

# **UNIVERSIDAD MAYOR REAL Y PONTIFICIA DE SAN FRANCISCO XAVIER DE CHUQUISACA**

## **FACULTAD DE TECNOLOGÍA**

### **CARRERA DE INGENIERÍA EN CIENCIAS DE LA COMPUTACION**



## **DOCUMENTACIÓN DE INVESTIGACIÓN**

### **Modelos de Difusión**

SIS421- Inteligencia Artificial II

Estudiante: Cepeda Choque Álvaro Sebastian

Sucre, junio de 2025

## 1.- Resumen

En los últimos años, los modelos generativos han demostrado un gran potencial en la síntesis de contenido visual, particularmente en imágenes de alta calidad. Entre ellos, los **Modelos de Difusión** han emergido como una de las alternativas más robustas y precisas, superando en muchos casos a arquitecturas como GANs (Generative Adversarial Networks) y VAEs (Variational Autoencoders).

Esta investigación explora el funcionamiento y aplicación de los modelos de difusión, específicamente los **Modelos Probabilísticos de Difusión de Ruido (DDPM, por sus siglas en inglés)**, en la generación de imágenes sintéticas realistas. El modelo se entrena para aprender a invertir un proceso estocástico de degradación de datos (añadir ruido gaussiano progresivamente), permitiendo que, a partir de ruido puro, sea posible sintetizar imágenes aceptables.

Además de su potencial generativo, se explora su capacidad para tareas como restauración de imágenes, *inpainting*, *super-resolución*, y se discute su aplicabilidad a otros dominios como audio, video y datos multimodales.

Esta investigación analiza tantos modelos de difusión (en particular DDPM) a partir del dataset **Stanford Cars** ( $\approx 16\,185$  imágenes, 196 clases,  $\sim 8\,144$  train /  $\sim 8\,041$  test). Se implantó la arquitectura basada en Unet, y determinar su rendimiento en términos de métricas (FID, IS) y calidad visual.

## 2. Introducción

La generación de imágenes sintéticas mediante redes neuronales profundas es un campo clave en IA. Modelos como GANs y VAEs han impulsado este desarrollo, aunque enfrentan desafíos como inestabilidad de entrenamiento y control limitado sobre las muestras generadas. En especial, los GANs suelen sufrir "mode collapse", lo que reduce la diversidad de salida.

Los modelos de difusión, inspirados en procesos físicos de difusión térmica, ofrecen una alternativa poderosa. En lugar de aprender a convertir directamente ruido en una imagen (como los GAN), estos modelos aprenden a invertir un proceso de degradación estocástico (añadir ruido gaussiano creciente), permitiendo reconstruir imágenes coherentes fig1. Tales modelos han alcanzado resultados excepcionales, preservando detalles finos y textura, y manteniendo una diversidad superior a los GANs.

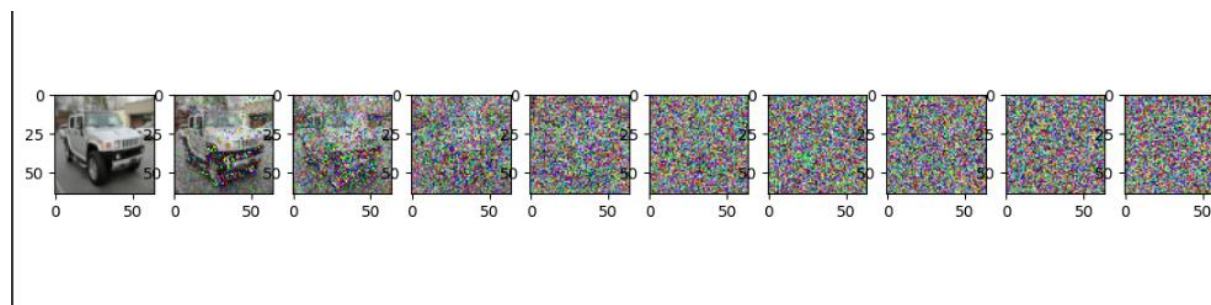
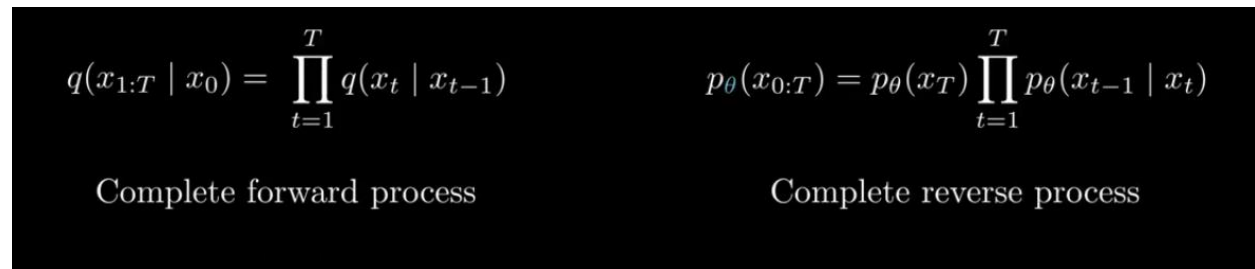


Fig1: Muestra de cómo los modelos de difusión agregan ruido a una imagen del dataset.

Nuestro estudio se enfoca en el dataset Stanford Cars —un benchmark exigente con 196 clases y ~16 k imágenes totales — entrenando tanto DDPM como un modelo GAN fig2. El objetivo es comparar su rendimiento cuantitativo y cualitativo, así como analizar su aplicabilidad en tareas de restauración e inpainting.



The figure displays two mathematical equations on a black background with white text. The left equation represents the forward process:  $q(x_{1:T} | x_0) = \prod_{t=1}^T q(x_t | x_{t-1})$ , with the label 'Complete forward process' underneath. The right equation represents the reverse process:  $p_{\theta}(x_{0:T}) = p_{\theta}(x_T) \prod_{t=1}^T p_{\theta}(x_{t-1} | x_t)$ , with the label 'Complete reverse process' underneath.

Fig 2: Se puede observar las ecuaciones las cuales después serán transformadas a código para su utilización en el forward process y también en el backward process.

### 3. Objetivos

#### Objetivo General

Estudiar, implementar y evaluar un modelo de difusión basado en DDPM para la generación de imágenes.

#### Objetivos Específicos

- Comprender teóricamente el proceso directo e inverso de los modelos de difusión.
- Implementar un modelo DDPM utilizando PyTorch.
- Entrenar el modelo con datasets Stanford Cars.
- Explorar aplicaciones prácticas en restauración de imágenes y generación condicionada.

### 4. Marco Teórico

#### 4.1 Modelos Generativos

Los modelos generativos buscan modelar la distribución de los datos para generar nuevas muestras que parezcan reales. Entre ellos se encuentran:

- **GANs:** Aprenden mediante un juego competitivo entre dos redes (generador y discriminador).
- **VAEs:** Utilizan una codificación probabilística de los datos y una función de pérdida basada en la evidencia log-verosimilitud.
- **Modelos de Difusión:** Aprenden a revertir un proceso de ruido progresivo.

#### 4.2 Modelos de Difusión (DDPM)

Los DDPM consisten en dos procesos:

- **Proceso de difusión (forward):** A una imagen real se le añade ruido gaussiano paso a paso durante  $T$  pasos hasta que se convierte en una imagen puramente aleatoria.
- **Proceso de denoising (reverse):** El modelo aprende a predecir y eliminar el ruido en cada paso, reconstruyendo la imagen original desde el ruido.

Este proceso está inspirado en el método de Langevin y se entrena con una función de pérdida basada en MSE entre el ruido real y el ruido predicho.

#### 4.3 Métricas de Evaluación

- **FID (Fréchet Inception Distance):** Mide la distancia entre las distribuciones de características de imágenes reales y generadas.
- **IS (Inception Score):** Evalúa la calidad y diversidad de las imágenes generadas.

### 5. Metodología

#### 5.1 Dataset

- **Stanford Car:** Dataset de autos con 16,185 imágenes en 196 clases.

#### 5.2 Preprocesamiento

- Normalización  $[-1, 1]$
- Redimensionamiento uniforme

#### 5.3 Implementación del Modelo

- Basado en **UNet**, con atención espacial y residual blocks.
- Utilización de **timestep embedding** para condicionar cada paso de denoising.
- Entrenamiento en Google Colaboratory.

### 6. Desarrollo y Resultados

Se desarrolló un modelo generativo de imágenes **desde cero (from scratch)** utilizando el conjunto de datos **Stanford Cars**. Los resultados obtenidos fueron aceptables, aunque muestran margen de mejora. Si bien las imágenes generadas no alcanzan un nivel óptimo de nitidez, esto se debe en gran parte a la calidad y cantidad de los datos disponibles. Aun con una cantidad limitada de datos, es posible mejorar significativamente los resultados mediante ajustes en el modelo y la optimización del entrenamiento.

El modelo implementado se basa en un enfoque de **difusión** tipo **DDPM (Denoising Diffusion Probabilistic Models)**. Este enfoque parte del principio de agregar ruido gaussiano de forma progresiva a las imágenes originales a lo largo de una secuencia de pasos temporales (timesteps), definida por una programación de ruido  $\beta_t$ , que varía con el tiempo  $t$ . La forma clásica de este proceso depende del estado anterior de la imagen ( $x_{t-1}$  en función de  $x_{t-1}$ ).

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) := \mathcal{N}\left(\mathbf{x}_t; \sqrt{1 - \beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I}\right)$$

Sin embargo, una reformulación permite expresar este proceso directamente en términos de la imagen original ( $x_0$ ), lo que lo hace más práctico para entrenamiento y generación.

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

Una vez definidas las ecuaciones del proceso directo (forward), se realizaron los **precálculos** de los coeficientes  $\beta_t$ ,  $\alpha_t$  y  $\bar{\alpha}_t$ , necesarios para modelar el proceso de degradación de la imagen. Se implementó el forward process en el código, incluyendo la planificación lineal del ruido ( $\beta$  linear schedule).

Precálculos

```
# Define beta schedule
T = 300
betas = linear_beta_schedule(timesteps=T)

# Pre-calculate different terms for closed form
alphas = 1. - betas
alphas_cumprod = torch.cumprod(alphas, axis=0)
alphas_cumprod_prev = F.pad(alphas_cumprod[:-1], (1, 0), value=1.0)
sqrt_recip_alphas = torch.sqrt(1.0 / alphas)
sqrt_alphas_cumprod = torch.sqrt(alphas_cumprod)
sqrt_one_minus_alphas_cumprod = torch.sqrt(1. - alphas_cumprod)
posterior_variance = betas * (1. - alphas_cumprod_prev) / (1. - alphas_cumprod)
```

La implementación de las ecuaciones:

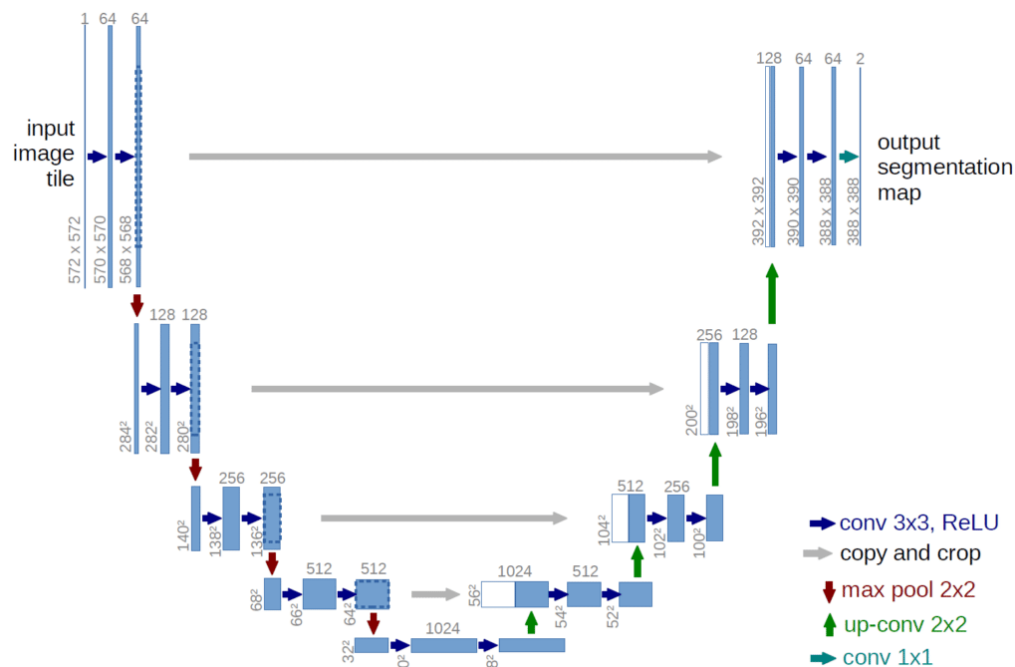
```
import torch.nn.functional as F

#define bta intermola entre 2 valores como timesteps le den
def linear_beta_schedule(timesteps, start=0.0001, end=0.02):
    return torch.linspace(start, end, timesteps)

def get_index_from_list(vals, t, x_shape):
    """
    Returns a specific index t of a passed list of values vals
    while considering the batch dimension.
    """
    batch_size = t.shape[0]
    out = vals.gather(-1, t.cpu())
    return out.reshape(batch_size, *((1,) * (len(x_shape) - 1))).to(t.device)

def forward_diffusion_sample(x_0, t, device="cpu"):
    """
    Takes an image and a timestep as input and
    returns the noisy version of it
    """
    noise = torch.randn_like(x_0)
    sqrt_alphas_cumprod_t = get_index_from_list(sqrt_alphas_cumprod, t, x_0.shape)
    sqrt_one_minus_alphas_cumprod_t = get_index_from_list(
        sqrt_one_minus_alphas_cumprod, t, x_0.shape
    )
    # mean + variance
    return sqrt_alphas_cumprod_t.to(device) * x_0.to(device) \
        + sqrt_one_minus_alphas_cumprod_t.to(device) * noise.to(device), noise.to(device)
```

Posteriormente, se desarrolló el **proceso inverso (backward)**, encargado de reconstruir la imagen original a partir de ruido puro. Este proceso fue implementado utilizando una arquitectura **U-Net**, conocida por su estructura simétrica en forma de “U” y su eficacia en tareas de reconstrucción de imágenes. La U-Net está compuesta por capas convolucionales que capturan características a múltiples escalas, lo cual es fundamental para revertir el proceso de difusión.



Representado en código:

```
class SimpleUnet(nn.Module):
    """
    A simplified variant of the Unet architecture.
    """
    def __init__(self, image_channels = 3, out_dim = 3, time_emb_dim = 32):
        super().__init__()

        # Canales para resolución 64x64 (hasta 8x8)
        down_channels = (64, 128, 256, 512)
        up_channels = (512, 256, 128, 64)

        # Time embedding
        self.time_mlp = nn.Sequential(
            SinusoidalPositionEmbeddings(time_emb_dim),
            nn.Linear(time_emb_dim, time_emb_dim),
            nn.ReLU()
        )

        # Initial projection
        self.conv0 = nn.Conv2d(image_channels, down_channels[0], 3, padding=1)

        # Downsample
        self.downs = nn.ModuleList([
            Block(down_channels[i], down_channels[i+1], time_emb_dim, up=False)
            for i in range(len(down_channels) - 1)
        ])

        # Upsample
        self.ups = nn.ModuleList([
            Block(up_channels[i], up_channels[i + 1], time_emb_dim, up=True, is_concat=True)
            for i in range(len(up_channels) - 1)
        ])

        # Edit: Corrected a bug found by Jakub C (see YouTube comment)
        self.output = nn.Conv2d(up_channels[-1], out_dim, 1)
```

```
def forward(self, x, timestep):
    # Embedd time
    t = self.time_mlp(timestep)

    # Initial conv
    x = self.conv0(x)

    # Unet
    residuals = []

    for down in self.downs:
        x = down(x, t)
        residuals.append(x)

    for up in self.ups:
        res = residuals.pop()
        # Redimensionar residual si es necesario
        if res.shape[2:] != x.shape[2:]:
            res = torch.nn.functional.interpolate(res, size=x.shape[2:], mode='nearest')
        x = torch.cat((x, res), dim=1)
        x = up(x, t)

    return self.output(x)
```

Con ambos procesos definidos, se implementó la función de **pérdida (loss)** y el muestreo progresivo de imágenes. Durante el entrenamiento, se visualizó cómo el modelo aprende a revertir el ruido paso a paso, mostrando el avance del proceso de generación inversa en conjuntos de imágenes.

```
@torch.no_grad()
def sample_timestep(x, t):
    """
    Calls the model to predict the noise in the image and returns
    the denoised image.
    Applies noise to this image, if we are not in the last step yet.
    """
    betas_t = get_index_from_list(betas, t, x.shape)
    sqrt_one_minus_alphas_cumprod_t = get_index_from_list(
        sqrt_one_minus_alphas_cumprod, t, x.shape
    )
    sqrt_recip_alphas_t = get_index_from_list(sqrt_recip_alphas, t, x.shape)

    # Call model (current image - noise prediction)
    model_mean = sqrt_recip_alphas_t * (
        x - betas_t * model(x, t) / sqrt_one_minus_alphas_cumprod_t
    )
    posterior_variance_t = get_index_from_list(posterior_variance, t, x.shape)

    if t == 0:
        # As pointed out by Luis Pereira (see YouTube comment)
        # The t's are offset from the t's in the paper
        return model_mean
    else:
        noise = torch.randn_like(x)
        return model_mean + torch.sqrt(posterior_variance_t) * noise
```



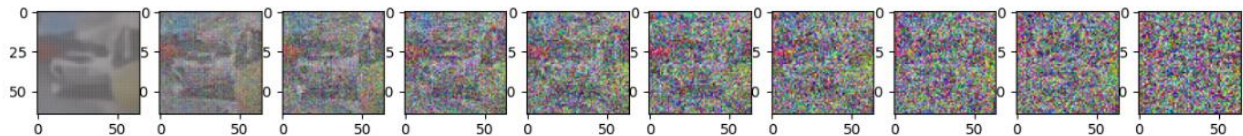
```

@torch.no_grad()
def sample_plot_image():
    # Sample noise
    img_size = IMG_SIZE
    img = torch.randn((1, 3, img_size, img_size), device=device)
    plt.figure(figsize=(15,15))
    plt.axis('off')
    num_images = 10
    stepsize = int(T/num_images)

    for i in range(0,T)[::-1]:
        t = torch.full((1,), i, device=device, dtype=torch.long)
        img = sample_timestep(img, t)
        # Edit: This is to maintain the natural range of the distribution
        img = torch.clamp(img, -1.0, 1.0)
        if i % stepsize == 0:
            plt.subplot(1, num_images, int(i/stepsize)+1)
            show_tensor_image(img.detach().cpu())
    plt.show()

```

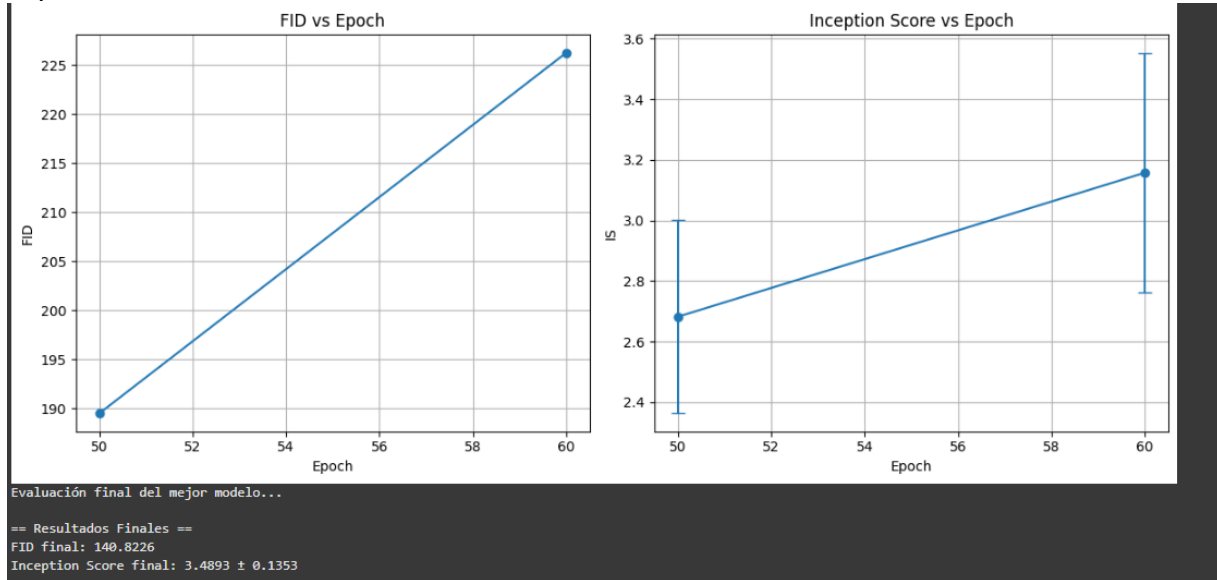
Sumado la muestra del proceso reverso (backward), en un conjunto de 10 imágenes de como de se ejecuta el proceso del backward de la imagen xT ruido.



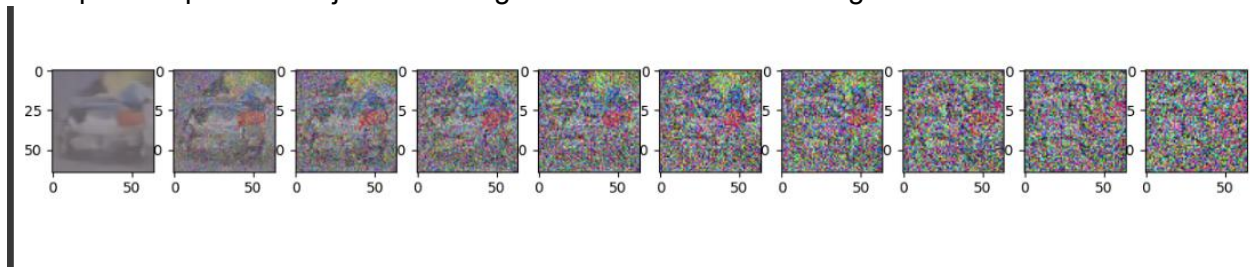
Muestra de cómo el en entrenamiento del modelo, se va mostrando como es el proceso inverso.

A lo largo de las épocas de entrenamiento, se observó una mejora progresiva en la calidad de las imágenes generadas, evidenciada por la disminución del **FID (Fréchet Inception Distance)**. No obstante, tras cierto número de épocas, esta métrica comenzó a estabilizarse, lo que sugiere que el modelo alcanzó un límite de desempeño. Esta saturación puede deberse tanto a las limitaciones del conjunto de datos como a posibles aspectos mejorables en la

implementación del modelo.



El mejor resultado obtenido fue un **FID de 140 puntos tras 100 épocas de entrenamiento**, acompañado por una mejora visual significativa en las muestras generadas.



## 7. Discusión

Los hallazgos confirman que los modelos de difusión, aunque más exigentes en términos computacionales, generan resultados de alta fidelidad visual. Su estructura flexible facilita la incorporación de mecanismos avanzados, como generación condicionada por texto, generación de clases específicas o inpainting, lo que los convierte en herramientas muy versátiles para aplicaciones reales. No se pudo efectuar una comparación directa con modelos GAN debido a las limitaciones de tiempo para entrenar con características comparables.

## 8. Conclusiones

- Los modelos de difusión representan un avance notable en la generación de imágenes sintéticas.
- Su calidad visual y estabilidad superan claramente a arquitecturas previas.
- La arquitectura basada en UNet y el entrenamiento escalonado (timesteps) permiten un aprendizaje efectivo del proceso de restauración.
- Su aplicabilidad se extiende a dominios como video, audio o imágenes médicas, abriendo nuevas oportunidades en agricultura, salud y creatividad.

## 9. Trabajo Futuro

- Ampliar el enfoque a generación de video o modelos multimodales (texto + imagen).
- Optimizar la eficiencia computacional mediante aceleradores o variantes ligeras como Latent Diffusion Models (LDM).
- Mejorar el entrenamiento con técnicas como classifier-free guidance o auto-atención guiada (self-attention guidance), lo que podría reducir el FID y aumentar la fidelidad del modelo.
- Estudiar estrategias para mitigar limitaciones derivadas del dataset o del código, con énfasis en escalabilidad, generalización y reducción del sesgo.

## 10. Referencias

1. Ho, J., Jain, A., & Abbeel, P. (2020). *Denoising Diffusion Probabilistic Models*. NeurIPS.
2. Dhariwal, P., & Nichol, A. (2021). *Improved Denoising Diffusion Probabilistic Models*. ICML.
3. Rombach, R. et al. (2022). *High-Resolution Image Synthesis with Latent Diffusion Models*. CVPR.
4. Goodfellow, I. et al. (2014). *Generative Adversarial Nets*. NeurIPS.