
whisper_streaming

Whisper realtime streaming for long speech-to-text transcription and translation

Turning Whisper into Real-Time Transcription System

Demonstration paper, by Dominik Macháček, Raj Dabre, Ondřej Bojar, 2023

Abstract: Whisper is one of the recent state-of-the-art multilingual speech recognition and translation models, however, it is not designed for real-time transcription. In this paper, we build on top of Whisper and create Whisper-Streaming, an implementation of real-time speech transcription and translation of Whisper-like models. Whisper-Streaming uses local agreement policy with self-adaptive latency to enable streaming transcription. We show that Whisper-Streaming achieves high quality and 3.3 seconds latency on unsegmented long-form speech transcription test set, and we demonstrate its robustness and practical usability as a component in live transcription service at a multilingual conference.

Paper PDF, Demo video

Slides – 15 minutes oral presentation at IJCNLP-AACL 2023

Please, cite us. ACL Anthology, Bibtex citation:

```
1 @inproceedings{machacek-etal-2023-turning,
2   title = "Turning Whisper into Real-Time Transcription System",
3   author = "Macháček, Dominik and
4     Dabre, Raj and
5     Bojar, Ondřej",
6   editor = "Saha, Sriparna and
7     Sujaini, Herry",
8   booktitle = "Proceedings of the 13th International Joint Conference
9     on Natural Language Processing and the 3rd Conference of the
10     Asia-Pacific Chapter of the Association for Computational
11     Linguistics: System Demonstrations",
12   month = nov,
13   year = "2023",
14   address = "Bali, Indonesia",
15   publisher = "Association for Computational Linguistics",
16   url = "https://aclanthology.org/2023.ijcnlp-demo.3",
17   pages = "17--24",
18 }
```

Installation

- 1) `pip install librosa soundfile` – audio processing library
- 2) Whisper backend.

Several alternative backends are integrated. The most recommended one is faster-whisper with GPU support. Follow their instructions for NVIDIA libraries – we succeeded with CUDNN 8.5.0 and CUDA 11.7. Install with `pip install faster-whisper`.

Alternative, less restrictive, but slower backend is whisper-timestamped: `pip install git+https://github.com/linto-ai/whisper-timestamped`

Thirdly, it's also possible to run this software from the OpenAI Whisper API. This solution is fast and requires no GPU, just a small VM will suffice, but you will need to pay OpenAI for api access. Also note that, since each audio fragment is processed multiple times, the price will be higher than obvious from the pricing page, so keep an eye on costs while using. Setting a higher chunk-size will reduce costs significantly. Install with: `pip install openai`

For running with the openai-api backend, make sure that your OpenAI api key is set in the `OPENAI_API_KEY` environment variable. For example, before running, do: `export OPENAI_API_KEY=sk-xxx` with `sk-xxx` replaced with your api key.

The backend is loaded only when chosen. The unused one does not have to be installed.

3) Optional, not recommended: sentence segmenter (aka sentence tokenizer)

Two buffer trimming options are integrated and evaluated. They have impact on the quality and latency. The default “segment” option performs better according to our tests and does not require any sentence segmentation installed.

The other option, “sentence” – trimming at the end of confirmed sentences, requires sentence segmenter installed. It splits punctuated text to sentences by full stops, avoiding the dots that are not full stops. The segmenters are language specific. The unused one does not have to be installed. We integrate the following segmenters, but suggestions for better alternatives are welcome.

- `pip install opus-fast-mosestokenizer` for the languages with codes `as bn ca cs de el en es et fi fr ga gu hi hu is it kn lt lv ml mni mr nl or pa pl pt ro ru sk sl sv ta te yue zh`
- `pip install tokenize_uk` for Ukrainian – `uk`
- for other languages, we integrate a good performing multi-lingual model of `wtpsplit`. It requires `pip install torch wtpsplit`, and its neural model `wtp-canine-s-12l-no-adapters`. It is downloaded to the default huggingface cache during the first use.
- we did not find a segmenter for languages `as ba bo br bs fo haw hr ht jw lb ln lo mi nn oc sa sd sn so su sw tk tl tt` that are supported by Whisper and not by `wtpsplit`. The default fallback option for them is `wtpsplit` with unspecified language. Alternative suggestions welcome.

In case of installation issues of opus-fast-mosestokenizer, especially on Windows and Mac, we recommend using only the “segment” option that does not require it.

Usage

Real-time simulation from audio file

```
1 usage: whisper_online.py [-h] [--min-chunk-size MIN_CHUNK_SIZE] [--
  model {tiny.en,tiny,base.en,base,small.en,small,medium.en,medium,
  large-v1,large-v2,large-v3,large}] [--model_cache_dir
  MODEL_CACHE_DIR] [--model_dir MODEL_DIR] [--lan LAN] [--task {
  transcribe,translate}]
2                               [--backend {faster-whisper,whisper_timestamped
  ,openai-api}] [--vad] [--buffer_trimming {
  sentence,segment}] [--buffer_trimming_sec
  BUFFER_TRIMMING_SEC] [--start_at START_AT]
3                               [--offline] [--comp_unaware]
4                               audio_path
5 positional arguments:
6   audio_path            Filename of 16kHz mono channel wav, on which
  live streaming is simulated.
7
8 options:
9   -h, --help            show this help message and exit
10  --min-chunk-size MIN_CHUNK_SIZE
11                        Minimum audio chunk size in seconds. It waits
  up to this time to do processing. If the
  processing takes shorter time, it waits,
  otherwise it processes the whole segment
  that was received by this time.
12  --model {tiny.en,tiny,base.en,base,small.en,small,medium.en,medium,
  large-v1,large-v2,large-v3,large}
13                        Name size of the Whisper model to use (default:
  large-v2). The model is automatically
  downloaded from the model hub if not present
  in model cache dir.
14  --model_cache_dir MODEL_CACHE_DIR
15                        Overriding the default model cache dir where
  models downloaded from the hub are saved
16  --model_dir MODEL_DIR
17                        Dir where Whisper model.bin and other files are
  saved. This option overrides --model and --
  model_cache_dir parameter.
18  --lan LAN, --language LAN
19                        Source language code, e.g. en,de,cs, or 'auto'
  for language detection.
20  --task {transcribe,translate}
```

```

21                                     Transcribe or translate.
22  --backend {faster-whisper,whisper_timestamped,openai-api}
23                                     Load only this backend for Whisper processing.
24  --vad                               Use VAD = voice activity detection, with the
    default parameters.
25  --buffer_trimming {sentence,segment}
26                                     Buffer trimming strategy -- trim completed
    sentences marked with punctuation mark and
    detected by sentence segmenter, or the
    completed segments returned by Whisper.
    Sentence segmenter must be installed for "
    sentence" option.
27  --buffer_trimming_sec BUFFER_TRIMMING_SEC
28                                     Buffer trimming length threshold in seconds. If
    buffer length is longer, trimming sentence/
    segment is triggered.
29  --start_at START_AT               Start processing audio at this time.
30  --offline                         Offline mode.
31  --comp_unaware                     Computationally unaware simulation.

```

Example:

It simulates realtime processing from a pre-recorded mono 16k wav file.

```

1  python3 whisper_online.py en-demo16.wav --language en --min-chunk-size
    1 > out.txt

```

Simulation modes:

- default mode, no special option: real-time simulation from file, computationally aware. The chunk size is `MIN_CHUNK_SIZE` or larger, if more audio arrived during last update computation.
- `--comp_unaware` option: computationally unaware simulation. It means that the timer that counts the emission times “stops” when the model is computing. The chunk size is always `MIN_CHUNK_SIZE`. The latency is caused only by the model being unable to confirm the output, e.g. because of language ambiguity etc., and not because of slow hardware or suboptimal implementation. We implement this feature for finding the lower bound for latency.
- `--start_at START_AT`: Start processing audio at this time. The first update receives the whole audio by `START_AT`. It is useful for debugging, e.g. when we observe a bug in a specific time in audio file, and want to reproduce it quickly, without long waiting.
- `--offline` option: It processes the whole audio file at once, in offline mode. We implement it to find out the lowest possible WER on given audio file.

Output format

```
1 2691.4399 300 1380 Chairman, thank you.
2 6914.5501 1940 4940 If the debate today had a
3 9019.0277 5160 7160 the subject the situation in
4 10065.1274 7180 7480 Gaza
5 11058.3558 7480 9460 Strip, I might
6 12224.3731 9460 9760 have
7 13555.1929 9760 11060 joined Mrs.
8 14928.5479 11140 12240 De Kaiser and all the
9 16588.0787 12240 12560 other
10 18324.9285 12560 14420 colleagues across the
```

See description [here](#)

As a module

TL;DR: use `OnlineASRProcessor` object and its methods `insert_audio_chunk` and `process_iter`.

The code `whisper_online.py` is nicely commented, read it as the full documentation.

This pseudocode describes the interface that we suggest for your implementation. You can implement any features that you need for your application.

```
1 from whisper_online import *
2
3 src_lang = "en" # source language
4 tgt_lang = "en" # target language -- same as source for ASR, "en" if
   translate task is used
5
6 asr = FasterWhisperASR(lang, "large-v2") # loads and wraps Whisper
   model
7 # set options:
8 # asr.set_translate_task() # it will translate from lang into English
9 # asr.use_vad() # set using VAD
10
11 online = OnlineASRProcessor(asr) # create processing object with
   default buffer trimming option
12
13 while audio_has_not_ended: # processing loop:
14     a = # receive new audio chunk (and e.g. wait for min_chunk_size
        seconds first, ...)
15     online.insert_audio_chunk(a)
16     o = online.process_iter()
17     print(o) # do something with current partial output
18 # at the end of this audio processing
19 o = online.finish()
20 print(o) # do something with the last output
21
22
```

```
23 online.init() # refresh if you're going to re-use the object for the
    next audio
```

Server – real-time from mic

`whisper_online_server.py` has the same model options as `whisper_online.py`, plus `--host` and `--port` of the TCP connection and the `--warmup-file`. See the help message (`-h` option).

Client example:

```
1 arecord -f S16_LE -c1 -r 16000 -t raw -D default | nc localhost 43001
```

- `arecord` sends realtime audio from a sound device (e.g. mic), in raw audio format – 16000 sampling rate, mono channel, S16_LE – signed 16-bit integer low endian. (use the alternative to `arecord` that works for you)
- `nc` is netcat with server’s host and port

Background

Default Whisper is intended for audio chunks of at most 30 seconds that contain one full sentence. Longer audio files must be split to shorter chunks and merged with “init prompt”. In low latency simultaneous streaming mode, the simple and naive chunking fixed-sized windows does not work well, it can split a word in the middle. It is also necessary to know when the transcript is stable, should be confirmed (“committed”) and followed up, and when the future content makes the transcript clearer.

For that, there is LocalAgreement-n policy: if n consecutive updates, each with a newly available audio stream chunk, agree on a prefix transcript, it is confirmed. (Reference: CUNI-KIT at IWSLT 2022 etc.)

In this project, we re-use the idea of Peter Polák from this demo: https://github.com/petrik/transformers/blob/online_decode/examples/pytorch/online-decoding/whisper-online-demo.py However, it doesn’t do any sentence segmentation, but Whisper produces punctuation and the libraries `faster-whisper` and `whisper_transcribed` make word-level timestamps. In short: we consecutively process new audio chunks, emit the transcripts that are confirmed by 2 iterations, and scroll the audio processing buffer on a timestamp of a confirmed complete sentence. The processing audio buffer is not too long and the processing is fast.

In more detail: we use the init prompt, we handle the inaccurate timestamps, we re-process confirmed sentence prefixes and skip them, making sure they don’t overlap, and we limit the processing buffer window.

Performance evaluation

See the paper.

Contributions

Contributions are welcome. We acknowledge especially:

- The GitHub contributors for their pull requests with new features and bugfixes.
- The translation of this repo into Chinese.
- Ondřej Plátek for the paper pre-review.
- Peter Polák for the original idea.
- The UEDIN team of the ELITR project for the original `line_packet.py`.

Contact

Dominik Macháček, machacek@ufal.mff.cuni.cz