Evaluating the Computation versus Communication Trade-off on IoT devices for Speech Recognition Applications

Marie Shaw
Department of ECE
mnshaw@andrew.cmu.edu

Namrita Murali
Department of ECE
npmurali@andrew.cmu.edu

***Abstract*—In an era of smart homes and smart devices, embedded devices are becoming increasingly common as a form of data collection and processing. Moreover, an increasing number of smart home devices are being controlled by voice input, which makes audio processing on embedded devices crucial. Many speech recognition engines perform complex Machine Learning algorithms on audio samples to interpret voice input. In order to perform quick and accurate processing of speech data, devices need to determine whether to run these speech recognition engines on device or communicate data to a server to offload recognition and have the results communicated back to the device. Since most embedded devices are energy and network constrained, there are energy and time tradeoffs between processing locally or externally. This paper analyzes how performance and energy consumption varies when performing speech recognition processing locally on an embedded device or communicating data to a server to process externally.**

***Index Terms*—Speech Recognition, Machine Learning, Performance, Embedded Systems, Raspberry Pi, Power Consumption**

I. Introduction

A. Motivation

There are around 23.14 billion connected devices in the world in 2018, and that number is projected to more than triple by the year 2025 [1]. Connected embedded devices are low power, small in size, and usually serve a specific computing purpose. Due to their size and special-purpose nature, small embedded devices are typically both power, memory, and network constrained. Typically, embedded devices have a large quantity of sensors for the special-purpose task they are designed to perform, and are responsible for collecting and processing data from their surrounding environment. With the collection of data comes data processing, usually done through machine learning algorithms.

Small embedded devices with limited power and memory that perform heavy machine learning computation face very slow performance and poor battery life. With more of these connected devices appearing in people's homes and many other environments, there is a need to process huge amounts of data with high performance. The choice of performing machine learning computation on an embedded device or sending the data to an external server for processing is crucial to the tradeoff between energy consumption and performance.

Speech recognition is a very common application of embedded devices data processing, Almost one in five Americans has access to a smart speaker, like a Google Home or Amazon Alexa [2]. Most smart home devices are voice controlled, and require speech processing in order to interpret commands.

There are advantages and disadvantages to performing the computation online and offloading it. On a power constrained device, training and running a sophisticated machine learning model may take too much time for the immediate feedback necessary. However, on a networking constrained device, sending large amounts of data may also take too much time and energy. The real crux of this problem lies in the energy and performance tradeoff between computation and communication on embedded devices.

B. Related Work

Most of the previous research done in the area of comparing online and offline machine learning methods focuses on the difference in accuracy of models that are trained locally and models that are trained on an external server [4], [5]. While these papers mention the differences in performance between the two methods, they highlight that although offline models can be faster, online models have higher accuracy.

Other related work has been done to profile power consumption on various devices. These devices include desktop computers, laptops, smartphones, tablets, and other embedded devices. These studies indicate how usage patterns and applications affect power consumption of a device. While this area of research compares the power consumption of running specific applications on a range of different devices, most of the applications do not involve heavy machine learning computation.

More specifically, there has been a lot of analysis on the power consumption of Raspberry Pis. One paper analyzes how different types of operations induce varying power consumption on the Raspberry Pi as compared to performing the same operations on other devices [5]. It determined that on average, the Raspberry Pi had much less power consumption on key functional operations than other larger devices. However, this study only looked at comparing energy usage from key functions, such as device start-up or downloading a file. None of these operations required heavy machine learning computation

as a speech recognition model would. The study also did not measure the power consumption if tasks were offloaded, so there is no communication component.

There are also several online resources that describe how to run various speech recognition engines on Raspberry Pis, which we will refer to for implementation of various parts of our project [3], [4].

II. Technical Description

A. Hardware

We chose to use a Raspberry Pi 3 as the embedded system to collect data from. Raspberry Pis are very accessible due to their low cost, and as a result are commonly used in embedded system applications. Raspberry Pi 3s also run an ARM Cortex CPU, which is similar to many smartphones and other embedded devices. Raspberry Pi 3s also have built-in WiFi, which allows for network communication, which is a necessary component of our analysis. As a result, we thought they would be a good candidate for power and performance analysis. For the server to offload computation to, we used a Macbook Air laptop because of accessibility, and the hardware can comparably emulate common servers.

The technical specification for both of these pieces of hardware is detailed below:

Raspberry Pi 3 Model B

- ARM Cortex-A53 Quad-Core 1.2GHz
- 1GB SDRAM
- Wireless LAN and Bluetooth on board

MacBook Air

- Intel i5 Dual-Core 1.6GHz
- 8GB LP DDR3 RAM

In order to monitor the energy usage of the Raspberry Pi, we will use an external power meter. USB power meters are easy to use and relatively accurate. The specific USB meter we chose has an error rate of $\pm 1\%$, which is highly accurate for our needs.

Below is a list of the hardware we purchased for our experiments:

- Raspberry Pi 3
- USB Microphone
- USB Power Meter

B. Algorithms

Since most smart home devices run speech recognition engines to interpret commands, we decided to use an existing speech recognition engine to perform processing. Speech recognition engines run underlying complex machine learning models that very by engine. Some speech recognition optimize for accuracy, while others optimize for speed. When considering various options for Raspberry Pis, we considered speech recognition engines that were cocmmonly run on embedded devices, but were robust with relatively good accuracy.

There are two main speech recognition tools that we considered:

1) **CMU Sphinx a.k.a. PocketSphinx**
   CMU Sphinx is an open-source large vocabulary, speaker-independent continuous speech recognition engine. It is specifically tuned for handheld and mobile devices, but it can work on desktop devices as well. It provides an API for processing and decoding audio files. The main advantages of using PocketSphinx are that is is very stable, allows for reentrant decoding, and significantly reduced memory consumption than other speech recognition engines. While we are not measuring the energy consumption across different engines, it is beneficial that it has a smaller memory footprint so that we are able to run it on the Raspberry Pi. The library is 44MB in total, which is very lightweight compared to most data processing libraries. More information can be found here: https://cmusphinx.github.io/
2) **Google Speech API** The Google Speech-to-Text engine provides an API to access different speech recognition capabilities. Automatic Speech Recognition in particular is run using a deep neural network. The advantages of using the Google Speech-to-Text engine are they it has been extensively developed by Google developers, so the algorithms have been optimized well for performance. However, since the engine is very large, it may induce higher energy costs on device. It is easy to use on any device for any application, so we can run it on both the Raspberry Pi and the server. More information can be found here: https://cloud.google.com/speech-to-text/

These can both be run on board and offline.

After analysis of both tools, we decided to go with CMU PocketSphinx for several reasons. First, PocketSphinx is specifically made to run on devices with less processing power like mobile devices, which means it will run well on a Raspberry Pi. Its lightweight nature allows Additionally, we found that the procedure for running speech recognition onboard would be more simple with PocketSphinx. PocketSphinx allows for more customization, which will allow us to do further experimentation that we cannot with the Google API, such as changing the vocabulary size. Google Speech Recognition also imposes a limit on the number of words processed, and has a cost incurred with processing more than that amount. Since PocketSphinx is free of charge, we can experiment with many different sentences, making it the right choice for this project.

C. Server

To determine where to offload our computation to, we compared various servers. The main options were to use the AWS or Google Cloud IoT Core services to set up an AWS or Google Cloud VM instance, or to use our own laptop device.

We decided against using the AWS or Google Cloud IoT Core services because although they provide a good way to collect data from the Raspberry Pi, we would have to set up additional services in order to process and send back the data

to the Raspberry Pi. Moreover, the service seems useful for situations in which we are communicating with many devices, but would be overkill for our project which only uses a single Raspberry Pi. The overhead of these services is unnecessary for our simple use case.

After deciding to have an accessible server, such as a laptop or VM on the cloud, we decided to use a local machine rather than a virtual one. There are two major differences in these types of machines. The first is that a VM on the cloud would have more computing power. For example, our laptops have only 2 or 4 CPUs, while we can spin up a VM with 64 CPUs. The other is that there are more difficulties with setting our experiment up with a VM. Connecting the Raspberry Pi to a laptop is much easier, and monitoring traffic between the two devices is easier with a local machine. We determined that a laptop has significantly more resources than a Raspberry Pi already which will suffice for a comparison, and that it was the best option to quickly get started with our project. Therefore we decided that we will proceed with using a laptop as a server, and leave further analysis with a VM as future work.

D. Audio Samples

We experimented with both collecting samples through the USB microphone and through audio clips from databases online. We concluded that using a database with audio samples would allow for more standardization among the audio samples in terms of vocabulary and fluctuation, and allow us to vary the different voice genders. We used several different databases to retrieve audio samples from, but used the common Harvard Sentences Library that has been studied a lot in speech recognition applications. The Open Speech Repository provides sample sentences from the Harvard Sentences Library, which is where we pulled most of the large samples from. We also used another database that provides English sentences for the use of learning English. In both cases, we restricted the vocabulary size to under 100 words. We thought this would be a good source of sentences with high sound quality. Links to both of these databases can be found here:

- Open Speech Repository
- English Sentences with Audio Repository

We split the audio samples used into three different categories:

- Small samples: one word sentences
- Medium samples: 2-10 word sentences
- Large samples: 11-50 word sentences

We decided to use 10 samples per category to process enough data in each category to be able to correlate sample size to performance.

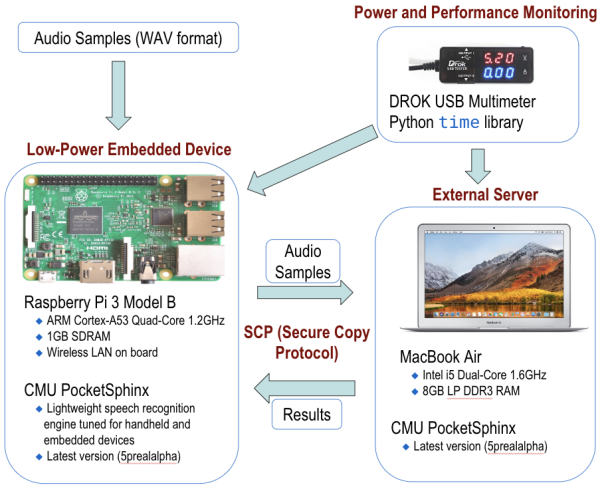
E. System Overview

Figure 1 is an overview of our system architecture. We will highlight the procedure in further detail below.

F. Procedure

The general procedure can be broken down into two main experiments: local processing and server processing.

Fig. 1. System Overview



Both the Raspberry Pi and the laptop have CMU PocketSphinx installed and run a similar program. The program is a Python script that takes in an audio file in the WAV format, runs the speech recognition engine on the audio file and pipes the result to a file. We pipe the results to the file in order to measure their accuracy. There is also timing code in both applications using Python's `time` library to measure the time it takes for the speech recognition to complete.

The communication between the Pi and the server occurs through SCP or Secure Copy Protocol. SCP is a network protocol that supports secure file transfers between two hosts on a network. We chose this protocol since most speech data that is processed on smart home devices is confidential information and needs to be transferred securely on the network.

In order to measure energy and power consumption, we monitor the USB meter and record the current and voltage instances at the program start and program completion. The power measured is from a delta of the starting current and maximum current while the computation is taking place. The energy measured is based on the power and the time recorded. Similarly, the power and energy are measured when the Raspberry Pi communicates the audio files to the server by measuring the delta of the starting current and maximum current while the files are being communicated.

1) In the first phase, we run the speech processing application locally on the Raspberry Pi and measure the energy, power consumption and time it takes to run. This phase is run for each of the audio samples (in all three categories), and all measurements are recorded. The files are downloaded to the Raspberry Pi separately from this phase, as we do not want to include the power and performance metrics of downloading the files in our analysis.
2) The second phase is to send the audio samples via WiFi to a server, run the application on the server, and measure the power consumption of the communication of the

data to the server and the results back to the Raspberry Pi. This phase is also run in successive iterations, one for each audio sample, and all measurements are recorded.

3) Based on the data collected, we compare the power and energy consumption of the two experiments for each category of audio samples, and perform analysis to determine how the size of the audio sample affects the power and energy in both experiments.
4) Since all of the results are piped to files, we calculate the word error rate or the accuracy of the speech engine for each of the audio samples processed. The word error rate is determined from a program we created to compare deltas in sentences. This measurement can be used in future work to determine if more accurate speech recognition engines have a higher power consumption than less accurate ones.

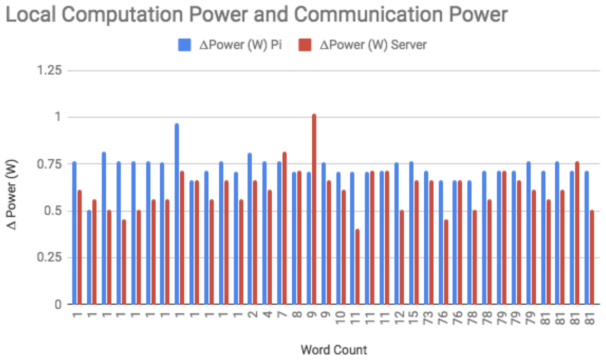
III. Results

The following graphs represent our results from running the local processing experiment and the server processing experiment.

We found that running speech recognition locally took more time, power, and energy than running speech recognition on a server.

In Figures 2-4, the horizontal axis represents each audio sample we tested. The label corresponds to the word count of the sample. The blue bars represent the Pi and the red bars represent the server.

Fig. 2. Power Used in Local vs. Server computation



From Figure 2, we can see that in both cases, the power used was somewhat comparable, but slightly higher in the Pi. It is important to note that the power represented by the server is the power used to communicate the audio samples to the server and the results back to the Pi. The Pi used very similar amount of power to communicate as to compute, which means either the SCP protocol is an expensive operation, or the speech recognition engine is extremely lightweight and power efficient.

From Figure 3 however, we can see that the energy used was much higher for the Pi than the server. This is due to the fact that energy is dependent on both power and time, and on

Fig. 3. Energy used in local vs. server computation

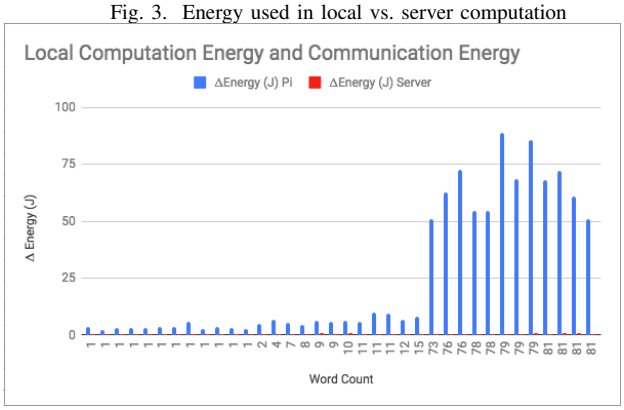
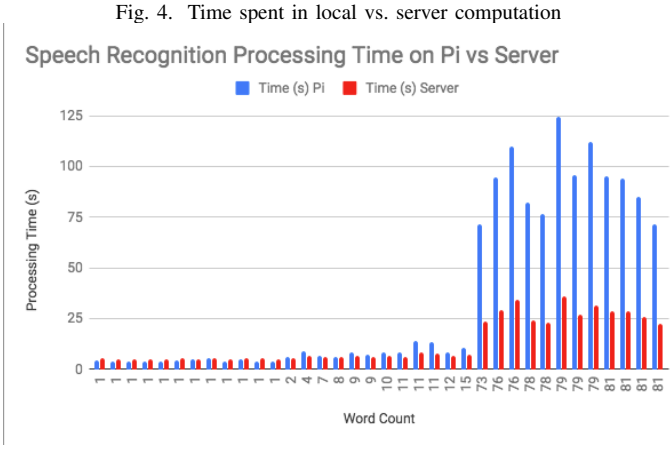


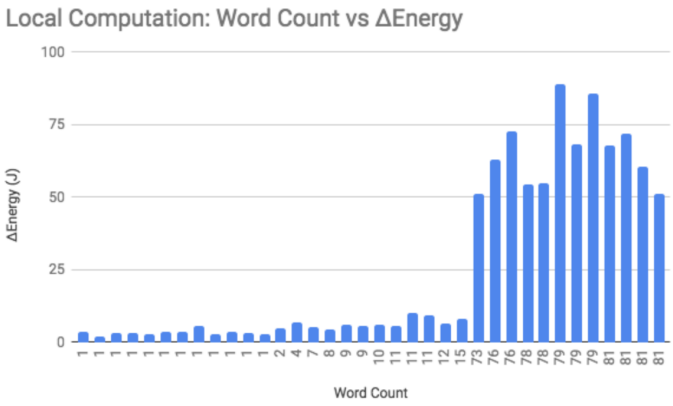
Fig. 4. Time spent in local vs. server computation



the Pi, the processing took much longer than the server, as we can see in Figure 4.

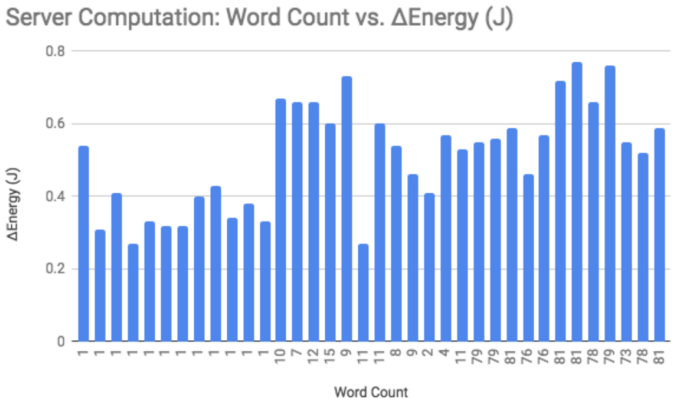
Figure 4 shows that the time spent for speech recognition was higher on the Pi, and that the difference increased as the audio sample size increased. For small word sizes, we can see that the computation took almost the same time on both the Pi and the server.

Fig. 5. Word Count vs. Energy for local computation



In Figure 5 and 6, we can see the relationship between energy expended and word count for local and server com-

Fig. 6. Word Count vs. Energy for server computation



putation. We can see that the energy increases with word count for local computation, but doesn't change much for the communication cost of server computation. This suggests that while energy cost for computation corresponds to the length of the sample, the energy cost for communication doesn't increase as acutely.

Fig. 7. Average comparisons

	Small Samples	Medium Samples	Large Samples
Ratio of Computation to Communicaton Energy	9.23	11.89	108.3
Ratio of Computation to Communication Power	1.29	1.1	1.17
Ratio of Raspberry Pi Processing to Server Processing Time	0.84	1.33	3.33

Figure 7 shows average ratios of energy, power, and time across the different sample sizes. It's clear that computation is inferior to communication across most benchmarks.

We also measured the accuracy of the CMU PocketSphinx Library using a Python script we created to measure the Word Error Rate. We found that 3 out of the 10 small one word audio samples were recognized correctly. There was an average of a 49% word error rate for medium samples, and an average of a 58% word error rate for large samples. Note that higher accuracy corresponds to a lower word error rate. In order to calculate word error rate, we wrote a Python script that uses dynamic programming principles to determine the error rate based on the number of word substitutions, deletions, and insertions in the sample.

IV. Analysis

In conclusion, local computation of speech recognition for the Raspberry Pi is much less energy efficient compared to the communication costs of moving the audio to a server of off-board computation. Computation on the server also was faster than local computation. This was true across different audio sample sizes.

For very small sentences, however, since the amount of energy for local computation and power was approximately the same on the server and the pi, it may not be as efficient to process the data externally, since hosting a server to process data is an additional incurred cost. For one word, doing the processing on the board was faster than sending it off the board. However, since most speech recognition applications do not simply require the processing of one word, and instead require processing of large sentences with more complex vocabularies, using a server to offload processing makes sense for most voice-controlled devies.

We observed low accuracy of speech recognition which we attribute to the PocketSphinx algorithm. Because it was built for use on mobile and embedded systems, it was not as accurate as other speech recognition programs. While using a more computationally intensive speech recognition program may have lead to higher accuracy, we believe it would have only increased the difference in energy spent between computation and communication.

V. Milestones and Goals

The goal of this project is to analyze how energy and processing performance vary when data processing occurs on an embedded device and when it occurs on an external server, specifically related to speech recognition applications.

This project was broken up into a few milestones in order to accomplish this goal. We were able to complete each milestone.

- Determine exact tools, algorithm, and server
- Run speech recognition on the board
- Run speech recognition on a server
- Compare energy usage of each method

A. Milestone One

The first milestone is to determine the exact necessary tools and resources. We have outlined some options for materials and methods above. We selected the speech recognition computation and installed the necessary tools on both the Raspberry Pi and the server we plan to offload computation to.

B. Milestone Two

The next milestone is to run the computation on the board. This can be done by downloading the speech computation software onto the Raspberry Pi. Then, we can play a series of speech samples and observe the results to confirm that it works.

C. Milestone Three

The third milestone is to run the computation on a server. To do this, we must set up a server and be able to communicate to it with the Raspberry Pi. Then, we must place the speech recognition software on the server and write a script such that it takes in audio data from the Raspberry Pi and sends back the results of the speech recognition.

D. Milestone Four

The fourth milestone is to measure the energy usage of each method. This can be done by using one of the power measurement tool we described above and measuring time in our script.

While we were able to accomplish all of the goals that we set for this assignment, we were unable to achieve some of

the reach goals we had originally outlined in our plan. Some of those included comparing the performance of different speech recognition engines and determining performance of more intensive speech samples with larger vocabulary sizes or different languages. However, the nature of this project is portable to a different speech recognition engines, as the part of the program that calls the library simply needs to call a different one to test. Similarly, the program can work on any type of audio samples, so in order to test on more complicated samples, we would simply need to acquire those samples and run the same program on them.

VI. Future Work

There is still much work that can be done to further investigate the topic of comparing running speech recognition locally on an IoT device and offloading the computation to a server. Different kinds of hardware, servers, and algorithms could be tested. This could greatly affect the on-board computation cost as well as latency and accuracy. As mentioned, a larger sample set could be used in order to get more accurate data. Based on time restraints we only tested 30 samples. Trying to find more accurate methods of power measurement would also be an area of improvement. Although the USB power meter gave us rough measurements, there are probably more granular tools that can be used.

VII. Division of Work

Namrita set up the Raspberry Pi, and was responsible for writing the program to run speech recognition locally. Marie was responsible for writing the server-side program to communicate data from the Pi and run speech recognition on the server. She also wrote the script for determining the word error rate. We both collaborated on the contents of the main script and running the experiments.

VIII. Website Link

https://github.com/mnshaw/iot-computation-vs-communication

References

[1] https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

[2] https://techcrunch.com/2018/03/07/47-3-million-u-s-adults-have-access-to-a-smart-speaker-report-says/

[3] https://wolfpaulus.com/embedded/raspberrypi2-sr/

[4] https://oscarliang.com/raspberry-pi-voice-recognition-works-like-siri/

[5] https://ieeexplore.ieee.org/abstract/document/7472671

[6] https://mediatum.ub.tum.de/doc/1399984/1399984.pdf

[7] Bekaroo, Girish Santokhee, Aditya. (2016). Power consumption of the Raspberry Pi: A comparative analysis. 361-366. 10.1109/EmergiTech.2016.7737367.