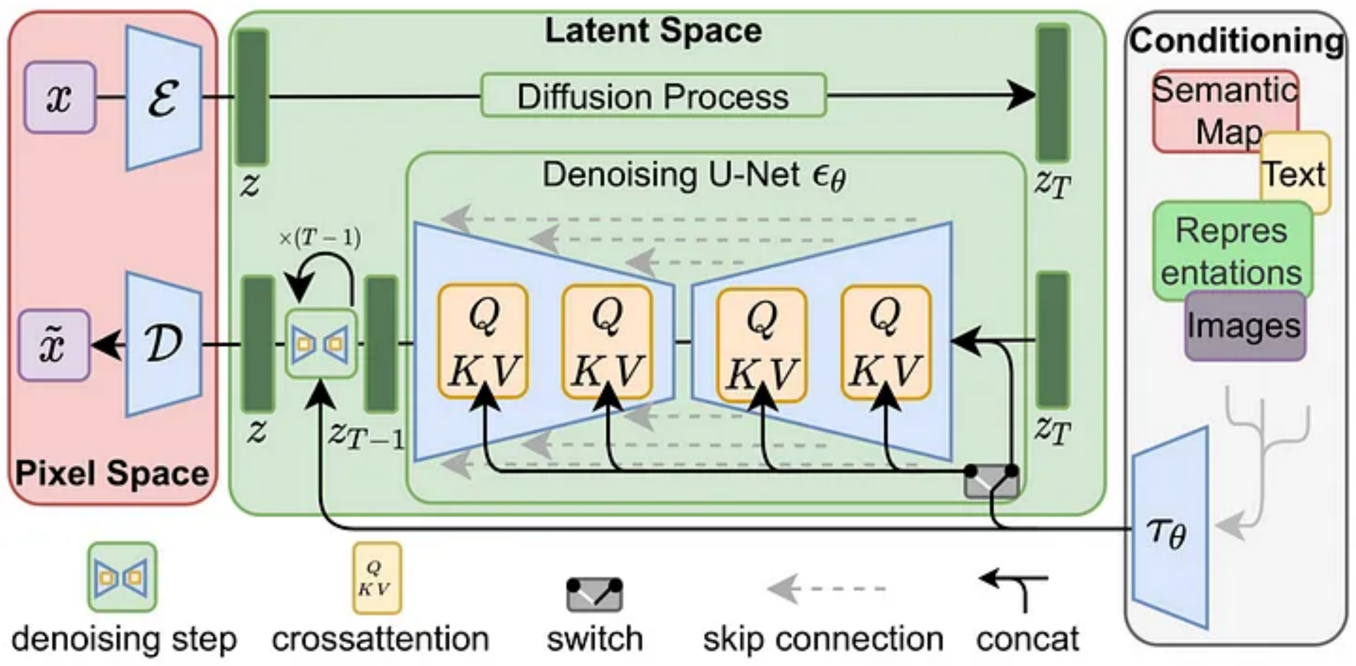

What is Stable Diffusion?

Stable Diffusion is an AI model that creates images from text by starting with pure noise (like static) and gradually **removing the noise step by step** until the image matches the description. It learns how to turn noise into meaningful images using millions of examples, combining deep neural networks with attention mechanisms to understand both visual patterns and text prompts.

Before starting, there are some topics we need to cover, so let's start with them.

KL Divergence

KL divergence (Kullback-Leibler divergence) is a concept in information theory and probability that measures how different one probability distribution is from another.

Think of KL divergence as answering the question: *How much information is lost if we approximate one probability distribution with another?*

It quantifies how much one probability distribution Q differs from another probability distribution P. The more different they are, the higher the KL divergence.

Formula

The KL divergence between two distributions P and Q is given by:

$$D_{KL}(P||Q) = \sum_x P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

Where:

- P is the **true distribution** (what the data really looks like)
- Q is the **approximate distribution** (what we're trying to model)
- x represents the possible outcomes

What does it actually mean?

- If P(x) and Q(x) are **identical**, then:

$$D_{KL}(P||Q) = 0$$

- If Q is **very different** from P, the divergence will be large.

How is KL Divergence Used in Stable Diffusion?

In **Stable Diffusion** (and VAEs), KL divergence is used during the training phase to:

1. Push the encoded latent space close to a normal Gaussian distribution.
2. Ensure that the noise added during denoising follows the prior distribution.

The loss function often looks like:

$$L = L_{reconstruction} + \beta D_{KL}(q(z|x)||p(z))$$

More on this later

> *Jensen's Inequality*

Jensen's Inequality tells us how a **convex function** (a function is convex if its curve always bends upwards — like a bowl.) interacts with the **average of random variables.**

In simple words:

$$f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$$

where:

- X is a Random variable (like position, energy, or vibration)
- f(x) is a convex function
- E[X] is the **average** value of X (expected value)
- E[f(X)] is the average of the function applied to X

What all of this basically means is that if you first **average the raw data** and then apply the function, you'll always get **less or equal information** than if you apply the function first and then average.

> *Markov chain*

A **Markov Chain** is a mathematical model that describes a sequence of events where the **next state depends only on the current state** — not on the entire history of previous states.

Markov Chains are used whenever you need to **model randomness over time** — especially in:

- Text Generation (like LLMs)
- Image Generation (like Stable Diffusion)
- Reinforcement Learning
- Time Series Forecasting
- Hidden Markov Models (HMMs) in speech recognition

> *Working*

1. Forward Diffusion Process

- The forward process is like breaking down an ordered system into chaos.
- We gradually add Gaussian noise over **T** timesteps.
- At every step, the image becomes noisier.

We define a **Markovian forward diffusion** that gradually adds Gaussian noise to an image $\mathbf{x_0}$ over time t, creating a noised version $\mathbf{x_t}$. This process is modeled as:

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

Where:

- $\mathbf{x_0}$ → The original clean image
- $\mathbf{x_t}$ → Noisy image at timestep **t**
- α_t→ Noise scaling factor at timestep **t**
- N → Gaussian distribution

→ **What is Markovian Forward Diffusion?**

- A **Markovian process** simply means that the **future state** depends **only on the present state** — not on the entire history.
- Mathematically:

$$q(x_t|x_{t-1}, x_{t-2}, ..., x_0) = q(x_t|x_{t-1})$$

So, the probability of $\mathbf{x_t}$ only depends on the **previous step $\mathbf{x_{t-1}}$**, making the process **memoryless.**

By leveraging the **reparametrization trick**, we can write the diffusion process directly as:

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

→ **What is reparametrization trick?**

- In the forward diffusion process, we start with an image x0x_0x0, and after T timesteps, it becomes pure noise:

$$x_T \sim \mathcal{N}(0, I)$$

- Now the whole game is about **reversing this process**.
- What we are actually training the model to do is predict the **noise** at each step:

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

- But the catch is, the whole reverse diffusion process is stochastic (random) because noise is involved at each step.
- If the process is random, how can the model be trained through backpropagation?
- You cannot directly backpropagate through **random noise sampling** because neural networks are **deterministic** — they can't optimize parameters through pure randomness.

That's Where the **Reparameterization Trick** Comes In:

Instead of sampling noise directly like this:

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$$

We now **reparameterize** the noise:

$$\epsilon = \epsilon_\theta(x_t, t) + z$$

Where:

- **$\epsilon\theta(x_t, t)$** is the **deterministic neural network prediction**.
- $z \sim N(0,I)$ is the **pure Gaussian noise**.

Now rewrite the whole diffusion equation like this:

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} (\epsilon_\theta(x_t, t) + z)$$

Now the entire process becomes **differentiable**.

The randomness is pushed into the separate z term, which **does not depend on the model parameters**.

Now, during backpropagation Only **$\epsilon\theta(x_t, t)$** gets updated, while the pure Gaussian noise stays fixed.

→ **Full Pipeline**

1. Start with pure noise **x_t**$\sim N(0,I)$

2. For each timestep t:

- Predict the noise: **$\epsilon\theta(x_t, t)$**
- Sample new noise: z~N(0,I)
- Reconstruct the clean image:

$$x_{t-1} = \frac{x_t - \sqrt{1-\alpha_t}\epsilon_\theta(x_t, t)}{\sqrt{\alpha_t}} + \sqrt{1-\alpha_t}z$$

2. Reverse Diffusion Process

In the reverse process, we want to **denoise the pure Gaussian noise** step by step until we regenerate the original data **x_0**.

The Reverse Diffusion Equation

The forward diffusion process follows this Markov chain:

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

The **Reverse Diffusion Process** is the exact opposite:

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

Let's break this down

a. The Reverse Mean $\mu\theta(x_t, t)$

- The reverse mean is:

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

- We start with the noisy sample **x_t**
- Predict what the noise **should have been** using the neural network
- Subtract the noise to get a slightly denoised version
- Scale everything back by the inverse diffusion factor

b. The Variance $\Sigma\theta(x_t, t)$

- The variance controls **how much randomness** we inject at each step.
- For most diffusion models, it's fixed as:

$$\Sigma_\theta(x_t, t) = \sigma_t^2 I$$

- Where:

$$\sigma_t^2 = \beta_t$$

The model can also predict this variance, but most papers keep it fixed.

Thus, the final reverse diffusion equation is:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}\left(\frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t)\right), \beta_t I\right)$$

Recap:

- Start with random Gaussian noise
- At each step, subtract the predicted noise (order from chaos)
- Add a little bit of fresh Gaussian noise (to maintain stochasticity)
- Repeat this for **T** steps

3. Variational Lower Bound & Training Objective

Given **data $\mathbf{x}_0$**, we want to model the probability distribution:

$$p_\theta(x_0)$$

But directly maximizing this likelihood is **impossible**.

Instead, we turn to **Variational Inference** — where we approximate this likelihood using a **Variational Lower Bound (VLB)**.

The full training objective is maximizing the **log-likelihood** of the data:

$$\log p_\theta(x_0)$$

But we can’t compute this directly.

Instead, we derive a lower bound using **Jensen’s Inequality**:

$$\log p_\theta(x_0) \geq \mathbb{E}_q \left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}\right]$$

This lower bound is called the **ELBO** (Evidence Lower Bound):

$$\mathcal{L}_{\text{VLB}}(\theta) = \mathbb{E}_q \left[\log p_\theta(x_{0:T}) - \log q(x_{1:T}|x_0)\right]$$

Let’s break this down

First, rewrite the joint probability of the entire Markov chain:

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

And the forward process is:

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

Now the ELBO becomes:

$$\mathcal{L}_{\text{VLB}} = \mathbb{E}_q \left[\log p(x_T) + \sum_{t=1}^{T} \log \frac{p_\theta(x_{t-1}|x_t)}{q(x_t|x_{t-1})}\right]$$

Now the ELBO is **three separate losses:**

$$\mathcal{L}_{\text{VLB}} = \mathcal{L}_{\text{reconstruction}} + \sum_{t=2}^{T} \mathcal{L}_{\text{KL}}(t) + \mathcal{L}_{\text{prior}}$$

→ **Reconstruction Loss**

$$\mathcal{L}_{\text{reconstruction}} = -\mathbb{E}_q \left[\log p_\theta(x_0|x_1)\right]$$

The main loss is the **Kullback-Leibler Divergence** between the true forward distribution and the predicted reverse distribution:

$$\mathcal{L}_{\text{KL}}(t) = D_{KL}\left(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t)\right)$$

Finally, at the last timestep we compute **Prior Matching Loss:**

$$\mathcal{L}_{\text{prior}} = D_{KL}(q(x_T|x_0) \parallel p(x_T))$$

But since both are Gaussian, this KL divergence has a **closed form**:

$$\mathcal{L}_{\text{prior}} = \frac{1}{2}\left(||x_T||^2 - d\right)$$

This is so genius because:

- The model **never** predicts the image directly
- It only predicts the **noise** at each step

That's why the entire training objective is:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t,x_0,\epsilon}\left[||\epsilon - \epsilon_\theta(x_t, t)||^2\right]$$

The full loss is:

$$\mathcal{L} = \sum_{t=1}^{T} \lambda_t D_{KL}\left(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t)\right)$$

But the simple version (used in 99% of papers) is:

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{t,x_0,\epsilon}\left[||\epsilon - \epsilon_\theta(x_t, t)||^2\right]$$

> *Example Walkthrough of Stable Diffusion*

1. Dataset & Input Preparation

Let's say we have a dataset of **cat images** $x \in R^{3x256x256}$

2. Forward Diffusion (Markovian Process)

We gradually add Gaussian noise to the image in **T=1000** steps using the forward diffusion process:

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

where:

- **β_t** is the noise schedule (small noise at the beginning, larger noise at the end)
- **t** is the time step
- **$\sqrt{(1 - \beta_t)}$** scales down the image
- **β_t**I is the Gaussian noise added at each step

Instead of sampling one step at a time, we can directly jump to any time step **t**:

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

with:

$$\bar{\alpha}_t = \prod_{i=1}^{t}(1-\beta_i)$$

Example:

Let's say:

- **β_t**=0.01
- t=100
- **x_0** is an image of a cat

Then:

$$\bar{\alpha}_{100} = \prod_{i=1}^{100}(1-0.01) = (0.99)^{100} \approx 0.37$$

The noisy image becomes:

$$x_{100} \sim \mathcal{N}(0.37 \cdot x_0, 0.63 \cdot I)$$

3. Reverse Diffusion Process (Model Training Objective)

Now the goal is to **reverse the noise** and recover the original image.

We train a neural network **ϵθ(x_t, t)** to predict the noise at each step:

$$\epsilon_\theta(x_t, t) \approx \epsilon \sim \mathcal{N}(0, I)$$

The denoising step is:

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

Where:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t))$$

4. Training Objective (Variational Lower Bound)

The network is trained to minimize the **simplified evidence lower bound (ELBO)**:

$$L_\theta = \mathbb{E}_{x_0,t,\epsilon}\left[||\epsilon - \epsilon_\theta(x_t, t)||^2\right]$$

This means the model is trained to **predict the noise** added at each timestep.

Example Training Step

Let,

- **Image** = Cat
- **x_0** = Cat Image
- **t** = 500
- **β_t** = 0.02

→ **Forward Diffusion**

$$x_{500} = \sqrt{0.2} \cdot x_0 + \sqrt{0.8} \cdot \epsilon$$

→ **Neural Network Prediction:**

$$\epsilon_\theta(x_{500}, 500) \approx \epsilon$$

→ **Loss Function**

$$L = ||\epsilon - \epsilon_\theta(x_{500}, 500)||^2$$

→ **Gradient Descent**

$$\theta = \theta - \eta \nabla_\theta L$$

After training, we generate images starting from pure Gaussian noise:

- Start with x_t~N(0,I)
- Sample iteratively:

$$x_{t-1} \sim p_\theta(x_{t-1}|x_t)$$

until t=0.

Conclusion

To summarize:

- Sample an image x_0 from the dataset
- Randomly choose a timestep t~U(1,T)
- Generate noisy image:

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon$$

- Predict the noise:

$$\hat{\epsilon} = \epsilon_\theta(x_t, t)$$

- Compute Loss:

$$L = ||\epsilon - \hat{\epsilon}||^2$$

So, this was it! Hopefully, you liked this in the next article, we’ll code this up using PyTorch!