Whisper Model of Huggingface

**Whisper**

Whisper is a pre-trained model for automatic speech recognition (ASR) and speech translation. Trained on 680k hours of labelled data, Whisper models demonstrate a strong ability to generalise to many datasets and domains **without** the need for fine-tuning.

Whisper was proposed in the paper [Robust Speech Recognition via Large-Scale Weak Supervision](https://arxiv.org/abs/2212.04356) by Alec Radford et al from OpenAI. The original code repository can be found [here](https://github.com/openai/whisper).

**Update:** following the release of the paper, the Whisper authors announced a large-v2 model trained for 2.5x more epochs with regularization. This large-v2 model surpasses the performance of the large model, with no architecture changes. Thus, it is recommended that the large-v2 model is used in-place of the original large model.

**Disclaimer**: Content for this model card has partly been written by the Hugging Face team, and parts of it were copied and pasted from the original model card.

**Model details**

Whisper is a Transformer based encoder-decoder model, also referred to as a *sequence-to-sequence* model. It was trained on 680k hours of labelled speech data annotated using large-scale weak supervision.

The models were trained on either English-only data or multilingual data. The English-only models were trained on the task of speech recognition. The multilingual models were trained on both speech recognition and speech translation. For speech recognition, the model predicts transcriptions in the *same* language as the audio. For speech translation, the model predicts transcriptions to a *different* language to the audio.

Whisper checkpoints come in five configurations of varying model sizes. The smallest four are trained on either English-only or multilingual data. The largest checkpoints are multilingual only. All ten of the pre-trained checkpoints are available on the [Hugging Face Hub](https://huggingface.co/models?search=openai/whisper). The checkpoints are summarised in the following table with links to the models on the Hub:

| **Size** | **Parameters** | **English-only** | **Multilingual** |
| --- | --- | --- | --- |
| tiny | 39 M | [✓](https://huggingface.co/openai/whisper-tiny.en) | [✓](https://huggingface.co/openai/whisper-tiny) |
| base | 74 M | [✓](https://huggingface.co/openai/whisper-base.en) | [✓](https://huggingface.co/openai/whisper-base) |
| small | 244 M | [✓](https://huggingface.co/openai/whisper-small.en) | [✓](https://huggingface.co/openai/whisper-small) |
| medium | 769 M | [✓](https://huggingface.co/openai/whisper-medium.en) | [✓](https://huggingface.co/openai/whisper-medium) |
| large | 1550 M | x | [✓](https://huggingface.co/openai/whisper-large) |
| large-v2 | 1550 M | x | [✓](https://huggingface.co/openai/whisper-large-v2) |

**Usage**

To transcribe audio samples, the model has to be used alongside a [WhisperProcessor](https://huggingface.co/docs/transformers/model_doc/whisper" \l "transformers.WhisperProcessor).

The WhisperProcessor is used to:

1. Pre-process the audio inputs (converting them to log-Mel spectrograms for the model)
2. Post-process the model outputs (converting them from tokens to text)

The model is informed of which task to perform (transcription or translation) by passing the appropriate "context tokens". These context tokens are a sequence of tokens that are given to the decoder at the start of the decoding process, and take the following order:

1. The transcription always starts with the <|startoftranscript|> token
2. The second token is the language token (e.g. <|en|> for English)
3. The third token is the "task token". It can take one of two values: <|transcribe|> for speech recognition or <|translate|> for speech translation
4. In addition, a <|notimestamps|> token is added if the model should not include timestamp prediction

Thus, a typical sequence of context tokens might look as follows:

<|startoftranscript|> <|en|> <|transcribe|> <|notimestamps|>

Which tells the model to decode in English, under the task of speech recognition, and not to predict timestamps.

These tokens can either be forced or un-forced. If they are forced, the model is made to predict each token at each position. This allows one to control the output language and task for the Whisper model. If they are un-forced, the Whisper model will automatically predict the output langauge and task itself.

The context tokens can be set accordingly:

model.config.forced\_decoder\_ids = WhisperProcessor.get\_decoder\_prompt\_ids(language="english", task="transcribe")

Which forces the model to predict in English under the task of speech recognition.

**Transcription**

**English to English**

In this example, the context tokens are 'unforced', meaning the model automatically predicts the output language (English) and task (transcribe).

>>> from transformers import WhisperProcessor, WhisperForConditionalGeneration

>>> from datasets import load\_dataset

>>> *# load model and processor*

>>> processor = WhisperProcessor.from\_pretrained("openai/whisper-large")

>>> model = WhisperForConditionalGeneration.from\_pretrained("openai/whisper-large")

>>> model.config.forced\_decoder\_ids = None

>>> *# load dummy dataset and read audio files*

>>> ds = load\_dataset("hf-internal-testing/librispeech\_asr\_dummy", "clean", split="validation")

>>> sample = ds[0]["audio"]

>>> input\_features = processor(sample["array"], sampling\_rate=sample["sampling\_rate"], return\_tensors="pt").input\_features

>>> *# generate token ids*

>>> predicted\_ids = model.generate(input\_features)

>>> *# decode token ids to text*

>>> transcription = processor.batch\_decode(predicted\_ids, skip\_special\_tokens=False)

['<|startoftranscript|><|en|><|transcribe|><|notimestamps|> Mr. Quilter is the apostle of the middle classes and we are glad to welcome his gospel.<|endoftext|>']

>>> transcription = processor.batch\_decode(predicted\_ids, skip\_special\_tokens=True)

[' Mr. Quilter is the apostle of the middle classes and we are glad to welcome his gospel.']

The context tokens can be removed from the start of the transcription by setting skip\_special\_tokens=True.

**French to French**

The following example demonstrates French to French transcription by setting the decoder ids appropriately.

>>> from transformers import WhisperProcessor, WhisperForConditionalGeneration

>>> from datasets import Audio, load\_dataset

>>> *# load model and processor*

>>> processor = WhisperProcessor.from\_pretrained("openai/whisper-large")

>>> model = WhisperForConditionalGeneration.from\_pretrained("openai/whisper-large")

>>> forced\_decoder\_ids = processor.get\_decoder\_prompt\_ids(language="french", task="transcribe")

>>> *# load streaming dataset and read first audio sample*

>>> ds = load\_dataset("common\_voice", "fr", split="test", streaming=True)

>>> ds = ds.cast\_column("audio", Audio(sampling\_rate=16\_000))

>>> input\_speech = next(iter(ds))["audio"]

>>> input\_features = processor(input\_speech["array"], sampling\_rate=input\_speech["sampling\_rate"], return\_tensors="pt").input\_features

>>> *# generate token ids*

>>> predicted\_ids = model.generate(input\_features, forced\_decoder\_ids=forced\_decoder\_ids)

>>> *# decode token ids to text*

>>> transcription = processor.batch\_decode(predicted\_ids)

['<|startoftranscript|><|fr|><|transcribe|><|notimestamps|> Un vrai travail intéressant va enfin être mené sur ce sujet.<|endoftext|>']

>>> transcription = processor.batch\_decode(predicted\_ids, skip\_special\_tokens=True)

[' Un vrai travail intéressant va enfin être mené sur ce sujet.']

**Translation**

Setting the task to "translate" forces the Whisper model to perform speech translation.

**French to English**

>>> from transformers import WhisperProcessor, WhisperForConditionalGeneration

>>> from datasets import Audio, load\_dataset

>>> *# load model and processor*

>>> processor = WhisperProcessor.from\_pretrained("openai/whisper-large")

>>> model = WhisperForConditionalGeneration.from\_pretrained("openai/whisper-large")

>>> forced\_decoder\_ids = processor.get\_decoder\_prompt\_ids(language="french", task="translate")

>>> *# load streaming dataset and read first audio sample*

>>> ds = load\_dataset("common\_voice", "fr", split="test", streaming=True)

>>> ds = ds.cast\_column("audio", Audio(sampling\_rate=16\_000))

>>> input\_speech = next(iter(ds))["audio"]

>>> input\_features = processor(input\_speech["array"], sampling\_rate=input\_speech["sampling\_rate"], return\_tensors="pt").input\_features

>>> *# generate token ids*

>>> predicted\_ids = model.generate(input\_features, forced\_decoder\_ids=forced\_decoder\_ids)

>>> *# decode token ids to text*

>>> transcription = processor.batch\_decode(predicted\_ids, skip\_special\_tokens=True)

[' A very interesting work, we will finally be given on this subject.']

**Evaluation**

This code snippet shows how to evaluate Whisper Large on [LibriSpeech test-clean](https://huggingface.co/datasets/librispeech_asr):

>>> from datasets import load\_dataset

>>> from transformers import WhisperForConditionalGeneration, WhisperProcessor

>>> import torch

>>> from evaluate import load

>>> librispeech\_test\_clean = load\_dataset("librispeech\_asr", "clean", split="test")

>>> processor = WhisperProcessor.from\_pretrained("openai/whisper-large")

>>> model = WhisperForConditionalGeneration.from\_pretrained("openai/whisper-large").to("cuda")

>>> def map\_to\_pred(batch):

>>> audio = batch["audio"]

>>> input\_features = processor(audio["array"], sampling\_rate=audio["sampling\_rate"], return\_tensors="pt").input\_features

>>> batch["reference"] = processor.tokenizer.\_normalize(batch['text'])

>>>

>>> with torch.no\_grad():

>>> predicted\_ids = model.generate(input\_features.to("cuda"))[0]

>>> transcription = processor.decode(predicted\_ids)

>>> batch["prediction"] = processor.tokenizer.\_normalize(transcription)

>>> return batch

>>> result = librispeech\_test\_clean.map(map\_to\_pred)

>>> wer = load("wer")

>>> print(100 \* wer.compute(references=result["reference"], predictions=result["prediction"]))

3.0003583080317572

**Long-Form Transcription**

The Whisper model is intrinsically designed to work on audio samples of up to 30s in duration. However, by using a chunking algorithm, it can be used to transcribe audio samples of up to arbitrary length. This is possible through Transformers [pipeline](https://huggingface.co/docs/transformers/main_classes/pipelines#transformers.AutomaticSpeechRecognitionPipeline) method. Chunking is enabled by setting chunk\_length\_s=30 when instantiating the pipeline. With chunking enabled, the pipeline can be run with batched inference. It can also be extended to predict sequence level timestamps by passing return\_timestamps=True:

>>> import torch

>>> from transformers import pipeline

>>> from datasets import load\_dataset

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

>>> pipe = pipeline(

>>> "automatic-speech-recognition",

>>> model="openai/whisper-large",

>>> chunk\_length\_s=30,

>>> device=device,

>>> )

>>> ds = load\_dataset("hf-internal-testing/librispeech\_asr\_dummy", "clean", split="validation")

>>> sample = ds[0]["audio"]

>>> prediction = pipe(sample.copy(), batch\_size=8)["text"]

" Mr. Quilter is the apostle of the middle classes, and we are glad to welcome his gospel."

>>> *# we can also return timestamps for the predictions*

>>> prediction = pipe(sample.copy(), batch\_size=8, return\_timestamps=True)["chunks"]

[{'text': ' Mr. Quilter is the apostle of the middle classes and we are glad to welcome his gospel.',

'timestamp': (0.0, 5.44)}]

Refer to the blog post [ASR Chunking](https://huggingface.co/blog/asr-chunking) for more details on the chunking algorithm.

**Fine-Tuning**

The pre-trained Whisper model demonstrates a strong ability to generalise to different datasets and domains. However, its predictive capabilities can be improved further for certain languages and tasks through *fine-tuning*. The blog post [Fine-Tune Whisper with 🤗 Transformers](https://huggingface.co/blog/fine-tune-whisper) provides a step-by-step guide to fine-tuning the Whisper model with as little as 5 hours of labelled data.

**Evaluated Use**

The primary intended users of these models are AI researchers studying robustness, generalization, capabilities, biases, and constraints of the current model. However, Whisper is also potentially quite useful as an ASR solution for developers, especially for English speech recognition. We recognize that once models are released, it is impossible to restrict access to only “intended” uses or to draw reasonable guidelines around what is or is not research.

The models are primarily trained and evaluated on ASR and speech translation to English tasks. They show strong ASR results in ~10 languages. They may exhibit additional capabilities, particularly if fine-tuned on certain tasks like voice activity detection, speaker classification, or speaker diarization but have not been robustly evaluated in these areas. We strongly recommend that users perform robust evaluations of the models in a particular context and domain before deploying them.

In particular, we caution against using Whisper models to transcribe recordings of individuals taken without their consent or purporting to use these models for any kind of subjective classification. We recommend against use in high-risk domains like decision-making contexts, where flaws in accuracy can lead to pronounced flaws in outcomes. The models are intended to transcribe and translate speech, use of the model for classification is not only not evaluated but also not appropriate, particularly to infer human attributes.

**Training Data**

The models are trained on 680,000 hours of audio and the corresponding transcripts collected from the internet. 65% of this data (or 438,000 hours) represents English-language audio and matched English transcripts, roughly 18% (or 126,000 hours) represents non-English audio and English transcripts, while the final 17% (or 117,000 hours) represents non-English audio and the corresponding transcript. This non-English data represents 98 different languages.

As discussed in [the accompanying paper](https://cdn.openai.com/papers/whisper.pdf), we see that performance on transcription in a given language is directly correlated with the amount of training data we employ in that language.

**Performance and Limitations**

Our studies show that, over many existing ASR systems, the models exhibit improved robustness to accents, background noise, technical language, as well as zero shot translation from multiple languages into English; and that accuracy on speech recognition and translation is near the state-of-the-art level.

However, because the models are trained in a weakly supervised manner using large-scale noisy data, the predictions may include texts that are not actually spoken in the audio input (i.e. hallucination). We hypothesize that this happens because, given their general knowledge of language, the models combine trying to predict the next word in audio with trying to transcribe the audio itself.

Our models perform unevenly across languages, and we observe lower accuracy on low-resource and/or low-discoverability languages or languages where we have less training data. The models also exhibit disparate performance on different accents and dialects of particular languages, which may include higher word error rate across speakers of different genders, races, ages, or other demographic criteria. Our full evaluation results are presented in [the paper accompanying this release](https://cdn.openai.com/papers/whisper.pdf).

In addition, the sequence-to-sequence architecture of the model makes it prone to generating repetitive texts, which can be mitigated to some degree by beam search and temperature scheduling but not perfectly. Further analysis on these limitations are provided in [the paper](https://cdn.openai.com/papers/whisper.pdf). It is likely that this behavior and hallucinations may be worse on lower-resource and/or lower-discoverability languages.

**Broader Implications**

We anticipate that Whisper models’ transcription capabilities may be used for improving accessibility tools. While Whisper models cannot be used for real-time transcription out of the box – their speed and size suggest that others may be able to build applications on top of them that allow for near-real-time speech recognition and translation. The real value of beneficial applications built on top of Whisper models suggests that the disparate performance of these models may have real economic implications.

There are also potential dual use concerns that come with releasing Whisper. While we hope the technology will be used primarily for beneficial purposes, making ASR technology more accessible could enable more actors to build capable surveillance technologies or scale up existing surveillance efforts, as the speed and accuracy allow for affordable automatic transcription and translation of large volumes of audio communication. Moreover, these models may have some capabilities to recognize specific individuals out of the box, which in turn presents safety concerns related both to dual use and disparate performance. In practice, we expect that the cost of transcription is not the limiting factor of scaling up surveillance projects.

**BibTeX entry and citation info**

@misc{radford2022whisper,

doi = {10.48550/ARXIV.2212.04356},

url = {https://arxiv.org/abs/2212.04356},

author = {Radford, Alec and Kim, Jong Wook and Xu, Tao and Brockman, Greg and McLeavey, Christine and Sutskever, Ilya},

title = {Robust Speech Recognition via Large-Scale Weak Supervision},

publisher = {arXiv},

year = {2022},

copyright = {arXiv.org perpetual, non-exclusive license}

}

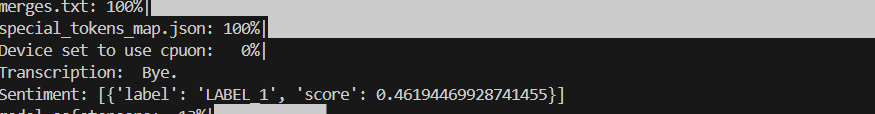
Installations

Pip install torch

Pip install torchaudio

pip install openai-whisper -> to convert audio to text

pip install ffmpeg-python -> for audio files



On gitignore

venv/

\_\_pycache\_\_/

\*.pyc

git rm -r --cached venv/

git lfs install

git lfs track "venv/Lib/site-packages/\*\*/\*.dll"

git lfs track "Lib/site-packages/\*\*/\*.dll"

git add .gitattributes

 git commit -m "Track large DLL files with Git LFS"

 git filter-branch --index-filter 'git rm --cached --ignore-unmatch <path/to/large/file>' --prune-empty -- --all

In my case;

git filter-branch --index-filter 'git rm --cached --ignore-unmatch venv/Lib/site-packages/torch/lib/dnnl.dll' --prune-empty -- --all

you can do it using the git filter-branch command (as mentioned earlier):

1. **Identify the large files:** The error messages clearly point to these files:
   * Lib/site-packages/llvmlite/binding/llvmlite.dll
   * Lib/site-packages/torch/lib/torch\_cpu.dll
   * venv/Lib/site-packages/torch/lib/dnnl.dll
2. **Run the git filter-branch command:** This command will go through your entire commit history and remove the specified files.

Bash

git filter-branch --index-filter 'git rm --cached --ignore-unmatch Lib/site-packages/llvmlite/binding/llvmlite.dll' --prune-empty -- --all

git filter-branch --index-filter 'git rm --cached --ignore-unmatch Lib/site-packages/torch/lib/torch\_cpu.dll' --prune-empty -- --all

git filter-branch --index-filter 'git rm --cached --ignore-unmatch venv/Lib/site-packages/torch/lib/dnnl.dll' --prune-empty -- --all

* + --index-filter: This option allows you to modify the index (staging area) for each commit.
  + 'git rm --cached --ignore-unmatch <path>': This command removes the specified file from the index but keeps it in your working directory (if it exists there in that commit). --ignore-unmatch prevents an error if the file doesn't exist in a particular commit.
  + --prune-empty: This removes any commits that become empty after filtering.
  + -- --all: This processes all branches and tags in your repository.

**Be patient, as this process can take some time, especially for repositories with a long history.**

1. **After git filter-branch completes, you will have rewritten your Git history.** Now you need to **force-push** your changes to the remote repository:

Bash

git push --force origin main

**Warning:** Force-pushing **overwrites the history** of the main branch on the remote repository. **If you are collaborating with others, they will need to take steps to reconcile their local repositories with the rewritten history (usually by fetching and then resetting or rebasing). Make sure everyone on the team is aware of this before you force-push.**

1. **Clean up Git's reflog and garbage collection:** After a history rewrite, it's good practice to clean up Git's reflog and perform garbage collection to reclaim space:

Bash

git reflog expire --expire=now --all

git gc --prune=now --aggressive

**Alternative (Potentially Simpler if the files were added in recent commits):**

If these large files were only introduced in the last few commits, you might be able to use git rebase -i (interactive rebase) to edit those commits and remove the files. This is generally safer than git filter-branch if the problematic commits are recent.

1. **Identify the last few commits:** Use git log --oneline to see the recent commit history.
2. **Initiate interactive rebase:** If, for example, the problematic commits are within the last 3 commits, run:

Bash

git rebase -i HEAD~3

1. **In the editor that opens, find the lines corresponding to the commits where the large files were added.** Change the word pick to edit for those commits.
2. **Save and close the editor.** Git will stop at the first commit you marked for editing.
3. **For each commit where you stopped:**
   * Run git rm --cached <path/to/large/file>.
   * Run git commit --amend --no-edit to replace the old commit with the modified one (without the large file).
   * Run git rebase --continue to move to the next commit you marked for editing.
4. **Once the rebase is finished, you'll need to force-push:**

Bash

git push --force origin main

**Which method to choose?**

* If the large files have been in the repository for a long time and across many commits, git filter-branch is more comprehensive.
* If the large files were only added in the last few commits, git rebase -i can be a bit cleaner and potentially safer.

**Given the persistent errors, it's highly likely that the files are in your Git history, so you will need to use one of these history-rewriting methods.**