[Repositorio oficial]

Instalación

```
Unset
pip install -U openai-whisper
```

Dependencias

```
Unset
# on Ubuntu or Debian
sudo apt update && sudo apt install ffmpeg

# on Windows using Chocolatey (https://chocolatey.org/)
choco install ffmpeg

# on Windows using Scoop (https://scoop.sh/)
scoop install ffmpeg

# Otra dependencia
pip install setuptools-rust
```

Available models and languages

There are six model sizes, four with English-only versions, offering speed and accuracy tradeoffs. Below are the names of the available models and their approximate memory requirements and inference speed relative to the large model. The relative speeds below are measured by transcribing English speech on a A100, and the real-world speed may vary significantly depending on many factors including the language, the speaking speed, and the available hardware.

Size	Parameters	English-only model	Multilingual model	Required VRAM	Relative speed
tiny	39 M	`tiny.en`	`tiny`	~1 GB	~10x
base	74 M	`base.en`	`base`	~1 GB	~7x
small	244 M	`small.en`	`small`	~2 GB	~4x
medium	769 M	`medium.en`	`medium`	~5 GB	~2x
large	1550 M	N/A	`large`	~10 GB	1x
turbo	809 M	N/A	`turbo`	~6 GB	~8x

El MSI GP66 Leopard 10UE está equipado con una tarjeta gráfica NVIDIA GeForce RTX 3060 Laptop GPU que cuenta con 6 GB de VRAM GDDR6 por lo que habría que utilizar el modo “turbo” o “medium”

Cómo usar

```
Unset
# Ejemplo 1 (Inglés por defecto)
whisper audio.flac audio.mp3 audio.wav --model turbo

# Elegir idioma
whisper japanese.wav --language Japanese

# Traducir idioma
whisper japanese.wav --language Japanese --task translate

# Help
whisper --help

# See tokenizer.py for the list of all available languages.
```

Python usage (Importante usar GPU)

```
Unset
import whisper

model = whisper.load_model("turbo")
result = model.transcribe("audio.mp3")
# result = model.transcribe("audio.mp3", task="translate")

print(result["text"])
```

Internamente, el método transcribe() lee todo el archivo y procesa el audio con una ventana deslizante de 30 segundos, realizando predicciones autorregresivas de secuencia a secuencia en cada ventana.

A continuación, se muestra un ejemplo del uso de whisper.detect_language() y whisper.decode(), que proporcionan un acceso de nivel inferior al modelo.

```
Unset
import whisper

model = whisper.load_model("turbo")

# load audio and pad/trim it to fit 30 seconds
audio = whisper.load_audio("audio.mp3")
audio = whisper.pad_or_trim(audio)

# make log-Mel spectrogram and move to the same device as the model
mel = whisper.log_mel_spectrogram(audio, n_mels=model.dims.n_mels).to(model.device)

# detect the spoken language
_, probs = model.detect_language(mel)
print(f"Detected language: {max(probs, key=probs.get)}")

# decode the audio
options = whisper.DecodingOptions()
result = whisper.decode(model, mel, options)

# print the recognized text
print(result.text)
```

The following script transcribes an audio file using OpenAI's Whisper model and organizes the text into a structured PDF. It segments the transcription into 5-minute intervals, adding time markers and formatting the text for readability.

```
Unset
import whisper
import torch
from reportlab.lib.pagesizes import A4
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer

# Verificar si CUDA está disponible
print("CUDA available:", torch.cuda.is_available())
print("GPU Name:", torch.cuda.get_device_name(0) if torch.cuda.is_available() else "No GPU
detected")

# Cargar el modelo Whisper
model = whisper.load_model("turbo")
result = model.transcribe("Vision7.mp3", word_timestamps=True)  # Activar marcas de tiempo

# Configurar el PDF con hoja A4
pdf_filename = "Vision_27_02_2025.pdf"
```

```
doc = SimpleDocTemplate(pdf_filename, pagesize=A4, leftMargin=50, rightMargin=50,
topMargin=50, bottomMargin=50)

# Crear estilos de texto (justificado y para los indicadores de minutos)
styles = getSampleStyleSheet()
styles.add(ParagraphStyle(name="Justify", alignment=4))  # Justificar el texto
styles.add(ParagraphStyle(name="TimeMarker", fontSize=14, spaceBefore=10, spaceAfter=5,
textColor="blue", bold=True))

# Obtener la transcripción con timestamps
segments = result["segments"]

# Organizar el texto en bloques de 5 minutos
content = []
current_time = 0
text_block = ""

for segment in segments:
   start_time = segment["start"]  # Tiempo de inicio en segundos
   text = segment["text"]

   # Si han pasado 5 minutos, agregar un indicador y reiniciar el bloque de texto
   if start_time >= current_time:
       if text_block:
           content.append(Paragraph(text_block, styles["Justify"]))
           content.append(Spacer(1, 12))  # Espaciado entre bloques

       # Agregar marcador de minuto
       minutes = int(current_time // 60)
       content.append(Paragraph(f"🕒 Minuto {minutes}", styles["TimeMarker"]))
       content.append(Spacer(1, 8))

       text_block = ""  # Reiniciar bloque de texto
       current_time += 300  # Avanzar 5 minutos

   text_block += text + " "

# Agregar el último bloque de texto al contenido
if text_block:
   content.append(Paragraph(text_block, styles["Justify"]))

# Generar el PDF
doc.build(content)
print(f"Transcription saved as {pdf_filename}")
```