CSE 403: Machine Learning

Chapter 8: Generative Models

Md Atikuzzaman
Lecturer
Department of Computer Science & Engineering
atik@cse.green.edu.bd

GREEN UNIVERSITY OF BANGLADESH

Outline

What Are Generative Models?

- Generative models try to learn how the data itself is distributed.
- Instead of only predicting labels, they try to model how samples are produced.
- Once trained, they can often **generate new data** that resembles the training data.

Core goal

Learn a probability distribution such as

$$p(x) \quad \text{or} \quad p(x, y)$$

A generative model tries to answer: **how could this data have been generated?**

Why Are Generative Models Important?

- They can synthesize new images, text, audio, or tabular data
- They can learn compact and useful representations
- They can support unsupervised or semi-supervised learning
- They are useful when labeled data is limited

Typical motivations

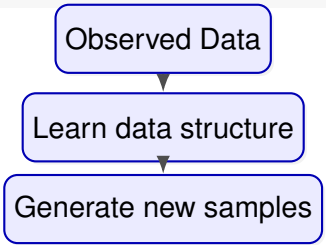
- realistic image generation
- missing data imputation
- anomaly detection
- data augmentation
- creative AI systems

Generative vs Discriminative Models

Generative Model

Learns how data is produced

$$p(x), \quad p(x, y), \quad \text{or } p(x \mid y)$$

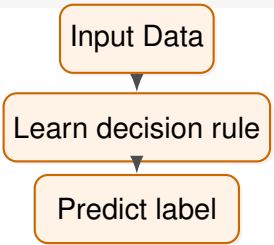


Question: How could this data arise?

Discriminative Model

Learns how to predict labels

$$p(y \mid x)$$



Question: Which label fits this input?

Generative models try to understand the **data distribution**. Discriminative models try to learn the **decision boundary**.

Generative vs Discriminative Models

Generative Models

- model how data is produced
- learn $p(x)$ or $p(x, y)$
- can often generate samples

Discriminative Models

- focus on decision boundaries
- learn $p(y \mid x)$ or direct mapping $x \to y$
- mainly used for prediction

Discriminative models answer **which class?** Generative models answer **how was this produced?**

Intuitive Comparison

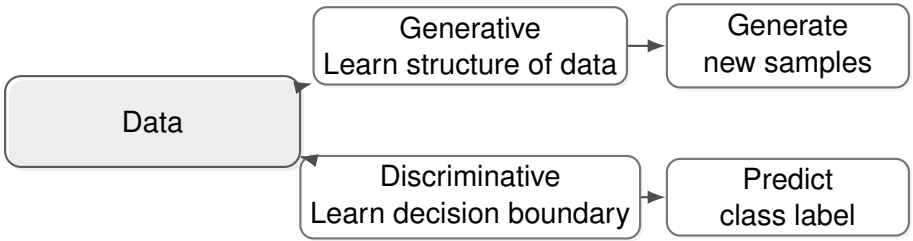
Example: Cat vs Dog Classification

- A discriminative model learns how to separate cats from dogs
- A generative model learns what cat images and dog images look like

Interpretation

- discriminative: $p(y \mid x)$
- generative: $p(x \mid y)$ and possibly $p(y)$, or directly $p(x)$

Visual Comparison: Two Views of Learning



Basic Probability Concepts

Probability distribution

A probability distribution tells us how likely different outcomes are.

Random variables

- x: observed data
- y: class label or latent factor
- z: hidden or latent variable

- Generative modeling relies heavily on probability and uncertainty

Joint, Marginal, and Conditional Probability

Joint Probability:
Probability of observing x and y together

$$p(x, y)$$

Conditional Probability:
Probability of x given y

$$p(x \mid y) = \frac{p(x, y)}{p(y)}$$

Marginal Probability:
Probability of x regardless of other variables

$$p(x) = \sum_{y} p(x, y) \quad \text{or} \quad p(x) = \int p(x, z)\, dz$$

Why Latent Variables Matter

Latent variable

A latent variable is not directly observed, but it explains variation in the data.

- In image generation, a latent code may represent style, pose, or shape
- In text generation, it may capture topic or semantic structure

Generative view

$$z \sim p(z) \qquad \rightarrow \qquad x \sim p(x \mid z)$$

Many modern generative models create data by first sampling a hidden representation z.

Simple Generative Models

- Before deep generative models, classical probabilistic models were widely used
- They are still important for intuition and foundations

Examples

- Gaussian models
- Naive Bayes

Gaussian Generative Models

Idea

Assume the data comes from a Gaussian distribution:

$$x \sim \mathcal{N}(\mu, \Sigma)$$

- μ: mean
- Σ: covariance

Use

Useful for modeling continuous data with a simple probabilistic form.

Naive Bayes as a Simple Generative Model

Core Assumption:
Features are conditionally independent given the class

$$p(x \mid y) = \prod_{j=1}^{d} p(x_j \mid y)$$

Prediction Rule:

$$p(y \mid x) \propto p(x \mid y)\, p(y)$$

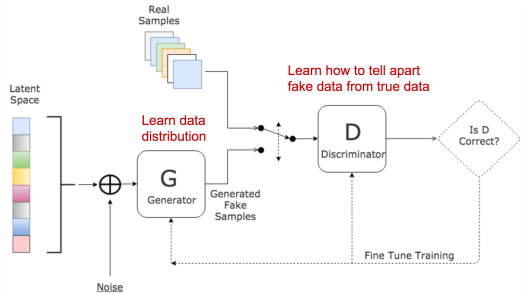
Key Points:

- Easy to train
- Effective for text classification
- A classic generative classifier

GAN: Main Idea

Two networks compete

- **Generator** G: maps noise $z \sim p_z$ to fake samples $G(z)$
- **Discriminator** D: estimates whether a sample is real or generated



GAN training is a two-player game: G learns to fool D, while D learns to detect generated samples.

GAN Objective and Optimal Discriminator

Minimax objective

$$\min_G \max_D \; \mathbb{E}_{x \sim p_r}[\log D(x)] + \mathbb{E}_{z \sim p_z}[\log(1 - D(G(z)))]$$

If G is fixed, the optimal discriminator is

$$D^*(x) = \frac{p_r(x)}{p_r(x) + p_g(x)}.$$

- At the ideal solution, $p_g = p_r$.
- Then $D^*(x) = \frac{1}{2}$: the discriminator cannot tell real and fake apart.

What Does GAN Training Minimize?

Connection to Jensen–Shannon divergence

When the discriminator is optimal,

$$L(G, D^*) = 2D_{\mathrm{JS}}(p_r \,\|\, p_g) - 2\log 2.$$

- GANs try to make the generated distribution p_g match the real data distribution p_r.
- The original objective can suffer when the two distributions have little overlap.

Common practical generator loss

$$\max_G \; \mathbb{E}_{z\sim p_z}[\log D(G(z))]$$

This non-saturating version often gives stronger gradients early in training.

Common GAN Challenges

- **Training instability**: simultaneous updates may oscillate rather than converge.
- **Vanishing gradients**: a near-perfect discriminator can give weak learning signal to G.
- **Mode collapse**: the generator produces limited varieties of samples.
- **Evaluation difficulty**: visual quality, diversity, and coverage are not captured by one simple loss.

GANs can generate sharp samples, but the adversarial game makes optimization delicate.

Improving GAN Training: WGAN Intuition

Motivation

The original GAN objective is closely related to JensenShannon (JS) divergence, which may be uninformative when real and generated distributions have disjoint support.

Wasserstein GAN idea

Use the Wasserstein distance, also called Earth Mover's distance, as a smoother measure of distribution mismatch.

- Replace discriminator with a critic that scores samples.
- Enforce a Lipschitz constraint on the critic.
- The loss is often more correlated with sample quality during training.

For more reading: https://lilianweng.github.io/posts/2017-08-20-gan/

Autoencoders: Concept

Main Idea

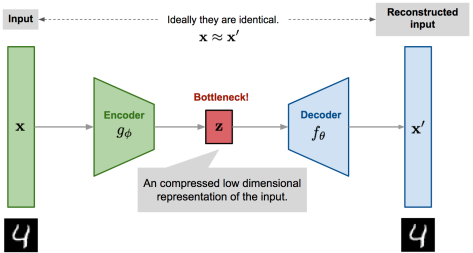
An autoencoder is a neural network designed to learn an identity function in an unsupervised way to reconstruct the original input, while compressing the data through a narrow bottleneck layer to discover an efficient representation.

- **Encoder (g_ϕ)**: Translates high-dimensional input x into a latent, low-dimensional code $z = g_\phi(x)$.
- **Decoder (f_θ)**: Recovers the data from the code to output a reconstruction $x' = f_\theta(z)$.

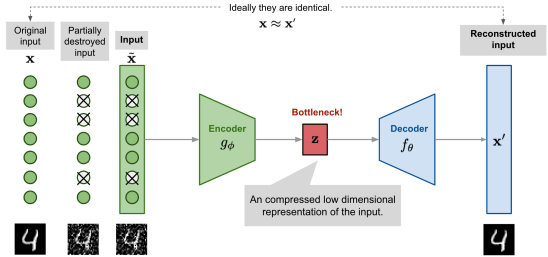
Dimensionality Reduction

Like Principal Component Analysis (PCA), it achieves dimensionality reduction, but utilizes non-linear transformations to capture complex latent variables.

Basic Autoencoder Architecture



(a) Autoencoder model architecture



(b) Denoising autoencoder model architecture

Training Objective:
The parameters (θ, φ) are learned jointly to minimize reconstruction error:

$$L_{AE}(\theta, \varphi) = \frac{1}{n} \sum_{i=1}^{n} \left(x^{(i)} - f_\theta(g_\varphi(x^{(i)}))\right)^2$$

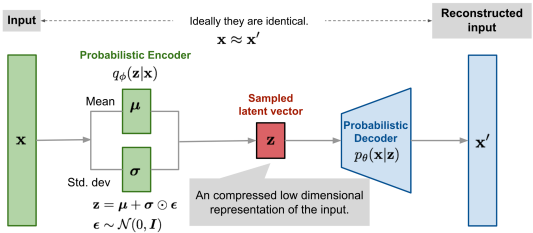
Limitations of Standard Autoencoders

- **Overfitting:** If there are too many network parameters, it risks memorizing the data instead of learning useful features.
- **Lack of Generative Capability:** A standard autoencoder maps inputs to fixed, discrete vectors. The latent space is not continuous, meaning random points sampled from it will not decode into meaningful data.

To solve this, Variational Autoencoders (VAEs) transition from deterministic mappings to a probabilistic graphical model.

VAE: A Probabilistic Model

- Instead of mapping the input to a fixed vector, VAEs map it to a *distribution*.
- **Probabilistic Encoder:** $q_\varphi(z|x)$ approximates the intractable true posterior $p_\theta(z|x)$.
- **Probabilistic Decoder:** $p_\theta(x|z)$ calculates the likelihood of generating true data given the latent code.



Generative Process

To generate a sample resembling real data:

1. Sample a latent variable from a prior: $z \sim p_\theta(z)$ (often $\mathcal{N}(0, I)$).
2. Generate the data point from the conditional distribution: $x \sim p_\theta(x|z)$.

VAE Objective: ELBO

Evidence Lower Bound (ELBO)

Since the exact data likelihood $p_\theta(x)$ is intractable, we optimize its lower bound:

$$\log p_\theta(x) \geq \underbrace{\mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)]}_{\text{reconstruction likelihood}} - \underbrace{D_{KL}(q_\phi(z|x) \,\|\, p_\theta(z))}_{\text{KL divergence penalty}}$$

- **Reconstruction Likelihood:** Encourages the decoder to accurately reconstruct the input from the sampled latent variable.
- **KL Divergence:** Acts as a regularizer, squeezing the approximate posterior $q_\phi(z|x)$ to match the prior $p_\theta(z)$ (usually standard normal).

The Reparameterization Trick

The Problem

We cannot compute gradients for backpropagation through a stochastic sampling process $z \sim q_\varphi(z|x)$.

The Solution

Decouple the randomness from the network parameters. Instead of sampling z directly, we sample a random noise variable ϵ and apply a deterministic transformation:

$$\epsilon \sim \mathcal{N}(0, I), \qquad z = \mu_\varphi(x) + \sigma_\varphi(x) \odot \epsilon$$

- The randomness is now contained entirely in the external ϵ.
- The network outputs $\mu_\varphi(x)$ and $\sigma_\varphi(x)$, allowing gradients to backpropagate cleanly through the deterministic nodes.

Beyond VAE: β-VAE

Lilian Weng's review culminates in extensions like β-VAE, which adjusts the ELBO formulation to prioritize representation disentanglement.

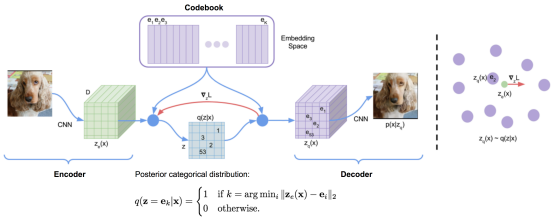
β-VAE Objective

$$\mathcal{L}_{BETA}(\varphi, \theta) = -\mathbb{E}_{z \sim q_\varphi(z|x)}[\log p_\theta(x|z)] + \beta D_{KL}\big(q_\varphi(z|x) \,\|\, p_\theta(z)\big)$$

- **Standard VAE:** $\beta = 1$.
- **Disentanglement ($\beta > 1$):** Applying a stronger penalty on the KL divergence limits the representation capacity of z.
- **Result:** It forces the network to learn conditionally independent, disentangled generative factors, at the cost of a slight trade-off in reconstruction fidelity.

VQ-VAE: Discrete Latent Representation

- Instead of a continuous distribution, VQ-VAE uses a *discrete codebook*.
- **Encoder:** maps input to latent vectors $z_e(x)$.
- **Vector Quantization:** replaces $z_e(x)$ with nearest codebook entry $z_q(x)$.
- **Decoder:** reconstructs input from quantized latent $z_q(x)$.



Generative Process:

1. Select discrete latent code: $z_q \in \mathcal{Z}$ (codebook)
2. Generate data: $x \sim p_\theta(x \mid z_q)$

For more reading: https://lilianweng.github.io/posts/2018-08-12-vae/

Why Flow-Based Models?

- GANs can produce sharp samples but do not directly give exact likelihood.
- VAEs optimize a lower bound rather than exact likelihood.
- Flow-based models explicitly learn a tractable density $p(x)$.

Core idea

Transform a simple base variable into data using an invertible function:

$$z \sim p_Z(z), \qquad x = f_\theta(z), \qquad z = f_\theta^{-1}(x).$$

Flows trade architectural flexibility for exact density evaluation and exact latent inversion.

For more reading: https://lilianweng.github.io/posts/2018-10-13-flow-models/

Change of Variables

Density under an invertible transformation

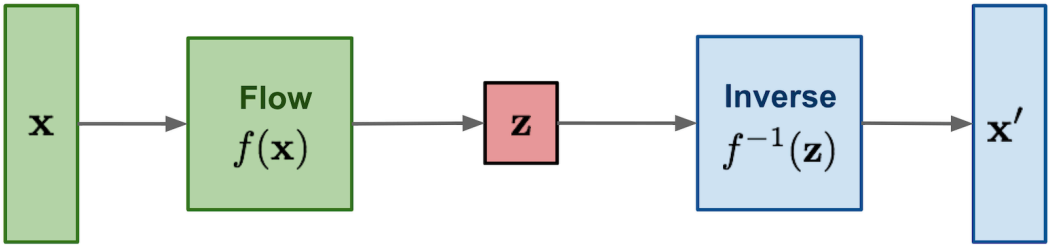
$$p_X(x) = p_Z\left(f_\theta^{-1}(x)\right) \left| \det \frac{\partial f_\theta^{-1}(x)}{\partial x} \right|.$$

Training objective

$$\max_\theta \sum_{i=1}^{n} \log p_X(x^{(i)})$$

- The determinant term accounts for how the transformation expands or contracts volume.
- Architectures must make inversion and determinant computation efficient.

Normalizing Flow Pipeline



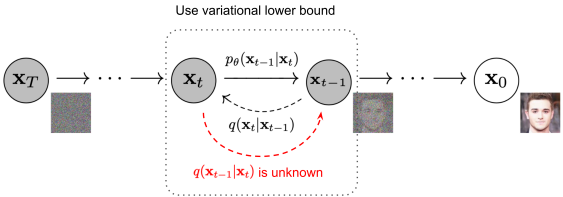
Examples

RealNVP, NICE, Glow, Masked Autoregressive Flow, Inverse Autoregressive Flow.

Typical tradeoff

Exact likelihood and reversible mapping, but constrained model design and potentially expensive computation.

Diffusion Models



Core idea

Learn to reverse a gradual noising process.

Forward process

$$x_0 \to x_1 \to \cdots \to x_T$$

Add Gaussian noise step by step until the sample becomes nearly pure noise.

Reverse process

$$x_T \to x_{T-1} \to \cdots \to x_0$$

Learn a neural network that removes noise step by step.

Generation begins with noise and repeatedly denoises it into a data-like sample.

Diffusion Models: Forward & Reverse Process

Forward Diffusion (Noising)
Gradually adds Gaussian noise to data:

$$q(x_t \mid x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

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

Closed-form sampling:

$$q(x_t \mid x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$$

where $\alpha_t = 1 - \beta_t$, $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$

Transforms data into pure noise

Reverse Diffusion (Denoising)
Learns to recover data from noise:

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

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

In practice, model predicts noise:

$$\epsilon_\theta(x_t, t)$$

Gradually converts noise into realistic data

Diffusion Model: Training and Sampling

Algorithm 1: Training

1. **repeat**
2. $x_0 \sim q(x_0)$
3. $t \sim \text{Uniform}(\{1, \ldots, T\})$
4. $\epsilon \sim \mathcal{N}(0, I)$
5. Take gradient step on

$$\nabla_\theta \left\| \epsilon - \epsilon_\theta\left(\sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon,\ t\right) \right\|^2$$

6. **until** converged

Algorithm 2: Sampling

1. $x_T \sim \mathcal{N}(0, I)$
2. **for** $t = T, \ldots, 1$ **do**
3. $z \sim \mathcal{N}(0, I)$ if $t > 1$, else $z = 0$
4.

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) + \sigma_t z$$

5. **end for**
6. **return** x_0

The model learns to predict the noise added at each timestep.

Conditioned Diffusion and Guidance

Conditioned generation

$$\epsilon_\theta(x_t, t, c)$$

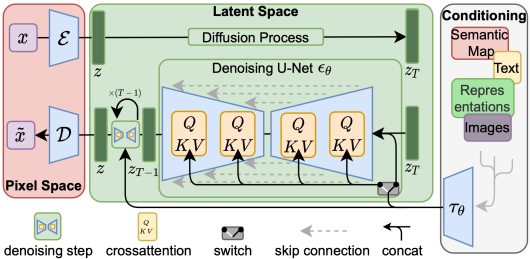
where c can be a class label, text prompt, image, or other condition.

- Classifier guidance uses an external classifier to steer sampling.
- Classifier-free guidance trains conditional and unconditional behavior together.
- Higher guidance can improve alignment, but may reduce diversity.

Modern text-to-image systems use conditioning and guidance to control what the model generates.

Latent Diffusion Model (LDM)

- Diffusion is performed in a *compressed latent space* instead of pixel space.
- **Autoencoder:** encodes image $x \to z$ using VAE and decodes $z \to x$ using VQVAE
- **Diffusion Model:** learns noise prediction in latent space.
- **Efficiency:** reduces computation while preserving high-quality generation.



Generative Process:

1. Encode image: $z = \mathcal{E}(x)$
2. Apply diffusion: $z_T \sim \mathcal{N}(0, I)$, then denoise $z_t \to z_0$
3. Decode latent: $x = \mathcal{D}(z_0)$

For more reading: https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Why Bayesian Optimization?

- Deep generative models often have many hyperparameters
- Manual tuning or grid search can be expensive
- Bayesian optimization helps search more efficiently

Typical tuning targets

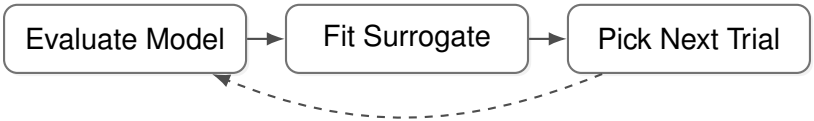
- learning rate
- latent dimension
- regularization strength
- batch size
- network depth

Basic Idea of Bayesian Optimization

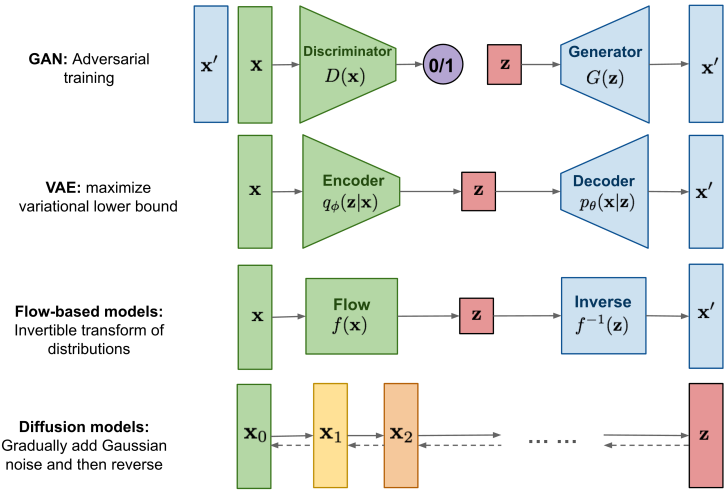
Core idea

Build a surrogate model of performance as a function of hyperparameters, then choose promising next settings to evaluate.

- model uncertainty about unknown performance
- explore promising and uncertain regions
- reduce number of expensive training runs



GAN, VAE, Flow, and Diffusion: Architecture



GAN, VAE, Flow, and Diffusion: A Comparison

Family	Training signal	Main strength	Main limitation
GAN	adversarial game	sharp, realistic samples	unstable training and mode collapse
VAE	maximize ELBO	stable training and latent structure	samples may be blurry
Flow	exact likelihood	tractable density and inversion	restricted invertible architectures
Diffusion	denoising objective	high-quality, diverse samples	slow sampling and high compute
Latent Diffusion	latent-space denoising	efficient, high-quality generation, , diverse samples	depends on autoencoder quality

No single family is universally best. The choice depends on whether we prioritize likelihood, sample quality, latent control, efficiency, or stability.

Applications of Generative Models

Media and language

- image, video, and audio synthesis
- image editing and inpainting
- text generation and summarization
- style transfer and translation

Data and science

- data augmentation
- anomaly detection
- molecule and protein design
- simulation and forecasting

Choosing a Model for an Application

- Need **sharp images**: consider GANs or diffusion models.
- Need **structured latent space**: consider VAEs.
- Need **exact likelihood**: consider normalizing flows.
- Need **text- or class-controlled generation**: consider conditional diffusion or conditional GANs.

The application should determine the modeling choice, not the other way around.

Why Evaluation Is Difficult

- A good generative model should produce realistic, diverse, and useful samples.
- Visual quality alone can hide mode collapse.
- Likelihood can be high even when perceptual quality is poor.
- Human preference, downstream usefulness, and statistical metrics often disagree.

Evaluation should combine qualitative inspection, quantitative metrics, and task-specific tests.

Evaluation of Generative Models

Criterion	What it measures	Standard metrics
Sample Quality	Perceptual realism of generated samples	FID↓, IS↑, KID↓, LPIPS↓, PSNR↑, SSIM↑, human evaluation
Diversity	Coverage of data distribution (mode coverage)	Precision/Recall, Coverage, Density
Likelihood	Fit to true data distribution	NLL↓, Bits/Dim↓
Condition Alignment	Consistency with input condition (text/label)	CLIP Score↑, Accuracy↑
Robustness & Utility	Usefulness for downstream tasks	Task accuracy, Transfer performance

A strong generative model should achieve low FID, high diversity, good likelihood estimates, and strong alignment with conditioning signals. No single metric fully captures performance.

Ethical Issues and Risks

- generative models can create misleading synthetic content
- outputs may reproduce or amplify training-data bias
- generated media can be used for impersonation or misinformation
- copyright, consent, and data provenance can become unclear

Responsible use requires

- transparency about synthetic content
- careful dataset governance
- safety filters and misuse monitoring
- human oversight in high-stakes settings

Limitations of Generative Models

- training and sampling can be computationally expensive
- outputs depend strongly on training data quality
- models may generate plausible but false details
- evaluation remains imperfect
- synthetic data is not always a safe substitute for real data

Generative models learn statistical structure. They do not guarantee truth, fairness, or real-world validity.

Key Takeaways

- Generative models learn data distributions and can generate new samples.
- VAEs optimize an ELBO that balances reconstruction and latent regularization.
- GANs use an adversarial game; they can produce sharp samples but are hard to train.
- Flow-based models use invertible transformations for exact likelihood.
- Diffusion models learn to reverse a gradual noising process and now dominate many image-generation tasks.
- Evaluation should consider quality, diversity, likelihood, condition alignment, and usefulness.
- Ethical use requires attention to bias, misinformation, provenance, and human oversight.

Generative modeling is a family of approaches, not one algorithm. Each method makes a different tradeoff between likelihood, quality, controllability, stability, and speed.

References

1. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*, MIT Press.
2. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer.
3. D. P. Kingma, M. Welling, *Auto-Encoding Variational Bayes*.
4. I. Goodfellow et al., *Generative Adversarial Nets*.
5. I. Gulrajani et al., *Improved Training of Wasserstein GANs*.
6. D. Rezende, S. Mohamed, *Variational Inference with Normalizing Flows*.
7. J. Ho, A. Jain, P. Abbeel, *Denoising Diffusion Probabilistic Models*.
8. L. Weng, *From GAN to WGAN*, Lil'Log, 2017.
9. L. Weng, *From Autoencoder to Beta-VAE*, Lil'Log, 2018.
10. L. Weng, *Flow-based Deep Generative Models*, Lil'Log, 2018.
11. L. Weng, *What are Diffusion Models?*, Lil'Log, 2021.