Generative Models and Diffusion Model

# Part 1: General Step-by-Step Workflow of a Generative Model

## Step 1: Data Collection & Preprocessing

Collect a large dataset from the domain you want to generate (images, text, audio, etc.)

Preprocess data:

* - Images: normalize to [0, 1] or [-1, 1], resize, etc.
* - Text: tokenize and encode as sequences of integers.
* - Audio: transform into spectrograms or raw waveforms.

## Step 2: Choose Model Architecture

Different types of models define the generation process differently:

| Model Type | Latent Space? | Generation Style |

|------------|----------------|------------------|

| GPT | No | Token by token |

| VAE | Yes | Sample latent → Decode |

| GAN | Yes | Sample latent → Generator |

| Diffusion | No (usually) | Sample noise → Iterative denoising |

## Step 3: Define the Objective Function (Loss)

This depends heavily on the model type:

* - Autoregressive (e.g., GPT): cross-entropy loss
* - VAE: ELBO = reconstruction loss + KL divergence
* - GAN: adversarial loss (minimax game)
* - Diffusion: MSE between noise added and predicted noise

## Step 4: Train the Model

* - Feed forward your data into the model
* - Compute the loss
* - Backpropagate gradients
* - Use optimizers like Adam or RMSProp
* - For diffusion: train with noise schedule

## Step 5: Evaluate the Model

* - Use domain-specific metrics (e.g., IS, FID for images)
* - Check for overfitting, mode collapse, diversity

## Step 6: Generation (Inference Phase)

* - Use the trained model to generate new samples
* - Diffusion: start with noise, apply reverse denoising

## Step 7: Postprocessing (if needed)

* - Convert token IDs to text
* - Convert normalized arrays to RGB images
* - Final cleanup (filters, etc.)

## Optional: Fine-Tuning or Reinforcement

* - Fine-tune on domain-specific data
* - Use reinforcement learning for targeted generation (e.g., RLHF)

# Part 2: Diffusion Model

## 1. Dataset Preparation

Start with a dataset—usually images (e.g., CIFAR-10, CelebA).

Normalize input values to [-1, 1] or [0, 1].

## 2. Forward Process – Add Noise

Progressively corrupt the original image by adding Gaussian noise using a noise schedule.

Key variables: x₀ (clean image), xₜ (noisy image at timestep t), betas (noise schedule), alpha\_hat (cumulative product).

## 3. Model Architecture – Noise Predictor

A U-Net is trained to predict the added noise.

Incorporates time embedding to handle different timesteps.

## 4. Training Loop

1. Sample a real image

2. Randomly choose a timestep

3. Add noise using the forward process

4. Train the model to predict that noise using MSE loss

## 5. Sampling (Reverse Process)

Start from pure Gaussian noise and iteratively denoise using the trained model.

## 6. Advanced Tricks (Optional)

* - Classifier-Free Guidance: better conditional generation
* - Latent Diffusion: perform diffusion in compressed space
* - DDIM: deterministic and faster sampling
* - ControlNet: add conditioning like sketches, depth maps

## Summary of Developer Flow

| Step | Description |

|------|-------------|

| 1. Load data | Normalize and batch image data |

| 2. Add noise | Simulate diffusion using betas |

| 3. Train model | Predict the noise that was added |

| 4. Sample | Start from noise, denoise gradually |

| 5. Output | Final denoised image is your generated sample |