Universidad Zaragoza
1542

Trabajo Fin de Máster

Generación de piezas audiovisuales a través de agentes autónomos

Generation of audiovisual pieces through autonomous agents

Autora

María García Cutando

Director

Eduardo Lleida Solano

Titulación de la autora

Máster Universitario en Ingeniería de Telecomunicación

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2024

AGRADECIMIENTOS

En primer lugar, deseo expresar mi más sincero agradecimiento a mi tutor, Eduardo, por su constante orientación y apoyo durante todo el proceso de elaboración de este Trabajo Fin de Máster. Gracias a su guía, he podido ampliar mis conocimientos y explorar diversos enfoques, logrando así cumplir con los objetivos propuestos.

También quiero agradecer a mis compañeros y amigos de la universidad, cuyas valiosas sugerencias y consejos han contribuido significativamente a mejorar la calidad de este trabajo. Su disposición para colaborar, tanto en el ámbito académico como en el personal, ha sido fundamental, brindándome un entorno de apoyo y complicidad.

Extiendo mi gratitud al grupo de investigación *ViVoLab*, por proporcionarme los recursos y el entorno adecuado para desarrollar este proyecto. El ambiente de trabajo y el compañerismo que experimenté en el grupo me facilitaron el progreso, permitiéndome enfrentar los desafíos académicos y profesionales con entusiasmo y dedicación. Agradezco también a RTVE por proporcionar una de las bases de datos esenciales para la implementación y evaluación del sistema, lo que fue clave para la ejecución exitosa del mismo.

A mi familia y amigos, quiero darles las gracias por su comprensión, motivación y apoyo incondicional a lo largo de este periodo. Su cercanía ha sido esencial para superar los momentos difíciles y me ha impulsado a continuar hasta alcanzar mis metas.

Finalmente, agradezco a todas aquellas personas que, de una manera u otra, han contribuido al éxito de este trabajo. Sus aportes, aunque no siempre visibles, han sido fundamentales para su realización.

RESUMEN

Desde la antigüedad, la humanidad ha reconocido la importancia de archivar y organizar la información. Este reconocimiento ha impulsado una evolución desde repositorios simples hasta avanzados sistemas digitales capaces de gestionar grandes volúmenes de datos, incluidos textos y contenido multimedia.

Con la llegada de los computadores, la necesidad de métodos eficientes para el almacenamiento masivo y la búsqueda y recuperación de información se hizo aún más evidente. En el siglo XXI, los avances tecnológicos han revolucionado la gestión del contenido audiovisual y la preservación del patrimonio cultural. Sin embargo, también han generado desafíos, como la sobrecarga de información y la creciente demanda de calidad por parte de los usuarios, lo que ha hecho necesario implementar estrategias de búsqueda que aprovechen las tecnologías emergentes.

Aunque los sistemas tradicionales de búsqueda basados en palabras clave han sido fundamentales, presentan limitaciones significativas al interpretar el lenguaje natural y el contexto de las consultas, lo que puede llevar a resultados imprecisos. La búsqueda semántica ha surgido como una solución clave, permitiendo un análisis profundo del significado del contenido y proporcionando resultados contextualmente relevantes. Los agentes autónomos, basados en grandes modelos de lenguaje, han mejorado significativamente la comprensión de las consultas, aumentando así la eficiencia en la recuperación de información.

Este trabajo propone una plataforma unificada que integra estas tecnologías para facilitar la navegación en colecciones de vídeo. La solución incluye la unificación del contenido textual y visual en un espacio vectorial común, optimizando la indexación y segmentación semántica mediante técnicas avanzadas. Además, se implementa un agente autónomo capaz de analizar guiones y coordinar herramientas para realizar dichas búsquedas.

En resumen, el propósito principal es desarrollar una aplicación que asista en la ideación periodística, facilitando a los usuarios la exploración de recursos audiovisuales mediante el uso de guiones. La efectividad de este sistema se ha evaluado utilizando dos bases de datos reconocidas para garantizar la relevancia de los resultados y la validez de las conclusiones.

Índice

Palabras Clave

1. ***CLIP**: Contrastive Language-Image Pre-Training*
2. ***CoT**: Chain of Thought*
3. ***HDBSCAN**: Hierarchical Density-Based Spatial Clustering of Applications with Noise*
4. ***HNSW**: Hierarchical Navigable Small World*
5. ***IR**: Information Retrieval*
6. ***KTS**: Kernel Temporal Segmentation*
7. ***LLM**: Large Language Model*
8. ***MA**: Mean Average*
9. ***MSR-VTT**: Microsoft Research Video to Text*
10. **RTVE**: Radio Televisión Española
11. ***TA**: Text Aggregation*
12. ***UMAP**: Uniform Manifold Approximation and Projection*
13. ***ViT**: Vision Transformer*

Capítulo 1

Introducción

1.1. Contextualización

A lo largo del tiempo, la gestión de datos ha evolucionado significativamente, pasando de registros en papel a vastos repositorios digitales que incluyen tanto textos como contenido multimedia. La humanidad ha reconocido la importancia de archivar y localizar información desde tiempos inmemoriales. Con la llegada de las computadoras, el almacenamiento masivo de datos se facilitó, lo que hizo crucial desarrollar métodos eficientes para buscar dentro de estos grandes volúmenes de datos [1].

Para comprender mejor esta evolución, es fundamental explorar la historia de la recuperación de información. Sus raíces se remontan a la antigüedad, cuando la función organizativa se percibía como un avance significativo. Las bibliotecas, en consecuencia, se transformaron de simples depósitos a centros de catalogación e indexación.

A finales de la década de 1880, *Herman Hollerith* desarrolló la tabuladora de tarjetas perforadas [2], lo cual facilitó el procesamiento del censo de los Estados Unidos en 1890. Posteriormente, en 1931, *Emanuel Goldberg* presentó la "Máquina Estadística" [3], el primer dispositivo electromecánico capaz de recuperar documentos microfilmados utilizando metadatos. En 1945, *Vannevar Bush* publicó "*As We May Think*" [4], donde proponía el "*Memex*", un dispositivo hipotético diseñado para interconectar máquinas de búsqueda y almacenamiento. Este concepto anticipó características de las modernas bases de datos y la hipertextualidad en la *web*. En 1950, *Calvin Mooers* acuñó el término "*information retrieval*" (*IR*) [5], formalizándolo como un campo de estudio.

Años más tarde, en 1957, *H.P. Luhn* propuso el uso de palabras como unidades de indexación y la medición del solapamiento de palabras para la recuperación de documentos [6]. Durante la década de 1960, *Cyril W. Cleverdon*, a través de sus experimentos de *Cranfield* [7], estableció principios y métricas de evaluación fundamentales, como la precisión y la exhaustividad (*precision and recall*), para medir la efectividad de los sistemas de *IR*. *Gerard Salton*, junto con sus estudiantes de

las universidades de *Harvard* y *Cornell*, desarrollaron el sistema *SMART* [8], lo que permitió probar nuevas metodologías para mejorar la calidad de la búsqueda en un entorno controlado. De este trabajo surgieron conceptos clave como el modelo de espacio vectorial (*VSM*) [9].

En la década de 1970, *S.E. Robertson* introdujo el "principio de clasificación probabilística" [10], optimizando la clasificación de documentos según su probabilidad de relevancia. Simultáneamente, *Cornelis J. van Rijsbergen* formuló la "hipótesis de agrupación" [11], sugiriendo que los documentos agrupados por temas similares tienen relevancia conjunta.

A finales de los años 80 y principios de los 90, ocurrieron avances cruciales en el campo de la recuperación de información. En 1989, *Tim Berners-Lee* presentó la *World Wide Web* en el *CERN* [12], revolucionando el acceso a la información. En 1990, se introdujo el *Latent Semantic Indexing* (*LSI*) [13], mejorando la comprensión semántica en la recuperación de información. En 1992, la primera conferencia *TREC* [14] evaluó metodologías de recuperación de texto, impulsando la investigación en *IR* a gran escala. A finales de los 90, los motores de búsqueda *web* integraron características avanzadas, optimizando la gestión de los datos.

En el siglo XXI, la *IR* ha avanzado con la introducción de métodos de aprendizaje automático y procesamiento del lenguaje natural. Estas tecnologías han mejorado la capacidad de los sistemas para manejar grandes volúmenes de datos, proporcionar resultados más precisos y relevantes, y realizar búsquedas multimodales [15].

La evolución de los sistemas de recuperación de información ha sido esencial para el desarrollo del término "periodismo digital", que caracteriza la comunicación informativa que aprovecha tecnologías emergentes, abarcando plataformas en línea, televisión y radio digital [16]. Estos sistemas permiten a los periodistas acceder rápidamente a grandes volúmenes de datos, facilitando la creación de contenido más profundo y bien documentado. Esta unión ha transformado la manera en que la sociedad se informa y consume noticias, adaptándose a las nuevas tendencias de consumo y distribución de información. Según el Estudio General de Medios (EGM) de la AIMC (Asociación para la Investigación de Medios de Comunicación), la segunda ola de 2024 revela que el 89.1 % de la población encuestada en España utiliza *Internet* (véase figura 1.1), subrayando su función central como instrumento de búsqueda para información, comunicación y entretenimiento (véase figura 1.2).

En resumen, la historia de la gestión y recuperación de información revela una evolución constante, adaptándose a las demandas cambiantes de cada época y aprovechando las innovaciones tecnológicas para satisfacer esta necesidad. Esto ha beneficiado a todo tipo de usuarios y ha desempeñado un papel crucial en la era digital.

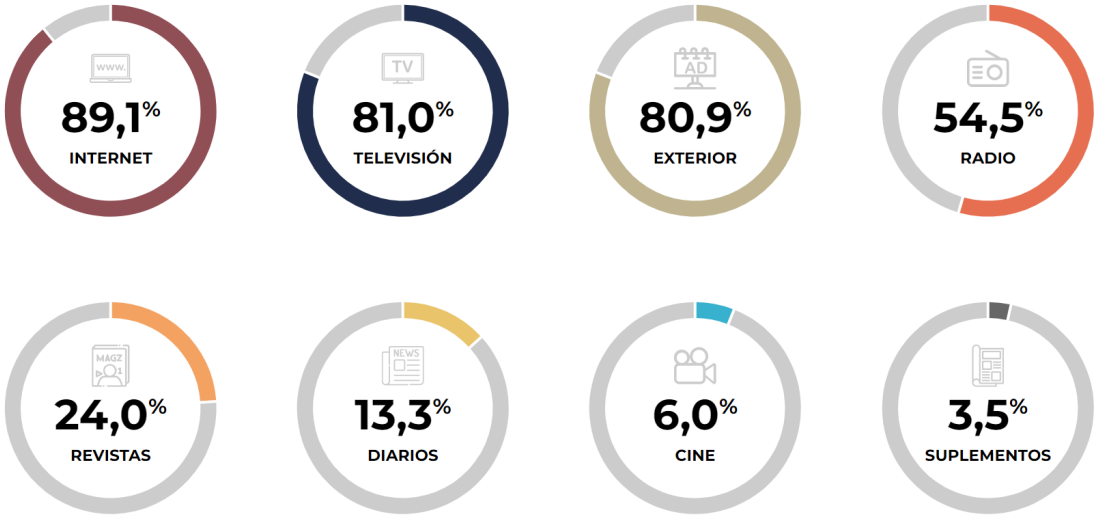


Figura 1.1: Audiencia general de medios, 2ª ola de 2024.[1]

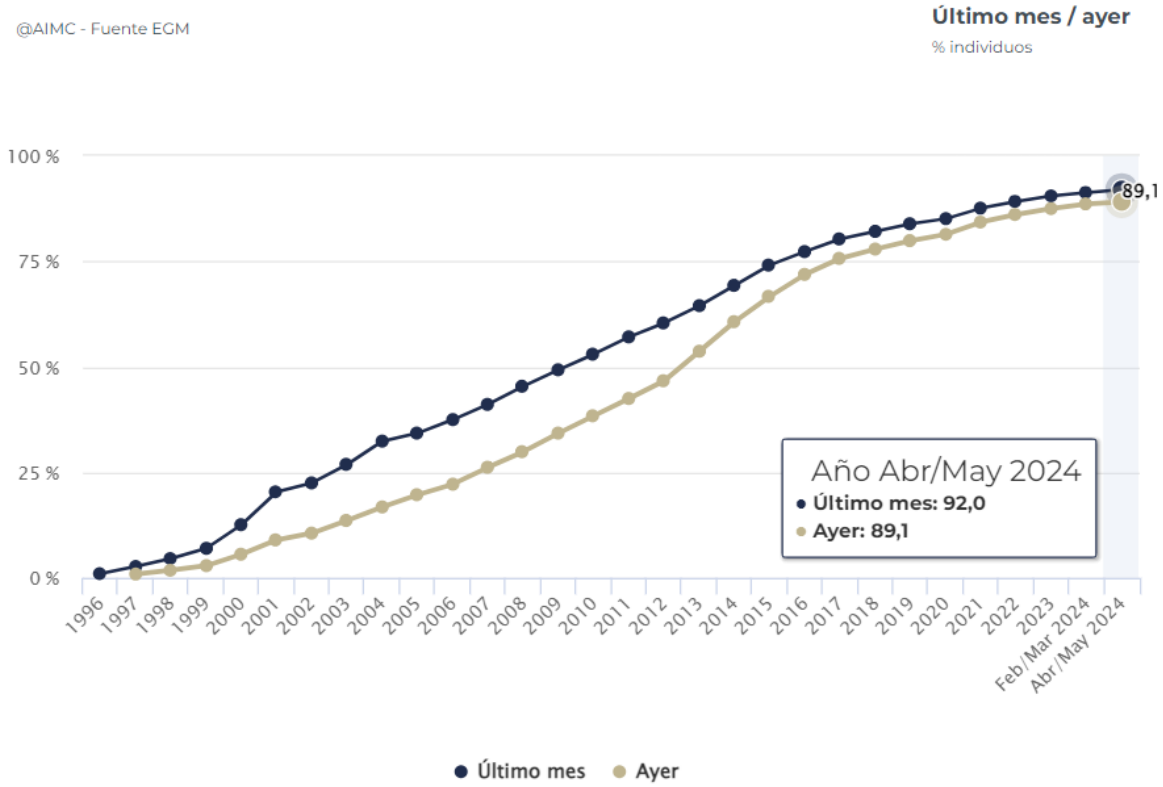


Figura 1.2: Evolución del uso de *Internet*.[2]

[1]Fuente: AIMC. Datos EGM: Resumen General. Disponible en: https://www.aimc.es/egm/datos-egm-resumen-general

[2]Fuente: AIMC. Audiencia de *Internet* en el EGM. Disponible en: https://www.aimc.es/egm/audiencia-internet-egm

1.2. Motivación y Objetivos

Amaya Noain, en su artículo de 2022 titulado "*Addressing the Impact of Artificial Intelligence on Journalism: the perception of experts, journalists and academics*" [17], analiza cómo las tecnologías emergentes están transformando el periodismo. Toni Vilalta, responsable de producto en *VSN* (*Video Stream Networks*), destaca la capacidad de los algoritmos para procesar grandes volúmenes de datos y facilitar la creación de contenidos periodísticos mediante secuencias automáticas e imágenes pertinentes. Del mismo modo, Miguel Rodríguez, periodista de datos en *Press Association* y *Urbs Media*, enfatiza que la automatización puede liberar a los periodistas de tareas repetitivas. En conjunto, estas opiniones subrayan la creciente necesidad de herramientas avanzadas en el ámbito periodístico.

El progreso tecnológico, junto con la digitalización del contenido audiovisual histórico, ha mejorado significativamente el acceso a archivos y ha fomentado la producción de contenido multimedia, consolidando un patrimonio audiovisual de gran importancia [18]. Estos avances permiten prevenir la degradación del material, reutilizar recursos, gestionarlos eficientemente mediante bases de datos, reducir costos de producción y democratizar el acceso a la creación audiovisual. No obstante, persisten desafíos como la sobrecarga de información y las crecientes expectativas de calidad [19], lo que exige estrategias efectivas para captar la atención de la audiencia.

Para abordar estos desafíos y evaluar las oportunidades y posibilidades que ofrecen las nuevas tecnologías, es esencial identificar y analizar las necesidades y tareas clave que facilitan el proceso de ideación de piezas audiovisuales.

La creación de una obra audiovisual a partir de un guion, que en su fase inicial se presenta en formato textual, requiere acceso a una diversidad de vídeos que puedan servir como referencia. Aunque los vídeos están compuestos por secuencias de imágenes y pistas de audio, este trabajo se centra exclusivamente en el análisis de la narrativa visual, dejando de lado el componente auditivo.

El uso de extensas bibliotecas de archivos audiovisuales representa una tarea laboriosa y costosa, por lo que resulta imprescindible emplear un modelo de representación capaz de capturar el contenido visual y vincularlo con la información textual del guion. Este modelo debe permitir que tanto las imágenes como el texto se integren en un mismo dominio. Para construir este sistema, es fundamental capturar y almacenar únicamente la información más relevante de los vídeos, facilitando así la gestión de grandes volúmenes de datos.

En este contexto, dado el enfoque en la problemática y comprensión de los vídeos a partir de su representación, se propone implementar un método de segmentación orientado a comprimir las características de los fotogramas tanto a nivel semántico como temporal, con el fin de indexarlas en una base de datos, optimizando así tanto el almacenamiento como el proceso de búsqueda posterior.

Una vez indexados los recursos, es crucial desarrollar un sistema de búsqueda eficiente que permita localizar aquellos que estén alineados con la temática de la nueva producción. Para optimizar el tiempo del usuario y evitar que tenga que realizar búsquedas manuales para cada elemento del guion, se plantea la automatización de los procesos de búsqueda, permitiendo que el sistema comprenda el guion en su totalidad e interprete todas las consultas en una sola interacción.

Para seleccionar las tecnologías adecuadas para este proyecto, es preciso estudiar las limitaciones presentes en los sistemas tradicionales. Por un lado, los usuarios tienden a formular consultas a los motores de búsqueda utilizando palabras clave que consideran adecuadas para obtener la información deseada [20]. Los motores de búsqueda, a su vez, emplean estas consultas para explorar sus bases de datos. Sin embargo, este método presenta tres limitaciones críticas: la dificultad del usuario para definir con precisión sus objetivos mediante palabras clave, la incapacidad de los motores para interpretar el lenguaje natural debido a la complejidad derivada de la abundancia de sinónimos y términos polisémicos [21], y el hecho de que para realizar búsquedas textuales, es necesario, metadatar los vídeos añadiendo descripciones en texto a cada uno de los fotogramas.

En este contexto, el desarrollo de sistemas de búsqueda semántica se torna esencial frente a las búsquedas literales, donde la falta de coincidencias exactas puede resultar en resultados insatisfactorios. Los sistemas de búsqueda semántica analizan el significado global tanto del contenido como de la consulta [22]. La creación de un núcleo conceptual facilita la desambiguación de los términos de la consulta, logrando así una mayor coherencia y verosimilitud en los resultados. Este enfoque, al no depender exclusivamente de palabras clave o etiquetas específicas, permite capturar datos que no han sido etiquetados previamente o que podrían haber pasado desapercibidos durante la recopilación inicial. Además, al ser sistemas no supervisados, se elimina la necesidad de metadatar los vídeos, lo que garantiza que las colecciones no requieran una contextualización explícita. Esto es posible gracias a la granularidad de este tipo de sistemas, que permite extraer y utilizar la información relevante de manera efectiva.

Para implementar estas soluciones en un sistema, la incorporación de agentes autónomos basados en modelos de lenguaje se presenta como una herramienta eficiente y altamente beneficiosa. Un agente es un sistema inteligente diseñado para tomar decisiones y ejecutar acciones de manera autónoma para alcanzar un objetivo. Gracias a su capacidad para comprender y procesar el lenguaje natural, pueden interpretar con precisión las consultas de los usuarios y ejecutar las acciones necesarias para realizar búsquedas en amplias colecciones [23]. Este enfoque no solo optimiza el análisis de las consultas, sino que también mejora la eficiencia en la recuperación al permitir, mediante una única interacción, definir múltiples búsquedas.

Este trabajo tiene como propósito integrar estas propuestas de innovación tecnológica (la segmentación, la búsqueda semántica y el uso de agentes autónomos) en los procesos de gestión, accesibilidad y recuperación de información, abordando las limitaciones previamente identificadas. En consecuencia, se ha establecido el siguiente objetivo principal:

Desarrollo de un Asistente para el Proceso de Ideación:

Desarrollar una herramienta a través de una interfaz sencilla e intuitiva que simplifique e inspire a los profesionales del ámbito periodístico en la generación de nuevas piezas audiovisuales basadas en recursos de archivo histórico.

Con el fin de alcanzar este propósito, se han establecido los siguientes objetivos específicos:

Integración de Contenido Textual y Visual:

Integrar el contenido textual y visual en un espacio vectorial, permitiendo su comparación directa. Esto facilita a los periodistas la identificación y utilización de los recursos más pertinentes para sus producciones, asegurando coherencia tanto narrativa como visual.

Investigación y Análisis de Tecnologías de Segmentación Semántica:

Investigar y analizar cómo las tecnologías de segmentación semántica pueden mejorar la indexación de contenidos. La segmentación semántica ofrece una indexación óptima de los archivos históricos, reduciendo el volumen de datos a almacenar e incrementando la calidad de las búsquedas, lo que resulta en consultas más rápidas y relevantes.

Desarrollo y Evaluación de un Sistema de Búsquedas Semánticas:

Crear y evaluar un sistema de búsquedas semánticas capaz de recuperar recursos de archivo histórico de manera efectiva, garantizando que los resultados sean relevantes y contextualmente apropiados. La evaluación de este sistema es fundamental para asegurar su utilidad en el proceso creativo de los periodistas.

Implementación de un Agente Autónomo:

Implementar un agente autónomo que comprenda el guion de una pieza audiovisual y coordine las herramientas necesarias para su creación. Este agente debe identificar y ejecutar las tareas requeridas, proporcionando una visión integral de los elementos consultados y facilitando la composición de la pieza.

Estos objetivos específicos están diseñados para desarrollar una herramienta de apoyo que potencie la capacidad de los periodistas en la producción de contenido audiovisual de alta calidad (véase figura 1.3).

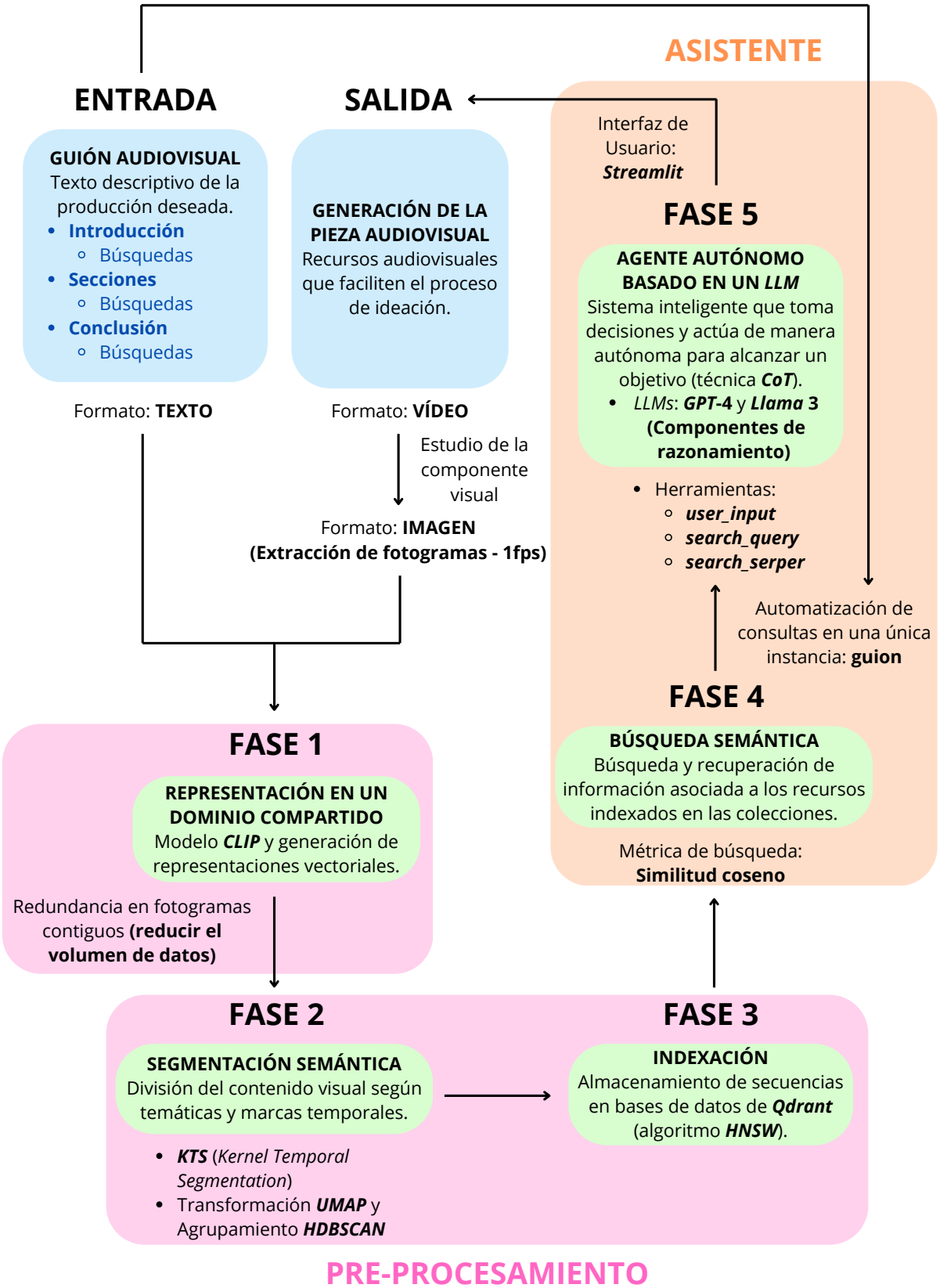


Figura 1.3: Diagrama de bloques que ilustra los desafíos abordados en este trabajo.

1.3. Planificación del Trabajo

En la figura 1.4 se presenta el diagrama de *Gantt*, que detalla la planificación del proyecto e ilustra la secuencia de tareas necesarias para su desarrollo. El proceso comienza con una exhaustiva revisión bibliográfica, cuyo objetivo es construir una base teórica sólida sobre la problemática abordada, además de profundizar en el estado del arte y en las tecnologías emergentes relacionadas. A continuación, se lleva a cabo la etapa de segmentación semántica de los vídeos, en la que se evalúan dos enfoques para el análisis de datos. Paralelamente, se investiga el sistema de indexación utilizado, así como los mecanismos de búsqueda y recuperación de información. El trabajo también abarca el diseño y ajuste de agentes autónomos, elementos clave para la automatización de diversas tareas dentro del sistema. Asimismo, se seleccionan los modelos de lenguaje empleados para la interpretación de las consultas de los usuarios. Otro aspecto relevante ha sido la caracterización de las bases de datos requeridas. En este contexto, se programan las fases correspondientes al procesamiento de datos, la ejecución de las funcionalidades y la implementación del asistente interactivo para el usuario final. Finalmente, se realiza un análisis exhaustivo de los resultados obtenidos y se procede a la presentación de las conclusiones derivadas del estudio.

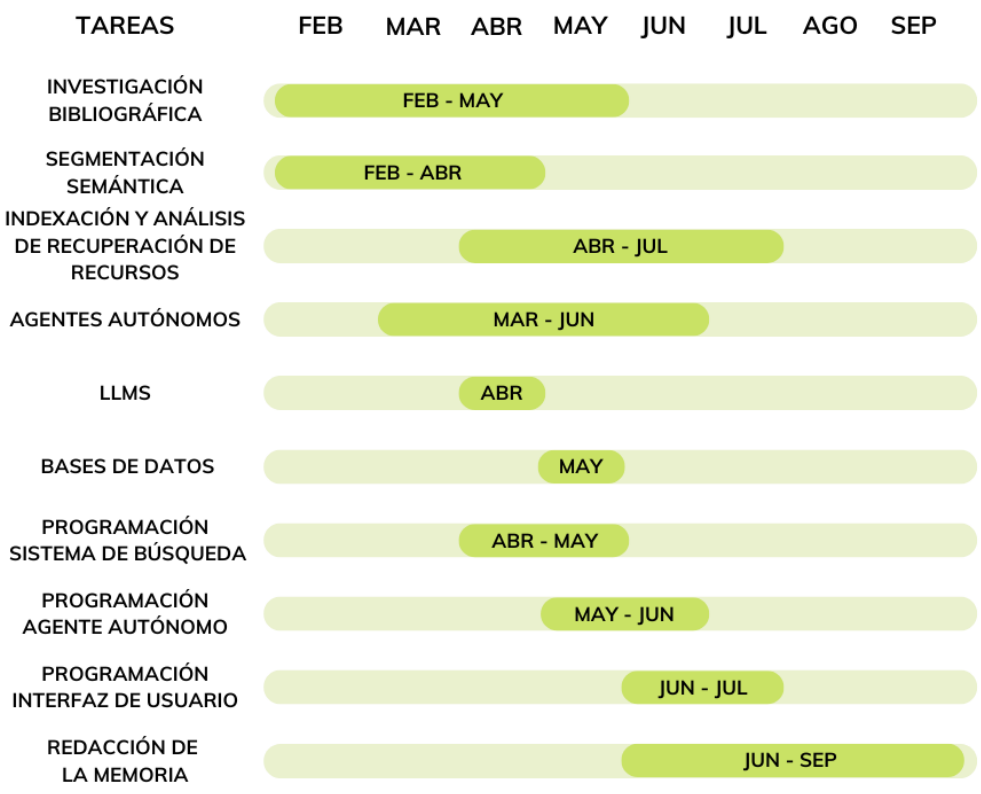


Figura 1.4: Diagrama de *Gantt* para el desarrollo del trabajo.

1.4. Esctructura de la Memoria

Esta memoria se encuentra compuesta por siete capítulos y cinco anexos, a continuación, se procede a enunciar brevemente el contenido de cada uno de ellos.

Capítulo 1. Introducción:

Este capítulo proporciona una breve contextualización del tema abordado, expone la motivación detrás del trabajo y define sus objetivos. Además, se detalla la planificación temporal y se presenta la estructura general de la memoria.

Capítulo 2. Representación Vectorial del Contenido Visual:

En este capítulo se analiza el uso del modelo multimodal *CLIP*, el cual integra texto e imágenes para la representación vectorial de las consultas y el contenido visual de los vídeos.

Capítulo 3. Segmentación y Búsqueda Semántica:

Introduce dos métodos de segmentación empleados para el indexado eficiente del contenido previamente representado vectorialmente. Asimismo, describe el proceso de búsqueda semántica destinado a la recuperación de recursos audiovisuales.

Capítulo 4. Desarrollo del Agente Autónomo:

En este capítulo se define el concepto de agente autónomo basado en grandes modelos de lenguaje, junto con las herramientas propuestas para la implementación y configuración del sistema.

Capítulo 5. Bases de Datos:

Este capítulo expone y caracteriza en detalle las bases de datos utilizadas para la implementación y evaluación del sistema propuesto.

Capítulo 6. Metodología Experimental y Resultados:

Recoge los experimentos realizados para evaluar el sistema desarrollado, con el fin de validar el cumplimiento de los objetivos planteados. Se exploran tanto las capacidades como las limitaciones del sistema para la recuperación de información.

Capítulo 7. Conclusiones y Líneas Futuras:

Finalmente, en este capítulo se extraen las conclusiones derivadas del estudio realizado. Se proponen líneas futuras que podrían ampliar y perfeccionar el sistema, en consonancia con las necesidades y desafíos identificados durante la investigación.

Capítulo 2

Representación Vectorial del Contenido Visual

En este capítulo, se aborda la metodología utilizada para la representación vectorial del contenido audiovisual, un proceso fundamental para el análisis y la comprensión de vídeos. La sección inicial detalla el procedimiento de extracción de fotogramas, empleando herramientas avanzadas como *OpenCV* para capturar y procesar imágenes a intervalos específicos del vídeo. Posteriormente, se describe la utilización del modelo *CLIP* (*Contrastive Language-Image Pre-Training*) para la generación de la representación semántica definida como *embedding*. Esta técnica permite convertir cada fotograma en una representación vectorial densa, facilitando su manipulación.

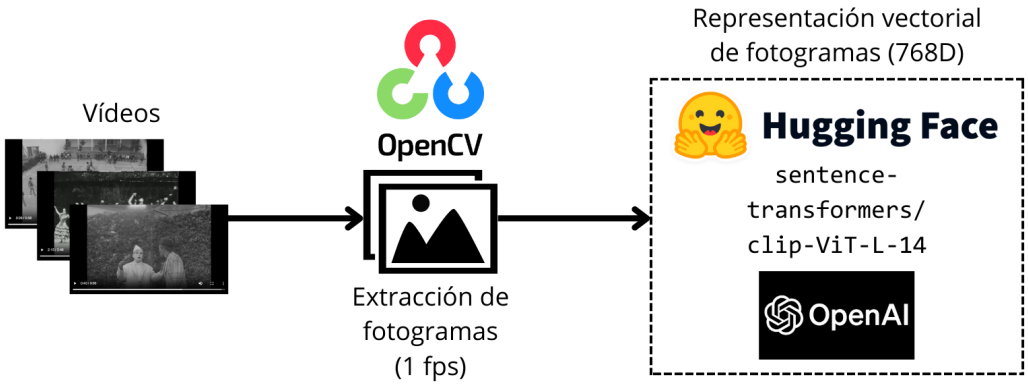


Figura 2.1: Diagrama de extracción de fotogramas y representación vectorial asociada.

2.1. Extracción de Fotogramas

El proceso de extracción de fotogramas establece los criterios para gestionar el contenido de los vídeos en el contexto de la recuperación de información. Para ello, se utiliza *OpenCV*[1] (*Open Source Computer Vision Library*), una biblioteca de *software* de código abierto y multiplataforma diseñada para aplicaciones de visión por computadora y aprendizaje automático. Esta biblioteca permite realizar diversas tareas de procesamiento de imágenes y análisis de vídeos.

Durante la fase de segmentación semántica, se extraen fotogramas a intervalos determinados, con una frecuencia de un fotograma por segundo (1 fps). Este enfoque optimiza la cantidad de datos a procesar sin comprometer la coherencia visual del vídeo, ya que la captura a 1 fps aprovecha la redundancia visual entre fotogramas consecutivos, permitiendo una mayor eficiencia en el procesamiento y manteniendo una representación visual precisa de los cambios significativos en la secuencia.

El procedimiento comienza con la apertura del archivo de vídeo utilizando la función `cv2.VideoCapture`, especificando el parámetro `apiPreference = cv2.CAP_FFMPEG`. *FFmpeg*[2] (*Fast Forward Moving Picture Experts Group*) es una colección de *software* libre que al integrarse con *OpenCV*, permite una decodificación rápida y un procesamiento optimizado, asegurando una captura eficiente.

A continuación, se utiliza un bucle para iterar sobre el vídeo, avanzando fotograma a fotograma con `video.grab()` sin cargar cada imagen en memoria, lo que optimiza el uso de recursos. En los intervalos definidos, el fotograma actual se extrae con `video.retrieve()` y se convierte del formato *BGR* (*Blue Green Red*) a *RGB* (*Red Green Blue*) mediante `cv2.cvtColor`, ya que las etapas posteriores requieren *RGB*.

Una vez convertidos, los fotogramas se transforman en objetos de la biblioteca *PIL*[3] (*Python Imaging Library*) con `Image.fromarray`. Este paso es fundamental porque el modelo de representación *CLIP* requiere que las imágenes estén en dicho formato, garantizando así la compatibilidad y precisión en el análisis.

Finalmente, se registran los tiempos asociados a cada fotograma, calculados en función de la posición del fotograma y la velocidad del vídeo, asegurando la correcta sincronización de eventos a lo largo del archivo.

En resumen, la extracción de fotogramas a 1 fps permite capturar y analizar la narrativa visual de los vídeos de manera eficiente, preservando la coherencia e integridad de la información visual, y facilitando su análisis posterior.

[1] *OpenCV*. Documentación disponible en: https://opencv.org/
[2] *FFmpeg*. Documentación disponible en: https://ffmpeg.org/
[3] *Pillow*. Documentación disponible en: https://pillow.readthedocs.io/

2.2. El Modelo *CLIP*

El modelo *CLIP* (*Contrastive Language-Image Pre-training*) [24] desarrollado por *OpenAI* representa un avance significativo en el establecimiento de conexiones entre representaciones visuales y textuales en un espacio vectorial compartido. A diferencia de los métodos tradicionales de aprendizaje supervisado, que dependen de etiquetas predefinidas, *CLIP* utiliza descripciones en lenguaje natural para caracterizar el contenido visual.

Este modelo integra dos componentes principales: un codificador de imágenes y un codificador de texto, ambos entrenados conjuntamente mediante un enfoque contrastivo. Los codificadores transforman imágenes y textos en representaciones vectoriales conocidas como *embeddings*, las cuales capturan las características esenciales de los datos transformados, permitiendo una representación eficaz de ambos formatos en un mismo espacio vectorial multimodal de alta dimensión.

El codificador de imágenes emplea la arquitectura *Vision Transformer* [25] (*ViT-L*/14), diseñada específicamente para tareas de visión por computadora, generando *embeddings* visuales que extraen el contenido de las imágenes. Por su parte, el codificador de texto, basado en un *Transformer* [26] para el procesamiento de lenguaje natural, convierte descripciones textuales en *embeddings* que reflejan la semántica del texto, admitiendo una longitud máxima de 77 *tokens*[4] para el texto. En este trabajo, las descripciones se han traducido al inglés para mejorar su representación, dado que *CLIP* ha sido preentrenado con una mayor cantidad de *tokens* en inglés en comparación con otros idiomas. Estos *embeddings* multimodales facilitan que el modelo relacione imágenes y texto de manera coherente.

El entrenamiento de *CLIP* se realiza utilizando un extenso conjunto de datos obtenidos de diversas fuentes en *Internet*, como redes sociales, bases de datos de imágenes y *webs* de noticias. Este enfoque elimina la necesidad de anotaciones manuales exhaustivas, ya que las fuentes proporcionan imágenes con descripciones textuales. La diversidad de datos expone al modelo a aprender múltiples conceptos y a capturar una amplia gama de estilos y formatos lingüísticos.

CLIP emplea lotes de N pares (imagen, texto) y optimiza una pérdida de entropía cruzada simétrica basada en las similitudes coseno calculadas para todos los emparejamientos posibles dentro de un lote (la definición de similitud coseno se encuentra en la subsección 3.2.2). Estas comparaciones se almacenan en una matriz de tamaño $N \times N$, donde cada fila compara un texto con todas las imágenes del lote y

[4]Un *token* es una unidad elemental de datos en el procesamiento de lenguaje natural, que puede corresponder a una palabra, una parte de una palabra o un símbolo, empleada para el análisis y la manipulación del texto.

cada columna una imagen con todos los textos. En esta matriz, los valores más altos se encuentran en la diagonal, lo que representa las correspondencias correctas entre imagen y texto. En contraste, los elementos fuera de la diagonal, que indican pares incorrectos, presentan valores más bajos, reflejando una menor similitud a medida que se alejan de la diagonal. El objetivo es maximizar la coincidencia para los pares correctos y minimizarla para los incorrectos. En este contexto, se entrena un parámetro de temperatura asociado a la función *softmax*, que regula la escala de las similitudes coseno calculadas, afinando la capacidad del modelo para diferenciar entre pares correctos e incorrectos y evitando la inestabilidad en el entrenamiento provocada por el escalado excesivo de los *logits*[5]. Este enfoque contrastivo permite a *CLIP* aprender a mapear los datos a un espacio de representación donde los elementos relacionados se encuentran cercanos entre sí y los no relacionados, separados.

Además, *CLIP* destaca por su capacidad para, durante el proceso de inferencia, operar sin la necesidad de etiquetas predefinidas, lo que le permite generalizar de manera efectiva a nuevos dominios y conceptos no observados durante su entrenamiento, gracias a la riqueza de las representaciones aprendidas. Este enfoque de aprendizaje, denominado *zero-shot*, demuestra un alto rendimiento en diversas tareas, sin requerir supervisión adicional específica para cada una de ellas. Esto proporciona una ventaja significativa en términos de generalización y flexibilidad, permitiendo al modelo aprender representaciones transferibles y eficaces en múltiples contextos y tareas, utilizando directamente la similitud coseno para comparar datos.

La figura 2.2 ilustra el proceso de entrenamiento. Este modelo entrena previamente un codificador de imágenes y un codificador de texto para predecir las correspondencias entre imágenes y textos en el conjunto de datos. Estos codificadores generan *embeddings* y la similitud coseno se usa para optimizar dichas correspondencias. Posteriormente, este mecanismo se emplea para convertir a *CLIP* en un clasificador de tipo *zero-shot*, prediciendo la clase del título que mejor se empareja con una imagen dada.

En conclusión, la arquitectura de *CLIP* y su enfoque contrastivo de entrenamiento permiten una integración efectiva de información visual y textual, resultando en un modelo robusto y versátil que puede aplicarse a una amplia variedad de tareas de visión por computadora y procesamiento de lenguaje natural. En la tarea abordada en este trabajo, *CLIP* se destaca como una herramienta competitiva para el proceso de segmentación semántica y, en última estancia, para la realización de búsquedas multimodales.

[5]Los *logits* son las predicciones sin procesar generadas por un modelo de red neuronal, que posteriormente se normalizan mediante funciones de activación, como *softmax*, para convertirlas en probabilidades.

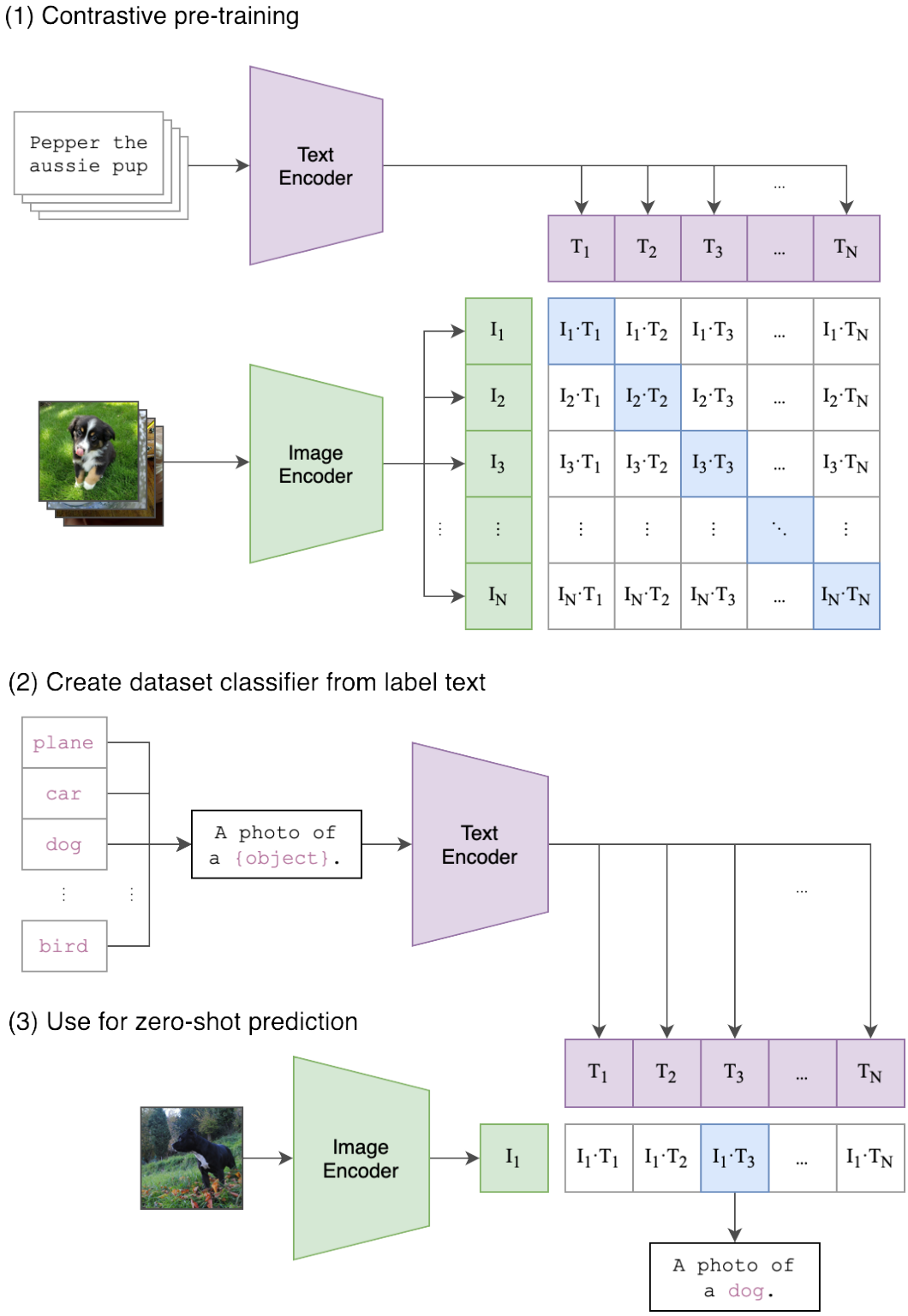


Figura 2.2: Diagrama de la arquitectura y funcionamiento de *CLIP*.

2.3. Generación de *Embeddings*

En esta sección se describe el proceso de generación de *embeddings* para el análisis de vídeos, fundamental para convertir el contenido visual en un formato procesable. Los *embeddings* son representaciones vectoriales que capturan las características más relevantes de los fotogramas en un espacio multidimensional, facilitando su análisis, comparación e identificación de variaciones significativas, lo que permite segmentar el vídeo en partes coherentes. Para ello, se emplea el modelo preentrenado *CLIP ViT-L/14*, disponible en la plataforma de *Hugging Face*[6], mediante la implementación proporcionada por la biblioteca *Sentence Transformers* (*SBERT*) [27], que facilita el mapeo de imágenes y texto en un espacio común.

El entorno se configura para usar una *GPU* (*Graphics Processing Unit*), si está disponible, para optimizar el procesamiento de grandes volúmenes de datos. El modelo se carga mediante el comando: `SentenceTransformer('clip-ViT-L-14', device=self.device)` y se activa la función `model.eval()`, que desactiva capas específicas que operan de forma distinta durante el entrenamiento y la inferencia. Además, se emplea `torch.no_grad()` para desactivar el cálculo de gradientes, lo que reduce el uso de memoria y acelera el proceso de inferencia, ya que en esta fase solo se busca obtener los resultados sin modificar los parámetros del modelo.

Los *embeddings* se generan procesando los fotogramas mediante el codificador de imágenes *ViT* (*Vision Transformer*) de *CLIP*. Este proceso captura tanto el contenido como las relaciones espaciales del fotograma, generando una representación vectorial densa y normalizada de 768 dimensiones, utilizando el comando `model.encode(frame, normalize_embeddings=True)`.

El modelo *ViT*, cuya estructura se detalla en el anexo A, añade una capa adicional de normalización a los *embeddings* combinados de bloques y posiciones antes de entrar en el *Transformer*. Esta normalización asegura que los vectores tengan longitud unitaria, lo que facilita los cálculos de similitud en etapas posteriores. Los *embeddings* generados se almacenan en una matriz $N \times 768$, donde cada fila representa la vectorización de un fotograma.

[6]Modelo disponible en: https://huggingface.co/sentence-transformers/clip-ViT-L-14

Capítulo 3

Segmentación y Búsqueda Semántica

En el siguiente capítulo se presentan métodos especializados en la segmentación de material audiovisual, así como en la indexación y recuperación de información mediante bases de datos vectoriales. La segmentación semántica constituye una estrategia práctica para dividir vídeos en segmentos significativos, reflejando los cambios en el contenido y facilitando su análisis y comprensión. Asimismo, la búsqueda semántica permite la recuperación eficiente de información relevante en grandes conjuntos de datos, aprovechando la similitud semántica entre elementos.

3.1. Segmentación Semántica de Vídeos

La segmentación de vídeos basada en semántica tiene como propósito gestionar el contenido visual de manera eficiente a través de la comprensión del significado de los elementos y acciones presentes en el vídeo. Este proceso abarca la identificación y delimitación de segmentos pertinentes dentro de un vídeo, optimizando su procesamiento y exploración posterior. Como resultado, se logra un análisis rápido e intuitivo de la narrativa, mejorando el almacenamiento de información y eliminando redundancias sin pérdida de calidad. En este contexto, se exploran dos enfoques principales:

- ***KTS* (*Kernel Temporal Segmentation*):** Emplea métodos basados en núcleo para detectar puntos de cambio significativos en el contenido de los vídeos, permitiendo una segmentación precisa de las variaciones visuales.

- **Transformación *UMAP* y Agrupamiento *HDBSCAN*:** Combina la reducción de dimensionalidad de *UMAP* con el agrupamiento de *HDBSCAN* para segmentar el contenido de manera eficiente y organizada.

3.1.1. *KTS*: *Kernel Temporal Segmentation*

El algoritmo *KTS*[1] (*Kernel Temporal Segmentation*) [28] es una técnica avanzada empleada para segmentar vídeo en segmentos temporales no solapados, mediante la detección de "puntos de cambio" en la estructura del vídeo. Este método se basa en la identificación de transiciones en el contenido del vídeo, permitiendo dividirlo en segmentos que reflejan cambios en la semántica o dinámica de la escena.

A diferencia de la detección de límites de tomas (*shot boundary detection*), que típicamente se centra en transiciones abruptas y evidentes, como cambios de cámara, la detección de puntos de cambio (*kernel-based change point detection*) adopta un enfoque estadístico más general. *KTS* es capaz de identificar cambios sutiles en el contenido semántico, tales como variaciones en el ambiente o en las acciones realizadas en la escena. Resulta especialmente útil para obtener fragmentos significativos que faciliten un análisis detallado de la estructura temporal en función de la narrativa visual.

El algoritmo 1 presenta un resumen detallado del procedimiento, el cual se describe paso a paso a continuación.

Algoritmo 1 *Kernel Temporal Segmentation*

Entrada: Secuencia temporal de descriptores $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}$

1: Calcular la matriz de similitudes coseno K:
$$K_{i,j} = S_C(\mathbf{x}_i, \mathbf{x}_j) = \frac{\mathbf{x}_i \cdot \mathbf{x}_j}{\|\mathbf{x}_i\| \|\mathbf{x}_j\|}$$
2: Calcular las sumas acumulativas de K
3: Calcular varianzas no normalizadas
$$v_{t,t+d} = \sum_{i=t}^{t+d-1} K_{i,i} - \frac{1}{d} \sum_{i,j=t}^{t+d-1} K_{i,j}$$
$$t = 0, \ldots, n-1, \quad d = 1, \ldots, n-t$$
4: Programación dinámica para la detección de puntos de cambio:
$$L_{i,j} = \text{mín}_{t=i,\ldots,j-1}(L_{i-1,t} + v_{t,j}), \quad L_{0,j} = v_{0,j}$$
$$i = 1, \ldots, m_{\text{máx}}, \quad j = 1, \ldots, n$$
5: Seleccionar el número óptimo de puntos de cambio:
$$m^* = \arg\text{mín}_{m=0,\ldots,m_{\text{máx}}} L_{m,n} + Cg(m,n)$$
6: Encontrar posiciones de puntos de cambio retrocediendo:
$$t_{m^*} = n, \quad t_{i-1} = \arg\text{mín}_t(L_{i-1,t} + v_{t,t_i})$$
$$i = m^*, \ldots, 1$$

Salida: Posiciones de puntos de cambio $t_0, \ldots, t_{m^*-1}$

[1] El código del algoritmo *KTS* está disponible en https://github.com/TatsuyaShirakawa/KTS/

En el capítulo anterior, se ha expuesto la transformación de fotogramas de vídeo en representaciones vectoriales utilizando el codificador de imágenes de *CLIP*, con el propósito de capturar características semánticas complejas. Estas representaciones se comparan mediante la métrica de similitud coseno.

Dada una secuencia de descriptores[2] de fotogramas $\mathbf{x}_i \in \mathbf{X}$, $i = 0, \ldots, n-1$, se construye una matriz de similitudes coseno K comparando cada par de descriptores $\mathbf{x}_i$ y $\mathbf{x}_j$. Para optimizar los cálculos posteriores, se obtienen las sumas acumulativas de los elementos de dicha matriz, lo que permite la suma eficiente de cualquier submatriz de K (véase anexo B). Para cada posible punto de inicio t y duración del segmento d, se calcula la varianza no normalizada del segmento, evaluando la dispersión de las similitudes dentro de ese intervalo temporal.

El siguiente paso consiste en actualizar la matriz de costos acumulados L mediante programación dinámica a través de un *forward pass*[3]. Este procedimiento permite identificar la configuración óptima de puntos de cambio en un vídeo, minimizando la varianza interna de los segmentos temporales.

Durante este proceso, el algoritmo itera sobre el número de puntos de cambio, i, desde 1 hasta $m_{\text{máx}}$, donde $m_{\text{máx}}$ es el número máximo permitido de puntos de cambio, correspondiente al número total de fotogramas menos uno, y simultáneamente sobre la posición de cada fotograma, j, buscando en cada combinación el punto de cambio previo, t, que minimice la función objetivo acumulada. En este contexto, $L_{i,j}$ representa la suma del costo acumulado hasta t con $i-1$ cambios, más la varianza del segmento entre t y j.

Una vez completadas las iteraciones, se procede a seleccionar el número óptimo de puntos de cambio minimizando la función objetivo total $J_{m,n}$:

$$\underset{m;\, t_0,\ldots,t_{m-1}}{\text{Minimizar}} \quad J_{m,n} := L_{m,n} + Cg(m,n) \tag{3.1}$$

Esta función se descompone en dos términos: $L_{m,n}$, que cuantifica la varianza dentro de los segmentos, y $g(m,n)$, que introduce una penalización por un número excesivo de segmentos. La constante C ajusta la importancia del término de penalización.

[2]Los descriptores son las representaciones vectoriales de cada uno de los fotogramas que componen el vídeo muestreado, normalizados según la norma euclídea ($\|\cdot\|_2$).

[3]En programación dinámica, "*forward pass*" se refiere a la fase en la que se calculan y almacenan secuencialmente los valores óptimos de la función objetivo para todas las combinaciones de subproblemas, avanzando desde el primer estado hasta el último.

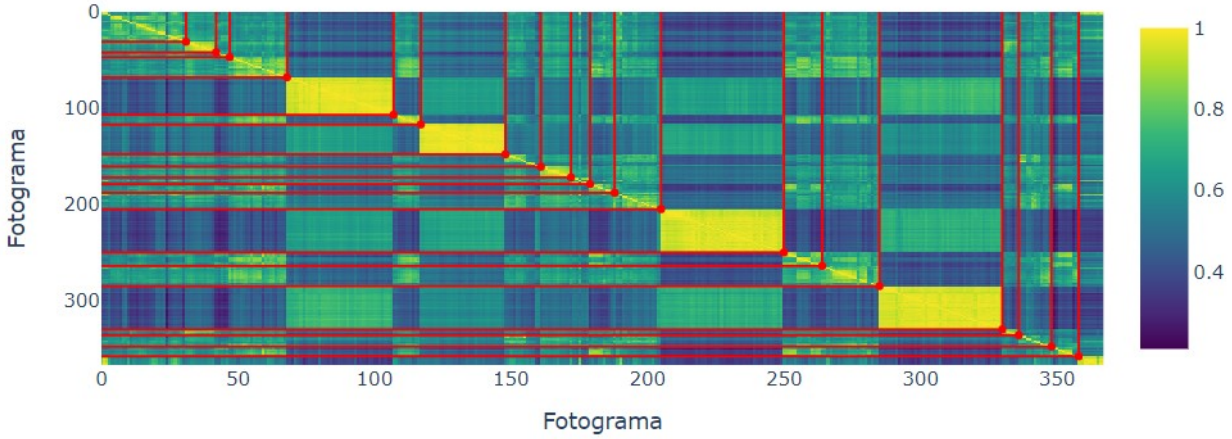
Para la penalización, se aplica el criterio *BIC* (*Bayesian Information Criterion*) [29]. El *BIC* busca un equilibrio entre precisión y simplicidad penalizando los modelos más complejos para evitar el sobreajuste. En este contexto, aunque aumentar el número de segmentos reduce $L_{m,n}$, también incrementa la complejidad del modelo. Por ello, esta penalización ayuda a equilibrar la subsegmentación y la sobresegmentación, regulando el número de puntos de cambio m y asegurando un balance adecuado entre la varianza dentro de los segmentos y la complejidad del modelo. Se expresa como $g(m, n) = m(\log(n/m) + 1)$, donde m es el número de puntos de cambio y n es la longitud del vídeo.

Posteriormente, se reconstruye la segmentación del vídeo mediante un proceso de *backtracking*[4] para determinar las posiciones exactas de los puntos de cambio calculados, encontrando así los límites de los segmentos temporales. El costo total de tiempo de ejecución del algoritmo es $O(m_{\text{máx}}n^2)$. La penalización introduce una sobrecarga computacional mínima porque el algoritmo de programación dinámica ya calcula $L_{i,n}$ para todas las posibles cantidades de segmentos.

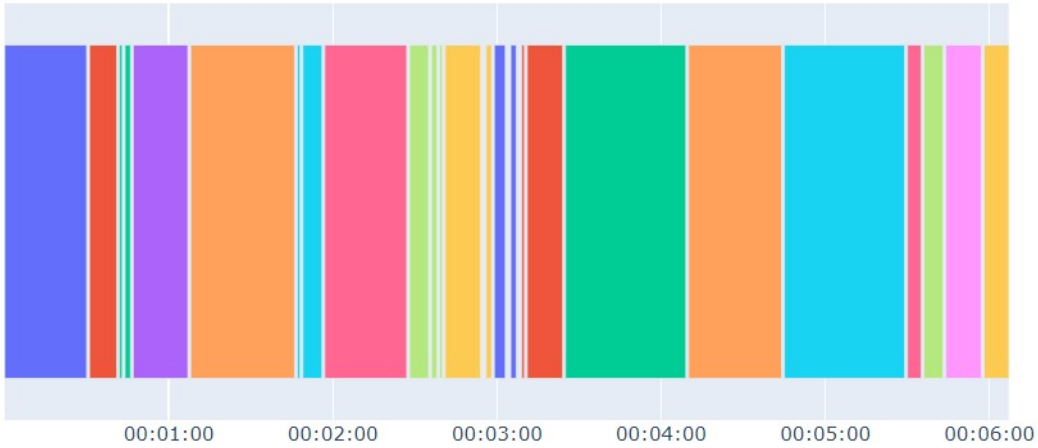
Una vez determinadas las posiciones de los puntos de cambio, la secuencia de fotogramas puede segmentarse de acuerdo con estas posiciones. Dado que los tiempos correspondientes a cada fotograma son conocidos, se pueden definir claramente los segmentos y asignarles el tiempo correspondiente a cada punto de cambio.

El algoritmo *KTS* ofrece una metodología para segmentar vídeos en intervalos homogéneos en términos de contenido visual, basándose en las características de los fotogramas para facilitar una administración semántica eficaz de los datos.

[4]El "*backtracking*" es un proceso de búsqueda inversa que permite recuperar la secuencia de decisiones que llevaron a la solución óptima.



(a) Representación de la matriz de similitudes coseno, con los puntos de cambio estimados mediante el algoritmo *KTS* destacados en rojo.



(b) Línea temporal de las escenas estimadas, segmentadas según los puntos de cambio identificados.

Figura 3.1: Ejemplo de segmentación semántica aplicada a un vídeo utilizando el algoritmo *KTS*.

3.1.2. Transformación *UMAP* y Agrupamiento *HDBSCAN*

Otra de las soluciones propuestas para optimizar la segmentación semántica en vídeos consiste en combinar la transformación *UMAP* con el algoritmo *HDBSCAN*, con el fin de reducir la dimensionalidad de los *embeddings* extraídos de los fotogramas de vídeo en el capítulo anterior y agrupar los vectores que representen contenido similar (véase figura 3.3). Este enfoque reduce la complejidad espacial de los datos mientras preserva su estructura, facilitando la identificación y agrupación eficiente de patrones relevantes.

UMAP (*Uniform Manifold Approximation and Projection*) [30] es una técnica avanzada para la reducción de dimensionalidad no lineal y la visualización de datos complejos. A diferencia de otros métodos, que tienden a priorizar las relaciones locales sobre las globales, *UMAP* equilibra ambas proyectando los datos en un espacio de menor dimensionalidad sin perder su estructura subyacente. Esta técnica se basa en el aprendizaje de variedades (*manifold learning*), cuyo objetivo es descubrir la estructura interna de datos de alta dimensionalidad [31].

El proceso de *UMAP* comienza construyendo un grafo que conecta cada punto de datos con sus vecinos más cercanos, el cual se transforma en un "conjunto simplicial difuso"[5] que captura la geometría de la variedad en la que se distribuyen los datos. Posteriormente, se optimiza esta representación para reducir la dimensionalidad, minimizando la entropía cruzada entre las representaciones en ambos espacios.

Se asume que, aunque los datos estén en un espacio de muchas dimensiones, pueden organizarse en torno a una estructura de menor dimensión o "*manifold*". Esto facilita la identificación de relaciones no lineales más complejas y la revelación de la geometría intrínseca de los datos, mejorando su visualización y análisis.

En este contexto, *UMAP* reduce la dimensionalidad de los *embeddings*, transformando las 768 dimensiones originales a solo dos. Esto permite una representación gráfica más intuitiva en un plano bidimensional, mejorando la eficiencia en el análisis y facilitando la interpretación visual de los resultados.

El modelo *UMAP* se configura mediante una serie de parámetros seleccionados para maximizar la calidad de la proyección en el espacio reducido. Para calcular las distancias entre los puntos en el espacio original, se utiliza la métrica `'cosine'`, una medida eficaz que captura las disimilitud entre los *embeddings*. La fuerza de repulsión se fija en un valor de 2, controlando la dispersión de los puntos y evitando su superposición, lo que asegura una distribución equilibrada.

Durante la optimización, se emplean 20 muestras negativas por cada muestra positiva, lo que mejora significativamente la calidad de la proyección final aunque implica un mayor tiempo computacional, un compromiso necesario para obtener resultados más precisos. Para preservar la estructura local de los datos, se consideran los 30 vecinos más cercanos en la construcción del grafo, lo que garantiza la coherencia de las relaciones entre los puntos y una adecuada captura de la topología subyacente.

[5] Un conjunto simplicial difuso (*fuzzy simplicial set*) es una estructura que une puntos de datos, indicando no solo si están relacionados, sino también la fuerza de esa conexión.

En el espacio reducido, se adopta la métrica `'euclidean'` para medir las distancias, facilitando así el análisis geométrico de los datos proyectados. Finalmente, se establece una semilla con valor 42 para el generador de números aleatorios, lo que asegura la reproducibilidad de los resultados en ejecuciones posteriores, un aspecto crucial para validar la consistencia del modelo.

El modelo *UMAP* configurado se entrena y transforma los *embeddings* al nuevo espacio, generando una matriz que contiene las representaciones de los datos originales en baja dimensionalidad.

```
umap_model = umap.UMAP(n_components=2, metric='cosine',
↪  repulsion_strength=2.0, negative_sample_rate=20, n_neighbors=30,
↪  output_metric='euclidean', random_state=42)
mapper = umap_model.fit(emb)
array_umap = mapper.transform(emb)
```

Tras la transformación de los datos, se implementa el algoritmo *HDBSCAN* (*Hierarchical Density-Based Spatial Clustering of Applications with Noise*) [32], un método de agrupamiento que supera las limitaciones de *DBSCAN* mediante la introducción de una estructura jerárquica. Esta permite identificar grupos de datos en diferentes niveles de densidad, lo cual es esencial en escenarios donde la distribución de los datos es heterogénea, como la segmentación semántica de vídeo, donde las temáticas pueden variar considerablemente a lo largo del tiempo. A diferencia de *DBSCAN*, que depende de un umbral de densidad fijo, *HDBSCAN* ajusta dinámicamente su enfoque a las características de los datos, otorgándole gran flexibilidad para detectar patrones complejos.

El algoritmo comienza con la transformación del espacio de datos, distinguiendo entre áreas densas y dispersas. Este proceso se basa en la distancia al k-ésimo vecino más cercano de cada punto, lo que permite separar eficazmente las regiones densas del ruido. Luego, se emplea una métrica de alcanzabilidad mutua, que considera tanto la distancia real entre dos puntos como la densidad de las áreas en las que se encuentran. Esto ayuda a construir un árbol de expansión mínima, una estructura gráfica que conecta los puntos con el mínimo número de conexiones necesarias, identificando grupos densos de manera eficiente.

Una vez creado el árbol de expansión mínima, *HDBSCAN* genera una jerarquía de clústeres mediante la fusión progresiva de puntos en función de su distancia. Esta jerarquía se representa a través de un dendrograma[6], que captura las relaciones entre los clústeres antes de extraer los grupos finales.

[6]El dendrograma es un diagrama de árbol que muestra cómo se forman las agrupaciones de datos.

Para reducir la complejidad y el ruido, el algoritmo aplica un umbral de tamaño mínimo de clúster, descartando los grupos que no alcanzan este criterio y mejorando así la precisión del agrupamiento.

HDBSCAN selecciona los clústeres definitivos evaluando su persistencia a través de diferentes niveles de densidad. Los grupos más estables, es decir, aquellos que permanecen consistentes a lo largo de una amplia gama de densidades, se consideran los más representativos, lo que asegura la robustez de los clústeres extraídos.

Los puntos que no pertenecen a ningún clúster estable son etiquetados como ruido y se les asigna el valor de etiqueta -1, mientras que los puntos que forman parte de clústeres estables reciben una etiqueta que indica el clúster específico al que pertenecen. Esta información se emplea para la segmentación del vídeo, ya que al aplicar el algoritmo *HDBSCAN* sobre las representaciones generadas por *UMAP* de los fotogramas, los agrupa en clústeres. Dado que los puntos cercanos se consideran semánticamente similares según lo definido por *CLIP*, cada clúster representa una temática distinta.

Para iniciar el proceso de segmentación, se configura el parámetro `epsilon`. Este parámetro controla la distancia máxima permitida entre dos puntos para que puedan ser agrupados en el mismo clúster, estableciendo así el umbral de proximidad necesario para fusionar puntos en áreas de alta densidad. De esta manera, se evita que los grupos se dividan innecesariamente.

Cuando se utiliza un valor grande de `epsilon`, se permite una mayor distancia entre puntos para ser considerados parte del mismo grupo. Esto facilita la agrupación general de puntos que, aunque están relacionados de manera más amplia, no están necesariamente muy cercanos entre sí. Por ello, se realiza una primera agrupación por escenas[7], optando por un valor de 2 para `epsilon`.

Por otro lado, un valor pequeño de `epsilon` reduce la distancia máxima permitida entre puntos dentro del mismo clúster, permitiendo así una agrupación más detallada y específica. Esto facilita la identificación de subgrupos de puntos que están más estrechamente relacionados en términos de similitud. Para realizar una segunda agrupación, esta vez por secuencias[8], se utiliza un valor de 1, permitiendo una separación más concreta de las temáticas del vídeo y un análisis más fino del contenido.

[7]Una escena es un segmento de vídeo que representa un conjunto de planos relacionados temáticamente y temporalmente.

[8]Una secuencia representa una unidad más pequeña y específica de acción o eventos dentro de una escena general.

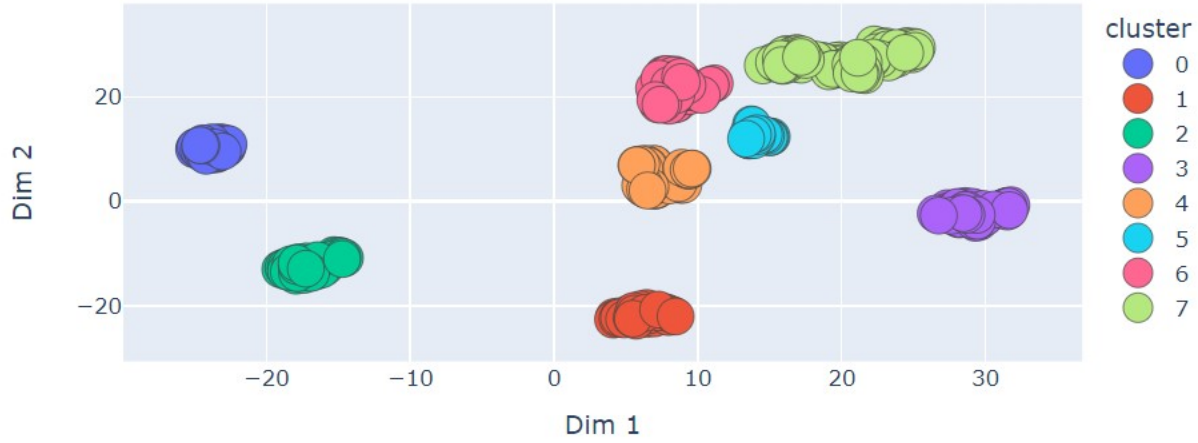
Además, se configuran parámetros adicionales para garantizar la validez de los grupos y minimizar el impacto del ruido en los datos. Se establece un mínimo de 10 muestras por clúster para asegurar que sean lo suficientemente grandes y significativos. Asimismo, se requiere que un punto tenga al menos un vecino cercano para considerarlo como central, incluso en áreas de baja densidad.

El comando empleado para realizar ambas agrupaciones es el siguiente:

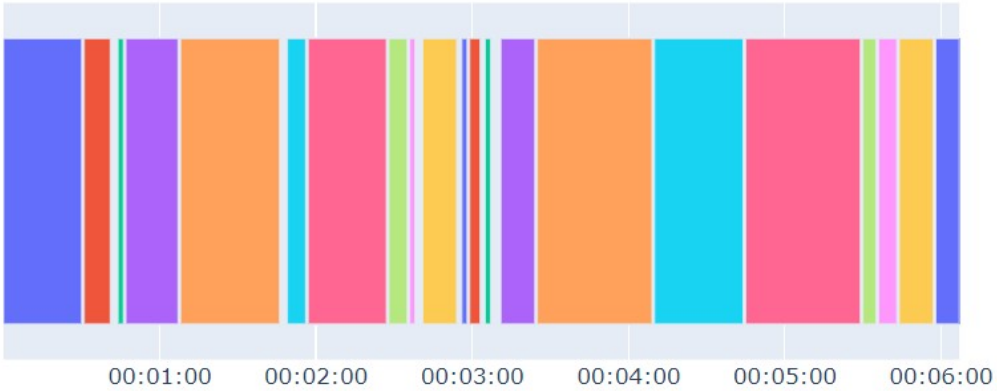
```
hdbscan_labels = hdbscan.HDBSCAN(min_cluster_size=10, min_samples=1,
↪  cluster_selection_epsilon=epsilon).fit_predict(array_umap)
```

La estrategia de doble segmentación no solo optimiza el proceso de agrupamiento, sino que también facilita el almacenamiento de datos al reducir el número de *embeddings* necesarios para caracterizar los vídeos, conservando únicamente la información más relevante. Las etiquetas obtenidas, junto con las marcas de tiempo, se utilizan para segmentar el vídeo, basándose en los cambios de etiquetas a lo largo del tiempo.

Se considera una escena cuando presenta que al menos tres fotogramas pertenecen al mismo clúster; a este clúster se le reaplica el algoritmo para identificar las secuencias dentro de la escena. Para cada segmento identificado, se calcula y normaliza el *embedding* promedio, que se almacena junto con los tiempos de inicio y fin del segmento, así como otros metadatos relevantes, como el nombre, el tema del vídeo, y otros detalles.



(a) Gráfico de dispersión de las representaciones *UMAP* etiquetadas según el clúster asignado tras aplicar el algoritmo *HDBSCAN*.



(b) Línea temporal de las escenas calculadas, separadas según los cambios en el etiquetado inicial de *HDBSCAN* a medida que avanza el vídeo.

Figura 3.2: Ejemplo de segmentación semántica aplicada a un vídeo utilizando el método *UMAP+HDBSCAN*.

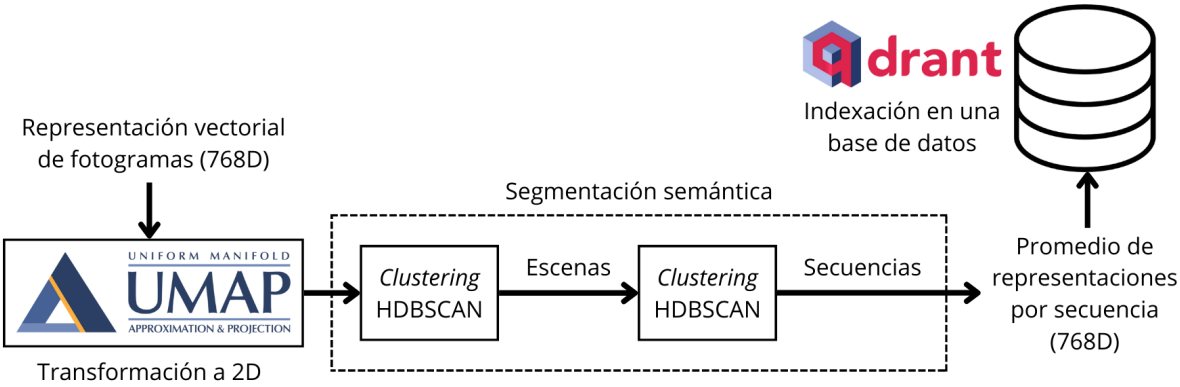


Figura 3.3: Diagrama de segmentación semántica e indexado.

Se ha observado que ambos métodos de segmentación de vídeos ofrecen prestaciones similares. Aunque el algoritmo *KTS* ha sido ampliamente utilizado como referencia en este tipo de sistemas, el método *UMAP+HDBSCAN* resulta más adecuado para los objetivos de este estudio. Este enfoque no busca una segmentación completamente precisa, sino que prioriza el agrupamiento de fotogramas de manera semántica y temporal, lo cual facilita la representación del contenido visual del vídeo. Esta representación es crucial para permitir consultas textuales en el mismo dominio. Además, el método reduce significativamente el volumen de datos a almacenar y acelera el proceso de segmentación, completándolo en segundos en comparación con los varios minutos que requiere el algoritmo *KTS*.

3.2. Indexado y Búsqueda con *Qdrant*

3.2.1. *Qdrant*: Base de Datos Vectorial

En la figura 3.3 se ilustra que el proceso de indexación de las representaciones de cada secuencia se realiza utilizando la base de datos vectorial *Qdrant*, un motor de búsqueda especializado en la similitud de vectores. *Qdrant* proporciona una *API* (*Application Programming Interface*) para almacenar, buscar y gestionar puntos vectoriales (*embeddings*) junto con metadatos adicionales (*payload*), y está diseñado para aplicaciones que requieren búsquedas semánticas.

Para iniciar la indexación, es necesario especificar la ubicación de almacenamiento de datos. Con este fin, se ha configurado un cliente en la nube de *Qdrant*, utilizando la clave *API* correspondiente para acceder al servicio:

```
cliente = QdrantClient(
    url = "https://endpoint.cloud.qdrant.io",   # Servicio Qdrant
    api_key = "su_clave_api",                   # Clave de la API
)
```

Se ha creado una colección para cada base de datos, con el propósito de facilitar las búsquedas posteriores. La configuración de las colecciones especifica un tamaño vectorial de 768 dimensiones y emplea la similitud coseno como métrica de distancia para determinar la similitud entre los elementos indexados y los vectores consulta:

```
client.recreate_collection(collection_name=database,
↪  vectors_config=models.VectorParams(size=768,
↪  distance=models.Distance.COSINE))
```

Cada colección se actualiza dinámicamente a medida que se calculan los *embeddings* que representan cada secuencia. Para evitar sobrescribir entradas existentes, se ajusta el índice de cada punto al momento de su inserción:

```
client.upsert(
    collection_name = database,
    points = [
        models.PointStruct(
            vector = emb.tolist(),
            payload = metadata[i],
            id = ids.count + i
        )
        for i, emb in enumerate(average_embedding)
    ]
)
```

Una vez indexados todos los elementos, se cierra el cliente para finalizar la conexión, `client.close()`.

3.2.2. Búsqueda Semántica y Métrica de Similitud

La búsqueda semántica basada en vectores es un desafío significativo cuando se trata de identificar recursos similares dentro de conjuntos de datos masivos. Este tipo de búsqueda se basa en la comparación de representaciones vectoriales de los datos, lo que resulta eficaz para capturar relaciones semánticas complejas. Sin embargo, a medida que el volumen de datos aumenta, calcular la distancia entre el vector de consulta y cada elemento del conjunto de datos se vuelve ineficiente.

Para abordar este problema, *Qdrant* implementa el algoritmo *HNSW* (*Hierarchical Navigable Small World*) [33], una técnica avanzada para la búsqueda de vecinos más cercanos en grandes conjuntos de datos. *HNSW* organiza la información en un grafo jerárquico multinivel. En los niveles superiores, los nodos están más dispersos y conectados por enlaces largos, lo que permite realizar una búsqueda inicial rápida.

A medida que se desciende en la jerarquía, los nodos están más densamente conectados, lo que permite refinar la búsqueda de manera precisa. Este enfoque aprovecha la propiedad de los "mundos pequeños" de los grafos, donde la mayoría de los nodos se alcanzan con pocos saltos, reduciendo así el número de comparaciones y mejorando la eficiencia de la búsqueda sin comprometer significativamente la precisión. El método de búsqueda aproximada de *HNSW* puede configurarse mediante el parámetro `exact_search=False`.

El proceso de búsqueda comienza codificando la consulta en un *embedding*. Luego, este *embedding* se procesa mediante la función `similarity_search`, que recupera los puntos más cercanos al vector de consulta utilizando la similitud coseno como métrica de comparación. Esta métrica, indicada previamente en la configuración de la base de datos, evalúa la afinidad entre vectores normalizados calculando el producto escalar entre ellos:

$$S_C(\boldsymbol{a}, \boldsymbol{b}) = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{a}\|\|\boldsymbol{b}\|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}} \tag{3.2}$$

donde $\boldsymbol{a}$ y $\boldsymbol{b}$ son los vectores que representan los elementos a comparar.

Los valores de la similitud coseno oscilan entre -1 y 1. Valores cercanos a 1 indican alta similitud, lo que refleja características semánticas comunes. Valores próximos a 0 sugieren que los vectores son ortogonales en el espacio vectorial, señalando poca relación entre sus características. Finalmente, valores cercanos a -1 denotan una mayor disimilitud.

```
def similarity_search(client, database, embedding, value, threshold,
↪  exact_search=False):
    search_results = client.search(
        collection_name = database,
        query_vector = embedding,
        limit = value,
        with_payload = True,
        with_vectors = False,
        score_threshold = threshold,
        search_params = models.SearchParams(exact=exact_search),
        timeout = 120,
    )
    return search_results
```

Los parámetros de la función incluyen la base de datos en la que se realiza la consulta, el vector de consulta, el número máximo de resultados deseados, y un umbral de puntuación mínima que filtra los resultados para devolver solo aquellos con una similitud superior a la especificada. Además, se establece un tiempo máximo de 120 segundos para completar la búsqueda. El parámetro `exact_search` define si la búsqueda debe ser precisa o aproximada. Cuando se configura `False`, la búsqueda prioriza la velocidad sobre la exhaustividad.

Capítulo 4

Desarrollo del Agente Autónomo

En el campo de la inteligencia artificial y la computación, los agentes autónomos representan un avance significativo hacia sistemas más inteligentes y autosuficientes. Estos agentes, diseñados para operar de forma independiente y ejecutar tareas complejas sin intervención humana constante, se fundamentan en grandes modelos de lenguaje que les proporcionan la capacidad de razonamiento y comprensión necesarias para seleccionar las herramientas y acciones apropiadas.

Este capítulo se centra en el desarrollo de agentes autónomos mediante *LangChain*, una plataforma versátil para la creación de aplicaciones basadas en grandes modelos de lenguaje. Se analiza cómo modelos como *GPT*-4 y *Llama* 3 proporcionan la base cognitiva fundamental para estos agentes. Además, se describen las herramientas y metodologías empleadas para alcanzar los objetivos del sistema global, integrando de manera efectiva estas componentes.

4.1. Grandes Modelos de Lenguaje: Componentes de Razonamiento

Los modelos de lenguaje (*LMs*, por sus iniciales en inglés) están diseñados para calcular la probabilidad generativa de secuencias de palabras al considerar la información contextual de las palabras precedentes, lo que les permite predecir las palabras siguientes. Esto se logra mediante el uso de estrategias de selección de palabras, que permiten a los *LMs* generar textos en lenguaje natural de manera competente [34].

Además de crear secuencias de texto coherentes, los avances recientes en este campo han dado lugar a la creación de arquitecturas más poderosas, que mejoran la capacidad de comprensión y producción del lenguaje. Este progreso ha culminado en los llamados grandes modelos de lenguaje (*LLMs*, por sus siglas en inglés), que representan un salto significativo en la evolución del procesamiento del lenguaje natural.

Los *LLMs* han revolucionado el procesamiento del lenguaje natural mediante el uso de redes neuronales a gran escala, estableciendo nuevos estándares en el campo de la inteligencia artificial. Estos modelos se entrenan con inmensos volúmenes de datos textuales provenientes de diversas fuentes, como libros, artículos y contenido *web*. A diferencia de los enfoques tradicionales, que dependen de reglas explícitas, los *LLMs* adquieren conocimiento a través de la exposición a datos, desarrollando una comprensión profunda y matizada del lenguaje humano.

Diseñados para emular el funcionamiento del cerebro humano, los *LLMs* son capaces de identificar patrones complejos en la información textual. Su capacidad para procesar y comprender aspectos avanzados del lenguaje, como la gramática, la semántica y el contexto, les permite adaptarse a una amplia diversidad de temáticas y estilos de escritura. Esta habilidad para captar complejidades lingüísticas, superando las limitaciones de los enfoques tradicionales, les otorga la facultad de generar respuestas coherentes y contextualmente apropiadas, a menudo indistinguibles del lenguaje producido por humanos.

El sofisticado entrenamiento de los *LLMs* les permite ejecutar una variedad de tareas complejas relacionadas con el lenguaje natural, como la generación de texto, el análisis de información y la respuesta a preguntas. Una de sus ventajas más notables es su capacidad de generalización, resultado de su extensa exposición a datos, lo que les permite abordar diversas tareas sin requerir un entrenamiento especializado para cada aplicación. Esta versatilidad facilita su implementación en diferentes entornos, consolidando su relevancia en múltiples funciones y aplicaciones.

En este trabajo, se explora el componente crucial de razonamiento de los *LLMs*, que les permite determinar objetivos intermedios y finales, identificar las herramientas necesarias para alcanzarlos, y decidir el orden óptimo de acciones a seguir. Gracias a su capacidad para modelar dependencias a largo plazo y comprender el contexto global de un texto, estos modelos pueden tomar decisiones y adaptarse a nuevas situaciones, lo que es esencial para su eficiencia en diversas aplicaciones, como el desarrollo de agentes.

Aunque el concepto de agente autónomo se detalla en la siguiente sección, en resumen, un agente es un sistema inteligente diseñado para tomar decisiones y ejecutar acciones de manera autónoma para alcanzar un objetivo. En este contexto, los *LLMs* son fundamentales para procesar el lenguaje natural, actuando como mecanismo de razonamiento. Al comprender el entorno y el contexto de las consultas, pueden interpretar y responder estas consultas complejas, razonar sobre las acciones disponibles, seleccionar las herramientas adecuadas y coordinar una secuencia de operaciones para lograr objetivos específicos. Los grandes modelos de lenguaje investigados son *GPT*-4 y *Llama* 3 (véase anexo C).

4.1.1. *GPT*-4

El modelo *GPT*-4[1] (*Generative Pre-trained Transformer*), desarrollado por *OpenAI*, representa un avance significativo en el campo de los *LLMs*. En este trabajo se emplea la versión `gpt-4`, específicamente el modelo `gpt-4-0613`, conocido por su alto rendimiento en razonamiento, precisión y generación de texto, aunque conlleva un coste económico.

Reconocido como uno de los modelos de lenguaje más avanzados, *GPT*-4 se aplica en el ámbito de los agentes autónomos para tareas de razonamiento y toma de decisiones, permitiendo a estos agentes interpretar consultas complejas y generar respuestas precisas. En la plataforma *LangChain*, la integración de *GPT*-4 se realiza mediante la clase de conversación de *OpenAI*, utilizando el comando `llm = ChatOpenAI(model="gpt-4", temperature=0)`.

4.1.2. *Llama* 3

Llama 3, desarrollado por *Meta*, es un destacado modelo de lenguaje para la generación de texto, conocido por su notable capacidad de comprensión. En este trabajo, se ha implementado la versión de 70,000 millones de parámetros (`llama3-70b-8192`[2]), optimizada específicamente para aplicaciones de diálogo, donde ha superado a numerosos modelos de código abierto en términos de rendimiento.

Este modelo es particularmente eficaz como asistente de conversación. *Llama* 3 permite integrarse en agentes autónomos, mejorando así su capacidad para procesar el lenguaje y facilitando interacciones más naturales y efectivas con los usuarios. A través de la plataforma *Groq*[3], que emplea unidades de procesamiento especializadas conocidas como *LPUs* (*Language Processing Units*) para acelerar la inferencia de inteligencia artificial, es posible conectarlo con herramientas como *LangChain*, dedicadas al desarrollo de agentes. *Groq* está diseñada para ejecutar estos modelos con alta eficiencia, ofreciendo velocidad, baja latencia y escalabilidad, lo que la convierte en una solución ideal para aplicaciones en tiempo real. Los desarrolladores pueden generar una clave *API* para integrar estos modelos en sus aplicaciones, facilitando su implementación en entornos de producción. Para optimizar su uso y mejorar la seguridad, se configura la clave *API* como una variable de entorno, eliminando la necesidad de incluirla en cada solicitud y reduciendo el riesgo de exposición accidental en el código fuente. La integración de *Llama* 3 en *LangChain*, se realiza mediante la clase de conversación de *Groq* con el comando: `llm = ChatGroq(temperature=0, model_name="llama3-70b-8192")`.

[1] *GPT*-4. Informe técnico disponible en: https://openai.com/index/gpt-4-research/
[2] *Llama* 3. Disponible en: https://huggingface.co/meta-llama/Meta-Llama-3-70B-Instruct
[3] *Groq*. Documentación disponible en: https://groq.com/

4.2. Agentes en *LangChain*

LangChain[4] es una biblioteca de código abierto que facilita la integración de grandes modelos de lenguaje con una amplia gama de componentes y herramientas, permitiendo a los desarrolladores crear aplicaciones sofisticadas y escalables. Además de ser compatible con *APIs* externas, lo que habilita el acceso a recursos adicionales de información y procesamiento.

Un agente autónomo es un sistema que ejecuta tareas de forma independiente, utilizando un *LLM* para gestionar el flujo de control de una aplicación, es decir, la secuencia de instrucciones que el agente debe seguir para cumplir sus objetivos. Estos agentes gestionan la interacción entre modelos de lenguaje, herramientas y procesos, adaptándose al contexto y a los objetivos definidos. La arquitectura modular de *LangChain* otorga a los agentes acceso a herramientas específicas y les permite emplear plantillas de mensajes para guiar sus interacciones, lo que les capacita para manejar tareas complejas sin necesidad de intervención humana constante.

La planificación en agentes autónomos se estructura en dos niveles: a corto plazo, para la ejecución inmediata de acciones, y a largo plazo, para la definición de secuencias que completen tareas complejas. Esta organización permite a los agentes descomponer tareas amplias en subobjetivos más manejables.

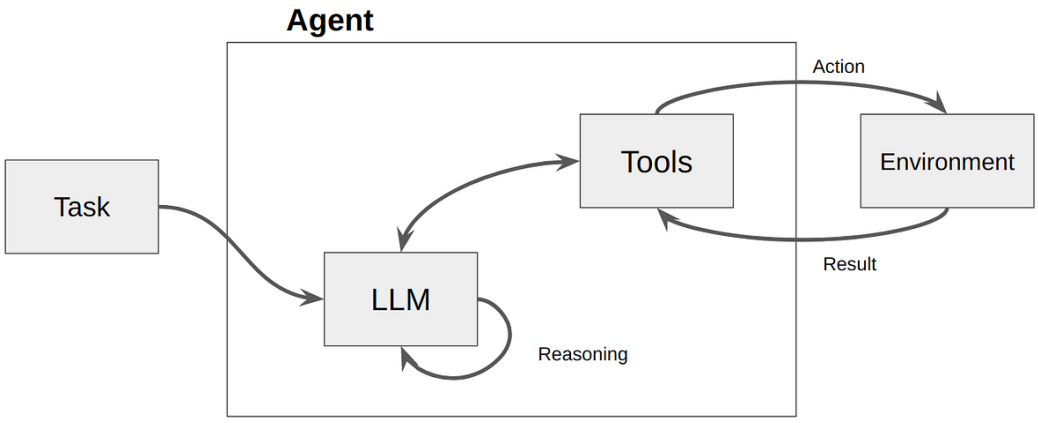


Figura 4.1: Flujo de trabajo de un agente autónomo basado en un *LLM*.

Un enfoque relevante en este contexto es el "*Chain of Thought*" (*CoT*), que se refiere a la capacidad de los agentes para razonar de manera secuencial, mejorando así su eficacia en la resolución de problemas al hacer explícitos los pasos intermedios y las decisiones que guían el proceso cognitivo [35]. Esta técnica resulta particularmente valiosa cuando se enfrentan a tareas que requieren múltiples operaciones encadenadas.

[4] *LangChain*. Documentación disponible en: https://www.langchain.com/

LangChain emplea mecanismos como el "*function calling*", que permite la invocación de funciones específicas mediante solicitudes estructuradas en *JSON* (*JavaScript Object Notation*) que incluyen el nombre de la función y los parámetros requeridos, facilitando la selección precisa de acciones. Además, su arquitectura cognitiva optimiza la resolución de tareas complejas, organizando la información y las herramientas de manera eficiente, apoyándose en el razonamiento secuencial que promueve el enfoque de *CoT*.

Primero, se configura el modelo de lenguaje, como *GPT*-4 y *Llama* 3, estableciendo el parámetro de temperatura a 0 para que las respuestas generadas sean determinísticas. La temperatura es un parámetro que controla la aleatoriedad de las predicciones del modelo; un valor de 0 indica que el modelo genere siempre la misma respuesta para una entrada dada. Dado que se espera que el sistema global funcione como una interacción continua entre el usuario y el agente, se ha optado por emplear un modelo de conversación que utiliza mensajes de *chat*[5] como entrada y devuelve respuestas en formato de *chat*.

LangChain ofrece integraciones con muchos proveedores de modelos y expone una interfaz que permite interactuar con ellos en diferentes modos. Se establece una plantilla de mensajes que define su rol y la entrada del usuario, reservando un espacio para almacenar los mensajes intermedios generados por el agente. El mensaje inicial del agente indica que actúa como un asistente competente, aunque limitado en su capacidad para acceder a información en tiempo real.

```
prompt = ChatPromptTemplate.from_messages(
    [
        (
            "system",
            """Eres un asistente muy poderoso, pero no conoces los
            ↪  acontecimientos actuales.""",
        ),
        ("user", "{input}"),
        MessagesPlaceholder(variable_name="agent_scratchpad"),
    ]
)
```

A continuación, se definen las herramientas asociadas al agente, que contribuyen a la resolución de consultas: `tools = [user_input, search_query, search_serper]`. Posteriormente, el modelo de lenguaje se vincula con estas herramientas, permitiendo al agente utilizarlas durante su ejecución: `llm_with_tools = llm.bind_tools(tools)`.

[5]En este contexto, un *chat* se refiere al intercambio de mensajes de texto entre el usuario y el agente a través de una interfaz, lo que permite una interacción inmediata entre ambas partes.

```
agent = (
    {
        "input": lambda x: x["input"],
        "agent_scratchpad": lambda x: format_to_tool_messages(
            x["intermediate_steps"]
        ),
    }
    | prompt
    | llm_with_tools
    | ToolsAgentOutputParser()
)
```

La arquitectura del agente se implementa mediante funciones lambda que procesan tanto la entrada del usuario como los mensajes intermedios, convirtiéndolos en un formato adecuado para su posterior utilización. Luego, se aplica una plantilla de mensajes que organiza la información de manera estructurada, y se integra el modelo de lenguaje con las herramientas disponibles para facilitar su interacción. Finalmente, se incorpora un *parser* encargado de analizar y transformar los resultados en acciones o respuestas finales del agente, en un formato más comprensible para su ejecución. Este *parser* utiliza el parámetro `tool_calls`, si está presente, para obtener los nombres y entradas de las herramientas; en caso contrario, se asume que `AIMessage` constituye la salida final. La figura 4.2 ilustra un ejemplo del proceso de formateo de las tareas dirigidas al agente.

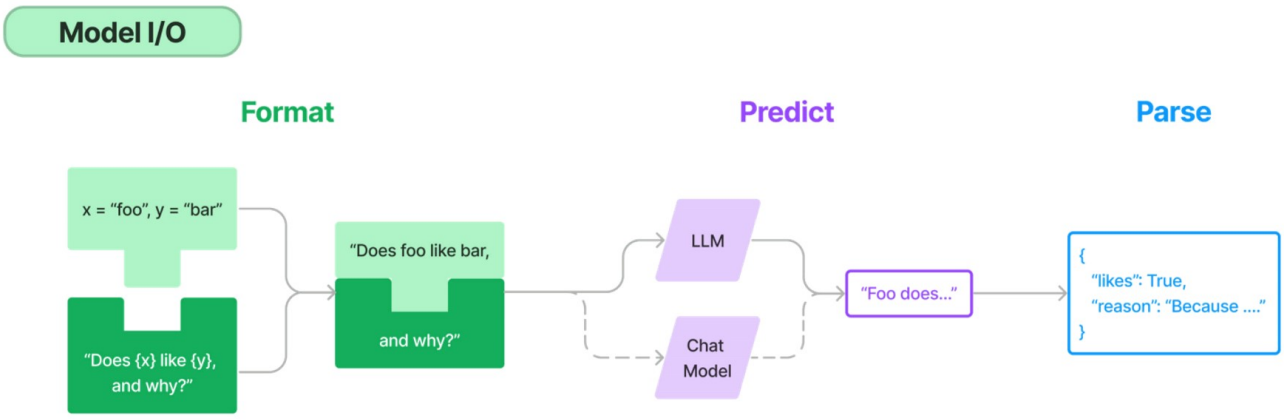


Figura 4.2: Ejemplo de comportamiento del modelo *LLM* en la interacción de entrada y salida gestionada por un agente.[6]

[6]Fuente: *LangChain*.

Por último, el ejecutor del agente se instancia con la siguiente instrucción:

```
agent_executor = AgentExecutor(agent=agent, tools=tools,
↪  return_intermediate_steps=True)
```

Este componente gestiona el uso del agente, determinando de manera eficiente cómo y cuándo deben invocarse las herramientas para cumplir con la tarea asignada. Además, se habilita el argumento `return_intermediate_steps`, lo que permite registrar un historial detallado de los pasos intermedios realizados por el agente durante su ejecución, útil para analizar el proceso de toma de decisiones del agente.

4.2.1. Herramientas Disponibles

Como se mencionó previamente, el agente desarrollado en este trabajo aprende a invocar funciones y *APIs* externas para llevar a cabo tareas específicas y adquirir información adicional que no está incorporada en los pesos del *LLM*.

Las herramientas actúan como interfaces que permiten al agente interactuar con el entorno. Estas se definen mediante la anotación `@tool` y se describen por su nombre, una explicación de sus funcionalidades, y las instrucciones precisas para su uso. Esta información es esencial para que el agente identifique la función correspondiente, ya que cada herramienta representa una acción específica que debe ejecutar.

En esta sección se explican las herramientas disponibles para el agente, resaltando sus propiedades. Entre ellas, se incluyen: ***user_input***, que permite realizar solicitudes de ayuda; ***search_query***, diseñada para realizar búsquedas en colecciones de vídeo utilizando texto o imagen; y ***search_serper***, que facilita la búsqueda en la *web* de imágenes relacionadas con una temática específica. La figura 4.3 ilustra las conexiones entre el agente y estas herramientas, proporcionando un resumen gráfico de su funcionamiento.

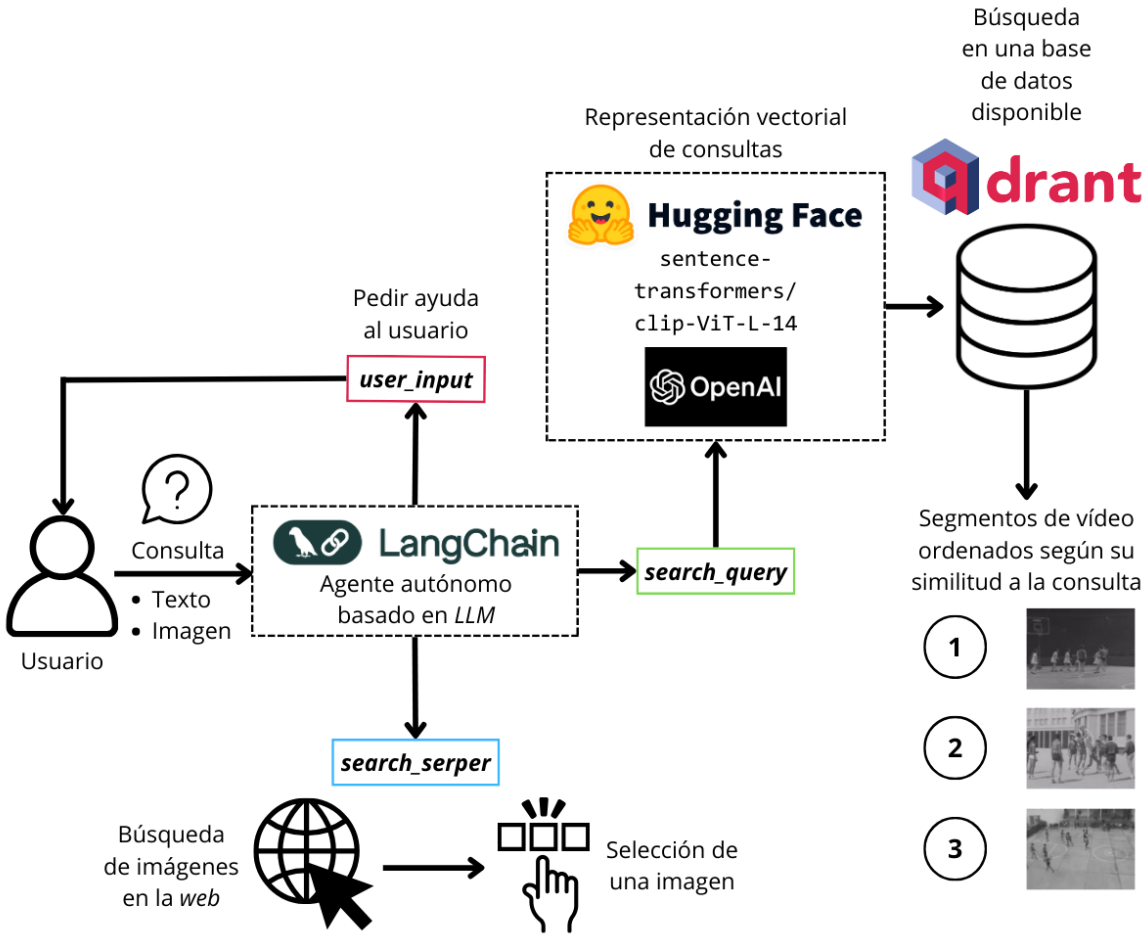


Figura 4.3: Diagrama de interacción y procesamiento de consultas mediante el agente autónomo basado en un *LLM*.

Solicitud de Asistencia al Usuario

Cuando el agente no puede responder a una pregunta o requiere información adicional, ya sea porque no comprende completamente la consulta o porque sus herramientas no son suficientes para resolverla, recurre a la herramienta de entrada del usuario. Esta herramienta permite que el sistema solicite los datos necesarios al usuario y suspenda la ejecución hasta recibir una respuesta, la cual se presenta en la interfaz mediante un *widget*[7] de entrada de texto. De este modo, permite corregir errores o proporcionar aclaraciones sobre la consulta inicial, mejorando la interacción entre el sistema y el usuario y ayudando a prevenir respuestas inexactas.

[7]Un *widget* es un elemento gráfico interactivo en una interfaz que facilita la interacción del usuario con el sistema para la ejecución de acciones específicas o la visualización de datos.

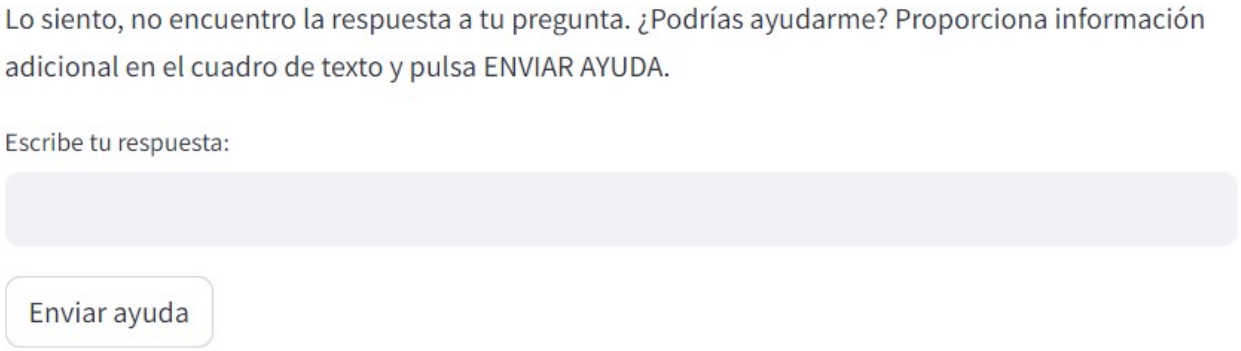


Figura 4.4: Herramienta ***user_input***: Solicitud de asistencia al usuario. Surge cuando el agente no logra interpretar adecuadamente la consulta realizada. Al incorporar las correcciones necesarias para que el agente resuelva la consulta, la respuesta modificada se convierte en la nueva entrada del sistema.

Búsqueda de Vídeos en Colecciones

La siguiente herramienta facilita la realización de búsquedas semánticas en bases de datos vectoriales para recuperar recursos audiovisuales sobre una temática específica, permitiendo al usuario elegir entre consultas basadas en texto o en imágenes. Integra todo el proceso, desde la selección de la modalidad de entrada hasta la presentación organizada de los resultados.

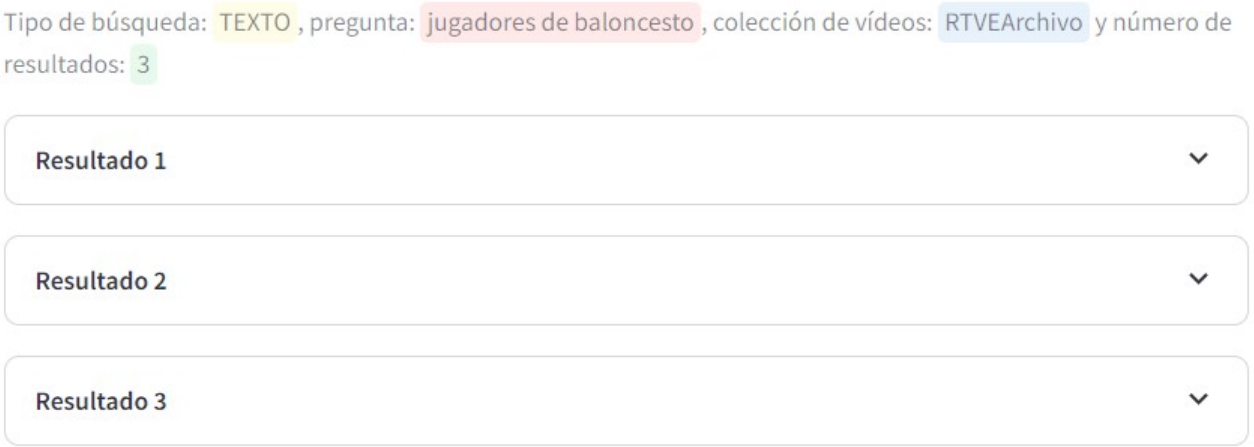


Figura 4.5: Herramienta ***search_query***: Búsqueda de vídeos en colecciones. En este ejemplo, se ha realizado una búsqueda de tres recursos audiovisuales relacionadas con la temática "jugadores de baloncesto" en la colección RTVEArchivo.

Dependiendo del tipo de consulta, la herramienta aplica la transformación correspondiente: si se trata de texto, primero traduce la consulta al inglés y luego genera el vector de características utilizando el codificador de texto del modelo *CLIP*. En el caso de imágenes, se procesa la imagen en formato *PIL Image* y se obtiene su vector de características utilizando el codificador de imágenes. Durante este proceso, el modelo se congela, dado que se está realizando una inferencia, asegurando así la estabilidad de las predicciones.

Ambas modalidades de consulta, texto e imagen, se tratan de manera uniforme en cuanto a su representación en un espacio vectorial compartido, alineándose con las representaciones vectoriales de los vídeos almacenados en la base de datos. El agente se conecta a la base de datos *Qdrant*, donde busca los elementos más similares al *embedding* generado, basándose en la similitud coseno dentro de la colección seleccionada por el usuario. Este proceso permite recuperar los elementos de vídeo más relevantes en función de la consulta, como se ha explicado más detalladamente en la subsección 3.2.2.

Una vez identificados los elementos más similares, la herramienta organiza y presenta los resultados. Se proporciona no solo la información básica y los metadatos de las secuencias de vídeo más similares, sino que también se ilustra la escena específica del vídeo donde se encuentra el contenido relacionado con la consulta (figura 4.6).

Adicionalmente, se genera una representación visual mediante un gráfico *UMAP*, que permite comparar la distribución de todos los fotogramas de la secuencia de vídeo con el vector de la consulta original, y así realizar una búsqueda más fina entre ellos. De este modo, el sistema identifica y presenta el fotograma con la menor distancia euclídea, es decir, el más cercano a la consulta y, por tanto, el más similar según dicha métrica, lo que permite al usuario verificar visualmente la precisión de la búsqueda (figura 4.7).

Finalmente, los resultados se reordenan según las comparaciones en la representación *UMAP*, proporcionando al usuario una perspectiva adicional sobre la relevancia de los resultados obtenidos (figura 4.8).

Esta herramienta permite realizar búsquedas multimodales avanzadas en bases de datos indexadas previamente, explotando la capacidad del modelo *CLIP* para representar tanto contenido visual como textual en un espacio vectorial común, optimizando así la identificación de similitudes entre diferentes tipos de datos. Además, analiza la métrica de similitud coseno, que es especialmente eficaz para búsquedas semánticas, ya que permite identificar no solo coincidencias exactas, sino también relaciones conceptuales o temáticas entre los elementos consultados y los almacenados en la base de datos.

Resultado 1

Información del resultado obtenido durante el proceso de búsqueda en secuencias:

Similitud coseno: 0.2491

Temática: Sports

Título: BALONCESTO

Vídeo: I00785860.mp4

Escena: 0

Tiempo de inicio de la escena: 00:00:00,040

Tiempo de fin de la escena: 00:00:41,040

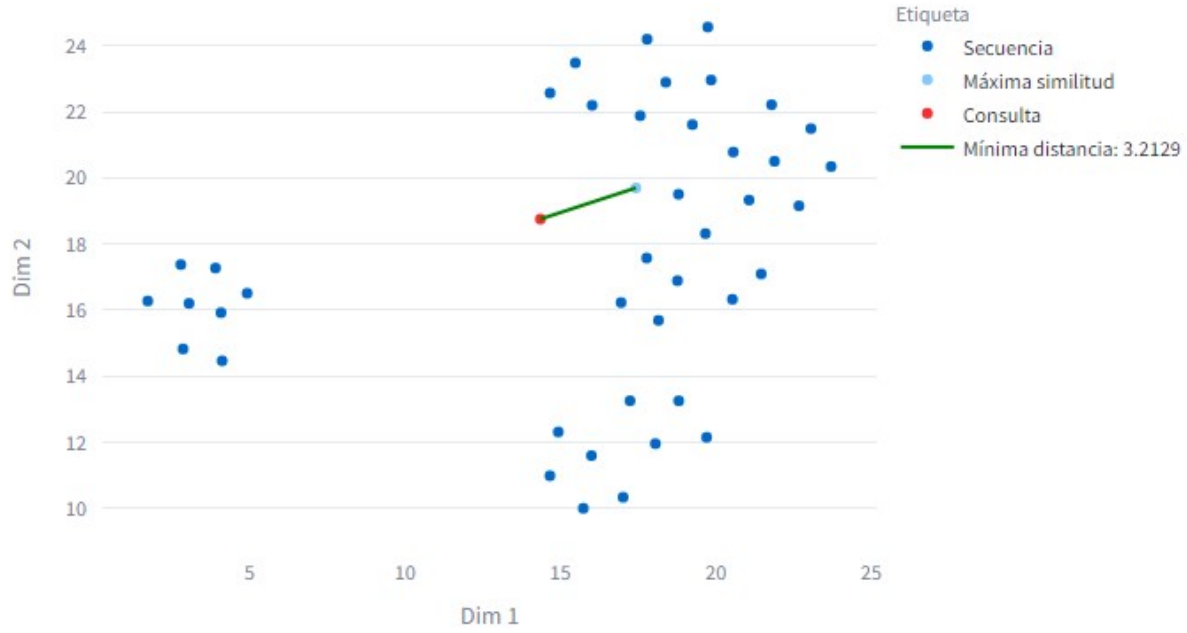
Secuencia de la escena: 0

Vídeo de la escena encontrada en la base de datos.

Figura 4.6: Continuación del ejemplo mostrado en la figura 4.5: Resultado con el mayor valor de similitud coseno entre la representación vectorial de la consulta y el promedio de los vectores que componen la secuencia del vídeo.

Información del resultado derivada del proceso de búsqueda fina dentro de la secuencia:

Representación UMAP de los fotogramas de la secuencia y la consulta realizada.



Fotograma con mínima distancia euclídea: 3.2129

Figura 4.7: Continuación del ejemplo: Representación *UMAP* de los fotogramas correspondientes al resultado 1, junto con el fotograma de menor distancia euclídea.

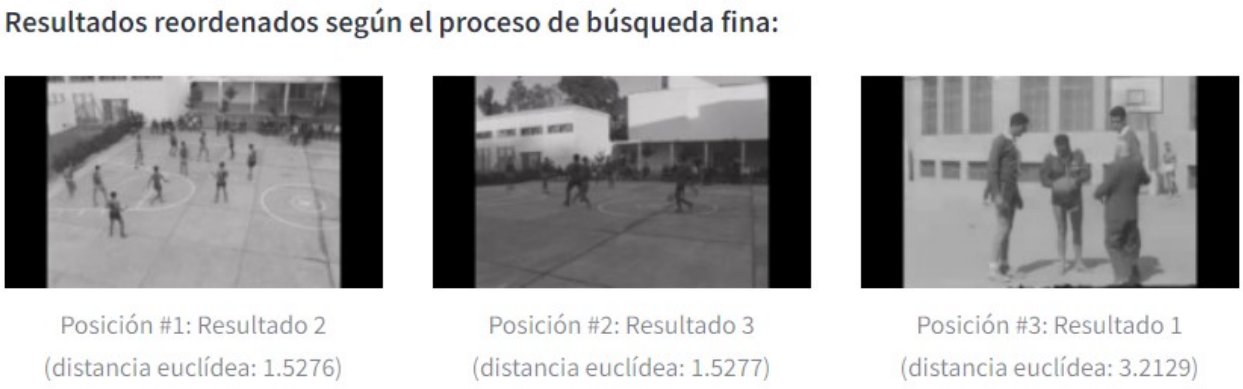


Figura 4.8: Continuación del ejemplo: Reordenamiento de los resultados según la distancia euclídea entre las representaciones *UMAP* de los fotogramas y la consulta.

Búsqueda de Imágenes de Muestra en la *Web*

La búsqueda en la *web* es una de las funcionalidades más versátiles y potentes que los agentes autónomos pueden emplear. Al integrar capacidades de búsqueda dentro del marco de *LangChain*, estos agentes acceden a una vasta cantidad de recursos informativos, lo que mejora significativamente la precisión y relevancia de las respuestas generadas.

En este contexto, destaca la herramienta de búsqueda *Serper*, diseñada específicamente para la recuperación de imágenes en línea sobre temas específicos. Basada en la *API* de *Serper*, esta herramienta aprovecha el motor de búsqueda ultrarrápido de *Google* para recuperar un número determinado de imágenes relevantes, entre 1 y 10, relacionadas con la consulta indicada. Para llevar a cabo esta búsqueda, se utiliza el código `search = GoogleSerperAPIWrapper(type="images")`, que invoca el motor de búsqueda mediante `results = search.results(query)`.

Durante el proceso, se identifican las *URLs* (*Uniform Resource Locators*) de las imágenes más pertinentes, y aquellas cuya dirección es válida se descargan utilizando el comando *cURL*[8] (*Client URL*). Este comando, `curl --retry 5 -o {image_path} {image}`, garantiza la fiabilidad de la descarga, ya que intenta repetirla hasta cinco veces en caso de errores. Las imágenes descargadas se almacenan localmente, y su integridad se verifica mediante su apertura. Solo las imágenes que superan esta verificación se presentan al usuario, y sus direcciones se registran para su uso en futuras tareas.

[8] *cURL* es una herramienta de línea de comandos que permite transferir datos entre un dispositivo y un servidor utilizando diversos protocolos.

Una característica adicional de esta herramienta es la posibilidad de que el usuario seleccione una de las imágenes recuperadas para utilizarla en tareas posteriores, como búsquedas en colecciones de vídeos. Esta funcionalidad incrementa la flexibilidad del sistema, permitiendo una experiencia de usuario más eficiente y personalizada, al tiempo que facilita la obtención de imágenes de muestra para su uso en otras aplicaciones.

Esta herramienta de búsqueda avanzada ofrece la capacidad de recuperar información visual actualizada, algo que el modelo de lenguaje no puede realizar de manera independiente. Así, la integración de esta herramienta con el modelo de lenguaje amplía las capacidades del sistema, ofreciendo soluciones más completas y eficaces para la recuperación de información.

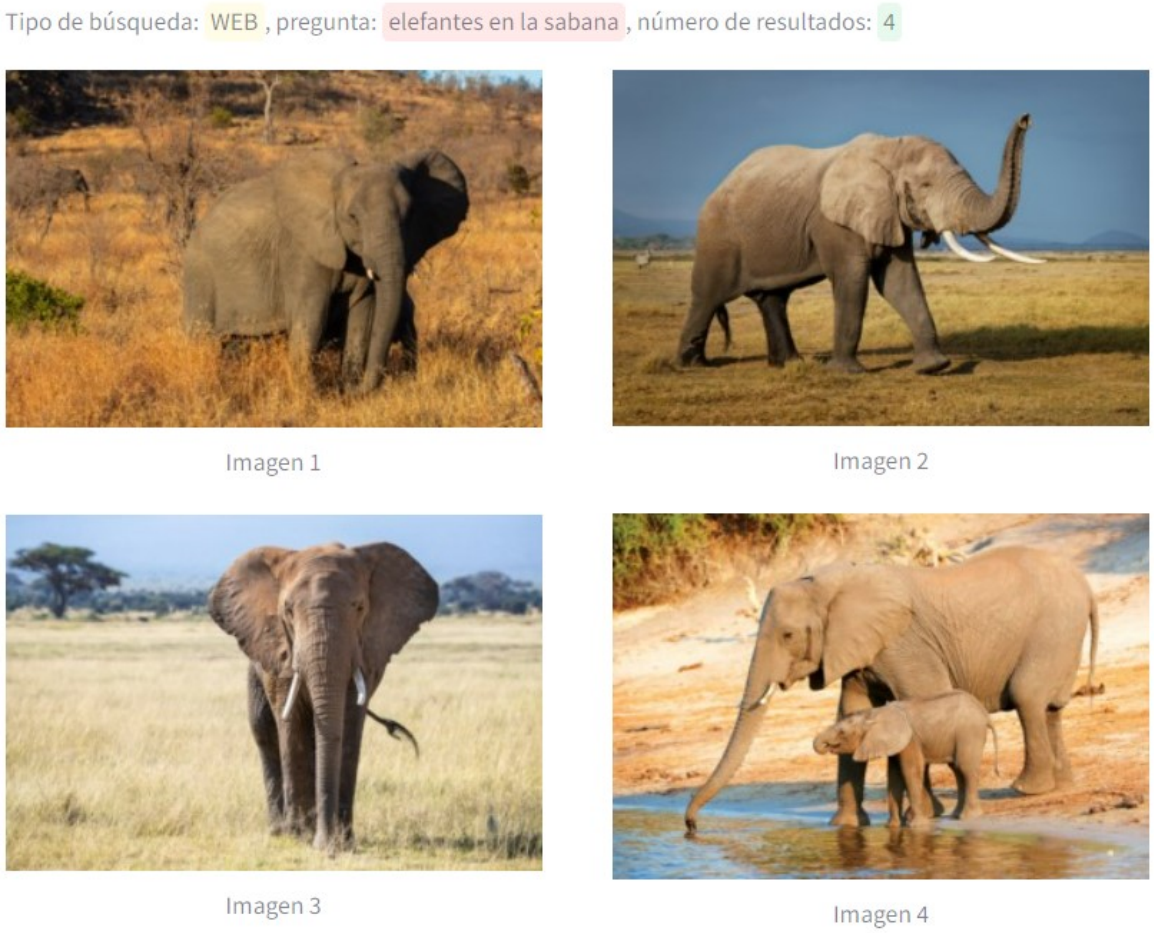


Figura 4.9: Herramienta ***search_serper***: Búsqueda de imágenes de muestra en la *web*. En este ejemplo, se ha realizado una búsqueda de cuatro imágenes relacionadas con la temática "elefantes en la sabana".

4.2.2. Descomposición de Tareas Complejas

La técnica de cadena de pensamiento (*CoT*) mejora significativamente el rendimiento de los modelos en tareas complejas al instruirlos a descomponer las tareas difíciles en subtareas más manejables, facilitando así una mejor comprensión del proceso de razonamiento paso a paso.

El agente autónomo procesa la entrada del usuario en un flujo continuo, gestionando acciones y observaciones de manera secuencial e iterativa para resolver la consulta. Para habilitar la transmisión en tiempo real de los pasos necesarios para alcanzar el resultado final, se utiliza el comando `agent_executor.stream({"input": input})`, el cual estructura el proceso mediante una evaluación secuencial y condicional de los fragmentos de datos, según su contenido.

Dato	**Contenido**
Acciones	`"actions"`: Mensajes de *chat* correspondientes a la invocación de la acción.
Observaciones	`"steps"`: Registro completo de las acciones del agente hasta el momento, incluida la acción actual y sus observaciones (mensaje de *chat* con las respuestas de la invocación de funciones).
Respuesta final	`"output"`: Mensajes de *chat* con el resultado final.

Tabla 4.1: Descripción del contenido de cada tipo de fragmento de datos involucrado en el proceso de razonamiento del agente.

Durante este proceso, el agente alterna entre la ejecución de acciones y la observación de los resultados obtenidos, repitiendo este ciclo hasta alcanzar el objetivo deseado (véase un ejemplo en el anexo D). En cada iteración, los fragmentos de datos se clasifican en tres categorías principales: acciones, observaciones y resultados finales.

Si un fragmento contiene acciones, estas se añaden a una lista de tareas, generándose un mensaje que especifica la herramienta a emplear y la entrada correspondiente. Cuando el fragmento corresponde a una observación, el sistema presenta el resultado proporcionado por la herramienta utilizada. Este proceso continúa hasta que se obtiene un resultado final, momento en el cual concluye la intervención del agente.

Al finalizar, se muestra un resumen de las herramientas utilizadas a lo largo del procedimiento, indicando el éxito de la operación y marcando la tarea como completada. Para iniciar una nueva consulta, el usuario debe repetir el mismo procedimiento.

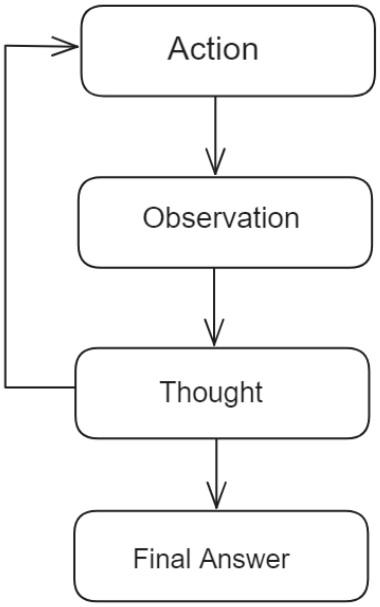


Figura 4.10: Proceso de razonamiento para la toma de decisiones y ejecución de tareas.

En caso de que un fragmento de datos no pertenezca a ninguna de las categorías previstas (acciones, observaciones o resultados finales), se genera una excepción para manejar el error, lo que sugiere que los datos son inválidos o no se ajustan al flujo esperado, interrumpiendo la operación del agente.

Este enfoque estructurado garantiza que las acciones del agente queden registradas de manera precisa y que tanto los resultados como las excepciones se gestionen eficazmente. Así, el proceso de resolución de consultas mediante el concepto *CoT* demuestra la capacidad de razonamiento del agente, ofreciendo una solución robusta y eficiente en la ejecución de tareas complejas.

Capítulo 5

Bases de Datos

La fiabilidad de los sistemas de recuperación de información es un aspecto complejo de medir, como se discutirá más adelante. No obstante, es imprescindible contar con un dato objetivo que permita evaluar el funcionamiento del sistema. Por consiguiente, para la implementación de las etapas previamente descritas en este trabajo y para la evaluación global del mismo, se han utilizado dos bases de datos etiquetadas: *MSR-VTT* y RTVEArchivo.

5.1. *MSR-VTT*: *Microsoft Research Video to Text*

La base de datos *MSR-VTT* (*Microsoft Research Video to Text*) [36] fue creada por *Microsoft Research* para facilitar la descripción de vídeos mediante lenguaje natural, abarcando una gran escala de datos, diversidad de categorías, y riqueza en contenido y descripciones.

MSR-VTT se desarrolló seleccionando las 257 principales consultas de búsqueda de vídeos en un motor comercial, cubriendo 20 categorías[1] generales. De cada consulta, se extrajeron los 118 mejores resultados, eliminando vídeos duplicados, cortos o de baja calidad visual. Posteriormente, se segmentaron los vídeos mediante histogramas de color, se filtraron y luego se seleccionaron aleatoriamente 10,000 *clips*[2]. Cada *clip* tiene una duración de entre 10 y 30 segundos, acumulando un total de 41.2 horas de contenido.

Cada *clip* de vídeo en color fue descrito con aproximadamente 20 oraciones naturales en inglés por múltiples anotadores mediante *Amazon Mechanical Turk* (véase tabla 5.1).

[1] Estas 20 categorías incluyen música, personas, juegos, deportes (acciones), noticias (eventos/política), educación, programas de televisión, películas (comedia), animación, vehículos (autos), tutoriales, viajes, ciencia (tecnología), animales (mascotas), niños (familia), documentales, comida (bebida), cocina, belleza (moda) y publicidad.

[2] Un *clip* de vídeo se define como un segmento breve y continuo de un vídeo que captura una escena específica.

Se eliminaron las oraciones duplicadas y las demasiado cortas, obteniendo un total de 200,000 pares de *clip*-oración, proporcionando un rico conjunto de datos textuales que reflejan con precisión las escenas de los vídeos.

	1. *A black and white horse runs around.* 2. *A horse galloping through an open field.* 3. *A horse is running around in green lush grass.* 4. *There is a horse running on the grassland.* 5. *A horse is riding in the grass.*
	1. *A woman giving speech on news channel.* 2. *Hillary Clinton gives a speech.* 3. *Hillary Clinton is making a speech at the conference of mayors.* 4. *A woman is giving a speech on stage.* 5. *A lady speak some news on TV.*
	1. *A child is cooking in the kitchen.* 2. *A girl is putting her finger into a plastic cup containing an egg.* 3. *Children boil water and get egg whites ready.* 4. *People make food in a kitchen.* 5. *A group of people are making food in a kitchen.*

Tabla 5.1: Ejemplos de los *clips* y las oraciones etiquetadas del conjunto de datos *MSR-VTT*. Se proporcionan tres muestras, cada una contiene cuatro fotogramas para representar el *clip* de vídeo y cinco oraciones etiquetadas manualmente.

Para fines de entrenamiento, validación y prueba, los *clips* se distribuyeron en función de las consultas de búsqueda originales. Así, se asignaron 6,513 *clips* para entrenamiento, 2,990 para prueba y 497 para validación, con una división de 65 %, 30 % y 5 % respectivamente. En el trabajo en cuestión, se utiliza exclusivamente el subconjunto de prueba. Esto permite un manejo de datos adecuado para la evaluación del sistema de búsqueda y recuperación de información en condiciones controladas y comparables.

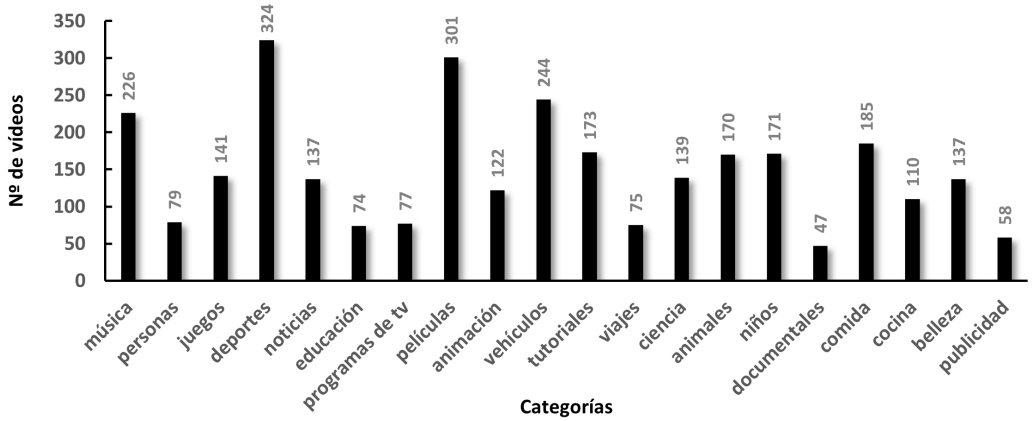


Figura 5.1: Distribución de categorías de vídeo en el subconjunto de prueba del conjunto de datos *MSR-VTT*.

En conclusión, la base de datos *MSR-VTT* se consolida como un recurso esencial y de referencia en la intersección de vídeo y lenguaje, proporcionando un banco de pruebas sólido para la investigación y el desarrollo en el ámbito del análisis de vídeo. Su versatilidad y la variedad de categorías que abarca destacan especialmente en sistemas de búsqueda y recuperación de información, permitiendo una exploración amplia y diversa de posibilidades.

5.2. Archivo de RTVE (Radio Televisión Española)

RTVEArchivo es una base de datos que contiene fragmentos del Archivo de RTVE[3] (Radio Televisión Española), una de las colecciones audiovisuales más importantes de España por su volumen y la riqueza de los documentos que alberga. Este archivo digital almacena más de un millón de horas de emisión, cubriendo más de 50 años de historia. Además de conservar material audiovisual, desempeña un papel vital en la difusión del patrimonio cultural e histórico de España, ofreciendo acceso a contenido significativo tanto a nivel nacional como internacional. La digitalización de estos fondos constituye un proceso esencial y continuo, diseñado para prevenir la pérdida de esta invaluable memoria histórica. Esta iniciativa busca asegurar la accesibilidad de los documentos audiovisuales, actuando como un testimonio duradero de la historia nacional.

RTVEArchivo reúne una colección de 199 vídeos con duraciones que oscilan entre 17 segundos y 6.5 minutos. Cada vídeo está acompañado de un título y hasta tres observaciones, como se muestra en la tabla 5.2. Las etiquetas asociadas han sido elaboradas por profesionales de la documentación audiovisual, garantizando una alta calidad y precisión en la caracterización del contenido, reflejando la dedicación en la construcción de esta base de datos.

[3]RTVE. Página *web* oficial: https://www.rtve.es/

MOTOCROSS: *COMPETITION AND PRIZE DELIVERY.*

1. *People with motorcycles.*
2. *Extreme sports.*
3. *Motocross racing.*

Tabla 5.2: Muestra de un vídeo de RTVEArchivo, representada con cuatro fotogramas y acompañada de su título y observaciones traducidas al inglés.

A diferencia de la base de datos *MSR-VTT*, que presenta vídeos en color, RTVEArchivo se compone mayormente de vídeos en blanco y negro, muchos de ellos históricos y de gran antigüedad. Esta particularidad resalta la singularidad y el valor patrimonial del archivo de RTVE.

Asimismo, algunos vídeos incluyen especificaciones temáticas que facilitan su categorización. La figura 5.2 ilustra la distribución temática de los vídeos.

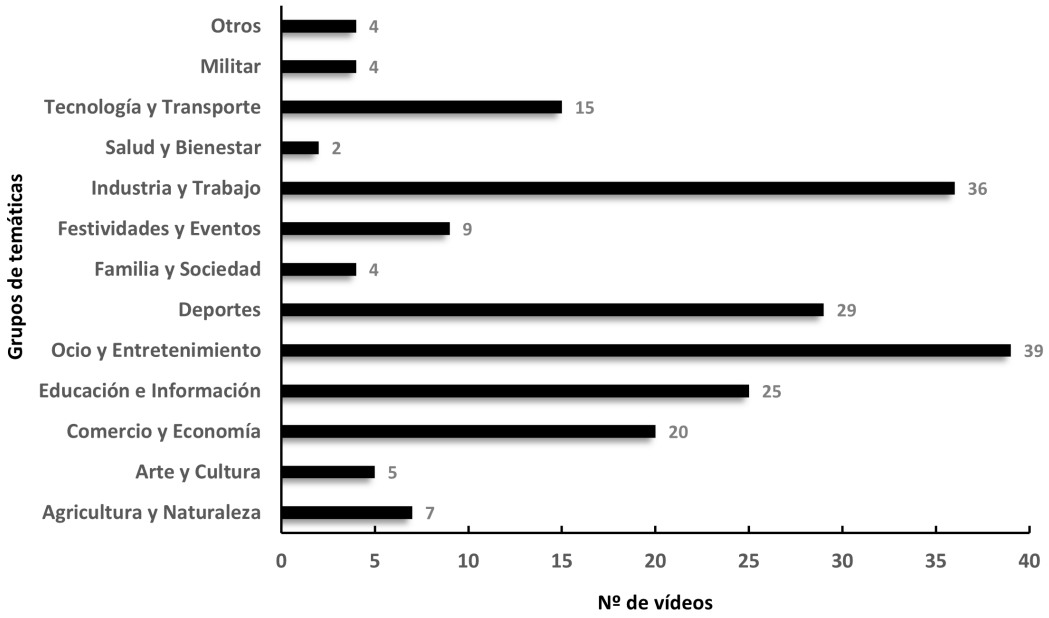


Figura 5.2: Distribución temática de los vídeos en el conjunto de datos RTVEArchivo.

En resumen, RTVEArchivo es un recurso excepcional para la investigación académica y aplicaciones prácticas, especialmente en contextos históricos. Su colección es ideal para evaluar sistemas de búsqueda y recuperación, así como para la conservación y remasterización de material audiovisual.

Capítulo 6

Metodología Experimental y Resultados

6.1. Evaluación de la Búsqueda Semántica

Evaluar la fiabilidad de un sistema de búsqueda semántica es un desafío complejo debido a la naturaleza de su funcionamiento, que se basa en la interpretación del significado de los términos en lugar de su coincidencia literal. Este enfoque ofrece ventajas significativas en comparación con otros tipos de búsquedas, como la capacidad de realizar consultas con alta granularidad y recuperar elementos que no están explícitamente etiquetados con una temática concreta. Sin embargo, también presenta dificultades a la hora de medir objetivamente su rendimiento, ya que, en presencia de datos con características semánticamente similares, pero etiquetados de manera diferente, diversos resultados pueden considerarse válidos. Por esta razón, en este trabajo se ha decidido analizar el peor caso posible, validando como respuestas correctas únicamente aquellas que coinciden exactamente con la etiqueta específica. Esto implica que los resultados reflejen un límite mínimo, ya que se descartan recursos similares que, aunque posiblemente semánticamente correctos, corresponden a otros vídeos distintos al de la descripción proporcionada.

Los experimentos se han realizado utilizando la herramienta de búsqueda de vídeos ***search_query*** en dos bases de datos: RTVEArchivo y *MSR-VTT*, previamente caracterizadas en el capítulo anterior. Los vídeos de estas colecciones están etiquetados con diversas descripciones de su contenido, las cuales se han empleado como consultas para el sistema. En el caso de RTVEArchivo, cada vídeo cuenta con entre una y tres descripciones, mientras que los vídeos de *MSR-VTT* tienen 20 descripciones disponibles.

La herramienta se configura para conectarse a las bases de datos de *Qdrant* y recuperar los 50 resultados más similares a la consulta formulada. Los diferentes experimentos se basan en la construcción de las consultas, que incluyen combinaciones aleatorias de descripciones y, en algunos casos, el título del vídeo. Para la formación del vector de consulta, se han empleado dos métodos de análisis: *Text Aggregation* (*TA*) y *Mean Average* (*MA*). El método *TA* consiste en concatenar las descripciones para formar una consulta densa, truncándola si excede el límite de *tokens* permitido por el modelo *CLIP*, y luego procesarla mediante el codificador de texto para obtener el *embedding* de la consulta. El método *MA*, por su parte, codifica las descripciones individualmente y calcula el promedio de dichas representaciones.

Como métrica de fiabilidad, se ha registrado la posición del resultado correcto en la lista de resultados devueltos, permitiendo recopilar estadísticas de *recall*. Esta métrica mide la fracción de instancias relevantes (en este caso, recursos audiovisuales) que son recuperadas en diferentes puntos de corte (*top* 1, 5, 10 y 50). Un recurso se considera recuperado si aparece en la lista de resultados dentro del rango especificado; de lo contrario, no se cuenta como recuperado. El *recall* se calcula mediante la siguiente fórmula:

$$Recall = \frac{\text{Instancias relevantes recuperadas}}{\text{Todas las instancias relevantes}} \tag{6.1}$$

Base de datos **RTVEArchivo**				
Experimento	***R*@1**	***R*@5**	***R*@10**	***R*@50**
1desc	34.7	59.8	67.3	87.9
3desc+*MA*	39.7	66.3	74.4	91.5
3desc+*TA*	42.7	69.3	77.4	91.5
1desc+tit+*MA*	43.7	71.9	80.4	93.5
1desc+tit+*TA*	49.2	76.4	81.9	95.0
3desc+tit+*MA*	50.3	71.9	81.4	94.0
3desc+tit+*TA*	**51.3**	**72.4**	81.4	96.0
Base de datos ***MSR-VTT***				
Experimento	***R*@1**	***R*@5**	***R*@10**	***R*@50**
1desc	23.6	44.5	53.7	73.9
3desc+*MA*	39.2	63.2	73.3	89.4
3desc+*TA*	42.1	68.3	77.0	92.7
10desc+*MA*	**52.1**	**76.2**	84.2	95.7

Tabla 6.1: Resultados de *recall* (%) en búsquedas semánticas para RTVEArchivo y *MSR-VTT* a través de diferentes experimentos.

Los resultados de los experimentos aparecen resumidos en la tabla 6.1, que proporciona una visión detallada del rendimiento del sistema en términos de fiabilidad en distintos escenarios. En la tabla, "Xdesc" representa el número de descripciones aleatorias que componen la consulta (donde X puede ser 1, 3 o 10), y "tit" indica la inclusión del título del vídeo en la consulta. Estos datos son fundamentales para comprender las capacidades y limitaciones del sistema en contextos reales, donde una misma etiqueta puede referirse a diferentes contenidos semánticamente similares. Los resultados se presentan en porcentajes para reflejar la proporción de consultas en las que se ha recuperado el resultado correcto en cada experimento.

6.2. Evaluación del Agente Autónomo

Debido a la falta de guiones de prueba, no ha sido posible realizar una evaluación objetiva del funcionamiento del agente autónomo. Sin embargo, se han obtenido los siguientes resultados a partir de una evaluación subjetiva.

Al concatenar diferentes consultas, el agente ha demostrado ser capaz de descomponer las tareas en el orden correcto, seleccionando adecuadamente las herramientas y parámetros requeridos. No obstante, dado que las consultas deben incluir todos los parámetros necesarios para el uso de cada herramienta, en casos donde se especifica correctamente la tarea, el agente puede identificarla pero, si no se proporcionan todos los detalles necesarios, tiende a "alucinar", es decir, a inventar información infundada para suplir la falta de algún parámetro específico. Este fenómeno ocurre con mayor frecuencia cuando el usuario no especifica la colección en la que desea realizar la búsqueda. Para mitigar estos errores, se han añadido excepciones en el código fuente.

En cuanto a las diferencias entre ambos modelos de lenguaje, se observaron pocas discrepancias. Ambos cumplieron con eficacia su función de razonamiento. Una diferencia es que *GPT*-4 suele comprender mejor el proceso de pensamiento de principio a fin, mientras que *Llama* 3, en ocasiones, continúa "pensando" después de haber completado su tarea. Por esta razón, se le ha programado para que finalice su ejecución una vez que la consulta haya concluido, evitando así entrar en un bucle innecesario.

Capítulo 7

Conclusiones y Líneas Futuras

Las conclusiones expuestas en este capítulo responden, junto a los resultados obtenidos y a los desafíos enfrentados a lo largo del desarrollo del trabajo, a los objetivos planteados inicialmente en la introducción.

Integración de Contenido Textual y Visual:

La arquitectura del modelo *CLIP* ha demostrado ser eficaz en la comparación de texto e imagen dentro de un espacio vectorial común. Esta capacidad permite generar representaciones que capturan las características semánticas de ambos tipos de contenido, facilitando la realización de búsquedas multimodales basadas en su significado intrínseco.

Investigación y Análisis de Tecnologías de Segmentación Semántica:

El algoritmo *KTS* y la combinación de *UMAP* y *HDBSCAN* han demostrado ser herramientas eficaces para la segmentación semántica de vídeos, permitiendo reducir el volumen de datos almacenados sin comprometer la integridad ni la relevancia de los resultados. La elección de *UMAP+HDBSCAN* se justifica no solo por su eficiencia en la indexación del contenido, sino también por su mayor rapidez de procesamiento, lo que constituye una ventaja operativa significativa al agilizar el análisis de los vídeos.

Desarrollo y Evaluación de un Sistema de Búsquedas Semánticas:

La herramienta ***search_query*** ha demostrado ser eficaz en la recuperación de recursos audiovisuales. A pesar de las dificultades encontradas al evaluar la fiabilidad del sistema, el análisis del peor caso posible muestra resultados favorables. Se concluye que el sistema es efectivo para realizar búsquedas semánticas y tiene un gran potencial cuando las consultas son adecuadamente optimizadas. Se ha observado una mejora significativa en la precisión del sistema al utilizar consultas formadas con un mayor número de descripciones, esto es, más detalladas. Además, el método *TA* generalmente produce mejores resultados que *MA*, especialmente en las métricas de *recall* $R@1$ y $R@5$.

Implementación de un Agente Autónomo:

Los grandes modelos de lenguaje *GPT*-4 y *Llama* 3 han demostrado su capacidad para habilitar a los agentes autónomos en la comprensión de entradas complejas, descomponiéndolas en pasos más simples y desarrollando flujos de trabajo eficaces basados en la técnica *CoT*. Estos agentes pueden identificar, organizar y coordinar las herramientas y acciones necesarias para resolver tareas difíciles de manera eficiente. Además, el acceso gratuito a *Llama* 3 lo hace una opción atractiva para la implementación del agente, minimizando costos sin sacrificar su funcionalidad.

Finalmente, se cumple el objetivo principal de este trabajo:

Desarrollo de un Asistente para el Proceso de Ideación:

La interfaz descrita en el anexo E, muestra la aplicación desarrollada, que integra todas las tecnologías estudiadas. Esta interfaz permite a los usuarios, en particular a profesionales del periodismo, recuperar imágenes y recursos contextualmente adecuados que les inspiren en el proceso creativo de la composición y generación de nuevas piezas audiovisuales. La simplicidad del sistema, permite la introducción de una imagen o un guion predefinido como consulta, para mejorar notablemente la experiencia del usuario en la obtención de ideas.

Como líneas futuras, se propone la incorporación de un módulo que reorganice los resultados candidatos según su relevancia para la consulta, empleando un *reranker* multimodal que actúe como un segundo filtro. Este módulo compararía imágenes y texto para determinar si los recursos audiovisuales recuperados, aunque semánticamente similares, son verdaderamente pertinentes para la consulta específica. Esta mejora aumentaría la robustez del sistema, incrementando su precisión y utilidad al priorizar los resultados más relevantes. Además, se plantea realizar una evaluación del agente autónomo en un entorno periodístico de televisión, lo que permitiría transitar hacia una prueba de concepto sólida y específica para este contexto.

Capítulo 8

Bibliografía

[1] Amit Singhal and I. Google. Modern Information Retrieval: A Brief Overview. *IEEE Data Engineering Bulletin*, 24:35–43, January 2001.

[2] Herman Hollerith. An electric tabulating system. *The Quarterly, Columbia University School of Mines*, X(16):238–255, April 1889.

[3] Michael Buckland. Zeiss ikon's "statistical machine". *Zeiss Historica Journal*, 17(1):6–7, 1995.

[4] Vannevar Bush. As We May Think. *The Atlantic Monthly*, 176(1):101–108, July 1945.

[5] Calvin N. Mooers. *The theory of digital handling of non-numerical information and its implications to machine economics*. Number 48 in Zator Technical Bulletin. Zator Co., Boston, 1950.

[6] Hans Peter Luhn. A Statistical Approach to Mechanized Encoding and Searching of Literary Information. *IBM J. Res. Dev.*, 1:309–317, 1957.

[7] Cyril W. Cleverdon. The significance of the cranfield tests on index languages. In *Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '91, page 3–12, New York, NY, USA, 1991. Association for Computing Machinery.

[8] G. Salton and M. E. Lesk. The SMART automatic document retrieval systems—an illustration. *Commun. ACM*, 8(6):391–398, Jun 1965.

[9] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Commun. ACM*, 18:613–620, 1975.

[10] S.E. Robertson and K. Sparck Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27(3):129–146, May 1976.

[11] C. J. van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, London, UK, 2nd edition, 1979.

[12] Tim Berners-Lee. Information management: A proposal. *CERN*, March 1989.

[13] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[14] Donna K. Harman. Overview of the first Text REtrieval Conference (TREC-1). In *Proceedings of the First Text REtrieval Conference (TREC-1)*, pages 1–20. NIST Special Publication 500-207, March 1993.

[15] Parminder Kaur, Husanbir Pannu, and Avleen Malhi. Comparative analysis on cross-modal information retrieval: A review. *Computer Science Review*, 39:100336, 02 2021.

[16] Ramón Salaverría. Periodismo digital: 25 años de investigación. Artículo de revisión. *Profesional de la información*, 28(1), Enero 2019.

[17] Amaya Noain-Sánchez. Addressing the Impact of Artificial Intelligence on Journalism: the perception of experts, journalists and academics. *Communication & Society*, 35(3):105–121, 2022.

[18] Rick Prelinger. Archives and Access in the 21st Century. *Cinema Journal*, 46:114–118, March 2007.

[19] Mark Sanderson and Bruce Croft. The History of Information Retrieval Research. *Proceedings of the IEEE*, 100(13):8, 2012.

[20] M. Taube, C. D. Gull, and I. S. Wachtel. Unit Terms in Coordinate Indexing. *American Documentation*, 3(4):213–218, January 1952.

[21] Diksha Khurana, Aditya Koli, Kiran Khatter, and Sukhdev Singh. Natural language processing: state of the art, current trends and challenges. *Multimedia Tools and Applications*, 82(3):3713–3744, 2023.

[22] Wang Wei, Payam Barnaghi, and Andrzej Bargiela. Search with meanings: An overview of semantic search systems. *International Journal of Communications of SIWN*, 3, 01 2008.

[23] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Zhao, Zhewei Wei,

and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18, March 2024.

[24] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning Transferable Visual Models From Natural Language Supervision. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR, 18–24 Jul 2021.

[25] Alexander Kolesnikov, Alexey Dosovitskiy, Dirk Weissenborn, Georg Heigold, Jakob Uszkoreit, Lucas Beyer, Matthias Minderer, Mostafa Dehghani, Neil Houlsby, Sylvain Gelly, Thomas Unterthiner, and Xiaohua Zhai. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*, 2021.

[26] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[27] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[28] Danila Potapov, Matthijs Douze, Zaid Harchaoui, and Cordelia Schmid. Category-Specific Video Summarization. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 540–555, Cham, 2014. Springer International Publishing.

[29] Gideon Schwarz. Estimating the Dimension of a Model. *The Annals of Statistics*, 6(2):461–464, 1978.

[30] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. UMAP: Uniform Manifold Approximation and Projection. *Journal of Open Source Software*, 3(29):861, 2018.

[31] Marina Meilă and Hanyu Zhang. Manifold Learning: What, How, and Why. *Annual Review of Statistics and Its Application*, 11:393–417, 2024.

[32] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. Density-Based Clustering Based on Hierarchical Density Estimates. In Jian Pei, Vincent S. Tseng, Longbing Cao, Hiroshi Motoda, and Guandong Xu, editors, *Advances in Knowledge Discovery and Data Mining*, pages 160–172, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[33] Yu A Malkov and Dmitry A Yashunin. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2018.

[34] Yutao Zhu, Huaying Yuan, Shuting Wang, Jiongnan Liu, Wenhan Liu, Chenlong Deng, Haonan Chen, Zhicheng Dou, and Ji-Rong Wen. Large Language Models for Information Retrieval: A Survey, 2024.

[35] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of Thought Prompting Elicits Reasoning in Large Language Models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[36] Jun Xu, Tao Mei, Ting Yao, and Yong Rui. MSR-VTT: A Large Video Description Dataset for Bridging Video and Language. June 2016.

[37] Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences, 2023.

[38] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints, 2023.

Lista de Figuras

Lista de Tablas

Anexos

Anexo A

ViT: Vision Transformer

El *Vision Transformer* (*ViT*) es un modelo innovador que adapta la arquitectura *Transformer*, originalmente diseñada para el procesamiento de lenguaje natural, a tareas de visión por computadora. Este enfoque introduce una metodología efectiva y diferente en comparación con los modelos tradicionales, como las redes neuronales convolucionales. A continuación, se describe su arquitectura y funcionamiento, con apoyo de la figura A.1.

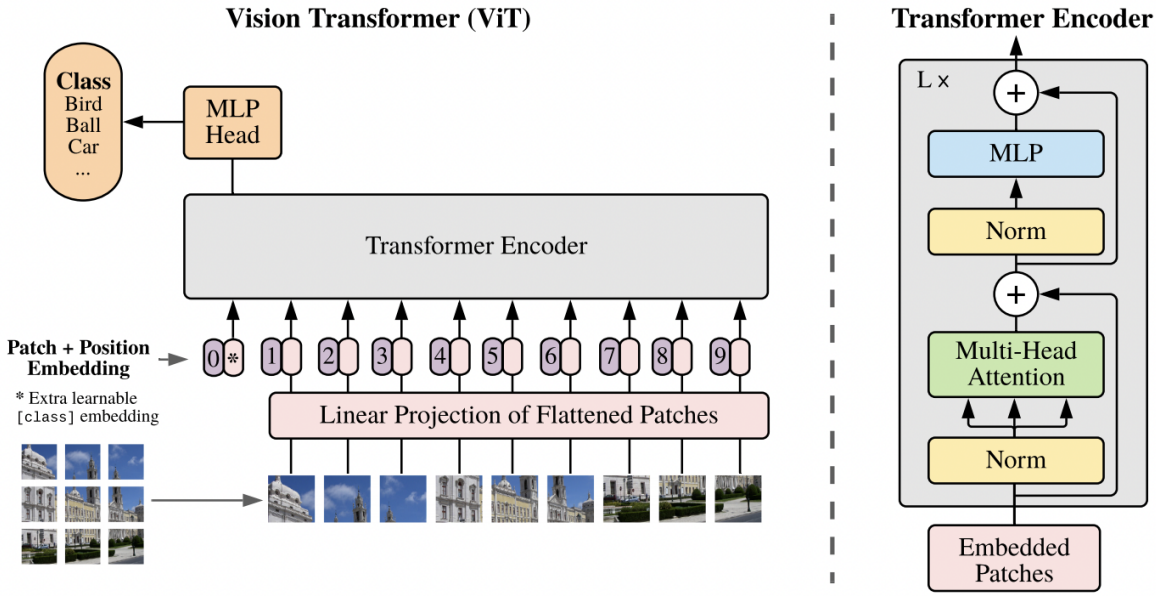


Figura A.1: Arquitectura del *Vision Transformer*.

El proceso comienza con la descomposición de la imagen de entrada en pequeños bloques de tamaño fijo, generalmente de 16×16 píxeles. A diferencia del enfoque convencional que trata las imágenes como matrices bidimensionales de píxeles, el *ViT* reorganiza cada uno de estos bloques en un vector unidimensional mediante un proceso de "aplanamiento" (*flattening*).

Posteriormente, estos vectores se someten a una proyección lineal que los transforma en vectores de dimensión fija, lo que estandariza la representación de cada bloque y facilita su procesamiento posterior. Los vectores resultantes, denominados *patch embeddings*, encapsulan una representación compacta y uniforme de los fragmentos de la imagen. Esta conversión simplifica la manipulación de la imagen en una secuencia de vectores, un formato más adecuado para la arquitectura *Transformer*.

Para conservar la disposición espacial de los datos, se añade información posicional a los *patch embeddings*. Estos *positional embeddings* se suman a los vectores de los bloques, preservando la ubicación original de cada bloque dentro de la imagen. Esto permite al modelo no solo reconocer las características visuales de los bloques, sino también comprender su disposición espacial en la imagen global, aspecto crucial para la interpretación precisa del contenido visual.

Los vectores enriquecidos con información posicional se introducen en un codificador *Transformer*, el núcleo del *ViT*. Este codificador alterna entre mecanismos de *Multi-Head Self-Attention* (*MSA*) y bloques de *Multilayer Perceptron* (*MLP*). El *self-attention* permite al modelo analizar simultáneamente las interacciones entre diferentes regiones de la imagen, lo cual es esencial para captar la estructura global y evaluar las relaciones entre los bloques. Cada capa dentro del codificador se refuerza con normalización y conexiones residuales, mejorando la estabilidad del entrenamiento y la convergencia del modelo.

Finalmente, durante el proceso de codificación, se introduce un *token* especial `[CLS]` (*classification token*) al inicio de la secuencia de vectores de bloques. A medida que los datos avanzan a través de las capas del *Transformer*, este *token* acumula una representación densa de la imagen completa. El resultado es un vector de tamaño fijo, el *image embedding*, que captura las características relevantes de los bloques y sus relaciones espaciales dentro de la imagen.

El uso de *ViT* en la arquitectura de *CLIP* mejora la gestión de las relaciones espaciales y las conexiones complejas en las imágenes, logrando una comprensión más detallada del contenido visual. Esto permite que *CLIP*, junto con el preentrenamiento en grandes conjuntos de datos, vincule imágenes y texto de manera efectiva, sencilla y escalable.

Anexo B

Computación de Sumas Acumulativas

En este anexo, se presenta el procedimiento para computar las sumas acumulativas requeridas en la optimización de la función objetivo J del algoritmo *KTS*. La matriz K es una matriz cuadrada de dimensiones $n \times n$ donde cada elemento $K_{i,j}$ representa la similitud coseno entre los descriptores i y j.

Para mejorar la eficiencia en los cálculos, se introduce el vector $K1$, que contiene las sumas acumulativas de los elementos diagonales de K. Este vector se obtiene utilizando el comando `K1 = np.cumsum([0] + list(np.diag(K)))`. Matemáticamente, $K1$ se define como:

$$K1_i = \sum_{j=0}^{i-1} d_j \quad \text{para } i = 1, 2, \ldots, n+1, \text{ donde } d = [0, K_{1,1}, K_{2,2}, \ldots, K_{n,n}] \tag{B.1}$$

Aquí, $K1$ es un vector que comienza con cero, seguido por las sumas acumulativas de las similitudes coseno de cada descriptor consigo mismo.

Seguidamente, se define la matriz $K2$, que almacena las sumas acumulativas dobles de K. Esto se realiza mediante el comando `K2 = np.zeros((n+1, n+1))`. Matemáticamente, $K2$ se expresa como:

$$K2_{i,j} = \sum_{k=1}^{i} \sum_{l=1}^{j} K_{k,l}, \quad \text{para } i = 1, 2, \ldots, n \text{ y } j = 1, 2, \ldots, n \tag{B.2}$$

En $K2$, los elementos de la primera fila y columna permanecen en cero. Cada elemento $K2_{i,j}$ representa la suma acumulativa de los elementos de K desde la posición $(1, 1)$ hasta (i, j), integrando las similitudes coseno de subconjuntos de descriptores.

Para el cálculo la matriz de dispersión J, que recoge las varianzas no normalizadas, se emplean $K1$ y $K2$. Las varianzas no normalizadas son una medida de dispersión que cuantifica la variabilidad de las similitudes coseno entre descriptores en intervalos dados, sin ajustar los valores por el tamaño del intervalo. Esto implica que no se realiza una corrección por el número de observaciones, lo que se traduce en una forma directa de medir la dispersión sin normalización adicional. Matemáticamente, la varianza no normalizada se define como:

$$\text{Varianza No Normalizada} = \sum_{i=1}^{n}(\mathbf{x}_i - \bar{\mathbf{x}})^2 \tag{B.3}$$

donde $\mathbf{x}_i$ representa cada valor del conjunto de datos y $\bar{\mathbf{x}}$ es la media de los datos.

En este contexto, la varianza no normalizada se calcula restando el promedio de las similitudes cruzadas de la suma de las similitudes diagonales del segmento:

$$v_{t,t+d} = \sum_{i=t}^{t+d-1} K_{i,i} - \frac{1}{d}\sum_{i,j=t}^{t+d-1} K_{i,j} \tag{B.4}$$

El primer término se desglosa como:

$$\sum_{i=t}^{t+d-1} K_{i,i} = \sum_{i=1}^{t+d-1} K_{i,i} - \sum_{i=1}^{t-1} K_{i,i} = K1(t+d-1) - K1(t-1) \tag{B.5}$$

Y el segundo término se detalla de la siguiente manera:

$$\begin{aligned}\sum_{i,j=t}^{t+d-1} K_{i,j} = \sum_{i,j=1}^{t+d-1} K_{i,j} + \sum_{i,j=1}^{t-1} K_{i,j} - \sum_{i=1}^{t+d-1}\sum_{j=1}^{t-1} K_{i,j} - \sum_{i=1}^{t-1}\sum_{j=1}^{t+d-1} K_{i,j} = \\ = K2(t+d-1, t+d-1) + K2(t-1, t-1) - \\ - K2(t+d-1, t-1) - K2(t-1, t+d-1)\end{aligned} \tag{B.6}$$

Por lo tanto, la ecuación B.4 en función de $K1$ y $K2$ se formula como:

$$\begin{aligned}v_{t,t+d} = K1(t+d-1) - K1(t-1) - \frac{1}{d}[K2(t+d-1, t+d-1) + \\ + K2(t-1, t-1) - K2(t+d-1, t-1) - K2(t-1, t+d-1)]\end{aligned} \tag{B.7}$$

Descomponer las sumas acumulativas de esta manera, en lugar de calcular directamente las varianzas no normalizadas, permite optimizar el proceso. Las operaciones acumulativas son más eficientes desde el punto de vista computacional y facilitan el manejo de grandes volúmenes de datos, mejorando así la velocidad y precisión del algoritmo *KTS*.

La figura B.1 ilustra un ejemplo visual de la matriz de similitudes K y sus regiones correspondientes, facilitando la comprensión del cálculo de las varianzas entre intervalos del vídeo para lograr esa segmentación semántica deseada.

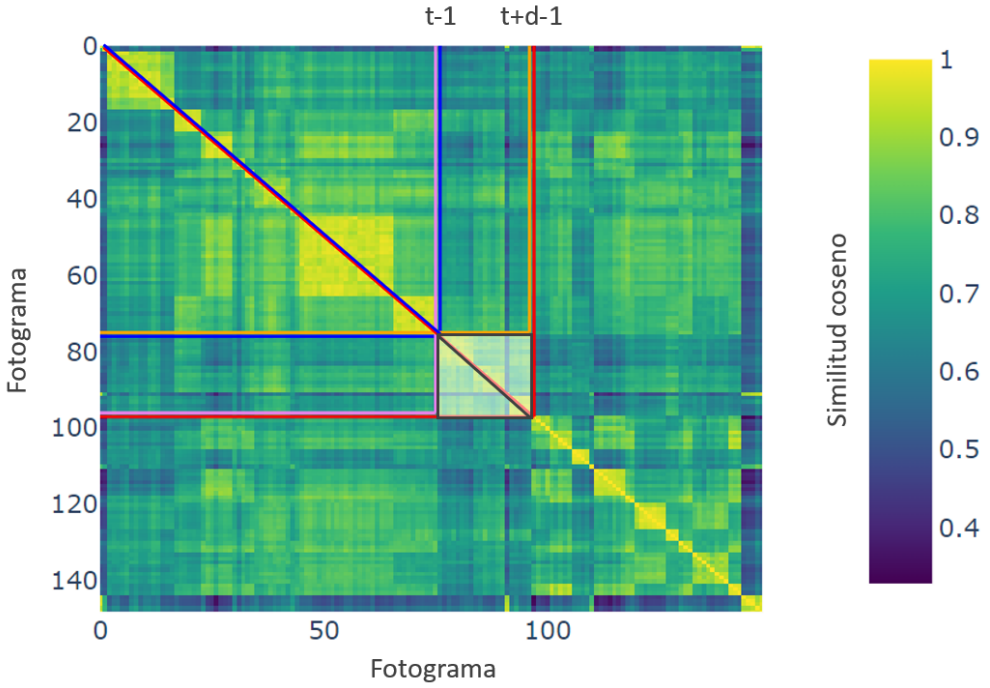


Figura B.1: Proceso de cómputo de sumas acumulativas en un intervalo con inicio en el punto t y una duración de d.

Se resaltan varios recuadros, los cuales indican las distintas regiones de interés necesarias para el cálculo de las sumas acumulativas:

- **Región roja**: Esta área abarca los elementos desde el inicio de la matriz K hasta el punto $t+d-1$. La diagonal roja corresponde al término $K1(t+d-1)$, mientras que el recuadro rojo se refiere al término $K2(t+d-1, t+d-1)$.

- **Región azul**: Esta sección representa los elementos de la matriz K desde el origen hasta el punto $t-1$. La diagonal azul está asociada con el término $K1(t-1)$ y el recuadro azul con el término $K2(t-1, t-1)$.

- **Región naranja**: Este recuadro incluye los elementos comprendidos entre los puntos $t+d-1$ y $t-1$, correspondiendo al término $K2(t+d-1, t-1)$.

- **Región violeta**: Esta área abarca los elementos entre los puntos $t-1$ y $t+d-1$, asociados con el término $K2(t-1, t+d-1)$.

- **Región gris**: Esta sección comprende los elementos desde el punto t hasta el punto $t+d-1$ de la matriz K, intervalo destacado en blanco. La diagonal gris representa el primer término de la ecuación B.4, mientras que el recuadro gris corresponde al segundo término de la misma.

Anexo C

Caracterización de *LLMs*

C.1. *GPT*-4

El modelo *GPT*-4, desarrollado por *OpenAI*, representa un avance significativo en el ámbito de los modelos de lenguaje de gran escala. En este trabajo se emplea específicamente la versión `gpt-4`, en su iteración actual `gpt-4-0613`, la cual ha demostrado un rendimiento superior en comparación con versiones previas, destacándose especialmente por su razonamiento, precisión y generación de contenido seguro. *GPT*-4 sobresale en la inferencia y generación de texto, lo que lo hace fundamental para aplicaciones en diversas áreas, desde la interacción con usuarios hasta el procesamiento de información compleja.

Basado en la arquitectura *Transformer*, *GPT*-4 fue preentrenado con el objetivo de predecir el siguiente *token* en un documento, utilizando una vasta colección de datos provenientes tanto de fuentes públicas como licenciadas. Posteriormente, su comportamiento fue afinado con "aprendizaje por refuerzo con retroalimentación humana"[1], lo que permitió que *GPT*-4 se alineara con mayor precisión a los valores y preferencias humanas. Esta alineación ha sido reforzada mediante retroalimentación continua por usuarios de *ChatGPT* y más de 50 expertos, resultando en mejoras significativas en términos de veracidad, control y cumplimiento de límites predefinidos.

El desarrollo de *GPT*-4 se llevó a cabo sobre una supercomputadora co-diseñada en colaboración con *Microsoft Azure*. Esta plataforma permitió, tras pruebas preliminares con *GPT*-3.5, corregir errores y fortalecer las bases teóricas del modelo. Como resultado, *GPT*-4 logró una estabilidad sin precedentes, posicionándose como un *LLM* con un rendimiento predecible, lo que refuerza su escalabilidad y confiabilidad futuras, además de su creatividad y capacidad para manejar instrucciones más complejas.

[1] *RLHF* (*Reinforcement Learning from Human Feedback*) es una técnica que permite ajustar las respuestas del modelo de acuerdo con las expectativas humanas, basándose en evaluaciones de preferencia en lugar de depender exclusivamente de señales automáticas o métricas predefinidas. [37]

GPT-4 es capaz de procesar un contexto de hasta 8,192 *tokens*[2], lo que habilita su uso en análisis extendidos y conversaciones prolongadas. La integración de *GPT*-4 en aplicaciones se realiza de manera segura y eficiente mediante *APIs* especializadas, optimizando el flujo de trabajo al configurar las claves *API* como variables de entorno. El acceso al modelo requiere registrarse en la *API* de *OpenAI* y obtener una clave correspondiente, con un costo de 30$ por cada millón de *tokens* de entrada y 60$ por cada millón de *tokens* de salida.

C.2. *Llama* 3

Llama 3, desarrollado por *Meta*, es otro gran modelo de lenguaje natural preentrenado y ajustado para la generación de texto, destacándose por su notable capacidad de comprensión. En este trabajo, se ha implementado la versión *Llama* 3 de 70,000 millones de parámetros (`llama3-70b-8192`), optimizado específicamente para aplicaciones de diálogo, donde ha demostrado un rendimiento superior a numerosos modelos de código abierto en términos de utilidad y seguridad.

Este modelo utiliza una arquitectura *Transformer* auto-regresiva optimizada, con una ventana de contexto de 8,192 *tokens* y un *tokenizador*[3] que maneja un vocabulario de 128,000 *tokens*. Estas características le permiten procesar grandes volúmenes de texto de manera precisa. Además, para mejorar la eficiencia en la inferencia, emplea la técnica *GQA*[4] (*Grouped-Query Attention*). *Llama* 3 fue preentrenado con más de 15 billones de *tokens* de datos públicos, recopilados hasta diciembre de 2023, lo que garantiza un rendimiento actualizado y robusto. Los datos utilizados para el "ajuste fino" incluyen conjuntos de instrucciones de acceso público y más de 10 millones de ejemplos anotados manualmente.

El desarrollo de *Llama* 3 involucró el uso de bibliotecas de entrenamiento personalizadas y la infraestructura del *SuperCluster* de Investigación de *Meta*, acumulando 6.4 millones de horas de *GPU*. Las *GPUs* empleadas fueron del tipo H100-80GB, con un consumo de energía de 700 W cada una. A pesar de que la generación de este modelo produjo 1,900 tCO2eq, *Meta* ha compensado completamente estas emisiones a través de su programa de sostenibilidad, asegurando la neutralización del impacto ambiental.

[2]Los *tokens* son fragmentos de palabras (1,000 *tokens* equivalen aproximadamente a 750 palabras).
[3]El *tokenizador* es la herramienta que se utiliza para dividir un texto en unidades discretas (*tokens*).
[4]*Grouped-Query Attention* (*GQA*) es una técnica que optimiza la eficiencia en modelos de lenguaje al agrupar y procesar consultas similares de manera conjunta, mejorando la velocidad y reduciendo la carga computacional sin comprometer la precisión. [38]

Anexo D

Flujo de Trabajo del Agente Autónomo

Como se detalla en la sección 4.2, el mecanismo de *function calling* se emplea para invocar las herramientas a través de solicitudes estructuradas en formato *JSON*. En este anexo, la tabla D.2 muestra los esquemas generados durante el flujo de trabajo del agente autónomo frente a una consulta compleja (véase la tabla D.1, que ilustra en el guion audiovisual de ejemplo dicha consulta), la cual necesita ser descompuesta en subtareas más manejables.

A partir del guion, el *LLM* del agente debe extraer las tareas necesarias para crear la pieza audiovisual. A continuación, se muestra un ejemplo de la búsqueda de recursos requeridos para la creación de la pieza titulada "La Historia del Baile":

Tarea 1: Buscar en la colección RTVEArchivo dos recursos audiovisuales relacionados con "bailes tradicionales".

Tarea 2: Buscar en la *web* una imagen representativa de "baile moderno".

Tarea 3: Buscar en la colección *MSR-VTT* un recurso que corresponda a la imagen encontrada en la tarea anterior.

GUION AUDIOVISUAL

INTRODUCCIÓN

Música	[Música animada y moderna]
Pantalla	[Pantalla con el título: "La Historia del Baile"]
Narrador	"¡Bienvenidos a un fascinante recorrido por la evolución del baile a lo largo de los siglos!"

SECCIÓN 1: BAILES TRADICIONALES

Pantalla	[Mostrar dos recursos de RTVEArchivo sobre bailes tradicionales]
Narrador	"Iniciamos con los bailes tradicionales, un vistazo a cómo nuestras culturas expresaban su identidad y pasión a través del movimiento."

SECCIÓN 2: BAILE MODERNO

Pantalla	[Mostrar una imagen de baile moderno encontrada en la *web*]
Narrador	"Ahora, exploramos el baile moderno, donde la innovación y la energía vibrante se combinan para crear nuevas formas de expresión."
Pantalla	[Mostrar un *clip* de *MSR-VTT* relacionado con la imagen de baile moderno]
Narrador	"Este fragmento captura la esencia dinámica y diversa que define al baile en la actualidad."

CONCLUSIÓN

Pantalla	[Pantalla final con el título y créditos]
Narrador	"Desde las raíces profundas de nuestras tradiciones hasta la innovación del presente, el baile sigue siendo una expresión en constante transformación. ¡Gracias por unirse a este viaje a través del tiempo y el movimiento!"

Tabla D.1: Ejemplo de guion audiovisual utilizado como entrada de consulta para el asistente.

Acción

Invocando la herramienta: "search_query" con la siguiente entrada:


```
{
    "collection": "RTVEArchivo",
    "query": "bailes tradicionales",
    "k": 2
}
```


Observación

Resultado de la herramienta:


```
[
    {
        "UMAP": [11.583260536193848, -8.780284881591797],
        "reference": "I00876098",
        "scene": 5,
        "segment": 0,
        "start": "00:00:20,040",
        "start_scene": "00:00:20,040",
        "stop": "00:00:28,040",
        "stop_scene": "00:00:28,040",
        "title": "6 NEWS IN 1 MINUTE: HEADER AND NEWS",
        "topic": "Educational"
    },
    {
        "UMAP": [1.622897028923035, 14.693753242492676],
        "reference": "I00786018",
        "scene": 2,
        "segment": 4,
        "start": "00:03:06,040",
        "start_scene": "00:02:41,040",
        "stop": "00:03:23,040",
        "stop_scene": "00:03:23,040",
        "title": "LUIS CARRERO BLANCO, DEPUTY SECRETARY OF THE
        ↪  PRESIDENCY OF THE GOVERNMENT, IN EQUATORIAL GUINEA",
        "topic": "Culture"
    }
]
```


Acción

Invocando la herramienta: "search_serper" con la siguiente entrada:


```
{
    "query": "baile moderno",
    "k": 1
}
```

Observación

Resultado de la herramienta: `["image1.jpg"]`

Acción

Invocando la herramienta: `"search_query"` con la siguiente entrada:


```
{
    "collection": "MSR-VTT",
    "query": "image1.jpg",
    "k": 1
}
```


Observación

Resultado de la herramienta:


```
[
    {
        "UMAP": [-18.08305549621582, 14.4802827835083],
        "category": 3,
        "scene": 0,
        "segment": 0,
        "start": "00:00:00,040",
        "start_scene": "00:00:00,040",
        "stop": "00:00:10,040",
        "stop_scene": "00:00:10,040",
        "url": "https://www.youtube.com/watch?v=9XtknhTGbtU",
        "video_id": "video7183"
    }
]
```


Respuesta final

Finalización del agente: `success`

Tabla D.2: Flujo de trabajo del agente para resolver el ejemplo de guion audiovisual planteado utilizando la técnica *CoT*.

Anexo E

Interfaz de Usuario del Asistente

La aplicación del asistente desarrollada en este trabajo se presenta mediante una interfaz implementada con *Streamlit*[1], una plataforma de código abierto que simplifica la creación de aplicaciones *web* interactivas utilizando el lenguaje de programación *Python*. Esta herramienta permite visualizar y compartir resultados de forma eficiente, sin necesidad de conocimientos avanzados en desarrollo *web*. En este anexo, se muestra la interfaz mencionada, acompañada de una muestra de los resultados generados en respuesta a la consulta planteada en el anexo anterior.

La aplicación comienza explicando cómo funciona el asistente (véase figura E.2), guiando al usuario en la realización de consultas y la exploración de las distintas funcionalidades disponibles. Además, se incluye un ejemplo práctico para facilitar su comprensión. Un selector permite elegir entre dos tipos de búsqueda: una textual, basada en una temática específica, o una visual, cargando una imagen en el sistema. Luego, se presenta un área de texto para introducir las especificaciones de la búsqueda y, en el caso de una consulta visual, se habilita un cuadro adicional. Finalmente, un botón muestra los resultados junto con las herramientas utilizadas.

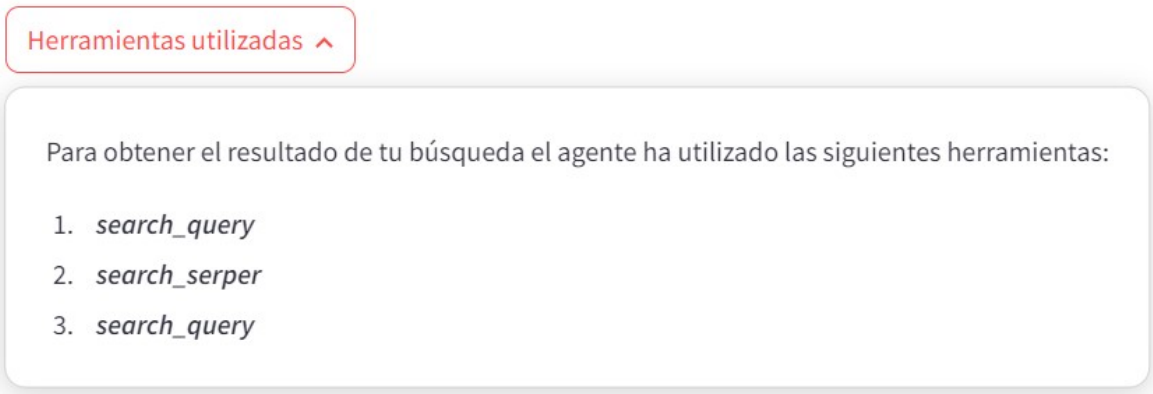


Figura E.1: Herramientas empleadas en el proceso de resolución de la consulta.

[1] *Streamlit*. Documentación disponible en: https://streamlit.io/

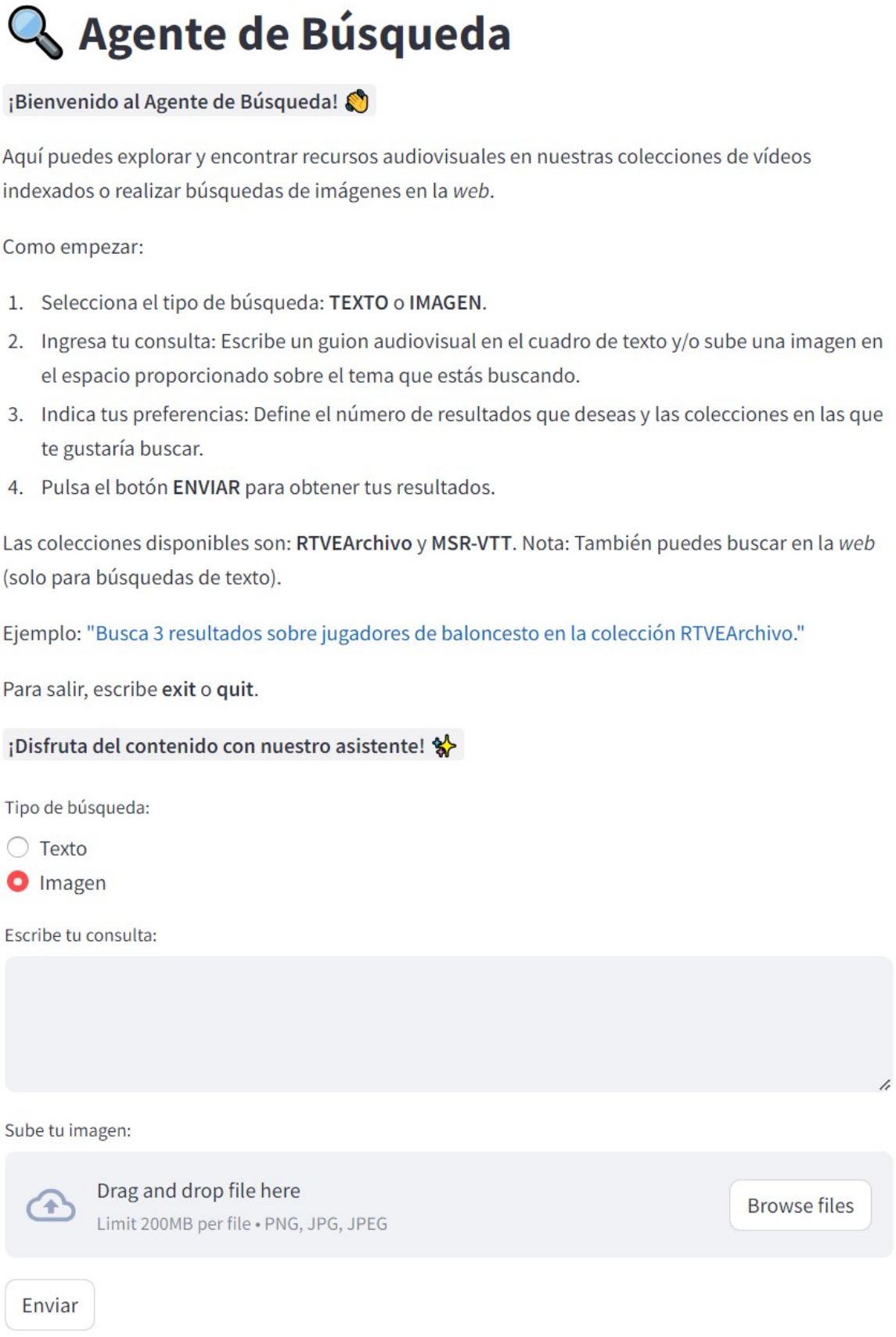


Figura E.2: Visualización de la interfaz gráfica de la aplicación.

Debido al comportamiento de la plataforma, cada vez que se presiona un botón, la aplicación se reinicia por completo, perdiendo el estado previo de los elementos. En este caso, la ejecución del agente y la visualización de los resultados están anidadas en el elemento `st.button("Enviar")`, por lo que al pulsarlo, el *script*[2] se reinicia, modificando el estado del botón y eliminando los estados anteriores. Para conservar la información entre ejecuciones, se han implementado variables de estado mediante `st.session_state`, que permiten mantener los valores entre sesiones. Se han configurado funciones como `st.cache_data` y `st.cache_resource` para almacenar en caché datos y recursos globales, necesarios para el correcto rendimiento del asistente.

Además, se incluyen mensajes informativos para diferentes situaciones: advertencias (`st.warning`) que recuerdan al usuario las instrucciones, notificaciones de error (`st.error`) en caso de que ocurra un fallo o excepción durante el proceso, y confirmaciones de éxito (`st.success`) cuando la tarea se ha completado correctamente.

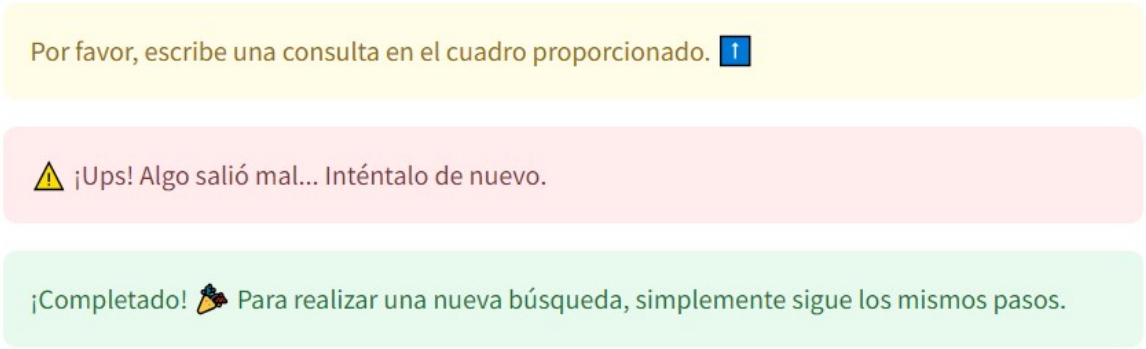


Figura E.3: Mensajes informativos según su tipo: advertencia, error o éxito.

Las figuras a continuación ilustran los resultados obtenidos tras la consulta del usuario, los cuales son de utilidad para la elaboración de dicho guion audiovisual.

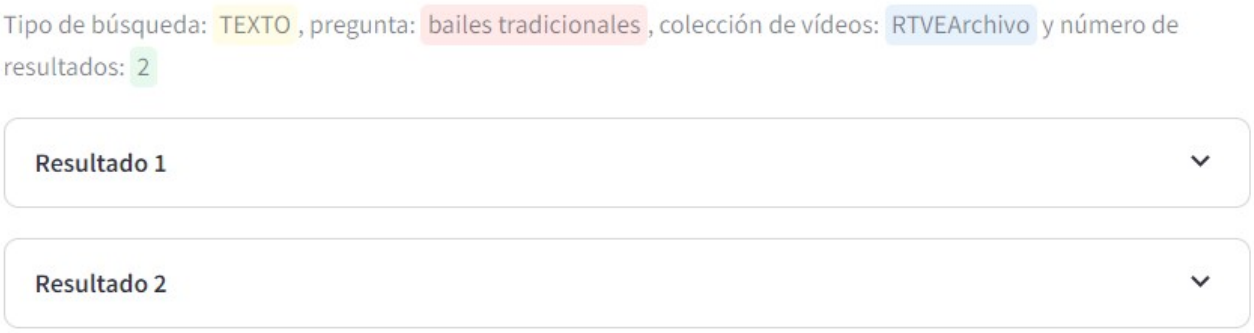


Figura E.4: **Tarea 1**: Buscar en la colección RTVEArchivo dos recursos audiovisuales relacionados con "bailes tradicionales".

[2]En programación, un *script* es un conjunto de código diseñado para ejecutar una tarea específica.

Resultado 1

Información del resultado obtenido durante el proceso de búsqueda en secuencias:

Similitud coseno: 0.2649

Temática: Educational

Título: 6 NEWS IN 1 MINUTE: HEADER AND NEWS

Vídeo: I00876098.mp4

Escena: 5

Tiempo de inicio de la escena: 00:00:20,040

Tiempo de fin de la escena: 00:00:28,040

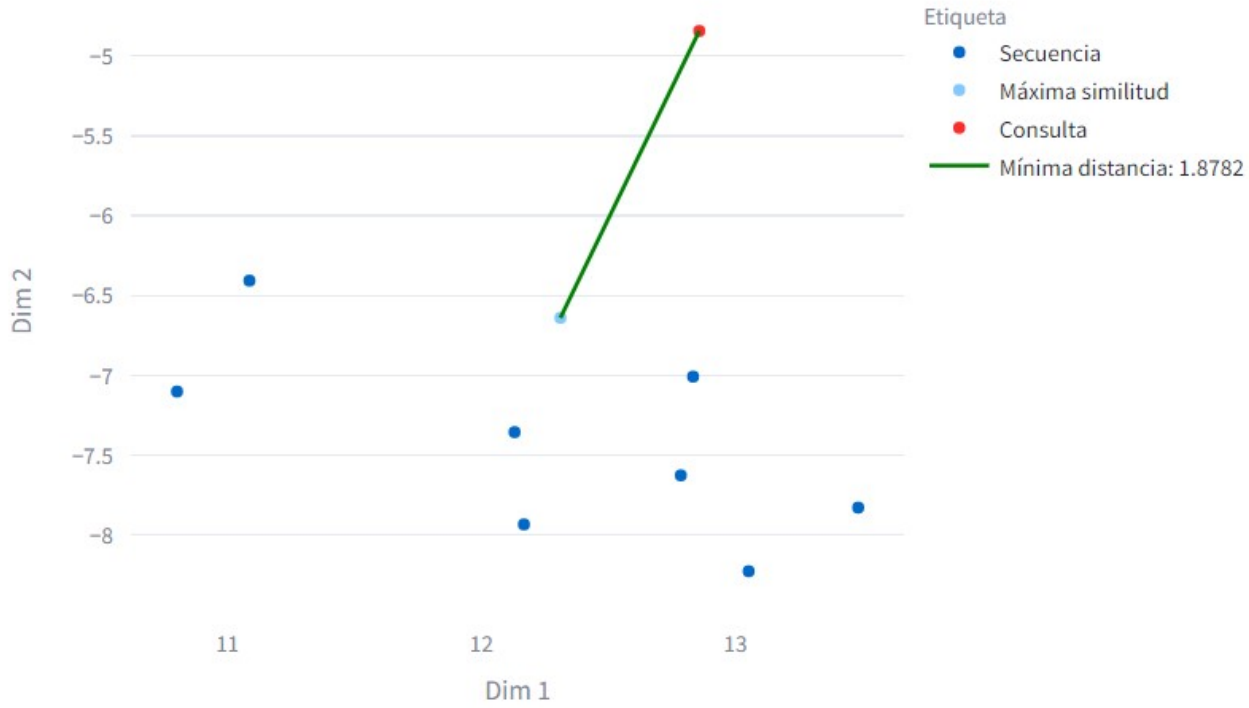
Secuencia de la escena: 0

Vídeo de la escena encontrada en la base de datos.

Figura E.5: **Tarea 1**: Resultado 1 (parte 1).

Información del resultado derivada del proceso de búsqueda fina dentro de la secuencia:

Representación UMAP de los fotogramas de la secuencia y la consulta realizada.



Fotograma con mínima distancia euclídea: 1.8782

Figura E.6: **Tarea 1**: Resultado 1 (parte 2).

Resultado 2

Información del resultado obtenido durante el proceso de búsqueda en secuencias:

Similitud coseno: 0.2599

Temática: Culture

Título: LUIS CARRERO BLANCO, DEPUTY SECRETARY OF THE PRESIDENCY OF THE GOVERNMENT, IN EQUATORIAL GUINEA

Vídeo: I00786018.mp4

Escena: 2

Tiempo de inicio de la escena: 00:02:41,040

Tiempo de fin de la escena: 00:03:23,040

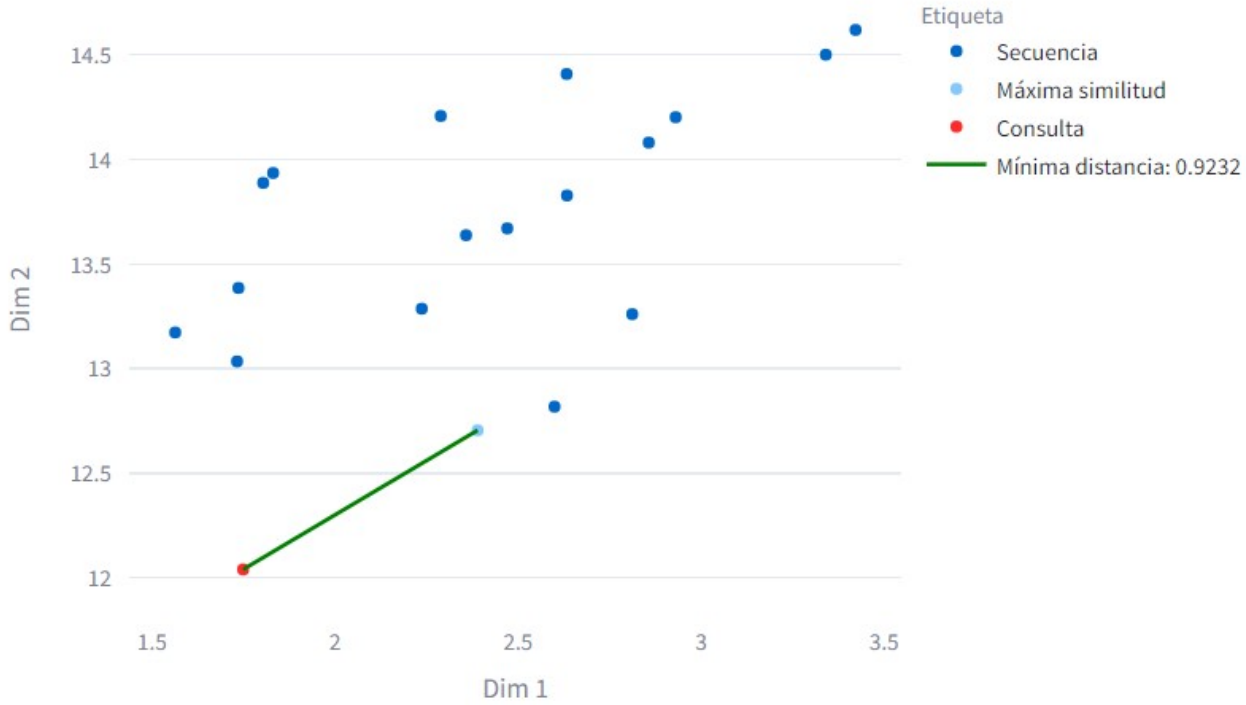
Secuencia de la escena: 4

Vídeo de la escena encontrada en la base de datos.

Figura E.7: **Tarea 1**: Resultado 2 (parte 1).

Información del resultado derivada del proceso de búsqueda fina dentro de la secuencia:

Representación UMAP de los fotogramas de la secuencia y la consulta realizada.



Fotograma con mínima distancia euclídea: 0.9232

Figura E.8: **Tarea 1**: Resultado 2 (parte 2).

Figura E.9: **Tarea 1**: Reorganización de los resultados 1 y 2.

Figura E.10: **Tarea 2**: Buscar en la *web* una imagen representativa de "baile moderno".

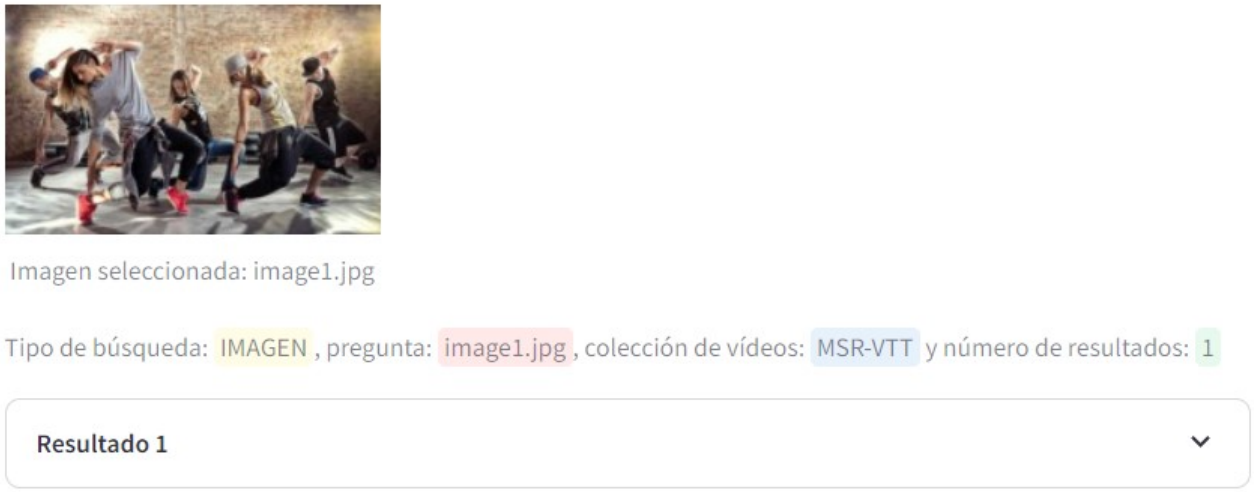


Figura E.11: **Tarea 3**: Buscar en la colección *MSR-VTT* un recurso que corresponda a la imagen encontrada en la tarea anterior.

Resultado 1

Información del resultado obtenido durante el proceso de búsqueda en secuencias:

Similitud coseno: 0.7366

Categoría: 3

Título: video7183

Vídeo: video7183.mp4

Escena: 0

Tiempo de inicio de la escena: 00:00:00,040

Tiempo de fin de la escena: 00:00:10,040

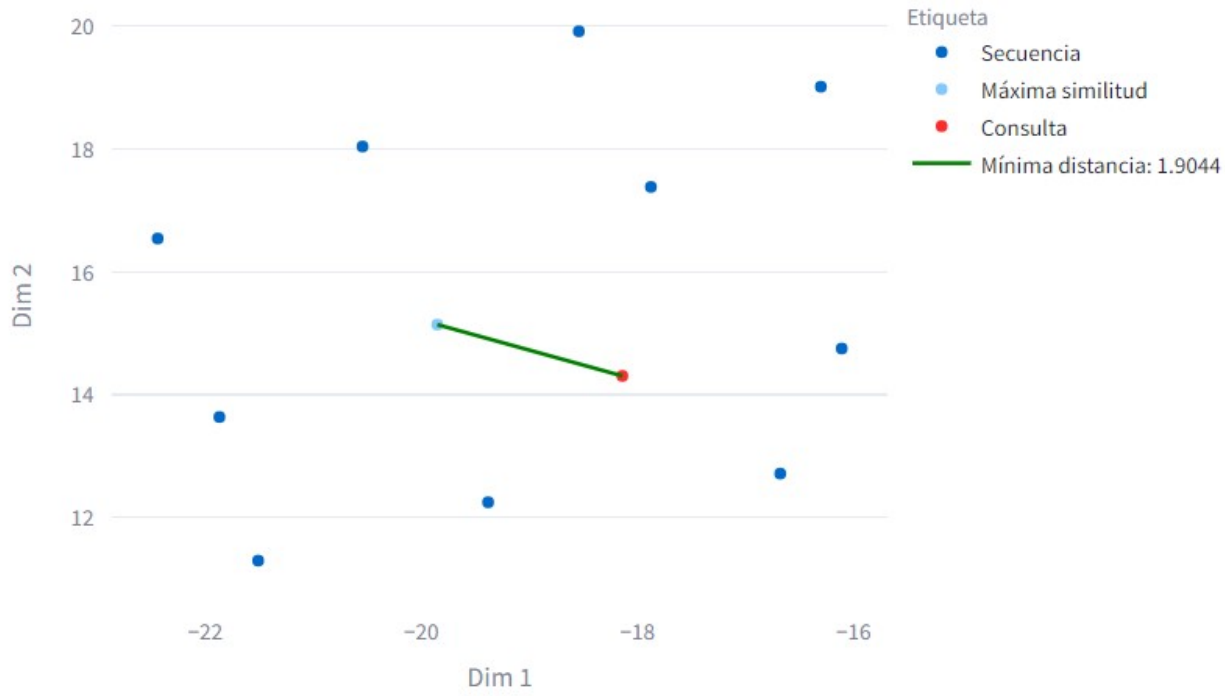
Secuencia de la escena: 0

Vídeo de la escena encontrada en la base de datos.

Figura E.12: **Tarea 3**: Resultado 1 (parte 1).

Información del resultado derivada del proceso de búsqueda fina dentro de la secuencia:

Representación UMAP de los fotogramas de la secuencia y la consulta realizada.



Fotograma con mínima distancia euclídea: 1.9044

Figura E.13: **Tarea 3**: Resultado 1 (parte 2).