Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

# Trabajo de Fin de Grado

Grado en Ingeniería Informática

---

# Aplicación de técnicas de inteligencia artificial para la generación automática de escenarios a partir de lenguaje natural

*Application of artificial intelligence techniques for the automatic generation of scenarios from natural language*

Pablo Pérez González

---

La Laguna, 17 de Junio de 2024

D. **Rafael Arnay del Arco**, profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

D. **José Demetrio Piñeiro Vera**, profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutor

**C E R T I F I C A (N)**

Que la presente memoria titulada:

*“Aplicación de técnicas de inteligencia artificial para la generación automática de escenarios a partir de lenguaje natural”*

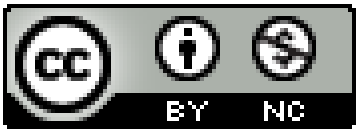
ha sido realizada bajo su dirección por D. **Pablo Pérez González**.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 17 de junio de 2024

# Agradecimientos

A mi madre, a María y a Joaquín.
Sin ellos, nada de esto sería posible.

# Licencia



**Resumen**

*El objetivo de este trabajo ha sido el desarrollo y aplicación de técnicas de inteligencia artificial para la generación automática de escenarios tridimensionales. En particular, se crea la disposición de mobiliario en habitaciones, a partir de lenguaje natural y el modelo de inteligencia artificial (IA) Stable Diffusion. Se lleva a cabo un proceso de ajuste fino de la inteligencia artificial, mediante el cual se adaptan los parámetros y se entrena el modelo utilizando un conjunto de datos de entrenamiento desarrollado por el autor de este trabajo con el fin de mejorar su capacidad para generar escenarios coherentes y relevantes. A través de la implementación y evaluación de un sistema basado en el modelo Stable Diffusion ajustado, se demuestra la viabilidad y eficacia de esta aproximación en la generación automática de escenarios. Además, este modelo está integrado en un proyecto en Unity a través de un servidor Flask desarrollado para facilitar la interacción entre la IA y el entorno de desarrollo de Unity. Los resultados obtenidos muestran que el modelo, entrenado con el conjunto de datos desarrollado por el autor, puede ser una herramienta poderosa y precisa para la creación automatizada de escenarios en diversos campos, con futuras aplicaciones para el diseño de mapas de videojuegos.*

**Palabras clave:** inteligencia artificial, Stable Diffusion, procesamiento de lenguaje natural, entrenamiento, ajuste fino, modelos de IA, generación de imágenes, generación automática de contenido, Flask, Unity, desarrollo de videojuegos.

**Abstract**

*The objective of this project has been the development and application of artificial intelligence techniques to generate three-dimensional scenarios. Specifically, it creates the disposition of furniture in rooms, from natural language and the Stable Diffusion artificial intelligence (AI) model. There is a fine-tuning process of the model with a dataset created by the author of this project to improve the ability to generate consistent scenarios. With software based in the fine-tuned model, feasibility is demonstrated to generate scenarios automatically. Also, the model is integrated in a Unity project through a Flask server developed to help the interaction between AI and the Unity development environment. The obtained results show that the model trained with the author's dataset can be an accurate and powerful tool for the automated generation of scenarios into different fields, with future applications in the design of videogame maps.*

**Keywords:** artificial intelligence, Stable Diffusion, natural language processing, training, fine-tuning, AI models, image generation, automated content generation, Flask, Unity, game development.

# Índice general

**Capítulo 1 Introducción** **11**
1.1 Objetivos 11
1.1.1 Creación del conjunto de datos 11
1.1.2 Entrenar un modelo generador de imágenes a partir de texto 11
1.1.3 Desarrollo de una herramienta que use el modelo 12
1.2 Inteligencia Artificial 12
1.3 Redes Neuronales Artificiales 12
1.4 Procesamiento del Lenguaje Natural 13
1.5 Aplicaciones de la Generación Automática de Escenarios 14
**Capítulo 2 Estado del arte** **15**
2.1 Contexto y avances previos en generación automática de imágenes 15
2.2 Aplicaciones de los modelos de inteligencia artificial generadores de imágenes 16
2.3 Modelos, herramientas y códigos de fine-tuning encontrados basados en IA para la generación automática de imágenes 17
**Capítulo 3 Fases y desarrollo del proyecto** **18**
3. 1 Tecnologías usadas 18
3. 1.1. Tecnologías relacionadas con el modelo 20
3.1.2 Tecnologías empleadas que utilizan el modelo 21
3.1.3 Otras tecnologías 22
3. 2 Metodología de trabajo 22
3.2.1. Enfoque metodológico 22
3.2.2. Fases del proyecto 22
3.2.3. Evaluación y revisión 23
3.3 Conjunto de datos 24
3.4 Modelo empleado 25
3.5 Evolución de la interfaz conectada al modelo 27
**Capítulo 4 Resultados** **33**
Pruebas 34
**Capítulo 5 Conclusiones y líneas futuras** **37**
**Capítulo 6 Summary and Conclusions** **39**
**Capítulo 7 Presupuesto** **41**
7.1 Tabla de Presupuesto 41

**Anexo I - Repositorios de GitHub relevantes** **42**
1. Repositorio de este proyecto 42
2. ComfyUI 42
3. OneTrainer 42
4. Replicación de DALL-E 42
**Bibliografía** **43**

# Índice de figuras

Figura 1.1: Perceptrón multicapa, un tipo de Red Neuronal Artificial……13
Figura 2.1: Comparación de Stable Diffusion, DALL-E y MidJourney……17
Figura 3.1: Funcionamiento de un modelo de difusión……20
Figura 3.2: Arquitectura de la redes U-Net……...20
Figura 3.3: Arquitectura del modelo Stable Diffusion……22
Figura 3.4: Ejemplo de la primera versión del conjunto de datos……...25
Figura 3.5: Imagen del conjunto final de entrenamiento y su descripción……26
Figura 3.6: Configuración del entrenamiento para un modelo ……27
Figura 3.7: Interfaz del apartado de la pestaña de pruebas……28
Figura 3.8: Diagrama de ComfyUI que permite la generación de imágenes……29
Figura 3.9: Pseudocódigo del método principal de la aplicación Flask……..30
Figura 3.10: Código Python encargado del análisis de la imagen simplificado para un único mueble……31
Figura 3.11: Llamada a una API con Unity……31
Figura 3.12: Atributos de la clase *ComfyPOST* y sus valores predeterminados……..32
Figura 3.13. Arquitectura completa del proyecto……...33
Figura 4.1. Vista de la aplicación Flask en el navegador……34

# Índice de tablas

Tabla 5.1: Resultados de ejecución 1……35
Tabla 5.2: Resultados de ejecución 2……36
Tabla 5.3: Resultados de ejecución 3……37
Tabla 7.1: Resumen de tipos……42

# Capítulo 1 Introducción

Los avances en la inteligencia artificial crecen exponencialmente. Gracias a esto, muchas labores extensas o complejas se pueden resolver de forma rápida y sencilla. A la hora de crear entornos virtuales, puede resultar muy tedioso o complicado colocar todos los elementos del escenario en el lugar correcto. Este trabajo demuestra que dicha labor se puede automatizar gracias a la Inteligencia Artificial, específicamente en el mobiliario de una habitación.

## 1.1 Objetivos

Para este trabajo se han propuesto tres objetivos principales, los cuales se explicarán en las siguientes secciones.

### 1.1.1 Creación del conjunto de datos

El proyecto consiste en un modelo de Aprendizaje Automático, por lo que es necesario un conjunto de datos para que el modelo pueda aprender. Por esto mismo, obtener un conjunto de datos variado, consistente y de gran tamaño es primordial, siendo el objetivo más complejo e importante de toda la fase de desarrollo.

Se pretenden crear mapas de habitaciones cuadradas desde un punto de vista cenital con diverso mobiliario en diversas posiciones, por lo que se utilizará la aleatoriedad para que se generen habitaciones de diferentes tamaños, con más o menos mobiliario, variado y con muebles de distinto tamaño. Todo elemento debe ser representado por cuadrados de colores indicativos sobre qué mueble es. Cada tipo de mobiliario debe tener un color único y muy diferente para que el modelo sea capaz de diferenciarlos correctamente. Por ejemplo, todas las sillas deben ser del mismo color pero muy diferentes de las puertas o el suelo. Además, no todas las posiciones son permitidas, pues las puertas deben estar en algún lateral de la habitación y ningún elemento debe superponerse con otros. A partir de estas imágenes pseudoaleatorias, se tratará de obtener una o varias descripciones de la misma, de modo que el modelo tenga variedad de descripciones, permitiendo al usuario usar un lenguaje más natural y diverso para generar las imágenes.

### 1.1.2 Entrenar un modelo generador de imágenes a partir de texto

Tras obtener un conjunto de datos consistente y de calidad, el siguiente objetivo prioritario es entrenar un modelo Text-To-Images con dicho conjunto de datos. Tras esto, el modelo debe ser capaz de generar imágenes de mapas de mobiliario de habitaciones vistas desde arriba a partir de una descripción que referencie el mobiliario, su tamaño y/o la posición de algunos o todos los muebles.

Esto implica que el modelo debe haber comprendido qué es cada elemento del mobiliario, diversas localizaciones y tamaños. Por ejemplo, el modelo debe ser capaz de diferenciar una silla de una mesa, algo grande de lo pequeño y entender referencias como “a la izquierda”, “arriba” o “en el centro” en inglés.

### 1.1.3 Desarrollo de una herramienta que use el modelo

Una vez el modelo esté entrenado, debe implementarse algún tipo de herramienta que emplee el modelo, preferiblemente orientada hacia el campo de desarrollo de videojuegos y generación de espacios tridimensionales. Para ello deberá conectarse el modelo con la herramienta mediante algún tipo de servidor con una API o similar. Esta herramienta debe ser capaz de comunicarse con el modelo, de modo que sea capaz de generar el escenario mediante una frase en inglés que describa la habitación generada.

## 1.2 Inteligencia Artificial

La inteligencia artificial, IA en adelante, es una disciplina y conjunto de conocimientos o capacidades que son expresadas por algún sistema informático o algoritmo que sea capaz de imitar el comportamiento o conocimiento humano (Inteligencia artificial, 2024). Con el transcurso del tiempo, este campo ha avanzado rápidamente, permitiendo el desarrollo de nuevas tecnologías relevantes. La IA está basada en modelos o algoritmos que permiten el procesamiento y análisis de grandes cantidades de datos para sintetizar patrones, permitiendo la capacidad de tomar decisiones o realizar predicciones. Los sistemas de IA pueden utilizar diversos enfoques, como redes neuronales, algoritmos de aprendizaje automático o técnicas de procesamiento de lenguaje natural, entre otros. En este proyecto se utilizan todos los previamente mencionados. A continuación, se hará una pequeña introducción a cada enfoque y posteriormente se tratará en profundidad cada tema.

Las redes neuronales artificiales son un enfoque común en la IA. Estas redes están inspiradas en el funcionamiento del sistema nervioso humano y se componen de unidades interconectadas llamadas neuronas artificiales. Estas neuronas se organizan en capas y se utilizan para procesar información y realizar cálculos complejos.

El aprendizaje automático es otra área clave de la IA, que se centra en desarrollar algoritmos que permiten a las máquinas aprender a partir de los datos sin una programación explícita. Los algoritmos de aprendizaje automático pueden clasificar datos, hacer predicciones o generar nuevas soluciones a partir de patrones identificados en los conjuntos de datos de entrenamiento. En el caso de este trabajo, se generarán imágenes.

La IA también puede utilizar técnicas de procesamiento del lenguaje natural para comprender y generar lenguaje humano. Esto implica la capacidad de reconocer y analizar el habla, traducir entre diferentes idiomas, generar texto coherente y responder preguntas basadas en el lenguaje humano. Aplicado a este proyecto, el procesamiento de lenguaje natural permite interpretar y comprender aquel escenario que quiere generarse.

## 1.3 Redes Neuronales Artificiales

Al igual que la mayoría de las Inteligencias Artificiales más potentes utilizadas en la actualidad, la generación de este tipo de contenido se basa en el uso de redes neuronales. Estas redes son modelos computacionales que consisten en un conjunto de nodos interconectados, conocidos como neuronas, organizados en una o más capas (Red neuronal artificial, 2024). La información de entrada fluye a través de la red neuronal, pasando por las neuronas de entrada y sometiéndose a diversas operaciones, hasta llegar a las neuronas de salida donde se generan los valores resultantes. Estos modelos, al ser capaces de aprender y autodeterminarse en lugar de ser programados de manera explícita, han demostrado su eficacia en el campo del aprendizaje automático.

Los avances recientes en *hardware* han permitido un notable desarrollo de las redes neuronales, aprovechando la capacidad de procesamiento paralelo, una tarea especialmente adecuada para las tarjetas gráficas que han experimentado un aumento significativo en su potencia computacional en los últimos años. Esto ha desencadenado una nueva revolución en el campo de la Inteligencia Artificial, abriendo ampliamente las puertas a la investigación de nuevas arquitecturas de redes neuronales diseñadas específicamente para optimizar determinados tipos de tareas.

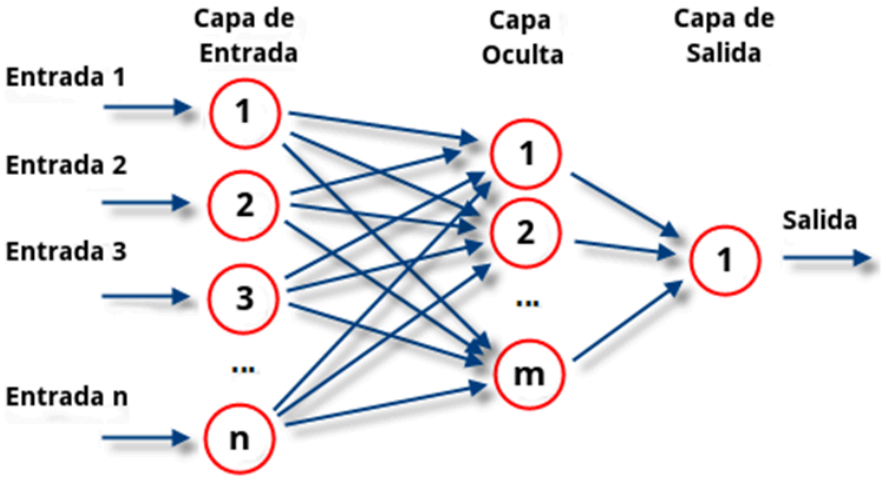



Figura 1.1: Perceptrón multicapa, un tipo de Red Neuronal Artificial (Perceptrón multicapa, 2024).

## 1.4 Procesamiento del Lenguaje Natural

Este ámbito de la Inteligencia Artificial se especializa en estudiar la interacción entre máquinas y seres humanos a través del lenguaje natural. Hay diversos modelos que buscan alcanzar múltiples objetivos en esta área, tales como la creación de texto, el reconocimiento de voz, la traducción automática y la respuesta adecuada a un texto. Los componentes para realizar estas tareas son el análisis morfológico, sintáctico, semántico, pragmático, planificación de la frase y, finalmente, su generación. Como es lógico, no siempre se emplean todos los componentes, sino aquellos que son necesarios para realizar la tarea deseada. No obstante, este proyecto se enfocará en el análisis semántico del lenguaje, esencial para entender el texto del usuario y generar contenidos a partir de él.

La mayoría de modelos de Procesamiento de Lenguaje Natural se basan en los principios de tokenización y *word embedding*. La tokenización implica descomponer el texto en unidades más pequeñas llamadas *tokens*, que pueden ser caracteres, palabras completas o partes de estas. Asimismo, el *word embedding* es una técnica que asigna un vector que contiene información semántica a cada palabra. Esto permite asociar o disociar los vectores (palabras) según distintos contextos gramaticales. De esta manera, las palabras pueden representarse matemáticamente en un espacio n-dimensional, donde, por ejemplo, un coche puede estar más cerca de un camión y más lejos de un pantalón.

El concepto del procesamiento del lenguaje natural existe desde 1950 gracias a Alan Turing, cuando publicó un artículo (Turing, 1950) en el que propuso el actual Test de Turing. Originalmente se emplearon diversos paradigmas y algoritmos estadísticos para enfrentar problemas de Procesamiento del Lenguaje Natural. Con el paso del tiempo se pasó al empleo de las Redes Neuronales Recurrentes, que procesan secuencialmente las palabras de un texto mediante una red neuronal, de modo que el resultado del texto que se analiza en el momento sirve como entrada para la red con la siguiente palabra. Sin embargo, este tipo de redes presenta una limitación al tratar el lenguaje natural. Conforme se analiza el texto, los pesos de las primeras palabras disminuyen, ocasionando una pérdida de memoria y resultados poco precisos en textos largos. Para remediar esto, se han desarrollado los mecanismos de atención, que permiten a una red neuronal asignar un vector de atención a cada palabra de un texto. Este vector indica la relevancia que el modelo otorga a las otras palabras al procesar la palabra de entrada y se establecen conexiones entre ellas sin importar su distancia.

Los mecanismos de atención son cruciales en los *transformers* (Vaswani, 2017), un tipo de red neuronal creada para reemplazar a las recurrentes, logrando grandes avances en el procesamiento del lenguaje natural. La introducción de los *transformers* en 2017 facilitó el desarrollo de modelos de lenguaje mucho más potentes, precisos y realistas. Además, se emplean en modelos que vinculan texto con imágenes, un aspecto que se explicará de forma más detallada con posteridad. Uno de los modelos más prominentes en el procesamiento del lenguaje natural, basado en transformers, es GPT-3.5 (Brown, 2020), desarrollado por OpenAI, utilizado en ChatGPT y uno de los elementos esenciales de DALL-E (Ramesh, 2021), un modelo de generación automática de imágenes (Schüller, 2022).

## 1.5 Aplicaciones de la Generación Automática de Escenarios

La generación automática de imágenes a partir de lenguaje natural es una tecnología con un amplio rango de aplicaciones prácticas en diferentes campos. Una de las áreas en las que esta tecnología puede tener un impacto significativo es la industria del entretenimiento. En este sector, la generación automática de imágenes puede ser empleada para crear escenarios visuales para películas, videojuegos y animaciones. Utilizando el poder del lenguaje natural, es posible traducir descripciones textuales en imágenes realistas y envolventes que ayuden a dar vida a las historias y mundos ficticios.

Además de la industria del entretenimiento, la generación automática de imágenes también encuentra aplicaciones en el ámbito educativo. En este contexto, esta tecnología puede ser utilizada para generar ejercicios interactivos, simulaciones y material de estudio que enriquezcan la experiencia de aprendizaje de los estudiantes. Por ejemplo, se pueden crear imágenes ilustrativas que ayuden a visualizar conceptos abstractos, escenarios históricos o procesos científicos complejos. Esto fomenta la comprensión y participación activa de los estudiantes, facilitando su aprendizaje y promoviendo un ambiente educativo más dinámico y estimulante.

La generación automática de imágenes también puede tener un impacto en otros campos, como la publicidad y el diseño. En la publicidad, esta tecnología puede ser utilizada para crear imágenes personalizadas que se adapten a las preferencias y necesidades del público objetivo. En el diseño, la generación automática de imágenes puede ser empleada para explorar diferentes opciones creativas y facilitar el proceso de conceptualización y prototipado.

# Capítulo 2 Estado del arte

## 2.1 Contexto y avances previos en generación automática de imágenes

Actualmente la sociedad experimenta un crecimiento tecnológico exponencial en el campo de la inteligencia artificial (IA). Esta se encuentra en un desarrollo constante donde diversos modelos aparecen continuamente y otros se consideran obsoletos muy rápidamente. En especial, las IAs que generan imágenes a partir de texto son de los modelos más populares, junto a los modelos de generación de texto tipo GPT. El avance en los modelos que generan imágenes a partir de frases (Text-To-Image) ha sido vertiginoso. Los modelos pioneros fueron DALL-E (Ramesh, 2021), Stable Diffusion (Rombach, 2022) y MidJourney (MidJourney, 2022), donde el primer lanzamiento ocurrió en 2021.

DALL-E (Ramesh, 2021) fue lanzado en enero de 2021, siendo el primer modelo de generación de imágenes a partir de lenguaje natural tal y como los conocemos hoy en día. Este modelo está desarrollado por la compañía americana OpenAI y actualmente cuenta con tres versiones. Referente al funcionamiento, DALL-E es una implementación multimodal del modelo GPT-3, también desarrollado por OpenAI, para el procesamiento de lenguaje natural, usando 12 mil millones de parámetros para comprender el texto de entrada. Internamente, funciona junto a otro modelo llamado CLIP (Contrastive Language-Image Pre-training) (Radford, 2021), de la misma empresa, que se encarga de comprender, clasificar y seleccionar las mejores imágenes generadas.

Stable Diffusion (Rombach, 2022) se publicó en agosto de 2022 por Runway y LMU Múnich. Este modelo tiene la peculiaridad de que es de código abierto, por lo que no tiene un límite de imágenes a generar por el usuario. Es un modelo de difusión, por lo que las imágenes parten de un ruido gaussiano, donde se va definiendo la imagen poco a poco hasta obtener el resultado final.

MidJourney (MidJourney, 2022) se convirtió en beta abierta en julio de 2022. MidJourney tiene la singularidad de que solamente se puede utilizar desde el bot de Discord oficial. Al igual que Stable Diffusion, es un modelo de difusión. De las tres alternativas mostradas, este modelo es el que hace imágenes más realistas.

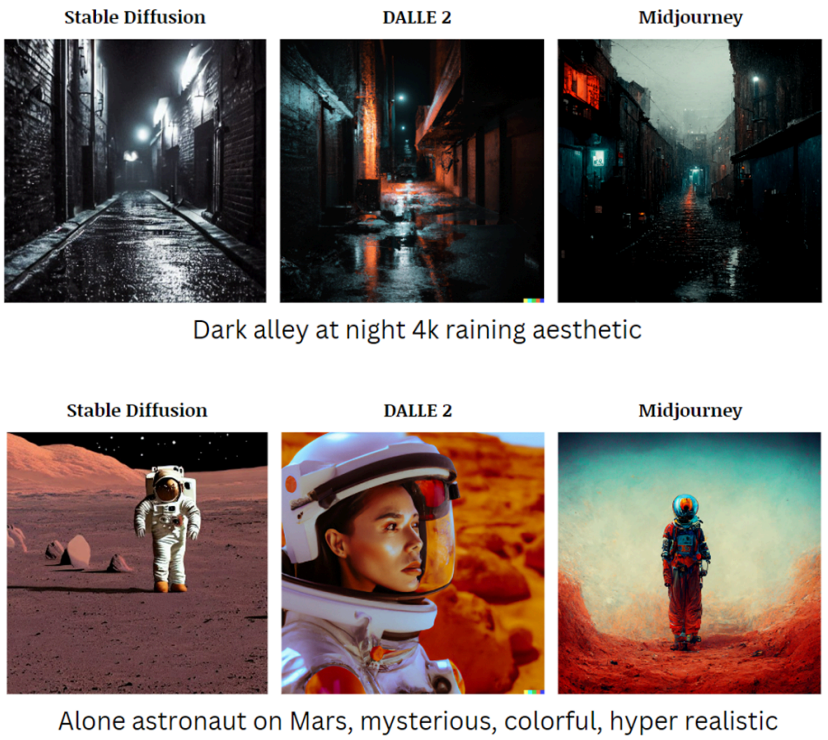



Figura 2.1: Comparación de Stable Diffusion, DALL-E y MidJourney (Islam, 2022).

## 2.2 Aplicaciones de los modelos de inteligencia artificial generadores de imágenes

Los generadores automáticos de imágenes tienen múltiples aplicaciones en diferentes ámbitos. A continuación, se nombrarán algunos para los que puede utilizarse y cómo puede facilitar la vida de los usuarios que lo utilicen.

- Generación de arte: Una de las aplicaciones es generar obras artísticas originales tales como pinturas, ilustraciones, diseños abstractos o demás. No solo permite crear obras enteras, también puede servir como inspiración o crear bocetos para artistas.
- Diseño de personajes. Los desarrolladores de videojuegos y artistas pueden utilizar estos modelos para facilitar el proceso de creación de personajes únicos y detallados. Proporcionando características y descripciones, el modelo puede crear imágenes de personajes que luego pueden ser refinadas y utilizadas.
- Diseño de interiores. Los modelos de generación de imágenes pueden ser utilizados para generar imágenes de interiores de calidad, lo que puede ser útil en la industria del diseño de interiores y la arquitectura. Al proporcionar descripciones de espacios o características específicas, el modelo puede ayudar a visualizar y explorar diferentes ideas de diseño.
- Creación de fondos y paisajes. Se pueden generar imágenes realistas de paisajes a partir de descripciones o de imágenes de referencia. Esto puede ser útil para la creación de fondos en películas, animaciones, videojuegos o incluso para la generación de fondos de pantalla.
- Investigación científica. Desde un aspecto más técnico, se puede utilizar la inteligencia artificial para simular y visualizar datos de diversas investigaciones. Por ejemplo, en astronomía, se puede utilizar para generar imágenes de galaxias o fenómenos cósmicos basándose en datos observacionales.

- Diseño de productos. Los diseñadores industriales y de productos pueden utilizar estos modelos para generar imágenes conceptuales de productos antes de fabricar prototipos físicos. Esto puede ayudar a visualizar y evaluar diferentes diseños y características sin incurrir en los costos y el tiempo asociados con la fabricación.

## 2.3 Modelos, herramientas y códigos de *fine-tuning* encontrados basados en IA para la generación automática de imágenes

Se han encontrado diversos modelos y códigos capaces de ser reentrenados. Los enlaces a los diferentes códigos se encuentran en el Anexo.

- Generación de imágenes desde frases de texto con VQGAN (Esser, 2021) y CLIP. Este código es capaz de generar una serie de imágenes a partir de un texto descriptivo en inglés. Para eso utiliza VQGAN, un tipo de red neuronal capaz de generar imágenes a partir de una codificación intermedia, y CLIP, cuya labor es relacionar información en forma de texto con imágenes. Finalmente genera un vídeo con la evolución de las imágenes generadas. (Crowson, s. f.).

- Repositorio de OpenAI con DALL-E. Repositorio oficial OpenAI en GitHub en el que se encuentra la implementación de DALL-E con Pytorch (Paszke, 2019), biblioteca de aprendizaje automático de código abierto basada en la biblioteca de Torch, utilizado para aplicaciones como visión artificial y procesamiento de lenguaje natural. (OpenAI, 2021).

- Replicación de DALL-E con Pytorch. Repositorio no oficial en GitHub que implementa una versión de DALL-E con Pytorch, permitiendo el ajuste fino. (Wang, 2023).

- Código de ajuste fino utilizando una replicación de DALL-E. Este cuadernillo utiliza la replicación de DALL-E previamente listada para, con una base de datos propia de aves, reentrenar un modelo que posteriormente permita generar imágenes específicas de pájaros. (Wang, 2023).

- ComfyUI: Herramienta de código abierto que permite visualizar y aplicar LoRAs (Low-Rank Adaptation) (Hu, 2021) a modelos, así como introducir un texto y generar imágenes con un modelo seleccionado. Es compatible con diversos modelos y tiene una interfaz basada en nodos donde conectar diferentes elementos, ya sea modelos, entradas de texto, imágenes (como entrada o generadas), etc. (Comfyanonymus, 2024).

- OneTrainer: Herramienta de código abierto que permite entrenar de forma fácil y mediante una interfaz gráfica modelos de Stable Diffusion. (Nerogar, 2024).

- Diversos modelos de Stable Diffusion publicados (Rombach, 2022).

# Capítulo 3 Fases y desarrollo del proyecto

## 3.1 Tecnologías usadas

La generación automática de imágenes a partir de lenguaje natural ha sido objeto de investigación en el campo de la IA. Diversos enfoques, como el uso de redes neuronales y algoritmos de aprendizaje automático, han sido explorados para generar imágenes coherentes y relevantes basadas en descripciones textuales. En este ámbito destacan algunos modelos de inteligencia artificial como DALL-E (Ramesh, 2021), Text To Image (DeepAI, s. f.), MidJourney (MidJourney, 2022) o Stable Diffusion (Rombach, 2022).

Centrándonos en el modelo que se ha empleado en este trabajo, se explicará qué es un modelo de difusión, una red U-Net (Ronneberger, 2015) y posteriormente se entrará en detalles sobre el modelo elegido: Stable Diffusion.

Los modelos de difusión son modelos de aprendizaje automático. El objetivo de estos modelos es aprender la estructura de un conjunto de datos para luego ser capaz de generar elementos nuevos. En el ámbito de la generación de imágenes, el proceso de entrenamiento se basa en una idea relativamente simple: se añade ruido a cada imagen del conjunto de entrenamiento hasta difuminarla completamente y posteriormente se trata de eliminar dicho ruido y obtener la imagen inicial. Como se puede entender, el entrenamiento tiene dos fases:

1. Difusión hacia delante: Consiste en el proceso de difuminar la imagen. Se realiza de forma progresiva, obteniendo una secuencia de imágenes cada vez más difuminada donde se conoce cuánto ruido se añade en cada iteración.
2. Difusión hacia detrás: Es el proceso inverso, donde se trata de reconstruir la imagen. Esto se hará en el mismo número de pasos que hicieron falta en la fase anterior, siguiendo una distribución normal inversa. Calcular esta distribución completamente es imposible, pero es posible calcular una aproximación, que es la forma matemática de expresar “calcular una imagen con menos ruido que la anterior”. Para este proceso se suele usar la arquitectura U-Net (Ronneberger, 2015), usada en modelos como DALL-E, MidJourney o Stable Diffusion.

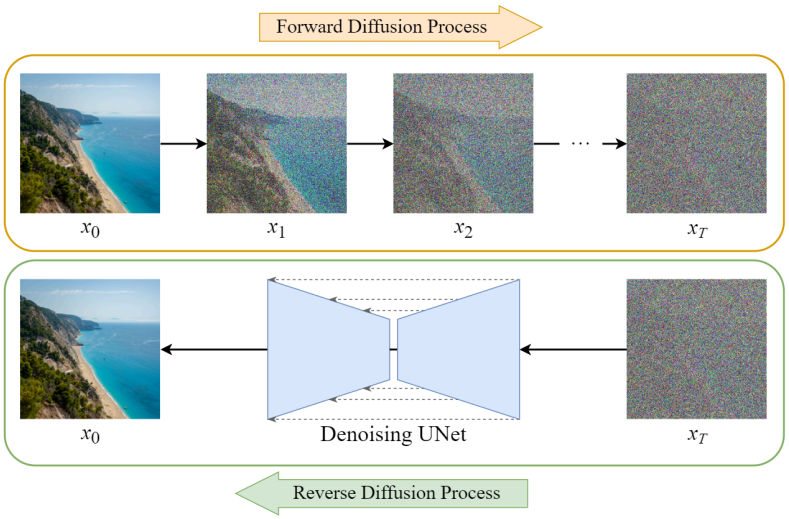



Figura 3.1: Funcionamiento de un modelo de difusión (Hernández, 2023)

U-Net es una red neuronal convolucional desarrollada para la segmentación semántica de imágenes biomédicas, aunque también es empleada en modelos de difusión para eliminar iterativamente el ruido en imágenes.

La segmentación semántica consiste en el etiquetado de cada píxel de una imagen con una clase referente a lo que se está representando. U-Net es de las redes neuronales más utilizadas para conseguir esto.

La arquitectura de U-Net consiste en una ruta de contracción y otra de expansión. Durante la contracción, la información espacial se reduce mientras que la información de las características aumenta. Es decir, se reduce la imagen pero se aumenta la información del contexto. Durante la expansión, se combina la información espacial y las características, donde se representan las características y el tamaño de la trama aumenta gradualmente. (U-Net, 2024.).

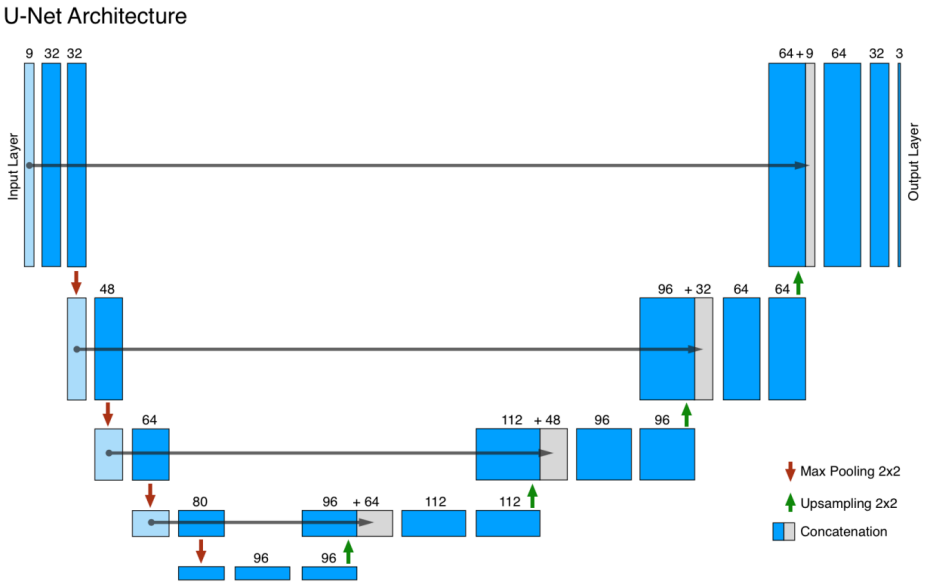



Figura 3.2: Arquitectura de la redes U-Net (Liani, s. f.).

El ajuste fino, conocido como fine-tuning, es una técnica fundamental en el ámbito de la inteligencia artificial (IA) que permite adaptar modelos pre-entrenados a tareas o dominios específicos. Esta estrategia ha demostrado ser altamente efectiva para mejorar la capacidad de los modelos de IA en la generación de contenido basado en lenguaje natural. Cuando nos referimos a modelos pre-entrenados de IA, como el modelo GPT-3, que han sido entrenados con una gran cantidad de datos de texto, el ajuste fino brinda la oportunidad de personalizar y perfeccionar aún más su rendimiento. Esto se logra al entrenar el modelo con conjuntos de datos específicos y ajustar sus parámetros para adaptarlo a una tarea o dominio en particular.

El proceso de ajuste fino implica una etapa adicional de entrenamiento posterior al pre-entrenamiento inicial. Durante esta etapa, se utilizan conjuntos de datos específicos relacionados con la tarea o el dominio objetivo. El modelo pre-entrenado se adapta y aprende de los datos específicos de la tarea, lo que le permite comprender y generar contenido más relevante y coherente en relación con esa tarea específica.

Además del uso de datos específicos, el ajuste fino también implica la optimización de hiperparámetros y configuraciones del modelo para mejorar su rendimiento en la tarea objetivo. Esto puede implicar ajustar la tasa de aprendizaje, el tamaño de los lotes de entrenamiento, la arquitectura del modelo o la longitud de las secuencias utilizadas durante el entrenamiento.

El ajuste fino ha sido ampliamente utilizado con éxito en diversas aplicaciones de IA, como la generación de textos, la traducción automática, la respuesta a preguntas y otras tareas relacionadas con el procesamiento del lenguaje natural o el tema de este trabajo, la generación de imágenes. Al personalizar y adaptar los modelos pre-entrenados a tareas específicas, el ajuste fino se ha revelado como una técnica efectiva para mejorar la calidad y la relevancia de los resultados generados por los modelos de IA.

En el caso particular de Stable Diffusion, hay una técnica de ajuste fino llamada LoRA (Low-Rank Adaptation) (Hu, 2021). Esta técnica tiene la ventaja de que reduce significativamente el número de parámetros de entrenamiento. Además, tiene la particularidad de que solamente se entrena un pequeño conjunto de pesos, permitiendo un entrenamiento muy veloz, eficiente y que como resultado obtiene modelos ligeros. Por otro lado, estos modelos no se pueden usar independientemente, ya que requiere algún tipo de conexión con el modelo completo y original de Stable Diffusion.

### 3. 1.1. Tecnologías relacionadas con el modelo

El modelo de Inteligencia Artificial empleado es Stable Diffusion, en particular la versión 1.5. Se optó por usar este modelo ya que, al ser de código abierto, hay un gran volumen de información accesible sobre cómo funciona, el modo de entrenamiento y herramientas que facilitan el proceso.

Stable Diffusion es un modelo de aprendizaje automático desarrollado por Runway y LMU Múnich con la colaboración de Stability AI, la actual empresa relacionada al modelo. Además, es de código abierto y se puede utilizar para generar imágenes a partir de texto o modificar imágenes ya creadas.

Referente a su arquitectura, Stable Diffusion utiliza una variante de los modelos de difusión y consta de tres partes: Un *autoencoder* (VAE), U-Net y un codificador de texto. El modelo VAE (Variational AutoEncoder) (Kingma, 2013) tiene dos partes, un codificador y un decodificador. Durante el entrenamiento del modelo de difusión, el codificador convierte imágenes, por ejemplo de 512x512x3, en una representación latente con menor

dimensión, por ejemplo de 64x64x4, para el proceso de difusión hacia delante. Estas imágenes codificadas son denominadas *latentes*. Tras esto, se aplica la difusión hacia delante y hacia detrás, previamente explicadas (ver 3.1) y se obtiene otra imagen latente, obtenida en la última capa de la red U-Net. Esta nueva imagen latente se utiliza como entrada para el decodificador del VAE, obteniendo de nuevo una imagen. El proceso de difusión hacia atrás (eliminación de ruido) puede condicionarse de forma flexible a una cadena de texto, una imagen u otro tipo de información. Esta información sirve de entrada para la red U-Net. Para codificar el texto de entrada se utiliza el codificador CLIP (Radford, 2021). En la Figura 3.3 se aprecia un esquema de la arquitectura de Stable Diffusion (Mishra, 2023).

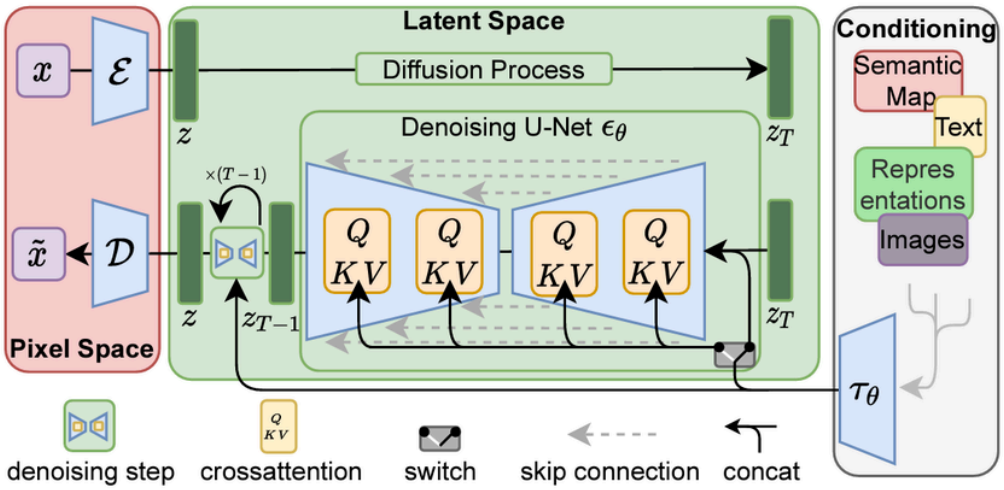



Figura 3.3: Arquitectura del modelo Stable Diffusion (Rombach, 2022).

Donde $x$ representa una imagen, ~$x$ representa la imagen generada, ε es el codificador del modelo VAE, Ɗ es el decodificador VAE, $z$ es una imagen latente y T$\Theta$ es el codificador CLIP.

Para entrenar el modelo se utilizó la herramienta OneTrainer (Nerogar, 2024), un programa de código abierto para todo el proceso de entrenamiento, ya que permite la generación de LoRAs, así como modelos completamente re-entrenados o embebidos, de una forma muy sencilla. Esta herramienta fue elegida por su interfaz gráfica y su sencillez, así como la posibilidad de generar diversos *samples* cada cierto tiempo, permitiendo observar la evolución del modelo. Estos *samples* funcionan mediante un *prompt* específico y utiliza el modelo a medio entrenar con dicha entrada, generando la imagen con la información que tiene actualmente el modelo.

### 3.1.2 Tecnologías empleadas que utilizan el modelo

Una vez entrenado el modelo, se empleó ComfyUI (Comfyanonymus, 2024) para utilizar el modelo, así como enlazarlo con la aplicación final. Esta herramienta, también de código abierto, es un *GUI* (Graphical User Interface) y *backend* de Stable Diffusion modular. Permite cargar diversos modelos de generación de imágenes, introducir varios *prompts* y *prompts* negativos, así como posibilidades de hacer varias tareas relacionadas con este ámbito. Entre ellas se encuentra *text2img*, *img2img* o *inpainting*. Además, la interfaz modular a modo de grafos/nodos es muy fácil de usar. Se decidió usar esta herramienta dada su potencia y la capacidad de permitir el uso de interfaz gráfica y APIs.

Para el desarrollo de una aplicación web que conecta ComfyUI se utilizó Flask. Se utilizó por su sencillez y porque está basado en Python, lenguaje que se usa en la documentación de la API de ComfyUI. Otro motivo para su uso fue que se permite el desarrollo de una API y una página web de forma simultánea. Además, se empleó el uso de la librería OpenCV para el análisis de la imagen generada. Por otro lado, se empleó HTML5 y CSS para la interfaz gráfica de dicha aplicación.

El producto final utiliza Unity debido a que es un motor de videojuegos conocido y utilizado previamente en el transcurso del grado.

### 3.1.3 Otras tecnologías

Aparte de estas tecnologías, también se han empleado herramientas como Git con GitHub y Google Drive para el almacenamiento y seguimiento del proyecto. Se decidió usar GitHub para almacenar los diversos códigos fuentes empleados durante el desarrollo del proyecto y Google Drive para todo lo referente a la documentación y diversas actas.

Por último, el *IDE* (Integrated Development Environment) empleado ha sido Visual Studio Code, dada su versatilidad y sencillez. El sistema operativo empleado para la ejecución de todas las herramientas anteriormente nombradas ha sido Windows 10.

## 3. 2 Metodología de trabajo

En este apartado se describe la metodología utilizada para el desarrollo del proyecto, detallando las fases y las técnicas empleadas para la planificación, implementación y evaluación del proyecto.

### 3.2.1. Enfoque metodológico

Para el desarrollo de este proyecto, se ha seguido una metodología ágil con adaptaciones personalizadas. La flexibilidad de la metodología ágil permitió acomodar las iteraciones necesarias para el desarrollo del modelo de IA y la integración con Unity. Las reuniones quincenales con los tutores sirvieron como puntos de control y retroalimentación continua.

### 3.2.2. Fases del proyecto

El proyecto se ha dividido en varias fases, cada una con tareas y objetivos específicos:

- Planificación y recolección de requisitos:
    - Reuniones iniciales: Se realizaron reuniones con los tutores para definir los objetivos y el alcance del proyecto, así como para establecer las expectativas y los criterios de éxito.
    - Definición del alcance: Se establecieron los límites del proyecto y se especificaron las funcionalidades principales, incluyendo la generación de imágenes por IA y su representación tridimensional en Unity o las alternativas similares.

- Pruebas iniciales:
    - Investigación y análisis: Búsqueda intensiva de diversos modelos de generación de imágenes, así como diferentes algoritmos, herramientas o repositorios que permitan el reentrenamiento del modelo.
    - Entrenamiento de prueba: Ejecución de algoritmos, herramientas y repositorios para conocer la factibilidad de uso de cara a todo el proyecto.
    - Validación: Se comprueba si se puede ejecutar, los resultados y aquellos con problemas se intentan resolver para ejecutarlo.
- Desarrollo del conjunto de datos:
    - Investigación y análisis: Se investigaron diversas técnicas para la generación de conjuntos de datos adecuados para entrenar el modelo de IA basado en Stable Diffusion.
    - Generación del conjunto de datos: Se desarrolló un *script* en Python para generar un conjunto de imágenes con sus descripciones correspondientes. Este *script* fue iterativamente mejorado para optimizar la calidad del conjunto de datos.
- Entrenamiento del primer modelo:
    - Entrenamiento inicial: Se utilizó el conjunto de datos generado para entrenar un modelo temporal de IA. Este modelo sirvió como punto de partida y referencia para futuras mejoras.
    - Validación inicial: Se realizaron pruebas iniciales para validar el desempeño del modelo temporal y se identificaron áreas de mejora.
- Desarrollo de la aplicación web con Flask:
    - Diseño de la aplicación: Se diseñó una arquitectura para la aplicación utilizando Flask, asegurando una estructura modular y escalable.
    - Implementación: Se desarrollaron las funcionalidades básicas de la aplicación, incluyendo la interfaz de usuario, la API y la integración con el modelo de IA.
    - Pruebas y validación: Se realizaron pruebas continuas para asegurar la funcionalidad y usabilidad de la aplicación web, ajustando el desarrollo conforme se recibía retroalimentación.
- Desarrollo del proyecto en Unity:
    - Integración inicial con Unity: Se inició el desarrollo en Unity para representar tridimensionalmente las imágenes generadas por el modelo de IA.
    - Iteraciones de desarrollo: Durante el desarrollo en Unity, se hicieron mejoras continuas en la aplicación web para garantizar la compatibilidad y funcionalidad integradas.
    - Pruebas de integración: Se realizaron pruebas para asegurar que las imágenes generadas pudieran ser representadas correctamente en un entorno tridimensional.
- Mejora del conjunto de datos y entrenamiento final:
    - Mejora del *script* de generación del conjunto de datos: Se revisó y mejoró el *script* de generación de datos para aumentar la calidad y relevancia del conjunto de datos.
    - Entrenamiento del modelo final: Utilizando el conjunto de datos mejorado, se entrenó el modelo de IA final.
    - Validación y evaluación: Se llevaron a cabo pruebas para validar el desempeño del modelo final y asegurar que cumpliera con los requisitos del proyecto, comparándolo con el modelo original.

### 3.2.3. Evaluación y revisión

- Reuniones quincenales: Reuniones regulares con los tutores para mostrar avances, recibir retroalimentación y resolver dudas.
- Evaluaciones iterativas: Al final de cada fase, se realizaron evaluaciones para asegurar el cumplimiento de los objetivos y se hicieron ajustes según fuera necesario.
- Revisión final: Al completar el proyecto, se llevó a cabo una revisión para validar todos los componentes del sistema y asegurar su funcionamiento integrado.

## 3.3 Conjunto de datos

Desde el principio del desarrollo se optó por la generación de mapas de colores donde cada color representa un objeto, donde el suelo es representado por el azul oscuro, las mesas por el verde, las sillas por el amarillo y las puertas por el magenta. Estos colores se eligieron automáticamente tratando de distribuir los 4 elementos por todo el espacio de color RGB. Dada la naturaleza prototípica del proyecto solo se desarrollaron estos 3 muebles. Además, se implementó mediante el paradigma de programación orientada a objetos, donde cada habitación es un objeto y mediante métodos se añaden nuevos muebles, se genera la descripción y la imagen.

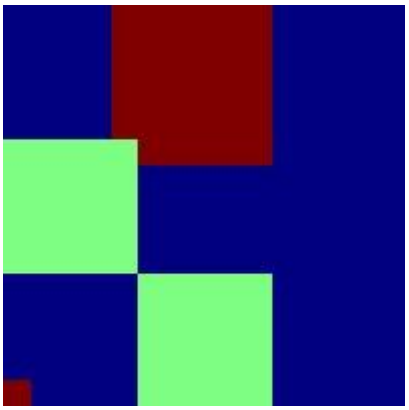


Figura 3.4: Ejemplo de la primera versión del conjunto de datos.

*Prompt: Overhead shot of a square room where there ar and 2 tables at center, center left, 2 doors at center, bottom left.*

A la hora de crear el conjunto de datos, primero se genera la imagen y posteriormente la descripción. La imagen representa una habitación, que es una matriz de valores enteros, donde dependiendo del valor de la casilla se sabe qué elemento es y mediante el número de posiciones contiguas con el mismo valor se sabe el tamaño del mismo. Para indicar la posición, se dividió la habitación en 9 partes iguales, haciendo un espacio de tamaño 3x3 bloques. Originalmente no había ninguna restricción a la hora de colocar los diversos muebles en el mapa, pero se optó por limitar el tamaño máximo y mínimo de los elementos a insertar. Posteriormente, se decidió prohibir la generación de puertas si no estaban contiguas a un lateral. Finalmente, dado que la superposición de objetos generaba problemas en el proyecto de Unity, se decidió eliminar la posibilidad de que hubiesen elementos superpuestos en un mismo bloque. Además, se emplea la aleatoriedad, ya que cuando se decide insertar un elemento, este es generado con

tamaños y posiciones pseudoaleatorios dentro del bloque deseado y dado un tamaño relativo.

Una vez obtenida la matriz deseada, se genera la descripción, siempre en inglés, mediante un análisis de la matriz y un inicio pseudoaleatorio. Este comienzo siempre contendrá un *token* identificativo y único “TFGPabloMap”. Esto se decidió porque la presencia de un *token* único permite que el modelo sea capaz de diferenciar conceptos existentes de las particularidades del conocimiento a inferir (Santana, 2022). Si no se pusiera un *token* único e identificativo, al indicar, por ejemplo, *table* podría generar cualquier tipo de mesa, de cualquier forma y color. Al incluir dicho *token,* se le indica al modelo que las mesas se deberán generar siempre como cuadrados del mismo color.

Tras el identificador, se genera un inicio donde aleatoriamente se combinan diferentes sintagmas para obtener diferentes inicios coherentes, permitiendo al modelo más comprensión y un abanico más amplio de posibles descripciones. Los sintagmas que se eligieron son “*map”,* "*of a*", “*square*”, “*room*”, “*where there*” y “*with*”. Se establecen varias configuraciones donde según una probabilidad, estos sintagmas se añaden o no, a excepción de “*where there*” y “*with*”, que si se añade uno no se puede añadir el otro.

Para la versión final, se optó por la adición del término “*only*”, ya que el modelo generaba más elementos de los deseados. Algunos ejemplos de inicio pueden ser:

*TFGPabloMap with only…*

*TFGPabloMap map of a square room where there is only…*

Tras el inicio, comienza la descripción de la habitación en sí. Se recorre la matriz y se inserta el objeto, su tamaño y posición. Para mayor entendimiento del modelo y más flexibilidad a la hora del uso del mismo, el tamaño y posición se añaden o no según el azar.

Una vez sabido cómo se genera una imagen con su descripción, se desarrolló una función capaz de generar una serie de habitaciones con sus descripciones en archivos con extensión JPG y TXT. Utilizando esta función se generó un conjunto de datos de 150.000 pares de imagen-descripción. Se decidió la generación de un conjunto tan grande ya que, al ser imágenes tan distintas a la realidad, el modelo debe hacer un gran esfuerzo para identificar y comprender que un mueble se representa como un cuadrado de color.

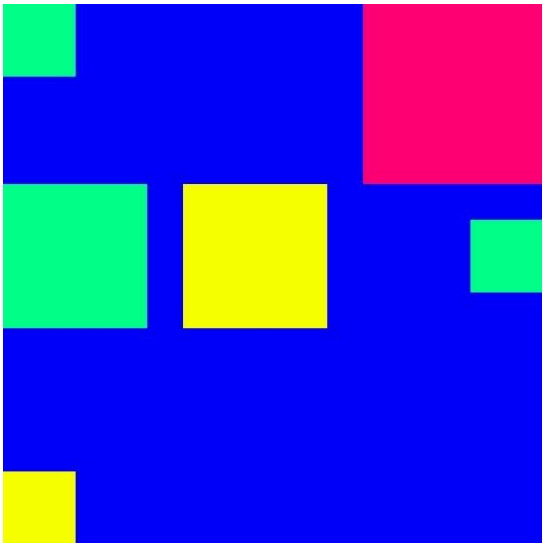


Figura 3.5: Imagen del conjunto final de entrenamiento y su descripción.

*TFGPabloMap square room with only a medium table in center right, a very big table in center left, a table in top left, a chair in center, a chair in left bottom and a very big door in top.*

El código que genera el conjunto de datos se puede encontrar en el repositorio de este proyecto en GitHub (Pérez, 2024).

## 3.4 Modelo empleado

El modelo empleado durante este proyecto fue un LoRA (Hu, 2021) de Stable Diffusion 1.5. Se decidió usar un LoRA porque, haciendo una relación entre el resultado, el tiempo de entrenamiento y la facilidad de uso, es la opción ideal, ya que genera buenos resultados en un tiempo de entrenamiento relativamente aceptables. Originalmente se intentó emplear un modelo tipo DALL-E entrenado manualmente desde cero mediante un cuaderno Jupyter en Google Colab, pero la incompatibilidad de versiones de las librerías y todo el tiempo invertido en tratar que funcionase hizo que se optase por el uso de OneTrainer (Nerogar, 2024) para modelos Stable Diffusion, ya que esta aplicación de escritorio no generó ningún problema a la hora de ejecutarse.

El conjunto de entrenamiento obtenido es de 150.000 elementos. Dado el tamaño del conjunto de datos, se decidió que se realizaran únicamente 4 épocas de entrenamiento.

Finalmente, el modelo resultado es capaz de comprender y generar correctamente la imagen deseada, aunque con un fallo: al modelo se le solicita que genere una serie de mobiliario en la habitación, pero no que el resto de la habitación permanezca vacía, por lo que usualmente añade puertas, mesas o sillas no indicadas en el texto. Es por esto de la existencia del *token* "*only*" en las descripciones de las imágenes de entrenamiento, pero el resultado tras un nuevo entrenamiento fue inservible, ya que el modelo entrenado con esta adaptación no fue capaz de inferir información relevante. Por eso, el modelo final no contiene las últimas modificaciones del conjunto de datos: impedir el solapamiento y la existencia del *token* "*only*".

Una de las ventajas principales de haber usado OneTrainer es la facilidad de su uso. Todo el proceso de entrenamiento se puede ejecutar desde una interfaz gráfica y permite, cada un cierto número de pasos indicados, probar el modelo mientras se entrena, permitiendo observar su avance. En la Figura 3.6 se puede observar la interfaz para entrenar el modelo donde se introducen todos los hiperparámetros del entrenamiento. Los más destacados son el optimizador, el tamaño de los lotes de entrenamiento, el número de épocas de entrenamiento a realizar o la configuración de las redes U-Net.

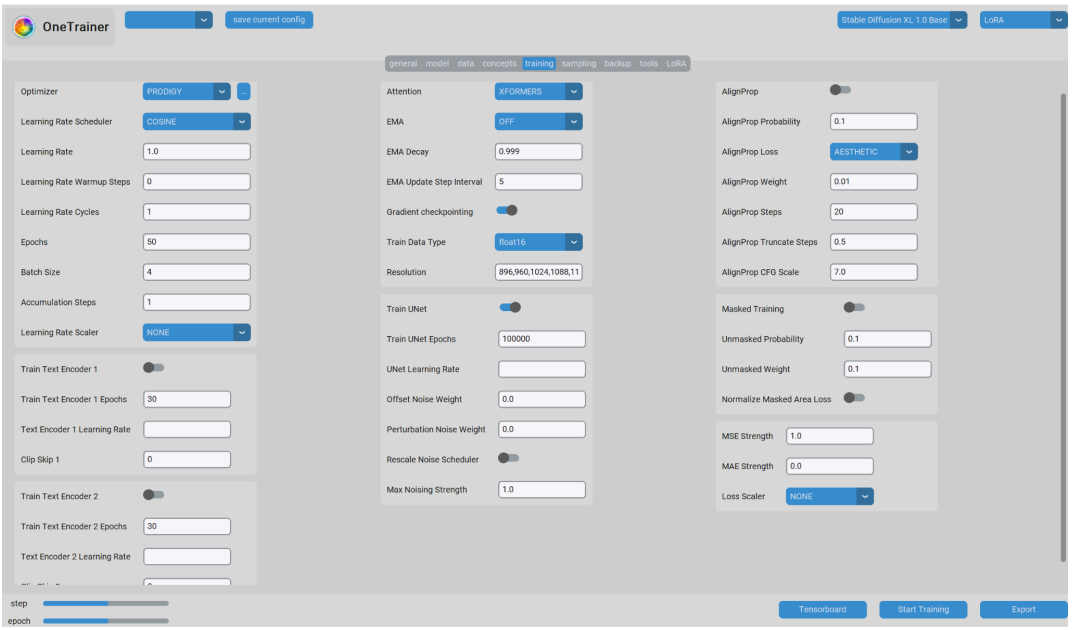



Figura 3.6. Configuración del entrenamiento para un modelo (Nerogar, 2024).

Por otro lado, en la Figura 3.7 se muestra la vista de la pestaña de pruebas, donde se puede observar la capacidad de generar imágenes del modelo en entrenamiento. Asimismo, tras introducirse una descripción a tratar, genera imágenes. Esta habilidad puede ejecutarse de dos maneras: se puede generar una única imagen, como en la figura, o indicar que cada ciertas iteraciones se generen automáticamente, permitiendo la posibilidad de ver la mejora del modelo.

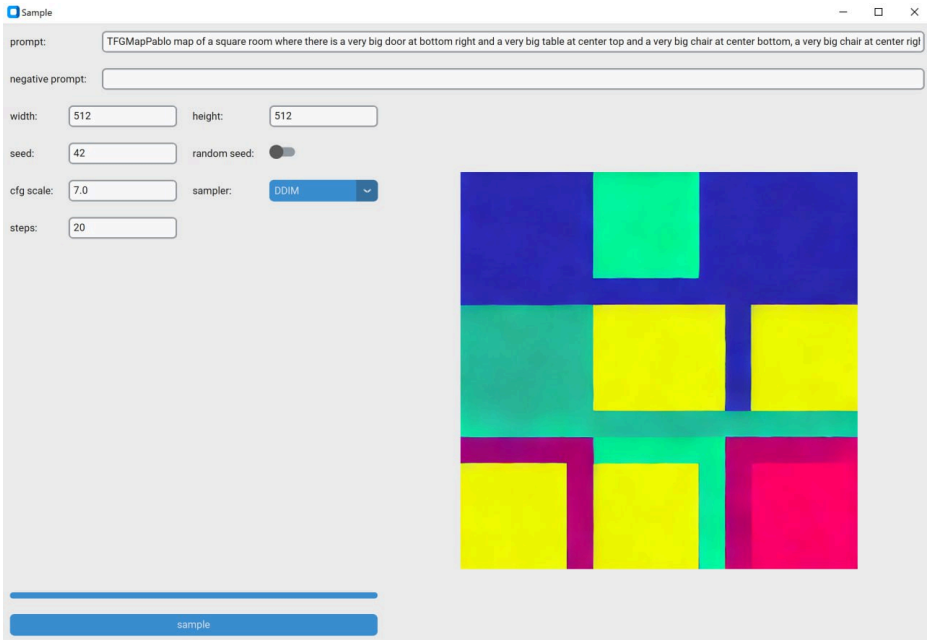



Figura 3.7. Interfaz del apartado de la pestaña de pruebas.

## 3.5 Evolución de la interfaz conectada al modelo

En relación con el desarrollo de la interfaz conectada al modelo, se encontraron gran cantidad de alternativas. Sin embargo, se optó por un proyecto en Unity capaz de generar una habitación de forma tridimensional, conectado con el modelo mediante una aplicación web. El desarrollo se planteó de la manera más modular posible, de forma que cada parte sea totalmente independiente y fácilmente escalable.

En primer lugar, se generó un *workflow* en ComfyUI capaz de cargar el modelo, el LoRA y mediante una descripción generar una imagen. El diagrama resultante se puede ver en la Figura 3.8, donde se aprecia la conexión de los diferentes nodos. En ellos, se puede observar cómo el flujo comienza cargando el modelo Stable Diffusion y enlazando el LoRA. Tras esto, se tiene dos nodos referentes a las descripciones (positivas y negativas) conectadas al modelo y a un nodo denominado *KSampler*, el encargado de generar la imágen. Este nodo recibe como entrada un modelo ajustado (LoRA), las dos descripciones y una imagen latente vacía, se encarga de juntar toda la información y generar una imagen latente. Una vez obtenida, se emplea un nodo que la decodifica para su posterior visualización. Tras esto, ya se permite la generación de imágenes.

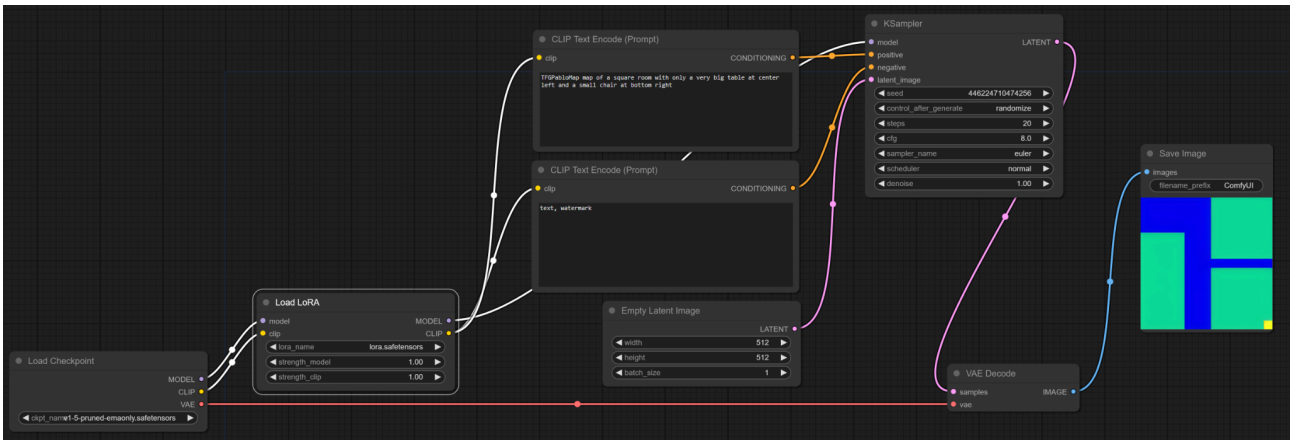



Figura 3.8: Diagrama de ComfyUI que permite la generación de imágenes

Posteriormente, se obtuvo el *workflow* resultante en formato JSON, necesario para la API de ComfyUI que se utiliza en la aplicación Flask encargada de gestionar la creación de imágenes. Se desarrolló una interfaz web (Figura 4.1) donde se permite introducir el texto y se descarga la imagen generada, pero también se desarrolla una API que permite una solicitud POST con el texto y devuelve un objeto JSON con la información de todos los elementos (Figura 3.9). Esta información se obtuvo mediante un análisis con OpenCV de la imagen generada. Primero, se creó una máscara binaria para cada tipo de mueble, es decir una imagen donde los píxeles con valor distinto de cero representan un tipo de mueble, permitiendo separar la información según el mobiliario. A cada máscara se le aplicaron operaciones morfológicas (morfología matemática, 2021), ya que los elementos contiguos finalmente se fusionan en Unity, así que se intentan modificar los bordes de cada elemento para separar elementos similares contiguos. Las operaciones aplicadas fueron de apertura (*cv2.morphologyEx(máscara, cv2.MORPH_OPEN)* en Python) seguida de una de cierre (*cv2.morphologyEx(máscara, cv2.MORPH_CLOSE)*), ya que la apertura permite eliminar píxeles distintos de cero que tengan vecinos próximos iguales a cero, y separar así el mobiliario al eliminar el ruido y abrir pequeñas conexiones entre los objetos, mientras que el cierre puede recuperar parte de los detalles de los objetos que se perdieron durante la apertura. Luego, se obtuvieron los contornos utilizando la función *cv2.findContours()* y finalmente los centroides gracias a los momentos de la imagen (momentos de imagen, 2023). Con estos datos se generó un diccionario de Python con la siguiente información: posición de la esquina superior izquierda de cada mueble, la altura, la anchura y la posición del centroide (Figura 3.10). La aplicación podría ser utilizada como resultado final, pero debido al deseo de recrear la imagen tridimensionalmente, se creó un proyecto en Unity.

```
Unset
def index():
  Si es una solicitud POST:
    is_web_request = False;
    try:
      Se intenta obtener el prompt por petición API;
    except:
      Si da error se lee desde la web;
      is_web_request = True;

    Se carga el JSON con el workflow de ComfyUI;
    Se ajusta el JSON para que el prompt tenga el texto del usuario;
    Se envía la solicitud a ComfyUI con el JSON adaptado;
    De la respuesta, se obtiene el ID único de la imagen a generar;
    Se espera hasta que la imagen esté generada y se almacena;
    Se aplica el análisis de la imagen que obtiene las coordenadas de cada
mueble;
    Si is_web_request:
      Se descarga la imagen;
    Si no:
      Se devuelve un JSON con la información de los muebles;
    Fin if
  Fin if
Fin método
```

Figura 3.9. Pseudocódigo del método principal de la aplicación Flask.

```python
Python
import cv2

def imageTreatment(filename):
    frame = cv2.imread(filename)

    # Yellow
    chair_lower_color_bounds = (0,150,150)
    chair_upper_color_bounds = (100,255,255)

    # Masks
    try:
        chair_mask = cv2.inRange(frame, chair_lower_color_bounds,
                                 chair_upper_color_bounds)
    except Exception as err:
        raise RuntimeError('Error creating the masks', err.args)

    # op. morfologicas
    chair_mask_processed = cv2.morphologyEx(chair_mask, cv2.MORPH_OPEN, None)
    chair_mask_processed = cv2.morphologyEx(chair_mask_processed,
                                            cv2.MORPH_CLOSE, None)

    # Apply contours and centroid for each mask
    try:
        chair_regs = regionProps(chair_mask_processed.copy())
    except Exception as err:
        raise RuntimeError('Error getting region props of chair', err.args)

    regs_data = {
        'chairs': chair_regs['data'],
    }
```

```
    return regs_data


# Returns the posittion and dimensions of the contours and its centroid
def regionProps(mask):
    contours, hierarchy  = cv2.findContours(mask, cv2.RETR_EXTERNAL,
                                            cv2.CHAIN_APPROX_SIMPLE)
    mask_copy = cv2.cvtColor(mask, cv2.COLOR_GRAY2BGR)  # Convert to colour
    contours_data = []
    for contour in contours:
        # Calculate area of the region
        area = cv2.contourArea(contour)
        if area > 0:
            # Calculate bounding rectangle dimensions
            x, y, w, h = cv2.boundingRect(contour)

            # Calculate perimeter of the region
            perimeter  = cv2.arcLength(contour, True)

            # Calculate centroid of the region
            M = cv2.moments(contour)
            centroid_x  = int(M["m10"] / M["m00"])
            centroid_y  = int(M["m01"] / M["m00"])

            # Create a dict of the result data
            contours_data.append({
                'x': x,
                'y': y,
                'w': w,
                'h': h,
                'centroid_x ': centroid_x ,
                'centroid_y': centroid_y
            })
    return {
        'mask': mask_copy,
        'data': contours_data
        }
```

Figura 3.10. Código Python encargado del análisis de la imagen simplificado para un único mueble.

El proyecto en Unity está conectado a la API en Flask mediante un *script* que se aplica en un objeto vacío, donde se define la clase ComfyPOST, empleada en todo el proyecto. Para conectarse a la aplicación Flask y hacer una solicitud POST, ComfyPOST debe tener un método que devuelva un objeto de tipo *IEnumerator*, pues la llamada a la API se hace empleando una corrutina y en su interior ejecutar la función *UnityWebRequest.Post()*.

```
C/C++
using UnityEngine;
using UnityEngine.Networking;
using System.Collections;
using System.Collections.Generic;

public class ComfyPOST : MonoBehaviour {
    public string apiUrl = "http://127.0.0.1:5000/test-api";
    public string prompt = "";
```

```
    void Start() {
        string json = "{\"prompt\": \"" + prompt + "\"}";
        StartCoroutine(Upload(json));
    }

    IEnumerator Upload(string json) {
        using (UnityWebRequest www = UnityWebRequest.Post(apiUrl, json,
               "application/json")) {
            Debug.Log("Sent POST request");
            yield return www.SendWebRequest();

            if (www.result != UnityWebRequest.Result.Success) {
                Debug.LogError(www.error);
            } else {
                Debug.Log("Received POST data");
                byte[] bytes_data = www.downloadHandler.data;
                string jsonString =
                       System.Text.Encoding.UTF8.GetString(bytes_data);
                RoomData data =
                         JsonConvert.DeserializeObject<RoomData>(jsonString);

                // Resto del programa en el caso de obtener los datos
                // correctamente
            }
    }
}
```

Figura 3.11. Llamada a una API con Unity.

Además de los atributos observables en la Figura 3.11, este *script* tiene otros relevantes. Entre ellos están la escala a generar los objetos, el factor de escala de toda la habitación, los diversos *prefabs* de los muebles y el tamaño de la habitación total (sin tener en cuenta el factor de escala) que debe ser del tamaño en píxeles de las imágenes generadas.

```C/C++
public string apiUrl = "http://127.0.0.1:5000/test-api";
public string prompt = "TFGMapPablo map of a square room with a very big door at
left, a table at center and a chair at top left, a small door at center right";
public GameObject chairPrefab;
public GameObject doorPrefab;
public GameObject tablePrefab;
public float scale = 1f;
public float scaleFactor = 50f;
public float roomHeight = 512f;
public float roomWidth = 512f;
```

Figura 3.12. Atributos de la clase ComfyPOST y sus valores predeterminados.

Una vez se ejecuta el entorno, se realiza una solicitud POST a la aplicación Flask con la descripción de la habitación deseada. Al recibir el objeto JSON, este se recorre generando instancias de los *prefabs* en las coordenadas indicadas y tiene en cuenta la escala y el factor de escala deseados. Gracias a los datos recibidos, es fácil conocer la posición y tamaño de los objetos, pero la rotación no está incluida, así que se ideó según el tipo de mueble. En el caso de las mesas, al ser elementos simétricos, se obvia este paso. Por otro lado, se comprueba la posición de la puerta y según el borde de la habitación que esté en contacto, se rotará de forma que la puerta se abra hacia el interior de la habitación. Las sillas, sin embargo, calcularán cuál es la mesa más cercana y se rotarán de forma que estén orientadas en dirección a la mesa lo máximo posible, pues se restringió la rotación de la silla en múltiplos de 90º para mantener la perpendicularidad y ser paralelas con las paredes de la habitación.

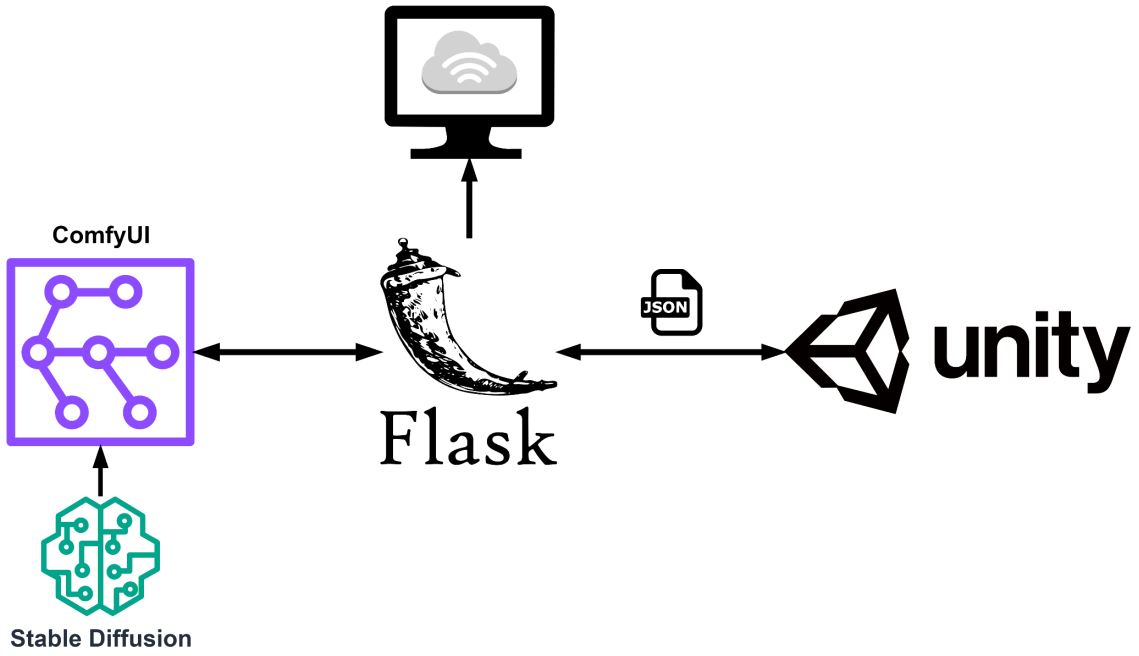



Figura 3.13: Arquitectura completa del proyecto

# Capítulo 4 Resultados

En el presente capítulo se muestran diversos resultados, tanto la imagen generada por ComfyUI como el resultado en el proyecto de Unity. De todos los entrenamientos del modelo realizados, el obtenido tras insertar el *token* "*only*" no generó resultados concluyentes, pues no fue capaz de inferir el conocimiento y generar nada de lo solicitado en las descripciones. Por esto, se empleó una versión del modelo anterior, aunque se decidió aplicar descripciones como si dicho entrenamiento hubiera sido exitoso. Para un resultado más atractivo, se creó un entorno con paredes y suelos texturizados en Unity. El tiempo requerido para generar cada imagen se calculó en base al ordenador que se utilizó para el desarrollo de todo este proyecto. Las características del mismo son:

- Procesador Intel Core i5 4460 - 3.2GHz.
- Tarjeta gráfica NVIDIA GeForce GTX 1050Ti.
- 12GB DDR3 de memoria RAM.
- Placa base Acer Aspire XC-705.

Como se comentó en el capítulo anterior, en la aplicación Flask también se desarrolló una interfaz web, la cual tiene un cuadro de texto donde introducir la descripción de la imagen a generar y, tras darle al botón "Enviar", se crea la imagen y se descarga automáticamente.


Figura 4.1. Vista de la aplicación Flask en el navegador.

## Pruebas

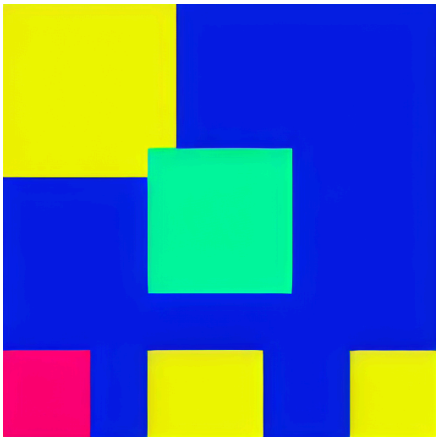
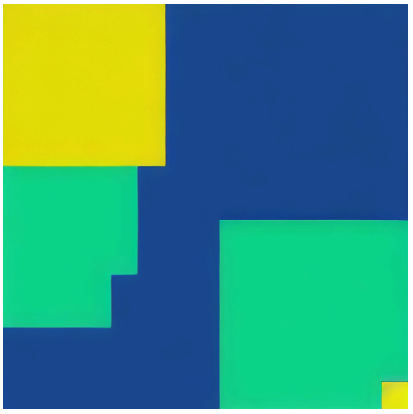
<table>
<tr><td colspan="2">Prompt: TFGPabloMap map of a square room with only a very big table at center, a medium chair at bottom right and a big door at bottom left</td><td>Tiempo:<br>109.01s</td></tr>
<tr><td>Imagen generada:<br>  </td><td colspan="2">Habitación generada:<br>  </td></tr>
<tr><td colspan="3">Resultado:<br>  </td></tr>
<tr><td colspan="3">Observaciones: Se añaden dos sillas no solicitadas: la de la parte inferior central y la superior izquierda. De resto, se aprecia la correcta rotación de la puerta, así como de las dos sillas inferiores.</td></tr>
</table>

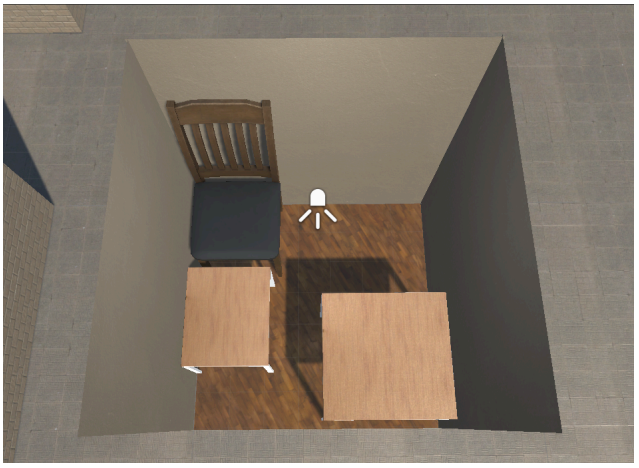
Tabla 4.1: Resultados de ejecución 1

| Prompt: TFGPabloMap map of a square room with only a very big table at center left and a small chair at bottom right | Tiempo: 110.66s |
|---|---|

Imagen generada:



Habitación generada:



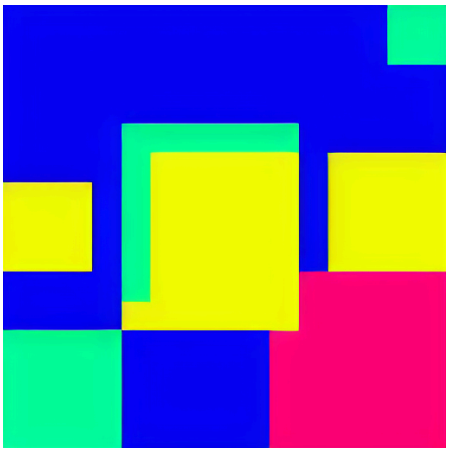
Resultado:

Observaciones: Se añade la mesa inferior derecha y la silla de la parte superior. La silla solicitada pequeña en la parte inferior derecha existe, aunque tan pequeña que es difícil de apreciar. Además, la imagen generada tiene tonos ligeramente diferentes a las demás generaciones.

Tabla 4.2: Resultados de ejecución 2

| Prompt: TFGPabloMap map of a square room with a big table at center, a chair at left center, a chair at right center, a medium table at top right | Tiempo: 110.20s |
|---|---|

Imagen generada:



Habitación generada:

Resultado:

Observaciones: Como se puede observar, genera tres elementos de más: la puerta, una mesa y una silla que genera solapamientos. Además, la imagen generada contiene otra pequeña silla en el centro que, dado que está contigua a la central, el análisis de la imagen lo interpretó como una única silla.

Tabla 4.3: Resultados de ejecución 3

# Capítulo 5 Conclusiones y líneas futuras

En conclusión, este proyecto ha permitido generar un escenario de una habitación en un entorno tridimensional, cumpliendo los objetivos exigidos: construir un conjunto de datos, entrenar un modelo de generación de imágenes y la implementación de una herramienta que lo utilice. Sin embargo, está lejos de un uso práctico o comercial. Como se puede observar en el Capítulo 4, se genera correctamente una habitación con los muebles solicitados, pero se añaden elementos no solicitados. Esto demuestra que el sistema desarrollado es un prototipo útil, requiriendo un desarrollo de un mejor modelo. Sin embargo, se reflejan usos relevantes a la hora de generar automáticamente escenarios, por ejemplo, para un videojuego de múltiples habitaciones o de tipo *roguelike*, permitiendo la generación del entorno/nivel.

Originalmente, este modelo iba a emplear una replicación de DALL-E de código abierto, que mediante Google Colab se entrenaría y permitiría la posibilidad de obtener un modelo entrenado desde cero, tipo de modelo ideal para un conjunto de entrenamiento tan diferente a lo conocido. Sin embargo, esta replicación estaba obsoleta y sin el correcto mantenimiento, por lo que se dificultó gravemente hallar la versión específica de Python, así como las de todas las librerías necesarias. Pese haber encontrado las versiones y desarrollar un script en el Colab del proyecto que instalaba todas las dependencias y la versión de Python específica, se actualizaban diferentes librerías cada poco tiempo, de forma que la ejecución de esta replicación, usada en TFGs anteriores, fue imposible.

Por un lado, el entrenamiento del modelo no es perfecto. Pese a que se genera lo solicitado, también se añaden elementos extras. Este inconveniente ocurre debido a que las imágenes son muy diferentes a lo que el modelo original conoce y le resulta muy difícil inferir el conocimiento. Como se comentó en capítulos anteriores, con la adaptación de indicar que únicamente se creen los muebles solicitados, el modelo resultó inservible. Sin tener esto en cuenta, el resultado sería el ideal, pero dada la importancia del modelo en el proyecto, se impide un uso más amplio o comercial. Una forma de resolver este problema habría sido generar un modelo completo, en vez de un LoRA, ya que no tiene conocimientos del que partir y tendría una mayor capacidad de aprendizaje, pero esta solución fue imposible en este Trabajo Final de Grado porque eso requeriría una potencia de cómputo inalcanzable para el alumno o inversión económica. Otra solución alternativa más simple sería la eliminación de lo referente al modelo de generación de imágenes (Stable Diffusion y ComfyUI) y sustituirlo por un modelo LLM (Large Language Model) (modelo de lenguaje grande, 2024) de código abierto, como Llamafile (Mozilla, 2024). En este caso, se le enviaría un mensaje donde se requiera un objeto JSON con la información de la imagen y demás datos para generar la imagen mediante OpenCV u otras opciones como ImageMagick. Tras esto, se adaptaría la aplicación Flask y, gracias a la modularidad, el proyecto en Unity se mantendría intacto.

Por otro lado, dada la cantidad de servicios necesarios para ejecutar esta herramienta, si se desea añadir como *asset* en la Unity Asset Store, debería eliminarse la posibilidad de modificar la ruta de la API y conectándose a la página donde esté alojada la API desarrollada en Flask automáticamente.

En tercer lugar, la página web implementada para generar imágenes es bastante simple, ya que solamente descarga la imagen tras crearla. ComfyUI permite muchas operaciones con imágenes ya creadas que mediante su API se podrían implementar, de forma que el usuario pueda usarlas desde la página web. Entre todas las posibilidades, las más llamativas son modificar una imagen mediante un *prompt*, *inpainting* (permitiendo la modificación de las imágenes generadas) o composición por áreas (permitiendo múltiples entradas de texto y, quizás, múltiples habitaciones).

En definitiva, este prototipo es una demostración de la potencia y versatilidad que tienen los modelos de generación de imágenes y cómo pueden aplicarse para obtener entornos virtuales. Gracias a su modularidad, hay infinitas líneas futuras de investigación. Quizás, en un futuro, podría generarse un entorno completo, facilitando muchísimo la labor de diseño.

# Capítulo 6 Summary and Conclusions

In conclusion, this project has allowed us to generate a scene of a room in a 3D environment, meeting the required objectives: create a training dataset, train a image generator model and the development of any software that uses it. However, it is far from commercial or practice usage. As you can see in Chapter 5, the room is well generated with the furniture required, but unwanted elements are added. This proves that the developed system is a useful prototype, requiring a better model. Nevertheless, there is an outstanding usage to automatically generate scenarios, per example, for a videogame with multiple rooms or a kind of roguelike game, allowing the generation of the environment/level.

Originally, the model was an open source DALL-E replication and through Google Colab it would be trained and enabled the possibility of getting a model trained from zero, the perfect type of model for the dataset because it is very different from the well known elements. Unfortunately, this replication is obsolete and without the correct maintenance, so getting the appropriate Python's and library's version was very difficult. Despite having developed a Colab notebook that installs all the dependencies and the correct Python version, new libraries were updated frequently, so the use of this replication, used in previous TFGs, was impossible.

On one hand, the model training is not perfect. It generates the required furniture, but also unrequired elements. This obstacle happens because the training images are very different from what the model knows and it is hard to infer knowledge. In previous chapters, it was mentioned that the adaptation of indicate that only create the required elements made an useless model. Without taking this into account, the result would be perfect, but with the high importance of the model in this project, it is impossible to use it for commercial or wider use. The way to solve this problem is training a complete model from zero and not using a LoRA because the empty model doesn't have any knowledge to start with and has more capacity to learn, but this solution was impossible because that requires a computing power or an economic inversion unreachable for the student. Another simpler solution is the removal of the generating images model (Stable Diffusion and ComfyUI) and substitute them for an open source LLM model (Large Language Model) (modelo de lenguaje grande, 2024), like Llamafile (Mozilla, 2024). In this case, there would be a message sent to the model that would require a JSON object with the information of the image and all the data for the generation using OpenCV or another software like ImageMagick. After that, the Flask app would be adapted and, thanks to the modular development, the Unity project would be unmodified.

On the other hand, there are a lot of necessary services to run the tool. If there was an intention to add it as an asset to the Unity Asset Store, the possibility of modifying the API path in the Unity project must be deleted and the connection to the Flask API would be automatically done.

In third place, the developed website that generates the images is very simple, the unique option is to generate them. ComfyUI is able to do a lot of operations with images via API and this tools could be an interesting expansion to the website. Some of the most striking possibilities are to modify the generated image through another prompt, inpainting (enabling the modification of the generated images) or the area composition (making possible to use multiple prompts and, maybe, multiple rooms).

Definitely, this prototype is a demonstration of the power and the versatility that the models of image generation have and how they can be applied to create virtual environments. Thanks to modularity, there are infinite future lines of research. In the future, maybe a complete environment could be created, helping the work of design.

# Capítulo 7 Presupuesto

Para la elaboración de este proyecto se ha utilizado únicamente software de acceso gratuito o de código abierto, por lo que no es necesario incluir el coste de ninguna tecnología, solamente el coste por el trabajo realizado según las tareas, así como el coste eléctrico de mantener un ordenador encendido por días.

Asumiendo que el sueldo promedio de un ingeniero en España es de 30 €/h en el mercado laboral y teniendo en cuenta el consumo energético utilizado durante las más de 100 horas totales de entrenamiento de los modelos, se ha desarrollado la siguiente tabla.

## 7.1 Tabla de Presupuesto

| Tipos | Descripción | Horas | Precio Total (€) |
|---|---|---|---|
| **Desarrollo del conjunto de datos** | Desarrollo de funcionalidades, ya sea la creación del conjunto de datos como modificar los datos de entrenamiento. | 60 | 1800 |
| **Entrenamiento del modelo** | Entrenamiento del modelo, así como la investigación pertinente y todo el tiempo de entrenamiento. | 100 | 3000 |
| **Desarrollo de la herramienta que emplea el modelo** | Creación de la aplicación Flask, así como de su interfaz web, el proyecto en Unity y la configuración del mismo. | 100 | 3000 |
| **Pruebas y validaciones** | Diversas pruebas y comprobaciones del correcto resultado del conjunto de entrenamiento, de la capacidad y precisión del modelo entrenado, de la aplicación Flask y del proyecto en Unity. | 40 | 1200 |
| **Total** | Suma total de horas y costos. | **300** | **9000** |

**Tabla 7.1:** Resumen de tipos.

# Anexo - Repositorios de GitHub relevantes

## 1. Repositorio de este proyecto

Todo el código escrito, así como otros detalles de este proyecto se encuentran en https://github.com/pablo-pg/TFG/.

## 2. ComfyUI

La herramienta está completamente en el repositorio oficial. En este, se puede encontrar información acerca de la instalación y su uso. https://github.com/comfyanonymous/ComfyUI.

## 3. OneTrainer

Gracias a esta herramienta, la creación de LoRAs fue muy sencillo. El repositorio oficial contiene guía de instalación y uso. https://github.com/Nerogar/OneTrainer.

## 4. Replicación de DALL-E

Modelo y algoritmos que se iban a emplear originalmente en el proyecto. Pese a estar obsoleto y sin mantenimiento aparente, es una alternativa a tener en cuenta. https://github.com/lucidrains/DALLE-pytorch.

# Bibliografía


Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., … (2020). *Language Models are Few-Shot Learners*. https://arxiv.org/abs/2005.14165.

Comfyanonimus (2024). *ComfyUI*. En GitHub. https://github.com/comfyanonymous/ComfyUI.

Crowson, K. (s. f.). *VQGAN+CLIP (z+quantize method con augmentations, interfaz amigable)*. En Google Colab. https://colab.research.google.com/drive/1go6YwMFe5MX6XM9tv-cnQiSTU50N9EeT.

Esser, P., Rombach, R., Ommer, B. (2021). *Taming Transformers for High-Resolution Image Synthesis*. https://arxiv.org/abs/2012.09841.

Hernández A., J. (2023). *¿Qué son los modelos de difusión en Inteligencia Artificial?* https://empresas.blogthinkbig.com/que-son-los-modelos-de-difusion-en-inteligencia-artificial/.

Hu, E., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., … (2021). *LoRA: Low-Rank Adaptation of Large Language Models*. https://arxiv.org/abs/2106.09685.

Inteligencia artificial. (2024). En *Wikipedia.* https://es.wikipedia.org/w/index.php?title=Inteligencia_artificial&oldid=160718139.

Islam, A. (2022). *How Do DALL·E 2, Stable Diffusion, and Midjourney Work?* https://www.marktechpost.com/2022/11/14/how-do-dall%C2%B7e-2-stable-diffusion-and-midjourney-work/.

Kingma, P., Welling, M. (2013). *Auto-Encoding Variational Bayes*. https://arxiv.org/abs/1312.6114.

Liani, M. (s.f.). *DNND 3: the U-Net Architecture*. https://maxliani.wordpress.com/2023/04/07/dnnd-3-the-u-net-architecture/.

MidJourney (2022). *MidJourney*. https://www.midjourney.com/home.

Mishra, O. (2023). *Stable Diffusion Explained*. En Medium. https://medium.com/@onkarmishra/stable-diffusion-explained-1f101284484d.

Modelo de lenguaje grande. (2024). En *Wikipedia*. https://es.wikipedia.org/w/index.php?title=Modelo_de_lenguaje_grande&oldid=160580777.

Momentos de imagen. (2023). En *Wikipedia*. https://es.wikipedia.org/w/index.php?title=Momentos_de_imagen&oldid=152872788.

Morfología matemática. (2021). En *Wikipedia*. https://es.wikipedia.org/w/index.php?title=Morfolog%C3%ADa_matem%C3%A1tica&oldid=139256085.

Mozilla. (2024). *Llamafile*. En GitHub. https://github.com/Mozilla-Ocho/llamafile.

Nerogar (2024). *OneTrainer*. En GitHub. https://github.com/Nerogar/OneTrainer.

OpenAI (2021). *DALL-E*. En GitHub. https://github.com/openai/dall-e.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J.,Chanan, G., … (2019). *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. https://arxiv.org/abs/1912.01703.

Perceptrón multicapa. (2024). En *Wikipedia*. https://es.wikipedia.org/w/index.php?title=Perceptr%C3%B3n_multicapa&oldid=159957275.

Pérez, P. (2024). *TFG*. GitHub. https://github.com/pablo-pg/TFG/.

Radford, A., Wook Kim, J., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., … (2021). *Learning Transferable Visual Models From Natural Language Supervision.* https://arxiv.org/abs/2103.00020.

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., ... (2021). *Zero-Shot Text-to-Image Generation*. https://arxiv.org/abs/2102.12092.

Red neuronal artificial. (2024). En *Wikipedia*. https://es.wikipedia.org/w/index.php?title=Red_neuronal_artificial&oldid=160745697.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B. (2022). *High-Resolution Image Synthesis with Latent Diffusion Models*. https://arxiv.org/abs/2112.10752.

Rombach, R., Esser, P. (2022). *stable-diffusion-v1-5*. Hugging Face. https://huggingface.co/runwayml/stable-diffusion-v1-5.

Ronneberger, O., Fischer, P., Brox, T. (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. https://arxiv.org/abs/1505.04597.

Santana, C. (2022). *TUTORIAL 👉 ¡Entrena a la IA con tu CARA! - 100% GRATIS Y SIN GPUs (Stable Diffusion y Dreambooth)*. YouTube. https://youtu.be/rgKBjRLvjLs?si=i8OSchmKE8pR82EI.

Schüller Perdigón, M. (2022). *Generación automática de texturas para videojuegos a partir de una descripción textual*. [Trabajo Final de Grado]. Universidad de La Laguna.

Turing, A. (1950). *Computing Machinery and Intelligence.* Mind. https://doi.org/10.1093/mind/LIX.236.433.

U-Net. (2024). En *Wikipedia*. https://es.wikipedia.org/w/index.php?title=U-Net&oldid=158704581.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., ... (2017). *Attention Is All You Need*. https://arxiv.org/abs/1706.03762.

Wang, P. (2023). *DALL-E - colab*. En Google Colab. https://colab.research.google.com/drive/1dWvA54k4fH8zAmiix3VXbg95uEIMfqQM?usp=sharing.

Wang, P. (2023). *DALLE-pytorch*. En GitHub. https://github.com/lucidrains/DALLE-pytorch.