WHISPER (Hugging Face)



Adrián Yared Armas de la Nuez

**Contenido**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

[**1. Objetivo 2**](#_2c35568x6wg8)

[**2. Resolución de la actividad 2**](#_bxwib9khtdww)

[**2.1 Instalación 2**](#_t8cwbo6n4466)

[**2.1.1 Torch 2**](#_zhg4jaty4h1i)

[**2.1.2 Transformers 2**](#_xhqn4dwz98ao)

[**2.1.3 Ffmpeg 2**](#_crxutfoja69v)

[**2.1.4 Whisper 2**](#_8u9z6n841hmv)

[**2.1.5 Accelerate 2**](#_myqozxq8e91v)

[**2.1.6 Librosa 2**](#_v7kvtv2e497w)

[**2.2 Error y solución 2**](#_392qqtey01ra)

[**2.2.1 ¿Qué es Chocolatey? 3**](#_lma3y8i87sld)

[**2.2.2 Descarga de Chocolatey 3**](#_6f6cys9pqakc)

[**2.2.3 Descarga Ffmpeg a través de Chocolatey 3**](#_ckbxnsndnw9p)

[**2.3 Selección y descarga del audio 3**](#_mfkugwt6i3p)

[**2.3.1 Descarga 4**](#_aig92oe7k2m0)

[**2.3.2 Añado el audio al entorno de trabajo 4**](#_b3syprlzujbl)

[**2.4 Uso del modelo whisper 4**](#_thesbsc8fn2i)

[**2.4.1 Instalación 4**](#_mqg2kyleknq)

[**2.4.2 Código base 4**](#_d0o3lu4bs98i)

[**2.5 Código final y explicación 5**](#_gmu75pp9nqxc)

[**2.5.1 Imports 5**](#_g50nm4vhtuug)

[**2.5.2 Configuración de dispositivo 6**](#_un82j3ezow2i)

[**2.5.3 Carga del modelo 6**](#_1mtv216773z7)

[**2.5.4 Procesador 6**](#_ebf4hlqoyly4)

[**2.5.5 Pipeline 6**](#_b58w7c8fzpwc)

[**2.5.6 Transcripción 6**](#_jxcqsnepzb8d)

[**2.5.7 Formateo 7**](#_qpfuwyjabv48)

[**2.5.8 Muestra resultado 7**](#_s6pzcj2bbuo1)

[**2.6 Código de comparación 7**](#_b2fu96q15se1)

[**2.6.1 Normalización 7**](#_wpcels4iagjs)

[**2.6.2 Texto a comparar 8**](#_d8151va7ni9h)

[**2.6.3 Lectura del archivo txt 8**](#_yg46ujumio49)

[**2.6.4 Normalización de los textos 9**](#_7lmilxuswa7m)

[**2.6.5 Comparar textos 9**](#_s3qebnm5md1y)

[**2.6.5.1 Código 9**](#_1wzt3nsqnq2s)

[**2.6.5.2 Resultado 9**](#_l9wk39rl4ysz)

[**2.6.5.3 Calculo de error 9**](#_396x9xhnqige)

[**2.6.5.4 Muestra similitud y error 9**](#_280m1jxn4ix)

## 

## 

## 

## 

## 

## 

## 

## 

## **1. Objetivo**

El propósito de esta tarea es que el alumnado se familiarice con el modelo Whisper de

OpenAl, disponible en Hugging Face, y lo utilice para transcribir el audio de una canción

descargada. Se espera que el alumnado realice un análisis crítico del proceso, documente

los pasos seguidos y proponga mejoras o dificultades encontradas.

## **2. Resolución de la actividad**

### **2.1 Instalación**

#### **2.1.1 Torch**



#### **2.1.2 Transformers**



#### **2.1.3 Ffmpeg**



#### **2.1.4 Whisper**



#### **2.1.5 Accelerate**

Requiere la dependencia accelerate para otras librerías anteriores.



#### **2.1.6 Librosa**

Requiere la dependencia librosa para otras librerías anteriores.



### **2.2 Error y solución**

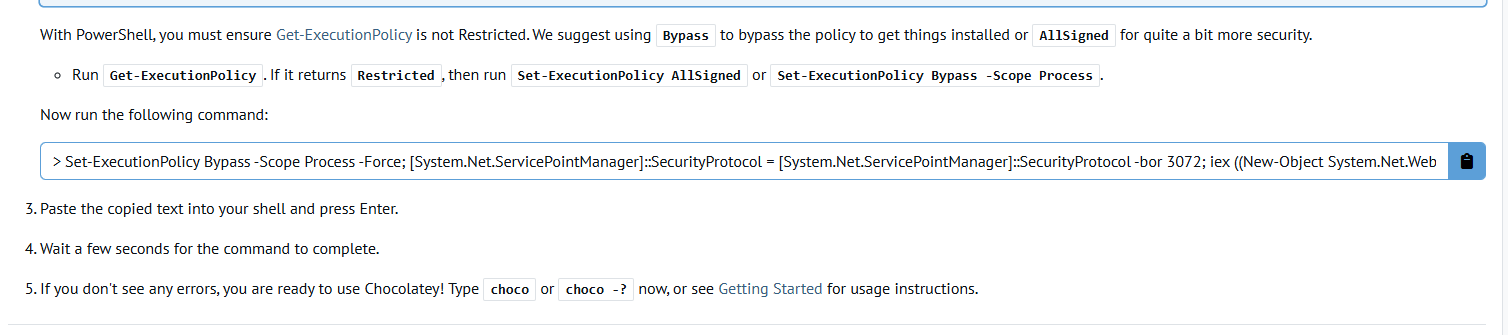
A la hora de la ejecución del código me decía que ffmpeg seguía sin estar instalado correctamente, por lo que Adolfo me recomendó la instalación de Chocolatey:  
<https://chocolatey.org/install>

#### **2.2.1 ¿Qué es Chocolatey?**

Chocolatey es un administrador de paquetes para Windows que permite instalar, actualizar y gestionar software desde la línea de comandos de forma rápida y automatizada.

#### **2.2.2 Descarga de Chocolatey**

Comando en la web:



Ejecución en la consola:



#### **2.2.3 Descarga Ffmpeg a través de Chocolatey**



### **2.3 Selección y descarga del audio**

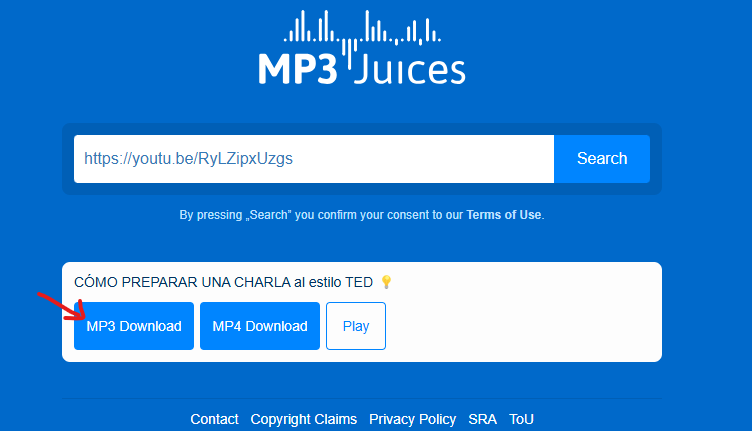
Video de la descarga:

<https://youtu.be/RyLZipxUzgs>

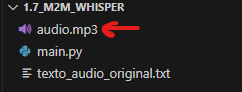


#### **2.3.1 Descarga**

A través de la web <https://emp3juice.la/> descargué el audio del video:



#### **2.3.2 Añado el audio al entorno de trabajo**



### **2.4 Uso del modelo whisper**

#### **2.4.1 Instalación**

Previamente hice la instalación del modelo:



#### **2.4.2 Código base**

A través de la página del modelo, <https://huggingface.co/openai/whisper-large-v3> , he obtenido el código base de funcionamiento del modelo:  
import torch

from transformers import AutoModelForSpeechSeq2Seq, AutoProcessor, pipeline

from datasets import load\_dataset

device = "cuda:0" if torch.cuda.is\_available() else "cpu"

torch\_dtype = torch.float16 if torch.cuda.is\_available() else torch.float32

model\_id = "openai/whisper-large-v3"

model = AutoModelForSpeechSeq2Seq.from\_pretrained(

model\_id, torch\_dtype=torch\_dtype, low\_cpu\_mem\_usage=True, use\_safetensors=True

)

model.to(device)

processor = AutoProcessor.from\_pretrained(model\_id)

pipe = pipeline(

"automatic-speech-recognition",

model=model,

tokenizer=processor.tokenizer,

feature\_extractor=processor.feature\_extractor,

torch\_dtype=torch\_dtype,

device=device,

)

dataset = load\_dataset("distil-whisper/librispeech\_long", "clean", split="validation")

sample = dataset[0]["audio"]

result = pipe(sample)

print(result["text"])

### **2.5 Código final y explicación**

#### **2.5.1 Imports**

import torch

import unicodedata # Manejo de Unicode

import re # Manipulación de texto

from transformers import AutoModelForSpeechSeq2Seq, AutoProcessor, pipeline # Herramientas de Hugging Face

from difflib import SequenceMatcher # Comparar secuencias y calcular la similitud

#### **2.5.2 Configuración de dispositivo**

Selecciona la GPU:

device = "cuda:0" if torch.cuda.is\_available() else "cpu"

torch\_dtype = torch.float16 if torch.cuda.is\_available() else torch.float32

#### **2.5.3 Carga del modelo**

Carga el modelo preentrenado whisper-large-v3 de OpenAI con el tipo de datos adecuado, optimiza el uso de memoria en CPU y lo envía al dispositivo seleccionado (En mi caso la CPU):

# Cargar modelo

model\_id = "openai/whisper-large-v3"

model = AutoModelForSpeechSeq2Seq.from\_pretrained(

model\_id, torch\_dtype=torch\_dtype, low\_cpu\_mem\_usage=True, use\_safetensors=True

)

model.to(device)

#### **2.5.4 Procesador**

Carga los datos de entrada y salida:

processor = AutoProcessor.from\_pretrained(model\_id)

#### **2.5.5 Pipeline**

Crea un pipeline usando el modelo cargado y las características:

# Pipeline de reconocimiento de voz

pipe = pipeline(

"automatic-speech-recognition",

model=model,

tokenizer=processor.tokenizer,

feature\_extractor=processor.feature\_extractor,

torch\_dtype=torch\_dtype,

device=device,

)

#### **2.5.6 Transcripción**

Usa el Pipe para convertirlo a texto:

# Transcripción del audio

result = pipe("audio.mp3", return\_timestamps=True)

texto = result["text"]

#### **2.5.7 Formateo**

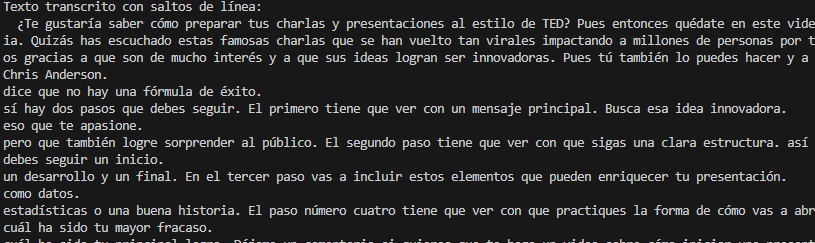
Formateo los saltos de línea para no mostrar todo en una línea:

# Formatear el texto con saltos de línea

texto\_con\_saltos = texto.replace(", ", ".\n")

print("Texto transcrito con saltos de línea:\n", texto\_con\_saltos)

#### **2.5.8 Muestra resultado**

Muestra del resultado formateado:  


### **2.6 Código de comparación**

#### **2.6.1 Normalización**

Para comparar los textos de manera más exacta, creé el código que normaliza el texto (quité espacios, párrafos, convertí todo a minúscula y quite los signos de puntuación):

# Normalización del texto

def normalizar(texto):

texto = unicodedata.normalize("NFD", texto) # Descomponer caracteres

texto = texto.encode("ascii", "ignore").decode("utf-8") # Eliminar tildes

texto = re.sub(r"[^\w\s]", "", texto) # Quitar signos de puntuación

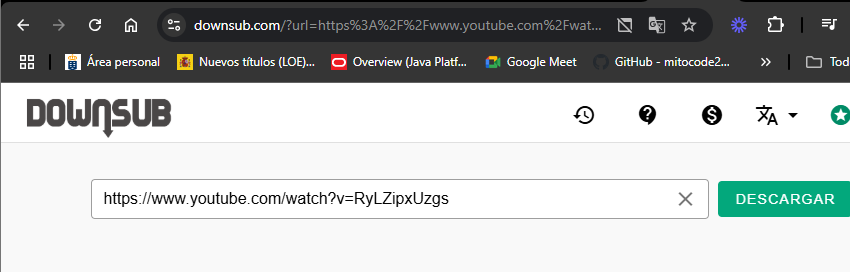
texto = texto.lower() # Convertir a minúsculas

texto = re.sub(r"\s+", "", texto) # Eliminar todos los espacios y saltos de línea

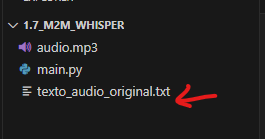
return texto

#### **2.6.2 Texto a comparar**

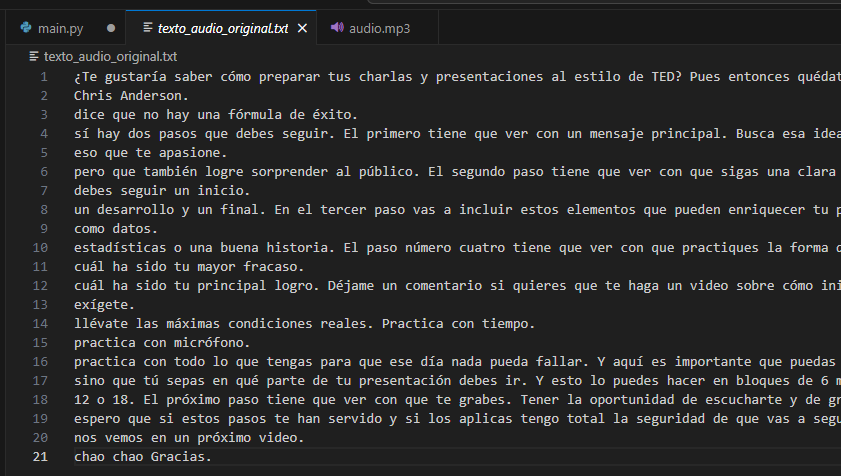
Puse el video en una página para descargar los subtítulos y tener una base:



Pero ya que esta web usará un modelo parecido y no se trata de comparar modelos, comparé con los subtítulos de youtube. Y corregí eso manualmente, y lo añadí al proyecto.



Resultado:



#### **2.6.3 Lectura del archivo txt**

# Leer y normalizar el texto original desde un archivo

with open("texto\_audio\_original.txt", "r", encoding="utf-8") as file:

texto\_original = file.read()

#### **2.6.4 Normalización de los textos**

Uso la función de normalizado que había creado anteriormente y le paso ambos textos, el original y el generado:

texto\_normalizado = normalizar(texto)

texto\_normalizado\_original = normalizar(texto\_original)

#### **2.6.5 Comparar textos**

Paso ambas cadenas de texto, y uso la librería SequenceMatcher (la cual compara dos cadenas de texto) para obtener el porcentaje de similitud

##### **2.6.5.1 Código**

# Comparar los textos

def calcular\_similitud(texto1, texto2):

matcher = SequenceMatcher(None, texto1, texto2)

return matcher.ratio() \* 100 # Convertir a porcentaje

##### **2.6.5.2 Resultado**

##### **2.6.5.3 Calculo de error**

Calcula el error restando al 100% el porcentaje de acierto:

similitud = calcular\_similitud(texto\_normalizado, texto\_normalizado\_original)

error = 100 - similitud

##### **2.6.5.4 Muestra similitud y error**

Print del resultado en modo porcentual:

# Mostrar resultados

print(f"Porcentaje de similitud: {similitud:.2f}%")

print(f"Porcentaje de error: {error:.2f}%")

El porcentaje de acierto es extremadamente alto debido a la claridad y pausas entre palabras, así como la ausencia de ruido de entorno, como podría ser en el caso de la música de fondo en una canción.

