Tiny Anomaly Detection with On-device Training on LoRaWAN Enabled Microcontrollers

Abstract

Never before has machine learning been characterised by such innovative waves of technology. At the same time, the rise of low-budget single-board microcontrollers allows service providers to meet privacy, low latency and energy efficiency requirements by deploying artificial intelligence on the edge. Increasing computing capabilities of these tiny smart devices makes it possible to deploy machine learning models not only for inference but also for training. In this study, we focus on implementation and evaluation of unsupervised machine learning models which can be deployed and trained on tiny low-power long-range network enabled devices for anomaly detection. First, we study various traditional and neural network based machine learning algorithms which can be employed under low computing and memory resource constraints. Next, we evaluate these algorithms in terms of accuracy and search for their optimal hyperparameter configurations. Finally, several of the most accurate machine learning algorithms found are implemented on multiple modern microcontrollers and evaluated in terms of training latency, inference rate and power consumption.

Keywords:
TinyML, deep learning, anomaly detection, on-device training, LoRaWAN

1. Introduction

Healthcare is one of the most extensive factors that need to be efficiently managed for the development of any country, as it is Artificial intelligence (AI) and machine learning (ML) are revolutionising almost every industry with a seemingly endless list of applications ranging from object recognition in autonomous vehicles [1] to helping doctors detect and diagnose diseases [2]. At the same time, the recent progress in development of low-budget sensors and single-board computers has enabled deploying AI on the edge which allows service providers to increase availability of their ML-driven services and applications as well as deliver lower latency, more secure data privacy and better energy efficiency. Furthermore, increasing computing and connectivity capabilities of modern microcontrollers has made it possible to deploy tiny machine learning (tinyML) models not only for inference but also for training which opens new possibilities as well as poses new challenges when deploying AI/ML applications on the edge [3, 4].

Modern microcontrollers are often equipped with integrated low-power radio transceivers for wireless connectivity. As a rule, smart devices used in an application transmit the data collected on the edge to a gateway where this data is aggregated and sent via a broadband network connection to a cloud platform for storage and/or further processing [5]. There are several wireless connectivity protocols that often used in such tinyML applications, the most popular of which include unlicensed WiFi and low-power wide-area-networking (LPWAN) via LoRa or Sigfox, as well as licensed cellular radio such as LTE for machines (LTE-M) and narrow band (NB) internet-of-things (IoT). In our research, we rely on microcontrollers that utilise LPWAN over LoRa. Such microcontrollers are able to communicate without an interruption at a distance of 15-30 kilometres in flat rural areas [6] and 2-5 kilometres in urban environments [7]. LoRaWAN does not require big amounts of energy, therefore the batteries of the sensors and single-board computers that employ this communication technology can last for years before they run out [8]. Furthermore, LoRa networks are easy to deploy and use due to its relatively simple architecture. Finally, LoRaWAN can run on unlicensed radio networks across the globe without need to pay an operator for using the network which reduces the costs of the deployment. The LoRaWAN features listed above make it an ideal solution for applications that do not require large data transmissions with low latency such as smart farming [9, 10, 11, 12], indoor localisation [13, 14] and energy consumption monitoring and optimisation [15].

Speaking of machine learning methods, they are usually divided into the following three main categories: supervised, unsupervised and reinforcement learning. The methods in the first category, i.e. supervised learning, require large quantities of human-labelled data available for learning functional mappings between the training samples observed and the desired outputs. Unsupervised learning algorithms are designed to infer the underlying structure of the data under consideration without depending on external resources such as human supervision. Finally, reinforcement learning (RL) algorithms search for optimal actions in uncertain time-varying environments with the help of intelligently shaped reward signals. Another category of AI/ML algorithms which is worth mentioning is the methods that follow a deep learning approach. This approach relies on multi-layer neural networks: the first layers as a rule are used to find compact low-dimensional representations of high-dimensional data whereas later layers are responsible for achievement of the task given. Deep learning models can be employed in supervised, unsupervised and reinforcement learning algorithms. Many tinyML researchers recognize the power and importance of deep learning and explore its potential to be deployed even on microcontrollers with limited computing and memory resources [16, 17].

In our research, we focus on unsupervised machine learning models which can be deployed on tiny microcontrollers for anomaly detection. As a rule an anomaly detection model is trained in an unsupervised way using the features of event patterns which represent a certain normal behaviour and then it is used to identify outliers, i.e. the patterns that deviate significantly from the norms established. Such an approach can be effectively used to detect never seen before anomalous patterns that may be indicative of a fault in a system [18] or even a deliberate attack [19]. The approach employed in the majority of the studies devoted to tinyML for anomaly detection involves training the normal behaviour model offline on a powerful device using a large dataset and then deploying the model trained on a microcontroller for inference only [20]. However, in certain applications it is sometimes difficult or even impossible to collect such a dataset in advance and therefore the model of normal behaviour in such cases is supposed to be trained directly on-device [21]. This poses additional constraints on the AI/ML model to be deployed as not only should it fit into small memory of the target microcontroller while being accurate enough at the inference time, but it is also supposed to have low latency and power consumption during the training stage [22].

In this study in particular, we concentrate on anomaly detection algorithms that can be trained on LoRaWAN enabled tiny single-board computers. Our contribution to the field of study is threefold. First, we implement and evaluate multiple anomaly detection algorithms in terms of accuracy, false positive and true positive rates under the constraints posed by limited computing and memory resources available at microcontrollers. Second, we compare the most accurate algorithms in terms of data processing rate and power consumption. Third, we use the best of the algorithms tested to implement a prototype of the device that can be deployed on the edge to detect anomalous vibrations and report the detection results over LoRaWAN. The rest of the document is organised as follows. Section 2 surveys the anomaly detection algorithms evaluated as well as briefly describes the approach used for hyperparameter optimisation in our experiments. Numerical evaluation results are presented in Section 3. Section 4 covers the most important details of the working prototype implementation using one of the AI/ML models tested. Section 5 concludes the study and outlines future work.

2. Theoretical background

In this section, we first summarise multiple anomaly detection algorithms that can be deployed on tiny microcontrollers for on-device training. After that, we briefly describe how one can conduct a search for optimal hyperparameters for the anomaly detection methods selected.

2.1. Anomaly detection

The anomaly detection methods we implement and test in this study can be divided into three categories: the ones based on stream clustering, traditional neural networks and deep learning. The first category of methods classifies a sample as an outlier by measuring its distance from the clusters obtained for the normal data. These clusters can be found using one of the stream clustering techniques such as scalable k-means++ (SKM) [23], CluStream (CS) [24], weighted affinity propagation (WAP) [25], weighted c-means (WCM) [26] and many others [27].

ScalableKMeans++ is a lightweight version of popular k-means clustering algorithm: it first samples several centroid candidates from the data batch based on the distance from the sample to the current cluster centroids; after that, new cluster centroids are selected by using weighted k-means where the weight for each sample is calculated as the sum of weights of the samples which are the closest to the corresponding candidates.

CluStream is the algorithm which relies on creating and updating micro-clusters: each new sample in the data stream is either assigned to the closest micro-cluster if it does not increase the average distortion of this micro-cluster significantly, or a new micro-cluster is created using this new sample as its centroid; in the latter case if the number of micro-clusters exceeds the predefined threshold, two closest micro-clusters merge together; at the end of the clustering process, centroids of the resulting micro-clusters are clustered using the same weighted k-means.

Weighted affinity propagation is a weighted version of affinity propagation algorithm which uses similarity, availability and responsibility matrices to pass messages between data points and find the most optimal exemplars; the weighted version of the algorithm operates in batches and takes into account the weights and distortions calculated for exemplars obtained at the previous iteration.

Weighted c-means is a version of soft clustering algorithm c-means which similarly calculates the cluster centroids using the membership matrix each element of which is equal to the probability of the sample belonging to one of the current clusters; the weight of each centroid is updated with every new data batch using the membership matrix calculated for the current cluster centroids and new samples stacked together.

The next category of algorithms is based on traditional neural networks trained in an unsupervised way. Two of the most famous algorithms in this category are self-organising maps (SOM) [28] and growing neural gas (GNG) [29]. These algorithms suit well for the task of anomaly detection on microcontrollers by design: they compress the data into a set of representative units in an incremental manner by taking small subsets of data at a time. Such a set of representative units found for a set of normal samples can further act as the model of normal behaviour to detect anomalies in the inference stage.

Self-organising map is a competitive learning algorithm that allows one to compress high-dimensional data and create a network that stores information in such a way that any topological relationships within the dataset are maintained; for this purpose, a grid of neurons, each of which is fully-connected to the input layer is formed and updated during the training.

Growing neural gas is also a topology learning algorithm that models a data space through interconnected units that stand on the most populated areas of that space; it partitions data into independent networks which reflect the number of clusters in the dataset.

The last but not the least category of methods is based on the deep learning approach. The deep learning based methods for anomaly detection tested in this study are several variations of autoencoder including undercomplete autoencoder (UAE), sparse autoencoder (SAE), denoising autoencoder (DAE) and variational autoencoder (VAE) as well as deep self-organising map [30] and deep support vector data description (DSVDD) [31].

Autoencoder aims to adjust its parameters in such a way that the output layer is equal to the input one despite the information bottleneck caused by the hidden layers; the role of the loss function is often played by the reconstruction error which is the difference between the input and the output; anomalous data point fed to the autoencoder model results in something that is quite different from the expected output, and therefore a large error value. Autoencoder variations include but not limited to undercomplete autoencoder whose hidden layer size is much smaller than input and output ones, sparse auto-encoder which includes in its loss function not only the reconstruction error but also a special term which grows with the number of units which fire in the bottleneck layer, denoising autoencoder for which Gaussian noise is added to the inputs during the training, and variational autoencoder, the encoder of which returns a distribution over the latent space instead of a single point.

Deep self-organising map consists of an autoencoder connected to a self-organising map; the loss function is the weighted sum of the autoencoder' reconstruction error and the distance to the map's best matching unit.

Deep support vector data description learns a neural network transformation from an input space to an output space that attempts to map most of the data network representations into a hypersphere of minimum volume; mappings of normal examples fall within, whereas mappings of anomalies fall outside the hypersphere.

2.2. Automatic machine learning

In order to find optimal hyperparameters for the anomaly detection algorithms selected, automated machine learning (AutoML) approach can be employed. AutoML is usually defined as automating the process of building and optimising AI/ML models without human assistance [32]. AutoML can automate the process of training and deploying machine learning models and therefore advance AI/ML research by saving time and resources required for finding optimal model parameters. Furthermore, it makes state-of-the-art ML models as well as other recent developments in

AI accessible to non-experts which also accelerates adoption of AI/ML applications in both academia and industry. As an immediate consequence of the growing demand for AutoML, multiple open-source AutoML frameworks that are easy to use even for researchers not familiar with AI/ML concepts have recently been developed [33]. Some such frameworks focus specifically on improving certain models, e.g. Auto-Keras [34] searches for optimal neural network architectures developed in Tensorflow [35] whereas Auto-Sklearn [36] is designed to work with the models implemented with Scikit-learn Python package [37]. Other frameworks such as Ray [38], Flaml [39] and several others allow their users to implement and optimise virtually any custom AI/ML model needed.

Most of the AutoML frameworks focus on hyperparameter optimization, namely they aim to minimise the AI/ML model's loss function calculated for a validation subset of the target dataset [32]. The methods used for this purpose may vary from straightforward random and grid search to more advanced techniques such as Bayesian optimization [40]. The latter allows to reduce computing costs of finding an optimal hyperparameter configuration compared to the straightforward approaches which involve assessing each individual configuration separately. The Bayesian optimization approach relies on using a Gaussian process as a surrogate to estimate the expected value and variance of the loss function given the hyperparameter configuration. A Gaussian process is a probability distribution over random variable functions such that every finite subset of them follows a multivariate normal distribution. Prior knowledge about the likelihood of each such function is described by a prior mean function and a positive-definite kernel. Given a set of observations, namely the hyperparameter combinations tested and the resulting loss values, the space of possible functions can be reduced via updating the means and variances of the random variables via Gaussian process regression. Choice of the kernel function may have a significant impact on the correctness of the regression and therefore it has to be selected properly based on a prior assumption of the target function [41]. Once the function space has been reduced, using the conditional probability rule, one can estimate the posterior distribution of the function for a new hyperparameter configuration. Bayesian optimization uses an acquisition function to find the best hyperparameter configuration for the resulting surrogate Gaussian process function. One popular choice for such an acquisition function is expected improvement over the best previously observed objective value [42]. While the Gaussian process has become a standard surrogate for modelling an objective function in Bayesian optimization, an ensemble of regression trees [43] or even a neural network [44] can also be used for this purpose.

3. Numerical results

In this section, we present the results of the experiments conducted. First, we overview the datasets used to train anomaly detection models in our experiments and single-board computers on which the models trained are deployed. Next, we compare the algorithms trained in terms of accuracy, true positive and false positive rates and carry out a search for optimal hyperparameters for the most accurate methods tested. Finally, we evaluate the algorithms selected in terms of data processing rate and power consumption.

3.1. Datasets and devices

First, we test the anomaly detection algorithms overviewed in the previous section using two publicly available benchmark datasets: Outlier Detection DataSets (ODDS) [45] and Skoltech Anomaly Benchmark (SKAB) [46]. We have picked 20 datasets from each benchmark and limited the maximum number of samples extracted from each dataset by 20000. If the number of samples exceeds this threshold, 20000 samples are randomly selected from the samples available. It is also worth mentioning that in ODDS benchmark datasets there is one record per data point, whereas in SKAB datasets, samples are essentially multivariate time series collected with multiple sensors over some time period. To extract features from these time series, we use a window of length 32 that moves with step 1. For the sequence of vectors in each such time window, we calculate its average and use it as the feature vector. All feature vectors are transformed into range [0, 1] using min-max standardisation.

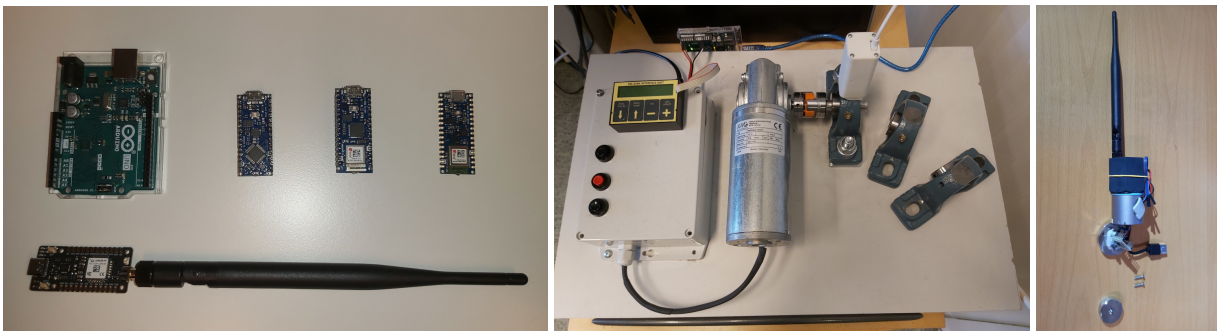
In addition to that, we have built a testing environment to evaluate the algorithms mentioned in the real use case scenario. The environment includes an electric motor and several bearings: one is normal whereas others have various defects. The speed of the motor is controlled by the code running on a microcontroller which randomly samples the speed values from a certain interval. Another single-board computer with an inertial measurement unit (IMU) is installed on top of the bearing in order to classify vibrations recorded as either produced by the normal or one of the defective bearings (see Figure 1b). We have collected several minutes of the IMU data for each of the bearings used.

Board	CPU	RAM	Flash
Arduino Uno Rev3	AVR, 16 MHz	2 KB	32 KB
Arduino Nano Every	megaAVR, 16 MHz	6 KB	48 KB
Arduino Nano 33 IoT	ARM Cortex-M0+, 48 MHz	32 KB	256 KB
Arduino Nano 33 BLE Sense	ARM Cortex-M4, 64 MHz	256 KB	1024 KB
LoRa-E5 mini	ARM Cortex-M4, 48 MHz	64 KB	256 KB

Table 1: Development boards used for anomaly detection algorithm evaluation in the study.

We split the resulting time series using the moving window approach and then apply fast Fourier transform (FFT) to extract necessary features. As previously, min-max standardisation is employed to scale the feature vectors into the [0, 1] range.

Speaking of the devices used, we first test the code using several Arduino boards with various microcontrollers. These include Uno Rev3, Nano Every, Nano 33 IoT and Nano 33 BLE Sense. In addition, we use the LoRa-E5 mini development board equipped with the LoRaWAN connectivity module. The selected board specifications can be found in Table 1. Since we focus on training anomaly detection algorithms on-device, RAM volume becomes the most important characteristic that defines the size of the AI/ML model that can be deployed for learning normal behaviour patterns. The data processing rates are mostly defined by the CPU type.



(a) Boards used in the study. (b) Bearing test environment. (c) Prototype.

Figure 1: Materials used in the research. Fig. (a) shows single-board computers on which AI/ML models have been deployed: (top, from left to right) Arduino Uno Rev 3, Arduino Nano Every, Arduino Nano 33 IoT, Arduino Nano 33 BLE Sense, and LoRa-E5 mini (bottom). The bearing test environment used for anomalous vibration detection is shown in Fig. (b). Fig. (c) shows the anomalous vibration detection device prototype implemented using the LoRa-E5 mini development board.

3.2. Detection accuracy evaluation

To compare the anomaly detection algorithms summarised in Section 2 in terms of accuracy under various board memory constraints, we implement those algorithms in Python on a PC. To calculate the value of RAM used we simply count all the variables used in our implementation of the algorithm under consideration and multiply it by 2 bytes for integers and 4 bytes - for floating point numbers. We test the algorithms implemented with several arbitrarily chosen combinations of hyperparameters and select the best combination for the comparison. The hyperparameters that affect the model size the most are the number of micro-clusters and batch/reservoir size in the case of stream clustering based algorithms and the number of layers and the number of neurons in the case of neural network based ones. If for a certain hyperparameter combination the model does not fit the memory constraints, we adjust the combination slightly, taking instead the maximum possible values found by decreasing the hyperparameter values proportionally until they satisfy the memory constraints. This may lead to the situation when the model implemented for a smaller RAM size sometimes allows one to obtain slightly better accuracy compared to the one calculated for bigger memory values.

Algorithm	Board				
	Arduino Uno Rev3	Arduino Nano Every	Arduino Nano 33 IoT	Arduino Nano 33 BLE Sense	LoRa-E5 mini
SKM	80.3738	80.9814	80.9536	80.9536	80.9536
CS	**81.8369**	**82.1188**	82.0130	82.0130	82.0130
WAP	80.4589	81.9943	**82.3531**	**82.3531**	**82.3531**
WCM	80.9659	81.5710	81.7194	81.7194	81.7194
SOM	79.0377	79.4520	79.4736	79.4736	79.3984
GNG	78.4472	78.0739	77.9621	77.9621	77.9621
UAE	61.2263	61.2758	61.3074	60.0393	59.9435
SAE	61.9630	62.0459	61.8365	61.2463	61.6242
DAE	62.2700	61.9851	61.7839	61.1683	61.1061
VAE	62.8957	60.4938	62.5878	60.6721	59.3226
DSOM	76.4634	76.5790	77.2092	76.0702	76.8779
DSVDD	79.2322	80.9131	82.0816	81.4344	81.7941

Table 2: Average anomaly detection accuracy (%) calculated for various anomaly detection algorithms implemented under several board memory constraints using ODDS and SKAB benchmark datasets. The higher the value the better.

For the accuracy evaluation, we split the datasets into training-plus-validation (around 70%) and inference (around 30%) parts and use only normal samples from the former to train the algorithms. We adjust each inference subset in such a way that it contains an equal number of normal and abnormal samples which allows us to compare the algorithms in terms of accuracy. For each board and each algorithm, three experiments are carried out with different random seeds and the average accuracy value is calculated. The results obtained for each benchmark dataset separately can be found in Tables A.7–A.11 of Appendix Appendix A. The average accuracy values are listed in Table 2.

As one can see, all the algorithms based on stream clustering provide the best results for anomaly detection in each of the benchmark datasets used: CluStream is the most accurate when the memory size is low whereas weighted affinity propagation outperforms analogues for bigger RAM values. Speaking of neural network based algorithms, all the auto-encoder based models tested have resulted in quite low accuracy values compared to other methods. Interestingly enough, even results of the deep self-organising map model in which auto-encoder is also used are worse compared to the standard SOM model. The only deep learning model that shows promising results is deep support vector data description. Thus, based on the accuracy values obtained, we select the following six algorithms for further evaluation: CS, SKM, WAP, WCM, SOM and DSVDD.

Next, we evaluate the algorithms selected using the data generated in the bearing test environment. The dependence of true positive rate (TPR) on false positive rate (FPR) under LoRaWAN enabled LoRa-E5 mini memory constraints can be found in Figure 2a. It is worth saying that we are interested the most in the methods that provide for higher detection rates in case of low FPR values. For this reason, the figure shows the dependence of TPR on FPR for the cases when the latter is below 5%. In this context, anomaly detection methods based on stream clustering algorithms, scalable k-means++ and CluStream as well as deep learning based support vector data description show the most promising results as one can notice from the figure. Dependence of TPR on FPR for memory constraints of Arduino boards used can be found in Figures B.5a–B.8a of Appendix Appendix B. As one can notice when looking at the figures, DSVDD becomes less efficient when the RAM size decreases, as this deep learning model requires more trainable parameters compared to straightforward clustering techniques. Based on the results obtained we select SKM, CS and DSVDD for further evaluation.

3.3. Hyperparameter optimization

In order to improve the anomaly detection accuracy values achieved by the algorithms selected we use two lightweight open-source AutoML frameworks Ray and Flaml. The former uses Bayesian optimization to find an optimal hyperparameter combination whereas the latter relies on some custom optimization algorithm called blended search, the description of which in detail can be found in study [47]. We conduct the search for the same memory

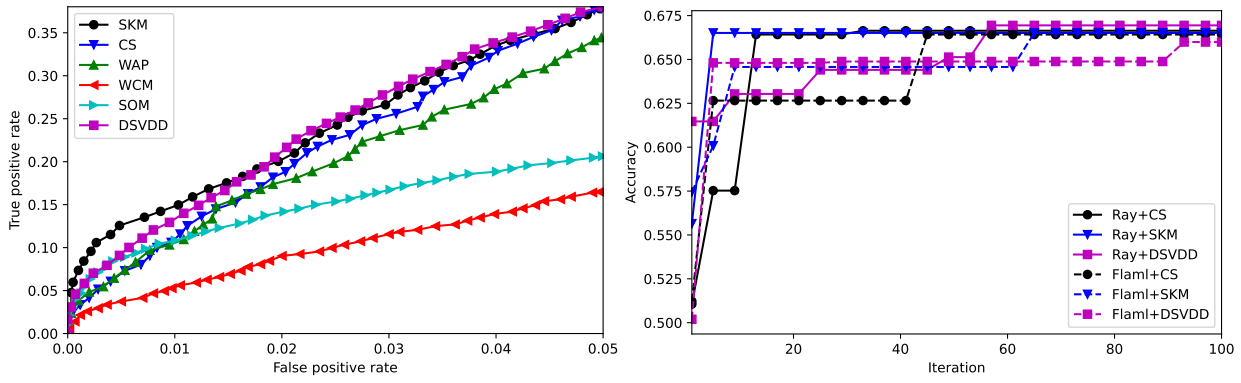
constraints and the same bearing test data as previously. For each algorithm and each RAM value, the search is carried out for 100 iterations. In the case of scalable k-means++, the hyperparamters are the number of clusters and the batch size whereas in the case of CluStream those are the number of clusters and the number of microclusters. For DSVDD, we search for an optimal two-layer fully-connected network architecture. Figures 2b and B.5b–B.8b show the improvement of the anomaly detection accuracy over the search iterations for LoRa-E5 and Arduino boards respectively. The final accuracy values can be found in Table 3.

As one can notice from the table and the figures, all three anomaly detection algorithms tested allow one to reach similar accuracy values equal to 66.6–66.9%. Furthermore, employing CluStream shows the most promising results for first three lower RAM values whereas DSVDD slightly outperforms other algorithms in case of the boards with bigger memory capacity. Speaking of the hyperparameters found, in the case of scalable k-means++, both Ray and Flaml lead to the same result for all five boards tested: 3 clusters with 4 samples per batch. CluStream hyperparameter combinations obtained with the AutoML frameworks employed are slightly different between each other, but they remain the same for each of the boards used and they are respectively 3 clusters, 4 microclusters and 4 clusters, 5 microclusters. For some reason, Flaml with its blended search algorithm fails to find the CluStream solution with 3 clusters which is slightly more accurate as one can notice from Table 3. In the case of DSVDD, the number of neurons per layer differs depending on the RAM volume available. As one can notice, the DSVDD model found for LoRa-E5 slightly outperforms the one obtained for Arduino Nano 33 BLE Sense even though the latter has four times bigger RAM capacity. This can be explained by the way we search for optimal network architecture: for each board, we limit the number of neurons per layer based on memory available on the board. Since, Arduino Nano 33 BLE Sense has bigger RAM value, the range of possible combinations is much bigger, therefore, it is more challenging for the AutoML frameworks used to find an optimal hyperparameter combination for this board given the limited number of search iterations. Thus, we later implement the network architecture found for LoRa-E5 on the Arduino board.

It is also worth mentioning that we have tried to increase the accuracy for SOM and WCM using the AutoML approach, but have not managed to improve it significantly. Speaking of WAP, the number of clusters in this algorithm is not defined explicitly, but instead it depends on the preference parameter and the data stream itself. If this parameter is not selected properly for the data stream analysed, there is a probability that the model becomes too large to fit the memory during the training. It is therefore very difficult to implement this anomaly detection algorithm on a tiny microcontroller without knowing the data in advance. For this reason, we have decided to leave this algorithm for future work when we find a simple way to limit the number of clusters found during the training.

3.4. Data processing rate and power consumption assessment

To evaluate each algorithm's data processing rate during both the training and the inference stage, we implement them on each of the boards mentioned. The hyperparameters used for the implementations are the ones that provide



(a) Dependence of TPR on FPR.

(b) Accuracy improvement with AutoML.

Figure 2: Anomaly detection algorithm evaluation under LoRa-E5 mini memory constraints using the data generated in the bearing test environment. Figure 2a shows dependence of true positive rate on false positive rate for various anomaly detection algorithms for low FPR values (below 5%). Results of the experiments on optimal hyperparameter search, namely improvement of the anomaly detection accuracy, using two AutoML frameworks is shown in Figure 2b.

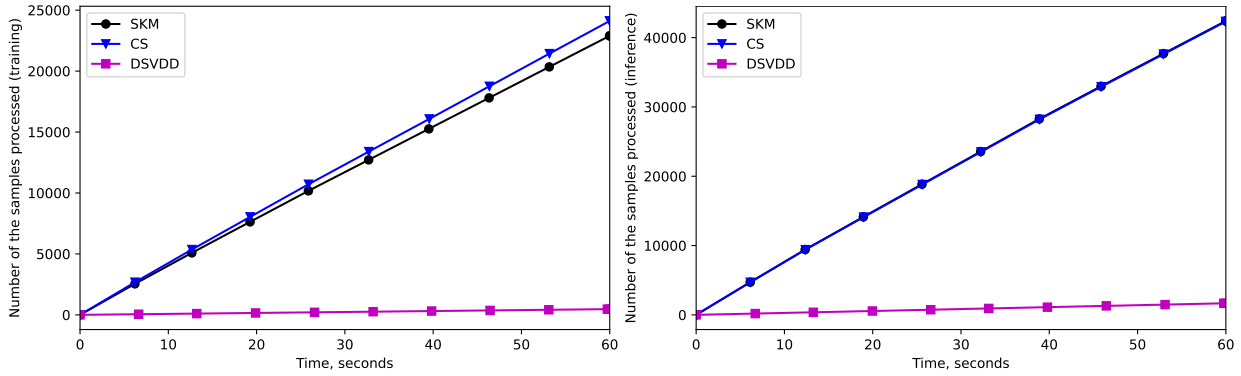
AutoML	Algorithm	Board				
		Arduino Uno Rev3	Arduino Nano Every	Arduino Nano 33 IoT	Arduino Nano 33 BLE Sense	LoRa-E5 mini
RAY	CS	**66.6361**	**66.6361**	**66.6361**	66.6361	66.6361
RAY	SKM	66.5111	66.5111	66.5111	66.5111	66.5111
RAY	DSVDD	62.0187	64.4496	65.4609	**66.8445**	**66.9417**
FLAML	CS	66.4194	66.4194	66.4194	66.4194	66.4194
FLAML	SKM	66.5111	66.5111	66.5111	66.5111	66.5111
FLAML	DSVDD	62.0187	63.6884	64.4052	66.1416	65.9916

Table 3: Average anomaly detection accuracy (%) calculated for various anomaly detection algorithms implemented under several board memory constraints using the data collected in the bearing test environment. Hyperparameters for the algorithms are found using two AutoML frameworks: Ray and Flaml. The higher the value the better.

Algorithm	Board				
	Arduino Uno Rev3	Arduino Nano Every	Arduino Nano 33 IoT	Arduino Nano 33 BLE Sense	LoRa-E5 mini
SKM	3 clusters with 4 samples per batch				
CS	3 clusters and 4 microclusters				
DSVDD	network: 12-9-4	network: 12-25-22	network: 12-36-22	network: 12-110-85	network: 12-110-85

Table 4: Hyperparameter combinations found using the AutoML approach for three best anomaly detection algorithms implemented under several board memory constraints.

the best accuracy values among the combinations found with Ray and Flaml. These combinations can be found in Table 4.



(a) Data processing rate (training).

(b) Data processing rate (inference).

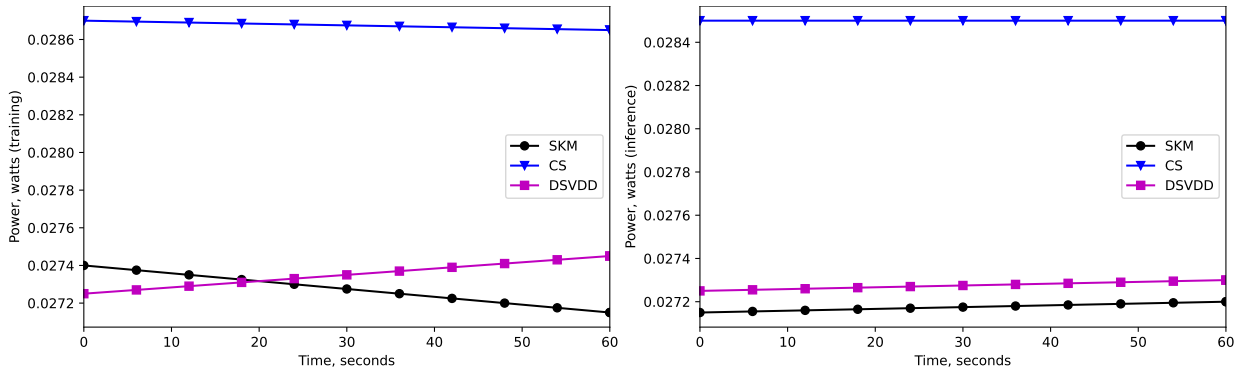
Figure 3: The number of samples that can be processed by each of three algorithms implemented on LoRa E5 during the training and inference stage. Duration of each stage is one minute.

To calculate the training and the inference data rate of each algorithm implementation, we count the number of samples which can be processed by the algorithm during a one minute interval. The results obtained for LoRa-E5 can be found in Figure 3. For Arduino boards, the experiment results can be found in Figures B.5c–B.8c and Figures B.5d–B.8d for the training and the inference stage respectively. In addition, Table 5 shows the average time of data samples which can be processed in one second during the training and the inference stage on each of the boards

Stage	Algorithm	Board				
		Arduino Uno Rev3	Arduino Nano Every	Arduino Nano 33 IoT	Arduino Nano 33 BLE Sense	LoRa-E5 mini
Training	SKM	**103.37**	**106.47**	**107.19**	287.97	381.61
Training	CS	48.92	50.01	104.26	**483.01**	**401.92**
Training	DSVDD	73.15	12.60	20.12	28.45	7.85
Inference	SKM	**350.18**	**360.45**	555.58	**1001.32**	**706.82**
Inference	CS	349.67	359.80	**555.59**	1000.48	705.46
Inference	DSVDD	175.06	35.62	60.98	122.61	27.88

Table 5: The number of samples that can be processed in one second by each of three algorithms implemented on the LoRa-E5 board at the training and inference stage measured in the bearing test environment. The higher the value the better.

tested. As one can see from the results obtained, in most of the cases, stream clustering based anomaly detection methods provide for significantly faster data processing rate values during both the training and inference stage. The only exception is that on Arduino Uno the DSVDD model outperforms CluStream during the training stage which can be explained by the tiny size of the deep learning model deployed in that case. Another interesting observation is that during the training stage, SKM outperforms CS on the boards with AVR CPUs, whereas in the case of the more powerful Cortex-M4, the situation is the opposite. During the inference stage, both stream clustering based algorithms process the same number of samples as expected.



(a) Power consumption (training).

(b) Power consumption (inference).

Figure 4: The energy consumed by each of three algorithms implemented on the LoRa-E5 board during the training and inference stage. Duration of each stage is one minute.

For the energy efficiency assessment, we perform the same experiments as previously. To evaluate the energy consumed, we use Joulescope precision direct current energy analyzer to measure the electric current. The voltage supplied is the same for each board and it is equal to 5 volts. For LoRa-E5 the resulting power values are shown in Figure 4. For Arduino boards, the results can be seen in Figures B.5e–B.8e and Figures B.5f–B.8f for the training and the inference stage respectively. As one can notice looking at the figures, the energy values for each board are almost exactly the same for each of the algorithms tested. It can be explained by the fact that the most of the current consumption comes from other components of the board such as LEDs and voltage regulators and it does not really depend on the machine learning algorithm running on its microcontroller. Nevertheless, we can compare the energy required to process one data sample by each algorithm during the training and the inference stage. These values in milliwatts can be found in Table 6. As one can see from the results presented, stream clustering based anomaly detection methods outperform deep learning based DSVDD significantly in terms of energy efficiency.

Stage	Algorithm	Board				
		Arduino Uno Rev3	Arduino Nano Every	Arduino Nano 33 IoT	Arduino Nano 33 BLE Sense	LoRa-E5 mini
Training	SKM	**2.1385**	**1.6319**	**1.0941**	0.3135	0.0727
Training	CS	4.5033	3.4549	1.1015	**0.1862**	**0.0725**
Training	DSVDD	3.0222	14.7017	5.8424	3.1243	3.5430
Inference	SKM	0.6304	**0.4934**	0.2114	0.0896	**0.0391**
Inference	CS	**0.6295**	0.4953	**0.2114**	**0.0893**	0.0411
Inference	DSVDD	1.2619	5.2011	1.9306	0.7306	0.9952

Table 6: Energy (milliwatts) required for processing one data sample by each of three algorithms implemented at the training and inference stage measured in the bearing test environment. The lower the value the better.

3.5. Results discussion

The results obtained allow us to conclude that despite the recent advancements in deep learning, the methods that rely on neural networks are not always a good choice for deploying on resource-constrained microcontrollers especially on the ones with less powerful CPUs and smaller RAM volumes such as Arduino Uno and Arduino Nano Every. In this case, the anomaly detection algorithms based on stream clustering look like a more promising solution as they are nearly as accurate as deep learning based methods and much faster during both the training and the inference stage. In particular, in our experiments conducted in the bearing test environment, scalable k-means++ is probably the best choice for deploying on the most resource-constrained boards. Even though its accuracy is slightly lower than the one obtained with CluStream, when the number of clusters is low, SKM can process two times more samples during the training stage. In the case of the boards with bigger quantities of computing and memory resources such as Arduino Nano 33 BLE Sense and LoRa-E5 mini, the situation is not that one-sided. In our experiments on these boards, neural network based DSVDD outperforms other algorithms in terms of accuracy, but it is not nearly that efficient in terms of the data processing rate compared to the stream clustering based anomaly detection methods. Thus, the algorithm selection in that case would depend on the characteristics of the data stream observed in the environment in which the device with the model is expected to be deployed.

4. Prototype implementation

The research described in this study has been carried out as a part of the project one of the goals of which is to implement a device that can be employed in an industrial environment for detection of anomalous vibrations. In mechanisms that include rotating parts, for example fans and pumps, such anomalous vibrations can indicate that there is a critical defect in one of the parts which may result in the mechanism malfunction. Thus, there is a need for such a tinyML-driven device that can be easily attached to a normally functioning industrial mechanism to train its machine learning model which can be then used to detect anomalous vibrations if they take place on this or other mechanism of the same kind.

In order to implement such a device, we use the DSVDD model as it has provided for the best results in terms of accuracy for boards with bigger RAM values such as Arduino Nano 33 BLE Sense and LoRa-E5 mini whereas the sample processing rate reached by the model satisfies the requirements posed by the data stream observed in the environment under consideration. The energy consumption is also not a problem in our particular use case as there is a source of electricity in the area where the device is planned to be deployed and therefore it can be powered over a USB cable.

The resulting anomalous vibration detection device prototype (see Figure 1c) consists of the LoRa-E5 mini development board and LSM9DS1 inertial measurement unit (IMU) connected through an inter-integrated circuit communication channel. Power for the device is fed through a USB connection. The chassis has a powerful magnet to attach the device to the mechanism on which model training and/or inference is going to take place. The device is a class A LoRa-device meaning communication is always initiated by the device, and downlink is only possible in two receive

windows after the uplink message. The downlink capability is not utilised in our prototype. While it is possible to create your own LoRa network, our prototype utilises a nationwide network provided by Digita.

The prototype functions as follows. During the first stage that lasts for one day, the tinyML model deployed is trained. After that, the device switches into the inference mode. In this mode, vibration data samples are observed and classified for one minute. During this one minute interval, an anomaly score value is calculated as the mean of the distances between the data samples observed and the centre of the DSVDD model. This anomaly score is then sent to an Azure cloud database via a LoRaWAN gateway. Overall, each LoRa message has a 9 byte payload. In addition to the two-byte anomaly score it includes a one-byte alert value, and three two-byte latest IMU measurements. Once the message has been sent, the device enters a five-minute sleep period during which the inference does not take place. After successful testing in our bearing environment, the resulting prototype has been deployed on a pump in the maintenance room of our university office building for detection of abnormal behavioural patterns.

5. Conclusion

In this study, we have evaluated multiple anomaly detection algorithms in terms of accuracy, false positive and true positive rates that can be implemented on tiny microcontrollers with limited computing and memory resources. In addition, we have measured the data processing and power consumption rates during the training and the inference stages for three of the most accurate algorithms evaluated. Finally, we have used the best of the algorithms tested to implement the anomalous vibration detection device that can be deployed on the edge and report the detection results over LoRaWAN. In the future, we are going to develop novel anomaly detection methods which can be implemented and trained under the constraints posed by limited computing and memory resources available at microcontrollers. Furthermore, we are planning to extend the scope of the research by comparing various supervised and reinforcement machine learning algorithms that can be deployed on resource-constrained devices in realistic use case scenarios.

Acknowledgments

This research has been done as a part of ADDing VAlue by Computing in Manufacturing (coADDVA) project funded by the European Regional Development Fund (ERDF) and the Regional Council of Central Finland.

Data availability

Anomaly detection datasets used in the study as well as the algorithm implementations and scripts for the experiments carried out can be found in our Github repository https://github.com/mizolotu/tinyml_experiments.

Appendix A. Evaluation of anomaly detection algorithms on benchmark datasets

Dataset	Algorithm accuracy, %											
	SKM	CS	WAP	WCM	SOM	GNG	UAE	SAE	DAE	VAE	DSOM	DSVDD
annthyroid	62.00	60.13	61.10	62.12	**63.65**	59.19	50.01	50.30	50.38	50.01	58.49	63.50
breastw	**98.86**	**98.86**	**98.86**	98.30	97.35	97.73	**98.86**	**98.86**	**98.86**	**98.86**	93.56	96.59
cardio	74.62	71.25	75.23	76.74	66.16	**82.18**	56.50	56.60	56.55	56.85	67.62	65.76
cover	68.96	**84.14**	69.87	67.18	81.09	77.95	50.00	50.00	50.00	50.00	60.99	66.12
glass	50.00	53.25	50.00	54.88	51.22	50.00	58.54	58.54	58.54	**64.63**	59.35	50.00
http	99.59	99.82	99.77	99.55	99.02	99.79	99.92	99.92	99.92	**99.95**	99.76	99.54
ionosphere	71.48	**93.33**	86.67	88.89	80.37	81.11	67.41	67.41	67.41	66.30	77.41	76.67
letter	50.28	50.83	53.17	52.83	53.44	52.33	50.00	50.00	50.00	50.00	**53.83**	53.78
mammography	62.93	**80.36**	71.20	59.65	62.22	53.00	60.00	59.89	59.89	60.09	55.08	65.74
pendigits	65.45	63.04	64.64	68.44	75.09	**75.63**	50.00	50.00	50.00	50.00	52.41	62.47
pima	**55.00**	53.00	52.00	51.50	52.67	50.50	51.00	51.00	51.00	51.50	52.33	53.00
satellite	72.35	69.28	71.67	**72.87**	72.53	70.65	50.00	50.00	50.00	50.00	68.90	71.05
satimage-2	96.93	96.95	97.29	**97.64**	95.75	96.99	50.00	50.00	50.00	50.00	63.74	91.88
seismic-bumps	50.31	50.93	50.00	50.62	50.24	51.97	51.24	51.24	51.24	51.45	**52.01**	50.80
shuttle	97.91	97.79	97.93	97.79	98.42	98.19	98.14	98.14	98.14	98.14	62.61	**98.46**
smtp	**79.86**	79.77	79.62	79.69	79.59	79.74	50.00	50.00	50.00	50.00	79.77	79.72
thyroid	77.26	82.93	78.23	**85.92**	84.20	80.68	51.84	51.41	52.45	51.59	76.83	77.64
vertebral	**64.29**	50.00	50.00	50.00	50.40	50.00	50.00	50.00	50.00	50.00	50.00	50.00
vowels	64.41	55.63	65.12	57.12	62.57	56.23	50.00	50.00	50.00	51.25	65.01	**67.20**
wbc	58.22	63.38	**71.83**	64.79	63.38	57.75	53.05	50.47	53.52	54.46	64.55	71.83
valve1_0	87.06	88.46	88.46	**88.81**	85.90	75.52	50.00	50.00	50.00	50.00	69.11	80.77
valve1_1	86.15	**87.91**	84.15	83.45	78.87	83.10	50.00	50.00	50.00	50.00	83.10	87.44
valve1_2	77.42	73.05	70.57	**90.78**	74.70	58.87	50.00	50.00	50.00	50.00	70.69	73.52
valve1_3	96.60	96.71	96.48	96.48	96.60	95.77	95.66	95.89	95.77	96.13	96.48	**96.83**
valve1_4	76.92	**84.27**	74.13	75.52	66.67	55.24	50.00	50.00	50.00	50.00	62.94	63.29
valve1_5	87.50	88.08	88.19	**88.54**	87.38	88.19	53.12	53.24	56.71	51.62	87.50	85.19
valve1_6	**91.26**	91.14	90.91	90.21	90.21	90.21	80.89	75.41	76.57	87.76	90.21	89.28
valve1_7	86.77	83.97	81.68	84.35	81.17	80.53	90.08	**90.71**	80.92	85.88	86.13	82.70
valve1_8	**96.83**	95.77	94.01	94.37	81.46	94.37	50.00	50.00	50.00	50.12	90.85	95.19
valve1_9	95.80	97.90	96.85	**98.25**	93.71	93.01	53.73	52.33	58.28	50.58	94.06	93.71
valve1_10	97.18	**97.54**	96.83	**97.54**	94.25	96.48	84.74	88.73	87.21	90.38	95.31	96.48
valve1_11	83.69	84.04	83.10	83.80	**86.27**	83.10	50.00	50.82	51.53	50.70	84.86	82.39
valve1_12	91.08	**99.65**	90.49	89.79	98.00	89.79	50.00	50.00	50.00	50.00	91.67	90.73
valve1_13	92.61	92.02	90.85	92.25	78.52	92.25	76.64	88.15	81.10	87.79	**93.08**	92.02
valve1_14	88.89	89.36	89.36	88.77	**89.72**	88.65	73.40	75.77	86.41	86.52	89.13	89.48
valve1_15	92.66	**95.45**	94.76	**95.45**	82.17	94.76	52.10	53.50	57.11	52.91	**95.45**	94.06
valve2_0	91.67	**95.95**	90.71	90.00	92.98	85.36	50.00	50.00	50.00	50.00	91.31	91.79
valve2_1	**91.85**	90.89	90.29	91.37	91.13	89.57	68.59	80.70	79.74	85.25	90.53	91.37
valve2_2	90.71	**92.38**	90.71	90.36	81.90	90.36	50.00	50.00	50.00	50.00	90.60	90.48
valve2_3	91.59	**94.25**	91.59	92.04	90.56	91.15	73.60	79.50	81.56	85.10	91.30	90.86

Table A.7: Accuracy (%) of various anomaly detection algorithms implemented under Arduino Uno Rev3 memory constraints using ODDS and SKAB benchmark datasets.

Dataset	Algorithm accuracy, %											
	SKM	CS	WAP	WCM	SOM	GNG	UAE	SAE	DAE	VAE	DSOM	DSVDD
annthyroid	62.00	60.13	62.94	62.53	**63.65**	58.06	50.04	50.13	50.40	50.01	59.69	62.64
breastw	98.86	98.86	98.86	98.30	97.54	97.73	98.86	98.86	98.86	**99.05**	94.89	98.48
cardio	77.49	76.13	81.87	76.49	65.36	**82.18**	56.34	56.50	56.65	56.70	64.30	70.09
cover	68.96	**82.90**	78.55	67.25	81.42	62.34	50.00	50.00	50.00	50.00	59.22	68.36
glass	50.00	51.22	50.00	54.88	51.22	50.00	58.54	58.54	58.54	**63.41**	54.88	50.00
http	99.59	99.82	99.67	99.55	99.02	98.82	99.92	99.92	99.92	**99.95**	99.88	99.61
ionosphere	83.70	**93.33**	80.00	88.89	74.81	81.11	67.41	68.15	67.41	69.26	82.22	84.81
letter	54.39	51.44	53.17	53.33	**58.06**	52.33	50.00	50.00	50.00	50.00	55.06	53.67
mammography	62.93	**80.36**	71.20	62.71	62.22	51.69	59.89	59.89	59.89	60.09	53.06	64.61
pendigits	65.45	63.04	66.62	76.08	78.01	**83.38**	50.00	50.00	50.00	50.00	50.58	81.21
pima	**55.00**	53.00	52.00	51.17	52.67	50.50	51.00	51.00	51.00	51.50	52.00	52.17
satellite	71.79	73.11	71.27	72.87	**73.15**	69.97	50.00	50.00	50.00	50.00	70.42	72.17
satimage-2	97.19	96.95	**97.47**	97.34	94.97	93.89	50.00	50.00	50.00	50.00	79.83	96.92
seismic-bumps	50.55	50.93	50.00	50.62	50.86	**51.97**	51.24	51.24	51.24	51.45	51.90	50.35
shuttle	97.91	97.85	98.27	97.89	98.42	98.18	98.14	98.14	98.14	98.14	64.26	**98.83**
smtp	79.86	79.77	79.69	79.71	79.59	**79.87**	50.00	50.00	50.00	50.00	76.40	76.61
thyroid	77.26	82.93	83.74	83.74	**84.20**	80.68	52.29	52.27	53.40	51.88	75.31	75.85
vertebral	**64.29**	50.00	50.00	50.00	50.40	50.00	50.00	50.00	50.00	50.00	51.59	50.00
vowels	64.41	56.52	65.84	61.03	58.36	56.23	50.00	50.00	50.00	50.59	63.35	**76.93**
wbc	63.38	66.67	**75.35**	71.13	63.38	57.75	52.58	50.47	53.76	53.05	60.80	71.60
valve1_0	87.06	88.46	85.31	**88.81**	85.90	75.52	50.00	50.00	50.00	50.00	75.17	83.92
valve1_1	86.15	87.91	**88.73**	83.45	78.87	83.10	50.00	50.00	50.00	50.00	87.44	87.56
valve1_2	77.42	73.05	86.88	**90.78**	74.70	58.87	50.00	50.00	50.00	50.00	68.56	80.50
valve1_3	96.60	96.71	96.48	96.48	96.60	95.77	95.66	95.89	95.66	91.31	**97.42**	96.48
valve1_4	76.92	84.27	**85.31**	72.03	71.45	55.24	50.00	50.00	50.00	50.00	66.43	60.14
valve1_5	87.50	88.31	86.11	**88.54**	87.38	88.19	52.89	54.63	52.20	51.74	85.76	87.38
valve1_6	91.26	91.14	90.21	90.21	90.21	90.21	69.11	72.61	74.94	62.59	**91.61**	89.28
valve1_7	86.77	84.48	84.73	83.72	81.17	80.53	80.03	**90.84**	88.55	66.79	86.77	82.82
valve1_8	**96.83**	95.77	94.01	95.42	81.46	94.37	50.00	50.00	50.00	50.12	83.33	95.42
valve1_9	95.80	**97.90**	96.50	96.85	93.71	93.01	51.52	53.03	52.10	51.52	92.66	93.24
valve1_10	97.18	**97.77**	96.83	97.54	94.25	96.48	85.92	76.53	85.56	80.75	96.60	96.83
valve1_11	83.69	84.04	83.10	83.22	**86.27**	83.10	51.17	51.53	52.70	50.00	84.04	82.86
valve1_12	91.08	**99.65**	98.24	93.78	98.00	89.79	50.00	50.00	50.00	50.00	84.86	97.18
valve1_13	92.61	92.02	90.85	**94.48**	92.37	92.25	76.17	84.04	85.45	74.77	92.49	92.96
valve1_14	88.89	89.36	**90.43**	**90.43**	89.72	88.65	77.78	85.82	75.18	86.64	89.83	89.95
valve1_15	92.66	**95.45**	93.71	**95.45**	82.17	94.76	52.33	57.34	56.76	50.12	92.54	94.41
valve2_0	91.67	**95.95**	93.21	90.00	92.98	85.36	50.00	50.00	50.00	50.00	94.29	95.48
valve2_1	91.85	90.89	90.65	**92.45**	91.13	89.57	86.09	85.13	72.90	66.91	90.65	90.77
valve2_2	90.71	**92.38**	90.36	90.36	81.90	90.36	50.00	50.00	50.00	50.00	91.79	92.26
valve2_3	91.59	**94.25**	91.59	93.36	90.56	91.15	76.11	79.35	88.20	81.42	91.30	92.18

Table A.8: Accuracy (%) of various anomaly detection algorithms implemented under Arduino Nano Every memory constraints using ODDS and SKAB benchmark datasets.

Dataset	Algorithm accuracy, %											
	SKM	CS	WAP	WCM	SOM	GNG	UAE	SAE	DAE	VAE	DSOM	DSVDD
annthyroid	62.00	60.13	61.97	62.53	**63.65**	58.06	50.44	50.41	50.14	50.00	58.31	61.88
breastw	**98.86**	**98.86**	**98.86**	98.30	97.54	97.73	**98.86**	**98.86**	**98.86**	**98.86**	92.42	98.11
cardio	77.49	71.25	81.87	79.00	65.36	**82.18**	56.50	56.50	56.50	56.60	65.21	69.39
cover	68.96	**82.90**	78.55	67.25	81.42	67.18	50.00	50.00	50.00	50.00	58.59	73.67
glass	50.00	51.22	50.00	56.10	51.22	50.00	58.54	58.54	58.54	**63.01**	57.72	51.22
http	99.59	99.82	99.67	99.60	99.02	99.37	99.92	99.92	99.92	**99.96**	99.77	99.53
ionosphere	82.59	**93.33**	84.44	88.89	74.81	81.11	67.04	67.04	67.78	70.00	78.52	86.67
letter	54.39	52.89	53.67	53.33	58.06	52.33	50.00	50.00	50.00	50.00	**58.17**	54.11
mammography	62.93	**80.36**	71.20	63.13	62.22	52.27	59.94	59.89	59.89	60.09	54.40	64.05
pendigits	65.45	63.04	66.62	76.08	78.01	**83.38**	50.00	50.00	50.00	50.00	50.91	77.74
pima	**55.00**	53.00	52.00	51.67	52.67	50.50	51.00	51.00	51.00	51.50	54.67	51.33
satellite	71.79	71.99	72.53	**73.08**	73.08	69.97	50.00	50.00	50.00	50.00	68.07	72.15
satimage-2	97.19	96.95	**97.47**	97.34	94.97	93.89	50.00	50.00	50.00	50.00	81.82	96.76
seismic-bumps	50.55	51.73	50.31	50.62	50.86	**51.97**	51.24	51.24	51.24	51.24	50.69	50.14
shuttle	97.91	97.85	98.52	97.98	98.42	97.79	98.14	98.14	98.14	98.14	68.08	**98.54**
smtp	**79.86**	79.77	79.69	79.71	79.59	69.82	50.00	50.00	50.00	50.00	76.37	79.74
thyroid	77.26	82.93	83.47	83.74	**84.20**	80.68	52.31	52.04	53.06	50.68	74.83	76.80
vertebral	**64.29**	50.00	50.00	50.00	50.40	50.00	50.00	50.00	50.00	50.00	53.97	50.00
vowels	64.41	56.52	67.97	61.03	58.36	56.23	50.00	50.00	50.00	50.00	61.33	**87.96**
wbc	63.38	66.20	**75.35**	70.42	64.32	57.75	50.47	51.41	52.35	52.82	64.08	71.60
valve1_0	87.06	88.46	85.31	**88.81**	85.90	75.52	50.00	50.00	50.00	50.00	81.59	88.00
valve1_1	86.15	87.91	89.79	83.45	78.87	83.10	50.00	50.00	50.00	50.00	86.97	**92.02**
valve1_2	77.42	73.05	90.43	**90.78**	74.70	58.87	50.00	50.00	50.00	50.00	80.38	88.42
valve1_3	96.60	**96.71**	96.48	96.48	96.60	95.77	95.42	95.77	95.77	96.13	96.36	96.24
valve1_4	76.92	84.27	**85.31**	72.03	71.45	55.24	50.00	50.00	50.00	50.00	69.93	74.24
valve1_5	87.50	88.31	86.11	**88.54**	87.38	88.19	53.01	53.01	57.06	52.08	87.50	86.92
valve1_6	**91.26**	91.14	90.21	90.21	90.21	90.21	80.65	80.42	84.62	86.48	**91.26**	89.98
valve1_7	86.77	84.48	84.73	85.50	81.17	80.53	89.19	85.11	**89.57**	87.40	84.99	82.57
valve1_8	**96.83**	95.77	94.01	95.42	81.46	94.37	50.00	50.00	50.00	50.00	76.06	95.89
valve1_9	95.80	**97.90**	96.50	96.85	93.71	93.01	50.00	54.20	52.45	53.85	92.31	95.57
valve1_10	97.18	**97.77**	96.83	97.54	94.25	96.48	77.46	90.61	76.29	87.68	95.07	97.07
valve1_11	83.69	84.04	83.10	83.92	**86.27**	83.10	50.00	51.53	50.47	50.00	83.57	82.75
valve1_12	91.08	**99.65**	98.24	95.07	98.00	89.79	50.00	50.00	50.00	50.00	92.37	97.42
valve1_13	92.61	92.02	92.96	**94.48**	92.37	92.25	86.03	76.64	87.09	87.32	91.67	91.43
valve1_14	88.89	89.36	**90.43**	89.01	89.72	88.65	70.33	83.33	79.91	84.99	88.77	89.48
valve1_15	92.66	**95.45**	93.71	**95.45**	82.17	94.76	50.00	58.04	52.10	50.00	94.64	94.99
valve2_0	91.67	**95.95**	93.21	90.00	92.98	85.36	50.00	50.00	50.00	50.00	93.69	93.69
valve2_1	**91.85**	90.89	90.65	91.73	91.13	89.57	86.33	74.58	72.66	81.06	91.37	91.01
valve2_2	90.71	92.38	90.36	90.36	81.90	90.36	50.00	50.00	50.00	50.00	90.36	**93.21**
valve2_3	91.59	**94.25**	91.59	93.36	90.56	91.15	69.47	75.22	75.96	83.63	91.59	91.00

Table A.9: Accuracy (%) of various anomaly detection algorithms implemented under Arduino Nano 33 IoT memory constraints using ODDS and SKAB benchmark datasets.

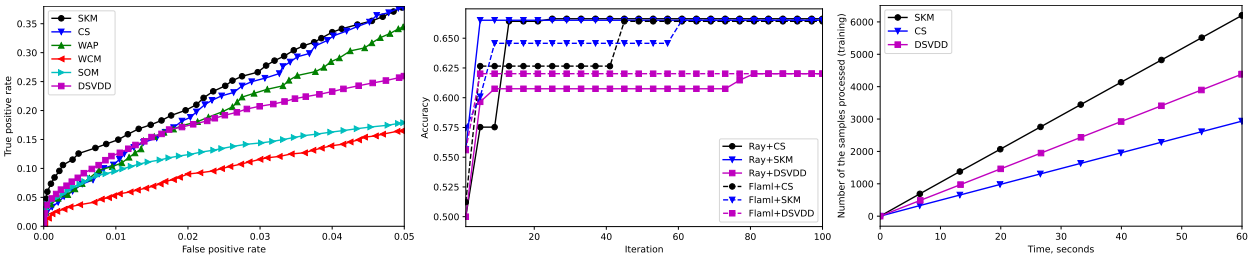
Dataset	Algorithm accuracy, %											
	SKM	CS	WAP	WCM	SOM	GNG	UAE	SAE	DAE	VAE	DSOM	DSVDD
annthyroid	62.00	60.13	61.97	62.53	**63.65**	58.06	50.40	50.04	50.29	50.00	57.43	61.50
breastw	98.86	98.86	98.86	98.30	97.54	97.73	98.86	98.86	98.86	**99.05**	89.20	98.30
cardio	77.49	71.25	81.87	79.00	65.36	**82.18**	56.34	56.50	56.50	56.60	61.28	69.64
cover	68.96	**82.90**	78.55	67.25	81.42	67.18	50.00	50.00	50.00	50.00	58.38	72.96
glass	50.00	51.22	50.00	56.10	51.22	50.00	58.54	58.54	58.54	**63.01**	56.50	50.00
http	99.59	99.82	99.67	99.60	99.02	99.37	99.92	99.92	99.92	**99.96**	99.78	99.55
ionosphere	82.59	**93.33**	84.44	88.89	74.81	81.11	67.41	67.04	68.15	68.52	80.00	83.70
letter	54.39	52.89	53.67	53.33	**58.06**	52.33	50.00	50.00	50.00	50.00	56.00	54.28
mammography	62.93	**80.36**	71.20	63.13	62.22	52.27	59.89	59.89	59.89	60.19	53.59	65.80
pendigits	65.45	63.04	66.62	76.08	78.01	**83.38**	50.00	50.00	50.00	50.00	51.03	73.75
pima	**55.00**	53.00	52.00	51.67	52.67	50.50	51.00	51.00	51.00	51.50	53.00	52.17
satellite	71.79	71.99	72.53	**73.08**	73.08	69.97	50.00	50.00	50.00	50.00	67.92	71.50
satimage-2	97.19	96.95	**97.47**	97.34	94.97	93.89	50.00	50.00	50.00	50.00	74.08	96.73
seismic-bumps	50.55	51.73	50.31	50.62	50.86	**51.97**	51.24	51.24	51.24	51.24	50.73	50.14
shuttle	97.91	97.85	98.52	97.98	98.42	97.79	98.14	98.14	98.14	98.14	61.83	**99.11**
smtp	**79.86**	79.77	79.69	79.71	79.59	69.82	50.00	50.00	50.00	50.00	76.33	79.77
thyroid	77.26	82.93	83.47	83.74	**84.20**	80.68	52.36	52.22	52.31	50.68	70.93	79.89
vertebral	**64.29**	50.00	50.00	50.00	50.40	50.00	50.00	50.00	50.00	50.00	54.37	50.00
vowels	64.41	56.52	67.97	61.03	58.36	56.23	50.00	50.00	50.00	50.00	58.24	**81.02**
wbc	63.38	66.20	**75.35**	70.42	64.32	57.75	50.00	50.00	50.47	51.88	57.51	65.96
valve1_0	87.06	88.46	85.31	**88.81**	85.90	75.52	50.00	50.00	50.00	50.00	80.77	88.34
valve1_1	86.15	87.91	89.79	83.45	78.87	83.10	50.00	50.00	50.00	50.00	88.85	**90.38**
valve1_2	77.42	73.05	90.43	**90.78**	74.70	58.87	50.00	50.00	50.00	50.00	73.40	84.99
valve1_3	96.60	96.71	96.48	96.48	96.60	95.77	95.54	95.77	95.66	96.01	**97.77**	96.60
valve1_4	76.92	84.27	**85.31**	72.03	71.45	55.24	50.00	50.00	50.00	50.00	68.41	66.78
valve1_5	87.50	88.31	86.11	**88.54**	87.38	88.19	51.74	53.24	52.89	52.31	**88.54**	87.04
valve1_6	**91.26**	91.14	90.21	90.21	90.21	90.21	66.78	73.08	72.14	67.95	91.03	89.98
valve1_7	86.77	84.48	84.73	85.50	81.17	80.53	79.13	87.02	**88.55**	85.75	86.26	82.82
valve1_8	**96.83**	95.77	94.01	95.42	81.46	94.37	50.00	50.00	50.00	50.00	70.42	95.31
valve1_9	95.80	**97.90**	96.50	96.85	93.71	93.01	50.00	50.00	51.52	50.82	92.89	94.06
valve1_10	97.18	**97.77**	96.83	97.54	94.25	96.48	76.17	76.06	76.53	78.52	95.77	96.95
valve1_11	83.69	84.04	83.10	83.92	**86.27**	83.10	50.00	50.00	50.00	50.12	83.33	83.57
valve1_12	91.08	**99.65**	98.24	95.07	98.00	89.79	50.00	50.00	50.00	50.00	93.54	98.12
valve1_13	92.61	92.02	92.96	**94.48**	92.37	92.25	75.82	88.26	76.29	86.50	92.25	91.67
valve1_14	88.89	89.36	**90.43**	89.01	89.72	88.65	72.93	85.58	78.72	70.80	89.01	89.72
valve1_15	92.66	**95.45**	93.71	**95.45**	82.17	94.76	50.00	50.00	50.58	50.00	94.29	95.34
valve2_0	91.67	**95.95**	93.21	90.00	92.98	85.36	50.00	50.00	50.00	50.00	95.36	94.88
valve2_1	**91.85**	90.89	90.65	91.73	91.13	89.57	68.11	73.26	73.14	65.95	91.01	91.37
valve2_2	90.71	92.38	90.36	90.36	81.90	90.36	50.00	50.00	50.00	50.00	89.88	**92.86**
valve2_3	91.59	**94.25**	91.59	93.36	90.56	91.15	71.24	74.19	85.40	71.39	91.89	90.86

Table A.10: Accuracy (%) of various anomaly detection algorithms implemented under Arduino Nano 33 Sense memory constraints using ODDS and SKAB benchmark datasets.

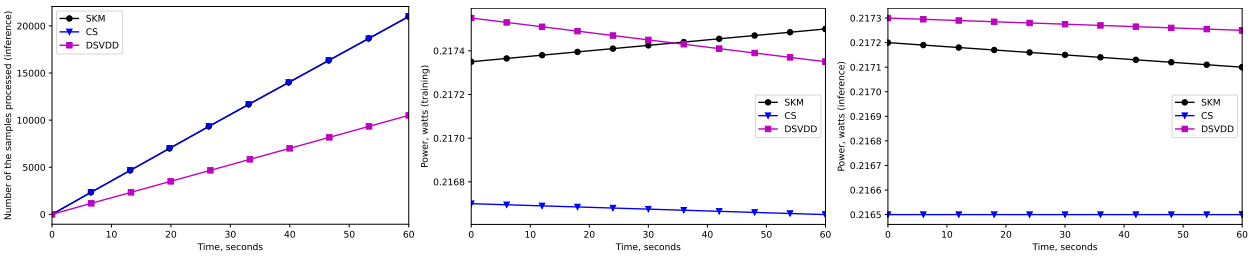
Dataset	Algorithm accuracy, %											
	SKM	CS	WAP	WCM	SOM	GNG	UAE	SAE	DAE	VAE	DSOM	DSVDD
annthyroid	62.00	60.13	61.97	62.53	**63.65**	58.06	50.35	50.10	50.39	50.00	58.15	62.09
breastw	**98.86**	**98.86**	**98.86**	98.30	97.54	97.73	**98.86**	**98.86**	**98.86**	**98.86**	87.31	97.73
cardio	77.49	71.25	81.87	79.00	65.36	**82.18**	56.50	56.34	56.65	56.55	62.79	68.08
cover	68.96	**82.90**	78.55	67.25	81.42	67.18	50.00	50.00	50.00	50.00	58.56	75.42
glass	50.00	51.22	50.00	56.10	51.22	50.00	58.54	58.54	58.54	**63.01**	59.76	51.22
http	99.59	99.82	99.67	99.60	99.02	99.37	99.92	99.92	99.92	**99.96**	99.67	99.87
ionosphere	82.59	**93.33**	84.44	88.89	74.81	81.11	66.67	67.41	68.52	67.41	82.96	84.81
letter	54.39	52.89	53.67	53.33	**58.06**	52.33	50.00	50.00	50.00	50.00	55.44	54.67
mammography	62.93	**80.36**	71.20	63.13	62.22	52.27	59.89	59.89	59.89	59.99	53.96	66.38
pendigits	65.45	63.04	66.62	76.08	78.01	**83.38**	50.00	50.00	50.00	50.00	51.35	74.17
pima	**55.00**	53.00	52.00	51.67	52.67	50.50	51.00	51.00	51.00	51.50	51.33	51.50
satellite	71.79	71.99	72.53	**73.08**	73.08	69.97	50.00	50.00	50.00	50.00	68.58	71.88
satimage-2	97.19	96.95	**97.47**	97.34	94.97	93.89	50.00	50.00	50.00	50.00	76.64	97.31
seismic-bumps	50.55	51.73	50.31	50.62	50.86	**51.97**	51.24	51.24	51.24	51.24	50.73	50.14
shuttle	97.91	97.85	98.52	97.98	98.42	97.79	98.14	98.14	98.14	98.14	65.19	**98.54**
smtp	79.86	79.77	79.69	79.71	76.58	69.82	50.00	50.00	50.00	50.00	79.75	**83.01**
thyroid	77.26	82.93	83.47	83.74	**84.20**	80.68	52.18	52.79	53.49	50.68	76.03	77.19
vertebral	**64.29**	50.00	50.00	50.00	50.40	50.00	50.00	50.00	50.00	50.00	54.37	50.00
vowels	64.41	56.52	67.97	61.03	58.36	56.23	50.00	50.00	50.00	50.00	62.93	**80.84**
wbc	63.38	66.20	**75.35**	70.42	64.32	57.75	50.00	50.47	50.94	51.88	56.81	66.20
valve1_0	87.06	88.46	85.31	**88.81**	85.90	75.52	50.00	50.00	50.00	50.00	77.62	88.58
valve1_1	86.15	87.91	89.79	83.45	78.87	83.10	50.00	50.00	50.00	50.00	**90.14**	89.67
valve1_2	77.42	73.05	90.43	**90.78**	74.70	58.87	50.00	50.00	50.00	50.00	78.84	88.18
valve1_3	96.60	**96.71**	96.48	96.48	96.60	95.77	86.97	95.66	95.54	95.89	96.48	96.36
valve1_4	76.92	84.27	**85.31**	72.03	71.45	55.24	50.00	50.00	50.00	50.00	71.79	68.76
valve1_5	87.50	88.31	86.11	**88.54**	87.38	88.19	51.62	52.89	53.36	53.01	87.73	86.81
valve1_6	**91.26**	91.14	90.21	90.21	90.21	90.21	67.60	73.54	76.46	65.27	90.56	90.56
valve1_7	**86.77**	84.48	84.73	85.50	81.17	80.53	80.03	86.39	78.63	72.14	84.99	81.04
valve1_8	**96.83**	95.77	94.01	95.42	81.46	94.37	50.00	50.00	50.00	50.00	78.05	95.31
valve1_9	95.80	**97.90**	96.50	96.85	93.71	93.01	50.00	50.00	52.56	50.00	94.99	97.55
valve1_10	97.18	**97.77**	96.83	97.54	94.25	96.48	73.12	76.29	79.81	66.43	96.36	96.95
valve1_11	83.69	84.04	83.10	83.92	**86.27**	83.10	50.00	51.53	50.00	50.00	84.74	82.75
valve1_12	91.08	**99.65**	98.24	95.07	98.00	89.79	50.00	50.00	50.00	50.00	89.32	99.30
valve1_13	92.61	92.02	92.96	94.48	92.37	92.25	71.71	88.62	83.45	70.19	**95.19**	92.96
valve1_14	88.89	89.36	**90.43**	89.01	89.72	88.65	74.35	85.46	77.42	69.27	89.72	89.36
valve1_15	92.66	95.45	93.71	95.45	82.17	94.76	52.33	51.17	50.58	50.00	88.69	**96.15**
valve2_0	91.67	**95.95**	93.21	90.00	92.98	85.36	50.00	50.00	50.00	50.00	93.93	93.81
valve2_1	**91.85**	90.89	90.65	91.73	91.13	89.57	73.86	84.53	72.30	56.71	89.93	91.37
valve2_2	90.71	92.38	90.36	90.36	81.90	90.36	50.00	50.00	50.00	50.00	90.36	**94.40**
valve2_3	91.59	**94.25**	91.59	93.36	90.56	91.15	72.86	74.19	76.55	74.78	93.36	90.86

Table A.11: Accuracy (%) of various anomaly detection algorithms implemented under LoRa-E5 mini memory constraints using ODDS and SKAB benchmark datasets.

Appendix B. Evaluation of anomaly detection algorithms in the bearing test environment

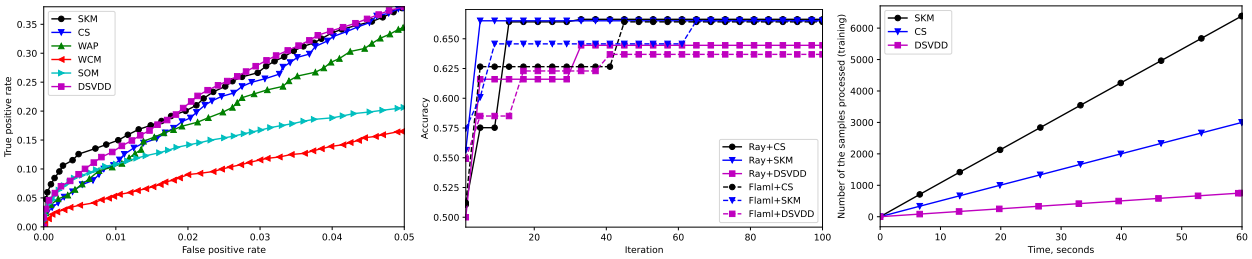


(a) Dependence of TPR on FPR. (b) Accuracy improvement with AutoML. (c) Data processing rate (training).

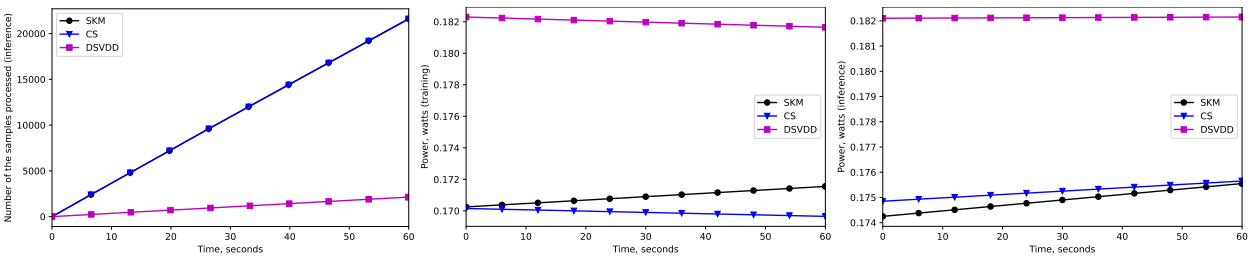


(d) Data processing rate (inference). (e) Power consumption (training). (f) Power consumption (inference).

Figure B.5: Evaluation of anomaly detection algorithms in the bearing test environment on Arduino Uno Rev3. Figure B.5a shows dependence of true positive rate on false positive rate for six anomaly detection algorithms. Accuracy improvement for three best algorithms with two AutoML frameworks can be seen in Figure B.5b. Figures B.5c and B.5d show how many samples can be processed by each of these three algorithms during the training and inference stage respectively; each of the stages lasts for one minute. Power consumption for the same three algorithms during the training and inference stage is shown respectively in Figures B.5e and B.5f.

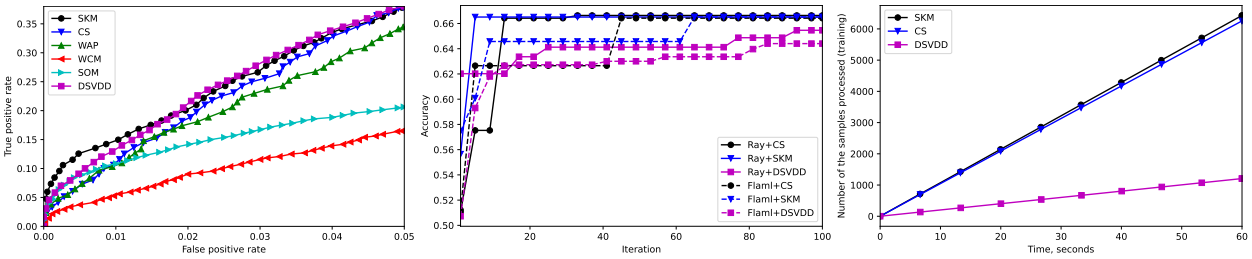


(a) Dependence of TPR on FPR. (b) Accuracy improvement with AutoML. (c) Data processing rate (training).

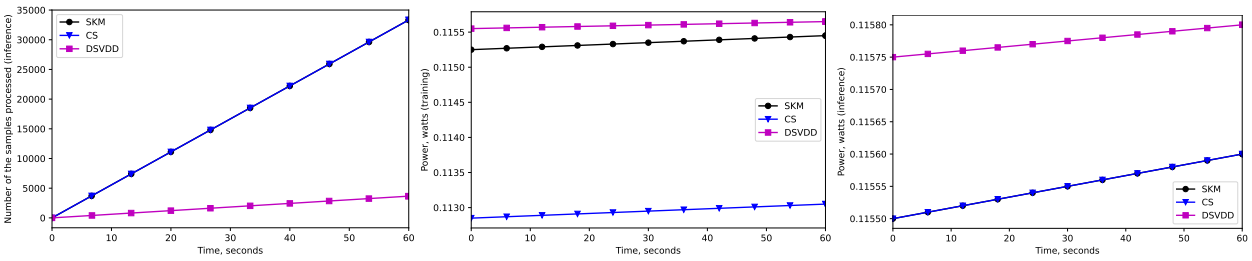


(d) Data processing rate (inference). (e) Power consumption (training). (f) Power consumption (inference).

Figure B.6: Evaluation of anomaly detection algorithms in the bearing test environment on Arduino Nano Every. Figure B.6a shows dependence of true positive rate on false positive rate for six anomaly detection algorithms. Accuracy improvement for three best algorithms with two AutoML frameworks can be seen in Figure B.6b. Figures B.6c and B.6d show how many samples can be processed by each of these three algorithms during the training and inference stage respectively; each of the stages lasts one minute. Power consumption for the same three algorithms during the training and inference stage is shown respectively in Figures B.6e and B.6f.

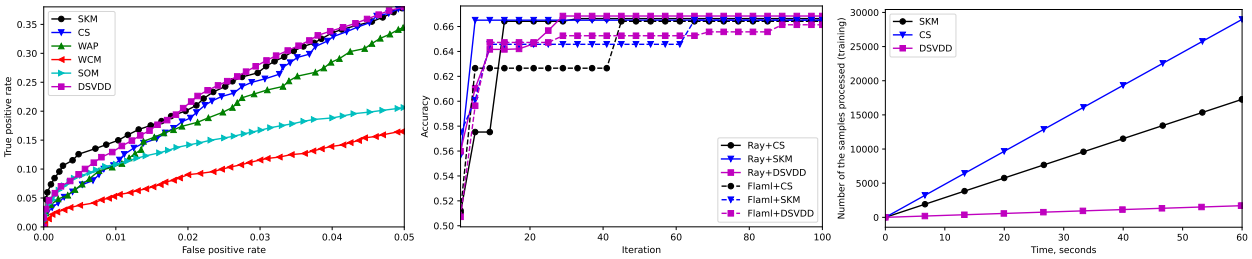


(a) Dependence of TPR on FPR. (b) Accuracy improvement with AutoML. (c) Data processing rate (training).

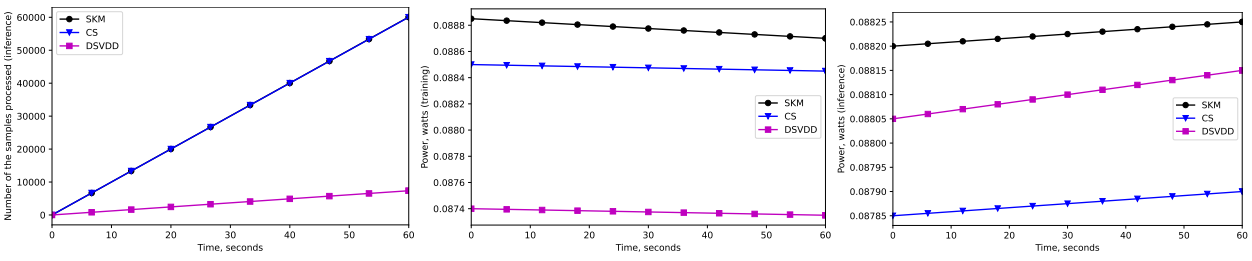


(d) Data processing rate (inference). (e) Power consumption (training). (f) Power consumption (inference).

Figure B.7: Evaluation of anomaly detection algorithms in the bearing test environment on Arduino Nano 33 IoT. Figure B.7a shows dependence of true positive rate on false positive rate for six anomaly detection algorithms. Accuracy improvement for three best algorithms with two AutoML frameworks can be seen in Figure B.7b. Figures B.7c and B.7d show how many samples can be processed by each of these three algorithms during the training and inference stage respectively; each of the stages lasts one minute. Power consumption for the same three algorithms during the training and inference stage is shown respectively in Figures B.7e and B.7f.



(a) Dependence of TPR on FPR. (b) Accuracy improvement with AutoML. (c) Data processing rate (training).



(d) Data processing rate (inference). (e) Power consumption (training). (f) Power consumption during (inference).

Figure B.8: Evaluation of anomaly detection algorithms in the bearing test environment on Arduino Nano 33 BLE Sense. Figure B.8a shows dependence of true positive rate on false positive rate for six anomaly detection algorithms. Accuracy improvement for three best algorithms with two AutoML frameworks can be seen in Figure B.8b. Figures B.8c and B.8d show how many samples can be processed by each of these three algorithms during the training and inference stage respectively; each of the stages lasts one minute. Power consumption for the same three algorithms during the training and inference stage is shown respectively in Figures B.8e and B.8f.

References

[1] H. Fujiyoshi, T. Hirakawa, T. Yamashita, Deep learning-based image recognition for autonomous driving, IATSS research 43 (4) (2019) 244–252.

[2] Y. Liu, A. Jain, C. Eng, D. H. Way, K. Lee, P. Bui, K. Kanada, G. de Oliveira Marinho, J. Gallegos, S. Gabriele, et al., A deep learning system for differential diagnosis of skin diseases, Nature medicine 26 (6) (2020) 900–908.

[3] H. Ren, D. Anicic, T. A. Runkler, Tinyol: Tinyml with online-learning on microcontrollers, in: IJCNN, IEEE, 2021, pp. 1–8.

[4] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, S. Han, On-device training under 256kb memory, in: Annual Conference on Neural Information Processing Systems (NeurIPS), 2022.

[5] S. A. R. Zaidi, A. M. Hayajneh, M. Hafeez, Q. Ahmed, Unlocking edge intelligence through tiny machine learning (tinyml), IEEE Access 10 (2022) 100867–100877.

[6] J. Petajajarvi, K. Mikhaylov, A. Roivainen, T. Hanninen, M. Pettissalo, On the coverage of lpwans: range evaluation and channel attenuation model for lora technology, in: 2015 14th international conference on its telecommunications (itst), IEEE, 2015, pp. 55–59.

[7] P. Jörke, S. Böcker, F. Liedmann, C. Wietfeld, Urban channel models for smart city iot-networks based on empirical measurements of lora-links at 433 and 868 mhz, in: 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), IEEE, 2017, pp. 1–6.

[8] P. Mayer, M. Magno, T. Brunner, L. Benini, Lora vs. lora: In-field evaluation and comparison for long-lifetime sensor nodes, in: 2019 IEEE 8th International Workshop on Advances in Sensors and Interfaces (IWASI), IEEE, 2019, pp. 307–311.

[9] V. M. Suresh, R. Sidhu, P. Karkare, A. Patil, Z. Lei, A. Basu, Powering the iot through embedded machine learning and lora, in: 2018 IEEE 4th World Forum on Internet of Things (WF-IoT), IEEE, 2018, pp. 349–354.

[10] Y.-C. Chang, T.-W. Huang, N.-F. Huang, A machine learning based smart irrigation system with lora p2p networks, in: 2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS), IEEE, 2019, pp. 1–4.

[11] J. Cardoso, A. Glória, P. Sebastião, A methodology for sustainable farming irrigation using wsn, nb-iot and machine learning, in: 2020 5th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM), IEEE, 2020, pp. 1–6.

[12] C. Zhu, S. He, Y. Yan, J. Xiao, Temperature monitoring and analysis of low-power breeding pigs based on machine learning, in: 2022 8th International Conference on Control, Automation and Robotics (ICCAR), IEEE, 2022, pp. 173–177.

[13] R. Adeogun, I. Rodriguez, M. Razzaghpour, G. Berardinelli, P. H. Christensen, P. E. Mogensen, Indoor occupancy detection and estimation using machine learning and measurements from an iot lora-based monitoring system, in: 2019 Global IoT Summit (GIoTS), IEEE, 2019, pp. 1–5.

[14] B. Jones, U. Raza, A. Khan, Tiny but mighty: Embedded machine learning for indoor wireless localization, in: 2023 IEEE 20th Consumer Communications & Networking Conference (CCNC), IEEE, 2023, pp. 176–181.

[15] A. Javed, H. Larijani, A. Wixted, Improving energy consumption of a commercial building with iot and machine learning, IT Professional 20 (5) (2018) 30–38.

[16] C. A. Estrebou, M. Fleming, M. D. Saavedra, F. Adra, A. E. De Giusti, Lightweight convolutional neural networks framework for really small tinyml devices, in: SmartTech-IC, Springer, 2022, pp. 3–16.

[17] S. Gupta, S. Jain, B. Roy, A. Deb, A tinyml approach to human activity recognition, in: Journal of Physics: Conference Series, Vol. 2273, IOP Publishing, 2022, p. 012025.

[18] M. Antonini, M. Pincheira, M. Vecchio, F. Antonelli, An adaptable and unsupervised tinyml anomaly detection system for extreme industrial environments, Sensors 23 (4) (2023) 2344.

[19] W. G. Negera, F. Schwenker, T. G. Debelee, H. M. Melaku, D. W. Feyisa, Lightweight model for botnet attack detection in software defined network-orchestrated iot, Applied Sciences 13 (8) (2023) 4699.

[20] Y. Sun, T. Chen, Q. V. H. Nguyen, H. Yin, Tinyad: Memory-efficient anomaly detection for time series data in industrial iot, IEEE Transactions on Industrial Informatics (2023).

[21] P. Andrade, I. Silva, G. Signoretti, M. Silva, J. Dias, L. Marques, D. G. Costa, An unsupervised tinyml approach applied for pavement anomalies detection under the internet of intelligent vehicles, in: 2021 IEEE International Workshop on Metrology for Industry 4.0 & IoT (MetroInd4. 0&IoT), IEEE, 2021, pp. 642–647.

[22] N. L. Giménez, F. Freitag, J. Lee, H. Vandierendonck, Comparison of two microcontroller boards for on-device model training in a keyword spotting task, in: 2022 11th Mediterranean Conference on Embedded Computing (MECO), IEEE, 2022, pp. 1–4.

[23] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, S. Vassilvitskii, Scalable k-means++, arXiv preprint arXiv:1203.6402 (2012).

[24] C. C. Aggarwal, S. Y. Philip, J. Han, J. Wang, A framework for clustering evolving data streams, in: Proceedings 2003 VLDB conference, Elsevier, 2003, pp. 81–92.

[25] X. Zhang, C. Furtlehner, M. Sebag, Data streaming with affinity propagation, in: Joint european conference on machine learning and knowledge discovery in databases, Springer, 2008, pp. 628–643.

[26] C.-H. Li, W.-C. Huang, B.-C. Kuo, C.-C. Hung, A novel fuzzy weighted c-means method for image classification, International Journal of Fuzzy Systems 10 (3) (2008) 168–173.

[27] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. d. Carvalho, J. Gama, Data stream clustering: A survey, ACM Computing Surveys (CSUR) 46 (1) (2013) 1–31.

[28] T. Kohonen, The self-organizing map, Proceedings of the IEEE 78 (9) (1990) 1464–1480.

[29] B. Fritzke, A growing neural gas network learns topologies, Advances in neural information processing systems 7 (1994).

[30] F. Forest, M. Lebbah, H. Azzag, J. Lacaille, Deep embedded som: joint representation learning and self-organization, reconstruction 500 (2000) 500.

[31] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, M. Kloft, Deep one-class classification, in: ICML, PMLR, 2018, pp. 4393–4402.

[32] A. Karras, C. Karras, N. Schizas, M. Avlonitis, S. Sioutas, Automl with bayesian optimizations for big data management, Information 14 (4) (2023) 223.
[33] X. He, K. Zhao, X. Chu, Automl: A survey of the state-of-the-art, Knowledge-Based Systems 212 (2021) 106622.
[34] H. Jin, Q. Song, X. Hu, Auto-keras: An efficient neural architecture search system, in: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, 2019, pp. 1946–1956.
[35] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015).
URL `https://www.tensorflow.org/`
[36] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, F. Hutter, Efficient and robust automated machine learning, in: Advances in Neural Information Processing Systems 28 (2015), 2015, pp. 2962–2970.
[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, Journal of Machine Learning Research 12 (2011) 2825–2830.
[38] R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, I. Stoica, Tune: A research platform for distributed model selection and training, arXiv preprint arXiv:1807.05118 (2018).
[39] C. Wang, Q. Wu, M. Weimer, E. Zhu, Flaml: A fast and lightweight automl library, Proceedings of Machine Learning and Systems 3 (2021) 434–447.
[40] J. Snoek, H. Larochelle, R. P. Adams, Practical bayesian optimization of machine learning algorithms, Advances in neural information processing systems 25 (2012).
[41] K. Swersky, D. Duvenaud, J. Snoek, F. Hutter, M. A. Osborne, Raiders of the lost architecture: Kernels for bayesian optimization in conditional parameter spaces, arXiv preprint arXiv:1409.4011 (2014).
[42] D. R. Jones, M. Schonlau, W. J. Welch, Efficient global optimization of expensive black-box functions, Journal of Global optimization 13 (4) (1998) 455.
[43] F. Hutter, H. H. Hoos, K. Leyton-Brown, Sequential model-based optimization for general algorithm configuration, in: Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5, Springer, 2011, pp. 507–523.
[44] C. White, W. Neiswanger, Y. Savani, Bananas: Bayesian optimization with neural architectures for neural architecture search, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35, 2021, pp. 10293–10301.
[45] S. Rayana, Odds library, `http://odds.cs.stonybrook.edu` (2016).
[46] I. D. Katser, V. O. Kozitsin, Skoltech anomaly benchmark (skab), `https://www.kaggle.com/dsv/1693952` (2020). doi:10.34740/KAGGLE/DSV/1693952.
[47] C. Wang, Q. Wu, S. Huang, A. Saied, Economic hyperparameter optimization with blended search strategy, in: International Conference on Learning Representations, 2021.