Preprint: Architecture of an AI on Device Board for Local, Automated Training and Inference

Marvin Schacht*, Julien Murach*, Rüdiger Machhamer*, Achim Guldner*,
Lejla Begic Fazlic*, Lars Creutz*, Stefan Naumann*, Klaus-Uwe Gollmer* and Guido Dartmann*
*Institute for Software Systems (ISS), Trier University of Applied Sciences,
Birkenfeld, Germany

***Abstract*—This paper presents the conceptualization and implementation of an EdgeAI board prototype consisting of a high-power model-node for training of AI models and a low-power sensing-node in the form of a TinyML for model inference and data collection, coupled with a shared storage unit for data exchange. The usage of AI applications in wide-ranging areas, interspersed with different infrastructures, causes the increasing need for grid expansion and performance. Furthermore, with increasing complexity and the resulting cost of energy, data, and hardware requirements, assessments of resource- and energy-efficiency are often neglected. In this paper, we address these emerging problem areas with innovative AI hardware solutions. In this paper we demonstrate an AI on Device implementation without external transfer requirements, present the hardware and software concept and implement an exemplary use-case. It is shown how these methodologies achieve a significant gain in energy and resource efficiency while maintaining the desired performance. Furthermore the transfer of existing methodologies to an AI on Device model is demonstrated to show the adaptability and interoperability of different application areas of the AI sector. The result of this work is an energy-efficient cyber-physical system, combining sensory hardware with AI capabilities on a singular device.**

I. Introduction

With rapid advancements in the AI-sector, and growing capabilities in the field of sensory data for AI usage, we determine a growing need of energy efficient hardware. Furthermore, with an expanding application field in areas without network coverage, arises also the need for local hardware without the necessity of data transmission. These considerations lead to the creation of our AI on Device board. The components of the board consist of a low-power sensing-node and a higher-power model-node. The combination of these two nodes enable the energy-efficient creation of locally executing machine learning models with minimal data transmission. The board is intended as an example of an implementation for a cost-effective and intuitive AI on Device system. In this implementation, we show how an energy-efficient implementation for local ML-computation can be achieved with intuitive operation for the user. The sensing-node serves as a unit for recording multimodal time series data and for inference of the AI models. The model-node creates new models for classification tasks or regression analyses at user-defined intervals. The models are calculated on the basis of either previously recorded or in real time collected, locally stored measurement data.

The implementation of AI procedures on a low-power node is generally referred to as TinyML [1]. In previous applications, external hardware is used to create and check TinyML models. The resulting model is then transferred to the TinyML board for inference and put into operation. If continuous further development of the model is desired, the TinyML board must often transfer the data to the external hardware [2]. The external hardware then generates a new AI model based on the measured data, which is then again transferred to the TinyML-board and so on.

If operations are focused in remote and rural areas like forests or in the field, there is often a lack of resources, such as energy supply, and available technologies for transmitting data, due to poor network coverage. There are also situations in which privacy is key and network data transfer is best avoided at all, like in the case of safety-critical machine halls. We therefore developed an AI on Device system that is self-sufficient and therefore independent of external devices and services (cloud, transmission technologies, energy suppliers), independently calculates new AI models with low-complexity algorithms, and continuously develops the models further. Care was taken to ensure that the energy requirement is low. This is achieved, for example, by only activating the model-node when the project is initialized or changed, and when new models are calculated. The simple installation and dispersion of new AI on Device nodes also makes it easier to implement distributed learning applications. Here, the surrounding distributed nodes exchange their sub-models with each other in order to create improved and more effective new models. This strengthens the privacy of AI processes, as the raw data remains local to the end node [3]. Another focus lies on the use of the board for university educational purposes. It should therefore be designed in such a way that its didactic use is made possible by low entry barriers. So that students can use existing example projects to independently understand the effect of the respective parameters of the different algorithms and implement their own practical AI projects.

A. Related Work

TinyML and AI on Device are two emerging research areas in the field of machine learning and artificial intelligence. While TinyML focuses on implementing machine learning models on resource-constrained devices, AI on Device focuses

on processing data and executing AI algorithms on the device itself, without relying on a constant internet connection or cloud infrastructure.

In recent years, there has been a significant surge in research focused on deploying machine learning models on resource-constrained devices, commonly known as TinyML. R. Study [4] present a comprehensive approach to TinyML, offering techniques to execute complex machine learning models on small, energy-efficient devices. Their work delves into optimization methodologies such as quantization, binarization, and clustering, aimed at enhancing performance and energy efficiency for small Internet of Things (IoT) devices.

A thorough survey [2] on AI on Device provides a panoramic view of research endeavors and practical applications in this domain. Their survey encompasses various AI on Device architectures, techniques, and challenges encountered in implementing AI on edge devices. Moreover, they examine application domains such as intelligent sensor networks, autonomous vehicles, and robotics, elucidating the diverse landscape of AI on Device applications.

An extensive overview of machine learning applications within embedded and mobile devices, emphasizing distinct optimization strategies and application domains is offered by [5]. Their study systematically categorizes optimization techniques and explores application areas, providing insights into the wide-ranging impact of machine learning on embedded and mobile platforms.

In the realm of distributed computing paradigms, fog computing has emerged as a prominent framework for accommodating the needs of IoT and data-intensive applications. A foundational overview of fog computing, delineating its principles, architectures, and applications is provided by [6]. They elucidate the advantages of fog computing over conventional cloud-based approaches and discuss various architectures and technologies pivotal in fog computing environments. Furthermore, they highlight numerous application domains wherein fog computing mitigates the challenges posed by IoT and data-intensive applications, underscoring its significance in contemporary computing landscapes.

Other popular boards for edge computing include the Jetson Nano from Nvidia and the Coral Dev Board from Google. These and other boards were reviewed in the paper ”DeepEdgeBench: Benchmarking Deep Neural Networks on Edge Devices” [7] in application scenarios in connection with neural networks. This work deals with the benchmarking of deep neural networks on edge devices. It examines the performance of different models in terms of real-time processing, accuracy and energy efficiency. The goal is to establish a standardized method for benchmarking and to improve the development of AI applications on edge devices. By examining other work, it has been found that these single board systems make up the largest share of the edge computing environment. A variety of microcontrollers can be used to infer ML models. Commonly used boards include the Sparkfun Edge, Arduino Nano 33 BLE Sense, the STM32F Discovery and Apollo3. These and other boards, as well as supporting TinyML software, are presented and examined in the paper by [1].

II. Concept

A. Cloud-Edge-Continuum

B. Approach

Parts of this work have been investigated in the Master thesis of the second author [8]. The entire system is divided into two subsystems, as shown in Figure 1. The first subsystem, the sensing-node (blue), is used to record sensor data streams, to infer the TinyML models, to control the whole system, and to communicate with neighbor nodes. A management and computation unit, the model-node (yellow), is used as the second subsystem. This is used to initialize and monitor the process and to train and compute TinyML models. To increase energy efficiency, the model-node is only activated when necessary. This includes activities such as automatic further recalculation and training of the AI models or user interaction with the system. To simplify data exchange between the nodes, the units have a common storage unit. The measurement and configuration files of the projects are stored in this unit.

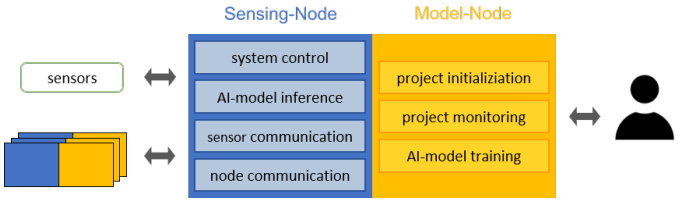

Fig. 1. Task distribution of the EdgeAI-board subsystems

Figure 2 shows the concept of the general program flow of the AI on Device board. The path shown in yellow corresponds to the tasks of the model-node, the blue path to those of the sensing-node. The project initialization takes place on the model-node. In this step, the user transfers all necessary data to the model-node. This includes a unique project name and the application code for the sensing-node and the model-node. The code for the model-node consists of the algorithm for creating and testing the model. In order to build the first model, data must also be provided by the user. The code for the sensing-node contains the recording and storage of the sensor data streams, the inference of the model and the trigger events for the new model building process. In the case of a federated approach the sensing-node would also manage inter-node communication. In the next step, the first AI model of the project is created using this initial data. Subsequently, the code containing the AI model is compiled and transferred to the sensing-node.

After the program has been transferred the sensing-node operates as the master of the system and manages the communication between the subsystems and external nodes. When the project has been initialized, the model-node only observes the communication interface until an instruction from the sensing-node arrives. If the model-node is neither used by the user nor the sensing-node, it is completely switched off to save energy. The program routine continues on the sensing-node. In case of idle times, the sensing-node can also be set to sleep or deep sleep mode to further save resources. If a user-defined event occurs during the processing of the routine that requires a new

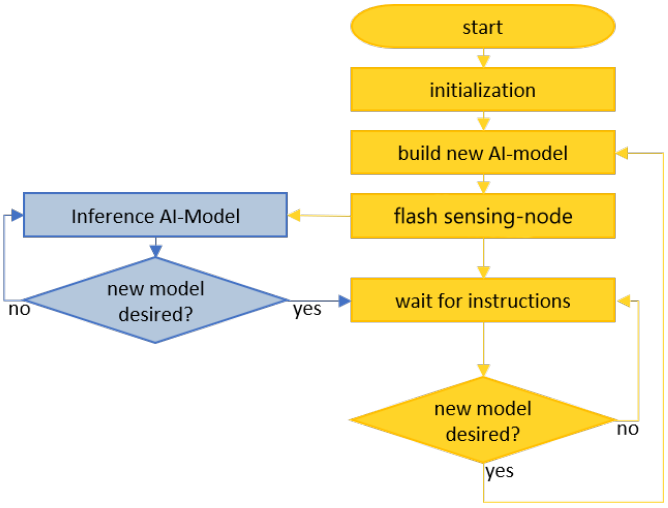

Fig. 2. General program flow chart

AI model to be created, the sensing-node sends the according command to the model-node. If the model-node is still in sleep mode, it must first be booted up. It is then notified that a new model has to be created. First, the model-node loads all newly recorded data from the shared storage medium. It then generates a new model from the existing data. The composition of the data is defined by the user. If the newly created model meets the success rate specified by the user for their specific use-case, it is transferred to the sensing-node.

This overall scenario is then looped. Data is continuously recorded, new models are created, and transferred to the sensing-node until the process is terminated manually or by software.

III. Hardware Selection

This section describes the the main components of the platform. The end-product consists of an ESP-8266 as the sensing-node, the Raspberry Pi4 as the model-node and a microSD-card as shared storing unit.

A. Sensing-Node

The sensing-node is the 32-bit microcontroller ESP8266 from ESPRESSIF [9]. It provides the communication interfaces UART, SPI, I^2C and WiFi. This processor is also supported by many libraries, which simplifies the implementation of various sensors and actuators. It also has a low average power consumption, which is about 70–80mA at 3.35V in standard operation [10]. If no WiFi communication is needed, the node can be operated in the modem-sleep mode. This reduces the power requirement to 15mA. In deep sleep mode, the requirement is only 20µA [10]. The selection of the sensing-node is also based on past research projects such as Cosy [11], in which the microcontroller has already been in use in various demonstrator implementations with low-cost ML models.

B. Model-Node

The model-node is used to calculate the AI models, initiate, and monitor the projects. The key factors in the selection of this node are the properties of the computing unit, RAM, energy requirements, programmability, operability, affordability, interfaces and the manufacturer's support.

The computing power should have a sufficient ratio between the computing time and the energy demand. The RAM requirement depends on several factors. It is influenced by the used operating system, programs and algorithms. The RAM should be as flexible as possible so that it can be adapted to the project depending on the budget and requirements. To determine the minimum RAM requirements, some typical TinyML algorithms were analyzed. The following algorithms were measured using the Scikit-Learn framework: K-Means, Linear Regression, Multilayer Perceptron and Support Vector Classification. The measurement methodology was based on [12]. The measurement scenario was based on a synthetic dataset containing 500.000 data points with 9 features. Each Algorithm was executed 10 times to get better accuracy for the runtime. The used measurement scripts can be accessed in Gitlab[1].

Table I shows the RAM requirements and execution time of common TinyML algorithms. The Algorithms were each run 10 times and the results averaged. The results show an estimate of the required RAM cost for hardware selection purposes. The requirement is below 2GB for all measurements. By assuming that the RAM requirement is almost hardware-independent, the minimum RAM requirement of the system results from the requirement of the operating system with graphical user interface (GUI), the project software and other necessary tools such as IDEs. All necessary tools amount to a RAM requirement of at least 2GB.

When considering the energy requirements of the system, the runtime of the algorithms is decisive. The faster the algorithms are completed, the earlier the node can be switched off, saving energy. Therefore, more emphasis should first be placed on the ratio of runtime to general energy consumption of the computing system.

The operating system with GUI should provide intuitive usability. Further software-dependent requirements are the possibility to use the Arduino IDE, the graphical development tool Node-RED [13] and a flexible choice of programming language to provide good programmability. For programming the sensing-node, the model-node must provide a UART interface as well as the interface for accessing the storage medium. An SD card is suitable as a storage medium, because it can be easily put into operation with the ESP8266 and the Raspberry Pi, flexibly adapted to the storage requirements of the application, and the card can be replaced independently by a layperson, e.g., in case of a malfunction. To use an SD card, the model-node must have the supported interfaces: SPI or SDIO. The manufacturer's support refers to the continuous

[1]All measurement scripts are available at https://gitlab.rlp.net/vski-engineering-group/paper/development-of-an-autonomous-edge-ai-board-prototype-for-local-automated-training-and-inference-of-tinyml-algorithms/codeanddata.git

TABLE I
RAM REQUIREMENTS OF COMMON TINYML ALGORITHMS

Algorithm	Avg. complete RAM [GB]	Avg. scenario duration [s]	Avg. scenario power [W]	Avg. scenario energy [J]
KNN	1.567	20.802	29.783	619.546
Linear Regression	1.585	2.17	48.919	106.154
Multilayer Perceptron	1.546	125.51	29.518	3704.8
Random Forest	1.767	140.4	29.5	4141.8

TABLE II
RASPBERRY PI 4 SCENARIO MEASUREMENTS

Algorithm	Avg. scenario RAM [GB]	Avg. duration [s]	Avg. power [W]	Avg. energy [J]
Baseline	619	600	3.036	n.a.
KNN	0.703	63.96	4.234	270.807
Linear Regression	0.724	21.984	6.299	138.477
Multilayer Perceptron	0.704	312.276	4.389	1370.58
Random Forest	0.8	661.708	4.271	2826.155

technical support in English and a long-term availability of the hardware.

Several candidates were first identified for the model-node and tested for the requirements mentioned. These includes the NanoPi Neo4, the Pine64 SOQuartz, the NVIDIA Jetson Nano, the Raspberry Pi Compute Module 4 and the HiKey970. Due to the requirements, we decided on the Raspberry Pi Compute Module 4. Besides the computing power and user-friendliness, it offers a very good manufacturer support with a large community. The variable selection of the RAM allows the board to be adapted to the project in terms of performance and finances. Furthermore, the production is guaranteed until 2028, which offers a fundamental planning guarantee. For better connectivity, we decided to use the regular Raspberry Pi 4 instead of the Compute Module 4. This has no difference in the hardware specifications and offers the advantage, that the developed board, in form of a HAT can be easily connected to existing systems via the 40-pin header [14].

The manufacturer allows the customer to choose the RAM size between 1, 2, 4 and 8 GB based on their own project requirements. Various Linux distributions can be used as the operating system on this board. This fulfills the required specifications in terms of programmability and usability. For professional use, an operating system without GUI is recommended, since this contributes to further energy reduction [15]. However, for development and for beginners, the use of an OS with GUI is recommended. For both variants, the Raspberry Pi offers a compatible operating system. It also provides the required interfaces UART, SPI and SDIO.

The board manufacturer has English-speaking support, as well as a large community with well-maintained forums. The manufacturer guarantees the availability of the board until at least January 2026.

In order to estimate the power requirements of the model-node, the baseline and maximum requirements of the system were measured. The baseline is the average resource requirements of the system in an 600s time interval.

Table II presents the results of the measurements previously performed on the computer for the Raspberry Pi 4. In addition to the values considered before, the average power required and the average energy consumption were determined. These show that the algorithms do not exceed a power level of 7 Watts. Despite the increased real runtime, which is on average about 5 times the duration of the centralized system based on the tested algorithms, there is an increase in energy efficiency due to the lower basic consumption of the systems. Considering an algorithm that would take 1s on the centralized system with a power usage of 50W. This would result in an energy consumption of 50 Joule. In direct comparison, the AI on Device system with a usage of 6W and a runtime of on average 5s results in an energy consumption of 30 Joule. While the runtime is significantly higher, the overall consumption in this scenario is lowered to about 60% of the original usage. In most scenarios, this should justify the increased processing time. In the measurements the only Algorithm that performed worse on the Pi is the Linear Regression, which has a slight increase in energy consumption due to the long processing time on the Pi system.

Based on the measurements, the requirements, as well as the preliminary work from [15], in which the energy and resource consumption of an AI on Device node was compared to a centralized system, as well as its resource usage optimized along the life-cycle, we decided to use the above-mentioned hardware configuration.

C. Shared storage unit

An SD card is used as the shared storage unit. It stores the project configuration, measurements and model data. When choosing the SD card, attention must be paid to selecting the minimum required storage capacity, since the power requirement depends on the storage volume of the card [16], [17]. Using the SD card on both nodes at the same time will cause a dysfunction. The memory is NAND-based and when used simultaneously, the signals from the nodes interfere with each other. This results in a race condition. Therefore, a solution for mutually exclusive access is necessary, where only one node can access the memory unit at a time. This enforces a

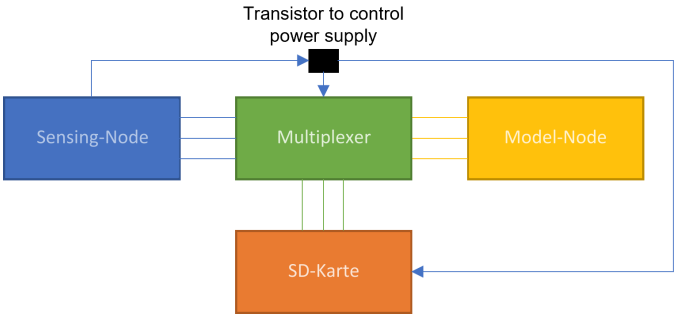

Fig. 3. Multiplexer based SD-card connection

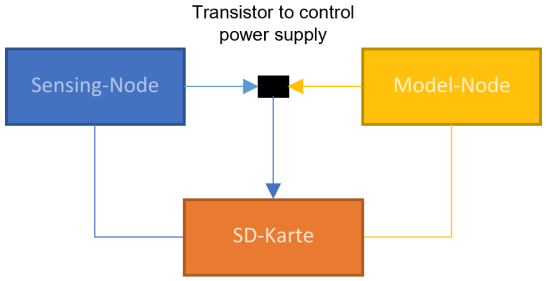

Fig. 4. Pin based SD-card connection

controlled program flow and further protects against unwanted accesses in critical time periods. This access control provides hardware protection against deadlocks and data manipulation. Since simultaneous access to an SD card is not possible, access was implemented on a token basis. Two approaches were considered.

Figure 3 shows an approach that implements a hardware-based switching procedure of the SD-card connection with the help of a multiplexer. This offers the advantage that the pins of the two nodes accessing the memory are always galvanically isolated from each other. This separation means that mutual manipulation of the nodes during communication with the memory unit is not possible. It is important to note, that during switching, there may be no active connection between the memory unit and one of the nodes.

The other method Figure 4 does not use a multiplexer. Here, the configuration of the GPIO pins of the nodes is changed in a dependent sequence so that only one node has access to the memory unit and no signal manipulation occurs. To manage this, the GPIO pins of the idle system are set to input. The power to the SD card is then cut off and resupplied by the operating component. Through this method the active Node can smoothly take control over the storage unit

Due to the additional financial and energy costs involved in operating a multiplexer, the focus was on the second approach. Furthermore, the elimination of additional hardware components precludes a potential source of error.

IV. Implementation

In this section, the implementation of the concept is described in detail. First, the hardware structure is presented and explained. Afterwards, the created software, as well as the overall flow of the system, is described.

A. Hardware

The main components are the model-node (Raspberry Pi 4) and the sensing-node (ESP8266) as well as the SD card adapter. The model-node is connected to the adapter via SDIO and the sensing-node with SPI. A 5V power supply is used as power source. When calculating the expected current demand, it is assumed that the main components are active at the same time. The maximum current is the sum of the maximum partial currents. The Raspberry Pi expects a current of 3A at 5V [14], the ESP8266 needs 70-80 mA at 3.35V as mentioned before. The 3.3V operating voltage of the SD card and the ESP8266 is stepped down by an LM2596S voltage converter [18].

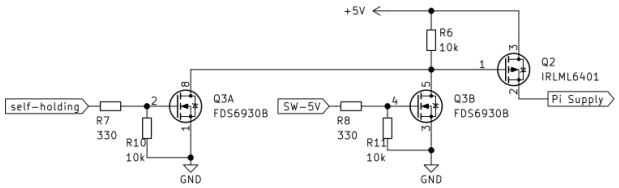

Fig. 5. Pi Power supply control

The power supplies of the SD card adapter and the model-node are switchable via transistors as shown in Figures 4 and 5. They are only activated when needed. The SD card adapter can be controlled by both nodes. The model-node is switched on and off by the sensing-node. However, it has a self-holding function. This is necessary when transferring the new model to the sensing-node. During this process, the GPIO configurations of the sensing-node change. Also the pin to control the power supply of the model-node changes its state during this process. This would turn off the power supply to the model-node and the flash process would be interrupted.

The power supply of the model-node and the SD card adapter can be switched via software and jumpers. The sensing-node can be reset by the model-node. In case of a complete shutdown of the node, the jumper has to be removed, which is plugged in during regular operation. Thus, during troubleshooting, each component can be switched on and off manually. This is necessary to prevent mutual influence on the other subsystems in case of an error. The model-node can also be switched on via the jumper if the user wants to interact with the system, e.g., if they want to influence the active program flow, re-initialize, or monitor the current state.

Furthermore, the requirements of the sensing-node wiring (ESP8266) must be considered. To transfer a new program to the node, the node must be set to flash mode. This mode is set when the GPIO pins 0 and 15 are at GND potential during a reset of the node. This wiring can be done automatically by the model-node. Afterwards, the model-node can transfer the new model via UART interface to the sensing-node. The node has to be restarted without pulling the mentioned GPIO pins to GND potential. By connecting the sensing-node to the

other hardware components, the GPIO pin 0 and the ADC pin as well as the I^2C pins can be used to connect sensors and actuators. The I^2C bus allows to use GPIO pin extension. The schematic and the parts list can be viewed in the Git repository. [2].

B. Software

As operating system, either Raspberry OS or Raspberry OS Lite is used on the model-node. To establish the connection between the model-node and the SD card, the operating system's device-tree must be changed by an overlay. The boot flow of the Raspberry Pi differs at this point from the regular flow of other embedded systems. It starts from the GPU core, which activates a boot loader via the ROM memory. This loads a second boot loader, which is located on the system memory of the Raspberry Pi. The device-tree is stored in this, describing the hardware configurations of the system. Overlays can be used to overwrite the system configuration, so that the functionalities can be adapted to the user's demands. The used overlay allows the usage of the SDIO interface to establish the connection to the SD card. In addition, the parallel use of the SDIO and WiFi interfaces is integrated, as the model-node could also require network access in the future.

As mentioned in section III-C, simultaneous access to the SD card by the nodes must be avoided. The mutual locking is token-based. The token allocation is performed by the sensing-node, which controls the overall process as master. If there is no active connection to the SD card, it is de-energized to save energy. At this time, all accessing GPIO pins of both nodes are switched to inputs. If a node requests access, the token is allocated and the accessing node turns on the supply. Similarly, the GPIO pins of the accessing node are switched as outputs and communication between the node and SD card can be executed. After the communication is finished, the node logs off and the GPIO pins are configured as inputs again. To completely log off the SD card, it has to be reset. This is done by switching off the power supply automatically after the access is finished. The token is then returned to the master.

For compiling and flashing the sensing-node programs, the CLI version of the Arduino IDE is used.

For controlling the system, an application including a GUI for the model-node was implemented. As an initial step in the project initialization, a project name, the storage path for the project, the initial training data, the sensing-node program and the program for creating the AI model must be specified. Then, a project folder is created, and all initial configuration data is stored as a copy there. The user interface is shown in Figure 6 and Figure 7.

After the initialization of the user, the TinyML model is created, using the initial data. This model must be provided in a suitable programming language that is compatible with the sensing-node. We use the Arduino IDE, so the models must be in C or C++. It is possible to use suitable frameworks for this platform, or to convert models from other languages. In the paper [19], TinyML frameworks were summarized. Available C and C++ libraries are presented, but they are not usable with the ESP8266. However, the Python frameworks MicroMLGen [20] and emlearn [21] are compatible. These allow to convert several Scikit-Learn models into C++. This model has to be integrated into the program routine of the sensing-node.

For the sensing-node, a library was created to simplify the program generation of new programs and use-cases. Functions of the library consist of, e.g.the communication between the nodes, the storage of the sensor data, and the access management of the memory unit.

Communication between the sensing-node and the model-node happens via the serial UART interface. This is used to manage the general program flow and to exchange information about the current process. The communication is bidirectional, so that parallel communication is possible.

The recorded measurement data can be written directly to the SD card or first collected and transferred as a block. For this purpose, a heap buffer is created that only writes the measurement data to the SD card when it is full or the program flow requires it, such as the creation of a new model or by a user-defined event. This reduces the data volume of the connection management, since the communication channel

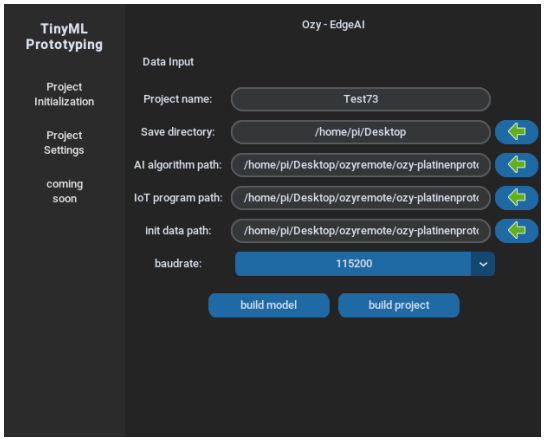

Fig. 6. AI on Device GUI - Data

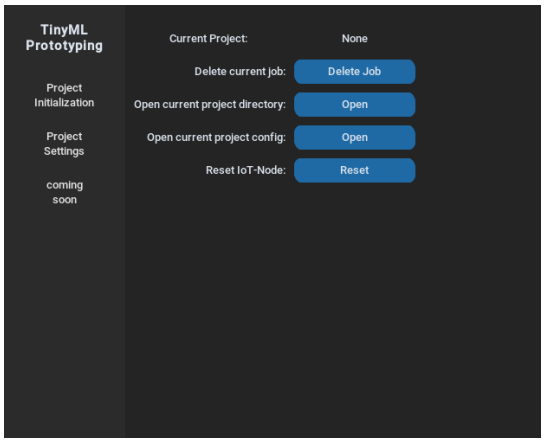

Fig. 7. AI on Device GUI - Settings

[2]All measurement scripts are available at https://gitlab.rlp.net/vski-engineering-group/paper/development-of-an-autonomous-edge-ai-board-prototype-for-local-automated-training-and-inference-of-tinyml-algorithms/codeanddata.git

must be established less frequently and the duration of the active SD card is reduced.

C. Process flow

Figure 8 summarizes the finalized flow of the system application. First, a new project with the associated data must be created on the model-node. This includes the initial data, a Python application that contains the machine learning approach, and an Arduino file that controls the sensor communication and the execution of the models. Furthermore, parameters such as the name of the project, minimum success rate and conditions for the calculation of new models are defined in this step. In the next step, a basic model is defined from the input data and transferred to the sensing-node. From this point forward, the model-node is in a communication monitoring state in which it simply waits for new commands to be received from the sensing-node. Now the sensing-node process begins. It performs its tasks in a continuously running loop. First, it is in idle mode, in which new commands can be accepted. It then switches to measurement mode, in which it records data from the sensors. This data is used in evaluation mode, in which it is processed by the current machine learning model. The data is then stored on the storage unit, where it can be viewed at a later time or used by the model-node to re-train new models. At the end of each loop, the system checks whether the conditions for calculating a new model are met. As already mentioned, these were defined by the user in the initialization phase. If the conditions are not met, the old model continues to be used and the loop starts again. However, if a new model is required, a new model sequence begins. A command is sent to the model-node via the communication interface. This copies the buffered data of the storage unit from the previous model runs and uses it to re-train a new model. If this new model meets the requirements specified by the user, it is transferred to the sensing-node again, and the cycle begins anew.

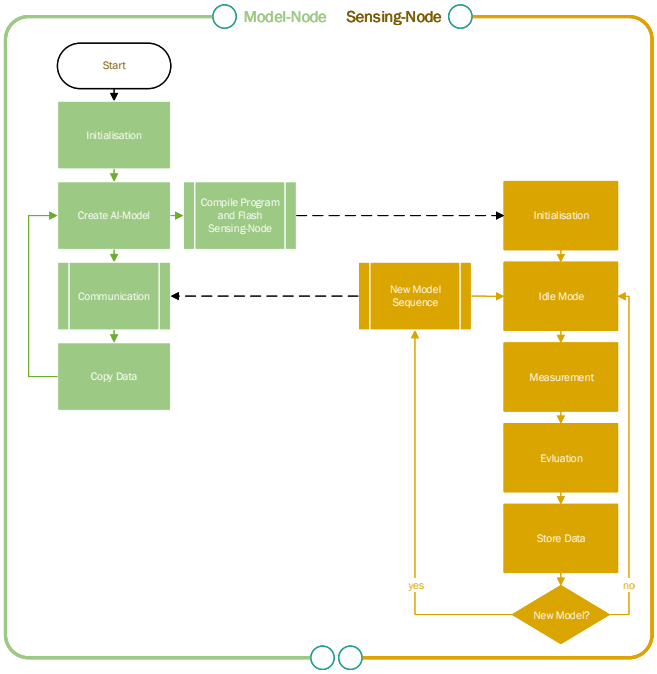

Fig. 8. Flow Chart EdgeAI

V. Example/Application

A. Use Case Setup

As an implementation example, an electronic nose based on gas sensors was implemented using the AI on Device board. This application is a gas value-based liquid substance detection, which is based on the Bosch BME680 environmental sensor. For classification of the different substances, the VOC (volatile organic compounds) values of the substances are recorded in the temperature range from 200°C to 400°C. They are classified using a Random Forest Classifier and assigned to a previously trained label. The sensing-node was extended for the use case with a button, a BME680 environmental sensor for data recording and an LCD display for presenting the measurement results.

B. Use Case Execution

First, a new project must be created using the GUI-application. A project name must be specified, as well as the program of the sensing-node, the classification algorithm, and the initial VOC data including the labels. After providing the data, an EdgeAI-project with a standardized folder structure is created. It contains a setup folder with the initial algorithms, a data folder, and a model folder to store future model iterations. The first AI model is computed, and the resulting program is transferred to the sensing-node. This is done by generating the corresponding C-code of the AI-model in form of a header file that is attached to the Arduino program file. The model-node then goes into idle state and waits for further instructions from the sensing-node. If the model-node is no longer required, it shuts down. This process is only model-node sided, so no communication is necessary apart from flashing the program to the sensing-node.

The installed button has two different functions, depending on the duration of activation. When the button is pressed shortly, the existing model is used to identify the presence of the liquid substance. After inferring the classification algorithm that was delivered by the model-node the result is shown on the LCD display. If the button is pressed for a long time, a new substance is added to the classification.

To do this, the environmental sensor records the VOC values of the substance. The number of measurement iterations was determined at the project's initialization by the user. The more data is recorded, the better the model can be trained. However, more time and energy is needed to take the measurements and to calculate the model. The measurement data is then transferred to the SD card. Afterwards, the sensing-node boots the model-node and sends a request for a new AI model.

The model-node starts the routine and copies the previously recorded data from the storage unit to the project folder. The new AI model is then created. The data used for the creation must first be prepared. This means that it must be available in the required quality and formatting. Quality and format are

predefined by the user. The data is then reduced in order to minimize the calculation time. To do this, a feature selection algorithm is used, so that sufficient features are still available for classification. The new AI model is then calculated and checked to ensure that it meets the quality requirements. To do this, the r-squared score of the newly created model is considered and checked to see whether the quality specified by the user is achieved. If the AI model does not achieve the required accuracy, the model is stored in the project data but not transferred to the sensing-node. The new data recorded for this model is also not used to create new models. However, if the new AI model meets the criteria, the model is transferred to the sensing-node via the flash process. The loop is then completed.

C. Results

The developed concept offers an innovative, energy-efficient solution to an emerging problem of the declining expansion of rural areas in light of the increased emergence of process-supporting AI solutions. The presented combination of hardware components allows the execution of complex machine learning algorithms with extreme performance efficiency. In addition, the completely internal memory management eliminates the need for infrastructural network expansion, which benefits the implementation of AI applications with high data requirements in rural regions. Furthermore, existing fields of application can be converted to an EdgeAI model thanks to the good adaptability of the hardware solution. This also leads to an increase in performance efficiency in better developed areas and additionally offers interfaces for the application of federated learning processes, which further increases the effectiveness of the models to be calculated by a significant proportion through the inter-connectivity of individual distributed EdgeAI nodes.

New concepts in the areas of hardware and software concerning remote AI computation were presented, aiming to provide a solution to the problems described. A use case exemplifies the realization of the concept and demonstrates the implementation. In addition, extensive energy consumption studies were undertaken to identify suitable hardware components that meet our performance requirements. The results of the research will lead to a new hardware solution that supports the rural sector in its efforts to expand artificial intelligence and make existing systems more energy-efficient and environmentally conscious.

D. Discussion and Outlook

Our research has led us to the conclusion that there are a large number of advantages of using AI on Device over conventional implementations of machine learning applications. However, there are some aspects that need to be considered separately in this use case. External error detection proves to be complex, due to the automatic implementation method, which is not linked to external networking. However, a certain degree of networking is still required when monitoring the collected data and for external error detection. Furthermore, there is a disadvantage in terms of cost when calculating federated models, as each node must have a CPU as a computing unit.

This is contrasted by the advantages of a far more independent and autonomously functioning system, which enables the use of independent, local hardware. The use, evaluation and versioning of different models can also be carried out locally via the integrated memory of the overall system. The large number of models created by distributing multiple nodes also opens up the possibility of implementing federated learning approaches. In these approaches, a global master model is created through the inflow and merging of many individual locally calculated sub-models. In the planned implementation, the required sub-models are calculated via the distributed, individual model-nodes. In addition to the energy savings already mentioned by using an AI on Device system compared to a regular system, this offers a further decisive advantage and additional energy optimization capabilities.

The Open Neural Network Exchange (ONNX) can be used to manage the generated models. This framework offers the possibility of managing and importing machine learning models independently of languages and ML frameworks, which enables the sensing-node to be used independently of the source system.

The switch from ESP8266 to the ESP32 also represents a further project approach. This switch offers the possibility of process parallelization thanks to its 2-core system. This is to be used for simultaneous status monitoring and communication with the model-node, as well as parallel execution of the existing model. An implementation in Node-Red and Ardublock is also planned for the further course of the project. This is intended particularly for teaching and for the creation of low-level applications by inexperienced users. It will also incorporate ready-made, frequently used machine learning algorithms. A terminal-based version, for more experienced users, is also planned, which will further increase energy efficiency. This will be achieved by using a low-energy operating system without a graphical user interface.

ACKNOWLEDGEMENTS

This work was partly funded by the German Federal Ministry of Food and Agriculture (BMEL) project ”KI-Pilot” under Grant 2820KI001, the Carl Zeiss Foundation project ”KI-GenF“ (www.umwelt-campus.de/en/acknowledgements-resource-and-energy-efficiency-of-aiot), the German Federal Ministry for the Environment, Nature Conservation, Nuclear Safety, and Consumer Protection (BMUV) project ”KIRA” under Grant 67KI32013B, and the German Federal Ministry for Economic Affairs and Climate Action (BMWK) project ”EASY” under Grant 01MD22002D. We gratefully acknowledge Swen Haab for providing necessary data resources to accomplish this research work. The text was linguistically improved by using ChatGPT4.

REFERENCES

[1] P. P. Ray, “A review on tinyml: State-of-the-art and prospects.” ScienceDirect, 2022, pp. 1595–1623.

[2] R. Singh and S. S. Gill, “Edge ai: A survey,” *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 71–92, 2023.

[3] M. Chen, D. Gündüz, K. Huang, W. Saad, M. Bennis, A. V. Feljan, and H. V. Poor, "Distributed learning in wireless networks: Recent progress and future challenges," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 12, pp. 3579–3605, 2021.

[4] R. Immonen and T. Hämäläinen, "Tiny machine learning for resource-constrained microcontrollers," *Journal of Sensors*, vol. 2022, pp. 1–11, 11 2022.

[5] T. S. Ajani, A. L. Imoize, and A. A. Atayero, "An overview of machine learning within embedded and mobile devices–optimizations and applications," *Sensors*, vol. 21, no. 13, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/13/4412

[6] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog computing: Principles, architectures, and applications," *CoRR*, vol. abs/1601.02752, 2016. [Online]. Available: http://arxiv.org/abs/1601.02752

[7] S. Baller, A. Jindal, M. Chadha, and M. Gerndt, "Deepedgebench: Benchmarking deep neural networks on edge devices," in *2021 IEEE International Conference on Cloud Engineering (IC2E)*. Los Alamitos, CA, USA: IEEE Computer Society, oct 2021, pp. 20–30. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/IC2E52221.2021.00016

[8] J. Murach, "Entwicklung eines autonomen edgeai-platinen-prototypen für die lokale knotenbasierte erstellung eines ki-modells," Trier, April 2022.

[9] Espressif Systems, "Esp8266ex datasheet," https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf, 2019.

[10] J. Mesquita, D. Guimarães, C. Pereira, F. Santos, and L. Almeida, "Assessing the esp8266 wifi module for the internet of things," in *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2018, pp. 784–791.

[11] "Cosy," https://cosy.umwelt-campus.de/, accessed: 2023-07-19.

[12] A. Guldner, R. Bender, C. Calero, G. S. Fernando, M. Funke, J. Gröger, L. M. Hilty, J. Hörnschemeyer, G.-D. Hoffmann, D. Junger, T. Kennes, S. Kreten, P. Lago, F. Mai, I. Malavolta, J. Murach, K. Obergöker, B. Schmidt, A. Tarara, J. P. De Veaugh-Geiss, S. Weber, M. Westing, V. Wohlgemuth, and S. Naumann, "Development and evaluation of a reference measurement model for assessing the resource and energy efficiency of software products and components—green software measurement model (gsmm)," *Future Generation Computer Systems*, vol. 155, pp. 402–418, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X24000384

[13] Nodered, "Nodered - low-code programming for event-driven applications," https://nodered.org, accessed:.

[14] Raspberry Pi (Trading) Ltd., "Raspberry pi 4 model b datasheet," https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf, 2023.

[15] A. Guldner and J. Murach, "Measuring and assessing the resource and energy efficiency of artificial intelligence of things devices and algorithms," in *Advances and New Trends in Environmental Informatics*, V. Wohlgemuth, S. Naumann, G. Behrens, H.-K. Arndt, and M. Höb, Eds. Cham: Springer International Publishing, 11 2022, pp. 185–199.

[16] SMART Embedded Products, "Smart modular microsd memory cards," http://www.amtron.com/micro_sd_card/SMART%20Modular/smart_microsd_slc_datasheet.pdf, 2018.

[17] Kingston Technology, "microsdhc memory card flash storage media," https://www.kingston.com/datasheets/SDCIT-specsheet-8gb-32gb_us.pdf.

[18] Texas Instruments, "Raspberry pi 4 model b datasheet," https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf, 2023.

[19] P. Ray, "A review on tinyml: State-of-the-art and prospects," *Journal of King Saud University - Computer and Information Sciences*, vol. 34, 11 2021.

[20] eloquentarduino, "Micromlgen." [Online]. Available: https://github.com/eloquentarduino/micromlgen

[21] emlearn, "emlearn." [Online]. Available: https://github.com/emlearn/emlearn