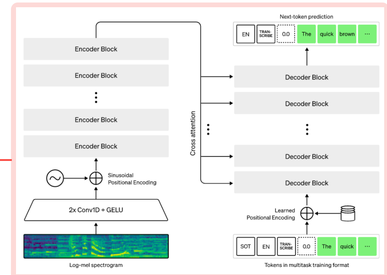
# Introduction about Whisper

## Introduction:




Whisper is **an encoder-decoder auto-regressive model** which was trained on audio translation and transcription tasks.

Given audio data, the model is **able to generate the corresponding text.**

This notebook will allow you to play around with the model and evaluate it in a few lines of code. If you **want more details about Whisper**, checkout each 'model card' (see for example `['whisper-tiny']` (https://huggingface.co/openai/whisper-tiny) ) or the [original paper] (https://openai.com/blog/whisper/).

### Encoder

A log-mel spectrogram **is extracted from a raw audio using a [`Processor`]** (https://huggingface.co/docs/transformers/main_classes/processors), before it is passed to the encoder.

### Decoder

The decoder inputs are **text tokens**, and **special tokens** such as **"<|startoftranscript|>"**, **"<|transcribe|>"** and **"<|en|>"** are used to specify the desired task and the language of the audio.

Two different models were trained, an English only and a multilingual model. English only checkpoints have the `.en` suffix on the Hub and use a different tokenizer vocabulary.

## package installation

Installing Whisper

The commands below will install the Python packages needed to use Whisper models and evaluate the transcription results.

```
[ ] %%capture
    !pip install git+https://github.com/huggingface/transformers.git
    !pip install datasets
    !pip install evaluate
    !wget https://cdn-media.huggingface.co/peech_samples/sample1.flac
    !wget https://huggingface.co/datasets/osanseviero/dummy_ja_audio/resolve/main/result.flac
    !pip install gradio
    !pip install jiwer
```

## Loading the model and the processor

```
import torch
from transformers import WhisperForConditionalGeneration, WhisperProcessor

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

model = WhisperForConditionalGeneration.from_pretrained("openai/whisper-small").to(device)
processor = WhisperProcessor.from_pretrained("openai/whisper-small")
```

load the model and processor

## Simple Inference

Let's grab a single audio file in English from a dataset and let's see how well it works!

```
[ ] from datasets import load_dataset
    ds = load_dataset("hf-internal-testing/librispeech_asr_dummy", "clean", split="validation")
```

load the dataset

```
audio_sample = ds[3]

text = audio_sample["text"].lower()
speech_data = audio_sample["audio"]["array"]
speech_file = audio_sample["file"]

text
```

```
'he has grave doubts whether sir frederick leighton's work is really greek after all and can discover in it but little of rocky ithaca'
```

```
[ ] from IPython.display import Audio, display

display(Audio(speech_file))
```

▶ 0:00 / 0:09

display the audio player to play the sounds

### inference process

1. use the **processor feature extractor** to pre-process the data
2. Use the **model.generate** method to **generate the transcript**
3. Use the **processor tokenizer to decode the ids** generated by the model (that is, convert the ids to their corresponding tokens). This can do some additional things such as normalizing (e.g. remove capitalization)

```
inputs = processor.feature_extractor(speech_data, return_tensors="pt", sampling_rate=16_000).input_features.to(device)
predicted_ids = model.generate(inputs, max_length=480_000)
processor.tokenizer.batch_decode(predicted_ids, skip_special_tokens=True, normalize=True)[0]
```

```
' he has grave doubts whether sir frederick layton is work is really greek after all and can discover in it but little of rocky ithaca'
```

exactly the same as the ground truth

## Step by Step

### 1. audio is represented as an array

```
[ ] print(len(speech_data), speech_data)

    150480 [-0.00045776 -0.00039673 -0.00048828 ... -0.00021362 -0.00018311
     -0.00027466]
```

### 2. Using the **feature extractor**, we **pre-process the audio** to a format usable by the model (i.e. extract the **log-mel spectrogram from the audio**).

```
inputs = processor.feature_extractor(speech_data, return_tensors="pt", sampling_rate=16_000).input_features.to(device)
inputs

tensor([[[-0.3468, -0.0435, -0.0998,  ..., -0.8371, -0.8371, -0.8371],
         [ 0.8187, -0.0464, -0.8371,  ..., -0.8371, -0.8371, -0.8371],
         [ 0.8028, -0.2122, -0.2224,  ..., -0.8371, -0.8371, -0.8371],
         ...,
         [-0.8371, -0.8371, -0.8371,  ..., -0.8371, -0.8371, -0.8371],
         [-0.8371, -0.8371, -0.8371,  ..., -0.8371, -0.8371, -0.8371],
         [-0.8371, -0.8371, -0.8371,  ..., -0.8371, -0.8371, -0.8371]]],
       device='cuda:0')
```

```
print(inputs.shape)

torch.Size([1, 80, 3000])
```

**Calculation Example**

Suppose the audio is sampled at 16,000 Hz and has a duration of 30 seconds. The total number of samples $N$ would be:

$N = 16,000 \times 30 = 480,000 \text{ samples}$

**Before Log-Mel Spectrogram**

Before creating the log-Mel spectrogram, the raw audio typically has the following dimensions:

1. **Channels**: Audio can have 1 (mono) or 2 (stereo) channels.
2. **Samples**: The length of the audio in terms of samples.

For example, a stereo audio signal with 2 channels and $N$ samples would have a shape of $[2, N]$. A mono audio signal with $N$ samples would have a shape of $[N]$ or $[1, N]$.

**After Log-Mel Spectrogram**

The log-Mel spectrogram has the following dimensions:

1. **Frequency Bins**: This represents the number of Mel filter banks used. In Whisper, this is typically set to 80.
2. **Time Frames**: This depends on the length of the audio signal and the hop length used when creating the Mel spectrogram.

Now, if we use an FFT size $n\_fft = 400$ and a hop length $hop\_length = 160$ for the Mel spectrogram:

$$\text{Time Frames} = \left\lfloor \frac{480,000 - 400}{160} \right\rfloor + 1 = 2999$$

Thus, the resulting log-Mel spectrogram would have a shape of:

$$\text{Shape} = [80, 2999]$$

FFT means Fast Fourier Transform (an advanced and quite complex concept in signal processing, **optional to understand)**

### 3. We pass this. input to the model generate methods

```
predicted_ids = model.generate(inputs, max_length=480_000)
predicted_ids

tensor([[50258, 50259, 50359, 50363,   634,   575, 12525, 22618,  1968,  6144,
           35617, 20908,   756,   311,   589,   307,   534, 10281,   934,   439,
           293,   393,  4411,   294,   309,   457,   787,   295, 33368,   286,
           392,  6628,    13, 50257]], device='cuda:0')
```

model predict the text

### 4. Tokenizer decodes the ids to actual readable text

```
processor.tokenizer.batch_decode(predicted_ids)

['<|startoftranscript|><|en|><|transcribe|><|notimestamps|> He has grave doubts whether Sir Frederick Layton's work is really Greek after all and can discover in it but little of rocky Ithaca.<|endoftext|>']
```

decode and normalize to a sentence

```
[ ] processor.tokenizer.batch_decode(predicted_ids, skip_special_tokens=True)[0]

    ' He has grave doubts whether Sir Frederick Layton's work is really Greek after all and can discover in it but little of rocky Ithaca.'
```

get rid of special tokens

## Too much code!

```
/usr/local/lib/python3.10/dist-packages/transformers/generation/utils.py:1346: UserWarning: Using `max_length`'s default (448) to control the gene
  warnings.warn(
{'text': " He has grave doubts whether Sir Frederick Layton's work is really Greek after all and can discover in it but little of rocky Ithaca."}
```

just two lines of code and done!

```
[ ] from transformers import pipeline

    pipe = pipeline("automatic-speech-recognition", model="openai/whisper-small")
    pipe(speech_file)
```

pipeline is a very convenient tool provided by OpenAI

```
from transformers import pipeline
import gradio as gr

pipe = pipeline("automatic-speech-recognition", model="openai/whisper-small")

def inference(speech_file):
  return pipe(speech_file)["text"]

gr.Interface(inference,gr.Audio(type="filepath"),"text").launch()
```

```
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
Note: opening Chrome Inspector may crash demo inside Colab notebooks.

To create a public link, set `share=True` in `launch()`.
```

gradio is a visualization tool

## Inference in Other Languages

```
from datasets import load_dataset

sample = load_dataset("osanseviero/dummy_ja_audio")["train"]["audio"][0]
speech_data = sample["array"]
```

identify text and translate it to other languages at the same time

```
inputs = processor.feature_extractor(speech_data, return_tensors="pt", sampling_rate=16_000).input_features.to(device)
forced_decoder_ids = processor.get_decoder_prompt_ids(language="ja", task="translate")
predicted_ids = model.generate(inputs, max_length=480_000, forced_decoder_ids=forced_decoder_ids)
processor.tokenizer.batch_decode(predicted_ids, skip_special_tokens=True, normalize=True)[0]
```

```
' キュクロスに監視されているらしい'
```

processor to extract the feature
set up forced_decoder_id --> this can force the model to generate japanese characters

Presented with xmind