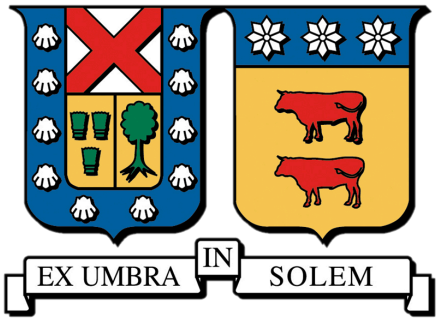
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE ELECTRÓNICA
VALPARAÍSO – CHILE



Diseño y Desarrollo de Plataforma de Apreciación Artistica para Personas con Discapacidad Visual

Alexey Nikolay Mitjaew Hupat

Memoria de Titulación para optar al título de Ingeniero Civil Telemático

Profesor Guía:	Patricio Olivares Roncagliolo
Profesor Correferente:	Nicolás Torres Rudloff

Diciembre 2025

Agradecimientos

A mi familia,
mi perro Elvis
y el buen café.

Resumen

Este documento expone el desarrollo de una plataforma accesible de apreciación artística para público con discapacidad visual. La propuesta consiste en transformar obras en representaciones sonoras multifacéticas, que integran descripciones narrativas detalladas, contexto histórico-artístico y atmósferas auditivas generadas mediante modelos multimodales de lenguaje, sistemas de síntesis de voz de alta y modelos de generación sonora basada en texto.

La solucion expuesta propone facilitar el acceso al arte mediante una experiencia adaptada, con el objetivo de ofrecer alternativas de apreciación sensorial para personas con discapacidad visual.

Palabras clave: *Accesibilidad / Arte inclusivo / Inteligencia artificial / Sonificación*

Lista de Figuras

Lista de Tablas

Contenidos

Introducción

El acceso equitativo al arte constituye un componente esencial del desarrollo cultural y social, sin embargo, las personas con discapacidad visual continúan enfrentando barreras significativas para disfrutar de obras visuales en condiciones comparables al resto del público. Aunque las tecnologías de apoyo han avanzado, aún existe una brecha entre la experiencia estética que ofrecen los museos y plataformas digitales tradicionales y las necesidades sensoriales de quienes no pueden percibir elementos visuales directamente.

En este contexto, la inteligencia artificial emerge como una oportunidad para reimaginar la forma en que se transmite el contenido artístico, permitiendo crear experiencias inmersivas que integren narrativa, sonido y contextualización cultural. Este proyecto desarrolla una plataforma que transforma obras de arte en representaciones sonoras enriquecidas, combinando modelos multimodales capaces de interpretar imágenes, sistemas de síntesis de voz de alta naturalidad y tecnologías generativas orientadas a la creación de paisajes sonoros coherentes con la obra original.

La presente investigación se enmarca en los principios de accesibilidad universal y diseño inclusivo, procurando que la experiencia resultante no solo sea funcional, sino también significativa desde una perspectiva estética. La plataforma busca responder a la necesidad de ofrecer alternativas sensoriales que amplíen el acceso al patrimonio artístico y cultural, fortaleciendo la inclusión mediante herramientas tecnológicas avanzadas.

Objetivo General

Mejorar la experiencia estética y promover la accesibilidad universal a las obras de arte para personas con discapacidad visual mediante el diseño e implementación de una plataforma accesible que utilice inteligencia artificial para transformar automáticamente imágenes de obras artísticas en paisajes sonoros narrativos, contextuales y ambientales.

Objetivos Particulares

- **Proveer un catálogo curado de obras de dominio público**, basado en fuentes autoritativas y verificadas, que permita a los usuarios acceder a contenido confiable, culturalmente riguroso y legalmente seguro.
- **Generar descripciones auditivas de alta calidad** utilizando modelos de lenguaje multimodales capaces de interpretar intuitivamente los elementos visuales presentes en cada obra, garantizando una representación objetiva y comprensible de su contenido.
- **Crear narraciones históricas y biográficas** fundamentadas en fuentes reconocidas, con el fin de contextualizar la obra, su autor(a) y su relevancia artística dentro de un marco cultural accesible.
- **Producir ambientes sonoros inmersivos** que recreen atmósferas coherentes con la escena o época sugerida por la obra, mediante modelos generativos capaces de transformar imágenes en audio evocativo y sensorialmente enriquecido.
- **Diseñar una interfaz centrada en accesibilidad**, que permita la navegación autónoma de personas en situación de discapacidad visual, incorporando criterios y normativas vigentes en materia de accesibilidad universal y diseño inclusivo.

Estructura

El presente análisis incluye:

- ***Revisión del estado del arte*** *sobre modelos generativos y estandares de accesibilidad.*
- ***Evaluación del marco teórico utilizado*** *para el desarrollo de la plataforma.*
- ***Descripción de la plataforma desarrollada*** *a nivel de componentes.*
- ***Revisión de resultados obtenidos y conclusiones al respecto****.*

Marco Teórico

Modelos de Lenguaje Multimodales

Los modelos de lenguaje (LLMs, por sus siglas en inglés) representan sistemas computacionales diseñados para procesar, comprender y generar texto en lenguaje natural mediante arquitecturas basadas en aprendizaje profundo. Estos modelos, entrenados en vastos corpus textuales, capturan patrones lingüísticos, semánticos y contextuales, permitiendo tareas como la traducción automática, la generación de respuestas coherentes o el análisis de sentimientos. Su evolución ha dado paso a los modelos de lenguaje multimodales, los cuales extienden estas capacidades al integrar múltiples formas de información, como imágenes y audio.

Estos modelos multimodales no solo interpretan contenido visual o auditivo, sino que también generan descripciones detalladas de escenas, objetos o composiciones artísticas, traduciendo elementos visuales en narrativas estructuradas. Además, su capacidad multimodal les permite procesar entradas de voz y participar en flujos conversacionales voz a voz, facilitando interacciones más naturales y accesibles. Esta convergencia de modalidades (texto, imagen y sonido) facilita la creación de sistemas más ricos y adaptables, capaces de ofrecer interpretaciones contextualizadas y personalizadas según las necesidades de percepción del usuario.

Modelos Generativos de Audio

Los modelos generativos de audio representan sistemas capaces de crear sonidos, música y voces artificiales a partir de descripciones semánticas o patrones aprendidos. Estos modelos, basados en arquitecturas de aprendizaje profundo, pueden sintetizar desde elementos literales —como el canto de pájaros, el sonido de una tormenta o pasos sobre grava— hasta composiciones musicales complejas, ajustándose a estilos, géneros o emociones específicas. Su funcionamiento se sustenta en la interpretación de instrucciones textuales o parámetros acústicos, permitiendo generar audio coherente y contextualizado

sin necesidad de muestras preexistentes. Esta capacidad no solo amplía las posibilidades creativas en producción musical y diseño sonoro, sino que también facilita la creación de entornos auditivos personalizados, como paisajes sonoros para aplicaciones de accesibilidad o asistentes de voz con tonos y matices más naturales.

Estándares de Accesibilidad

Los estándares de accesibilidad proporcionan el marco normativo que guía el diseño de interfaces inclusivas, asegurando que los contenidos digitales sean utilizables por personas con diversas capacidades sensoriales, motoras o cognitivas. Entre los más relevantes se encuentran las Pautas de Accesibilidad para el Contenido Web (WCAG), que establecen criterios relacionados con la perceptibilidad, operabilidad, comprensibilidad y robustez de los sistemas web. En el contexto chileno, estas directrices se integran mediante la Ley N.° 20.422, que regula la igualdad de oportunidades e incorpora exigencias específicas para tecnologías asistivas. La plataforma desarrollada en este proyecto adopta estos lineamientos para garantizar compatibilidad con lectores de pantalla, navegación por teclado, descripciones alternativas, controles auditivos accesibles y una estructura de interacción que facilite la exploración autónoma.

El cumplimiento de estos estándares es esencial para asegurar que la solución propuesta no solo sea técnicamente avanzada, sino también verdaderamente inclusiva y centrada en las necesidades de la comunidad usuaria.

Estado del Arte

Modelos de Lenguaje Multimodales

Durante los últimos años los modelos múltimodales de lenguaje han experimentado avances significativos, integrando capacidades de procesamiento de texto, imagen, audio y video. Este progreso ha sido impulsado por arquitecturas unificadas basadas en Transformers que permiten compartir un espacio de representación común entre modalidades. Avances como los *Vision-Language Encoders (VLMs)*, los *Audio-Text aligners* y la *Tokenización Unificada* han permitido integrar imágenes, audio y texto como secuencias compatibles. Además, técnicas como el *cross-attention multimodal*, la *late fusion optimizada* y el *modality routing* han mejorado la coherencia entre modalidades, habilitando tareas complejas de razonamiento y generación conjunta en múltiples formatos.

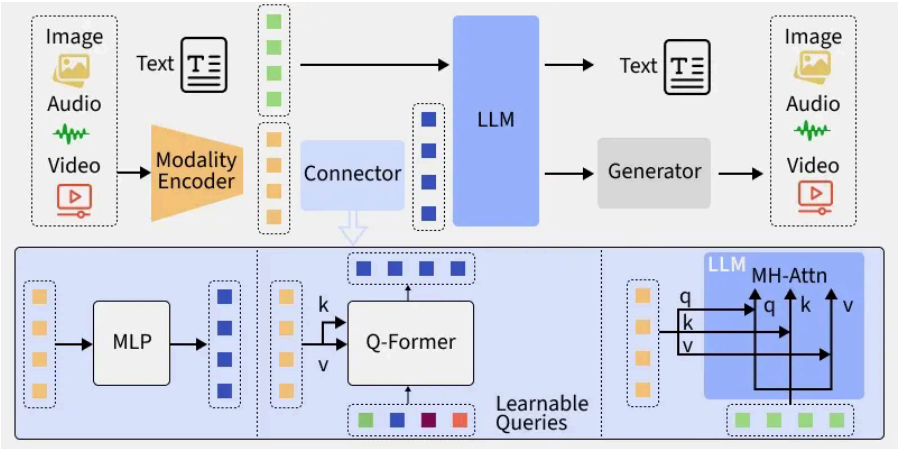


Figura 1: Multimodal LLMs

Entre los avances destacados hasta la fecha (2025) se encuentran:

- **Llama v4 Maverick (Meta)**: Modelo multimodal con descripción semántica detallada de imágenes y análisis de escenas. Pesos liberados para investigación.
- **GPT-4o (OpenAI) y Gemini 1.5 (Google)**: Modelos con integración nativa de texto, imagen y audio. Sin pesos liberados.
- **Voice-to-Voice (V2V)**: Sistemas como Whisper v3 (OpenAI) y SeamlessM4T (Meta) para transcripción y traducción en tiempo real. Voicebox (Meta) genera voz sintética con control de emociones. Whisper v3 tiene pesos liberados; SeamlessM4T y Voicebox no.

Modelos de Texto a Voz

En los últimos años, arquitecturas como Tacotron 2, VITS, Kokoro TTS y modelos basados en difusión/transformers han ganado relevancia. Estos modelos, generalmente ligeros en cantidad de parámetros, son razonables de autohostear con GPUs de grado consumidor e incluso en CPU en algunos casos. Generan voces naturales con control sobre prosodia, entonación y emociones. Kokoro TTS, de código abierto, destaca por su expresividad y ajuste de voces mediante manipulación del espacio latente, evitando reentrenamiento.

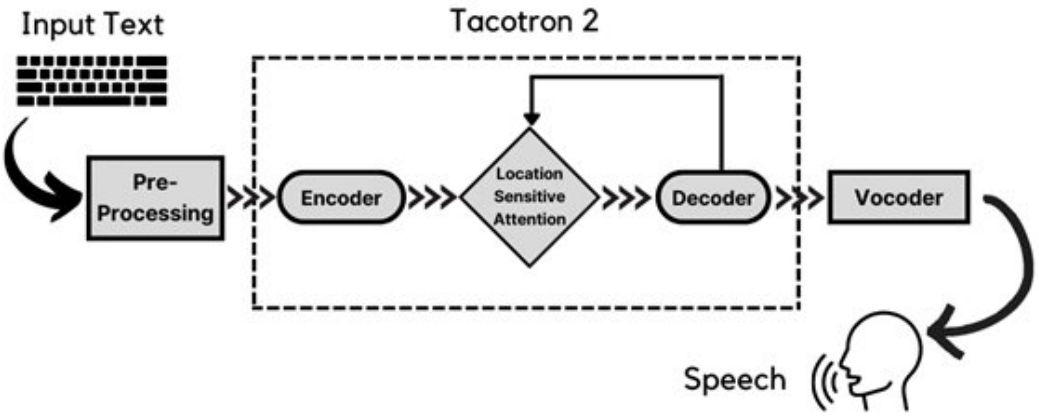


Figura 2: Tacotron 2 Arquitecture

Además de los modelos de código abierto, proveedores comerciales como Google Cloud, Amazon Polly, Microsoft Azure, ElevenLabs y OpenAI ofrecen soluciones TTS de alta calidad. Estas plataformas incorporan voces multilingües, opciones de personalización avanzada y capacidades de clonación vocal.

No obstante los avances observados, subsisten desafíos técnicos comunes en las alternativas investigadas, como la generación de expresividad controlable, la adaptación a entornos acústicos adversos y la síntesis eficiente de secuencias de larga duración.

Modelos Generativos de Ambiente Sonoro

La generación de ambientes sonoros de alta calidad sigue en fase experimental, sin soluciones comerciales consolidadas. Modelos como CLAP permiten mapear texto y audio en un espacio semántico compartido, aunque no generan audio directamente.

Entre los modelos evaluados para la plataforma se encuentran:

- **I Hear Your True Colors**: La arquitectura propuesta en este paper combina el uso de un VQVAE, transformers y CLIP. El VQVAE extrae representaciones jerárquicas como secuencias discretas, los transformers las modelan de forma autorregresiva y CLIP alinea el audio con lo visual. Aunque el código es abierto, no existen fuentes públicas de pesos preentrenados. Sin versiones comerciales.
- **AudioLDM**: Utiliza difusión para generar espectrogramas condicionados por texto, incorporando embeddings de CLAP para mejorar la alineación semántica entre descripciones textuales y contenido sonoro. Presenta resultados prometedores en coherencia semántica y diversidad acústica. El modelo preentrenado se puede encontrar con pesos abiertos.

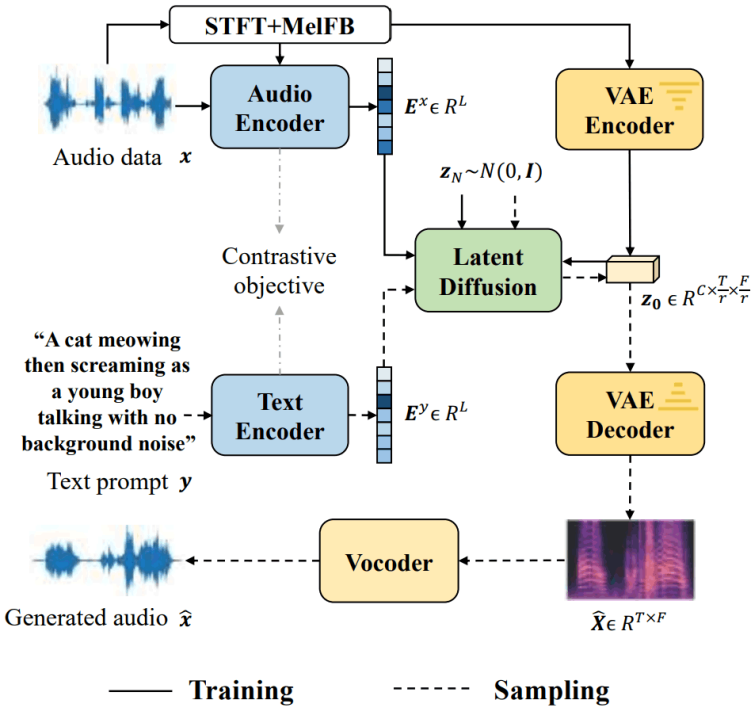


Figura 3: AudioLDM

Desarrollo de la Plataforma

La arquitectura propuesta sigue un modelo cliente-servidor e incorpora un *API Gateway* como componente central. Este elemento cumple dos funciones principales: por un lado, centraliza las operaciones computacionalmente intensivas (principalmente inferencia de modelos de aprendizaje automático y las solicitudes a APIs externas), y por otro, gestiona un catálogo centralizado de recursos. Además, se desarrolla un *frontend* que aplica principios de accesibilidad web, con especial atención a las necesidades de usuarios con discapacidad visual.

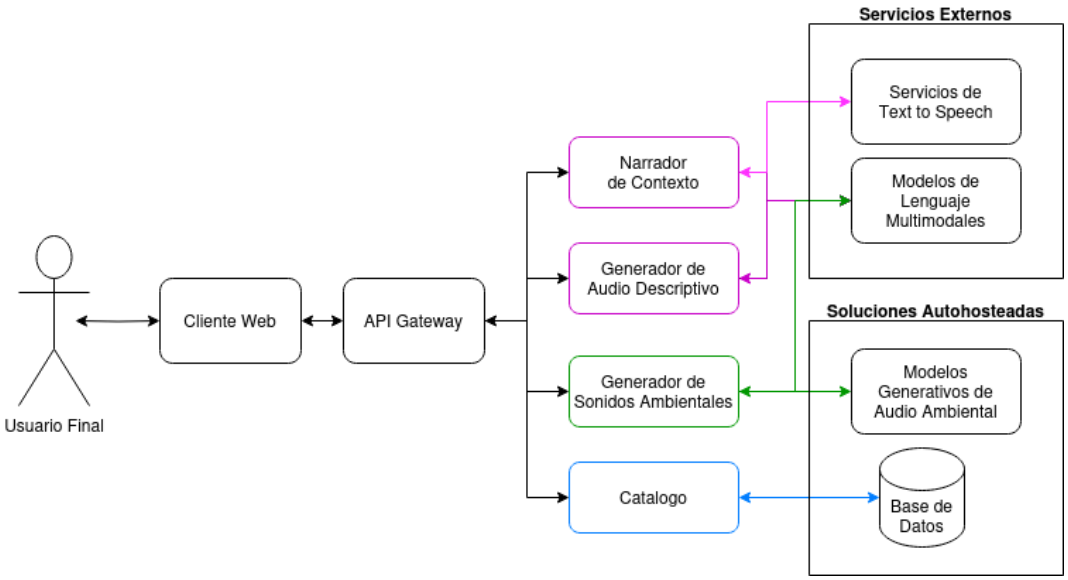


Figura 4: Arquitectura General de Sistema

API Gateway

La API Gateway consiste en un servidor de Django, encargado de controlar 4 servicios:

Narrador de Contexto

Este módulo se encarga de producir descripciones textuales detalladas y estructuradas de los contenidos visuales presentes en las obras. Para ello, emplea modelos de lenguaje multimodales, los cuales procesan tanto elementos gráficos como metadatos asociados mediante prompts especializados. Dichos prompts están diseñados para extraer información semántica relevante, garantizando una representación fiel y contextualizada del material original **(ANEXO PROMPT AQUI)**. Posteriormente, las descripciones generadas son convertidas a formato de audio mediante sistemas de síntesis de voz (TTS) **(ANEXO TTS AQUI)**, optimizados para ofrecer una experiencia auditiva clara y accesible.

Generador de Audio Descriptivo

Este módulo genera narraciones auditivas que contextualizan históricamente cada obra. Para ello, se recopila información de fuentes autoritativas (en este caso, se procesó manualmente un conjunto de datos con artículos de Wikipedia vinculados a cada pieza) **(ANEXO DATASET AQUI)**. A partir de estos datos, se elabora un relato contextualizado y adaptado al caso de uso **(ANEXO PROMPT AQUI)**, que posteriormente se convierte en audio mediante modelos de síntesis de voz (TTS) **(ANEXO TTS AQUI)**, asegurando una experiencia auditiva coherente y accesible.

Generador de Sonidos Ambientales

Este módulo se encarga de la generación de ambientes sonoros que representan los contenidos literales (no abstractos) de sus imagenes de entrada. La heuristica utilizada consiste en:

- Dividir el espacio en cuadrantes (en nuestro caso 9) **(ANEXO)**
- Procesar cada cuadrante con LLMs multimodales, obteniendo salidas estructuradas en JSON **(ANEXO)** representando cada elemento detectado.
- Por cada cuadrante generar una mezcla de sonido integrando todos los elementos detectados en cada respectiva sección. **(ANEXO)**

Catálogo de Obras/Imagenes

Se elaboró un conjunto de datos en formato JSON que incluye 30 obras artísticas, conteniendo todo el material requerido para el funcionamiento de los sistemas previamente descritos. Este dataset incorpora, además, enlaces a los artículos correspondientes de Wikipedia asociados a cada obra artística. **(ANEXO)**

Frontend

El frontend se implementó utilizando NextJS para proporcionar acceso al contenido gestionado por la API Gateway. En su diseño, se incorporaron textos alternativos y etiquetas HTML semánticas con el objetivo de garantizar una integración completa con herramientas de asistencia para usuarios con discapacidad visual. Asimismo, se desarrolló una interfaz de usuario que facilita la navegación mediante atajos de teclado, optimizando la accesibilidad y usabilidad del sistema.

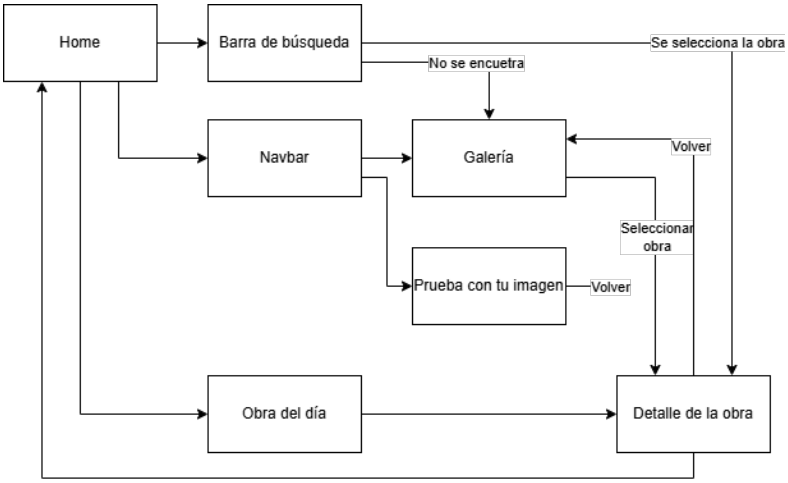


Figura 5: Flujo de Navegación Frontend

Limitaciones

Por limitaciones computacionales durante el desarrollo, los módulos no se integraron directamente en la API Gateway. En su lugar, se implementó un prototipo funcional en un cuadernillo Jupyter (https://www.kaggle.com/code/alexeymitjaew/sound-generation--pipeline) para validar el diseño y flujo de procesamiento. Esto permitió generar y preprocesar los conjuntos de datos necesarios, que fueron incorporados a la base de datos para su uso en la versión final.

Resultados

Conclusiones

[1] said [2] said

Bibliografía

[1] Ö. Aksın *et al.*, «Effect of immobilization on catalytic characteristics of saturated Pd-N-heterocyclic carbenes in Mizoroki-Heck reactions», *J.~Organomet. Chem.*, vol. 691, n.º 13, pp. 3027-3036, 2006.

[2] G. Westfahl, «The True Frontier». pp. 55-65.

Anexo

Utilidades Generales

En la prueba de concepto se emplearon modelos de lenguaje proporcionados por la plataforma *Groq*, accedidos mediante el cliente de Python de OpenAI. Este estándar es compatible con la mayoría de los proveedores de servicios de inferencia para modelos de lenguaje.

```python
client = openai.OpenAI(
    base_url = "https://api.groq.com/openai/v1",
    api_key = user_secrets.get_secret("GROQ_API_KEY")
)
```

Código 1: Inicialización de Cliente

Adicionalmente a lo anterior, se implementó una abstracción genérica para el procesamiento de **prompts** que contienen argumentos delimitados por llaves. Esta solución permite definir parámetros mediante notación de llaves (por ejemplo, `{clave}`) para su posterior sustitución mediante argumentos nombrados al invocar la función (ejemplo: `PARSE_PROMPT(prompt, clave=valor)`).

```python
def PARSE_PROMPT(BASE_PROMPT, **kwargs):
    for key in kwargs:
        BASE_PROMPT = BASE_PROMPT.replace(
            '{' + str(key) + '}',
            str(kwargs[key])
        )
    return BASE_PROMPT
```

Código 2: Utilidad de Procesado de Prompts

Pipeline de Traducción

La pipeline de traducción implementa un flujo automatizado para convertir textos de inglés a español. Utiliza las abstracciones definidas previamente para procesar un **prompt** estructurado que especifica reglas claras de traducción, como mantener el significado original, evitar redundancias y conservar el tono del texto fuente. La función `translate` integra este **prompt** con el texto de entrada mediante la utilidad de parsing de argumentos, enviando la solicitud al modelo y retornando la traducción generada de manera directa y sin modificaciones adicionales.

```python
TRANSLATOR_PROMPT = \
"""Traduce el siguiente texto al español de manera directa y clara.
Considera lo siguiente:
- Mantén el significado exacto del texto original.
- NO agregues explicaciones, comentarios o interpretaciones.
- NO seas redundante.
- Conserva el estilo y tono original del texto.
- Escribe solo la traducción, sin títulos ni subtítulos.

Texto original: {TEXT}
Traducción:"""

def translate(text):
    completion = client.chat.completions.create(
        model="openai/gpt-oss-120b",
        messages=[{
            "role": "user",
            "content": [{
                "type": "text",
                "text": PARSE_PROMPT(TRANSLATOR_PROMPT, TEXT=text)
            }]
        }],
        temperature=0,
        max_completion_tokens=1024,
        top_p=1,
        stream=False,
        stop=None,
    )
    content = completion.choices[0].message.content
    return content
```

Código 3: Script de Traducción

Pipeline de Extracción de Fuentes

Para extraer información procesada de fuentes autoritativas se procedió con el desarrollo de utilidades para extraer datos desde una URL y procesarlo en forma de bulletpointpara presentar los resultados de manera estructurada y clara en forma de viñetass.

```python
def extract_text_from_url(url: str) -> str:
    headers = {
        "User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0",
    }
    response = requests.get(url, headers=headers).raise_for_status()
    soup = BeautifulSoup(response.text, 'html.parser')

    for tag in soup(['script', 'style']):
        tag.decompose()

    paragraphs = [p.get_text(strip=True) for p in soup.find_all('p')]
    return '\n\n'.join(paragraphs)
```

Código 4: Scraper Extractor de Información

```python
SOURCE_EXTRACTOR_PROMPT = """Eres un extractor de información precisa y concisa. Tu tarea es leer el texto de un artículo y devolver una lista de viñetas (•) con todos los hechos y datos relevantes, escritos en español natural y claro.

Instrucciones:
1. Identifica los hechos principales, cifras, declaraciones, fechas, nombres y conclusiones clave del texto.
2. No incluyas opiniones, lenguaje publicitario o frases irrelevantes.
3. Si el texto está en inglés u otro idioma, traduce los puntos al español correctamente.
4. Usa frases breves pero completas, cada una iniciando con un punto (•).
5. Mantén el tono informativo y objetivo.

Texto del artículo: {TEXT}
Tu respuesta:"""
```

Código 5: Prompt de Extracción de Información

```python
def extract_url_context(url):
    text = extract_text_from_url(url)[:8_000]

    completion = client.chat.completions.create(
        model="openai/gpt-oss-120b",
        messages=[{
            "role": "user",
            "content": [{
                "type": "text",
                "text": PARSE_PROMPT(SOURCE_EXTRACTOR_PROMPT,
                TEXT=text)
            }]
        }],
        temperature=0,
        max_completion_tokens=2048,
        top_p=1,
        stream=False,
        stop=None,
    )
    content = completion.choices[0].message.content
    return content
```

Código 6: Script de Extracción de Información

Generación de Contenido General

Para los metadatos generales de la plataforma, aunque no visibles en el **frontend** de la prueba de concepto, se pobló la base de datos con períodos, técnicas pictóricas y breves biografías de autores históricos. Estos metadatos, aunque no visibles en el **frontend** de la prueba de concepto, proporcionan información contextual esencial para la interpretación y categorización de las obras de arte que en una etapa posterior del proyecto podrían enriquecer la experiencia del usuario.

Pasa la generación de estos se usaron dos enfoques distintos. Para técnicas pictóricas y períodos artísticos, se empleó generación directa con modelos de lenguaje por su estandarización. En cambio, para biografías de autores se extrajo información de Wikipedia para evitar imprecisiones en datos biográficos complejos, priorizando la calidad de la fuente.

Como se ha detallado previamente, la generación de este contenido requiere únicamente el nombre de la técnica o estilo pictórico a describir. Además, se han implementado consideraciones específicas en cuanto a la cadencia y el estilo del texto generado, con el objetivo de optimizar su procesamiento mediante modelos de síntesis de voz (TTS).

```python
TECHNIQUE_PROMPT = \
"""Explícame en detalle una técnica de pintura o dibujo.
Considera lo siguiente:
- Debe estar en un formato adecuado para narración.
- Debe ser conciso y corto en duración.
- NO digas en resumen.
- NO seas redundante.
- NO utilices títulos ni subtitulos, solo escribe en parrafos narrados.

Técnica: {TECHNIQUE}
Explicación:"""

def get_technique(technique):
    completion = client.chat.completions.create(
        #model="llama-3.1-8b-instant",
        model="llama-3.3-70b-versatile",
        messages=[{
            "role": "user",
            "content": [{
                "type": "text",
                "text": PARSE_PROMPT(TECHNIQUE_PROMPT, TECHNIQUE=technique)
            }]
        }],
        temperature=0,
        max_completion_tokens=1024,
        top_p=1,
        stream=False,
        stop=None,
    )
    content = completion.choices[0].message.content
    return content
```

Código 7: Generador de Texto de Técnica

```python
PERIOD_PROMPT = \
"""Explícame en detalle un estilo pictórico en particular.
Considera lo siguiente:
- Debe estar en un formato adecuado para narración.
- Debe ser conciso y corto en duración.
- NO digas en resumen.
- NO seas redundante.
- NO utilices títulos ni subtitulos, solo escribe en parrafos narrados.

Estilo pictórico: {PERIOD}
Explicación:"""

def get_period(period):
    completion = client.chat.completions.create(
        #model="llama-3.1-8b-instant",
        model="llama-3.3-70b-versatile",
        messages=[{
            "role": "user",
            "content": [{
                "type": "text",
                "text": PARSE_PROMPT(PERIOD_PROMPT, PERIOD=period)
            }]
        }],
        temperature=0,
        max_completion_tokens=1024,
        top_p=1,
        stream=False,
        stop=None,
    )
    return completion.choices[0].message.content
```

Código 8: Generador de Texto de Periodo

Se utilizó el script de extracción de información (Código 6) para procesar una breve biografía narrativa del autor, tomando datos de Wikipedia.

```python
AUTHOR_BIO_PROMPT = \
"""Eres un experto en redacción biográfica y narrativa.
Tu tarea es leer el siguiente texto y redactar una biografía breve y fluida del autor o artista mencionado, adecuada para acompañar una ficha de obra de arte.
Instrucciones:
1. Resume los aspectos esenciales de la vida y obra del autor: formación, estilo, periodo histórico, aportes y relevancia artística.
2. Mantén un tono natural, informativo y narrativo, sin usar listas ni formato enciclopédico.
3. Evita redundancias, tecnicismos innecesarios y frases genéricas.
4. No incluyas fechas o datos no presentes en el texto fuente.
5. No menciones el proceso de redacción ni expresiones como “esta biografía trata sobre”.
6. Si el texto está en otro idioma, tradúcelo al español de manera natural.
7. Extensión máxima: dos párrafos.
8. Evita completamente el uso de caracteres especiales

Texto: {TEXT}
Tu respuesta:"""

def get_bio(url):
    text = extract_url_context(url)
    completion = client.chat.completions.create(
        model="llama-3.3-70b-versatile",
        messages=[{
            "role": "user",
            "content": [{
                "type": "text",
                "text": PARSE_PROMPT(AUTHOR_BIO_PROMPT, TEXT=text)
            }]
        }],
        temperature=0,
    )
    return completion.choices[0].message.content
```

,

Código 9: Author Bio Generation

Utilidades Generales de Procesamiento de Imágenes

A continuación se definen utilidades de procesamiento de imágenes para el preprocesamiento básico antes de realizar llamadas a proveedores de servicios de inferencia o su uso en modelos autohospedados.

- **`resize_if_oversized`** ajusta imágenes manteniendo proporciones, evitando que excedan un tamaño máximo para optimizar almacenamiento y rendimiento.
- **`download_img`** descarga imágenes desde URLs usando un **user-agent** personalizado (para evitar bloqueos de conexión) y las redimensiona si superan el límite, garantizando su adaptación al sistema.
- **`image_to_base64`** convierte imágenes PIL a base64, facilitando su transmisión y almacenamiento como texto plano para integración en APIs o bases de datos.

```python
def resize_if_oversized(image, max_dim=1024):
    width, height = image.size
    if max(width, height) <= max_dim:
        return image  # nothing to do

    # scale preserving proportions
    scale = max_dim / max(width, height)
    new_size = (int(width * scale), int(height * scale))
    return image.resize(new_size, Image.Resampling.LANCZOS)

def download_img(url, max_dim: int = 1024):
    headers = { "User-Agent": "Chrome/51.0.2704.103 Safari/537.36" }
    r = requests.get(url, headers=headers)
    im = Image.open(BytesIO(r.content))
    im = resize_if_oversized(im, max_dim)
    return im

def image_to_base64(image: Image.Image, format="PNG") -> str:
    """Convert PIL Image to base64 string."""
    buffered = BytesIO()
    image.save(buffered, format=format)
    return base64.b64encode(buffered.getvalue()).decode("utf-8")
```

Código 10: Utilidades de Procesamiento de Imágenes

Narrador de Contexto

Para generar narraciones se usó el módulo de extracción de información (Código 6) junto con un prompt que adapta el texto extraído a un formato adecuado para narración oral.

```python
SOURCE_SUMMARY_PROMPT = \
"""Eres un experto en redacción narrativa.
Tu tarea es leer el siguiente texto y escribir un resumen breve (máximo dos párrafos) en un estilo natural, fluido y adecuado para narración oral o escrita.

Instrucciones:
1. Mantén un tono narrativo, como si estuvieras contando los hechos de manera clara y atractiva.
2. Sé conciso: evita repeticiones, rodeos o frases innecesarias.
3. No incluyas títulos, subtítulos ni listas.
4. No uses expresiones como “en resumen”, “este artículo trata sobre”, ni menciones al proceso de resumen.
5. Si el texto está en otro idioma, tradúcelo al español con naturalidad.
6. Enfócate en los hechos principales y el contexto historico de la producción de la obra.

Texto: {TEXT}
Tu respuesta:"""

def get_narration(url):
    text = extract_url_context(url)
    completion = client.chat.completions.create(
        model="llama-3.3-70b-versatile",
        messages=[{
            "role": "user",
            "content": [{
                "type": "text",
                "text": PARSE_PROMPT(SOURCE_SUMMARY_PROMPT,
                TEXT=text)
            }]
        }],
        temperature=0,
    )
    return completion.choices[0].message.content
```

Código 11: Generación de texto para narraciones

Generación de Audio Descriptivo

Se utilizan utilidades de procesamiento de imagenes definidas en Código 10. Para la generación del texto se emplea un **prompt** estructurado que guía al modelo en la creación de descripciones objetivas del contenido de la obra con cadencia de narración oral.

```python
DESCRIPTION_PROMPT = \
"""Describe los elementos retratados en la imagen.
Considera lo siguiente:
- Debe estar en un formato adecuado para narración.
- Debe ser conciso y corto en duración.
- NO digas en resumen.
- NO seas redundante.
- NO utilices títulos ni subtitulos, solo escribe en parrafos narrados.

Elementos retratados: """

def get_description(url):
    image = download_img(url)
    b64_image = image_to_base64(image)
    completion = client.chat.completions.create(
        model="meta-llama/llama-4-scout-17b-16e-instruct",
        messages=[{
            "role": "user",
            "content": [
                { "type": "text",
                  "text": DESCRIPTION_PROMPT },
                { "type": "image_url",
                  "image_url": {
                    "url": f"{b64_image}"
                  } }
            ]
        }],
        temperature=0,
    )
    return completion.choices[0].message.content
```

Código 12: Generación de texto para audios dscriptivos

Generación de Sonidos Ambientales

La euristica utilizada para la generación de sonidos ambientales consiste en 3 pasos:

1. **Segmentación de la imagen** (Código 13): La obra se divide en cuadrantes mediante un algoritmo de particionamiento espacial.
2. **Extracción semántica de elementos** (Código 14): Mediante Llama 4 Scout 17B se analizan los cuadrantes para detectar objetos, seres vivos y elementos con potencial sonoro. La salida en JSON incluye descripción, clasificación (fondo/primer plano) y ubicación relativa en la cuadrícula.
3. **Síntesis de audio ambiental** (Código 15): Utilizando modelos generativos de audio basados en difusión (AudioLDM), se produce una composición sonora multicanal que integra los elementos detectados. Cada componente sonoro se ajusta en términos de volumen y posición espacial relativa según su ubicación en la cuadrícula original, creando una experiencia auditiva coherente con la distribución visual de la obra.

```python
def get_quadrants(image: Image, N:int=3):
    total_width, total_height = image.size
    w, h = total_width//N, total_height//N
    w_pos = [(w*i, w*(i+1)) for i in range(N)]
    h_pos = [(h*i, h*(i+1)) for i in range(N)]
    boxes = [
        (w_pos[i][0], h_pos[j][0], w_pos[i][1], h_pos[j][1])
        for j in range(N) for i in range(N)
    ]
    cropped = [ image.crop(box)
                for box in boxes ]
    coords= [
      ( (i/N + (i+1)/N)/2,
        (j/N + (j+1)/N)/2 )
      for j in range(N) for i in range(N)
    ]

    return cropped, coords
```

Código 13: Segmentación de imagen en cuadrantes

```python
EXTRACTOR_PROMPT = \
"""Extract audio ambience generation keys from the following image.

Consider the following:
- Your output should be a list of json objects containing the keys:
    {"is_background": bool, "object": string}
- Only indicate objects with sound ambience relevance.
- Ignore abstract elements
- Indicate 3 elements at most
- Group elements when there are more than 1 depicted
- You should only output ONE list
- Alive elements should be added with their corresponding sound description (bear growling, dog barking)
- Non alive elements should be specified as the element itself
- Output just the required JSON results

JSON RESULT:"""

def get_semantic_elements(image: Image, max_attempts=3):
    b64_image = image_to_base64(image)
    for _ in range(max_attempts):
        try:
            completion = client.chat.completions.create(
                model="meta-llama/llama-4-scout-17b-16e-instruct",
                messages=[{
                "role": "user",
                "content": [
                { "type": "text",
                    "text": EXTRACTOR_PROMPT },
                { "type": "image_url",
                    "image_url": {
                    "url": f"{b64_image}" } }
                ]
                }],
                temperature=0,
            )
            content = completion.choices[0].message.content
            return json.loads(content)
        except:
            continue
    return []
```

Código 14: Extracción semántica de elementos en la obra

```python
from diffusers import AudioLDMPipeline
from pydub import AudioSegment
import numpy as np
import tempfile
import os
import torch
VOLUME_DB_BACKGROUND = -20
VOLUME_DB_FOREGROUND = -5

pipe = AudioLDMPipeline.from_pretrained(
    "cvssp/audioldm"
    torch_dtype=torch.float16
).to("cuda")

def tensor_to_audiosegment(audio_tensor, sample_rate=16000):
    samples = (audio_tensor * 32767).astype(np.int16)
    return AudioSegment(
        samples.tobytes(),
        frame_rate=sample_rate,
        sample_width=2,  # 16-bit = 2 bytes
        channels=1
    )

def generate_ambient_sound(scene, filename):
    final_mix = AudioSegment.silent(duration=20000)
    for i, obj in enumerate(scene):
        prompt = obj['object']
        negative_prompt = ["low quality, average quality"]
        result = pipe(
            [prompt],
            negative_prompt=negative_prompt,
            num_inference_steps=30,
            audio_length_in_s=20,
            return_dict=True
        )
        audio_tensor = result.audios[0].squeeze()
        audio_segment = tensor_to_audiosegment(audio_tensor)
        volume_db = VOLUME_DB_BACKGROUND if obj["is_background"] else VOLUME_DB_FOREGROUND
        audio_segment = audio_segment + volume_db
        final_mix = final_mix.overlay(audio_segment)

    final_mix.export(f"{filename}.wav", format="wav")
```

Código 15: Generación de sonido ambiental con AudioLDM

Text to Speech

```python
# !pip install -q kokoro>=0.9.4 soundfile
# !apt-get -qq -y install espeak-ng > /dev/null 2>&1

from kokoro import KPipeline
from IPython.display import display, Audio
import soundfile as sf
import torch

tts_pipeline = KPipeline(lang_code='e')python
```

Código 16: Setup Kokoro TTS

```python
 def tts(text, filename):
    generator = tts_pipeline(
        text,
        voice='em_santa',
        speed=1, split_pattern=r'\n+'
    )

    pause_duration = 0.5
    silence = np.zeros(int(24000 * pause_duration), dtype=np.float32)
    all_audio = []

    for i, (gs, ps, audio) in enumerate(generator):
        all_audio.append(audio)
        all_audio.append(silence)

    final_audio = np.concatenate(all_audio)
    sf.write(f"{filename}.wav", final_audio, 24000)
```

Código 17: Generacón de audio con Kokoro TTS