Trabajo de Fin de Grado

Generación de datasets sintéticos y auto-etiquetados para entrenamiento de métodos de IA utilizando aprendizaje profundo generativo.

TITULACIÓN: Grado en Ingeniería Informática

AUTOR: Jose R. Peña Seco

TUTORIZADO POR:
Nelson Monzón López y Jonay Suárez Ramírez

Fecha 10 de julio de 2025

ULPGC Universidad de Las Palmas de Gran Canaria | EII ESCUELA DE INGENIERÍA INFORMÁTICA

TFT04

SOLICITUD DE DEFENSA DE TRABAJO DE FIN DE TÍTULO

D. Jose R. Peña Seco, autor del trabajo Generación de datasets sintéticos y autoetiquetados para entrenamiento de métodos de IA utilizando aprendizaje profundo generativo, correspondiente a la titulación Grado en Ingeniería Informática, en colaboración con la empresa Qualitas Artificial Intelligence and Science S.A.

SOLICITA

que se inicie el procedimiento de defensa del mismo, para lo que se adjunta la documentación requerida, haciendo constar que

[X] se autoriza / [ ] no se autoriza la grabación en audio de la exposición y turno de preguntas.

Asimismo, con respecto al registro de la propiedad intelectual/industrial del TFT, declara que:

[ ] Se ha iniciado o hay intención de iniciarlo (defensa no pública).

[X] No está previsto.

Y para que así conste firma la presente. (fecha en firma electrónica)

El/La estudiante

Fdo.————————

A rellenar y firmar **obligatoriamente** por el/la/los/las tutores

En relación con la presente solicitud, se informa:(firmar donde corresponda)

Positivamente (en caso de detección de copia, esta firma quedará invalidada)	Negativamente (justificación en TFT05)
Fdo.————————	Fdo.————————

DIRECTORA DE LA ESCUELA DE INGENIERÍA INFORMÁTICA

Agradecimientos

A mi padre, que ya no está entre nosotros pero sigue más presente que nunca en mi vida.

A toda mi familia, en especial a mi madre, mi hermano, a Leonor, a Anabel y a Gonzalo, por ofrecerme su amor y apoyo incondicional.

A mi pareja, Alba, por haberme acompañado a lo largo de todo este recorrido. Gracias por su paciencia, por ser una fuente constante de inspiración y por compartir sus ideas y conocimientos conmigo.

A mis tutores, Nelson Monzón y Jonay Suárez, cuyo acompañamiento ha sido clave para que este proyecto pudiera llevarse a cabo. Su guía, criterio y apoyo constante han sido determinantes en cada una de las etapas del trabajo. Sin ellos este proyecto no habría dado fruto.

A todo el equipo de QAISC, por su constante disposición para colaborar y brindar apoyo siempre que fue necesario, así como a la empresa por proporcionar los recursos y facilidades que hicieron posible el desarrollo de este trabajo. En especial, a Andrés Alonso por su valiosa orientación y por aportar una perspectiva de negocio clave para definir tanto el problema como el producto final desarrollado.

A todos mis amigos de la universidad. En especial a Antonio Aparicio por haber sido mi compañero durante estos cuatro años de trabajo intenso.

Finalmente, quisiera expresar mi agradecimiento al Centro de Tecnologías de la Imagen (CTIM) de la Universidad de Las Palmas de Gran Canaria (ULPGC)

Resumen

La creación de *datasets* de calidad es uno de los principales desafíos que enfrentan los equipos de inteligencia artificial (IA), tanto en investigación como en el sector privado, al momento de entrenar sus modelos. Este proceso, además de ser tedioso y repetitivo, requiere una inversión significativa de tiempo por parte de investigadores y profesionales del sector.

Aunque en apariencia se trata de una tarea sencilla, la realidad es muy distinta debido a múltiples factores: la dificultad para obtener datos equilibrados, la variabilidad en la calidad de los datos, la dependencia de variables externas, los elevados costos y la lentitud del etiquetado manual, así como la escasez de ejemplos representativos de casos extremos en los que los modelos suelen cometer errores. Superar estas barreras es, paradójicamente, el factor más determinante para el éxito de un proyecto de inteligencia artificial.

El presente Trabajo Fin de Título tiene como objetivo abordar esta problemática mediante el uso de técnicas de aprendizaje profundo generativo, en particular modelos de difusión, para aumentar y generar datasets sintéticos de detección de objetos en el campo de la visión por computador. En concreto, se centra como caso de estudio en el aumento de datos para la detección de embarcaciones en entornos marítimos. Para ello, se propone el desarrollo de varios *pipelines* de generación de imágenes sintéticas autoetiquetadas, junto con una plataforma que permita almacenar, visualizar y re-etiquetar, de ser necesario, dichos *datasets*.

Plabras clave: Modelos de difusión, generación de datasets sintéticos, Inteligencia Artificial, Aprendizaje profundo generativo

Abstract

The creation of high-quality *datasets* is one of the main challenges faced by artificial intelligence (AI) teams, both in research and the private sector, when training their models. This process, in addition to being tedious and repetitive, requires a significant investment of time from researchers and professionals in the field.

Although it may seem like a simple task, the reality is quite different due to multiple factors: the difficulty of obtaining balanced datasets, variability in data quality, dependence on external variables, high costs, the slow pace of manual labeling, and the scarcity of representative examples of extreme cases where models tend to make mistakes. Overcoming these barriers is, paradoxically, the most critical factor for the success of an artificial intelligence project.

This *Final Degree Project* aims to address this issue through the use of generative deep learning techniques, specifically diffusion models, to enhance and generate synthetic datasets for object detection in the field of *computer vision*. As a case study, the work focuses on data augmentation for vessel detection in maritime environments. To achieve this, the development of several *pipelines* for generating self-labeled synthetic images is proposed, along with a *full-stack* platform that enables AI teams to store, visualize and re-label those *datasets* if needed.

Keywords: Diffusion models, synthetic dataset generation, Artificial Intelligence, Generative Deep Learning.

Índice general

Índice de Algoritmos

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

¨Lo que no puedo crear no lo entiendo.¨

Richard Feynmann

El aprendizaje profundo generativo es una de las tecnologías más innovadoras de la última década, ya que redefine los límites de lo que un ordenador puede lograr. Su impacto radica en la capacidad de replicar cualidades tradicionalmente consideradas exclusivas de la condición humana, como la creatividad, la originalidad y la imaginación, integrándolas en un sistema informático. En la actualidad, existen modelos capaces de conversar, pintar y crear con un nivel de sofisticación que, en tareas específicas, puede ser indistinguible o incluso superar el rendimiento humano.

No obstante, toda esta nueva generación de modelos se basa en una idea clave: modelar la distribución de probabilidad subyacente a un conjunto de datos reales. Una vez aprendida, es posible generar nuevos ejemplos muestreando a partir de ella. Sin embargo, lograrlo no es sencillo: requiere arquitecturas capaces de capturar patrones complejos y mecanismos específicos para realizar un muestreo coherente y realista. Se puede observar un diagrama de este proceso en la figura 1.1.

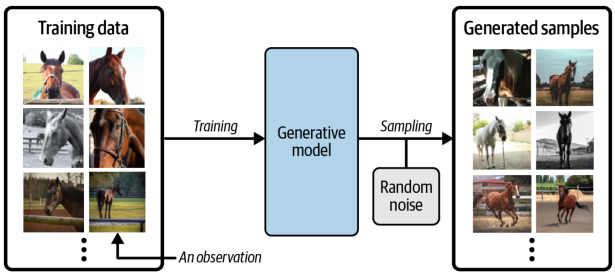


Ilustración 1.1: Un modelo generativo entrenado para crear imágenes fotorealistas de caballos [**Fuente**: [24], pag. 32]

La capacidad de los modelos generativos para producir datos similares a un conjunto original los convierte en una herramienta ideal para la generación y aumento sintético de *datasets*. Este Trabajo de Fin de Grado tiene como objetivo desarrollar varios *pipelines* de generación de datos sintéticos para tareas de detección de objetos mediante modelos de difusión. Dado el amplio alcance de este objetivo, el trabajo se centra en un caso de uso concreto: el aumento de datos para la detección de embarcaciones en entornos marítimos. Además, se propone construir un sistema que permita a los investigadores ejecutar estos *pipelines*, visualizar, re-etiquetar y filtrar los ejemplos generados, y exportarlos en un formato compatible con el entrenamiento de modelos. Para ello, se integran servicios de inteligencia artificial, herramientas web y utilidades de línea de comandos.

1.1. Antecedentes y propósito

Este trabajo se desarrolla en colaboración con *Qualitas Artificial Intelligence and Science* y el *Centro de Tecnologías de la Imagen*, con el objetivo de impulsar la investigación en inteligencia artificial aplicada a los retos de la economía azul. Como caso de estudio, se aborda la mejora del *tracker* de embarcaciones marítimas utilizado en producción dentro de la empresa.

En este contexto, el objetivo del TFG es optimizar la generación y adquisición de datos, aumentando la eficiencia y productividad de sus líneas de investigación.

1.2. Objetivos

El objetivo principal de este trabajo es desarrollar *pipelines* genéricos para la generación y ampliación de *datasets* en tareas de visión por computador, en concreto, detección de objetos.

Paralelamente, se plantea el desarrollo de un sistema que facilite a los investigadores la gestión de sus *datasets*, permitiendo la generación de datos sintéticos mediante los *pipelines* mencionados, la re-etiquetación de datos en caso de errores y la descarga de los conjuntos de datos en formatos adaptados a sus necesidades.

Este objetivo puede desglosarse en los siguientes componentes:

1. Aplicar y reforzar los conocimientos del alumno en sistemas inteligentes y herramientas para el análisis de datos.

2. Adquirir una base sólida en los fundamentos del aprendizaje profundo generativo y ampliarla mediante el estudio del estado del arte en esta área.

3. Implementar y desplegar modelos del estado del arte en aprendizaje profundo generativo, evaluando su rendimiento y posibles aplicaciones en la creación de *datasets* sintéticos.

4. Desarrollar uno o varios *pipelines* específicos para la generación de *datasets* sintéticos utilizando las metodologías avanzadas previamente estudiadas.

5. Generar uno o varios *datasets* sintéticos empleando los *pipelines* desarrollados.

6. Realizar una comparación exhaustiva entre la calidad de los *datasets* sintéticos generados y *datasets* reales.

7. Documentar de manera detallada y estructurada todo el proceso, incluyendo las etapas de diseño, pruebas y resultados.

8. Fomentar el trabajo colaborativo y la interacción con profesionales de QAISC, brindando al estudiante una experiencia inmersiva en un entorno laboral y en la dinámica de proyectos de I+D

1.3. Principales aportaciones

Las contribuciones de este TFT al entorno socioeconómico y científico son significativas. Como se observa en la figura 1.2, la visión por computador desempeña un papel fundamental en múltiples sectores de la economía global, permitiendo avances en áreas como el reconocimiento facial, el análisis de comportamiento y la detección de anomalías, mejorando así la seguridad en espacios públicos, infraestructuras críticas y propiedades comerciales.

En el ámbito local, la empresa colaboradora *QAISC* aplica estas tecnologías para fortalecer la economía azul en Canarias mediante el análisis costero y la detección y seguimiento de embarcaciones, optimizando la gestión del tráfico marítimo y la seguridad en las aguas del archipiélago.

Por ello, la aceleración en la creación y adquisición de datos no solo impulsa el desarrollo tecnológico y científico, sino que también tiene un impacto directo en la competitividad económica tanto a nivel global como en el contexto local canario, promoviendo la innovación y la sostenibilidad en sectores clave.

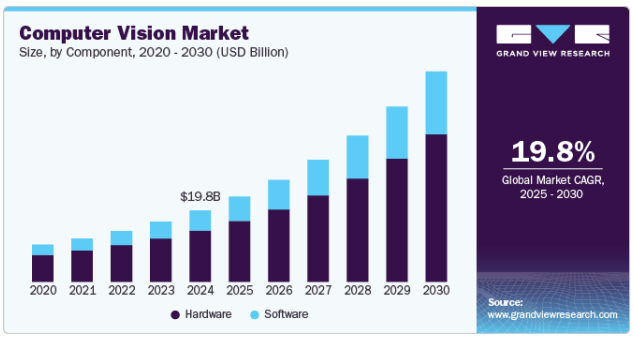


Ilustración 1.2: Evolución del mercado de la visión por computador [GrandViewResearch]

1.4. Competencias específicas

En esta sección, se aborda la justificación de competencias específicas cubiertas durante el desarrollo de TFG

"**CI15** Conocimiento y aplicación de los principios fundamentales y técnicas básicas de los sistemas inteligentes y su aplicación práctica."

La competencia **CI15** se respalda mediante el uso de métodos de IA generativa para la generación de datos sintéticos, demostrando la capacidad de aplicar técnicas de sistemas inteligentes para desarrollar soluciones prácticas.

"**CI13** Conocimiento y aplicación de las herramientas necesarias para el almacenamiento, procesamiento y acceso a los sistemas de información, incluidos los basados en web."

La competencia **CI13** queda acreditada con el diseño y despliegue de la infraestructura web necesaria para la plataforma *full-stack*.

"**CI12** Conocimiento y aplicación de las características, funcionalidades y estructura de las bases de datos, que permitan su adecuado uso, y el diseño y el análisis e implementación de aplicaciones basadas en ellos."

La competencia **CI12** se evidencia en la creación de una base de datos relacional destinada al almacenamiento de los datos de la aplicación web y los *datasets* generados.

"**CI8** Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados."

La competencia **CI8** se justifica por la selección de paradigmas y lenguajes de programación adecuados tanto para la plataforma web como para los *pipelines* de IA. Además, se han utilizado diversas bibliotecas y herramientas *DevOps* como Docker para garantizar un despliegue robusto, seguro y eficiente.

"**CI17** Capacidad para diseñar y evaluar interfaces persona computador que garanticen la accesibilidad y usabilidad a los sistemas, servicios y aplicaciones informáticas."

La competencia **CI17** se respalda con el diseño e implementación de la interfaz *front-end*, facilitando el uso de los *pipelines* desarrollados por los investigadores. Para ello, se empleó *Figma*, siguiendo un proceso de diseño iterativo que abarcó desde *wireframes* y *mockups* hasta la creación del prototipo y el producto final.

1.5. Alineamiento con los objetivos de desarrollo sostenible

Este *TFT* está alineado con el Objetivo de Desarrollo Sostenible (ODS) número 9: *Industria, Innovación e Infraestructuras*. La inteligencia artificial requiere infraestructuras tecnológicas sólidas y grandes volúmenes de datos, cuya obtención es costosa y laboriosa. Este

Cuadro 1.1: Grado de relación del TFT con los objetivos de desarrollo sostenible.

ODS	Grado de relación			
	0 No procede	1 Bajo	2 Medio	3 Alto
1 Fin de la Pobreza	x			
2 Hambre cero	x			
3 Salud y Bienestar	x			
4 Educación de calidad	x			
5 Igualdad de género	x			
6 Agua limpia y saneamiento	x			
7 Energía Asequible y no contaminante	x			
8 Trabajo decente y crecimiento económico	x			
9 Industria, Innovación e Infraestructuras				x
10 Reducción de las desigualdades	x			
11 Ciudades y comunidades sostenibles	x			
12 Producción y consumo sostenibles	x			
13 Acción por el clima	x			
14 Vida submarina	x			
15 Vida de ecosistemas terrestres	x			
16 Paz, justicia e instituciones sólidas	x			
17 Alianzas para lograr objetivos	x			

trabajo aborda esta problemática mediante la generación y ampliación de *datasets* sintéticos con modelos de difusión, facilitando el acceso a datos de calidad y acelerando la investigación en visión por computador.

Además, al proporcionar herramientas para gestionar y re-etiquetar datos de manera eficiente, esta solución fortalece la infraestructura tecnológica necesaria para impulsar aplicaciones de IA en sectores clave como la economía azul. En consecuencia, este trabajo contribuye a la innovación en infraestructuras digitales, optimizando la productividad y el uso eficiente de los recursos tecnológicos.

1.6. Estructura del documento

Este documento está organizado en las siguientes secciones:

1. **Estado del arte y marco teórico:** En esta sección se presenta una revisión del estado del arte y el marco teórico relacionado con la inteligencia artificial, con un enfoque específico en el aprendizaje profundo generativo. Se detallan las técnicas más relevantes que han sido parte de la literatura, aquellas que se aplican actualmente y las métricas utilizadas para evaluar estos modelos. Además, se abordan las técnicas tradicionales de aumento de datos y generación de *datasets*, así como una sección final dedicada al aumento de datos mediante modelos de difusión.

2. **Metodología, tecnologías y modelos de difusión utilizados:** Aquí se describen las metodologías, herramientas y modelos de difusión empleados en el desarrollo del proyecto.
3. **Diseño del sistema:** En esta sección se presenta una visión general del sistema desarrollado en su conjunto, sin profundizar en los detalles específicos de los servicios o módulos que lo conforman.
4. **Generación de datos sintéticos:** En esta sección se describe el trabajo de investigación llevado a cabo durante el diseño e implementación de los distintos *pipelines* de aumento de datos. Asimismo, se analizan los experimentos realizados para evaluar la robustez de los métodos propuestos y los resultados obtenidos.
5. **Plataforma Web:** En esta sección se describe la plataforma web, abarcando tanto el *frontend* como el *backend*.
6. **Conclusiones y trabajo futuro:** Se exponen las conclusiones obtenidas a lo largo del desarrollo del trabajo, así como las posibles líneas de investigación y mejoras para futuros desarrollos.

Ejemplo para el glosario de términos (ver al final del documento):

This final report has been produced using Latex, a tool specially suitable for technical documents that include mathematics.

Capítulo 2

Estado del arte y marco teórico

≪Lo que no puedo crear no lo entiendo.≫

Richard Feynmann

Para abordar los objetivos propuestos en este Trabajo de Fin de Título, es necesario comprender dos ramas distintas pero complementarias: la inteligencia artificial generativa aplicada a la síntesis de imágenes y las técnicas tradicionales de aumento de datos. Ambas convergen en un mismo punto: el aumento de datos utilizando modelos generativos.

En cuanto a la primera rama, este capítulo tiene como propósito establecer el marco teórico mínimo indispensable para comprender el estado del arte actual en modelos generativos, centrándonos en los modelos de difusión latente. Aunque al principio pueda parecer una exposición de diversas arquitecturas sin relación clara entre ellas, veremos que los avances recientes se apoyan fundamentalmente en cuatro arquitecturas clave: los autoencoders variacionales (VAEs), las Redes Generativas Antagónicas (GANs), los Transformers y los Modelos de Difusión (DMs). A continuación, y con el objetivo de entender el estado del arte actual, se analizará la evolución de los modelos que hoy en día ofrecen el mejor rendimiento, los modelos de difusión latente, así como las mejoras teóricas que han hecho posibles dichos resultados.

En lo que respecta a la segunda rama, se presentará un análisis general de las técnicas tradicionales de aumento de datos utilizadas en visión por computador, explorando sus principios, limitaciones y aplicaciones.

Finalmente, ambas líneas confluirán en la generación de datos sintéticos mediante modelos de difusión. Esta nueva perspectiva sobre el aumento de datos no solo amplía el conjunto de datos disponibles, sino que también permite mejorar significativamente el rendimiento de los modelos de aprendizaje automático.

2.1. Aprendizaje profundo generativo

El **aprendizaje profundo generativo** es una rama del aprendizaje profundo cuyo objetivo es entrenar modelos capaces de generar nuevos datos que sean similares a los de un conjunto de datos original.

Para comenzar, necesitamos un **conjunto de datos de referencia**, conocido como *datos de entrenamiento*. Este conjunto está compuesto por numerosos ejemplos del tipo de dato que queremos generar, y cada uno de estos ejemplos se denomina *observación*. Cada observación está formada por un conjunto de *características*; por ejemplo, en el caso de las imágenes, estas características serían los valores individuales de cada píxel.

El objetivo del modelo generativo es aprender las complejas reglas y patrones que gobiernan las relaciones entre las características presentes en las distintas observaciones del conjunto de datos. A diferencia de los modelos deterministas, un modelo generativo debe tener un enfoque **probabilístico**, ya que los datos que observamos no son fijos ni únicos, sino que forman parte de una variabilidad inherente.

De manera más formal, consideremos un conjunto de datos $\mathbf{x}$, tal que $\mathbf{x} \sim p_{\text{real}}(x)$. El objetivo de nuestro modelo es aproximar la distribución de los datos reales mediante una distribución modelada $p_{\text{model}}(x)$, de forma que se cumpla:

$$p_{\text{model}}(x) \approx p_{\text{real}}(x)$$

Bajo esta premisa, para la generación de nuevos ejemplos originales es necesario, a su vez, encontrar la manera de muestrear la distribución aprendida $p_{\text{model}}(x)$. El proceso descrito anteriormente se puede observar en la figura 2.1.

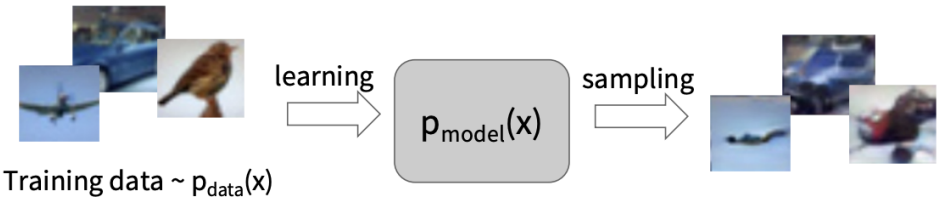


Ilustración 2.1: Ilustración del proceso de creación de un modelo generativo [Fei-Fei Li]

Los *modelos generativos* pueden clasificarse en función de la manera en que modelan la distribución de probabilidad $p_{\text{model}}(x)$ como se observa en la figura 2.2. Esta clasificación se puede dividir en dos grandes categorías:

1. **Densidad implícita:** En este tipo de modelos, la distribución de probabilidad $p_{\text{model}}(x)$ no está definida de forma explícita. En cambio, el modelo se entrena para generar muestras que se asemejan a los datos reales, sin proporcionar una expresión cerrada o evaluable de esta distribución. El ejemplo más famoso de este tipo son las **GANs** de sus siglas en inglés *Generative Adversarial Networks*.

2. **Densidad explícita:** En esta categoría, el modelo define de forma explícita una función para $p_{\text{model}}(x)$. Existen dos subcategorías:

   a) **Modelos de densidad aproximada:** que introducen la noción de espacio latente y intentan modelar una aproximación de la función de densidad conjunta. En esta categoría entran los **VAEs** y los **modelos de difusión**.

   b) **Modelos de densidad tratable:** intentan establecer restricciones en la arquitectura del modelo para que la función de densidad sea fácil de calcular y computable. Los ejemplos más famosos son los **modelos autoregresivos**, como los *Transformers*, y los **modelos de flujos normalizados**.

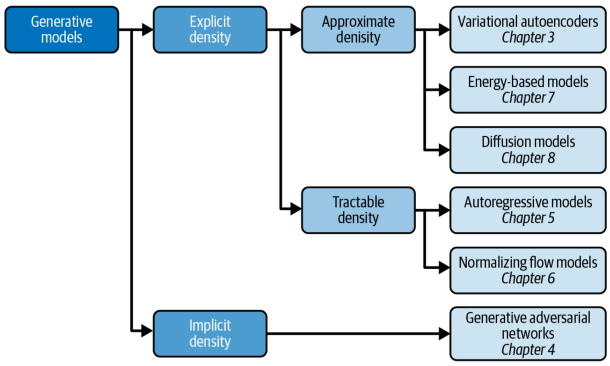


Ilustración 2.2: Esquema de la taxonomía de los modelos generativos [24]

2.1.1. Autoencoders variacionales

Los *autoencoders variacionales* (VAEs, por sus siglas en inglés) son modelos fundamentales dentro del campo del aprendizaje profundo. Fueron propuestos por Diederik P. Kingma y Max Welling en 2013 en su influyente artículo *Auto-Encoding Variational Bayes* [39]. Actualmente, este tipo de arquitecturas *Encoder-Decoder* sigue siendo ampliamente utilizado en una variedad de modelos de *estado del arte*, como forma de transformar datos de entrada en representaciones vectoriales, o *vectores latentes*, dentro de un espacio de menor dimensión, denominado *espacio latente*. Esta técnica ofrece ventajas significativas en términos de eficiencia y flexibilidad en los modelos. Por ejemplo, los modelos de difusión actuales, como se discutirá más adelante en esta memoria, no operan añadiendo y quitando ruido a las imágenes, sino que realizan este proceso sobre un vector latente que encapsula el contenido de la imagen. Por esta razón, muchos modelos de difusión modernos son conocidos como *Latent Diffusion Models* (LDMs).

Comencemos hablando de las arquitecturas *Autoencoder*. Estas fueron propuestas en los primeros días del aprendizaje profundo moderno por David E. Rumelhart et al. [60] en 1987, y más tarde, en 2006, volverían a ser objeto de estudio [29]. La idea básica es entrenar una red neuronal, llamada *Encoder* ($\mathcal{E}$), para que convierta un dato de entrada $\mathbf{x}$ en una representación de menor dimensión dentro de un espacio latente $\mathbf{z}$. Luego, otra red neuronal, denominada *Decoder* ($\mathcal{D}$), debe aprender a reconstruir el dato original $\mathbf{x} \approx \mathbf{x}'$. Este proceso

puede observarse en las ecuaciones 2.1 y 2.2. A su vez se puede visualizar la correspondencia entre una imagen y el espacio latente en la figura 2.3.

$$\mathbf{z} = \mathcal{E}_\theta(\mathbf{x}) \tag{2.1}$$

$$\mathbf{x}' = \mathcal{D}_\alpha(\mathbf{z}) = \mathcal{D}_\alpha(\mathcal{E}_\theta(\mathbf{x})) \tag{2.2}$$

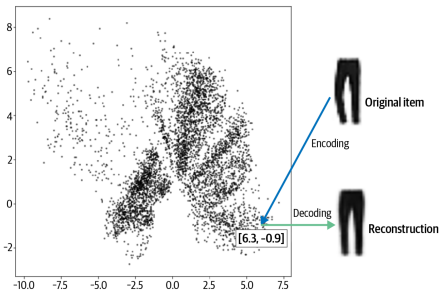


Ilustración 2.3: Ejemplo del proceso de codificación y decodificación al espacio latente con el dataset Fashion-MNIST [24]

El proceso de entrenamiento consiste en minimizar el error de reconstrucción entre la imagen original $\mathbf{x}$ y su reconstrucción $\mathbf{x}'$. Para ello, se puede emplear como función de pérdida el error cuadrático medio o la entropía cruzada. Se pueden observar ambas funciones en las ecuaciones 2.32.4.

$$\mathcal{L}_{\text{MSE}}(\theta, \alpha) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[\|\mathbf{x} - \mathcal{D}_\alpha(\mathcal{E}_\theta(\mathbf{x}))\|^2\right] \tag{2.3}$$

$$\mathcal{L}_{\text{BCE}}(\theta, \alpha) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}\left[-\mathbf{x}^\top \log \mathbf{x}' - (1-\mathbf{x})^\top \log(1-\mathbf{x}')\right] \tag{2.4}$$

El problema del proceso descrito anteriormente es la forma de nuestro espacio latente que estará lleno de regiones vacías y dispersas. Es por ello que la diferencia clave entre un *autoencoder* tradicional y un *VAE* radica en su espacio latente. Mientras que en un *autoencoder* el vector latente no tiene un significado explícito, en un *VAE* el vector latente representa una muestra de una distribución gaussiana multivariable $\mathcal{N}(\mu, \Sigma)$. Los parámetros de dicha distribución, media y desviación, serán modelados mediante redes neuronales. Se puede observar una comparación entre el *VAE* y un *autoencoder* tradicional en la figura 2.4.

Las modificaciones necesarias en nuestra arquitectura son relativamente simples. Para facilitar la explicación, supondremos que la distribución gaussiana que modelamos en el espacio latente es multivariada, pero con matriz de covarianza diagonal. Esto implica que cada dimensión del vector latente se modela de forma independiente mediante una distribución normal univariada. En este caso, el *encoder* genera dos vectores de salida: la media $\boldsymbol{\mu}_\theta$ y la desviación estándar $\boldsymbol{\sigma}_\theta$, ambos de dimensión igual al espacio latente.

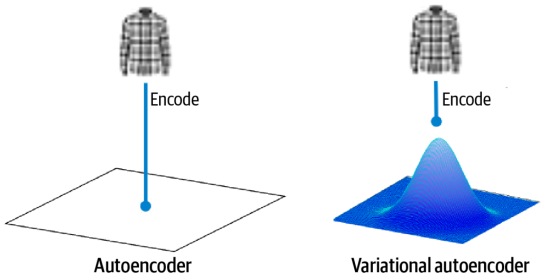


Ilustración 2.4: Diferencias entre los *encoders* en un *autoencoder* y en un *VAE* [24]

Posteriormente, se genera una muestra $\mathbf{z}$ a partir de dicha distribución, que será utilizada como vector latente. Sin embargo, este paso de muestreo introduce una operación estocástica no diferenciable, lo que impide aplicar directamente el algoritmo de *backpropagation* sobre los parámetros de la red. Para resolver este problema, el artículo original [39] propone una técnica conocida como el *truco de reparametrización* (*reparametrization trick*), que permite reescribir la operación de muestreo como una transformación determinista y diferenciable:

$$\mathbf{z} = \boldsymbol{\mu}_\theta + \boldsymbol{\sigma}_\theta \cdot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(0, \mathbf{I}) \tag{2.5}$$

Esta reformulación permite que el gradiente fluya a través de $\boldsymbol{\mu}_\theta$ y $\boldsymbol{\sigma}_\theta$ durante el entrenamiento. Para ilustrar su diferenciabilidad, consideremos una sola dimensión de la variable latente:

$$\frac{\partial z}{\partial \mu} = 1, \quad \frac{\partial z}{\partial \sigma} = \epsilon \tag{2.6}$$

Finalmente, la función de pérdida empleada en los autoencoders variacionales es la conocida como *Evidence Lower Bound* (ELBO), cuya forma negativa se minimiza durante el entrenamiento. Esta función combina dos términos: el primero es el error de reconstrucción del dato original, comúnmente modelado mediante la entropía cruzada (si los datos son binarios), y el segundo es la divergencia de Kullback-Leibler D_{KL}, que mide la distancia entre la distribución latente aproximada $q_\phi(\mathbf{z}|\mathbf{x})$ —obtenida del encoder— y una distribución prior estándar, generalmente una normal multivariada $\mathcal{N}(\mathbf{0}, \mathbf{I})$:

$$\mathcal{L}(\theta, \phi; \mathbf{x}) = -\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x}|\mathbf{z})] + D_{\mathrm{KL}}[q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})] \tag{2.7}$$

Este enfoque permite no solo reconstruir los datos, sino también estructurar el espacio latente de forma continua, compacta y con capacidad generativa, al alinear las distribuciones latentes con una prior conocida.

2.1.2. Redes generativas adversarias (GANs)

Las *redes generativas adversarias* (GANs, por sus siglas en inglés) han sido uno de los marcos más influyentes en el desarrollo de la síntesis de imágenes. Fueron propuestas por *Goodfellow et al.* en 2014 [26], y desde entonces su uso se ha popularizado, especialmente en tareas de generación de imágenes , y fueron los modelos con mayor rendimiento hasta la llegada de los modelos de difusión. Algunos ejemplos destacados incluyen modelos como *DCGAN* [57], *Wasserstein GAN* [4], entre otros.

El funcionamiento de las *GANs* puede interpretarse como un juego *minmax* entre dos redes neuronales: un *discriminador* $\mathcal{D}(x)$, cuya tarea es distinguir si un dato proviene del conjunto real o ha sido generado artificialmente, y un *generador* $\mathcal{G}(z)$, que toma como entrada una muestra de ruido $z \sim p_z(z)$ y produce datos sintéticos con el objetivo de engañar al discriminador.

El objetivo del generador es producir muestras tan realistas que el discriminador las clasifique como verdaderas. Este objetivo puede expresarse como:

$$\min_{\mathcal{G}} V(\mathcal{G}) = \mathbb{E}_{z\sim p_z(z)} \left[\ln\left(1 - \mathcal{D}(\mathcal{G}(z))\right)\right] \tag{2.8}$$

Por otro lado, el discriminador debe ser entrenado para distinguir correctamente entre datos reales y sintéticos, maximizando la probabilidad de clasificar correctamente ambas clases:

$$\max_{\mathcal{D}} V(\mathcal{D}) = \mathbb{E}_{x\sim p_{\text{real}}(x)} \left[\ln(\mathcal{D}(x))\right] + \mathbb{E}_{z\sim p_z(z)} \left[\ln(1 - \mathcal{D}(\mathcal{G}(z)))\right] \tag{2.9}$$

La formulación conjunta de este juego de optimización se representa mediante la siguiente expresión *minmax*:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{G}, \mathcal{D}) = \mathbb{E}_{x\sim p_{\text{real}}(x)} \left[\ln(\mathcal{D}(x))\right] + \mathbb{E}_{z\sim p_z(z)} \left[\ln(1 - \mathcal{D}(\mathcal{G}(z)))\right] \tag{2.10}$$

El proceso de entrenamiento de una GAN se lleva a cabo en dos fases principales que se repiten de forma alterna: una dedicada al entrenamiento del discriminador y otra al entrenamiento del generador.

En la primera fase, se entrena al discriminador utilizando un conjunto de observaciones reales y otro de imágenes falsas generadas por el generador. Esta etapa se plantea como un problema de clasificación supervisada, en el que las imágenes reales se etiquetan con un 1 y las sintéticas con un 0.

Una vez que el discriminador ha sido entrenado, se procede a optimizar el generador. En esta segunda fase, se congelan los pesos del discriminador y se entrena únicamente el generador. Aunque se sigue utilizando el discriminador como parte del proceso, ahora el objetivo es diferente: las imágenes generadas por el generador se etiquetan con un 1, de

manera que el generador aprende a producir ejemplos que el discriminador clasifique como reales.

Estas dos fases se alternan durante múltiples iteraciones, lo que permite que tanto el generador como el discriminador mejoren progresivamente. El resultado esperado es que el generador llegue a producir datos tan realistas que el discriminador tenga dificultades para diferenciarlos de las observaciones originales. Este proceso de entrenamiento se ilustra en la Figura 2.5.

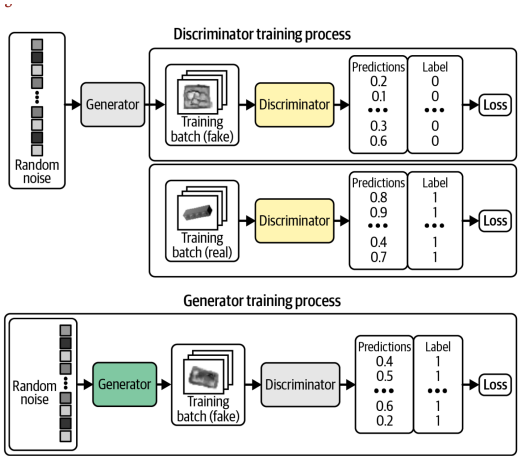


Ilustración 2.5: Proceso de entrenamiento de una red GAN [24]

2.1.3. Transformers

Hablar del estado del arte en Inteligencia Artificial resulta inconcebible sin mencionar *Attention is All You Need* [71], el trabajo seminal de Vaswani et al. que introdujo la arquitectura **Transformer**, basada en el mecanismo de **atención** (*attention*).

Originalmente concebida para el procesamiento de lenguaje natural (NLP), la arquitectura **Transformer** ha demostrado una capacidad de generalización sobresaliente, siendo adaptada con gran éxito a múltiples dominios más allá del texto.

En visión por computador, destacan variantes como **Vision Transformer (ViT)** [17], que reformula tareas de clasificación de imágenes en términos de atención, y **Diffusion Transformer (DiT)** [52], que sustituye redes convolucionales tradicionales por módulos Transformer en el contexto de modelos de difusión. Un ejemplo notable es **Stable Diffusion 3**, que emplea un **Multimodal Diffusion Transformer (MMDiT)** [20] como *backbone*, desplazando a la clásica U-Net [59]. Este cambio será tratado con mayor detalle en la sección dedicada a la evolución de los LDMs (2.2).

Para comprender el Transformer, es necesario entender primero el mecanismo de atención. En tareas de NLP, el significado de una palabra depende críticamente de su contexto léxico y posicional. Métodos previos como LSTM [33] intentaban modelar esta dependencia de forma secuencial mediante redes neuronales recurrentes, pero presentaban limitaciones de

escalabilidad. El mecanismo de atención, en cambio, permite modelar estas relaciones de forma completamente paralela y más eficiente.

El primer paso consiste en la tokenización del texto, es decir, dividir la frase en unidades básicas llamadas *tokens*. Estos tokens se transforman en vectores de dimensión fija n, llamados *embeddings*. Estos *embeddings* se pueden crear mediante modelos como *Word2Vec* [49] o *GloVe* [53]. Así, una frase de m tokens se representa como una matriz $T \in \mathbb{R}^{m \times n}$. A esta representación se le suman vectores de *embeddings* posicionales para conservar información sobre el orden. Esto se hace ya que, por definición, el mecanismo de atención es invariante ante permutaciones.

A partir de T, se generan tres matrices: **Query** Q, **Key** K y **Value** V, comúnmente obtenidas mediante proyecciones lineales aprendibles: $Q = TW^Q$, $K = TW^K$, $V = TW^V$. El mecanismo de atención permite generar a partir de cada *embedding* una nueva representación obtenida mediante la combinación lineal de los *embeddings* del resto de palabras. Los pesos de dicha combinación lineal se pueden obtener a partir de productos escalares entre Q y K, normalizados con *softmax*, que vendrían a representar la distancia entre cada uno de los *embeddings* y el significado que un *token* aporta a otro *token*:

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.11}$$

El factor de escalado $\sqrt{d_k}$ se introduce para evitar que los productos escalares crezcan excesivamente, lo cual podría saturar la función *softmax* y dificultar el entrenamiento.

Una extensión fundamental es la **atención multi-cabeza** (*Multi-Head Attention*), donde se calculan varias atenciones independientes en subespacios distintos de los *embeddings*. Capturando así distintos tipos de relaciones semánticas. Cada cabeza realiza su propia proyección, y las salidas se concatenan y proyectan nuevamente:

$$\begin{aligned} &\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, \ldots, head_h)W^O \\ &\text{donde } head_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2.12}$$

Además, cuando el modelo debe integrar información de distintas secuencias se utiliza **cross attention**, donde Q, K y V provienen de fuentes diferentes.

La arquitectura completa del Transformer se muestra en la figura 2.6. Consta de un *encoder*, que procesa la secuencia de entrada, y un *decoder*, que genera la salida incorporando tanto el contexto previo generado como la información codificada por el *encoder* mediante atención cruzada.

La generalización de esta arquitectura al dominio visual requiere únicamente redefinir el concepto de *token*. Mientras que en NLP un token representa una palabra o subpalabra, en visión por computador se segmenta la imagen en bloques de tamaño fijo. Por ejemplo, una imagen de 256×256 píxeles puede dividirse en 16×16 bloques, produciendo 256 *tokens visuales*. Cada bloque se proyecta a un vector de *embedding*, al que también se añade un *embedding*

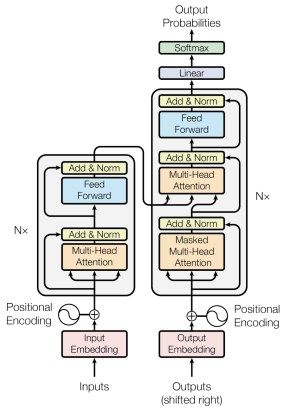


Ilustración 2.6: Arquitectura del *Transformer* [71]

posicional. Este enfoque es el núcleo de **Vision Transformer (ViT)**, cuya arquitectura se muestra en la figura 2.7.

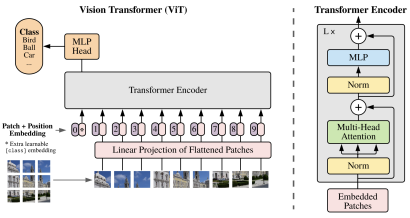


Ilustración 2.7: Arquitectura de Vision Transformer (ViT) [17]

2.1.4. Modelos de difusión (DDPMs, LDMs, DiTs)

2.1.4.1. Denosing Diffusion Probabilistic Models (DDPMs)

Los modelos de difusión son el eje central de este trabajo y, al mismo tiempo, el núcleo del estado del arte en la síntesis de imágenes y video. Estos fueron propuestos por primera vez en el artículo *Deep Unsupervised Learning using Nonequilibrium Thermodynamics* [67] y posteriormente popularizados por Jonathan Ho, Ajay Jain y Pieter Abbeel en *Denoising Diffusion Probabilistic Models* [31] (DDPM).

En términos generales, consisten en dos fases: una primera, denominada *forward process*, en la que se añade ruido gaussiano de manera iterativa a las observaciones originales en un número de pasos (*steps*) t; y una segunda, en la que un modelo aprende a revertir ese proceso eliminando dicho ruido *denoising process*. Una vez entrenado, el modelo es capaz de transformar muestras de ruido gaussiano en imágenes nítidas y coherentes con los datos originales de forma iterativa en un número de *steps*.

El proceso de difusión puede modelarse como una cadena de Markov. Dada una imagen x_{t-1}, se genera una versión más ruidosa x_t según la siguiente distribución:

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}\, x_{t-1}, \beta_t I) \tag{2.13}$$

Aquí, β_t es la varianza, es decir, controla la magnitud del ruido añadido en el paso t. Es recomendable que este valor varíe con el tiempo: en las primeras etapas, cuando la imagen aún conserva su estructura, se añade poco ruido, mientras que en etapas posteriores, cuando la imagen ya está degradada, se puede aumentar la magnitud del ruido. Esta secuencia de valores se conoce como *schedule*. En [31] se utiliza un esquema lineal desde $\beta_1 = 10^{-4}$ hasta $\beta_T = 0{,}02$ con $T = 1000$ pasos.

Si bien la ecuación anterior describe cómo obtener una imagen más ruidosa a partir de una menos ruidosa, resulta útil expresar directamente x_t en función de la imagen original x_0 y del paso t. Para ello, se definen:

$$\alpha_t = 1 - \beta_t, \quad \bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i \tag{2.14}$$

Con esta notación, se puede escribir:

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}\, x_0, (1-\bar{\alpha}_t) I) \tag{2.15}$$

Esta forma evita la necesidad de calcular recursivamente todos los pasos intermedios, y bastaría con muestrear la distribución anterior. Este proceso se puede observar en la figura 2.8.

Al igual que con los VAEs 2.1.1 utilizamos el truco de reparametrización para el muestreo:

$$x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon, \quad \epsilon \sim \mathcal{N}(0, I) \tag{2.16}$$

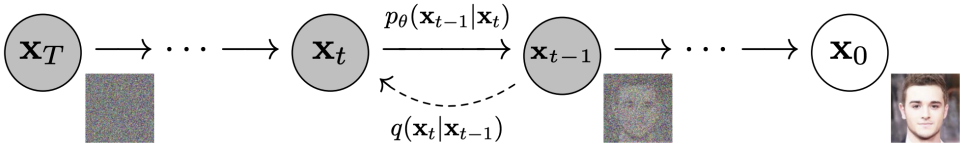


Ilustración 2.8: Proceso de difusión progresiva (forward process) [31]

Es importante denotar que de la misma expresión 2.16, despejando x_0, se puede extraer una ecuación que nos permita, dada la imagen degradada y el ruido añadido, recuperar la imagen original como se observa en 2.17. Esto es importante ya que se usará más tarde para hacer inferencia sobre el modelo.

$$x_0 = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(x_t - \frac{1-\bar{\alpha}_t}{\sqrt{1-\bar{\alpha}_t}} \epsilon \right) \tag{2.17}$$

Durante el entrenamiento, el objetivo es enseñar a una red neuronal $\epsilon_\theta(x_t, t)$ a predecir el ruido ϵ presente en una muestra x_t dada la información del paso t muestreado aleatoriamente. Esto se realiza minimizando el error cuadrático entre el ruido real y el predicho por el modelo $\epsilon_\theta(x_t, t)$, como se detalla en el algoritmo 1.

Algoritmo 1 Proceso de entrenamiento de un DDPM [31]

1: **repeat**
2: Muestrear $x_0 \sim q(x_0)$
3: Muestrear $t \sim \text{Uniform}(\{1, \dots, T\})$
4: Muestrear $\epsilon \sim \mathcal{N}(0, I)$
5: Calcular $x_t = \sqrt{\bar{\alpha}_t}\, x_0 + \sqrt{1 - \bar{\alpha}_t}\, \epsilon$
6: Actualizar los parámetros minimizando:

$$\nabla_\theta \left\| \epsilon - \epsilon_\theta(x_t, t) \right\|^2$$

7: **until** convergencia

Durante la inferencia, el objetivo es dado una muestra de ruido gaussiano ir eliminando dicho ruido hasta obtener una imagen nítida. Para ello se comienza en el instante $t = T$, siendo T el número de *steps* de eliminación de ruido que va a realizar nuestro modelo, hasta llegar a $t = 1$. Durante cada uno de estos pasos se intentará predecir el ruido en dicho instante utilizando nuestra red $\epsilon_\theta(x_t, t)$ y se eliminará de la imagen x_t utilizando la expresión 2.17. Este proceso es detallado en el algoritmo 2.

Algoritmo 2 Proceso de inferencia de un DDPM [31]

1: Muestrear ruido $\mathbf{x}_T \sim \mathcal{N}(0, I)$
2: **for** $t = T, ..., 1$ **do**
3: Muestrear $z \sim \mathcal{N}(0, I)$ if $t > 1$, else $\mathbf{z} = 0$
4: Reducir ruido calculando $\mathbf{x}_{t-1}$

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

5: **end for**
6: **return** $\mathbf{x}_0$

Para la tarea de predicción del ruido, se emplea habitualmente una arquitectura de tipo U-Net como la observada en la figura 2.9. Aunque originalmente fue diseñada para segmentación de imágenes biomédicas [59], ha demostrado ser muy eficaz en el proceso de eliminación de ruido por su capacidad excepcional de extracción de características, su entendimiento del contexto, su segmentación precisa y la propiedad que tiene de preservar la dimensionalidad [78]. Para hacer alusión a la red de predicción de ruido se utiliza comúnmente la palabra *backbone* ya que constituye el eje central del modelo.

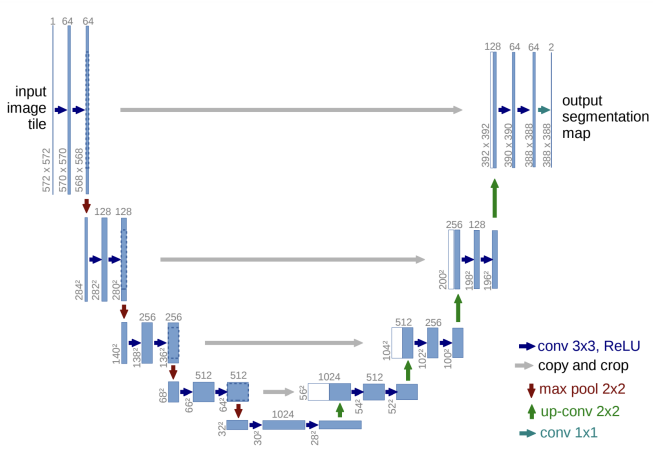


Ilustración 2.9: Arquitectura de red U-Net utilizada para predecir el ruido [59]

2.1.4.2. Modelos de difusión Latente (LDMs)

Uno de los mayores retos de los modelos de difusión previamente estudiados es el alto coste computacional que requiere su entrenamiento para la síntesis de imágenes en alta resolución. Esto se debe a que dichos modelos operan en el espacio de píxeles, lo cual implica una gran dimensionalidad, provocando una elevada demanda computacional y energética.

Para abordar esta limitación, Rombach et al., en su artículo *High-Resolution Image Synthesis with Latent Diffusion Models* [58], proponen los LDMs, por sus siglas en inglés *Latent Diffusion Models*.

La propuesta consiste en utilizar un autoencoder antes del proceso de difusión, con el fin de comprimir la imagen original en un vector del espacio latente de menor dimensionalidad, mediante un encoder $\mathcal{E}$. Posteriormente, este vector se emplea durante el proceso de difusión, añadiéndole ruido gaussiano en cada paso. Se entrena entonces un modelo para predecir dicho ruido, y finalmente se decodifica el vector latente utilizando un decoder $\mathcal{D}$. Esto da lugar a una función de pérdida como la que se muestra en la ecuación 2.18.

Si observamos el algoritmo de entrenamiento expuesto en el capítulo anterior (véase la Figura 1), la expresión es esencialmente la misma, con la diferencia de que en lugar de usar x_t como entrada para la red ϵ_θ, se utiliza z_t, que no es más que el vector latente $z_0 = \mathcal{E}(x_0)$ con el ruido correspondiente añadido.

$$\mathcal{L}_{LDM} := \mathbb{E}_{\mathcal{E}(x),\epsilon\sim\mathcal{N}(0,1),t}\left[\|\epsilon - \epsilon_\theta(z_t, t)\|^2\right] \tag{2.18}$$

En la figura 2.10 se observa la arquitectura del modelo, en la que se distinguen claramente el autoencoder y la U-Net encargada de predecir el ruido.

Si observamos el diagrama en la Figura 2.10, podemos notar la presencia de un módulo de condicionamiento. Aunque este será tratado con mayor detalle en la siguiente sección, este consiste en imponer ciertas condiciones iniciales que la imagen generada debe cumplir,

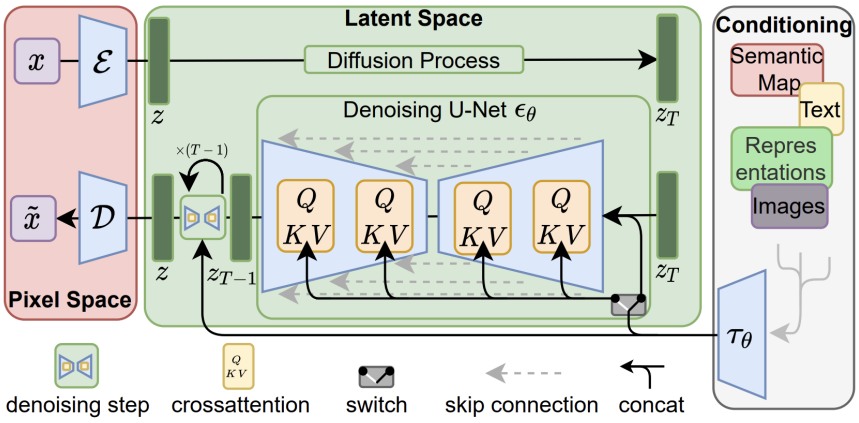


Ilustración 2.10: Diagrama de la arquitectura de un modelo de difusión latente [58]

denotadas como y. Para ello, se utiliza un *encoder* $\tau_\theta(y)$ que transforma estas condiciones en *embeddings* adecuados. El condicionamiento más popular es el textual, la gran mayoría de modelos de base son entrenados para hacer síntesis de imágenes a partir de texto *text-to-image*.

Dichos *embeddings* se integran en la U-Net mediante capas de atención cruzada, las cuales permiten fusionar la información de los mapas de características de la U-Net con la información del condicionamiento. Esto se expresa formalmente en la Ecuación 2.19, donde $\phi_i(z_t)$ representa un mapa de características intermedio de la U-Net aplanado.

$$Q = W_Q^{(i)} \cdot \phi_i(z_t), K = W_K^{(i)} \cdot \tau_\theta(y), V = W_V^{(i)} \cdot \tau_\theta(y) \tag{2.19}$$

El estudio y la interpretación de los mapas de atención cruzada sigue siendo un área activa de investigación en el estado del arte. Sin embargo, se ha demostrado que estos mapas permiten a los modelos desarrollar una comprensión profunda de atributos clave como la forma, el color, la profundidad y otros aspectos relevantes de los objetos generados [76].

Si visualizamos estos mapas, por ejemplo, promediándolos y reescalándolos a resoluciones comunes como 64×64, 32×32, 16×16 o 8×8 (que corresponden a las dimensiones de los mapas de características de la U-Net), podemos observar cómo el modelo identifica las regiones de la imagen donde se encuentran los diferentes objetos, como se muestra en la Figura 2.11. Esta capacidad se está utilizando en el estado del arte para generar etiquetas en datos sintéticos, constituyendo una técnica de aumento de datos [74][73]. Más adelante, se discutirán estos y otros métodos de aumento de datos basados en modelos de difusión.

2.1.4.3. Generación condicionada

Los LDMs como hemos visto anteriormente muestran capacidades excepcionales para la síntesis de imágenes, sin embargo existen muchos casos donde es deseable establecer unas

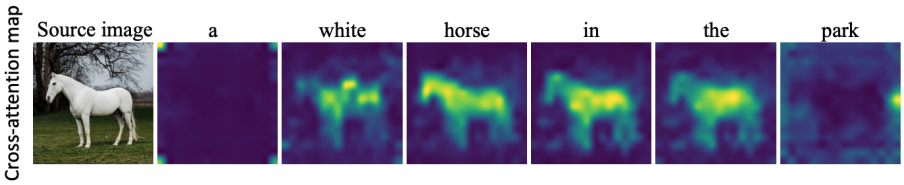


Ilustración 2.11: Visualización de los mapas de atención cruzada de Stable Diffusion según *token* de entrada textual [76]

condiciones iniciales en orden de controlar el resultado final. Esto es conocido como síntesis condicionada de imágenes y existen muchas tareas de síntesis según el dominio al que pertenece la condición inicial como se ilustra en la figura 2.12.

De una manera mas formal dada una distribución p_{model} queremos muestrear la misma de forma condicionada por un valor y que pertenece al dominio de una tarea en concreto (texto, audio, pose etc.) $y \in \mathcal{D}_{\mathcal{L}}$.

$$x \sim p_{model}(x|y), y \in \mathcal{D}_{\mathcal{L}} \tag{2.20}$$

Las tareas disponibles de síntesis condicionada se pueden agrupar en siete categorías [78] :

1. **Síntesis de texto a imagen** (*text-to-image*): genera imágenes de acuerdo con instrucciones textuales.
2. **Restauración de imágenes** (*image restoration*): recupera imágenes limpias a partir de sus versiones degradadas.
3. **Conversión de señal visual a imagen** (*visual signal to image*): transforma señales visuales como bocetos, mapas de profundidad o poses humanas en imágenes correspondientes.
4. **Edición de imágenes** (*image editing*): modifica imágenes originales con información semántica, estructural o de estilo proporcionada.
5. **Personalización** (*customization*): crea diferentes versiones editadas de un objeto personal especificado mediante imágenes dadas.
6. **Composición de imágenes** (*image composition*): combina objetos y fondos especificados en distintas imágenes en una sola imagen.
7. **Control de diseño** (*layout control*): gestiona la disposición espacial de los objetos en imágenes sintetizadas utilizando información espacial proporcionada del primer plano y del fondo.

El estado del arte nos ofrece múltiples formas de entrenar a un modelo de difusión para realizar estas tareas como se observa en la figura 2.13. Este entrenamiento se puede realizar en diferentes etapas:

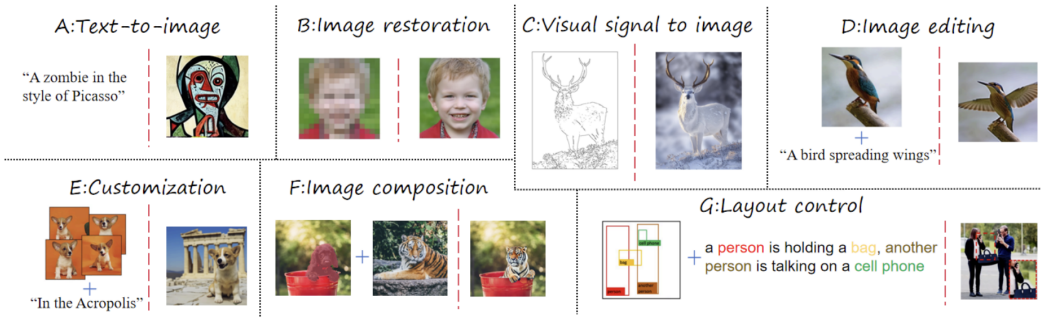


Ilustración 2.12: Representación de las diferentes tareas de síntesis condicionada de imágenes de forma visual [78].

1. **Etapa de entrenamiento**: Esta forma consiste en entrenar un modelo de difusión de cero con un Dataset que contiene emparejadas las imágenes y su condicionamiento. Esta es la técnica más intensa en computación.
2. **Etapa de re-adaptación**: Consiste en un modelo entrenado para una tarea diferente, normalmente se usa el *backbone* de un modelo imagen a texto, hacerle un *fine tuning* para que sea capaz de realizar una tarea diferente. Esta opción permite reducir los recursos computacionales necesarios. El ejemplo más interesante según nuestro criterio es **ControlNet** 2.23.
3. **Etapa de especialización:** Consiste en, dado un modelo que acepte condicionamiento, imagen a texto normalmente, entrenar un encoder que proyecte otro tipo de condicionamiento al espacio aceptado por el modelo original. Es la técnica más eficiente pues no se re-entrena el *backbone*. Los ejemplos más interesantes son **LoRa** [34] y **Text Inversion** [25]

2.1.4.4. Diffusion Transformers (DiT)

El **Diffusion Transformer (DiT)** fue propuesto en 2023 por Peebles et al. [52]. En este trabajo se introduce un modelo de difusión latente (LDM) que reemplaza la clásica arquitectura **U-Net** por una arquitectura basada en **Transformers**, siguiendo el diseño del *Vision Transformer (ViT)* [17].

Es importante señalar que el proceso de entrenamiento e inferencia de la difusión permanece esencialmente igual: se parte de una representación latente contaminada con ruido, y el modelo aprende a predecir dicho ruido progresivamente en cada paso del proceso inverso. La diferencia fundamental reside en el tipo de red neuronal utilizada como *backbone* para realizar esta predicción: en lugar de una red convolucional como la U-Net, DiT emplea un Transformer operando sobre tokens visuales.

Para que un Transformer pueda operar sobre imágenes, estas deben transformarse primero en una secuencia de tokens. Esto se consigue dividiendo la imagen en pequeños parches de tamaño fijo (por ejemplo, 16×16 píxeles), aplanando cada uno de ellos y proyectándolos a

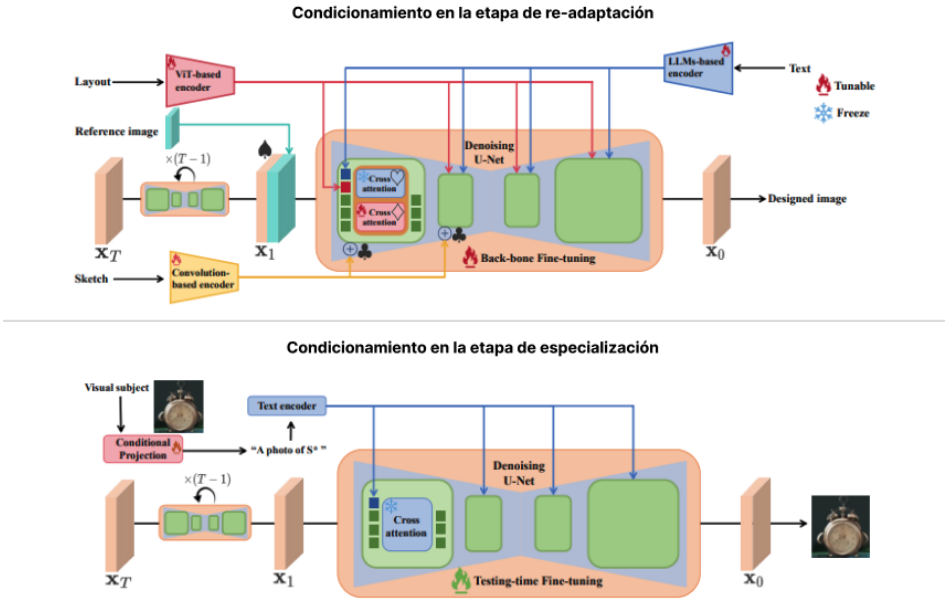


Ilustración 2.13: Figura comparativa de la adaptación a un condicionamiento específico en la etapa de re-adaptación y de especialización, composición de dos figuras extraídas de [78]

un espacio vectorial, como explicamos anteriormente con el ViT.

El condicionamiento del modelo puede incorporarse de varias formas. Dos estrategias comunes consisten en concatenar esta información a los embeddings de la imagen, o bien utilizar mecanismos de *cross-attention* para integrar directamente los embeddings del condicionamiento con los de la imagen.

La arquitectura general del DiT puede observarse en la figura 2.14.

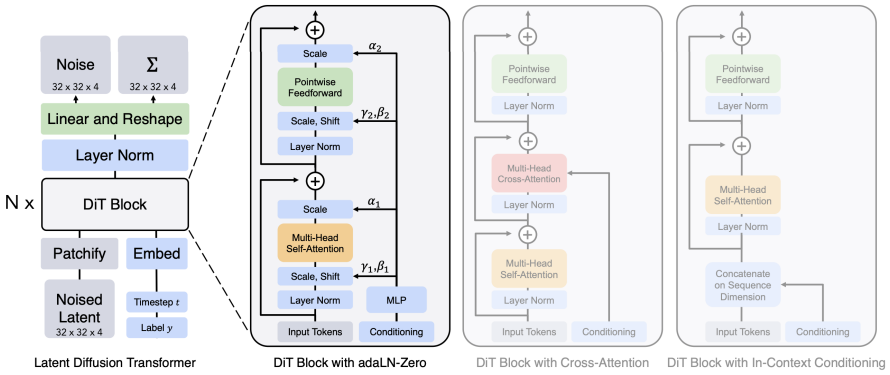


Ilustración 2.14: Arquitectura del Diffusion Transformer (DiT) [52]

Los últimos modelos de *Stable Diffusion* emplean *Transformer* como base para sus *backbones*. Es por ello que conocer esta arquitectura resulta imprescindible para comprender el estado del arte, aunque como veremos próximamente dichas redes sean bastante más com-

plejas que la mencionada aquí.

2.1.5. Métricas para la evaluación de modelos generativos

2.1.5.1. Fréchet Inception Distance (FID)

El **Fréchet Inception Distance** (FID) es una métrica ampliamente utilizada para evaluar la calidad de imágenes generadas por modelos generativos, especialmente en el contexto de generación de imágenes sintéticas. FID compara la distribución estadística de características extraídas de imágenes reales y generadas utilizando una red neuronal preentrenada, típicamente Inception v3.

Formalmente, se calcula asumiendo que las características extraídas de ambas distribuciones (reales y generadas) siguen una distribución gaussiana multivariable con medias μ_r, μ_g y matrices de covarianza Σ_r, Σ_g, respectivamente. La distancia FID se define como la distancia de Fréchet entre estas dos distribuciones:

$$\text{FID} = \|\mu_r - \mu_g\|_2^2 + \text{Tr}\left(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{\frac{1}{2}}\right) \tag{2.21}$$

donde $\|\cdot\|_2$ es la norma Euclidiana y $\text{Tr}(\cdot)$ es la traza de una matriz.

Valores bajos de FID indican que las distribuciones de características de las imágenes generadas y reales son similares, lo que implica una alta calidad y diversidad en las imágenes sintéticas. En cambio, valores altos reflejan una mayor discrepancia y, por tanto, menor calidad.

2.1.5.2. CLIP Score

El **CLIP Score** es una métrica utilizada para evaluar la correspondencia entre imágenes generadas y descripciones textuales, basada en el modelo CLIP (Contrastive Language–Image Pre-training) desarrollado por OpenAI.

CLIP consiste en dos encoders: uno para texto y otro para imágenes, entrenados conjuntamente para proyectar ambos tipos de datos a un espacio latente compartido donde se mide la similitud.

Dado un texto t y una imagen generada x, se obtienen sus vectores latentes normalizados $\mathbf{v}_t$ y $\mathbf{v}_x$, respectivamente. El CLIP Score se calcula como el coseno de su ángulo, es decir:

$$\text{CLIP Score} = \cos(\mathbf{v}_t, \mathbf{v}_x) = \frac{\mathbf{v}_t \cdot \mathbf{v}_x}{\|\mathbf{v}_t\|\|\mathbf{v}_x\|}. \tag{2.22}$$

Este valor oscila entre -1 y 1, donde valores cercanos a 1 indican una alta similitud semántica entre la imagen y el texto, es decir, que la imagen generada corresponde bien a la descripción textual.

El CLIP Score es especialmente útil para evaluar modelos de generación de imágenes condicionadas por texto, complementando métricas como FID que solo miden calidad visual sin considerar la coherencia semántica.

2.2. Historia y evolución de los modelos de difusión Open Source

Si bien anteriormente hemos abordado las bases teóricas que sustentan el estado del arte en modelos generativos, ahora nos centraremos en cómo dichas bases han evolucionado hasta alcanzar los modelos de difusión hiperrealistas que dominan actualmente la síntesis de imágenes.

Aunque mencionaremos algunos de los modelos más relevantes en esta evolución, nos centraremos principalmente en aquellos que sean open source o cuya investigación haya sido publicada de forma accesible. Es importante destacar que modelos propietarios como Midjourney, DALL-E (OpenAI), Imagen (Google) o Sora (OpenAI) —aunque han tenido un impacto considerable en la historia reciente de los modelos generativos— serán omitidos, ya que no cuentan con artículos científicos, acceso libre a sus pesos, ni arquitecturas totalmente abiertas al público.

Bajo este criterio, Stable Diffusion se posiciona como el principal objeto de estudio. No solo por haber sido desarrollado a partir de los primeros trabajos sobre Latent Diffusion Models (LDMs), sino también porque ofrece un ecosistema completamente abierto: artículos científicos detallados, *datasets* públicos, arquitecturas reproducibles y pesos descargables. Además, la enorme comunidad que se ha formado a su alrededor ha impulsado el desarrollo de numerosos *fine-tunes*, extensiones y optimizaciones, enriqueciendo aún más su impacto y evolución.

2.2.1. Stable Diffusion

Latent Diffusion

Aunque no existe una versión oficial 1.0 de Stable Diffusion, se considera como tal el primer modelo publicado en diciembre de 2021 por CompVis [58], disponible en su repositorio de GitHub [10].

Este modelo fue entrenado tanto sin condicionamiento como con condicionamiento, usando *datasets* como CelebA-HQ [36], FFHQ [37], LSUN [75] para el entrenamiento no condicionado, y LAION-400M [65], MS COCO [43], OpenImages [40] y COCO [7] para tareas condicionadas (texto a imagen, segmentación, superresolución, inpainting).

Stable Diffusion 1.x

En agosto de 2022, CompVis lanzó oficialmente las versiones 1.1 a 1.4 de Stable Diffusion [11], enfocadas en generación de imágenes a partir de texto con *datasets* LAION-5B y varian-

tes, resoluciones 256 × 256 y 512 × 512, y el modelo *CLIP ViT-L/14* [56] para codificación textual.

Las diferencias entre versiones fueron principalmente el número de pasos de entrenamiento, subconjuntos del dataset y la técnica de *Classifier-free guidance* aplicada en 1.3 y 1.4 (eliminación del condicionamiento textual en un 10 % de los ejemplos), manteniendo la misma arquitectura.

Stable Diffusion 1.5 y 2.x

En octubre de 2022, RunwayML lanzó Stable Diffusion 1.5, un *fine-tune* de la versión 1.2. Posteriormente, StabilityAI lanzó las versiones 2.0 y 2.1 [StabilityAI], que introdujeron mejoras como generación a 768 × 768 píxeles, uso de OpenCLIP-ViT/H [mlfoundations] y mejor tiempo de inferencia [61].

SDXL

En julio de 2023 se lanzó SDXL [55], que presenta un backbone con 2.6B parámetros y dos encoders de texto cuyas salidas se concatenan para mejorar el condicionamiento textual.

Además, incorpora un *pipeline* de dos pasos con un modelo base y un refinador inspirado en eDiff-I [5] y SEdit [48], mejorando la calidad local y el alineamiento con el texto. Esto se puede observar en la figura 2.15.

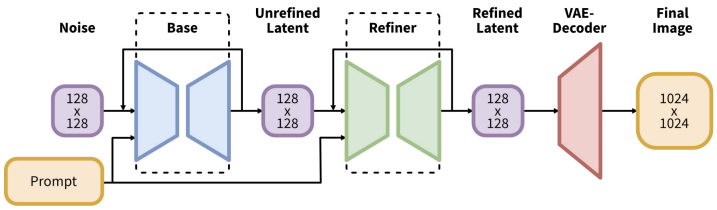


Ilustración 2.15: Diagrama de la arquitectura SDXL [55]

SDXL Turbo

En noviembre de 2024, se lanzó SDXL Turbo, una versión destilada que usa *Adversarial Diffusion Distillation* (ADD) [64] para reducir a tan solo 4 pasos la inferencia con calidad similar a SDXL.

ADD combina entrenamiento adversarial (modelo estudiante vs discriminador) y distilación (imitación de un modelo profesor), usando un dataset de imágenes reales y los modelos estudiante y profesor. El procedimiento incluye *forward diffusion*, reconstrucción, evaluación adversarial y cálculo de pérdida de destilación. El funcionamiento de esta técnica se puede observar en la figura 2.16.

Stable Cascade

En febrero de 2024 se presentó Stable Cascade (SDC), basado en la arquitectura Würstchen propuesta por Pablo Pernías et al. [54]. SDC opera sobre un espacio latente ultra

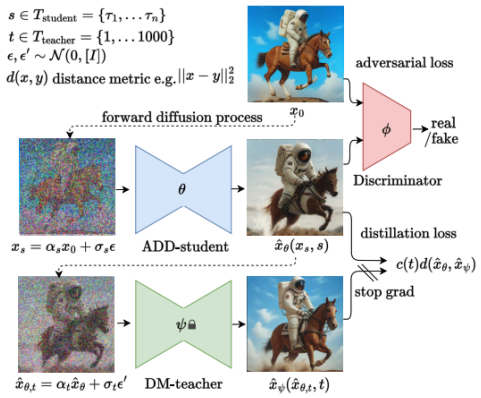


Ilustración 2.16: Funcionamiento de la técnica ADD [64].

reducido (compresión $42:1$) con tres módulos: un decodificador VQGAN y dos modelos de difusión latente f_θ y f_v. Esto se observa en la figura 2.17.

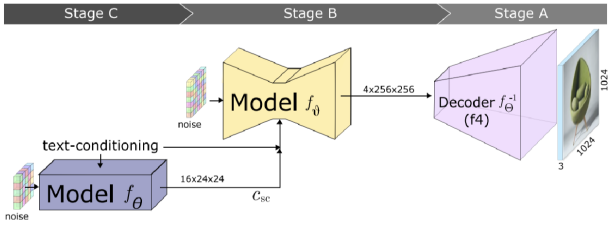


Ilustración 2.17: Fase de inferencia de Stable Cascade [54].

Gracias a esta estrategia, Stable Cascade reduce el coste computacional de entrenamiento aproximadamente **ocho veces** respecto a modelos como Stable Diffusion 2.1.

Stable Diffusion 3.X

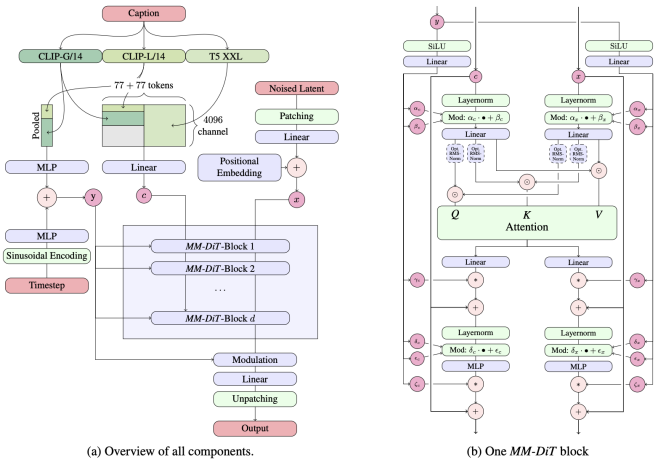
Publicado en febrero de 2024 [20], Stable Diffusion 3 incorpora el *Multimodal Diffusion Transformer (MMDiT)*, basado en DiT [52].

Además, adopta **flujos rectificados** [44] como formalismo para el mecanismo de difusión, que aprenden una trayectoria determinista entre distribuciones mediante una EDO, facilitando inferencia más eficiente.

La Figura 2.18 muestra su arquitectura. Similar a SDXL, SD3 cuenta con una versión destilada usando LADD (Latent Adversarial Diffusion Distillation) [63].

2.2.2. Flux 1.1

Flux 1.1 [2] es un modelo de pesos abiertos desarrollado por Blackforest Labs a finales de 2024, un laboratorio de investigación fundado ese mismo año por ex-miembros de StabilityAI. Aunque se trata de una entidad privada y su investigación no es completamente abierta, Flux



(a) Overview of all components. (b) One *MM-DiT* block

Ilustración 2.18: Arquitectura Stable Diffusion 3 [20]

1.1 se posiciona entre los modelos con mejor rendimiento del mercado, motivo por el cual ha sido utilizado a lo largo de esta investigación.

A pesar de que su arquitectura completa no ha sido publicada, se han revelado algunos detalles relevantes en su anuncio de lanzamiento. En particular, se ha indicado que utiliza un *backbone* basado en **Transformers** [52] y que implementa **flujos rectificados** [44], lo que sugiere una arquitectura cercana a la de Stable Diffusion 3. También se ha confirmado que el modelo cuenta con 12 mil millones de parámetros y emplea optimizaciones modernas como embeddings posicionales rotatorios (*RoPE*) [69] y mecanismos de atención paralelizados [15].

Flux 1.1 está disponible en tres variantes:

1. **FLUX.1 [pro]**: Versión más potente, accesible únicamente mediante API o soluciones empresariales personalizadas.
2. **FLUX.1 [dev]**: Variante de pesos abiertos para uso no comercial, destilada directamente de la versión *pro*, con calidad comparable y mayor eficiencia.
3. **FLUX.1 [schnell]**: Versión optimizada para desarrollo local y uso personal, distribuida bajo licencia Apache 2.0.

2.3. Entrenamiento y condicionamiento de modelos de difusión latente

2.3.1. ControlNET

ControlNet [Zhang et al.], mencionado anteriormente, es un método diseñado para adaptar modelos de difusión preentrenados a nuevas formas de condicionamiento. Para ello, se introduce una red paralela a los bloques de la arquitectura original. Concretamente, se con-

gela el bloque original $\mathcal{F}(x;\Theta)$, y se añade una copia entrenable que recibe como entrada la suma del mapa original x y un nuevo condicionamiento c, este último procesado primero mediante una convolución 1×1. La salida de esta rama paralela se somete a otra convolución 1×1 y se suma a la salida original, tal como se expresa en la Ecuación 2.23. En su implementación en *Stable Diffusion*, se clona el encoder de la U-Net replicando sus pesos y se inicializan las convoluciones en cero. De esta forma el comportamiento del modelo original no se verá alterado por nuestra red paralela al principio del entrenamiento. Un diagrama del funcionamiento de esta técnica puede observarse en la figura 2.19.

$$y = \mathcal{F}(x;\Theta) + \mathcal{Z}(\mathcal{F}(x + \mathcal{Z}(c;\Theta_{z1});\Theta_c);\Theta_{z2}) \tag{2.23}$$

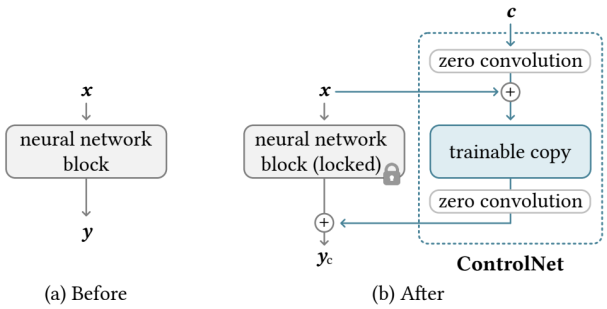


Ilustración 2.19: Diagrama de la aplicación de ControlNet a una red neuronal genérica [Zhang et al.].

2.3.2. LoRA

Low-rank adaptation (LoRA) [34] es una técnica para realizar *fine-tuning* eficiente en modelos grandes, optimizando solo un pequeño subconjunto de parámetros. Aunque inicialmente creada para NLP, su uso se ha extendido a modelos de difusión debido a su bajo coste computacional y de almacenamiento.

Consiste en modificar los pesos de una capa $W_0 \in \mathbb{R}^{d\times k}$ añadiendo una matriz de bajo rango $\Delta W = BA$, con $B \in \mathbb{R}^{d\times r}$ y $A \in \mathbb{R}^{r\times k}$, donde $r \ll \text{mín}(d,k)$. Durante el entrenamiento, W_0 se mantiene fijo y solo se optimizan A y B. El *forward pass* se puede definir de la siguiente forma:

$$h = x(W_0 + \Delta W) = xW_0 + xBA. \tag{2.24}$$

Para evitar alterar inicialmente el comportamiento del modelo, A se inicializa con ruido gaussiano y B con una matriz de ceros, garantizando que la salida sea equivalente al modelo original al comienzo del *fine-tune*. LoRA está pensado para entrenar el modelo a aprender nuevos conceptos (personas, objetos, escenarios inéditos), y no para nuevas tareas de condicionamiento. Sin embargo, puede combinarse con métodos como InstructPix2Pix [6] para

adaptaciones más específicas, como en Flux1.1, que ofrece LoRAs para inpainting, canny y *depth estimation*. Se puede observar un diagrama de LoRA en la figura 2.20.

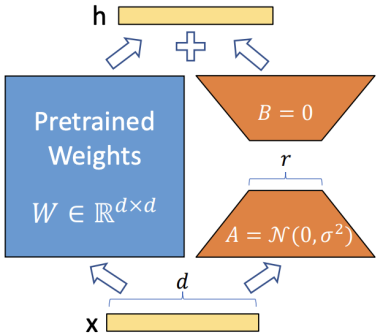


Ilustración 2.20: Diagrama de la técnica LoRA [34].

2.4. Aumento de datos

En el contexto del aprendizaje profundo, la calidad del conjunto de datos es tan importante como la arquitectura del modelo. Un buen conjunto de datos puede hacer que un modelo relativamente simple obtenga resultados sobresalientes, mientras que un conjunto de baja calidad puede inutilizar incluso a los modelos más avanzados, con miles de millones de parámetros. La calidad de los datos, por tanto, puede ser el factor determinante entre el éxito y el fracaso de un sistema.

La calidad depende principalmente de dos aspectos: la naturaleza intrínseca de los datos y su etiquetado. En cuanto a la naturaleza, los problemas más comunes incluyen el exceso de información irrelevante o ruidosa, la escasa representación de ciertas características clave, o las inconsistencias entre diferentes fuentes de datos [47]. En cuanto al etiquetado, es fácil introducir errores —especialmente en *datasets* obtenidos mediante *crowdsourcing*— lo que puede introducir sesgos que perjudican seriamente la capacidad de generalización del modelo [12].

Dado el elevado coste asociado a la obtención de datos etiquetados de calidad, el *data augmentation* se presenta como una estrategia eficaz para mejorar la capacidad de generalización de los modelos sin necesidad de recopilar nuevos datos. Esta sección se centra en su aplicación al ámbito de la visión por computador, diferenciando entre técnicas clásicas y enfoques modernos basados en modelos generativos.

2.4.1. Taxonomía

Siguiendo el formalismo propuesto por Wang et al. [72], el proceso de aumento de datos puede modelarse como una función f_θ parametrizada, que transforma un conjunto de datos etiquetado $\mathcal{D}_L = \{(\mathbf{X}, y)\}$ en un nuevo conjunto aumentado $\tilde{\mathcal{D}}_L = \{(\tilde{\mathbf{X}}, \tilde{y})\}$:

$$f_\theta : \mathcal{D}_L \to \tilde{\mathcal{D}}_L$$

Según el número de muestras empleadas en el proceso de generación, las técnicas de aumento pueden clasificarse en tres grandes grupos:

Aumento individual: transforma una muestra con una perturbación ϵ:

$$\tilde{x} = \lambda_1 x_i + \lambda_2 \epsilon(x_i), \quad \tilde{y} = y_i$$

Puede ser:

1. **Basado en valores:** cambia píxeles (color, brillo, ruido).
2. **Basado en estructuras:** altera geometría (rotación, traslación).

Aumento múltiple: combina varias muestras:

$$\tilde{\mathbf{x}} = \sum \mathbf{x}_i w_i, \quad \tilde{y} = \sum y_i w_i$$

Distinguimos:

1. **Mezcla de valores:** interpolación a nivel de píxeles.
2. **Mezcla estructural:** fusión de regiones completas.

Aumento poblacional: genera datos desde una distribución estimada $P(X)$:

$$\tilde{x} \sim P(X), \quad \tilde{y} \sim P(y|\tilde{x})$$

Puede basarse en:

1. **Distribución interna (vanilla).**
2. **Fuentes externas (exógena).**

2.4.2. Técnicas tradicionales de aumento de datos

2.4.2.1. Eliminación de píxeles

Estas técnicas eliminan regiones específicas de las imágenes de entrada para simular oclusiones y promover el aprendizaje de representaciones robustas. Ejemplos destacados:

1. *Cutout* [16]: aplica máscaras aleatorias para eliminar regiones continuas.
2. *Random Erasing* [81]: emplea máscaras cuadradas con relleno de color aleatorio, y elige aleatoriamente su ubicación y tamaño.

3. *Hide-and-Seek* [82]: divide la imagen en una cuadrícula de parches y oculta aleatoriamente varios de ellos.
4. *GridMask* [9]: utiliza una máscara de cuadrados equiespaciados y distribuidos uniformemente sobre la imagen.

2.4.2.2. Transformaciones fotométricas

Alteran los valores de color de la imagen sin modificar su estructura espacial como se observa en la figura 2.21. Entre las más comunes:

1. **Escala de grises**: Convierte cada píxel del espacio RGB a escala de grises.
2. **Inversión de color**: invierte cada píxel en el espacio RGB.
3. **Color casting**: modifica la intensidad de un canal (R, G o B) manteniendo los demás constantes.
4. **Adición de ruido**: introduce pequeñas variaciones aleatorias (por ejemplo, ruido gaussiano).

Ilustración 2.21: Ejemplo de múltiples transformaciones fotométricas [38]

2.4.2.3. Transformaciones geométricas

Aplican transformaciones afines (traslaciones, rotaciones, volteos) o no afines (perspectiva, *stretching*, distorsiones) como se puede observar en la figura 2.22. Algunos métodos relevantes:

1. **Patch Shuffle** [35]: reorganiza aleatoriamente parches de la imagen.
2. **Patch Reordering** [66]: reordena parches según una heurística determinada.

Ilustración 2.22: Ejemplo de múltiples transformaciones geométricas [38]

2.4.2.4. Recortado de imágenes

Consiste en seleccionar un área de la imagen original y eliminar el resto. Esto reduce el tamaño de almacenamiento y permite generar múltiples ejemplos a partir de una sola muestra. Las versiones más avanzadas utilizan modelos de atención o criterios estéticos para elegir la región recortada [8]. Se puede observar un ejemplo de dicha técnica en la figura 2.23.

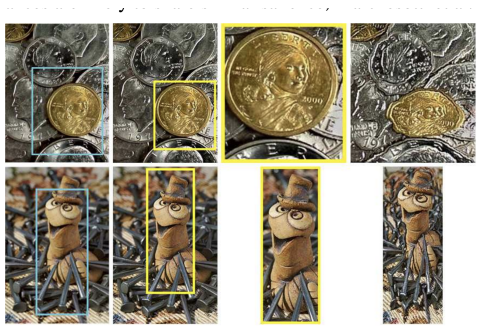


Ilustración 2.23: Ejemplo de recortado de imágenes basado en atención [8]

2.4.2.5. Aumento de datos automático

Automatiza la selección de técnicas de aumento de datos y sus parámetros mediante aprendizaje automático. El trabajo que sienta las bases de esta categoría es *AutoAugment* [13] empleando un controlador basado en aprendizaje por refuerzo. Variantes posteriores mejoran su eficiencia:

1. **Fast AutoAugment** [42]
2. **RandAugment** [14]
3. **Population Based Augmentation (PBA)** [30]

2.4.2.6. Mezcla de imágenes

Estas técnicas combinan imágenes a nivel de píxeles mediante interpolación. Ejemplos clave:

1. **MixUp** [79]: interpola linealmente tanto imágenes como etiquetas.
2. **CutMix** [77]: reemplaza una región rectangular de una imagen con un parche de otra.
3. **AugMix** [28]: aplica múltiples transformaciones clásicas y luego las combina.
4. **AutoMix** [45]: emplea redes neuronales para aprender la combinación óptima, técnica conocida como *neural blending*.

Podemos encontrar una comparativa entre estos métodos en la figura 2.24.

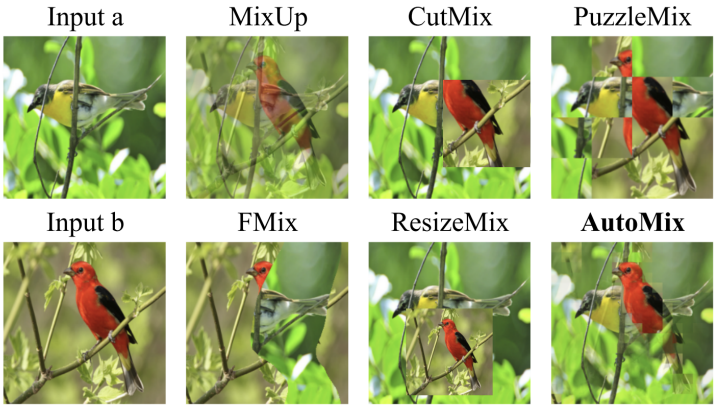


Ilustración 2.24: Comparación de múltiples técnicas de mezcla de imágenes [45]

2.4.3. Aumento de datos mediante modelos de difusión

Los modelos de difusión se han consolidado como una herramienta eficaz para la generación sintética de imágenes, gracias a su capacidad para producir muestras de alta calidad y diversidad. Esto ha impulsado su uso en tareas de aumento de datos, ya sea como complemento a conjuntos reales o como fuente primaria de datos sintéticos.

No obstante, su aplicación práctica conlleva varios desafíos, entre ellos la generación de anotaciones fiables y la definición adecuada de los condicionamientos textuales o *prompts* necesarios para guiar la síntesis.

Con el fin de categorizar los distintos enfoques desarrollados en la literatura, proponemos una taxonomía basada en el **grado de interacción con el modelo de difusión** como se observa en la figura 2.25, que distingue tres niveles:

- ✔ **Caja negra (black-box):** el modelo se emplea como un componente cerrado, sin acceso a sus representaciones internas ni modificaciones estructurales. Los sistemas se construyen a través de *pipelines* externos que pueden incluir modelos de lenguaje, clasificación o segmentación.
- ✔ **Caja gris (gray-box):** se accede a representaciones intermedias del modelo —como los mapas de atención— sin alterar su arquitectura, permitiendo extraer información útil durante el proceso de generación.
- ✔ **Caja blanca (white-box):** el modelo se modifica directamente, ya sea incorporando nuevas cabezas, condicionamientos o ajustando sus pesos para tareas específicas.

Dicha taxonomía puede ser observada en la figura 2.25.

Caja negra: uso externo del modelo sin modificación interna

DA-Fusion [70] genera variantes de imágenes originales mediante modificaciones sutiles para mejorar la generalización. Utiliza la generación condicionada SEdit [48] para editar

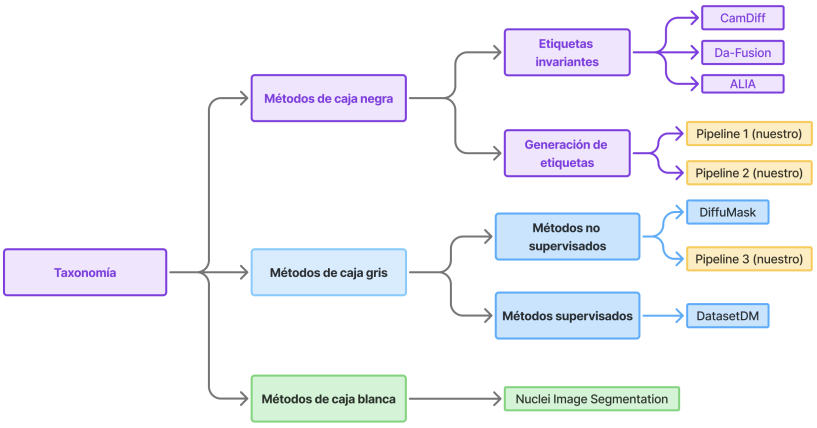


Ilustración 2.25: Taxonomía según el nivel de intervención sobre el modelo de difusión

imágenes con control, combinando la imagen y una descripción textual genérica, por ejemplo, *A photo of a class* .

Para clases difíciles de generar, se recurre a Textual Inversion [25], que aprende nuevos embeddings para representar conceptos específicos, facilitando la síntesis de clases poco comunes. Este método se ve en la figura 2.26.

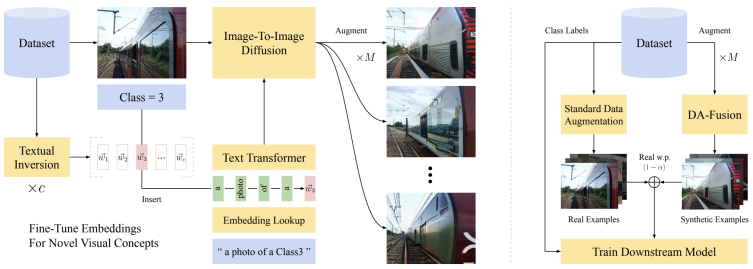


Ilustración 2.26: Funcionamiento de DA-Fusion [48]

ALIA [19] emplea técnicas de generación *image-to-image* como Instruct Pix2Pix, guiados por descripciones generadas con BLIP y enriquecidas por GPT-4. Se filtran imágenes defectuosas utilizando CLIP y clasificadores específicos entrenados por clase. ALIA detecta tres errores: fallo total (imagen inválida), fallo de identidad (igual a la original) y corrupción de clase (imagen válida con etiqueta incorrecta). Este método se observa en la figura 2.27.

CamDiff [46] mejora el aumento para detección de objetos camuflados (COD) insertando objetos salientes como *hard negatives* mediante *inpainting*.

Divide la imagen en una cuadrícula 3×3, toma la región central como el *bounding box* del objeto camuflado, selecciona aleatoriamente una región periférica, la recorta, aplica inpainting sobre un porcentaje con una clase aleatoria, y la reintroduce. Se valida la coherencia con CLIP. Se puede observar su funcionamiento en la figura 2.28.

SGAFC [18] aumenta datos para tareas de conteo, generando imágenes condicionadas tanto por descripciones textuales (BLIP2) como por mapas de densidad, con la ayuda de un

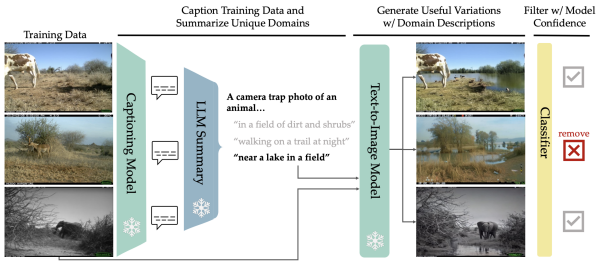


Ilustración 2.27: Funcionamiento de ALIA [19]

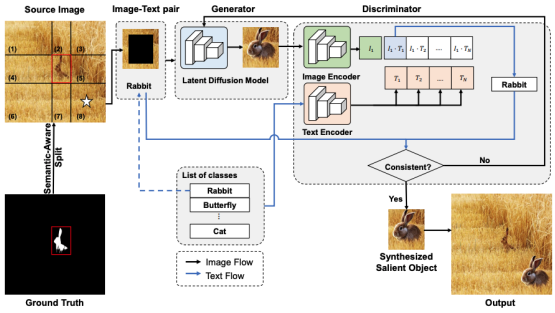


Ilustración 2.28: Funcionamiento de CamDiff [46]

ControlNet entrenado para esta tarea. Se puede observar su funcionamiento en la figura 2.29

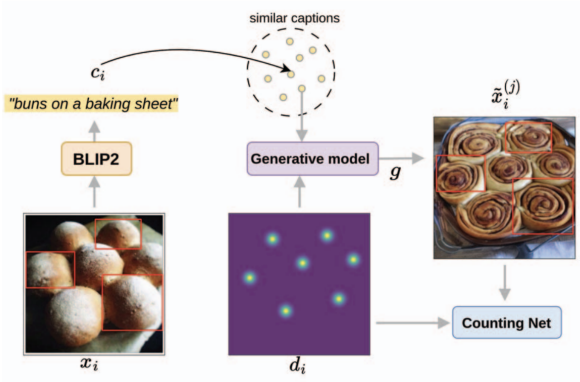


Ilustración 2.29: *Pipeline* de SGAFC [18]

Data Augmentation for Object Detection via Controllable Diffusion Models [21]: Proponen dos técnicas de aumento de datos para tareas de detección utilizando modelos de difusión. La primera es elegir una muestra y hacer *inpainting* sobre la imagen original utilizando como máscara una *bounding box* aleatoria. La segunda es utilizar un modelo de difusión condicionado por detección de bordes *HED* y reciclar las anotaciones de la imagen original utilizando CLIPs para filtrar detecciones incorrectas. Se puede observar un diagrama de su método en la figura 2.30.

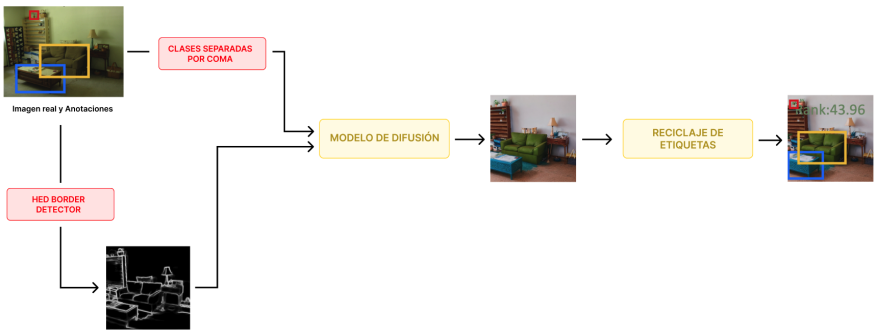


Ilustración 2.30: Método de aumento de datos mediante modelo de difusión condicionado por detección de bordes.

Caja gris: uso de representaciones internas del modelo

DifuMask [74] DifuMask [74] utiliza los mapas de atención cruzada entre tokens de texto y mapas intermedios de la U-Net en difusión texto-imagen para extraer segmentaciones semánticas de las imágenes generadas.

Extrae y reescala mapas de atención cruzada entre texto y mapas de características de la U-NET $\mathcal{A}_j^{s,t}$ de distintas capas s y *timesteps* t, normalizándolos y promediándolos para obtener un mapa global por token:

$$\hat{\mathcal{A}}_j = \frac{1}{|S||T|} \sum_{s \in S, t \in T} \frac{\mathcal{A}_j^{s,t}}{\text{máx}(\mathcal{A}_j^{s,t})}. \tag{2.25}$$

Estos mapas se convierten en máscaras binarias mediante umbral adaptativo refinado con DenseCRF y AffinityNet, combinando ambas con IoU para obtener la máscara final.

DatasetDM [73]

entrena una red neuronal para que, a partir de los mapas de atención cruzada y los mapas de características de la U-net se puedan obtener etiquetas de segmentación de instancias, semántica, estimación de pose y profundidad.

Se promedian mapas de atención y se concatenan con mapas de características $\mathcal{F}$ de la U-Net, procesándolos con una convolución 1×1:

$$\hat{\mathcal{F}} = \text{Conv}\left(\text{concatenate}(\mathcal{F}, \hat{\mathcal{A}})\right).$$

Los tensores resultantes se reescalan, concatenan y pasan a un *perception decoder* especializado según la tarea como se observa en la figura 2.31.

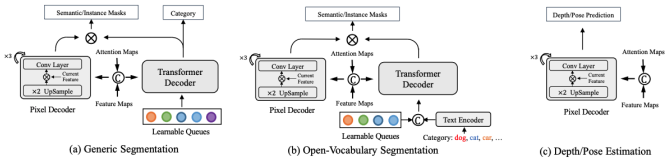


Ilustración 2.31: Decodificadores por tarea en DatasetDM [73]

Caja blanca: modificación del modelo de difusión

InstaGen [23] modifica la arquitectura del modelo de difusión para añadir una cabeza de detección inspirada en Grounded DINO. Esta cabeza aprovecha los mapas características generados por la U-Net, permitiendo generar imágenes y sus anotaciones de forma simultánea. Requiere realizar un fine-tune del modelo original. Esto se puede observar en la figura 2.32.

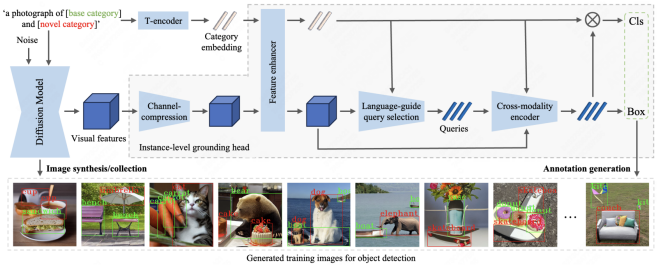


Ilustración 2.32: Arquitectura conjunta de InstaGen [23]

Capítulo 3

Metodología, tecnologías y modelos de difusión utilizados

3.1. Herramientas y tecnologías utilizadas

3.1.1. Aplicaciones

ComfyUI

ComfyUI es una herramienta *open-source* que permite construir flujos de generación de imágenes mediante modelos de difusión a bajo nivel usando una interfaz gráfica basada en nodos. Su diseño requiere conocimientos técnicos sobre arquitecturas como UNet, CLIP o VAE y sobre el funcionamiento intrínseco de un LDM. Ofrece un control granular y una API vía *WebSockets* para su integración en otros sistemas, lo que facilita tanto el prototipado como la puesta en producción.

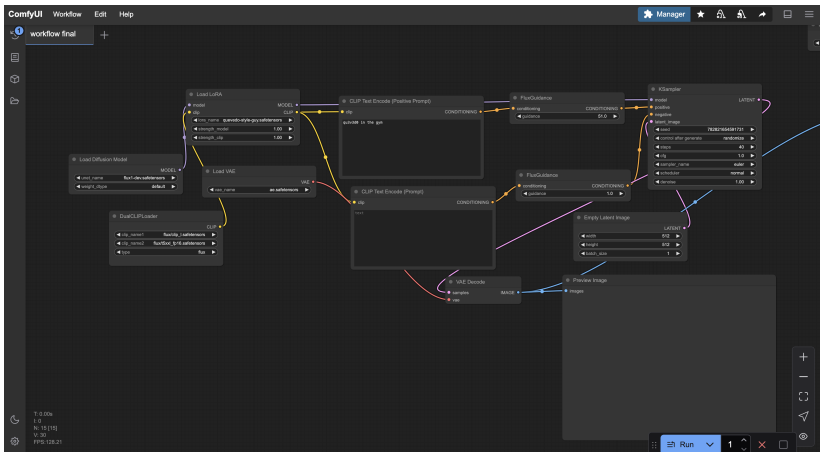


Ilustración 3.1: Ejemplo de flujo *text-to-image* en ComfyUI utilizando FLUX y LoRA.

n8n

n8n es una plataforma *low-code* orientada a la creación de automatizaciones a través de flujos visuales basados en nodos. Permite realizar llamadas HTTP, manejar datos estructurados, y conectarse con servicios externos y bases de datos de forma sencilla.

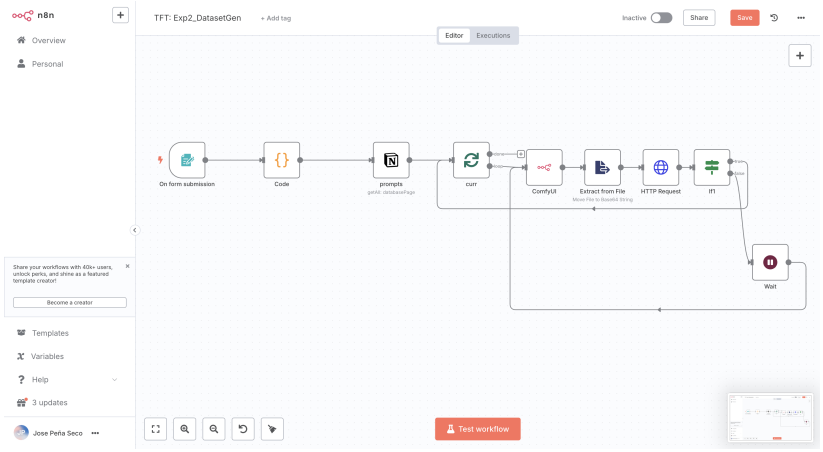


Ilustración 3.2: Flujo de trabajo en n8n.

3.1.2. Tecnologías

Diffusers

Diffusers es una librería desarrollada por HuggingFace que proporciona componentes de bajo nivel para trabajar con modelos de difusión de forma granular, incluyendo la carga y ejecución de UNets, CLIPs y VAEs. Ofrece una mayor accesibilidad a los pesos y capas internas que herramientas como ComfyUI. Esto permite realizar flujos mas complejos y experimentos que requieran acceso directo a las entrañas de los modelos.

PyTorch

PyTorch es una de las librerías de *deep learning* más utilizadas actualmente. Su adopción generalizada se debe en parte a que muchas otras librerías específicas, como Diffusers, están construidas sobre ella. Sus dos pilares fundamentales son el cálculo tensorial y la diferenciación automática. Para ello proporciona la clase Tensor, que permite habilitar el calculo de gradientes de forma automática mediante el parámetro *require_grad* haciendo prácticamente trivial aplicar métodos de optimización como descenso por el gradiente. Además, proporciona una API flexible y modular para la construcción de redes neuronales (incluyendo capas *fully connected*, convolucionales, transformers...), algoritmos de optimización, métricas para la evaluación de los modelos, entre otros. Por último, tiene implementadas varias métricas de evaluación de modelos generativos tales como el FID y el CLIP score.

React

React es una librería de JavaScript desarrollada por Meta que permite construir interfaces de usuario interactivas de forma modular y escalable. Su sistema basado en componentes

reutilizables y la gestión eficiente del estado han convertido a React en un estándar para el desarrollo de aplicaciones web modernas.

Django & Django Rest Framework

Django es un *framework* de Python para el desarrollo web, que incluye ORM, sistema de plantillas, enrutamiento y otras herramientas. Django Rest Framework extiende esta funcionalidad para facilitar la creación de APIs RESTful, permitiendo una integración eficiente entre el backend y otros servicios.

Fabric.js

Fabric.js es una biblioteca de JavaScript que facilita la creación y manipulación de gráficos en lienzos HTML5 (canvas). Proporciona una API potente y sencilla para trabajar con objetos como rectángulos, círculos, imágenes y texto, permitiendo transformaciones como escalado, rotación, agrupación y eventos interactivos. Fabric.js es especialmente útil en aplicaciones web que requieren funciones de edición visual como la construida en este trabajo.

FastAPI

FastAPI es una librería moderna y minimalista de Python diseñada para construir APIs REST de manera eficiente y sencilla. Su principal enfoque es ofrecer solo las herramientas esenciales para gestionar solicitudes HTTP, manteniendo una estructura limpia y fácil de entender. Destaca por su compatibilidad total con la programación asíncrona de Python, lo que le permite manejar múltiples peticiones de forma concurrente y eficiente, a diferencia de frameworks como Django, cuyo ORM está diseñado para operar de manera síncrona. Gracias a su rendimiento, simplicidad y soporte nativo para tipado estático y documentación automática, FastAPI se posiciona como una opción ideal para desplegar microservicios, especialmente aquellos que integran modelos de inteligencia artificial u otros componentes livianos.

Docker

Docker es una plataforma de virtualización basada en contenedores que permite empaquetar y distribuir aplicaciones junto con sus dependencias, garantizando portabilidad y reproducibilidad.

Autodistill

Autodistill es una librería en Python que facilita el autoetiquetado de datos mediante modelos de vocabulario abierto como GroundedSAM o GroundedDINO. Está diseñada para procesos de destilación, en los que se utiliza un modelo grande para generar etiquetas con las que entrenar modelos más pequeños y eficientes.

Git y GitHub

Se ha utilizado Git como sistema de control de versiones y GitHub como servidor remoto.

3.2. Elección de modelos de difusión

El primer paso en el desarrollo de este TFG fue seleccionar el modelo de síntesis de imágenes más adecuado, considerando los siguientes criterios:

1. **Calidad de síntesis:** La generación de datasets sintéticos y el aumento de datos requiere imágenes realistas y detalladas. Se priorizarán modelos que ofrezcan alta calidad visual dentro de las limitaciones del hardware disponible.
2. **Limitaciones de hardware:** Los modelos generativos suelen ser, como hemos visto anteriormente, intensos en recursos y especialmente en memoria de GPU (VRAM). Es por ello que es necesario asegurar que el modelo pueda ser ejecutado con los recursos que disponemos.
3. **Capacidad de condicionamiento:** No todos los modelos permiten los mismos tipos de entradas (como se explica en la sección 2.1.4.3). Por eso, se dará preferencia a los modelos que acepten más formas de entrada, como texto, imágenes o máscaras. Esto nos da más flexibilidad a la hora de construir los pipelines para el aumento de datos.

3.2.1. Calidad de síntesis

Para identificar los modelos con mejor calidad en la síntesis de imágenes, utilizamos, por un lado, *benchmarks* tanto cualitativos como cuantitativos, y por otro, dos experimentos específicos centrados en el dominio de interés.

Evaluación mediante benchmarks

Para la evaluación cualitativa, utilizamos el *Image Arena Leaderboard* de HuggingFace [hug], una plataforma donde los usuarios comparan pares de imágenes generadas por distintos modelos. A partir de estas elecciones humanas se calculan las puntuaciones ELO de cada modelo (figura 3.3).

Los modelos mejor valorados en el momento de nuestra selección eran *Flux 1.1 [dev]* y *Stable Diffusion 3.5 Large*, ya que *HiDreaml1-Dev*, aunque actualmente en primer lugar, fue lanzado en 2025 y no estaba disponible entonces. También se consideraron versiones anteriores como *SD3.5 Medium*, *SDXL 1.0*, *SD 2.1* y *SD 1.5*.

Como complemento, consultamos el estudio de Lei et al. [41], que compara diversos modelos —incluidos algunos propietarios— mediante métricas cuantitativas estándar (figura 3.4).

Ambas evaluaciones, cualitativa y cuantitativa, coinciden en señalar a *Stable Diffusion 3.5 Large* y *Flux 1.1 [dev]* como los modelos *Open Source/Open Weights* de mayor calidad disponibles en ese momento.

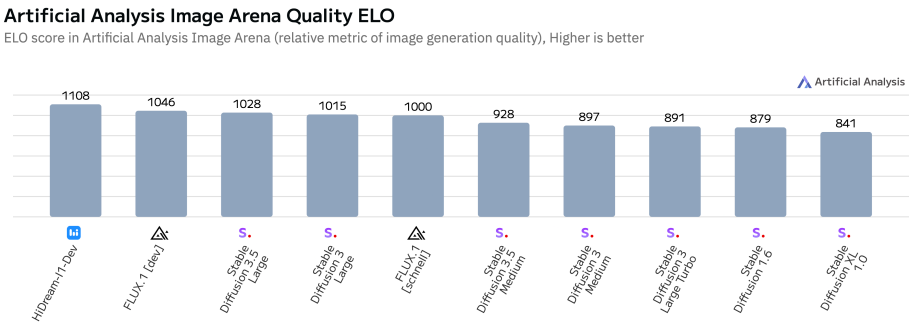


Ilustración 3.3: Comparativa entre modelos Open Source y Open Weights en la Image Arena Leaderboard de HuggingFace [hug].

Model	CLIPScore	HPSv2	Aesthetic Score	GPT-4o	Human
FLUX.1	28.49	0.28	6.09	7.29	**9.38**
Ideogram2.0	28.88	**0.30**	6.22	7.57	7.08
Dall-E3	29.31	0.28	6.13	6.87	5.90
Midjourney	**30.70**	0.29	6.27	**8.61**	8.68
SD3	29.56	**0.30**	6.38	7.01	6.87
Jimeng	29.93	**0.30**	**6.46**	6.66	7.56

Ilustración 3.4: Comparativa de modelos de difusión mediante métricas cuantitativas [41].

Evaluación mediante experimentos propios

En primer lugar, se generaron 40 *prompts* centrados en escenas marítimas con ayuda de GPT-4o, a partir de los cuales se sintetizaron imágenes utilizando ambos modelos. Estas imágenes se evaluaron mediante la métrica *CLIP Score*, que mide la coherencia entre la imagen generada y la descripción textual.

Luego, se recopilaron 40 imágenes reales de entornos marítimos. A partir de ellas, se generaron descripciones utilizando GPT-4o como modelo de visión-lenguaje. Estas descripciones se usaron para crear nuevas imágenes condicionadas y se compararon con las originales mediante el *FID score*. Los resultados cuantitativos se resumen en la tabla 3.1.

Métrica	**Stable Diffusion 3.5 (SD3.5)**	**Flux 1.1 [dev]**
CLIP Score	21.1673	20.6988
FID	182.04	184.4

Cuadro 3.1: Comparativa entre SD3.5 y Flux 1.1 [dev] en síntesis de entornos marítimos basada en CLIP Score y Aesthetic Score.

Además, se realizó una evaluación cualitativa. En la figura 3.5 se muestran ejemplos representativos de imágenes generadas por ambos modelos con sus respectivos *prompts*. En general, la calidad es comparable, aunque con diferencias notables: Flux ofrece una iluminación más realista y un mejor manejo de la perspectiva, mientras que SD3.5 produce imágenes

algo más planas pero con un mayor nivel de detalle. También se observó que Flux necesita menos *steps* para generar imágenes limpias, lo que mejora los tiempos de inferencia.

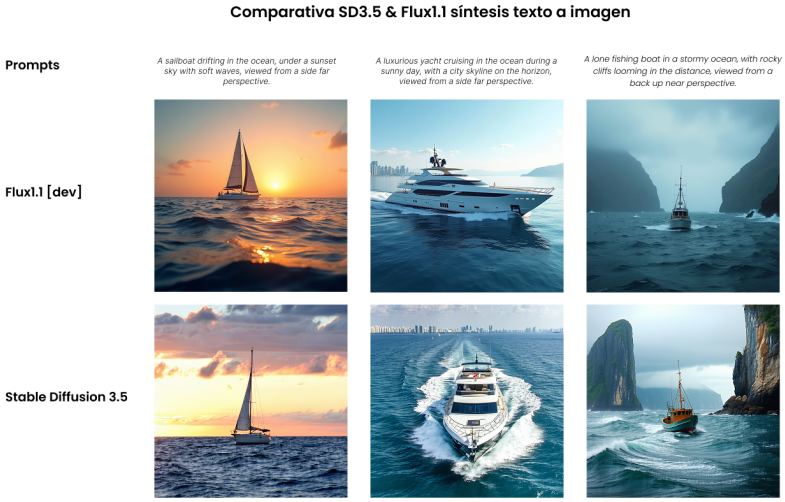


Ilustración 3.5: Comparativa de síntesis entre SD3.5 Large y Flux 1.1 [dev] utilizando 30 steps.

3.2.2. Limitaciones hardware

A lo largo del desarrollo de este trabajo se utilizó el siguiente hardware:

1. **Tarjetas gráficas:** RTX 3060 (usada en la fase inicial) con 12 GB de VRAM y RTX 3090 (utilizada en la fase final) con 24 GB de VRAM.
2. **Procesador:** AMD Ryzen 5 1600, con 6 núcleos y 12 hilos.
3. **Memoria RAM:** 54 GB.

Investigación previa

Como punto de partida, se investigaron los requisitos de VRAM de distintos modelos de difusión. La figura 3.6 muestra una estimación del consumo de memoria para varios de ellos, lo que sirvió como guía inicial para evaluar su compatibilidad con el hardware disponible.

No obstante, estas cifras deben interpretarse con cautela, ya que el consumo real puede variar en función de la implementación específica del modelo y del uso de técnicas de optimización como la cuantización o el *CPU offloading*. Por otra parte, durante el desarrollo se contempló la ejecución de los modelos en dos entornos distintos: la librería *Diffusers* y la interfaz *ComfyUI*, lo que añade otra fuente de variabilidad en el uso de memoria. Por este motivo, se consideró necesario realizar pruebas en ambos entornos para comprobar de forma empírica la viabilidad de cada modelo.

GPU Device	VRAM (GB)	Stable Diffusion 3.5 Medium (2.5B)	SDXL (6.5B including refiner)	Playground v2.5 (3.5B)	AuraFlow v0.2 (8.7B)	Stable Diffusion 3.5 Large / Large Turbo (8.1B)	FLUX.1 [dev] (12B)	FLUX.1 [schnell] (12B)
NVIDIA GeForce RTX 4060	8	!	!	!	!	!	!	!
NVIDIA GeForce RTX 3080	10	✓	!	!	!	!	!	!
NVIDIA GeForce RTX 3060 NVIDIA GeForce RTX 4070 AMD Radeon RX 7700 XT	12	✓	!	!	!	!	!	!
NVIDIA GeForce RTX 4060 Ti NVIDIA GeForce RTX 4070 Ti NVIDIA GeForce RTX 4080 AMD Radeon RX 7800 XT AMD Radeon RX 7600 XT	16	✓	✓	✓	!	!	!	!
AMD Radeon RX 7900 XT	20	✓	✓	✓	✓	!	!	!
NVIDIA GeForce RTX 3090 NVIDIA GeForce RTX 4090 AMD Radeon 7900XTX	24	✓	✓	✓	✓	✓	!	!
NVIDIA H100 AMD Instinct MI250X AMD Instinct MI300A AMD Instinct MI300X	32 (or greater)	✓	✓	✓	✓	✓	✓	✓

Hardware compatibility and VRAM requirements for open-image base models.

✓ *indicates the model runs on this device without any performance tradeoffs.*

! *indicates the model requires performance-compromising optimizations, such as quantization or sequential offloading, to run on this device.*

Ilustración 3.6: Comparativa de uso de VRAM entre varios modelos de difusión [3].

Ejecución en *ComfyUI*

Para llevar a cabo estas pruebas, se implementó un flujo de trabajo en *ComfyUI* para la generación de imágenes a partir de texto, utilizando los modelos Stable Diffusion 3.5 Large y Flux 1.1 [dev]. Se comprobó que ambos podían ejecutarse con el hardware disponible, y se compararon sus tiempos de inferencia en función del número de *steps*, usando una resolución de 512x512 píxeles (tabla 3.2).

Modelo	**20 steps**	**30 steps**	**40 steps**
Stable Diffusion 3.5 Large	22.11s	23.66s	24.71s
Stable Diffusion 3.5 Large (*Prompt* en caché)	3.22s	4.79s	6.33s
Flux 1.1 [dev] con *offloading*	35.14s	38.92s	42.29s
Flux 1.1 [dev] con *offloading* (*Prompt* en caché)	7.45s	10.98s	14.54s

Cuadro 3.2: Tiempos de inferencia (en segundos) para SD3.5 y Flux 1.1 [dev] en *ComfyUI*, según el número de *steps*.

Ejecución en *Diffusers*

Una vez verificada la ejecución de los modelos en *ComfyUI*, se intentó realizar inferencias utilizando la librería *Diffusers*. Sin embargo, los modelos más avanzados no pudieron ejecutarse en la GPU RTX 3090, incluso aplicando técnicas de *offloading*. Se probaron variantes como *Stable Diffusion 3.5 Large*, *Medium*, *Flux [dev]* y *Flux [schnell]*, sin éxito.

El modelo más exigente que se logró ejecutar en este entorno fue *Stable Diffusion XL (SDXL)*. Esta limitación se atribuye a que la implementación de *Diffusers* está menos optimizada en comparación con *ComfyUI*, lo que incrementa el uso efectivo de memoria.

3.2.3. Capacidad de condicionamiento

En cuanto a la capacidad de condicionamiento, se priorizó el modelo que ofreciera una mayor variedad de modos de entrada, ya que esto influye directamente en las posibilidades de crear técnicas de aumento de datos.

La tabla 3.3 muestra una comparación entre Stable Diffusion 3.5 Large y Flux 1.1 [dev], basada únicamente en los modos de condicionamiento oficialmente soportados por sus desarrolladores. No se han tenido en cuenta extensiones o funcionalidades añadidas por la comunidad.

Condicionamiento	**SD3.5**	**Flux 1.1 [dev]**
Canny (bordes)	Sí, mediante ControlNet	Sí, nativo / LoRA
Depth (profundidad)	Sí, mediante ControlNet	Sí, nativo / LoRA
Blur (imagen borrosa)	Sí, mediante ControlNet	No
Inpainting (relleno)	No	Sí, nativo / LoRA
Image variants (variantes de imagen)	No	Sí, nativo / LoRA
Pose (postura)	No	No

Cuadro 3.3: Comparativa de modos de condicionamiento entre Stable Diffusion 3.5 y Flux 1.1 [dev], indicando el método oficial de implementación.

3.2.3.1. Conclusiones

Tras el análisis comparativo, se establecieron las siguientes decisiones en función del entorno de ejecución y los recursos disponibles:

- ✔ Cuando sea posible utilizar *ComfyUI*, se optará por **Flux 1.1 [dev]**, debido a su mayor versatilidad en modos de condicionamiento y a la alta calidad visual de las imágenes generadas. Aunque presenta tiempos de inferencia más elevados y peores métricas cuantitativas que *Stable Diffusion 3.5 Large*, ofrece mejores resultados a nivel cualitativo en aspectos como realismo, iluminación y coherencia espacial. Dado que el trabajo no requiere ejecución en tiempo real, este coste adicional es aceptable.
- ✔ En entornos donde se utilice la librería *Diffusers*, se empleará *Stable Diffusion XL (SDXL)*, al ser el modelo de mayor calidad que puede ejecutarse satisfactoriamente en dicho entorno.
- ✔ En contextos con recursos limitados de VRAM, se recurrirá a *Stable Diffusion 2.1*, por ofrecer un equilibrio adecuado entre calidad y requisitos computacionales.

Capítulo 4

Diseño del sistema

Esta sección ofrece una visión general del sistema desarrollado. Se describe el módulo orquestador encargado de ejecutar los distintos *pipelines* de generación y etiquetado automático de datos sintéticos, así como su integración con servicios externos de inteligencia artificial y su comunicación con la base de datos a través de una API REST. Asimismo, se detalla la interacción del usuario con la aplicación web para gestionar, visualizar y exportar los datos generados, así como su preparación para ser utilizados en tareas de entrenamiento con modelos de detección como YOLO.

4.1. Visión general de la arquitectura

El sistema adopta una arquitectura orientada a servicios (SOA) compuesta por tres bloques principales: (1) una librería que implementa los distintos *pipelines* e incluye una interfaz de línea de comandos (*CLI*) para facilitar su uso; (2) servicios de inteligencia artificial externos, empleados para tareas como generación de imágenes y segmentación; y (3) una interfaz *web* que permite visualizar, filtrar, re-etiquetar y exportar los datos generados en un formato compatible con *Ultralytics*. Los módulos están desacoplados y se comunican entre sí mediante *APIs RESTful*. El diseño se ha desarrollado siguiendo buenas prácticas de ingeniería de software, como la inversión de dependencias y su inyección, utilizando los lenguajes *Python* y *TypeScript*. La figura 4.1 muestra el esquema general del sistema.

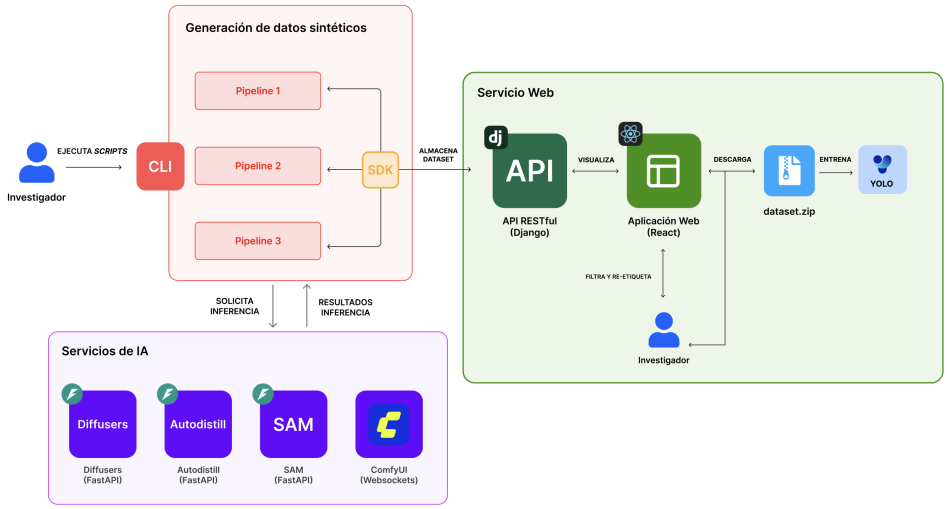


Ilustración 4.1: Visión general del sistema de *Datagen*

El flujo seguido durante el desarrollo y la realización de experimentos con el sistema es el siguiente: en primer lugar, se crea un nuevo *dataset* desde la plataforma *web*, especificando sus clases y etiquetas. A continuación, se ejecuta uno de los *pipelines* de generación a través de la interfaz de línea de comandos (*CLI*), a la que se le debe proporcionar la *id* del *dataset* que servirá como almacenamiento y un fichero *CSV* con la especificación del conjunto de datos a generar, esta especificación dependerá del *pipeline* ejecutado.

Una vez finalizada la generación, el usuario revisa los datos mediante las herramientas de visualización, el lienzo de re-etiquetado y las acciones en cascada, que permiten filtrar, seleccionar y eliminar rápidamente las imágenes incorrectas. Posteriormente, se utiliza la funcionalidad de *snapshots* para congelar una versión consistente del *dataset*, lista para su descarga. Finalmente, se combinan los datos descargados con el *dataset* real y se procede al entrenamiento correspondiente. Este flujo se representa en la figura 4.2.

4.2. Generación de datos sintéticos y servicios de inteligencia artificial

Esta es la parte más crítica del sistema y la que aporta mayor valor científico a este trabajo. Consiste en un conjunto de servicios de inteligencia artificial orquestados de forma reactiva por los distintos *pipelines* de aumento de datos. Su integración y funcionamiento se ilustran en el diagrama de la figura 4.3.

Los servicios de inteligencia artificial integrados en el sistema se agrupan en cuatro componentes principales: (1) el servicio de *autodistill*, que incluye los modelos *GroundedSAM* y *GroundingDino*; (2) el servicio de segmentación, que ejecuta *SAM*; (3) el servicio de difusión, basado en la librería *diffusers*, que incorpora modelos generativos junto con utilidades para la extracción de mapas de atención cruzada; y (4) *ComfyUI*, que alberga modelos más pesados

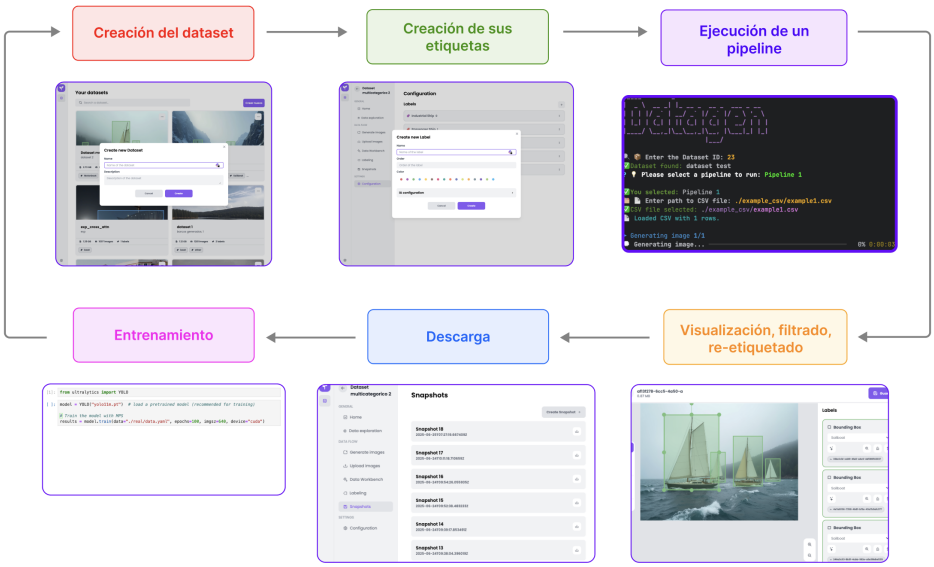


Ilustración 4.2: Flujo de creación y validación experimental

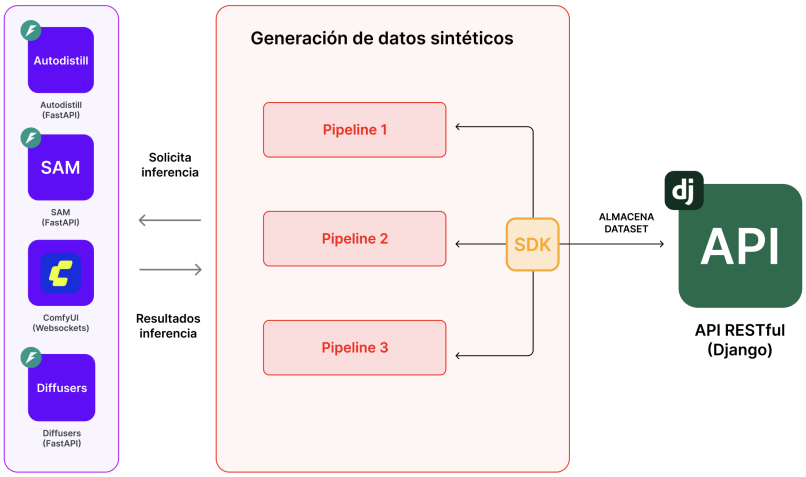


Ilustración 4.3: Integración de los *pipelines* con los servicios de inteligencia artificial

orientados a generación de alta calidad.

Todos los servicios se han desarrollado en *Python* utilizando *FastAPI*, aplicando el principio de inversión de dependencias para facilitar su extensibilidad, algo especialmente relevante en un entorno en constante evolución.

Para ello, se definen interfaces abstractas —como `Segmentator`, con un único método `segment`— que actúan como capa de abstracción entre la lógica de la API y las implementaciones concretas de los modelos. Por ejemplo, la clase `SAM2Segmentator` implementa esta interfaz dentro del servicio de segmentación. El endpoint `/segment_image/` utiliza una factoría que instancia dinámicamente una implementación específica de `Segmentator` según los parámetros de entrada.

Este enfoque permite incorporar nuevos modelos sin necesidad de modificar el resto del código: basta con implementar la interfaz y registrarla en la factoría. Esto dota a los servicios desarrollados de modularidad y escalabilidad.

A su vez, para garantizar la ejecución de dichos servicios en cualquier entorno y sistema se han *dockerizado*.

A continuación, se muestran fragmentos representativos de la interfaz y el endpoint:

```
class Segmentator(ABC):
    @abstractmethod
    def segment(self, image: Image.Image,
    bboxes: Optional[list[int]],
    points: Optional[list[list[int]]]) -> Result:
        ...

@app.post("/segment_image/")
async def segment(body: RequestBody) -> Result:
    image_data = base64.b64decode(body.image)
    image = Image.open(io.BytesIO(image_data))
    segmentator = SegmentatorFactory().create_from(name=body.segmentator,
                                                   model=body.model)
    result = segmentator.segment(image, body.bboxes, body.points)
    return result
```

Por otra parte, el módulo de *ComfyUI* se instaló a partir de su repositorio oficial y se configuró como un servicio en segundo plano sobre *Linux*, asegurando así su disponibilidad y ejecución continua. La configuración empleada se puede observar en la figura 4.4.

```
[Unit]
Description=Comfy Launch Service
After=network.target

[Service]
ExecStart=/root/.local/bin/comfy launch -- --normalvram --listen 0.0.0.0
WorkingDirectory=/root/
Restart=always
User=root

[Install]
WantedBy=multi-user.target

"/etc/systemd/system/comfy.service" 14 lines, 234 bytes
```

Ilustración 4.4: Servicio de ejecución de ComfyUI

Para su integración con el sistema, se implementó en la librería de *pipelines* una clase ‘ComfyUIImageGenerator‘, encargada de comunicarse con la *API* de *WebSockets* de *ComfyUI* y gestionar el proceso de generación de imágenes.

```
class ComfyuiImageGenerator(ImageGenerator):
def __init__(self, server_ip="comfyui.autoescuelaseco.cloud", port=80,
workflow=None):
```

```
        ...
    def queue_prompt(self, prompt):
        ...
        req = urllib
            .request
            .Request(f"https://{self.server_address}/prompt", data=data)
        return json.loads(urllib.request.urlopen(req).read())

    def get_images(self, prompt):
        ...
        ws = websocket.WebSocket()
        ws.connect(f"wss://{self.server_address}/ws?clientId={self.client_id}")
        ...
        return output_images

    def generate(self):
        prompt = json.loads(json.dumps(self.prompt_template))
        ...
        return pillow_images
```

La librería que implementa los distintos *pipelines* de generación se estructura en tres partes: por un lado, los módulos encargados de la conexión con servicios externos de inteligencia artificial; por otro, utilidades internas necesarias para construir los flujos de generación; y, por último, una interfaz de línea de comandos.

Todas las implementaciones siguen un enfoque común: definir interfaces que actúan como capa de abstracción entre los *pipelines* y sus dependencias. Esto permite sustituir o ampliar componentes —como modelos generativos o servicios externos— sin modificar la lógica del *pipeline*, haciendo el sistema más flexible y mantenible.

Un ejemplo de este enfoque es la interfaz 'ImageGenerator', para la cual se han desarrollado implementaciones concretas utilizando la *API* de *Flux*, la *API* de *Stable Diffusion*, el servicio de *ComfyUI* (como se explicó anteriormente) y la librería *Diffusers*.

```
class ImageGenerator(BaseModel):
    @abstractmethod
    def generate(self) -> Image.Image:
        ...
```

Finalmente, se desarrolló una interfaz de línea de comandos que permite ejecutar todos los *pipelines* de forma sencilla e intuitiva para cualquier usuario final. Para su implementación se utilizaron las librerías *rich* y *questionary*. En la figura 4.5 se muestra dicha interfaz en ejecución.

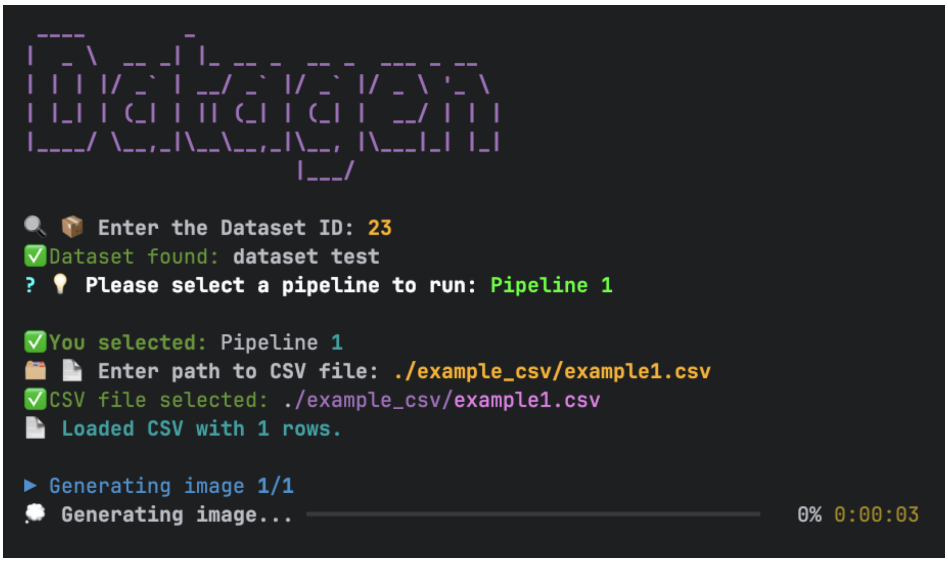


Ilustración 4.5: Visualización de la interfaz de línea de comandos para la ejecución de *pipelines*

4.3. Visualización, re-etiquetado y filtrado de datos incorrectos

Durante esta fase se desarrolló un *frontend* y un *backend*. Por una parte, el *frontend* fue desarrollado utilizando *TypeScript* y *React*, con *Vite* como herramienta de construcción. Para los estilos se empleó *TailwindCSS*, junto con la librería de componentes *Ark UI*. Por otra parte, el *backend* fue implementado con *Django* y *Django REST Framework*. Esto se puede observar en la figura 4.6.

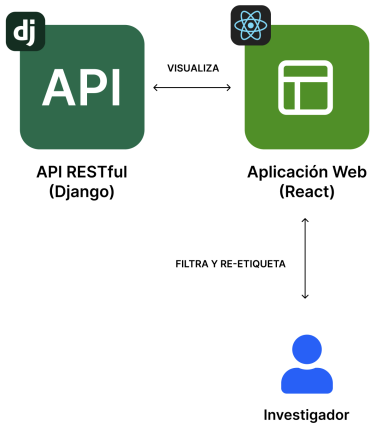


Ilustración 4.6: Diagrama de la fase de Visualización, re-etiquetado y filtrado de datos incorrectos

La arquitectura de el *backend* combina el patrón básico MVP de Django —adecuado para estructuras CRUD simples— con una estructura modular basada en *Serializers* y *ViewSets*, lo que facilita el desarrollo de una API REST mantenible y escalable.

```
class DatasetViewSet(viewsets.ModelViewSet):
    queryset = Dataset.objects.all().order_by('-id')
    serializer_class = DatasetSerializer
    permission_classes = [permissions.IsAuthenticated]

    def get_queryset(self):
        return self.queryset.filter(user=self.request.user)

    def perform_create(self, serializer):
        user = self.request.user
        serializer.save(user=user)
```

Para los casos más complejos que requieren lógica de negocio difícil de modelar con un enfoque CRUD tradicional, se adoptó el enfoque de *Domain-Driven Design* (DDD), separando las capas de dominio, infraestructura y aplicación. Un ejemplo de esta arquitectura se encuentra en el caso de uso de la descarga de un *dataset* para tareas de detección en formato *Ultralytics*, donde la lógica se organiza en torno a un comando, un caso de uso y sus correspondientes dependencias que se inyectan en el constructor, facilitando así la escalabilidad y el mantenimiento del sistema.

```
@dataclass
class DownloadDatasetForDetectionCommand:
    dataset_id: int

class DownloadDatasetForDetectionUseCase:
    @transaction.atomic
    def execute(self, command: DownloadDatasetForDetectionCommand):
        if not Dataset.objects.filter(pk=command.dataset_id).exists():
            return Result.failure({
                "message": "Dataset not found",
                "error_code": 1})
        ...
        return Result.ok({
            "dataset_id": dataset.id,
            "snapshot_id": snapshot.id,
            "snapshot_url": 'media/snapshots/dataset_{}__snapshot_{}.zip'
            .format(dataset.id, snapshot.id)
        })
```

El *frontend* es una aplicación *Client Side Rendering*, es decir, el proceso de renderización se produce en el cliente. Este fue modularizado mediante la creación de múltiples componentes y páginas. Los componentes se implementaron como componentes de *React* funcionales, es decir, son una función que retorna *JSX*. Esto se puede observar en el siguiente fragmento de código:

```
type FormikItemPickerProps<T> = {
  name: string;
```

```
  inputs: (T &FormItem)[];
  children?: (item: T & FormItem, active : boolean) => React.ReactNode;
} & Partial<FormItemPickerProps<T>>;

export function FormikItemPicker<T>({ name, inputs, children, ...props}:
FormikItemPickerProps<T>) {
  const [field, , helpers] = useField(name);

  return (
    <FormItemPicker
      inputs={inputs}
      value={field.value}
      onChange={(id) => helpers.setValue(id)}
      children={children}
      {...props}
    />
  );
}
```

Para la creación y validación de formularios se utilizó la librería *Formik* junto con *Yup* definiendo esquemas de validación. Esto se puede observar a continuación:

```
const validationSchema = Yup.object({
    email: Yup.string().required('Email is required'),
    password: Yup.string().required('Password is required'),
});
```

4.4. Descarga de los datos y entrenamiento de modelos

Finalmente, el usuario puede generar una versión descargable del conjunto de datos en un fichero `.zip`, estructurado según el formato de *Ultralytics*, tal y como se muestra en la figura 4.7.

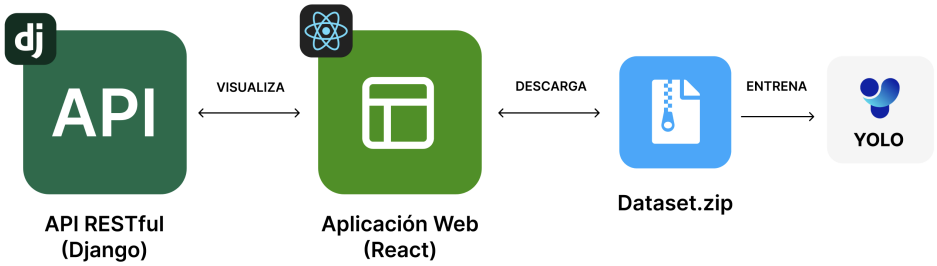


Ilustración 4.7: Flujo de descarga de datos y entrenamiento del detector YOLO

El formato de descarga está estructurado en un archivo comprimido `.zip`, el cual contiene las siguientes carpetas: `images` y `labels`. La carpeta `images` incluye todas las imágenes que

forman parte del conjunto de datos, mientras que la carpeta `labels` contiene los archivos de etiquetas asociados a cada imagen.

Cada archivo de etiqueta está en formato de texto plano `.txt` y sigue el estándar utilizado por *Ultralytics* para la anotación de objetos en tareas de detección. En cada archivo `.txt`, cada línea corresponde a un objeto detectado en la imagen y se compone de cinco elementos, separados por espacios. El primer elemento es un valor entero que representa la categoría del objeto, correspondiente a un índice dentro del conjunto de clases definido. Los cuatro elementos restantes representan las coordenadas de la *bounding box* normalizada, es decir, las posiciones del objeto en la imagen en relación con su altura y anchura, en el formato `(x_center, y_center, width, height)`. Estos valores están normalizados para que estén en el rango $[0, 1]$.

Este formato es compatible con *Ultralytics*, que es ampliamente utilizado en tareas de detección de objetos y es especialmente conocido por su implementación de modelos como YOLO (You Only Look Once).

Capítulo 5

Generación de datos sintéticos

A la hora de diseñar un *pipeline* de generación de datos sintéticos es fundamental distinguir entre dos aspectos fundamentales: el diseño del conjunto de datos y la creación individual de muestras.

El diseño del conjunto de datos parte de la identificación de los factores que deben estar representados en las muestras generadas (como clases, condiciones, contextos o perspectivas). Cada uno de estos factores puede tomar distintos valores, y su combinación define el espacio total de configuraciones posibles, que denotamos como $\mathcal{A} = \mathcal{A}^{(1)} \times \mathcal{A}^{(2)} \times \cdots \times \mathcal{A}^{(k)}$. Una configuración concreta $a = (a^{(1)}, a^{(2)}, \ldots, a^{(k)}) \in \mathcal{A}$ representa una muestra deseada, y debe traducirse en un conjunto de entradas o condicionamientos que sean compatibles con el modelo generativo.

Una vez seleccionada una configuración a, se construyen sus correspondientes condicionamientos mediante una función f, de forma que $f(a) = (c^{(1)}, c^{(2)}, \ldots, c^{(m)})$. Estos condicionamientos se utilizan como entrada de un *pipeline* de generación que produce tanto la imagen sintética x como sus etiquetas asociadas y, expresado como $P(c^{(1)}, c^{(2)}, \ldots, c^{(m)}) = (x, y)$. Este proceso busca mantener la coherencia con el diseño original del conjunto de datos, asegurar una calidad visual suficiente y, en la medida de lo posible, automatizar la obtención de etiquetas útiles para tareas supervisadas. Este formalismo puede verse ilustrado en forma de diagrama en la figura 5.1.

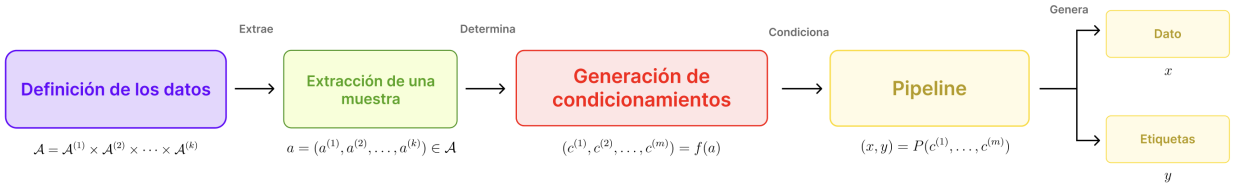


Ilustración 5.1: Diagrama de flujo de un *pipeline* de generación de datos sintéticos

5.1. Pruebas iniciales

En las primeras etapas del desarrollo, las limitaciones de hardware marcaron las decisiones iniciales. Al disponer únicamente de una *RTX 3060 de 12GB*, fue necesario adaptar la elección del modelo a estas restricciones. Por ello, y siguiendo el análisis previo de selección, se optó por utilizar *Stable Diffusion 2*.

El objetivo inicial fue desarrollar un proceso capaz de generar datos sintéticos con calidad suficiente como para formar un *dataset* completo, sin necesidad de combinarse con datos reales. Para ello, se estableció como requisito que las imágenes generadas debían ser visualmente realistas y coherentes, tanto en composición como en perspectiva y color.

A su vez, para abordar la tarea de detección, el objetivo fue definir una estrategia que permitiera extraer automáticamente las cajas delimitadoras de las instancias generadas. Esto implicaba que el sistema no solo debía producir imágenes realistas, sino también generar sus correspondientes etiquetas, lo cual requería un flujo adicional capaz de asociar cada instancia con su caja delimitadora.

A partir de esta premisa, se exploraron distintas estrategias de generación, que se describen a continuación.

Inpainting de objetos

La primera aproximación consistió en emplear *inpainting* como mecanismo para generar barcos sobre imágenes generadas sintéticamente y utilizar directamente la máscara empleada como *bounding box*. La idea era simple: si el modelo de difusión generaba el barco dentro de los bordes definidos por la máscara, esta misma podría reutilizarse como la anotación. Este proceso puede observarse en la figura 5.2.

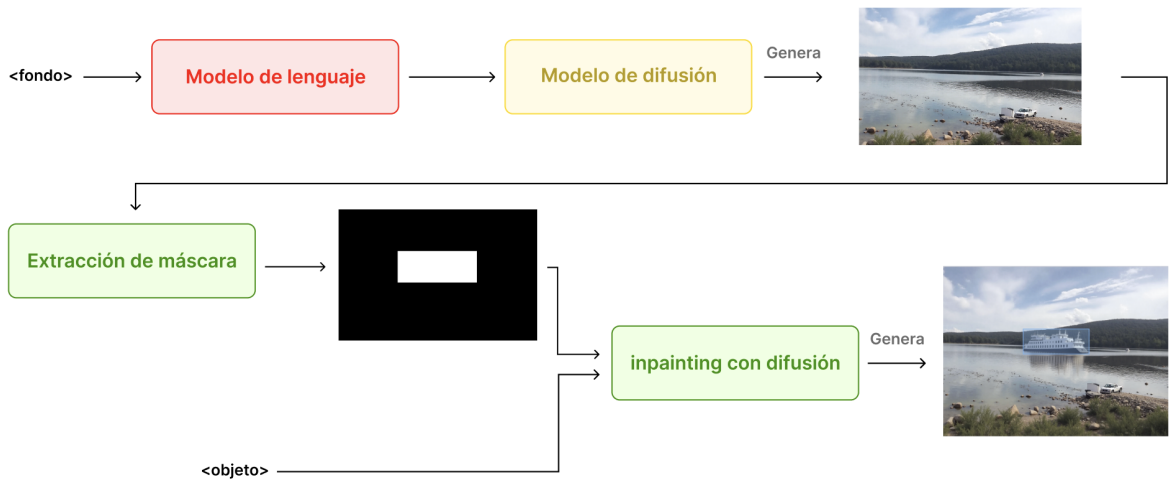


Ilustración 5.2: Diagrama conceptual método de *inpainting*

Para poner a prueba esta idea, se seleccionaron manualmente zonas oceánicas en imágenes reales y se aplicaron máscaras sobre ellas. Los modelos lograron generar barcos sintéticos de forma visualmente realista e integrados con el fondo. Sin embargo, el enfoque presentó varias limitaciones:

1. **Tamaño inconsistente:** el tamaño del barco variaba mucho entre generaciones, lo que impedía que la máscara original coincidiera con el objeto generado.
2. **Posición impredecible del objeto:** el modelo no respetaba una posición fija dentro de la máscara, generando el barco en lugares distintos cada vez.

Esto se puede observar en la figura 5.3, donde el objeto no coincide en tamaño con la máscara y su ubicación resulta impredecible.

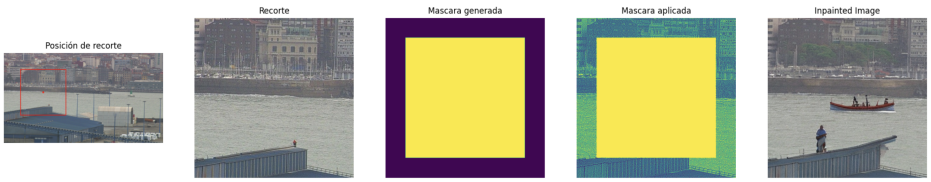


Ilustración 5.3: Ejemplo de *inpainting* sobre un fondo real, comprobando que la máscara no equivale a la *bounding box*

Ante las limitaciones del primer enfoque, se probó una segunda estrategia: extraer la máscara del objeto comparando la imagen original con la generada tras aplicar *inpainting*, asumiendo que los píxeles modificados correspondían al objeto. Sin embargo, el modelo reconstruía toda la región enmascarada, incluso las zonas que visualmente parecían iguales, lo que impedía aislar el objeto debido a pequeñas variaciones de color. Se intentó suavizar estas diferencias comparando promedios de regiones (por ejemplo, bloques de 8×8 píxeles), pero las discrepancias seguían presentes en casi toda el área. Un intento fallido de extraer la máscara de esta forma se puede observar en la figura 5.4.

Ilustración 5.4: Ejemplo del intento de extraer la segmentación de instancia a partir de comparar promedios de grupos de píxeles entre imagen original y generada

Una posible solución a esta limitación es utilizar un modelo de segmentación como SAM para refinar la máscara de *inpainting* y así extraer la etiqueta con mayor precisión, como se muestra en la figura 5.5. Esta estrategia fue explorada en profundidad en el *pipeline* 2.

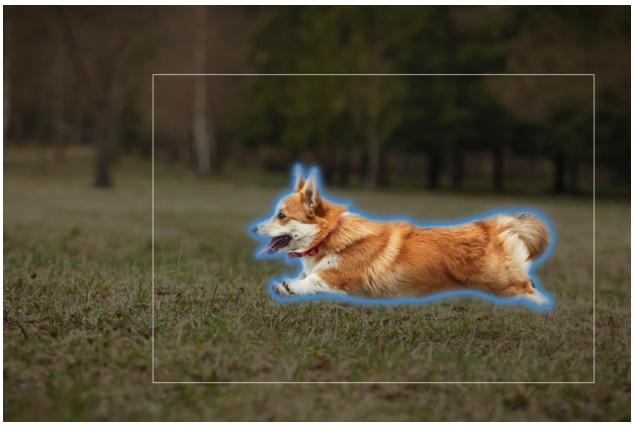


Ilustración 5.5: Ejemplo de SAM extrayendo un objeto a partir de una caja delimitadora de mayor tamaño

Además de las limitaciones específicas relacionadas con la extracción automática de etiquetas, este enfoque presenta otros retos generales:

1. **Elección del punto de inserción:** es necesario definir un mecanismo preciso y controlado que determine en qué parte de la imagen se aplicará el *inpainting*.
2. **Tamaño de la máscara:** también se requiere una forma sistemática de ajustar el tamaño de la región enmascarada al tamaño esperado del objeto. Aunque el modelo genera el barco en cualquier parte de la máscara, su tamaño tiende a adaptarse a las dimensiones de esta: máscaras grandes inducen objetos más grandes, y viceversa.

Composición de imágenes

La segunda aproximación se basó en un *pipeline* de composición de imágenes. La idea consistía en generar de forma independiente un barco y un fondo marítimo sintéticos, e insertar posteriormente el barco en una región adecuada del fondo. Para lograr una integración visual más realista, se aplicó *inpainting* en los bordes del objeto insertado. Este enfoque, además, permitía obtener fácilmente una *bounding box* precisa, ya que bastaba con delimitar el área ocupada por el barco tras colocarlo y escalarlo sobre el fondo. Esto se puede observar en la figura 5.6.

Para validar esta hipótesis, se realizaron experimentos insertando barcos reales (con el fondo previamente eliminado) sobre imágenes reales de entornos marítimos. Posteriormente, se aplicaron diferentes tipos de máscaras de *inpainting* en los bordes del objeto, tanto en la parte interior como exterior, para mejorar su integración visual.

Un ejemplo de este procedimiento puede verse en la figura 5.7, donde un barco carguero ha sido integrado correctamente en una escena marina. En este caso, la *bounding box* obtenida es precisa.

Sin embargo, a pesar del resultado exitoso, este método presenta una alta variabilidad y

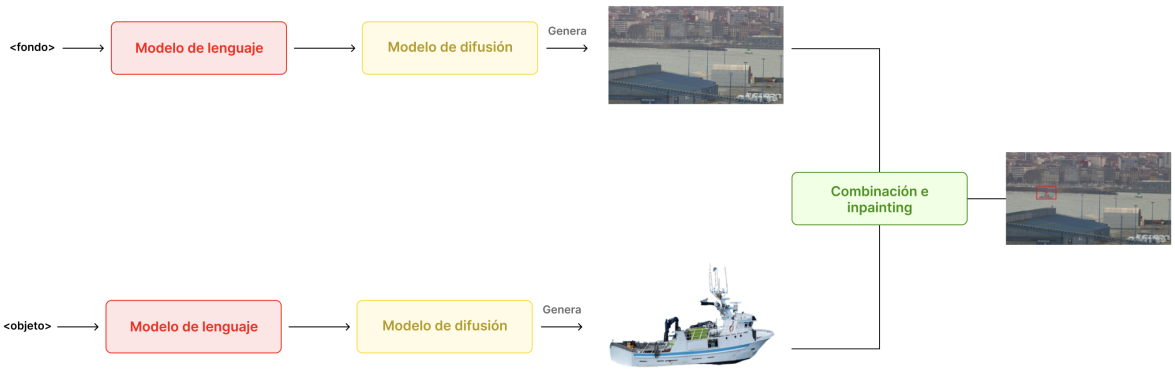


Ilustración 5.6: Diagrama conceptual del método de composición de imágenes

Ilustración 5.7: Resultado de integración de un barco real sobre una imagen utilizando *inpainting* en los bordes

una serie de dificultades prácticas importantes:

1. **Incompatibilidad de perspectivas**: si la perspectiva del barco no coincide con la del fondo (por ejemplo, un barco visto desde arriba sobre un mar con vista lateral), el resultado será poco realista, independientemente del *inpainting* aplicado. Una posible solución explorada fue el uso de mapas de bordes como Canny para condicionar la generación y así alinear las perspectivas de barco y fondo. No obstante, esto implicaría disponer de pares de entrada (Canny del barco y del fondo) con estructuras geométricas compatibles, lo cual introduce una dificultad adicional significativa en la preparación de los datos.

2. **Necesidad de armonización**: es fundamental ajustar el color del barco al fondo. Durante las pruebas, se emplearon diversos métodos de armonización disponibles en la librería Libcom [51], pero los resultados no siempre fueron satisfactorios.

3. **Eliminación del fondo**: ni Stable Diffusion ni Flux 1.1 por defecto generan objetos con fondo completamente transparente o mono-color. Esto fue probado generando imágenes con prompts tales como *"A photo of a fishing ship with white background"*, por lo que se necesita un proceso adicional de eliminación de fondo antes de la inserción.

4. **Posibilidad de corrupción de la *bounding box***: cuanto mayor sea el grosor de la máscara usada para *inpainting*, mejor se integra el barco en la imagen, pero también aumenta la probabilidad de que la *bounding box* resultante no se corresponda exactamente con la silueta del objeto, como se aprecia en la figura 5.8.

5. **Escalado del objeto**: debe definirse también un criterio para reescalar el barco antes de insertarlo, ya que este parámetro afecta tanto a la integración visual como a la validez de la *bounding box*.

Ilustración 5.8: Ejemplo donde la *bounding box* se ve alterada por una máscara de *inpainting* demasiado extensa

Además, este *pipeline* requería múltiples pasos intermedios (recorte, alineación, fusión, armonización y *inpainting*), lo que aumentaba su complejidad.

En conclusión, aunque esta solución permitía generar datos con etiquetas automáticas más controladas que con *inpainting* puro, dependía de muchos parámetros y se intentó buscar soluciones alternativas.

Inpainting de fondos

Otra estrategia explorada fue un *pipeline* basado en invertir el enfoque habitual: en lugar de generar primero el fondo y después insertar los objetos, se comenzaba generando los barcos. Estos se posicionaban aleatoriamente sobre una imagen vacía, y a continuación se aplicaba *inpainting* sobre toda la imagen, excepto en las regiones correspondientes a los barcos. De este modo, el modelo de difusión generaba un fondo coherente que integraba visualmente los objetos en la escena. Este método se puede observar en forma de diagrama en la figura 5.9.

Ilustración 5.9: Diagrama conceptual del método de *Inpainting* de fondos

Este enfoque presentaba múltiples ventajas:

1. Se elimina el problema de determinar manualmente la región de colocación: es el modelo quien adapta el fondo a los objetos, y no al revés.
2. La *bounding box* de cada barco permanece inalterada, ya que el fondo se modifica, pero los objetos se mantienen intactos.
3. La coherencia en la perspectiva de los barcos puede lograrse generando uno primero, extrayendo su contorno mediante Canny y utilizando este como condición para generar el resto de los objetos.
4. El tamaño de los barcos puede establecerse de forma arbitraria, ya que el fondo se adapta automáticamente a ellos.

Para validar este enfoque, primero se generaron imágenes de barcos a partir de distintos *prompts*, y se les eliminó el fondo utilizando el método *Background Remover* de la API de StabilityAI. En paralelo, se generaron fondos marítimos sintéticos. Luego los barcos se posicionaron aleatoriamente sobre estos fondos, evitando solapamientos. Y, finalmente, se aplicó *inpainting* sobre toda la imagen excepto las regiones correspondientes a los barcos, para integrar visualmente ambos elementos. Se puede observar ejemplos de muestras sintetizadas con el procedimiento anterior en la figura 5.10.

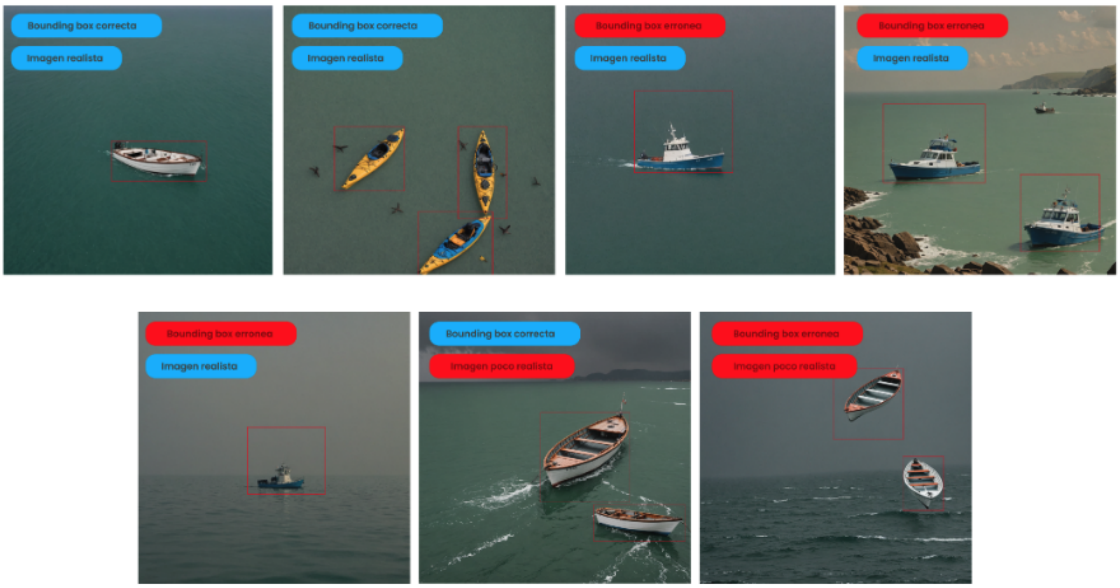


Ilustración 5.10: Ejemplos utilizando el *pipeline* de *inpainting* del fondo según si es una imagen realista y si su bounding box es correcta

Los resultados fueron en general satisfactorios desde un análisis cualitativo; sin embargo, se presentaron casos en los que el modelo no pudo generar un fondo coherente, ocasionando que algunos barcos aparecieran en el cielo o en contextos erróneos, como se observa en la figura 5.10. Además, en ocasiones el modelo modificaba partes del barco fuera de la región destinada al *inpainting*, lo que provocaba la corrupción de la *bounding box*.

Clustering de mapas de atención cruzada

La última estrategia explorada consistió en aprovechar la información interna del modelo de difusión para extraer de forma aproximada las etiquetas de las imágenes generadas. Tal como se explicó en la sección 2.1.4.2, durante el proceso de difusión el texto se introduce como condicionamiento mediante capas de atención cruzada entre los mapas de características de la U-Net y los *embeddings* del *prompt*.

A partir de estas capas de atención es posible obtener un mapa de calor para cada *token*. En dichos mapas, las zonas con mayor activación suelen corresponder a las regiones de la imagen donde se ha representado el concepto asociado a ese token. Así, mediante técnicas de umbralizado y *clustering*, se puede estimar la posición del objeto generado y extraer una caja delimitadora aproximada. Este proceso queda ilustrado en la figura 5.11.

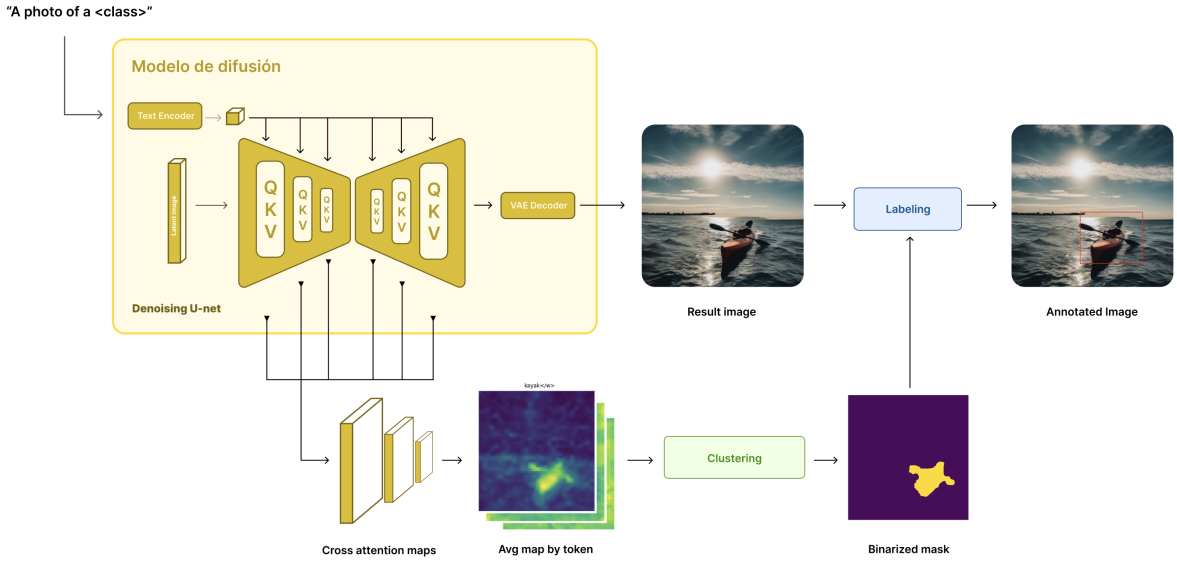


Ilustración 5.11: Ejemplo de método de *clustering* utilizando los mapas de atención cruzada.

La obtención de dichos mapas puede observarse en la siguiente expresión:

$$M_{b,i}(x,y) = \frac{1}{|T|\,|L|} \sum_{t=1}^{|T|} \sum_{\ell=1}^{|L|} \text{resize}_{(H_0,W_0)}\Big[\sum_{h=1}^{H} A_{t,\ell}[\,B+b,\ h,\ :,:,\ i\,] \Big]$$

Aquí, el tensor $A_{t,\ell} \in \mathbb{R}^{2B\times H\times H_{t,\ell}\times W_{t,\ell}\times S}$ representa el mapa de atención generado en la capa ℓ durante el paso de difusión t. La primera mitad de este tensor ($2B$) corresponde a la parte *incondicional* en caso de usar *classifier-free guidance* [32], esta se desecha; se emplea únicamente la parte *condicional* (índice $B+b$). Se suman las H cabezas de atención y luego se interpola cada resultado a una resolución común $H_0 \times W_0$, ya que las salidas de cada bloque de la U-Net presentan resoluciones diferentes (64x64, 32x32, 16x16 y 8x8). Finalmente, se promedian todos los pasos temporales y capas del modelo, obteniendo así un mapa de atención medio por *token*.

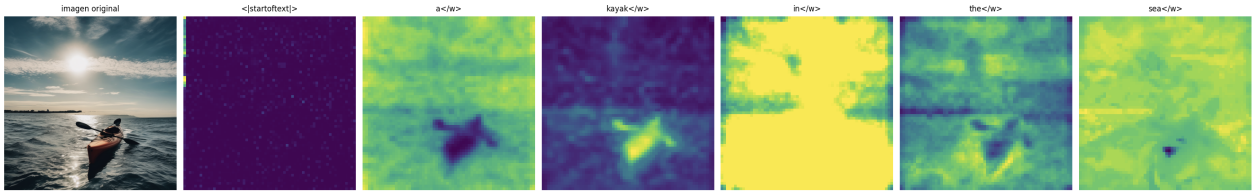


Ilustración 5.12: Visualización de los mapas atención cruzada promedios por *token* generados por *SDXL* mediante el *prompt* "A kayak in the sea"

Un ejemplo del resultado de la operación anterior puede observarse en la figura 5.12.

A partir de ahí, se selecciona el mapa correspondiente al objeto de interés (en el ejemplo anterior, el token kayak) y se le aplica una técnica de umbralizado para obtener una máscara binaria. Esta máscara permite calcular una *bounding box* aproximada que delimita la posición del objeto generado en la imagen. Este enfoque fue evaluado utilizando el modelo *SDXL*, aplicando distintos *prompts* y técnicas de binarización. Los experimentos permitieron identificar tanto el potencial del método como sus principales limitaciones.

Algunas ventajas son las siguientes:

1. **Versatilidad del método:** Puede aplicarse a una amplia variedad de dominios y objetos (animales, personas, vehículos, etc.). Siempre que el modelo pueda sintetizar el objeto, es posible extraer su caja delimitadora.
2. **Anotación automática coherente:** No es necesario decidir manualmente el tamaño o la posición del objeto; el modelo toma esas decisiones y ofrece una forma de recuperar esa información de forma precisa y consistente.

Sin embargo, si bien soluciona muchos problemas vistos en los otros métodos tiene una serie de inconvenientes:

1. **Dificultad con oclusiones**: Cuando se generan múltiples objetos similares, las zonas de atención pueden solaparse en el mapa de calor, haciendo imposible separar instancias individuales si están parcialmente ocluidas.
2. **Tokens compuestos**: Algunos objetos se representan mediante varios tokens (por ejemplo, *sailboat* como *sail* + *boat*). En estos casos, se requiere una estrategia para combinar correctamente los mapas de atención asociados.
3. **Dependencia del modelo**: La correspondencia entre tokens y regiones de atención varía entre modelos, ya que depende tanto del encoder textual como del entrenamiento. Esto dificulta la generalización del método a otros modelos de difusión. Por ejemplo, en algunos modelos la información espacial del objeto no se concentra en el token principal (como el nombre del objeto), sino en elementos aparentemente secundarios del texto, como los artículos o adjetivos.
4. **Bounding boxes inconsistentes**: La precisión de las cajas generadas depende del umbral utilizado. Sin refinamiento posterior, pueden obtenerse cajas excesivamente holgadas o recortadas, como ya ocurría en el enfoque basado en *inpainting*.

Podemos observar una comparación entre diferentes técnicas de umbralizado utilizadas a lo largo de las pruebas en el anexo 8.

Conclusiones

Las conclusiones extraídas tras estas pruebas iniciales son las siguientes:

1. **Estimación de la *bounding box*:** Extraer de forma directa una *bounding box* exacta durante el proceso de generación resulta complejo en la práctica. En lugar de buscar precisión absoluta desde el inicio, se plantea como solución realista obtener una primera aproximación —una máscara que cubra con holgura el objeto— y refinarla posteriormente con herramientas especializadas. Modelos como *SAM* ofrecen una vía efectiva para mejorar la segmentación sin añadir demasiada complejidad.
2. **Simplicidad del *pipeline*:** Los *pipelines* más complejos tienden a requerir múltiples parámetros y puntos de ajuste, lo que introduce mayor fragilidad y limita su reutilización en otros contextos. Por tanto, los enfoques que parten de estructuras sencillas pero generalizables —como aprovechar mecanismos internos del modelo de difusión o aplicar reglas simples para el posicionamiento y escalado de objetos— han demostrado ser más prácticos y eficientes, especialmente cuando se busca automatizar el proceso.
3. **Priorizar el aumento de datos:** Aunque el objetivo inicial era generar *datasets* sintéticos totalmente autónomos, se concluyó que esto no siempre es necesario para obtener valor. En muchos casos, los datos generados pueden utilizarse eficazmente como técnica de aumento, complementando conjuntos reales y mejorando la robustez del modelo. Este uso alternativo permite aprovechar datos sintéticos con menores exigencias visuales, ampliando así el margen de aplicabilidad de las técnicas probadas.

A partir de estas conclusiones, se procedió a definir un entorno experimental concreto y desarrollar tres *pipelines* distintos orientados a evaluar el potencial del aumento de datos mediante imágenes generadas sintéticamente.

5.2. Entorno experimental

Con el objetivo de ofrecer un entorno controlado donde se puedan diseñar y probar los *pipelines* desarrollados, se ha propuesto generar un conjunto de datos similar a un *dataset* de entrenamiento desarrollado por QAISC [62], utilizado en el entrenamiento de sus modelos de detección de embarcaciones marítimas.

Con el fin de aumentar la velocidad de los experimentos, se ha decidido trabajar con un subconjunto reducido del dataset de entrenamiento, compuesto por 900 imágenes y una única etiqueta llamada *boat*.

Para evaluar el rendimiento del conjunto de datos generado se propone un enfoque doble. En primer lugar, se calculará el *Fréchet Inception Distance (FID)* entre las imágenes reales y

las sintéticas, como métrica cuantitativa del realismo de las imágenes generadas. El objetivo será minimizar esta distancia, indicando mayor similitud entre ambos dominios.

En segundo lugar, se entrenará el modelo *YOLOv11m* bajo tres configuraciones: (1) solo con imágenes reales, (2) solo con imágenes sintéticas, y (3) con una combinación de ambas. En todos los casos, la validación se realizará exclusivamente sobre imágenes reales, permitiendo evaluar la capacidad de generalización de cada modelo. Como métricas principales se emplearán **AP50** y **AP**.

La métrica **AP** (Average Precision) mide el área bajo la curva de precisión-recall, promediando los valores obtenidos en distintos umbrales de *IoU (Intersection over Union)*, típicamente desde 0.5 hasta 0.95 en pasos de 0.05. Esto la convierte en una métrica exigente, ya que considera tanto la precisión de la detección como la calidad de la localización. Por su parte, **AP50** corresponde a la precisión promedio calculada con un único umbral de IoU = 0.5, lo que la hace más permisiva. En este caso, una predicción se considera correcta si se solapa al menos un 50 % con la anotación real. Ambas métricas se utilizan de forma complementaria: AP50 refleja la capacidad del modelo para detectar objetos, mientras que AP evalúa la precisión espacial de las predicciones. Su combinación permite obtener una visión más completa y rigurosa del rendimiento del modelo.

Para la ejecución de dichos entrenamientos se utilizó la librería *ultralytics* con una configuración de 500 épocas y 640 de resolución de imagen. El comando empleado para todos los entrenamientos se muestra a continuación:

```
from ultralytics import YOLO
model = YOLO("yolo11m.pt")  # load a pretrained model (recommended for training)
results = model.train(data="{dataset}", epochs=500, imgsz=640, device=0)
```

Algoritmo 5.1: Entrenamiento del modelo YOLO con Ultralytics

5.3. Pipeline 1

Este *pipeline* plantea la destilación del conocimiento de un modelo fundacional de gran tamaño en un modelo más compacto, apto para su ejecución en dispositivos *edge*. Para ello, se generan imágenes sintéticas que son etiquetadas automáticamente mediante el modelo grande y utilizadas posteriormente para entrenar el modelo ligero. El modelo en cuestión puede observarse de forma conceptual en la figura 5.13.

Aunque este enfoque es aplicable a una amplia variedad de modelos, en los experimentos se propuso una configuración concreta compuesta por tres etapas. En primer lugar, se utilizó el modelo generativo Flux 1.1 [dev] para sintetizar imágenes a partir de *prompts* generados automáticamente por un modelo de lenguaje (LLM). A continuación, dichas imágenes fueron etiquetadas de forma automática utilizando el marco Autodistill, que se apoya en Grounded-SAM como modelo fundacional de segmentación y detección. Este proceso se puede observar en la figura 5.14.

Finalmente, las imágenes anotadas se emplearon para entrenar Yolo11M, un modelo ligero

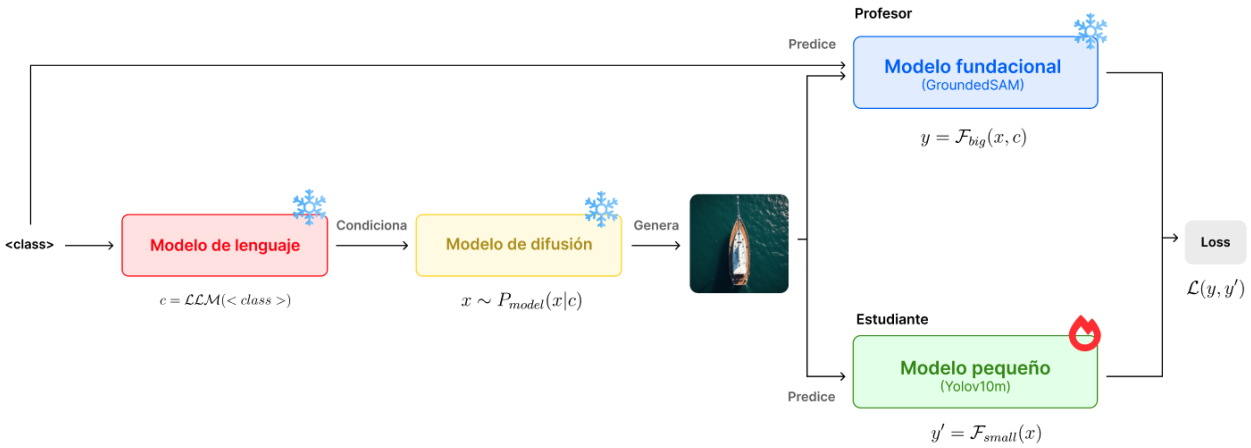


Ilustración 5.13: Diagrama conceptual del *pipeline 1*

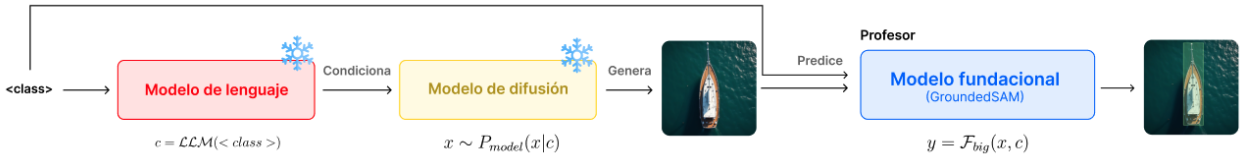


Ilustración 5.14: Diagrama de la generación de imágenes sintéticas en el *pipeline 1*

optimizado para su ejecución en dispositivos *edge*. Este proceso se puede visualizar en la figura 5.15.

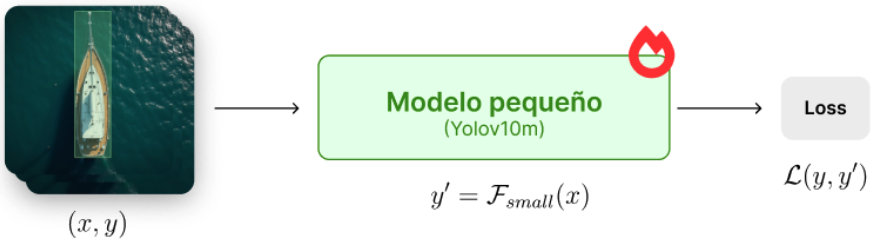


Ilustración 5.15: Diagrama de la fase de entrenamiento en el *pipeline 1*

Si bien este flujo puede, en teoría, implementarse de forma completamente automática, se optó por dividirlo en dos fases diferenciadas: generación y etiquetado de imágenes sintéticas, y posteriormente entrenamiento del modelo destino. Esta separación facilita el control del proceso y permite analizar de forma más precisa la calidad de las imágenes generadas y sus anotaciones antes del entrenamiento.

En los escenarios multiclase se identificaron limitaciones en los modelos fundacionales encargados del etiquetado, que detectaban correctamente categorías generales como *boat*, pero no otras más específicas como *cargoship*. Para solventarlo, se generaron imágenes usando *prompts* específicos (p. ej. *cargoship*) y se permitía que el modelo etiquetara como *boat*, reasignando manualmente esta etiqueta a *cargoship* al almacenarla. Así, se asumía que la imagen

generada correspondía con la categoría solicitada, mejorando la representación de clases poco frecuentes en el conjunto. Esto se puede observar en la figura 5.16.

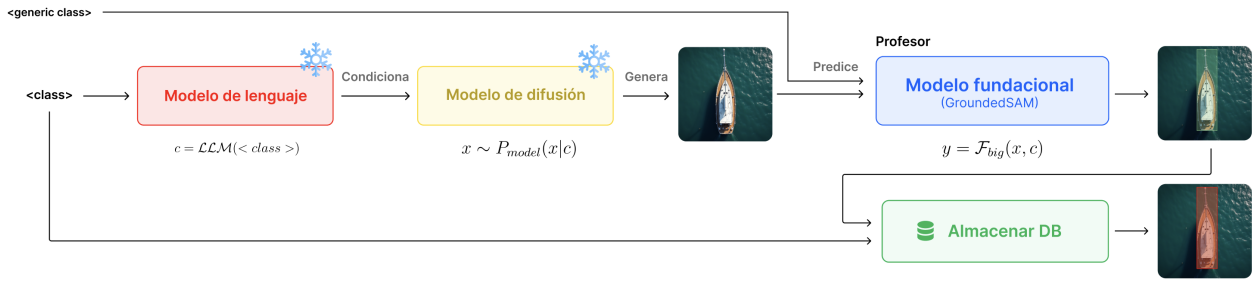


Ilustración 5.16: Diagrama de creación de *datasets* con múltiples categorías

Definición del conjunto de datos a crear

Para estructurar la generación de imágenes sintéticas de forma controlada, se definieron cuatro dimensiones clave que caracterizan cada escena: el tipo de embarcación, el entorno, la condición ambiental y la perspectiva de la cámara. Cada una de estas dimensiones se restringió a un conjunto específico de opciones, lo que permitió establecer una base clara sobre la que construir las descripciones textuales. La definición exacta de estas variables y sus valores posibles se detalla en el anexo 8.

Con este marco, el objetivo fue generar descripciones completas combinando estas variables, de forma que sirvieran como entrada textual al modelo generativo. Para automatizar este proceso, se utilizó ChatGPT con el modelo GPT4o, al que se le proporcionó una descripción del dominio y las variables disponibles. El modelo generó 90 *prompts* distintos: 10 por cada tipo de barco definido y 10 adicionales que mezclaban varios tipos en una misma escena. El texto utilizado para guiar esta generación se incluye en el anexo 8. Este proceso puede visualizarse de forma esquemática en la figura 5.17.

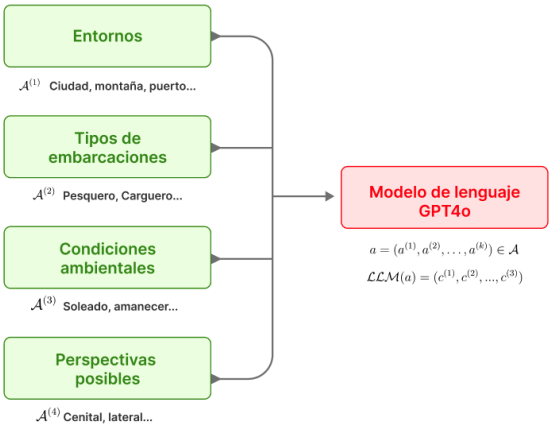


Ilustración 5.17: Diagrama del flujo de generación de *prompts*

Cada *prompt* fue empleado para generar 10 imágenes, lo que resultó en un total de 900 imágenes sintéticas. Finalmente, se verificó que la distribución de variables en el conjunto

resultante fuese razonablemente equilibrada para garantizar la diversidad y representatividad del dataset.

Generación de los datos

En la etapa de síntesis, se diseñó un flujo de trabajo en *ComfyUI* utilizando el modelo *Flux 1.1 [dev]* con los siguientes hiperparámetros: 30 *steps*, *scheduler* lineal, *guidance scale* de 7.5 y resolución de 1024x768 píxeles. Para automatizar el proceso, se configuró un flujo en *n8n* que se integraba con la API de *ComfyUI* y ejecutaba 10 inferencias por cada *prompt* generado. Cada una de las imágenes resultantes se almacenaba automáticamente en la plataforma *Datagen* (véase la sección 6).

Una vez sintetizadas todas las imágenes, se implementó un segundo flujo en *n8n* que, mediante la API de *Datagen*, etiquetaba cada imagen utilizando la clase *boat* con el modelo *Grounded SAM*. Para ello se uso el servicio de *autodistill* mencionado en la sección 4.

Como resultado final, se obtuvo un conjunto de imágenes generadas y etiquetadas de forma automática, similares a las mostradas en la figura 5.18.

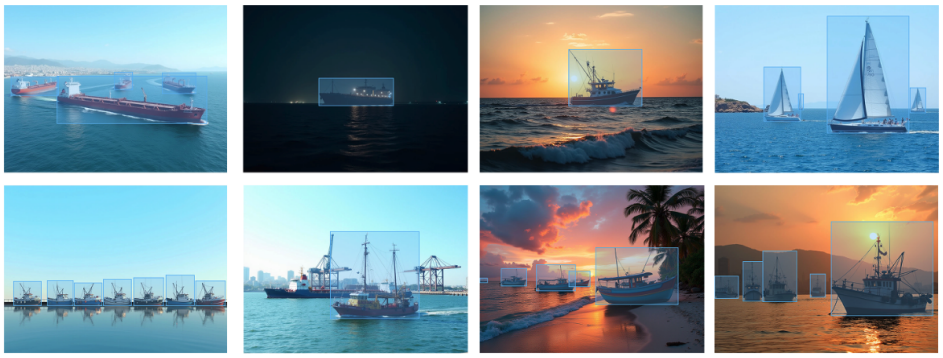


Ilustración 5.18: Ejemplos de muestras del dataset generado con el *pipeline* 1

Algoritmos

Podemos observar en el algoritmo 3 el pseudocódigo empleado para la generación del *dataset*.

Algoritmo 3 Generación y etiquetado de imágenes sintéticas de barcos

Require: Conjunto de **tipos de barco** $\mathcal{B}$,
conjunto de **entornos** $\mathcal{E}$,
conjunto de **condiciones ambientales** $\mathcal{C}$,
conjunto de **perspectivas** $\mathcal{P}$,
número de imágenes por *prompt* $n_{\text{img}} = 10$,
parámetros de síntesis: pasos $s = 30$, resolución $R = (1024, 768)$
Ensure: Conjunto de imágenes sintéticas D_{img} y sus anotaciones D_{ann}

$\mathcal{S} \leftarrow$ GenerarPromptsChatGPT$(\mathcal{B}, \mathcal{E}, \mathcal{C}, \mathcal{P})$ ▷ 90 *prompts*
$D_{\text{img}} \leftarrow \emptyset$; $D_{\text{ann}} \leftarrow \emptyset$
for all $prompt \in \mathcal{S}$ **do** ▷ Recorrer cada *prompt*
  for $i \leftarrow 1$ **hasta** n_{img} **do** ▷ 10 imágenes por *prompt*
    $img \leftarrow$ Flux($prompt$, R, s)
    if CalidadVisual(img) = **baja then** ▷ Filtro manual / automático
      continue
    end if
    $segmentations \leftarrow$ GroundedSAM(img, clase="boat")
    $boxes \leftarrow$ ExtractBoundingBoxes($segmentations$)
    if $boxes = \emptyset$ **then** ▷ No se detectó barco
      continue
    end if
    append(D_{img}, img); append(D_{ann}, $boxes$)
  end for
end for
return $D_{\text{img}}, D_{\text{ann}}$

Evaluación

Los resultados del *pipeline 1* muestran que el uso de imágenes sintéticas, aunque de menor calidad visual que las reales (según lo indica su FID de 59.45 para entrenamiento y 71.42 para validación), permite mejorar significativamente el rendimiento de modelos de detección cuando se combinan con datos reales.

En el escenario **monocategórico**, el entrenamiento con 900 imágenes reales alcanza una precisión base de **0.6807** en AP50 y **0.4451** en AP. Cuando se entrena exclusivamente con 900 imágenes sintéticas, el rendimiento cae notablemente, alcanzando solo **0.4771** en AP50 y **0.3051** en AP, lo que supone una reducción aproximada del **30 %**. Esto pone de manifiesto que, por sí solas, las imágenes generadas no son suficientes para igualar la calidad de entrenamiento proporcionada por datos reales.

Sin embargo, la combinación de datos reales con sintéticos sí resulta beneficiosa. Añadiendo 900 imágenes sintéticas a las 900 reales se logra una mejora de **+6.9 %** en AP50 y **+9 %** en AP, superando el rendimiento de la base original. Este efecto positivo es aún más evidente al aumentar la cantidad de datos sintéticos: con 3500 imágenes sintéticas y las mismas 900 reales se alcanza una mejora de **+8.8 %** en AP50 y **+14.6 %** en AP.

Este patrón también se repite en un escenario de escasez de datos reales. Con tan solo 400 imágenes reales, el rendimiento base es de **0.58** en AP50 y **0.372** en AP. Al añadir 900 imágenes sintéticas, el modelo mejora hasta **0.674** en AP50 y **0.457** en AP, lo que representa incrementos de **+16 %** y **+22 %** respectivamente. Esto evidencia que los datos sintéticos pueden ser especialmente útiles para compensar la falta de datos reales.

En el caso **multicategórico**, donde se incluyen múltiples clases, los beneficios también son claros. Con 1000 imágenes reales, el modelo alcanza **0.682** en AP50 y **0.461** en AP. Al añadir 2700 imágenes sintéticas, se obtiene una mejora hasta **0.714** en AP50 y **0.511** en AP, lo que implica un incremento relativo de **+5 %** y **+10.8 %** respectivamente. Esta mejora sugiere que el *pipeline 1* también es eficaz para enriquecer la representación multiclase, mejorando la cobertura del dominio sin requerir recolección adicional de datos reales.

Pipeline	**FID (entrenamiento)**	**FID (validación)**
Pipeline 1	59.45	71.42

Cuadro 5.1: Valores de FID entre imágenes reales y sintéticas generadas por el *pipeline* 1, tanto para el conjunto de entrenamiento como de validación.

Configuración monocategórica (Base: 900 reales)				
Configuración	**Reales**	**Sintéticas**	**AP50**	**AP**
Solo reales (base)	900	0	0.6807	0.4451
Solo sintéticas	0	900	0.4771 (-30 %)	0.3051 (-32 %)
Mixto 1:1	900	900	0.7280 (+6.9 %)	0.4856 (+9 %)
Mixto 1:4	900	3500	0.7400 (+8.8 %)	0.5100 (+14.6 %)
Configuración monocategórica (Base: 400 reales)				
Solo reales (base reducida)	400	0	0.5800	0.3720
400 reales + 900 sintéticas	400	900	0.6740 (+16 %)	0.4570 (+22 %)
Configuración multicategórica				
Solo reales (base)	1000	0	0.682	0.461
Mixto	1000	2700	0.714 (+5 %)	0.511 (+10.8 %)

Cuadro 5.2: Resultados de AP50 y AP para diferentes configuraciones de entrenamiento con el *pipeline 1*.

Clase	AP50 R	AP50 R+S	% Mejora	AP R	AP R+S	% Mejora
Total	0.682	0.714	+4.69 %	0.461	0.511	+10.85 %
Industrial	0.688	0.719	+4.51 %	0.435	0.490	+12.64 %
Passenger	0.828	0.835	+0.85 %	0.665	0.707	+6.32 %
Sailboat	0.845	0.843	-0.24 %	0.610	0.643	+5.41 %
Human P.	0.526	0.589	+11.98 %	0.294	0.369	+25.51 %
Motorboat	0.524	0.586	+11.83 %	0.299	0.345	+15.38 %

Cuadro 5.3: Comparación de AP50 y AP entre entrenamiento con solo imágenes reales y con reales + sintéticas.

5.4. Pipeline 2

El *pipeline 2* se basa en la generación de objetos mediante *inpainting* sobre fondos sintéticos. En primer lugar, se genera una imagen base con un modelo de difusión, asegurándose de que no contenga instancias de las clases objetivo. De esta forma se evita la aparición de objetos no etiquetados en las muestras finales. Este proceso se resume en la figura 5.19.

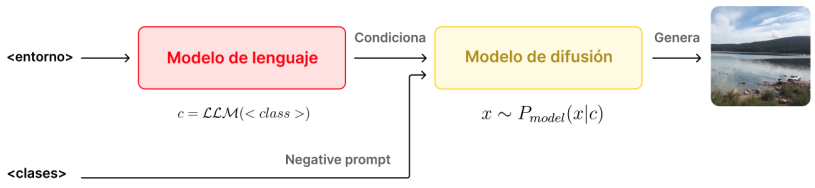


Ilustración 5.19: Diagrama de la generación de fondos en el *pipeline 2*

A continuación, se identifica la región de interés mediante segmentación —por ejemplo, el mar en el caso de generar barcos— y se divide la imagen en una cuadrícula uniforme de $n \times n$ celdas, donde n es un parámetro del método. Se seleccionan aquellas celdas cuyo solapamiento con la región segmentada supere un umbral μ, otro parámetro. A partir de estas celdas se construye la máscara de *inpainting*, como se ilustra en la figura 5.20.

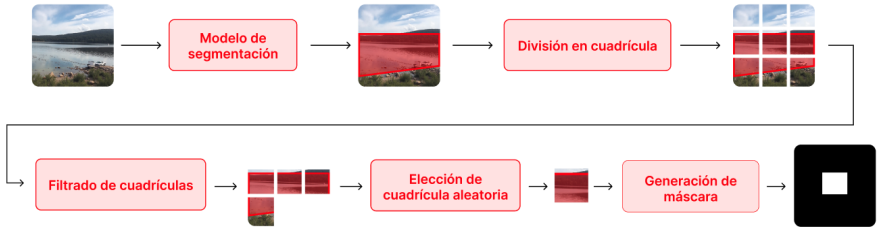


Ilustración 5.20: Diagrama de la generación de máscaras de *inpainting* en el *pipeline 2*

Luego, sobre una muestra de estas celdas se aplica *inpainting* para insertar los objetos deseados —como distintos tipos de embarcaciones— de forma coherente en la escena. Finalmente, se utiliza un modelo de segmentación como *SAM* para refinar la región de *inpainting* y extraer la etiqueta final con mayor precisión. Esto se puede observar en la figura 5.21

Definición del conjunto de datos

Aunque inicialmente se propuso generar fondos mediante *prompts* diseñados con ChatGPT, en el experimento se optó por una estrategia alternativa basada en *img2img* sobre imágenes reales del dataset original, con el objetivo de reducir la distancia FID del conjunto final.

Se seleccionaron 80 imágenes del conjunto de entrenamiento y se eliminaron los objetos presentes mediante *Lama inpainting*, obteniendo fondos limpios. A partir de estas imágenes, se generaron descripciones automáticas usando GPT-4o, que luego se combinaron con las 24 configuraciones de entorno y condiciones ambientales definidas previamente en el *pipeline 1*, dando lugar a 1920 *prompts*.

Estas descripciones se usaron para guiar a Flux 1.1 [dev] con LoRAs de Canny e InstructPix2Pix. Se aplicó *img2img* utilizando los bordes extraídos de las imágenes originales (reescaladas a 1024x768) como entrada. Se generaron 1108 imágenes, de las cuales se seleccionaron manualmente las 1000 más coherentes visualmente.

Los hiperparámetros utilizados fueron: 30 *steps*, *schedule* lineal, y *guidance scale* de 10

Ilustración 5.21: Diagrama de la extracción de etiquetas durante el *pipeline 2*

para el prompt y 1.2 para el *negative prompt*. Se pueden observar varios ejemplos de las imágenes resultantes en la figura 5.22.

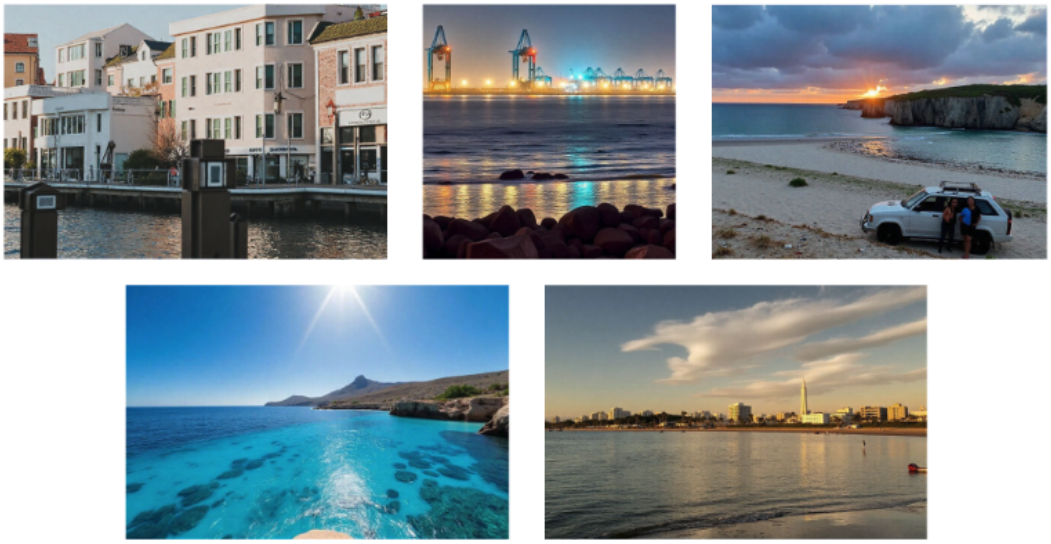


Ilustración 5.22: Imágenes resultantes de la generación de fondos del *pipeline 2*

Inserción de objetos y generación de etiquetas

Una vez generados los fondos, se insertaron instancias de barcos mediante un proceso controlado de *inpainting*. Como se describió anteriormente, se segmentó la región de interés (el mar) con *GroundedSAM*, se dividió la imagen en una cuadrícula de $n \times n$ celdas y se seleccionaron aquellas cuyo solapamiento con la región segmentada superaba un umbral $\mu = 0{,}5$.

Sobre una selección aleatoria de estas celdas se aplicó *inpainting* con el modelo *Flux 1.1 [dev]*, utilizando 40 *steps*, *guidance scale* de 40 y un *schedule* lineal. El *prompt* correspondía a una clase aleatoria del conjunto.

La máscara de *inpainting* se refinó con *SAM* para obtener la *bounding box*, y se validó con *CLIP* (*ViT-B/32*) conservando solo aquellas inserciones con una puntuación superior a 0.85 frente a las clases posibles (incluyendo 'mar" y 'other").

Para cada fondo, se probaban tres cuadrículas posibles: 2×2, 3×3 y 4×4, eligiendo aleatoriamente entre $n-1$ y n celdas para realizar las inserciones. Este procedimiento generó un total de 1300 imágenes.

Las imágenes generadas mostraron una notable variedad. En algunos casos, se obtuvieron resultados de alta calidad visual, con barcos bien integrados y etiquetas refinadas, como se observa en la figura 5.23.

Sin embargo, también se produjeron errores de síntesis. En ocasiones, la máscara de *inpainting* resultaba excesiva para el objeto generado, o el fondo carecía de la calidad suficiente para una integración coherente. Aunque el filtro con *CLIP* eliminó muchos de estos casos, algunos lograron superar el umbral de validación, como se aprecia en la figura 5.24. Por ello,

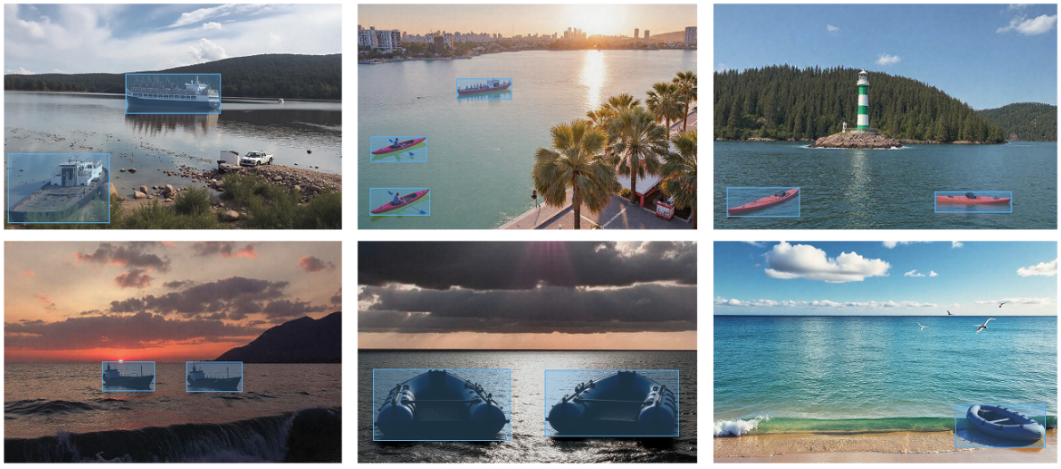


Ilustración 5.23: Ejemplos de imágenes del dataset formado con *inpainting*

se aplicó un segundo filtrado manual para garantizar la calidad, conservando finalmente las 1000 imágenes originales.

Ilustración 5.24: Imágenes sintetizadas incorrectamente utilizando *inpainting*

Algoritmos

Podemos observar a continuación los dos algoritmos que constituyen este *pipeline*, el primero de generación de fondos sintéticos sin barcos 4 y segundo la inserción de barcos en dichos fondos 5.

Algoritmo 4 Generación de fondos sintéticos sin barcos

Require: Conjunto de imágenes originales D_{orig} (80 fotografías),
lista de **entornos** E (Tabla 8.1),
lista de **condiciones climatológicas** C (Tabla 8.3),
lista de clases negativas N (todas las clases del *dataset*),
resolución objetivo $R = (1024, 768)$
Ensure: Conjunto filtrado de fondos sintéticos D_{fondos} (1 000 imágenes)
$D_{\text{fondos}} \leftarrow \emptyset$
for all $img_{\text{orig}} \in D_{\text{orig}}$ **do** ▷ Recorrer las 80 imágenes
    $img_r \leftarrow \text{RESIZE}(img_{\text{orig}}, R)$
    $img_{\text{clean}} \leftarrow \text{LAMAINPAINTING}(img_r)$ ▷ Eliminar objetos
    $bordes \leftarrow \text{CANNY}(img_{\text{clean}})$
    $caption_{\text{base}} \leftarrow \text{GPT4OCAPTION}(img_{\text{clean}})$
    for all $ent \in E$ **do** ▷ Primero: variar el entorno
        $caption_{ent} \leftarrow \text{GPT4OVARIATION}(caption_{\text{base}}, ent)$
        for all $cli \in C$ **do** ▷ Después: variar el clima
            $caption_{ent,cli} \leftarrow \text{GPT4OVARIATION}(caption_{ent}, cli)$
            $img_{\text{gen}} \leftarrow \text{FLUXGENERATE}(bordes, caption_{ent,cli}, N)$
            $\text{APPEND}(D_{\text{fondos}}, img_{\text{gen}})$
        end for
    end for
end for
$D_{\text{fondos}} \leftarrow \text{MANUALFILTER}(D_{\text{fondos}}, 1000)$

Algoritmo 5 Inserción de barcos y etiquetado automático

Require: Fondos sintéticos D_{fondos},
categorías de barcos $\mathcal{C}$ (Tabla 8.2),
conjunto de divisiones $\mathcal{D} = \{2, 3, 4\}$ (mallas 2×2, 3×3, 4×4),
umbral de solapamiento μ,
umbral CLIP $\tau = 0.85$
Ensure: Imágenes con barcos sintetizados y *bounding boxes* válidas
for all $img_{\text{fondo}} \in D_{\text{fondos}}$ **do**
  $mask_{\text{mar}} \leftarrow \text{GROUNDEDSAM}(img_{\text{fondo}})$
  for all $d \in \mathcal{D}$ **do** ▷ Tres tamaños de malla
    $grid \leftarrow \text{CREARMALLA}(img_{\text{fondo}}, d)$
    $celdas_{\text{válidas}} \leftarrow \{c \mid \text{IOU}(c, mask_{\text{mar}}) \geq \mu\}$
    $k \leftarrow \text{RANDOMINT}(d-1,\ d)$ ▷ Número de barcos
    $celdas_{\text{obj}} \leftarrow \text{RANDOMSAMPLE}(celdas_{\text{válidas}}, k)$
    $img_{\text{cur}} \leftarrow img_{\text{fondo}}$; $bboxes \leftarrow [\,]$
    for all $c \in celdas_{\text{obj}}$ **do**
      $cat \leftarrow \text{RANDOMCHOICE}(\mathcal{C})$ ▷ Elegir clase (Tabla 8.2)
      $prompt \leftarrow$ "*a single alone* " $+ cat$
      $img_{\text{inpaint}} \leftarrow \text{COMFYGENERATOR}(img_{\text{cur}}, c.\text{mask}, prompt)$
      $bbox_{\text{ref}} \leftarrow \text{REFINEBBOX}(img_{\text{inpaint}}, c.\text{bbox})$
      $crop \leftarrow \text{CROP}(img_{\text{inpaint}}, bbox_{\text{ref}})$
      $score \leftarrow \text{CLIP}(crop, [cat, \text{"}boat\text{"}, \text{"}sea\text{"}, \text{"}other\text{"}])$
      if $score_{cat} \geq \tau$ **then** ▷ Validación semántica
        $img_{\text{cur}} \leftarrow img_{\text{inpaint}}$
        APPEND($bboxes$, $bbox_{\text{ref}}$)
      end if
    end for
    if LEN($bboxes$) $= 0$ **then continue**
    end if ▷ Descartar inserción vacía
    $id \leftarrow \text{UPLOADIMAGE}(img_{\text{cur}}, 17)$
    UPLOADBBOXES(id, $bboxes$, label=7)
  end for
end for

Evaluación

Los resultados del *pipeline 2*, aunque globalmente inferiores a los obtenidos con el *pipeline 1*, siguen siendo positivos en ciertos contextos. A nivel visual, las imágenes generadas presentan un realismo cualitativo superior, con una iluminación más natural y una apariencia menos artificial. Sin embargo, esto no se traduce en una mayor similitud estadística con los datos reales, como reflejan los valores de *Fréchet Inception Distance (FID)*, que son significativamente más altos: **79.07** en entrenamiento y **91.90** en validación, lo que representa un incremento del **33 %** y **28 %**, respectivamente, respecto al *pipeline 1*.

Desde el punto de vista del rendimiento en detección, entrenar únicamente con 900 imáge-

nes sintéticas generadas por este *pipeline* conduce a un resultado muy deficiente: **0.2016** en AP50 y **0.0939** en AP, con caídas del **71 %** y **79 %** frente al modelo entrenado exclusivamente con imágenes reales. Este descenso notable sugiere que, en solitario, los datos sintéticos producidos no aportan suficiente valor como para reemplazar a los datos reales.

No obstante, al combinar estas 900 imágenes sintéticas con otras 900 reales, se observa una mejora modesta pero consistente: se alcanzan **0.7042** en AP50 y **0.4640** en AP, lo que supone incrementos de **+3 %** y **+4.2 %** sobre la base original. Estos resultados demuestran que, si bien la calidad estructural de las imágenes puede no ser óptima, su inclusión como complemento en el entrenamiento sí contribuye a una mejora del modelo.

Además, en escenarios con pocos datos reales (por ejemplo, 400 imágenes), añadir 900 sintéticas también ofrece beneficios: incrementa de **0.58** a **0.612** en AP50, y de **0.372** a **0.394** en AP, lo que representa mejoras del **+5.5 %** y **+6 %**, respectivamente. Aunque estas ganancias son inferiores a las observadas con otros métodos

Pipeline	**FID (entrenamiento)**	**FID (validación)**
Pipeline 2	79.07 (+33 % respecto al *pipeline 1*)	91.90 (+28 % respecto al *pipeline 1*)

Cuadro 5.4: Valores de FID entre imágenes reales y sintéticas generadas por los distintos *pipelines*.

Configuración monocategórica (Base: 900 reales)				
Configuración	**Reales**	**Sintéticas**	**AP50**	**AP**
Solo reales (base)	900	0	0.6807	0.4451
Solo sintéticas	0	900	0.2016 (-71 %)	0.0939 (-79 %)
Mixto 1:1	900	900	0.7042 (+3 %)	0.4640 (+4.2 %)
Configuración monocategórica (Base: 400 reales)				
Solo reales (base reducida)	400	0	0.5800	0.3720
400 reales + 900 sintéticas	400	900	0.6120 (+5.5 %)	0.3940 (+6 %)

Cuadro 5.5: Resultados de AP50 y AP para diferentes configuraciones de entrenamiento con el *pipeline 2*.

Limitaciones

Este enfoque presenta varias limitaciones. En primer lugar, el etiquetado automático puede fallar cuando se generan múltiples objetos en una misma región de *inpainting*, o cuando aparecen elementos adicionales como reflejos, diques u otros objetos marinos junto al barco. Además, el tiempo de ejecución aumenta con el número de objetos a posicionar, y aunque el uso de un esquema en cuadrícula permite paralelizar el proceso, esto requiere mayor capacidad computacional respecto al método anterior.

También se depende de un modelo externo (como SAM) para refinar las *bounding boxes*, lo que introduce una posible fuente de error. Por último, en imágenes con fondos de baja calidad, puede generarse contenido incorrecto (por ejemplo, objetos que no son barcos). Aunque se

aplica un filtro basado en CLIP para mitigar estos casos, no siempre resulta completamente fiable.

5.5. Pipeline 3

El *pipeline 3* se basa en el *clustering* no supervisado de los mapas de atención cruzada extraídos de la U-Net del modelo de difusión. Para ello, se calculan los mapas de atención promedio por *token*, como se describe en la sección 5.1.

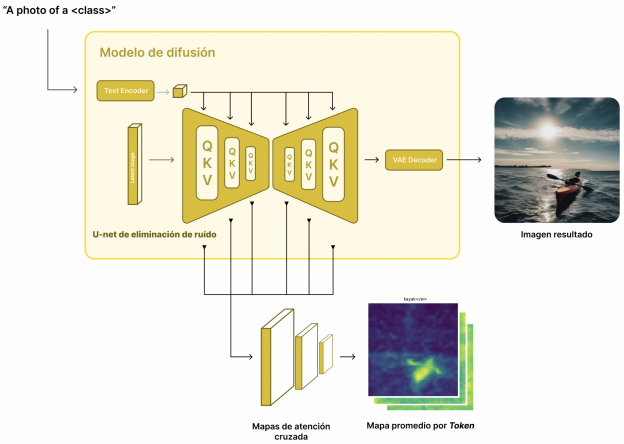


Ilustración 5.25: Extracción de mapas de atención cruzada promedio en el *pipeline 3*

A continuación, en caso de que la palabra objetiva esté compuesta por varios *tokens*, se promedia también entre ellos. Posteriormente, se aplica una corrección *gamma* para resaltar las regiones con mayor activación y, posteriormente, se ejecuta un algoritmo de *k-means* con dos *clusters* para separar objeto y fondo. Este proceso de binarización se puede observar en la figura 5.26.

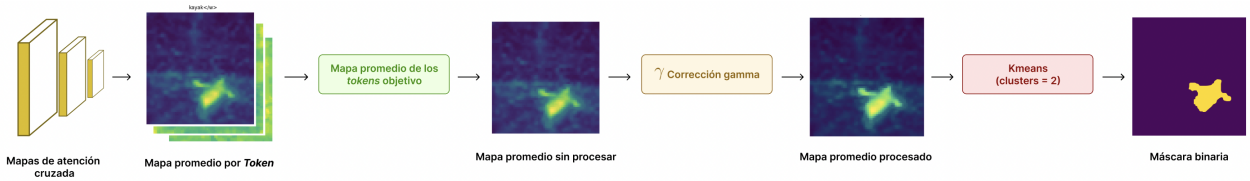


Ilustración 5.26: Binarización de los mapas de atención cruzada en el *pipeline 3*

La segmentación resultante se convierte en una máscara binaria, a partir de la cual se identifican regiones conectadas de píxeles que permiten extraer las etiquetas correspondientes. Finalmente, se utiliza SAM para refinar la máscara obtenida. Cuyo proceso se define en la figura 5.27

Definición del conjunto de datos

Para construir el conjunto de datos, se empleó GPT4o como generador de descripciones textuales. A diferencia del *pipeline 1*, este enfoque introdujo dos cambios clave:

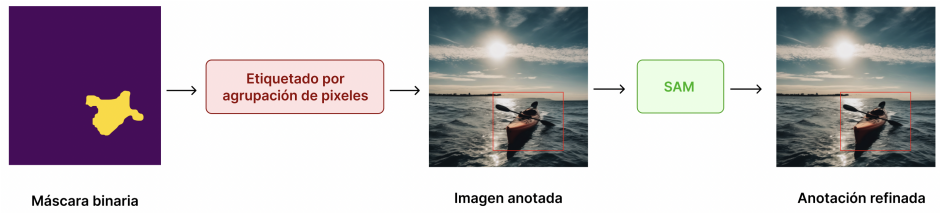


Ilustración 5.27: Extracción de la máscara refinada en el *pipeline 3*

1. Las imágenes generadas debían contener únicamente una instancia de la clase objetivo, dado que el método de extracción de etiquetas no admitía múltiples objetos por imagen.
2. Se simplificó la generación de *prompts*, omitiendo detalles como entorno, condiciones climáticas y perspectiva, dejando estos aspectos abiertos a la interpretación del modelo de lenguaje.

Bajo estas condiciones, se generaron *160 prompts* para cada una de las siguientes clases: *'cargoship', 'sailboat', 'yacht', 'motorboat', 'fishing boat', 'inflatable boat'* y *'vessel boat'*, sumando un total de *1120* descripciones. La generación se realizó mediante la API de ChatGPT con el modelo *gpt-4*, utilizando el siguiente *prompt* de instrucción:

```
Create a vivid, one-phrase prompt starting with 'A single ...' for Stable
Diffusion. Do not split compound words like 'cargoship'. Emphasize the
object's size (small, big, huge), its distance (far away, near), dominant
colors if logical, and its detailed environment (e.g., port, open sea,
beach, lake, industrial dock, etc.).
```

Generación de imágenes

Para la generación de imágenes se empleó un modelo *fine-tuned* de *SDXL*, concretamente la variante *epicrealismxl*, utilizando la librería *Diffusers*. La configuración utilizada incluyó 20 *steps*, un *guidance scale* de 7.5 y el planificador (*scheduler*) *Euler*. Además, para la extracción de mapas de atención cruzada se integró la librería *attention-map-diffusers*.

Generación de etiquetas mediante mapas de atención

Una vez generadas las imágenes, se extrajo la segmentación a partir de los mapas de atención cruzada (*cross-attention*) de la U-Net. Para ello, se aplicó primero una corrección gamma ($\gamma = 2{,}0$) para enfatizar las regiones más activas, seguida de un algoritmo de *K-Means* con dos clústeres para separar objeto y fondo. A partir de la máscara binaria generada, se identificaron las regiones correspondientes a los objetos mediante componentes conexas (`label` de *scipy*), y se refinaron utilizando *Segment Anything (SAM)*. La figura 5.28 muestra algunos ejemplos del resultado final.

Algoritmos

Podemos observar en el algoritmo 6 el pseudocódigo del método desarrollado.

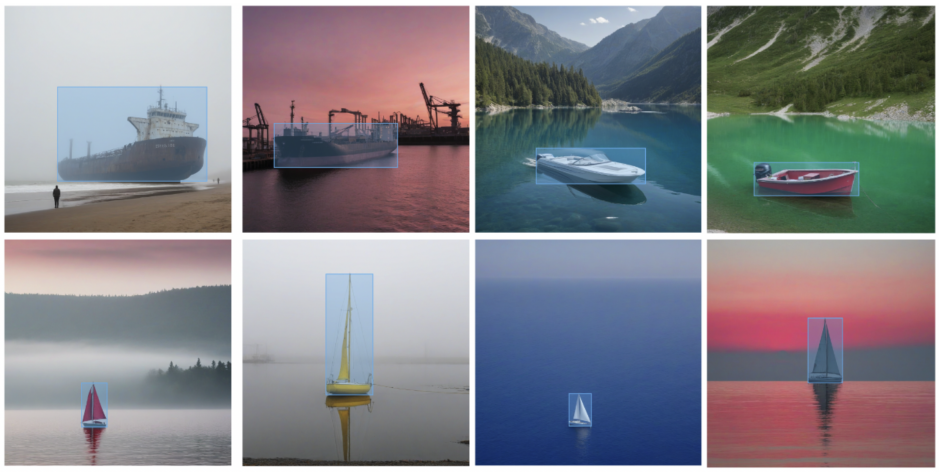


Ilustración 5.28: Ejemplos de imágenes generadas con sus respectivas etiquetas extraídas mediante atención cruzada y post-procesamiento.

Evaluación

Tal y como reflejan las Tablas 5.6 y 5.7, el *pipeline 3* ofrece un rendimiento intermedio entre las propuestas anteriores. A nivel visual, las imágenes generadas presentan una fidelidad aceptable, aunque inferior a la del *pipeline 1*, con valores de FID de **75.57** en entrenamiento y **87.60** en validación. Estos valores suponen un incremento del **27 %** y **22 %**, respectivamente, respecto al *pipeline 1*, pero siguen siendo mejores que los registrados por el *pipeline 2*.

En cuanto al rendimiento en tareas de detección, entrenar únicamente con 900 imágenes sintéticas produce una fuerte degradación en la precisión del modelo: se alcanza un **AP50** de **0.3094** y un **AP** de **0.1960**, lo que representa una caída del **45 %** y **44 %**, respectivamente, respecto al entrenamiento con 900 imágenes reales. A pesar de ello, estos resultados siguen siendo superiores a los obtenidos con los datos sintéticos del *pipeline 2*, lo que indica una mayor utilidad del conjunto generado.

La combinación de imágenes sintéticas con datos reales vuelve a demostrar ser una estrategia efectiva. Con 900 imágenes reales y 900 sintéticas, se alcanzan valores de **0.7083** en AP50 y **0.4684** en AP, lo que representa mejoras del **+4.05 %** y **+5.2 %** sobre el modelo entrenado únicamente con datos reales. Además, en situaciones de escasez de datos, como el caso con solo 400 imágenes reales, la inclusión de 900 imágenes sintéticas permite mejorar el rendimiento desde **0.58** a **0.615** en AP50 y de **0.372** a **0.401** en AP, lo que supone aumentos del **+6 %** y **+7.7 %**, respectivamente.

En conjunto, estos resultados consolidan al *pipeline 3* como una alternativa válida para el aumento de datos, especialmente cuando se busca un equilibrio entre calidad visual y aporte efectivo al rendimiento del modelo.

Algoritmo 6 Generación de imágenes y extracción automática de *bounding boxes*

Require: Lista de **clases** $\mathcal{B} = \{cargoship, sailboat, yatch, \dots\}$,
número de **prompts** por clase $n_p = 160$ (total 1 120),
modelo generativo *Diffusers* G (*epicrealismxl*),
extractor de mapas de atención E,
función RefineBBox basada en SAM,
área mínima aceptable $A_{\text{mín}}$,
dataset de destino con `id`=19
Ensure: Conjunto de imágenes D_{img} y sus anotaciones D_{ann}

$D_{\text{img}} \leftarrow \emptyset$; $D_{\text{ann}} \leftarrow \emptyset$
for all $label \in \mathcal{B}$ **do** ▷ Una clase cada vez
  for $i \leftarrow 1$ **hasta** n_p **do** ▷ 160 *prompts*
    $prompt \leftarrow$ ChatGPT("A single `label`...")
    $img \leftarrow G$.generate($prompt$, steps= 20)
    $A \leftarrow E$.compute($prompt$, G) ▷ Mapas de atención
    $M \leftarrow$ AverageCrossAttention(A, $label$) ▷ Ecuación (1)
    $M' \leftarrow$ Gamma(M, 2,0) ▷ Corrección gamma
    $mask \leftarrow$ KMeansSegmentation(M', $k = 2$)
    $contours \leftarrow$ label($mask$)
    $bboxes \leftarrow \emptyset$
    for all $c \in contours$ **do**
      $box \leftarrow$ BoundingRect(c)
      if Area(box) $< A_{\text{mín}}$ **then continue**
      end if
      $box \leftarrow$ RefineBBox(img, box) ▷ SAM
      append($bboxes$, box)
    end for
    if $bboxes = \emptyset$ **then continue**
    end if ▷ Descartar imagen vacía
    append(D_{img}, img); append(D_{ann}, $bboxes$)
  end for
end for
return $D_{\text{img}}, D_{\text{ann}}$

Pipeline	FID (entrenamiento)	FID (validación)
Pipeline 3	75.57 (+27 % respecto al *pipeline 1*)	87.60 (+22 % respecto al *pipeline 1*)

Cuadro 5.6: Valores de FID entre imágenes reales y sintéticas generadas por los distintos *pipelines*.

5.5.0.1. Resultados generales

Los resultados obtenidos a lo largo de los distintos experimentos ponen de manifiesto la utilidad del uso de datos sintéticos como técnica de aumento en tareas de detección de

Configuración monocategórica (Base: 900 reales)				
Configuración	**Reales**	**Sintéticas**	**AP50**	**AP**
Solo reales (base)	900	0	0.6807	0.4451
Solo sintéticas	0	900	0.3094 (-45 %)	0.1960 (-44 %)
Mixto 1:1	900	900	0.7083 (+4.05 %)	0.4684 (+5.2 %)
Configuración monocategórica (Base: 400 reales)				
Solo reales (base reducida)	400	0	0.5800	0.3720
400 reales + 900 sintéticas	400	900	0.6150 (+6 %)	0.4010 (+7.7 %)

Cuadro 5.7: Resultados de AP50 y AP para diferentes configuraciones de entrenamiento con el *pipeline 3*.

objetos. Los resultados globales de todos los experimentos se pueden observar en la tabla 5.8. En términos generales, se pueden extraer las siguientes conclusiones de dicha información:

1. El entrenamiento exclusivamente con imágenes sintéticas ofrece un rendimiento claramente inferior al entrenamiento con datos reales, con caídas de hasta un **71 %** en AP50 (*pipeline* 2). No obstante, la severidad de esta degradación varía significativamente entre *pipelines*, siendo menos pronunciada en el caso del *pipeline* 1.

2. La combinación de datos reales y sintéticos es beneficiosa en todos los escenarios. Conjuntos mixtos permiten superar sistemáticamente el rendimiento obtenido con datos reales en solitario, con incrementos de hasta **+14.6 %** en AP (*pipeline* 1). Esto confirma que los datos sintéticos enriquecen el espacio de representación del modelo sin degradar la calidad del entrenamiento.

3. El volumen de imágenes sintéticas importa: cuanto mayor es la proporción de sintéticas añadidas, mayor es el incremento en el rendimiento (siempre que estén combinadas con datos reales). Este efecto se hace más evidente en entornos con escasez de datos reales, donde la ganancia relativa puede superar el **+20 %**.

4. Comparando entre *pipelines*, el ***pipeline* 1** ofrece el mejor balance entre realismo (FID) y utilidad para el entrenamiento. Aunque sus imágenes presentan menor calidad visual que las del *pipeline* 2, su impacto positivo en la precisión del modelo es superior, especialmente en configuraciones reducidas. El ***pipeline 3*** se posiciona como una alternativa intermedia, mejorando en FID respecto al *pipeline* 2 y en rendimiento respecto a entrenamientos únicamente sintéticos.

5. Finalmente, estos resultados refuerzan la idea de que los datos sintéticos no deben verse como un sustituto completo de los reales, sino como un complemento estratégico para reforzar el aprendizaje en escenarios de baja disponibilidad de datos, mejorar la generalización del modelo y acelerar el desarrollo de sistemas de visión artificial robustos.

Pipeline 1				
Configuración monocategórica (Base: 900 reales)				
Entrenamiento	**Reales**	**Sintéticas**	**AP50**	**AP**
Base reales	900	-	0.6807	0.4451
Solo sintéticas	-	900	0.4771 (-30 %)	0.3051 (-32 %)
Reales + 900 sintéticas	900	900	0.7280 (+6.9 %)	0.4856 (+9 %)
Reales + 3500 sintéticas	900	3500	0.7400 (+8.8 %)	0.5100 (+14.6 %)
Configuración monocategórica (Base: 400 reales)				
Base reales	400	-	0.5800	0.3720
Reales + 900 sintéticas	400	900	0.6740 (+16 %)	0.4570 (+22 %)
Configuración multicategórica				
Base reales	1000	-	0.6820	0.4610
Reales + 2700 sintéticas	1000	2700	0.7140 (+5 %)	0.5110 (+10.8 %)
Pipeline 2				
Configuración monocategórica (Base: 900 reales)				
Base reales	900	-	0.6807	0.4451
Solo sintéticas	-	900	0.2016 (-71 %)	0.0939 (-79 %)
Reales + 900 sintéticas	900	900	0.7042 (+3 %)	0.4640 (+4.2 %)
Configuración monocategórica (Base: 400 reales)				
Base reales	400	-	0.5800	0.3720
Reales + 900 sintéticas	400	900	0.6120 (+5.5 %)	0.3940 (+6 %)
Pipeline 3				
Configuración monocategórica (Base: 900 reales)				
Base reales	900	-	0.6807	0.4451
Solo sintéticas	-	900	0.3094 (-45 %)	0.1960 (-44 %)
Reales + 900 sintéticas	900	900	0.7083 (+4.1 %)	0.4684 (+5.2 %)
Configuración monocategórica (Base: 400 reales)				
Base reales	400	-	0.5800	0.3720
Reales + 900 sintéticas	400	900	0.6150 (+6 %)	0.4010 (+7.7 %)

Cuadro 5.8: Resumen global de resultados de los tres pipelines con métricas de AP50 y AP.

Capítulo 6

Plataforma *Web*

6.1. Funcionalidades desarrolladas

6.1.1. Gestión de *datasets*

La plataforma permite crear y eliminar *datasets*, asignándoles nombre, descripción y color identificativo. En el listado de conjuntos disponibles, se muestra una previsualización con una imagen etiquetada, junto con información clave como las etiquetas definidas, número total de imágenes, número de anotaciones y tamaño en disco. Esta vista facilita la gestión y supervisión de múltiples *datasets* de forma rápida y visual. Podemos observar el listado de múltiples conjuntos de datos en la figura 6.1.

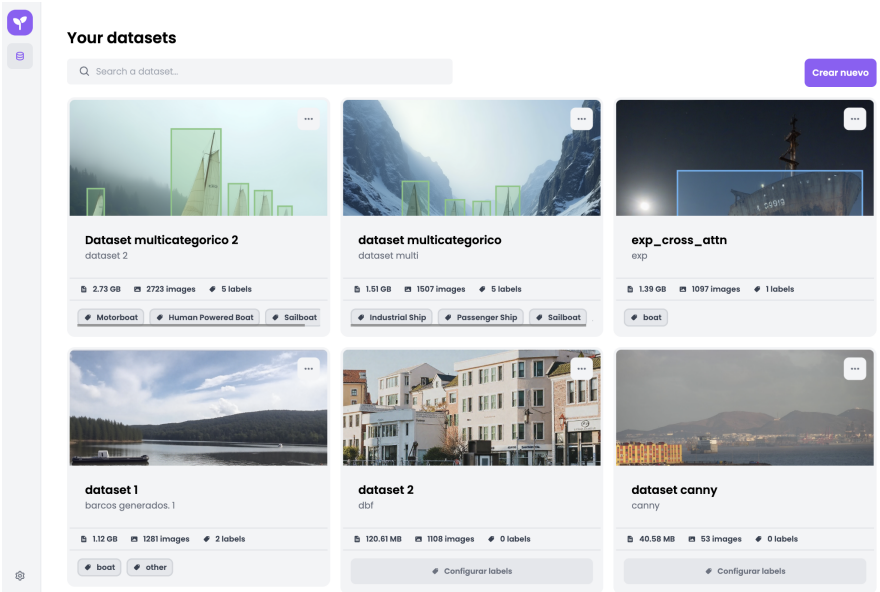


Ilustración 6.1: Listado de conjuntos de datos en Datagen

6.1.2. Subida de imágenes

Cada dataset admite la carga de imágenes mediante ficheros comprimidos en formato `.zip`. El sistema permite incluir en estos archivos imágenes en distintos formatos, como JPEG o PNG, que son extraídas e incorporadas automáticamente al dataset correspondiente. Esta funcionalidad facilita la carga masiva y estructurada del material visual que será utilizado en los procesos posteriores. Podemos observar la vista utilizada para esta funcionalidad en la figura 6.2.

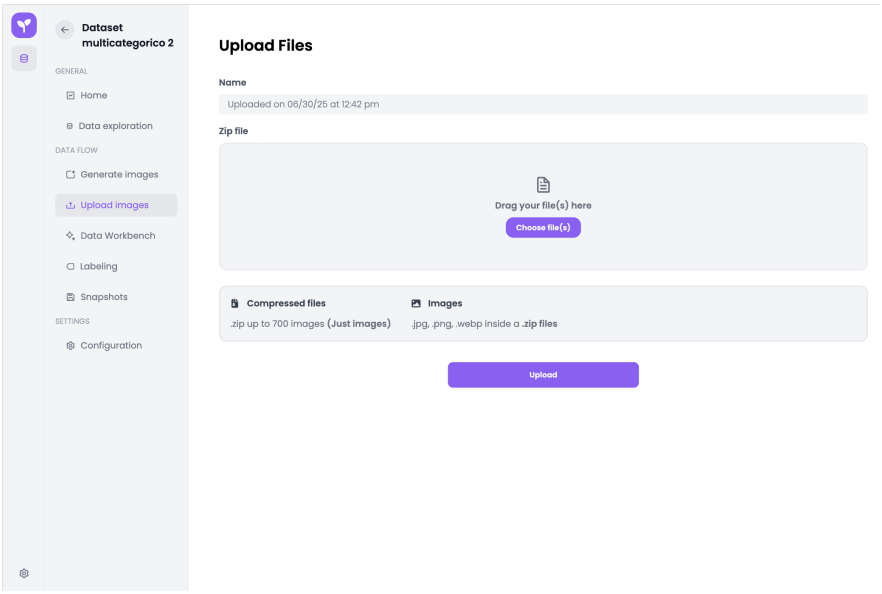


Ilustración 6.2: Ventana de subida de imágenes en Datagen

6.1.3. Exploración y filtrado de imágenes

Una vez subidas, las imágenes pueden explorarse a través de una vista que muestra miniaturas junto con las etiquetas asignadas a cada una. La interfaz permite seleccionar múltiples imágenes simultáneamente y realizar operaciones en bloque, como su eliminación, lo que agiliza significativamente la limpieza de datos generados de forma automática o con errores. Se puede observar un ejemplo con selección múltiple en la figura 6.3.

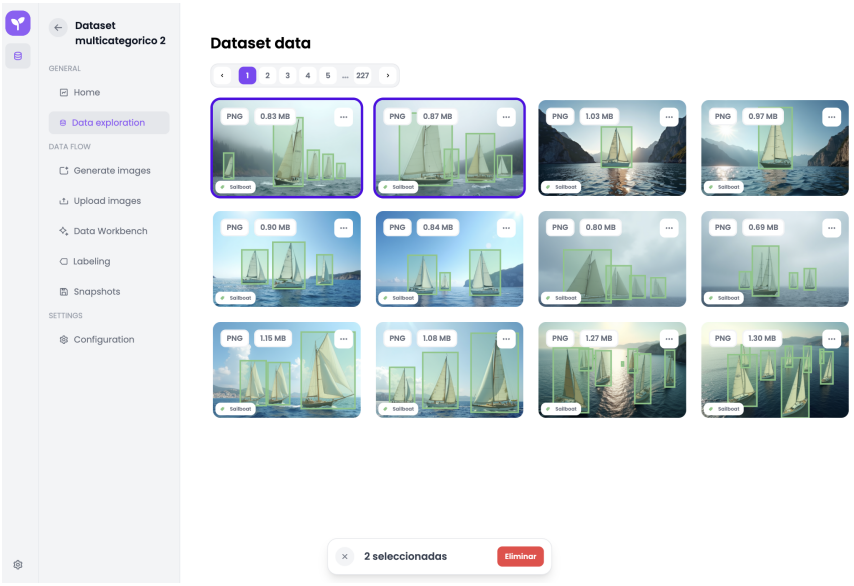


Ilustración 6.3: Listado de imágenes de un *dataset* en Datagen

6.1.4. Configuración de etiquetas

El sistema ofrece una sección específica para la creación, modificación y gestión de etiquetas asociadas a cada dataset. Se puede definir el nombre de cada clase, establecer un color, y asignarles un identificador numérico que será utilizado para asociar cada etiqueta a una anotación a la hora de exportar el *dataset* en formato ultralytics. Se pueden observar las etiquetas de un conjunto de datos en la figura 6.4.

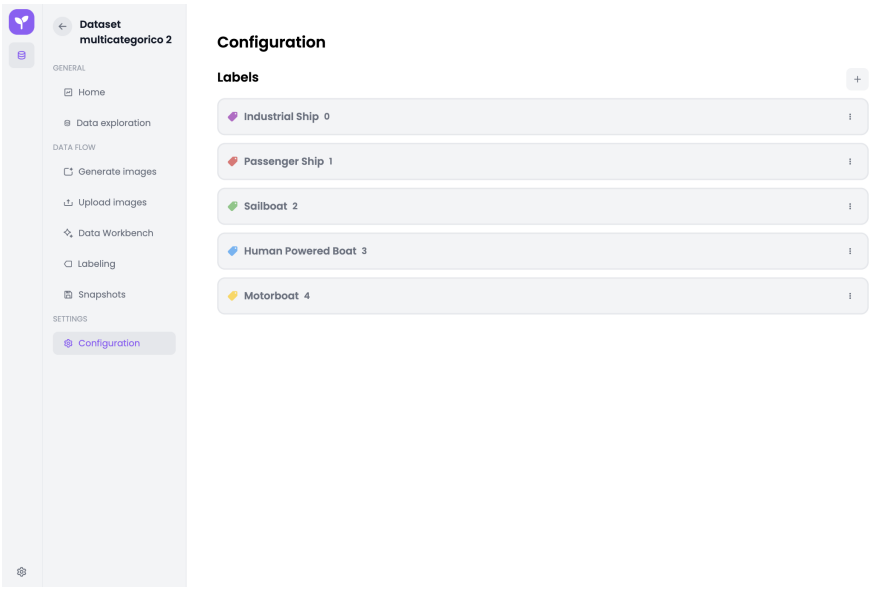


Ilustración 6.4: Configuración de etiquetas de un *dataset* en Datagen

6.1.5. Control de versiones y exportación

La plataforma permite capturar el estado de un dataset mediante snapshots, es decir, versiones congeladas que pueden ser almacenadas, consultadas y descargadas posteriormente. Estas versiones se pueden exportar en el formato utilizado por la librería Ultralytics, estructurando automáticamente las carpetas `train/images` y `train/labels`, y generando los archivos de anotaciones con el formato requerido (clase, *bounding box* normalizada). Esta funcionalidad se puede observar en la figura 6.5.

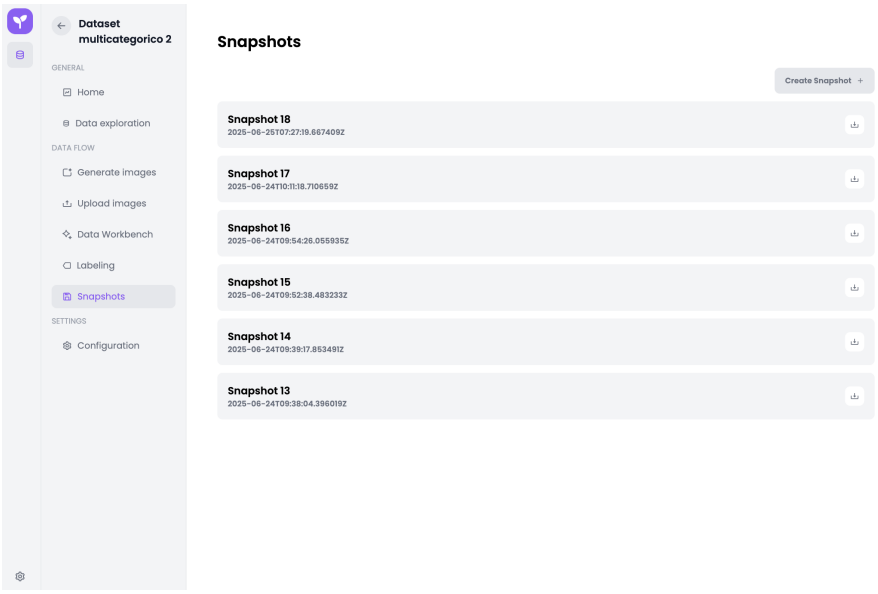


Ilustración 6.5: Creación de múltiples versiones de un mismo *dataset* en Datagen

6.1.6. Re-etiquetado y visualización

Para revisar y corregir manualmente las anotaciones, se incluye un canvas interactivo que permite dibujar, modificar y eliminar *bounding boxes* directamente sobre las imágenes. Las cajas pueden cambiarse de tamaño o clase sin necesidad de recrearlas, y se ofrece soporte para navegación (desplazamiento y zoom) dentro de la imagen. Esta herramienta facilita un re-etiquetado eficiente de los datos, especialmente útil tras procesos automáticos de anotación que requieran validación humana. Esta funcionalidad puede observarse en la figura 6.6.

6.2. Modelo de datos

A continuación se describen las entidades principales que conforman el sistema:

1. **Dataset:** Representa un conjunto de datos agrupados.
   a) `id` : Identificador único.

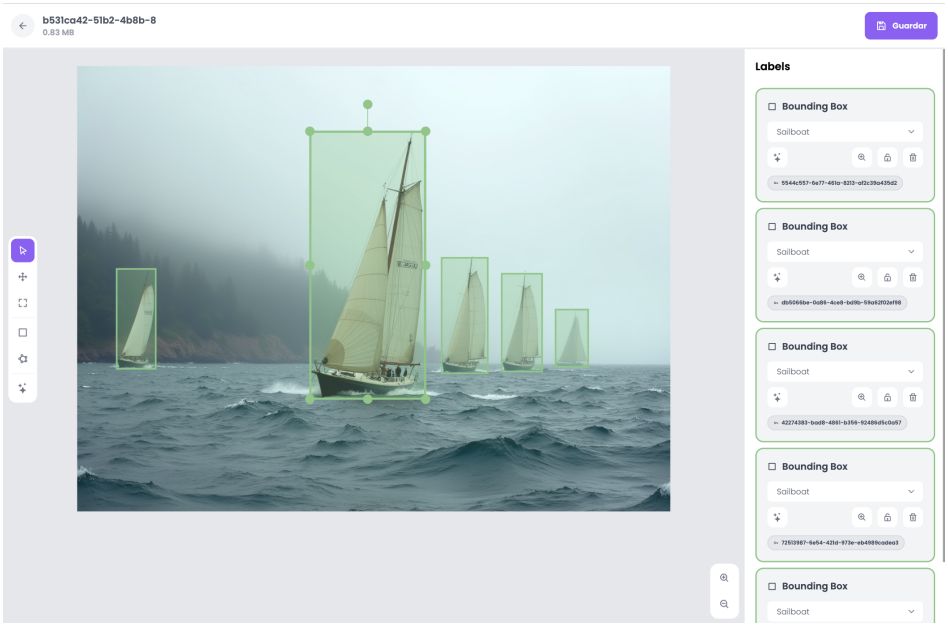


Ilustración 6.6: *Canvas* de re-etiquetado y visualización de imágenes

b) `user`: Usuario propietario del dataset.

c) `name`: Nombre descriptivo.

d) `description`: Descripción del dataset.

2. **Label:** Etiqueta o clase utilizada para clasificar elementos en un dataset.

a) `id`: Identificador único.

b) `dataset`: Dataset asociado.

c) `name`: Nombre de la etiqueta.

d) `color`: Color representativo de la etiqueta.

3. **DatasetImage:** Imagen perteneciente a un dataset.

a) `id` : Identificador único.

b) `dataset`: Dataset asociado.

c) `image`: Archivo de imagen.

d) `is_synthetic`: Indica si la imagen es sintética.

e) `job`: Trabajo asociado (opcional).

f) `batch`: Lote de datos al que pertenece (opcional).

g) `done`: Indica si la imagen fue procesada.

h) `reviewed`: Indica si la imagen fue revisada.

4. **Annotation:** Anotación aplicada a una imagen.
   a) `id (UUID)`: Identificador único.
   b) `type`: Tipo de anotación (por ejemplo, bounding box, segmentación).
   c) `label`: Etiqueta asociada.
   d) `image`: Imagen asociada.
   e) `is_synthetic`: Indica si la anotación fue generada sintéticamente.
   f) `data`: Datos de la anotación en formato JSON.
5. **Job:** Tarea de etiquetado o procesamiento asignada a usuarios.
   a) `id` : Identificador único.
   b) `dataset`: Dataset asociado.
   c) `owner`: Usuario que creó el trabajo.
   d) `assignee`: Usuario asignado.
   e) `batch`: Lote de datos asociado (opcional).
   f) `timestamp`: Fecha y hora de creación.
   g) `done`: Indica si el trabajo fue completado.
   h) `reviewed`: Indica si el trabajo fue revisado.
6. **DataBatch:** Lote de datos para organización dentro de un dataset.
   a) `id` : Identificador único.
   b) `name`: Nombre del lote.
   c) `dataset`: Dataset asociado.
   d) `timestamp`: Fecha y hora de creación.
   e) `workbench`: Indica si está activo para trabajo.
7. **DatasetSnapshot:** Versión guardada de un dataset.
   a) `id`: Identificador único.
   b) `dataset`: Dataset asociado.
   c) `file`: Archivo de snapshot (generalmente zip).

Podemos observar un diagrama de la base de datos en la figura 6.7.

Todos estos modelos se representan mediante clases utilizando el ORM de Django, lo que permite mapear directamente las entidades de la base de datos a objetos del lenguaje. Un ejemplo de esta representación puede observarse en el siguiente fragmento de código:

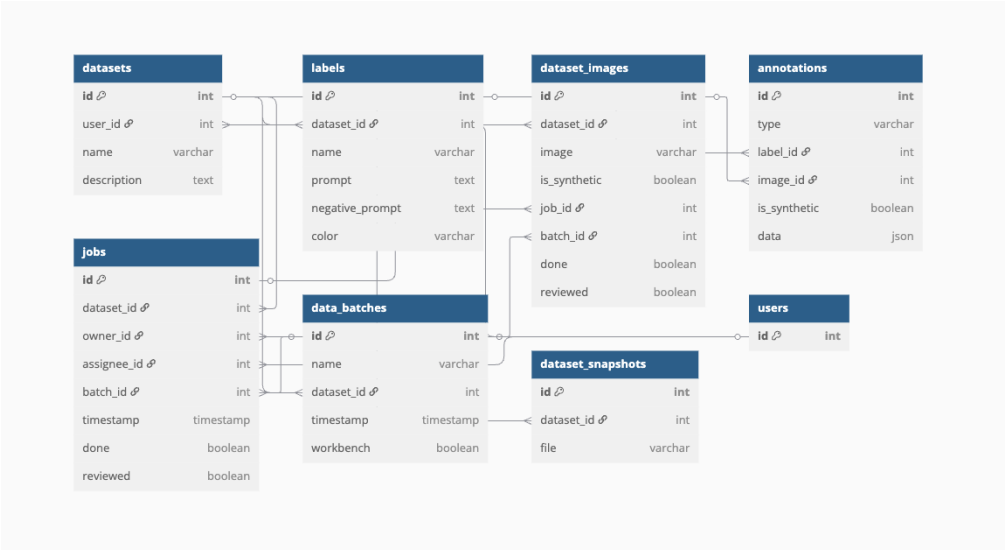


Ilustración 6.7: Diagrama de la base de datos

```
class Dataset(Entity):
    user = models.ForeignKey(settings.AUTH_USER_MODEL, on_delete=models.CASCADE)
    name = models.CharField(max_length=100)
    description = models.TextField()

    def __str__(self):
        return self.name
```

6.3. API REST

El diseño de la API REST constituye una parte fundamental del sistema. Durante el desarrollo, esta API se ha utilizado para todas las pruebas, incluyendo el almacenamiento de imágenes generadas, la modificación y extracción de etiquetas, entre otras funcionalidades. En esta sección se describen los distintos endpoints que conforman la API y su propósito dentro del sistema.

6.3.0.1. Dataset

Entidad que representa un *Dataset*.

1. **GET /datasets/**: Listar todos los *datasets*.
2. **POST /datasets/**: Crear un nuevo *dataset*.
3. **GET /datasets/{dataset_id}/**: Obtener detalles de un dataset específico.

4. **PUT/PATCH /datasets/{dataset_id}/**: Actualizar un dataset existente.
5. **DELETE /datasets/{dataset_id}/**: Eliminar un *dataset*.

6.3.0.2. Label

Representa los distintos tipos de etiquetas para los objetos dentro de un *Dataset*.

1. **GET /datasets/{dataset_id}/labels/**: Listar etiquetas del dataset.
2. **POST /datasets/{dataset_id}/labels/**: Crear una nueva etiqueta.
3. **GET /datasets/{dataset_id}/labels/{label_id}/**: Obtener detalles de una etiqueta específica.
4. **PUT/PATCH /datasets/{dataset_id}/labels/{label_id}/**: Actualizar una etiqueta.
5. **DELETE /datasets/{dataset_id}/labels/{label_id}/**: Eliminar una etiqueta.

6.3.0.3. DatasetImage

Representa una muestra dentro del *Dataset*; todas las muestras son imágenes acompañadas de sus metadatos.

1. **GET /datasets/{dataset_id}/images/**: Listar imágenes del dataset.
2. **POST /datasets/{dataset_id}/images/**: Subir una nueva imagen.
3. **GET /datasets/{dataset_id}/images/{image_id}/**: Obtener detalles de una imagen específica.
4. **PUT/PATCH /datasets/{dataset_id}/images/{image_id}/**: Actualizar una imagen.
5. **DELETE /datasets/{dataset_id}/images/{image_id}/**: Eliminar una imagen.

6.3.0.4. Annotation

Anotaciones realizadas sobre una muestra del *Dataset*.

1. **GET /datasets/{dataset_id}/images/{image_id}/annotations/**: Listar anotaciones de una imagen.
2. **POST /datasets/{dataset_id}/images/{image_id}/annotations/**: Crear una anotación para la imagen.
3. **GET /datasets/{dataset_id}/images/{image_id}/annotations/{annotation_id}/**: Obtener detalles de una anotación.

4. **PUT/PATCH /datasets/{dataset_id}/images/{image_id}/annotations/{annotation_id}**
   Actualizar una anotación.
5. **DELETE /datasets/{dataset_id}/images/{image_id}/annotations/{annotation_id}/**:
   Eliminar una anotación.

6.3.0.5. Job

Tarea de etiquetado o procesamiento de muestras dentro del *Dataset*, compuesta por una o varias muestras y asignada a uno o varios miembros del equipo.

1. **GET /datasets/{dataset_id}/jobs/**: Listar trabajos.
2. **POST /datasets/{dataset_id}/jobs/**: Crear un nuevo trabajo.
3. **GET /datasets/{dataset_id}/jobs/{job_id}/**: Obtener detalles de un trabajo.
4. **PUT/PATCH /datasets/{dataset_id}/jobs/{job_id}/**: Actualizar un trabajo.
5. **DELETE /datasets/{dataset_id}/jobs/{job_id}/**: Eliminar un trabajo.

6.3.0.6. DataBatch

Grupo de muestras dentro del *Dataset* con el propósito de organización y gestión.

1. **GET /datasets/{dataset_id}/batches/**: Listar lotes de datos.
2. **POST /datasets/{dataset_id}/batches/**: Crear un lote de datos.
3. **GET /datasets/{dataset_id}/batches/{batch_id}/**: Obtener detalles de un lote.
4. **PUT/PATCH /datasets/{dataset_id}/batches/{batch_id}/**: Actualizar un lote.
5. **DELETE /datasets/{dataset_id}/batches/{batch_id}/**: Eliminar un lote.

6.3.0.7. DatasetSnapshot

Versión almacenada del dataset. Estas versiones se guardan usualmente en formato ZIP y pueden descargarse en cualquier momento.

1. **GET /datasets/{dataset_id}/snapshots/**: Listar snapshots.
2. **POST /datasets/{dataset_id}/snapshots/**: Crear un snapshot.
3. **GET /datasets/{dataset_id}/snapshots/{snapshot_id}/**: Obtener detalles de un snapshot.
4. **DELETE /datasets/{dataset_id}/snapshots/{snapshot_id}/**: Eliminar un snapshot.

Capítulo 7

Conclusiones y trabajo futuro

7.1. Conclusiones

A lo largo de este trabajo hemos comprobado que los datos sintéticos generados mediante modelos de difusión pre-entrenados representan una estrategia efectiva de *data augmentation*, capaz de mejorar el rendimiento de modelos de detección cuando se combinan con conjuntos de datos reales.

No obstante, aún queda un largo camino por recorrer para que estos datos sintéticos puedan considerarse un sustituto válido de los *datasets* reales, a pesar de su notable capacidad para replicar sus distribuciones.

Actualmente, en el estado del arte, persisten dos desafíos clave: por un lado, cómo extraer etiquetas precisas y útiles de los datos sintéticos generados; y por otro, cómo lograr que la generación de dichos datos se asemeje lo máximo posible a las características y variedad presentes en los conjuntos de datos reales.

7.2. Trabajo futuro

Líneas de investigación futuras:

1. Evaluar el rendimiento de las técnicas de aumento de datos propuestas utilizando *datasets* públicos.
2. Analizar la efectividad de dichas técnicas en dominios distintos al marítimo.
3. Profundizar en métodos para extraer etiquetas de detección a partir de mapas de atención cruzada, abordando especialmente el problema de las oclusiones.
4. Investigar cómo diferentes estrategias de condicionamiento en los modelos afectan el valor del *FID* entre datos reales y sintéticos.

Líneas de desarrollo futuras:

1. Implementar en la *CLI* la opción de generar automáticamente ficheros *csv* compatibles con cada uno de los *pipelines* mediante integraciones con múltiples *LLMs*.
2. Integrar los *pipelines* desarrollados en la plataforma para facilitar la generación eficiente y sencilla de datos sintéticos.
3. Implementar un sistema de múltiples usuarios que permita la creación de equipos y la gestión colaborativa de usuarios.
4. Añadir interacción en tiempo real durante el proceso de etiquetado.
5. Optimizar la gestión y subida de archivos para mejorar la eficiencia.
6. Incorporar la funcionalidad de etiquetado para videos.

Capítulo 8

Anexos

Anexo 1: *Prompts* utilizados durante la generación de datos

Prompt ChatGPT experimento 1

You are an advanced prompt generator for Stable Diffusion, specializing in nautical scenes. Your task is to generate a total of 100 unique and creative prompts by combining the following categories:

Ship types (which I will provide) Environments (which I will provide with a example prompt) Weather conditions (which I will provide with a example prompt) Perspectives(which I will provide with a example prompt)

Prompt generation rules: 90 prompts must feature only one type of ship. These can show a single ship .ª single yacht at night..", few of them "Few open motorboats....ºr many of them "Many open cargoships gathering...".

10 prompts must include multiple types of ships together. You may describe scenes like .ª merchant ship sailing past a warship in the distance", or .ª crowded harbor where fishing boats and military vessels intermingle".

Each prompt must include exactly one environment, one weather condition, and one perspective.

Ensure an even distribution across all categories: Each ship type must appear the same number of times across the 90 single-ship prompts. The same goes for environments, weather conditions, and perspectives.

Vary the style and level of detail of the prompts. But keep the next structure: quantity + type of boats + environment description + perspective.

Use vivid, sensory, and narrative-rich descriptions that evoke a mood, a moment, or a story. Example: A cargo ship sailing alone in the dark night with faint lights from a port in

the distance, viewed from a back down near perspective.

Do not mention specific artist names or art styles unless instructed to.

Output format: A numbered list from 1 to 100 Each line should be a complete, imaginative, and detailed prompt written in English, ready for Stable Diffusion.

Wait until I provide the lists of: Ship types Environments Weather conditions Perspectives

Once you receive those, generate the 100 prompts following all the instructions above.

Answer Ok to this message and when I provide the list mentioned before just output the prompts with enumerated from 1 to 90.

Anexo 2: Descripciones textuales de los conjuntos de datos sintetizados

Definición del conjunto de datos del *Pipeline 1*

Nombre	**Prompt**
Montaña	Ocean with a background of rocky mountains and low clouds.
Ciudad	Ocean with a background of a coastal city.
Playa	Background of a white sand beach and palm trees.
Puerto	Ocean with a background of a port with docked ships and cargo cranes.
Solo Cielo	Background of a clear blue sky, without clouds.
Bosque	Dense and green forest, with tall trees and soft shadows.

Cuadro 8.1: Entornos posibles

Nombre	**Prompt**
Carguero	Cargo ship.
Kayak	Kayak.
Barco pesquero	Fishing boat.
Yate	Yacht.
Lancha hinchable	Inflatable boat (dinghy).
Crucero	Cruise ship.
Lancha abierta	Open motorboat.
Velero	Sailboat.

Cuadro 8.2: Tipos de embarcación

Nombre	Prompt
Soleado	Bright sunny day.
Atardecer	Colorful sunset sky.
Noche	Dark night with little light.
Amanecer	Sunrise with soft light.

Cuadro 8.3: Condiciones ambientales

Anexo 3: Comparación múltiples métodos de umbralizado de los mapas de atención cruzada

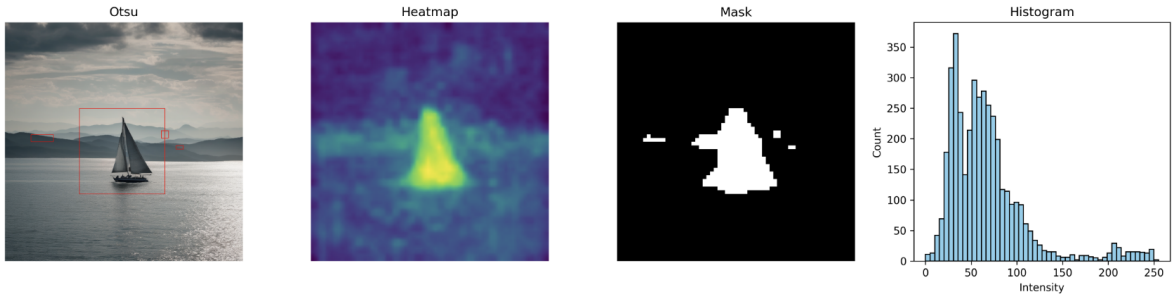


Ilustración 8.1: Umbralizado meidante Otsu

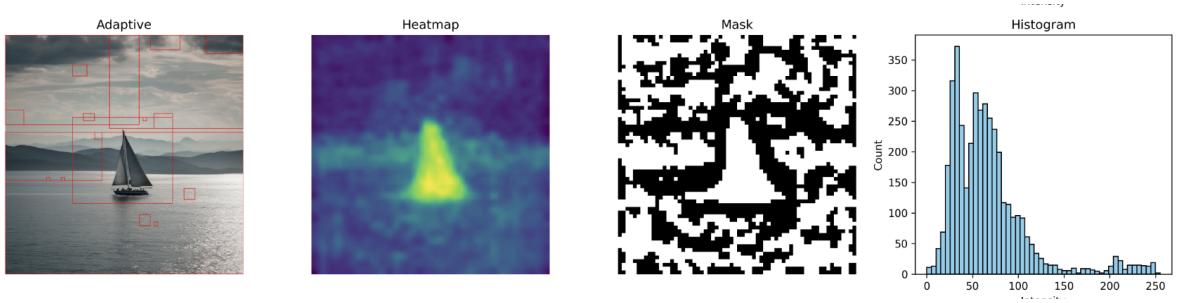


Ilustración 8.2: Umbralizado mediante umbral adaptativo

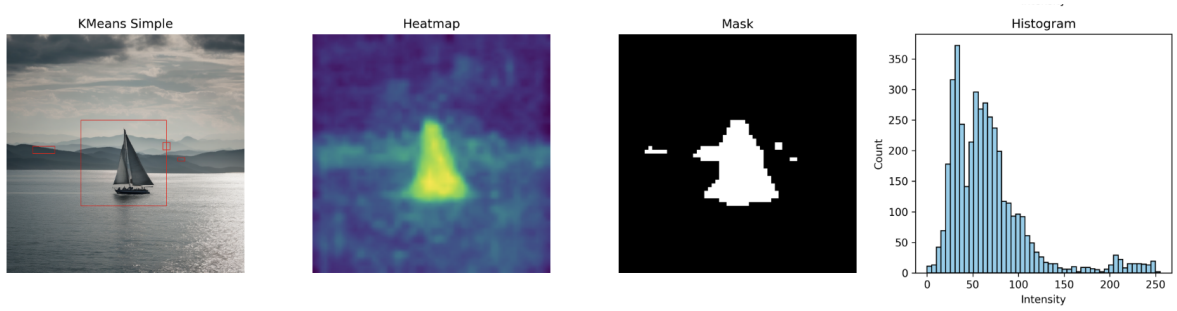


Ilustración 8.3: Umbralizado mediante KMeans

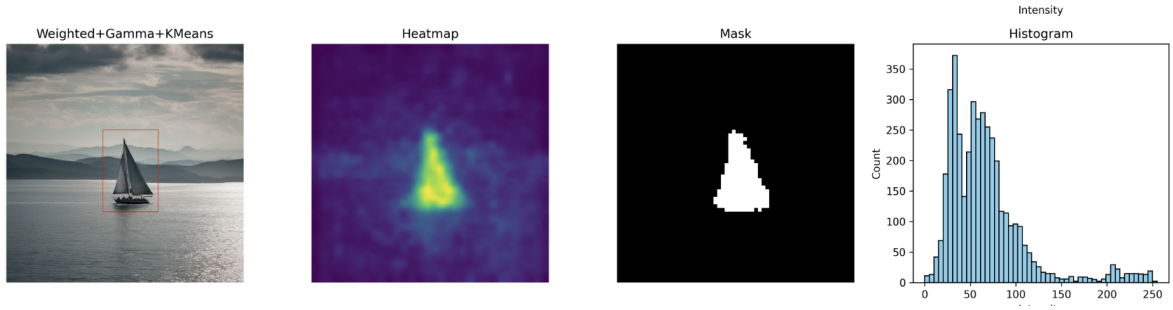


Ilustración 8.4: Umbralizado mediante KMeans añadiendo correción gamma con un factor *2.0*

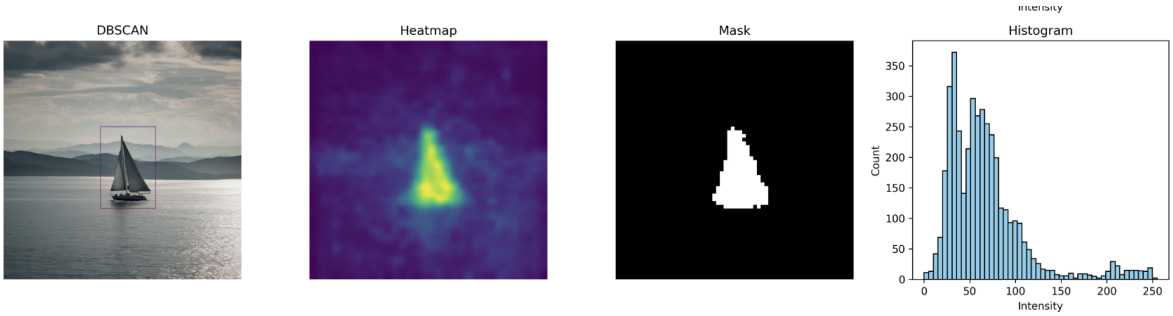


Ilustración 8.5: Umbralizado mediante DBSCAN

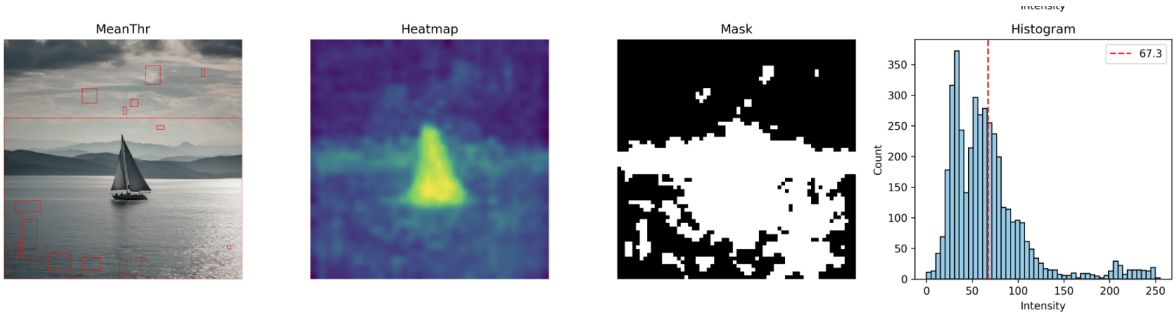


Ilustración 8.6: Umbralizado mediante el valor promedio de los píxeles

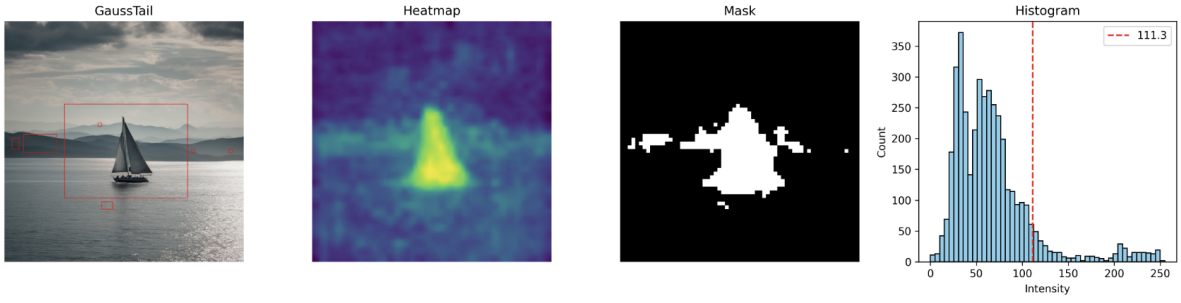


Ilustración 8.7: Umbralizado mediante la cola de la distribución *gaussiana* aproximada

Bibliografía

[hug] Text to image leaderboard - a hugging face space by artificialanalysis. [Online; accessed 2025-05-14].

[2] (2024). Black forest labs - frontier ai lab. [Online; accessed 2025-04-29].

[3] Admin, W. B. I. (2001). Introducing stable diffusion 3.5 — stability ai. [Online; accessed 2025-05-15].

[4] Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein gan.

[5] Balaji, Y., Nah, S., Huang, X., Vahdat, A., Song, J., Zhang, Q., Kreis, K., Aittala, M., Aila, T., Laine, S., Catanzaro, B., Karras, T., and Liu, M.-Y. (2023). ediff-i: Text-to-image diffusion models with an ensemble of expert denoisers.

[6] Brooks, T., Holynski, A., and Efros, A. A. (2023). Instructpix2pix: Learning to follow image editing instructions.

[7] Caesar, H., Uijlings, J., and Ferrari, V. (2018). Coco-stuff: Thing and stuff classes in context.

[8] Chen, J., Bai, G., Liang, S., and Li, Z. (2016). Automatic image cropping: A computational complexity study. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 507–515.

[9] Chen, P., Liu, S., Zhao, H., Wang, X., and Jia, J. (2024). Gridmask data augmentation.

[10] CompVis. Compvis/latent-diffusion: High-resolution image synthesis with latent diffusion models. [Online; accessed 2025-04-23].

[11] CompVis. Compvis/stable-diffusion-v1-1 · hugging face. [Online; accessed 2025-04-23].

[12] Cordeiro, F. R. and Carneiro, G. (2020). A survey on deep learning with noisy labels: How to train your model when you cannot trust on the annotations?

[13] Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. (2019). Autoaugment: Learning augmentation policies from data.

[14] Cubuk, E. D., Zoph, B., Shlens, J., and Le, Q. (2020). Randaugment: Practical automated data augmentation with a reduced search space. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 18613–18624. Curran Associates, Inc.

[15] Dehghani, M., Djolonga, J., Mustafa, B., Padlewski, P., Heek, J., Gilmer, J., Steiner, A., Caron, M., Geirhos, R., Alabdulmohsin, I., Jenatton, R., Beyer, L., Tschannen, M., Arnab, A., Wang, X., Riquelme, C., Minderer, M., Puigcerver, J., Evci, U., Kumar, M., van Steenkiste, S., Elsayed, G. F., Mahendran, A., Yu, F., Oliver, A., Huot, F., Bastings, J., Collier, M. P., Gritsenko, A., Birodkar, V., Vasconcelos, C., Tay, Y., Mensink, T., Kolesnikov, A., Pavetić, F., Tran, D., Kipf, T., Lučić, M., Zhai, X., Keysers, D., Harmsen, J., and Houlsby, N. (2023). Scaling vision transformers to 22 billion parameters.

[16] DeVries, T. and Taylor, G. W. (2017). Improved regularization of convolutional neural networks with cutout.

[17] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale.

[18] Doubinsky, P., Audebert, N., Crucianu, M., and Le Borgne, H. (2024). Semantic generative augmentations for few-shot counting. In *2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 5431–5440.

[19] Dunlap, L., Umino, A., Zhang, H., Yang, J., Gonzalez, J. E., and Darrell, T. (2023). Diversify your vision datasets with automatic diffusion-based augmentation. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S., editors, *Advances in Neural Information Processing Systems*, volume 36, pages 79024–79034. Curran Associates, Inc.

[20] Esser, P., Kulal, S., Blattmann, A., Entezari, R., Müller, J., Saini, H., Levi, Y., Lorenz, D., Sauer, A., Boesel, F., Podell, D., Dockhorn, T., English, Z., Lacey, K., Goodwin, A., Marek, Y., and Rombach, R. (2024). Scaling rectified flow transformers for high-resolution image synthesis.

[21] Fang, H., Han, B., Zhang, S., Zhou, S., Hu, C., and Ye, W.-M. (2024). Data augmentation for object detection via controllable diffusion models. In *2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1246–1255.

[Fei-Fei Li] Fei-Fei Li, E. A. Cs231: Lecture 13, generative models. [Online; accessed 2025-04-20].

[23] Feng, C., Zhong, Y., Jie, Z., Xie, W., and Ma, L. (2024). Instagen: Enhancing object detection by training on synthetic dataset.

[24] Foster, D. (2023). *Generative Deep Learning: Teaching machines to paint, write, compose, and play (Second Edition)*. O'reilly.

[25] Gal, R., Alaluf, Y., Atzmon, Y., Patashnik, O., Bermano, A. H., Chechik, G., and Cohen-Or, D. (2022). An image is worth one word: Personalizing text-to-image generation using textual inversion.

[26] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial networks.

[GrandViewResearch] GrandViewResearch. Computer vision market size, share trends analysis report.

[28] Hendrycks, D., Mu, N., Cubuk, E. D., Zoph, B., Gilmer, J., and Lakshminarayanan, B. (2020). Augmix: A simple data processing method to improve robustness and uncertainty.

[29] Hinton, G. E. (2008). Reducing the dimensionality of data with neural.

[30] Ho, D., Liang, E., Chen, X., Stoica, I., and Abbeel, P. (2019). Population based augmentation: Efficient learning of augmentation policy schedules. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2731–2741. PMLR.

[31] Ho, J., Jain, A., and Abbeel, P. (2020). Denoising diffusion probabilistic models.

[32] Ho, J. and Salimans, T. (2022). Classifier-free diffusion guidance.

[33] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

[34] Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. (2021). Lora: Low-rank adaptation of large language models.

[35] Kang, G., Dong, X., Zheng, L., and Yang, Y. (2017). Patchshuffle regularization.

[36] Karras, T., Aila, T., Laine, S., and Lehtinen, J. (2018). Progressive growing of gans for improved quality, stability, and variation.

[37] Karras, T., Laine, S., and Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4396–4405.

[38] Kaur, P., Khehra, B. S., and Mavi, E. B. S. (2021). Data augmentation for object detection: A review. In *2021 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 537–543.

[39] Kingma, D. P. and Welling, M. (2022). Auto-encoding variational bayes.

[40] Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Kolesnikov, A., Duerig, T., and Ferrari, V. (2020). The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *International Journal of Computer Vision*, 128(7):1956–1981.

[41] Lei, J., Zhang, R., Hu, X., Lin, W., Li, Z., Sun, W., Du, R., Zhuo, L., Li, Z., Li, X., Zhao, S., Guo, Z., Lu, Y., Gao, P., and Li, H. (2025). Imagine-e: Image generation intelligence evaluation of state-of-the-art text-to-image models.

[42] Lim, S., Kim, I., Kim, T., Kim, C., and Kim, S. (2019). Fast autoaugment. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

[43] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollár, P. (2015). Microsoft coco: Common objects in context.

[44] Liu, X., Gong, C., and Liu, Q. (2022a). Flow straight and fast: Learning to generate and transfer data with rectified flow.

[45] Liu, Z., Li, S., Wu, D., Liu, Z., Chen, Z., Wu, L., and Li, S. Z. (2022b). Automix: Unveiling the power of mixup for stronger classifiers.

[46] Luo, X.-J., Wang, S., Wu, Z., Sakaridis, C., Cheng, Y., Fan, D.-P., and Gool, L. V. (2023). Camdiff: Camouflage image augmentation via diffusion. *CAAI Artificial Intelligence Research*, 2:9150021.

[47] Maharana, K., Mondal, S., and Nemade, B. (2022). A review: Data pre-processing and data augmentation techniques. *Global Transitions Proceedings*, 3(1):91–99. International Conference on Intelligent Engineering Approach(ICIEA-2022).

[48] Meng, C., He, Y., Song, Y., Song, J., Wu, J., Zhu, J.-Y., and Ermon, S. (2022). Sdedit: Guided image synthesis and editing with stochastic differential equations.

[49] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space.

[mlfoundations] mlfoundations. mlfoundations/open_clip: An open source implementation of clip. [Online; accessed 2025-04-24].

[51] Niu, L., Cong, W., Liu, L., Hong, Y., Zhang, B., Liang, J., and Zhang, L. (2021). Making images real again: A comprehensive survey on deep image composition. *arXiv preprint arXiv:2106.14490*.

[52] Peebles, W. and Xie, S. (2023). Scalable diffusion models with transformers.

[53] Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global vectors for word representation. In Moschitti, A., Pang, B., and Daelemans, W., editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

[54] Pernias, P., Rampas, D., Richter, M. L., Pal, C., and Aubreville, M. (2024). Würstchen: An efficient architecture for large-scale text-to-image diffusion models. In *The Twelfth International Conference on Learning Representations*.

[55] Podell, D., English, Z., Lacey, K., Blattmann, A., Dockhorn, T., Müller, J., Penna, J., and Rombach, R. (2023). Sdxl: Improving latent diffusion models for high-resolution image synthesis.

[56] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. (2021). Learning transferable visual models from natural language supervision.

[57] Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks.

[58] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2022). High-resolution image synthesis with latent diffusion models.

[59] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation.

[60] Rumelhart, D. E. and McClelland, J. L. (1987). *Learning Internal Representations by Error Propagation*, pages 318–362.

[61] Salimans, T. and Ho, J. (2022). Progressive distillation for fast sampling of diffusion models.

[62] Sánchez Cordero, K. (2024). Detección y seguimiento de embarcaciones mediante métodos de inteligencia artificial y cámaras ptz. B.S. thesis.

[63] Sauer, A., Boesel, F., Dockhorn, T., Blattmann, A., Esser, P., and Rombach, R. (2024). Fast high-resolution image synthesis with latent adversarial diffusion distillation.

[64] Sauer, A., Lorenz, D., Blattmann, A., and Rombach, R. (2023). Adversarial diffusion distillation.

[65] Schuhmann, C., Vencu, R., Beaumont, R., Kaczmarczyk, R., Mullis, C., Katta, A., Coombes, T., Jitsev, J., and Komatsuzaki, A. (2021). Laion-400m: Open dataset of clip-filtered 400 million image-text pairs.

[66] Shen, X., Tian, X., Sun, S., and Tao, D. (2019). Patch reordering: a novel way to achieve rotation and translation invariance in convolutional neural networks.

[67] Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., and Ganguli, S. (2015). Deep unsupervised learning using nonequilibrium thermodynamics.

[StabilityAI] StabilityAI. stabilityai/stable-diffusion-2-base · hugging face. [Online; accessed 2025-04-23].

[69] Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., and Liu, Y. (2023). Roformer: Enhanced transformer with rotary position embedding.

[70] Trabucco, B., Doherty, K., Gurinas, M., and Salakhutdinov, R. (2023). Effective data augmentation with diffusion models.

[71] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2023). Attention is all you need.

[72] Wang, Z., Wang, P., Liu, K., Wang, P., Fu, Y., Lu, C.-T., Aggarwal, C. C., Pei, J., and Zhou, Y. (2024). A comprehensive survey on data augmentation.

[73] Wu, W., Zhao, Y., Chen, H., Gu, Y., Zhao, R., He, Y., Zhou, H., Shou, M. Z., and Shen, C. (2023). Datasetdm: Synthesizing data with perception annotations using diffusion models. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S., editors, *Advances in Neural Information Processing Systems*, volume 36, pages 54683–54695. Curran Associates, Inc.

[74] Wu, W., Zhao, Y., Shou, M. Z., Zhou, H., and Shen, C. (2024). Diffumask: Synthesizing images with pixel-level annotations for semantic segmentation using diffusion models.

[75] Yu, F., Seff, A., Zhang, Y., Song, S., Funkhouser, T., and Xiao, J. (2016). Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop.

[76] Yu, X., Li, G., Lou, W., Liu, S., Wan, X., Chen, Y., and Li, H. (2023). Diffusion-based data augmentation for nuclei image segmentation. In Greenspan, H., Madabhushi, A., Mousavi, P., Salcudean, S., Duncan, J., Syeda-Mahmood, T., and Taylor, R., editors, *Medical Image Computing and Computer Assisted Intervention – MICCAI 2023*, pages 592–602, Cham. Springer Nature Switzerland.

[77] Yun, S., Han, D., Oh, S. J., Chun, S., Choe, J., and Yoo, Y. (2019). Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

[78] Zhan, Z., Chen, D., Mei, J.-P., Zhao, Z., Chen, J., Chen, C., Lyu, S., and Wang, C. (2024). Conditional image synthesis with diffusion models: A survey.

[79] Zhang, H., Cisse, M., Dauphin, Y. N., and Lopez-Paz, D. (2018). mixup: Beyond empirical risk minimization.

[Zhang et al.] Zhang, L., Rao, A., and Agrawala, M. Adding conditional control to text-to-image diffusion models.

[81] Zhong, Z., Zheng, L., Kang, G., Li, S., and Yang, Y. (2017a). Random erasing data augmentation.

[82] Zhong, Z., Zheng, L., Kang, G., Li, S., and Yang, Y. (2017b). Random erasing data augmentation.