Testing the Massively Multilingual Speech (MMS) Model that Supports 1162 Languages

Explore the cutting-edge multilingual features of Meta's latest automatic speech recognition (ASR) model

Luís Roque
Towards Data Science

Luís Roque
Published in

Towards Data Science
·
8 min read
·
May 26

Introduction

Massively Multilingual Speech (MMS)[1] is the latest release by Meta AI (just a few days ago). It pushes the boundaries of speech technology by expanding its reach from about 100 languages to over 1,000. This was achieved by building a single multilingual speech recognition model. The model can also identify over 4,000 languages, representing a 40-fold increase over previous capabilities.

The MMS project aims to make it easier for people to access information and use devices in their preferred language. It expands text-to-speech and speech-to-text technology to underserved languages, continuing to reduce language barriers in our global world. Existing applications can now include a wider variety of languages, such as virtual assistants or voice-activated devices. At the same time, new use cases emerge in cross-cultural communication, for example, in messaging services or virtual and augmented reality.

In this article, we will walk through the use of MMS for ASR in English and Portuguese and provide a step-by-step guide on setting up the environment to run the model.

Figure 1: Massively Multilingual Speech (MMS) is capable of identifying over 4,000 languages and supports 1162 (source)

This article belongs to "Large Language Models Chronicles: Navigating the NLP Frontier", a new weekly series of articles that will explore how to leverage the power of large models for various NLP tasks. By diving into these cutting-edge technologies, we aim to empower developers, researchers, and enthusiasts to harness the potential of NLP and unlock new possibilities.

Articles published so far:

- Summarizing the latest Spotify releases with ChatGPT
- Master Semantic Search at Scale: Index Millions of Documents with Lightning-Fast Inference Times using FAISS and Sentence Transformers
- Unlock the Power of Audio Data: Advanced Transcription and Diarization with Whisper, WhisperX, and PyAnnotate

Whisper JAX vs PyTorch: Uncovering the Truth about ASR Performance on GPUs

Vosk for Efficient Enterprise-Grade Speech Recognition: An Evaluation and Implementation Guide

As always, the code is available on my Github.

The Approach to build the Massively Multilingual Speech Model

Meta used religious texts, such as the Bible, to build a model covering this wide range of languages. These texts have several interesting components: first, they are translated into many languages, and second, there are publicly available audio recordings of people reading these texts in different languages. Thus, the main dataset where this model was trained was the New Testament, which the research team was able to collect for over 1,100 languages and provided more than 32h of data per language. They went further to make it recognize 4,000 languages. This was done by using unlabeled recordings of various other Christian religious readings. From the experiments results, even though the data is from a specific domain, it can generalize well.

These are not the only contributions of the work. They created a new preprocessing and alignment model that can handle long recordings. This was used to process the audio, and misaligned data was removed using a final cross-validation filtering step. Recall from one of our previous articles that we saw that one of the challenges of Whisper was the incapacity to align the transcription properly. Another important step of the approach was the usage of wav2vec 2.0, a self-supervised learning model, to train their system on a massive amount of speech data (about 500,000 hours) in over 1,400 languages. The labeled dataset we discussed previously is not enough to train a model of the size of MMS, so wav2vec 2.0 was used to reduce the need for labeled data. Finally, the resulting models were then fine-tuned for a specific speech task, such as multilingual speech recognition or language identification.

The MMS models were open-sourced by Meta a few days ago and were made available in the Fairseq repository. In the next section, we cover what Fairseq is and how we can test these new models from Meta.

Overview of the Fairseq Repository: A Powerful Toolkit for Sequence-to-Sequence Learning

Fairseq is an open-source sequence-to-sequence toolkit developed by Facebook AI Research, also known as FAIR. It provides reference implementations of various sequence modeling algorithms, including convolutional and recurrent neural networks, transformers, and other architectures.

The Fairseq repository is based on PyTorch, another open-source project initially developed by the Meta and now under the umbrella of the Linux Foundation. It is a very powerful machine learning framework that offers high flexibility and speed, particularly when it comes to deep learning.

The Fairseq implementations are designed for researchers and developers to train custom models and it supports tasks such as translation, summarization, language modeling, and other text generation tasks. One of

the key features of Fairseq is that it supports distributed training, meaning it can efficiently utilize multiple GPUs either on a single machine or across multiple machines. This makes it well-suited for large-scale machine learning tasks.

Massively Multilingual Speech Models

Fairseq provides two pre-trained models for download: MMS-300M and MMS-1B. You also have access to fine-tuned models available for different languages and datasets. For our purpose, we test the MMS-1B model fine-tuned for 102 languages in the FLEURS dataset and also the MMS-1B-all, which is fine-tuned to handle 1162 languages (!), fine-tuned using several different datasets.

Implementing Automatic Speech Recognition with Massively Multilingual Speech

Remember that these models are still in research phase, making testing a bit more challenging. There are additional steps that you would not find with production-ready software.

First, you need to set up a .env file in your project root to configure your environment variables. It should look something like this:

```
CURRENT_DIR=/path/to/current/dir
AUDIO_SAMPLES_DIR=/path/to/audio_samples
FAIRSEQ_DIR=/path/to/fairseq
VIDEO_FILE=/path/to/video/file
AUDIO_FILE=/path/to/audio/file
RESAMPLED_AUDIO_FILE=/path/to/resampled/audio/file
TMPDIR=/path/to/tmp
PYTHONPATH=.
PREFIX=INFER
HYDRA_FULL_ERROR=1
USER=micro
MODEL=/path/to/fairseq/models_new/mms1b_all.pt
LANG=eng
```

Next, you need to configure the YAML file located at fairseq/examples/mms/asr/config/infer_common.yaml. This file contains important settings and parameters used by the script.

In the YAML file, use a full path for the checkpoint field like this (unless you are using a containerized application to run the script):

```
checkpoint:
/path/to/checkpoint/${env:USER}/${env:PREFIX}/${common_eval.results_path}
```

This full path is necessary to avoid potential permission issues unless you are running the application in a container.

If you plan on using a CPU for computation instead of a GPU, you will need to add the following directive to the top level of the YAML file:

```
common:
  cpu: true
```

This setting directs the script to use the CPU for computations.

We use the dotevn python library to load these environment variables in our Python script. Since we are overwriting some system variables, we will need to use a trick to make sure that we get the right variables loaded. We use thedotevn_valuesmethod and store the output in a variable. This ensures that we get the variables stored in our .envfile and not random system variables even if they have the same name.

```
config = dotenv_values(".env")

current_dir = config['CURRENT_DIR']
tmp_dir = config['TMPDIR']
fairseq_dir = config['FAIRSEQ_DIR']
video_file = config['VIDEO_FILE']
audio_file = config['AUDIO_FILE']
audio_file_resampled = config['RESAMPLED_AUDIO_FILE']
model_path = config['MODEL']
model_new_dir = config['MODELS_NEW']
lang = config['LANG']
```

Then, we can clone the fairseq GitHub repository and install it in our machine.

```
def git_clone(url, path):
    """
    Clones a git repository

    Parameters:
    url (str): The URL of the git repository
    path (str): The local path where the git repository will be cloned
    """
    if not os.path.exists(path):
        Repo.clone_from(url, path)

def install_requirements(requirements):
    """
    Installs pip packages

    Parameters:
    requirements (list): List of packages to install
    """
    subprocess.check_call(["pip", "install"] + requirements)


git_clone('https://github.com/facebookresearch/fairseq', 'fairseq')
install_requirements(['--editable', './'])
```

We already discussed the models that we use in this article, so let’s download them to our local environment.

```
def download_file(url, path):
    """
```

```
    Downloads a file

    Parameters:
    url (str): URL of the file to be downloaded
    path (str): The path where the file will be saved
    """
    subprocess.check_call(["wget", "-P", path, url])

download_file('https://dl.fbaipublicfiles.com/mms/asr/mms1b_fl102.pt',
model_new_dir)
```

There is one additional restriction related to the input of the MMS model, the sampling rate of the audio data needs to be 16000 Hz. In our case, we defined two ways to generate these files: one that converts video to audio and another that resamples audio files for the correct sampling rate.

```
def convert_video_to_audio(video_path, audio_path):
    """
    Converts a video file to an audio file

    Parameters:
    video_path (str): Path to the video file
    audio_path (str): Path to the output audio file
    """
    subprocess.check_call(["ffmpeg", "-i", video_path, "-ar", "16000",
audio_path])

def resample_audio(audio_path, new_audio_path, new_sample_rate):
    """
    Resamples an audio file

    Parameters:
    audio_path (str): Path to the current audio file
    new_audio_path (str): Path to the output audio file
    new_sample_rate (int): New sample rate in Hz
    """
    audio = AudioSegment.from_file(audio_path)
    audio = audio.set_frame_rate(new_sample_rate)
    audio.export(new_audio_path, format='wav')
```

We are now ready to run the inference process using our MMS-1B-all model, which supports 1162 languages.

```
def run_inference(model, lang, audio):
    """
    Runs the MMS ASR inference

    Parameters:
    model (str): Path to the model file
    lang (str): Language of the audio file
    audio (str): Path to the audio file
    """
    subprocess.check_call(
        [
```

```
            "python",
            "examples/mms/asr/infer/mms_infer.py",
            "--model",
            model,
            "--lang",
            lang,
            "--audio",
            audio,
        ]
    )

run_inference(model_path, lang, audio_file_resampled)
```

Automatic Speech Recognition Results with Fairseq

In this section, we describe our experimentation setup and discuss the results. We performed ASR using two different models from Fairseq, MMS-1B-all and MMS-1B-FL102, in both English and Portuguese. You can find the audio files in my GitHub repo. These are files that I generated myself just for testing purposes.

Let’s start with the MMS-1B-all model. Here is the input and output for the English and Portuguese audio samples:

Eng: just requiring a small clip to understand if the new facebook research model really performs on

Por: ora bem só agravar aqui um exemplo pa tentar perceber se de facto om novo modelo da facebook research realmente funciona ou não vamos estar

With the MMS-1B-FL102, the generated speech was significantly worse. Let’s see the same example for English:

Eng: just recarding a small ho clip to understand if the new facebuok research model really performs on speed recognition tasks lets see

While the speech generated is not super impressive for the standard of models we have today, we need to address these results from the perspective that these models open up ASR to a much wider range of the global population.

Conclusion

The Massively Multilingual Speech model, developed by Meta, represents one more step to foster global communication and broaden the reach of language technology using AI. Its ability to understand over 4,000 languages and function effectively across 1,162 of them increases accessibility for numerous languages that have been traditionally underserved.

Our testing of the MMS models showcased the possibilities and limitations of the technology at its current stage. Although the speech generated by the MMS-1B-FL102 model was not as impressive as expected, the MMS-1B-all model provided promising results, demonstrating its capacity to transcribe speech in both English and Portuguese. Portuguese has been one of those

underserved languages, specially when we consider Portuguese from Portugal.

Feel free to try it out in your preferred language and to share the transcription and feedback in the comment section.

Keep in touch: LinkedIn

References

[1] — Pratap, V., Tjandra, A., Shi, B., Tomasello, P., Babu, A., Kundu, S., Elkahky, A., Ni, Z., Vyas, A., Fazel-Zarandi, M., Baevski, A., Adi, Y., Zhang, X., Hsu, W.-N., Conneau, A., & Auli, M. (2023). Scaling Speech Technology to 1,000+ Languages. arXiv.