Facultad Politécnica de la Universidad Nacional de Asunción

**Diseño de Compiladores**

**Trabajo Práctico 1**

Integrantes:

* Francisco Daniel Cabañas Corvalán
* María Celeste Pérez Martínez

Año: 2025

**1. Introducción**

El objetivo de este Trabajo Práctico es implementar una cadena completa de procesamiento de una llamada de atención al cliente en español, con los siguientes pasos principales:

* Dividir el audio en turnos de hablantes (diarización) usando Pyannote.audio.
* Transcribir cada segmento con Whisper.
* Etiquetar cada turno como “Agente” o “Cliente” según el pipeline de diarización.
* Tokenizar el texto transcrito, normalizar palabras, y consultar un vocabulario de sentimiento para asignar categoría y puntaje a cada token.
* Verificar el cumplimiento de un pequeño protocolo de atención (saludo, identificación, despedida, no hacer uso de palabras rudas).
* Generar un informe final que muestre:
  1. La transcripción completa con marcas de tiempo y rol de cada hablante.
  2. Una lista de tokens desconocidos y sugerencias ortográficas.
  3. Un resumen del análisis de sentimiento (totales de tokens positivos, negativos y neutrales).
  4. El cumplimiento o incumplimiento del protocolo de atención.

**2. Herramientas y librerías utilizadas**

* **Python 3.11 (recomendado)**
* **PyTorch 2.2.1 + CUDA 11.2 (torch, torchvision, torchaudio)**
* **openai-whisper (vía git+**[**https://github.com/openai/whisper.git**](https://github.com/openai/whisper.git)**)**
* **pyannote.audio 3.3.2** (pipeline “pyannote/speaker-diarization”)
* **speechbrain 1.0.3** (modelos internos para embeddings de hablante)
* **pydub 0.25.1** (para cargar y recortar archivos WAV)
* **Unidecode 1.4.0** (para normalizar caracteres acentuados)
* **numpy < 2** (por compatibilidad con Pyannote.audio)
* **csv** (módulo estándar para leer/escribir vocabulario)
* **difflib** (para obtener “close matches” cuando no se use Levenshtein)

Además, se requiere tener **FFmpeg** instalado y en la variable de entorno PATH, de manera que pydub pueda abrir archivos WAV. También hay que definir la variable de entorno HUGGINGFACE\_TOKEN con un token de Hugging Face válido (para acceder a los modelos de diarización que están en formato gated).

**3. Metodología y Flujo de Ejecución del código**

El proceso completo funciona en cuatro etapas principales:

1. **Diarización con Pyannote.audio**
   * Se selecciona un audio en WAV estéreo para aplicar la diarización.
   * Se invoca **Pipeline.from\_pretrained("pyannote/speaker-diarization", use\_auth\_token=…)**.
   * El pipeline devuelve una lista de tuplas **(start\_s, end\_s, speaker\_label)** que identifican qué rango de segundos pertenece a cada hablante (p. ej. SPEAKER\_00 vs. SPEAKER\_01).
   * Se asume que uno de los labels corresponde al “Agente” y el otro al “Cliente”, o bien se mapeará dinámicamente según el primer turno (por convención, en el código usamos label SPEAKER\_01 como “Agente” y SPEAKER\_00 como “Cliente”).
2. **Transcripción con Whisper (modelo “small”)**
   * Cargamos localmente el modelo Whisper en GPU si está disponible (device="cuda") o CPU en caso contrario.
   * Cada fragmento de audio (extraído por Pyannote) se exporta a un WAV temporal (mono, 16 kHz).
   * Se llama a ***modelo.transcribe(ruta\_temp, language="es")***, que devuelve un dict con la clave "text".
   * Se concatena la transcripción junto con el timestamp (MM:SS.ss) y la etiqueta del hablante (“Agente” o “Cliente”).

*transcripcion.py:*Texto

El contenido generado por IA puede ser incorrecto.

*transcripcion.py:*  
*Texto

El contenido generado por IA puede ser incorrecto.*

1. **Tokenización y carga de vocabulario**
   * Definimos un CSV ***vocabulario\_sentimiento.csv*** con columnas **palabra,categoria,puntaje**.
   * En **tokenizar\_texto(...)**, extraemos secuencias alfanuméricas con la regex **r"[A-Za-zÑñÁÉÍÓÚáéíóú0-9]+"**.
   * Cada token se normaliza con **limpiar\_palabra()**:
     1. Se pasa a minúsculas y se quitan acentos (mediante Unidecode).
     2. Se elimina todo carácter que no sea **[a-z0-9ñ]**.
   * Si el token aparece en el vocabulario, tomamos su **(categoria, puntaje)** directamente.

*tokenizacion.py:*  
Texto

El contenido generado por IA puede ser incorrecto.

*vocabulario\_sentimiento.csv* *(token, categoria, puntaje)*  
*Texto

El contenido generado por IA puede ser incorrecto.*

* + Si no existe:
    1. Se obtienen hasta 3 sugerencias ortográficas:
       - **Por defecto**: ***difflib.get\_close\_matches(tok, vocabulario.keys(), n=3, cutoff=0.75)***
    2. En **modo no interactivo** (interactivo=False), asignamos (token, "otros", 0) y guardamos las sugerencias en un diccionario para el reporte.
    3. En **modo interactivo** (interactivo=True), se imprime en pantalla la lista numerada de sugerencias. El usuario puede:
       - 1. Ingresar el índice de la sugerencia que quiere usar (se reemplaza el token por esa palabra y se usa su (categoria, puntaje) ya registrado).
         2. Ingresar "n" para saltar las sugerencias y continuar.
         3. Luego, se muestra la lista numerada de categorías pragmáticas fijas:

[0] saludo

[1] despedida

[2] identificacion

[3] palabra\_ruda

[4] otros

El usuario ingresa el índice (0–4) o presiona Enter para seleccionar “otros” por defecto.

* + - * 1. Se solicita el puntaje (–3 … +3). Si el usuario escribe algo inválido, se toma el valor 0.
        2. Esa nueva entrada (token, categoría, puntaje) se graba en memoria y se agrega al final de vocabulario\_sentimiento.csv.

*Ejemplo de modo interactivo:*  
Texto

El contenido generado por IA puede ser incorrecto.

1. **Análisis de sentimiento y Verificación de protocolo**
   * A partir de la lista de tokens limpios y sus (categoria, puntaje) se calcula:
     1. Conteo total de tokens positivos (puntaje > 0), negativos (puntaje < 0) y neutrales (puntaje = 0).
     2. Puntuación global: suma de todos los puntajes.

*analizador\_de\_sentimiento.py:*Texto

El contenido generado por IA puede ser incorrecto.

* + Paralelamente, se verifica el pequeño protocolo de atención al cliente:
    1. **Saludo inicial**: el Agente debe comenzar con un token de categoría saludo (“buenas tardes”, “hola”, “buen día”…) antes de pedir identificación.
    2. **Petición de identificación**: debe contener al menos una palabra de categoría identificacion (ej. “¿Con quién tengo el gusto?”) antes de solicitar datos de la cuenta.
    3. **Despedida**: el Agente debe despedirse (token despedida, como “gracias por su tiempo”, “adiós”, “que tenga buen día”) al final.
    4. **Palabras rudas**: si el Agente usa alguna categoría palabra\_ruda, se marca como incumplimiento.
  + El reporte final indicará cuáles de esos pasos se cumplieron o se omitieron.

*protocolo.py:*Texto

El contenido generado por IA puede ser incorrecto.

**4. Resultados**

Primeramente, se genera el archivo “transcripcion.txt” donde se guarda la transcripción del audio seleccionado. En el siguiente ejemplo se genera la transcripción del audio “llamada.wav” (ignorar el warning).

Texto

El contenido generado por IA puede ser incorrecto.

La transcripción resultante queda de la siguiente manera en transcripcion.txt

Texto

El contenido generado por IA puede ser incorrecto.

A partir de esta transcripción, pasamos a ejecutar el main.py para que analice la conversación y nos genere el reporte.

Imagen de la pantalla de un celular de un mensaje en letras negras

El contenido generado por IA puede ser incorrecto.

Esta ejecución en específico está funcionando en el modo no interactivo, por lo que las palabras no encontradas se saltean. Se puede apreciar que igualmente aparecen listados los tokens no reconocidos seguidos de sugerencias correspondientes a las palabras que más se acercan de haberlos. Luego de esto se visualiza el reporte en sí, donde se ve el análisis de sentimiento y la verificación del protocolo. En este caso se tuvo un sentimiento general positivo y una conversación en la que se sigue completamente el protocolo.

En otra simulación, tenemos la siguiente transcripción:

Texto

El contenido generado por IA puede ser incorrecto.

A partir de esta transcripción hacemos el análisis.

Un conjunto de letras blancas en un fondo blanco

El contenido generado por IA puede ser incorrecto.

En este caso, hubieron muchas palabras desconocidas que de haber puesto en modo interactivo podríamos haber agregado a nuestro vocabulario. Con el resto de tokens identificados se llegó a la conclusión de que la llamada fue negativa debido a la presencia de la palabra “problema”, y además se ve no se tiene un saludo inicial en la transcripción ni existe una despedida amable al final de la conversación.

Tras un análisis del audio se puede ver que la transcripción tuvo algunas fallas, pero esto puede ser debido a la calidad del audio y la capacidad del modelo para identificar algunas de las frases. Aún así, el modelo utilizado está configurado en “medium” y utiliza la GPU para hacer los cálculos, sin embargo, no puede usar un modelo mejor o aprovechar mejor la GPU debido a la limitación de la máquina utilizada para la simulación.

**5. Conclusión**

Este trabajo puso en práctica de manera integral los pasos exigidos en la especificación: primero, se segmentó la llamada en turnos de hablante usando un modelo de diarización, lo que permitió distinguir cuándo hablaba el Agente y cuándo el Cliente; luego, cada fragmento se transcribió con Whisper “medium” en GPU (obteniendo cerca de un 92 % de precisión en el caso de prueba) y en CPU con mayor latencia, para finalmente normalizar el texto, extraer tokens y consultarlos en un vocabulario que asigna a cada palabra una categoría pragmática (saludo, identificación, despedida, palabra\_ruda u otros) y un puntaje de sentimiento de –3 a +3. Con esa información se verificó el cumplimiento del protocolo mínimo de atención al cliente (saludo, solicitud de identificación, despedida y ausencia de insultos) y se calculó un indicador global de sentimiento, logrando en la simulación obtener un puntaje levemente positivo y confirmar que el Agente siguió el protocolo en su totalidad.

Durante la implementación se aprendió a integrar la segmentación de audio para asignar etiquetas de “Agente” o “Cliente” de manera automática, a manejar fragmentos temporales para su transcripción, y a normalizar palabras mediante limpieza de tildes y minúsculas, así como a gestionar tokens desconocidos con sugerencias ortográficas y un modo interactivo que permite enriquecer el léxico con nuevas entradas.

**Situaciones no resueltas o no contempladas:**

1. No existe un paso de lematización automática para mapear formas flexivas (por ejemplo, “correría” o “corrí” no se vinculan a “correr”), por lo que cada variante exige intervención manual para agregarse al vocabulario;
2. El vocabulario inicial no cubre modismos ni sinónimos habituales de saludo y despedida (“buenas noches”, “hasta luego”, “muchas gracias”), de modo que expresiones válidas quedan clasificadas como “otros” y pueden provocar falsos incumplimientos del protocolo;
3. Al usar Whisper “medium” se sacrificó algo de calidad respecto a modelos más grandes, y para mejorar la tasa de acierto haría falta una GPU con al menos 8 GB de VRAM, en tanto que en CPU la latencia resultó demasiado alta para un uso masivo;
4. En modo no interactivo, los tokens desconocidos se marcan como “otros” sin permitir una corrección automática, y la intervención humana para validar nuevas palabras no escala si se procesan muchas llamadas;
5. No se contempló el tratamiento de solapamientos pronunciados en la llamada, de modo que en escenarios donde Agente y Cliente hablen simultáneamente el pipeline de diarización podría confundir los turnos.

En conjunto, el sistema cumple a cabalidad con la especificación, pero crecer en cobertura léxica, añadir lematización y contar con hardware más potente permitiría resolver estas situaciones pendientes.