要训练 Sherpa-ONNX 的模型，你需要准备一个合适的数据集并按照 Sherpa 或其他相关的深度学习语音识别框架的训练流程进行训练。Sherpa 是一个轻量级的端到端语音识别（ASR）框架，它支持多种模型格式，包括 ONNX，用于在各种设备上进行高效推理。

### 1. 选择和准备数据集

为了训练一个 Sherpa-ONNX 模型，你需要一个用于语音识别（ASR）的数据集。常用的语音数据集包括：

- \*\*LibriSpeech\*\*：一个大型的英语语音识别数据集，包含 1000 小时的英文朗读音频及其转录文本。

- \*\*Common Voice\*\*：由 Mozilla 提供的开源语音数据集，包含多种语言的音频数据。

- \*\*AISHELL-1\*\*：一个中文语音识别数据集，包含 170 小时的中文普通话音频数据。

- \*\*TED-LIUM\*\*：一个用于语音识别的 TED 演讲数据集，包含多种语言和口音的音频。

你可以从这些数据集的官方网站下载它们：

- [LibriSpeech](http://www.openslr.org/12/)

- [Common Voice](https://commonvoice.mozilla.org/)

- [AISHELL-1](http://www.openslr.org/33/)

- [TED-LIUM](http://www.openslr.org/51/)

下载这些数据集并解压缩到你的本地目录中。每个数据集通常包含音频文件和相应的转录文本。

### 2. 安装和准备 Sherpa

要训练 Sherpa 模型，你需要安装一些必要的依赖项和工具。可以使用以下步骤来安装 Sherpa 的依赖项：

1. \*\*克隆 Sherpa 仓库\*\*：

```bash

git clone https://github.com/k2-fsa/sherpa.git

cd sherpa

```

2. \*\*安装依赖项\*\*：

使用 `pip` 安装 Python 依赖项：

```bash

pip install -r requirements.txt

```

### 3. 配置训练参数

在 Sherpa 目录中，有一个配置文件或 Python 脚本用于定义训练的参数和超参数。你需要根据你的数据集进行相应的调整。通常你需要配置以下几个部分：

- \*\*数据集路径\*\*：指定数据集的存储路径，例如 LibriSpeech 或 Common Voice。

- \*\*模型架构\*\*：选择适当的模型架构（如 Transformer、Conformer 等）。

- \*\*训练超参数\*\*：设置批大小（batch size）、学习率（learning rate）、优化器（optimizer）、训练 epoch 数量等。

### 4. 训练模型

一旦准备好数据集并配置好参数，就可以开始训练模型。假设你使用的是一个 `train.py` 脚本，你可以使用以下命令启动训练过程：

```bash

python train.py --config=config.yaml

```

或使用更复杂的命令，指定数据路径和其他参数：

```bash

python train.py --data-dir /path/to/dataset --batch-size 32 --epochs 50

```

训练过程将会根据你配置的参数和数据集进行。根据数据集的大小和模型的复杂性，训练时间可能会很长。

### 5. 导出模型为 ONNX 格式

训练完成后，你需要将模型导出为 ONNX 格式，以便用于推理。大多数深度学习框架（如 PyTorch、TensorFlow）都支持将模型导出为 ONNX 格式。

假设你使用的是 PyTorch，可以使用以下代码来导出模型：

```python

import torch.onnx

# 假设你已经加载了训练好的模型

model = ... # 加载模型

dummy\_input = torch.randn(1, 16000) # 模拟输入数据

torch.onnx.export(model, dummy\_input, "sherpa\_model.onnx", export\_params=True, opset\_version=11)

```

### 6. 使用 Sherpa-ONNX 进行推理

导出模型后，你可以使用 Sherpa-ONNX 库进行推理。确保你的 ONNX 模型和 Sherpa-ONNX 版本兼容，并按照 Sherpa 文档中的说明进行推理。

### 总结

要训练 Sherpa-ONNX 模型，需要：

1. \*\*下载并准备一个语音识别数据集\*\*（如 LibriSpeech、Common Voice）。

2. \*\*安装 Sherpa 并配置训练环境\*\*。

3. \*\*配置训练参数\*\*（数据集路径、模型架构等）。

4. \*\*训练模型并导出为 ONNX 格式\*\*。

5. \*\*使用 Sherpa-ONNX 进行推理\*\*。

## **csukuangfj/sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20 (Bilingual, Chinese + English)[](https://k2-fsa.github.io/sherpa/onnx/pretrained_models/online-transducer/zipformer-transducer-models.html" \l "csukuangfj-sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20-bilingual-chinese-english" \o "Permalink to this heading)**

This model is converted from

<https://huggingface.co/csukuangfj/k2fsa-zipformer-chinese-english-mixed>

which supports both Chinese and English. The model is contributed by the community and is trained on tens of thousands of some internal dataset.

In the following, we describe how to download it and use it with [sherpa-onnx](https://github.com/k2-fsa/sherpa-onnx).

### **Download the model[](https://k2-fsa.github.io/sherpa/onnx/pretrained_models/online-transducer/zipformer-transducer-models.html" \l "id28" \o "Permalink to this heading)**

Please use the following commands to download it.

cd /path/to/sherpa-onnx

wget https://github.com/k2-fsa/sherpa-onnx/releases/download/asr-models/sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20.tar.bz2

*# For Chinese users, you can use the following mirror# wget https://hub.nuaa.cf/k2-fsa/sherpa-onnx/releases/download/asr-models/sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20.tar.bz2*

tar xvf sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20.tar.bz2

rm sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20.tar.bz2

Please check that the file sizes of the pre-trained models are correct. See the file sizes of \*.onnx files below.

sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20$ ls -lh \*.onnx

-rw-r--r-- 1 kuangfangjun root 13M Mar 31 21:11 decoder-epoch-99-avg-1.int8.onnx

-rw-r--r-- 1 kuangfangjun root 14M Feb 20 20:13 decoder-epoch-99-avg-1.onnx

-rw-r--r-- 1 kuangfangjun root 174M Mar 31 21:11 encoder-epoch-99-avg-1.int8.onnx

-rw-r--r-- 1 kuangfangjun root 315M Feb 20 20:13 encoder-epoch-99-avg-1.onnx

-rw-r--r-- 1 kuangfangjun root 3.1M Mar 31 21:11 joiner-epoch-99-avg-1.int8.onnx

-rw-r--r-- 1 kuangfangjun root 13M Feb 20 20:13 joiner-epoch-99-avg-1.onnx

### **Decode a single wave file[](https://k2-fsa.github.io/sherpa/onnx/pretrained_models/online-transducer/zipformer-transducer-models.html" \l "id29" \o "Permalink to this heading)**

**Hint**

It supports decoding only wave files of a single channel with 16-bit encoded samples, while the sampling rate does not need to be 16 kHz.

#### **fp32[](https://k2-fsa.github.io/sherpa/onnx/pretrained_models/online-transducer/zipformer-transducer-models.html" \l "id30" \o "Permalink to this heading)**

The following code shows how to use fp32 models to decode a wave file:

cd /path/to/sherpa-onnx

./build/bin/sherpa-onnx **\** --tokens=./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/tokens.txt **\** --encoder=./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/encoder-epoch-99-avg-1.onnx **\** --decoder=./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/decoder-epoch-99-avg-1.onnx **\** --joiner=./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/joiner-epoch-99-avg-1.onnx **\** ./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/test\_wavs/1.wav

**Note**

Please use ./build/bin/Release/sherpa-onnx.exe for Windows.

**Caution**

If you use Windows and get encoding issues, please run:

CHCP 65001

in your commandline.

You should see the following output:

OnlineRecognizerConfig(feat\_config=FeatureExtractorConfig(sampling\_rate=16000, feature\_dim=80), model\_config=OnlineTransducerModelConfig(encoder\_filename="./sherpa-onnx-streaming-zipformer-en-2023-02-21/encoder-epoch-99-avg-1.onnx", decoder\_filename="./sherpa-onnx-streaming-zipformer-en-2023-02-21/decoder-epoch-99-avg-1.onnx", joiner\_filename="./sherpa-onnx-streaming-zipformer-en-2023-02-21/joiner-epoch-99-avg-1.onnx", tokens="./sherpa-onnx-streaming-zipformer-en-2023-02-21/tokens.txt", num\_threads=2, debug=False), endpoint\_config=EndpointConfig(rule1=EndpointRule(must\_contain\_nonsilence=False, min\_trailing\_silence=2.4, min\_utterance\_length=0), rule2=EndpointRule(must\_contain\_nonsilence=True, min\_trailing\_silence=1.2, min\_utterance\_length=0), rule3=EndpointRule(must\_contain\_nonsilence=False, min\_trailing\_silence=0, min\_utterance\_length=20)), enable\_endpoint=True, max\_active\_paths=4, decoding\_method="greedy\_search")

2023-04-01 06:22:23.030317206 [E:onnxruntime:, env.cc:251 ThreadMain] pthread\_setaffinity\_np failed for thread: 604942, index: 16, mask: {17, 53, }, error code: 22 error msg: Invalid argument. Specify the number of threads explicitly so the affinity is not set.

2023-04-01 06:22:23.030315351 [E:onnxruntime:, env.cc:251 ThreadMain] pthread\_setaffinity\_np failed for thread: 604941, index: 15, mask: {16, 52, }, error code: 22 error msg: Invalid argument. Specify the number of threads explicitly so the affinity is not set.

sampling rate of input file: 16000

wav filename: ./sherpa-onnx-streaming-zipformer-en-2023-02-21/test\_wavs/0.wav

wav duration (s): 6.625

Started

Done!

Recognition result for ./sherpa-onnx-streaming-zipformer-en-2023-02-21/test\_wavs/0.wav:

AFTER EARLY NIGHTFALL THE YELLOW LAMPS WOULD LIGHT UP HERE AND THERE THE SQUALID QUARTER OF THE BROTHELS

num threads: 2

decoding method: greedy\_search

Elapsed seconds: 0.815 s

Real time factor (RTF): 0.815 / 6.625 = 0.123

#### **int8[](https://k2-fsa.github.io/sherpa/onnx/pretrained_models/online-transducer/zipformer-transducer-models.html" \l "id31" \o "Permalink to this heading)**

The following code shows how to use fp32 models to decode a wave file:

cd /path/to/sherpa-onnx

./build/bin/sherpa-onnx **\** --tokens=./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/tokens.txt **\** --encoder=./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/encoder-epoch-99-avg-1.int8.onnx **\** --decoder=./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/decoder-epoch-99-avg-1.int8.onnx **\** --joiner=./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/joiner-epoch-99-avg-1.int8.onnx **\** ./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/test\_wavs/1.wav

**Note**

Please use ./build/bin/Release/sherpa-onnx.exe for Windows.

**Caution**

If you use Windows and get encoding issues, please run:

CHCP 65001

in your commandline.

You should see the following output:

OnlineRecognizerConfig(feat\_config=FeatureExtractorConfig(sampling\_rate=16000, feature\_dim=80), model\_config=OnlineTransducerModelConfig(encoder\_filename="./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/encoder-epoch-99-avg-1.int8.onnx", decoder\_filename="./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/decoder-epoch-99-avg-1.int8.onnx", joiner\_filename="./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/joiner-epoch-99-avg-1.int8.onnx", tokens="./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/tokens.txt", num\_threads=2, debug=False), endpoint\_config=EndpointConfig(rule1=EndpointRule(must\_contain\_nonsilence=False, min\_trailing\_silence=2.4, min\_utterance\_length=0), rule2=EndpointRule(must\_contain\_nonsilence=True, min\_trailing\_silence=1.2, min\_utterance\_length=0), rule3=EndpointRule(must\_contain\_nonsilence=False, min\_trailing\_silence=0, min\_utterance\_length=20)), enable\_endpoint=True, max\_active\_paths=4, decoding\_method="greedy\_search")

2023-04-01 06:24:10.503505750 [E:onnxruntime:, env.cc:251 ThreadMain] pthread\_setaffinity\_np failed for thread: 604982, index: 16, mask: {17, 53, }, error code: 22 error msg: Invalid argument. Specify the number of threads explicitly so the affinity is not set.

2023-04-01 06:24:10.503503942 [E:onnxruntime:, env.cc:251 ThreadMain] pthread\_setaffinity\_np failed for thread: 604981, index: 15, mask: {16, 52, }, error code: 22 error msg: Invalid argument. Specify the number of threads explicitly so the affinity is not set.

sampling rate of input file: 16000

wav filename: ./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/test\_wavs/1.wav

wav duration (s): 5.100

Started

Done!

Recognition result for ./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/test\_wavs/1.wav:

这是第一种第二种叫呃与 ALWAYS ALWAYS什么意思啊

num threads: 2

decoding method: greedy\_search

Elapsed seconds: 0.551 s

Real time factor (RTF): 0.551 / 5.100 = 0.108

### **Real-time speech recognition from a microphone[](https://k2-fsa.github.io/sherpa/onnx/pretrained_models/online-transducer/zipformer-transducer-models.html" \l "id32" \o "Permalink to this heading)**

cd /path/to/sherpa-onnx

./build/bin/sherpa-onnx-microphone **\** --tokens=./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/tokens.txt **\** --encoder=./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/encoder-epoch-99-avg-1.onnx **\** --decoder=./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/decoder-epoch-99-avg-1.onnx **\** --joiner=./sherpa-onnx-streaming-zipformer-bilingual-zh-en-2023-02-20/joiner-epoch-99-avg-1.onnx

### 1. 准备环境和依赖

首先，确保你的环境中有必要的依赖项，包括 PyTorch、CUDA（如果你使用 GPU 训练）、ONNX、以及其他深度学习相关的库。你还需要克隆 Sherpa 的 GitHub 仓库来获取必要的代码和工具。

bash

复制代码

# 克隆 Sherpa 的 GitHub 仓库

git clone https://github.com/k2-fsa/sherpacd sherpa

安装相关依赖项（Python包）：

bash

复制代码

pip install -r requirements.txt

### 2. 准备数据集

Sherpa 模型是基于 ASR（自动语音识别）的模型。要训练类似 Sherpa-ONNX 的 Zipformer 模型，你需要以下数据：

* **语音数据**：可以使用公开的语音数据集，例如 LibriSpeech、AISHELL-1、Common Voice 等等。
* **文本转录数据**：每个音频文件需要有相应的文本转录。

你可以根据目标语言（如中文、英文）混合多种数据集来创建一个中英双语数据集。

### 3. 数据预处理

数据预处理是训练语音识别模型的重要步骤。你需要对音频数据进行特征提取（如 MFCC、Mel-spectrogram），以及对文本数据进行分词和标注。

你可以使用 Kaldi 或者 PyTorch 提供的工具包进行特征提取。Sherpa 项目通常使用 k2 和 torchaudio 来处理音频。

### 4. 模型架构设计

Sherpa 使用的 Zipformer 是一种改进的变压器架构。你可以根据 Sherpa 项目中的代码实现来配置和调整模型架构参数，例如：

* **编码器**：多层深度变压器架构，专注于处理语音特征。
* **解码器**：使用序列到序列模型来预测文字。
* **联合模块**：融合编码器和解码器的输出，形成最终的预测结果。

你可以在 sherpa 项目中找到这些模型的配置，并根据需要进行调整。

### 5. 训练模型

使用 PyTorch 和 k2 框架训练模型。在训练前，请确保设置好训练的超参数，例如：

* 学习率
* 批量大小
* 训练轮数（epochs）
* 优化器（如 Adam）

可以编写训练脚本来控制模型训练过程，或修改现有的训练代码。

bash

复制代码

# 示例：启动训练脚本

python train.py --config ./configs/zipformer\_config.yaml

### 6. 导出 ONNX 模型

训练完成后，导出 PyTorch 模型为 ONNX 格式。PyTorch 提供了将模型导出为 ONNX 格式的简单接口：

python

复制代码

import torch

# 假设 encoder, decoder, joiner 是你训练的模型实例

torch.onnx.export(encoder, dummy\_input, "encoder.onnx")

torch.onnx.export(decoder, dummy\_input, "decoder.onnx")

torch.onnx.export(joiner, dummy\_input, "joiner.onnx")

### 7. 推理与测试

导出模型后，你可以使用 Sherpa 的推理代码或工具对新训练的模型进行测试和验证，确保其在推理时的性能和准确度符合预期。

### 8. 优化和微调

根据模型的表现，可以对数据集、模型架构、超参数等进行调整，以优化模型性能。对于大规模训练任务，你可能需要使用分布式训练或混合精度训练来加速训练过程。

要访问此数据集，您需要同意共享您的联系信息。

该仓库是公开访问的，但您必须接受条件才能访问其文件和内容。

通过点击下方的“访问仓库”按钮，您也同意不尝试识别 Common Voice 数据集中的说话者身份。

要查看条件并访问此数据集的内容，请点击此处。

### Common Voice Corpus 12.0 数据集卡

#### 数据集摘要

Common Voice 数据集包含独特的 MP3 文件和相应的文本文件。数据集中许多 26119 小时的录音还包括人口统计元数据，例如年龄、性别和口音，这有助于提高语音识别引擎的准确性。

当前数据集由 104 种语言的 17127 小时已验证音频组成，但总会有更多的声音和语言被添加。查看 [Languages 页面](https://commonvoice.mozilla.org/languages)来请求添加新的语言或开始贡献。

#### 支持的任务和排行榜

使用 Common Voice 数据集训练的模型的结果可以通过 [🤗 Autoevaluate Leaderboard](https://huggingface.co/autoeval) 查看。

#### 支持的语言

Abkhaz, Arabic, Armenian, Assamese, Asturian, Azerbaijani, Basaa, Bashkir, Basque, Belarusian, Bengali, Breton, Bulgarian, Cantonese, Catalan, Central Kurdish, Chinese (China), Chinese (Hong Kong), Chinese (Taiwan), Chuvash, Czech, Danish, Dhivehi, Dutch, English, Erzya, Esperanto, Estonian, Finnish, French, Frisian, Galician, Georgian, German, Greek, Guarani, Hakha Chin, Hausa, Hill Mari, Hindi, Hungarian, Igbo, Indonesian, Interlingua, Irish, Italian, Japanese, Kabyle, Kazakh, Kinyarwanda, Korean, Kurmanji Kurdish, Kyrgyz, Latvian, Lithuanian, Luganda, Macedonian, Malayalam, Maltese, Marathi, Meadow Mari, Moksha, Mongolian, Nepali, Norwegian Nynorsk, Occitan, Odia, Persian, Polish, Portuguese, Punjabi, Quechua Chanka, Romanian, Romansh Sursilvan, Romansh Vallader, Russian, Sakha, Santali (Ol Chiki), Saraiki, Sardinian, Serbian, Slovak, Slovenian, Sorbian, Upper, Spanish, Swahili, Swedish, Taiwanese (Minnan), Tamil, Tatar, Thai, Tigre, Tigrinya, Toki Pona, Turkish, Twi, Ukrainian, Urdu, Uyghur, Uzbek, Vietnamese, Votic, Welsh, Yoruba.

#### 如何使用

`datasets` 库允许您在纯 Python 中加载和预处理数据集。可以使用 `load\_dataset` 函数一次性将数据集下载并准备到本地驱动器上。

例如，要下载印地语的配置，只需指定相应的语言配置名称（例如，“hi”代表印地语）：

```python

from datasets import load\_dataset

cv\_12 = load\_dataset("mozilla-foundation/common\_voice\_12\_0", "hi", split="train")

```

使用 `datasets` 库，您还可以通过添加 `streaming=True` 参数来实时流式加载数据集。在流模式下加载数据集会一次加载数据集的单个样本，而不是将整个数据集下载到磁盘。

```python

from datasets import load\_dataset

cv\_12 = load\_dataset("mozilla-foundation/common\_voice\_12\_0", "hi", split="train", streaming=True)

print(next(iter(cv\_12)))

```

您还可以使用自己的数据集（本地或流式）直接创建一个 PyTorch 数据加载器。

\*\*本地\*\*

```python

from datasets import load\_dataset

from torch.utils.data.sampler import BatchSampler, RandomSampler

cv\_12 = load\_dataset("mozilla-foundation/common\_voice\_12\_0", "hi", split="train")

batch\_sampler = BatchSampler(RandomSampler(cv\_12), batch\_size=32, drop\_last=False)

dataloader = DataLoader(cv\_12, batch\_sampler=batch\_sampler)

```

\*\*流式\*\*

```python

from datasets import load\_dataset

from torch.utils.data import DataLoader

cv\_12 = load\_dataset("mozilla-foundation/common\_voice\_12\_0", "hi", split="train")

dataloader = DataLoader(cv\_12, batch\_size=32)

```

了解有关加载和准备音频数据集的更多信息，请访问 [Hugging Face 博客](https://hf.co/blog/audio-datasets)。

#### 数据集结构

##### 数据实例

典型的数据点包含音频文件的路径及其对应的句子。附加字段包括口音、年龄、客户端 ID、点赞数、点踩数、性别、区域设置和片段。

```json

{

'client\_id': 'd59478fbc1ee646a28a3c652a119379939123784d99131b865a89f8b21c81f69276c48bd574b81267d9d1a77b83b43e6d475a6cfc79c232ddbca946ae9c7afc5',

'path': 'et/clips/common\_voice\_et\_18318995.mp3',

'audio': {

'path': 'et/clips/common\_voice\_et\_18318995.mp3',

'array': array([-0.00048828, -0.00018311, -0.00137329, ..., 0.00079346, 0.00091553, 0.00085449], dtype=float32),

'sampling\_rate': 48000

},

'sentence': 'Tasub kokku saada inimestega, keda tunned juba ammust ajast saati.',

'up\_votes': 2,

'down\_votes': 0,

'age': 'twenties',

'gender': 'male',

'accent': '',

'locale': 'et',

'segment': ''

}

```

##### 数据字段

- \*\*client\_id\*\* (string): 录音者的 ID

- \*\*path\*\* (string): 音频文件的路径

- \*\*audio\*\* (dict): 包含下载的音频文件路径、解码后的音频数组和采样率的字典

- \*\*sentence\*\* (string): 用户被提示说出的句子

- \*\*up\_votes\*\* (int64): 音频文件获得的点赞数

- \*\*down\_votes\*\* (int64): 音频文件获得的点踩数

- \*\*age\*\* (string): 说话者的年龄（例如，青少年，二十多岁，五十多岁）

- \*\*gender\*\* (string): 说话者的性别

- \*\*accent\*\* (string): 说话者的口音

- \*\*locale\*\* (string): 说话者的区域设置

- \*\*segment\*\* (string): 通常是空字段

#### 数据分割

语音材料已被细分为开发、训练、测试、验证、无效、报告和其他部分。

- \*\*验证数据\*\*：经过审核员验证并获得高质量上投票的数据。

- \*\*无效数据\*\*：被审核员判定为低质量数据并获得下投票的数据。

- \*\*报告数据\*\*：因各种原因被报告的数据。

- \*\*其他数据\*\*：尚未被审核的数据。

开发、测试、训练数据都是经过审核，且被认为质量高的数据，已被划分为开发、测试和训练集。

#### 数据集使用的其他考虑因素

- \*\*个人和敏感信息\*\*：数据集包含那些在网上捐赠自己声音的人的录音。您同意不尝试识别 Common Voice 数据集中的说话者身份。

#### 授权信息

公共领域，CC-0。

#### 引用信息

```

@inproceedings{commonvoice:2020,

author = {Ardila, R. and Branson, M. and Davis, K. and Henretty, M. and Kohler, M. and Meyer, J. and Morais, R. and Saunders, L. and Tyers, F. M. and Weber, G.},

title = {Common Voice: A Massively-Multilingual Speech Corpus},

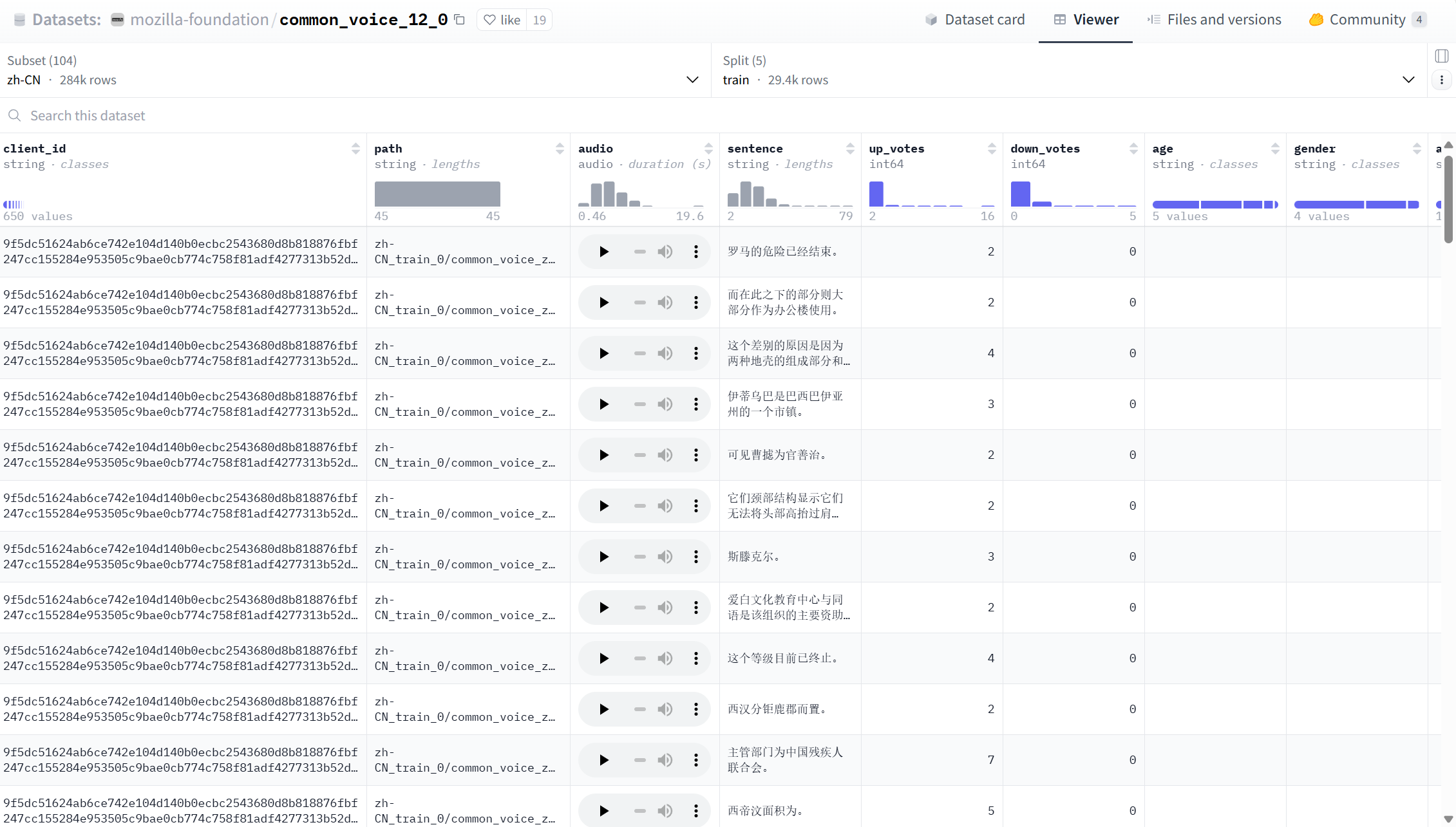
booktitle = {Proceedings of the 12th Conference on Language Resources and Evaluation (LREC 2020)},

pages = {4211--4215},

year = 2020

}

```



是的，您可以将数据集和模型下载到本地计算机，并在离线环境中进行推理或转换为 ONNX 格式。以下是如何进行的步骤：

### 1. 在线下载数据集和模型

首先，在有网络连接的电脑上下载所需的数据集和模型。

#### 下载数据集

使用 `datasets` 库在线下载数据集并保存到本地。

```python

from datasets import load\_dataset

# 使用 `use\_auth\_token=True` 登录后下载数据集

dataset = load\_dataset("mozilla-foundation/common\_voice\_12\_0", "as", use\_auth\_token=True)

# 保存到本地

dataset.save\_to\_disk("/path/to/save/dataset")

```

#### 下载预训练模型

假设您使用的是 Hugging Face 的 `transformers` 库训练了一个 ASR 模型，也可以通过以下方式下载模型并保存到本地。

```python

from transformers import AutoModelForSequenceClassification

# 下载预训练模型

model = AutoModelForSequenceClassification.from\_pretrained("model\_name")

# 保存模型到本地

model.save\_pretrained("/path/to/save/model")

```

### 2. 将数据集和模型转移到离线电脑

将下载好的数据集和模型复制到离线计算机中。可以使用 USB 驱动器、外部硬盘等介质进行传输。

### 3. 在离线电脑上加载数据集和模型

#### 加载数据集

```python

from datasets import load\_from\_disk

# 从本地加载数据集

dataset = load\_from\_disk("/path/to/save/dataset")

```

#### 加载模型

```python

from transformers import AutoModelForSequenceClassification

# 从本地加载模型

model = AutoModelForSequenceClassification.from\_pretrained("/path/to/save/model")

```

### 4. 转换模型为 ONNX 格式

安装 `onnx` 和 `onnxruntime` 库，然后使用 `transformers.onnx` 工具将 Hugging Face 模型转换为 ONNX 格式。

```bash

pip install onnx onnxruntime transformers[onnx]

```

#### 使用 `transformers` 工具转换模型

可以通过以下命令将模型转换为 ONNX 格式：

```bash

python -m transformers.onnx --model=/path/to/save/model /path/to/save/onnx\_model

```

#### 使用 ONNX Runtime 进行推理

安装 `onnxruntime` 后，可以在 Python 中进行推理：

```python

import onnxruntime as ort

import numpy as np

# 加载 ONNX 模型

onnx\_model = "/path/to/save/onnx\_model/model.onnx"

ort\_session = ort.InferenceSession(onnx\_model)

# 准备输入数据

input\_data = {"input\_ids": np.array([[101, 102]])} # 示例输入

# 推理

outputs = ort\_session.run(None, input\_data)

print(outputs)

```

### 总结

通过以上步骤，您可以在在线计算机上下载数据集和模型，然后将其移到离线计算机上进行推理，并转换为 ONNX 格式。这种方法允许您在没有互联网连接的环境中使用 Hugging Face 生态系统进行机器学习推理。

本文提供了一个使用 Hugging Face Transformers 在任意多语种语音识别 (ASR) 数据集上微调 Whisper 的分步指南。同时，我们还深入解释了 Whisper 模型、Common Voice 数据集以及微调等理论知识，并提供了数据准备和微调的相关代码。如果你想要一个全部是代码，仅有少量解释的 Notebook，可以参阅这个 **[Google Colab](https://link.zhihu.com/?target=https://colab.research.google.com/github/sanchit-gandhi/notebooks/blob/main/fine_tune_whisper.ipynb" \t "https://zhuanlan.zhihu.com/p/_blank)**。

## **目录**

1. **[简介](https://zhuanlan.zhihu.com/p/639993123/edit" \l "%E7%AE%80%E4%BB%8B)**
2. **[在 Google Colab 中微调 Whisper](https://zhuanlan.zhihu.com/p/639993123/edit" \l "%E5%9C%A8-google-colab-%E4%B8%AD%E5%BE%AE%E8%B0%83-whisper)**
   1. **[准备环境](https://zhuanlan.zhihu.com/p/639993123/edit" \l "%E5%87%86%E5%A4%87%E7%8E%AF%E5%A2%83)**
   2. **[加载数据集](https://zhuanlan.zhihu.com/p/639993123/edit" \l "%E5%8A%A0%E8%BD%BD%E6%95%B0%E6%8D%AE%E9%9B%86)**
   3. **[准备特征提取器、分词器和数据](https://zhuanlan.zhihu.com/p/639993123/edit" \l "%E5%87%86%E5%A4%87%E7%89%B9%E5%BE%81%E6%8F%90%E5%8F%96%E5%99%A8%E5%88%86%E8%AF%8D%E5%99%A8%E5%92%8C%E6%95%B0%E6%8D%AE)**
   4. **[训练与评估](https://zhuanlan.zhihu.com/p/639993123/edit" \l "%E8%AE%AD%E7%BB%83%E4%B8%8E%E8%AF%84%E4%BC%B0)**
   5. **[构建演示应用](https://zhuanlan.zhihu.com/p/639993123/edit" \l "%E6%9E%84%E5%BB%BA%E6%BC%94%E7%A4%BA%E5%BA%94%E7%94%A8)**
3. **[结束语](https://zhuanlan.zhihu.com/p/639993123/edit" \l "%E7%BB%93%E6%9D%9F%E8%AF%AD)**

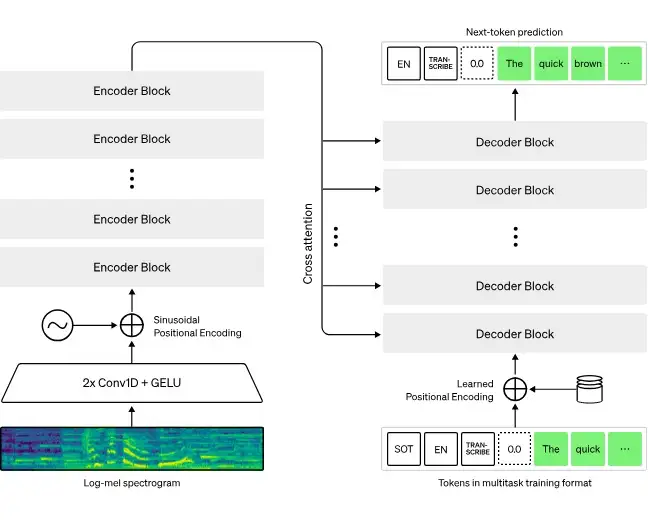
## **简介**

Whisper 是一系列用于自动语音识别 (automatic speech recognition，ASR) 的预训练模型，它由来自于 OpenAI 的 Alec Radford 等人于 **[2022 年 9 月](https://link.zhihu.com/?target=https://openai.com/blog/whisper/" \t "https://zhuanlan.zhihu.com/p/_blank)** 发布。与 **[Wav2Vec 2.0](https://link.zhihu.com/?target=https://arxiv.org/abs/2006.11477" \t "https://zhuanlan.zhihu.com/p/_blank)** 等前作不同，以往的模型都是在未标注的音频数据上预训练的，而 Whisper 是在大量的 **已标注** 音频转录数据上预训练的。其用于训练的标注音频时长高达 68 万小时，比 Wav2Vec 2.0 使用的未标注训练数据 (6 万小时) 还多一个数量级。更妙的是，该预训练数据中还含有 11.7 万小时的多语种数据。因此，Whisper 训得的 checkpoint 可应用于超过 96 种语言，这其中包含不少 *数据匮乏* 的小语种。

这么多的标注数据使得我们可以直接在 *有监督* 语音识别任务上预训练 Whisper，从标注音频转录数据 1 中直接习得语音到文本的映射。因此，Whisper 几乎不需要额外的微调就已经是高性能的 ASR 模型了。这让 Wav2Vec 2.0 相形见绌，因为 Wav2Vec 2.0 是在 *无监督* 掩码预测任务上预训练的，所以其训得的模型仅从未标注的纯音频数据中习得了从语音到隐含状态的[中间映射](https://zhida.zhihu.com/search?q=%E4%B8%AD%E9%97%B4%E6%98%A0%E5%B0%84&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank)。虽然无监督预训练能产生高质量的语音表征，但它 **学不到**语音到文本的映射，要学到语音到文本的映射只能靠微调。因此，Wav2Vec 2.0 需要更多的微调才能获得较有竞争力的性能。

在 68 万小时标注数据的加持下，预训练 Whisper 模型表现出了强大的泛化到多种数据集和领域的能力。其预训练 checkpoint 表现出了与最先进的 ASR 系统旗鼓相当的性能: 在 LibriSpeech ASR 的[无噪测试](https://zhida.zhihu.com/search?q=%E6%97%A0%E5%99%AA%E6%B5%8B%E8%AF%95&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank)子集上的单词错误率 (word error rate，WER) 仅为约 3%，另外它还在 [TED-LIUM](https://zhida.zhihu.com/search?q=TED-LIUM&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank) 上创下了新的记录 - 4.7% 的 WER ( *详见* **[Whisper 论文](https://link.zhihu.com/?target=https://cdn.openai.com/papers/whisper.pdf" \t "https://zhuanlan.zhihu.com/p/_blank)** 的表 8)。Whisper 在预训练期间获得的广泛的多语种 ASR 知识对一些数据匮乏的小语种特别有用。稍稍微调一下，预训练 checkpoint 就可以进一步适配特定的数据集和语种，从而进一步改进在这些语种上的识别效果。

Whisper 是一个基于 transformer 的编码器 - 解码器模型 (也称为 *序列到序列* 模型)，它将音频的[频谱图](https://zhida.zhihu.com/search?q=%E9%A2%91%E8%B0%B1%E5%9B%BE&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank)特征 *序列* 映射到文本的词 \_序列\_。首先，通过特征提取器将原始音频输入变换为[对数梅尔声谱图](https://zhida.zhihu.com/search?q=%E5%AF%B9%E6%95%B0%E6%A2%85%E5%B0%94%E5%A3%B0%E8%B0%B1%E5%9B%BE&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank) (log-Mel spectrogram)。然后，transformer 编码器对声谱图进行编码，生成一系列编码器隐含状态。最后，解码器基于先前输出的词以及编码器隐含状态，自回归地预测下一个输出词。图 1 是 Whisper 模型的示意图。



**图 1:** Whisper 模型，该模型是标准的基于 transformer 的编码器-解码器架构。首先将对数梅尔声谱图输入到编码器，然后将编码器生成的最终隐含状态通过[交叉注意机制](https://zhida.zhihu.com/search?q=%E4%BA%A4%E5%8F%89%E6%B3%A8%E6%84%8F%E6%9C%BA%E5%88%B6&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank)输入给解码器。最后，解码器基于编码器隐含状态和先前的输出词，自回归地预测下一个输出词。图源: **[OpenAI Whisper 博客](https://link.zhihu.com/?target=https://openai.com/blog/whisper/" \t "https://zhuanlan.zhihu.com/p/_blank)**。

在[序列到序列模型](https://zhida.zhihu.com/search?q=%E5%BA%8F%E5%88%97%E5%88%B0%E5%BA%8F%E5%88%97%E6%A8%A1%E5%9E%8B&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank)中，编码器负责从语音中提取出重要特征，将输入转换为一组隐含状态表征。解码器扮演语言模型的角色，处理隐含状态表征并生成对应的文本。我们把在模型架构 **内部** 集成语言模型的做法称为 \_深度融合\_。与之相对的是 \_浅融合\_，此时，语言模型在 **外部**与编码器组合，如 CTC + n-gram ( *详见* **[Internal Language Model Estimation](https://link.zhihu.com/?target=https://arxiv.org/pdf/2011.01991.pdf" \t "https://zhuanlan.zhihu.com/p/_blank)** 一文)。通过深度融合，可以用同一份训练数据和[损失函数](https://zhida.zhihu.com/search?q=%E6%8D%9F%E5%A4%B1%E5%87%BD%E6%95%B0&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank)对整个系统进行端到端训练，从而获得更大的灵活性和更优越的性能 ( *详见* **[ESB Benchmark](https://link.zhihu.com/?target=https://arxiv.org/abs/2210.13352" \t "https://zhuanlan.zhihu.com/p/_blank)**)。

Whisper 使用[交叉熵目标函数](https://zhida.zhihu.com/search?q=%E4%BA%A4%E5%8F%89%E7%86%B5%E7%9B%AE%E6%A0%87%E5%87%BD%E6%95%B0&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank)进行预训练和微调，交叉熵目标函数是训练序列标注模型的[标准目标函数](https://zhida.zhihu.com/search?q=%E6%A0%87%E5%87%86%E7%9B%AE%E6%A0%87%E5%87%BD%E6%95%B0&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank)。经过训练，模型可以正确地对目标词进行分类，从而从预定义的词汇表中选出输出词。

Whisper 有五种不同尺寸的 checkpoint。其中，四个小尺寸 checkpoint 又各有两个版本: 英语版和多语种版，而最大的 checkpoint 只有多语种版。所有九个预训练 [checkpoints](https://zhida.zhihu.com/search?q=checkpoints&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank) 都可以在 **[Hugging Face Hub](https://link.zhihu.com/?target=https://huggingface.co/models?search=openai/whisper" \t "https://zhuanlan.zhihu.com/p/_blank)** 上找到。下表总结了这些 checkpoint 的信息及其 Hub 链接:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 尺寸 | 层数 | 宽 | 多头注意力的头数 | 参数量 | 英语 checkpoint | 多语种 checkpoint |
| tiny | 4 | 384 | 6 | 39 M | ✓ | ✓ |
| base | 6 | 512 | 8 | 74 M | ✓ | ✓ |
| small | 12 | 768 | 12 | 244 M | ✓ | ✓ |
| medium | 24 | 1024 | 16 | 769 M | ✓ | ✓ |
| large | 32 | 1280 | 20 | 1550 M | x | ✓ |

下面，我们将以多语种版的 **[small](https://link.zhihu.com/?target=https://huggingface.co/openai/whisper-small" \t "https://zhuanlan.zhihu.com/p/_blank)**checkpoint (参数量 244M (~= 1GB)) 为例，带大家走一遍微调模型的全过程。我们将使用 **[Common Voice](https://link.zhihu.com/?target=https://huggingface.co/datasets/mozilla-foundation/common_voice_11_0" \t "https://zhuanlan.zhihu.com/p/_blank)** 数据集里的小语种数据来训练和评估我们的系统。通过这个例子，我们将证明，仅需 8 小时的训练数据就可以微调出一个在该语种上表现强大的[语音识别模型](https://zhida.zhihu.com/search?q=%E8%AF%AD%E9%9F%B3%E8%AF%86%E5%88%AB%E6%A8%A1%E5%9E%8B&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank)。

1 Whisper 的名称来自于 “Web-scale Supervised Pre-training for Speech Recognition (网络规模的有监督[语音识别预训练模型](https://zhida.zhihu.com/search?q=%E8%AF%AD%E9%9F%B3%E8%AF%86%E5%88%AB%E9%A2%84%E8%AE%AD%E7%BB%83%E6%A8%A1%E5%9E%8B&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank))” 的首字母缩写 “WSPSR”。

## **在 Google Colab 中微调 Whisper**

### **准备环境**

在微调 Whisper 模型时，我们会用到几个流行的 Python 包。我们使用 datasets 来下载和准备训练数据，使用 transformers 来加载和训练 Whisper 模型。另外，我们还需要 [soundfile](https://zhida.zhihu.com/search?q=soundfile&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank) 包来预处理音频文件，evaluate 和 jiwer 来评估模型的性能。最后，我们用 gradio 来为微调后的模型构建一个亮闪闪的演示应用。

!pip install datasets>=2.6.1

!pip install git+https://github.com/huggingface/transformers

!pip install librosa

!pip install evaluate>=0.30

!pip install jiwer

!pip install gradio

我们强烈建议你直接将训得的模型 checkpoint 上传到 **[Hugging Face Hub](https://link.zhihu.com/?target=https://huggingface.co/" \t "https://zhuanlan.zhihu.com/p/_blank)**。Hub 提供了以下功能:

* [集成版本控制](https://zhida.zhihu.com/search?q=%E9%9B%86%E6%88%90%E7%89%88%E6%9C%AC%E6%8E%A7%E5%88%B6&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank): 确保在训练期间不会丢失任何模型 checkpoint。
* Tensorboard 日志: 跟踪训练过程中的重要指标。
* 模型卡: 记录模型的用法及其应用场景。
* 社区: 轻松与社区进行分享和协作！

将 Python notebook 连上 Hub 非常简单 - 只需根据提示输入你的 Hub 身份验证令牌即可。你可以在 **[此处](https://link.zhihu.com/?target=https://huggingface.co/settings/tokens" \t "https://zhuanlan.zhihu.com/p/_blank)** 找到你自己的 Hub [身份验证令牌](https://zhida.zhihu.com/search?q=%E8%BA%AB%E4%BB%BD%E9%AA%8C%E8%AF%81%E4%BB%A4%E7%89%8C&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank):

**from** huggingface\_hub **import** notebook\_login

notebook\_login()

**打印输出:**

Login successful

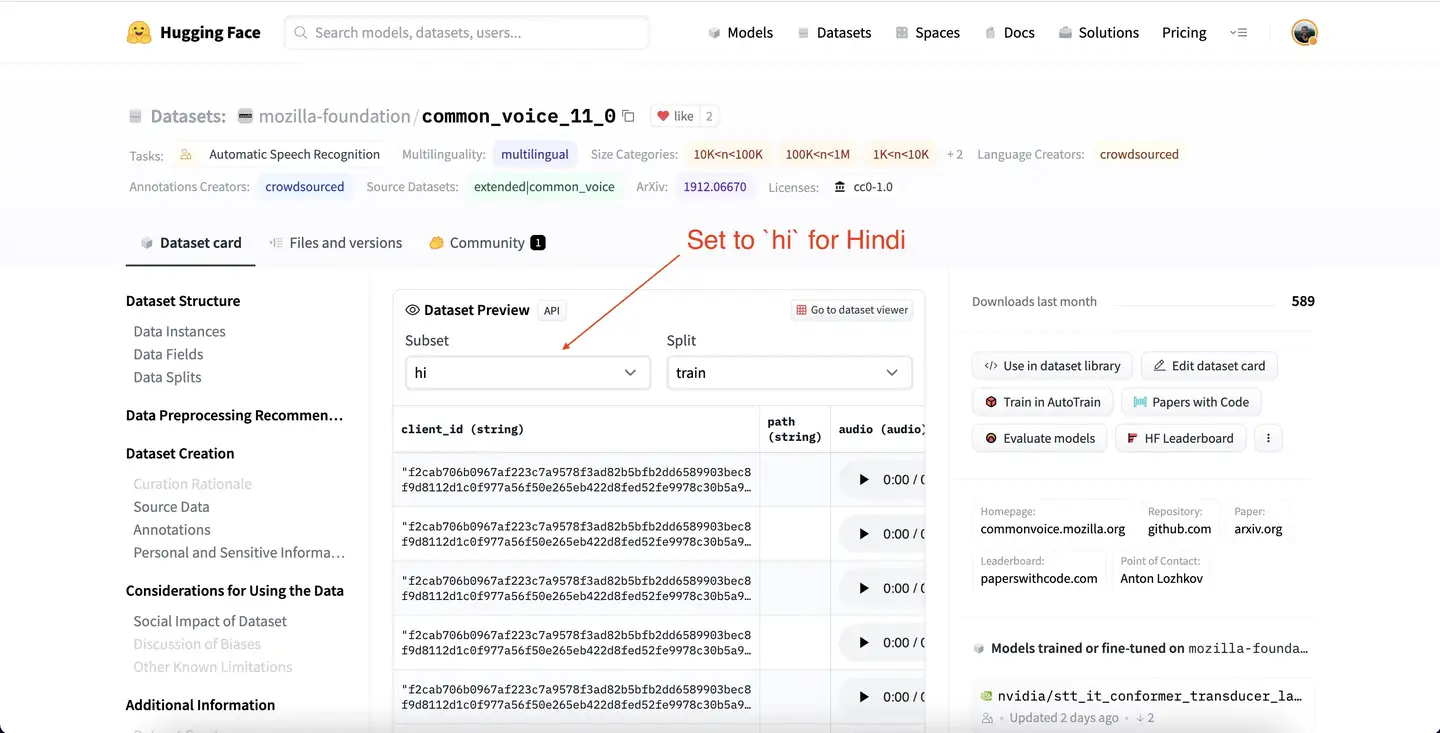
Your token has been saved to /root/.huggingface/token

### **加载数据集**

Common Voice 由一系列众包数据集组成，其中包含了用各种语言录制的维基百科文本。本文使用的是最新版本的 Common Voice 数据集 (**[版本号为 11](https://link.zhihu.com/?target=https://huggingface.co/datasets/mozilla-foundation/common_voice_11_0" \t "https://zhuanlan.zhihu.com/p/_blank)**)。语种上，我们选择用 ***[印地语](https://link.zhihu.com/?target=https://en.wikipedia.org/wiki/Hindi" \t "https://zhuanlan.zhihu.com/p/_blank)*** 来微调我们的模型。印地语是一种在印度北部、中部、东部和西部使用的印度 - 雅利安语。Common Voice 11.0 中有大约 12 小时的标注印地语数据，其中 4 小时是测试数据。

我们先看下 Hub 上的 Common Voice 数据集页面: **[mozilla-foundation/common\_voice\_11\_0](https://link.zhihu.com/?target=https://huggingface.co/datasets/mozilla-foundation/common_voice_11_0" \t "https://zhuanlan.zhihu.com/p/_blank)**。如果你是首次查看此页面，系统会要求你接受其使用条款，同意后就可以访问数据集了。

一旦身份验证成功，你就会看到数据集预览。[数据集预览](https://zhida.zhihu.com/search?q=%E6%95%B0%E6%8D%AE%E9%9B%86%E9%A2%84%E8%A7%88&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank)展示了数据集的前 100 个样本。更重要的是，它还加载了可供实时收听的音频。我们可以在下拉菜单选择 hi 来选择 Common Voice 的印地语子集 ( hi 是印地语的语言标识符代码):



点击第一个音频的播放按钮，你就可以收听音频并看到相应的文本了。你还可以滚动浏览训练集和测试集中的样本，以更好地了解待处理音频和文本数据。从语调和风格可以看出，这些音频是旁白录音。你可能还会注意到录音者和录音质量的巨大差异，这是[众包数据](https://zhida.zhihu.com/search?q=%E4%BC%97%E5%8C%85%E6%95%B0%E6%8D%AE&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank)的一个共同特征。

使用 Datasets 来下载和准备数据非常简单。仅需一行代码即可完成 Common Voice 数据集的下载和准备工作。由于印地语数据非常匮乏，我们把 训练集 和 验证集合并成约 8 小时的训练数据，而测试则基于 4 小时的 测试集:

**from** datasets **import** load\_dataset, DatasetDict

common\_voice **=** DatasetDict()

common\_voice["train"] **=** load\_dataset("mozilla-foundation/common\_voice\_11\_0", "hi", split**=**"train+validation", use\_auth\_token**=**True)common\_voice["test"] **=** load\_dataset("mozilla-foundation/common\_voice\_11\_0", "hi", split**=**"test", use\_auth\_token**=**True)

**print**(common\_voice)

**打印输出:**

DatasetDict({

train: Dataset({

features: ['client\_id', 'path', 'audio', 'sentence', 'up\_votes', 'down\_votes', 'age', 'gender', 'accent', 'locale', 'segment'],

num\_rows: 6540

})

test: Dataset({

features: ['client\_id', 'path', 'audio', 'sentence', 'up\_votes', 'down\_votes', 'age', 'gender', 'accent', 'locale', 'segment'],

num\_rows: 2894

})

})

大多数 ASR 数据集仅包含输入[音频样本](https://zhida.zhihu.com/search?q=%E9%9F%B3%E9%A2%91%E6%A0%B7%E6%9C%AC&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank) ( audio) 和相应的转录文本 ( sentence)。 Common Voice 还包含额外的元信息，例如 accent 和 locale，在 ASR 场景中，我们可以忽略这些信息。为了使代码尽可能通用，我们只考虑基于输入音频和转录文本进行微调，而不使用额外的元信息:

common\_voice = common\_voice.remove\_columns(["accent", "age", "client\_id", "down\_votes", "gender", "locale", "path", "segment", "up\_votes"])

除了 Common Voice，Hub 上还有不少其他多语种 ASR 数据集可供使用，你可以点击链接: **[Hub 上的 ASR 数据集](https://link.zhihu.com/?target=https://huggingface.co/datasets?task_categories=task_categories:automatic-speech-recognition&sort=downloads" \t "https://zhuanlan.zhihu.com/p/_blank)** 了解更多。

### **准备特征提取器、分词器和数据**

ASR 的流水线主要包含三个模块:

1. 对原始音频输入进行预处理的特征提取器
2. 执行序列到[序列映射](https://zhida.zhihu.com/search?q=%E5%BA%8F%E5%88%97%E6%98%A0%E5%B0%84&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank)的模型
3. 将模型输出转换为文本的分词器

在 Transformers 中，Whisper 模型有自己的特征提取器和分词器，即 **[WhisperFeatureExtractor](https://link.zhihu.com/?target=https://huggingface.co/docs/transformers/main/model_doc/whisper%23transformers.WhisperFeatureExtractor" \t "https://zhuanlan.zhihu.com/p/_blank)** 和 **[WhisperTokenizer](https://link.zhihu.com/?target=https://huggingface.co/docs/transformers/main/model_doc/whisper%23transformers.WhisperTokenizer" \t "https://zhuanlan.zhihu.com/p/_blank)**。

下面，我们逐一详细介绍特征提取器和分词器！

### **加载 WhisperFeatureExtractor**

语音可表示为随时间变化的一维数组，给定时刻的数组值即表示信号在该时刻的 \_幅度\_，而我们可以仅从幅度信息重建音频的频谱并恢复其所有声学特征。

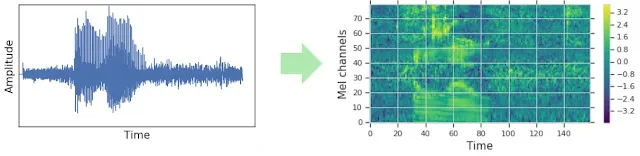
由于语音是连续的，因此它包含无数个幅度值，而计算机只能表示并存储有限个值。因此，我们需要通过对语音信号进行[离散化](https://zhida.zhihu.com/search?q=%E7%A6%BB%E6%95%A3%E5%8C%96&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank)，即以固定的时间间隔对连续信号进行 \_采样\_。我们将每秒采样的次数称为 \_采样率\_，通常以样本数/秒或 *赫兹 (Hz)* 为单位。高采样率可以更好地逼近连续语音信号，但同时每秒所需的存储量也更大。

需要特别注意的是，输入音频的采样率需要与模型期望的采样率相匹配，因为不同采样率的音频信号的分布是不同的。处理音频时，需要使用正确的采样率，否则可能会引起意想不到的结果！例如，以 16kHz 的采样率采集音频但以 8kHz 的采样率收听它，会使音频听起来好像是半速的。同样地，向一个需要某一采样率的 ASR 模型馈送一个错误采样率的音频也会影响模型的性能。Whisper 特征提取器需要采样率为 16kHz 的音频输入，因此输入的采样率要与之相匹配。我们不想无意中用慢速语音来训练 ASR！

Whisper 特征提取器执行两个操作。首先，填充或截断一批音频样本，将所有样本的输入长度统一至 30 秒。通过在序列末尾添加零 (音频信号中的零对应于无信号或静音)，将短于 30 秒的样本填充到 30 秒。而对超过 30 秒的样本，直接截断为 30 秒就好了。由于这一批数据中的所有样本都被填充或截断到统一长度 (即 30 s) 了，因此将音频馈送给 Whisper 模型时就不需要注意力掩码了。这是 Whisper 的独门特性，其他大多数音频模型都需要用户提供一个注意力掩码，详细说明填充位置，这样模型才能在自[注意力机制](https://zhida.zhihu.com/search?q=%E6%B3%A8%E6%84%8F%E5%8A%9B%E6%9C%BA%E5%88%B6&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank)中忽略填充部分。经过训练的 Whisper 模型可以直接从语音信号中推断出应该忽略哪些部分，因此无需注意力掩码。

Whisper 特征提取器执行的第二个操作是将第一步所得的音频变换为对数梅尔声谱图。这些频谱图是信号频率的直观表示，类似于傅里叶变换。图 2 展示了一个声谱图的例子，其中 y 轴表示梅尔频段 (Mel channel)，对应于特定的频段，x 轴表示时间，颜色对应于给定时刻该频段的对数强度。Whisper 模型要求输入为对数梅尔声谱图。

梅尔频段是语音处理的标准方法，研究人员用它来近似表示人类的听觉范围。对于 Whisper 微调这个任务而言，我们只需要知道声谱图是语音信号中频率的直观表示。更多有关梅尔频段的详细信息，请参阅 **[梅尔倒谱](https://link.zhihu.com/?target=https://en.wikipedia.org/wiki/Mel-frequency_cepstrum" \t "https://zhuanlan.zhihu.com/p/_blank)** 一文。



**图 2：** 将音频信号变换为对数梅尔声谱图。左图：一维音频离散信号。右图：对应的对数梅尔声谱图。图源：**[谷歌 SpecAugment 博文](https://link.zhihu.com/?target=https://ai.googleblog.com/2019/04/specaugment-new-data-augmentation.html" \t "https://zhuanlan.zhihu.com/p/_blank)**.

幸运的是， Transformers Whisper 特征提取器仅用一行代码即可执行填充和声谱图变换两个操作！我们使用以下代码从预训练的 checkpoint 中加载特征提取器，为音频数据处理做好准备:

**from** transformers **import** WhisperFeatureExtractor

feature\_extractor **=** WhisperFeatureExtractor**.**from\_pretrained("openai/whisper-small")

### **加载 WhisperTokenizer**

现在我们加载 Whisper 分词器。Whisper 模型会输出词元，这些词元表示预测文本在词典中的索引。分词器负责将这一系列词元映射为最终的文本字符串 (例如 [1169, 3797, 3332] -> “the cat sat”)。

过去，当使用编码器模型进行 ASR 时，我们需使用 ***="[https://distill.pub/2017/ctc/](https://link.zhihu.com/?target=https://distill.pub/2017/ctc/" \t "https://zhuanlan.zhihu.com/p/_blank)">连接时序分类法 (Connectionist Temporal Classification，CTC)***进行解码。在使用 CTC 进行解码时，我们需要为每个数据集训练一个 CTC 分词器。但使用编码器 - 解码器架构的一个优势是我们可以直接使用预训练模型的分词器。

Whisper 分词器在 96 种语种数据上预训练而得，因此，其 **[字节对 (byte-pair)](https://link.zhihu.com/?target=https://huggingface.co/course/chapter6/5?fw=pt%23bytepair-encoding-tokenization" \t "https://zhuanlan.zhihu.com/p/_blank)**覆盖面很广，几乎包含了所有语种。就印地语而言，我们可以加载分词器并将其直接用于微调。仅需指定一下目标语种和任务，分词器就会根据这些参数将语种和任务标记添加为输出序列的前缀:

**from** transformers **import** WhisperTokenizer

tokenizer **=** WhisperTokenizer**.**from\_pretrained("openai/whisper-small", language**=**"Hindi", task**=**"transcribe")

我们可以通过对 Common Voice 数据集的第一个样本进行编解码来验证分词器是否正确编码了印地语字符。在对转录文本进行编码时，分词器在序列的开头和结尾添加“特殊标记”，其中包括文本的开始/结尾、语种标记和任务标记 (由上一步中的参数指定)。在解码时，我们可以选择“跳过”这些特殊标记，从而保证输出是纯文本形式的:

input\_str **=** common\_voice["train"][0]["sentence"]labels **=** tokenizer(input\_str)**.**input\_idsdecoded\_with\_special **=** tokenizer**.**decode(labels, skip\_special\_tokens**=**False)decoded\_str **=** tokenizer**.**decode(labels, skip\_special\_tokens**=**True)

**print**(f"Input: {input\_str}")**print**(f"Decoded w/ special: {decoded\_with\_special}")**print**(f"Decoded w/out special: {decoded\_str}")**print**(f"Are equal: {input\_str == decoded\_str}")

**打印输出:**

Input: खीर की मिठास पर गरमाई बिहार की सियासत, कुशवाहा ने दी सफाईDecoded w**/** special: **<|**startoftranscript**|><|**hi**|><|**transcribe**|><|**notimestamps**|>**खीर की मिठास पर गरमाई बिहार की सियासत, कुशवाहा ने दी सफाई**<|**endoftext**|>**Decoded w**/**out special: खीर की मिठास पर गरमाई बिहार की सियासत, कुशवाहा ने दी सफाईAre equal: True

### **组装一个 WhisperProcessor**

为了简化使用，我们可以将特征提取器和分词器 *包进* 到一个 WhisperProcessor 类，该类继承自 WhisperFeatureExtractor 及 WhisperTokenizer，可根据需要用于音频处理和模型预测。有了它，我们在训练期间只需要保留两个对象: processor 和 model 就好了。

**from** transformers **import** WhisperProcessor

processor **=** WhisperProcessor**.**from\_pretrained("openai/whisper-small", language**=**"Hindi", task**=**"transcribe")

### **准备数据**

我们把 Common Voice 数据集的第一个样本打印出来，看看数据长什么样:

**print**(common\_voice["train"][0])

**打印输出:**

{'audio': {'path': '/home/sanchit\_huggingface\_co/.cache/huggingface/datasets/downloads/extracted/607848c7e74a89a3b5225c0fa5ffb9470e39b7f11112db614962076a847f3abf/cv-corpus-11.0-2022-09-21/hi/clips/common\_voice\_hi\_25998259.mp3',

'array': array([0.0000000e+00, 0.0000000e+00, 0.0000000e+00, **...**, 9.6724887e-07,

1.5334779e-06, 1.0415988e-06], dtype**=**float32),

'sampling\_rate': 48000},

'sentence': 'खीर की मिठास पर गरमाई बिहार की सियासत, कुशवाहा ने दी सफाई'}

可以看到，样本含有一个一维音频数组及其对应的转录文本。上文已经多次谈及采样率，以及将音频的采样率与 Whisper 模型所需的采样率 (16kHz) 相匹配的重要性。由于现在输入音频的采样率为 48kHz，所以在将其馈送给 Whisper 特征提取器之前，我们需要将其 \_[下采样](https://zhida.zhihu.com/search?q=%E4%B8%8B%E9%87%87%E6%A0%B7&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank)\_至 16kHz。

我们将使用 dataset 的 **[cast\_column](https://link.zhihu.com/?target=https://huggingface.co/docs/datasets/package_reference/main_classes.html?highlight=cast_column%23datasets.DatasetDict.cast_column" \t "https://zhuanlan.zhihu.com/p/_blank)** 方法将输入音频转换至所需的采样率。该方法仅指示 datasets 让其在首次加载音频时 \_即时地\_对数据进行重采样，因此并不会改变原音频数据:

**from** datasets **import** Audio

common\_voice **=** common\_voice**.**cast\_column("audio", Audio(sampling\_rate**=**16000))

重新打印下 Common Voice 数据集中的第一个音频样本，可以看到其已被重采样:

**print**(common\_voice["train"][0])

**打印输出:**

{'audio': {'path': '/home/sanchit\_huggingface\_co/.cache/huggingface/datasets/downloads/extracted/607848c7e74a89a3b5225c0fa5ffb9470e39b7f11112db614962076a847f3abf/cv-corpus-11.0-2022-09-21/hi/clips/common\_voice\_hi\_25998259.mp3',

'array': array([ 0.0000000e+00, 0.0000000e+00, 0.0000000e+00, **...**,

**-**3.4206650e-07, 3.2979898e-07, 1.0042874e-06], dtype**=**float32),

'sampling\_rate': 16000},

'sentence': 'खीर की मिठास पर गरमाई बिहार की सियासत, कुशवाहा ने दी सफाई'}

酷！我们可以看到音频已被下采样到 16kHz 了。数组里面的值也变了，现在的 1 个幅度值大致对应于之前的 3 个幅度值。

现在我们编写一个函数来为模型准备数据:

1. 调用 batch["audio"] 加载和重采样音频数据。如上所述， Datasets 会即时执行任何必要的重采样操作。
2. 使用特征提取器将一维音频数组变换为对数梅尔声谱图特征。
3. 使用分词器将录音文本编码为 ID。

**def** **prepare\_dataset**(batch):

*# load and resample audio data from 48 to 16kHz*

audio **=** batch["audio"]

*# compute log-Mel input features from input audio array*

batch["input\_features"] **=** feature\_extractor(audio["array"], sampling\_rate**=**audio["sampling\_rate"])**.**input\_features[0]

*# encode target text to label ids*

batch["labels"] **=** tokenizer(batch["sentence"])**.**input\_ids

**return** batch

我们可以用 dataset 的 .map 方法在所有训练样本上应用上述函数:

common\_voice **=** common\_voice**.**map(prepare\_dataset, remove\_columns**=**common\_voice**.**column\_names["train"], num\_proc**=**4)

好了！训练数据准备完毕！我们继续看看如何使用这些数据来微调 Whisper。

**注意**: 目前 datasets 主要使用 **[torchaudio](https://link.zhihu.com/?target=https://pytorch.org/audio/stable/index.html" \t "https://zhuanlan.zhihu.com/p/_blank)** 和 [librosa]([https://librosa.org](https://link.zhihu.com/?target=https://librosa.org" \t "https://zhuanlan.zhihu.com/p/_blank) /doc/latest/index.html) 来进行音频加载和重采样。如果你自己定制一个数据加载/采样函数的话，你完全可以直接通过 "path" 列获取音频文件路径而不用管 "audio" 列。

## **训练与评估**

至此，数据已准备完毕，可以开始训练了。训练的大部分繁重的工作都会由 **[Trainer](https://link.zhihu.com/?target=https://huggingface.co/transformers/master/main_classes/trainer.html?highlight=trainer" \t "https://zhuanlan.zhihu.com/p/_blank)** 来完成。我们要做的主要有:

* 定义数据整理器 (data collator): 数据整理器获取预处理后的数据并将其转换为 PyTorch 张量。
* 评估指标: 我们使用 **[单词错误率 (word error rate，WER)](https://link.zhihu.com/?target=https://huggingface.co/metrics/wer" \t "https://zhuanlan.zhihu.com/p/_blank)** 指标来评估模型，因此需要定义一个 compute\_metrics 函数来计算它。
* 加载预训练 checkpoint: 我们需要加载预训练 checkpoint 并正确配置它以进行训练。
* 定义训练参数: Trainer 在制订训练计划时需要用到这些参数。

微调完后，我们需要使用测试数据对其进行评估，以验证最终模型在印地语上的语音识别效果。

### **定义数据整理器**

序列到序列[语音模型](https://zhida.zhihu.com/search?q=%E8%AF%AD%E9%9F%B3%E6%A8%A1%E5%9E%8B&zhida_source=entity&is_preview=1" \t "https://zhuanlan.zhihu.com/p/_blank)的数据整理器与其他任务有所不同，因为 input\_features 和 labels 的处理方法是不同的: input\_features 必须由特征提取器处理，而 labels 由分词器处理。

input\_features 已经填充至 30s 并转换为固定维度的对数梅尔声谱图，我们所要做的只剩将其转换为 PyTorch 张量。我们用特征提取器的 .pad 方法来完成这一功能，且将其入参设为 return\_tensors=pt。请注意，这里不需要额外的填充，因为输入维度已经固定了，所以我们只需要简单地将 input\_features 转换为 PyTorch 张量就好了。

另一方面，labels 数据之前并未填充。所以，我们首先要使用分词器的 .pad 方法将序列填充至本 batch 的最大长度。然后将填充标记替换为 -100，这样它们就可以 **不** 用参与损失的计算了。然后我们把 SOT 从序列的开头去掉，稍后训练的时候我们再把它加回来。

我们可以利用之前定义的 WhisperProcessor 来执行特征提取和分词操作:

**import** torch

**from** dataclasses **import** dataclass**from** typing **import** Any, Dict, List, Union

@dataclass**class** **DataCollatorSpeechSeq2SeqWithPadding**:

processor: Any

**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*

*# first treat the audio inputs by simply returning torch tensors*

input\_features **=** [{"input\_features": feature["input\_features"]} **for** feature **in** features]

batch **=** self**.**processor**.**feature\_extractor**.**pad(input\_features, return\_tensors**=**"pt")

*# get the tokenized label sequences*

label\_features **=** [{"input\_ids": feature["labels"]} **for** feature **in** features]

*# pad the labels to max length*

labels\_batch **=** self**.**processor**.**tokenizer**.**pad(label\_features, 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)

*# if bos token is appended in previous tokenization step,*

*# cut bos token here as it's append later anyways*

**if** (labels[:, 0] **==** self**.**processor**.**tokenizer**.**bos\_token\_id)**.**all()**.**cpu()**.**item():

labels **=** labels[:, 1:]

batch["labels"] **=** labels

**return** batch

我们初始化一下刚刚定义的数据整理器:

data\_collator **=** DataCollatorSpeechSeq2SeqWithPadding(processor**=**processor)

### **评估指标**

接下来要定义评估指标。我们将使用词错误率 (WER) 指标，它是评估 ASR 系统的“标准”指标。有关其详细信息，请参阅 WER **[文档](https://link.zhihu.com/?target=https://huggingface.co/metrics/wer" \t "https://zhuanlan.zhihu.com/p/_blank)**。下面，我们从 Evaluate 中加载 WER 指标:

**import** evaluate

metric **=** evaluate**.**load("wer")

然后我们只需要定义一个函数来接受模型输出并返回 WER 指标。这个名为 compute\_metrics 的函数首先将 -100 替换为 label\_ids 中的 pad\_token\_id (以便在计算损失时将其忽略)。然后，将预测到的 ID 和 label\_ids 解码为字符串文本。最后，计算输出文本和真实文本之间的 WER:

**def** **compute\_metrics**(pred):

pred\_ids **=** pred**.**predictions

label\_ids **=** pred**.**label\_ids

*# replace -100 with the pad\_token\_id*

label\_ids[label\_ids **==** **-**100] **=** tokenizer**.**pad\_token\_id

*# we do not want to group tokens when computing the metrics*

pred\_str **=** tokenizer**.**batch\_decode(pred\_ids, skip\_special\_tokens**=**True)

label\_str **=** tokenizer**.**batch\_decode(label\_ids, skip\_special\_tokens**=**True)

wer **=** 100 **\*** metric**.**compute(predictions**=**pred\_str, references**=**label\_str)

**return** {"wer": wer}

### **加载预训练 checkpoint**

现在我们加载预训练 Whisper small 模型的 checkpoint。同样，可以通过使用 transformers 很轻松地完成这一步！

**from** transformers **import** WhisperForConditionalGeneration

model **=** WhisperForConditionalGeneration**.**from\_pretrained("openai/whisper-small")

原始 Whisper 模型在自回归生成开始之前强制添加了若干前缀词元 ID (**[forced\_decoder\_ids](https://link.zhihu.com/?target=https://huggingface.co/docs/transformers/main_classes/text_generation%23transformers.generation_utils.GenerationMixin.generate.forced_decoder_ids" \t "https://zhuanlan.zhihu.com/p/_blank)**)。这些词元 ID 主要用于在零样本 ASR 任务中标识语种和任务。因为我们现在是对已知语种 (印地语) 和任务 (转录) 进行微调，所以我们要将 forced\_decoder\_ids 设置为 None。另外，模型还抑制了一些词元 (**[suppress\_tokens](https://link.zhihu.com/?target=https://huggingface.co/docs/transformers/main_classes/text_generation%23transformers.generation_utils.GenerationMixin.generate.suppress_tokens" \t "https://zhuanlan.zhihu.com/p/_blank)**)，这些词元的对数概率被强置为 -inf，以保证它们永远不会被采样到。我们会用一个空列表覆盖 suppress\_tokens，即我们不抑制任何词元:

model**.**config**.**forced\_decoder\_ids **=** Nonemodel**.**config**.**suppress\_tokens **=** []

### **定义训练参数**

最后一步是定义与训练相关的所有参数，下面对其中一部分参数进行了解释:

* output\_dir: 保存模型权重的本地目录，它也会是 **[Hugging Face Hub](https://link.zhihu.com/?target=https://huggingface.co/" \t "https://zhuanlan.zhihu.com/p/_blank)** 上的模型存储库名称。
* generation\_max\_length: 评估阶段，自回归生成的最大词元数。
* save\_steps: 训练期间，每 save\_steps 步保存一次中间 checkpoint 并异步上传到 Hub。
* eval\_steps: 训练期间，每 eval\_steps 步对中间 checkpoint 进行一次评估。
* report\_to: 训练日志的保存位置，支持 azure\_ml 、comet\_ml 、mlflow 、neptune 、tensorboard 以及 wandb 这些平台。你可以按照自己的偏好进行选择，也可以直接使用缺省的 tensorboard 保存至 Hub。

如需更多其他训练参数的详细信息，请参阅 Seq2SeqTrainingArguments **[文档](https://link.zhihu.com/?target=https://huggingface.co/docs/transformers/main_classes/trainer%23transformers.Seq2SeqTrainingArguments" \t "https://zhuanlan.zhihu.com/p/_blank)**。

**from** transformers **import** Seq2SeqTrainingArguments

training\_args **=** Seq2SeqTrainingArguments(

output\_dir**=**"./whisper-small-hi", *# change to a repo name of your choice*

per\_device\_train\_batch\_size**=**16,

gradient\_accumulation\_steps**=**1, *# increase by 2x for every 2x decrease in batch size*

learning\_rate**=**1e-5,

warmup\_steps**=**500,

max\_steps**=**4000,

gradient\_checkpointing**=**True,

fp16**=**True,

evaluation\_strategy**=**"steps",

per\_device\_eval\_batch\_size**=**8,

predict\_with\_generate**=**True,

generation\_max\_length**=**225,

save\_steps**=**1000,

eval\_steps**=**1000,

logging\_steps**=**25,

report\_to**=**["tensorboard"],

load\_best\_model\_at\_end**=**True,

metric\_for\_best\_model**=**"wer",

greater\_is\_better**=**False,

push\_to\_hub**=**True,)

**注意**: 如果不想将模型 checkpoint 上传到 Hub，你需要设置 push\_to\_hub=False。

我们可以将训练参数以及模型、数据集、数据整理器和 compute\_metrics 函数一起传给 Trainer:

**from** transformers **import** Seq2SeqTrainer

trainer **=** Seq2SeqTrainer(

args**=**training\_args,

model**=**model,

train\_dataset**=**common\_voice["train"],

eval\_dataset**=**common\_voice["test"],

data\_collator**=**data\_collator,

compute\_metrics**=**compute\_metrics,

tokenizer**=**processor**.**feature\_extractor,)

有了这些，就可以开始训练了！

### **训练**

要启动训练，只需执行:

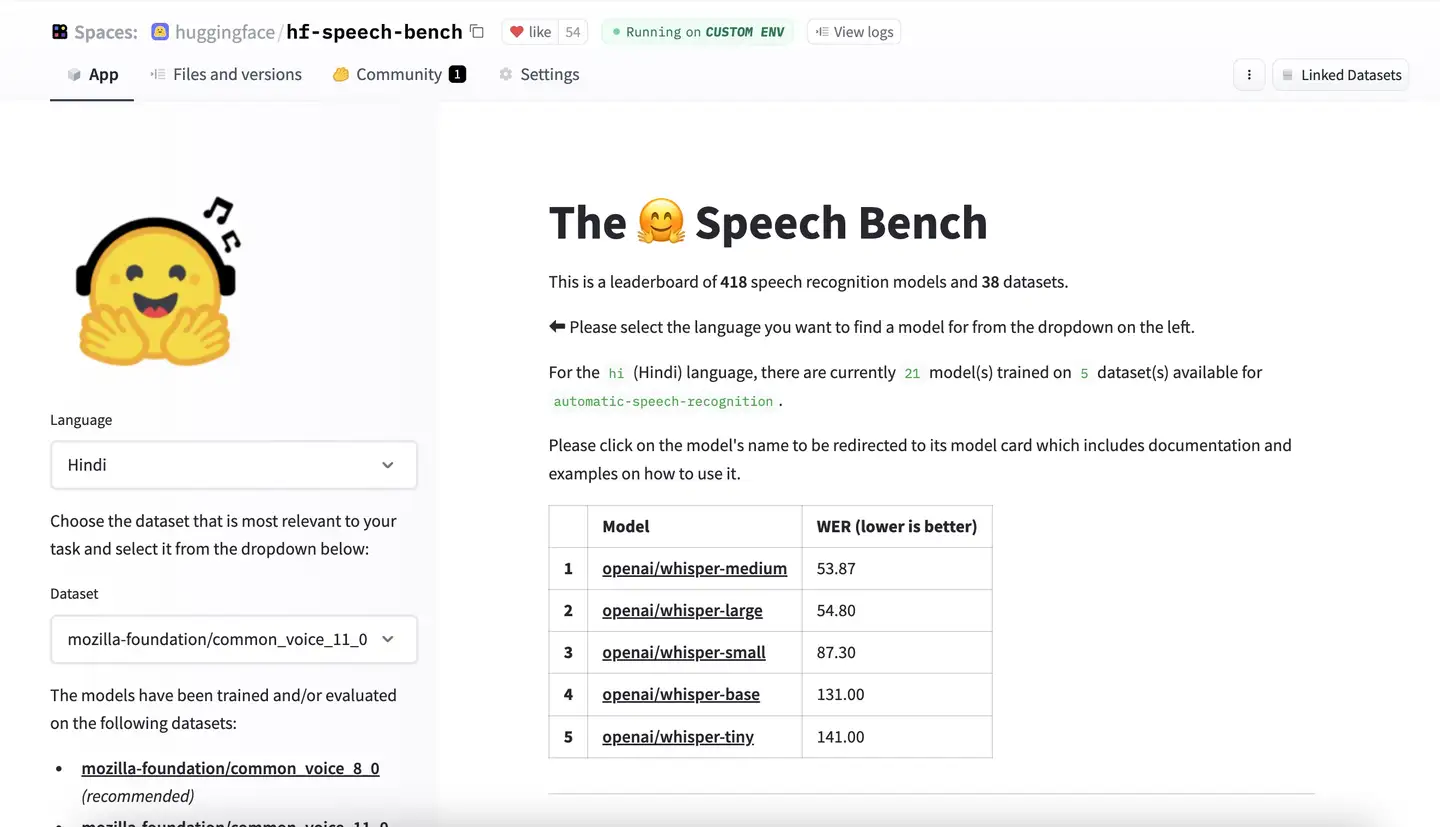
trainer**.**train()

训练大约需要 5-10 个小时，具体取决于你的 GPU 或 Google Colab 后端的 GPU。根据 GPU 的情况，你可能会在开始训练时遇到 CUDA 内存耗尽错误。此时，你可以将 per\_device\_train\_batch\_size 逐次减少 2 倍，同时增加 **[gradient\_accumulation\_steps](https://link.zhihu.com/?target=https://huggingface.co/docs/transformers/main_classes/trainer%23transformers.Seq2SeqTrainingArguments.gradient_accumulation_steps" \t "https://zhuanlan.zhihu.com/p/_blank)** 进行补偿。

**打印输出:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 步数 | 训练损失 | 轮数 | 验证损失 | WER |
| 1000 | 0.1011 | 2.44 | 0.3075 | 34.63 |
| 2000 | 0.0264 | 4.89 | 0.3558 | 33.13 |
| 3000 | 0.0025 | 7.33 | 0.4214 | 32.59 |
| 4000 | 0.0006 | 9.78 | 0.4519 | 32.01 |
| 5000 | 0.0002 | 12.22 | 0.4679 | 32.10 |

最佳 WER 是 32.0% —— 对 8 小时的训练数据来说还不错！那与其他 ASR 系统相比，这个表现到底处于什么水平？为此，我们可以查看 **[hf-speech-bench](https://link.zhihu.com/?target=https://huggingface.co/spaces/huggingface/hf-speech-bench" \t "https://zhuanlan.zhihu.com/p/_blank)**，这是一个按语种和数据集对模型分别进行 WER 排名的排行榜。



微调后的模型显著提高了 Whisper small checkpoint 的零样本性能，也突出展示了 Whisper 强大的迁移学习能力。

当将训练结果推送到 Hub 时，只需配置适当的关键字参数 (key-word arguments，kwargs) 就可以自动将 checkpoint 提交到排行榜。如需适配自己的数据集、语种和模型名称，仅需对下述代码作出相应的修改即可:

kwargs = {

"dataset\_tags": "mozilla-foundation/common\_voice\_11\_0",

"dataset": "Common Voice 11.0", # a 'pretty' name for the training dataset

"dataset\_args": "config: hi, split: test",

"language": "hi",

"model\_name": "Whisper Small Hi - Sanchit Gandhi", # a 'pretty' name for your model

"finetuned\_from": "openai/whisper-small",

"tasks": "automatic-speech-recognition",

"tags": "hf-asr-leaderboard",

}

现在，只需执行 push\_to\_hub 命令就可以将训练结果上传到 Hub 了:

trainer**.**push\_to\_hub(**\*\***kwargs)

任何人可以用你的模型的 Hub 链接访问它。他们还可以使用标识符 "your-username/the-name-you-picked"加载它，例如:

**from** transformers **import** WhisperForConditionalGeneration, WhisperProcessor

model **=** WhisperForConditionalGeneration**.**from\_pretrained("sanchit-gandhi/whisper-small-hi")processor **=** WhisperProcessor**.**from\_pretrained("sanchit-gandhi/whisper-small-hi")

虽然微调后的模型在 Common Voice Hindi 测试数据上的效果还不错，但其效果远算不上最优。本文的目的仅为演示如何在任意多语种 ASR 数据集上微调预训练的 Whisper checkpoint，对效果并未做太多深究。如需提升效果，你还可以尝试更多技巧，如优化训练超参 (例如 *learning rate* 和 *dropout*) 、使用更大的预训练 checkpoint ( medium 或 large) 等。

### **构建演示应用**

现在模型已经微调结束，我们开始构建一个演示应用来展示其 ASR 功能！我们将使用 Transformers pipeline 来完成整个 ASR 流水线: 从对音频输入进行预处理一直到对模型输出进行解码。我们使用 **[Gradio](https://link.zhihu.com/?target=https://www.gradio.app/" \t "https://zhuanlan.zhihu.com/p/_blank)** 来构建我们的交互式演示。 Gradio 提供了最直截了当的构建机器学习演示应用的方法，我们可以用它在几分钟内构建一个演示应用！

运行以下代码会生成一个 Gradio 演示应用，它用计算机的麦克风录制语音并将其馈送给微调后的 Whisper 模型以转录出相应的文本:

**from** transformers **import** pipeline**import** gradio **as** gr

pipe **=** pipeline(model**=**"sanchit-gandhi/whisper-small-hi") *# change to "your-username/the-name-you-picked"*

**def** **transcribe**(audio):

text **=** pipe(audio)["text"]

**return** text

iface **=** gr**.**Interface(

fn**=**transcribe,

inputs**=**gr**.**Audio(source**=**"microphone", type**=**"filepath"),

outputs**=**"text",

title**=**"Whisper Small Hindi",

description**=**"Realtime demo for Hindi speech recognition using a fine-tuned Whisper small model.",)

iface**.**launch()

## **结束语**

通过本文，我们介绍了如何使用 Datasets、Transformers 和 Hugging Face Hub 一步步为多语种 ASR 微调一个 Whisper 模型。如果你想自己尝试微调一个，请参阅 **[Google Colab](https://link.zhihu.com/?target=https://colab.research.google.com/github/sanchit-gandhi/notebooks/blob/main/fine_tune_whisper.ipynb" \t "https://zhuanlan.zhihu.com/p/_blank)**。如果你有兴趣针对英语和多语种 ASR 微调一个其它的 Transformers 模型，请务必参考下 **[examples/pytorch/speech-recognition](https://link.zhihu.com/?target=https://github.com/huggingface/transformers/tree/main/examples/pytorch/speech-recognition" \t "https://zhuanlan.zhihu.com/p/_blank)**。

import requests  
  
# 替换为你的 Hugging Face 访问令牌  
headers = {  
 "Authorization": "hf\_EUujIWzXvDjTXtNIHjQBlcXTIZoaNULYlV"  
}  
  
# 测试 token 是否有效  
response = requests.get("https://huggingface.co/api/whoami-v2", headers=headers)  
print(response.json())  
  
  
# 替换为文件的实际 URL  
file\_url = "https://huggingface.co/datasets/mozilla-foundation/common\_voice\_12\_0/resolve/main/audio/zh-CN"  
output\_path = r"D:\桌面\2024\_09sherpa\zh-CN.zip" # 确保路径和文件名正确  
  
# 发起请求并下载文件  
response = requests.get(file\_url, headers=headers, proxies={"http": "http://127.0.0.1:7890", "https": "http://127.0.0.1:7890"})  
  
# 检查请求是否成功  
if response.status\_code == 200:  
 with open(output\_path, "wb") as file:  
 file.write(response.content)  
 print(f"File downloaded to {output\_path}")  
else:  
 print(f"Failed to download file. Status code: {response.status\_code}")  
  
  
  
import requests  
  
proxies = {  
 "http": "http://127.0.0.1:7890",  
 "https": "http://127.0.0.1:7890",  
}  
  
response = requests.get("https://api.ipify.org", proxies=proxies)  
print(response.text)  
  
  
from datasets import load\_dataset  
  
# Login using e.g. `huggingface-cli login` to access this dataset  
ds = load\_dataset("mozilla-foundation/common\_voice\_12\_0", "zh-CN", timeout=60)  
  
datasets.save\_to\_disk(" D:\桌面\/2024\_09sherpa")

raise e1 from None

File "D:\Annaconda\envs\src\lib\site-packages\datasets\load.py", line 1854, in dataset\_module\_factory

raise ConnectionError(f"Couldn't reach '{path}' on the Hub ({e.\_\_class\_\_.\_\_name\_\_})") from e

ConnectionError: Couldn't reach 'mozilla-foundation/common\_voice\_12\_0' on the Hub (ProxyError)