Reproducing Distil-Whisper

Keywords: distil-whisper, reproduction, troubleshooting, whisper, Google Cloud Engine, audio-to-text

Which team member did what:

- Jelle: Evaluate existing code (English model)
- Cinta: Hyperparameter check (English model)
- Mees: Train a Dutch model
- Quirine: Evaluate the Dutch model

Introduction

The goal of this project was to reproduce the results of one table of the Distil-Whisper paper by Gandhi, Patrick, and Rush (2023). We attempted to distill our own English model of Whisper following the authors' method to evaluate the performance in comparison to their results, as well as distill and evaluate a model trained on Dutch audio datasets. In this blog post, we present our methods, results, and issues encountered during the reproduction project.

Background information

Whisper is a versatile speech recognition (speech-to-text) model (Radford et al, 2022). It is a multitasking model that can perform multilingual voice recognition, speech translation, and language identification and is trained on a large dataset that includes various types of speech audio.

Gandhi et al. (2023) introduced a strategy in which pseudo-labeling is used to compile a large-scale open-source dataset which can be used to reduce the Whisper model to a more manageable version known as Distil-Whisper. This reduced model is obtained through 'knowledge distillation', a method of 'compressing' transformer models so that they are smaller in size and thus faster at inference time, but without significant loss of performance.

According to the authors, Distil-Whisper is 49% smaller, 5.8 times faster, and within 1% performance on out-of-distribution (OOD) short-form audio. The performance of in-distribution (ID) short-form audio does not seem to differ between whisper-medium and distil-whisper-medium, which corresponds to a Word Error Rate (WER) of 14.

Moreover, Distil-Whisper maintains the robustness of the Whisper model to difficult acoustic conditions, while being less prone to hallucination errors. This is specifically highlighted by Table 16 in the paper, which shows the average WER scores for the Whisper and Distil-Whisper checkpoints across the four OOD short-form test sets.

Furthermore, the paper demonstrates that Distil-Whisper can be used alongside Whisper using speculative decoding to achieve the same outputs as the original model with a two times faster Real Time Factor (RTF). In other words, the original and distilled model can be used together to obtain transcriptions faster without a loss in performance.

Definitions

The following terms and abbreviations are used throughout this text.

Word Error Rate (WER):
The Word Error Rate is a common metric used to measure the performance of language models. It is calculated by first determining the number of errors, which is the difference between the original and transcribed text, taking into account wrong substitutions, insertions, and deletions. This is then divided by the total number of words.

Real-Time Factor (RTF):
The Real-Time Factor is a metric used to measure the speed of generating an output. This factor is calculated by first measuring the time it takes to generate a transcription, also referred to as inference. The inference time is then divided by the duration of the input audio fragment. This is done to account for different lengths of fragments.

Datasets

The datasets we used in this reproduction project (Common Voice 13 and VoxPopuli) are hosted on Huggingface, an open-source machine learning platform that allows users to browse through models and datasets uploaded by other users. Common Voice is an audio dataset in which contributors record themselves reading various texts in a variety of languages (Ardila et al., 2020). VoxPopuli is a multilingual voice dataset made up of recordings from European Parliament events from 2009-2020 (Wang et al., 2021). The speakers in this political oratory are primarily non-native speakers.

Reproduction

Plan

As mentioned in the introduction, Table 16 highlights how Distil-Whisper retains the robustness of the Whisper model in terms of WER, therefore we would like to focus our assignment towards the reproduction of the findings represented in Table 16 by Gandhi et al.

Because of the large amount of time required to obtain all data in the table, we have decided on reproducing a ‘smaller’ version of the table. Firstly, we only train on two datasets: Common Voice 13 and VoxPopuli. We also evaluate the model on these datasets, together with one OOD dataset: FLEURS. Secondly, as the authors did not create a distilled version of the tiny model and large-v2 might require too many resources, we will only reproduce the medium models: medium.en and distil-medium.en.

The table below shows a summary of what is included within our reproduction study through highlighted values.

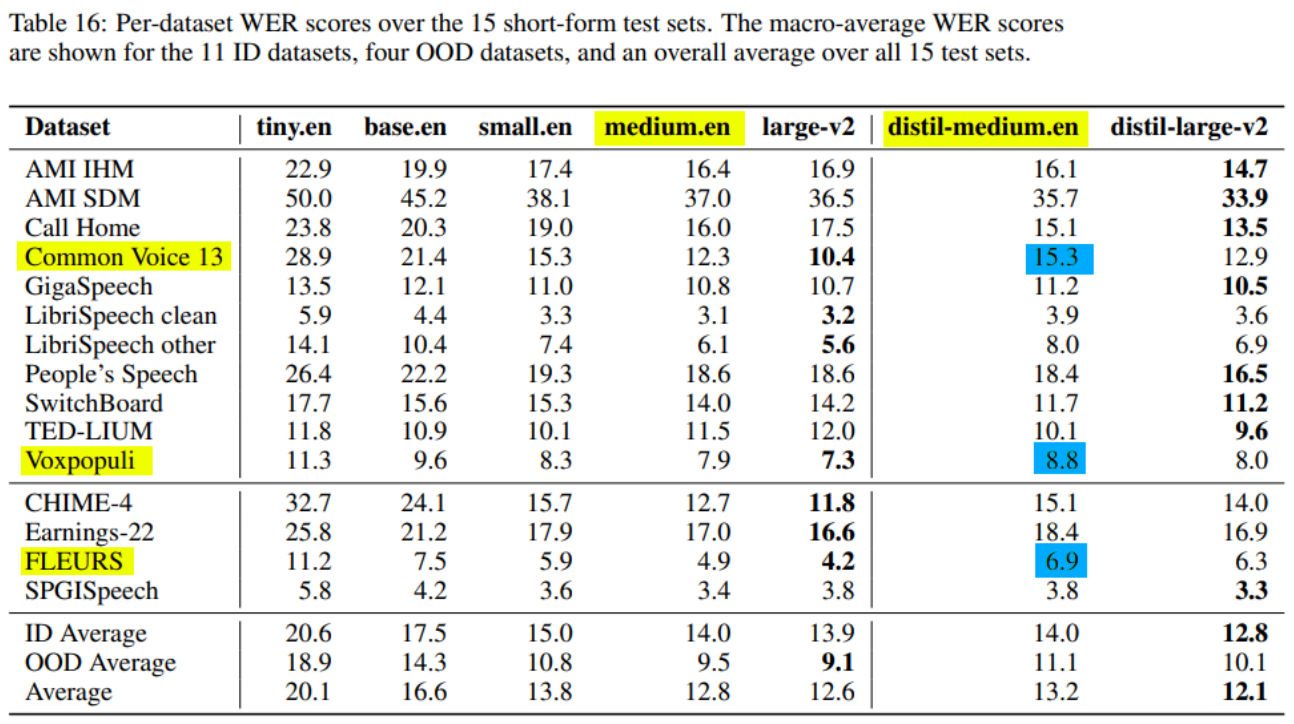


Table 16: Per-dataset WER scores over the 15 short-form test sets. The macro-average WER scores are shown for the 11 ID datasets, four OOD datasets, and an overall average over all 15 test sets.

Dataset	tiny.en	base.en	small.en	medium.en	large-v2	distil-medium.en	distil-large-v2
AMI IHM	22.9	19.9	17.4	16.4	16.9	16.1	**14.7**
AMI SDM	50.0	45.2	38.1	37.0	36.5	35.7	**33.9**
Call Home	23.8	20.3	19.0	16.0	17.5	15.1	**13.5**
Common Voice 13	28.9	21.4	15.3	12.3	**10.4**	15.3	12.9
GigaSpeech	13.5	12.1	11.0	10.8	10.7	11.2	**10.5**
LibriSpeech clean	5.9	4.4	3.3	3.1	**3.2**	3.9	3.6
LibriSpeech other	14.1	10.4	7.4	6.1	**5.6**	8.0	6.9
People's Speech	26.4	22.2	19.3	18.6	18.6	18.4	**16.5**
SwitchBoard	17.7	15.6	15.3	14.0	14.2	11.7	**11.2**
TED-LIUM	11.8	10.9	10.1	11.5	12.0	10.1	**9.6**
Voxpopuli	11.3	9.6	8.3	7.9	**7.3**	8.8	8.0
CHIME-4	32.7	24.1	15.7	12.7	**11.8**	15.1	14.0
Earnings-22	25.8	21.2	17.9	17.0	**16.6**	18.4	16.9
FLEURS	11.2	7.5	5.9	4.9	**4.2**	6.9	6.3
SPGISpeech	5.8	4.2	3.6	3.4	3.8	3.8	**3.3**
ID Average	20.6	17.5	15.0	14.0	13.9	14.0	**12.8**
OOD Average	18.9	14.3	10.8	9.5	**9.1**	11.1	10.1
Average	20.1	16.6	13.8	12.8	12.6	13.2	**12.1**

Our team was split into two factions: two members would focus on the English model by means of evaluating existing code (Jelle Woudstra) and the tuning of hyperparameters (Cinta Parengkuan), while the other two would train (Mees Chammat) and evaluate (Quirine Engbers) a Dutch version of the model.

Training & evaluation steps

The code used in the distil-whisper paper is available on GitHub. It contains a training folder with a README that describes the following training steps:

- Step 0: Setting up the environment
- Step 1: Pseudo-labeling the datasets with the original whisper model
- Step 2: Initialization of the distil-whisper model
- Step 3: Training of the distil-whisper model
- Step 4: Evaluation of the distil-whisper model

In this reproduction, we consistently used commit '398e93b' (dated March 21) of the code for every step up to and including the training, as to avoid pulling new upstream changes from the repository that change our setup and result in errors. Only evaluation used the latest version at the time of writing, commit '7838def'.

Issues

While attempting to reproduce the training steps from the paper, we ran into several problems, ranging from the datasets being too large to fit on disk to GPU compute limitations and from unresponsive terminals to incompatible libraries and modules. We tried to run through the training steps locally (on our laptops and PCs), on Kaggle, with a Google Compute Engine (GCE) VM, and on Google Colab. In the end, Google Colab connected to a GCE VM was the most promising setup that allowed us to run through all training steps at least once. The next section highlights the key issues and bottlenecks we encountered for each of the 4 environments (Local, Kaggle, GCE VM, Colab) we used.

Local

The easiest first step was to try and run the training code locally, on our own devices. Setting up the environment (Step 0) went fairly smoothly. The only major issue discovered in Step 0 was that the run_pseudo_labelling.py script contained a call to a function from the transformer library that is now deprecated. Once this root of the problem was identified, it was solved by replacing the text 'model = model.to_bettertransformer()' at line 532 with 'pass'.

The real problems with our local attempt started at the pseudo-labelling step (Step 1). The distil-large-v2 model did not fit in our GPU's VRAM (NVIDIA GeForce GTX 1060 Laptop version, 6 GB), so we tried to use whisper-tiny instead.

Furthermore, the audio files were transcribed at a very slow pace: transcribing a single sample from the Dutch validation or test split from the commonvoice 13 dataset took about 30 minutes. Multiply that by the 560 samples in each of these splits, and it's clear that this would take too long if we were to pseudo-label training sets with tens of thousands of samples. For comparison, the authors were able to pseudo-label the Hindi train split in about 1 hour on an A100 GPU, while we only had access to GTX 1060s. We could not noticeably speed up this process by changing parameters like the number of workers.

Therefore, reproducing the results locally was not feasible. We recommend only attempting a local setup if you have access to a T4, A100, or other powerful high-memory GPU for machine-learning applications.

Kaggle

Our next attempt was in [Kaggle](), a platform that provides two 16 GB T4 GPUs for free. Step 0 was eventually completed successfully, but not without hiccups. Some setup steps require commands that prompt for user input (e.g. submitting the Hugging Face token, setting up accelerate, and starting wandb), and for some reason, we were unable to provide input into Kaggle's terminal/console to answer these prompts. We tried private tabs, turning off extensions, different browsers, and different devices/OSes, but to no avail. This is likely a problem on Kaggle's side.

Some of these prompts could be replaced with alternative, prompt-less commands (e.g. by using huggingface-cli login --token <TOKEN_ID>). This was a time-consuming process as most of these commands are not properly documented anywhere.

The initialization and training steps (Step 2 & 3) were impossible to pass because we could not create and use a Hugging Face repository for our trained model from Kaggle's command line. However, the pseudo-labelling (Step 1) and evaluation (Step 4) were doable, so we were able to use Kaggle purely for pseudo-labelling and evaluation to save on Google Cloud credits (see next section).

In short, we do not recommend using Kaggle for reproducing this paper as long as their interface has this command line bug. Once it is fixed, we think it would be the perfect free alternative to Google Cloud.

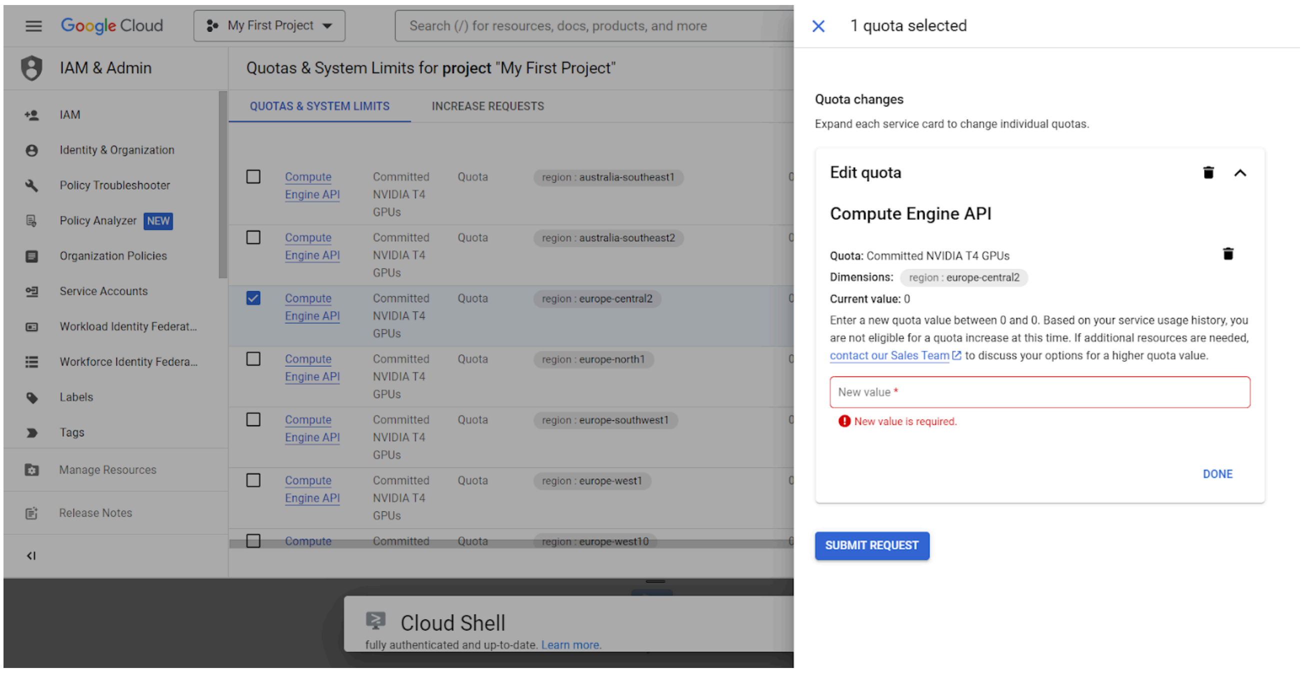
Google Compute Engine VM

Setting up a Virtual Machine in Google Cloud Compute was a challenge in and of itself because there are many options to choose from; Python version, CUDA version, amount of RAM, type and number of GPUs, disk size…

Unfortunately, none of our group had experience with the configurations, and even our TA could not provide us with guidelines. This resulted in a trial-and-error process.

- 'E2' with 4GB RAM and Python 3.7. This was not sufficient because the huggingface-hub library (required for the transformer library which is listed in the requirements.txt in distil-whisper's repo) required a Python version later than 3.7, even though this GitHub issue stated that Python 3.7 would be sufficient.
- 'E2' with 4GB RAM and Python 3.10. This time Step 0 ran flawlessly, but the pseudo-labeling (Step 1) caused a memory overflow problem. 4GB RAM is not enough.
- 'E2' with 16GB RAM and Python 3.10. With this VM, we ran into unsolvable issues in the pseudo-labelling (1), training (3), and evaluation (4) steps.
    - In Step 1, we ran into the issue "slow_conv2d_cpu" not implemented for 'Half'.
    - In Step 3, we got the following error: `RuntimeError: Input type (torch.FloatTensor) and weight type (torch.HalfTensor) should be the same or input should be a MKLDNN tensor and weight is a dense tensor.` `Implying that the input and weights must be the same type of tensor.` We tried several solutions found online: forcing the training to run on the GPU (no change), updating the dataset and transformer library (no change), manually transforming the weights matrix or input matrix (memory overflow), and trying different commit versions of distil-whisper (newer/older versions either caused new errors or the same error)
    - In Step 4, there was a mismatch of libraries, because the model needed to be in a specific format. This was odd given that Kaggle (and Colab, see below) did not have this issue. Trying to down-/upgrade to different library versions used in Kaggle gave a warning that this was not possible. It is still unclear which library caused this issue.

Another annoying problem with GCE is that it was insanely hard for another team member to even set up the VM. Google did not offer them an option to increase their GPU quota, and they had to contact Customer Support 3 times only to be told non-business consumers need to pay for support. See the screenshots below.



Quotas & System Limits for **project** "My First Project"

☐	Compute Engine API	Committed NVIDIA T4 GPUs	Quota	region : australia-southeast1
☐	Compute Engine API	Committed NVIDIA T4 GPUs	Quota	region : australia-southeast2
✓	Compute Engine API	Committed NVIDIA T4 GPUs	Quota	region : europe-central2
☐	Compute Engine API	Committed NVIDIA T4 GPUs	Quota	region : europe-north1
☐	Compute Engine API	Committed NVIDIA T4 GPUs	Quota	region : europe-southwest1
☐	Compute Engine API	Committed NVIDIA T4 GPUs	Quota	region : europe-west1
☐	Compute	Committed	Quota	region : europe-west10

fully authenticated and up-to-date. Learn more.

1 quota selected

Quota changes

Expand each service card to change individual quotas.

Edit quota

Compute Engine API

Quota: Committed NVIDIA T4 GPUs

Dimensions: region : europe-central2

Current value: 0

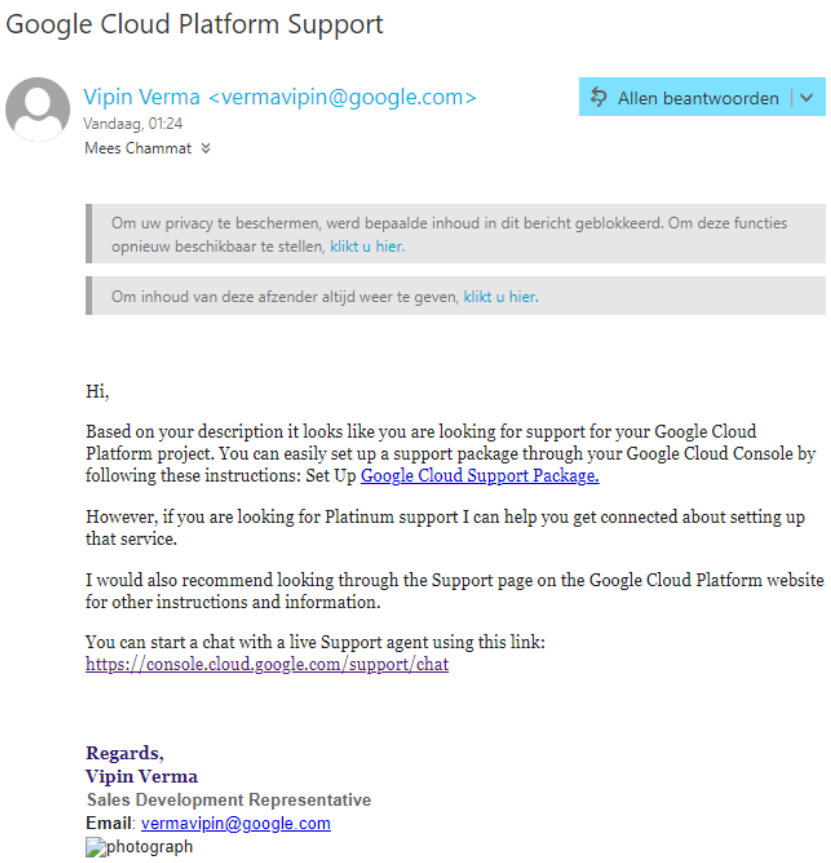
Enter a new quota value between 0 and 0. Based on your service usage history, you are not eligible for a quota increase at this time. If additional resources are needed, contact our Sales Team to discuss your options for a higher quota value.

New value *

New value is required.

DONE

SUBMIT REQUEST



Google Cloud Platform Support

Vipin Verma <vermavipin@google.com>
Vandaag, 01:24
Mees Chammat

Om uw privacy te beschermen, werd bepaalde inhoud in dit bericht geblokkeerd. Om deze functies opnieuw beschikbaar te stellen, klikt u hier.

Om inhoud van deze afzender altijd weer te geven, klikt u hier.

Hi,

Based on your description it looks like you are looking for support for your Google Cloud Platform project. You can easily set up a support package through your Google Cloud Console by following these instructions: Set Up Google Cloud Support Package.

However, if you are looking for Platinum support I can help you get connected about setting up that service.

I would also recommend looking through the Support page on the Google Cloud Platform website for other instructions and information.

You can start a chat with a live Support agent using this link:
https://console.cloud.google.com/support/chat

Regards,
Vipin Verma
Sales Development Representative
Email: vermavipin@google.com
photograph

We suspect this is because they started the 300$ free trial on Google Cloud, which excludes GPU access. On a secondary Google account, there was not even an option to see GPUs. A week later, it was suddenly possible to request GPU quota increase. After that, everything worked well, but we ran into the same issues as mentioned previously.

To summarize, we were unable to use the VM for pseudo-labelling, training, and evaluation. We gave up once we found out Colab was more promising. We do not recommend using the GCE VM directly for this project.

Google Colab

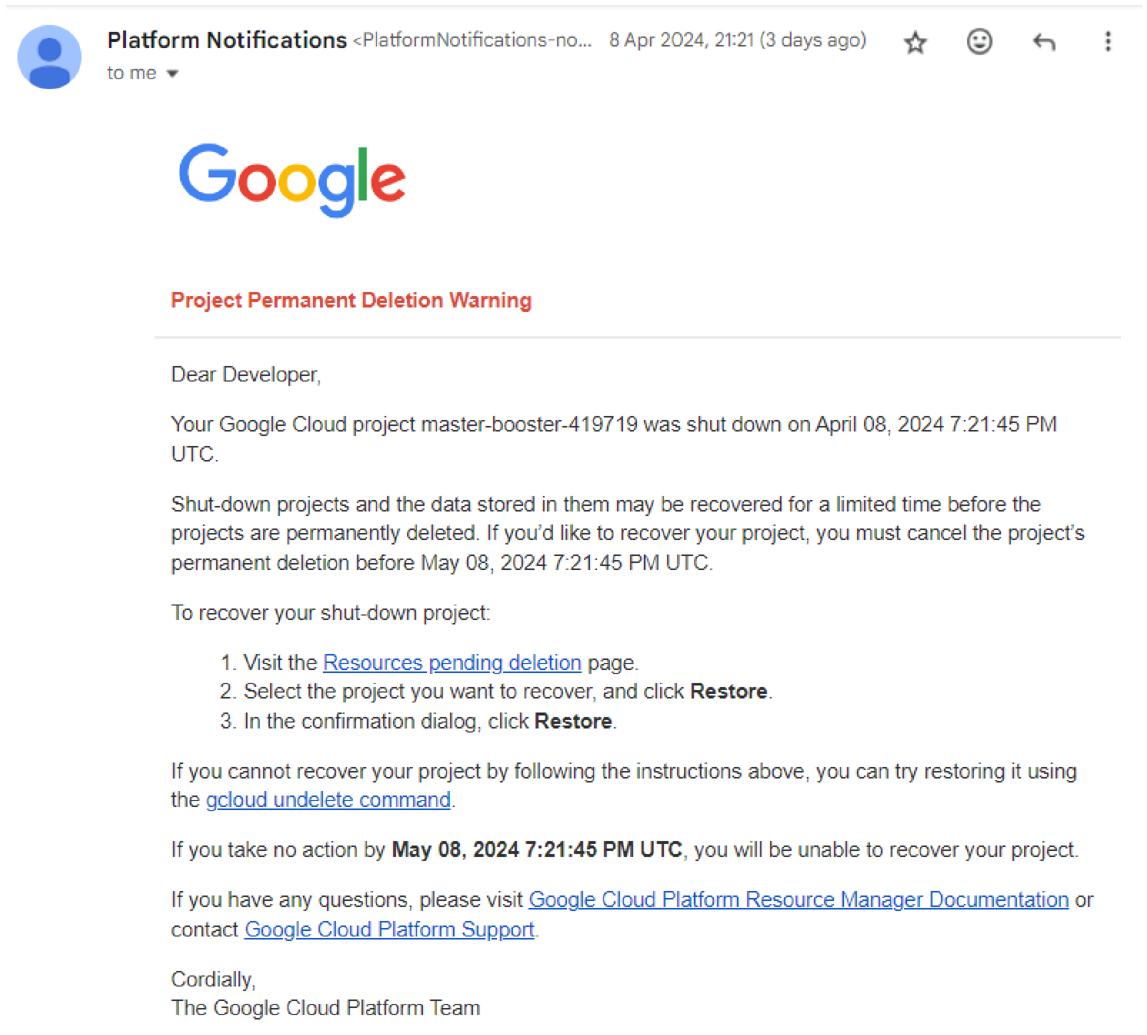
While one group member was troubleshooting the GCE VM, another tried running the training steps in Google Colab, as this platform offers a free T4 GPU based on availability, so we could at least test some of our code this way.

Step 0 was similar to local setup, and both pseudo-labelling (1) and initialization (2) worked too. The only problem with Colab was that the free runtime could time out at any moment, so the training step (3) was impossible to do. Luckily, Colab can be connected to a GCE VM, and that is what we did and eventually used.

We do not recommend using the free version of Google Colab for training or evaluating a large dataset of the model, only for standalone pseudo-labelling or evaluating small datasets.

Google Colab + GCE VM (recommended working setup)

By connecting our GCE VMs to Google Colab, we were able to run all training steps more reliably. The VM configuration we used is 'n1-highmem-2', 16GB RAM, 300GB disk space, with one T4 GPU, located in zone eu-central2. This amounted to a monthly cost of $350, which is the cheapest configuration we could get. This does mean we could only use GCE for about a week per person until we ran out of Education credits ($50). In fact, we all ran out of credits at the end of the project, see the screenshot below.



Platform Notifications <PlatformNotifications-no... 8 Apr 2024, 21:21 (3 days ago)
to me

Google

Project Permanent Deletion Warning

Dear Developer,

Your Google Cloud project master-booster-419719 was shut down on April 08, 2024 7:21:45 PM UTC.

Shut-down projects and the data stored in them may be recovered for a limited time before the projects are permanently deleted. If you'd like to recover your project, you must cancel the project's permanent deletion before May 08, 2024 7:21:45 PM UTC.

To recover your shut-down project:

1. Visit the Resources pending deletion page.
2. Select the project you want to recover, and click **Restore**.
3. In the confirmation dialog, click **Restore**.

If you cannot recover your project by following the instructions above, you can try restoring it using the gcloud undelete command.

If you take no action by **May 08, 2024 7:21:45 PM UTC**, you will be unable to recover your project.

If you have any questions, please visit Google Cloud Platform Resource Manager Documentation or contact Google Cloud Platform Support.

Cordially,
The Google Cloud Platform Team

Still, we highly recommend this method if you have the financial resources to use it, especially if a local approach is not feasible.

Method

Our Google Colab/Jupyter notebooks can be found in our Github repository. In this section, we explain the steps needed to run through the entire training sequence from Step 0 to Step 4, as well as the code we used in our Jupyter notebooks.

Step 0: Setting up the environment

In this step, we install and import all required libraries, repositories and modules to start the training sequence. The code for this is self-explanatory and most of it was copied straight from the README on Github. There are, however, a few necessary tweaks that we added ourselves:

- We added code cells that allow us to delete and download folders from Google Colab, since the Colab GUI does not let us do this directly.
- There is a code cell that automatically removes the "to_bettertransformer()" function from run_pseudo_labelling.py (this resolves the issue mentioned under "Issues - Local" in this blog post)
- The manual *accelerate* setup is skipped by using the function write_basic_config().
- We set the locale of the notebook to UTF-8, else the accelerate bash scripts won't run
- We modify a PyTorch environment variable to lower the amount of GPU memory that PyTorch occupies.

Step 1: Pseudo-labelling

Pseudo-labelling can be done using an accelerate command that launches run_pseudo_labelling.py, which is given in the distil-whisper training tutorial. We used the following argument values:

```
!accelerate launch distil-whisper/training/run_pseudo_labelling.py \
  --model_name_or_path "openai/whisper-medium" \ # whisper-large-v2 does not fit in GPU
  --dataset_name "facebook/voxpopuli" \
  --dataset_config_name "nl" \
  --dataset_split_name "train" \
  --text_column_name "normalized_text" \
  --id_column_name "audio_id" \
  --output_dir "./voxpopuli_nl_TRAIN_pseudo_labelled" \
  --wandb_project "distil-whisper-labelling" \
  --per_device_eval_batch_size 16 \
  --dtype "float16" \
  --dataloader_num_workers 1 \ # must be set to 1, only 1 GPU is used
  --preprocessing_num_workers 1 \
  --logging_steps 500 \
  --max_label_length 128 \
  --report_to "wandb" \
  --language "nl" \
  --task "transcribe" \
  --return_timestamps \
  --attn_type "flash_attn" \
  --streaming False \
  --generation_num_beams 1 \
  --decode_token_ids False \
  --push_to_hub
```

Note the following important changes compared to the script given by the authors:

- We use whisper-medium as the model for pseudo-labelling, as whisper-large-v2 does not fit in the T4's memory (16GB). By setting per_device_batch_size to 16, we can just about fit both the medium model and the batch of audio files in memory.
- We set both num_workers parameters to 1 because we only have 1 GPU.

Below are the dataset-specific arguments we used for the other training sets. Note that the evaluation and test sets were also pseudo-labelled with corresponding settings (e.g. dataset_split_name "validation", see distil-whisper tutorial on GitHub).

	Common Voice 13	voxpopuli
EN	`--dataset_name "mozilla-foundation/common_voice_13_0" \ --dataset_config_name "en" \ --dataset_split_name "train" \ --text_column_name "sentence" \ --id_column_name "path" \ --output_dir "./common_voice_13_0_TRAIN_pseudo_labelled" \`	`--dataset_name "facebook/voxpopuli" \ --dataset_config_name "en" \ --dataset_split_name "train" \ --text_column_name "normalized_text" \ --id_column_name "audio_id" \ --output_dir "./voxpopuli_TRAIN_pseudo_labelled" \`
NL	`--dataset_name "mozilla-foundation/common_voice_13_0" \ --dataset_config_name "nl" \ --dataset_split_name "train" \ --text_column_name "sentence" \ --id_column_name "path" \ --output_dir "./common_voice_13_0_nl_TRAIN_pseudo_labelled" \`	`--dataset_name "facebook/voxpopuli" \ --dataset_config_name "nl" \ --dataset_split_name "train" \ --text_column_name "normalized_text" \ --id_column_name "audio_id" \ --output_dir "./voxpopuli_nl_TRAIN_pseudo_labelled" \`

Step 2: Initialization

For initialization, we followed the same steps as the authors: creating a model repo on HuggingFace and initializing the student model using create_student_model.py. See our Colab notebook for details.

Step 3: Training

For the training step, we trained the student model on the pseudo-labelled VoxPopuli and CommonVoice 13 training splits and evaluated it on the corresponding validation splits. This could again be done using an accelerate command.

English model

The hyperparameter that we wanted to vary was the learning_rate, thus we aimed to train the English model twice: once with learning_rate 0.0001 (original learning rate used by the authors) and once with learning_rate 0.0001. We used the following parameters for training the English model:

```
!accelerate launch /content/distil-whisper/training/run_distillation.py \
  --model_name_or_path "/content/distil-whisper-nl/distil-large-v2-init" \
  --teacher_model_name_or_path "openai/whisper-medium" \
  --train_dataset_name
"JelleWo/common_voice_13_0_en_VALTEST_pseudo_labelled+JelleWo/vox_populi_en_VALTEST_pseud
o_labelled" \
  --train_dataset_config_name "en+en" \
  --train_split_name "validation+validation" \
  --text_column_name "raw_text+sentence" \
  --train_dataset_samples "10+10" \
  --eval_dataset_name
"JelleWo/common_voice_13_0_en_VALTEST_pseudo_labelled+JelleWo/vox_populi_en_VALTEST_pseud
o_labelled" \
  --eval_dataset_config_name "en+en" \
  --eval_split_name "test+test" \
  --eval_text_column_name "raw_text+sentence" \
  --eval_steps 1000 \
  --save_steps 1000 \
  --warmup_steps 50 \
  --learning_rate 0.0001 \
  --lr_scheduler_type "constant_with_warmup" \
  --logging_steps 10 \
  --save_total_limit 1 \
  --max_steps 1000 \
  --wer_threshold 10 \
  --per_device_train_batch_size 8 --per_device_eval_batch_size 8 \
  --dataloader_num_workers 1 \
  --preprocessing_num_workers 2 \
  --ddp_timeout 7200 \
  --dtype "float16" \
  --output_dir "/content/distil-whisper-eng/" \
  --do_train \
  --do_eval False \
  --gradient_checkpointing \
  --overwrite_output_dir \
  --predict_with_generate \
  --freeze_encoder \
  --streaming True \
  --push_to_hub \
  --language "en"
```

Dutch model

We used the following parameters for training the Dutch model:

```
!accelerate launch /content/distil-whisper/training/run_distillation.py \
  --model_name_or_path "/content/distil-whisper-nl/distil-large-v2-init" \
  --teacher_model_name_or_path "openai/whisper-medium" \
  --train_dataset_name
"monsoonery/voxpopuli_nl_TRAIN_pseudo_labelled+monsoonery/common_voice_13_0_nl_TRAIN_pseu
do_labelled" \
  --train_dataset_config_name "nl+nl" \
  --train_split_name "train+train" \
  --text_column_name "normalized_text+sentence" \
  --train_dataset_samples "10+10" \
  --eval_dataset_name
"monsoonery/voxpopuli_nl_EVAL_pseudo_labelled+monsoonery/common_voice_13_0_nl_EVAL_pseudo
_labelled" \
  --eval_dataset_config_name "nl+nl" \
  --eval_split_name "validation+validation" \
  --eval_text_column_name "normalized_text+sentence" \
  --eval_steps 1000 \
  --save_steps 1000 \
  --warmup_steps 50 \
  --learning_rate 0.0001 \
  --lr_scheduler_type "constant_with_warmup" \
  --logging_steps 10 \
  --save_total_limit 1 \
  --max_steps 1000 \
  --wer_threshold 10 \
  --per_device_train_batch_size 8 --per_device_eval_batch_size 8 \
  --dataloader_num_workers 1 \
  --preprocessing_num_workers 2 \
  --ddp_timeout 7200 \
  --dtype "float16" \
  --output_dir "/content/distil-whisper-nl/" \
  --do_train \
  --do_eval False \
  --gradient_checkpointing \
  --overwrite_output_dir \
  --predict_with_generate \
  --freeze_encoder \
  --streaming True \
  --push_to_hub \
  --language "nl"
```

Note again the following changes to the argument values:

- We set streaming True, as the combined datasets were too large to fit on the 300GB disk available on GCE.
- We set the per_device_batch_size parameters to 8 because 16 would give us a CUDA out-of-memory error.

Step 4: Evaluation

In the last step, we executed the run_eval.py script from the ‘7838def’ commit using accelerate, which returns the WER and RTF values. Our trained Dutch model was evaluated on the test splits from CommonVoice, Voxpopuli, and FLEURS. We used the parameters shown in the code block below:

```
python ./run_eval.py \
  --model_name_or_path "./distil-whisper-nl" \
  --dataset_name
"monsoonery/common_voice_13_0_nl_TEST_pseudo_labelled+Quirina/voxpopuli_nl_TEST_pseudo_la
belled+google/fleurs" \
  --dataset_config_name "nl+nl+nl_nl" \
  --dataset_split_name "test+test+test" \
  --text_column_name "sentence+normalized_text+transcription" \
  --batch_size 16 \
  --dtype "float16" \
  --generation_max_length 128 \
  --language "nl" \
  --attn_implementation "sdpa" \
  --streaming
```

Note the following manual change made to the run_eval.py script:

- Any appearance of 'attn_implementation' has to be removed in the code, because version 4.37.0 of the transformer library does not have this parameter.

Results

English audio transcribing

The main goal was the reproduction of the English version of the distil-whisper model. This was unfortunately not feasible due to the computational and time-related costs of the model, and thus we have no results to present for this model

Not only were the English training datasets too large to fit on disk, but they also took too long to be pseudo-labelled, which quickly depleted our 50$ GCE credits. We were unable to find smaller English training sets, so we decided to train on validation sets and evaluate on the test sets. Even with the dataset size issue resolved, new errors arose while trying to train the English model, which was time-consuming to resolve. Eventually, we simply burned through our 50$ budgets.

Dutch audio transcribing

The training sequence for the Dutch model was successful. The results, however, are terrible (Table 1). The WER is very high for all three datasets, but the RTF is also high so it transcribed the audio quite fast. We see that the Common Voice dataset has the best performance with a WER of 32.4. This is because the Common Voice dataset was larger, with more audio files in all splits. Performance was worst on the FLEURS dataset. This is to be expected as it is an out-of-distribution (OOD) dataset.

Table 2:
Evaluation results of distil-whisper-nl, our Dutch model.

Dataset	**WER**	**RTF**	**Time**
Common Voice 13 (monsoonery/common_voice_13_0_nl_TEST_pseudo_labelled)	32.4	20.2	1:16:33
Voxpopuli (Quirina/voxpopuli_nl_TEST_pseudo_labelled)	55.1	32.3	0:07:35
FLEURS (google/fleurs)	67.0	37.9	0:03:40

Table 2 shows the per-dataset WER of our model compared to distil-medium-en, which is most similar to our Dutch model in terms of model size (there is, unfortunately, no Dutch-only Whisper model to compare our results to). What we can see from this comparison is that our model performs very badly. This is mainly due to the fact that it was only trained on two datasets while distil-medium was trained on eight.

Table 2:
Reproduction of (part of) Table 16 from the distil-whisper paper; per-dataset WER of the short-form test sets.

Dataset	**distil-medium.en**	**distil-whisper-nl (our model)**
Common Voice 13 (monsoonery/common_voice_13_0_nl_TEST_pseudo_labelled)	15.3	32.4
Voxpopuli	8.8	55.1

(Quirina/voxpopuli_nl_TEST_pseudo_labelled)		
FLEURS (google/fleurs)	6.9	67.0

Conclusion

This blog post aimed to present our methods, issues, and results by reproducing table 16 from the paper on distil-whisper. During our attempt to work with this model's given training code, we ran into several challenges primarily caused by its high computing and storage requirements. These factors caused a deviation from our initial goals of reproducing the paper's results by training and evaluating a) an English model and b) a Dutch model. Instead, most time was spent on devising strategies on how to run the training code with limited resources.

We tried several options with varying success. Firstly, a local setup, which would be perfectly feasible if a machine was available with high-compute GPUs, such as the Tesla T4 or A100. Whichever GPU is picked, must have at least 16GB RAM to attempt a reproduction similar to ours. At least 300GB of disk space is also highly recommended; more if you wish to train on all datasets like the authors. Secondly, we tried online platforms such as Kaggle and Google Compute Engine (with and without Google Colab). While these platforms offer access to much-needed GPUs and storage options suited to train this model, they also come with difficulties: setting up the platforms is cumbersome and GCE is not free. We only recommend these methods if you have the resources to do so.

By focusing on an approach using Google Colab connected to a custom GCE VM, we were able to eventually distill a Dutch model and at least run the full training sequence once. The results were, however, unfavorable. The model has worse performance than described in the paper. While this can be credited to the many changes we had to make to obtain a minimum working solution, it does mean that we were unable to reproduce the results from the paper, despite our ambitions to train an English model. Thus we cannot make any remarks about the results presented by the authors: resource restrictions prevented us from doing so.

References

Ardila, R., Branson, M., Davis, K., Kohler, M., Meyer, J., Henretty, M., . . . Weber, G. (2020). Common Voice: a Massively-Multilingual Speech Corpus. arXiv. https://doi.org/10.48550/arXiv.1912.06670

Gandhi, S., Patrick, V. P., & Rush, A. M. (2023). Distil-Whisper: Robust knowledge distillation via Large-Scale Pseudo Labelling. arXiv. https://doi.org/10.48550/arXiv.2311.00430

Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., & Sutskever, I. (2022). Robust speech recognition via Large-Scale Weak Supervision. arXiv. https://doi.org/10.48550/arXiv.2212.04356

Wang, C., Rivière, M., Lee, A., Wu, A., Talnikar, C., Haziza, D., . . . Dupoux, E. (2021). VoxPopuli: A Large-Scale Multilingual Speech Corpus for Representation Learning, Semi-Supervised Learning and Interpretation. Proceedings of the 59th Annual Meeting of the Association for