Speech Recognition Evaluation

Dan DeGenaro
Georgetown University
drd92@georgetown.edu

Abstract

In this assignment, we introduce a speech model evaluation benchmark to the interface and systematically evaluate a collection of Whisper variants across three small test sets.

1 Introduction

1.1 Why evaluate Whisper variants?

The Whisper family of speech recognition models (Radford et al., 2023) offer significant performance-efficiency tradeoffs. In general, a larger model has a greater "capacity," and can perform better on whatever task on which it's trained, especially if the training data is scaled alongside the model size (Kaplan et al., 2020; Radford et al., 2023). However, as we will see, smaller models run much faster, thereby yielding the aforementioned tradeoff.

There are reasons to prioritize both performance and speed. An ideal model is both performant and fast, but in practice, it is hard to make a sharp gain in one dimension without worsening in the other. This can be illustrated with a Pareto scatter plot, in which the two desirable quantities are placed on orthogonal axes, as in Figure 1. An ideal model would score infinitely close to the origin in this plot.

A documentary linguist may be willing to let a speech recognition model run for quite a long time to get the highest quality transcript possible, thereby minimizing further manual correction after the fact, thus preferring models near the top-left corner of the plot (maximum performance). Conversely, a software engineer prototyping an application involving speech recognition may be satisfied with a fast-to-load, fast-to-run model regardless of its low performance for testing purposes, thereby preferring the bottom-right corner of the plot (maximum efficiency). Finally, later on in such development, the software engineer is likely

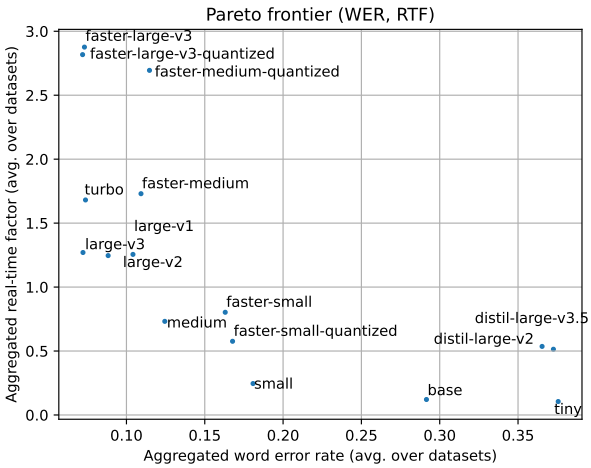

Figure 1: A Pareto scatter plot of aggregate performance and efficiency.

to move to the bottom-left corner of the plot, choosing a model that will run relatively quickly on an end-user's device in real-time, while still giving relatively accurate transcriptions (balance).

Throughout this paper, we operationalize "performance" as word error rate (WER) over a test set, and we operationalize "speed" or "efficiency" as real-time factor (RTF).

1.2 Research questions

We investigate the following questions:

- In a real-world deployment setting (running a local model, such as a linguist might want to do), which models yield optimal performance (operationalized as word error rate (WER)), efficiency (operationalized as real-time factor (RTF)), and balance?
- How do such metrics change with respect to data domain?
- How do such metrics change with respect to data language?

1.3 Overview of Whisper architectures

In the original Whisper paper (Radford et al., 2023), the authors produce a variety of model sizes trained on the same data, yielding a broad selection of models with different performance-efficiency tradeoffs. The authors natively implement these models in PyTorch, with 32-bit floating point numbers for weights. These are `tiny`, `base`, `small`, `medium`, and `large`.

The authors later released additional models, including `large-v2` and `large-v3`, which are the same architecture as `large`, retroactively re-named `large-v1`, but with additional training. `large-v2` was trained with the same data as `large-v1`, but for 2.5 times as many epochs through that data, along with some regularization to correspondingly reduce overfitting. `large-v3` further improves on `large-v2` by using 128 mel bins instead of 80 as input features, and additionally adds support for Cantonese.

They also later released `turbo`, which was produced by pruning and fine-tuning `large-v3`. The pruning process involved eliminating 28 out of `large-v3`'s 32 decoder layers, yielding a similarly-performing model with far fewer parameters.

Follow-up work introduced `faster-whisper` implementations (SYSTRAN, 2025), which are architecturally identical to their native PyTorch counterparts, but with all data structures and operations on them implemented in C++ using the CTranslate2 library (OpenNMT, 2025).

Still further follow-up work introduced the `distil-whisper` family of models (Gandhi et al., 2023). Distilled variants are designed as scaled-down versions of the original Whisper architectures and are trained via a distillation loss, with corresponding native Whisper models providing a reference distribution over tokens from which the distilled variants can learn. The distillation loss is defined to be the KL-divergence $D_{KL}(p||q)$ as in Equation 1.

$$D_{KL}(p||q) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \qquad (1)$$

where p is the "true" distribution produced by the native Whisper model and q is the "approximation" produced by the distilled variant.

Another work of note is WhisperX (Bain et al., 2023), which first applies Voice Activity Detection (VAD) to an audio input before transcribing via Whisper using parallelized batch processing, yielding a speed-up and a more accurate time-alignment.

1.4 Best tradeoffs

In the following work, we select `small`, `medium`, and `large-v3` as offering the best tradeoffs across all native implementations of Whisper architectures. We find that `small` is the most efficient model (lowest RTF) offering reasonable performance (operationalized as a WER < 0.5 on each test set individually), that `large-v3` has the highest overall performance (lowest WER), and that `medium` offers a balance between performance and efficiency (roughly between the other two on the Pareto scatter plot for each test set).

2 Methods

2.1 Hardware specifications

All experiments with local models are conducted on a 16 GB Apple M4 chip, whose memory is shared by the CPU and MPS device, with all computation handled by the MPS system when available, and otherwise on the CPU. Notably, only `distil-whisper` (Gandhi et al., 2023) variants are implemented only with operations available to the MPS architecture, so native Whisper implementations and all `faster-whisper` (SYSTRAN, 2025) implementations are run on the CPU. Unless suffixed by `-quantized`, all models evaluated are loaded in full precision (`float32`). `-quantized` models are loaded in half precision – specifically, `bfloat16` (Intel, 2018).

2.2 Model selection

2.2.1 Native PyTorch Whisper variants

We evaluate the architectures listed in Table 1 (usages also visualized in Figure 2). Native implementations (full-precision, implemented in PyTorch (Paszke et al., 2019)), are without any special prefix or suffix. Note that version suffixes such as `-v2` are not considered special suffixes for this purpose.

The prefix `faster-` indicates that the CTranslate2 (OpenNMT, 2025) implementation known as `faster-whisper` (SYSTRAN, 2025) for that model was used, rather than the native PyTorch.

The `distil-` prefix indicates that a distilled form of the model was used, rather than the original model (Gandhi et al., 2023). Notably, the distilled models are only distilled over English training data, and as such, tend to perform very poorly

Architecture	CPU Usage (GB)
tiny	0.14
base	0.27
faster-small-quantized	0.47
faster-small	0.66
faster-medium-quantized	0.81
small	0.90
faster-large-v3-quantized	1.11
distil-large-v2	2.82
distil-large-v3.5	2.82
medium	2.84
turbo	3.01
faster-medium	3.66
large-v1	5.74
large-v2	5.74
large-v3	5.74
faster-large-v3	5.80

Table 1: Whisper variants tested and their respective memory usage on an M4 chip's CPU.

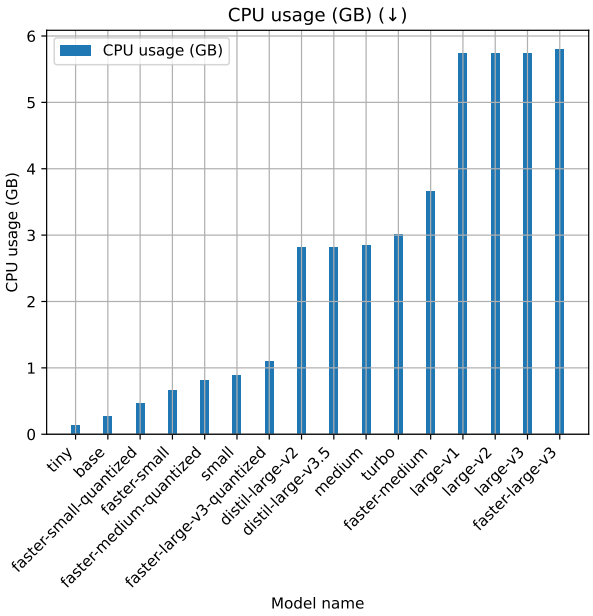

Figure 2: A visualization of the same information as in Table 1.

on languages other than English.

The suffix `-quantized` indicates that a quantized version of the corresponding model was used, rather than the original precision (`float32`). In all cases, the quantization employed is `bfloat16` (Intel, 2018).

2.2.2 Rationale for best tradeoffs

Our rationale for selecting these variants is as follows. We first select every Whisper architecture offered by OpenAI's Whisper library, which are all the variants without special prefixes or suffixes. This yields the full range of architecture sizes. We then select three top-performing models from these, ranked by performance, efficiency above a performance cutoff, and a balance between performance and efficiency. The top-performing model (lowest WER) was `large-v3`. The most efficient model having a WER below 0.5 on every test set was `small`. We therefore choose the "balanced" model to be `medium`, which lies roughly between `small` and `large-v3` on the Pareto scatter plot for each test set. These relationships can be seen clearly in Figures 3, 4, and 5.

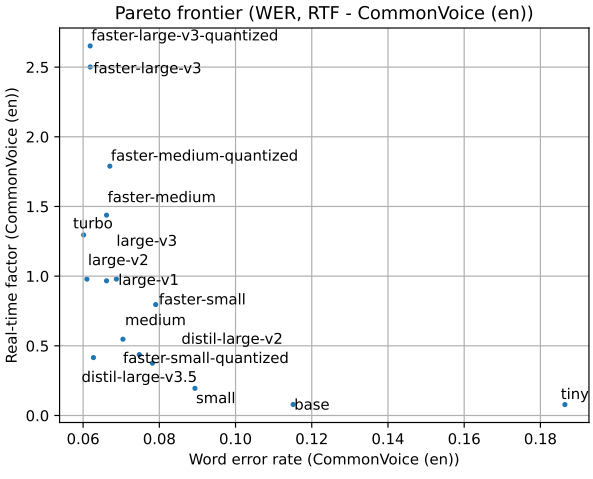

Figure 3: A Pareto scatter plot of aggregate performance and efficiency on a small subset of the English CommonVoice dataset.

2.3 Datasets

We evaluate every Whisper implementation we select on three different test sets:

1. Subset of English CommonVoice, delta segment 19.0 (2024-09-13) (Ardila et al., 2020)

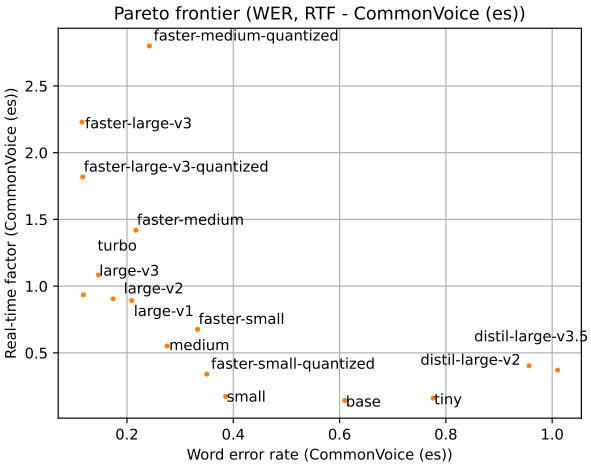

Figure 4: A Pareto scatter plot of aggregate performance and efficiency on a small subset of the Spanish CommonVoice dataset.

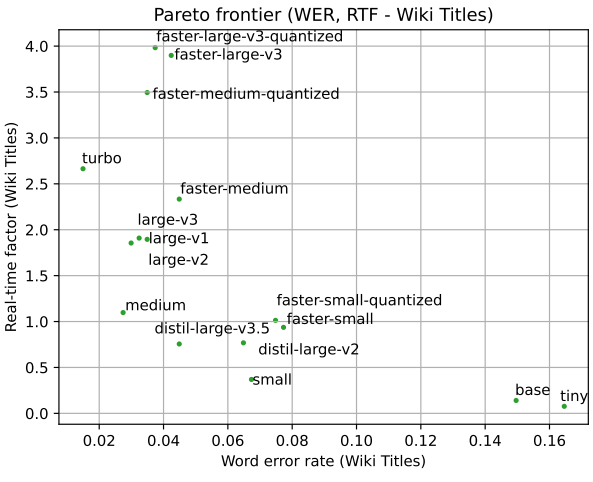

Figure 5: A Pareto scatter plot of aggregate performance and efficiency over the Wiki Titles dataset.

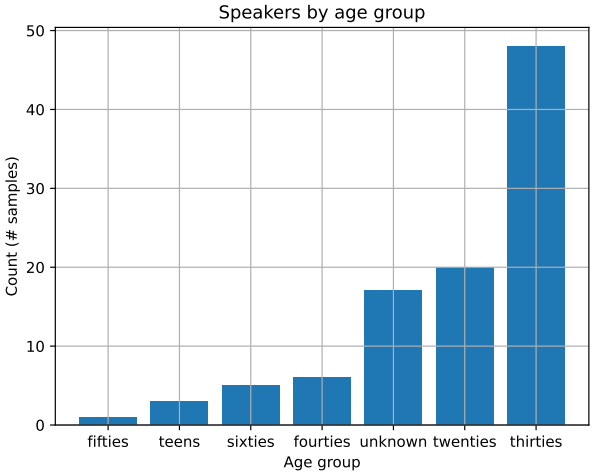

Figure 6: English CommonVoice delta segment 19.0, broken down by reported age group.

2. Subset of Spanish CommonVoice, delta segment 19.0 (2024-09-13) (Ardila et al., 2020)

3. Wiki Titles (original work)

For English CommonVoice, we select the 100 longest validated sentences (by character count) in delta segment 19.0, with this delta segment being one of the smallest (and therefore most convenient) deltas recently released by Mozilla.

For Spanish CommonVoice, not enough of the sentences are validated, so we select the 100 longest sentences (also by character count) categorized as "other" (not validated or invalidated), also from delta segment 19.0. We point to low WER for larger Whisper variants on this dataset as evidence of its validity.

All CommonVoice files are mono-channel `.mp3` files with 32 kHz sampling. Breakdowns by age, gender, and accent groups for both English and Spanish are available in Figures 6, 7, 8, 9, 10, and 11. CommonVoice recordings are highly variable in their recording conditions and audio quality due to the crowd-sourcing approach used to build CommonVoice. In general, there are few guarantees about either conditions or quality, which both largely depend on the given participant's choices and access.

Finally, the Wiki Titles dataset consists of the first author reading aloud a selection of 100 random English Wikipedia article titles, with the given title serving as the gold transcript for each corresponding audio sample. All recordings were made using the built-in microphone of a 14-inch Macbook Pro using the Voice Memos application

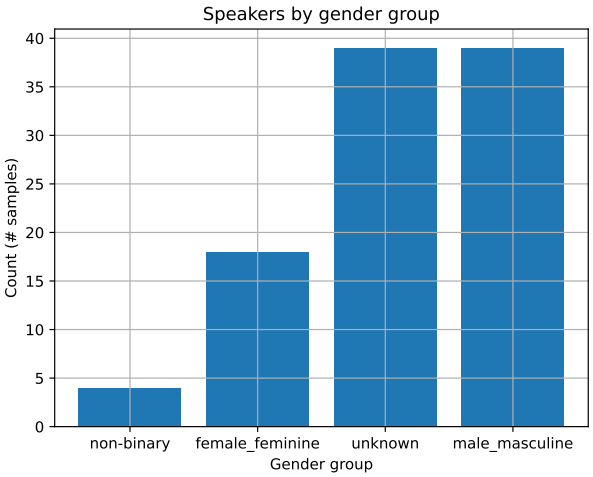

Figure 7: English CommonVoice delta segment 19.0, broken down by reported gender group.

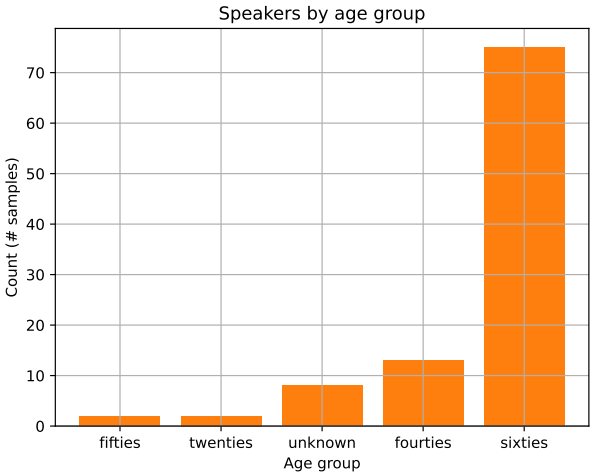

Figure 9: Spanish CommonVoice delta segment 19.0, broken down by reported age group.

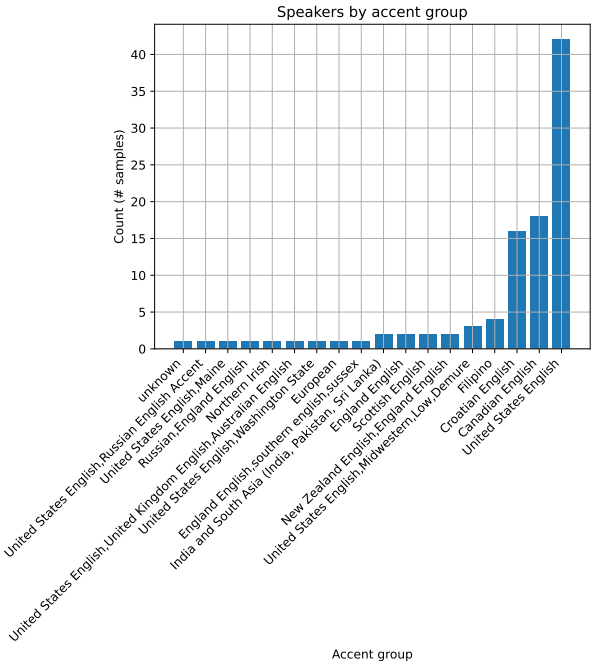

Figure 8: English CommonVoice delta segment 19.0, broken down by reported accent group.

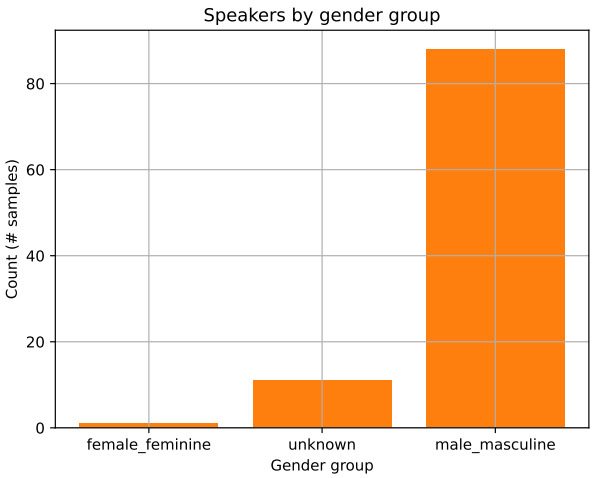

Figure 10: Spanish CommonVoice delta segment 19.0, broken down by reported gender group.

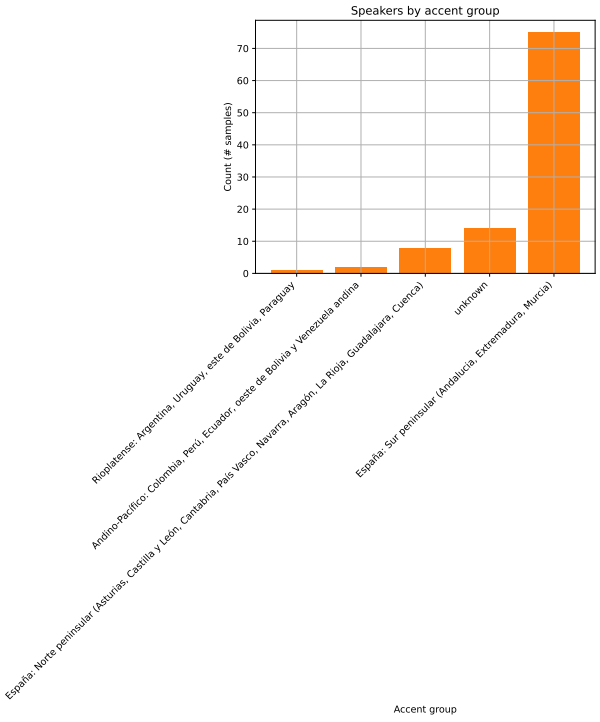

Figure 11: Spanish CommonVoice delta segment 19.0, broken down by reported accent group.

and saved as mono-channel `.m4a` files, with 48 kHz sampling. The recording environment had a quiet, but consistent, buzzing in the background from HVAC and lighting systems. Mouse clicks are often audible at the end of each recording.

Basic statistics about the lengths of transcriptions (in characters) and corresponding audio samples (in seconds) can be found in Table 2.

2.4 Evaluation methodology

We measure word error rate (WER) using the `jiwer` library (Jitsi, 2025). We pre-process all references (gold transcriptions/Wikipedia titles) and hypotheses (Whisper outputs) by removing all non-alpha-numeric characters (Spanish characters having diacritics are considered alpha-numeric) and replacing all multiple whitespaces with a single space character, as well as lowercasing. We make this choice to avoid penalizing the model for punctuating or casing incorrectly, given that punctuation and casing choices are heavily modified by the Whisper post-processing pipeline, which will greatly "hinder" performance when evaluating on Wikipedia titles, which are by nature punctuated and cased very differently from most audio transcriptions. WER is calculated independently for each dataset, over the entire given dataset in each case. An aggregate WER is calculated by averaging the three datasets' overall WERs, yielding a WER that treats each overall performance on a given dataset as equal to all others.

We measure real-time factor (RTF) by measuring the latency or total processing time of an audio file (or a collection of them) – including time needed to pre-process, process, and post-process – as compared to its (their collective) duration as in Equation 2. We treat each dataset as a collection of audio files for this purpose, and compute an overall RTF for each dataset, as well as an aggregate RTF calculated similarly to the aggregate WER.

$$\text{RTF} = \frac{\text{total processing time (s)}}{\text{audio duration (s)}} \tag{2}$$

The aggregate measures are intended to account for variance between datasets such as is seen in Table 2, ensuring the aggregate measures are not biased towards one dataset or another, which would cause them to reflect performance on one dataset more than another.

3 Results

3.1 Native model comparison

A comparison of native implementations by WER is given in Table 3. It is clear that `turbo` and the `large-v{1,2,3}` variants are the top performers, though `medium` and even `small` are quite good on English data.

A similar table for RTF is given in Table 4. It is clear that smaller models such as `tiny` and `base` are faster, though `small`, `medium`, and even the `large-v{1,2,3}` models still sometimes have RTF < 1.0, indicating a speedup over real time. Surprisingly, despite using identical hardware and an identical loading process, `turbo` was the slowest model.

Finally, a comparison of loading times and CPU usages is given in Table 5. Predictably, smaller models by parameter count load faster and use less memory.

From this table, we identify three models from the native implementation family to study further in optimized implementations. Namely, we select `large-v3` as it is (one of) the most performant models, `small` as it is the fastest model having a WER < 0.5 on every dataset, and `medium` as a tradeoff roughly in the middle between large-v3 and `small` in terms of both performance and efficiency.

	CommonVoice (en)	CommonVoice (es)	Wiki Titles
min chars	50	98	5
mean chars	71.56	103.56	26.51
max chars	113	129	58
min audio length (s)	3.17	5.44	1.66
mean audio length (s)	6.25	7.95	3.04
max audio length (s)	10.91	13.18	5.33

Table 2: Dataset statistics over transcription lengths (in characters) and audio lengths (in seconds). Each dataset has 100 samples.

Model name	CV-en	CV-es	Wiki Titles	Agg.
`tiny`	0.19	0.78	0.16	0.38
`base`	0.12	0.61	0.15	0.29
`small`	0.09	0.39	0.07	0.18
`medium`	0.07	0.28	0.03	0.12
`turbo`	**0.06**	0.15	**0.01**	**0.07**
`large-v1`	0.07	0.21	0.03	0.10
`large-v2`	**0.06**	0.17	0.03	0.09
`large-v3`	0.07	**0.12**	0.03	**0.07**

Table 3: WER by model and dataset. A WER of 0 means the model is perfect. **CV-en** is CommonVoice English and **CV-es** is CommonVoice Spanish. Best results on each dataset (and in aggregate) are **bold**, while second best results are underlined. Ties are included. **Aggregate** WER is an average of the WERs of the three datasets, and should not be seen as the overall WER across all data used in this evaluation.

Model name	CV-en	CV-es	Wiki Titles	Agg.	Overall
`tiny`	**0.08**	0.16	**0.08**	**0.11**	**0.12**
`base`	**0.08**	**0.14**	0.14	0.12	**0.12**
`small`	0.19	0.17	0.37	0.25	0.22
`medium`	0.55	0.55	1.10	0.73	0.65
`turbo`	1.30	1.08	2.66	1.68	1.44
`large-v1`	0.98	0.89	1.90	1.26	1.10
`large-v2`	0.98	0.90	1.86	1.25	1.10
`large-v3`	0.97	0.94	1.91	1.27	1.12

Table 4: RTF by model and dataset. An RTF of 0 means the model is infinitely fast. **CV-en** is CommonVoice English and **CV-es** is CommonVoice Spanish. Best results on each dataset (and in aggregate) are **bold**, while second best results are underlined. Ties are included. **Aggregate** RTF is an average of the RTFs of the three datasets, and should not be seen as the overall RTF across all data used in this evaluation, which is the **Overall** column.

Model name	Loading time (s)	CPU Usage (GB)
`tiny`	**1.06**	**0.14**
`base`	1.10	0.27
`small`	2.51	0.90
`medium`	10.54	2.84
`turbo`	58.98	3.01
`large-v1`	141.47	5.74
`large-v2`	131.33	5.74
`large-v3`	119.94	5.74

Table 5: Loading time in seconds and CPU Usage in GB by model.

3.2 Optimized implementations

We select various optimizations, including C++ implementations (`faster-` variants), and corresponding quantized variants (`faster-*-quantized`), all of them 8-bit integer quantizations. We also select two distilled variants (`distil-` variants). Comparisons are shown in Tables 6, 7, 8, and 9.

Notably, distilled variants fail on Spanish, achieving WER > 1.0, indicating complete inability to transcribe languages other than English. This is unsurprising, as these distilled variants were distilled only using English training data, and there is little to no reason for them to be able to accurately transcribe Spanish. In truth, these models are misnamed - they should all have the `.en` suffix used by the native English-only implementations in their naming convention.

In general, larger models are slower, but more accurate. However, overall, on an M4 chip, it is clear that a native PyTorch implementation should be selected, as C++ implementations cannot be run on the MPS hardware, making them extremely slow. In principle, C++ implementations are faster, and likely will be in the future when more operations are implemented by the MPS hardware that are needed by the CTranslate2 library in order to run on MPS.

`faster-` models use less memory and are faster to load. `faster-*-quantized` models use even less memory, but at the cost of loading a bit more slowly. Both variants achieve similar, if not better, WER as compared to their native implementations, but have much higher RTF due to being confined to the CPU. Only `distil-` models yield any benefit in terms of RTF due to being smaller, but implementable on MPS hardware. They are, of course, smaller and therefore faster to load as well as compared to native implementations, but at the cost of losing their multilingual capabilities.

Compared to SOTA, as of 10/27/2025, NVIDIA's `parakeet-tdt-0.6b-v3` (Sekoyan et al., 2025) achieves a WER of only 0.03 on the Spanish CommonVoice dataset, whereas the top-performing Whisper model, as evaluated here, achieves 0.12 (tie across `large-v3` variants). A CommonVoice English benchmark is not available, nor is the custom Wiki Titles dataset a benchmark.

We visualize key results in Figures 12, 13, 14, and 15.

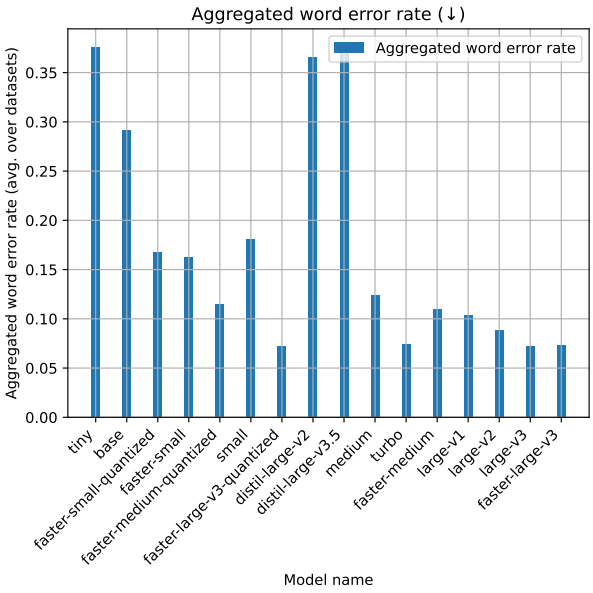

Figure 12: Aggregated WER by model.

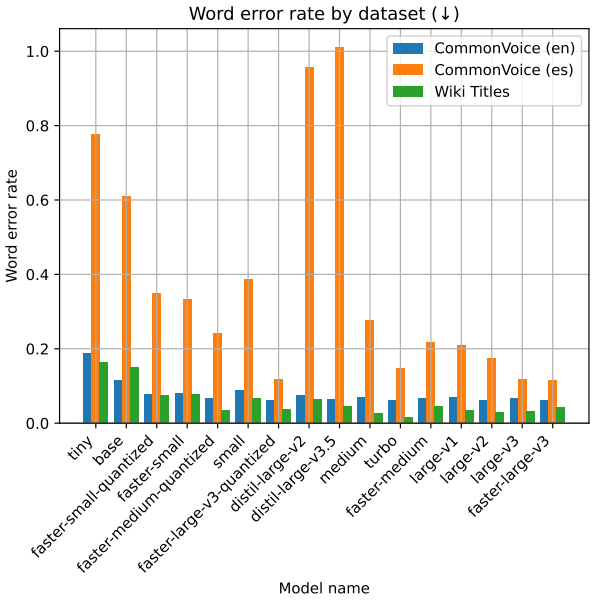

Figure 13: WER by model and dataset.

Model	faster-small-quantized	faster-small	small
WER (CV-en)	**0.08**	**0.08**	0.09
WER (CV-es)	0.35	**0.33**	0.39
WER (Wiki Titles)	**0.07**	0.08	**0.07**
Agg. WER	0.17	**0.16**	0.18
RTF (CV-en)	0.38	0.80	**0.19**
RTF (CV-es)	0.34	0.68	**0.17**
RTF (Wiki Titles)	1.01	0.94	**0.37**
Agg. RTF	0.58	0.80	**0.25**
Overall RTF	0.47	0.77	**0.22**
Loading time (s)	0.46	**0.42**	2.51
CPU Usage (GB)	**0.47**	0.66	0.90

Table 6: Comparison of various implementations of `small`. `faster-small` is the C++ implementation in full precision (FP32), whereas `faster-small-quantized` is in 8-bit precision (INT8). The best model (column) for each metric (row) is **bolded**, while the second-best is underlined.

Model	faster-medium-quantized	faster-medium	medium
WER (CV-en)	**0.07**	**0.07**	**0.07**
WER (CV-es)	0.24	**0.22**	0.28
WER (Wiki Titles)	**0.03**	0.04	**0.03**
Agg. WER	**0.11**	**0.11**	0.12
RTF (CV-en)	1.79	1.44	**0.55**
RTF (CV-es)	2.8	1.42	**0.55**
RTF (Wiki Titles)	3.49	2.33	**1.10**
Agg. RTF	2.69	1.73	**0.73**
Overall RTF	2.56	1.59	**0.65**
Loading time (s)	2.19	**1.84**	10.54
CPU Usage (GB)	**0.81**	3.66	2.84

Table 7: Comparison of various implementations of `medium`. `faster-medium` is the C++ implementation in full precision (FP32), whereas `faster-medium-quantized` is in 8-bit precision (INT8). The best model (column) for each metric (row) is **bolded**, while the second-best is underlined.

Model	large-v2	distil-large-v2
WER (CV-en)	**0.06**	0.07
WER (CV-es)	**0.17**	0.96
WER (Wiki Titles)	**0.03**	0.06
Agg. WER	**0.09**	0.37
RTF (CV-en)	0.98	**0.44**
RTF (CV-es)	0.90	**0.40**
RTF (Wiki Titles)	1.86	**0.77**
Agg. RTF	1.25	**0.54**
Overall RTF	1.10	**0.48**
Loading time (s)	131.33	**4.40**
CPU Usage (GB)	5.74	**2.82**

Table 8: Comparison of various implementations of `large-v2`. `distil-large-v2` is a distilled form of `large-v2` which was distilled only on English. The best model (column) for each metric (row) is **bolded**, while the second-best is underlined.

Model	`faster-large-v3-quantized`	`faster-large-v3`	`large-v3`	`distil-large-v3.5`
WER (CV-en)	**0.06**	**0.06**	0.07	**0.06**
WER (CV-es)	**0.12**	**0.12**	**0.12**	1.01
WER (Wiki Titles)	0.04	0.04	**0.03**	0.04
Agg. WER	**0.07**	**0.07**	**0.07**	0.37
RTF (CV-en)	2.65	2.5	0.97	**0.42**
RTF (CV-es)	1.82	2.23	0.94	**0.37**
RTF (Wiki Titles)	3.98	3.9	1.91	**0.76**
Agg. RTF	2.82	2.88	1.27	**0.51**
Overall RTF	2.5	2.62	1.27	**0.51**
Loading time (s)	**5.46**	7.56	119.94	6.15
CPU Usage (GB)	**1.11**	5.8	5.74	2.82

Table 9: Comparison of various implementations of `large-v3(.5)`. `faster-large-v3` is the C++ implementation in full precision (FP32), whereas `faster-large-v3-quantized` is in 8-bit precision (INT8). `distil-large-v3.5` is a distilled form of `large-v3.5` which was distilled only on English. The best model (column) for each metric (row) is **bolded**, while the second-best is underlined.

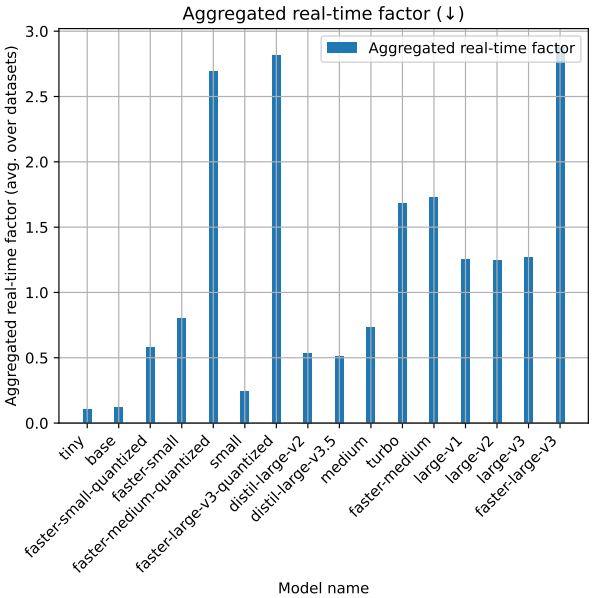

Figure 14: Aggregated RTF by model.

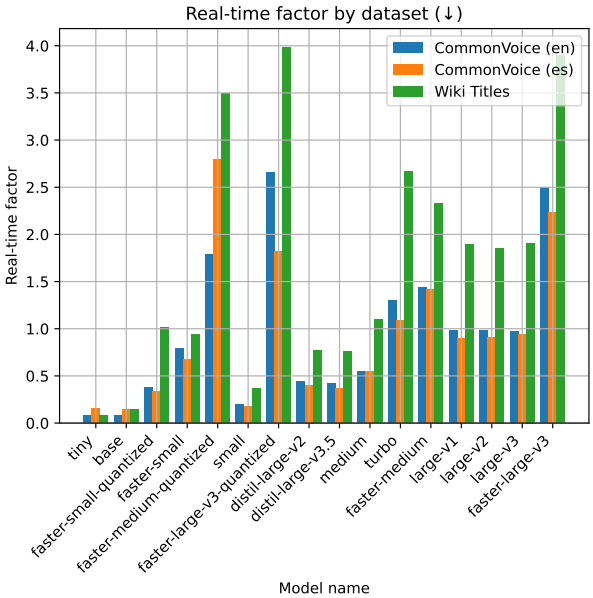

Figure 15: RTF by model and dataset.

4 Discussion

4.1 Best models

4.1.1 Fastest with acceptable WER

We consider `small` to be the fastest with acceptable accuracy, which we define as a WER < 0.5 on all three datasets, independently. It is about five times as fast as `large-v3` going by RTF, and only a few percentage points behind in WER on English. It is significantly worse than `large-v3` on Spanish, though not unusable, especially in the case of feeding speech to a competent LLM which could accurately determine the intended meaning. This would be a good model for prototyping an application where an LLM will parse the transcription and correct it or use it in a non-mission-critical situation.

4.1.2 Most accurate

We consider `large-v3` to be the most accurate, and as it can be run on an M4 chip, it is still "affordable." It requires about 6 GB of memory, and uses a bit more during decoding. This would be a good model for deploying an application in which the ASR system is accessed through an API. The hosting service could engineer the process quite well and run the model on specialized hardware, making it a very accurate transcription service, yet still fast. This model could also be used on-device by linguists who need the most accurate transcription possible, but are worried about data privacy, and are willing to wait a while for a more accurate transcription, thereby minimizing the need for manual corrections.

4.1.3 Best balance

We consider `medium` to offer the best balance. It has both RTF and WER roughly midway between those of `small` and `large-v3` (or is closer than other versions of Whisper to this). This would be practical for an amateur with an interest in speech recognition, for on-device speech recognition where an approximate transcription to be sent to an LLM or as a text is okay, etc.

4.2 Optimization impacts

Unfortunately, faster- models are not actually faster when the native models can be implemented on an M4 MPS device, but the `faster-` models cannot. They are significantly slower, generally one-half to one-quarter the speed, with quantization sometimes even hurting speed on the CPU.

On the other hand, distilled models are genuinely faster, given their PyTorch implementations can be executed on the M4 MPS device. However, their severe loss in accuracy on languages other than English is not practical in cases where those languages are expected.

In general, on a present-day Mac, native Whisper implementations are still the best choice.

4.3 Practical deployment

If writing an application that runs speech recognition locally on a Mac, native Whisper implementations are best. Depending on the accuracy needed (say, an application to be used by a linguist versus for an LLM chatbot), a much larger (smaller) model should be used to prioritize performance (efficiency). If many files are going to be processed at once, batch processing could greatly reduce RTF, yielding a more practical use case for larger models being used by linguists or others processing large datasets.

4.4 Dataset insights

Audio conditions were not a key factor in this study, though of course are impactful more generally. Noisy environments, background music, and other non-speech sounds as well as low sampling rates can reduce the quality of transcriptions as the model is not necessarily optimized to separate speech from non-speech, though in principle, Whisper has also been trained to ignore non-speech.

Shorter audio files such as those in the self-recorded Wiki Titles dataset take proportionally longer to process, likely due to Whisper's preprocessor which pads audio to 30 seconds in general. This means that the RTF is artifically reduced as compared to the actual duration of the audio - it is not really a "fair" comparison in that way. There is also no non-American-accented speech in the Wiki Titles dataset, though there are some rare words. This generally yielded a lower WER than for CommonVoice English, indicating that Whisper is more attuned to rare words in an American accent than it is to common words in other accents.

WER for Spanish is generally higher across models, most notably for the distil- variants which were not trained on non-English data. The best-performing model tested on CommonVoice Spanish in this study is nine points behind the current SOTA.

5 Integration with chatbot

My application allows the user to select any ASR model they prefer, allowing the user to decide for themself what the appropriate performance-efficiency tradeoff is. However, I would probably recommend `medium`, the balanced model, to any user of my application. Especially when dealing with English, `medium` is much faster than any larger model, but has a very similar WER. Its Spanish WER is not too bad either, indicating some reasonable multilingual capabilities. In any case, when passing the transcribed message to an LLM, the LLM will likely be robust enough in most cases to handle some transcription mistakes. In the worst case, the user may have to repeat themself occasionally due to a transcription mistake.

I did not need to make any modifications as compared to Assignment 2 to achieve this - it was already achieved. I suppose I could include a recommendation in the `README.md` file. My final system runs reasonably fast as I tested it. I do not have a systematic way to evaluate it, but it transcribed my speech quite quickly, and the lag was clearly mostly from the LLM.

6 Conclusion

In general, Mac users with an M4 chip should stick to native Whisper implementations from the `openai-whisper` library until more kernels are written for MPS implementing key operations needed by CTranslate2 implementations of PyTorch models using architectures such as those underlying Whisper.

If accuracy is a must and the user is willing to wait a long time for a more accurate transcription, such as a linguist might (or anyone else working with a large amount of audio data that needs to be transcribed quite accurately), `large-v3` is the appropriate model due to its high accuracy. If, on the other hand, a user is a software engineer prototyping software which must give a reasonable approximation of what a good ASR model would produce as a transcript, but which must execute/load fairly quickly, `small` is the appropriate model due to its not-too-unreasonable WER (even on languages other than English) but very fast loading time and low RTF. Finally, if balance is desired, such as in a non-critical application (perhaps interaction with a chatbot via speech, wherein the LLM is likely robust to transcription errors, and an error won't trigger any serious mishap), a model like `medium` is appropriate given its balance between reasonably low RTF and reasonably low WER.

Users should also be sure that their selected model can be loaded into MPS memory. Macs generally have at least 16 GB of RAM, all of which is accessible as VRAM to the MPS device. Therefore, any Whisper variant can be run locally on a Mac with plenty of room to spare, as the largest variants use only $\sim$ 6 GB.

This study has many limitations. We test only on very small datasets, which are not necessarily representative of either English or Spanish more generally. Additionally, there is little variability in terms of recording quality and accent, particularly for the Wiki Titles dataset, which was all recorded in the same location, on the same day, by the same speaker. There is some amount of diversity in CommonVoice, but not much. We also do not test on hardware other than the M4 chip, and our comparisons are not necessarily "fair" due to the CTranslate2 implementations not being available on the MPS device, which supports efficient, parallelized computation for native PyTorch implementations.

In this assignment, I learned that optimized models have a ways to go, and there is more engineering to be done to make them play nicely with different hardware, as well as a need for distilled multilingual models.

References

Rosana Ardila, Megan Branson, Kelly Davis, Michael Kohler, Josh Meyer, Michael Henretty, Reuben Morais, Lindsay Saunders, Francis Tyers, and Gregor Weber. 2020. Common Voice: A Massively-Multilingual Speech Corpus. In *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pages 4218–4222, Marseille, France. European Language Resources Association.

Max Bain, Jaesung Huh, Tengda Han, and Andrew Zisserman. 2023. WhisperX: Time-Accurate Speech Transcription of Long-Form Audio. In *Proc. Interspeech 2023*, pages 4489–4493.

Sanchit Gandhi, Patrick von Platen, and Alexander M. Rush. 2023. Distil-Whisper: Robust Knowledge Distillation via Large-Scale Pseudo Labelling. *arXiv preprint*. ArXiv:2311.00430 [cs].

Intel. 2018. bfloat16 - Hardware Numerics Definition.

Jitsi. 2025. jitsi/jiwer. Original-date: 2018-06-19T00:38:38Z.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray,

Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling Laws for Neural Language Models. *arXiv preprint*. ArXiv:2001.08361 [cs].

OpenNMT. 2025. OpenNMT/CTranslate2. Original-date: 2019-09-23T08:10:42Z.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, and 2 others. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine Mcleavey, and Ilya Sutskever. 2023. Robust Speech Recognition via Large-Scale Weak Supervision. In *Proceedings of the 40th International Conference on Machine Learning*, pages 28492–28518. PMLR. ISSN: 2640-3498.

Monica Sekoyan, Nithin Rao Koluguri, Nune Tadevosyan, Piotr Zelasko, Travis Bartley, Nikolay Karpov, Jagadeesh Balam, and Boris Ginsburg. 2025. Canary-1B-v2 & Parakeet-TDT-0.6B-v3: Efficient and High-Performance Models for Multilingual ASR and AST. *arXiv preprint*. ArXiv:2509.14128 [cs].

SYSTRAN. 2025. SYSTRAN/faster-whisper. Original-date: 2023-02-11T09:17:27Z.