Article

Performance and Efficiency Evaluation of ASR Inference on the Edge

Santosh Gondi [1,*] and Vineel Pratap [2]

[1] Facebook Inc., Menlo Park, CA 94025, USA
[2] Facebook AI Research, Menlo Park, CA 94025, USA; vineelkpratap@fb.com
* Correspondence: sgondi@fb.com

Abstract: Automatic speech recognition, a process of converting speech signals to text, has improved a great deal in the past decade thanks to the deep learning based systems. With the latest transformer based models, the recognition accuracy measured as word-error-rate (WER), is even below the human annotator error (4%). However, most of these advanced models run on big servers with large amounts of memory, CPU/GPU resources and have huge carbon footprint. This server based architecture of ASR is not viable in the long run given the inherent lack of privacy for user data, reliability and latency issues of the network connection. On the other hand, on-device ASR (meaning, speech to text conversion on the edge device itself) solutions will fix deep-rooted privacy issues while at same time being more reliable and performant by avoiding network connectivity to the back-end server. On-device ASR can also lead to a more sustainable solution by considering the energy vs. accuracy trade-off and choosing right model for specific use cases/applications of the product. Hence, in this paper we evaluate energy-accuracy trade-off of ASR with a typical transformer based speech recognition model on an edge device. We have run evaluations on Raspberry Pi with an off-the-shelf USB meter for measuring energy consumption. We conclude that, in the case of CPU based ASR inference, the energy consumption grows exponentially as the word error rate improves linearly. Additionally, based on our experiment we deduce that, with PyTorch mobile optimization and quantization, the typical transformer based ASR on edge performs reasonably well in terms of accuracy and latency and comes close to the accuracy of server based inference.

Keywords: automatic speech recognition; ASR; edge inference; Raspberry Pi; transformers; PyTorch; GreenAI

check for updates

Citation: Gondi, S.; Pratap, V. Performance and Efficiency Evaluation of ASR Inference on the Edge. *Sustainability* **2021**, *13*, 12392. https://doi.org/10.3390/su132212392

Academic Editors: Arun Kumar Sangaiah, Xi Zheng, Changqing Luo, Shuihua Wang, Ankit Chaudhary and Eklas Hossain

Received: 18 September 2021
Accepted: 5 November 2021
Published: 10 November 2021

1. Introduction

Automatic speech recognition (ASR) is a process of converting audio to text. Applications of ASR include dictation, accessibility, hearables, voice assistants, AR/VR applications, among other things. Speech recognition has improved greatly in recent times because of the adoption of deep learning based techniques for training and inferencing [1]. However, most of the state-of-the-art speech recognition models are deployed based on cloud computing architectures where input from user devices is sent to the server for processing and results are returned back to the device. This model can not guarantee the privacy and security of user-sensitive audio data. This architecture is also prone to reliability, latency, and availability issues because of the reliance on the network.

Machine learning practitioners typically optimize for the accuracy of the model to achieve state of the art results. Recent studies focusing on GreenAI [2,3] reveal that after a certain point, achieving linear incremental improvements to accuracy (in this case the WER) leads to exponential increase in the energy cost for training and inference. The carbon footprint of these applications will worsen the global energy crisis as deep learning based AI applications obtain widely adopted [4,5]. Specifically for ASR applications, one solution is to adopt a balanced approach by using an ASR model based on its energy, accuracy, and performance trade-offs. For example, in the case of on-device dictation applications, minor and infrequent ASR errors can be tolerated as long as the app running on the device can provide a way to correct some of these errors via keyboard or scribble. Having lower WER will definitely improve the user experience. However, if the lower WER burns battery at higher rate, a sensible product decision would be to settle for an acceptable WER within the energy budget and provide other UI affordances to satisfy product or application requirements. With experimental evaluations in this paper we attempt to inform the research community about the energy and accuracy trade-offs of ASR inference on the edge.

On-device ASR inherently provides privacy and security to sensitive user data and is more reliable and performant by precluding the need for network connectivity. Apart from these obvious reasons, on-device ASR inference is also a more sustainable solution in terms of energy efficiency. For cloud based processing, the audio needs to be streamed to the server. There is also the energy cost of Wi-Fi/LTE for connection establishment and data transfer [6], routing across the internet and computation in data centers. Moreover, the computations on data centers are more expensive than on a typical hand held device given the multicore CPUs and associated cooling mechanisms. All these energy costs will be saved by on-edge processing.

Related to our work, Ref. [7] evaluated the on-device speech recognition performance with DeepSpeech [8], Kaldi [9], and Wav2Letter [10] models. Effects of thermal throttling on CNN vision models was researched in [11]. In [12], authors evaluate the performance, power consumption, and thermal impact of CNN based vision inference on edge devices. Authors in [3] predict the energy cost of NLP inference using software approach. In this paper, our objective is to measure the trade-off between accuracy, energy, and performance of ASR with a typical transformer based model on the edge. To the best of our knowledge there are no prior articles or research work which assess ASR inference on edge in terms of performance and efficiency. Many of the on-edge evaluation papers focus on CNN based models for computer vision.

We used Raspberry Pi for evaluations for a few reasons. It is an easily programmable and configurable Linux based device. We can control the environment, such as co-running apps to ensure that there are no side effects during evaluations. We could easily use off-the-shelf power meters to measure steady state and inference time energy consumption. Research completed in this paper [4] shows that hardware based energy evaluations are at-least 20 pcs more accurate than software based energy management solutions. Further, the hardware specs of Raspberry Pi 4 are comparable to smart speaker, smart displays, AR/VR products, etc. This makes Pi a good proxy device to assess ASR performance and efficiency of typical edge devices.

Major contributions of this paper are as follows:

- We present a process for measuring energy consumption of ASR inference on Raspberry Pi using an off-the-shelf energy meter;
- We measure and analyze the accuracy and energy efficiency of the ASR inference with a transformer based model on an edge device and show how energy and accuracy vary across different sized models;
- We examine the performance and computational efficiency of ASR process in terms of CPU load, memory footprint, load times, and thermal impact of various sized models;
- We also compare on-edge WER with that of server's for the same dataset.

The rest of the paper is organized as follows: In the background section we discuss ASR and transformers. In the experimental setup, we go through the steps for preparing the model and setting up, for inferencing and energy measurements. We go over the accuracy, performance and efficiency metrics in the results section. Finally, we conclude with a summary and outlook.

2. Background

ASR

ASR is the process of converting audio signals to text. At high level, ASR consists of an acoustic model, a pronunciation model and a language model to convert raw audio to text [13]. Over time, DL systems [1,14] such as CNN [15,16] and RNN [17] were applied to different ASR components. Since then, ASR model architectures have evolved to convert audio signals to text directly, which are known as end-to-end and sequence-to-sequence models. These architectures have simplified the training and implementation of ASR models. The most successful end-to-end ASR systems are based on connectionist temporal classification (CTC) [18], recurrent neural network (RNN) transducer (RNN-T) [17], and attention-based encoder-decoder architectures [19]. Recently, hybrid model systems have shown significant improvements in accuracy for streaming ASR [20,21]. Transformer is a sequence-to-sequence architecture originally proposed for machine translation [22]. It has since been adopted for ASR [23,24], with the difference being that the input is audio frames instead of the text as in translation tasks.

ASR is a CPU and memory intensive task. The resource budget to consider on edge device for ASR purposes are CPU, memory, storage, energy/battery life, as well as the thermal budget (if it is a mobile or a wearable device). Apart from resource constraints, other long-term disadvantages include, less frequent model updates (in case of cloud, continuous updates, and applying hot patches is easier), lack of real-time data for improving the model overtime, lack of metadata about ASR process for debugging [25], etc. The overall benefits of privacy, reliability, energy conservation of on-device processing outweigh the engineering challenges which anyway need to be addressed by edge AI community.

In this paper, we evaluate the accuracy, performance, and efficiency of inference with a transformer based ASR model on a Raspberry Pi device. We have used fairseq speech-to-text models [24] using same architecture as in [22]. It is a transformer based encoder, decoder with seq-2-seq loss. Small model has self-attention dimension of 256 with 12 layers of encoder, 6 layers of decoder. Similarly, the medium model has self-attention dimension of 512 with 12 layers of encoder, 6 layers of decoder, and large model has self-attention dimension of 1024 with 12 layers of encoder, 6 layers of decoder. In addition, it also has input subsampler before the transformer encoder to project the input into transformer dimension, as well as to downsample input sequence. It is trained with a Librispeech 960hr of labeled training dataset. Model accepts 80-channel log mel-filter bank extracted features with 25 ms window size and 10 ms shift. Additionally, utterance-level cepstral mean and variance normalization (CMVN) [26] transform is applied before feeding to subsampler. The decoder uses 10,000 unigram vocabulary.

We experimented with beam search width of 1 and 5 variants for the all the considered models during decoding. Beam search is a common heuristic algorithm for decoding structured predictors [27]. The decoder in these models use a sentence piece tokenizer. Using a language model on top, should further improve the WER. We consider experimenting with different types transformer architectures and language model as future work. In this study, we explore the relationship between WER and energy consumption for different sized transformer based ASR systems. The models used in this paper are not currently state-of-the-art in-terms of accuracy. However, we chose these models as a starting point because of they are easily accessible and programmable through fairseq and huggingface platforms.

Figure 1 shows the simplified flow of the ASR process with this model.

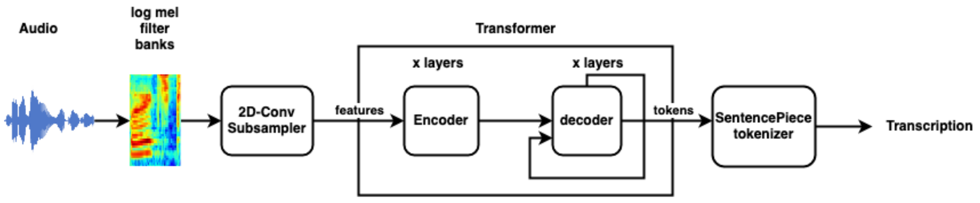

Figure 1. Speech2Text inference.

3. Experimental Setup

3.1. Model Preparation

We have used PyTorch [28] based models for evaluation. PyTorch is an open source machine learning framework based on torch library. Figure 2 shows the steps for preparing the models for inferencing on edge devices.

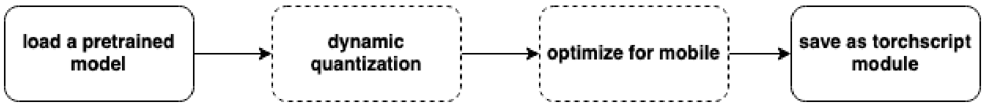

Figure 2. Model preparation steps.

We first go through a few of the PyTorch tools and APIs used in our evaluation.

3.1.1. TorchScript

TorchScript is the means by which PyTorch models can be optimized, serialized and saved in intermediate representation (IR) format. torch.jit (https://pytorch.org/docs/stable/jit.html, accessed on 5 November 2021) APIs are used for converting, saving, and loading PyTorch models as ScriptModules. TorchScript itself is a subset of the Python language. Hence, sometimes, a model written in Python needs to be simplified to convert it into a script module. TorchScript module can be created either using tracing or scripting methods. Tracing works by executing the model with sample inputs and capturing all computations whereas, scripting does static inspection to go through the model recursively. The advantage of scripting over tracing is that it correctly handles the loops and control statements in the module. A saved script module can then be loaded either in a Python or C++ environment for inferencing purposes. For our evaluation, we generated ScriptModules for Speech2Text models after applying quantization and optimizations.

3.1.2. PyTorch Mobile Optimizations

PyTorch provides a set of APIs for optimizing the models for mobile platforms. It uses module fusing, operator fusing, quantization, among other things to optimize the models. By quantization, model weights and computation are reduced to int8 precision from float precision. This reduces memory, storage and computation cycles of the model.

3.1.3. Speech2Text Model

Pretrained Speech2Text models are imported using the fairseq (https://github.com/pytorch/fairseq/tree/master/examples/speech_to_text, accessed on 5 November 2021) framework. Fairseq is a sequence modeling toolkit that allows researchers and developers to train custom models for various text and speech related tasks. Some minor syntactic changes, such as type hints, were needed to convert the generator model to a TorchScript module. For this evaluation, we have considered s2t_transformer_s, s2t_transformer_m, and s2t_transformer_l architectures. Table 1 shows the model details for each of the quantized and original versions of models. Given the CPU, Memory, and disk constraints on Raspberry Pi, we made use of Google Colab for importing, optimizing, and saving the model as TorchScript module. The saved modules were copied to Raspberry Pi for inferencing. The decoding uses a beam search decoder with beam size of 5 and a Sentence-Piece tokenizer.

Table 1. Model details.

Model Name	Size	Config	Parameters
Speech2Text small quantized	80 MB		
Speech2Text small	125 MB	256d, 12 L enc./6 L dec.	30 Million
Speech2Text medium quantized	184 MB		
Speech2Text medium	299 MB	512d, 12 L enc./6 L dec.	71 Million
Speech2Text large quantized	617 MB		
Speech2Text large	1100 MB	1024d, 12 L enc./6 L dec.	268 Million

3.2. Datasets

We use Librispeech [29], a standard ASR bench-mark that contains 1000 h of labeled English speech from audiobooks. It has a combination of train, dev, and test datasets. For evaluation, test and dev datasets are used.

3.3. Raspberry Pi Setup

Raspberry Pi 4 B (https://www.raspberrypi.org/products/raspberry-pi-4-model-b/, accessed on 5 November 2021) is used for evaluation. It is a basic headless module (Figure 3) with no additional peripherals attached to it. Relevant device specs are provided in Table 2. The default Raspberry Pi OS is 32 bit which is not compatible with PyTorch. Therefore, we installed a 64 bit OS version on Pi.

Table 2. Raspberry Pi 4 B spec.

Name	Spec
Chip	BCM2711
CPU	Quad core Cortex-A72 (ARM v8) 64-bit SoC
Clock speed	1.5 GHz
RAM	4 GB SDRAM
Caches	32 KB data + 48 KB instruction L1 cache per core. 1 MB L2 cache
Storage	32 GB micro SD card
OS	64 bit Raspberry Pi OS
Python version	3.7
Power supply	5V DC via USB-C connector

- PyTorch was built and installed from source on Pi with XNNPACK and QNNPACK cmake configs. Quantized Neural Networks Package (QNNPACK) (https://github.com/pytorch/QNNPACK, accessed on 5 November 2021) is a mobile-optimized library for low-precision high-performance tensor computations. Similarly, XNNPACK (https://github.com/google/XNNPACK, accessed on 5 November 2021) is a mobile optimized libary for higher precision tensor computations;
- Since the library for MFCC feature computations could not find on Raspberry Pi, the MFCC features for librispeech datasets were pre-generated on server using fairseq audio_utils (https://github.com/pytorch/fairseq/blob/master/fairseq/data/audio/audio_utils.py, accessed on 5 November 2021). They were then saved as NumPy files and used them as input to models on Raspberry Pi;
- The device backend was set to qnnpack during inference in the script as follows: torch.backends.quantized.engine = 'qnnpack'.

3.4. Energy and Temperature Measurements

We used a USB power meter [30] for measuring the power consumption on Raspberry Pi. Figure 3 shows the setup with energy monitoring.

Following are the steps for measuring energy during inference:

- Store the list of audio samples along with ground truths in a USB-thumb drive and plug it into Raspberry Pi;

- Power up Raspberry Pi through the USB meter as shown in Figure 3;
- Login to Raspberry Pi and run the energy evaluation script;
- The script first reads the audio manifest file, logs the timestamp to a file and waits for a few seconds in steady state. The power consumed by Raspberry Pi at this state is considered a steady state value;
- Load the model and start the audio inference. Record load and inference time stamps. Exit the evaluation script;
- Download the amps/voltage series from energy meter through an android app;
- Post-process the logs from evaluation script and the time series values from USB power meter. Compute watts, and mWh when the script was idle and during inferencing. Compute the delta to calculate the actual power consumption by the inference process.

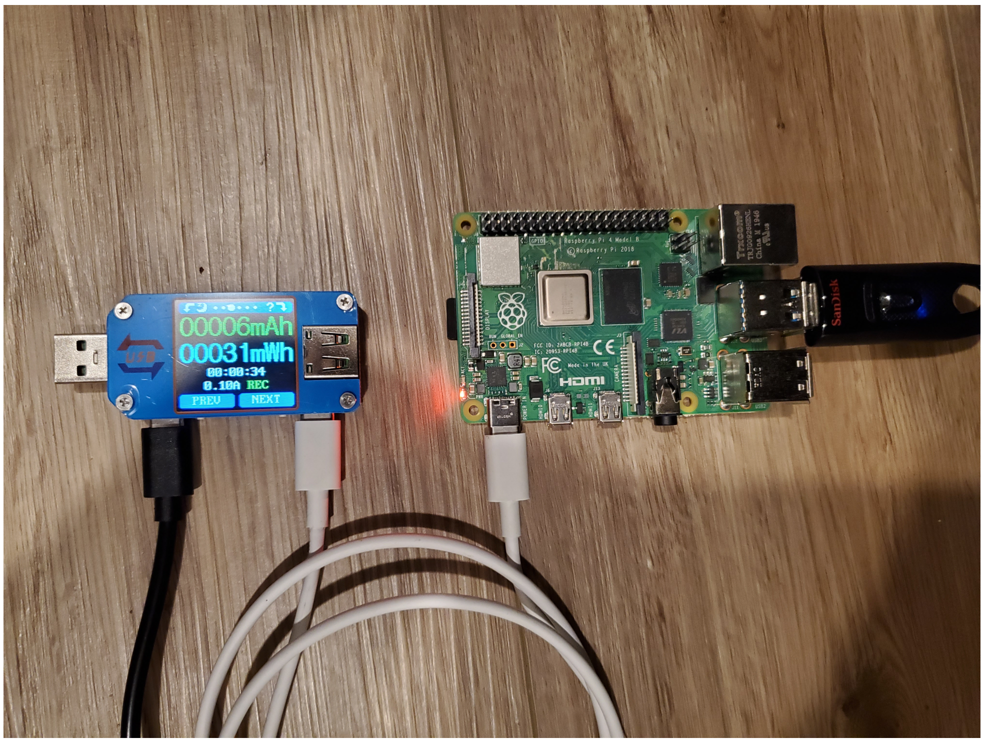

Figure 3. Energy monitoring setup.

CPU temperature and frequency are monitored using vcgencmd measure_temp (https://www.raspberrypi.org/documentation/computers/os.html#vcgencmd, accessed on 5 November 2021) and vcgencmd measure_clock arm on Raspberry Pi. We collect these values continuously in a separate script while ASR is ongoing. The default frequency scaling governor on Raspberry Pi is ondemand (https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt, accessed on 5 November 2021). In this mode, CPU frequency remains at 1.5 GHz as long as the temperate is below the max threshold of 82 °C. If the temperature goes beyond that threshold, it starts to reduce CPU frequency up-to the lower limit of 600 MHz. In our experiments, we also set the frequency scaling governor to powersave (which effectively sets the CPU frequency to 600 MHz) to check the performance of various models with lower CPU frequency.

4. Results

We evaluate different sized Speech2Text models with beam width of 5 and 1 configurations for accuracy, performance and efficiency. The word-error-rate (WER) metric is used for measuring accuracy. WER is defined in Equation (1). The real-time-factor (RTF) is used to measure performance. RTF is defined in Equation (2). We measure energy, model load time, memory footprint, and CPU percentage of the execution script to evaluate the efficiency of the model inference.

$$\text{WER} = (\text{substitutions} + \text{insertions} + \text{deletions})/\text{number of words in audio} \tag{1}$$

$$\text{RTF} = (\text{read time} + \text{inference time} + \text{decoding time})/\text{total utterance duration} \tag{2}$$

Librispeech test and dev datasets together contain ~21 h of audio. It would have taken us many days to execute the inference on all combinations of models with all the audio in these datasets. To save time, for this experiment, we randomly sampled 30% of the audio files in each of the four datasets for inference and evaluation. The same sampled datasets were used in all the on-device and on-server inference test runs thus ensuring that the results are comparable.

4.1. WER

Table 3 shows the WER of various models.

Table 3. WER.

Dataset	Model	Device WER		Server WER	
		Beam5	**Beam1**	**Beam5**	**Beam1**
test-clean	S2T small quant	4.7%	4.7%		
	S2T small	4.6%	4.6%	4.4%	4.6%
	S2T medium quant	4.2%	4.5%		
	S2T medium	3.9%	4.0%	3.9%	4.0%
	S2T large quant	3.7%	3.7%		
	S2T large	3.7%	3.7%	3.7%	3.7%
test-other	S2T small quant	11.7%	11.7%		
	S2T small	11.0%	11.0%	10.4%	11.0%
	S2T medium quant	10.4%	11.2%		
	S2T medium	9.4%	10.0%	9.4%	10.0%
	S2T large quant	9.6%	10.3%		
	S2T large	9.4%	10.2%	9.4%	10.2%
dev-clean	S2T small quant	4.2%	4.2%		
	S2T small	3.7%	3.7%	3.6%	3.7%
	S2T medium quant	3.6%	3.9%		
	S2T medium	3.3%	3.4%	3.3%	3.4%
	S2T large quant	3.4%	3.6%		
	S2T large	3.1%	3.4%	3.1%	3.4%
dev-other	S2T small quant	11.1%	11.1%		
	S2T small	10.7%	10.7%	9.7%	10.7%
	S2T medium quant	9.8%	10.6%		
	S2T medium	9.1%	9.4%	9.1%	9.4%
	S2T large quant	8.8%	9.6%		
	S2T large	8.8%	9.3%	8.8%	9.3%

WER is slightly higher for the quantized models compared to the unquantized float versions. This is the cost-benefit trade-off of accuracy vs. RTF and efficient inference. Test-other and dev-other datasets have a higher WER compared to test-clean and dev-clean datasets. This is also expected because other datasets are noisier compared to clean ones. WER improves as the model obtains bigger with more number of encoder/decoder layers and parameters. For medium and large models, WER is generally lower for beam size of 5 configuration compared to beam size of 1. Since beam size of 1 means greedy search, it might exclude some of the early hypothesis which might yield better overall match at the later stages of decoder search.

The server side inference evaluation was carried out on a Google Colab instance. Essentially the same pre-trained model is imported on Colab instance except without the

PyTorch dynamic quantization and mobile optimization steps. For medium and large models, the Colab WER is same as observed on the edge. However, on smaller model, the on-device WER is higher than the server based one. We are still trying to understand this specific behavior.

4.2. RTF

In our experiments, RTF is dominated by model inference time >99% compared to other two factors in Equation (2). Table 4 shows the RTF for all models in both beam size 1 and beam size 5 configurations. RTF does not vary between different datasets for the same models. Hence, we show RTF (avg, mean, and p75) per model instead of one per dataset.

RTF improved by 10% for small quantized model compared to unquantized floating point model. Similarly, for medium and large models the improvement is by 20% and 50%, respectively. RTF is ∼30% lower for medium and large models with beam width 1 compared to 5, whereas for small model there is no difference between two configurations. Improvements in RTF going from unqunatized to quantized, larger to smaller and beam 5 to beam 1 are expected because there will be lesser number of computations and search involved. We do not observe the improvement in RTF on small model by reducing the beam size possibly because the small model is quite refined by PyTorch mobile optimizations.

Overall, the small and medium models have RTF less than 1.0 in normal mode, making them potentially suitable for real-time inference.

Table 4. RTF.

		Beam5			**Beam1**	
Model	**Avg**	**Mean**	**P75**	**Avg**	**Mean**	**P75**
S2T small quant	0.29	0.28	0.33	0.29	0.27	0.33
S2T small	0.33	0.32	0.38	0.33	0.33	0.37
S2T medium quant	0.62	0.61	0.68	0.40	0.37	0.45
S2T medium	0.77	0.77	0.84	0.52	0.50	0.56
S2T large quant	1.08	1.09	1.16	0.77	0.76	0.81
S2T large	2.12	2.12	2.23	1.54	1.52	1.59

4.3. Temperature

We measure the impact of long running ASR process on CPU temperature. Temperature increase is particularly significant for wearable and mobile devices where the devices are closer to the human body and, therefore, tolerance for increase is very limited, unlike for the machines in datacenter or other stand-alone home devices. As shown in Table 5, the temperature increases close to max level 80 °C in normal mode a 28 °C increase from steady state (52 °C). In power save mode, the increase is only by 14 °C. Note that these increases do not happen for single inference session but after few minutes of continuous inference. The increase also depends on ambient temperature.

Table 5. Temperature.

Mode	**Temperature (°C)**	**Increase (°C)**
ondemand	80.0	28.0
powersave	66.0	14.0

Out of curiosity, we ran ASR inference on small quantized model for long time (>1 h) to check the effects of thermal throttling on CPU frequency and ASR RTF. Figure 4 shows the weighted-average variation in temperate, RTF, and CPU frequency. The Linux dynamic-voltage-frequency-scaling (DVFS) reduces the average CPU frequency when the temperature approaches 82 °C (thermal limit) which, in turn, increases the RTF (40%). The throttling limit can reach very quickly if the ambient temperature is higher.

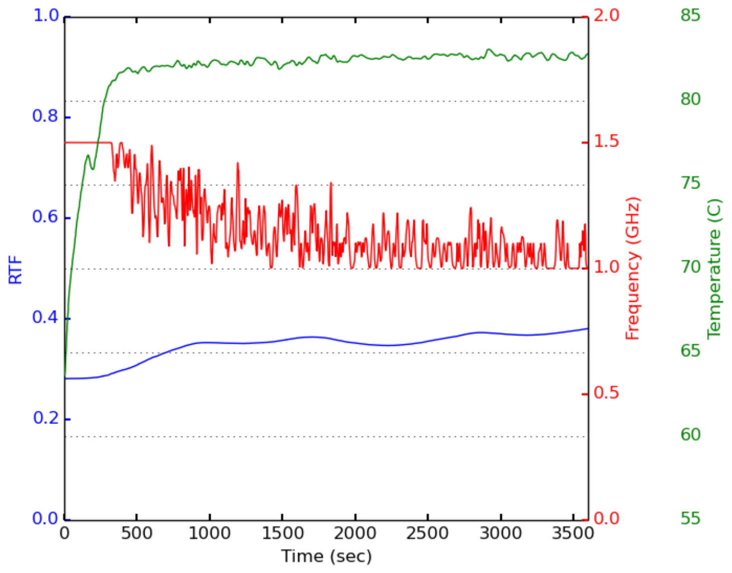

Figure 4. CPU temperature, frequency, and ASR RTF variation.

4.4. *Energy*

Figures 5 and 6 show the watts consumption over time for small quantized model in normal and powersave modes. The steady state wattage in normal mode is 3.6 W (5.2 V × 0.7 A) on Raspberry Pi. This increases by 1.6 W when the ASR inference script is active. Similarly, in powersave mode, the increase is 0.65 W during ASR inference from steady state.

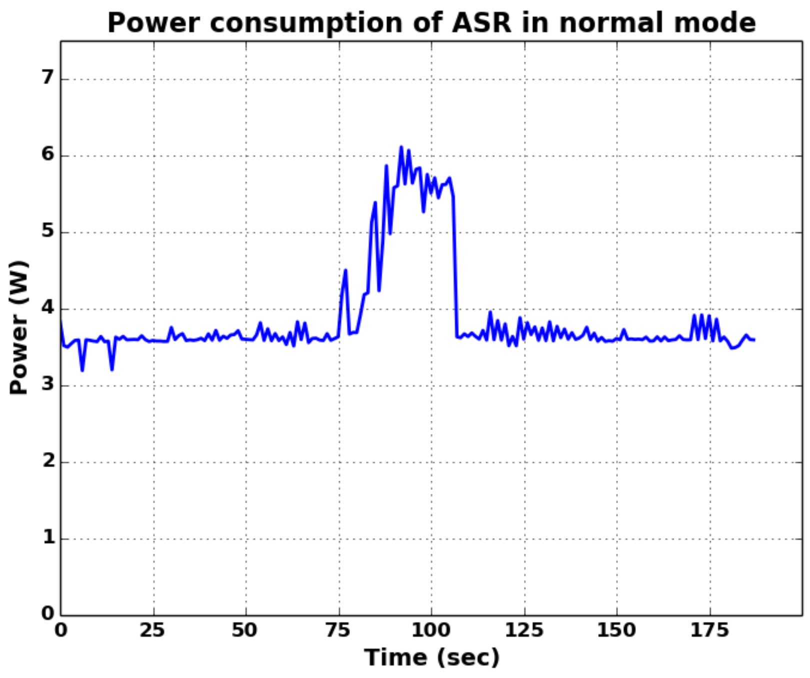

Figure 5. Normal mode power. ASR power consumption in normal mode is ∼1.6 W.

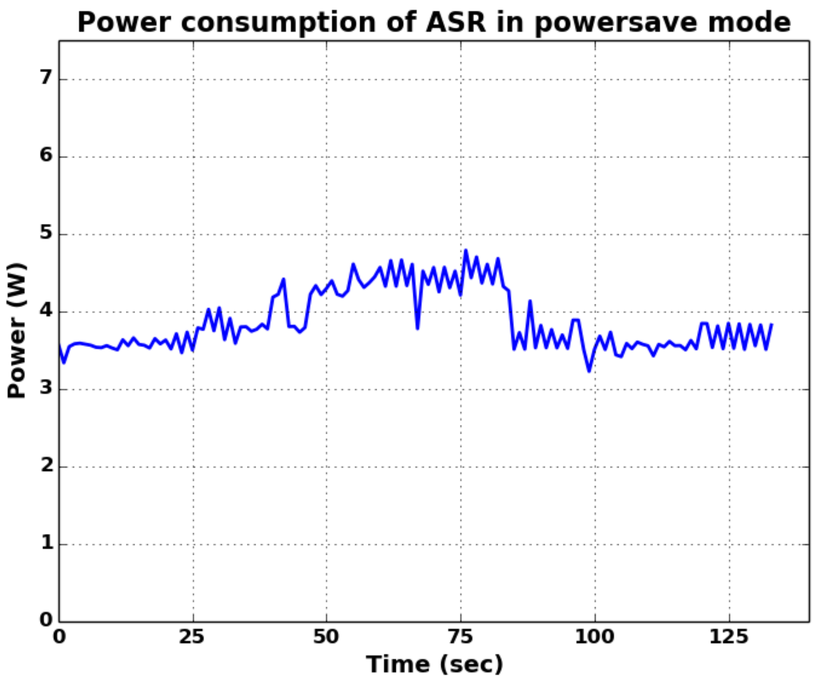

Figure 6. Powersave mode power. ASR power consumption in powersave mode is ∼0.65 W.

Table 6 shows the ASR energy consumption of both beam size 1 and 5 configurations for all models for a 10 s long utterance. Small model consumes 5× to 8× less power compared to larger models.

For comparison, if we run small quantized ASR model on Apple watch with a 303.8 mAh (1.17 Wh) battery capacity, ASR for a 10 s utterance would roughly consume ∼0.08% of battery capacity. This is only for the computation part. In real scenarios, such as dictation, voice commands, etc., we also need to account for power used by microphone for input capture and UI display for any associated output during the usage.

Table 6. Energy table (mWh).

Model	Beam5	Beam1
S2T small quant	1.32	1.41
S2T small	1.53	1.58
S2T medium quant	3.57	2.24
S2T medium	4.77	2.90
S2T large quant	7.13	5.10
S2T large	13.62	9.13

Figures 7 and 8 show the variation of energy consumption vs. WER. We have chosen test-clean dataset as a baseline for graphs since the energy consumption does not vary much across the different datasets. The energy on Y axis is plotted in log scale. The liner/geometric line of growth on the log scale, for both beam 1 and 5 configurations, clearly points toward exponential growth in energy as WER rate decreases incrementally. Improving WER on quantized model by 1.2% resulted in ∼4× to 5× increase in energy consumption. In case of unquantized model, 0.9% WER improvements resulted in ∼6× to 9× increase in energy consumption. Although the energy measurements on this model clearly show exponential growth, we also want to validate the same with larger set of models as part of future work.

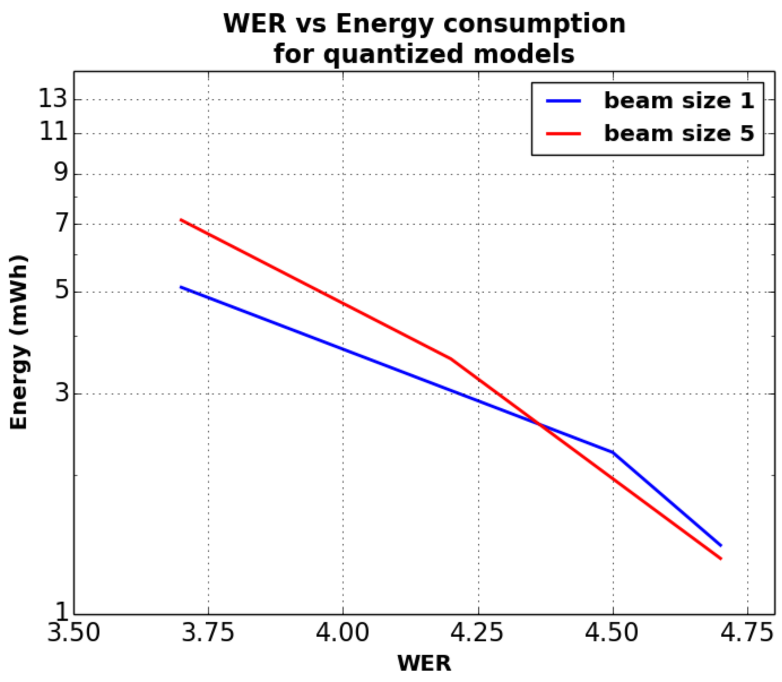

Figure 7. Energy vs. WER for quantized models. Incremental WER improvement leads to a factor of magnitude higher energy consumption.

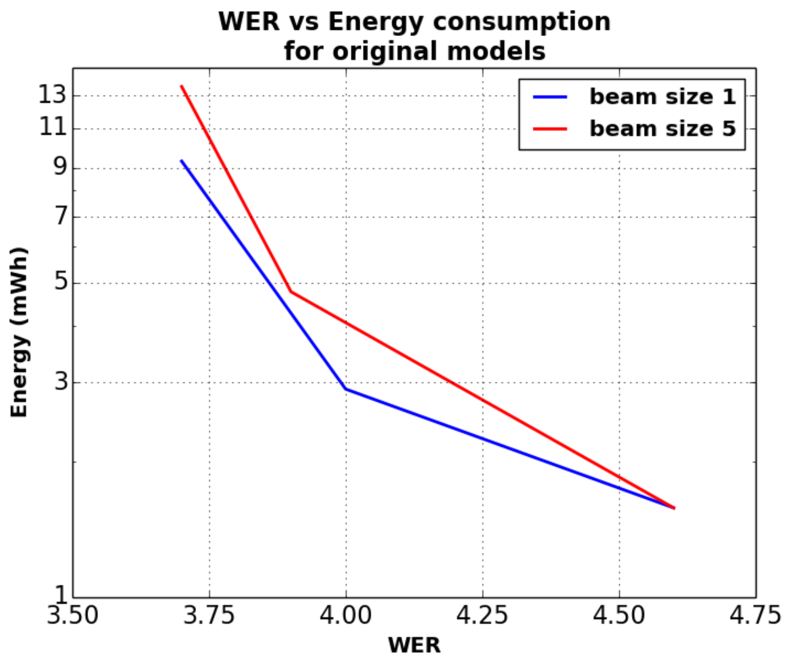

Figure 8. Energy vs. WER for original models. WER vs. energy consumption growth is even steeper for original models compared to quantized versions.

4.5. Memory and CPU

Figures 9 and 10 shows the memory and CPU consumption of all the models used in this experiment. small quantized model memory footprint is 295 MB. By quantization, memory footprint reduced by 35% for small model. For larger models, quantization reduces the memory by higher margins; 50% for medium model and 68% for larger model. The models with decoder beam size of 1 generally have slightly lower memory footprint compared to models with beam size 5. This is not surprising given that the decoder scanning space is smaller with beam 1. The overall memory accounting is a complicated process. We deduce that all processes have common memory pages because of shared libraries, code and other process infra related pages. Therefore, quantized small model does not show large gains in memory as compared to bigger models. The footprint largely co-relates to model size and the number of parameters in the models.

The memory values presented here are avg RES (resident set size) values from Linux top (https://man7.org/linux/man-pages/man1/top.1.html, accessed on 5 November 2021) command. RES includes clean, dirty, and the shared memory pages of the process. One thing to notice is that almost all the model data pages are clean (read-only). These clean pages can be swapped (on systems with swap space) out by OS on memory pressure without terminating the process because of out-of-memory (OOM) condition.

CPU utilization is ∼95% in all cases. ASR is a CPU intensive process. All the models fully utilize all four available cores during inferencing.

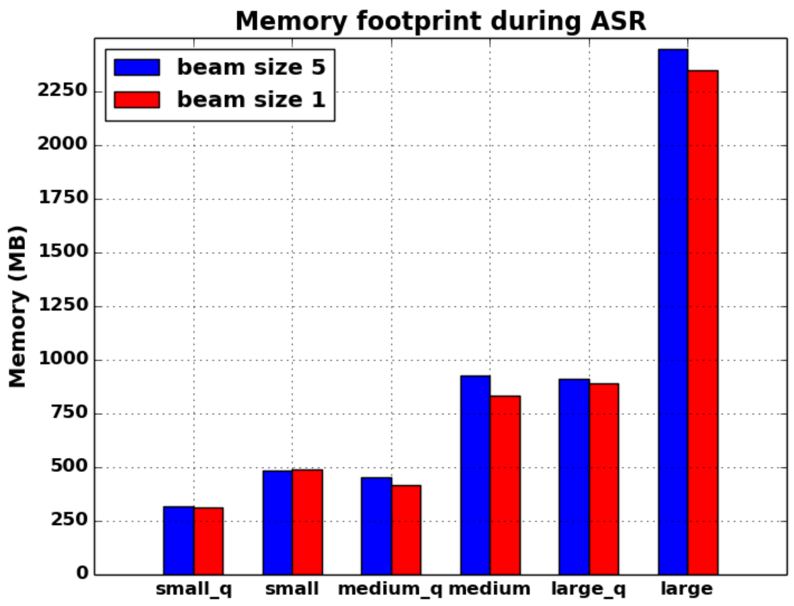

Figure 9. Memory footprint. Quantized model memory footprint is 30% to 60% smaller corresponding to the original float models.

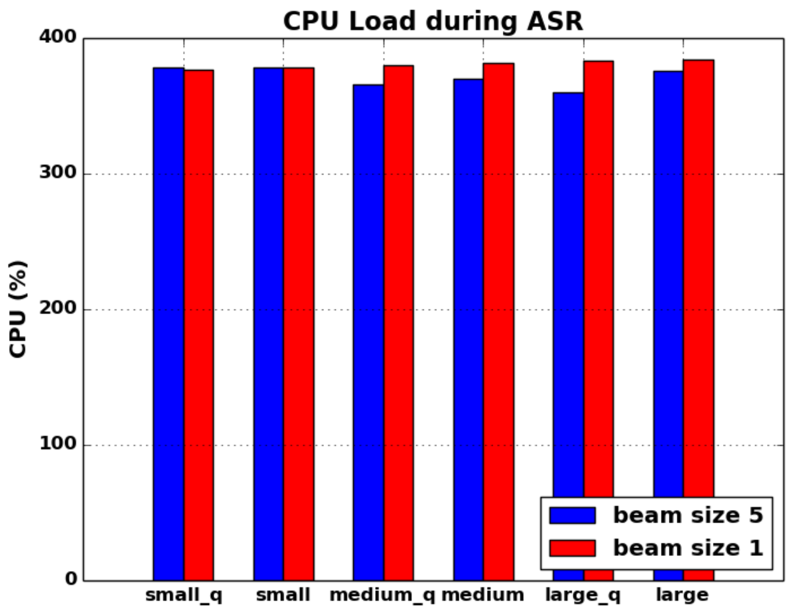

Figure 10. CPU load. Raspberry Pi CPU capacity is 400%. CPU load during ASR is almost 400% for all the models.

Model Load Time

Table 7 shows the model load times of all the models used in this experiment. The load times are measured from clean state, i.e., after the reboot, the first-time loads are considered. Typically, the operating system caches the file pages in the memory. So, the load times can be smaller on subsequent uses. The small quantized model load time avg is 2.2 s. This is small enough to be loaded on-demand in an application for real-time dictation, voice commands, etc. The other load times roughly co-relate to the model sizes and parameters as shown in Table 1.

Table 7. Model load times.

Model	**Avg (s)**
Speech2Text small quantized	2.2
Speech2Text small	4.2
Speech2Text medium quantized	3.8
Speech2Text medium	8.4
Speech2Text large quantized	9.1
Speech2Text large	28.7

5. Conclusions

We presented a methodology for measuring energy consumption on Raspberry Pi using off-the-shelf USB power meter. We evaluated the ASR accuracy, performance and efficiency of a transformer based speech2text models on Raspberry Pi. With WER $\sim$10%, <300 MB memory footprint and RTF well below 1.0, quantized small architecture models can be efficiently deployed on mobile and wearables devices. The small quantized model is also quite efficient in terms of power consumption. It consumes $\sim$0.1% of battery for 10 s audio inference on a smaller device, such as an Apple Watch 3. Larger models can be used to achieve higher accuracy at the expense of higher RTF, memory, and power. For medium and large models using lower beam size, RTF and energy consumption significantly reduce, albeit with a small degradation in WER. We believe accuracy, latency, and energy trade-offs in on-edge AI is a rich area to explore considering different types of architectures, number of hyper parameters, decoders with language model, etc. With this initial study we could deduce that inference energy consumption increases exponentially in relation to linear improvements in WER. Machine learning developers continue to research ways to improve the model accuracy. It is a well established knowledge in the research community that the training of ASR models with lower WER will also require factor of magnitude higher energy. With our study we could reason that inference is no different in that regard. Hence, we conclude that a careful consideration and balance of application's WER requirements and energy budget is crucial before deploying an ASR model for edge inferencing. We also conclude that, temperature throttling is not an issue for short duration ASR inference but something to keep in mind during continuous long duration inference as it can affect the RTF because of throttling.

In the future, we are planning to explore streaming ASR energy consumption with latest model architectures. We are also planning to build a generic system using off-the-shelf hardware and software packages to be able to measure energy vs. accuracy trade-offs on any models agnostic to ML platforms.

Author Contributions: Conceptualization—S.G. and V.P.; methodology—S.G. and V.P.; setup and experiments—S.G.; original draft preparation—S.G.; review and editing—S.G. and V.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Publicly available Librispeech datasets were used in this study. This data can be found here: https://www.openslr.org/12 (accessed on 5 November 2021).

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DL	Deep learning
CPU	Central processing unit
GPU	Graphics processing unit
ASR	Automatic speech recognition
HMM	Hidden Markov model
RNN	Recurrent neural network
RNNT	Recurrent neural network transducer
CNN	Convolutional neural network
LSTM	Long short-term memory
Speech2Text	Speech to text transformer model from fairseq
GMM	Gaussian mixture model
DNN	Deep neural network
CTC	Connectionist temporal classification
CMVN	Cepstral mean and variance normalization
MFCC	Mel-frequency cepstral coefficients
WER	Word error rate
RTF	Real time factor
mWh	Milliwatt-hours
AR/VR	Augmented reality/Virtual reality

References

1. Hinton, G.; Deng, L.; Yu, D.; Dahl, G.E.; Mohamed, A.R.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Sainath, T.N.; et al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Process. Mag.* **2012**, *29*, 82–97. [CrossRef]
2. Schwartz, R.; Dodge, J.; Smith, N.; Etzioni, O. Green AI. *Commun. ACM* **2020**, *63*, 54–63. [CrossRef]
3. Cao, Q.; Lal, Y.K.; Trivedi, H.; Balasubramanian, A.; Balasubramanian, N. IrEne: Interpretable Energy Prediction for Transformers. In Proceedings of the ACL/IJCNLP, Virtual Event, 1–6 August 2021.
4. Cao, Q.; Balasubramanian, A.; Balasubramanian, N. Towards Accurate and Reliable Energy Measurement of NLP Models. *arXiv* **2020**, arXiv:2010.05248.
5. Strubell, E.; Ganesh, A.; McCallum, A. Energy and Policy Considerations for Modern Deep Learning Research. *Proc. AAAI Conf. Artif. Intell.* **2020**, *34*, 13693–13696. [CrossRef]
6. Wu, C.J.; Brooks, D.; Chen, K.; Chen, D.; Choudhury, S.; Dukhan, M.; Hazelwood, K.; Isaac, E.; Jia, Y.; Jia, B.; et al. Machine Learning at Facebook: Understanding Inference at the Edge. In Proceedings of the 2019 IEEE International Symposium on High Performance Computer Architecture (HPCA), Washington, DC, USA, 16–20 February 2019; pp. 331–344. [CrossRef]
7. Peinl, R.; Rizk, B.; Szabad, R. Open Source Speech Recognition on Edge Devices. In Proceedings of the 2020 10th International Conference on Advanced Computer Information Technologies (ACIT), Deggendorf, Germany, 16–18 September 2020; pp. 441–445. [CrossRef]
8. Hannun, A.Y.; Case, C.; Casper, J.; Catanzaro, B.; Diamos, G.F.; Elsen, E.; Prenger, R.J.; Satheesh, S.; Sengupta, S.; Coates, A.; et al. Deep Speech: Scaling up end-to-end speech recognition. *arXiv* **2014**, arXiv:1412.5567.
9. Povey, D.; Ghoshal, A.; Boulianne, G.; Burget, L.; Glembek, O.; Goel, N.; Hannemann, M.; Motlicek, P.; Qian, Y.; Schwarz, P.; et al. The Kaldi Speech Recognition Toolkit. In Proceedings of the IEEE 2011 Workshop on Automatic Speech Recognition and Understanding, Waikoloa, HI, USA, 11–15 December 2011; IEEE Signal Processing Society: Piscataway, NJ, USA, 2011.
10. Pratap, V.; Hannun, A.; Xu, Q.; Cai, J.; Kahn, J.; Synnaeve, G.; Liptchinsky, V.; Collobert, R. Wav2Letter++: A Fast Open-source Speech Recognition System. In Proceedings of the ICASSP 2019—2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brighton, UK, 12–17 May 2019; pp. 6460–6464. [CrossRef]
11. Benoit-Cattin, T.; Velasco-Montero, D.; Fernández-Berni, J. Impact of Thermal Throttling on Long-Term Visual Inference in a CPU-Based Edge Device. *Electronics* **2020**, *9*, 2106. [CrossRef]
12. Hadidi, R.; Cao, J.; Xie, Y.; Asgari, B.; Krishna, T.; Kim, H. Characterizing the Deployment of Deep Neural Networks on Commercial Edge Devices. In Proceedings of the 2019 IEEE International Symposium on Workload Characterization (IISWC), Orlando, FL, USA, 3–5 November 2019; pp. 35–48. [CrossRef]
13. ASR. Available online: http://www.cs.columbia.edu/~mcollins/6864/slides/asr.pdf (accessed on 5 November 2021).

14. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef]
15. Gu, J.; Wang, Z.; Kuen, J.; Ma, L.; Shahroudy, A.; Shuai, B.; Liu, T.; Wang, X.; Wang, G.; Cai, J.; et al. Recent advances in convolutional neural networks. *Pattern Recognit.* **2018**, *77*, 354–377. [CrossRef]
16. Abdel-Hamid, O.; Mohamed, A.R.; Jiang, H.; Deng, L.; Penn, G.; Yu, D. Convolutional Neural Networks for Speech Recognition. *IEEE/ACM Trans. Audio Speech Lang. Process.* **2014**, *22*, 1533–1545. [CrossRef]
17. Graves, A. Sequence Transduction with Recurrent Neural Networks. *arXiv* **2012**, arXiv:1211.3711.
18. Graves, A.; Fernández, S.; Gomez, F.; Schmidhuber, J. Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In Proceedings of the ICML '06: 23rd International Conference on Machine Learning, Pittsburgh, PA, USA, 25–29 June 2006; Association for Computing Machinery: New York, NY, USA, 2006; pp. 369–376. [CrossRef]
19. Bahdanau, D.; Cho, K.; Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv* **2015**, arXiv:1409.0473.
20. Lüscher, C.; Beck, E.; Irie, K.; Kitza, M.; Michel, W.; Zeyer, A.; Schlüter, R.; Ney, H. RWTH ASR Systems for LibriSpeech: Hybrid vs Attention—w/o Data Augmentation. *arXiv* **2019**, arXiv:1905.03072.
21. Baquero-Arnal, P.; Jorge, J.; Giménez, A.; Silvestre-Cerdà, J.A.; Iranzo-Sánchez, J.; Sanchís, A.; Saiz, J.C.; Juan-Císcar, A. Improved Hybrid Streaming ASR with Transformer Language Models. In Proceedings of the INTERSPEECH, Shanghai, China, 25–29 October 2020.
22. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, L.U.; Polosukhin, I. Attention is All you Need. In *Advances in Neural Information Processing Systems*; Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2017; Volume 30.
23. Baevski, A.; Zhou, H.; Mohamed, A.; Auli, M. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. *arXiv* **2020**, arXiv:2006.11477.
24. Wang, C.; Tang, Y.; Ma, X.; Wu, A.; Okhonko, D.; Pino, J. fairseq S2T: Fast Speech-to-Text Modeling with fairseq. In Proceedings of the 2020 Conference of the Asian Chapter of the Association for Computational Linguistics (AACL): System Demonstrations, Suzhou, China, 4–7 December 2020.
25. Véstias, M.P. A Survey of Convolutional Neural Networks on Edge with Reconfigurable Computing. *Algorithms* **2019**, *12*, 154. [CrossRef]
26. Prasad, N.V.; Umesh, S. Improved cepstral mean and variance normalization using Bayesian framework. In Proceedings of the 2013 IEEE Workshop on Automatic Speech Recognition and Understanding, Olomouc, Czech Republic, 8–12 December 2013; pp. 156–161. [CrossRef]
27. Meister, C.; Vieira, T.; Cotterell, R. Best-First Beam Search. *Trans. Assoc. Comput. Linguist.* **2020**, *8*, 795–809. [CrossRef]
28. PyTorch. Available online: https://pytorch.org/ (accessed on 5 November 2021).
29. Panayotov, V.; Chen, G.; Povey, D.; Khudanpur, S. Librispeech: An ASR corpus based on public domain audio books. In Proceedings of the 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), South Brisbane, QLD, Australia, 19–24 April 2015; pp. 5206–5210. [CrossRef]
30. Makerhawk. Um25c Usb Power Meter. Available online: https://www.makerhawk.com/ (accessed on 5 November 2021).