



Pázmány Péter Katolikus Egyetem
Információs Technológiai és Bionikai Kar

Student Research Society report
TDK dolgozat

Electrical Load Classification with Edge Computing solutions

Hálózati fogyasztók klasszifikációja edge computing megoldásokkal

Németh Dániel István
Mérnök informatikus BSc

2022

Témavezető:
dr. Tornai Kálmán

Kivonat

A megújuló energiaforrások elterjedésével egyre nehezebb feladat az elektromos hálózat terhelésének és termelésének kiegyensúlyozása. A folyamatosan változó terhelés mellett a termelés oldalán is megjelentek a nem vezérelhető nap- és szél erőművek. Ezen megújuló energiaforrások kihasználásának maximalizálásához szükség van a hálózat fogyasztói összetételének ismeretére, valamint a fogyasztók vezérlésére, ütemezésére. A hálózat fogyasztóinak ismerete lehetőséget ad a hálózat terhelésének előrejelzésére. Bizonyos fogyasztók működése ütemezésével a hálózat terhelése időzíthető, lehetővé téve a megújuló energiaforrások kihasználásának maximalizálását.

A kutatás korábbi lépéseiben megépítésre került egy mérőeszköz prototípus, mely képes olyan mérési adatokat rögzíteni, amivel különböző klasszifikációs módszerekkel lehetséges a csatlakoztatott fogyasztó pontos felismerése. A mérőeszköz egy dimmer segítségével levágja a fogyasztó feszültségforrását és méri a fogyasztó reakcióját. A mérési adatokat többféle klasszifikációs módszerrel vizsgáltam, ezek közül a legpontosabb a Convolutional Neural Network (CNN) alapú klasszifikáció volt. A kutatás során megvizsgálásra került, hogy kevesebb adattal is el lehet-e érni hasonlóan pontos eredményeket, ehhez kerültek definiálásra mérési profilok. Négy különböző mérési eljárást megvizsgálva, a klasszifikációs pontosság csak kis mértékben romlik. Az eredményekből TDK dolgozat és nemzetközi konferenciára cikk is készült.

A különböző mérési profilok, valamint a különböző klasszifikációs módszerek vizsgálatának célja az volt, hogy mind felhő alapú klasszifikációhoz, mind edge computing megoldásokhoz nyújtson lehetséges megoldást. A mostani TDK dolgozat az edge computing megoldás megvalósítását tartalmazza. Az eredeti prototípus nem hálózaton keresztül továbbította a méréseket, valamint a klasszifikáció is számítógépen futott. Új prototípus eszközt kellett építeni, mely képes egyszerre WiFi-n kommunikálni, a klasszifikációt futtatni és a mérést elvégezni. Az eredeti prototípusban használt ESP32 mikrokontrollerrel nem volt megvalósítható a precíz mérés és WiFi kapcsolat párhuzamosan, így két mikrokontrollerre volt szükség. Az új prototípusban egy RP2040 mikrokontroller felel a mérés elvégzéséért, míg az ESP32 végzi a klasszifikációt és kommunikál WiFi-n keresztül.

A megépített prototípus validálása mérésekkel is megtörtént. Az eredeti kutatásban használt eszközök újra megmérésre kerültek, az adatokból került tanításra a CNN. Ezután a mérőeszközzel újabb méréseket végeztem, ekkor a mikrokontroller végezte a klasszifikációt. Az eredmények alapján a rendszer pontosan képes meghatározni a csatlakoztatott fogyasztót 99% feletti pontossággal. A teljes folyamat kevesebb, mint öt másodpercet vesz igénybe.

Abstract

With the spread of renewable energy sources, load balancing of the grid becomes increasingly difficult. Aside from the constantly changing load of the grid, uncontrollable components such as solar or wind farms appear on the production side as well. In order to maximize the utilization of these renewable energy resources, knowledge about the grid's load composition is required as well as the ability to control or schedule individual loads. Scheduling certain loads enable the grid load to be adjusted to meet renewable energy production levels.

In the previous part of this research, a measurement prototype device was built that records characteristic data of the connected load. This measurement can then be classified with different methods, enabling the accurate detection of the connected load. The prototype device uses an AC dimmer to cut off the voltage supply of the connected load and records the load's response. The measurements were classified with different methods, from which Convolutional Neural Networks (CNN) were the most accurate. During the research, different measurement methods (measurement profiles) were tested to determine whether similarly accurate results are possible with fewer data collected. From the four different measurement profiles examined, I have shown that the accuracy only decreases slightly. These results were published in an international conference paper as well as a TDK report.

The goal is to provide solutions for both edge computing and cloud-based use cases by examining different classification methods and measurement profiles. This TDK report shows a complete edge computing solution with the classification running on the microcontroller. The original prototype used serial communication and the classification was run on a computer. A new prototype device had to be designed and built, which was capable of communicating over WiFi as well as running the measurement and the classification. The ESP32 used in the original prototype could not take accurate measurements while associated to a WiFi AP, so two separate microcontrollers were required. The new prototype uses the RP2040 microcontroller for the measurements, while the ESP32 is responsible for the classification and WiFi communication.

The new prototype's capabilities were validated with real-world measurements. The loads used in the previous stage of the research were measured again, and a CNN was trained. This model was then loaded to the ESP32. Further measurements were taken with the CNN inference running on the prototype. Based on the results, the prototype can accurately classify the connected load with over 99% accuracy. The entire process takes less than five seconds.

Contents

1	Introduction	5
2	Background and previous work	7
2.1	Electrical Load Classification	7
2.2	Edge Computing	9
2.3	TinyML – Machine Learning on microcontrollers	10
2.3.1	TensorFlow Lite Micro [20]	12
2.3.2	TFLM on the ESP32	14
2.4	Previous work – SP4LC	15
2.4.1	Convolutional Neural Network classification results	17
2.4.2	Summary of different classification methods	19
3	New prototype device	20
3.1	Choosing the measurement profile	20
3.2	Initial attempts with ESP32	21
3.2.1	Using ADC1	21
3.2.2	ADC noise	22
3.2.3	Scheduling issues when WiFi is active	23
3.2.4	Attempts with Direct Memory Access	25
3.3	Working prototype with RP2040 and ESP32	25
4	Validation and experimental results	29
5	Conclusions	32
5.1	Future work	32
	Acknowledgements	34

1 Introduction

Renewable energy resources are becoming a significant factor in electricity production. Rooftop solar panels on residential buildings, as well as solar and wind farms, are electricity producers whose output cannot be scheduled. The fluctuating demand and production levels make balancing the electrical grid difficult. To fully utilize the non-controllable renewable resources, the grid load has to be adjusted to match the production levels. This requires control on the load side. This can be done either on the grid level or on the household level with dynamically changing electricity prices. In both cases, automated control is required. Controlling individual loads can be achieved using smart plugs, but current solutions on the market lack load recognition capabilities. These smart plugs can be controlled usually via a smartphone app, and only time-based scheduling is possible.

In my previous research published in [1], a smart plug was built which is capable of cutting off the voltage supply of the load. The device records the load's power consumption during the measurement when different cutoff ratios are applied. This data can then be used to identify the connected load. The prototype communicated over a serial connection, and the classification was done on a computer. In [1], it was shown that it was possible to achieve above 99% accuracy. Different measurement methods (measurement profiles) were examined to determine how accuracy decreases with less data collected. From the measurements with four different measurement profiles, we can conclude that accuracy only decreases slightly with less data. Different classification methods were examined as well as different measurement profiles, to give solutions for both cloud-based classification where there are no resource constraints as well as for edge computing solutions with strict resource constraints.

This TDK report shows a complete edge computing solution with the classification running on the microcontroller. A new prototype device was designed and built, which is capable of communicating over WiFi as well as running the measurement and the classification. The ESP32 used in the original prototype could not take accurate measurements while associated with a WiFi AP, so two separate microcontrollers were required. The new prototype uses the RP2040 microcontroller for the measurements, while the ESP32 is responsible for the classification and WiFi communication.

The new prototype's capabilities were validated with real-world measurements. The loads used in the previous stage of the research were measured again, and a CNN was trained. This model was then loaded to the ESP32. Further measurements were taken with the CNN inference running on the prototype and the prototype communicating over WiFi. Based on the results, the prototype can accurately classify the connected load with over 99% accuracy. The entire process of measurement and classification takes less than five seconds.

Based on my previous results published in [1], it has been invited as an extended paper in a special issue of Springer's *Communications in Computer and Information Science* book series. The submitted extended paper contained results presented in this TDK report as well as further research enabling faster CNN inference on the Smart Plug thanks to optimizations. The paper is under review.

2 Background and previous work

This chapter contains the related literature. Section 2.1 summarizes the related work in Electrical Load Classification. Section 2.2 contains a summary of recent literature related to edge computing, while Section 2.3 summarizes the research in the topic of TinyML, running classification tasks on microcontrollers. Section 2.4 contains the details of the previous steps of this research published in [1].

2.1 Electrical Load Classification

Load scheduling is a key problem that arises with the introduction of non-controllable electricity producers to the grid. In a smart grid, balancing can be achieved by scheduling loads on the demand side, allowing full utilization of renewable resources. Controlling the smart grid's load is critical for renewables' utilization of renewables [2]. Manar Jaradat et al. in [2], examine the problem of Demand-Side Management and model it as a linear programming problem with the goal of fully utilizing the available renewable energy while also minimizing the total cost of electricity purchased from the grid.

Current smart plugs on the market have only limited capabilities. These devices essentially act as relays that can be turned on or off remotely and, in some cases, provide power consumption information. This already has some benefits, especially regarding accessibility [3]. But smart plugs on the market lack load classification capabilities. Industry lags behind research in this area [4]. With current solutions on the market, the user has to individually input the details of the load connected and also has to control the load manually or, in some cases, provide the scheduling information. In [4], environmental sensor data such as temperature, ambient noise, and humidity are combined with smart plugs to create a shared knowledge-based system. With the data from the environmental sensors, the system can estimate whether the load will be used within the next hour or what its power consumption will be. The proposed system, EnAPlug consists of multiple smart plugs communicating with each other, and sharing knowledge.

Load monitoring can be achieved with non-intrusive or intrusive solutions [5]. Non-Intrusive Load Monitoring (NILM) is achieved using Smart Meters. In NILM solutions, the power consumption is measured at a single point, recording the total load of the household. To determine the load composition, disaggregation is required to determine the individual appliances contributing to the total load. Data collected by Smart Meters can help the operation of the Smart Grid by providing load data which can be used to estimate future grid load. Intrusive Load Monitoring (ILM) solutions use multiple measuring devices within the household or building. In [5], ILM is split into three

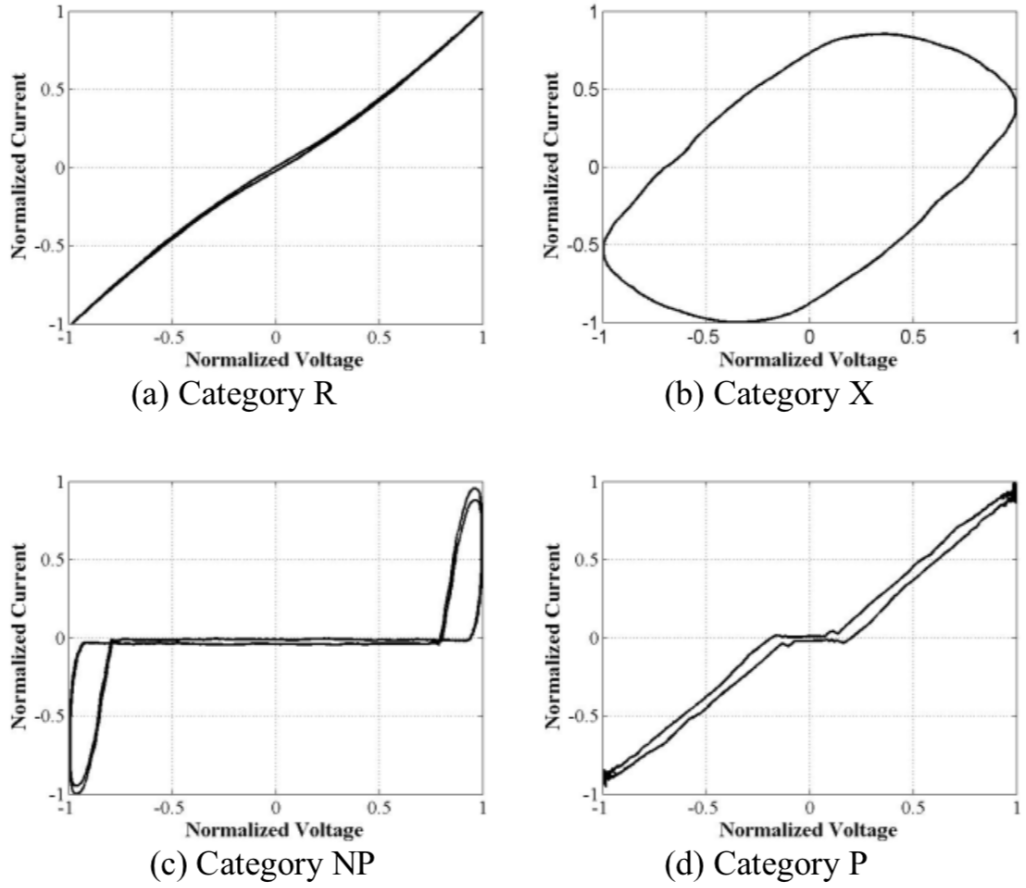


Figure 2.1: Voltage-Current trajectories of different types of loads. Source: [7]

categories: In ILM 1, power consumption levels are measured for each zone within the building or household. In ILM 2, each outlet is measured individually, while in ILM 3, each appliance's load is recorded separately.

The smart home system proposed in [6] features smart plugs which communicate with a central node. By obtaining an Electric Load Signature (ELS) from the power consumption data of the load, the system can accurately classify the connected load. The ELS was obtained by recording the active and apparent power, voltage, current, and power factor values for each load for one hour with one second resolution. With a Decision Tree-based classification method, 93.5% accuracy was achieved. Using a Naive Bayes algorithm, the accuracy rate was 99.9%.

An interesting load classification method is presented in [7] using Voltage-Current trajectories. Different types of loads draw different trajectories, as can be seen in Figure 2.1. Based on the trajectories, the following categories were distinguished:

- Resistive Loads (R)
- Reactive Loads (X)
- Electronic Loads without Power Factor Correction (NP)

- Electronic Loads with Power Factor Correction (P)
- Complex Structure Loads (M)
- Linear Loads (T)
- Phase Angle Controllable Loads (PAC)

After converting the trajectories to binary matrices, a Supervised Self-Organizing Map (SSOM)-based classification was used. An average success rate of over 99% was achieved.

2.2 Edge Computing

In recent years with the increased processing capabilities of end-user devices as well as microcontrollers, it has become possible to run Neural Network-based classifications on the edge. In [8], several advantages are listed:

- Lower latency and improved real-time processing.
- Increased robustness thanks to decentralization. Network failures or cyber attacks have less impact.
- Computational requirements on the server-side are reduced and the volume of data transferred is also decreased.
- Increased privacy and security.

There are several application areas in which edge computing solutions offer advantages over traditional cloud-based solutions:

- Image Processing and Computer Vision tasks can benefit from edge computing in several areas. Decreased latency allows faster reaction times. Increased privacy as only the key information leaves the device, not the entire frame.
- Fast reaction times and network independence are crucial for fully autonomous self-driving vehicles.
- Speech and keyword recognition benefit from increased privacy as well as reduced network bandwidth usage and less computational requirements on the cloud side.
- Smart home systems benefit from all aspects listed previously. Increased response times and reliability can be achieved if the system only relies on local network communication to function.
- Smart infrastructure devices usually communicate over metered networks where data usage reduction can offer significant cost savings, especially with the recent spread of NB-IoT and LTE-M communication technologies. Transmitting less data can also offer energy savings which increases the battery life of the device.

In [9] a unified energy management framework is introduced with the goal of maximizing the utilization of renewable energy for an edge computing platform. There were four functional modules running in the edge device: A Weather Forecast module responsible for processing historical as well as current weather data. An Energy Generation Prediction Module responsible for estimating the solar panel’s future output, the Weather Forecast module’s output is also used as input. An Energy Scheduler Module responsible for choosing the power input of the load. The final module is a Data Processing module responsible for load balancing of the Computing Platform. The proof-of-concept system used Raspberry Pi-s and also included batteries to store the output of the solar panels. For the proof-of-concept system, the only loads used were a cluster of Raspberry Pi Single Board Computers (SBCs). The results published in the paper state that an energy bill reduction of 86% is possible. If their solution were to be extended to household appliances, the smart plug system proposed in this TDK report could be a key component in estimating the future load of the grid as well as enabling load schedule.

Edge computing can also be done on the end user’s already existing devices instead of using dedicated SBCs. One related example is in [10]. A Non-Intrusive Load Monitoring (NILM) solution is proposed running on an android device using the TensorFlow Lite [11] framework. A deep convolutional network (MobileNet) was used for load disaggregation, and after training, the model was exported to TensorFlow Lite, which used post-training quantization. The precision of the learning parameters is lowered from 32-bit floating point values to 8-bit integer values. This conversion affected the model’s accuracy.

In the case of the system proposed in this work, the training of the CNN is done on a computer or server, but there are solutions in literature for running model training on the edge. One example is [12] where a distributed machine learning model is proposed. A distributed gradient descent algorithm is introduced, each node on the edge only accesses a portion of the training samples and uses those for the iteration. After multiple local steps, the nodes communicate with each other to exchange the update information. This method was validated with experiments using Raspberry Pi SBCs as well as computers on a local network. During the validation SVM, linear regression, K-means, and CNN training was performed.

2.3 TinyML – Machine Learning on microcontrollers

TinyML is a relatively new topic. Training and running the inference of Neural Networks requires significant computational capabilities and thus is traditionally run on computers with powerful GPUs. The recent advancements in low-power machine learning solutions, such as for smartphones and the increased computational capabilities of microcontrollers, paved the way for machine learning on microcontrollers.

In [13], several benefits of TinyML are presented:

- Intelligent IoT devices: The large amount of sensor data generated on the device can fully be processed, so data that previously could not be transmitted due to bandwidth or cloud-side processing limitations can be used.

- **Network Bandwidth:** Reducing the required bandwidth that leaves the microcontroller can be beneficial both in terms of power usage and network congestion, especially if there are a large number of IoT devices on the network.
- **Security and Privacy:** Less sensitive data is transmitted to the cloud, which means that the consequences of that data being captured by a third-party are less severe.
- **Latency:** In safety critical systems such as in healthcare or autonomous vehicles, the latency of cloud-based ML is unacceptable and can have serious consequences. TinyML offers independence from network communication and eliminates the latency caused by communicating with a remote server.
- **Reliability:** TinyML can operate even in remote areas where there is no network connectivity.
- **Energy Efficiency:** Most microcontroller-based devices use wireless communication protocols for data transmission. In some cases, the power used for transmitting the data is more than the power required for the microcontroller to run the classification.
- **Potential Crash:** If the cloud-provided services are offline, this impacts the IoT devices running in different households. TinyML solutions have increased independence from the cloud-based parts of the system and if designed correctly, can function even without those services.
- **Data Filtration:** If data processing is done on the IoT device, then transmitting irrelevant data can be avoided. An example given in [13] is a surveillance system used to detect anomalies. Most frames captured by the cameras contain no relevant information, so sending them to the server would be a waste of network bandwidth.
- **Low Cost:** Data usage reduction can offer huge cost savings both on the server side and on the transmission side as well.

There are several recent solutions in literature which use TinyML. In [14], TinyML is used to achieve camera-based autonomous driving with a mini-vehicle. An STM32 MCU was used with GAP8, a parallel low-power SoC. It was compared with an NXP k64f. The LeNet5 CNN architecture was used with modifications. A closed-loop learning method for TinyML applications using tinyCNN models was shown.

In [15], a state-of-the-art keyword spotting solution was developed for microcontrollers using a Legendre Memory Unit-based (LMU) RNN. A Hardware Aware Training (HAT) method was used. In HAT, the quantization of the network parameters for the microcontroller is also considered during the training phase. With this method, the model can be specifically trained to run fast and produce accurate results on the target hardware. Using a smaller network thanks to LMU, P. Blouw et al. in [15] were able to achieve a 14 $\times$ reduction of power usage while maintaining similar accuracy results as other state-of-the-art keyword spotting solutions.

In [16], TinyML is used for signal processing. A deep neural network was run on the microcontroller to achieve orthogonal frequency-division multiplexing signal detection as well as channel estimation. A key part of the research in [16] was reducing the model size as well as the computational complexity by splitting a dense weight matrix into smaller cascading dense matrices. On-device training is also utilized for smaller layers using backpropagation.

Wearable sensors can also benefit from TinyML solutions. In [17], it is shown that the battery lifetime of a wearable activity sensor can be extended from weeks to years by using an SVM-based classification method and thus avoiding sending raw data over BLE. Significant energy savings can be achieved by decreasing the amount of data transmitted. The average power consumption of the nRF51822-based SPHERE wearable activity sensor was $887\mu W$ when raw data was transmitted. Using classification on the microcontroller, the power consumption is reduced to $11.6\mu W$. In [18] an other Human Activity Recognition (HAR) method is examined using an Arduino BLE 33 Sense microcontroller. A one-dimensional Convolutional Neural Network-based model was used to classify the acceleration data. A TensorFlow Lite model was exported and uploaded to the microcontroller. The model achieved up to 97% accuracy. Using the quantization capabilities offered by TFLite to reduce the file size and computational complexity, no reduction in model accuracy was observed.

Typically, Neural Network models use floating-point numbers for calculations, but on some microcontrollers, floating-point operations are not supported in hardware and instead have to be emulated. This causes a significant processing overhead for low-end microcontrollers. To mitigate that, TensorFlow Lite [11] has post-training quantization capabilities, where the model weights can be converted to 16-bit floating point values or 8-bit integer values. For the integer quantization, TensorFlow requires a representative sample dataset to determine the range of values for the hidden layers. Another solution is presented in [19] where SeeDot, a domain-specific language, is introduced for Machine Learning inference algorithms. SeeDot outputs fixed-point C code, which results in significantly faster execution times than using emulated floating-point operations. In [19], FPGA acceleration of SeeDot code is also shown. The output of the SeeDot C compiler and additional optimization methods are used to convert the C code to Verilog code. The performance of SeeDot was evaluated on an Arduino Uno as well as an MKR1000 microcontroller and a Xilinx Arty FPGA board. For microcontrollers, the speedup was in the range of 2.4 to 82.2 times depending on the dataset and model used.

2.3.1 TensorFlow Lite Micro [20]

TensorFlow Lite Micro (TFLM, TFLite Micro) [20] allows existing TensorFlow models to be easily ported to microcontrollers. It was designed with portability in focus to enable its usage on most microcontrollers. An interpreter-based solution is used, and TFLM assumes that the microcontroller has no dynamic memory allocation capabilities or filesystem support. This means that loading the model into memory and providing memory space for TFLM is the programmer's responsibility. TFLM's benefits are listed

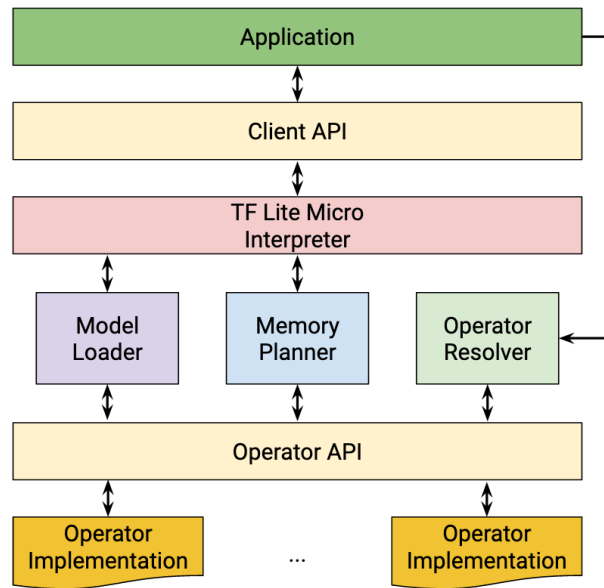


Figure 2.2: Structure of the TensorFlow Lite Micro framework’s implementation. Source: [20]

in [20] as:

- Portable and flexible interpreter-based approach which is easy to adapt to new applications
- Hardware agnostic design by minimizing the usage of external dependencies
- Hardware vendors can add platform-specific optimizations for each kernel
- Kernel optimizations can be easily integrated by hardware vendors
- TFLM utilizes the already existing model conversion and optimization methods in TensorFlow Lite
- Benchmarks that are adopted by benchmark bodies like MLPerf are provided
- Support for popular Google applications that are in production

The structure of the implementation of TFLM can be seen in Figure 2.2. Individual operator implementations can be replaced by ones optimized by hardware vendors for specific platforms. Several optimization solutions are used to minimize the computational and memory overhead caused by using an interpreter-based approach. To create and run a TFLM application, the following steps are involved:

1. The neural network model has to be loaded to memory on the microcontroller.
2. The Tensor Arena has to be supplied to TFLM. The arena is a chunk of contiguous memory that holds all the data required by TFLM. The arena has to be provided since the TFLM library doesn’t assume the microcontroller has dynamic memory

allocation capabilities. Because of this, the arena size has to be determined by trial and error. Luckily, TFLM allocates all required memory during the initialization step from the arena, so the error is raised only during this step.

3. Running the inference. A pointer to the model inputs is needed for this. Then TFLM runs the inference as a blocking function call.
4. After the inference, a pointer to the location of the model's output can be retrieved from TFLM.

2.3.2 TFLM on the ESP32

The ESP32 is supported by the TensorFlow Lite Micro framework. Several constraints need to be considered when designing a neural network to run on a microcontroller. The relevant resource constraints of the ESP32 microcontroller are listed here not only for TFLM, but also for WiFi communication and ADC usage.

CPU

The ESP32 is an Xtensa single- or dual-core 32-bit LX6 microprocessor clocked at 240 MHz [21]. The two cores can theoretically run different tasks simultaneously. However, I encountered some limitations that would suggest that in some cases, the two tasks on separate cores can affect each other. The ESP32 runs a modified version of FreeRTOS [22]. The default configuration of the ESP32 is that CPU Core 0 runs tasks responsible for wireless connectivity (WiFi, Bluetooth) while Core 1 runs the application. The default tick rate of the scheduler is 1000Hz, which means that in every millisecond, the execution of the currently running task is interrupted to run the scheduler. The scheduler then can choose to run other tasks depending on their priorities.

Storage

Most ESP32 boards have 4MB flash storage which can be partitioned depending on the application. Some can be used as SPIFFS (SPI Flash Filesystem), while the other section contains the program code. Part of the storage can be allocated to enable Over-The-Air updates. There are also ways to expand the storage like SD cards.

RAM

The ESP32 has 520KB of SRAM [21]. An external PSRAM (Pseudostatic RAM) module can be connected to add additional memory, however, there are restrictions on its usage compared to the built-in RAM. Because of the 32-bit architecture, the ESP32 can only address, at most the lower 4MB of the PSRAM using the libraries provided with the SDK.

ADC

The ADC is an important feature used in the original measurement prototype device introduced in Section 2.4 for measuring the voltage and current values of the connected load. The ESP32 has two 12-bit SAR (Successive Approximation Register) ADCs, supporting 18 analog enabled pins. [23]. Out of ADC1 and ADC2, ADC2 cannot be used if WiFi is active since ADC2 is used by the WiFi module.

2.4 Previous work – SP4LC

The inspiration for the measurement device came from [24] where a bidirectional triode thyristor was used to manipulate the voltage supply of the load. The response of the load was recorded by an Arduino microcontroller, and the results were classified using a Fully Connected Neural Network.

In SP4LC [1], the measurement method is significantly faster, and we have tried several different classification methods such as SVM, FCNN, LSTM, and CNN. The goal was to examine these methods and provide options for Cloud-based and Edge-based classification solutions. After the initial results with SVM classification, we examined whether it was possible to achieve similarly good results with fewer measurement data. We ran simulations on measurement matrices extrapolated from the original measurements to achieve that. Based on the results, we selected a total of four measurement profiles and collected at least 250 measurements for each load with each measurement profile. Measurement profiles are defined as:

$$\{< r, percentage_min - percentage_max >, h, d\}$$

Where the parameters are the following:

- r - the number of different cutoff ratios
- $percentage_min$ - the minimal cutoff ratio
- $percentage_max$ - the maximum cutoff ratio
- h - the number of cycles the AC signal is cut for each cutoff ratio
- d - the number of cycles where the AC signal is not modified between measuring with two different cutoff ratios

The cutoff ratios are evenly spaced between $percentage_min$ and $percentage_max$.

The following measurement profiles were selected:

- TEST_ORIG : $\{< 14, 10\% - 75\% >, h = 20, d = 16\}$ Measurement time: 488 AC cycles (9.76s)
- TEST_HALVED : $\{< 7, 15\% - 75\% >, h = 10, d = 8\}$ Measurement time: 118 AC cycles (2.36s)

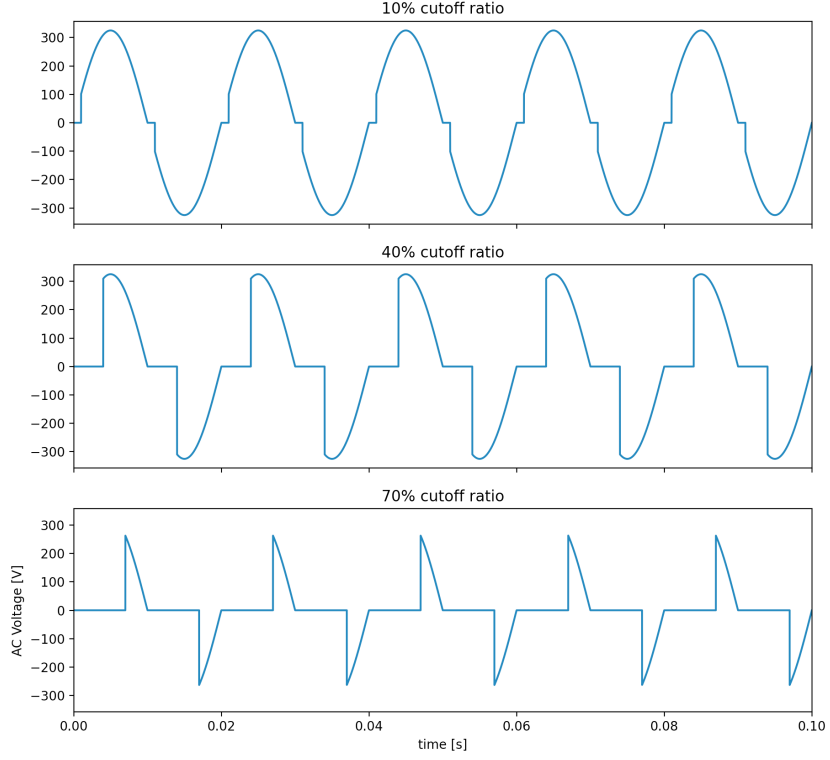


Figure 2.3: Voltage cutoff method with different cutoff ratios.

		1	2	3	4	5	6	7	8	9	h=10	1		d
percentage MIN	15%	62.69	62.90	59.70	63.38	62.01	62.06	62.97	62.64	60.99	62.47	IDLE	...	IDLE
	25%	55.60	60.52	61.32	61.22	61.34	59.41	61.02	60.95	62.02	61.71	IDLE	...	IDLE
	35%	47.64	54.58	54.82	52.98	55.12	53.83	54.91	54.51	54.31	54.95	IDLE	...	IDLE
	45%	36.38	44.05	44.04	44.49	37.23	45.59	43.54	45.25	36.30	35.61	IDLE	...	IDLE
	55%	24.07	30.16	19.20	31.62	19.35	31.71	31.11	19.53	31.17	20.42	IDLE	...	IDLE
	65%	9.29	9.55	16.81	18.09	8.24	9.89	16.07	17.74	17.93	9.55	IDLE	...	IDLE
percentage MAX	75%	4.26	6.96	4.23	4.61	4.38	4.55	1.99	3.36	7.68	3.50	END OF MEASUREMENT		

Figure 2.4: Composition of the Power measurement matrix using the TEST_HALVED measurement profile.

- TEST_TINY : $\{< 2, 10\% - 75\% >, h = 12, d = 4\}$ Measurement time: 28 AC cycles (0.56s)
- TEST_FOUR : $\{< 4, 15\% - 75\% >, h = 8, d = 4\}$ Measurement time: 44 AC cycles (0.88s)

The measurement methodology is the following. For each ratio value (r), the device cuts the voltage supply of the connected load to the given ratio, as can be seen in Figure

2.3. This happens for h number of AC cycles. An AC cycle consists of two consecutive periods between AC voltage zero crossing (ZC) events. After that, the device lets the load receive uninterrupted power for d number of cycles before continuing the measurement with the following cutoff ratio. For each AC cycle during the measurement, three values are calculated: The Real Power, the RMS Voltage, and the RMS Current. It is calculated from several ADC measurement values. The formulas for calculating the values are the following:

$$V_{RMS} = \sqrt{\frac{1}{n} \cdot \sum_{k=1}^n U[k]^2} \quad (2.1)$$

$$I_{RMS} = \sqrt{\frac{1}{n} \cdot \sum_{k=1}^n I[k]^2} \quad (2.2)$$

$$P = \frac{1}{n} \cdot \sum_{k=1}^n U[k]I[k] \quad (2.3)$$

The following devices were measured:

- ipad10W - A 10W Apple USB adapter charging an iPad
- usbapple5V1A - A 5W Apple USB adapter
- usb5V1A - A 5W generic USB adapter
- batterycharger4A - A four ampere 'smart' lead-acid battery charger
- batterycharger800mA - An 800mA traditional lead-acid battery charger
- fan - A fan
- hairdryer - A hairdryer
- incandescentbulb - An incandescent light bulb
- irlamp - An infrared heat lamp
- laptop - A laptop charger charging the laptop
- monitor - An LCD screen
- solderingiron - A soldering iron

A visualization of the measurement profiles and the measurement matrices can be seen in Figure 2.4.

2.4.1 Convolutional Neural Network classification results

CNNs are widely used in literature as well as in industry for image classification tasks. We can interpret the power matrices of the measurements as low-resolution images. A visualisation of different measurement matrices can be seen in Figure 2.5. We can see that for example, the USB loads have similar matrices with a characteristic marking

on the bottom left corner. Since the 3×3 kernel used for the convolution operations was larger than the TEST_TINY matrices' number of rows, only the TEST_ORIG, TEST_HALVED and TEST_FOUR measurement profiles were examined. The CNN had two convolutional layers (48 and 64 features) followed by two fully connected layers (48 and 64 neurons) and a final SoftMax layer. Between the convolutional and the FC layers, SoftMax activation function was used to normalize the output of the convolutional layers.

The network could not distinguish between the incandescent light bulb and the infrared heat lamp. As it turns out, the infrared heat lamp was in fact just an incandescent bulb emitting infrared radiation. This inability to distinguish between different loads with the same internal structure showed that the model's capability to generalize the features. The two incandescent bulbs were used as one common class, the classification accuracy results can be seen in the following table:

Used data Training samples per class	P 150		P, V, I 50	
Accuracy	avg	worst	avg	worst
TEST_ORIG	99.92%	99.56%	99.84%	99.50%
TEST_HALVED	99.90%	99.56%	99.86%	99.44%
TEST_FOUR	99.35%	75.44%	98.81%	82.17%

The average and the worst accuracy is shown for 100 runs of the entire training and inference run. The first method used only the Power matrices and 150 samples for training the model, while for the other model only 50 samples were used, but the input contained the Voltage matrix values (divided by 230) as well as the Current matrix values.

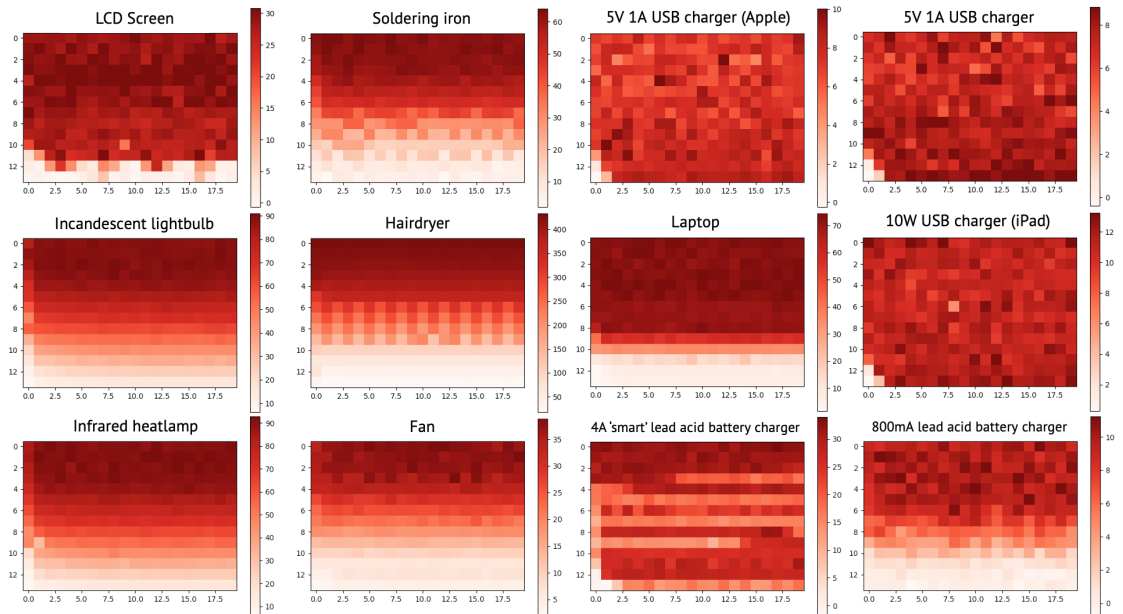


Figure 2.5: Measurement matrices of the examined loads.

Classification method	SVM	FC	CNN	CNN
Number of samples (train-test)	30 - 220	100 - 150	150 - 100	50 - 200
TEST_ORIG	99.56%	99.51%	99.92%	99.84%
TEST_HALVED	98.74%	98.52%	99.90%	99.86%
TEST_TINY	97.40%	97.88%	-	-
TEST_FOUR	99.35%	98.50%	99.35%	98.81%

Figure 2.6: Summary of different classification results. (Average accuracy results, based on 100 runs.)

2.4.2 Summary of different classification methods

The results of different classification methods can be seen in Figure 2.6. We can see that all examined methods can provide accurate results. With less data, the classification accuracy only decreases slightly. In terms of computational complexity, SVM requires the least amount of resources. SVM also required the least amount of samples thanks to the carefully selected features extracted from the matrices. In terms of accuracy, CNNs offer the best solution, but are the most resource-intensive solution.

At first SVM seemed like the ideal solution for classification on the microcontroller while CNN is the ideal solution for cloud-based classification, but as it turns out, with the advances in the computational capabilities of microcontrollers, it is possible to run CNNs on powerful microcontrollers like the ESP32.

3 New prototype device

In Section 2.4.2, I have shown that CNN can achieve the best accuracy results of the methods examined, but it is also the most resource-intensive option. Convolutional Neural Networks are usually trained on GPUs because of that. In an edge-computing scenario, it would be infeasible to run the training of a CNN on a microcontroller. It, however, is possible to run the inference on the microcontroller. In this scheme, the microcontroller would take measurements and run the inference locally. The microcontroller can also send the collected data if needed, along with the type of load classified to the server. Running the inference on the microcontroller makes the entire system more reliable as it can function without a centralized infrastructure. If there is a centralized infrastructure, running the inference locally can improve the reliability of the system as the smart plug does not require a constant internet connection, and intermittent network issues would not affect its operation.

Another consideration is wireless bandwidth and data usage. The transmission of the recognized load's identifier results in significantly less network traffic than sending the entire measurement to the server. In the case of WiFi, this is not significant, but with technologies like NB-IoT and LTE-M, data usage reduction can offer significant cost savings.

For the CNN classification, I used the TensorFlow library [11], which has a variant for edge computing cases, TensorFlow Lite. It is capable of running the inference of the Neural Network on devices such as the Raspberry Pi. There is also TensorFlow Lite Micro [20] used for running the inference of Neural Networks on microcontrollers. This library supports the ESP32 and I used this library for the new prototype device.

3.1 Choosing the measurement profile

TensorFlow models can be exported to files. The execution graph of the model is saved to the file along with the trained weights. Such a file can then be read with TFLite or TFLite Micro, and the inference can be run. The size of this file depends on the size of the Neural Network as well as the optional quantization.

The Convolutional Neural Networks (where only Power measurement values were used) were exported as `.tflite` files. Their sizes were the following:

Measurement profile	Input size	Model size
TEST_ORIG	14×20	2785 KB
TEST_HALVED	7×10	623 KB
TEST_FOUR	4×8	279 KB

It would be difficult to fit the original model on the microcontroller because of its size. The model not only has to fit into the Flash, but it also has to fit into RAM. With the introduction of measurement profiles, we can use a profile that requires a smaller file. The TEST_FOUR and TEST_HALVED can fit into the Flash storage to the SPIFFS filesystem, but in both cases, the onboard RAM would not be enough, so PSRAM is required. Since the original SP4LC prototype used an ESP32 board without PSRAM, another ESP32 board had to be used for the new prototype which featured PSRAM. The new prototype was also necessary due to the WiFi-ADC2 limitation described in Section 2.3.2.

Running the inference requires allocating memory for the tensors (Tensor arena). However, the size of the tensor arena is not calculated from the TFLite model. The TFLite Micro library recommends setting the size of the arena by trial and error. The size of the arena is determined by the number of tensors required in parallel for the inference. Changing the model may require changing the arena size. This needs to be considered if the model is updated Over-The-Air since OTA update of the model is possible thanks to the interpreter-type approach of TFLite Micro.

The inference time also depends on the size of the model, more specifically, the number of calculations required and the precision of the calculations. Running inference with the TEST_FOUR model on the microcontroller takes 350ms. For the TEST_HALVED model, the inference time is around 760ms.

Since the model file has to be loaded to memory, PSRAM is needed as the model as well as the arena would not fit inside the internal RAM. The size of the model also impacts the startup time of the microcontroller. Copying the TEST_HALVED model from SPIFFS to PSRAM takes 10.2 seconds.

The TEST_HALVED measurement profile and model were chosen to run on the microcontroller since they offered better accuracy.

3.2 Initial attempts with ESP32

The initial idea was to use the ESP32 for WiFi connectivity, CNN classification, and for running the measurement as well. This turned out to be problematic. In theory, a task running on one core of the ESP32 should not affect a task running on the other core of the ESP32, however in practice, running the WiFi connectivity on Core 0 and the measurement on Core 1 would result in unacceptable levels of error in the measurements.

3.2.1 Using ADC1

The first problem appeared when I used ADC1 to measure the voltage and current transformer outputs. I noticed that ADC1 seemed to capture fewer samples for one AC period than ADC2. I created a simple program that continuously measured with the ADC in total taking 10000 samples. The measurement times were the following:

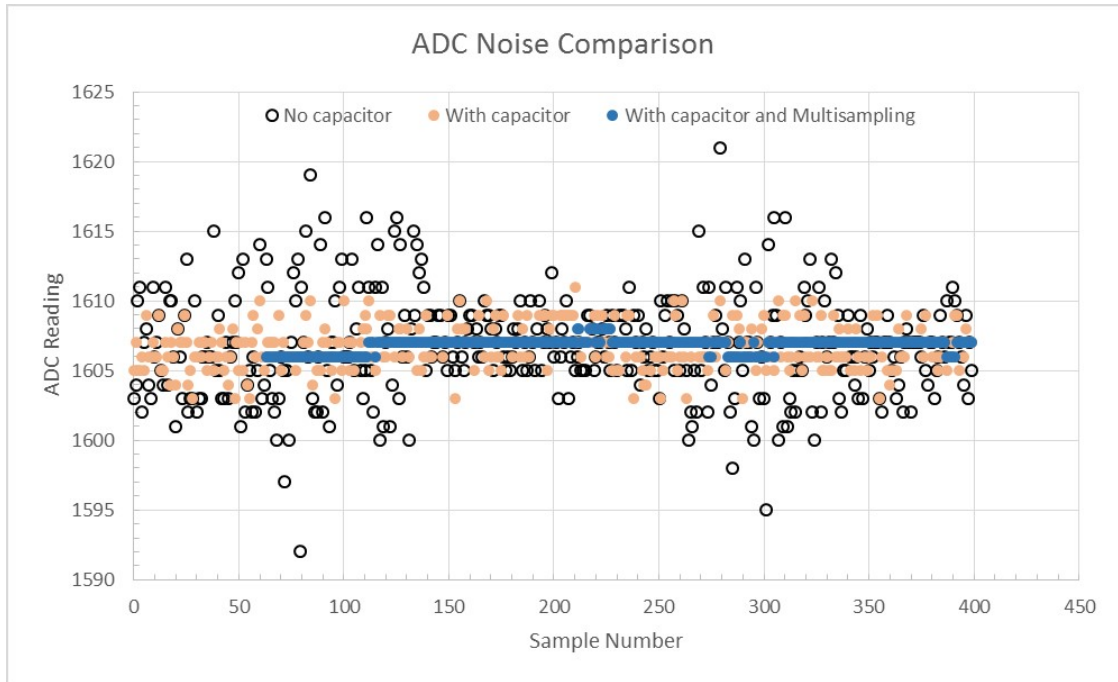


Figure 3.1: ADC noise minimization techniques recommended by Espressif. (Source: [23])

Pin	Measurement time
ADC1 CH7	450 ms
ADC1 CH6	451 ms
ADC2 CH2	306 ms
ADC2 CH3	307 ms

There is a significant speed difference, ADC1 seems about 1.5 times slower than ADC2.

3.2.2 ADC noise

The effective ADC resolution also decreased if WiFi was active on the ESP32. The ADC resolution had to be reduced from 12 bits to 9 bits. From the documentation provided by Espressif (Figure 3.1), we can see that the ESP32 ADC is not very accurate by default. Capacitors were added to the ADC pins, but in our case, multisampling was not an option as in Section 3.2.1 it was already shown that ADC1 measurements took longer than ADC2 measurements.

Additionally, the center value of the ADC readings (the ADC value at 0V for the transformer and at 0A for the Current Transformer) was lower than without using WiFi. It meant that the zero values had to be measured again. As the WiFi driver can use a significant amount of current, it may result in a voltage drop on the supply rail of the ESP32. This may affect the ADC readings as well.

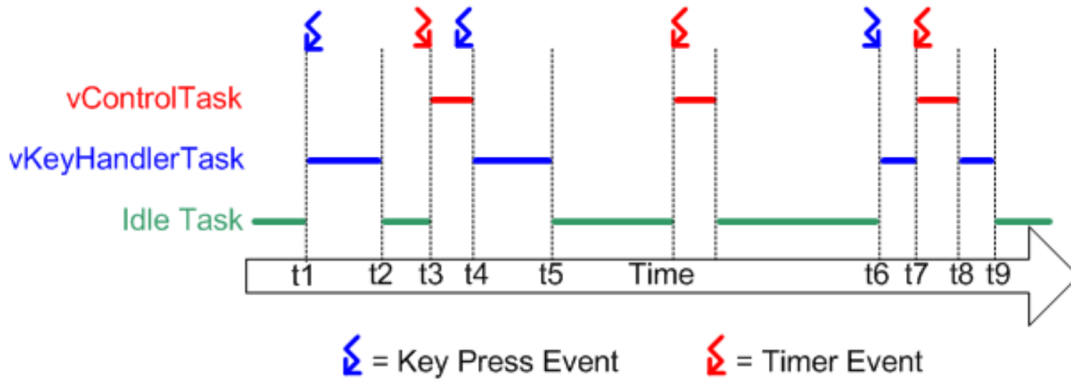


Figure 3.2: FreeRTOS task scheduling example. (Source: [25])

3.2.3 Scheduling issues when WiFi is active

The next problem I noticed is that consecutive measurements between two AC zero crossing events had different amounts of samples. When WiFi was off, this effect was minimal. However, with WiFi active, the difference was clear; there were only a few cases when the same amount of samples was captured, and the difference was as high as ± 10 samples. This asymmetry resulted in measurement values as high as 9W when no load was connected. This level of error is over the acceptable limit.

FreeRTOS runs on the ESP32, and the scheduler runs every 1ms. An example of task scheduling can be seen in Figure 3.2. It can be seen that there can be external triggers and scheduled events that interrupt the currently running task. The exact source of the difference in the number of samples collected is likely due to interrupts or the scheduler. Since suspending the scheduler (calling `vTaskSuspendAll()`) only suspends the scheduler on the core the task is running on [22], running WiFi on Core 0 and the measurement on Core 1 would theoretically not interfere with each other if the WiFi stack suspends the scheduler during critical operations. However, in practice, WiFi also affects the tasks running on Core 1.

To verify that the measurement errors were caused by using WiFi, I ran 10 full measurements, which measured a total of 700 AC cycles. For each AC cycle, data about the first and second parts (separated by a zero crossing event) was collected. The data contained the number of samples collected for the whole cycle and the difference in the number of samples between the first and second parts. This test was run in three scenarios:

- No WiFi: The WiFi was turned off at startup.
- WiFi only: The device was associated with a WiFi AP during the measurement.
- WiFi and Webserver: The device was associated with a WiFi AP during the measurement, and a Webserver was running as well. The tests were run over the serial port of the ESP32; the webserver was in an idle state.

	Number of samples for one AC cycle				Difference of samples between 1st and 2nd part of the AC cycle			
	min	max	mean	stdev	min	max	mean	stdev
No WiFi	242	252	249.67	2.03	0	4	2.07	1.89
WiFi only	242	261	251.06	3.05	0	9	2.36	2.17
WiFi and Webserver	243	261	249.66	3.26	0	9	2.13	2.09

Figure 3.3: ADC sample collection statistics. A total of 700 AC cycles were measured.

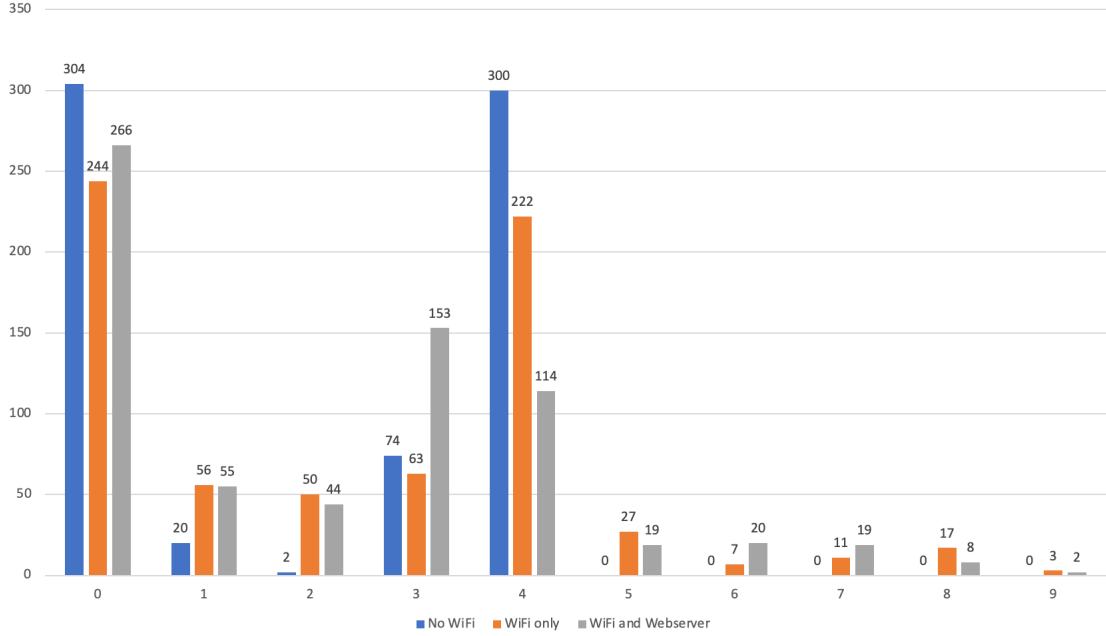


Figure 3.4: Distribution of the difference of ADC samples collected between the two halves of the AC cycle

The measurement results are shown in a table in Figure 3.3. The distribution of the difference of ADC samples collected between the two halves of the AC cycle can be seen in Figure 3.4. It can be seen that the difference when WiFi is turned off is between 0 and 4 samples. The standard deviation of the samples is also small. The distribution changes when WiFi is used. The difference can be as high as 9 samples, meaning that there is non-deterministic behavior introduced when WiFi is active. The standard deviation is also higher. Interestingly, the maximum number of samples collected in one AC cycle is slightly higher when WiFi is active. When the webserver is active, the difference is still high, and the standard deviation of the number of samples for one AC cycle is the highest of the three scenarios. The webserver runs on its own task on the ESP32, so the scheduler may schedule it to run in an undesired time. In theory, the WiFi task running on Core 0 should not interfere with a different task running on Core 1, but as I demonstrated, WiFi affects tasks on the other core as well.

3.2.4 Attempts with Direct Memory Access

So far, I used the built-in `adc1_get_raw()` function to collect one sample from the ADC. This process involved the CPU, but as I have shown in the previous section, using WiFi affects other tasks. A solution to this problem can be to use DMA (Direct Memory Access). DMA doesn't involve the CPU and writes data directly to memory. If the allocated buffer is full, an interrupt is triggered, informing the CPU. The ESP32 has DMA capabilities through its I2S (Inter-IC Sound) interface [26]. An external ADC can be used, but using the internal ADC is also supported. Ideally, DMA can be used to measure faster and at regular intervals without CPU overhead. Nevertheless, as I demonstrate in this section, due to the lack of documentation of ESP32 I2S DMA API and possible bugs in the API, using DMA to collect measurements is unreliable and cannot be used for accurate AC power measurement.

First of all, the only official documentation is in the Espressif ADC documentation page [23] where examples cannot be found. The only official example for using the ADC with DMA is located in the esp-idf GitHub repository [27]. This example program was only added in version 4.4 released on the 26th of January 2022. As of the writing of this work, PlatformIO does not use this version, and the ESP32 Arduino version 2.0.3 based on the esp-idf version 4.4.1 was released only a month ago. The example does not work with previous esp-idf versions. As most ADC DMA codes found in forums use deprecated functions, those could not be used with up-to-date SDK versions. Due to a possible bug in the low-level ADC DMA code of the ESP32, the measurements in the ADC buffer had a random order instead of the pattern defined in the code and the number of samples for the different ADC channels also varied randomly. For this reason, I scrapped the idea of using only the ESP32 for the measurements and instead looked for another microcontroller to run the measurement while the ESP32 is responsible for WiFi communication and running the CNN inference.

3.3 Working prototype with RP2040 and ESP32

The RP2040-based Raspberry Pi Pico microcontroller was used alongside the ESP32 in the new prototype device. The RP2040 [28] is a relatively new microcontroller announced in January 2021 by the Raspberry Pi Foundation. Although it was released recently, it has extensive documentation and software support just like the Raspberry Pi SBCs. The RP2040 can run MicroPython, a high-level Python-based embedded operating system designed for prototyping; and a C SDK is provided for low-level programming. For running the measurement, the execution speed of the code is crucial for fast calculations and ADC measurements, so the C SDK was used instead of MicroPython. The RP2040 features a 12-bit ADC with 3 channels wired to GPIO pins on the Pico board.

For the new prototype, a new dimmer module was used which featured an increased current rating of 8A as well as a built-in power supply capable of powering the prototype. However, this dimmer module behaved slightly differently when running the measurement. With the original method, the microcontroller's output for the AC dimmer stayed

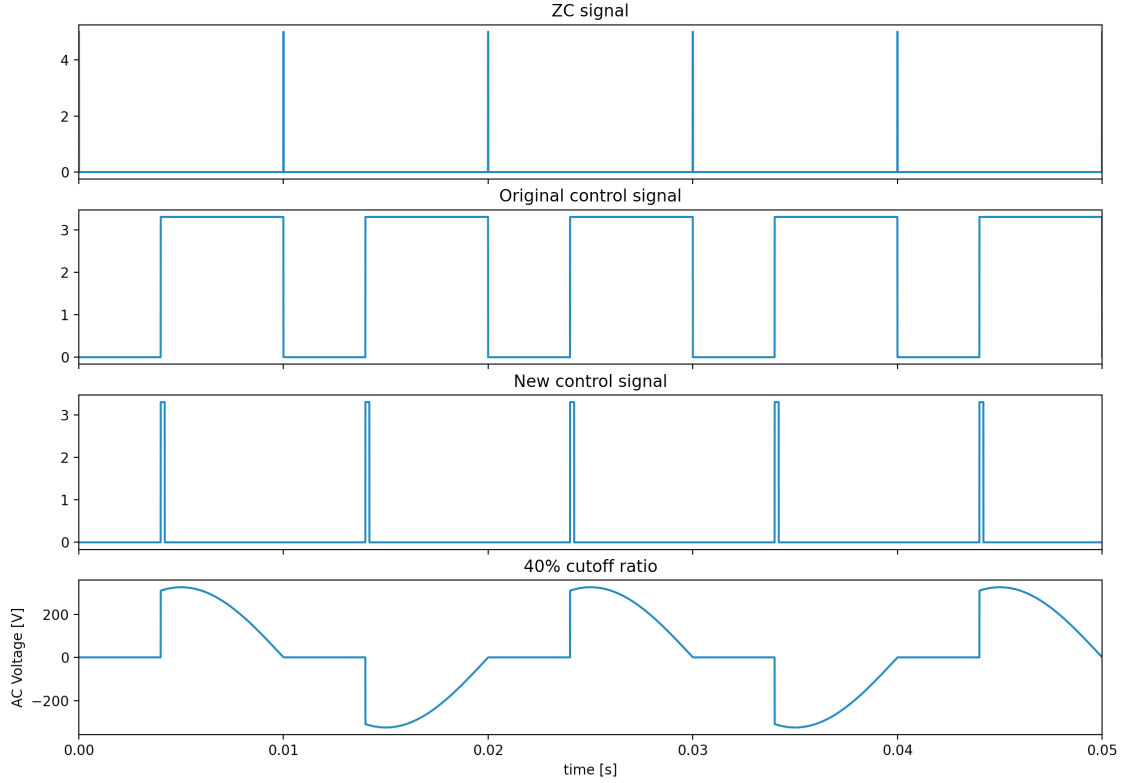


Figure 3.5: Comparison of the dimmer control signals used by the original and new prototype devices. In the old prototype, the output stayed high till the next ZC event. In the new prototype, the output stays on for $500\mu s$ only.

on after the TRIAC was turned on and was turned off at the next ZC event. In the case of the new dimmer, if the control signal for the AC dimmer stayed on until the next ZC event, no dimming was achieved for some loads. To mitigate that, the control signal was changed so that $500\mu s$ after the TRIAC output is activated, the control signal is turned off. This doesn't affect the TRIAC's output until the next zero crossing, since the TRIAC remains in a conducting state. Figure 3.5 shows a comparison of the new and old dimmer control signals. The control signal generation on the RP2040 was achieved by using high-precision timers set when a ZC event occurred.

The RP2040's ADC was much faster than the ESP32's. Even after creating an 8-sample multisampling solution for the minimization of the ADC noise, 179-181 Voltage, Current, and Power samples could be calculated for one half of an AC cycle. In the original prototype, only around 140 samples could be collected and multisampling was not used. This meant that the new prototype's measurements better reflected the actual curves of the signals and thus the actual Power, Current, and Voltage values. The RP2040 is also a dual-core microcontroller as the ESP32. Both cores of the RP2040 were utilized to achieve fast measurements. One core was responsible for running the continuous measurements, while the other core communicated with the ESP32 and processed the measurement data.

The schematic for the new prototype can be seen in Figure 3.6 and a photo of the new prototype can be seen in Figure 3.7. Communication between the RP2040 and the

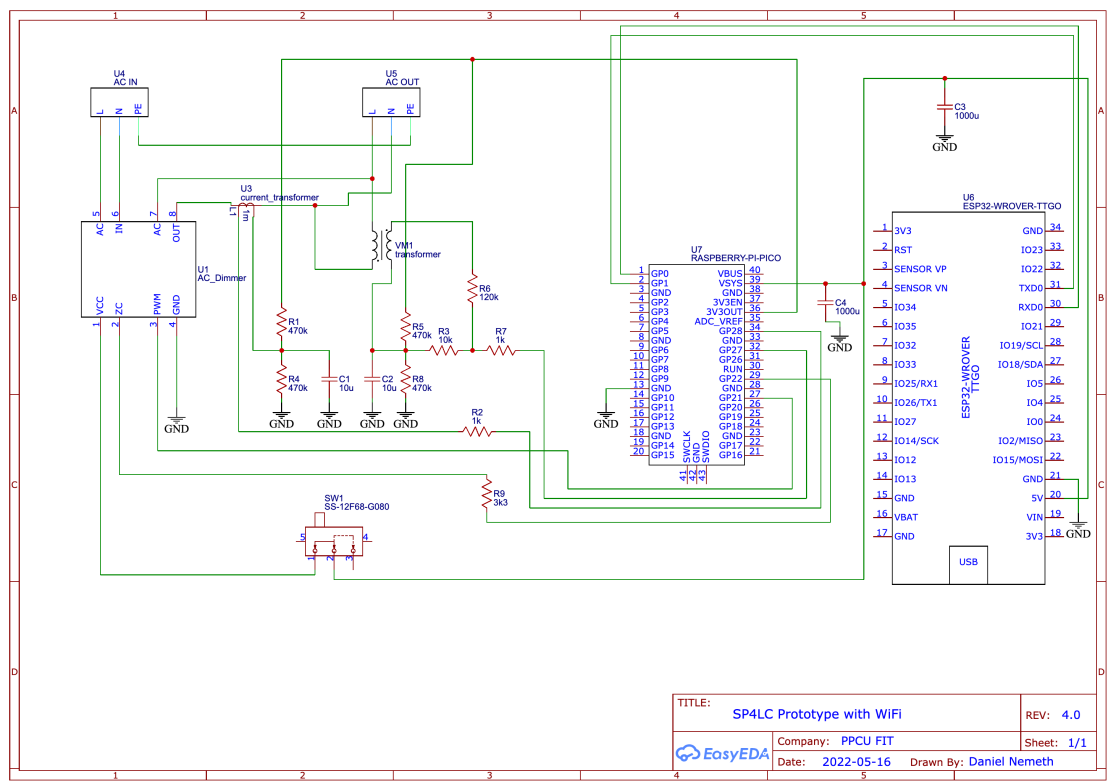


Figure 3.6: Schematic of the new prototype device featuring the Raspberry Pi Pico and the ESP32.

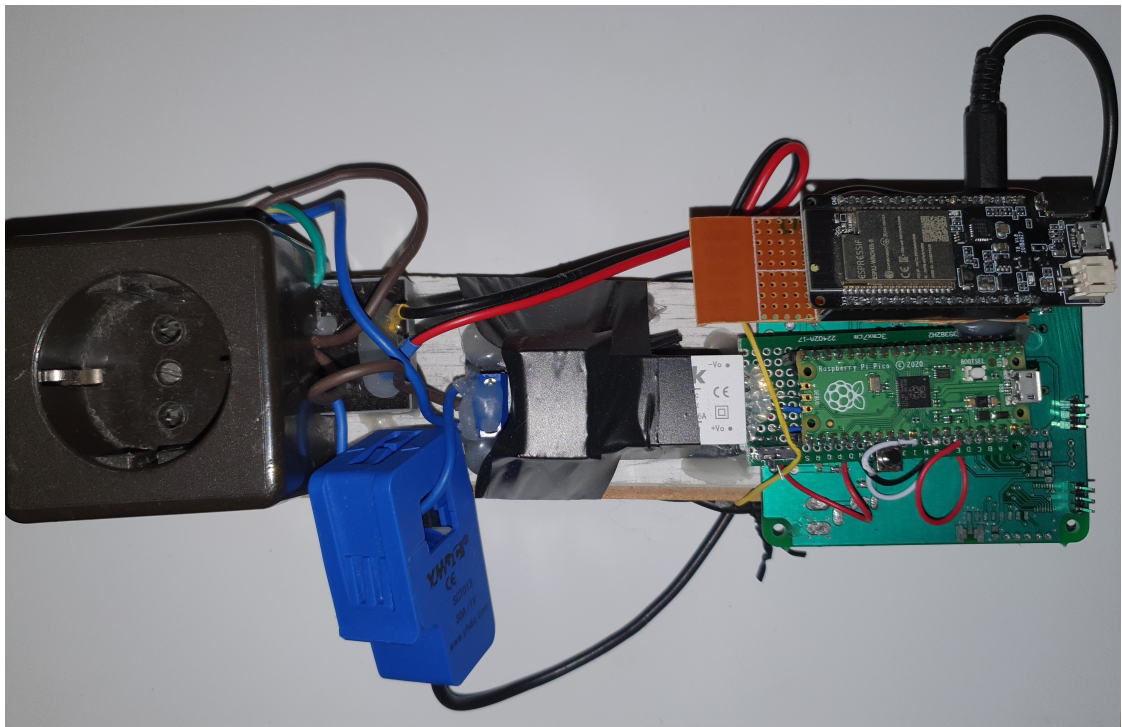


Figure 3.7: New prototype device featuring the Raspberry Pi Pico and the ESP32.

ESP32 is achieved over a serial connection with a Baud rate of 115200. The ESP32 can query the RP2040 for the current power consumption of the connected load and can also turn on or off the output of the dimmer. A measurement can also be initiated by the ESP32. After finishing the measurement, the RP2040 sends the measurement data to the ESP32 via the serial connection. The ESP32 then runs the inference of the CNN model. Since the ESP32 can draw large amounts of current if it uses its WiFi radio, capacitors of $1000\mu F$ capacity were connected to the input voltage sources of both the RP2040 and the ESP32.

After startup, the ESP32 connects to the WiFi AP and runs a webserver. During startup, the TFLite model is loaded into the PSRAM of the ESP32, the tensor arena is allocated and the TFLite Micro library is initialized. The functions of the prototype can be accessed over a REST API as well as a webserver. In both cases, the ESP32 responds to HTTP requests made to specific endpoints: A measurement can be taken with or without classification, the current power of the connected load can be accessed and the load can be turned off and on remotely.

4 Validation and experimental results

To validate that the entire system works, I first took measurements with the new prototype device running the TEST_HALVED measurement profile. 250 measurements were taken with each of the loads introduced in Section 2.4. Then the model was trained. As in the case of the CNN classification, a common USB class was used as well as a common incandescents class. So the classes were the following:

- USB : ipad10W, usbapple5V1A, usb5V1A
- batterycharger4A
- batterycharger800mA
- fan
- hairdryer
- incandescents: incandescentbulb, irlamp
- laptop
- monitor
- solderingiron

Of the 250 samples for each load, 50 were used for testing and 200 were used to train the network. The network achieved 100% accuracy on the 50 test samples for each load. The trained model was exported and uploaded to the ESP32. To verify that the power consumption of the connected load contributed only slightly to the 100% accuracy rate, I ran the training also with normalized inputs, where each matrix was divided by its mean value. The results of this classification are shown in Figure 4.1. We can see that the average accuracy is 98.27%, while the worst accuracy was 92.89% from 10 runs of the classification. This means that even without the actual power values, the CNN method can distinguish different loads accurately.

For the validation of the actual system, one load from each class was measured again. These measurements were done on a different day than the measurements used to train the CNN model. For each load measured, 100 measurements were taken and the ESP32 ran the classification. The inference time on the ESP32 was in the range of 758ms - 801ms. Since the measurement with the TEST_HALVED profile takes 2.36s, the entire process from measurement to inference can be done in under 3.5 seconds. The confusion matrix of the validation can be seen in Figure 4.2. The accuracy is 99.44%. We can see that of the 900 total samples measured, 895 were correctly classified. Examples of the measurement matrices recorded during the validation are shown in Figure 4.3.

	USB	batterycharger4A	batterycharger800mA	fan	hairdryer	incandescents	laptop	monitor	solderingiron
USB	50	0.4	0	0	0	0	0	0	0
batterycharger4A	0	49.2	0	0	0.3	0	0	0.6	0
batterycharger800mA	0	0	50	0	0	0	0	0	0
fan	0	0	0	49	0	0	0	0	5.1
hairdryer	0	0.1	0	0	49.7	0	0	0	0
incandescents	0	0	0	0	0	50	0	0	0
laptop	0	0.3	0	0	0	0	50	0	0
monitor	0	0	0	0	0	0	0	49.4	0
solderingiron	0	0	0	1	0	0	0	0	44.9

Figure 4.1: Confusion matrix (average of 10 runs) of CNN classification with each input matrix divided by its mean value. The average accuracy is 98.27%, the worst accuracy was 92.89%.

	USB	batterycharger4A	batterycharger800mA	fan	hairdryer	incandescents	laptop	monitor	solderingiron
USB	100	0	0	0	0	0	0	0	0
batterycharger4A	0	98	0	0	0	0	0	3	0
batterycharger800mA	0	2	100	0	0	0	0	0	0
fan	0	0	0	100	0	0	0	0	0
hairdryer	0	0	0	0	100	0	0	0	0
incandescents	0	0	0	0	0	100	0	0	0
laptop	0	0	0	0	0	0	100	0	0
monitor	0	0	0	0	0	0	0	97	0
solderingiron	0	0	0	0	0	0	0	0	100

Figure 4.2: Confusion matrix of CNN classification run on the prototype device. The accuracy is 99.44%

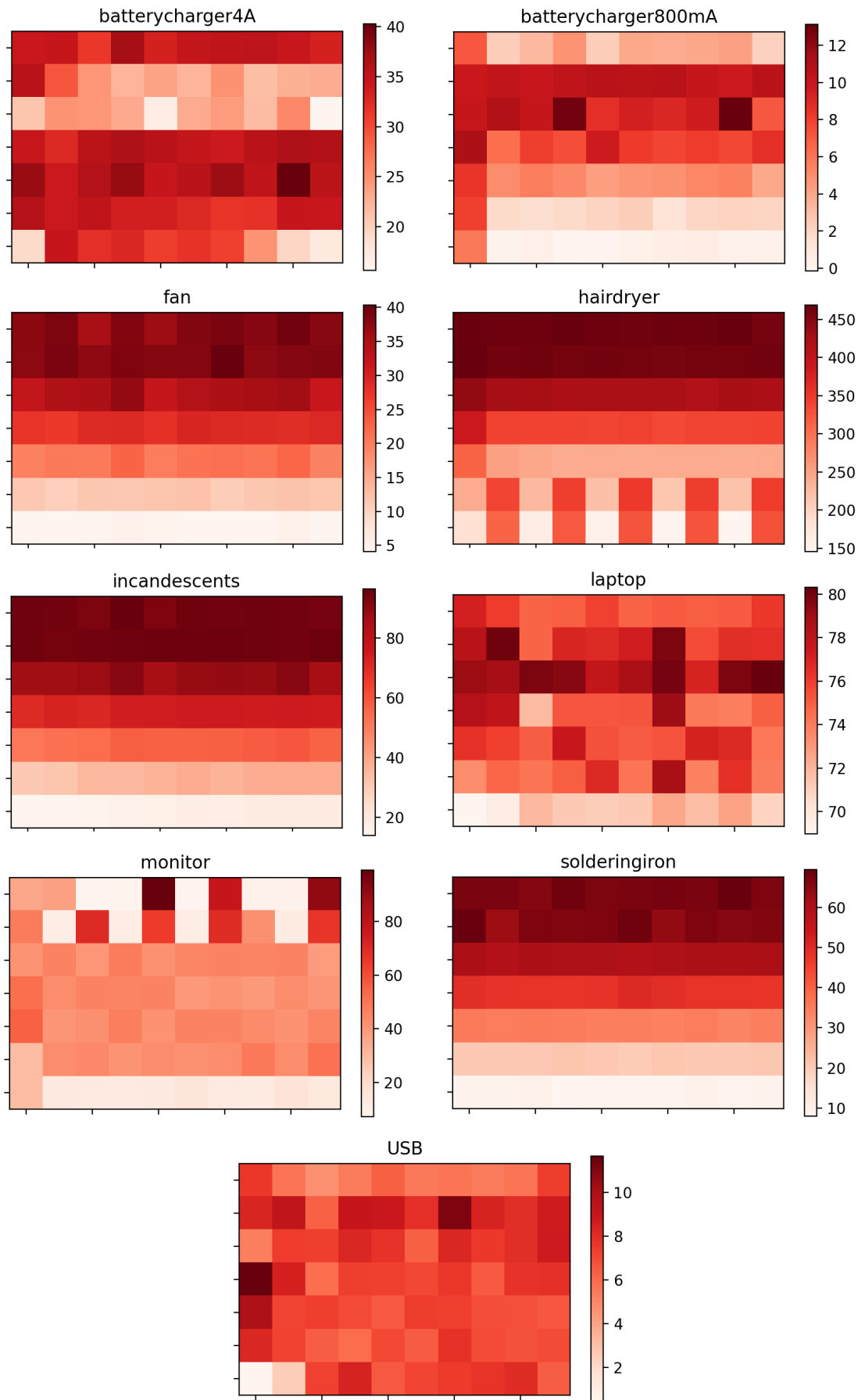


Figure 4.3: Measurement matrices for different loads measured during the validation.

5 Conclusions

In this work, I have shown a working prototype of a smart plug utilizing edge computing and TinyML technologies. In less than 3.5 seconds, the device can determine the type of load connected to the prototype. The accuracy rate is 99.44%. It is important to emphasize that the full CNN inference runs on the resource-constrained environment presented by the microcontroller. To achieve that, all measurement profiles had to be examined and the model with the best accuracy was chosen from the ones that were possible to run on the ESP32.

With a fast response time, a smart grid system can quickly react to the new load connected. Controlling the load is also possible via WiFi. The smart plug keeps an active WiFi connection so communication with it is always possible. The entire prototype is powered using the power supply inside the AC dimmer module. The plug uses the relatively new RP2040 microcontroller for controlling the measurement, while the ESP32 is responsible for running the CNN inference and WiFi connectivity. Communication between the microcontrollers is achieved via a serial connection. The ESP32 provides a webserver for controlling the measurement and the load.

Running the classification on the smart plug reduces network traffic as well as processing requirements on the server-side. The prototype can function independently, no server or internet connection is required which increases the robustness of the system. Local decisions made by the smart plug are also possible as the load is classified on the plug.

5.1 Future work

Several optimization options are planned to be examined in the future. The model quantization options in TFLite were not utilized for the prototype, but using them may offer classification time reductions and reductions in storage and memory requirements. The performance of these quantization options can be examined to determine how they affect the accuracy of the classification. Methods such as Neural Architecture Search (NAS) can also be utilized to find other CNN-based model structures that can achieve the same level of accuracy with less computational complexity.

Other microcontrollers may also be considered for faster NN inference. The recently released ESP32-S3 microcontroller has vector instruction capabilities with which TFLite Micro inference can be optimized resulting in significant speedups as shown in [29].

Another area planned for future research is CNN-based Open Set Recognition. An Open Set Recognition-based method like OpenMAX enables the detection of samples that do not belong to any class used for training the model. This would allow the

network to detect previously unseen loads. If it could be implemented on the prototype, then the smart plug could detect if a previously unseen load is connected. This would allow the system to automatically profile the new device by taking measurements and the user would only be prompted to label the load and enter the scheduling information.

Acknowledgements

Supported by the ÚNKP-21-1 new national excellence program of the ministry for innovation and technology from the source of the national research, development, and innovation fund.



Based on my previous results published in [1], it has been invited as an extended paper in a special issue of Springer's *Communications in Computer and Information Science* book series. The submitted extended paper contained results presented in this TDK report as well as further research enabling faster CNN inference on the Smart Plug thanks to optimizations. The paper is under review.

Bibliography

- [1] D. Németh. and K. Tornai., “Sp4lc: A method for recognizing power consumers in a smart plug”, in *Proceedings of the 11th International Conference on Smart Cities and Green ICT Systems - SMARTGREENS*, INSTICC, SciTePress, 2022, pp. 69–77, ISBN: 978-989-758-572-2. DOI: 10.5220/0010982800003203.
- [2] M. Jaradat, M. Jarrah, Y. Jararweh, M. Al-Ayyoub, and A. Bousselham, “Integration of renewable energy in demand-side management for home appliances”, in *2014 International Renewable and Sustainable Energy Conference (IRSEC)*, 2014, pp. 571–576.
- [3] P. Mtshali and F. Khubisa, “A smart home appliance control system for physically disabled people”, in *2019 Conference on Information Communications Technology and Society (ICTAS)*, 2019, pp. 1–5.
- [4] L. Gomes, F. Sousa, and Z. Vale, “An intelligent smart plug with shared knowledge capabilities”, *Sensors*, vol. 18, no. 11, p. 3961, Nov. 2018. DOI: 10.3390/s18113961.
- [5] A. Ridi, C. Gisler, and J. Hennebert, “A survey on intrusive load monitoring for appliance recognition”, in *2014 22nd International Conference on Pattern Recognition*, 2014, pp. 3702–3707.
- [6] A. F. da S. Veloso, R. G. de Oliveira, A. A. Rodrigues, R. A. L. Rabelo, and J. J. P. C. Rodrigues, “Cognitive smart plugs for signature identification of residential home appliance load using machine learning: From theory to practice”, in *2019 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2019, pp. 1–6.
- [7] L. Du, D. He, R. G. Harley, and T. G. Habetler, “Electric load classification by binary voltage–current trajectory mapping”, *IEEE Transactions on Smart Grid*, vol. 7, no. 1, pp. 358–365, 2016.
- [8] M. G. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, “Machine learning at the network edge: A survey”, *ACM Comput. Surv.*, vol. 54, no. 8, Oct. 2021, ISSN: 0360-0300. DOI: 10.1145/3469029. [Online]. Available: <https://doi.org/10.1145/3469029>.
- [9] X. Chang, W. Li, C. Xia, *et al.*, “From insight to impact: Building a sustainable edge computing platform for smart homes”, in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, Dec. 2018, pp. 928–936. DOI: 10.1109/PADSW.2018.8644647.

- [10] S. Ahmed and M. Bons, “Edge computed nilm: A phone-based implementation using mobilenet compressed by tensorflow lite”, in *Proceedings of the 5th International Workshop on Non-Intrusive Load Monitoring*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 44–48, ISBN: 9781450381918. [Online]. Available: <https://doi.org/10.1145/3427771.3427852>.
- [11] Martin Abadi, Ashish Agarwal, Paul Barham, *et al.*, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: <https://www.tensorflow.org/>.
- [12] S. Wang, T. Tuor, T. Salonidis, *et al.*, “When edge meets learning: Adaptive control for resource-constrained distributed machine learning”, in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, Apr. 2018, pp. 63–71. DOI: 10.1109/INFOCOM.2018.8486403.
- [13] D. L. Dutta and S. Bharali, “Tinyml meets iot: A comprehensive survey”, *Internet of Things*, vol. 16, p. 100461, 2021, ISSN: 2542-6605. DOI: <https://doi.org/10.1016/j.iot.2021.100461>. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2542660521001025>.
- [14] M. de Prado, M. Rusci, R. Donze, *et al.*, “Robustifying the deployment of tinyML models for autonomous mini-vehicles”, in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, May 2021. DOI: 10.1109/iscas51556.2021.9401154. [Online]. Available: <https://doi.org/10.1109%5C%2Fiscas51556.2021.9401154>.
- [15] P. Blouw, G. Malik, B. Morcos, A. R. Voelker, and C. Eliasmith, *Hardware aware training for efficient keyword spotting on general purpose and specialized hardware*, 2020. DOI: 10.48550/ARXIV.2009.04465. [Online]. Available: <https://arxiv.org/abs/2009.04465>.
- [16] H. Liu, W. Ziping, H. Zhang, B. Li, and C. Zhao, “Tiny machine learning (tinyml) for efficient channel estimation and signal detection”, *IEEE Transactions on Vehicular Technology*, pp. 1–1, 2022, ISSN: 1939-9359. DOI: 10.1109/TVT.2022.3163786.
- [17] X. Fafoutis, L. Marchegiani, A. Elsts, J. Pope, R. Piechocki, and I. Craddock, “Extending the battery lifetime of wearable sensors with embedded machine learning”, in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, Feb. 2018, pp. 269–274. DOI: 10.1109/WF-IoT.2018.8355116.
- [18] A. Ghibellini, L. Bononi, and M. Di Felice, “Intelligence at the iot edge: Activity recognition with low-power microcontrollers and convolutional neural networks”, in *2022 IEEE 19th Annual Consumer Communications Networking Conference (CCNC)*, Jan. 2022, pp. 707–710. DOI: 10.1109/CCNC49033.2022.9700665.
- [19] S. Gopinath, N. Ghanathe, V. Seshadri, and R. Sharma, “Compiling kb-sized machine learning models to tiny iot devices”, in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*,

- ser. PLDI 2019, Phoenix, AZ, USA: Association for Computing Machinery, 2019, pp. 79–95, ISBN: 9781450367127. DOI: 10.1145/3314221.3314597. [Online]. Available: <https://doi.org/10.1145/3314221.3314597>.
- [20] R. David, J. Duke, A. Jain, *et al.*, “Tensorflow lite micro: Embedded machine learning for tinyml systems”, in *Proceedings of Machine Learning and Systems*, A. Smola, A. Dimakis, and I. Stoica, Eds., vol. 3, 2021, pp. 800–811. [Online]. Available: <https://proceedings.mlsys.org/paper/2021/file/d2ddea18f00665ce8623e36bd4e3c7c5-Paper.pdf>.
 - [21] Espressif, *Esp32 datasheet*. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.
 - [22] —, *Esp32 freertos*. [Online]. Available: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html>.
 - [23] —, *Esp32 adc*. [Online]. Available: <https://docs.espressif.com/projects/esp-idf/en/v4.2/esp32/api-reference/peripherals/adc.html>.
 - [24] T. Petrović and H. Morikawa, “Active sensing approach to electrical load classification by smart plug”, in *2017 IEEE Power Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, 2017, pp. 1–5.
 - [25] FreeRTOS, *Real time scheduling*. [Online]. Available: <https://www.freertos.org/implementation/a000008.html>.
 - [26] Espressif, *Inter-ic sound (i2s)*. [Online]. Available: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/i2s.html>.
 - [27] E. A. D. Example, *Inter-ic sound (i2s)*. [Online]. Available: https://github.com/espressif/esp-idf/tree/master/examples/peripherals/adc/dma_read.
 - [28] R. P. Foundation, *Rp2040*. [Online]. Available: <https://www.raspberrypi.com/products/rp2040/>.
 - [29] Espressif, *Optimised neural network functions for espressif chipsets*. [Online]. Available: <https://github.com/espressif/esp-nn>.