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)**
- **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:

```python
def transcribir_con_diarizacion(ruta_wav: str, modelo):
    print("🎧 Cargando audio con pydub...")
    audio_completo = AudioSegment.from_wav(ruta_wav)
    segmentos = diarizar_audio(ruta_wav)

    # Mapeo de etiquetas del diarizador a roles "Agente"/"Cliente"
    speaker_map = {}
    next_role = "Agente"

    # Abrir archivo de salida
    with open("transcripcion.txt", "w", encoding="utf-8") as f:
        print("\n📝 Transcribiendo y guardando en 'transcripcion.txt'...\n")
        for start_s, end_s, speaker in segmentos:
            # Asignar rol si es la primera vez que aparece este speaker
            if speaker not in speaker_map:
                speaker_map[speaker] = next_role
                next_role = "Cliente" if next_role == "Agente" else "Agente"

            role = speaker_map[speaker]
            start_ms = int(start_s * 1000)
            end_ms   = int(end_s   * 1000)
            fragmento = audio_completo[start_ms:end_ms]

            minutos = int(start_s // 60)
            segundos = start_s % 60
            timestamp = f"{minutos:02d}:{segundos:05.2f}"

            print(f"▶ [{timestamp}] [{role}] …", end="", flush=True)
            texto = transcribir_fragmento_whisper(fragmento, modelo)
            print(" listo.")

            # Escribir línea en el archivo de forma específica para tokenización:
            #   <Rol>: <texto>\n
            linea = f"{role}: {texto}\n"
            f.write(linea)

    print("\n✅ Transcripción completa guardada en 'transcripcion.txt'.")
```

transcripcion.py:

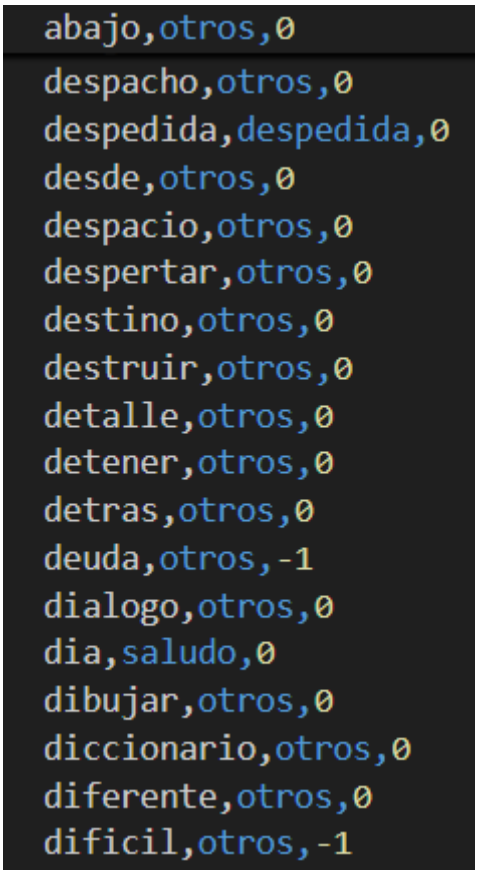
```python
def diarizar_audio(ruta_wav: str):
    """
    Usa Pyannote.audio para obtener segmentos diarizados.
    Requiere que la variable de entorno HUGGINGFACE_TOKEN esté presente.
    Retorna lista de tuplas: (start_s, end_s, speaker_label).
    """
    print("🔍 Iniciando diarización con Pyannote...")
    hf_token = os.getenv("HUGGINGFACE_TOKEN")
    if not hf_token:
        print("[ERROR] No se encontró la variable HUGGINGFACE_TOKEN.")
        sys.exit(1)

    pipeline = Pipeline.from_pretrained(
        "pyannote/speaker-diarization",
        use_auth_token=hf_token
    )
    diarization = pipeline({"uri": "llamada_tp1", "audio": ruta_wav})

    segmentos = []
    for turn, _, speaker in diarization.itertracks(yield_label=True):
        segmentos.append((turn.start, turn.end, speaker))
    return segmentos
```

3. **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:

```python
def tokenizar_texto(
    texto_transcrito: str,
    vocabulario: Dict[str, Tuple[str,int]],
    max_sugerencias: int = 3,
    cutoff: float = 0.75,
    interactivo: bool = False
) -> Tuple[List[Tuple[str,str,int]], Dict[str, List[str]]]:
    raw_tokens = re.findall(r"[A-Za-zÑñÁÉÍÓÚáéíóú0-9]+", texto_transcrito)
    tokens_info: List[Tuple[str,str,int]] = []
    sugerencias: Dict[str, List[str]] = {}

    for tok in raw_tokens:
        # Omitir secuencias puramente numéricas
        if tok.isdigit():
            continue

        tok_clean = limpiar_palabra(tok)
        if not tok_clean:
            continue

        # Si el token ya existe en el vocabulario, anexamos directo
        if tok_clean in vocabulario:
            categoria, puntaje = vocabulario[tok_clean]
            tokens_info.append((tok_clean, categoria, puntaje))
            continue

        # Generar sugerencias ortográficas
        matches = difflib.get_close_matches(
            tok_clean, vocabulario.keys(),
            n=max_sugerencias, cutoff=cutoff
        )
        sugerencias[tok_clean] = matches

        if not interactivo:···
        # ----------------------
        # MODO INTERACTIVO AQUI
        # ----------------------
```

vocabulario_sentimiento.csv (token, categoria, puntaje)



- Si no existe:
  - Se obtienen hasta 3 sugerencias ortográficas:
    - **Por defecto**: ***difflib.get_close_matches(tok, vocabulario.keys(), n=3, cutoff=0.75)***
  - En **modo no interactivo** (interactivo=False), asignamos (token, "otros", 0) y guardamos las sugerencias en un diccionario para el reporte.
  - 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.

4. Se solicita el puntaje (–3 … +3). Si el usuario escribe algo inválido, se toma el valor 0.
5. Esa nueva entrada (token, categoría, puntaje) se graba en memoria y se agrega al final de vocabulario_sentimiento.csv.

Ejemplo de modo interactivo:

```
Token desconocido: 'mira'
  Sugerencias cercanas:
    [0] mirar
    [n] Ninguna de las anteriores
  ¿Reemplazar por alguna sugerencia? (índice o 'n'): 0
  → Usando sugerencia: 'mirar' (categoría='otros', puntaje=0)

Token desconocido: 'alejandra'
  Sin sugerencias cercanas.

  Escoja categoría pragmática para este token:
    [0] saludo
    [1] despedida
    [2] identificacion
    [3] palabra_ruda
    [4] otros
  Ingrese el número de la categoría (o Enter para 'otros'): 4
  Puntaje de sentimiento para este token (–3…+3, defecto=0): 0
```

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

analizador_de_sentimiento.py:

```python
def analizar_sentimiento(tokens: List[str], lexicon_sentimientos: Dict[str, int]) -> Dict[str, object]:

    for tok in tokens:
        peso = lexicon_sentimientos.get(tok)
        if peso is None:
            tokens_no_lexico.append(tok)
            continue

        puntaje_total += peso

        if peso > 0:
            count_positivas += 1
            if peso > max_pos_weight:
                max_pos_weight = peso
                max_pos_word   = tok
        elif peso < 0:
            count_negativas += 1
            if (max_neg_weight == 0) or (peso < max_neg_weight):
                max_neg_weight = peso
                max_neg_word   = tok

    if puntaje_total > 0:
        sentimiento_general = f"Positivo (+{puntaje_total})"
    elif puntaje_total < 0:
        sentimiento_general = f"Negativo ({puntaje_total})"
    else:
        sentimiento_general = "Neutral (0)"

    return {···
```

- 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:

```python
def verificar_protocolo(
    tokens_info_agente: List[Tuple[str,str,int]]
) -> Dict[str, object]:
    # 1) Saludo inicial
    saludo_ok = any(categoria == "saludo" for (_, categoria, _) in tokens_info_agente)
    # 2) Identificación: basta con que exista algún token con categoría "identificacion"
    identificacion_ok = any(categoria == "identificacion" for (_, categoria, _) in tokens_info_agente)
    # 3) Palabras rudas: todos los tokens cuya categoría sea "palabra_ruda"
    palabras_rudas_usadas = [tok for (tok, categoria, _) in tokens_info_agente if categoria == "palabra_ruda"]
    # 4) Despedida amable: basta con que exista algún token de categoría "despedida"
    despedida_ok = any(categoria == "despedida" for (_, categoria, _) in tokens_info_agente)
    return {
        "saludo":         {"ok": saludo_ok},
        "identificacion": {"ok": identificacion_ok},
        "rudas":          {"lista": palabras_rudas_usadas},
        "despedida":      {"ok": despedida_ok}
    }
```

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).

```
(venv) λ python transcripcion.py llamada.wav
🔄 Cargando Whisper 'medium' en dispositivo: cuda
🎧 Cargando audio con pydub...
🔍 Iniciando diarización con Pyannote...
Lightning automatically upgraded your loaded checkpoint from
ilities.upgrade_checkpoint C:\Users\franc\.cache\torch\pyanno
Model was trained with pyannote.audio 0.0.1, yours is 3.3.2.
Model was trained with torch 1.10.0+cu102, yours is 2.2.1+cu

🎤 Transcribiendo y guardando en 'transcripcion.txt'...

▶ [00:01.50] [Agente] … listo.
▶ [00:07.89] [Cliente] … listo.
▶ [00:13.77] [Agente] … listo.
▶ [00:20.16] [Cliente] … listo.
▶ [00:29.41] [Agente] … listo.
▶ [00:44.53] [Cliente] … listo.
▶ [00:47.70] [Agente] … listo.
▶ [00:51.87] [Agente] … listo.
▶ [00:54.99] [Cliente] … listo.
▶ [00:57.84] [Agente] … listo.

✅ Transcripción completa guardada en 'transcripcion.txt'.
```

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

transcripcion.txt

```
Agente: Buenas tardes, le habla Anna de Soporte Tecnico de la Introática Azul. ¿Con quién tengo el gusto?
Cliente: Hola soy Carlos Gomez, compré un televisor anoche y no enciende.
Agente: entiendo Carlos. ¿Podría confirmarme el modelo y si el enchufar los ilumina alguna luz?
Cliente: es un Smart TV X200, al conectarlo la luz frontal parpadea y luego se apaga
Agente: Gracias, por eso un fallo de firmware. Le guiaré paso a paso. Desconecte el cable, espere 30 segundos y
Cliente: Listo Ana, ya enciende correctamente.
Agente: Excelente. He autorizado el sistema y todo está funcionando.
Agente: ¿Algo más en lo que puedo ayudarle hoy?
Cliente: No todo bien, muchas gracias por su ayuda.
Agente: Gracias a usted Carlos, que tenga muy buen día.
```

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

```
(venv) λ python main.py

--- TOKENS NO RECONOCIDOS ---
  - 'tardes'  →  Sugerencias: ['tarde', 'tres']
  - 'habla'  →  Sugerencias: ['hablar']
  - 'entiendo'  →  Sugerencias: ['entender']
  - 'los'  →  Sugerencias: ['lo']
  - 'guiare'  →  Sugerencias: ['guia']
  - 'espere'  →  Sugerencias: ['esperar']
  - 'segundos'  →  Sugerencias: ['segundo']
  - 'presionado'  →  Sugerencias: ['presion']
  - 'esta'  →  Sugerencias: ['estar', 'esto', 'este']
  - 'ayudarle'  →  Sugerencias: ['ayudar']
  - 'tenga'  →  Sugerencias: ['tengo']
  - 'buen'  →  Sugerencias: ['bueno', 'buenas', 'bien']
  - 'compre'  →  Sugerencias: ['comprar']
  - 'enciende'  →  Sugerencias: ['encender', 'entender']
  - 'tv'  →  (Sin sugerencias)
  - 'x200'  →  (Sin sugerencias)
  - 'conectarlo'  →  Sugerencias: ['conectar', 'contar']
  - 'ayuda'  →  Sugerencias: ['ayudar']

--- ANÁLISIS DE SENTIMIENTO ---
Sentimiento general: Positivo (+3)
Palabras positivas: 3
  → Palabra más positiva: gracias, +1
Palabras negativas: 0
  → Sin palabras negativas.

--- VERIFICACIÓN DEL PROTOCOLO (Agente) ---
Saludo inicial: OK
Identificación del cliente: OK
Uso de palabras rudas: Ninguna detectada
Despedida amable: OK

==================== FIN REPORTE ====================
```

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:

transcripcion.txt

```
Agente: este es el titular de la...
Cliente: ¿Eeeh, sí, sí, sí? ¿Soy yo mismo?
Agente: ¿Me puede decir su nombre, por favor?
Cliente: ¿Y Gabriel?
Agente: Mire, señor Gabriel, le llamo de movicelgo para ofrecerle una promoción. Consiste en la instalación d
Cliente: Perdón señorita, pero es que exactamente ¿quién es usted?
Agente: mi nombre es marcela restrepo de movicelgo y estamos llamando
Cliente: Sí, sí, Marcela, disculpe, pero para nuestra seguridad me gustaría comprobar algunos datos antes de
Agente: No, no tengo problema señor.
Cliente: ¿Desde qué teléfono me llama? Es que en la pantallita aquí del mío sale número privado.
Agente: El interno mide 10-04.
Cliente: Ya, y para que departamento de Moiselgo trabaja usted?
Agente: Telemark internetivo.
Cliente: Perdón, ¿me podría dar su identificación de trabajadora de Movicelgo?
Agente: Señor, disculpe, pero creo que toda esa información no es necesaria.
Cliente: entonces pues lamentablemente voy a tener que colgar porque es que no tengo la seguridad de hablar c
Agente: Pero yo le puedo garantizar...
Cliente: la Marcela. Cada vez que yo llamo a Movicelgo antes de poder comenzar cualquier trámite estoy obliga
Agente: Está bien señor, para que esté más tranquilo le informo que mi número de empleada es el 3459-2-B-2.
Cliente: Un momentico mientras lo verifico. No se retire Marcela.
```

A partir de esta transcripción hacemos el análisis.

```
  - 'piensa' → (Sin sugerencias)
  - 'chuta' → (Sin sugerencias)
  - 'pelota' → (Sin sugerencias)
  - 'repetirle' → Sugerencias: ['repetir']
  - 'analista' → Sugerencias: ['analisis']
  - 'ventas' → Sugerencias: ['ventana']
  - 'quejas' → (Sin sugerencias)
  - 'reclamos' → (Sin sugerencias)
  - 'solicitudes' → (Sin sugerencias)
  - 'nuevas' → (Sin sugerencias)
  - 'viejas' → (Sin sugerencias)
  - 'atendidas' → (Sin sugerencias)
  - 'apenas' → (Sin sugerencias)
  - 'entrenando' → (Sin sugerencias)
  - 'tiene' → (Sin sugerencias)
  - 'ni' → (Sin sugerencias)
  - 'preguntando' → (Sin sugerencias)
  - 'asi' → Sugerencias: ['si']
  - 'quiere' → Sugerencias: ['querer']
  - 'conmigo' → (Sin sugerencias)
  - 'identificarse' → (Sin sugerencias)
  - 'momentico' → Sugerencias: ['momento']
  - 'verifico' → Sugerencias: ['edificio']
  - 'retire' → Sugerencias: ['retirar', 'repetir']

--- ANÁLISIS DE SENTIMIENTO ---
Sentimiento general: Negativo (-1)
Palabras positivas: 0
  → Sin palabras positivas.
Palabras negativas: 1
  → Palabra más negativa: problema, -1

--- VERIFICACIÓN DEL PROTOCOLO (Agente) ---
Saludo inicial: Faltante
Identificación del cliente: OK
Uso de palabras rudas: Ninguna detectada
Despedida amable: Faltante

==================== FIN REPORTE ====================
```

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.