GENERATIVEZOO: A UNIFIED REPOSITORY OF GENERATIVE MODELS FOR COMPUTER VISION

TECHNICAL REPORT
v1.3.0

Francisco Caetano*, **Christiaan Viviers,** **Peter H.N. De With,** **Fons van der Sommen**
Department of Electrical Engineering
Eindhoven University of Technology
*f.caetano@tue.nl

ABSTRACT

Generative models have become a cornerstone of modern computer vision, driving significant advancements in applications such as image synthesis, data augmentation, style transfer, and inpainting. Despite their widespread use, the field of generative modeling remains fragmented, making it challenging for researchers and practitioners to access, compare, and evaluate the wide variety of models available. To address these challenges, we introduce the GenerativeZoo, a comprehensive and organized repository that aggregates a diverse collection of state-of-the-art generative models, with a specific focus on synthetic image generation. By consolidating these models into a single, unified, and user-friendly platform, the GenerativeZoo simplifies the process of working with generative methods, promoting easier experimentation and faster adoption of these powerful tools. Designed to be scalable, flexible, and highly accessible, this platform aims to accelerate research in generative image generation, bridging the gap between cutting-edge generative approaches and their real-world applications in machine learning and computer vision. The GenerativeZoo is publicly available [1].

Keywords Autoregressive Models · Diffusion Models · Flow Matching · Generative Adversarial networks · Generative Modelling · Model Zoo · Normalizing Flows · Open Source · Score Matching · Variational Autoencoders

1 Introduction

Generative models have become a key component of modern computer vision, driving advances in various applications such as image synthesis [1], data augmentation [2], style transfer [3], and inpainting [4]. These models allow for the creation of high-quality synthetic data, which is invaluable in scenarios where real data is scarce or difficult to obtain. Moreover, generative models have found creative uses in art, entertainment, and simulation, showcasing the broad potential of AI-driven content generation.

However, working with generative models poses several challenges. The field is fragmented, with numerous models and implementations scattered across different papers, research groups, and platforms. This lack of a unified resource makes it difficult for researchers and practitioners to access, evaluate, and compare the performance of different generative models. Additionally, the absence of standardized frameworks for training, sampling, and evaluating models complicates their adoption and integration into practical applications.

To address these challenges, we introduce the GenerativeZoo, a comprehensive repository that aggregates a diverse collection of state-of-the-art generative models, focusing on synthetic image generation. The repository provides a standardized framework for training, sampling, and evaluating generative models, allowing users to easily experiment with and deploy a wide range of image generation techniques. By consolidating multiple models into a single, user-friendly platform, the GenerativeZoo aims to simplify the process of working with generative models and accelerate the adoption of these powerful tools. The main contributions of this repository are as follows:

[1] https://caetas.github.io/generativezoo.html

- A centralized platform that aggregates diverse generative models for synthetic image generation, offering easy access to a wide range of techniques.
- Standardization of workflows for training, sampling, and evaluating models, simplifying their integration into existing research and applications.
- A user-friendly environment enables researchers to experiment with state-of-the-art generative models with minimal setup while facilitating cross-disciplinary collaboration by providing a common platform to explore various generative methods and their applications in computer vision.
- Continuous updates and expansion to include new models and advances in generative modeling techniques.

The GenerativeZoo provides a scalable, flexible, and accessible resource for accelerating research in synthetic image generation, helping to bridge the gap between cutting-edge generative methods and their practical applications in machine learning and computer vision.

2 Generative Models

Generative models are a class of statistical models designed to generate new data instances by capturing the underlying distribution of the data. Unlike discriminative models, which focus on distinguishing between different types of data, generative models learn the complete data distribution, enabling them to create new samples that closely resemble the training data. Formally, generative models estimate the joint probability $p(X, Y)$, which represents the likelihood of both the data and its associated labels. In contrast, discriminative models capture the conditional probability $p(Y|X)$, which represents the likelihood of a label given an input.

Understanding the data distribution $p(X)$ is fundamental to the success of generative models and has applications beyond merely generating new data. It allows systems to assess whether an object has been encountered before, make informed decisions by properly weighing probabilities, and quantify uncertainty about the environment or predictions. Furthermore, estimating $p(X)$ enables active learning by identifying rare or uncertain objects that may benefit from human labeling, thus improving the model's performance iteratively. Finally, having an accurate estimate of $p(X)$ facilitates the synthesis of realistic new data, which can augment training datasets or support simulations. We follow the categorization of generative models by Tomczak [5], identifying four major groups relevant to image generation tasks, all of which are included in the GenerativeZoo.

- *Latent Variable Models*: These models introduce hidden variables to explain the underlying structure of the data. Variational Autoencoders (VAEs) [6] and Generative Adversarial Networks (GANs) [7] are common examples in this category.
- *Flow-Based Models*: These models use invertible transformations to map data from a simple distribution (such as a Gaussian) to the data distribution, allowing for both sampling and exact likelihood evaluation. RealNVP [8] and Glow [9] are examples of flow-based models.
- *Autoregressive Generative Models (ARM)*: These models generate data sequentially, predicting each part of the data conditioned on the previous parts. Examples include the PixelCNN [10] and Transformers [11].
- *Score-Based Models*: These models focus on estimating the gradients of data distributions directly, enabling a continuous transformation from noise to data, through strategies like Score Matching [12] or Flow Matching [13]. Despite their distinct characteristics, they are often discussed in relation to latent variable models.

2.1 Latent Variable Models

Latent variable models are designed to capture the underlying factors of high-dimensional data by introducing hidden variables that represent essential unobserved aspects of the data. In this framework, the generative process involves sampling from a low-dimensional latent space and then generating the observed data based on these latent variables.

Let $x \in \mathbb{X}^D$ represent high-dimensional data and let $z \in \mathbb{Z}^M$ represent low-dimensional latent variables (e.g., $\mathbb{Z} = \mathbb{R}^M$) that capture the hidden factors crucial for generation. Mathematically, the joint distribution of the data x and the latent variables z is factorized as $p(x, z) = p(x|z)p(z)$. To generate data, we first sample from the prior distribution of the latent variables, $z \sim p(z)$, and then sample the observed data from the posterior, i.e., the conditional distribution $p(x|z)$. Since we only observe x during training, the latent variables z are marginalized, leading to the marginal likelihood

$$p(x) = \int p(x|z)p(z)\, dz. \tag{1}$$

2.1.1 Variational Autoencoders

The Variational Autoencoder [6] starts as a way to approximate the solution to this integral. The approach is an application of variational inference [14]. Let us consider a family of variational distributions parameterized by ϕ, $q_\phi(z)$. Then, the logarithm of the marginal distribution in Equation 1 can be approximated as

$$\begin{aligned}\ln p(x) &= \ln \int \frac{q_\phi(z)}{q_\phi(z)} p(x|z)p(z)\, dz \\ &= \ln \mathbb{E}_{z\sim q_\phi(z)}\left[\frac{p(x|z)p(z)}{q_\phi(z)}\right] \\ &\geq \mathbb{E}_{z\sim q_\phi(z)}\left[\ln\left(\frac{p(x|z)p(z)}{q_\phi(z)}\right)\right] \\ &= \mathbb{E}_{z\sim q_\phi(z)}\left[\ln p(x|z)\right] - \mathbb{E}_{z\sim q_\phi(z)}\left[\ln q_\phi(z) - \ln p(z)\right].\end{aligned} \tag{2}$$

To improve efficiency, an amortized variational posterior is introduced; instead of computing a separate variational posterior $q_\phi(z)$ for each input x, we learn a function, typically a neural network, that directly produces the parameters of the posterior $q_\phi(z|x)$ for each input. This reduces the computational cost by reusing the same model to estimate the posterior for any input, rather than recalculating it each time. With this amortized variational posterior, the log-likelihood can be bounded as

$$\ln p(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}\left[\ln p(x|z)\right] - \mathbb{E}_{z\sim q_\phi(z|x)}\left[\ln q_\phi(z|x) - \ln p(z)\right]. \tag{3}$$

This leads to an autoencoder-like model in which the encoder $q_\phi(z|x)$ and the decoder $p_\theta(x|z)$ are both stochastic. This model is known as a Variational Autoencoder, and the lower bound is called the evidence lower bound (ELBO). The first term in the ELBO, $\mathbb{E}_{z\sim q_\phi(z|x)}\left[\ln p(x|z)\right]$, represents the negative reconstruction error, while the second term, $\mathbb{E}_{z\sim q_\phi(z|x)}\left[\ln q_\phi(z|x) - \ln p(z)\right]$, serves as a regularizer, often referred to as the Kullback–Leibler (KL) divergence. Essentially, the reconstruction term optimizes the encoding-decoding process, while the regularization term aligns the encoder distributions with the prior distribution.

2.1.2 Hierarchical VAEs

Hierarchical VAEs [15, 16] extend the standard VAE framework by introducing a hierarchy of latent variables, allowing for more expressive modeling of complex data distributions. Let us consider a two-level Hierarchical VAE where z_2 represents the highest-level latent variable and z_1 the lowest-level one. In these models, the generative process is defined by a top-down sequence of conditional distributions: $p(x|z_1)$, $p(z_1|z_2)$, and $p(z_2)$. A neural network parameterizes the conditional distributions $p(x|z_1)$ and $p(z_1|z_2)$, producing their respective parameters (e.g., means and variances). The prior $p(z_2)$ is typically chosen to be a simple distribution, such as a standard Gaussian, and does not require a neural network. The variational posterior follows the same hierarchical structure, decomposed as

$$Q(z_1, z_2|x) = q(z_1|z_2, x)q(z_2|x). \tag{4}$$

This formulation adds x as a conditioning variable in the posterior, tightly coupling the generative distributions and the variational posteriors through a shared top-down path. Such shared parameterization serves as a useful inductive bias, enhancing the ability of the model to capture complex dependencies in the data. These hierarchical VAEs have been further refined in models like BIVA [17], and NVAE [18], showcasing diverse architectural innovations while adhering to this shared top-down modeling framework.

2.1.3 Diffusion Models

Diffusion Models [19, 20] can be comprehended within the framework of a Hierarchical VAE, where the bottom-up trajectory, which corresponds to the variational posterior, is characterized by a diffusion process such as Gaussian diffusion. Meanwhile, the top-down trajectory is parameterized by deep neural networks, representing a reversed diffusion process. Interestingly, the bottom-up path in diffusion models can be fixed, which means that it does not require learnable parameters. In this setup, the diffusion process introduces Gaussian noise at each layer during the bottom-up path. By the final layer, this noise accumulates such that the latent variable resembles a standard Gaussian distribution. As the process is explicitly designed to converge to a standard Gaussian in the last layer, this design inherently avoids the issue of posterior collapse commonly seen in Hierarchical VAEs [21].

Similarly to VAEs, diffusion models aim to model a distribution over data $p_\theta(x)$, with an additional set of latent variables $z_{1:T} = [z_1, \ldots, z_T]$. The marginal likelihood can be obtained by integrating the latent variables:

$$p_\theta(x) = \int p_\theta(x, z_{1:T})\, dz_{1:T}. \tag{5}$$

With $x \in \mathbb{R}^D$ and $z_i \in \mathbb{R}^D$ for $i = 1, \ldots, T$, the joint distribution $p_\theta(x, z_{1:T})$ can be structured as a first-order Markov chain with Gaussian transitions

$$p_\theta(x, z_{1:T}) = p_\theta(x|z_1) \left(\prod_{i=1}^{T-1} p_\theta(z_i|z_{i+1}) \right) p_\theta(z_T). \tag{6}$$

The latent variables share the same dimensionality as the data, akin to flow-based models, as we will explore in Section 2.2. This setup corresponds to a hierarchical latent variable model, if we approximate the posterior through the family of variational distributions

$$Q_\phi(z_{1:T}|x) = q_\phi(z_1|x) \prod_{i=2}^{T} q_\phi(z_i|z_{i-1}). \tag{7}$$

Unlike traditional approaches, where normal distributions parameterized by neural networks are used, Diffusion Models define the variational posteriors through a Gaussian forward diffusion process. Each step $q_\phi(z_i|z_{i-1})$ simply scales the previous variable z_{i-1} by $\sqrt{1-\beta_i}$ and adds noise with variance β_i. Using the reparameterization trick to represent z_{i-1}, we can formulate it as

$$q_\phi(z_i|z_{i-1}) = \mathcal{N}\left(z_i \mid \sqrt{1-\beta_i} z_{i-1}, \beta_i \mathbf{I}\right), \text{with } z_i = \sqrt{1-\beta_i} z_{i-1} + \sqrt{\beta_i} \odot \epsilon, \tag{8}$$

where $\epsilon \sim \mathcal{N}(0, \mathbf{I})$. It is important to mention that in this formulation $z_0 = x$.

Given that the primary distinction between a Diffusion Model and a Hierarchical VAE lies in how the variational posteriors are defined and the dimensionality of the latent variables, the overall framework remains fundamentally similar. Consequently, the learning objective is the same as in other VAEs - the ELBO. Ho *et al.* [20] introduced a pivotal shift in how the reverse diffusion process is trained that significantly boosted its computational efficiency; instead of reconstructing the image at a given timestep, the model is trained by predicting the noise added during the forward diffusion process. This change not only simplifies the training objective but also aligns with the mathematical properties of the diffusion process. To understand this, consider forward diffusion, which progressively corrupts an input image x_0 by adding Gaussian noise at each step. The forward distribution for a given timestep t can be written as

$$q(z_t|x_0) = \mathcal{N}(z_t|\sqrt{\bar{\alpha}_t} x_0, (1-\bar{\alpha}_t)\mathbf{I}), \tag{9}$$

where $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$. This equation allows us to directly sample z_t at any step t without iterating through all preceding timesteps, providing a computational advantage. To reverse this diffusion, the model approximates the conditional distributions $p_\theta(z_{t-1}|z_t)$ to recover x_0. The reverse mean $\mu_\theta(z_t)$ is parameterized such that it aligns with the analytically derived forward mean

$$\tilde{\mu}_t(z_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t} x_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t} z_t. \tag{10}$$

Instead of directly predicting $\tilde{\mu}_t$, the model predicts the noise $\epsilon_\theta(z_t)$ used to corrupt x_0. This formulation reduces the task of learning $\mu_\theta(z_t)$ to predicting the added noise ϵ, simplifying the optimization and improving numerical stability. With $\epsilon_\theta(z_t)$ parameterized by a neural network, the reverse mean is expressed as

$$\mu_\theta(z_t) = \frac{1}{\sqrt{\alpha_t}} \left(z_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(z_t) \right). \tag{11}$$

The stochastic training method, in which a random time step t is chosen during each optimization step, significantly reduces memory requirements, allowing model training with thousands of time steps. The training objective is further simplified by re-expressing it in terms of noise prediction. Specifically, the model is trained to minimize the error between the true noise ϵ and the predicted noise ϵ_θ, with the simplified training loss

$$\mathcal{L}_t = \mathbb{E}_{t, x_0, \epsilon} \left[\|\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon, t)\|^2 \right]. \tag{12}$$

Song *et al.* [22] introduced Denoising Diffusion Implicit Models (DDIMs), an efficient class of iterative implicit probabilistic models that share the same training procedure as DDPMs. DDIMs can significantly accelerate the sampling process, achieving up to a 10-fold improvement in speed. Starting from the probability distribution $p_\theta(x_{1:T})$, a sample x_{t-1} can be generated from a given sample x_t

$$x_{t-1} = \underbrace{\sqrt{\bar{\alpha}_{t-1}} \left(\frac{x_t - \sqrt{1-\bar{\alpha}_t}\, \boldsymbol{\epsilon}_\theta(x_t)}{\sqrt{\bar{\alpha}_t}} \right)}_{\text{"predicted } x_0\text{"}} + \underbrace{\sqrt{1-\bar{\alpha}_{t-1}-\sigma_t^2} \cdot \boldsymbol{\epsilon}_\theta(x_t)}_{\text{"direction pointing to } x_t\text{"}} + \underbrace{\sigma_t \boldsymbol{\epsilon}_t}_{\text{"random noise"}}. \tag{13}$$

Here, $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ represents standard Gaussian noise that is independent of x_t, and $\bar{\alpha}_0 = 1$. The generative process can be adapted by choosing different values for σ_t, which affects the sampling dynamics while using the same trained model $\boldsymbol{\epsilon}_\theta$. Special cases of DDIM arise from specific choices of σ_t. When σ_t is defined as $\sqrt{\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \frac{1-\bar{\alpha}_t}{\bar{\alpha}_{t-1}}}$, the forward process becomes Markovian, and the generative process follows the standard DDPM formulation. On the other hand, setting $\sigma_t = 0$ for all t results in a deterministic forward process, where the generative dynamics skip the addition of random noise.

2.1.4 Generative Adversarial Networks

Generative Adversarial Networks [7] transform the previous problem of calculating the integral into a problem of sampling from the prior,i.e., the generator can constitute an implicit unknown distribution, as long as we can sample from it. GANs can be viewed as a min-max game between two competing models: the generator and the discriminator. The generator is responsible for creating fake data (such as images), while the discriminator's task is to distinguish real data (from the true data distribution) from the fake data generated by the generator. Over time, the generator improves its ability to create more realistic data, while the discriminator becomes better at identifying fakes. This dynamic creates a competitive environment where both models learn from each other, pushing each other toward improvement.

The discriminator is a classifier, $D_\alpha : X \to [0, 1]$, that takes an input x and returns a probability representing the likelihood that x is real (i.e., it comes from the true data distribution). On the other hand, the generator is a function, $G_\beta : Z \to X$, that generates data from a latent noise variable z. The goal of the generator is to produce data that is so realistic that the discriminator cannot distinguish it from real data. Mathematically, this interaction is captured by the following objective function

$$\min_\beta \max_\alpha \mathbb{E}_{x \sim p_{\text{real}}} [\log D_\alpha(x)] + \mathbb{E}_{z \sim p(z)} [\log (1 - D_\alpha(G_\beta(z)))] . \tag{14}$$

In this equation, the first term, $\mathbb{E}_{x \sim p_{\text{real}}}[\log D_\alpha(x)]$, represents the discriminator's goal of correctly classifying real data as real. The second term, $\mathbb{E}_{z \sim p(z)}[\log(1 - D_\alpha(G_\beta(z)))]$, represents the generator's goal of producing fake data that the discriminator will classify as real. Essentially, the generator is trying to "fool" the discriminator into classifying its generated data as real, while the discriminator tries to resist being fooled. This objective is referred to as the adversarial loss, as the generator and discriminator are in direct opposition, with each aiming to outperform the other. Through this adversarial training process, both models continuously improve, ultimately leading to the generation of highly realistic data by the generator.

2.2 Flow-based Models

Flow-based methods in generative modeling leverage the concept of applying bijective transformations to a continuous random variable, enabling the construction of complex distributions from simpler ones. Specifically, if we begin with a random variable z drawn from a known distribution $p(z)$, we can transform it into a variable x using a bijective transformation f, and the distribution of x is given by

$$p(x) = p(z = f^{-1}(x)) \left| \frac{\partial f^{-1}(x)}{\partial x} \right| . \tag{15}$$

The derivative term is the absolute value of the Jacobian determinant of the inverse of the transformation function, f^{-1}, and is crucial for correctly normalizing the distribution of x after the transformation. The Jacobian determinant essentially scales the distribution by accounting for the changes in volume caused by the transformation f. In other words, it normalizes the distribution, ensuring that the transformation does not artificially inflate or shrink the probability density. This is expressed mathematically as

$$\left| \frac{\partial f^{-1}(x)}{\partial x} \right| = \left| \det J_{f^{-1}}(x) \right| , \text{with } J_{f^{-1}}(x) = \begin{bmatrix} \frac{\partial f_1^{-1}}{\partial x_1} & \cdots & \frac{\partial f_1^{-1}}{\partial x_D} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_D^{-1}}{\partial x_1} & \cdots & \frac{\partial f_D^{-1}}{\partial x_D} \end{bmatrix} . \tag{16}$$

By the inverse function theorem, the distribution of x can be rewritten as

$$p(x) = p(z = f^{-1}(x)) \left| Jf(z) \right|^{-1} . \tag{17}$$

2.2.1 Normalizing Flows

A Normalizing Flow [23] is a generative model built upon a sequence of invertible transformations $f_k : \mathbb{R}^D \to \mathbb{R}^D$ parameterized by neural networks that transform a simple base distribution, like a Gaussian, into a complex target

distribution. The previous principles of change of variables extend naturally to the modeling of complex, high-dimensional data distributions, such as images. Starting from a known base distribution, such as a standard Gaussian $\pi(z_0) = \mathcal{N}(z_0 \mid 0, I)$, the probability density function of the transformed variable x is given by

$$p(x) = \pi\left(z_0 = f^{-1}(x)\right) \prod_{i=1}^{K} \left|J_{f_i}(z_{i-1})\right|^{-1}. \tag{18}$$

The logarithm of the resulting density $p(x)$ provides further insight into the components of the model

$$\ln p(x) = \ln \mathcal{N}\left(z_0 = f^{-1}(x) \mid 0, I\right) - \sum_{i=1}^{K} \ln \left|J_{f_i}(z_{i-1})\right|, \tag{19}$$

with the first term corresponding to the log-density of the base distribution after applying the inverse transformation f^{-1}. The second term, $\sum_{i=1}^{K} \ln |J_{f_i}(z_{i-1})|$, acts as a regularizer that penalizes changes in volume induced by the transformations. This ensures that the final distribution is properly normalized and captures the effect of local scaling adjustments caused by the transformations. The neural networks are required to have two essential properties: invertibility, to ensure that we can evaluate both the forward and inverse transformations during training and inference; efficiency, with a simplified computation of $\ln |J_{f_i}|$. Thus, the design of the Normalizing Flow architecture focuses on invertible neural networks with tractable Jacobian determinants.

2.3 Autoregressive Models

Consider a high-dimensional random variable $x \in X^D$, where $X = \mathbb{R}$. Our objective is to model the joint distribution $p(x)$ of x. Instead of directly parameterizing $p(x)$, we can reformulate the joint distribution using the product rule, breaking it down into simpler conditional distributions. This allows us to express $p(x)$ as

$$p(x) = p(x_1) \prod_{d=2}^{D} p(x_d \mid x_{<d}), \tag{20}$$

where $x_{<d} = [x_1, x_2, \ldots, x_{d-1}]^\top$ represents the set of variables preceding x_d. For instance, if $x = [x_1, x_2, x_3]^\top$, then the decomposition is $p(x) = p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2)$. The product rule provides a systematic way to decompose the joint distribution into conditional distributions. While this factorization is mathematically elegant, directly modeling all the conditional distributions $p(x_d \mid x_{<d})$ poses significant practical challenges. The problem arises because it would require D separate models — one for each conditional distribution, with increasing complexity as the conditioning set grows. This is computationally infeasible and inefficient, especially for high-dimensional data. Autoregressive models provide a more efficient solution by using a single, shared model for all conditional distributions $p(x_d \mid x_{<d})$.

To make this feasible, assumptions about the dependency structure of the data are necessary. One such assumption is finite memory, where each variable x_d is conditioned only on a fixed number of preceding variables. This approach significantly reduces the complexity of the model by limiting the size of the conditioning set. However, finite memory introduces limitations. By restricting the dependencies to a fixed number of variables, the model sacrifices its ability to capture long-range dependencies. This trade-off is particularly problematic in domains such as image processing, where long-range correlations are essential for capturing complex patterns. For instance, understanding the structure of an image often requires integrating information across distant regions, which a short-memory model cannot achieve. A possible solution is to use a hidden context that acts as a memory, enabling the model to learn long-range dependencies [24, 25]. Alternatively, Convolutional neural networks (CNNs) can also effectively model long-range dependencies through Causal Convolutions [26]. However, convolutional layers are limited by their fixed neighborhood grid, making them less suitable for irregular structures; attention mechanisms, further improved by the introduction of Transformers [11], excel at learning long-range connections.

2.4 Score-based Models

The strength of diffusion models lies in their ability to process data through thousands of small, incremental noise removal steps, guided by a shared neural network. This approach leverages deep hierarchical structures to achieve impressive generative results. However, taking the idea further—toward infinitely many steps—introduces the need to handle continuous time, which can only be achieved through differential equations. Score-based models offer an alternative to diffusion by focusing on estimating the gradients of data distributions (scores) directly, enabling a continuous transformation from noise to data. This approach provides a different mechanism for transforming noise into data and serves as the foundation for building continuous-time generative models.

Building on the goal of transforming noise into data iteratively, Score-based Models leverage Stochastic Gradient Langevin Dynamics (SGLD) [27], a method grounded in stochastic updates. This approach allows us to sample from the real data distribution, $p_{\text{real}}(x)$, by refining an initial random point in the data space. Starting with $\hat{x}$, a randomly chosen point where $p_{\text{real}}(\hat{x}) \approx 0$, the method iteratively updates this point to align it with regions of higher probability. The update rule is given by

$$x_{t+\Delta t} = x_t + \alpha \nabla_{x_t} \ln p_{\text{real}}(x_t) + \eta\epsilon, \tag{21}$$

where x_t is the current sample at time t, $\nabla_{x_t} \ln p_{\text{real}}(x_t)$ is the gradient of the log-probability at x_t, guiding the sample toward high-probability regions. The step size $\alpha > 0$ controls the magnitude of the gradient update, while $\eta > 0$ scales the noise term $\epsilon \sim \mathcal{N}(0, I)$, which ensures exploration of the data space. Through this iterative process, the method generates samples that align with p_{real}, effectively transforming noise into valid data.

2.4.1 Score Matching Models

One major challenge in applying SGLD is that the real data distribution, $p_{\text{real}}(x)$, is unknown. While we could turn to generative models like GANs, VAEs, or normalizing flows to approximate $p_{\text{real}}(x)$, there is an alternative approach. Upon closer examination of the SGLD update equation, we realize that $p_{\text{real}}(x)$ itself is not necessary - what we truly need is the gradient of its logarithm, $\nabla_x \ln p_{\text{real}}(x)$, which is referred to as the score function. The score function provides a vector field that points toward the modes of the real data distribution. For example, in the case of a multimodal distribution, the score function can be visualized as vectors on a grid pointing toward high-probability regions, guiding Langevin dynamics toward these areas.

This insight allows us to reformulate the problem: instead of modeling the data distribution $p_{\text{real}}(x)$, we focus on learning the score function. Since the score function is a gradient, it has the same shape as x and can be represented as a vector. By learning $s_\theta(x)$, we enable SGLD to produce samples that follow the real data distribution without explicitly modeling $p_{\text{real}}(x)$. To model this, we optimize a regression objective [28, 29], minimizing the difference between the true score function and a parameterized approximation, $s_\theta(x)$, using

$$\mathcal{L}(\theta) = \frac{1}{2} \int \|s_\theta(x) - \nabla_x \ln p_{\text{real}}(x)\|^2 p_{\text{real}}(x)\, dx. \tag{22}$$

The challenge in directly using this objective arises because $p_{\text{data}}(x)$ is a discrete distribution represented as a sum of Dirac delta functions, which is non-differentiable. Since Dirac's delta function is not differentiable with respect to x, we cannot apply standard gradient-based methods to solve the optimization problem. To overcome this issue, a small Gaussian noise with variance σ^2 is introduced to the data. Specifically, the data points are perturbed by adding noise $\tilde{x}_n = x_n + \sigma \cdot \epsilon$, where $\epsilon \sim \mathcal{N}(0, I)$ is sampled from a standard normal distribution. This transformation turns the data distribution into a Gaussian mixture, where each data point x_n is now the mean of a Gaussian distribution with variance σ^2

$$q_{\text{data}}(\tilde{x}_n) = \frac{1}{N} \sum_{n=1}^{N} \mathcal{N}(\tilde{x}_n | x_n, \sigma^2). \tag{23}$$

By using this smoothed distribution $q_{\text{data}}(\tilde{x}_n)$ instead of the original non-differentiable $p_{\text{data}}(x_n)$, a differentiable objective can be formulated. This allows us to apply gradient-based optimization methods to solve the problem, making the learning of the score function feasible. The closed form of the score function for a Gaussian distribution allows us to compute the gradient of the log-likelihood with respect to the perturbed data $\tilde{x}_n$

$$\begin{aligned} \nabla_{\tilde{x}} \ln \mathcal{N}(\tilde{x}_n | x_n, \sigma^2) &= \nabla_{\tilde{x}} \left(-\ln(2\pi\sigma^D) - \frac{1}{2\sigma^2} (\tilde{x}_n - x_n)^2 \right) \\ &= -\frac{1}{\sigma^2} \left((x_n + \sigma \cdot \epsilon) - x_n \right) \\ &= -\frac{1}{\sigma} \epsilon. \end{aligned} \tag{24}$$

By applying the result of Equation 24 to Equation 22, we can write the differentiable objective as

$$\begin{aligned} \mathcal{L}(\theta) &= \frac{1}{2N} \sum_{n=1}^{N} \int \left\| s_\theta(\tilde{x}) + \frac{1}{\sigma} \epsilon \right\|^2 N(\epsilon | 0, \sigma^2 I) d\epsilon \\ &= \frac{1}{N} \sum_{n=1}^{N} \mathbb{E}_{N(\epsilon|0,I)} \left[\frac{1}{2\sigma^2} \|\epsilon - \tilde{s}_\theta(\tilde{x})\|^2 \right], \end{aligned} \tag{25}$$

where $\tilde{s}_\theta(x) = -\sigma s_\theta(\tilde{x})$. Learning the score model $\tilde{s}_\theta(\tilde{x})$ by optimizing this objective score matching.

2.4.2 Flow Matching Models

Flow Matching [30] is introduced by considering the following ordinary differential equation (ODE)

$$\frac{dx_t}{dt} = v(x_t, t), \tag{26}$$

where $v(x_t, t)$ represents the vector field that defines the dynamics of the system. The vector field governs how the data evolves over time, and by parameterizing it with a neural network $v_\theta(x_t, t)$, the model becomes a Neural ODE [31]. To remain consistent with the Flow Matching literature, the time progresses from noise ($t = 0$) to data ($t = 1$), defining the forward dynamics and the generative process. This differs from diffusion models (and score-based models), where the time direction is reversed, evolving from data to noise. Furthermore, it is assumed that the initial distribution $q_0(x)$, such as a standard Gaussian, is known, and the target data distribution $q_1(x)$ is the distribution that needs to be matched. The goal is to learn a trajectory from the noise distribution $q_0(x)$ to the data distribution $q_1(x)$. Given this, the output is obtained by solving the ODE, i.e., integrating over time t, which transforms the noise into data

$$x_1 = x_0 + \int_0^1 v_\theta(x_t, t)dt. \tag{27}$$

If the vector field $v(x_t, t)$ and the distributions $p_t(x)$ were known, instead of learning a distribution, the objective would be to model the vector field $v_\theta(x_t, t)$. The training approach would involve solving a regression problem, where the objective is to minimize the difference between the model's vector field and the true vector field. This would be done by optimizing the objective

$$\mathcal{L}_{FM}(\theta) = \mathbb{E}_{t\sim U(0,1), x_t\sim p_t(x)} \left[\|v_\theta(x_t, t) - v(x_t, t)\|^2 \right]. \tag{28}$$

Here, for each time t sampled uniformly at random, x_t is sampled from the distribution $p_t(x)$, and the goal is to minimize the difference between the model's vector field $v_\theta(x_t, t)$ and the true vector field $v(x_t, t)$. However, the assumption that both the vector field $v(x_t, t)$ and the distributions $p_t(x)$ are known is unrealistic. To address this issue, a modified approach can be introduced by introducing an additional variable z, which is sampled from a known distribution $q(z)$. The conditional ODE then takes the form

$$\frac{dx_t}{dt} = v(x_t, t; z), \tag{29}$$

where z can represent additional information, such as the data x_1. The conditional flow matching loss is then formulated as

$$\mathcal{L}_{CFM}(\theta) = \mathbb{E}_{t\sim U(0,1), x_t\sim p_t(x|z), z\sim q(z)} \left[\|v_\theta(x_t, t) - v(x_t, t; z)\|^2 \right]. \tag{30}$$

In this setup, the objective is to minimize the difference between the model's vector field $v_\theta(x_t, t)$ and the true conditional vector field $v(x_t, t; z)$, where x_t is sampled from the distribution $p_t(x|z)$ conditioned on z, and z is sampled from the known distribution $q(z)$. One approach to address the conditional formulation is to set z as a data point x_1, making $q(z) = q_1(z)$, where $q_1(z)$ is the data distribution [13]. In this scenario, $z \equiv x_1$ is sampled from the data distribution, $x_1 \sim q_1(x)$. The conditional dynamics are then defined using a mean and a standard deviation function

$$\begin{aligned} \mu(z, t) &= tx_1, \\ \sigma(z, t) &= t\sigma_{\text{const}} - t + 1, \end{aligned} \tag{31}$$

where $\sigma_{\text{const}} > 0$ is a smoothing constant. These functions result in the following conditional probability path and conditional vector field

$$\begin{aligned} p_t(x|z) &= \mathcal{N}\left(x \mid tx_1, (t\sigma_{\text{const}} - t + 1)^2 \mathbf{I}\right), \\ v(x, t; z) &= \frac{x_1 - (1 - \sigma_{\text{const}})x}{1 - (1 - \sigma_{\text{const}})t}. \end{aligned} \tag{32}$$

Here, the conditional probability path $p_t(x|z)$ describes a trajectory starting from the standard Gaussian distribution $p_0(x) = \mathcal{N}(x \mid 0, \mathbf{I})$ and progressing to a Gaussian distribution centered at x_1 with standard deviation σ_{const}, i.e., $p_1(x) = \mathcal{N}(x \mid x_1, \sigma^2_{\text{const}}\mathbf{I})$. The vector field $v(x, t; z)$ is derived analytically using a theorem that ensures the smooth interpolation between the two distributions, aligning the dynamics with the desired conditional trajectory. This approach facilitates learning the conditional vector field while leveraging the data distribution directly.

3 Related Work

The landscape of generative model repositories has seen significant growth over the years, but many existing platforms still fail to bridge the gap between usability and flexibility. There are specialized platforms that focus on aggregating variations of specific types of generative models. Examples include *PyTorch-VAE*[2] and *PyTorch-GANs*[3], which provide comprehensive implementations of various VAE and GAN architectures, respectively. These repositories excel at offering well-documented and modular implementations, allowing researchers and practitioners to experiment with specific model families in depth. However, their specialized nature often limits their scope to a single class of generative models, making them less suited for exploring and benchmarking a broader range of architectures or state-of-the-art techniques.

One major body of work includes educational repositories such as *Deep Generative Modeling*[4] [5] and *Understanding Deep Learning*[5] [32], which cover a broad spectrum of models with toy examples and introductory materials. These repositories provide valuable insight into generative models but are limited to simple examples that do not scale well for real-world applications. Their primary goal is to introduce concepts, not necessarily to offer fully-fledged tools for experimentation or deployment. Another relevant group of repositories focuses on older generative models, including *ModelZoo*[6] and *Generative Models*[7]. These repositories provide a solid foundation for classical image synthesis techniques, but they are limited by their narrow scope and outdated methods. These platforms offer a curated set of models but lack the diversity and modern tools needed for more advanced generative tasks.

A more recent initiative, *MONAI Generative Models* [8] [33], focuses on newer architectures such as VQ-GANs, ARMs, and Diffusion Models, covering state-of-the-art techniques. However, it requires substantial integration effort and is designed around a notebook-first mentality, making it less user-friendly for those who seek a more streamlined, accessible solution for experimentation. Although the repository provides cutting-edge models, it can be challenging to integrate these tools into broader workflows without extensive customization. Despite these advances, there remains a gap for a platform that balances ease of use with the ability to customize and experiment. GenerativeZoo addresses this need by offering a simple, unified interface for modern image generation tasks while maintaining flexibility for deeper exploration. By making model access and evaluation more straightforward, it promotes quick trial and development processes in the creation of synthetic images.

4 Methodology

The GenerativeZoo was designed with three foundational pillars to guide its development: Aggregation, Standardization, and Usability. Aggregation aims to unify a diverse set of generative models within a single platform, fostering a comprehensive exploration of their capabilities. Usability focuses on simplifying tasks such as training, sampling, and evaluating out-of-distribution (OOD) detection, ensuring accessibility for researchers and practitioners. Lastly, standardization establishes a consistent framework for working with these models, promoting ease of use and comparability across different approaches.

Moreover, the GenerativeZoo was developed to cater to a diverse audience. It is particularly appealing to beginners, offering a more playful and pedagogic approach to learning about generative models. Users with little programming experience or limited knowledge of deep learning can engage with the models in a fun and interactive way. At the same time, the platform is also designed to meet the needs of more advanced users and professionals. Its highly customizable and hackable environment allows experienced users to modify, experiment, and adapt the models to suit their specific research or application needs.

4.1 Aggregation

4.1.1 Datasets

The GenerativeZoo provides built-in support for 14 datasets, including 6 grayscale and 9 RGB datasets. Alongside these, the GenerativeZoo includes tools for automatic dataset downloading or detailed instructions for obtaining and

[2] https://github.com/AntixK/PyTorch-VAE
[3] https://github.com/eriklindernoren/PyTorch-GAN
[4] https://github.com/jmtomczak/intro_dgm
[5] https://udlbook.github.io/udlbook/
[6] https://modelzoo.co/model/generative_zoo
[7] https://github.com/wiseodd/generative-models
[8] https://github.com/Project-MONAI/GenerativeModels

preparing datasets manually. While users are free to implement their own dataloaders, these pre-integrated options simplify the process and ensure seamless compatibility with the platform. The list of currently supported datasets is summarized in Table 1.

Table 1: List of datasets available in the GenerativeZoo, categorized by their format. Legend: *Manual download required.

Grayscale Datasets	RGB Datasets
MNIST [34]	CIFAR-10 [35]
FashionMNIST [36]	CIFAR-100 [35]
ChestMNIST++ [37]	SVHN [38]
OctMNIST++ [37]	Places365 [39]
PneumoniaMNIST++ [37]	DTD [40]
TissueMNIST++ [37]	ImageNet-200* [41]
	Horse2Zebra* [42]
	ImageNet-1K [43]
	CelebA [44]

4.1.2 Models

The GenerativeZoo framework currently aggregates a total of 27 models spanning 4 distinct generative model families and 9 different model types, reflecting the breadth of approaches in modern generative modeling. These include Latent Variable Models (VAEs, GANs, and Diffusion Models), Score-based Models (Score Matching and Flow Matching), Autoregressive Models, and Flow-based Models. The implementation of these models draws from 20 different codebases or repositories, ensuring robust and diverse foundations for development. Table 2 provides a comprehensive summary of these models, categorized by family, type, and name.

4.2 Standardization

To ensure standardization across the GenerativeZoo, a set of six main guidelines was established to unify the organization and usability of models:

1. *Self-contained Code:* Each model's implementation is designed to be self-contained, allowing users to understand and use it independently without requiring external resources.
2. *Model Methods:* Training and sampling functionalities are implemented as methods within the model class to ensure a consistent and intuitive structure across all models.
3. *Command Line Control:* Training and evaluation can be executed via command line arguments, enabling flexible and reproducible experimentation.
4. *Full Documentation:* Comprehensive documentation accompanies each model, detailing the purpose and usage of all parameters to assist users in leveraging the models effectively.
5. *Example Script:* An example script is provided with each model to demonstrate its main functionalities, such as training and sampling, offering users practical guidance.
6. *References:* Each model includes references to the original research papers and the repositories that contributed to its implementation, ensuring proper attribution and facilitating reproducibility.

4.3 Usability

The GenerativeZoo is designed to maximize usability by simplifying setup, experiment tracking, and training configurations:

1. *Installation Options:* The Zoo offers multiple installation methods to accommodate user preferences. It includes a `requirements.txt` file, which should be installed within a virtual environment created using `venv`. For Linux users, a Makefile is provided to streamline this process. Detailed setup instructions are provided in the README file. Additionally, the Zoo can be deployed as a Docker container using the provided Dockerfile, which handles the setup of CUDA and the CudaToolkit.

Table 2: List of the models available in the GenerativeZoo, categorized by family and type. The repositories that were used to build the implementations in the GenerativeZoo are also listed.

Model Family	Model Type	Model Name	Base Repositories
Latent Variable	VAE	Vanilla VAE [6]	PyTorch-VAE
		Conditional VAE [45]	PyTorch-VAE
		Hierarchical VAE [18]	NVAE
	GAN	Adversarial VAE [46]	PyTorch-VAE
		DC-GAN [47]	Conditional-GAN
		Conditional GAN [48]	Conditional-GAN
		CycleGAN [42]	PyTorch-GAN
		Prescribed GAN [49]	PresGANs
		Wasserstein GAN [50]	WGAN-GP
	Diffusion Models	DDPM [20]	minDiffusion, DDIM, Guided-Diffusion
		Conditional DDPM [51]	Conditional Diffusion, Guided-Diffusion
		Diffusion AE [52]	Generative Models
	Stable Diffusion	Stable Diffusion + LoRA [53]	Diffusers
		ControlNet [54]	Diffusers
		InstructPix2Pix [55]	Diffusers
Score-based	Score Matching	SGM [12]	SGM Tutorial, Guided-Diffusion
		NCSNv2 [56]	Score SDE PyTorch
	Flow Matching	Flow Matching [13]	Conditional FM, Guided-Diffusion
		Conditional Flow Matching [13]	Conditional FM, Guided-Diffusion
		Rectified Flows [57]	minRF
Autoregressive	Transformer	VQ-VAE + Transformer [58]	Generative Models
		VQ-GAN + Transformer [59]	Generative Models
	PixelCNN	PixelCNN [10]	UVA DLC Notebooks
Flow-based	Normalizing Flows	Vanilla Flow [23]	UVA DLC Notebooks
		RealNVP [8]	RealNVP
		GLOW [9]	Glow-PyTorch
		Flow++ [60]	FlowPlusPlus

2. *Experiment Tracking:* All models in the Zoo integrate experiment tracking using the Weights & Biases platform. Instructions for creating an account and configuring logging are provided, and users can disable logging if preferred.

3. *Accelerate Integration:* To support multi-GPU and mixed precision training, the `Accelerate` library is included, enabling efficient training without requiring users to modify the code.

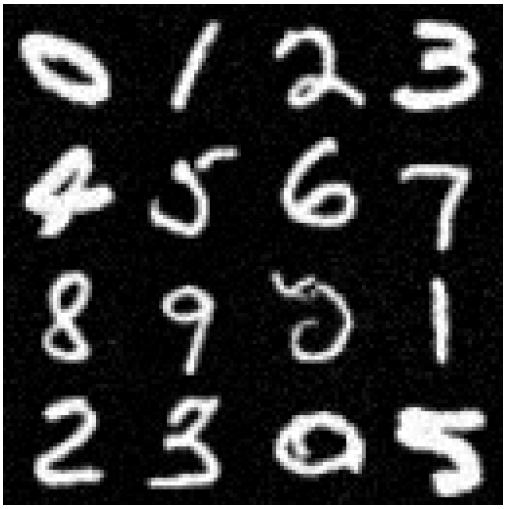
5 Applications

The applications of the GenerativeZoo will expand as new models and use cases are incorporated. While some applications leverage the general capabilities of generative models (e.g. sampling), others are tailored to specific types of generative models, highlighting their unique strengths and specialized functionalities.

5.1 Sampling

The most fundamental capability of the GenerativeZoo is sampling, which involves drawing new samples from the learned distribution of a generative model. Figure 1 illustrates this basic functionality using examples generated from MNIST, a simple grayscale dataset of handwritten digits, demonstrating the foundational capabilities of the GenerativeZoo.

However, as highlighted in Sections 3 and 4.1.1, the GenerativeZoo aims to extend well beyond these basic examples. Figure 2 showcases its capability to handle RGB image generation, ranging from simpler outputs such as those in

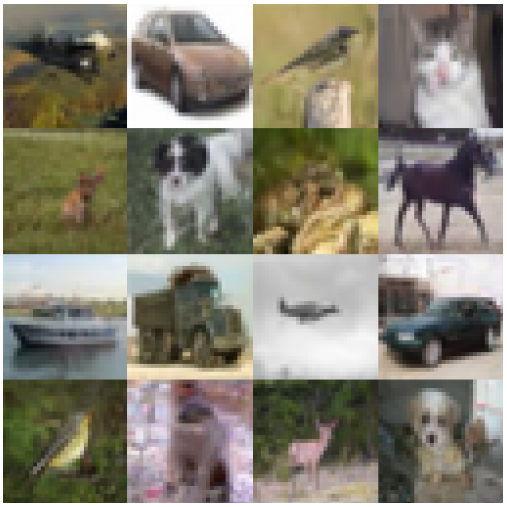


(a) Conditional samples generated by an SGM trained on MNIST.

(b) Unconditional samples generated by a Flow Matching model trained on FashionMNIST.

Figure 1: Grayscale samples generated by models trained using the GenerativeZoo.

Figure 2a, produced by a DDPM trained on CIFAR-10, to more complex, high-resolution images, such as the ones shown in Figure 2b, generated for a custom face dataset.



(a) Conditional samples generated by a DDPM trained on CIFAR-10.

(b) Unconditional samples generated by a DDPM model trained on CelebA.

Figure 2: RGB samples generated by models trained using the GenerativeZoo.

5.2 Out-of-Distribution Detection

OOD detection is carried out in a range of models offered by the GenerativeZoo, employing different methods inherent to each model. These methods encompass evaluating the likelihood of unseen samples, assessing reconstruction error, and employing perceptual loss. For example, Figure 3 demonstrates semantic OOD detection executed by a DDPM trained on the MNIST dataset attempting to identify FashionMNIST samples, drawing inspiration from the DDPM-OOD approach [61]. In this scenario, the anomaly score is formulated as a weighted sum of both the perceptual loss between the original input and its reconstructed version and the mean squared error.

All models in the Zoo that support OOD detection present their results in a standardized format, which includes a histogram of the computed anomaly scores, the Area Under the ROC Curve (AUROC), and the False Positive Rate at

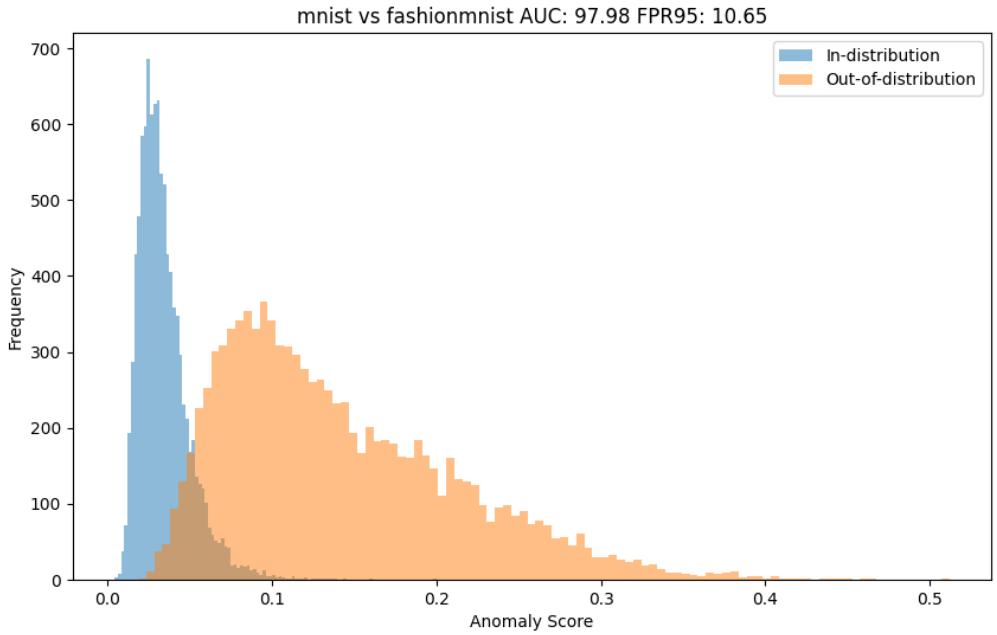


Figure 3: Semantic OOD detection using a DDPM trained on MNIST to identify FashionMNIST samples.

95% recall (FPR95). This approach follows the OpenOOD benchmark [62] ensuring clear comparisons and enables reliable benchmarking across different models.

6 Limitations and Future Work

6.1 Benchmark

We plan to begin benchmarking the models in the GenerativeZoo. This process will be gradual, with the first application to be benchmarked being *Image Generation*, evaluated through the Fréchet Inception Distance (FID). Following this, out-of-distribution detection will be benchmarked for models that fully support it, leveraging the OpenOOD benchmark [62]. An additional goal is to provide reference checkpoints for the models trained for these benchmarks.

6.2 Future Dataset Support

We are working towards supporting the automatic download of the *ImageNet-200* and *Horse2Zebra* datasets in the GenerativeZoo. Furthermore, *Imagenette*[9], a subset of 10 easily classified classes from *ImageNet-1K* (tench, English springer, cassette player, chain saw, church, French horn, garbage truck, gas pump, golf ball, parachute), will be added for fast prototyping on higher resolutions before transitioning to *ImageNet-1K*.

6.3 Future Models

The GenerativeZoo has three defined objectives for future expansion. First, we aim to expand the coverage of Autoregressive models by introducing *PixelCNN++* [63], followed by adding *ZigMa* [64], which will be the first state-space model available in the Zoo. Finally, we intend to complement the current offerings of Diffusion models by incorporating the *Diffusion Transformer* [65].

6.4 Future Applications

Future applications will include a broader coverage of out-of-distribution detection. Moreover, Inpainting and Super-resolution will start being explored, particularly for Diffusion Models and Score-based approaches.

[9] https://github.com/fastai/imagenette

6.5 User Interface

Graphical user interfaces (GUIs) will be considered as demonstrations for some models within the GenerativeZoo. These GUIs will serve as intuitive tools to make certain applications more accessible, allowing users to easily visualize and interact with the generated results. While the core focus of the GenerativeZoo remains on providing a flexible and hackable environment for both beginners and professionals, these visual interfaces will further enrich the usability of the platform for a broader audience.

7 Conclusion

The GenerativeZoo provides a unified and accessible platform to address the fragmentation in the field of generative modeling. Designed to accommodate users ranging from beginners to professionals, it offers a standardized framework and integrated tools for efficient workflows across diverse datasets and applications. The platform incorporates usability-enhancing features such as seamless experiment tracking and support for multi-GPU and mixed-precision training, ensuring a streamlined experience for users. Current functionalities include image generation and out-of-distribution detection, with planned expansions into inpainting, super-resolution, and rigorous benchmarking. By fostering consistency, scalability, and usability, the Zoo bridges the gap between advanced generative modeling research and practical applications, establishing a foundation for continued innovation in machine learning and computer vision.

Acknowledgments

This research was funded by the Xecs Eureka TASTI Project. This work used the Dutch national e-infrastructure with the support of the SURF Cooperative using grant no. EINF-10021.

References

[1] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10684–10695, 2022.

[2] A. Antoniou, "Data augmentation generative adversarial networks," *arXiv preprint arXiv:1711.04340*, 2017.

[3] O. Kara, B. Kurtkaya, H. Yesiltepe, J. M. Rehg, and P. Yanardag, "Rave: Randomized noise shuffling for fast and consistent video editing with diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6507–6516, 2024.

[4] A. Lugmayr, M. Danelljan, A. Romero, F. Yu, R. Timofte, and L. Van Gool, "Repaint: Inpainting using denoising diffusion probabilistic models," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11461–11471, 2022.

[5] J. M. Tomczak, *Deep Generative Modeling*. Springer, 2022.

[6] D. P. Kingma, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[7] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[8] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *arXiv preprint arXiv:1605.08803*, 2016.

[9] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," *Advances in neural information processing systems*, vol. 31, 2018.

[10] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, *et al.*, "Conditional image generation with pixelcnn decoders," *Advances in neural information processing systems*, vol. 29, 2016.

[11] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[12] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, "Score-based generative modeling through stochastic differential equations," *arXiv preprint arXiv:2011.13456*, 2020.

[13] Y. Lipman, R. T. Chen, H. Ben-Hamu, M. Nickel, and M. Le, "Flow matching for generative modeling," *arXiv preprint arXiv:2210.02747*, 2022.

[14] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul, "An introduction to variational methods for graphical models," *Machine learning*, vol. 37, pp. 183–233, 1999.

[15] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling, "Improved variational inference with inverse autoregressive flow," *Advances in neural information processing systems*, vol. 29, 2016.

[16] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther, "Ladder variational autoencoders," *Advances in neural information processing systems*, vol. 29, 2016.

[17] L. Maaløe, M. Fraccaro, V. Liévin, and O. Winther, "Biva: A very deep hierarchy of latent variables for generative modeling," *Advances in neural information processing systems*, vol. 32, 2019.

[18] A. Vahdat and J. Kautz, "Nvae: A deep hierarchical variational autoencoder," *Advances in neural information processing systems*, vol. 33, pp. 19667–19679, 2020.

[19] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International conference on machine learning*, pp. 2256–2265, PMLR, 2015.

[20] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in neural information processing systems*, vol. 33, pp. 6840–6851, 2020.

[21] L. Maaløe, M. Fraccaro, and O. Winther, "Semi-supervised generation with cluster-aware generative models," *arXiv preprint arXiv:1704.00637*, 2017.

[22] J. Song, C. Meng, and S. Ermon, "Denoising diffusion implicit models," *arXiv preprint arXiv:2010.02502*, 2020.

[23] D. Rezende and S. Mohamed, "Variational inference with normalizing flows," in *International conference on machine learning*, pp. 1530–1538, PMLR, 2015.

[24] A. Graves and A. Graves, "Long short-term memory," *Supervised sequence labelling with recurrent neural networks*, pp. 37–45, 2012.

[25] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[26] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.

[27] M. Welling and Y. W. Teh, "Bayesian learning via stochastic gradient langevin dynamics," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 681–688, Citeseer, 2011.

[28] A. Hyvärinen and P. Dayan, "Estimation of non-normalized statistical models by score matching.," *Journal of Machine Learning Research*, vol. 6, no. 4, 2005.

[29] A. Hyvärinen, "Some extensions of score matching," *Computational statistics & data analysis*, vol. 51, no. 5, pp. 2499–2512, 2007.

[30] Y. Lipman, M. Havasi, P. Holderrieth, N. Shaul, M. Le, B. Karrer, R. T. Chen, D. Lopez-Paz, H. Ben-Hamu, and I. Gat, "Flow matching guide and code," *arXiv preprint arXiv:2412.06264*, 2024.

[31] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, "Neural ordinary differential equations," *Advances in neural information processing systems*, vol. 31, 2018.

[32] S. J. Prince, *Understanding deep learning*. MIT press, 2023.

[33] W. H. Pinaya, M. S. Graham, E. Kerfoot, P.-D. Tudosiu, J. Dafflon, V. Fernandez, P. Sanchez, J. Wolleb, P. F. Da Costa, A. Patel, *et al.*, "Generative ai for medical imaging: extending the monai framework," *arXiv preprint arXiv:2307.15208*, 2023.

[34] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[35] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[36] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[37] J. Yang, R. Shi, D. Wei, Z. Liu, L. Zhao, B. Ke, H. Pfister, and B. Ni, "Medmnist v2-a large-scale lightweight benchmark for 2d and 3d biomedical image classification," *Scientific Data*, vol. 10, no. 1, p. 41, 2023.

[38] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet, "Multi-digit number recognition from street view imagery using deep convolutional neural networks," *arXiv preprint arXiv:1312.6082*, 2013.

[39] A. López-Cifuentes, M. Escudero-Vinolo, J. Bescós, and Á. García-Martín, "Semantic-aware scene recognition," *Pattern Recognition*, vol. 102, p. 107256, 2020.

[40] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, "Describing textures in the wild," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3606–3613, 2014.

[41] Y. Le and X. Yang, "Tiny imagenet visual recognition challenge," *CS 231N*, vol. 7, no. 7, p. 3, 2015.

[42] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 2223–2232, 2017.

[43] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, pp. 211–252, 2015.

[44] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[45] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," *Advances in neural information processing systems*, vol. 28, 2015.

[46] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644*, 2015.

[47] A. Radford, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[48] M. Mirza, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[49] A. B. Dieng, F. J. Ruiz, D. M. Blei, and M. K. Titsias, "Prescribed generative adversarial networks," *arXiv preprint arXiv:1910.04302*, 2019.

[50] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," *Advances in neural information processing systems*, vol. 30, 2017.

[51] J. Ho and T. Salimans, "Classifier-free diffusion guidance," *arXiv preprint arXiv:2207.12598*, 2022.

[52] K. Preechakul, N. Chatthee, S. Wizadwongsa, and S. Suwajanakorn, "Diffusion autoencoders: Toward a meaningful and decodable representation," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10619–10629, 2022.

[53] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," *arXiv preprint arXiv:2106.09685*, 2021.

[54] L. Zhang, A. Rao, and M. Agrawala, "Adding conditional control to text-to-image diffusion models," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3836–3847, 2023.

[55] T. Brooks, A. Holynski, and A. A. Efros, "Instructpix2pix: Learning to follow image editing instructions," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 18392–18402, 2023.

[56] Y. Song and S. Ermon, "Improved techniques for training score-based generative models," *Advances in neural information processing systems*, vol. 33, pp. 12438–12448, 2020.

[57] P. Esser, S. Kulal, A. Blattmann, R. Entezari, J. Müller, H. Saini, Y. Levi, D. Lorenz, A. Sauer, F. Boesel, *et al.*, "Scaling rectified flow transformers for high-resolution image synthesis," in *Forty-first International Conference on Machine Learning*, 2024.

[58] A. Van Den Oord, O. Vinyals, *et al.*, "Neural discrete representation learning," *Advances in neural information processing systems*, vol. 30, 2017.

[59] P. Esser, R. Rombach, and B. Ommer, "Taming transformers for high-resolution image synthesis," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12873–12883, 2021.

[60] J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel, "Flow++: Improving flow-based generative models with variational dequantization and architecture design," in *International conference on machine learning*, pp. 2722–2730, PMLR, 2019.

[61] M. S. Graham, W. H. Pinaya, P.-D. Tudosiu, P. Nachev, S. Ourselin, and J. Cardoso, "Denoising diffusion models for out-of-distribution detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2948–2957, 2023.

[62] J. Yang, P. Wang, D. Zou, Z. Zhou, K. Ding, W. Peng, H. Wang, G. Chen, B. Li, Y. Sun, *et al.*, "Openood: Benchmarking generalized out-of-distribution detection," *Advances in Neural Information Processing Systems*, vol. 35, pp. 32598–32611, 2022.

[63] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications," *arXiv preprint arXiv:1701.05517*, 2017.

[64] V. T. Hu, S. A. Baumann, M. Gui, O. Grebenkova, P. Ma, J. Fischer, and B. Ommer, "Zigma: A dit-style zigzag mamba diffusion model," in *European Conference on Computer Vision*, pp. 148–166, Springer, 2025.

[65] W. Peebles and S. Xie, "Scalable diffusion models with transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4195–4205, 2023.