



UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

INGENIERÍA INFORMÁTICA

DESARROLLO DE UN SISTEMA DE RECUPERACIÓN DE IMÁGENES BASADA EN I.A. GENERATIVA

Autora:

Carlota de la Vega Soriano

Director:

Jesús Chamorro Martínez



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y
TELECOMUNICACIONES

Granada, junio de 2025

Desarrollo de un sistema de recuperación de imágenes basada en I.A. Generativa

Carlota de la Vega Soriano

Palabras clave: recuperación de imágenes, inteligencia artificial generativa, CBIR, GAN, procesamiento de lenguaje natural

Resumen

El creciente volumen de contenidos multimedia ha hecho cada vez más necesarios los sistemas de recuperación de información visual (CBIR). Tradicionalmente, estos sistemas se han basado en descriptores de bajo nivel extraídos directamente de las imágenes, lo que dificulta que las consultas reflejen de forma semántica las necesidades del usuario. Para superar esta limitación, el presente proyecto plantea la incorporación de técnicas de inteligencia artificial generativa como solución innovadora, concretamente para la generación de imágenes consulta a partir de descripciones textuales con alto contenido semántico.

Este trabajo tiene como objetivo principal el desarrollo de módulos para la plataforma Java Multimedia Retrieval (JMR) que permitan la integración de consultas textuales como mecanismo de entrada en sistemas CBIR. Para ello, se han definido tres objetivos específicos: la revisión del estado del arte en generación de imágenes a partir de texto, el desarrollo de algoritmos que traduzcan descripciones lingüísticas en representaciones visuales, y la implementación de un prototipo funcional de recuperación basado en texto.

La metodología seguida ha estado basada en el modelo en cascada, estructurada en fases secuenciales de análisis, diseño, implementación y pruebas. Se ha realizado una evaluación comparativa de distintas arquitecturas generativas, incluyendo GAN, cGAN y AttnGAN, así como el uso de modelos preentrenados como Stable Diffusion. Para el entrenamiento y evaluación, se han empleado distintos conjuntos de datos, como MNIST, CIFAR, COCO y Stanford Dogs, ajustando los modelos a diferentes niveles de complejidad semántica.

Entre los principales resultados alcanzados, destaca la validación de un sistema funcional capaz de transformar texto en imágenes consulta, y su integración en la arquitectura de la JMR. Las pruebas realizadas evidencian una mejora en la capacidad de formulación de consultas por parte del usuario, acercando el sistema CBIR a una experiencia más intuitiva y semánticamente rica.

Como conclusión, se demuestra que la inteligencia artificial generativa puede enriquecer significativamente la interacción con sistemas de recuperación visual, permitiendo consultas más expresivas y adaptadas al lenguaje humano. Este enfoque abre nuevas posibilidades en aplicaciones donde la precisión semántica en la búsqueda de imágenes es esencial, como la educación, la medicina o el diseño.

Development of an image retrieval system based on Generative A.I.

Carlota de la Vega Soriano

Keywords: image retrieval, generative artificial intelligence, CBIR, GAN, natural language processing

Abstract

The increasing volume of multimedia content has made visual information retrieval systems (CBIR) more necessary than ever. Traditionally, these systems have relied on low-level descriptors extracted directly from images, which limits their ability to reflect user intent semantically. To overcome this limitation, this project proposes the integration of generative artificial intelligence techniques as an innovative solution, specifically for generating query images from text descriptions with high semantic content.

The main objective of this work is to develop modules for the Java Multimedia Retrieval (JMR) platform to allow textual queries as input to CBIR systems. To this end, three specific goals were defined: reviewing the state of the art in text-to-image generation, developing algorithms to transform linguistic descriptions into visual representations, and implementing a functional retrieval prototype based on text.

The methodology followed was based on the waterfall model, structured into sequential phases of analysis, design, implementation, and testing. A comparative evaluation of different generative architectures was conducted, including GAN, cGAN, and AttnGAN, along with the use of pretrained models such as Stable Diffusion. For training and evaluation, various datasets were used — MNIST, CIFAR, COCO, and Stanford Dogs — adjusting models to different levels of semantic complexity.

Among the main results, the project validates a functional system capable of transforming text into query images and integrating it into the JMR architecture. The tests show an improvement in users' ability to formulate expressive queries, making CBIR systems more intuitive and semantically rich.

In conclusion, generative artificial intelligence proves to significantly enhance interaction with visual retrieval systems by enabling more expressive, human-like queries. This approach opens up new possibilities in applications where semantic accuracy in image search is critical, such as education, medicine, or design.

Yo, **Carlota de la Vega Soriano**, alumna de la titulación Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 77203307N, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

D. Jesús Chamorro Martínez, Profesor del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado Desarrollo de un sistema de recuperación de imágenes basada en I.A. Generativa, ha sido realizado bajo su supervisión por Carlota de la Vega Soriano y autorizo la defensa de dicho trabajo ante el tribunal que corresponda. Y para que conste, expiden y firman el presente informe en Granada a 16 de junio de 2025.

El director:

D. Jesús Chamorro Martínez

Agradecimientos

A mis padres, por su apoyo constante, por enseñarme a no rendirme y por estar siempre a mi lado en cada paso, incluso cuando no sabían exactamente a qué me estaba enfrentando. Gracias por hacer esto posible.

A mi hermana Martina, por ser mi inspiración y mi ejemplo a seguir. Gracias por ser mi guía sin proponértelo.

A Aida, por acompañarme cada día, por no dejar que me cayera, por cuidarme y por sacarme una sonrisa siempre que más lo he necesitado. Has sido y eres mi refugio.

A Salma, por estar desde siempre y no moverse. Gracias por recordarme quién soy incluso cuando yo lo olvidaba.

A Jesús, por ser mucho más que un profesor y un tutor. Gracias por acompañarme con cercanía y confianza, y por abrirme las puertas a una nueva familia.

A todos los que me han acompañado en este camino, gracias.

Índice

1. Introducción	21
1.1. Objetivos	22
2. Planificación inicial del trabajo	24
2.1. Planificación temporal	24
2.1.1. Fase 1: Recolección y análisis de requisitos	24
2.1.2. Fase 2: Diseño	24
2.1.3. Fase 3: Implementación y experimentación	25
2.1.4. Fase 4: Integración, pruebas y validación	25
2.1.5. Fase 5: Documentación y entrega	25
2.1.6. Cronograma visual del proyecto	25
2.2. Planificación de recursos	26
2.3. Planificación económica	27
2.4. Ajustes realizados sobre la planificación inicial	27
3. Metodología	28
3.1. Enfoque metodológico	28
3.2. Herramientas y tecnologías utilizadas	28
3.3. Datasets utilizados	29
3.4. Modelos y arquitecturas exploradas	29
3.5. Justificación técnica y decisiones adoptadas	30
I Desarrollo de modelos	31
4. Estado del Arte	32
4.1. Antecedentes	32
4.2. Sistemas de Recuperación de Imágenes por Rasgos Visuales (CBIR)	32
4.2.1. Definición	32
4.2.2. Características	32
4.2.3. Medidas de similitud	32
4.2.4. Aplicaciones	33
4.2.5. Diferencias entre TBIR y CBIR	33
4.3. Descriptoros visuales de bajo nivel	34
4.3.1. Representación visual según MPEG-7	34
4.3.2. Descriptor de color dominante	35
4.3.3. Descriptor de color escalable	35
4.3.4. Descriptor de estructura de color	35
4.4. Redes Neuronales	36
4.4.1. Fundamentos	36
4.4.2. Funciones de activación	36
4.4.3. Tipos	37
4.4.4. Aplicaciones	38
4.5. Redes Generativas Adversarias (GAN)	38
4.5.1. Componentes	38
4.5.2. Proceso de entrenamiento	39
4.5.3. Efectos del ruido en la generación de imágenes	40
4.5.4. Aplicaciones	40
4.6. Redes Generativas Adversarias Condicionales (cGAN)	41
4.6.1. Función de las Etiquetas	41
4.6.2. Proceso de entrenamiento	41
4.6.3. Aplicaciones	42

4.7.	Redes Generativas Adversarias con Atención (AttnGAN)	43
4.7.1.	Arquitectura	43
4.7.2.	DAMSM	43
4.7.3.	Mecanismos de atención	43
4.7.4.	Métricas de evaluación	44
4.7.5.	Proceso de entrenamiento	44
4.7.6.	Aplicaciones	44
4.8.	Procesamiento del lenguaje natural	45
4.8.1.	Preparación del texto	45
4.8.2.	Tokenización	45
4.8.3.	Representación del texto	45
4.9.	Generación de imágenes a partir de texto	46
4.9.1.	Ejemplos	46
4.9.2.	Consideraciones Éticas y de Privacidad	46
4.10.	Modelos de Difusión y Stable Diffusion	48
4.10.1.	Fundamentos de los modelos de difusión	48
4.10.2.	Transición hacia la difusión latente	48
4.10.3.	Stable Diffusion	48
4.10.4.	Ventajas frente a otras arquitecturas	49
4.10.5.	Aplicaciones prácticas y uso en este trabajo	50
4.10.6.	Limitaciones y desafíos actuales	50
5.	Entrenamiento y experimentación con modelos generativos	51
5.1.	Primera aproximación: GAN	51
5.1.1.	Librerías, herramientas y datasets	51
5.1.2.	Pruebas y resultados	51
5.1.3.	Conclusión	52
5.2.	Condicionabilidad: cGAN	53
5.2.1.	Librerías y Herramientas	53
5.2.2.	Procesamiento de Texto	53
5.2.3.	Pruebas preliminares con MNIST	53
5.2.4.	Datasets	54
5.2.5.	Pruebas y resultados	54
5.2.6.	Pruebas con diferentes números de épocas	54
5.2.7.	Visualización de pérdidas	56
5.2.8.	Conclusión	56
5.3.	Atención textual: AttnGAN	56
5.3.1.	Configuración del entrenamiento	56
5.3.2.	Proceso de entrenamiento	57
5.3.3.	Resultados y evaluación visual	57
5.4.	Modelo final	57
5.4.1.	Motivación para el uso de modelos preentrenados	58
5.4.2.	Descripción del modelo: Stable Diffusion	58
5.4.3.	Evaluación inicial del modelo	59
5.4.4.	Exploración de técnicas de optimización	59
5.4.5.	Optimización seleccionada: modificación del espacio latente	60
5.5.	Evaluación y resultados	61
5.5.1.	Limitaciones del modelo base	61
5.5.2.	Evaluación de coherencia semántica con CLIP	62
5.5.3.	Ánálisis del coste de entrenamiento	63
5.5.4.	Evaluación de la generalización del modelo	63
5.6.	Pruebas del sistema de recuperación de imágenes	65
5.6.1.	Metodología de recuperación	65
5.6.2.	Casos de prueba	65

5.6.3. Análisis de resultados	67
5.7. Conclusiones	67
II Desarrollo de software	68
6. Requisitos y análisis	69
6.1. Especificación de requisitos	69
6.1.1. Requisitos funcionales	69
6.1.2. Requisitos no funcionales	70
6.2. Historias de usuario	71
6.3. Modelo de caso de uso	72
6.4. Modelos de comportamiento	73
6.4.1. Diagramas de secuencia	74
6.4.2. Trazabilidad entre casos de uso, historias de usuario y requisitos	78
6.4.3. Diagrama de actividad	78
7. Diseño	80
7.1. Diseño de la arquitectura	80
7.2. Modelo conceptual	80
7.3. Diseño del modelo de clases	81
7.3.1. Modelo de clases en Java	81
7.3.2. Modelo de clases en Python	83
7.4. Diseño de la interfaz	83
7.4.1. Diagrama de flujo de interacción	84
7.4.2. Bocetos	84
7.4.3. Wireframes	86
7.4.4. Prototipo en Figma	88
7.4.5. Usabilidad	88
8. Implementación	90
8.1. Implementación de la API para la Generación de Imágenes	90
8.1.1. Arquitectura de la API	90
8.1.2. Endpoints implementados	91
8.1.3. Flujo de interacción	92
8.1.4. Entorno y despliegue	92
8.1.5. Conclusión	92
8.2. Integración con la plataforma JMR (Java)	92
8.2.1. Arquitectura de integración	92
8.2.2. Flujo de generación de imagen	93
8.2.3. Selección y gestión de la API	93
8.2.4. Consulta sin visualización	93
8.2.5. Historial reutilizable	93
8.2.6. Gestión de errores	93
8.2.7. Comunicación con el sistema generativo: descriptores de imagen	94
8.3. Consideraciones de seguridad y rendimiento	94
8.3.1. Gestión del entorno con Poetry	94
8.3.2. Contenerización y portabilidad con Docker	95
8.3.3. Automatización del desarrollo con GitHub Workflows	95
8.3.4. Seguridad en la API REST	95
8.3.5. Optimización del rendimiento	96
8.3.6. Preparación para producción y despliegue escalable	96
8.3.7. Reflexión final	97
8.4. Pruebas y validación	97

8.4.1. Pruebas sobre la API REST	97
8.4.2. Pruebas en la aplicación JMR	98
8.4.3. Validación de consistencia e integridad	99
8.4.4. Conclusión	99
9. Manual de usuario	100
9.1. Introducción	100
9.2. Inicio de la aplicación	100
9.3. Descripción de la interfaz	100
9.4. Selección de API para generación de imágenes	101
9.5. Cómo generar una imagen	102
9.6. Cómo generar una consulta sin mostrar la imagen	104
9.7. Reutilizar imágenes generadas	105
9.8. Errores y soluciones comunes	106
10. Conclusiones	107
10.1. Líneas futuras de trabajo	107
III Anexos	108
A. Capturas del prototipo Figma	109
B. Glosario de términos	113
B.1. Modelado y entrenamiento	113
B.2. Redes neuronales y arquitectura	113
B.3. Datasets y datos	114
B.4. Interfaz y diseño	115
C. Código fuente del sistema desarrollado	116

Índice de figuras

1.	Resultados en Google Imágenes	21
2.	Imagen generada por el sistema propuesto con el prompt: “ <i>a boy eating blue candies</i> ”	22
3.	Cronograma visual del proyecto, agrupado por fases y tareas.	26
4.	Estructura de una red neuronal	36
5.	GPT-4	38
6.	Proceso de entrenamiento de una GAN	39
7.	Proceso de entrenamiento de una cGAN	42
8.	Procesamiento del lenguaje	45
9.	IA Midjourney	46
10.	Pipeline general del modelo Stable Diffusion. Fuente: [15]	49
11.	Imágenes generadas con el modelo entrenado con MNIST	53
12.	Curva de pérdidas durante el entrenamiento de la cGAN	56
13.	Mejor imagen generada por AttnGAN durante el entrenamiento con el prompt: “ <i>A man with a red jacket riding a motorcycle in the woods</i> ”	57
14.	Imagen generada con Stable Diffusion v1.4 a partir del prompt “ <i>A beautiful landscape with mountains and a river</i> ”	59
15.	Ejemplos de imágenes generadas tras la especialización del modelo con el prompt “ <i>a photo of a golden retriever wearing sunglasses</i> ”	61
16.	Resultado generado por el modelo preentrenado para el prompt: “ <i>a photo of a golden retriever</i> ”	61
17.	Resultado generado tras la especialización del modelo para el prompt: “ <i>a photo of a golden retriever</i> ”	62
18.	Comparación visual del CLIP Score relativo.	62
19.	Imagen generada por el modelo base con el prompt “ <i>a man sitting on a bench in a park</i> ”	64
20.	Imagen generada por el modelo especializado con el mismo prompt: “ <i>a man sitting on a bench in a park</i> ”	64
21.	Recuperación visual a partir de la imagen generada con el prompt “ <i>a golden retriever in the park</i> ”	65
22.	Recuperación visual a partir de la imagen generada con el prompt “ <i>a lot of colorful sprinkles</i> ”	66
23.	Recuperación visual a partir de la imagen generada con el prompt “ <i>a cloudy day</i> ”, tras visualizarla previamente en la aplicación.	66
24.	Diagrama de caso de uso	73
25.	Diagrama de secuencia del uso de descriptores	75
26.	Diagrama de secuencia de la generación de una imagen	75
27.	Diagrama de secuencia del guardado de una imagen	76
28.	Diagrama de secuencia del conjunto de datos COCO	76
29.	Diagrama de secuencia del entrenamiento	77
30.	Diagrama de secuencia de pérdidas	77
31.	Diagrama de actividad general del sistema	79
32.	Arquitectura lógica del sistema integrado	80
33.	Representación esquemática del modelo conceptual del sistema	81
34.	Diagrama de clases de los descriptores generativos en Java	82
35.	Diagrama de clases del módulo de integración en Java	82
36.	Diagrama de clases del sistema generativo en Python	83
37.	Flujo de interacción entre el usuario, la API generativa y el sistema CBIR	84
38.	Boceto: propuesta inicial del layout general	85
39.	Boceto: navegación entre secciones	85
40.	Boceto: detalle de interacción en la vista de resultados	86
41.	Wireframe: pantalla principal con área de generación	86
42.	Wireframe: visualización detallada de una imagen generada	87
43.	Wireframe: pantalla de resultados con opciones de filtrado	87
44.	Vista general del flujo de navegación entre pantallas del prototipo en Figma	88
45.	Diagrama de componentes de la API generativa	90

46.	Endpoints de la API generativa	91
47.	Prueba del endpoint GET /models/list/ desde Swagger	98
48.	Mensaje de error al intentar generar un prompt vacío	99
49.	Inicio de la aplicación con las dos nuevas funcionalidades	100
50.	Menú desplegable para seleccionar entre API local u online	101
51.	Ventana emergente para introducir el token de Hugging Face	101
52.	Botón para generar una imagen	102
53.	Ventana interna con área para escribir y botón de visualizar	102
54.	Ventana interna con área escrita con una descripción de ejemplo	103
55.	Ejemplo de visualización	103
56.	Escribir la descripción en la barra superior	104
57.	Botón de búsqueda para ejecutar la consulta	104
58.	Resultados visuales recuperados de la base de datos	105
59.	A la izquierda, vista del historial con una entrada seleccionada. A la derecha, la imagen generada que se abre al seleccionar esa entrada.	105
60.	Pantalla principal del sistema	109
61.	Ventana emergente de generación de imágenes	109
62.	Imagen generada y visualizada en una ventana	110
63.	Galería de resultados con miniaturas	110
64.	Histórico de prompts con miniatura	111
65.	Resultado de abrir un prompt del histórico	111
66.	Transición desde pantalla principal a ventana de generación	111
67.	Transición desde histórico a ventana de visualización de imagen	112

Índice de tablas

1.	Presupuesto del proyecto	27
2.	Comparativa TBIR vs CBIR	34
3.	Resumen de experimentos con GAN: configuración y resultados	52
4.	Comparativa entre entrenar con menos épocas y entrenar con más épocas	55
5.	Resumen funcional de los componentes principales de Stable Diffusion	58
6.	Similitud relativa medida con CLIP	62
7.	Recursos técnicos del servidor utilizado para el entrenamiento final	63
8.	Requisito Funcional RF 1.1	69
9.	Requisito Funcional RF 2.1	69
10.	Requisito Funcional RF 3.1	69
11.	Requisito Funcional RF 4.1	70
12.	Requisito No Funcional RNF 1: Rendimiento y Eficiencia	70
13.	Requisito No Funcional RNF 2: Mantenimiento	70
14.	Requisito No Funcional RNF 3: Compatibilidad	70
15.	Requisito No Funcional RNF 4: Experiencia de Usuario y Accesibilidad	70
16.	Historia de Usuario 1	71
17.	Historia de Usuario 2	71
18.	Historia de Usuario 3	72
19.	Relación entre requisitos y sus historias de usuario	72
20.	Relación entre casos de uso, historias de usuario y requisitos	78

1. Introducción

En los últimos años, el volumen de contenidos multimedia disponibles en plataformas digitales ha crecido exponencialmente, impulsado por redes sociales, bancos de imágenes, repositorios científicos o plataformas de comercio electrónico. Este crecimiento ha intensificado la necesidad de sistemas eficientes para acceder a dicha información de forma rápida, precisa y alineada con la intención del usuario.

En este contexto, los sistemas de recuperación de información visual, comúnmente conocidos como CBIR (Content-Based Image Retrieval), han supuesto un gran avance, al permitir realizar búsquedas en grandes colecciones de imágenes basándose en características visuales como el color, la forma o la textura. Plataformas ampliamente utilizadas como Google ofrecen funcionalidades como la búsqueda inversa por imagen o sugerencias visuales similares. No obstante, incluso estos sistemas, altamente optimizados, se enfrentan a una limitación crítica: no comprenden el significado profundo de una descripción textual compleja.

Por ejemplo, si un usuario introduce en Google Imágenes la búsqueda: “niño comiendo caramelos azules”, los resultados incluyen imágenes de niños comiendo dulces, pero no reflejan de forma precisa todos los elementos descritos. Aparecen niños con piruletas, niños comiendo otros tipos de dulces o con ropa azul, pero no una escena específica con caramelos azules como los que el usuario ha imaginado. Esta limitación es evidente en la Figura 1.

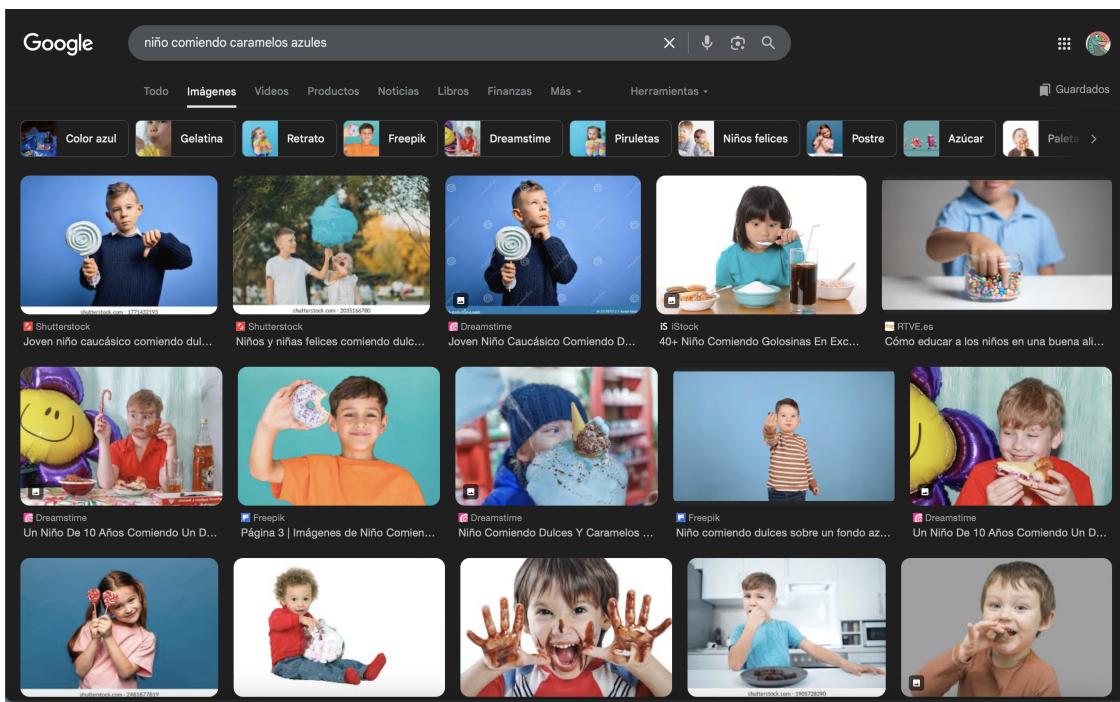


Figura 1: Resultados en Google Imágenes

Frente a esta limitación, los modelos de inteligencia artificial generativa permiten abordar el problema desde una nueva perspectiva: la generación de contenido visual a partir de texto. En lugar de buscar coincidencias entre imágenes etiquetadas, el sistema genera una imagen completamente nueva que refleja de forma fiel los elementos semánticos de la descripción. En la Figura 2, puede observarse cómo el sistema propuesto ha generado una imagen que representa explícitamente la escena solicitada.

Este proyecto se sitúa en la intersección entre la recuperación visual tradicional y la generación de contenido sintético. Mediante el uso de modelos generativos, se desarrolla un sistema capaz de responder a consultas textuales generando imágenes que actúan como “consulta visual”, fusionando así lo mejor del CBIR con las capacidades expresivas de la IA generativa. Esta solución se integra dentro de la plataforma Java Multimedia Retrieval (JMR), proporcionando un sistema de recuperación de imágenes más flexible, accesible y centrado en el usuario.



Figura 2: Imagen generada por el sistema propuesto con el prompt: “*a boy eating blue candies*”

Además del valor técnico, esta tecnología abre nuevas posibilidades para mejorar la accesibilidad en la interacción con sistemas visuales, especialmente para usuarios que no disponen de una imagen de referencia. Poder expresar una búsqueda en lenguaje natural y recibir un resultado visual adecuado representa un avance importante hacia sistemas más inclusivos y adaptativos.

Para validar esta propuesta, se han entrenado y comparado diferentes arquitecturas generativas, se han evaluado sus resultados tanto cualitativa como cuantitativamente, y se ha implementado un prototipo funcional conectado a un sistema de recuperación visual real.

El potencial de este tipo de sistemas va más allá del ámbito académico, con aplicaciones en sectores como la publicidad personalizada, los asistentes creativos, la educación visual o la generación de contenido bajo demanda en plataformas digitales.

1.1. Objetivos

El presente Trabajo Fin de Grado tiene como objetivo principal el desarrollo de un sistema de recuperación de imágenes (CBIR) que permita realizar consultas a través de descripciones textuales con alto contenido semántico, generando dinámicamente imágenes de referencia mediante técnicas de inteligencia artificial generativa. Este enfoque persigue reducir la brecha entre la intención del usuario y los métodos tradicionales de recuperación, proporcionando una experiencia de búsqueda más natural, intuitiva y flexible.

La investigación y el desarrollo en este ámbito se basan en la necesidad actual de sistemas capaces de entender y procesar descripciones humanas de manera efectiva, especialmente en escenarios donde las imágenes de consulta no están disponibles o no son suficientemente representativas.

Para alcanzar el objetivo general, se han establecido los siguientes objetivos específicos:

- **O1. Revisión del estado del arte:** Investigar las técnicas y modelos actuales de generación de imágenes a partir de texto, centrándose en las arquitecturas GAN, cGAN, AttnGAN y modelos de difusión como Stable Diffusion. Se evaluarán los enfoques existentes en la literatura para identificar las ventajas y limitaciones de cada metodología, así como las tendencias recientes en la combinación de visión por computador y procesamiento del lenguaje natural.
- **O2. Desarrollo de algoritmos de generación:** Diseñar e implementar algoritmos que permitan transformar descripciones textuales en representaciones visuales coherentes y relevantes. Estos algoritmos serán evaluados utilizando diferentes conjuntos de datos con el objetivo de analizar su capacidad para manejar distintos niveles de complejidad semántica. Se prestará especial atención a la optimización del entrenamiento, la mejora de la calidad de las imágenes generadas y la reducción de la inestabilidad típica en redes generativas.
- **O3. Implementación de un prototipo funcional:** Integrar los algoritmos desarrollados en la plataforma Java Multimedia Retrieval (JMR), habilitando un nuevo modo de consulta que permita a los

usuarios introducir descripciones textuales en lenguaje natural y recuperar información visual relevante. Este prototipo combinará los resultados de la generación de imágenes con los mecanismos clásicos de búsqueda CBIR, ofreciendo así un sistema híbrido que optimiza tanto la accesibilidad como la precisión de las búsquedas.

El cumplimiento de los objetivos planteados tendrá un impacto significativo tanto a nivel práctico como académico:

- **Aplicaciones prácticas en la industria y la investigación:** La posibilidad de realizar búsquedas visuales a partir de texto abre nuevas oportunidades en campos como la educación (búsqueda de materiales visuales educativos específicos), la medicina (búsqueda de imágenes médicas por descripción de patologías), la seguridad (generación de imágenes de referencia en investigaciones) y el comercio electrónico (búsqueda de productos basados en descripciones de características).
- **Avance en la integración de IA generativa y recuperación de información:** Este proyecto se sitúa en el cruce entre la visión por computador, el procesamiento de lenguaje natural y los sistemas de recuperación de información, contribuyendo al conocimiento actual mediante el estudio y evaluación de arquitecturas generativas para tareas de recuperación de imágenes.
- **Facilitación de interacciones más naturales con sistemas de información:** Al permitir consultas basadas en lenguaje natural, se mejora considerablemente la experiencia de usuario, haciendo más accesibles los sistemas de recuperación visual para personas no expertas en tecnologías digitales.
- **Apertura a futuras líneas de investigación:** El prototipo desarrollado servirá como base para futuras investigaciones, como la generación de vídeos a partir de texto, la mejora de la precisión semántica en modelos generativos o la combinación de información multimodal (texto, audio, imagen) en sistemas CBIR.

En definitiva, este proyecto se enmarca en una tendencia creciente hacia sistemas más inteligentes, adaptativos y centrados en el usuario, donde la comprensión semántica del lenguaje natural y su traducción a contenido visual de alta calidad se convierten en un componente clave para la innovación tecnológica.

2. Planificación inicial del trabajo

El desarrollo del presente Trabajo Fin de Grado se ha planteado inicialmente según una planificación estructurada a tres niveles: temporal, de recursos y económico. Esta planificación se organizó siguiendo un modelo en cascada, en el que cada fase se apoya en los resultados de la anterior.

2.1. Planificación temporal

El desarrollo del proyecto se ha llevado a cabo a lo largo de varios meses, iniciándose en septiembre y finalizando en junio, siguiendo una estructura secuencial por fases:

2.1.1. Fase 1: Recolección y análisis de requisitos

(septiembre – octubre) Esta primera fase se centra en la recolección y el análisis de los requisitos del sistema, estableciendo una base para el posterior desarrollo.

- **Creación de un documento de requisitos:** Se elaborará un documento detallado que incluya tanto los requisitos funcionales como los no funcionales del sistema. Este se usará como referencia para el resto de etapas del proyecto.
- **Redacción de historias de usuario:** Para capturar los requisitos desde la perspectiva del usuario final, se redactarán historias de usuario detalladas que describan las funcionalidades deseadas. Estas ayudarán a mantener el enfoque en el objetivo del usuario durante todo el desarrollo.
- **Diagramas de casos de uso:** Se crearán diagramas de casos de uso para representar gráficamente las interacciones entre los usuarios y el sistema, asegurando una comprensión clara de los flujos de trabajo y ayudando a la identificación de posibles brechas en los requisitos.

2.1.2. Fase 2: Diseño

(noviembre – diciembre) En la segunda etapa se desarrolla el diseño detallado del sistema, asegurando el cumplimiento de los requisitos especificados durante la etapa anterior.

- **Definición de la arquitectura del sistema:** Se decidirá la estructura general del sistema, incluyendo la elección de tecnologías, plataformas y patrones de diseño. Esto garantiza que el sistema soporte los requisitos funcionales y no funcionales establecidos.
- **Creación de diagramas de clases:** Se desarrollarán diagramas de clases detallados que muestren la estructura del sistema, incluyendo las relaciones entre las distintas clases y sus atributos y métodos. Estos proporcionarán una vista clara de cómo los diferentes componentes del sistema interactúan entre sí.
- **Desarrollo de wireframes:** Se diseñarán wireframes que representen esquemáticamente la interfaz de usuario. Los wireframes realizados se podrán utilizar como guía visual inicial, mostrando la disposición de los elementos en la pantalla.
- **Diagramas de flujo:** Se crearán diagramas de flujo para representar la interacción del usuario con el sistema, desde la navegación entre pantallas hasta la ejecución de las diferentes funciones.
- **Prototipo funcional:** Se desarrollará un prototipo interactivo que simule la apariencia y el comportamiento del producto final, permitiendo tanto la realización de pruebas de usabilidad como la validación del diseño antes del desarrollo. Proporciona la oportunidad de realizar ajustes antes de realizar otras etapas más costosas.

2.1.3. Fase 3: Implementación y experimentación

(enero – abril) Esta fase se centra en la implementación del sistema, transformando el diseño detallado en las etapas anteriores en un código funcional.

- **Escrutura del código:** Se procederá a la codificación de los diferentes componentes del sistema siguiendo las especificaciones y el diseño definidos. Se aplicarán los estándares establecidos para garantizar la calidad y la consistencia del código.
- **Integración de componentes:** Se integrarán los módulos y componentes del sistema, asegurando que funcionen de manera cohesiva y que se comuniquen correctamente entre sí. Será testeada mediante tests de integración para verificar su funcionamiento.
- **Revisión del código:** Se realizará una revisión exhaustiva del código desarrollado utilizando, si es necesario, herramientas automatizadas para asegurar su calidad, mantenibilidad a largo plazo y el cumplimiento de los requisitos y estándares.

2.1.4. Fase 4: Integración, pruebas y validación

(mayo) En esta última fase del proyecto se verifican y validan los componentes para asegurar que se cumplen los requisitos establecidos.

- **Desarrollo de un plan de pruebas:** Se elaborará un plan detallado que especificará los tipos de pruebas a realizar, los casos a ejecutar y los criterios de aceptación. Garantizará una cobertura de todas las funcionalidades y escenarios posibles.
- **Realización de pruebas unitarias:** Se llevarán a cabo pruebas unitarias para validar cada componente individual del sistema. Estas se centran en verificar que cada unidad de código funcione según lo esperado y que los resultados sean consistentes.
- **Pruebas de integración:** Se realizarán pruebas de integración para asegurar que los componentes funcionan correctamente en conjunto. Esto incluye la verificación de la comunicación entre los módulos y de la transferencia de datos.
- **Identificación y corrección de errores:** Se identificarán y corregirán los errores encontrados durante las pruebas. Estos problemas serán documentados y corregidos.

2.1.5. Fase 5: Documentación y entrega

(mayo – junio): Durante esta fase se abordó la redacción completa de la memoria del proyecto, estructurando los contenidos técnicos y experimentales en capítulos claros y coherentes. Se organizaron todas las evidencias generadas a lo largo del desarrollo —gráficas de pérdidas, ejemplos de imágenes generadas, comparativas entre modelos y configuraciones— para respaldar las conclusiones obtenidas. Además, se redactaron secciones críticas como la justificación metodológica, los aprendizajes derivados de los experimentos y las limitaciones encontradas. Paralelamente, se preparó el material para la defensa, incluyendo una presentación visual, guion de exposición y demostraciones, asegurando que todo el proceso fuera comunicable tanto a perfiles técnicos como no especializados.

2.1.6. Cronograma visual del proyecto

La Figura 3 presenta el cronograma visual del proyecto, estructurado por tareas y fases temporales. El eje vertical representa las tareas ordenadas según su aparición a lo largo del desarrollo, mientras que el eje horizontal refleja el avance temporal, dividido en cinco fases distribuidas entre los meses de septiembre de 2024 y junio de 2025.

Cada barra coloreada indica el periodo estimado de ejecución de una tarea específica, permitiendo visualizar su inicio, duración y posible solapamiento con otras actividades. Esta representación permite identificar claramente los bloques de trabajo más intensivos, así como la transición progresiva entre análisis, diseño, implementación, pruebas y documentación. Las tareas están agrupadas lógicamente: desde la creación de

requisitos y diagramas al inicio, pasando por la escritura del código y su integración, hasta la redacción final de la memoria y preparación de la defensa.

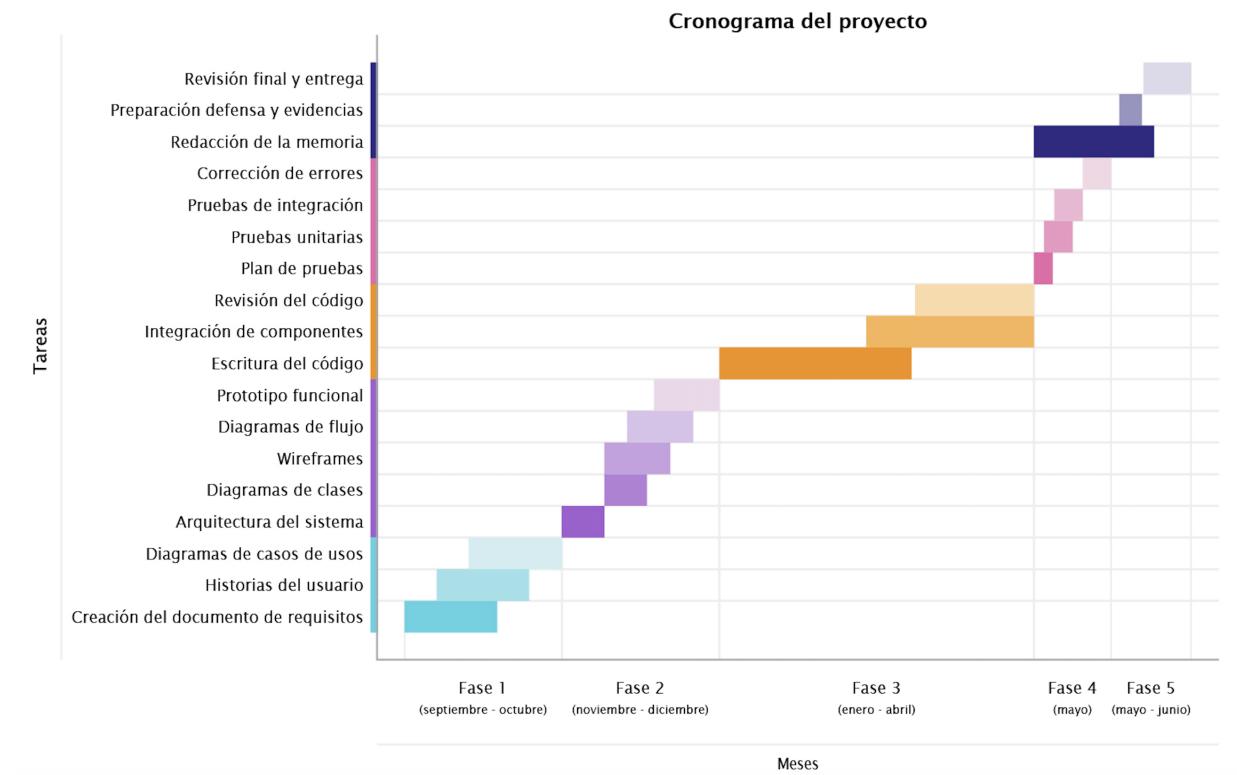


Figura 3: Cronograma visual del proyecto, agrupado por fases y tareas.

2.2. Planificación de recursos

Recursos humanos:

- Alumna: Carlota de la Vega Soriano, encargada de todo el ciclo de vida del desarrollo.
- Tutor académico: Jesús Chamorro Martínez, apoyo en la supervisión y validación del enfoque técnico y científico.

Recursos técnicos:

- Ordenador personal (MacBook Pro 2019, CPU sin GPU dedicada).
- Plataformas de computación en la nube: Kaggle (30h/semana con GPU gratuita) y Google Colab Pro (100 unidades informáticas).
- Librerías y frameworks: PyTorch, TensorFlow, Keras, diffusers, transformers, Matplotlib, entre otras.
- Datasets utilizados: MNIST, CIFAR-10/100, COCO, Stanford Dogs.

Recursos software adicionales:

- Google Drive para gestión de datasets y modelos.
- Jupyter Notebooks y entornos virtuales gestionados con Poetry.
- Repositorios oficiales de Hugging Face y GitHub para pruebas con modelos preentrenados.

2.3. Planificación económica

El presupuesto inicial estimado fue el siguiente:

Gastos elegibles	Unidades	Coste por unidad	Importe solicitado
GASTOS DE PERSONAL			
Total gastos de contratación de personal	1	10.71€/h	4284€
GASTOS DE EJECUCIÓN			
Ordenador (Precio: 2462€)	1	118.37€	118.37€
Servidor para ejecución	1	0.899€/h	359.60€
Costes indirectos (10 % presupuesto total)			476.20€
Total incentivo solicitado			5238.17€

Tabla 1: Presupuesto del proyecto

2.4. Ajustes realizados sobre la planificación inicial

Durante el desarrollo del trabajo, se han producido diversos ajustes sobre la planificación inicial, que reflejan tanto dificultades técnicas como decisiones informadas para optimizar el rendimiento:

- **Cambio de framework:** aunque inicialmente se planteó el uso de TensorFlow, se decidió migrar a PyTorch para trabajar con AttnGAN y Stable Diffusion de forma más estable y con mejor soporte en la comunidad investigadora.
- **Uso de modelos preentrenados:** debido a las limitaciones computacionales, especialmente en Google Colab, se optó en parte por utilizar versiones preentrenadas de modelos como AttnGAN y Stable Diffusion (v1.5 y XL), evaluando su rendimiento sin necesidad de entrenamiento completo desde cero.
- **División del dataset COCO:** se decidió trabajar con subconjuntos seleccionados aleatoriamente debido a restricciones de memoria en entornos gratuitos.
- **Cambio de estrategia de representación textual:** se probaron distintas técnicas (One-Hot, Word2Vec), pero se optó finalmente por utilizar las descripciones originales del dataset COCO preprocesadas, mejorando así la eficiencia y preservando el contenido semántico.
- **Evaluación iterativa:** se optó por entrenamientos cortos con cambios progresivos en hiperparámetros, en lugar de una única sesión extensa, para facilitar el análisis comparativo de resultados.

Estas adaptaciones han permitido mantener la viabilidad del proyecto sin comprometer sus objetivos, y han fomentado la toma de decisiones técnicas fundamentadas, muy valiosas en un entorno profesional real.

3. Metodología

La metodología seguida en este proyecto se ha estructurado en fases bien definidas, siguiendo un modelo en cascada adaptado, con componentes iterativos durante la implementación y evaluación de modelos. Esta elección se justifica por la necesidad de disponer de un marco claro y planificado que permitiera abordar la complejidad técnica del problema, sin renunciar a la flexibilidad necesaria en tareas de investigación experimental. El enfoque ha combinado la planificación estructurada con decisiones ágiles en momentos clave, permitiendo adaptarse a las limitaciones computacionales y a la evolución natural del trabajo técnico.

3.1. Enfoque metodológico

La estructura metodológica constó de las siguientes etapas:

- **Análisis:** Revisión del estado del arte en CBIR, TBIR y generación de imágenes a partir de texto. Se investigaron tanto los fundamentos teóricos como implementaciones reales de modelos generativos, sistemas de recuperación de imágenes y técnicas de preprocesamiento de texto.
- **Diseño:** Selección de arquitecturas candidatas (GAN, cGAN, AttnGAN, Stable Diffusion), definición de criterios de evaluación (precisión visual, coherencia semántica, coste computacional), selección de datasets y definición de flujos de trabajo para comparar cada modelo en condiciones controladas.
- **Implementación:** Desarrollo de experimentos con múltiples modelos generativos. Cada arquitectura se entrenó o adaptó con un conjunto de datos específico. Se incluyó registro exhaustivo de parámetros, tiempos de ejecución, fallos y observaciones para garantizar la trazabilidad del proceso.
- **Evaluación:** Los resultados fueron analizados mediante métodos cualitativos (evaluación visual, revisión semántica de correspondencia texto-imagen, análisis de fidelidad perceptiva) y cuantitativos ligeros (evolución de la función de pérdida). Se comparó la calidad visual, la estabilidad del modelo, la sensibilidad a la semilla y el tiempo de respuesta.

3.2. Herramientas y tecnologías utilizadas

Dado el entorno académico y las restricciones de acceso a hardware dedicado, se priorizó el uso de herramientas flexibles, reproducibles y ampliamente utilizadas en la comunidad científica:

- **Lenguaje de programación:** Python 3.10, por su compatibilidad con la mayoría de librerías de deep learning modernas y por su amplio soporte para prototipado rápido y visualización.
- **Frameworks:** PyTorch fue el framework principal, por su facilidad para modificar arquitecturas, acceso a modelos preentrenados y mejor integración con herramientas modernas (como Hugging Face Diffusers). TensorFlow se usó exclusivamente en etapas tempranas para pruebas con cGAN. FastAPI se empleó para exponer los modelos generativos mediante una API REST, facilitando su integración con plataformas externas como JMR.
- **Plataformas de ejecución:**
 - **Google Colab Pro:** se utilizó para entrenamientos intensivos de corta duración y ejecución de modelos preentrenados como Stable Diffusion. Su acceso a GPUs potentes (como T4 o A100) permitió ejecutar ciclos de entrenamiento y generación de imágenes en tiempos razonables. Además, el entorno ofrecía soporte para cuadernos colaborativos, lo cual facilitó el desarrollo iterativo y la compartición de experimentos.
 - **Kaggle:** se empleó como entorno complementario, especialmente útil para la ejecución por lotes y el entrenamiento de modelos menos exigentes como AttnGAN. A pesar de la limitación semanal de tiempo en GPU (30 horas), su integración con notebooks versionables y datasets alojados facilitó el control de experimentos. También resultó útil para entrenar modelos con datasets más pesados sin necesidad de configuraciones locales complejas.

- **Local:** el desarrollo y pruebas ligeras se realizaron en un MacBook Pro de 2019, sin GPU dedicada. Esta limitación motivó la implementación de flujos de trabajo optimizados, con procesamiento por lotes pequeños, validación parcial de scripts y uso intensivo de mock datasets para pruebas funcionales. Asimismo, se aprovechó este entorno para tareas de preprocesamiento textual, visualización de resultados y despliegue de pruebas de API mediante FastAPI.
- **Gestión de entornos:** Uso de Poetry para la creación de entornos virtuales aislados y control exacto de versiones de librerías, lo cual permitió reproducir los experimentos a lo largo del tiempo y evitar conflictos de dependencias.
- **Visualización:** Matplotlib y Seaborn se emplearon para representar gráficas de pérdidas, evolución de métricas y comparación de resultados. También se usó PIL para el procesamiento y visualización de imágenes generadas.
- **Diseño de interfaz:** Figma fue utilizado para prototipar la integración visual de la aplicación generativa en la plataforma JMR, facilitando el diseño de la experiencia de usuario y la validación temprana de flujos.

3.3. Datasets utilizados

Se utilizó una variedad de datasets, seleccionados en función del objetivo del experimento (baja, media y alta complejidad):

- **MNIST:** dataset sencillo de imágenes de dígitos manuscritos. Se utilizó para validar el funcionamiento básico de arquitecturas GAN y cGAN y observar el comportamiento en entornos sin complejidad semántica.
- **CIFAR-10 y CIFAR-100:** se emplearon para trabajar con imágenes a color y múltiples clases. Permitieron experimentar con generación condicional usando etiquetas y observar la capacidad de los modelos para diferenciar categorías con ruido limitado.
- **COCO:** dataset clave para tareas de texto a imagen. Gracias a sus múltiples captions por imagen, fue ideal para entrenar modelos como AttnGAN o probar prompts en Stable Diffusion. Se seleccionaron subconjuntos balanceados debido a limitaciones computacionales.
- **Stanford Dogs:** conjunto especializado, con alto grado de variación visual intra-clase. Se utilizó para comprobar si los modelos preentrenados eran capaces de capturar detalles finos cuando el prompt incluía atributos específicos (raza, color, postura).

3.4. Modelos y arquitecturas exploradas

Se probaron cuatro enfoques principales:

- **GAN:** arquitectura básica para familiarización con la estructura generador-discriminador. Se entrenó desde cero con MNIST.
- **cGAN:** variante condicional en la que el generador recibe una etiqueta como entrada. Se usó con CIFAR-10 para verificar la generación dirigida. Las pruebas evidenciaron inestabilidad y poca precisión sin ajustes complejos.
- **AttnGAN:** modelo avanzado con mecanismo de atención para mapear palabras del texto a regiones de la imagen. Se utilizó versión preentrenada, con pruebas sobre COCO. Los resultados mejoraron notablemente la coherencia semántica con respecto a cGAN.
- **Stable Diffusion:** modelo basado en difusión latente, altamente estable y con gran capacidad semántica. Se utilizó en dos versiones (v1.5 y XL) para generar imágenes a partir de descripciones tanto simples como especializadas. Se evaluaron tiempo de generación, coherencia semántica y calidad visual.

3.5. Justificación técnica y decisiones adoptadas

Durante la ejecución se tomaron decisiones clave para mantener la viabilidad técnica sin comprometer la profundidad experimental:

- **Cambio de framework:** la elección de PyTorch frente a TensorFlow permitió trabajar con arquitecturas como AttnGAN y acceder a modelos y scripts actualizados con mejor documentación y soporte.
- **Uso de modelos preentrenados:** dado que entrenar desde cero modelos como AttnGAN o SD requiere semanas en entornos con GPU, se optó por usar pesos preentrenados oficiales, garantizando consistencia en los resultados y permitiendo centrar el análisis en la interacción texto-imagen.
- **Reducción de tamaño de datasets:** para entrenamientos rápidos y comparaciones ágiles, se crearon subconjuntos de COCO con variedad semántica. Esto permitió obtener resultados interpretables en menos tiempo.
- **Selección de prompts:** en las pruebas con SD se diseñaron prompts de distinta longitud, estilo y precisión para evaluar cómo el modelo respondía ante distintos niveles de detalle lingüístico.
- **Evaluación iterativa:** cada prueba se acompañó de visualización de resultados, almacenamiento de outputs, y revisión subjetiva por parte de la autora y otros usuarios, para estimar la calidad percibida y la fidelidad del modelo al texto.
- **Documentación del proceso:** todas las pruebas fueron registradas (scripts, configuraciones, checkpoints, ejemplos generados) para garantizar la reproducibilidad de los resultados y facilitar el análisis posterior en la memoria.

Parte I

Desarrollo de modelos

4. Estado del Arte

4.1. Antecedentes

El desarrollo de este proyecto se enmarca en un contexto en el que la cantidad de contenidos multimedia ha crecido exponencialmente, especialmente en plataformas digitales como redes sociales, bancos de imágenes o repositorios científicos. Esta disponibilidad masiva de imágenes ha motivado la creación de sistemas de recuperación visual que ayuden al usuario a encontrar contenido relevante de forma rápida y precisa.

Además, la evolución del aprendizaje automático ha permitido aplicar técnicas de análisis visual más avanzadas, así como métodos para procesar lenguaje natural y generar contenido de manera automática. Esta evolución tecnológica sienta las bases del enfoque adoptado en el presente trabajo.

A partir de estos antecedentes, se presenta a continuación una revisión detallada de las principales líneas de investigación actuales relacionadas con la recuperación visual basada en contenido, el procesamiento de lenguaje natural y los modelos generativos condicionados por texto.

4.2. Sistemas de Recuperación de Imágenes por Rasgos Visuales (CBIR)

La recuperación de imágenes basada en contenido es un campo de la recuperación de información enfocado en la recuperación de imágenes basada en las características visuales de estas. En CBIR, el usuario selecciona una imagen de referencia y obtiene como resultado las imágenes de una base de datos de datos similares a la introducida.

4.2.1. Definición

Los sistemas CBIR determinan las imágenes más similares analizando el contenido visual de la imagen de referencia y comparándolo con las imágenes almacenadas en la base de datos. Específicamente, CBIR evalúa características visuales como formas, colores, texturas e información espacial para calcular la similitud entre la imagen de consulta y las imágenes de la base de datos en función de estas propiedades [7].

4.2.2. Características

En los sistemas de Recuperación de Imágenes Basadas en Contenido (CBIR), las características visuales desempeñan un papel fundamental en la representación y comparación de las imágenes. Estas pueden ser clasificadas en dos categorías principales:

- **Características globales:** Estas describen la imagen en su totalidad, proporcionando información sobre elementos que abarcan toda la imagen, como la distribución general de colores, la forma global y la textura general.
- **Características locales:** En contraste, las características locales describen patrones o estructuras específicas dentro de la imagen. Se centran en áreas más pequeñas de la imagen y pueden incluir detalles como puntos de interés, bordes, texturas locales y puntos clave.

La combinación de características globales y locales permite una representación más completa y detallada de la imagen, lo que facilita la comparación y recuperación de imágenes basada en sus atributos visuales.

4.2.3. Medidas de similitud

Una vez que se han extraído las características de las imágenes, es necesario evaluar la similitud entre las imágenes de consulta y las imágenes almacenadas en la base de datos. Para lograr esto, se utilizan medidas de similitud, que se pueden dividir en dos tipos:

- **Distancia.**

- Las medidas de distancia cuantifican la diferencia o disimilitud entre dos vectores de características. La distancia se calcula como la magnitud del desplazamiento necesario para ir desde un punto hasta otro en un espacio métrico. En el contexto de CBIR, las imágenes más similares a la imagen de consulta son aquellas cuya distancia a la imagen de consulta es más pequeña.

- Algunas medidas de distancia comunes incluyen la distancia euclídea, la distancia de Mahalanobis y la distancia de Manhattan. Estas medidas se utilizan para calcular la diferencia entre los valores de características de dos imágenes y determinar su grado de similitud.

■ Métricas de similitud.

- Las métricas de similitud, por otro lado, cuantifican la similitud entre dos vectores de características. Estas métricas miden la similitud angular o la correlación entre los vectores de características y proporcionan una medida de la proximidad entre las imágenes.
- Una métrica de similitud común es la similitud del coseno, que mide el coseno del ángulo entre dos vectores de características. Cuanto más cercano sea el valor del coseno a 1, mayor será la similitud entre las imágenes.

4.2.4. Aplicaciones

El sistema de Recuperación de Imágenes Basada en Contenido ofrece una variedad de aplicaciones en distintos sectores. En la industria de la moda, optimiza la búsqueda de productos, permitiendo a los usuarios encontrar prendas y accesorios similares a partir de una imagen, lo que mejora la experiencia de compra en línea y la recomendación de productos. Además, en seguridad y vigilancia, el CBIR facilita la re-identificación de individuos en diferentes escenarios, utilizando características visuales como la ropa y el peinado para rastrear a personas específicas.

Por otro lado, en el ámbito del comercio electrónico, mejora la navegación y selección de productos al permitir a los usuarios descubrir artículos similares basados en una imagen de referencia, aumentando así la satisfacción del cliente y las ventas en línea. Asimismo, en aplicaciones de teledetección y análisis geoespacial, el CBIR es útil para el análisis de imágenes satelitales, ayudando en la identificación y clasificación de características terrestres y cambios ambientales. [1]

Estos ejemplos de aplicaciones exponen la versatilidad del Sistema de Recuperación de Imágenes Basada en Contenido. Su capacidad para mejorar la experiencia del usuario, optimizar la búsqueda de productos y facilitar análisis detallados en teledetección resalta su importancia en la transformación digital actual.

4.2.5. Diferencias entre TBIR y CBIR

En el ámbito de la recuperación de imágenes, existen dos enfoques principales que difieren significativamente en su funcionamiento: la Recuperación de Imágenes Basada en Texto (Text-Based Image Retrieval, TBIR) y la Recuperación de Imágenes Basada en Contenido (Content-Based Image Retrieval, CBIR). Ambos tienen como objetivo encontrar imágenes relevantes, pero lo hacen desde perspectivas distintas.

El enfoque TBIR se basa en la utilización de anotaciones textuales asociadas a las imágenes. Estas pueden ser palabras clave, frases o descripciones completas, y suelen ser generadas manualmente por expertos. Por tanto, la calidad de la recuperación depende en gran medida de la precisión de estas anotaciones y de la interpretación semántica del lenguaje. Esto implica una alta dependencia de la intervención humana, lo que puede generar problemas de subjetividad, ambigüedad o inconsistencias entre distintas personas.

Por el contrario, CBIR prescinde de anotaciones y basa su funcionamiento en el análisis automático del contenido visual de las imágenes. Este sistema extrae características como colores, formas, texturas o patrones espaciales, y utiliza medidas de similitud para comparar imágenes entre sí. Esta aproximación es más objetiva y escalable, ya que permite procesar grandes volúmenes de imágenes sin necesidad de intervención manual.

En términos de eficiencia, CBIR es generalmente más rápido una vez construido, puesto que evita el costoso proceso de etiquetado. Además, al estar basado en propiedades visuales directamente extraídas de la imagen, puede alcanzar niveles de precisión más consistentes y reproducibles, aunque en ciertos casos puede carecer de la riqueza semántica que aporta el lenguaje natural en TBIR.

Ambos enfoques presentan ventajas y limitaciones, y su elección depende del contexto de aplicación: TBIR puede ser más intuitivo para el usuario en sistemas donde se dispone de descripciones detalladas, mientras que CBIR resulta más útil en entornos donde la anotación manual no es viable o se requiere automatización a gran escala.

Característica	TBIR	CBIR
Método de búsqueda	Utiliza palabras clave, anotaciones o descripciones textuales	Analiza características visuales como formas, colores, texturas e información espacial
Proceso de anotación	Anotación manual por expertos	No requiere anotación manual, analiza el contenido visual directamente
Dependencia humana	Alta dependencia de percepciones e interpretaciones humanas	Menos dependencia de la interpretación humana, más objetivo
Fiabilidad de etiquetas	Puede ser menos fiable debido a variaciones en la interpretación humana	Mayor fiabilidad al comparar características visuales objetivas
Eficiencia	Menos eficiente debido al tiempo requerido para la anotación manual	Más eficiente al analizar directamente el contenido visual
Escalabilidad	Puede ser menos escalable debido al trabajo manual requerido	Más escalable ya que no depende de la anotación manual
Precisión en la recuperación	Varía según la precisión de las anotaciones y etiquetas	Alta precisión al basarse en características visuales

Tabla 2: Comparativa TBIR vs CBIR

4.3. Descriptores visuales de bajo nivel

En el contexto de la recuperación de información visual, los descriptores constituyen una herramienta fundamental para representar el contenido de una imagen de forma computacionalmente interpretable. Su propósito es establecer una correspondencia entre los píxeles que conforman la imagen y los conceptos visuales que los humanos pueden percibir, clasificar y comparar. Esta representación intermedia es clave en tareas como la búsqueda de imágenes por similitud, la clasificación automática o la generación de anotaciones semánticas.

Siguiendo el estándar MPEG-7, ampliamente adoptado para la descripción de contenidos multimedia, los descriptores visuales se agrupan en dos categorías principales:

- **Descriptores de información general:** engloban características visuales de bajo nivel como el color, la forma, la textura o la distribución espacial de estos atributos. Son independientes del dominio y pueden aplicarse a cualquier tipo de imagen.
- **Descriptores de información específica de dominio:** están orientados a representar información sobre objetos, escenas o eventos particulares, incluyendo personas, acciones o relaciones espaciales concretas entre elementos de la imagen.

Dado que este trabajo se enmarca en el desarrollo de un sistema de recuperación y generación de imágenes de propósito general, se ha optado por utilizar descriptores de información general, y en particular aquellos centrados en el análisis del color. Esta elección se justifica tanto por su bajo coste computacional como por su elevada capacidad discriminativa en tareas de recuperación por similitud perceptual.

4.3.1. Representación visual según MPEG-7

MPEG-7 (Multimedia Content Description Interface) es un estándar desarrollado por el grupo MPEG del consorcio ISO/IEC, diseñado específicamente para permitir la descripción estructural y semántica de

contenidos audiovisuales. Proporciona un marco unificado para representar tanto metadatos de catálogo (como título o autor), como propiedades visuales y auditivas extraídas automáticamente.

Dentro de este marco, los descriptores visuales de bajo nivel permiten codificar numéricamente aspectos como el color, la forma o la textura de una imagen. Esta codificación facilita tareas de indexación, búsqueda, filtrado o clasificación de contenidos visuales a gran escala.

A continuación se describen los principales descriptores de color definidos en MPEG-7 y utilizados en este proyecto.

4.3.2. Descriptor de color dominante

Este descriptor proporciona una representación concisa de los colores más relevantes presentes en una imagen o región. A diferencia de un histograma tradicional que considera la frecuencia absoluta de todos los colores, el descriptor de color dominante selecciona un número reducido de colores representativos, junto con información adicional que permite caracterizar su distribución y coherencia espacial.

Se define mediante una colección de tuplas:

$$F = \{(C_i, P_i, V_i), s\}, \quad i = 1, 2, \dots, N$$

donde:

- C_i : color dominante representado como un vector en un espacio de color determinado (por ejemplo RGB o YCbCr).
- P_i : proporción de píxeles que presentan el color C_i .
- V_i : variación del color respecto al dominante (opcional).
- s : medida de coherencia espacial del color en la imagen (opcional).

Este descriptor es útil en aplicaciones donde el color global o los tonos dominantes son claves para la recuperación, como en catálogos de productos, arte o publicidad visual.

4.3.3. Descriptor de color escalable

El descriptor de color escalable representa la distribución de color global mediante un histograma codificado de forma jerárquica, lo que permite un ajuste progresivo de la precisión frente al coste computacional. Está basado en la aplicación de una transformada de Haar sobre el histograma de color generado en el espacio HSV.

Su diseño permite trabajar a diferentes niveles de detalle, siendo especialmente adecuado para sistemas que operan con distintas capacidades de almacenamiento o procesamiento. A mayor número de bits utilizados, mayor precisión en la representación y en la recuperación de imágenes similares.

4.3.4. Descriptor de estructura de color

Este descriptor extiende el concepto de histograma de color al incorporar información sobre la disposición espacial de los colores. Para ello, aplica una ventana deslizante sobre la imagen y contabiliza no solo la presencia de colores, sino también su estructura local.

Captura, por tanto, tanto la distribución global como los patrones espaciales de color, lo que le permite diferenciar imágenes que comparten la misma proporción cromática pero presentan composiciones visuales diferentes. Esta propiedad lo convierte en una herramienta potente para tareas de clasificación y búsqueda con sensibilidad estructural.

En resumen, los descriptores visuales definidos por MPEG-7 proporcionan una base sólida para representar imágenes de forma eficiente y discriminativa. En el marco de este trabajo, se han seleccionado y utilizado aquellos basados en el color por su equilibrio entre expresividad, coste computacional y aplicabilidad general. Su uso ha sido clave tanto para tareas de recuperación de imágenes como para establecer representaciones intermedias en los procesos de generación asistida por IA.

4.4. Redes Neuronales

En el ámbito de la inteligencia artificial y el aprendizaje automático, las redes neuronales han surgido como una potente herramienta para modelar y analizar datos complejos. Representan una vanguardia en estos campos, impulsando innovaciones tecnológicas y transformando la manera en que interactuamos con la información y el mundo que nos rodea.

4.4.1. Fundamentos

Las redes neuronales se basan en la simulación de la estructura y el funcionamiento del cerebro humano para procesar información [3]. Están compuestas por un conjunto de nodos conectados entre sí que reciben y procesan información y generan una salida en consecuencia. Estos nodos, llamados neuronas artificiales, pueden ser ajustados para mejorar la precisión de la salida, permitiendo así un aprendizaje y adaptación de la red.

La estructura de una red neuronal se compone de múltiples capas de neuronas interconectadas entre sí. Estas capas incluyen una capa de entrada, que recibe los datos iniciales y los introduce en la red, y una capa de salida, que produce el resultado final o la predicción deseada. Además, entre la capa de entrada y la capa de salida, puede haber una serie de capas ocultas que desempeñan un papel crucial en el procesamiento y la transformación de la información a medida que se propaga a través de la red.

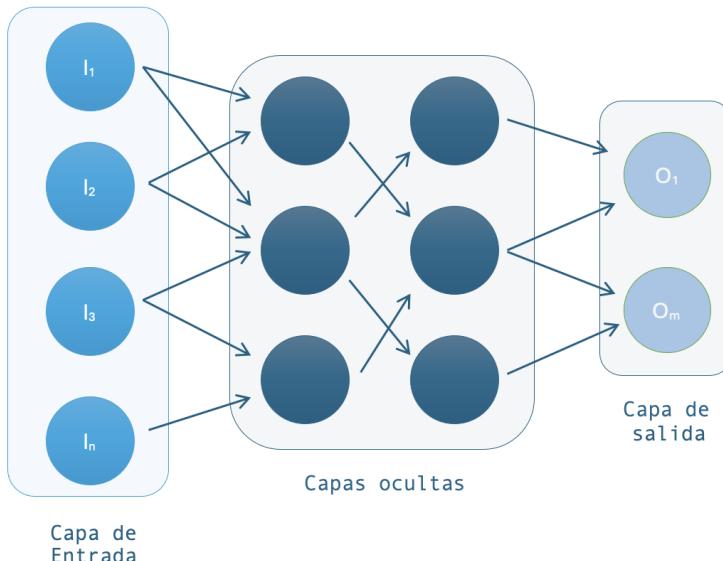


Figura 4: Estructura de una red neuronal

Cada neurona en la red está conectada con otras neuronas mediante conexiones, similares a las sinapsis en el sistema nervioso biológico. Estas conexiones determinan la fuerza y la influencia de la información transmitida entre las neuronas. Durante el proceso de entrenamiento de la red neuronal, estos pesos sinápticos se ajustan y optimizan continuamente a través de algoritmos de aprendizaje con el objetivo de mejorar la precisión y la eficiencia de la red en la tarea específica para la que ha sido diseñada.

El proceso de entrenamiento es un ciclo iterativo que implica alimentar la red con datos de entrada y comparar la salida producida con la salida esperada. Si existiese una discrepancia entre estas, se realizarían ajustes en los pesos de las conexiones.

4.4.2. Funciones de activación

Las funciones de activación son una parte crucial del funcionamiento y la capacidad de aprendizaje de las redes neuronales. Estas funciones determinan la salida de cada neurona en función de la entrada que recibe,

proporcionando una forma de introducir no linealidad en el modelo y permitir a la red aprender y modelar realciones complejas en los datos.

Existen dos tipos principales de funciones de activación: lineales y no lineales. Las funciones lineales son más simples y se limitan a multiplicar la entrada por un peso y sumar un término de sesgo, lo que resulta en una transformación lineal de los datos. Por otro lado, las funciones no lineales introducen la no linealidad, permitiendo a la red capturar y representar relaciones más complejas entre las variables de entrada y salida.

Algunas de las funciones de activación más comunes utilizadas en las redes neuronales son Sigmoid, ReLu y Tanh.

- La **función Sigmoid** tiene una forma característica de “S” y se define matemáticamente como

$$f(x) = \frac{1}{1 + \exp(-x)} \quad (1)$$

Esta función produce una salida en el rango de 0 a 1, lo que la hace especialmente útil en problemas de clasificación binaria donde se busca predecir una probabilidad entre dos clases.

- La **función ReLu** es una de las funciones de activación más populares en las redes neuronales modernas. Se define como

$$f(x) = \max(0, x) \quad (2)$$

lo que significa que la función retorna cero para valores negativos y la misma entrada para valores positivos.

- La **función Tanh** es similar a la Sigmoid, pero produce una salida en el rango de -1 a 1. Matemáticamente se define como

$$f(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \quad (3)$$

Esta función es útil en problemas de clasificación binaria y regresión donde las salidas pueden ser negativas.

4.4.3. Tipos

Cada red neuronal está diseñada para atender requisitos diferentes en el proceso de datos según las necesidades del usuarios. Las más prominentes en aplicaciones actuales son las redes feedforward, recurrentes y convolucionales.

- Las **redes feedforward** representan el modelo más básico de red neuronal. En este tipo de arquitectura, la información se mueve en una dirección única, desde la entrada hasta la salida, sin la presencia de ciclos o conexiones retroalimentadas. Cada capa procesa la información y la transmite a la siguiente capa, siguiendo este flujo unidireccional hasta obtener la salida final. Este diseño de red es especialmente adecuado para aplicaciones de clasificación y predicción, como el reconocimiento de imágenes o la detección de actividades fraudulentas.
- Las **redes neuronales recurrentes** (RNN) se caracterizan por tener conexiones que retroalimentan las neuronas entre sí. Esta estructura permite que la salida de una neurona sea utilizada como entrada para otra, creando así una “memoria” que conserva información sobre los datos de entrada previos. Debido a su capacidad de retención de información, las RNN son especialmente efectivas en tareas que implican el procesamiento de secuencias, como el análisis de lenguaje natural y la predicción meteorológica.
- Las **redes convolucionales** (CNN) fueron específicamente diseñadas para el procesamiento de imágenes y videos. En estas redes, las neuronas se agrupan en capas convolucionales, donde cada neurona se conecta únicamente con una región local de la capa anterior mediante una operación de convolución, definida por

$$S(i, j) = (K * I)(i, j) = \sum_m \sum_n I(m, n) \cdot K(i - m, j - n) \quad (4)$$

Esta organización permite que la CNN identifique características particulares en una imagen, como contornos y texturas, sin importar su ubicación dentro de la misma.

4.4.4. Aplicaciones

En la actualidad, las redes neuronales desempeñan un papel fundamental en la transformación de diversos campos tecnológicos, y su aplicación se extiende a una gran variedad de industrias y áreas de investigación. A medida que esta tecnología avanza, se desarrollan nuevas aplicaciones que aprovechan sus capacidades de aprendizaje y adaptabilidad para resolver problemas complejos y mejorar la eficiencia en diferentes sectores.

Por ejemplo, el transformador de lenguaje basado en redes neuronales GPT-4 ha marcado un hito significativo en el campo del procesamiento del lenguaje natural. Este modelo es utilizado en diversas plataformas y servicios, como ChatGPT, un asistente virtual para la interacción con los humanos; DALL-E, que genera imágenes únicas a partir de descripciones textuales; y Riffusion, una herramienta para la composición musical y la creación de melodías basadas en texto.



Figura 5: GPT-4

Estos ejemplos muestran la versatilidad de las redes neuronales en la actualidad, demostrando su potencial para impulsar la innovación en múltiples ámbitos.

4.5. Redes Generativas Adversarias (GAN)

Las Redes Generativas Antagónicas (GAN) son una arquitectura especializada de redes neuronales profundas que consta de dos componentes principales: el generador y el discriminador. Estas redes operan en un ciclo de retroalimentación constante, compitiendo entre sí en un proceso antagónico para generar datos sintéticos de alta calidad que sean indistinguibles de los datos reales. Este carácter antagónico y su capacidad para generar datos sintéticos versátiles hacen que las GAN sean una herramienta poderosa con aplicaciones innovadoras en diversos campos del aprendizaje automático y la inteligencia artificial. [8]

4.5.1. Componentes

Las GAN se distinguen por su arquitectura única, compuesta por dos elementos que trabajan en sinergia: el generador y el discriminador. Mientras que el generador se encarga de producir instancias de datos que imitan a los datos reales, el discriminador actúa como un crítico, evaluando y diferenciando entre los datos generados y los datos reales. Esta dinámica antagónica entre el generador y el discriminador impulsa la mejora continua y la generación de datos sintéticos cada vez más realistas.

1. Generador:

- El generador, concebido como un “artista digital”, opera en un espacio de alta dimensionalidad utilizando un conjunto inicial de números aleatorios (ruido). Este proceso puede ser visualizado como la falsificación de una obra de arte en un lienzo en blanco, donde el generador transforma hábilmente este ruido en datos representativos de la distribución de los datos reales.
- La esencia del generador reside en su capacidad para generar datos que, al ser evaluados, resulten indistinguibles de los datos reales. Este desafío intrínseco impulsa al generador a capturar de manera fiel las características inherentes al conjunto de datos de referencia.

2. Discriminador:

- El discriminador desempeña el papel de un crítico discerniente. Su tarea consiste en evaluar los datos presentados y distinguir de manera aguda entre los auténticos y aquellos generados por el modelo. Su evolución se centra en la mejora constante de esta capacidad discriminatoria.
- La meta primordial del discriminador es perfeccionar su capacidad de discernimiento entre datos reales y generados. Su agudeza se traduce en un desafío constante para el generador, estimulando un proceso de mejora continua.

4.5.2. Proceso de entrenamiento

El entrenamiento de las Redes Generativas Adversarias permite a la red aprender y perfeccionar su capacidad para generar datos sintéticos de alta calidad. En este proceso, el generador busca generar datos sintéticos que sean cada vez más indistinguibles de los datos reales, mientras que el discriminador se esfuerza por mejorar su capacidad para distinguir entre datos generados y datos reales.

1. **Inicialización.** En este paso, tanto el generador como el discriminador se preparan para el proceso de entrenamiento. Sin experiencias previas, se encuentran en un estado de latencia y preparados para el aprendizaje y la evolución.
2. **Generación y evaluación.** El generador toma una entrada de ruido aleatorio y la utiliza para generar datos falsos que se asemejen a los datos de entrenamiento reales. Estos datos generados son sometidos al juicio del discriminador, que evalúa su autenticidad, marcando el comienzo de un ciclo de retroalimentación competitiva entre el generador y el discriminador.
3. **Retroalimentación competitiva.** El discriminador proporciona retroalimentación al generador informándole sobre los errores en la generación de datos, quien la utiliza para ajustar sus parámetros de manera que los datos generados se vuelvan más realistas y mejoren su capacidad de engañar al discriminador.
4. **Iteración.** El proceso de generación, evaluación y retroalimentación se repite en ciclos sucesivos. Con cada ciclo de entrenamiento, el generador y el discriminador mejoran gradualmente su desempeño, lo que conduce a una mejora continua en la capacidad del generador para producir datos realistas y en la capacidad del discriminador para identificar lo auténtico.

En resumen, el proceso de entrenamiento en una GAN implica una competencia continua entre el generador y el discriminador. Este ciclo iterativo conduce a la convergencia hacia una distribución de datos generados que es indistinguible de la distribución de datos reales.

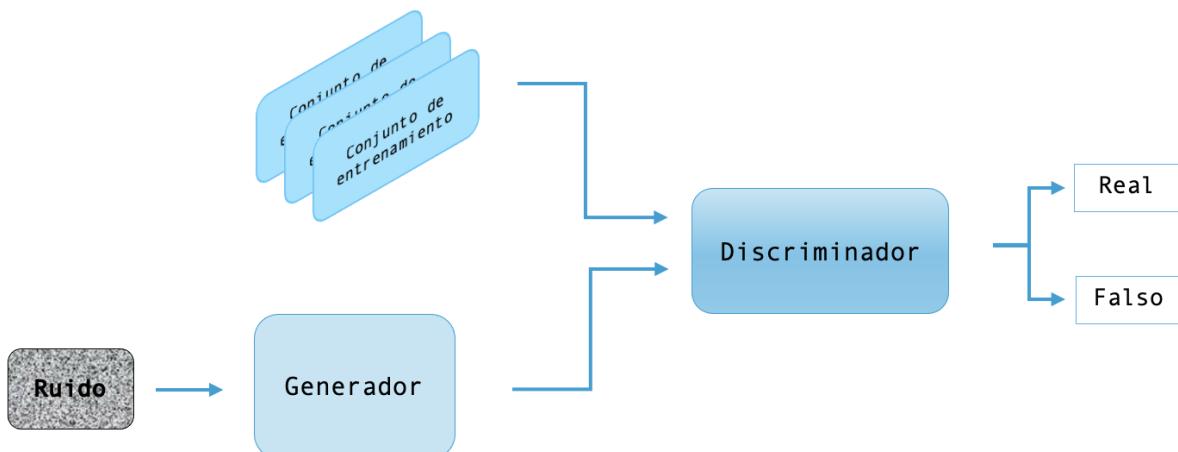


Figura 6: Proceso de entrenamiento de una GAN

4.5.3. Efectos del ruido en la generación de imágenes

En las Redes Generativas Adversarias, el ruido juega un papel crucial en el proceso de generación de imágenes, ya que sirve como entrada al generador para producir diversas representaciones a partir de una misma arquitectura. Dependiendo de la cantidad y la naturaleza del ruido inyectado, se pueden observar los siguientes efectos:

- **Diversidad de las imágenes:**

- Ruido alto: Incluir un nivel elevado de ruido (alta desviación estándar) en el generador incrementa la variabilidad de las imágenes generadas. Esto puede resultar en una mayor diversidad de imágenes, aunque con el riesgo de introducir artefactos visuales o distorsiones no deseadas.
- Ruido bajo: Al reducir el nivel de ruido (baja desviación estándar), las imágenes tienden a ser más uniformes y menos diversas. Aunque esto puede reducir la aparición de artefactos, también limita la capacidad del modelo para generar imágenes novedosas.

- **Calidad de la imagen:**

- Ruido alto: Un nivel elevado de ruido puede afectar negativamente la calidad de las imágenes generadas, haciéndolas borrosas o con artefactos visibles. Esto ocurre especialmente cuando el scheduler no logra eliminar completamente las perturbaciones en las primeras fases del proceso de denoising.
- Ruido bajo: Reducir el nivel de ruido inicial puede mejorar la claridad de las imágenes, produciendo resultados más nítidos y definidos. Sin embargo, si el ruido es demasiado bajo, el modelo podría perder capacidad para generar variedad, comprometiendo la diversidad en los resultados.

- **Capacidad de generalización:**

- Ruido alto: Un generador entrenado con entradas de ruido alto tiende a generalizar mejor, ya que se entrena a partir de una amplia gama de entradas. Esto puede mejorar su rendimiento al generar imágenes realistas en distintos contextos.
- Ruido bajo: Al trabajar con menos variabilidad en las entradas, el generador puede ajustarse demasiado a los datos de entrenamiento, reduciendo su capacidad para generalizar y producir imágenes variadas en nuevos escenarios.

- **Estabilidad del entrenamiento:**

- Ruido alto: Inyectar demasiado ruido puede hacer que el proceso de entrenamiento sea más instable, dado que el modelo debe lidiar con una mayor variabilidad en las entradas. Esto puede provocar oscilaciones en la calidad de las imágenes generadas y dificultar la convergencia.
- Ruido bajo: Reducir el nivel de ruido tiende a estabilizar el entrenamiento, ya que las entradas son más consistentes. Sin embargo, esto puede limitar la capacidad del generador para mejorar en situaciones más complejas o variadas.

En conclusión, la correcta gestión del nivel de ruido en las GANs es esencial para encontrar un equilibrio entre la diversidad de las imágenes generadas y la estabilidad del proceso de entrenamiento, lo que permite obtener imágenes de alta calidad y con buena capacidad de generalización.

4.5.4. Aplicaciones

Las Redes Generativas Adversarias han transformado diversos sectores gracias a su capacidad para generar datos y contenidos de alta calidad de manera artificial. Una de las aplicaciones más notables es la generación de imágenes realistas, donde pueden crear imágenes detalladas a partir de descripciones textuales o modificar imágenes existentes. Esto incluye tareas como la mejora de resolución de imágenes, la conversión de imágenes en blanco y negro a color, así como la creación de rostros, personajes y animales para industrias como la animación y los videojuegos.

Además, las GANs son fundamentales en la creación de datos sintéticos para el entrenamiento de modelos de aprendizaje automático. Generar datos sintéticos permite ampliar conjuntos de datos limitados o sesgados, mejorando el rendimiento de los modelos en tareas específicas. Por ejemplo, en el ámbito de la detección de fraudes pueden generar transacciones fraudulentas simuladas, permitiendo entrenar sistemas de detección de fraudes con una mayor precisión.

Otra aplicación clave es el completado de datos faltantes en conjuntos de datos incompletos. Las GANs pueden predecir y generar datos que llenen los vacíos en los conjuntos de datos, lo cual es especialmente útil en áreas como la cartografía geotérmica o la captura y almacenamiento de carbono. En estos casos pueden generar imágenes de la superficie subterránea a partir de datos topográficos limitados, proporcionando información crucial para la toma de decisiones en estos campos.

Estas aplicaciones representan solo una muestra de cómo las Redes Generativas Adversarias están impulsando avances significativos en distintas áreas, ampliando las fronteras de lo que es posible lograr mediante la inteligencia artificial y el aprendizaje profundo. Cada nueva implementación desafía los límites actuales, creando nuevas oportunidades para la innovación y el desarrollo tecnológico. [4]

4.6. Redes Generativas Adversarias Condicionales (cGAN)

Las Redes Generativas Adversarias Condicionales (cGAN) son una extensión de las GAN convencionales, introducidas en 2014 por Mehdi Mirza y Simon Osindero [9]. A diferencia de las GAN estándar, que generan imágenes basadas únicamente en ruido aleatorio, las cGAN incorporan información adicional en forma de etiquetas condicionales, lo que permite un control más específico sobre las imágenes generadas. Esta capacidad de generar imágenes condicionales hace que las cGAN sean especialmente útiles en tareas que requieren resultados personalizados o categorizados según criterios específicos.

4.6.1. Función de las Etiquetas

Las etiquetas desempeñan un papel central en las cGAN, al proporcionar información contextual durante el proceso de entrenamiento. Estas etiquetas, que pueden representar categorías, atributos u otras características, se incorporan tanto en el generador como en el discriminador, permitiendo que ambos aprendan de forma más dirigida y precisa. Las funciones principales de las etiquetas en cada componente de la red son las siguientes:

- **Condición en el Generador:** Las etiquetas se concatenan con el vector de ruido que sirve como entrada al generador. Esta combinación de ruido y etiquetas actúa como una guía explícita, especificando el tipo de imagen que el generador debe crear. Por ejemplo, si la etiqueta corresponde a una categoría como “gato” o “perro”, el generador utilizará esta información para generar una imagen que se ajuste a dicha categoría.
- **Condición en el Discriminador:** En el discriminador, las etiquetas se incorporan junto con las imágenes, permitiendo que el discriminador no solo evalúe si una imagen es real o generada, sino también si esa imagen corresponde a la etiqueta condicional proporcionada. Este proceso condiciona las decisiones del discriminador, mejorando su capacidad para detectar inconsistencias entre las imágenes generadas y las etiquetas asociadas.

Al integrar las etiquetas en ambos componentes, las cGAN permiten un nivel de control más fino sobre las imágenes generadas, lo que las convierte en una herramienta valiosa para aplicaciones donde se requiere una generación de imágenes personalizada o controlada. Esta capacidad condicional no solo mejora la flexibilidad de las GAN, sino que también amplía su aplicabilidad en campos como la creación de contenido visual, reconstrucción de imágenes y modelado tridimensional.

4.6.2. Proceso de entrenamiento

El proceso de entrenamiento de las cGAN sigue un enfoque similar al de las GAN convencionales, pero con la adición de información condicional en forma de etiquetas.

- Inicialización:** Tanto el generador como el discriminador se inicializan en un estado de latencia preparados para el aprendizaje, al igual que en las GANs convencionales.
- Generación y evaluación condicionada:** El generador toma una entrada de ruido aleatorio y las etiquetas condicionantes para generar datos falsos que se asemejen a los datos de entrenamiento reales. Estos datos generados son evaluados por el discriminador, que considera tanto la autenticidad de los datos como las etiquetas condicionantes.
- Retroalimentación competitiva condicionada:** El discriminador proporciona retroalimentación al generador no solo sobre los errores en la generación de datos sino también sobre cómo se ajustan a las etiquetas condicionantes. Esto permite que el generador ajuste sus parámetros de manera más precisa.
- Iteración condicionada:** El proceso de generación, evaluación y retroalimentación se repite en ciclos sucesivos, incorporando la información condicional en cada paso. Con cada ciclo de entrenamiento, tanto el generador como el discriminador mejoran su desempeño en relación con las etiquetas condicionantes, lo que resulta en una generación de datos más precisa y controlada.

En resumen, las cGAN añaden una capa adicional de complejidad y control al proceso de entrenamiento de las GAN, permitiendo una generación de imágenes más personalizada y dirigida mediante el uso de etiquetas condicionantes.

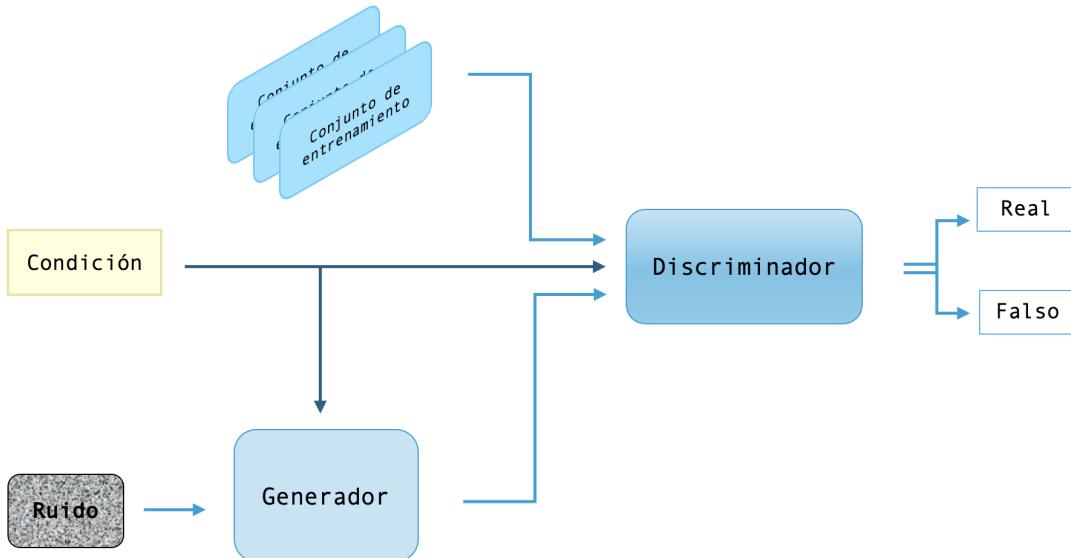


Figura 7: Proceso de entrenamiento de una cGAN

4.6.3. Aplicaciones

Las cGAN han encontrado aplicaciones en una variedad de campos debido a su capacidad para generar imágenes de manera condicional. Algunas de las aplicaciones más destacadas incluyen:

- **Traducción de Imagen a Imagen:** Las cGAN han revolucionado la traducción de imágenes con métodos como Pix2Pix, permitiendo la evolución de imágenes mediante la consideración de información adicional como etiquetas o mapas de bordes. Estas redes posibilitan la reconstrucción detallada de objetos a partir de bordes, la síntesis de fotografías de alta calidad guiadas por mapas de etiquetas, y la colorización precisa de imágenes en blanco y negro.
- **Creación de Imágenes a partir de Texto:** Es posible generar fotografías de alta calidad basadas en descripciones textuales. La riqueza del vocabulario utilizado permite la creación de imágenes sintéticas mucho más precisas y detalladas, capturando los matices y detalles específicos descritos en el texto.

- **Generación de Vídeo:** En el ámbito de la generación de vídeo, ofrecen la capacidad de predecir fotogramas futuros basados en una selección de imágenes previas. Esto es especialmente útil en aplicaciones como la interpolación de fotogramas y la generación de contenido de vídeo realista y fluido.
- **Generación de Rostros:** Estas redes pueden ser entrenadas para crear imágenes de rostros con características particulares, como el color del cabello, el color de los ojos y otros rasgos faciales, produciendo resultados realistas y detallados.

Esta flexibilidad y capacidad de adaptación hacen que las cGAN sean una herramienta poderosa en el campo del aprendizaje profundo y la generación de contenido visual.

4.7. Redes Generativas Adversarias con Atención (AttnGAN)

El modelo AttnGAN fue un gran avance en la generación de imágenes a partir de descripciones textuales. La característica clave de AttnGAN es su mecanismo de atención, que permite al modelo centrarse en diferentes partes de la descripción textual mientras genera la imagen. Este mecanismo de atención es particularmente útil para generar detalles complejos que se mencionan en las descripciones. [13]

4.7.1. Arquitectura

La arquitectura de AttnGAN se compone de múltiples niveles de generación, cada uno enfocado en un nivel diferente de detalle en la imagen. La generación se lleva a cabo en tres niveles principales:

- **Nivel de características globales:** El primer nivel se encarga de generar una versión básica de la imagen que capture la estructura global basada en la descripción textual. En este nivel, el generador utiliza un vector latente y el embedding de la frase para crear una representación inicial de la imagen.
- **Nivel de atención fina:** En el segundo nivel, el mecanismo de atención permite al generador centrarse en palabras específicas de la descripción para agregar detalles más finos a la imagen. Aquí, se utilizan representaciones de palabras para ajustar diferentes áreas de la imagen, como la textura o los colores.
- **Nivel de resolución alta:** Finalmente, el tercer nivel mejora aún más la resolución de la imagen y refina los detalles generados en los niveles anteriores, garantizando que la salida final tenga una alta calidad visual. Este proceso iterativo asegura que la imagen generada se mantenga coherente con la descripción textual proporcionada.

4.7.2. DAMSM

Un componente esencial de AttnGAN es el Deep Attentional Multimodal Similarity Model (DAMSM), diseñado para mejorar la coherencia entre las descripciones textuales y las imágenes generadas. DAMSM se entrena para alinear las representaciones de las descripciones de texto con las características de las imágenes, aprendiendo una representación conjunta de texto e imagen. Esto se logra mediante una pérdida de similitud que mide si una imagen generada se alinea con su descripción textual correctamente. El modelo intenta minimizar la distancia entre el embedding del texto y el embedding de la imagen correspondiente, ayudando al generador a producir imágenes que reflejen mejor las descripciones.

El DAMSM juega un papel crucial en el entrenamiento de AttnGAN, ya que no solo evalúa la calidad de la imagen generada, sino también cuán bien la imagen representa el contenido de la descripción. Este enfoque mejora significativamente la calidad semántica de las imágenes generadas, permitiendo que el modelo entienda descripciones más complejas y detalladas.

4.7.3. Mecanismos de atención

El concepto de atención se popularizó inicialmente en el campo del procesamiento de lenguaje natural (NLP) con el trabajo de Vaswani, titulado *Attention is All You Need*. Este enfoque demostró que, al permitir que un modelo se concentre en diferentes partes de una secuencia de entrada, se podían obtener mejores resultados en tareas como la traducción automática y el resumen de texto. [6]

La idea de la atención fue adaptada posteriormente para tareas de visión por computadora, como la generación de imágenes a partir de texto. En el caso de AttnGAN, el mecanismo de atención permite que el generador ajuste la contribución de cada palabra de la descripción en diferentes regiones de la imagen. Esto resulta en imágenes más precisas y detalladas en comparación con modelos que no utilizan atención.

4.7.4. Métricas de evaluación

Evaluar la calidad de las imágenes generadas por AttnGAN requiere el uso de métricas especializadas. Las dos métricas más comunes son el *Inception Score* (IS) y la *Fréchet Inception Distance* (FID):

- **Inception Score (IS):** Evalúa la calidad de las imágenes generadas analizando cómo una red preentrenada clasifica las imágenes. Un alto IS indica que las imágenes generadas son de alta calidad y que pertenecen a una variedad de categorías.
- **Fréchet Inception Distance (FID):** Mide la distancia entre las distribuciones de características de las imágenes generadas y las imágenes reales. A diferencia del IS, la FID es más sensible a la calidad visual de las imágenes y es capaz de detectar artefactos en las imágenes generadas. Una FID más baja indica que las imágenes generadas son más similares a las imágenes reales.

4.7.5. Proceso de entrenamiento

El proceso de entrenamiento de AttnGAN sigue una estructura iterativa similar a las cGAN, pero incorpora un mecanismo de atención para mejorar la precisión en la generación de detalles visuales basados en descripciones textuales. El entrenamiento del modelo puede desglosarse en los siguientes pasos:

1. **Inicialización:** El generador y el discriminador se inicializan para el entrenamiento. Adicionalmente, el modelo DAMSM se entrena para alinear las representaciones textuales y visuales.
2. **Generación basada en niveles de atención:** A partir de una descripción textual, el generador crea una representación inicial de la imagen que captura las características globales. Luego, mediante el mecanismo de atención, el modelo ajusta detalles específicos de acuerdo con palabras clave en la descripción.
3. **Evaluación por el discriminador y DAMSM:** El discriminador evalúa la autenticidad de la imagen generada, mientras que DAMSM mide la similitud entre la descripción textual y la imagen producida. La combinación de estos dos enfoques permite ajustar el generador para mejorar tanto la autenticidad visual como la coherencia semántica.
4. **Retroalimentación iterativa:** El proceso de generación y evaluación se repite en varias iteraciones. Cada ciclo ajusta los parámetros del generador para refinar la calidad de las imágenes, incorporando una correspondencia más precisa con las descripciones textuales.

4.7.6. Aplicaciones

Al igual que las cGAN, AttnGAN ha encontrado aplicaciones en diversos campos que requieren una correspondencia precisa entre texto e imagen. Algunas de las aplicaciones más notables incluyen:

- **Creación de Imágenes a partir de Texto en Detalle:** AttnGAN es particularmente adecuado para generar imágenes con detalles complejos. Esto permite crear imágenes de calidad a partir de descripciones textuales detalladas, como se utiliza en diseño de moda, generación de paisajes o escenas específicas [13].
- **Edición de Imágenes Guiada por Texto:** AttnGAN permite la edición de imágenes existente utilizando descripciones textuales. Al cambiar palabras clave en el texto, el modelo puede modificar detalles específicos en la imagen sin alterar su estructura general.

- **Creación de Imágenes Médicas Sintéticas:** En el ámbito de la medicina, AttnGAN puede ser empleado para generar imágenes médicas sintéticas, especialmente en tareas que requieren la adición de características específicas en imágenes como tumores o lesiones en escaneos médicos, permitiendo una generación controlada en base a descripciones anatómicas [2].
- **Aplicaciones en Publicidad y Diseño Creativo:** AttnGAN es utilizado en la generación de contenido visual para campañas publicitarias, donde la descripción textual es crucial para representar productos con características específicas o crear ambientes temáticos detallados.

Con su capacidad de captar detalles finos y específicos a partir de descripciones textuales, AttnGAN representa una herramienta poderosa para aplicaciones que requieren precisión visual y semántica en la generación de imágenes.

4.8. Procesamiento del lenguaje natural

El procesamiento del lenguaje natural (NLP) es una disciplina de la inteligencia artificial que se encarga de dotar a los sistemas informáticos de la capacidad de comprender, interpretar, y generar lenguaje humano de manera eficiente. Utilizando una combinación de lingüística computacional y modelos de aprendizaje automático, el NLP permite a las máquinas procesar tanto texto como voz, con el fin de captar la intención y el sentimiento detrás de las comunicaciones humanas.



Figura 8: Procesamiento del lenguaje

4.8.1. Preparación del texto

En esta etapa, se lleva a cabo la preparación del texto para su análisis posterior. Esto implica una serie de pasos de limpieza de datos, como la eliminación de caracteres especiales, la conversión a minúsculas y la exclusión de palabras vacías (stop words). El objetivo es asegurar que el texto esté en un formato adecuado para su procesamiento posterior.

4.8.2. Tokenización

Durante la tokenización, el texto se divide en unidades más pequeñas y significativas, conocidas como “tokens”. Estos tokens pueden ser palabras individuales o frases completas, dependiendo del contexto y los objetivos del análisis. La tokenización proporciona una base fundamental para el procesamiento y análisis subsiguientes del texto, ya que permite a los algoritmos obtener una comprensión básica de la estructura y el contexto del texto.

En el caso de la tokenización de palabras, el texto se separa en tokens individuales basados en espacios en blanco, lo que permite identificar y aislar cada palabra dentro del texto. Por otro lado, en la tokenización de frases, el texto se divide en tokens basados en puntos o signos de puntuación, lo que facilita la identificación de unidades más grandes de texto con significado coherente.

4.8.3. Representación del texto

Adicionalmente, cada token generado durante el proceso de tokenización se convierte en una representación numérica comprensible por el modelo de procesamiento del lenguaje natural. Esto es esencial para permitir que el modelo comprenda y procese el texto de manera efectiva. Esta conversión se lleva a cabo mediante diversas técnicas de vectorización de texto, entre las cuales destacan la codificación one-hot y la incrustación de palabras (word embeddings).

- La **codificación one-hot** consiste en asignar un valor binario único a cada token, creando un vector donde cada posición corresponde a una palabra única en el vocabulario. Por ejemplo, si tenemos un vocabulario de 1000 palabras, cada token se representaría como un vector de 1000 elementos, con un 1 en la posición que corresponde a la palabra en cuestión y ceros en todas las demás posiciones.
- Por otro lado, las **incrustaciones de palabras** (word embeddings) son representaciones vectoriales de palabras que capturan el significado semántico y la relación contextual entre ellas. Estas incrustaciones se generan mediante técnicas de aprendizaje automático y aprendizaje profundo, donde las palabras se representan como vectores densos de valores reales en un espacio de características de alta dimensionalidad.

Independientemente de la técnica utilizada, esta representación numérica de los tokens es esencial para alimentar el modelo de generación de imágenes en la siguiente etapa del proceso. Al transformar el texto en vectores numéricos, el modelo puede procesar la información de manera eficiente y generar contenido visual coherente y significativo basado en el contexto del texto de entrada.

4.9. Generación de imágenes a partir de texto

En la actualidad, se está presenciando un notable auge en el desarrollo de la Inteligencia Artificial Generativa, una vertiente diseñada para generar datos sintéticos innovadores basados en patrones extraídos de conjuntos de datos.

Este paradigma no se limita exclusivamente al procesamiento de datos en formato texto, sino que, además, abarca la capacidad de manipular imágenes, videos y audio; dando lugar a una diversidad de aplicaciones prometedoras que están emergiendo en distintos ámbitos.

4.9.1. Ejemplos

Los avances en inteligencia artificial están revolucionando la forma en que se diseñan productos industriales. Por ejemplo, se ha desarrollado un modelo llamado IA Midjourney que sirve como una herramienta creativa para representar ideas de manera rápida y efectiva; este modelo es capaz de transformar instrucciones detalladas en texto en representaciones visuales realistas de una amplia gama de productos industriales, desde cascos de motocicleta hasta zapatillas personalizadas de marcas reconocidas como Nike. Además, se ha observado que combinar múltiples tareas en una sola sesión puede mejorar la capacidad del modelo para generar imágenes variadas. Esta práctica también permite analizar los resultados en función de la popularidad del producto y explorar la variabilidad en las soluciones propuestas al representar objetos existentes. [5]



Figura 9: IA Midjourney

4.9.2. Consideraciones Éticas y de Privacidad

El desarrollo de la Inteligencia Artificial Generativa (IAG) representa un cambio disruptivo en la forma de concebir, diseñar y producir contenidos digitales. Estas tecnologías permiten generar imágenes, videos,

audios o incluso obras interactivas a partir de simples descripciones textuales, eliminando barreras técnicas y democratizando el acceso a herramientas de creación visual. El impacto de esta transformación ya se vislumbra en sectores como el diseño gráfico, la publicidad, la producción audiovisual, la educación y la realidad aumentada, entre otros.

No obstante, el avance acelerado de la IAG plantea una serie de dilemas éticos, sociales y legales de gran envergadura, que deben ser abordados de forma transversal y proactiva. A continuación, se detallan algunas de las principales consideraciones que afectan al uso responsable de estas tecnologías:

- **Derechos de autor y propiedad intelectual:** Las obras generadas por algoritmos no cuentan con un creador humano directo, lo que plantea un vacío legal sobre su titularidad. ¿Pertenece el contenido al desarrollador del modelo, al usuario que lo genera, o a nadie? Esta indefinición complica tanto la comercialización como la protección de las creaciones, y puede incentivar prácticas poco éticas como la apropiación indebida o el plagio algorítmico.
- **Naturaleza jurídica de los contenidos generados:** Es necesario revisar las leyes de propiedad intelectual para incluir categorías específicas para las obras creadas mediante IAG. Además, surge la necesidad de establecer criterios para distinguir entre creaciones humanas, asistidas por IA y totalmente sintéticas.
- **Transparencia, trazabilidad y responsabilidad algorítmica:** En el contexto de la creación de contenido automatizado, es fundamental garantizar la trazabilidad del proceso generativo y la explicabilidad de los modelos. Esto es especialmente importante en aplicaciones sensibles (como medios de comunicación o salud), donde la veracidad, la intención y la autoría son factores críticos.
- **Identificación de contenidos sintéticos (deepfakes):** La creciente sofisticación de las imágenes y videos generados con IA plantea riesgos de desinformación, suplantación de identidad, manipulación política o fraudes digitales. La incorporación de marcas de agua digitales, metadatos incrustados o sistemas de verificación de autenticidad podría ser clave para distinguir entre contenido real y sintético.
- **Privacidad y uso de datos sensibles:** Los modelos de IA suelen entrenarse con grandes volúmenes de datos extraídos de internet o bases públicas. En muchos casos, estos datos incluyen rostros, ubicaciones, nombres o información privada que no ha sido recolectada con consentimiento explícito. Esto plantea serias implicaciones en términos de derechos fundamentales y cumplimiento normativo (como el RGPD en Europa).
- **Sesgos algorítmicos y reproducción de estereotipos:** Las IAG pueden reproducir o amplificar sesgos presentes en los datos con los que fueron entrenadas. Esto puede derivar en la generación de contenido discriminatorio o excluyente, reforzando estereotipos de género, raza o clase social. La revisión crítica del dataset y la introducción de mecanismos de mitigación de sesgos resultan imprescindibles.
- **Uso malintencionado o delictivo:** Como toda tecnología poderosa, la IAG puede ser empleada con fines maliciosos: generación de pornografía no consensuada, campañas de desinformación, fraude visual o creación de identidades falsas. Es necesario implementar barreras técnicas (filtros, restricciones de contenido, auditorías) y políticas públicas que regulen su uso y difusión.
- **Impacto en el empleo y la autoría creativa:** El auge de herramientas que automatizan procesos creativos podría afectar a profesionales del diseño, la ilustración o la escritura, especialmente en tareas de baja especialización. Es importante fomentar un marco de convivencia entre creatividad humana e inteligencia artificial, promoviendo la transparencia y el reconocimiento del trabajo original.

En resumen, el progreso de la Inteligencia Artificial Generativa ofrece un potencial inmenso, pero también una serie de riesgos éticos y legales que requieren atención urgente por parte de gobiernos, entidades tecnológicas, comunidades científicas y la ciudadanía en general. La implementación de principios como la transparencia, la equidad, la privacidad y la rendición de cuentas debe guiar el diseño y despliegue de estas tecnologías, para asegurar un desarrollo justo, inclusivo y sostenible.

4.10. Modelos de Difusión y Stable Diffusion

En los últimos años, los modelos generativos basados en procesos de difusión han cobrado una importancia creciente como alternativa a las arquitecturas GAN tradicionales. A diferencia de estas últimas, los modelos de difusión no requieren un discriminador y tienden a ser más estables durante el entrenamiento, además de ofrecer un mayor control semántico durante la generación.

4.10.1. Fundamentos de los modelos de difusión

Los modelos de difusión simulan un proceso estocástico en el que una imagen se convierte progresivamente en ruido gaussiano a través de una secuencia de pasos temporales. A continuación, el modelo se entrena para invertir este proceso, es decir, para reconstruir una imagen coherente a partir del ruido puro en varios pasos. Este enfoque fue formalizado inicialmente en los modelos DDPM (Denoising Diffusion Probabilistic Models) [16], y posteriormente optimizado en variantes como DDIM.

La formulación teórica se basa en procesos de difusión continua, descritos por la ecuación de Fokker–Planck:

$$\frac{\partial p(x, t)}{\partial t} = -\nabla \cdot (f(x)p(x, t)) + \nabla^2(g(x)p(x, t)) \quad (5)$$

donde el objetivo es aprender una función de denoising $f_\theta(x, t)$ capaz de revertir el ruido inyectado en cada paso del proceso. La pérdida comúnmente empleada en estos modelos mide la diferencia entre el ruido predicho y el ruido real aplicado.

4.10.2. Transición hacia la difusión latente

Uno de los principales retos de los modelos de difusión tradicionales es su alto coste computacional, ya que operan directamente sobre imágenes de alta resolución. Para abordar esta limitación, surgieron los modelos de difusión latente, que trasladan el proceso de generación a un espacio comprimido de menor dimensión, manteniendo la información semántica esencial.

4.10.3. Stable Diffusion

Entre los modelos de difusión latente, destaca **Stable Diffusion** [17], desarrollado por Stability AI y basado en la arquitectura Latent Diffusion Model (LDM). Este modelo ha ganado gran popularidad por su capacidad de generar imágenes fotorrealistas de alta calidad condicionadas por descripciones textuales (prompts), manteniendo una eficiencia computacional razonable.

Una de las características más destacables de Stable Diffusion es su disponibilidad como software de código abierto. Esto ha facilitado su adopción masiva en comunidades académicas, creativas y profesionales, y ha contribuido de forma significativa a la democratización del acceso a modelos generativos de alta calidad.

Stable Diffusion está compuesto por tres bloques principales:

- Un **autoencoder variacional** (VAE), que se encarga de transformar imágenes del espacio RGB (alta dimensión) a un espacio latente comprimido y semánticamente significativo. Esta representación latente conserva la información visual esencial y permite operar con mayor eficiencia computacional. Una vez generado el resultado en el espacio latente, el VAE también es responsable de decodificarlo de nuevo a imagen visible, asegurando que se mantenga la coherencia visual respecto al contenido original.
- Un **modelo de difusión U-Net**, que actúa directamente sobre la representación latente de las imágenes. Esta red convolucional profunda está entrenada para eliminar progresivamente el ruido introducido en el espacio latente, reconstruyendo paso a paso una imagen coherente. Su arquitectura simétrica con conexiones de salto permite capturar tanto información global como detalles locales, y puede ser condicionada por información externa, como texto, durante el proceso de denoising.
- Un **codificador textual**, basado en modelos como CLIP o T5, cuya función es convertir la descripción textual (*prompt*) en un vector de características semánticas. Este vector se utiliza como guía condicional en el proceso de generación. Gracias a los mecanismos de atención cruzada implementados en la U-Net,

las representaciones textuales influyen directamente en distintas regiones de la imagen, permitiendo un alto grado de control sobre los elementos visuales generados en función del texto.

El modelo introduce un mecanismo de atención cruzada (cross-attention) que asocia partes específicas del texto con regiones concretas de la imagen, permitiendo un control detallado sobre el contenido generado.

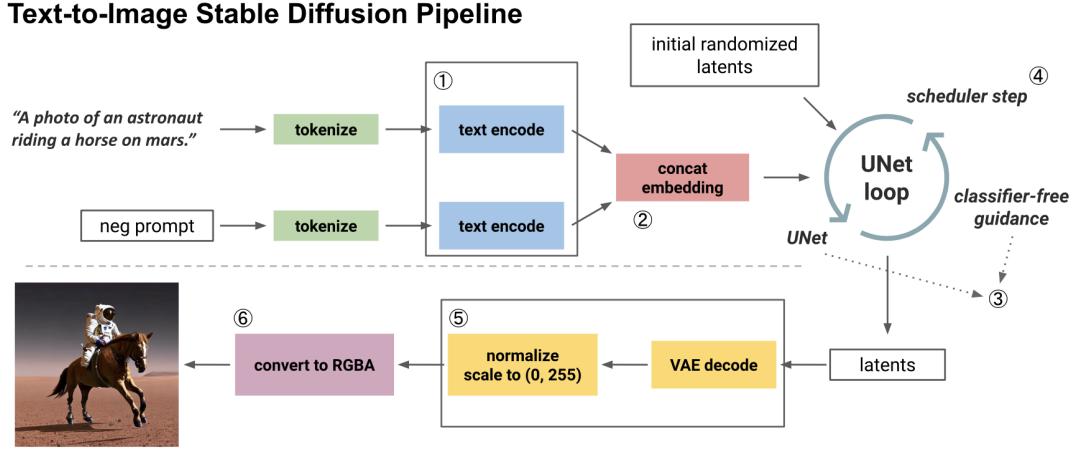


Figura 10: Pipeline general del modelo Stable Diffusion. Fuente: [15]

La Figura 10 muestra el flujo de generación de imágenes en Stable Diffusion. A partir de un texto de entrada (*prompt*), se obtiene un embedding que guía el proceso de denoising del modelo UNet en el espacio latente. Una vez finalizado el proceso, el resultado se decodifica a imagen mediante un VAE y se convierte a formato visual estándar.

4.10.4. Ventajas frente a otras arquitecturas

Stable Diffusion ofrece múltiples ventajas en comparación con arquitecturas anteriores como GANs o modelos híbridos como AttnGAN:

- **Mayor coherencia semántica:** gracias al uso de embeddings textuales generados por modelos como CLIP o T5, y su integración mediante mecanismos de atención cruzada en la red U-Net, el modelo es capaz de asociar con precisión conceptos textuales con regiones específicas de la imagen generada. Esto resulta en una representación visual más fiel al contenido del prompt.
- **Mejor estabilidad:** a diferencia de las GAN, que dependen del equilibrio entre generador y discriminador, los modelos de difusión no requieren esta dinámica adversarial. Esto reduce significativamente problemas como el colapso de modo (donde el generador produce salidas repetitivas) o la inestabilidad del entrenamiento.
- **Coste computacional reducido:** al operar en un espacio latente comprimido mediante un autoencoder variacional, el modelo evita los altos requerimientos de memoria y procesamiento asociados con la manipulación directa de imágenes de alta resolución. Esta eficiencia permite su uso incluso en entornos con recursos limitados.
- **Modularidad:** la arquitectura de Stable Diffusion permite modificar, sustituir o mejorar individualmente componentes como el codificador de texto, el modelo de difusión o el decodificador VAE. Esto facilita la incorporación de mejoras incrementales sin tener que reentrenar todo el sistema desde cero.

Además, el trabajo de Rombach et al. [17] proporciona evidencia empírica de que el uso del espacio latente preserva la estructura semántica de las imágenes, manteniendo una alta calidad visual con menor coste computacional.

4.10.5. Aplicaciones prácticas y uso en este trabajo

El modelo Stable Diffusion ha demostrado ser eficaz en una amplia gama de aplicaciones prácticas:

- **Generación de arte conceptual y prototipos visuales:** especialmente útil para diseñadores, artistas digitales y estudios creativos que buscan materializar ideas a partir de descripciones verbales.
- **Síntesis de datos para sistemas de visión artificial:** puede generar imágenes etiquetadas sintéticamente para entrenar clasificadores, detectores o segmentadores en entornos donde los datos reales son escasos o costosos de obtener.
- **Producción de contenido multimedia:** desde elementos gráficos en videojuegos hasta campañas publicitarias personalizadas, permite generar activos visuales adaptados a necesidades específicas.

En este proyecto, se ha utilizado Stable Diffusion como motor generativo dentro del sistema JMR. Gracias a una API REST, el usuario puede introducir un prompt textual en lenguaje natural y recibir una imagen generada que sirve como entrada visual al sistema CBIR (Content-Based Image Retrieval). Esta funcionalidad extiende las capacidades de búsqueda y análisis de la plataforma, haciendo el sistema más accesible y flexible.

4.10.6. Limitaciones y desafíos actuales

A pesar de sus beneficios, es importante considerar algunas limitaciones inherentes a Stable Diffusion:

- **Velocidad de generación:** aunque más rápido que modelos de difusión que operan en pixel space, sigue siendo más lento que las GAN una vez entrenadas, ya que requiere múltiples pasos de inferencia para refinar la imagen.
- **Requisitos de memoria:** los modelos de difusión siguen siendo exigentes en términos de memoria GPU, especialmente al trabajar con prompts largos o al generar imágenes de alta resolución.
- **Falta de control preciso:** en casos donde el prompt es ambiguo, contradictorio o demasiado abierto, el modelo puede generar imágenes que no reflejan adecuadamente la intención del usuario.
- **Cuestiones éticas:** al estar entrenado con grandes volúmenes de datos de internet, el modelo puede reproducir sesgos sociales o generar contenido problemático. Además, existe un debate abierto sobre los derechos de autor en imágenes generadas a partir de entrenamiento con obras protegidas.

Estas limitaciones no invalidan su utilidad, pero deben ser cuidadosamente consideradas en contextos profesionales, educativos o institucionales donde la transparencia, la precisión y la responsabilidad son prioritarias.

En consecuencia, cualquier despliegue real del sistema basado en Stable Diffusion debe ir acompañado de estrategias de mitigación ética, validación técnica rigurosa y un marco de uso responsable que garantice su fiabilidad y sostenibilidad a largo plazo.

5. Entrenamiento y experimentación con modelos generativos

El proceso de desarrollo del sistema generativo se ha abordado desde una perspectiva experimental, orientada a identificar, comparar y refinar diferentes arquitecturas capaces de transformar texto en imagen. Este capítulo recoge las principales fases de exploración realizadas, desde aproximaciones iniciales con redes generativas adversarias simples hasta la implementación final con modelos de difusión.

Durante esta etapa se han probado múltiples configuraciones, técnicas de entrenamiento y datasets, en un proceso iterativo que ha permitido evaluar los puntos fuertes y débiles de cada solución. Las pruebas realizadas han sido fundamentales no solo para mejorar la calidad de las imágenes generadas, sino también para asegurar que el sistema pudiera adaptarse a las restricciones técnicas del entorno y responder adecuadamente a las necesidades planteadas en los requisitos del proyecto.

En las siguientes secciones se describen en detalle las distintas aproximaciones evaluadas, los criterios utilizados para su comparación y las decisiones técnicas que condujeron al diseño final del sistema.

5.1. Primera aproximación: GAN

Como punto de partida en el desarrollo del sistema generativo, se optó por implementar una Red Generativa Adversaria (GAN) clásica. Esta elección respondió a su simplicidad conceptual y a la extensa documentación existente que la convierte en una excelente base para experimentar con modelos generativos. La implementación inicial permitió adquirir una comprensión profunda del funcionamiento adversarial entre el generador y el discriminador, y sentar las bases para una posterior transición hacia arquitecturas más avanzadas.

Las GANs tradicionales operan sin condicionamiento explícito, es decir, generan imágenes sin ningún tipo de control externo sobre el contenido. Si bien esto limita la aplicabilidad directa para sistemas de búsqueda visual guiados por texto, su valor como primera aproximación radicó en que permitieron validar el flujo de entrenamiento, optimizar el entorno de ejecución y establecer un punto de referencia inicial de calidad de las imágenes generadas.

5.1.1. Librerías, herramientas y datasets

Para la implementación se utilizaron **Keras** y **TensorFlow**, dos frameworks ampliamente adoptados en la comunidad de aprendizaje profundo. Keras, con su API de alto nivel, facilitó la definición de modelos de forma modular y legible, mientras que TensorFlow proporcionó soporte para operaciones eficientes en GPU y una gestión estable del grafo computacional. Esta combinación permitió iterar rápidamente en la definición y entrenamiento de las redes.

En cuanto a los datos, se seleccionaron dos conjuntos integrados en la propia librería de Keras: **CIFAR-10** y **CIFAR-100**. Ambos constan de 60.000 imágenes a color de resolución 32x32 píxeles, distribuidas equitativamente entre clases. CIFAR-10 contiene 10 categorías genéricas (como coches, animales, aviones), lo que lo convierte en un dataset óptimo para pruebas preliminares. Por su parte, CIFAR-100 agrupa las imágenes en 100 clases más específicas, lo que representa un desafío considerable mayor para la generalización del modelo generador.

5.1.2. Pruebas y resultados

La arquitectura definida para esta fase inicial consistió en una red generativa compuesta por varias capas convolucionales transpuestas, activadas con **ReLU**, que transformaban un vector latente aleatorio de 100 dimensiones en una imagen RGB de 32x32 píxeles. El discriminador, en contrapartida, estaba constituido por una red con capas convolucionales convencionales y activations **LeakyReLU**, cuya salida final indicaba la probabilidad de que una imagen fuese real o generada.

Durante el entrenamiento con CIFAR-10 se llevaron a cabo múltiples iteraciones para estudiar el comportamiento del modelo:

En una primera fase experimental, se utilizó una tasa de aprendizaje de 0.0002 junto con el optimizador **Adam**, obteniéndose imágenes que reflejaban estructuras rudimentarias, pero carecían de nitidez y riqueza de detalle. Uno de los primeros hallazgos fue que el discriminador aprendía más rápidamente que el generador, generando un desequilibrio que provocaba el estancamiento del entrenamiento.

Para mitigar este efecto, se redujo la tasa de aprendizaje a 0.0001, lo que produjo una convergencia más suave. Sin embargo, las imágenes seguían mostrando ruido y distorsiones. En una tercera iteración, se decidió aumentar la capacidad del generador, introduciendo capas adicionales e incrementando la dimensionalidad de las capas intermedias. Este ajuste permitió generar imágenes ligeramente más definidas, aunque seguían siendo poco coherentes a nivel semántico.

Con el dataset CIFAR-100, el proceso se replicó bajo condiciones similares. No obstante, el salto en complejidad del conjunto de datos se tradujo en mayores dificultades para el modelo. En las etapas iniciales, las imágenes generadas eran considerablemente más borrosas y la red mostraba dificultades para aprender patrones distinguibles. Para contrarrestar el sobreajuste del discriminador, se introdujeron capas **Dropout** y se aplicaron nuevas reducciones a la tasa de aprendizaje. Se exploraron, además, modificaciones estructurales en el generador, como la inclusión de **BatchNormalization** y ajustes en las funciones de activación, pero sin éxito significativo en la calidad final.

Los resultados más representativos de estos experimentos se sintetizan en la Tabla 3, donde se detallan los principales hiperparámetros, ajustes arquitectónicos realizados y observaciones obtenidas para cada fase de entrenamiento con los datasets CIFAR-10 y CIFAR-100.

Experimento	Hiperparámetros	Modificaciones	Resultados
CIFAR-10: Fase 1	LR=0.0002, Adam, 50 épocas	Arquitectura estándar de DCGAN	Imágenes básicas con baja resolución y poco detalle.
CIFAR-10: Fase 2	LR=0.0001, Adam	Sin cambios en la arquitectura	Mejor estabilidad, pero mucho ruido.
CIFAR-10: Fase 3	LR=0.0001, más capas, mayor dimensionalidad	Generador más profundo, mayor capacidad	Ligera mejora en detalle, pero sin correspondencia clara con clases.
CIFAR-100: Fase 1	LR=0.0002, Adam	Arquitectura igual que CIFAR-10	Alta dificultad de convergencia. Imágenes más borrosas.
CIFAR-100: Fase 2	LR=0.0001, Dropout en Discriminador	Regularización para evitar sobreajuste	Mejor equilibrio de pérdidas, pero sin mejora clara en calidad.
CIFAR-100: Fase 3	LR=0.0001, arquitectura ampliada	Capas adicionales, mayor profundidad en generador	Imágenes aún con ruido y baja fidelidad semántica.

Tabla 3: Resumen de experimentos con GAN: configuración y resultados

5.1.3. Conclusión

La primera aproximación mediante una GAN básica resultó útil desde una perspectiva formativa y técnica: permitió familiarizarse con los ciclos de entrenamiento, con la interacción generador-discriminador y con el entorno de ejecución completo. Además, ofreció una referencia sobre la calidad base que se puede obtener sin condicionamiento.

Sin embargo, también dejó patente que este enfoque no era suficiente para el objetivo del proyecto: generar imágenes condicionadas por descripciones textuales. La incapacidad de controlar el contenido generado y la baja calidad semántica de las salidas evidenciaron la necesidad de incorporar mecanismos de control explícito. Así, esta experiencia motivó el paso a la siguiente etapa: el uso de redes generativas adversarias condicionales, que permitirían introducir un vector de características (por ejemplo, proveniente de una descripción textual) como condición del proceso generativo.

5.2. Condisionalidad: cGAN

A diferencia de las tradicionales, las Redes Generativas Adversarias Condicionales (cGAN) permiten generar imágenes a partir de una condición explícita, como un vector de características derivado de una descripción textual. Esta capacidad resultaba fundamental para los objetivos del proyecto, por lo que se planteó como un enfoque prometedor desde el inicio.

5.2.1. Librerías y Herramientas

La implementación de la cGAN se llevó a cabo empleando herramientas ya utilizadas en la fase anterior con GAN básicas, como Keras y TensorFlow, por su robustez y facilidad de desarrollo. Además, se utilizaron NumPy y Matplotlib para el preprocesamiento de datos y la visualización de resultados, respectivamente. El dataset elegido fue COCO (Common Objects in Context), que destaca por incluir una gran variedad de imágenes, cada una acompañada por cinco descripciones textuales detalladas. Esto lo hace ideal para experimentar con modelos de generación condicionada por texto.

5.2.2. Procesamiento de Texto

Para representar las descripciones textuales de entrada, se probaron distintos enfoques. Inicialmente se utilizó *one-hot encoding*, por su simplicidad de implementación, aunque se descartó rápidamente debido a su alta dimensionalidad y su incapacidad para capturar relaciones semánticas entre palabras.

Posteriormente, se probó Word2Vec, que transforma palabras en vectores de características en un espacio semántico. Esta técnica mejoró la representación textual, pero mostró limitaciones al no capturar adecuadamente la estructura de las frases ni las relaciones entre los objetos.

Finalmente, se optó por trabajar directamente con las descripciones originales del dataset COCO, que ya estaban preprocesadas y estructuradas. Estas descripciones fueron tokenizadas y limpiadas para eliminar ruido textual. Se utilizaron embeddings preentrenados como los de CLIP, que transforman las frases en vectores latentes significativos para la generación de imágenes. Esta solución resultó más robusta, computacionalmente eficiente y adecuada para preservar la semántica de las descripciones complejas.

5.2.3. Pruebas preliminares con MNIST

Antes de abordar un dataset complejo como COCO, se realizó una prueba de concepto utilizando el conjunto de datos MNIST. Este dataset contiene imágenes de dígitos del 0 al 9, por lo que era ideal para comprobar la capacidad de la cGAN para generar imágenes condicionadas.

Se codificaron los dígitos como vectores one-hot y se concatenaron con el ruido aleatorio que se proporciona al generador. El modelo se entrenó durante 50 épocas con una tasa de aprendizaje de 0.0002 y un tamaño de batch de 64. Al principio, las imágenes eran borrosas, pero al finalizar el entrenamiento, la cGAN era capaz de generar dígitos nítidos y variados, respetando la condición de entrada. Este comportamiento puede observarse en la Figura 11, que muestra los dígitos generados al final del proceso. Esto confirmó la viabilidad del enfoque condicional.

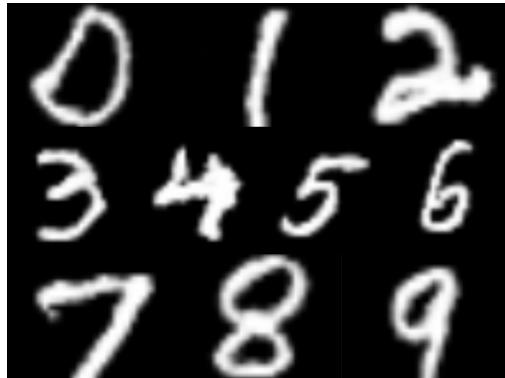


Figura 11: Imágenes generadas con el modelo entrenado con MNIST

5.2.4. Datasets

El modelo fue entrenado con el dataset COCO en dos variantes. Por un lado, se utilizó el conjunto completo, que ofrece una gran variedad de escenarios y contextos, lo que mejora la generalización del modelo. Sin embargo, su alto requerimiento computacional exigía un entorno potente y tiempos prolongados de entrenamiento.

Por otro lado, se trabajó con subconjuntos aleatorios del dataset, lo que permitía experimentar con mayor agilidad y requerimientos más modestos de memoria. Si bien esto limitaba la generalización, resultó útil para afinar hiperparámetros y realizar pruebas rápidas.

5.2.5. Pruebas y resultados

La arquitectura de la cGAN fue adaptada para trabajar con entrada condicionada, lo que implicó modificaciones específicas tanto en el generador como en el discriminador. El generador fue diseñado para recibir un vector de ruido aleatorio concatenado con un vector de características textuales, representando la descripción de la imagen deseada. Este vector conjunto alimentaba una red de capas transpuestas convolucionales que generaban una imagen de salida. Por su parte, el discriminador fue ajustado para aceptar como entrada tanto la imagen generada como la descripción textual correspondiente, evaluando si la imagen no solo era realista, sino también coherente con el texto proporcionado.

Durante el proceso de entrenamiento, se llevaron a cabo múltiples fases de pruebas, con ajustes progresivos que permitieron estudiar el comportamiento del modelo ante diferentes configuraciones:

- **Primera etapa:** Se utilizó una tasa de aprendizaje de 0.0002 y se mantuvo una arquitectura básica en el generador y el discriminador. En esta configuración, el modelo logró generar imágenes con formas básicas y contornos generales, pero sin mucho detalle. Las imágenes eran a menudo borrosas y poco realistas, especialmente cuando las descripciones textuales eran más complejas o contenían múltiples objetos.
- **Segunda etapa:** Se redujo la tasa de aprendizaje a 0.0001, buscando una mayor estabilidad en el proceso de entrenamiento. Este ajuste permitió al modelo generar imágenes con mayor nitidez en los bordes y una ligera mejora en la estructura general de los objetos. No obstante, aún persistía cierta borrosidad, especialmente cuando el texto incluía relaciones espaciales o adjetivos calificativos que requerían una interpretación semántica más profunda.
- **Tercera etapa:** Se realizaron modificaciones más profundas en la arquitectura, incrementando la profundidad de las redes LSTM utilizadas para procesar el texto y añadiendo capas convolucionales adicionales al generador. Estos cambios permitieron al modelo capturar mejor los detalles visuales descritos en las anotaciones textuales. Las imágenes generadas mostraban una mayor fidelidad a las descripciones, especialmente en contextos sencillos o escenas con un número limitado de elementos. Sin embargo, cuando las descripciones eran más complejas, que incluían múltiples objetos en relación o elementos abstractos, el modelo aún presentaba dificultades para generar imágenes coherentes.

En conjunto, estas pruebas demostraron que la arquitectura de la cGAN, con un ajuste adecuado de hiperparámetros y arquitectura, era capaz de generar imágenes razonablemente alineadas con descripciones textuales simples, aunque todavía mostraba limitaciones en cuanto a la representación semántica profunda y la calidad visual en escenarios complejos.

5.2.6. Pruebas con diferentes números de épocas

Se compararon dos enfoques de entrenamiento para evaluar cuál ofrecía mejores resultados bajo distintas condiciones de entrenamiento y disponibilidad de recursos computacionales:

- **Entrenamiento en múltiples fases cortas (menos épocas):** Este enfoque consistía en entrenar el modelo con un número reducido de épocas, interrumpir el proceso, analizar los resultados obtenidos y realizar ajustes en los hiperparámetros o en la arquitectura antes de retomar el entrenamiento. Su principal ventaja radica en la velocidad de iteración, permitiendo una retroalimentación rápida sobre el comportamiento del modelo. Esto resultó útil especialmente en las primeras fases de desarrollo,

donde se requería probar distintas combinaciones de parámetros y realizar modificaciones frecuentes. Sin embargo, se observó que este tipo de entrenamiento limitaba el potencial de aprendizaje del modelo, ya que no se alcanzaban suficientes ciclos de optimización para capturar patrones más complejos del conjunto de datos. Además, las mejoras en la calidad de las imágenes generadas tendían a estancarse tras unas pocas épocas.

- **Entrenamiento continuo con muchas épocas:** En contraste, este enfoque implicaba ejecutar el entrenamiento de manera prolongada, permitiendo que el modelo tuviese tiempo suficiente para ajustar de forma más precisa los pesos de sus redes. Esta estrategia fue especialmente efectiva una vez se contaba con una configuración inicial bien afinada, ya que permitía que el modelo convergiera hacia una solución más estable y generara imágenes de mayor calidad. Las imágenes resultantes presentaban mayor nivel de detalle y coherencia con las descripciones textuales. No obstante, este método también conllevaba riesgos, como el sobreajuste, especialmente si los hiperparámetros no estaban correctamente definidos. Además, al requerir mayor tiempo y recursos computacionales, su ejecución resultaba menos viable en fases iniciales de desarrollo o cuando se contaba con recursos limitados.

Ambos métodos ofrecieron ventajas complementarias. El entrenamiento en fases cortas fue ideal para la exploración rápida y la experimentación con diferentes configuraciones, mientras que el entrenamiento extendido resultó más adecuado en etapas finales, cuando se buscaba optimizar la calidad del modelo ya ajustado. En el contexto del proyecto, se utilizó una combinación de ambos enfoques: primero se realizaron múltiples entrenamientos breves para afinar la arquitectura y los hiperparámetros, y posteriormente se aplicó un entrenamiento largo con la configuración seleccionada para maximizar el rendimiento final del modelo. Las diferencias entre ambos enfoques, así como sus principales ventajas y desventajas, se resumen en la Tabla 4.

Aspecto	Entrenar varias veces con menos épocas	Entrenar una sola vez con más épocas
Velocidad de iteración	Alta: permite detectar problemas y ajustar configuraciones rápidamente	Baja: requiere más tiempo para identificar problemas o ajustar parámetros
Potencial de aprendizaje	Limitado, ya que el modelo no alcanza su máximo potencial	Alto, permite que el modelo ajuste mejor los pesos y logre mayor detalle
Riesgo de sobreajuste	Bajo, debido a la menor cantidad de épocas	Alto, especialmente si la configuración inicial no es óptima
Calidad de las imágenes	Adecuada para configuraciones iniciales, pero se estanca en calidad final	Mejor calidad, con imágenes más detalladas y coherentes
Flexibilidad en ajustes	Alta: permite ajustar configuraciones con mayor frecuencia	Baja: los errores solo se detectan al final del proceso
Uso de recursos	Menor uso de recursos en cada iteración	Mayor uso de recursos debido a entrenamientos más largos
Idoneidad	Ideal para experimentación y ajustes rápidos	Adecuado para optimización final con configuraciones ajustadas

Tabla 4: Comparativa entre entrenar con menos épocas y entrenar con más épocas

5.2.7. Visualización de pérdidas

Las curvas de pérdida registradas durante el entrenamiento (Figura 12) muestran que el discriminador aprende rápidamente en las primeras épocas, con una pérdida que disminuye de forma pronunciada. En contraste, el generador progresó más lentamente, con una pérdida que decrece gradualmente. Este comportamiento es común en las cGAN, y si no se controla, puede llevar a un desequilibrio en el entrenamiento. En este caso, el discriminador tendía a dominar, lo que podría afectar negativamente la calidad de las imágenes generadas.

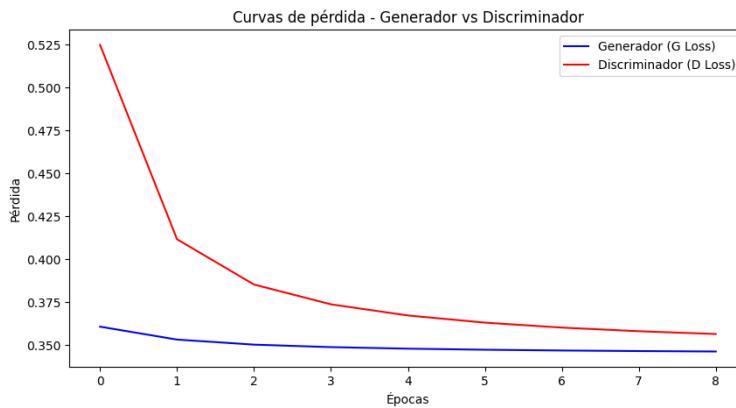


Figura 12: Curva de pérdidas durante el entrenamiento de la cGAN

5.2.8. Conclusión

La cGAN permitió integrar las descripciones textuales en el proceso de generación de imágenes, obteniendo mejores resultados que con la GAN básica. Sin embargo, el modelo mostró dificultades para representar con nitidez descripciones complejas. Esto motivó la exploración de modelos más sofisticados como la AttnGAN, que incorpora mecanismos de atención para mejorar la correspondencia entre texto e imagen.

5.3. Atención textual: AttnGAN

AttnGAN representa una evolución respecto a la cGAN, especialmente en cuanto a estabilidad durante el entrenamiento y eficiencia en la gestión de recursos computacionales. A diferencia de las cGANs implementadas en TensorFlow, AttnGAN se desarrolló sobre PyTorch, lo que permitió un mayor control sobre la asignación de memoria y una mejor capacidad para manejar grandes volúmenes de datos. Este modelo está específicamente diseñado para generar imágenes a partir de descripciones textuales mediante el uso de mecanismos de atención, que permiten asociar palabras o fragmentos específicos del texto con regiones concretas de la imagen. Este enfoque favorece una mayor coherencia semántica entre el contenido textual y visual, y mejora el nivel de detalle generado en las imágenes.

5.3.1. Configuración del entrenamiento

El modelo fue entrenado utilizando el dataset COCO completo, el cual proporciona una gran variedad de imágenes junto con múltiples descripciones por imagen, lo que resulta ideal para tareas de generación condicionada por texto. Gracias a la eficiencia de PyTorch en la gestión de memoria, no fue necesario trabajar con subconjuntos del dataset, como sí ocurrió en el caso de la cGAN. El entrenamiento se realizó en la plataforma Kaggle, que ofrece acceso gratuito a GPU pero impone una limitación de 30 horas semanales de ejecución. Esta restricción obligó a limitar el entrenamiento a un máximo de 40 épocas por ejecución completa.

A pesar de esta limitación temporal, la estabilidad del entrenamiento fue notable. PyTorch permitió mantener un tamaño de batch razonable sin generar errores de memoria, lo que favoreció una convergencia

progresiva y consistente. Asimismo, el modelo integra mecanismos de atención en múltiples niveles: en cada etapa del proceso generativo, el sistema aprende a enfocar su atención en las partes más relevantes de la descripción textual, optimizando así la correspondencia entre texto e imagen.

5.3.2. Proceso de entrenamiento

El entrenamiento de AttnGAN se dividió en varias fases, cada una diseñada para evaluar y refinar la calidad de las imágenes generadas. Aunque el límite de 40 épocas restringía el alcance del entrenamiento, fue posible observar una mejora progresiva en la correspondencia entre texto e imagen. En comparación con la cGAN, donde la pérdida era altamente inestable, AttnGAN mostró una evolución más suave y coherente, aunque no se generaron curvas de pérdida formales para su análisis.

En ausencia de métricas cuantitativas detalladas, se utilizó una estrategia de evaluación visual para examinar las imágenes generadas a lo largo del entrenamiento. Esta evaluación permitió ajustar los hiperparámetros, como la tasa de aprendizaje y la arquitectura de las capas intermedias, en función de la calidad perceptual observada. En futuras implementaciones, contar con un sistema de logging más completo permitiría extraer información cuantitativa útil para complementar esta evaluación.

5.3.3. Resultados y evaluación visual

A nivel de resultados, AttnGAN logró generar imágenes que reflejaban de forma razonable la estructura y los elementos clave descritos en el texto. En particular, se utilizó como entrada el prompt “*A man with a red jacket riding a motorcycle in the woods*”, que describe una escena compleja con múltiples objetos y relaciones espaciales. La imagen generada, mostrada en la Figura 13, transmite adecuadamente la intención general del texto: puede apreciarse una figura central clara, asociada al conductor y su motocicleta, rodeada de un entorno con colores verdes y marrones que sugiere vegetación.

No obstante, las imágenes seguían presentando borrosidad y falta de precisión en los detalles, lo que limitaba su aplicabilidad en contextos que requieran imágenes de alta fidelidad.

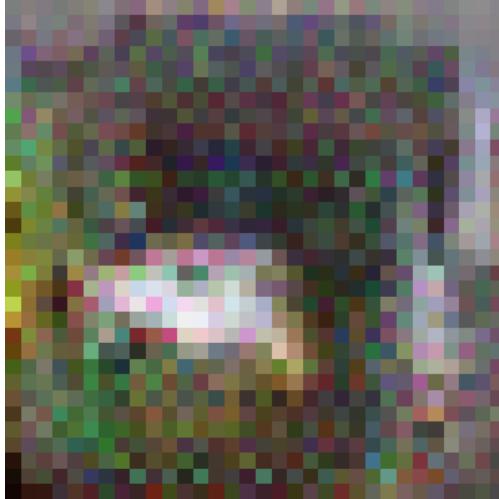


Figura 13: Mejor imagen generada por AttnGAN durante el entrenamiento con el prompt: “*A man with a red jacket riding a motorcycle in the woods*”

5.4. Modelo final

Tras una fase intensiva de desarrollo e implementación de modelos generativos propios como GAN, cGAN y AttnGAN, se identificaron varias limitaciones que obstaculizaban el objetivo principal del proyecto: obtener imágenes visualmente coherentes y de calidad a partir de descripciones textuales. Entre los principales desafíos se encontraron la baja fidelidad visual de las imágenes generadas, la inestabilidad durante el entrenamiento y la elevada demanda de recursos computacionales. Ante este escenario, se optó por adoptar un enfoque basado

en modelos preentrenados de alta calidad, siendo *Stable Diffusion v1.4* el seleccionado por su equilibrio entre rendimiento, accesibilidad y resultados visuales.

5.4.1. Motivación para el uso de modelos preentrenados

El uso de modelos desarrollados desde cero permitió adquirir una comprensión profunda sobre los mecanismos de generación de imágenes, los procesos de codificación semántica del texto y el funcionamiento interno de arquitecturas como UNet o LSTM. Sin embargo, los resultados obtenidos no alcanzaban el nivel de calidad deseado, especialmente al trabajar con descripciones complejas del dataset COCO. Además, las restricciones computacionales (limitación de GPU, RAM y tiempo de entrenamiento) impedían escalar las pruebas de manera eficaz. Por ello, se decidió migrar a un modelo preentrenado que ofreciera resultados competitivos desde el inicio sin necesidad de un proceso de entrenamiento completo, siendo Stable Diffusion una de las soluciones más robustas disponibles actualmente para la generación de imágenes a partir de texto.

5.4.2. Descripción del modelo: Stable Diffusion

Stable Diffusion es un modelo generativo de código abierto basado en el paradigma de *modelos de difusión latente*. A diferencia de los enfoques GAN, donde dos redes adversarias compiten entre sí, este modelo transforma una distribución de ruido gaussiano hacia una imagen coherente mediante un proceso de denoising progresivo. Todo el proceso se ejecuta dentro de un espacio latente comprimido, lo cual mejora considerablemente la eficiencia computacional sin sacrificar calidad visual.

Componentes funcionales del modelo

El modelo se compone de varios módulos interconectados que trabajan en conjunto para transformar texto en imágenes. A continuación se presenta un resumen de los componentes principales y sus funciones:

Componente	Función principal
UNet2DConditionModel	Red convolucional profunda que refina la imagen ruidosa en múltiples pasos, guiada por el texto. Se encarga del proceso de denoising en el espacio latente.
CLIPTextModel (ViT-L/14)	Codificador textual que convierte la descripción de entrada en una representación semántica densa utilizada como guía para el proceso de generación.
AutoencoderKL (VAE)	Encargado de mapear imágenes reales al espacio latente y decodificar las imágenes generadas desde dicho espacio a píxeles reales.
Scheduler (DDIM)	Controla el ruido en cada paso y define la trayectoria de desdenoising, determinando el número de pasos y el ritmo de evolución.

Tabla 5: Resumen funcional de los componentes principales de Stable Diffusion

Parámetros técnicos

Stable Diffusion v1.4 se caracteriza por una serie de parámetros técnicos que definen su arquitectura y funcionamiento. Estos parámetros son cruciales para entender el rendimiento del modelo y su capacidad de generar imágenes de alta calidad a partir de descripciones textuales. A continuación se detallan los más relevantes:

- **Resolución de entrada:** 512x512 píxeles.
- **Tamaño del espacio latente:** 64x64 píxeles.
- **Pasos de inferencia:** 50 (utilizando el scheduler DDIM).
- **Guidance scale (CFG):** 7.5, valor que controla el grado de alineamiento entre texto e imagen.

- **Codificador textual:** CLIP ViT-L/14.
- **Parámetros de UNet:** Aproximadamente 860 millones.

5.4.3. Evaluación inicial del modelo

Como prueba exploratoria se utilizó el siguiente prompt genérico:

“A beautiful landscape with mountains and a river”

La imagen generada (Figura 14) reflejó una notable fidelidad visual y semántica con respecto al texto, validando la capacidad del modelo para interpretar y representar con precisión elementos geográficos y naturales. Se observaron detalles como la correcta proporción entre los elementos, una distribución armónica del paisaje y un estilo visual coherente con la descripción dada.

Este resultado preliminar puso de manifiesto la solidez del modelo en tareas de generación generalista, especialmente con prompts descriptivos de carácter amplio. Asimismo, sirvió como punto de partida para contrastar su rendimiento en escenarios más exigentes o especializados, como la generación de razas de perros o conceptos personalizados que requieren una mayor precisión semántica.



Figura 14: Imagen generada con Stable Diffusion v1.4 a partir del prompt *“A beautiful landscape with mountains and a river”*.

5.4.4. Exploración de técnicas de optimización

Pese a los buenos resultados iniciales, se evaluaron diversas estrategias para personalizar o mejorar el rendimiento del modelo. Estas técnicas permiten adaptar el modelo a tareas específicas, aumentar su capacidad expresiva o mejorar la coherencia semántica de las imágenes generadas con respecto al texto de entrada. A continuación se describen las principales aproximaciones consideradas:

- **Fine-tuning del modelo completo:** Ajuste de todos los pesos del modelo utilizando nuevos datos específicos. Esta técnica permite una especialización profunda, aunque conlleva un alto coste computacional y riesgo de sobreajuste si el conjunto de datos es reducido.
- **Modificación del espacio latente:** Rediseño o ajuste del espacio latente en el que se realiza la generación, con el fin de mejorar la calidad representacional. Al trabajar en una representación comprimida, pequeñas mejoras en este espacio pueden traducirse en cambios significativos en la calidad y coherencia de las imágenes generadas.
- **Reemplazo del VAE:** Sustitución del autoencoder variacional por uno más avanzado o con mejores propiedades de reconstrucción, lo cual puede influir positivamente en el nivel de detalle de las imágenes y reducir artefactos visuales en la salida final.

- **Técnica LoRA:** (Low-Rank Adaptation) Permite modificar solo un subconjunto reducido de los parámetros del modelo mediante descomposición de matrices. Esto hace posible una adaptación eficiente con muy pocos recursos, sin necesidad de reentrenar el modelo completo.
- **Cambio de función de pérdida:** Implementación de nuevas funciones de coste que optimicen no solo la precisión pixel a pixel, sino también la coherencia semántica o perceptual. Se pueden emplear pérdidas basadas en embeddings de CLIP o en distancias perceptuales como LPIPS.
- **Modificaciones estructurales en UNet:** Inclusión de capas de atención adicionales (como Self-Attention o Cross-Attention), ajuste del número de bloques residuales o cambios en la arquitectura general. Estas modificaciones pueden aumentar la capacidad del modelo para capturar dependencias espaciales complejas.

Cada una de estas técnicas fue evaluada en términos de complejidad, coste computacional y mejora esperada en la calidad de las imágenes generadas. Se priorizó la búsqueda de un enfoque que permitiera una especialización rápida y efectiva sin requerir un reentrenamiento exhaustivo del modelo completo.

5.4.5. Optimización seleccionada: modificación del espacio latente

Tras comparar las distintas técnicas, se optó por modificar el espacio latente utilizando un enfoque inspirado en DreamBooth, que permite introducir nuevas clases visuales en el modelo sin reentrenar su totalidad. Esta técnica se basa en un entrenamiento ligero y dirigido, en el que el modelo aprende a asociar un término inventado con un concepto visual específico.

Configuración del proceso

Para llevar a cabo la especialización del modelo, se utilizó el siguiente conjunto de parámetros y configuraciones:

- **Datos:** 60 imágenes de perros del dataset Stanford Dogs.
- **Prompt condicional:** ‘‘a photo of a sks dog’’.
- **Entrenamiento:**
 - Congelación de CLIPTextModel y AutoencoderKL.
 - Entrenamiento solo de UNet2DConditionModel.
 - Optimización con AdamW, batch size 1, learning rate 5×10^{-6} .
 - 1000 pasos de entrenamiento.

Resultados

El proceso de especialización del modelo se completó en aproximadamente 5 horas, utilizando un servidor con dos GPUs NVIDIA TITAN RTX. El modelo resultante mostró una notable mejora en la generación de imágenes de perros: las imágenes generadas reflejaban con alta precisión los rasgos morfológicos de cada raza, se mantuvo la coherencia semántica con los nuevos *prompts* específicos, y el entrenamiento resultó eficiente en tiempo y recursos, siendo compatible con entornos como Kaggle o servidores personales con GPU.

Como se ilustra en la Figura 15, el modelo es capaz de generar imágenes realistas y coherentes a partir de descripciones detalladas, como el prompt ‘‘a photo of a golden retriever wearing sunglasses’’, capturando tanto la raza como los elementos distintivos mencionados en el texto.



Figura 15: Ejemplos de imágenes generadas tras la especialización del modelo con el prompt “*a photo of a golden retriever wearing sunglasses*”.

5.5. Evaluación y resultados

Para validar la efectividad del proceso de especialización, se llevó a cabo una evaluación comparativa entre el modelo preentrenado y el modelo ajustado. Esta evaluación se centró en tres aspectos clave: fidelidad visual, coherencia semántica y adecuación morfológica a partir de un mismo prompt textual.

5.5.1. Limitaciones del modelo base

Aunque el modelo preentrenado Stable Diffusion proporciona resultados visuales aceptables en muchos casos, se observaron importantes deficiencias al generar imágenes de conceptos específicos. Estas limitaciones afectan principalmente la coherencia anatómica y el realismo visual, lo que compromete su aplicabilidad en contextos especializados. A modo de ejemplo, se utilizó el prompt: “*a photo of a golden retriever*”.

Resultado con el modelo preentrenado

La imagen generada por el modelo base presenta notables defectos: desalineación de los ojos, artefactos digitales en el hocico y una postura general poco natural. Aunque el color del pelaje sugeriría la raza objetivo, la representación morfológica no es fiel ni reconocible.



Figura 16: Resultado generado por el modelo preentrenado para el prompt: “*a photo of a golden retriever*”.

Resultado tras la especialización del modelo

Tras aplicar la técnica de modificación del espacio latente, el modelo fue capaz de representar de forma mucho más precisa la raza solicitada. La imagen resultante muestra un perro con expresión realista, pelaje detallado, proporciones correctas y una pose reconocible. El resultado evidencia una mejora tanto estética como semántica.



Figura 17: Resultado generado tras la especialización del modelo para el prompt: “*a photo of a golden retriever*”.

5.5.2. Evaluación de coherencia semántica con CLIP

Para respaldar cuantitativamente esta mejora, se empleó el modelo CLIP ViT-L/14, que permite calcular la similitud entre imágenes y texto. Se utilizó el concepto de *CLIP Score relativo*, que compara la afinidad de dos imágenes frente a un mismo prompt.

Imagen generada	CLIP Score relativo
Modelo preentrenado	0.0659
Modelo especializado	0.9341

Tabla 6: Similitud relativa medida con CLIP

Para facilitar la interpretación de los resultados, la Figura 18 muestra una representación gráfica de los valores obtenidos. Esta visualización permite apreciar de forma más clara la diferencia de rendimiento semántico entre ambos modelos ante un mismo prompt, reflejando el impacto directo de la especialización sobre la alineación texto-imagen.

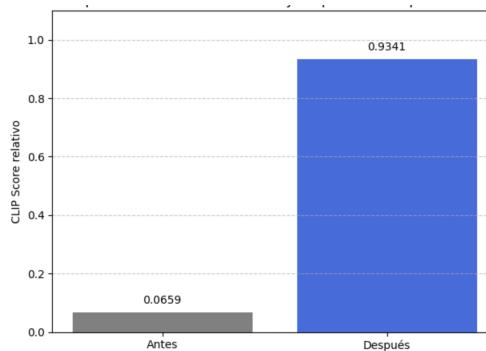


Figura 18: Comparación visual del CLIP Score relativo.

En esta gráfica se visualiza de forma clara la mejora lograda. Mientras que el modelo original apenas logra asociar la imagen con el concepto de “golden retriever”, el modelo especializado alcanza un nivel de coherencia semántica casi perfecto. Esto confirma que la técnica utilizada no solo mejora la calidad visual, sino también la capacidad del modelo para representar correctamente conceptos específicos.

La mejora obtenida es significativa: el modelo especializado alcanza una puntuación de similitud casi 15 veces superior a la del modelo base. Este incremento cuantitativo confirma que la adaptación del espacio latente no solo mejora la fidelidad visual, sino que también optimiza la coherencia semántica desde la perspectiva de modelos multimodales como CLIP, lo que refuerza su utilidad en tareas de búsqueda visual guiada por lenguaje.

5.5.3. Análisis del coste de entrenamiento

Para evaluar la viabilidad del entrenamiento personalizado, se midieron los principales factores que impactan en el coste computacional del proceso. Estos valores permiten estimar la escalabilidad del enfoque en distintos entornos de ejecución.

Recurso	Especificaciones del servidor utilizado
Procesador	AMD Ryzen Threadripper 2920X, 12 núcleos físicos, 24 hilos, hasta 3.5 GHz
Memoria RAM	62 GiB totales, disponibles: 60 GiB libres
GPU	2x NVIDIA TITAN RTX, 24 GiB de VRAM cada una
Sistema operativo	Ubuntu 24.04, kernel 6.8.0-59-generic, arquitectura x86_64
CUDA y Drivers	CUDA 12.4, Driver NVIDIA 550.163.01
Duración del entrenamiento	Aprox. 5 horas
Tamaño del modelo entrenado	4.0K (tamaño en disco del directorio stable-dog-output)
Frameworks utilizados	diffusers , transformers , PyTorch, torchvision
Técnicas aplicadas	DreamBooth, checkpointing, half precision (float16), batch size adaptativo

Tabla 7: Recursos técnicos del servidor utilizado para el entrenamiento final

5.5.4. Evaluación de la generalización del modelo

Además de mejorar la generación de razas específicas como el *golden retriever*, resulta clave comprobar si el modelo especializado conserva su capacidad para generar imágenes no relacionadas con el entrenamiento. Para ello, se utilizó un prompt genérico:

“a man sitting on a bench in a park”

La generación se realizó antes y después de aplicar la especificación para evaluar posibles pérdidas de generalidad.

Antes del entrenamiento especializado

El modelo preentrenado genera una escena coherente: un hombre de espaldas sentado en un banco, con árboles bien definidos y composición equilibrada. El resultado es visualmente aceptable y semánticamente correcto.



Figura 19: Imagen generada por el modelo base con el prompt “*a man sitting on a bench in a park*”.

Después del entrenamiento especializado

Tras la especialización en razas de perro, el modelo mantiene su capacidad para representar correctamente conceptos no entrenados. La escena generada presenta un entorno similar, con árboles, banco y persona reconocibles, sin signos de sobreajuste. Esto sugiere que la adaptación ha sido localizada y no ha afectado negativamente a la generalización.



Figura 20: Imagen generada por el modelo especializado con el mismo prompt: “*a man sitting on a bench in a park*”.

Conclusión

La comparación cualitativa sugiere que el modelo mantiene una buena capacidad de generalización tras la especialización. Es capaz de generar imágenes coherentes incluso para descripciones no incluidas en el entrenamiento, lo que refuerza la aplicabilidad del enfoque de DreamBooth en contextos donde es crucial preservar el conocimiento base del modelo.

5.6. Pruebas del sistema de recuperación de imágenes

Tras validar la generación y la calidad de las imágenes mediante el sistema desarrollado, se procedió a evaluar su uso como punto de partida para tareas de recuperación visual dentro de la plataforma JMR. El objetivo de estas pruebas fue comprobar que las imágenes generadas por el modelo podían utilizarse eficazmente como consultas en un sistema basado en similitud visual, recuperando contenidos relevantes desde una base de datos preexistente.

5.6.1. Metodología de recuperación

Cada prueba consistió en generar una imagen a partir de un *prompt* textual y utilizarla como imagen de consulta en el sistema de recuperación implementado. Para llevar a cabo la comparación visual, se utilizaron los descriptores definidos por el estándar MPEG-7, explicados previamente en el Estado del Arte (Sección 2.3). Concretamente, se aplicaron los siguientes descriptores de color:

- **Descriptor de estructura de color**, que aporta sensibilidad espacial y permite diferenciar imágenes con similares proporciones de color pero distinta disposición.
- **Descriptor de color escalable**, empleado en algunos casos para observar el comportamiento a distintos niveles de granularidad.

5.6.2. Casos de prueba

A continuación se presentan tres casos representativos en los que se utilizó una imagen generada como consulta y se evaluó el resultado de la recuperación visual, teniendo en cuenta distintas casuísticas de interacción:

- **Caso 1 – Recuperación de razas de perro (Golden Retriever)**: Se generó la imagen a partir del prompt “*a golden retriever in the park*” y, sin visualizarla explícitamente, se utilizó directamente desde el cuadro de texto superior como imagen de consulta. El sistema recuperó imágenes de perros de aspecto y coloración similares, como se muestra en la Figura 21, validando la efectividad del descriptor de estructura de color para este tipo de contenido.

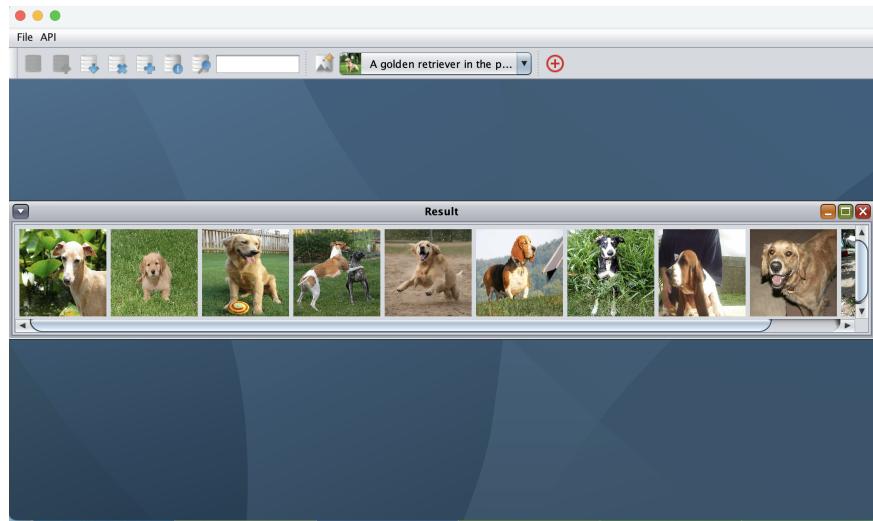


Figura 21: Recuperación visual a partir de la imagen generada con el prompt “*a golden retriever in the park*”.

- **Caso 2 – Recuperación de patrones cromáticos abstractos**: En este experimento se usó el prompt “*a lot of colorful sprinkles*”. De forma análoga al caso anterior, la búsqueda se lanzó directamente desde el cuadro de texto superior sin abrir previamente la imagen generada. A pesar de la

naturaleza abstracta de la imagen, el descriptor de estructura de color fue capaz de identificar con precisión imágenes con patrones cromáticos y distribución similares. Los resultados pueden observarse en la Figura 22.

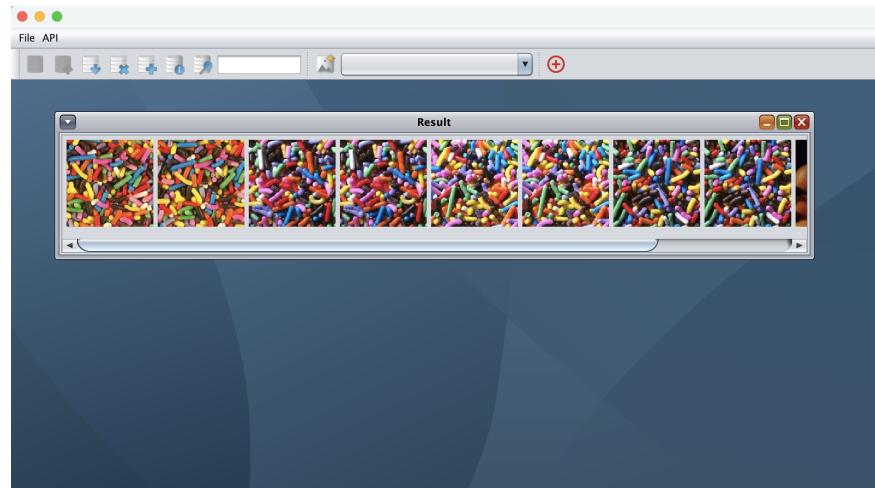


Figura 22: Recuperación visual a partir de la imagen generada con el prompt “*a lot of colorful sprinkles*”.

- **Caso 3 – Recuperación de paisajes artísticos:** En este caso, la imagen fue generada con el prompt “*a cloudy day*” y visualizada previamente en la ventana de imagen interna. Posteriormente, se utilizó dicha imagen visualizada como entrada para la búsqueda por similitud. La Figura 23 muestra los resultados, que incluyen paisajes con estructuras visuales y cromáticas similares, reflejando la sensibilidad del sistema a los tonos verdes, marrones y grises predominantes.

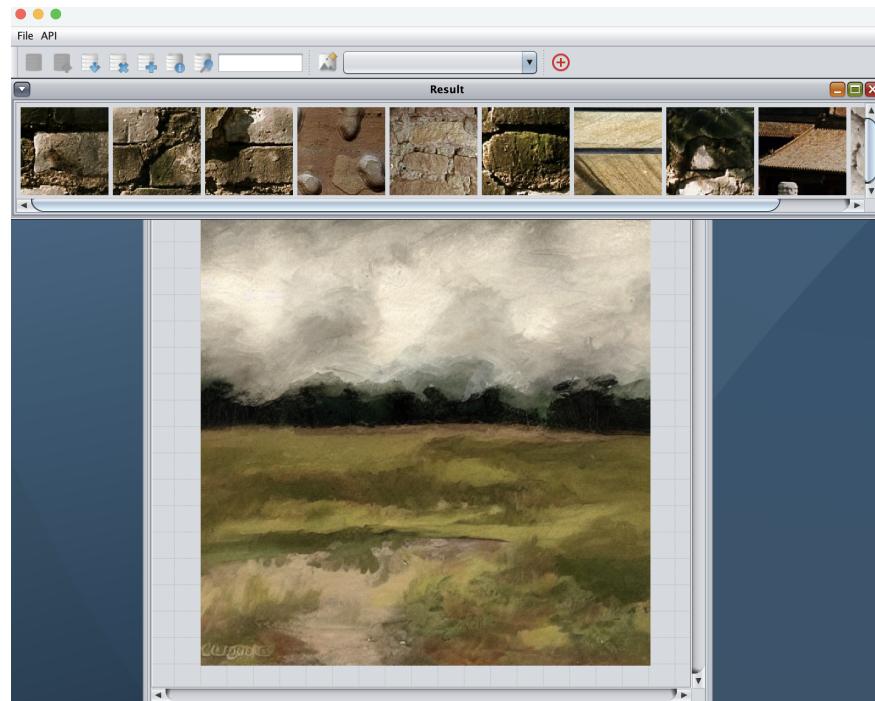


Figura 23: Recuperación visual a partir de la imagen generada con el prompt “*a cloudy day*”, tras visualizarla previamente en la aplicación.

5.6.3. Análisis de resultados

Los casos de prueba realizados muestran un comportamiento coherente del sistema en distintos contextos de consulta. El uso de imágenes generadas como entrada ha resultado eficaz tanto en escenas estructuradas (como retratos de animales) como en patrones visuales abstractos y paisajes artísticos.

Se observó que el descriptor de estructura de color fue especialmente robusto en todos los casos, logrando identificar similitudes perceptuales relevantes incluso cuando las imágenes diferían en contenido semántico pero compartían distribuciones cromáticas o composiciones similares. En el caso del paisaje (*a cloudy day*), la inclusión del descriptor de color escalable permitió una recuperación más precisa en niveles de detalle finos, especialmente al tratarse de gradientes suaves y variaciones tonales sutiles.

En cuanto a las formas de interacción, tanto la búsqueda directa desde el cuadro de texto como la reutilización tras visualizar la imagen demostraron ser funcionales. La posibilidad de lanzar consultas sin necesidad de abrir manualmente la imagen generada optimiza el flujo de trabajo, mientras que la reutilización posterior ofrece flexibilidad en casos donde se desea confirmar visualmente el contenido antes de iniciar la recuperación.

En resumen, los resultados refuerzan la idea de que el sistema no solo genera imágenes coherentes, sino que también las convierte en representaciones visuales útiles para tareas de recuperación basada en contenido, cerrando así el ciclo entre lenguaje natural, generación visual e indexación semántica.

5.7. Conclusiones

Los resultados obtenidos demuestran que las imágenes generadas por el sistema no solo presentan coherencia visual y semántica con los *prompts* originales, sino que también son efectivas como entrada para tareas de recuperación basada en contenido visual. Esta doble funcionalidad—generación e indexación—consolida el papel del sistema como puente entre el lenguaje natural y la búsqueda visual, permitiendo transformar descripciones textuales en imágenes interpretables por algoritmos de recuperación.

En los distintos casos de estudio realizados, se ha evidenciado la versatilidad del sistema ante diversos tipos de contenido:

- En el caso del **golden retriever**, la imagen generada permitió recuperar con éxito otras imágenes de perros de aspecto similar, validando la capacidad del sistema para tratar conceptos visuales bien definidos y estructurados.
- En la prueba con **sprinkles de colores**, se comprobó la sensibilidad del sistema a patrones cromáticos abstractos, lo que pone de manifiesto su aplicabilidad más allá de objetos concretos.
- En el escenario del **paisaje nublado**, la combinación de descriptores permitió capturar matices tonales y composiciones complejas, como las propias de escenas artísticas o estilizadas.

A nivel de implementación, la utilización de los descriptores visuales definidos por MPEG-7—en particular el de *estructura de color* y, en menor medida, el *color scalable*—resultó una elección acertada. Ambos descriptores ofrecieron un equilibrio adecuado entre coste computacional y capacidad discriminativa, permitiendo recuperar imágenes relevantes incluso en bases de datos con alta variabilidad visual.

Por otro lado, las distintas casuísticas de uso (búsqueda directa desde el cuadro de texto o reutilización tras visualización) demostraron que el sistema se adapta a diferentes flujos de trabajo, manteniendo la consistencia funcional en todos los casos.

En definitiva, se ha validado que el sistema desarrollado puede integrarse eficazmente en plataformas de recuperación visual y servir como base para futuros desarrollos donde la interacción entre texto e imagen sea fundamental. Esto abre la puerta a aplicaciones en múltiples dominios como educación, accesibilidad, archivos visuales o interfaces conversacionales basadas en imágenes generadas.

Parte II

Desarrollo de software

6. Requisitos y análisis

El análisis del sistema es una etapa esencial en el desarrollo de software. En esta se examinan las necesidades, requisitos y funcionalidades del sistema a implementar, ayudando a la comprensión de los usuarios finales, sus contextos de uso y los objetivos que esperan alcanzar.

6.1. Especificación de requisitos

Los requisitos del sistema se dividen en dos grandes categorías: funcionales y no funcionales. Estos permiten establecer de forma clara qué debe cumplir la solución propuesta para ser viable y útil. A continuación, se presenta una recopilación estructurada de los requisitos identificados.

6.1.1. Requisitos funcionales

Los siguientes requisitos funcionales describen las capacidades que el sistema debe implementar para cumplir con los objetivos definidos.

Código	RF 1.1
Nombre	Generación de Imágenes Basada en Texto
Descripción	Permitir la generación de imágenes a partir de descripciones textuales proporcionadas por el usuario
Precondición	Acceso al sistema de generación de imágenes mediante la interfaz de usuario
Postcondición	Imagen generada y visualizada en la interfaz del usuario

Tabla 8: Requisito Funcional RF 1.1

Código	RF 2.1
Nombre	Compatibilidad y Escalabilidad en la JMR
Descripción	Integración del sistema con la infraestructura existente en la JMR sin interrumpir otras funcionalidades
Precondición	Acceso a la infraestructura actual de la JMR
Postcondición	El sistema de generación de imágenes opera correctamente dentro del entorno de la JMR

Tabla 9: Requisito Funcional RF 2.1

Código	RF 3.1
Nombre	Gestión de Resultados de Generación de Imágenes
Descripción	Permitir a los usuarios guardar y etiquetar las imágenes generadas
Precondición	Imagen generada y lista para almacenamiento o uso posterior
Postcondición	La imagen está guardada y etiquetada correctamente

Tabla 10: Requisito Funcional RF 3.1

Código	RF 4.1
Nombre	Interfaz de Usuario Intuitiva y Accesible
Descripción	Proporcionar una interfaz amigable y accesible para que los usuarios generen imágenes a partir de texto
Precondición	Acceso al sistema de generación de imágenes a través de la interfaz
Postcondición	Los usuarios pueden navegar y utilizar fácilmente las funcionalidades

Tabla 11: Requisito Funcional RF 4.1

6.1.2. Requisitos no funcionales

Los requisitos no funcionales del sistema establecen criterios relacionados con la calidad, rendimiento, mantenimiento y accesibilidad.

Código	RNF 1
Nombre	Rendimiento y Eficiencia
Descripción	El sistema debe ser capaz de generar imágenes en un tiempo razonable, optimizando el uso de memoria y recursos de GPU

Tabla 12: Requisito No Funcional RNF 1: Rendimiento y Eficiencia

Código	RNF 2
Nombre	Mantenimiento
Descripción	El sistema debe ser modular, permitiendo actualizaciones y mantenimiento sin afectar el funcionamiento general de la JMR

Tabla 13: Requisito No Funcional RNF 2: Mantenimiento

Código	RNF 3
Nombre	Compatibilidad
Descripción	El sistema debe ser compatible con la infraestructura actual de la JMR. También debe permitir la integración con otros módulos o sistemas externos en el futuro.

Tabla 14: Requisito No Funcional RNF 3: Compatibilidad

Código	RNF 4
Nombre	Experiencia de Usuario y Accesibilidad
Descripción	El sistema debe cumplir con los estándares de accesibilidad para asegurar su uso por personas con diferentes capacidades y necesidades. La interfaz debe ser responsive y adaptarse correctamente a distintos dispositivos.

Tabla 15: Requisito No Funcional RNF 4: Experiencia de Usuario y Accesibilidad

6.2. Historias de usuario

Con el objetivo de validar y contextualizar los requisitos identificados, se han definido varias historias de usuario que ilustran cómo diferentes perfiles interactúan con el sistema. Estas historias permiten anticipar escenarios de uso realistas y orientar el diseño de funcionalidades clave.

La Tabla 16 presenta el caso de un investigador, cuyo objetivo es experimentar con modelos de generación de imágenes y evaluar su precisión antes de integrarlos en la plataforma. Este perfil pone el foco en la calidad técnica del sistema generativo y su interoperabilidad con el entorno de recuperación visual.

Historia de Usuario 1	
Rol del Usuario	Investigador en Inteligencia Artificial y Aprendizaje Profundo
Necesidad	Desarrollar y evaluar modelos de generación de imágenes a partir de texto para integrarlos en una plataforma de recuperación de información visual (JMR)
Funcionalidades	Implementar un modelo optimizado para generación de imágenes basada en descripciones textuales Evaluar la precisión y coherencia de las imágenes generadas en relación a la descripción Proporcionar un módulo de prueba y validación de las imágenes generadas antes de la integración en la JMR
Beneficio	Avanzar en las capacidades de los sistemas CBIR al permitir búsquedas más naturales y accesibles basadas en texto, mejorando la precisión en la recuperación de imágenes

Tabla 16: Historia de Usuario 1

La Tabla 17 aborda el punto de vista del administrador de la JMR, responsable de integrar y mantener el sistema dentro de una infraestructura ya existente. Aquí, los aspectos de compatibilidad, usabilidad y estabilidad del sistema cobran especial relevancia.

Historia de Usuario 2	
Rol del Usuario	Administrador de la JMR
Necesidad	Integrar un sistema de generación de imágenes a partir de texto que sea compatible con la infraestructura actual de la JMR
Funcionalidades	Implementar una interfaz de usuario intuitiva que permita a los usuarios ingresar descripciones textuales para la generación de imágenes Asegurar la compatibilidad y escalabilidad del módulo de generación de imágenes dentro de la infraestructura existente de la JMR Supervisar el rendimiento y la estabilidad del sistema para garantizar una experiencia fluida al usuario
Beneficio	Ofrecer un sistema CBIR basado en texto que potencie las funcionalidades de la JMR, proporcionando una experiencia avanzada y personalizada de búsqueda visual para los usuarios

Tabla 17: Historia de Usuario 2

Por último, en la Tabla 18 se presenta al usuario final, quien se beneficia directamente de la generación de imágenes como forma de consulta. Este perfil prioriza la facilidad de uso, la calidad de los resultados generados y la capacidad de aplicar el sistema en contextos reales.

Historia de Usuario 3	
Rol del Usuario	Usuario Final del Sistema de Recuperación de Imágenes
Necesidad	Utilizar un sistema que permita recuperar imágenes relevantes basadas en descripciones textuales para apoyar investigaciones o proyectos específicos
Funcionalidades	Generar imágenes basadas en consultas textuales específicas que describan escenarios, objetos o características Acceder a una interfaz intuitiva que permita introducir consultas de texto y visualizar resultados de forma rápida y precisa Poder almacenar o compartir las imágenes generadas para análisis posterior o trabajo colaborativo
Beneficio	Facilitar el acceso a imágenes específicas mediante descripciones, eliminando la necesidad de búsquedas visuales tradicionales y agilizando los procesos de investigación y aprendizaje

Tabla 18: Historia de Usuario 3

El conjunto de estas historias permite cubrir una perspectiva holística del sistema: desde el desarrollo técnico hasta el uso práctico. A continuación, se presenta una tabla que muestra la correspondencia entre los requisitos definidos y las historias de usuario que los motivan. Esta relación asegura que cada funcionalidad o característica del sistema responde directamente a una necesidad concreta, lo que refuerza la coherencia y la orientación al usuario del diseño.

Código del Requisito	Descripción resumida	Historia de Usuario
RF 1.1	Generación de imágenes a partir de texto	HU1, HU3
RF 2.1	Integración con la plataforma JMR	HU2
RF 3.1	Almacenamiento y etiquetado de imágenes	HU3
RF 4.1	Interfaz intuitiva y accesible	HU2, HU3
RNF 1	Eficiencia y rendimiento	HU1, HU2
RNF 2	Mantenimiento modular del sistema	HU2
RNF 3	Compatibilidad con la infraestructura JMR	HU2
RNF 4	Accesibilidad y experiencia de usuario	HU3

Tabla 19: Relación entre requisitos y sus historias de usuario

6.3. Modelo de caso de uso

El modelo de caso de uso presentado en la Figura 24 muestra las interacciones principales entre los actores y las funcionalidades del sistema de generación de imágenes a partir de texto integrado en la plataforma

JMR. Este diagrama incluye la generación de imágenes desde una descripción textual, la visualización y personalización de resultados, el uso opcional de descriptores avanzados ofrecidos por la JMR, y la posibilidad de guardar o reutilizar las imágenes generadas.

Los actores principales identificados en el sistema son:

- **Usuario final:** persona que introduce una descripción textual y visualiza o descarga las imágenes generadas.
- **Administrador:** encargado de configurar parámetros del sistema, gestionar el uso del generador y mantener la integración con la plataforma JMR.

Este modelo permite delimitar con claridad las funcionalidades accesibles para cada tipo de usuario y ha servido como base para la extracción de requisitos funcionales. Además, constituye un punto de partida para el desarrollo de los diagramas de secuencia y de actividad, que detallan la lógica interna del sistema y el flujo de datos entre sus componentes.

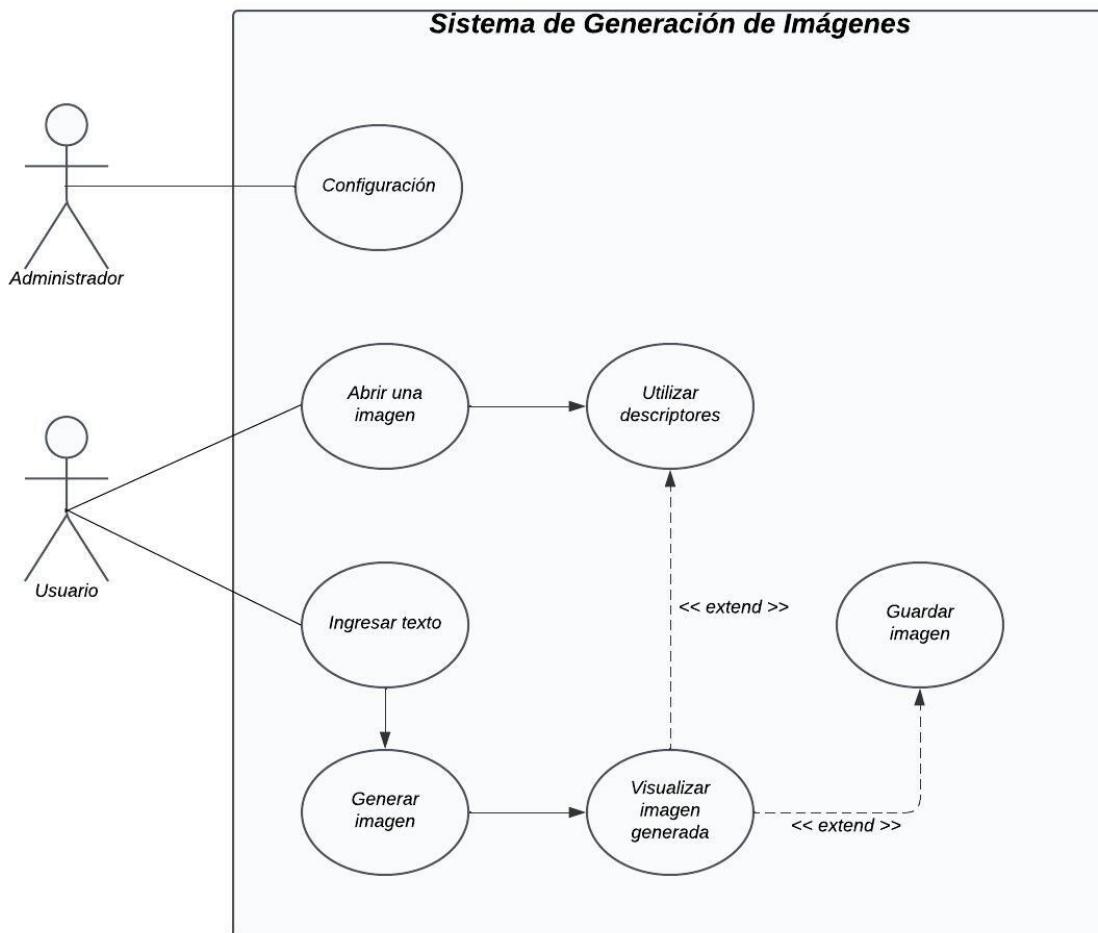


Figura 24: Diagrama de caso de uso

6.4. Modelos de comportamiento

Los modelos de comportamiento permiten representar la lógica dinámica del sistema, mostrando cómo interactúan sus componentes y usuarios en tiempo real. A través de diagramas de secuencia y un diagrama

de actividad, se describen distintos flujos clave: desde la generación de imágenes a partir de texto, hasta la gestión de resultados y el entrenamiento del modelo.

Estos diagramas permiten visualizar cómo se comunican los distintos módulos (como la API de generación, el módulo de entrenamiento o los sistemas de almacenamiento), y ayudan a validar que las funcionalidades implementadas siguen una lógica coherente con los requisitos definidos.

6.4.1. Diagramas de secuencia

Los diagramas de secuencia representan interacciones temporales entre los actores y los componentes internos del sistema, reflejando cómo se produce la comunicación entre ellos en distintos escenarios clave. A continuación, se presentan los diagramas más relevantes que ilustran la lógica de funcionamiento de cada flujo:

- La Figura 25 muestra el proceso mediante el cual un usuario selecciona un descriptor visual desde la interfaz del sistema. El flujo se inicia con la acción del actor, quien elige el descriptor deseado; esta solicitud se transmite al controlador de descriptores, que se encarga de procesarla y consultar al módulo correspondiente. El módulo aplica el descriptor y devuelve el resultado, que finalmente se muestra al usuario.
- La Figura 26 detalla la generación de una imagen a partir de una descripción textual. El actor introduce un prompt desde la interfaz, que es enviado al controlador de generación. Este comunica la petición a la API de generación, la cual procesa el texto y genera la imagen correspondiente. El resultado es devuelto y mostrado al usuario.
- La Figura 27 representa el proceso de almacenamiento de una imagen generada. El usuario elige la opción de guardar y proporciona una ruta de destino. La solicitud es gestionada por el controlador de imágenes, que delega en el sistema de archivos local la escritura del fichero. Una vez finalizado el proceso, se notifica al usuario.
- La Figura 28 ilustra el procedimiento para la obtención y preparación de datos desde el conjunto COCO. El sistema de gestión de datos solicita al servidor las imágenes y descripciones, las cuales se obtienen desde la base de datos. Una vez recuperados, los datos son preprocesados y enviados de vuelta listos para ser utilizados en el entrenamiento.
- La Figura 29 muestra el flujo correspondiente al entrenamiento de un modelo generativo. El sistema de procesamiento de datos envía los datos al servidor de entrenamiento, donde se realiza el preprocesamiento. Posteriormente, el servidor entrena el modelo y almacena los resultados en la base de datos de modelos. Al finalizar, se notifica el éxito del entrenamiento.
- Finalmente, la Figura 30 detalla el cálculo y monitorización de pérdidas durante el proceso de entrenamiento. A partir de la solicitud del usuario, se inicia el entrenamiento y se registran las pérdidas del generador y el discriminador. Estos valores se visualizan gráficamente y se devuelven al sistema como salida.

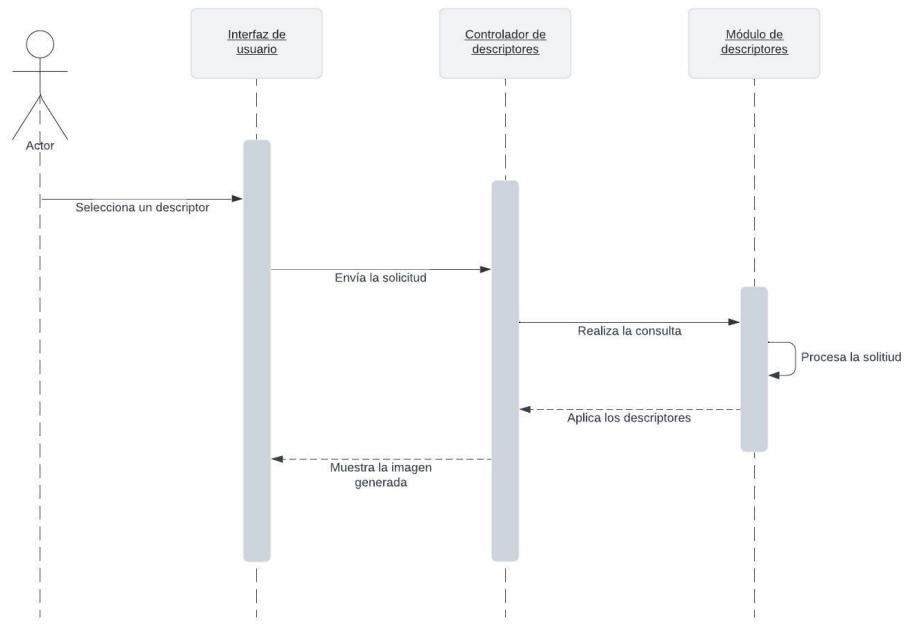


Figura 25: Diagrama de secuencia del uso de descriptoros

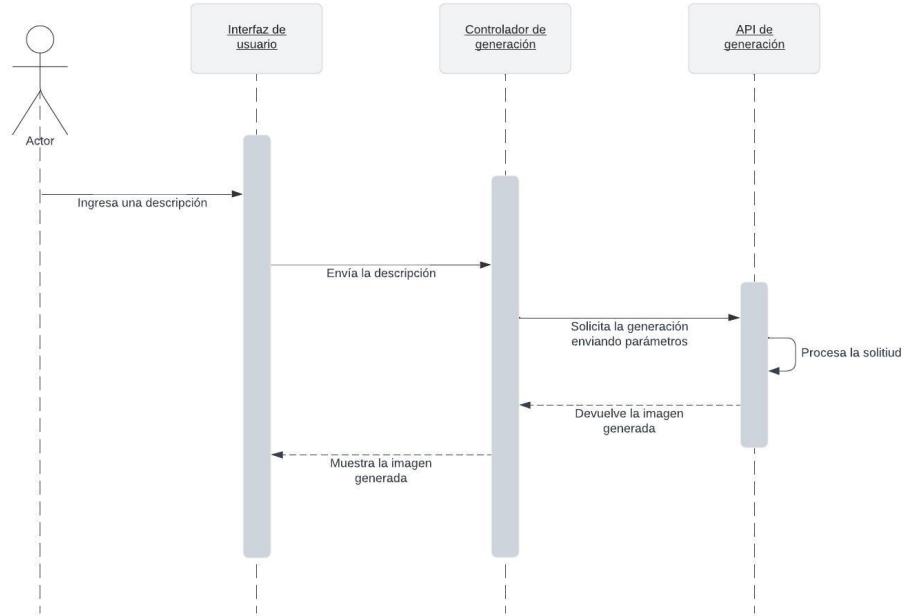


Figura 26: Diagrama de secuencia de la generación de una imagen

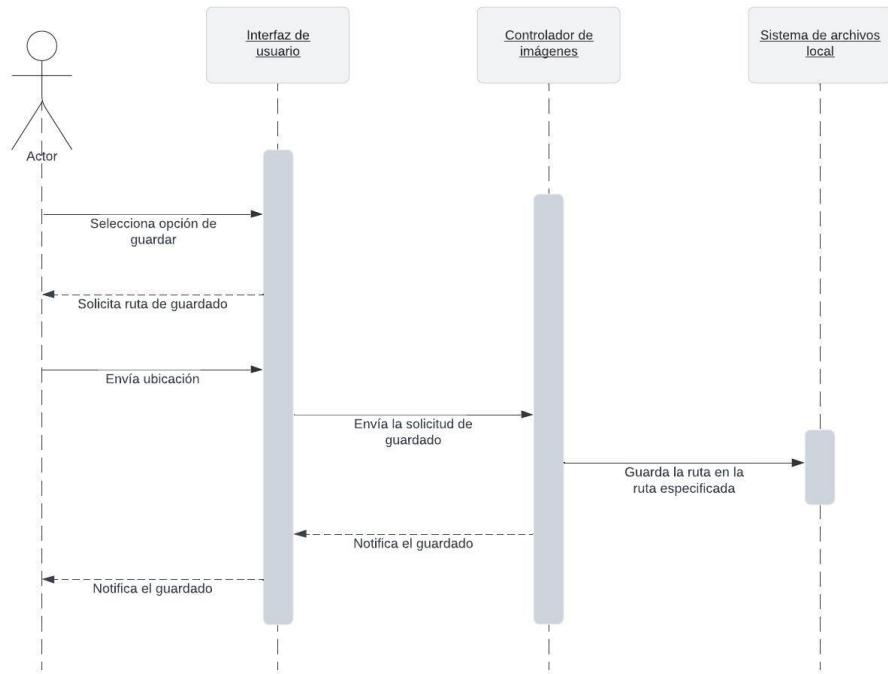


Figura 27: Diagrama de secuencia del guardado de una imagen

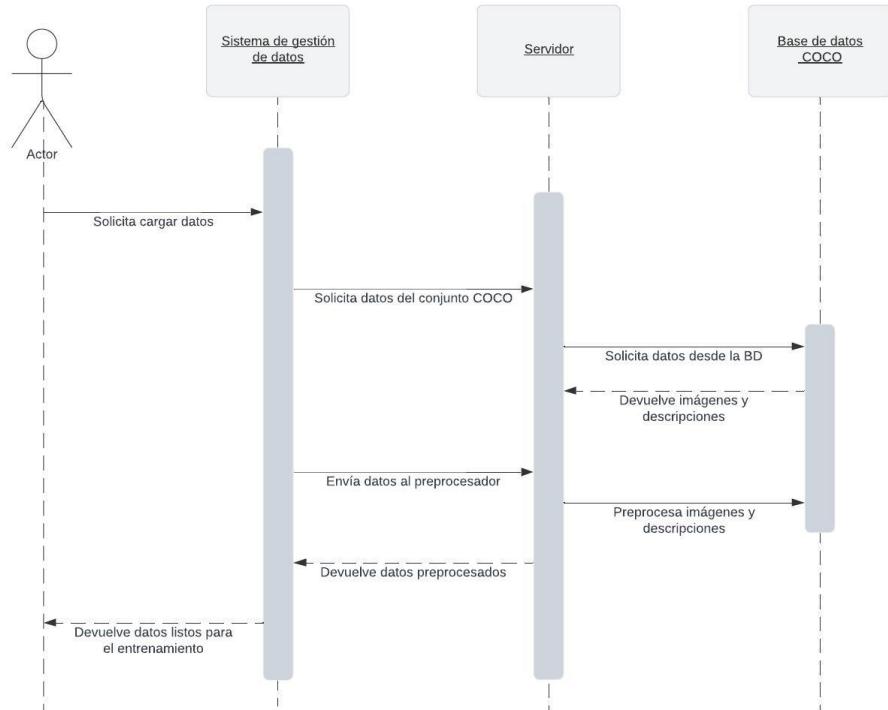


Figura 28: Diagrama de secuencia del conjunto de datos COCO

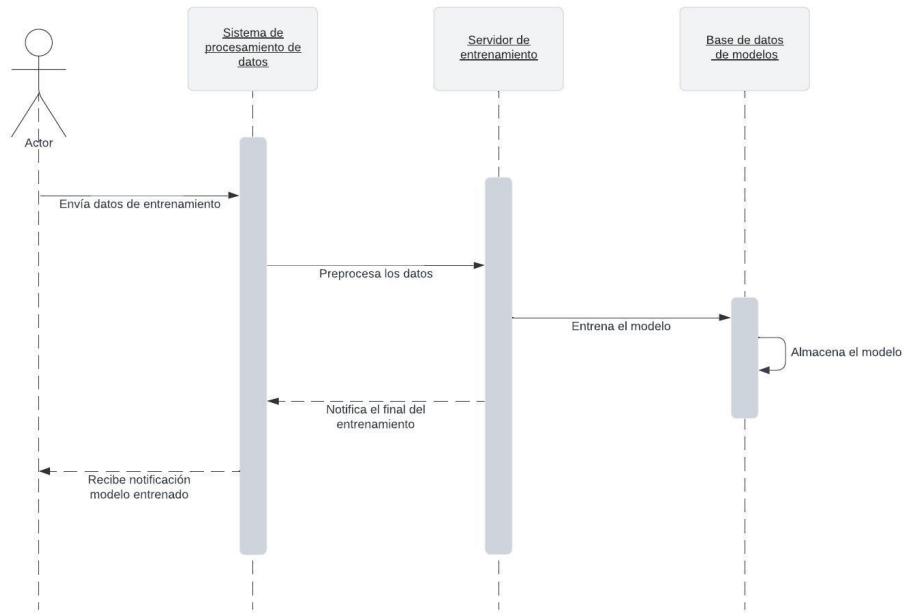


Figura 29: Diagrama de secuencia del entrenamiento

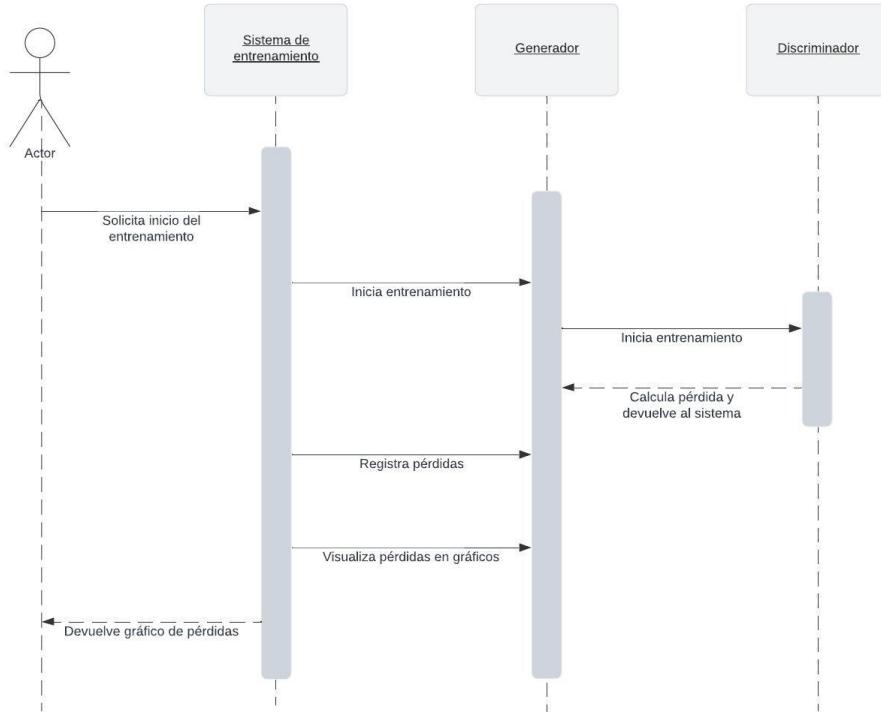


Figura 30: Diagrama de secuencia de pérdidas

6.4.2. Trazabilidad entre casos de uso, historias de usuario y requisitos

Para asegurar la coherencia entre el análisis de requisitos, las historias de usuario y los casos de uso definidos, se ha elaborado una matriz de trazabilidad. Esta permite verificar que cada funcionalidad descrita en los casos de uso responde a una necesidad real expresada por los usuarios, y que cuenta con respaldo técnico mediante los requisitos previamente establecidos. De este modo, se garantiza una cobertura completa de los objetivos del sistema y se facilita su validación durante las fases de diseño e implementación.

Caso de Uso	Descripción	Historias de Usuario	Requisitos
CU1	Generar imagen desde texto	HU1, HU3	RF 1.1, RF 4.1, RNF 1, RNF 4
CU2	Visualizar y guardar resultados	HU3	RF 3.1, RF 4.1, RNF 4
CU3	Configurar y mantener el sistema	HU2	RF 2.1, RNF 2, RNF 3
CU4	Entrenamiento y evaluación de modelo	HU1	RNF 1, RNF 2
CU5	Selección y aplicación de descriptores	HU2	RF 2.1, RNF 3

Tabla 20: Relación entre casos de uso, historias de usuario y requisitos

6.4.3. Diagrama de actividad

La Figura 31 representa el flujo general de uso del sistema desde la perspectiva del usuario final. Este diagrama describe el camino típico que sigue un usuario: desde introducir una descripción, visualizar los resultados generados, hasta decidir si desea guardar o personalizar la imagen. También permite identificar puntos de decisión y posibles bifurcaciones en el flujo de interacción.

El diagrama está dividido en tres columnas que representan los principales participantes del flujo: el Usuario, la Aplicación local y la API de generación. El proceso comienza con la ejecución de la aplicación, seguida de la carga de los datos necesarios. En caso de error durante esta etapa, el sistema informa al usuario y finaliza el flujo.

Si la carga es exitosa, el usuario introduce una descripción textual que es recogida por la aplicación y enviada a la API. Esta procesa la información y genera una imagen, la cual es enviada codificada de vuelta a la aplicación local. Si hay un error en la generación, el sistema puede manejarlo y reportarlo.

La aplicación decodifica la información recibida, muestra la imagen generada y permite al usuario visualizarla. Finalmente, el usuario puede aplicar descriptores adicionales sobre la imagen, completando así el ciclo de interacción. Este flujo refleja de manera clara y estructurada la experiencia de uso prevista para la funcionalidad principal del sistema.

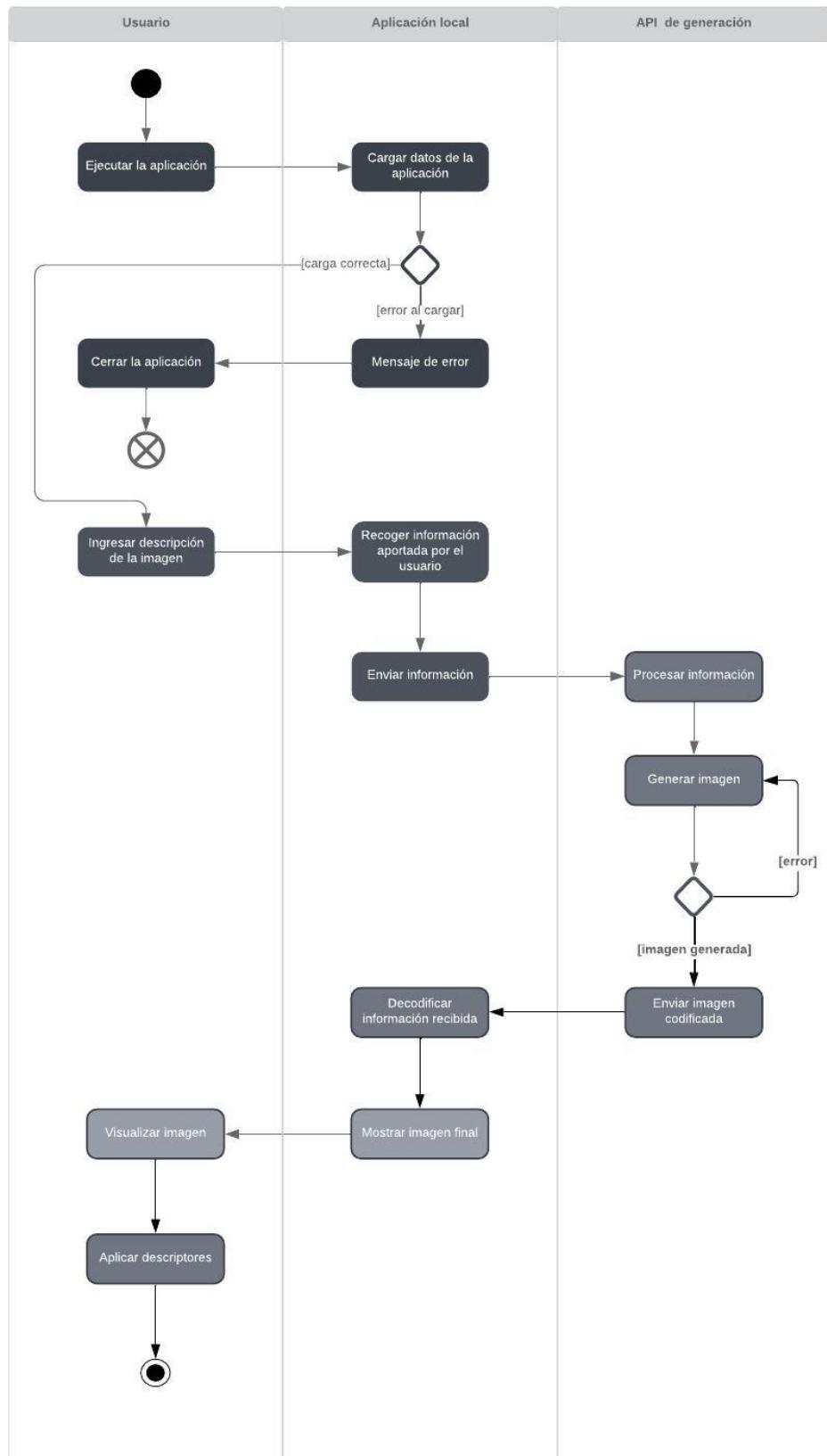


Figura 31: Diagrama de actividad general del sistema

7. Diseño

El diseño del sistema desarrollado ha sido clave para garantizar la viabilidad técnica, la modularidad de los componentes y una experiencia de usuario intuitiva. En este apartado se describen los principales aspectos del diseño arquitectónico, conceptual y de interfaz de la solución implementada, incluyendo tanto el funcionamiento interno de los módulos como su interacción con el usuario final.

7.1. Diseño de la arquitectura

El sistema se integra en la plataforma Java Multimedia Retrieval (JMR), ampliando sus capacidades de búsqueda visual mediante la incorporación de un módulo generativo desarrollado en Python. La arquitectura sigue un enfoque modular y desacoplado, en el que los componentes desarrollados en Python se comunican con JMR a través de una API REST.

- **Interfaz de usuario (JMR):** permite introducir descripciones textuales como entrada para la generación de consultas visuales.
- **Módulo generativo (Python):** expone una API REST que recibe descripciones textuales y devuelve imágenes generadas mediante modelos de difusión. Este módulo está encapsulado en una aplicación ligera desplegable de forma independiente.
- **Módulo de integración (Java):** dentro de JMR, se encarga de enviar peticiones HTTP a la API generativa, recuperar la imagen resultante y tratarla como una consulta visual.
- **Módulo CBIR (JMR):** compara la imagen generada con una base de datos de imágenes mediante descriptoros visuales y métricas de similitud.

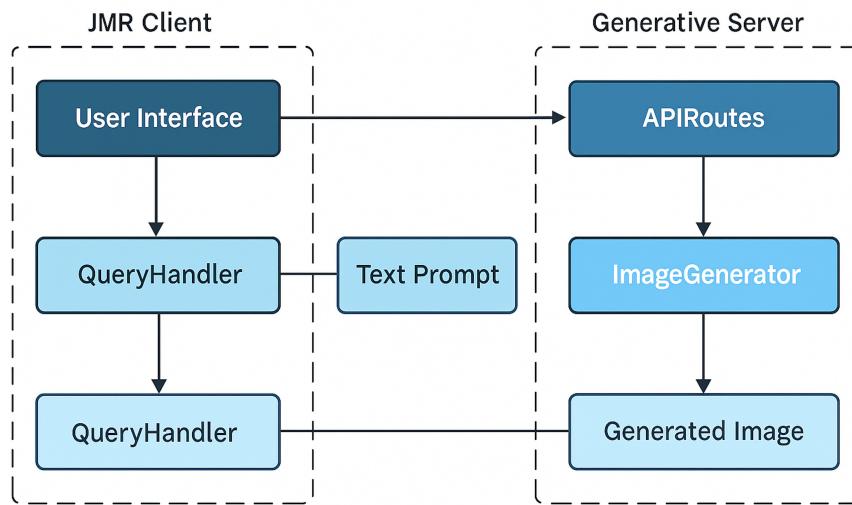


Figura 32: Arquitectura lógica del sistema integrado

Esta arquitectura distribuida permite mejorar la mantenibilidad, facilitar la evolución del modelo generativo y adaptar el sistema a diferentes entornos sin afectar a la aplicación principal.

7.2. Modelo conceptual

El modelo conceptual del sistema refleja los elementos fundamentales que intervienen en el proceso de generación y búsqueda visual. La Figura 33 resume gráficamente este flujo, destacando la interacción entre el usuario, la API generativa y el motor de recuperación visual (CBIR).

- **Usuario:** introduce una descripción textual a través de la interfaz JMR.
- **Prompt o descripción textual:** entrada en lenguaje natural que sirve como semilla para la generación de una imagen.
- **Imagen generada:** salida del modelo de IA a partir del prompt introducido por el usuario.
- **Resultado de búsqueda:** conjunto de imágenes similares recuperadas por el motor CBIR.
- **Configuración del modelo:** conjunto de parámetros que determinan el comportamiento del generador (modelo base, pasos, guidance, etc.).

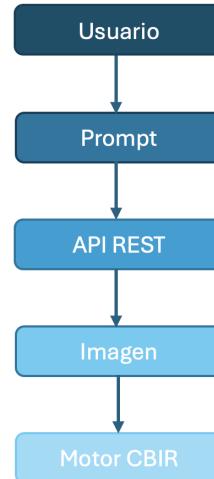


Figura 33: Representación esquemática del modelo conceptual del sistema

Este modelo permitió definir los flujos de información, el diseño de las peticiones API y las entidades clave del sistema.

7.3. Diseño del modelo de clases

Dado que el sistema combina componentes en dos lenguajes distintos, se ha documentado por separado el modelo de clases tanto del cliente Java como del servidor Python.

7.3.1. Modelo de clases en Java

El modelo de clases Java desarrollado para este proyecto abarca dos niveles complementarios de abstracción: por un lado, la lógica de integración con el sistema generativo a través de descriptores de imagen; por otro, la estructura de la interfaz gráfica (JMR) que permite la interacción del usuario con dicho sistema.

En primer lugar, el diseño de la jerarquía de descriptores se recoge en la Figura 34. En el centro de esta jerarquía se encuentra la clase abstracta `AbstractPromptImageDescriptor`, que encapsula el flujo completo desde el prompt textual hasta la generación de la imagen y la extracción de sus descriptores visuales. Esta clase proporciona una interfaz común para todos los descriptores generativos, y delega en sus subclases la implementación concreta del método de generación.

Dos implementaciones heredan de esta clase:

- `PromptGeneratedImageDescriptor`, que realiza llamadas a una API REST para generar la imagen desde un modelo online.
- `PromptGeneratedImageDescriptorLocal`, que opera contra una instancia local de la API, útil en entornos sin conectividad externa.

Ambos descriptores se apoyan en clases como `DescriptorList` y utilizan un `Comparator` genérico para facilitar la comparación y evaluación posterior de las imágenes generadas. Esta arquitectura está diseñada para ser fácilmente extensible, permitiendo incorporar nuevos mecanismos de generación sin alterar el resto del sistema.

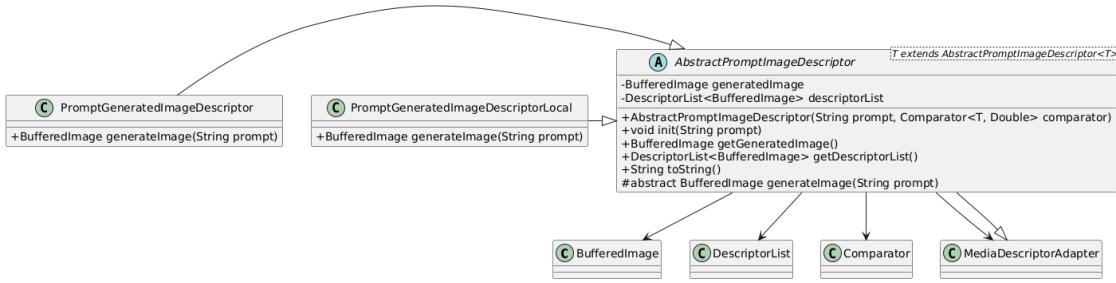


Figura 34: Diagrama de clases de los descriptores generativos en Java

Por otro lado, la estructura de la interfaz de usuario de JMR se representa en la Figura 35. En este nivel, la clase principal `MainWindow` actúa como núcleo de la aplicación cliente, gestionando la inicialización de componentes, el manejo de ventanas internas y la interacción con el usuario. Esta clase instancia los descriptores adecuados en función del modo seleccionado (local u online) y se comunica con el resto de la interfaz para mostrar, almacenar y reutilizar los resultados generados.

`MainWindow` interactúa con las siguientes clases clave:

- `PromptWindow`, que permite introducir prompts y lanzar la generación.
- `InternalWindow` y `ListInternalWindow`, dedicadas a mostrar imágenes individuales o resultados de búsqueda visual.
- `ImagePromptItem`, que almacena pares prompt-imagen y facilita su recuperación desde el historial.
- Clases auxiliares como `Circulo` o `ColorSetPanel`, utilizadas en funcionalidades visuales complementarias.

Este diseño modular facilita la independencia entre interfaz y lógica generativa, lo que permite actualizar o sustituir el backend sin necesidad de modificar la interfaz. Asimismo, permite escalar el sistema hacia nuevos escenarios de uso o modelos generativos adicionales.

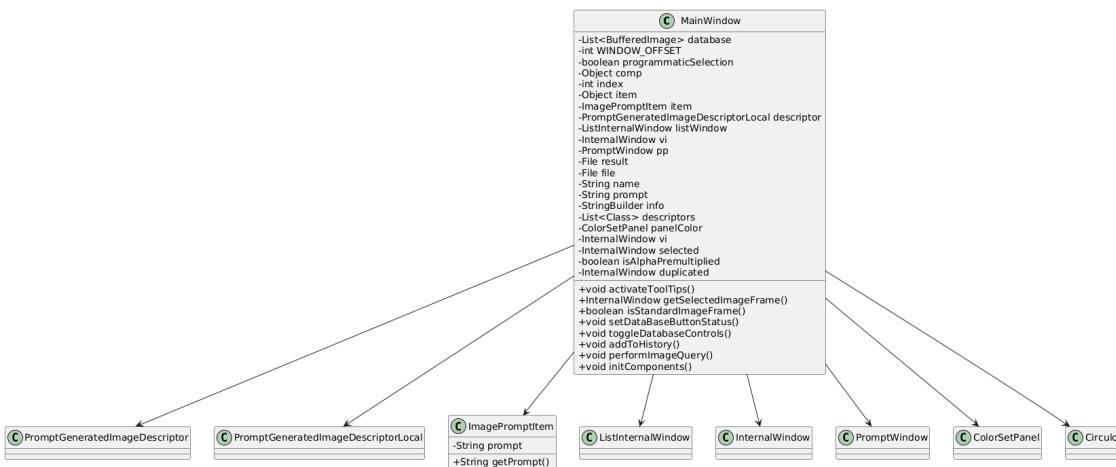


Figura 35: Diagrama de clases del módulo de integración en Java

7.3.2. Modelo de clases en Python

El modelo de clases en Python, representado como una abstracción de los módulos funcionales, refleja la lógica interna del sistema generativo. En el centro se encuentra la clase conceptual `DogTrainer`, que agrupa todos los elementos necesarios para el entrenamiento y generación de imágenes mediante DreamBooth.

Esta clase maneja:

- La descarga y filtrado del dataset (`download_and_extract_dataset()`, `select_images()`).
- El entrenamiento del modelo personalizado (`train_dreambooth()`).
- La generación de imágenes y compresión de resultados (`generate_images()`, `zip_results()`).

Además, se incluye la clase `DogDataset`, que estructura el conjunto de datos utilizado para el entrenamiento, y `GenerateRequest`, que define los parámetros necesarios para realizar una petición de generación.

En cuanto a la API, se representan los distintos módulos como clases con sus funciones principales:

- `ModelOperations` para gestión de modelos (`upload`, `delete`, `list`),
- `ImageOperations` para generación y descarga de imágenes,
- `Validation` para la comprobación estructural de los modelos subidos.

Este enfoque facilita la mantenibilidad del sistema y su extensión a nuevas funcionalidades, garantizando una separación clara entre el entrenamiento, la generación y la exposición mediante API.

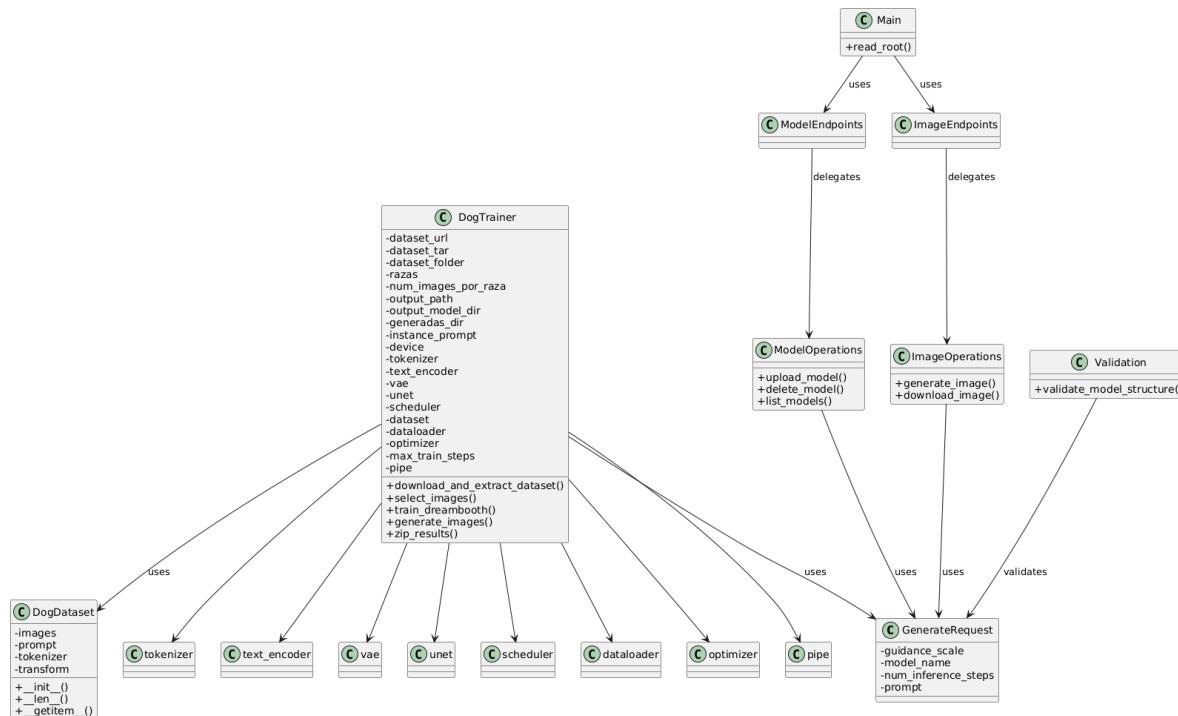


Figura 36: Diagrama de clases del sistema generativo en Python

7.4. Diseño de la interfaz

El diseño de la interfaz de usuario ha sido un componente clave para facilitar la interacción con el sistema de generación y búsqueda de imágenes. Este apartado presenta la evolución del diseño, desde los primeros esquemas conceptuales hasta el prototipo final interactivo. Se parte del análisis de flujo de interacción, se muestran los bocetos iniciales y wireframes funcionales, y finalmente se presenta el prototipo desarrollado en Figma. Además, se analizan los principios de usabilidad aplicados para garantizar una experiencia fluida e intuitiva para el usuario final.

7.4.1. Diagrama de flujo de interacción

El siguiente diagrama de flujo describe la secuencia de operaciones que se ejecutan desde la introducción del prompt por parte del usuario hasta la obtención de los resultados visuales. Este flujo ilustra la lógica general del sistema, destacando las decisiones clave y la coordinación entre los distintos módulos (interfaz gráfica, API de generación y sistema CBIR).

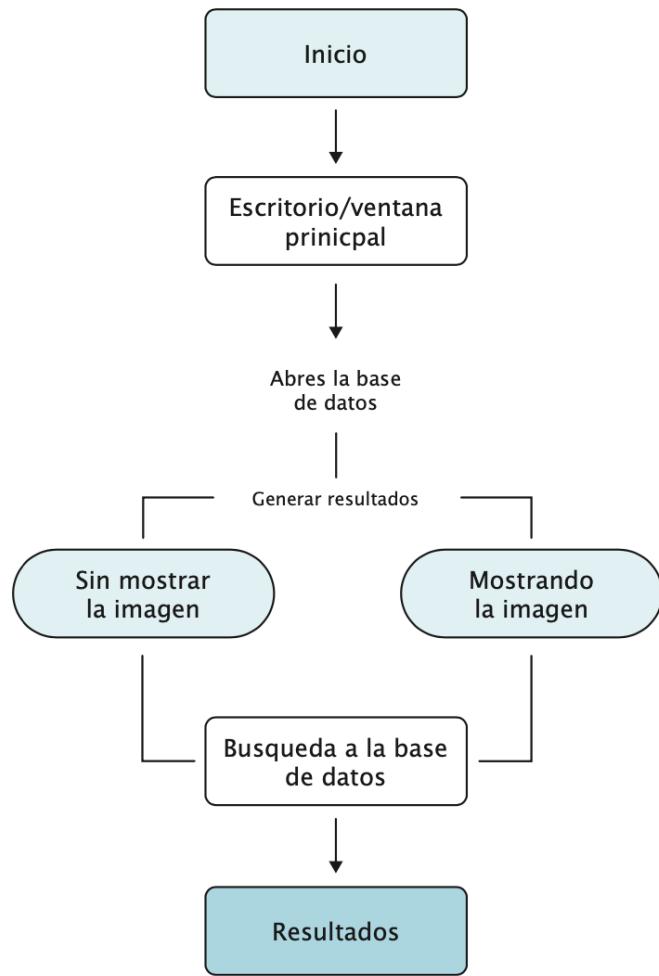


Figura 37: Flujo de interacción entre el usuario, la API generativa y el sistema CBIR

7.4.2. Bocetos

Los primeros bocetos se realizaron a mano con el objetivo de definir la estructura inicial de la interfaz, priorizando la disposición de los componentes principales: el área de entrada del prompt, el botón de generación, la galería de resultados y las opciones adicionales de filtrado o guardado. Estos bocetos permitieron una exploración rápida de ideas antes de pasar a herramientas digitales.

El primer boceto muestra la vista general del escritorio o ventana principal de la aplicación. En la parte superior se encuentra una barra de herramientas con botones y opciones de interacción. La zona central queda libre para cargar ventanas internas según las acciones del usuario, como la generación o visualización de imágenes.

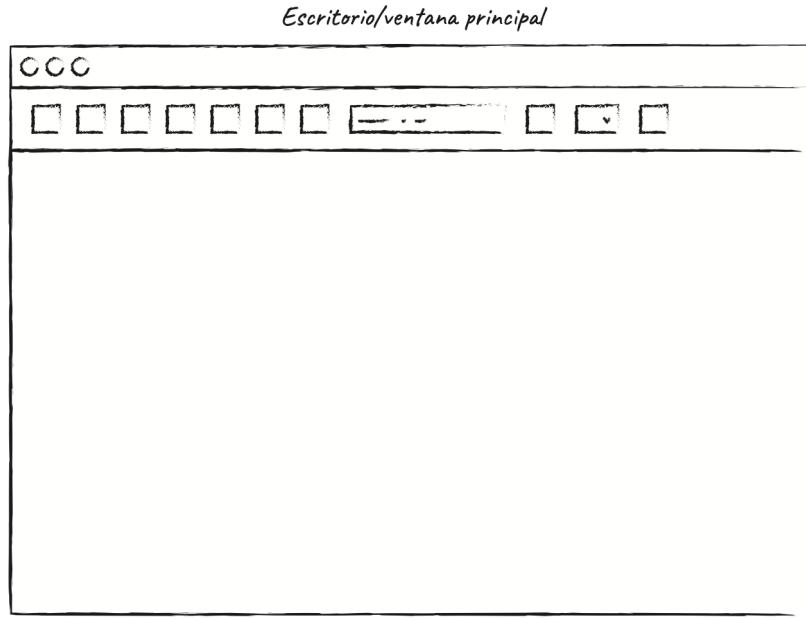


Figura 38: Boceto: propuesta inicial del layout general

El segundo boceto representa la ventana de generación de imagen. Esta aparece superpuesta sobre el escritorio y contiene una zona de entrada de texto donde el usuario puede escribir el prompt, junto con un botón de acción y una barra de carga que indica el estado del proceso de generación.

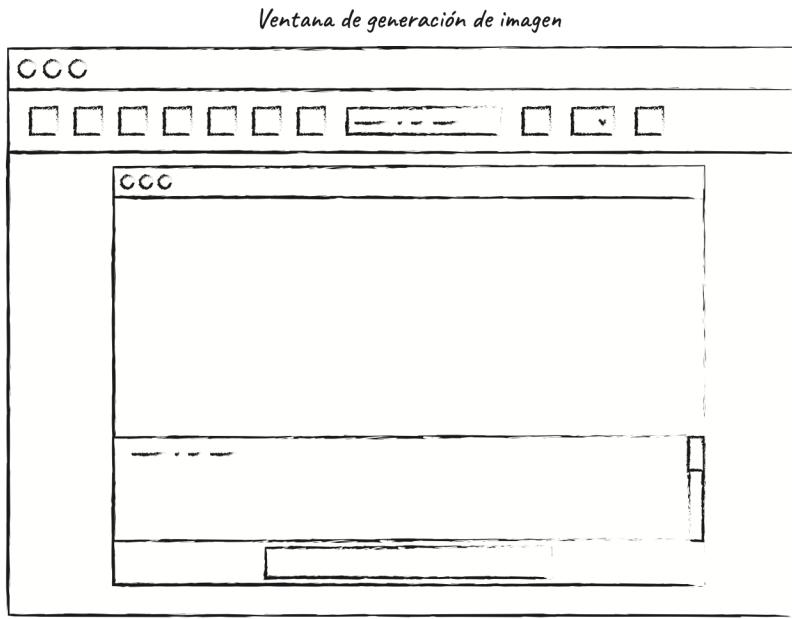


Figura 39: Boceto: navegación entre secciones

El tercer boceto corresponde a la vista de resultados generados. En la parte superior se visualiza una galería de imágenes miniatura, mientras que en la zona inferior se muestra una imagen ampliada seleccionada por el usuario. Este diseño facilita la exploración visual y la interacción posterior, como aplicar descriptores o realizar búsquedas visuales.

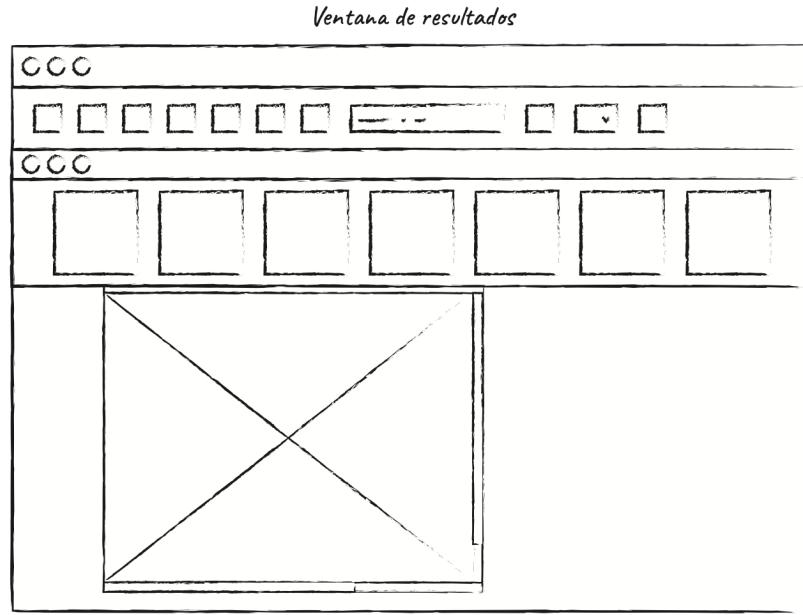


Figura 40: Boceto: detalle de interacción en la vista de resultados

7.4.3. Wireframes

A partir de los bocetos iniciales, se desarrollaron wireframes de baja fidelidad utilizando herramientas digitales. Estos permitieron refinar la experiencia de usuario, definir la jerarquía visual de cada componente y estructurar la navegación entre secciones clave del sistema.

La primera pantalla corresponde al escritorio o ventana principal del sistema. En ella se muestra la interfaz base del entorno JMR, con su barra de herramientas y el área central donde se mostrarán las distintas ventanas internas.

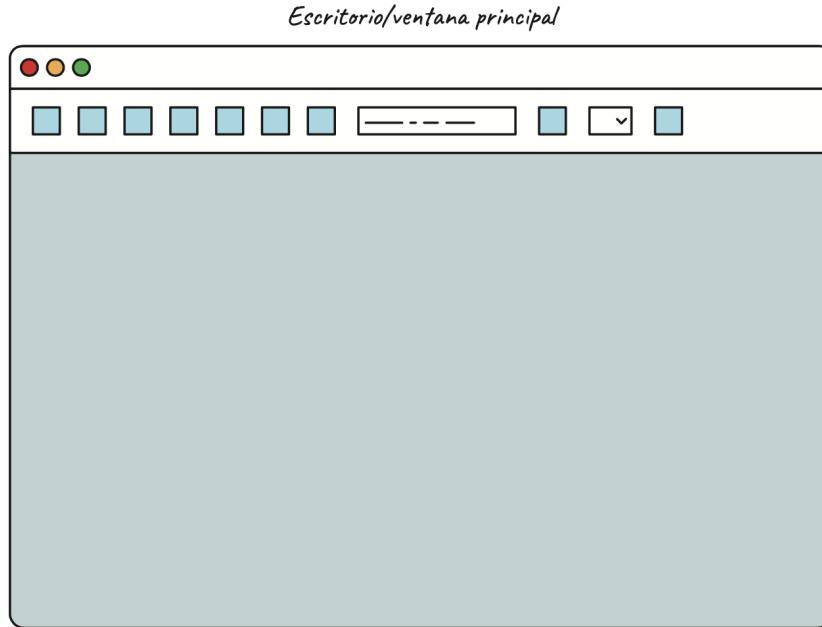


Figura 41: Wireframe: pantalla principal con área de generación

La segunda imagen representa la ventana de generación de imagen. Esta ventana se abre dentro del escritorio cuando el usuario decide introducir un prompt. Incluye un área de visualización de la imagen generada, un campo de entrada de texto y un botón de acción.



Figura 42: Wireframe: visualización detallada de una imagen generada

El tercer wireframe muestra la pantalla de resultados, donde se visualiza la imagen generada y las imágenes recuperadas mediante el sistema CBIR. En la parte superior se encuentra la galería de resultados similares y, debajo, una vista ampliada de la imagen seleccionada.

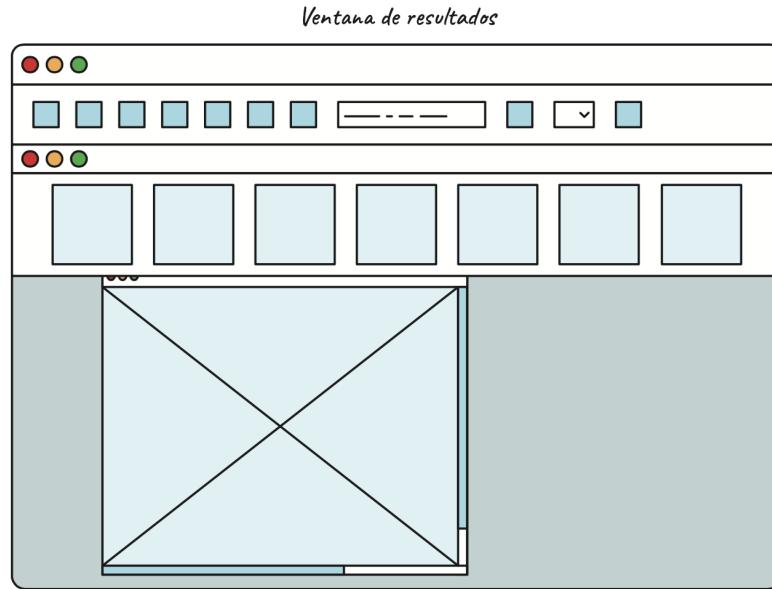


Figura 43: Wireframe: pantalla de resultados con opciones de filtrado

7.4.4. Prototipo en Figma

Como parte final del diseño de la interfaz, se construyó un prototipo funcional en Figma basado en los wireframes anteriores. Este prototipo permitió simular la navegación entre ventanas y validar la experiencia de usuario en un entorno gráfico realista.

La interacción está compuesta por distintas pantallas: la ventana principal de escritorio, la ventana emergente de generación, la vista con imagen generada, la galería de resultados y el histórico de prompts. También se definieron transiciones claras entre cada pantalla, incluyendo retroalimentación tras la generación o selección de imágenes.

Este flujo fue clave para observar posibles puntos de mejora en usabilidad y validar la navegación global del sistema antes de implementarlo.

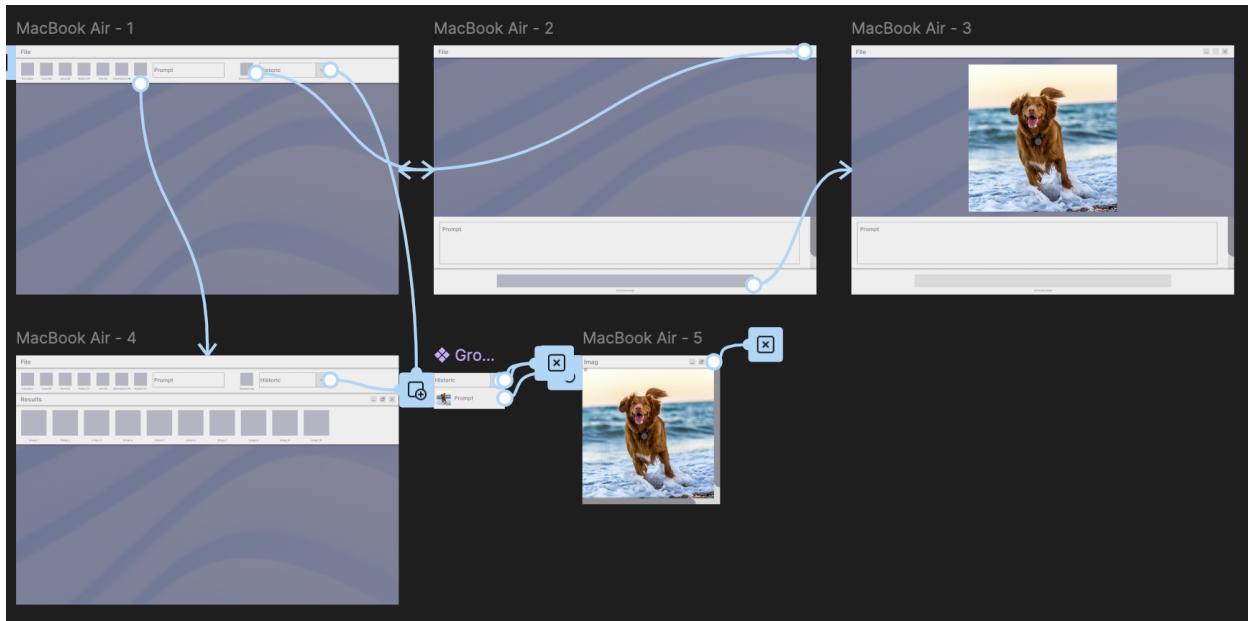


Figura 44: Vista general del flujo de navegación entre pantallas del prototipo en Figma

Las capturas detalladas de las pantallas y sus interacciones están disponibles en el Anexo A.

7.4.5. Usabilidad

El diseño de la interfaz se ha guiado por principios fundamentales de usabilidad con el objetivo de ofrecer una experiencia accesible, eficiente y satisfactoria para todo tipo de usuarios, independientemente de su nivel técnico. A partir de las historias de usuario definidas (HU1 y HU2), se identificaron tres necesidades clave: introducir descripciones textuales de forma clara, visualizar resultados generados de forma comprensible, y reutilizar esas imágenes para buscar contenido visual similar en la base de datos.

Con base en estos objetivos, se aplicaron los siguientes principios de diseño centrado en el usuario:

- **Interacción simple y directa:** La interfaz principal presenta solo los elementos esenciales para la tarea principal: un campo para introducir el prompt, un botón para iniciar la generación y un área de visualización de la imagen resultante. Esta simplicidad evita la sobrecarga cognitiva y facilita que el usuario comprenda rápidamente cómo usar el sistema sin necesidad de asistencia externa.
- **Retroalimentación inmediata y comprensible:** Tras introducir el prompt, el sistema proporciona indicadores visuales (como animaciones de carga) que confirman que la solicitud está siendo procesada. Al completarse la generación, la imagen se muestra de forma destacada, acompañada de una acción clara para continuar con el flujo (por ejemplo, usarla como consulta en el sistema CBIR). Esto mantiene al usuario informado en todo momento y refuerza su confianza en el sistema.

- **Consistencia visual y funcional:** Se respetaron convenciones de diseño ampliamente reconocidas: uso de etiquetas descriptivas, alineaciones verticales, botones claramente diferenciados y jerarquía visual basada en el tamaño y el color. Esto reduce el tiempo de aprendizaje y evita comportamientos inesperados.
- **Accesibilidad y legibilidad:** Se priorizó el uso de tipografía clara, buen contraste entre texto y fondo, y tamaños adecuados de los elementos interactivos para facilitar el uso tanto en pantallas grandes como en dispositivos de menor tamaño. Asimismo, se evitó el uso de términos técnicos complejos, apostando por un lenguaje neutro y comprensible.
- **Minimización de errores y puntos de fricción:** El flujo está diseñado para prevenir errores comunes, como el envío de prompts vacíos o duplicados. Además, en caso de fallo (por ejemplo, si no se genera una imagen), el sistema informa con mensajes explicativos que permiten al usuario comprender lo ocurrido y actuar en consecuencia.

En conjunto, estas decisiones de diseño han permitido construir una interfaz que no solo es funcional, sino también intuitiva y centrada en las necesidades reales del usuario. Esto resulta especialmente importante en un sistema como este, donde se combinan tecnologías avanzadas (IA generativa y recuperación de imágenes) con una interacción aparentemente simple basada en lenguaje natural.

8. Implementación

El desarrollo del sistema ha supuesto un proceso iterativo en el que se han combinado técnicas de inteligencia artificial generativa, diseño de arquitectura modular y principios de integración software. El objetivo principal ha sido construir una solución capaz de generar imágenes a partir de descripciones textuales, integrándola de forma fluida en la plataforma Java Multimedia Retrieval (JMR). Para lograrlo, se abordaron distintas fases de implementación: desde la creación de una API REST funcional y flexible en Python, hasta la extensión del cliente JMR en Java para aprovechar dicho sistema generativo.

El detalle completo de la implementación a nivel de código no se incluye en esta memoria, ya que el objetivo es presentar una visión general de la arquitectura, las decisiones técnicas y los retos abordados durante el desarrollo. Para consultar la implementación completa, incluyendo los scripts de entrenamiento, el servidor de inferencia, la integración con JMR y la interfaz gráfica, se han habilitado tres repositorios independientes en GitHub: TFG, TFG-JMR y TFG-UI, todos ellos accesibles bajo solicitud al usuario `carlotiii30`.

Los enlaces y descripciones detalladas de estos repositorios pueden consultarse en el Apéndice C.

8.1. Implementación de la API para la Generación de Imágenes

Uno de los objetivos principales del proyecto fue desarrollar un sistema que permitiera la generación de imágenes a partir de descripciones textuales, accesible desde la plataforma JMR. Para ello, se implementó una API REST utilizando el framework *FastAPI*, desplegada en local mediante Docker. Esta solución actúa como puente entre el sistema generativo y la interfaz de usuario, facilitando la separación de responsabilidades y permitiendo futuras ampliaciones sin comprometer la estabilidad del entorno principal.

8.1.1. Arquitectura de la API

La API está diseñada para ser modular, escalable y fácilmente integrable en otros sistemas. Su estructura se organiza en varios módulos independientes:

- **Módulo de generación de imágenes:** Encargado de cargar el modelo seleccionado y generar imágenes a partir de un prompt textual.
- **Módulo de gestión de modelos:** Permite la subida, eliminación y listado de modelos de generación disponibles en el sistema.
- **Interfaz RESTful:** Define los distintos endpoints disponibles para interactuar con la API, conforme al paradigma CRUD.

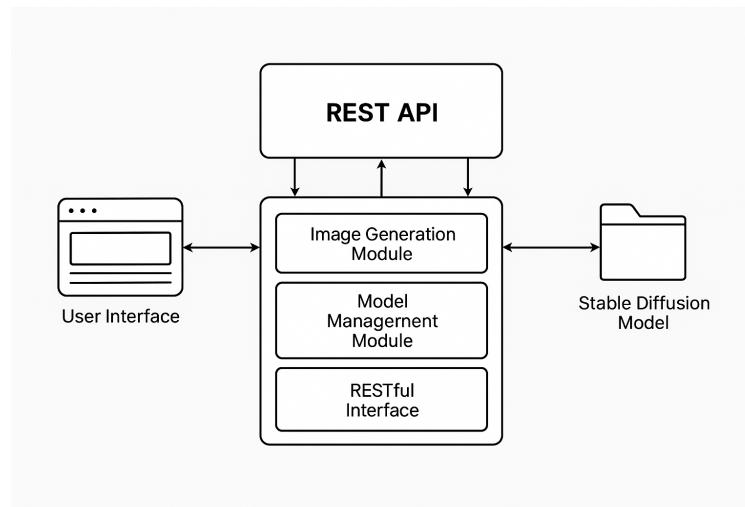


Figura 45: Diagrama de componentes de la API generativa

8.1.2. Endpoints implementados

Generación de imágenes

- **POST /images/generate/**

- **Descripción:** genera una imagen a partir de una descripción textual.
- **Entrada (JSON):**
 - `model_name` (opcional, por defecto `stable_diffusion`)
 - `prompt` (requerido)
 - `num_inference_steps` (opcional, por defecto 50)
 - `guidance_scale` (opcional, por defecto 7.5)
- **Salida:** JSON con la ruta de la imagen generada.

- **GET /images/download/{image_name}**

- **Descripción:** descarga una imagen generada.
- **Entrada:** nombre del archivo.
- **Salida:** archivo binario de imagen.

Gestión de modelos

- **POST /models/upload/**

- **Descripción:** permite subir un nuevo modelo en formato ZIP.
- **Entrada:** archivo ZIP.
- **Salida:** JSON de confirmación y ruta del modelo.

- **DELETE /models/delete/{model_name}**

- **Descripción:** elimina un modelo previamente subido.
- **Entrada:** nombre del modelo.
- **Salida:** mensaje de confirmación.

- **GET /models/list/**

- **Descripción:** devuelve un listado de todos los modelos disponibles.
- **Entrada:** ninguna.
- **Salida:** JSON con los nombres de los modelos.

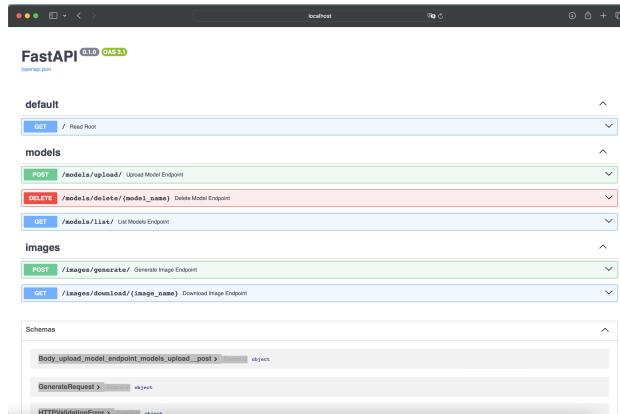


Figura 46: Endpoints de la API generativa

8.1.3. Flujo de interacción

El flujo de uso parte de una descripción textual introducida en la interfaz de JMR. Esta se envía a través de un endpoint POST a la API, que responde con una imagen generada. La API también permite cambiar o cargar nuevos modelos desde la interfaz gráfica de la plataforma, lo que facilita la gestión dinámica sin necesidad de reiniciar el sistema.

8.1.4. Entorno y despliegue

La API se ejecuta en contenedores Docker configurados para su despliegue local. Esta elección permitió realizar pruebas aisladas, reproducibles y sin afectar al sistema anfitrión. Gracias a esta arquitectura, la API puede integrarse fácilmente en entornos más complejos como Kubernetes, o adaptarse para ejecutarse en servidores dedicados con balanceo de carga.

8.1.5. Conclusión

La API desarrollada con FastAPI proporciona una interfaz robusta y extensible para la generación de imágenes a partir de texto. Su diseño modular, combinado con validaciones estructurales y la posibilidad de gestionar modelos dinámicamente, garantiza una base sólida sobre la que seguir evolucionando el sistema. Aunque aún quedan aspectos por reforzar en seguridad y escalabilidad, la solución actual cumple eficazmente su propósito en entornos de desarrollo y validación funcional.

8.2. Integración con la plataforma JMR (Java)

Una parte clave del proyecto ha sido extender la plataforma Java Multimedia Retrieval (JMR) para permitir la generación de imágenes a partir de descripciones textuales. Esta funcionalidad ha sido implementada respetando la arquitectura modular existente y proporcionando una experiencia de usuario coherente con el resto del sistema.

8.2.1. Arquitectura de integración

El flujo completo de integración entre la interfaz y el sistema generativo se basa en la arquitectura descrita previamente en la Figura 35. Según dicho esquema, la estructura se traduce en una serie de componentes interconectados que permiten gestionar el envío de prompts, la visualización de resultados y la reutilización de imágenes generadas.

A continuación se describen los principales elementos que componen esta arquitectura:

- **Clase MainWindow:** se encarga de inicializar los componentes principales de la interfaz. Gestiona el cambio entre API local y online mediante un sistema de menús interactivo, permitiendo introducir un token cuando se selecciona la API online.
- **Clase PromptWindow:** ventana interna dedicada a la generación de imágenes. Incluye un cuadro de texto para el prompt, una imagen por defecto y un botón para lanzar la generación. La imagen generada se actualiza en la misma ventana, y el prompt se guarda automáticamente.
- **Clase InternalWindow:** subclase personalizada de ventana interna que permite visualizar las imágenes generadas y contiene la lógica necesaria para gestionarlas (título, visualización, reuso, etc.).
- **Clase ImagePromptItem y ImagePromptListRenderer:** definen la estructura visual y funcional del historial. Cada entrada del historial representa un par imagen-prompt, y puede seleccionarse para su reutilización.
- **Clase ListInternalWindow:** muestra visualmente los resultados de una consulta en la base de datos a partir de una imagen generada, en el caso de realizar una búsqueda sin visualización explícita.

8.2.2. Flujo de generación de imagen

Cuando el usuario accede a la opción **Generar Imagen**, se abre una instancia de **PromptWindow**. Esta ventana contiene:

- Un cuadro de texto para introducir la descripción (prompt).
- Una imagen de marcador por defecto.
- Un botón que al pulsarse envía una petición HTTP a la API REST configurada (local u online) usando el prompt indicado.

La respuesta es una imagen en binario que se muestra en la misma ventana y se almacena localmente. Además, se añade una nueva entrada al historial mediante una instancia de **ImagePromptItem**, que queda registrada para consultas posteriores desde un menú desplegable.

8.2.3. Selección y gestión de la API

Desde el menú superior de la clase **MainWindow**, el usuario puede seleccionar si desea usar una API local o remota:

- Al seleccionar **Online API**, se abre un cuadro de diálogo solicitando un token. Si se proporciona correctamente, se guarda para las siguientes peticiones.
- Si se cancela o el token es inválido, el sistema vuelve automáticamente a **Local API**.

Este mecanismo asegura que la opción activa siempre sea válida, y se mantiene durante toda la sesión.

8.2.4. Consulta sin visualización

En el caso de realizar una consulta directa (sin mostrar la imagen), el prompt se toma del cuadro de texto principal y se lanza una generación en segundo plano. La imagen generada se usa como entrada para una búsqueda visual mediante el motor de recuperación de JMR. Los resultados se muestran en una instancia de **ListInternalWindow**, sin mostrar la imagen generada al usuario.

8.2.5. Historial reutilizable

El historial se gestiona mediante una lista desplegable que utiliza un renderizador personalizado para mostrar cada entrada de forma clara y compacta. El usuario puede:

- Visualizar entradas previas seleccionándolas desde el historial.
- Reabrir la imagen correspondiente en una nueva **InternalWindow**.
- Consultar la descripción asociada al pasar el cursor por encima.

Este historial se actualiza dinámicamente cada vez que se genera una nueva imagen.

8.2.6. Gestión de errores

Se han implementado cuadros de diálogo para mostrar mensajes claros ante situaciones como:

- Prompt vacío o inválido.
- Fallo de conexión con la API.
- Token incorrecto.
- Imagen generada no disponible en disco.
- Estructura inválida del modelo cargado.

Estas validaciones mejoran la robustez y evitan que el sistema se bloquee ante entradas inesperadas.

8.2.7. Comunicación con el sistema generativo: descriptores de imagen

La integración entre la interfaz de usuario y la API REST del sistema generativo se estructura en torno a una jerarquía de clases que encapsulan el proceso de generación de imágenes a partir de texto. Esta arquitectura, representada en la Figura 36, permite abstraer los detalles técnicos de la comunicación con el backend, garantizando una interfaz uniforme y extensible.

A continuación se describen las clases principales:

- **AbstractPromptImageDescriptor**: clase base que define el comportamiento común a todos los descriptores basados en texto. Contiene los campos necesarios para almacenar el prompt, el nombre del modelo empleado y la imagen generada (de tipo **BufferedImage**). Define además el método abstracto **generate()**, que debe ser implementado por las subclases para llevar a cabo el proceso de generación correspondiente.
- **PromptGeneratedImageDescriptor**: implementación para generación remota. El método **generate()** realiza una petición HTTP POST al endpoint `/images/generate/`, enviando los datos del prompt y del modelo. Como respuesta, recibe una ruta relativa de la imagen generada, que posteriormente se descarga mediante un GET a `/images/download/{filename}`.
- **PromptGeneratedImageDescriptorLocal**: implementación para generación local, en entornos donde la API está desplegada en localhost. Aunque reutiliza la lógica de generación remota, modifica la URL base y simplifica el flujo de validación al no depender de red.

La clase **MainWindow** se encarga de instanciar el descriptor adecuado en función del modo seleccionado por el usuario (API local u online), sin que el resto de la aplicación tenga que preocuparse por los detalles de la comunicación. Esta separación de responsabilidades permite que otros componentes, como los encargados de mostrar o guardar imágenes, interactúen con los descriptores de forma homogénea a través del método **getGeneratedImage()**, que garantiza que la imagen ha sido generada y devuelve un objeto **BufferedImage** listo para ser usado.

Gracias a esta arquitectura, el sistema es fácilmente extensible. Es posible incorporar nuevos tipos de descriptores, ya sea para otros modelos generativos o servicios externos, sin necesidad de modificar la lógica central de la aplicación. Además, al encapsular dentro de cada descriptor la gestión de errores y validaciones, se facilita el mantenimiento del código y se mejora la robustez del sistema.

8.3. Consideraciones de seguridad y rendimiento

Durante el desarrollo del sistema de generación de imágenes, se han contemplado múltiples aspectos relacionados con la seguridad del servicio, la gestión del entorno de ejecución, la optimización del rendimiento y la preparación para entornos de producción. Estas consideraciones resultan fundamentales para garantizar la integridad de los datos, la reproducibilidad de los experimentos y la escalabilidad futura del sistema en contextos reales.

8.3.1. Gestión del entorno con Poetry

Para asegurar un entorno de ejecución reproducible, coherente y fácilmente desplegable, se ha utilizado **Poetry** como gestor de dependencias y empaquetado. Esta herramienta ha permitido:

- Especificar con precisión las versiones de todas las librerías utilizadas, como **diffusers**, **transformers**, **torch**, **Pillow**, entre otras, evitando incompatibilidades entre ellas.
- Garantizar la replicabilidad del entorno en distintas máquinas mediante los archivos **pyproject.toml** y **poetry.lock**, facilitando el trabajo colaborativo o la migración a servidores externos.
- Aislar completamente el entorno de desarrollo de otras instalaciones globales de Python, reduciendo el riesgo de conflictos entre proyectos.
- Preparar el proyecto para su distribución como paquete Python, en caso de que se quisiera liberar o integrar como dependencia en otros sistemas.

8.3.2. Contenerización y portabilidad con Docker

Para facilitar el despliegue del sistema en distintos entornos (local, servidor o nube), se ha incorporado **Docker** como herramienta de contenerización. Esto ha permitido encapsular todo el entorno de ejecución en una imagen portátil y reproducible, con las siguientes ventajas:

- Elimina dependencias externas del sistema operativo anfitrión, asegurando que todas las versiones de librerías y herramientas se mantengan consistentes.
- Facilita el despliegue del sistema en servidores remotos, entornos cloud o plataformas de orquestación como Kubernetes.
- Permite realizar pruebas locales sin afectar al entorno del desarrollador.
- Mejora la seguridad al encapsular los procesos y evitar accesos arbitrarios al sistema anfitrión.

El archivo `Dockerfile` define el entorno con precisión, incluyendo la instalación de `poetry`, las dependencias del proyecto y la ejecución del servidor con `unicorn`. Para facilitar la ejecución, se han definido scripts automatizados que permiten iniciar el contenedor con un solo comando.

Esta estrategia prepara el sistema para un despliegue más robusto y escalable en futuras fases del proyecto.

8.3.3. Automatización del desarrollo con GitHub Workflows

Para facilitar el mantenimiento del proyecto, asegurar su correcto funcionamiento ante cambios y preparar su evolución hacia entornos más robustos, se ha integrado un sistema de automatización basado en **GitHub Workflows**. Esta funcionalidad forma parte de GitHub Actions, un conjunto de herramientas para integrar CI/CD (Integración y Entrega Continua) directamente desde el repositorio.

Se han definido distintos workflows automatizados para el repositorio principal del sistema generativo (TFG), entre los que destacan:

- **Ejecutar tests automáticamente en cada push o pull request:** se lanza una validación con `pytest` sobre los módulos internos del servidor FastAPI, verificando la correcta respuesta ante entradas válidas, malformadas y modelos incompletos.
- **Revisión de estilo y formato:** se integró un paso de validación con `black` y `flake8` para asegurar la consistencia en el estilo de código y detectar errores sintácticos comunes.
- **Instalación del entorno con Poetry:** el workflow incluye pasos para instalar las dependencias mediante `Poetry`, validando que el proyecto puede reproducirse correctamente en cualquier entorno limpio, tal como ocurriría en un servidor de producción.

Estos procesos contribuyen a mantener la calidad del código, detectar errores de forma temprana y garantizar la estabilidad del sistema ante futuras modificaciones. Además, sientan las bases para una futura entrega continua y despliegue automático en entornos reales.

8.3.4. Seguridad en la API REST

Dado que el sistema de generación se expone a través de una interfaz web (API REST), es fundamental considerar posibles vectores de ataque relacionados con el envío de solicitudes maliciosas, el uso no autorizado o la exposición de recursos sensibles. Aunque esta versión del sistema no incluye mecanismos avanzados de autenticación o autorización, se han implementado algunas medidas preliminares de validación estructural y se identifican los siguientes aspectos clave para futuras iteraciones:

- **Validación de entradas:** actualmente, FastAPI permite tipado estático de parámetros, pero no se han definido aún restricciones estrictas sobre la longitud o estructura de los textos recibidos. Esto puede derivar en uso abusivo o errores inesperados ante entradas malformadas. Se recomienda implementar validaciones adicionales a nivel de contenido y lógica.

- **Validación de estructura de modelos:** se ha incorporado un sistema de validación estructural para los modelos subidos por el usuario. Esta validación, implementada mediante la función, comprueba que la carpeta del modelo contenga todos los subdirectorios y archivos esperados —como `config.json`, `model.safetensors`, o `tokenizer_config.json`—. Si falta alguno de estos elementos, se lanza una excepción `HTTPException` con un código de error 400, evitando así el uso de modelos incompletos o manipulados que puedan comprometer el funcionamiento del sistema.
- **Control del acceso a archivos:** actualmente no se ha desarrollado un sistema de permisos que restrinja el acceso directo a imágenes generadas o modelos almacenados. El sistema confía en una organización interna de rutas, lo cual no es suficiente para prevenir accesos arbitrarios si la API se expusiera públicamente. Se recomienda proteger los endpoints y establecer rutas temporales con tokens de acceso.
- **Límites de uso:** el servicio no impone cuotas por usuario, IP ni número de peticiones. En un entorno expuesto, esto puede derivar en ataques de denegación de servicio (DoS) o consumo excesivo de recursos. Se recomienda implementar mecanismos como rate limiting, autenticación básica o tokens temporales.
- **Manejo de errores:** aunque FastAPI ofrece gestión automática de errores comunes, no se ha personalizado la respuesta ante excepciones críticas. Actualmente, algunos errores podrían devolver trazas del servidor, lo que podría exponer rutas internas o detalles sensibles del sistema. Se sugiere capturar explícitamente excepciones clave y devolver mensajes controlados y neutros.

En resumen, aunque el sistema aún no está preparado para un entorno productivo expuesto, ya se han incorporado medidas preventivas como la validación estructural de modelos, que garantiza la integridad mínima antes de permitir su ejecución. Estas medidas constituyen una base sobre la que construir mecanismos más robustos de autenticación, protección de recursos y resiliencia frente a ataques externos.

Todas estas cuestiones han sido tenidas en cuenta de cara a un posible despliegue en producción. Se recomienda como trabajo futuro incorporar autenticación de usuarios, límites de uso, logs de auditoría y un tratamiento más robusto de la validación de parámetros para proteger el sistema frente a usos indebidos.

8.3.5. Optimización del rendimiento

Para maximizar la eficiencia del sistema, especialmente en entornos con recursos limitados, se han llevado a cabo diversas estrategias:

- **Carga única del modelo:** el modelo generativo se inicializa una única vez al arrancar el servidor, evitando recargas innecesarias en cada solicitud.
- **Uso eficiente del espacio en disco:** las imágenes generadas se almacenan temporalmente en disco con nombres únicos, y se reutilizan si ya existen para un mismo prompt y configuración.
- **Configuración adaptable:** aunque se ha utilizado `float32` durante el desarrollo, el sistema está preparado para funcionar con `float16` o `bfloat16` en entornos con soporte, permitiendo reducir a la mitad el uso de memoria sin pérdida significativa de calidad.
- **Pipeline modular:** la arquitectura permite cambiar fácilmente el modelo base, el VAE o el codificador de texto, permitiendo futuras mejoras sin afectar al resto del sistema.

8.3.6. Preparación para producción y despliegue escalable

Aunque el presente sistema ha sido desarrollado en un entorno controlado, se han identificado y documentado las acciones necesarias para su migración a entornos reales de producción. Entre las medidas consideradas:

- **Autenticación:** implementación de tokens JWT para identificar usuarios y restringir el acceso a usuarios registrados.
- **Rate limiting:** limitación de solicitudes por IP o por usuario en función de cuotas diarias o mensuales.

- **Contenerización:** uso de Docker para encapsular el entorno, garantizando portabilidad y facilitando el despliegue en VPS, Kubernetes o servicios cloud.
- **Proxy inverso con Caddy o NGINX:** para mejorar la seguridad en las conexiones HTTP y permitir balanceo de carga entre instancias.
- **Logging y monitorización:** integración de herramientas de seguimiento de logs y métricas del sistema para supervisar el uso real del servicio.

8.3.7. Reflexión final

Estas medidas y estrategias no solo han permitido desarrollar un sistema funcional en el presente, sino que establecen las bases para una posible evolución futura hacia un servicio robusto y escalable. La combinación de buenas prácticas en gestión de entornos, principios básicos de seguridad y una arquitectura eficiente posiciona este proyecto como un prototipo avanzado, listo para crecer hacia escenarios reales de uso y despliegue institucional o comercial.

8.4. Pruebas y validación

Para validar el correcto funcionamiento del sistema desarrollado, se han realizado pruebas tanto sobre la API REST de generación como sobre la integración completa dentro de la plataforma Java JMR. Estas pruebas tenían como objetivo verificar la funcionalidad de los endpoints, la integridad del flujo de generación, la correcta gestión del historial y la interacción con la base de datos visual.

8.4.1. Pruebas sobre la API REST

La API desarrollada con FastAPI cuenta con una interfaz interactiva (Swagger UI), accesible en `/docs`, desde la cual es posible probar directamente todos los endpoints definidos. Se realizaron pruebas manuales de los principales puntos del sistema:

- **GET /models/list/**: verifica que la API devuelve correctamente los modelos disponibles. En la prueba, se recibió el modelo por defecto "stable".
- **POST /images/generate/**: permite generar una imagen a partir de un prompt textual. En la prueba realizada se utilizó el prompt "*a sunset on the beach*" con parámetros por defecto. La respuesta fue satisfactoria, devolviendo el nombre del archivo generado.
- **GET /images/download/{image_name}**: descarga la imagen generada, lo que permitió comprobar la existencia y legibilidad del archivo.
- **Manejo de errores**: se realizaron pruebas con modelos inexistentes, prompts vacíos o estructuras incorrectas. La API devolvió códigos HTTP coherentes (400 o 422) y mensajes de error controlados, como se especifica en la validación estructural.

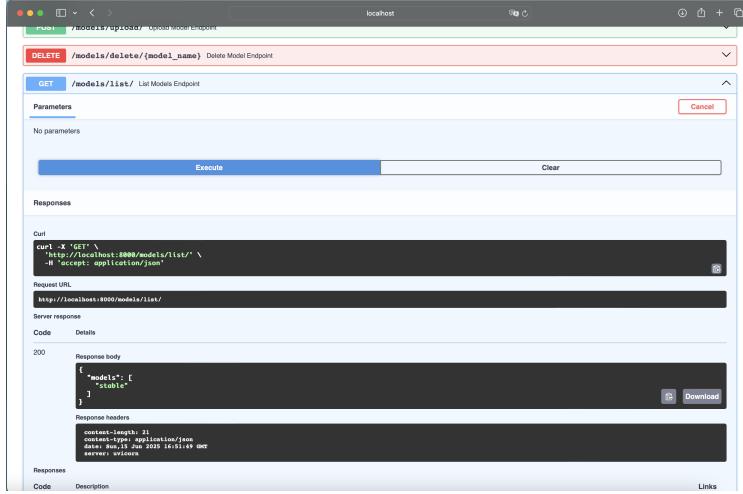


Figura 47: Prueba del endpoint GET /models/list/ desde Swagger

Además de estas pruebas manuales, se desarrollaron tests automáticos utilizando `pytest` para las funciones internas del servidor, incluyendo validaciones sobre el manejo de prompts, la existencia del modelo solicitado y la correcta generación y localización de las imágenes. Estos tests permitieron asegurar la estabilidad del sistema ante cambios futuros y facilitar su mantenimiento.

8.4.2. Pruebas en la aplicación JMR

La validación funcional dentro de la interfaz gráfica de JMR se llevó a cabo comprobando las siguientes funcionalidades:

- **Selección de API local/online:** se comprobó que el sistema gestionaba correctamente el cambio entre API, solicitando el token cuando era necesario y manteniendo el estado activo durante la sesión.
- **Generación desde interfaz:** se probó la creación de una imagen a partir de un prompt, observando la aparición automática en la ventana interna junto con la actualización del historial.
- **Consulta sin visualización:** se utilizó el cuadro de texto superior para lanzar una búsqueda desde descripción sin visualizar la imagen. El sistema generó correctamente la imagen en segundo plano y devolvió los resultados similares en la base de datos.
- **Historial de imágenes:** se validó que las entradas quedaban correctamente almacenadas, eran accesibles desde el desplegable, y al seleccionarlas se abría la ventana correspondiente.
- **Reutilización de imágenes generadas:** se reutilizaron imágenes del historial para lanzar nuevas consultas, sin volver a generar los archivos en disco.
- **Errores controlados:** se forzaron errores como prompts vacíos, imágenes eliminadas o modelos mal estructurados, observando que el sistema respondía con mensajes claros sin bloquear la aplicación.

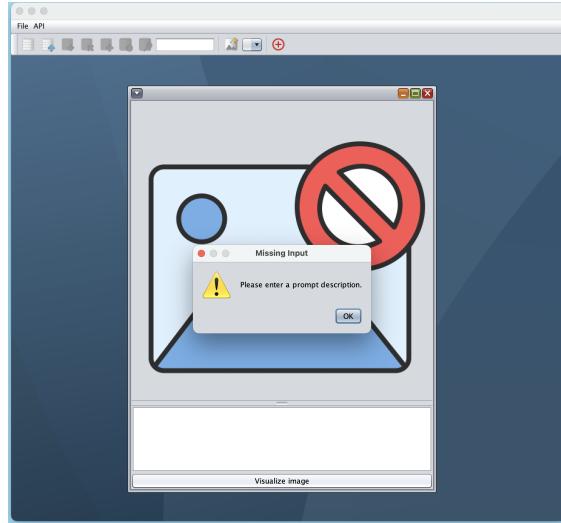


Figura 48: Mensaje de error al intentar generar un prompt vacío

8.4.3. Validación de consistencia e integridad

Para confirmar que el sistema funcionaba de forma coherente y sin redundancias, se realizaron pruebas cruzadas:

- Se compararon imágenes generadas con prompts similares para validar la coherencia del modelo.
- Se comprobó que no se duplicaban archivos si se reutilizaba el mismo prompt.
- Se examinó el rendimiento en generación en entorno local y se registró el tiempo medio de respuesta.

8.4.4. Conclusión

Las pruebas realizadas han demostrado que el sistema es funcional, coherente y robusto en entornos controlados. La API responde adecuadamente ante entradas válidas y malformadas, mientras que la integración con JMR mantiene una experiencia fluida para el usuario final. Aunque aún pueden incorporarse mecanismos adicionales de seguridad y autenticación, el sistema se encuentra en un estado válido para su uso en escenarios de validación funcional y pruebas de concepto.

9. Manual de usuario

9.1. Introducción

Se ha integrado en la aplicación JMR nuevas funcionalidades que permiten generar imágenes a partir de descripciones textuales de forma sencilla y eficiente. Con estas funcionalidades, puedes:

- Elegir la fuente de generación de imágenes, ya sea un modelo local o una API online.
- Generar imágenes directamente desde descripciones en lenguaje natural.
- Consultar en la base de datos imágenes similares a las generadas.
- Consultar un historial de imágenes generadas previamente.
- Consultar en la base de datos imágenes similares a un prompt.

Este manual proporciona una guía paso a paso para utilizar esta funcionalidad, e incluye ejemplos prácticos y una descripción detallada de cada acción disponible en la interfaz.

9.2. Inicio de la aplicación

1. Abre la aplicación JMR haciendo clic en el ícono correspondiente en tu escritorio o desde el menú de inicio de tu sistema operativo.
2. Una vez abierta, observarás la interfaz principal de JMR, donde se encuentran las dos nuevas funcionalidades de generación de imágenes.

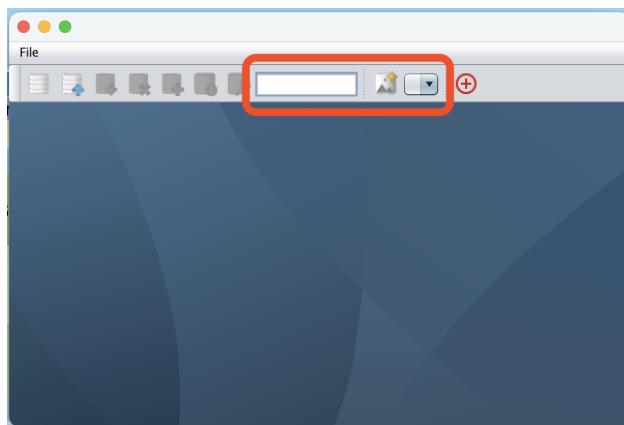


Figura 49: Inicio de la aplicación con las dos nuevas funcionalidades

9.3. Descripción de la interfaz

Las nuevas funcionalidades de generación de imágenes están integradas en la barra superior y presentan los siguientes elementos:

1. **Elección de API:** Un menú desplegable que permite seleccionar entre la API local o una API online para la generación de imágenes.
2. **Cuadro de texto:** Espacio donde puedes escribir la descripción textual de la imagen que deseas buscar en la base de datos.
3. **Generar imagen:** Abre una ventana con un área de texto en la que puedes describir la imagen a generar. Dentro de esta hay un botón para generar y visualizar la imagen descrita.
4. **Lista desplegable:** Lista desplegable que muestra un historial de las imágenes generadas junto a sus descripciones. Permite seleccionar una entrada anterior para visualizarla nuevamente.

9.4. Selección de API para generación de imágenes

El sistema permite elegir entre dos modos de conexión para la generación de imágenes: utilizando un modelo alojado localmente en el dispositivo, o bien accediendo a una API online como la de Hugging Face. Esta flexibilidad permite al usuario adaptar el sistema a diferentes contextos de uso.

Modo de selección

En la barra superior de la aplicación se encuentra un menú llamado **API**. Al hacer clic sobre él, se despliega un submenú con las siguientes opciones:

- **Local API:** utiliza el modelo alojado en el sistema local (por defecto).
- **Online API:** permite realizar peticiones a un servicio externo. Es necesario introducir un token de autenticación para acceder a esta opción.

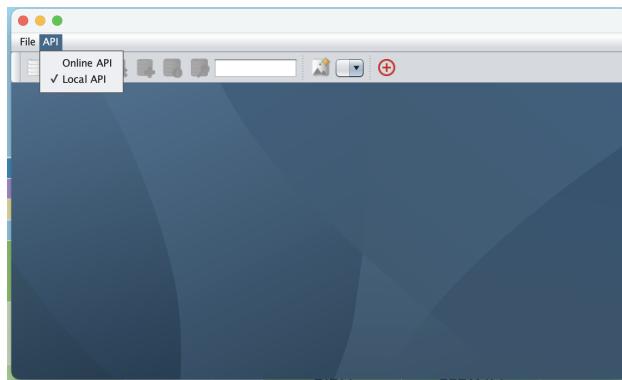


Figura 50: Menú desplegable para seleccionar entre API local u online

Uso de la Online API

Si se selecciona la opción **Online API**, aparecerá un cuadro de diálogo solicitando al usuario que introduzca su **Hugging Face API Token**.

- Si se introduce un token válido, el sistema comenzará a utilizar la API online para generar las imágenes.
- Si se cancela el proceso o se introduce un token vacío, el sistema volverá automáticamente a utilizar la opción **Local API**.

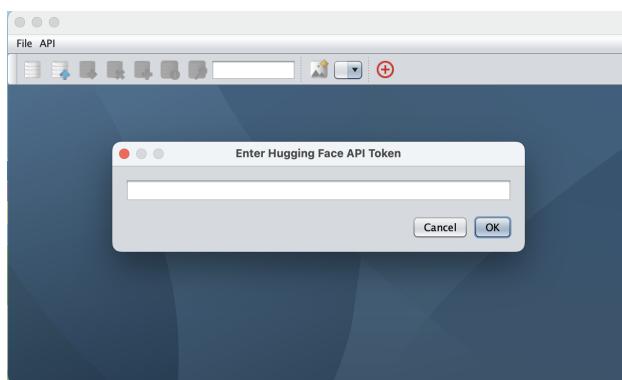


Figura 51: Ventana emergente para introducir el token de Hugging Face

Consideraciones adicionales

- La elección de API se conserva durante la sesión activa de la aplicación.
- Se recomienda utilizar la API online solo en entornos con conexión estable y disponer de un token previamente generado desde la plataforma de Hugging Face.

9.5. Cómo generar una imagen

Para generar una nueva imagen a partir de una descripción textual, sigue los siguientes pasos:

Paso 1: Acceder al generador

1. Haz clic en el botón *Generar Imagen*, situado en la barra superior de la aplicación.

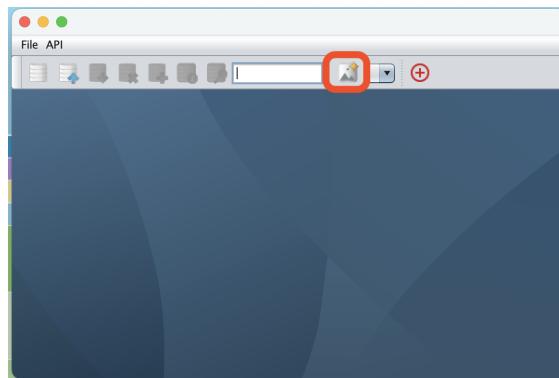


Figura 52: Botón para generar una imagen

2. Se abrirá una ventana interna que contiene:

- Un cuadro de texto para introducir la descripción.
- Una imagen por defecto como marcador de posición.
- Un botón con la etiqueta *Visualizar Imagen*.

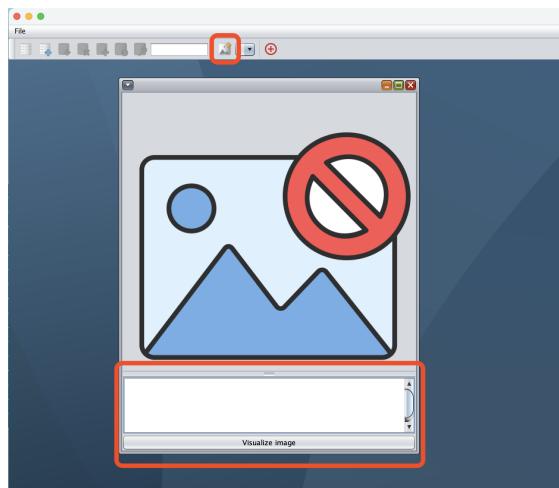


Figura 53: Ventana interna con área para escribir y botón de visualizar

Paso 2: Escribir la descripción

1. En el cuadro de texto, introduce una descripción lo más detallada posible de la imagen que deseas generar.
2. Ejemplo: “*A dog running on the beach with its friend.*”

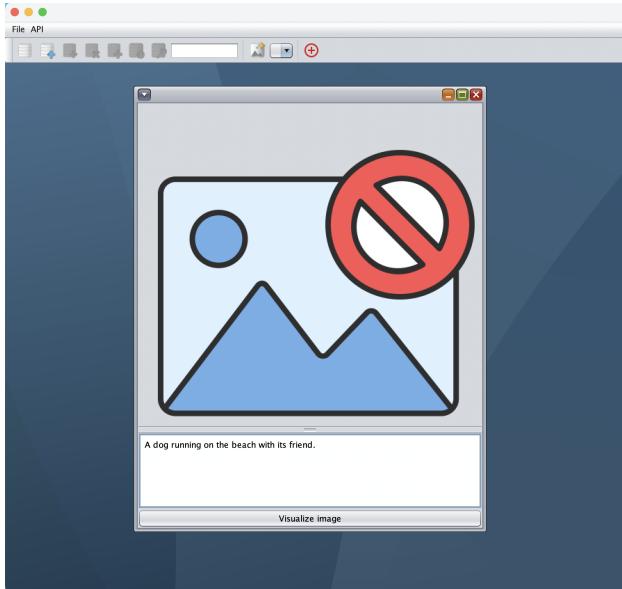


Figura 54: Ventana interna con área escrita con una descripción de ejemplo

Paso 3: Generar y visualizar la imagen

1. Haz clic en el botón *Visualizar Imagen*.
2. La imagen generada sustituirá a la imagen por defecto en la misma ventana.
3. Además, la imagen y su descripción se guardarán automáticamente en el historial de generación.

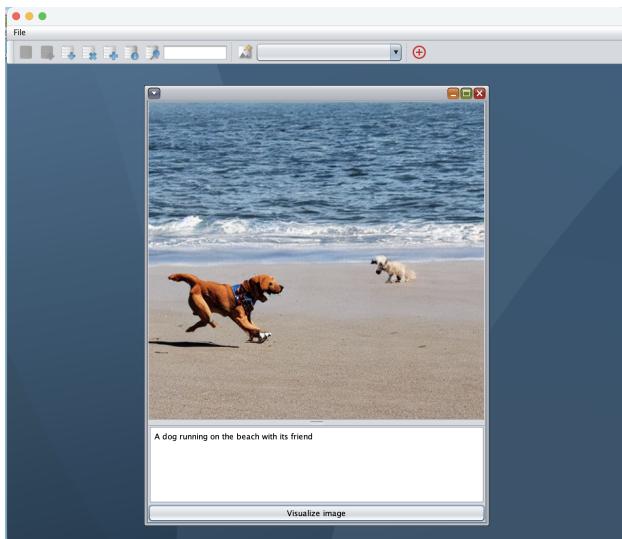


Figura 55: Ejemplo de visualización

9.6. Cómo generar una consulta sin mostrar la imagen

Esta funcionalidad permite lanzar una consulta basada en una descripción textual sin necesidad de generar ni visualizar la imagen en pantalla. En su lugar, se realiza una búsqueda en la base de datos para recuperar resultados visualmente similares.

Paso 1: Escribir la descripción

1. Introduce el texto descriptivo en el cuadro situado en la barra de herramientas superior.
2. Ejemplo: “*A lot of colorful sprinkles.*”

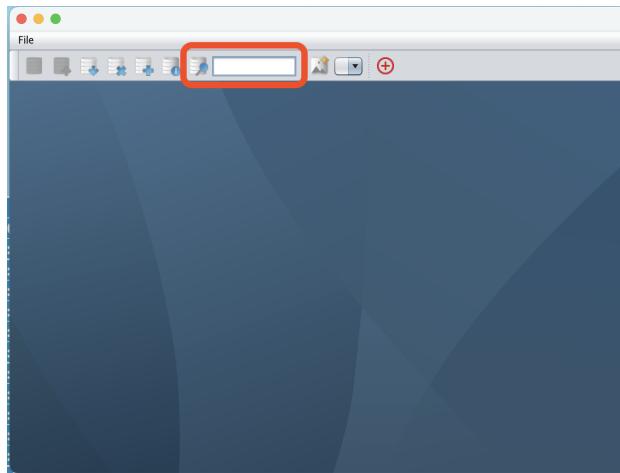


Figura 56: Escribir la descripción en la barra superior

Paso 2: Ejecutar la consulta

1. Haz clic en el botón de búsqueda situado junto al cuadro de texto.
2. El sistema generará la imagen en segundo plano y lanzará una búsqueda automática en la base de datos utilizando la imagen como consulta.

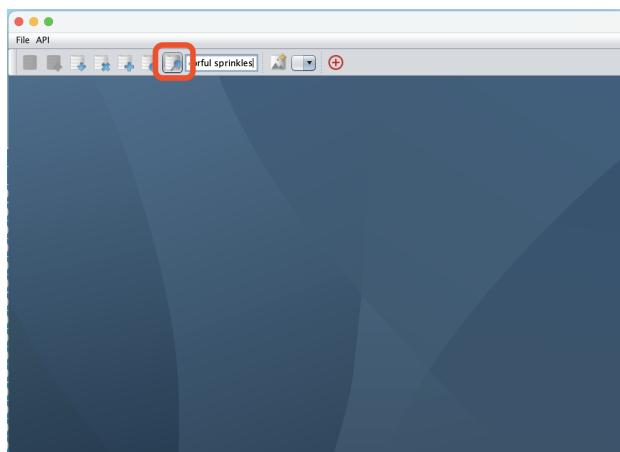


Figura 57: Botón de búsqueda para ejecutar la consulta

Paso 3: Visualización de resultados

1. Se abrirá una nueva ventana con los resultados visuales recuperados desde la base de datos, similares a la imagen generada a partir de tu descripción.

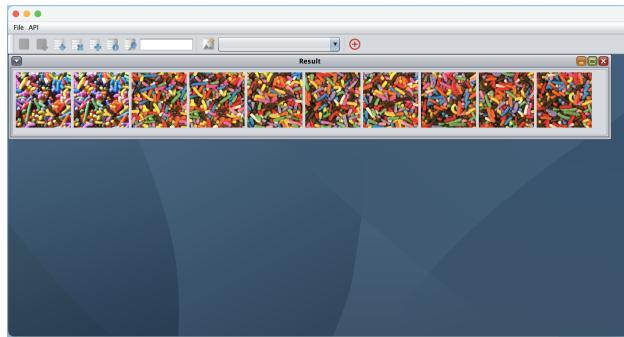


Figura 58: Resultados visuales recuperados de la base de datos

9.7. Reutilizar imágenes generadas

Todas las imágenes generadas se almacenan automáticamente en el historial, junto con la descripción (prompt) que se utilizó para crearlas. Este historial se encuentra accesible mediante una lista desplegable en la parte superior de la interfaz.

Para reutilizar una imagen generada anteriormente:

1. Despliega la lista situada en la parte superior.
2. Selecciona una de las entradas del historial (aparecen truncadas si son largas, pero se muestran completas en su tooltip).
3. La imagen correspondiente se abrirá en una nueva ventana interna, con el texto de la descripción como título.

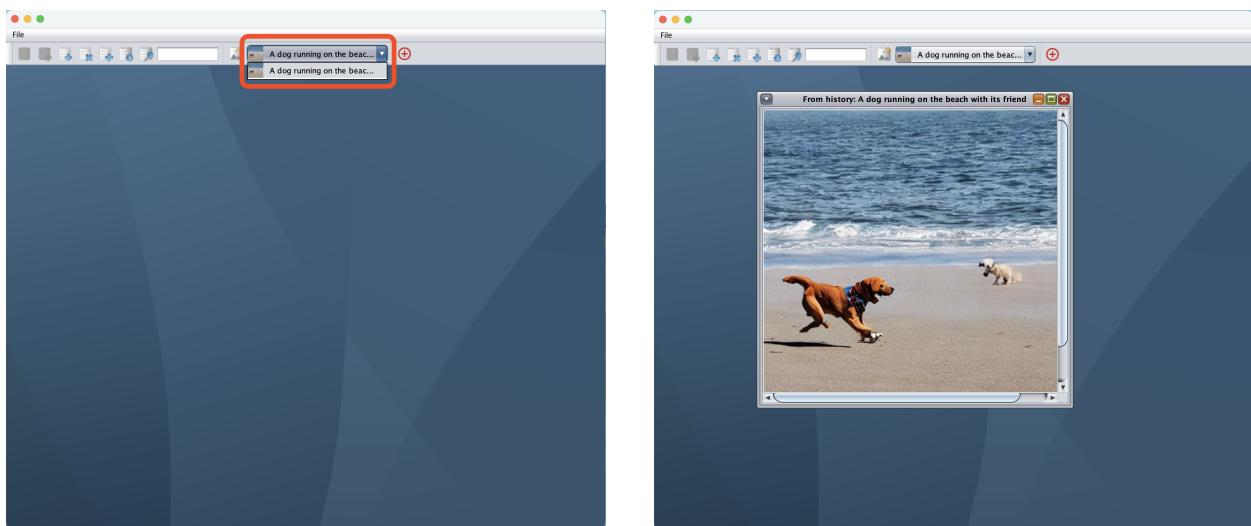


Figura 59: A la izquierda, vista del historial con una entrada seleccionada. A la derecha, la imagen generada que se abre al seleccionar esa entrada.

9.8. Errores y soluciones comunes

Durante el uso del sistema, pueden producirse ciertos errores que impidan temporalmente la generación de imágenes o la interacción con algunos elementos de la interfaz. A continuación se describen los más frecuentes y las acciones recomendadas para resolverlos:

■ Error al generar la imagen

- **Mensaje:** “No se pudo generar la imagen. Verifique su descripción.”
- **Causa:** El prompt puede estar vacío o ser demasiado ambiguo.
- **Solución:** Introduce una descripción más detallada y coherente. Por ejemplo, en lugar de escribir “*un animal*”, puedes escribir “*a cat with green eyes sitting on a sofa*”.

■ Error al cargar el modelo

- **Mensaje:** “El modelo seleccionado no es compatible.”
- **Causa:** Se ha cargado un archivo ZIP con estructura incorrecta o faltan archivos esenciales del modelo.
- **Solución:** Asegúrate de que el ZIP contenga los archivos requeridos como `config.json`, `model.safetensors`, `tokenizer_config.json`, etc. Utiliza modelos que hayan sido exportados siguiendo el formato esperado por la API.

■ Error de conexión al utilizar la API online

- **Mensaje:** “Error de autenticación con la API. Token inválido o expirado.”
- **Causa:** El token de Hugging Face es incorrecto o ha caducado.
- **Solución:** Verifica que has introducido el token correctamente. Puedes generar un nuevo token accediendo a tu cuenta en [<https://huggingface.co/settings/tokens>].

■ Error de red al generar con API online

- **Mensaje:** “No se pudo contactar con el servidor de generación de imágenes.”
- **Causa:** Falta de conexión a Internet o fallo del servidor externo.
- **Solución:** Comprueba tu conexión a Internet. En caso de persistir el error, cambia temporalmente a la opción Local API desde el menú superior.

■ Error al abrir una imagen desde el historial

- **Mensaje:** “La imagen seleccionada no se encuentra disponible.”
- **Causa:** El archivo correspondiente ha sido eliminado o movido manualmente.
- **Solución:** Evita modificar directamente la carpeta donde se almacenan las imágenes generadas. Si necesitas liberar espacio, utiliza la función de limpieza integrada en versiones futuras.

Para cualquier otro error no contemplado aquí, se recomienda reiniciar la aplicación y consultar la documentación técnica o contactar con el equipo de soporte.

10. Conclusiones

Este proyecto ha demostrado la viabilidad de integrar un sistema generativo de imágenes a partir de descripciones textuales dentro de una plataforma multimedia existente. A lo largo del desarrollo, se ha abordado con éxito la construcción de un servicio basado en modelos de difusión preentrenados, encapsulado en una API REST eficiente y gestionado mediante herramientas modernas como FastAPI, Poetry y Docker. Esta integración ha permitido establecer un puente entre el entorno Python de generación y la interfaz Java del sistema JMR, facilitando una experiencia fluida para el usuario final.

Uno de los aspectos más enriquecedores del proyecto ha sido, sin duda, la fase de experimentación con distintas arquitecturas generativas. Desde los primeros intentos con GANs y cGANs hasta la posterior exploración de AttnGAN y, finalmente, el uso de modelos basados en difusión como Stable Diffusion, cada etapa ha aportado un aprendizaje profundo sobre los retos técnicos, las limitaciones de cada enfoque y las posibilidades que ofrecen. Este proceso experimental ha sido esencial no solo para llegar a una solución funcional, sino también para consolidar una comprensión crítica del funcionamiento interno de los modelos generativos y sus implicaciones prácticas en la generación de contenido visual.

El sistema desarrollado ha cumplido satisfactoriamente con los objetivos planteados, permitiendo generar imágenes coherentes a partir de descripciones en lenguaje natural. Además, ha mostrado un comportamiento robusto durante las pruebas, tanto en términos de calidad de las imágenes como en su integración con la plataforma cliente. La validación visual de los resultados, junto con la implementación de un flujo controlado de entrada, generación y visualización, refuerzan la utilidad práctica del sistema como herramienta de consulta visual avanzada.

Durante el desarrollo, también se han aplicado buenas prácticas de diseño, como la separación de responsabilidades entre módulos, la validación estructural de los modelos subidos y el uso de entornos reproducibles. Estas decisiones han contribuido a dotar al sistema de una arquitectura modular y fácilmente extensible, preparada para futuras iteraciones o despliegues en producción.

10.1. Líneas futuras de trabajo

Aunque el proyecto alcanza un grado alto de completitud, existen múltiples caminos para su evolución. En primer lugar, se plantea la optimización del rendimiento mediante técnicas de compresión o el uso de modelos más ligeros, con el fin de facilitar su ejecución en dispositivos con recursos limitados. Asimismo, resulta clave incorporar mecanismos de seguridad adicionales, como límites de uso, autenticación de usuarios y validación semántica de los prompts, especialmente en escenarios donde el servicio pueda ser expuesto públicamente.

Otra línea de trabajo prometedora es el entrenamiento personalizado de los modelos con conjuntos de datos específicos, lo que permitiría adaptar la generación de imágenes a dominios concretos, como patrimonio cultural, medicina o diseño de producto. Además, la posibilidad de integrar funciones de edición interactiva o regeneración parcial de imágenes abriría nuevas puertas a la creatividad y el control por parte del usuario.

Finalmente, avanzar hacia la evaluación automática de resultados mediante métricas objetivas y explorar nuevos entornos de despliegue, como arquitecturas serverless o plataformas móviles, supondría un paso importante hacia la consolidación del sistema como herramienta accesible, escalable y útil en contextos reales.

Parte III

Anexos

A. Capturas del prototipo Figma

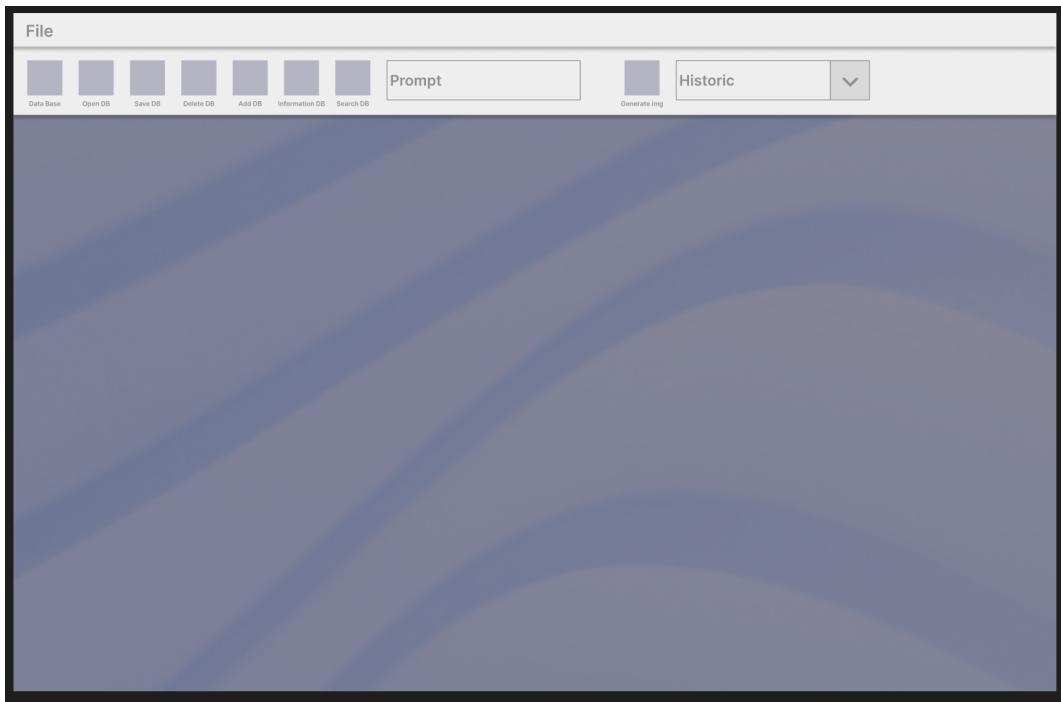


Figura 60: Pantalla principal del sistema

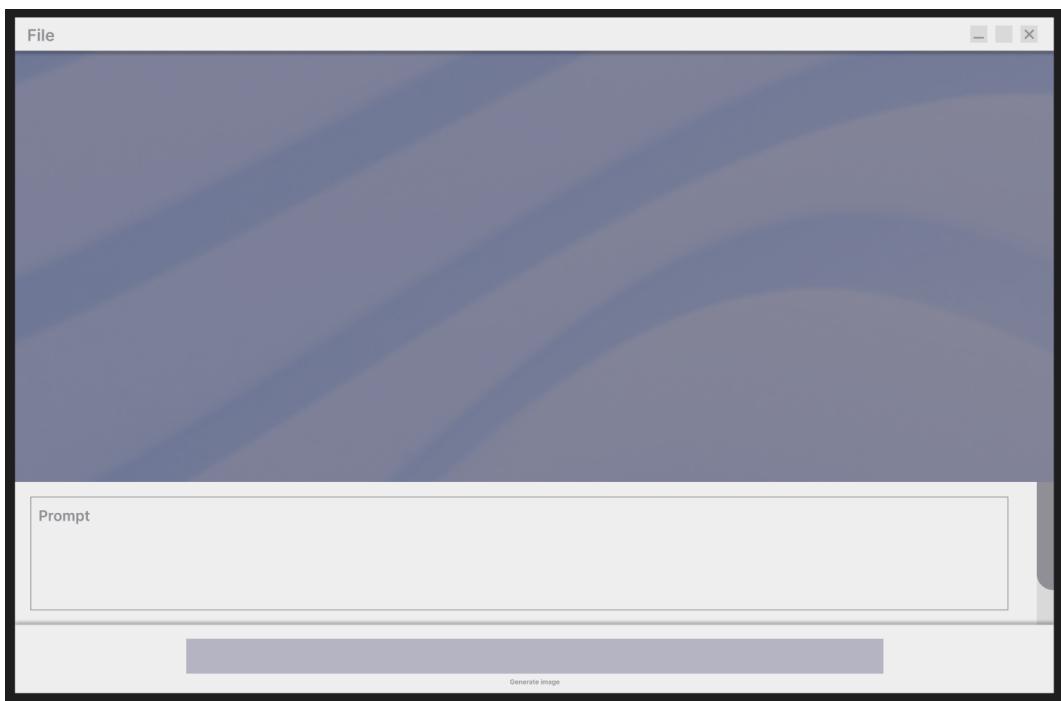


Figura 61: Ventana emergente de generación de imágenes

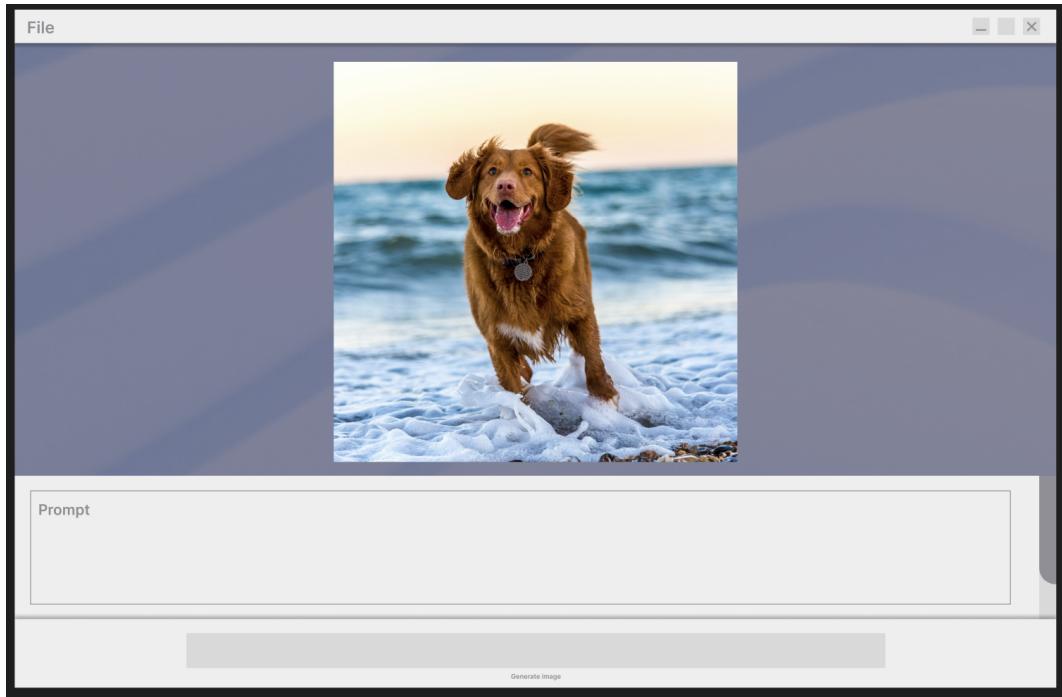


Figura 62: Imagen generada y visualizada en una ventana

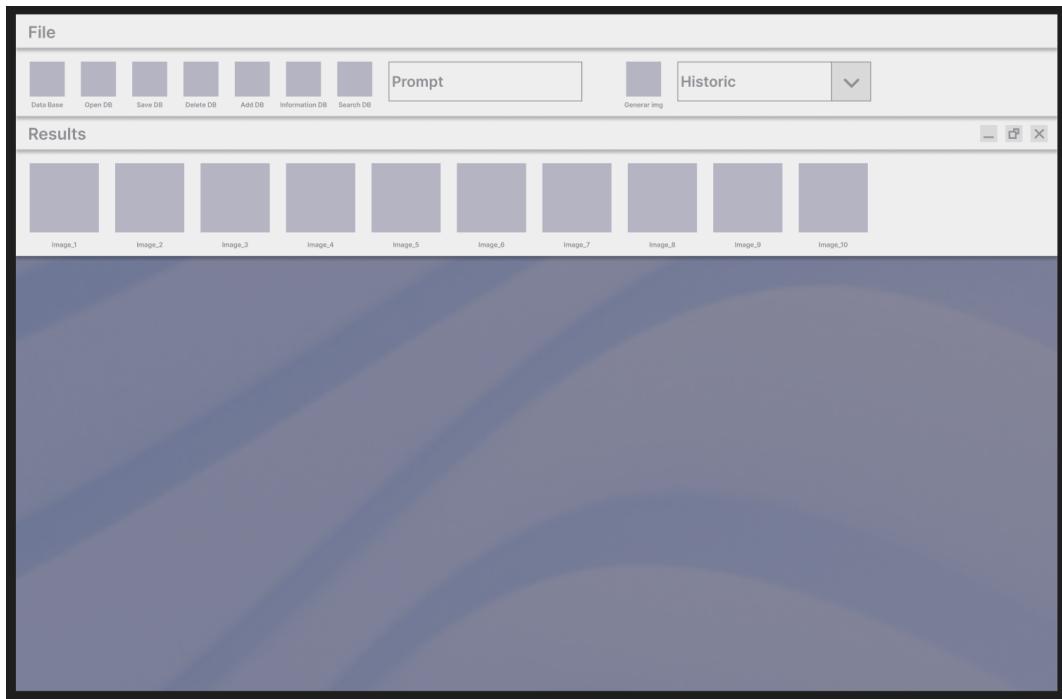


Figura 63: Galería de resultados con miniaturas

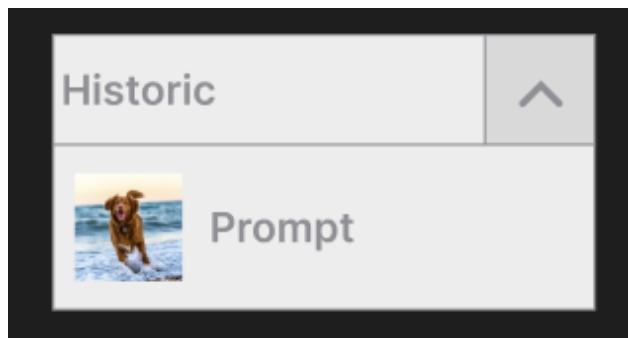


Figura 64: Histórico de prompts con miniatura

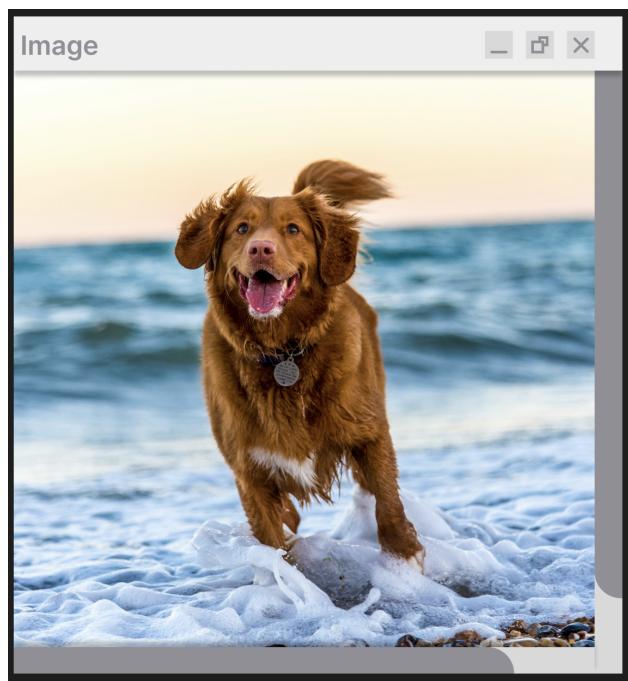


Figura 65: Resultado de abrir un prompt del histórico



Figura 66: Transición desde pantalla principal a ventana de generación

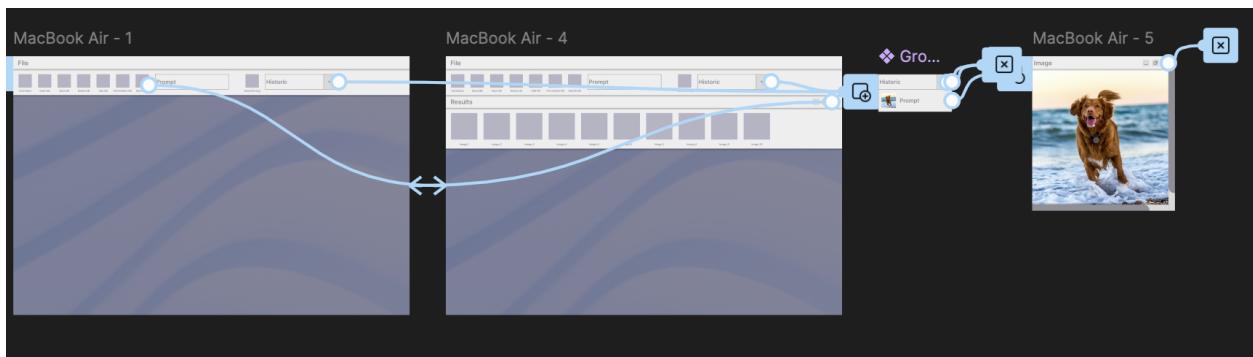


Figura 67: Transición desde histórico a ventana de visualización de imagen

B. Glosario de términos

B.1. Modelado y entrenamiento

Batch: Conjunto de ejemplos utilizados en una sola iteración de entrenamiento. El uso de batches permite un entrenamiento más eficiente y estable.

Early stopping: Técnica de optimización que se utiliza durante el entrenamiento de modelos de aprendizaje profundo. Consiste en detener el entrenamiento cuando el modelo deja de mejorar en las métricas de evaluación durante un número determinado de iteraciones, con el fin de evitar el sobreajuste y ahorrar recursos computacionales.

Entropía cruzada binaria: Función de pérdida utilizada para medir la diferencia entre dos distribuciones de probabilidad en problemas de clasificación binaria. La entropía cruzada binaria penaliza las predicciones que están lejos de las verdaderas etiquetas, con el objetivo de que el modelo aprenda a minimizar esta diferencia.

Época (Epoch): Pasada completa a través de todo el conjunto de datos durante el entrenamiento del modelo. Durante cada epoch, el modelo actualiza sus parámetros a medida que aprende a partir de los datos de entrenamiento. Generalmente, se requieren múltiples epochs para que un modelo converja a una solución óptima.

Función de Pérdida: Medida utilizada para evaluar la eficacia de un modelo durante el entrenamiento.

Hiperparámetro: Parámetro externo al modelo que se establece antes del proceso de entrenamiento. No son aprendidos por el modelo, pero afectan su rendimiento.

Normalización: Proceso de ajustar los valores de datos para que estén en un rango específico, generalmente entre 0 y 1, para mejorar la estabilidad y la velocidad de entrenamiento del modelo.

Optimización: Proceso de ajustar los parámetros de la red neuronal durante el entrenamiento para minimizar la función de pérdida.

Tasa de Aprendizaje: Hiperparámetro que controla el tamaño de los ajustes que realiza el modelo en sus pesos durante cada actualización. Un valor demasiado alto puede causar inestabilidad, mientras que un valor demasiado bajo puede hacer que el modelo aprenda muy lentamente o se estanque.

Prompt: Entrada textual que se utiliza como condición o guía para que un modelo generativo produzca una imagen. Los prompts describen el contenido esperado de forma natural o estructurada, y su calidad influye directamente en la coherencia semántica de la salida.

B.2. Redes neuronales y arquitectura

Conditional Generative Adversarial Network (cGAN): Tipo de red neuronal generativa que, a partir de un conjunto de datos y una condición específica, genera nuevas muestras similares a las del conjunto de datos original, condicionadas por etiquetas.

Colapso de modo: Problema en el entrenamiento de una GAN en el que el generador aprende a producir un conjunto limitado de imágenes muy similares, perdiendo la diversidad en las salidas. Esto ocurre cuando el generador deja de explorar nuevas combinaciones y se enfoca en un solo tipo de resultado que engaña al discriminador.

Dimensiones latentes: Tamaño del vector de ruido que se usa como entrada al generador. Este vector suele tener una distribución aleatoria, y sus dimensiones determinan la complejidad de las imágenes que el generador puede producir.

Discriminador: Parte de una red generativa adversaria cuyo objetivo es distinguir entre datos reales y datos generados. Su función es mejorar la calidad de las imágenes generadas.

Generador: Parte de una red generativa adversaria que aprende a crear datos similares a los del conjunto de datos de entrenamiento. Su objetivo es engañar al discriminador.

Neurona: Unidad de procesamiento que recibe una o varias entradas, las procesa mediante una función matemática y produce una salida. Se organizan en capas y forman la estructura básica de las redes neuronales, permitiendo al modelo aprender y tomar decisiones.

Red Neuronal: Modelo computacional inspirado en la estructura del cerebro humano, compuesto por capas de neuronas interconectadas.

Ruido aleatorio: Entrada inicial al generador que sigue una distribución normalmente distribuida (generalmente una distribución normal). El generador toma este ruido y lo transforma en una imagen. Este proceso asegura que las imágenes generadas sean diversas, ya que pequeñas variaciones en el ruido conducen a la creación de imágenes diferentes.

Atención (Attention): Mecanismo utilizado en redes neuronales que permite al modelo centrarse en las partes más relevantes de una secuencia de entrada. En generación de imágenes condicionadas por texto, la atención permite al modelo alinear palabras concretas del prompt con regiones específicas de la imagen.

Stable Diffusion: Modelo generativo basado en técnicas de difusión que transforma ruido aleatorio en imágenes coherentes guiadas por texto. Se entrena para invertir un proceso de degradación iterativa (ruido progresivo) y reconstruir imágenes de alta calidad a partir de descripciones.

GAN (Generative Adversarial Network): Arquitectura de red neuronal compuesta por dos componentes —generador y discriminador— que compiten entre sí. El generador intenta crear datos falsos convincentes mientras que el discriminador aprende a distinguirlos de los datos reales.

B.3. Datasets y datos

COCO (Common Objects in Context): Conjunto de datos que contiene imágenes con anotaciones detalladas. Incluye más de 330.000 imágenes con etiquetas y una resolución estándar de aproximadamente 640x480 píxeles.

MNIST: Conjunto de datos que contiene 70.000 imágenes de dígitos escritos a mano, divididas en un conjunto de entrenamiento y uno de prueba. Cada imagen es de 28x28 píxeles en escala de grises.

Token: Unidad básica de datos que representa una parte de un texto. Se utilizan para descomponer el texto en partes manejables, que pueden ser procesadas por modelos de lenguaje y redes neuronales.

Imagen sintética: Imagen generada artificialmente mediante modelos de IA en lugar de capturada por dispositivos físicos. Se utiliza tanto para evaluación como para enriquecer datasets en tareas de clasificación o recuperación.

B.4. Interfaz y diseño

Socket: Interfaz de software que permite la comunicación entre dos aplicaciones a través de una red. Define un punto final para la comunicación, especificando la dirección IP y el puerto en el que una aplicación escucha o envía datos.

Wireframe: Esquema visual básico de una interfaz de usuario, que muestra la disposición y estructura de los elementos clave sin entrar en detalles de diseño o contenido específico.

JMR (Java Multimedia Retrieval): Plataforma de recuperación multimedia desarrollada en Java que permite realizar búsquedas basadas en similitud visual mediante el uso de descriptores. En este trabajo, ha sido extendida para integrar capacidades de generación de imágenes a partir de texto.

C. Código fuente del sistema desarrollado

El desarrollo del sistema se ha dividido en tres módulos principales, cada uno gestionado mediante un repositorio independiente en GitHub. A continuación se describen los repositorios y su contenido:

- **Backend generativo (Python)**

Repositorio: <https://github.com/carlotiii30/TFG>

Contiene la implementación completa del sistema de generación de imágenes. Incluye los scripts de entrenamiento, el servidor de inferencia desarrollado con FastAPI, y la configuración del entorno mediante Poetry y Docker. Este componente gestiona el uso de modelos preentrenados como Stable Diffusion.

- **Extensión JMR (Java)**

Repositorio: <https://github.com/carlotiii30/TFG-JMR>

Desarrollado como una ampliación de la plataforma Java Multimedia Retrieval (JMR), este módulo integra el acceso a la API generativa. Implementa la lógica de envío de prompts, gestión de imágenes generadas y visualización dentro de la aplicación JMR.

- **Interfaz gráfica de usuario (Java)**

Repositorio: <https://github.com/carlotiii30/TFG-UI>

Aplicación Java diseñada como capa de presentación para el usuario final. Permite introducir descripciones textuales, lanzar consultas de generación y visualizar los resultados de forma intuitiva y accesible.

Todos los repositorios están actualmente marcados como privados. Para su revisión académica, el acceso puede ser concedido bajo solicitud al usuario de GitHub `carlotiii30`.

Referencias

- [1] Li, Xiaoqing, Jiansheng Yang y Jinwen Ma: *Recent developments of content-based image retrieval (CBIR)*. Neurocomputing, 452:675–689, 2021, ISSN 0925-2312. <https://www.sciencedirect.com/science/article/pii/S0925231220319044>.
- [2] Armanious, Karim, Chenming Jiang, Marc Fischer, Thomas Küstner, Tobias Hepp, Konstantin Nikolaou, Sergios Gatidis y Bin Yang: *MedGAN: Medical image translation using GANs*. Computerized Medical Imaging and Graphics, 79:101684, 2020, ISSN 0895-6111. <https://www.sciencedirect.com/science/article/pii/S0895611119300990>.
- [3] Huet, Pablo, 2023. <https://openwebinars.net/blog/que-son-las-redes-neuronales-y-sus-aplicaciones/>.
- [4] 2023. <https://aws.amazon.com/es/what-is/gan/#:~:text=Un%20sistema%20de%20redes%20generativas,salida%20es%20falsa%20o%20real>.
- [5] Calabuig Llamas, Marta, Jorge Alcaide Marzal y José Antonio Diego Más: *Modelos de generación de imágenes mediante texto en el diseño conceptual de productos. Un caso de estudio empleando Midjourney*. 2023.
- [6] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser y Illia Polosukhin: *Attention is All You Need*. En *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. <https://arxiv.org/abs/1706.03762>.
- [7] Simic, Milos: *What is Content-Based Image Retrieval?* <https://www.baeldung.com/cs/cbir-tbir>, 2024.
- [8] *Redes Generativas Antagónicas*. <https://es.mathworks.com/discovery/generative-adversarial-networks.html>. Consultado en abril de 2025.
- [9] *What is a Conditional Generative Adversarial Network (cGAN)?* <https://datacientest.com/en/what-is-a-conditional-generative-adversarial-network-cgan>, 2023.
- [10] *What is a Conditional GAN (cGAN)?* <https://www.educative.io/answers/what-is-a-conditional-gan-cgan>, 2024.
- [11] Tunstall, Lewis, Leandro von Werra y Thomas Wolf: *Natural Language Processing with Transformers*. O'Reilly Media, 2022, ISBN 9781098103244. <https://transformersbook.com/>.
- [12] Zhang, Han, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang y Dimitris Metaxas: *StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks*. En *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, páginas 5907–5915, 2017.
- [13] Xu, Tao, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang y Xiaodong He: *AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks*. En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, páginas 1316–1324, 2018.
- [14] Rombach, Robin, Andreas Blattmann, Dominik Lorenz, Patrick Esser y Björn Ommer: *High-Resolution Image Synthesis with Latent Diffusion Models*. En *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, páginas 10684–10695, 2022. <https://arxiv.org/abs/2112.10752>.
- [15] MLC AI: *Stable Diffusion Walkthrough*. <https://github.com/mlc-ai/web-stable-diffusion>, 2023. Accedido el 4 de junio de 2025.
- [16] Ho, Jonathan, Ajay Jain y Pieter Abbeel: *Denoising Diffusion Probabilistic Models*. arXiv preprint arXiv:2006.11239, 2020.

- [17] Rombach, Robin, Andreas Blattmann, Dominik Lorenz, Patrick Esser y Björn Ommer: *High-Resolution Image Synthesis with Latent Diffusion Models*. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, páginas 10684–10695, 2022.
- [18] Goodfellow, Ian, Yoshua Bengio y Aaron Courville: *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [19] Szeliski, Richard: *Computer Vision: Algorithms and Applications*. Springer, 2010, ISBN 9781848829343. <https://szeliski.org/Book/>.
- [20] Microsoft, LinkedIn Learning y: *Fundamentos profesionales de IA generativa por Microsoft y LinkedIn*. <https://www.linkedin.com/learning/paths/fundamentos-profesionales-de-ia-generativa-por-microsoft-y-linkedin>, 2023. Curso completado por la autora como parte de su formación en IA generativa.