Recuperación de Información y Recomendaciones en la Web

Edición 2024 - Grupo 1 - Entrega Final

Integrantes

Diego Amorena 5.011.502-7
Alexei Guchin 5.078.761-4
Federico Luna 4.997.613-9
Ignacio Remersaro 4.950.811-6

Índice

Problema

El problema en el que se basa este proyecto es la extensa cantidad de tiempo que se requiere para mirar una clase grabada o video educativo. Estas clases pueden durar horas, pueden tener una mala calidad de audio o video, consumir mucho ancho de banda o tener cortes que hagan dificultosa la comprensión de un estudiante. Esto puede provocar múltiples inconvenientes y frustración en el mismo. Todo esto disminuye la productividad y deteriora la motivación del estudiante. Otro gran problema, es la dificultad a la hora de buscar información en los videos, ya que es muy difícil recordar el momento exacto y la clase donde se habló de ese tema particular, del curso que está buscando.

Se acotó inicialmente el problema a la Facultad de Ingeniería de la UdeLaR, más en detalle, a la plataforma educativa OpenFING[1] desarrollada en esta facultad que contiene videos de muchos cursos, separados en Teóricos y Prácticos por el año de dictado.

Descripción

El proyecto consiste en la implementación de un sistema de transcripción de videos de la plataforma OpenFING para obtener subtítulos y generar resúmenes basándose en las preferencias del usuario. El proceso comienza con la selección de videos educativos, que pasan por un proceso de transcripción automática, utilizando la herramienta Whisper de OpenAI[2] que convierte la voz del video en texto.

Una vez obtenidos los subtítulos, estos son introducidos en un gran modelo de lenguaje (LLM), como Gemini[3]. Utilizando un prompt predefinido que a futuro podrá ser ajustado, mediante ciertos parámetros ingresados por el usuario (dentro de una lista de parámetros permitidos), se genera un resumen a partir de la transcripción del video.

Los resúmenes producidos, se almacenan posteriormente en una base de datos. Este almacenamiento no solo permite la conservación de la información sino que también facilita el acceso y la recuperación de contenidos por parte de los usuarios. Además, desde la web, se podrá realizar una búsqueda de resúmenes o videos a partir de consultas utilizando palabras clave. El proceso de resumen de las transcripciones, se realiza bajo demanda, asegurando que los recursos computacionales se utilizan de manera eficiente y que los resúmenes se generen y actualicen constantemente, cuando se añaden videos nuevos a la plataforma.

[1] https://open.fing.edu.uy/
[2] https://openai.com/index/whisper/
[3] https://gemini.google.com/app

Herramientas

El proyecto está enfocado principalmente en la plataforma OpenFING que proporciona videos estudiantiles de materias de la Facultad de Ingeniería de la Universidad de la República[4]. Mediante la técnica de Web Scraping, se obtendrá toda la información relacionada a los videos de los cursos (nombre del curso, nombre de la clase, link, etc.). Esta información será mostrada en nuestro sitio web, donde se podrá ver la clase, pero además, se podrán generar resúmenes de las clases y/o buscar resúmenes previamente realizados.

El backend fue realizado en el lenguaje Python, junto con la biblioteca FastAPI[5] que provee una manera simple de publicar una API y así lograr la interconexión entre el backend y el frontend.

Se utilizaron las siguientes herramientas de inteligencia artificial, para obtener las transcripciones y los resúmenes:

- **Google Gemini:** Google Gemini es un gran modelo de lenguaje (LLM) desarrollado por Google en el año 2023. Está diseñado para procesar y generar texto y, en algunos casos, imágenes, utilizando técnicas avanzadas de inteligencia artificial.
- **OpenAI Whisper:** Es una herramienta de transcripción de audio a texto desarrollada por OpenAI en el año 2022. Está diseñada para convertir el habla en texto escrito de manera precisa y eficiente. Whisper utiliza modelos avanzados de aprendizaje profundo para reconocer y transcribir el audio en varios idiomas, incluso en entornos con ruido de fondo o con múltiples hablantes. La herramienta toma como entrada un archivo de video o de audio y luego retorna un archivo JSON con las transcripciones junto con ciertos metadatos.

Para persistir los datos de los videos, transcripciones y resúmenes se utilizó la base de datos MongoDB[6] y para el filtrado y búsqueda de texto en resúmenes se utilizó ElasticSearch[7]. Se utilizó Docker[8] para desplegar la base de datos de Mongo y el motor ElasticSearch en un entorno local.

- **MongoDB:** Es una base de datos no relacional que almacena datos en documentos con formato BSON, muy similar a JSON, lo que permite una mayor flexibilidad en la estructura de los datos. A diferencia de las bases de datos relacionales, no utiliza tablas ni filas, sino colecciones y documentos. Es ideal para aplicaciones que requieren manejo de datos no estructurados o semiestructurados y es escalable, lo que facilita el crecimiento y manejo de grandes volúmenes de datos.
- **ElasticSearch:** Es un motor de búsqueda y análisis de datos en tiempo real. Permite realizar búsquedas muy eficientes y precisas, sobre una enorme cantidad de textos.

[4] https://www.fing.edu.uy/
[5] https://fastapi.tiangolo.com/
[6] https://www.mongodb.com/
[7] https://www.elastic.co/es/
[8] https://www.docker.com/

Utiliza una estructura de índices para organizar la información, facilitando búsquedas eficientes y rápidas.

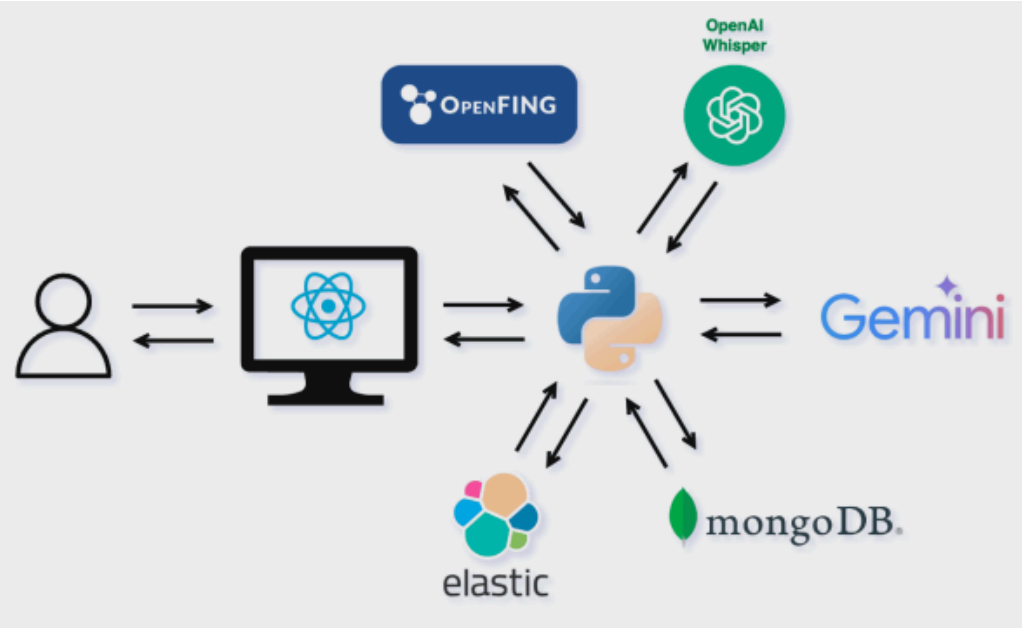
- **Docker:** Es una plataforma que permite crear, desplegar y ejecutar aplicaciones en contenedores. Los contenedores son entornos ligeros y portátiles que contienen todo lo necesario para que una aplicación funcione, incluyendo el código, las bibliotecas y las dependencias. Esto asegura que la aplicación se ejecute de manera consistente en diferentes entornos.
- **React:** Es una biblioteca de JavaScript desarrollada por Facebook para construir interfaces de usuario. Se basa en componentes reutilizables, que son bloques de construcción independientes. React facilita la creación de aplicaciones web interactivas y dinámicas al renderizar solo los componentes necesarios cuando cambian los datos.

El frontend fue realizado en una página web dinámica, implementada con el framework React[9] que consumirá los endpoints definidos en la API de Python.

Se utilizó GitHub y Discord para la comunicación y coordinación de trabajo dentro del equipo.

Solución planteada

Luego de haber descrito todas las herramientas utilizadas en el proyecto. A continuación, se describe el proceso utilizado para abordar el problema. En la siguiente imagen, se puede apreciar la arquitectura del sistema.



[9] https://react.dev/

El módulo de scrapping fue implementado utilizando la librería de Python BeautifulSoup[10]. Este módulo descarga el contenido de la página principal de OpenFING e ingresa a cada curso y va guardando los links y los nombres de cada clase. Esta información se agrupa por curso, es decir, tenemos un listado de cursos que dentro tiene un listado de clases. Luego estos datos son almacenados en la base de datos Mongo para la persistencia. Estos datos son utilizados para poder mostrar las clases y cursos disponibles con sus links en el front-end.

Módulo de descarga de videos, este es el módulo que se encarga de bajar los videos de OpenFING. Primero verifica que el video de la clase no esté en el sistema, en caso de ya estar almacenado localmente, simplemente lo devuelve. En caso contrario, descarga el HTML de la clase utilizando el link de la misma, scrapea el HTML para obtener el link del archivo MP4 del video, lo descarga y almacena para futuras llamadas.

Módulo de transcripción, primero se verifica que la transcripción de la clase no esté guardada en Mongo, en este caso simplemente se recupera de la base y se retorna la transcripción. En caso contrario, recibe el video como entrada y utilizando el modelo de OpenAI Whisper, realiza una transcripción en español del video. Luego, ésta es guardada en Mongo, utilizando el hash del URL de la clase como identificador.

Módulo generador de resúmenes, al igual que el módulo de transcripción, primero verifica si existe el resumen en la base de datos, en caso positivo la devuelve. Si no está en la base de datos, utilizando el siguiente prompt: "*Debes resumir el siguiente texto correspondiente a una transcripción de una clase, no te inventes nada, debes ser lo más preciso posible, utiliza bulletpoints. Texto:",* y la transcripción se envía a la API de Gemini para que genere un conciso y preciso resumen de la clase. Una vez generado el resumen, el mismo es almacenado en formato markdown en la base de datos y lo retorna al módulo que lo invocó.

Módulo repositorio, está separado en dos, uno para las transcripciones y otro para los resúmenes. Estos módulos se encargan de conectarse a la base de datos Mongo y realizar las operaciones necesarias. En nuestro caso, se implementaron métodos para obtener y guardar las transcripciones y resúmenes de los documentos BSON.

Como **motor de búsqueda**, se utiliza Elasticsearch, este se aplica cuando un usuario realiza una consulta a través de la página web. Inicialmente, se crea un índice para textos en español y se define un analizador en español, esto es muy útil para la normalización de palabras en español. Se define la estructura de los documentos con los siguientes campos: transcripción, resumen, url_clase y tipo_modelo. Los 2 primeros campos son de tipo texto y son los utilizados para indexar y realizar la búsqueda de texto. El url_clase se utiliza para recuperar el documento directamente y el tipo_modelo como metadato que indica el tipo de modelo de Whisper que se está utilizando, el modelo base que es gratis o el premium que es pago. Además de la recuperación y el guardado de documentos, este módulo tiene una función que se encarga de reindexar, es decir, borra el índice y carga los documentos de Mongo y los vuelve a indexar para estar actualizado. Para la búsqueda se genera una query multi_match que intenta

[10] https://www.crummy.com/software/BeautifulSoup/bs4/doc/

matchear con los campos de transcripción y resumen utilizando el analizador en español (spanish analyzer).

```python
    def search_transcription(self, query: str, size=10):
        # Elasticsearch search with Spanish analyzer
        response = self.es.search(
            index=self.index_name,
            body={
                "query": {
                    "multi_match": {
                        "query": query,
                        "fields": ["transcription", "summary"],
                        "analyzer": "spanish",
                    }
                }
            },
            size=size,
        )
        return response["hits"]["hits"]
```

```python
def create_index(self):
    self.es.indices.delete(index=self.index_name, ignore_unavailable=True)
    self.es.indices.create(
        index=self.index_name,
        body={
            "settings": {
                "analysis": {
                    "analyzer": {
                        "spanish_analyzer": {"type": "spanish"},
                    }
                }
            },
            "mappings": {
                "properties": {
                    "transcription": {
                        "type": "text",
                        "analyzer": "spanish_analyzer",
                        "search_analyzer": "spanish_analyzer",
                    },
                    "summary": {
                        "type": "text",
                        "analyzer": "spanish_analyzer",
                        "search_analyzer": "spanish_analyzer",
                    },
                    "class_url": {"type": "keyword"},
                    "model_type": {"type": "keyword"},
                }
            },
        },
    )

```

Controlador FastApi en este módulo se definen las API Rest del proyecto. Algunos de los endpoints son:

- ***/transcribe*** que recibe el URL de la clase y devuelve la transcripción utilizando el módulo de transcripción.
- ***/summarize_transcription*** recibe una transcripción y devuelve un resumen de la clase utilizando el módulo generador de resúmenes.
- ***/summarize*** realiza el trabajo de los dos endpoints anteriores, recibe el URL de una clase y devuelve su resumen.
- ***/reindex*** manda a reindexar el módulo de Elasticsearch.
- ***/search*** recibe una consulta con los términos que se desean buscar en las clases, y utilizando el módulo de Elasticsearch devuelve el top N de resultados con el mayor score.

Problemas encontrados

En esta sección se detallarán algunos de los problemas e impedimentos que se encontraron durante el desarrollo del proyecto. A su vez, se explican las soluciones desarrolladas o las posibles soluciones que se pueden desarrollar en un futuro para contrarrestar estos impedimentos.

Uno de los impedimentos que se encontró en el proyecto, es el limitado ancho de banda en la descarga de los videos que ofrece OpenFING. OpenFING limita la velocidad de descarga a aproximadamente 1 MB/s, eso es sin considerar el posible tráfico de múltiples estudiantes utilizando la aplicación concurrentemente, estresando los servidores, provocando un menor ancho de banda en la descarga de los videos. Durante el transcurso del proyecto, se observó que la descargas de videos podrían llegar a tardar casi 20 minutos, por lo que resulta impráctico utilizar la aplicación si se tiene que esperar estos tiempos para conseguir el resumen de cada clase.

Está cuestión podría solucionarse si se logra trabajar con el equipo de OpenFING, permitiendo que la IP de nuestro servidor deployado por nuestra aplicación no caiga dentro de las limitaciones de sobrecarga de tráfico que poseen actualmente. Algo interesante también sería que se permita descargar únicamente el audio de la clase. El problema con estas soluciones, es que ninguna puede provenir exclusivamente de lado de nuestro grupo y requiere colaboración externa.

Otra posible solución es el prefetching de videos, con esto nos referimos a comenzar a descargar los videos previó a que el estudiante lo necesite. Esto se haría por ejemplo descargando no solo la clase que el estudiante solicita, sino que también comenzar la descarga de la próxima clase del curso.

Otro problema encontrado es la limitación en el peso máximo de los archivos que recibe Whisper como entrada. Whisper permite pasar videos de hasta 25 megabytes[11], mientras que algunas clases pueden pesar más de un gigabyte.

Para esto se pueden aplicar dos soluciones simples e intuitivas. La primera es extraer el video de cada una de nuestras clases, quedándonos únicamente con el audio de la clase, ya que esto último es lo único que nos concierne. Esto reduce la mayor parte del peso de cualquier clase.

La otra solución es dividir la clase en secciones de 25 megabytes, pero puede resultar problemático si se interrumpen frases debido al límite de peso, resultando en transcripciones menos precisas y de peor calidad tanto al comienzo de la transcripción como al final. Esto podría solucionarse parcialmente realizando un análisis del audio y partiendo en fragmentos en los que se encuentra un silencio en el audio, en puntos, comas, etc.

[11] https://platform.openai.com/docs/guides/speech-to-text/longer-inputs

Un problema más por el cual se tuvo que lidiar, es la cantidad máxima de tokens que toma Gemini. Encontramos que no siempre es posible añadir toda la transcripción realizada por Whisper a Gemini para que produzca un resumen. La versión gratis de Gemini permite enviarle 32.000 tokens en la prompt[12]. En nuestro caso, el prompt genérico ante toda consulta contiene 25 tokens, y como aproximadamente 75 palabras son 100 tokens, se tiene que la transcripción usando este modelo no puede ser de más de 24.000 palabras, esto queda muy justo para clases largas y podría truncar el final de la misma. Sin embargo usando el modelo pago de Gemini 1.5 Pro se tiene 1.048.576 tokens de entrada, es decir 786.432 palabras son más que suficientes para las clases.

Para esto, se puede resolver dividiendo la transcripción en partes, de manera de evitar esta limitante. Sin embargo las clases, en su mayoría, deben de considerarse como una unidad completa y dividirlas en secciones puede empeorar la calidad del resumen resultante.

También es posible tratar de investigar por otros modelos grandes de lenguaje que se encuentran disponibles para tratar de sobrepasar esta limitante. Además de la cantidad de tokens que puede recibir, se debe evaluar otras características acerca del modelo antes de reemplazar el modelo, como por ejemplo, la velocidad de respuesta, la calidad del resumen, la eficiencia, el costo y qué tan complicado puede ser adaptar el código al nuevo modelo.

[12] https://artificialanalysis.ai/models/gemini-pro

Trabajo futuro

El diseño actual del frontend, aunque es funcional, podría beneficiarse de un rediseño para hacerlo más intuitivo y visualmente atractivo. React.js ya provee formas simples para crear interfaces dinámicas y responsivas. Se podría agregar una paleta de colores, fuentes de texto y diseños más avanzados. Además, agregar elementos interactivos, animaciones y transiciones pueden mejorar considerablemente la experiencia de usuario.

Un trabajo futuro imprescindible es ofrecer opciones de personalización de los resúmenes generados, lo que podría aumentar significativamente la utilidad de la aplicación y transformarla en una herramienta más personal para los usuarios. Por ejemplo, se podría permitir a los usuarios seleccionar el nivel de detalle del resumen, ajustándose a diferentes niveles de conocimiento, como resúmenes más simples para principiantes y más complejos para expertos. También podríamos ofrecer opciones para generar resúmenes cortos y concisos o resúmenes más largos y detallados, dependiendo de las necesidades del usuario. Además, sería de gran utilidad poder proporcionar opciones para adaptar el tono del resumen, permitiendo un estilo más coloquial o más formal.

Se podría también incorporar modelos de aprendizaje profundo que puedan ajustar automáticamente el estilo y la complejidad del resumen basado en las preferencias del usuario. Esto simplifica la idea de la personalización, ya que de lo contrario, en caso de no contar con estos modelos, se debería tener prompts especializados para cada tipo o personalización del resumen.

Para mejorar la precisión de la transcripción en nuestra aplicación, una posible línea de trabajo futuro sería fine-tunear[13] el modelo de Whisper. Esta mejora sería muy útil para videos con audio de baja calidad, que es una situación que se presenta con cierta frecuencia en los videos de OpenFING.

Dadas las transcripciones generadas por Whisper, que incluyen metadatos con los minutos de cada fragmento de transcripción (timestamps), podríamos implementar una funcionalidad que, al buscar un término en el buscador, referencie el momento exacto donde se menciona en el video. Esto permitiría a los usuarios saltar directamente al momento del video donde se discute el tema, mejorando la eficiencia de la aplicación y el aprendizaje de los usuarios.

Otra línea de trabajo futura sería la generación de subtítulos para videos utilizando los timestamps de las transcripciones que genera Whisper. Estos subtítulos podrían ser agregados a la plataforma OpenFING, facilitando el acceso al contenido educativo para usuarios con diferentes necesidades, con dificultades auditivas o quienes prefieren leer el contenido en lugar de escucharlo.

Otra idea para el futuro, que es en cierto modo, fácil de implementar (al menos para una cantidad chica de lenguajes) sería dar la opción de traducir las transcripciones y resúmenes a

[13] https://www.ibm.com/topics/fine-tuning

diferentes idiomas. Actualmente, Whisper dice soportar 57 lenguajes[14], aunque no con todos logra las mismas métricas de performance. En la imagen se puede ver que los idiomas más comunes y más hablados son los que tienen menor tasa de error de palabras por video (*WER, Word Error Rate*).

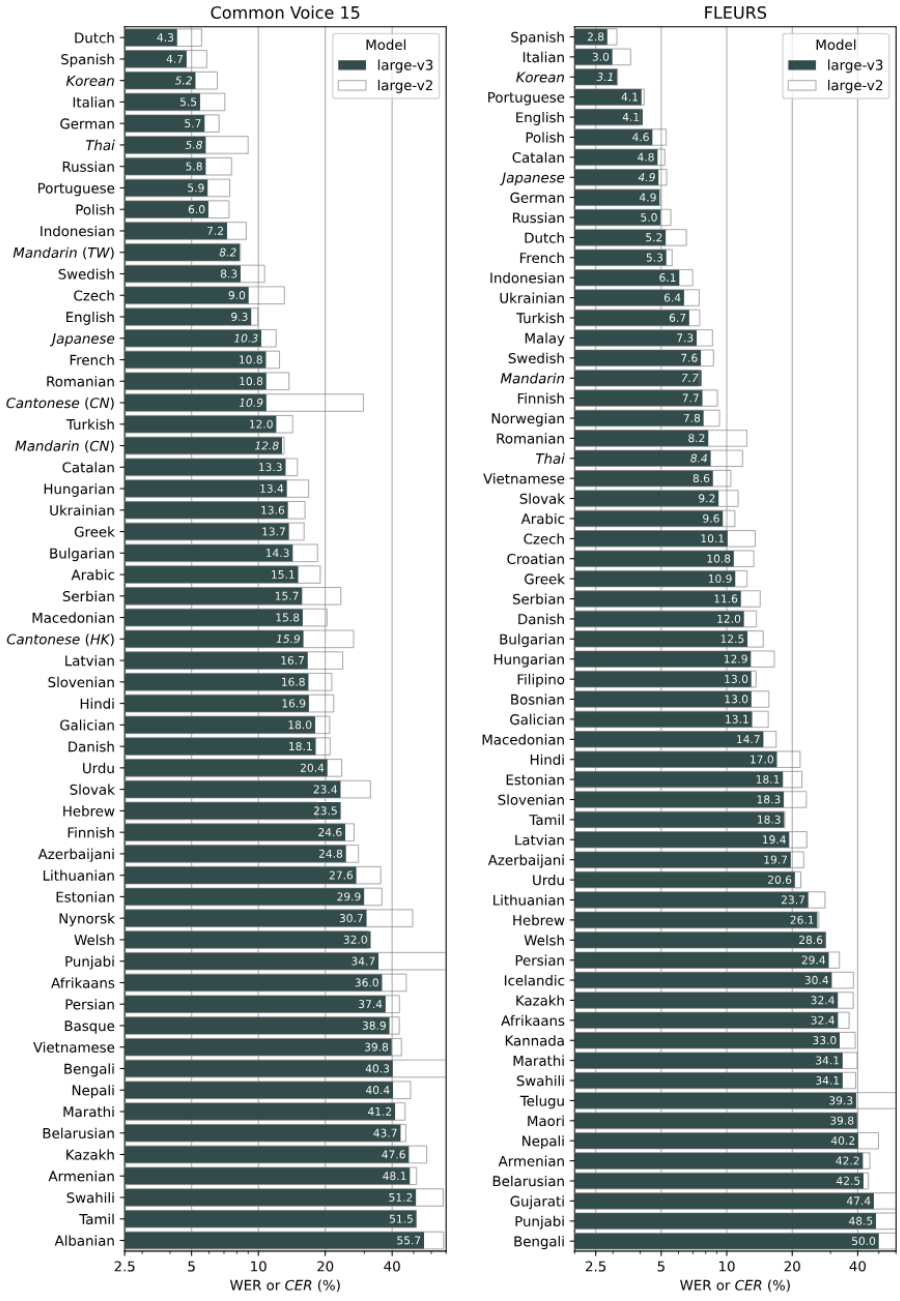


Imagen 1: Métricas WER (tasa de error de palabras) y CER (tasa de terror de caracteres) de los modelos Whisper por idioma, evaluadas en los datasets Common Voice 15 y Fleurs.

[14] https://platform.openai.com/docs/guides/speech-to-text/supported-languages

Conclusiones

En primera instancia, se entiende que el proyecto implicó el manejo de nuevas técnicas, como el uso de base de datos, herramientas de aprendizaje automático, motores de búsqueda y técnicas de scraping. De las cuales se logró entender y hacer uso de cada una de las mismas, siendo estas aplicadas al caso del proyecto.

Basándose en lo expuesto, se puede concluir que se logró hacer una implementación funcional del problema propuesto, donde se garantiza el uso eficiente de recursos y rápidos niveles de respuesta, además de ser una solución escalable y aplicable fácilmente a otros casos. Por una parte, se entiende que al generar resúmenes bajo demanda y almacenar las transcripciones en una base de datos, el sistema garantiza un uso eficiente de los recursos computacionales. En cuanto a la recuperación de datos, el almacenamiento en bases de datos, combinado con la herramienta ElasticSearch, proporciona una manera poderosa para la recuperación rápida y eficiente de información.

Si bien la propuesta planteada es funcional, ésta no es una solución definitiva al mismo, ya que no se considera una solución completa. Se detectaron problemas que surgen de las tecnologías en sí, así como limitaciones propias de la propuesta inicial, que se enfocó principalmente en la funcionalidad del proyecto. A pesar de ello, se logró definir una serie de trabajos futuros y se propusieron soluciones para cada uno de ellos.

Por último, este proyecto no solo beneficia a los estudiantes de la Facultad de Ingeniería de la Universidad de la República, sino que también sienta un precedente para otras instituciones educativas, ya que la tarea de recuperación de vídeos y síntesis, no es dedicada solamente al entorno planteado. El sistema actualmente maneja solamente videos de OpenFING, pero fue pensado e implementado para que sea extendido de manera muy simple a otras plataformas educativas con una mayor cantidad de videos.

Referencias

OpenFing. Retrieved June 24, 2024, from https://open.fing.edu.uy/

Inicio | Facultad de Ingeniería. Retrieved June 24, 2024, from https://www.fing.edu.uy/

MongoDB: The Developer Data Platform | MongoDB. Retrieved June 28, 2024, from https://www.mongodb.com/

FastAPI. Retrieved June 25, 2024, from https://fastapi.tiangolo.com/

Beautiful Soup Documentation — Beautiful Soup 4.12.0 documentation. (n.d.). Crummy. Retrieved June 25, 2024, from https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Gemini 1.0 Pro - Quality, Performance & Price Analysis. (n.d.). Artificial Analysis. Retrieved June 27, 2024, from https://artificialanalysis.ai/models/gemini-pro

guide, s. (n.d.). Gemini - chat to supercharge your ideas. Retrieved June 30, 2024, from https://gemini.google.com/app

openai/whisper: Robust Speech Recognition via Large-Scale Weak Supervision. (n.d.). GitHub. Retrieved June 25, 2024, from https://github.com/openai/whisper

Speech to text. (n.d.). OpenAI API. Retrieved June 28, 2024, from https://platform.openai.com/docs/guides/speech-to-text/quickstart

Speech to text. (n.d.). OpenAI API. Retrieved June 28, 2024, from https://platform.openai.com/docs/guides/speech-to-text/longer-inputs

Speech to text. (n.d.). OpenAI API. Retrieved June 27, 2024, from https://platform.openai.com/docs/guides/speech-to-text/supported-languages

What is Fine-Tuning? (2024, March 15). IBM. Retrieved June 27, 2024, from https://www.ibm.com/topics/fine-tuning