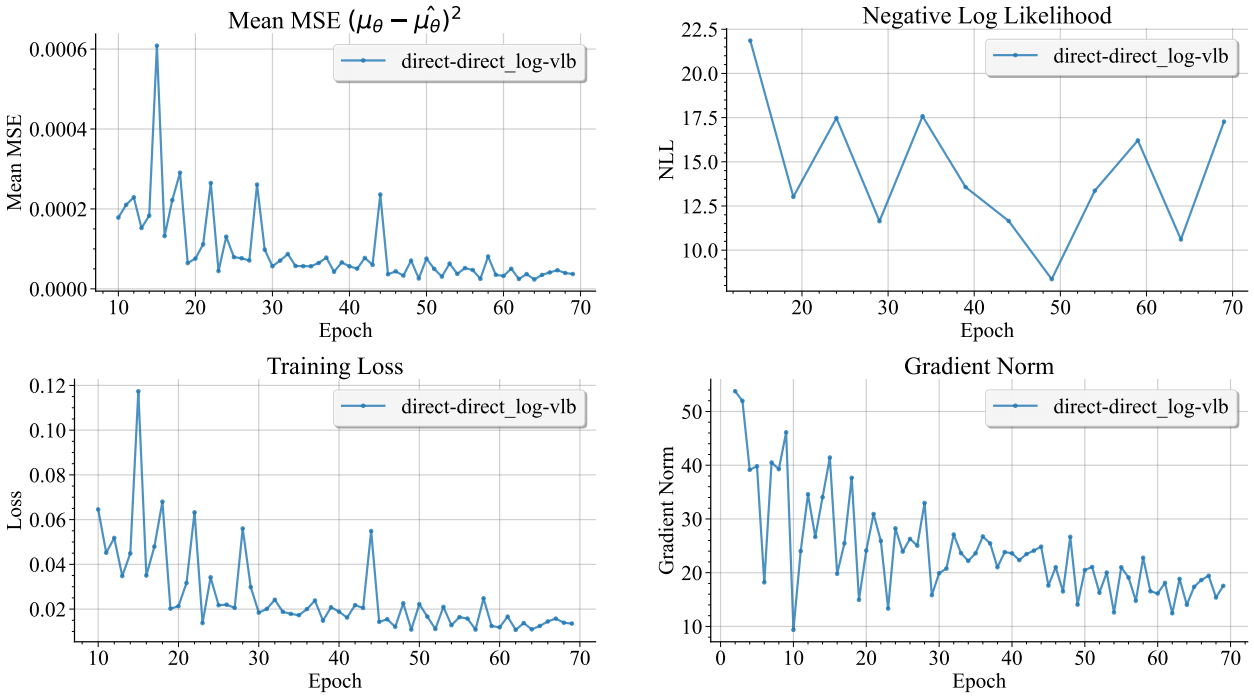


Figure 4.4: Training characteristics of direct-direct_log-vlb parameterization.

Results confirm that training was more stable. There are significantly lower metric values by orders of magnitude and a noticeable convergence curve, which is a good sign. On the other

hand, the performance of NLL is worrying as it is not showing continuous improvement during training and seems to oscillate around some value. The perceptual quality of the generated samples has two sides. Firstly, the images look far from random noise; there are basic image features and blobs. However, in terms of generating actual objects, the results were very poor and bore no visual resemblance to the original training images. The outputs continued to fail to form coherent objects, often degenerating into amorphous, blob-like structures.

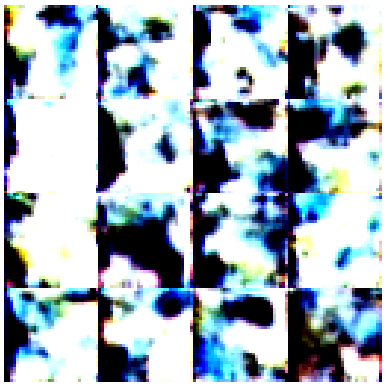


Figure 4.5: Sample images in Experiment 2

This observation suggests that seemingly low loss values and smooth training curves do not necessarily translate into strong generation quality. In our case, training appears stable while the sampled images remain poor, indicating that stability alone is not the limiting factor. A natural next step is to examine where the modeling error concentrates—starting with the accuracy of the mean estimator.

4.4 Experiment 3: Fixed Variance

The results of Experiment 2 demonstrate that variance prediction can be made numerically stable by selecting the correct parameterisation. However, this raises the question of whether it is necessary to predict the variance at all.

Direct inspection of the posterior variance in Equation **??** provides a simple yet significant observation. The target variance $\hat{\sigma}_t^2$ does not depend on the noisy image x_t nor the clean image x_0.

$$\hat{\sigma}_t^2 = \frac{(1-\alpha_t)(1-\gamma_{t-1})}{1-\gamma_t} \tag{4.5}$$

It is fully determined by the timestep t via the predefined noise schedule parameters. Consequently, during training, the exact target variance is known at every timestep. One might ask whether it is necessary to train the model to predict this, or whether it would be better to simply use a fixed and known value ($\hat{\sigma^2}$). In order to verify this, we formulate the Hypothesis **??**.

Hypothesis 4 *There is no need to optimise the variance, as it is sufficient to use the known posterior variance $\sigma_\theta^2 = \hat{\sigma}^2$ instead.*

To evaluate hypothesis **??**, we modify the training setup by setting variance to the fixed value and the network will predict only the mean μ_θ as presented in Figure **??**.

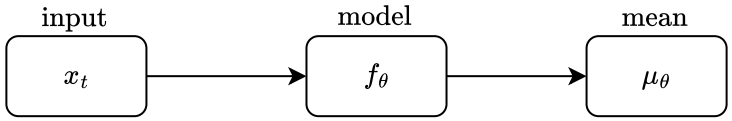


Figure 4.6: Fixed Variance parameterization.

This procedure is fully compatible with our configuration. Only the first C channels of the model will be used, so there is no need to worry about the loss function at this point, as it remains compatible. See the results of Experiment 3 and direct comparison with Experiment 2 in Figure **??**.

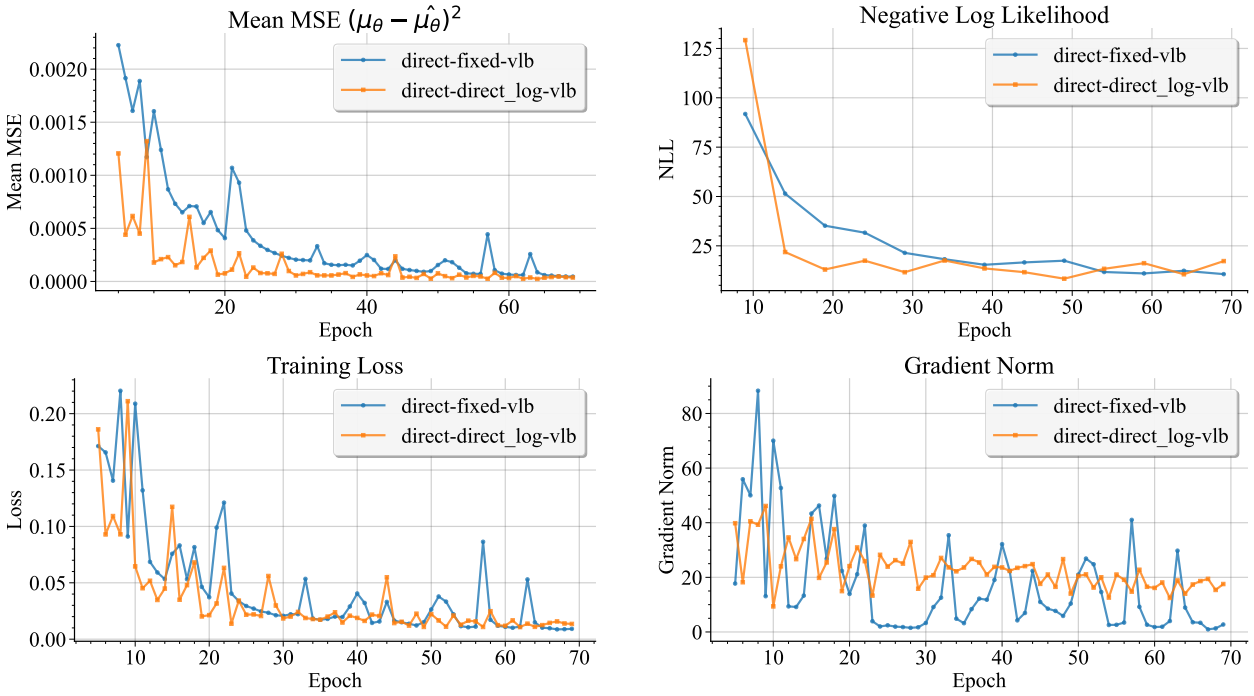


Figure 4.7: Training characteristics of direct-fixed-vlb parameterization.

Both runs appear similar and achieve comparable performance at the end of the training process. The trainable variance configuration appears to exhibit smaller gradient norm fluctuations. This may be because the posterior variance is only optimal when the mean is perfectly predicted, which never occurs. Learned variance adapts to actual mean errors, increasing uncertainty when predictions are inaccurate in order to reduce penalties and stabilise early optimisation.

The quality of the generated images still leaves much to be desired. They resemble those from Experiment 2, only with a different tone. As before, nothing resembling real objects can be found in them. See the samples in Figure **??**.

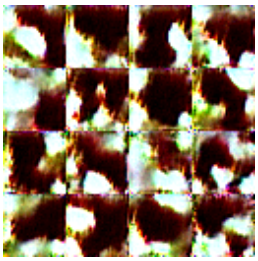


Figure 4.8: Sample images in Experiment 3

4.5 Experiment 4: Simpler Training Objective

Taking into account the previous results and the images obtained, it can be concluded that firstly, the accuracy of estimating $\hat{\mu}$ is insufficient and secondly, the gradient characteristics are still not ideal, showing high levels of variance and fluctuation. Before analysing the estimation of $\hat{\mu}$, we can apply a simple change to the loss function that may stabilise the gradient flow. After setting the variance to a fixed value $\hat{\sigma}^2$, see what form VLB takes as shown in Equation **??**.

$$\begin{aligned} L_t(\mu_\theta, \sigma_\theta^2) &= \frac{1}{2}\Big[\log\frac{\sigma_\theta^2}{\hat{\sigma}_t^2} - 1 + \frac{\hat{\sigma}_t^2}{\sigma_\theta^2} + \frac{(\hat{\mu}-\mu_\theta)^2}{\sigma_\theta^2}\Big] = \frac{1}{2}\Big[\log\frac{\hat{\sigma}_t^2}{\hat{\sigma}_t^2} - 1 + \frac{\hat{\sigma}_t^2}{\hat{\sigma}_t^2} + \frac{(\hat{\mu}-\mu_\theta)^2}{\hat{\sigma}_t^2}\Big] \\ &= \frac{1}{2}\left[\log(1) + 1 + \frac{(\hat{\mu}-\mu_\theta)^2}{\hat{\sigma}_t^2} - 1\right] = \frac{1}{2\hat{\sigma}_t^2}(\hat{\mu}-\mu_\theta)^2 \end{aligned} \tag{4.6}$$

It simplifies into a weighted MSE on mean prediction. The weightning factor functions solely as a scaling factor for the gradients, which varies across timesteps. The fundamental distinction between unweighted and weighted versions is tension between statistical correctness and practical optimization dynamics. The unweighted formulation treats all timesteps equally, allowing the network to distribute its representational capacity uniformly across the entire denoising trajectory. To illustrate the impact Figure **??** presents the weighting values across different timesteps.

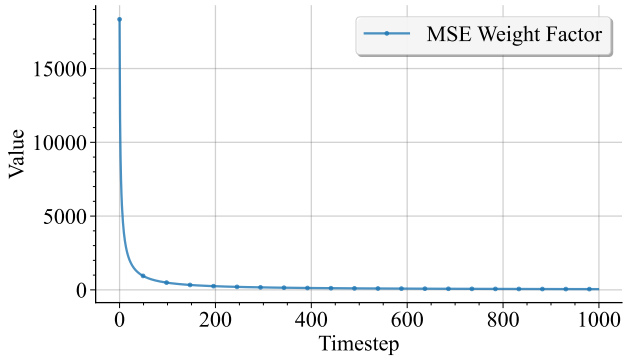


Figure 4.9: Weighting factor $\hat{\sigma}^2$ across timesteps.

We observe quite wide range of approximately $[50, 10^5]$ which may cause severe training imbalances, where early timesteps dominate the loss function and gradient updates. A prac-

tical engineering solution could be to disable the weighting factor entirely, reducing the loss to a simple unweighted MSE objective that treats all timesteps equally as shown in Equation ??.

$$L_t(\mu_\theta, \sigma_\theta^2) = (\hat{\mu} - \mu_\theta)^2 \tag{4.7}$$

In order to verify the effectiveness of this approach, we conduct Experiment 4 using the unweighted MSE loss as described in Equation ??, while keeping all other configurations identical to Experiment 3. See the training characteristics in Figure ??.

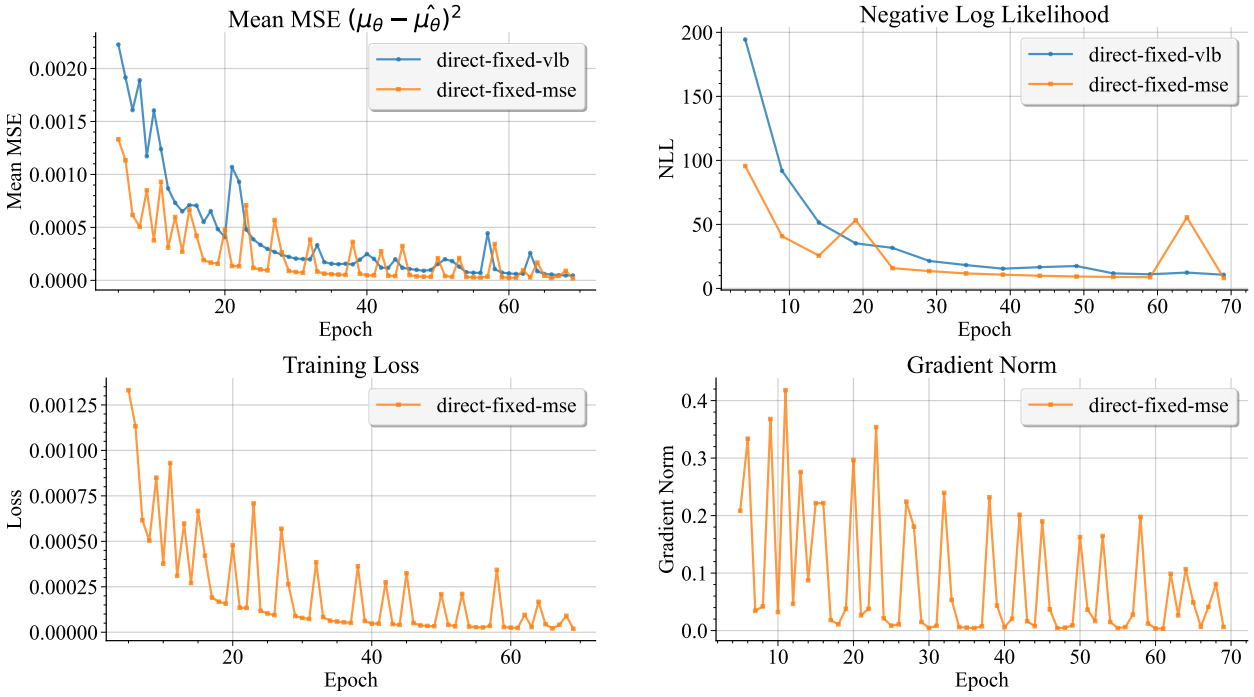


Figure 4.10: Training characteristics of direct-fixed-vlb parameterization.

The metric curves demonstrate that the convergence dynamics of both the mean MSE and the NLL are similar to those observed in the previous experiment. One positive difference is that the unweighted variant converges slightly faster in early epochs. Another positive outcome is that the gradient magnitudes have decreased significantly compared to before as assumed to happen in this experiment. However, it is still somewhat concerning that the trend does not resemble a clean, well-behaved convergence curve. Overall, the trajectories are fairly interpretable, with clear fluctuations visible throughout training. Please note that the training loss and gradient-norm traces shown in the earlier experiment have been omitted here, since they are on a different value scale and are therefore not directly comparable on the same chart.

After running the sampling procedure, we obtain the images presented in Figure ??.

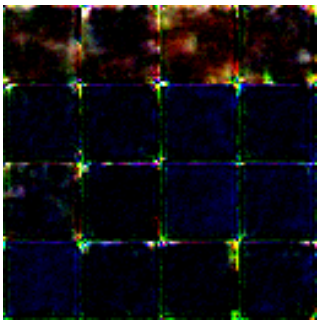
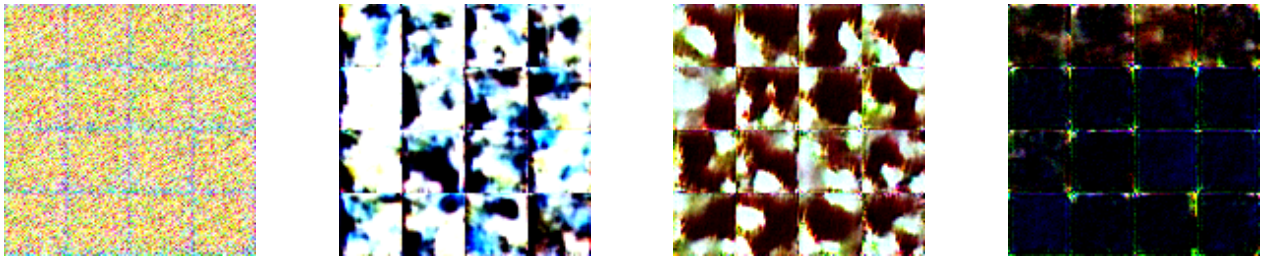


Figure 4.11: Sample images in Experiment 4

In the absence of any improvement in the absolute values of the metrics, the generated images did not resemble the original objects. On the other hand, it was an interesting journey through different ideas and improvements. As a summary of this chapter, see the sampled images from each consecutive experiment and observe how they evolved from random noise in Figure **??**.



(a) Experiment 1 (b) Experiment 2 (c) Experiment 3 (d) Experiment 4

Figure 4.12: Sample quality progression across all four experiments.

4.6 Summary and Implications

In simple words, the experiments weren't successfull. None of the models was capable of generating high quality samples, even though the theoretical framework was carefuly implemented. That was also an intention to show that not always formulations success in practice. Nevertheless we observed and learned a lot.

In order to proceed further, we need to identify better way of modelling μ_θ as it appears to be the main bottleneck: *Precision of mean estimation isn't good enough.*

Chapter 5

Gradual Improvements

Chapter 4 established that mean prediction accuracy constitutes the fundamental bottleneck in diffusion model training. Although we successfully addressed gradient scaling issues and training stability, the core challenge of learning accurate mean estimates remained unresolved. This chapter explores alternative parameterization strategies that exploit the mathematical structure of the posterior mean to overcome these limitations.

We begin by analyzing why direct mean prediction proves difficult, then investigate alternative learning targets that may offer more favorable optimization landscapes while maintaining theoretical compatibility with the established framework.

5.1 Understanding Mean Estimation Difficulties

The posterior mean $\hat{\mu}_t$ presents a challenging optimization target due to its composite structure. From Chapter 2, we know it can be expressed as a weighted combination of the current noisy observation x_t and the clean target x_0:

$$\hat{\mu}_t = C_t \cdot x_t + C_0 \cdot x_0 \tag{5.1}$$

where C_t and C_0 are timestep-dependent coefficients determined by the noise schedule. When the network directly predicts $\hat{\mu}_t$, it faces a compound learning problem: it must simultaneously recover the clean image x_0 from the noisy input while also learning the correct timestep-dependent weighting between the noisy observation and the denoised result. This creates optimization difficulties because:

- **Entangled objectives**: The network cannot separately optimize denoising quality and coefficient application
- **Timestep complexity**: Different timesteps require dramatically different mixing ratios, forcing the network to learn a complex time-varying transformation
- **Loss signal dilution**: Errors in the reconstruction get mixed with errors in the weighting scheme, making gradient signals less informative

5.2 Experiment 5: XStart Mean Strategy

An interesting direction is to parametrize network to predict clean image x_0 instead of the reversed process mean as showed in Figure **??**.

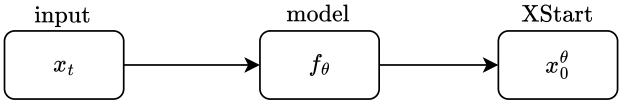


Figure 5.1: Neural network parameterization with XStart Mean strategy.

Crucially, this straightforward method of presenting neural network output can be fully aligned with our existing formulations through already known algebraic transformation. We can analytically compute the posterior mean μ_θ from the predicted clean image x_0^θ using the established posterior formula which we recall in Equation **??**.

$$\mu_\theta = \frac{\sqrt{\alpha_t}(1-\gamma_{t-1})}{1-\gamma_t}x_t + \frac{\sqrt{\gamma_{t-1}}(1-\alpha_t)}{1-\gamma_t}x_0^\theta \tag{5.2}$$

As x_t is known and output of the neural network provides x_0^θ, we can directly compute μ_θ whenever needed as shown in Figure **??**.

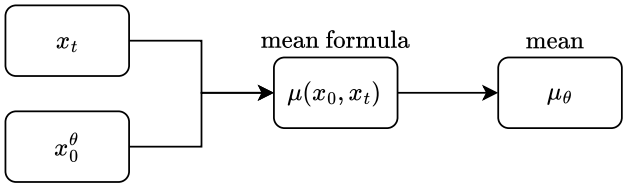


Figure 5.2: Computation μ_θ from predicted clean image x_0^θ.

The computed μ_θ can then be employed in the L_{VLB} loss function, maintaining mathematical consistency with our established framework.

To evaluate the XStart mean parameterization, we adopt the experimental setup from Chapter 4, specifically employing the fixed variance configuration while modifying only the mean parameterization. This choice isolates the impact of the mean strategy, allowing direct comparison with prior results. See the training characteristics and comparison with Experiment 4 in Figure **??**. In order to improve visual clarity, we used log scale for Mean MSE and NLL charts.

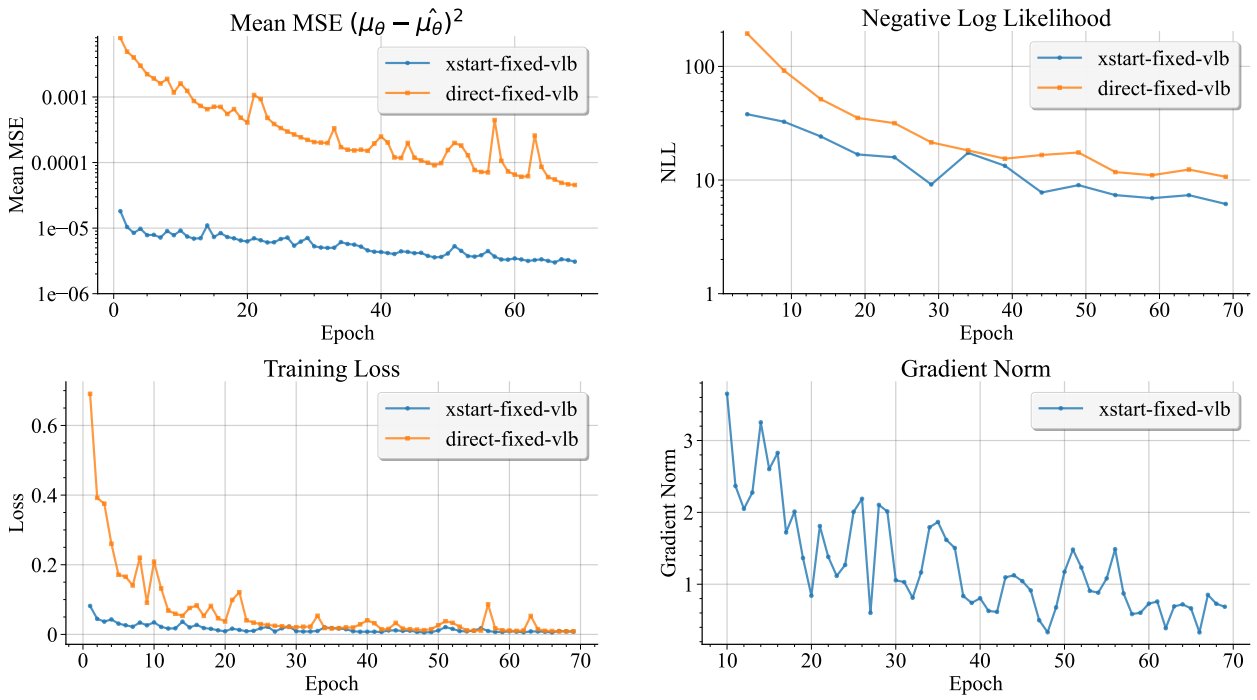
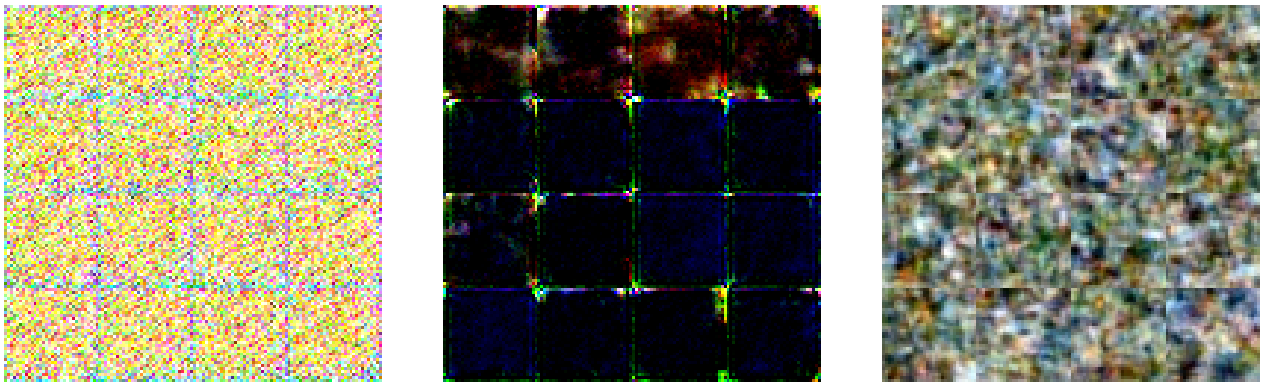


Figure 5.3: Training characteristics of xstart-fixed-vlb parameterization.

The most notable improvement in this experiment is the increased precision of mean estimation: mean MSE improves by approximately one order of magnitude, together with substantially quicker convergence. There is also a clear acceleration and improvement in NLL. Since both this experiment and the previous one optimize the same L_{VLB} the loss curves can be compared directly. The trend confirms a markedly more effective optimisation trajectory. Although the gradient norms still fluctuate, they remain within a relatively small range and decrease overall over the course of training.

In order to visualize the qualitative impact of the XStart strategy, we present generated samples in Figure **??** alongside samples from earlier experiments for direct comparison.



(a) Experiment 1 (b) Experiment 4 (c) Experiment 5

Figure 5.4: Generated samples comparison between Experiment 1, 4 and 5.

Despite the notable improvements across the quantitative metrics, we still do not obtain high-quality samples that convincingly represent real world entities. That said, the qualitative behavior differs substantially from earlier experiments: instead of collapsing into amorphous noise or random blobs, the current outputs more often exhibit recognizable basic shapes and repeated features. While far from satisfactory, this shift suggests the model is beginning to capture non-trivial image structure and provides a more promising starting point for further refinement.

5.3 Experiment 6: Simpler XStart Strategy

In Experiment 5, we leveraged the fact that the diffusion parameterization allows us to predict x_0^θ rather than the μ_θ explicitly thanks to the posterior mean formula which lets us retain the same L_{VLB} objective. Empirically, this improves the behavior of training as gradients are backpropagated through x_0^θ instead of directly through μ_θ. However it does not eliminate the underlying scaling issues highlighted in Experiment 4 which we also implicitly faced in the last experiment.

In order to understand scaling issues we need to analyze the loss function more closely and derive its explicit form when using XStart Mean Strategy. We know from Chapter 4 that the L_t has the following form for fixed variance setting:

$$L_t(\mu_\theta, \sigma_\theta^2) = \frac{1}{2\hat{\sigma}_t^2}(\hat{\mu}_t - \mu_\theta)^2$$

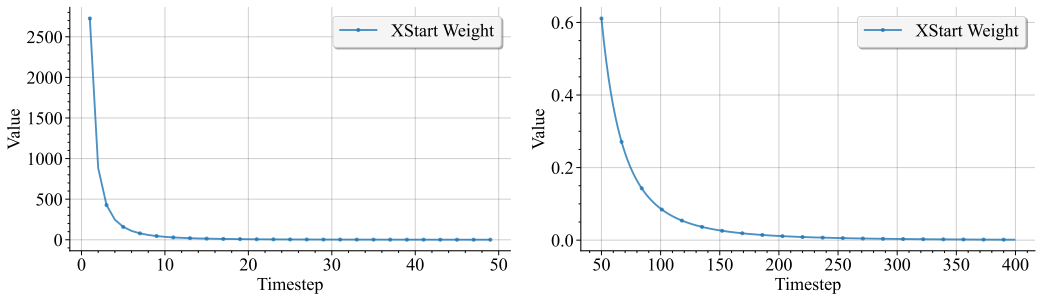
In addition to leverage the generalized form of the posterior mean from Equation **??**:

$$\mu_t = C_t \cdot x_t + C_0 \cdot x_0$$

We substitute it into the L_t formula Equation **??**.

$$\begin{aligned} L_t(\mu_\theta, \sigma_\theta^2) &= \frac{1}{2\hat{\sigma}_t^2}(\hat{\mu}_t - \mu_\theta)^2 \\ &= \frac{1}{2\hat{\sigma}_t^2}\left[(C_t \cdot x_t + C_0 \cdot x_0) - \left(C_t \cdot x_t + C_0 \cdot x_0^\theta\right)\right]^2 \\ &= \frac{1}{2\hat{\sigma}_t^2}\left(C_0(x_0 - x_0^\theta)\right)^2 = \frac{1}{2\hat{\sigma}^2}\left[\frac{\sqrt{\gamma_{t-1}}(1-\alpha_t)}{1-\gamma_t}\left(x_0 - x_0^\theta\right)\right]^2 \\ &= \frac{\gamma_{t-1}(1-\alpha_t)^2}{2\hat{\sigma}_t^2(1-\gamma_t)^2}(x_0 - x_0^\theta)^2 \end{aligned} \tag{5.3}$$

The terms involving x_t cancel as they appear identically in both expressions. Furthermore, C_0 can be factored out before squaring. It reveals that the XStart Strategy with fixed variance transforms the L_{VLB} into a straightforward MSE loss between the true and predicted clean images with a timestep dependent weighting factor. See the weighting factor value across timesteps in Figure **??**.



(a) Weighting factor during $t \in [1, 50]$ (b) Weighting factor during $t \in [50, 400]$

Figure 5.5: XStart weighting factor across timesteps.

At early timesteps up to $t = 10$, the factor exhibits large scaling with values ranging from 3000 to 100. This transitions to moderate values around 0.6 at timestep 50, then steadily decreases toward zero for later timesteps. This creates a severely imbalanced optimization landscape where early timesteps dominate the loss function while later timesteps contribute negligibly. Following the insights from Chapter 4 regarding the potential benefits of uniform timestep treatment, we eliminate this problematic weighting entirely as shown in Equation **??**:

$$L_t = (x_0 - x_0^\theta)^2 \tag{5.4}$$

In order to verify the effectiveness of this approach, we conduct Experiment 6 using the unweighted MSE loss as described in Equation **??**, while keeping all other configurations identical to Experiment 5. See the training characteristics in Figure

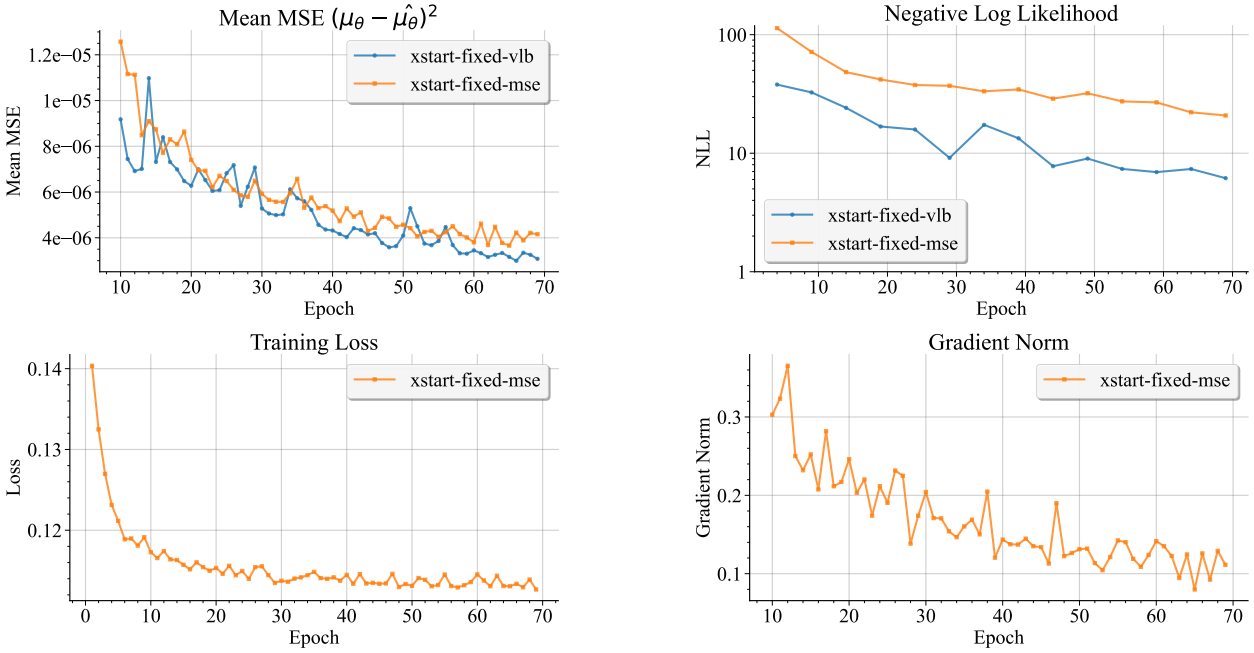
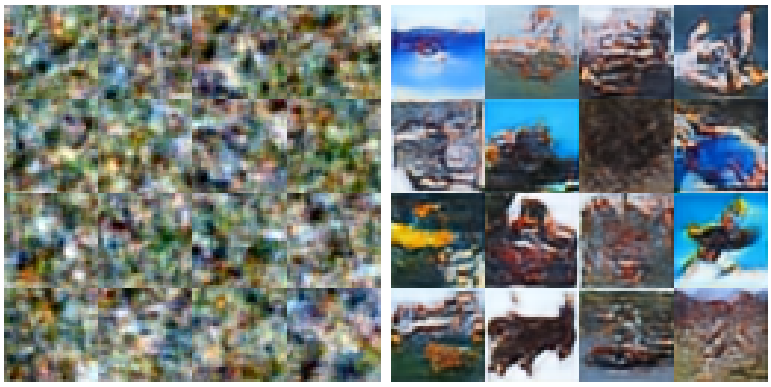


Figure 5.6: Training characteristics of xstart-fixed-mse parameterization.

The training loss curve remains stable throughout the run, indicating well behaved optimization dynamics. However, compared to the previous configuration, the mean estimation is similar but slightly worse, suggesting a small degradation in reconstruction precision. A comparable trend is observed for NLL, where the model exhibits similar behavior yet consistently underperforms, resulting in a slightly worse overall likelihood. See generated samples in Figure **??** alongside samples from earlier experiment for direct comparison.



(a) Experiment 5 (b) Experiment 6

Figure 5.7: Generated samples comparison between Experiment 5 and 6.

Although this experiment yields slightly worse Mean MSE and NLL, the generated samples represent a significant qualitative breakthrough. While the images remain far from ideal quality, we now observe more realistic and complex shapes and objects. They are still difficult to recognize specific categories but is is definitelly an improvement from provious experiments. It is worth noting that improved visual structure can arise even when these scalar metrics worsen slightly.

5.4 Experiment 7: Epsilon Mean Strategy

The promising results of the XStart approach encourage to explore other alternative parametrizations. In this section we will again explore relationship between clean image x_0 and added noise ϵ during creation of noisy images x_t. Essentially, image degradation is caused by the addition of noise. Perhaps it is possible to gain an advantage over this by designing a model that can predict what noise has been added to a noisy image rather than directly regressing the reverse step mean or x_0 as in previous experiment.

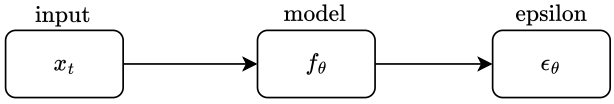


Figure 5.8: Neural network parameterization with XStart Mean strategy.

We recall the forward diffusion closed-form expression from Equation **??** which relates the clean image x_0, the noisy observation x_t, and the added noise ϵ.

$$x_t = \sqrt{\gamma_t} x_0 + \sqrt{1 - \gamma_t}\epsilon \tag{5.5}$$

We can rearrange Equation **??** in order to obtain formula for the clean image x_0 showing the dependence on both the noisy observation x_t and the added noise ϵ as shown in Equation **??**.

$$x_0 = \frac{x_t - \sqrt{1 - \gamma_t}\epsilon_t}{\sqrt{\gamma_t}} \tag{5.6}$$

Beyond this point, we already established in Equation **??** that the posterior mean μ_θ can be computed directly from x_0^θ. In result , if we can predict ϵ_θ we can compute x_0^θ using Equation **??** and subsequently evaluate μ_θ as before. It establishes a clear computational chain where each transformation is analytically determined as shown in Figure **??**.

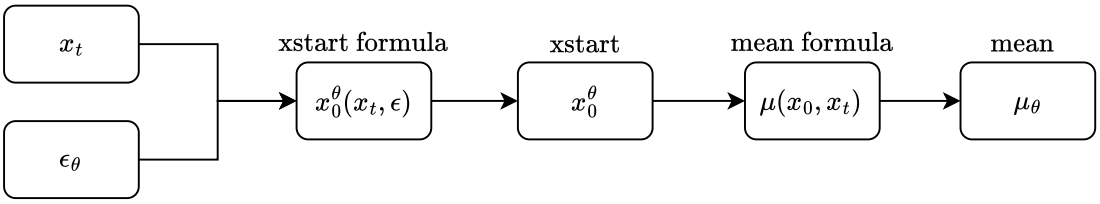


Figure 5.9: Neural network parameterization with XStart Mean strategy.

Predicting ϵ_θ offers a compelling alternative because the network targets the precise quantity that was added to x_0 rather than learning to reconstruct original image. This chain of analytical computations eliminates the need for the network to learn complex inverse transformations, focusing solely on the fundamental noise estimation task.

In order to verify the performance Epsilon Mean Strategy, we conduct Experiment 7 using the fixed variance and L_{VLB} loss. See the training characteristics in Figure **??**.

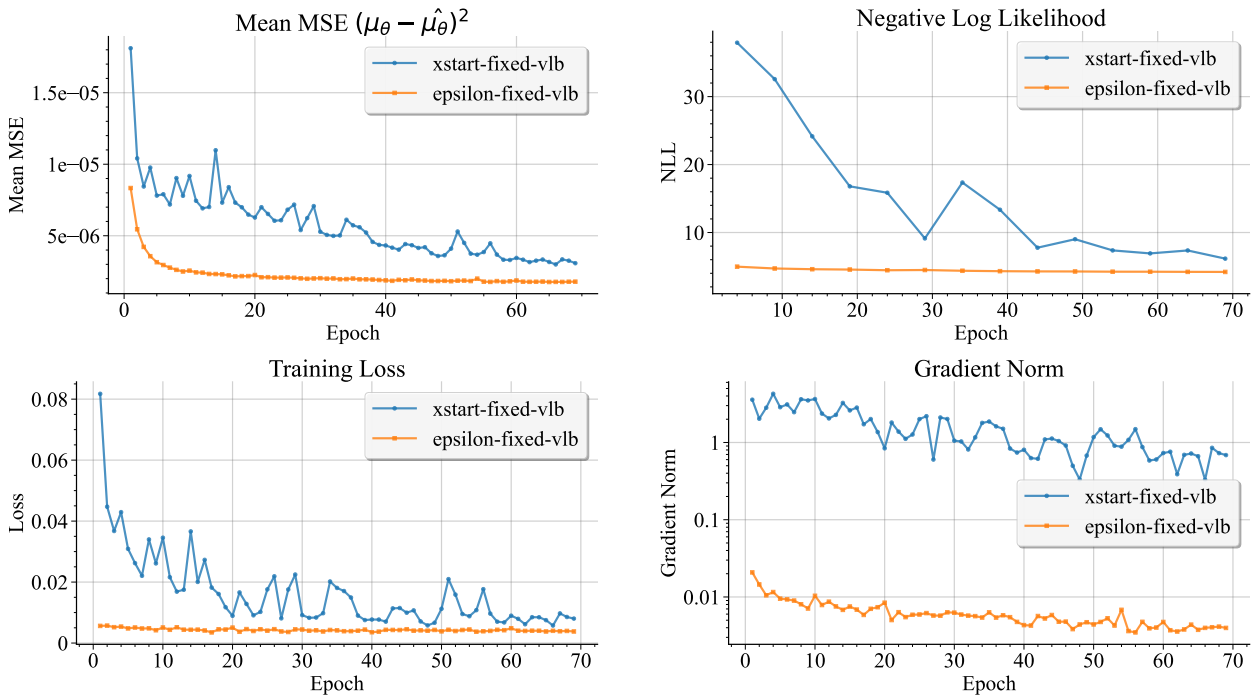
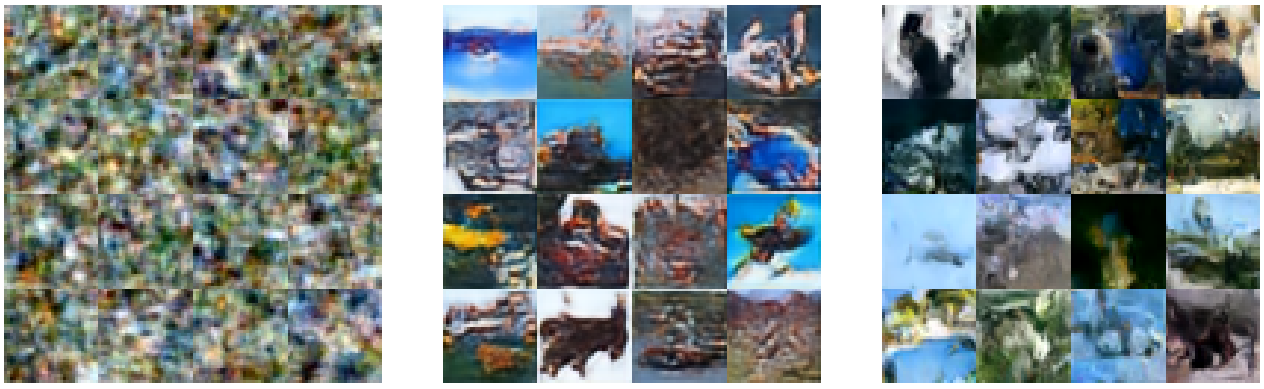


Figure 5.10: Training characteristics of xstart-fixed-mse parameterization.

We obtain quite optimistic training characteristics. Mean estimation accuracy improved further surpassing the XStart strategy. The NLL also showed marked improvement, being lower than 10 from early epochs indicating better overall model fit. The training loss curve remains stable throughout the run, indicating well behaved optimization dynamics. In order to confirm the qualitative impact of the Epsilon Mean Strategy, we present generated samples in Figure **??** alongside samples from earlier experiments for direct comparison.



(a) Experiment 5 (b) Experiment 6 (c) Experiment 7

Figure 5.11: Generated samples comparison between Experiment 5, 6 and 7.

Certain shapes are clearly visible on the one hand, and the characteristic artefacts have disappeared in comparison with Experiment 6. Nevertheless, these are not yet clear objects belonging to known categories. It will come as no surprise that in the next experiment we

will address the problematic scaling in terms of loss, which we have already experienced in previous iterations.

5.5 Experiment 8: Simpler Epsilon Mean Strategy

For the sake of formality, we demonstrate in Equation ?? what form L_t takes in the case of the Epsilon Strategy and how the problematic scaling can be removed. We use C_{xstart} constant in L_t from Equation ?? in order to simplify notation. We substitute x_0 and x_0^θ from Equation ?? into the MSE loss expression.

$$\begin{aligned} L_t &= C_{xstart}\left(x_0 - x_0^\theta\right)^2 = C_{xstart}\left(\frac{x_t - \sqrt{1-\gamma_t}\epsilon}{\sqrt{\gamma_t}} - \frac{x_t - \sqrt{1-\gamma_t}\epsilon_\theta}{\sqrt{\gamma_t}}\right)^2 \\ &= C_{xstart}\left(\frac{\sqrt{1-\gamma_t}}{\sqrt{\gamma_t}}(\epsilon - \epsilon_\theta)\right)^2 = C_{xstart}\frac{1-\gamma_t}{\gamma_t}(\epsilon - \epsilon_\theta)^2 \end{aligned} \tag{5.7}$$

We finally eliminate the weighting to obtain the unweighted MSE loss between true and predicted noise as shown in Equation ??.

$$L_t = (\epsilon_t - \epsilon_\theta)^2 \tag{5.8}$$

We evaluate the Epsilon Mean Strategy maintaining the fixed variance setting while modifying only the prediction target and loss function. See the training characteristics in Figure ??.

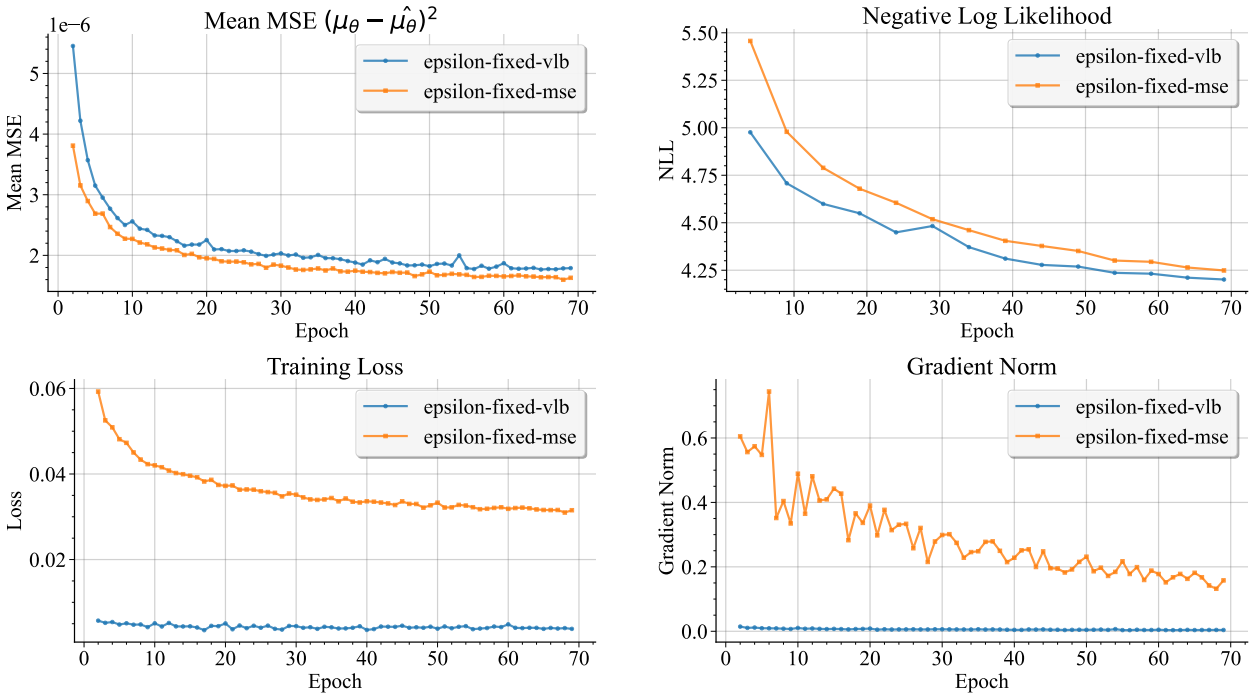
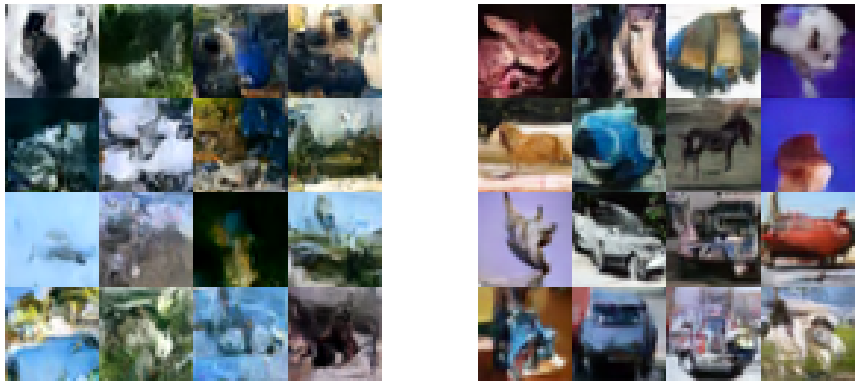


Figure 5.12: Training characteristics of epsilon-fixed-mse parameterization.

We can see another improvement in the precision of mean estimation, which was our main goal. Although we achieved a slightly weaker result for NLL, this is because the training objective in this experiment is different. NLL reflects the full VLB, and we will return to this in the future. Therefore, the structural quality of the generated images should be evaluated. We present a comparison in Figure **??** between samples from Experiment 7 and Experiment 8.



(a) Experiment 7 (b) Experiment 8

Figure 5.13: Generated samples comparison between Experiment 7 and 8

We can now observe and distinguish certain objects and unambiguous features of categories such as cars, horses and aeroplanes. While not all results are clear, we avoided cherry-picking the best ones. In addition, as we mentioned at the very beginning, the training duration was significantly shorter than in the case of SOTA.

5.6 Experiment 9: Epsilon Strategy - Longer

So far, we have stuck to fairly short training sessions of 70 epochs, which has allowed us to effectively evaluate different configurations and approaches. Epsilon Strategy clearly stands out from the other configurations, so to gain a better understanding, we conducted additional training iterations of up to 200 epochs to examine the potential and quality of the generated images. The results are presented in Figure **??**.

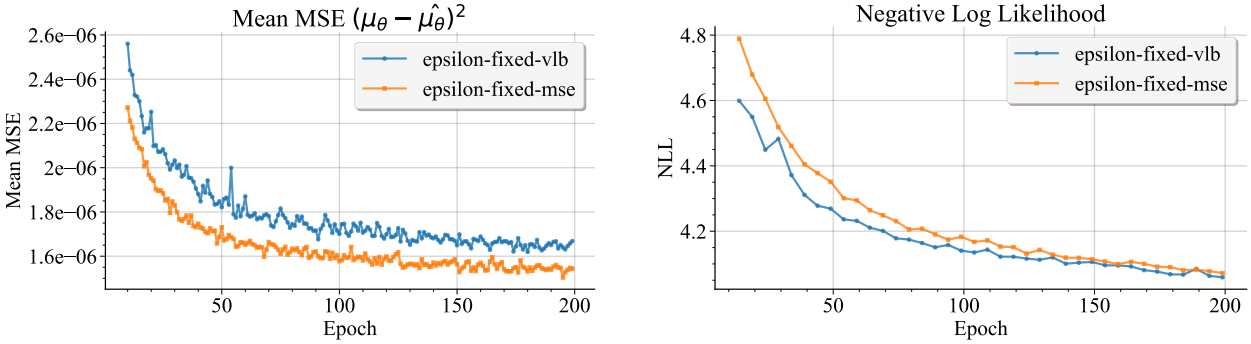
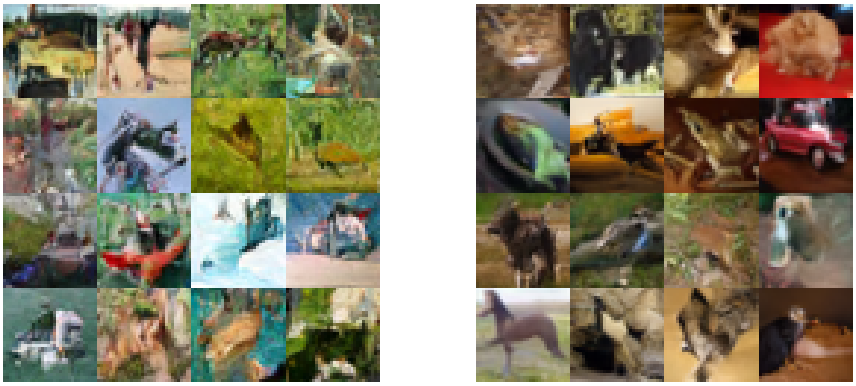


Figure 5.14: Long training of epsilon-fixed-mse parameterization.

It is clear that longer training would have achieved better results in both metrics considered. Figure **??** presents a comparison of generated samples between Experiment 7 and Experiment 8 after extended training.



(a) Experiment 7 (b) Experiment 8

Figure 5.15: Generated samples comparison between Experiment 7 and 8

Using full VLB improves the quality, sometimes enabling specific objects to be recognised, but specific artefacts and imperfections can be seen in the images. However, the best quality results so far were obtained using the simplified criterion, with categories such as car, bird, horse, frog and quadruped being recognisable.

To summarize collected metrics across all experiments, we present a comparison table in Table **??**.

Run	Mean	Variance	Loss	Mean MSE	NLL
1	Direct	Direct	VLB	0.08	3425
2	Direct	Direct Log	VLB	2.36e-5	8.35
3	Direct	Fixed	VLB	4.54e-5	10.66
4	Direct	Fixed	MSE	1.92e-5	8.07
5	XStart	Fixed	VLB	2.99e-6	6.16
6	XStart	Fixed	MSE	3.66e-6	20.81
7	Epsilon	Fixed	VLB	1.62e-6	4.06
8	Epsilon	Fixed	MSE	1.50e-6	4.07

Table 5.1: Summary of experiments with different mean strategies and loss functions.

Small summary about the table.
Summary to the chapter?

Chapter 6

Enhanced Variance Parameterization

In previous chapters, we developed a robust approach to mean estimation by abandoning the original VLB framework and using unweighted MSE objectives. This simplification yielded significant benefits in terms of training stability and performance. However, this approach may not fully capture the underlying probabilistic structure of the generative process. Reintroducing a controlled VLB component could encourage better alignment with the theoretical foundation, all the while maintaining optimisation stability.

6.1 Hybrid Loss Function

One of the problems with using fixed variance is that it is only the optimal choice when the mean is optimal too, which is not always obvious. Trained variance could serve as a natural regulariser. One interesting proposal to achieve this was to use separate learning for the mean and the variance, which would define this approach as *hybrid* as shown in Equation **??**.

$$L_{\text{hybrid}} = L_{\text{MSE}} + \lambda L_{\text{VLB}} \tag{6.1}$$

The first term maintains direct supervision for mean estimation, while the second term enables variance training through the VLB objective. In order to make variance training as an addition it is controlled by hyperparameter that balances the two objectives. Empirical investigation [**nichol2021improved**] reveals that small values of $\lambda = 0.001$ effectively balance these objectives without destabilizing training.

Gradient Isolation Strategy

The hybrid formulation introduces a subtle but important implementation issue: both loss terms depend on the predicted mean, and their gradients can therefore become inadvertently coupled. Without explicit isolation, the parameters influencing μ_θ would receive updates from both L_{MSE} and L_{VLB}, which can lead to gradient interference and obscure the intended roles of the two objectives. We address this by applying a stop-gradient operation between the components, ensuring that each term contributes only to the aspect it is designed to optimize.

Concretely, after the model produces its predictions, we first compute L_{MSE} and then detach the mean from the computational graph so that this term is treated as a constant in subsequent computations. We then evaluate L_{VLB} using the detached values, preventing gradients from L_{VLB} from propagating through the pathways already optimized by L_{MSE}. Figure **??** illustrates this isolation step schematically.

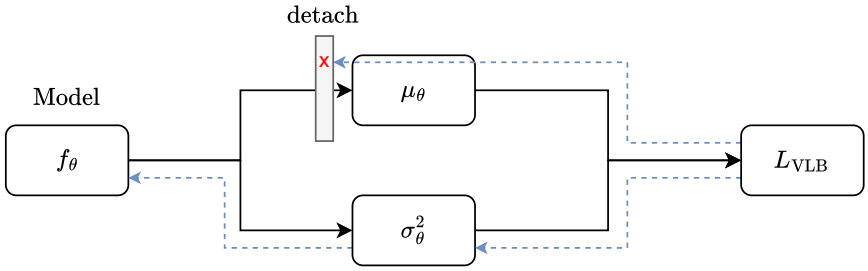


Figure 6.1: Gradient flow isolation in hybrid training: variance learning uses detached mean estimates

In order to evaluate the effectiveness of the hybrid loss with gradient isolation, we conduct experiments comparing this approach against two earlier experiments with both L_{VLB} and L_{MSE} objectives with epsilon parameteizaion. See the results in Figure **??** and Table **??**.

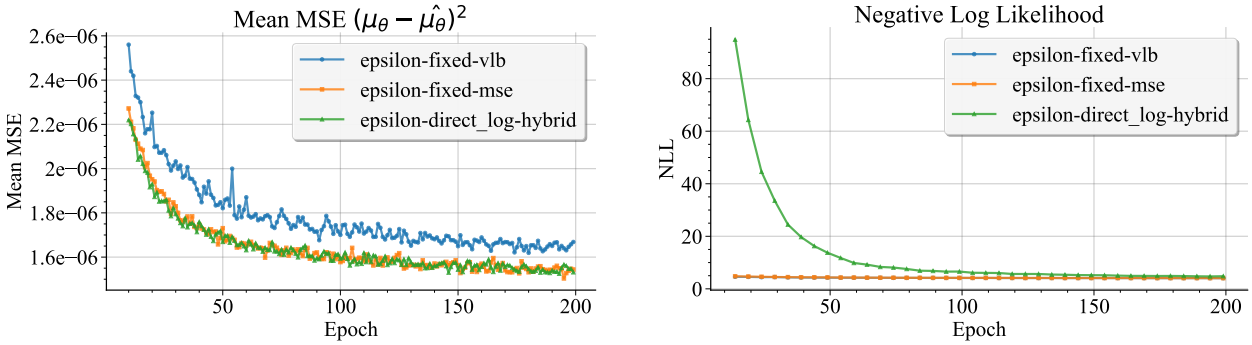
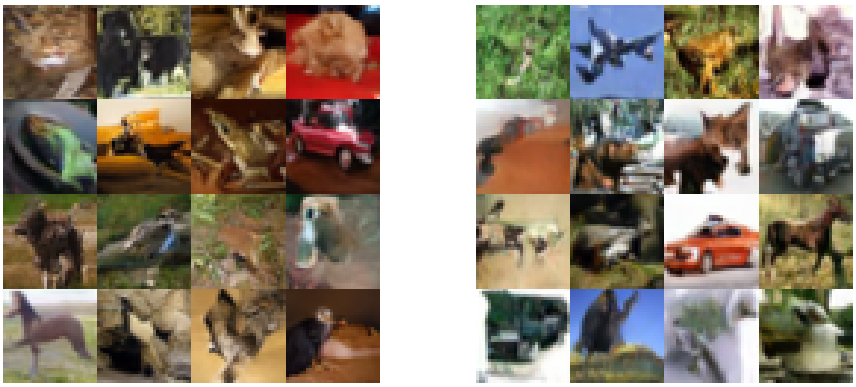


Figure 6.2: Training characteristics of hybrid loss with gradient isolation.

Run	Mean	Variance	Loss	Mean MSE	NLL
7	Epsilon	Fixed	VLB	1.62e-6	4.06
8	Epsilon	Fixed	MSE	1.50e-6	4.07
9	Epsilon	Direct Log	Hybrid	1.53e-6	4.71

Table 6.1: Summary of experiments with different mean strategies and loss functions.

The model was successfully trained and achieved comparable results. Note the steady improvement in NLL. Ultimately, the final value is slightly worse, but it seems there was potential for further convergence with a longer training period.



(a) Experiment 8 (b) Experiment 9

Figure 6.3: Generated samples comparison between Experiment 8 and 9

The generated images in Figure **??** confirm the high quality of the hybrid approach and demonstrate its effectiveness. However, pushing the variance toward a useful calibration for reasonable NLL is noticeably slower and more delicate to optimize, which motivated quite interesting idea with the trainable-range variance parameterization.

6.2 Trainable Range Variance Parameterization

In the DDPM work [**ho2020denoising**] an alternative fixed variance parameterization alongside the theoretical posterior variance was considered. They proposed using variance **??** from the forward process instead of derived posterior variance.

$$\sigma_t^2 = 1 - \alpha_t \tag{6.2}$$

The motivation is straightforward: since the forward process injects noise with variance σ_t^2, it is natural to ask whether the reverse process should use the same variance scale. However, empirical results did not show a meaningful performance difference between these two variance choices giving roughly similar results. The reason is that both options are in fact very similar, different only during the early timesteps. In Figure **??** we show the ratios of both variance choices across timesteps.

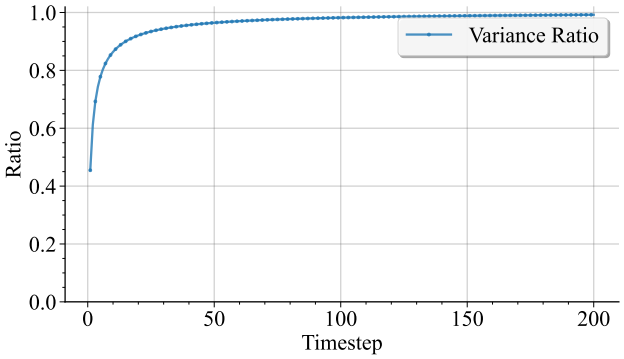


Figure 6.4: Ratio between variance bounds across timesteps

Notable difference between variance options exist during the early timesteps, while for timesteps with $t > 100$, the ratio converges to near 1. Based on this, the interesting idea was proposed of parameterising variance so that the model estimates the interpolation v transforming variance estimation into a bounded interpolation task as shown in Equation **??**.

$$\log \sigma_\theta^2 = v \log \sigma_{\text{large}}^2 + (1 - v) \log \sigma_{\text{small}}^2 \tag{6.3}$$

In the original implementation, the assumed output range for the ratio is $v \in (-1, 1)$, centered around zero. However, Equation **??** requires v to lie within the range $(0, 1)$. To achieve this, we apply a simple shift and scaling operation to map the model's output into the desired range as shown in Equation **??**.

$$v = \frac{\text{out} + 1}{2} \tag{6.4}$$

where out represents the raw network output in the range $(-1, 1)$. The resulting parameterization provides a principled method for learning optimal variance structures that respect both theoretical bounds while enabling adaptive uncertainty modeling tailored to the specific denoising context and dataset characteristics.

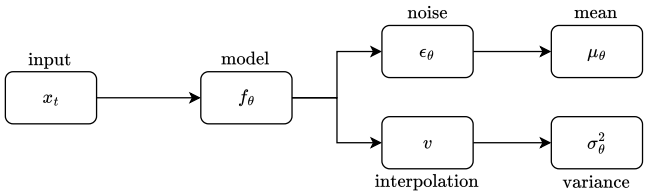


Figure 6.5: Interpolated variance architecture for hybrid loss training.

We evaluate the hybrid loss function with direct variance estimation and range-based variance estimation, comparing the results with the Epsilon Mean Simple configuration.

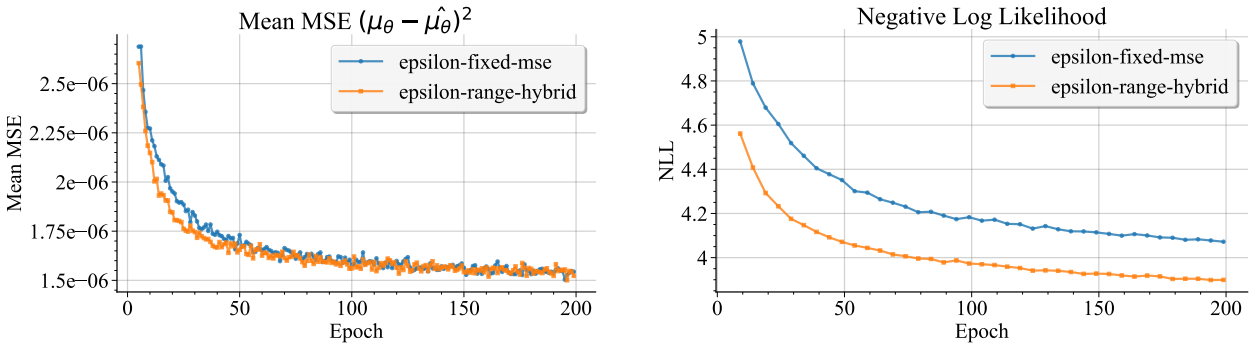


Figure 6.6: Training characteristics of hybrid loss with range-based variance estimation.

In a sense, another step forward has been taken, as the mean is reaching a lower value, albeit slightly, but faster. This may indicate the aspect we have already mentioned: that fixed variance is ideal only when the mean is optimal. This is not true, especially at the

beginning of training. In this case, trained variance can act as a regulariser. Additionally, we have achieved the best NLL result to date, which confirms the earlier hypothesis thanks to the additional term in L_{hybrid}, the objective is closer to the original L_{VLB}.

Before presenting the images generated from the previous experiment, we will extend the experiment to explore the limits of this method, as this will be the last configuration in this work. Training was carried out for 1000 epochs, and the results can be found in Figure **??**, Table **??** and Figure **??**.

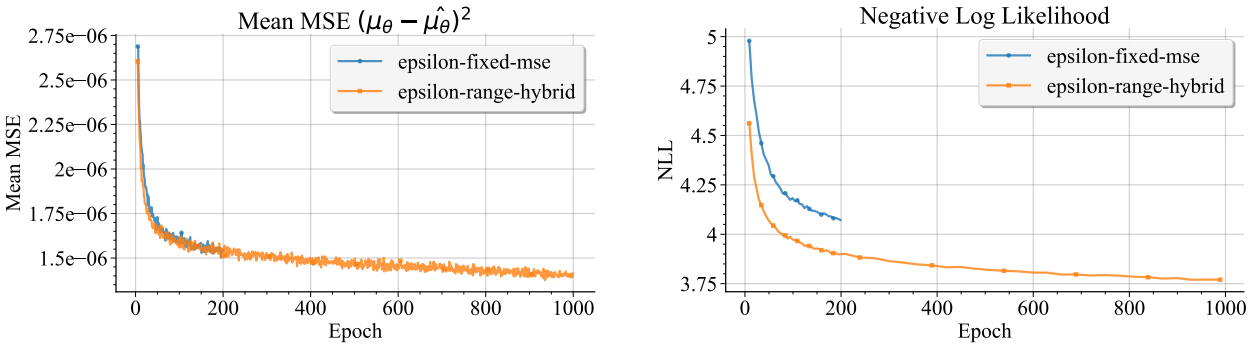


Figure 6.7: Training characteristics of hybrid loss with range-based variance estimation.

Run	Mean	Variance	Loss	Mean MSE	NLL
7	Epsilon	Fixed	VLB	1.62e-6	4.06
8	Epsilon	Fixed	MSE	1.50e-6	4.07
9	Epsilon	Direct Log	Hybrid	1.53e-6	4.71

Table 6.2: Summary of experiments with different mean strategies and loss functions.

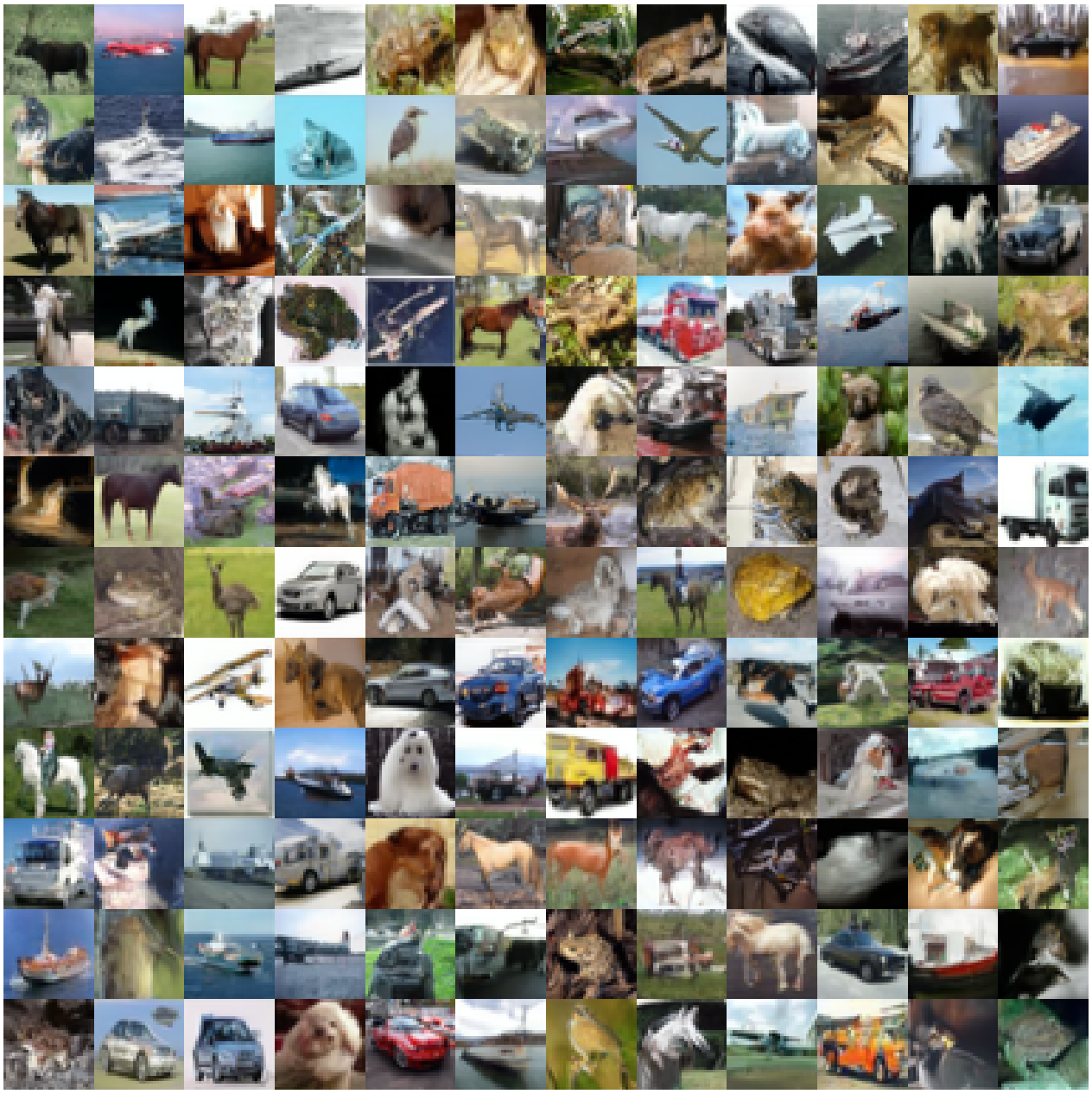


Figure 6.8: Generated samples from the final hybrid loss experiment after 1000 epochs.

The improvements, however, remain subtle on the CIFAR dataset, which has relatively limited visual and structural complexity. It is expected that the advantages of the Hybrid Loss formulation would become more pronounced on higher-resolution datasets like LSUN or ImageNet.