Assignment 3: Whisper Optimization

Emma Rafkin
Georgetown University
epr41@georgetown.edu

Abstract

In this assignment, I evaluate Whisper across three dimensions: accuracy, time, and compute resources. I evaluate 8 models on three datasets: Librispeech, FLEURS Swedish, and one which I made myself. The best models were also tested using Faster Whisper and quantization. The best model overall was integrated into my chatbot.
Code for this project is hosted on Github[1].

1 Introduction

Whisper is a powerful Automatic Speech Recognition (ASR) and Automatic Machine Translation (AMT) model that is trained on hundreds of thousands of hours of weakly labeled audio data. It is open source and has been used as a state of the art (SOTA) system since it was released in 2022. Whisper v2 added more data on top of v1 by labeling unlabeled audio using v1. Whisper v3 is trained on even more audio, for longer, and with larger mel-frequency bins. Recently, Whisper released a native optimized version called Whisper Turbo, only available in the large size. Distil Whisper trains a smaller student model using the original Whisper models as teacher models. Finally Whisper can be optimized using Faster-Whisper which is built on CTranslate2 as well as using quantization. Most of the variants (except turbo) are released in multiple sizes. Performance varies across model sizes, as well as training techniques. Therefore, similar to the LLM exploration in assignment 1, it is important to test a variety of models to ascertain which is the best for the use case at hand.

I plan to use this model in my chatbot. I have a CPU on with a relatively old Mac Intel chip, therefore it is extremely important that the model be lightweight in order to work on my machine. I only speak English and Spanish which are two high resource languages, therefore intensive multilingual capabilities are not as important for my use case but would certainly be an added bonus. Throughout this experiment, I evaluate different Whisper models in order to determine which is the best for my system. I look at three dimensions of success: transcription accuracy, latency, and CPU usage. Since all three of these dimensions are important for my system, I chose the best overall to integrate into my chatbot.

2 Methods

2.1 Hardware Specifications

Initially, the phase 1 experiments were run on a Mac Intel Chip. This is an older chip meaning that no GPU is available and many ML libraries are not optimized for it. Notably, the Intel chip does *not* support floating point 16 optimization or Faster Whisper, which will be discussed further in Section 3.2.

Therefore, in order to get the optimization experiments to work, I then ran the best three models again on a Ubuntu machine with 125GB of RAM with 8 cores of CPU for both the optimized and non-optimized model versions for a fair comparison.

2.2 Model Selection

The models were initially chosen based on my local laptop's limitations. I wanted to make sure that the model could run on my CPU in case I want to run my final chatbot on my machine. Therefore, the largest Whisper models were not considered because they would not run on this machine. Whisper versions were only available across the large version for native Whisper and DistilWhisper. With this limitation, I tried to vary the versions testing v2 for DistilWhisper large and Whisper-Turbo for version 3.5. I also wanted to test all the sizes:

[1] https://github.com/erafkin/ling4467_assignment_1

Model
openai/whisper-tiny
openai/whisper-small
openai/whisper-medium
openai/whisper-base
distil-whisper/distil-large-v3.5
distil-whisper/distil-large-v2
openai/whisper-tiny.en
openai/whisper-large-v3-turbo

Table 1: Models evaluated, ordered by size ascending

base, tiny, small, medium, and large. Additionally, some of the models are finetuned to English only. I thought that this would be interesting to test, even though it would definitely fail on the non-English dataset. Generally, for my use case of my personal chatbot, I will be speaking in English if it significantly outperformed its multilingual counterpart in English it might be worth integrating into the final syste. Table 1 outlines the models tested. After running all of the models on my local CPU-only constrained machine, I chose the models that had the overall lowest WER combined with generally low RTF. For my personal system, I was evaluating on best overall when it comes to accuracy vs compute/speed.

2.3 Datasets

My English-only dataset was Librispeech (Panayotov et al., 2015). The non-English dataset that I used was the Swedish split of FLEURS (Conneau et al., 2022). I choose Swedish because I thought it would be interesting to see Whisper performance on a medium-resource langauge–not giving it something it has barely seen but also not giving it a language as easy as English. Both datasets are read speech, Librispeech being audiobooks and FLEURS being read samples of sentences from Wikipedia by three different native speakers. Librispeech has a “clean" split which was used for this project. I sampled the first 100 samples from the test splits. This unfortunately means that the data was unshuffled, which is particularly harmful for the Librispeech dataset as it means that there were not that many different speakers tested. This was done so that the whole dataset did not have to be downloaded, which optimized the system performance, because the size of the Librispeech download was overloading the system. According to the OpenASR Leaderboard (Srivastav et al., 2023), the SOTA performance on Librispeech is NVIDIA's Canary-Qwen-2.5b with w WER of 1.4. It is difficult to find one source of truth about the SOTA for Swedish ASR models, however KBLab recently finetuned Whisper on 50,000 hours of transcribed Swedish speech and achieved a WER of 5.4 with their largest model (Vesterbacka et al.).

For my own dataset, I had ChatGPT generate 100 short sentences of varied length.[2] This dataset is helpful because it is guaranteed that there was no data leakage as there is no way Whisper could have been trained on this. The sentences were recorded using a script similar to the audio recording script written for assignment 2, although it was edited to allow the recorded to end the recording by hitting the Enter key rather than waiting for silence. These sentences were all recorded by me in a quiet room on my personal laptop. The goal was to achieve high quality recordings that would be similar to my final chatbot use case: me speaking into my laptop.

2.4 Evaluation Methodology

In order to evaluate each model, for each transcription I calculated WER as the accuracy measure, RTF as the latency measure, and the CPU usage as the compute resource measure. WER was calculated using the Jiwer (Morris et al., 2004) python library. RTF was calculated using $(end\_time - start\_time)/duration\_of\_audio$.

3 Results

3.1 Phase 1: Native Model Comparison

I ran each dataset through all 8 selected models and then averaged the three measurement scores across each model/dataset pairing. Results for phase 1 can be seen in Table 2. From these calculations, I tried to pick the three best overall models. Since none of the Whisper Large versions were able to run on my CPU, I chose the largest that was able to run, distil-large-v3.5, because it has the lowest WER and relatively low RTF and CPU compared to the other big models. It should be noted that this model did extremely poorly on Swedish. Then I chose the Whisper Small and Medium models because they were by far the fastest to run and had good results for my dataset, which is the one I am optimizing for. Whisper tiny is not ideal for Swedish speech (and likely other non-English languages), but Whisper small improved on Whisper tiny significantly (although 23% WER is still not ideal). Whisper tiny.en only slightly improved on

[2]The sentences can be found in this script

Model	Dataset	WER %	RTF	CPU
distil-large-v2	**Fleurs**	105.4606	1.16465	1.16465
	Librispeech	3.093	2.100003	2.100003
	Local	0.5779	3.49161	3.49161
distil-large-v3.5	**Fleurs**	111.7755	1.1671	1.1671
	Librispeech	2.4447	2.049898	2.0498981
	Local	0.4161	3.450904	3.450904
whisper-base	**Fleurs**	43.5322	0.449274	0.449274
	Librispeech	4.7673	0.51665	0.51665
	Local	1.8816	0.696826	0.696826
whisper-large-v3-turbo	**Fleurs**	8.3569	2.219244	2.219244
	Librispeech	2.2562	4.065874	4.065874
	Local	0.4392	6.895355	6.895355
whisper-medium	**Fleurs**	13.3634	3.376804	3.376804
	Librispeech	3.072	4.733922	4.733922
	Local	0.5548	6.532167	6.532167
whisper-small	**Fleurs**	23.3351	1.198859	1.198859
	Librispeech	3.9884	1.572344	1.572344
	Local	0.5548	2.019912	2.019912
whisper-tiny	**Fleurs**	56.1315	0.234593	0.234593
	Librispeech	7.2538	0.268476	0.268476
	Local	1.7522	0.325837	0.325837
whisper-tiny.en	**Fleurs**	680.6947	0.853793	0.853793
	Librispeech	5.4738	0.222696	0.222696
	Local	1.2344	0.251218	0.251218

Table 2: Phase 1 Results

Whisper tiny for English, and was so horrible at Swedish that I figured that I would not use it for phase 2 because there was no benefit and it rules out any multilingual capabilities. Whisper Medium had better WER than Whisper small, especially on FLEURS. However the RTF and CPU were much higher. I wanted to see how this would do with the optimization because it from the results from phase 1 it looked like this model had the most to gain from a package like Faster-Whisper.

3.2 Phase 2: Optimization Results

After completing phase 1, I realized that Faster Whisper would not work on my local machine. I attempted to downgrade all the packages, change the compute type, lower the number of beams, and change how the audio was loaded in–the techniques suggested by many online forums. It appears that Faster Whisper simply will not work on a Mac Intel chip. Therefore, I used the Ubuntu machine described in Section 2.1 for phase 2. In order to make a fair comparison between the optimized and non-optimized versions, I reran phase 1 on that machine for those three selected models. Results can be seen in Table 3. This machine was more powerful and has a GPU, so this significantly sped things up and lowered compute costs for the CPU, especially for Whisper Medium, as expected.

I ran each dataset through the model loaded in with Faster-Whisper. Additionally, rather than using Float32, I tried using both Float16 and Int8 for optimization. Results can be seen in Table 4. Comparing these results to those in Table 3 demonstrates that running model using Faster Whisper did very little to help with the RTF and CPU usage on the Ubuntu machine. This could be because it is mainly running on the GPU anyway, so it is possible that if GPU resources were measured there might be more of an effect with using Faster Whisper. Additionally, the RTF was already really low on the Ubuntu machine so it is possible that there is not much that can be done to make the model much faster. Accuracy stayed more or less the same. The quantization options also did not improve the latency or the CPU much, but float16 slightly improved on both measurements over int8. However, float16 yielded slightly worse WER than int8. This is likely due to the nondeterministic nature of the models rather than the quantization.

All of the models tested did much worse than the SOTA on the Swedish data, which is unsurprising because the SOTA is finetuned for Swedish. The models also did worse than SOTA on Librispeech but not as dramatically. Overall, it appears that the larger the model the better the WER, but the tradeoff for latency and compute is really high. Quantization and Faster Whisper appear to help slightly, but the biggest aid in running these models is GPU access.

4 Discussion

Of the three models in phase 2, the best model for English WER was the DistilWhisper, because as with other NLP tasks, generally accuracy improves as model size increases. However, DistilWhisper's monolingual training leads to an extremely high WER, making it a bad selection for a general system. The RTF and CPU usage across the three models were similar, therefore it doesn't really matter which one would be chosen for the final system if it is being run on the GPU. If the model is being run on a CPU, then it makes sense to look at the results in Table 2. In that case, Whisper small is by far the best option, because RTF and CPU are much lower than Whisper medium and the WER is similar.

All of these tests were run serially rather than using batch processing. This was done to emulate the final deployment scenario of the chatbot, where data is processed serially. However, the pipeline object and models are optimized for batch processing and so it is likely that the RTF would significantly reduce in this scenario across the board. It would be useful to test batch processing performance for a different system.

All of the data tested as clean audio, and therefore it is unclear how these models would perform on noisy data. However, there was a difference in latency between these datasets. It seems that loading the local audio lead to RTF slowdowns particularly on the CPU in comparison to the datasets loaded and processed by Huggingface. This is an interesting finding and demonstrates some sort of inefficiency with the Huggingface datasets library that is unclear how to fix. This will be relevant to the final use case, as the chatbot pipeline loads in local audio file as well. None of the models were particularly great at non-English data, but the larger the model the better it performed on FLEURS. If the final chatbot were to be commonly used for multilingual purposes, then it would be important to use the largest model possible.

Model	Dataset	WER %	RTF	CPU
distil-large-v3.5	**Fleurs**	111.7755	0.025927	0.025927
	Librispeech	2.4447	0.040921	0.040921
	Local	0.4161	0.061329	0.061329
whisper-small	**Fleurs**	23.3351	0.048601	0.048601
	Librispeech	3.9884	0.053986	0.053986
	Local	0.5548	0.057284	0.057284
whisper-medium	**Fleurs**	13.3634	0.101695	0.101695
	Librispeech	3.072	0.120192	0.120192
	Local	0.5548	0.133306	0.133306

Table 3: Phase 1 Results rerun on new machine. This machine is more powerful and the GPU was used, so this significantly sped things up and lowered compute costs for the CPU.

Quantization	Model	Dataset	WER %	RTF	CPU
Float16	**distil-large-v3-ct**	**Fleurs**	110.61557	0.031334	0.031334
		Librispeech	2.7243	0.043186	0.043186
		Local	0.4161	0.079578	0.079578
	whisper-medium	**Fleurs**	13.1788	0.031356	0.031356
		Librispeech	3.8986	0.044941	0.044941
		Local	0.5548	0.060271	0.060271
	whisper-small	**Fleurs**	23.3824	0.021571	0.021571
		Librispeech	4.0598	0.024433	0.024433
		Local	0.749	0.031079	0.031079
Int8	**distil-large-v3-ct**	**Fleurs**	110.79296	0.05806	0.05806
		Librispeech	2.765	0.120432	0.120432
		Local	0.4161	0.219316	0.219316
	whisper-medium	**Fleurs**	13.4706	0.033256	0.033256
		Librispeech	3.368	0.048769	0.048769
		Local	0.5548	0.064534	0.064534
	whisper-small	**Fleurs**	23.3276	0.022278	0.022278
		Librispeech	4.2418	0.025365	0.025365
		Local	0.749	0.031732	0.031732

Table 4: Phase 2 results, running data through models loaded via Faster-Whisper and changing quantization

5 Integration with Chatbot

For my chatbot, I decided to run a check when I instantiate the model to see if the device that the pipeline is being run on has access to a GPU. If so, I run Whisper medium and if not I run Whisper small.

I chose these models because they appeared to have the besat balance between WER, latency, and compute resources. WER is important because this is a point of failure in the pipeline–if the transcription is wrong than the LLM answer will likely be incorrect or irrelevant. However, these smaller models generally did quite well transcribing my English speech in a clean environment, which is the use case of my system. Additionally, as mentioned in 2.1, these models are expensive to run on my system, so it is important that they are as lightweight as I can make them. Therefore, Whisper small seemed like the best balance, but when a GPU is available, I can afford to use a more powerful model.

6 Conclusion

This experiment demonstrated that ASR models improve as they grow in size and are trained on more data, but that growth is costly in latency and compute resources. While the DistilWhisper models can be quite effective, they do experience significant accuracy loss, especially on non-English data. Faster Whisper and quantization are beneficial for optimization, but not nearly as much as simply running the models using a GPU. A major limitation of my study is that I did not measure GPU consumption. It is possible that Faster Whisper and quantization significantly help with GPU consumption. When running these models serially and unbatched, one has to select a model that strikes a good balance between accuracy, latency, and compute cost. This study showed that the models performance shift significantly depending on the dataset that they are measured on. My study only tested a subset of domains of data, so if there is a usecase in a less general domain (e.g. medical), then it is imperative to test using a dataset that is in that domain.

References

Alexis Conneau, Min Ma, Simran Khanuja, Yu Zhang, Vera Axelrod, Siddharth Dalmia, Jason Riesa, Clara Rivera, and Ankur Bapna. 2022. Fleurs: Few-shot learning evaluation of universal representations of speech. *Preprint*, arXiv:2205.12446.

Andrew Morris, Viktoria Maier, and Phil Green. 2004. From wer and ril to mer and wil: improved evaluation measures for connected speech recognition.

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. 2015. Librispeech: An asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210.

Vaibhav Srivastav, Somshubra Majumdar, Nithin Koluguri, Adel Moumen, Sanchit Gandhi, et al. 2023. Open automatic speech recognition leaderboard. `https://huggingface.co/spaces/hf-audio/open_asr_leaderboard`.

Leonora Vesterbacka, Faton Rekathati, Robin Kurtz, Justyna Sikora, and Agnes Toftgård. Welcome KB-Whisper, a new fine-tuned Swedish Whisper model!