2022 IEEE 18th International Conference on Automation
Science and Engineering (CASE)
August 20-24, 2022. Mexico City, Mexico

2022 IEEE 18th International Conference on Automation Science and Engineering (CASE) | 978-1-6654-9042-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/CASE49997.2022.9926482

Performance Evaluation of AI Algorithms on Heterogeneous Edge Devices for Manufacturing

Bernhard Rupprecht[1], Dominik Hujo[1], Birgit Vogel-Heuser[2], *Senior Member, IEEE*

***Abstract*—Novel Artificial Intelligence (AI) approaches try to process an excessive amount of field-level data. However, challenges arise as network bandwidth is limited, and thus this data cannot be entirely transferred to the cloud for further processing. Edge computing tries to overcome that limitation by bringing the computational resources closer to the data generating sources. However, edge devices are also constraint by both CPU power and memory, and strict real-time requirements of the manufacturing domain have to be met. Thus, the selection of suitable devices for specific AI algorithms poses a severe challenge. Currently, the choice is often made by a trial-and-error approach or by selecting more powerful devices than needed. This paper tries to address those challenges by showing relevant aspects for algorithm benchmarking in the manufacturing domain. Selected algorithms, namely Grubbs Test, Butterworth Filter, DBSCAN, Random Forest, Support Vector Machine, Matrix Multiplication, and Matrix Inversion, are examined. Analysis of their theoretical time and space complexity sheds some light on the behaviour of the algorithms with respect to their input data points. In addition, relevant metrics for the manufacturing domain, such as execution time, memory consumption, and energy consumption, are identified. This paper furthermore examines the algorithm behaviour on various heterogeneous hardware devices, such as PLCs, an MCU, IPCs, a single-board computer, and a dedicated edge device. Altogether, this paper can guide selecting suitable algorithms and hardware to equip Cyber-Physical Production Systems (CPPS) with novel data processing solutions thoughtfully. Moreover, the presented metrics can support the creation of novel ML benchmarks for smart manufacturing (SM).**

I. Introduction

Due to the availability of vast amounts of data in I4.0 systems, novel big data approaches trying to use the additional information are not far to seek [1]. However, despite numerous applications and benefits, only about 13-20% of big data projects generate valuable and sustainable solutions [2]. For example, challenges arise during data analysis and interpretation, as many data push devices at the shop-floor to their limits because of their limited computing power and bounded network capabilities. These limitations pose additional challenges in efficient and fast data-processing as many manufacturing processes often need fast and high quality decisions despite limited computational resources [3]. However, novel interconnected approaches, like cloud computing, can help to overcome those challenges by providing hardware with high computational capabilities [4]. In contrast, the edge computing paradigm tries to overcome limited network capabilities and enable real-time data processing by bringing computational resources closer to the data generating sources [5]. In terms of software, novel Machine Learning (ML) algorithms can automatically identify patterns from high dimensional data sets and can be used in data preparation to address data-related issues, such as outliers detection [6]. Nonetheless, new challenges arise as suitable ML algorithms have to be selected with hardware architectures and devices. Currently, this selection process is mostly performed by a trial-and-error approach or by selecting more powerful devices than needed. Notably, algorithms heavily differ in their time- and space-complexity, as well as in their actual ML tasks. In addition, comparison of edge suitable hardware is rare. ML research often considers metrics not suitable to meet all requirements of the manufacturing domain, such as real-time constraints. There is a lack of an overview guiding the algorithm and hardware selection process for novel use cases. Furthermore, the behaviour of ML algorithms on single hardware devices is primarily unknown. Thus, this paper tries to overcome the above-mentioned challenges by investigating the following contributions:

- **C1** An overview of the properties of selected ML algorithms supports their feasible deployment in the manufacturing domain, including the retrofit of existing systems.
- **C2** The examination of the behavior of basic ML Algorithms on heterogeneous hardware platforms enables to choose appropriate ML inference hardware for novel use-cases prior to their implementation.
- **C3** The introduction of metrics tailored to the manufacturing domain supports the assessment of ML algorithms in the context of SM. It can support the creation of novel ML benchmarks in the manufacturing domain.
- **C4** The examination of the behavior of basic ML Algorithms on heterogeneous hardware platforms reveals unexpected behaviour and specialities.

The remainder of this paper is structured as follows: Section II introduces state-of-the-art literature in the field of edge computing and identifies the addressed research gap. Section III describes the considered algorithms. The following Section IV, states the concept for algorithm evaluation, including relevant metrics. The findings of this paper can be found in Section V. Finally, the last Section VI summarises the main contributions and gives an outlook on further research.

The authors are with the Institute of Automation and Information Systems, TUM School of Engineering and Design, Technical University of Munich, Germany
[1]firstname.lastname@tum.de
[2]vogel-heuser@tum.de

II. State-of-the-Art in Edge Applications and Edge Benchmarks

In the field of ML algorithm assessment on edge devices, research currently addresses two domains. Firstly, many specific ML algorithm implementations that process unique data sets exist. Secondly, ML benchmarks and metrics are used to assess algorithms on edge devices.

A. ML Edge Applications

Considering edge applications, Lee *et al.* [7] present a storing, monitoring, and preprocessing system using Node-RED. Downsampling and data normalization were applied as preprocessing tasks. Data were then fed into a decision tree. The algorithm runs on a MCU, representing an edge device. Yan *et al.* [8] developed a tool wear prediction method based on edge data processing. This method makes use of data preprocessing on edge devices to reduce network load and transmission delays. Data collected from sensors is down-sampled to minimize data size. The preprocessed data are then transmitted to the cloud, where it is normalized and a Neural Network (NN) performs the prediction task. A nearly unsupervised approach is proposed by Krüger *et al.* [9]. They present a novel approach for quality prediction on resource constraint edge devices. A Random-Forest-Regression (RFR) with 100 trees and a K-Nearest-Neighbour-Regression (KNNR) are used for quality prediction tasks. The approach is evaluated by performing anomaly detection on a powder metallurgy production process. The production cycle time of 10s is stated as a time constraint for the prediction.

Furthermore, many publications that review and summarise various ML implementations in SM exist. For instance, Fahle *et al.* [10] present a systematic review of ML techniques in the factory environment. Considered algorithms are NNs, DT, SVM, RF, and KNN. It includes applications related to manufacturing process planning, predictive maintenance, quality control, etc.. Moreover, Zhang *et al.* [11] present and classify a vast number of SM applications making use of six ML algorithms for PdM: Logistic Regression (LR), SVM, RF, ANN, DNN, and AE. They conclude that both ML and DL applications can complete the PdM task with high accuracy up to 100%. Angelopoulos *et al.* [12] published a survey focusing on fault detection, prediction, and prevention. They present various publications addressing fault detection by different ML algorithms, such as SVM, K-Means, or DBSCAN. Finally, Dogan and Birant [13] propose another literature review and group SM approaches in the four groups: scheduling, monitoring, quality, and failure. A vast number of specific implementations with algorithms in the domains of supervised and unsupervised learning, association rule mining, ensemble learning, and deep learning are presented.

Altogether, the presented implementations are usually focused to one specific use-case, not necessarily in the manufacturing domain. Results obtained by those implementations are not always valid for other domains or use-cases, as results can heavily differ on different data sets.

B. ML and Edge Benchmarks

Considering ML benchmarks, the MLCommons MLPerf benchmark [14, 15], is a semantic-level benchmark that represents a consensus on the best benchmarking techniques evaluated by over 200 ML engineers. Tasks covered by MLPerf are image classification, objection detection, and language translation. MLPerf defines tasks and rules to perform the benchmark but leaves the implementations to the user guaranteeing maximum flexibility. Depending on the evaluation scenarios, MLPerf defines different metrics. Examples are latency, latency-bounded throughput, throughput, and a maximum number of inferences per query. DAWNBench [16] is a benchmark that focuses on end-to-end training performance evaluation at a predefined accuracy level. Considered tasks are image classification and question answering. As the benchmark only specifies the task, it facilitates innovations considering new architectures, algorithms, and hardware. Proposed metrics are training time, training/inference costs (in USD), and inference latency. Another benchmark is DeepBench [17] by Baidu Research. DeepBench evaluates operations important to DL on different hardware platforms. Notably, DeepBench does not consider actual DL tasks ready for applications. Instead, low level operations that form the basis of many DL models, like matrix multiplication, are examined. This enables the comparison of different hardware, like Powerful Graphics Processing Units (GPUs), mobile, and embedded devices, independent of the application. Some considered metrics are latency, precision, inference time, and Floating Point Operations per Second.

Furthermore, an increasing number of benchmarks focus on ML on edge devices. For instance, Hao *et al.* [18] introduce Edge AIBench, a benchmark for end-to-end edge computing. The benchmark considers client-side devices, the edge computing layer, as well as cloud servers. Luo *et al.* [19] propose AIoT Bench to evaluate the AI ability of mobile and embedded devices. Covered application scenarios are image recognition, speech recognition, and natural language processing. Notably, micro workloads, which are the basic operations to compose different networks (e.g. a convolutional operation), are also considered. As hardware, android devices, and a single board computer are evaluated. For more information on edge benchmarks, the reader is referred to Varghese *et al.* [20] as they present a comprehensive survey on edge performance benchmarking. The benchmarks are divided into two groups: explicit performance benchmarking, which is research to develop a benchmarking method, and implicit performance benchmarking, which compares different hardware, algorithms, or evaluates benchmarks without presenting a benchmarking method itself. 21 explicit and 99 implicit benchmarking applications are reviewed.

Altogether, a vast number of ML and edge benchmarks exists. To the best of the author's knowledge, there currently exists no ML benchmark specifically tailored to the requirements of the manufacturing domain. Furthermore, no benchmark focuses on data preprocessing tasks especially relevant for edge devices or performs algorithm evaluation on a standard PLC.

III. Algorithm Selection

In order to address **C1**, this section presents and analyses the properties of selected algorithms.

A. Selection of Basic Calculation Operations

Many ML algorithms have an underlying set of basic mathematical operations. The two operations **matrix multiplication** and **inversion** were chosen because of their significant relevance in various fields of ML and control theory. Applications are, solving linear equations, finite element methods, digital filtering, image processing, and generalized predictive control [21, 22]. This paper considers symmetric, dense matrix multiplication with equal size for both matrices. For matrix inversion, the Gauss Jordan algorithm is used as the most recommended approach [23].

Based on the use-case presented by Valencia-Palomo *et al.* [22], Model-Predictive Control (MPC) can make use of matrix multiplication and inversion. In their example, calculating the control law requires one matrix inversion and three matrix multiplications at a maximum dimension of 10x10. An estimation of the execution time for the MPC algorithm can be made if the computation times of basic mathematical operations are known (see subsection IV-A).

B. Introduction of selected Data Processing and ML Algorithms

This paper presents a selection of popular algorithms with different time and space complexity suitable for low power edge devices. Since legacy devices and low power MCUs should be supported as well, the use of Neural Networks was omitted. The algorithms belong to different application fields, like data preprocessing, ML, and filtering, addressing the heterogeneity in the field of data processing.

The **Grubbs Test** is an algorithm used for outlier detection. Grubbs Test identifies a single outlier at a time in normally distributed data. Use-cases for Grubbs Test in industry can be, for example, determining outliers in a machine temperature data stream [24] or preprocessing of signals on edge devices. A popular approach for signal denoising is the so-called low-pass **Butterworth filter**. In terms of applications, Zhang *et al.* [25] use a Butterworth filter to denoise signals originating from mining machinery. A general approach to identify noise in a data set is through clustering. Density-based clustering is a paradigm with the **Density-Based Scan Clustering** (DBSCAN) Algorithm as well-known representative. Considering applications, Jin *et al.* [26] propose a framework that makes use of DBSCAN clustering for outlier and defect detection of wafers. For supervised learning, **Random Forest** (RF) algorithms are a popular choice. As the time complexity of a RF algorithm depends on the average tree depth and the number of trees in the forest, this paper uses two separate algorithms to evaluate the respective behaviour. Considering RF applications, Kuppers *et al.* [27] make use of a RF to determine the operating state of a machine in less than 1 ms, proving the real-time suitability of RF approaches. Another approach in the domain of supervised learning are **Support Vector Machines** (SVMs). Since the computational complexity of an SVM heavily depends on the kernel function, this paper only considers the most basic linear version of a binary SVM. Rostami *et al.* [28] present an extensive review about data mining with SVMs for quality assessment in manufacturing. They review numerous publications with their respective use-cases. To summarise this section, table I provides an overview of the presented algorithms.

IV. Concept of Algorithm Evaluation

In order to assess the selected algorithms, suitable metrics have to be defined first. However, this task is challenging as the algorithm's requirements differ depending on the application scenario [20]. Therefore, this paper aims to provide a selection of generally applicable metrics for SM (**C2**).

A. Performance Metrics

Performance metrics are, for instance, suitable to compare the behaviour of selected algorithms on different hardware devices. Firstly, the **execution time** of an algorithm is an important performance measure. As manufacturing tasks usually have time constraints and often have to be performed in real-time, considering the execution time is crucial. Also, literature makes wide use of this approach (compare [29, 30]). Besides, considering execution times of basic operations create a foundation for algorithm execution time evaluation (see subsection III-A). However, measuring only one execution time is not sufficient since real-world execution times can heavily differ. Reasons are, for example, errors within the CPU, uncertainty effects in cache utilization, and effects of the underlying operating system. [31]. Thus, the algorithms are repeated a fixed number of times (e.g. 1000), obtaining the execution times' mean, median, and standard deviation.

Ignatov *et al.* [32] use a similar approach. Notably, the very first execution of the algorithm is recorded separately and not included in the mean calculation. The reason is that the first execution can heavily differ in its time behaviour since data is loaded into the CPU memory.

Besides execution time, **memory utilization** is another important performance metric. As edge devices usually have constraint memory, evaluating the algorithms' memory requirements is crucial. The memory requirements can be divided into pre-runtime memory requirements and runtime memory requirements [33]. Moreover, the data can be stored in different locations in the device, such as cache, RAM, or flash. Thus, evaluation of memory usage can be difficult.

B. Numerical Accuracy

Numerical accuracy heavily influences memory consumption, execution times, and energy consumption with a proportional relationship [34]. As some of the considered CPUs in this paper make use of a 32 or 64 Bit FPU, the algorithms are benchmarked for single (32 Bit)- and double (64 Bit)-precision values. With this approach, similar to the approach of Ramya Rani [35], the influence of the FPUs can be evaluated because 32 Bit FPUs cannot be used to calculate double-precision values efficiently [36]. For real application

TABLE I
OVERVIEW OF PRESENTED ALGORITHMS AND BASIC OPERATIONS (N: MATRIX DIMENSION, n: NUMBER OF DATA POINTS, k: NUMBER OF TREES, $D(n)$: AVERAGE TREE DEPTH, f: NUMBER OF FEATURES).

Algorithm	Domain	Applications	Time Complexity	Space Complexity
Standard Matrix Multiplication	Basis Operation	ML, MPC	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2)$
Gauss Jordan Matrix Inversion	Basis Operation	ML, MPC	$\mathcal{O}(N^3)$	$\mathcal{O}(N^2)$
Grubbs Test	Statistic	Outlier Detection	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Butterworth Filter	Signal processing	Preprocessing	$\mathcal{O}(n)$	$\mathcal{O}(n)$
DBSCAN	Clustering	Outlier Detection	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
Random Forest (Prediction)	Classification	Fault detection	$\mathcal{O}(k * D(n))$	$\mathcal{O}(k * \#nodes)$
SVM (Prediction)	Classification	Fault detection	$\mathcal{O}(f)$	$\mathcal{O}(f)$

TABLE II
OVERVIEW OF CONSIDERED HARDWARE (?: NO INFORMATION).

Category	Device	Clock Speed	RAM	L1 ICache/DCache [kB]
Microcontroller	Infineon PSOC63	150 MHz	288KB SRAM	8 (flash cache)
PLC	STW ESX.3cm	300 MHz	288 KB SRAM/8 MB SDRAM	16/16
PLC	Siemens S7-1500	?	1.5MB (program)/5MB (data)	?/?
Single-board PC	Beaglebone black	1 GHz	512MB DDR3	32/32
dedicated Edge device	Siemens IoT2050	1 GHz	2GB DDR4	32/32
IPC	Beckhoff CX9020	1 GHz	1 GB DDR3	32/32
IPC	Beckhoff CX2040	2.1 GHz	4 GB DDR3	32/48
Consumer PC	HP Spectre 13-v002ng	2.5 GHz	8GB DDR3	32/48

scenarios, the precision has to be chosen according to the requirements, such as accuracy, time, or energy constraints.

C. Energy Consumption

As edge devices or smart sensors are often directly deployed in the field and sometimes run on batteries, the energy consumption of the devices is a major concern [37]. However, state-of-the-art edge benchmarks mainly focus on quality metrics discarding energy consumption at all [20]. Thus, this paper examines the energy consumption of the selected algorithms on an exemplary basis. According to Li *et al.* [38], the energy consumption of an algorithm is influenced by its time and space complexity as well as the input scale. Furthermore, the number of memory accesses is a key factor [39]. Within this paper, two metrics inspired by Wang *et al.* [40] are proposed. First, energy consumption to execute one iteration of the algorithm (Energy per iteration) with a fixed number of data points. Second, energy consumption per data point indicates how much energy the algorithm consumes to process one data point.

D. Algorithm Implementation

Considering hardware, in order to address **C2** and **C4**, a total of eight devices out of six heterogeneous categories is considered (see table II). The devices represent standard equipment prevailing in the automation domain, including a legacy Siemens PLC. Due to the slow adoption of new technologies in this domain, novel edge accelerators are not considered. To implement the algorithms in a similar fashion on all devices, the code was programmed manually in C or IEC61131-3 Structured Text. For training the SVM and RF algorithm, the *scikit learn 0.24.2* framework [41] in the programming language Python has been used. The trained models were then exported to C-Code using the tool sklearn-porter [42]. As a compiler, gcc is always used for the devices programmed in C-Code. The compilation was done without the use of any optimization flags (default setting). Just the -lm flag was used to load needed math libraries. The devices STW and PSOC make use of adapted gcc compilers that are included in the provided toolchain. For the S7 PLC and the Beckhoff IPCs, the respective proprietary compiler was used. For an exemplary examination of memory usage, the algorithms were run on the beaglebone using the valgrind-3.17.0 [43] framework. The tool "Massif" included in valgrind is a heap profiler measuring a program's stack and heap memory usage during runtime. In order to calculate the mean execution time of the algorithm, we executed the code 1000 times (100 times on the PSOC) with the same inputs. All graphics considering execution time show the mean values respectively.

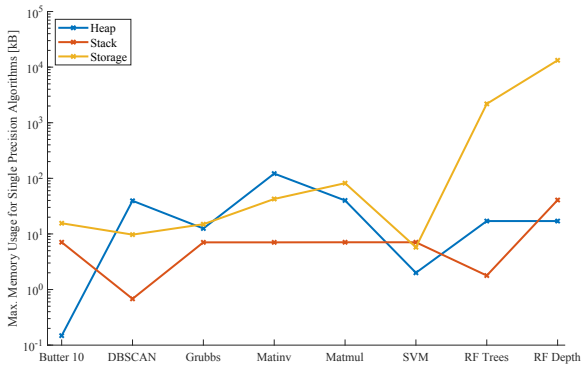
V. ALGORITHM ASSESSMENT WITH RESPECT TO SM METRICS

This section evaluates the algorithms against the previously introduced metrics and shows device specific specialities.

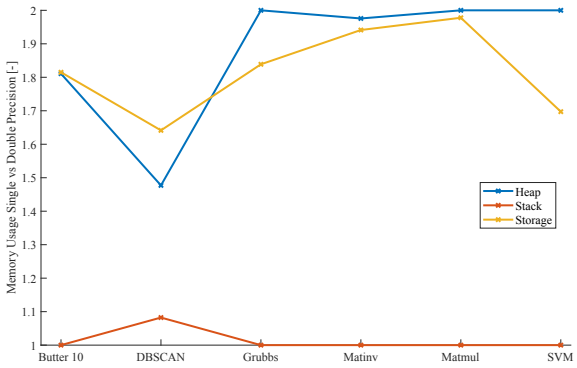
A. Memory Consumption of Algorithms

The memory usage of the selected single and double-precision algorithms was evaluated on the beaglebone black. As the results are all implementation-specific and obtaining those results is sometimes hardly possible, especially on microcontrollers with no operating system, the goal of this section is to provide a rough comparison of the algorithms on a single piece of hardware. The results are shown in figure 1.

Matrix Inversion consumes the largest amount of heap as it has to allocate the augmented matrix with size N*2N. Considering peak stack usage, values are rather similar. The RF algorithms need a lot more storage prior to algorithm execution than the others, as all the model information is stored in the C code. Thus, lack of memory is the major concern for RF prediction. Furthermore, it can be seen that all algorithms except DBSCAN need about two times as much



(a) Peak memory usage for single precision algorithms.



(b) Ratio of memory usage of double-precision vs. single-precision algorithms.

Fig. 1. Memory consumption on beaglebone black.

static storage for algorithm execution when using double-precision values. Also, heap storage doubles as more memory has to be allocated. Interestingly, peak stack usage stays about the same. The most probable reason is that in stack, only initialized, nonstatic variables are stored. Those variables are mainly used for the timer functionality that does not change between single and double-precision evaluations. For DBSCAN, the behaviour differs. It is likely possible that the different implementations of the mathematical functions (e.g., pow vs. powf) discussed in subsection V-C are responsible. In practice, memory overflow can be a real concern during algorithm implementation. For example, overflow of the L1-cache of a device can heavily influence deviations from first to subsequent algorithm executions (see subsection V-D).

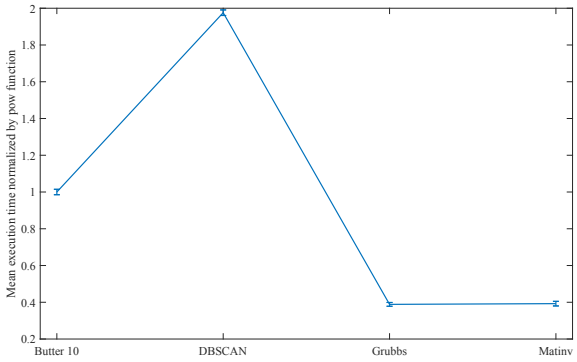
B. Influence of parallel Load on Algorithm Execution

Usually, the considered algorithms are not solely executed on the devices as additional communication, data processing, or higher priority control tasks have to run in parallel. Thus, this section evaluates the performance of the matrix multiplication algorithm when running a parallel artificial load task that runs cyclically every 20ms for 10ms. As hardware, the beaglebone, IoT2050, CX9020, and the S7 PLC were chosen to represent relevant edge devices and PLCs. For the beaglebone black and the S7 PLC, execution times double as the higher priority load task uses 50% of the CPU power. On the IoT2050, the algorithm execution time stays roughly the same as its CPU has four cores. Thus, the load task is executed in a parallel manner on one core of the CPU and the algorithm on another. For the CX9020, the deviation is comparably high but still within the standard deviation for very small matrix dimensions and execution times. For dimensions 10 to 30, executing an additional load task does not influence the algorithm's execution time to a great extent, because the load task is executed with a fixed pre-set cycle time of 20 ms but does only take 10 ms of computation time. Thus, the remaining 10 ms in each cycle can be used for the matrix multiplication task. Up to a matrix dimension of around 30, the computation time is shorter than 10 ms. However, as the time grows with $\mathcal{O}(N^3)$, the 10 ms period is exceeded for higher dimensions. Therefore, the matrix multiplication algorithm is interrupted by the next cycle of the higher priority load task. Thus, the execution time heavily rises about 2.5 times for higher matrix dimensions compared to running the algorithm alone, as overhead of the scheduling and operating system occurs as well.

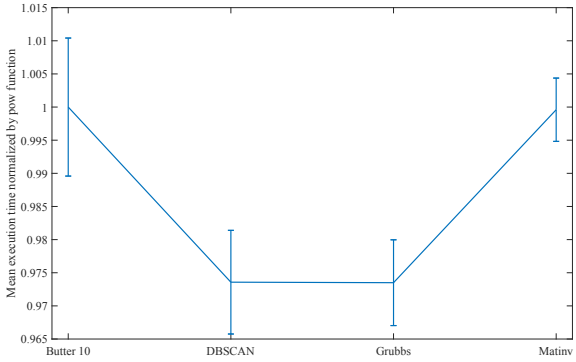
C. Single versus Double-Precision Accuracy

Consideration of numerical accuracy aims at finding a suitable trade-off between accuracy and computational resource consumption. In terms of execution time, calculating single instead of double-precision values can significantly reduce computational performance, for example on the PSOC MCU, as calculating double-precision values takes about three times longer. The reason is that the PSOC is equipped with a 32 Bit FPU, which double (64 Bit) values cannot take advantage of. This behaviour can be observed on all considered algorithms running on the PSOC and the STW PLC with a 32 Bit FPU. The S7 PLC, the HP spectre, and the CX2040 execute double-precision algorithms slightly slower than the single-precision ones. The most probable reason is the lack of an FPU paired with the longer time needed to load double values. The CX9020 behaves similar to the S7 PLC as its FPU can also handle double-precision values.

Surprisingly, the beaglebone and the IoT2050 show contrary behaviour for DBSCAN. For these devices, executing the single-precision version of DBSCAN takes a lot longer. The pow function used for calculating the euclidean distance has been identified as responsible for this behaviour. In the C math library, two versions of the pow function exist. Pow() uses double-precision values as in/outputs. Powf() takes single-precision values. Figure 2a shows the time behaviour of the pow compared to the powf function. As the euclidean distance only uses the second power, a simple double and float multiplication was also benchmarked. With default compiler optimization, calculating powf takes about twice as long as calculating pow. Also, simple multiplications are much faster. However, when using -O3 compiler optimization (see figure 2b), the compiler replaces pow and powf with simple multiplications. Thus, overall time decreases, and the



(a) Mean execution time and standard deviation with default optimization.



(b) Mean execution time and standard deviation with -O3 optimization.

Fig. 2. Evaluation of the execution time of pow(x,2) and powf(x,2) function on the beaglebone black.

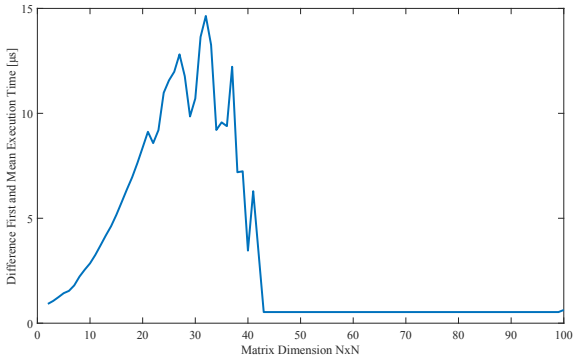


Fig. 3. Difference first and mean execution time on PSOC- single-precision.

difference between the math function and multiplication vanishes. Besides, performing float calculations is now slightly faster, what is the expected behaviour. A similar effect can be observed for the Grubbs Test algorithm, as it makes also use of the pow function. However, this effect cannot be observed on all hardware because the PSOC, the STW PLC, the CX9020, the CX2040, and the S7 PLC make use of vendor-specific functions. The S7 PLC and the CX9020 execute double-precision DBSCAN always about 2% slower than the single-precision version. Interestingly, the CX2040 executes double-precision DBSCAN about 2% faster than single-precision. The reason can be that the CX2040 natively supports a 64-Bit instruction set. Thus, it can be said that dependent on the used libraries and hardware, single-precision calculation is not always faster than double-precision.

D. PSOC- Deviation First and Mean Execution Time

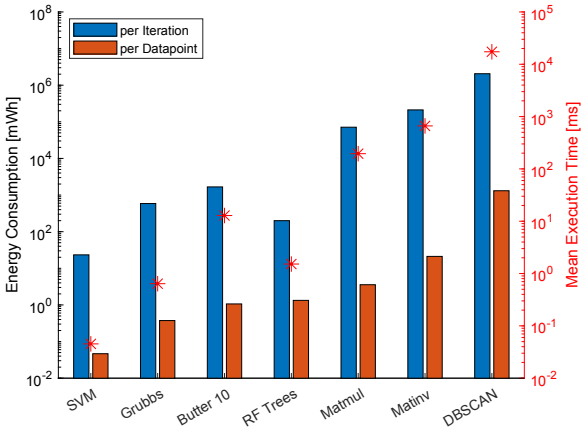
Taking a closer look at the difference between the very first execution time of the algorithm matrix multiplication and the subsequent mean execution times, it becomes clear that the PSOC device shows a characteristic behaviour (see figure 3). First, the difference grows and declines rapidly at a matrix dimension around 30, probably because of the 8 kB flash cash available in the MCU. For low matrix dimensions, the data is loaded in the cache during the first execution, increasing execution time. The subsequent executions can then be computed faster. For higher dimensions, the small cache of 8 kB cannot accommodate all relevant data. Thus, the data is loaded directly from flash, minimizing the difference between first and mean execution time. Altogether, developers should consider cache sizes to estimate the device's behavior.

E. Energy Consumption- Example

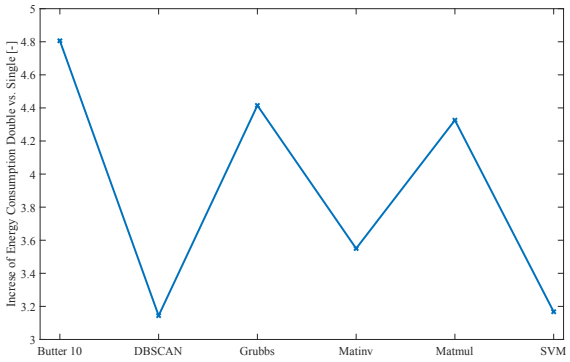
First, the idle consumption of the PSOC MCU is measured. If no program is running, the idle power consumption is 5.841mW. After algorithm execution the power consumption is 30.591 to 31.383mW. The reason for the difference is that after the algorithm is finished, the underlying benchmark program is still running to provide basic functionalities, like data output. Figure 4a presents the obtained energy consumption for the different algorithms. Furthermore, the algorithm's execution time is displayed. It can be observed that in both single and double-precision, the energy consumption is directly influenced by the runtime of the algorithms. The only exception is the RF algorithm which consumes more energy than the Butterworth algorithm despite a shorter execution time per iteration. The reason can be that a huge amount of data has to be loaded for the RF algorithm. Altogether, this paper can confirm the claim by Li *et al.* [38] that energy consumption directly depends on the time complexity for all algorithms except the RF algorithm. If a lot of data has to be loaded for the algorithm, the behaviour can differ. As depicted in figure 4b using double-precision needs about four times more energy than single-precision.

F. Application Example: Model Predictive Control

The results of the MPC use case introduced in subsection III-A are displayed in table III. The total execution times for the matrix operations heavily depends on the device. Assuming a typical time constraint of about 10 ms [44], all devices can meet those requirements. However, when considering the fact that additional tasks have to be executed on the device, the low power devices, like the S7 or STW PLC, could be too slow. In this case, a more powerful device would be a more suitable choice. Table III includes the



(a) Energy consumption in mWh for PSOC MCU and mean execution time- single-precision.



(b) Energy consumption increase double vs. single-precision for PSOC MCU.

Fig. 4. Overview of energy consumption of PSOC MCU for the selected algorithms.

TABLE III
MEAN EXECUTION TIMES OF 10X10 MATRIX MULTIPLICATION AND INVERSION- SINGLE-PRECISION (TIMES IN MS).

Hardware	Time Matrix Multiplication	Time Matrix Multiplication with Load Task	Time Matrix Inversion	Time Total Use-Case
PSOC	0.1285	-	0.7206	1.1061
BeagleBone	0.1074	0.2117	0.1363	0.4585
IoT2050	0.0546	0.0544	0.1966	0.3604
S7-1500	0.8260	1.7000	1.8661	4.3441
STW	0.1348	-	1.9709	2.3753
CX9020	0.3005	-	0.3497	1.2512
CX2040	0.0193	0.35258	0.0434	0.1013
HP Spectre	0.0034	-	0.0128	0.0230

time for matrix multiplication with additional load task on selected devices. In order to evaluate the total time of the use-case with load task, additional research has to be carried out to determined execution times of matrix inversion with parallel load. Furthermore, the real MPC execution time will be longer due to subsequent addition and subtractions.

VI. CONCLUSION AND OUTLOOK

This paper shows relevant aspects for algorithm benchmarking in the context of smart manufacturing. Several ML and preprocessing algorithms, namely Grubbs Test, Butterworth Filter, DBSCAN, Random Forest, and a Support Vector Machine, are presented. Furthermore, the basic mathematical operations Matrix Multiplication and Matrix Inversion are considered. Measurements could confirm the theoretical time complexity for all algorithms except the RF algorithm on all hardware devices. Contrary to the assumption, single-precision calculation is not always faster than double-precision if the algorithm makes use of the ”math.h” library. Besides, the availability of a FPU heavily influences performance. Moreover, data cache size can play a major role in algorithm execution, especially when executing the algorithm the first time. Besides, a strong influence on execution time could be measured by simultaneously executing a higher priority plant control task. Considering the energy consumption, a relation of execution time and consumed energy could be found. Moreover, this paper shows that using double-precision on an MCU heavily increases energy consumption. Finally, one use case shows how the results obtained in this paper facilitate a well-educated hardware selection for the considered algorithms.

In general, this paper can confirm the stated contributions. **C1** is confirmed by implementing state-of-the-art algorithms on legacy hardware, such as a S7-1500 PLC. Moreover, time- and space complexity analysis shows expected behaviour dependent on the data dimensions. As shown with the considered use-case, **C2** is fulfilled as well. Already considering hardware capabilities prior to algorithm execution can support an efficient algorithm implementation process and limit the number of trial-and-error approaches. Also, **C3** can be confirmed as the proposed metrics in section IV strongly support the algorithm assessment. Moreover, those metrics are tailored to the manufacturing domain by considering constraints, like energy consumption or execution time for evaluating real-time capabilities. Finally, **C4** is fulfilled by the revealed device specific specialities in section V. Altogether, this paper can guide selecting suitable algorithms and hardware to equip CPPS with novel data processing solutions thoughtfully. Moreover, the presented metrics can further support novel ML benchmarks for smart manufacturing.

Additional research should address the industrial applicability of the proposed findings’. Firstly, more ML algorithms can be selected. Popular choices are K-means clustering or Neutral Networks. Furthermore, the number of considered hardware devices can be extended to FPGAs, edge accelerators, and GPUs. Moreover, the considered metrics can be augmented dependent on the industrial use case. For example, network latency and bandwidth can play an important role. In addition, an upper bound worst-case execution time should be determined for the different algorithms, especially if the execution of the algorithm is considered critical. Finally, all presented findings and examinations could be integrated into a framework suitable for benchmarking edge devices in SM.

ACKNOWLEDGMENTS

This research is part of the project "Sensor-integrating Gear" (SIZA), funded by the German Research Foundation (DFG) under the project number 466653706. Furthermore, this work was supported by the Bavarian Ministry of Economic Affairs, Energy and Technology via the project AIValve (Grant No. DIK0116/01).

REFERENCES

[1] *Smart Maintenance für Smart Factories: Mit intelligenter Instandhaltung die Industrie 4.0 vorantreiben*, ser. acatech Position. München: Utz, Oktober 2015.

[2] C. A. Escobar, M. E. McGovern, and R. Morales-Menendez, "Quality 4.0: a review of big data challenges in manufacturing," *Journal of Intelligent Manufacturing*, 2021.

[3] I. Weiss, B. Vogel-Heuser, P. Holstein, and E. Trunzer, "Machine-Learning Models on the Edge to reduce Data Volume in Wide-Area Networks between various Production Sites," in *IECON 2020*, IEEE, 2020, pp. 3831–3835.

[4] B. Vogel-Heuser and L. Ribeiro, "Bringing Automated Intelligence to Cyber-Physical Production Systems in Factory Automation," in *CASE 2018*, B. Vogel-Heuser, Ed., IEEE, pp. 347–352.

[5] S. Trinks and C. Felden, "Edge Computing architecture to support Real Time Analytic applications," in *IEEE Big Data 2018*, IEEE, pp. 2930–2939.

[6] M. Ghahramani, Y. Qiao, M. Zhou, A. O Hagan, and J. Sweeney, "AI-based modeling and data-driven evaluation for smart manufacturing processes," *IEEE/CAA Journal of Automatica Sinica*, vol. 7, no. 4, pp. 1026–1037, 2020.

[7] J. Lee, J. Lee, and J. Jeong, "Design and Implementation of Injection Data Preprocessing & Monitoring System Based on Node-RED," in *ISMICT 2021*, IEEE, pp. 19–23.

[8] D. Yan, X. Ding, S. Pan, and H. Huang, "Tool Wear Prediction Based on Edge Data Processing and Deep Learning Model," *Journal of Physics: Conference Series*, vol. 1820, no. 1, 2021.

[9] M. Krüger, B. Vogel-Heuser, I. Weiss, and E. Trunzer, "Data-Driven Product Quality Monitoring in Quality-Critical Forming Processes," in *CESCIT 2021*, 2021.

[10] S. Fahle, C. Prinz, and B. Kuhlenkötter, "Systematic review on machine learning (ML) methods for manufacturing processes – Identifying artificial intelligence (AI) methods for field application," *Procedia CIRP*, vol. 93, pp. 413–418, 2020.

[11] W. Zhang, D. Yang, and H. Wang, "Data-Driven Methods for Predictive Maintenance of Industrial Equipment: A Survey," *IEEE Systems Journal*, vol. 13, no. 3, pp. 2213–2227, 2019.

[12] A. Angelopoulos *et al.*, "Tackling Faults in the Industry 4.0 Era-A Survey of Machine-Learning Solutions and Key Aspects," *Sensors*, vol. 20, no. 1, 2019.

[13] A. Dogan and D. Birant, "Machine learning and data mining in manufacturing," *Expert Systems with Applications*, vol. 166, 2021.

[14] V. J. Reddi *et al.*, "MLPerf Inference Benchmark," in *ISCA 2020*, pp. 446–459.

[15] Y.-H. Chang, J. Pu, W.-m. Hwu, and J. Xiong, "MLHarness: A scalable benchmarking system for MLCommons," *BenchCouncil Transactions on Benchmarks, Standards and Evaluations*, vol. 1, no. 1, 2021.

[16] C. Coleman *et al.*, *DAWNBench: An End-to-End Deep Learning Benchmark and Competition*, 2018.

[17] Baidu Research. (). DeepBench, [Online]. Available: https://github.com/baidu-research/DeepBench.

[18] T. Hao *et al.*, "Edge AIBench: Towards Comprehensive End-to-End Edge Computing Benchmarking," in *Benchmarking, Measuring, and Optimizing*, ser. LNCS, vol. 11459, Springer International Publishing, 2019, pp. 23–30.

[19] C. Luo *et al.*, "AIoT Bench: Towards Comprehensive Benchmarking Mobile and Embedded Device Intelligence," in *Benchmarking, Measuring, and Optimizing*, ser. LNCS, vol. 11459, Springer International Publishing, 2019, pp. 31–35.

[20] B. Varghese *et al.* (2020). A Survey on Edge Performance Benchmarking, [Online]. Available: https://arxiv.org/pdf/2004.11725.

[21] G. Sharma, A. Agarwala, and B. Bhattacharya, "A fast parallel Gauss Jordan algorithm for matrix inversion using CUDA," *Computers & Structures*, vol. 128, pp. 31–37, 2013.

[22] G. Valencia-Palomo, K. R. Hilton, and J. A. Rossiter, "Predictive Control implementation in a PLC using the IEC 1131.3 programming standard," in *ECC 2009*, IEEE, 2009, pp. 1317–1322.

[23] S Chetan, K. Sourabh, V Lekshmi, S Sudhakar, and J Manikandan, "Design and Evaluation of Floating point Matrix Operations for FPGA based system design," *Procedia Computer Science*, vol. 171, pp. 959–968, 2020.

[24] C. M. Ryan, A. Parnell, and C. Mahoney. (). Real-Time Anomaly Detection for Advanced Manufacturing: Improving on Twitter's State of the Art, [Online]. Available: https://arxiv.org/pdf/1911.05376.

[25] Z. Zhang, J. Hui, S. Gao, Z. Shi, X. Zhang, and H. Fan, "Research on threshold denoising method of mining machinery," in *ITNEC 2020*, IEEE, 2020, pp. 1236–1240.

[26] C. H. Jin, H. J. Na, M. Piao, G. Pok, and K. H. Ryu, "A Novel DBSCAN-Based Defect Pattern Detection and Classification Framework for Wafer Bin Map," *IEEE Transactions on Semiconductor Manufacturing*, vol. 32, no. 3, pp. 286–292, 2019.

[27] F. Kuppers, J. Albers, and A. Haselhoff, "Random Forest on an Embedded Device for Real-time Machine State Classification," in *EUSIPCO 2019*, IEEE, 2019, pp. 1–5.

[28] H. Rostami, J.-Y. Dantan, and L. Homri, "Review of data mining applications for quality assessment in manufacturing industry: support vector machines," *International Journal of Metrology and Quality Engineering*, vol. 6, no. 4, p. 401, 2015.

[29] A. Ivanov, Z. Natalia, and A. Veronika, "CPU vs GPU Performance of MATLAB Clustering Algorithms," in *Distributed Computer and Communication Networks*, ser. Communications in Computer and Information Science, vol. 1337, Springer Nature, 2021, pp. 43–56.

[30] B. Vogel-Heuser, E. Trunzer, D. Hujo, and M. Sollfrank, "(Re)deployment of Smart Algorithms in Cyber–Physical Production Systems Using DSL4hDNCS," *Proceedings of the IEEE*, vol. 109, no. 4, pp. 542–555, 2021.

[31] F. J. Cazorla, L. Kosmidis, E. Mezzetti, C. Hernandez, J. Abella, and T. Vardanega, "Probabilistic Worst-Case Timing Analysis," *ACM Computing Surveys*, vol. 52, no. 1, pp. 1–35, 2019.

[32] A. Ignatov *et al.*, "AI Benchmark: All About Deep Learning on Smartphones in 2019," in *ICCVW 2019*, IEEE, pp. 3617–3635.

[33] S. Bianco, R. Cadene, L. Celona, and P. Napoletano, "Benchmark Analysis of Representative Deep Neural Network Architectures," *IEEE Access*, vol. 6, 2018.

[34] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[35] N. Ramya Rani, "Implementation of Embedded Floating Point Arithmetic Units on FPGA," *Applied Mechanics and Materials*, vol. 550, pp. 126–136, 2014.

[36] S. Gopinath, N. Ghanathe, V. Seshadri, and R. Sharma, "Compiling KB-sized machine learning models to tiny IoT devices," in *PLDI 2019*, ser. ACM, pp. 79–95.

[37] J. Mocnej, M. Miškuf, P. Papcun, and I. Zolotová, "Impact of Edge Computing Paradigm on Energy Consumption in IoT," *IFAC*, vol. 51, no. 6, pp. 162–167, 2018.

[38] Q. Li, B. Guo, Y. Shen, J. Wang, Y. Wu, and Y. Liu, "An Embedded Software Power Model Based on Algorithm Complexity Using Back-Propagation Neural Networks," in *GreenCom-CPSCom 2010*, IEEE, 2010, pp. 454–459.

[39] H. Wang, "Research on real-time reliability evaluation of CPS system based on machine learning," *Computer Communications*, vol. 157, pp. 336–342, 2020.

[40] Y. Wang *et al.*, "Benchmarking the Performance and Energy Efficiency of AI Accelerators for AI Training," in *CCGRID 2020*, IEEE, 2020, pp. 744–751.

[41] (). Sklearn.datasets.make_classification — scikit-learn 0.24.2 documentation, [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html.

[42] Darius Morawiec. (). Sklearn-porter, [Online]. Available: https://github.com/nok/sklearn-porter.

[43] (). Valgrind, [Online]. Available: https://valgrind.org/.

[44] B. Vogel-Heuser, "Automation in the Wood and Paper Industry," in *Springer Handbook of Automation*, S. Y. Nof, Ed., Springer Berlin Heidelberg, 2009, pp. 1015–1026.