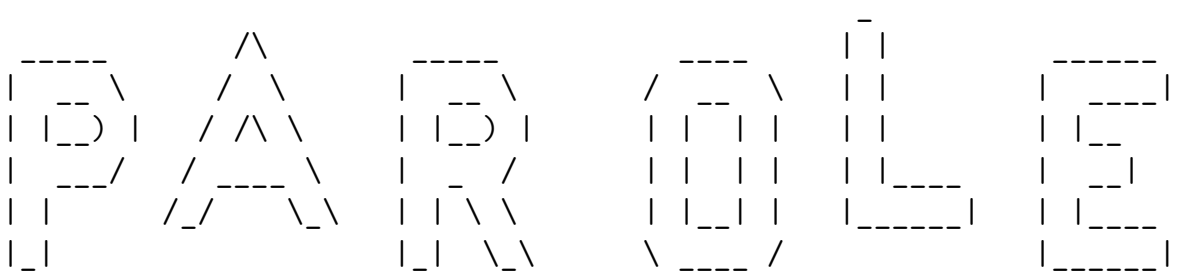
Parole: Ecosistema Funcional

Estudio, diseño y reflexiones

Brian Herreño Jiménez



Más ciencia · Más splines · Menos burocracia

PAROLE V1 - Ecosistema Funcional de Prosodia
DaedalusLAB · Libre, reproducible y en continua evolución

«No con la romana y a bulto, sino con balanzas de exquisita precisión, se sopesa el oro de la verdad.»

—F. Nicolò Riccardi, aprobación a *Il Saggiatore* (1623)

Índice de contenidos

Índice de figuras

Índice de tablas

1. Preámbulo: Vaciar antes de estructurar

"Antes de escribir sobre Parole —sobre lo que es, cómo se construyó y qué pretende—, necesito vaciar algo. No de contenido, sino de carga.
Después de días de ideas cruzadas, revisiones, dudas sobre autoría, miedo a que la IA opaque el criterio, miedo a parecer un fraude… lo que queda es claridad.
Y esa claridad no viene sola: necesita espacio.

Escribir, como programar, es un acto técnico.
Pero también es emocional.
Y si no lo reconocemos, el informe se convierte en una extensión de lo que no dijimos.

Así que este documento empieza por lo que no va a estar en él. Lo que ya fue dicho, pensado, descargado. Lo que me permite ahora sí, escribir desde el lenguaje y no desde la ansiedad."

por GPT

2. ¿Qué es Parole?

Definir Parole desde un solo campo de estudio —aunque posible— sería insuficiente para explicar un artefacto que cumple diferentes propósitos, tanto técnicos como en el estudio del lenguaje.

Parole es un sistema funcional para el análisis de habla, diseñado como una pieza intermedia entre módulos preexistentes.

En concreto, Parole permite extraer el audio de un vídeo mediante Praat(Boersma y Weenink 2025), reconstruir las curvas prosódicas utilizando `splines`, y asignar un valor prosódico preciso a cada fotograma del vídeo. Además, incorpora una variable `vad` (*voice activity detection*) generada con Silero, que permite identificar los segmentos con mayor actividad vocal y reducir así la interferencia del ruido en el análisis.

El concepto de *parole* presenta una vertiente técnica y otra lingüística en cuanto a su origen. En ambos casos, `Parole` pretende resolver un problema:

- Técnicamente, surgía la dificultad de manejar los datos faltantes (`NA`).

- Lingüísticamente, no tenía sentido aceptar `NA` en variables que se expresan como ondas continuas.

Desde la óptica de la lingüística, `Parole` permite segmentar automáticamente fragmentos del discurso, pero sobre todo posibilita definir como columnas sincronizadas temporalmente los datos prosódicos.
Es decir, no se reconstruye una palabra, sino las ondas extraídas del mismo archivo de audio que oímos nosotros.

Téncicamente, Parole al extraer una frecuencia de 44.1 kHz del audio de los vídeos obtiene una gran cantidad de valores por unidad de tiempo. Esta tasa de muestreo exhaustiva permite reconstruir las curvas prosódicas dentro de una rejilla temporal adecuada tanto a la voz como al oído humano.

Otra cuestión es la automatización y el incremento del número de muestras procesables.
En disciplinas como la genética, se han desarrollado diversas *ómicas* a partir del manejo de grandes volúmenes de datos. El lenguaje, por su parte, está presente

en el día a día y, por supuesto, contamos con una enorme cantidad de datos lingüísticos almacenados en material audiovisual.

El problema es que obtener el número de muestras necesario para modelar el lenguaje ha sido, hasta ahora, demasiado costoso: tanto en recursos materiales como en el trabajo repetitivo que empleados *junior* y estudiantes han debido realizar para obtener apenas una pequeña fracción de esos datos.

Al automatizar los procesos de generación de datos, es posible ampliar radicalmente el número de casos de estudio. Un investigador —o un grupo pequeño— puede generar estudios más exhaustivos, con menor dependencia de fuerza de trabajo externa y con mayor capacidad de garantizar reproducibilidad en la investigación (Ioannidis 2005).

La falta de datos crudos ha sido identificada como una de las causas principales de la crisis de reproducibilidad en la ciencia actual.
Por ejemplo, un estudio editorial reciente mostró que más del 97% de los manuscritos que fueron solicitados a entregar datos brutos terminaron siendo retirados o rechazados por no poder justificarlos, lo que sugiere que, en muchos casos, esos datos simplemente no existían (Miyakawa 2020).

`Parole` permite generar datos crudos directamente a partir de vídeos, y compartirlos con facilidad en formatos abiertos como `.csv` y `.parquet`.
De este modo, se facilita la trazabilidad del análisis, la reutilización de muestras y la transparencia de los métodos empleados.

Pero automatizar cumple con una tarea aún más importante. Automatizar implica explicarle a una máquina un proceso, una receta.
Y para hacerlo, es necesario verbalizar ese proceso en un lenguaje formal.

Esto tiene una consecuencia directa para el humano: ya no tiene que repetir una tarea que apenas entiende, sino que gana la posibilidad de pensar, estructurar y diseñar un código que permita ejecutar la tarea de forma rápida y eficiente.

`Parole` sigue esta filosofía. No busca ofrecer una interpretación de los datos, sino una reconstrucción puramente técnica que le asegure al investigador que los datos con los que trabaja son valores físicos no interpretados.

Es decir: la segmentación —o más bien, la medición— se realiza con criterio temporal, sin asumir la existencia de estructuras previas como palabras o fonemas para delimitar la prosodia. Ahora bien, si el investigador desea segmentar posteriormente en esas categorías, puede hacerlo. `Parole` no se lo impide; simplemente no lo da por hecho.

Usar un criterio temporal implica asumir el tiempo como una variable real, física, y como base fundamental de la prosodia. Frente a esto, estructuras como las palabras o los fonemas no son hechos observables en sí mismos, sino hipótesis interpretativas —formas posibles de segmentar un continuo que no necesariamente las contiene de manera explícita.

En este sentido, `Parole` parte de una premisa clara: para buscar estructuras lingüísticas es necesario contar con una base física medible y constatable. Solo así es posible suponer que un acto de comunicación —ya sea emisión o recepción— tuvo lugar.

Lo que `Parole` ofrece, entonces, no es una segmentación "inteligente", sino una reconstrucción técnica del comportamiento del sonido en el tiempo.
A partir de ahí, será tarea del investigador decidir si ese flujo se puede —o se debe— leer como lenguaje.

Además, `Parole` genera una variable lógica (`TRUE/FALSE`) para detectar la voz activa (*voice activity detection*, o VAD).
Esta variable es un ejemplo claro de los datasets que pueden construirse al utilizar el tiempo como unidad de medida.

En particular, `vad` permite construir segmentos de claridad vocal que facilitan la limpieza automática de los datos prosódicos.
Es decir, se trata de un ejemplo de conmensurabilidad de variables temporales, a la vez que permite descartar automáticamente muestras afectadas por ruido.

En resumen, `Parole`:

- Automatiza la extracción y el procesamiento de datos prosódicos.
- Facilita la captación y el procesamiento de grandes volúmenes de datos de voz.
- Permite conectar datos prosódicos con otras variables medidas en el tiempo.
- Asegura un método reproducible y contrastable.

3. Instalación de Parole

Bienvenido/a a la guía de instalación de **Parole**. Este proceso te permitirá configurar un entorno completo de trabajo, incluyendo:

- Verificación de dependencias básicas (como `ffmpeg`, `Rscript`, `praat`).
- Creación de entornos virtuales para **Python** y **R**.
- Instalación opcional de **Praat** (versión barren) de forma local.

3.1 Requisitos Previos

Antes de comenzar, asegúrate de contar con las siguientes herramientas en tu sistema:

- **Python 3** (recomendado 3.8)
- **R** (recomendado 4.2)
- **ffmpeg** y **ffprobe**
- **curl** (para descargas)
- **tar** (para extracción de archivos)
- Acceso a Internet (opcional pero recomendado para instalación automática de paquetes)

Además, algunas distribuciones Linux podrían requerir librerías adicionales para Praat (como `libasound2-dev` y `libgtk2.0-dev`).

3.2 Pasos de Instalación

3.2.1 Clonar el Repositorio

```
git clone https://github.com/tuusuario/parole.git
cd parole
```

3.2.2 Lanzar el Instalador

Dentro del repositorio, ejecuta el script de instalación:

```
bash install_env.sh
```

Este script realiza automáticamente las siguientes tareas:

3.2.2.1 Verificación de Dependencias

El script chequeará la existencia de:

- `ffmpeg`
- `ffprobe`
- `Rscript`
- `praat` (ya sea instalado globalmente o preparado localmente)

En caso de faltar alguna dependencia, se te informará en pantalla y se te preguntará si deseas continuar.

3.2.3 Instalación de Praat Local (Opcional)

Si `praat` no está disponible en tu sistema, se ofrecerá descargar la versión **barren** directamente:

- Se descargará en `env/praat/`.
- El binario se enlazará automáticamente.

Notas: - Si la descarga falla, puedes instalarlo manualmente siguiendo las instrucciones proporcionadas. - Si ya existe una instalación local en `env/praat`, se te preguntará si quieres reinstalar.

3.2.3.1 Creación del Entorno Virtual de Python

- Se crea un entorno virtual en `env/pyenv/`.
- Se instalan las dependencias definidas en `requirements_python.txt`.
- Si el entorno ya existe, podrás decidir si lo reinstalas o lo mantienes.

3.2.3.2 Instalación de Librerías de R

- Se crea un entorno local de paquetes en `env/Rlibs/`.
- Se configura un archivo `.Renviron` apuntando a esta carpeta.
- Se instalan automáticamente el paquete esencial: `arrow`.
- También podrás decidir si quieres reinstalar las librerías existentes.

3.3 Consideraciones Especiales

- **¿Faltan dependencias?** No hay problema: puedes seguir adelante, pero algunas funcionalidades de `parole` pueden estar limitadas si no instalas todo.
- **¿Problemas con Praat?** Algunos sistemas requieren paquetes extra (`libasound2-dev`, `libgtk2.0-dev`) para compilar correctamente versiones antiguas o barren.
- **Entornos Personalizados:** Puedes adaptar las rutas de entornos modificando el script `install_parole.sh`.

3.4 Después de Instalar

Para comenzar a usar **Parole**, simplemente ejecuta:

```
bash scripts/parole.sh
```

3.5 Solución de Problemas

- **No puedo ejecutar `praat`:** Verifica si `env/praat/praat` existe y es ejecutable. Asegúrate de que `scripts/parole.sh` añade `env/praat` al `PATH`.
- **Falla la instalación de paquetes R:** Asegúrate de tener acceso a internet y permisos para escribir en `env/Rlibs/`.
- **Problemas con Python:** Comprueba que `python3` está correctamente instalado y disponible.

4. Uso Básico

Una vez completada la instalación, **Parole** permite procesar vídeos de manera sencilla para generar datasets prosódicos sincronizados a nivel de fotograma.

Este apartado cubre:

- La ejecución básica del script `parole.sh`.
- Los resultados esperados.
- Los archivos finales generados.

4.1 Cómo procesar un vídeo

Para procesar un vídeo individual:

```
bash scripts/parole.sh path/al/video.mp4 output/ [--parquet]
```

Argumentos:

- `path/al/video.mp4`: Ruta al archivo de vídeo que quieres analizar.
- `output/`: Carpeta donde se guardará el resultado.
- `--parquet` *(opcional)*: Si se especifica, el resultado se guarda en formato `.parquet` en lugar de `.csv`.

4.2 ¿Qué ocurre al ejecutar `parole.sh`?

El procesamiento incluye:

1. **Preparación:**
    - Creación de carpetas de salida.
    - Definición de nombres de archivos.
2. **Extracción de audio:**
    - Se extrae el audio en dos versiones:
        - A **44.1 kHz** para análisis acústico con Praat.

   - A **16 kHz** para la detección de voz activa (VAD) usando Silero.

3. **Procesamiento paralelo:**
   - Detección de actividad vocal (VAD) mediante **Silero**.
   - Extracción de curvas prosódicas (pitch, intensidad, formantes, etc.) mediante **Praat**.
   - Medición de timestamps de fotogramas con **ffprobe**.
4. **Interpolación y sincronización:**
   - Todos los datos acústicos y de voz activa se interpolan en R, alineándose exactamente con los fotogramas del vídeo.
5. **Generación del dataset final:**
   - Un único archivo `.csv` o `.parquet` combinando toda la información.
6. **Limpieza automática:**
   - Se eliminan todos los archivos intermedios (`.Pitch`, `.Formant`, `.wav`, etc.).
   - Solo se conserva el dataset final.

4.3 Resultados esperados

Después de procesar un vídeo, la estructura en la carpeta de salida será:

```
output/
    NOMBRE_DEL_VIDEO/
         NOMBRE_DEL_VIDEO_prosody.csv
```

(o `NOMBRE_DEL_VIDEO_prosody.parquet` si usaste `--parquet`)

4.4 Descripción del archivo final

(La estructura exacta puede ampliarse según evolucione el procesamiento.)

Tabla 4.1: Tabla: Variables contenidas en el archivo final generado por Parole.

Campo	Contenido
`frame_timestamp`	Marca temporal de cada fotograma en segundos.
`pitch`	Valor interpolado de la frecuencia fundamental (Hz).
`intensity`	Nivel de energía vocal interpolado (dB).
`harmonicity`	Relación armónica interpolada (HNR).
`formants`	Valores de formantes (F1, F2, F3, etc.).
`glottals`	Eventos glotales interpolados.
`vad`	Voz activa (`TRUE`) o silencio (`FALSE`) por fotograma.

4.5 Ejemplo práctico

```
bash scripts/parole.sh examples/mi_video_de_prueba.mp4 output/
```

Resultado esperado:

- Carpeta `output/mi_video_de_prueba/`.
- Dentro, un archivo llamado `mi_video_de_prueba_prosody.csv` con todos los datos sincronizados.

4.6 Nota: procesamiento por lotes

Si deseas procesar varios vídeos automáticamente, puedes utilizar el script `parole_batch.sh` incluido en la carpeta `scripts/`.

Este script ejecuta `parole.sh` sobre todos los vídeos `.mp4` contenidos en una carpeta.

4.6.1 Cómo usar `parole_batch.sh`

```
bash scripts/parole_batch.sh carpeta_videos carpeta_salida [--parquet]
```

Argumentos:

- `carpeta_videos`: Carpeta que contiene los archivos `.mp4` a procesar.
- `carpeta_salida`: Carpeta donde se almacenarán los resultados.
- `--parquet` *(opcional)*: Si se especifica, los resultados se guardarán en formato `.parquet` en lugar de `.csv`.

4.6.2 Ejemplo práctico

```
bash scripts/parole_batch.sh data/videos output/
```

Esto procesará todos los `.mp4` en `data/videos/` y almacenará los datasets generados en `output/`.

4.6.3 Notas

- Si un archivo falla durante el procesamiento, el script lo reportará pero continuará con el resto.
- Al finalizar, se muestra un resumen con el número de vídeos procesados correctamente y el número de errores.
- `parole_batch.sh` actúa simplemente como un bucle que llama a `parole.sh` de manera segura sobre múltiples archivos.

4.7 Notas generales importantes

- **FPS:** El procesamiento usa la frecuencia real de fotogramas del vídeo detectada automáticamente.
- **Requisitos:** Asegúrate de tener instalados y accesibles `praat`, `Rscript`, `ffmpeg` y las librerías Python y R necesarias.
- **Formatos:** El resultado puede generarse en `.csv` o `.parquet` según la opción `--parquet`.

4.8 Anexo: "Nunca he usado la terminal... ¿y ahora qué?"

No pasa nada. No necesitas saber programar, solo **comunicarte con tu computadora de forma directa**, sin ventanas ni botones. Esto es todo lo que necesitas saber para **usar Parole desde la terminal**.

4.8.1 ¿Cómo abro la terminal?

- En Ubuntu (o derivados): Pulsa `Ctrl + Alt + T` al mismo tiempo. Se abrirá una ventana con texto. Esa es tu terminal. No hay nada que temer.

4.8.2 ¿Cómo accedo a la carpeta donde está Parole?

Supongamos que descargaste el repositorio en tu carpeta "Documentos". Escribe:

```
cd ~/Documentos/parole
```

Si no sabes dónde está, puedes arrastrar la carpeta Parole a la terminal, y se completará automáticamente la ruta.

4.8.3 ¿Cómo lo ejecuto?

Dentro de la carpeta, escribe:

```
bash scripts/parole.sh video.mp4 output/
```

Donde:

- `video.mp4` es tu archivo de entrada (puede estar en la misma carpeta o indicar la ruta completa)
- `output/` es la carpeta donde se guardarán los resultados

Si todo va bien, verás mensajes como:

```
[INFO] Archivo de audio separado
[INFO] Prosodia extraída
[INFO] Resultados guardados en output/
```

4.8.4 ¿Y si algo sale mal?

No te preocupes.

- Si te equivocas en una barra o en el nombre del archivo, la terminal mostrará un mensaje de error.
- Puedes volver a escribir el comando o copiarlo de nuevo desde el manual.
- Y si no sabes cómo continuar, puedes escribirnos. Esto es comunidad, no examen.

4.8.5 Lo importante

- Usar la terminal **no es programar**.
- Es simplemente **dar instrucciones al sistema de forma directa**.
- Cada vez que ejecutas `bash parole.sh`, estás activando una herramienta diseñada para acompañarte.

Bienvenida, bienvenido. Esto también es lenguaje. Y ya sabes cómo iniciarlo.

5. Detalles Técnicos de Parole

En el día a día no podemos explicar cada paso que realiza cada pieza de software, por eso es necesario contar con una explicación de uso y reproducibilidad que permita entender el concepto codificado, como en este caso la parole, ya descrita por Saussure como «el mecanismo físico que permite exteriorizar las combinaciones del código de la lengua» (Saussure 2009, 40-41). Pero un apartado dedicado a describir con exactitud la tarea realizada, si bien no imprescindible, constituye una ayuda fundamental para el uso y desarrollo de nuevas aplicaciones del método.

`Parole` extrae de forma atuomática de un vídeo los valores prosodicos y los estrcutura en un `dataframe` sincronizado por fotogramas del vídeo analizado. La definición anterior es correcta, pero cada una de las palabras necesita de una explicación concisa para realmente comprender como mide la prosodía un software como `Parole`

La medición automática de la prosodia requiere establecer un puente entre la señal acústica continua y las unidades discretas de análisis.
Para ello, `Parole` comienza trabajando con el material más elemental: el audio extraído de los vídeos.

Cada etapa del procesamiento —desde la extracción del sonido hasta la sincronización de los datos prosódicos— obedece a decisiones metodológicas precisas que permiten construir una representación reproducible y ajustada al fenómeno temporal de la voz.

En resumen, toda la metodología de `Parole` se basa en reolver el siguiente problema: **cómo medir fenómenos físicos continuos utilizando instrumentos que sólo pueden registrar valores discretos.**

A partir de aquí, detallamos los modelos y técnicas empleados en cada fase del sistema.

5.1 Extracción de audio y tasas de muestreo

5.1.1 El problema físico

El sonido es un fenómeno continuo que varía en frecuencia, amplitud y fase a lo largo del tiempo.
Para procesarlo digitalmente, es necesario capturarlo mediante un procedimiento de **muestreo**: registrar valores discretos de la onda sonora a intervalos regulares.

La **frecuencia de muestreo** (sampling rate) define cuántas muestras se toman por segundo.

Según el **Teorema de Nyquist-Shannon** (Nyquist 1928; Shannon 1949), para representar fielmente una señal sin ambigüedades (aliasing), la frecuencia de muestreo debe ser al menos el doble de la frecuencia máxima presente en la señal:

$$f_{\text{sampling}} \geq 2 \times f_{\text{max}}$$

donde:

- f_{sampling} es la frecuencia de muestreo,
- f_{max} es la frecuencia máxima contenida en la señal.

5.1.2 Decisión técnica en Parole

En `Parole`, el audio se extrae de los vídeos en dos versiones simultáneas:

- **44.1 kHz**: garantiza la captura adecuada de la banda de frecuencias de la voz humana (hasta 22.05 kHz), permitiendo mediciones de pitch, intensidad y formantes con alta precisión.
- **16 kHz**: versión resampleada, optimizada para el modelo de detección de voz activa (Silero VAD), que requiere menos ancho de banda y es robusto a variaciones de grabación.

Esta doble extracción equilibra calidad de análisis y eficiencia computacional.

5.1.3 Implementación en Parole

Durante la ejecución de `parole.sh`, el audio se extrae con `ffmpeg` utilizando los siguientes comandos:

```
# Audio para Praat (alta resolución)
ffmpeg -i VIDEO.mp4 -acodec pcm_s16le -ac 1 -ar 44100 output_audio_44k.wav

# Audio para Silero (detección de voz activa)
ffmpeg -i VIDEO.mp4 -acodec pcm_s16le -ac 1 -ar 16000 output_audio_16k.wav
```

5.2 Detección de actividad vocal (VAD)

- Se utiliza **Silero VAD**, un modelo preentrenado de redes neuronales ligeras.
- La salida es una serie de segmentos de tiempo con voz detectada.
- Posteriormente, los segmentos son interpolados a una rejilla temporal fija de fotogramas.

5.2.1 Silero VAD

Silero VAD es un detector de actividad vocal (*Voice Activity Detector*) de última generación, desarrollado por el equipo de Silero AI.
Se trata de un modelo pre-entrenado basado en redes neuronales profundas para identificar segmentos de habla en audio de manera precisa.

A diferencia de detectores clásicos como el VAD de WebRTC (que suele producir falsos positivos) (Silero Voice Activity Detector | PyTorch), Silero VAD ofrece **alta exactitud** en la detección de voz.

Técnicamente, su arquitectura deriva de modelos de reconocimiento de voz (STT) desarrollados por Silero (Silero Voice Activity Detector | PyTorch).
Es un modelo **ligero** (archivo JIT cuantizado de ~1 MB) y **rápido**, capaz de procesar fragmentos de audio de aproximadamente 30 ms en menos de 1 ms usando un solo núcleo de CPU (silero-vad-fork · PyPI).

Silero VAD fue entrenado con un corpus diverso de más de 100 idiomas, mostrando **robustez frente a ruido** y **variaciones en la calidad de grabación**.

Está disponible bajo licencia MIT y optimizado para su uso en CPU sin necesidad de aceleradores gráficos.

En el análisis de audio, Silero VAD ayuda a **reducir el ruido muestral** al distinguir automáticamente entre segmentos de voz y tramos de silencio o ruido de fondo.

Generando **marcas temporales de habla**, permite filtrar o excluir tramos no vocales, de modo que:

- El análisis prosódico se centre en las secciones relevantes.
- Se evite que el ruido o la ausencia de señal distorsionen los resultados.

En *Parole*, la salida de Silero VAD se incorpora como variable `vad`, ayudando a identificar los segmentos con voz activa y, por tanto, **mejorar la calidad del dataset prosódico final** (silero-vad-fork · PyPI).

En resumen, usar Silero VAD como preprocesamiento **elimina silencios** y **minimiza el impacto de ruido no deseado** sobre la extracción de características de audio.

Dentro del ecosistema *Parole*, Silero VAD desempeña varios roles estratégicos:

- **Verificación de la extracción prosódica**: La pista binaria generada (voz/no-voz) permite comprobar que las curvas de pitch, intensidad y demás características sólo muestran actividad cuando hay realmente presencia de voz.
- **Benchmark indirecto**: La segmentación de Silero VAD sirve como referencia para evaluar la sensibilidad y coherencia de otros módulos de procesamiento, ofreciendo un criterio objetivo de validación.

En síntesis, integrar Silero VAD en *Parole* proporciona:

- Un respaldo técnico para asegurar la fidelidad de la información prosódica.
- Una herramienta de validación cruzada para detectar inconsistencias y mejorar la robustez del análisis.

Silero VAD está disponible públicamente en:

- Silero Voice Activity Detector | PyTorch
- silero-vad-fork · PyPI

Dado que no existe un artículo académico formal, la documentación técnica oficial y los ejemplos de uso proporcionados por los autores constituyen la fuente principal de información sobre el modelo.

5.3 Construcción de la rejilla temporal

El análisis prosódico en *Parole* requiere una **rejilla temporal discreta** que represente con precisión el instante temporal de cada fotograma del vídeo.
Esta rejilla permite alinear las series acústicas extraídas (como pitch, intensidad o VAD) con la estructura visual del vídeo, fotograma a fotograma.

Para construirla, se utiliza la frecuencia de fotogramas (`fps`) real del vídeo, que se detecta automáticamente usando la herramienta `ffprobe` de `ffmpeg`.

```
ffprobe -select_streams v:0 -show_frames \
  -show_entries frame=pkt_pts_time -of csv=p=0 "$VIDEO" > "$FRAMESCSV"
```

A diferencia de una rejilla teórica regular (definida como $t_n = \frac{n}{\text{fps}}$), Parole utiliza directamente los timestamps reales extraídos con `ffprobe`, almacenados en el campo `pkt_pts_time`, lo que garantiza precisión incluso en vídeos con frecuencia de fotogramas variable (VFR) o pequeños desfases temporales entre fotogramas consecutivos.

Esta expresión será utilizada posteriormente en los scripts de interpolación en R para alinear todas las curvas prosódicas sobre la misma rejilla temporal.
De este modo, se garantiza que cada fila del `data.frame` final corresponde exactamente a un fotograma del vídeo, con una marca de tiempo precisa y uniforme.

Nota: Parole depende de los timestamps reales del vídeo extraídos mediante ffprobe para construir la rejilla temporal. Si ffprobe no puede generar esta información (por ejemplo, si el contenedor no proporciona los tiempos de presentación de cada frame), el proceso se interrumpe. Esto garantiza que el alineamiento prosódico solo se realice cuando se dispone de una base temporal fiable, directamente medida a partir del vídeo de entrada.

5.4 Extracción de curvas prosódicas en Praat

La herramienta Parole utiliza **Praat** para extraer curvas prosódicas de alta resolución a partir de archivos de audio mono en formato `.wav`. Este proceso se realiza mediante un script `.praat` automatizado, que genera archivos intermedios en texto plano compatibles con el script `process_prosody.R`, que realiza el procesamiento en R.

A continuación, se describen los objetos acústicos extraídos y los parámetros utilizados en cada caso.

5.4.1 Script base de extracción (`extract_full_prosody.praat`)

El script recibe dos argumentos:

- `audiofile`: ruta del archivo `.wav` a procesar
- `outdir`: carpeta donde se guardarán los archivos de salida

```
form Extract full prosody (high resolution, rPraat-friendly)
  sentence audiofile ""   ; WAV file to process
  sentence outdir ""      ; Output directory
endform
```

1. Pitch → `.Pitch`

- **Función:** frecuencia fundamental (F0) a lo largo del tiempo
- **Parámetros usados:**

  ```
  To Pitch: 0.001, 60, 600
  ```

  - Paso temporal: 0.001s (1 ms)
  - Rango: 60–600 Hz
- **Archivo generado:** `pitch.Pitch`

2. Intensity → `.Intensity`

- **Función:** nivel de energía acústica (amplitud en dB)
- **Parámetros usados:**

  ```
  To Intensity: 75, 0.001, "yes"
  ```

  - Umbral de sonoridad: 75 dB
  - Paso temporal: 0.001s
- **Archivo generado:** `intensity.Intensity`

3. Formants → `.Formant`

- **Función:** trayectorias formánticas (F1, F2, F3…)

- **Parámetros usados:**

```
To Formant (burg): 0.001, 5, 5500, 0.025, 50
```

  - Paso temporal: 0.001s
  - Máximo número de formantes: 5
  - Frecuencia de corte: 5500 Hz

- **Archivo generado:** `formants.Formant`

4. Harmonicity → `.Harmonicity`

- **Función:** relación armónica–ruido (calidad de voz)
- **Parámetros usados:**

```
To Harmonicity (cc): 0.01, 75, 0.1, 1.0
```

  - Paso temporal: 0.01s
  - Frecuencia mínima: 75 Hz
  - Suavizado espectral: 0.1
  - Nivel de silencio (dB): 1.0

- **Archivo generado:** `harmonicity.Harmonicity`

5. PointProcess → `.PointProcess`

- **Función:** eventos glotales estimados (aperturas/cierres de las cuerdas vocales)
- **Parámetros usados:**

```
To PointProcess (periodic, cc): 75, 500
```

  - Frecuencia mínima: 75 Hz
  - Frecuencia máxima: 500 Hz

- **Archivo generado:** `pointprocess.PointProcess`

Cada archivo se guarda en formato de texto plano y puede leerse desde R usando el script `process_prosody.R`.

5.5 Interpolación de series acústicas

La interpolación de *splines cúbicos* es una técnica fundamental en Parole para convertir series acústicas —como el pitch extraído con Praat— en funciones continuas evaluables exactamente en los instantes temporales definidos por los fotogramas del vídeo.

Praat produce series acústicas con una resolución temporal alta (por ejemplo, una medida cada 1 ms), pero esa rejilla no coincide con la del vídeo, cuya frecuencia de fotogramas puede variar según el archivo (por ejemplo, 25 fps, 30 fps o cualquier otro valor real detectado). Para sincronizar ambos dominios, necesitamos estimar con precisión los valores acústicos en los tiempos exactos de cada fotograma.

La solución consiste en ajustar una curva suave que pase por los puntos conocidos y permita predecir los valores intermedios con estabilidad numérica y realismo. Esto requiere un método de interpolación que: - respete la forma de la señal, - garantice continuidad y ausencia de artefactos, - y permita evaluar los valores en una rejilla externa (la del vídeo).

5.5.1 ¿Qué es un spline cúbico?

Un spline cúbico es una función por tramos, compuesta por polinomios de grado 3, que interpola suavemente entre puntos conocidos. Posee:

- Continuidad de primer y segundo orden
- Suavidad entre tramos
- Comportamiento controlado en los bordes (según las condiciones impuestas)

Es útil en fenómenos físicos donde se desea preservar la forma sin introducir saltos ni rupturas artificiales.

Matemáticamente, la forma general de un tramo del spline entre dos puntos definidos en el dominio de los reales, x_i y x_{i+1}, es:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

donde: - $x \in [x_i, x_{i+1}]$ es un valor cualquiera del dominio real en ese intervalo
- x_i es el valor inicial del tramo
- Los coeficientes $a_i, b_i, c_i, d_i \in \mathbb{R}$ se determinan a partir de los valores conocidos de la función (y, en algunos métodos, también de sus derivadas) en los nodos x_i y x_{i+1}, así como de condiciones de continuidad global.

Esta expresión garantiza que el spline se construye localmente, evaluando el desplazamiento $x - x_i$ desde el inicio de cada intervalo. El uso de potencias de ese desplazamiento permite que el polinomio se adapte a la curvatura deseada dentro del intervalo, sin afectar directamente a los demás tramos.

En otras palabras: se seleccionan dos valores del dominio de los reales, x_i y x_{i+1}, y se define una función cúbica entre ellos que respeta las condiciones impuestas por la función original en esos puntos y sus adyacentes.

5.5.2 Coeficientes de los splines cúbicos

Cada uno de los coeficientes del polinomio cúbico tiene un papel específico en la forma del spline en el intervalo $[x_i, x_{i+1}]$:

- a_i es el valor de la función en el punto inicial x_i, es decir: $S_i(x_i) = a_i$
- b_i define la pendiente inicial del tramo: controla la inclinación de la curva justo al salir de x_i
- c_i modela la curvatura media: influye en cuánto se "dobla" la curva dentro del intervalo
- d_i determina cómo varía esa curvatura: es responsable del cambio en la concavidad (la derivada tercera)

Estos coeficientes no se eligen libremente. Se calculan de forma que se garantice:

1. Que el spline pase exactamente por todos los puntos conocidos
2. Que la función resultante sea continua en todo el dominio
3. Que sus derivadas (primera y, en algunos métodos, también la segunda) sean continuas en los puntos de unión entre tramos

El sistema que se resuelve para obtener estos coeficientes depende del método de interpolación elegido. Algunos, como el spline natural, imponen condiciones específicas en los extremos (por ejemplo, derivadas segundas nulas). Otros, como los métodos monótonos (`fmm`, `monoH.FC`), ajustan derivadas para evitar oscilaciones o preservar la forma local de los datos.

5.5.3 Ejemplo aplicado (resolución explícita)

Dados tres puntos:

$$(x_1, y_1) = (0.00,\ 100), \quad (x_2, y_2) = (0.04,\ 120), \quad (x_3, y_3) = (0.08,\ 110)$$

Queremos construir un spline cúbico entre x_1 y x_2 de la forma:

$$S(x) = a + b(x - x_1) + c(x - x_1)^2 + d(x - x_1)^3$$

Dado que $x_1 = 0$, la expresión se simplifica a:

$$S(x) = a + bx + cx^2 + dx^3$$

Paso 1: calcular a y b

Sabemos que:

$$a = y_1 = 100$$

$$b = m_1 = \frac{y_2 - y_1}{x_2 - x_1} = \frac{120 - 100}{0.04 - 0.00} = \frac{20}{0.04} = 500$$

Paso 2: calcular m_2

Estimamos la derivada final como pendiente secante entre x_2 y x_3:

$$m_2 = \frac{y_3 - y_2}{x_3 - x_2} = \frac{110 - 120}{0.08 - 0.04} = \frac{-10}{0.04} = -250$$

Paso 3: montar el sistema para c y d

Sea $h = x_2 - x_1 = 0.04$. Impongamos:

$$S(x_2) = a + bh + ch^2 + dh^3 = y_2$$
$$S'(x_2) = b + 2ch + 3dh^2 = m_2$$

Sustituyendo:

- $a = 100$, $b = 500$
- $h = 0.04$, $h^2 = 0.0016$, $h^3 = 0.000064$

Obtenemos:

$$c \cdot 0.0016 + d \cdot 0.000064 = 20 - 500 \cdot 0.04 = 0$$

$$2c \cdot 0.04 + 3d \cdot 0.0016 = -250 - 500 = -750$$

Paso 4: resolver el sistema

Primera ecuación:

$$0.0016c + 0.000064d = 0 \quad (1)$$

Segunda ecuación:

$$0.08c + 0.0048d = -750 \quad (2)$$

Multiplicamos (1) por 50:

$$0.08c + 0.0032d = 0$$

Restamos a (2):

$$(0.0048 - 0.0032)d = -750 \Rightarrow 0.0016d = -750 \Rightarrow d = -468750$$

Sustituimos en (1):

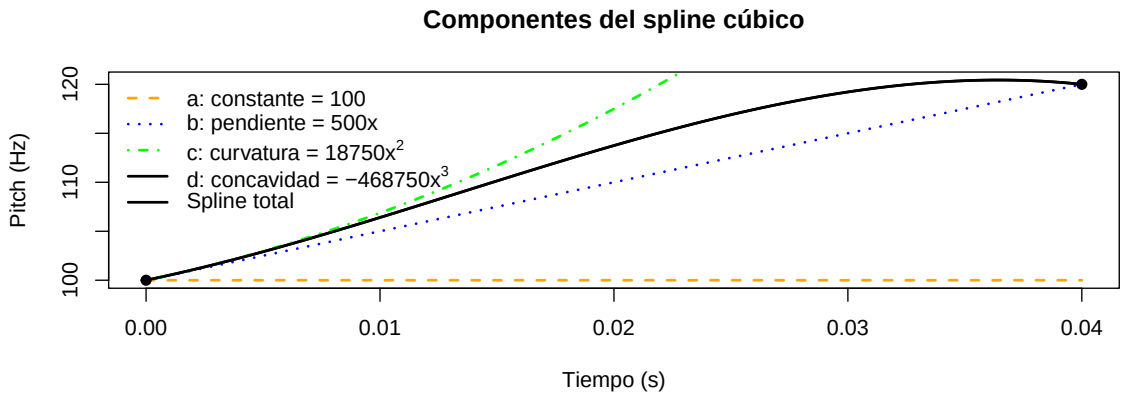


Figura 5.1: Construcción progresiva de un spline cúbico. La curva negra representa el spline total ajustado entre los puntos ((0.00, 100)) y ((0.04, 120)), mientras que las líneas de colores muestran la contribución sucesiva de cada término: naranja ((a), constante), azul ((b), pendiente), verde ((c), curvatura). La concavidad ((d)) se incorpora en la forma final. Cada componente se añade de forma acumulativa, permitiendo visualizar su efecto geométrico sobre la curva.

$$0.0016c + 0.000064 \cdot (-468750) = 0 \Rightarrow c = 18750$$

Resultado final:

$$S(x) = 100 + 500x + 18750x^2 - 468750x^3$$

Este polinomio representa el spline cúbico que interpola suavemente entre los puntos $(0.00,\ 100)$ y $(0.04,\ 120)$, usando derivadas estimadas en los extremos.

5.5.4 Métodos disponibles en R

La función `splinefun()` del paquete base `{stats}` permite seleccionar distintos algoritmos:

Método	Basado en…	Propósito principal
`"natural"`	Spline cúbico clásico	Derivada segunda = 0 en los extremos (bordes suaves)
`"fmm"`	Fritsch y Butland (1984)	Preservar la forma local, evitar overshoot
`"monoH.FC"`	Fritsch y Carlson (1980)	Asegurar monotonía estricta en los datos

Método	Basado en…	Propósito principal
`"periodic"`	Spline cúbico periódico clásico	Forzar continuidad cíclica entre extremos

Nota: tanto `fmm` como `monoH.FC` están diseñados para evitar overshoot, pero lo hacen con heurísticas distintas.

5.5.4.1 Método `fmm` en Parole

La función `splinefun()` del paquete base `{stats}` de R permite aplicar tanto *splines naturales* como el método `"fmm"`, según la opción seleccionada (R Core Team 2024).

Parole utiliza por defecto `splinefun(method = "fmm")` por las siguientes razones:

- Preserva la forma local de las curvas acústicas (ideal para voz hablada)
- Evita oscilaciones artificiales entre picos y valles
- No impone suavidad innecesaria en transiciones abruptas (como plosivas o pausas)
- Es robusto y estable incluso en tramos breves con pocos puntos

Este método se basa en el algoritmo de **Fritsch y Butland** (Fritsch y Butland 1984), que ajusta las derivadas de los puntos de control para garantizar que la curva interpolada **no exceda los valores extremos definidos por los datos** (*overshoot*), preservando la morfología original de las series.

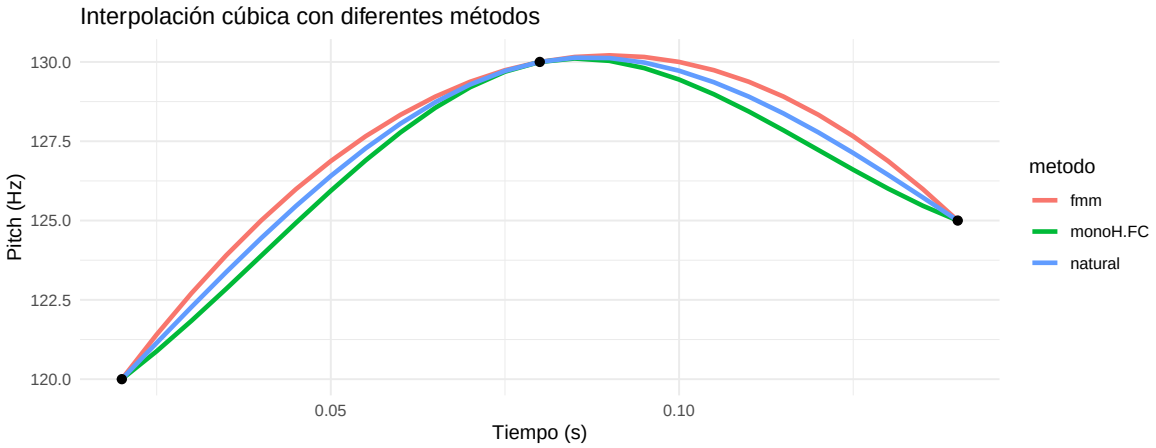
5.5.5 ¿Se puede cambiar el método?

Sí. Parole no impone un único método. El investigador puede experimentar con:

```
splinefun(x, y, method = "natural")
splinefun(x, y, method = "monoH.FC")
```

Cambiar el método puede ser útil si:

- Se analizan voces sostenidas o melódicas
- Se quiere imponer suavidad global
- Se comparan respuestas ante jitter, shimmer o vibrato



5.5.6 Ejemplo reproducible

A continuación se muestra un ejemplo aplicado con tres métodos distintos para interpolar una curva de pitch:

5.5.7 Reflexión final

El spline es una decisión técnica y conceptual: define cómo le damos forma a lo que no está medido, cómo estimamos sin imponer, y cómo cuidamos la fidelidad del fenómeno.

Parole no ofrece un solo camino, sino una estructura donde el investigador puede —y debe— elegir con conciencia.

5.6 Procesamiento de curvas prosódicas con `process_prosody.R`

Este script es el encargado de leer los archivos de salida generados por Praat (`.Pitch`, `.Formant`, `.Intensity`, etc.), interpolarlos y producir un archivo unificado con las curvas sincronizadas por tiempo, listo para su análisis o visualización posterior.

5.6.1 Uso

```
Rscript process_prosody.R <video_dir> <out_file>
```

- `<video_dir>`: carpeta que contiene los archivos `.Pitch`, `.Formant`, `.Intensity`, `.Harmonicity`, `.PointProcess` junto con un archivo `_frame_timestamps.csv` y opcionalmente `_silero_segments.csv`.
- `<out_file>`: nombre del archivo de salida. Puede ser `.csv` o `.parquet`.

5.6.2 Requisitos

- Solo una dependencia externa: el paquete `arrow` para guardar archivos en formato Parquet.
- Utiliza `parallel` de R base para hacer el parseo en paralelo (no requiere instalación).

5.6.3 Etapas principales del procesamiento

5.6.3.1 Lectura y parseo de archivos Praat

Se utilizan funciones propias para leer los siguientes objetos:

- `Pitch` → frecuencia, intensidad y fuerza de la señal fundamental
- `Intensity` → energía acústica por frame
- `Harmonicity` → relación armónica/ruido
- `Formants` → trayectorias F1, F2, … F5 con frecuencia y ancho de banda
- `PointProcess` → eventos glotales como vector de tiempos

5.6.3.2 Interpolación y sincronización

- Se cargan los timestamps del video (`_frame_timestamps.csv`).
- Se interpola cada curva usando `splinefun(..., method = "fmm")`, asegurando continuidad y estabilidad.
- Si hay archivo Silero (`_silero_segments.csv`), se marca la presencia o ausencia de voz por frame.

5.6.3.3 Exportación de datos

El resultado es una tabla `df_final` con una fila por frame y columnas como:

- `frame_timestamp`, `pitch`, `intensity`, `harmonicity`, `formant.1_frequency`, …

- `vad` (si hay VAD disponible)

Se guarda como `.csv` o `.parquet`, según lo indique la extensión de `<out_file>`.

Este procesamiento permite llevar los datos de salida de Praat a un formato limpio y alineado para su posterior análisis, visualización o modelado.

5.6.4 Composición final del dataset

La salida final del script `process_prosody.R` consiste en una tabla donde cada fila representa un frame del vídeo original, alineado con las curvas prosódicas extraídas e interpoladas. Entre las variables principales se encuentran el tiempo (`frame_timestamp`), la frecuencia fundamental (`pitch`), la intensidad acústica (`intensity`), la relación armónica-ruido (`harmonicity`), así como los primeros tres formantes estimados (`formant.1_frequency`, `formant.2_frequency`, `formant.3_frequency`). Además, si se dispone de información de detección de voz, se incluye una columna binaria (`vad`) que indica si en ese instante hay habla detectada.

Un fragmento representativo del resultado se presenta en la Tabla 5.2.

Tabla 5.2: Fragmento representativo del archivo de salida de Parole (primeras y últimas filas). Todas las medidas están expresadas en unidades estándar: pitch en Hz, intensity en dB, harmonicity en dB y formant.x_frequency en Hz. La columna vad indica presencia (TRUE) o ausencia (FALSE) de voz estimada.

frame_id	frame_timestamp	pitch	intensity	harmonicity	formant.1_frequency	formant.2_frequency	formant.3_frequency	vad
1	0.00	531.56	114.01	33.15	-6248.19	8645.15	10436.95	FALSE
2	0.03	315.49	53.12	3.65	1013.83	1459.94	3135.42	FALSE
3	0.07	318.72	54.53	0.89	788.02	1416.04	2958.90	FALSE
4	0.10	321.99	60.63	11.09	674.63	1340.93	3144.58	FALSE
5	0.13	319.32	60.21	14.15	904.80	1601.50	2907.50	FALSE
6	0.17	301.56	60.04	15.20	899.66	1832.93	2704.78	FALSE
128	4.24	247.65	59.18	12.21	890.07	1882.97	2821.72	TRUE
129	4.27	239.37	57.69	9.32	735.06	1850.60	2842.10	TRUE
130	4.30	220.53	57.62	12.85	935.33	1731.09	2864.56	TRUE
131	4.34	216.98	59.64	15.58	832.39	1738.21	2877.89	TRUE
132	4.37	213.51	67.30	11.10	937.25	1514.48	2654.88	TRUE

Con este procedimiento, Parole convierte los archivos acústicos crudos en un conjunto de curvas prosódicas alineadas y procesadas, preparadas para su análisis cuantitativo o visualización sincronizada con otros niveles del lenguaje.

A través de la extracción en Praat, la interpolación con splines cúbicos (fmm), y la exportación estructurada en tablas por frame, se garantiza un flujo de trabajo reproducible, interpretable y adaptable a distintos dominios lingüísticos o experimentales.

Bibliografía académica

Boersma, Paul, y David Weenink. 2025. *Praat: doing phonetics by computer*. Institute of Phonetic Sciences, University of Amsterdam.

Fritsch, F. N., y J. Butland. 1984. «A Method for Constructing Local Monotone Piecewise Cubic Interpolants». *SIAM Journal on Scientific and Statistical Computing* 5 (2): 300-304. https://doi.org/10.1137/0905021.

Fritsch, F. N., y R. E. Carlson. 1980. «Monotone Piecewise Cubic Interpolation». *SIAM Journal on Numerical Analysis* 17 (2): 238-46. https://doi.org/10.1137/0717021.

Ioannidis, John P. A. 2005. «Why Most Published Research Findings Are False». *PLoS Medicine* 2 (8): e124. https://doi.org/10.1371/journal.pmed.0020124.

Miyakawa, Tsuyoshi. 2020. «No raw data, no science: another possible source of the reproducibility crisis». *Molecular Brain* 13 (1). https://doi.org/10.1186/s13041-020-0552-2.

Nyquist, Harry. 1928. «Certain Topics in Telegraph Transmission Theory». *Transactions of the American Institute of Electrical Engineers* 47 (2): 617-44. https://doi.org/10.1109/T-AIEE.1928.5055024.

R Core Team. 2024. *splinefun: Interpolating Splines*. https://stat.ethz.ch/R-manual/R-devel/library/stats/html/splinefun.html.

Saussure, Ferdinand de. 2009. *Curso de lingüística general*. Traducido por Mauro Armiño. 7ª reimpresión. Madrid: Ediciones Akal.

Shannon, Claude E. 1949. «Communication in the Presence of Noise». *Proceedings of the IRE* 37 (1): 10-21. https://doi.org/10.1109/JRPROC.1949.232969.

Referencias técnicas

ffprobe ffprobe Documentation | FFmpeg Disponible en: https://ffmpeg.org/ffprobe.html

Silero VAD Silero Voice Activity Detector | PyTorch https://pytorch.org/hub/snakers4_silero-vad_vad/ silero-vad-fork | PyPI https://pypi.org/project/silero-vad-fork/