12

Embedded Edge Intelligent Processing for End-To-End Predictive Maintenance in Industrial Applications

Ovidiu Vermesan[1] and Marcello Coppola[2]

[1]SINTEF AS, Norway
[2]STMicroelectronics, France

Abstract

This article advances innovative approaches to the design and implementation of an embedded intelligent system for predictive maintenance (PdM) in industrial applications. It is based on the integration of advanced artificial intelligence (AI) techniques into micro-edge Industrial Internet of Things (IIoT) devices running on Arm® Cortex® microcontrollers (MCUs) and addresses the impact of a) adapting to the constraints of MCUs, b) analysing sensor patterns in the time and frequency domain and c) optimising the AI model architecture and hyperparameter tuning, stressing that hardware–software co-exploration is the key ingredient to converting micro-edge IIoT devices into intelligent PdM systems. Moreover, this article highlights the importance of end-to-end AI development solutions by employing existing frameworks and inference engines that permit the integration of complex AI mechanisms within MCUs, such as NanoEdge™ AI Studio, Edge Impulse and STM32 Cube.AI. Both quantitative and qualitative insights are presented in complementary workflows with different design and learning components, as well as in the backend flow for deployment onto IIoT devices with a common inference platform based on Arm® Cortex®-M-based MCUs. The use case is an n-class classification based on the vibration of generic motor rotating equipment. The results have been used to lay down the foundation

of the PdM strategy, which will be included in future work insights derived from anomaly detection, regression and forecasting applications.

Keywords: predictive maintenance, smart sensors systems, industrial internet of things, industrial internet of intelligent things, vibration analysis, machine learning, deep learning architecture, edge-embedded devices.

12.1 Introduction and Background

Leveraging AI methods and techniques at the edge is vital for increasing the performance and capabilities of the intelligent sensor systems and IIoT devices used in industrial manufacturing. For many intelligent applications, the edge AI processing concept is reflected in the emergence of different edge layers (micro-, deep-, meta-edge). The edge processing continuum includes the sensing, processing and communication devices (micro-edge) close to the physical industrial assets under monitoring, the gateways and intelligent controllers processing devices (deep-edge), and the on-premise multi-use computing devices (meta-edge). This continuum creates a multi-level structure that moves up in processing, intelligence, and connectivity capability.

Micro-edge devices are typically small sensors and actuators equipped with microcontrollers (MCUs) based on Arm® Cortex®-M cores (e.g., M0, M0+, M3, M4, M7) or open-source RISC-V instruction set architecture, circuits with memory, serial ports, peripherals, and wireless capabilities and designed to perform and extend the specific tasks of embedded systems.

Developing AI functionalities for micro-edge devices is a complex process that has increased potential in various industrial applications, including manufacturing. In industrial manufacturing, the implementation of machine learning (ML) and deep learning (DL) models on micro-edge-embedded devices has an absolute advantage for condition monitoring and PdM/prescriptive maintenance (PsM) operations for industrial motors/equipment. Using AI-enabled micro-edge devices for motors/equipment monitoring in industrial processes can prevent downtime by alerting users to perform preventative maintenance based on equipment real-time conditions.

There are several works that provide a comprehensive review of frameworks available in the market that currently permit the integration of complex ML and DL mechanisms within MCUs [1][4].

This article researches and investigates different approaches to using ML and DL technologies to bring AI capabilities to micro-edge devices and applies these capabilities for classification for PdM industrial applications. The goal is to implement ML and DL techniques in low-energy systems, including sensors, to perform intelligent automated tasks, such as PdM and PsM.

The approaches used in this article illustrate how to optimise ML and DL models for resource-constrained micro-edge-embedded devices. The article gives an overview of the data acquisition and prediction aspects of ML and DL, discusses how to build ML and DL models targeting micro-edge devices and presents the experimental results using different tools and approaches.

The article is organised into five sections. The introduction on intelligent edge processing real-time maintenance systems and description of data-, model- and knowledge-driven methods for time series is included in Section 12.1. Section 12.2 describes the architecture and design of motor classification for PdM, including methods and possible end-to-end flows and presents the use case, i.e., motor classification. Section 12.3 introduces the implementation of the classification use case using three existing platforms. Section 12.4 highlights specific experiments performed and the results that were achieved through the lens of employing different tools. Section 12.5 addresses future research challenges and discusses the key open issues related to AI techniques and methods in implementing intelligent edge processing real-time maintenance systems for the purposes of industrial applications.

12.2 Machine and Deep Learning for Embedded Edge Predictive Maintenance

For industrial manufacturing facilities using motors in the process line, the maturity of maintenance practices is a crucial determinant of the ability to operate reliably and profitably without interruption. Condition-based monitoring maintenance (CBM) addresses uptime and maintenance costs by monitoring one or several critical measurements for the motors, such as temperature, vibration, oil analysis and current, which are used as indicators of an out-of-specification condition. Maintenance tasks are performed when needed. PdM applies a more extensive set of input data and more analysis to provide a more reliable indicator of the overall health and condition of the motor as well as a more accurate prediction of a possible failure and what action should be considered to prevent it.

With PdM, the motors are serviced considering the actual wear and tear and service needs, reducing unexpected outages, making fewer scheduled maintenance repairs or replacements, and using fewer maintenance resources (including spare parts and supplies) while simultaneously decreasing failures. PdM provides the prerequisite foundation for PsM and autonomous maintenance (by executing actions automatically, without human intervention). PsM builds on the infrastructure and data collected for PdM, following the various corrective actions taken by maintenance personnel and the resulting outcomes.

Figure 12.1 illustrates a typical industrial motor with a rotor, stator, bearings, and shaft as essential components for the engine's normal operation.

The various components conditions and operations are possible causes that can generate anomalous behaviour, thus defining various abnormal states (classes). A large amount of historical and real-time information is required to identify, classify, and predict motor's possible failures. AI-based ML and DL algorithms are suitable to deal with these types of tasks.

This paper focuses on AI-based PdM approaches, which learn from historical and real-time data and recommend the best timing and course of action for a given set of conditions and sub conditions employing ML and DL models implemented using micro-edge-embedded devices. For example, the implementation of an ML solution into a PdM application includes several steps: data preparation, feature engineering, algorithm selection and parameter tuning.

The interaction between the edge IIoT devices, ML and DL have opened opportunities for new data-driven approaches for PdM solutions in industrial processes. In this paper, different techniques and tools were successfully tested using various methods based on ML and DL to predict the state of industrial motors and to detect and classify motors conditions based on trained data. The PdM monitoring has been tested on measurements

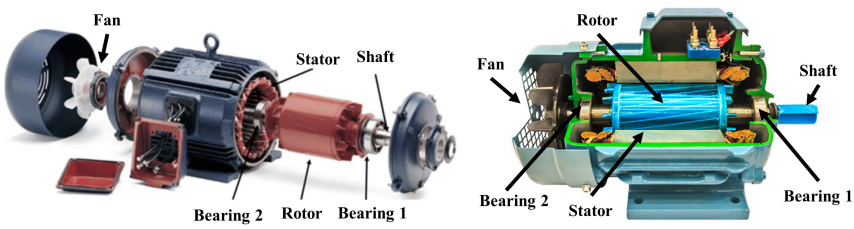

Figure 12.1 Industrial motor components **[5] [6]**

performed on bench motors using computation at the micro-edge, allowing real-time acquisition, processing, and wireless communication.

12.3 Approaches for Predictive Maintenance

AI-based PdM approaches [2][3][7], employing ML and DL models implemented using micro-edge-embedded devices, are designed on different hardware platforms and software suites, generating embedded code, and performing learning and inference engine optimisations. Depending on the application and the frameworks and inference engines for integrating AI mechanisms within MCUs, several variants of the workflows are used.

This paper focuses on NanoEdge^{TM} AI (NEAI) Studio [14], Edge Impulse (EI) [8][10] and STM32 Cube.AI [10][13].

Table 12.1 gives an overview of the features of these frameworks, which support the workflows of ML and DL model development and deployment on microcontroller class devices. AI/ ML models work on frameworks such as Keras, ONNX, Lasagne, Caffe, Convetjs etc.

Table 12.1 Frameworks and inference engines for integrating AI mechanisms within MCUs

Framework	Platforms	Models	Training Libraries
Edge Impulse (EI)	Arm® Cortex®-M, TI CC1352P, Arm® Cortex®-M -A, Espressif ESP32, Himax WE-I Plus, TENSAI SoC	NN, k-means, regressors (including feature extraction)	Tensor Flow, Scikit-Learn
Nano Edge AI Studio (NEAI)	Arm® Cortex®-M (STM32 series)	Unsupervised learning	-
STM32Cube.AI	Arm® Cortex®-M (STM32 series)	NN, k-means, SVM, RF, kNN, DT, NB, regressors	PyTorch, Scikit-Learn, Tensor Flow, Keras, Caffe, MATLAB, Microsoft Cognitive Toolkit, Lasagne, ConvNetJS

12.3.1 Hardware and Software Platforms

The experiments in this paper perform the processing of various types of input data, including three-axis vibration, temperature, and device logs. The data for the experiments was collected from bench motors using a STWIN Sensor Tile Wireless Industrial Node IIoT device.

This micro-edge IIoT device comprises of three axis ultrawide bandwidth (DC to 6 kHz) acceleration sensor (ISM330DHCX), a 12-bit analog-to-digital converter, a user-configurable digital filter chain, a temperature sensor, and a serial peripheral interface. The micro electro mechanical systems (MEMS) vibration sensor has a selectable sensitivity (± 2, ± 4, ± 8, or ± 16 g) and processing capabilities ensured by an Arm® Cortex®-M4 processor (120 MHz, 640 KB RAM, 2 MB Flash). The micro-edge device can be powered externally or by an internal lithium-ion battery and has BLE and Wi-Fi connectivity.

The design flow allows collecting or uploading training data from micro-edge devices, labelling the data, training an ML model, and launching and monitoring ML models in a production environment.

The PdM AI-based design flow uses the sensors and hardware platforms, software development kits (SDKs), frameworks and inference engines for integrating AI mechanisms within MCUs to generate code to be deployed on MCUs that allow running AI models in embedded systems by performing predictions at the edge. The ML and DL models deployed on the micro-edge devices become part of the firmware flashed into the MCUs.

A micro-edge AI processing flow is illustrated in Figure 12.2.

The AI-based flow uses an embedded compiler that can convert models to C/C++ to increase the efficiency of models trained on the edge platform and reduce RAM, storage usage and code size by tens of percent.

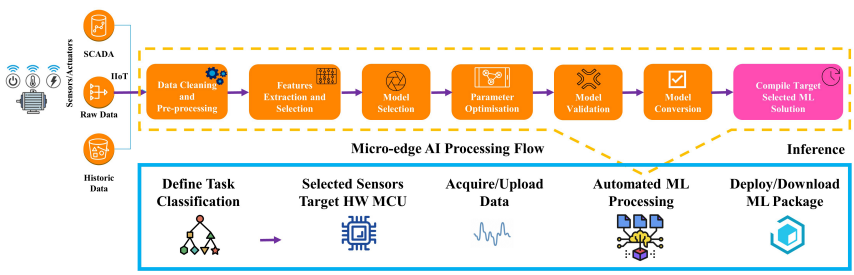

Figure 12.2 Micro-edge AI processing flow

12.3.2 Motor Classification Use Case

The use case analysed in this article is the classification of the state of a motor based on the vibration measurements using an accelerometer sensor from an IIoT device. The signals covering all states to be classified were collected using a built-in three-axis accelerometer (ISM330DHCX) to measure the accelerations of three orthogonal directions.

In general, the n-class classification of n different states uses static models with pretrained libraries.

The classes were defined based on conditions (motor speeds) and sub-conditions (malfunctions). The motor was operating at fixed speeds, which were divided into three classes based on various percentages of the maximum speed (50%, 75% and 100%). A malfunction of the motor (motor fan trepidations) was added to the second class to obtain a new class. The classes defined are:

- MOTOR_OFF: just record signals when nothing is happening
- MOTOR_ON_NORMAL_50: the motor is running at 50% of the maximum speed
- MOTOR_ON_NORMAL_75: the motor is running at 75% of the maximum speed
- MOTOR_ON_NORMAL_75_B: the motor fan produces additional trepidations to the motor, while the motor is running at 75% of the maximum speed
- MOTOR_ON_NORMAL_MAX: motor is running at maximum speed.

12.4 Experimental Setup

The design and implementation steps and the experimental setup of the end-to-end (E2E) classification application use two main primary flows, including NEAI Studio and EI. The former creates ML static libraries based on unsupervised algorithms, while the later employs deep neural networks (NNs) for the classification task. A third flow was branched out from EI into Python using Tensor Flow's Keras API, and the resulted model was fed onto STM32Cube.AI.

The experimental process started by collecting the vibration signals from the micro-edge IIoT device mounted on the motor, through a simple data-logger application in real-time. The recorded signals were then analysed in both the time and frequency domain, filtrated, and datasets were prepared for each flow. The classification AI models were then built in each flow, using

the accelerometer spectral features (e.g., root mean square (RMS), frequency and amplitude of spectral power peaks, etc.) and optimise the performance. In the end the three models were deployed and integrated with the firmware using STM32 CubeIDE. Finally, inference classifications were run to note the performance of the implementations and deployments.

12.4.1 Signal Data Acquisition and Pre-processing

Prior to acquiring the signals, a thorough analysis of the vibration patterns of the motor have been conducted, landing to the conclusion that the most suitable sampling frequency to capture vibration patterns is 1667 Hz.

Both NEAI and EI offer several ways to take the measurements from the sensor IIoT device directly from within their GUIs. Acquiring signals with datalogger functionality in NEAI seemed to be the most straightforward data acquisition approach as it only requires the SD card. In the experimental use case, a simple logger application was used that reads and logs the raw accelerometer sensor data directly on the serial port, so that logs can be retrieved from a computer using serial tools such as Tera Term or from the console of the integrated development environment (IDE).

For the three-axis accelerometer sensor, a collection of signals (split in 60% training, 20% validation and 20% test) was acquired for each of the classes, with a buffer size of 512 samples on each axis, in total 1536 values per signal. Thus, with a sampling frequency of 1667 Hz, each buffer represents a snapshot of approximately 300 milliseconds of the accelerometer temporal vibration data, which is sufficient to capture the essence of the motor vibration patterns. The vibration signals collected are visualised as shown in Figure 12.3, in both temporal and frequency plots for the accelerometer sensor Z-axis for each of the two classes.

To be able to better differentiate the individual classes and thus ensuring high accuracy score, the recorded signals were processed in frequency domain. Filter settings was activated in the signals pre-processing steps. By providing filtering, only the frequencies that represent the characteristics of the motor vibration are kept, and the rest are attenuated. Filtering techniques also help to eliminate high frequency noise that interfere with the vibration signal, and eliminate frequencies for transitions between states, which would normally yield unknown class.

The recorded signals for each class were downloaded and then converted into a format accepted by EI, to ensure the same signals are being used for the signal processing, thus yielding similar results.

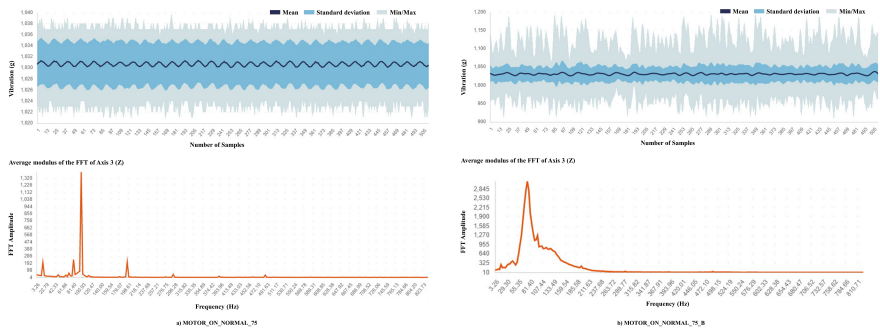

Figure 12.3 Visualisation of two selected classes signals in both temporal and frequency domain with NEAI

Till acceptable quality-labelled data sets were arrived at, several iterations were performed, and this included recording new signals without background noise, collecting/recording longer signals and even changing the categorisation of classes.

12.4.2 Feature Extraction, ML/DL Model Selection and Training

Both NEAI and EI offer an automated mechanism for generating the AI model architecture and training, although the mechanisms differ since NEAI employs unsupervised algorithms, whereas EI employs DL NNs.

The benchmarking process for n-class classification with NEAI involves searching through a pool of ML algorithms and tests combinations of three elements: pre-processing, ML algorithms (e.g., random forest-RF, support vector machines-SVM, etc.) and hyper-parameters for each model. Each combination results in a library that is evaluated for accuracy, confidence and memory usage, and the results provide a ranking of these libraries. Accuracy reflects the library's ability to correctly attribute each signal to the correct class, whereas confidence reflects the library's ability to separate the n-classes.

Figure 12.4 shows that the top library for the PdM classification case has an accuracy of 100%, confidence 99.94%, uses the RF algorithm, and takes 6.2kB RAM and 8.3 kB Flash. 100% means that all classes are completely separated, there is no overlap.

In the "Confusion Matrix", the 200 number means that the performance for each class is 100%, i.e., all 200 signals extracted from initial data (20% of 1000 signals) have been properly classified.

Figure 12.4 Benchmarking with NEAI

In the EI platform, a *Spectral Analysis* signal processing block was used to apply a filter, perform spectral analysis, and extract frequency and spectral power data. A useful aspect of the platform is the possibility to visualise and explore the features (Figure 12.5). The fact that the features are visually clustered is a good indication that the model can be trained to perform the classification. During the first iterations, the features overlapped to a significant degree and were intertwined, and the trained model had difficulties differentiating between classes. This problem was addressed by collecting more signals and increasing the size of the sampling signal to better capture signal patterns.

It is also possible to calculate and visualise feature importance when generating the features, indicating how important the features are for each class compared with all other classes. RMS and peak values of vibration along the three-axis proved to be the most important features in determining the class in this case. Based on this information, the dimension reduction algorithms can be used to simplify the model by deleting the less important or redundant information from the data set to make it manageable while maintaining relevance and performance.

To implement the solution in EI, a classification learning block was used, which employs TensorFlow with Keras. It takes the features from *Spectral Analysis* signal processing block and learns to distinguish between the five classes. The strategy adopted was to start with a small deep NN model and experiment with it, i.e., two dense layers, using EI graphical user interface (GUI). Most of the experimentations have been performed around an architecture consisting of multiple dense layers and dropout layers. Convolutional layers were also included.

Figure 12.5 Snapshots of Feature Explorer in EI based on the pre-processing block early in the process.

At the end of the training, the model's performance and the confusion matrix of the validation data can be evaluated. Figure 12.6 shows an accuracy and a loss on the training and validation datasets, comparable with the results obtained with NEAI with a different model architecture. To avoid overfitting, the learning rate was reduced, and more data was collected, and the model was re-trained.

12.4.3 Optimisation and Tuning Performance

Developing the most efficient ML/DL flows for the classification PdM application was challenging. It required many iterative experiments and insights into the workings of motor vibration patterns, digital signal processing, AI algorithms, architectures, and microcontrollers. Nevertheless, both NEAI and EI provided automation and transparency for these processes, though to varying degrees.

Figure 12.6 Confusion Matrix and Data Explorer based on full training set: Correctly Classified (Green) and Misclassified (Red).

For the NEAI classification, the learning is fixed at library generation based on the data provided for each class. The benchmarking implementation includes patented elements; thus, the internal working of the engine is not transparent. Nevertheless, multiple benchmarks can be created, and a high degree of automation allows for the best results to be obtained from signal capturing and formatting. The benchmarking process takes around 60 minutes when running on a processing unit with 6 CPU cores.

EI offers a higher degree of transparency and control over the model architecture and hyperparameters. The strategy adopted for the case of EI was to start from a simple model, experiment with it and improve it into a deeper and wider model. For this improvement step and for validation purposes, a parallel sub-flow was branched out from the flow with EI to conduct experiments in a Python framework. The training was launched in both EI and Python and compared throughout. The updated architecture and

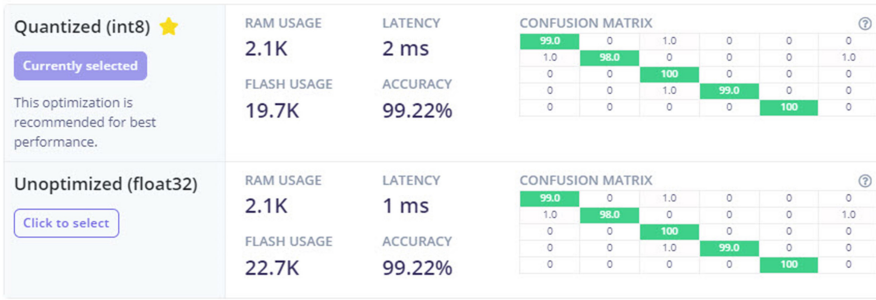

Figure 12.7 A comparison between int8 quantized and unoptimized versions of the same model, showing the difference in performance and results.

hyperparameters were exchanged back and forth between the EI and Python frameworks.

The improvements consisted in making the model deeper by adding more layers, and wider by increasing the number of hidden units, changing the activation and optimisation functions, learning rate, fitting more data.

While the improvement process was run manually in Python, the EI's Edge Optimized Neural (EON^{TM}) Compiler [9] can be used to find the best solution for the Arm® Cortex®-M-based MCUs, i.e., the most optimal combination of processing block and ML model for the given set of constraints, including latency, RAM usage, and accuracy. Currently, there are a limited number of MCUs that are supported and does not include the MCU of STWIN IIoT device (Arm® Cortex®-M4 MCU STM32L4R9), which operates at a frequency of up to 120MHz. Nevertheless, the estimated on-device performance could be evaluated for Cortex-M4F 80MHz, to determine the impact of optimisations such as quantisation across different slices of the datasets (Figure 12.7).

12.4.4 Testing

ML/DL model testing usually refers to the evaluation of the trained model on the testing dataset to analyse how well the model performs against unseen data. However, model testing in NEAI and EI provide more than that. Both platforms provide a microcontroller emulator to test and debug the generated model prior to its deployment on the device.

As part of the NEAI toolkit, a microcontroller emulator is provided for each library to test and debug the generated model without the need to download, link or compile. Test signals can be imported from file; however,

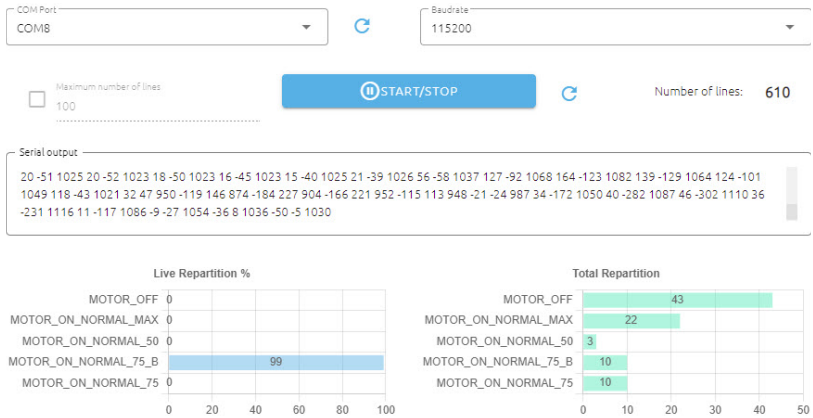

Figure 12.8 Evaluation of trained model using NEAI Emulator with live streaming.

the signals were imported live from the same datalogger application through serial port, in this way ensuring completely new signals, not seen before. The classification is automatically run using the live signals, while changing motor speeds and triggering shaft disturbances, to switch between classes and cover all five states and classes.

The results are presented in Figure 12.8, showing that the classifier managed to properly reproduce and detect all classes with reasonable certainty percentages.

In EI, the trained model was evaluated by assessing the accuracy using the test dataset. To ensure unbiased evaluation of model effectiveness, the test samples were not used directly or indirectly during training. The EI emulator took care of extracting the features from the test set, running the trained model, and reporting the performance in the confusion matrix. The results are shown in Figure 12.9.

12.4.5 Deployment

In the context of micro-edge embedded systems, model deployment is dependent on the hardware/software platform and is more or less automated, and in essence comprises three steps: the first is a format conversion of the fully trained model; the second is a weight/model compression to reduce the amount of memory to store the weights in the target hardware platform and to simplify the computation so it can run efficiently on target processors. The third entails compiling the model and generating the code to be integrated with the MCUs firmware.

Figure 12.9 EI model testing with test datasets.

The back-end flow consists of wrapping an STM32CubeIDE project with the generated files from the three deployed models, adding functionality on top such as retrieving the accelerometer values to be fed to the classification function and displaying the result, then compiled, built, and flashed onto the MCU target.

The flow exhibits some particularities in the case of the three model deployments.

In the case of NEAI, the selected model is deployed in the form of a static library (libneai.a), an AI header file (NanoEdgeAI.h) containing functions and variable definitions, and a knowledge header file (knowledge.h) containing the model's knowledge. In this case, first the knowledge was initialised, then the NanoEdge AI classifier was run, and the output was print to the serial port.

For the EI deployment, the CMSIS-PACK [11][12] for STM32 packaged all signal processing blocks, configuration and learning blocks up into a single library (.pack file), which was then added to the STM32 project using the CubeMX packages manager. This is currently only supported for C++ applications using CubeIDE.

Figure 12.10 Live classification streaming with detected state and confidence (with Tera Term)

The third flow was branched out from EI and further developed in a Python framework using TensorFlow's Keras API. The resulted model was converted into optimised C code with STM32 Cube.AI, an extension of the CubeMX tool, which offers simple and efficient interoperability with other ML frameworks.

12.4.6 Inference

Inference classifications have been conducted with all applications running directly from the target hardware platform on the micro-edge IIoT devices, producing classification in real-time.

The state machine consists mainly of two states with two functions "init" and "inferencing", respectively, with the former initialising the deep NN model and the latter being a continuously running function for collecting raw data from the sensors on the micro-edge IIoT device and making classifications in real-time. A snapshot from the classification based on the NEAI model is shown in Figure 12.10.

The "?" indicate the state switching, which happens after several consecutive confirmations of inference result is encounter, and this number is programmable.

12.5 Discussion and Future Work

Embedding trained models into the firmware code enables AI/ML capabilities of intelligent edge devices. Employing different frameworks that permit the integration of complex AI mechanisms within MCUs - such as NEAI Studio, EI and STM32 Cube.AI - for deploying AI-based PdM solutions into micro-edge embedded devices provides designers with the flexibility to optimise implementation by experimenting with deployment on the same hardware platform target using several frameworks and inference engines. The different workflows can be matched to the PdM application requirements for generating embedded code and performing learning and inference engine optimisations.

ML and NNs can now be efficiently deployed on resource-constrained devices, which enable cost-efficient deployment, widespread availability, and the preservation of sensitive data in PdM applications. However, the trade-offs associated with optimisation methods, software frameworks and hardware architecture on performance metrics, such as inference latency and energy consumption, are yet to be studied and researched in depth.

This preliminary work allowed for the exploration of different scenarios to evaluate trade-offs between computational cost and performance on actual classification tasks, laying the foundation for further investigations of more complex PdM systems using various AI-based techniques. Future work will aim to enlarge comparison and benchmarking by considering more edge ML and DL technologies, workflows, and datasets. A more generic and complete PdM strategy must include insights from other applications, such as anomaly detection, regression, and forecasting.

Acknowledgements

This work is conducted under the framework of the ECSEL AI4DI "Artificial Intelligence for Digitising Industry" project. The project has received funding from the ECSEL Joint Undertaking (JU) under grant agreement No 826060. The JU receives support from the European Union's Horizon 2020 research

and innovation programme and Germany, Austria, Czech Republic, Italy, Latvia, Belgium, Lithuania, France, Greece, Finland, Norway.

References

[1] R. Sanchez-Iborra and A.F. Skarmeta, "TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities," in *IEEE Circuits and Systems Magazine*, vol. 20, no. 3, pp. 4-18, third quarter 2020. https://doi.org/10.1109/MCAS.2020.3005467

[2] T. Hafeez, L. Xu and G. Mcardle, "Edge Intelligence for Data Handling and Predictive Maintenance in IIoT," in *IEEE Access*, Vol. 9, pp. 49355-49371, 2021. https://doi.org/10.1109/ACCESS.2021.3069137

[3] Y. Liu, W. Yu, T. Dillon, W. Rahayu and M. Li, "Empowering IoT Predictive Maintenance Solutions With AI: A Distributed System for Manufacturing Plant-Wide Monitoring," in *IEEE Transactions on Industrial Informatics*, vol. 18, no. 2, pp. 1345-1354, Feb. 2022. https://doi.org/10.1109/TII.2021.3091774

[4] H. Wang, H. Sayadi, S.M. Pudukotai Dinakarrao, A. Sasan, S. Rafatirad and H. Homayoun, "Enabling Micro AI for Securing Edge Devices at Hardware Level," in *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 4, pp. 803-815, Dec. 2021. https://doi.org/10.1109/JETCAS.2021.3126816

[5] F. Cipollini, L. Oneto, A. Coraddu, et al. "Unsupervised Deep Learning for Induction Motor Bearings Monitoring". Data-Enabled Discov. Appl. 3, 1, 2019. https://doi.org/10.1007/s41688-018-0025-2

[6] M. Guenther. 6 Ways to Improve Electric Motor Lubrication for Better Bearing Reliability. Available online at: https://blog.chesterton.com/lubrication-maintenance/improving-electric-motor-lubricaiton/

[7] C. Kammerer, M. Gaust, M. Küstner, P. Starke, R. Radtke, and A. Jesser, "Motor Classification with Machine Learning Methods for Predictive Maintenance," *IFAC-PapersOnLine*, vol. 54, no. 1, pp. 1059–1064, 2021. https://doi.org/10.1016/j.ifacol.2021.08.126

[8] Edge Impulse. Available online at: https://www.edgeimpulse.com

[9] EON Tuner. Available online at: https://docs.edgeimpulse.com/docs/eon-tuner

[10] J. Jongboom, 2020. "Learning for all STM32 developers with STM32Cube.AI and Edge Impulse". Available online at: https://www.edgeimpulse.com/blog/machine-learning-for-all-stm32-developers-with-stm32cube-ai-and-edge-impulse

[11] ARM-NN. 2020. Available online at: https://github.com/ARM-software/armnn
[12] CMSIS-NN. 2020. Available online at: https://arm-software.github.io/CMSIS_5/NN/html/
[13] STM32Cube.AI 2020. Available online at: https://www.st.com/en/embedded-software/x-cube-ai.html
[14] NanoEdge^{TM} AI Studio. Automated Machine Learning (ML) tool for STM32 developers. Available online at: https://www.st.com/en/development-tools/nanoedgeaistudio.html