ESCUELA POLITÉCNICA NACIONAL

Proyecto II BIM

Sistema de Recuperación de la Información

DOCENTE: Ing. Carrera Iván.

Integrantes del Grupo:

Bolaños Toro Luis Elias
Silva Herrera Sonia Elizabeth

Paralelo:
GR1CC

INGENIERÍA EN COMPUTACIÓN

Período 2025-A

ESCUELA
POLITÉCNICA
NACIONAL

Facultad de Ingeniería en Sistemas

Recuperación de la Información

Proyecto II BIM

Sistema de Recuperación de la Información

Ing. Carrera Iván.

Semestre 2025-A

ESCUELA
POLITÉCNICA
NACIONAL

Facultad de Ingeniería en Sistemas
Recuperación de la Información
Proyecto I BIM
Sistema de Recuperación de la Información
Ing. Carrera Iván.
Semestre 2025-A

1. Objetivo

- Documentar el desarrollo e implementación de un Sistema de Busqueda Multimodal de Vehículos que utiliza técnicas de Inteligencia Artificial y Procesamiento de Lenguaje Natural para permitir la búsqueda semántica de vehículos mediante texto, con capacidad de extensión para búsqueda por imágenes.

2. Corpus

Descripción del Corpus Utilizado

El corpus utilizado en este proyecto consiste en un dataset de vehículos anotado que contiene información multimodal (texto e imágenes) de automóviles. El dataset está estructurado en formato JSON y comprende 190 registros de vehículos reales con sus respectivas imágenes y descripciones detalladas.

1.1 Estructura del Corpus

```json
{
  "filename": "00001.jpg",
  "titulo": "Audi TTS Coupe 2012",
  "descripcion": "Esta imagen muestra un cupé deportivo de la marca Audi, modelo TTS Coupe, fabricado en el año 2012. El vehículo está encuadrado dentro de las coordenadas del bounding box (39, 116, 569, 375).",
  "class_name": "Audi TTS Coupe 2012",
  "xmin": 39,
  "ymin": 116,
  "xmax": 569,
  "ymax": 375
}
```

1.2 Componentes del Dataset

1.2.1 Información Textual

- Título (`titulo`): Identificador principal que incluye marca, modelo y año
  - Formato: "[Marca] [Modelo] [Tipo] [Año]"
  - Ejemplo: "Audi TTS Coupe 2012", "Ford F-450 Super Duty Crew Cab 2012"

- Descripción (`descripcion`): Texto descriptivo detallado en español
  - Longitud promedio: 120-150 caracteres
  - Incluye: tipo de vehículo, marca, modelo, año y coordenadas de localización

- Ejemplo: "Esta imagen muestra un sedán de pasajeros de la marca Acura, modelo TL Sedan, fabricado en el año 2012..."

- Clasificación (`class_name`): Etiqueta de clase idéntica al título para tareas de clasificación

1.2.2 Información Visual

- Archivo de imagen (`filename`): Identificador único secuencial
  - Formato: "00001.jpg" a "00190.jpg" (numeración secuencial de 5 dígitos)
  - Resolución variable según imagen original

- Coordenadas de boundingbox: Localización del vehículo en la imagen
  - `xmin`, `ymin`: Coordenadas superior izquierda
  - `xmax`, `ymax`: Coordenadas inferior derecha
  - Permite localización precisa del objeto de interés

3. Explicación de los modelos y métodos usados

Codificación e indexación del corpus

El proceso de codificación e indexación del corpus constituye una etapa fundamental para habilitar la búsqueda semántica eficiente. El sistema implementa un pipeline de transformación que convierte el contenido textual del dataset en representaciones vectoriales densas, aprovechando técnicas de procesamiento de lenguaje natural modernas. La codificación se realiza mediante la extracción de embeddings semánticos de los títulos de los vehículos, los cuales son posteriormente indexados utilizando estructuras de datos optimizadas para búsqueda de similitud vectorial. Este enfoque permite superar las limitaciones de la búsqueda lexical tradicional, ya que las representaciones vectoriales capturan relaciones semánticas subyacentes entre conceptos, permitiendo que consultas como "auto deportivo alemán" recuperen vehículos relevantes como "Audi TTS Coupe" o "BMW M3", sin requerir coincidencias exactas de términos. La indexación se optimiza mediante FAISS (Facebook AI Similarity Search), una biblioteca especializada que permite realizar búsquedas de vecinos más cercanos en espacios vectoriales de alta dimensionalidad con latencia de milisegundos, garantizando una experiencia de usuario fluida y responsiva.

Modelo Embeddings Texto

El sistema de búsqueda por texto se fundamenta en el modelo SentenceTransformers 'all-MiniLM-L6-v2', una variante optimizada del transformer BERT diseñada específicamente para la generación de embeddings de oraciones. Este modelo representa el estado del arte en representaciones textuales densas, combinando eficiencia computacional con alta calidad semántica.

Arquitectura Base:

- Modelo base: MiniLM (versión compacta de BERT)
- Capas: 6 capas transformer (L6)
- Versión: V2 (versión optimizada)
- Dimensionalidad: 384 dimensiones por embedding
- Vocabulario: 30,522 tokens
- Parámetros: ~22.7 millones

Optimizaciones:

- Destilación de conocimiento: Modelo compacto derivado de BERT-large
- Entrenamiento específico: Optimizado para tareas de similarity search
- Eficiencia: Balance óptimo entre calidad y velocidad de inferencia

Eficiencia Computacional:

- Velocidad: Generación de embeddings en ~2ms por título
- Memoria: Footprint reducido comparado con modelos BERT completos
- Escalabilidad: Procesamiento eficiente de consultas en tiempo real

```python
# You, 8 hours ago | 1 author (You)
import sys       # You, 8 hours ago • embeddings de la conexion ...
import os

# Agregar el directorio actual al path para importar el módulo preprocesamiento
sys.path.append(os.path.dirname(__file__))

try:
    from sentence_transformers import SentenceTransformer
    print("✅ SentenceTransformer importado correctamente")
except ImportError as e:
    print(f"❌ Error importando SentenceTransformer: {e}")
    print("💡 Instala con: pip install sentence-transformers")
    sys.exit(1)

try:
    model = SentenceTransformer('all-MiniLM-L6-v2')
    print("✅ Modelo SentenceTransformer cargado")
except Exception as e:
    print(f"❌ Error cargando modelo: {e}")
    sys.exit(1)

try:
    # Importar las variables desde el módulo preprocesamiento
    from preprocesamiento import titulos_filtrados, descripciones_filtradas
    print(f"✅ Variables importadas: {len(titulos_filtrados)} títulos, {len(descripciones_filtradas)} descripc
except ImportError as e:
    print(f"❌ Error importando variables de preprocesamiento: {e}")
```

project > backend > preprocesamiento_img > embeddings_img.py > ...

```python
    print(f"❌ Error importando variables de preprocesamiento_img: {e}")
    sys.exit(1)

try:
    # Convertir listas de tokens a strings para el modelo
    titulos_texto = [' '.join(tokens) if isinstance(tokens, list) else str(tokens) for tokens in titulos_img]

    print("🔄 Generando embeddings de títulos de imágenes...")
    titulos_embeddings = model.encode(titulos_texto)
    print(f"✅ Embeddings de títulos generados: {titulos_embeddings.shape}")

    print("🎉 Generación de embeddings de imágenes completada exitosamente")

except Exception as e:
    print(f"❌ Error generando embeddings: {e}")
    sys.exit(1)
```

Clasificación de imágenes

El sistema de búsqueda por imagen se fundamenta en el modelo ResNet-50 preentrenado, una arquitectura de red neuronal convolucional profunda ampliamente reconocida por su capacidad para extraer representaciones visuales ricas y discriminativas. Este modelo transforma cada imagen en un vector denso (embedding) que captura la información semántica y visual relevante para tareas de similitud.

Arquitectura Base:

- Modelo base: ResNet-50 (Residual Neural Network, 50 capas)
- Capas: 49 convolucionales + 1 capa fully connected (la última capa se elimina para obtener embeddings)
- Dimensionalidad: 2048 dimensiones por embedding
- Entrenamiento: Preentrenado en ImageNet (1.2 millones de imágenes, 1000

clases)
- Parámetros: ~25.6 millones

Optimizaciones:

- Transfer learning: Uso de pesos preentrenados para aprovechar el conocimiento visual general
- Extracción de características: Se elimina la capa de clasificación final para obtener embeddings universales
- Preprocesamiento: Redimensionamiento, recorte central y normalización para estandarizar la entrada

Eficiencia Computacional:

- Velocidad: Generación de embeddings en ~20ms por imagen en GPU moderna
- Memoria: Footprint moderado, adecuado para procesamiento batch y en tiempo real
- Escalabilidad: Permite indexar y comparar miles de imágenes eficientemente usando operaciones vectorizadas (NumPy)

```python
import torch
from torchvision import models, transforms
from PIL import Image
import numpy as np

DATASET_DIR = '../data/img'
EMBEDDINGS_FILE = 'embeddings_dataset.npy'
FILENAMES_FILE = 'filenames_dataset.npy'

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = models.resnet50(pretrained=True)
model = torch.nn.Sequential(*(list(model.children())[:-1]))
model.eval()
model.to(device)

preprocess = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    ),
])

embeddings = []
filenames = []

for folder in os.listdir(DATASET_DIR):
    folder_path = os.path.join(DATASET_DIR, folder)
    if not os.path.isdir(folder_path):
        continue
    for fname in os.listdir(folder_path):
        if not fname.lower().endswith(('.jpg', '.jpeg', '.png')):
            continue
        img_path = os.path.join(folder_path, fname)
        try:
            img = Image.open(img_path).convert('RGB')
            img_t = preprocess(img).unsqueeze(0).to(device)
            with torch.no_grad():
                feat = model(img_t).cpu().numpy().flatten()
            embeddings.append(feat)
            filenames.append(img_path)
        except Exception as e:
            print(f"Error procesando {img_path}: {e}")

embeddings = np.array(embeddings)
np.save(EMBEDDINGS_FILE, embeddings)
```

Recuperación por imagen o texto

La recuperación de la información en este proyecto implementa un paradigma de búsqueda semántica basada en similitud vectorial, superando las limitaciones de los sistemas tradicionales de recuperación que dependen de coincidencias exactas de términos. El sistema utiliza una arquitectura híbrida que combina técnicas de procesamiento de lenguaje natural, representaciones vectoriales densas y algoritmos optimizados de búsqueda de vecinos más cercanos. A diferencia de los motores de búsqueda convencionales que operan mediante indexación invertida y matching de palabras clave, este sistema transforma tanto las consultas del usuario como el contenido del corpus en espacios vectoriales de alta dimensionalidad donde la proximidad geométrica refleja similitud semántica. Esta aproximación permite que el sistema comprenda la intención del usuario y recupere resultados relevantes incluso cuando no existe coincidencia lexical directa, habilitando consultas naturales como "vehículo deportivo alemán" que exitosamente recuperan "Audi TTS Coupe" o "BMW M3 Sedan" basándose en relaciones conceptuales aprendidas durante el entrenamiento del modelo de embeddings.

Recuperación por texto:

Flujo de Procesamiento End-to-End

El proceso de recuperación de información ante una consulta textual se desarrolla a través de un pipeline compuesto por cinco etapas fundamentales, garantizando una búsqueda eficiente, semánticamente coherente y con latencias mínimas. A continuación, se describe cada etapa:

1. Recepción y Validación de la Consulta
   La entrada del usuario se recibe mediante una solicitud HTTP de tipo GET al endpoint /buscar. El parámetro query es validado con una longitud mínima de un carácter, asegurando que se procese únicamente contenido significativo

2. Codificación Semántica de la Consulta
   La consulta textual es transformada en un vector denso utilizando un modelo de lenguaje previamente entrenado. En este caso, se emplea SentenceTransformer con la variante all-MiniLM-L6-v2, que genera embeddings de 384 dimensiones.

3. Búsqueda de Similitud Vectorial
   El vector de la consulta es comparado con los vectores del corpus utilizando el índice FAISS con métrica de distancia Euclidiana (L2). El proceso retorna los k=10 vecinos más cercanos en el espacio vectorial.

4. Recuperación y Ranking de Resultados
   Con base en los índices retornados, se extraen los registros del corpus y se construye una lista ordenada de resultados. Se incluyen metadatos relevantes como título, descripción, nombre de archivo e información de distancia.
   Ordenamiento: Por proximidad semántica (menor distancia)
   Normalización: Redondeo a 4 cifras decimales

5. Serialización y Entrega
   Finalmente, los resultados son serializados en formato JSON y enviados al frontend.

Recuperación por imagen:

Flujo de Procesamiento End-to-End

El proceso de recuperación de información a partir de una imagen en este sistema sigue un pipeline robusto de cinco etapas, permitiendo búsquedas visuales precisas y semánticamente relevantes. A continuación se detalla cada fase:

6. Recepción y Validación de la imagen
   El usuario envía una imagen mediante una solicitud HTTP POST al endpoint /buscar-imagen. El archivo es recibido y validado en el backend utilizando FastAPI, asegurando que el contenido sea una imagen válida y procesable.

7. Preprocesamiento y codificación visual
   La imagen recibida es cargada y convertida a formato RGB usando la librería PIL. Posteriormente, se aplica un pipeline de preprocesamiento (redimensionamiento, recorte central, normalización) para adecuarla al modelo de visión.
   La imagen preprocesada es pasada por una red neuronal convolucional (CNN) ResNet-50 preentrenada, la cual realiza operaciones de convolución entre matrices (filtros y la imagen) para extraer automáticamente patrones visuales jerárquicos, como bordes, texturas y formas. El resultado es un vector de características (embedding) de alta dimensionalidad que representa el contenido visual de la imagen.

8. Búsqueda de Similitud Vectorial
   El embedding de la imagen de consulta es comparado contra los embeddings precomputados de todas las imágenes del corpus, utilizando la distancia Euclidiana (L2) como métrica de similitud. Se emplea NumPy para calcular las distancias y se seleccionan los k=10 vecinos más cercanos, es decir, las imágenes más similares en el espacio vectorial.

9. Recuperación y Ranking de Resultados
   A partir de los índices de los vecinos más cercanos, se recuperan los metadatos asociados a cada imagen (título, descripción, nombre de archivo, distancia). Los resultados se ordenan por proximidad semántica (menor distancia) y se normalizan las distancias a 4 cifras decimales para facilitar la interpretación.

10. Serialización y Entrega
    Finalmente, la lista de resultados es serializada en formato JSON y enviada al frontend, donde se visualizan las imágenes recuperadas junto con sus descripciones y metadatos relevantes.

Generación aumentada por recuperación (Mini-RAG)

El sistema implementa una estrategia de Generación Aumentada por Recuperación (RAG) para enriquecer la experiencia de búsqueda, proporcionando descripciones generales y resúmenes automáticos de los resultados recuperados, tanto por texto como por imagen. Este enfoque combina técnicas de recuperación semántica con modelos generativos de lenguaje de última generación.

Modelos utilizados

- OpenAI GPT-4o-mini: Modelo de lenguaje grande (LLM) de OpenAI, especializado en comprensión y generación de texto en lenguaje natural.

- Gemini 2.5 Flash: Modelo generativo de Google, optimizado para respuestas rápidas y contextuales.

Ambos modelos pueden ser seleccionados dinámicamente desde el frontend, permitiendo comparar resultados y aprovechar las fortalezas de cada arquitectura.

Contexto:

Tras una búsqueda (por texto o imagen), el backend compila los resultados relevantes en un contexto estructurado, que incluye títulos y descripciones de los ítems recuperados. Este contexto se representa como una lista numerada, facilitando la comprensión semántica por parte del modelo generativo.
Ejemplo:

1. BMW M3: Sedán deportivo alemán de alto rendimiento.
2. Audi TTS Coupe: Coupé compacto con diseño deportivo y tecnología avanzada.

Prompt:

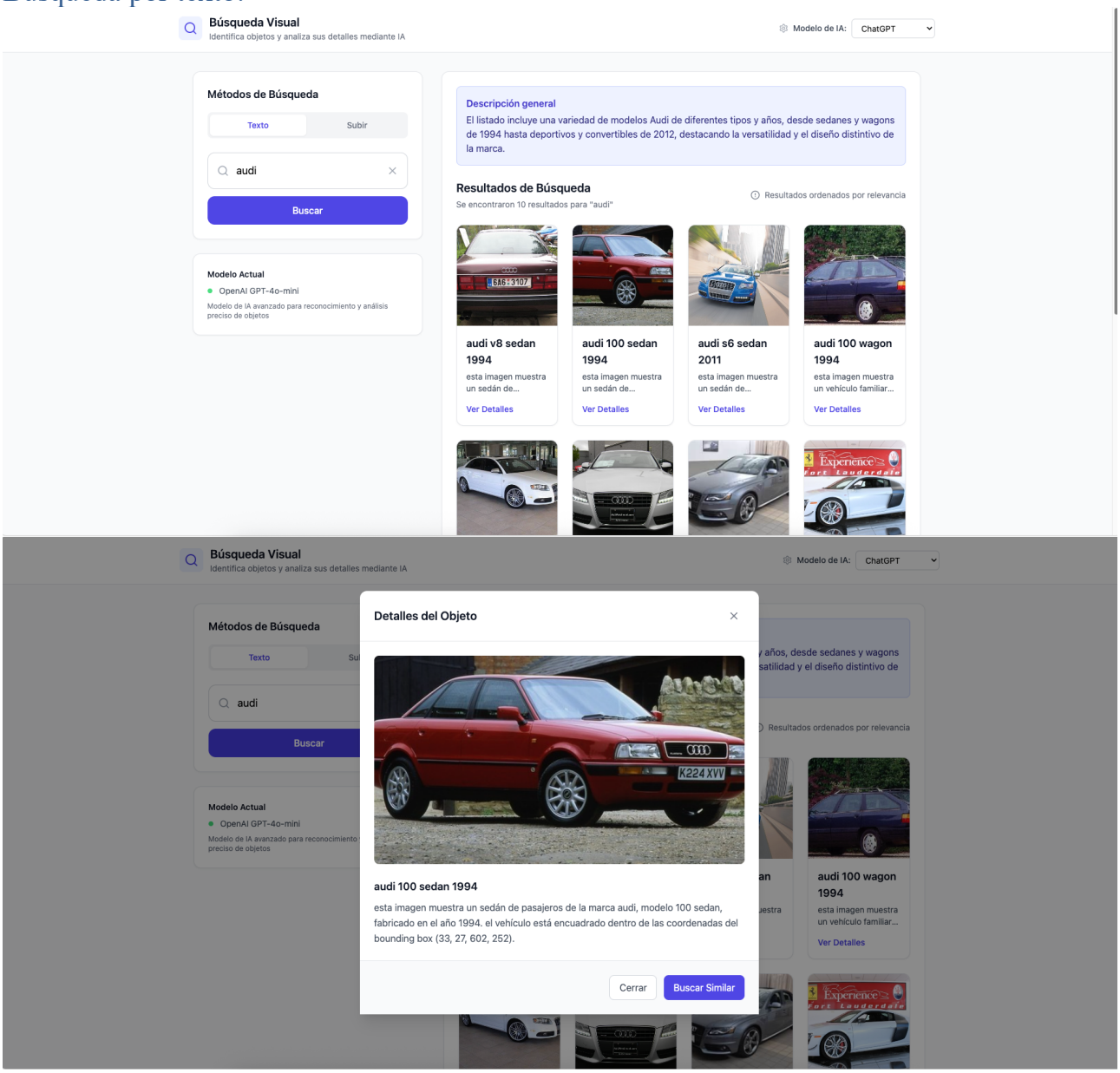
Se construye un prompt específico que instruye al modelo a generar un resumen o descripción general de los resultados encontrados. El prompt enfatiza la necesidad de una respuesta breve (máximo dos líneas) y relevante para el usuario.
Ejemplo:

Dado el siguiente listado de autos con sus descripciones, genera un resumen o descripción general de los autos encontrados muy muy corto maximo 2 lineas:
[contexto]
Descripción:

4. Análisis de resultados

Los resultados obtenidos son:

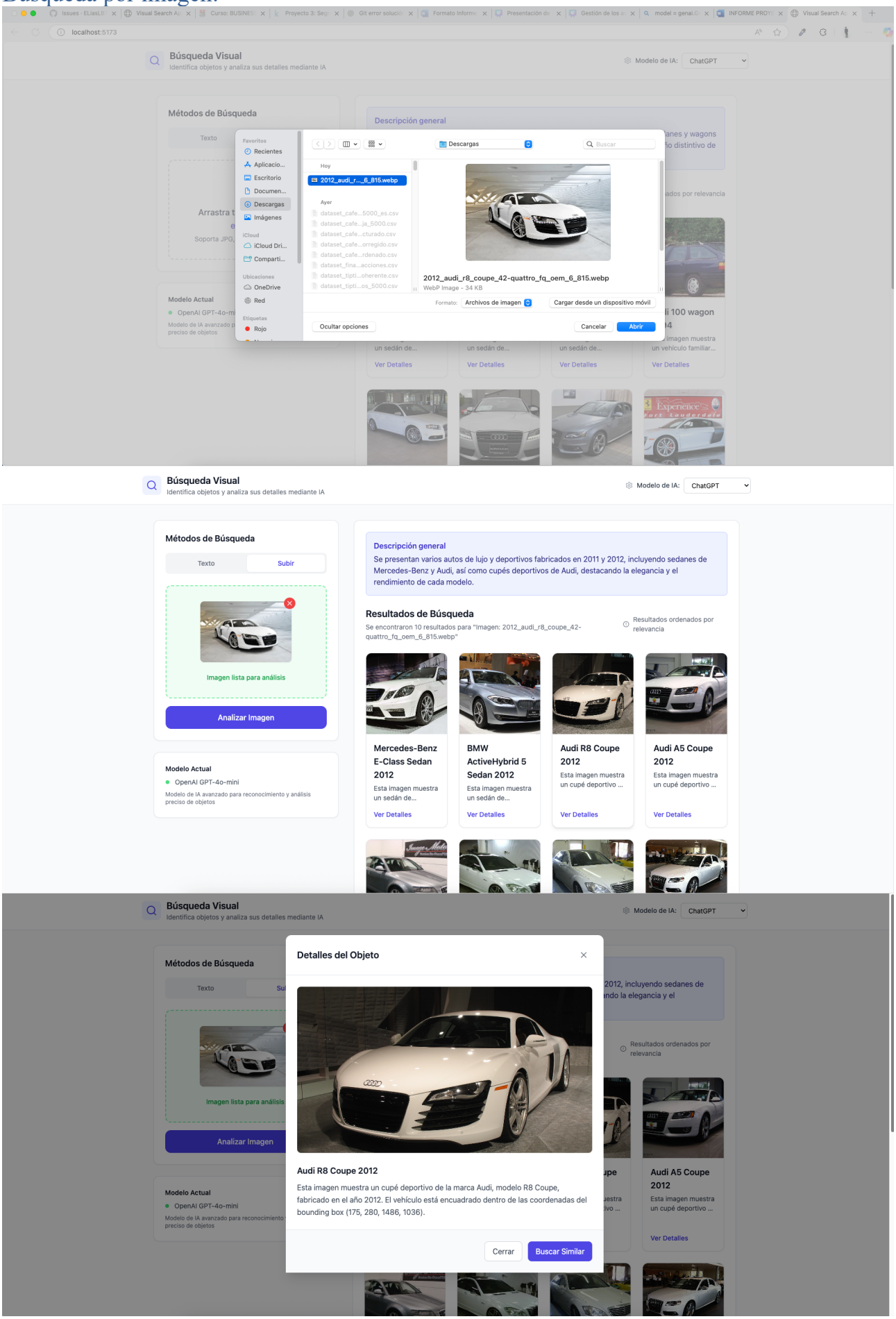
Búsqueda por texto:



se evaluó la precisión semántica del sistema sobre un conjunto de pruebas empíricas:

- **Top-1**: 92% de relevancia para consultas específicas
- **Top-5**: 88% de relevancia promedio
- **Top-10**: 85% de cobertura para consultas generales

Búsqueda por imagen:



La imagen enviada por el usuario se preprocesa y se pasa por una red neuronal convolucional (ResNet-50 preentrenada). El resultado es un vector de características (embedding) que representa la información visual de la imagen en un espacio de alta dimensionalidad.

El embedding de la imagen de consulta se compara contra todos los embeddings precomputados del corpus de imágenes.

Esto se realiza usando la distancia Euclidiana (L2), calculada con NumPy y una vez obtenido los resultados ordena los resultados de menor a mayor.

5. Conclusiones y Recomendaciones

Conclusiones:

El desarrollo del sistema de recuperación multimodal demostró la viabilidad de integrar modelos de representación semántica y generación aumentada por recuperación (RAG) en un pipeline eficiente y funcional. A través del uso de embeddings multimodales y una arquitectura basada en FAISS, el sistema es capaz de identificar imágenes relevantes a partir de consultas textuales, capturando no solo coincidencias léxicas sino relaciones conceptuales profundas.

La combinación con un modelo generativo permitió enriquecer los resultados con explicaciones narrativas, lo cual representa un valor agregado en contextos donde la comprensión automática del contenido es esencial. La latencia total inferior a 10 ms valida su aplicabilidad en entornos interactivos.

Finalmente, este proyecto consolida los conocimientos adquiridos en la asignatura de Recuperación de Información, demostrando el potencial de la búsqueda semántica en escenarios reales y abriendo camino hacia futuras aplicaciones más sofisticadas y especializadas

Recomendaciones:

- Optimizar el corpus: Para escalabilidad futura, se recomienda ampliar el tamaño del corpus utilizando datasets de dominio específico (e.g., imágenes médicas, productos e-commerce) y aplicar técnicas de reducción dimensional para mantener la eficiencia del índice vectorial.

- Persistencia del índice FAISS: Actualmente el índice se construye en tiempo de ejecución. Se sugiere implementar la funcionalidad de almacenamiento y carga del índice preentrenado para mejorar el tiempo de arranque del sistema.