# HiMLEdge - An Energy-Aware Optimization Framework for Hierarchical Machine Learning in Wireless Sensor Systems at the Edge

**Julio Wissing**[1*], **Stephan Scheele**[1], **Aliya Mohammed**[1], **Dorothea Kolossa**[2], **Ute Schmid**[1]

[1]Fraunhofer Institute for Integrated Circuits IIS, Erlangen Germany

[2]Ruhr-University Bochum, Bochum Germany

julio.wissing@iis.fraunhofer.de

## Abstract

Smart sensor systems are a key factor in ensuring sustainable computing by enabling machine learning algorithms to be executed at the data source. This is particularly helpful when working with moving parts or in remote areas, where no tethered deployment is possible. However, including computations directly at the measurement device places an increased load on the power budget. Therefore, we introduce the Hierarchical Machine learning framework "HiMLEdge" which enables highly specialized models that are tuned with an energy-aware multi-criteria optimization. We evaluate our framework using prognostic health management in a three-part feasibility study: First, an exhaustive search is applied to find hierarchical taxonomies, which we benchmark against hand-tuned flat classifiers. This test shows a decrease in power consumption of up to 47.63% for the hierarchical approach. Second, the search strategy is improved with Reinforcement Learning. As a novel contribution, we include real measurements in the reward function, instead of using a surrogate metric. Third, we conduct tests on a system level including communication and system-off power draw. In this scenario, the found hierarchical model is able to perform four times more readings per hour than a flat classifier while achieving the same five years of battery life.

## 1 Introduction

The advancement of artificial intelligence (AI) might be one of the biggest impact factors when it comes to achieving the United Nations' Sustainable Development Goals (SDGs) [Vinuesa *et al.*, 2020]. From combating climate change [Cowls *et al.*, 2021], monitoring and managing natural disasters, to explainable AI for health care [Bruckert *et al.*, 2020], there are many possible applications for machine learning (ML) or AI to improve on the SDGs.

Previously, the dominant data processing method followed a centralized architecture, in which sensor data is transferred to cloud resources and processed remotely using complex machine learning models. However, this approach has the downside of a high energy consumption overhead due to the excessive communication of unfiltered sensor data at a high rate. Particularly, considering that many applications include an event or anomaly detection, there is a waste of energy due to avoidable data transmissions. In recent years, the increasingly widespread adoption of machine learning in edge computing under the term "tinyML" enabled intelligent applications on resource-constrained IoT devices, such that the on-device execution of machine learning models is becoming a considerable alternative to the centralized approach of data processing [Ren *et al.*, 2021]. This allows for wireless applications that directly process data where it is produced, increasing security and omitting the need for an energy-hungry cloud infrastructure. Nonetheless, a key requirement of such wireless systems is energy efficiency, a mandatory factor to ensure long-lasting battery life of months or even years for tiny embedded sensor systems. Achieving this goal is a challenge demanding highly efficient ML-Models, which are optimized with respect to energy consumption and accuracy.

Hence, we introduce our framework HiMLEdge that makes use of *automatically optimized* hierarchical machine learning models, which represent the decision task as a classification hierarchy and allow for lazily triggered computations that only consume the energy needed. We see this technology as an enabling factor for many applications that are currently not practically solvable. Especially in remote areas with no connection to direct power (e.g., undeveloped areas, rain forests, or the sea), on device computing can be very beneficial [Schwartz *et al.*, 2021].

## 2 Related Works

Hierarchical classification has already been applied in several application scenarios [Zhou *et al.*, 2017; Pech *et al.*, 2021], often with the motivation to improve the quality of the classification results rather than to reduce energy consumption. However, previous work has shown that hierarchical classification can also be used as a partitioning method to increase the energy efficiency of a ML-model by using cascaded processing [Goetschalckx *et al.*, 2018], modularizing its classification taxonomy [Akbari *et al.*, 2018] or by distributing its subcomponents and workload across multiple edge devices [Thomas *et al.*, 2019].

While the algorithm selection problem (ASP) for a non-

*Contact Author

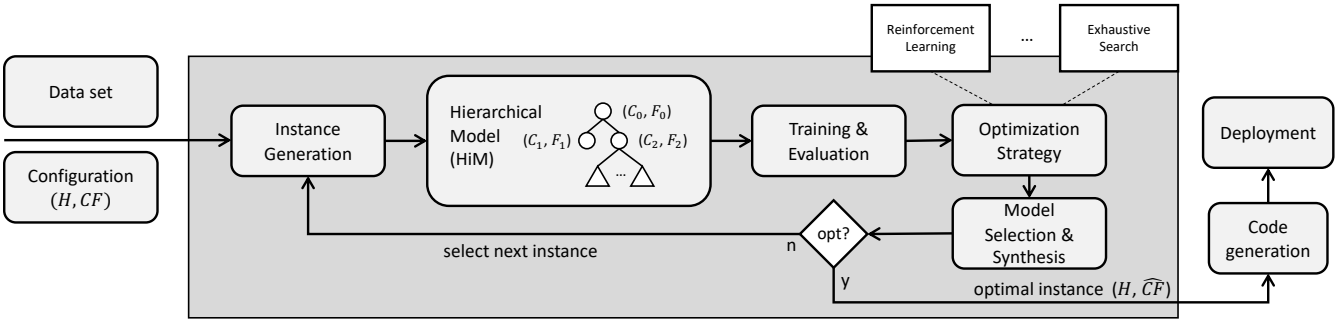



Figure 1: Visualization of the HiMLEdge-Framework.

hierarchical (flat) classifier is already a research field on its own [Bischl *et al.*, 2012], it becomes a more complex task when switching to a multi-model approach. [Adams *et al.*, 2019] solve the ASP for hierarchical taxonomies with reinforcement learning by interpreting the selection of each classifier in the hierarchy as actions. The agent then collects a reward based on accuracy *and* theoretical computational complexity. Building up on this idea, we include energy measurements during the search to verify Adams et al.'s surrogate energy metric. In contrast to previous works we also include the feature extraction in the search space to only compute necessary features at every step in the hierarchy. HiMLEdge allows us to streamline the process of training, optimizing and deploying hierarchical machine learning models towards embedded devices.

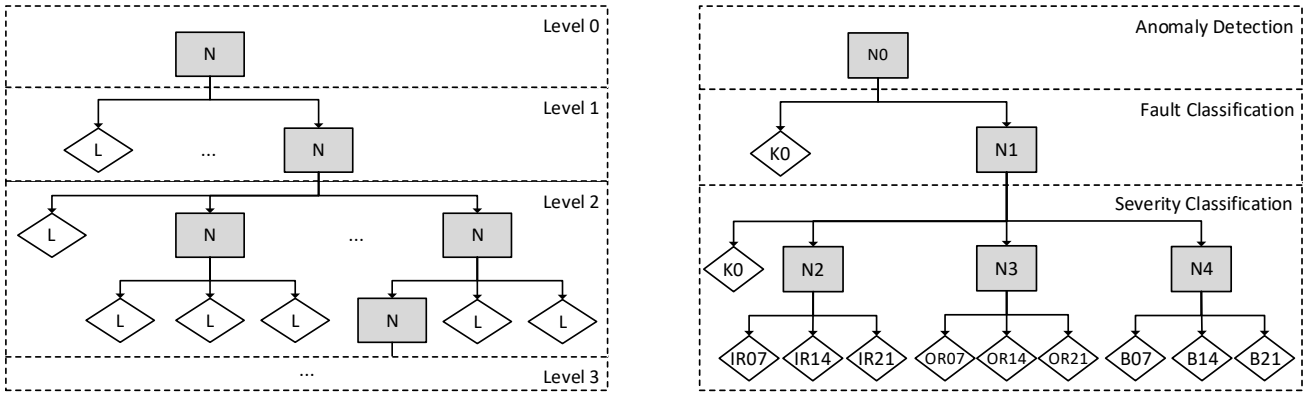
## 3 The HiMLEdge Framework

The HiMLEdge framework makes use of hierarchical machine learning to find an energy efficient pipeline. Based on a class hierarchy, the classification task is partitioned into a class taxonomy of decisions. The class taxonomy can be represented as a directed acyclic graph (DAG) or similarly a poset $(N, \prec)$, where $N$ is a finite set of nodes (one for each class) and a partial order $\prec$ over $N$ that is asymmetric, antireflexive and transitive. This structure can be simplified to a tree known from a classical hierarchy scheme, e.g., as shown in Fig. 2a, where levels or even single nodes can be represented by a unique classifier instance consisting of a feature extraction and a model solving stage. We will denote such hierarchical classifiers by *Hierarchical Model* (HiM) in the following.

Contrarily, a flat classification approach is represented by a single complex multilabel classifier, responsible for classifying all labels. [Silla and Freitas, 2011] introduce an abstract description of such hierarchical structures, describing multiple approaches of which we use the *local classifier per parent node* variant to build a HiM for energy-efficient execution. In this approach each node inside the HiM is modularised and can be represented by a different classifier instance to achieve the highest degree of flexibility. The modularity of hierarchical models has the advantages that (i) similarly to a divide-and-conquer algorithm, a complex multilabel classification problem is partitioned into simpler sub-problems that can be solved more efficiently; (ii) for each node we can select the best suitable and problem-specific classification algorithm; (iii) the node-wise modularization makes it possible to optimize individual parts of a hierarchical model, to adapt them more efficiently to changes and to scale the model to a distributed system architecture. This enables the combination of a broad variety of target platforms starting from specialized custom circuits such as neuromorphic hardware, embedded microcontrollers, or multi device networks.

The HiMLEdge framework[1] as shown in Fig. 1 is able to handle the complete process of generating, training, selecting and transpiling to C-code of a HiM for an embedded platform. In the optimization process the framework generates a HiM in a JSON-like representation out of a possible permutation of given feature sets and class taxonomies. Any HiM formulated in this representation can then be read in, trained and evaluated by the framework, building the basis for the optimization process. Training and testing of a HiM is made possible with a DAG based recursive algorithm, following the taxonomy from top to bottom. Each node is modeled to have access to its own classifier as well as feature extraction and holds references to its following node(s). In that way, each node can call the train/predict function of the next node(s) or terminate the recursion if a leaf node has been reached. For the ASP the user can choose an optimization strategy, which is able to include direct feedback of the energy consumption with real measurements of the HiM model in question, or use an approximation for the energy consumption to find an HiM without measurement hardware. Currently, the framework is able to utilize an exhaustive search approach or a reinforcement learning based optimization like shown in [Adams *et al.*, 2019], but will be extended with further techniques like e.g. Bayesian Optimization [Kurian *et al.*, 2021] or Evolutionary Algorithms [Aquino-Brítez *et al.*, 2021] in the future. To describe the classifier and feature selection steps we will use the following abbreviations: DT=Decision Tree, LR=Logistic Regression, SVM=Support Vector Machine, RF=Random Forest, MLP=Mulit-Layer-Perceptron, KNN=k-Nearest Neighbours, PTP=Peak-To-Peak, RMS=Root Mean Square, FC=Frequency Centroid, RVF=Root Variance Frequency, $F_{max}$=Maximum Frequency, and MF=Mean Frequency.

[1] https://github.com/Fraunhofer-IIS/HiMLEdge




(a) Arbitrary hierarchical taxonomy

(b) Hierachical taxonomy used for the CWRU dataset

Figure 2: Illustrations of hierarchical taxonomies in a general example (left) and applied to the CWRU dataset (right). For the applied case the labels no fault (K0), Inner Race fault (IR07-IR21), Outer Race fault (OR07-OR21) and Ball fault (B07-B21) are shown as leafs.

### 3.1 Optimization problem

We use a local classifier per parent node approach, because each classifier and feature extraction in every node of the HiM can be chosen independently. Let $\mathcal{C} = \{c_1, ..., c_j\}$ and $\mathcal{F} = \{F_1, ..., F_k\}$ be finite sets of classifier labels and feature sets respectively. The ASP can be described by the optimization problem to maximize the reward function

$$R(\hat{CF}) = \lambda \mathcal{A}(\hat{CF}) + \frac{1}{\lambda}\hat{E}(\hat{CF}), \tag{1}$$

where $\hat{CF}$ is a finite set index by $N$ that contains for each node $n \in N$ a classifier-feature pair $(\hat{C}_n, \hat{F}_n)$, which combines a single classifier $\hat{C}_n \in \mathcal{C}$ with a set of feature labels $\hat{F}_n \in \mathcal{P}(\mathcal{F})$. The reward consists of a weighted sum between the accuracy $\mathcal{A}$ and the approximated energy consumption $\hat{E}$, which may be replaced by a real measurement. The weighting factor $\lambda$ is a hyperparameter and able to shift the optimization towards energy consumption or classification performance. In our testing $\lambda$=2 showed a good trade off between both criteria, while a smaller value lead to trivial classifiers and an increased $\lambda$ to inefficient models.

### 3.2 Reinforcement Learning

As an alternative approach to the exhaustive search, we also test Reinforcement Learning (RL) to solve the ASP for a HiM. RL is a machine learning method that aims to learn the optimal sequence of actions called *policy* required to reach a specific goal. It consists of an *environment*, and an *agent* that exists in certain *states*. The agent is the main actor, that interacts with the environment by performing actions. The agent gets the feedback from the environment as a *reward* and a new *state*. The agent's goal is to maximize its reward over time. In [Adams *et al.*, 2019], the authors propose a RL method for the ASP of hierarchical classification problems. We extend this method by testing it on an embedded device and performing real-life measurements. In this context, the set of states are considered to be the different levels in the hierarchy. The set of actions are the different classification algorithms. In every episode, the agent selects a classifier type and trains it, then conducts inference on a random test instance of the dataset. Based on the generated prediction, the agent moves to the new state in the next level and collects a reward. An episode is complete when the agent reaches a terminal state. The reward function is similarly defined as in Eq. 2, with $\hat{E}(\hat{C}, \hat{F})$ being replaced by the measurement

$$R(\hat{CF}) = \lambda \mathcal{A}(\hat{CF}) - (1-\lambda)\hat{E}(\hat{CF}). \tag{2}$$

Here we use a different weighting method, where $\lambda$ is between 0 and 1 to keep compatibility with the work of Adams et al.. The complete process of training the RL-Agent to select a classifier for each node can be seen in Alg. 1. As shown in the literature, we apply the Monte-Carlo on-policy strategy, where the Q-table is updated after every episode. $X$ and $Y$ are the sets of data samples and their corresponding labels, respectively. $\epsilon$ is the parameter to balance exploitation and exploration in RL algorithms. $\alpha$ is the learning rate, and $\gamma$ is the reduction factor. In order to extend the framework with direct feedback from the embedded device, we implement a Host to embedded device communication runtime with direct feedback from the SMU (see Fig. 3). First, the Host conducts the action selection and the classifier training. It then sends the trained classifier's parameters and a test instance to the embedded device that uses the sent parameters to perform the inference. Simultaneously, the embedded device triggers a measurement of the energy consumption while inference is performed. Lastly, the Host receives the prediction from the embedded device and the energy consumption from the SMU. This information is used to update its Q-table and continues to perform a new episode with a new test instance. After several episodes the agent learns an optimal policy, which is a sequence of classifier types for every level in the hierarchy.

## 4 Feasibility study

To show the real world benefits of applying automated hierarchical machine learning, we conduct a feasibility study using the HiMLEdge framework with three experiments. We

**Algorithm 1:** Reinforcement Learning Algorithm for Hierarchical Classification

**Input :** ε, α, γ, λ, $X$, $Y$
**Output:** The optimal policy π*

1 Initialize Q-table, **for** $episode = 0...Episodes$ **do**
2  **for** $i = 0...I$ **do**
3  Select $X_k$ and $Y_k$ for hierarchy level k
4  Select testing instance $x_i$, $y_i$
5  Build training set: $X_k \setminus x_i$, $Y_i \setminus y_n$
6  Select an action type $a \in A$
7  Train classifier of type $a$ using training set
8  Predict label for $x_i$ using trained classifier
9  Move to next state $s'$ based on the prediction
10  Update Q($s$,$a$)
11  **if** $s'$ *is a leaf node* **then**
12  Break;
13  **else**
14  Return to line 3;
15  **end**
16  **for** $s \in S$ **do**
17  $\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a)$
18  **end**
19  **end**
20 **end**

start by applying an exhaustive search algorithm to find the best working combination of feature extraction and classification methods in a condition monitoring scenario for each node in a hierarchy. This gives us an overview about the complete search space and see the maximum possible reduction in theoretical energy consumption. We then proceed by benchmarking the best performing HiMs with measurements on an embedded device and compare them to flat classifieres with the same accuracy.

Afterwards, we continue by reproducing the reinforcement learning based algorithm selection introduced by [Adams *et al.*, 2019]. We run this algorithm with direct feedback from measurements instead of using a theoretical computation complexity measure, which directly influences the reward function during training. After training, we compare the found HiM with measurement feedback to a HiM found with the computational complexity approximation as shown in the literature. This should show the Reinforcement Learning Agent's ability to find well performing architectures in terms of accuracy and energy consumption and also verify whether the approximation is in line with the measurements.

As a third step we conduct tests on a system level view of energy consumption by simulating an application scenario including communication, data retrieval and classification. In that way we compare the system level energy consumption of a HiM with a flat classifier and a non classification scenario (just sending sensor data).

## 4.1 Dataset

For evaluation we chose condition monitoring as a target application, which is part of the SDG Goal 9 working towards sustainable industrialization. Therefore, we are using

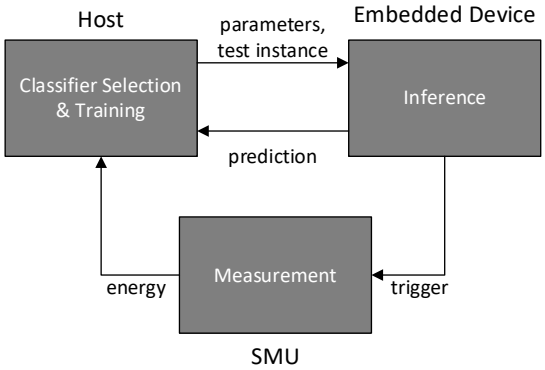



Figure 3: Communication paths during training of the reinforcement learning agent that includes measurement feedback.

the well studied CWRU-Bearing dataset [Neupane and Seok, 2020], which was recorded using a 2HP motor connected to a dynamometer via a torque transducer/encoder. This also gives us the opportunity to see the applicability of a different dataset to hierachical machine learning in comparison to [Adams *et al.*, 2019]. The bearings used in this test are supporting the motor shaft and have been artificially damaged at different locations with fault depths ranging from 0.007" to 0.021". The vibration data included in the dataset was collected using single-axis accelerometers with a sampling rate of 12kS/s (fan-end). The data is separated into windows of 512 samples, with a split of 60% training, 20% validation and, 20% test data.

## 4.2 Measurement setup

All classifiers used by the HiMLEdge framework are trained using the Scikit-Learn module in python. Before any data is evaluated, the input features are scaled to have zero mean and unit variance. For the features in the frequency space, a Fast fourier transform is used with an FFT size equal to the number of samples in a window. The calculation of the statistical features in both the time and frequency domain can be found in [Lei, 2017]. The Hyperparameters of each classifier are tuned using a grid search approach based on a validation set. After training, the selected classifiers are ported to plain C using the micromlgen[2] module and compiled using gcc with optimization (-O3) enabled.

The ported classifiers are tested on an Arduino Nano 33 BLE Sense by using a subset of windows from the testset. The energy consumption is measured with an industrial source measurement unit (PXIe-4145). To reduce the influence of background systems and sensors, we first measure the base power consumption of the microcontroller during idle. This base power is subtracted from the power consumption measured during inference, resulting in the energy consumption in Joule per inference.

## 4.3 Exhaustive Search

As first trial we use an exhaustive sesmarch algorithm to obtain a complete overview of the search space. In this way, we are able to construct a knowledge base, which can later

[2]https://github.com/eloquentarduino/micromlgen

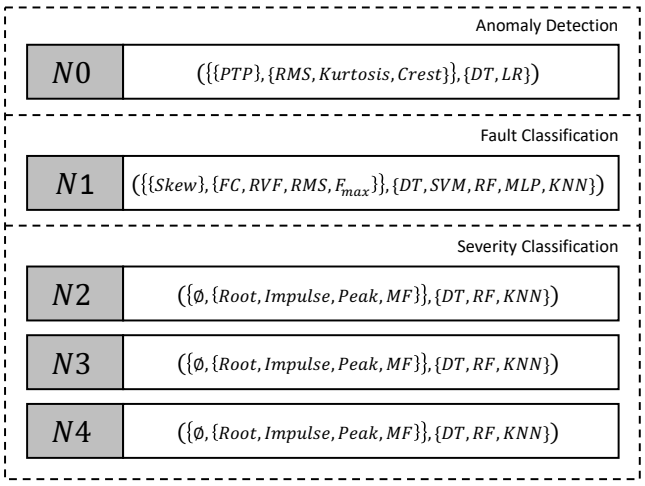



Figure 4: Possible tuple choices for each node during the exhaustive search.

be used to form heuristics for future search algorithms. We separate the classification task in three levels: Anomaly detection (i=0), fault classification (i=1) and severity detection (i=1,2,3) (cf. fig. 2b). Due to the large search space we chose to approximate the energy consumption with the mean latency per inference $\tau$ measured on the host PC, which is normalized with respect to the highest measured latency during the search. This results in

$$\hat{E}(\hat{CF}) = (1 - \frac{\tau(\hat{CF})}{\tau_{max}}). \quad (3)$$

The two best performing HiMs are transpiled to plain C and deployed to the embedded device. As baseline models, two flat classifiers (SVM and RF) are compared to the found HiMs in terms of energy consumption and accuracy. The optimization procedure picks out of a node indexed configuration $CF = \{(C_0, F_0), (C_1, F_1), \ldots\}$ where each $C_n \in \mathcal{P}(\mathcal{C}) \setminus \emptyset$ and $F_n \subseteq \mathcal{P}(\mathcal{F})$. For the hierarchy in Fig. 2b, we let the optimization pick out of the sets shown in Fig. 4, which use the following configuration $\{(C_0, F_0), (C_1, F_1), \ldots, (C_4, F_4)\}$ with

$$C_0 = \{\text{DT, LR}\},$$
$$F_0 = \{\{\text{PTP}\}, \{\text{RMS, Kurtosis, Crest}\}\},$$
$$C_1 = \{\text{DT, SVM, RF, MLP, KNN}\},$$
$$F_1 = \{\{\text{Skew}\}, \{\text{FC, RVF, RMS, F}_{\text{max}}\}\},$$
$$C_2 = C_3 = C_4 = \{\text{DT, RF, KNN}\},$$
$$F_2 = F_3 = F_4 = \{\emptyset, \{\text{Root, Impulse, Peak, MF}\},$$

leading to a total of 12964 permutations.

The results shows improvements in energy consumption for both HiMs over the baseline (cf. Fig. 5). A significantly influencing factor for the hierarchical model is the fault-detection, which is presumably responsible for most of the computations in the model. This is because of the inclusion of an FFT in the feature extraction process and the more complicated classification task. Therefore, we will continue comparing the hierarchical model with a RF at its core with the RF baseline and the SVM core with the SVM baseline.

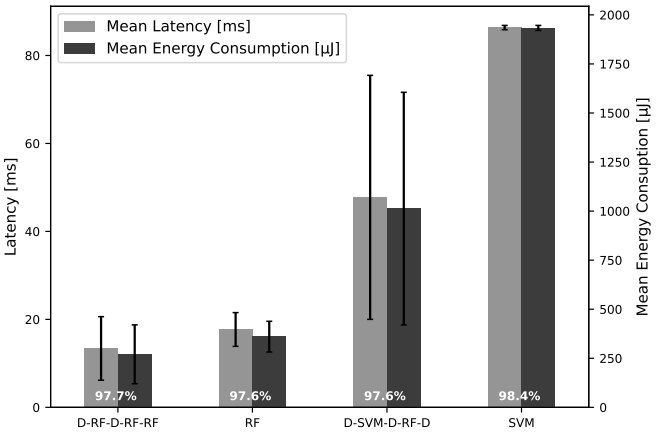



Figure 5: Measured energy per inference from the exhaustive search results.

Comparing the RF-based models, a decrease in both mean energy consumption and latency of 24.96% can be achieved using the hierarchical approach while obtaining a slight increase in accuracy of 0.1%. This improvement is most likely connected to the distribution of the dataset, where 25% of the cases are non-faulty. Here, the use of a DT with only time-domain features is enough for the decision to stop the computation. The additional overhead introduced by the hierarchical structure might be compensated by the severity detection, where in the case of the found model for some fault classes no additional features are needed.

When comparing the two SVM-based approaches, the gap between the flat classifier and the hierarchical model increases further with a decrease of 47.63% in energy consumption, but with a slightly worse accuracy of 97.6% (SVM-baseline 98.4%). This increase in energy efficiency for the hierarchical model can be explained with the complexity of a SVM, which in the worst case can be O($n^3$) [Ns, 2015]. With the higher number of input features as well as possible classes, the cubic complexity becomes a problem in this scenario.

### 4.4 Reinforcement Learning

To test the capabilites of the RL-Based selection approach we narrow down the search space to stay compareable to [Adams *et al.*, 2019]. For all nodes ($n \in \{0, 1, \ldots, 4\}$) the agent can pick out of the set $C_n = \{\text{DT, RF, MLP, SVM}\}$. Logistic regression and KNN classifiers were left out in this experiment as they did not show adequate results during the exhaustive search. Additionally, the RFs have been limited to a maximum of ten estimators due to higher memory constraints introduced by the runtime in the background. To keep the comparability with Adams et al.'s work, the features are not optimized in the progress and the same feature sets as found in Sec. 4.3 are used. With feedback from measurements the experiment resulted in the optimal policy {DT, MLP, RF, MLP, RF }, which differs compared to the policy found when using adams et al.'s approximation ({DT, RF, DT, DT, DT }). During testing we explored that even though an MLP had a much higher complexity measure in comparison to a RF,

the measurement resulted only in a slight increase in energy consumption. Because of the higher accuracy of the MLP, the agent probably decided to lean towards picking the MLP over the RF. Additionally, the training process converged after only a few hours of training on a single desktop PC, while the exhaustive search was run in the course of multiple days. This shows the importance of a suitable algorithm selection techniques that does not only save energy for the inference, but also needs less power during optimization and training. Without a proper approach, the ASP becomes unsolvable for hierarchical classification with an exhaustive search in increasing search spaces and the positive impact on sustainability becomes questionable.

### 4.5 System Level Energy Consumption

As a last test we want to evaluate a system view to get insight in the power consumption in an application scenario. Before, we always focused on optimizing the energy consumption of the machine learning pipeline isolated from the total power draw of other components and communication. Therefore, in the following we will calculate a system level energy score per hour

$$E_{sys} = T[\alpha E_{fault} + (1-\alpha)E_{normal}] + E_{off}, \quad (4)$$

where $T$ denotes the readings per hour, $\alpha$ the ratio of faulty events scaling the energy consumption in the respective cases $E_{fault}$ and $E_{normal}$, and the system-off power consumption per hour $E_{off}$. With this setting we compare three cases: A flat classifier (RF), the best performing HiM from Sec. 4.3 and sending data only. In the case of classification, the result is sent via BLE if a fault has been detected, while in the sending data case the complete window holding 512 values is always sent via BLE to the Host. The goal of this test is to find the maximum possible duty cycle $T$ to achieve 5 years of battery life with a single CR2477 coin cell battery holding 1000mAh (10800J @ 3V) of charge in the respective scenarios and thereby see the impact on energy consumption of hierarchical classification in an application scenario.

It is clearly visible that the automatically found HiM is able to achieve the best duty cycle that is able to perform 3.95 times more classifications per hour in comparison to the flat model, leading to a near real time monitoring of the machine in question. Additionally, the big gap between the two classification scenarios and the wireless sending mode shows that filtering and classifying the sensor data should always be the preferred method if applicable. Considering that in other more extreme cases BLE might not be available due to its low range, different communication methods might draw even more energy. Therefore, an efficient filtering method should be especially useful in remote areas. Additionally, when looking at this test from a battery life point of view, the more efficient algorithms could also be used to increase battery life, or decrease the battery capacity instead of applying a higher duty cycle. This perspective helps to improve sustainability in an industry 4.0 scenario or make more applications possible.

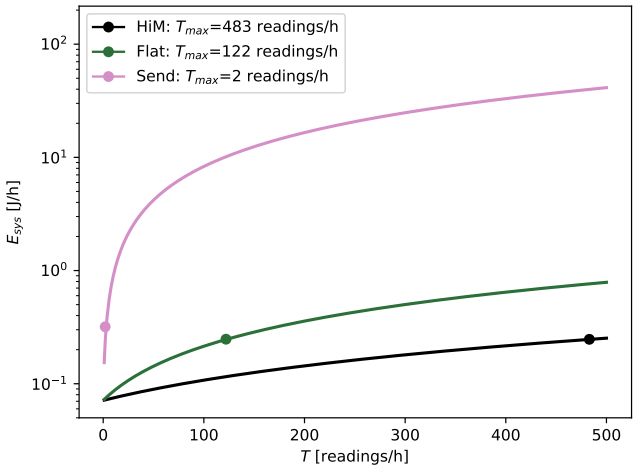



Figure 6: System level view on energy consumption. The maximum possible duty cycle for 5 years of run time are marked for the respective cases.

## 5 Discussion

With the HiMLEdge framework we were able to evaluate 12964 hierarchical model architectures to find the best performing model w.r.t. accuracy *and* energy usage. The found HiMs achieve a reduction in energy of up to 47.63% over the baseline. With lazily triggered computations, only the necessary energy is consumed during inference. These improvements are mainly linked to (i) the anomaly-detection as the first stage of evaluation and (ii) the modularized selection of features and classifiers. The exploration of RL as a selection method for HiMs shows the potential behind more efficient optimization algorithms. Also, the different policies obtained by performing measurements in comparison to an approximation demonstrates that the viability of the approximation should always be ensured to produce accurate results. The system level view on hierarchical classification exhibits the strength of efficient processing at the edge that lead to a near real time execution of the monitoring while using only a coin cell battery for five years of run time.

However, there are additional points that need to be investigated in future works. Further optimization techniques should be explored apart from like e.g. Bayesian Optimization [Kurian *et al.*, 2021] or Evolutionary Algorithms [Aquino-Brítez *et al.*, 2021]. Even though RL showed promising results, it is unclear how it would handle an increased search space. The optimization process might also be positively influenced by a surrogate model that precisely estimates the energy consumption of a pipeline [García-Martín *et al.*, 2019]. Additionally, further modularization of features could improve the energy consumption by e.g. utilizing a Multirate Filterbank [Vaidyanathan, 1990] to decompose the input signals. Furthermore, the hierarchy is constructed with expert knowledge. Therefore, an automatic approach to learn hierarchical structures should be investigated.

## References


[Adams *et al.*, 2019] Stephen Adams, Ryan Meekins, Peter A. Beling, Kevin Farinholt, Nathan Brown, Sherwood Polter, and Qing Dong. Hierarchical fault classification for resource constrained systems. *Mechanical Systems and Signal Processing*, 134:106266, December 2019.

[Akbari *et al.*, 2018] Ali Akbari, Jian Wu, Reese Grimsley, and Roozbeh Jafari. Hierarchical Signal Segmentation and Classification for Accurate Activity Recognition. In *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers*, UbiComp '18, pages 1596–1605, New York, NY, USA, October 2018. Association for Computing Machinery.

[Aquino-Brítez *et al.*, 2021] Diego Aquino-Brítez, Andrés Ortiz, Julio Ortega, Javier León, Marco Formoso, John Q. Gan, and Juan José Escobar. Optimization of Deep Architectures for EEG Signal Classification: An AutoML Approach Using Evolutionary Algorithms. *Sensors*, 21(6):2096, January 2021. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

[Bischl *et al.*, 2012] Bernd Bischl, Olaf Mersmann, Heike Trautmann, and Mike Preuß. Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In *Proceedings of the 14th annual conference on Genetic and evolutionary computation*, GECCO '12, pages 313–320, New York, NY, USA, July 2012. Association for Computing Machinery.

[Bruckert *et al.*, 2020] Sebastian Bruckert, Bettina Finzel, and Ute Schmid. The Next Generation of Medical Decision Support: A Roadmap Toward Transparent Expert Companions. *Frontiers in Artificial Intelligence*, 3, 2020.

[Cowls *et al.*, 2021] Josh Cowls, Andreas Tsamados, Mariarosaria Taddeo, and Luciano Floridi. The AI Gambit — Leveraging Artificial Intelligence to Combat Climate Change: Opportunities, Challenges, and Recommendations. SSRN Scholarly Paper ID 3804983, Social Science Research Network, Rochester, NY, March 2021.

[García-Martín *et al.*, 2019] Eva García-Martín, Crefeda Faviola Rodrigues, Graham Riley, and Håkan Grahn. Estimation of energy consumption in machine learning. *Journal of Parallel and Distributed Computing*, 134:75–88, December 2019.

[Goetschalckx *et al.*, 2018] Koen Goetschalckx, Bert Moons, Steven Lauwereins, Martin Andraud, and Marian Verhelst. Optimized Hierarchical Cascaded Processing. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(4):884–894, December 2018. Conference Name: IEEE Journal on Emerging and Selected Topics in Circuits and Systems.

[Kurian *et al.*, 2021] John Joy Kurian, Marcel Dix, Ido Amihai, Glenn Ceusters, and Ajinkya Prabhune. BOAT: A Bayesian Optimization AutoML Time-series Framework for Industrial Applications. In *2021 IEEE Seventh International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 17–24, August 2021.

[Lei, 2017] Yaguo Lei. 2 - Signal processing and feature extraction. In Yaguo Lei, editor, *Intelligent Fault Diagnosis and Remaining Useful Life Prediction of Rotating Machinery*, pages 17–66. Butterworth-Heinemann, January 2017.

[Neupane and Seok, 2020] Dhiraj Neupane and Jongwon Seok. Bearing Fault Detection and Diagnosis Using Case Western Reserve University Dataset With Deep Learning Approaches: A Review. *IEEE Access*, 8, 2020. Conference Name: IEEE Access.

[Ns, 2015] Abdiansah Ns. Time complexity analysis of support vector machines (SVM) in LibSVM. *International Journal of Computer Applications*, 128, October 2015.

[Pech *et al.*, 2021] Martin Pech, Jaroslav Vrchota, and Jiří Bednář. Predictive maintenance and intelligent sensors in smart factory: Review. *Sensors*, 21(4), 2021.

[Ren *et al.*, 2021] Haoyu Ren, Darko Anicic, and Thomas A. Runkler. The synergy of complex event processing and tiny machine learning in industrial IoT. In *Proceedings of the 15th ACM International Conference on Distributed and Event-based Systems*, DEBS '21, pages 126–135, New York, NY, USA, June 2021. Association for Computing Machinery.

[Schwartz *et al.*, 2021] Daniel Schwartz, Jonathan Michael Gomes Selman, Peter Wrege, and Andreas Paepcke. Deployment of Embedded Edge-AI for Wildlife Monitoring in Remote Regions. In *2021 20th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1035–1042, December 2021.

[Silla and Freitas, 2011] Carlos N. Silla and Alex A. Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1):31–72, January 2011.

[Thomas *et al.*, 2019] Anthony Thomas, Yunhui Guo, Yeseong Kim, Baris Aksanli, Arun Kumar, and Tajana S. Rosing. Hierarchical and distributed machine learning inference beyond the edge. In *2019 IEEE 16th International Conference on Networking, Sensing and Control (ICNSC)*, pages 18–23, 2019.

[Vaidyanathan, 1990] P.P. Vaidyanathan. Multirate digital filters, filter banks, polyphase networks, and applications: a tutorial. *Proceedings of the IEEE*, 78(1):56–93, January 1990. Conference Name: Proceedings of the IEEE.

[Vinuesa *et al.*, 2020] Ricardo Vinuesa, Hossein Azizpour, Iolanda Leite, Madeline Balaam, Virginia Dignum, Sami Domisch, Anna Felländer, Simone Daniela Langhans, Max Tegmark, and Francesco Fuso Nerini. The role of artificial intelligence in achieving the Sustainable Development Goals. *Nature Communications*, 11(1):233, January 2020. Number: 1 Publisher: Nature Publishing Group.

[Zhou *et al.*, 2017] Funa Zhou, Yulin Gao, and Chenglin Wen. A Novel Multimode Fault Classification Method Based on Deep Learning. *Journal of Control Science and Engineering*, 2017, March 2017. Publisher: Hindawi.