# Shap-MeD

Nicolas Laverde *, Melissa Robles †, Johan Rodríguez ‡
Universidad de los Andes. Bogotá, Colombia
E-mail: *n.laverdem@uniandes.edu.co, †mv.robles@uniandes.edu.co y ‡jd.rodriguezp1234@uniandes.edu.co

***Resumen*—Presentamos Shap-MeD, un modelo generativo de texto a objeto 3D especializado en el dominio biomédico. El objetivo de este trabajo es desarrollar un asistente que facilite el modelado 3D de objetos médicos, reduciendo los tiempos de desarrollo. El modelado 3D en medicina tiene diversos usos, entre los cuales se encuentran: simulación o planificación de procedimientos quirúrgicos, diseño de implantes protésicos personalizados, educación médica, creación de modelos anatómicos y desarrollo de prototipos de investigación. Para lograr esto, utilizamos Shap-e, un modelo generativo de texto a objeto 3D de código abierto de OpenAI, al que le aplicamos un fine-tuning con un dataset de objetos biomédicos. Nuestro modelo alcanzó un MSE de 0.089 en la generación de latentes en el conjunto de evaluación, mientras que Shap-e obtuvo un MSE de 0.147. Además, realizamos una evaluación cualitativa en la que comparamos nuestro modelo con otros en la generación de objetos biomédicos, y encontramos que nuestro modelo presenta una mayor precisión en las estructuras.**

## I. Introducción

La nueva tendencia en salud son los tratamientos personalizados, dejando atrás los enfoques universales y los tratamientos de *un solo diseño para todos*. Según el Servicio de Salud Nacional de Inglaterra, los tratamientos universales presentan una ineficiencia del 70 % en los pacientes, lo que marca la necesidad de tratamientos personalizados [1]. Sin embargo, aplicar un enfoque de tratamientos personalizados resulta inviable con los procesos de fabricación tradicionales, ya que conlleva tareas intensivas en tiempo y esfuerzo, mientras que los productos en 3D han sido bien recibidos por los pacientes [2]. Esta tendencia genera la necesidad de que la industria transite hacia nuevas tecnologías que permitan la personalización.

La impresión 3D fue introducida hace más de tres décadas. Desde entonces, esta tecnología ha transformado radicalmente la industria de la manufactura. El propósito original de esta tecnología era la creación de prototipos de ingeniería, como partes de autos, accesorios de moda e incluso modelos de casas, que eran producidos mediante esta tecnología. De hecho, hoy en día, aún se utiliza para estos fines [3, 4, 5]. Sin embargo, las aplicaciones no se limitan a eso. La impresión 3D ha revolucionado sin duda el sistema de salud, ya que puede producir objetos a medida, lo que la hace perfecta para la personalización de prótesis, implantes, sistemas de entrega de medicamentos, tejidos y dispositivos médicos [6]. En la Figura 1 se pueden visualizar las cinco grandes categorías en las que se emplea la impresión en 3D.

La impresión 3D aborda una de las principales problemáticas en el ámbito de la salud: la escasez de órganos para trasplantes. Según las estadísticas sobre donación de órganos, aproximadamente 20 pacientes fallecen diariamente mientras esperan la oportunidad de recibir un trasplante [7]. Además de la limitada disponibilidad de donantes de órganos, otro de los desafíos más significativos es garantizar la biocompatibilidad del órgano donado [8]. La impresión en 3D tiene la capacidad de generar tejidos con dimensiones y materiales específicos para cada paciente, lo que puede contribuir significativamente a los trasplantes de órganos [9].

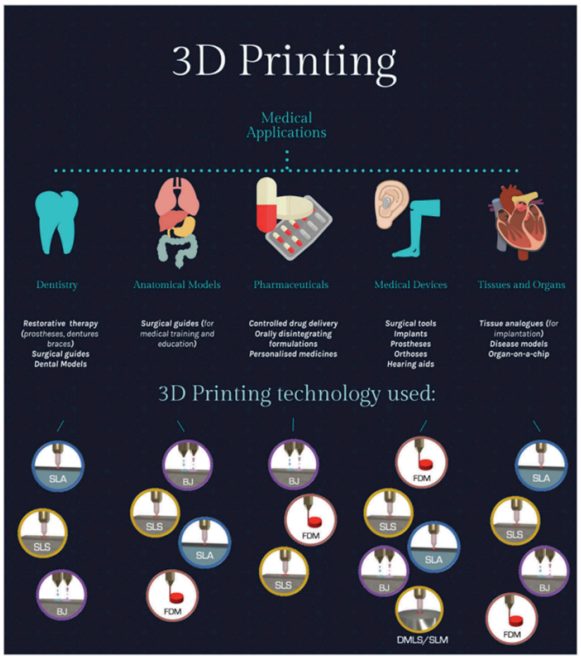


Figura 1: Aplicaciones actuales de la impresión 3D en medicina y atención médica.

Los avances en la impresión en 3D para tratamientos personalizados han sido significativos. Las estructuras pueden generarse a partir de un archivo digital en 3D utilizando software de *diseño asistido por computadora* (CAD) y técnicas de visión por computadora, que incluyen resonancias magnéticas o tomografías en 3D [10]. Aunque los resultados son muy positivos, es importante destacar que la técnica de visión por computadora tiene un límite de personalización determinado por las imágenes obtenidas.

El objetivo de este proyecto es desarrollar un asistente dedicado a la creación de modelos anatómicos y órganos, condicionado por texto o imagen. Esto se hace con el fin de facilitar la creación de prototipos de investigación de tejidos y explorar la viabilidad de su utilización en trasplantes prostéticos. Adicionalmente, se contempla su aplicación como una herramienta educativa, beneficiando a estudiantes en el aprendizaje y comprensión de la anatomía y la medicina. Siguiendo con la idea de personalización en los tratamientos, buscamos incorporar información específica del paciente para el diseño de estas representaciones en 3D. Lograremos esto mediante un *fine-tuning* de modelos generativos basados en modelos de difusión con objetos biomédicas. Aunque se presentarán estrategias que generan mallas en 3D condicionadas más adelante, es relevante destacar que ninguna de ellas se enfoca específicamente en el dominio biomédico.

## II. Antecedentes

Determinar la metodología de representación de objetos en tres dimensiones constituye un desafío por la complejidad inherente a las formas, las variaciones en la iluminación y las dimensiones espaciales. Diversas estrategias son empleadas para abordar distintos aspectos de la representación tridimensional, tales como el uso de mallas poligonales, nubes de puntos y aproximaciones basadas en funciones implícitas. A modo de ejemplos, se encuentran los formatos STL, XYZ y NeRF, los cuales se corresponden, respectivamente, con cada una de las técnicas anteriormente expuestas. En esta sección se describen las distintas representaciones necesarias para entender los modelos utilizados durante el proyecto.

### *II-A. STL*

La representación tridimensional de un objeto mediante un archivo STL implica una descripción geométrica mediante mallas poligonales. Este formato de archivo almacena la superficie del objeto mediante una red de triángulos, donde cada vértice define un punto en el espacio tridimensional. Una representación STL, por lo tanto, es una matriz de dimensiones $(T, 3, 3)$, en donde $T$ representa el número de triángulos representados y cada triángulo se representa por una matriz de dimensiones $3 \times 3$ de la forma

$$\begin{bmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix}.$$

También se encuentran los componentes del vector normal unitario al triángulo, asegurándose que apunte hacia afuera de acuerdo con el modelo. Esta representación permite la transferencia de modelos 3D entre diferentes programas y sistemas, facilitando su utilización en aplicaciones de diseño, ingeniería y fabricación.

### *II-B. OBJ*

El formato OBJ, conocido también como archivo Wavefront OBJ, es una representación sencilla para objetos 3D, desarrollada para el visualizador *The Advanced Visualizer* por la compañía Wavefront Technologies [11]. Este formato consiste en un archivo de texto plano que resume la información del objeto, incluyendo detalles de textura y color. Debido a estas características, el formato OBJ ha ganado mayor popularidad en la impresión 3D en comparación con el formato STL.

Este formato posibilita la representación de planos más complejos que la simple disposición de triángulos. Se establece un conjunto $V$ de vértices $v_1, v_2, \ldots, v_n \in \mathbb{R}^3$ y se definen subconjuntos de estos vértices que conforman las superficies o hiperplanos correspondientes. Además, el archivo de texto incluye información relativa a las texturas y colores de las superficies definidas.

A diferencia del formato STL, esta modalidad de archivo no solo mejora significativamente la representación tridimensional al presentar más flexibilidad sobre los hiperplanos, sino que también permite la inclusión de curvas y superficies *Freeform*. Estas curvas se construyen mediante la interpolación polinomial entre un conjunto finito de vértices, lo que resulta en la aproximación de superficies suaves y tridimensionales que se ajustan adecuadamente a los vértices seleccionados, generando así una estructura más continua.

### *II-C. NeRF*

La representación 3D de NeRF se propuso en 2022 [12] como una alternativa a las representaciones clásicas utilizadas hasta el momento. Esta se basa en la construcción de una función implícita que mapea tuplas $(\mathbf{x}, \mathbf{d})$ de coordenadas y direcciones a tuplas $(\mathbf{c}, \sigma)$ de colores y densidades. Las coordenadas de entrada, denotadas por $\mathbf{x} = (x, y, z) \in \mathbb{R}^3$, se representan como puntos, mientras que las direcciones se expresan mediante dos ángulos, $(\theta, \phi) \in \mathbb{R}^2$, que representan el punto de vista desde el cual se observa el objeto. Por otro lado, los colores siguen la representación RGB $(c_R, c_G, c_B)$, y las densidades son números reales que simbolizan la opacidad o la *cantidad de materia* en una determinada ubicación dentro del volumen tridimensional de la escena. En resumen, la función se puede caracterizar como

$$\begin{aligned} F_\Theta : \mathbb{R}^5 &\to \mathbb{R}^4 \\ (\mathbf{x}, \mathbf{d}) &\mapsto (\mathbf{c}, \sigma). \end{aligned}$$

La forma en la que se parametriza la función es por medio de una red neuronal clásica Multi Layer Perceptron (MLP) con pesos $\Theta$, lo que le da el nombre de *función implícita* a $F_\Theta$. A partir de esta, dada una dirección, es posible construir una imagen asignando a cada uno de los píxeles el color obtenido por la red neuronal.

### *II-D. STF*

La representación STF (Signed Distance Functions and Texture Fields) se refiere a la formulación de una función implícita como método de representación tridimensional, caracterizada por generar como salidas distancias con signo o colores con textura. Las representaciones SDF (Signed Distance Functions) se definen a partir de una función

$$\begin{aligned} G_\Theta : \mathbb{R}^3 &\to \mathbb{R} \\ (x, y, z) &\mapsto d. \end{aligned}$$

donde $|d|$ representa la distancia del punto $(x, y, z)$ al punto más cercano de la superficie del objeto, y el signo $\operatorname{sign}(d)$ es negativo si el punto está dentro del objeto y positivo si se encuentra fuera del mismo. Esta función es considerada *implícita*, dado que la superficie del objeto está definido como las coordenadas $(x, y, z)$ que satisfacen $G_\Theta(x, y, z) = 0$. Esta representación SDF ha sido utilizada en modelos generativos como en DMTet [13]. Otras aproximaciones sugieren el uso de información adicional a la distancia con signo, agregando información de textura (Texture Fields) y color en formato RGB.

## III. Trabajo Relacionado

### *III-A. Point-E*

Al momento de publicar Point-e [14] en el año 2022, todos los modelos del estado del arte requerían un alto tiempo de inferencia en múltiples GPU para producir solo un ejemplo, lo cual hacía imposible democratizar la generación de contenido 3D a partir de texto al público. Por ello, en este artículo se plantea una alternativa que genera una nube de puntos 3D a partir de texto. Para ello, se define una nube de puntos como un arreglo de tuplas, donde cada tupla representa un punto que tiene 3 entradas para las coordenadas tridimensionales y otras tres para los canales de color RGB.

Para lograr generar nubes de puntos, como la presentada anteriormente, se tiene una arquitectura basada en transformers [15] y

modelos de difusión Gaussiana [16], los cuales parten de un proceso de adición de ruido por pasos definido como

$$q(x_t|x_{t-1}) : \mathcal{N}(x_t; \sqrt{1-\beta}x_{t-1}, \beta_t\mathbf{I})$$

Donde $x_{t-1}$ y $x_t$ son dos pasos consecutivos de adición de ruido a un ejemplo $x$, que en este caso es una nube de puntos, y el proceso generativo está modelado con una distribución normal condicionada al ejemplo con adición de ruido en una etapa anterior $x_{t-1}$, y una variable de programación de ruido $\beta$ que se encarga de ajustar el ruido gradualmente en cada paso, limitando la varianza de este para evitar que haya saltos muy grandes de ruido entre pasos. Este proceso de adición de ruido se aproxima con una arquitectura transformer [15], que se encarga de modelar la probabilidad condicional inversa $p_\theta(x_{t-1}|x_t)$, por medio de la predicción del ruido $\epsilon$.

Si bien, este proceso es ideal para generar nubes de puntos con significado a partir de nubes de puntos creadas por ruido distribuido normalmente, es necesario hacer modificaciones para poder generarlas con base a un texto dado. Por lo tanto, generalmente se tiene un proceso de condicionamiento, en que se concatena la nube de puntos creada por ruido y una representación textual como entrada de la arquitectura transformer para poder utilizar texto como entrada en tiempo de inferencia [17]. No obstante, en la experimentación para crear Point-e [14] se encontró de forma empírica que se obtenía un mejor resultado condicionando sobre imágenes en vez de texto, por lo que se tiene un arquitectura de difusión basada en transformers que parte de una representación de imagen dada por un modelo ViT-L/14 CLIP [18] y una nube de puntos ruidosa para predecir una nube de puntos con sentido que represente lo que está en la imagen, como se puede ver en la Figura 2.

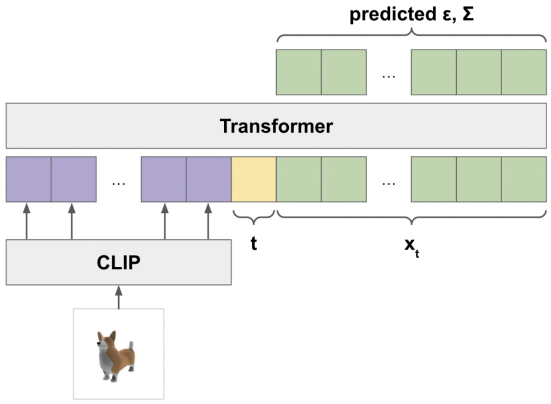


Figura 2: Arquitectura de difusión basada en transformers para generar una nube de puntos con sentido a partir de una imagen y una nube de puntos creada con ruido distribuido normalmente [14]

Como se observa, esta arquitectura solamente permite generar nubes de puntos a partir de imágenes, por lo que es necesario introducir un componente que genere imágenes a partir de texto. Este componente es un modelo GLIDE [19], que consiste en un modelo de difusión condicionado a texto, que a diferencia de CLIP [18], solo necesita texto sin etiqueta de clasificación para generar imágenes. Para poder usar GLIDE de manera óptima se hizo fine-tuning con un dataset propio compuesto de millones de imágenes que se crearon a partir de renders 3D con Blender [20], creando 20 imágenes por modelo, cada una dada por un ángulo de cámara aleatorio.

Esta arquitectura se divide en tres componentes generativos consecutivos:

1. **Generador de texto a imagen**: Se usa una arquitectura GLIDE para convertir de texto a imagen.
2. **Generador de nube de puntos de baja resolución**: Se usa una arquitectura transformer como la presente en la figura 2 para crear una nube de puntos de baja resolución de 1.000 puntos a partir de la imagen generada en el paso anterior.
3. **Generador de nube de puntos de alta resolución**: Al igual que en el paso anterior, se usa una arquitectura transformer como la de la Figura 2, que está condicionada a la imagen generada en el paso 1. No obstante, esta arquitectura también se condiciona a la nube de puntos generada en el paso 2, por lo que parte de los puntos de la nube de puntos usados en la entrada se reemplazan por la nube de puntos creada en el paso 2 para generar una nube de puntos de alta resolución de 4.000 puntos.

En la arquitectura descrita anteriormente, se tiene el detalle de entrenamiento del primer paso por medio de tuplas de texto y vistas sintéticas creadas a partir de un dataset de objetos 3D. Para completar el entrenamiento del flujo completo, se crean nubes de puntos de muy alta resolución a partir de las 20 vistas que se tienen para cada objeto y se hace un muestreo para obtener nubes de 4.000 puntos y 1.000 puntos respectivamente. Las primeras nubes se usan para entrenar el componente del paso 3 y las segundas para entrenar el componente del paso 2, luego de aplicar el proceso de adición de ruido descrito en [16].

### *III-B. Shap-e*

Shap-e [21] se configura como un modelo generativo condicional que, a partir de textos o imágenes, genera una representación tridimensional. Este desarrollo, concebido por OpenAI, se distingue de Point-e al generar directamente los parámetros de una función implícita, que puede emplearse como representación de un objeto tridimensional, como se requiere en las representaciones NeRF (II-C) y STF (II-D). La estructura del modelo consta de dos componentes principales: un *encoder* que produce los parámetros de una función implícita a partir de una representación de un objeto tridimensional y un modelo de difusión condicional ya sea a imágenes o descripciones de texto.

La arquitectura del encoder se refleja en la Figura 3. Esta parte del modelo toma como entrada dos representaciones distintas de un objeto tridimensional: una representación en forma de nube compuesta por 16,000 puntos y una serie de vistas en formato RGBA del objeto desde diversos ángulos. Estas entradas son sometidas a capas de atención cruzada, seguidas por una arquitectura de transformer que genera una secuencia de representaciones de los inputs. Estas representaciones son utilizadas como parámetros en las funciones implícitas que finalmente representan el objeto tridimensional. Específicamente, la salida del encoder puede concebirse como los pesos de una red neuronal MLP como la utilizada en NeRF, en la cual, a partir de una entrada $(x, y, z)$, se generan tres salidas coherentes con las representaciones de funciones implícitas:

- $\sigma$: Representa la densidad en la representación NeRF.
- $RGB$: Representa el color.

- $SDF$: Representa la distancia con signo, característica de la representación STF.

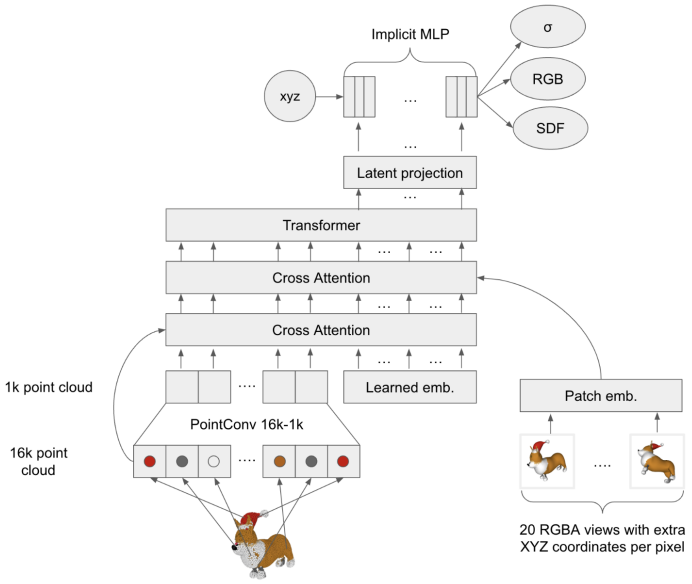


Figura 3: Representación del encoder de Shap-e. Obtenido en [21].

Inicialmente, el encoder se entrena únicamente para obtener representaciones NeRF. Para esto, se utiliza una mezcla de dos funciones de pérdida, utilizando una función conjunta

$$\mathcal{L}_{NeRF} = \mathcal{L}_{RGB} + L_T.$$

La función de pérdida $\mathcal{L}_{RGB}$ compara los colores de los objetos 3D reales y los colores obtenidos a partir de la función implícita generada por el encoder. Más formalmente, se obtienen aleatoriamente un conjunto $R$ de 4.096 rayos (direcciones) del objeto 3D y se comparan los colores estimados de cada rayo con la función implícita y el color real utilizando la norma $L_1$ de la diferencia. La comparación de la función implícita obtenida se realiza por medio de dos renderizados: uno grueso ($c$) que proporciona una aproximación inicial menos detallada de la escena, y uno fino ($f$) que refina la esta primera estimación, proporcionando una representación más detallada y precisa de la escena. Con estos renderizados se calcula la función de pérdida siguiendo la fórmula

$$\mathcal{L}_{RGB} = \mathbb{E}_{r\in R}\left[||\hat{C}_c(\mathbf{r}) - C(\mathbf{r})||_1 + ||\hat{C}_f(\mathbf{r}) - C(\mathbf{r})||_1\right],$$

en donde $C(\mathbf{r})$ representa el color real y $\hat{C}_c(\mathbf{r})$ y $\hat{C}_f(\mathbf{r})$ los colores predichos del renderizado grueso y fino respectivamente.

Por otro lado, la función de pérdida $L_T$ compara la *transmitancia* [1] real y la compara con la generada a partir de la función implícita. Al igual que para $\mathcal{L}_{RGB}$, se compara a partir de la norma $L_1$ de la diferencia entre los valores reales y los estimados en el mismo conjunto de rayos para ambos renderizados:

$$\mathcal{L}_T = \mathbb{E}_{r\in R}\left[||\hat{T}_c(\mathbf{r}) - T(\mathbf{r})||_1 + ||\hat{T}_f(\mathbf{r}) - T(\mathbf{r})||_1\right].$$

Posteriormente, se emplea la red preentrenada para llevar a cabo el Fine-Tuning considerando la representación STF. Con este propósito, se define una función de pérdida conjunta dada por

$$\mathcal{L}_{FT} = \mathcal{L}_{NeRF} + \mathcal{L}_{STF},$$

[1]Transmitancia: propiedad que describe cómo la luz viaja a través de un medio u objeto. Representa la fracción de luz que atraviesa un punto a lo largo de un rayo sin ser absorbida ni dispersada. Se calcula a partir de las densidades $\sigma$.

donde $\mathcal{L}_{STF}$ compara la malla real con la malla reconstruida. Para la construcción de la malla a partir de las salidas del encoder, se define una grilla de dimensiones $128 \times 128 \times 128$, y se calculan los valores asociados a la función en dicha grilla. Con esta información, se generan $N$ imágenes mediante el proceso de renderizado, las cuales se comparan utilizando la norma $L_2$ con las reales para obtener el valor final de la función de pérdida $\mathcal{L}_{STF}$ siguiendo la fórmula

$$\mathcal{L}_{SF} = \frac{1}{N \cdot s^2}\sum_{i=1}^{N} ||\text{Render}(Mesh_i) - \text{Image}_i||_2^2,$$

en donde $s$ es la resolución de las imagenes generadas, $Mesh_i$ es la malla construida para la muestra $i$ y $\text{Image}_i$ es un renderizado objetivo RGBA para la imagen $i$.

La segunda componente del modelo es un modelo de difusión condicional basado en transformers, al igual que en Point-e (III-A). A diferencia de este último, la entrada no es una nube de puntos, sino una secuencia de vectores latentes que representan los pesos de una función implícita. Esta secuencia tiene dimensiones $1024 \times 1024$, lo que la integra en la arquitectura como una secuencia de 1024 tokens. Para la parte condicional del modelo de difusión, se adopta la misma aproximación que en Point-e (III-A), utilizando embeddings de CLIP tanto para textos como para imágenes. Finalmente, a diferencia del modelo de difusión original, se busca minimizar el error de $x_0$ y $x_t$ y no el error del ruido, como se muestra a continuación:

$$L_{x_0} = \mathbb{E}_{x_0\sim q(x_0),\epsilon\sim\mathcal{N}(0,I),t\sim U[1,T]}\|x_\theta(x_t,t) - x_0\|^2$$

Los resultados del artículo se presentan mediante métricas que evalúan la eficiencia del encoder, tales como las métricas PSNR y la R-Precision de CLIP, acompañadas de un análisis cualitativo de las muestras generadas. El modelo exhibe mejoras significativas en comparación con Point-e en la tarea condicional basada en textos, mientras que en la tarea condicional basada en imágenes muestra resultados comparables. Esto se puede evidenciar en los ejemplos proporcionados en el artículo, comparando las salidas de ambos modelos (Figura 4). Por último, Shap-e demuestra una inferencia más rápida que Point-e al no requerir un modelo adicional de difusión para el upsampling, lo que representa una ventaja en comparación con el modelo previamente analizado.

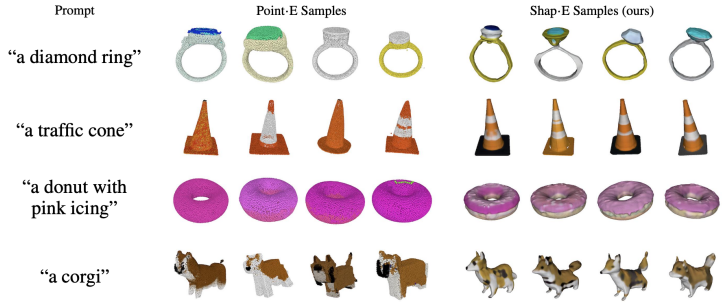


Figura 4: Comparación de resultados Shap-e vs Point-e. Imagen obtenida de [21].

### *III-C. Otros modelos*

*III-C1. Modelo de Reconstrucción de Gran Escala:* El Modelo de Reconstrucción de Gran Escala [22] (LRM, por sus siglas en inglés) introduce un enfoque basado en datos para la reconstrucción

3D a partir de una única imagen, utilizando una arquitectura de codificador-decodificador basada en Transformers [15] de gran envergadura. Este modelo se compone de tres componentes principales:

- **Codificador de Imagen:** LRM utiliza inicialmente un Vision Transformer (ViT) [23], específicamente DINO [24], para codificar la imagen en tokens de características por parche. DINO es escogido por su capacidad para aprender de manera autodidacta sobre la estructura y textura del contenido relevante en las imágenes, lo cual es crucial para que LRM reconstruya la geometría y el color en el espacio 3D.
- **Decodificador de Imagen a Triplano:** Se implementa un decodificador para proyectar características de la imagen y de la cámara en embeddings espaciales-posicionales aprendibles, traduciéndolos a representaciones triplanas. Este decodificador actúa como una red previa que proporciona información geométrica y de apariencia necesaria, compensando las ambigüedades inherentes a la reconstrucción a partir de una única imagen.
- **Representación 3D:** Se adopta la representación triplana como una característica compacta y expresiva. Cada uno de los planos, $(64 \times 64) \times d_T$, se usa para proyectar cualquier punto 3D dentro del cuadro delimitador del objeto NeRF y consultar características de puntos mediante interpolación bilineal, que luego se decodifican en color y densidad usando una MLP.

Finalmente, para abordar el problema Text-to-3D se tendría que utilizar un modelo previo al LRM para generar la imagen de entrada, como por ejemplo Stable Diffusion [25].

*III-C2. Modelo Gaussiano Multivista Grande:* El Modelo Gaussiano Multivista Grande (LGM, por sus siglas en inglés) es un marco diseñado para generar objetos en 3D de alta resolución a partir de un prompt o una imagen [26]. De este trabajo, se identificaron dos ideas principales:

- **Representación del objeto 3D:** Se propone utilizar características de modelos gaussianos multivistas como representación de la malla. Posteriormente, esta se fusiona para una renderización diferenciable.
- **Estructura 3D:** Se utiliza una U-net asimétrica como estructura principal para la generación de la malla. Esta opera en las imágenes multivistas, las cuales pueden ser producidas a partir de texto o de una imagen individual, haciendo uso de modelos de difusión multivistas.

Los modelos gaussianos multivistas utilizan una colección de gaussianas en 3D para representar un objeto. Cada gaussiana está definida por un centro $x \in \mathbb{R}^3$, un factor de escalamiento $s \in \mathbb{R}^3$, y un cuaternión de rotación $q \in \mathbb{R}^4$. Además, se incluyen un valor de opacidad $\alpha \in \mathbb{R}$ y un valor para el color $c \in \mathbb{R}^C$. Estos parámetros generan la gaussiana. Para renderizar estas gaussianas se utiliza la composición alfa para cada píxel.

En la U-net asimétrica, cada píxel extraído del mapa de características se trata como una gaussiana en 3D. Se emplea una U-net asimétrica para utilizar imágenes de mayor resolución al tiempo que se controla el número de gaussianas. En cuanto a rendimiento, este modelo supera a Shap-e y Point-e (ver Figura 5).

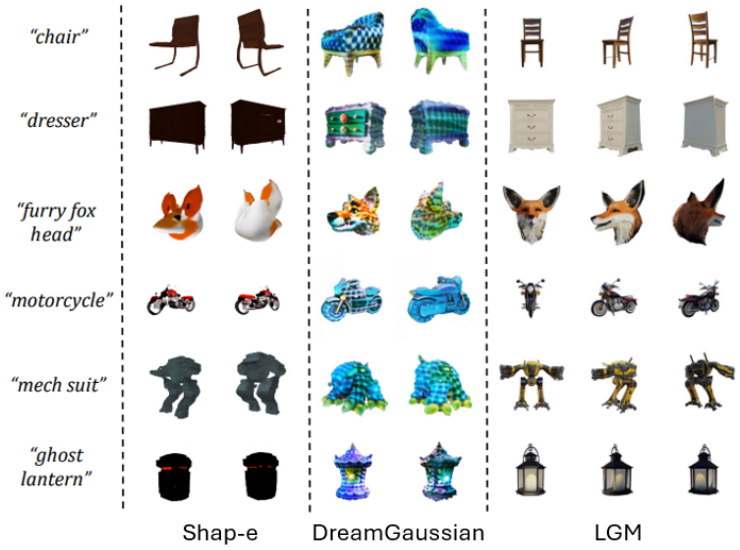


Figura 5: Comparación de resultados LGM vs Shap-e

## IV. Dataset

Para llevar a cabo la fase de fine-tuning del modelo, se utilizó el dataset *MedShapeNet* como fuente de mallas biomédicas [27]. Este dataset agrupa 23 fuentes distintas, que suman más de 100,000 mallas con sus respectivas anotaciones (ver Figura 6). A diferencia de datasets existentes, como ShapeNet, que se componen de modelos tridimensionales de diseño asistido por computadora (CAD) de objetos del mundo real (como aviones, automóviles, sillas y escritorios), MedShapeNet proporciona formas tridimensionales extraídas de datos de imágenes de pacientes reales, abarcando tanto a sujetos saludables como patológicos.

Los autores de MedShapeNet crearon dos modos de interacción para utilizar este dataset: una interfaz basada en internet que brinda acceso a los datos originales de formas en alta resolución, y una API que permite a los usuarios interactuar con los datos de mallas a través de Python.

Las figuras en 3D se almacenan en los formatos estándar para estructuras de datos geométricas, es decir, NIfTI (.nii) para cuadrículas de vóxeles, estereolitografía (.stl) para mallas y formato de archivo poligonal (.ply) para nubes de puntos, lo que facilita una vista previa rápida de las formas a través de softwares existentes. El conjunto de datos contiene información sobre 50 categorías diferentes, de las cuales el 30 % corresponde a órganos y el 70 % restante a huesos, músculos, arterias, venas y diversos instrumentos quirúrgicos. Entre los órganos se identificaron algunas categorías con problemas en las mallas presentadas. En particular, estos problemas se encontraron en la categoría de *cerebro*, como se muestra en la Figura 7, donde varios ejemplares presentaban mallas que, a simple vista, no correspondían con la categoría asignada. Otra categorías que presentaba problemas era *heart*, en donde si bien se evidencia la forma de un corazón, la mayoría de ejemplares presentan agujeros, posiblemente correspondientes a otras arterias o venas cercanas.

Debido a los problemas detectados en algunas categorías y a la limitación de recursos computacionales, se seleccionaron un total de 3,589 mallas para llevar a cabo el fine-tuning. Las mallas seleccionadas representan el 10 % del total de datos de órganos en MedShapeNet y pertenecen a las categorías de aorta, hígado, riñón y corazón (Figura 7). Entre estas, la categoría con mayor

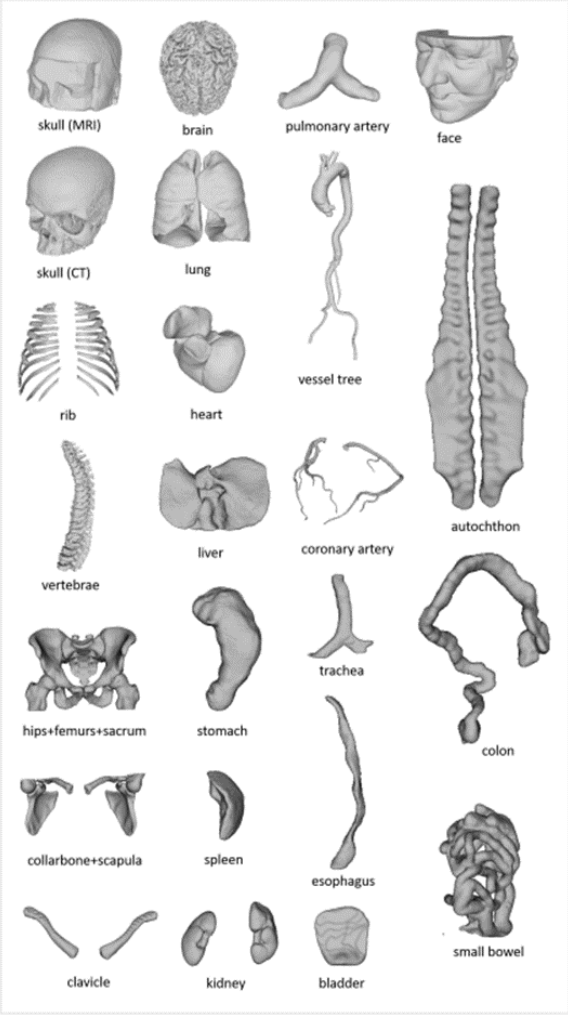


Figura 6: Mallas de ejemplo en MedShapeNet, incluidos varios huesos (cráneos, costillas y vértebras), órganos (cerebro, pulmón, corazón, hígado), vasos (árbol de vasos aórticos y arteria pulmonar) y músculos.

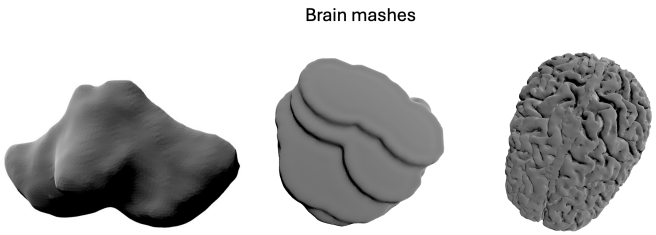


Figura 7: Ejemplos de mallas de la categoría de cerebro. A la izquierda y el centro, dos ejemplo de objetos mal clasificadas. A la derecha un ejemplo correcto de cerebro.

representación es *Aorta*, la arteria más grande del cuerpo humano. A su vez, la categoría menor representada entre los datos elegidos para el entrenamiento es *Heart*, con un total de 277 registros. Como se comentó previamente, esta última clase presenta ejemplares con agujeros. Al incluirla en el dataset para el fine-tuning se prueba la importancia de la calidad de los datos y la complejidad de las mallas seleccionadas en el modelo. Varios ejemplos de las categorías seleccionadas se muestran en la Figura 9.

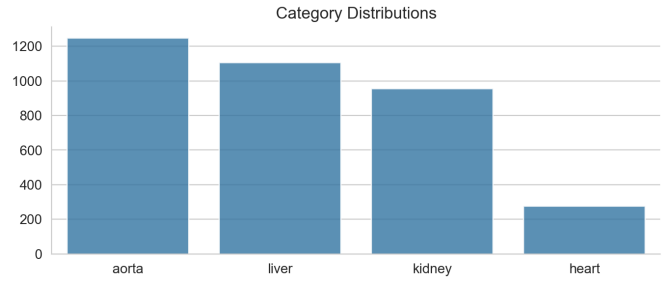


Figura 8: Distribución categorías seleccionadas

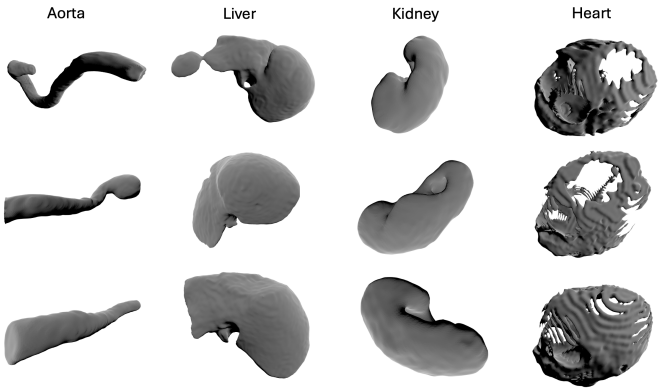


Figura 9: Ejemplos categorías seleccionadas

## V. Metodología

En la sección III se mencionaron cuatro arquitecturas candidatas a ser usadas en el presente problema biomédico. Estas cuatro arquitecturas se basan en codificación de texto/imagen a una representación 3D y poseen repositorios de código abierto, lo cual permite explorar su funcionamiento al detalle. Teniendo esto en cuenta, seguimos el siguiente proceso experimental:

1. Selección de modelo para fine-tuning
2. Preprocesamiento de datos
3. Parámetros del fine-tuning
4. Despliegue del modelo

### *V-A. Selección de modelo para fine-tuning*

Entendemos que hacer fine-tuning de modelos entrenados con grandes datasets genéricos siempre es una opción ideal, ya sea en el ámbito de modelos de texto o de imágenes. En este caso particular, buscamos aplicar esta técnica al contexto de objetos 3D con objetos biomédicos. De esta manera, aprovechamos todo el conocimiento previo del modelo pre-entrenado con una amplia cantidad de objetos. Como se mencionó anteriormente, se llevó a cabo una investigación sobre modelos existentes que convierten texto en objetos 3D. Sin embargo, encontramos inconvenientes en algunos de estos modelos, lo que nos obligó a seleccionar únicamente uno de ellos.

Veamos primero LRM y LGM. Estos dos modelos son los más recientes dentro de nuestro grupo de modelos de texto a objeto 3D. Ambos fueron entrenados con Objaverse [28], un conjunto de datos con más de 800,000 objetos 3D genéricos y sus respectivas anotaciones (prompts). Esto los convierte en excelentes candidatos para aprovechar su conocimiento, no solo por la cantidad de datos con los que fueron entrenados, sino también porque sabemos que obtienen mejores resultados (ver Figura 5). Sin embargo, son

modelos bastante grandes. LGM fue entrenado con 32 NVIDIA A100 (80G) durante 4 días, y LRM con 128 NVIDIA A100 (40G) durante 3 días. Esto los descarta de inmediato.

Nos quedarían Point-e y Shap-e. Ambos modelos son de OpenAI, pero Shap-e es más reciente. Sin embargo, Shap-e fue entrenado con aproximadamente un millón más de objetos. Adicionalmente, estos modelos fueron entrenados con 8 NVIDIA V100, muchas menos que LRM y LGM. Por estas razones, elegimos Shap-e como nuestro modelo para realizar el fine-tuning. Más adelante, analizaremos las implicaciones de haber seleccionado Shap-e como modelo en el dataset.

### *V-B. Preprocesamiento de datos y extracción de latentes*

El pipeline de preprocesamiento es realmente simple y se puede dividir en dos partes:

1. **Conversión de STL a OBJ:** Dado que los datos de MedShapeNet estaban en formato .stl y el modelo de Shap-e requiere datos en formato .obj, fue necesario realizar una conversión de formatos previa a la obtención de las representaciones latentes de las mallas. Inicialmente, se utilizó la API de Aspose 3D [29]; sin embargo, debido a una restricción que limitaba la conversión a un máximo de 50 mallas, se optó por emplear la librería de código abierto Open3D [30].
2. **Extracción de latentes:** Después de obtener los archivos .obj, utilizamos el *transmitter* (o encoder) de Shap-e para representarlos en el espacio latente. Este espacio latente corresponde a los pesos de la MLP, los cuales codifican el objeto utilizando NeRF, como se menciona en la Sección III-B. No realizamos fine-tuning al *transmitter*, es decir, congelamos sus pesos, pero lo utilizamos para convertir nuestro dataset en la representación latente. Con estas representaciones, realizamos el fine-tuning del modelo generativo de Shap-e.

### *V-C. Parámetros de fine-tuning*

El fine-tuning, en general, tiene algunas buenas prácticas. Con el fine-tuning, queremos aprovechar el aprendizaje de un modelo para adaptarlo a nuestra tarea, en este caso, la generación de mallas en el dominio biomédico. Para ello, actualizamos los pesos del modelo preentrenado con nuestro conjunto de datos. Sin embargo, es crucial tener cuidado de no dañar el conocimiento que ya posee el modelo, en este caso, el modelo de difusión. Por esta razón, utilizamos una tasa de aprendizaje baja de $1 \times 10^{-5}$. Adicionalmente, empleamos un batch size de 8 y entrenamos durante 25 épocas. Realizamos el fine-tuning en una GPU NVIDIA A40-24C (24G).

### *V-D. Evaluación del modelo*

Ahora bien, evaluamos nuestro modelo con fine-tuning de dos maneras. En la primera forma, utilizamos un enfoque cuantitativo, donde el *Mean Squared Error* (MSE) del modelo de difusión es nuestra métrica de evaluación. Comparamos esta métrica con el MSE de Shap-e sin fine-tuning. Para ello, dividimos el dataset en tres partes: un 80 % para entrenamiento, un 10 % para evaluación y el 10 % restante para validación. Cabe destacar que este modelo de difusión trabaja en la representación latente de los objetos y no en los objetos propiamente dichas. Esta es la gran diferencia entre Shap-e y Point-e.

Evaluar modelos generativos puede ser un desafío debido a su naturaleza de generar contenido nuevo. Esto también ocurre con los modelos de texto e imágenes. Por esta razón, muchos optan por evaluadores humanos para determinar la calidad del contenido generado. Es por esto que nuestro segundo método de evaluación es cualitativo. Comparamos nuestro modelo con fine-tuning frente a Shap-e, Point-e, LGM, y LRM, proporcionándoles el mismo prompt en el dominio biomédico. De forma visual, determinamos qué modelo obtiene los mejores resultados.

### *V-E. Despliegue del modelo*

El despliegue se hizo por medio de Streamlit, un framework de python para despliegue de aplicaciones de ciencia de datos y machine learning. Este framework funciona como una maquina de estados, donde se tienen almacenados una serie de elementos que son accedidos desde un front-end que se puede visualizar desde el navegador. Todo lo que se despliega con este framework funciona como un monolito, por lo que tanto backend como frontend se encuentran en la misma maquina, sin necesidad de utilizar elementos adicionales como docker o Triton.

Para ello, se tiene una arquitectura interna, en que se tienen dos subdivisiones. La primera subdivisión consiste en el backend, manejado directamente por la maquina de estados, donde se almacena el modelo fine-tuneado de Shap-e y sus pesos y el prompt con que el usuario quiere generar un objeto. Esta maquina de estados pasa el prompt al modelo preentrenado para generar una predicción, la cual se transforma a un objeto en un archivo con extensión .ply de manera local, gracias a funciones auxiliares de Shap-e. La segunda subdivisión es el frontend, que se encarga de actualizar la maquina de estados con el input de usuario, que son el prompt de generación y la versión de modelo a seleccionar. Una vez el usuario ingresa estos datos por medio de la interfaz gráfica en un navegador, la maquina de estados se actualiza con la versión del modelo y el prompt, acción que causa que el backend cargue los pesos de modelo correspondientes y los use para generar el objeto en formato .ply asociada al prompt de usuario.

## VI. Resultados y análisis

### *VI-A. Aprendizaje en el fine-tuning*

Durante las 25 épocas de entrenamiento del fine tuning, se observó una notable disminución en la función de pérdida MSE tanto en el conjunto de datos de entrenamiento como en el conjunto de datos de prueba (ver Figura 10). Inicialmente, la función de pérdida registró valores por encima de 0.14 para ambos conjuntos de datos, indicando una discrepancia entre las predicciones del modelo y los valores reales. Sin embargo, a medida que avanzaba el proceso de entrenamiento, se evidenció una mejora, alcanzando valores cercanos a 0.09 al finalizar el mismo. Este descenso en la función de pérdida refleja una mayor precisión y capacidad predictiva del modelo luego del fine-tuning sobre los datos pre-entrenados.

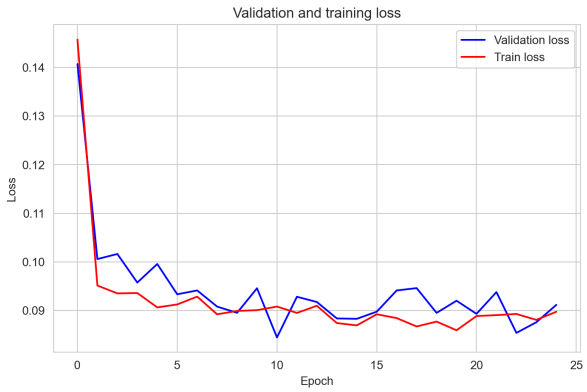


Figura 10: Función de pérdida MSE en datos de entrenamiento y validación durante el fine-tuning.

Otra forma de visualizar el aprendizaje del modelo es observar su mejora a medida que avanzan las épocas de entrenamiento (ver Figura 11). De manera notable, se puede observar que el objeto (en este caso, un hígado) va adquiriendo mejoras en su estructura anatómica a medida que aumentan las épocas de entrenamiento.

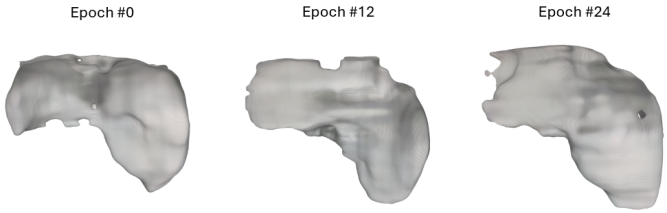


Figura 11: Evolución del objeto (hígado) a lo largo de las épocas de entrenamiento.

### *VI-B. Comparación cuantitativa*

Para la evaluación cuantitativa, utilizamos la métrica MSE en el dataset de evaluación en nuestro modelo y lo comparamos con el MSE de Shap-e sin fine-tuning. En la Tabla I se puede observar la diferencia entre los dos modelos. Se evidencia que nuestro modelo supera a Shap-e, obteniendo un MSE menor por dos órdenes de magnitud. Esto es similar a lo que ocurre en otros modelos generativos. Generalmente, los modelos generativos reconocidos son entrenados con datos genéricos, como los modelos de lenguaje y los modelos de imágenes, y son buenos para tareas generales. Sin embargo, si se desea especializar en una tarea específica (e.g., generación de textos de un género literario específico para los modelos de lenguaje o imágenes reales de personas para los modelos de visión), lo mejor es aprovechar el conocimiento del modelo y ajustarlo a la tarea específica con datos del dominio. Podemos observar que esto también se cumple en nuestro caso de generación de objetos biomédicos. Sin embargo, aún falta validar esto de manera visual. En la Sección VI-C veremos si esto también se cumple con la evaluación cualitativa.

| **Model** | **MSE in latents** |
|---|---|
| Shap-MeD | 0.089 |
| Shap-e | 0.147 |

Tabla I: Tabla comparativa entre nuestro modelo y Shap-e, utilizando como métrica el MSE de los latentes en el dataset de evaluación.

### *VI-C. Comparación cualitativa*

Comparamos otros modelos de texto a 3D con el nuestro. En la Figura 12, se puede observar que nuestro modelo presenta una mayor precisión en la estructura de los órganos. Aunque LGM también obtiene buenos resultados, al analizar los detalles, el hígado no tiene bordes muy definidos, lo que reduce la precisión de su estructura. Además, la aorta generada por LGM no tiene una buena coherencia estructural, ya que presenta partes desconectadas.

Por otro lado, se pueden apreciar ventajas aún más significativas en comparación con Shap-e. Por ejemplo, la aorta en Shap-e se representa incorrectamente como un corazón con otros componentes, y los demás órganos carecen de la precisión necesaria para representar fielmente su estructura real.

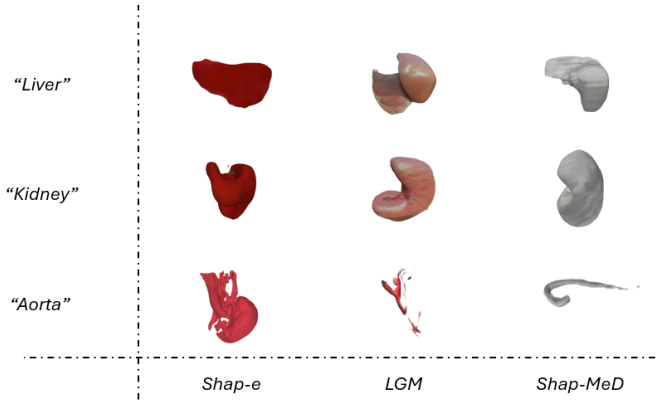


Figura 12: Comparación con otros modelos texto-a-3D.

## VII. Conclusiones y trabajo futuro

El propósito de este proyecto fue desarrollar una herramienta capaz de transformar texto en objetos 3D específicos para el ámbito biomédico, con el fin de agilizar el proceso de modelado. El modelado en 3D desempeña un papel fundamental en el campo de la salud, ya que se utiliza en diversas áreas, como simulación y planificación quirúrgica, diseño de implantes prostéticos personalizados, desarrollo de implantes dentales, educación médica, y creación de modelos anatómicos.

Para lograr esto, empleamos un modelo de OpenAI llamado Shap-e, un modelo generativo de texto a objetos 3D entrenado con más de un millón de objetos genéricos. Luego, realizamos un fine-tuning con objetos biomédicos específicos, como hígados, aortas, riñones y corazones. En el conjunto de evaluación, conseguimos un MSE de 0.089, mientras que Shap-e registró un MSE de 0.147. Además, llevamos a cabo una evaluación cualitativa donde nuestro modelo superó a otros modelos más grandes considerados como el estado del arte.

A pesar de los buenos resultados obtenidos, este proyecto aún tiene un potencial de crecimiento considerable. En primer lugar, es crucial considerar tanto la calidad como la cantidad del conjunto de datos utilizado. Hemos observado que algunos objetos no están renderizados correctamente, lo que disminuye el rendimiento del fine-tuning del modelo. Además, es importante mencionar que nuestro modelo supera a otros modelos más grandes debido a su especialización en nuestra tarea específica. Sin embargo, creemos que podríamos obtener resultados aún mejores si logramos realizar un fine-tuning en alguno de estos modelos más grandes, como LGM o LRM. Estos modelos, además de ser más complejos, incorporan otros mecanismos que podrían mejorar el rendimiento en comparación con Shap-e.

## Referencias


[1] NHS. "IMPROVING OUTCOMES THROUGH PERSONALISED MEDICINE". En: (ago. de 2016), págs. 1-18. URL: https://www.england.nhs.uk/wp-content/uploads/2016/09/improving-outcomes-personalised-medicine.pdf.

[2] Alvaro Goyanes et al. "Patient acceptability of 3D printed medicines". En: *International Journal of Pharmaceutics* 530.1 (2017), págs. 71-78. ISSN: 0378-5173. DOI: https://doi.org/10.1016/j.ijpharm.2017.07.064. URL: https://www.sciencedirect.com/science/article/pii/S0378517317306725.

[3] C. Barnatt. *3D Printing: The Next Industrial Revolution*. ExplainingTheFuture.com, 2013. ISBN: 9781484181768. URL: https://books.google.com.co/books?id=Tvb6nAEACAAJ.

[4] A. Chowdhry. *What can 3D printing do? Here are 6 creative examples*. Accessed: 2024-03-08. 2013. URL: https://www.forbes.com/sites/amitchowdhry/2013/10/08/what-can-3d-printing-do-here-are-6-creative-examples/?sh=6901d5e15491.

[5] C. Lee Ventola. "Medical Applications for 3D Printing: Current and Projected Uses". En: *P & T : a peer-reviewed journal for formulary management* 39.10 (oct. de 2014), págs. 704-11. ISSN: 1052-1372. DOI: 10.4069/pj.2014.4910704. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4189697/.

[6] Cecilia Sahlgren et al. "Tailored Approaches in Drug Development and Diagnostics: From Molecular Design to Biological Model Systems". En: *Adv Healthc Mater* 6.21 (nov. de 2017), 10.1002/adhm.201700258. ISSN: 2192-2659 (Electronic), 2192-2640 (Linking). DOI: 10.1002/adhm.201700258. URL: https://onlinelibrary.wiley.com/doi/full/10.1002/adhm.201700258.

[7] S. V. Shinde et al., eds. *Disruptive Developments in Biomedical Applications*. 1st. CRC Press, 2022. DOI: 10.1201/9781003272694. URL: https://doi.org/10.1201/9781003272694.

[8] Karla Jiménez Oliver. "Overview of Organ Donation". En: *Mexican Journal of Medical Research ICSA* 11.21 (ene. de 2023), págs. 55-63. DOI: 10.29057/mjmr.v11i21.10007. URL: https://repository.uaeh.edu.mx/revistas/index.php/MJMR/article/view/10007.

[9] XuanQi Zheng et al. "3D Bioprinting in Orthopedics Translational Research". En: *Journal of Biomaterials Science, Polymer Edition* 30.13 (2019). PMID: 31124402, págs. 1172-1187. DOI: 10.1080/09205063.2019.1623989. eprint: https://doi.org/10.1080/09205063.2019.1623989. URL: https://doi.org/10.1080/09205063.2019.1623989.

[10] Sarah J Trenfield et al. "3D Printing Pharmaceuticals: Drug Development to Frontline Care". En: *Trends in Pharmacological Sciences* 39.5 (mayo de 2018), págs. 440-451. ISSN: 1873-3735 (Electronic), 0165-6147 (Linking). DOI: 10.1016/j.tips.2018.02.006. URL: https://www.sciencedirect.com/science/article/pii/S0165614718300440.

[11] Wavefront Technologies. "Wavefront Advanced Visualizer". En: (1980).

[12] Ben Mildenhall et al. "NeRF". En: *Communications of the ACM* 65 (1 ene. de 2022), págs. 99-106. ISSN: 15577317. DOI: 10.1145/3503250.

[13] Tianchang Shen et al. *Deep Marching Tetrahedra: a Hybrid Representation for High-Resolution 3D Shape Synthesis*. URL: https://nv-tlabs.github.io/DMTet/..

[14] Alex Nichol et al. "Point-E: A System for Generating 3D Point Clouds from Complex Prompts". En: (dic. de 2022). URL: http://arxiv.org/abs/2212.08751.

[15] Ashish Vaswani et al. *Attention Is All You Need*. arXiv:1706.03762 [cs]. Ago. de 2023. DOI: 10.48550/arXiv.1706.03762. URL: http://arxiv.org/abs/1706.03762 (visitado 09-03-2024).

[16] Jonathan Ho, Ajay Jain y Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. arXiv: 2006.11239 [cs.LG].

[17] Aaron van den Oord, Oriol Vinyals y Koray Kavukcuoglu. *Neural Discrete Representation Learning*. 2018. arXiv: 1711.00937 [cs.LG].

[18] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV].

[19] Alex Nichol et al. *GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models*. 2022. arXiv: 2112.10741 [cs.CV].

[20] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam, 2018. URL: http://www.blender.org.

[21] Heewoo Jun y Alex Nichol. "Shap-E: Generating Conditional 3D Implicit Functions". En: (mayo de 2023). URL: http://arxiv.org/abs/2305.02463.

[22] Yicong Hong et al. *LRM: Large Reconstruction Model for Single Image to 3D*. arXiv:2311.04400 [cs]. Nov. de 2023. DOI: 10.48550/arXiv.2311.04400. URL: http://arxiv.org/abs/2311.04400 (visitado 08-03-2024).

[23] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. arXiv:2010.11929 [cs] version: 2. Jun. de 2021. DOI: 10.48550/arXiv.2010.11929. URL: http://arxiv.org/abs/2010.11929 (visitado 09-03-2024).

[24] Mathilde Caron et al. *Emerging Properties in Self-Supervised Vision Transformers*. arXiv:2104.14294 [cs]. Mayo de 2021. DOI: 10.48550/arXiv.2104.14294. URL: http://arxiv.org/abs/2104.14294 (visitado 10-03-2024).

[25] Robin Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models*. arXiv:2112.10752 [cs]. Abr. de 2022. DOI: 10.48550/arXiv.2112.10752. URL: http://arxiv.org/abs/2112.10752 (visitado 10-03-2024).

[26] Jiaxiang Tang et al. "LGM: Large Multi-View Gaussian Model for High-Resolution 3D Content Creation". En: *arXiv preprint arXiv:2402.05054* (2024).

[27] Jianning Li et al. *MedShapeNet – A Large-Scale Dataset of 3D Medical Shapes for Computer Vision*. 2023. arXiv: 2308.16139 [cs.CV].

[28] Matt Deitke et al. "Objaverse: A Universe of Annotated 3D Objects". En: *arXiv preprint arXiv:2212.08051* (2022).

[29] Apose. "Aspose.3D for Python via .NET". En: (2002). URL: https://reference.aspose.com/3d/python-net/.

[30] Qian-Yi Zhou, Jaesik Park y Vladlen Koltun. "Open3D: A Modern Library for 3D Data Processing". En: *arXiv:1801.09847* (2018).