

INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY
CAMPUS SANTA FE

MODELADO GENERATIVO BASADO EN DIFUSIÓN PARA LA RECONSTRUCCIÓN DE IMÁGENES DE *PICO-BANANA-400K*

Presenta:

Alyson Melissa Sánchez Serratos
Miguel Ángel Pérez Ávila

Tutor:

Dr. Octavio Navarro Hinojosa

Fecha:

Jueves, 04 de diciembre de 2025



TABLA DE CONTENIDOS

- | | | | |
|----|-------------------------|----|--------------------------------|
| 01 | Introducción | 05 | Entrenamiento |
| 02 | Marco Teórico | 06 | Experimentaciones y Resultados |
| 03 | Arquitectura del Modelo | 07 | Discusión |
| 04 | Dataset PicoBanana | 08 | Conclusiones y Trabajo Futuro |

INTRODUCCIÓN

Los Modelos Probabilísticos de Difusión y Denoising (Denoising Diffusion Probabilistic Models, DDPM) se han consolidado como uno de los enfoques más relevantes dentro del modelado generativo contemporáneo. Su fundamento consiste en aprender a invertir un proceso progresivo de degradación por ruido, de manera que el modelo sea capaz de generar una muestra visual partiendo exclusivamente de ruido aleatorio.

En este trabajo se presenta la implementación de un modelo DDPM entrenado con el dataset PicoBanana, un conjunto de imágenes publicado por Apple que constituye un recurso adecuado para estudiar la capacidad del modelo de aprender estructuras visuales complejas. Este proyecto aborda el proceso de difusión directo e inverso, el entrenamiento del modelo y la evaluación de los resultados obtenidos con el fin de estudiar el comportamiento del enfoque y sus posibles áreas de mejora.

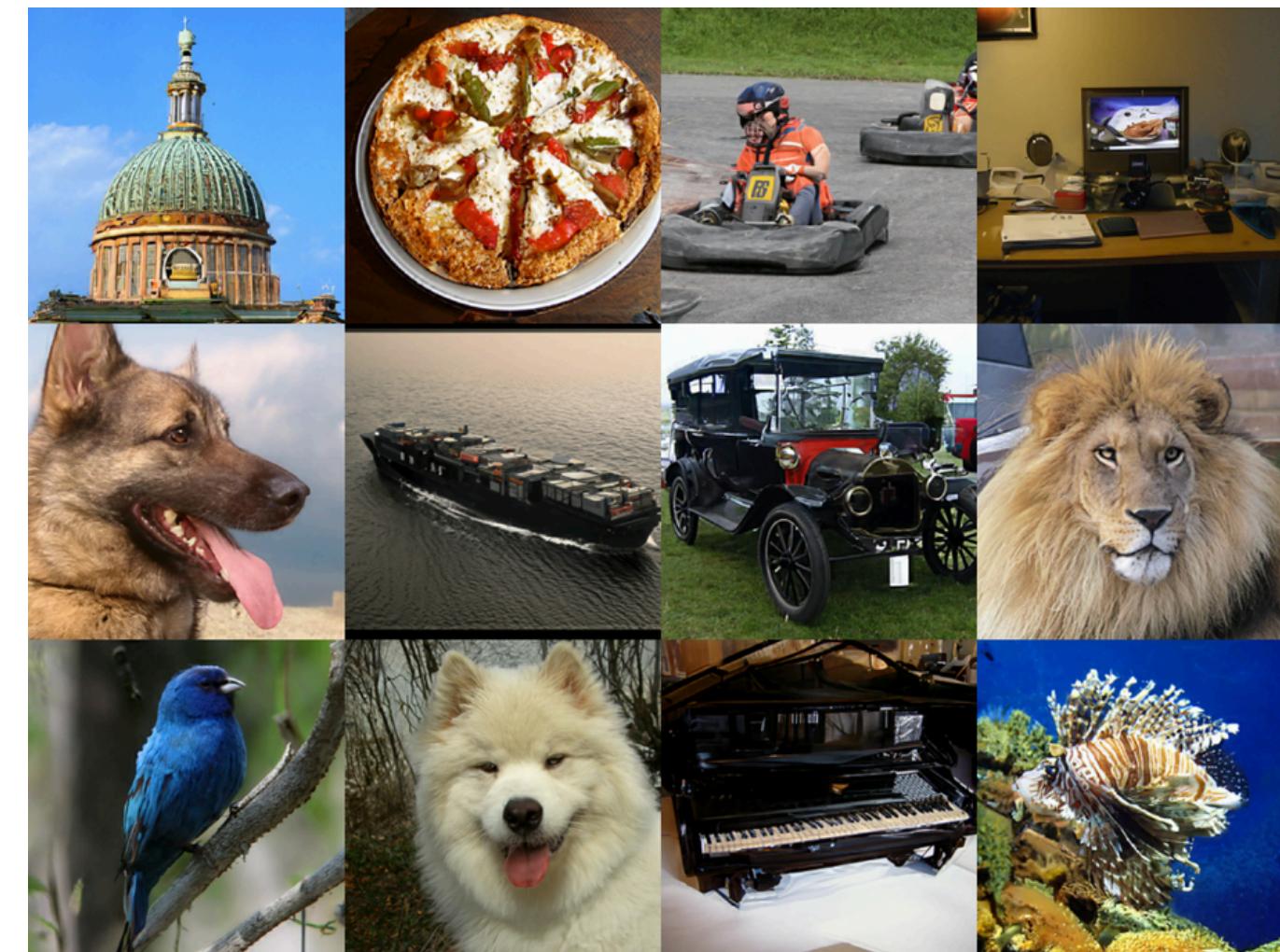


Figura 0. Visualización de imágenes sintéticas creadas por CDM en distintas clases de ImageNet a resolución 256×256.
Fuente: Ho, J., Saharia, C., Chan, W., Fleet, D. J., Norouzi, M., & Salimans, T. (2021). Cascaded Diffusion Models for High Fidelity Image Generation. arXiv (Cornell University). <https://doi.org/10.48550/arxiv.2106.15282>

MARCO TEÓRICO



Figura 1. Representación ficticia creada con IA (Midjourney): El Papa con abrigo y Trump detenido.

Fuente: Gascón, M., & Gascón, M. (2023, 31 marzo). Se acabó Midjourney gratis: la plataforma de inteligencia artificial cobrará por generar imágenes. 20minutos. <https://www.20minutos.es/tecnologia/aplicaciones/se-acabo-midjourney-gratis-la-plataforma-de-inteligencia-artificial-cobrara-por-generar-imagenes-5115095/>

1. Modelos Generativos

Los modelos generativos son un tipo de modelos de aprendizaje automático cuyo objetivo es aprender la distribución subyacente de los datos para poder generar nuevas muestras que sigan patrones similares a los del conjunto original. A diferencia de los modelos discriminativos, que se enfocan en clasificar o predecir etiquetas, los modelos generativos buscan capturar la estructura estadística completa de los datos.

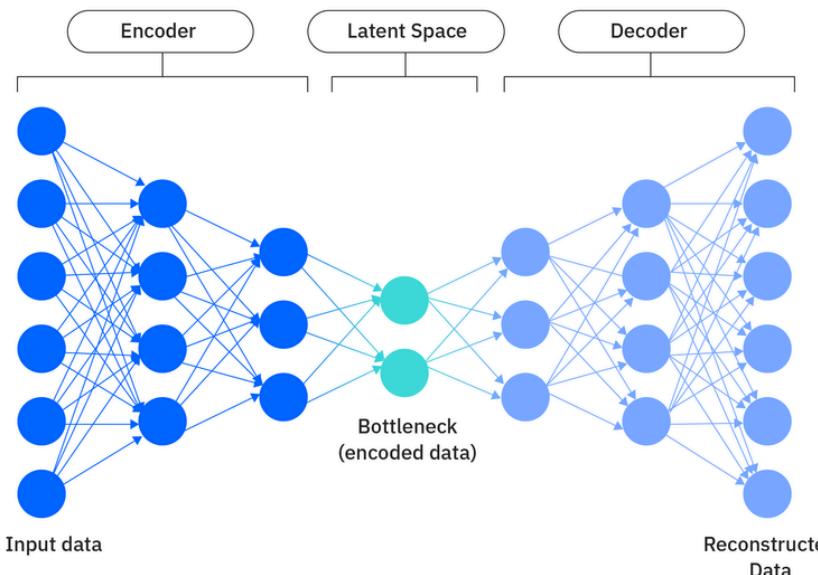
A lo largo de los últimos años han surgido distintas familias de modelos generativos, cada una con enfoques y capacidades diferentes. Entre las más influyentes se encuentran:

1. Modelos Generativos



Autoencoders Variacionales (VAE)

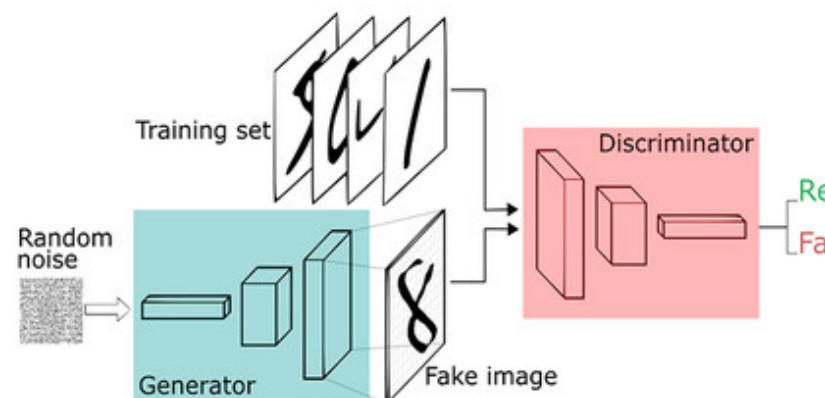
01



Los VAE se construyen con un codificador que comprime la imagen en una distribución latente (media y varianza), de la cual se muestrea un vector que el decodificador utiliza para reconstruir la imagen, permitiendo generar variaciones realistas del input.

Redes Generativas Adversarias (GANs)

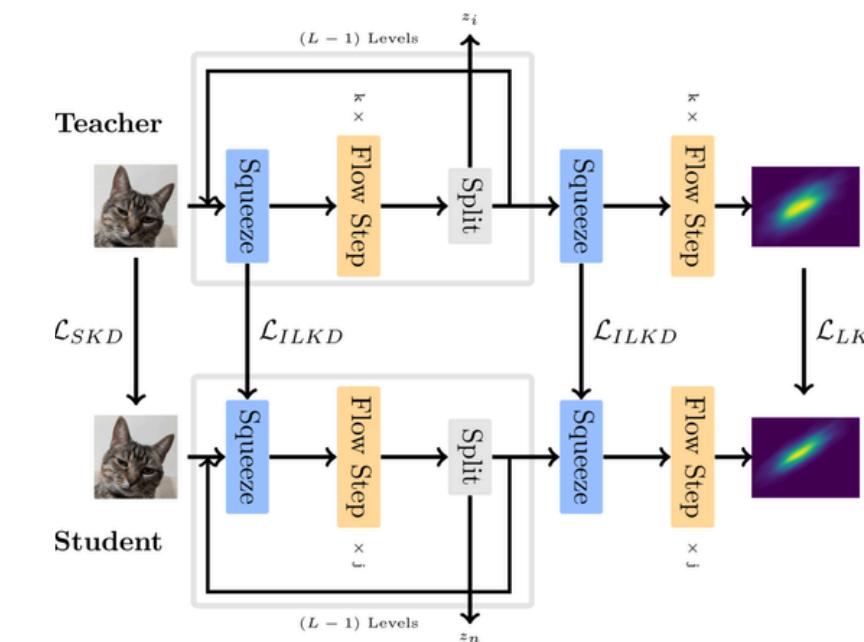
02



Las GANs consisten en dos redes que compiten entre sí: un generador, que produce muestras sintéticas, y un discriminador, que intenta distinguir entre muestras reales y generadas.

Normalizing Flows

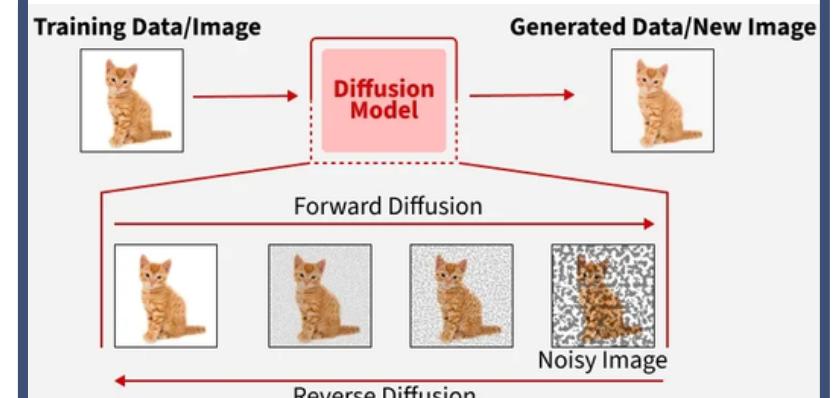
03



Los normalizing flows generan datos transformando una distribución simple, como una gaussiana, en una distribución compleja mediante una serie de transformaciones completamente invertibles.

Modelos de difusión

04



Los modelos de difusión generan datos invirtiendo un proceso en el que inicialmente se añade ruido de manera progresiva a una muestra real. El modelo aprende a revertir ese proceso.

2. Modelos de Difusión



Los modelos de difusión se basan en dos etapas complementarias que permiten transformar una imagen real en ruido y, posteriormente, generar una nueva imagen a partir de ese ruido.

1. Proceso de Difusión Directa (Forward)

Se inicia con una imagen real x_0 . En cada paso del proceso se añade una pequeña cantidad de ruido gaussiano controlado, de manera que la imagen se va degradando progresivamente. Tras suficientes pasos, se obtiene x_T , una muestra que se aproxima a ruido gaussiano puro.

2. Proceso de Difusión Inversa (Reverse)

Una vez alcanzado el estado ruidoso x_T , el objetivo es revertir ese proceso. Para ello, se entrena un modelo que predice el ruido presente en cada paso y permite reconstruir x_{t-1} a partir de x_t . Repitiendo esta operación desde $t=T$ hasta $t=1$, es posible generar una nueva muestra sintética que sigue la distribución de las imágenes reales.

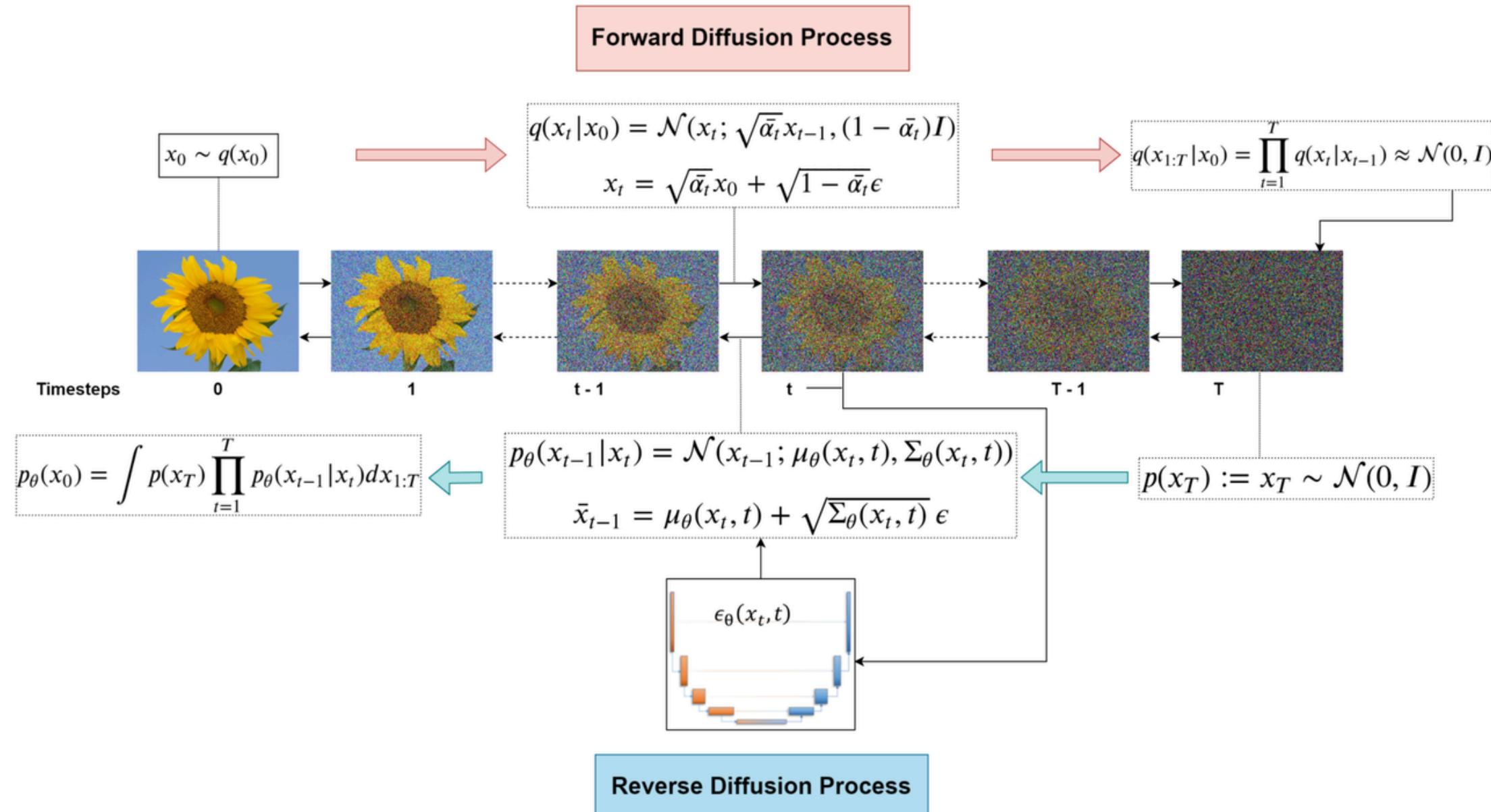


Figura 2. Diagrama del proceso de difusión, mostrando la degradación progresiva por ruido y la reconstrucción inversa aprendida por el modelo.
Fuente: Singh, V., & Singh, V. (2024). InDepth Guide to Denoising Diffusion Probabilistic Models (DDPM). LearnOpenCV. <https://learnopencv.com/denoising-diffusion-probabilistic-models/>

2.1. Proceso de Difusión Directa

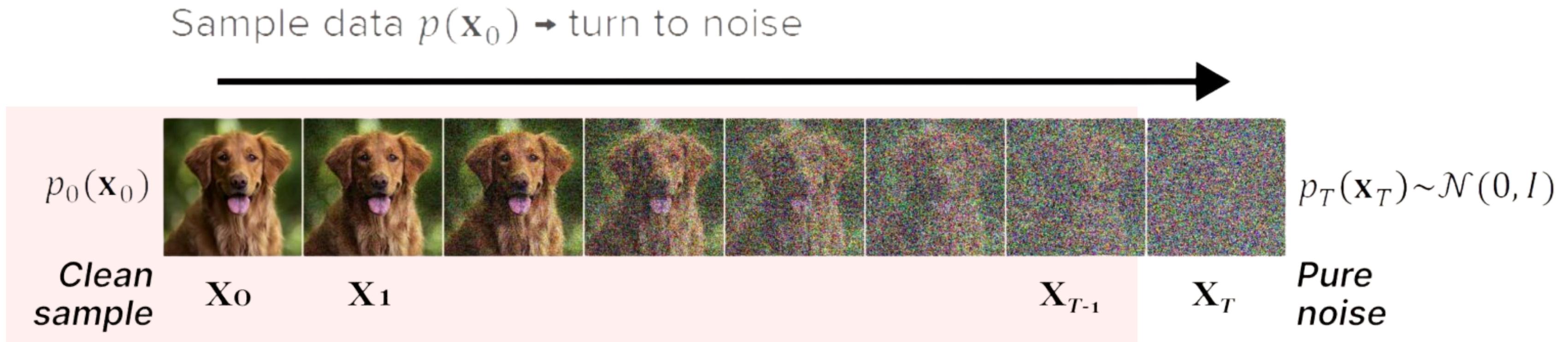


Figura 3. Secuencia del Forward Process: Transformación de la Imagen Original en Distribuciones de Ruido.
Fuente: Calibraint. (2024, 13 agosto). A Simple Guide to Diffusion Models. Calibraint. <https://www.calibraint.com/blog/beginners-guide-to-diffusion-models>

El proceso de difusión hacia adelante es una **cadena de Markov** que comienza con los datos originales x y termina en una muestra de ruido ϵ . En cada paso t , los datos se corrompen añadiendo ruido gaussiano. El nivel de ruido aumenta con t hasta llegar a 1 en el paso final T . En este punto, x_T es completamente aleatorio e independiente de x . Este proceso es completamente conocido, determinístico en su programación y no requiere aprendizaje.

2.1. Proceso de Difusión Directa

En el paso t , la nueva imagen ruidosa x_t se obtiene como: $x_t = \sqrt{1 - \beta_t}x_{t-1} + \sqrt{\beta_t}\varepsilon$

donde:

- x_{t-1} : es la imagen (o estado) en el paso anterior del proceso de difusión. Es la versión parcialmente ruidosa de la imagen en el paso $t-1$.
- x_t : La imagen ruidosa en el paso actual. Es el resultado de degradar x_{t-1} añadiéndole una cantidad controlada de ruido.
- β_t : Es un coeficiente de variancia entre 0 y que controla la cantidad de ruido gaussiano añadido en el paso t . Este parámetro define cuánta parte de la señal original se elimina y cuánta variabilidad aleatoria se introduce en ese paso del proceso forward.
 - Valores pequeños → poca degradación por paso.
 - Valores mayores → degradación más rápida.
- ε : Es el ruido gaussiano estándar. Representa la perturbación aleatoria introducida en este paso $\varepsilon \sim \mathcal{N}(0, I)$

Primer término

- Indica que no se elimina completamente la imagen previa.
- La cantidad de señal conservada se reduce en función de β_t .
- Si β_t es pequeño, se preserva más información; si es grande, se conserva menos.

$$\sqrt{1 - \beta_t}x_{t-1}$$

Segundo término

- Es el ruido gaussiano que se agrega en este paso.
- Su intensidad depende de β_t .
- Este ruido es lo que gradualmente empuja a la imagen hacia una distribución completamente aleatoria.

$$\sqrt{\beta_t}\varepsilon$$

2.1.1. Scheduled Beta

El parámetro de varianza β_t puede fijarse en una constante o elegirse como una escala a lo largo de los T intervalos de tiempo. De hecho, se puede definir una escala de varianza, que puede ser lineal, cuadrática, coseno, etc. Los autores originales del DDPM utilizaron una escala lineal que aumentaba de $\beta_1 = 10^{-4}$ a $\beta_T = 0.02$.

El beta schedule es importante porque controla la velocidad con la que la imagen se degrada, garantiza que el proceso directo se mantenga estable y bien definido, permite que el estado final x_T sea esencialmente ruido gaussiano y determina en gran medida la calidad y estabilidad del proceso inverso que el modelo debe aprender.

Por ejemplo, las representaciones latentes en el último cuarto del linear schedule se aproximan a ruido puro, mientras que el cosine schedule introduce ruido de manera más gradual.



Figura 4. Visualización de muestras latentes bajo scheduled lineal (arriba) y coseno (abajo).

Fuente: Nichol, A., & Dhariwal, P. (2021). Improved denoising diffusion probabilistic models. arXiv (Cornell University). <https://doi.org/10.48550/arxiv.2102.09672>



2.1.2. Fórmula Cerrada del Proceso Forward

Para simplificar la descripción matemática del proceso forward, se redefine el parámetro de ruido como $\alpha_t = 1 - \beta_t$. Esta sustitución permite expresar la transición entre pasos de manera más compacta. Con esta notación, el estado ruidoso en el paso t se describe como:

$$x_t = \sqrt{\alpha_t} x_{t-1} + \sqrt{1 - \alpha_t} \varepsilon$$

Esta ecuación muestra que el proceso forward combina una parte determinista, la señal residual de la imagen previa, con una componente estocástica representada por el ruido gaussiano.

A partir de esta formulación es posible expandir la ecuación de manera recursiva. Cada paso multiplica la señal restante por un nuevo factor $\sqrt{\alpha_t}$, de modo que, después de varias iteraciones, la señal acumulada corresponde al producto de todos los α_k aplicados hasta ese momento. Este producto se denomina alfa acumulado:

$$x_t = \underbrace{\sqrt{\alpha_t \alpha_{t-1} \dots \alpha_1}}_{\sqrt{\bar{\alpha}_t}} x_0 + (\text{combinación de ruidos}) \quad \bar{\alpha}_t = \prod_{k=1}^t \alpha_k$$

Gracias a esta acumulación es posible escribir una expresión cerrada del proceso forward:

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

Esta forma es especialmente útil porque permite calcular directamente el estado de la imagen en cualquier paso t sin necesidad de simular toda la cadena completa.



2.1.2. Fórmula Cerrada del Proceso Forward

2.1.2.1. Demostración

Para $t=1$

$$x_1 = \sqrt{\alpha_1}x_0 + \sqrt{1 - \alpha_1}\varepsilon_1 \quad \text{Aquí es claro: la señal } x_0 \text{ está multiplicada por } \sqrt{\alpha_1}.$$

Para $t=2$

$$x_2 = \sqrt{\alpha_2}x_1 + \sqrt{1 - \alpha_2}\varepsilon_2 \quad \text{y ahora sustituimos } x_1 : x_2 = \sqrt{\alpha_2}(\sqrt{\alpha_1}x_0 + \sqrt{1 - \alpha_1}\varepsilon_1) + \sqrt{1 - \alpha_2}\varepsilon_2$$

Distribuimos $\sqrt{\alpha_2}$:

$$x_2 = \underline{\sqrt{\alpha_2}\sqrt{\alpha_1}x_0} + \sqrt{\alpha_2}\sqrt{1 - \alpha_1}\varepsilon_1 + \sqrt{1 - \alpha_2}\varepsilon_2 \quad \text{Ya aparece el producto de alphas } \alpha_2\alpha_1 \text{ en: } \sqrt{\alpha_2}\sqrt{\alpha_1}x_0 = \sqrt{\alpha_2\alpha_1}x_0$$

Si se sigue este patrón, después de t pasos, la parte de señal se puede escribir como: $\sqrt{\alpha_t\alpha_{t-1}\dots\alpha_1}x_0$



2.1.2. Fórmula Cerrada del Proceso Forward

2.1.2.1. Demostración

Por otro lado, en cada paso se añade un nuevo ruido gaussiano independiente. Al expandir la ecuación de forma recursiva, la parte de ruido en x_t queda como una suma ponderada de muchos términos:

$$\alpha_1 \varepsilon_1 + \alpha_2 \varepsilon_2 + \cdots + \alpha_t \varepsilon_t$$

Dado que todas las ε_t son ruidos gaussianos e independientes, cualquier combinación lineal de ellas también es una variable gaussiana con media cero. Por esta razón, todos esos ruidos pueden agruparse en un único ruido equivalente

$$\alpha_1 \varepsilon_1 + \cdots + \alpha_t \varepsilon_t = \sqrt{\sigma^2} \varepsilon$$

La varianza total acumulada coincide exactamente con $1 - \bar{\alpha}_t$, lo que permite escribir la fórmula cerrada del proceso forward como se definió anteriormente:

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

2.1. Proceso de Difusión Inversa

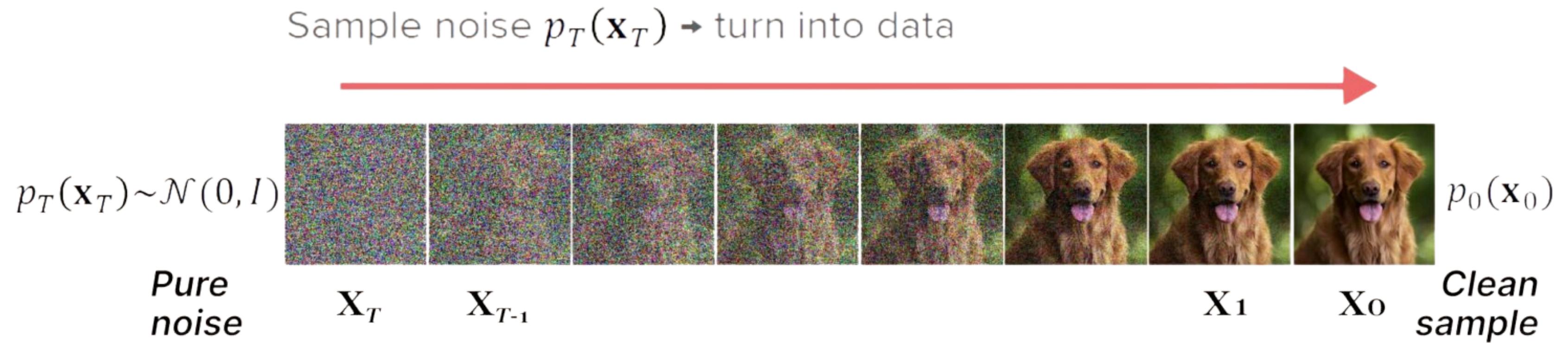


Figura 5. Secuencia del Reverse Process: Evolución desde ruido gaussiano hasta la recuperación de la estructura latente.
Fuente: Calibraint. (2024, 13 agosto). A Simple Guide to Diffusion Models. Calibraint. <https://www.calibraint.com/blog/beginners-guide-to-diffusion-models>

El proceso inverso constituye el núcleo generativo de los DDPM. Su objetivo es revertir la degradación impuesta durante el proceso forward y reconstruir una imagen paso a paso, comenzando desde ruido gaussiano puro. A diferencia del proceso directo, el proceso inverso **no puede calcularse de forma exacta, por lo que el modelo debe aprender a aproximarla**.

2.1. Proceso de Difusión Inversa

Durante el proceso forward, cada estado x_t se obtiene degradando la señal anterior y añadiendo ruido controlado. Para invertir esta cadena, el modelo necesita estimar el ruido que fue añadido en cada paso. Esa es la tarea de la red neuronal, generalmente un U-Net, que produce una predicción del ruido $\varepsilon_\theta(x_t, t)$. Con esa predicción, el modelo puede aproximar la distribución inversa $p_\theta(x_{t-1} | x_t)$ y reconstruir un estado menos ruidoso.

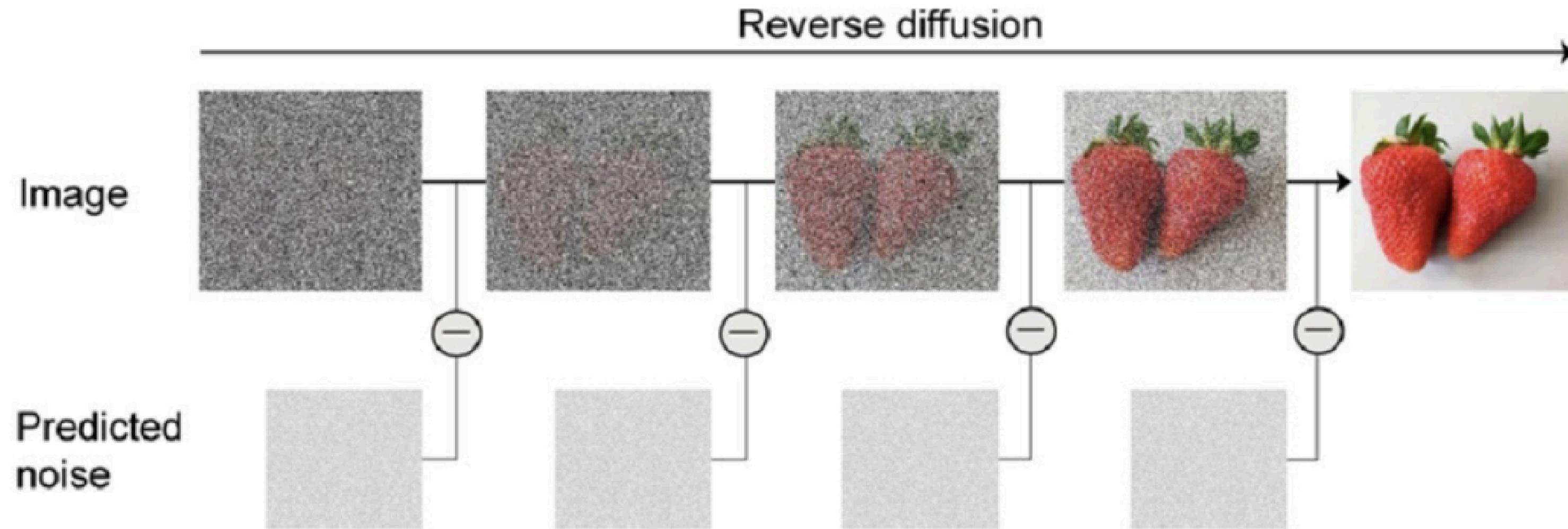


Figura 6. Predicción del Ruido Residual y Eliminación Iterativa para Reconstruir la Imagen Original.

Fuente: Yoon, S., Lee, Y., Jung, E., & Ahn, T. I. (2024). Agricultural Applicability of AI based Image Generation. Journal Of Bio-Environment Control, 33(2), 120-128. <https://doi.org/10.12791/ksbec.2024.33.2.120>

2.1. Proceso de Difusión Inversa

A continuación, se muestra la fórmula que describe el paso que permite reconstruir x_{t-1} a partir de x_t , invirtiendo el proceso de difusión al eliminar el ruido estimado por el modelo y recuperar progresivamente la señal.

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

donde:

$\frac{1}{\sqrt{\alpha_t}}$ Es el inverso del factor con el que el forward process redujo la señal en el paso t .

- “Deshace” la atenuación aplicada durante el proceso forward.
- Si el forward multiplicó la señal por t , entonces el reverse debe dividirla por ese mismo factor para recuperar su escala.

x_t El estado ruidoso actual en el paso t .

- Es el punto de partida desde el cual queremos “retroceder” al estado menos ruidoso x_{t-1} .

$\varepsilon_\theta(x_t, t)$ Predicción del ruido hecha por el modelo (UNet).

- Es el punto de partida desde el cual queremos “retroceder” al estado menos ruidoso x_{t-1} .

$\frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}}$ Coeficiente determinista derivado del forward process.

- $1 - \alpha_t$: cantidad de ruido añadido en el paso t .
- $1 - \bar{\alpha}_t$: ruido acumulado hasta el paso t .
- Escala la predicción del ruido para eliminar exactamente la parte de ruido que el forward añadió en ese paso.

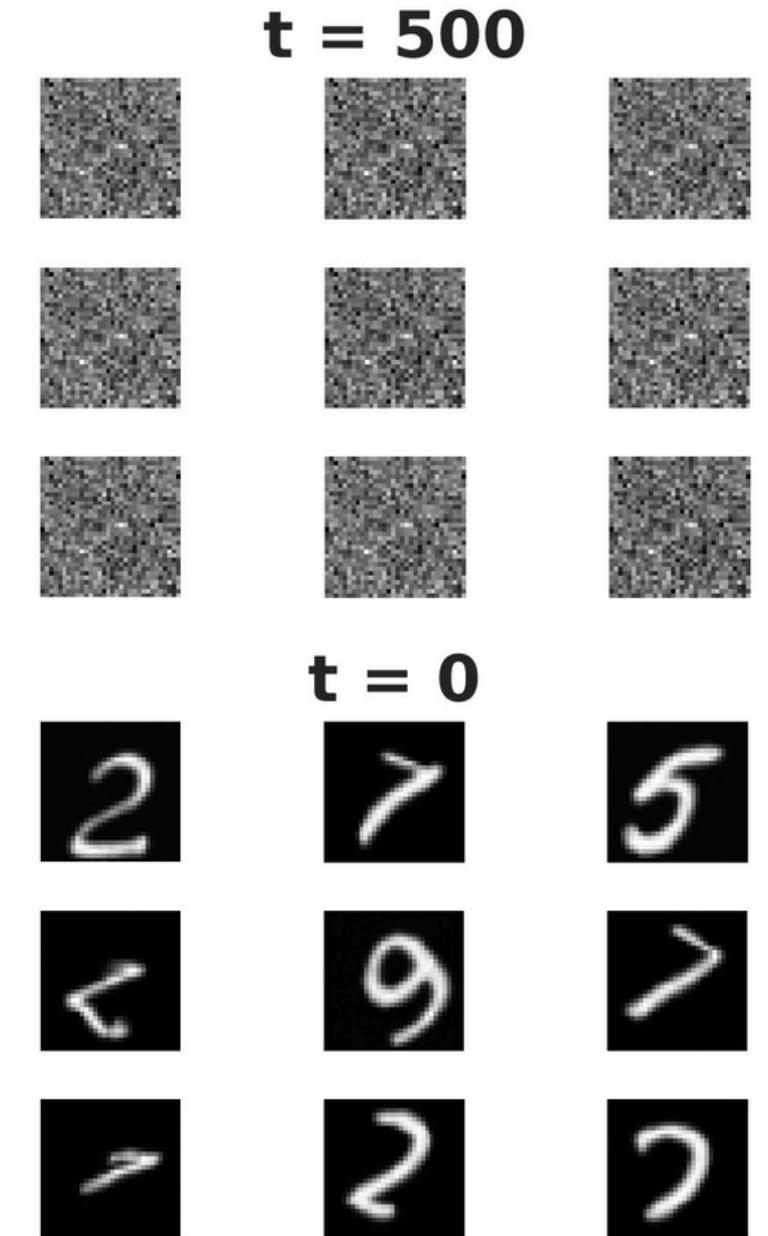


Figura 7. Visualización de imágenes derivadas de idéntico ruido gaussiano en $t = 500$.
Fuente: Higham, C. F., Higham, D. J., & Grindrod, P. (2023). Diffusion Models for Generative Artificial Intelligence: An Introduction for Applied Mathematicians. arXiv (Cornell University). <https://doi.org/10.48550/arxiv.2312.14977>

2.1. Proceso de Difusión Inversa



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

Intento del modelo de reconstruir la señal limpia antes del paso t.

- Quita al estado ruidoso x_t la cantidad estimada de ruido correspondiente a ese paso.

$$\frac{1}{\sqrt{\alpha_t}}(\dots)$$

Reescala la señal corregida por la predicción del ruido.

- Recupera la escala de señal que el forward había comprimido.

$$\sigma_t z$$

Términos Estocásticos en el Proceso Inverso.

- Recupera la escala de señal que el forward había comprimido.
- z es ruido gaussiano estándar.
 - No corresponde al ruido del forward, es un ruido nuevo que proviene de la varianza del proceso inverso.
- σ_t . Controla cuánta estocasticidad introducir.
 - Es un coeficiente determinista que controla la magnitud del ruido injectado durante el paso inverso.

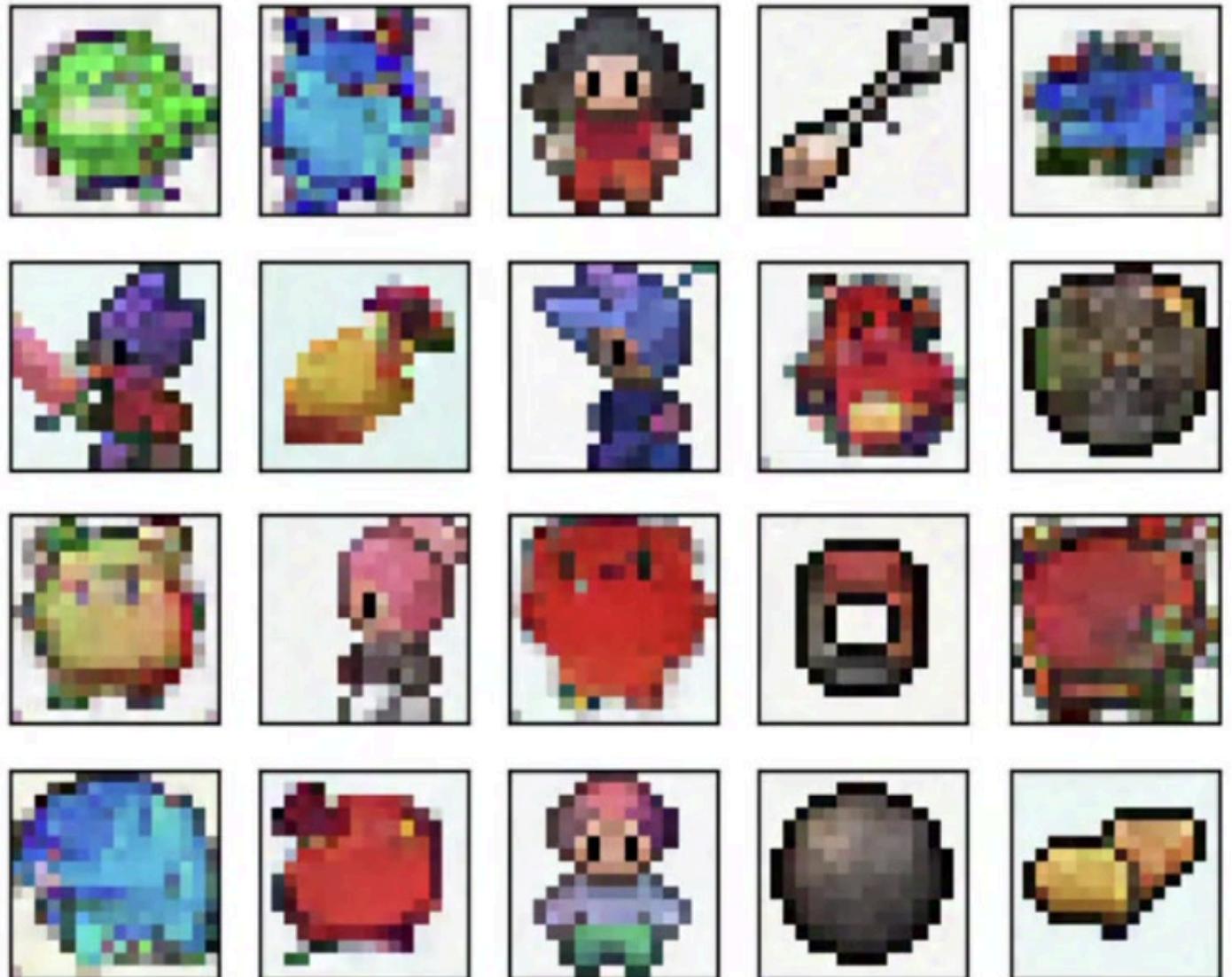


Figura 8. Proceso de difusión inversa: transición desde ruido gaussiano hasta la reconstrucción de imágenes de Pixel Art.

Fuente: Elgazar, E. (2023). Diffusion model U-Net [Notebook]. Kaggle.
<https://www.kaggle.com/code/ebrahimelgazar/diffusion-model-u-net/notebook>

ARQUITECTURA DEL MODELO

La arquitectura utilizada en los modelos de difusión está diseñada específicamente para aproximar el proceso inverso, es decir, para predecir el ruido añadido en cada paso del forward. El componente central es una U-Net, una arquitectura convolucional con forma de “U” que combina compresión y expansión espacial, junto con un sistema de embeddings temporales que informan al modelo en qué paso de difusión se encuentra.

La U-Net recibe como entrada una imagen ruidosa xt y el paso de tiempo t , y devuelve una estimación del ruido $\varepsilon_\theta(x_t, t)$. Está compuesta por:

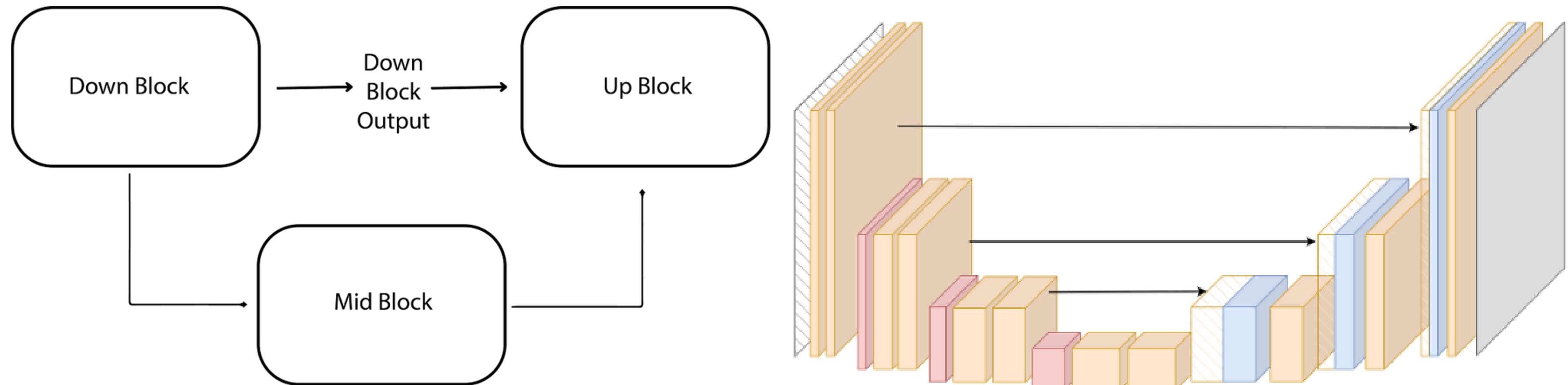


Figura 9. Representación de la arquitectura U-Net empleada como base en el proyecto de difusión.

Fuente: Sayed, E. (2024, diciembre 6). DDPM PyTorch implementation from scratch. Medium. <https://medium.com/@sayedebad.777/ddpm-pytorch-implementation-from-scratch-36b647f5dd82>

1. Time Embedding

En los modelos de difusión, la red debe procesar imágenes que contienen distintos niveles de ruido según el paso temporal t . Por esta razón, el modelo necesita una representación explícita del tiempo que le indique cuánto ruido ha sido aplicado y cómo debe ajustar su comportamiento en ese paso. Este rol lo cumplen los embeddings temporales.

Primero, t se codifica mediante un embedding sinusoidal fijo, que captura de forma continua la posición temporal. Luego, este embedding se pasa por una MLP entrenable, que genera una representación más expresiva. El embedding final se inyecta en los bloques de la U-Net, modulando sus activaciones para que el modelo procese correctamente cada nivel de ruido.

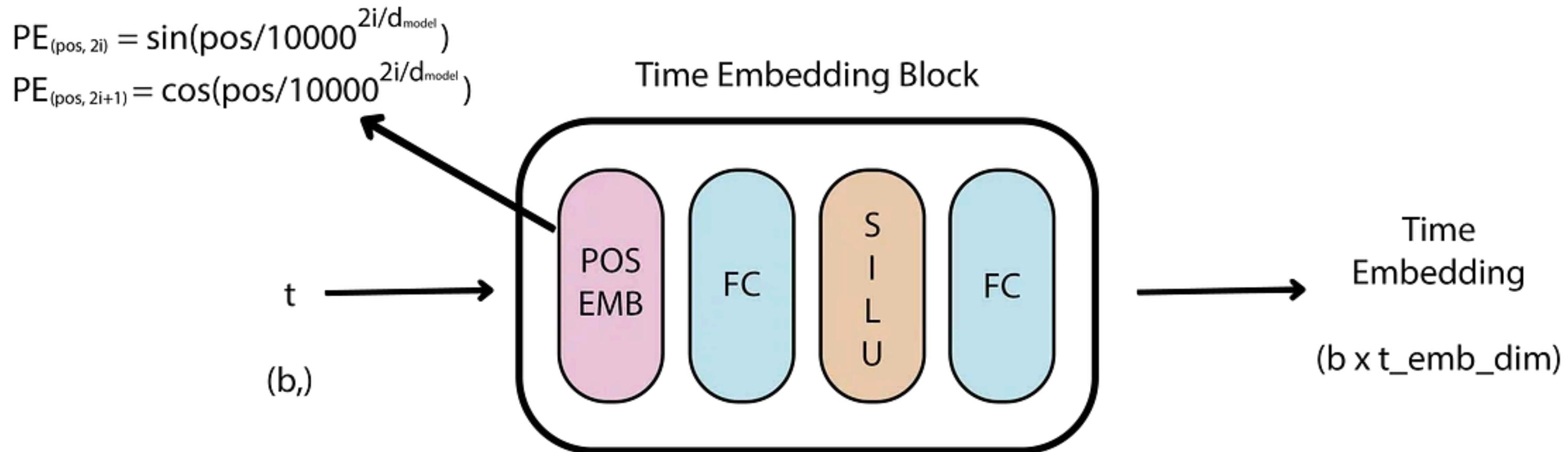


Figura 10. Representación del mecanismo de time embedding empleado para incorporar la información de los pasos de ruido.

Fuente: Sayed, E. (2024, diciembre 6). DDPM PyTorch implementation from scratch. Medium. <https://medium.com/@sayedebad.777/ddpm-pytorch-implementation-from-scratch-36b647f5dd82>

2. Down Block (Encoder)

El Down Block es la parte de la U-Net encargada de reducir la resolución de la imagen ruidosa mientras aumenta la profundidad de los canales, permitiendo que el modelo capture información más global y estructuras de mayor escala.

ResBlocks: Procesan la imagen ruidosa extrayendo patrones y detalles locales, mientras usan el embedding temporal para ajustar su comportamiento según el nivel de ruido del paso t .

Self-Attention: Analiza relaciones entre píxeles distantes para que el modelo entienda la estructura global de la imagen y mantenga coherencia entre regiones separadas.

Downsampling: Reduce el tamaño espacial de la imagen para que la red pueda concentrarse en información más global y aprender representaciones más abstractas de alto nivel.

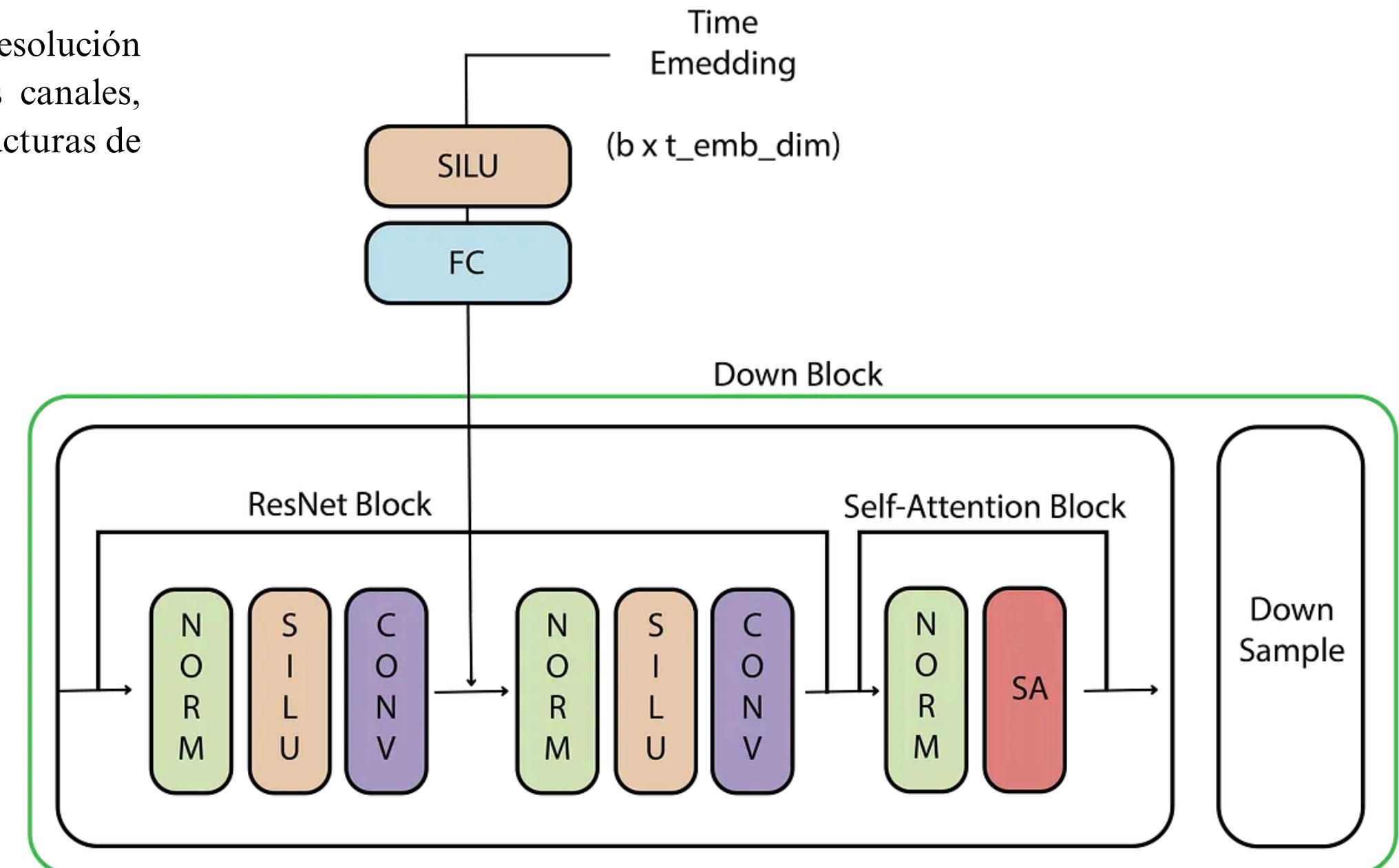


Figura 11. Esquema del Down Block de U-Net, mostrando las operaciones de convolución y reducción espacial.
Fuente: Sayed, E. (2024, diciembre 6). DDPM PyTorch implementation from scratch. Medium. <https://medium.com/@sayedebad.777/ddpm-pytorch-implementation-from-scratch-36b647f5dd82>

3. Mid Block

El Mid Block es la parte más profunda de toda la U-Net, donde la representación de la imagen llega a su punto de mayor compresión. Aquí, la red ya no trabaja con la imagen en su forma original, sino con un mapa muy reducido que contiene la información esencial extraída por todos los Down Blocks anteriores.

En este nivel, el modelo combina dos operaciones clave:

- **Convoluciones (ResBlocks):** siguen refinando la información comprimida, ayudando a distinguir qué detalles son realmente importantes y cuáles pueden ignorarse.
- **Self-Attention:** permite que el modelo relacione regiones distantes dentro de este mapa comprimido, integrando la información para tener una “visión completa” del contenido, incluso si partes que antes estaban lejos ahora están juntas por la compresión.

La función principal del Mid Block es organizar y entender la información global de la imagen antes de comenzar la etapa de reconstrucción.

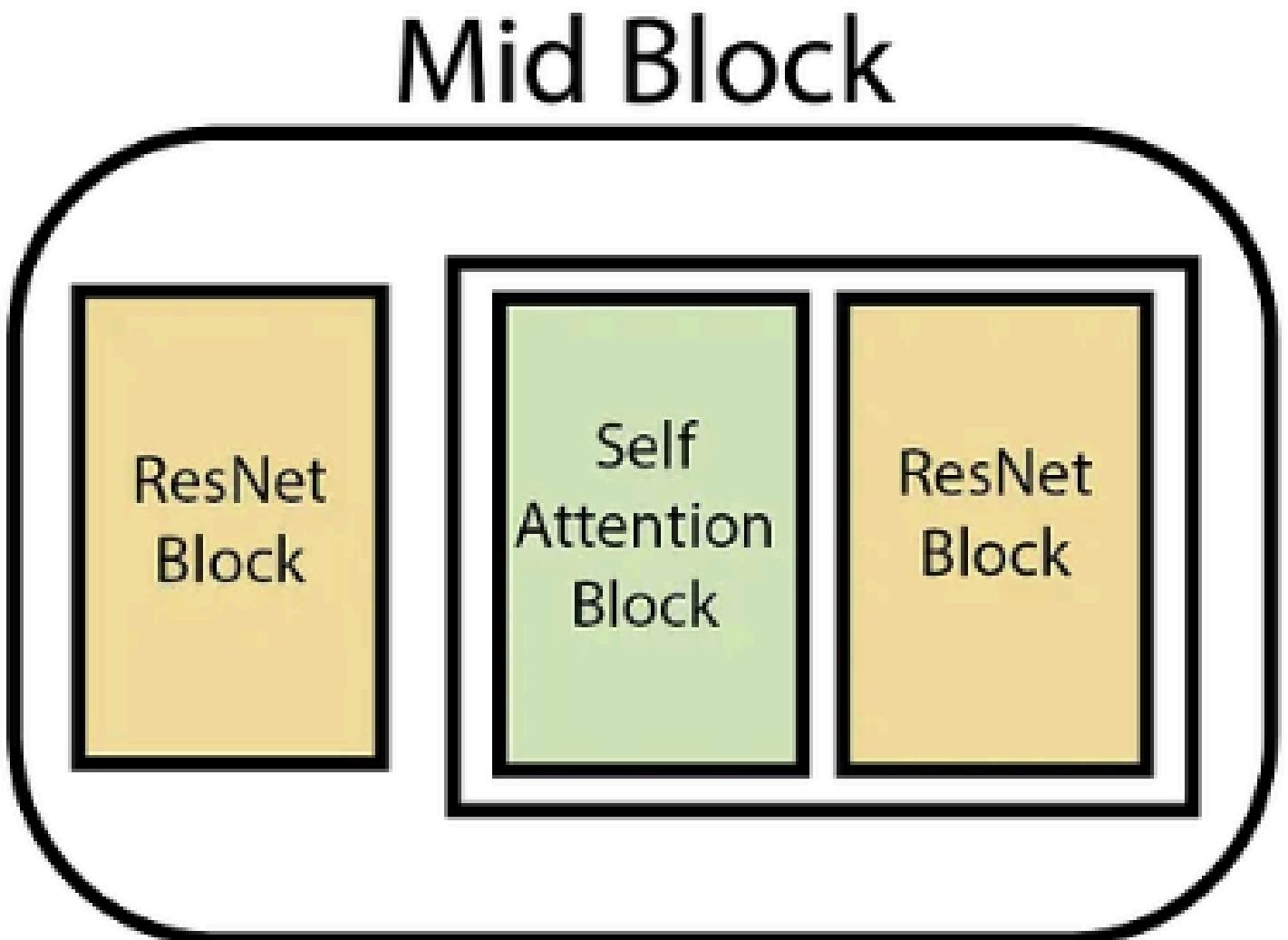


Figura 12. Esquema del Middle Block de U-Net, encargado de procesar las representaciones latentes antes de la etapa de decodificación.
Fuente: Sayed, E. (2024, diciembre 6). DDPM PyTorch implementation from scratch. Medium. <https://medium.com/@sayedebad.777/ddpm-pytorch-implementation-from-scratch-36b647f5dd82>

4. Up Block (Decoder)

El Up Block reconstruye la imagen aumentando la resolución y recuperando detalles que se guardaron durante el descenso. Sus pasos principales son:

- **Upsampling:** aumenta la resolución del mapa proveniente del Mid Block.
- **Skip Connection:** combina ese mapa ampliado con la salida del Down Block correspondiente, recuperando detalles locales.
- **ResBlocks:** refinan la información fusionada para integrar correctamente detalles locales y estructura global.
- **Self-Attention:** permite que el modelo relacione áreas alejadas entre sí, ayudando a mantener una estructura coherente y correctamente conectada mientras la imagen se reconstruye.

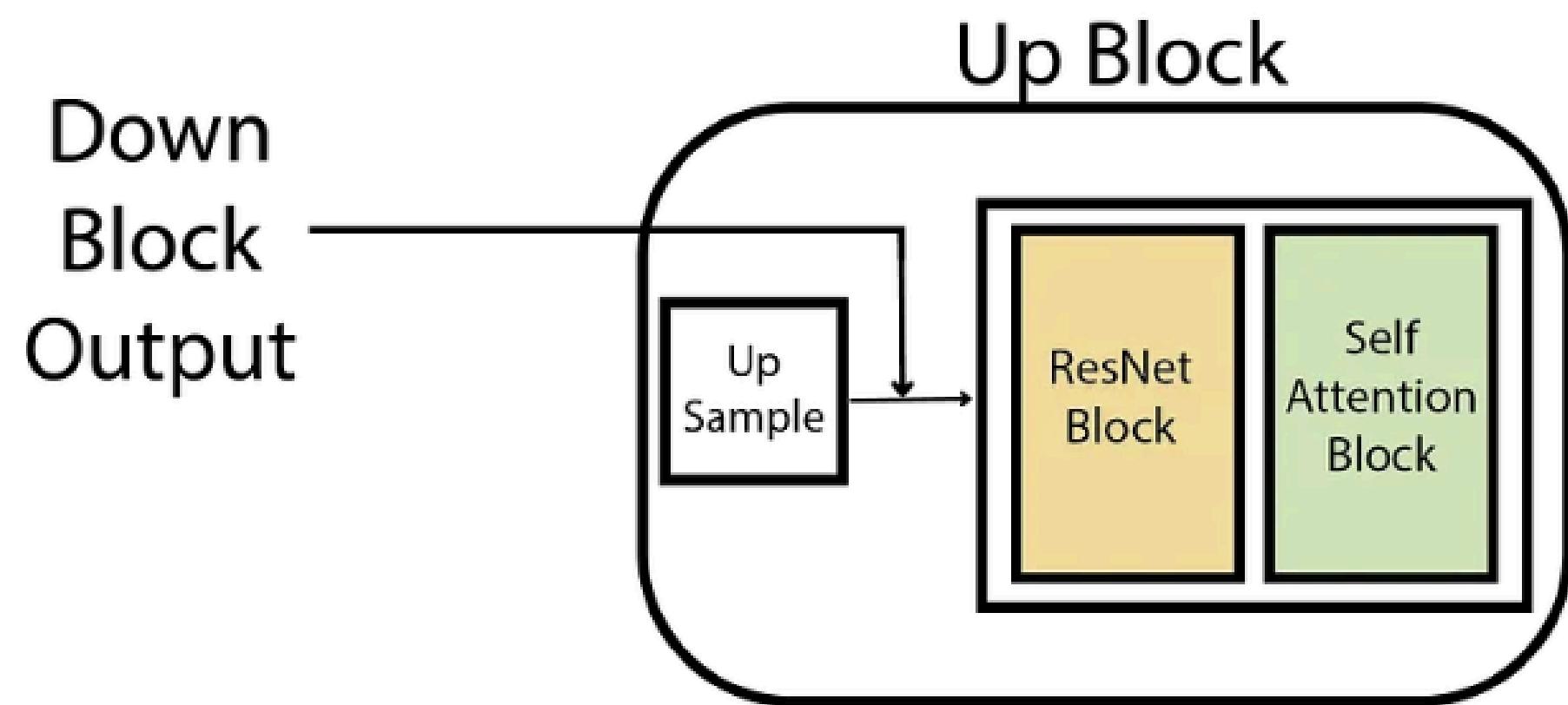


Figura 13. Representación del Up Block en U-Net, encargado de la decodificación y reconstrucción progresiva de la imagen.
Fuente: Sayed, E. (2024, diciembre 6). DDPM PyTorch implementation from scratch. Medium. <https://medium.com/@sayedebad.777/ddpm-pytorch-implementation-from-scratch-36b647f5dd82>

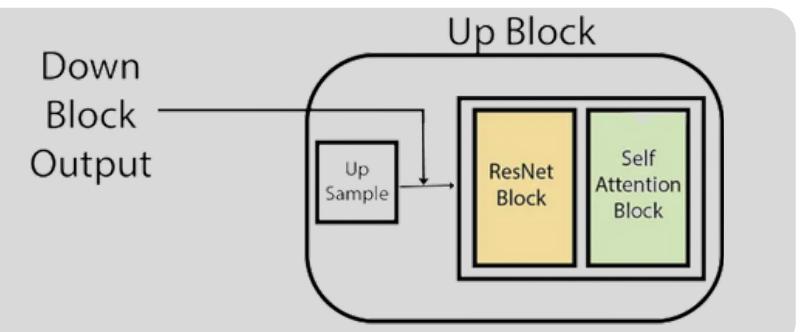


5. Skip Connections

Las skip connections son conexiones directas que permiten que cierta información se salte operaciones intermedias y llegue más adelante en la red sin perderse. En una U-Net y en los DDPM, estas conexiones aparecen en los siguientes niveles:

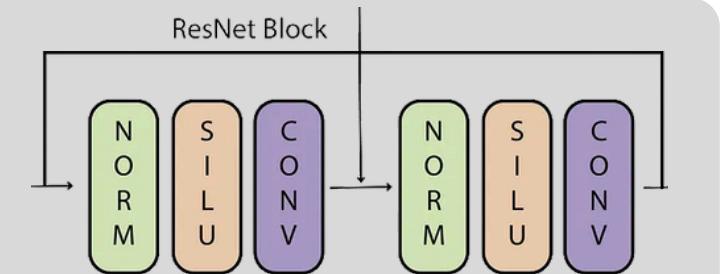
1. Skip connections entre Down Blocks y Up Blocks (nivel U-Net)

- Permiten recuperar detalles finos que se pierden al bajar la resolución.
- Permiten que el decoder reconstruya texturas, bordes y formas sin depender solo de la representación comprimida.
- Evitan la pérdida de información importante durante el paso por la parte profunda de la red.



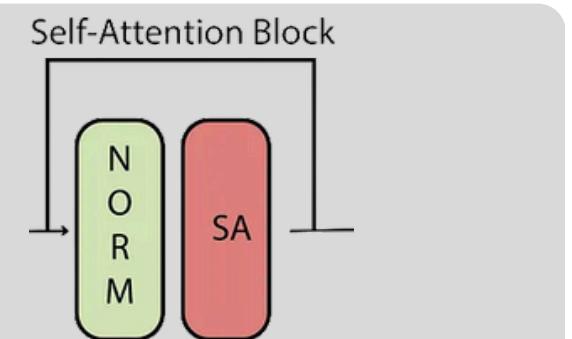
2. Skip Connection en ResNet Block

La salida del bloque se suma a su entrada original para conservar información importante y permitir que el bloque aprenda solo la corrección necesaria, manteniendo estabilidad y evitando pérdida de detalles.



3. Skip Connection en Self-Attention Block

Después de aplicar la atención, la salida se suma nuevamente a la entrada inicial, combinando información global con la original sin reemplazarla, de modo que se mantienen los detalles locales mientras se añade contexto global.



DATASET PICOBANANA-400K

Pico-Banana-400K es un conjunto de datos a gran escala, publicado por Apple en octubre de 2025, diseñado específicamente para avanzar la investigación en edición de imágenes guiada por texto. Contiene aproximadamente 400 mil tripletas compuestas por una imagen original extraída de Open Images, una instrucción de edición en lenguaje natural y la versión editada correspondiente. Las ediciones fueron generadas por el modelo Nano-Banana y evaluadas mediante un sistema automático basado en Gemini-2.5-Pro.

El dataset cubre 35 tipos de operaciones agrupadas en 8 categorías semánticas, que van desde ajustes fotométricos simples hasta transformaciones complejas de objetos, escenas, estilo y composición. Incluye imágenes con resolución entre 512 y 1024 píxeles, abarcando un amplio espectro visual: objetos, personas, texto, escenas urbanas y contenido variado del mundo real.

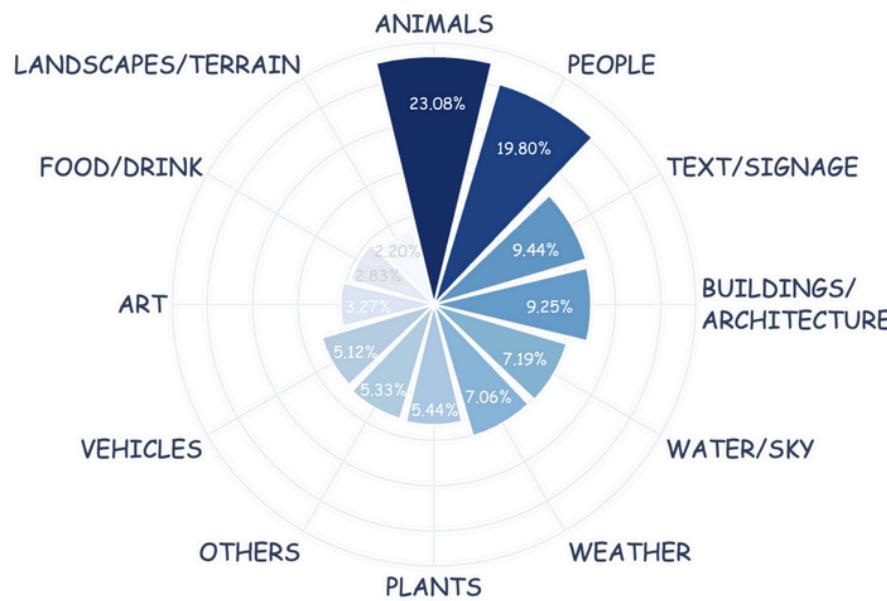


Figura 15. Distribución del contenido de las instrucciones de edición de imágenes en el dataset PicoBanana-400K.

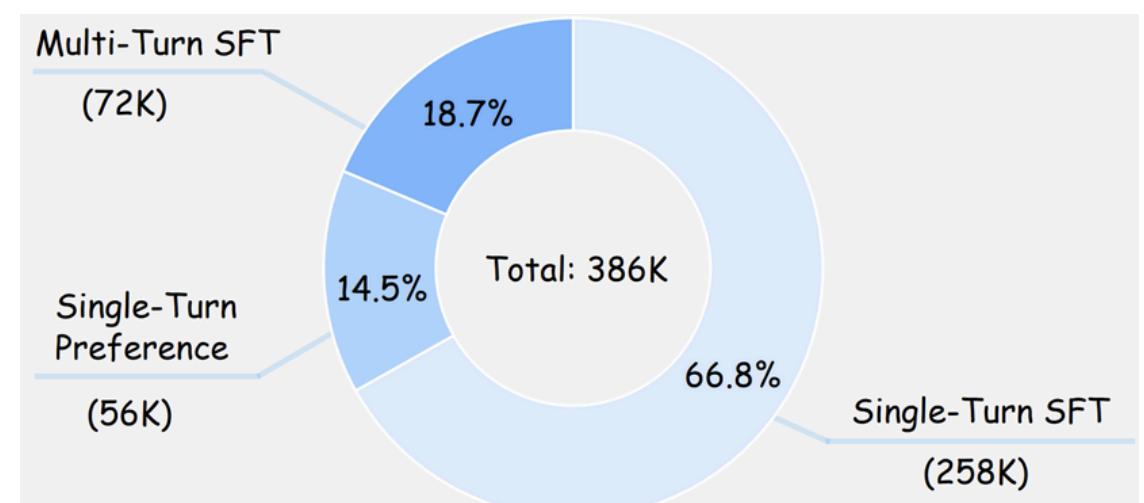


Figura 16. Visualización de la composición del dataset PicoBanana-400K según los tipos de datos de entrenamiento utilizados.

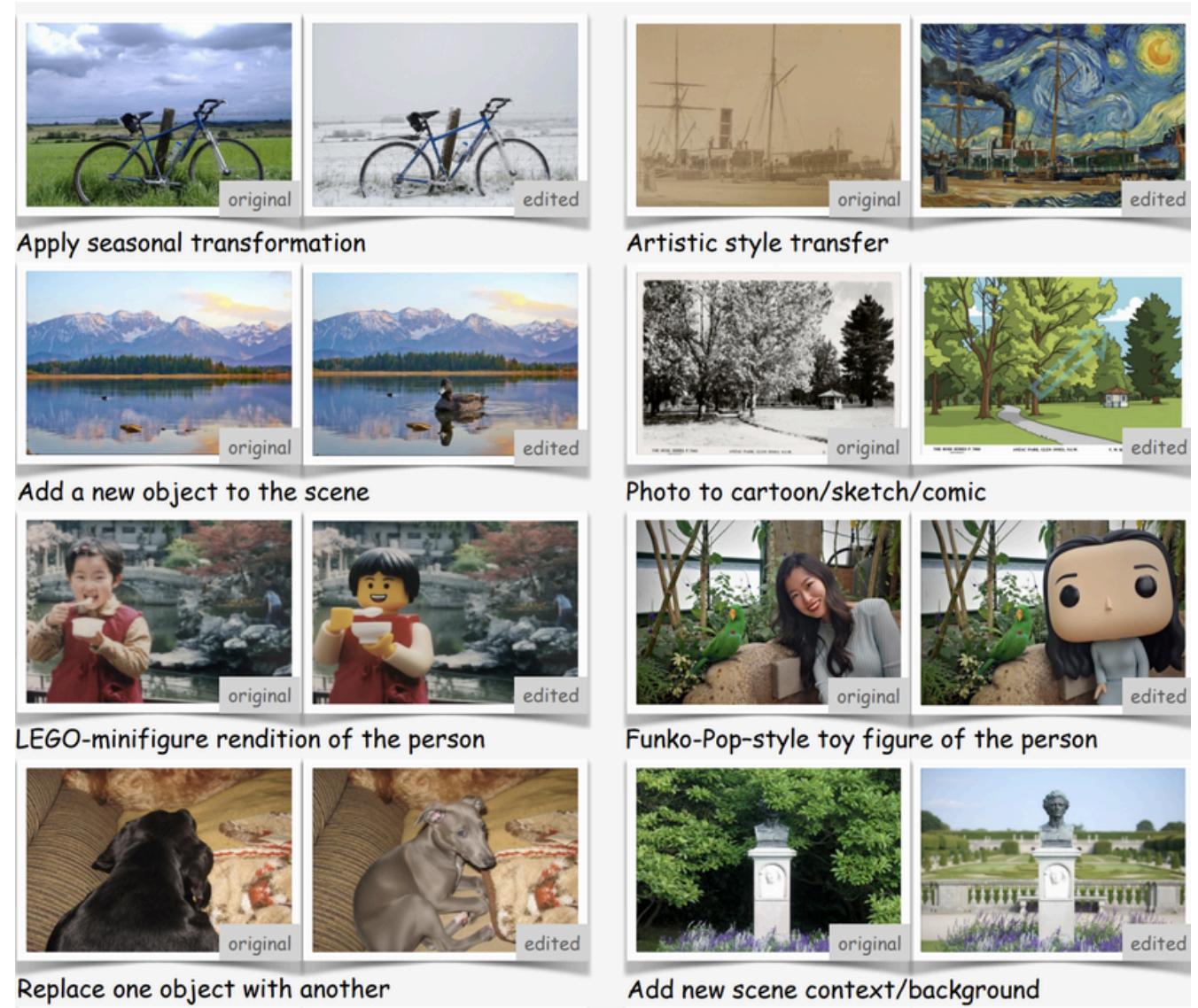


Figura 14. Ejemplo ilustrativo de las categorías del dataset PicoBanana-400K y su proceso de transformación.
Fuente: Song, L., Yang, Y., Lu, J., Hu, W., & Gan, Z. (2025). Pico-Banana-400K: A Large-Scale Dataset for Text-Guided Image Editing. arXiv (Cornell University). <https://doi.org/10.48550/arxiv.2510.19808>

1. Imágenes de Muestra del Dataset

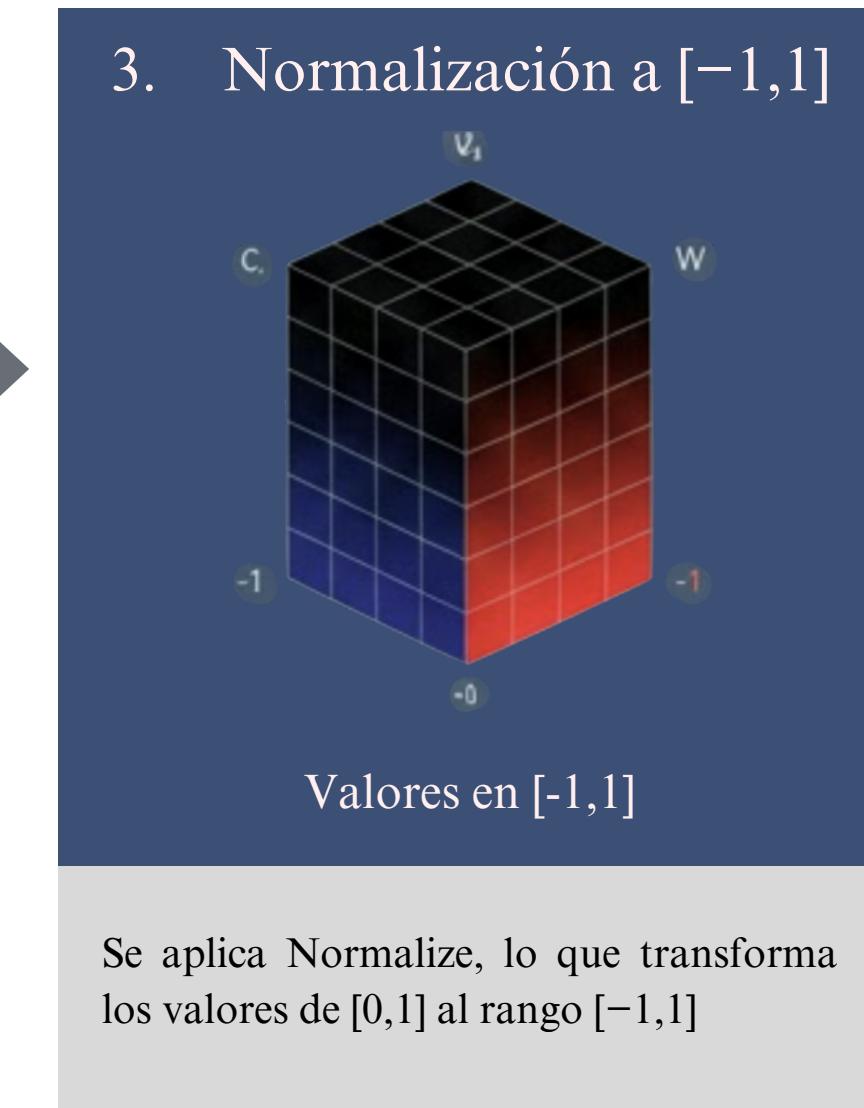
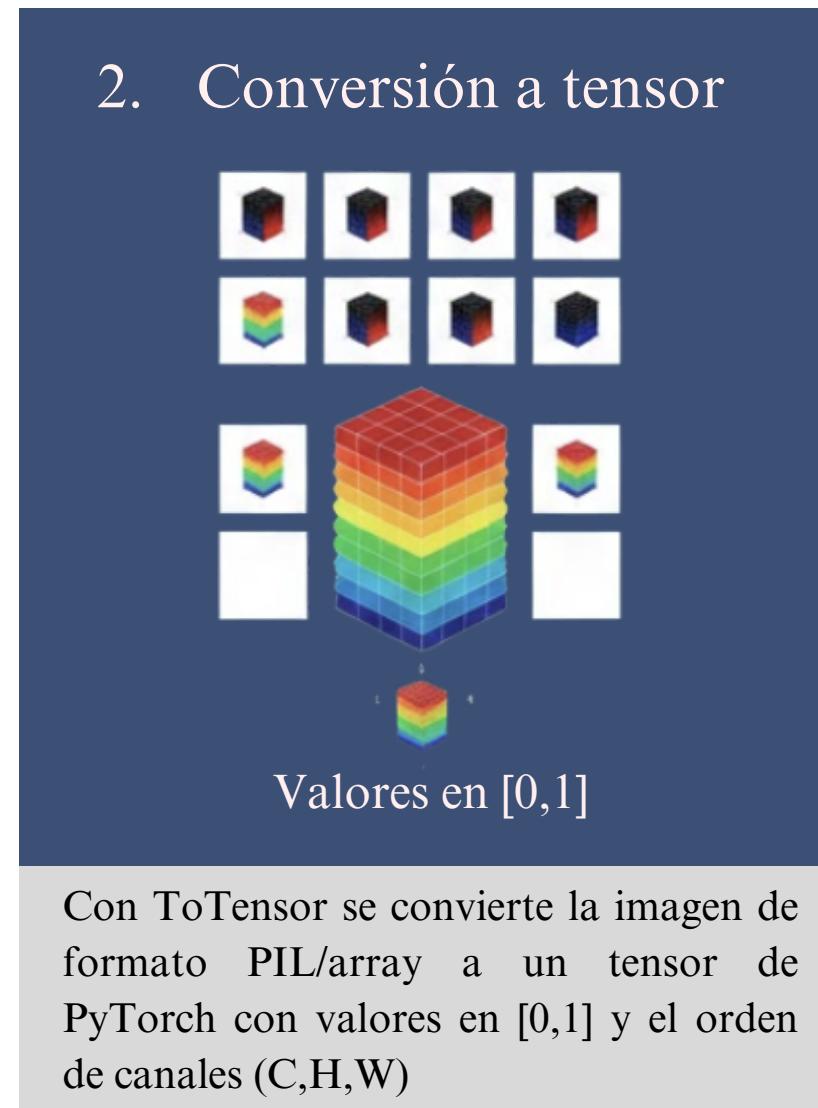
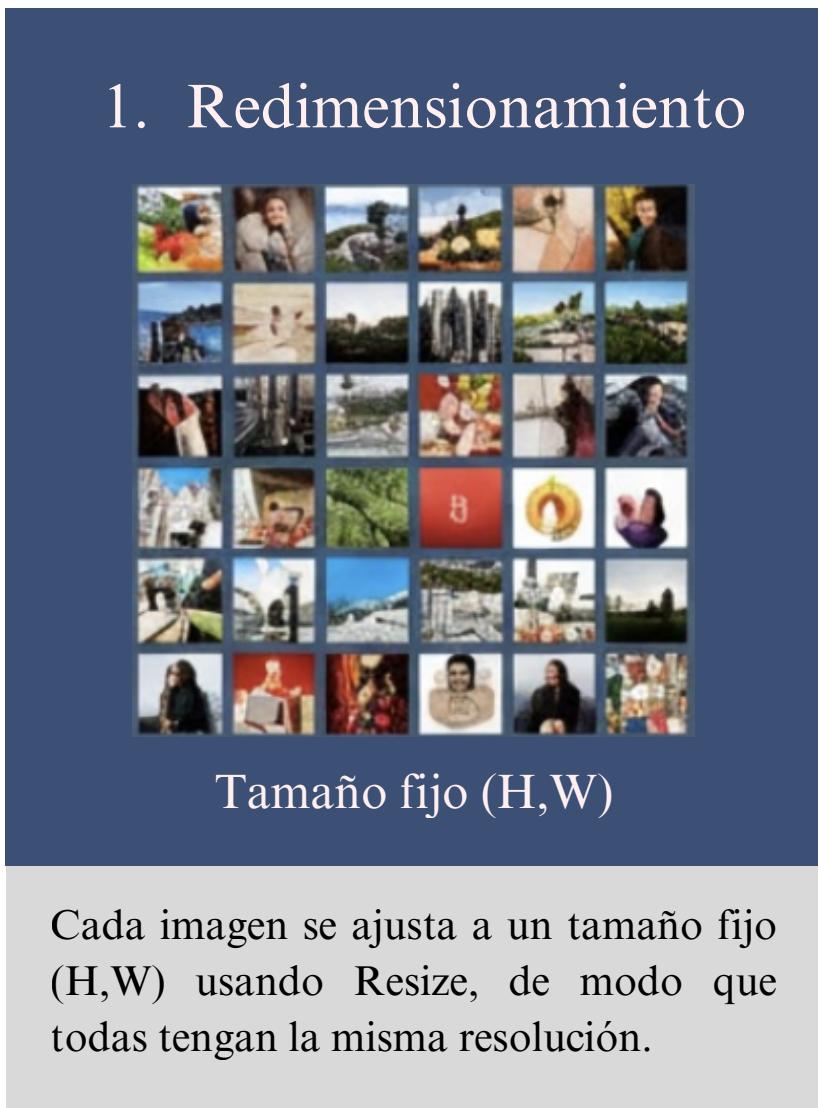


Figura 17. Conjunto de imágenes seleccionadas del dataset PicoBanana-400K que reflejan su amplitud y diversidad visual.

Fuente: Song, L., Yang, Y., Lu, J., Hu, W., & Gan, Z. (2025). Pico-Banana-400K: A Large-Scale Dataset for Text-Guided Image Editing. arXiv (Cornell University). <https://doi.org/10.48550/arxiv.2510.19808>

ENTRENAMIENTO

1. Preparación del dato



2. Generación de x_t

Una vez que las imágenes limpias x_0 han sido transformadas, se construyen los tripletes (x_t, t, ϵ) que se utilizan para entrenar el modelo.

- Para cada imagen del batch, primero se genera un tensor de ruido gaussiano ϵ con la misma forma que la imagen.
- Luego, se elige un paso temporal t de manera aleatoria entre 0 y $T-1$. Esto hace que el modelo entrene con distintos niveles de degradación, desde imágenes casi limpias hasta imágenes muy ruidosas.
- Con la imagen original x_0 , el ruido ϵ y el tiempo t , el módulo de difusión forward calcula usando la fórmula cerrada del proceso: esta ecuación permite obtener la imagen ruidosa exacta correspondiente al paso t sin simular todos los pasos anteriores.

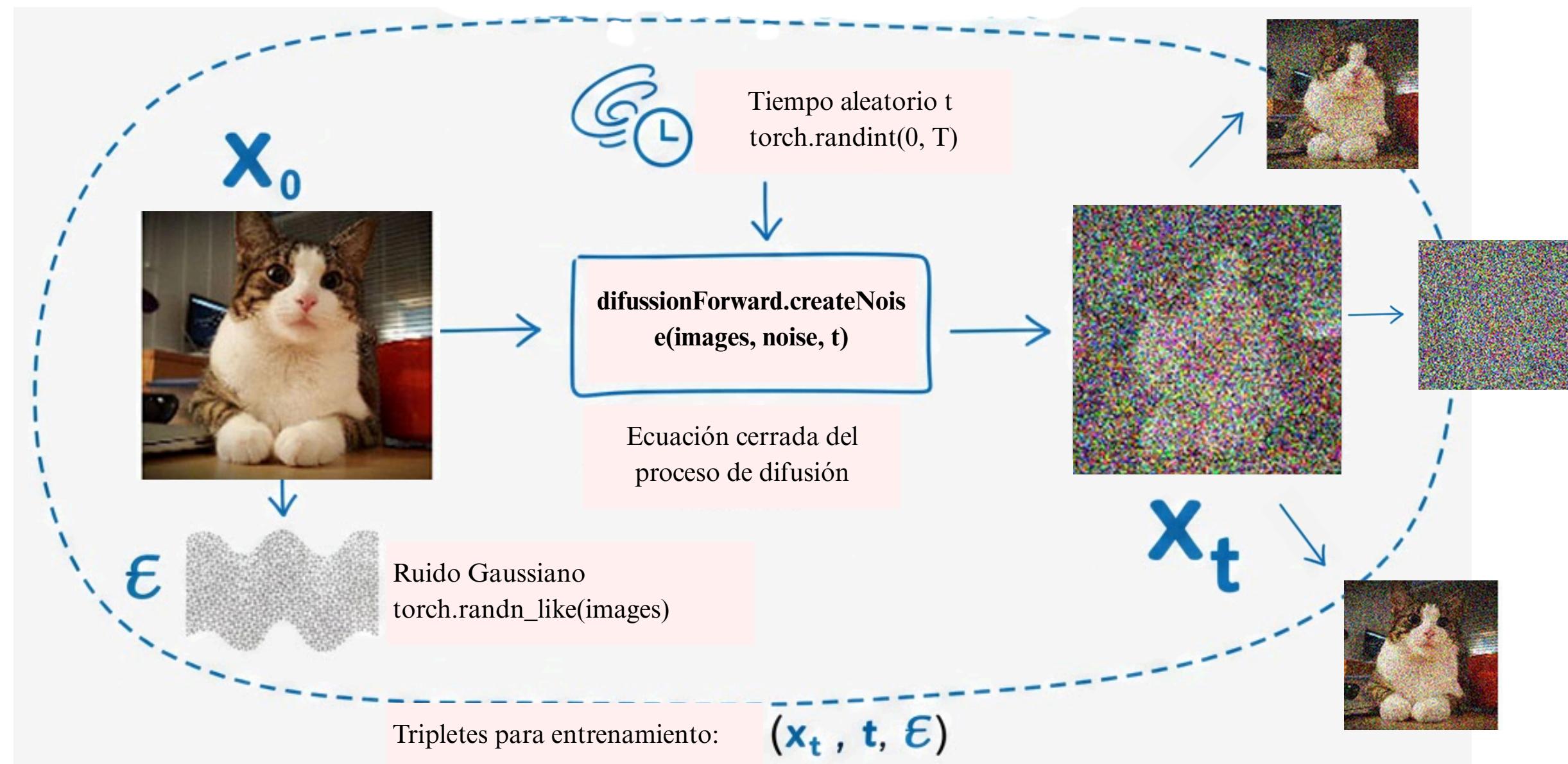


Figura 18. Diagrama de Flujo del Proceso 'Forward' de Difusión: Construcción del triplete para el Entrenamiento Denoising.
Fuente: Imagen generada por Google Gemini y modificada por los autores. <https://gemini.google.com/share/2a196ca0fcf2>

3. Predicción del Ruido y Pérdida

Una vez generados los tripletes (x_t, t, ϵ) , el modelo recibe la imagen ruidosa junto con el embedding temporal y produce una estimación del ruido añadido en ese paso, $\epsilon_\theta(x_t, t)$. Esta predicción es el elemento central del entrenamiento, ya que será comparada con el ruido real ϵ para evaluar qué tan bien está aprendiendo la U-Net a reconocer y separar el ruido de la estructura original de la imagen.

Para entrenar esta habilidad, se utiliza una función de pérdida basada en el error cuadrático medio entre el ruido real y el ruido predicho: $\mathcal{L} = \mathbb{E}[\|\epsilon - \epsilon_\theta(x_t, t)\|^2]$

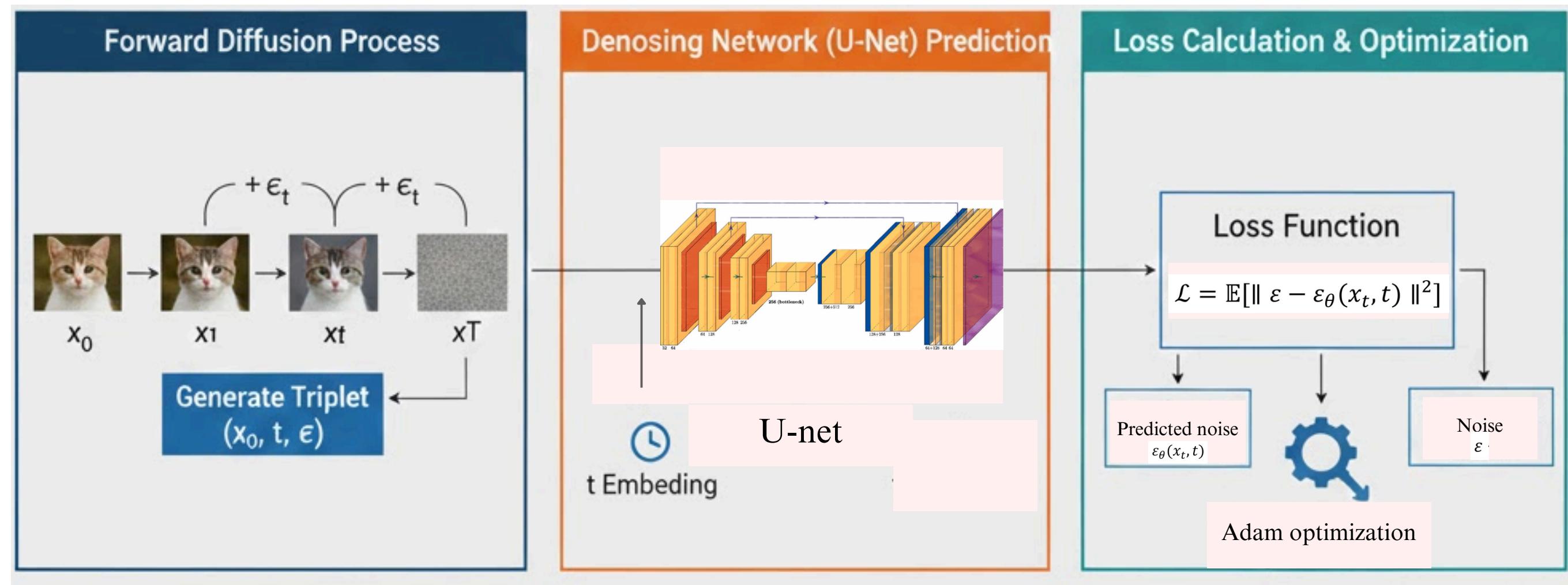


Figura 19. Visión General del Proceso de Entrenamiento DDPM. Fuente: Imagen generada por Google Gemini y modificada por los autores.

Fuente: Imagen generada por Google Gemini y modificada por los autores. <https://gemini.google.com/sh>

4. Predicción del Ruido y Pérdida

En su formulación original, los modelos de difusión se entrenaban optimizando la ELBO (Evidence Lower Bound), que es una función que proviene de la teoría de modelos generativos probabilísticos. La ELBO mide qué tan bien un modelo puede explicar los datos observados bajo un proceso probabilístico complejo. La ELBO intenta asegurar que el modelo imite todo el comportamiento probabilístico del forward y del reverse process.

Sin embargo, la ELBO completa es larga, difícil de calcular en la práctica y contiene muchos términos que aportan muy poco al entrenamiento.

El trabajo de Ho et al. (2020) mostró algo clave:

1. El término más importante de la ELBO, y el que realmente influye en la calidad de generación, es la diferencia entre el ruido real ε y el ruido predicho ε_θ
2. Los otros términos de la ELBO son constantes o tienen muy poco efecto, por lo que optimizarlos no cambia sustancialmente los resultados.
3. Minimizar el MSE del ruido es matemáticamente equivalente al término dominante del ELBO, pero mucho más estable y simple de entrenar.

Loss basada en ELBO

$$\mathcal{L}_{\text{ELBO}} = \mathbb{E} \left[D_{\text{KL}}(q(x_T | x_0) \| p(x_T)) + \sum_{t=2}^T D_{\text{KL}}(q(x_{t-1} | x_t, x_0) \| p_\theta(x_{t-1} | x_t)) - \log p_\theta(x_0 | x_1) \right]$$

Loss simplificada

$$\mathcal{L}_{\text{simple}} = \mathbb{E} [\|\varepsilon - \varepsilon_\theta(x_t, t)\|^2]$$

INFERENCIA

Durante la inferencia, el modelo ya no recibe imágenes reales. En su lugar, comienza desde ruido puro x_T y aplica el proceso inverso de difusión paso a paso hasta obtener una imagen limpia. A diferencia del entrenamiento, aquí sí se reconstruyen imágenes utilizando la ecuación del reverse process.

1. Se crea un tensor x_T muestreado de una distribución gaussiana. Esta es la imagen totalmente ruidosa desde la que comenzará la generación.
2. En cada timestep, el modelo recibe x_t y el embedding temporal t , y produce una estimación del ruido $\epsilon_\theta(x_t, t)$.
3. Usando la predicción del ruido, se calcula x_{t-1} mediante la fórmula del reverse process, que elimina parte del ruido y ajusta la escala para imitar el paso anterior del proceso forward.
4. Se agrega $\sigma_t z$, un pequeño ruido gaussiano, que introduce variabilidad para permitir generar múltiples imágenes diferentes desde el mismo punto inicial.
5. El ciclo continúa desde $t=T$ hasta $t=1$. Después del último paso, el modelo obtiene x_0 , una imagen limpia totalmente generada a partir de ruido.

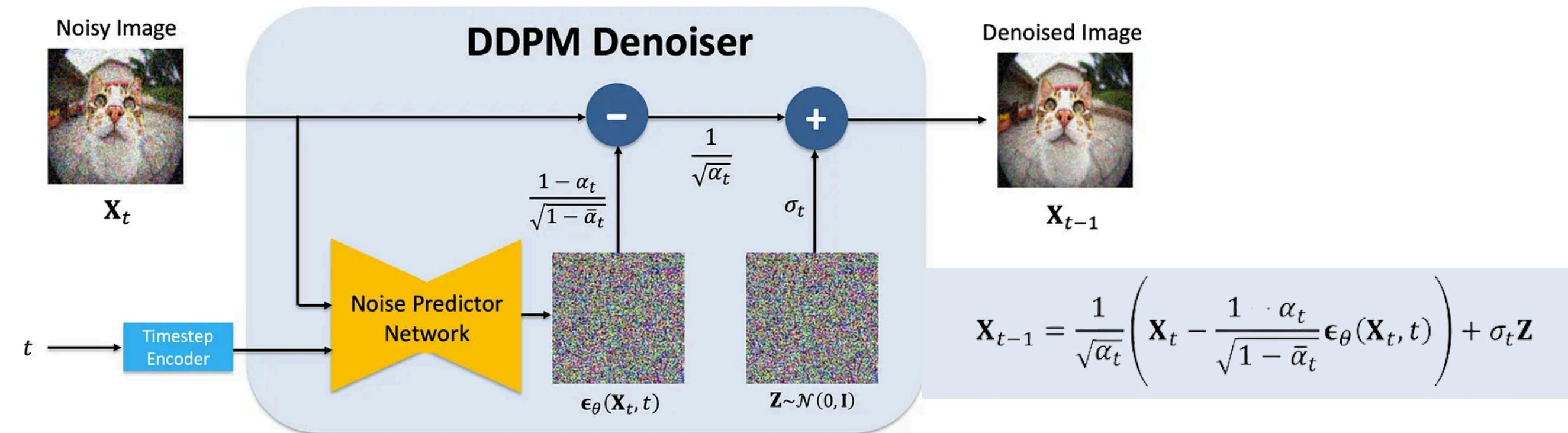
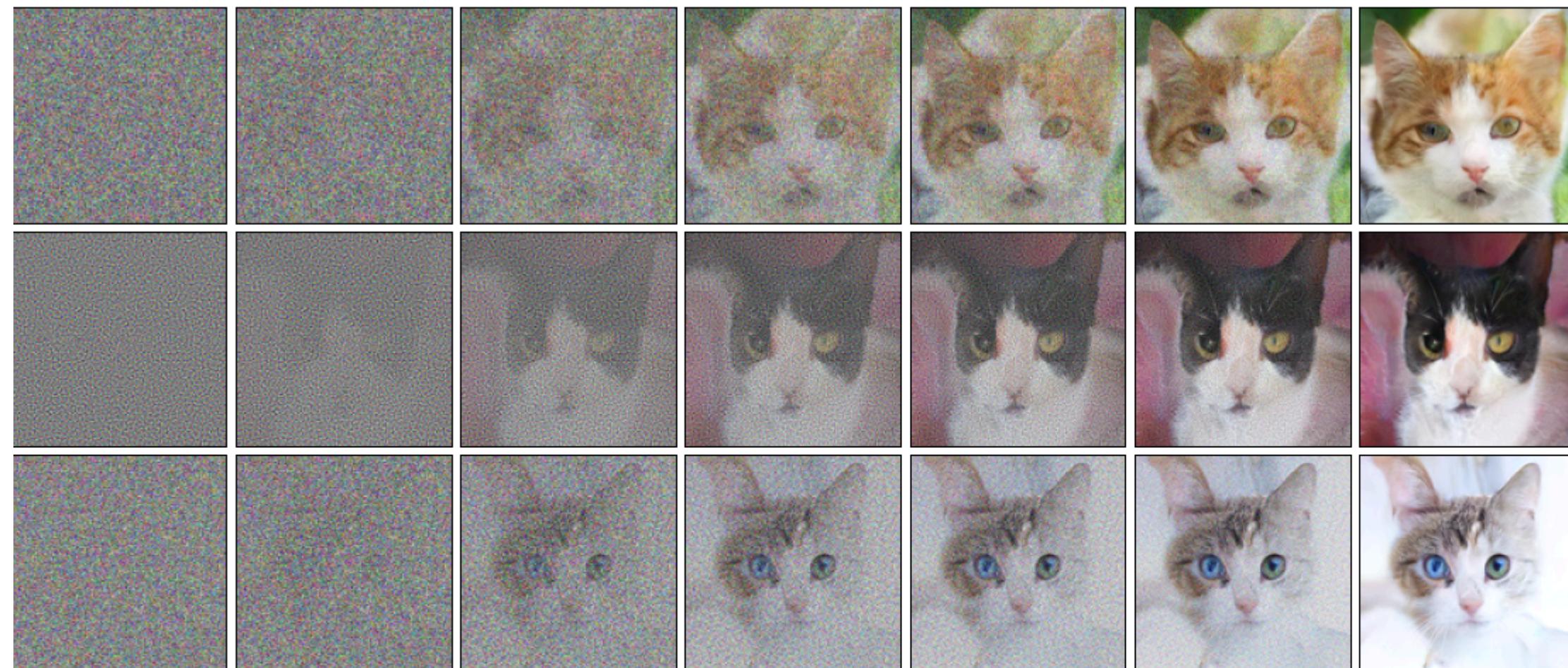


Figura 20. Esquema del proceso de inferencia en un modelo DDPM.

Fuente: Lmpo. (2023, junio 14). Mastering denoising diffusion probabilistic models. Medium. <https://medium.com/@lmpo/mastering-denoising-diffusion-probabilistic-models-8654cb6f6eff>



EXPERIMENTACIÓN Y RESULTADOS



1. Configuración del entorno

- El entrenamiento del modelo se realizó en un entorno de cómputo acelerado con GPU, utilizando una NVIDIA RTX A4500 con 20 GB de memoria.
- Durante el entrenamiento, la GPU presentó una utilización cercana al 99%, reflejando la intensidad computacional del proceso, y un consumo de memoria aproximado de 19 GB de los 20 GB disponibles,

5.1 Parámetros Utilizados de Entrenamiento

El entrenamiento se realizó utilizando las siguientes configuraciones y parámetros:

- **Optimizador:** Adam.
- **Tasa de aprendizaje:** 1e-4
- **Scheduler:** scheduler lineal de betas
- **β inicial:** 10^{-4}
- **β final:** 0.02
- **Número de canales de entrada:** Todas las imágenes se procesaron como 3 canales (RGB),

5.1 Parámetros Utilizados de Arquitectura

El arquitectura del modelo se realizó utilizando las siguientes configuraciones y parámetros:

- **Dimensión del time embedding:** 128
- **Número de cabezas de atención:** 4 cabezas en los bloques dedicados a capturar dependencias globales.
- **Canales del Down Block:** [32, 64, 128, 256], definen la expansión progresiva de canales a medida que la U-Net reduce la resolución.
- **Canales de Mid Block:** [256, 256, 128], mantienen un cuello de botella consistente para capturar representaciones profundas.
- **Canales del Up Block:** [256, 128, 64, 16], controlan la etapa de reconstrucción, reduciendo canales conforme se recupera la resolución inicial.

2. Experimento A

Este experimento corresponde a la primera versión del modelo entrenado sobre el dataset PicoBanana utilizando imágenes redimensionadas a 64×64 px, resolución inspirada en configuraciones clásicas como CIFAR y en el tamaño reducido del dataset original. Esta versión sirvió como referencia base para evaluar el comportamiento del DDPM con menor consumo de memoria y mayor velocidad de entrenamiento.

Configuración del Dataset

- **Resolución:** 64×64 píxeles
- **Número total de muestras:** 21, 896
- **Batch size:** 4
- **Número de workers:** 24
- **Número de épocas:** 100
- **Early stopping:** paciencia de 5 épocas
- **Número de timesteps:** 1000

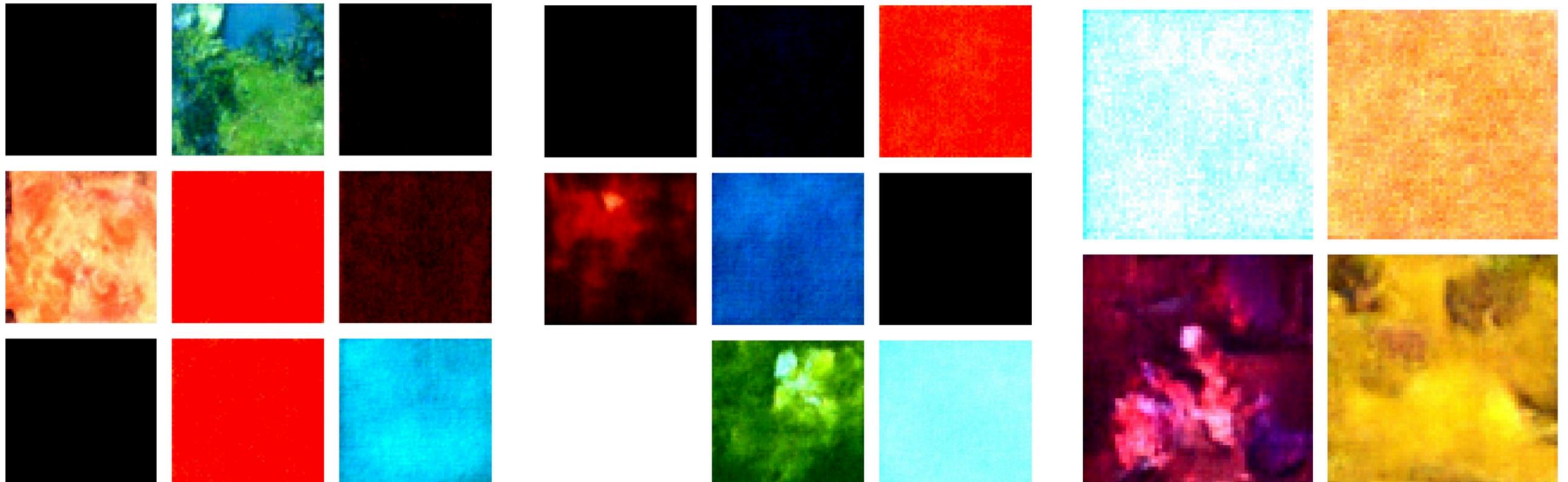


Figura 21. Resultados de generación sintética con DDPM entrenado en PicoBanana (64×64 px).

2. Experimento B

Este experimento se llevó a cabo utilizando una versión redimensionada del dataset CIFAR, con el objetivo específico de verificar si el rendimiento observado en experimentos previos se debía a un error en el modelo o a las limitaciones impuestas por entrenar con imágenes muy pequeñas (64×64 px). Para ello, se incrementó la resolución a 84×84 px y se utilizó un conjunto de datos más grande y uniforme.

Configuración del Dataset

- **Resolución:** 84×84 píxeles
- **Número total de muestras:** 60, 000
- **Batch size:** 4
- **Número de workers:** 24
- **Número de épocas:** 200
- **Early stopping:** paciencia de 40 épocas
- **Número de timesteps:** 1000

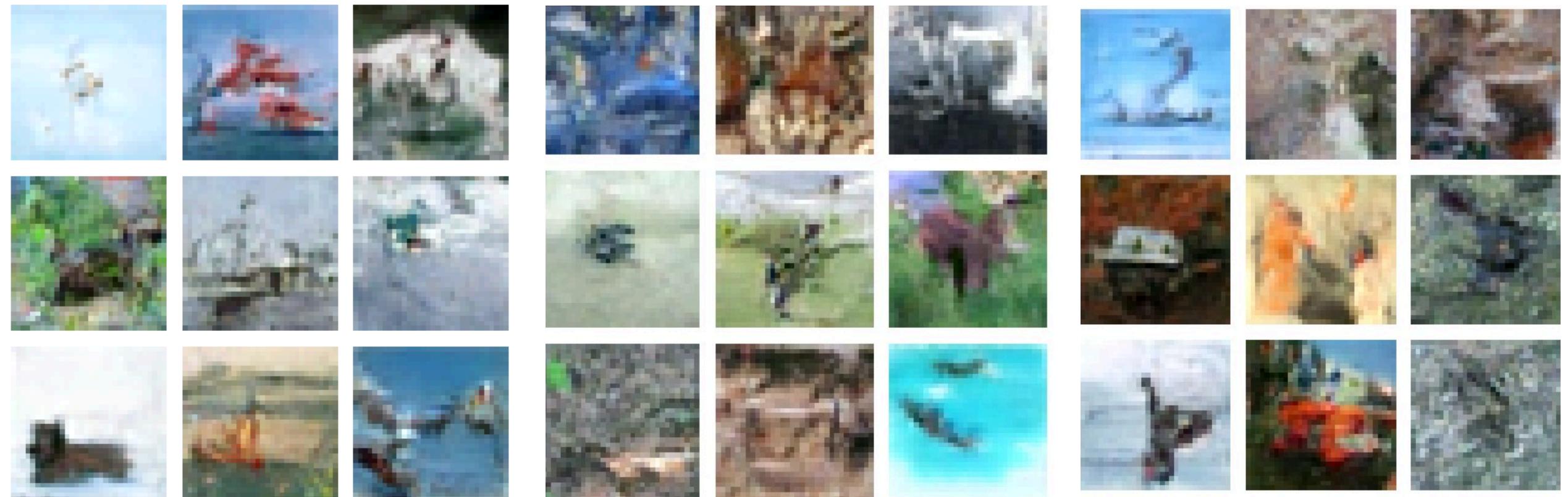


Figura 22. Imágenes sintéticas generadas tras entrenar el DDPM en CIFAR (84×84 px).

2. Experimento B

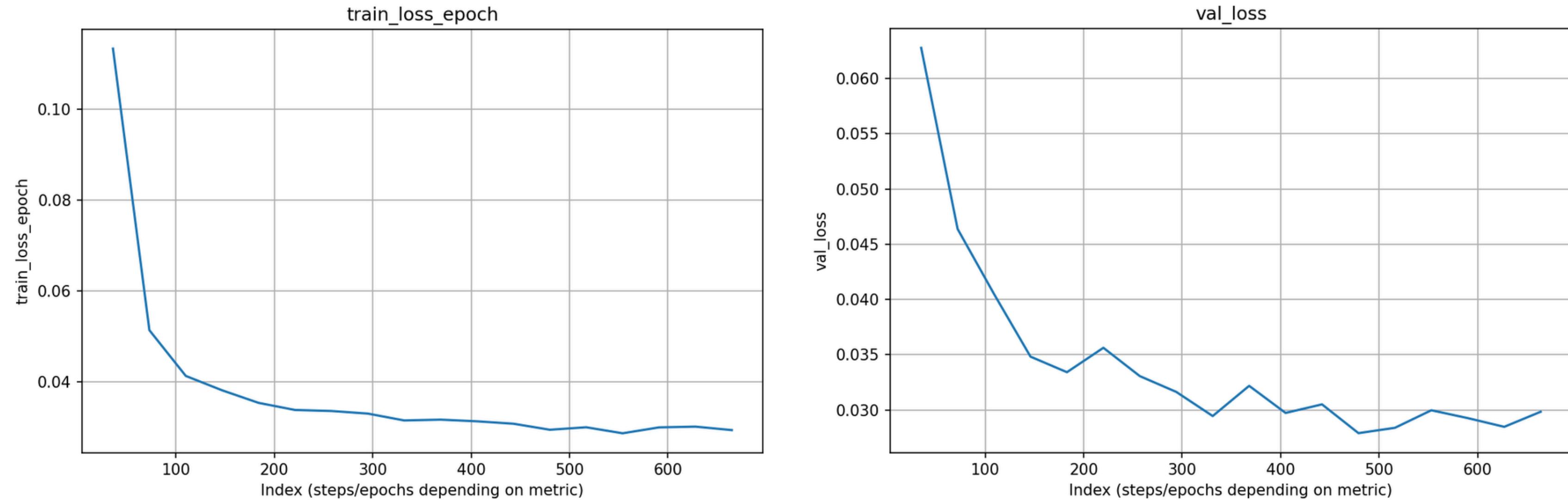


Figura 23. Evolución de la pérdida durante entrenamiento y validación para el modelo entrenado en CIFAR (84×84 px).

2. Experimento C

Luego de confirmar en el experimento con CIFAR que los problemas previos no provenían del modelo sino del tamaño reducido de las imágenes, se realizó un nuevo entrenamiento utilizando el dataset PicoBanana redimensionado a 80×80 px. Este experimento buscó evaluar si un aumento moderado en la resolución permitía capturar mejor la estructura visual del dataset y mejorar la calidad del aprendizaje del DDPM.

Configuración del Dataset

- **Resolución:** 80×80 píxeles
- **Número total de muestras:** 21, 896
- **Batch size:** 4
- **Número de workers:** 24
- **Número de épocas:** 200
- **Early stopping:** paciencia de 40 épocas
- **Número de timesteps:** 500

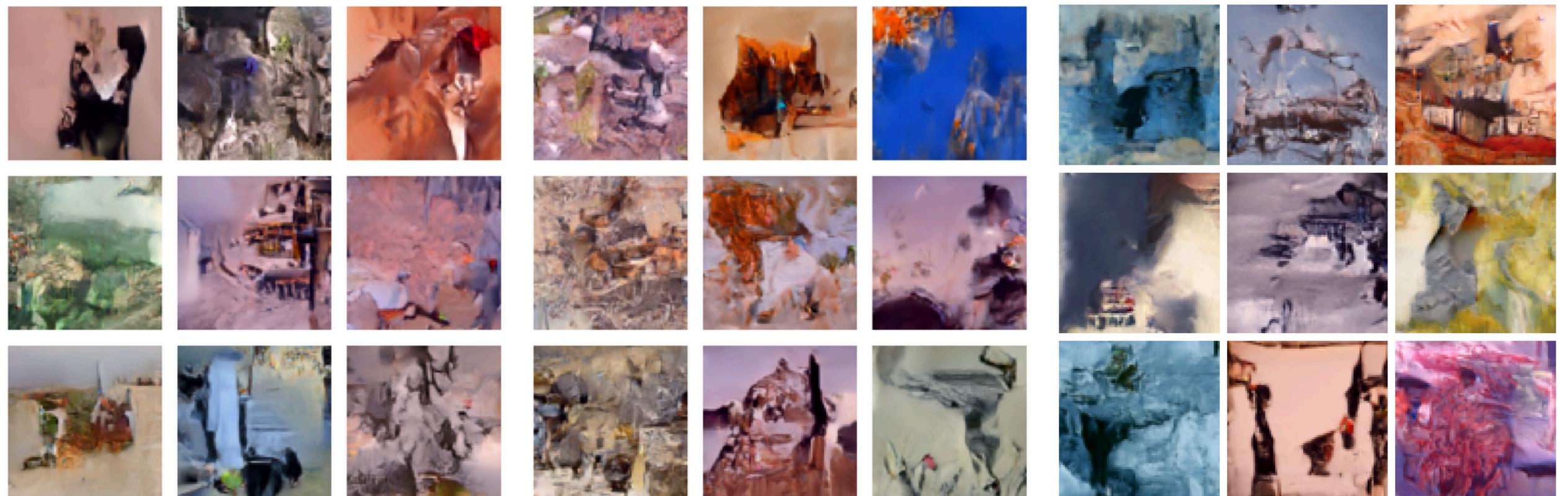


Figura 24. Resultados de generación del modelo entrenado con PicoBanana a 80×80 px.

2. Experimento C

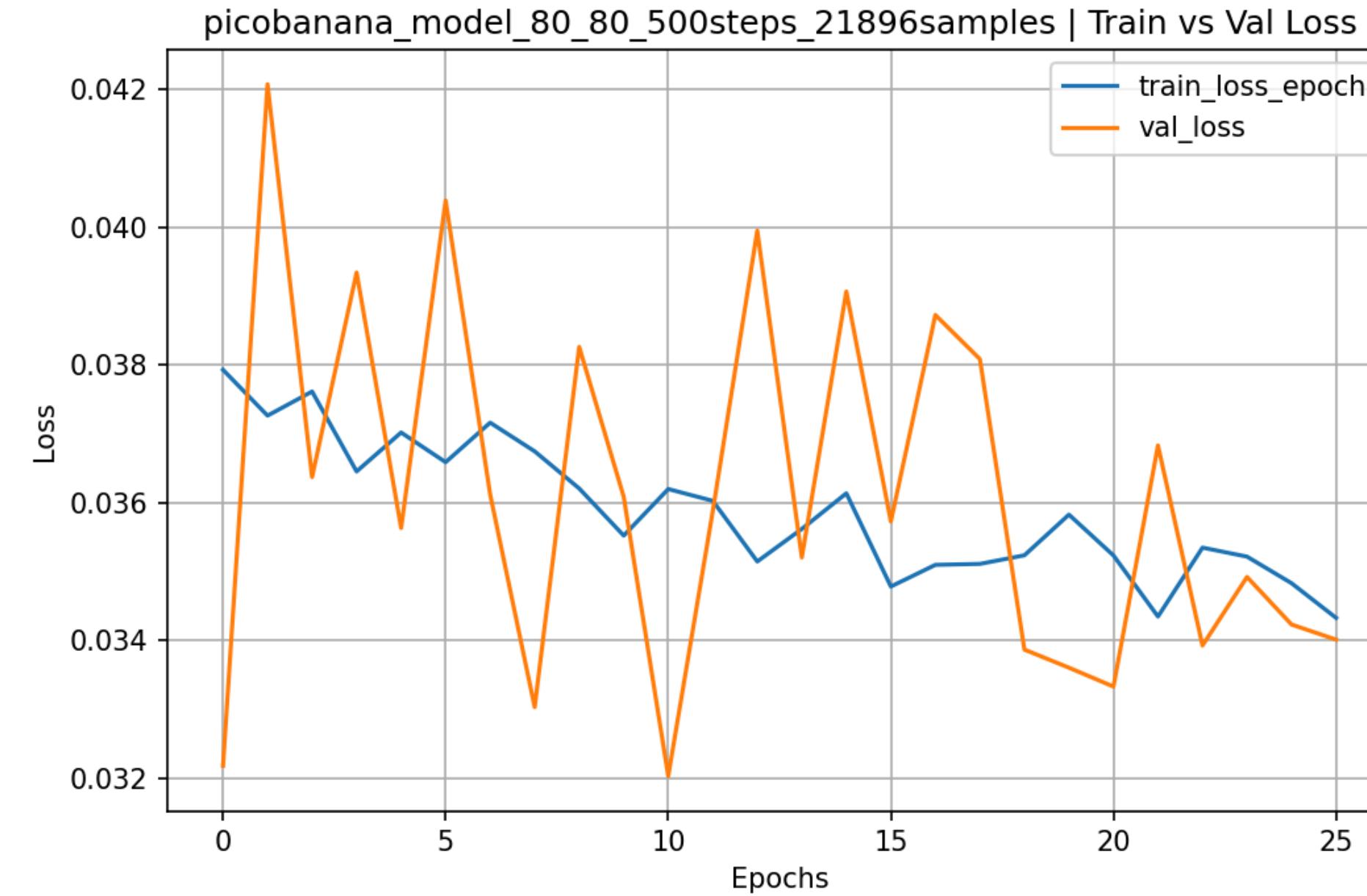


Figura 25. Evolución de la pérdida durante entrenamiento y validación para el modelo entrenado en PicoBanana (80×80 px).

2. Experimento D

Después de confirmar que incrementar la resolución mejoraba significativamente el desempeño del modelo en PicoBanana, se llevó a cabo un experimento final utilizando todas las imágenes disponibles del conjunto y redimensionadas a 84×84 px, el tamaño máximo manejable dentro de las limitaciones de memoria de la GPU. Este experimento representa la versión más completa del entrenamiento, tanto en cantidad de datos como en resolución.

Configuración del Dataset

- **Resolución:** 84×84 píxeles
- **Número total de muestras:** 257, 730
- **Batch size:** 4
- **Número de workers:** 24
- **Número de épocas:** 200
- **Early stopping:** paciencia de 40 épocas
- **Número de timesteps:** 500



Figura 26. Muestras sintéticas obtenidas del entrenamiento final con PicoBanana completo (84×84 px).

2. Experimento D

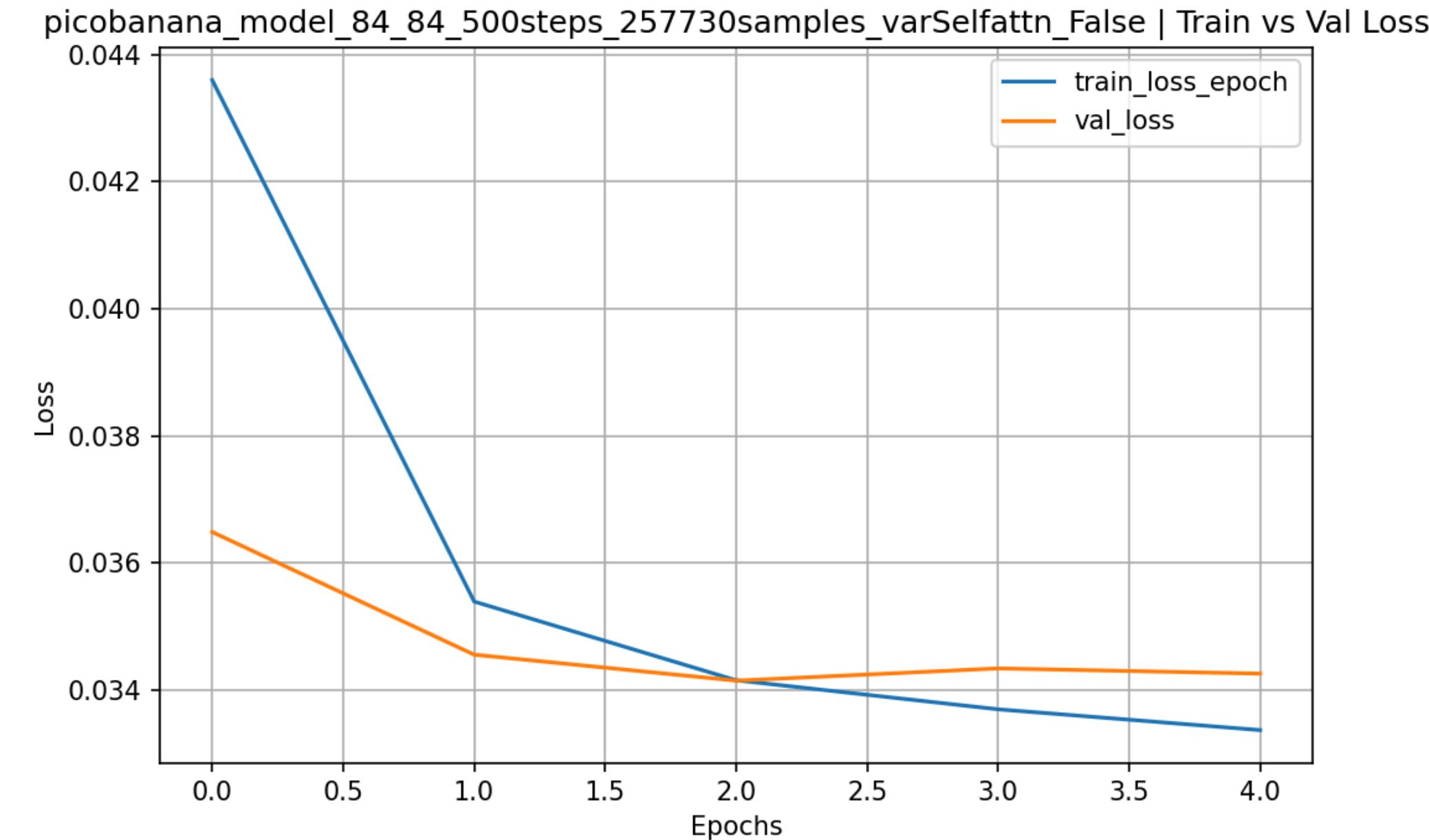


Figura 27. Evolución de la pérdida durante entrenamiento y validación para el modelo entrenado en PicoBanana (84×84 px).

2. Experimento E

Tras observar que PicoBanana presenta una enorme variabilidad en sus imágenes (distintos objetos, escenas, estilos y composiciones) se llevó a cabo un experimento adicional utilizando el dataset CelebA, caracterizado por contener rostros humanos en condiciones visuales relativamente homogéneas. El objetivo fue evaluar cómo se comporta el DDPM al entrenarse con un dataset más consistente y con menor diversidad semántica, permitiendo analizar si la arquitectura se beneficia de esta homogeneidad para mejorar estabilidad y calidad en la generación.

Configuración del Dataset

- **Resolución:** 76×76 píxeles
- **Número total de muestras:** 202, 599
- **Batch size:** 2
- **Número de workers:** 24
- **Número de épocas:** 200
- **Early stopping:** paciencia de 5 épocas
- **Número de timesteps:** 500

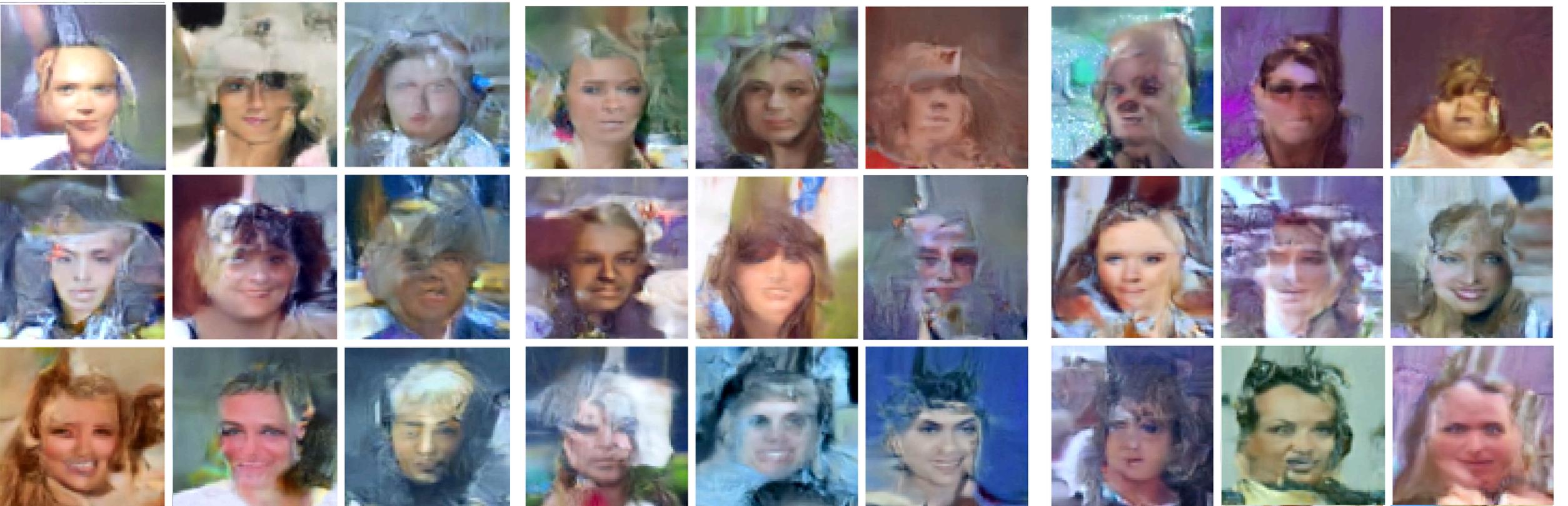


Figura 28. Imágenes generadas por el modelo entrenado en CelebA (76×76 px).

2. Experimento E



Tecnológico de Monterrey
Escuela de Ingeniería y Ciencias



2. Experimento E

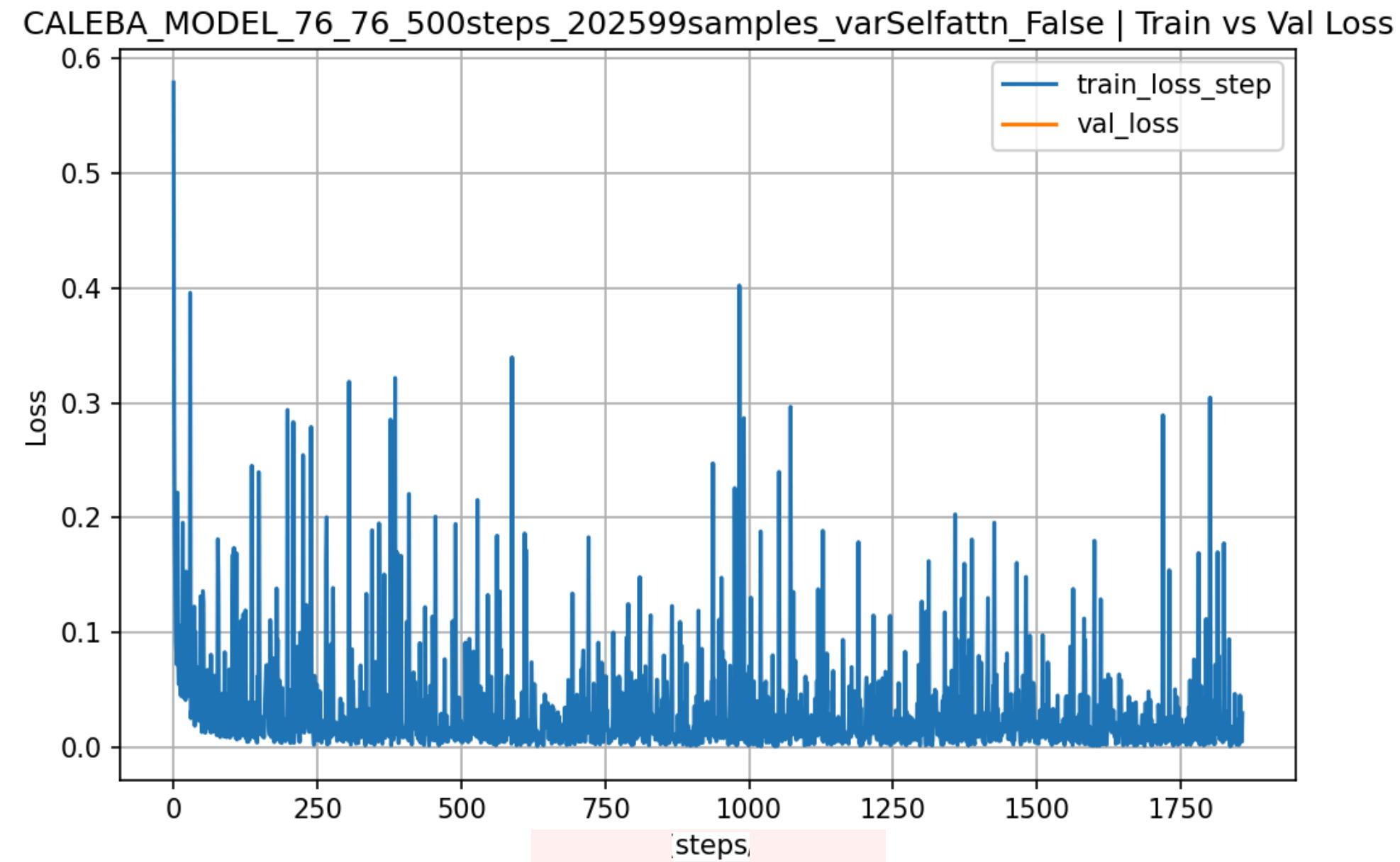


Figura 29. Evolución de la pérdida durante entrenamiento para el modelo entrenado en CelebA (76×76 px).

DISCUSIÓN

Los experimentos mostraron que el rendimiento del DDPM depende fuertemente de la resolución de las imágenes y del tamaño del dataset. Las versiones entrenadas con resoluciones mayores (80×80 y 84×84 px) y con más datos mostraron mejoras claras frente a los modelos iniciales de 64×64 px.

Sin embargo, las limitaciones de memoria de la GPU impidieron incrementar la capacidad del modelo: no fue posible aumentar canales, profundidad, ni el número de cabezas de atención; además, se tuvo que reducir los timesteps de 1000 a 500 y trabajar con batch sizes muy pequeños (2–4). Esto provocó que, en datasets grandes, el número efectivo de épocas fuese bajo (2–3).

A pesar de estas restricciones, el modelo respondió mejor cuando las imágenes tenían mayor resolución y cuando el dataset era grande o más homogéneo (como CelebA), lo cual indica que la arquitectura es estable, pero necesita más capacidad y memoria para aprovechar completamente la difusión.

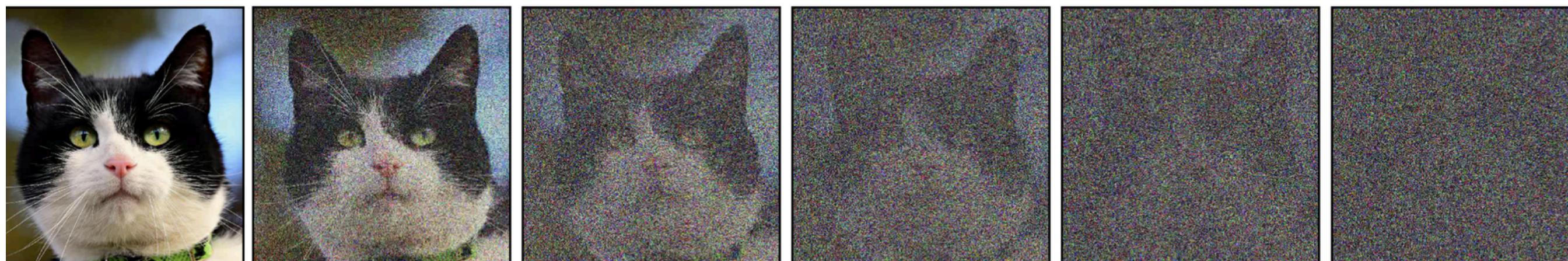


Figura 29. Proceso de forward en un modelo de difusión aplicado a una imagen de un gato

Fuente: Vandersanden, J., Holl, S., Huang, X., & Singh, G. (2024). Edge-preserving noise for diffusion models. arXiv (Cornell University). <https://doi.org/10.48550/arxiv.2410.01540>

RESULTADOS Y TRABAJO FUTURO

Resultados

- El desempeño del DDPM depende fuertemente de la resolución de las imágenes y de la cantidad de datos: al aumentar ambos, la calidad de las muestras sintéticas mejora de manera notable.
- Las limitaciones de memoria influyeron directamente en los resultados, restringiendo el número de timesteps, el batch size, la cantidad de atención, y la capacidad total de la U-Net.
- Experimentos con datasets más homogéneos (como CelebA) mostraron que la arquitectura es estable y capaz de aprender correctamente cuando las condiciones visuales son consistentes.
- En general, el modelo demostró buen comportamiento, pero su rendimiento sugiere que requiere más capacidad computacional para desplegar su máximo potencial.

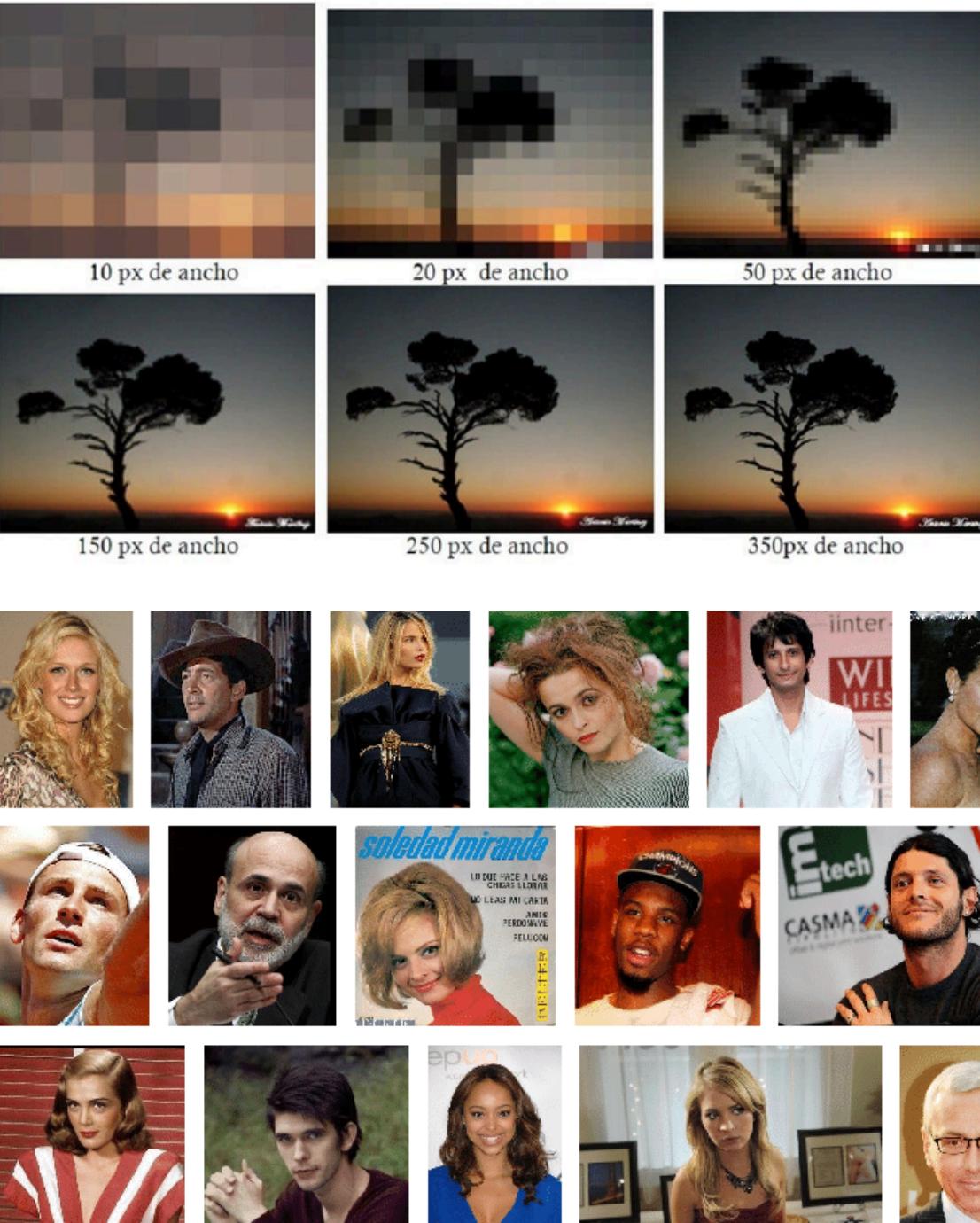


Figura 30. Imagen observada desde distintas resoluciones, ilustrando cómo la calidad visual varía según el nivel de detalle.

Fuente: Tafur del Aguila, A. L. (2020, septiembre 7). Resolución y formatos. Medium. <https://medium.com/@a20193010/resoluci%C3%ADn-y-formatos-289a8cceb637>

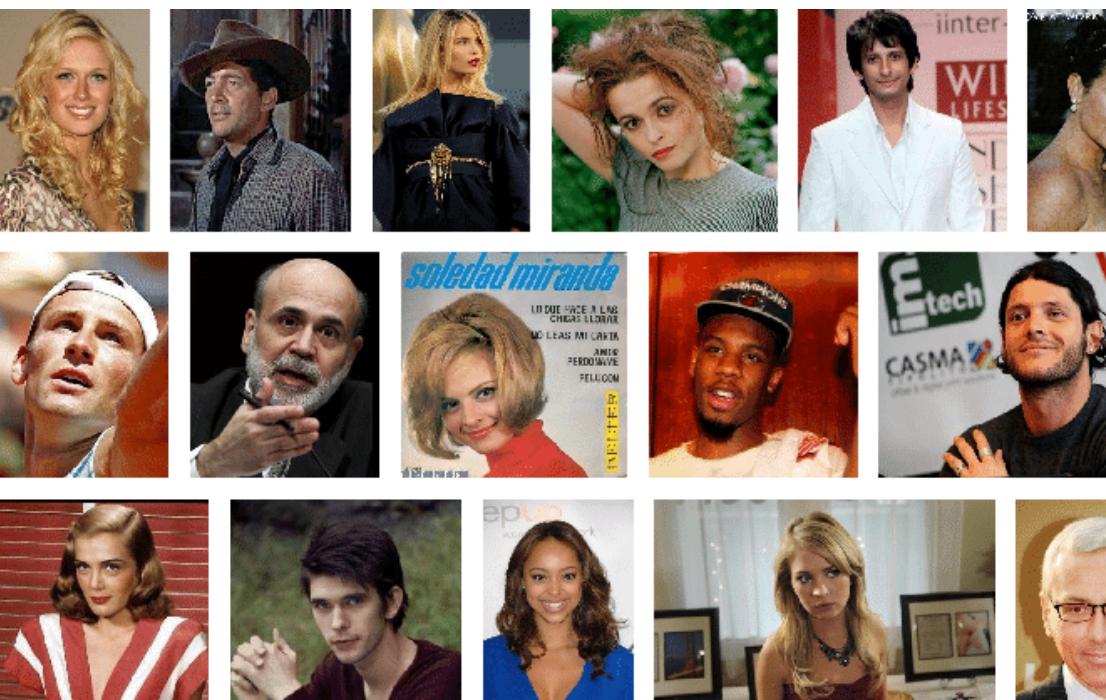


Figura 31. Ejemplos representativos del dataset CelebA utilizados durante el entrenamiento.

Fuente: Correia, J., Martins, T., & Machado, P. (2019). Evolutionary data augmentation in deep face detection. Proceedings Of The Genetic And Evolutionary Computation Conference Companion, 163-164. <https://doi.org/10.1145/3319619.3322053>

RESULTADOS Y TRABAJO FUTURO

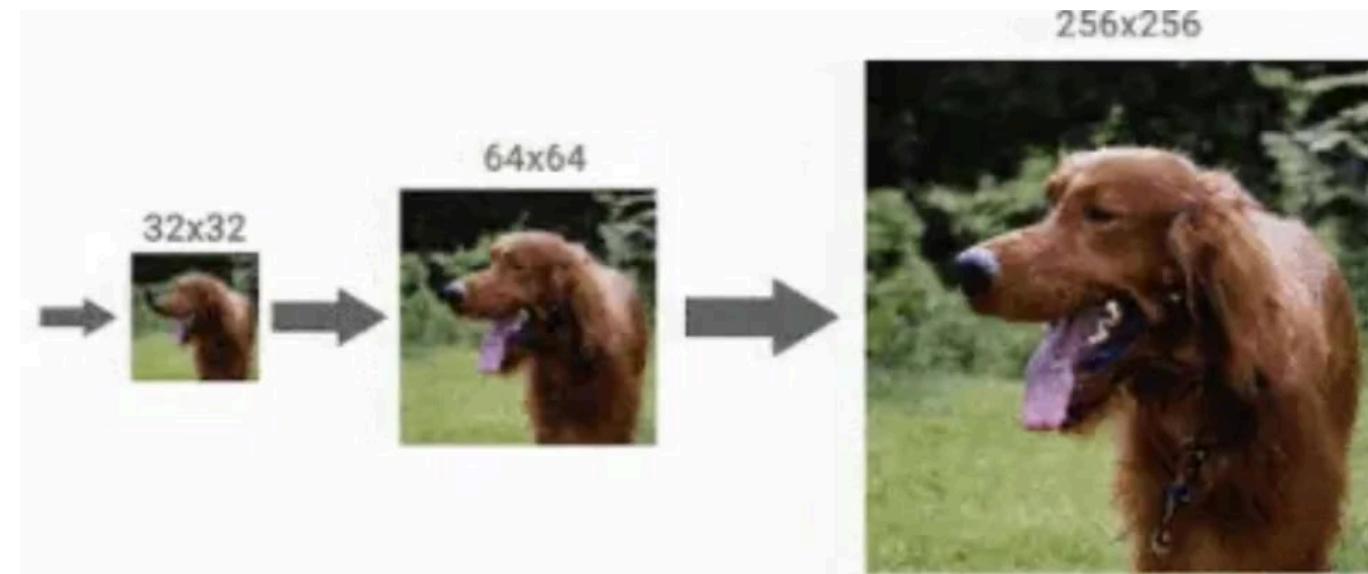


Figura 32. Modelo de difusión en cascada compuesto por un modelo base y dos modelos de superresolución

Fuente: Ho, J., Salimans, T., Gritsenko, A., Chan, W., Norouzi, M., & Fleet, D. J. (2021). Cascaded diffusion models for high fidelity image generation. arXiv. https://cascaded-diffusion.github.io/assets/cascaded_diffusion.pdf

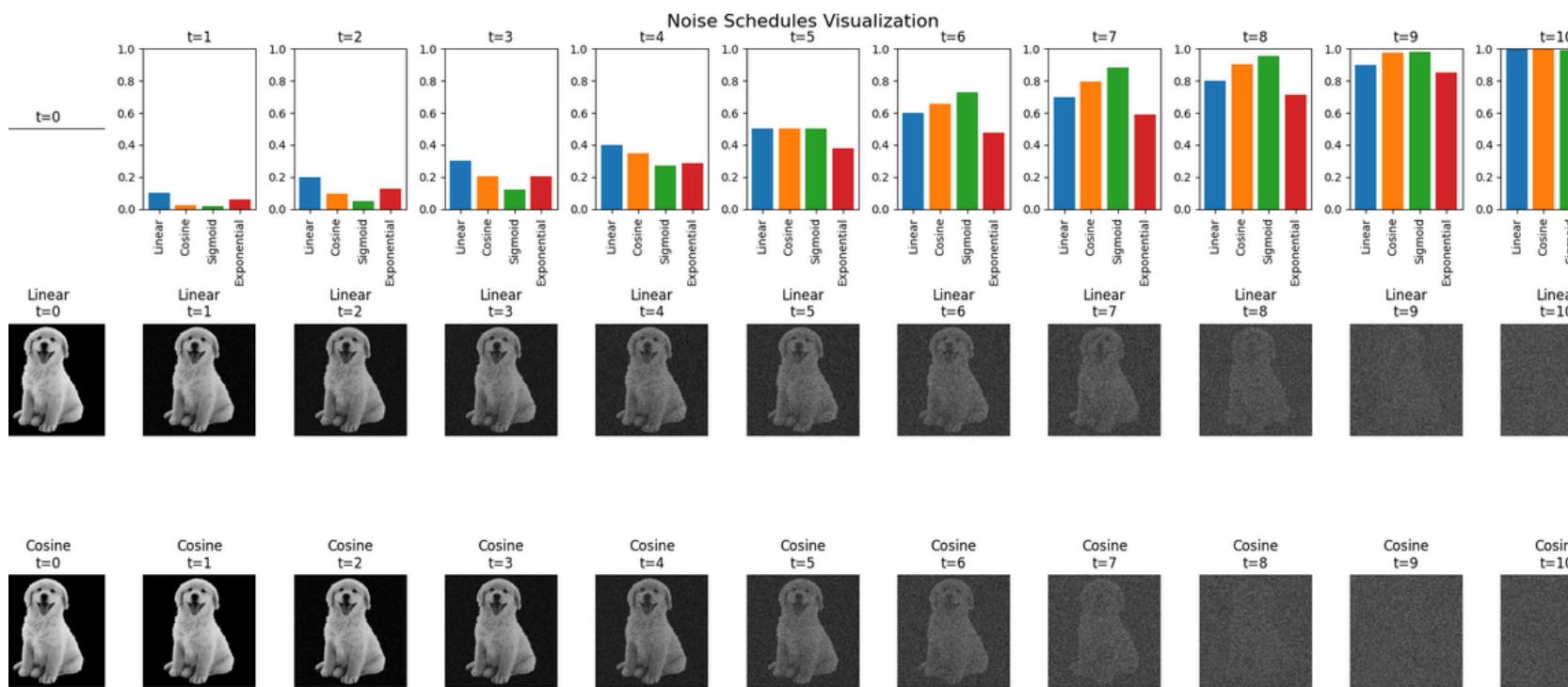


Figura 33. Representación del sonido bajo diferentes schedules de difusión

Fuente: Annemsony. (2022, marzo 28). Introduction to diffusion models. Medium. <https://medium.com/@annemsony/introduction-to-diffusion-models-56beecd6e5c1d>

Trabajo Futuro

- Entrenar modelos de difusión en cascada para generar imágenes de baja resolución primero y refinarlas después, permitiendo mayores resoluciones sin exceder la memoria disponible.
- Evaluar cuantitativamente la calidad de las imágenes mediante métricas estándar como Fréchet Inception Distance (FID)
- Explorar el uso de schedulers más avanzados para el proceso de difusión, como variantes cosine, quadratic o sigmoid, que podrían mejorar la estabilidad del entrenamiento y la calidad de las muestras generadas.
- Ajustar la arquitectura: aumentar canales, capas y número de cabezas de atención cuando se cuente con mayor capacidad de memoria.

DDPM IMAGE GENERATOR

La solución integra un sistema full-stack diseñado para realizar inferencia con modelos DDPM previamente entrenados. El backend trabaja exclusivamente con modelos serializados (.pt/.pth) y ejecuta el proceso de denoising para generar imágenes bajo demanda.

Technology Stack

- **Backend (Flask + PyTorch)**
 - Flask para la REST API.
 - PyTorch para cargar y ejecutar modelos ya entrenados.
 - Model Manager para validación, carga dinámica y caching de modelos.
 - Pillow y NumPy para el procesamiento y conversión de imágenes.
- **Frontend (React + Vite)**
 - Interfaz moderna para solicitar imágenes y visualizar resultados.
 - Comunicación con el backend vía JSON.
 - Tailwind CSS para estilos.

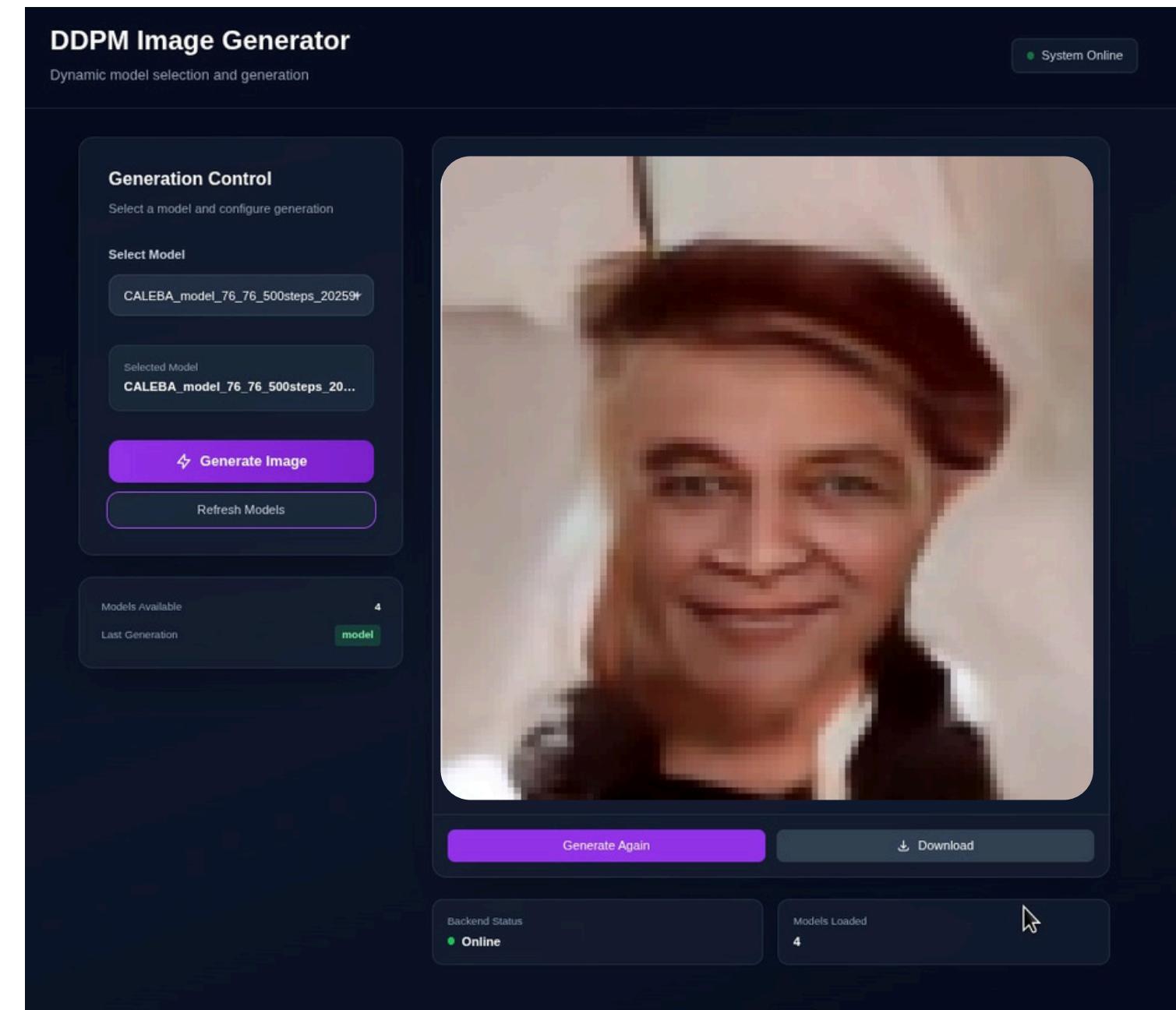


Figura 34. Sistema de generador de imágenes por DDPM

DDPM IMAGE GENERATOR

Antes de desplegar nuevas versiones del DDPM, se evaluaría offline usando:

- Baselines internos: comparar contra versiones anteriores del modelo (ej. $64 \times 64 \rightarrow 80 \times 80 \rightarrow 84 \times 84$).
- Métricas objetivas como FID, útiles para modelos generativos donde la calidad visual no es trivial de medir.
- Pruebas de robustez para asegurarnos de que el modelo no falla con imágenes atípicas del dataset.

Pipeline de Inferencia según la Fase

- Batch inference: útil cuando queremos evaluar toda una versión del modelo DDPM generando miles de muestras para cálculo de FID.
- Online inference: el modo de uso actual en la app, donde el usuario solicita una imagen y el backend Flask ejecuta el denoising paso a paso en tiempo real.

Riesgos Operativos y Cómo Afectan a Nuestro Modelo

- Data drift: el modelo fue entrenado con PicoBanana/CelebA; si las imágenes futuras tienen otra distribución, la calidad de generación puede degradarse.
- Edge cases: timesteps muy bajos, resoluciones no soportadas o imágenes con canales distintos podrían romper la inferencia.
- Degenerate feedback loops: si se reentrenara el modelo con imágenes generadas por sí mismo (sin control), la calidad se deterioraría rápidamente.

Monitoreo en Producción para Nuestro DDPM

- Métricas del modelo: latencia del denoising, tiempo por iteración, tasa de fallos al cargar modelos.
- Monitoreo de features: detectar si los tensores de entrada/salida salen de rango ($[0,1]$, $[-1,1]$) o si hay anomalías en predicciones de ruido.
- Alertas: activadas si FID sube en validaciones automáticas o si el modelo falla al generar imágenes

Estrategias de Despliegue Seguro

- Shadow deployment: enviar peticiones tanto al modelo viejo como al nuevo sin mostrar el resultado al usuario. Ideal para comparar calidad generativa y latencia.
- A/B testing: mostrar un porcentaje pequeño de imágenes generadas con un nuevo modelo y medir interacción/tiempo de generación.
- Canary release: habilitar gradualmente un nuevo modelo si pasa métricas de calidad (por ejemplo, $\text{FID} < \text{umbral}$).



REFERENCIAS

1. Ho, J., Saharia, C., Chan, W., Fleet, D. J., Norouzi, M., & Salimans, T. (2021). Cascaded Diffusion Models for High Fidelity Image Generation. arXiv (Cornell University).
<https://doi.org/10.48550/arxiv.2106.15282>
2. Gascón, M., & Gascón, M. (2023, 31 marzo). Se acabó Midjourney gratis: la plataforma de inteligencia artificial cobrará por generar imágenes. 20minutos.
<https://www.20minutos.es/tecnologia/aplicaciones/se-acabo-midjourney-gratis-la-plataforma-de-inteligencia-artificial-cobrara-por-generar-imagenes-5115095/>
3. Singh, V., & Singh, V. (2024). InDepth Guide to Denoising Diffusion Probabilistic Models (DDPM). LearnOpenCV. <https://learnopencv.com/denoising-diffusion-probabilistic-model>
4. Calibraint. (2024, 13 agosto). A Simple Guide to Diffusion Models. Calibraint. <https://www.calibraint.com/blog/beginners-guide-to-diffusion-models>
5. Nichol, A., & Dhariwal, P. (2021). Improved denoising diffusion probabilistic models. arXiv (Cornell University). <https://doi.org/10.48550/arxiv.2102.09672>
6. Yoon, S., Lee, Y., Jung, E., & Ahn, T. I. (2024). Agricultural Applicability of AI based Image Generation. Journal Of Bio-Environment Control, 33(2), 120-128.
<https://doi.org/10.12791/ksbec.2024.33.2.120>
7. Higham, C. F., Higham, D. J., & Grindrod, P. (2023). Diffusion Models for Generative Artificial Intelligence: An Introduction for Applied Mathematicians. arXiv (Cornell University).
<https://doi.org/10.48550/arxiv.2312.14977>
8. Elgazar, E. (2023). Diffusion model U-Net [Notebook]. Kaggle. <https://www.kaggle.com/code/ebrahimelgazar/diffusion-model-u-net/notebook>
9. Sayed, E. (2024, diciembre 6). DDPM PyTorch implementation from scratch. Medium. <https://medium.com/@sayedebad.777/ddpm-pytorch-implementation-from-scratch-36b647f5dd82>
10. Song, L., Yang, Y., Lu, J., Hu, W., & Gan, Z. (2025). Pico-Banana-400K: A Large-Scale Dataset for Text-Guided Image Editing. arXiv (Cornell University).
<https://doi.org/10.48550/arxiv.2510.19808>
11. Lmpo. (2023, junio 14). Mastering denoising diffusion probabilistic models. Medium. <https://medium.com/@lmpo/mastering-denoising-diffusion-probabilistic-models-8654cb6f6eff>
12. Vandersanden, J., Holl, S., Huang, X., & Singh, G. (2024). Edge-preserving noise for diffusion models. arXiv (Cornell University). <https://doi.org/10.48550/arxiv.2410.01540>
13. Tafur del Aguila, A. L. (2020, septiembre 7). Resolución y formatos. Medium. <https://medium.com/@a20193010/resoluci%C3%B3n-y-formatos-289a8cceb637>
14. Correia, J., Martins, T., & Machado, P. (2019). Evolutionary data augmentation in deep face detection. Proceedings Of The Genetic And Evolutionary Computation Conference Companion, 163-164. <https://doi.org/10.1145/3319619.3322053>
15. Ho, J., Salimans, T., Gritsenko, A., Chan, W., Norouzi, M., & Fleet, D. J. (2021). Cascaded diffusion models for high fidelity image generation. arXiv. https://cascaded-diffusion.github.io/assets/cascaded_diffusion.pdf