FOSTERLANG

Fostering Linguistic Capital: A Roadmap for Reversing the Diversity Crisis and Activating Societal Benefits in Europe

Instructional training materials: Training materials from capacity-building Workshop

Deliverable D5.3

Grant Agreement No.: 101178387

Hands on session

1. **ASR demonstrations for Basque and Spanish using Whisper and Wav2Vec**
2. **Fine-Tuning Wav2Vec BERT for Low Resource Mongolian Language**

Prepared by: Dr. Arvind Kumar

Team WP5, MPI

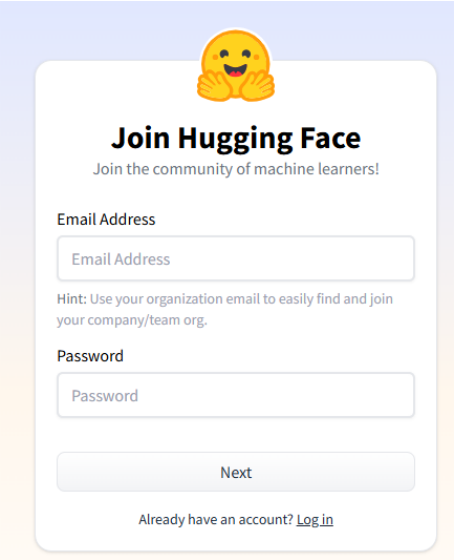
Funded by the European Union

Introduction:

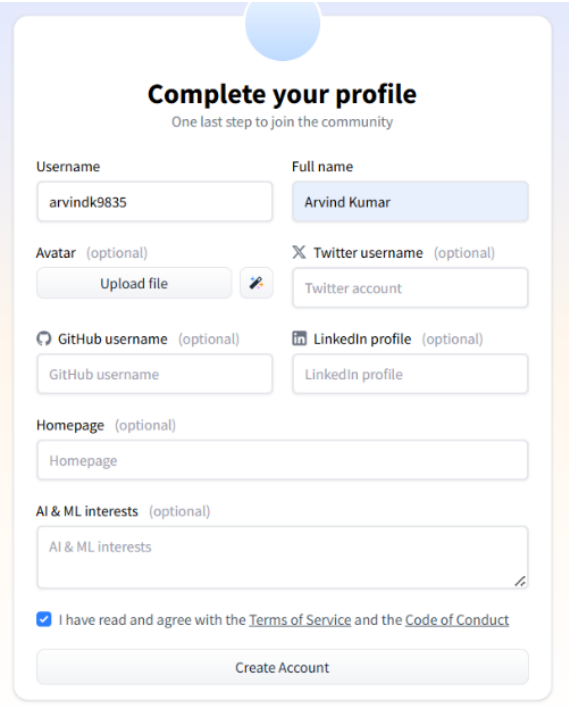
Wav2Vec2-BERT by MetaAI is the result of a series of improvements based on an original model: Wav2Vec2, a pre-trained model for Automatic Speech Recognition (ASR) released in September 2020 by Alexei Baevski, Michael Auli, and Alex Conneau. With as little as 10 minutes of labeled audio data, Wav2Vec2 could be fine-tuned to achieve 5% word-error rate performance on the LibriSpeech dataset, demonstrating for the first time low-resource transfer learning for ASR. In this exercise, we will be using Mongolian language as an example to fine-tune a model.

Section 1: Creating account on Hugging Face

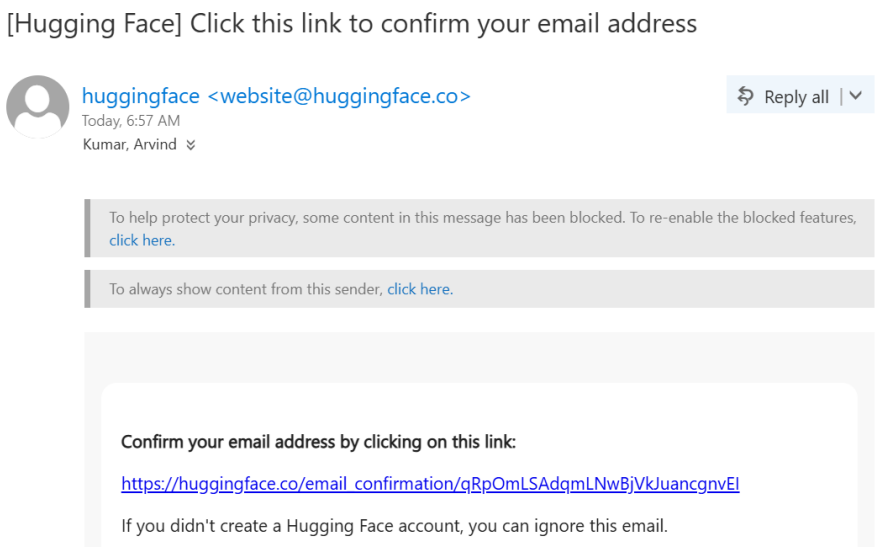
Step 1: Visit https://huggingface.co/join and enter your email id and password



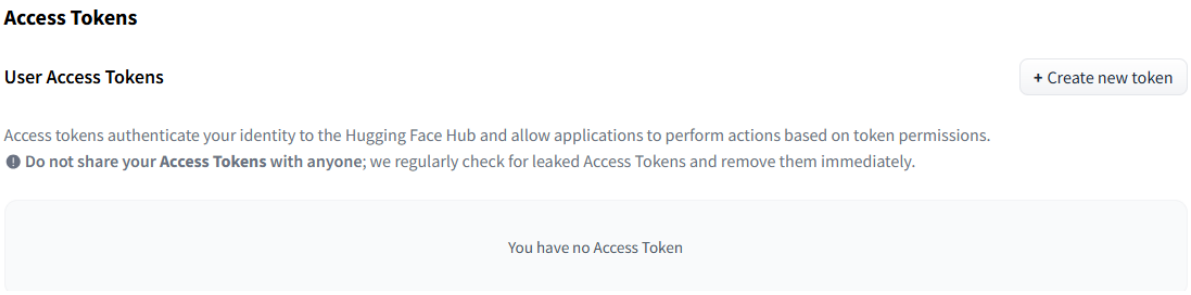
Step 2: Enter your unique username and full name and create your account



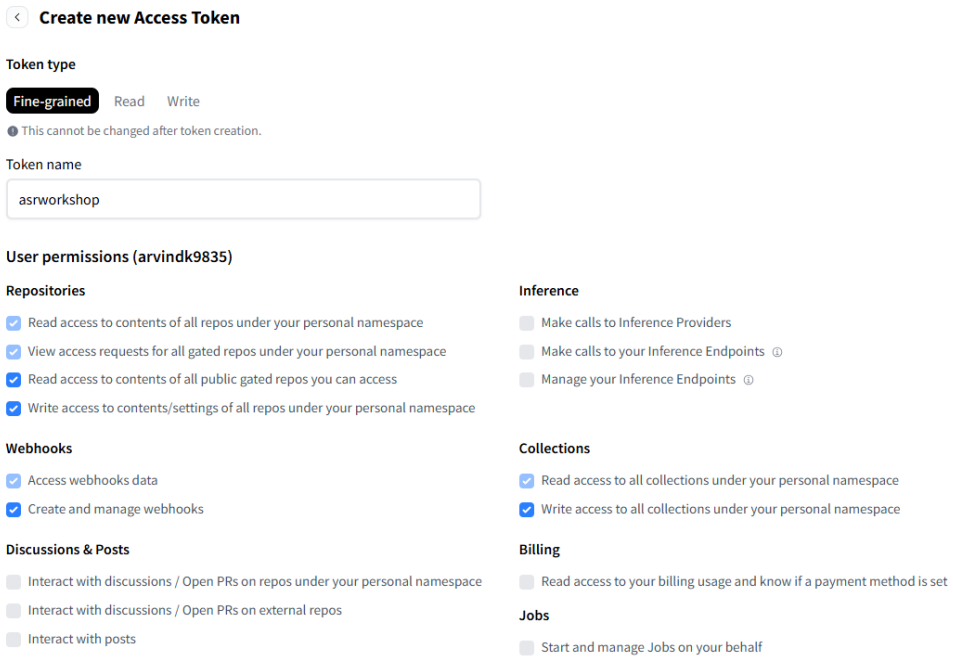
Step 3: Check your registered email and activate your account



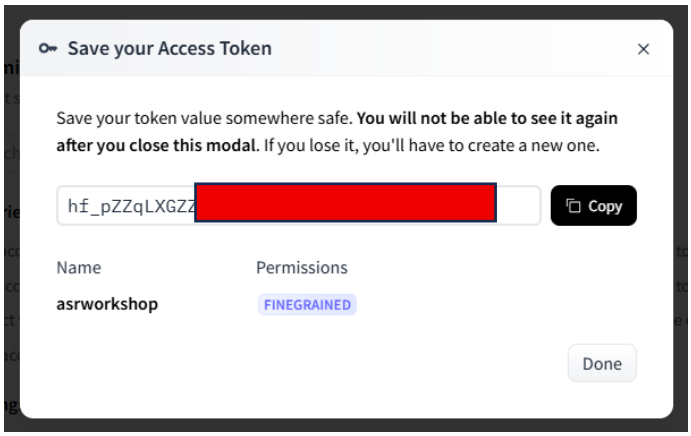
Step 4: Go to profile->Access Tokens->Create new token



Step 5: Rename the token name and give necessary read and write permissions



Step 6: Save your token in notepad or any safe place. Your token is successfully created.



Section 2: Google Colab

Step 1: Open Google Colab Secrets

1. Open your notebook in Google Colab
2. In the left sidebar, click the 🔑 **Secrets** icon.

Step 2: Add the Hugging Face Token

1. Click **"+ Add new secret"**
2. Enter:

Field	Value
Name	HF_TOKEN
Value	paste your Hugging Face token

3. Enable:
    - **Notebook access** → ON

Using the name HF_TOKEN is recommended because Hugging Face libraries automatically detect it.

Step 3: Access the Token in Colab

```
from google.colab import userdata
from huggingface_hub import login

# Get token from Colab Secrets
hf_token = userdata.get('HF_TOKEN')

# Login to Hugging Face
login(hf_token)

print("Logged into Hugging Face successfully!")
```

Section 2.1: ASR Demo on Basque

```
#installation of libraries
!pip install -q transformers torchaudio librosa soundfile accelerate
!pip install datasets==3.2.0


#Step1: Loading a sample Basque wav file from common-voice
from datasets import load_dataset
dataset_eu = load_dataset(
    "fsicoli/common_voice_16_0",
    "eu",
    split="test[:5]",
    trust_remote_code=True
)

#Step2: Saving the Basque wav file
import soundfile as sf
sample_eu = dataset_eu[0]
print(sample_eu["sentence"])
audio_array = sample_eu["audio"]["array"]
sampling_rate = sample_eu["audio"]["sampling_rate"]

sf.write("basque.wav", audio_array, sampling_rate)

#Step3: Play the file
from IPython.display import Audio
Audio("basque.wav")

#Step4: Load Whisper model
from transformers import pipeline
import torch

device = 0 if torch.cuda.is_available() else -1

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

```
#Step5: Run Whisper on Basque
result = whisper_pipe(
    "basque.wav",
    generate_kwargs={"language": "basque"}
)

print("Whisper Basque Transcription:")
print(result["text"])
print("Basque Ground Truth:")
print(sample_eu["sentence"])

#Step6: Load Wav2Vec2/XLS-R Model for Basque
import librosa
import torch

from transformers import (
    Wav2Vec2Processor,
    Wav2Vec2ForCTC
)

processor_eu = Wav2Vec2Processor.from_pretrained(
    "cahya/wav2vec2-large-xlsr-basque"
)

model_eu = Wav2Vec2ForCTC.from_pretrained(
    "cahya/wav2vec2-large-xlsr-basque"
)

#Step7: Load basque audio
speech, sr = librosa.load("basque.wav", sr=16000)

#Step8: Run Basque Wav2Vec2 ASR
inputs = processor_eu(
    speech,
    sampling_rate=16000,
    return_tensors="pt",
    padding=True
)
```

```
with torch.no_grad():
    logits = model_eu(inputs.input_values).logits

predicted_ids = torch.argmax(logits, dim=-1)

transcription = processor_eu.batch_decode(predicted_ids)

print("Wav2Vec2 Basque Transcription:")
print(transcription[0])
print("Basque Ground Truth:")
print(sample_eu["sentence"])
```

Section 2.2: ASR Demo for Spanish

```
#using a sample Spanish file and transcript from Kaggle
#https://www.kaggle.com/datasets/bryanpark/spanish-single-speaker-speech-dataset

#Step1: Upload WAV file
from google.colab import files
uploaded = files.upload()
# Get uploaded filename
wav_file = list(uploaded.keys())[0]
print("Uploaded file:", wav_file)

#Step2: Run Whisper on Spanish
result = whisper_pipe(
    "spanish.wav",
    generate_kwargs={"language": "spanish"}
)

print("Whisper Spanish Transcription:")
print(result["text"])
print("Spanish Ground Truth:")
print("Durante nuestra conversación advertí que la multitud aumentaba, apretándose más")

#Step3: Load Wav2Vec Spanish Model
import librosa
import torch
```

```
from transformers import (
    Wav2Vec2Processor,
    Wav2Vec2ForCTC
)

processor_es = Wav2Vec2Processor.from_pretrained(
    "mrm8488/wav2vec2-large-xlsr-53-spanish"
)

model_es = Wav2Vec2ForCTC.from_pretrained(
    "mrm8488/wav2vec2-large-xlsr-53-spanish"
)

#Step4: Load Spanish audio
speech, sr = librosa.load("spanish.wav", sr=16000)

#Step5: Run Spanish Wav2Vec2 ASR
inputs = processor_es(
    speech,
    sampling_rate=16000,
    return_tensors="pt",
    padding=True
)

with torch.no_grad():
    logits = model_es(inputs.input_values).logits

predicted_ids = torch.argmax(logits, dim=-1)

transcription = processor_es.batch_decode(predicted_ids)

print("Wav2Vec2 Basque Transcription:")
print(transcription[0])
print("Spanish Ground Truth:")
print("Durante nuestra conversación advertí que la multitud
aumentaba, apretándose más")
```

Section 2.3: Fine-Tuning Wav2Vec BERT for Low Resource Mongolian Language

Step 1: Installing necessary libraries

```
!pip install datasets==3.2.0
!pip install --upgrade transformers
!pip install torchaudio
!pip install jiwer
!pip install accelerate -U
!pip install evaluate
```

Step 2: Reading the Mongolian dataset

```
from datasets import load_dataset, Audio
from evaluate import load
metric = load("wer")
common_voice_train = load_dataset("fsicoli/common_voice_16_0", "mn",
split="train+validation", token=True)
common_voice_test = load_dataset("fsicoli/common_voice_16_0", "mn",
split="test", token=True)
```

Step 3: Removing unnecessary information about the audio

```
common_voice_train = common_voice_train.remove_columns(["accent",
"age", "client_id", "down_votes", "gender", "locale", "segment",
"up_votes"])

common_voice_test = common_voice_test.remove_columns(["accent",
"age", "client_id", "down_votes", "gender", "locale", "segment",
"up_votes"])
```

Step 4: Display some random samples of the dataset

```
from datasets import ClassLabel
import random
import pandas as pd
from IPython.display import display, HTML

def show_random_elements(dataset, num_examples=10):
    assert num_examples <= len(dataset), "Can't pick more elements
than there are in the dataset."
    picks = []
    for _ in range(num_examples):
        pick = random.randint(0, len(dataset)-1)
```

```
        while pick in picks:
            pick = random.randint(0, len(dataset)-1)
        picks.append(pick)

    df = pd.DataFrame(dataset[picks])
    display(HTML(df.to_html()))

show_random_elements(common_voice_train.remove_columns(["path",
"audio"]), num_examples=10)
```

Step 5: Remove special character from the text

```
import re
chars_to_remove_regex = '[\,\?\.\!\-\;\:\"\“\%\‘\”\□\'\»\«]'

def remove_special_characters(batch):
    # remove special characters
    batch["sentence"] = re.sub(chars_to_remove_regex, '',
batch["sentence"]).lower()

    return batch

common_voice_train =
common_voice_train.map(remove_special_characters)
common_voice_test = common_voice_test.map(remove_special_characters)


show_random_elements(common_voice_train.remove_columns(["path","audi
o"]))
```

Step 6: Transforms the string into a set of chars and create dictionary

```
def extract_all_chars(batch):
  all_text = " ".join(batch["sentence"])
  vocab = list(set(all_text))
  return {"vocab": [vocab], "all_text": [all_text]}


vocab_train = common_voice_train.map(extract_all_chars,
batched=True, batch_size=-1, keep_in_memory=True,
remove_columns=common_voice_train.column_names)
vocab_test = common_voice_test.map(extract_all_chars, batched=True,
batch_size=-1, keep_in_memory=True,
remove_columns=common_voice_test.column_names)
```

```
#enumerated dictionary
vocab_list = list(set(vocab_train["vocab"][0]) |
set(vocab_test["vocab"][0]))

vocab_dict = {v: k for k, v in enumerate(sorted(vocab_list))}
vocab_dict
```

Step 7: Removing Latin Characters from the dictionary

```
def remove_latin_characters(batch):
    batch["sentence"] = re.sub(r'[a-z]+', '', batch["sentence"])
    return batch

# remove latin characters
common_voice_train = common_voice_train.map(remove_latin_characters)
common_voice_test = common_voice_test.map(remove_latin_characters)

# extract unique characters again
vocab_train = common_voice_train.map(extract_all_chars,
batched=True, batch_size=-1, keep_in_memory=True,
remove_columns=common_voice_train.column_names)
vocab_test = common_voice_test.map(extract_all_chars, batched=True,
batch_size=-1, keep_in_memory=True,
remove_columns=common_voice_test.column_names)
vocab_list = list(set(vocab_train["vocab"][0]) |
set(vocab_test["vocab"][0]))

vocab_dict = {v: k for k, v in enumerate(sorted(vocab_list))}
vocab_dict

#adding token for blank space, unknown character and padding
vocab_dict["|"] = vocab_dict[" "]
del vocab_dict[" "]
vocab_dict["[UNK]"] = len(vocab_dict)
vocab_dict["[PAD]"] = len(vocab_dict)
len(vocab_dict)
```

Step 8: Creating vocabulary as json file

```
import json
```

```
with open('vocab.json', 'w') as vocab_file:
    json.dump(vocab_dict, vocab_file)
```

Step 9: Create Tokenizer

```
from transformers import Wav2Vec2CTCTokenizer

tokenizer = Wav2Vec2CTCTokenizer.from_pretrained("./",
unk_token="[UNK]", pad_token="[PAD]", word_delimiter_token="|")
```

Step 10: Save the tokenizer to the Hub

```
repo_name = "w2v-bert-2.0-mongolian-colab-CV16.0"
tokenizer.push_to_hub(repo_name)
```

Step 11: Convert 1D audio signal into two-dimensional matrix of log-mel spectrogram values

```
from transformers import SeamlessM4TFeatureExtractor
feature_extractor =
SeamlessM4TFeatureExtractor.from_pretrained("facebook/w2v-bert-2.0")
```

Step 12: Wrapping of the feature extractor and tokenizer into a single Wav2Vec2BertProcessor

```
from transformers import Wav2Vec2BertProcessor

processor =
Wav2Vec2BertProcessor(feature_extractor=feature_extractor,
tokenizer=tokenizer)
processor.push_to_hub(repo_name)
```

Step 13: Looking into the audio signal and the path of the audio file

```
common_voice_train[0]["path"]
common_voice_train[0]["audio"]
```

Step 14: Resampling the audio files from 48K to 16K

```
common_voice_train = common_voice_train.cast_column("audio",
Audio(sampling_rate=16_000))
common_voice_test = common_voice_test.cast_column("audio",
Audio(sampling_rate=16_000))
```

Step 15: Checking the audio signal for resampling

```
common_voice_train[0]["audio"]
```

Step 16: Listening to some audio samples

```
import IPython.display as ipd
import numpy as np
import random

rand_int = random.randint(0, len(common_voice_train)-1)

print(common_voice_train[rand_int]["sentence"])
ipd.Audio(data=common_voice_train[rand_int]["audio"]["array"],
autoplay=True, rate=16000)
```

Step 17: Checking if the data is correctly prepared

```
rand_int = random.randint(0, len(common_voice_train)-1)

print("Target text:", common_voice_train[rand_int]["sentence"])
print("Input array shape:",
common_voice_train[rand_int]["audio"]["array"].shape)
print("Sampling rate:",
common_voice_train[rand_int]["audio"]["sampling_rate"])
```

Step 18: Preparing data for training

```
def prepare_dataset(batch):
    audio = batch["audio"]
    batch["input_features"] = processor(audio["array"],
sampling_rate=audio["sampling_rate"]).input_features[0]
    batch["input_length"] = len(batch["input_features"])

    batch["labels"] = processor(text=batch["sentence"]).input_ids
    return batch


common_voice_train = common_voice_train.map(prepare_dataset,
remove_columns=common_voice_train.column_names)
common_voice_test = common_voice_test.map(prepare_dataset,
remove_columns=common_voice_test.column_names)
```

Step 19: Preparing data collator to input features and labels

```
import torch

from dataclasses import dataclass, field
from typing import Any, Dict, List, Optional, Union

@dataclass
class DataCollatorCTCWithPadding:

    processor: Wav2Vec2BertProcessor
    padding: Union[bool, str] = True

    def __call__(self, features: List[Dict[str, Union[List[int],
torch.Tensor]]]) -> Dict[str, torch.Tensor]:
        # split inputs and labels since they have to be of different
lengths and need
        # different padding methods
        input_features = [{"input_features":
feature["input_features"]} for feature in features]
        label_features = [{"input_ids": feature["labels"]} for
feature in features]

        batch = self.processor.pad(
            input_features,
            padding=self.padding,
            return_tensors="pt",
        )

        labels_batch = self.processor.pad(
            labels=label_features,
            padding=self.padding,
            return_tensors="pt",
        )
        # replace padding with -100 to ignore loss correctly
        labels =
labels_batch["input_ids"].masked_fill(labels_batch.attention_mask.ne
(1), -100)

        batch["labels"] = labels
```

```
        return batch

data_collator = DataCollatorCTCWithPadding(processor=processor,
padding=True)
```

Step 20: Using WER as a metric

```
from evaluate import load
wer_metric = load("wer")
```

Step 21: Defining the compute metrics used for decoding

```
def compute_metrics(pred):
    pred_logits = pred.predictions
    pred_ids = np.argmax(pred_logits, axis=-1)

    pred.label_ids[pred.label_ids == -100] =
processor.tokenizer.pad_token_id

    pred_str = processor.batch_decode(pred_ids)
    # we do not want to group tokens when computing the metrics
    label_str = processor.batch_decode(pred.label_ids,
group_tokens=False)

    wer = wer_metric.compute(predictions=pred_str,
references=label_str)

    return {"wer": wer}
```

Step 22: Defining the model and setting hyper-parameter to train Wav2Vec2-BERT

```
from transformers import Wav2Vec2BertForCTC

model = Wav2Vec2BertForCTC.from_pretrained(
    "facebook/w2v-bert-2.0",
    attention_dropout=0.0,
    hidden_dropout=0.0,
    feat_proj_dropout=0.0,
    mask_time_prob=0.0,
    layerdrop=0.0,
    ctc_loss_reduction="mean",
```

```
    add_adapter=True,
    pad_token_id=processor.tokenizer.pad_token_id,
    ignore_mismatched_sizes=True,
    vocab_size=len(processor.tokenizer),
)

from transformers import TrainingArguments

training_args = TrainingArguments(
  output_dir=repo_name,
  per_device_train_batch_size=16,
  gradient_accumulation_steps=2,
  eval_strategy="steps",
  num_train_epochs=10,
  gradient_checkpointing=True,
  fp16=True,
  save_steps=600,
  eval_steps=300,
  logging_steps=300,
  learning_rate=5e-5,
  warmup_steps=500,
  save_total_limit=2,
  push_to_hub=False,
)
```

Step 23: Defining the trainer and start training

```
from transformers import Trainer

trainer = Trainer(
    model=model,
    data_collator=data_collator,
    args=training_args,
    compute_metrics=compute_metrics,
    train_dataset=common_voice_train,
    eval_dataset=common_voice_test,
)

trainer.train()
trainer.push_to_hub()
```

Step	Training Loss	Validation Loss	Wer
300	1.712700	0.647740	0.517892
600	0.349300	0.615849	0.442027
900	0.180500	0.525088	0.367305
1200	0.075400	0.528768	0.324016

Step 24: Loading a pre-trained model and evaluate

```
from transformers import AutoModelForCTC, Wav2Vec2BertProcessor

model = AutoModelForCTC.from_pretrained("ylacombe/w2v-bert-2.0-mongolian-colab-CV16.0")
processor = Wav2Vec2BertProcessor.from_pretrained("ylacombe/w2v-bert-2.0-mongolian-colab-CV16.0")

sample = common_voice_test[0]
input_features =
torch.tensor(sample["input_features"]).to("cuda").unsqueeze(0)

model.to("cuda") # Move the model to the CUDA device

with torch.no_grad():
    logits = model(input_features).logits

pred_ids = torch.argmax(logits, dim=-1)[0]

#print the decoded output and reference output
print(processor.decode(pred_ids))
print(processor.decode(sample["labels"]).lower())
```

References:

[1] https://huggingface.co/blog/fine-tune-w2v2-bert

[2] https://mozilladatacollective.com/datasets/cmn2hwe0d01n8mm07wug9r5he

[3] https://www.kaggle.com/datasets/bryanpark/spanish-single-speaker-speech-dataset

[4] Link of the Google-Colab file
https://colab.research.google.com/drive/1a2RTbknRCb8mzKun0feqZvhdjv1P5QmO?usp=sharing