



**INSTITUTO POLITÉCNICO
NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO**



ESCOM – Licenciatura en Ciencia de Datos

Proyecto Final

Procesamiento de Lenguaje Natural

PhD. Joel Omar Juárez Gambino

Alumnos

Daniel Armas Ramírez Emiliano Prezas Bernal

Karla Esther Escamilla Gachuz Nathaly Dorado Alcalá

Contenido

Introducción	3
Corpus y Datasets	3
Recursos adicionales.....	4
Solución.....	5
Experimentos y resultados	7
Conclusions and future work	399

Introducción

En nuestro planeta se hablan cerca de 44 idiomas, aunque esa cifra no es exacta, ya que existen más dialectos en la mayoría de los países, el idioma es vital para dar a conocer nuestros pensamientos, pero es casi imposible saber todos los idiomas. Es por ello que investigadores y empresas han usado la tecnología para ayudar a tratar de automatizar la traducción de un idioma a otro, es ahí donde el Procesamiento del Lenguaje Natural y el Aprendizaje automático entran en acción, la traducción automática de texto ha sido pilar para el área de PLN (Procesamiento del Lenguaje Natural), comenzando desde reglas hechas a mano, pasando a modelos estadísticos como el modelo oculto de Markov, técnicas de minería de texto, hasta llegar a técnicas de Deep Learning con redes neuronales recurrentes y una arquitectura que transforma muchas industrias, el Transformer.

Este recorrido ha permitido crear soluciones más rápidas y eficientes de entrenar, incluso al punto de generar la voz traducida, aunque existe un punto aquí, son voces ya definidas, no existen herramientas o aplicaciones que permitan traducir tu mensaje con tu voz, es de ahí que nace nuestra idea de proyecto.

Con esta aplicación se podrá realizar la grabación de un mensaje, seleccionar un idioma al que se desea traducir y hacer lo posible por clonar la voz, esto a través de modelos que son el Estado del Arte como Whisper y XTTS.

Corpus y Datasets

Dataset para XTTS

- CSTR VCTK Corpus: English Multi-speaker Corpus for CSTR Voice Cloning Toolkit (version 0.92) [<https://datashare.ed.ac.uk/handle/10283/3443>]
- Device Recorded VCTK (Small subset version) [<https://datashare.ed.ac.uk/handle/10283/3038>]

Dataset para Whisper

- El dataset no es abierto, solo se especifica que se usaron 680,000 horas de audio etiquetado, de esas 680,000 horas, 117,000 horas cubren 96 lenguajes diferentes y 125,000 horas de X→inglés datos de traducción.

Dataset de Bark

- El dataset fue creado por suno.ai pero no se dan detalles técnicos, ya que es cerrado.

Entrada real utilizada:

- Se utilizó voz grabada directamente por el usuario como corpus dinámico.
- Se conservan dos archivos de audio producto de los experimentos de transcripción y traducción: record(hindi).wav y record(español).wav

Recursos adicionales

Modelos LLM utilizados:

- Whisper (OpenAI) para transcripción y detección de idioma.
- NLLB (facebook/nllb-200-distilled-600M) para traducción automática.
- XTTS v2 (coqui.ai) para clonación multilingüe de voz.
- Bark (suno.ai) para síntesis de voz multilingüe.

Librerías y tecnologías:

- De tecnologías utilizamos Python.
- Y de librerías utilizamos las siguientes: torch, transformers, TTS, whisper, bark, sounddevice, tkinter, simpleaudio.
- Para reconocer los archivos de audio y transformarlos al espectrograma mel-log que necesita Whisper, es necesario instalar la librería externa ffmpeg del sitio oficial (para ejecución local).

Solución

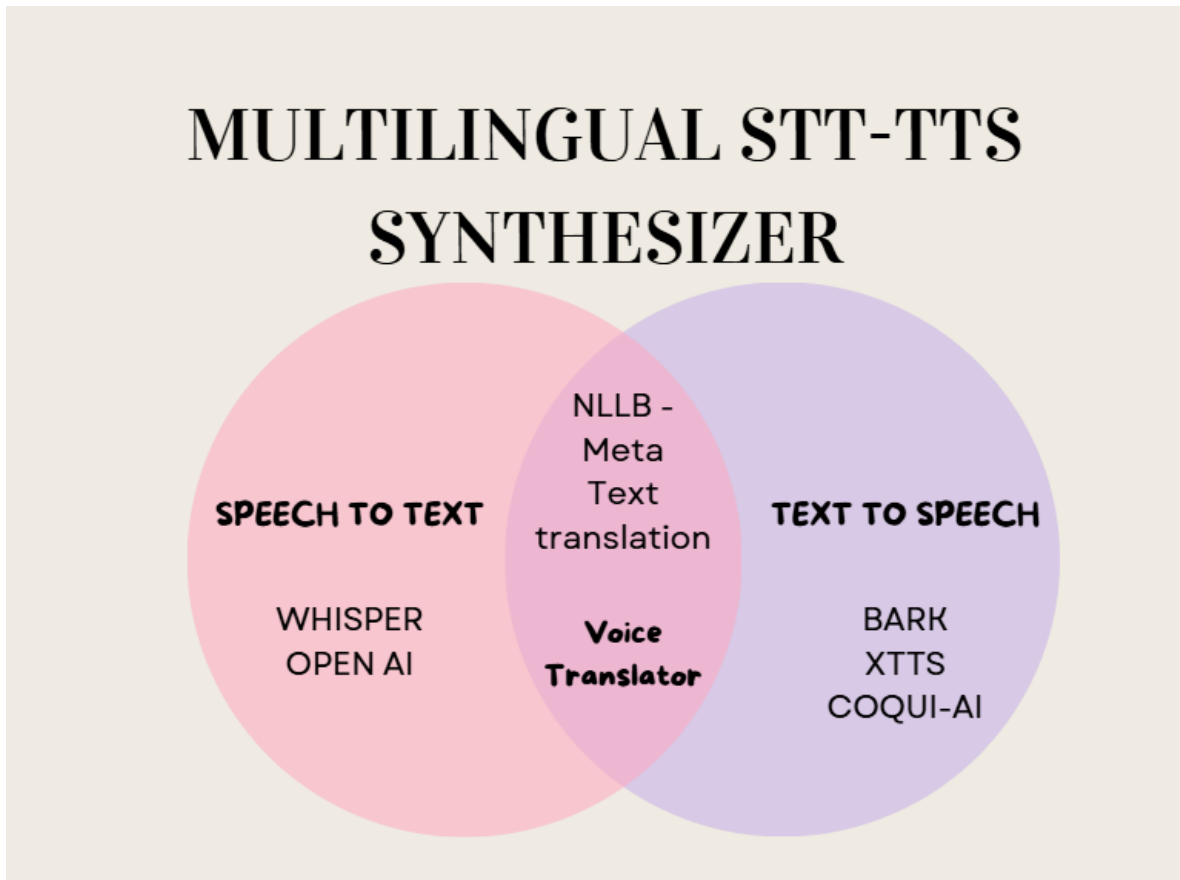


Ilustración 1. Elaboración propia

Nuestra aplicación fue diseñada como un pipeline modular de procesamiento de lenguaje natural multilingüe con soporte de voz. El flujo del sistema permite capturar un mensaje hablado, traducirlo al idioma seleccionado, y generar una versión hablada del mensaje traducido, ya sea utilizando la voz original del usuario (clonación) o una voz sintética configurable (género e idioma).

Como se muestra en la siguiente imagen, el proceso inicia cuando el usuario graba un mensaje. Este es procesado por el modelo Whisper, que se encarga de transcribir el audio y detectar el idioma original. Después, el usuario selecciona el idioma al que desea traducir el contenido.

A partir de ahí, el usuario puede elegir dos caminos:

1. Clonación de voz: el modelo XTTS v2 genera un nuevo archivo de audio utilizando la voz original del usuario, ahora en el idioma destino.
2. Síntesis de voz: aquí entra el modelo Bark, que permite generar una voz artificial configurable por género e idioma.

Sin embargo, es importante mencionar que, aunque la interfaz gráfica incluye la opción de síntesis con Bark, no se logró integrar funcionalmente esta opción a la aplicación final debido a incompatibilidades con la versión de torch y el entorno local de independientes, donde se verificó su capacidad para generar audio natural multilingüe.

Por tanto, Bark aparece en la interfaz, pero actualmente genera un error al intentar ejecutarse desde ella. Se dejó en la interfaz con el fin de en un futuro poder implementarlo de manera satisfactoria cuando actualicen su compatibilidad con torch.

ejecución. A pesar de ello, sí se realizaron experimentos exitosos con Bark en notebooks independientes, donde se verificó su capacidad para generar audio natural multilingüe.

Por tanto, Bark aparece en la interfaz, pero actualmente genera un error al intentar ejecutarse desde ella. Se dejó en la interfaz con el fin de en un futuro poder implementarlo de manera satisfactoria cuando actualicen su compatibilidad con torch.

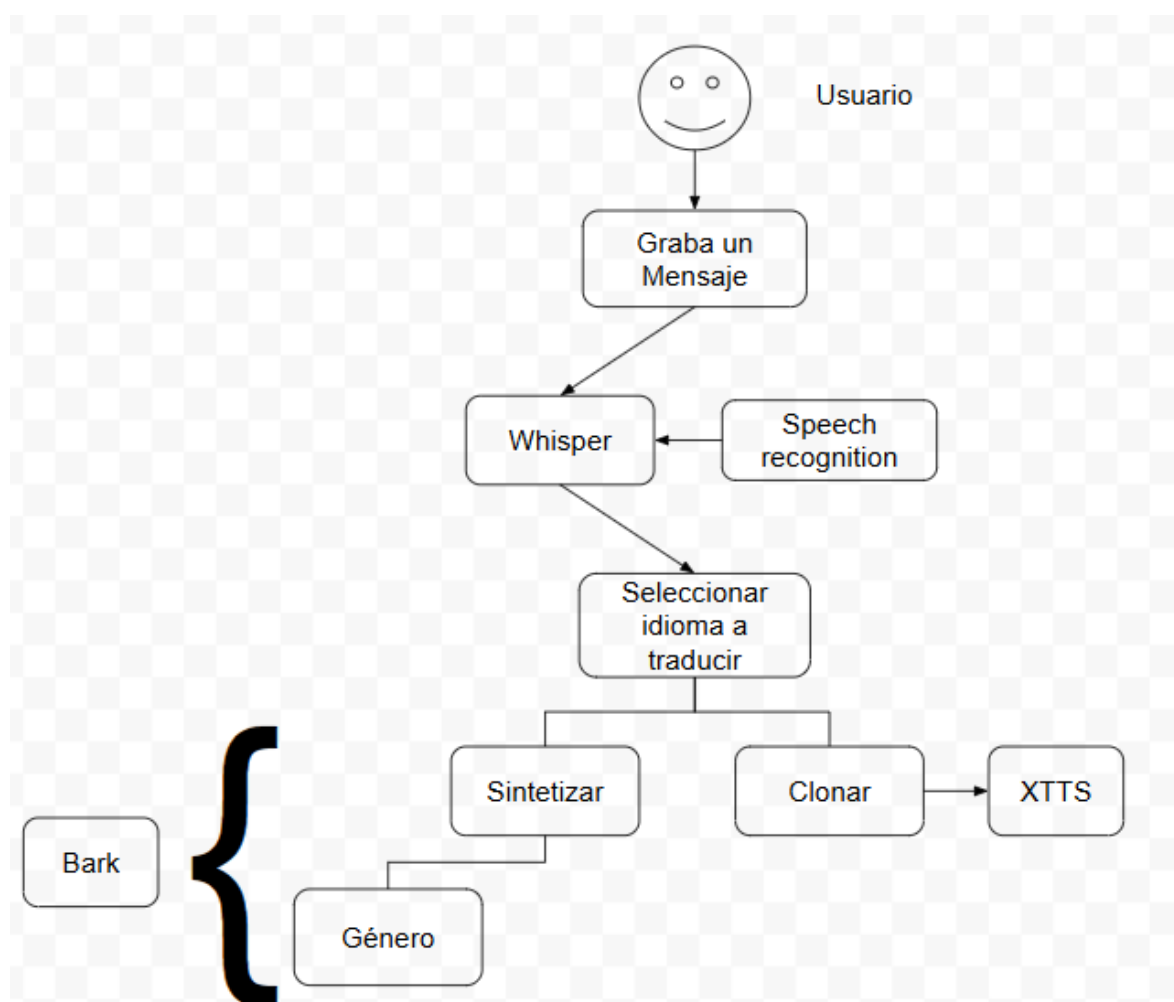


Ilustración 2. Elaboración propia

Experimentos y resultados

Modelos de Automatic Speech Recognition (ASR).

1. Wav2vec53spanish del CIEMPIESS-UNAM Project

El modelo acústico "*wav2vec2-large-xlsr-53-spanish-ep5-944h*" es adecuado para el Reconocimiento Automático del Habla (ASR) en español. Es el resultado del ajuste fino del modelo "*facebook/wav2vec2-large-xlsr-53*" durante 5 épocas, utilizando aproximadamente 944 horas de datos en español recopilados o desarrollados por el Proyecto CIEMPIESS-UNAM desde 2012.

La mayoría de los datos están disponibles en la página oficial del Proyecto CIEMPIESS-UNAM: <http://www.ciempiess.org/>.

El resto se puede encontrar en repositorios públicos como LDC o OpenSLR.

Requerimientos:

- transformers y datasets de Huggingface
- sounddevice
- wavio
- ipyweb rtc y notebook
- ffmpeg
- libportaudio2

Importaciones primarias:

```
import os
import numpy as np

try:
    import tensorflow
except ImportError:
    pass

import torch
import pandas as pd
import torchaudio

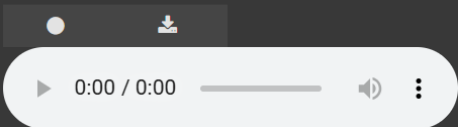
from ipywebRTC import AudioRecorder, CameraStream
from IPython.display import Audio, display
import ipywidgets as widgets

DEVICE = "cuda" if torch.cuda.is_available() else "cpu"

from google.colab import output
output.enable_custom_widget_manager()
```

Interfaz y conversión de archive

```
camera = CameraStream(constraints={'audio': True, 'video': False})
recorder = AudioRecorder(stream=camera)
recorder
```



```
with open('recording.webm', 'wb') as f:
    f.write(recorder.audio.value)
!ffmpeg -i recording.webm -ac 1 -f wav my_recording.wav -y -hide_banner -loglevel panic
```


Prueba de transcripción:

```
from transformers import pipeline

transcriber = pipeline("automatic-speech-recognition",
                        model="carlosdanielhernandezmena/wav2vec2-large-xlsr-53-spanish-ep5-944h")
audio_file = 'my_recording.wav'
transcriber(audio_file)

Device set to use cpu
{'text': 'colá es catorce de junio las diez de la noche'}
```

Como podemos observar, no se reconoce de manera efectiva una palabra de uso común; como lo es “hola”, que es transcrito como “colá”. Tal vez esto es consecuencia del uso de datasets en español aunque principalmente mexicanos, también incluyendo audios de otras regiones latinas, lo que pudo causar interferencia en la entonación de la palabra y dar como resultado una “a” con acento.

2. Whisper de Open AI (transcripción)

Whisper es un sistema de reconocimiento automático del habla (ASR), entrenado con 680 000 horas de datos multilingües y multitarea supervisados, obtenidos de la web.

- Es compatible con más de 90 de los idiomas más populares.
- Tiene una arquitectura con enfoque simple de punto a punto implementado como un transformador codificador-decodificador.
- El audio de entrada se divide en fragmentos de 30 segundos, se convierte en un espectrograma log-Mel y luego se pasa a un codificador.

Se entrena un decodificador para predecir el subtítulo de texto correspondiente, mezclado con tokens especiales que dirigen al modelo simple para realizar tareas como identificación de idioma, marcas de tiempo a nivel de frase, transcripción de habla multilingüe y traducción de habla al inglés. Como se muestra en la ilustración:

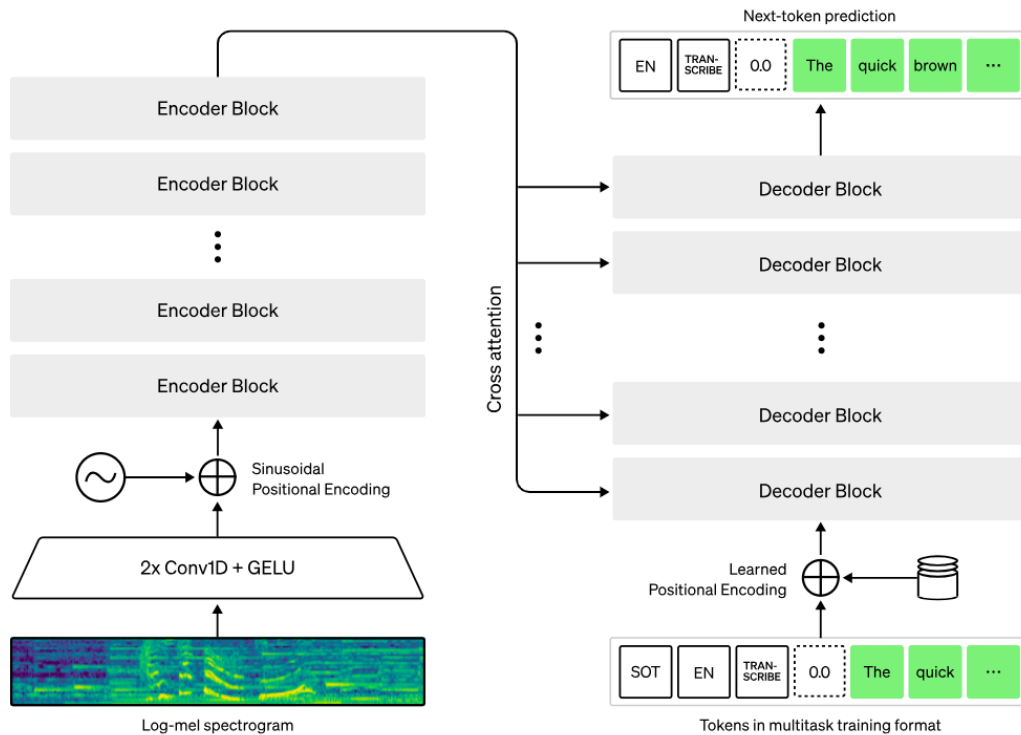


Ilustración 3. From Whisper's OPEN AI documentation

Podemos observar a detalle la forma de transcripción:

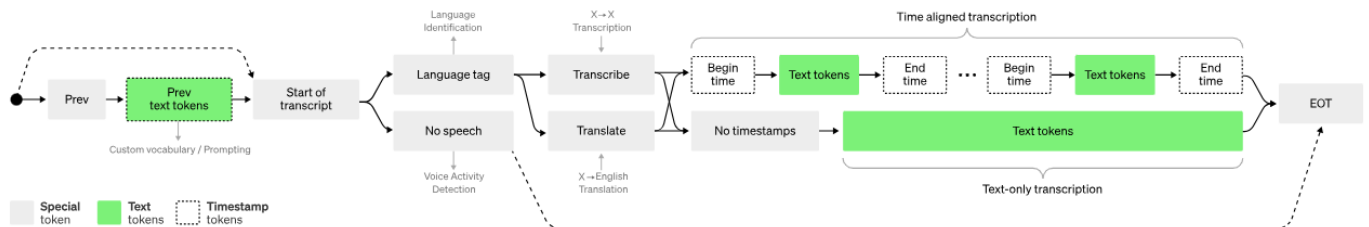


Ilustración 4. From Whisper's OPEN AI documentation

Requerimientos

- transformers y datasets de Huggingface
- sounddevice
- wavio
- ipywebbrtc y notebook
- ffmpeg
- libportaudio2

Importaciones primarias

```
import os
import numpy as np

try:
    import tensorflow
except ImportError:
    pass

import torch
import pandas as pd
import whisper
import torchaudio

from ipywebRTC import AudioRecorder, CameraStream
from IPython.display import Audio, display
import ipywidgets as widgets

DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
```

Ilustración 5

Grabando la voz en audio

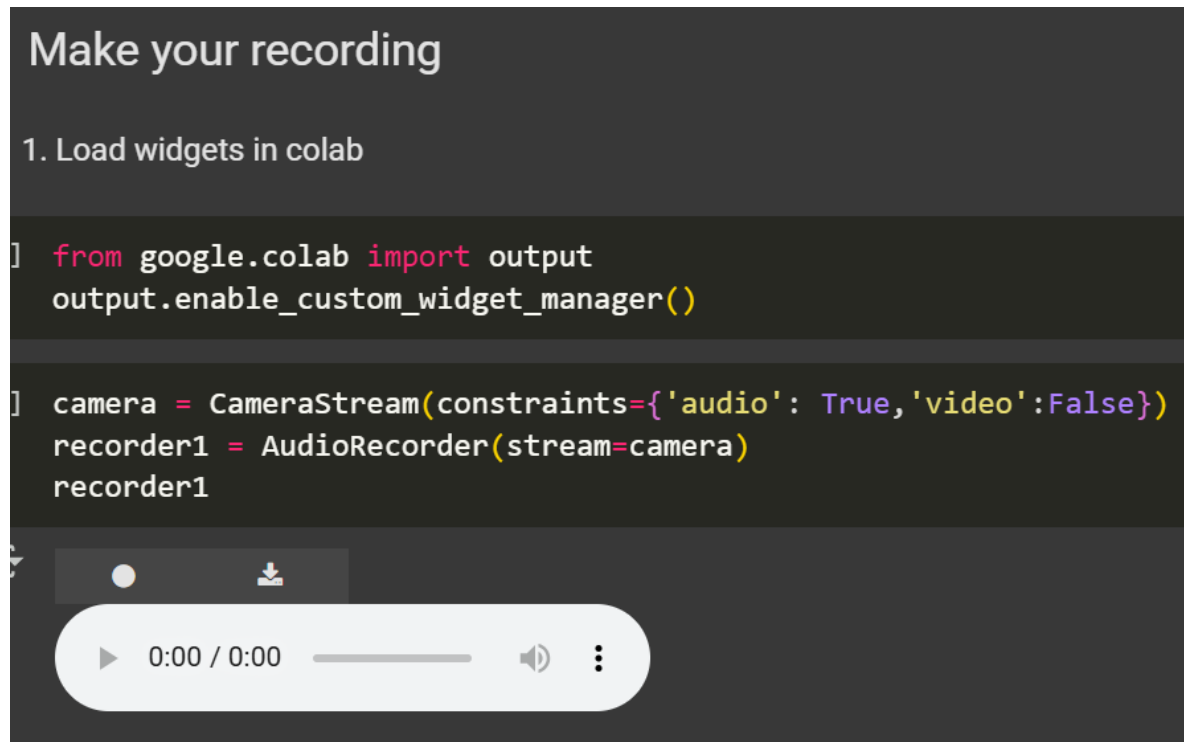


Ilustración 6

Transformación de audio al espectro compatible con Whisper

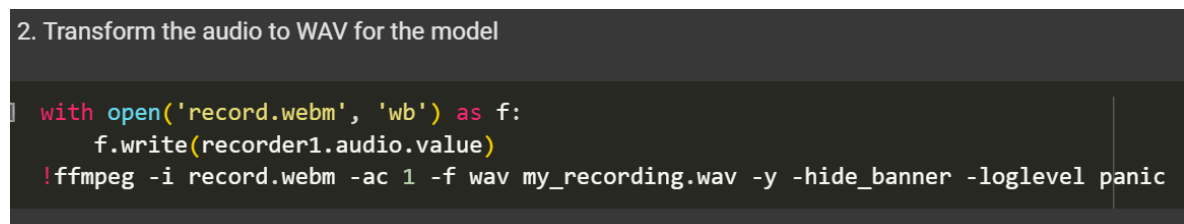


Ilustración 7

Cargando el modelo

Select Language and mode

1. We set the model for multilingual listening obtaining the set of languages

```
language_options = whisper.tokenizer.TO_LANGUAGE_CODE
language_list = list(language_options.keys())
language_list
```

Load Whisper model

```
wmodel = whisper.load_model("base")
print(
    f"Model is {'multilingual' if wmodel.is_multilingual else 'English-only'} "
    f"and has {sum(np.prod(p.shape) for p in wmodel.parameters()):,} parameters."
)
```

```
Model is multilingual and has 71,825,920 parameters.
```

Ilustración 8

EXPERIMENTO 1 CON WHISPER

Ajuste de parámetros para recibir específicamente audio en español

```
options = whisper.DecodingOptions(language='spanish',
                                   task='transcribe',
                                   without_timestamps=True)
options
```

Ilustración 9

Test de transcripción de *frase simplificada moderadamente larga en español*

✓ Start transcribing

Time to load our file and see the results

```
[ ] audio = whisper.load_audio("my_recording.wav")
    audio = whisper.pad_or_trim(audio)
    mel = whisper.log_mel_spectrogram(audio).to(model.device)
    result = model.decode(mel, options)
```

```
[ ] result.text
```

```
⇒ 'Hola, es 14 de junio, 10 de la noche, este experimento es con Whisper.'
```

Ilustración 10

Como observamos, el desempeño con Whisper parece ser mucho mejor que el modelo anterior, detectando no solamente la palabra “Hola”, sino también signos de puntuación y capitalización de nombres propios, lo que resulta excelente para la síntesis de voz, al menos desde el idioma español.

EXPERIMENTO 2 CON WHISPER

Ajustando los parámetros para recibir cualquier audio y detectar el idioma automáticamente

```
options = whisper.DecodingOptions(task='transcribe', without_timestamps=True)
options
```

Ilustración 11

Test de transcripción en un idioma menos popular, con un alfabeto distinto al latino y siendo además la letra parte de una canción.

```
▼ Start transcribing

Time to load our file and see the results

▶ audio = whisper.load_audio("record.webm")
  audio = whisper.pad_or_trim(audio)
  mel = whisper.log_mel_spectrogram(audio).to(wmodel.device)
  result = wmodel.decode(mel, options)

[ ] result.text

↔ 'ज़ साया बाशान को जान राणगगे मेरे नाब को'
```

Ilustración 12

Detección automática del idioma

```
Which language did I speak?

_, probs = wmodel.detect_language(mel)
detected_lang = max(probs, key=probs.get)

print(f"Language code: {detected_lang}")

✓ Language code: hi
```

Ilustración 13

Parece que tuvimos éxito identificando el idioma original del audio. La muestra fue proporcionada por un hablante nativo de otro idioma oriental (urdu), como parte de un experimento para la posterior traducción. Por la parte de la transcripción, el

texto resulta acertado y preciso para el audio, confirmado por el usuario de la muestra de audio.

Modelos de Traducción de Texto

1. No Language Left Behind (NLLB) de Meta

NLLB es un avanzado sistema de traducción automática neuronal desarrollado por Meta AI, orientado a reducir las barreras lingüísticas en todo el mundo. Este modelo traduce directamente entre 200 idiomas, incluyendo tanto los de alta y baja disponibilidad de recursos, emplea una arquitectura de "Mixture of Experts" (modelo condicional con módulos especializados), diseñada para mejorar la precisión en lenguas poco representadas

Entrenado con conjuntos de datos cuidadosamente seleccionados, equilibrando idiomas con muchos y pocos recursos, y evaluado en más de 40 000 direcciones de traducción, alcanzando un aumento del 44 % en BLEU frente al estado anterior del arte.

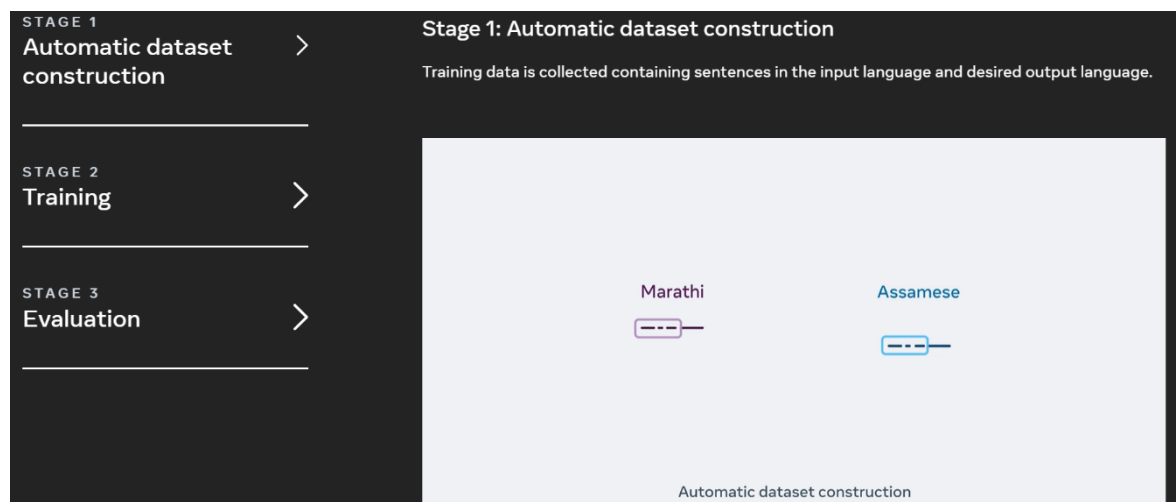


Ilustración 14. From NLLB Meta's documentation

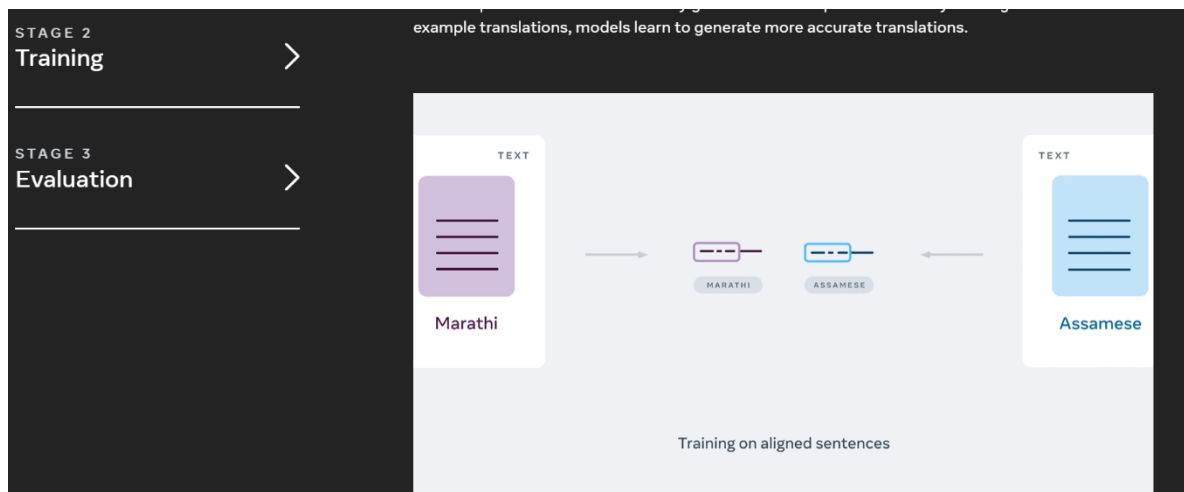


Ilustración 15. From NLLB Meta's documentation

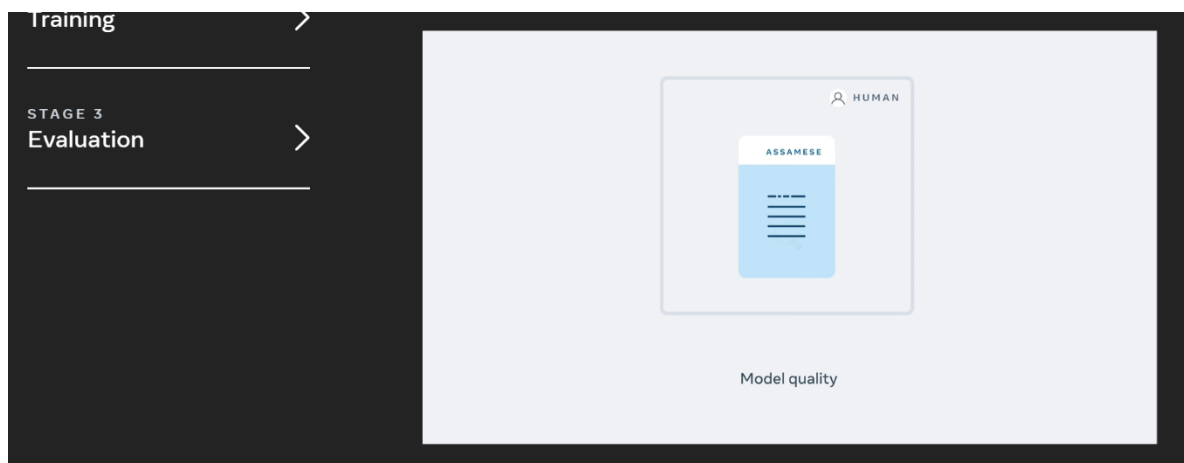


Ilustración 16. From NLLB Meta's documentation

Importaciones primarias:

1. Building translation architecture

```
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
import torch

#pretrained Meta translator
model_name = "facebook/nllb-200-distilled-600M"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSeq2SeqLM.from_pretrained(model_name).to(DEVICE)
```








 /usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (<https://huggingface.co/settings/tokens>),
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
warnings.warn(
tokenizer_config.json: 100%  564/564 [00:00<00:00, 22.6kB/s]
sentencepiece.bpe.model: 100%  4.85M/4.85M [00:00<00:00, 9.13MB/s]
tokenizer.json: 100%  17.3M/17.3M [00:00<00:00, 33.4MB/s]
special_tokens_map.json: 100%  3.55k/3.55k [00:00<00:00, 8.34kB/s]
config.json: 100%  846/846 [00:00<00:00, 40.4kB/s]
pytorch_model.bin: 100%  2.46G/2.46G [02:08<00:00, 20.3MB/s]

Ilustración 17

EXPERIMENTO 1 DE TRADUCCIÓN CON FRASE COTIDIANA

Testings:

```
Coreano

corpus = result.text
idioma = "coreano"

traduccion = translate(corpus, idioma)
print(f"Texto original: {corpus}")
print(f"Traducción a {idioma}: {traduccion}")

Texto original: Hola, hoy es 15 de julio.
Traducción a coreano: 안녕하세요, 오늘은 7월 15일입니다.

Inglés

idioma = "inglés"

traduccion = translate(corpus, idioma)
print(f"Texto original: {corpus}")
print(f"Traducción a {idioma}: {traduccion}")

Texto original: Hola, hoy es 15 de julio.
Traducción a inglés: Hello, today is July 15th.
```

Ilustración 18

Japonés

```
idioma = "japonés"

traduccion = translate(corpus, idioma)
print(f"Texto original: {corpus}")
print(f"Traducción a {idioma}: {traduccion}")
```

30]

```
.. Texto original: Hola, hoy es 15 de julio.
Traducción a japonés: こんにちは 今日は7月15日です
```

Chino

```
idioma = "chino"

traduccion = translate(corpus, idioma)
print(f"Texto original: {corpus}")
print(f"Traducción a {idioma}: {traduccion}")
```

31]

```
.. Texto original: Hola, hoy es 15 de julio.
Traducción a chino: 你好,今天是7月15日.
```

Ilustración 19

Francés

```
idioma = "franc  s"

traduccion = translate(corpus, idioma)
print(f"Texto original: {corpus}")
print(f"Traducci  n a {idioma}: {traduccion}")
```

Texto original: Hola, hoy es 15 de julio.

Traducci  n a franc  s: Bonjour, aujourd'hui est le 15 juillet.

Italiano

```
idioma = "italiano"

traduccion = translate(corpus, idioma)
print(f"Texto original: {corpus}")
print(f"Traducci  n a {idioma}: {traduccion}")
```

Texto original: Hola, hoy es 15 de julio.

Traducci  n a italiano: Ciao, oggi    il 15 luglio.

Ilustraci  n 20

Ruso

```
idioma = "ruso"

traduccion = translate(corpus, idioma)
print(f"Texto original: {corpus}")
print(f"Traducción a {idioma}: {traduccion}")
```

Texto original: Hola, hoy es 15 de julio.
Traducción a ruso: Здравствуйте, сегодня 15 июля.

Urdu

```
idioma = "urdu"

traduccion = translate(corpus, idioma)
print(f"Texto original: {corpus}")
print(f"Traducción a {idioma}: {traduccion}")
```

Texto original: Hola, hoy es 15 de julio.
Traducción a urdu: ہیلو، آج 15 جولائی ہے

Ilustración 21

Alemán

```
idioma = "alemán"

traduccion = translate(corpus, idioma)
print(f"Texto original: {corpus}")
print(f"Traducción a {idioma}: {traduccion}")
```

Texto original: Hola, hoy es 15 de julio.
Traducción a alemán: Hallo, heute ist der 15. Juli.

Ilustración 22

EXPERIMENTO 2 DE TRADUCCIÓN CON FRASE EN ALFABETO ORIENTAL, FRASE SUBJETIVA DE CANCIÓN.

Testing

✓ 5. Testing

Korean

```
[ ] corpus = result.text
    idioma = "korean"

    traduccion1 = translate(corpus, idioma)
    print(f"Original text: {corpus}")
    print(f"Translated to {idioma}: {traduccion1}")
```

⇒ Original text: ज़ साया बाशान को जान गाणगगे मेरे नाब को
Translated to korean: 저 는 내 종을 알게 될 것 이다

Es aventurada en la traducción, la escritura tiene algunas fallas gramaticales, como “저 는” debería aparecer como “저는”.

English

```
[ ] idioma = "english"

    traduccion2 = translate(corpus, idioma)
    print(f"Texto original: {corpus}")
    print(f"Traducción a {idioma}: {traduccion2}")
```

⇒ Texto original: ज़ साया बाशान को जान गाणगगे मेरे नाब को
Traducción a english: I will know my Lord

La traducción no atrapa el concepto original de la frase.

```
Japanese

idioma = "japanese"

traduccion4 = translate(corpus, idioma)
print(f"Texto original: {corpus}")
print(f"Traducción a {idioma}: {traduccion4}")

⇒ Texto original: ज्ञ साया बाशान को जान साणमगे मेरे नाब को
Traducción a japanese: () () () () () () () () () () () () () () () () () () () ()
```

(en el original sí se percibe el alfabeto correspondiente, la traducción era aproximada)

```
Chinese

idioma = "chinese"

traduccion5 = translate(corpus, idioma)
print(f"Texto original: {corpus}")
print(f"Traducción a {idioma}: {traduccion5}")

⇒⇒ Texto original: سلام کیسے ہیں آپ
Traducción a chinese: 你好,你怎么样?
```

Traducción aproximada.

French

```
▶ idioma = "french"

traduccion6 = translate(corpus, idioma)
print(f"Texto original: {corpus}")
print(f"Traducción a {idioma}: {traduccion6}")
```

⇒ Texto original: سلام کیسے ہیں آپ
Traducción a french: Bonjour, comment allez-vous ?

Traducción más cercana.

Italian

```
[ ] idioma7 = "italian"

traduccion7 = translate(corpus, idioma)
print(f"Texto original: {corpus}")
print(f"Traducción a {idioma}: {traduccion7}")
```

⇒ Texto original: سلام کیسے ہیں آپ
Traducción a urdu: سلام آپ کیسے ہیں

Traducción precisa.

Russian

```
idioma = "russian"

traduccion8 = translate(corpus, idioma)
print(f"Texto original: {corpus}")
print(f"Traducción a {idioma}: {traduccion8}")
```

⇒ Texto original: ज़ साया बाशान को जान गाणगगे मेरे नाब को
Traducción a russian: И я знаю, что ты не знаешь моего имени.

Traducción extraña, utiliza el alfabeto ruso para producir el sonido de la palabra en hindi, pero no la traduce.

Urdu

```
[ ] idioma = "urdu"

traduccion9 = translate(corpus, idioma)
print(f"Texto original: {corpus}")
print(f"Traducción a {idioma}: {traduccion9}")
```

⇒ Texto original: ज़ साया बाशान को जान गाणगगे मेरे नाब को
Traducción a urdu: اور میری جان کو بھی جان لے گا

German

```
[ ] idioma = "german"

traduccion10 = translate(corpus, idioma)
print(f"Texto original: {corpus}")
print(f"Traducción a {idioma}: {traduccion10}")
```

⇒ Texto original: ज़ साया बाशान को जान गाणगगे मेरे नाब को
Traducción a german: Und die Schatten werden mich erfahren.

Tanto en urdu como en alemán las traducciones son precisas.

2. Whisper de Open AI (traducción)

Entrenado con conjuntos de datos cuidadosamente seleccionados, equilibrando idiomas con muchos y pocos recursos, y evaluado en más de 40 000 direcciones de traducción, alcanzando un aumento del 44 % en BLEU frente al estado anterior del arte

Importaciones primarias:

```
import os
import numpy as np

try:
|   import tensorflow
except ImportError:
|   pass

import torch
import pandas as pd
import whisper
import torchaudio

from ipywebRTC import AudioRecorder, CameraStream
from IPython.display import Audio, display
import ipywidgets as widgets

DEVICE = "cuda" if torch.cuda.is_available() else "cpu"
```

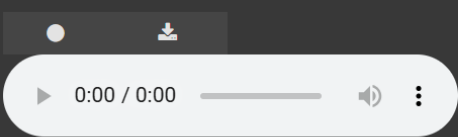
Ilustración 23

```
from google.colab import output
output.enable_custom_widget_manager()
```

Ilustración 24

Interfaz y conversión de archive

```
camera = CameraStream(constraints={'audio': True, 'video': False})
recorder = AudioRecorder(stream=camera)
recorder



with open('recording.webm', 'wb') as f:
    f.write(recorder.audio.value)
!ffmpeg -i recording.webm -ac 1 -f wav my_recording.wav -y -hide_banner -loglevel panic
```

Ilustración 25

Prueba de traducción:

Modelos de Text To Speech (TTS)

1. Bark (artificial voice synthesizing)

Languages supported

- Language
- English (en)
- German (de)
- Spanish (es)
- French (fr)
- Hindi (hi)
- Italian (it)
- Japanese (ja)
- Korean (ko)
- Polish (pl)
- Portuguese (pt)
- Russian (ru)
- Turkish (tr)
- Chinese, simplified (zh)

Our language list

- 'spanish': 'spa_Latn',
- 'urdu': 'urd_Arab',
- 'russian': 'rus_Cyrl',
- 'english': 'eng_Latn',
- 'korean': 'kor_Hang',
- 'japanese': 'jpn_Jpan',
- 'chinese': 'zho_Hans',
- 'french': 'fra_Latn',
- 'italian': 'ita_Latn',
- 'german': 'deu_Latn'

Voice generation function

```
3. Generating voice

def generate_bark_audio(text, language, gender="neutral", filename=None):
    language = language.lower()
    gender = gender.lower()

    if language not in languages:
        raise ValueError(f"Unsupported language: {language}. Available options: {list(languages.keys())}")

    if gender not in ["male", "female", "neutral"]:
        raise ValueError("Invalid gender. Choose 'male', 'female', or 'neutral'.")

    #corresponding voice
    voices = voice_presets.get(language)

    if voices is None:
        print(f"No voices defined for '{language}'. Using Spanish female voice as fallback.")
        voice_preset = "v2/es_speaker_1"
    elif gender == "neutral":
        voice_preset = list(voices.values())[0]
    else:
        voice_preset = voices.get(gender)
        if voice_preset is None:
            print(f"No '{gender}' voice found for '{language}'. Using female voice as fallback.")

    #generating audio
    inputs = processor(text, voice_preset=voice_preset)
    audio_array = model.generate(**inputs).cpu().numpy().squeeze()
    sample_rate = model.generation_config.sample_rate

    if filename is None:
        filename = f"audio_{language}_{gender}.wav"
    elif not filename.endswith(".wav"):
        filename += ".wav"

    scipy.io.wavfile.write(filename, rate=sample_rate, data=audio_array)
    print(f"Audio saved as: {filename}")

    return Audio(audio_array, rate=sample_rate)
```

Ilustración 26

```
#generating audio
inputs = processor(text, voice_preset=voice_preset)
audio_array = model.generate(**inputs).cpu().numpy().squeeze()
sample_rate = model.generation_config.sample_rate

if filename is None:
    filename = f"audio_{language}_{gender}.wav"
elif not filename.endswith(".wav"):
    filename += ".wav"




scipy.io.wavfile.write(filename, rate=sample_rate, data=audio_array)
print(f"Audio saved as: {filename}")

return Audio(audio_array, rate=sample_rate)
```

Ilustración 27

Testing

```
generate_bark_audio(  
    text=traduccion2,  
    language="english",  
    gender="male"  
)
```

en_speaker_6_semantic_prompt.npy: 100%  2.60k/2.60k [00:00<00:00, 109kB/s]
en_speaker_6_coarse_prompt.npy: 100%  7.55k/7.55k [00:00<00:00, 487kB/s]
en_speaker_6_fine_prompt.npy: 100%  15.0k/15.0k [00:00<00:00, 1.17MB/s]
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior.
Setting `pad_token_id` to `eos_token_id`:10000 for open-end generation.
The attention mask is not set and cannot be inferred from input because pad token is same as eos token. As a consequence, you may observe unexpected behavior.
Audio saved as: audio_english_male.wav

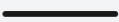



▶ 0:02 / 0:02  🔊 ⋮

Ilustración 28

```
generate_bark_audio(  
    text=traduccion8,  
    language="russian",  
    gender="male"  
)
```

ru_speaker_0_semantic_prompt.npy: 100%  4.03k/4.03k [00:00<00:00, 213kB/s]
ru_speaker_0_coarse_prompt.npy: 100%  11.9k/11.9k [00:00<00:00, 842kB/s]
ru_speaker_0_fine_prompt.npy: 100%  23.6k/23.6k [00:00<00:00, 1.08MB/s]
The attention mask and the pad token id were not set. As a consequence, you may observe unexpected behavior.
Setting `pad_token_id` to `eos_token_id`:10000 for open-end generation.
Audio saved as: audio_russian_male.wav

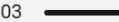
▶ 0:03 / 0:03  🔊 ⋮


Ilustración 29

Obtuvimos resultados poco naturales, tuvimos problemas con la parametrización y otros detalles. La generación de voz resultó insatisfactoria, con una tasa de **10min por cada 4seg de audio generado**, lo que resulta bastante lento.

Es evidente por la documentación que con más tiempo de pruebas unitarias y fine-tuning podemos lograr resultados mucho mejores, como asignar una emoción,

añadir risas o pausas, o elegir el género del hablante, etc.

```
[ ] generate_bark_audio(  
    text=traduccion1,  
    language="korean",  
    gender="male"  
)
```



The image shows three progress bars for audio generation prompts. Each bar is green and indicates 100% completion. The first bar is for 'ko_speaker_0_semantic_prompt.npy' with a size of 2.70k/2.70k and a speed of 115kB/s. The second bar is for 'ko_speaker_0_coarse_prompt.npy' with a size of 7.86k/7.86k and a speed of 779kB/s. The third bar is for 'ko_speaker_0_fine_prompt.npy' with a size of 15.6k/15.6k and a speed of 1.41MB/s.

Prompt Name	Progress	Size	Speed
ko_speaker_0_semantic_prompt.npy	100%	2.70k/2.70k	115kB/s
ko_speaker_0_coarse_prompt.npy	100%	7.86k/7.86k	779kB/s
ko_speaker_0_fine_prompt.npy	100%	15.6k/15.6k	1.41MB/s

2. XTTS (voice cloning)

Languages supported

- Arabic,
- Brazilian Portuguese,
- Chinese,
- Czech,
- Dutch,
- English,
- French,
- German,
- Italian,
- Polish,
- Russian,
- Spanish,
- Turkish

Our language list

- 'spanish': 'spa_Latn',
- 'urdu': 'urd_Arab',
- 'russian': 'rus_Cyrl',
- 'english': 'eng_Latn',
- 'korean': 'kor_Hang',
- 'japanese': 'jpn_Jpan',
- 'chinese': 'zho_Hans',
- 'french': 'fra_Latn',
- 'italian': 'ita_Latn',
- 'german': 'deu_Latn'

Librerías necesarias:

```
[ ] import torch.serialization
    from TTS.api import TTS
    from TTS.tts.configs.xtts_config import XttsConfig
    from TTS.config.shared_configs import BaseDatasetConfig
    from TTS.tts.models.xtts import XttsArgs

    torch.serialization.add_safe_globals([XttsConfig])
    torch.serialization.add_safe_globals([BaseDatasetConfig])
    torch.serialization.add_safe_globals([XttsArgs])

    try:
        from TTS.tts.configs.xtts_config import XttsAudioConfig
        torch.serialization.add_safe_globals([XttsAudioConfig])
    except ImportError:
        pass

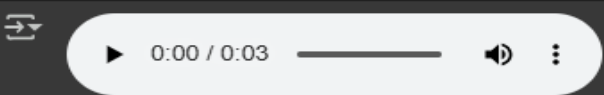
    device = "cuda" if torch.cuda.is_available() else "cpu"

    tts = TTS(model_name="tts_models/multilingual/multi-dataset/xtts_v2", progress_bar=True).to(device)
```

Ilustración 30

Prueba sin clonación de voz

```
[ ] #Prueba básica sin clonación
    from IPython.display import Audio
    tts.tts_to_file(text=traduccion_ingles,
                    file_path="ejemplo_clonacion_ingles.wav",
                    speaker_wav="voz_referencia.wav",
                    language="en")
    Audio("ejemplo_clonacion_ingles.wav")
```



```
[ ] tts.tts_to_file(
    text=traduccion_coreano,
    file_path="ejemplo_clonacion_coreano.wav",
    speaker_wav="voz_referencia.wav", # Audio de referencia para clonar
    language="ko",
)
Audio("ejemplo_clonacion_coreano.wav")
```

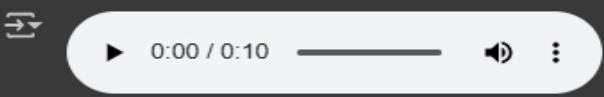


Ilustración 26

Ilustración 31

Testing

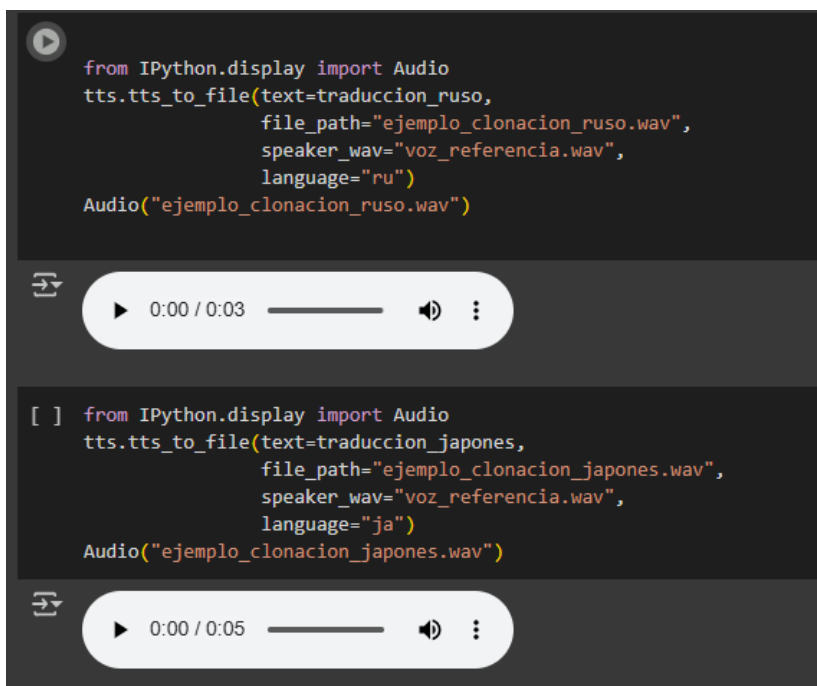


Ilustración 32

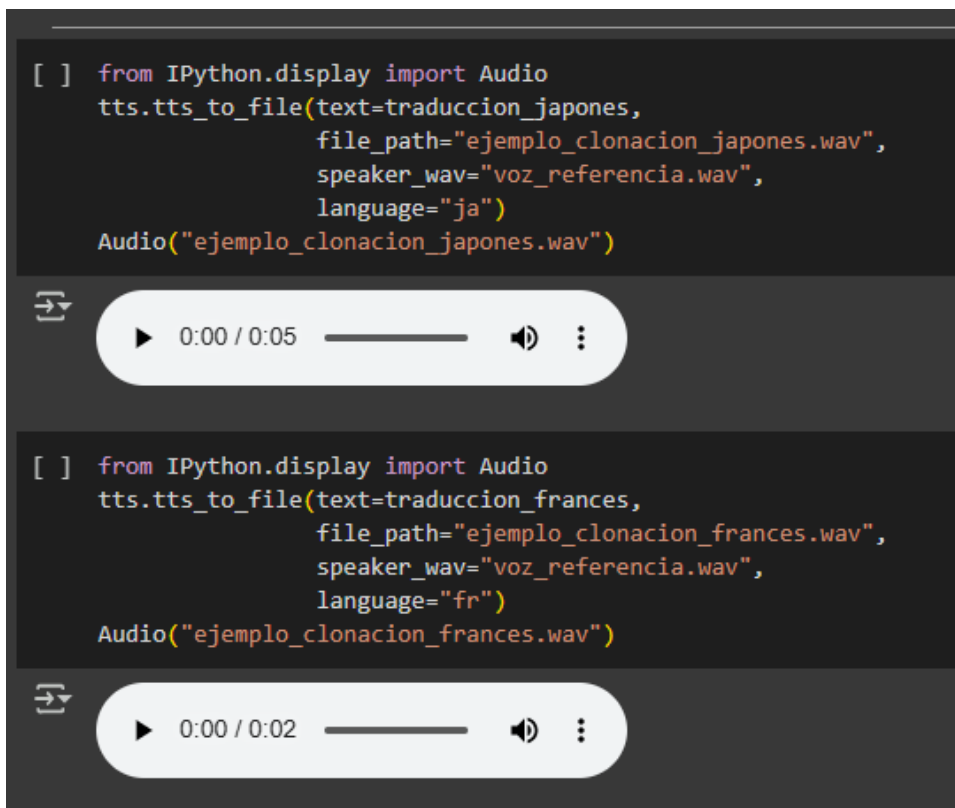


Ilustración 33

Interfaz grafica

como vemos esta es la interfaz gráfica, la cual podemos separar en 3 módulos; el primer módulo: la carga del audio ya sea grabado al momento o cargado, segundo modulo: la traducción: al momento de elegir el idioma y darle al botón de traducir el texto, se hará la traducción. Y el ultimo modulo que sería el audio generado, para esta entrega se logro entregar funcional la clonación de voz, pero

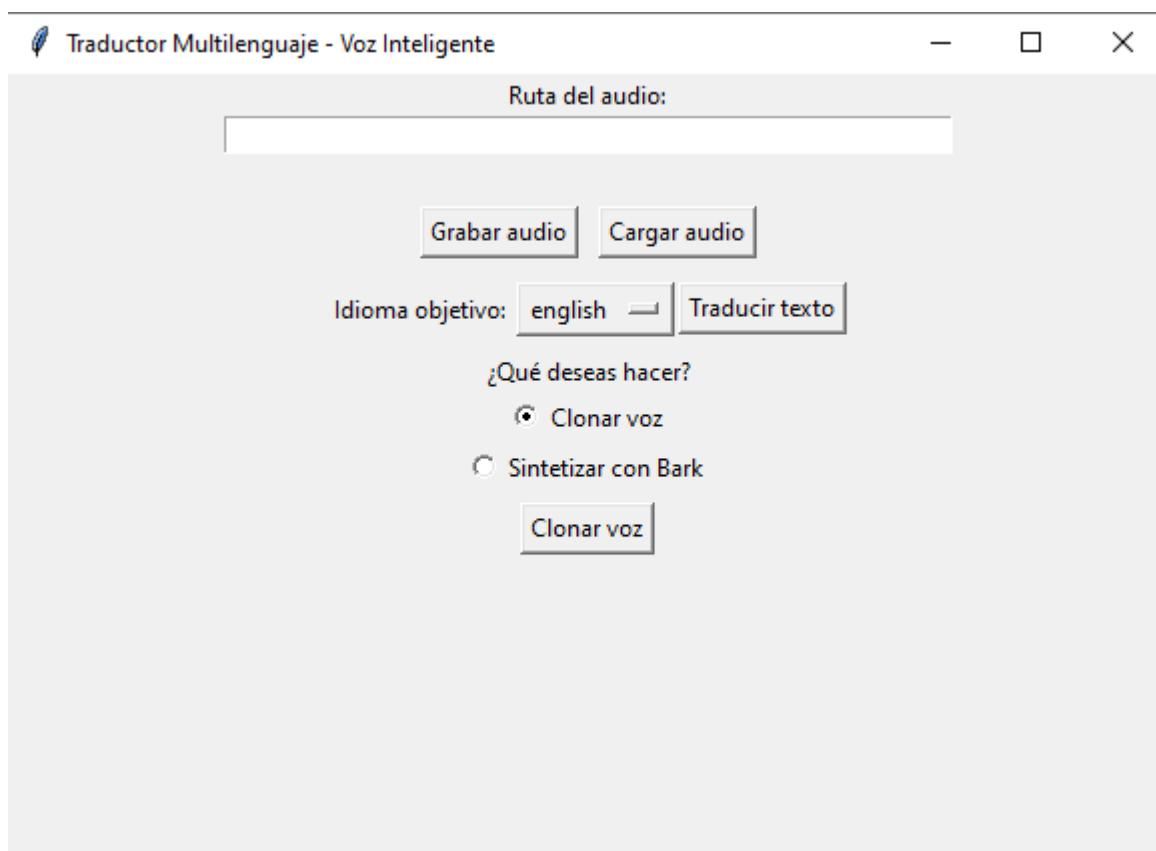


Ilustración 34

Grabando audio:

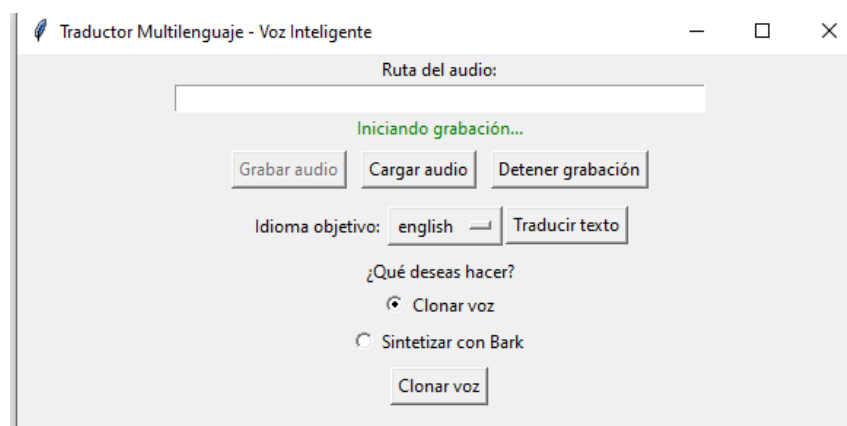


Ilustración 35

Cuando se procesa el audio, se hace la transcripción automática y detecta el idioma del audio

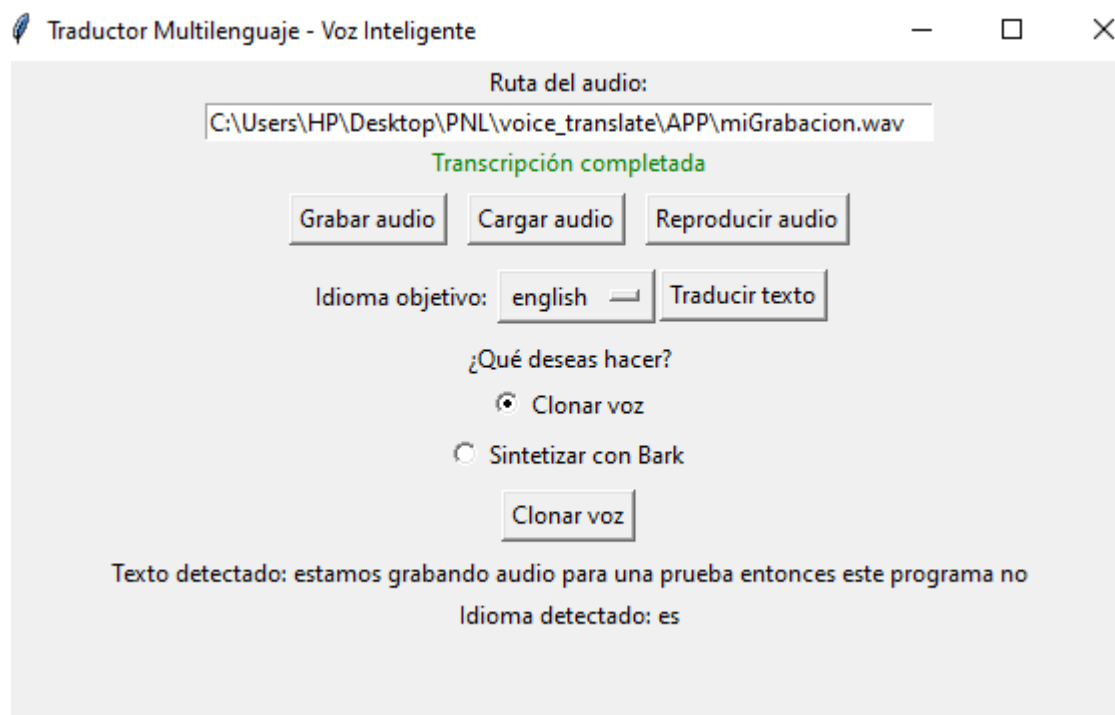


Ilustración 36

Posteriormente, elegimos el idioma y le damos al botón de traducir texto, y se va desplegar en un label la traducción del texto.

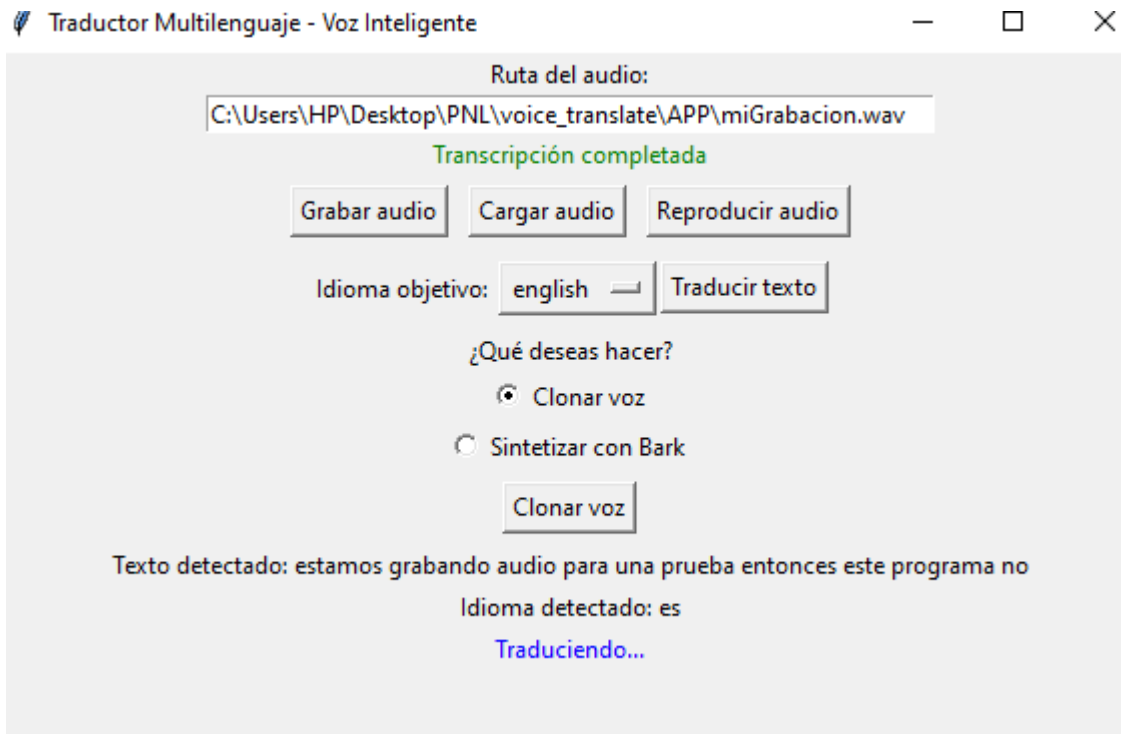


Ilustración 37

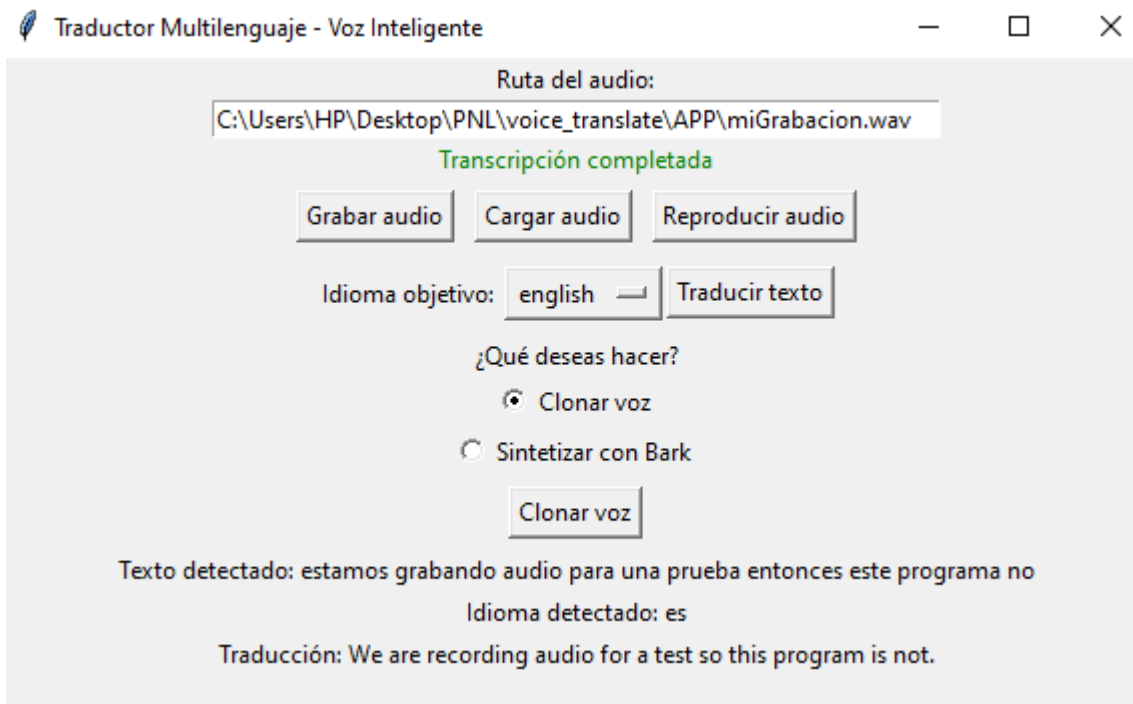


Ilustración 38

Y después en clonar voz, se nos pondrá el label en verde avisando que esta clonando la voz y después nos aparecerá un botón para escuchar el audio

generado.

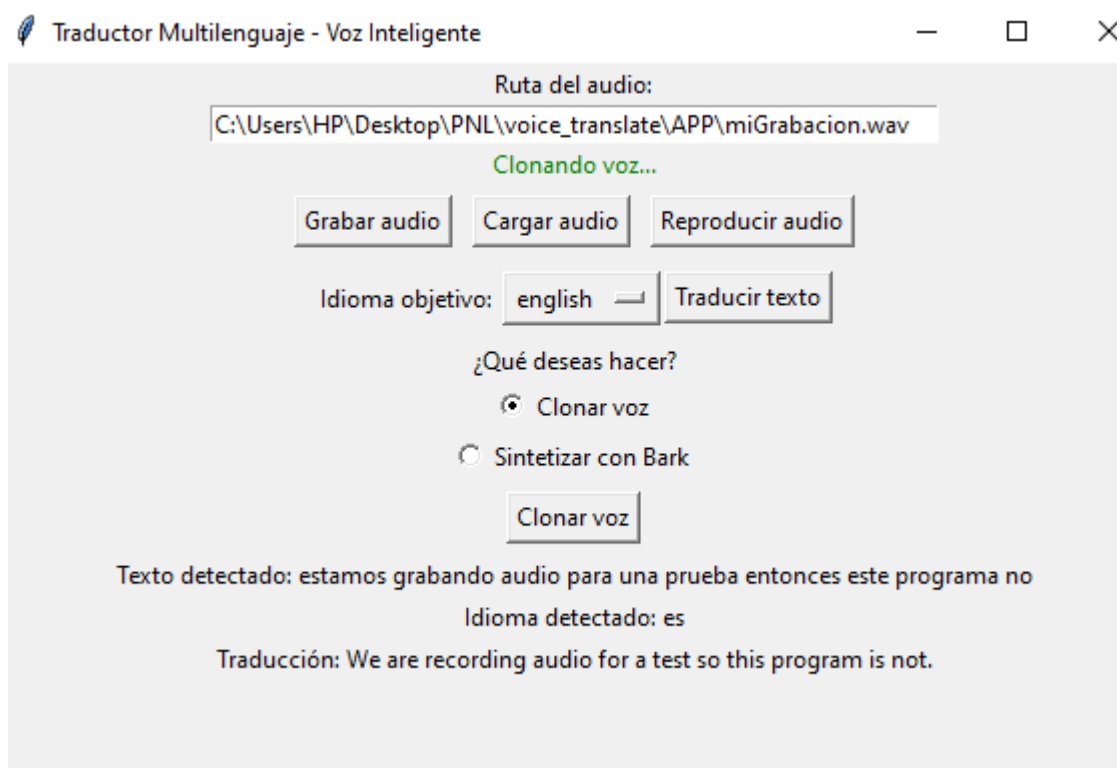


Ilustración 39

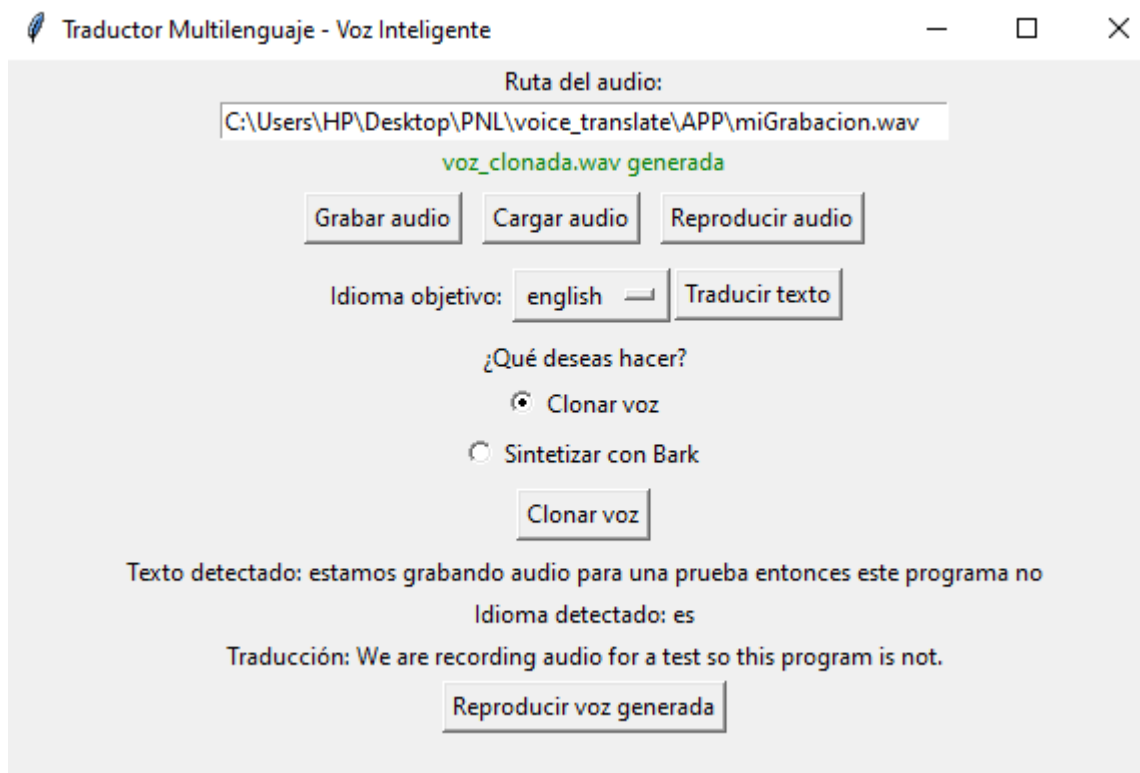


Ilustración 40

Como ya lo explicamos anteriormente, la parte de sintetizar con Bark no se logró implementar bien en la interfaz gráfica, ya que, tiene una incompatibilidad con la versión de torch, pero en las capturas de pantalla de arriba de los notebooks están los experimentos que hicimos con ellos. No obstante, decidimos implementar su código y su parte gráfica para posteriormente implementarla cuando se actualice a la versión de torch compatibles con los otros modelos.

Dicho esto, explicaremos como funciona su interfaz, le damos clic a sintetizar con Bark y se nos desplegará unos nuevos botones, para sintetizar el texto y elegir el género de la voz.

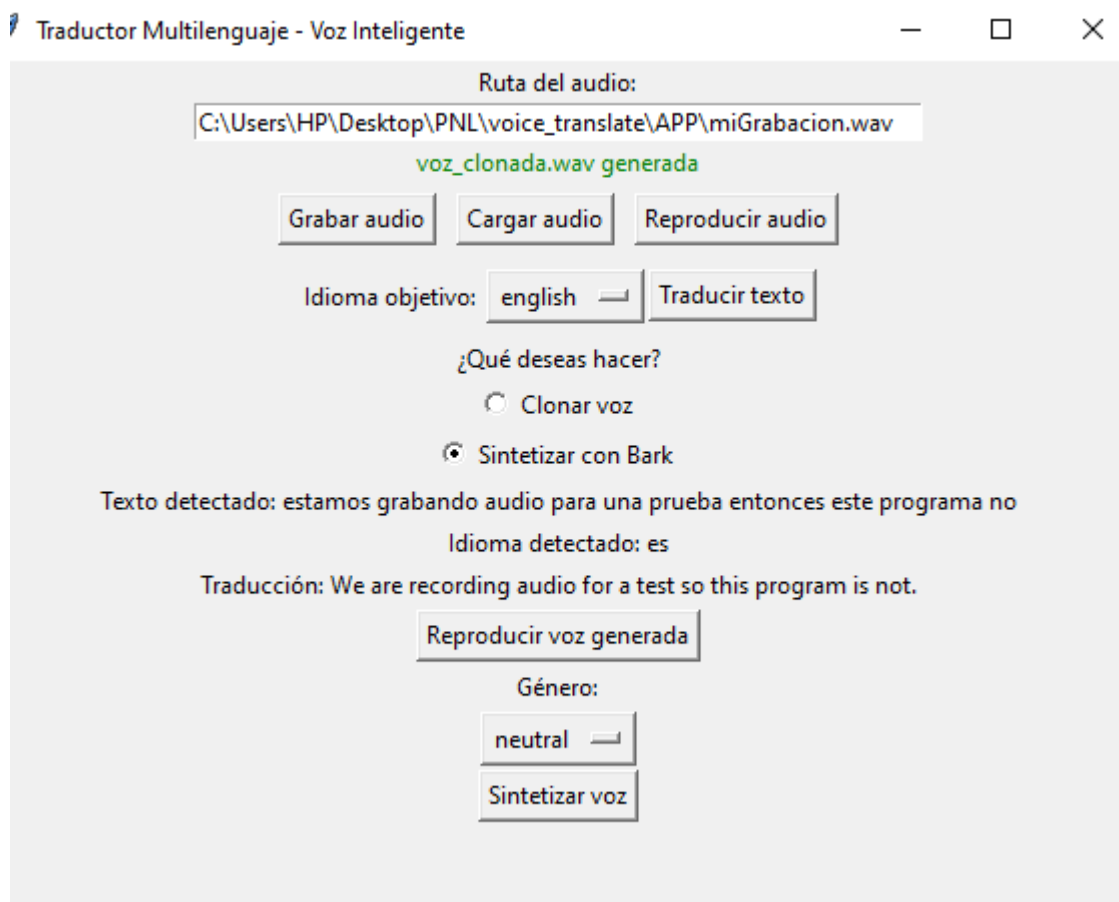


Ilustración 41

Conclusions and future work

Conclusion general:

El desarrollo de esta aplicación demostró que es posible construir una solución integral de procesamiento de lenguaje natural multilingüe con entrada y salida por voz, combinando

herramientas de última generación en una única interfaz funcional. La arquitectura propuesta permitió integrar eficientemente distintas etapas del procesamiento:

- **Captura y transcripción de voz mediante el modelo Whisper**, que se desempeñó con gran precisión en diversos idiomas y contextos, logrando un reconocimiento fluido incluso con ruido ambiental leve o pronunciación variada.
- **Traducción utilizando NLLB**, lo que nos permitió mantener una cobertura amplia de idiomas, incluyendo lenguas con bajos recursos. Sus traducciones fueron funcionales, naturales y gramaticalmente coherentes.

También fue muy interesante la implementación del módulo de generación de audio en el idioma destino utilizando dos enfoques:

- XTTS para clonación de voz, lo que añadió una capa de personalización y autenticidad.
- Bark para síntesis de voz artificial, probada en notebooks y capaz de producir resultados naturales con elección de género e idioma.

Limitaciones técnicas encontradas

Aunque Bark fue probado con éxito en notebooks independientes, no se pudo integrar directamente a la interfaz de usuario, debido a incompatibilidades con la versión de torch instalada en el entorno local. Esta función aparece en la interfaz, pero al activarla, se produce un error que impide su ejecución. Esta limitación es tratada como un punto de mejora.

Otra limitante fue, que algunos modelos, como Whisper, consumen una cantidad considerable de memoria, lo que limita su uso en dispositivos con pocos recursos o sin aceleración por GPU, especialmente extrañada en nuestros experimentos de síntesis de voz parametrizada como Bark.

Por otro lado, la interfaz fue construida en Tkinter, lo cual es suficiente para una demostración local, pero no ideal para aplicaciones web o móviles, limitando el alcance inmediato del sistema.

Además, en la fase de traducción factores de emotividad, frases subjetivas como las encontradas en canciones y otros experimentos de tono de dicción pueden hacer que incluso un modelo poderoso y contextualizado como NLLB pueda resultar erróneo.

Limitaciones de tiempo nos impidieron realizar más pruebas unitarias de parametrización con el modelo Bark, lo que resultó en audios poco naturales y lentos de generar.

Trabajo futuro propuesto:

Para mejorar y escalar esta solución, proponemos los siguientes desarrollos:

- Actualización del entorno de ejecución para integrar completamente el modelo Bark a la interfaz, ajustando la versión de torch y estabilizando su implementación, a si mismo, agregarle a la interfaz de “sintetizar con bark” una opción de emociones.
- Un ajuste de parámetros o un procesamiento especializado en interpretación de emociones del texto y contextos culturales para el área musical por ejemplo.
- Optimización de recursos computacionales, utilizando versiones reducidas o cuantizadas de los modelos, para permitir su ejecución en tiempo real en equipos sin GPU.
- Inclusión de detección de emociones o contexto, para mejorar la naturalidad de la voz sintetizada o personalizada.