Generative Modeling with Diffusion

Justin Le[†]

Project advisors: Sebastien Motsch[‡] and Johannes Brust[§]

Abstract. We provide an overview of the diffusion model as a method to generate new samples. Generative models have been recently adopted for tasks such as art generation (Stable Diffusion, Dall-E) and text generation (ChatGPT). Diffusion models in particular apply noise to sample data and then "reverse" this noising process to generate new samples. We will formally define these noising and denoising processes, then present algorithms to train and generate with a diffusion model. Afterward, we will explore a potential application of diffusion models in improving classifier performance on imbalanced data.

1. Introduction.

Generative models create new data instances. Recently, generative models have seen applications in image generation [15] and text generation [14]. In this article, we are interested in generative models that create data instances that "mimic" some given sample data. Doing so will allow us to sample arbitrarily many points from the distribution underlying the original sample data. Figure 1 depicts this process applied to image generation. In a broad sense, the model receives a sample of training images, which "learns" the structure of the images, then uses that knowledge to create new images that mimic the original sample. Common architectures used for generative models include Generative Adversarial Networks [6, 19], Variational Autoencoders [10], and Transformers [18, 9]. For this article, we will focus on discussing the mechanics of diffusion models [7, 20, 16]. In particular, we seek to give a formal overview of how diffusion models are trained and how they generate new samples.

In Section 4, we explore an application of diffusion models for improving classifier performance (this is in contrast to the typical application of diffusion models as image generators [7]). To this end, we will focus on a dataset of credit card transactions, of which an exceedingly small minority are fraudulent, and the rest are legitimate. Our aim is to use diffusion models to generate data mimicking the fraudulent transactions, then augmenting the training data with this generated data before training a classifier.

1.1. Diffusion Models.

Diffusion models iteratively apply noise to data until the initial distribution of the data is unrecognizable, and then, after learning this "noising phase", reverse the noising to recover the initial sample. We will refer to this noising phase as the *forward process* and this denoising phase the *reverse process*. More specifically, the forward process will transform the sample data into a standard normal distribution (that is, normal with mean 0 and unit variance). Then, to generate new data, we sample *new* points from a standard normal distribution and apply the reverse process to recover an approximation of the sample data. This scheme will allow us to generate arbitrarily many points that mimic the initial sample.

[†]Arizona State University, Tempe, AZ (jdle4@asu.edu).
[‡]Arizona State University, Tempe, AZ (smotsch@asu.edu).
[§]Arizona State University, Tempe, AZ (jjbrust@asu.edu).

In addition to converging to a standard normal, we would also like the noising process to be continuous, converge quickly, and have randomness enforced at each time step. The randomness in the noising process is key, as this is what allows the generated data to be slightly different from the original sample when we denoise the data. To formally define the noising process, we will present the Ornstein-Uhlenbeck equation in Section 2, which is a type of Stochastic Differential Equation (SDE). The solution to this equation will satisfy the above properties and will therefore be used to define the forward process. After defining the forward process, we can then deduce the reverse process and use this to generate samples.

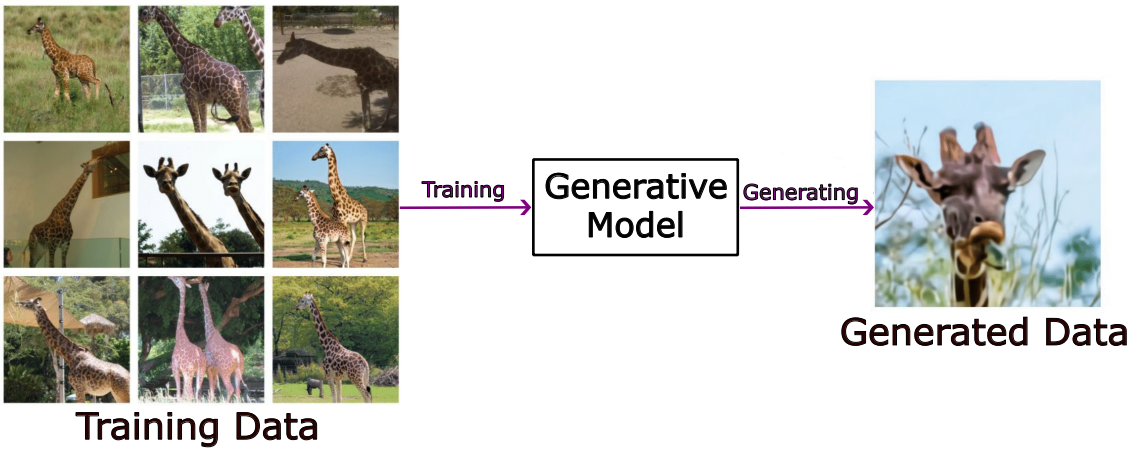


Figure 1: A sketch of the pipeline for image generation. On the left are real images of giraffes, and on the right is an image of a giraffe generated by Dall-E (found in [15]).

2. Theory of Diffusion Models.

Our first step towards developing diffusion models is to formally define the forward process. We achieve this with the following SDE.

2.1. The Ornstein-Uhlenbeck Equation.

We consider the Ornstein-Uhlenbeck equation in $\mathbb{R}^d$ given by

$$\begin{cases} \mathrm{d}\mathbf{X}_t &= -\mathbf{X}_t\,\mathrm{d}t + \sqrt{2}\,\mathrm{d}\mathbf{B}_t \\ \mathbf{X}(0) &= \mathbf{X}_0 \end{cases}. \tag{2.1}$$

In contrast to an ODE, whose solution is a function from $[0,\infty)$ to $\mathbb{R}^d$, the solution $\mathbf{X}_t$ to (2.1) is a stochastic process, which can be thought of as a function from $[0,\infty)$ into the space of random variables on $\mathbb{R}^d$. Likewise, the initial condition $\mathbf{X}_0$ is our original sample rather than a point in $\mathbb{R}^d$. The overarching goal of a diffusion model is to generate samples from the underlying distribution of $\mathbf{X}_0$.

The term $-\mathbf{X}_t\mathrm{d}t$ is known as the *drift* and is the non-stochastic element of the SDE. In-

tuitively, this drift term contributes exponential decay, as it is known that the ODE

$$\begin{cases} \mathrm{d}\mathbf{X}_t &= -\mathbf{X}_t \, \mathrm{d}t \\ \mathbf{X}(0) &= x_0 \end{cases}$$

has as its solution

$$\mathbf{X}_t = x_0 e^{-t}.$$

The stochastic process $\mathbf{B}_t \sim \mathcal{N}(\mathbf{0}, t\mathbf{I})$ is responsible for enforcing randomness in this SDE. We call $\mathbf{B}_t$ a *Wiener process* (or *Brownian motion*). A Wiener process can be thought of as modeling particles dispersing in space over time.

We begin by stating the well-known solution to the Ornstein-Uhlenbeck equation.

Proposition 2.1. *The solution to* (2.1) *is given by*

$$\mathbf{X}_t = e^{-t}\mathbf{X}_0 + +\sqrt{2}\int_0^t e^{-(t-s)} \, \mathrm{d}\mathbf{B}_s, \tag{2.2}$$

and the distribution of $\mathbf{X}_t$ *satisfies*

$$\mathbf{X}_t = e^{-t}\mathbf{X}_0 + \sqrt{1 - e^{-2t}}\,\mathbf{Z}, \tag{2.3}$$

where $\mathbf{Z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

This solution $\mathbf{X}_t$ is the forward process described in Subsection 1.1. See Appendix A and [5] for more information of how this solution is obtained.

To condense our notation, we will denote

$$\gamma_t = \gamma(t) := e^{-t} \tag{2.4}$$

$$\beta_t = \beta(t) := \sqrt{1 - e^{-2t}} \tag{2.5}$$

for all $t > 0$. Then, (2.3) becomes

$$\mathbf{X}_t = \gamma_t \mathbf{X}_0 + \beta_t \mathbf{Z}. \tag{2.6}$$

In particular, $\mathbf{X}_t \sim \mathcal{N}(\gamma_t \mathbf{X}_0, \beta_t^2 \mathbf{I})$ for all $t \in \mathbb{R}$. Since $\lim_{t\to\infty} \gamma_t = 0$ and $\lim_{t\to\infty} \beta_t = 1$, we have that $\mathbf{X}_t$ converges in distribution to a standard normal as $t \to \infty$. Moreover, γ_t and β_t are continuous and converge exponentially to their respective limits, meaning the forward process has the desired properties of randomness, rapid convergence to a standard normal, and continuity.

2.2. Discretizing the Time Domain. We now consider a discretization of the time domain to simulate solutions to (2.1). Fixing $N \in \mathbb{N}$, let $\{\Delta t_n\}_{n=1}^N$ be a finite sequence of positive real numbers (we discuss a more detailed scheme to define the Δt_n in Appendix B). Then, define $t_0 = 0$ and $t_n = \sum_{k=1}^{n} \Delta t_k$ for $n = 1, \dots, N$. In other words, the t_n represent our discrete time values while the Δt_n describe the steps in time. We denote

$$\mathbf{X}_n = \mathbf{X}_{t_n}, \quad \gamma_n = \gamma_{t_n}, \quad \beta_n = \beta_{t_n}.$$

Using this discretization, we immediately obtain a recursive formula for computing $\mathbf{X}_n$.

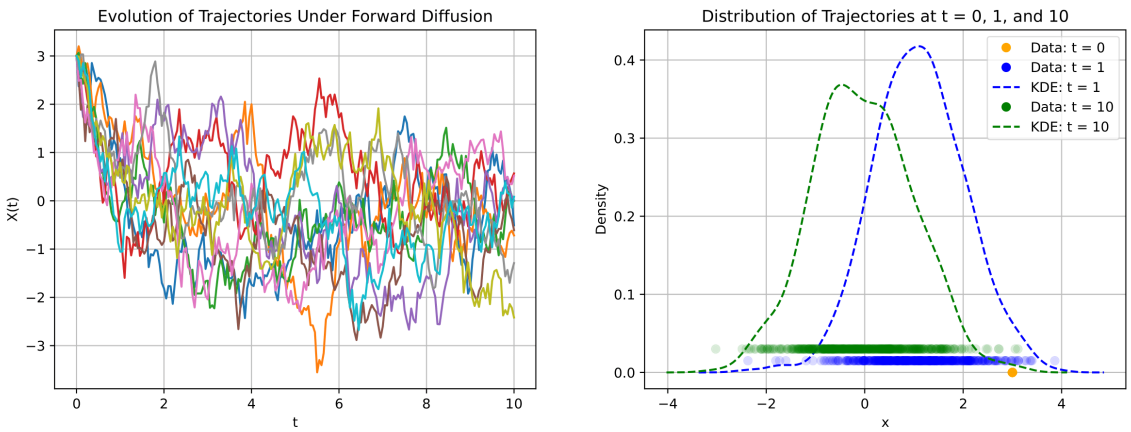
Corollary 2.2. *For all $n = 0, \dots, N-1$, the distribution of $\mathbf{X}_{n+1}$ satisfies*

$$\mathbf{X}_{n+1} = \gamma(\Delta t_{n+1})\mathbf{X}_n + \beta(\Delta t_{n+1})\mathbf{Z}_{n+1}, \tag{2.7}$$

where $\mathbf{Z}_1, \dots, \mathbf{Z}_N \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ are i.i.d. standard normal random variables.

This result is obtained in the same fashion as (2.6) – see Appendix A for more details.

To simulate the forward process on sample data, we take $x_0 \in \mathbb{R}^d$ and iteratively apply (2.7) with x_0 in place of $\mathbf{X}_0$ and $\varepsilon_n \in \mathbb{R}^d$ sampled from a standard normal distribution in place of $\mathbf{Z}_n$. Note that the ε_n are sampled independently in each iteration. Figure 2 shows the trajectories and kernel density estimate (KDE) under (2.7) with $\mathbf{X}_0$ being sampled from a Dirac mass centered at 3. Importantly, despite the fact that all 10 trajectories begin at the same position, each of the trajectories split off into different directions because of the randomness enforced in (2.7). Moreover, as t increases, the distribution of $\mathbf{X}_t$ begins to resemble that of a standard normal.



(a) 10 forward trajectories

(b) KDE (500 trajectories)

Figure 2: Trajectories and KDE with $\mathbf{X}_0 = 3$ under the forward process.

2.3. Reverse Diffusion. We have seen that the forward diffusion process takes any initial distribution and transforms it into a standard normal. We now ask whether or not the forward diffusion process can be reversed. That is, starting from data sampled from a standard normal distribution, we want to know if the forward diffusion process can be applied in reverse to obtain data that mimics the original sample. In Figure 3, we demonstrate these forward and reverse processes applied to a simple dataset in $\mathbb{R}^2$. Once again, the forward process begins with the original sample data and iteratively applies noise. To generate a new sample with the reverse process, we begin by sampling *new* points from a standard normal and iteratively denoising the data. The resulting "denoised" data resembles the original sample, but is not exactly the same. Like with the forward process, the reverse process does obey an SDE (this is explored in [17]). In this paper, we are not interested in this reverse SDE – instead, we will deduce the reverse process by leveraging the discrete time steps we discussed in Subsection 2.2. Doing so will give us a discretized version of the reverse process.

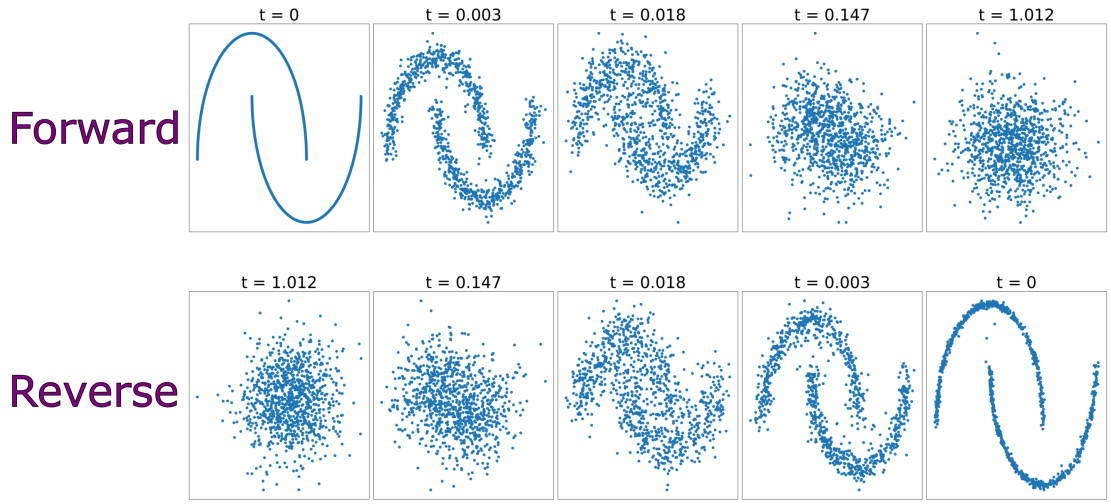


Figure 3: A timeline of the forward process (top) and the reverse process (bottom) applied to the Moons dataset from Scikit-Learn [13]. The original sample is in the top-left corner and the generated data is in the bottom-right corner.

Recall that the discretized forward diffusion process gives a finite sequence of random variables $\mathbf{X}_0, \mathbf{X}_1, \ldots, \mathbf{X}_N$. We will assume N and t_N are chosen such that $\mathbf{X}_N$ is approximately standard normal for the given initial condition $\mathbf{X}_0$. Ideally, we would want to determine the conditional density $\rho(x_n \,|x_{n+1})$ (that is, given the future position of a trajectory, we want to determine the distribution of its present position). However, deducing this conditional density is an ill-posed problem. This is because *every* initial condition converges to a standard normal under the forward diffusion process. Therefore, if we start from $\mathbf{X}_N \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and have no information on the initial condition $\mathbf{X}_0$, the conditional density $\rho(x_{N-1} \,|\, x_N)$ is not unique.

Instead, we will look to deduce $\rho(x_n \,|\, x_{n+1}, x_0)$ – that is, given the future *and initial* po-

sition of a trajectory, we want to determine the distribution of its present position. Starting from $\mathbf{X}_N$ and iteratively computing the conditional density of the previous time step will result in an approximation of $\rho(x_0)$. The following proposition addresses how this conditional density is obtained:

Proposition 2.3. *Let $1 \leq n \leq N-1$ and suppose $\mathbf{X}_0 = x_0$ and $\mathbf{X}_{n+1} = x_{n+1}$. Then,*

$$\rho(x_n \,|\, x_{n+1}, x_0) \sim \mathcal{N}(\overline{\mu}, \overline{\sigma}_n^2), \tag{2.8}$$

with $\overline{\mu} = \overline{\mu}(x_{n+1}, x_0, t_{n+1})$ and $\overline{\sigma}_n^2$ defined as

$$\overline{\mu} = \frac{\gamma(\Delta t_{n+1})\beta_n^2}{\beta_{n+1}^2} \cdot x_{n+1} + \frac{\gamma_n \beta (\Delta t_{n+1})^2}{\beta_{n+1}^2} \cdot x_0 \tag{2.9}$$

$$\overline{\sigma}_n^2 = \frac{\beta(\Delta t_{n+1})^2 \beta_n^2}{\beta_{n+1}^2}. \tag{2.10}$$

See Appendix C for a detailed proof of this result.

With this proposition, we can determine the distribution of $\mathbf{X}_n$ if we know the values taken on by $\mathbf{X}_0$ and $\mathbf{X}_{n+1}$. In fact, this proposition tells us that the discretized reverse diffusion process is nothing more than computing a mean and variance, then sampling from a normal distribution – Figure 4 shows a sketch of this procedure. This result can also be extended to $n = 0$. In particular, if $n = 0$, then (2.9) and (2.10) yield $\overline{\mu} = x_0$ and $\overline{\sigma}_n^2 = 0$. This is consistent with the fact that the conditional distribution of $\mathbf{X}_0$ given $\mathbf{X}_0 = x_0$ is x_0 with probability 1. What's more, the discretized reverse diffusion will tend towards x_0 as n tends to 0, confirming that this reverse diffusion process does indeed "reverse" the forward process outlined in Subsection 2.1.

Importantly, the variance term $\overline{\sigma}_n^2$ only depends on the iteration n (i.e., there is no dependence on the position of the trajectory). Because we control these iterations, the variance of the discretized reverse diffusion process is always known. On the other hand, the mean $\overline{\mu}$ depends on both x_{n+1} and x_0. In other words, $\overline{\mu}$ is the only quantity of interest in the reverse diffusion process that encodes information from the distribution of $\mathbf{X}_0$.

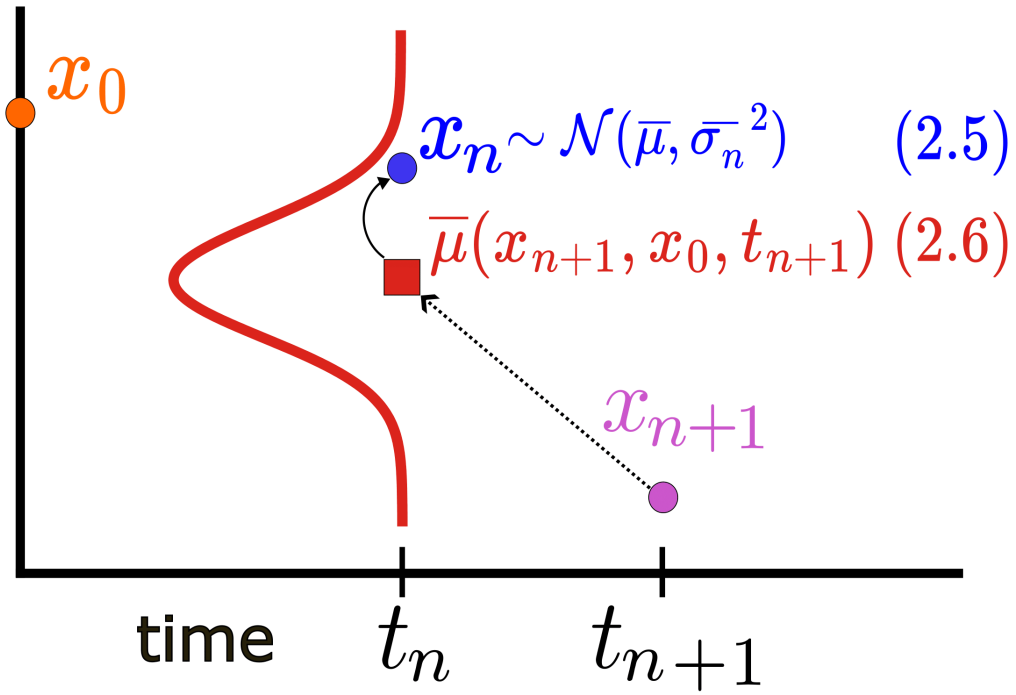


Figure 4: A sketch of the discretized reverse diffusion process with the initial condition x_0 known. Starting at x_{n+1}, we compute $\overline{\mu}$ with (2.9), then compute $\overline{\sigma}_n^2$ with (2.10), and finally sample $x_n \sim \mathcal{N}(\overline{\mu}, \overline{\sigma}_n^2)$ as in (2.8).

3. Building a Diffusion Model. When generating data with the reverse diffusion process, we must begin by sampling points from a standard normal. As a result, when applying the reverse diffusion process to these points, the initial condition x_0 (i.e., the value taken on by $\mathbf{X}_0$) is not known, meaning $\overline{\mu}$ cannot be directly computed through (2.9). To circumvent this, we will construct a scheme to estimate $\overline{\mu}$ while only knowing x_{n+1} and t_{n+1}. For this, we employ techniques from machine learning to "learn" the value of $\overline{\mu}$ from the original sample. Letting θ be the set of parameters of our model, we will let $\hat{\mu}_\theta = \hat{\mu}_\theta(x_{n+1}, t_{n+1})$ denote the expectation of $\mathbf{X}_n$ predicted by our model and $\overline{\mu}(x_{n+1}, x_0, t_{n+1})$ denote the true expectation of $\mathbf{X}_n$ given $\mathbf{X}_0 = x_0$ and $\mathbf{X}_{n+1} = x_{n+1}$. A brief outline for training this model is as follows:

1. Apply the forward diffusion process to x_0 for a finite number of steps.
2. For each step, compute $\overline{\mu}(x_{n+1}, x_0, t_{n+1})$ with (2.9).
3. Use the model to predict $\hat{\mu}_\theta(x_{n+1}, t_{n+1})$ for each step.
4. Update θ to minimize the distance $\|\overline{\mu} - \hat{\mu}_\theta\|$.

With this scheme, the information from $\mathbf{X}_0$ gets encoded into the parameter set θ. In other words, our model learns the distribution underlying the original sample.

3.1. Alternative Methods to Approximate $\overline{\mu}$. Since our model will know the values of x_{n+1} and t_{n+1}, the only unknown quantity in the expression for $\overline{\mu}$ is x_0. Thus, instead of predicting $\overline{\mu}$ directly with our model, we may instead opt to predict x_0 with our model and then use this prediction to compute an estimate of $\overline{\mu}$. This way, we avoid using the model to predict all the terms which are already known (namely, the terms dependent on t_{n+1} and x_{n+1}).

We also have another method for estimating $\overline{\mu}$. Using (2.6), we write

$$x_0 = \frac{x_{n+1} - \beta_{n+1}\varepsilon_0}{\gamma_{n+1}}, \tag{3.1}$$

where ε_0 is sampled from a standard normal distribution. Substituting this into (2.9) and simplifying will yield

$$\overline{\mu} = \frac{1}{\gamma(\Delta t_{n+1})}\left(x_{n+1} - \frac{\beta(\Delta t_{n+1})^2}{\beta_{n+1}}\varepsilon_0\right). \tag{3.2}$$

Intuitively, one can think of ε_0 as the "noise" that transforms x_0 into x_{n+1}. By predicting this noise term ε_0, we can estimate the value of $\overline{\mu}$. Using (2.9) and (3.2), we now define two alternative methods for determining $\hat{\mu}_\theta$:

$$\hat{\mu}_\theta = \hat{\mu}_\theta(x_{n+1}, t_{n+1}) = \frac{\gamma(\Delta t_{n+1})\beta_n^2}{\beta_{n+1}^2} \cdot x_{n+1} + \frac{\gamma_n \beta(\Delta t_{n+1})^2}{\beta_{n+1}^2} \cdot (\hat{x_0})_\theta \tag{3.3}$$

$$\hat{\mu}_\theta = \hat{\mu}_\theta(x_{n+1}, t_{n+1}) = \frac{1}{\gamma(\Delta t_{n+1})}\left(x_{n+1} - \frac{\beta(\Delta t_{n+1})^2}{\beta_{n+1}}\varepsilon_\theta\right), \tag{3.4}$$

where $(\hat{x_0})_\theta = (\hat{x_0})_\theta(x_{n+1}, t_{n+1})$ and $\varepsilon_\theta = \varepsilon_\theta(x_{n+1}, t_{n+1})$ are (respectively) the estimates of x_0 and ε_0 produced by our model. To summarize, we have three methods to estimate $\overline{\mu}$:

1. Use the model to estimate $\hat{\mu}_\theta(x_{n+1}, t_{n+1})$.
2. Use the model to estimate $(\hat{x_0})_\theta$ and compute $\hat{\mu}_\theta$ with (3.3).
3. Use the model to estimate ε_θ and compute $\hat{\mu}_\theta$ with (3.4).

Note that methods 2 and 3 can be thought of as equivalent up to a change of variables. However, recall that the unconditional distribution of the ε_0 (i.e., when the initial condition is not known) is standard normal. Thus, the distribution of the ε_0 is potentially simpler than that of the x_0 and therefore may be easier to predict. With this idea, we will choose to employ method 3 for estimating $\overline{\mu}$, and the training and generating algorithms discussed in Subsection 3.2 and Subsection 3.3 will use method 3.

With our method for estimating $\overline{\mu}$ determined, we must address how θ is optimized such that $\hat{\mu}_\theta$ approximates $\overline{\mu}$.

3.2. Learning from Forward Diffusion. Let $\{x_n\}_{n=1}^N$ be a discretized sample path of the forward process starting at x_0. Solving for ε_0 in (3.1) gives us

$$\varepsilon_0 = \varepsilon_0(x_{n+1}, x_0, t_{n+1}) = \frac{x_{n+1} - \gamma_{n+1}x_0}{\beta_{n+1}} \tag{3.5}$$

for all time steps $n = 0, \ldots, N-1$. In the pursuit of generating new datapoints, we are looking to learn ε_0 without using the initial distribution $\mathbf{X}_0$. To achieve this, we will simulate trajectories of the forward process where the initial condition is the sample data. At each step, our model will predict $\varepsilon_\theta(x_{n+1}, t_{n+1})$ as an estimate of $\varepsilon_0(x_{n+1}, x_0, t_{n+1})$. Then, the model will be tuned to minimize the distance between ε_θ and ε_0. In doing so, the model will learn how to predict ε_0 without ever seeing the initial condition. Algorithm 3.1 gives a description of how a diffusion model is trained on a single point x_0.

Algorithm 3.1 Training

Parameters: $x_0 \in \mathbb{R}^d$, $N \in \mathbb{N}$, $0 < t_1 < \ldots < t_N$, θ
repeat
  Compute $x_1, \ldots, x_N$ using x_0 with (2.7)
  for $n = 0, \ldots, N-1$ **do**
    Solve for ε_0 with (3.5)
    Predict $\varepsilon_\theta = \varepsilon_\theta(x_{n+1}, t_{n+1})$ with the model
  end for
  Compute the mean square error (MSE) loss with

$$L[\theta] = \frac{1}{N} \sum_{n=1}^{N} ||\varepsilon_0 - \varepsilon_\theta||^2 \tag{3.6}$$

  Take one gradient descent step on (3.6)
until converged

3.3. Generating with Reverse Diffusion. After training the model on multiple trajectories under the forward process starting at the points in the original sample, we are now able to synthesize arbitrarily many points that "mimic" our original data. Algorithm 3.2 describes how to generate a new data point. This algorithm simulates the reverse diffusion process – at each step, $\varepsilon_\theta(x_{n+1}, t_{n+1})$ is computed as an approximation of the true noise $\varepsilon_0(x_{n+1}, x_0, t_{n+1})$. This ε_θ is used to estimate $\overline{\mu}$, which then "guides" the reverse trajectories towards an estimate of the initial condition.

Algorithm 3.2 Generating

Parameters: $N \in \mathbb{N}$, $0 < t_1 < \ldots < t_N$, θ
Sample $x_N \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $n = N-1, \ldots, 0$ **do**
  Predict $\varepsilon_\theta(x_{n+1}, t_{n+1})$
  Compute $\hat{\mu}_\theta = \hat{\mu}_\theta(x_{n+1}, t_{n+1})$ with (3.4)
  Compute $\overline{\sigma}_n$ with (2.10)
  Sample $z \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $x_n = \hat{\mu}_\theta + \overline{\sigma}_n z$
end for
return x_0

4. Applications for Improving Classifier Performance. Now, we consider diffusion models as a method to improve classifier performance. For this, we will use a dataset of credit card transactions [3, 4] from Kaggle (https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud). This dataset contains information on 284,807 transactions, of which only 492 are fraudulent (this gives a fraud prevalence rate of approximately $1.73 \cdot 10^{-3}$). Because of this, we are interested in generating data that mimics the fraudulent transactions, and then appending this generated data to the training data. In doing so, we may see improvement in a classifier's ability to detect fraud. In particular, we will focus on the precision and recall of a classifier when trained with and without diffusion augmented data, which we define as follows:

$$\text{Precision} = \frac{\text{Number of fraudulent transactions correctly labeled as fraud}}{\text{Total number of transactions labeled as fraud}},$$

$$\text{Recall} = \frac{\text{Number of fraudulent transactions correctly labeled as fraud}}{\text{Total number of fraudulent transactions}}.$$

To append synthetic data, we first split the original data into train and test sets. Then, we train a diffusion model on the fraudulent transactions from the training data, generate data with this trained diffusion model, then append this synthetic data to the training data. Because the credit card data has 29 features, we cannot directly visualize the original and synthetic data. Instead, we use dimensionality reduction to obtain a broad idea of the structure of the data. In Figure 5, we reduce the dimension of the original and synthetic fraudulent data using Uniform Manifold Approximation and Projection (UMAP). This dimensionality reduction technique leverages theory from algebraic topology to construct a lower-dimensional representation of the data in such a way that preserves topological structure [11]. In this reduced state, the synthetic data mostly agrees with the original fraud data, suggesting that the diffusion model was able to capture the structure underlying the fraudulent transaction data even in 29 dimensions.

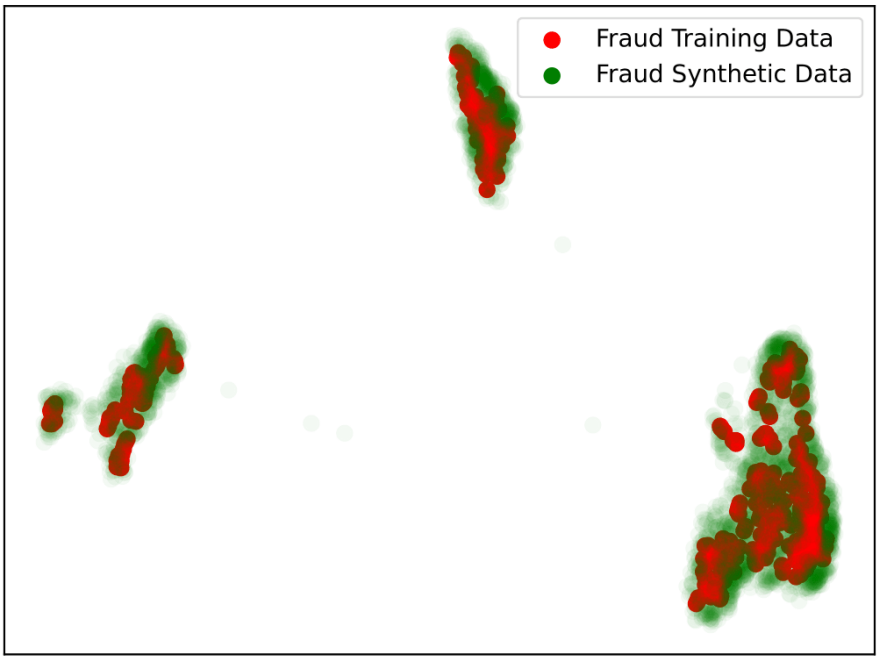


Figure 5: A plot of the training and synthetic fraud data after applying dimensionality reduction with UMAP.

4.1. Classification on the Credit Card Dataset. After augmenting the training data with data generated by a diffusion model, we now perform classification. We use the XGBoost [2] and Random Forest [8] classifiers and train them on the unmodified training data and training data augmented with our diffusion models. Table 2 and Table 3 show the precision, recall, and F_1 score (which is the harmonic mean of precision and recall) of XGBoost and Random Forest applied to the test set after being trained with these two methods. Notably, adding synthetic data improves the recall of both classifiers, but it comes at the tradeoff of precision, meaning both classifiers improve at identifying fraud with the cost of more "false accusations" of fraud.

The GitHub repository containing our implementation of diffusion models, visualization, and classification can be found here: https://github.com/justinle4/Diffusion. In particular, we utilized the PyTorch [12] library in Python to design and train the neural network underlying our diffusion model.

Table 1: XGBoost Results

Method	Precision	Recall	F_1 Score
No Augmentation	**0.8901**	0.8265	0.8571
Diffusion	0.8737	**0.8469**	**0.8601**

Precision, recall, and F_1 score for XGBoost on a test set after being trained with and without data augmentation with diffusion models. The maximum value in each column is bolded.

Table 2: Random Forest Results

Method	Precision	Recall	F_1 Score
No Augmentation	**0.9524**	0.8163	0.8791
Diffusion	0.9053	**0.8776**	**0.8912**

Precision, recall, and F_1 score for Random Forest on a test set after being trained with and without data augmentation with diffusion models. The maximum value in each column is bolded.

5. Closing Remarks. In this work, we discussed the mechanics of diffusion models for generating new samples. To do so, we started by defining a forward process as modeled by a linear SDE. From this forward process, a discretized solution to the reverse process could be obtained. The diffusion model is then trained by "witnessing" the forward process (i.e., learning to estimate the noise ε_0), and generates points by inferring the reverse process.

As an application to classification problems in imbalanced datasets, we found that data augmentation with diffusion improved recall on the credit card dataset for two tree-based classifiers while the precision noticeably suffers when augmenting data. That is, models trained on diffusion-augmented data can better detect fraud when it occurs, with the tradeoff of making more "false accusations" of fraud. For classification problems in which failing to detect the minority class is more costly than incorrectly labeling data as belonging to the minority class, data augmentation with diffusion may be advantageous.

For future work on diffusion models, we can consider implementing the Negative Prompting algorithm [1], which currently only sees applications in image generation. The Negative Prompting algorithm allows one to generate data that mimics one class while avoiding another class. On the mathematical front, connections between the reverse SDE in [17] and partial differential equations can be investigated utilizing a result known as Itô's formula (discussed in [5]). Such an approach would

Appendix A. Solving the Ornstein-Uhlenbeck Equation. Solving (2.1) and SDEs more broadly requires the development of a new type of integration called "Itô integration". This is

what allows us to make sense of integrating against $\mathrm{d}\mathbf{B}_t$, which, unlike the traditional notions of integration, integrates with respect to a stochastic process. A rigorous development of the Itô integral and the necessary properties of the integral to obtain (2.3) can be found in Evans' text on SDE [5]. The key lemma from this text is that integrating against $\mathrm{d}\mathbf{B}_t$ does not yield a number, but rather a random variable.

Lemma A.1. *Let $0 \leq a < b$ and $g \in \mathbb{L}^2(a,b)$. Then, $\int_a^b g(s)\,\mathrm{d}\mathbf{B}_s$ is normally distributed with*

$$\mathbb{E}\left[\int_a^b g(s)\,\mathrm{d}\mathbf{B}_s\right] = 0$$

and

$$\mathbb{E}\left[\left(\int_a^b g(s)\,\mathrm{d}\mathbf{B}_s\right)^2\right] = \int_a^b g^2(s)\,ds.$$

The proof of this lemma can also be found in [5] and is one of the key properties of the Itô integral. With this lemma, (2.3) can easily be obtained.

Proof. **(Proposition 2.1)** Our first step towards solving (2.1) is to make use of the method of integrating factors. Set $\mathbf{Z}_t = e^t\mathbf{X}_t$ and observe that

$$\begin{aligned}\mathrm{d}\mathbf{Z}_t &= e^t\mathbf{X}_t\,\mathrm{d}t + e^t\mathrm{d}\mathbf{X}_t = e^t\mathbf{X}_t\,\mathrm{d}t + e^t(-\mathbf{X}_t\,\mathrm{d}t + \sqrt{2}\mathrm{d}\mathbf{B}_t)\\ &= \sqrt{2}e^t\mathrm{d}\mathbf{B}_t.\end{aligned}$$

Integrating both sides over $[0,t]$ then yields

$$\mathbf{Z}_t = \mathbf{Z}_0 + \sqrt{2}\int_0^t e^s\,\mathrm{d}\mathbf{B}_s \implies \mathbf{X}_t = e^{-t}\mathbf{X}_0 + \sqrt{2}\int_0^t e^{-(t-s)}\,\mathrm{d}\mathbf{B}_s,$$

establishing (2.2). From here, we use Lemma A.1 to deduce

$$\mathbb{E}\left[\sqrt{2}\int_0^t e^{-(t-s)}\,\mathrm{d}\mathbf{B}_s\right] = (\sqrt{2}e^{-t})\mathbb{E}\left[\int_0^t e^s\,\mathrm{d}\mathbf{B}_s\right] = 0$$

and

$$\begin{aligned}\mathbb{E}\left[\left(\sqrt{2}\int_0^t e^{-(t-s)}\,\mathrm{d}\mathbf{B}_s\right)^2\right] &= (2e^{-2t})\mathbb{E}\left[\left(\int_0^t e^s\,\mathrm{d}\mathbf{B}_s\right)^2\right] = (2e^{-2t})\int_0^t e^{2s}\,ds = e^{-2(t-s)}\Big|_{s=0}^t\\ &= 1 - e^{-2t}.\end{aligned}$$

Now, we conclude

$$\begin{aligned}\mathrm{Var}\left[\sqrt{2}e^{-t}\int_0^t e^s\,\mathrm{d}\mathbf{B}_s\right] &= \mathbb{E}\left[\left(\sqrt{2}e^{-t}\int_0^t e^s\,\mathrm{d}\mathbf{B}_s\right)^2\right] - \mathbb{E}\left[\sqrt{2}e^{-t}\int_0^t e^s\,\mathrm{d}\mathbf{B}_s\right]^2\\ &= 1 - e^{-2t}.\end{aligned}$$

In other words,

$$\sqrt{2}e^{-t}\int_0^t e^s \, \mathrm{d}\mathbf{B}_s \sim \mathcal{N}(\mathbf{0}, 1 - e^{-2t}),$$

so we can rewrite $\mathbf{X}_t$ as follows:

$$\mathbf{X}_t = e^{-t}\mathbf{X}_0 + \sqrt{1 - e^{-2t}}\,\mathbf{Z}$$

with $\mathbf{Z} \sim \mathcal{N}(0, \mathbf{I})$. This is (2.3). ■

To obtain (2.7), we simply repeat the steps above, but instead integrating over $[t_n, t_{n+1}]$ rather than $[0, t]$.

Appendix B. Time Discretization. Here, we describe an algorithm to choose the values of Δt_n. Importantly, we do not explicitly set the values of Δt_n; rather, we control the values of β_n, where the β_n are as defined in (2.5). This is because β_n^2 is the variance of $\mathbf{X}_n$ at each time step, and having control over the variability of $\mathbf{X}_n$ is more immediately interpretable than the size of a time step (that is, we do not know a priori what makes a "good" choice for a time step size). These β_n are chosen to be equally spaced between the user-defined values β_1 and β_N, and Δt_n is then obtained using β_n. With this scheme, Δt_n increases with respect to n, meaning more time steps will be found closer to 0 (i.e., when the "drift" portion of forward diffusion is most pronounced). Hence, a model trained with such a time discretization scheme will more thoroughly learn the drift, which is crucial when generating points with the reverse process. In contrast, diffusion models trained using uniform time steps tend to produce synthetic data that is less faithful to the original sample since more weight is placed on learning the forward process for large t. This is the stage of forward diffusion when the data has already become noisy and remains noisy, so learning these latter parts of the forward process are not as important for inferring the reverse process.

Algorithm B.1 Time Discretization

Parameters: $N \in \mathbb{N}$, $0 < \beta_1 < \beta_N < 1$.
for $n = 1, \ldots, N$ **do**
  $\beta_n = \beta_1 + \dfrac{\beta_N - \beta_1}{N - 1} \cdot (n - 1)$
  $\Delta t_n = -\dfrac{1}{2}\ln(1 - \beta_n)$
  $t_n = \displaystyle\sum_{k=1}^{n} \Delta t_k$
end for
return $(t_1, \ldots, t_N), (\Delta t_1, \ldots, \Delta t_N)$

Appendix C. Deriving the Discretized Reverse Diffusion. We present a proof to Proposition 2.3, which crucially leverages our discrete time steps by using Bayes' theorem.

Proof. **(Proposition 2.3)** First, Bayes' theorem gives us

$$\rho(x_n \,|\, x_{n+1}, x_0) = \frac{\rho(x_{n+1} \,|\, x_n, x_0)\rho(x_n \,|\, x_0)}{\rho(x_{n+1} \,|\, x_0)}.$$

Note that by (2.7), the $\mathbf{X}_n$ all have the Markov property (in other words, the position of a trajectory at one time step depends only on its position at the previous time step). Hence, $\rho(x_{n+1} \,|\, x_n, x_0) = \rho(x_{n+1} \,|\, x_n)$. Moreover, x_{n+1} and x_0 are fixed, so the joint density $\rho(x_{n+1} \,|\, x_0)$ is a constant. So, we have $\rho(x_{n+1} \,|\, x_n) \sim \mathcal{N}(\gamma(\Delta t_{n+1})x_n, \beta(\Delta t_{n+1})^2)$ by Corollary 2.2 and $\rho(x_n \,|\, x_0) \sim \mathcal{N}(\gamma_n x_0, \beta_n^2)$ by Proposition 2.1. Since we already know that $\rho(x_n \,|\, x_{n+1}, x_0)$ is a density, we only need to consider the kernel of the right hand side, meaning all constants can be ignored. In particular, we will take all terms that are not dependent on x_n and collect them into constants that we will denote as C_1 and C_2:

$$\begin{aligned}
\rho(x_n \,|\, x_{n+1}, x_0) &= \frac{\rho(x_{n+1} \,|\, x_n)\rho(x_n \,|\, x_0)}{\rho(x_{n+1} \,|\, x_0)} \\
&\propto \rho(x_{n+1} \,|\, x_n)\rho(x_n \,|\, x_0) \\
&\propto \exp\left(-\frac{1}{2\beta(\Delta t_{n+1})^2} \cdot [x_{n+1} - \gamma(\Delta t_{n+1})x_n]^2\right) \cdot \exp\left(-\frac{1}{2\beta_n^2} \cdot [x_n - \gamma_n x_0]^2\right) \\
&= \exp\left(-\frac{x_{n+1}^2 - 2\gamma(\Delta t_{n+1})x_{n+1}x_n + \gamma(\Delta t_{n+1})^2 x_n^2}{2\beta(\Delta t_{n+1})^2}\right) \cdot \exp\left(-\frac{x_n^2 - 2\gamma_n x_n x_0 + \gamma_n^2 x_0^2}{2\beta_n^2}\right) \\
&= \exp\left(-\frac{\beta_n^2(x_{n+1}^2 - 2\gamma(\Delta t_{n+1})x_{n+1}x_n + \gamma(\Delta t_{n+1})^2 x_n^2) + \beta(\Delta t_{n+1})^2(x_n^2 - 2\gamma_n x_n x_0 + \gamma_n^2 x_0^2)}{2\beta(\Delta t_{n+1})^2\beta_n^2}\right) \\
&= \exp\left(-\frac{[\beta_n^2\gamma(\Delta t_{n+1})^2 + \beta(\Delta t_{n+1})^2]x_n^2 - 2[\gamma(\Delta t_{n+1})\beta_n^2 x_{n+1} + \gamma_n\beta(\Delta t_{n+1})^2 x_0]x_n + C_1}{2\beta(\Delta t_{n+1})^2\beta_n^2}\right).
\end{aligned}$$

Notice that

$$\begin{aligned}
\beta_n^2\gamma(\Delta t_{n+1})^2 + \beta(\Delta t_{n+1})^2 &= (1 - e^{-2t_n})e^{-2\Delta t_n} + (1 - e^{-2\Delta t_n}) \\
&= e^{-2\Delta t_n} - e^{-2t_{n+1}} + (1 - e^{-2\Delta t_n}) \\
&= 1 - e^{-2t_{n+1}} \\
&= \beta_{n+1}^2.
\end{aligned}$$

Using this fact, we find

$$\begin{aligned}
\rho(x_n \,|\, x_{n+1}, x_0) &\propto \exp\left(-\frac{\beta_{n+1}^2 x_n^2 - 2[\gamma(\Delta t_{n+1})\beta_n^2 x_{n+1} + \gamma_n\beta(\Delta t_{n+1})^2 x_0]x_n}{2\beta(\Delta t_{n+1})^2\beta_n^2}\right) \cdot \exp\left(\frac{C_1}{2\beta(\Delta t_{n+1})^2\beta_n^2}\right) \\
&\propto \exp\left(-\frac{\beta_{n+1}^2 x_n^2 - 2[\gamma(\Delta t_{n+1})\beta_n^2 x_{n+1} + \gamma_n\beta(\Delta t_{n+1})^2 x_0]x_n}{2\beta(\Delta t_{n+1})^2\beta_n^2}\right) \\
&\propto \exp\left(-\frac{x_n^2 - 2 \cdot \frac{\gamma(\Delta t_{n+1})\beta_n^2 x_{n+1}}{\beta_{n+1}^2}x_n - 2 \cdot \frac{\gamma_n\beta(\Delta t_{n+1})^2 x_0}{\beta_{n+1}^2}x_n}{2 \cdot \frac{\beta(\Delta t_{n+1})^2\beta_n^2}{\beta_{n+1}^2}}\right) \\
&= \exp\left(-\frac{\left(x_n - \frac{\gamma(\Delta t_{n+1})\beta_n^2 x_{n+1}}{\beta_{n+1}^2} - \frac{\gamma_n\beta(\Delta t_{n+1})^2 x_0}{\beta_{n+1}^2}\right)^2 + C_2}{2 \cdot \frac{\beta(\Delta t_{n+1})^2\beta_n^2}{\beta_{n+1}^2}}\right) \\
&\propto \exp\left(-\frac{\left(x_n - \left(\frac{\gamma(\Delta t_{n+1})\beta_n^2}{\beta_{n+1}^2} \cdot x_{n+1} + \frac{\gamma_n\beta(\Delta t_{n+1})^2}{\beta_{n+1}^2} \cdot x_0\right)\right)^2}{2 \cdot \frac{\beta(\Delta t_{n+1})^2\beta_n^2}{\beta_{n+1}^2}}\right).
\end{aligned}$$

We recognize this as the kernel of a $\mathcal{N}(\overline{\mu}, \overline{\sigma}_n^2)$ distribution. ∎

Acknowledgments. This article began as my honors thesis for my Bachelor's degree at Arizona State University's School of Mathematical and Statistical Sciences. I would like to thank my advisors, Dr. Sebastien Motsch and Dr. Johannes Brust, for their mentorship, expertise, and inspiration. Thank you as well to the School of Mathematical and Statistical Sciences and Barrett, the Honors College at Arizona State University for giving me the support to succeed as an undergrad. Finally, thank you to my high school calculus teacher Mr. Tutt, who ignited my passion for mathematics and started me on this incredible journey.

REFERENCES

[1] M. Armandpour, A. Sadeghian, H. Zheng, A. Sadeghian, and M. Zhou, *Re-imagine the negative prompt algorithm: Transform 2d diffusion into 3d, alleviate Janus problem and beyond*, 2023, arXiv:2304.04968 [cs.CV].

[2] T. Chen and C. Guestrin, *Xgboost: A scalable tree boosting system*, in Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, New York, NY, USA, 2016, Association for Computing Machinery, p. 785–794, doi:10.1145/2939672.2939785, https://doi.org/10.1145/2939672.2939785.

[3] A. Dal Pozzolo, G. Boracchi, O. Caelen, C. Alippi, and G. Bontempi, *Credit card fraud detection: A realistic modeling and a novel learning strategy*, IEEE Trans. Neural Netw. Learn. Syst., PP (2017), pp. 1–14, doi:10.1109/TNNLS.2017.2736643.

[4] A. Dal Pozzolo, O. Caelen, Y.-A. Le Borgne, S. Waterschoot, and G. Bontempi, *Learned lessons in credit card fraud detection from a practitioner perspective*, Expert Syst. Appl., 41 (2014), p. 4915–4928, doi:10.1016/j.eswa.2014.02.026.

[5] L. Evans, *An Introduction to Stochastic Differential Equations*, American Mathematical Society, 2013.

[6] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative adversarial networks*, Commun. ACM, 63 (2020), p. 139–144, doi:10.1145/3422622, https://doi.org/10.1145/3422622.

[7] J. Ho, A. Jain, and P. Abbeel, *Denoising diffusion probabilistic models*, in Advances in Neural Information Processing Systems, vol. 33, Curran Associates, Inc., 2020, pp. 6840–6851, https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf.

[8] T. K. Ho, *Random decision forests*, in Proceedings of 3rd International Conference on Document Analysis and Recognition, vol. 1, 1995, pp. 278–282 vol.1, doi:10.1109/ICDAR.1995.598994.

[9] Y. Jiang, S. Chang, and Z. Wang, *TransGAN: Two pure transformers can make one strong GAN, and that can scale up*, in Advances in Neural Information Processing Systems, M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, eds., vol. 34, Curran Associates, Inc., 2021, pp. 14745–14758, https://proceedings.neurips.cc/paper_files/paper/2021/file/7c220a2091c26a7f5e9f1cfb099511e3-Paper.pdf.

[10] P. D. Kingma and M. Welling, *Auto-encoding variational Bayes*, in Proceedings of 2nd International Conference on Learning Representations, 2014, arXiv:1312.6114 [stat.ML].

[11] L. McInnes, J. Healy, N. Saul, and L. Grossberger, *UMAP: Uniform manifold approximation and projection*, J. Open Source Softw., 3 (2018), p. 861, doi:10.21105/joss.00861.

[12] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, *Pytorch: An imperative style, high-performance deep learning library*, in Advances in Neural Information Processing Systems, vol. 32, Curran Associates, Inc., 2019, https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-learn: Machine learning in Python*, J. Mach.

Learn. Res., 12 (2011), pp. 2825–2830, http://jmlr.org/papers/v12/pedregosa11a.html.

[14] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, *Improving language understanding by generative pre-training*, 2018, https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

[15] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, *Zero-shot text-to-image generation*, in Proceedings of the 38th International Conference on Machine Learning, M. Meila and T. Zhang, eds., vol. 139 of Proceedings of Machine Learning Research, PMLR, 18–24 Jul 2021, pp. 8821–8831, https://proceedings.mlr.press/v139/ramesh21a.html.

[16] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, *Deep unsupervised learning using nonequilibrium thermodynamics*, in International conference on machine learning, PMLR, 2015, pp. 2256–2265, https://proceedings.mlr.press/v37/sohl-dickstein15.html.

[17] A. Thiéry, *Denoising diffusion probabilistic models (DDPM)*, 2013, https://alexxthiery.github.io/notes/DDPM/DDPM.html.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, *Attention is all you need*, in Advances in Neural Information Processing Systems, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds., vol. 30, Curran Associates, Inc., 2017, https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[19] K. Wang, C. Gou, Y. Duan, Y. Lin, X. Zheng, and F.-Y. Wang, *Generative adversarial networks: introduction and outlook*, IEEE/CAA J. Autom. Sin., 4 (2017), pp. 588–598, doi:10.1109/JAS.2017.7510583.

[20] Y. Zhu, K. Zhang, J. Liang, J. Cao, B. Wen, R. Timofte, and L. Van Gool, *Denoising diffusion models for plug-and-play image restoration*, in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, June 2023, pp. 1219–1229, arXiv:2305.08995 [cs.CV].