UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ciencia e Ingeniería de Datos

SISTEMA DE IA GENERATIVA PARA IMÁGENES

Trabajo final de Aprendizaje Automático III

Autor: Adam Maltoni
Autor: Ibón de Mingo Arroyo
Tutor: Alberto Suárez González

mayo 2025

Agradecimientos

A nuestras familias, nuestros amigos y, por supuesto, a nuestros profesores, en especial a Alberto Suárez, por su guía y sabiduría compartida a lo largo de este proyecto.

Resumen

El objetivo de este proyecto es construir y diseñar un sistema de generación de imágenes basado en modelos de difusión para crear imágenes en blanco y negro o a color. La característica principal del sistema es la flexibilidad, ya que permite que el usuario experimente con diferentes partes del proceso de generación de imágenes de una forma sencilla e intuitiva.

En módulo comienza con la implementación de tres tipos distintos de procesos de difusión para la inyección de ruido: *Variance Exploding* (VE) basado en movimiento browniano, *Variance Preserving* (VP) que utiliza *Ornstein-Uhlenbeck*, y *Sub-Variance Preserving* (Sub-VP), una variante de VP.

Para la generación de imágenes, implementamos varios algoritmos de muestreo que transforman el ruido en imágenes sintéticas. Entre ellos se encuentran el integrador de Euler-Maruyama (que es el más sencillo a nivel teórico), el método *Predictor-Corrector*, *Probability Flow ODE* (que tiene enfoque determinista), y *Exponential Integrator*. Estas diferentes técnicas de muestreo permiten al usuario experimentar con la calidad, velocidad y diversidad de las imágenes que se generan.
También incorporamos dos tipos de esquemas para la intensidad del ruido: el esquema lineal, más directo y predecible, y el esquema coseno, que es más suave y natural. Conocer el funcionamiento de ambos esquemas ayuda a mejorar la calidad final de las imágenes.

El sistema también incluye un módulo de métricas cuantitativas que sirven para evaluar de manera más objetiva la calidad de las imágenes generadas. Las tres métricas implementadas son las siguientes *Bits Per Dimension (BPD)* que evalúa la comprensión del modelo sobre los datos, *Fréchet Inception Distance (FID)* que compara la similitud estadística con imágenes reales, e *Inception Score (IS)* mide la calidad y diversidad de las imágenes generadas

Finalmente, nuestro sistema no se limita a tan solo generar imágenes. Implementamos mecanismos de control como el muestreo condicionado por clases (donde se puede especificar que clase se quiere generar, por ejemplo un 3 del *dataset MNIST*) y la imputación, que permite editar partes específicas de una imagen manteniendo el resto intacto.

Este trabajo representa una plataforma experimental completa que ayuda a iniciarse con modelos de difusión usando una metodología sencilla y práctica.

Palabras clave

IA generativa, modelos de difusión, condicionamiento, aprendizaje automático, *sampling*, inyección de ruido, métricas, rendimiento

Índice

1

Introducción

El propósito de este proyecto ha sido crear un sistema generativo de inteligencia artificial generativa, basado en modelos de difusión, capaz de generar imágenes a color de forma controlada y flexible. Nuestra idea principal era no solo construir un modelo funcional, sino también investigar las distintas variantes del módulo y observar cómo las diferentes configuraciones afectan a las imágenes generadas. Para la realización del módulo, se utilizan conceptos clave del aprendizaje automático, como la modelación probabilística, los procesos estocásticos y el aprendizaje de funciones *score*. Estos elementos teóricos y computacionales son esenciales para entrenar y guiar el modelo en el proceso de generación de imágenes desde ruido puro hacia imágenes coherentes.

El módulo realizado contiene cinco componentes principales para su funcionalidad. El primero de ellos incluye los tres procesos de difusión que se utilizan para ir introduciendo ruido poco a poco en los datos (*Variance exploding*, *Variance preserving* y Sub-Variance preserving). Después se encuentran los algoritmos de muestreo empleados para revertir ese proceso y sintetizar imágenes que son *Euler Maruyama*, *Predictor corrector*, *Exponential integrator* y *Probability flow ODE*. El módulo también contiene esquemas de programación del ruido (Lineal y Coseno) las métricas utilizadas para evaluar la calidad de las imágenes generadas por el modelo (*BPD*, *FID*, *IS*) y, finalmente, mecanismos que permiten controlar la generación mediante condicionamiento por clase o completado de imágenes.

2

Estado del arte

La generación de imágenes sintéticas mediante inteligencia artificial es una de las áreas que más se han desarrollado dentro del aprendizaje automático en los últimos años. En este proyecto se diseña un sistema generativo capaz de producir imágenes de forma configurable y controlada. El sistema desarrollado se apoya en un modelo de score basado en una arquitectura tipo U-Net, que está entrenado para aprender el comportamiento inverso del proceso de degradación de una imagen en ruido. A través de este modelo, y mediante el uso de distintos *samplers* e integradores, se consigue generar imágenes realistas partiendo de ruido puro. Además, se incluye un método para condicionar la generación por clase y otro para la imputación de datos faltantes en imágenes.

Los modelos de difusión han ganado mucha relevancia en los últimos años dentro del campo de la inteligencia artificial generativa. Su funcionamiento se basa en un proceso en dos fases que transforma una imagen en ruido y luego intenta reconstruirla de nuevo. Esta idea está siendo muy eficaz para generar imágenes de alta calidad, y es la base de algunos de los sistemas de generación de imágenes actuales.

El proceso de difusión comienza con una imagen de nuestro conjunto de entrenamiento a la que se le va aplicando ruido progresivamente hasta convertirla en ruido puro. A este proceso le llamamos proceso directo (o forward en inglés). Este paso no requiere aprendizaje, ya que el comportamiento está determinado por una ecuación diferencial estocástica que determina todos los pasos del proceso de inyección de ruido en la imagen.

El aprendizaje ocurre durante el entrenamiento del modelo, que se centra en el proceso inverso. En este paso, la red neuronal se entrena para estimar el score, es decir, el gradiente del logaritmo de la probabilidad de los datos en cada instante de ruido. Para ello, observa muchas muestras intermedias generadas por el proceso directo y aprende a predecir en qué dirección debería moverse el ruido para volver a parecerse a una imagen real. El modelo que hemos utilizado se basa en esta idea y emplea una red neuronal del tipo U-Net para estimar el score.

Existen distintas formas de definir la evolución del ruido en el proceso directo, lo cual afecta no solo al entrenamiento del modelo, sino también al proceso inverso y a la calidad de las imágenes que se generan. En este proyecto se han trabajado tres variantes principales: VE (Variance Exploding), VP (Variance Preserving), y Sub-VP. Cada variante tiene una ecuación diferencial estocástica distinta para llevar a cabo el proceso de difusión, lo cual implica que tienen distintos comportamientos prácticos.

En el caso de VE (Variance Exploding), la varianza del ruido crece con el tiempo, haciendo que las imágenes se vayan degradando más rápido hacia ruido puro. Este proceso simula un movimiento browniano y no tiene un término de *drift*, solo un crecimiento en la dispersión. Por otro lado, en VP (Variance Preserving), el ruido se añade de forma que la varianza total de la muestra se mantiene constante a lo largo del proceso. A medida que pasa el tiempo, la imagen se vuelve más borrosa, pero sin que el ruido se dispare. En este proyecto, el proceso VP se basa en una ecuación del tipo Ornstein-Uhlenbeck, en la que tanto el término de drift como el término de difusión dependen del tiempo. Finalmente, Sub-VP (Sub-Variance Preserving) es una variante del VP que modifica los coeficientes de la ecuación del término de drift (es más complejo, ya que hay que integrar el esquema de ruido) consiguiendo una transición más progresiva entre imagen y ruido.

Por otro lado, a la hora de utilizar VP y Sub-VP hay que tener en cuenta que dependen de una función que define la cantidad de ruido se va añadiendo en cada paso del proceso *forward*. Esta función se conoce como *noise schedule*. En este trabajo se han implementado dos opciones: un *schedule* lineal, que añade ruido de forma constante, y un *schedule* basado en la función coseno, que ajusta la intensidad del ruido de manera más suave y progresiva, lo cual produce mejores resultados en ciertos escenarios.

Una vez se tiene entrenada la red neuronal con el score, se lleva a cabo el proceso inverso para reconstruir imágenes desde ruido aleatorio. Para ello, es necesario aplicar métodos de integración numérica, conocidos como *samplers*, que permiten simular el camino desde el ruido puro hasta la reconstrucción de una imagen parecida a las muestras utilizadas durante el entrenamiento. Como el proceso inverso de difusión no puede resolverse de forma exacta en la mayoría de los casos ya que depende de ecuaciones diferenciales estocásticas complejas, se recurre a estos esquemas para construir trayectorias aproximadas que el modelo puede seguir durante la generación.

En este sistema se han implementado las siguientes opciones: el integrador de *Euler-Maruyama*, el método *predictor-corrector*, el *probability flow ODE* y un *integrador exponencial*. Cada uno de estos métodos obtiene resultados distintos dependiendo del contexto y del proceso de difusión utilizado.

Euler-Maruyama es uno de los métodos más básicos para resolver ecuaciones diferenciales estocásticas como las que aparecen en el proceso inverso. Va simulando el proceso inverso poco a poco, combinando el término de drift y el término de difusión con la

dirección indicada por el score. Es un método sencillo y rápido, aunque no siempre se obtienen resultados buenos, especialmente si se usan pocos pasos. Con el método predictor-corrector se consigue mejorar esta aproximación ya que se combina una primera estimación (*predictor*) con una corrección posterior (*corrector*), lo cual obtiene imágenes de mejor calidad aunque un número alto de pasos del corrector puede hacer que sea computacionalmente costoso. Por otro lado, el *probability flow ODE* convierte la ecuación diferencial estocástica del proceso inverso en una ecuación diferencial ordinaria (ODE) que es determinista. Por lo tanto la parte aleatoria del proceso queda eliminada lo cual genera imágenes más parecidas entre sí. Finalmente, el integrador exponencial es un método que ayuda a reconstruir imágenes desde el ruido utilizando funciones exponenciales para ajustar mejor cada paso del proceso inverso. Además, permite elegir si se quiere añadir ruido o no: si el parámetro de ruido (sigma) es cero, el método se vuelve completamente determinista al igual que *probability flow ODE*.

Además de generar imágenes desde ruido puro, los modelos de difusión permiten incorporar mecanismos de control explícito sobre el proceso generativo. Uno de los más destacados es el condicionamiento por clase (*class conditioning*), que permite guiar la generación hacia una categoría específica, como un dígito o un tipo de objeto.

Durante el entrenamiento, el modelo aprende a predecir la puntuación (score) o el ruido añadido en cada paso de tiempo, dado no solo el estado actual de la imagen x_t y el instante t, sino también una condición externa, como una etiqueta de clase y. Esta información condicional se introduce a través de un *embedding* que se inyecta en la arquitectura (por ejemplo, en una red tipo U-Net) junto al embedding temporal. De esta manera, el modelo aprende una función score condicional $s(x_t, t \mid y)$ que apunta en la dirección del gradiente del logaritmo de la probabilidad condicional $\nabla_x \log p(x_t \mid y)$.

Durante la generación, se puede reforzar esta guía mediante una técnica conocida como *classifier-free guidance*, pudiendo entrenar el modelo sin la condición (y ausente) $s(x_t, t)$ (no condicionada) o con condición $s(x_t, t \mid y)$. Durante el muestreo, se calcula una combinación con un parámetro de escala para pesar el efecto de la condición, guiando la muestra hacia regiones de alta probabilidad para la clase y con más o menos fuerza.

Alternativamente, si se dispone de un modelo auxiliar que estima la probabilidad de clase $p(y \mid x_t)$, también puede utilizarse el gradiente de log-verosimilitud de la clase ($\nabla_x \log p(y \mid x_t)$) como una corrección al score no condicionado, lo que permite una guía directa basada en el gradiente del clasificador.

Otro mecanismo importante es la imputación, que permite completar regiones ausentes o dañadas de una imagen. Esto se logra aprovechando la naturaleza iterativa y reversible del proceso de difusión. En este caso, se parte de una imagen x_0 parcialmente observada y se aplica el proceso de difusión hacia delante ($x_0 \rightarrow x_t$) únicamente en las regiones dañadas, mientras se mantiene fijo el contenido observado. Luego, se ejecuta el proceso inverso ($x_t \rightarrow$

x_0) restringiendo las actualizaciones únicamente a las regiones faltantes.

Formalmente, si M es una máscara binaria que indica las regiones ausentes (M=1 donde hay que imputar), el paso de score se modifica en cada iteración para preservar las partes observadas, asegurando que el modelo respete el contexto observado y genere solo en las zonas libres, produciendo resultados coherentes con la información existente.

En ambos casos el principio subyacente es que el modelo aprende una distribución sobre el espacio de imágenes y se fuerza la “navegación” de forma controlada mediante condiciones externas o restricciones locales.

3

Desarrollo del software

3.1. Planificación del trabajo

El planteamiento del proyecto ha seguido una dinámica exploratoria y evolutiva, adaptándose a las necesidades técnicas y conceptuales a medida que se adquiría un conocimiento más profundo del dominio. Desde el inicio, el objetivo fue construir un sistema generativo de imágenes basado en procesos de difusión configurable, modular y comparable con las propuestas más recientes de la literatura científica. El desarrollo se enmarca en el contexto de un proyecto académico, pero con una clara intención de mantener estándares profesionales tanto en la organización como en la implementación del software.

La primera fase del trabajo consistió en entender con solidez los fundamentos teóricos de los procesos de difusión estocásticos y su aplicación a la generación de imágenes. Esta etapa implicó un análisis exhaustivo de artículos relevantes y un esfuerzo considerable por clarificar y unificar notación, técnicas y formulaciones. Aunque esta parte no está directamente reflejada en líneas de código, ha sido esencial para guiar con coherencia el diseño del sistema.

Una vez asentados los conceptos clave, se procedió a organizar el desarrollo en tareas claras y bien delimitadas. Para ello se optó por construir de forma estructurada el proceso completo mediante un diagrama de Gantt , lo cual permitió documentar el flujo real de trabajo, a pesar de que muchas decisiones técnicas se tomaron sobre la marcha en función de las necesidades del momento. Para estructurar estas tareas se utilizó un archivo CSV especialmente diseñado, que recoge el nombre de cada tarea, sus fechas de inicio y fin, su categoría y nivel de avance. Este archivo se utilizó como base para generar visualizaciones del progreso del proyecto y para estructurar la narrativa de la memoria.

El diseño de tareas se realizó con un enfoque realista y modular. Se identificaron bloques funcionales dentro del sistema —como la implementación de distintos procesos de difusión (VE, VP y Sub-VP), los samplers (Euler-Maruyama, Probability Flow ODE, etc.), y las funciones de evaluación— y se asignaron a uno u otro miembro del equipo según su experiencia e interés. Las tareas de arquitectura del sistema, diseño de interfaces, integración

continua, y documentación recayeron principalmente en Ibon, mientras que Adam participó en la experimentación, validación y pruebas con técnicas de generación controlada como la imputación o el muestreo condicional.

A lo largo del proyecto se utilizaron ClickUp como plataforma de referencia para el control de tareas, aunque con limitaciones debido a las restricciones de su plan gratuito. A partir del archivo CSV elaborado manualmente, se llevó a cabo una organización clara de los componentes del sistema, incluyendo no solo el código, sino también la documentación, las pruebas y los notebooks de uso que ilustran las capacidades del paquete. Aunque el desarrollo técnico concluyó antes del 1 de abril, la planificación permitió dedicar las semanas posteriores a la validación, la preparación de ejemplos reproducibles y la redacción del informe final.

Este enfoque ha permitido mantener una visión global del proyecto en todo momento, controlar su avance y asegurar que el resultado final sea coherente, funcional y bien documentado. El cronograma que se presenta a continuación refleja no solo el contenido técnico del trabajo, sino también el esfuerzo de planificación y reflexión que ha guiado su desarrollo desde el primer momento.

A continuación, se presenta una previsualización del diagrama de Gantt del trabajo completo, y se puede consultar entero y en detalle en Anexos.

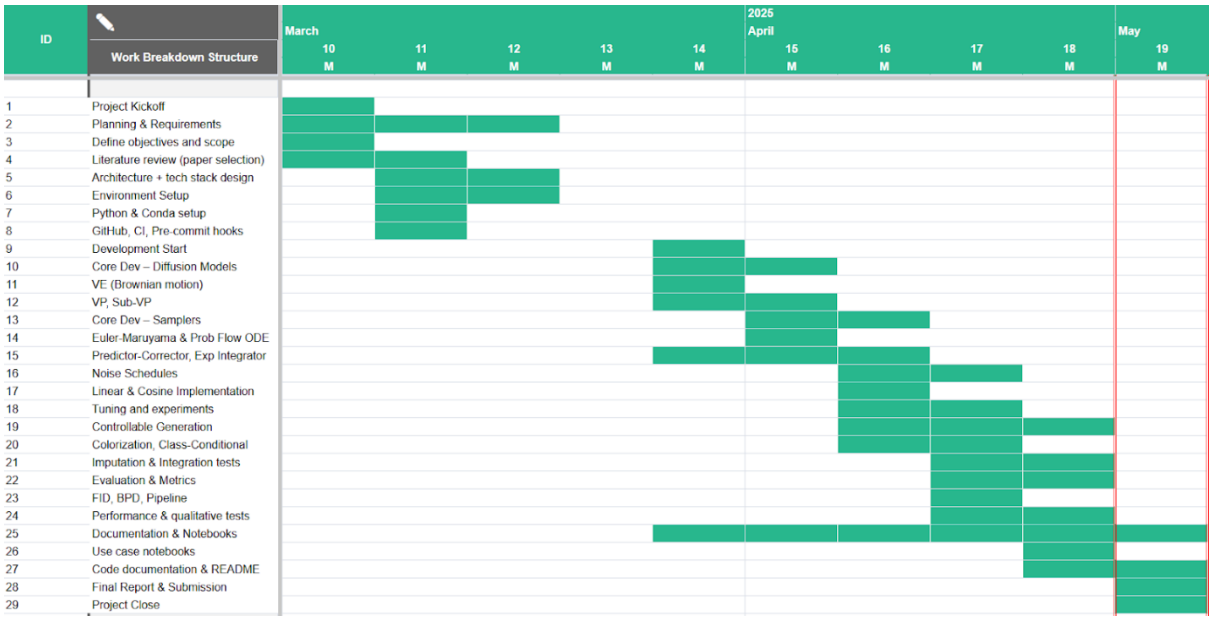


Figura 1: Previsualización del Diagrama de Gantt efectuado

3.2. Análisis de requisitos del proyecto

Requisitos Funcionales

RF1 – Carga de conjuntos de datos personalizados
El sistema debe permitir al usuario introducir sus propios conjuntos de datos de imágenes para tareas de entrenamiento o generación.
RF1.1. El sistema debe aceptar datasets que contengan imágenes en escala de grises o RGB.
RF1.2. Todas las imágenes deben tener las siguientes dimensiones: (número de imágenes, canales, alto, ancho) .
RF1.3. El sistema debe validar que las imágenes cumplen estas condiciones y rechazar las que no lo hagan.

RF2 – Preprocesamiento de imágenes
El sistema debe aplicar preprocesamiento consistente a los datos de entrada.
RF2.1. Las imágenes deben normalizarse al rango [−1,1][-1, 1].
RF2.2. Las imágenes deben convertirse a tensores *PyTorch* y reorganizarse por canal si es necesario.
RF2.3. El sistema debe manejar de forma transparente imágenes en blanco y negro o a color.

RF3 – Selección del modelo de difusión
El sistema debe permitir seleccionar el tipo de proceso de difusión antes del entrenamiento.
RF3.1. Deben estar disponibles las variantes VP, Sub-VP y VE.
RF3.2. Las variantes VP y Sub-VP deben ser compatibles con todos los métodos de muestreo implementados.
RF3.3. La variante VE no puede ser usada con el método Exponential Integrator.
RF3.4. El sistema debe advertir al usuario si se elige una combinación no válida.

RF4 – Configuración del planificador de ruido
El sistema debe permitir seleccionar el noise schedule para el proceso de difusión.
RF4.1. Deben implementarse al menos los *schedules* de tipo lineal y cosenoidal.
RF4.2. La elección del schedule debe reflejarse tanto en la fase de entrenamiento como en la de generación.

RF5 – Entrenamiento del modelo *score-based*
El sistema debe proporcionar una función que entrene un modelo score basado en aprendizaje supervisado.
RF5.1. El entrenamiento debe registrar la evolución de la función de pérdida.
RF5.2. El usuario debe poder ajustar parámetros como épocas, *batch size* y *learning rate*.
RF5.3. Debe existir una función explícita llamada desde el exterior, por ejemplo entrenar(), que encapsule el proceso completo.

RF6 – Generación de imágenes mediante muestreo
El sistema debe ofrecer una función para generar imágenes desde ruido.
RF6.1. Esta función debe implementar el proceso inverso de difusión utilizando un método de integración compatible.
RF6.2. El modelo debe generar un lote de imágenes y visualizar los resultados en el

entorno.
RF6.3. La función debe ser invocable externamente, por ejemplo *sample_and_plot()*, sin necesidad de modificar el código base.

RF7 – Selección del integrador de muestreo
El sistema debe permitir elegir el método de integración para generar imágenes.
RF7.1. Deben estar disponibles los métodos: Euler-Maruyama, Predictor-Corrector, Probability Flow ODE y Exponential Integrator.
RF7.2. La selección debe ser configurable por el usuario antes de iniciar el muestreo.

RF8 – Generación controlada por clase

El sistema debe permitir condicionar la generación a clases específicas cuando el dataset lo permita.
RF8.1. El usuario debe poder especificar una clase (por ejemplo, dígitos en *MNIST*).
RF8.2. El sistema debe verificar que el modelo ha sido entrenado de forma condicional antes de proceder.

RF9 – Imputación de imágenes incompletas
El sistema debe admitir la reconstrucción de imágenes parcialmente observadas.
RF9.1. El usuario debe poder definir una máscara binaria que indique las regiones observadas.
RF9.2. El modelo debe completar las regiones ausentes mediante muestreo condicionado.
RF9.3. Deben visualizarse tanto la imagen incompleta como la reconstruida.

RF10 – Evaluación de la calidad de las imágenes generadas
El sistema debe ofrecer métricas cuantitativas de calidad.
RF10.1. Deben implementarse las métricas *Bits Per Dimension (BPD)*, *Fréchet Inception Distance (FID)* e *Inception Score (IS)*.

RF11 – Guardado y carga de modelos entrenados
El sistema debe permitir guardar modelos entrenados y cargar modelos preexistentes.
RF11.1. El usuario debe poder especificar la ruta donde desea guardar el modelo.
RF11.2. El sistema debe guardar tanto los pesos como la configuración empleada.
RF11.3. El usuario debe poder cargar un modelo desde cualquier ruta válida para realizar muestreo o evaluación.

RF12 – Documentación de uso y notebooks ejecutables
El sistema debe incluir ejemplos completos del flujo de uso.
RF12.1. Deben entregarse notebooks independientes para las tareas de entrenamiento, generación y evaluación.
RF12.2. Los notebooks deben incluir explicaciones breves y celdas ordenadas, ejecutables sin intervención manual.

RF13 – Pruebas del sistema
El sistema debe incluir pruebas para garantizar la estabilidad básica del software.
RF13.1. Deben implementarse tests unitarios simples para funciones clave, como carga de datos, normalización o muestreo.
RF13.2. Las pruebas deben ejecutarse automáticamente antes del entrenamiento mediante un entorno de integración continua (si está habilitado).

RF14 – Validación mínima del rendimiento del modelo
El sistema debe ser capaz de entrenar modelos que generen imágenes razonables en datasets sencillos.
RF14.1. En el dataset MNIST, el modelo debe alcanzar una calidad visual aceptable en las muestras generadas.
RF14.2. En el dataset CIFAR-10, el modelo debe generar muestras borrosas pero con estructuras visuales reconocibles.

Requisitos No Funcionales

RNF1: Compatibilidad con distintos tipos de hardware
El sistema debe poder ejecutarse tanto en GPU como en CPU, aunque se recomienda el uso de GPU para acelerar los procesos de entrenamiento y muestreo. En caso de no disponer de GPU, el sistema debe seguir funcionando correctamente, aunque con mayores tiempos de espera. No debe ser necesario realizar modificaciones en el código para cambiar entre uno u otro dispositivo, ya que la detección debe hacerse automáticamente.

RNF2: Modularidad del código
El diseño del sistema debe estar organizado de forma modular, permitiendo que las distintas partes (modelo de difusión, tipo de *sampler*, planificador de ruido o métodos de conditioning) puedan sustituirse o configurarse de manera independiente. Esta separación facilita el mantenimiento del código, la experimentación con nuevas ideas y la extensión del sistema con funcionalidades adicionales.

RNF3: Facilidad de uso a través de notebooks
El sistema se controla principalmente a través de notebooks de Jupyter, que deben estar bien organizados, explicados y estructurados para que un usuario con conocimientos básicos de aprendizaje automático pueda entender y ejecutar cada paso sin dificultad. Las celdas deben funcionar de manera secuencial y sin errores al ejecutar el notebook desde cero.

RNF4: Optimización razonable del tiempo de ejecución
Los tiempos de ejecución deben mantenerse dentro de márgenes aceptables, especialmente al trabajar con conjuntos de datos pequeños. El entrenamiento de modelos con MNIST o una clase de CIFAR-10 no debería superar una hora en GPU, y el muestreo de imágenes debe realizarse en tiempos que permitan obtener resultados sin esperas excesivas, idealmente en cuestión de segundos o pocos minutos.

RNF5: Gestión automática de modelos
Al finalizar el entrenamiento, el sistema debe guardar automáticamente el modelo generado en una ubicación estándar con un nombre representativo que permita identificar la configuración usada. Además, al iniciar un nuevo experimento, se debe comprobar si ya existe un modelo equivalente y, en ese caso, permitir su reutilización para evitar pérdidas innecesarias de tiempo y recursos.

RNF6: Organización clara de los archivos y carpetas

El sistema debe mantener una estructura de carpetas ordenada para guardar modelos entrenados, resultados generados e imágenes. Esta organización ayuda a no perder

información entre ejecuciones distintas y permite que el usuario identifique fácilmente dónde se almacenan los modelos o los outputs sin tener que revisar el código. Las rutas deben estar definidas de forma coherente y predecible, y los archivos deben tener nombres descriptivos para reconocer su contenido.

RNF7: Configuración sencilla y explícita de los parámetros
Los parámetros clave de cada experimento deben estar recogidos de forma clara al inicio de los notebooks o scripts, permitiendo al usuario ajustar fácilmente aspectos como el tipo de modelo, el número de pasos, el método de conditioning o el dataset a utilizar. Esta accesibilidad facilita la exploración de variantes sin necesidad de modificar partes profundas del código.

Casos de uso

Caso de uso 1: Entrenamiento de un modelo desde cero

Un usuario quiere entrenar un modelo de difusión score-based con un conjunto de datos propio (por ejemplo, MNIST).
Flujo:

1. El usuario proporciona una carpeta con imágenes propias, en escala de grises o RGB.

2. A través de un script o notebook, indica el tipo de proceso de difusión (VP, Sub-VP o VE), el planificador de ruido (lineal o cosenoidal) y otros parámetros como épocas, *batch size* y *learning rate*.

3. El sistema entrena el modelo score, muestra la pérdida durante las épocas y guarda el modelo final y su configuración.

Caso de uso 2: Generación de imágenes a partir de ruido

El usuario desea generar nuevas imágenes a partir de un modelo entrenado.
Flujo:

1. El usuario carga un modelo previamente entrenado desde la ruta deseada.

2. Elige un método de integración compatible (por ejemplo, *Predictor-Corrector*).

3. Llama a la función *sample_and_plot()* con los parámetros deseados (número de muestras, pasos, etc.).

4. El sistema genera imágenes desde ruido y las visualiza.

Caso de uso 3: Evaluación de la calidad del modelo

El usuario desea comparar la calidad de varios modelos entrenados.
Flujo:

1. El usuario carga un modelo y genera un conjunto de imágenes.
2. El sistema calcula las métricas BPD, FID e IS para estas muestras.
3. El usuario visualiza y compara los resultados, registrados junto al experimento.

Caso de uso 4: Generación condicional por clase

El usuario quiere generar imágenes que representen una clase específica (por ejemplo, dígitos "3").
Flujo:

1. Entrena un modelo condicional o carga uno ya entrenado.
2. Llama a la función de muestreo, indicando la clase deseada.
3. El sistema genera imágenes condicionadas a esa etiqueta.

Caso de uso 5: Imputación de imágenes incompletas

El usuario tiene imágenes con regiones ausentes y quiere que el modelo las complete.
Flujo:

1. Proporciona imágenes incompletas y una máscara binaria que indica las regiones observadas.
2. El sistema aplica muestreo condicionado para completar la imagen.
3. Se visualizan tanto la imagen original, como la incompleta y la reconstruida.

3.3. Diseño

El diseño del proyecto se ha estructurado cuidadosamente para proporcionar una arquitectura modular, extensible y reutilizable centrada en modelos generativos por difusión. A nivel de organización, se ha optado por una separación clara entre componentes: los procesos de difusión, los integradores numéricos (*samplers*), la arquitectura neuronal (*ScoreNet*), el entrenamiento, la generación de imágenes, la configuración por *YAML* y los sistemas de condicionamiento como la imputación o la generación por clase. Esta separación permite experimentar con distintas variantes (e.g., cambiar de VP a VE o alternar samplers como Euler-Maruyama, Predictor-Corrector, etc.) sin alterar el núcleo del sistema.

La clase central *DiffusionExperiment* actúa como orquestador principal, ocultando la lógica interna al usuario final y permitiendo configurar todo el sistema a partir de un diccionario o un archivo YAML, de forma profesional y escalable. El diseño admite tanto el uso interactivo desde scripts como desde una línea de comandos o configuración en proyectos reales. La arquitectura de red usada es una *U-Net* adaptada con *embeddings* de Fourier para el tiempo y capas adicionales para manejar el conditioning, lo que permite al modelo aprender distribuciones condicionales, crucial en tareas como imputación o generación controlada por clase.

Se ha hecho hincapié en separar lógica de entrenamiento y de sampling, respetando las ecuaciones de evolución del score en cada caso. Asimismo, el sistema incluye soporte completo para *logging*, fijación de *seeds*, evaluación y visualización, así como integración futura con métricas como FID. Todo este diseño permite al usuario experimentar, analizar y escalar su trabajo con facilidad, manteniendo una base robusta y profesional.

A continuación se proporciona una *preview* del diagrama de clases del sistema, que se podrá consultar completo y de manera más cómoda acudiendo a la sección de Anexos.

Figura 2: Previsualización del Diagrama de Clases efectuado

3.4. Validación y pruebas

Para verificar que la funcionalidad interna del proyecto es la esperada, se han realizado tanto pruebas unitarias como pruebas de rendimiento para todos los componentes del sistema de generación de imágenes.

En primer lugar, se han desarrollado pruebas unitarias que validan el comportamiento de cada uno de los procesos de difusión y de los samplers implementados en el proyecto. Estas pruebas comprueban que las salidas son coherentes y que el funcionamiento de las funciones internas es el esperado bajo ciertas condiciones.

En cuanto a las pruebas de rendimiento, se ha creado un script que se encarga de medir los tiempos de ejecución de cada fase del proceso. Para ello, se han evaluado todas las combinaciones posibles entre los tres elementos principales: el tipo de proceso de difusión, el sampler utilizado y el *scheduler*. En total, se han analizado 19 combinaciones distintas, utilizando un entrenamiento con 10 épocas. Además, se ha comparado el rendimiento en CPU frente a GPU, para comprobar que la eficiencia mejora con el uso de GPU.

Todos los resultados de estas pruebas (de rendimiento) se han almacenado para poder realizar un análisis posterior claro y ordenado. Se han generado gráficas que ilustran los tiempos de ejecución para visualizar las diferencias que existen por tipo de hardware (CPU vs GPU) o por proceso de difusión.

La visualización de los resultados obtenidos puede consultarse libremente en los Anexos del proyecto.

3.5. Desarrollo y garantía de calidad de software

Durante el desarrollo del sistema, se han aplicado diversas prácticas de calidad orientadas a mejorar la mantenibilidad, la reproducibilidad y la fiabilidad del código. El proyecto se ha gestionado mediante un repositorio en *GitHub*, lo que permite control de versiones, colaboración ordenada y trazabilidad de los cambios realizados.

Como política de estilo, se han seguido las guías de codificación de *PEP8* para Python, utilizando herramientas como *black* para el formateo automático y *flake8* para la detección de errores sintácticos y advertencias. La organización del código en módulos, funciones reutilizables y notebooks bien estructurados facilita su comprensión y extensión.

Se han implementado tests automatizados básicos para verificar componentes clave del sistema, como la correcta aplicación de las transformaciones de ruido, la carga de datos y el comportamiento de los *samplers*.

En cuanto a la documentación del proyecto, se ha optado por hacer uso de generación automática de esta, puesto que produce resultados muy efectivos en poco tiempo y permite obtener documentación interactiva y fácilmente usable e interpretable. En este caso, se ha utilizado *DeepWiki*, una herramienta de documentación automática desarrollada por *CognitionLabs*.

3.6. Desarrollo y garantía de calidad de software

Licencia de uso

Dado que este proyecto ha sido desarrollado en el contexto de una asignatura universitaria, su distribución se realiza con fines exclusivamente académicos. El código puede utilizarse y modificarse libremente en ese contexto, sin un esquema formal de licencia comercial. En caso de reutilización externa, se recomienda acompañar el código de una nota que aclare su origen académico y su carácter experimental.

Aspectos éticos

Aunque el sistema permite generar imágenes sintéticas, su alcance ha estado restringido a datasets públicos y simples (MNIST y CIFAR-10), sin riesgo de replicar patrones discriminatorios ni de generar contenidos inapropiados. No se han utilizado datos sensibles ni modelos pre entrenados con sesgos externos.

4

Ejemplos de uso

Para mostrar el funcionamiento y las capacidades del sistema, se han preparado varios notebooks en los que se enseña el uso de las herramientas ofrecidas: entrenamiento, generación de imágenes, evaluación y aplicaciones avanzadas. En cada uno de ellos se intenta exponer el manejo del modelo y facilitar su comprensión.

En el primer notebook se realiza el entrenamiento de modelos utilizando únicamente las imágenes del dígito 3 del MNIST. Aquí se explican y se implementan los tres tipos de procesos de difusión disponibles: VE (*Variance Exploding*), VP *(Variance Preserving*) y sub-VP. Se muestra al usuario cómo utilizar las diferentes configuraciones de entrenamiento, incluyendo los esquemas de ruido (lineal o coseno). Los modelos generados en este paso se guardan en una carpeta *checkpoints* y se reutilizan en los siguientes notebooks.

El segundo notebook está centrado en el proceso de muestreo y visualización. A partir de los modelos entrenados el notebook 01, se prueban los distintos algoritmos de generación de imágenes: *Euler-Maruyama*, *Predictor-corrector*, *Probability flow ODE* y *Exponential integrator*. Se explican los efectos de cambiar ciertos hiperparámetros del muestreo en cada sampler ,además de como modificar el número de pasos de los samplers. En esta demo se visualizan los resultados de todos los samplers combinados con los 3 modelos de difusión entrenados.

En el tercer notebook se aplican tanto el entrenamiento como el muestreo explicados en las demos anteriores, pero esta vez sobre un dataset a color. Se utiliza la clase *'boat'* (8) del dataset *CIFAR-10*. Este caso sirve para comprobar que el sistema es capaz de generalizar y adaptarse a la generación de imágenes a color, mostrando que apenas hay diferencias respecto a la generación en blanco y negro.

El cuarto notebook está dedicado a la evaluación del rendimiento de los modelos. Es un notebook muy básico que toma un modelo previamente entrenado y muestra de forma sencilla métricas como el BPD, FID (Fréchet Inception Distance) e IS (Inception Score), que permiten medir la calidad de las imágenes generadas. Para cada una de las métricas se da una breve explicación para que el usuario comprenda su funcionamiento.

Finalmente, el quinto notebook (05) trata sobre el uso del modelo para el condicionamiento y la imputación de imágenes. En el caso del conditioning, se generan imágenes a partir de una información previa o parcial, como una clase o una sección visible. En la parte de imputación, se demuestra cómo el modelo es capaz de completar imágenes que tienen partes borradas o faltantes, generando contenido coherente con el resto de la imagen. Esta demo se realiza con distintos dígitos del MNIST y se muestra como entrenar los modelos para realizar estas tareas. Tras ello se visualizan los resultados y se finaliza la demo.

5

Conclusión

Este trabajo ha mostrado cómo los modelos de difusión pueden usarse de forma efectiva para generar imágenes controladas y manipular datos visuales de manera precisa. Entrenando desde cero una red basada en U-Net, junto con información sobre el tiempo y la clase deseada, se ha conseguido incluso hacer cosas como guiar el proceso generativo para producir imágenes que responden claramente a una categoría indicada. Para ello, se aplicó una técnica conocida como *classifier-free guidance*, que ajusta el resultado durante la generación usando gradientes del embedding de clase.

Los resultados muestran que el modelo es capaz de generar imágenes coherentes y reconocibles a partir de ruido, lo cual demuestra que ha aprendido relaciones útiles entre las clases y las formas visuales. También se ha visto que ajustar la intensidad de la guía según el paso de tiempo mejora el equilibrio entre precisión visual y variedad en las muestras.

Sin embargo, trabajar con recursos computacionales limitados ha supuesto algunas restricciones. Por ejemplo, ha sido necesario reducir el tamaño del modelo y el número de pasos de entrenamiento, lo que afecta a la calidad final de las imágenes en comparación con sistemas más grandes. A pesar de estas limitaciones, los resultados obtenidos son muy prometedores: muestran que incluso con medios modestos, los modelos de difusión pueden generar imágenes realistas y controladas, y abren el camino a aplicaciones como la edición parcial, la reconstrucción de imágenes o la creación de contenido visual a partir de descripciones estructuradas.

Bibliografía

Papers y artículos académicos

[1] Dhariwal, P., & Nichol, A. (2021). Diffusion models beat GANs on image synthesis. arXiv. https://arxiv.org/abs/2105.0523

[2] Ho, J., Jain, A., & Abbeel, P. (2020). *Denoising diffusion probabilistic models*. arXiv. https://arxiv.org/abs/2006.11239

[3] Zhao, W., Bai, L., Rao, Y., Zhou, J., & Lu, J. (s.f.). UniPC: A unified predictor-corrector framework for fast sampling of diffusion models. Tsinghua University https://arxiv.org/pdf/2302.04867

[4] Zhang, Q., & Chen, Y. (s.f.). Fast sampling of diffusion models with exponential integrator. Georgia Institute of Technology. https://arxiv.org/pdf/2204.13902

[5] Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., & Poole, B. (2021). Score-based generative modeling through stochastic differential equations. International Conference on Learning Representations (ICLR). https://openreview.net/forum?id=PxTIG12RRHS

[6] Suárez, A. (2025). Diapositivas de teoría y prácticas del curso de Aprendizaje Automático [Material de clase]. Universidad Autónoma de Madrid.

Recursos y blogs

[7] Song, Y. (2021). Diffusion score matching explained [Blog]. https://yang-song.net/blog/2021/score/

[8] Weng, L. (2021). What are diffusion models? https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

[9] Yi, M. (2023). Probability flow ODEs: A gentle mathematical blog. https://mingxuan-yi.github.io/blog/2023/prob-flow-ode/#21-probability-flow-odes

Vídeos y tutoriales

[10] CVPR. (2022). Tutorial on diffusion models. https://cvpr2022-tutorial-diffusion-models.github.io/

[11] DeepLearning.ai. (2022). How diffusion models work (short) [Video]. YouTube. https://www.youtube.com/watch?v=9zKuYvjFFS8

[12] Henry AI Labs. (2022). Score-based diffusion explained [Video]. YouTube. https://www.youtube.com/watch?v=B4oHJpEJBAA

[13] Kilcher, Y. (2022). Diffusion models explained [Video]. YouTube. https://www.youtube.com/watch?v=w8YQcEd77_o

Herramientas y entornos utilizados

[14] ClickUp. (s.f.). ClickUp [Herramienta de gestión de proyectos]. https://clickup.com

[15] CompVis. (2022). Latent diffusion models [Repositorio GitHub]. https://github.com/CompVis/latent-diffusion

[16] draw.io. (s.f.). Draw.io [Aplicación de diagramación]. https://app.diagrams.net

[17] GitHub, Inc. (s.f.). GitHub [Plataforma de desarrollo colaborativo]. https://github.com

[18] Harris, C. R., et al. (2020). NumPy. Nature, 585(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

[19] Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90–95.

[20] Jupyter Development Team. (2016). Jupyter Notebooks – A publishing format for reproducible computational workflows. In F. Loizides & B. Schmidt (Eds.), Positioning and power in academic publishing: Players, agents and agendas (pp. 87–90). IOS Press.

[21] Microsoft Corporation. (s.f.). Visual Studio Code [Editor de código]. https://code.visualstudio.com

[22] Notion Labs Inc. (s.f.). Notion [Aplicación de productividad]. https://www.notion.so

[23] OpenAI. (2023). ChatGPT [Asistente para apoyo no relacionado con código]. https://chat.openai.com

[24] Python Software Foundation. (2023). Python (versión 3.12) [Lenguaje de programación]. https://www.python.org

[25] Cognition Labs. (2025). *DeepWiki: Documentación interactiva impulsada por IA para repositorios de GitHub* [Software]. https://deepwiki.com

Apéndices

A

Enlace a archivos realizados

A continuación se presenta un código QR con un enlace a los archivos complementarios, como diagramas, creados durante el desarrollo del proyecto.

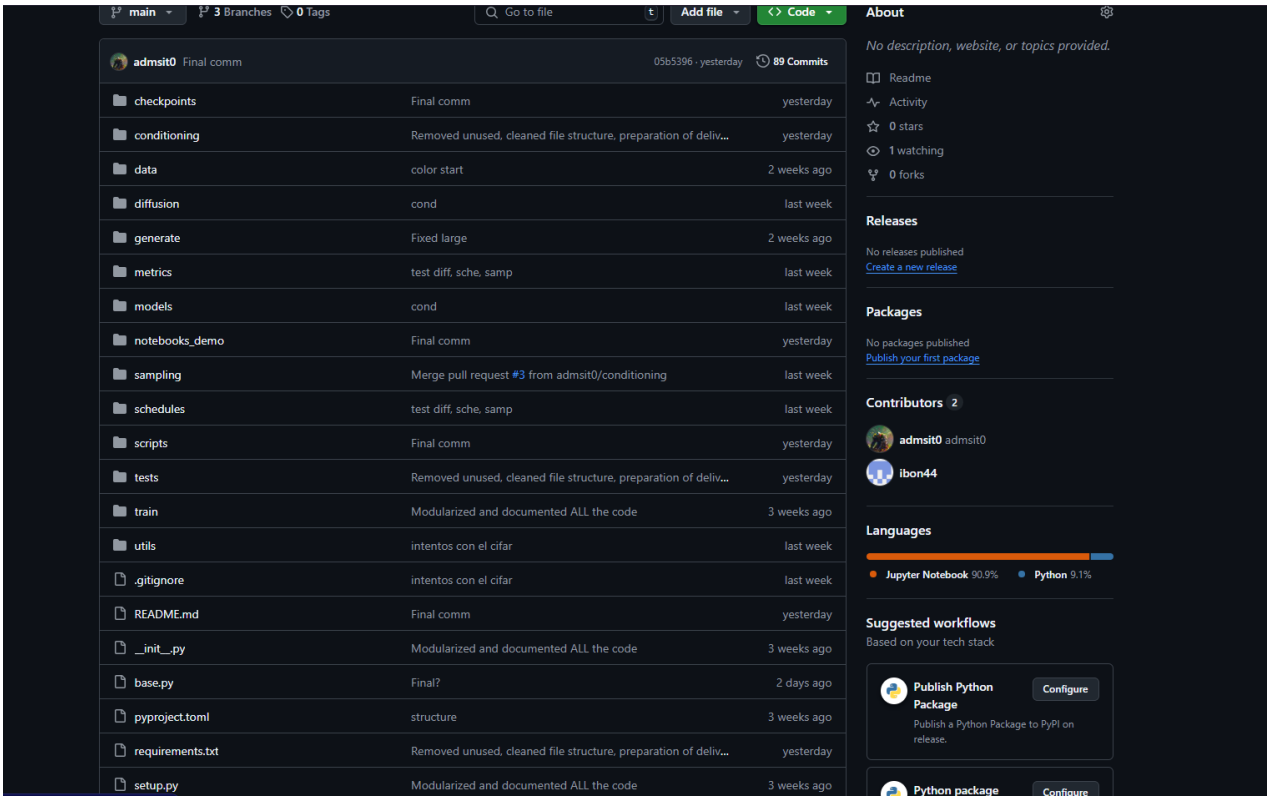
Google drive repo link

B

Codebase pública del proyecto

A continuación se presenta un enlace a los archivos de código creados durante el desarrollo del proyecto.

[imageGenerativeAI - GitHub]

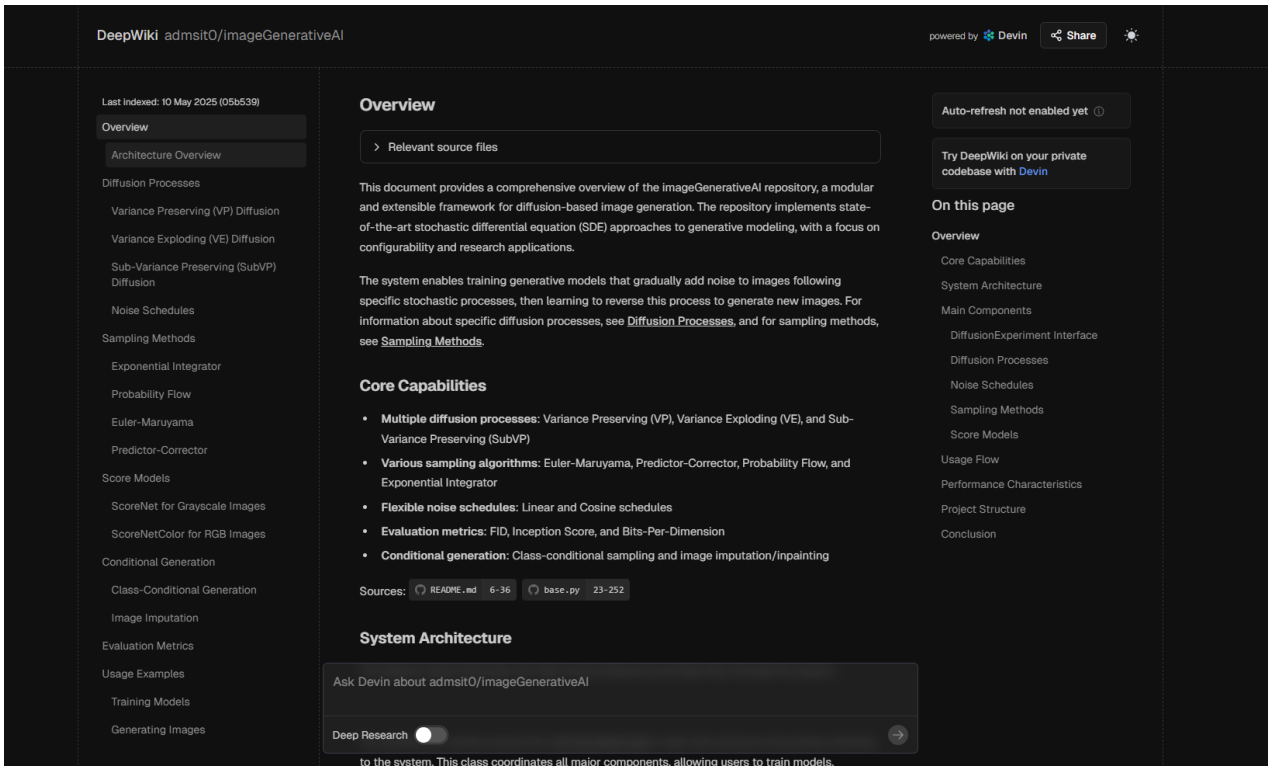


C

Enlace a la documentación

A continuación se presenta un enlace a la documentación generada a modo explicativo y aclarativo del sistema desarrollado.

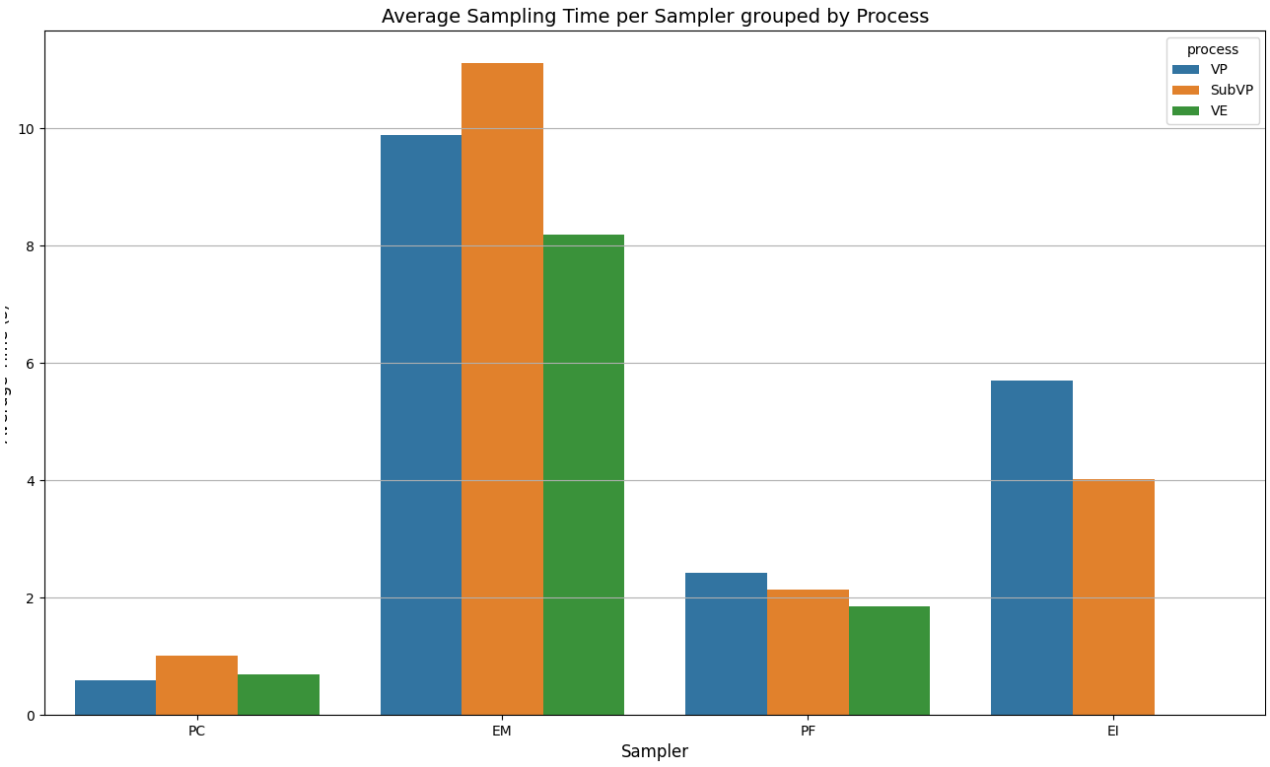
Project documentation

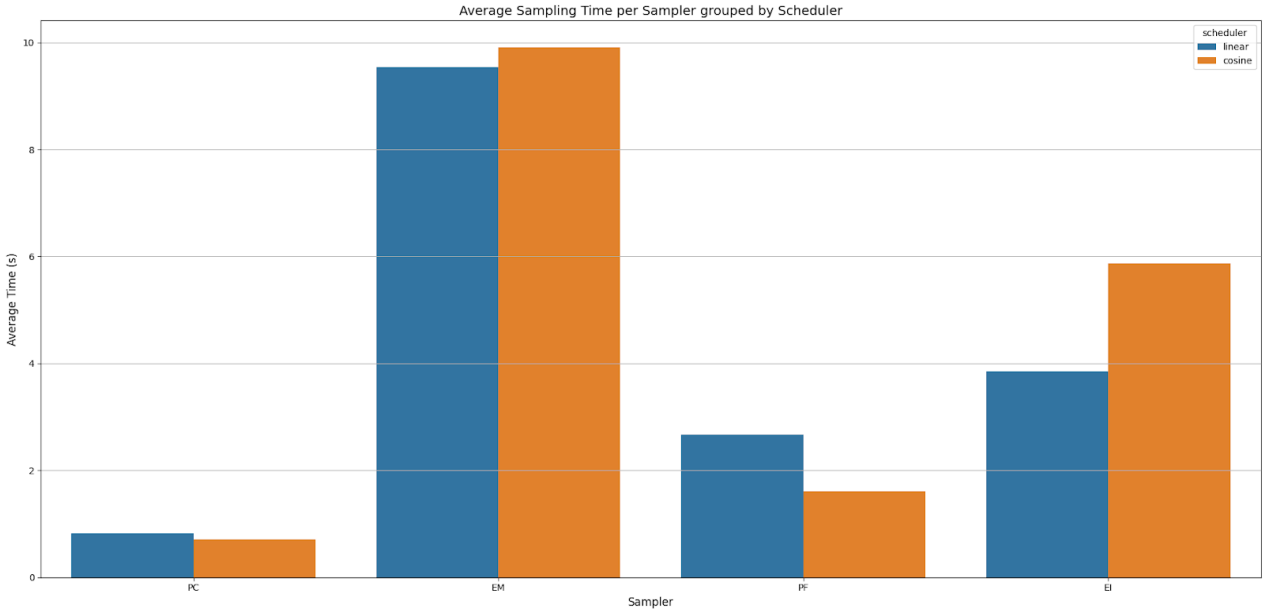


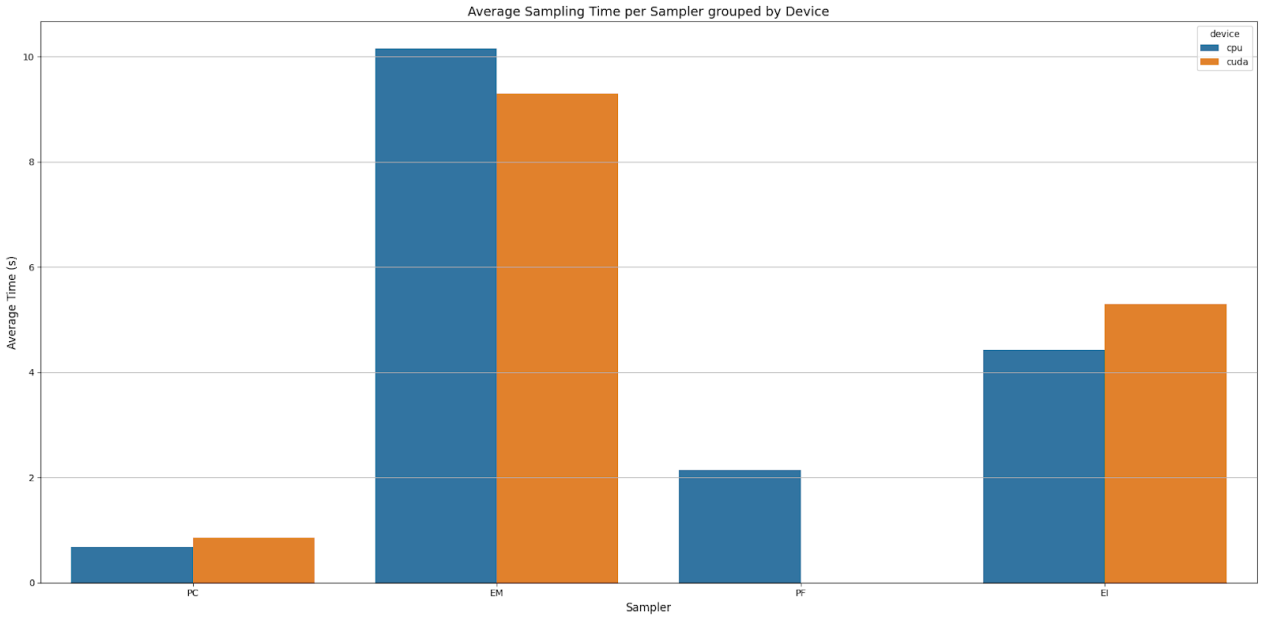
D

Salidas pruebas rendimiento

Sampling y generación







NOTA: las pruebas han sido realizadas sobre un tamaño de muestra muy pequeño, de modo que la relación entre los resultados CPU/GPU no es fiel a la realidad. Para evaluar esto, se puede estudiar la tabla de tiempo de entrenamiento, presentada a continuación.

Entrenamiento

Table 1: Comparación de tiempos promedio de muestreo (en segundos) entre CPU y GPU por tipo de proceso y scheduler. Con 1 imagen del dataset CIFAR-10 ("boat"), durante 500 épocas. Media sobre 2 iteraciones, con precisión de $\pm$ 2 min, y están todos sujetos a variaciones por factores externos como uso de la máquina, procesos de fondo, etc...

Proceso	Scheduler	CPU	GPU
VE	linear	28min	3h10min
VP	linear	34min	4h05min
SubVP	linear	36min	4h02min
VE	cosine	32min	3h17min
VP	cosine	30min	3h57min
SubVP	cosine	41min	4h06min

E

Fórmulas, desarrollos teóricos y justificaciones

Procesos de Difusión en Modelos Generativos

1. Proceso Forward (SDE directo)

$$d\mathbf{x}_t = f(\mathbf{x}_t, t)\,dt + g(t)\,d\mathbf{W}_t \tag{1}$$

donde:

- $f(\mathbf{x}_t, t)$: coeficiente de **drift**,
- $g(t)$: coeficiente de **difusión**,
- $\mathbf{W}_t$: proceso de Wiener (ruido gaussiano).

2. Proceso Backward (SDE inverso)

$$d\mathbf{x}t = \left[f(\mathbf{x}_t, t) - g(t)^2 \nabla \mathbf{x} \log p_t(\mathbf{x})\right] dt + g(t)\,d\mathbf{W}_t \tag{2}$$

3. ODE de Flujo de Probabilidad (versión determinista)

$$\frac{d\mathbf{x}t}{dt} = f(\mathbf{x}_t, t) - \frac{1}{2} g(t)^2 \nabla \mathbf{x} \log p_t(\mathbf{x}) \tag{3}$$

Variantes de Proceso de Difusión

4. Variance Exploding (VE)

Forward:

$$d\mathbf{x}_t = g(t)\,d\mathbf{W}_t \tag{4}$$

Solución:

$$\mathbf{x}_t = \mathbf{x}_0 + \sigma_t \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(0, \mathbf{I}), \quad \sigma_t^2 = \int_0^t g^2(s)\,ds \tag{5}$$

Backward:

$$d\mathbf{x}t = -g(t)^2 \nabla \mathbf{x} \log p_t(\mathbf{x})\,dt + g(t)\,d\mathbf{W}_t \tag{6}$$

5. Variance Preserving (VP) / Ornstein-Uhlenbeck

Forward:

$$d\mathbf{x}_t = -\frac{1}{2}\beta(t)\,\mathbf{x}_t\,dt + g(t)\,d\mathbf{W}_t \tag{7}$$

Solución:

$$\mu(\mathbf{x}_0, t) = \mathbf{x}_0 \cdot \exp\left(-\frac{1}{2}\int_0^t \beta(s)\,ds\right) \tag{8}$$

$$\sigma^2(t) = \int_0^t \exp\left(-\int_s^t \beta(u)\,du\right) g^2(s)\,ds \tag{9}$$

Backward:

$$dxt = \left[-\frac{1}{2}\beta(t)\mathbf{x}_t + g^2(t)\nabla \mathbf{x} \log p_t(\mathbf{x}) \right] dt + g(t)\, d\mathbf{W}_t \tag{10}$$

6. Sub-Variance Preserving (Sub-VP)

Forward:

$$d\mathbf{x}_t = -\frac{1}{2}\beta(t)\, \mathbf{x}_t\, dt + g(t)\, d\mathbf{W}_t \tag{11}$$

donde:

$$g(t) = \beta(t) \cdot \left(1 - \exp\left(-2 \int_0^t \beta(s)\, ds\right)\right)$$

Backward:

$$dxt = \left[-\frac{1}{2}\beta(t)\mathbf{x}_t + g^2(t)\nabla \mathbf{x} \log p_t(\mathbf{x}) \right] dt + g(t)\, d\mathbf{W}_t \tag{12}$$

7. Programación del Ruido (*Noise Schedulers*)

Linear

$$\beta_t = \beta_{\min} + \frac{t}{T}(\beta_{\max} - \beta_{\min}) \tag{13}$$

Cosine

$$\alpha_t = \cos^2\left(\frac{t/T + s}{1 + s} \cdot \frac{\pi}{2}\right) \tag{14}$$

$$\beta_t = \min(0.999, 1 - \frac{\alpha_t}{\alpha_{t-1}}) \tag{15}$$

8. Samplers

8. Euler Solver (for SDE)

$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + f(\mathbf{x}_t, t)\Delta t + g(t)\Delta \mathbf{w} \tag{16}$$

donde $\Delta \mathbf{w} \sim \mathcal{N}(0, \Delta t)$

8.1 Exponential Integrator (EXP)

$$\mathbf{x}_{t+\Delta t} = \exp(A\Delta t)\mathbf{x}_t + A^{-1}(\exp(A\Delta t) - I)f(\mathbf{x}_t) \tag{17}$$

8.2 Predictor-Corrector (PC)

Predictor (Euler):

$$\bar{\mathbf{x}}_{t+\Delta t} = \mathbf{x}_t + f(\mathbf{x}_t, t)\Delta t \tag{18}$$

Corrector (Langevin):

$$\mathbf{x}_{t+\Delta t} = \bar{\mathbf{x}}_{t+\Delta t} + \epsilon \nabla_{\mathbf{x}} \log p(\mathbf{x}) + \sqrt{2\epsilon}\mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(0, \mathbf{I}) \tag{19}$$

8.3 Probability Flow ODE (PFLOW)

$$\frac{d\mathbf{x}_t}{dt} = f(\mathbf{x}_t, t) \tag{20}$$

9. Metrics

9.1 Bits-Per-Dimension (BPD)

$$\text{BPD} = -\frac{1}{\log 2} \cdot \frac{1}{D} \log p(\mathbf{x}) \tag{21}$$

9.2 Frechet Inception Distance (FID)

$$\text{FID} = \|\mu_r - \mu_g\|^2 + \text{Tr}\left(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{1/2}\right) \tag{22}$$

9.3 Inception Score (IS)

$$\text{IS} = \exp\left(\mathbb{E}_{\mathbf{x}}\left[D_{\text{KL}}(p(y|\mathbf{x})\|p(y))\right]\right) \tag{23}$$

F

Imágenes de muestra

A continuación se presentan algunas de las imágenes obtenidas a lo largo del proyecto a modo de muestra o demo de su funcionalidad.

Dígito 3 MNIST con Vp (coseno) y Probability Flow ODE

Dígito 3 MNIST con Vp (lineal) y Predictor Corrector

Dígito 3 MNIST con Sub-VP (lineal) y Exponential Integrator

Dígito 3 MNIST con Sub-VP (lineal) y Exponential Integrator

Dígito 3 MNIST con VP (coseno) y Euler-Maruyama

Dígito 3 MNIST con VP (lineal) y Exponential Integrator

Dígito 3 MNIST con Sub-VP (coseno) y Probability Flow ODE

Dígito 3 MNIST con VE y Probability Flow ODE

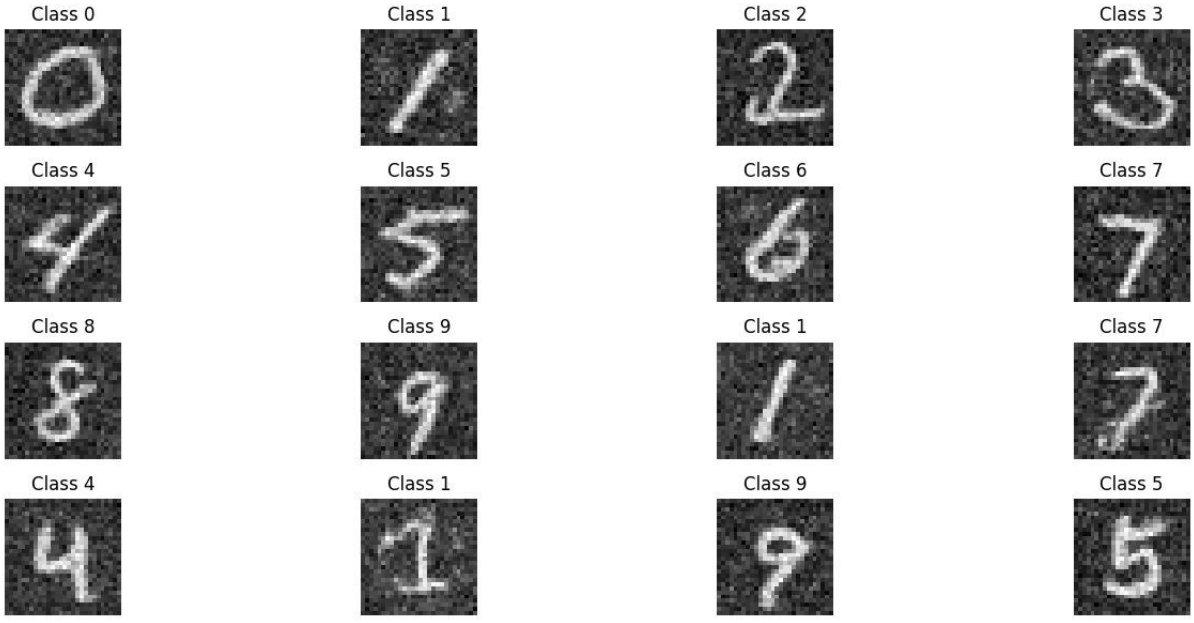
Barcos CIFAR-10 con VE y Predictor Corrector

Barcos CIFAR- 10 con VE y Euler-Maruyama

Barcos con Sub-VP y Predictor Corrector

Barcos con Vp y Predictor Corrector

Generación condicional por clase



Imputación de máscaras/regiones faltantes