Making Machine Learning More Energy Efficient by Bringing It Closer to the Sensor

Marius Brehler, Lucas Camphausen, Benjamin Heidebroek, Dennis Krön, Henri Gründer, and Simon Camphausen, *Fraunhofer IML, 44227, Dortmund, Germany*

Processing data close to the sensor on a low-cost, low-power embedded device has the potential to unlock new areas for machine learning (ML). Whether it is possible to deploy such ML applications or not depends on the energy efficiency of the solution. One way to realize lower energy consumption is to bring the application as close as possible to the sensor. We demonstrate the concept of transforming an ML application running near the sensor into a hybrid near-sensor in-sensor application. This approach aims to reduce overall energy consumption and we showcase it using a motion classification example, which can be considered a simpler subproblem of activity recognition. The reduction of energy consumption is achieved by combining a convolutional neural network with a decision tree. Both applications are compared in terms of accuracy and energy consumption, illustrating the benefits of the hybrid approach.

Tiny machine learning (*TinyML*) refers to the deployment of machine learning (ML) on low-cost, low-power embedded devices. Processing the data close to the sensor has the potential to unlock new areas for ML. This can also improve data privacy as the raw data do not need to be processed by other external devices. If the raw data do not have to be sent to other devices for further processing, the necessary communication decreases. This can therefore also reduce energy consumption, which is a key factor for many applications.

According to García-Martín et al.,[1] ML researchers have mainly been focused on producing highly accurate models without considering energy consumption as an important factor, whereas computer architecture researchers have been investigating energy consumption for decades. However, low energy consumption is of special importance for TinyML and always-on applications. Low energy consumption is necessary for several reasons. Depending on the area of application, a huge number of embedded solutions will only be deployed if the maintenance effort can be kept to a minimum. If neither a continuous power supply nor energy harvesting is possible due to environmental conditions, the devices must be powered by batteries. As a consequence, energy consumption determines whether a TinyML application can be deployed or not, as changing the battery at short intervals is not an option. If these requirements can be fulfilled, such devices can, for example, be used in logistics to track and analyze transport routes, with the goal of optimizing them and thus reducing CO_2 emissions. However, such devices can only have a positive environmental impact if they are used as long as possible to further keep electronic waste to a minimum.[2] Low energy consumption contributes to this.

One approach to reduce energy consumption is near-sensor and in-sensor computing, where tasks such as data processing are performed near or even within the sensor.[3] In a near-sensor computing architecture, a processing unit or accelerator resides close to the sensors to execute specific computational tasks. Although a huge amount of energy in off-chip processing is consumed by the dynamic random-access memory, near-sensor processing drastically reduces the amount of used energy by using only the on-chip static random-access memory. In in-sensor computing architecture, the sensor itself can process sensory information, which tends to offer even more potential to reduce energy consumption.

In this article, we demonstrate how an example ML application can be transformed from a near-sensor

Digital Object Identifier 10.1109/MM.2023.3316348
Date of publication 19 September 2023; date of current version 2 November 2023.

into a hybrid near-sensor in-sensor ML application to reduce overall energy consumption. Instead of using only a near-sensor convolutional neural network (CNN), the CNN is combined with a decision tree (DT) that is executed in the sensor. The concept is illustrated based on the application of motion classification. After introducing the motion classification example and the corresponding CNN, the hybrid approach is described. Subsequently, the accuracy and energy consumption of the different approaches are compared with each other. The primary objective is to highlight the key factors influencing energy consumption and to assess the impact of the approach.

MOTION CLASSIFICATION

Motion classification can be considered as a simpler subproblem of activity recognition. The area of application for motion detection is highly versatile. It ranges from medical applications, where it can be used to monitor medical conditions, to logistical applications. The main goal is to analyze motion data to map them to predefined categories. Motion classification normally involves several steps typical to pattern recognition tasks. These include the feature extraction, feature selection, and classification of the data based on the extracted features. The exemplary CNN used in this article is able to classify eight different motions, as illustrated in Figure 1.

If no significant motion is detected, this is assigned to the class *nothing* (class C_0). The *left-right* motion (class C_1) takes place along the *y*-axis, the *push-pull* motion (class C_2) along the *x*-axis, and the *up-down* motion (class C_3) along the *z*-axis. Furthermore, the CNN can classify four types of circles. The *clockwise circle* (class $C_{4,0}$) and *counter-clockwise circle* (class $C_{4,1}$), visualized in Figure 1(e), take place in the *yz*-plane. In contrast to these, the *forward circle* (class $C_{4,2}$) and *backward circle* (class $C_{4,3}$) take place in the *xz*-plane, as illustrated in Figure 1(f).

The detection and classification is based on the data of an accelerometer. Here, the accelerometer is configured to capture samples with a sample rate of 52 Hz. The captured samples are processed by a low-pass digital filter before being passed to the CNN. A sequence x, passed as input to the CNN, consists of 104 samples for each of the three axes of motion, which corresponds to a sequence of 2 s. With this, the CNN can then be written as $g_{\text{CNN}}(x) : \mathbb{R}^{3\times104} \rightarrow \{C_0, C_1, C_2, C_3, C_{4,0}, C_{4,1}, C_{4,2}, C_{4,3}\}$.

The nonpublic dataset for the motion classification example comprises a total of 28,665 samples, which are approximately equally distributed. The training set utilizes 60% of the samples, while 30% are used for validation. The remaining 10% form the test dataset.

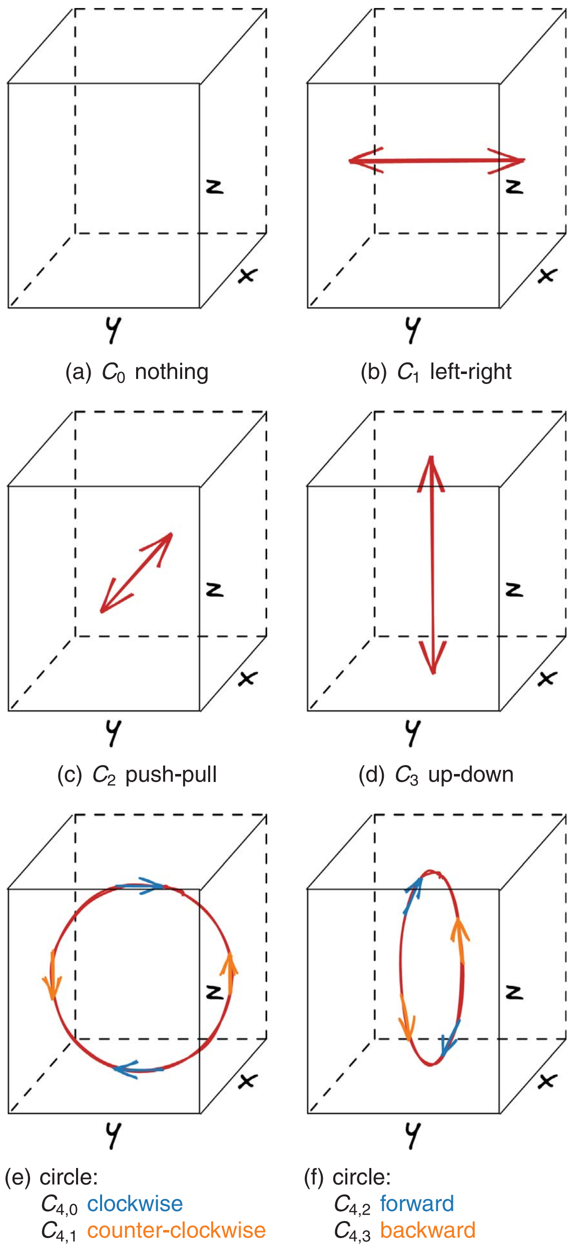

FIGURE 1. Different motion classes. (a) C_0 nothing. (b) C_1 left-right. (c) C_2 push-pull. (d) C_3 up-down. (e) $C_{4,0}$ clockwise circle and $C_{4,1}$ counter-clockwise circle. (f) $C_{4,2}$ forward circle and $C_{4,3}$ backward circle.

The architecture of the CNN is visualized in Figure 2. The CNN contains four temporal convolution layers with a rectified linear unit (ReLU) activation function, which extract features along the time-axis for each channel separately. For each layer, the kernel sizes

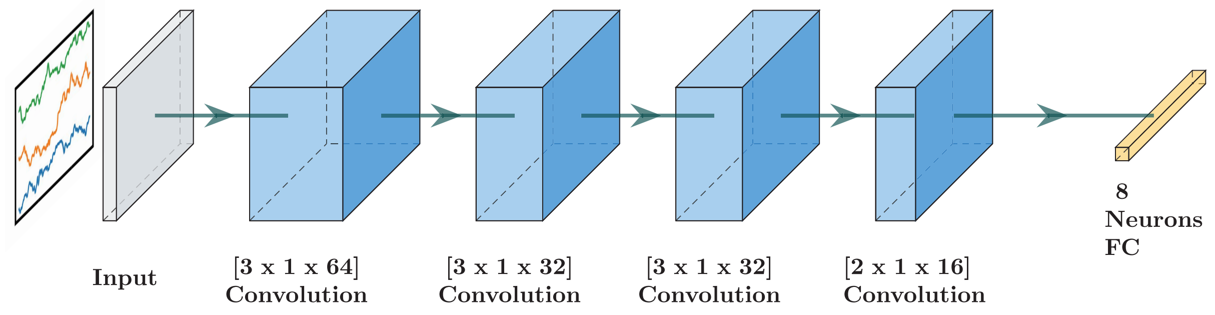

FIGURE 2. Architecture of the CNN. The input (grey) consists of 104 samples for each of the three accelerometer axes. This is passed to temporal convolutional layers (blue), each with ReLU activation function. For each convolutional layer, the square brackets indicate the kernel sizes and the number of computed feature maps. These are followed by an FC layer with eight neurons and softmax function.

were determined experimentally. One fully connected (FC) layer with a softmax function is used to classify the extracted temporal features. The use of temporal convolutions is inspired by the work of Yang et al.,[4] in which a temporal CNN is used for human activity recognition.

The CNN was trained over 60 epochs with an initial learning rate of 0.01 and a time-based learning rate decay. Additionally, a dropout layer with a dropout rate of 0.5 was added before the FC layer during training to avoid overfitting. For the exemplary use case, the CNN achieved an accuracy of 88%.

NEAR-SENSOR EXECUTION

The CNN is compiled and executed with the end-to-end ML compiler and runtime Intermediate Representation Execution Environment (IREE). More specifically, we make use of TinyIREE,[5] a subset of IREE options. TinyIREE is used to generate a compact workload and comes with a runtime library optimized for embedded systems without an operating system: so-called bare-metal systems. The used sensor is an ISM330DHCX microelectromechanical systems (MEMS) 3-D accelerometer on an X-NUCLEO-IKS02A1 industrial-motion MEMS sensor expansion board. The sensor expansion board is utilized in conjunction with an nRF52840 DK single-board development kit, which is equipped with an nRF52840 system on chip (SoC). The nRF52840 SoC features an Arm Cortex-M4 CPU with a floating point unit and runs at 64 MHz. It provides 1 MB of flash and 256 kB of random-access memory (RAM). The ISM330DHCX is connected to the nRF52840 DK using $\mathrm{I^2C}$ in standard mode with a bit rate of 100 kbit/s.

FROM NEAR- TO IN-SENSOR MOTION CLASSIFICATION

The used ISM330DHCX MEMS 3-D accelerometer has an embedded Machine Learning Core (MLC), which can run up to eight DTs on the ISM330DHCX.[6] As classification trees can be represented as DTs,[7] at least a portion of the accelerometer data can be directly processed in the sensor. This allows us to move from a near-sensor to a hybrid near-sensor in-sensor ML application. The approach is closely related to the use of wake-up detectors.[8] However, it differs from most of the wake-up detectors in that the data are not solely analyzed to wake up more complex hardware. A part of the evaluation is already conducted in the sensor itself, eliminating the need to involve the CPU.

The inputs for a DT running on the ISM330DHCX are given by statistical measures computed on a fixed number of accelerometer samples. In our example, we use the mean, variance, energy, range, and zero-crossings features for each axis separately. As for the CNN, 104 samples are used. The feature computation is then given by $h(x) : \mathbb{R}^{3\times104} \to \mathbb{R}^{3\times5}$.

In principle, a full in-sensor application can be realized by using a DT, which classifies all the eight motions visualized in Figure 1. However, our experiments have shown that the ability of a DT to distinguish among these eight, partly very similar motions, is limited. An overall accuracy of 73% was obtained for a DT with a depth of seven, in contrast to the 88% achieved by the CNN. This particular DT consists of 109 nodes, including 55 leaves. In our experiments, a maximum accuracy of 78% was obtained with a DT consisting of 1305 nodes. However, the MLC is limited to the execution of a DT with up to 512 nodes, which is

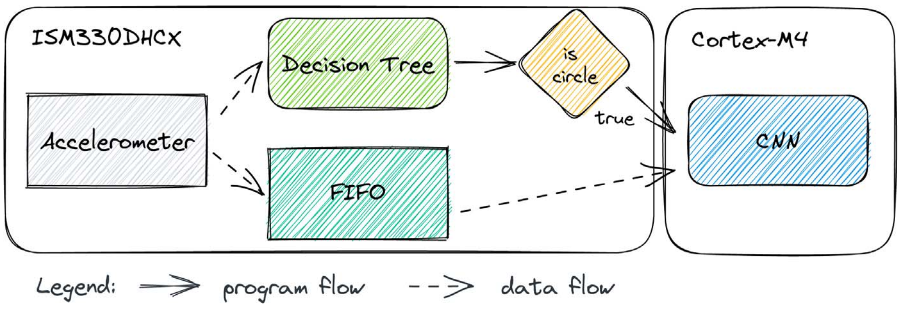

FIGURE 3. Simplified program and data flow of the hybrid near-sensor in-sensor ML application.

why the much smaller DT is being compared. Wrong results especially occurred in the classification of the different circle classes, previously shown in Figure 1(e) and (f). This shows that the ability of the DT to distinguish between complex, and at the same time similar motions, is limited.

THE PRESENTED APPROACH TRIES TO OFFLOAD AS MUCH PROCESSING TO THE SENSOR AS POSSIBLE TO INCREASE THE SLEEP TIME OF THE CPU.

Therefore, a hybrid approach was chosen, in which a DT is combined with a CNN, similar to the work of Daghero et al.[9] This is referred to as *DT+CNN* in the following. For this purpose, a meta-class that covers all four circular motions was created, denoted as C_4. Furthermore, a new DT was trained, using the same input features and the identical input count as before. The DT can be written as $g_{\mathrm{DT}}(h(x)): \mathbb{R}^{3\times104} \to \{C_0, C_1, C_2, C_3, C_4\}$. The final DT consists of 91 nodes, including 46 leaves, and features a depth of seven. By merging the different circular motions into one meta-class, the DT was able to achieve an accuracy of 93% for this modified classification task. Whenever the meta-class C_4 is detected by the DT, the CNN is used to process the same data and to distinguish among the circle classes. To be able to process the data with the CNN that the DT already processed previously, the data are temporarily stored in a first-in, first-out (FIFO) buffer on the ISM330DHCX. A simplified program and data flow of the application is shown in Figure 3. As long as only the DT processes the data, the SoC is in sleep mode. Only if the meta-class C_4 is detected by the DT, the CPU is awakened from sleep to fetch the data sample by sample from the FIFO and execute the CNN.

The combined DT+CNN classifier can be written as an ensemble $f(g_{\mathrm{DT}}, g_{\mathrm{CNN}}, h, x): \mathbb{R}^{3\times104} \to \{C_0, C_1, C_2, C_3, C_{4,0}, C_{4,1}, C_{4,2}, C_{4,3}\}$. Following the notation from Li et al.,[10] the ensemble is given by

$$f(g_{\mathrm{DT}}, g_{\mathrm{CNN}}, h, x) = \begin{cases} g_{\mathrm{DT}}(h(x)), & \text{if} g_{\mathrm{DT}}(h(x)) \\ & \quad \in \{C_0, \ldots, C_3\} \\ g_{\mathrm{CNN}}(x), & \text{if} g_{\mathrm{DT}}(h(x)) = C_4 \end{cases}. \tag{1}$$

This is a very simple approach to form an ensemble based on the two classifiers. There are several more complex options to combine a DT and a CNN, as described in Li et al.[10] The presented approach tries to offload as much processing to the sensor as possible to increase the sleep time of the CPU.

CLASSIFICATION PERFORMANCE

The DT+CNN achieves an accuracy of 87%, close to 88% when fully relying on the CNN, hereinafter referred to as *CNN only*. Accuracy is defined as the ratio between the number of correct predictions and the total number of predictions. Figure 4 shows the

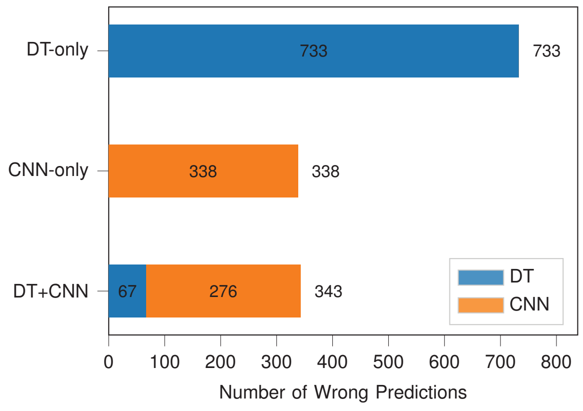

FIGURE 4. Number of wrong predictions by the different ML approaches.

number of wrong predictions for the different approaches. Using the DT+CNN approach, the individual contribution of the DT and the CNN to the number of wrong predictions is as follows. A total of 2730 predictions were performed. In 192 cases, a wrong class was predicted by the DT. Here, 125 of these 192 wrong predictions are false-positive circles, meaning that the motion is classified as a circle even though the motion is of another class. In this case, however, it is possible that the CNN will correct the prediction of the DT as a prediction of class C_4 is always re-evaluated by the CNN. Thus, only 67 wrong predictions remain, which are not processed by the CNN, and the DT only contributes with these to the total number of wrong predictions. In the example, 56 of the 67 wrong predictions are false-negative circles, meaning that the motion is not classified as a circle even though the motion was a circle. The remaining 11 false-negative predictions belong to classes C_1 and C_2 but have been classified as C_0.

The CNN contributes to the total number of wrong predictions, with 276 true-negative and false-positive predictions. Among these are 25 cases in which a circular motion is not predicted as a circle class at all. Furthermore, seven *up-down* and six *left-right* motions were classified as circles. The most challenging task for the CNN is to distinguish among the different circular motions, especially between those that are performed in the same motion plane. This is the case for the DT+CNN as well as for the CNN-only approach.

However, the results show that using a DT for the detection of the circle meta-class C_4 does not significantly decrease the performance. This is different when using a DT to also distinguish among the individual circle classes, as already evident from the accuracy. In the evaluated example, 773 predictions would be incorrect using the DT-only approach. As the DT+CNN approach shows an accuracy that is sufficient for the application under consideration and comparable to the CNN-only approach, its impact on energy consumption is evaluated.

MEASUREMENT SETUP

The setup for energy consumption measurements is described in the following.

In our sample applications, the so-called *System ON* mode is used to send the CPU to sleep to save power after finishing a workload. In this mode, the CPU continues at the halt point after a wake-up. Even higher energy savings can be achieved by using the *System OFF* mode, in which more power is saved by retaining much less information while the system is in sleep. If the *System OFF* mode is used, the system is reset when it wakes up. To avoid a reconfiguration of the sensor and to keep previous prediction results, one can define the RAM sections to be retained when the system is in *System OFF* mode. Even in the *System ON* mode, it is possible to save further power by not retaining the entire RAM. However, this was not used here. The general-purpose input–output (GPIO) peripherals stay configured in both modes.

First, the total energy consumption is measured. All the measurements are performed with an nRF Power Profiler Kit II (PPK2) using the Power Profiler app of the nRF Connect for Desktop framework. The PPK2 is used in source meter mode to measure the energy consumption of the entire system, including peripherals such as the accelerometer. Here, the PPK2 delivers the power to the nRF52840 DK by using it as an external power source. During all the measurements, the power is supplied at 3 V.

To examine the impact of our hybrid approach in more detail, the subsystems, namely, the nRF52840 SoC and ISM330DHCX, were analyzed individually. To analyze the energy consumption of a specific subsystem, the PPK2 is used in ampere meter mode. In this case the power must be supplied externally, e.g., via USB, a coin cell, or by an external power supply. Here, a second PPK2 is used in source mode as the latter. The nRF52840 DK allows us to measure energy consumption of the SoC in isolation of the other components. The board was prepared accordingly. To measure energy consumption of the ISM330DHCX, an ammeter can be connected to the appropriate jumper on the sensor expansion board.

ENERGY CONSUMPTION

Finally, the energy consumption is evaluated, and the impact of bringing the ML application closer to the sensor is discussed. In this context, both the total energy consumption of the embedded device and the energy consumption of the subsystems are analyzed.

Characteristic current profiles for the entire system are visualized in Figure 5. Here, the CNN-only and DT+CNN approaches are compared with each other. The current profiles were measured in two separate experiments using the same sequence of motions. After a period without motion, circular motions were performed for roughly 4 s, followed by another period without motion. The current consumption for an excerpt of this sequence of motions, corresponding to the classes C_0, C_4, C_4, and C_0, is visualized in Figure 5. For the purpose of visualization, the time stamps were aligned to each other. Based on the current profiles, distinct

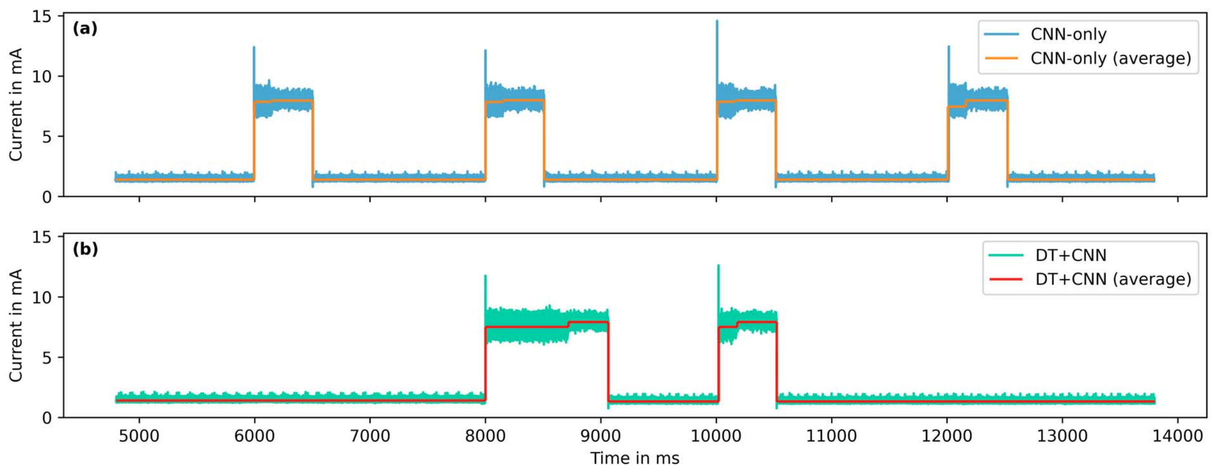

FIGURE 5. Current over time of the entire system for (a) the CNN-only application and (b) DT+CNN application.

phases of energy consumption can be identified. For each of the phases, the average current is calculated and plotted accordingly. Basically, one can distinguish between a phase with low average energy consumption and a phase with high average energy consumption. The phase with low energy consumption results from the sleep of the CPU. In both applications, the samples are temporarily stored in the FIFO buffer. Although this is strictly necessary in the DT+CNN approach to be able to process the same samples with the DT and the CNN, it is used in the CNN-only application to avoid losing samples while the CPU executes the CNN.

THE ENERGY CONSUMPTION SIGNIFICANTLY INCREASES AS SOON AS THE CPU IS ACTIVE.

Furthermore, this allows us to send the system to a lower power mode during collection of the samples. Thus, the main difference is when the CPU has to be awakened, which determines the length of the phase of low average energy consumption. Whereas with the DT+CNN approach the CPU is only awakened to execute the CNN when the DT predicts class C_4, the CNN needs to be executed every 2 s in the CNN-only application. This is evident in the corresponding power profile. The energy consumption significantly increases as soon as the CPU is active. This phase of high average energy consumption can be further partitioned into two subphases: a phase of lower energy consumption followed by a phase of higher average energy consumption. The former varies in its duration and corresponds to reading samples from the FIFO. Although this is constant for the CNN-only application, the time in the DT+CNN application depends on the number of samples buffered in the FIFO. As the oldest data must be read first, it may be necessary to retrieve more samples than for just the last 2 s. After retrieving the samples, the CNN is executed, which corresponds to the most energy-hungry phase. Apart from this, both applications show similar energy consumption profiles.

To assess the differences between the two applications more clearly, the subsystems are analyzed separately. The average current consumption of the SoC and the ISM330DHCX, with and without executing a DT, is shown in Figure 6. The SoC shows the same characteristic current levels that were already evident in Figure 5. The associated phases can be attributed to sleep, data fetching, CNN execution, and returning to sleep. When reading the samples from the FIFO,

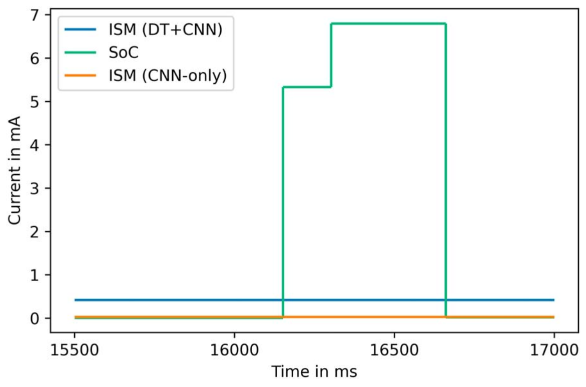

FIGURE 6. Average current over time of the nRF52840 SoC executing the CNN (green) and the ISM330DHCX with DT disabled (orange) and DT enabled (blue).

TABLE 1. Energy consumption for 60 s when a single circular motion occurs.

	CNN only	DT+CNN
Entire system	550.8 mJ	259.2 mJ
ISM330DHCX	5.6 mJ	75.7 mJ
nRF52840 SoC	293.4 mJ	15.7 mJ

$\sim$5.3 mA are required, whereas with 6.8 mA, an even higher demand persists when executing the CNN. Although in sleep, the SoC only required a current of 10 μA. Especially the demand in sleep mode can be minimized even further, as mentioned earlier.

When used within the CNN-only application, the ISM330DHCX requires an average current of $\sim$32 μA, labeled "ISM (CNN-only)" in Figure 6. However, when executing the DT+CNN application, an average current drain of approximately 420 μA is measured for the ISM330DHCX, labeled "ISM (DT+CNN)." Whether energy can be saved by the hybrid approach largely depends on how often a motion is detected that is to be classified by the CNN.

To compare the DT+CNN application to the CNN-only solution, we tested both applications with the same input sequence and determined the energy consumption. The input sequence has a duration of 60 s and comprises 29 occurrences of class C_0 and a single circular motion. Measurement results for the entire system as well as for the subsystems are given in Table 1. The difference between the energy consumption of the entire system and the sum of the energy consumed by the individual subsystems can be attributed to additional active components and peripherals on the nRF52840 DK and the X-NUCLEO-IKS02A1 sensor expansion board.

For the given example, the use of the DT+CNN approach would enable energy savings. However, the advantage diminishes the more often the CNN has to be executed. This is due to the fact that the ISM330DHCX requires more energy when executing the DT. Here, the worst case would be execution every 2 s. In this scenario, the CPU would be active for both cases equally often and for the same length of time, but the ISM would require more power. In our example and in this particular case, the CPU would consume the same amount of energy for both applications, but the ISM would consume more energy by running the DT.

It should be noted that the SoC must also be awakened for a short period of time to store the classification result if the DT predicts a noncircular motion. However, in this case, neither must the FIFO be read, nor must the CNN be executed. Instead, a counter can be increased. Furthermore, it is worth mentioning that this is only necessary when a prediction changes, avoiding a fixed wake-up interval.

The amount of energy saved therefore depends on the expected motions. Nevertheless, energy can clearly be saved by designing hybrid near-sensor in-sensor ML applications.

CONCLUSION

We presented how a near-sensor ML application can be transformed into a hybrid near-sensor in-sensor ML application with the goal of reducing energy consumption. In the sample application, a CNN used for motion classification was combined with a DT. The latter was executed in the sensor, allowing an increase of the time that the CPU is asleep and thus decreasing the required energy without a significant loss of accuracy. Low energy consumption can be essential in terms of whether or not a TinyML application can be deployed.

WHETHER ENERGY CAN BE SAVED BY THE HYBRID APPROACH LARGELY DEPENDS ON HOW OFTEN A MOTION IS DETECTED THAT IS TO BE CLASSIFIED BY THE CNN.

With sensors that are equipped with more general processing units, like the ISM330IS,[11] it becomes even more interesting to implement hybrid near-sensor in-sensor applications or to even shift the ML entirely to the sensor. Such sensors have the potential to allow multistage classifiers to be implemented, with several stages executed in the sensor. Determining which stages should be executed within the sensor versus close to the sensor is of particular interest. In this context, compilers such as (tiny) IREE,[5] which are designed to compile ML workloads for heterogeneous devices, could evolve as valuable tools in the future.

However, in connection with the design of increasingly complex TinyML applications the paradigm shift presented in Warden et al.,[12] should be considered.

ACKNOWLEDGMENTS

This work was supported by the German Federal Ministry of Education and Research and the state of North Rhine-Westphalia as part of the Lamarr Institute for Machine Learning and Artificial Intelligence.

REFERENCES

1. E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, “Estimation of energy consumption in machine learning,” *J. Parallel Distrib. Comput.*, vol. 134, pp. 75–88, Dec. 2019, doi: 10.1016/j.jpdc.2019.07.007.
2. S. Prakash et al., “Is TinyML sustainable? Assessing the environmental impacts of machine learning on microcontrollers,” 2023, *arXiv:2301.11899*.
3. F. Zhou and Y. Chai, “Near-sensor and in-sensor computing,” *Nature Electron.*, vol. 3, no. 11, pp. 664–671, Nov. 2020, doi: 10.1038/s41928-020-00501-9.
4. J. B. Yang, M. N. Nguyen, P. P. San, X. L. Li, and S. Krishnaswamy, “Deep convolutional neural networks on multichannel time series for human activity recognition,” in *Proc. 24th Int. Conf. Artif. Intell.*, 2015, pp. 3995–4001.
5. H.-I. C. Liu, M. Brehler, M. Ravishankar, N. Vasilache, B. Vanik, and S. Laurenzo, “TinyIREE: An ML execution environment for embedded systems from compilation to deployment,” *IEEE Micro*, vol. 42, no. 5, pp. 9–16, Sep./Oct. 2022, doi: 10.1109/MM.2022.3178068.
6. “ISM330DHCX: Machine learning core,” STMicroelectronics International N.V., Geneva, Switzerland, Appl. Note AN5392 (Rev 5), Apr. 2022. [Online]. Available: https://www.st.com/resource/en/application_note/an5392-ism330dhcx-machine-learning-core-stmicroelectronics.pdf
7. W.-Y. Loh, “Classification and regression trees,” *Wires Data Mining Knowl. Discovery*, vol. 1, no. 1, pp. 14–23, Jan. 2011, doi: 10.1002/widm.8.
8. M. Gazivoda and V. Bilas, “Always-on sparse event wake-up detectors: A review,” *IEEE Sensors J.*, vol. 22, no. 9, pp. 8313–8326, May 2022, doi: 10.1109/JSEN.2022.3162319.
9. F. Daghero, D. J. Pagliari, and M. Poncino, “Two-stage human activity recognition on microcontrollers with decision trees and CNNs,” in *Proc. 17th Conf. Ph.D Res. Microelectron. Electron. (PRIME)*, Villasimius, Italy, 2022, pp. 173–176, doi: 10.1109/PRIME55000.2022.9816745.
10. P. Li, Z. Qin, X. Wang, and D. Metzler, “Combining decision trees and neural networks for learning-to-rank in personal search,” in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2019, pp. 2032–2040, doi: 10.1145/3292500.3330676.
11. “ISM330IS: Always-on 3-axis accelerometer and 3-axis gyroscope with ISPU - Intelligent sensor processing unit,” STMicroelectronics International N.V., Geneva, Switzerland, Appl. Note AN5850 (Rev 1), Aug. 2022. [Online]. Available: https://www.st.com/resource/en/application_note/an5850-ism330is-alwayson-3axis-accelerometer-and-3axis-gyroscope-with-ispu–intelligent-sensor-processing-unit-stmicroelectronics.pdf
12. P. Warden et al., “Machine learning sensors,” 2022, *arXiv:2206.03266*.

MARIUS BREHLER is a senior scientist with Fraunhofer IML, Dortmund, 44227, Germany. His research interests include compilers for machine learning and machine learning on resource-constrained devices. Brehler received his Dr.-Ing. degree in electrical engineering from Technische Universität Dortmund. He is the corresponding author of this article. Contact him at marius.brehler@iml.fraunhofer.de.

LUCAS CAMPHAUSEN is a research associate with Fraunhofer IML, Dortmund, 44227, Germany. His research interests include different aspects of resource-aware machine learning and compilers for machine learning. Camphausen received his M.Sc. degree in mathematics from Westfälische Wilhelms-Universität Münster and his M.Sc. degree in computer science from Technische Universität Dortmund. Contact him at lucas.camphausen@iml.fraunhofer.de.

BENJAMIN HEIDEBROEK was a student research assistant (at the time of writing) with Fraunhofer IML, Dortmund, 44227, Germany. His research interests include embedded security and machine learning. Heidebroek received his B.Sc. degree in IT security from Ruhr-Universität Bochum. Contact him at benjamin.heidebroek@ruhr-uni-bochum.de.

DENNIS KRÖN was a student research assistant (at the time of writing) with Fraunhofer IML, Dortmund, 44227, Germany. His research interests include human activity recognition and machine learning. Krön received his B.Sc. degree in computer science from Technische Universität Dortmund. Contact him at dennis.kroen@tu-dortmund.de.

HENRI GRÜNDER is a student research assistant with Fraunhofer IML, Dortmund, 44227, Germany. His research interests include various areas in embedded system software. Gründer received his B.Sc. degree in computer science from Technische Universität Dortmund. Contact him at henri.gruender@iml.fraunhofer.de.

SIMON CAMPHAUSEN is a research associate with Fraunhofer IML, Dortmund, 44227, Germany. His research interests include compilers for machine learning workloads and execution of machine learning models on memory-constrained systems. Camphausen received his M.Sc. degree in mathematics from Westfälische Wilhelms-Universität Münster and his M.Sc. degree in computer science from Technische Universität Dortmund. Contact him at simon.camphausen@iml.fraunhofer.de.