

Robust Speech Challenge 🤖

Welcome to the robust speech recognition challenge 🗣️ !

The goal of this event is to build **robust, real-world** speech recognition (ASR) systems in as many languages as possible 🌍🌍🌍. If necessary and available, free access to a V100S 32 GB GPU will kindly be provided by the [OVHcloud team](#) 🚀. This document summarizes all the relevant information required for the speech community event 📄.

To sign-up, please see [this forum post](#) 😊. Please make sure to:

- Read it in detail
- Fill the google form
- Join our Discord server in the #join-sprint channel.

Table of Contents

- [TLDR:](#)
- [Important dates](#)
- [How to install pytorch, transformers, datasets](#)
- [Data and Preprocessing](#)
- [How to fine-tune an acoustic model](#)
- [How to fine-tune with OVH could](#)
- [How to combine n-gram language models with acoustic model](#)
- [Evaluation](#)
- [Prizes](#)
- [Communication and Problems](#)
- [Talks](#)
- [General Tips & Tricks](#)

TLDR

Participants are encouraged to leverage pre-trained speech recognition checkpoints, preferably [facebook/wav2vec2-large-xlsr-53](#), to train a speech recognition system in a language of their choice.

Speech recognition systems should be trained using **PyTorch**, 🤖 **Transformers**, and, 🤖 **Datasets**. For more information on how to install the above libraries, please read through [How to install pytorch, transformers, datasets](#).

Participants can make use of whatever data they think is useful to build a speech recognition system for **real-world** audio data - **except** the Common Voice "test" split of their chosen language. The section [Data and preprocessing](#) explains in more detail what audio data can be used, how to find suitable audio data, and how the audio data can be processed.

For training, it is recommended to use the [official training script](#) or a modification thereof. A step-by-step guide on how to fine-tune an acoustic model for a speech recognition system can be found under [How to fine-tune an acoustic model](#). If possible it is encouraged to fine-tune the acoustic models on local GPU machines, but if those are not available, the OVH could team kindly provides a limited number of GPUs for the event. Simply fill out [this google form](#) to get access to a GPU. For more information on how to train an acoustic model on one of OVH's GPU - see [How to fine-tune a speech recognition model with OVHcloud](#).

The performance of speech recognition system can often significantly be improved by adding a language model for decoding. For more information on how to add a language model, please take a look at [How to combine n-gram language models with speech recognition models](#).

During the event, the speech recognition system will be evaluated on both the Common Voice "test" split of the participants' chosen language as well as the *real-world* "dev" data provided by the Hugging Face team. At the end of the robust speech recognition challenge, the speech recognition system will also be evaluated on the *real-world* "test" data provided by the Hugging Face team. Each participant should add an `eval.py` script to her/his model repository in a specific format that lets one easily evaluate the speech recognition system on both Common Voice's "test" data as well as the *real-world* audio data. Please read through the [Evaluation](#) section to make sure your evaluation script is in the correct format. Speech recognition systems with evaluation scripts in an incorrect format can sadly not be considered for the Challenge.

At the end of the event, the best performing speech recognition system will receive a prize 🏆 - more information regarding the prizes can be found under [Prizes](#).

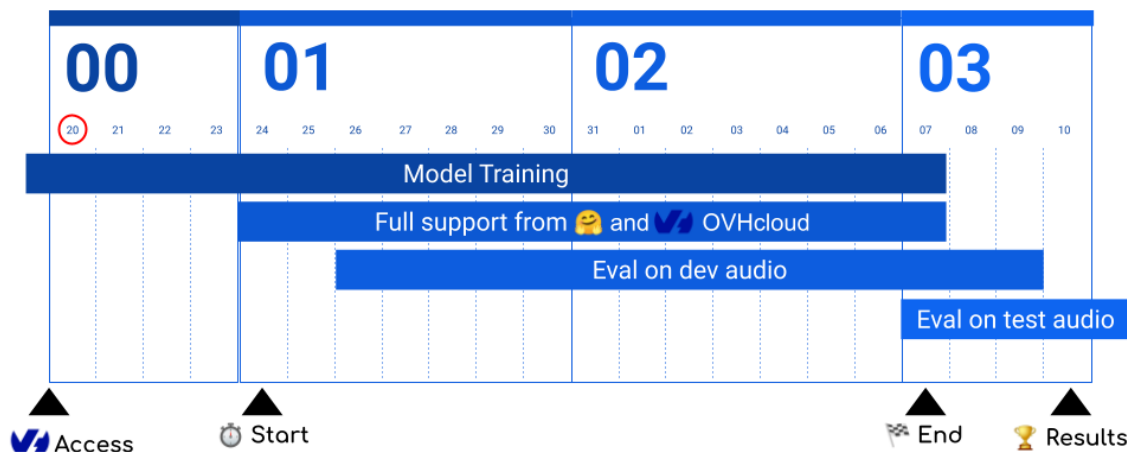
We believe that framing the event as a competition is more fun, but at the core, the event is about creating speech recognition systems in as many languages as possible as a community. This can be achieved by working together, helping each other to solve bugs, share important findings, etc... 😊

Note: Please, read through the section on [Communication & Problems](#) to make sure you know how to ask for help, etc... All important announcements will be made on discord. Please make sure that you've joined [this discord channel](#)

Also, please make sure that you have been added to the [Speech Event Organization](#). You should have received an invite by email. If you didn't receive an invite, please contact the organizers, *e.g.* Anton, Patrick, or Omar directly on discord.

Important dates

Timeline



Data and preprocessing

In this section, we will quickly go over how to find suitable training data and how to preprocess it.

To begin with, **all data except Common Voice's "test" data can be used as training data.** The exception includes all Common Voice versions as the test data split of later Common Voice versions often overlaps with the one

of previous versions, *e.g.* the test data of Common Voice 7 in English is to a big part identical to the test data of Common Voice 6 in English:

```
load_dataset("mozilla-foundation/common_voice_7_0", "en", split="test")
```

includes more or less the same data as

```
load_dataset("mozilla-foundation/common_voice_6_1", "en", split="test")
```

However, we strongly encourage participants to make use of Common Voice's other splits, *e.g.* "train" and "validation". For most languages, the Common Voice dataset offers already a decent amount of training data. It is usually always advantageous to collect additional data. To do so, the participants are in a first step encouraged to search the Hugging Face Hub for additional audio data, for example by selecting the category "[speech-processing](#)". All datasets that are available on the Hub can be downloaded via the 🤗 Datasets library in the same way Common Voice is downloaded. If one wants to combine multiple datasets for training, it might make sense to take a look at the [interleave_datasets](#) function.

In addition, participants can also make use of their audio data. Here, please make sure that you **are allowed to use the audio data**. *E.g.*, if audio data is taken from media platforms, such as YouTube, it should be verified that the media platform and the owner of the data have given her/his approval to use the audio data in the context of machine learning research. If you are not sure whether the data you want to use has the appropriate licensing, please contact the Hugging Face team on discord.

Next, let's talk about preprocessing. Audio data and transcriptions have to be brought into the correct format when training the acoustic model (example shown in [How to fine-tune an acoustic model](#)). It is recommended that this is done by using 🤗 Datasets `.map()` function as shown [here](#). As can be see we can pass some characters that will be removed from the transcriptions, *e.g.*: `--chars_to_ignore , ? . ! - \; \: \\" ` % ' " 🔪 \` on the official "[Single GPU Example](#)". The participants are free to modify this preprocessing by removing more characters or even replacing characters as it is done in the [official blog post](#). **However**, there are some rules regarding what characters are allowed to be removed/replaced and which are not. These rules are not this straightforward and therefore often have to be evaluated case-by-case. It is allowed (and recommended) to normalize the data to only have lower-case characters. It is also allowed (and recommended) to remove typographical symbols and punctuation marks. A list of such symbols can *e.g.* be found [here](#) - however here we already must be careful. We should **not** remove a symbol that would change the meaning of the words, *e.g.* in English, we should not remove the single quotation mark `'` since it would change the meaning of the word `"it's"` to `"its"` which would then be incorrect. So the golden rule here is to not remove any characters that could change the meaning of a word into another word. This is not always obvious and should be given some consideration. As another example, it is fine to remove the "Hyphen-minus" sign `-` since it doesn't change the meaning of a word to another one. *E.g.* `"fine-tuning"` would be changed to `"finetuning"` which has still the same meaning.

Since those choices are not always obvious when in doubt feel free to ask on Discord or even better post your question on the forum, as was done, *e.g.* [here](#).

How to install relevant libraries

The following libraries are required to fine-tune a speech model with 🤗 Transformers and 🤗 Datasets in PyTorch.

- [PyTorch](#)
- [Transformers](#)
- [Datasets](#)

We recommend installing the above libraries in a [virtual environment](#). If you're unfamiliar with Python virtual environments, check out the [user guide](#). Create a virtual environment with the version of Python you're going to use and activate it.

You should be able to run the command:

```
python3 -m venv <your-venv-name>
```

You can activate your venv by running

```
source ~/.<your-venv-name>/bin/activate
```

To begin with please make sure you have PyTorch and CUDA correctly installed. The following command should return `True` :

```
python -c "import torch; print(torch.cuda.is_available())"
```

If the above command doesn't print `True` , in the first step, please follow the instructions [here](#) to install PyTorch with CUDA.

We strongly recommend making use of the provided PyTorch examples scripts in [transformers/examples/pytorch/speech-recognition](#) to train your speech recognition system. In all likelihood, you will adjust one of the example scripts, so we recommend forking and cloning the 🤗 Transformers repository as follows.

1. Fork the [repository](#) by clicking on the 'Fork' button on the repository's page. This creates a copy of the code under your GitHub user account.
2. Clone your fork to your local disk, and add the base repository as a remote:

```
$ git clone https://github.com/<your Github handle>/transformers.git
$ cd transformers
$ git remote add upstream https://github.com/huggingface/transformers.git
```

3. Create a new branch to hold your development changes. This is especially useful to share code changes with your team:

```
$ git checkout -b a-descriptive-name-for-my-project
```

4. Set up a PyTorch environment by running the following command your virtual environment:

```
$ pip install -e ".[torch-speech]"
```

(If transformers was already installed in the virtual environment, remove it with `pip uninstall transformers` before reinstalling it in editable mode with the `-e` flag.)

If you have already cloned that repo, you might need to `git pull` to get the most recent changes in the `transformers` library.

Running this command will automatically install `torch` and the most relevant libraries required for fine-tuning a speech recognition system.

Next, you should also install the 🤗 Datasets library. We strongly recommend installing the library from source to profit from the most current additions during the community week.

Simply run the following steps:

```
$ cd ~/
$ git clone https://github.com/huggingface/datasets.git
$ cd datasets
$ pip install -e ".[streaming]"
```

If you plan on contributing a specific dataset during the community week, please fork the datasets repository and follow the instructions [here](#).

To verify that all libraries are correctly installed, you can run the following command in a Python shell. It verifies that both `transformers` and `datasets` have been correctly installed.

```
from transformers import AutoModelForCTC, AutoProcessor
from datasets import load_dataset

dummy_dataset = load_dataset("common_voice", "ab", split="test")

model = AutoModelForCTC.from_pretrained("hf-internal-testing/tiny-random-wav2vec2")
model.to("cuda")

processor = AutoProcessor.from_pretrained("hf-internal-testing/tiny-random-wav2vec2")

input_values = processor(dummy_dataset[0]["audio"]["array"], return_tensors="pt",
sampling_rate=16_000).input_values
input_values = input_values.to("cuda")

logits = model(input_values).logits

assert logits.shape[-1] == 32
```

How to finetune an acoustic model

In this section, we show you how to fine-tune a pre-trained [XLS-R Model](#) on the [Common Voice 7 dataset](#).

We recommend fine-tuning one of the following pre-trained XLS-R checkpoints:

- [300M parameters version](#)
- [1B parameters version](#)
- [2B parameters version](#)

To begin with, please note that to use the Common Voice dataset, you have to accept that **your email address** and **username** are shared with the mozilla-foundation. To get access to the dataset please click on "Access repository" [here](#).

Next, we recommended that you get familiar with the XLS-R model and its capabilities. In collaboration with [Fairseq's Wav2Vec2 team](#), we've written "[Fine-tuning XLS-R for Multi-Lingual ASR with 🤖 Transformers](#)" which gives an in-detail explanation of how XLS-R functions and how it can be fine-tuned.

The blog can also be opened and directly fine-tuned in a google colab notebook. In this section, we will explain how to fine-tune the model on a local machine.

1. Log in

To begin with, you should check that you are correctly logged in and that you have `git-lfs` installed so that your fine-tuned model can automatically be uploaded.

Run:

```
huggingface-cli login
```

to login. It is recommended to login with your access token that can be found under your hugging face profile (icon in the top right corner on [hf.co](#), then Settings -> Access Tokens -> User Access Tokens -> New Token (if haven't generated one already)

You can then copy-paste this token to log in locally.

2. Create your model repository

First, let's make sure that `git-lfs` is correctly installed. To so, simply run:

```
git-lfs -v
```

The output should show something like `git-lfs/2.13.2 (GitHub; linux amd64; go 1.15.4)`. If your console states that the `git-lfs` command was not found, please make sure to install it [here](#) or simply via:

```
sudo apt-get install git-lfs
```

Now you can create your model repository which will contain all relevant files to reproduce your training. You can either directly create the model repository on the Hub (Settings -> New Model) or via the CLI. Here we choose to use the CLI instead.

Assuming that we want to call our model repository *xls-r-ab-test*, we can run the following command:

```
huggingface-cli repo create xls-r-ab-test
```

You can now see the model on the Hub, e.g. under <https://huggingface.co/hf-test/xls-r-ab-test>.

Let's clone the repository so that we can define our training script inside.

```
git lfs install
git clone https://huggingface.co/hf-test/xls-r-ab-test
```

3. Add your training script and `run` -command to the repository

We encourage participants to add all relevant files for training directly to the directory so that everything is fully reproducible.

Let's first copy-paste the official training script from our clone of `transformers` to our just created directory:

```
cp ~/transformers/examples/pytorch/speech-recognition/run_speech_recognition_ctc.py
./
```

Next, we'll create a bash file to define the hyper-parameters and configurations for training. More detailed information on different settings (single-GPU vs. multi-GPU) can be found [here](#).

For demonstration purposes, we will use a dummy XLS-R model `model_name_or_path="hf-test/xls-r-dummy"` on the very low-resource language of "Abkhaz" of [Common Voice 7](#): `dataset_config_name="ab"` for just a single epoch.

Before starting to train, let's make sure we have installed all the required libraries. You might want to run:

```
pip install -r ~/transformers/examples/pytorch/speech-recognition/requirements.txt
```

Alright, finally we can define the training script. We'll simply use some dummy hyper-parameters and configurations for demonstration purposes.

Note that we add the flag `--use_auth_token` so that datasets requiring access, such as [Common Voice 7](#) can be downloaded. In addition, we add the `--push_to_hub` flag to make use of the [Trainers push_to_hub functionality](#), so that your model will be automatically uploaded to the Hub.

Let's copy the following code snippet in a file called `run.sh`

```
echo '''python run_speech_recognition_ctc.py \
  --dataset_name="mozilla-foundation/common_voice_7_0" \
  --model_name_or_path="hf-test/xls-r-dummy" \
  --dataset_config_name="ab" \
  --output_dir="." \
  --overwrite_output_dir \
  --max_steps="10" \
  --per_device_train_batch_size="2" \
  --learning_rate="3e-4" \
  --save_total_limit="1" \
  --evaluation_strategy="steps" \
  --text_column_name="sentence" \
  --length_column_name="input_length" \
  --save_steps="5" \
  --layerdrop="0.0" \
  --freeze_feature_encoder \
  --gradient_checkpointing \
  --fp16 \
  --group_by_length \
  --push_to_hub \
  --use_auth_token \
  --do_train --do_eval''' > run.sh
```

4. Start training

Now all that is left to do is to start training the model by executing the run file.

```
bash run.sh
```

The training should not take more than a couple of minutes. During the training intermediate saved checkpoints are automatically uploaded to your model repository as can be seen [on this commit](#).

At the end of the training, the [Trainer](#) automatically creates a nice model card and all relevant files are uploaded.

5. Tips for real model training

The above steps illustrate how a model can technically be fine-tuned. However as you can see on the model card [hf-test/xls-r-ab-test](#), our demonstration has a very poor performance which is not surprising given that we trained for just 10 steps on a randomly initialized model.

For real model training, it is recommended to use one of the actual pre-trained XLS-R models:

- [300M parameters version](#)
- [1B parameters version](#)
- [2B parameters version](#)

Also, the hyper-parameters should be carefully chosen depending on the dataset. As an example, we will fine-tune the 300M parameters model on Swedish on a single TITAN RTX 24GB GPU.

The model will be called `"xls-r-300m-sv"`. Following the above steps we first create the model:

```
huggingface-cli repo create xls-r-300m-sv
```

, clone it locally (assuming the `<username>` is `hf-test`)

```
git clone hf-test/xls-r-300m-sv
```

, and, define the following hyperparameters for training

```
echo '''python run_speech_recognition_ctc.py \
--dataset_name="mozilla-foundation/common_voice_7_0" \
--model_name_or_path="facebook/wav2vec2-xls-r-300m" \
--dataset_config_name="sv-SE" \
--output_dir="." \
--overwrite_output_dir \
--num_train_epochs="50" \
--per_device_train_batch_size="8" \
--per_device_eval_batch_size="8" \
--gradient_accumulation_steps="4" \
--learning_rate="7.5e-5" \
--warmup_steps="2000" \
--length_column_name="input_length" \
--evaluation_strategy="steps" \
--text_column_name="sentence" \
--chars_to_ignore , ? . ! \- \; \: \" \" % \" \" - ' ... - \
--save_steps="500" \
--eval_steps="500" \
--logging_steps="100" \
--layerdrop="0.0" \
```



```

--activation_dropout="0.1" \
--save_total_limit="3" \
--freeze_feature_encoder \
--feat_proj_dropout="0.0" \
--mask_time_prob="0.75" \
--mask_time_length="10" \
--mask_feature_prob="0.25" \
--mask_feature_length="64" \
--gradient_checkpointing \
--use_auth_token \
--fp16 \
--group_by_length \
--do_train --do_eval \
--push_to_hub''' > run.sh

```

The training takes *ca.* 7 hours and yields a reasonable test word error rate of 27% as can be seen on the automatically generated [model card](#).

The above-chosen hyperparameters probably work quite well on a range of different datasets and languages but are by no means optimal. It is up to you to find a good set of hyperparameters.

How to finetune with OVH cloud



For a more detailed guide on setting up OVHcloud please watch this video: <https://youtu.be/XkMnYocAEO0>

Creating an OVHCloud account

TIP: If you haven't created a project on OVHcloud yet, make sure you've received your GPU voucher code *beforehand*, so that you can skip entering the credit card information.

1. If you're a US citizen, create an account via [OVHcloud.CA](#). If you're from anywhere else in the world, create an account via [OVHcloud.COM](#).
2. Once logged in, click **Public Cloud** from the top menu and then click **Create your first OVH Public Cloud project**. Then enter a project name (e.g. "huggingface"), enter your voucher code, and click **Continue** -> **Create my project**. *Note: if you see a request for credit card details during the last step, and you can't skip it, then your voucher code is invalid. Please report it to the [#ovh-support](#) channel on Discord.*

Setting up an AI notebook

1. Go to the **Public Cloud** page and select **Project Management** -> **Users & Roles** from the menu on the left.
2. Click **+ Add user**. Write a user description (e.g. **AI Trainer**), and select an **AI Training Operator** user role. Click **Confirm**.
3. Write down the *username* and *password* (at the top of the screen) somewhere. They will be needed during step 7.
4. Select **AI & Machine Learning** -> **AI Training** from the menu on the left. Click **+ Launch a new job** on the AI Training page.
5. On the **Launch a new job** page:
 - In **1. Choose a region** select a region closest to you.

- In 2. Enter the Docker image select Custom image -> `baaastijn/ovh_huggingface`.
 - You can skip steps 3. and 4. if you will be using the Hugging Face Hub to store the models after training.
 - In 5. Configure your job select 1 GPU.
 - Validate the info and Create the job.
6. On the AI Training Jobs screen wait until the job's status changes from Pending to Running.
 7. Click HTTP Access from the Job's details page and log in with the AI training user you've created earlier. Once logged in, you can close the page and click HTTP Access to launch a JupyterLab notebook.
 8. Awesome, now you have a free GPU-enabled Jupyter instance!

Note: If you're an experienced Docker user, feel free to create a custom docker image with all of the needed packages like the one in step 5. The Dockerfile for it is available here: [baaastijn/Dockerimages](https://github.com/baaastijn/Dockerimages). Once you've built your image, push it to <https://hub.docker.com/> and select it during the OVHcloud job creation.

For more quick tutorials about OVHcloud AI products, check out the showcase <https://vimeo.com/showcase/8903300>

How to combine n-gram with acoustic model

Having trained a speech recognition model with CTC as shown in the section above, one can further improve the model's performance by adding an **n-gram language model** to the decoding process of the model. By doing so, we are replacing the naive greedy decoding with **n-gram-boosted** beam search decoding.

N-gram language models can be built on CPU in just a few minutes. *N-gram-boosted* beam search decoding noticeably slows down the inference time, but also yields significant word error rates improvements - usually between 10-40 %.

You can find an in-detail blog post on how to build an *n-gram* [here](#). The blog post can be opened in a google colab and by adapting three lines of the example for your use case, one can directly create an *n-gram* in the google colab. The blog post gives in-detail instructions on how to build an n-gram and how to add it to your trained speech recognition model.

- why one should add an *n-gram* to her/his speech recognition system,
- how to build an *n-gram*, and,
- how to add the built *n-gram* the speech recognition system for seamless decoding

Our previously trained model - [xls-r-300m-sv](#) - enjoys a 30% word error rate reduction after having added an n-gram. As shown in the example of the blog post, we strongly advise participants to upload all files required for combining the *n-gram* with a trained speech recognition model directly into the same model repository.

Evaluation

Finally, we have arrived at the most fun part of the challenge - sitting back and watching the model transcribe audio. If possible, every participant should evaluate the speech recognition system on the test set of Common Voice 7 and ideally also on the real-world audio data (if available). For languages that have neither a Common Voice evaluation dataset nor a real world evaluation dataset, please contact the organizers on Discord so that we can work together to find some evaluation data.

As a first step, one should copy the official `eval.py` script to her/his model repository. Let's use our previously trained [xls-r-300m-sv](#) again as an example.

Assuming that we have a clone of the model's repo under `~/xls-r-300m-sv`, we can copy the `eval.py` script to the repo.

```
cp ~/transformers/examples/research_projects/robust-speech-event/eval.py ~/xls-r-300m-sv
```

Next, we should adapt `eval.py` so that it fits our evaluation data. Here it is important to keep the `eval.py` file in the following format:

- 1. The following input arguments should not be changed and keep their original functionality/meaning (being to load the model and dataset): `--model_id` , `--dataset` , `--config` , `--split` . We recommend to not change any of the code written under `if __name__ == "__main__":` .
- 2. The function `def log_results(result: Dataset, args: Dict[str, str])` should also not be changed. The function expects the above names attached to the `args` object as well as a `datasets.Dataset` object, called `result` which includes all predictions and target transcriptions under the names `"predictions"` and `"targets"` respectively.
- 3. All other code can be changed and adapted. Participants are especially invited to change the `def normalize_text(text: str) -> str:` function as this might be a very language and model-training specific function.
- 4. **Important:** It is not allowed to "cheat" in any way when it comes to pre-and postprocessing. In short, "cheating" refers to any of the following:
 - a. Somehow giving the model access to the target transcriptions to improve performance. The model is not allowed to use the target transcriptions to generate its predictions.
 - b. Pre-processing the target transcriptions in a way that makes the target transcriptions lose their original meaning. This corresponds to what has already been said in [Data and Preprocessing](#) and is somewhat of a grey zone. It means that one should not remove characters that would make a word to lose its meaning. E.g., it is not allowed to replace all `e` in English with `i` and simply make the model learn that `e` and `i` are the same letter for a better word error rate. This would destroy the meaning of words such as `fell` -> `fill` . However, it is totally fine to normalize (e.g. lowercase) all letters, remove punctuation. There can be a lot of language-specific exceptions and in case you are not sure whether your target transcription pre-processing is allowed, please ask on the Discord channel.

Uff, that was a lot of text describing how to make sure your `eval.py` script is in the correct format. If you have any questions, please ask openly in Discord.

Great, now that we have adapted the `eval.py` script, we can lean back and run the evaluation. First, one should evaluate the model on Common Voice 7's test data. This might already have been done for your acoustic model during training but in case you added an *n-gram* language model after having fine-tuned the acoustic model, you should now see a nice improvement.

The command to evaluate our test model [xls-r-300m-sv](#) on Common Voice 7's test data is the following:

```
cd xls-r-300m-sv
./eval.py --model_id ./ --dataset mozilla-foundation/common_voice_7_0 --config sv-SE
--split test --log_outputs
```

To log each of the model's predictions with the target transcriptions, you can just add the `--log_outputs` flag.

Running this command should automatically create the file: `mozilla-foundation_common_voice_7_0_sv-SE_test_eval_results.txt` that contains both the word- and character error rate.

In a few days, we will give everybody access to some real-world audio data for as many languages as possible. If your language has real-world audio data, it will most likely have audio input of multiple minutes. 😊Transformer's [ASR pipeline](#) supports audio chunking out-of-the-box. You only need to specify how long each audio chunk should be (`chunk_length_s`) and how much audio stride (`stride_length_s`) each chunk should use. For more information on the chunking works, please have a look at [this nice blog post](#).

In the case of `xls-r-300m-sv` , the following command can be run:

```
cd xls-r-300m-sv
./eval.py --model_id hf-test/xls-r-300m-sv --dataset <to-be-announced> --config sv -
-split validation --chunk_length_s 5.0 --stride_length_s 1.0 --log_outputs
```

Great, now you should have successfully evaluated your model. Finally, there is one **important** thing you should do so that your model is taken into account for the final evaluation. You should add two tags to your model, one being `robust-speech-event` , one being the ISO code of your chosen language, e.g. `"sv"` for the exemplary model we used above. You can find a list of all available languages and their ISO code [here](#).

To add the tags, simply edit the README.md of your model repository and add

```
- "sv"
- "robust-speech-event"
```

under `tags:` as done [here](#).

To verify that you've added the tags correctly make sure that your model appears when clicking on [this link](#).

Great that's it! This should give you all the necessary information to evaluate your model. For the final evaluation, we will verify each evaluation result to determine the final score and thereby the winning models for each language.

The final score is calculated as follows:

```
FINAL_SCORE = 1/3 * WER_Common_Voice_7_test + 1/3 * WER_REAL_AUDIO_DEV + 1/3 *
WER_REAL_AUDIO_TEST
```

The dataset `WER_REAL_AUDIO_TEST` is hidden and will only be published at the end of the robust speech challenge.

If there is no real audio data for your language the final score will be computed solely based on the Common Voice 7 test dataset. If there is also no Common Voice 7 test dataset for your language, we will see together how to score your model - if this is the case, please don't be discouraged. We are especially excited about speech recognition systems of such low-resource languages and will make sure that we'll decide on a good approach to evaluating your model.

Prizes

TODO(Patrick, Omar, ...)

Communication and Problems

If you encounter any problems or have any questions, you should use one of the following platforms depending on your type of problem. Hugging Face is an "open-source-first" organization meaning that we'll try to solve all problems in the most public and most transparent way possible so that everybody in the community profits.



The following table summarizes what platform to use for which problem.

- Problem/question/bug with the 🤖 Datasets library that you think is a general problem that also impacts other people, please open an [Issues on Datasets](#) and ping @anton-l and @patrickvonplaten.
- Problem/question/bug with the 🤖 Transformers library that you think is a general problem that also impacts other people, please open an [Issues on Transformers](#) and ping @anton-l and @patrickvonplaten.
- Problem/question with a modified, customized training script that is less likely to impact other people, please post your problem/question [on the forum](#) and ping @anton-l and @patrickvonplaten.
- Questions regarding access to the OVHcloud GPU, please ask in the Discord channel **#ovh-support**.
- Other questions regarding the event, rules of the event, or if you are not sure where to post your question, please ask in the Discord channel **#sprint-discussions**.



Talks

We are very excited to be hosting 2 days of talks from Kensho-Technologies, Mozilla's Common Voice, Meta AI Research and Hugging Face.

Thursday, January 20th

Speaker	Topic	Time	Video
Patrick von Platen, Hugging Face	Introduction to Robust Speech Challenge	4h30pm - 5h00pm UTC	
Raymond Grossman and Jeremy Lopez, Kensho-Technologies	Pyctcdecode & Speech2text decoding	5h30pm - 6h00pm UTC	

Friday, January 21th

Speaker	Topic	Time	Video
Gabriel Habayeb, Mozilla Common Voice	Unlocking global speech with Mozilla Common Voice	4h30pm - 5h00pm UTC	
Changhan Wang, Meta AI Research	XLS-R: Large-Scale Cross-lingual Speech Representation Learning on 128 Languages	5h30pm - 6h00pm UTC	

Talks & Speakers

Patrick von Platen, Research Engineer, Hugging Face

- Talk: Introduction to Robust Speech Challenge
- Abstract: In this talk, Patrick outlines the Robust Speech Challenge and gives tips and tricks on how to train and evaluate speech recognition systems with 🤖 Transformers and 🤖 Datasets, and PyTorch.

- Speaker info: Patrick von Platen is a research engineer at Hugging Face and one of the core maintainers of the popular Transformers library. He specializes in speech recognition, encoder-decoder models, and long-range sequence modeling. Before joining Hugging Face, Patrick researched speech recognition at Uber AI, Cambridge University, and RWTH Aachen University.

Raymond Grossman, Jeremy Lopez, Machine Learning Engineer, Kensho Technologies

- Talk: PyCTCDecode & Speech2text decoding
- Abstract: PyCTCDecode is a fast and feature-rich CTC beam search decoder for speech recognition written in Python, providing n-gram (kenlm) language model support similar to PaddlePaddle's decoder, but incorporating many new features such as byte pair encoding and real-time decoding to support models like Nvidia's Conformer-CTC or Facebook's Wav2Vec2.
- Speaker info :
 - Raymond works as a machine learning engineer at Kensho Technologies, specializing in speech and natural language domains. Before coming to Kensho, he studied mathematics at Princeton and was an avid Kagglor under the moniker @ToTrainThemIsMyCause.
 - Jeremy is a machine learning engineer at Kensho Technologies and has worked on a variety of different topics including search and speech recognition. Before working at Kensho, he earned a PhD in experimental particle physics at MIT and continued doing physics research as a postdoc at the University of Colorado Boulder.

Gabriel Habayeb, Data Engineer, Common Voice @ Mozilla

- Talk: Unlocking global speech with Mozilla Common Voice
- Abstract: Hear from Common Voice Data Engineer Gabriel Habayeb (Mozilla Foundation) as he talks about how Common Voice makes it easy to crowdsource voice data in global languages, as well as getting key insights into the dataset itself, how we maintain quality, use metadata - and our plans for the future!
- Speaker info: Gabriel is a software developer with the Common Voice team at the Mozilla Foundation with a focus on data engineering. Before joining the Foundation, he spent the last six years working across different industries, including education, enterprise and not-for-profit organizations.

Changhan Wang, Main author of XLS-R and Research Engineer, Meta AI Research

- Talk: XLS-R: Large-Scale Cross-lingual Speech Representation Learning on 128 Languages
- Abstract: In this talk, Changhan will present XLS-R, a large-scale model for cross-lingual speech representation learning based on wav2vec 2.0. XLS-R has up to 2B parameters and was trained on nearly half a million hours of publicly available speech audio in 128 languages, an order of magnitude more public data than the largest known prior work. On the CoVoST-2 speech translation benchmark, XLS-R improves the previous state of the art by an average of 7.4 BLEU over 21 translation directions into English. For speech recognition, XLS-R improves over the best known prior work on BABEL, MLS, CommonVoice as well as VoxPopuli, lowering error rates by 14-34% relative on average. XLS-R also sets a new state of the art on VoxLingua107 language identification. The XLS-R team hopes to work together with the open-source community to improve speech processing tasks for many more languages of the world.

General Tips and Tricks

- Memory efficient training:

In case, you are getting out-of-memory errors on your GPU, we recommend to use [bitsandbytes](#) to replace the native memory-intensive Adam optimizer with the one of `bitsandbytes`. You can simply run the script

`./run_speech_recognition_ctc_bnb.py` provided in this folder that makes use of `bitsandbytes` instead of the official one.

- Dataset streaming

TODO(Patrick)