ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

PROTOTIPO DE DESCRIPCIÓN DE FIGURAS GEOMÉTRICAS PARA NO VIDENTES

DESCRIPCIÓN DE FIGURAS GEOMÉTRICAS (TIKZ SVG) USANDO UN LLM

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN

MATEO STEVEN VITERI HIDALGO

mateo.viteri@epn.edu.ec

DIRECTOR: PhD. ANA MARÍA ZAMBRANO VIZUETE

ana.zambrano@epn.edu.ec

DMQ, Agosto 2024

CERTIFICACIONES

Yo, Mateo Steven Viteri Hidalgo declaro que el Trabajo de Integración Curricular aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

Mateo Steven Viteri Hidalgo

Certifico que el presente trabajo de integración curricular fue desarrollado por Mateo Steven Viteri Hidalgo, bajo mi supervisión.

Dra. Ana María Zambrano Vizuete

DIRECTOR

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

_____

SR. MATEO STEVEN VITERI HIDALGO

_____

DRA. ANA MARÍA ZAMBRANO VIZUETE

DEDICATORIA

A mis padres y mi hermano que han sido un pilar fundamental en mi vida, que con su amor y apoyo incondicional han sido la base para lograr cada éxito en mi vida personal y académica. A ellos que han forjado mis principios y valores mediante el cariño y su ejemplo para ser una persona de bien, que con su motivación constante me han enriquecido de la fortaleza necesaria para escalar cada peldaño y superar cualquier obstáculo en esta etapa.

A toda mi familia que siempre ha estado pendiente de mi y me ha acompañado en todo momento durante esta etapa académica.

A todos mis amigos, compañeros de profesión y personas que compartieron conmigo desde el inicio de mi formación universitaria, que siempre me motivaron a seguir adelante y no rendirme, aportando momentos agradables y una convivencia que dejó una huella imborrable en mi vida.

Una dedicatoria especial a mi querido abuelo Julio Viteri Espinel, quien siempre estuvo a mi lado motivándome y dándome sus sabios consejos para sobrellevar cualquier eventualidad que se me pueda presentar. Sin su cariño y apoyo incondicional esto no hubiera sido posible. Aunque no pudo acompañarme hasta el final del camino, siempre lo recordaré y lo mantendré vivo en mi corazón.

A todos ellos les dedico este logro.

Mateo Steven Viteri Hidalgo

AGRADECIMIENTO

Un especial y sentido agradecimiento a la Dra. Ana Zambrano, quien fue la directora del presente Trabajo de Integración Curricular, y al Dr. Felipe Grijalva por su orientación experta, su paciencia y su compromiso inquebrantable durante este proceso. Su sabiduría y apoyo han sido fundamentales para completar este Trabajo de Integración Curricular.

También quiero agradecer a todos los ingenieros que han colaborado en mi formación personal y profesional, brindándome su confianza y paciencia, además de ser un ejemplo en la enseñanza. Agradezco profundamente a los ingenieros Julio Caiza, Pablo Hidalgo, Danny Guamán y Ana Rodríguez por su gran vocación para transmitir sus conocimientos y demostrar una educación de calidad.

Finalmente, agradezco a todo el personal administrativo y a la coordinación de la carrera que me ha apoyado en diversas circunstancias dentro de esta etapa académica.

ÍNDICE DE CONTENIDO

RESUMEN

Este documento presenta el Trabajo de Integración Curricular que persigue el objetivo de desarrollar un servicio que genere descripciones detalladas y contextualizadas de figuras en formato SVG o TikZ, presentes en documentos académicos y científicos en formato LaTeX, mediante el uso de modelos LLM para la generación automática, con el fin de facilitar el entendimiento, recreación e interpretación de estos elementos visuales para todas las personas, incluso personas con un cierto grado de discapacidad visual, contribuyendo así al aprendizaje inclusivo.

Durante el primer capitulo se expone la motivación e investigación de la problemática que se pretende resolver mediante el uso de tecnologías y soluciones de software. Se presenta el objetivo general de crear un servicio que utilice Modelos de Lenguaje Grandes (LLM), en particular el modelo Meta AI *Llama 2*, junto con técnicas como *Retrieval-Augmented Generation* (RAG) para generar descripciones contextualizadas de figuras en formatos como SVG o TikZ.

El capítulo dos partes por un estudio del Estado del Arte en el uso de LLM para la descripción de figuras, presentando diferentes proyectos e investigaciones que sirven de base para el desarrollo de este Trabajo de Integración Curricular. Así también, se definen todas las consideraciones de hardware y software que se requieren para implementar el servicio, y se describe la arquitectura del sistema y las fases de desarrollo, incluyendo el diseño, codificación e implementación del servicio generador de descripciones.

Finalmente, en el capitulo tres se resumen y se analizan los resultados obtenidos del servicio generador de descripciones, estableciendo escenarios de prueba y validando su funcionalidad mediante una encuesta cerrad. Así también, se establecen las conclusiones y recomendaciones sobre el Trabajo de Integración Curricular, tomando en cuenta aspectos para un trabajo futuro sobre este.

PALABRAS CLAVE: Descripciones, documentos académicos, figuras, Modelos de Lenguaje Grandes, procesamiento textual, retroalimentación, servicio generador.

ABSTRACT

This document presents the Curricular Integration Project that aims to develop a service to generate detailed and contextualized descriptions of figures in SVG or TikZ format, present in academic and scientific documents in LaTeX format, using LLM models for automatic generation. The goal is to facilitate the understanding, recreation, and interpretation of these visual elements for everyone, including people with a certain degree of visual impairment, thus contributing to inclusive learning.

The first chapter presents the motivation and research of the problem that is intended to be solved through the use of technologies and software solutions. It introduces the general objective of creating a service that uses Large Language Models (LLMs), particularly the Meta AI Llama 2 model, along with techniques like Retrieval-Augmented Generation (RAG) to generate contextualized descriptions of figures in formats like SVG or TikZ.

Chapter two begins with a study of the State of the Art in the use of LLMs for figure description, presenting different projects and research that serve as the foundation for the development of this Curricular Integration Project. It also defines all the hardware and software considerations required to implement the service and describes the system architecture and development phases, including the design, coding, and implementation of the description generator service.

Finally, chapter three summarizes and analyzes the results obtained from the description generator service, establishing test scenarios and validating its functionality through a closed survey. It also sets out the conclusions and recommendations for the Curricular Integration Project, considering aspects for future work on this topic.

KEYWORDS: Academic documents, descriptions, feedback, figures, generator service, Large Language Models, text processing.

1 INTRODUCCIÓN

En la era actual, caracterizada por los avances tecnológicos, persisten desafíos significativos en el ámbito de la educación y el aprendizaje inclusivo para personas con discapacidades físicas. Particularmente, los estudiantes con discapacidad visual enfrentan limitadas alternativas para su auto preparación y comprensión de diversos temas de estudio. Los documentos académicos y científicos publicados, tales como libros, artículos y otros materiales, carecen con frecuencia de soluciones efectivas para facilitar el entendimiento de elementos ilustrativos por parte de este grupo estudiantil. Esta carencia complejiza su curva de aprendizaje y establece una brecha sustancial en comparación con aquellos estudiantes sin discapacidades visuales.

La interpretación de elementos visuales en documentos académicos y científicos contribuye notablemente a la construcción de nuevos conocimientos. Sin embargo, las alternativas disponibles para estudiantes no videntes son escasas, ya que no basta con mencionar simplemente lo que representa una figura, sino que es imperativo buscar mecanismos que permitan suplir, en la medida de lo posible, el proceso de visualización e interpretación de las ilustraciones. Esto posibilitaría que el estudiante pueda recrearlas mentalmente y, por consiguiente, comprenderlas para complementar su conocimiento.

Motivado por el reto de reducir esta brecha a través de las nuevas tecnologías, el presente Trabajo de Integración Curricular propone implementar y desplegar un prototipo de servicio capaz de generar descripciones detalladas de las diversas figuras contenidas en documentos en formato LaTeX. Se pretende que estas descripciones contribuyan al proceso de entendimiento, recreación e interpretación de las figuras incluso para personas con cierta discapacidad visual.

Para llevar esto a cabo, acorde con ciertas pautas que menciona la organización W3C [1] y las recomendaciones para descripción de figuras en artículos científicos expuestos por *Enago Academy* [2], se emplean modelos de lenguaje natural (LLM, por sus siglas en inglés *Large Lenguage Model*) en conjunto con técnicas de *Prompt Engineering*, con el fin de generar automáticamente descripciones contextualizadas de figuras codificadas en formatos como SVG o TikZ, presentes en artículos científicos. Esta solución cimenta sus bases con la línea de diversas investigaciones sobre el uso de inteligencia artificial para mejorar las descripciones de ilustraciones para personas con discapacidad visual [3].

La solución que se presenta en el documento pretende contribuir significativamente al entendimiento de elementos visuales en documentos técnicos y científicos por parte de

personas con discapacidad visual, fungiendo como una alternativa que propenda al aprendizaje inclusivo en las publicaciones académicas y científicas, utilizando técnicas y tecnologías de tendencia accesibles para el sector de la investigación.

1.1 OBJETIVO GENERAL

Desarrollar un servicio que genere descripciones detalladas y contextualizadas de figuras en formato SVG o TikZ, presentes en documentos académicos y científicos en formato LaTeX, mediante el uso de modelos LLM para la generación automática, con el fin de facilitar el entendimiento, recreación e interpretación de estos elementos visuales para personas con discapacidad visual, contribuyendo así al aprendizaje inclusivo.

1.2 OBJETIVOS ESPECÍFICOS

Se plantean los siguientes objetivos específicos:

1. Analizar las capacidades de los modelos de lenguaje grandes (LLM) de acceso libre en tareas visuales mediante procesamiento textual, apoyándose de un exhaustivo estudio del Estado del Arte.
2. Desarrollar y desplegar un prototipo de servicio que demuestre el funcionamiento de la descripción de figuras codificadas, haciendo uso de técnicas de *Prompt Engineering* y de generación contextualizada en las descripciones generadas por el modelo LLM.
3. Evaluar el rendimiento y la calidad de descripciones generadas mediante la selección de diferentes documentos LaTeX que contengan figuras codificadas, definiendo métricas cuantitativas y cualitativas.

1.3 ALCANCE

El programa propuesto en el presente componente del Trabajo de Integración Curricular tiene que ver con el uso de modelos LLM de código abierto para generar descripciones automáticas y contextualizadas de figuras codificadas, presentes en artículos científicos en formato LaTeX. Este programa será desarrollado en lenguaje Python, haciendo uso de librerías para consumir el modelo LLM cargado localmente mediante APIs. Se emplearán técnicas de generación contextualizada para que las descripciones de las figuras se complementen con el código que las genera y con lo que el documento LaTeX mencione sobre ellas. Como resultado final se pretende implementar un script de Python que permita extraer el texto y las figuras de un documento de LaTeX, específicamente artículos científicos, y aplicar técnicas de Prompt Engineering y de generación

contextualizada para obtener descripciones detalladas de todas las figuras codificadas presentes en dicho documento, mejorando así la experiencia de usuario y la interpretación de estas figuras.

Para llevar a cabo el presente proyecto, se destacan las siguientes fases:

A. Fase de Análisis

Se realizará un estudio del Estado del Arte relacionado con este Trabajo de Integración Curricular, proporcionando una base sólida para el desarrollo del prototipo de servicio. Se usará el modelo "*Meta AI Llama 2*" (*Large Language Model Meta AI*) [4], analizando su funcionamiento, los requisitos necesarios, tanto en términos de hardware como de software, sus capacidades en el procesamiento textual de figuras codificadas, y la implementación del modelo en un servidor para su uso como generador de texto.

B. Fase de Diseño

Se elaborará un esquema de diseño del programa, mediante la definición de tareas y actividades, acorde a una metodología Kanban. Se elaborará un diagrama de flujo que describa el funcionamiento del programa generador de descripciones, desde el uso del LLM en un script de Python hasta la implementación de técnicas de contextualización y de *Prompt Engineering* para darle contexto al modelo, obteniendo así las descripciones requeridas.

C. Fase de Implementación

Partiendo de las actividades definidas en la fase de Diseño y del diagrama de flujo, se empezará con la codificación del servicio a manera de script de Python, haciendo uso de las librerías disponibles que permitan llevar a cabo el esquema de diseño para la generación de descripciones.

D. Fase de Pruebas

Una vez completada la fase anterior, se procederá a poner en marcha diferentes pruebas del programa con varios artículos en formato LaTeX que contengan figuras en formato SVG o TikZ, evaluando las descripciones que el servicio genere desde un enfoque de calidad y coherencia.

1.4 MARCO TEÓRICO

En este apartado se destacan diferentes tópicos utilizados como base teórica para el desarrollo de este componente, dando cumplimiento con las fases mencionadas en el apartado anterior.

1.4.1 Modelos de Lenguaje Grandes (LLM) [5] [6]

Los Modelos de Lenguaje Grandes (LLM, por sus siglas en inglés *Large Language Model)* son sistemas funcionales de Inteligencia Artificial basados en redes neuronales profundas entrenadas sobre grandes cantidades de datos. Estos modelos tienen la capacidad de comprender y generar lenguaje natural de manera coherente y contextualizada.

Estos modelos son la base de lo que hoy se conocen como “Chatbots de IA”, capaces de generar respuestas en lenguaje natural a partir de un requerimiento del usuario, adoptando un papel de asistente, brindando la información que se le solicite, desde su base de datos pre entrenada. Han perfeccionado las capacidades de IA generativa como la inferencia de texto, la calidad de respuesta y su precisión, la coherencia de la generación a partir de un contexto definido, inferencia para parafraseo y traducción, generación de códigos de programación, entre muchas otras.

La base del funcionamiento de los LLM se centra en la parametrización y captura de patrones complejos en el lenguaje. Estos son entrenados con grandes cantidades de datos de texto, como libros, artículos y páginas web, utilizando técnicas de aprendizaje profundo, permitiéndoles aprender las relaciones estadísticas entre las palabras y frases, y cómo se usan en diferentes contextos.

Así también, la composición de un modelo de lenguaje largo se define por redes neuronales artificiales, mismas que funcionan como un símil del cerebro humano, permitiéndole producir textos fluidos y coherentes. Estas redes neuronales son las que se encargan de reconocer estos patrones y parámetros en los datos de entrenamiento y a generar nuevos datos similares.

Todo este proceso se sintetiza en una arquitectura, denominada Arquitectura de Transformadores, la cual rige el funcionamiento de las redes neuronales, permitiendo a los LLM prestar atención a diferentes partes del texto de entrada de manera simultánea, mejorando sustancialmente la generación de texto de salida.

Esta arquitectura es la base para el procesamiento de Texto en Lenguaje Natural (NLP) [7], compuesta de un codificador y un decodificador. El codificador procesa el texto de

entrada y genera una representación vectorial que captura el significado del texto. El decodificador utiliza esta representación para generar texto de salida, palabra por palabra.

Además, con el extenso entrenamiento, les permite adquirir un profundo conocimiento de la gramática, la semántica y las relaciones conceptuales mediante un aprendizaje no supervisado y autónomo, consiguiendo la predicción de la siguiente palabra más apropiada según el contexto y los patrones aprendidos.

No obstante, el rendimiento de estos modelos puede ser mejorado aún más mediante técnicas como el “ajuste rápido”, *Promp Engineering* o el Aprendizaje por Refuerzo con Retroalimentación Humana (RLHF). Estas tácticas permiten mitigar sesgos, discursos ofensivos y respuestas objetivamente incorrectas, conocidas como "alucinaciones", que suelen ser subproductos no deseados del entrenamiento con grandes cantidades de datos no estructurados. Abordar estos aspectos es crucial para garantizar que los LLM de nivel empresarial sean confiables, no expongan a las organizaciones a responsabilidades legales ni dañen su reputación.

1.4.2 Meta AI Llama 2 [8] [9]

Con el evidente crecimiento de los modelos LLM para su uso tanto comercial como en el campo de la investigación, varias empresas y startups han lanzado LLM de código abierto, disponibles para su uso e implementación. Uno de los más relevantes y ampliamente usados en el campo de la investigación es Llama, de Meta AI.

Los modelos Llama se destacan por su eficiencia en inferencia y su rendimiento comparable a modelos más grandes, a pesar de tener un tamaño más reducido. La versión de Llama más estable es “*Llama 2*”, fungiendo como un LLM de gran desempeño en tareas de comprensión y generación de lenguaje natural. Este modelo ha sido previamente entrenado y ajustado con un tamaño que varía en escala de entre 7 mil millones y 70 mil millones de parámetros, basándose en una arquitectura *Transformer*, características que hacen de este modelo una opción balanceada entre rendimiento en razonamiento y respuesta, así como en eficiencia computacional.

En la siguiente Tabla 1 se puede destacar algunas características de “*Llama 2*” respecto de otros modelos LLM de código abierto.

Tabla 1.1. Comparación de las características del LLM "*Llama 2"* respecto de otros LLM de código abierto.

Modelo	Parámetros	Fortalezas	Debilidades
Llama 2 Large	70 mil millones	Alto rendimiento en tareas de razonamiento y respuesta a preguntas	Menos preciso que otros LLM para generar texto creativo
Llama 2 Medium	7 mil millones	Buen equilibrio entre rendimiento y eficiencia computacional	Menos preciso que otros LLM para tareas específicas
GPT-3	175 mil millones	Genera texto de alta calidad, traduce idiomas, escribe diferentes tipos de contenido creativo	Requiere recursos computacionales considerables
Jurassic-1 Jumbo	178 mil millones	Genera diferentes formatos de texto creativo	Menos preciso que otros LLM para responder preguntas
Megatron-Turing NLG	530 mil millones	Responde preguntas de manera completa e informativa	Requiere recursos computacionales muy considerables
BLOOM	176 mil millones	Comprensión y generación de texto multilingüe	Menos preciso que otros LLM para tareas específicas

1.4.3 Oobabooga Web UI como herramienta de testing de LLM [10] [11]

Para poder probar los diferentes modelos LLM de código abierto que se encuentran disponibles, se puede hacer uso de diferentes herramientas para *testing* de modelos. Una de ellas es la Web UI de *Oobabooga*. *Oobabooga Web UI* es una herramienta de código abierto desarrollada en *Gradio* (biblioteca de código abierto de Python), diseñada para facilitar el *testing* y la evaluación de modelos de *Machine Learning*, como los LLM. Esta herramienta proporciona una interfaz web intuitiva que permite interactuar con los LLM a través de una conversación en línea.

Oobabooga Web UI ofrece una interfaz versátil con tres modos diferentes: predeterminado (dos columnas), cuaderno y chat, lo que permite adaptarse a las necesidades y preferencias de cada usuario.

Una de las características principales es su compatibilidad con múltiples backends de modelos, incluyendo Transformers, llama.cpp (a través de llama-cpp-python[1]),

[1] llama.cpp es una implementación en C++ del modelo de lenguaje LLaMa, optimizada para eficiencia y rendimiento en diversas plataformas. Si se requiere su integración en entornos de desarrollo Python se puede usar el puente llama-cpp-python.

ExLlamaV2, *AutoGPTQ*, *AutoAWQ, GPTQ-for-LLaMa y QuIP*[2]. Esta flexibilidad permite cargar y ejecutar una amplia gama de modelos LLM, incluso algunos modelos comerciales, facilitando su comparación y evaluación.

La herramienta ofrece una gran cantidad de extensiones, tanto integradas como contribuidas por la comunidad, que amplían sus capacidades. Algunas de estas extensiones incluyen *Coqui TTS*[3] para salidas de voz realistas, *Whisper STT*[4] para entradas de voz, traducción, pipelines multimodales, bases de datos vectoriales, integración con *Stable Diffusion* y muchas más. Además, cuenta con soporte para LoRA (*Low-Rank Adaptation*), lo que permite entrenar nuevos LoRAs con datos propios y cargar/descargar LoRAs durante la generación de texto.

La gran ventaja de la *Web UI Oobabooga* es que incluye una API consumible y compatible con OpenAI [12], lo que facilita la integración con otras aplicaciones y servicios, siendo un complemento necesario para su uso en scripts o programas de terceros.

1.4.4 Técnica Retrieval-Augmented Generation (RAG) [13]

La técnica *Retrieval-Augmented Generation* (RAG) es un paradigma innovador del Procesamiento de Lenguaje Natural, proveniente del Aprendizaje por Refuerzo con Retroalimentación Humana (RLHF)[5], el cual combina las fortalezas de modelos de recuperación y modelos generadores de lenguaje natural, con el objetivo de mejorar la calidad y precisión del texto generado aprovechando mecanismos externos de recuperación de información.

El proceso RAG se compone de dos etapas fundamentales. En la primera etapa, se utiliza un modelo de recuperación para identificar y extraer información relevante de una base de conocimientos externa, como documentos, artículos o bases de datos específicas del dominio, pasajes relevantes que puedan proporcionar contexto o información específica requerida para una consulta determinada. En la segunda etapa, esta información recuperada se utiliza como entrada adicional para un modelo

[2] ExLlamaV2 es una herramienta diseñada para optimizar la ejecución de modelos de lenguaje grandes en hardware específico. AutoGPTQ, AutoAWQ, GPTQ-for-LLaMa y QuIP son técnicas de cuantización de modelos LLM para reducir su tamaño y acelerar su inferencia.

[3] Coqui TTS es un sistema de texto a voz (Text-to-Speech) de código abierto que permite la generación de voces sintéticas realistas.

[4] Whisper STT es lo opuesto de Coqui TTS, diseñado para proporcionar transcripciones precisas y rápidas de audio en diversos idiomas

[5] RLHF (Reinforcement learning from human feedback) es una técnica en la cual los modelos de inteligencia artificial se entrenan utilizando retroalimentación proporcionada por humanos para mejorar su rendimiento.

generador, como un LLM, el cual produce respuestas o textos coherentes y contextualmente precisos en base a la solicitud inicial.

El funcionamiento combinado de estos dos componentes desarrolla una arquitectura de generación con retroalimentación, resumida en las siguientes fases [14]:

a) **Codificación de consultas:** En esta fase una consulta de entrada es codificada en una representación vectorial utilizando un codificador de red neuronal. Estos vectores codificados se los conoce como *Embeddings*, que no son más que representaciones de la consulta de entrada en un espacio vectorial diferente, es decir, una reinterpretación del texto de entrada en otro dominio, generalmente de tipo numérico.

b) **Recuperación**: La consulta codificada se utiliza para buscar documentos relevantes en un corpus grande. Esto se puede hacer utilizando técnicas de recuperación densas (por ejemplo, utilizando similitud de vectores) o métodos dispersos tradicionales como la intervención humana para proporcionar los documentos donde se requiera obtener la información.

c) **División y codificación de documentos:** Con los documentos recuperados, para poder ser transformados a *Embeddings*, se realiza previamente un proceso de fragmentación o división, donde cada fragmento es codificado a una representación vectorial, al igual que con la petición de entrada.

d) **Indexación:** Los *Embeddings* de los documentos y de la petición de entrada se fusionan para formar un vector de indexación, conocido como *Indexer*, que se convertirá en la entrada completa para el modelo generador.

e) **Recuperación:** En este punto donde se tiene el vector de indexación con los *Embeddings* de la entrada y los documentos, se realiza el proceso de recuperación, llamado *Retrieval* en inglés, donde se realizan operaciones de similitud y de búsqueda de coincidencias entre todos los elementos del vector, para así obtener un nuevo vector de *Embeddings,* denominado *Retriever*, pero que contenga únicamente la información relevante y coherente con la entrada inicial. Las operaciones realizadas para obtener el *Retriever* están basadas generalmente en operaciones matemáticas o a su vez en algoritmos de minería de datos.

f) **Generación:** El *Retriever* es introducido como entrada en el modelo generativo, y este produce el texto de salida final. Este texto de salida pasa previamente por

un decodificador de *Embeddings,* denominado *Parser,* que es lo que se le entrega al usuario final.

En la Figura 1.1 Arquitectura de la técnica de recuperación-generación aumentada (RAG). se puede observar una síntesis gráfica de la arquitectura de la técnica RAG, combinando las funciones del modelo recuperador y del modelo generador en un mismo eje funcional.

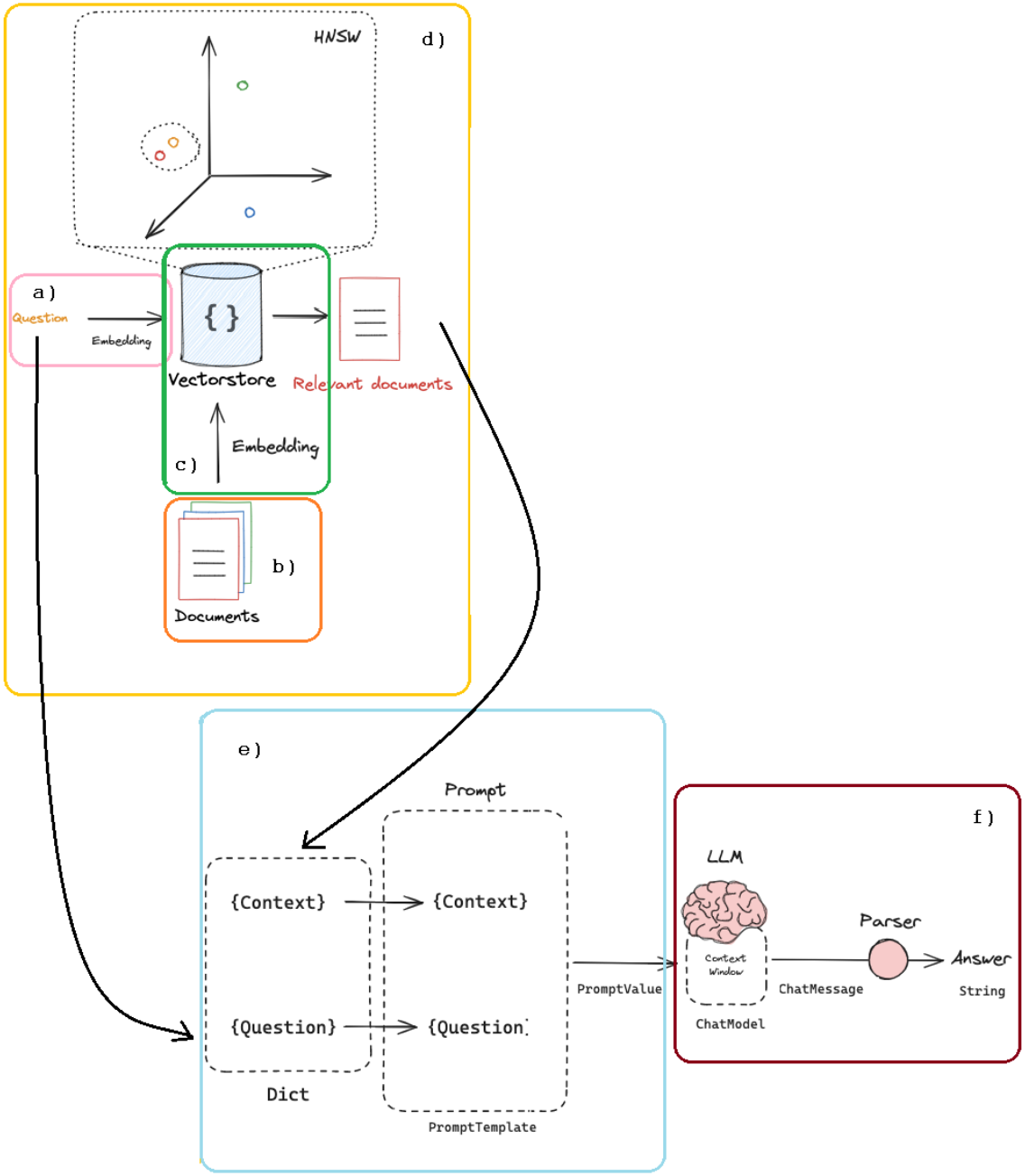


Figura 1.1 Arquitectura de la técnica de recuperación-generación aumentada (RAG).

1.4.5 Metodología Kanban [15]

La metodología Kanban, derivada del término japonés que significa "letrero" o "cartel publicitario", es un método visual de gestión del flujo de trabajo destinado a optimizar la eficiencia y la productividad. Originalmente desarrollado por Toyota a finales de la década de 1940 para mejorar los procesos de fabricación, Kanban se ha adaptado a diversas industrias, incluyendo el desarrollo de software, operaciones de TI y la gestión de proyectos. Se centra en controlar el progreso de las tareas de un proyecto y el tiempo necesario para completarlas, enfatizando la mejora continua, la visualización del trabajo y la limitación del trabajo en progreso para mejorar el flujo y reducir el desperdicio.

Kanban define cuatro principios básicos que son los siguientes [16]:

1. **Enfocarse en lo que se hace ahora:** Comprender plenamente los procesos actuales.
2. **Adoptar un desarrollo evolutivo incremental:** Promover mejoras continuas e incrementales en lugar de cambios radicales.
3. **Respetar los roles y responsabilidades:** Reconocer y utilizar los roles y responsabilidades existentes como base para la mejora.
4. **Fomentar liderazgo multinivel:** Empoderar a todos los miembros del equipo para contribuir a la mejora de los procesos.

Así también, Kanban define seis prácticas básicas que complementan estos principios, que son [16]:

1. **Visualización del flujo de trabajo:** Utilizar un tablero virtual o físico para representar visualmente el flujo de trabajo. Este tablero se denomina Tablero Kanban.
2. **Limitación del trabajo en progreso (WIP):** Establecer límites en la cantidad de elementos de trabajo en cada etapa para evitar la sobrecarga y garantizar un flujo fluido.
3. **Gestión activa del flujo de trabajo:** Supervisar el flujo de elementos de trabajo a lo largo del proceso para identificar cuellos de botella y mejorar la eficiencia.
4. **Creación de políticas de proceso explícitas:** Definir y comunicar claramente las políticas de proceso para garantizar la coherencia y la comprensión.

5. **Implementación circuitos de retroalimentación:** Utilizar mecanismos de retroalimentación como revisiones periódicas para recopilar comentarios y realizar los ajustes necesarios.
6. **Evolución colaborativa**: Desarrollar e implementar cambios controlados que contribuyan a una mejora de los procesos de forma colaborativa.

La metodología Kanban establece, en las prácticas básicas antes mencionados, que se debe ilustrar el flujo de trabajo mediante un tablero. Este tablero que propone se denomina Tablero Kanban.

En su forma más básica, un Tablero Kanban cuenta con tres columnas: "Por Hacer", "En Progreso" y "Finalizado". Sin embargo, se pueden agregar o modificar columnas para adaptarlo a las necesidades específicas de cada proyecto [17]. Cada tarea se representa mediante una tarjeta que se mueve a través de las columnas conforme avanza el trabajo, proporcionando información relevante sobre la tarea, como descripciones, responsables, estimaciones de tiempo y requisitos para la siguiente etapa [17].

En la Figura 1.2 se puede observar un esquema básico de un Tablero Kanban que es de vital importancia para el desarrollo de este componente.

Figura 1.2 Esquema básico de un Tablero Kanban [18].

2 METODOLOGÍA

En este capítulo se presentará, en primera instancia, un estudio del Estado del Arte sobre el uso de modelos LLM para describir figuras no rasterizadas[6], además de presentar el tablero Kanban con todas las actividades realizadas para lograr la implementación del servicio.

En los siguientes **Apartado 2.3** y **Apartado 2.4** se definirán los requisitos de implementación para el modelo "*Meta AI Llama 2*" y para el servicio generador de descripciones, a nivel de software y hardware, así como las consideraciones de diseño para su implementación, respectivamente. Más adelante, en el **Apartado 2.5** se presentará el proceso de implementación del servicio.

2.1 ESTADO DEL ARTE EN EL USO DE LLM PARA DESCRIBIR FIGURAS

Los avances e investigaciones sobre los modelos de lenguaje grande (LLM) han llevado el procesamiento del lenguaje natural a niveles superiores, permitiendo la comprensión de figuras a partir de su vectorización o codificación. Específicamente, en estas figuras los LLM han demostrado tener un buen rendimiento, en términos de comprensión y procesamiento. Sin embargo, al realizar un estudio del Estado del Arte, se evidencia que, a pesar de la creciente capacidad de los LLM para el procesamiento textual en tareas visuales, aún no existen soluciones significativas que aprovechen dicha capacidad para generar descripciones comprensibles de figuras codificadas o vectorizadas, facilitando su interpretación incluso para personas que no pueden ver la ilustración de forma explícita.

A continuación, se exponen los estudios más relevantes sobre el uso de modelos LLM en tareas visuales, estableciendo un contraste entre el procesamiento basado en texto con figuras vectorizadas o codificadas, y el procesamiento basado en visión por computadora con figuras rasterizadas.

El artículo "*Delving into LLMs' visual understanding ability using SVG to bridge image and text*" [19], investiga la capacidad de los modelos LLM de comprender figuras codificadas en formato *Scalable Vector Graphics* (SVG). Este estudio destaca las pruebas realizadas en diferentes modelos, como GPT4-brief, GPT-CoT, LLaVa y CNN+MLP, para tareas que implican transformaciones de forma, color y tamaño

[6] Una figura rasterizada es una imagen que se compone de una matriz de píxeles, donde cada píxel tiene un color específico y se depende de la resolución.

mediante representaciones SVG. La evaluación consiste en formular preguntas a los modelos a través de *prompts* relacionados con las tareas visuales, analizando así sus respuestas para evaluar sus capacidades de razonamiento y comprensión de los códigos SVG. Aunque el estudio identifica limitaciones en la comprensión y razonamiento de imágenes por parte de los LLM, también enfatiza el potencial de estos modelos para tareas que van más allá de aplicaciones basadas en procesamiento de texto, señalando oportunidades de mejora mediante técnicas de *Fine-Tuning*, *Prompt Engineering* y entrenamiento con *Data Sets* específicos.

Un estudio similar fue publicado por la Universidad de Wisconsin–Madison y la Universidad de Illinois *Urbana-Champaign* en el artículo "*Leveraging Large Language Models for Scalable Vector Graphics-Driven Image Understanding*" [20]. Este artículo propone el uso de modelos LLM para comprender y manipular ilustraciones en formato SVG, demostrando una reducción significativa en la brecha entre el procesamiento visual y textual de estos modelos. En la Figura 2.1 se puede visualizar el contraste que realizan entre una solución basada en métodos visuales (izquierda) y una solución basada en modelos de lenguaje (derecha), haciendo uso del código de la figura SVG.

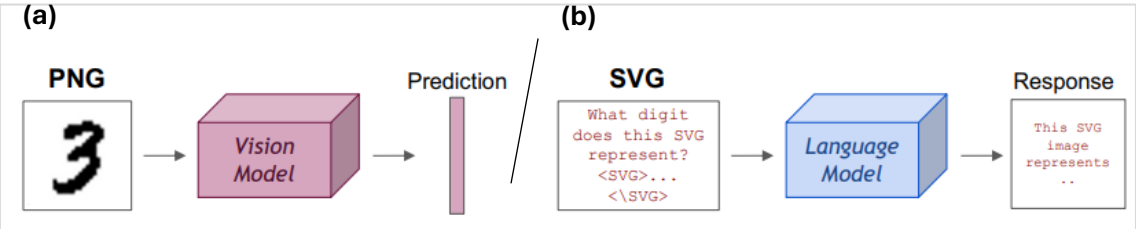


Figura 2.1 Contraste entre: (a) el procesamiento visual de figuras PNG y (b) el procesamiento textual de figuras SVG [20].

Para llevar a cabo este procesamiento textual de figuras, las imágenes rasterizadas se convierten a formato vectorial SVG [20]. Con estas imágenes SVG, se genera un *Dataset* y se entrena al LLM "*Vicuna*", además de aplicar técnicas de ajuste fino (*Fine-Tuning)* y *Prompt Engeenering*. De esta manera, se obtienen descripciones de una imagen SVG generadas por el procesamiento textual del LLM. Los resultados obtenidos demuestran la capacidad del LLM para realizar tareas visuales, como comprender claramente transformaciones simples de forma, color y tamaño al analizar el código SVG sin ninguna información a nivel de píxel, reforzando así la premisa estudiada en el artículo mencionado anteriormente [19].

Por otro lado, la iniciativa propuesta por la Asociación de Investigadores en Ciencia de Datos (ADaSci) en su artículo "*Image Captioning with Mistral 7B LLM: A Hands-on Guide*" [21] se alinea por el método del procesamiento visual, lo que llaman "Visión por Computadora", utilizando técnicas de entrenamiento visual como BLIP (*Bootstrapping Language-Image Pre-training* [21]) para generar descripciones de imágenes. Expone una guía de uso de este marco BLIP para configurarlo en el LLM "*Mistral*" de 7 billones de parámetros, y así obtener descripciones de cualquier imagen ingresada. De esta forma, el LLM puede procesar la imagen y entender visualmente su contenido, pero siempre bajo un esquema de funcionamiento visión-lenguaje.

Con un enfoque similar, la empresa Labeller ha desarrollado una solución homónima, detallada en su artículo "*LLM-Powered Image Caption Generation - Challenges, and Applications*" [22]. Este artículo presenta los avances en la generación de subtítulos de imágenes impulsados por modelos LLM, combinando la visión por computadora y el procesamiento de lenguaje natural, y explica cómo estos se integran en su solución.

En su estudio, descomponen el subtitulado de imágenes en la Extracción de Características de la Imagen utilizando redes neuronales convolucionales (CNN) o transformadores de visión (ViTs), la Codificación de Características en un formato adecuado para el modelo de lenguaje, y la Generación de Subtítulos usando LLM donde las características de la imagen codificadas se introducen en un modelo de lenguaje grande, que genera los subtítulos de la imagen a través de modelado de secuencias y generación de texto.

Implementar este esquema ofrece ventajas significativas para generar descripciones precisas de imágenes, con aplicaciones en áreas como la medicina, la seguridad, el comercio electrónico y la creación de contenido. Sin embargo, también depende en gran medida del procesamiento visual de imágenes rasterizadas, lo que implica un alto consumo de recursos computacionales y operativos.

Por este motivo, el presente Trabajo de Integración Curricular adopta la premisa del procesamiento textual de imágenes vectorizadas y codificadas con un LLM, basándose en los estudios que exponen los artículos "*Delving into LLMs' visual understanding ability using SVG to bridge image and text*" y "*Leveraging Large Language Models for Scalable Vector Graphics-Driven Image Understanding*". Los resultados obtenidos en estos estudios, como se ilustran en la Figura 2.2, han demostrado un rendimiento destacado de los LLM en la comprensión de tareas visuales sin el uso de mecanismos de visión por computadora, trabajando con imágenes no rasterizadas como el formato SVG, lo que reduce las limitaciones de alto consumo de recursos computacionales.

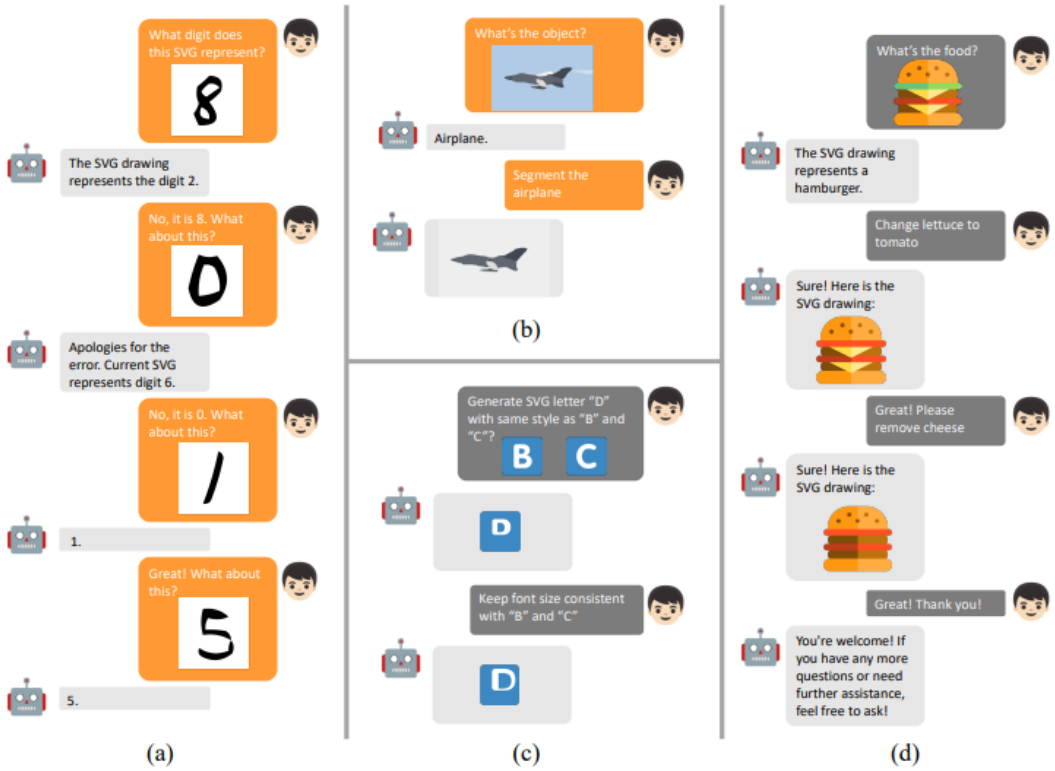


Figura 2.2 Resultados obtenidos del procesamiento de figuras SVG. (a) El entendimiento del LLM mediante retroalimentación humana logra mejorar. (b) El LLM puede distinguir y segmentar la figura SVG. (c)La generación de figuras similares logra mejorar. Y, (d) el procesamiento del LLM logra identificar y manipular partes específicas de una figura SVG [21].

Es así que, se opta por una solución que permita generar descripciones detalladas a partir del entendimiento del código de las figuras utilizando un LLM de código abierto, como lo es el modelo "*Meta Llama 2*", en el caso específico de artículos científicos realizados en formato LaTeX, que utiliza TikZ para generar figuras mediante comandos. Las figuras en formato TikZ y SVG son ampliamente utilizadas en estos artículos, ya que permiten crear representaciones gráficas de la temática expuesta en el escrito a partir de códigos genéricos, los cuales se convierten en ilustraciones al compilar el archivo LaTeX.

Además, la solución se apoya en las oportunidades de mejora que menciona el artículo "*Delving into LLMs' visual understanding ability using SVG to bridge image and text*" donde, para un mejor procesamiento textual de las figuras codificadas, se implementan técnicas de ingeniería de *prompts* y aprendizaje por refuerzo con retroalimentación humana (RLHF), como lo es RAG (*Retrieval-Augmented Generation*). Esto pretende

obtener descripciones detalladas no solo basadas en el código TikZ o SVG, sino también en el contexto de lo que el artículo menciona sobre estas figuras.

De esta manera, se implementa una solución que aprovecha las capacidades de los LLM en tareas visuales sin consumir una cantidad considerable de recursos computacionales y operativos, en contraste con las soluciones que proponen ADaSci y Labeller, las cuales dependen en gran medida de la extracción de características de imágenes con visión artificial. Las metodologías de ADaSci y Labeller, al utilizar técnicas como BLIP y redes neuronales convolucionales para la extracción y codificación de características visuales, requieren una infraestructura computacional robusta y un manejo extensivo de datos visuales rasterizados [23]. Por otro lado, la propuesta de este Trabajo de Integración Curricular se enfoca en la interpretación directa del código vectorial SVG y TikZ, lo que no solo reduce el costo computacional, sino que también simplifica el proceso de generación de descripciones comprensibles, haciéndola más accesible y eficiente, especialmente en contextos donde los recursos son limitados. Además, al proporcionar descripciones detalladas de figuras científicas, se busca también que personas con discapacidades visuales se vean beneficiadas en la comprensión en de artículos en el ámbito académico, contribuyendo en cierto grado a la inclusividad de los documentos científicos.

2.2 ESTABLECIMIENTO DEL TABLERO KANBAN

En esta sección se expone el tablero Kanban diseñado para este Trabajo de Integración Curricular. Se ha seleccionado la herramienta *Microsoft Planner* [24] [7] debido a su eficacia para programar actividades y gestionar proyectos.

El tablero Kanban, como se puede observar en la Figura 2.3, se divide en las fases de **Análisis**, **Diseño**, **Implementación** y **Pruebas**. Cada una de estas fases contiene una lista detallada de actividades específicas a realizar. El progreso del Trabajo de Integración Curricular se regirá por las fases estipuladas en el tablero. Una vez completadas las fases de Análisis y Diseño, se procederá con la implementación del servicio generador de descripciones, seguida de la fase final que incluye la obtención de resultados y la evaluación del servicio mediante pruebas de su funcionalidad.

[7] Microsoft Planner es una solución integrada de Office 365 con una interfaz visual estilo Kanban para crear tableros y listas de tareas, facilitando la gestión de proyectos.

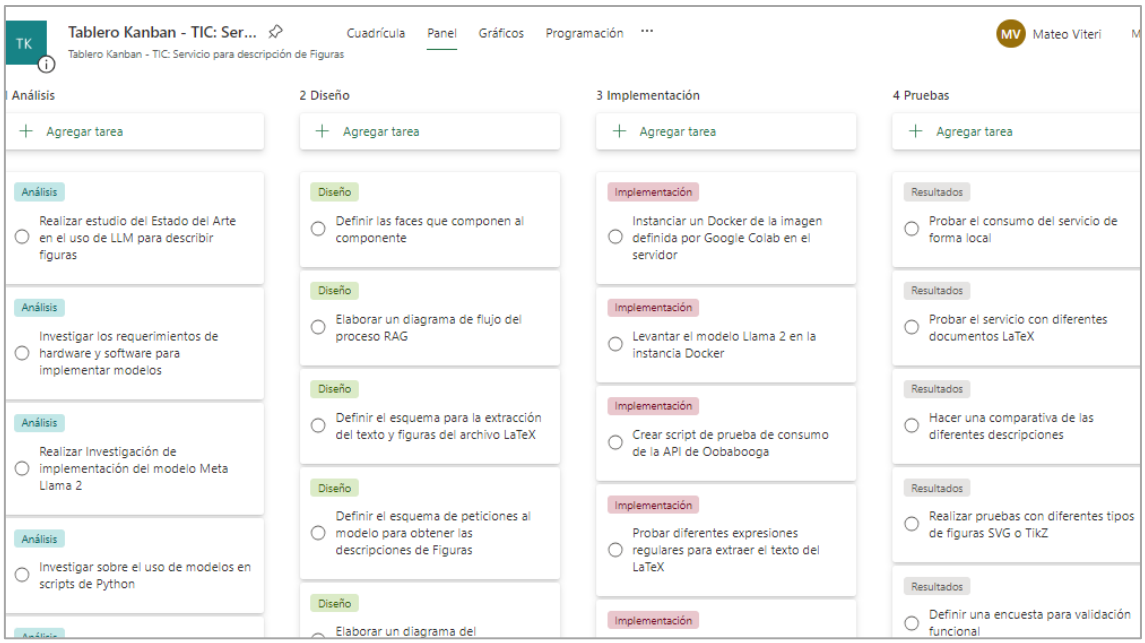


Figura 2.3 Tablero Kanban seccionado por las diferentes fases que componen al Trabajo de Integración Curricular.

2.3 REQUERIMIENTOS DE IMPLEMENTACIÓN DE OOBABOOGA Y META LLAMA 2

En este apartado se detallan los requerimientos mínimos para poder instalar y levantar el servicio de la web UI de Oobabooga y el LLM Meta Llama 2, tanto a nivel de hardware como de software.

La base del funcionamiento del modelo se centra en el uso de memoria GPU para su procesamiento y capacidad de inferencia, por lo que es crucial cubrir los requerimientos de hardware para que el modelo funcione correctamente y con el mayor rendimiento posible.

2.3.1 Consideraciones Hardware

Para implementar la web UI de Oobabooga, los requerimientos a nivel de Hardware no resultan ser tan exigentes. Esta herramienta puede funcionar sobre cualquier sistema operativo de los que se encuentran en el mercado. Los aspectos más importantes son la capacidad de almacenamiento y la memoria RAM del equipo donde se va a levantar la herramienta. Tomando de referencia requisitos mínimos para un correcto funcionamiento se recomienda lo siguiente [25]:

- **Procesador:** Intel Core i5 o AMD Ryzen 5.
- **Memoria RAM**: 8 GB de RAM DDR4.
- **Disco duro:** 256 GB de espacio libre.
- **Sistema operativo:** mayor a Windows 8 o distribuciones basadas en Debian como Ubuntu.

Si bien la documentación oficial no menciona algún impedimento en equipos con sistema operativo IOS se recomienda implementar la herramienta en sistemas conocidos y probados como los que se mencionaron anteriormente. La mayoría de las implementaciones se han realizado sobre un sistema operativo Linux ya que se tiene mayor control sobre su configuración, en contraste con un sistema operativo Windows.

Sin embargo, los requisitos de Oobabooga se deben ajustar con los requisitos para instalar y ejecutar los LLM. Los modelos LLM en su mayoría basan su procesamiento en la memoria virtual del equipo, es decir la memoria GPU, ya que involucran una gran cantidad de operaciones matemáticas, especialmente multiplicaciones y sumas de matrices, tareas que pueden realizarse en paralelo para reducir el tiempo de procesamiento y un funcionamiento más optimo. La memoria GPU contiene una gran cantidad de memoria virtual (VRAM) para facilitar el manejo de los enormes volúmenes de datos y parámetros involucrados en los modelos de lenguaje largo.

En el caso de los modelos "*Llama*" de Meta, equilibran adecuadamente el rendimiento con el consumo de recursos de hardware. El modelo Llama 2 tiene diferentes versiones en función de la cantidad de parámetros con los que fue entrenado, variando entre 7 billones y 70 billones de parámetros, siendo este último el de mayor rendimiento. Cada versión del modelo requiere una cantidad mínima de memoria VRAM y un determinado número de núcleos para su funcionamiento en modo "*model-parallelism*"[8]. Esto quiere decir que, ante la gran cantidad de parámetros, los modelos pueden ser divididos entre varios núcleos GPU. Especialmente el modelo de 70 billones para poder operar debidamente debe ser divido en mínimo 8 núcleos GPU.

En la siguiente Tabla 2 se puede visualizar la comparativa de los recursos usados entre las diferentes versiones del modelo Llama 2.

[8] "*model-parallelism*" es una técnica para dividir un LLM entre varios núcleos GPU.

Tabla 2.1. Comparativa de los recursos hardware necesarios para diferentes versiones de Llama 2 [26].

Modelo	RAM Requerida	GPU Requerida	Almacenamiento
Llama 2 (7B)	16-32 GB	1 GPU de gama media/alta	20-40 GB
Llama 2 (13B)	32-64 GB	2 o 4 GPUs de alta gama	40-80 GB
Llama 2 (70B)	Más de 128 GB	8 o más GPUs de alta gama	Más de 200 GB

Para este caso, se considera utilizar el modelo Llama 2 de 70 billones, por lo que los recursos mínimos de hardware para el funcionamiento del modelo, combinado con los requerimientos de la web UI de Oobabooga son los siguientes:

- **Memoria RAM**: 128 GB de RAM.
- **Memoria VRAM**: 40 GB por GPU.
- **Núcleos GPU**: Mínimo 4 GPUs de alta gama (como NVIDIA A100 o V100).
- **Almacenamiento:** 300 GB de espacio libre.

Aunque el modelo de 70 billones de parámetros requiere una gran cantidad de recursos computacionales, se le puede aplicar una técnica de cuantización[9]. Esta técnica consiste en reducir el tamaño de bits con el que se representa un parámetro, con la finalidad de reducir su tamaño en almacenamiento y su consumo de memoria virtual, para así aumentar su eficiencia. Si bien la precisión de los parámetros del modelo se verá ligeramente afectada, los beneficios son más sobresalientes, ya que se puede conseguir un modelo más compacto y rápido.

De esta manera, el modelo Llama 2 de 70 billones cuantizado puede funcionar con una VRAM mínima de 80 GB, en un solo núcleo GPU.

2.3.2 Consideraciones Software

En primer lugar, es esencial considerar los requerimientos más importantes a nivel de software, particularmente los relacionados con los controladores de la tarjeta gráfica que se utilizará en el sistema. Para las tarjetas gráficas de la marca NVIDIA, es fundamental asegurarse de que el controlador esté actualizado y sea compatible con la versión de

[9] Cuantizar un modelo quiere decir reducir su tamaño en memoria y aumentar la eficiencia de la inferencia, particularmente en hardware con recursos limitados.

CUDA [10] correspondiente. Esta verificación es crucial, ya que para la solución que se pretende implementar se requiere utilizar bibliotecas como *PyTorch* y *TensorFlow*, que permiten aprovechar los recursos de la GPU en tareas de *Machine Learning*. Estas bibliotecas funcionan de manera óptima con las versiones de CUDA 11.8 o CUDA 12.1, lo cual es indispensable para maximizar la capacidad de procesamiento de la memoria GPU [27]. Específicamente, la web UI de *Oobabooga* hace uso de *PyTorch* para el control de los modelos que vayan a ser implementados.

Con lo que respecta al uso de la web UI de *Oobabooga*, es necesario definir una arquitectura de implementación. La documentación oficial recomienda levantar esta web UI en entornos virtuales o contenedores, con la finalidad de separar los proyectos y tener mayor facilidad en la instalación de paquetes y dependencias, además de encapsular todos los recursos necesarios para ejecutar algún proyecto en particular. Para aplicaciones basadas en *Machine Learning* esto es crucial, incluso para poder exportar todo ese entorno de trabajo y reusarlo en otro equipo físico.

Si se sigue esta recomendación es necesario contar con una herramienta para usar entornos virtuales, como *virtualenv* o *conda* [11], o una herramienta para usar contenedores, como *Docker* [12]. Estos entornos virtualizados deben contar con la versión de Python 3.11 o superior, ya que es la versión compatible con la herramienta *Oobabooga*.

Así también, *Oobabooga* establece un listado de dependencias de Python como requerimientos, los cuales se ajustan al hardware del sistema donde se va a implementar. Estos requerimientos pueden ser optimizados para [11]:

- GPUs de la marca AMD, con soporte de instrucciones AVX2 [13].
- GPUs de la marca AMD, sin soporte de instrucciones AVX2.
- Computadoras Apple con procesadores Intel.
- Computadoras Apple con procesadores Apple Silicon.
- Solo uso de CPU, con soporte de instrucciones AVX2.

[10] CUDA (Compute Unified Device Architecture) es una plataforma de computación paralela desarrollada por NVIDIA que permite aprovechar el poder de procesamiento de la GPU.
[11] Virtualenv y Conda son herramientas comunes para usar entornos virtuales con Python.
[12] Docker es una plataforma de contenedores que facilita su creación, despliegue y gestión.
[13] AVX2 (Advanced Vector Extensions 2) son primitivas a nivel de CPU que permiten acelerar el rendimiento de ciertos tipos de cálculos matemáticos, como operaciones de punto flotante, enteros y multimedia.

- Solo uso de CPU, sin soporte de instrucciones AVX2.
- Computadores donde no se permite el uso de "*wheels*"[14] para la instalación de dependencias.

De esta forma, *Oobabooga* pretende ser compatible con las arquitecturas más comunes de los sistemas destinados a tareas de Aprendizaje Automático e Inteligencia Artificial.

Con estas consideraciones cubiertas, se puede descargar e implementar cualquier modelo LLM de código abierto para poder ser usado mediante la web UI de Oobabooga. El sitio web *Huggingface* [28] contiene una gran cantidad de muestras de modelos, entre ellos Llama 2, con todo lo necesario para ser descargados e implementados de forma local. Alternativamente, se puede solicitar acceso al modelo Llama 2 desde el sitio oficial de Meta [4].

2.4 DISEÑO

En este apartado, tal y como fue definido en el tablero Kanban de la Figura 2.2, se define el funcionamiento del servicio que permitirá generar las descripciones de las figuras SVG y TikZ que se encuentren dentro de un archivo LaTeX.

Primero, se definirá la arquitectura de implementación del servicio, seguida por la descripción de las diferentes fases que lo componen. Se detallará el funcionamiento de cada una de estas fases, abarcando desde el uso del modelo "*Llama 2*" con Python, el procesamiento del archivo LaTeX para extraer el texto relevante y los códigos de las figuras presentes, la implementación del proceso RAG (Retrieval-Augmented Generation), hasta el esquema de peticiones al modelo con técnicas de Prompt Engineering. Por último, se describirá el proceso de levantamiento del servicio generador de descripciones.

2.4.1 Arquitectura de implementación del servicio

El servicio generador de descripciones se monta sobre una instancia *Docker*, dentro de un servidor de sistema operativo Ubuntu, acorde a las recomendaciones dadas por Oobabooga como se mencionó en el apartado anterior. Este servidor, el cual únicamente es administrado mediante la terminal de comandos, cuenta una tarjeta gráfica NVIDA A100 de 80 GB de GPU, un controlador NVIDIA 550.54.15 y una versión CUDA 12.4, compatibles con versiones de *PyTorch* mayores a 2.1. Sobre este servidor se instancia

[14] Wheels es un formato de archivo binario para distribuir paquetes de Python.

un *Docker* con una imagen que cuenta con una versión de Python 3.11 y varias librerías de propósito general.

Dado que *Oobabooga* utiliza los puertos 7860 y 5000 para publicar su web UI y su API, respectivamente, es imperativo realizar un mapeo de puertos entre la instancia *Docker* y el servidor local. Esta configuración permitirá acceder a cada servicio desde fuera del contenedor.

Sin embargo, para sintetizar este proceso, se puede publicar la instancia *Docker* en un puerto específico, facilitando su uso como entorno de ejecución en *Google Colab* [15]. Así, un proyecto creado en *Google Colab* podrá acceder directamente a los servicios de Oobabooga mediante el mapeo de un puerto en la instancia Docker con un puerto del servidor local.

La Figura 2.4 ilustra el proceso de mapeo de puertos, representando una instancia *Docker* como un contenedor independiente (Contenedor 1) dentro del servidor (Host local). Este contenedor posee sus propios recursos y servicios que funcionan exclusivamente dentro de su entorno aislado. No obstante, para que estos servicios sean accesibles desde el exterior del contenedor, es necesario mapear los puertos internos del contenedor a los puertos del host. En otras palabras, se debe redirigir el tráfico que ingresa por un puerto del host local (por ejemplo, el puerto 8081) hacia un puerto expuesto del contenedor (por ejemplo, el puerto 8080), permitiendo así el acceso a un determinado servicio desde un agente externo al contenedor.

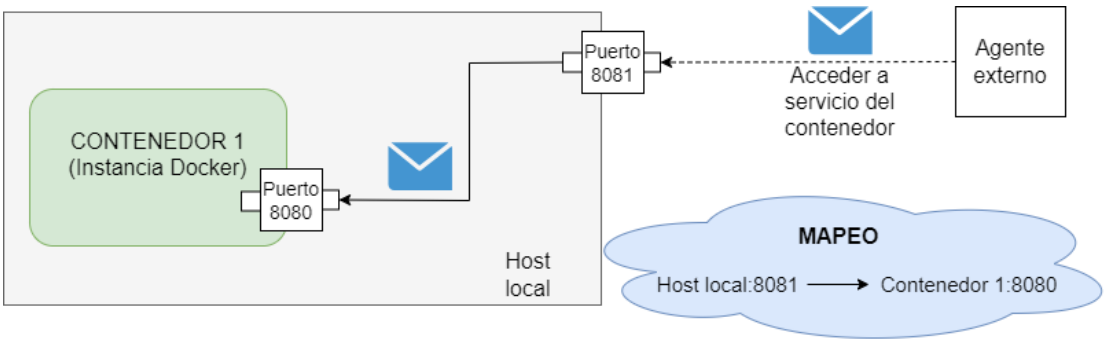


Figura 2.4 Ilustración del mapeo de puertos entre una instancia *Docker* con el host local.

[15] Usar la instancia Docker como entorno de ejecución en un proyecto de Google Colab facilita la implementación del servicio en portabilidad y escalabilidad, incluso para separar el servicio propiamente de la conexión directa con el servidor local.

Con este concepto claro, se define la imagen *Docker* para que la instancia se publique en su puerto 8080, y sea mapeado a un puerto libre del servidor local para poder ser accedido externamente.

Para terminar de configurar la imagen del *Docker*, se definen bibliotecas y dependencias adicionales que se requieren para poder implementar el servicio generador de descripciones, mismas que se resumen en la siguiente Tabla 2.2.

Tabla 2.2. Dependencias necesarias para el servicio generador de descripciones.

Recurso	Descripción
Sentence-Transformers [29]	Librería de Python para transformar oraciones en vectores de datos.
Faiss-cpu [30]	Librería de Python optimizada para el procesamiento de vectores densos y la búsqueda de similitudes.
Pylatexenc [31]	Librería de Python para la extracción y manipulación de texto en archivos LaTeX.
Requests [32]	Librería de Python que permite realizar peticiones HTTP.
JSON [33]	Librería de Python para trabajar con datos en formato JSON.
RE [34]	Librería de Python para crear y utilizar expresiones regulares.
Pandas [35]	Librería de Python que permite manipular y analizar conjuntos de datos estructurados y no estructurados.
Flask [36]	Librería de Python que se utiliza para crear aplicaciones o servicios web.
ZipFile [37]	Clase de Python que se permite crear, leer y extraer archivos comprimidos en formato ZIP.

De esta forma se asegura que, la instancia *Docker* contiene las bibliotecas necesarias para que tanto la web UI y la API de *Oobabooga* como el servicio para generar las descripciones funcione adecuadamente, con acceso total a la GPU del servidor local, y que todos los servicios sean accesibles desde agentes externos, como Google Colab.

Ahora bien, para conectar el proyecto de *Google Colab* con el servidor local para llevar a cabo lo anteriormente expuesto establece una conexión SSH (*Secure Shell*) de tipo "túnel". Esto implica establecer un túnel SSH local que redirige el tráfico desde un puerto del equipo donde se está ejecutando *Google Colab* hacia un puerto del servidor remoto,

el cual está mapeado con el puerto 8080 de la instancia Docker. De esta manera, se consigue un grado de independencia entre los servicios de Oobabooga y el proyecto donde se implementa el servicio que genera las descripciones de las figuras.

En la siguiente Figura 2.5 se puede apreciar el diagrama de implementación con todas las consideraciones mencionadas en este apartado.

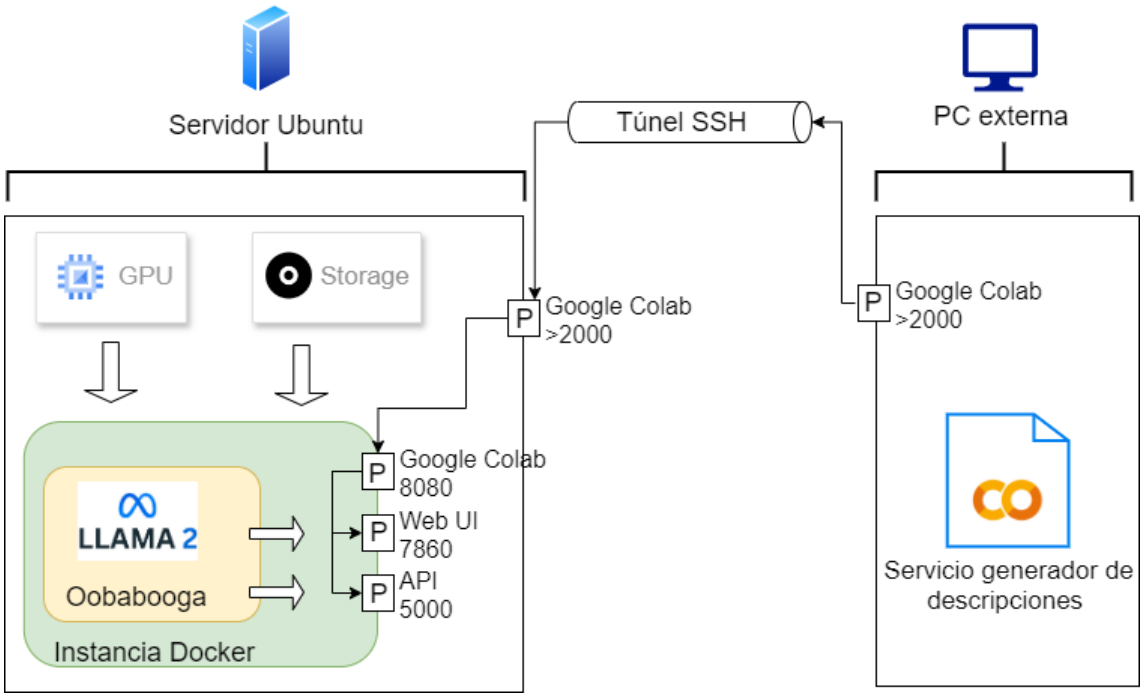


Figura 2.5 Diagrama que ilustra la arquitectura de implementación.

2.4.2 Definición de las fases del servicio generador de descripciones

El servicio generador de descripciones se compone de cuatro fases para su implementación: Procesamiento y extracción de texto y figuras, Implementación de la técnica RAG, Esquema de Peticiones y Levantamiento del Servicio.

La fase del **Procesamiento y Extracción de Texto y Figuras** del documento LaTeX inicia recibiendo un archivo comprimido con extensión .zip, que contiene el archivo principal con extensión .tex y otros archivos auxiliares que componen el documento LaTeX, como es común en artículos científicos. El archivo .tex se procesa para extraer exclusivamente el texto sin formato que conforma el documento, eliminando etiquetas y comentarios LaTeX, y de la misma manera para extraer los códigos de las figuras embebidas en el mismo.

Si el archivo .tex incluye referencias a figuras codificadas en archivos externos dentro del archivo .zip, se busca y extrae el contenido de dichos archivos de código de figuras. En caso de que se encuentren imágenes en formatos como .png, .jpg, .eps u otros, estas

serán omitidas en esta fase, cumpliendo con el alcance de este componente que se limita a figuras en formato SVG o TikZ.

El resultado de esta fase, como se muestra en la Figura 2.6, es una colección que contiene el texto sin formato del documento LaTeX y otra colección con el código de todas las figuras en formato SVG o TikZ presentes en el documento.

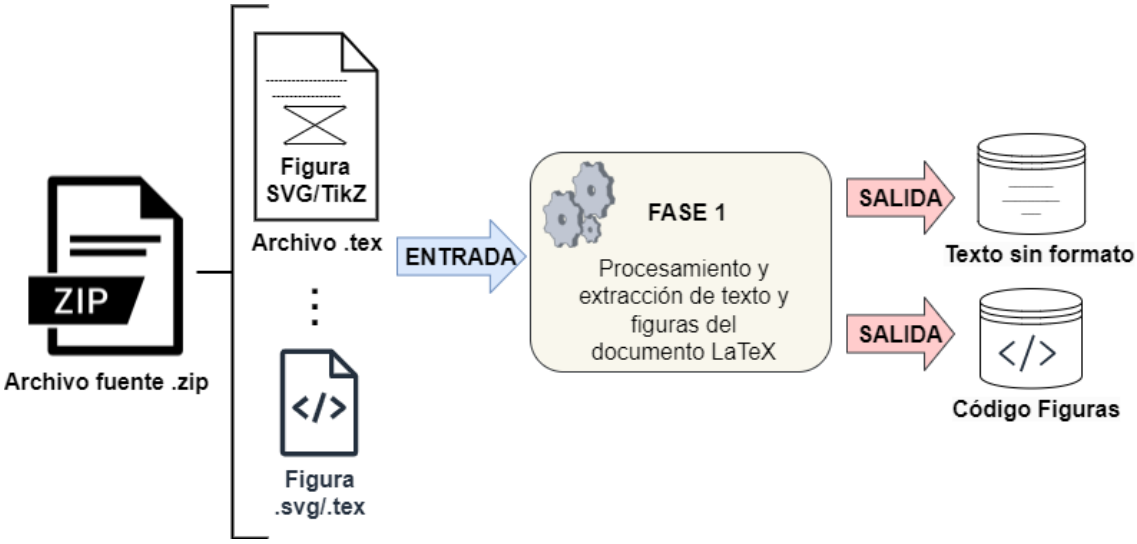


Figura 2.6 Fase del procesamiento del documento LaTeX, obteniendo como resultado el texto sin formato y los códigos de las figuras SVG o TikZ.

La siguiente fase se centra en la **Implementación de la Técnica RAG** (*Retrieval-Augmented Generation*) para la generación de texto mediante contextualización. Esta fase se divide en las siguientes sub-fases:

- **Fragmentación:** Con el texto sin formato resultante de la fase previa, se realiza una fragmentación dividiéndolo en oraciones compuestas.
- **Conversión:** Cada fragmento de texto es convertido a una representación vectorial, conocida como *Embeddings.*
- **Indexación:** Los *Embeddings* son almacenados en un vector de indexación, conocido como *Indexer*.
- **Contextualización:** En esta sub-fase, antes de enviar la petición al modelo Llama 2 para la generación de texto, dicha petición es sometida a un proceso de conversión para obtener su correspondiente *Embedding*. Luego, se realiza un proceso de correlación entre el *Embedding* de la petición y los *Embeddings* almacenados en el vector de indexación, basado en distancias vectoriales y similitudes semánticas, para identificar los *Embeddings* más relevantes.

- **Generación:** Los *Embeddings* más relevantes son reconvertidos a sus oraciones originales del texto fragmentado y se incorporan dentro de la ventana de contexto que se proporcionará al modelo "*Llama 2*". Esta ventana de contexto, que también incluye la petición original, es enviada al modelo "*Llama 2*", el cual generará la respuesta basada en la información relevante proporcionada.

Este proceso RAG se representa en el diagrama de flujo con notación BPMN[16] (*Business Process Modeling and Notation*) de la Figura 2.7, donde se logra que el modelo **"*Llama 2*" tenga un contexto del documento LaTeX**, para que así pueda generar descripciones de las figuras basadas explícitamente en la información de dicho documento.

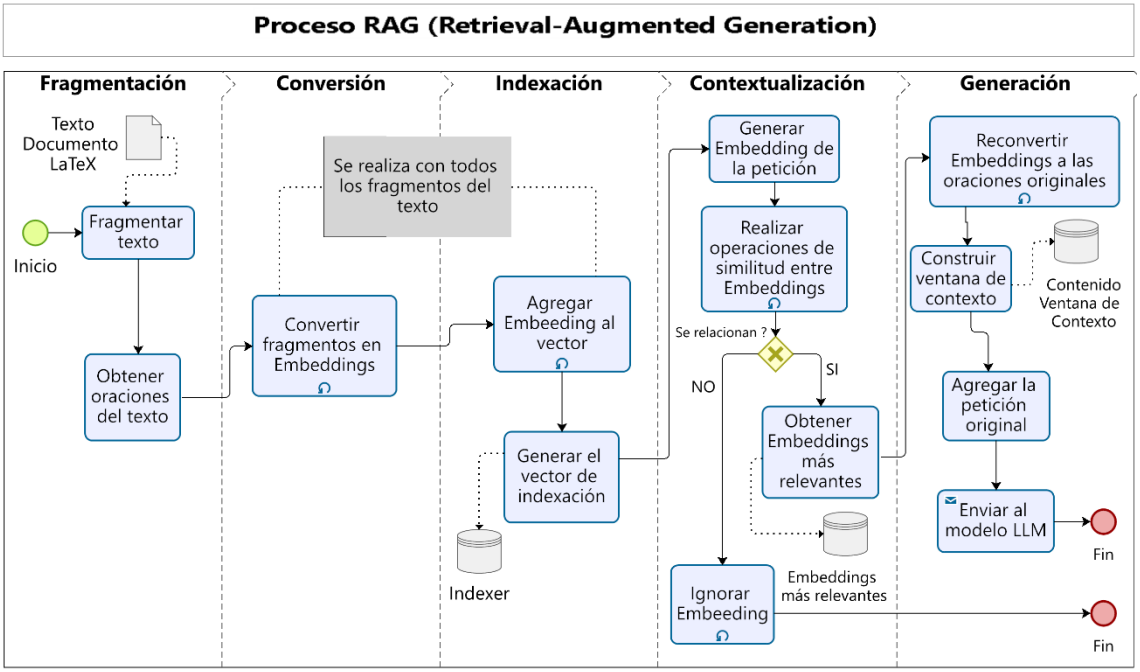


Figura 2.7 Diagrama de flujo con notación BPMN del proceso RAG a implementar.

Como siguiente fase se procede a definir el **Esquema de Peticiones** hacia el modelo "*Llama 2*" para la generación de descripciones. Este esquema se compone de tres peticiones en idioma inglés, cada una diseñada utilizando técnicas de *Prompt Engineering* para que el modelo considere detalles específicos de las figuras como las formas, colores y posiciones para poder describirlas, así como también solo responda con la descripción que se le solicita y nada más. Adicionalmente, para mejorar la

[16] La notación BPMN es una representación gráfica que permite modelar procesos de forma comprensible tanto para niveles técnicos como para ejecutivos.

precisión del modelo, se emplea la técnica "pensar dos veces"[17]. Esta técnica implica que se instruye explícitamente al modelo para que procese y reflexione sobre la petición antes de generar una respuesta, lo que resulta en descripciones más precisas.

Como se ilustra en la Figura 2.8, el esquema de peticiones se desarrolla de la siguiente manera:

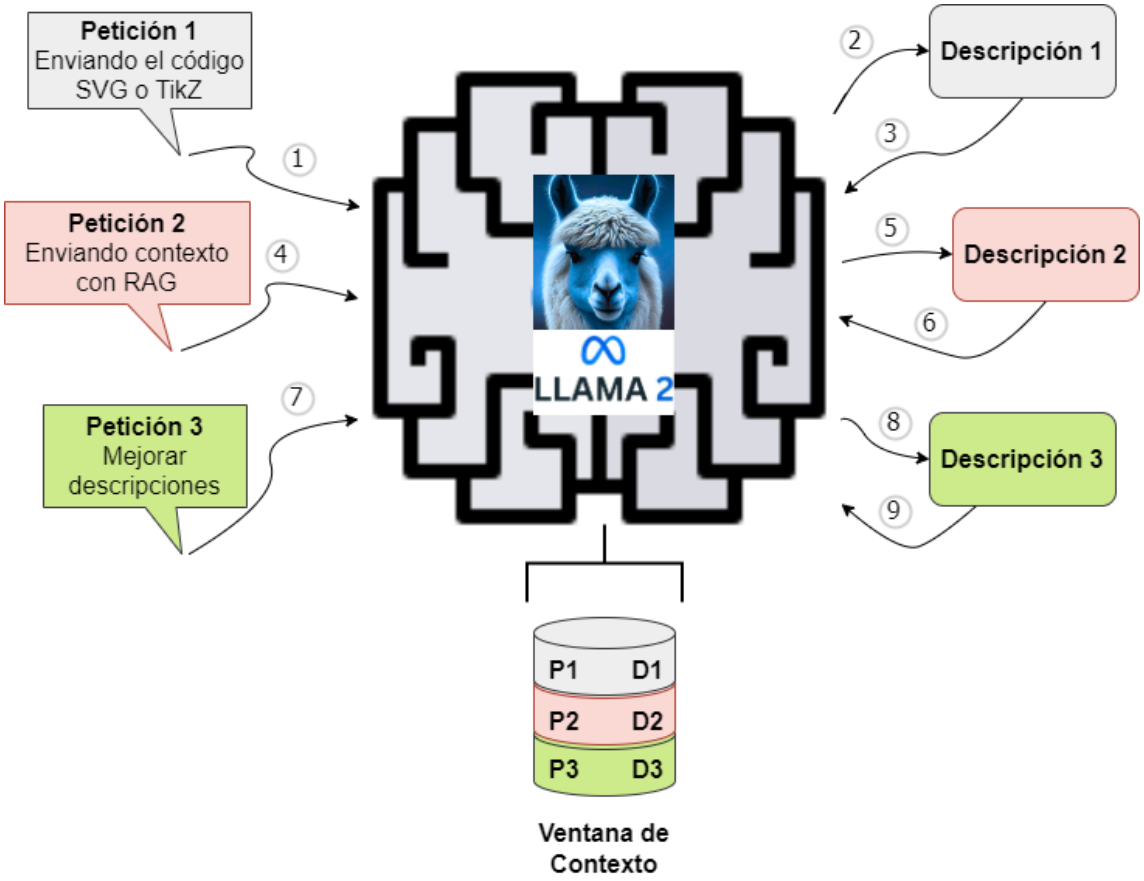


Figura 2.8 Esquema de peticiones al modelo "*Llama 2*" donde la (2)(3) primera descripción obtenida mediante (1) el envío del código como petición, la (5)(6) segunda descripción obtenida mediante (4) petición con RAG, y finalmente la (8)(9) tercera descripción obtenida mediante (7) petición de mejora. Es decir, la ventana de contexto almacena tanto las peticiones (P) como las respuestas (D).

1. **Primera Petición:** Se envía el código de la figura (SVG o TikZ) al modelo "*Llama 2*" sin contextualización previa. Esta solicitud inicial tiene como objetivo evaluar la capacidad del modelo para comprender y describir el código de la figura de manera básica. La petición es: "***I need you to describe a figure based on the following figure code: 'insert code'. Reflect and process this request twice***

[17] Pedirle a un LLM que "piense dos veces" antes de responder puede mejorar la inferencia para generar la respuesta.

to generate the description. Just respond with the description, nothing more."

2. **Segunda Petición:** Se busca mejorar la descripción generada previamente mediante la contextualización de la figura dentro del documento LaTeX. Para ello, se utiliza el proceso RAG descrito anteriormente, lo que permite al modelo tener una comprensión más profunda del contexto y mejorar la calidad de la descripción. La petición es: "***Previous figure appears in the document titled {title}, with name '{figure_name}', with label '{figure['label']}' and with caption {figure['caption']}. Improve previous description based only on the information provided in the context. Reflect and process this request twice to generate the description. Just respond with the description, nothing more.***"

3. **Tercera Petición:** La última petición se enfoca en refinar aún más la descripción anterior, prestando especial atención a los detalles. El objetivo es generar una descripción tan detallada y precisa que sea comprensible incluso para personas con discapacidad visual, ayudándolas a interpretar la figura. La petición es: "***From the last description you gave me, improve the description by considering all the details of the figure so that it can be understood even without seeing it, only through the description. Reflect and process this request twice to generate the description. Just respond with the description, nothing more.***"

Este esquema es posible ya que la ventana de contexto del modelo almacena tanto peticiones (P1, P2, P3) como respuestas (D1, D2, D3) bajo su determinado *template*[18], con lo cual el modelo puede entender e inferir la siguiente respuesta en base a las anteriores.

Finalmente, la última fase implica el **Levantamiento del Servicio** generador de descripciones. Este servicio define un único recurso accesible a través del punto final (o *endpoint**) "*/upload*", el cual acepta exclusivamente el método *POST*[19]. El propósito de este *endpoint* es recibir un archivo con extensión .zip para ejecutar el proceso descrito en las fases anteriores.

[18] El uso de templates sirve para definir roles dentro de la ventana de contexto, de tal forma que el modelo sepa diferenciar entre el texto ingresado por el usuario y el texto que está generando.
[19] El método POST permite a un cliente enviar datos al servidor web, siempre y cuando dicho servidor permita esta acción.

El servicio escucha en el puerto 8001, sin publicarse en Internet, siendo accesible únicamente de manera local. Por tanto, la URL para enviar los archivos .zip es ***http://127.0.0.1:8001/upload***. Es importante destacar que el puerto 8001 pertenece a la instancia Docker, no al host local, dado que dicha instancia se utiliza como entorno de ejecución.

Al recibir una petición *POST* con el archivo .zip, el servicio ejecutará todas las fases descritas previamente. Una vez finalizado el proceso, se generará una colección de figuras, cada una acompañada de su nombre, etiqueta, subtítulo y las tres descripciones generadas según el esquema de peticiones de la fase anterior. Esta colección será enviada en formato *JSON*[20], como se ilustra en la Figura 2.9, y el servicio permanecerá a la espera de nuevas peticiones *POST.*

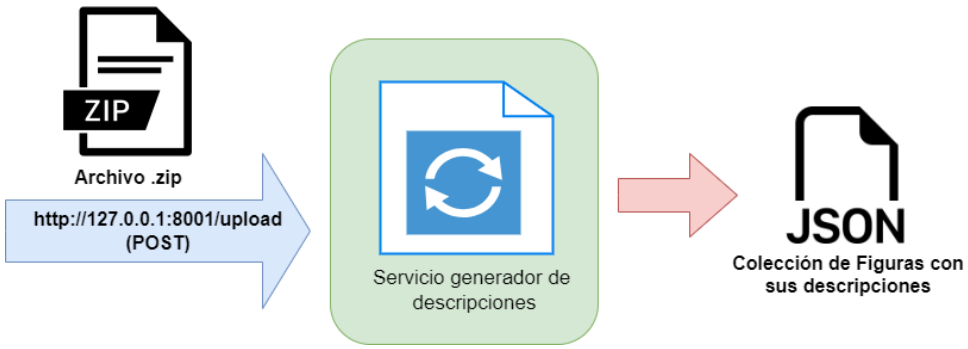


Figura 2.9 Diagrama de bloques del servicio generador de descripciones.

2.4.3 Diagrama de flujo del servicio generador de descripciones

El funcionamiento del servicio generador de descripciones se modela utilizando la notación BPMN, como se ilustra en la Figura 2.10. El proceso se inicia con la recepción de un archivo .zip mediante una petición POST entrante. A continuación, se extraen el texto sin formato y los códigos de las figuras para proceder con la preparación del proceso RAG, que se emplea en la segunda petición del esquema de peticiones definido en la sección anterior. El modelo "*Llama 2*" genera las descripciones en base a las tres peticiones, y estas se almacenan en una colección que se retorna como respuesta a la petición en formato JSON.

[20] JSON (JavaScript Object Notation) es el mejor formato para enviar y recibir datos, especialmente en el entorno web, bajo una estructura Llave-Valor.

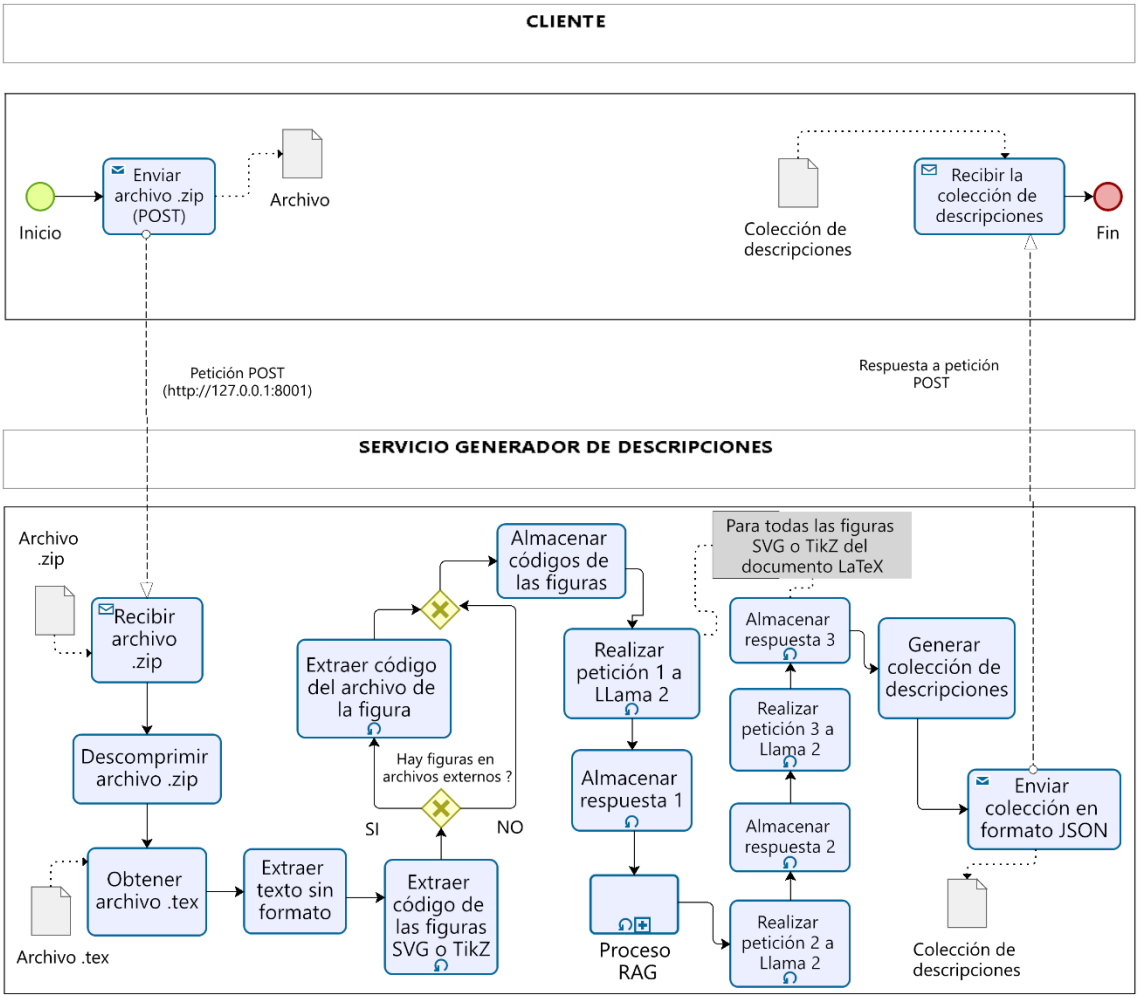


Figura 2.10 Diagrama del servicio generador de descripciones de figuras SVG o TikZ, presentes en documentos LaTeX.

2.5 IMPLEMENTACIÓN

Culminadas todas las tareas de la fase de Diseño, se procede con la siguiente fase establecida en el tablero Kanban de la Figura 2.2, la Implementación del servicio que propone este Trabajo de Integración Curricular.

En el **Apartado 2.5.1** se detalla la implementación del modelo Llama 2 mediante la solución de *Oobabooga* en la instancia *Docker* del servidor, mientras que en los **Apartados 2.5.2** y **2.5.3** siguientes se detalla la codificación y el levantamiento del servicio generador de descripciones, en un proyecto de Google Colab.

2.5.1 Levantamiento del servicio para uso del modelo Llama 2

Para levantar el servicio de *Oobabooga* con el modelo Llama 2 es necesario empezar con la definición de la imagen *Docker* con las especificaciones estipuladas en la arquitectura de implementación. Esta imagen se configura con los parámetros de la

imagen *Docker* de *Colab* publicada por Google [38], permitiendo que se pueda usar una instancia de esta imagen Docker como entorno de ejecución en *Google Colab*.

Por tanto, cuando se cree una instancia *Docker*, se genera un token único que se debe ingresar en un proyecto de Google Colab, en la sección Conectar, Conectar a un entorno de ejecución local, en el campo URL.

Sin embargo, previo a crear una instancia *Docker* con esta imagen, se le incluyen las consideraciones definidas en la arquitectura de implementación, con total acceso a la GPU del servidor, publicando la instancia en el puerto 8080, con una versión de *Python* 3.11, con todas las dependencias y controladores para gestionar la tarjeta gráfica NVIDIA A100.

Una vez configurada la imagen Docker se procede con la creación de la instancia mediante el comando que se presenta en el Código 2.5.1. Dado que se quiere dejar levantada esta instancia de forma persistente, se hace uso de la herramienta *Screen*[21] para que la instancia Docker se mantenga activa en segundo plano.

```
docker run --gpus=all -P --name="Llama_V2_MSVH" us-docker.pkg.dev/colab-images/public/runtime
```

Código 2.5.1 Comando para crear y correr la instancia *Docker*.

Los parámetros del Código 2.5.1 son:

- ***--gpus=all:*** Indica que la instancia tiene acceso a todas las GPUs disponibles del servidor.
- ***-P:*** Indica que se hará un mapeo del puerto 8080, que es el que se configuró para publicar la instancia, con un puerto libreo del servidor, generalmente mayor a 2000.
- ***--name:*** Hace referencia al nombre de la instancia *Docker*, en este caso, "*Llama_V2_MSVH*".
- ***us-docker.pkg.dev/colab-images/public/runtime:*** Indica la imagen *Docker* con la que se va a generar la instancia.

A este comando se le agrega un parámetro "***-v /media:/media***", lo que significa que se comparte la carpeta */media* entre el servidor y la instancia *Docker*. Esto es debido a que

[21] Screen permite gestionar múltiples sesiones de terminal dentro de una sola ventana de terminal.

en esta carpeta se va a realizar la instalación de la herramienta *Oobabooga*, como una medida de respaldo en caso de que la instancia *Docker* sufra algún fallo.

Una vez creada la instancia, se procede a ejecutarla con el comando mostrado en el Código 2.5.2, con los siguientes parámetros*:*

- ***-it:*** permite interactuar con la sesión de la terminal dentro del contenedor de forma interactiva.
- ***Llama_V2_MSVH:*** Hace referencia al nombre de la instancia *Docker*.
- ***bash:*** inicia una sesión interactiva de terminal tipo *Bash*[22]*.*

```
docker exec -it Llama_V2_MSVH bash
```

Código 2.5.2 Comando para ejecutar una terminal *bash* en la instancia *Docker*.

Ejecutada la terminal *Bash* dentro de la instancia, en el directorio */media,* se clona el repositorio *GitHub*[23] de *Oobabooga* [11]*,* con el comando mostrado en el Código 2.5.3, el cual descarga e instala la herramienta.

```
git clone https://github.com/oobabooga/text-generation-webui.git
```

Código 2.5.3 Comando para clonar el repositorio GitHub de *Oobabooga*.

Después se procede con la instalación de los requerimientos de la herramienta con el comando mostrado en el Código 2.5.4.

```
pip install -r requirements.txt
```

Código 2.5.4 Comando para crear y correr la instancia *Docker*.

Cuando la instalación se termine, se procede a descargar una muestra del modelo "*Llama 2*". En este caso, se descarga el modelo "*Llama 2 Chat"* de 70 billones de parámetros desde un repositorio de *Huggingface* [28]*.*

Con todos los requerimientos instalados, se procede a ejecutar el servicio de Oobabooga, tanto su web UI como su API, con el modelo "*Llama 2*" cargado, mediante el comando que se presenta en el Código 2.5.5.

[22] Bash (Bourne Again SHell) es un tipo de intérprete de comandos basado en el sistema operativo Unix.
[23] GitHub es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git.

```
python server.py --listen --model TheBloke_Llama-2-70B-Chat-fp16 --load-in-4bit --api
```

Código 2.5.5 Comando para ejecutar el servicio de Oobabooga con el modelo "*Llama 2"* levantado.

Los parámetros del Código 2.5.5 son:

- ***--listen:*** Indica que el servicio va a escuchar conexiones entrantes.
- ***--model TheBloke_Llama-2-70B-Chat-fp16:*** Indica que se debe cargar el modelo "*Llama 2*" de 70 billones de parámetros, previamente descargado.
- ***--load-in-4bit:*** Indica que el modelo se va a cuantizar a una precisión de 4 bits para reducir su tamaño y mejorar la eficiencia computacional.
- ***--api:*** Hace referencia a la activación y publicación de la API para su consumo desde programas externos.

De esta forma, el modelo "*Llama 2*" se encuentra levantado. Todo el proceso detallado de la instalación y el levantamiento del modelo "*Llama 2*" se puede encontrar en el apartado **Anexo I.**

2.5.2 Codificación del servicio generador de descripciones

Establecida la instancia *Docker* como entorno de ejecución en un proyecto de *Google Colab*, se procede con la codificación del funcionamiento del servicio generador de descripciones.

Se empieza por la fase de procesamiento y extracción del texto del documento LaTeX, como se estipuló en el **Apartado 2.4.2.** El objetivo principal es analizar el contenido de los documentos, eliminar líneas comentadas y vacías, y extraer tanto el texto sin formato como el código de las figuras.

Se hace uso de la librería *pylatexenc* para poder obtener el texto sin formato del archivo .tex, mediante la función *latex_to_text()*. Sin embargo, el resultado de aplicar dicha función presenta tres inconvenientes. El primero es que existen documentos LaTeX con líneas de comentarios en su contenido, los cuales la librería los toma como texto y por ende genera ruido en el resultado final. El segundo conflicto es que existen etiquetas que no soporta la librería y puede dar un error al obtener el texto. Y el tercero es que, al extraer el texto, el resultado se presenta con caracteres vacíos en donde estaban etiquetas LaTeX y se genera mucho ruido que dificulta su procesamiento.

El primer inconveniente se controla mediante la función *remove_commented_lines()* presentada en el Código 2.5.6, la cual recibe como argumento de entrada el documento LaTeX para crear una lista que contiene solo las líneas que no están comentadas, es decir, las líneas que no comiencen con el carácter '%', y finalmente reúne estas líneas para retornar el texto sin comentarios.

```
# Función para eliminar líneas comentadas
def remove_commented_lines(content):
    lines = content.split('\n')
    uncommented_lines = [line for line in lines if not
line.strip().startswith('%')]
    return '\n'.join(uncommented_lines)
```

Código 2.5.6 Función para eliminar líneas comentadas.

Por su parte, para solventar los otros inconvenientes se hace uso de expresiones regulares. Con la librería *re* se pule el procesamiento del documento LaTeX eliminando todas las etiquetas que la librería *pylatexenc* no puede soportar, macros de referencia, códigos especiales de alguna librería para LaTeX, entre otras. Así también, tal y como se puede evidenciar en el Código 2.5.7, se define la función *remove_empty_lines_and_align_text()* la cual recibe el texto obtenido con la función *latex_to_text()* y lo reestructura quitando saltos de línea y espacios en blanco con expresiones regulares, y lo organiza de una manera que se vuelva un texto conciso.

```
# Función para alinear el texto y eliminar saltos de línea vacíos
def remove_empty_lines_and_align_text(text):
    # Eliminar los saltos de línea donde no haya texto
    text = re.sub(r'\n\s*\n', '\n', text)
    # Eliminar tabulaciones que tengan las líneas de texto al inicio
    lines = text.splitlines()
    aligned_lines = []
    for line in lines:
        if line:
            aligned_lines.append(line.lstrip())
    return "\n".join(aligned_lines)
```

Código 2.5.7 Función para eliminar espacios en blanco y reestructurar el texto.

Continuando con el procesamiento del documento, se usan expresiones regulares para encontrar y extraer las figuras embebidas, para extraer los parámetros *label* y *caption* de las figuras, ya sea de las que se encuentren embebidas o las que se encuentren en archivos externos dentro del archivo .zip, para extraer el texto del documento sin las figuras, y para reemplazar las referencias de las figuras en el texto por su nombre, su *label* y su *caption*, ya que será de gran importancia para el proceso RAG que se describe

más adelante. En la Tabla 2.3 se detallan las expresiones regulares usadas y su propósito.

Tabla 2.3. Expresiones regulares utilizadas y su respectivo propósito.

Expresión regular	Propósito
r'\\href\{[^\}]*\}\{[^\}]*\}'	Eliminar macros \href del documento LaTeX.
r'\\begin\{figure\}(.*?)\\end\{figure\}'	Extraer todas las figuras presentes en el documento LaTeX.
r'\\includesvg(?:\[.*?\])?\{([^}]*)\}'	Obtener el nombre del archivo externo de la figura SVG importada en el documento LaTeX.
r'\\caption\{([^}]*)\}'	Obtener el *caption* de la figura del documento LaTeX.
r'\\label\{([\w:\s*-]+)\}'	Obtener el *label* de la figura del documento LaTeX.
r'Figure(?:\s*[:~])?\s*\\ref\{(.+?)\}'	Reemplazar las referencias a figuras en el documento LaTeX.
r'\\title(?:\[[^\]]*\])?\{([^\}]*)\}'	Obtener el título del documento LaTeX.
r'\\title(?:\[[^\]]*\])?\{([^\}]*)\}.*? \\begin\{document\}(.*?)\\end\{document\}'	Obtener el contenido entre las etiquetas *\title* y *\begin{document}*, y entre las etiquetas *\begin{document}* y *\end{document}*.

Todo esto es juntado dentro de la función *extract_figures_and_text()* que recibe como argumento de entrada la ruta del archivo .tex y ejecuta todo el procesamiento mencionado anteriormente, retornando al concluir una colección de los datos de las figuras (*figure_data*), que contiene el número de figura, el nombre, el formato y el código respectivo, y una variable que contiene el texto del documento LaTeX (text). El código de esta función se puede observar en el apartado **Anexo II** que contiene el programa completo.

Continuando con la siguiente fase de la implementación, se implementa la lógica del proceso RAG. Se define la función *fragment_and_get_embeddings()* que junta las sub-fases de Fragmentación y Conversión, tal y como se evidencia en el Código 2.5.8. La

función recibe como argumento de entrada el texto resultante de la función *extract_figures_and_text(),* y se fragmenta en oraciones compuestas con la librería *nltk*[24]. Después, se obtienen los *Embeddings* de todas las oraciones almacenadas en una lista, y se retorna esta lista de *Embeddings* junto con la lista de oraciones.

```
1.  # Función para fragmentar y obtener los embeddings del archivo
2.  def fragment_and_get_embeddings(text):
3.    # Se divide el texto en oraciones
4.    sentences = nltk.sent_tokenize(text)
5.    # Se generan los embeddings de las oraciones
6.    sentence_embeddings = embeddingModel.encode(sentences)
7.    return sentence_embeddings, sentences
```

Código 2.5.8 Función que representa las sub-fases de Fragmentación y Conversión del proceso RAG.

La siguiente sub-fase de Indexación se implementa en la función *indexing*() que, como se presenta en el Código 2.5.9, recibe como argumento de entrada la lista con los *Embeddings* para almacenarlos en un vector de indexación (*Indexer*) de tipo *FlatL2*[25] con la función *IndexFlatL2()* de la librería *Faiss*, el cual es retornado al culminar su ejecución.

```
1.  def indexing(embeddings):
2.    # Crear índice de tpo FlatL2 y almacenar embeddings
3.    indexer =
4.  faiss.IndexFlatL2(embeddingModel.get_sentence_embedding_dimension())
5.    indexer.add(embeddings)
6.    return indexer
```

Código 2.5.9 Función que representa la sub-fase de Indexación del proceso RAG.

Previo a culminar el proceso RAG, se codifica el uso del modelo Llama 2 de 70 billones de parámetros mediante el consumo de la API de Oobabooga. En el Código 2.5.10 se presenta la función *generate_text_adv()* que permite enviar las peticiones y obtener respuestas que genere el modelo Llama 2. En esta función se llama a la variable global *conversation_history* que representa la ventana de contexto del modelo, y se configura el *template* para las peticiones y respuestas. En este template se incluye la petición (*prompt*) que la función recibe como argumento de entrada, en el rol de usuario (*user*).

[24] NTLK (Natural Language Toolkit) es una biblioteca de Python que proporciona herramientas y recursos para el procesamiento de texto.
[25] Un vector de tipo FlatL2 crea un indexer "plano" donde los Embeddings se almacenan en su totalidad y se utiliza la distancia L2 (distancia euclidiana) para medir la similitud entre ellos.

Con la ventana de contexto armada, se envía en el cuerpo de una petición POST mediante la librería *requests*, hacia la URL "*http://localhost:5000/v1/chat/completions*". Si la comunicación con Llama 2 es exitosa se obtendrá la respuesta, que será agregada en la ventana de contexto, retornando dicha respuesta al culminar con la ejecución de la función, caso contrario esta retornará un mensaje de error.

```
1.  # Función consumir el modelo y enviar la figura códificada
2.  def generate_text_adv(prompt):
3.      global conversation_history
4.      url = "http://localhost:5000/v1/chat/completions"
5.      headers = {"Content-Type": "application/json"}
6.      # Agregar el prompt actual a la lista de historial
7.      conversation_history.append({"role": "user", "content": prompt})
8.      payload = {
9.          "messages": conversation_history,
10.         "mode": "instruct",
11.         "instruction_template": "Alpaca"
12.     }
13.     response = requests.post(url, json=payload, headers=headers)
14.     if response.status_code == 200:
15.         result = response.json()["choices"][0]["message"]["content"]
16.         # Agregar la respuesta del modelo al historial de la
17.         conversación
18.         conversation_history.append({"role": "assistant", "content":
19.         result})
20.         return result
21.     else:
22.         raise Exception(f"Error: {response.status_code} -
23.         {response.text}")
```

Código 2.5.10 Función para enviar las peticiones al modelo Llama 2 mediante la API de Oobabooga.

La función del Código 2.5.10 se modifica para agregar las sub-fases de Contextualización y Generación del proceso RAG, obteniendo así la nueva función *generate_text_rag()*. Esta función recibe como argumento de entrada la petición (*prompt*), el *Indexer* y la lista con las oraciones del texto. Las modificaciones más relevantes que presenta esta función se muestran en el Código 2.5.11, donde se obtienen el *Embedding* de la petición, se obtienen los *Embeddings* del *Indexer* que más concuerden con el *Embedding* de la petición, y así obtener las oraciones más relevantes.

Estas oraciones son agregadas en la ventana de contexto como contenido del sistema (*system*), es decir, será el contexto que el modelo "*Llama 2*" utilizará para generar la respuesta ante la petición, misma que también se agrega a la ventana de contexto, pero como contenido del usuario (*user*).

```
1.  # Codificar el prompt
2.  prompt_embedding = embeddingModel.encode([prompt])[0]
3.  # Buscar las oraciones más relevantes en el índice
4.  distances, indices = index.search(prompt_embedding.reshape(1, -1), 5)
5.  relevant_sentences = [sentences[idx] for idx in indices[0]]
6.  # Agregar las oraciones relevantes al historial de la conversación
7.  conversation_history.append({"role": "system", "content":
8.  "\n".join(relevant_sentences)})
9.  # Agregar el prompt actual al historial de la conversación
10. conversation_history.append({"role": "user", "content": prompt})
```

Código 2.5.11 Implementación de las sub-fases Contextualización y Generación del proceso RAG.

Finalmente, todas las funciones son llamadas en la función principal llamada *figures_descriptor()*. Esta función, presentada en el Código 2.7, recibe como parámetro de entrada la ruta del archivo .tex y llama a las funciones que se describieron anteriormente. Con la colección de figuras extraídas con la función *extract_figures_and_text()* se genera un bucle que recorra cada figura contenida en esta colección y se ejecuta el esquema de peticiones que se definió en el **Apartado 2.4.2**, enviando el código de la figura en la primera petición, aplicando la generación con RAG en la segunda petición y mejorando la calidad de la descripción en la tercera petición. Todas estas descripciones generadas por el modelo "*Llama 2*" son añadidas como atributos de cada figura en la colección, y esta colección resultante será lo que la función retorne al finalizar su ejecución.

Como se puede observar en el Código 2.5.12, en la línea 6 es donde se inicializa la variable global *conversation_history* que representa la ventana de contexto del modelo "*Llama 2*", y es reinicializada en cada iteración del bucle que recorre las figuras de la colección.

```
1.  # Función para la generación de descirpciones
2.  def figures_descriptor(file_path):
3.    global conversation_history
4.    global code_Figures
5.    # Se inicializa el historial de la conversación
6.    conversation_history = []
7.    # Se llama a la función que extrae el texto y las figuras
```

```
    figure_data, text = extract_figures_and_text(file_path)
    # Se obtienen los Embeddings de las oraciones
    embeddings, sentences = fragment_and_get_embeddings(text)
    # Se agrega los Embeddings al Indexer
    indexer = indexing(embeddings)
    # Se genera un bucle que recorra la colección de figuras
    for figure in figures_colec:
      if 'filename' in figure:
        figure_name = figure['filename']
      else:
        figure_name = figure['Figure']

      # Envió y respuesta de la petición 1
      prompt_basic = f"I need you to describe a figure based on the
following figure code: {figure['code']}. Reflect and process this
request twice to generate the description. Just respond with the
description, nothing more."
      description_base = generate_text_adv(prompt_basic)
      # Se agrega la descripción base a la Figura
      figure['description_base'] = description_base

      # Envió y respuesta de la petición 2
      prompt_RAG = f"Previous figure appears in the document titled
{title}, with name '{figure_name}', with label '{figure['label']}' and
with caption {figure['caption']}. Describe it as best as possible based
only on the information given you can identify. Just respond with the
description. nothing more."
      description_RAG = generate_text_rag(prompt_RAG, index, sentences)
      # Se agrega la descripción base a la Figura
      figure['description_RAG'] = description_rag

      # Envió y respuesta de la petición 3
      prompt_final = f"From the last description you gave me, try to
describe the equations, if they exist, and all the details of the
figure in a way that the figure can be understood without seeing it.
Just respond with the description. nothing more."
      description_final = generate_text_adv(prompt_final)
      # Se agrega la descripción base a la Figura
      figure['description_final'] = description_final

      # Se reinicializa el historial de la conversación
      conversation_history = []
    return figures_colec
```

Código 2.5.12 Función principal que realiza todo el proceso de generación de descripciones.

2.5.3 Levantamiento del servicio generador de descripciones

Como paso final, se implementa el servicio generador de descripciones, presentado en el Código 2.5.13, utilizando el framework *Flask*[26]. *Se* instancia un objeto llamado "*app"* de la clase homónima que permite crear un servicio web. El argumento de entrada "*__name__*" es una variable especial en Python que hace referencia al nombre del módulo actual.

Se define una ruta */upload* que acepta solicitudes HTTP de tipo POST. Cuando se reciba una petición POST se ejecutará la función *upload_file()*, la cual obtiene el archivo .zip que el cliente está enviando. Una vez recibido, se crea un directorio con el nombre del archivo.zip y aquí es descomprimido, para así obtener el archivo .tex. Cuando se encuentre el archivo .tex, se obtiene la ruta del archivo en la variable *tex_file_path*, misma que es el argumento de entrada de la función principal *figures_descriptor()*.

El resultado de la función *figures_descriptor()* es la colección con los datos de las figuras y sus descripciones, la cual será reconvertida en formato JSON mediante la función *jsonify()* y enviada como respuesta a la petición POST entrante, dando por finalizada la ejecución del servicio generador de descripciones.

Como se aprecia en el Código 2.5.13, el servicio se levanta al ejecutarse las líneas 49 y 50, escuchando peticiones en todas las interfaces de red (0.0.0.0), en el puerto 8001 de la instancia Docker.

```
1.  # Se instancia un objeto Flask para el servicio
2.  app = Flask(__name__)
3.
4.  @app.route('/upload', methods=['POST', 'GET'])
5.  def upload_file():
6.      file = request.files['file']
7.      if file.filename.endswith('.zip'):
8.        # Extraer el nombre base del archivo .zip sin la extensión
9.        zip_name = os.path.splitext(file.filename)[0]
10.       # Crear directorio con el nombre del archivo .zip
11.       zip_path = os.path.join('/media/Llama-MSVH/Archivos_TEX',
12. zip_name)
13.       os.makedirs(zip_path, exist_ok=True)
14.
15.      # Guardar el archivo zip en el directorio creado
16.       file_path = os.path.join(zip_path, file.filename)
17.       file.save(file_path)
```

[26] Flask es la librería de Python que se utiliza para crear aplicaciones o servicios web básicos.

```
18.         # Se imprime mensaje en consola cuando se reciba el archivo .zip
19          print(f"Archivo ZIP recibido: {file_path}")
20.
21.         # Se descomprime el .zip para obtener el archivo .tex
22.         with zipfile.ZipFile(file_path, 'r') as zip_ref:
23.           zip_ref.extractall(zip_path)
24.
25.         # Buscar el archivo .tex en el directorio extraído
26.           tex_file_path = None
27.           for root, dirs, files in os.walk(zip_path):
28.               for file in files:
29.                   if file.endswith('.tex'):
30.                       tex_file_path = os.path.join(root, file)
31.                       break
32.               if tex_file_path:
33.                   break
34.           if not tex_file_path:
35.               return "No .tex file found in the ZIP archive", 400
36.
37.           # Se llama a la función generadora de descripciones
38.           figures_desc_file = figures_descriptor(tex_file_path)
39.
40.           # Se imprime un mensaje para indicar que se está enviando la
41. información de respuesta
42.           print("Enviando la información...")
43.
44.           # Se convierte la colección de figuras en formato JSON y se
45. envía como respuesta
46.           return jsonify(response_data)
47.       else:
48.           return 'Formato de archivo no válido', 400
49.
50. # Programa principal que levanta el servicio
51. if __name__ == '__main__':
52.       app.run(host='0.0.0.0', port=8001)
```

Código 2.5.13 Servicio generador de descripciones de figuras implementado.

Todo el código completo del servicio generador de descripciones de figuras SVG o TikZ en documentos LaTeX, así como sus funciones y secciones claramente especificadas, se encuentra en el apartado **Anexo II.**

3 RESULTADOS Y DISCUSIÓN

En el presente Capítulo 3 se exponen los resultados obtenidos tras la implementación completa del servicio generador de descripciones.

El **Apartado 3.1** abarca un análisis exhaustivo de los resultados de las descripciones obtenidas, una comparación entre las descripciones generadas para cada petición definida en el esquema implementado, y pruebas del servicio utilizando varios artículos científicos escritos en LaTeX. El **Apartado 3.2** ofrece una validación de la funcionalidad del servicio mediante una encuesta digital realizada a diversos usuarios, incluyendo personas con discapacidad visual.

Finalmente, los **Apartado 3.3, Apartado 3.4 y Apartado 3.5** proporcionan la conclusión del presente Trabajo de Integración Curricular, junto con recomendaciones y posibles líneas de trabajo futuro.

3.1 ANÁLISIS DE RESULTADOS

Para realizar pruebas del servicio generador de descripciones implementado, se hace uso de alguna herramienta o programa que permita enviar una petición HTTP de tipo POST al servicio, conteniendo un archivo .zip en el cuerpo. Dicha petición se realiza sobre la URL *"**http://localhost:8001/upload**"*, donde el servidor mostrará un mensaje cuando reciba el archivo .zip exitosamente, tal y como se muestra en la Figura 3.1.

```
***   * Serving Flask app '__main__'
      * Debug mode: off
     INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment.
      * Running on all addresses (0.0.0.0)
      * Running on http://127.0.0.1:8001
      * Running on http://172.19.0.2:8001
     INFO:werkzeug:Press CTRL+C to quit
     Archivo ZIP recibido: /media/Llama-MSVH/Archivos_TEX/Paper2/Paper2.zip
```

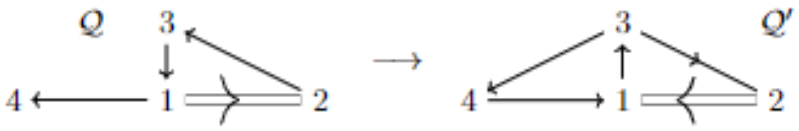
Figura 3.1 Mensajes que imprime el servidor cuando recibe un archivo .zip.

Una vez recibido el archivo .zip, se comienza con su procesamiento, extracción de texto y figuras, aplicación de RAG y del esquema de peticiones, para así culminar con una respuesta JSON con la colección de descripciones de todas las Figuras SVG o TikZ que el documento LaTeX contenga. El resultado que entrega el servicio contiene el identificador de la figura en base a su aparición en el archivo .tex, su formato (SVG o TikZ), su código, su subtítulo, su etiqueta, y las tres descripciones generadas en idioma inglés.

Utilizando una de las figuras del documento LaTeX titulado "*Integrable Systems and Cluster Algebras*", disponible en [39], en la Tabla 3.1 se presenta una comparación entre

las tres descripciones generadas, resultantes del esquema de peticiones implementado. La primera es una descripción breve obtenida a partir de la inferencia textual del código de la figura. La segunda descripción amplía la información de la primera, incorporando el contexto del documento LaTeX mediante retroalimentación. Finalmente, la tercera descripción proporciona un mayor detalle de la figura y su significado dentro del documento, demostrando una mejora notable que puede ser de gran utilidad para la comprensión de la figura cuando no pueda ser distinguida visualmente, así como para personas con cierto nivel de discapacidad visual.

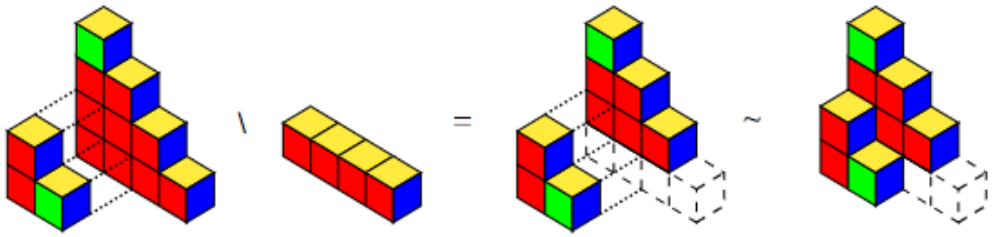
Tabla 3.1. Descripciones obtenidas del servicio generador de descripciones.

Figura con el subtítulo "*Quiver mutation at vertex 1.*"		
		
Descripción 1 (Código)	**Descripción 2 (RAG)**	**Descripción 3 (Detallada)**
The given figure code represents a quiver mutation at vertex 1, which transforms the quiver on the left-hand side into the quiver on the right-hand side. The quiver on the left-hand side has four vertices (1, 2, 3, and 4) and three edges between them, while the quiver on the right-hand side has the same vertices and edges, but with the orientation of the edges reversed. Specifically, the edge between vertices 1 and 2 is reversed,	The figure labeled 'FigQM' in the document titled 'Integrable systems and cluster algebras' depicts a quiver mutation at vertex 1. The quiver on the left-hand side has four vertices (1, 2, 3, and 4) and three edges between them, while the quiver on the right-hand side has the same vertices and edges, but with the orientation of the edges reversed. The figure is drawn using the `tikz` package in LaTeX. It serves as an example of a quiver mutation, which is a fundamental concept in the theory of cluster algebras.	The figure depicts a quiver mutation at vertex 1, which transforms the quiver on the left-hand side into the quiver on the right-hand side. The quiver on the left-hand side has four vertices (1, 2, 3, and 4) and three edges between them, represented by solid lines. The edges are labeled with integers, specifically 1, 2, and 3. The vertex 1 is surrounded by a circle in both quivers, indicating that it is the vertex being mutated. In the quiver on the right-hand side, the edges are reversed, meaning that the arrowheads point in the opposite direction.

and the edge between vertices 3 and 4 is reversed. The figure code uses the `tikz` package in LaTeX to draw the quivers.		Specifically, the edge between vertices 1 and 2 is reversed, as well as the edge between vertices 3 and 4. Additionally, the labeling of the edges is updated accordingly, with the edge between vertices 1 and 2 now labeled 2 and the edge between vertices 3 and 4 now labeled 3. The figure serves as an example of a quiver mutation, which is a fundamental concept in the theory of cluster algebras. Quiver mutations are used to transform one quiver into another by changing the orientation of certain edges. This process can be applied repeatedly, resulting in a sequence of quivers and mutations that can be used to study the properties of cluster algebras."

Por otro lado, en la Tabla 3.2 se presenta la descripción final de una de las figuras del documento LaTeX titulado " *Unexpected but recurrent phenomena for Quot and Hilbert schemes of points*", disponible en [40]. Esta figura es compuesta, con una representación tridimensional de varios bloques. En este caso, se puede evidenciar que la descripción generada no logra alcanzar una precisión adecuada, e incluso menciona detalles que no corresponden con la figura. Esto sugiere una falta de precisión en el procesamiento textual del modelo "*Llama 2*", siendo uno de los enfoques principales para el trabajo futuro.

Tabla 3.2. Descripciones finales obtenidas del servicio generador de descripciones donde se aprecia una falta de precisión.

Figura con el subtítulo "*Pictorial description of the algebra R/I . The socle boxes identified by the binomial generator in (1.6) are the green ones*"

Descripción 3 (Detallada)
1. The figure depicts a nested Hilbert scheme of points, with various rectangles, triangles, and lines representing different subschemes. 2. The top-left rectangle represents the optimal Hilbert function of the nested Hilbert scheme, with each cell in the rectangle corresponding to a subscheme. 3. The bottom-left triangle represents the tangent space of the nested Hilbert scheme, with each edge labeled with a number indicating the dimension of the tangent space at that point. 4. The top-right corner of the figure contains five rectangles representing the coordinate ring of a nested Hilbert scheme, with each rectangle corresponding to a different graded piece of the ring. The rectangles are labeled with the variables corresponding to the different graded pieces. 5. The bottom-right corner of the figure contains four triangles representing the ideal generated by the binomial generator, with each triangle corresponding to a different component of the ideal. The triangles are labeled with the variables corresponding to the different components. 6. The middle row of the figure contains six small rectangles representing the six non-reme lexically Ak Li functions of degree two that arise in the generic non-primitive case. The rectangles are labeled with the variables corresponding to the different non-reme lexically Ak Li functions. 7. The right column of the figure contains two vertical lines with numbers on them, indicating the degrees of the non-reme lexically Ak Li functions of degree two, with the left line representing the primitive case and the right line representing the generic non-primitive case.

8. The figure also includes various smaller rectangles and triangles, each representing a specific subscheme of the nested Hilbert scheme, and each labeled with the corresponding variables.
9. The rectangles and triangles in the figure are arranged in a pictorial way to convey the relationships between the subschemes and the tangent space of the nested Hilbert scheme.
10. The figure also includes arrows indicating the natural maps between the tangent spaces of the different subschemes and the nested Hilbert scheme."

Con estos resultados se puede demostrar que la capacidad de inferencia y el procesamiento textual del LLM "*Llama 2*" tiene un rendimiento propositivo en la tarea de describir figuras no rasterizadas, con sus limitaciones para ciertas figuras más compuestas.

Las descripciones de todas las figuras de estos documentos y otros más que han sido utilizados para las pruebas se encuentran en el apartado **Anexo III**.

3.2 VALIDACIÓN DE FUNCIONALIDAD

Para realizar las pruebas de validación de funcionalidad del servicio generador de descripciones, se busca obtener la retroalimentación de la calidad y precisión de las descripciones finales generadas por parte de los usuarios de prueba. Por ello, se definen dos escenarios para llevar a cabo estas pruebas: el primero va a comprender personas naturales y estudiantes, y el segundo comprenderá personas con cierta discapacidad visual, en colaboración con la Asociación de Invidentes Milton Vedado [27]. Para ambos escenarios de prueba se hace uso de un esquema de recopilación de información estructurada[28], como una encuesta. Los resultados obtenidos van a permitir medir la utilidad de este servicio para una mejor comprensión de las figuras mediante una descripción detallada.

3.2.1 Diseño de encuesta para validación del servicio

La encuesta persigue los siguientes objetivos:

- Evaluar la usabilidad del servicio generador de descripciones de figuras codificadas, presentes en documentos LaTeX.

[27] En el Apartado **Anexo IV** se encuentra el Acuerdo de Colaboración firmado por la asociación, sirviendo como documento que abala las pruebas realizadas.
[28] Una encuesta estructurada tiene preguntas cerradas con opciones de respuesta específicas.

- Validar si las descripciones generadas por el servicio mejoran el entendimiento de las figuras en comparación con sus subtítulos originales.

Para cumplir estos objetivos, se diseña una encuesta estructurada que incluye varias figuras. Cada figura está acompañada de su subtítulo original del documento de origen y de la descripción final generada por el servicio. En cada ítem de la encuesta, se solicita a los usuarios que evalúen si la descripción generada por el servicio mejora el entendimiento de la figura en comparación con su subtítulo original.

El formato de los ítems de la encuesta se ilustra en la Figura 3.2, donde se incluye un video con la sonificación de las descripciones de cada figura, haciendo uso del componente presentado en el Trabajo de Integración Curricular titulado "**Prototipo de Aplicación Web para generar archivos PDF inclusivos**" por Carlos León [].

La encuesta completa se puede observar en el apartado **Anexo V.**

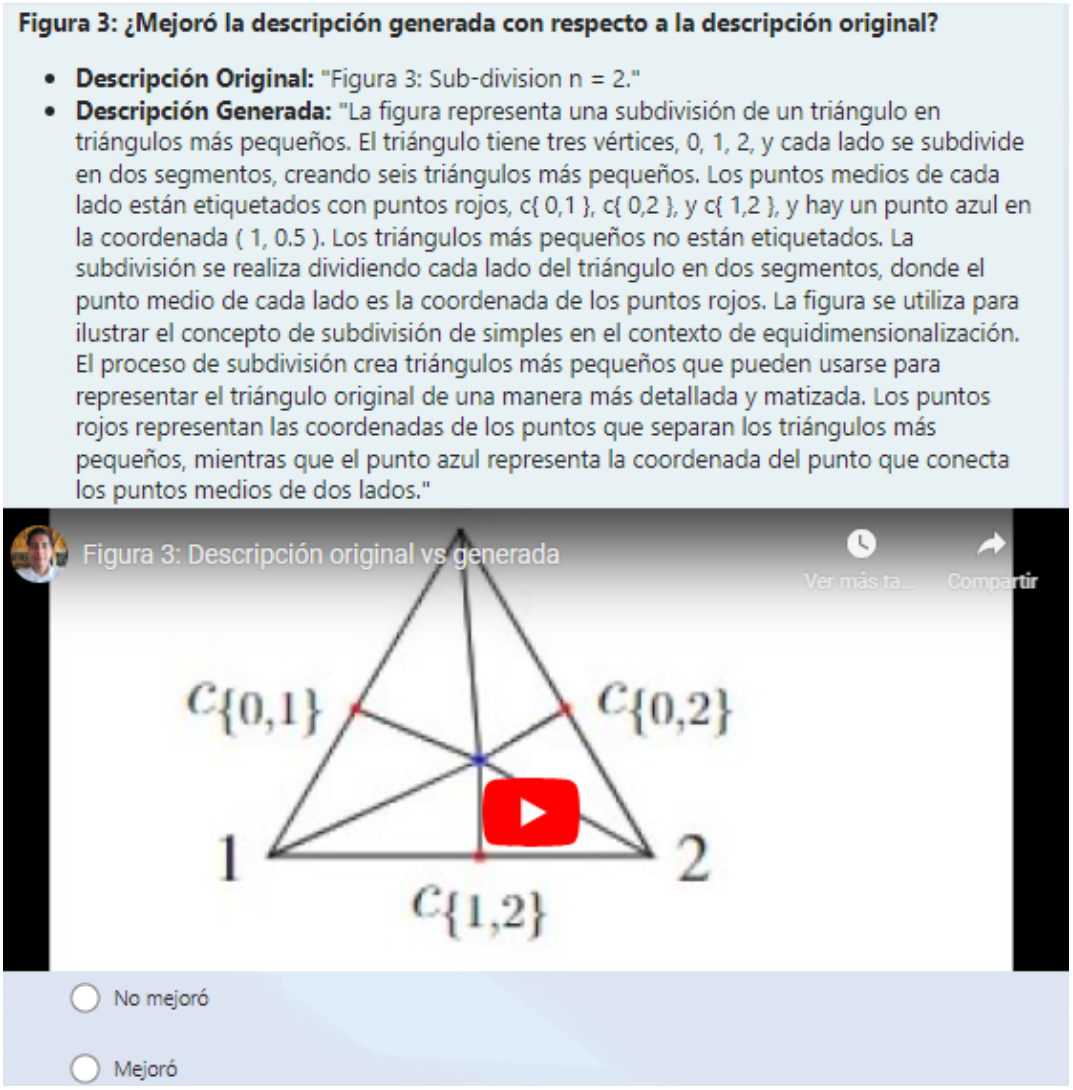


Figura 3.2 Formato de la encuesta para validación de funcionalidad del servicio.

3.2.2 Resultados de la encuesta

Los resultados de la encuesta realizada a una muestra de 20 personas, evaluando ambos escenarios de prueba, son satisfactorios. Como se muestra en la Figura 3.3, el 97% de los usuarios brindaron una retroalimentación positiva, destacando una mejora en la comprensión de todas las figuras mediante las descripciones generadas. En contraste, solo un 3% de los encuestados proporcionaron retroalimentación negativa en cuanto a la comprensión de alguna de las figuras, lo cual sugiere áreas de mejora futura para el servicio. En la Figura 3.4 se puede observar con más detalle los resultados obtenidos por pregunta, es decir, por cada figura presentada en la encuesta.

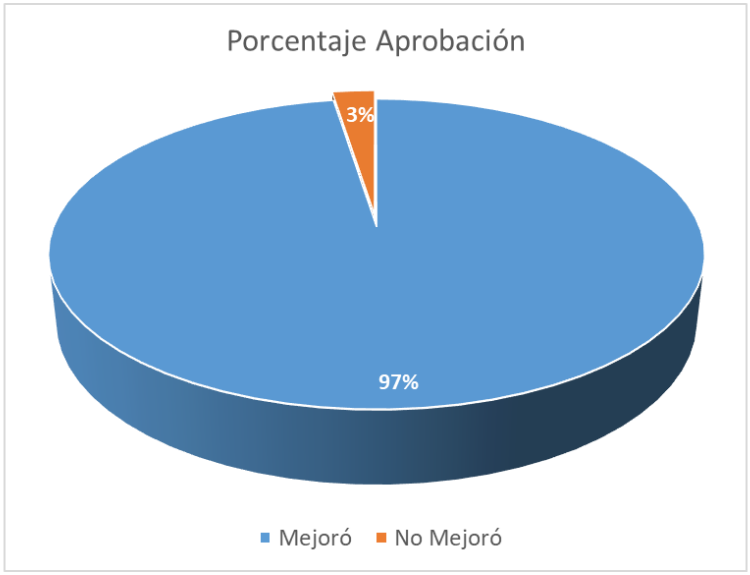


Figura 3.3 Gráfica estadística de resultados totales de la encuesta realizada.

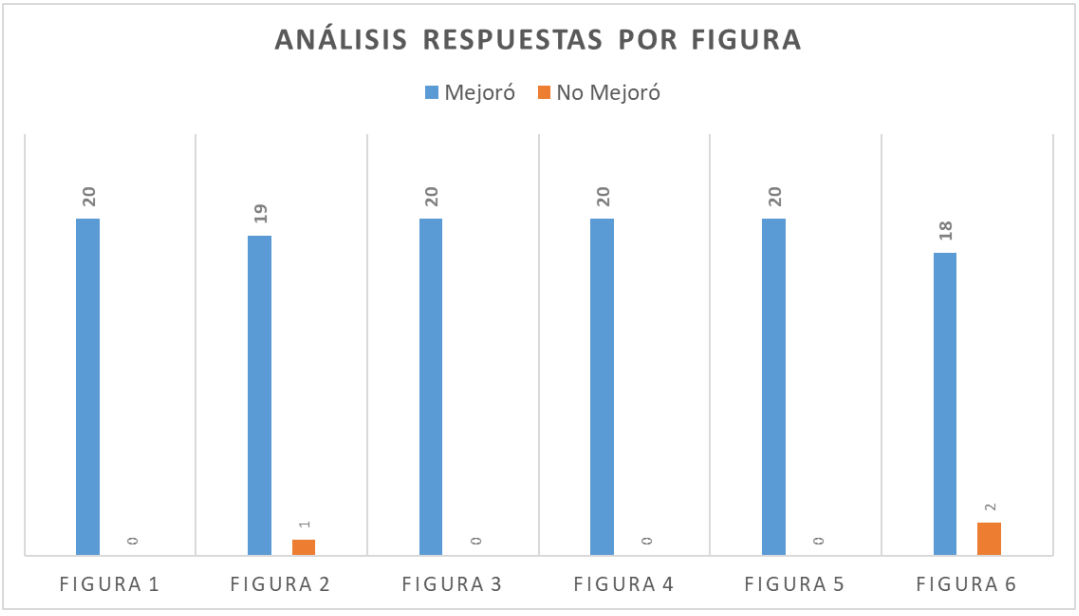


Figura 3.4 Gráfica estadística de resultados por pregunta de la encuesta realizada.

Por lo tanto, los resultados obtenidos demuestran y validan que la funcionalidad del servicio generador de descripciones de figuras SVG o TikZ cumple con los objetivos propuestos en este Trabajo de Integración Curricular.

En el apartado **Anexo VI** se encuentra el consolidado de las respuestas obtenidas de la encuesta.

3.3 FINALIZACIÓN DEL TABLERO KANBAN

Con la culminación del Trabajo de Integración Curricular, se finaliza el tablero Kanban, validando que todas las actividades de las diferentes fases se completaron según lo previsto en el plan. Esto se puede evidenciar en el tablero de la Figura 3.5.

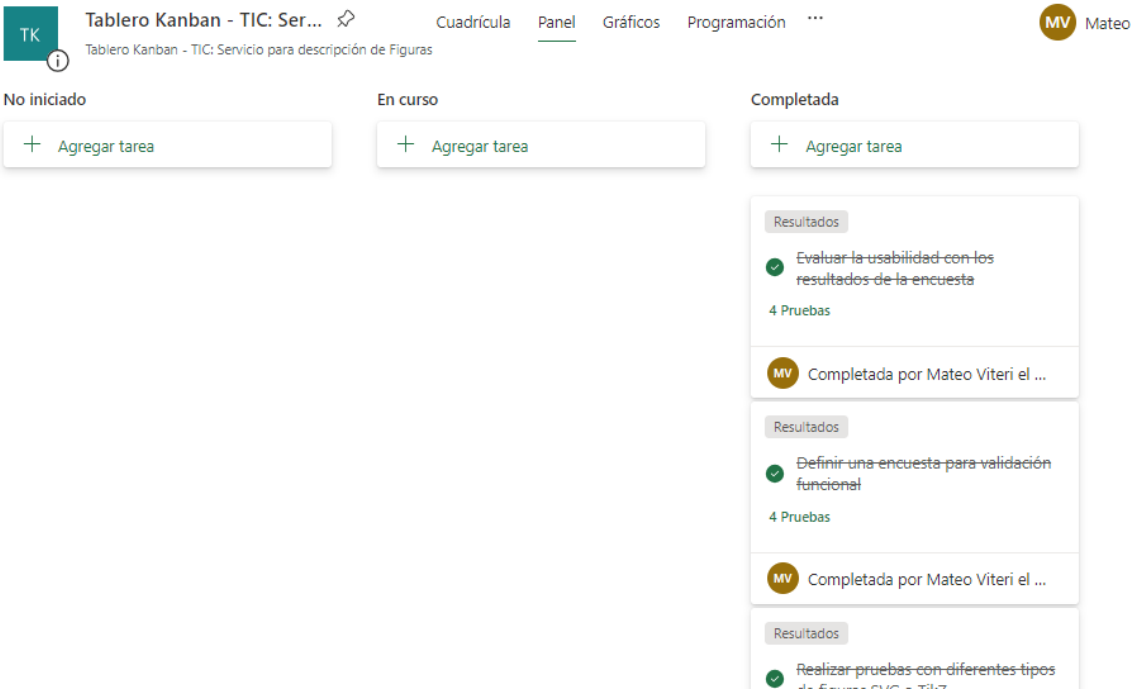


Figura 3.5 Tablero Kanban del Trabajo de Integración Curricular completado.

3.4 CONCLUSIONES

Como conclusiones obtenidas del presente Trabajo de Integración Curricular se tienen:

- El servicio generador de descripciones detalladas y contextualizadas de figuras SVG o TikZ en documentos académicos y científicos en formato LaTeX puede contribuir significativamente al entendimiento de las figuras y la temática presentada en dichos documentos. Además, este servicio mejora la inclusividad, haciendo que los contenidos sean accesibles para personas con discapacidades visuales.

- El servicio generador de descripciones demostró tener una precisión satisfactoria en la generación automática de descripciones para figuras en formato SVG o TikZ, cumpliendo con las funcionalidades necesarias para su aplicación práctica.
- El aprovechamiento de la capacidad de procesamiento textual en tareas visuales del LLM "*Llama 2*" de 70 mil millones de parámetros permitió generar descripciones detalladas de figuras no rasterizadas, siendo efectivos en la inferencia de detalles visuales basados en su código y el contexto proporcionado.
- Si bien actualmente existen soluciones para generar subtítulos de figuras mediante algoritmos de procesamiento en visión computacional, como se revisó en el estudio del Estado del Arte, la solución propuesta en este Trabajo de Integración Curricular ha logrado describir figuras no rasterizadas con resultados prometedores, utilizando una cantidad significativamente menor de recursos computacionales.
- Los resultados de la encuesta y las pruebas realizadas confirmaron que las descripciones generadas por el servicio mejoran significativamente la comprensión, recreación e interpretación de figuras para personas con un cierto grado de discapacidad visual. La inclusión de contextos específicos del documento original en las descripciones contribuyó a un entendimiento más completo de los elementos visuales.
- Aunque el servicio cumple con los objetivos propuestos, se identificaron ciertos aspectos a considerar en el trabajo futuro, como la precisión en la descripción de figuras compuestas y tridimensionales. Esto puede significar una limitante de los LLM, específicamente del modelo "*Llama 2*", en el procesamiento textual de figuras.

3.5 RECOMENDACIONES

Durante el desarrollo del presente Trabajo de Integración Curricular se han identificado ciertas consideraciones importantes al momento de implementar el servicio, así como el levantamiento del modelo "*Llama 2*". En base a esto se establecen las siguientes recomendaciones:

- Al momento de escoger el LLM con el que se vaya a trabajar, es importante revisar la documentación de los requerimientos de hardware y software que

implica su implementación, ya que esto permitirá dimensionar correctamente los recursos que debe tener el equipo donde se vaya a poner en marcha dicho modelo, evitando así inconvenientes.

- El uso de modelos de código abierto es más sencillo si se utilizan herramientas como *Oobabooga,* ya que contienen todos los componentes necesarios para levantar y usar dicho modelo. No obstante, herramientas como esta funcionan bajo ciertas condiciones, como la versión del controlador de la tarjeta gráfica, la versión de la arquitectura CUDA, en el caso de NVIDIA, la versión de Python, entre otras. Por ello es importante apoyarse de foros y guías de instalación para evitar inconvenientes, como la que se generó durante el desarrollo de este Trabajo de Integración Curricular, disponible en [41].

- Como buena práctica en la implementación de servicios, es recomendable utilizar entornos virtuales o contenedores. Estos entornos proporcionan un aislamiento eficaz del servicio respecto al sistema anfitrión, asegurando que las dependencias y configuraciones específicas del servicio no interfieran con otros procesos del sistema. Además, facilitan la gestión y despliegue del servicio, permitiendo replicar el entorno de manera consistente en diferentes sistemas y entornos de desarrollo.

- Al procesar documentos LaTeX, es recomendable seleccionar una muestra significativa de estos para identificar tanto los formatos comunes como los inusuales. Esto permite establecer expresiones regulares más genéricas, garantizando su eficacia en la mayoría de los casos. Además, es crucial utilizar las diversas bibliotecas disponibles en Python para el procesamiento de texto en archivos LaTeX. Estas bibliotecas pueden reducir significativamente la complejidad y heterogeneidad del procesamiento.

Así también, como trabajo futuro y posibles mejoras en el servicio generador de descripciones de figuras SVG o TikZ, se puede considerar lo siguiente:

- Este servicio tiene el potencial de ser expandido y adaptado a otros formatos de documentos y tipos de figuras, incrementando su aplicabilidad y beneficios. Por ello, se podría buscar la manera de integrarlo con otras herramientas de accesibilidad y la colaboración con comunidades de usuarios finales, adaptándolo para diversas necesidades.

- Se podría ampliar el conjunto de datos de entrenamiento con ejemplos más diversos y complejos. Incluir figuras tridimensionales y compuestas de diversos

campos científicos para mejorar la generalización del modelo. Se recomienda continuar trabajando en el perfeccionamiento de los modelos de procesamiento textual y explorar nuevas técnicas que puedan aumentar la exactitud y la utilidad de las descripciones generadas.

- Se podría desarrollar complementos o extensiones para editores LaTeX populares, permitiendo a los usuarios generar y visualizar descripciones directamente desde su entorno de trabajo. De esta forma se podría ofrecer una alternativa para mejorar la creación de artículos científicos en temas de inclusividad.

4 REFERENCIAS BIBLIOGRÁFICAS

[1] W. W. A. Initiative (WAI), «Sumario de WCAG 2», Web Accessibility Initiative (WAI). Accedido: 22 de mayo de 2024. [En línea]. Disponible en: https://www.w3.org/WAI/standards-guidelines/wcag/es

[2] E. Academy, «Effective Use of Tables and Figures in Research Papers», Enago Academy. Accedido: 22 de mayo de 2024. [En línea]. Disponible en: https://www.enago.com/academy/how-to-use-tables-and-figures-to-effectively-organize-data-in-research-papers/

[3] «Inteligencia artificial para mejorar las descripciones de las fotos para las personas ciegas y con discapacidad visual», Acerca de Meta. Accedido: 22 de mayo de 2024. [En línea]. Disponible en: https://about.fb.com/ltam/news/2021/01/como-facebook-utiliza-la-inteligencia-artificial-para-mejorar-las-descripciones-de-las-fotos-para-las-personas-ciegas-y-con-discapacidad-visual/

[4] «Meta Llama 2», Meta Llama. Accedido: 23 de mayo de 2024. [En línea]. Disponible en: https://llama.meta.com/llama2/

[5] «¿Qué son los grandes modelos de lenguaje (LLM)? | IBM». Accedido: 23 de mayo de 2024. [En línea]. Disponible en: https://www.ibm.com/mx-es/topics/large-language-models

[6] J. Franganillo, «Los grandes modelos de lenguaje: una oportunidad para la profesión bibliotecaria», *Anu. ThinkEPI*, vol. 17, oct. 2023, doi: 10.3145/thinkepi.2023.e17a28.

[7] «¿Cómo funcionan los Transformers? en Español | Aprende Machine Learning». Accedido: 29 de mayo de 2024. [En línea]. Disponible en: https://www.aprendemachinelearning.com/como-funcionan-los-transformers-espanol-nlp-gpt-bert/

[8] H. Touvron *et al.*, «LLaMA: Open and Efficient Foundation Language Models». arXiv, 27 de febrero de 2023. doi: 10.48550/arXiv.2302.13971.

[9] «Llama 2: Open Foundation and Fine-Tuned Chat Models | Research - AI at Meta». Accedido: 28 de mayo de 2024. [En línea]. Disponible en: https://ai.meta.com/research/publications/llama-2-open-foundation-and-fine-tuned-chat-models/

[10] G. Team, «Gradio». Accedido: 29 de mayo de 2024. [En línea]. Disponible en: https://gradio.app

[11] oobabooga, «oobabooga/text-generation-webui». 29 de mayo de 2024. Accedido: 28 de mayo de 2024. [En línea]. Disponible en: https://github.com/oobabooga/text-generation-webui

[12] «Crea un chatbot inteligente con la API de OpenAI», OpenWebinars.net. Accedido: 29 de mayo de 2024. [En línea]. Disponible en: https://openwebinars.net/blog/chatbot-api-openai/
[13] P. Lewis *et al.*, «Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks», en *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2020, pp. 9459-9474. Accedido: 30 de mayo de 2024. [En línea]. Disponible en: https://papers.nips.cc/paper/2020/hash/6b493230205f780e1bc26945df7481e5-Abstract.html
[14] *Learn RAG From Scratch – Python AI Tutorial from a LangChain Engineer*, (17 de abril de 2024). Accedido: 30 de mayo de 2024. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=sVcwVQRHIc8
[15] agnes sellgren, «The Official Guide to The Kanban Method | Kanban University». Accedido: 30 de mayo de 2024. [En línea]. Disponible en: https://kanban.university/kanban-guide/
[16] «What is Kanban Methodology? The Ultimate Guide | Wrike». Accedido: 30 de mayo de 2024. [En línea]. Disponible en: https://www.wrike.com/kanban-guide/what-is-kanban/
[17] M. Cupo, «Todo lo que tienes que saber sobre los tableros Kanban», monday.com Blog. Accedido: 30 de mayo de 2024. [En línea]. Disponible en: https://monday.com/blog/es/desarrollo/todo-sobre-los-tableros-kanban/
[18] «¿Qué Es Un Tablero Kanban? - Los Fundamentos». Accedido: 30 de mayo de 2024. [En línea]. Disponible en: https://www.nimblework.com/es/kanban/kanban-board/
[19] M. Cai, Z. Huang, Y. Li, H. Wang, y Y. J. Lee, «Delving into LLMs' visual understanding ability using SVG to bridge image and text», oct. 2023, Accedido: 16 de junio de 2024. [En línea]. Disponible en: https://openreview.net/forum?id=pwlm6Po61I
[20] M. Cai, Z. Huang, Y. Li, H. Wang, y Y. J. Lee, «Leveraging Large Language Models for Scalable Vector Graphics-Driven Image Understanding». arXiv, 9 de junio de 2023. doi: 10.48550/arXiv.2306.06094.
[21] S. Mehta, «Image Captioning with Mistral 7B LLM: A Hands-on Guide», Association of Data Scientists. Accedido: 15 de junio de 2024. [En línea]. Disponible en: https://adasci.org/image-captioning-with-mistral-7b-llm/
[22] «LLM-Powered Image Caption Generation - Challenges, and Applications», Labellerr. Accedido: 15 de junio de 2024. [En línea]. Disponible en: https://www.labellerr.com/blog/llm-powered-image-caption-generation-challenges-and-applications/
[23] «Insights And Best Practices For Building and Deploying Computer Vision Models», Labellerr. Accedido: 7 de julio de 2024. [En línea]. Disponible en: https://www.labellerr.com/blog/building-and-deploying-computer-vision-models/
[24] «Organizar las tareas de su equipo en Microsoft Planner - Soporte técnico de Microsoft». Accedido: 7 de julio de 2024. [En línea]. Disponible en: https://support.microsoft.com/es-es/office/organizar-las-tareas-de-su-equipo-en-microsoft-planner-c931a8a8-0cbb-4410-b66e-ae13233135fb
[25] «System requirements», GitHub. Accedido: 21 de junio de 2024. [En línea]. Disponible en: https://github.com/oobabooga/text-generation-webui/wiki/System-requirements
[26] «furiosa-ai/llama2». FuriosaAI, 13 de marzo de 2024. Accedido: 21 de junio de 2024. [En línea]. Disponible en: https://github.com/furiosa-ai/llama2
[27] «Start Locally», PyTorch. Accedido: 21 de junio de 2024. [En línea]. Disponible en: https://pytorch.org/get-started/locally/
[28] «TheBloke (Tom Jobbins)». Accedido: 22 de junio de 2024. [En línea]. Disponible en: https://huggingface.co/TheBloke

[29] «SentenceTransformers Documentation — Sentence Transformers documentation». Accedido: 22 de junio de 2024. [En línea]. Disponible en: https://sbert.net/
[30] «faiss-cpu: A library for efficient similarity search and clustering of dense vectors.»
[31] «Welcome to pylatexenc's documentation! — pylatexenc 3.0alpha000021 documentation». Accedido: 22 de junio de 2024. [En línea]. Disponible en: https://pylatexenc.readthedocs.io/en/latest/
[32] «requests: Python HTTP for Humans.» Accedido: 22 de junio de 2024. [OS Independent]. Disponible en: https://requests.readthedocs.io
[33] «Python JSON». Accedido: 22 de junio de 2024. [En línea]. Disponible en: https://www.w3schools.com/python/python_json.asp
[34] «re — Regular expression operations», Python documentation. Accedido: 22 de junio de 2024. [En línea]. Disponible en: https://docs.python.org/3/library/re.html
[35] «Pandas Tutorial». Accedido: 22 de junio de 2024. [En línea]. Disponible en: https://www.w3schools.com/python/pandas/default.asp
[36] «Welcome to Flask — Flask Documentation (3.0.x)». Accedido: 22 de junio de 2024. [En línea]. Disponible en: https://flask.palletsprojects.com/en/3.0.x/
[37] «zipfile — Work with ZIP archives», Python documentation. Accedido: 22 de junio de 2024. [En línea]. Disponible en: https://docs.python.org/3/library/zipfile.html
[38] «Colaboratory Local runtimes docs instructions for Colab Docker runtime image fail · Issue #3724 · googlecolab/colabtools», GitHub. Accedido: 23 de junio de 2024. [En línea]. Disponible en: https://github.com/googlecolab/colabtools/issues/3724
[39] M. Gekhtman y A. Izosimov, «Integrable systems and cluster algebras». arXiv, 11 de marzo de 2024. doi: 10.48550/arXiv.2403.07287.
[40] F. Giovenzana, L. Giovenzana, M. Graffeo, y P. Lella, «Unexpected but recurrent phenomena for Quot and Hilbert schemes of points». arXiv, 21 de junio de 2024. doi: 10.48550/arXiv.2403.03146.
[41] M. Viteri, «Parte 2 - Instalación de Oobabooga», Notion. Accedido: 10 de julio de 2024. [En línea]. Disponible en: https://fern-table-d85.notion.site/Parte-2-Instalaci-n-de-Oobabooga-37198da4c06f432b85426fea10b4b63c

5 ANEXOS

Los Anexos de este Trabajo de Integración Curricular están disponibles en el repositorio https://github.com/MaVity22/MSVH_TIC_2024. Se enumeran de la siguiente manera:

ANEXO I: Guía de instalación y levantamiento del LLM Meta Llama 2

ANEXO II: Código completo del Servicio Generador de Descripciones de Figuras SVG o TikZ en documentos LaTeX

ANEXO III: Descripciones generadas de figuras presentes en varios artículos escritos en LaTeX

ANEXO IV: Acuerdo de Consentimiento firmado por la Asociación de Invidentes Milton Vedado

ANEXO V: Encuesta utilizada para validar la funcionalidad del servicio

ANEXO VI: Resultados consolidados de la encuesta realizada