École Polytechnique Fédérale de Lausanne

Smallworld V2 : Adding Apple Connectivity and Evaluating Recommender Application

by André Ramon Zarza Tapia

Bachelor Project Report

Professor Anne-Marie Kermarrec
Project Advisor

Dr. Erick Lavoie
Project Supervisor

EPFL IC IINFCOM SaCS
BC Building, Office BC 347
CH-1015 Lausanne
Switzerland

January 7, 2022

Contents

Chapter 1

Introduction

Today's Internet connectivity relies heavily on a centralised infrastructure when direct connectivity between devices would be sufficient for tasks such as exchanging files or collaborating on shared documents. Romain Küenzi explores this idea in the Smallworld V1 project by developing a decentralised local network using the small and affordable *Raspberry Pi's Zero W*. With these he was able to develop a small local network in which the RPis could communicate by using the peer-to-peer gossiping protocol SSB and allow users to transfer data. Additionally, instructions for setting up Ethernet-over-USB on Linux, Windows and Android based devices were given but not for both MacOS and iOS devices.

To ensure that the use of a local network through peer-to-peer communication by using RPi's is a viable replacement of centralised infrastructures, all popular OS Environments should be able to use it. As both MacOS and iOS devices were not yet supported in Romain's setup, I added configurations to the RPi's and user devices as explained in the instructions of chapter 2. After following these instruction you should be able to SSH into the RPi through Ethernet-over-USB connectivity from both MacOS devices and iPhones. This ensures that all popular OS Environments are supported by the infrastructure developed by Romain.

In Smallworld V2 I evaluate the performance and energy consumption of different models of the Raspberry Pi with the aim of better understanding the economics of building peer-to-peer applications with these devices. For this I acquired the newer, more powerful yet still affordable Raspberry Pi 4 model and replicated the process of building a local network as done in the Smallworld V1 project. As well as the Raspberry Pi 4 model similar tests were ran on the Raspberry Pi 3B+ model which had Wi-Fi connectivity but did not offer the Ethernet-over-USB connectivity option as the other two models do.

The origin of this research project came about with the idea that over the past few decades, people have introduced various personal computers such as laptops, smartphones, smart tablets and so on into their everyday lives[7]. Numbers for mobile devices have recently been shown to even outnumber the people on earth with no signs of slowing down in the years to come. This sets

place for a growing concern that all these devices make use of vasts amount of energy and power and have shifted from the low-carbon enablers they used to be to power drainers. Therefore, I thought it interesting to investigate the power consumption and efficiency of low cost, credit card sized computers when it comes to popular application such as that of a recommender system.

To evaluate and better understand the economics of building peer-to-peer infrastructures and their real world usage capabilities, I used a movie recommendation application. The reason for using a recommendation application is that in the world of today, recommendation systems are highly popular in different industries such as e-commerce, media, banking, telecom and many more[5]. The importance and relevance of recommendation technologies has lead industry to increase its use of resources in order to process and deploy them. This is why the performance and energy usage for the Raspberry Pi's is tested with a recommender and predictor systems for movies in the MovieLens dataset. To evaluate the scalability of using different Raspberry Pi models for recommender systems the MovieLens datasets of 100 thousand, 1 million, 10 million, and 20 million ratings will be used.

Chapter 2

Adding MacOS and iPhone connectivity

In this chapter, I explain and demonstrate how to add MacOS and iPhone Ethernet-over-USB connectivity to the infrastructure developed by Romain in the Smallworld V1 project. By following these instructions[1] as an appendix to the work developed in Smallworld V1, you should be able to connect either the Raspberry Pi 4 or Zero to your MacBook or iPhone through SSH and share your internet connection to the RPi device.

2.1 Mac OS

Connecting the Raspberry Pi to MacOS devices is possible through simple configurations on both devices. The following are the instructions you must follow and do on the Raspberry Pi itself to allow connectivity through the USB interface on either the Raspberry Pi Zero or 4.

1. Ensure that Subsection 4.3.1 from the Smallworld V1 report is done to enable Ethernet-over-USB on the Raspberry Pi.
2. Change the file `/etc/dhcp.conf` on the Raspberry Pi by appending the following:

   ```
   profile usb0
   ip_address=192.168.6.1/24
   static routers=192.168.6.1
   ```

   This is the configuration of the DHCP client for the Ethernet-over-USB. It uses network interface usb0, provides the Raspberry Pi with IP address 192.168.6.1 and searches network for IP's with netmask 192.168.6.x.
3. Change the file `/etc/dhcpcd.conf` on the Raspberry Pi by appending the following:

[1] These instruction have been added as Appendix B in the Smallworld V1 project.

```
profile host_usb0
static ip_address=192.168.6.1/24
static routers=192.168.6.1

interface usb0
fallback host_usb0
```

After running the following commands on the Raspberry Pi one can proceed to connect the MacOS device to the raspberry pi by the USB-C port, hence also powering the Raspberry Pi 4. For the Raspberry Pi Zero, one can connect it through the USB port labeled USB which is connected to the VBUS and hence this connection can be also power the Raspberry Pi Zero. This connections are the simplest as they requires only one cable to power and tether the Raspberry Pi to the Mac. On the MacOS device the set up is fairly simple as no drivers must be downloaded or files changed in order for the connection to work[2].

1. First you must enable Wi-Fi sharing[3] through RNDIS/Ethernet.
   (a) Go to *System Preferences*.
   (b) Go to *Sharing*.
   (c) Under *Share your connection from:* select the *Wi-Fi* option from the drop down menu.
   (d) Then under *To computers using:* select only the *RNDIS/Ethernet Gadget* option.
   (e) Finally in the left side of the window under *Service* turn on the *Internet Sharing* option and a pop-should appear. The pop-up asks *Are you sure you want to turn on internet sharing?*, simply click on the start button on the pop-up.
2. Second you must configure the *RNDIS/Ethernet Gadget*. The Remote Network Driver Interface Specification (RNDIS) is a Microsoft proprietary protocol used mostly on top of USB connections. This provides a virtual Ethernet link to devices using Windows, Linux and FreeBSD operating systems.
   (a) Go to *System Preferences*.
   (b) Go to *Network*.
   (c) On the left side of the window select the *RNDIS/Ethernet Gadget* option and under *Configure IPv4* select *Using DHCP* from the drop down menu.

After completing these steps you should be able to connect your Mac Os laptop to the Raspberry Pi via SSH using the terminal and the following command `ssh pi@raspberrypi.local` and inputing the password `raspberry`.

[2] Images for these instructions can be found in Appendix A

[3] Some networks are protected and will not allows you to share your Wi-Fi connection through any means. This is the case with the EPFL network.

2.2 iPhone

To enable tethering from an iPhone device with capability of providing its own personal hotspot, some minor configurations[4] have to be made on both the Raspberry Pi and the iPhone.

1. On the Raspberry Pi it is sufficient to copy and paste these following terminal commands to enable iPhone tethering.

```
$ sudo apt-get install usbmuxd
$ sudo apt-get install libimobiledevice-utils
$ sudo apt-get install ipheth-utils
$ sudo apt-get install gvfs gvfs-backends gvfs-bin gvfs-fuse
$ sudo apt-get install openssh-server
```

   Definition of Packages[1]

   (a) *usbmuxd* : USB multiplexor daemon for iPhone and iPod Touch devices.

   (b) *ipheth-utils* : USB tethering driver support utilities for the iPhone.

   (c) *libimobiledevice-utils* : Utitilies for communicating with iPhone and other Apple devices.

   (d) *gvfs* : Userspace virtual filesystem where mounts run as separate processes which you talk to via D-Bus.

   (e) *gvfs-backends* : Userspace virtual filesystem - backends

   (f) *gvfs-fuse* : Userspace virtual filesystem - fuse server.

   (g) *gvfs-bin* : Userspace virtual filesystem - deprecated command-line tools.

   (h) *openssh-server* : OpenSSH is a powerful collection of tools for the remote control of, and transfer of data between, networked computers.

After running the following commands on the Raspberry Pi you can proceed to connect the iOS device via an USB-A to lightning port cable. After configuration on the raspberry pi is finished and you can proceed to enabling the connection from the iPhone device as explained below.

1. The first time the iPhone is connected to the Raspberry Pi, a pop-up message will appear asking "*Trust This Computer*?" which will require the following.

   (a) Click on the "*Trust*" button.

   (b) Input the device's 6 or 4 digit passcode set to unlock the iPhone when prompted to do so.

[4]The instructions for iPhone tethering can be found in the following page : `https://github.com/inikolaev/iphone-raspberry-pi`.

2. Once the passcode has been correctly put into the iPhone the next step is enabling the discovery of the device's hotspot by other devices. To enable it, follow these steps :
   (a) Go to *Settings*.
   (b) Go to *Personal Hotspot*
   (c) Turn on "*Allow Others to Join*"

After these steps are succesful the connection between the Raspberry Pi and the iPhone should work. This is indicated by a *blue* bubble with the hotspot symbol at the left top corner of the device. Once the previous configurations have been made on both the Raspberry pi and the iPhone, an SSH connection should be possible from the iPhone to the Raspberry Pi. Find the Raspberry Pi's IP address using the app *Net Analyzer* from the App Store and following these steps:

1. Open *Net Analyzer*.
2. Click on the *LAN* section at the bottom and then press *Scan* on the top right corner. This should show the IP of the iOS device and a second IP belonging to the Raspberry Pi.
3. Note down the IP address of the Raspberry Pi.
4. To check connection, a *Ping* option is available on the app. Input the IP of the Raspberry Pi in the *Domain Name/IP Adress* box and then *Start* at the top right corner. If the devices are connected then it should show a sequence numbers with a *green* dot next to them. If the dot is *grey* then connection failed.

After ensuring that a connection between both devices exists sshing is possible through the app *WebSSH* found on the App Store. To ssh to the Raspberry Pi you can do so by following these steps :

1. Click on the square with a plus icon at the top right corner.
2. Enter the following information:
   (a) Under *Host* enter the Raspberry Pi's IP address.
   (b) Under *User* enter *pi*.
   (c) Under *Password* enter *raspberry*.
3. Click on the check mark at the top right to validate the host's details.
4. A box should now appear with the following written on it *pi@XXX.XXX.XX.XX* where the *X*'s represent the Raspberry Pi's IP address.

5. Finally click on the box and the first time some pop-ups should appear however, press *continue* and now you should see a terminal indicating sshing into the Raspberry Pi worked.

After successfully following these instructions the possibility of connecting to the Raspberry Pi through the OTG port should be possible and all the popular operating systems would have been supported in Romain's original setup.

Chapter 3

Recommendation System

To evaluate the performance and economics of using Raspberry Pi's for peer-to-peer infrastructures, I used a movie recommender and ratings predictor system. The idea came about from the Milestone 1 project given in the course Systems for Data Science where a recommender system is implemented using the Movielens dataset[1]. The MovieLens datasets are datasets created by the GroupLens research lab in the Department of Computer Science and Engineering at the University of Minnesota. These datasets, first released in 1998, describe people's expressed preferences for movies. These preferences take the form of tuples, each the result of a person expressing a preference (a 0-5 star rating) for a movie at a particular time[3]. In this project a baseline for comparing different models of Rapsberry Pis as well as my own laptop will be created using the 100 thousand ratings dataset. This dataset comprises 100,000 ratings from 943 users on 1,682 movies. Every user hasn't rated every single movie but has rated at least 20 different movies, and each movie has been rated by at least one user. The reason for using this dataset as a baseline for the experiments is that it is a rather small dataset compared to the RAM available in the Raspberry Pis especially the Zero[2].

Recommender systems can be built using two different approaches, either a Content-based or a Collaborative Filtering approach to predict a user's preference for a set of items based on the user's historical behaviour. The Content-based approach uses information of the items' own features such as genre, director, origin and so on whereas the Collaborative Filtering approach is solely based on user's historical preferences[4]. In this project the recommender system will be built using the standard method of Collaborative Filtering known as the Nearest Neighbourhood algorithm. This method uses a measure of similarity between objects (in our case users) which then can be used to determine which users are most similar to a given user with respect to the movies they have rated. Once the top k most similar users are found for a specific user the system uses a weighted average of ratings from these k users with similarities as weights[6]. This recommendation application makes the assumption that users are similar if they have rated a similar

[1]The movielense datasets can be found here https://grouplens.org/datasets/movielens/
[2]The Raspberry Pi Zero has 512 MB of RAM.

set of movies as users that are alike would have watched similar movies. However, calculating similarities for large datasets can be computationally expensive hence, the recommendation system for this project is implemented in two different ways to be able to accommodate the small RAM size of the Raspberry Pi Zero (512MB) and explore the performance and economics of an RPi's infrastructure. These two implementations can be found in the script `ratings.py`[3] and `ratings_blocking.py`[4].

3.1 Recommendation Algorithm

In order to make recommendations and rating predictions the system uses a user-based Collaborative Filtering approach using the cosine similarity metric. The MovieLens ratings dataset can be represented as matrix $R_{u,f}$ where u is the number of users in the dataset, f the number films and $r_{i,j}$ is the rating given for film j given by user i. In the dataset rating $r_{i,j}$ is a value between $[1,5]$ and 0 in the case for which the user hasn't seen film j. With this matrix R we can then calculate the cosine similarity between user i and i' as follows:

$$cos\_similarity(i,i') = \frac{\sum_{j=1}^{f} i_j i'_j}{\sqrt{\sum_{j=1}^{f} i_j^2}\sqrt{\sum_{j=1}^{f} {i'_j}^2}}$$

where i_j and i'_j are components of vectors i and i' respectively of size f. These vectors represent the set of ratings that users i and i' have given and where i_j and i'_j takes a value between $[1,5]$ or 0 if that film has not been seen by the user.

The similarity calculation between all users can be achieved through a simple matrix-matrix multiplication. The matrix one would have to multiply is the normalised matrix of all ratings values in matrix R. This is to say that matrix R' would be matrix R but every rating value is divided by the square root of the sum of square vector components in an user row. Then similarity matrix S can be calculated by multiplying R' by its transpose resulting in matrix $S_{u,u}$ where value $s_{i,i'}$ represents the similarity value between users i and i'. This similarity is a value that ranges from 0 meaning that users are completely different to 1 meaning that users are identical. Once this similarity between users is calculated it is a matter of finding the top k users most similar to a given user by sorting the values in descending order.

The distances and indices of the top k neighbours for any given user are calculated using the

[3]This script is a slight adaptation of this script `https://github.com/mbodenham/k-nn-movie-recommender/blob/master/recommender.py`

[4]Both of the scripts can be found `https://gitlab.epfl.ch/azarza/recommender_evaluation`

sklearn.neighbors library and its methods. This library also calculates the similarities of all users hence similarity matrix S. Both distances and indices are represented as ndarrays which can be computationally costly hence in the second script the similarity matrix is calculated through the use of the more RAM efficient sparse matrices.

For a given user i the script creates a dictionary with the movies that all top k users for i have seen along with their ratings and the distance for that user to i. Afterwards, a set of the movies that user i hasn't seen but that at least 5 top k neighbours have is created and their predicted ratings are calculated. The ratings $p_{i,j}$ for film j that user i hasn't seen is predicted with the following formula.

$$p_{i,j} = \frac{\sum_{i' \in U}^{n} r_{i',j} * s_{i',i}}{\sum_{i' \in U}^{n} s_{i',i}}$$

where U is the set of nearest neighbours to user i that have seen film j and $s_{i',i}$ the similarity value between user i and user i'. Once the ratings for the films that user i hasn't seen are calculated the films are sorted in decreasing order of predicted rating and the top 10 are given as a recommendation to user i. In this way user-based Collaborative Filtering using the cosine similarity is achieved. However, the calculation for the matrix-matrix multiplication is expensive as with large dataset the matrix can become a square matrix of size $(\textit{number of users})^2$ that can create memory leaks in the Raspberry Pi's with a lower amount of RAM. Therefore an adaptation to the way in which matrix S was calculated is shown in `ratings_blocking.py` in order to handle the scaling factor of these recommender systems.

3.2 Computation By Blocks

In order to be able to fit the similarity matrix S into the RAM of the Raspberry Pi's, it had to be calculated in blocks. Another thing to consider in this instance is that ratings matrix R retrieved from the ratings data can be very big. For instance the 1 million dataset ratings matrix would be a $(3900, 6040)$ matrix with 23,556,000 ratings of which most are 0 since this set only contains 1 million ratings. Therefore, matrix R is treated a sparse matrix since sparse matrices only store the non-zero values of a dense matrix.

In python sparse matrices have a wide range choice of linear algebra functions that they can use to do operation similar to broadcasting with regular numpy arrays, hence making their calculation somewhat as efficient as normal dense matrices. Thus the calculation of the normalised ratings matrix is not impacted and in fact requires less RAM as I have decided to now represent matrix R' as a sparse matrix. Since matrix S is a result of a matrix-matrix multiplication

of R', a block of size n rows is taken from R' and multiplied by the transpose of R' resulting in similarity matrix Sn, u which is now much smaller than before if n is chosen to be small. Then matrix S is used to find the top k neighbours for the given n users and they are stored in matrix kNN of size (u, k). This method is used for scaling purposes and to evaluate the efficiency of using the Raspberry Pi's with larger datasets.

Chapter 4

Evaluation

In this chapter I evaluate the performance of various Raspberry Pi models as well as my own personal laptop with the recommender application introduced in the previous chapter. I also explore the feasibility of scaling such infrastructures and system by evaluating the recommender application with larger datasets and analysing their performance.

Raspberry Pis are single-board, affordable computers created by the UK-based charity Raspberry Pi Foundation whose aim is to put the power of computing and digital making into the hands of people all over the world[2]. These single-board computers can be bought at a price ranging from $10 for the smaller Zero models to $35 for the more powerful, larger 4 model[1]. The reasons for choosing Raspberry Pi models as stated in Romain's report are that, the Raspberry Pi community is very large with vasts amount of information and resources easily available to anyone that requires them. These boards also use a common operating system, Debian which gives one the standard tools any computer running a Linux distribution would have. And finally, as demonstrated in Smallworld V1 these boards posses the simplicity requirements to be used a dedicated device for direct connectivity and to build local networks. Thus, making them perfect candidates for testing their efficiency and power consumption when dealing with recommender systems.

4.1 Devices Under Test

The first model tested is the Raspberry Pi Zero W model. This is the most affordable model from the available Raspberry Pis coming in at just $10. It is also the smallest model in terms of size as well as RAM which stands at just 512MB. It has the single-core chip BCM2835 which clocks at a speed of 1GHz. This model also has a dedicated OTG USB port which can be used for

[1]References and specification for the Raspberry can be found on the Raspberry Pi website at `https://www.raspberrypi.org/`

Ethernet-over-USB as explained in chapter 2. This was the original model used in the Smallworld V1 project to deploy a local network, as it fit all the simplicity and affordability requirements for that project. In this project it is of interest as it can be considered a viable option to build scalable recommender systems on and as it is also the model that requires the least power as will be demonstrated below.

The second model test is the Raspberry Pi 3B+ model. This model comes in at about a price of $30 dollars as it is significantly larger than the Raspberry Pi Zero and offers more connectivity interfaces as well as slightly more computational power. This board has the quad-core processor Broadcom BCM2837B0 which clocks at a speed of 1.4 GHz, as well as 1GB of LPDDR2 SDRAM. It also offers Wi-Fi connectivity making it a viable option for the local network developed in Smallworld V1. This model being slightly bigger and offering more connectivity interfaces lacks an OTG USB dedicated port hence Ethernet-over-USB is not a possibility but it does posses an Ethernet dedicated port. It is interesting to test the efficiency of this board as it has more computational power than the previous model as well as faster and larger RAM and is a middle option between the two other boards being tested.

The final model tested is the Raspberry Pi 4 model. This model is the more expensive of the 3 coming in at about $35 but it is also considerably the most powerful. It has the quad-core Broadcom BCM2711 processor which clocks in at a speed of 1.5GHz as well as 8GBs of LPDDR4 RAM. The Raspberry pi 4 is also available with 1,2 or 4 GBs of LPDDR4 RAM. In this project we went for the largest available RAM as it was still an affordable option which would offer much more computational power than the other available options and would allow for the possibility of scaling.

All these models were then compared to my personal computer as well as it is the device I use for everyday tasks. My computer is a 2020 Mac Book Pro with a quad-core Intel Core i7 processor clocking in at 2.3GHz and with 32GB of LPDDR4X RAM. This is a relatively powerful computer with respect to most personal laptops but it is used as a mean of determining my own energy consumption and to compare against what a much more affordable smaller computer can do.

4.2 Methdology

In this project I tested the devices with the aim of answering the following research questions:

1. How efficient with respect to computation time and memory usage are the RPi's?
2. How much power is consumed by the recommender application functionality on the RPi platforms?

The power consumption and computational efficiency of these devices is important to study as different devices use battery and resources differently and in manners that could lead to

more sustainable computing. To study this, the devices were tested with the python script introduced in chapter 3 which implements a recommender and ratings predictor system for the MovieLens datasets. Within this script various measurements were introduced in order to determine computation time for various part of the system as well as memory consumption of the entire script. For power consumption measurements an external device called *PowersSpy 2* was used.

In the following subsections, this project answers question 1 through to 2 in order to reveal some insightful information on power and resource efficiency of RPi's in the context of recommender systems. The measurement methodology for each section is introduced within its respective section.

4.3 Computation Time

Computation Time is a measure of interest when dealing with recommender systems as it aims to service clients as efficiently and quickly as possible with recommendations on whatever item it is the user desires. Computation time takes into account multiple computation factors when it comes the recommendation system as this system must be trained in order to make the best possible recommendation. In this project different parts of the script were measured in order to determine the efficiency of the different Raspberry Pi models with respect to different tasks performed by the script.

Firstly, the script had to load the datasets to be able to process the data within them and be able to make recommendations. The datasets used throughout this test was the MovieLens 100 thousand ratings dataset introduced in chapter 3. The size of this dataset is of 1'979'173 bytes or about 2Mb. Two different tests were carried out in order to determine computational time. The first one was when a recommendation for a single user was being done in which the computational time measurements done are as follows[2] :

1. Time in seconds required to load the dataset as a *pandas dataframe*.
2. Time in seconds required to fit the kNN model on the given data.
3. Time in seconds to do predictions and recommendation for a given user A once the data had been loaded and the kNN model was fitted to the data.
4. Time in seconds taken to execute the entirety of the program.

The second test done was to determine the throughput of the devices by making recommendations for 100 users. The script would load the data and fit it similarly to the first test and from

[2]The information on how to run these tests can be found in the README.md file in gitlab of this project `https://gitlab.epfl.ch/azarza/decentralized-knn`

there it would proceed to do recommendation and ratings predictions for 100 randomised users. This test would measure:

1. Time in seconds taken to make recommendations and predict ratings for each of the 100 users resulting in 100 outputs of time.
2. The maximum time taken for a single user amongst the 100 users.
3. The minimum time taken for a single user amongst the 100 users.
4. The average time taken for a single user amongst the 100 users.

With this information we would be able to evaluate the efficiency of the devices with respect to making multiple consecutive recommendations. It serves the purpose of determining how good these devices would be in a distributed scenario where they would act as a dedicated device for running recommendation applications on. Instead of running single recommendations on the personal devices they could be done by a dedicated device such as the RPi's and then using the local network developed in the Smallworld V1 project, these recommendation could be distributed to each device it was meant to.

4.3.1 Results for the Raspberry Pi Zero model

Results for the first test :

Load Dataset (s)	Fit kNN (s)	Predict & Recommend (s)	Execute Program (s)
19.9965159893035	0.760776042938232	3.55016279220581	24.3916101455688

The result for the second test were the recommendation for 100 users is calculated is :

Total Time (s)	Maximum Time (s)	Minimum Time (s)	Average Time (s)
309.48774051666200	4.76732635498046	2.18227982521057	3.09487740516662

4.3.2 Results for the Raspberry Pi 3B+ model

Results for the first test :

Load Dataset (s)	Fit kNN (s)	Predict & Recommend (s)	Execute Program (s)
3.694392681121826	0.18730473518371582	0.787654161453247	4.687851905822754

The result for the second test were the recommendation for 100 users is calculated is :

Total Time (s)	Maximum Time (s)	Minimum Time (s)	Average Time (s)
75.213702678680400	1.052947521209710	0.609214544296264	0.752137026786804

4.3.3 Results for the Raspberry Pi 4 model

Results for the first test :

Load Dataset (s)	Fit kNN (s)	Predict & Recommend (s)	Execute Program (s)
1.6729826927185059	0.10517549514770508	0.335757970809936	2.1360373497009277

The result for the second test were the recommendation for 100 users is calculated is :

Total Time (s)	Maximum Time (s)	Minimum Time (s)	Average Time (s)
30.428322076797500	0.448853969573974	0.233684301376342	0.304283220767975

4.3.4 Results for the MacBook Pro

Results for the first test :

Load Dataset (s)	Fit kNN (s)	Predict & Recommend (s)	Execute Program (s)
0.34023022651672363	0.0013587474822998047	0.10560011863708496	0.4493403434753418

The result for the second test were the recommendation for 100 users is calculated is :

Total Time (s)	Maximum Time (s)	Minimum Time (s)	Average Time (s)
10.757601499557500	0.115493059158325	0.102632999420166	0.107576014995575

4.3.5 Results Analysis

From these results one can see that in the process of the recommender system what requires the most time is loading the dataset into memory in order to process it. This can be attributed to the size of the dataset and the efficiency of the pandas module's libraries to process and read data from files. The other parts of the script are rather fast in comparison, with fitting the model coming in as the fastest followed by making a recommendation of 10 films with the highest predicted ratings. This results was constant among all devices however the variance in time to execute these tasks between devices is rather large.

The slowest of them is the Raspberry Pi Zero as expected which is about 5 times slower than the slightly more powerful Raspberry Pi 3B+ model, about 12 times slower than the most powerful of the range; the Raspberry Pi 4 model and about 50 times slower than the MacBook Pro.

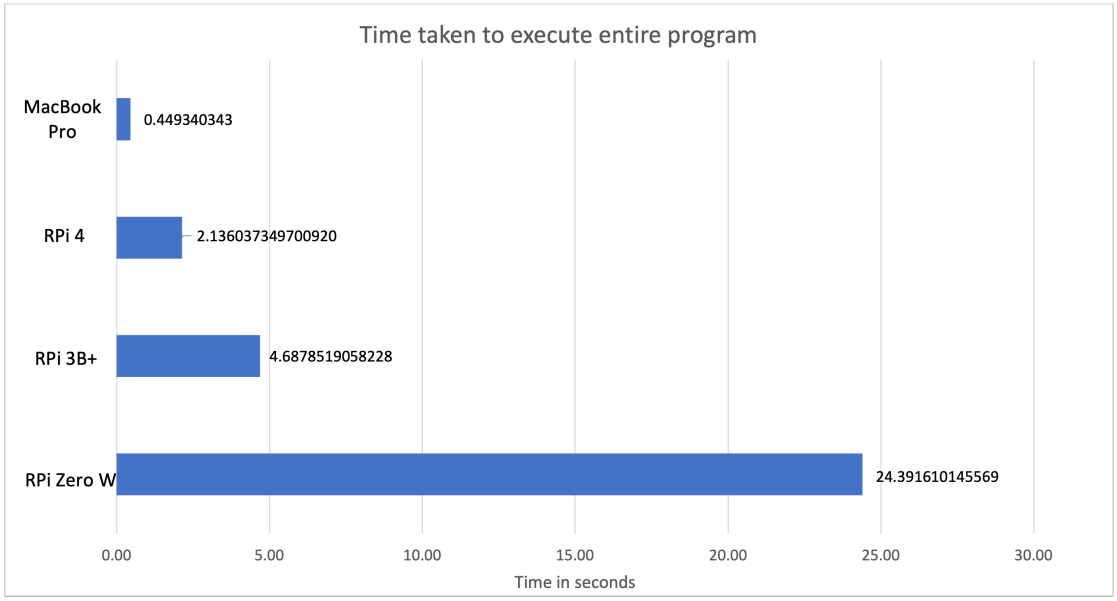


Figure 4.1: Time taken to execute entire program

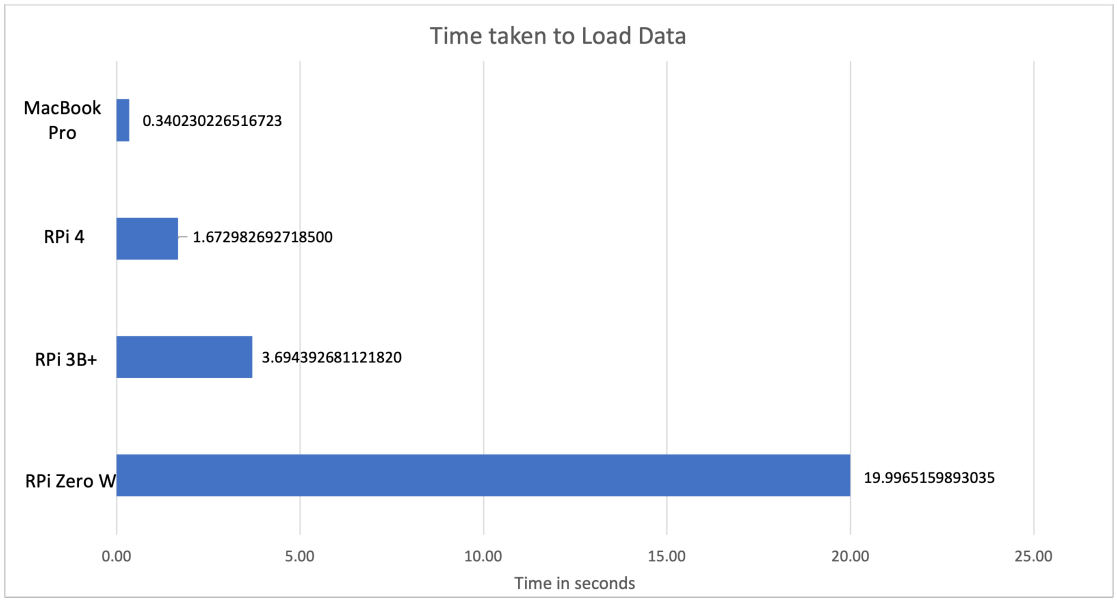


Figure 4.2: Time taken to load dataset

For the more powerful of the Raspberry Pi's, the model 4 it shows that is only about 5 times slower than a MacBook pro with much more computational power than the raspberry Pi 4. However, with respect to the other models the Raspberry Pi 4 model performed about 2 times better than its closest competitor with 8 times less RAM and slightly worse processor, Raspberry Pi 3b+ and about 12 times better than the smallest, least powerful Raspberry Pi Zero W. Overall, in terms of computational efficiency by these computer one can quickly see how a Raspberry Pi 4 with 8GB of RAM is capable of out performing its competitors.

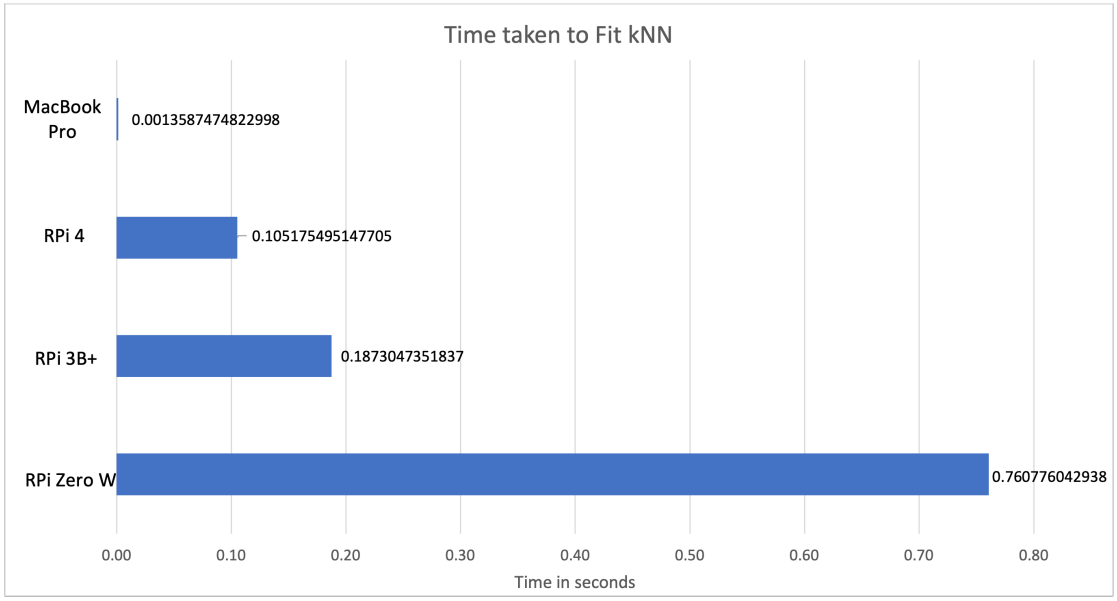


Figure 4.3: Time taken to fit kNN model

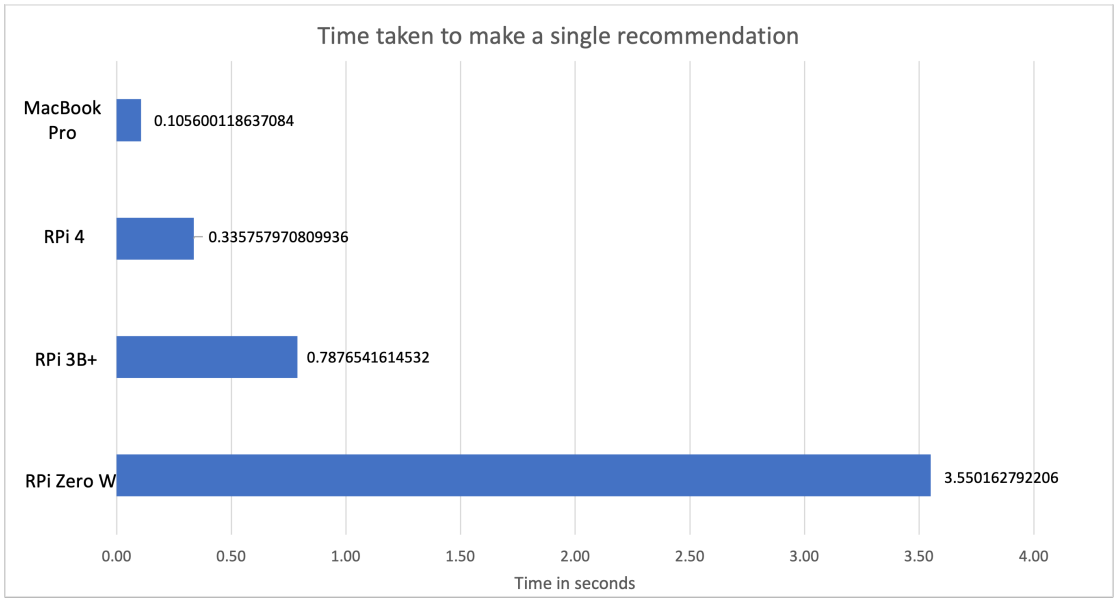


Figure 4.4: Time taken to make single recommendation

For the second computational time test the results showed that for the Raspberry Pi Zero W one can expect a large deviation between results in comparison with all other devices. As it can be seen the maximum computed time to make a recommendation for a single user was of 4.76732635498046 seconds whereas the minimum time was of 2.18227982521057 seconds which results in a spread of about 2.585 seconds. This would give enough time to make two prediction in the space of one prediction making it highly inefficient. For the other devices the spread was of 0.444 seconds and 0.215 which is considerably smaller when taking into account the average

and minimum time taken for a single recommendation.

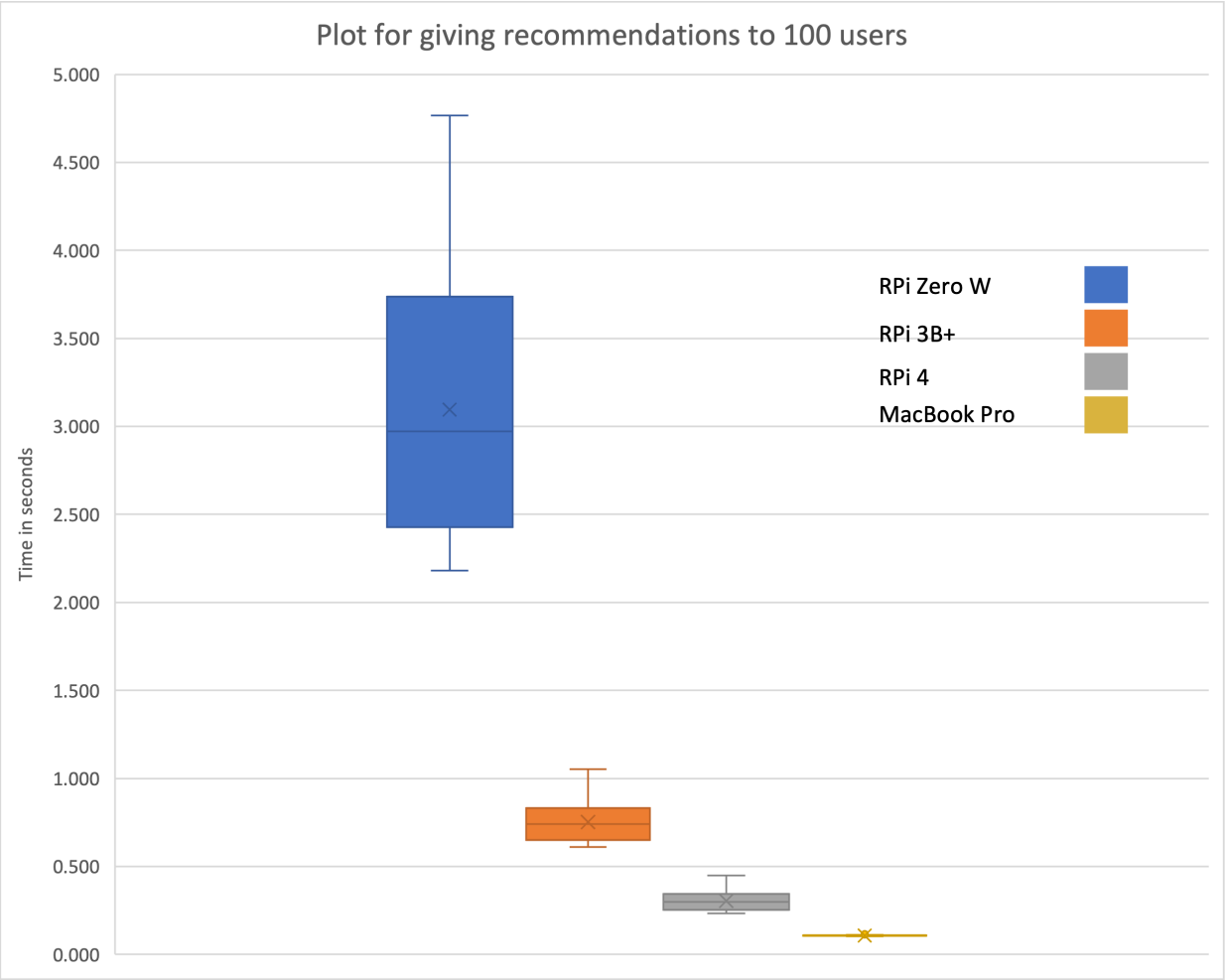


Figure 4.5: Throughput test results.

Another thing to take into consideration is the time taken to make a recommendation with respect to the time taken to fit the model and load the data. This is because such recommender systems can be built in such a way that wouldn't require loading the data every time a recommendation were to be done. For multiple users, their similarities with respect to every other users is calculated once when the model is fitted. Therefore, the fitting of the model can be done periodically and dataset for users recommendations can be also updated periodically thus, the model would be fitted only when required. This would mean that the Raspberry Pi Zero is only about 10 times slower than its more powerful competitor the Raspberry Pi 4 when it comes to make single user predictions. Nevertheless for the sake of efficiency, the Raspberry Pi 4 proves to be the most efficient of the 3 models.

In addition tests were made where the dataset was to be retrieved from an external hard drive and see if that changed the speed at which data was being loaded as it is the process that requires the most computational time. However, these results showed negligible difference in loading

times so the possibility of retrieving data from and external hard drive to make the system more efficient was quickly disregarded.

4.4 Memory Usage

To test memory usage a similar test to the one above was ran where dedicated python libraries were used in order to determine resource usage by the script. The way in which memory usage was calculated was by seeing how much memory the process of running the script would require in gigabytes and converting it to a percentage over the total amount of available RAM. For this the dedicated profiling library *psutil* in python was used. The *psutil* library is a cross-platform library for retrieving information on running processes and their system utilisation.

Two different measures were made in order two analyse two different aspects of memory consumption by the recommender script. The first measure is a representation of relative memory consumption by each device and the second measure is an absolute measure of the memory used in gigabytes. The reason for studying these two measures is that the first one serves as an indicator into the capabilities of these systems to scale when it comes to processing larger datasets. The second measure is used to understand the memory management efficiency of each device.

4.4.1 Results

Here, the results for relative memory consumption in percentages with respect to available memory as well as absolute memory consumption are presented. The Raspberry Pi Zero W has a RAM of size 512 MB, the Raspberry Pi 3B+ models has 1GB of RAM, the Raspberry Pi 4 model has 8GB of RAM and the MacBook Pro has 32GB of RAM.

Relative memory consumption in percentages:

RPI Zero W	RPi 3B+	RPi4	MacBook Pro
20.30 %	10.17 %	1.27 %	0.49%

Absolute memory consumption in gigabytes:

RPI Zero W	RPi 3B+	RPi4	MacBook Pro
0.101478576660	0.101696014404	0.101325988769	0.14881515502929

One can see that with a dataset of 2MB as is the 100 thousand rating dataset the Raspberry Pi Zero has to be able to allocate $\frac{1}{5}$ of its available RAM just to be able to process the data and calculate the similarities between users. The Raspberry Pi 3B+ model with a 1GB RAM uses $\frac{1}{10}$ of

its memory when processing data for the recommender system and Raspberry Pi 4 only uses a little over $\frac{1}{100}$ of its available RAM. However, from the second table of results one can clearly see that all three Raspberry Pi devices allocate an equal amount of memory in terms of gigabyte when processing the `ratings.py` script. Therefore, in terms of memory and resource allocation the three Raspberry Pi's are very similar but when it comes to the possibility of scaling to much larger datasets one can see the limitation that the devices with lower amounts of RAM would have. The possibility of scaling will be explored as a quick extension of these results.

4.5 Energy Usage

As more mobile devices weave themselves into our every day lives there is a growing concern that these devices have shifted from the low-carbon enablers they used to be to power drainers. As energy efficiency has emerged as an important design requirements for modern computing it is essential to take it into consideration when evaluating the efficiency and economics of a system. Since the Raspberry Pi is a low-cost, small single-board computer it is interesting to see the power consumption of these different models under the load of creating a recommender system.The idea that came about when discussing the possibility of using dedicated devices for tasks such as recommending for multiple users was that these such devices could be most cost efficient in the long run by diminishing power consumption of these tasks.

In order to test the power consumption of these devices an external power meter was used. This power meter is that *PowerSpy2* device from the company *Alciom*. The *PowerSpy2* is an external device to the Raspberry Pi's which acts as a socket to which they can be plugged in and their power analysed in real time. This device connects to a computer on which an application gives out graph readings of different aspects of the device's power consumption. For this project we are only interested in the real-time power consumption aspect of the readings.

The setup for these tests was just to simply connect the Raspberry Pi to the *PowerSpy2* device and then connect to it through SSH as to not have any other peripherals connected to it which would consume more energy. The purpose of this was to concentrate the power consumption of the device solely on the recommender task at hand as SSH and Wi-fi connectivity require minimal to negligible power consumption. Once the setup was done the `ratings.py` script was ran and a screenshot of the power consumption for this task was taken. Alongside with the readings of running the recommender system, baseline readings were taken when the Raspberry Pi devices were idle and not running any processes.

4.5.1 Results for the Raspberry Pi Zero model

As can be seen in the above results the Raspberry Pi Zero has a power consumption of about 0.9W in an idle state. Once the script is launched we can see the increase in power consumption

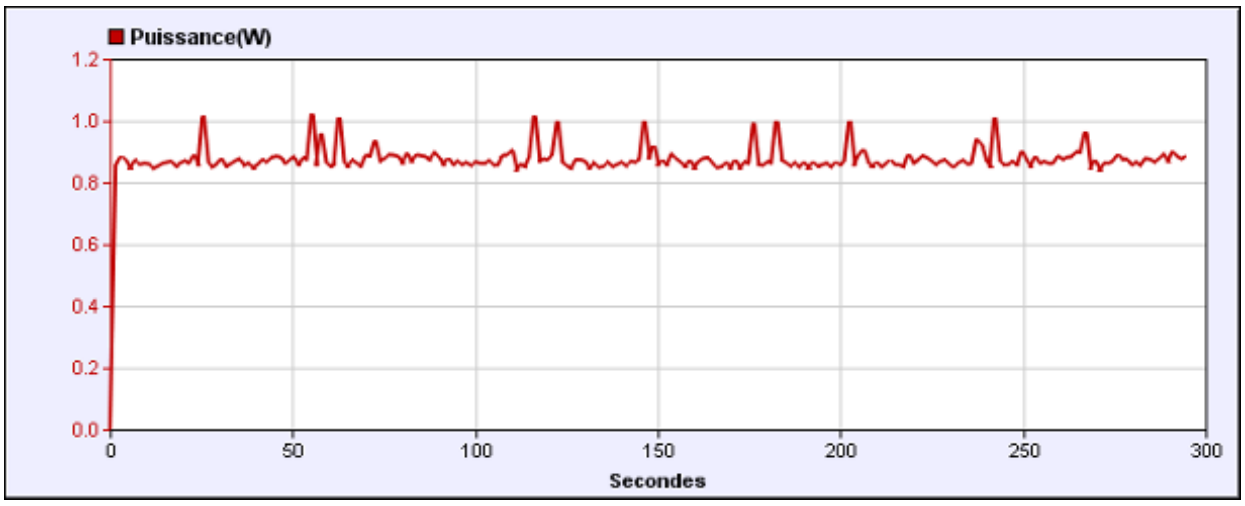


Figure 4.6: Baseline power readings for RPi Zero

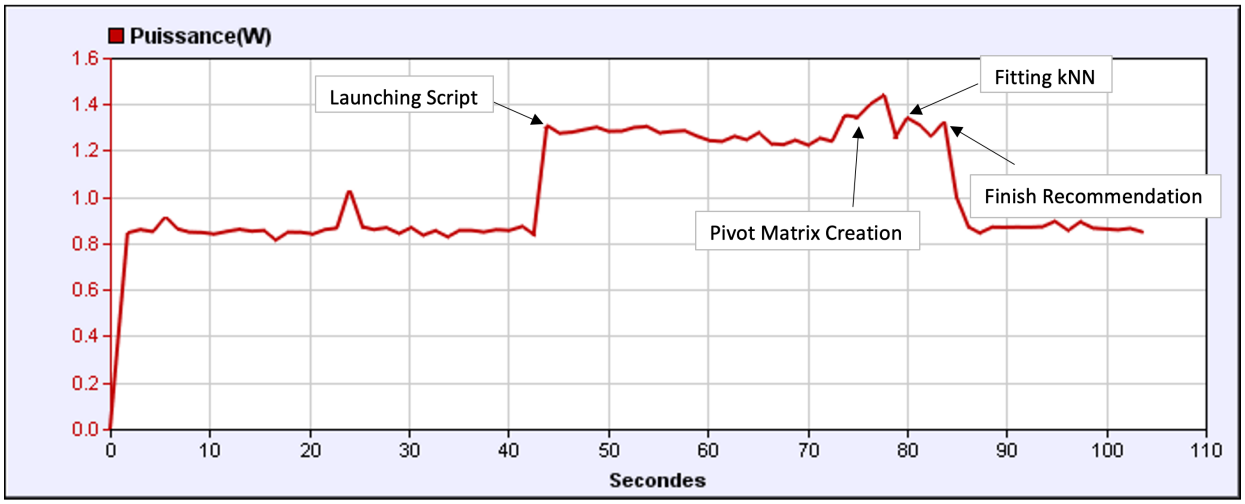


Figure 4.7: Power readings for RPi0 recommendation on single user

to about 1.3W where it remains rather stable the time it takes to load the data until the point in which the script creates a pivot matrix. A pivot matrix is the matrix $P_{u,f}$ of u number of rows and f number of films. This matrix is explained above is computationally expensive as it has a size of over 23 million integer values. That's why one can see the peak of the entire process be the point of creating the pivot matrix as it is the most computationally expensive operation of the script. Then on there are slight peaks at the time of fitting the kNN model and of making a recommendation. Making recommendation although quick requires some calculations such as that of the predicted ratings and of retrieving the list of recommended films from the films dataset[3]. Once the recommendations are done it returns to a stable power consumption state of 0.9W.

[3]The films dataset is noticeably smaller in comparison to the ratings data hence its loading time is negligible and there are no operation done on it hence it isn't taken into consideration.

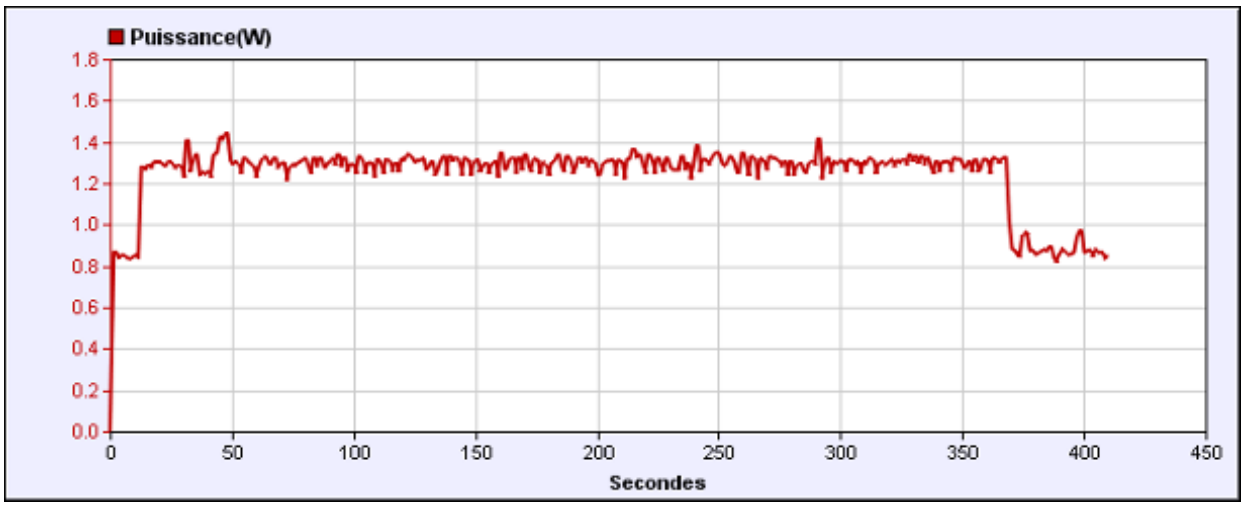


Figure 4.8: Power consumption for the recommendation of 100 users

Figure 4.8 shows the power consumption readings for the second computational time test executed. The power was read for these tests as well as such recommender systems aim to serve multiple users at a time from a single dedicated device. From the time readings at the bottom of the screenshot one can see that there was an increase in power consumption for about 360 seconds to an average of 1.3W. What this would represents is a total consumption of 0.00013 kWh over a period 360 seconds. This represents a total cost of 0.0027.- at the current electricity Swiss price of 0.208 francs per kWh[4]. At and idle state this models would represent a power consumption of 0.0216 kWh for a day which would come at a cost of 0.45.- per day.

This power consumption of this model is lower than other models of RPi's as it requires not to power a much more powerful CPU like the other have but processing large amounts of data are more computationally expensive overall as this models requires considerably more memory and computational time to complete a similar task.

4.5.2 Results for the Raspberry Pi 3B+ model

On the larger Raspberry Pi 3B+ model the baseline reading comes in at about 1.6W when is is at an idle state. This power consumption represent a total use of about 0.0384 kWh for a day which would cost about 0.80.- for a day to run. Once again one can see in figure 4.10 the different points in the power reading where the script executed different tasks which show a peak at the point of creation for the pivot matrix of the ratings data and at the time of recommendation. However once again these tasks are the ones that are least computationally expensive in terms of time in contrast to loading data.

[4]This price is retrieved from the website `https://www.globalpetrolprocess.com/Switzerland/electrivity_prices/` and is dates from June 2021

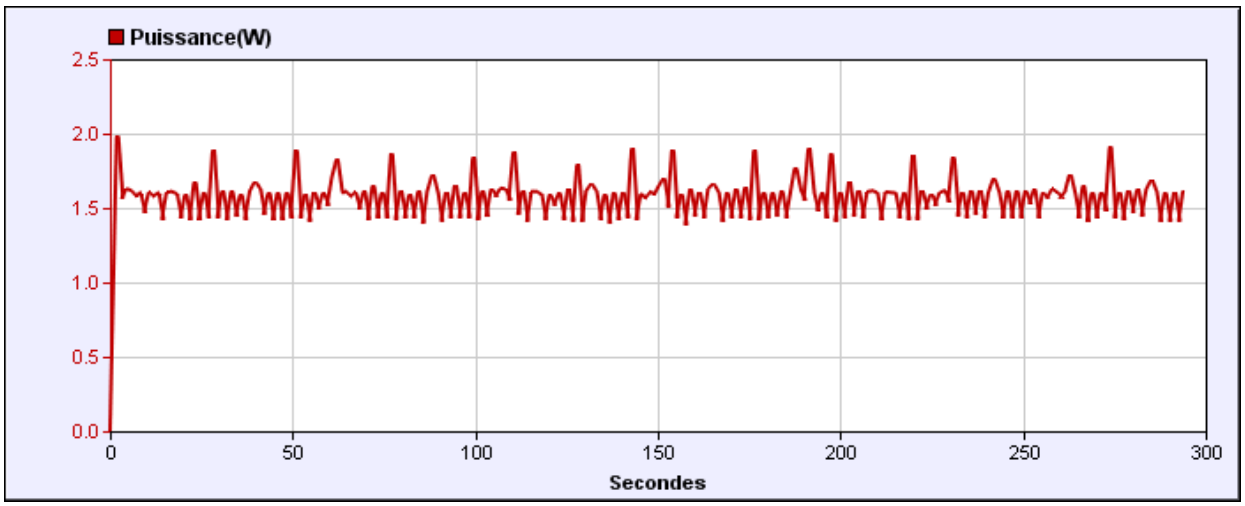


Figure 4.9: Baseline power readings for RPi 3B+

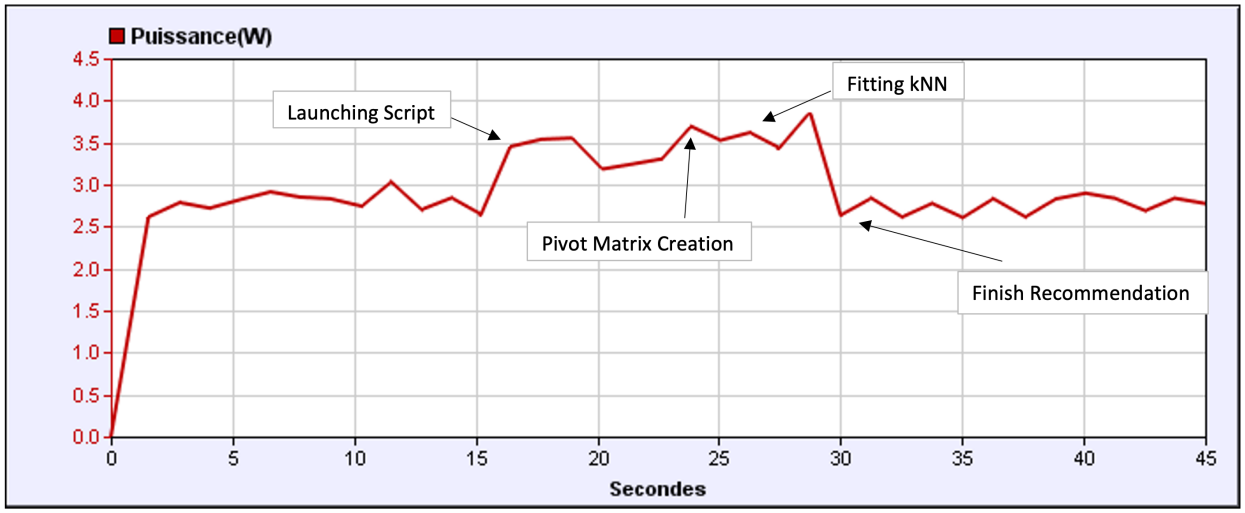


Figure 4.10: Power readings for RPi 3B+ recommendation on single user

In figure 4.11 one can see the overall power consumption for the recommendation of 100 users. One can see that this process being much quicker uses an overall decreased amount of energy in the process of making recommendations. This process represents a total energy consumption of about 0.000087 kWh which would come in at around a cost of 0.0018.- which is marginally cheaper than the 0.0027.- that it would cost for the same task with the Raspberry Pi Zero W.

4.5.3 Results for the Raspberry Pi 4 model

On the most powerful of the Raspberry Pi models the baseline power consumption is of 2.6W which is considerably larger than the 0.9W of the Zero model and the 1.6W of the 3B+ model.

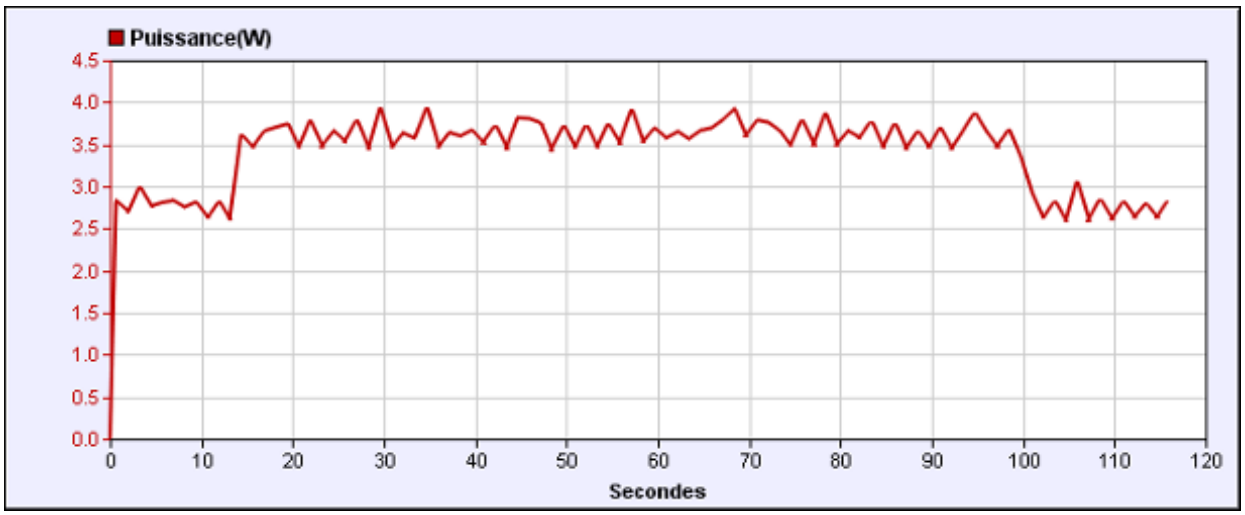


Figure 4.11: Power consumption for the recommendation of 100 users

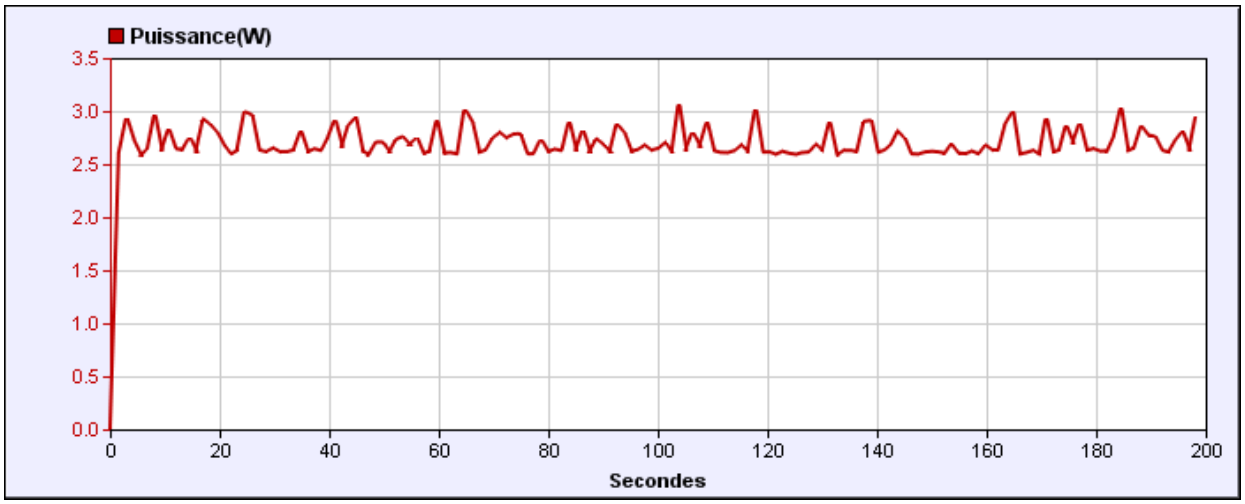


Figure 4.12: Baseline power readings for RPi 4

This can be attributed to it having the more powerful processor of them all. With respect to the power consumption reading for running the recommender system script one can see on figure 4.13 that there is a slight spike up to 5W at the time of running the script. It was difficult to place where the more computational expensive part for running the process on this computer were as the recommendation was done fairly quickly. As seen in the computational time readings the execution for the whole program took about 2.13 seconds yet the power consumption shows that it run for about 4 seconds. This discrepancy can be attributed to the *PowerSpy2* taking readings in 1 second time intervals and it missing the readings of when the script was done. The baseline power consumption represents a total cost of 1.29.- for a total consumption of 0.0624 kWh in a day. In comparison to the other two models this is just about .80.- more expensive than the Raspberry Pi Zero and only about .50.- more expensive than the Raspberry Pi 3b+ model.

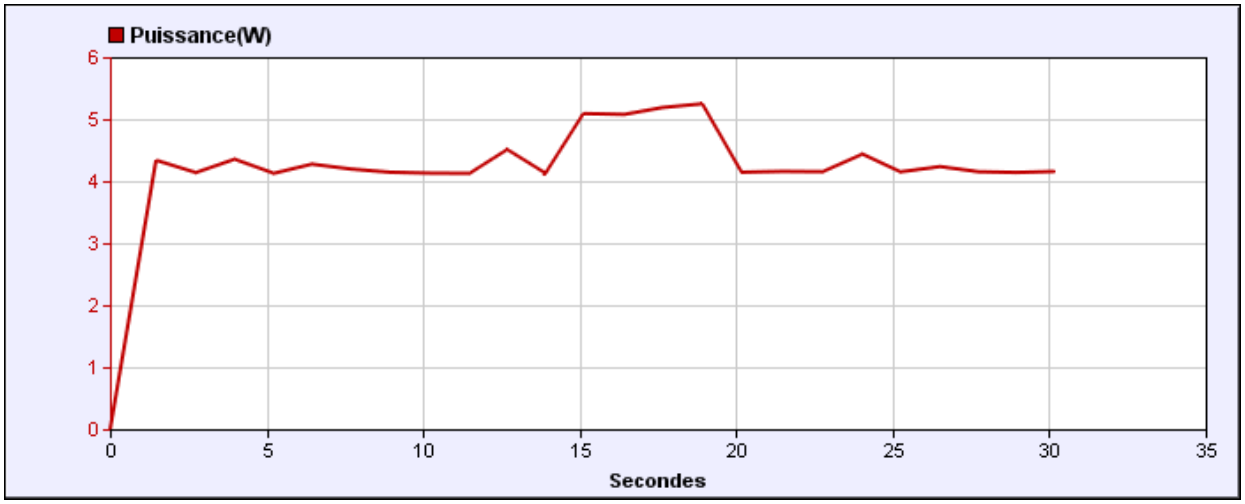


Figure 4.13: Power readings for RPi 4 recommendation on single user

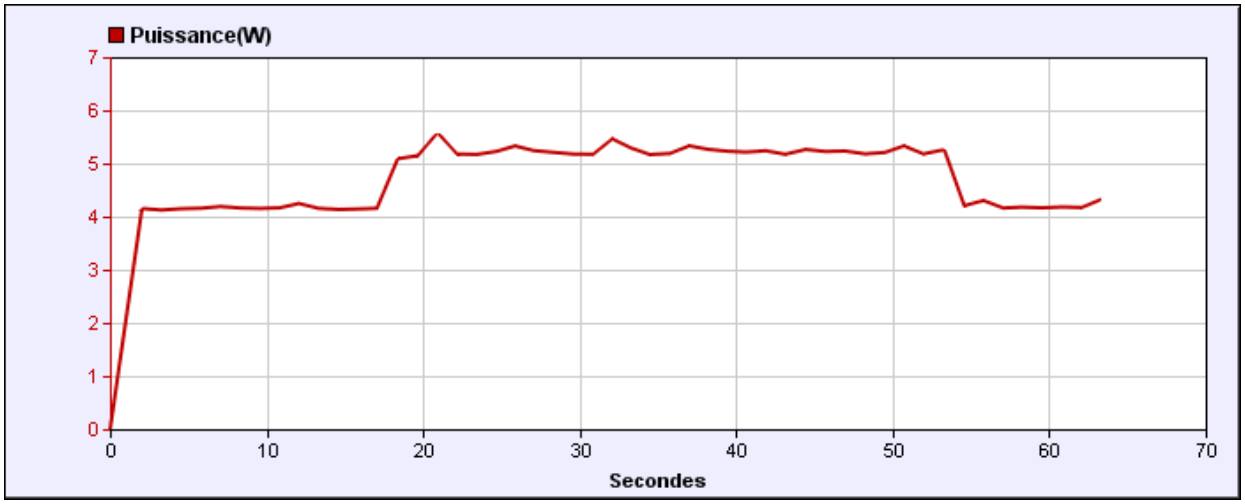


Figure 4.14: Power consumption for the recommendation of 100 users

With respect to the recommendation of 100 users the Raspberry Pi 4 peaks to about 5.1W for a period of 35 seconds which is coherent with the 30 seconds reading taken in the computational time testing. Another reason as to why there might be a slight discrepancy between the two readings is that 35 seconds represent the total time the CPU was active which also takes into account the loading of script information onto the CPU. However, the real runtime of the script is only taken into account from the moment the script is in CPU and runs completely.

To make the recommendation for 100 users it would represent a power consumption of 0.000050 kWh and it would cost the dedicated system about 0.0010.- to run the recommender for 100 users. This represent an efficient system when it comes to cost of operation because even though it does consume more energy the calculation times are significantly faster than the other models which in return would create an overall less expensive model to use as a dedicated

recommender system.

4.5.4 Results Analysis

After doing the various tests on the different RPi models I was able to form conclusions on both research questions set at the beginning of this evaluation. When it comes to the question on the devices efficiency with respect to computational time and memory usage, one can clearly determine that for the dataset used in testing the most efficient was the Raspberry Pi4. When it comes to the Raspberry Pi Zero W model the lack of RAM capacity and processor computational capacity, means that the computations for recommendations results in large values. As seen in the computational time part one can see how for 100 users the recommendation time varies widely with the maximum value doubling the time it would take for the fastest user. It also must allocate $\frac{1}{5}$ of its RAM in order to process the 2MB dataset it was presented with which indicates that it would struggle with larger datasets. In the world of today datasets can grow to be much much larger than the 2MB dataset presented therefore having the Raspberry Pi Zero as a dedicated system for a recommender can be excluded regardless of its diminished power consumptions.

The Raspberry Pi 3 can be seen as a middle compromise between the other two models. However, when it comes to its RAM one can quickly see that it is rather inefficient as with this dataset it must allocate about 10% of its RAM in order to handle it. For larger datasets this system would struggle in comparison to the 8GB model. Therefore, this model was discarded as a viable efficient dedicated recommender system.

Finally, the model 4 is clearly the most efficient of the systems with respect to computational time as it is much faster than the other Raspberry pi models. It uses significantly less relative memory as it only has to attribute about $\frac{1}{100}$ of its memory in order to handle the 100 thousand dataset. When it comes to its power consumption, the Raspberry Pi 4 is the most energy efficient due to the fact that the energy used per recommendation is lower as that computations are faster by a larger amount than the increase in energy usage. To be able to recommend 100 users with our dedicated script it is highly computationally efficient and as seen in 4.1 it is fractionally slower than a MacBook.

Therefore, after the evaluation of these devices performance and economics, Raspberry Pi's appear to be promising for a dedicated recommender system.

4.6 Scaling

An interesting reason for exploring dedicated systems for running applications is that of their capability to scale. With the case of the recommender system scaling means being able to

treat larger datasets of ratings and thus larger demographics. In this part I quickly evaluate the capabilities of scaling with the Raspberry Pi 4 on the larger MovieLens dataset. These datasets are the 1 million, 10 million, and 20 million ratings datasets. They have 6'040, 71'567, and 138'493 user respectively and 3'900, 10'681 and 27'278 films respectively. The fact that these datasets are much larger must make one take into account the way in which the similarities between users are calculated specially in the case of the 20 million ratings dataset. That's why in chapter 3 I introduced the second version of the recommender which uses blocks to calculate the similarities in a manner that would not overflow the memory and lead to the process killing itself. In this part I will present the results of running this second script on the Raspberry Pi 4 as it has been shown to be the most efficient system when it comes to using it as a dedicated device.

The tests done with these datasets were similar to those carried out above. However, extra specific parts of the code were measured as it is interesting for the sake of scalability. The first test is a recommendation for a single user but it focuses mainly on the time taken to execute different parts. The parts being measured are as following:

1. Time taken to read films data in seconds.
2. Time taken to read ratings data in seconds.
3. Time taken to create sparse matrix R[5].
4. Average time taken to calculate a block from the similarity matrix of size 200.
5. Average time to calculate the top k neighbours for a block of size 200.
6. Total time taken to execute the prediction for a single user.
7. Percentage of memory used by the process.

In the second test the throughput for 100 users is tested similarly to the test done in part 4.1 but only the average time taken to make a recommendation for a user is tested.

4.6.1 1 Million Ratings

This dataset has a total of 6'040 users which is roughly a similar size to the number of bachelor students at EPFL. The measurements for for the 1million dataset are as following:

1. Time taken to read films data in seconds. *0.03 seconds*
2. Time taken to read ratings data in seconds. *9.37 seconds*

[5]This matrix is a replacement for the pivot matrix in the script that doesn't use blocks as it eliminates all values that are 0.

3. Time taken to create sparse matrix R. *0.14 seconds*
4. Average time to calculate a block from the similarity matrix. *0.3906008 seconds*
5. Average time to calculate the top k neighbours for a block. *0.37010639 seconds*
6. Total time taken to execute the prediction for a single user. *0.02936249 seconds*
7. Percentage of memory used by the process. *2.16% of RAM*
8. Total time to make predictions for 100 users. *2.5331274 seconds*
9. Average time to recommend for an user out of 100 users. *0.02533127 seconds*

4.6.2 10 Million Ratings

This dataset has a total of 71'567 users which is roughly a similar size to the population of Luzern. The measurements for for the 10 million dataset are as following:

1. Time taken to read films data in seconds. *0.10 seconds*
2. Time taken to read ratings data in seconds. *94.11 seconds*
3. Time taken to create sparse matrix R. *0.44 seconds*
4. Average time to calculate a block from the similarity matrix. *8.4905066 seconds*
5. Average time to calculate the top k neighbours for a block. *2.831941518 seconds*
6. Total time taken to execute the prediction for a single user. *22.2339231 seconds*
7. Percentage of memory used by the process. *16.74% of RAM*
8. Total time to make predictions for 100 users. *0.9781501031538937 seconds*
9. Average time to recommend for an user out of 100 users. *0.00978150103 seconds*

4.6.3 20 Million Ratings

This dataset has a total of 138'493 users which is roughly a similar size to the population of Lausanne. The measurements for for the 20 million dataset are as following:

1. Time taken to read films data in seconds. *0.23 seconds*
2. Time taken to read ratings data in seconds. *184.98 seconds*

3. Time taken to create sparse matrix R. *0.88 seconds*
4. Average time to calculate a block from the similarity matrix. *23.711667048 seconds*
5. Average time to calculate the top k neighbours for a block. *5.829091124 seconds*
6. Total time taken to execute the prediction for a single user. *0.0712095642 seconds*
7. Percentage of memory used by the process. *33.0% of RAM*
8. Total time to make predictions for 100 users. *7.3438431606628 seconds*
9. Average time to recommend for an user out of 100 users. *0.07343843160 seconds*

4.6.4 Results Analysis

By blocking the code computational times for fitting the similarities amongst users increases considerably. Take for example the 1 million dataset. This dataset is a 10-fold increase in the size of that which was used for evaluating the performance of different Raspberry Pi models. But when fitting the kNN for the 100 thousand user it only took about 0.105 seconds whereas with the 1 million dataset it takes about 11 seconds. This increase in computational time is considerably much larger but it is a tradeoff that must be made in order to accommodate the 2% usage of RAM that this datasets requires. Therefore, though one would expect a significant increase in use of RAM when calculating these similarities by blocking the way in which it is done can reduce this so that the system can scale. When it comes to the 10 million dataset one can see how the recommendation for a single user takes about 20 seconds on average. This too can be reduced by further optimising the recommendation algorithm once the similarities have been calculated. But once again one can see how a set that is 100 times larger than that one used for the tests in part 4.1 through to 4.3 only requires about 16% of the available RAM. In comparison to the Raspberry Pi Zero which requires 20% for the 100 thousand dataset this method of computing the similarities is very memory efficient allowing for the Raspberry Pi 4 to scale well with such large datasets.

In the recommendation algorithm a list named *watched* is created which hold dictionaries for every single user with every movies they have and every rating given to said movie. This is computationally expensive and has a 10 fold increase in computation time as the watched list has the same size as the ratings dataset. This however, has very minimal impact on the recommendation times as can be seen. Once the list of dictionaries is created the recommendation for an individual user is very fast. the whole recommendation for 100 users would take the Raspberry Pi 4 less than a second with the 10 million dataset and 2.53 second with the 100 thousand dataset. This discrepancy would have to be studies further to analyse where the difference comes from. However, it is marginal when it comes to the recommendation of 100 users. Another interesting result was that scaling to a dataset with a similar size to the population size of Lausanne had a very small recommendation time per user. The recommendation time is a fraction of a second

once all the data has been process. Therefore, from these results one can clearly see that from a computational point of view having a Raspberry Pi 4 as a dedicated recommender system could be highly interesting as it scales very well in recommendation times. The part which can be left for improvement would be the computations of similarities as it is what is most computationally expensive in terms of computational time. This will be left as an improvement to the implementation of this recommender system in a distributed scenario with Raspberry Pi 4s.

As stated before using one of these devices as a dedicated device to compute similarities and give recommendations could be implemented in such a way that the ratings dataset is updated at different points in time and so that it isn't required for it to be calculated each time a recommendation for a user has to be made. I am leaving the implementation of such systems as an extension of this project.

Chapter 5

Possible Future Work

This project briefly explores the possibility of scaling the use of a dedicated device for a recommender system. The scaling is explored with the already implemented recommender system explained in chapter 3 using the blocks algorithm. By doing so I showed that it was possible to scale the recommender system to much larger datasets. However, the scaling analysis done above is one that can be researched further to improve the way in which the already implemented recommender system can become more efficient. The scaling exploration above was only done with a Raspberry Pi 4 however, it would be interesting to further optimise the blocking script to enable the other two Raspberry Pi models tested above to scale to larger datasets. By doing so one could explore the use of these devices in a distributed scenario of this project with multiple Raspberry Pi's. For example it would interesting to explore how these devices could implement a distributed recommender system for something such as restaurants at EPFL with student's dietary preferences as a replacement of the movies ratings in this project. With such one could explore the efficiency of these devices to collect data and update their recommendation system periodically like explained above as further improvement for this project.

An other area of interest for the Smallworld project would be to study the use of different interfaces to transfer data with the SSB protocol. As stated in chapter 2, although now all popular operating systems are supported for the Ethernet-over-USB option in the Smallworld setup, data transfer is not yet a supported option though this interface. This could largely broaden the spectrum of utilities for these dedicated devices using the SSB communication protocol.

Chapter 6

Conclusion

In this project I continued the work done by Romain Küenzi in SmallWorld V1 project by introducing the addition of Apple systems connectivity for the USB OTG option of the Raspberry Pi's Zero and 4. To do this I adapted some of the files previously changed in the Smallworld 1 project. For the MacOS environment it was a a matter of of updating the *dhcp* and *dhcpcd* files on the Raspberry Pi to add the *usb0* network interface as a connectivity interface. Afterwards on the MacOS it was a matter of allowing the possibility for Wi-Fi sharing which is a simple settings change. It is worthy of noting that some Wi-Fi networks do not allow you to share the connectivity to them through Ethernet like gadgets such as that of the EPFL. To then connect the Raspberry Pi to the Mac it is just a matter of connecting a single USB-C cable which both transfers data and power the Raspberry Pi 4. Similarly for the Raspberry Pi Zero where it can be powered through its second micro-USB port which is also the OTG port.

As for iPhone connectivity it is a matter of downloading iOS tethering libraries to the Raspberry Pi with the Debian package manager to the Raspberry Pi. On the iOS system one must just allow and trust the connection as well as turn on the iOS's hotspot. It is worthy of nothing that this only works with iOS devices that have the hotspot capabilities. Therefore, devices such as an iPod or iPad will not work with this set up. From then, it is a of matter using an SSH client app on the iOS device to connect to the Raspberry Pi. After doing both of these set ups the possibility of connecting to the Raspberry Pi through Ethernet-over-USB is achieved and now all popular OS environments are supported by the SmallWorld V1 project.

The second part of this project tackled the performance and economics of using Raspberry Pis as dedicated systems for popular application such as recommenders. In this case a movie recommender was implemented and used to evaluate the computational and power efficiency of different Raspberry Pi models. To do this a methodology was introduced where the Raspberry Pis were tested in similar conditions and their computational time, memory, and power efficiencies were measured. The economics of these systems were evaluated with respect to their power consumption and the prices of electricity in Switzerland. With these results I managed to come

to the conclusion that the Raspberry Pi model 4 is most efficient all-around with respect to recommender systems mainly for their capacity to upscale to much larger datasets, efficiency in computational time and economics. The cost of running this recommender system is very similar to that of the Raspberry Pi Zero who is 12 times less efficient than the RPi 4 model. Therefore, it is a viable option when it comes to using this model as a dedicated system for recommender systems.

With this project I managed to advance the development of the work started in the SmallWorld V1 project by ensuring that all popular OS environments were supported by it. I also set in place a test and methodology of determining the feasibility of using a Raspberry Pi 4 models as a dedicated device for a recommender system as it is most efficient in computational time, memory usage, and power consumption. This gives way to exploring the possible applications which could be taken on by a Raspberry Pi and with that diminishing costs of running such application on the dedicated devices to decrease the power enabling trend that computing systems of today have.

Bibliography

[1] Debian. *Packages*. Accessed: October 15, 2021. 2020. URL: https://packages.debian.org/sid/utils/.

[2] Aditya Santokhee Girish Bekaroo. "Power Consumption of the Raspberry Pi: A Comparative Analysis". In: *IEEE International Conference on Emerging Technologies and Innovative Business Practices for the Transformation of Societies* 20.3 (2016). DOI: doi : 10 . 1109 / emergitech.2016.7737367.

[3] F. Maxwell Harper and Joseph A. Konstan. "The MovieLens Datasets: History and Context". In: *ACM Transactions on Interactive Intelligent Systems* 5 (2015), pp. 1–19. DOI: doi.org/10.1145/2827872.

[4] Rahul Katarya. "An effective collaborative movie recommender system with cuckoo search". In: *Egyption Informatics Journal* 18 (2017), pp. 105–112. DOI: doi.org/10.1016/j.eij.2016.10.002.

[5] Lusi Li. "Economics of Recommender Systems in Online Marketplaces". In: (2017).

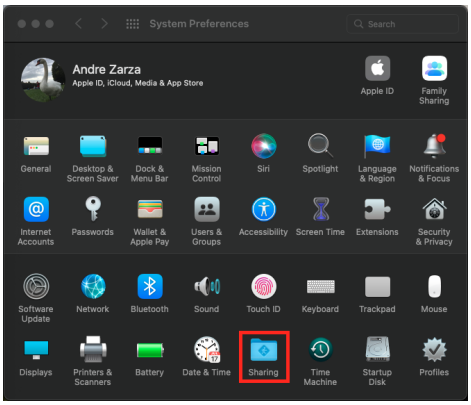
[6] Anand Nayyar Rishabh Ahuja Arun Solanki. "Movie Recommender System Using K-Means Clustering AND K-Nearest Neighbor". In: *International Conference on Cloud Computing Data Science and Engineering* (2019), pp. 263–268. DOI: 10.1109/CONFLUENCE.2019.8776969.

[7] Mark Weiser. "The Computer for the 21st Century". In: *ScientificAmerican* (1991), pp. 94–104.

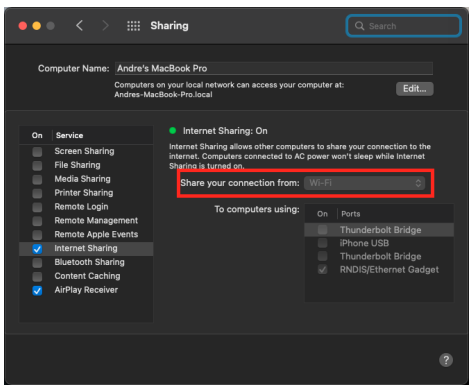
Appendix A

Appendix for Chapter 2

This is an appendix to chapter 2 containing images on the instructions you should follow to enable both MacOS and iOS connectivity to the Smallworld V1 setup. These images are in order according the the orther of the instructions for chapter 2.

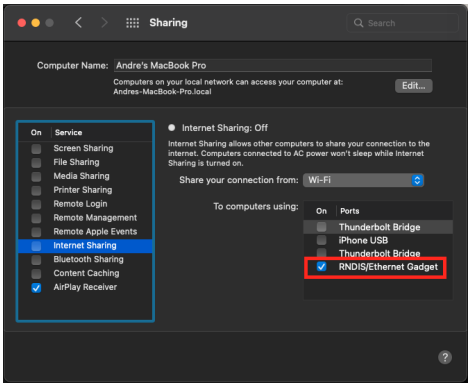


(a) Sharing

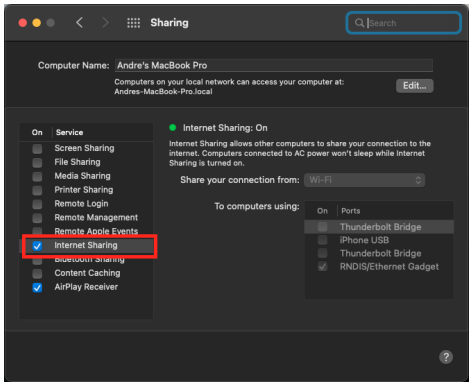


(b) Share Your Connection

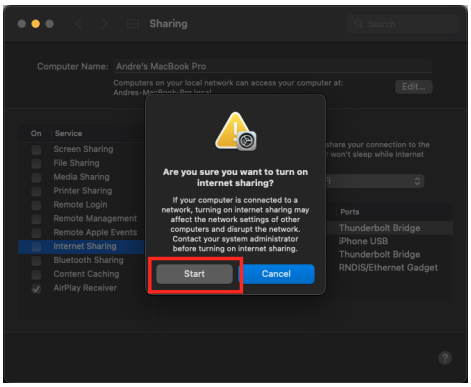
Figure A.1: Enabling Ethernet-over-USB on MacOS (pt1)



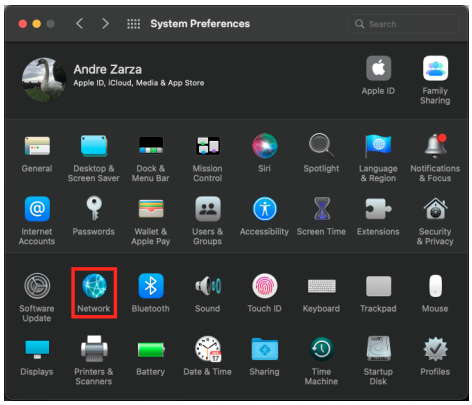
(a) RNDIS/Ethernet Gadget



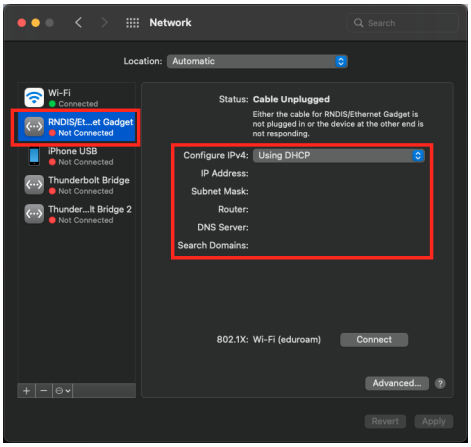
(b) Internet Sharing



(c) Question on sharing internet.



(d) Network Section



(e) DHCP setup

Figure A.2: Enabling Ethernet-over-USB on MacOS (pt2)