Energy Consumption Framework – Data Collection and Analysis

This document gives a short overview over the current process of Data Collection and the analysis thereof.
The purpose is to model the energy consumption of an esp32 with certain system metrics as indicators.

Data Collection

The data is collected through a circuit involving three elements. The esp32 is the object being observed, it is powered through a power supply in the breadboard and the esp32's 5V regulated input pin. The instructions the esp32 is supposed to compute are delivered through a tpc-server.

The second element is an INA219, this modem is designed to measure small currents and voltage. The range of measurements is perfect for the esp32's power consumption in the low three digit milli-amphere range. The INA219 is connected in series with the esp32's power circuit. It also has output pins, to deliver the collected metrics to the last element the raspberry pi.

The raspberry pi serves two purposes. The first is to collect the data from the INA219 and write it back to file, but it also coordinates this collection process with the esp32. Because the model takes some system metrics of the esp32 to predict the power consumption this collection process is necessary to make sure the data is collected at the right time. This process is described in figure 1 on the side.

The main goal is to start collection system metrics on the esp32, start collecting power consumption data and do the corresponding task at the same time. But also to stop all these three processes at the same time. The main work to observe is marked in blue.

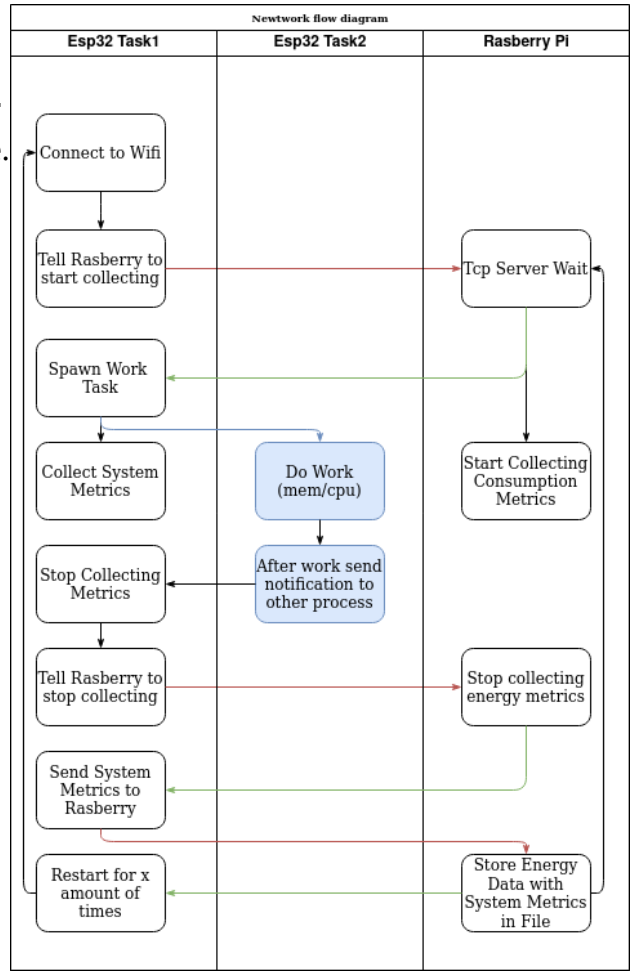


Figure 1: Network Flow Diagram

The circuit is explained here in figure 2:

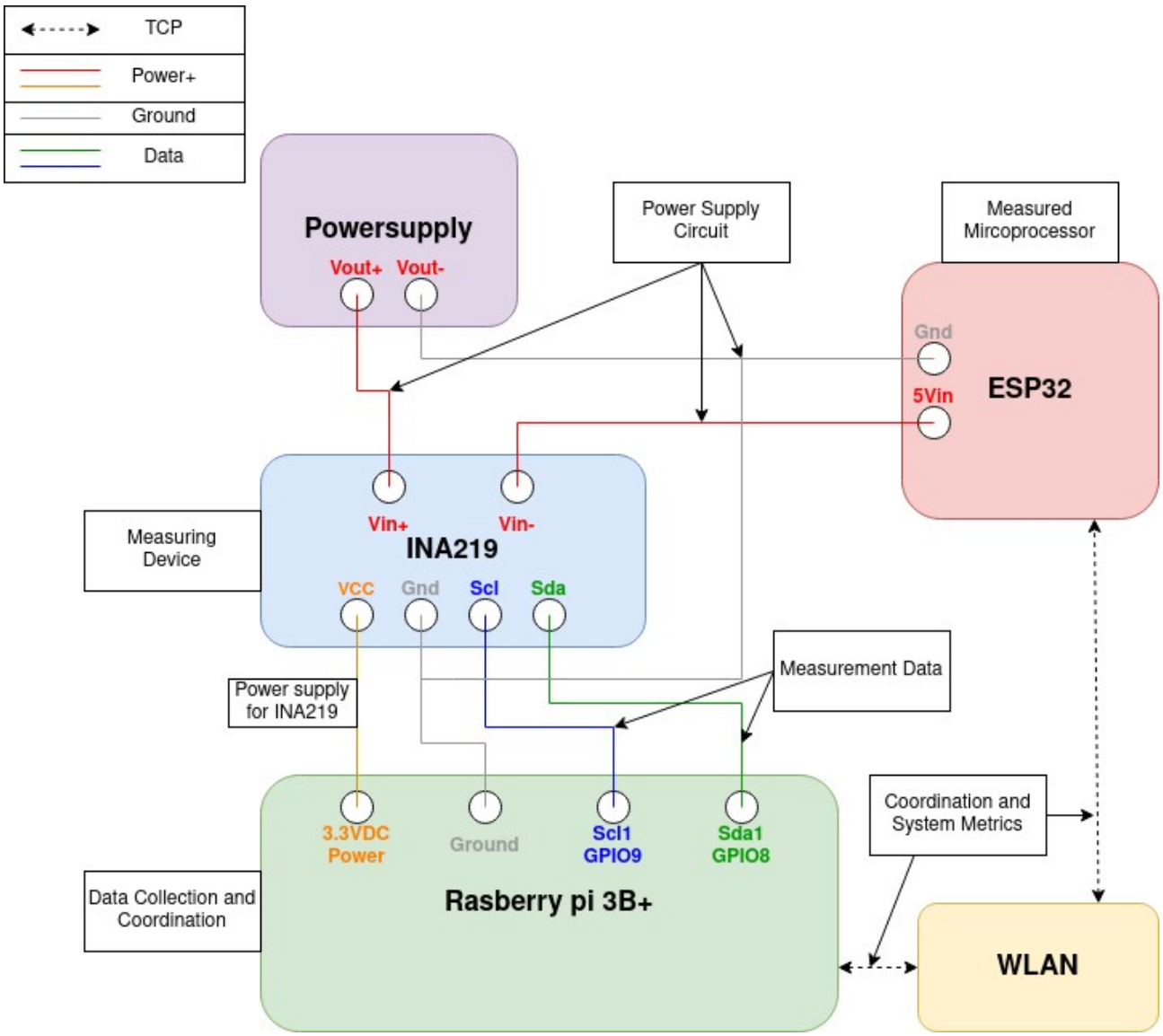


Figure 2: Circuit Design

Data Modeling:

The collected data is split between system metrics and power consumption metrics. The goal is to find a stable relation between these two metrics.

The general practice for modeling power consumption of embedded systems, is to break up the circuit into individual components. The biggest consumption that can be easily observed is the CPU.

Modeling the CPU power-consumption is typically done by executing single instructions for a prolonged period of time. Then averaging out the amount of instructions executed. Then one can compare these instructions for how much they impact the consumption of the CPU. In this case all instructions ran for the same time. But two factors were varied.

Firstly each instruction was measured with all three available CPU frequencies, namely 240mhz, 160mhz and 80mhz. Secondly through the use of pausing a task the instructions were run only a percentage of the overall run time (25%,50%,75%,100%). The measurements over 1 run are averaged out, to one value. Because when the task is not running, it will produce a certain consumption and then if its running it will produce a higher power-consumption. The average of these gives an assumption how much a task would consume, if its not running 100% of the time.

The observed relation between percentage of run time/CPU frequency and observed power consumption is observable and linear. Unfortunately even after averaging out results and improving the timing the data is very spread out.

Figure 3 and Figure 4 are each showing the relation of the add operations power consumption in relation to either the CPU frequency or the percentage of run time of each run. A linear regression is fitted over the data points.

Figure 3 is showing the CPU frequency which is only possible to set to 80,160 and 240. While the data is very spread out (a mean squared error of 1435), there is a clear linear relation. As expected a higher CPU frequency means more clock cycles in the same time period resulting in a higher power consumption. One explanation for the data variation could be that, the CPU utilization is not the same for all collected values here.

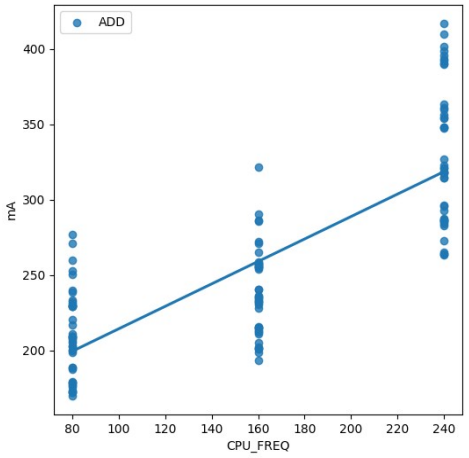


Figure 3: Add operation CPU frequency and power consumption

Figure 4 shows the percentage of time the work task (in this case adding two operands) is run in relation to the total run time of a run. This relation is not clearly observable by eye. But the regression fits a slightly less positive relation then the CPU frequency. Again one explanation could be, that for each percentage the CPU frequency varies, which seems to be a much more dominant factor, then the work time. The almost doubled mean squared error of 3172 proves the loss of explanatory power in this approach compared to the frequency. It makes sense, because even if no work is done, the CPU has to support a higher clock cycle anyway.

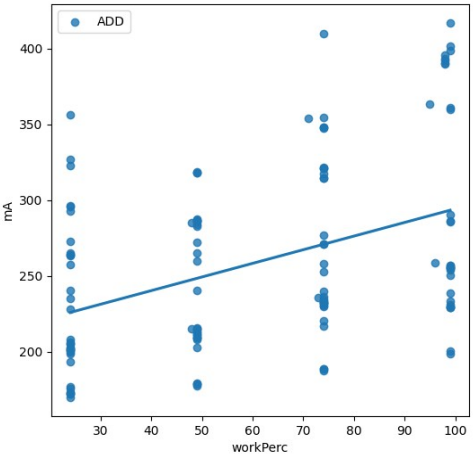


Figure 4: Add operation Worktime percentage and Powercunsumption

The next step is to eliminate the two factor percentage of work time and CPU frequency and see if the results become clearer.

Figure 5 shows the relation between percentage of run time and power consumption. But this time the CPU frequency of each run is plotted in different colors and regression lines. As expected before a big part of the variance in the data can be explained from the CPU frequency. The mean squared error for the 80,160 and 240 mhz goes down to about 513,449 and 685 respectively, which results in an average error of roughly 549 a third of the previous best case. The relation for the run time is still linear in each case. But the highest frequency shows an interesting correlation. The higher the CPU frequency is the more important the percentage of work time becomes. This could be explained by the amount of run operations. The difference between an idle task running on different frequencies seems to be a constant factor, which is easily observed at the 25 percent work time. The CPU frequency doubles from 80 to 160 and then doubles again from 160 to 240. The difference between the values at the 25 percentage line show that. The difference between 80 and 160 frequency seems about half the difference between 160 and 240, which makes sense as the difference in frequencies is also double (80 – 160 => 80 difference; 160 – 240 => 160 difference). But the 240mhz data points seem to have a higher gradient. Meaning for higher CPU frequencies the work time percentage plays a much larger role.

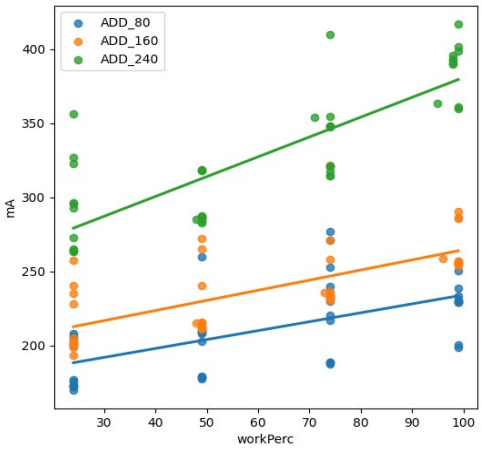


Figure 5: Add Operation Percentage Run Time with Limited Frequency

The updated Figure 6 also shows some explanatory power but less so then before. Some of the variance in CPU frequency can be explained through the percentage of run time the task actually runs. Similar to before the mean squared error reduces a lot to an average of 723 (25%=>640, 50%=>511, 75%=>1088, 100%=> 656). But mirrored to Figure 5 the explanatory power of the percentage is not as big as the Frequency, as can also be seen by the slightly larger mean squared error. But a similar trend is observable in the reverse fashion. The higher the work percentage the higher the impact of the frequency is, meaning the gradient increases. At least for the highest two percentages. The explanatory power of frequencies also seems lower the lower the work time percentage is. Where the lower two lines are really close to one another.

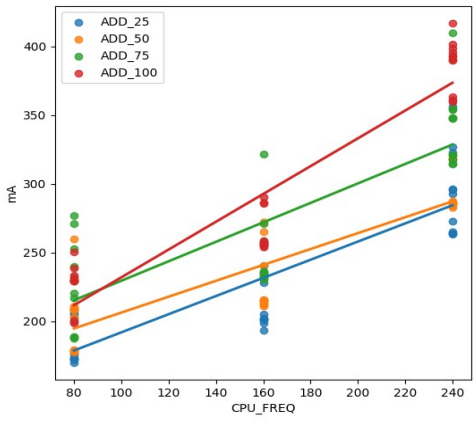


Figure 6: Add Operation CPU Frequency with Limited work percentage

There are a few learning's from this section, that are necessary to build the model:

1. There is a clear linear relation between Frequency and Power Consumption

2. There is a clear linear relation between Percentage of Run Time and Power Consumption

3. Higher Frequencies increase the importance of work time percentages

4. Higher work time percentages increase the importance of Frequencies

5. Frequency is the more dominant factor of explaining Power Consumption

The final question for this interaction is, if there is an observable combined influence of both CPU Frequency and work time percentage on power consumption.

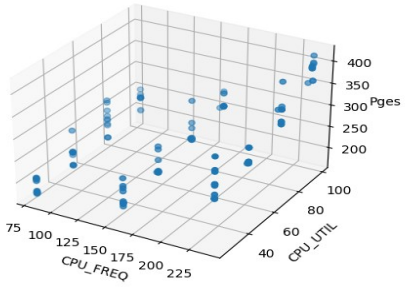


Figure 7: Add Operation CPU Frequency Percentage Work Time Power Consumption

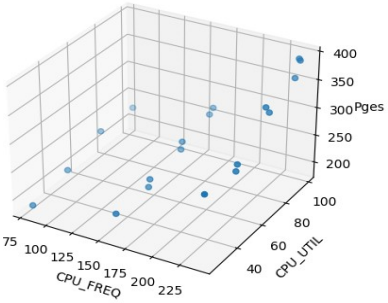


Figure 8: Add Operation CPU Frequency Percentage Work Time Power Consumption Mean

Figure 7 shows the three-dimensional relation between power consumption and CPU Frequency/Work Time. Figure 8 shows the same just that the power consumption is averaged out for every CPU Frequency/Work Time constellation to make the picture more readable.

As expected from the previous analysis increasing both Frequency and work time percentage increases the power consumption, as well as individually increasing each parameter. But the interesting fact is that this relation is not linear. It seems to have an exponential factor. Increasing both factors seems to increase the mean of the power consumption in more then just a linear relationship. Looking back at the two dimensional pictures this seems to hold for the CPU Frequency but not the work utilization. Figure 8 also reflects that, where the CPU Frequency curves upwards, while the utilization just linearly increases.

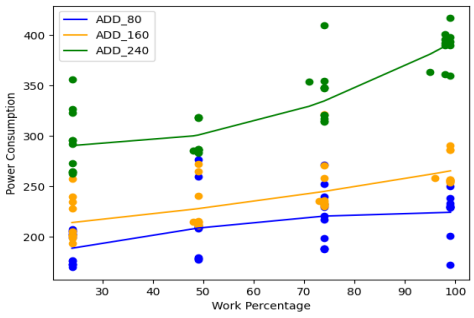


Figure 9: Add Operation Percentage Run Time with Limited Frequency Polynomial Degree 2

It makes sense to see if a polynomial regression can fit the Figure 5 and 6 in a more accurate way and reduce the error.

In Figure 9 a polynomial of degree 2 is used to fit the data and see if the error is reduced. For the 240mhz case the mean squared error is reduced from 685 to 555 so a big improvement. For the other two cases it is only decreases a bit (513 => 499 and 449 => 447). In these cases it does not make sense to fit a polynomial of degree 2 because over-fitting should be avoided.

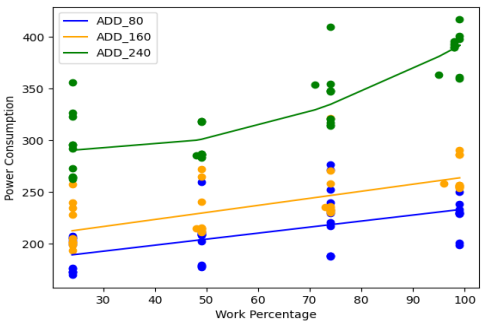


Figure 10: Add Operation Percentage Run Time with Limited Frequency Polynomial Degree 2 for 240mhz

Figure 10 shows the only polynomial of degree 2 is the 240mhz case. The respective errors are now 513,449 and 555 which results in an average of 505.

The lowest mean squared error is 505 which results in an average root error of 22mA. Which is actually a pretty good estimate already, considering the values range from 150mA to 400mA.

These two factors alone are quite complicated to model in their relationship to power consumption but the next step is more problematic and even more varied.

Operation Type

Too truly capture the energy consumption of a software project running on an embedded system, the software itself has to be observed. Most analysis focus on the before mentioned instruction sets. The idea is to measure different instruction performances and to build a model. One can then analyze the project for occurrences of these instructions to predict the energy measurements.

Some predictions from the previous segment, would be that these instructions are more differentiable at higher work percentage and higher frequencies. If a task is idle, it does not matter what it computes in relation to energy consumption. So higher work percentage results in less idle time and should result in higher differentiation between the actions. A higher frequency also means the values of the percentage data is higher spread out as seen in Figure 6.

Looking at other sources (https://eprints.soton.ac.uk/361481/1/hip3es_submission_3.pdf Figure 2) the difference between the different actions should be minimal. Also the difference between float and integers should not differ too much. The logic according to the paper is that all operations go through the same pipeline stages.

According to another paper (https://ars.copernicus.org/articles/2/215/2004/ars-2-215-2004.pdf) the biggest difference is in the size of the operands. The current project has not yet implemented looking at different operand sizes.

The next pictures show these differentiation's similar to Figure 3 and 4 without accounting the interplay of percentage and frequency. So a high variance is expected.

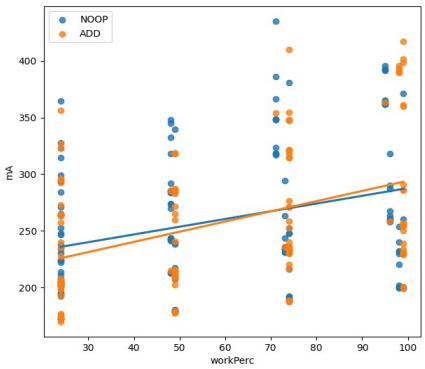


Figure 11: Noop vs Add work percentage

Figure 11 shows basically whats expected high variance and a low explanatory power in the low percentages. The difference between NOOP and Add can completely be accounted for by variance in testing, especially in the lower work percentages. But again as the work time is so low and these values are so close together, it is probably just not worth it to differentiate the actions in lower percentages.

Figure 12 shows an even worse outcome where the before mentioned issue becomes worse in higher frequencies. The next step is to take a close look for figure 12 and differentiate between the higher percentages as tried before.

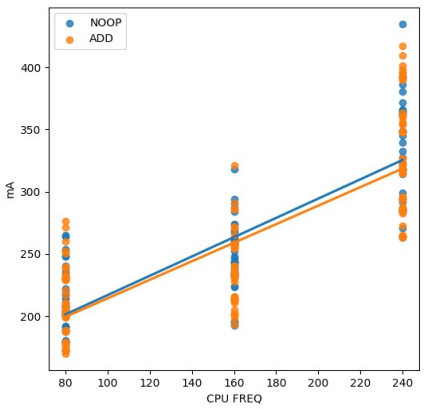


Figure 12: Noop vs Add Frequency

The difference between the add and noop operation are very small (Figure 13). As expected from the referenced paper there does not seem to be a big difference between the actual execution of an opcode. The power consumption seems to be mainly produced by the other pipeline stages such as e.g. fetching the operands.

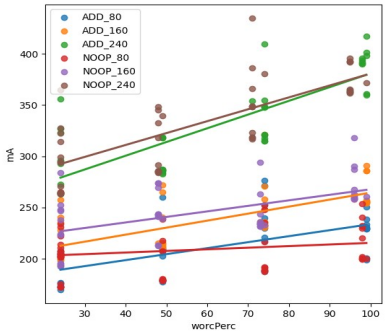
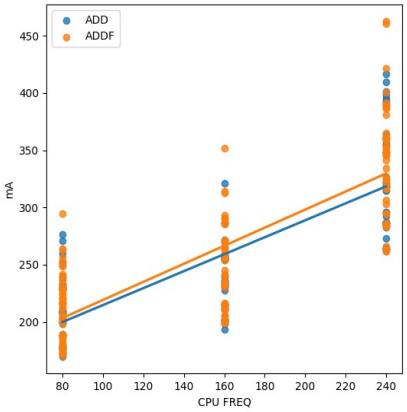
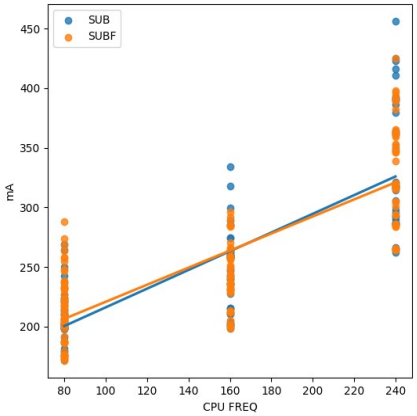
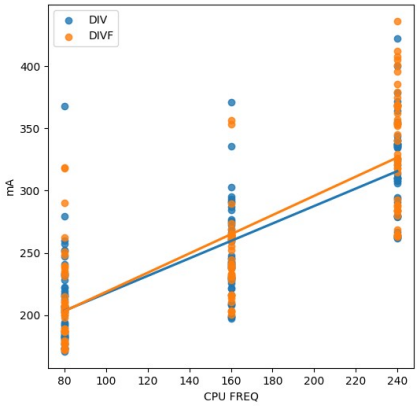


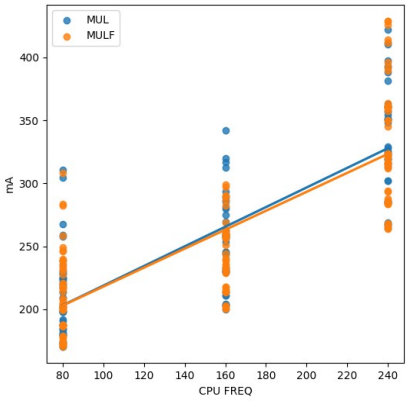
Figure 13: NOOP/Add Operation Percentage Run Time with Limited Frequency

The same can be seen for the integer vs floating operations:









Different Approach Random Forrest:

Because the main values used to differentiate aka Frequency and run time percentage are not continuous but distinct a Random Forrest might give a good approximation.

Pythons sci-kit offers a complete Random Forrest Generator. One can just feed data into the Regressor and get an output. Before only the factors were used that have a huge impact on the power consumption and that are not correlated with each other.

Just to see the difference the first tree(Random tree of the Random Forrest) is using all Features.

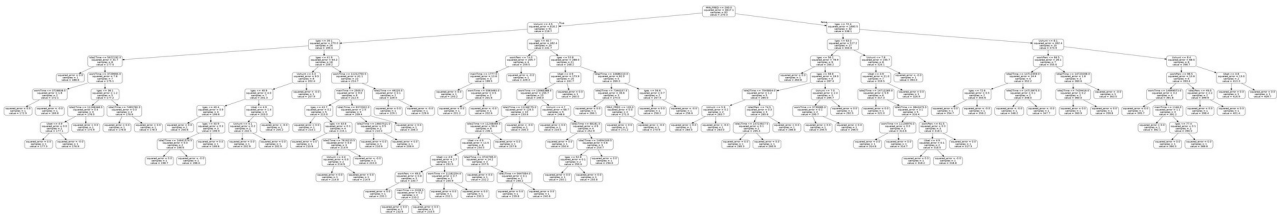


Figure 14: Random Tree with all Features

The generated tree is huge but it generates a almost non-existent mean squared error of 34 on the training data, which is a root error of roughly 6mA. Of course this is also due to over fitting. As explained before the next step is to generate a new data set to test the previous discussed data models on.

Looking at the tree some of the previously used features are still used at higher levels of the tree. The CPU Frequency is still at the top level, so as predicted earlier its the most important factor.

Afterwards some electrical data is used, but this is not a fair principle. The electrical data consisting of the voltage, voltage over the shunt resistor and the current. Obviously the shunt resistor and the current are at the next level of the tree. There is a direct electrical relation between the voltage over the shunt or the current and the used power consumption. So for the next Random Forrest these things should be left out:

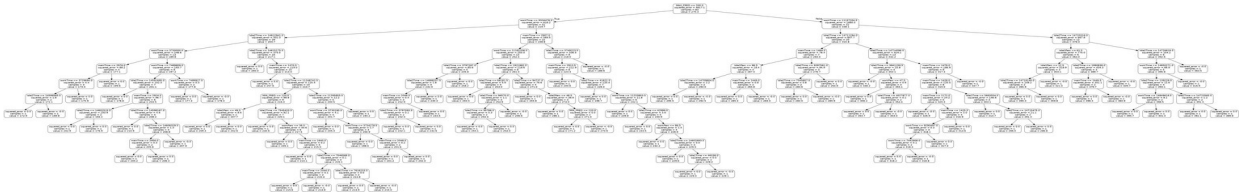


Figure 15: Random Tree without electrical data

Unfortunate but expected, the mean squared error increases to 91, which is a root error of 9.5mA. This is still a lot less then previous models, but again its on the training data. Looking at the features now the Tree uses CPU Frequency and work time as the main predictors. These are the same features used before. Although one can remove the work time or work percentage as they are the same values, just percentage is work time divided by run time. Maybe its better to use work time as it does not suffer from rounding.

The next Random Forrest has all duplicate or correlated features removed:

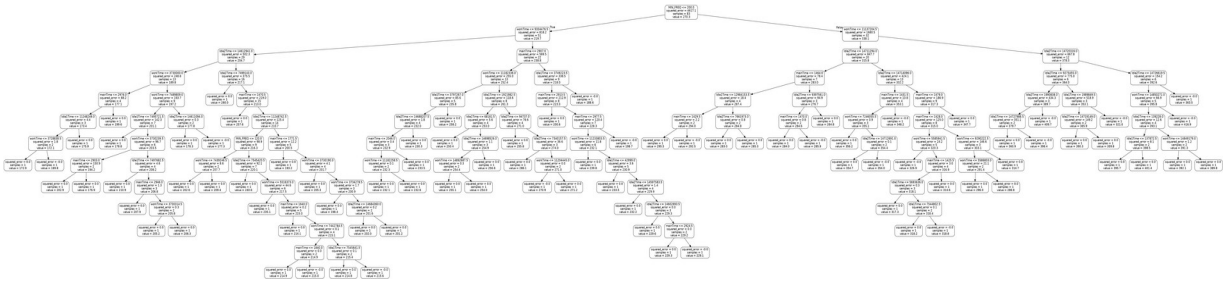


Figure 16: Random Tree without electrical data and without duplicates

As expected this does not change the mean squared error as no important data was lost. The top feature are still frequency and work time percentage. Interestingly this tree uses the idle time on the second processor as a further differentiation after the work time percentage. This is probably a case of over fitting as this value is just an inverse of the work time in relation to the total run time. Removing that feature results in this tree:

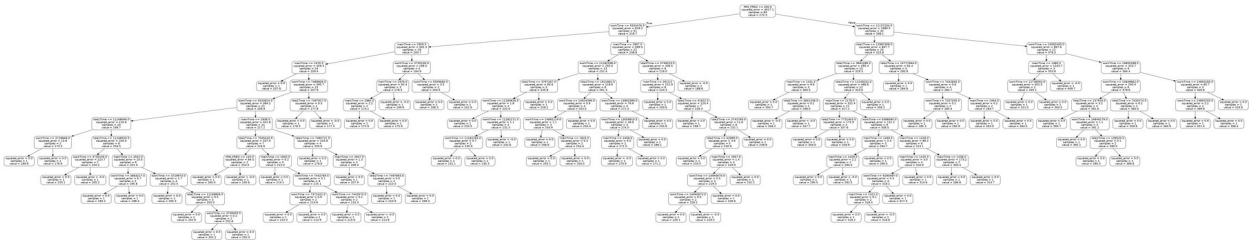


Figure 17: Random Tree without electrical data and without duplicates or inverse data

The mean squared error increases slightly to a 94 but that is probably just over fitting on training data. The main features are again the same as before but the accuracy on the training data is a lot higher. This is probably due to the data being very clustered and the main features not being continuous but rather distinct values. The average root error of 9-10mA is a very good predictor for power consumption.

Next Steps:

Observations from the past tests:

1. The root of the mean squared error seems to be small on the given training set

2. CPU Frequency and run time percentage are very important predictors of power consumption

3. Operand Type seems to have an almost non-existent effect on the power consumption.

4. Outliers are common

5. A mix of regression types in combination with grouping the values produces a low average error

6. Due to data clustering the Random Forrest achieves high accuracy on the training data

This leads to a few important next steps:

Build a test set, it is important to reproduce the collected values. The achieved accuracy is meaningless if its not reproducable

Try to reduce outliers. One important metric can be to enable light sleep mode. This mode reduces the amount of power consumption of non-used components. The biggest example is the WiFi unit

that sleeps if its not needed. One can actually manually set the component into light sleep and wake it up. If that reduces the amount of outliers then it might actually make sense to completely stop the background processes of the WiFi task. This wasn't done before, because reconnecting to the WiFi is incredibly flimsy, and increases the manual effort in running long time data collection.

Incorporate all observations made into one data model instead of manually switching functions. The before discussed random forest would actually make this possible. The only problem is that it turns the problem into a classification algorithm rather then a regression. But looking at Figure 10 the values are clustered into different areas anyway, because of the distinctive values used for prediction. So the regression only predicts one value per cluster anyway (<y>mA per Frequency/work percentage). Turning this into classifying the clusters makes a lot of sense.

Combine different instructions to test the transitivity of our model.

Model different aspects of the chip such as the memory, WiFi, Bluetooth components.

Test Data:

Running the experiment a second time gives a truly independent test data set. Now all the previous experiments can be tested for the mean squared error.

Firstly the tree in Figure 17 is used to predict the consumption on the test data set. The mean squared error(MSE) is now 8 times higher with a value of 820 which is a mean error of roughly 28mA. Even considering the variance of the data and that the range of values is between 150 and 400 the error is rather high with a percentage of 11 percent. One optimization could be to use the work percentage instead of the run time as the max run time could be slightly different each experiment. Removing the duplicates as explained before is actually beneficial and prevents over fitting the MSE would be higher with a value of 876. The MSE heavily weights outliers so other scores could be used such as the mean absolute error or the median absolute error, these result in an error of 22mA or 19mA each, which equates to a medium error of 7 percent. The mentioned paper on FLPA (https://ars.copernicus.org/articles/2/215/2004/ars-2-215-2004.pdf) achieves a median error of 5 percent of the power consumption. This is not far of with using a simpler model, but they are looking at different instructions which will be the next part.

Just combining all different operand data sets into one data frame and then running a random forest regressor increases the MSE of the training data to only 119. The test data has MSE of 1091 or a root MSE of 33, the median and mean absolute errors are still at 25 and 20. So a lot more outliers but the actual median prediction did not change much. One improvement is to introduce binary variables for each test set that divides into the type of operand. The newly created trees don't use the binary variables, proving that the difference between the operands is basically non-existent.

Reintroducing the voltage parameter to see if a fluctuation in the voltage could cause the variance. Luckily the experiment seems robust as introducing this variable increases the training data accuracy but reduced the test data accuracy, both in minimal amounts. Introducing the shunt resistor voltage drop reduces the prediction error to 16, 10, 5 (MSE, mean/median squared error), but there is a clear physical relation, so it makes sense.

The next point to analyze is the original model shown in Figure 10. Figure 18 shows the curve calculated from training data next to the test data. This model slightly outperforms the Random Forrest Regressor the MSE is 800 the root MSE is 28mA but the mean and median absolute error is 22mA and 21mA. Although this is somewhat expected as the most important factors in the random forest are also captured in this model.

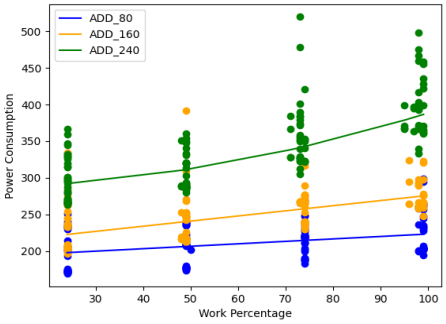


Schaubild 18: All operations work percentage with limited Frequency and test data

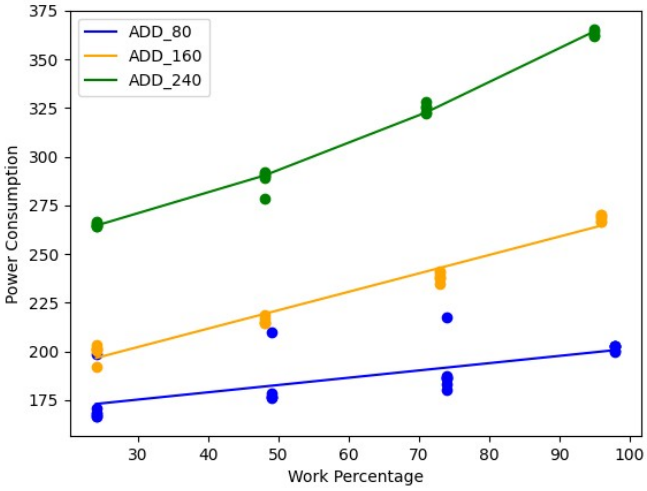
Training Data

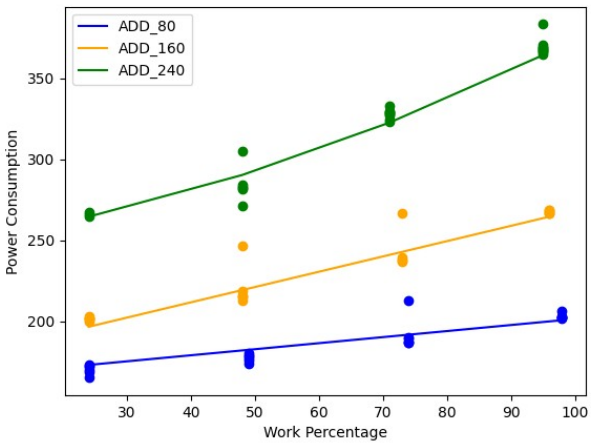
MSE (111+20+9)/3 => 46

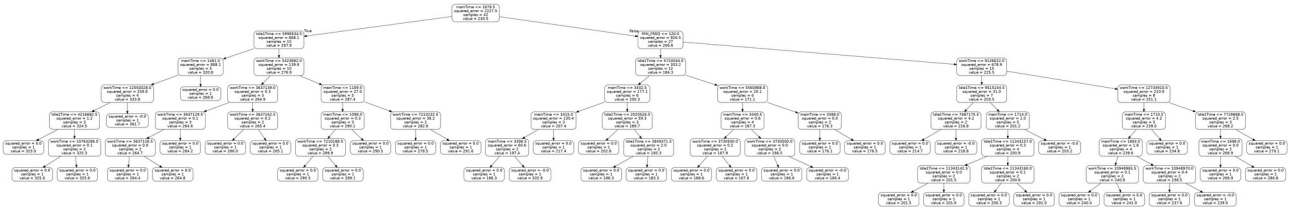
Test Data

MSE = 34, Root MSE => 5, MAE => 4, MEAE => 3

MSE in ratio => 2 percent







Training Data

MSE => 21

Test Data

MSE => 131, root MSE => 11, mae => 6, meae => 4

MSE in ratio => 5 percent