# Sistema de Verificación de Identidad por Voz (Biometría Vocal) Open Source

## Introducción

La **biometría vocal** o verificación de identidad por voz consiste en confirmar la identidad de una persona mediante las características únicas de su voz. A diferencia del reconocimiento de voz (entender lo que se dice), aquí el objetivo es reconocer **quién** habla, independientemente del contenido de la frase. Existen dos modalidades principales:  
- *Identificación de hablante*: determinar quién es un hablante desconocido comparándolo contra múltiples perfiles (responde a "¿quién está hablando?").  
- *Verificación de hablante*: confirmar si la voz de una persona coincide con la identidad que reclama (responde a "¿es esta voz de X?")[[1]](https://maelfabien.github.io/machinelearning/basics_speech/#:~:text=,according%20to%20the%20speaker%20identity)[[2]](https://maelfabien.github.io/machinelearning/basics_speech/#:~:text=,input%20stream%20into%20segments%20according). En autenticación se suele usar verificación (1:1 comparación) que es más sencilla que la identificación en conjunto grande[[3]](https://github.com/943274923/Speaker-Recognition#:~:text=Speaker%20recognition%20is%20the%20identification,little%20easier%20than%20speaker%20recognition).

Un sistema de verificación por voz típico se compone de un **front-end** de audio (captura, preprocesamiento y extracción de características) y un **back-end** biométrico (modelado de la voz y comparación contra modelos registrados)[[4]](https://maelfabien.github.io/machinelearning/basics_speech/#:~:text=The%20main%20steps%20when%20running,Verification%20pipeline%20are%20the%20following). En la etapa de *desarrollo* se entrena un modelo universal (*Universal Background Model*, UBM) o un extractor de embeddings con muchos datos de voz; luego en *enrolamiento* se registra la huella vocal del usuario (sea ajustando un modelo por usuario o almacenando su embedding característico); finalmente, en *verificación* se extraen características de una nueva muestra de voz y se comparan con el modelo/huella del hablante reclamado para decidir si aceptarla o rechazarla[[5]](https://maelfabien.github.io/machinelearning/basics_speech/#:~:text=%2A%20Development%3A%20learn%20speaker,This%20is%20the%20prediction%20part). Los sistemas pueden ser **textodependientes** (el usuario debe decir una frase clave fija, p. ej. *"Hola, soy Juan"*) o **textoindependientes** (cualquier discurso vale)[[6]](https://maelfabien.github.io/machinelearning/basics_speech/#:~:text=There%20are%202%20types%20of,speaker%20verification%20techniques). En aplicaciones prácticas (control de acceso, autenticación bancaria, etc.), un enfoque textoindependiente en español latino aumentaría la flexibilidad, aunque requiere más datos de voz para entrenar[[7]](https://maelfabien.github.io/machinelearning/basics_speech/#:~:text=%2A%20Text,the%20solution%20is%20more%20flexible). A continuación, se describen herramientas Python open source para construir dicho sistema en Linux, haciendo énfasis en el idioma español (latino, idealmente acento colombiano).

## Preprocesamiento de audio (Linux, Python y formatos)

El preprocesamiento asegura que las grabaciones de voz tengan calidad y formato consistente antes de extraer rasgos biométricos. Aspectos importantes incluyen:  
- **Captura y lectura de audio**: En Python se puede usar librerías como PyAudio (basada en PortAudio) para grabar desde micrófono en Linux (asegurando instalar las dependencias de PortAudio)[[8]](https://github.com/ppwwyyxx/speaker-recognition#:~:text=The%20Dockerfile%20can%20be%20used,started%20with%20the%20project%20easier). Alternativamente, sounddevice (Python) o herramientas CLI como arecord (ALSA) pueden usarse en Linux para obtener archivos WAV. Para leer/escribir archivos de audio, bibliotecas como scipy.io.wavfile o soundfile (pysoundfile) son útiles, soportando WAV PCM estándar.  
- **Formato uniforme**: Conviene trabajar con audio monofónico, frecuencia de muestreo estándar (por ejemplo 16 kHz) y resoluciones consistentes. Librerías como **Librosa** facilitan la carga de audio (con resampling) en arreglos NumPy[[9]](https://github.com/pyyush/VoiceID#:~:text=1.%20python%20,numpy). También se pueden usar bindings de **SoX** o FFmpeg (e.g. pydub) para convertir formatos, remover ruido de fondo y normalizar volúmenes, todas disponibles en Linux.  
- **Filtrado y segmentación**: Puede aplicarse detección de voz (VAD) para recortar silencios o ruido al inicio/final de las muestras. Existen implementaciones open source como webrtcvad (basada en WebRTC) o algoritmos de umbral energético. Por ejemplo, un proyecto de referencia empleó VAD mediante *Long-Term Spectral Divergence* para segmentar voz útil[[10]](https://github.com/ppwwyyxx/speaker-recognition#:~:text=Voice%20Activity%20Detection).  
- **Consideraciones Linux**: Todas las herramientas mencionadas son compatibles con Linux. De hecho, muchos proyectos de reconocimiento de hablantes se desarrollan y prueban principalmente en Linux[[11]](https://projets-lium.univ-lemans.fr/sidekit/overview/index.html#:~:text=SIDEKIT%20aims%20at%20providing%20the,for%20both%20Linux%20and%20MacOS). Es importante instalar los paquetes de sistema necesarios (por ejemplo, libportaudio para PyAudio, libsndfile para librosa/soundfile). Por lo general, las bibliotecas Python son multiplataforma y se instalan vía pip en Linux sin inconvenientes. Algunos proyectos proporcionan incluso contenedores Docker o scripts de instalación para facilitar la configuración en entornos Linux[[12]](https://github.com/ppwwyyxx/speaker-recognition#:~:text=Dependencies).

## Extracción de características de voz (features)

Una vez obtenido el audio limpio, se extraen características representativas de la voz del hablante. Estas *features* capturan propiedades del tracto vocal y forman la entrada del modelo biométrico. Las más comunes y efectivas en verificación de hablante son:

* **Coeficientes Cepstrales de Frecuencias Mel (MFCC)**: Es el estándar industrial para caracterizar espectro de voz. Los MFCC comprimen la información espectral enfatizando rasgos audibles relevantes para distinguir hablantes[[13]](https://github.com/ppwwyyxx/speaker-recognition#:~:text=Feature%3A). Python ofrece varias rutas open source para computar MFCC: la biblioteca **Librosa** puede calcular fácilmente espectrogramas Mel y MFCC; también python\_speech\_features (open source) ofrece funciones de MFCC y filtros de ventana; alternativamente, toolkits como **Kaldi** (C++/bash, con wrappers Python) tienen implementaciones altamente optimizadas de MFCC. En Python puro, librosa es popular por su simplicidad y uso extensivo en proyectos académicos[[9]](https://github.com/pyyush/VoiceID#:~:text=1.%20python%20,numpy).
* **Espectrogramas y otras características**: Un modelo moderno puede alimentarse del espectrograma Mel completo o del waveform crudo. Sin embargo, para un sistema tradicional, además de MFCC se consideran a veces **LPC** (coeficientes predictivos lineales) que modelan el tracto vocal, o **PLP** (Perceptual Linear Prediction). Por ejemplo, un proyecto open source clásico utilizó MFCC y LPC conjuntamente como features[[13]](https://github.com/ppwwyyxx/speaker-recognition#:~:text=Feature%3A). También es útil calcular deltas y delta-deltas (derivadas temporales de MFCC) para incorporar dinámica temporal.
* **Normalización**: Es importante normalizar las características (p. ej. restar la media y dividir por la desviación típica por coeficiente) antes de alimentar modelos, para compensar distintas condiciones de grabación. Técnicas como cepstral mean normalization (CMN) pueden aplicarse a las cepstras.
* **Extracción con herramientas de deep learning**: Alternativamente, frameworks como **Torchaudio** (de PyTorch) o **TensorFlow Audio** pueden extraer features en pipelines de deep learning. Por ejemplo, **SpeechBrain** utiliza internamente torchaudio para leer audio y normalizarlo a 16 kHz[[14]](https://huggingface.co/speechbrain/spkrec-xvect-voxceleb#:~:text=import%20torchaudio%20from%20speechbrain,encode_batch%28signal). Estas bibliotecas también permiten calcular filtros Mel o incluso aprender representaciones de audio crudo con modelos especiales (e.g. redes CNN sobre waveform). En un enfoque end-to-end, la extracción de features puede integrarse en la red neuronal (p. ej. usando capas de espectrograma dentro de la arquitectura del modelo).

En resumen, para un sistema open source en Python, se recomiendan **librosa** (lectura de WAV, MFCC), **NumPy/SciPy** (manipulación de señales), y eventualmente **OpenSMILE** (herramienta C++ ejecutable, de código abierto) si se quisieran extraer grandes conjuntos de características acústicas de forma modular. No obstante, para verificación de hablante en español, unos 20 coeficientes MFCC por trama con ventanas de ~20-30ms suelen ser suficientes como entrada al modelo[[13]](https://github.com/ppwwyyxx/speaker-recognition#:~:text=Feature%3A).

## Modelado biométrico del hablante (speaker recognition)

El corazón del sistema es el modelo que **distingue voces**. Existen enfoques clásicos basados en modelos estadísticos y enfoques modernos con redes neuronales profundas:

* **Modelos GMM-UBM e I-Vectores (enfoque clásico)**: Durante mucho tiempo, la verificación de locutor se basó en *Modelos de Mezcla de Gaussianas* (GMM) adaptados. Se entrena primero un **Modelo Universal de Fondo (UBM)** que representa la distribución genérica de características de voz en una población amplia[[5]](https://maelfabien.github.io/machinelearning/basics_speech/#:~:text=%2A%20Development%3A%20learn%20speaker,This%20is%20the%20prediction%20part). Posteriormente, para cada usuario inscrito, se adapta ligeramente el UBM a sus datos de voz para obtener un GMM específico del hablante. La decisión de verificación se toma comparando la probabilidad de que la nueva muestra provenga del modelo del hablante vs. del UBM (razón de verosimilitud)[[15]](https://maelfabien.github.io/machinelearning/basics_speech/#:~:text=The%20decision%20process%20in%20classic,from%20the%20Universal%20Background%20model). Este método probabilístico fue mejorado con la introducción de **i-vectores**: en lugar de comparar modelos GMM completos, se proyecta cada locutor a un vector de baja dimensión que captura su identidad vocal (i-vector) usando Análisis de Factoría (*Joint Factor Analysis*, JFA)[[16]](https://github.com/ppwwyyxx/speaker-recognition#:~:text=,JFA). Luego se compara i-vector de prueba contra i-vector del usuario (por ejemplo con una métrica de similitud o un clasificador PLDA). Herramientas open source como **Kaldi** implementan pipelines completos de i-vector+PLDA, y librerías Python como **SideKit** también soportan i-vectores y PLDA. SideKit, por ejemplo, provee toda la cadena desde VAD, MFCC, i-vector hasta evaluación de rendimiento, siendo un toolkit educativo eficiente para reconocimiento de hablantes[[17]](https://github.com/943274923/Speaker-Recognition#:~:text=SideKit%201). Estos métodos funcionan con español sin problema siempre que el UBM o i-vector extractor se entrene con voces en español. Se podría utilizar, por ejemplo, un corpus amplio de español latino para entrenar un UBM, o incluso adaptar un modelo UBM existente (entrenado en otro idioma) con datos en español para mejorar su sensibilidad a las características del habla hispana.
* **Embeddings neuronales (x-vectors, d-vectors, ECAPA-TDNN)**: Los avances recientes emplean **redes neuronales profundas** para extraer directamente una representación vectorial de la voz de cada persona, optimizada para separabilidad entre hablantes. Un ejemplo pionero son los **x-vectors**, obtenidos de una red tipo DNN/TDNN entrenada para clasificar cientos de speakers; la capa oculta penúltima produce un vector fijo (embedding) que caracteriza la voz[[18]](https://huggingface.co/speechbrain/spkrec-xvect-voxceleb#:~:text=Speaker%20Verification%20with%20xvector%20embeddings,on%20Voxceleb)[[14]](https://huggingface.co/speechbrain/spkrec-xvect-voxceleb#:~:text=import%20torchaudio%20from%20speechbrain,encode_batch%28signal). Para verificación, se compara el embedding de prueba con el del usuario (usando distancia coseno, similitud de Pearson o un modelo discriminante). SpeechBrain ofrece un modelo preentrenado de x-vectors entrenado en VoxCeleb1+2 que logra ~3.2% EER en inglés[[19]](https://huggingface.co/speechbrain/spkrec-xvect-voxceleb#:~:text=This%20repository%20provides%20all%20the,Voxceleb%201%2B%20Voxceleb2%20training%20data). Otro modelo muy efectivo es **ECAPA-TDNN**, una arquitectura refinada que mejora la discriminación; existen implementaciones abiertas (por ejemplo en **SpeechBrain** o **NVIDIA NeMo**). NeMo de NVIDIA proporciona modelos preentrenados *SpeakerNet, TitaNet-L* y *ECAPA-TDNN* disponibles para descarga, los cuales fueron entrenados en datasets multilingües (e.g. VoxCeleb) y funcionan de forma *independiente del idioma*[[20]](https://github.com/Rumeysakeskin/Speaker-Verification#:~:text=Speaker%20verification%20is%20verifying%20the,from%20language%20via%20NVIDIA%20NeMo). Estos modelos producen embeddings de alta calidad; la comparación suele hacerse vía similitud de coseno, y con un umbral adecuado se decide la aceptación[[21]](https://github.com/Rumeysakeskin/Speaker-Verification#:~:text=The%20cosine%20similarity%20metric%20was,used%20for%20prediction). Un repositorio muestra cómo usar los modelos de NeMo para extraer vectores de voz y compararlos con sklearn coseno fácilmente[[20]](https://github.com/Rumeysakeskin/Speaker-Verification#:~:text=Speaker%20verification%20is%20verifying%20the,from%20language%20via%20NVIDIA%20NeMo)[[21]](https://github.com/Rumeysakeskin/Speaker-Verification#:~:text=The%20cosine%20similarity%20metric%20was,used%20for%20prediction). De forma similar, la librería **Resemblyzer** expone un encoder preentrenado (basado en modelo de **d-vector** de Google) que convierte un clip de voz en un embedding de 256 dimensiones; es extremadamente práctico porque permite realizar verificación de hablante *sin* tener que entrenar un modelo neuronal desde cero[[22]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=Conclusion). De hecho, un ejemplo con Resemblyzer demostró reconocer a un hablante español (clip de Cristiano Ronaldo) comparando su embedding contra una base de datos vectorial de voces conocidas, logrando identificar correctamente la voz con solo unas pocas líneas de código[[23]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=,Siuuuuuuuuu%21%E2%80%9D)[[24]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=Top%20two%20speakers%3A%20). En resumen, los métodos de embeddings neuronales son la corriente principal hoy en día debido a su precisión superior en condiciones variables.

**Herramientas y librerías Python recomendadas** para la etapa de modelado biométrico incluyen:

* **SpeechBrain** (PyTorch): toolkit de código abierto para speech AI (ASR, audio, NLP). Incluye recetas para *speaker verification* y modelos preentrenados (x-vector, ECAPA). Por ejemplo, SpeechBrain permite cargar en una línea un modelo x-vector preentrenado en VoxCeleb y obtener embeddings de audio[[14]](https://huggingface.co/speechbrain/spkrec-xvect-voxceleb#:~:text=import%20torchaudio%20from%20speechbrain,encode_batch%28signal). Asimismo, brinda scripts para entrenamiento desde cero (recetas VoxCeleb) que se pueden adaptar a datos en español[[25]](https://huggingface.co/speechbrain/spkrec-xvect-voxceleb#:~:text=3). Licencia Apache 2.0.
* **SideKit**: paquete Python open source enfocado en reconocimiento/verificación de locutor (desarrollado por Larcher et al.). Soporta i-vectors, x-vectors y tiene componentes para todo el pipeline (VAD, MFCC, scoring). Se probó en Linux con Python3[[11]](https://projets-lium.univ-lemans.fr/sidekit/overview/index.html#:~:text=SIDEKIT%20aims%20at%20providing%20the,for%20both%20Linux%20and%20MacOS) y ofrece un entorno educativo para investigar algoritmos clásicos y DNN. No provee modelos preentrenados para español, pero permite entrenar fácilmente con un corpus propio[[17]](https://github.com/943274923/Speaker-Recognition#:~:text=SideKit%201).
* **Pyannote.audio**: librería Python (PyTorch) especializada en diarización, pero proporciona embeddings de speaker preentrenados que pueden aprovecharse para reconocimiento. Por ejemplo, pyannote.audio tiene modelos para extracción de *speaker embeddings* entrenados en VoxCeleb, útiles para comparar voces. El foco es diarización, pero la comunidad la usa también para tareas de verificación al comparar embeddings con métricas adecuadas.
* **NVIDIA NeMo**: aunque es un toolkit más amplio, su módulo de speaker recognition facilita el uso de modelos SOTA entrenados por NVIDIA (requiere PyTorch). Es open source (licencia Apache) y está optimizado para GPU. En Linux, NeMo puede instalarse via pip, y los modelos (ECAPA, etc.) se descargan desde NGC (NVIDIA GPU Cloud) o GitHub. Un beneficio es que estos modelos fueron entrenados con datos de varios idiomas, por lo que generalizan bien a voces en español[[20]](https://github.com/Rumeysakeskin/Speaker-Verification#:~:text=Speaker%20verification%20is%20verifying%20the,from%20language%20via%20NVIDIA%20NeMo).
* **Resemblyzer**: librería Python ligera para obtener embeddings de voz de forma inmediata[[22]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=Conclusion). Ideal para prototipos o aplicaciones con recursos limitados, ya que evita entrenar redes complejas. Con una simple instalación (pip install resemblyzer), permite codificar cualquier audio .wav en un vector; luego, la comparación de vectores (p. ej. distancia coseno) indica la similitud de hablantes. Es agnóstica al idioma (el modelo preentrenado fue entrenado en multitud de voces de distintos datos, incl. LibriSpeech, VoxCeleb, etc.), y funciona en Linux sin pasos adicionales. Un artículo demuestra su facilidad: *"Resemblyzer es la forma más sencilla de procesar audio y codificarlo en embeddings, sin necesidad de diseñar una red neuronal propia"*[[22]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=Conclusion).
* **Otros repositorios relevantes**: Existe una variedad de proyectos open source enfocados en biometría vocal: por ejemplo, **ppwwyyxx/speaker-recognition**[[26]](https://github.com/ppwwyyxx/speaker-recognition#:~:text=Model%3A) es un sistema completo (GMM-UBM, JFA, GUI básica) bajo licencia Apache 2.0 con ~677 estrellas, útil como referencia académica (aunque usa Python 2); **3D-Speaker**[[27]](https://github.com/modelscope/3D-Speaker#:~:text=3D,research%20of%20speech%20representation%20disentanglement) es un toolkit moderno que incorpora también verificación multimodal, con modelos avanzados (ResNet, ERes2Net, etc.) y un corpus propio (orientado a investigación, requiere GPU); **VoiceID** y **VoiceprintRecognition** son librerías Python más sencillas que integran motores como TensorFlow/Keras para voz, pero suelen depender de servicios externos o están menos enfocadas al español. En general, para un proyecto robusto en español se recomienda utilizar frameworks activos (SpeechBrain, PyTorch, etc.) o librerías especializadas (SideKit, Resemblyzer) y aprovechar modelos existentes como punto de partida.

## Modelos preentrenados y entrenamiento para español

Para el idioma español latino (acento colombiano, por ejemplo), es importante aprovechar o crear modelos que contemplen las características del habla local. Aunque la **verificación de hablante es en gran medida independiente del idioma** (un modelo entrenado en inglés puede reconocer voces españolas porque las huellas vocales son principalmente fisiológicas)[[20]](https://github.com/Rumeysakeskin/Speaker-Verification#:~:text=Speaker%20verification%20is%20verifying%20the,from%20language%20via%20NVIDIA%20NeMo), sí puede haber ligeras mejoras al incluir datos del idioma objetivo debido a variaciones de pronunciación y acento. A continuación se detalla cómo proceder:

* **Modelos preentrenados compatibles con español**: Varios modelos abiertos entrenados en grandes datasets internacionales están disponibles. Por ejemplo, SpeechBrain provee un modelo *x-vector* entrenado en VoxCeleb (inglés) listo para usar, que se ha utilizado exitosamente en otros idiomas[[19]](https://huggingface.co/speechbrain/spkrec-xvect-voxceleb#:~:text=This%20repository%20provides%20all%20the,Voxceleb%201%2B%20Voxceleb2%20training%20data). NVIDIA ofrece su modelo **ECAPA-TDNN** preentrenado que es uno de los estados del arte; dado que VoxCeleb incluye hablantes de múltiples orígenes, este modelo funciona bien con voces en español. La librería Resemblyzer, como se mencionó, usa un encoder preentrenado en miles de voces (a partir de datos públicos diversos) y ha demostrado funcionar con muestras en español (ejemplo: reconoció correctamente a un orador en español europeo usando su famosa frase)[[23]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=,Siuuuuuuuuu%21%E2%80%9D)[[24]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=Top%20two%20speakers%3A%20). En resumen, **se puede comenzar utilizando estos modelos tal cual**, evaluando su rendimiento con voces locales. Es esperable que funcionen razonablemente, ya que la biometría vocal trasciende el idioma.
* **Adaptación o fine-tuning**: Si se dispone de un conjunto de voz en español (por ejemplo, grabaciones de usuarios o corpus públicos), se puede *afinar* un modelo preentrenado. SpeechBrain facilita esto: uno puede cargar su modelo x-vector preentrenado y continuar el entrenamiento con datos españoles (ajustando el learning rate bajo) para especializarlo. Esto podría mejorar la discriminación de matices propios del español colombiano (tonos, dialecto). Alternativamente, con NeMo se puede hacer fine-tuning de ECAPA introduciendo audio en español y reentrenando algunas capas. Esta adaptación es opcional; para muchos casos, la performance de los modelos generales ya será adecuada.
* **Entrenamiento desde cero**: Si se busca un enfoque 100% a medida en español, es posible entrenar un modelo desde cero con corpus de habla española. Se necesitará un dataset con múltiples hablantes y suficientes muestras por hablante. Opciones de **corpus abiertos en español**:
* *Mozilla Common Voice (español)*: colección libre con miles de hablantes leyendo texto en español. Si bien su propósito es ASR, contiene muchas voces únicas con suficiente audio por persona para entrenamiento.
* *VoxCeleb-ESP*: un esfuerzo reciente para compilar un VoxCeleb en idioma español (castellano) con 160 celebridades de España, con ~7 horas de audio en condiciones diversas[[28]](https://arxiv.org/html/2401.09441v1#:~:text=This%20paper%20presents%20VoxCeleb,identification%20results%20suggest%20that%20the)[[29]](https://arxiv.org/html/2401.09441v1#:~:text=obtained%20the%20audio%20from%20YouTube,and%20a%20balanced%20gender%20distribution). Esta iniciativa sugiere que datasets españoles comparables a VoxCeleb están en desarrollo.
* *Corpus Ahumada*: corpus español enfocado en variabilidad controlada (entornos simulados)[[30]](https://arxiv.org/html/2401.09441v1#:~:text=European%20Portuguese%20datasets%20,on%20biometrics%20and%20secure%20applications).
* *SecuVoice*: base de datos de habla en español con dígitos grabados en smartphones, orientada a biometría y seguridad[[31]](https://arxiv.org/html/2401.09441v1#:~:text=12%20,on%20biometrics%20and%20secure%20applications).
* *VoxForge Spanish*: recopilación de voces voluntarias en español (aunque más pequeña).
* *Grabar voces localmente*: para un sistema corporativo o específico, se pueden grabar decenas de personas colombianas diciendo frases libres durante 1-2 minutos cada uno, obteniendo así un conjunto personalizado.

Con estos datos, se entrenaría el modelo siguiendo las recetas establecidas. Por ejemplo, SpeechBrain tiene una receta para VoxCeleb; adaptarla implicaría preparar los archivos de audio y sus etiquetas de hablante en formato adecuado y ejecutar el script de entrenamiento de embeddings[[25]](https://huggingface.co/speechbrain/spkrec-xvect-voxceleb#:~:text=3). Kaldi igualmente tiene una *recipe* para VoxCeleb (x-vector) que podría reutilizarse con un nuevo dataset. En Python puro, SideKit permite armar fácilmente el pipeline: extraer MFCC de cada audio (p. ej. con extract\_features.py[[32]](https://github.com/943274923/Speaker-Recognition#:~:text=data_preprocessing)), entrenar un UBM (UBM.py), derivar i-vectores (i-vector.py), etc., aunque para entrenamiento profundo probablemente se integraría con PyTorch.

* **Evaluación en español**: Al usar o entrenar modelos, es vital evaluarlos con datos de prueba en español para obtener métricas de rendimiento reales en ese idioma. Métricas comunes: EER (Equal Error Rate), minDCF, etc. Un modelo con bajo EER (<5%) en VoxCeleb podría tener un EER algo mayor en español si el audio tiene diferente perfil, pero en general los modelos SOTA reportan EER muy bajos (p.ej. SideKit reporta ~1.2% EER con un modelo ResNet34 en VoxCeleb[[33][34]](https://github.com/deep-privacy/sidekit#:~:text=,privacy%2Fsidekit%2Freleases%2Fdownload%2Fsidekit_v0.1%2F%20best_halp_clr_adam_aam0.2_30_b256_vox12.pt_epoch71)). Los resultados en español podrían evaluarse con porciones del corpus (si se dispone de ground truth de quién habla en cada clip). Si el rendimiento no es suficiente, considerar técnicas como aumentar los datos (data augmentation: añadir ruido, reverberación a las muestras de entrenamiento), o usar enfoques textodependientes (más confiables si el usuario pronuncia una frase fija definida en español). Sin embargo, en muchos casos un sistema textoindependiente bien entrenado será capaz de reconocer al hablante con alta precisión aun cambiando de idioma[[20]](https://github.com/Rumeysakeskin/Speaker-Verification#:~:text=Speaker%20verification%20is%20verifying%20the,from%20language%20via%20NVIDIA%20NeMo).

## Ejemplos de flujo de trabajo y arquitectura del sistema

A modo de resumen, se describe un **flujo típico** de un sistema de autenticación por voz open source:

1. **Enrolamiento (registro)**: El usuario provee varias muestras de voz en español (por ejemplo, hablando durante 1 minuto sobre cualquier tema o leyendo oraciones). Estas muestras se procesan: se aplica VAD para quitar silencios, se extraen características MFCC frame a frame, y luego se obtiene una representación compacta de la voz del usuario. Dependiendo del método escogido, esto podría ser:
2. Calcular y almacenar un **modelo GMM** adaptado al usuario a partir del UBM (text-independent).
3. Obtener un **embedding promedio**: si usamos un modelo de *speaker embedding* preentrenado, podemos pasar cada locución por el encoder y promediar los embeddings resultantes para obtener el vector representativo de la voz del usuario[[35]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=To%20recognize%20the%20user%20with,Ronaldo%E2%80%99s%20voice%20in%20the%20database)[[36]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=Convert%20the%20new%20voice%20into,embeddings). (También es válido guardar múltiples embeddings por usuario).
4. Almacenar la huella vocal: El resultado (modelo o vector) se guarda en la base de datos del sistema, asociado a la identidad del usuario. Si son vectores, pueden almacenarse en un archivo JSON/NumPy, una base de datos SQL normal, o incluso en un **vector database** optimizado para búsquedas de similitud como Qdrant, Milvus, FAISS, etc., si se planifica tener muchos usuarios y requerir identificación rápida[[37]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=QdrantDB%3A%20An%20Overview)[[38]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=Speaker%20Recognition%3A). Para simples verificaciones 1:1, con guardar el vector en una tabla es suficiente.
5. **Verificación (autenticación)**: Cuando el usuario solicita acceso mediante su voz, el sistema realiza estos pasos en tiempo real:
6. Captura audio desde el micrófono (por ejemplo usando PyAudio en Linux) durante la frase de autenticación. Conviene que la frase dure al menos ~3-5 segundos para obtener suficientes características (en textodependiente, la frase sería fija, en textoindependiente el usuario puede hablar libremente dentro de un tiempo).
7. Preprocesa el audio capturado: filtrado de ruido, recorte de silencios iniciales, remuestreo a la tasa estándar (16 kHz). Esto garantiza que esté en las mismas condiciones que las muestras de entrenamiento.
8. **Extracción de características** de la muestra: Se calculan los MFCC u otras features. Con esas features, el sistema produce una representación de la voz: por ejemplo, infiere el embedding con la red neuronal entrenada (pasando las características por la red o directamente el audio por el modelo end-to-end)[[14]](https://huggingface.co/speechbrain/spkrec-xvect-voxceleb#:~:text=import%20torchaudio%20from%20speechbrain,encode_batch%28signal).
9. **Comparación con el modelo registrado**: Aquí ocurre la verificación propiamente dicha. Si el enfoque es GMM-UBM, se calculará la verosimilitud de que la nueva voz pertenezca al modelo del usuario vs al UBM, derivando un puntaje de semejanza[[15]](https://maelfabien.github.io/machinelearning/basics_speech/#:~:text=The%20decision%20process%20in%20classic,from%20the%20Universal%20Background%20model). Si es por embeddings, se calcula la distancia o similitud entre el vector de la muestra y el vector guardado del usuario (por ejemplo, usando **distancia coseno** normalizada)[[21]](https://github.com/Rumeysakeskin/Speaker-Verification#:~:text=The%20cosine%20similarity%20metric%20was,used%20for%20prediction). Este puntaje se compara contra un **umbral** previamente establecido (p. ej., threshold calibrado para lograr cierto nivel de FAR/FRR aceptable).
10. **Decisión**: Si la similitud excede el umbral (es decir, la voz suena lo suficientemente cercana a la del usuario registrado), entonces el sistema acepta la identidad y permite el acceso. Si no, la deniega. Opcionalmente, si el sistema es de identificación entre varios usuarios, se podría elegir la identidad cuyo vector esté más próximo (vecino más cercano) a la muestra dada[[39]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=To%20get%20top%202%20similar,mode%20in%20top%203%20case)[[40]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=,len%28speakers%29%3A%20top_two_names.append%28speakers%5Baligned_id). En nuestro caso de verificación 1:1, solo se verifica la identidad reclamada.

Todo este flujo se puede implementar con herramientas open source mencionadas. Por ejemplo, usando **Resemblyzer** el pseudocódigo sería: cargar encoder, obtener emb\_ref = encoder.embed\_utterance(audio\_enrolado) y guardar emb\_ref. Luego en verificación: emb\_test = encoder.embed\_utterance(audio\_nuevo) y calcular sim = cosine\_similarity(emb\_test, emb\_ref)[[21]](https://github.com/Rumeysakeskin/Speaker-Verification#:~:text=The%20cosine%20similarity%20metric%20was,used%20for%20prediction); decidir autenticación si sim supera X. De hecho, un tutorial mostró una variante con DB de vectores donde consultan el embedding nuevo en Qdrant para encontrar la voz más similar, logrando identificar correctamente al hablante con puntajes de similitud ~0.7 (70%) en clips cortos[[41]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=%5BScoredPoint%28id%3D2%2C%20version%3D0%2C%20score%3D0.6956655%2C%20payload%3D,vector%3DNone)[[42]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=By%20the%20above%20result%20you,data%20and%20try%20this%20out). En sistemas GMM/i-vector, el flujo sería similar pero usando las funciones de la librería (por ej., con SideKit: extraer i-vector de muestra y comparar con i-vector de inscripción vía scoring PLDA).

1. **Retroalimentación y actualización**: Algunos sistemas refinan el modelo del usuario con el tiempo. Por ejemplo, después de autenticaciones exitosas, podrían incorporarse esas muestras para recalcular el vector medio o actualizar el modelo GMM del usuario, mejorando con más datos. Dado que trabajamos en un entorno open source, es posible implementar esa actualización continua.

En términos de **arquitectura de software**, una implementación open source típica en Linux podría organizarse en módulos: un **módulo de adquisición de audio** (interfaz con el micro), un **módulo de procesamiento de señal** (limpieza, VAD, MFCC), un **módulo de embeddings/modelos** (que carga el modelo preentrenado o parámetros necesarios), y un **módulo de decisión** (que realiza la comparación y aplica la lógica de autenticación). Estos módulos pueden integrarse en un solo servicio. Por ejemplo, el proyecto VoiceSens integró todo en una aplicación web Python, con opciones de "enroll" y "authenticate" en la interfaz[[43]](https://github.com/bedangSen/VoiceSens#:~:text=Image%3A%20source%3A%20imgur)[[44]](https://github.com/bedangSen/VoiceSens#:~:text=1,Authenticate%20an%20existing%20user). Aunque VoiceSens utilizaba IBM Watson para transcripción, su componente de modelado usaba GMM para generar la huella de voz de cada usuario[[45]](https://github.com/bedangSen/VoiceSens#:~:text=To%20Do). En nuestro caso, podemos mantener todo local: la lógica Python puede exponerse vía una API REST (Flask/FastAPI) o una app de escritorio, asegurando que toda la biometría se procese en el servidor Linux sin llamadas externas.

## Compatibilidad con Linux y consideraciones finales

Todos los recursos mencionados son **software de código abierto ejecutable en Linux**. Python es intrínsecamente multiplataforma, y librerías como librosa, PyTorch, SciPy, etc., funcionan sin problemas en sistemas Linux (Ubuntu, Debian, etc.). De hecho, muchas de estas herramientas fueron desarrolladas con Linux en mente, dada su popularidad en entornos de investigación y servidores. Por ejemplo, SIDEKIT explicitamente fue probado en Linux[[11]](https://projets-lium.univ-lemans.fr/sidekit/overview/index.html#:~:text=SIDEKIT%20aims%20at%20providing%20the,for%20both%20Linux%20and%20MacOS) y proyectos como el de Yuxin Wu proporcionan un Dockerfile para entorno Linux[[12]](https://github.com/ppwwyyxx/speaker-recognition#:~:text=Dependencies). Para asegurar compatibilidad:

* Utilice distribuciones comunes (Ubuntu 20.04+, etc.) e instale dependencias de audio (p. ej. sudo apt-get install portaudio19-dev antes de instalar PyAudio).
* Compruebe la disponibilidad de drivers de audio (ALSA/PulseAudio) si va a capturar del micrófono. En servidores sin entorno sonoro, se pueden usar archivos de audio pregrabados para las pruebas.
* Si emplea GPU para acelerar el modelado (opcional con PyTorch), en Linux deberá instalar los controladores CUDA apropiados. Herramientas como SpeechBrain y NeMo detectan automáticamente la GPU (device=\"cuda\")[[46]](https://github.com/deep-privacy/sidekit#:~:text=import%20torchaudio%20from%20sidekit,Xtractor)[[47]](https://github.com/deep-privacy/sidekit#:~:text=model_opts%5B,embedding_size), pero también funcionan en CPU (con menor velocidad).
* Todos los repositorios listados (SpeechBrain, SideKit, etc.) son de licencia permisiva (Apache 2.0, MIT o BSD) y se pueden integrar libremente en proyectos. Asegúrese de revisar las licencias de los *datasets* de entrenamiento si entrena con datos externos (por ejemplo, Common Voice es CC0, VoxCeleb requiere solicitud pero es gratuito para investigación).
* Documentación y soporte: Al ser open source, es útil apoyarse en las comunidades (foros de GitHub, documentación oficial). Por ejemplo, SpeechBrain tiene tutoriales en su documentación[[48]](https://speechbrain.readthedocs.io/en/latest/tutorials/basics/what-can-i-do-with-speechbrain.html#:~:text=What%20can%20I%20do%20with,), y SideKit ofrece un manual PDF detallado[[17]](https://github.com/943274923/Speaker-Recognition#:~:text=SideKit%201). No dude en consultar estas fuentes durante el desarrollo.

En conclusión, es **viable construir un sistema completo de verificación de identidad por voz en español latino usando solo componentes open source**. Combinando librerías de procesamiento de audio (librosa, PyAudio), extracción de características (MFCC con librosa o similares) y potentes modelos biométricos (ya sea i-vector/PLDA vía SideKit o embeddings neuronales vía SpeechBrain/Resemblyzer), se puede lograr una solución de autenticación por voz funcionando enteramente en Linux. Todos los bloques necesarios están disponibles como código abierto, y existen ejemplos y repositorios que sirven de guía para implementaciones típicas. Con los ajustes adecuados para el español (datos de entrenamiento locales, umbrales calibrados en población hispanohablante), el sistema resultante podrá autenticar usuarios de forma segura y eficiente mediante su voz, incluso con acento colombiano.

**Referencias**: Herramientas y conceptos citados se basan en fuentes abiertas y documentación relevante, por ejemplo SpeechBrain[[14]](https://huggingface.co/speechbrain/spkrec-xvect-voxceleb#:~:text=import%20torchaudio%20from%20speechbrain,encode_batch%28signal)[[25]](https://huggingface.co/speechbrain/spkrec-xvect-voxceleb#:~:text=3), proyectos de reconocimiento de hablante clásicos[[13]](https://github.com/ppwwyyxx/speaker-recognition#:~:text=Feature%3A)[[16]](https://github.com/ppwwyyxx/speaker-recognition#:~:text=,JFA), y artículos que demuestran la aplicación de embeddings preentrenados en casos de uso prácticos[[23]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=,Siuuuuuuuuu%21%E2%80%9D)[[22]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=Conclusion), entre otros. Estas referencias respaldan las recomendaciones técnicas y aseguran que el enfoque propuesto esté alineado con las mejores prácticas actuales en biometría vocal open source.

[[1]](https://maelfabien.github.io/machinelearning/basics_speech/#:~:text=,according%20to%20the%20speaker%20identity) [[2]](https://maelfabien.github.io/machinelearning/basics_speech/#:~:text=,input%20stream%20into%20segments%20according) [[4]](https://maelfabien.github.io/machinelearning/basics_speech/#:~:text=The%20main%20steps%20when%20running,Verification%20pipeline%20are%20the%20following) [[5]](https://maelfabien.github.io/machinelearning/basics_speech/#:~:text=%2A%20Development%3A%20learn%20speaker,This%20is%20the%20prediction%20part) [[6]](https://maelfabien.github.io/machinelearning/basics_speech/#:~:text=There%20are%202%20types%20of,speaker%20verification%20techniques) [[7]](https://maelfabien.github.io/machinelearning/basics_speech/#:~:text=%2A%20Text,the%20solution%20is%20more%20flexible) [[15]](https://maelfabien.github.io/machinelearning/basics_speech/#:~:text=The%20decision%20process%20in%20classic,from%20the%20Universal%20Background%20model) Basics of Speaker Verification -

<https://maelfabien.github.io/machinelearning/basics_speech/>

[[3]](https://github.com/943274923/Speaker-Recognition#:~:text=Speaker%20recognition%20is%20the%20identification,little%20easier%20than%20speaker%20recognition) [[17]](https://github.com/943274923/Speaker-Recognition#:~:text=SideKit%201) [[32]](https://github.com/943274923/Speaker-Recognition#:~:text=data_preprocessing) GitHub - 943274923/Speaker-Recognition: This is a Speaker Recognition and Verification project using SideKit-1.3.1

<https://github.com/943274923/Speaker-Recognition>

[[8]](https://github.com/ppwwyyxx/speaker-recognition#:~:text=The%20Dockerfile%20can%20be%20used,started%20with%20the%20project%20easier) [[10]](https://github.com/ppwwyyxx/speaker-recognition#:~:text=Voice%20Activity%20Detection) [[12]](https://github.com/ppwwyyxx/speaker-recognition#:~:text=Dependencies) [[13]](https://github.com/ppwwyyxx/speaker-recognition#:~:text=Feature%3A) [[16]](https://github.com/ppwwyyxx/speaker-recognition#:~:text=,JFA) [[26]](https://github.com/ppwwyyxx/speaker-recognition#:~:text=Model%3A) GitHub - ppwwyyxx/speaker-recognition: A Speaker Recognition System

<https://github.com/ppwwyyxx/speaker-recognition>

[[9]](https://github.com/pyyush/VoiceID#:~:text=1.%20python%20,numpy) GitHub - pyyush/VoiceID: End-to-End Text Independent Speaker Verification

<https://github.com/pyyush/VoiceID>

[[11]](https://projets-lium.univ-lemans.fr/sidekit/overview/index.html#:~:text=SIDEKIT%20aims%20at%20providing%20the,for%20both%20Linux%20and%20MacOS) An Overview of SIDEKIT - Projets

<https://projets-lium.univ-lemans.fr/sidekit/overview/index.html>

[[14]](https://huggingface.co/speechbrain/spkrec-xvect-voxceleb#:~:text=import%20torchaudio%20from%20speechbrain,encode_batch%28signal) [[18]](https://huggingface.co/speechbrain/spkrec-xvect-voxceleb#:~:text=Speaker%20Verification%20with%20xvector%20embeddings,on%20Voxceleb) [[19]](https://huggingface.co/speechbrain/spkrec-xvect-voxceleb#:~:text=This%20repository%20provides%20all%20the,Voxceleb%201%2B%20Voxceleb2%20training%20data) [[25]](https://huggingface.co/speechbrain/spkrec-xvect-voxceleb#:~:text=3) speechbrain/spkrec-xvect-voxceleb · Hugging Face

<https://huggingface.co/speechbrain/spkrec-xvect-voxceleb>

[[20]](https://github.com/Rumeysakeskin/Speaker-Verification#:~:text=Speaker%20verification%20is%20verifying%20the,from%20language%20via%20NVIDIA%20NeMo) [[21]](https://github.com/Rumeysakeskin/Speaker-Verification#:~:text=The%20cosine%20similarity%20metric%20was,used%20for%20prediction) GitHub - Rumeysakeskin/Speaker-Verification: Verifying the identity of a person from characteristics of the voice independent from language via NVIDIA NeMo models (ECAPA-TDNN, SpeakerNet, TitaNet-L).

<https://github.com/Rumeysakeskin/Speaker-Verification>

[[22]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=Conclusion) [[23]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=,Siuuuuuuuuu%21%E2%80%9D) [[24]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=Top%20two%20speakers%3A%20) [[35]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=To%20recognize%20the%20user%20with,Ronaldo%E2%80%99s%20voice%20in%20the%20database) [[36]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=Convert%20the%20new%20voice%20into,embeddings) [[37]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=QdrantDB%3A%20An%20Overview) [[38]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=Speaker%20Recognition%3A) [[39]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=To%20get%20top%202%20similar,mode%20in%20top%203%20case) [[40]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=,len%28speakers%29%3A%20top_two_names.append%28speakers%5Baligned_id) [[41]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=%5BScoredPoint%28id%3D2%2C%20version%3D0%2C%20score%3D0.6956655%2C%20payload%3D,vector%3DNone) [[42]](https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb#:~:text=By%20the%20above%20result%20you,data%20and%20try%20this%20out) Build an Audio-Driven Speaker Recognition System Using Open-Source Technologies — Resemblyzer and QdrantDB. | by Karan Shingde | Medium

<https://medium.com/@karanshingde/build-an-audio-driven-speaker-recognition-system-using-open-source-technologies-resemblyzer-and-6499cf0246eb>

[[27]](https://github.com/modelscope/3D-Speaker#:~:text=3D,research%20of%20speech%20representation%20disentanglement) GitHub - modelscope/3D-Speaker: A Repository for Single- and Multi-modal Speaker Verification, Speaker Recognition and Speaker Diarization

<https://github.com/modelscope/3D-Speaker>

[[28]](https://arxiv.org/html/2401.09441v1#:~:text=This%20paper%20presents%20VoxCeleb,identification%20results%20suggest%20that%20the) [[29]](https://arxiv.org/html/2401.09441v1#:~:text=obtained%20the%20audio%20from%20YouTube,and%20a%20balanced%20gender%20distribution) [[30]](https://arxiv.org/html/2401.09441v1#:~:text=European%20Portuguese%20datasets%20,on%20biometrics%20and%20secure%20applications) [[31]](https://arxiv.org/html/2401.09441v1#:~:text=12%20,on%20biometrics%20and%20secure%20applications) Voxceleb-ESP: preliminary experiments detecting Spanish celebrities from their voices

<https://arxiv.org/html/2401.09441v1>

[[33]](https://github.com/deep-privacy/sidekit#:~:text=,privacy%2Fsidekit%2Freleases%2Fdownload%2Fsidekit_v0.1%2F%20best_halp_clr_adam_aam0.2_30_b256_vox12.pt_epoch71) [[34]](https://github.com/deep-privacy/sidekit#:~:text=,privacy%2Fsidekit%2Freleases%2Fdownload%2Fsidekit_v0.1%2F%20best_halp_clr_adam_aam0.2_30_b256_vox12.pt_epoch71) [[46]](https://github.com/deep-privacy/sidekit#:~:text=import%20torchaudio%20from%20sidekit,Xtractor) [[47]](https://github.com/deep-privacy/sidekit#:~:text=model_opts%5B,embedding_size) GitHub - deep-privacy/sidekit: For further release go to: https://git-lium.univ-lemans.fr/speaker/sidekit

<https://github.com/deep-privacy/sidekit>

[[43]](https://github.com/bedangSen/VoiceSens#:~:text=Image%3A%20source%3A%20imgur) [[44]](https://github.com/bedangSen/VoiceSens#:~:text=1,Authenticate%20an%20existing%20user) [[45]](https://github.com/bedangSen/VoiceSens#:~:text=To%20Do) GitHub - bedangSen/VoiceSens: A Voice Biometric Application using Watson Speech to Text

<https://github.com/bedangSen/VoiceSens>

[[48]](https://speechbrain.readthedocs.io/en/latest/tutorials/basics/what-can-i-do-with-speechbrain.html#:~:text=What%20can%20I%20do%20with,) What can I do with SpeechBrain? - Read the Docs

<https://speechbrain.readthedocs.io/en/latest/tutorials/basics/what-can-i-do-with-speechbrain.html>