

Generative AI

What is Generative AI?

Generative AI is a subset of machine learning focused on generating new data samples that resemble a given dataset by learning its underlying probability distribution. This means the model tries to approximate the statistical patterns and structure of the data it is trained on. In simple terms, if a dataset is represented as a set of samples $X = \{x_1, x_2, \dots, x_n\}$, the goal is to model the probability distribution $p(x)$ that could have produced these samples.

For example, if you have a dataset of human faces, the model learns $p(x)$, where x represents the pixel values of an image of a face. After training, the model can generate new face images $x' \sim p(x)$, which are not exact copies of the training data but look **statistically similar**, as they follow the same underlying patterns.

Generative AI is not inherently a supervised learning task. It typically falls under **unsupervised learning** or **self-supervised learning** because the model does not rely on labeled data (input-output pairs). Unlike supervised learning, which predicts a label y , generative AI focuses on understanding and replicating the "rules" or "patterns" within the data itself.

- **Unsupervised Learning:** The model learns directly from data without explicit labels.
- **Self-Supervised Learning:** Some generative models, like diffusion models, use labels derived from the data itself (e.g., "noise levels" in diffusion processes), allowing them to learn more structured representations.

However, certain generative models can operate in **semi-supervised** or **supervised contexts**. For instance, **conditional GANs (cGANs)** generate data conditioned on a specific label y , effectively blending generative and supervised approaches by incorporating label information into the generation process.

Apart from Diffusion Models and cGANs, other notable examples of Generative AI models include Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs) in general, Energy-Based Models (EBMs), and Transformer-based Generative Models, among others. While there is no single unified approach that all Generative AI models use to achieve the goal of learning the underlying probability distribution $p(x)$, they generally share the following characteristics:

- a) They are probabilistic in nature because they involve modeling and sampling from a probability distribution $p(x)$, either explicitly or implicitly.
- b) Many generative models rely on the concept of a latent variable z , which represents an unobservable, low-dimensional abstract space. This latent space does not necessarily correspond to any physical or real-world input. Depending on the specific approach, z can either be a randomly sampled variable (as in models like GANs) or a representation extracted from input data (as in models like VAEs).
- c) Some generative models, such as conditional models (e.g., conditional GANs or conditional VAEs), incorporate additional inputs beyond the latent space, often representing real or physical variables (e.g., class labels, physical parameters, or external constraints). These models generate data conditioned on these inputs, enabling more targeted and context-specific generation.

Why is Generative AI Important in Environmental Data Analytics (EDA)?

Generative AI is critical in EDA because it addresses challenges like data sparsity, missing information, and measurement noise, which are common in environmental datasets. For instance, data collected from LiDAR, satellite imagery, or field sensors may have gaps due to occlusions, equipment limitations, or accessibility issues. Generative models can ***synthesize realistic missing data $x' \sim p(x)$ or augment datasets***, enabling better model training and analyses.

Example of Gen AI use cases in EDA include:

- **Imputation:** Filling gaps in time-series or spatial datasets.
- **Simulation:** Generating potential environmental scenarios (e.g., climate changes, deforestation patterns).
- **Data Augmentation:** Creating diverse training samples to improve machine learning model robustness.

By modeling multidimensional distributions, generative AI helps enhance the quality of environmental analysis and prediction ***under limited data conditions***.

While Generative AI is relevant to nearly all types of data in EDA, such as images, time-series, and spatial grids, this discussion will focus specifically on **point cloud data** as a special case. Point clouds pose unique challenges due to their irregular, unordered structure, making them a compelling domain for exploring the capabilities and applications of Generative AI.

The same ideas that enable applying ML models to point cloud data—accounting for their specific characteristics such as being unordered, irregular, and unstructured—are also relevant to Generative AI models. Generative AI for point clouds must similarly adapt to these properties, ensuring that the data's geometric and spatial integrity is preserved during generation. The following provide an overview of key Generative AI models that are particularly applicable to point cloud data.

Variational Autoencoders (VAEs)

VAEs are a type of Generative AI model that rely on an encoder-decoder architecture and explicitly estimate the probability density of the data $p(x)$ through a latent variable model.

The encoder learns the latent distribution z given the data x , which corresponds to the posterior distribution $p(z | x)$. According to Bayes' theorem:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}$$

Before training, $p(z)$ is the **prior distribution** over the latent variable z . It is assumed to be a simple, predefined distribution, commonly a standard Gaussian $N(0, I)$. This prior reflects our assumption about z 's distribution before observing any data x . Computing $p(z | x)$ exactly is intractable because it depends on $p(x)$, where:

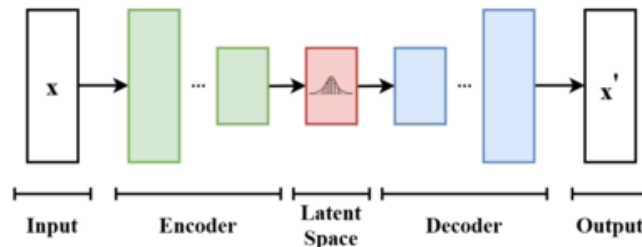
$$p(x) = \int p(x|z)p(z) dz$$

This integral is hard to compute because it requires summing over all possible z . VAEs approximate $p(z | x)$ with a simpler, learnable distribution $q(z | x)$, often modeled as a Gaussian parameterized by the encoder's outputs (mean and variance). The encoder in VAEs learns the parameters of $q(z | x)$ (often its mean and variance in the case of a Gaussian distribution) by mapping the input x to these parameters through a neural network. The parameters of this mapping are learned by optimizing the **Evidence Lower Bound (ELBO)**, which serves as the objective function for both the encoder and decoder.

The **decoder** in VAEs maps the latent variable z to $p(x | z)$, which represents the likelihood of reconstructing x from z . $p(x | z)$ is a conditional distribution that defines how x is generated given z . Just like the encoder models $q(z | x)$, the decoder models $p(x | z)$ as a simple distribution, such as a Gaussian or Bernoulli, depending on the type

of data x (e.g., continuous or binary). The decoder neural network takes z as input and outputs the parameters of the distribution $p(x | z)$, such as the mean and variance (for continuous data) or probabilities (for binary data). The decoder also contributes to optimizing the ELBO loss function, as the ELBO depends on both the encoder ($q(z | x)$) and the decoder ($p(x | z)$). Thus, the encoder and decoder are trained together using the ELBO as the loss function.

After training, the encoder is no longer required for generation. During generation, z is sampled directly from the prior distribution $p(z)$ (not $q(z | x)$), and the decoder maps this sampled z to $p(x | z)$, which generates new data x .



Schematic overview of the Variational Autoencoders (VAEs)

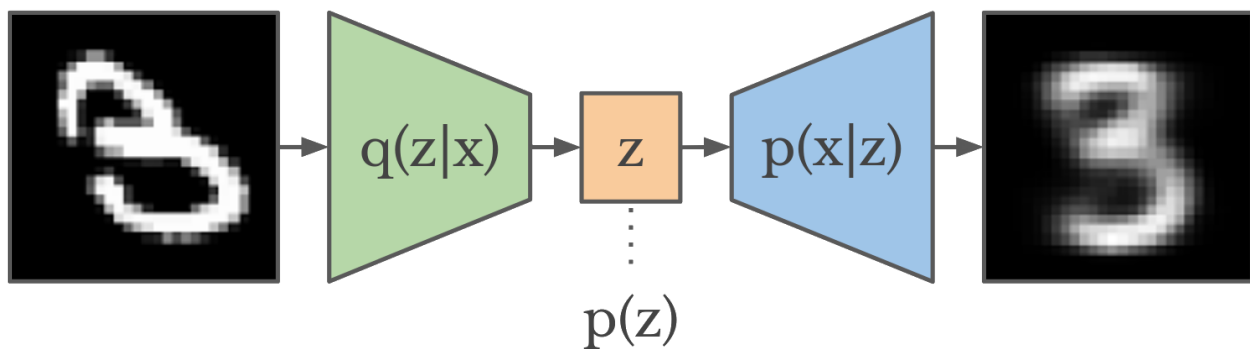
In the explanation of VAEs, the Evidence Lower Bound (ELBO) was mentioned multiple times. The ELBO consists of two key terms:

1. **KL Divergence Term:** This term is primarily relevant to the **encoder**, as it ensures that the learned posterior distribution $q(z | x)$ (approximated by the encoder) stays close to the prior distribution $p(z)$. By minimizing this term, the model regularizes the latent space to match the prior, preventing overfitting and encouraging smooth and organized latent representations.
2. **Reconstruction Term:** This term is primarily relevant to the **decoder**, as it measures how well the model reconstructs the input x from the latent variable z . By maximizing this term, the model learns to generate realistic reconstructions of x given z , ensuring that z captures meaningful information about x .

How VAEs Adapt to Point Cloud Data

The concept of VAEs can and has been applied to point cloud data by many. One popular architecture is the Point-VAE. Point-VAE applies the same general concepts and uses the same Evidence Lower Bound (ELBO) loss function but adapts the architecture and implementation to handle the unique properties of point clouds. Instead of using standard neural networks, Point-VAE typically employs architectures like PointNet or PointNet++ for the encoder, which are designed to process unordered and irregular

point cloud data. The decoder is designed to map the latent representation z back to a reconstructed point cloud. In addition to the ELBO loss, some implementations use specialized point cloud-specific loss functions, such as Chamfer Distance or Earth Mover's Distance (EMD), to ensure that the generated point cloud closely resembles the input cloud in terms of spatial structure. These specialized losses are often used in a **multi-loss framework** to enhance reconstruction quality.



Role of the encoder and decode in Variational Autoencoders (VAEs)

Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) are a type of **generative AI model** that use a game-theoretic approach involving two neural networks:

1. **Generator (G):**

- Takes random noise z sampled from a prior distribution $p(z)$ (e.g., Gaussian) and generates data x' that resembles the real data. The generator learns to map z to the data distribution $p(x)$.

2. **Discriminator (D):**

- Takes as input both real data x and generated data x' , and outputs the probability that the input is real (i.e., from the true data distribution) rather than fake (i.e., generated by G). The discriminator learns to distinguish between real and generated data.

The generator and discriminator compete in a **minimax game**:

- The **discriminator D** tries to correctly identify whether the data it sees is real or fake:
 - For real data x : $D(x) \rightarrow 1$ (i.e., classify it as real).

- For fake data $G(z): D(G(z)) \rightarrow 0$ (i.e., classify it as fake).
- The **generator G** tries to fool the discriminator:
 - For fake data $G(z): D(G(z)) \rightarrow 1$ (i.e., make it appear real).
 - To implement the above concept, GANs use the following adversarial loss function:

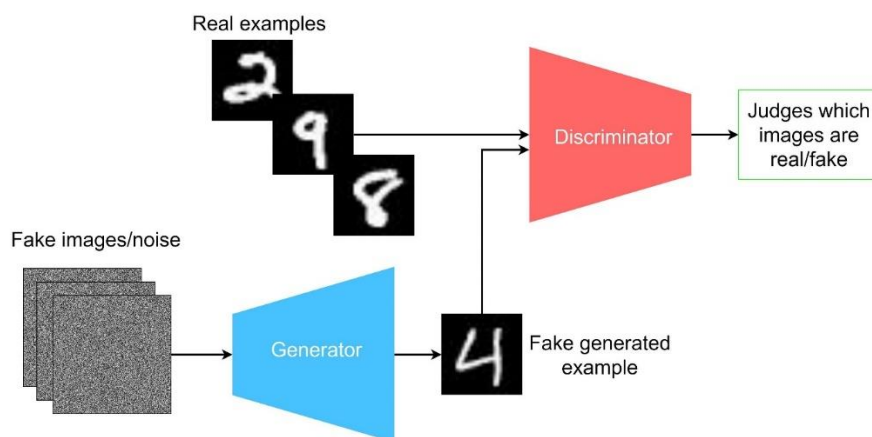
$$\min_G \max_D \mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{z \sim p(z)} [\log(1 - D(G(z)))].$$

The first term ($\log D(x)$) rewards the discriminator when it correctly classifies real data x as real ($D(x) \rightarrow 1$). The Second Term ($\log(1 - D(G(z)))$) rewards the discriminator when it correctly classifies fake data $G(z)$ as fake ($D(G(z)) \rightarrow 0$).

While both the generator G and discriminator D are trained using this adversarial loss, their objectives are distinct:

- The **discriminator (D)** tries to **maximize** the adversarial loss, improving its ability to distinguish real data from fake data.
- The **generator (G)** tries to **minimize** the adversarial loss, attempting to fool the discriminator into classifying fake data as real.

The optimal value of the adversarial loss is achieved when the generator produces data that is indistinguishable from real data, making the discriminator's output $D(G(z))$ equal to 0.5 for all samples. Ideally, the adversarial loss stabilizes at around $-2\log 2$ when the GAN reaches equilibrium. This reflects the discriminator's inability to distinguish real from fake data and the generator's success in fooling the discriminator, resulting in the discriminator outputting 0.5 for both real and fake data.



Schematic overview of Generative Adversarial Networks (GANs)

How GANs Adapt to Point Cloud Data

Applying GANs to point clouds requires specific adaptations to handle their unique properties, such as being **unordered**, **irregular**, and **sparse**. These adaptations impact both the **generator** and the **discriminator**.

1. Generator:

- The generator must convert a latent vector z , sampled from a prior distribution (e.g., Gaussian), into a realistic point cloud.
- Architectures such as **PointNet++ decoders** or fully connected networks reshaped into 3D points are common choices for the generator.

2. Discriminator:

- The discriminator evaluates whether a point cloud is real (X) or generated (X').
- Since point clouds are unordered, the discriminator typically employs architectures like **PointNet** or **PointNet++**, which are designed to extract features invariant to point order.

3. Loss Functions:

- In addition to the standard adversarial loss, **point cloud-specific loss functions** are often incorporated in a multi-loss framework to enhance the quality of generated data. Examples include:
 - **Chamfer Distance:** Measures the average distance between points in two point clouds, ensuring that the generated cloud aligns spatially with the real cloud.
 - **Earth Mover's Distance (EMD):** Measures the minimal cost of transforming one point cloud into another, ensuring precise spatial matching.

Diffusion models

Diffusion models are a class of generative models that produce data by iteratively denoising a noisy signal. They rely on the following key concepts:

1) Forward Process (Diffusion): The forward process is a **mathematical procedure** that progressively adds noise to the data x_0 over a series of time steps t , producing a sequence of increasingly noisy data x_t , until the data becomes nearly indistinguishable from random noise. The forward process **does not involve any neural networks or learning**. It is entirely predefined, governed by a **noise schedule** (e.g., Gaussian noise with increasing variance over time).

2) Reverse Process (Denoising): The reverse process is the **learnable component** of diffusion models. It involves a neural network that learns to reverse the forward process by gradually removing noise from x_t to reconstruct the original data x_0 .

A neural network $\epsilon_\theta(x_t, t)$ is trained to predict the noise ϵ added at each step t . By accurately predicting ϵ , the network iteratively removes noise to reconstruct the original data x_0 .

The model is trained to minimize the difference between the actual noise added during the forward process and the noise predicted by the denoising network:

$$\mathcal{L} = \mathbb{E}_{x_t, t, \epsilon} [\|\epsilon - \epsilon_\theta(x_t, t)\|^2]$$

Denoising diffusion models

● Forward / noising process



● Reverse / denoising process

○ Sample noise $p_T(x_T) \rightarrow$ turn into data

Schematic overview of Diffusion Models Work.

How Diffusion Models Adapt to Point Cloud Data

To apply diffusion models to **point clouds**, the architecture and loss functions are adapted to handle the unique properties of point clouds.

The denoising network must be able to process point clouds effectively. Common architectures include:

- **PointNet** or **PointNet++**, which handle unordered data and extract global features.
- Graph-based networks (e.g., Dynamic Graph CNNs) to capture relationships between points.

A multi-loss framework is often used, combining:

-
1. **Noise Prediction Loss:** Ensures the model can accurately predict and remove noise.
 2. **Chamfer Distance or EMD:** Ensures the generated point cloud resembles the real data spatially.
 3. **Regularization Terms:** To ensure a smooth and consistent latent space or handle additional features.