Práctica 3: Asistente inteligente para el transporte público

TAREA 1: Preparación del conjunto de datos.

Implementación:

Se han grabado sonidos de diferentes trenes de la red de Cercanías, como el sonido de las alarmas al abrirse y cerrarse las puertas, así como de los asientos plegables dentro de los vagones al levantarse un pasajero. Además, dichos sonidos se han unido con los audios grabados por otro compañero quien los realizó de forma casera, impactando distintos objetos para simular los golpes en los asientos, y arrastrando una maleta en distintas superficies de la calle. Asimismo, la mayoría de los sonidos de alarmas se han obtenido de diversos vídeos del transporte público en la plataforma de YouTube. De esta forma, se ha obtenido un conjunto de sonidos formado por 60 audios diferentes (20 audios para cada una de las clases), de 5 segundos cada uno.

Dificultades:

- La grabación de sonidos ha sido tediosa al tratarse de una recolección de una gran cantidad de audios. Ya que, a pesar de coger el tren frecuentemente, la diversidad de sonidos es escasa y el acceso a sonidos del transporte público en Internet es limitado. Es por ello por lo que finalmente se ha tenido que crear el conjunto de datos con la ayuda de los audios de otro compañero.

TAREA 2: Modelo clasificador basado en YAMNet.

Implementación:

Se ha empleado principalmente el código definido en el Tutorial 5 ("*Reconocimiento de sonidos personalizados con YAMNet"*). Se han reutilizado las secciones de código para la carga de ficheros de audio y su representación como forma de onda, y el uso del micrófono como fuente de audio en tiempo real. Así como el entrenamiento del modelo YAMNet utilizando aprendizaje por transferencia para adaptarlo a nuestros audios y las clases deseadas ('alarm', 'seat', 'luggage').

Los resultados obtenidos son bastante buenos, ya que no se ha encontrado ningún caso donde haya predicho incorrectamente el tipo de sonido.

Dificultades:

- No se ha encontrado ninguna dificultad. Pero sí se ha requerido un par de horas para entender al completo el código perteneciente al Tutorial 5.

TAREA 3: Integración de modelos de lenguaje.

Implementación:

Para esta tarea, se ha empleado como base el código del Tutorial 6 (*Generación de texto con Transformers.js*). Al igual que en dicho tutorial, se ha utilizado el modelo de lenguaje Qwen2.5-0.5B desde Transformers.js, para la generación de texto.

Por otro lado, se ha definido el siguiente prompt:

```
`You are a worker in the public transport sector in charge of transmitting announcements to passengers.

Your objective is to promote a more respectful and accessible environment, improving coexistence between passengers.

Your responsibility is to generate messages to display on screens next to the affected areas.

You MUST NOT include any type of Python code nor add any "Note: ".


Example 1:

Sound Type: Seat Bumps

Location: Carriage 5

Urgency: Moderate

Context: A passenger has hit the seat when getting up in Carriage 5. Generate a polite message to display on screens reminding passengers to handle the seats with care.

Answer: "Please make sure to handle the seats carefully when getting up to avoid bumps or accidents. Thank you for your collaboration."


Now, generate a message given the following characteristics:

Sound Type: ${sound_type}

Location: ${location}

Urgency: ${urgency}

Context: ${contexto}

Answer:`
```

En el primer párrafo, indicamos al modelo las instrucciones que debe de seguir, entre ellas el papel que desempeña, su objetivo, su responsabilidad, y alguna obligación. El segundo párrafo incluye un ejemplo para orientar al modelo (one-shot prompting). Por último, se le pide al modelo generar una respuesta dadas ciertas características como el tipo de sonido, la ubicación donde ha ocurrido el incidente, la urgencia, y el contexto de la situación.

Además, se ha establecido un número de tokens máximo de 30, y una temperatura de 0.2 para prevenir que el modelo alucine y genere información no relevante a lo que se le pide.

Dificultades:

- La mayor dificultad para esta tarea ha sido definir un prompt decente. Por ejemplo, se ha tenido que cambiar el idioma empleado del español al inglés ya que en español el modelo devolvía peores resultados. Asimismo, se ha tenido que especificar que no incluya ningún tipo de código ni nota en la respuesta debido a que siempre generaba una línea de código o nota como última frase. Sin embargo, actualmente las respuestas que se obtienen no son perfectas ya que a veces se incluyen algunas anotaciones después del mensaje.
- Otra dificultad encontrada es el largo tiempo de espera para la generación y traducción del código. Por ejemplo, el vídeo demostrativo tiene un duración original de 7 minutos, por lo que se ha procedido a su edición para su entrega.

TAREA 4: Integración de sistema de traducción automática.

Implementación:

Para la integración de las traducciones se ha utilizado el mismo modelo Qwen2.5-0.5B que en la tarea anterior. Se ha definido un nuevo prompt:

```
`You are a professional translator.

The output MUST ONLY be the translated sentence.

You MUST NOT include any type of Python code nor Notes.


Example 1:

Original Text: "Please make sure to handle the seats carefully when getting up to avoid bumps or accidents. Thank you for your collaboration."

Translation in Spanish: "Por favor, asegúrese de manejar los asientos con cuidado al levantarse para evitar golpes o accidentes. Gracias por su colaboración."
```

```
Example 2:

Original Text: "Please make sure to handle the seats carefully when getting up to avoid bumps or accidents. Thank you for your collaboration."

Translation in German: "Bitte achten Sie beim Aufstehen darauf, vorsichtig mit den Sitzen umzugehen, um Stöße oder Unfälle zu vermeiden. Vielen Dank für Ihre Zusammenarbeit."


Translate the following text in ${targetLanguage}:

Original Text: "${text}"

Translation in ${targetLanguage}:`
```

En el primer párrafo, indicamos al modelo las instrucciones que debe de seguir, entre ellas el papel que desempeña y algunas obligaciones. El segundo y tercer párrafos incluyen ejemplos para orientar al modelo (multishot prompting), uno para traducción en inglés y otro en alemán. Por último, se le pide al modelo traducir el texto generado en la tarea anterior (*text*) dado en un idioma específico (*targetLanguage*).

Además, se ha establecido un número de tokens máximo de 35, y una temperatura de 0.2 para prevenir que el modelo alucine y genere información no relevante a lo que se le pide.

Dificultades:

- En esta tarea se ha intentado utilizar diversos modelos de lenguaje especializados en la traducción de textos como M2M100, MarianMTModel, Mbart, entre otros. Sin embargo, no ha sido posible utilizar ninguno debido a errores como la indisponibilidad de estos a través de Transformers.js o al tratarse de modelos demasiado grandes. Es por ello por lo que finalmente se ha utilizado el mismo modelo que en la generación de texto, Qwen2.5-0.5B. Como se puede ver en el vídeo demostrativo, los resultados obtenidos no son del todo buenos. Por ejemplo, la palabra en inglés “suitcases” se traduce a “armarios” en español, lo cual es incorrecto. Así como la mala conjugación de verbos en español, “perdan” en vez de “pierdan”.

TAREA 5: Funcionalidad con un servicio text-to-speech.

Implementación:

Para la implementación de la funcionalidad text-to-speech, se ha utilizado el Web Speech API. De esta forma, se ha conseguido leer en alto el texto generado y sus traducciones

correspondientes para aquellos casos en los que las notificaciones auditivas sean adecuadas y necesarias.

Dificultades:

- La principal dificultad encontrada es que el Web Speech API no era capaz de leer en alto un texto en coreano. Como consecuencia, se ha tenido que descartar este idioma por el alemán. Sin embargo, una solución que se podría haber implementado tanto para el coreano como para el inglés y el alemán, sería establecer un Web Speech API para cada idioma ya que actualmente se utiliza el español como defecto para todos.

Reflexión sobre lo aprendido del proceso.

Durante la práctica se ha aprendido lo siguiente:

- La eficiencia de las Web Audio API y el modelo YAMNet como clasificador de audios. El modelo YAMNet nos ha permitido clasificar de forma bastante precisa nuestros audios personalizados a pesar de ser el número de audios de entrenamiento bastante pequeño (48). Esto ha sido gracias al aprendizaje por transferencia.
- A pesar de no obtener resultados perfectos en comparación con otros modelos más modernos como GPT-4o, el modelo Qwen2.5-0.5B ofrece buenos resultados en la generación de texto. Sin embargo, su rendimiento ha disminuido para la tarea de traducción al no estar especializado en esta.
- La implementación de text-to-speech ha sido relativamente sencilla y podría implementarse en futuros proyectos y facilitar la accesibilidad a personas con discapacidad visual.

Conexión de la práctica con su aplicabilidad en la vida real o futuros proyectos.

Esta práctica ha sido de gran utilidad para conocer mejor la aplicación de modelos con ficheros de audio, ya que anteriormente no había trabajado con este tipo de dato. Esto me permitirá adaptarme mejor a futuros proyectos donde se utilicen los audios como dato.

Asimismo, esta práctica está bastante conectada a su aplicabilidad en la vida real ya que aborda problemas actuales en el transporte público. Por ejemplo, la detección de sonidos permite evitar situaciones peligrosas y proteger a los pasajeros, como es el caso de la alarma al cerrarse las puertas. El uso de modelos lenguaje también permiten crear mensajes personalizados según la situación, en vez de mostrar mensajes muy genéricos.

Por último, la generación de mensajes inclusivos en múltiples idiomas o con soporte de texto a voz permite que las notificaciones lleguen a una audiencia diversa, incluyendo personas con discapacidades auditivas y/o visuales o con barreras lingüísticas.