Universidad de Valladolid

Máster universitario

# Ingeniería Informática



Trabajo Fin de Máster

## Proyecto Floorify: Generación de Planos Arquitectónicos Mediante Inteligencia Artificial Generativa

Realizado por **Giovane Eufrasio da Silva**

✠✠✠

**Universidad de Valladolid**

**27 de febrero de 2025**

Tutor: Valentín Cardeñoso Payo, Guillermo Menguez y Pablo Garcia Ullan

# Universidad de Valladolid


## Máster universitario en Ingeniería Informática

D. Valentín Cardeñoso tutor, profesor del departamento de DEPARTAMENTO DE INFORMÁTICA (ATC, CCIA, LSI), área de LENGUAJES Y SISTEMAS INFORMÁTICOS. D. Guillermo Menguez y D. Pablo Garcia Ullan, tutores por parte de la empresa.

**Expone**:

Que el alumno D. Giovane Eufrasio da Silva, ha realizado el Trabajo final de Máster en Ingeniería Informática titulado "Proyecto Floorify: Generación de Planos Arquitectónicos Mediante Inteligencia Artificial Generativa".

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Valladolid, 27 de febrero de 2025

Vº. Bº. del Tutor:

D. Valentín Cardeñoso Payo

Vº. Bº. del Tutores de Empresa:

D. Guillermo Menguez y D. Pablo Garcia Ullan

# Agradecimientos

Me gustaria expresar, en primer lugar, mi profunda gratitud a mis tutores **Guillermo Menguez** y **Pablo García Ullán** por la oportunidad de realizar mis prácticas en ***HP SCDS***, además de orientarme durante la fase de desarrollo del *TFM*. Agradezco enormemente vuestra paciencia y flexibilidad, especialmente porque me demoré un poco en finalizar la parte escrita de la investigación. También agradezco su comprensión de yo estar trabajar con un tema nuevo, cuyos resultados no fueron extremadamente positivos, pero que abrió puertas para nuevas y prometedoras investigaciones. Al final el campo de la *IA* generativa está en constante desarrollo y cada vez más presente en investigaciones multidisciplinarias.

También gustaría de agradecer a mi tutor **Profesor Valentín Cardeñoso Payo**, de la UVa, por su orientación durante el desarrollo del TFM y por proporcionarme el servidor del grupo ***ECA-SIMM*** para la realización de los ajustes experimentales.

Mis más sinceros agradecimientos a mi madre, **Elisangela Eufrasio**, quien siempre me brindó todo su apoyo y fuerza para seguir adelante, posibilitando que viniera a España a realizar este máster en la UVa. Siento que estoy cumpliendo un sueño tanto mío como de ella, que, a pesar de las dificultades de la vida, siempre creyó que el único camino hacia una vida mejor y más feliz sería a través de la educación y el aprendizaje. También agradezco a mi abuela materna y a toda mi familia por apoyar siempre mi decisión de seguir una carrera en el área de la tecnología. Además, quiero expresar mi agradecimiento a mi gran amigo **Alfonso González**, a quien tuve el privilegio de conocer durante mi estancia en España, ya que fue la persona que más me animó a no renunciar a mi máster. Siempre dándome fuerza y coraje para superar todos los desafíos, sin ti jamas habría llegado al final de esta aventura.

Agradezco profundamente a mis profesores del grado, **Diego Fiori**, quien identificó mi potencial y me incentivó a realizar este máster, y agradezco al **Omar Mozo**, sin ellos jamás habría tenido esta oportunidad en mi vida. ¡Muchas gracias!

Por último, también dedico estos agradecimientos a la memoria de mi querida amiga **Gabriela de Oliveira Vianna (Gabi)**, quien nos dejó a mediados de 2023, un mes antes de que aceptara embarcarme en esta travesía. Su recuerdo y anhelo eterno me acompañarán por siempre.

## Resumen

En el mundo contemporáneo, los modelos de Inteligencia Artificial han ganado una creciente visibilidad e importancia en la sociedad. En este contexto, presentamos el proyecto *Floorify*, que explora el uso de técnicas de *Fine-Tuning* en modelos de S*table Diffusion* con el objetivo de generar automáticamente planos arquitectónicos en *2D*. A partir de datos visuales y textuales, se evaluaron distintos métodos de ajuste fino, identificándose *Text-to-Image* como el más adecuado para los propósitos del proyecto. El objetivo fue entrenar modelos capaces de asociar descripciones textuales con imágenes de planos arquitectónicos, evaluando la calidad y coherencia de los resultados mediante métricas como *CLIP Score*. Como resultado, se logró ajustar el modelo para la generación de planos arquitectónicos en *2D*, sin embargo, los resultados aún presentaron limitaciones, tales como inexactitudes en la correspondencia entre texto e imagen y una calidad gráfica insatisfactoria, lo que abre oportunidades para futuras investigaciones.



## Descriptores

Proyecto *Floorify*, Inteligencia Artificial, Planos Arquitectónicos *2D*, Metodos de *Fine-Tuning*, *Stable Diffusion*, *Text-to-Image*, *CLIP Score*.

**Abstract**

In the contemporary world, Artificial Intelligence models have gained increasing visibility and importance in society. In this context, we present the Floorify project, which explores the use of Fine-Tuning techniques in Stable Diffusion models to automatically generate 2D architectural plans. Using visual and textual data, different fine-tuning methods were evaluated, with Text-to-Image identified as the most suitable for the project's objectives. The goal was to train models capable of linking textual descriptions with images of architectural plans, assessing the quality and coherence of the results using metrics such as CLIP Score. As a result, the model was successfully adjusted for 2D architectural plan generation. However, the outcomes still presented limitations, such as inaccuracies in the text-to-image relationship and unsatisfactory graphical quality, opening up opportunities for further research.

**Keywords**

Floorify Project, Artificial Intelligence, 2D Architectural Plans, Fine-Tuning Methods, Stable Diffusion, Text-to-Image, CLIP Score.

# Índice general

# Índice de figuras

# Índice de tablas

# 1: Introducción

En este primer capítulo, se expondrán los aspectos clave considerados en el desarrollo de esta investigación. Se abordará el problema planteado, el contexto en el que se enmarca, así como la motivación y justificación que lo sustentan. Además, se presentará la estructura que organiza el contenido del presente documento.

## 1.1. Planteamiento

En la actualidad, es cada vez más evidente que tecnologías que en épocas anteriores eran solo imaginativas o ficticias están cada vez más presentes en nuestra realidad, ya sea en los ámbitos de la tecnología móvil, espacial, industrial, entre otros. Sin embargo, una de las tecnologías más visibles en la actualidad es la Inteligencia Artificial. Este campo de la tecnología busca simular la inteligencia humana mediante automatizaciones, y se caracteriza por su gran amplitud, abarcando diversos tipos de estudios, investigaciones, técnicas de programación complejas, aplicaciones de conceptos matemáticos y reflexiones filosóficas [34]. Uno de los tipos de inteligencia artificial que ha captado mayor atención en los últimos años por parte de estudios, empresas e incluso de los medios de comunicación es el modelo generativo. Estos modelos tienen la capacidad de generar imágenes, vídeos y texto en función de lo que se les solicite o se entrene para crear de manera artificial. Una de las áreas que ha despertado un gran interés en cuanto a la automatización generativa es la creación de contenidos y la planificación de productos o proyectos específicos, los cuales a menudo deben ser presentados inicialmente a través de dibujos que reflejen el pensamiento y criterio a seguir para su ejecución, como ocurre con los planos arquitectónicos. En este sentido, numerosas empresas y universidades han impulsado investigaciones innovadoras con el fin de desarrollar nuevas soluciones y herramientas de trabajo.

A pesar de los numerosos beneficios que ofrece el uso de modelos generativos artificiales, surge la siguiente pregunta: ¿sería posible emplear una tecnología generativa ya entrenada para crear planos de construcción en formato 2D a partir de un simple *prompt* de texto? Para abordar este desafío, presentamos el proyecto *Florify*, cuyo objetivo es desarrollar una inteligencia artificial generativa capaz de generar planos arquitectónicos sencillos, automatizando así el proceso de creación y diseño de estos planos. En la sección de Contexto

del Trabajo, se detallará el origen de la idea y su relevancia para esta investigación, mientras que en los capítulos 3: Conceptos teóricos y 4: Estado del Arte se profundizará en la teoría y las investigaciones relacionadas con el tema.

## 1.2. Contexto del Trabajo

Este trabajo se enmarca dentro del contexto del TFM (Trabajo de Fin de Máster), con el objetivo de culminar el máster universitario en Ingeniería Informática, ofrecido por la Universidad de Valladolid. La temática abordada en esta investigación surgió a raíz de la asignatura de prácticas de *I+D+i* en informática, que se desarrolló en colaboración con la empresa *HP SCDS*, la cual propuso la idea y el tema de estudio.

La empresa *HP SCDS* es un centro de innovación e investigación de *HP*, que colabora con universidades para promover la innovación y la investigación en áreas de actualidad, con el fin de lograr avances tanto tecnológicos como humanos. El tema propuesto por *HP SCDS* en la asignatura de I+D+i en informática, abordado en esta investigación, se denominó *Florify*. Este proyecto tiene como objetivo desarrollar un modelo de inteligencia artificial generativa de imágenes, específicamente orientado a la creación de planos arquitectónicos sencillos en formato *2D*.

Dado el interés suscitado por el tema, se formalizó un convenio de prácticas que permitió convertirlo en el TFM con el apoyo de *HP SCDS*. Durante el período de la instancia, el objetivo principal de la empresa fue ofrecer una formación sólida en diversas tecnologías de inteligencia artificial generativa, además de facilitar la realización del estado del arte en *IA* y la creación de planos arquitectónicos mediante tecnologías de inteligencia artificial. Asimismo, se promovió el desarrollo de una *IA* generativa capaz de crear planos arquitectónicos.

## 1.3. Motivación y Justificación

Con la creciente presencia y avance de los medios de automatización laboral y los sistemas inteligentes en diversos sectores industriales y domésticos, se observa que muchas áreas del intelecto y el trabajo han logrado adaptar tareas rutinarias de manera más inteligente y eficiente, reduciendo la necesidad de intervención manual constante [12]. Como se ha demostrado en la sección de Planteamiento de este capítulo, una de las principales ventajas de utilizar la Inteligencia Artificial Generativa para automatizar los procesos de creación de planos arquitectónicos radica en su capacidad para adaptarse, generando numerosos beneficios tanto para los profesionales del diseño arquitectónico como para una amplia gama de personas. Además, esta tecnología aborda cuestiones sociales relevantes dentro de este ámbito, ofreciendo soluciones innovadoras que trascienden las necesidades tradicionales del sector.

Por ejemplo, personas sin conocimientos en dibujo arquitectónico y sin recursos para contratar a un especialista podrían crear los planos de sus futuras viviendas de manera

sencilla y sin costes adicionales. Estos planos podrían presentarse como bocetos ante los organismos responsables para su revisión técnica, detallada y posible aprobación, como los Colegios de Arquitectos, y, posteriormente, al Ayuntamiento para su aprobación final, conforme a la legislación vigente de la LOE (Ley de Ordenación de la Edificación) en España [23]. De este modo, una persona con pocos conocimientos en construcción podría obtener una primera visualización de un diseño arquitectónico simple, sin necesidad de contar con un profesional en la fase inicial del proyecto. Además, este enfoque representa un concepto completamente sostenible, eliminando el uso de papel y otros medios contaminantes comunes, tal como ya se realiza con *softwares* comerciales en los ámbitos de la arquitectura y la ingeniería. En la actualidad, existen programas informáticos comerciales que permiten este proceso de generación de planos. Sin embargo, la mayoría son de pago y requieren ciertos conocimientos en el uso de herramientas de diseño. Este tema, junto con las tecnologías existentes, será abordado en el 4: Estado del Arte.

Otro ejemplo relevante que justifica la investigación de una *IA* generativa de planos es la complejidad de su desarrollo. Aunque la idea pueda parecer simple y repetitiva, resulta extremadamente desafiante crear una *IA* capaz de generar automáticamente planos, integrando conceptos vectoriales, mediciones y todos los aspectos técnicos que generalmente realiza un ser humano en un software de diseño convencional. No obstante, el principal enfoque de esta investigación, tal como se describe en el capítulo 2: Objetivos del proyecto sobre los objetivos, será centrarse en la generación de planos sencillos a partir de *prompts* de texto. Además, lo que motiva esta investigación es la intención de abordar un tema complejo e innovador como la inteligencia artificial y sus diversas aplicaciones, un campo cada vez más relevante y en constante evolución en nuestra sociedad. Esta investigación también tiene como propósito explorar tanto conceptos novedosos como aprovechar tecnologías consolidadas, como la tecnología *Stable Diffusion*, con el fin de adaptar un modelo conforme a los objetivos establecidos en este estudio.

## 1.4. Estructura del documento

El presente documento se estructura de la siguiente forma:

- **Capítulo 1: Introducción.** En este capítulo se presenta el problema abordado en la investigación, el contexto en el que se enmarca y los aspectos que justifican su relevancia e importancia.
- **Capítulo 2: Objetivos del proyecto.** En este capítulo se detallan los objetivos generales y específicos que guiaron y orientaron la investigación, estableciendo la dirección del trabajo .
- **Capítulo 3: Conceptos teóricos.** Este capítulo facilita la comprensión del trabajo al explicar los conceptos teóricos fundamentales necesarios para el desarrollo de la investigación.

- **Capítulo 4: Estado del Arte.** En este capítulo se presenta el estado del arte recopilado durante el período de prácticas en la estancia *I+D+i*. Se identifican trabajos previos relacionados con el tema de la investigación y se analiza el panorama actual de las tecnologías de *IA* y generación de planos arquitectónicos disponibles en el mercado, tanto de código abierto como comerciales.

- **Capítulo 5: Técnicas y herramientas.** En este capítulo se describe la metodología utilizada en la investigación, detallando los requisitos esenciales para desarrollar el modelo generativo de planos arquitectónicos. Además, se explican los complementos utilizados, como la infraestructura, conjuntos de datos, métodos de ajuste fino y métricas aplicadas para evaluar tanto el desarrollo algorítmico como los resultados experimentales del modelo.

- **Capítulo 6: Aspectos relevantes del desarrollo del proyecto.** En este capítulo se narran los pasos del desarrollo, incluyendo las transformaciones en los conjuntos de datos seleccionados y el ajuste de modelos de *Stable Diffusion* mediante técnicas como *Dreambooth, LoRAs, Text-to-Image*, entre otros, con el fin de generar un modelo generativo adaptado a las necesidades del proyecto.

- **Capítulo 7: Discusión de Resultados.** En este capítulo se presentan los resultados obtenidos para cada modelo, evaluados mediante las métricas *CLIP Score* y *FID (Fréchet Inception Distance)*. Además, se analizan las funciones de pérdida durante el entrenamiento, realizando diferentes análisis y profundizando en cada una de las conclusiones derivadas de los resultados obtenidos.

- **Capítulo 8: Conclusiones y Líneas de trabajo futuras.** Este capítulo expone las conclusiones finales de la investigación y propone diversas formas de continuar el trabajo iniciado en el futuro, explorando nuevas posibilidades y desarrollos.

- **Apéndice A. Plan de Proyecto.** En este apéndice se describen las herramientas utilizadas y la metodología seguida en el proyecto, se detalla el plan inicial de trabajo y se realiza un breve seguimiento de la planificación y desarrollo del proyecto en sus primeras etapas.

- **Apéndice B. Manual de Instalación.** En este apéndice se describen las herramientas utilizadas y la metodología seguida en el proyecto, se detalla el plan inicial de trabajo y se realiza un breve seguimiento de la planificación y desarrollo del proyecto en sus primeras etapas.

# 2: Objetivos del proyecto

Para definir de manera más precisa la meta a alcanzar con esta investigación, hemos establecido una serie de objetivos que guiarán el desarrollo del proyecto en cuestión.

## Objetivos Generales

Se ha establecido un objetivo general que captura la esencia del problema, el cual orienta y da coherencia al desarrollo de la investigación:

- **Desarrollar un modelo capaz de generar imágenes de planos arquitectónicos en *2D* a partir de un simple *prompt* de texto introducido por el usuario.** El proyecto debe incluir una interfaz gráfica web que permita al usuario interactuar de manera intuitiva y generar las imágenes de planos arquitectónicos de acuerdo con especificaciones sencillas y comunes.

## Objetivos Específicos

Con base en el objetivo general, se plantean los siguientes objetivos específicos:

- **Llevar a cabo una investigación con el estado del arte sobre generación de imágenes mediante *IA* (Inteligencia Artificial) y generación de planos mediante *IA*.** En ambas investigaciones, será necesario verificar los modelos inteligentes y algoritmos existentes, tanto en modalidad pública (*Open Source*), disponibles para estudios y modificaciones, como los modelos comerciales, para entender cómo funcionan las aplicaciones disponibles en el mercado. Además, se investigarán métricas específicas aplicadas en modelos generativos de imágenes, para evaluar cómo validar el aprendizaje y la calidad de un modelo sencillo.
- **Buscar conjuntos de imágenes de planos arquitectónicos públicos disponibles en la web.** Todos los conjuntos de datos recopilados deben detallarse mediante una lista informativa, que incluya la diferencia entre ellos y el número

total de imágenes por conjunto. Después de analizar los conjuntos de datos, se seleccionará el que se utilizará para el entrenamiento del modelo. Posteriormente, se realizarán las transformaciones y limpiezas de datos necesarias para su aplicación en el entrenamiento del modelo.

- **Utilizar un modelo preexistente y entrenado para realizar un *Fine-Tuning*.** En caso de utilizar un modelo ya entrenado, será necesario ajustar los pesos del modelo al conjunto de imágenes seleccionado para el entrenamiento, realizar el entrenamiento de la *IA* y luego probar la generación de imágenes de planos.
- **Crear una demostración del modelo y realizar pruebas mediante una interfaz gráfica con la biblioteca de *Python*, *Gradio*, para verificar los resultados generativos en tiempo real.**
- **El sistema debe permitir la descarga de imágenes en diferentes formatos.** Los formatos de imagen disponibles deben incluir formatos *RGB* comunes como *JPG* o *PNG*, y, si es posible, formatos vectoriales editables como *SVG* y *AutoCAD*.
- **Probar el modelo generativo utilizando métricas algorítmicas específicas o mediante análisis y pruebas con observación humana.** Esto permitirá verificar la calidad de las imágenes generadas y la precisión en relación con los requisitos especificados en el *prompt* de texto que guía la generación de la imagen.

# 3: Conceptos teóricos

Para una mejor comprensión del tema tratado en este trabajo, es fundamental realizar un estudio bibliográfico y teórico sobre Inteligencia Artificial y sus diferentes áreas, Inteligencia Artificial Generativa, así como sobre Planos Arquitectónicos y las tecnologías asociadas. Además, se llevará a cabo un análisis de las tecnologías y modelos ya entrenados que actualmente se encuentran presentes en este campo. Los conceptos mencionados anteriormente están profundamente vinculados al proyecto desarrollado en esta investigación y serán abordados en este capítulo de manera estructurada por secciones.

## 3.1. Inteligencia Artificial (IA)

Actualmente, el campo de estudio y trabajo de la inteligencia artificial es uno de los más vastos y expansivos en el mercado tecnológico. Gracias al aumento y almacenamiento de datos por parte de diversas instituciones y empresas, ha sido posible desarrollar modelos de aprendizaje de alto rendimiento, que buscan automatizar y optimizar la eficiencia de procesos manuales y repetitivos que anteriormente eran realizados por seres humanos [46]. No obstante, a pesar del reconocimiento y prestigio que la inteligencia artificial ha adquirido en la actualidad, es importante entender que este concepto no es nuevo. Para que existan modelos de aprendizaje tan sofisticados como los actuales, fue necesario atravesar diferentes etapas de estudio, estabilidad y avance.

El término Inteligencia Artificial (*IA*) se define como una tecnología computacional capaz de simular la inteligencia humana en la resolución de problemas matemáticos, estadísticos y procesos repetitivos, con la capacidad de aprender patrones a partir de datos de diferentes conceptos. Además, posee una capacidad algorítmica para identificar materiales y elementos del entorno humano [46]. López y Messenger (2017) mencionan que los conceptos de inteligencia artificial deben ser entendidos a través de dos concepciones fundamentales: *IA* Débil e *IA* Fuerte, que se corresponden con las siguientes definiciones: "La *IA* es la ciencia e ingeniería que permite diseñar y programar ordenadores para realizar tareas que requieren inteligencia. La *IA* es la ciencia e ingeniería que permitirá replicar la inteligencia humana mediante máquinas"(López y Meseguer, 2017, p. 8) [61].

El concepto de inteligencia artificial se divide entre *IA* Débil e *IA* Fuerte. Esta distinción fue propuesta en 1980 por el filósofo John Searle, quien intentaba demostrar la dificultad de crear una *IA* Fuerte. Según sus estudios, la inteligencia artificial débil se refiere al diseño y creación de *IAs* que exhiben comportamientos inteligentes orientados a tareas muy específicas. En contraste, la *IA* Fuerte se relaciona con una inteligencia general, con conocimientos técnicos en diversas áreas, sentimientos, creatividad, y la capacidad de distinguir lo real de lo irreal en su razonamiento, lo que implica una simulación completa del cerebro humano. Según López y Meseguer (2017), los avances actuales en *IA* son ejemplos de *IA* Débil, que se enfoca en una inteligencia específica y no general [61].

El campo de la Inteligencia Artificial en la actualidad abarca una gran variedad de áreas de aprendizaje automático, técnicas de programación y campos de estudio. Entre los más conocidos se encuentran el *Machine Learning* y el *Deep Learning*, que se engloban dentro del campo denominado ciencia de datos. Este campo tiene como objetivo trabajar con grandes volúmenes de datos para realizar estudios, predicciones y deducciones mediante la aplicación de ciencias matemáticas y estadísticas a través de algoritmos [48]. Su propósito es explicar comportamientos, identificar tendencias e incluso hacer previsiones de mercado, ayudando a empresas y organismos públicos a obtener mejores perspectivas sobre mercados futuros, aumentar beneficios y atraer visibilidad pública, basándose en diversos tipos de análisis de datos y filtros [48]. Además, cada vez que se menciona alguna de las áreas de *IA*, estamos hablando también de la ciencia de datos, ya que todos los métodos de aprendizaje requieren conjuntos de datos que deben ser analizados, limpiados y generalmente ser de gran escala para entrenar nuevos modelos inteligentes. Este enfoque permite llevar a cabo automatizaciones para resolver problemas de regresión y clasificación, aplicándose a diversos fines y métodos en campos como Visión Computacional, Procesamiento de Lenguaje Natural e *IA* Generativa.

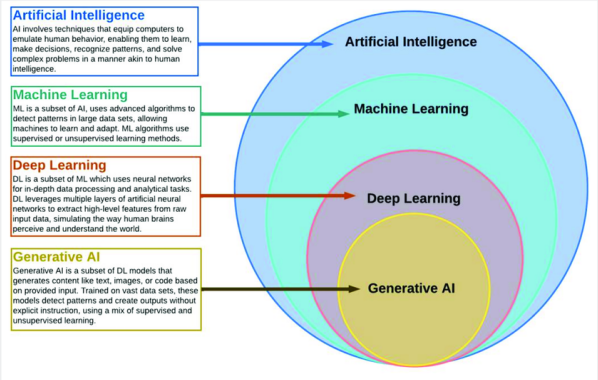


Figura 3.1: Una visión comparativa de la *IA* [59]

Como se ha mencionado anteriormente, el concepto de Inteligencia Artificial es amplio, abarcando diversas áreas y métodos aplicables en el estudio y desarrollo de nuevas tecnologías. En las subsecciones siguientes, exploraremos brevemente qué es *Machine Learning* y, en particular, *Deep Learning*, que serán fundamentales para comprender cómo

se caracteriza el desarrollo de *IAs* Generativas, así como su estrecha relación con el tema tratado en esta investigación.

## *Machine Learning*

El término "*Machine learning*"fue utilizado por primera vez en 1959 por Arthur Samuel, y se refiere a una tecnología y área de estudio que utiliza algoritmos matemáticos para que las computadoras identifiquen patrones en grandes volúmenes de datos con el objetivo de realizar predicciones o elaborar análisis predictivos sobre temas específicos. "*Machine learning*"se resume en el aprendizaje automático mediante *scripts*, aprovechando el auge de los datos almacenados en internet y por las empresas a partir de los años 2000. Según Alecrim (2018, trad. del portugués), el aprendizaje automático es un sistema que puede modificar su comportamiento de manera autónoma basándose en su propia experiencia [6]. Este proceso permite realizar modificaciones a través de reglas lógicas y reconocimiento de patrones dentro de un conjunto de datos, generados a través de reglas definidas para automatizar un sistema o función específica. Según Arthur Samuel (1959, citado por Géron 2019, p.4, trad. del portugués) [37], el aprendizaje automático es el campo de estudio que otorga a las computadoras la habilidad de aprender sin ser programadas explícitamente. De este modo, el concepto de *Machine Learning* se divide en dos vertientes principales: aprendizaje supervisado y no supervisado.

Sin embargo, actualmente el área de *Machine Learning* no se limita solo a estas dos vertientes. Hoy en día, también contamos con métodos de aprendizaje por refuerzo, que permiten la mejora continua de los modelos mediante la interacción con su entorno. A continuación, se presenta la definición de cada uno de estos enfoques:

- **Aprendizaje Supervisado:** Según Géron (2019, p. 8, trad. del portugués), En el aprendizaje supervisado, el conjunto de entrenamiento que se proporciona al algoritmo incluye las soluciones deseadas, llamadas etiquetas [37]. Las tareas de clasificación y regresión lineal son algoritmos típicos de aprendizaje supervisado. Las Definiciones de clasificación y regresión:
  - **Clasificación:** Se define Clasificación como .Acción o efecto de clasificar, de reunir en clases y en grupos respectivos, según un sistema o método. Un ejemplo es hacer la clasificación de correos electrónicos nuevos como *spam* o normales.
  - **Regresión linear:** Matemáticamente, la 'regresión lineal' es el 'proceso de trazar una línea recta a través de los datos en un diagrama de dispersión. La línea resume esos datos, lo cual es útil cuando hacemos predicciones.' (Khan Academy, 2022, trad. del portugués)[2]. Con los datos en regresión, es posible hacer predicciones de un mercado financiero y clasificaciones.
- **Aprendizaje no Supervisado:** El aprendizaje no supervisado tiene como objetivo aprender de forma autónoma, sin la necesidad de instrucciones o etiquetas de datos proporcionadas previamente. Este tipo de aprendizaje se utiliza para diversos fines

algorítmicos, como la detección de anomalías, la identificación de novedades y la búsqueda de agrupaciones naturales en grandes conjuntos de datos. En este enfoque, la inteligencia artificial intenta aprender de nuevos datos que difieren de los presentes en el conjunto de entrenamiento, realizando un análisis de patrones. Un ejemplo de este algoritmo sería el siguiente: si tienes miles de fotos de perros y el 1 % de ellas son de Chihuahuas, un algoritmo de detección de novedades no debería tratar las nuevas fotos de Chihuahuas como novedades. Por el contrario, los algoritmos de detección de anomalías podrían considerar estos perros como tan raros y diferentes de otros que probablemente los clasificarían como anomalías. (Géron, 2019, p. 11, trad. del portugués)[37]

- **Aprendizaje por Refuerzo:** El principal objetivo del aprendizaje por refuerzo es utilizar un algoritmo que aprenda a partir de su propia experiencia, empleando la técnica de ensayo y error. Este enfoque es ampliamente utilizado en áreas como juegos, gestión de recursos y robótica, ya que se basa en el método de recompensas para acelerar el aprendizaje del algoritmo. En el aprendizaje por refuerzo, los desarrolladores crean un sistema para recompensar comportamientos deseados y penalizar comportamientos negativos. Se asignan valores positivos a las acciones que se desean fomentar, incentivando al algoritmo a utilizarlas, mientras que se asignan valores negativos a las acciones indeseadas para desalentaras. De esta manera, se programa a la inteligencia artificial para buscar recompensas máximas y a largo plazo, con el objetivo de alcanzar una solución óptima. Estas metas a largo plazo ayudan a evitar que el aprendizaje se quede estancado en objetivos menos importantes. Con el tiempo, la *IA* aprende a evitar las acciones negativas y a centrarse en las positivas. (Hashemi-Pour, 2024, trad. del inglés)[38].

## Deep Learning (Redes Neuronales)

La subárea de Inteligencia Artificial denominada ***Deep Learning*** ofrece un enfoque más profundo del aprendizaje automático, utilizando redes neuronales profundas para resolver problemas complejos. Su objetivo es simular el cerebro biológico, pero de manera algorítmica, matemática y lógica [37]. Actualmente, existen diversos modelos de redes neuronales que se emplean en la práctica; algunos ejemplos incluyen: ***Multi-Layer Perceptron (MLP)***, ***Convolutional Neural Networks (CNN)***, ***Recurrent Neural Networks (RNN)***, ***Generative Adversarial Networks (GAN)***, ***Transformers***, entre otros modelos que están siendo desarrollados continuamente para aportar innovación y resolver problemas que requieren una gran cantidad de datos. Además, el *Deep Learning* es uno de los métodos más complejos para el desarrollo de modelos inteligentes.

El área de *Deep Learning* tiene principios fundamentales que deben ser considerados. No se detallarán todos en esta investigación, ya que son conocimientos básicos para los lectores de este trabajo. A diferencia de los métodos de *Machine Learning* que requieren la extracción manual de características y generalmente se basan en aprendizaje supervisado, los modelos de *Deep Learning* son capaces de aprender directamente de las muestras de entrenamiento, extrayendo automáticamente las características relevantes durante el

proceso de entrenamiento [37]. Aunque *Deep Learning* se considera más una técnica de aprendizaje no supervisado, existen diferentes aplicaciones para los distintos tipos de aprendizaje. A continuación, se enumeran los principios fundamentales de esta tecnología:

- **Redes Neuronales Artificiales:** El área de *Deep Learning* utiliza redes neuronales multicapa, ya que cada capa está formada por neuronas artificiales encargadas de procesar la información. Este principio es esencial para trabajar con el enfoque de aprendizaje profundo [37].
- **Entrenamientos:** Los procesos de entrenamiento en *Deep Learning* pueden ser largos y complejos de ajustar, ya que dependen del uso adecuado de optimizadores, funciones de activación y otros conceptos fundamentales que forman parte de una red neuronal, además de la correcta asignación de los pesos. Estos elementos deben aplicarse de manera precisa según los análisis realizados para resolver distintos problemas de aprendizaje. Para verificar los ajustes de los pesos, generalmente se utilizan algoritmos de optimización como el *Gradient Descent* (función de pérdida) [37].
- **Arquitecturas:** En la actualidad, existen diversas arquitecturas y modelos entrenados que se utilizan para abordar diferentes tipos de problemas, tales como clasificación y regresión. Además, todas las redes neuronales requieren grandes volúmenes de datos de entrenamiento para aprender de manera eficaz. Dependiendo del tipo de datos, como en el caso del trabajo con imágenes usando la arquitectura *CNN*, es fundamental contar con una infraestructura computacional adecuada, empleando *Graphics Processing Unit (GPU)* para procesar los datos de manera más ágil y eficiente durante las fases de entrenamiento [49]. Asimismo, casi todas las arquitecturas utilizan el algoritmo *Backpropagation*, que es clave para el ajuste de los pesos y la evaluación de la función de pérdida [37].

Actualmente, esta área se utiliza en problemas específicos que se han convertido en campos especializados de estudio y aplicación de diversas redes neuronales, las cuales permiten resolver una amplia gama de problemas e impulsar innovaciones. A continuación, se enumeran algunas de las áreas, aplicaciones y las redes neuronales más empleadas en cada uno de estos campos:

- **Visión Computacional:** Este campo tiene como objetivo entrenar redes neuronales para desarrollar modelos capaces de aprender a partir de la información contenida en imágenes digitales, videos y otros tipos de datos visuales. La finalidad es realizar tareas de reconocimiento y clasificación mediante la identificación de patrones que el modelo ha aprendido durante su entrenamiento. En este ámbito, las redes neuronales profundas, como las *Convolutional Neural Networks (CNN)*, son fundamentales. Estas redes permiten entrenar modelos utilizando imágenes etiquetadas para clasificar e identificar objetivos específicos dentro de una imagen o video [47].

- **Procesamiento de Lenguaje Natural (*NLP*):** Enfocado en la interacción y comprensión del lenguaje entre las computadoras y los seres humanos, el Procesamiento de Lenguaje Natural busca que las máquinas entiendan idiomas, habla, sentimientos y otros aspectos derivados del lenguaje humano, para interpretar y generar textos en lenguaje natural. Entre las herramientas más utilizadas en *NLP* se encuentran los *Transformers* para la generación de texto, traducción de idiomas y análisis de sentimientos. También se emplean Redes Neuronales Recurrentes (*RNN*) para el procesamiento de secuencias y *Long Short-Term Memory (LSTM)* para modelos de datos temporales. Un ejemplo destacado de esta tecnología es *ChatGPT* de *OpenAI*, un modelo diseñado para generar textos y responder preguntas de los usuarios humanos sobre una amplia variedad de temas [46].

- ***IA* Generativa:** Este campo ha ganado gran notoriedad en los últimos años y se centra en el desarrollo de modelos de aprendizaje profundo capaces de generar texto, imágenes, videos y otros contenidos a partir de lo que han aprendido durante su entrenamiento. La *IA* generativa combina diferentes tipos de tecnologías y modelos de aprendizaje para producir contenido. Esta área es una de las más complejas, ya que demanda una infraestructura de *hardware* avanzada para entrenar las redes neuronales. Entre las tecnologías más destacadas en este campo se encuentran las *Generative Adversarial Networks (GAN)*, *Transformers*, y *Stable Diffusion*, entre otras. En la próxima sección, se profundiza sobre este tema, que es el enfoque principal de este trabajo [13].

Los campos de estudio e investigación en el área de *Deep Learning* han experimentado una evolución constante con el paso del tiempo. Tal como se mencionó anteriormente, han surgido numerosas subáreas y ramificaciones a partir de esta evolución. No obstante, es evidente que cada una de estas ramificaciones requiere un perfeccionamiento y desarrollo continuo, tanto en términos algorítmicos como en la parte investigativa teórica. Este proceso ha ocurrido a lo largo de varias décadas, produciendo resultados asombrosos que han beneficiado a la sociedad. Sin embargo, a pesar de estos avances, aún no se ha logrado desarrollar una *IA* fuerte, como se explicó anteriormente.

## 3.2. Inteligencia Artificial Generativa

En los últimos años, la sociedad ha logrado crear innovaciones tecnológicas extraordinarias que, durante gran parte de la historia humana, solo existían en la imaginación. Una de estas grandes y revolucionarias creaciones son las *IAs* generativas, cuyo objetivo específico es generar contenidos de diversos tipos, como imágenes, vídeos, audio, textos, conversaciones y muchas otras cosas, de una manera diferente, a partir de contenidos que no existen, sin necesidad de ser creados por manos o intelectos humanos. Según Foster (2019), un modelo generativo puede definirse de manera amplia como aquel que describe cómo se genera un conjunto de datos en términos de un modelo probabilístico. Al tomar muestras de este modelo, es posible generar nuevos datos [28].

La inteligencia artificial generativa no es necesariamente una tecnología reciente. Históricamente, ya en la década de 1960 existían proyectos destinados a crear máquinas capaces de generar contenido de forma automática y lograr cierto grado de inteligencia para la automatización. Un ejemplo de ello es el proyecto *ELIZA*, de 1966, considerado el primer *chatbot* de la historia, presentado por el profesor Joseph Weizenbaum. *ELIZA* tenía como objetivo simular una conversación de psicoterapia. Sin embargo, los proyectos de esa época no contaban con los algoritmos complejos, la gran cantidad de datos para entrenamiento ni la tecnología actual, elementos esenciales para crear una *IA* que genere contenido de manera efectiva y precisa [57]. Este avance se estancó y solo se reactivó en 2014 con las tecnologías de aprendizaje profundo, aplicadas en una nueva arquitectura de *Deep Learning* denominada *Generative Adversarial Networks (GAN)*, presentada por el científico Ian Goodfellow [28]. Este estancamiento de la tecnología, también conocido como los períodos de invierno de la inteligencia artificial, constituyó un período histórico en el cual la *IA* no experimentó avances significativos debido a diversos desafíos estructurales, tanto tecnológicos como éticos. Estos desafíos estuvieron impulsados por el pensamiento crítico y las limitaciones tecnológicas de la época en la que ocurrieron los tres inviernos de la *IA* [75].

Actualmente, el campo de la *IA* generativa está en constante expansión, con diversos tipos de algoritmos y arquitecturas que tienen la capacidad de generar contenidos con una calidad sorprendente. En esta investigación, nos centraremos en la tecnología que dio origen y revolucionó las *IAs* generativas, conocida como *GAN*, y en la arquitectura que se empleará en esta investigación, denominada *Stable Diffusion*. En las siguientes subsecciones, explicaremos cada una de estas tecnologías, destacando sus conceptos y ventajas.

## *Generative Adversarial Networks* (*GAN*) - 2014

Las *GANs (Generative Adversarial Networks)* fueron una de las primeras arquitecturas generativas eficaces y se han demostrado sumamente útiles para generar imágenes, textos y música. Su arquitectura se basa en una lógica algorítmica de competencia adversarial, compuesta por dos elementos clave: el generador y el discriminador. El generador es el responsable de crear nuevos contenidos de manera realista, mientras que el discriminador tiene como objetivo identificar si el contenido generado es original o si ha sido producido por una *IA*, es decir, reconocer lo que sería un contenido "falso.º "*fake*". El proceso funciona como una competencia en la que el generador intenta constantemente mejorar la calidad de sus contenidos para engañar al discriminador, mientras que este último se esfuerza por mejorar sus capacidades para identificar correctamente los contenidos generados. La interacción continua entre ambos adversarios continúa hasta que el generador produce contenidos tan convincentes que el discriminador ya no puede diferenciar entre lo generado y lo real. Este proceso de competencia es lo que permite que las *GANs* mejoren con el tiempo, generando resultados de alta calidad que, en muchos casos, son indistinguibles de aquellos creados por seres humanos [28].

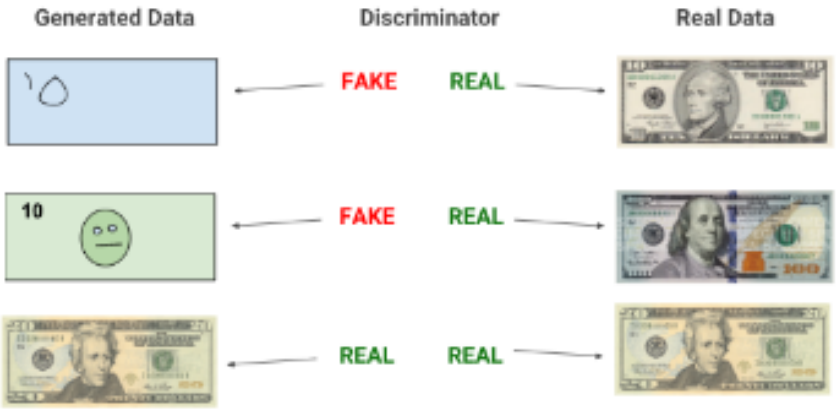


Figura 3.2: Ejemplo del generador y el discriminator de las GANs [35].

Para que toda esta lógica funcione y se generen imágenes de manera efectiva, la arquitectura de las *GANs* opera de forma compleja, requiriendo una ecuación extremadamente desafiante. El generador se basa en una red neuronal con arquitectura *MLP (Multi-Layer Perceptron)* o *CNN (Convolutional Neural Network)*, siendo esta última la más tradicional para el procesamiento de imágenes. El generador analizará un gran conjunto de datos de entrenamiento, identificando los atributos presentes en los mismos. Por otro lado, el discriminador también utiliza una red neuronal que examinará el conjunto de datos de entrenamiento y distinguirá los atributos de los datos de forma independiente. A través de este proceso, el generador y el discriminador interactúan y se entrenan de manera conjunta. A continuación, se detallan los pasos clave en este proceso:

- **Generador:** El generador modifica algunos atributos de los datos al agregar ruido o realizar cambios aleatorios sobre la imagen o los datos. Estos datos modificados son luego enviados al discriminador para su análisis [13].
- **Discriminador:** Por su parte, el discriminador, después de procesar los datos proporcionados por el generador, realiza un cálculo probabilístico para verificar si los datos generados realmente pertenecen al conjunto de datos original. El discriminador analiza estos datos, identificando sus atributos, y proporciona resultados y orientación para el generador. Este, a su vez, tiene como objetivo reducir la aleatorización del ruido sobre las imágenes, buscando así mantener un equilibrio en lo que se está generando [13].

Según *Amazon AWS*, el generador intenta maximizar la probabilidad de error del discriminador, mientras que este último busca minimizar dicha probabilidad de error. Durante las iteraciones de entrenamiento, tanto el generador como el discriminador evolucionan y se enfrentan de manera continua hasta alcanzar un estado de equilibrio. En este estado, el discriminador ya no es capaz de reconocer los datos sintetizados [13].

En este momento, el proceso de entrenamiento finaliza. A continuación, se presenta un ejemplo de la arquitectura de *GAN*:

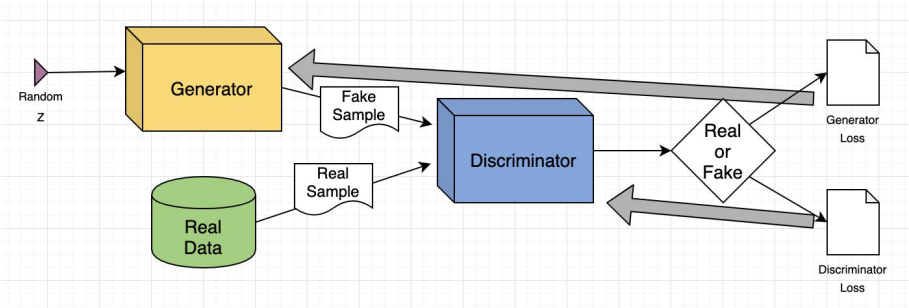


Figura 3.3: Arquitectura de las *GANs* [13].

Una de las dudas que siempre surgen al presentar la arquitectura de las *GAN* es cómo se generan realmente las imágenes y otros tipos de datos. En esta investigación, nos enfocaremos específicamente en el contexto de las imágenes, que pueden seguir el siguiente ejemplo presentado por *Amazon AWS* [13]. Si la arquitectura recibe fotos de rostros humanos en su conjunto de datos de entrenamiento, y el objetivo es modificar el rostro humano o crear uno nuevo, el generador debe identificar los principales atributos del rostro y generar algo aleatorio. A medida que el proceso avanza, el discriminador analiza la imagen generada e identifica los atributos de un rostro humano que se asemejan a los que están contenidos en el conjunto de datos de entrenamiento. Este proceso continúa hasta que el generador produce una imagen que cumple con los criterios del discriminador.

Un ejemplo similar ocurre con el cambio de estructuras y colores en diferentes objetos y muestras, como se describe en la figura 3.3. En este caso, el generador recibe imágenes de cebras en su conjunto de entrenamiento, mientras que el discriminador recibe imágenes tanto de cebras como de caballos, para aprender los atributos de ambos animales. A partir de ahí, el generador intenta crear una cebra que tenga los atributos de un caballo. El discriminador proporciona las indicaciones necesarias para que el generador modifique la imagen hasta que la cebra tenga la estructura de una cebra, pero con los atributos de un caballo, lo que resulta en la modificación de los atributos de la imagen generada [13].

Actualmente, los modelos *GAN* siguen siendo muy populares, con nuevos métodos y algoritmos aplicados a una amplia variedad de problemas con el objetivo de generar datos. Sin embargo, las *GANs* fueron el punto de partida para el desarrollo de nuevas tecnologías generativas, que abordaremos en este trabajo, y que han ganado un gran reconocimiento por parte de desarrolladores y empresas tecnológicas.

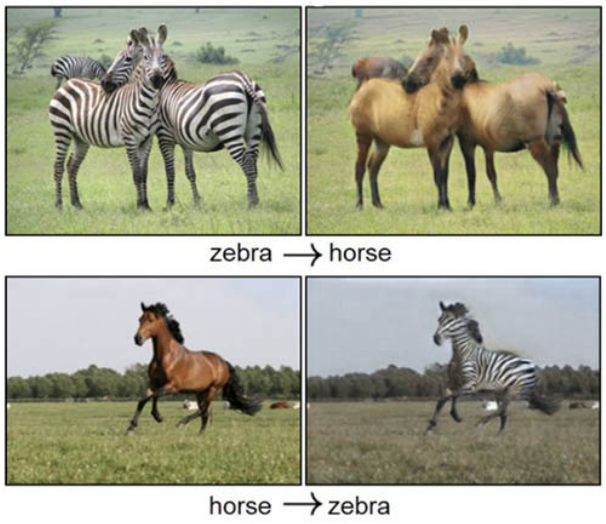


Figura 3.4: Ejemplos de imágenes generadas con *GANs*: cebra creada a partir de atributos de un caballo y viceversa [13].

## *Stable Diffusion* y Los Modelos de Difusión

Con el paso de los años, desde la aparición de las redes generativas adversarias (*GANs*), pioneras en el desarrollo de modelos inteligentes capaces de generar imágenes, han surgido otras tecnologías que han transformado el campo de la inteligencia artificial generativa. Uno de estos modelos es el que genera imágenes a través de la arquitectura de difusión, conocida como *Stable Diffusion*. Presentada por primera vez en 2022 y desarrollada por *Stability AI*, esta arquitectura fue considerada una revolución, ya que era capaz de generar imágenes de alta definición de manera sencilla y, además, era completamente de código abierto (*open source*), permitiendo que empresas, desarrolladores y estudiantes pudieran modificar o adaptar el código a sus necesidades. Esto promovió un mayor avance tecnológico y aprendizaje en el campo de la *IA* [67].

*Stable Diffusion* ha ganado mucha notoriedad no solo por ser de código abierto y generar imágenes de altísimo calidad, sino también por no requerir infraestructuras computacionales grandes, complejas y de alto costo, lo que permite a usuarios comunes o empresas con bajo presupuesto utilizarla, a diferencia de otras tecnologías. Además, cuenta con diferentes métodos de *Fine-Tuning* sencillos que permiten a cualquier persona adaptar unas pocas imágenes de manera simple a modelos pre-entrenados, lo que facilita la mezcla de sus datos y la creación de nuevos conceptos artísticos o experimentales.

Para entender mejor cómo funciona la arquitectura de *Stable Diffusion*, insertamos la figura 3.5 a continuación, que representa todos los componentes algorítmicos involucrados en la generación de imágenes a través de la manipulación de píxeles, ya sea mediante la interpretación de texto o el uso de otras imágenes [72]. En esta investigación, no profundizaremos en la teoría ni en la explicación detallada del concepto de imágenes y

píxeles, ya que se considera un tema de conocimiento básico para los lectores de este trabajo.

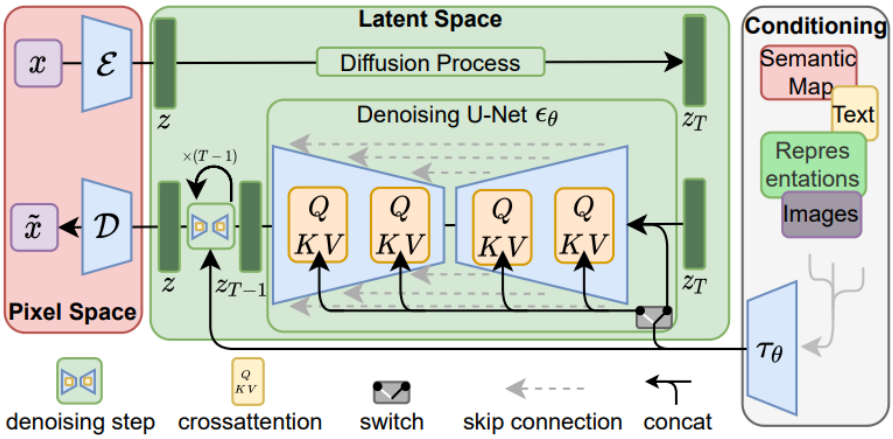


Figura 3.5: Arquitectura de la técnica de *Stable Diffusion* [72]

- ***Conditioning*:** Este término se refiere a los datos de entrada en la arquitectura, los cuales pueden estar compuestos por textos descriptivos, imágenes o representaciones semánticas. Estos datos son acondicionados por algoritmos específicos; en los casos de textos, que son los más comunes, el acondicionamiento se realiza mediante la *tokenización* de *CLIP (Contrastive Language-Image Pre-Training)*. CLIP analiza cada palabra del texto y la inserta en un vector específico. Este proceso se repite varias veces, enviando varios *tokens* al *predictor* de ruido *U-Net* mediante una transformación de texto en imágenes (píxeles) y capturas semánticas de los datos realizadas por el CLIP. Así se efectúan las primeras generaciones de imágenes llenas de ruido [11].

- ***Latent Space:*** El espacio latente es el proceso principal para generar imágenes, siendo el espacio que recibe texto o imágenes provenientes del "*Conditioning*", ya transformados. En el caso del texto, este llega como *tokens*, y dentro del espacio latente se realiza el mapeo en el que se crean los primeros píxeles, basados en los valores semánticos y características capturadas de los mensajes de texto, con la adición de ruido. A partir de esto, es posible llevar a cabo el proceso de difusión, que consiste en una serie gradual de eliminación de ruidos del vector latente para crear una representación limpia de las características recolectadas de las imágenes, y que coincida con la información entrenada en el modelo principal. De esta manera, se puede realizar el proceso de decodificación de vuelta al espacio de píxeles (*Pixel Space*), lo que da como resultado una imagen generada correspondiente al vector no latente, es decir, a los mensajes de texto [72].

- ***Denoising U-Net:*** La ***U-Net*** se conoce comúnmente como el núcleo de la arquitectura, ya que es en esta parte donde se realizan y crean las principales transformaciones.

Se encuentra dentro del componente "*Latent Space*". Esta sección de la arquitectura está compuesta por la *U-Net*, que incluye redes neuronales convolucionales preentrenadas con diferentes tipos de imágenes, y cuya función principal es predecir el ruido en las imágenes para llevar a cabo la limpieza de la foto y su posterior generación [11]. El ***Cross-Attention*** es otro componente crucial, ya que tiene la capacidad de enfocar y concentrar las semánticas y patrones de los textos, aplicándolos a la imagen después de las transformaciones realizadas por el *Conditioning*. Además, existen las *Keys*, *Queries* y *Values*, que son responsables de toda la lógica aplicada en el espacio latente. Las *Queries* se combinan con las *Keys* para calcular y ajustar los pesos de las redes neuronales, lo que permite un mejor control y enfoque del mecanismo en el espacio latente. Por lo tanto, podemos considerar el componente *Denoising U-Net* como el núcleo que tiene la capacidad de transformar una imagen formada por ruido aleatorio en una imagen coherente, hermosa y realista, alineada con el texto recibido a través del espacio no latente (*Conditioning*) [72].

Actualmente, *Stable Diffusion* ofrece diversos métodos y modelos disponibles en formato de código abierto (*Open Source*), los cuales pueden ser utilizados para una amplia gama de fines de desarrollo e investigación. Además, existe una gran diversidad de métodos para realizar *Fine-Tuning* en los modelos de difusión, a través de plataformas como *Stability AI*, *Hugging Face*, entre otras. En la actualidad, los métodos de *Fine-Tuning* son ampliamente utilizados en modelos generativos, ya que facilitan significativamente la adaptación de un modelo que ha sido entrenado con una gran cantidad de datos a una tarea específica, como la generación de contenido particular, ya sea de texto o imágenes. Cabe recordar que el *Fine-Tuning* se define como un método para ajustar los pesos de un modelo o red neuronal previamente entrenada sobre un conjunto de datos más pequeño y personalizado, estableciendo una nueva tarea específica con esos datos y entrenando el modelo en consecuencia [7].

Los métodos de *Fine-Tuning* de *Stable Diffusion* se abordarán con más detalle en el capítulo 4: Estado del Arte, ya que esta parte fue realizada de manera sistemática con el objetivo de estudiar y aplicar los métodos de *Fine-Tuning* de *Stable Diffusion* en el desarrollo de este trabajo, realizado durante el proceso de estancia de I+D+i. Además, en el capítulo mencionado, presentaremos en formato de tabla los principales modelos de diferentes tecnologías de generación de imágenes, además de *Stable Diffusion*, proporcionando un análisis más profundo sobre el tema.

## 3.3. Planos Arquitectónicos y Sus Tecnologías

Los planos arquitectónicos han sido de gran importancia en la vida humana desde que las primeras civilizaciones comenzaron a planificar sus asentamientos. Por ejemplo, en la civilización del Antiguo Egipto (c. 3100-332 a.C.), según teóricos egiptólogos, la construcción de las pirámides de *Guiza* se llevó a cabo utilizando planos de construcción extremadamente detallados. Estos planos, tal como han descubierto los arqueólogos,

indicaban que los antiguos egipcios posiblemente utilizaban cuerdas, varas y dibujos para planificar y alinear las pirámides con una precisión astronómica sobre la tierra [74]. Con el tiempo, los planos arquitectónicos se modernizaron y se volvieron cada vez más relevantes en la sociedad contemporánea. Durante mucho tiempo, estos planos fueron creados mediante dibujos manuales, hasta la llegada de tecnologías que brindaron soporte y optimización a un trabajo que, aunque puede parecer sencillo, puede ser extremadamente complejo, incluso para los algoritmos más sofisticados de la actualidad.

Los planos arquitectónicos se definen como un conjunto extenso de dibujos que describen cada parte de edificios y viviendas, detallando cómo deben construirse las edificaciones. Los detalles en un plano de construcción pueden variar, desde los materiales y acabados hasta la definición de toda la parte eléctrica y mecánica de un edificio. Estos conjuntos de dibujos se crean con el propósito de visualizar y planificar un proyecto que se llevará a cabo en una posible construcción, en colaboración con un equipo especializado [16]. La elaboración profesional de planos arquitectónicos suele ser responsabilidad de arquitectos, aunque es posible que personas no profesionales realicen bocetos, siempre y cuando estos sean posteriormente revisados por un equipo profesional y aprobados para ser considerados en el proceso de construcción. En España, por ejemplo, existe una ley denominada LOE (Ley de Ordenación de la Edificación), que establece que todos los planos arquitectónicos realizados por personas no profesionales deben ser revisados y mejorados por un colegio de arquitectos. Tras su aprobación, deben ser tramitados ante el ayuntamiento de la ciudad donde se desea construir el edificio. Lo mismo aplica a proyectos realizados directamente por un arquitecto [23].

A continuación, se enumeran algunos de los planos arquitectónicos que se realizan comúnmente en la actualidad:

- **Planos de Situación y de Casas:** Estos planos pueden presentarse en formatos *2D* o 3D y son particularmente útiles para visualizar un terreno urbano en su totalidad, además de servir como base para la construcción de una vivienda o la reforma de un espacio específico dentro de una edificación. Son los planos más conocidos, generalmente elaborados por arquitectos, y comúnmente se les denomina plantas bajas, ya que ofrecen una representación del contexto del plano desde una vista superior [16].
- **Planos Mecánicos y Eléctricos:** Los planos de este tipo tienen como objetivo mostrar todos los sistemas de tuberías y la instalación eléctrica de una edificación. A menudo, pueden estar incluidos junto con los planos de situación y de la casa. Este tipo de plano generalmente es elaborado por ingenieros especializados en cada área, quienes posteriormente lo combinan con los planos realizados por los arquitectos [16].

En el período contemporáneo, existen numerosas herramientas disponibles para la creación de planos arquitectónicos. Los *softwares* más utilizados actualmente incluyen *AutoCAD Architecture*, *Revit* (*AutoDesk y AutoCAD*), *Civil 3D*, *Draft IT*, entre otros [20].

Existe una amplia variedad de software en el mercado para la elaboración de plantas bajas, algunos más profesionales y de pago, mientras que otros son más simples y gratuitos. La elección de la herramienta adecuada generalmente depende de las necesidades específicas de cada usuario o del tipo de proyecto que se esté desarrollando.

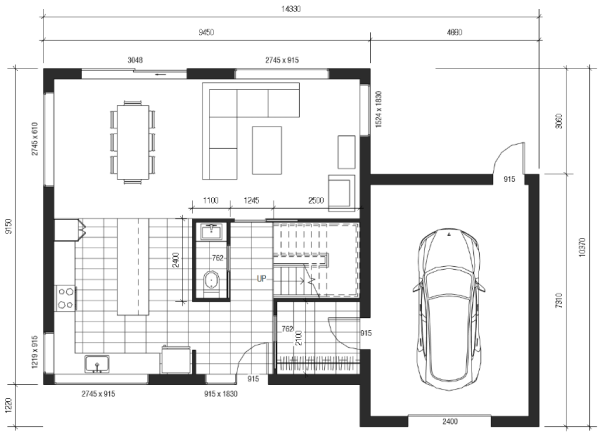


Figura 3.6: Ejemplo de plano arquitectónico *2D*, creado con *AutoCAD* (*AutoDesk Revit*) [66].

A pesar de la existencia de diversas herramientas para la creación de planos arquitectónicos, se observa una notable carencia tecnológica en cuanto a la ampliación y mejora de este tipo de procesos, especialmente en un momento en que se habla cada vez más de la automatización del trabajo. Actualmente, existe una gran necesidad de aplicar conceptos de inteligencia artificial para automatizar el proceso de diseño de planos de casas, ya que el proceso creativo a menudo puede ser repetitivo y llevar mucho tiempo de creación para los profesionales [53]. En el capítulo 4: Estado del Arte, nos centraremos en demostrar tecnologías y productos comerciales que ya utilizan *IA* generativa para crear planos, además de presentar investigaciones que intentan automatizar este proceso y guiar el rumbo de nuestra investigación.

# 4: Estado del Arte

Previo a este trabajo, se realizó un estudio exhaustivo sobre la cantidad de investigaciones y las tecnologías disponibles relacionadas con los temas de Generación de Imágenes mediante *IA* y Generación de Planos mediante *IA*. En este capítulo, se presenta una revisión detallada de los temas mencionados, destacando trabajos científicos clave y proporcionando una visión general de las últimas tecnologías disponibles, tanto en formato de código abierto (*Open Source*) como en productos comerciales. Cabe recordar que esta investigación se llevó a cabo originalmente durante la fase de la estancia I+D+i, como se mencionó en el capítulo 1: Introducción. Los temas mencionados ya fueron abordados de manera general en el capítulo 3: Conceptos teóricos, donde se presentaron las teorías y conceptos más tradicionales. En este capítulo, profundizaremos más en el conocimiento sobre estos temas, presentando aspectos más relevantes que fueron determinantes para la elección de la tecnología utilizada en el desarrollo del modelo generativo discutido en esta investigación.

## 4.1. Generación de imágenes mediante *IA*

Como se mencionó en el capítulo 3: Conceptos teóricos de esta investigación, los modelos generativos han experimentado una evolución constante, produciendo contenidos cada vez más realistas. Es importante recordar que estos modelos se basan en una gran cantidad de datos que han sido entrenados sobre su arquitectura, con el objetivo de generar contenido específico. En este trabajo, nos enfocaremos completamente en la generación de imágenes mediante *IA*, un campo que, como hemos visto, ha logrado consolidarse a través de diversas técnicas y arquitecturas de inteligencia artificial. Como se presentó en los conceptos teóricos, las inteligencias artificiales generativas alcanzaron su apogeo con las arquitecturas GANs, que fueron introducidas en 2014. Sin embargo, con el tiempo han surgido nuevas arquitecturas que, cada vez más, son mejores y más fáciles de usar para los desarrolladores.

En este capítulo, nos centraremos en detallar la tecnología de *Stable Diffusion* y los procesos de generación de imágenes con esta arquitectura, considerando las diversas

funcionalidades que, por varios motivos que se citarán, nos llevaron a elegirla para llevar a cabo el desarrollo del objetivo principal de este proyecto.

Como mencionamos anteriormente, la tecnología *Stable Diffusion* ha evolucionado notablemente, produciendo imágenes de alta definición cada vez más realistas. Además, ha facilitado su uso para diversos temas de desarrollo, principalmente porque es completamente *Open Source*. También ofrece métodos de *Fine-Tuning*, muchos de los cuales son proporcionados por la propia *Stability AI* a través de comunidades de desarrolladores y bibliotecas de programación de otras empresas tecnológicas. Asimismo, muchas comunidades de desarrolladores con un enfoque en Inteligencia Artificial han mejorado y modificado algunas funcionalidades y entrenamientos de los modelos de *Stable Diffusion*, lo que permite realizar *Fine-Tunings* con diferentes propósitos de manera más sencilla.

Los métodos de *Fine-Tuning* de *Stable Diffusion* son conocidos como ***Diffusers Training*** y son proporcionados por la empresa ***Hugging Face***. Su objetivo es ofrecer métodos algorítmicos ya preparados, para que los desarrolladores puedan ajustarlos fácilmente a sus necesidades. A continuación, listamos algunos de estos métodos:

- ***Dreambooth*:** Es un método utilizado para personalizar modelos de texto a imagen (*text-to-image*), como *Stable Diffusion*, utilizando solo unas pocas imágenes (generalmente de 3 a 5) de un tema específico. *DreamBooth* fue presentado por primera vez en el artículo científico titulado "*DreamBooth*: *Fine-Tuning Text-to-Image Diffusion Models for Subject-Driven Generation*"[65]. Este fue el primer método de ajuste fino (*fine-tuning*) desarrollado para *Stable Diffusion*, permitiendo que el desarrollador aplique únicamente unas pocas imágenes para personalizar el modelo, junto con una única frase que identifica el tipo de imagen que se desea generar. Es un método potente, capaz de adaptarse a diversos temas de imágenes, aunque presenta algunas limitaciones en cuanto al entrenamiento de imágenes y frases de identificación (*embeddings*) [41].

- ***Text-to-Image*:** Es el nuevo método de ajuste fino para las tecnologías de *Stable Diffusion*, que aunque aún se encuentra en fase experimental en 2024, ya está disponible para los desarrolladores. Este ajuste es fácil de aplicar y extremadamente poderoso, permitiendo ajustar conjuntos de imágenes de gran escala mediante un simple archivo *.Json*, que debe contener una variedad de frases relacionadas con cada imagen. Con este enfoque, es posible trabajar con temas más complejos en la generación de imágenes, lo que proporciona una amplia gama de interacciones para generar diferentes tipos de fotos, ilustraciones y dibujos [44].

- ***LoRa (Low-Rank Adaptation)*:** *LoRa* es una técnica que optimiza los métodos de *Fine-Tuning* de *Dreambooth* y *Text-to-Image*, permitiendo un ajuste más eficiente. *LoRa* fue presentado por primera vez en el artículo científico titulado "*LoRA: Low-Rank Adaptation of Large Language Models*"[22]. Inicialmente, su propósito era ser un método exclusivo para el procesamiento de lenguaje natural (generación de texto), pero debido a su versatilidad, se adaptó también para los ajustes finos en

la generación de imágenes. *LoRa* permite adaptar modelos previamente entrenados añadiendo pares de matrices de peso de descomposición de rango bajo, llamadas matrices de actualización, a los pesos existentes y entrenando solo estos nuevos pesos añadidos. Esto permite realizar el ajuste fino de manera más eficiente en términos de uso de memoria y procesamiento en la *GPU*, aunque no impide que se generen modelos más grandes en cuanto a ocupación de memoria, lo que puede requerir más tiempo de entrenamiento. En general, *LoRa* es una técnica eficiente para ajustar grandes modelos de difusión con menos esfuerzo computacional, manteniendo la capacidad del modelo para generar salidas de alta calidad [43].

Actualmente, todos los métodos mencionados anteriormente están disponibles de manera simplificada a través de una biblioteca llamada *Diffusers*, creada por la empresa *Hugging Face*. A través de esta biblioteca, los usuarios pueden acceder completamente a los algoritmos de *Stable Diffusion*, desarrollados por *Stability AI*. Esto facilita la posibilidad de realizar ajustes finos sin necesidad de adaptar todo un contexto algorítmico para un problema específico, ya que el *Fine-Tuning* proporciona esta capacidad de forma automática.

Con el respaldo de las comunidades y de la propia *Stability AI* para realizar ajustes finos en sus modelos, con el objetivo de resolver diversos problemas, parece muy sensato optar por un *Fine-Tuning* para generar imágenes relacionadas con nuestro tema, que es el desarrollo de un modelo capaz de generar imágenes de planos arquitectónicos. En las siguientes subsecciones, abordaremos algunos temas clave, como la autoría de las imágenes creadas por *IA*, así como los productos y tecnologías disponibles en el mercado en la actualidad.

## Autoría de Imágenes Creadas Por *IAs* Generativas

Un tema que debe ser abordado al referirse a la inteligencia artificial generativa es el de los derechos de autor sobre el contenido generado por la *IA*. Este es un tema que aún se encuentra en un intenso debate, dado que la tecnología ha logrado evolucionar y tener éxito en sus creaciones recientemente. Por ello, se examina y debate quién es el verdadero propietario de una imagen, texto o video creado por una *IA*. Según investigaciones realizadas, muchas empresas y especialistas en el tema defienden que solo la persona que generó el contenido, a través de un texto o *prompt*, es quien detenta todos los derechos de autor sobre el contenido específico generado por la *IA* [15].

En territorio español y europeo en general, se sostiene que las obras producidas de forma autónoma por agentes artificiales y robots no deben ser elegibles para protección de derechos de autor, con el fin de respetar el principio de originalidad, el cual está vinculado a una persona física (humana). No obstante, en ciertas situaciones, la *IA* puede actuar como una herramienta adicional para el autor, enriqueciendo su creatividad mediante instrucciones adecuadas [70]. Sin embargo, en los sitios web de muchas empresas que ofrecen productos de *IA* generativa, como *OpenAI*, se puede observar que se atribuye el derecho de autor a la persona que generó la imagen, quedando a su criterio cómo utilizarla.

En la sección de Productos y Tecnologías Disponibles y Comerciales de Generación de Imágenes de este capítulo, identificaremos cómo cada empresa atribuye los derechos de autor a sus usuarios.

## Métricas de Evaluación de Modelos Generativos

Con los avances de los modelos de *IA* generativos, se hace evidente la gran necesidad de realizar evaluaciones sobre la calidad de los contenidos generados. A diferencia de otros modelos de inteligencia artificial, cuyo objetivo es evaluar la precisión o la tasa de acierto de clasificaciones y predicciones, los modelos generativos solo disponen de la verificación de las tasas de aprendizaje, sin ser sometidos inicialmente a métricas específicas definidas para ellos. Las métricas de evaluación de modelos generativos son diversas, pero existe una gran complejidad en garantizar la calidad de los contenidos generados, ya que el proceso de evaluación del nivel de calidad de un modelo es detallado y varía considerablemente de un modelo a otro, siendo necesario emplear varias métricas para verificar la calidad [52].

Actualmente, los modelos generativos de imágenes requieren evaluaciones tanto algorítmicas como humanas para verificar la autenticidad, la calidad y el buen proceso creativo en la imagen generada. A continuación, listamos algunas de las métricas más utilizadas, según investigaciones:

- **Evaluación Humana (*Human Evaluation*):** La evaluación humana es indispensable para verificar la calidad de las imágenes generadas a partir de textos. Sin embargo, presenta algunas desventajas, ya que generalmente puede ser un proceso demorado y está sujeta a opiniones subjetivas. Por lo tanto, se considera una métrica complementaria en la evaluación de modelos generativos de imágenes [52].
- **Métricas Basadas en Píxeles (*Pixel-Based Metrics*):** Esta métrica utiliza algoritmos métricos como el Error Cuadrático Medio (*MSE*), el índice de similitud estructural (*SSIM*) y otros. Estos algoritmos consisten en comparar las imágenes generadas (píxeles) con imágenes reales del mismo dominio. La métrica evalúa la calidad de la imagen en función de la similitud entre los píxeles, proporcionando una evaluación cuantitativa de la precisión visual [52].
- **Métricas Basadas en Características (*Feature-Based Metrics*):** Esta métrica emplea métodos de evaluación como el *Inception Score (IS)*, la *Fréchet Inception Distance (FID)* y otros. Estos métodos comparan las distribuciones de características entre las imágenes generadas y las imágenes reales, y determinan cómo el modelo ha logrado preservar la calidad y diversidad del tema tratado en la imagen [52].
- **Métricas Basadas en Tareas (*Task-based Metrics*):** La evaluación de modelos generativos también puede involucrar el uso de métricas orientadas a tareas, evaluando qué tan bien las imágenes generadas cumplen con funciones posteriores, como clasificación, segmentación, etiquetado o recuperación. Esta métrica, sin embargo, no siempre es eficaz en todos los casos, ya que está más orientada a contextos textuales

que a imágenes. Algunos ejemplos de algoritmos utilizados incluyen la precisión de clasificación, la precisión de segmentación, la puntuación BLEU, entre otros [52].

- **Métricas Basadas en Novedad (*Novelty-Based Metrics*):** En este enfoque se utilizan métodos como la distancia al vecino más cercano (*nearest neighbor distance*), cobertura (*coverage*) y entropía (*entropy*). Estos métodos intentan evaluar la diferencia y diversidad de las imágenes generadas en comparación con las existentes dentro de un dominio de entrenamiento o de un tema similar. Sin embargo, es importante señalar que, aunque estas métricas destacan la creatividad, pueden no tener en cuenta el realismo y la relevancia de las imágenes creadas, favoreciendo resultados que pueden ser poco realistas o irrelevantes [52].

Como podemos observar, la calidad de las imágenes generadas puede evaluarse de diversas maneras. Sin embargo, los métodos cuantitativos (algorítmicos) suelen ser los más utilizados para definir la calidad, ya que el método cualitativo, que implica la visión y evaluación humana, puede generar resultados más subjetivos, siendo a menudo utilizado solo para complementar las métricas ya empleadas.

Otra opción para evaluar la calidad de un modelo generativo de imágenes es utilizando una herramienta innovadora lanzada por *OpenAI*, denominada ***CLIP* (*Contrastive Language-Image Pretraining*)**, un modelo entrenado para aprender y relacionar textos con imágenes [5] [40]. Esta tecnología es especialmente indicada para trabajar con modelos generativos denominados ***Text-to-Image***, ya que resulta muy útil para evaluar la coherencia entre las imágenes generadas y el texto utilizado [77].

El proceso de utilización de *CLIP* se ve influenciado en el momento de generar las imágenes, ya que su objetivo es calcular la similitud entre la descripción textual original y la imagen generada. *CLIP* proporciona una puntuación que indica el grado de similitud entre el texto y la imagen generada, cuanto mayor sea el valor o mas próximo esté de 1, mayor será la similitud y correspondencia. En comparación con otras métricas comúnmente utilizadas, como el *Inception Score (IS)* y el *Fréchet Inception Distance (FID)*, *CLIP* se presenta como una herramienta más inteligente y robusta. Mientras que las métricas IS y FID calculan la distancia entre características y la diversidad entre imágenes, *CLIP* supera estas tecnologías al realizar una evaluación de calidad similar, pero verificando la similitud entre el texto recibido y la imagen generada (*embeddings*) [42]. No obstante, es crucial considerar las métricas *IS* y *FID*, ya que fueron las primeras utilizadas para evaluar este tipo de inteligencia artificial, y siguen siendo las principales métricas orientadas a características [14].

La aplicación de cada tipo de métrica dependerá del tipo de sistema inteligente que se desee crear y probar, y muchas veces puede ser necesario utilizar más de una métrica. Cabe recordar que todas las métricas mencionadas tienen como objetivo evaluar el nivel creativo y generativo de la inteligencia artificial.

## Productos y Tecnologías Disponibles y Comerciales de Generación de Imágenes

En esta subsección, presentaremos algunas tablas obtenidas a partir de una recopilación de información, organizada de acuerdo con cada empresa y tecnología indicada, además de datos provenientes de las comunidades de desarrolladores. El propósito de incluir esta información es permitir una visualización clara de las diferencias entre cada tecnología generativa y proporcionar una visión general de lo que está disponible en el mercado tecnológico. A continuación, se presentan las tablas:

| **Sistema Inteligente** | ***DALL-E (Open AI)*** | | |
|---|---|---|---|
| **Modelo** | ***DALL-E Mini (Boris Dayma)*** | ***DALL-E 2*** | ***DALL-E 3*** |
| **Descripción** | *DALL-E Mini* es una *IA* generativa capaz de crear imágenes a partir de descripciones de texto. Utiliza técnicas de aprendizaje profundo para generar imágenes que coincidan con las descripciones proporcionadas. Es una tecnología *open source*, desarrollada por el programador *Boris Dayma* en 2022. | *DALL-E* es un modelo de *IA* desarrollado por *OpenAI* que genera imágenes a partir de descripciones textuales. Actualmente tiene dos modelos de pagos creados entre 2021 y 2024. | *DALL-E* es un modelo desarrollado por *OpenAI* que genera imágenes a partir de descripciones textuales. |
| **Costo General y Tipo 2024** | *Open-Source* (Abierto) | Comercial: De $0,16 por imagen a $0,18 por imagen | Comercial: De $0,40 por imagen a $0,120 por imagen |
| **Derechos del Autor** | Segunda la *Open AI*, üsted es propietario de las imágenes que crea con *DALL-E*, incluida el derecho a reimprimir, vender, y mercancías, independientemente de si su imagen se generó a través de un crédito gratuito o de pago." | | |
| **Infraestructura para la Aplicación** | *DALL-E* y *Dayma* no lo definen en su documentación, sin embargo, al ser un medio para manipular imágenes en *RGB*, lo mejor es tener una infraestructura con una GPU de 8 GB de VRAM o más. | | |

| | | |
|---|---|---|
| **Compatibilidad (*Fine-Tunning*)** | La documentación no deja clara la posibilidad de la existencia de *Fine-Tunning*, es decir, el modelo está completamente disponible en repositorios públicos en Internet y puede modificarse. | *OpenAI* no deja clara la posibilidad y no ofrece soporte en su documentación de productos y aplicaciones finales. |

Tabla 4.1: Modelos existentes referenciáis en generación de imágenes - Dalle-E

| **Sistema Inteligente** | ***Stable Diffusion* (*Stability AI*)** | | |
|---|---|---|---|
| **Modelo** | ***Stable Diffusion* 3** | ***Stable Diffusion XL*** | ***SDXL* Turbo** |
| **Descripción** | *Stable Diffusion* es un motor de inteligencia artificial diseñado para crear imágenes a partir de texto, fue creado por la empresa *Stability AI* en 2022. Actualmente cuenta con tres modelos principales para la generación de imágenes. | | |
| **Costo General y Tipo 2024** | *Open-Source* (Abierto), para uso no comercial, pero hay planes para asociarse a la tecnología y obtener algunas herramientas adicionales para desarrollar en su entorno, además del ser *model* ideal para que las empresas apliquen la tecnología con fines comerciales. | | |
| **Costo Comercial 2024** | Costo de $20.00, para el plan *Professional* o costo personalizado para el plan *Enterprise*, para uso comercial. | | |
| **Derechos del Autor** | Las imágenes de *Stable Diffusion* como cualquier otra forma de contenido creativo, están sujetas a protección de derechos de autor. | | |
| **Infraestructura para la Aplicación** | Mínimo: *GPU* - 6/8 *GB* de *VRAM* | | |

| | |
|---|---|
| **Compatibilidad (*Fine-Tunning*)** | Existen algunas posibilidades de *Fine-Tuning* que pueden ser visualizadas en la propia documentación de *Stability AI*. Para el *SDXL*, *XL* y Turbo, existen los tipos *Face Mode, Juggernaut XL, DreamShaper XL, RealVisiXL, Animagine XL, Object Mode, Juggernaut, RealCarto* y *Style Mode*. Además, hay algunos métodos de ajuste disponibles en *Huggin Face* llamados *DreamBooth*, *LoRA*, *Textual inversion*, *Text-to-image*, entre otros. Los métodos como *LoRA* pueden implementarse fácilmente utilizando *Python*. |

Tabla 4.2: Modelos existentes referenciáis en generación de imágenes - *Stable Diffusion*

| **Sistema Inteligente** | ***MidJourney*** | | |
|---|---|---|---|
| **Modelo** | ***Model* 6** | ***Model Niji* 6** | ***Model* 5.2** |
| **Descripción** | El Modelo *MidJourney* es una tecnología, de sistemas inteligentes, que tiene objetivo de crear imágenes a partir de texto, fue creada por un laboratorio independiente en Sao Francisco en 2022. | | |
| **Costo General 2024** | Los planes están todos pagados y están diseñados para todos los modelos. Están incluidos en los planes el *Basic*, *Standard*, Pro y Mega. Los precios van desde 10,00*hasta*120.00. | | |
| **Derechos del Autor** | Según *MidJourney*, los suscriptores de *MidJourney* tienen acceso a todas las imágenes que han creado, incluso si la suscripción ha caducado, y son libres de usar esas imágenes como deseen. | | |
| **Infraestructura para la Aplicación** | No es necesario descargar en la máquina local. El usuario puede generar las imágenes en el propio sitio web o en el telegrama de la herramienta *MidJourney*. | | |
| **Compatibilidad (*Fine-Tunning*)** | Existe la posibilidad de utilizar un sintonizador de estilo de herramienta, de los cuales es posible utilizar métodos de ajuste fino con la propia herramienta *MidJourney*. | | |

Tabla 4.3: Modelos existentes referenciáis en generación de imágenes - *MidJourney*

Al comparar los tres principales modelos presentados en las tablas anteriores, podemos observar que cada uno de ellos ofrece diversas opciones adicionales, además de presentar varios tipos de planes tanto comerciales como profesionales para satisfacer las necesidades de cada usuario. Al analizar las opciones disponibles, se destaca que todos los modelos generan imágenes a partir del texto proporcionado en el *prompt* de comandos de cada uno. La mayoría de las tecnologías tienen una vertiente de código abierto, lo que significa que

su código está disponible para ser modificado y adaptado a problemas específicos de algún proyecto o investigación.

Dentro de los modelos de código abierto, los de *Stable Diffusion* de *Stability AI* son los que más ventajas ofrecen, ya que todos sus modelos están disponibles de forma abierta. No obstante, cuentan con un plan de asociación de desarrolladores que otorga acceso a algunos recursos adicionales de forma ilimitada. Por otro lado, *DALL-E* de *OpenAI* solo tiene una versión de código abierto llamada *DALL-E Mini*, un proyecto basado en el sistema inteligente de *OpenAI* que fue creado por *Boris Dayma*. Según diversas fuentes, este modelo logra dar resultados sorprendentes y eficaces. También está *MidJourney*, que ofrece todas las versiones de su modelo bajo planes de pago, aunque en algunas excepciones de experimentación, los usuarios pueden generar algunas imágenes de forma gratuita a través de su extensión en *Discord*, una red social.

Un aspecto muy interesante a considerar en la investigación son las compatibilidades para realizar *Fine-Tuning* en un modelo, especialmente en aquellos que son de código abierto. Algunas tecnologías, como *Stable Diffusion*, cuentan con varios modelos ajustados con compatibilidad para realizar *Fine-Tuning*, cubriendo diferentes temas y métodos, como es el caso de *Dreambooth*, *LoRa* y otros. Sin embargo, sistemas inteligentes como *MidJourney* o *DALL-E* no cuentan con soporte ni modelos específicos para el refinamiento del modelo (*Fine-Tuning*).

## 4.2. Generación de Planos Arquitectónicos Mediante *IA*

Con los avances en las áreas de inteligencia artificial generativa, muchas empresas y emprendedores han encontrado una nueva fuente de innovación, motivados por la creación de soluciones basadas en lo que ya existe para automatizar parte de su trabajo diario o para utilizar la tecnología como un nuevo medio comercial. Aunque sigue siendo un tema complejo para aquellos sin el conocimiento adecuado en tecnología, muchas empresas han invertido considerablemente para desarrollar nuevos sistemas generativos, especialmente en áreas específicas de conocimiento y trabajo. Las organizaciones están invirtiendo cada vez más en estos sistemas para refinar, optimizar y simplificar muchos procesos de trabajo, cuyas aplicaciones pueden abarcar desde la mejora de experiencias de atención al cliente hasta la creación de nuevos productos [8].

Como se mencionó en el capítulo 3: Conceptos teóricos, los planos de una vivienda son la representación gráfica de dicho inmueble, sirviendo como una herramienta básica y un elemento esencial del diseño que permite plasmar y comunicar información precisa sobre un proyecto arquitectónico [9]. Hoy en día, la creación de muchos proyectos de construcción se lleva a cabo en software de modelado *2D* y *3D*, como *Revit*, *AutoDesk Studio*, entre otros. Además, algunos proyectos se desarrollan en herramientas de realidad aumentada o virtual, aunque la forma más tradicional sigue siendo el modelado *2D*, conocido como planos bajos. El proceso de desarrollo de un plano abarca varias etapas, desde la creación

de los primeros bocetos, que consideran los requisitos del cliente y del profesional, hasta el desarrollo de un proyecto estructural que garantice la seguridad del edificio y la creación de un proyecto arquitectónico definitivo. Hoy en día, el proceso de diseño ya cuenta con el apoyo de herramientas tecnológicas para los profesionales, pero creemos que sería mucho más ágil y eficaz utilizar un tipo de apoyo más inteligente que genere planos bajos listos, cumpliendo con los requisitos del cliente. Dado lo esencial de la representación gráfica y todo el trabajo involucrado en la creación de un proyecto de construcción, se vuelve crucial el uso de inteligencia artificial para generar imágenes y automatizar este trabajo creativo. Según Anglen, J. (2023), el futuro de la arquitectura y la construcción está siendo transformado rápidamente por los avances en inteligencia artificial y automatización. Un área particularmente interesante es el diseño generativo, donde algoritmos de *IA* pueden crear infinitas variaciones y optimizaciones del proyecto de un edificio para cumplir con los parámetros deseados [51].

Con base en investigaciones recientes, ya existen sistemas inteligentes que generan imágenes de planos bajos como herramientas comerciales, como es el caso de *Market.AI*, una plataforma de software integrada con inteligencia artificial (*IA*) desarrollada para arquitectos y profesionales en el campo de la planificación de viviendas y edificios. Esta tecnología permite generar rápidamente múltiples opciones de diseño para un proyecto determinado, considerando los requisitos de generación de planos de planta (*Floor Plan*) y diseño visual de espacios (*Designer Visual*). Además, ofrece la capacidad de generar planos a través de texto y realizar modificaciones en planos existentes mediante su integración con *AutoCAD*. La tecnología cuenta con un plan de pruebas [63]. Otro ejemplo de tecnología generativa comercial en este campo es *Getfloorplan*, un sistema donde el profesional especifica requisitos en la plataforma y esta genera los planos en formato *2D* y *3D*. Es una herramienta sencilla y eficaz como soporte para la creación de planos, aunque, al ser una tecnología comercial, no ofrece la posibilidad de pruebas gratuitas [31].

Al revisar trabajos científicos, encontramos pocos estudios que exploren el uso de la inteligencia artificial para generar planos arquitectónicos, especialmente en el contexto de tener como entrada un texto detallado. Lo que se encuentra con mayor frecuencia son investigaciones que utilizan imágenes o siluetas de planos para generar otros planos, manteniendo las condiciones propias de un plano arquitectónico, como se indica en la imagen. Este enfoque se conoce como el método *Pix-to-Pix*. El artículo científico que mejor describe este proceso es *"FloorDiffusion: Diffusion model-based conditional floorplan image generation method using parameter-efficient fine-tuning and image inpainting"* [55].

Este artículo tiene como objetivo presentar una forma de generar planos mediante ajustes finos con *Stable Diffusion*, utilizando la función *Pix-to-Pix*, cuyo propósito es mejorar una imagen o realizar el relleno de espacios vacíos visibles en una imagen. En la investigación, se utiliza esta técnica para crear nuevos planos a partir de siluetas e imágenes vacías, así como de imágenes que tienen menciones y condiciones específicas para realizar los debidos rellenos. El modelo generativo detallado y comparado se muestra en la figura 4.7, donde se observa la entrada y las condiciones impuestas en cada proceso de generación. Estas condiciones se definen mediante demarcaciones coloridas que identifican

cada habitación y espacio de un simple plano arquitectónico. Todo el desarrollo se realiza utilizando el método *LoRa*, aplicando la función *Pix-to-Pix*.

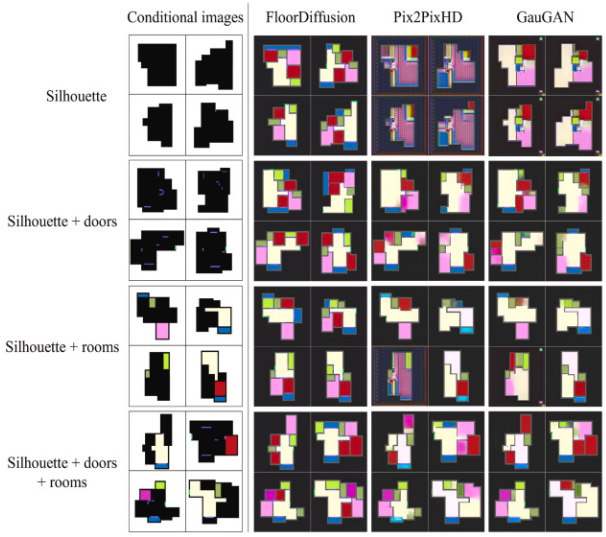


Figura 4.7: Comparación de imágenes de planos arquitectónicos generadas a partir de cuatro tipos de imágenes condicionales [55].

El artículo evidencia que el uso de *Stable Diffusion* es más adecuado para este problema generativo que la arquitectura tradicional de *GANs*, ya que es mucho más frecuente que los modelos entrenados con *GANs* generen imágenes con mucho ruido o distorsión debido a inestabilidades durante el entrenamiento. Además, las *GANs* no son capaces de generar condiciones no existentes que no hayan sido sometidas al entrenamiento, lo que requiere una gran cantidad de datos para obtener un modelo aceptable en términos de criterios condicionales. La arquitectura de difusión, en cambio, puede ofrecer mejores resultados en este sentido de relleno y mejora de condiciones generativas, ya que sus modelos contienen una enorme cantidad de datos aleatorios entrenados [55].

Muchos de los artículos consultados identifican las diversas posibilidades de generar planos arquitectónicos mediante *IA*. Sin embargo, la mayoría de las investigaciones se centran en condicionar la generación de imágenes utilizando la función *Pix-To-Pix*, que trabaja con el relleno de imágenes utilizando semántica e identificación de diferentes atributos en una imagen para procesarla y generar otra. Son pocos los estudios que tratan la función *Text-To-Pix* como un método ajustable para generar diferentes planos arquitectónicos. Sin embargo, podemos observar que muchas investigaciones mencionan esta función como un método utilizado en el contexto de generar imágenes de planos estándar, entrenados sobre modelos estándar sin la necesidad de realizar *Fine-Tuning*, lo que lo convierte en un posible apoyo creativo para arquitectos y personas comunes [81].

El artículo *.Automating Computational Design with Generative AI"* [54] presenta un intento de ajustar los modelos de difusión utilizando imágenes generadas automáticamente mediante algoritmos procedurales como conjunto de datos de entrenamiento, empleando el método *Dreambooth* para el ajuste. Sin embargo, los resultados obtenidos no fueron

tan efectivos, ya que produjeron imágenes no reconocibles o planos confusos. No obstante, mejoraron el contexto de *Stable Diffusion*, permitiendo generar imágenes basadas en patrones del modelo previamente entrenado. Los autores de la investigación sugieren que las deficiencias pueden deberse a que las imágenes no están etiquetadas adecuadamente, semánticamente, como sucede en las funciones comunes de *Pix-To-Pix*. Además, destacan la necesidad de utilizar imágenes más realistas para llevar a cabo esta tarea compleja, junto con un enfoque en la semántica y el texto descriptivo que se utilizará en el entrenamiento.

En este trabajo, utilizaremos tecnologías existentes para crear un sistema inteligente que, mediante la interpretación de texto, genere planos de casas creados a través de inteligencia artificial. Como se evidencia, utilizaremos la tecnología *Text-to-Image* para esta acción, realizando un ajuste fino sobre los modelos de *Stable Diffusion*, y considerando los problemas y deficiencias mencionados en el capítulo 4: Estado del Arte. Sin embargo, también realizaremos pruebas sobre estos mismos temas. Los detalles sobre el ajuste, el conjunto de datos utilizado y otros aspectos del desarrollo serán abordados en el capítulo 5: Técnicas y herramientas.

# 5: Técnicas y herramientas

En este capítulo presentaremos la metodología que se llevó a cabo para alcanzar los objetivos estipulados para esta investigación. Cabe recordar que la metodología descrita aquí comenzó a aplicarse durante el proceso de desarrollo de la estancia I+D+i en informática, ya comentado en capítulos anteriores. Nuestra investigación seguirá un método más experimental, ya que nuestro objetivo es evaluar la capacidad de ajustar un modelo ya entrenado y verificar su capacidad para generar planos arquitectónicos en formato 2D, de acuerdo con textos ingresados a través de prompts. La decisión de realizar una investigación más experimental surgió del proceso que hemos llevado a cabo probando diferentes métodos de ajustes finos en los modelos de Stable Diffusion.

Para el desarrollo del sistema generativo, seguimos algunos requisitos establecidos tras el establecimiento de los objetivos, las revisiones literarias y la verificación de tecnologías ya existentes que dieron origen al capítulo 4: Estado del Arte. Los requisitos establecidos son:

- Establecer una infraestructura computacional con GPU y encontrar un conjunto de imágenes de planos arquitectónicos para el desarrollo del modelo.
- Verificar los diversos tipos de métodos de Fine-Tuning de Stable Diffusion disponibles y probarlos para identificar cuál se adecuaba mejor a nuestro problema generativo.
- Crear una interfaz web integrando el sistema generativo, para realizar pruebas de manera sencilla.
- Probar el sistema generativo con métricas capaces de evaluar el sistema de manera eficiente.

En las próximas secciones de este capítulo, detallaremos cada uno de los requisitos, evidenciando las técnicas y herramientas empleadas para que pudiéramos desarrollar el sistema generativo durante el proceso de investigación y pruebas. Algunos aspectos, como el enlace al repositorio de código en línea donde se almacena el proyecto, junto

con información técnica más detallada sobre la configuración de entornos del servidor e instalación de bibliotecas específicas necesarias para la implementación, se encuentran disponibles en el manual de instalación de este proyecto, en el Apéndice B.

## 5.1. Infraestructura y Ambiente de Desarrollo del Proyecto

Para desarrollar los requisitos establecidos para el proyecto *Floorify*, tuvimos que elegir y definir una infraestructura que incluyera *GPU* (Unidad de Procesamiento Gráfico), lenguaje de programación y entorno de desarrollo, para que dicha acción pudiera ser completada. En cuanto a la infraestructura, se nos proporcionó una conexión a un servidor del grupo de investigación *ECA-SIMM*, de la Escuela de Informática de la Universidad de Valladolid, que contaba con un almacenamiento dinámico, con una *GPU NVIDIA GeForce A40* de 48 *GB*. El trabajo se realizó en un entorno compartido del uso de la *GPU* y del almacenamiento interno, en el que nuestra investigación utilizaba un total de 24 *GB* de *GPU*. El uso de la *GPU* era de gran importancia, ya que el entrenamiento de las redes neuronales que componían los modelos de *Stable Diffusion* requería el procesamiento de una gran cantidad de imágenes del tema a ser ajustado.

El lenguaje de programación elegido para el desarrollo de la investigación fue *Python*, un lenguaje de programación orientado a múltiples paradigmas [78]. Su elección se basó en la facilidad para manejar grandes cantidades de datos, además de la facilidad para manipular modelos de *Deep Learning* y estar completamente integrado con los modelos y métodos de *Stable Diffusion* a través de la biblioteca *Diffusion*. Además, decidimos trabajar con los entornos de programación Visual *Studio Code* y también se nos proporcionó por el grupo ECA-SIMM el entorno de desarrollo *Jupyter Notebook*, conocido por su fácil manipulación de todo el contenido relacionado con Inteligencia Artificial y Ciencia de Datos. Sin embargo, el entorno en el que realizamos más desarrollo y pruebas fue Visual *Studio Code* dentro de la conexión del servidor de *ECA-SIMM*, ya que la mayoría de los métodos de ajuste fino requerían que los *scripts* se ejecutaran localmente a través de archivos con extensión "*.bash*"(*Bourne Again Shell*), lo cual se detalla más en la sección de *Fine-Tuning* con *Stable Diffusion* y Sus Métodos.

## 5.2. Conjunto de Datos (*Dataset* de Imágenes)

Para el desarrollo del ajuste fino, necesitábamos un conjunto de imágenes en gran cantidad, además de que estas tuvieran frases o descripciones textuales relacionadas con cada imagen representada, para que el modelo pudiera aprender y adaptarse generando nuestras imágenes de acuerdo con su debido contexto textual entrenado. Sin embargo, fue uno de los ítems más difíciles de encontrar, ya que no había muchos conjuntos de datos de planos arquitectónicos disponibles públicamente, y rara vez encontrábamos conjuntos que contuvieran planos arquitectónicos de gran escala, relacionados con un conjunto de datos descriptivos de cada imagen. Para esto, realizamos un estudio sobre cada conjunto

disponible en internet para elegir con cuál sería más conveniente trabajar. A continuación se presentan las tablas que evidencian los conjuntos de datos disponibles y sus características:

| ***Dataset*** | ***CubiCasa5k*** | ***sudo-floor-plan-12k*** |
|---|---|---|
| **Total *samples*** | 5.000 | 12.000 |
| ***Size dataset files*** | 105 *GB* | *4GB* |
| **Etiquetas (*Tag*)** | Imágenes generadas a partir de la biblioteca *FloorPlanSVG Python*, tiene identificación, pero no tiene texto de descripción relacionado. | Los datos están etiquetados de forma predeterminada según sus índices informados en el alcance predeterminado de *HuginFace*. Tiene Descripciones relacionadas. |
| ***Data Argumentation*** | Existe la posibilidad de aplicar aumento de datos a todos los conjuntos de datos, sin embargo, algunos datos estarán en un formato más comprimido o vectorial, como es el caso de *CubiCasa5k*. | |
| **Informaciones Extras** | El conjunto de datos contiene un modelo de red neuronal que tiene como objetivo convertir una imagen de plano (dibujó) en una representación gráfica vectorial, para identificar mejor cada espacio en los planos de una casa. | Las imágenes contienen descripciones y diferentes tipos de imágenes identificativas con colores para efectuar la técnica *Pix-To-Pix*, pero las imágenes están en ángulos extraños. |
| **Enlace de la pagina web** | [56] | [71] |

Tabla 5.4: Recogido de *datasets* públicos - 1

| ***Dataset*** | ***FloorPlansV2*** | ***FloorPlanCAD*** | ***New Floorplan demo dataset*** |
|---|---|---|---|
| **Total *samples*** | 2.831 | 15.663 | 101 |
| ***Size dataset files*** | 1.13 *GB* | 6 *GB* | 8.4 *MB* |
| **Etiquetas (*Tag*)** | Los datos tienen una etiqueta específica según las clases a las que están asociados. Las clases se dividen por el número de habitaciones que hay en cada piso. | Hay un código identificativo en cada imagen. | No contiene etiquetas en las imagenes, pero todas las imágenes tiene una correlación con descripciones de texto. |
| ***Data Argumentation*** | Existe la posibilidad de aplicar aumento de datos a todos los conjuntos de datos, sin embargo, algunos datos estarán en un formato más comprimido o vectorial, como es el caso de *CubiCasa5k*. | | |
| **Extras** | | | Datos basasados en *CubiCasa5k* |
| **Enlace de la pagina web** | [50] | [60] | [76] |

Tabla 5.5: Recogido de *datasets* públicos - 2

Como podemos visualizar, cada conjunto de imágenes contenía sus ventajas y desventajas, y por eso optamos por trabajar con el conjunto de *CubiCasa5K*, ya que contenía una cantidad masiva de datos y sus imágenes eran de buena calidad y muy similares a las empleadas en el mundo real de la arquitectura de planos bajos en *2D*. Las imágenes del conjunto *CubiCasa5K* eran buenas porque fueron generadas a partir de transformaciones de imágenes de planos reales en imágenes vectoriales, lo que dejaba las imágenes en un contexto más claro y de buena calidad. También trabajamos con otros conjuntos de datos realizando entrenamientos y pruebas como el *dataset New FloorPlan demo Dataset*, que era basado en *CubiCasa5K*. Uno de los conjuntos que era muy interesante era el *Pseudo-floor-plan-12k*, ya que tenía buenas descripciones textuales e identificaciones claras de cada habitación dentro de la imagen, pero las imágenes contenían ángulos extraños que no serían muy útiles para realizar entrenamientos. Cabe recordar que para trabajar con el conjunto de *CubiCasa5K* tuvimos que crear una pequeña automatización que generara los contextos textuales para relacionar con cada imagen, formando el conjunto de datos textual para llevar a cabo el entrenamiento, combinando estos dos *embeddings* de texto e imagen. Esto y otros aspectos de los caminos de elección los explicamos con mayor claridad en el capítulo 6: Aspectos relevantes del desarrollo del proyecto.

## 5.3. *Fine-Tuning* en Modelos de *Stable Diffusion* y Sus Métodos

Después de estudiar diferentes tecnologías de generación de imágenes, decidimos que el mejor camino sería utilizar los modelos de *Stable Diffusion* para realizar un ajuste fino y así probar la capacidad de su tecnología para adaptarse a nuestro conjunto de datos. Con esto, decidimos realizar pruebas con los métodos de *Fine-Tuning* como *Dreambooth* [41], *Misto Line* [62], *LoRas* [43] y *Text-To-Image* [44]. Además, también probamos el modelo básico sin realizar ajuste fino para verificar la capacidad de la versión estándar de *Stable Diffusion* en generar planos arquitectónicos a partir de texto. Uno de los métodos de ajuste fino que incluimos en la lista sin realizar un estudio previo fue el *Misto Line* [62], creado y ajustado por la empresa *The Misto.AI*, miembros colaboradores de la comunidad de desarrollo *Hugging Face*. La decisión de incluirlo se tomó después de visualizar buenas evaluaciones sobre su método de ajuste fino en modelos de difusión y nivel de detalles que tenían vuestras imágenes generadas y para tener una referencia del funcionamiento entre el *text-to-image* y el *pix-to-pix*.

Las pruebas con cada método se realizaron utilizando una variedad de versiones de modelos de *Stable Diffusion*, siguiendo los consejos proporcionados por la propia documentación de *Diffusion*, que es la biblioteca de *Python* proporcionada por *Hugging Face*. Para los modelos de *Dreambooth* utilizamos el modelo "*stable-diffusion-v1-4*"[17] y para *LoRas* y *Text-To-Image* utilizamos el "*stable-diffusion-v1-5*"[18]. El *Misto Line* utilizaba por defecto el modelo "*stable-diffusion-xl-base-1.0*"[3]. Cada modelo tenía su ventaja en términos de mejor rendimiento y mejor comprensión del contexto entre las *embeddings* para generar imágenes. El único método que utilizaba un modelo diferente del estándar era *Misto Line*, que utilizaba el modelo más nuevo *SDXL* de *Stability AI*. Este modelo tenía mucha más capacidad de tener detalles en sus imágenes generadas y un mejor desempeño en la comprensión del contexto entre las *embeddings*, pero su estructura requería infraestructuras más potentes para realizar los ajustes finos. Los desafíos encontrados con el modelo utilizado en *Misto Line* y los otros métodos se detallan en el capítulo 6: Aspectos relevantes del desarrollo del proyecto.

## 5.4. Métricas de Evaluación de Modelo Generativo

Como ya se mencionó en el capítulo 4: Estado del Arte, es indispensable que el sistema generativo cuente con métodos de evaluación para verificar sus niveles de calidad tanto a nivel de imagen como de contexto entre *embeddings*. Por ello, decidimos utilizar dos métricas algorítmicas que se aplicarán al modelo después de los ajustes finos: *CLIP Score* (*Contrastive Language-Image Pretraining*) [40] y *Fréchet Inception Distance* [30]. Utilizaremos el *CLIP Score* en un contexto más evaluativo entre los *embeddings* para verificar en qué medida las imágenes generadas coinciden con el texto ingresado por el usuario. Esta métrica, creada por *OpenAI*, es adecuada para realizar la verificación contextual, sin embargo, dado que es bastante nueva, tiene sus limitaciones, ya que se trata

de una *IA* entrenada con una cantidad masiva de datos, pero no infinitos y totalmente actuales, ya que siempre está en actualización. El objetivo del *CLIP Score* será presentar valores altos o bajos de acuerdo con su evaluación. Siempre que el *score* sea alto o próximo a 1, representará un buen contexto semántico y de características entre los *embeddings*, mientras que un valor bajo indicará lo contrario, es decir, un contexto deficiente.

Para evaluar el contexto de calidad de la imagen en su totalidad a nivel de textura y características, utilizaremos el tradicional *FID*, que es una métrica orientada a la evaluación de características, indicando el nivel de las imágenes según su realismo y calidad de características. Dado que sus medidas se basan en el cálculo de la distancia entre imágenes de prueba, el objetivo será verificar que cuanto menor sea el valor del *FID*, mejor será la calidad de las imágenes, ya que se estará acercando al mismo nivel que las imágenes en cuestión en términos de características, o verificar si el valor es muy alto, demostrando baja cualidad en comparación con las imágenes de prueba [30]. En esta investigación, no utilizamos métricas de observación humana, ya que, como se mencionó en el capítulo 4: Estado del Arte, pueden introducir mucha subjetividad, además de que se utilizan más como métricas complementarias y no principales, como es el caso de las presentadas anteriormente.

# 6: Aspectos relevantes del desarrollo del proyecto

En este capítulo se presenta el desarrollo del modelo generativo, junto con los diversos experimentos realizados para validar la eficacia de los ajustes de modelos de *Stable Diffusion* en la generación de imágenes de planos arquitectónicos en formato *2D*. Cabe recordar que, para evaluar los modelos generativos ajustados, se llevarán a cabo pruebas sobre las imágenes generadas a partir de sus condiciones, empleando *Clip Score* y *FID (Fréchet Inception Distance)*.

Todos los *scripts* originales y el desarrollo detallado en esta sección se encuentran disponibles en el repositorio público, cuyo enlace se proporciona en el apéndice B de este trabajo.

## 6.1. Pruebas Mediante Modelo *Stable Diffusion* Estándar

Para iniciar la etapa de desarrollo, realizamos una serie de pruebas con el modelo estándar de *Stable Diffusion* con el objetivo de evaluar las imágenes generadas a partir de *prompts* de texto que describían las características deseadas en un plano arquitectónico [4]. A continuación, se presentan algunos ejemplos de los resultados obtenidos.

```
import requests

response = requests.post(
    f"https://api.stability.ai/v2beta/stable-image/generate/core",
    headers={
        "authorization": f"sk-
   rkyXQ2L0sm6s1uRG6zIRQdYBTNKTzLGy5zSgRIed8ELqXdFM",
        "accept": "image/*"
    },
    files={"none": ''},
    data={
```

```
11          "prompt": "Floor Plan 2D with 2 bedrooms, 1 bathroom and living
    room",
12          "output_format": "webp",
13      },)
```

**Fragmento de código** 6.1: Ejemplo de Generar Planos Arquitectonicos en Script Padron de Stable Diffusion Free

En el ejemplo mostrado en el fragmento de código 6.1, se presenta el código básico utilizado para las pruebas con el *script* estándar de *Stable Diffusion*, sin la aplicación de ningún ajuste fino. Las pruebas fueron realizadas a partir de dos generaciones de planos arquitectónicos, condicionados por las siguientes entradas: *House Plan 2D, One Bathroom and One Bedroom y House Plan 2D*. Las imágenes generadas a partir de estas pruebas pueden visualizarse en los ejemplos de la figura 6.8 y la figura 6.9.

Figura 6.8: Imagen generada con el modelo sin ajustes finos (*Stable Diffusion*) - 1

Figura 6.9: Imagen generada con el modelo sin ajustes finos (*Stable Diffusion*) - 2

Ambas imágenes presentan una buena calidad en términos de resolución, y muchas de las condiciones establecidas en el *prompt* de texto se cumplen. No obstante, observamos que era necesario generar más de cuatro imágenes antes de obtener resultados que comenzaran a satisfacer el mínimo de condiciones estipuladas. Es decir, el modelo requería múltiples intentos para mejorar la coherencia con el *prompt* y captar con mayor precisión el contexto del plano arquitectónico. La principal dificultad en la generación de imágenes radicaba en que el modelo no lograba producir imágenes en tonos de gris y, por defecto, tendía a generar representaciones en *3D* en lugar de planos bidimensionales. Para evaluar la relación entre el texto y las imágenes generadas, sometimos los resultados a pruebas algorítmicas

con *CLIP Score*. Decidimos no aplicar *FID* en esta etapa inicial, ya que para ello sería necesario contar con un conjunto de referencia bien definido para validar las características de las imágenes. En esta primera prueba, nuestro objetivo fue únicamente analizar la coherencia semántica entre los *embeddings* de texto e imagen generados a partir del *prompt* condicionado.

```python
from transformers import CLIPProcessor, CLIPModel
from PIL import Image
import torch

# Load the CLIP model and processor
model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")

# Load an image and a phrase
image = Image.open("/")
texts = ["Floor Plan 2D", "Floor Plan 2D with 2 bedrooms, 1 bathroom and living room"]

# Preprocess the image and the text
inputs = processor(text=texts[0], images=image, return_tensors="pt", padding=True)

# Pass the inputs through the model
outputs = model(**inputs)

# Extract the embeddings from the image and text
image_embeddings = outputs.image_embeds
text_embeddings = outputs.text_embeds

# Compute the similarity (cosine similarity)
similarity = torch.nn.functional.cosine_similarity(image_embeddings, text_embeddings)

# Display the similarity score
print("CLIP Score Value: " + str(similarity))
```

**Fragmento de código** 6.2: Script de aplicación de métricas de CLIP SCORE

El *script* presentado en la referencia 6.2 ha sido desarrollado y empleado para la aplicación de la métrica *CLIP (Contrastive Language-Image Pretraining)* de *OpenAI*, permitiendo calcular la similitud entre una imagen y un texto representados como *embeddings*, tal como se explicó previamente en este estudio en el capítulo 5: Técnicas y herramientas, sección 5.4. Su propósito es evaluar la correspondencia semántica entre las imágenes generadas y sus descripciones textuales. Para ello, el *script* carga el modelo pre-entrenado *clip-vit-base-patch32* [68] junto con su procesador, abre una imagen almacenada en disco y define una frase descriptiva, en este caso, *"Floor Plan 2D", "Floor Plan 2D with 2 bedrooms, 1 bathroom and living room"*. A continuación, preprocesa la imagen y el texto mediante el procesador de CLIP y los convierte en tensores de PyTorch, los cuales son procesados por el modelo para generar *embeddings* tanto de la imagen como del texto. Para cuantificar

su relación semántica, calcula la similitud del coseno entre los *embeddings* obtenidos y, finalmente, imprime el valor del *CLIP Score*, indicando el grado de coherencia entre la imagen generada y la descripción textual proporcionada. Este *script* será fundamental para la evaluación de futuros ajustes finos (*fine-tuning*) y desempeñará un papel clave en la validación de modelos optimizados en este estudio.

Las pruebas realizadas con los datos del *CLIP Score* se detallan en el capítulo 7: Discusión de Resultados. En las siguientes secciones, se presentará una descripción detallada de cada uno de los ajustes finos (*fine-tuning*) realizados, empleando distintos métodos de *Stable Diffusion*.

## 6.2. Transformaciones de Los Conjuntos de Datos Elegidos

Después de realizar un análisis detallado de los conjuntos de datos en el capítulo 5: Técnicas y herramientas, fue posible identificar tanto la variedad de opciones disponibles como sus respectivas limitaciones. Tras esta evaluación, se optó por trabajar con *CubiCasa5k*, ya que este conjunto de datos presentaba imágenes con una resolución adecuada, además de incluir descripciones textuales o simbólicas de los espacios representados en cada imagen. Sin embargo, se observó que las imágenes de mayor calidad dentro del conjunto estaban almacenadas en formatos vectoriales, como *.SVG*, debido a que *CubiCasa5k* empleaba una inteligencia artificial para convertir imágenes de planos arquitectónicos convencionales en representaciones vectoriales. Este formato, aunque ventajoso para ciertos análisis, no era el más adecuado para el propósito de este estudio. Por esta razón, se llevaron a cabo transformaciones en el conjunto de datos, con el objetivo de adecuarlo a los requisitos del modelo utilizado. Se optó por convertir todas las imágenes al formato *.JPG*, utilizando el lenguaje de programación *Python* junto con la biblioteca de procesamiento de imágenes *PIL (Python Imaging Library)*. Esta conversión permitió mantener la calidad visual necesaria para el entrenamiento del modelo, al tiempo que garantizaba la compatibilidad con los métodos de ajuste fino y generación de imágenes empleados en este trabajo.

Con este procedimiento, ya contábamos con las imágenes adecuadas para el entrenamiento. Para optimizar el proceso y facilitar la convergencia del modelo, decidimos trabajar exclusivamente con imágenes de planos arquitectónicos que representaban una única planta, lo que permitió simplificar el aprendizaje y filtrar la gran cantidad de datos disponibles. Como resultado, el conjunto de datos final utilizado para el entrenamiento quedó reducido a un total de 450 imágenes. Además, uno de nuestros objetivos era establecer una relación entre las imágenes y descripciones textuales mediante *embeddings*, lo que requería que cada imagen tuviera una descripción asociada. Sin embargo, *CubiCasa5k* no proporcionaba descripciones textuales detalladas de cada plano, sino que contenía información en forma de anotaciones simbólicas dentro de las propias imágenes. Para superar esta limitación, decidimos generar frases estandarizadas basadas en estos símbolos, permitiendo así establecer una correspondencia clara entre cada imagen y su descripción. Para la generación de estas descripciones, seguimos el estándar recomendado por *Hugging Face*, que sugiere

estructurar los datos en un archivo *.json* [26]. En este archivo, cada imagen es identificada mediante su nombre de archivo como clave, y su descripción se asocia como valor. Además, este archivo debía ser almacenado en la misma carpeta que las imágenes de entrenamiento para asegurar su correcta vinculación durante el ajuste fino. Con el fin de automatizar este proceso, utilizamos *Python* y la biblioteca de reconocimiento óptico de caracteres *paddleocr* [10], la cual permitió extraer y leer automáticamente los textos presentes en las imágenes, generando así las descripciones correspondientes sin intervención manual. Esta automatización queda reflejada en los fragmentos de código 6.3 y 6.4.

Finalmente, para garantizar la compatibilidad con los métodos de ajuste fino de *Stable Diffusion*, cambiamos el tamaño de todas las imágenes a una resolución de *512x512* píxeles, siguiendo las especificaciones recomendadas en la documentación oficial de los modelos de ajuste fino. En la continuación enseñamos el código para las transformaciones y creaciones del *dataset*:

```
from paddleocr import PaddleOCR
import cv2
import os
import json

#Script that performs text recognition on filtered images from CubiCasa5k to define the Json metadata for FineTuning.

#The script also creates json identification images and their respective IDs and texts detailing the architectural plans.

def detect_h_and_others(image_path):

        # Count occurrences
        habitacion_solo_count = detected_texts.count('H')
        habitacion2_solo_count = detected_texts.count('MH')
        cozina_solo_count = detected_texts.count('K')
        bano_solo_count = detected_texts.count('KPH')
        bano2_solo_count = detected_texts.count('WC')
        Salon_solo_count = detected_texts.count('OH')
        Undefined = detected_texts.count('UNDEFINED')

#{.........}

        # Create the result message
        result_message = (
            f"The text contains the following words:\n\n{detected_texts_summary}\n\n"
            f"'Bedroom' detected alone in the image {habitacion_solo_count + habitacion2_solo_count} time(s).\n\n"
            f"'Kitchen' detected alone in the image {cozina_solo_count} time(s).\n\n"
            f"'Bathroom' detected alone in the image {bano_solo_count + bano2_solo_count} time(s).\n\n"
            f"'Living room' detected alone in the image {Salon_solo_count} time(s).\n\n")
```

```
30 #{.........}
```

**Fragmento de código** 6.3: Script de detección de texto y creación de frases

```
1 {
2     "file_name": "Imagen_id_10796.jpg", "text": "Floor Plan 2D, 1
      Bedroom, 1 Kitchen, 1 Bathroom, 1 Living room"
3 },
4 {
5     "file_name": "Imagen_id_10004.jpg", "text": "Floor Plan 2D, 1
      Bedroom, 1 Kitchen, 1 Bathroom, 1 Living room"
6 },
7 {
8     "file_name": "Imagen_id_1072.jpg", "text": "Floor Plan 2D, 2
     Bedroom, 1 Kitchen, 1 Bathroom, 1 Living room"
9 },
```

**Fragmento de código** 6.4: Archivo JSON Generado Para el Entrenamiento

Otro conjunto de datos empleado en nuestros experimentos fue *New Floor Plan*, el cual contenía imágenes derivadas de *CubiCasa5k* junto con un conjunto de frases generadas de manera personalizada para cada imagen. Este *dataset* se encuentra disponible públicamente y fue utilizado exclusivamente en los entrenamientos realizados con el método de ajuste fino *LoRAs*. La decisión de utilizar *New Floor Plan* en este contexto se basó en dos factores clave, en primer lugar, el método *LoRAs* no requiere un conjunto de datos tan extenso como otros métodos de ajuste fino, lo que hacía viable su aplicación sobre este *dataset* en segundo lugar, queríamos evaluar el rendimiento del ajuste fino en imágenes sin transformaciones significativas, es decir, en las imágenes originales de *CubiCasa5k*, con el fin de analizar su impacto en la generación de planos arquitectónicos. La única modificación aplicada a *New Floor Plan* fue el cambio de tamaño de todas las imágenes a *512x512* píxeles, siguiendo la recomendación establecida en la documentación del método *LoRAs* para garantizar una correcta entrada de datos en el entrenamiento del modelo.

Para realizar los entrenamientos, utilizamos el conjunto de datos completo para el entrenamiento, sin dividirlo en subconjuntos de entrenamiento, validación y prueba. Esto se debe a que los modelos generativos, como *Stable Diffusion*, no emplean conjuntos de validación para detectar *overfitting* ni para ajustar hiperparámetros, ni tampoco utilizan conjuntos de prueba. En su lugar, la evaluación del modelo se basa en métricas específicas y aleatorias, ya sean algorítmicas o mediante evaluación visual humana, dado que se trata de un modelo generativo, estas métricas permiten medir la calidad de las imágenes generadas sin necesidad de recurrir a los enfoques tradicionales de validación y prueba [25].

## 6.3. *Fine-Tuning* de *Stable Diffusion* con el Método *Dreambooth* y *LoRas*

Tal como se demostró en las secciones anteriores, *DreamBooth* fue el primer método de ajuste fino presentado para la personalización de modelos de *Stable Diffusion*. Su versatilidad y facilidad de uso han sido ampliamente reconocidas, ya que permite realizar ajustes con un número reducido de imágenes sobre un tema específico, empleando una frase clave para iniciar el entrenamiento del modelo de difusión. Para utilizar este modelo, clonamos la biblioteca *Diffusion*, conforme a las indicaciones, dado que incluye todos los algoritmos necesarios para la realización de ajustes finos de manera local. Una de las características imprescindibles de esta biblioteca es que todos los métodos deben configurarse y ejecutarse mediante un archivo "*.bash*", lo que posibilita la realización del ajuste fino directamente en la *GPU* y optimiza su desempeño dentro de la arquitectura computacional.

```
import subprocess
import os
from PIL import Image

#{.........}

input_directory = "./"
output_directory = "./"


resize_images(input_directory, output_directory, size=(512, 512))

command = [
        "accelerate", "launch", "./diffusers/examples/dreambooth/
    train_dreambooth.py",
        "--pretrained_model_name_or_path=CompVis/stable-diffusion-v1-4",
        "--instance_data_dir=./",
        "--output_dir=./Dreambooth/Model/My_Model_Trained",
        "--instance_prompt=Floor Plan 2D",
        "--resolution=512",
        "--train_batch_size=1",
        "--gradient_accumulation_steps=1",
        "--learning_rate=5e-6",
        "--lr_scheduler=constant",
        "--lr_warmup_steps=0",
        "--max_train_steps=300",
        "--push_to_hub"
    ]

subprocess.run(command)
```

**Fragmento de código** 6.5: Script de uso del metodo dreambooth

Tal como se muestra en el fragmento de código, optamos por *Python* para configurar y ejecutar el ajuste fino de manera local, ya que simplificaba la configuración y ofrecía

los mismos resultados que un *script* "*.bash*". El conjunto de datos empleado para este entrenamiento fue *New Floor Plan*, del cual seleccionamos seis imágenes de planos. Según la arquitectura del ajuste fino, entre cinco y siete imágenes son suficientes para realizar los ajustes. No obstante, encontramos una limitación: no era posible entrenar las imágenes con diferentes frases, sino únicamente con una predeterminada, lo que restringía la versatilidad del ajuste fino y reducía las posibilidades de generar distintos planos a partir de texto. La resolución de las imágenes empleadas fue la estándar establecida, tal como se mencionó anteriormente en este capítulo. En cuanto a los demás parámetros, como la tasa de aprendizaje y los tamaños de lote, utilizamos los valores predeterminados proporcionados por los ejemplos de la propia biblioteca *Diffusion* de *Hugging Face.*


Figura 6.10: Imágenes Generadas Después de Ajuste Fino con *Dreambooth*

Los experimentos con *DreamBooth* no incluyeron una amplia variedad de frases de inferencia ni de parámetros de ajuste fino, ya que anticipamos que este método no lograría el objetivo de nuestra investigación: alcanzar una alta aleatoriedad en frases e imágenes. No obstante, decidimos evaluarlo para comprobar la efectividad de los ajustes finos y analizar cómo, a pesar de su simplicidad, podría generar planos arquitectónicos. Como se observa en la figura 6.10, las imágenes resultantes corresponden a planos arquitectónicos en *2D*, tal como se esperaba, sin embargo, muchas de ellas presentan inconsistencias, como colores sobresaturados. Además, las pruebas se limitaron a una única frase como parámetro, lo que restringió la generación de imágenes bajo distintas condiciones a partir de *prompts* de texto.

Tras los experimentos realizados con *DreamBooth*, decidimos emplear el otro método de ajuste fino de *Stable Diffusion*, denominado *LoRAs*. Este método compartía el mismo principio que el previamente probado en esta investigación, pero presentaba ventajas significativas: era más ligero de aplicar y permitía la incorporación de un mayor número de imágenes, lo que favorecía un aprendizaje más eficiente y resultados más precisos. No obstante, la principal limitación de *LoRAs* residía en la imposibilidad de utilizar múltiples frases para entrenar un gran conjunto de imágenes, lo que restringía su adaptabilidad a los requisitos específicos de nuestra investigación. A pesar de esta limitación contextual, decidimos evaluarlo con el propósito de analizar su desempeño en la generación de imágenes, dado que su método de ajuste fino era similar al de *DreamBooth*, pero más rápido y liviano de implementar. A continuación, se presentan el *script* de ajuste utilizado y resultados obtenidos:

```
1 import subprocess
2 import os
3 import wandb
```

```
4
5  #{.........}
6
7  # Define the training command
8  command_train = [
9      "accelerate", "launch", "./diffusers/examples/dreambooth/
       train_dreambooth_lora.py",
10     "--pretrained_model_name_or_path=runwayml/stable-diffusion-v1-5",
11     "--instance_data_dir=./Dataset",
12     "--output_dir=./Model_Test",
13     "--instance_prompt=floor plan 2D",
14     "--resolution=512",
15     "--train_batch_size=3",
16     "--gradient_accumulation_steps=3",
17     "--checkpointing_steps=100",
18     "--learning_rate=5e-5",
19     "--report_to=wandb",
20     "--lr_scheduler=constant",
21     "--lr_warmup_steps=0",
22     "--num_train_epochs=100",
23     "--max_train_steps=1000",
24     "--validation_prompt=floor plan 2D",
25     "--validation_epochs=50",
26     "--seed=0",
27     "--push_to_hub",
28     "--mixed_precision=fp16",
29     "--gradient_checkpointing"
30 ]
31
32 # Run the training command
33 subprocess.run(command_train)
```

**Fragmento de código** 6.6: Script de uso del metodo LoRas

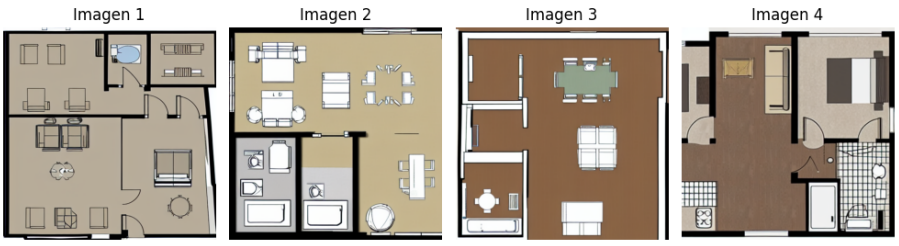


Figura 6.11: Experimento 1: imágenes generadas de pruebas mediante el método LoRas

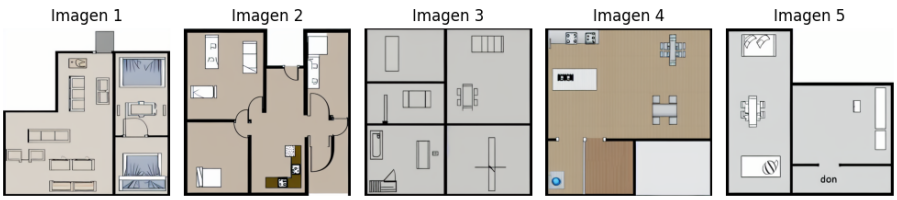


Figura 6.12: Experimento 2: imágenes de prueba generadas mediante el método LoRas

Las imágenes resultantes del *fine-tuning* con *LoRAs* mostraron resultados interesantes, aunque también inesperados. Muchas carecían de contexto y presentaban errores, como borrosidad y falta de coherencia con la solicitud inicial, que consistía en generar planos arquitectónicos básicos. De manera sorprendente, la mayoría de las imágenes generadas eran a color, a pesar de que las imágenes de entrenamiento estaban en escala de grises. Esto sugiere una capacidad intrínseca de *LoRAs* para completar y reinterpretar las imágenes durante el proceso de generación tras el ajuste fino.

El conjunto de datos utilizado fue *CubiCasa5k*, del cual se seleccionaron 100 muestras para el entrenamiento con el método *LoRas*. En total, se llevaron a cabo 35 pruebas, en las que se modificaron diversos parámetros del ajuste fino. Sin embargo, para este documento, los resultados se redujeron a solo dos pruebas, identificadas como las que presentaron la mayor evolución en el proceso de ajuste. Las frases utilizadas tanto para el entrenamiento como para la inferencia fueron de contexto limitado y poco detallado, debido a la incapacidad de ambos métodos, *DreamBooth* y *LoRAs*, de trabajar con múltiples *embeddings* relacionados.

Ante esta situación, se identificó la necesidad de un método que, por defecto, permita relacionar múltiples parámetros durante el entrenamiento. Este aspecto será abordado en la próxima sección de este capítulo (6.5).

## 6.4. *Fine-Tuning* de *Stable Diffusion* con el Método *Misto Line*

El método *Misto Line* fue elegido para hacer el ajuste fino de nuestro problema debido a su buena reputación en la comunidad de *Hugging Face*. Este método empleaba el modelo *SDXL* de *Stable Diffusion*, que no era el clásico utilizado para métodos de ajuste fino. Sus ventajas en rapidez y gran calidad de imágenes generadas llamaban mucho la atención. Sin embargo, después de estudiar con él más a fondo, nos dimos cuenta de que no era el método más adecuado para ajustar *embeddings* del tipo *text-to-image*, sino más bien para *pix-to-pix*, ya que es mejor para condicionamientos mediante líneas de boceto sobre una imagen. Los testes para los métodos del *Mixto Line* no fueron muy profundos, ya que no era un método muy específico para nuestro problema. A continuación, destacamos las imágenes generadas y sus respectivas evaluaciones utilizando solamente el *Clip Score*.

El fragmento de código a continuación nos muestra que podíamos enviar un *prompt* para detallar una condición de lo que queríamos visualizar sobre la imagen. Sin embargo, observamos en la primera prueba que la imagen generada seguiría más las expresiones de la figura enviada como ejemplo a seguir para generar una imagen. Este método simple es capaz de ajustarse a una sola imagen, pero siempre es necesario enviar una imagen de base para que pueda tener bocetos que condicionen su generación de imágenes. Además, dado que el modelo *SDXL* es muy detallista, podemos observar que solo sería posible generar imágenes de planos en *3D*, con tonalidades *RGB*.

```
from diffusers import ControlNetModel,
    StableDiffusionXLControlNetPipeline, AutoencoderKL
from diffusers.utils import load_image
from PIL import Image
import torch
import numpy as np
import cv2

prompt = "Floor Plan 2D, Many Details, Image of white background"
negative_prompt = 'low quality, bad quality, sketches'

image = load_image(f"teste5.png")

controlnet_conditioning_scale = 0.5

controlnet = ControlNetModel.from_pretrained(
    "TheMistoAI/MistoLine",
    torch_dtype=torch.float16,
    variant="fp16",
)
vae = AutoencoderKL.from_pretrained("madebyollin/sdxl-vae-fp16-fix",
    torch_dtype=torch.float16)
pipe = StableDiffusionXLControlNetPipeline.from_pretrained(
    "stabilityai/stable-diffusion-xl-base-1.0",
    controlnet=controlnet,
    vae=vae,
    torch_dtype=torch.float16,
)
pipe.enable_model_cpu_offload()

image = np.array(image)
image = cv2.Canny(image, 100, 200)
image = image[:, :, None]
image = np.concatenate([image, image, image], axis=2)
image = Image.fromarray(image)

images = pipe(
    prompt, negative_prompt=negative_prompt, image=image,
    controlnet_conditioning_scale=controlnet_conditioning_scale,
    ).images

```

```
39 images[0].save(f"hug_lab.png")
```

**Fragmento de código** 6.7: Ejemplo de generar planos arquitectonicos con el metodo Misto Line

En la figura 6.13, podemos visualizar la prueba generada con el método de *Misto Line*, la cual nos muestra claramente como el modelo *SDXL* tiene la capacidad de detallar las imágenes entrenadas para hacer sus inferencias, sin embargo vemos que las imágenes nos traen un contexto más *3D*, que no nos interesa traer en este punto de esta investigación. Para generar esta imagen, utilizamos una de las imágenes básicas, proporcionada por el conjunto de datos *CubiCasa5k*.

Figura 6.13: Imagen generada con el método *Misto Line*

| **Figuras de Teste** | **Frase de Inferencias** | **Evaluación *CLIP Score*** |
|---|---|---|
| Figura: 6.13 | *Floor Plan 2D, Many Details, Image of white background* | 0.2765 |

Tabla 6.6: Evaluación con *Clip Score* de la imagen generada con *Misto Line*

No realizamos muchas pruebas con *Misto Line* porque, al igual que *Dreambooth*, tenía un objetivo que no correspondía a nuestra meta, que era efectuar un entrenamiento masivo sobre un ajuste fino, hasta obtener imagenes igual planos bajos *2D*. Además, la tecnologia produce muchos más resultados cuando se condiciona con bocetos en una imagen, permitiendo así realizar rellenos e inserciones en una imagen con el método *pix-to-pix*. Sin embargo, pudimos visualizar cómo funcionaba su tecnología, además de realizar inferencias importantes y observar el nivel de detalles que se insertan, junto con un toque más artístico. Una de las cosas que no fue posible lograr fue que generara imágenes en tonos de gris y con un aspecto más de plano bajo, y si solamente como una imagen con aspecto mas *3D* y artístico, lo que nos llevó a descartar su uso para este proyecto.

## 6.5. *Fine-Tuning* de *Stable Diffusion* con el Método *Text_To_Image*

Después de realizar experimentos con los métodos de *Fine-Tuning* de *Dreabooth*, *LoRas* y *Misto Line*, pudimos evidenciar que los modelos de difusión eran capaces de ajustarse a las imágenes de nuestros diferentes conjuntos de datos. Sin embargo, un problema que enfrentábamos era la imposibilidad de relacionar múltiples *embeddings*, es decir, diversas imágenes relacionadas con un tipo o múltiples frases. Solo podíamos tener una frase estándar para todo un ajuste fino. Con este problema, nos dimos cuenta de que era necesario utilizar un método que relacionara estos *embeddings* de manera múltiple y eficiente. Fue entonces que encontramos un método que aún está en experimentación denominado *Text-to-Image* de la biblioteca *Diffusion*, el cual tiene el poder de realizar ajuste fino en modelos *Stable Diffusion* mediante diversas imágenes relacionadas con diversas frases. No obstante, este método nos requería una serie de imágenes adicionales en comparación con los otros métodos para realizar los entrenamientos. Dado que también teníamos dificultades para encontrar las imágenes necesarias, fue necesario utilizar las imágenes del *dataset CubiCasa5k* transformado, como se detalla en la sección 6.2 de este capítulo. Con las transformaciones del *dataset*, teníamos un total de 450 imágenes para entrenar, además de 450 frases en un archivo *.json* que fue ejemplificado en la documentación de *Diffusion* para ser utilizado en el método de ajuste fino.

Como el método cumplía con lo que considerábamos más adecuado para nuestra investigación, decidimos entrenar diferentes modelos con distintos parámetros para observar cómo cada modelo podría ajustarse a nuestro conjunto de datos y generar las imágenes, teniendo en cuenta el contexto textual de cada experimento. A continuación, detallamos los experimentos realizados:

```
command_train  = [
    "accelerate", "launch", "/home/data/giovan/testes/Dreambooth/Model/
   diffusers/examples/text_to_image/train_text_to_image.py",
    "--pretrained_model_name_or_path", model_name,
    "--train_data_dir", train_dir,
    "--use_ema",
    "--resolution", "512", "--center_crop", "--random_flip",
    "--train_batch_size", "4",
    "--gradient_accumulation_steps", "4",
    "--gradient_checkpointing",
    "--mixed_precision", "fp16",
    "--max_train_steps", "1500",
    "--learning_rate", "5e-6",
    "--max_grad_norm", "4",
    "--lr_scheduler", "constant", "--lr_warmup_steps", "0",
    "--output_dir", output_dir,
    "--logging_dir", "output_log1"
]
```

**Fragmento de código** 6.8: Ejemplo de entrenamiento de ajuste fino Text_To_Image

El entrenamiento, como se muestra en el fragmento 6.8, nos permite visualizar que en este primer modelo utilizamos el estándar especificado por la biblioteca *Diffusion*, lo cual generó un modelo generativo con el que realizamos dos diferentes inferencias que resultaron en las imágenes descritas en las figuras 6.14 y 6.15. Podemos observar en las figuras que, en cuanto a la calidad de imagen, los planos generados en formato *.png* nos presentan una buena calidad de píxeles, especialmente por haber utilizado las imágenes de *CubiCasa5k*, que contaban con buenos ejemplos transformados. En el contexto de la calidad, podemos notar que las imágenes tienen una mejor resolución, sin embargo, presentan descripciones erróneas en las imágenes y sin contexto, lo que podría indicarnos que no deberíamos haber utilizado las imágenes del conjunto con la descripción textual (anotaciones) adentro de las imágenes de cada habitación desde el inicio de los entrenamientos.

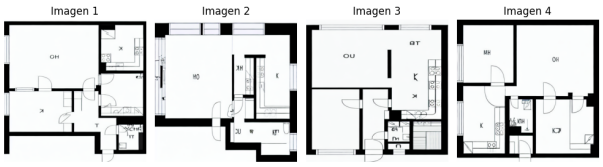


Figura 6.14: Experimento 1, Modelo1, *Prompt* 1: imágenes generadas de pruebas mediante el método *Text-to-Image*

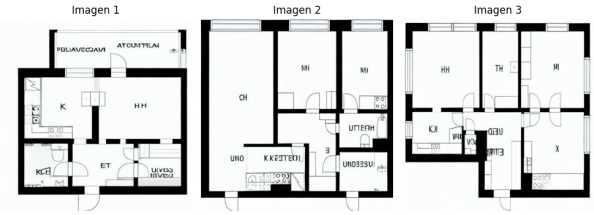


Figura 6.15: Experimento 2, Modelo1, *Prompt* 2: imágenes generadas de pruebas mediante el método *Text-to-Image*

En la continuación es demostrados los experimentos realizados con cambio en los parámetros del ajuste fino del método y nuevas inferencias realizadas:

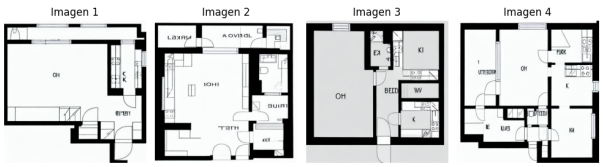


Figura 6.16: Experimento 1, Modelo2, *Prompt* 1: imágenes generadas de pruebas mediante el método *Text-toImage*


Figura 6.17: Experimento 2, Modelo2, *Prompt* 2: imágenes generadas de pruebas mediante el método *Text-to-Image*

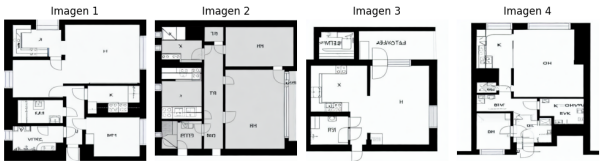


Figura 6.18: Experimento 1, Modelo3, *Prompt* 1: imágenes generadas de pruebas mediante el método *Text-to-Image*

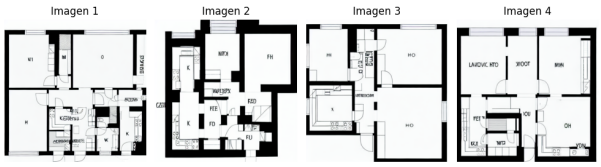


Figura 6.19: Experimento 2, Modelo3, *Prompt* 2: imágenes generadas de pruebas mediante el método *Text-to-Image*

Los cambios realizados en los entrenamientos de los modelos que presentaron los resultados anteriores incluyeron modificaciones en la tasa de aprendizaje, el tamaño del *batch* utilizado en cada entrenamiento y el ajuste del número de *steps* de entrenamiento para controlar los niveles de aprendizaje del modelo. Los resultados de estos entrenamientos se presentan en el Capítulo 7: Discusión de Resultados.

Una de las dificultades que encontramos al realizar los ajustes finos fueron las limitaciones con la infraestructura computacional, ya que muchas veces el ajuste generaba modelos muy grandes y la infraestructura no contaba con espacio de almacenamiento suficiente para soportarlos, o la memoria de la *GPU* superaba su límite, debido al procesamiento de una gran cantidad de algoritmos de difusión, a medida que aumentábamos la cantidad de datos utilizados para mejorar el ajuste fino.

Para un análisis más preciso de las pruebas realizadas, utilizamos el algoritmo *CLIP Score*, al igual que en los métodos experimentales anteriores, y también aplicamos la métrica *FID*, dado que este algoritmo parecía cumplir con los objetivos de nuestra investigación. Su implementación nos permitiría evaluar con mayor profundidad la calidad de las imágenes generadas. A continuación, presentaremos el proceso de implementación de ambas métricas, con el fin de evaluar la coherencia contextual entre los *embeddings* textuales y las características de cualidad de los modelos ajustados mediante el método *text-to-image*.

```
from transformers import CLIPProcessor, CLIPModel
from PIL import Image
import torch
import os

# List of image files for Test 01 - LoRa
Test_01_LoRa_Files = ["output_Teste_1.png", "output_Teste_2.png", "
    output_Teste_3.png", "output_Teste_4.png"]
Test_01_LoRa_Path = ..\Text_to_Image_Testes\Model_3\Prompt_1"

# Generate full file paths
Full_Files_Test_1 = [os.path.join(Test_01_LoRa_Path, file) for file in
    Test_01_LoRa_Files]
Image_Counter_Test_1 = 0

#{.........}

# Loop through each file
for Open_File in Full_Files_Test_1:

    Image_Counter_Test_1 += 1

    image = Image.open(Open_File)

    texts1 = ["Floor Plan 2D, 1 bedroom, 1 kitchen, 1 bathroom, 1 living
    room"]

    # Load the CLIP model and processor
    model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32")
    processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-
   patch32")

    # Preprocess the image and text
    inputs = processor(text=texts1, images=image, return_tensors="pt",
   padding=True)

    # Pass the inputs through the model
    outputs = model(**inputs)

    # Extract image and text embeddings
    image_embeddings = outputs.image_embeds
    text_embeddings = outputs.text_embeds

    # Compute similarity (cosine similarity)
    similarity = torch.nn.functional.cosine_similarity(image_embeddings,
    text_embeddings)

    # Display similarity score
    print("CLIP Score Value for Test 1, Figure " + str(
   Image_Counter_Test_1) + ": " + str(similarity))

```

```
45 #{.........}
```

**Fragmento de código** 6.9: Evaluacion con CLIP Score

Aplicamos el *CLIP Score* de la misma manera que se demostró en la sesión 6.3. Sin embargo, la diferencia en nuestra metodología radica en que incorporamos un bucle para evaluar las imágenes de acuerdo con el *prompt* y el modelo especificado en la ruta de acceso (*path*). Esto nos permitió verificar los cálculos y generar las tablas de resultados.

Después de llevar a cabo la evaluación de las imágenes generadas mediante *CLIP Score*, decidimos emplear la métrica *FID* para evaluar la calidad de los planos generados. Para esta evaluación, utilizamos una pequeña fracción de imágenes que no se emplearon en el ajuste fino, ya que contenían valores simbólicos (anotaciones) nulos. Esto nos llevó a descartar dichos planos de nuestro conjunto de entrenamiento y, en su lugar, utilizarlos para la evaluación con *FID*. Cada imagen generada por los modelos fue procesada para extraer sus características y compararlas con las imágenes originales, permitiendo así el cálculo del *FID*. Cabe recordar que un valor alto de *FID* indica una mayor diferencia entre las imágenes comparadas, lo cual, dependiendo del enfoque de investigación, puede considerarse un resultado no deseado. En nuestro estudio, establecemos que los resultados deben mantenerse en un promedio equilibrado, evitando valores extremos, ya que las estructuras de las imágenes deben asemejarse a los planos de entrenamiento, pero sin ser completamente idénticas, con el fin de preservar la diversidad en la generación de contenido arquitectónico.

A continuación, se presentan el fragmento de código de *FID* utilizado para realizar la evaluación de cada imagen.

```
1  import os
2  import torch
3  from pytorch_fid import fid_score
4
5  def main():
6      # Define the paths to the folders containing the reference and
       generated images
7      real_images_path = ..\FID_Evaluacion\Imagenes_Testes_Reais"
8      fake_images_path = ..\FID_Evaluacion\Imagenes_Pruebas"
9
10     # Calculate the FID score
11     fid_value = fid_score.calculate_fid_given_paths(
12         [real_images_path, fake_images_path],
13         batch_size=16,
14         dims=2048,
15         device='cuda' if torch.cuda.is_available() else 'cpu'
16     )
17
18     print(f'FID Score: {fid_value}')
```

**Fragmento de código** 6.10: Calculo de FID

## 6.6. Desarrollo de la Interfaz de Pruebas Utilizando *Gradio*

Con el objetivo de facilitar la visualización y prueba de los modelos generativos, optamos por desarrollar una interfaz web utilizando la biblioteca de *Python* denominada *Gradio*. Esta biblioteca fue diseñada específicamente para la evaluación de modelos de inteligencia artificial, así como de otros modelos matemáticos y predictivos, proporcionando una manera sencilla e interactiva de realizar pruebas sin necesidad de implementar una interfaz gráfica compleja.

En la continuación, se detallará el desarrollo de esta interfaz, describiendo su implementación, configuración y los resultados obtenidos a partir de su uso en la evaluación de los modelos generativos.

```
import gradio as gr
from diffusers import StableDiffusionPipeline
import torch

# Cargue el modelo entrenado para inferencia
output_dir = "./"
pipeline = StableDiffusionPipeline.from_pretrained(output_dir,
    torch_dtype=torch.float16).to("cuda")

# Funcion para generar imagenes.
def generate_images(prompt, width, height, num_steps, num_images=4):
    images = []
    for _ in range(num_images):
        result = pipeline(prompt, num_inference_steps=num_steps, height=
    height, width=width)
        if "images" in result:
            image = result["images"][0]
        else:
            raise ValueError("Unexpected output format from pipeline")
        images.append(image)
    return images

# Interfaz Gradio para generar imagenes.
interface = gr.Interface(
    fn=generate_images,
    inputs=[
        gr.Textbox(lines=2, placeholder="Escribe tu mensaje aqui...",
    label="Prompt"),
        gr.Slider(minimum=256, maximum=1024, step=64, value=512, label="
    Ancho de la imagen"),
        gr.Slider(minimum=256, maximum=1024, step=64, value=512, label="
    Altura de imagen"),
        gr.Slider(minimum=10, maximum=100, step=10, value=50, label="
    Numero de pasos")
    ],
    outputs=[gr.Image(type="pil", label=f"Imagen {i+1}") for i in range
    (4)],
```

```
31      title="Generador de Planos - Florify",
32      description="Ingrese un mensaje de texto y el modelo generara cuatro
        imagenes basadas en el."
33  )
34
35  # Inicie la interfaz
36  interface.launch()
```

**Fragmento de código** 6.11: Ejemplo del código de ejecución de interface gradio con el modelo generativo

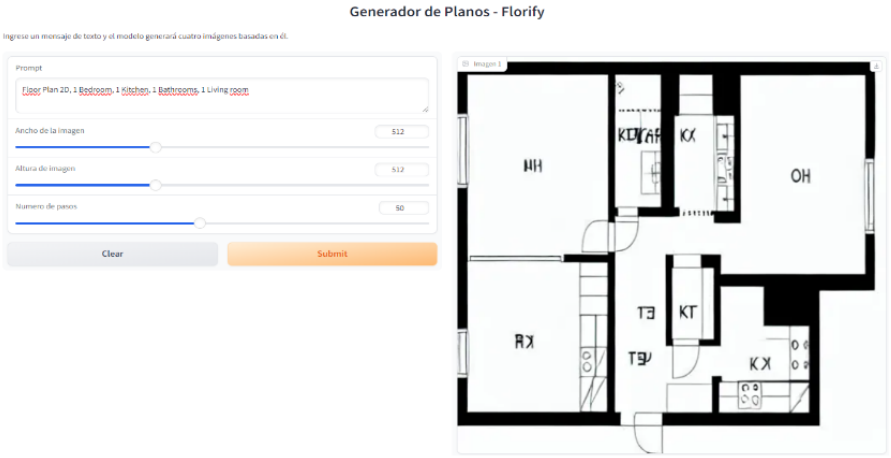


Figura 6.20: Interface de teste Del generador de planos arquitectónicos hecho con *Gradio*

A partir de la observación del fragmento 6.11 y la figura 6.20, se puede notar que ambos cuentan con un campo específico para la inserción de mensajes destinados a la generación de planos. Es importante recordar que, para garantizar el correcto funcionamiento del sistema, los mensajes deben seguir el mismo formato de estilo que fue definido en las frases de entrenamiento dentro del archivo *.json*, el cual fue previamente mencionado en las secciones anteriores 6.2. Para la interfaz web, continuamos con las pruebas utilizando el modelo experimental 3 ajustado con *text-to-image*, ya que este fue el modelo que obtuvo el mejor rendimiento en términos de coherencia contextual, de acuerdo con las evaluaciones realizadas mediante la métrica *CLIP Score*.

Durante la generación de los planos, establecimos que, por defecto, se realizarían cuatro inferencias por modelo, lo que nos permitiría disponer de múltiples opciones y, de esta manera, evaluar visualmente cuál era la mejor imagen generada. Sin embargo, los resultados obtenidos mostraron ciertas inconsistencias. Las imágenes generadas no lograban una puntuación alta en las métricas de evaluación, tales como *CLIP Score* y *FID*, y además fueron evaluadas visualmente por los investigadores de esta investigación, quienes identificaron que, en muchas ocasiones, las imágenes generadas no mantenían coherencia con la descripción proporcionada en el *prompt* de texto, temas que se tratarán en el próximo capítulo. 7: Discusión de Resultados. Debido a estos problemas, se decidió no incluir un conversor de imágenes que transformara las salidas en extensiones vectoriales u

otros formatos distintos al estándar *.jpg* en esta fase del estudio. No obstante, se considera que este aspecto puede ser una línea de investigación futura con el propósito de mejorar la precisión del sistema en la generación de planos arquitectónicos a partir de descripciones textuales.

# 7: Discusión de Resultados

En este capítulo, analizaremos los resultados obtenidos a partir de las implementaciones y experimentos realizados con base en el capítulo anterior 7: Discusión de Resultados. Dicho capítulo tuvo como propósito presentar los datos y *scripts* claves utilizados durante el desarrollo y la ejecución de los procesos de *Fine-Tuning*, lo que permitió extraer información valiosa para un análisis más profundo y específico. A partir de estos datos preliminares, este capítulo se enfocará en un estudio detallado de los resultados obtenidos, con el objetivo de evaluar de manera concluyente la efectividad del *Fine-Tuning* en el cumplimiento de los objetivos de esta investigación. Para ello, realizaremos un análisis textual de los siguientes métodos de ajuste fino: *DreamBooth, LoRAs y Text-to-Image*. Cabe destacar que no se incluirá un análisis del modelo ajustado *The Misto Line*, dado que se concluyó que dicho experimento no era preciso ni esencial para esta investigación, al no contribuir de manera significativa al objetivo principal del estudio, pero hicimos un pequeño análisis sobre esto en el capítulo anterior adentro de su sección 6.4. A continuación, se detallan el proceso y los resultados obtenidos.

## 7.1. Evaluación inicial con *Stable Diffusion* estándar

El primer experimento empleó el modelo estándar de *Stable Diffusion* para generar planos arquitectónicos sin realizar ajustes previos. Se generaron varias imágenes condicionadas por descripciones textuales como *"Floor Plan 2D with 2 bedrooms, 1 bathroom and living room"*. Aunque las imágenes resultaron visualmente atractivas, el modelo requirió múltiples intentos para cumplir con las condiciones específicas del *prompt*, mostrando una falta de consistencia inicial. Las evaluaciones mediante *CLIP Score* reflejaron una correlación positiva moderada entre texto e imagen, con valores cercanos a 0.30, lo cual indicó que los *embeddings* evaluados eran similares pero insuficientes para cumplir completamente con las expectativas del proyecto. En la continuación es posible mirar la tabla de prueba con *Clip Score*:

| **Figuras de Teste** | **Frase de Inferencias** | **Evaluación *CLIP Score*** |
|---|---|---|

| | | |
|---|---|---|
| Figura: 6.8 | *Floor Plan 2D with 2 bedrooms, 1 bathroom and living room* | 0.3084 |
| Figura: 6.9 | *Floor Plan 2D with 4 bedrooms, 2 bathroom, a kitchen and living room* | 0.3250 |

Tabla 7.7: Evaluación con *Clip Score* del modelo estándar *Stable Diffusion*

Como podemos visualizar, los resultados obtenidos a partir del *CLIP Score* son positivos y se acercan al valor de 1, lo que significa que los *embeddings* evaluados son muy similares y correspondientes. Además, se identifican diferencias entre ellos, ya que el valor positivo es bajo, lo que indica que son correspondencias, aunque existen algunas faltas de contexto en la imagen generada en relación con la condición especificada en el *prompt* de texto.

## 7.2. Evaluación del *Fine-Tuning* con *DreamBooth* y *LoRas*

El método *DreamBooth*, reconocido por su facilidad de uso y capacidad para personalizar modelos con pocas imágenes, fue el primer enfoque de ajuste fino implementado. Este método permitió entrenar el modelo utilizando el conjunto de datos *New Floor Plan*, compuesto por 6 imágenes seleccionadas. Aunque este enfoque simplificó el entrenamiento, presentó limitaciones significativas, ya que las imágenes generadas estaban condicionadas por una única frase de entrenamiento. Esto restringió la diversidad de resultados, haciendo que los planos generados fueran poco flexibles frente a distintas condiciones textuales.

Los resultados evaluados mediante *CLIP Score* arrojaron valores en un rango de 0.3034 a 0.3334, confirmando que las imágenes generadas cumplían parcialmente con las especificaciones. Sin embargo, los planos carecían de precisión y presentaban inconsistencias como saturación de colores y falta de coherencia estructural. Aunque el método *DreamBooth* demostró ser viable para personalizar modelos, sus limitaciones lo hicieron menos adecuado para los objetivos de este proyecto.

| **Figuras de Teste** | **Frase de Inferencias** | **Evaluación *CLIP Score*** |
|---|---|---|
| Imagen 1 | *Floor Plan 2D* | 0.3334 |
| Imagen 2 | *Floor Plan 2D* | 0.3273 |
| Imagen 3 | *Floor Plan 2D* | 0.3255 |
| Imagen 4 | *Floor Plan 2D* | 0.3034 |
| Imagen 5 | *Floor Plan 2D* | 0.3239 |

Tabla 7.8: Evaluación con *Clip Score* de las imágenes generadas con *Dreambooth* - 1

El método *LoRAs* se implementó como una alternativa más eficiente y flexible. Este enfoque permitió trabajar con un mayor volumen de datos, utilizando 100 imágenes seleccionadas de *CubiCasa5k*. Durante el entrenamiento, se realizaron múltiples pruebas con modificaciones en los parámetros, como la tasa de aprendizaje y el tamaño del lote. Aunque *LoRAs* presentó ventajas en términos de eficiencia computacional y capacidad para manejar grandes conjuntos de datos, las imágenes generadas carecieron de coherencia en algunos casos, mostrando problemas visuales como borrosidad y tonalidades inesperadas. Esto podría deberse a la limitada cantidad de imágenes en su conjunto de entrenamiento para el ajuste fino, lo que afectó la capacidad del modelo para generalizar correctamente y producir resultados visualmente más precisos y alineados con las descripciones textuales proporcionadas.

| **Figuras de Teste** | **Frase de Inferencias** | **Evaluación *CLIP Score*** |
|---|---|---|
| Imagen 1 | *Floor Plan 2D* | 0.3126 |
| Imagen 2 | *Floor Plan 2D* | 0.3152 |
| Imagen 3 | *Floor Plan 2D* | 0.3173 |
| Imagen 4 | *Floor Plan 2D* | 0.3206 |

Tabla 7.9: Experimento 1: evaluación de imagenes generadas con *LoRas*, mediante al *CLIP Score*

| **Figuras de Teste** | **Frase de Inferencias** | **Evaluación *CLIP Score*** |
|---|---|---|
| Imagen 1 | *Floor Plan 2D* | 0.3098 |
| Imagen 2 | *Floor Plan 2D* | 0.3200 |
| Imagen 3 | *Floor Plan 2D* | 0.2642 |
| Imagen 4 | *Floor Plan 2D* | 0.3029 |
| Imagen 5 | *Floor Plan 2D* | 0.3256 |

Tabla 7.10: Experimento 2: evaluación de imagenes generadas con *LoRas*, mediante al *CLIP Score*

Los resultados del *CLIP Score* presentados en las tablas abajo, oscilaron entre 0.3098 y 0.3256, indicando mejoras moderadas en la correlación entre texto e imagen 7.9 7.10. Sin embargo, estas mejoras no fueron suficientes para cumplir con las expectativas del proyecto. Además, las pruebas evidenciaron una tendencia del modelo a generar imágenes en color, a pesar de que los datos de entrenamiento estaban en escala de grises, lo que reveló un comportamiento inesperado del modelo al completar y personalizar espacios en blanco o en tonos de gris. Este fenómeno también podría estar relacionado con el reducido número de imágenes de entrenamiento.

## 7.3. Evaluación del *Fine-Tuning* con *Text-to-Image*

El método *Text-to-Image*, perteneciente a la biblioteca *Diffusion*, se implementó para abordar las limitaciones observadas en los enfoques anteriores. Este método permitió establecer relaciones más robustas entre múltiples *embeddings* de texto e imagen, facilitando el trabajo con un mayor volumen de imágenes y descripciones textuales. Para este ajuste fino, se utilizó el conjunto de datos transformado de *CubiCasa5k*, compuesto por 450 imágenes y sus respectivas descripciones en un archivo *.json*.

Se entrenaron tres modelos distintos con configuraciones variables, ajustando hiperparámetros clave como la tasa de aprendizaje, el tamaño del lote (*batch size*) y el número de *steps* de entrenamiento. El número de *steps* definía automáticamente el total de épocas; sin embargo, decidimos entrenar todos los modelos con el mismo valor predeterminado descrito en la documentación del ajuste fino. A medida que avanzamos en los experimentos, incrementamos gradualmente las tasas de aprendizaje y los tamaños del *batch*, asegurándonos de no sobrecargar la *GPU* para evitar interrupciones en el entrenamiento. En la siguiente tabla 7.11, se presentan los valores de los hiperparámetros utilizados, así como las pérdidas finales obtenidas y un grafico de apoyo hecho con *TensorBoard*. Como se puede observar, la pérdida total mostró una evolución a lo largo de los entrenamientos.

| Hiperparámetros y Total de Perda | Modelo 01 | Modelo 02 | Modelo 03 |
|---|---|---|---|
| Taja de Aprendizaje | *1e-05* | *3e-05* | *5e-06* |
| *train_batch_size* | 1 | 2 | 4 |
| *max_train_steps* | 1500 | 1500 | 1500 |
| Total de La Perda (Final) | 0,0406 | 0,0364 | 0,0277 |

Tabla 7.11: Datos de los entrenamientos y optimizaciones con el método *Text-to-Image*

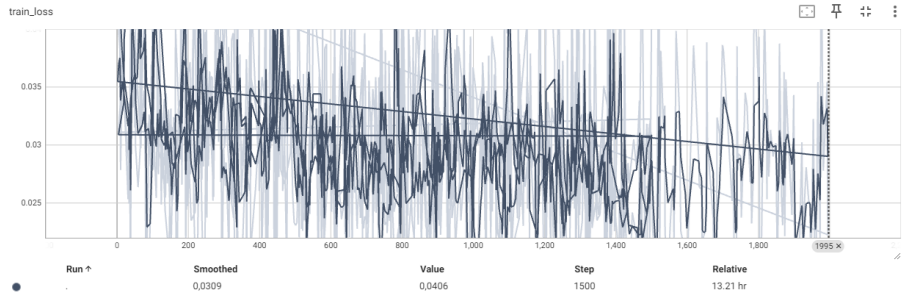


Figura 7.21: Gráfico ejemplo del entrenamiento del modelo ajustado 1

La Figura 7.21 muestra el historial de pérdida (*loss*) durante el entrenamiento de ajuste fino, el cual, como podemos identificar, es bastante agresivo, algo común en ajustes finos en modelos de *Transformers* y *Stable Diffusion* [39]. Además, podemos observar una gran oscilación que podría estar indicando una cierta tendencia al *overfitting* (sobreajuste). Sin embargo, esto no puede confirmarse con certeza, ya que optamos por no trabajar con *datasets* de validación para medir los niveles de pérdida del modelo. Como se identificó

en el Capítulo 5: Técnicas y herramientas, estos no son muy eficaces en modelos como *Stable Diffusion*, donde es necesario utilizar métricas específicas para evaluar la calidad de los modelos. Aun así, en el gráfico podemos notar una marcada tendencia a la oscilación, además de una disminución en los niveles de pérdida, lo que indica que el modelo está aprendiendo. Optamos por no mostrar los gráficos de los otros modelos, ya que presentaban prácticamente los mismos resultados en términos de oscilación, agresividad y reducción de la pérdida.

Los resultados de las evaluaciones con las métricas indicaron avances significativos en la calidad y el contexto de las imágenes generadas. En particular, los valores obtenidos mediante *CLIP Score* alcanzaron hasta 0.3412, lo que sugiere una mayor correlación entre las descripciones textuales y las imágenes producidas. Además, se utilizó la métrica *FID (Fréchet Inception Distance)* para evaluar la similitud entre las imágenes generadas y las originales, obteniendo resultados que reflejaron un equilibrio adecuado entre diversidad visual y coherencia estructural.

A pesar de las mejoras observadas en los modelos entrenados bajo este enfoque, también se identificaron ciertas limitaciones relacionadas con la coherencia entre el texto y la imagen. En algunos casos, las imágenes generadas contenían descripciones imprecisas (anotaciones) o inconsistencias estructurales que comprometían su aplicabilidad como planos arquitectónicos de alta precisión.

Los experimentos reflejados en la tabla muestran que el *CLIP Score* evalúa las imágenes en función de su contexto, validando su relevancia semántica. Sin embargo, se observa una repetición de tendencias en los resultados, en concordancia con pruebas previas realizadas mediante otros métodos, lo que sugiere la necesidad de exploraciones adicionales para optimizar la fidelidad semántica del modelo.

| **Modelo 1** | | | | |
|---|---|---|---|---|
| ***Prompt*** | **Figuras de Teste** | **Frase de Inferencias** | **Evaluación *CLIP Score*** | **Evaluación FID** |
| *Prompt* 1 | Figura 1 | *Floor Plan 2D, 1 bedroom, 1 kitchen, 1 bathroom, 1 living room* | 0.3353 | 97,777 |
| *Prompt 1* | Figura 2 | *Floor Plan 2D, 1 bedroom, 1 kitchen, 1 bathroom, 1 living room* | 0.3272 | 187,019 |

| | | | | |
|---|---|---|---|---|
| *Prompt 1* | Figura 3 | *Floor Plan 2D, 1 bedroom, 1 kitchen, 1 bathroom, 1 living room* | 0.3285 | 158,063 |
| *Prompt 1* | Figura 4 | *Floor Plan 2D, 1 bedroom, 1 kitchen, 1 bathroom, 1 living room* | 0.3408 | 122,168 |
| *Prompt 2* | Figura 1 | *Floor Plan 2D, 2 bedroom, 1 kitchen, 2 bathroom, 1 living room* | 0.3307 | 61,499 |
| *Prompt 2* | Figura 2 | *Floor Plan 2D, 2 bedroom, 1 kitchen, 2 bathroom, 1 living room* | 0.3245 | 66,054 |
| *Prompt 2* | Figura 3 | *Floor Plan 2D, 2 bedroom, 1 kitchen, 2 bathroom, 1 living room* | 0.3228 | 57,636 |

Tabla 7.12: Experimento 1 Modelo 1: evaluación de imágenes generadas con *Text-to-Image*, mediante al *CLIP Score*

| **Modelo 02** | | | | |
|---|---|---|---|---|
| ***Prompt*** | **Figuras de Teste** | **Frase de Inferencias** | **Evaluación *CLIP Score*** | **Evaluación *FID*** |
| *Prompt 1* | Figura 1 | *Floor Plan 2D, 1 bedroom, 1 kitchen, 1 bathroom, 1 living room* | 0.3290 | 370,998 |

| | | | | |
|---|---|---|---|---|
| *Prompt 1* | Figura 2 | *Floor Plan 2D, 1 bedroom, 1 kitchen, 1 bathroom, 1 living room* | 0.3333 | 59,675 |
| *Prompt 1* | Figura 3 | *Floor Plan 2D, 1 bedroom, 1 kitchen, 1 bathroom, 1 living room* | 0.3308 | 66,111 |
| *Prompt 1* | Figura 4 | *Floor Plan 2D, 1 bedroom, 1 kitchen, 1 bathroom, 1 living room* | 0.3362 | 52,619 |
| *Prompt 2* | Figura 1 | *Floor Plan 2D, 2 bedroom, 1 kitchen, 2 bathroom, 1 living room* | 0.3233 | 93,297 |
| *Prompt 2* | Figura 2 | *Floor Plan 2D, 2 bedroom, 1 kitchen, 2 bathroom, 1 living room* | 0.3326 | 64,571 |
| *Prompt 2* | Figura 3 | *Floor Plan 2D, 2 bedroom, 1 kitchen, 2 bathroom, 1 living room* | 0.3256 | 61,205 |
| *Prompt 2* | Figura 4 | *Floor Plan 2D, 2 bedroom, 1 kitchen, 2 bathroom, 1 living room* | 0.3302 | 55,226 |

Tabla 7.13: Experimento 2, Modelo 2: evaluación de imagenes generadas con *Text-to-Image*, mediante al *CLIP Score*

| **Modelo 03** |
|---|

| ***Prompt*** | **Figuras de Teste** | **Frase de Inferencias** | **Evaluación CLIP Score** | **Evaluación FID** |
|---|---|---|---|---|
| *Prompt 1* | Figura 1 | *Floor Plan 2D, 1 bedroom, 1 kitchen, 1 bathroom, 1 living room* | 0.3380 | 239,756 |
| *Prompt 1* | Figura 2 | *Floor Plan 2D, 1 bedroom, 1 kitchen, 1 bathroom, 1 living room* | 0.3289 | 57,785 |
| *Prompt 1* | Figura 3 | *Floor Plan 2D, 1 bedroom, 1 kitchen, 1 bathroom, 1 living room* | 0.3412 | 85,844 |
| *Prompt 1* | Figura 4 | *Floor Plan 2D, 1 bedroom, 1 kitchen, 1 bathroom, 1 living room* | 0.3291 | 91,106 |
| *Prompt 2* | Figura 1 | *Floor Plan 2D, 2 bedroom, 1 kitchen, 2 bathroom, 1 living room* | 0.3243 | 96,945 |
| *Prompt 2* | Figura 2 | *Floor Plan 2D, 2 bedroom, 1 kitchen, 2 bathroom, 1 living room* | 0.3293 | 77,581 |
| *Prompt 2* | Figura 3 | *Floor Plan 2D, 2 bedroom, 1 kitchen, 2 bathroom, 1 living room* | 0.3235 | 134,003 |

| | | | | |
|---|---|---|---|---|
| *Prompt 2* | Figura 4 | *Floor Plan 2D, 2 bedroom, 1 kitchen, 2 bathroom, 1 living room* | 0.3381 | 232,512 |

Tabla 7.14: Experimento 3, Modelo 3: evaluación de imagenes generadas con *Text-to-Image*, mediante al *CLIP Score*

Comparando los valores obtenidos en la evaluación mediante *CLIP Score* en cada experimento de los distintos modelos, se observa que las diferencias entre ellos son mínimas. Los resultados muestran una gran similitud tanto entre los modelos evaluados como con los valores obtenidos en imágenes generadas tras los ajustes finos con otros métodos. En la Figura 7.22, se presenta un gráfico con los valores más altos obtenidos en *CLIP Score* para cada experimento, destacando el mejor resultado registrado en cada modelo. Se evidencia que la variación entre los modelos es marginal, aunque el Modelo 3 muestra una ligera ventaja sobre los Modelos 1 y 2 en términos de coherencia contextual entre la descripción textual y la imagen generada.

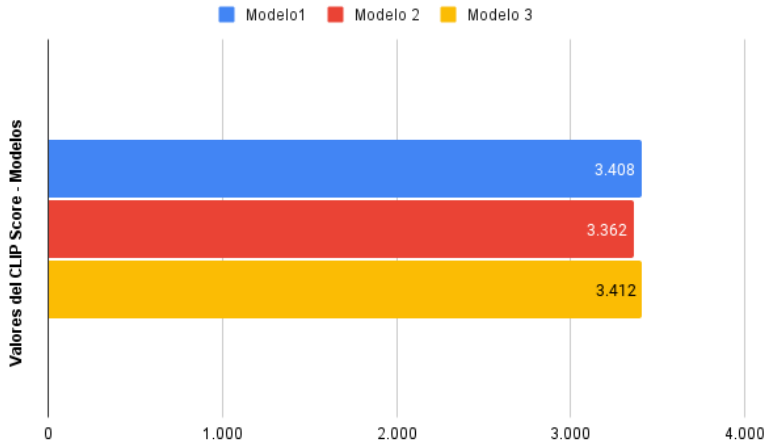


Figura 7.22: Comparación de los experimentos de modelos ajustados *Text_to_Image*

Para visualizar qué modelos obtuvieron los mejores resultados en relación con las características evaluadas mediante el cálculo de *FID*, se generó un gráfico que se presenta en la Figura 7.23. Este gráfico destaca el menor valor obtenido en la evaluación, lo que indica una mayor similitud entre las imágenes generadas y los planos arquitectónicos originales utilizados en el entrenamiento. A partir de los resultados, se observa que el Modelo 2 es el que mejor logra generar imágenes con características de alta calidad, mostrando una mayor correspondencia con los planos originales. Sin embargo, es importante considerar que, aunque los valores de *FID* reflejan esta similitud, aún pueden existir diferencias estructurales entre las imágenes generadas y los datos de referencia.

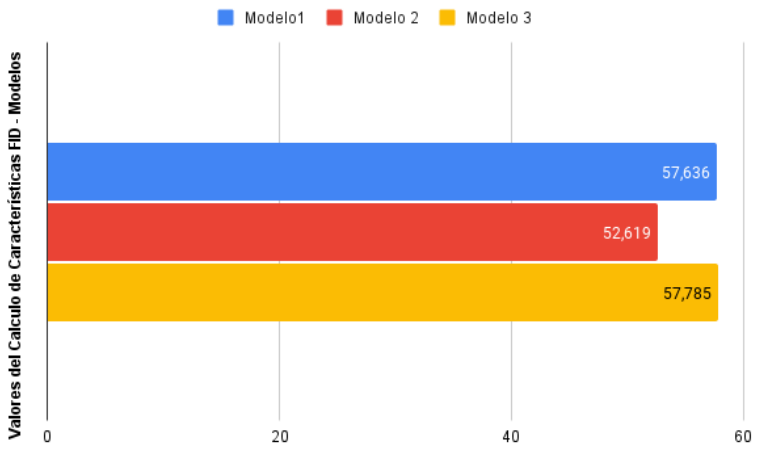


Figura 7.23: Comparación del calculo *FID* de los experimentos de los modelos ajustados *Text-to-Image*

Después de analizar ambos resultados con las dos métricas de evaluación, podemos identificar que, en el caso del *CLIP Score*, que evalúa la correspondencia entre el contexto textual y la imagen generada, el modelo que mejor desempeñó esta tarea fue el Modelo 3. Por otro lado, el *FID*, que mide la similitud entre las imágenes generadas y las imágenes reales de referencia, indica que el Modelo 2 presentó la mejor calidad, ya que obtuvo valores más bajos, lo cual es un indicador positivo en esta métrica. Estos resultados reflejan un empate en términos de calidad y evaluación, demostrando que cada modelo sobresale en un aspecto específico. Sin embargo, a pesar de las métricas obtenidas, un análisis más profundo revela que ambos modelos aún presentan problemas significativos en la generación de imágenes, la coherencia contextual entre los *embeddings* y la calidad visual de los planos generados. Estos resultados sugieren que es necesario continuar con investigaciones adicionales para mejorar el sistema generativo de planos arquitectónicos. En el Capítulo 8: Conclusiones y Líneas de trabajo futuras, se presentarán análisis más detallados y consideraciones finales sobre el desempeño del modelo y las oportunidades de mejora identificadas.

## 7.4. Comparación de Métodos y Experimentos

Al evaluar los métodos implementados en este trabajo, se identificaron diferencias clave en términos de calidad visual, coherencia textual y demanda computacional. En primer lugar, los enfoques basados en *DreamBooth* y *LoRAs* presentaron ciertas limitaciones debido a su dependencia de conjuntos de datos específicos y restricciones en la diversidad de frases de entrenamiento. En particular, estos métodos mostraron problemas de coherencia visual en la generación de planos arquitectónicos, lo que sugiere que pueden no ser la opción más óptima para el objetivo de la investigación. Por otro lado, el método *Text-to-Image*, implementado a través de la biblioteca *Diffusers*, mostró un desempeño superior en

términos de contextualización textual y calidad de los planos generados. Su capacidad para integrar múltiples *embeddings* permitió que los planos resultantes fueran más coherentes con las descripciones proporcionadas en el *prompt*, acercándonos al objetivo principal de esta investigación. No obstante, este método también enfrentó desafíos significativos relacionados con la infraestructura computacional, ya que el tamaño de los modelos generados y el consumo de memoria *GPU* fueron considerablemente más altos en comparación con *DreamBooth* y *LoRAs*.

En términos de evaluación cuantitativa, los resultados obtenidos mediante *CLIP Score* y *FID* revelan que cada modelo posee ventajas distintas. El modelo 3, entrenado con el método *Text-to-Image*, sobresalió en la generación de imágenes que guardaban mayor coherencia con el texto de entrada, obteniendo el mejor puntaje en *CLIP Score*. Esto sugiere que, desde una perspectiva de contextualización semántica, este modelo fue capaz de comprender y representar con mayor precisión las características arquitectónicas especificadas en el prompt. Por otro lado, el modelo 2 del *text-to-image*, demostró un mejor desempeño en términos de calidad estructural, según los valores obtenidos en la métrica *FID*. Este resultado indica que las imágenes generadas por este modelo presentaron mayor similitud con los planos arquitectónicos originales, lo que sugiere una menor distorsión y una mayor fidelidad respecto a los datos de entrenamiento.

Ambos métodos siguen presentando desafíos en la generación de planos con calidad óptima, ya que algunas imágenes generadas muestran inconsistencias estructurales y falta de alineación con los requerimientos del usuario. Por lo tanto, se evidencia la necesidad de optimizar los métodos implementados, buscando un equilibrio entre la contextualización textual y la fidelidad visual. Este análisis también nos llevó a reflexionar sobre la posibilidad de incorporar un método de *Pix-to-Pix* para mejorar el ajuste del modelo, permitiendo completar automáticamente las áreas faltantes en las imágenes generadas. Esta estrategia podría hacer que la generación de planos arquitectónicos fuera más coherente con las especificaciones solicitadas en los *prompts*, optimizando la precisión y utilidad de los resultados. Esta idea surgió en parte a partir de la evaluación de un modelo ajustado descartado en este proyecto, denominado *Misto Line*, cuyo desempeño no cumplió con las expectativas en términos de coherencia estructural y precisión en los detalles arquitectónicos y no parecía complementar en la investigación, pero algunas técnicas suyas pueden ser de gran utilidad para optimizar el proyecto.

# 8: Conclusiones y Líneas de trabajo futuras

Después de terminar el proceso de desarrollo de los ajustes finos y análisis de resultados sobre cada experimento realizado, pudimos concluir que logramos cumplir con algunos de nuestros objetivos y descubrir un espacio para nuevas investigaciones y mejoras para ajustar un modelo generativo de difusión en función de las necesidades del proyecto *Floorify*. Además, hemos descubierto un campo de investigación que no contiene grandes cantidades de estudios y trabajos previos, siendo generalmente los campos de investigación más enfocados en los métodos de ***pix-to-pix***, insertando las condiciones en las propias imágenes y no en texto para transformarlas en planos arquitectónicos, como lo hemos hecho en esta investigación experimental.

En nuestra investigación, logramos realizar el ajuste fino en varios métodos de *Stable Diffusion*, el cual era la mejor opción para llevar a cabo esta acción, ya que era completamente gratuito y contenía su código fuente en modo *Open Source*, lo que nos ayudó mucho y nos permitió realizar ajuste fino con 4 métodos diferentes. Llegamos a la conclusión de que el mejor método que cumplía con nuestra necesidad era el ***Text-to-Image***, pues nos permitía entrenar una cantidad masiva de datos relacionados, a pesar de ser un método que aún está en etapa de desarrollo hasta el año actual de esta investigación.

En las fases de desarrollo, fue posible observar que, después de los ajustes finos con el método mencionado anteriormente, nuestro modelo generativo generaba las imágenes de planos en *2D* de acuerdo con lo que se condicionaba en el *prompt* de texto, cumpliendo así uno de los principales objetivos de este proyecto. Sin embargo, con las evaluaciones realizadas utilizando el *CLIP Score*, pudimos verificar que las imágenes estaban de acuerdo con el contexto solicitado en el *prompt* textual, considerando el modelo 3 del experimento como el mejor en términos de contexto entre los *embeddings*, aunque con una puntuación muy baja, lo que nos señalaba la falta o divergencia de algunos contextos respecto a la imagen que se había solicitado. El cálculo del *FID*, que evaluaba las características de calidad de la imagen, nos arrojó excelentes resultados en algunos experimentos realizados con el método *Text-to-Image*, indicando que muchas de las imágenes estaban de acuerdo con las que se habían entrenado en el proceso de ajuste, considerando el modelo 2 del

experimento como el mejor resultado en este nivel. Además, evaluamos visualmente y pudimos notar que muchas de las imágenes presentaban errores en su contexto de dibujo y generación, generando contextos extraños y sin sentido para la visión humana. Esto nos demostró que, aunque los modelos lograban generar los planos *2D*, aún había mucho trabajo por hacer y mejoras por implementar para que pudieran alcanzar un nivel aceptable a nivel profesional como imágenes útiles de planos arquitectónicos.

Con esto, concluimos que, como posibles mejoras futuras para este experimento, sería conveniente intentar realizar modificaciones en el proceso de entrenamiento del método *Text-to-Image*, para visualizar si existe una mejor manera de optimizar su aprendizaje, además de probar con algún otro modelo de Inteligencia Artificial o con alguna otra arquitectura generativa de imágenes. También observamos que muchas veces el ***CLIP Score*** nos devolvía puntuaciones muy similares en las imágenes generadas por métodos que no tenían relación con diferentes *embeddings* en su entrenamiento, lo que podría representar un error por parte del ***CLIP Score***, ya que es una métrica constituida por un modelo de inteligencia artificial, el cual puede cometer errores, dado que su entrenamiento se basa en grandes cantidades de datos, pero no en un conjunto infinito. Con esto consideramos que sería interesante intentar mejorar este proyecto y realizar evaluaciones desde el punto de vista humano, para obtener una opinión más complementaria acerca del contexto generativo entre *prompt* y imagen.

Como lineas de trabajos futuros y mejorías, dejo las siguientes recomendaciones, tras las conclusiones obtenidas con esta investigación:

- **Mejora el contexto textual relacionado con las imágenes de entrenamiento para ajustar nuevamente el método *Text-To-Image*.** La mejora en el contexto textual debe centrarse en crear frases más naturales para ser entrenadas en el método, ya que en este proyecto, al investigar cómo funcionaba el método, terminamos insertando frases algo robóticas y simplemente enumerando cada habitación de un plano bajo.
- **Modificar el conjunto de datos *CubiCasa5k* añadiendo más muestras de planos.** Uno de los problemas que pueden estar provocando imágenes de baja calidad y poco contexto se debe a la escasez de muestras entrenadas en el modelo inteligente, ya que, como se investigó, fue muy difícil encontrar conjuntos de datos que contuvieran una gran cantidad de planos de buena calidad. Por lo tanto, creemos que sería de gran ventaja ampliar el número de muestras de entrenamiento para mejorar los resultados.
- **Intentar realizar un procesamiento de imágenes coloreando cada habitación con su debida identificación, es decir, implementar una variante de la técnica *Pix-to-Pix* para complementar el ajuste del modelo así como sus inferencias.** Una de las grandes ventajas de utilizar el método ***pix-to-pix*** es la facilidad con la que podemos identificar cada habitación mediante el uso de colores, como se demostró en el capítulo 4: Estado del Arte. Esta característica proporciona

una ventaja significativa al recibir condiciones de relleno en las siluetas de imágenes de planos arquitectónicos, lo cual resulta interesante para intentar aplicarlo en las imágenes de entrenamiento del método ***Text-to-Image*** con el fin de generar planos.

- **Realizar pruebas complementarias mediante la visión humana.** Una de las métricas que puede proporcionar un cierto *feedback* subjetivo, pero válido, además de complementar los algoritmos, es la métrica de percepción humana. Pensamos que, con mejoras en el ajuste del modelo generativo, sería ideal realizar pruebas con seres humanos para ver cómo son capaces de generar imágenes, además de insertar frases más naturales y elegir cuáles son las imágenes que tienen más contexto con lo que ellos ingresaron a través del *prompt* de texto.

Con esto concluimos esta investigación, dejando varios campos abiertos para que se puedan realizar más investigaciones y mejoras, hasta que el sistema generativo de planos arquitectónicos pueda generar imágenes útiles y de calidad. En este momento, podemos ver que nuestros resultados no tuvieron un gran avance positivo, pero pudimos verificar y experimentar con varios métodos de ajuste fino, además de llegar a diversas conclusiones sobre por qué hubo resultados negativos en nuestra investigación. Esto abre espacio para nuevos experimentos y pruebas en futuros proyectos, y con ello concluimos el proyecto *Floorify*.

# Apéndices

*Apéndice* $A$

# Plan de Proyecto

Para el desarrollo de este trabajo, se llevó a cabo una secuencia de metodologías y planificaciones, para que pudiéramos desarrollar todo el proyecto *Floorify* y así redactar toda la parte escrita del *TFM* con el tema del proyecto *Floorify* abordado en esta investigación. A continuación, detallamos todo el proceso de planificación utilizado.

## A.1. Planificación del Trabajo

La planificación del desarrollo del trabajo se inició tras la finalización de la estancia de I+D+i. Después de concluir la estancia, ya teníamos los experimentos y modelos ajustados y almacenados. A partir de esto, elaboramos la metodología para extraer los resultados algorítmicos de los experimentos, aunque aún no habíamos realizado las evaluaciones con las métricas determinadas en el capítulo de Técnicas y Herramientas. Con ello, elaboramos el siguiente plan a seguir para la realización del *TFM*, teniendo en cuenta que el Trabajo de Fin de Máster tiene una carga de 6 *ECTS*, correspondiente a 150 horas, las fases planificadas y la dedicación estimada a cada una de ellas son:

- **Fase 1: Estudio y Desarrollo de las evaluaciones algorítmicas (Metodología) de cada experimento realizado.** Cuando finalizamos la estancia de I+D+i, no habíamos concluido cómo serían las evaluaciones de cada experimento en cuanto al nivel de calidad de cada imagen generada. Con esto, el *TFM* dio inicio a estudios para llevar a cabo los procesos de metodología. **Dedicación estimada:** 24 horas
- **Fase 2: Documentación Inicial del *TFM* (Introducción, Objetivos, Marco Teórico y Estado del Arte).** Tras la conclusión de todos los estudios y pruebas algorítmicas para llevar a cabo el proceso de metodología de este trabajo, iniciaremos la redacción de la introducción, la definición de objetivos, además de realizar una revisión bibliográfica de los principales temas de la investigación, adaptando teorías ya vistas, y adaptando también el estado del arte realizado en el proceso de la estancia de *I+D+i* con *HP SCDS*. **Dedicación estimada:** 56 horas

- **Fase 3: Documentación Central del *TFM* (Técnicas y Herramientas y Aspectos Relevantes del Desarrollo del Proyecto).** Una vez realizado el proceso de estudio y desarrollo de la metodología del proyecto estipulado en la fase 1, llegó el momento de redactar sobre todas las herramientas utilizadas, además de detallar todo el proceso experimental con el modelo generativo y los resultados evaluativos obtenidos a partir de las métricas establecidas para evaluarlo. **Dedicación estimada:** 80 horas.
- **Fase 4: Documentación Final del *TFM* (Conclusiones y Apéndice Plan de Proyecto).** Para finalizar el trabajo, se realizará un repaso de todos los puntos principales del trabajo realizados en esta investigación, como los objetivos cumplidos y no cumplidos, y los resultados obtenidos a partir del desarrollo del modelo generativo y los experimentos realizados y evaluados mediante métricas específicas. También crearemos un apéndice que detallará todo el proceso de planificación que se llevó a cabo para la realización de la parte técnica y la redacción del *TFM* denominado Plan de Proyecto. **Dedicación estimada:** 8 horas.
- **Fase 5: Revisión y Corrección.** El último paso antes de dar por concluido totalmente el trabajo es recibir el *feedback* de los tutores, y para eso se revisa el trabajo entero y se aplican las correcciones y cambios sugeridos por los mismos. **Dedicación estimada:** 50 horas.

**Dedicación total estimada:** 218 horas.

## A.2. Ejecución del Trabajo

Este trabajo tuvo su inicio a partir de las prácticas de I+D+i en informática, de acuerdo con las asignaturas de prácticas curriculares, del Máster en Ingeniería Informática. La siguiente planificación y desarrollo de este *TFM* se llevó a cabo durante el curso 2023/2024, realizándose entre los meses de mayo y septiembre. El proceso de documentación se desarrolló entre agosto y septiembre, totalizando 2 meses de trabajo. A continuación, se detallan las fases de desarrollo y el tiempo real dedicado a cada una de ellas:

- **Fase 1:** Estudio y Desarrollo de las evaluaciones algorítmicas (Metodología) de cada experimento realizado. 30 horas de dedicación.
- **Fase 2:** Documentación Inicial del *TFM* (Introducción, Objetivos, Marco Teórico y Estado del Arte). 56 horas de dedicación.
- **Fase 3:** Documentación Central del *TFM* (Técnicas y Herramientas y Aspectos Relevantes del Desarrollo del Proyecto). 80 horas de dedicación.
- **Fase 4:** Documentación Final del *TFM* (Conclusiones, Apéndice Plan de Proyecto y Apéndice Manual Instalación). 8 horas de dedicación.

- **Fase 5:** Revisión y Corrección. 68 horas de dedicación.

Por tanto, la dedicación total real para la realización del Trabajo de Fin de Máster ha sido de 242 horas.

Apéndice $B$

# Manual de Instalación

Este apéndice tiene como objetivo proporcionar el repositorio oficial del proyecto, ya mencionado en el apéndice anterior, pero retomado aquí para definir los pasos necesarios para la instalación y configuración del entorno, permitiendo así la ejecución del proyecto y sus respectivos experimentos, incluyendo las dependencias y conjuntos de datos utilizados. Además, se presentará la infraestructura requerida y empleada para la reproducción del fine-tuning de un modelo de inteligencia artificial generativa basado en Stable Diffusion, además de demostrar la localización y el uso de las métricas de evaluación de contexto y calidad de los sistemas generativos empleados en el proyecto.

## B.1. Infraestructura y Dependencias Utilizadas

La infraestructura utilizada en este proyecto, como se mencionó en el capítulo 5: Técnicas y herramientas, se basó en el servidor *ECA-SIMM* de la Universidad de Valladolid. Este servidor está equipado con una *GPU NVIDIA GeForce A40* de 48 *GB* y almacenamiento dinámico, proporcionando la capacidad de procesamiento necesaria para el entrenamiento de modelos de *Stable Diffusion*, que requieren un alto rendimiento en el procesamiento de imágenes. La investigación se llevó a cabo en un entorno compartido, utilizando 24 *GB* de *VRAM*, un recurso fundamental para la optimización y ajuste fino de redes neuronales avanzadas. Generalmente, el desarrollo de proyectos de esta magnitud exige una infraestructura computacional física de alto rendimiento. Sin embargo, estudios recientes sugieren que *GPUs* con menor consumo energético y menor cantidad de *VRAM*, como la *NVIDIA RTX 3060 Ti* de 8 *GB*, pueden representar un requisito mínimo viable para la experimentación con *fine-tuning* de modelos *Stable Diffusion*, como se ha demostrado en este proyecto [58]. El sistema operativo de la máquina virtual de *ECA-SIM* es *Ubuntu 22.04.5 LTS (Jammy)*, con una arquitectura *x86_64 (64 bits)* y un *kernel 6.8.0-51-generic*.

Una vez confirmada la idoneidad de la *GPU* para el experimento, resulta esencial la configuración de un entorno de desarrollo adecuado, que no solo permita la ejecución y validación del código, sino que también garantice la instalación del lenguaje de programación

y sus respectivas dependencias. Para una guía técnica detallada sobre la instalación del entorno, en la siguiente sección Repositorios y Ejecuciones B.2 se proporcionan enlaces a repositorios en línea y de acceso público, donde se especifican los procedimientos para la instalación y ejecución del proyecto. Además, se describe el proceso para visualizar y descargar el modelo óptimamente ajustado en los experimentos de text-to-image. A continuación, se presenta el lenguaje de programación utilizado, las bibliotecas esenciales y el método más eficiente para la instalación de dependencias en un entorno *Windows*, junto con la *IDE (Integrated Development Environment)* empleada en el desarrollo de este trabajo.

## Herramientas de Desarollo

- ***Visual Studio Code.*** Editor de código fuente desarrollado por *Microsoft*. Es una herramienta ligera y altamente personalizable, utilizada por programadores para escribir, depurar y ejecutar código en diversos lenguajes de programación. Fue utilizado en toda la fase de programación, ya que teníamos que realizar los *fine-tunings* mediante la ejecución de archivos "*.bash*", a través de código *Python* sencillo. Llegamos a utilizar *Jupyter Notebook*, pero solo para realizar el procesamiento de imágenes, y fue utilizado muy poco. Es muy importante en la instalación que tu tengas instalado, ya que ayudara a ver el código y hacer cambios [19].

- ***Python.*** Lenguaje de programación recomendado para manipular grandes cantidades de datos, además de gestionar modelos y entrenamiento de inteligencia artificial de manera fácil e intuitiva. El lenguaje fue utilizado para desarrollar todo el proyecto *Floorify*, es muy importante que la tengas instalado para que pueda hacer la ejecución de los *scripts*. La versión *python* utilizada fue la *3.10.12* [29].

## Dependencias y Librerías

Todas las dependencias mencionadas a continuación forman parte del lenguaje de programación *Python* y están diseñadas para trabajar en conjunto, garantizando compatibilidad entre sus versiones. La instalación de estas dependencias sigue el método estándar de *Python* a través de *pip*. Todas las bibliotecas necesarias se instalan automáticamente mediante la ejecución del archivo "***install_dependencias_and_modelos.sh***", el cual se encarga de configurar el entorno de trabajo en *Linux* sin necesidad de instalación manual. En la siguiente sección B.2, se detalla el procedimiento de ejecución de este archivo, así como la configuración del entorno y la gestión de repositorios para el correcto funcionamiento del proyecto. Las tecnologías presentadas a continuación fueron utilizadas en diferentes etapas del desarrollo. La mayoría de ellas han sido desarrolladas por *Hugging Face* y fueron fundamentales para llevar a cabo el ajuste fino de modelos. Además, se empleó *PyTorch* para realizar inferencias y cargar modelos, y *Gradio* para ejecutar pruebas de manera gráfica, facilitando la interacción y evaluación de los modelos generados.

- ***Accelerate.*** Es una biblioteca desarrollada por *Hugging Face* que facilita la ejecución de código *PyTorch* en diversas configuraciones distribuidas, incluyendo soporte para entrenamiento en múltiples *GPUs*, *TPUs* y precisión mixta. Simplifica el proceso de escalado de modelos sin necesidad de modificar significativamente el código base [36]. La versión utilizada es la *1.3.0*.

- ***Datasets.*** Desarrollada por *Hugging Face*, esta biblioteca proporciona una colección amplia de conjuntos de datos para tareas de procesamiento de lenguaje natural y visión por computadora. Ofrece herramientas para cargar, preprocesar y manipular datos de manera eficiente, integrándose perfectamente con *PyTorch* y *TensorFlow* [27]. La versión utilizada es la 3.2.0.

- ***Diffusers.*** Es una biblioteca de *Hugging Face* que implementa modelos de difusión de última generación para la generación de imágenes, videos y audio en *PyTorch* y *FLAX*. Facilita la experimentación y el desarrollo de modelos generativos basados en procesos de difusión [79]. La versión utilizada es la *0.32.2*.

- ***Gradio.*** Es una biblioteca que permite crear interfaces de usuario interactivas para modelos de aprendizaje automático de manera sencilla. Facilita la creación de demostraciones web para probar y compartir modelos con otros, sin necesidad de conocimientos profundos en desarrollo web [1]. El *Gradio* fue utilizado para hacer la creación de nuestra interface gráfica de pruebas.La versión utilizada es la *5.15.0*.

- ***Huggingface_hub.*** Es una biblioteca que proporciona herramientas para interactuar con el *Hub* de *Hugging Face*, permitiendo la carga, descarga y gestión de modelos y conjuntos de datos [24]. Facilita la integración y el despliegue de modelos en diversas aplicaciones. La versión utilizada es la 0.28.1.

- ***Pytorch* o *torch.*** Es una biblioteca de código abierto para aprendizaje automático desarrollada por *Facebook's AI Research lab*. Ofrece una amplia gama de herramientas para construir y entrenar modelos de aprendizaje profundo, siendo ampliamente utilizada en investigación y producción [69]. La versión utilizada es la 2.6.0.

- ***Torch Vision.*** Es una biblioteca complementaria a *PyTorch* que proporciona conjuntos de datos, modelos pre-entrenado y transformaciones comunes para visión por computadora. Facilita el desarrollo de aplicaciones de visión al ofrecer componentes reutilizables y optimizados [64]. La versión utilizada es la 0.21.0.

- ***Transformers.*** La biblioteca *Transformers*, desarrollada por *Hugging Face*, proporciona modelos pre-entrenado para *NLP* y visión computacional, incluyendo *GPT*, y *Stable Diffusion*. Es de gran importancia para este proyecto, utilizada tanto para instanciar los algoritmos de ajuste fino de *Stable Diffusion* como para la aplicación de la métrica *CLIP Score*, permitiendo evaluar el contexto entre inferencias generadas [80]. La versión utilizada es la 4.49.0.

- ***Pythorch_Fid.*** La biblioteca *pytorch_fid* se utiliza para calcular el *Fréchet Inception Distance (FID)*, una métrica ampliamente aplicada en la evaluación de la calidad

de imágenes generadas por modelos generativos, comparando estadísticamente sus características con imágenes reales mediante redes neuronales pre-entrenadas. En este estudio, se empleó para evaluar las imágenes generadas a partir de inferencias en modelos ajustados con el método *text-to-image*, permitiendo un análisis cuantitativo de la similitud estructural y la diversidad de las imágenes sintetizadas [73]. La versión utilizada es la 0.3.0.

## ***Docker*** **en** ***Floorify*****: Contenedorización y Gestión del Entorno**

En esta subsección, nos centraremos en presentar la máquina virtual y el proceso de construcción, visualización y arquitectura del entorno de desarrollo del proyecto ***Floorify***. Sin embargo, para llevar a cabo el desarrollo del proyecto, contamos con el apoyo de la plataforma ***Docker***, utilizada para emular la máquina virtual y gestionar las dependencias, instalaciones, memoria y uso de la *GPU*. El contenedor de *Docker* que sirvió como base para la instancia virtual utilizada en esta investigación fue proporcionado por el profesor tutor **Valentín Cadeñoso Payo** y el grupo de investigación ***ECA-SIM***. En esta subsección, no abordaremos los aspectos relacionados con la infraestructura de la máquina virtual, ya que estos fueron tratados en el capítulo 5: Técnicas y herramientas y al inicio de esta sección.

El contenedor *Docker* fue utilizado para crear nuestra instancia de *TFM* e *I+D+i* en el desarrollo del proyecto *Floorify*, dado que la empresa *HP* no pudo proporcionarla para la realización de las prácticas. Recordemos que *Docker* no es más que una plataforma con la capacidad de empaquetar sistemas, *scripts* y datos, con el objetivo de distribuir y ejecutar aplicaciones de forma aislada y portátil, independientemente del entorno local [21]. Dentro de la infraestructura creada por *docker*, contamos con los recursos mencionados anteriormente, asignando un total de 24 *GB* de *GPU* para llevar a cabo los entrenamientos. Cabe destacar que esta cantidad fue la única disponible, ya que otros estudiantes también estaban en proceso de desarrollo de sus *TFM* y *TFG* y necesitaban utilizar la *GPU*.


Figura B.1: Ejemplo del la estancia remota utilizada - Proyecto *Floorify*

El contenedor fue personalizado con el nombre "*cubi*", en referencia al primer conjunto de datos investigado para este estudio científico, el cual se analiza en detalle en los capítulos 5: Técnicas y herramientas y 6: Aspectos relevantes del desarrollo del proyecto. En esta fase experimental, se ha configurado un entorno basado en *Docker* y *JupyterLab*, optimizado para la ejecución de modelos en *GPU NVIDIA* y gestionado con permisos específicos

para evitar problemas de acceso a archivos locales. Este entorno permite la ejecución de experimentos en un entorno portátil y reproducible, garantizando flexibilidad para distintos usuarios. Para iniciar el entorno, primero se construye la imagen *Docker* a partir de un *Dockerfile* ubicado en el directorio *./cubi/CubiCasa5k*. Durante la construcción, se incluyen variables de usuario y grupo (*UID, GID, UNAME, GNAME*), extraídas del sistema con *id -u y id -g*. Esto garantiza que el contenedor herede los permisos del usuario local, evitando conflictos al acceder a archivos montados. La imagen resultante se etiqueta como *cubi-jupyterlab*.

Antes de ejecutar el contenedor, se define un conjunto de variables de entorno dentro de un archivo *user.env*, donde se especifican los identificadores de usuario y grupo, el *token* de *Jupyter*, y la asignación de *GPU* con *NVIDIA_VISIBLE_DEVICES=0*. Este archivo se enlaza a la configuración del contenedor para que se mantengan los valores adecuados en cada sesión. Para la ejecución del contenedor, se emplea *docker-compose*, utilizando el archivo *compose.yaml*. Este archivo configura el servicio *jupyterlab*, asignando un nombre fijo al contenedor (*jp-cubi-giovane*) y exponiendo los puertos 8888, 6006 y 7960 para acceder a *JupyterLab* y otras herramientas. Además, se montan volúmenes desde el *host* para garantizar la persistencia de datos y resultados experimentales.

En casos donde se requiere una sesión interactiva, se utiliza un segundo archivo, *compose-interactive.yaml*, que permite ejecutar el contenedor en modo temporal. En esta configuración, se habilita el soporte para *GPU NVIDIA* y la compartición de memoria interprocesos, optimizando el rendimiento en tareas intensivas. Finalmente, para gestionar la ejecución y limpieza del entorno, se han definido *scripts* adicionales que eliminan archivos temporales y detienen el contenedor una vez finalizados los experimentos. Esta estructura automatiza el proceso, asegurando una configuración eficiente y escalable.

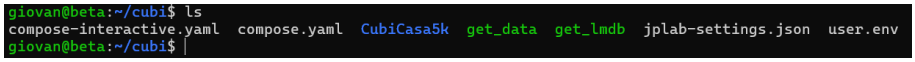


Figura B.2: Ejemplo del entorno *cubi* - Proyecto *Floorify*

Anteriormente, mencionamos brevemente la configuración del entorno de *JupyterLab*, el cual fue configurado por el profesor Valentín. Sin embargo, su uso fue limitado, ya que, como se evidencia en los capítulos 5: Técnicas y herramientas y 6: Aspectos relevantes del desarrollo del proyecto, los *fine tunings* se ejecutaban mayormente en formato *Bash*. Esto nos llevó a adaptarlos para su ejecución en *Python*, convirtiéndolos en archivos *.py* ejecutables dentro del entorno virtual. A pesar de ello, algunas tareas de preprocesamiento, como la generación de *datasets* y ciertas técnicas de manipulación de datos, se realizaron inicialmente en *Jupyter Notebook*. No obstante, posteriormente, todos los procesos fueron trasladados a *scripts .py* ejecutables para optimizar la ejecución y automatización. Del mismo modo, muchas de las configuraciones implementadas para trabajar con *CubiCasa5k* tuvieron un uso limitado dentro de *Jupyter Notebook* en el proyecto *Floorify*. La ejecución principal del proyecto se llevó a cabo a través del entorno *Docker*, utilizando la *GPU NVIDIA* directamente desde la terminal.

Dentro del entorno remoto, implementamos una carpeta específica para almacenar los modelos entrenados, ya que la memoria de la instancia virtual era limitada y no permitía guardar múltiples versiones de los modelos ajustados. Para solucionar este problema, el tutor de este trabajo proporcionó recursos adicionales mediante la habilitación de la carpeta ./data/, ubicada fuera del entorno principal. Esta carpeta fue configurada como un almacenamiento externo, integrándola al sistema mediante un montaje de volumen, lo que permitió acceder a ella de forma transparente desde el entorno de trabajo. De esta manera, los modelos eran generados dentro del entorno virtual y luego transferidos automáticamente a ./data/, asegurando su persistencia sin comprometer la memoria de la instancia

## B.2. Repositorio e instrucciones para su instalación y ejecución

En esta sección, se detallarán todos los repositorios que contienen el código y la estructura utilizados para llevar a cabo los experimentos. Además, se presentará el repositorio del modelo ajustado que obtuvo los mejores resultados según las métricas empleadas, como se mencionó previamente en el capítulo 7: Discusión de Resultados. Asimismo, se describirá el procedimiento completo para la clonación de los proyectos y del modelo, la instalación de sus dependencias y la ejecución final. De este modo, cualquier persona o evaluador independiente podrá replicar el experimento en su propio equipo, garantizando la reproducibilidad de los resultados obtenidos. Antes de continuar, asegúrese de contar con la infraestructura computacional mínima requerida, tal como se describe en la sección anterior de este apéndice. Esto garantizará que el entorno de ejecución sea adecuado para la correcta reproducción de los experimentos y el procesamiento eficiente del modelo.

### Repositorios

Este trabajo está compuesto por tres repositorios distintos, de los cuales dos contienen el desarrollo de los experimentos. Uno de ellos se encuentra en *GitLab*, es de acceso privado y pertenece a *HP SCDS*, donde fue utilizado para actividades de *I+D+i*. Por otro lado, el repositorio en *GitHub* alberga la última versión del proyecto *Floorify* desarrollada para el *TFM*, optimizada para la ejecución de experimentos y completamente pública, permitiendo el acceso al código fuente.

Adicionalmente, disponemos de un repositorio en *Hugging Face*, el cual almacena el modelo experimental ajustado que has obtenido mejor resultado. Este modelo puede ser descargado para realizar inferencias de manera inmediata, sin embargo, es importante disponer de suficiente capacidad de almacenamiento, ya que su tamaño es considerablemente grande. A continuación, se presentan los repositorios junto con sus respectivos enlaces:

- ***GitLab.*** Sistema de repositorios en línea dentro del servidor de la empresa donde realicé el desarrollo, que utiliza el sistema de control de versiones *Git*, en formato

pago y empresarial [33]. Utilizamos *GitLab* de la empresa *HP SCDS* para almacenar todo el desarrollo y los experimentos realizados en este proyecto, lo que permitió guardar y controlar las versiones a medida que realizábamos los experimentos.

- **Disponible de forma privada en:** Repositorio *HP SCDS Floorify*

- ***GitHub.*** Es una plataforma de alojamiento y gestión de proyectos de software que utiliza el sistema de control de versiones *Git*. Permite a los desarrolladores colaborar, compartir código, rastrear cambios, reportar problemas y organizar el trabajo en repositorios. Además, ofrece funcionalidades para la revisión de código, integración continua y documentación de proyectos, siendo ampliamente utilizada por desarrolladores y equipos para la colaboración y el desarrollo de software de forma distribuida y organizada [32]. Para la conclusión de este trabajo, optamos por publicar el proyecto en un repositorio de *GitHub* con el objetivo de hacerlo público, ya que el repositorio de *GitLab* se mantendrá privado para la empresa *HP SCDS*. De esta manera, la investigación realizada estará disponible para todos los interesados.

  - **Disponible de forma Publica en:** Repositorio Publico *Floorify* UVa

- ***Huggin Face Models.*** Es una plataforma líder en inteligencia artificial que proporciona modelos pre-entrenado, herramientas y *APIs* para procesamiento de lenguaje natural, visión por computadora y más. Permite compartir modelos, entrenar redes neuronales y acceder a bibliotecas como *Transformers* y *Diffusers*, facilitando el desarrollo de aplicaciones de aprendizaje automático. Además, dentro del entorno de *Hugging Face*, cada perfil tiene la capacidad de crear y almacenar sus propios modelos, con la posibilidad de integrarlos en *GitHub* o compartirlos públicamente con otros desarrolladores [45]. La plataforma desempeñó un papel fundamental en este trabajo, ya que nos permitió validar y evaluar nuestro modelo de *diffusion*, finalizando con el almacenamiento del mejor modelo obtenido durante la fase experimental de *text-to-image* de acuerdo con el capítulo 7: Discusión de Resultados.

  - **Disponible de forma Publica en:** Repositorio Publico del Modelo Experimental Ajustado Text_to_Image

## Instalación y Ejecución

Para instalar el proyecto *Floorify* y manipular los experimentos, es necesario que el usuario clone el repositorio desde *GitHub*, recordando que el enlace al repositorio se encuentra en la subsección anterior de este trabajo. Una vez completada la clonación, podrá explorar todo el contenido del repositorio, que está estructurado en cinco carpetas principales: ***Gradio_Script***, ***Imagenes_Test_Modelos***, ***Metrics_Evaluation_Adjusted_Models*** *y* ***Model_Test_Local*** *y* ***git_Img***. Además, se incluye un archivo independiente destinado a la instalación de dependencias de *Python*, así como la clonación del proyecto *Diffusers*, lo que permite acceder a los métodos de *fine-tuning* de la biblioteca *Hugging Face*.

A continuación, se detallará el procedimiento para ejecutar los comandos necesarios para instalar las dependencias y realizar la ejecución final del proyecto. Según sus necesidades, el usuario podrá optar por dos enfoques: probar el modelo ya ajustado, permitiendo la ejecución inmediata de inferencias, o ajustar un nuevo modelo con parámetros personalizados para generar un experimento optimizado. De esta manera, el proyecto ofrece flexibilidad tanto para la validación de modelos previamente entrenados como para la personalización de nuevos entrenamientos.

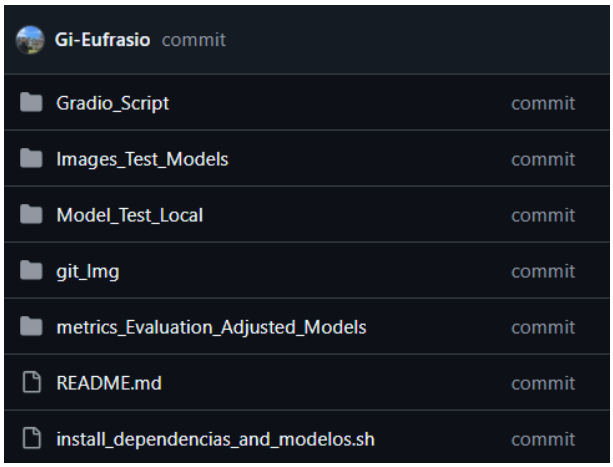


Figura B.3: Repositorio *Git Hub* - Proyecto *Floorify*

El siguiente comando garantiza la **clonación** del repositorio y la **ejecución de la instalación de dependencias**, a través del archivo `.sh` de instalación denominado ***install_dependencias_and_modelos.sh***. El uso del comando de permisos `chmod` puede ser opcional, dependiendo de cómo se haya clonado el proyecto. En algunos casos, el archivo de instalación de dependencias no se descarga con permisos de ejecución, por lo que es necesario aplicar el comando de permisos antes de ejecutarlo, asegurando así su correcta ejecución en el entorno de desarrollo..

```
$ git clone https://github.com/Gi-Eufrasio/Floorify-TFM.git
$ cd Floorify-TFM

$ chmod +x install_dependencias_and_modelos.sh #opcional
$ sudo ./install_dependencias_and_modelos.sh
```

**Fragmento de ejecución de código** B.1: Ejemplo de Clonacion del Proyecto Floorify

El siguiente comando debe ejecutarse en caso de que el usuario desee realizar un nuevo ajuste fino utilizando el conjunto de datos proporcionado por *CubiCasa5k*, el cual ha sido previamente ajustado y *preprocesado*, conforme se explica en el Capítulo 5: Técnicas y herramientas. Sin embargo, si el objetivo es únicamente realizar inferencias sobre el modelo ya ajustado o crear un nuevo conjunto de datos, no se recomienda descomprimir este archivo dentro del método *text_to_image_testes*, con el fin de evitar posibles conflictos

en la organización y estructura del proyecto.

```
$ cd Model_Test_Local/Text_to_Image_Testes/Model
$ unzip Dataset_Resize_Cubi_Casa_5K.zip
```

**Fragmento de ejecución de código** B.2: Ejemplo para descomprimir el conjunto de datos de entrenamiento del metodo Text_to_image

El último comando para ejecutar dentro del método ***text_to_image_testes*** permite tanto la realización de nuevos ***fine-tunings***, como su propio nombre indica el método ***Script_Entrenamiento_Fine_Tuning.py***, como también la ejecución de inferencias directamente desde el código a través del terminal. No obstante, es recomendable realizar todas las pruebas mediante ***Gradio***, ya que ofrece una interfaz web más intuitiva para la verificación de los experimentos. Sin embargo, si el usuario desea realizar pruebas rápidas después del ajuste fino, puede ejecutar el ***Script_inferencias.py*** para obtener resultados de manera inmediata.

```
$ cd Model_Test_Local/Text_to_Image_Testes/Model
$ python3 Script_Entrenamiento_Fine_Tuning.py
$ python3 Script_Inferencias.py
```

**Fragmento de ejecución de código** B.3: Experimentos de ajuste fino e inferencias con el metodo Text_to_image

Los comandos de ***Gradio*** se dividen en dos categorías. Si el usuario desea utilizar el modelo ajustado mediante ***Fine-Tuning*** para realizar inferencias con *Gradio*, es necesario ejecutar el archivo ***Script_Download_Model_Experiment.py***, lo que permitirá descargar el modelo en la carpeta ***text_to_image_testes***, garantizando así una ejecución fluida de las inferencias. Por otro lado, si el usuario ya ha realizado un nuevo ***fine-tuning***, basta con ejecutar ***Script_Inferencias_Gradio_Model.py***, que iniciará automáticamente la interfaz web con el modelo cargado. Es importante recordar que la descarga del modelo se realiza desde el repositorio de ***Hugging Face***, tal como se ejemplificó en la subsección anterior sobre los repositorios.

```
$ cd Floorify-TFM/Gradio_Script/
$ python3 Script_Download_Model_Experiment.py
$ python3 Script_Inferencias_Gradio_Model.py
```

**Fragmento de ejecución de código** B.4: Descargue del modelo ajustado del experimento Floorify y la ejecución de Gradio

Ambos los comandos a continuación tienen el mismo propósito: realizar la descompresión de conjuntos de datos, ejecutar nuevos *fine-tunings* y llevar a cabo inferencias a través de la ejecución de código desde el terminal. Sin embargo, a diferencia de los comandos mencionados anteriormente, los métodos utilizados para los experimentos en este caso son *Dreambooth* y *LoRAs*.

Para experimentar con las inferencias, es imprescindible realizar el *fine-tuning*, ya que, debido a las limitaciones de almacenamiento y repositorios señaladas en el Capítulo 6: Aspectos relevantes del desarrollo del proyecto, no se almacenaron los modelos finales. En su lugar, únicamente se guardaron los parámetros de entrenamiento, los cuales deben alcanzar el mismo o aproximado nivel métrico y de precisión que los obtenidos durante la fase de desarrollo y pruebas.

```
$ cd Floorify-TFM/Model_Test_Local/Dreambooth/Model
$ unzip Floor_Plan_2D_Dataset.zip
$ python3 script_Entrenamiento_Dreambooth.py
$ python3 script_Inferencias.py
```

**Fragmento de ejecución de código** B.5: Ejemplo para descomprimir el conjunto de datos de entrenamiento del metodo Dreambooth

```
$ cd Floorify-TFM/Model_Test_Local/LoRas/Model
$ unzip Dataset.zip
$ python3 script_Entrenamiento.py
$ python3 Inferencias.py
```

**Fragmento de ejecución de código** B.6: Ejemplo para descomprimir el conjunto de datos de entrenamiento del metodo LoRas

Si el usuario que manipula los *scripts* de los experimentos lo desea, puede realizar modificaciones y adaptaciones para otros contextos o para implementar las mejoras y refinamientos señalados en el capítulo 8: Conclusiones y Líneas de trabajo futuras. En esta sección no abordaremos una perspectiva sobre los *scripts* utilizados en este proyecto, dado que fueron considerados y ejemplificados en el capítulo 6: Aspectos relevantes del desarrollo del proyecto.

Para finalizar las ejecuciones, podemos realizar la evaluación de las inferencias obtenidas para determinar si la relación contextual entre los *embeddings* es adecuada utilizando el *CLIP Score* o si la imagen generada posee buena calidad y guarda similitud con alguna imagen de prueba del conjunto de datos, manteniendo, no obstante, una diversidad equilibrada. Para ello, es necesario configurar las carpetas de imágenes y conjuntos de datos ubicadas en ***metrics_Evaluation_Adjusted_Models***. Simplemente se deben colocar las imágenes extraídas de las inferencias en sus respectivas bases y ejecutar los *scripts* disponibles en la carpeta de métricas, lo que permitirá obtener los valores a través del terminal utilizando *Python*. Los comandos ejemplos están en la continuación:

```
$ cd Floorify-TFM/metrics_Evaluation_Adjusted_Models
$ python3 Evaluacion_CLIP_Dreambooth_and_Misto_Line.py
$ python3 Evaluacion_CLIP_LoRas.py
$ python3 Evaluacion_CLIP_Text_To_Image.py
$ python3 Evaluacion_FID_Text_To_Image.py
```

**Fragmento de ejecución de código** B.7: Ejemplo para Ejecutar los Scripts de Métricas con CLIP Score y FID

Para una mejor visualización del proceso de instalación, ejecución, *fine-tuning* e inferencias, acceda al repositorio de *GitHub* en la subsección anterior B.2. Allí encontrará detalles sobre la configuración del entorno, los *scripts* utilizados y ejemplos prácticos para replicar los experimentos.

# Bibliografía


[1] Abid, A., Abdalla, A., Abid, A., Khan, D., Alfozan, A., and Zou, J. Gradio: Hassle-free sharing and testing of ml models in the wild. *arXiv preprint arXiv:1906.02569* (2019).

[2] Academy, K. Revisão sobre regressão linear. https://pt.khanacademy.org/math/statistics-probability/describing-relationships-quantitative-data/introduction-to-trend-lines/a/linear-regression-review, 2021. [Internet; descargado 10-julio-2024].

[3] AI, S. Stable diffusion xl base 1.0. https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0, 2023. Accessed: 2025-02-16.

[4] AI, S. Stable image generation api - v2 beta. https://api.stability.ai/v2beta/stable-image/generate/core, 2025. Accessed: 2025-02-16.

[5] Alec Radford, Jong Wook Kim, C. H., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. Learning transferable visual models from natural language supervision. *artXiv 1*, 2103.00020 (2021).

[6] Alecrim, E. Machine learning: o que é e por que é tão importante. https://tecnoblog.net/responde/machine-learning-ia-o-que-e/, 2018. [Internet; descargado 10-julio-2024].

[7] Amanatullah. Fine-tuning the model: What, why, and how. https://medium.com/@amanatulla1606/fine-tuning-the-model-what-why-and-how-e7fa52bc8ddf, 2023. [Internet; descargado 01-agosto-2024].

[8] Amber Aragon. Advantages of generative ai-driven process automation. https://blogs.mulesoft.com/automation/ai-driven-process-automation/, 2024. [Internet; descargado 01-agosto-2024].

[9] Arquitasa. Planos vivienda. https://arquitasa.com/planos-vivienda/#:~:text=Los%20planos%20de%20una%20vivienda,precisa%20sobre%20un%20proyecto%20arquitect%C3%B3nico, 2023. [Internet; descargado 01-agosto-2024].

[10] Authors, P. Paddleocr: Multi-language optical character recognition tool. https://github.com/PaddlePaddle/PaddleOCR, 2023. Accessed: 2025-02-08.

[11] AWS. What is stable diffusion? https://aws.amazon.com/what-is/stable-diffusion/?nc1=h_ls, 2023. [Internet; descargado 01-agosto-2024].

[12] AWS. ¿qué es la automatización inteligente? https://aws.amazon.com/es/what-is/intelligent-automation/, 2023. [Internet; descargado 01-agosto-2024].

[13] AWS. What is gan? https://aws.amazon.com/es/what-is/gan/, 2024. [Internet; descargado 05-agosto-2024].

[14] Bugendai Tech. Performance metrics in evaluating stable diffusion models. https://www.bugendaitech.com/blogdetails/blog-details/performance-metrics-in-evaluating-stable-diffusion-models, 2023. [Internet; descargado 01-agosto-2024].

[15] Caldwell, M. What is an "author"? - copyright authorship of ai art through a philosophical lens. *Houston Law Review 61*, 411 (2023).

[16] Cedreo. 13 tipos de planos arquitectónicos. https://cedreo.com/es/blog/planos-arquitectonicos/, 2023. [Internet; descargado 05-agosto-2024].

[17] CompVis. Stable diffusion v1-4, 2022. Accessed: 2025-02-16.

[18] CompVis, and Runway. Stable diffusion v1-5. https://huggingface.co/runwayml/stable-diffusion-v1-5, 2022. Accessed: 2025-02-16.

[19] Corporation, M. Visual studio code. https://code.visualstudio.com/, 2024. Último acceso: 8 de febrero de 2025.

[20] DDL. 12 floor plan design tools and tips for beginners and experts. https://drylayout.com/en/articles/floor-design-plan, 2023. [Internet; descargado 05-agosto-2024].

[21] Docker, Inc. What is docker? https://docs.docker.com/get-started/docker-overview/, 2025. Accessed: 12-Feb-2025.

[22] Edward J. Hu, Yelong Shen, P. W., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models. *artXiv 2*, 2106.09685 (2021).

[23] España. Legislación consolidada: Ley 38/1999, de 5 de noviembre, de ordenación de la edificación. https://www.boe.es/buscar/act.php?id=BOE-A-1999-21567, 1999. [BOE núm. 266, de 6 de noviembre de 1999;Internet; descargado 02-agosto-2024].

[24] Face, H. The hugging face hub. https://huggingface.co/docs/huggingface_hub, 2023.

[25] Face, H. Evaluation of diffusion models. https://huggingface.co/docs/diffusers/en/conceptual/evaluation, 2024. Accessed: 2024-02-18.

[26] Face, H. Loading image datasets with metadata. https://huggingface.co/docs/datasets/v2.4.0/en/image_load#imagefolder-with-metadata, 2024. Accessed: 2025-02-08.

[27] Face, H. Hugging face datasets documentation. https://huggingface.co/docs/datasets/index, 2025. Accessed: 2025-02-20.

[28] Foster, D. *Generative Deep Learning: Teaching Machines to Paint, Write, Compose and Play.* O'Reilly, 2019.

[29] Foundation, P. S. Python 3.10.12. https://www.python.org/downloads/release/python-31012/, 2023. Último acceso: 8 de febrero de 2025.

[30] George Lawton. Fréchet inception distance (fid). https://www.techtarget.com/searchenterpriseai/definition/Frechet-inception-distance-FID, 2023. [Internet; descargado 01-agosto-2024].

[31] Getfloorplan. Floor plan in 24 hours. https://getfloorplan.com/, 2024. [Internet; descargado 01-agosto-2024].

[32] GitHub, Inc. Github: Where the world builds software. https://github.com/, 2024. Accessed: February 2024.

[33] GitLab Inc. Gitlab: The complete devops platform. https://about.gitlab.com/, 2024. Accessed: February 2024.

[34] Google Cloud. What is artificial intelligence (ai)? https://cloud.google.com/learn/what-is-artificial-intelligence?hl=en, 2024. [Internet; descargado 05-agosto-2024].

[35] Google Developer Machine Learning. Overview of gan structure. https://developers.google.com/machine-learning/gan/gan_structure?hl=es-419, 2022. [Internet; descargado 05-agosto-2024].

[36] Gugger, S., Debut, L., Wolf, T., Schmid, P., Mueller, Z., Mangrulkar, S., Sun, M., and Bossan, B. Accelerate: Training and inference at scale made simple, efficient and adaptable. https://github.com/huggingface/accelerate, 2022.

[37] Géron, A. *Mãos à Obra: Aprendizado de Máquina com Scikit-Learn, Keras & TensorFlow*. O'Reilly, 2021.

[38] Hashemi-Pour, C. Reinforcement learning. https://pt.khanacademy.org/math/statistics-probability/describing-relationships-quantitative-data/introduction-to-trend-lines/a/linear-regression-review, 2023. [Internet; descargado 10-julio-2024].

[39] Huang, Z., Zhou, P., Yan, S., and Lin, L. Scalelong: Towards more stable training of diffusion model via scaling network long skip connection, 2023.

[40] Huggin Face. Clip. https://huggingface.co/docs/transformers/model_doc/clip#overview, 2024. [Internet; descargado 01-agosto-2024].

[41] Huggin Face. Dreambooth. https://huggingface.co/docs/diffusers/v0.30.0/training/dreambooth, 2024. [Internet; descargado 01-agosto-2024].

[42] Huggin Face. Evaluating diffusion models. https://huggingface.co/docs/diffusers/main/en/conceptual/evaluation, 2024. [Internet; descargado 01-agosto-2024].

[43] Huggin Face. Lora. https://huggingface.co/docs/diffusers/v0.30.0/training/lora, 2024. [Internet; descargado 01-agosto-2024].

[44] Huggin Face. Text-to-image. https://huggingface.co/docs/diffusers/v0.30.0/training/text2image, 2024. [Internet; descargado 01-agosto-2024].

[45] Hugging Face, Inc. Hugging face: The ai community building the future. https://huggingface.co/, 2024. Accessed: February 2024.

[46] IBM. What is artificial intelligence (ai)? https://www.ibm.com/topics/artificial-intelligence, 2024. [Internet; descargado 05-agosto-2024].

[47] IBM. What is computer vision? https://www.ibm.com/topics/computer-vision, 2024. [Internet; descargado 05-agosto-2024].

[48] IBM. What is data science? https://www.ibm.com/topics/data-science, 2024. [Internet; descargado 01-agosto-2024].

[49] IBM. What is deep learning? https://www.ibm.com/topics/deep-learning, 2024. [Internet; descargado 05-agosto-2024].

[50] Jamie Parkinson. Floorplansv2. https://huggingface.co/datasets/jprve/FloorPlansV2, 2024. [Internet; descargado 01-agosto-2024].

[51] Jesse Anglen. Ai meets architecture: Generative design and the automated production of buildings. https://www.linkedin.com/pulse/ai-meets-architecture-generative-design-automated-buildings-anglen/, 2023. [Internet; descargado 01-agosto-2024].

[52] Jesús López Baeza-Rojano. How to evaluate generative image models. https://dagshub.com/blog/how-to-evaluate-generative-image-models/, 2024. [Internet; descargado 01-agosto-2024].

[53] Jobs.Archi. The challenges of technology for architects' work. https://jobs.archi/2024/03/21/the-challenges-of-technology-for-architects-work/, 2024. [Internet; descargado 01-agosto-2024].

[54] Joern Ploennigs, M. B. Automating computational design with generative ai. *artXiv 2*, 2307.02511 (2023).

[55] Jonghwa Shim, Jaeuk Moon, H. K., and Hwang, E. Floordiffusion: Diffusion model-based conditional floorplan image generation method using parameter-efficient fine-tuning and image inpainting. *Journal of Building Engineering 95*, 110320 (2024).

[56] kaleaht. Cubicasa5k: A dataset and an improved multi-task model for floorplan image analysis. https://github.com/CubiCasa/CubiCasa5k, 2019. [Internet; descargado 01-agosto-2024].

[57] Lawton, G. What is generative ai? everything you need to know. https://www.techtarget.com/searchenterpriseai/definition/generative-AI, 2024. [Internet; descargado 10-julio-2024].

[58] Lee, Y., Park, K., Cho, Y., Lee, Y.-J., and Hwang, S. J. Koala: Empirical lessons toward memory-efficient and fast diffusion models for text-to-image synthesis. https://arxiv.org/abs/2312.04005, 2024.

[59] Lily Zhuhadar. Unraveling ai complexity - a comparative view of ai, machine learning, deep learning, and generative ai. https://commons.wikimedia.org/wiki/File:Unraveling_AI_Complexity_-_A_Comparative_View_of_AI,_Machine_Learning,_Deep_Learning,_and_Generative_AI.jpg, 2023. [Internet; descargado 05-agosto-2024].

[60] Lingjie Zhu. Floorplancad dataset. https://floorplancad.github.io/, 2024. [Internet; descargado 01-agosto-2024].

[61] Lopez de Mataras Badia, R., and Meseguer Gonzales, P. *Inteligencia Artificial*. CSIC, 2017.

[62] Lvmin Zhang, Anyi Rao, M. A. Adding conditional control to text-to-image diffusion models. https://huggingface.co/TheMistoAI/MistoLine, 2023. [Internet; descargado 01-agosto-2024].

[63] Maket.AI. Generative design for residential planning. https://www.maket.ai/, 2024. [Internet; descargado 01-agosto-2024].

[64] Marcel, S., and Rodriguez, Y. Torchvision the machine-vision package. https://pytorch.org/vision/stable/index.html, 2010.

[65] Nataniel Ruiz, Yuanzhen Li, V. J., Pritch, Y., Rubinstein, M., and Aberman, K. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. *artXiv 2*, 2208.12242 (2022).

[66] Nicolas Catellier. 14 beginner tips to create a floor plan in revit. https://revitpure.com/blog/14-beginner-tips-to-create-a-floor-plan-in-revit, 2020. [Internet; descargado 05-agosto-2024].

[67] Onkar Mishra. Stable diffusion explained. https://medium.com/@onkarmishra/stable-diffusion-explained-1f101284484d, 2023. [Internet; descargado 01-agosto-2024].

[68] OpenAI. Clip vit-b/32. https://huggingface.co/openai/clip-vit-base-patch32, 2021. Accessed: 2025-02-08.

[69] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems 32* (2019).

[70] Patricia Ayala Jiménez. Spain: Can ai creations be protected by intellectual property? https://www.roedl.com/insights/intellectual-property/2023-2/spain-can-ai-creations-be-protected-by-intellectual-property, 2023. [Internet; descargado 01-agosto-2024].

[71] Polina Kazakova. Pseudo-floor-plan-12k. https://huggingface.co/datasets/zimhe/pseudo-floor-plan-12k, 2023. [Internet; descargado 01-agosto-2024].

[72] Robin Rombach, Andreas Blattamann, P. E., and Ommer, B. High-resolution image synthesis with latent diffusion models. *artXiv 2*, 2112.10752 (2022).

[73] Seitzer, M. pytorch-fid: FID Score for PyTorch. https://github.com/mseitzer/pytorch-fid, August 2020. Version 0.3.0.

[74] Shaw, I. *The Oxford History Of Ancient Egypt*. Oxford University Press, 2000.

[75] Taulli, T. *Introdução à Inteligência Artificial: uma Abordagem Não Técnica*. Novatech, 2020.

[76] umesh verma. New floorplan demo dataset. https://huggingface.co/datasets/umesh16071973/, 2024. [Internet; descargado 01-agosto-2024].

[77] Uni matrix Zero. Using clip score to evaluated images. https://unimatrixz.com/blog/latent-space-clip-score/, 2023. [Internet; descargado 01-agosto-2024].

[78] van Rossum et al., G. Python programming language. https://www.python.org/, 1991. Acesso em: Fev. 2025.

[79] von Platen, P., Patil, S., Lozhkov, A., Cuenca, P., Lambert, N., Rasul, K., Davaadorj, M., Nair, D., Paul, S., Berman, W., Xu, Y., Liu, S., and Wolf, T. Diffusers: State-of-the-art diffusion models. https://github.com/huggingface/diffusers, 2022.

[80] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., and Brew, J. Transformers: State-of-the-art natural language processing. https://huggingface.co/transformers/, 2020. Accessed: 2025-02-08.

[81] Xiaoyu Li, Jonathan Benjamin, X. Z. From text to blueprint: Leveraging text-to-image tools for floor plan creation. *artXiv 1*, 2405.17236 (2024).