

Error Rates Methodology

I have used Fleurs source dataset [from hugging face](#). I have used “test” subset, 12 languages with the following codes:

ar_eg de_de en_us es_419 fr_fr hi_in it_it nl_nl pt_br cmn_hans_cn ja_jp ko_kr

My QA data pipeline has the following four steps.

1. Audio preprocessing: convert audio archives into ZIP with 16-bit FLAC audio.
2. Transcribe speech to text, inserting the text into [SQLite](#) database.
3. Compute per-clip word error rates. This step adds another table into the same database file, same primary key as the first table.
4. Aggregate error rates across clips, produce a single number per dataset.

I am writing this document because the pipeline is quite complicated, many steps are introducing deviations both random and systematic, however I'd like my results to be somewhat reproducible.

All benchmarks were completed on nVidia 4070 Ti Super GPU, albeit I hope at least the GPU model shouldn't affect output text or the error metrics.

Audio Preprocessing

The source dataset contains audio in [wave format](#). Not just a regular wave format, google used [fp32](#) format for the PCM samples with widely different levels across clips.

For this reason, I have made a tool to convert audio data from *.tar.gz archives with *.wav files into *.zip archives with *.flac files, normalizing levels per clip. For some of the audio clips the levels in the Fleurs dataset were so low they required +70 dB amplification to scale into the full range of [-1.0 .. +1.0]

The output FLAC files are 16 bit/sample at 16 kHz. Despite FLAC is a lossless audio codec and the source audio files are using the same 16 kHz sample rate, the whole conversion process is lossy because scaling floats and rounding them to integers.

Transcribe Speech

To transcribe the dataset with [reference Voxtral](#) implementation, I have used transformers 5.6.2, mistral_common 1.11 and PyTorch 2.5.1 with CUDA backend.

To transcribe the dataset with my own implementation, I have used an internal tool which generates SQLite files in the same format. Note when using the file transcribe app I ship, the delay parameter is hardcoded to the maximum supported value of 2400 milliseconds.

Clip Error Rates

To compute per-clip error rates, I have used [whisper-normalizer 0.1.12](#) and [jiwer](#) 4.0.0 Python packages.

I took reference text from the 4-th column of these `test.tsv` tables. The [csv package](#) required custom setup to correctly parse these tables, note the last argument:
`reader = csv.reader( f, delimiter='\t', quoting=csv.QUOTE_NONE )`

Aggregate Rates

To compute per-dataset error rates, I have added 4 integer columns from the `WordErrorRate` table across all audio clips, then applied the following formula:

```
int mul = substitutions + deletions + insertions;  
int div = correct + substitutions + deletions;  
return mul / (double)div;
```

Source Codes

Audio Normalisation

Here's the C# function which transforms PCM data for the FLAC encoder. The function was written for .NET 10. In runtime, it requires an AMD64 processor which supports AVX2 instruction set extension.

<https://gist.github.com/Vrmac-softver/8664c35b494f865eeca3b7ab836761fb>

Reference Voxtral Transcribe

The constructor of the `AudioArchive` Python class sorts clips by uncompressed size in descending order. This way elements of each batch have roughly the same duration, and less time is wasted on partially complete batches.

The constant 7 for the batch size is the largest value which fits in 16 GB VRAM of my GPU for the short clips found in Fleurs dataset¹.

<https://gist.github.com/Vrmac-softver/db92fa3b40bed16a6a1ad33954a88817>

Per-Clip Error Rates

Python function which computes the error metrics for each clip:

<https://gist.github.com/Vrmac-softver/114ddea0767673df15b76b125abf4697>

¹ Reference Python version allocates progressively more VRAM during transcribe because language model KV caches. The clips which exceed 10m:50s are using 832 MB VRAM for the language model KV cache, short clips are using much less. My implementation allocates the memory in advance for reliability and performance.

Arabic Language Normaliser

The following function has been written by the NLP team at Elm Company's Applied Research division:

<https://gist.github.com/Vrmac-softver/a193ada14bf7b4e45bd8c511c68f29c9>

Japanese and Chinese Normalisers

<https://gist.github.com/Vrmac-softver/3512d8abb90deb7430ddac14c5fddaae>

Korean Space Normaliser

While using the QA tool I have built, I noticed the error tables for the language are dominated by space-only differences. Seems in the Korean language, the spaces between some words are optional. To fix the glitch, I added a function specifically for the Korean language which inserts or removes spaces based on the reference text, before supplying both strings to the generic language-agnostic normaliser and computing the error rates.

<https://gist.github.com/Vrmac-softver/31a971fdb76c00d2a6ef5b6b5a57e175>