Accepted for publication at ACM Transactions on Cyber-Physical Systems

RockNet: Distributed Learning on Ultra-Low-Power Devices

ALEXANDER GRÄFE, Institute for Data Science in Mechanical Engineering, RWTH Aachen University, Germany
FABIAN MAGER, Networked Embedded Systems Lab, TU Darmstadt, Germany
MARCO ZIMMERLING, Networked Embedded Systems Lab, TU Darmstadt, Germany
SEBASTIAN TRIMPE, Institute for Data Science in Mechanical Engineering, RWTH Aachen University, Germany

As Machine Learning (ML) becomes integral to Cyber-Physical Systems (CPS), there is growing interest in shifting training from traditional cloud-based to on-device processing (TinyML), for example, due to privacy and latency concerns. However, CPS often comprise ultra-low-power microcontrollers, whose limited compute resources make training challenging. This paper presents RockNet, a new TinyML method tailored for ultra-low-power hardware that achieves state-of-the-art accuracy in timeseries classification, such as fault or malware detection, without requiring offline pretraining. By leveraging that CPS consist of multiple devices, we design a distributed learning method that integrates ML and wireless communication. RockNet leverages all devices for distributed training of specialized compute efficient classifiers that need minimal communication overhead for parallelization. Combined with tailored and efficient wireless multi-hop communication protocols, our approach overcomes the communication bottleneck that often occurs in distributed learning. Hardware experiments on a testbed with 20 ultra-low-power devices demonstrate RockNet's effectiveness. It successfully learns timeseries classification tasks from scratch, surpassing the accuracy of the latest approach for neural network microcontroller training by up to 2x. RockNet's distributed ML architecture reduces memory, latency and energy consumption per device by up to 90 % when scaling from one central device to 20 devices. Our results show that a tight integration of distributed ML, distributed computing, and communication enables, for the first time, training on ultra-low-power hardware with state-of-the-art accuracy.

CCS Concepts: • **Computer systems organization → Sensor networks**.

Additional Key Words and Phrases: Distributed Learning, TinyML, Wireless Networks

1 Introduction

Machine Learning (ML) is a key component of Cyber-Physical Systems (CPS) [2, 22, 32, 43, 53], enabling them to adapt to new situations, extract valuable insights from their data, and increase their overall efficiency. Take smart devices such as cordless power tools in manufacturing [3] as an example. To minimize downtimes, the tools should be able to predict their wear and tear for predictive maintenance [64]. ML enables this by collecting data from all tools in the factory and extracting accurate wear models. Another example are wearable devices equipped with multiple inertial sensors [4, 76] that analyze gaits to evaluate orthoses, prosthetics, surgical procedures [9] or to detect diseases [78]. Here, ML creates models that accurately analyze gait patterns using data from multiple users [50, 78]. Other applications where ML enhances CPS include forest observation or wildfire detection, production, healthcare, smart cities, homes and power grids [2, 22, 43, 53, 56, 80, 85].

In the context of ML in CPS, it is crucial to differentiate between inference–the execution of a trained model–and online learning, which trains the model at runtime. Unlike inference, learning enables systems to adjust to changing operating conditions [46, 61]. Thus, learning provides a crucial foundation for flexible and self-adapting CPS and is a focus of this work.

There exist two ways to execute learning: *inside* a CPS on its devices or *outside* on a cloud server, see, e.g., the surveys [2, 21, 22, 62]. The strength of a cloud server lies in its significantly greater compute power compared

Authors' Contact Information: Alexander Gräfe, alexander.graefe@dsme.rwth-aachen.de, Institute for Data Science in Mechanical Engineering, RWTH Aachen University, Aachen, Germany; Fabian Mager, Networked Embedded Systems Lab, TU Darmstadt, Darmstadt, Germany; Marco Zimmerling, Networked Embedded Systems Lab, TU Darmstadt, Darmstadt, Germany; Sebastian Trimpe, Institute for Data Science in Mechanical Engineering, RWTH Aachen University, RWTH Aachen University, Germany.

to the CPS devices [21, 22, 62]. On the other hand, cloud processing comes with increased latency due to the communication between the CPS and the cloud [21, 22, 53, 62], overloads existing Internet communication infrastructure due to the rapidly increasing device numbers [11, 40], and raises privacy concerns from transmitting data outside the CPS [21, 40, 43, 62].

Given the relevance of these issues across various applications, there is an increasing interest in implementing learning *inside* CPS [21, 53]. Yet, many of the mentioned applications share the common feature of utilizing low-cost, ultra-low-power embedded devices, such as those found in smart sensors. These have severely constrained compute resources that are challenging for ML [21, 44, 66]. The research field aiming to address learning under such limited resources is known as TinyML. However, TinyML often sacrifices learning accuracy for resource efficiency [21], leading to suboptimal performance when running on ultra-low-power hardware. Additionally, many works on TinyML focus on the efficient execution of *already trained* models, whereas we investigate *actual learning*, a less explored TinyML branch [46]. To summarize, we ask: *Can we design a TinyML method for CPS that achieves state-of-the-art accuracy while learning on ultra-low-power hardware?*

1.1 Problem Setting

Abstracting from the aforementioned application examples, we consider scenarios consisting of N ultra-low-power devices together forming one CPS, like multiple tools in a smart factory or multiple smart devices in a smart home. Each device i holds local data $\mathcal{D}_i$ that it has collected. Each entry in $\mathcal{D}_i$ consists of a datapoint x with its class label c. The data contains sensitive information and should not be shared outside the CPS, making cloud processing not possible. However, contrasting federated learning scenarios, it is allowed to share data between devices as they lie in the same CPS.

For instance, in the aforementioned factory setting, each power tool collects and stores timeseries sensor data, such as accelerometer readings [28]. While the data from the tools should not leave the factory to not reveal details about production, it is safe to share between the tools. The same holds for a smart home. The data should not leave the home to not reveal information about its residents, but can be shared among devices inside the smart home.

Goal of the devices is to predict the class label of a given datapoint x. For this, the devices train a classifier $\hat{c} = h_W(x)$ on this data, where W are its trainable parameters and $\hat{c}$ is the predicted class. The goal of the learning process is to determine parameters W that minimize:

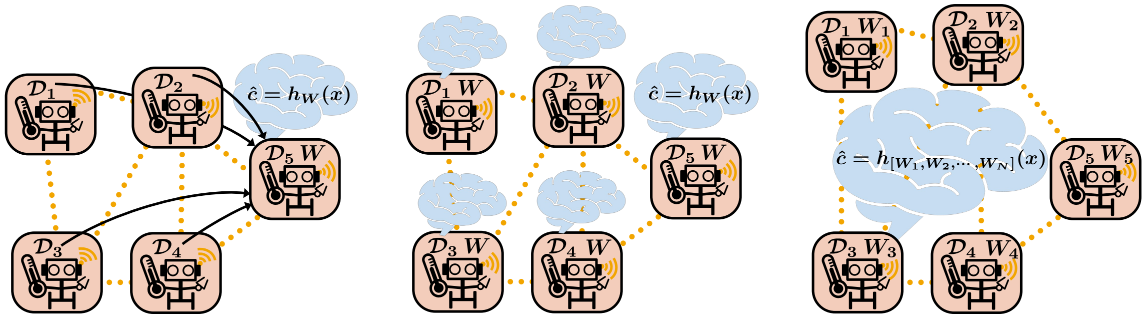
$$L(W) = \sum_{i=1}^{N} \sum_{(x,c)\in\mathcal{D}_i} \ell(h_W(x), c), \tag{1}$$

where ℓ is a suitable loss function, e.g., the cross-entropy loss [5]. The loss L incorporates the data of all devices. This is crucial for the model to generalize effectively across diverse data distributions, e.g., to handle local data shifts.

As the classifier must perform well on the combined data, the devices must communicate. For this, they use wireless communication to increase flexibility and cost efficiency compared to wired communication [3]. In many CPS applications, like large factories or underground mining, the devices have to cover large areas [3]. Thus, not every device may have a direct link to each other device. Devices with no direct link can communicate using devices in-between as relays (multi-hop communication). When the devices are mobile, like wearables or robot swarms, the network topology changes continuously.

Such a setting poses two major challenges:

- **C1 ML on Ultra-low-power Devices.** Ultra-low-power devices are limited in energy, compute power and memory. This significantly hinders the training process, as limited compute power and energy cause delays.



(a) Central TinyML (b) Federated TinyML (c) Split TinyML

Fig. 1. Central versus Federated versus Split TinyML. *Central TinyML: All data is sent to one device that runs the ML algorithm. Federated TinyML: Each device trains a model on its local data; these models are repeatedly shared and merged. Split TinyML: Devices collaboratively run the ML algorithm, each holding portions of the parameters W.*

Furthermore, the available memory may be insufficient to store and train the entire classifier, which is directly linked to accuracy, as usually larger models lead to higher accuracies.

C2 ML with Low-power Communication. To accommodate limited device energy, we must rely on low-power communication. However, this choice inherently restricts bandwidth and slows data exchange, thus impeding the learning process—a challenge that is further exacerbated by multi-hop communication.

Problem Statement. In summary, our objective is to develop a classifier h_W and a learning algorithm that yields parameters W minimizing loss (1) from distributed data $\mathcal{D}_i$. The classifier and learning algorithm shall be implemented inside the CPS on the devices communicating via a wireless multi-hop network while effectively addressing challenges **C1** and **C2**.

1.2 Contributions

For this problem, three fundamental approaches are conceivable (Fig. 1):

(1) **Central TinyML.** All devices transmit their data to one central device that executes the entire learning.
(2) **Federated TinyML.** Every device trains the entire model locally on its data. The devices repeatedly share and merge their models [24, 27, 29, 38, 39, 46, 89].
(3) **Split TinyML.** The devices split model execution and training among themselves. Every device i holds parts W_i of the classifier's parameters W [24, 37, 57, 77, 82].

Central TinyML's main limitation is its reliance on single-device execution, which inherently restricts resource availability. Consequently, models are confined in size and accuracy, and the training process is slowed due to these resource constraints.

To overcome these single-device limitations, distributed learning techniques such as federated learning and split learning pool resources of multiple devices within the CPS. Distributed learning has been highlighted as a key focus area for future research, as emphasized in “Control for Societal-scale Challenges: Road Map 2030” [1].

While the terms "federated learning" and "split learning" are sometimes used interchangeably, we adopt the definitions provided by Gao et al. [24] and Thapa et al. [77]. **Federated learning** employs data parallelism: each device trains the entire model on its local data and shares the resulting model with other devices. In contrast, **split learning** utilizes model parallelism, where each device trains different segments of the model and shares intermediate results.

Federated learning accelerates model training by enabling devices to train in parallel [83]. However, each device must possess sufficient resources—particularly memory—to handle local training of the entire model [77]. This requirement limits federated TinyML to smaller models, often resulting in reduced accuracy.

By distributing compute and memory demands across multiple devices, split learning overcomes these limitations, enabling the training of larger, more accurate models. However, despite these inherent advantages of split TinyML and extensive research on central and federated TinyML in recent years [7, 13, 14, 19, 24, 27, 29, 35, 44, 59, 61, 71, 88], achieving split TinyML on ultra-low-power hardware remains an open challenge.

Motivated by this, we present RockNet, the first split TinyML method that achieves state-of-the-art accuracy while training solely and from scratch on multiple ultra-low-power devices *without pretraining*. RockNet is a method for timeseries classification tasks common in CPS [22, 43, 53], e.g., anomaly detection, malware detection, fault detection and activity classification [28, 43, 53].

Core idea behind RockNet is its integration of ML and communication concepts. Specifically, RockNet builds on two main ideas:

(1) **ROCKET Classifiers Enable Efficient Split Learning:** RockNet builds on *Random Convolutional Kernel Transform* (ROCKET) classifiers [15], which achieve state-of-the-art accuracy in timeseries classification. Although these classifiers are known for their resource efficiency, they remain too demanding for a single ultra-low-power device. However, by employing an appropriate parallelization strategy, they can be efficiently distributed among multiple devices reducing the per-device resource consumption to a level suitable for typical ultra-low-power hardware (**C1**). Importantly, this parallelization strategy incurs minimal communication overhead, allowing us to meet communication constraints (**C2**).

(2) **The Mixer Protocol Supports the Parallelization of ROCKET Classifiers.** The second key ingredient of RockNet is the communication protocol Mixer [30]. This protocol is capable of bridging the gap between the parallelization strategy and multi-hop communication. It efficiently implements all-to-all communication, ensures precise network-wide synchronization, and guarantees reliable message exchange during training, all essential for our parallelization strategy. Furthermore, its optimal scalability with the number of devices enables effective utilization of communication resources (**C2**).

RockNet combines these two methods, ROCKET and Mixer, into a cohesive framework. In a hardware experiment with up to 20 devices (nRF52840, ARM Cortex M4, 64 MHz, 256 kB RAM) communicating via the physical layer of Bluetooth Low Energy, we demonstrate that RockNet enables ultra-low-power split TinyML for the first time. RockNet's accuracy significantly surpasses the accuracy of AIfES, a recent central TinyML approach for neural network (NN) training [88] and significantly reduces memory, latency, and energy consumption when scaling from few to many devices.

In summary, RockNet addresses the challenges associated with high-accuracy learning in ultra-low-power networks. It achieves training of high-accuracy classifiers on ultra-low-power devices and advances the state-of-the-art in split TinyML. It makes the following contributions:

(1) RockNet is the first TinyML method to achieve state-of-the-art accuracy in timeseries classification while training on ultra-low-power hardware. We enable this by integrating communication and ML research.
(2) RockNet is the first split learning method for decentralized ultra-low-power hardware communicating via a low-power wireless multi-hop network. It suits the conditions in many CPS scenarios as it does not rely on specific network topologies and supports moving devices.
(3) In experiments on real wireless hardware, we demonstrate the first learning of a non-linear timeseries classifier with state-of-the-art accuracy on ultra-low-power devices without offline pretraining. RockNet's accuracy is significantly larger than central NN training. When scaling from one to 20 devices, RockNet reduces memory per device by up to 93 %, latency by up to 89 % and energy per device by up to 86 %.

Table 1. Comparison of RockNet to related work on TinyML.

←Hardware→	Central TinyML	Distributed TinyML: Federated TinyML	Distributed TinyML: Split TinyML
Powerful Embedded Hardware	[52, 60, 70]	[24, 29, 87] [33, 51, 72]	[24, 25]
Ultra-low-power Hardware	[7, 13, 14] [19, 35, 41, 44] [59, 61, 71, 88] [41, 73]	[27]	[69], RockNet

←Methodology→

2 Related Work

RockNet is an online learning method. Therefore, we focus the discussion on related works that also performs actual training. We structure the discussion as shown in Table 1: We review works on central and distributed TinyML (Section 2.1), then discuss TinyML hardware implementations in Section 2.2 and at the end discuss networked solutions for distributed ML 2.3. We focus on the high-level concepts behind these methods and details that are relevant to our work.

2.1 TinyML Methodologies

2.1.1 Central TinyML. There are two main strategies for central TinyML. The first focuses on fine-tuning pretrained neural networks (NN), either by training the last layer [19, 61], tuning parts of hidden layers [7, 44], or by quantization and pruning [59]. However, fine-tuning requires good prior knowledge of the data the devices will encounter. Without this, especially in high privacy scenarios, we fine-tune out-of-distribution, which can work well in some but may not achieve good accuracy in other cases [75, 84].

The second strategy involves learning without pretraining [13, 14, 35, 52, 71, 88] resulting in suboptimal accuracy due to resource constraints. Some works focus on training linear classifiers only [35, 71, 73], and others train NNs [13, 14, 52, 88] [41] that must be small and thus have low accuracy.

RockNet enables training a classifier without pretraining, setting it apart from the first central TinyML strategy. Moreover, unlike the second strategy, by utilizing special resource-efficient classifiers in conjunction with split learning, we achieve high accuracy. Illustrating this, Section 4 compares RockNet to AIfES [88], a central TinyML method for NN training.

2.1.2 Distributed TinyML. As explained in Section 1.2, distributed learning has two branches: Federated and split learning.

Federated TinyML. Federated learning involves each device running the entire model on its local data. The devices repeatedly merge their models, e.g., by averaging parameters [24, 27, 29, 38, 39, 46, 89]. As a result, federated learning preserves privacy between devices by not sharing local data. A disadvantage of federated learning is that each device must run the ML pipeline for the entire model, which is challenging for resource-constrained devices [77].

In our setup, exchanging local data between devices does not raise significant privacy concerns (cf. Section 1.1); therefore, the primary advantage of federated learning is less pronounced. Consequently, its disadvantage, the requirement for each device to train the entire model, outweigh its benefits. Federated learning thus faces similar limitations as central TinyML, resulting in reduced accuracy when implemented on ultra-low-power hardware.

Split TinyML. With split learning, each device calculates different parts of the model, which effectively reduces memory utilization and compute load [24, 25, 34, 69, 77, 82]. However, split learning needs significant communication [25], which can negate the benefits of parallelism by increasing training times and energy usage. In

RockNet, we leverage split learning to reduce resource consumption per device. By considering communication costs during the design of ML and utilizing an efficient communication protocol, we can effectively reduce the latency and energy of communication.

2.2 Learning on Ultra-low-power Hardware

Investigating online learning on real hardware is essential, as this allows us to operate with realistic hardware constraints, shows what is feasible with state-of-the-art algorithms and gives us true real-world performance. Here, we focus on ultra-low-power hardware, specifically microcontrollers. This sets us apart from studies that explore learning on more powerful embedded hardware like Raspberry Pis or smartphones [24, 25, 29, 33, 51, 52, 69, 70] or microcontrollers with specialized tensor cores [60] (Table 1). These, although resource-constrained, still have significantly more resources than the ultra-low-power hardware we consider. Current TinyML methods for such ultra-low-power hardware focus on central [7, 19, 35, 44, 61, 71, 88] and federated TinyML [27]. Although these methods perform well within their setups, they suffer from the inherent issues of central and federated learning, as outlined above. Split learning can solve these issues. The only existing split learning framework for ultra-low-power devices is Globe2Train [69], which trains multiple base linear classifiers across multiple devices. However, Globe2Train relies on a central server to aggregate and distribute the training, whereas RockNet eliminates the need for any server-based orchestration by operating in a fully decentralized manner. Additionally, Globe2Train uses internet-based communication, while RockNet operates with purely ultra-low-power wireless communication.

To date, RockNet is the first work to bring decentralized split learning to ultra-low-power devices including ultra-low-power wireless communication. We show that by leveraging split learning on constrained hardware, RockNet achieves state-of-the-art accuracy in a variety of setups, thus expanding the possibilities of high-accuracy machine learning for this class of devices. To highlight the performance benefits, we compare RockNet against AIfES[88], a state-of-the-art solution for training neural networks on a single ultra-low-power device, and demonstrate that RockNet significantly outperforms it in accuracy.

2.3 Communication Solutions for Distributed Learning

A key component of distributed learning, whether federated or split, is the wireless communication between devices. Most existing approaches rely on high-power methods such as WiFi via a central router [24, 29] or WiFi mesh protocols [87]. These solutions not only consume far more power than what ultra-low-power hardware can support (which typically uses BLE or IEEE 802.15.4) but also require bandwidth levels that such hardware cannot provide.

The only current ultra-low-power communication solution for distributed learning was proposed by Gimenez et al. [27] for ultra-low-power federated learning. Their approach employs a LoRa mesh network with a peer-to-peer protocol, though it was evaluated only on three devices with one-hop communication.

In contrast, our solution supports low-power multi-hop communication and our evaluation demonstrates a true multi-hop network with up to 20 devices. Rather than relying on peer-to-peer exchanges, our method implements efficient round-based all-to-all communication, closely resembling the allgather and allreduce schemes common in many ML parallelization strategies [20, 90].

3 RockNet: Distributed Learning on Ultra-low-power Devices

This section introduces RockNet. We designed it to be capable of learning timeseries classification over a wireless network of ultra-low-power devices. First, we provide an overview of our solution, followed by a detailed explanation of its components. Afterwards, we explain how we optimize RockNet's resource consumption and provide a formal analysis of its resource consumption.

3.1 Overview

RockNet enables learning in CPS solving the problem defined in Section 1.1, especially under **C1** and **C2**. Based on the analysis of related work and the problem setting, we employ split learning to pool the constrained device resources. This approach overcomes the limitations of individual devices since each device only needs to store and compute parts of the model. Our split learning design follows the following strategy.

First, we use a suitable ML method that meets the following properties:

- **P1 Resource Efficiency and High Accuracy.** Although we increase available compute resources through split learning, the combined resources are still severely constraint (**C1**). The ML method must account for this and needs to be resource efficient, while yielding highly accurate classification results.
- **P2 Efficiently Parallelizable.** As we split the ML method among devices, it needs to be parallelizable. The parallelization must be communication efficient to minimize overhead.

Subsequently, we establish a parallelization strategy for the machine learning method. Ultimately, we implement this strategy on distributed hardware communicating over a wireless multi-hop network. To achieve this, we present a specialized, reliable, and efficient communication protocol.

Following this strategy, we use *Random Convolutional Kernel Transform classifiers* (ROCKET) [15] as ML method, because it fulfills **P1** and **P2** as we explain in the following. After designing a parallelization method for ROCKET, we derive requirements on the communication protocol. To fulfill these requirements, we exploit recent advances in ultra-low-power wireless networking by building on a communication protocol called Mixer [30] (details in Section 3.4). Mixer's efficient and reliable all-to-all data exchange is ideal for distributing ROCKET, where input data and intermediate results must be shared among devices. Additionally, Mixer provides network-wide time synchronization, which we exploit to efficiently parallelize ROCKET's compute.

3.2 Foundation: ROCKET Classifiers

ROCKET [15] combines efficiency with state-of-the-art accuracy (**P1**). In recent years, multiple derivatives of ROCKET emerged, like MiniROCKET [16], MultiROCKET [74] and HYDRA [17], which we summarize all under the term ROCKET in the following. ROCKET algorithms have proven to be very successful in various non-linear timeseries classification tasks. First, the algorithms have shown competitive accuracy to other state-of-the-art classifiers like NNs, HIVE-COTE [45], TS-CHIEF [67] and ResNet on the UCR archive [10, 12], which contains 128 toy and real-world problems like ECG classification. Thus, we refer to ROCKET as achieving state-of-the-art accuracy for timeseries classification. Since RockNet represents an exact distributed implementation of ROCKET, it inherits ROCKET's state-of-the-art accuracy. This enables us to combine ultra-low-power learning on distributed hardware with cutting-edge performance for the first time.

In recent years, ROCKET has been successfully used in different applications like breast cancer classification [58], depression detection [8], flight performance analysis of pilots [54], human activity recognition [6], astronomical seeing prediction [55], activity classification for power tools [28], and fault prediction for automated teller machines [81].

ROCKET not only achieves higher accuracy but is also significantly more computing efficient than state-of-the-art classifiers such as HIVE-COTE [45], TS-CHIEF [67], and ResNet. The high computing demands of these methods have hindered the realization of high-accuracy time series classification on ultra-low-power hardware. We demonstrate that ROCKET, when realized via RockNet, overcomes these limitations.

ROCKET first transforms a timeseries into a very large feature space ($\sim 1 \cdot 10^5$ features). Afterward, a linear classifier in this feature space classifies the timeseries. The feature space transformation is based on convolutions with randomly sampled kernels. After each convolution, ROCKET extracts features from the resulting timeseries, using the proportion of positive values (PPV) [15, 16, 74] or a dictionary method [17]. We can thus write its total behavior as

$$\begin{aligned}\hat{c} = h_W(x) &= \text{softmax}(Wf) \\ &= \text{softmax}(W\left[1, g(k_1 * x), g(k_2 * x), \ldots, g(k_{M-1} * x)\right]^T),\end{aligned} \tag{2}$$

where $\hat{c} \in [0,1]^{n_c \times 1}$ is the predicted class probability with n_c being the number of classes, $f \in \mathbb{R}^{M\times 1}$ is the feature vector with length F, M is the number of features, $k_i \in \mathbb{R}^{K_i}$ are the kernels of the convolution $k_i * x$ with length K_i and padding P_i, $x \in \mathbb{R}^T$ is the input timeseries with length T and g is the feature extraction function, generating a scalar out of a timeseries (e.g., PPV). The kernels k_i are sampled before training and remain constant. Therefore, ROCKET only needs to learn the linear classifier's weights W, e.g., using cross-entropy loss and gradient descent [15], making the training very efficient.

ROCKET thus offers the high accuracy and computing efficiency RockNet needs (**P1**). However, bringing training of ROCKET to ultra-low-power hardware is still an open challenge due to its large feature space. The features, weights and optimizer consume up to thousands of kilobytes of RAM, too much for common ultra-low-power devices.

To date, only inference of ROCKET is achieved on ultra-low-power hardware [28]. However, to achieve this, the approach reduces the number of features from 10,000 to 336 to fit in RAM and performs learning offline.

In contrast, RockNet can run ROCKET completely on ultra-low-power devices, solving **C1**, considering not only inference but also *training* of ROCKET on ultra-low-power hardware for the first time. It achieves this through split learning, distributing features and weights onto multiple devices. For this, RockNet exploits that we can efficiently parallelize ROCKET (**P2**) as we now demonstrate.

3.3 RockNet: Learning Architecture

The goal of RockNet is to perform the exact computations as ROCKET to achieve identical accuracy. We split ROCKET along the feature dimension. Each device i calculates different features f_i in f (Fig. 2)

$$f = \left[f_1^T, f_2^T, \ldots, f_N^T\right]^T. \tag{3}$$

Two strategies exist to partition the linear classifier: input and output partitioning [68]. With output partitioning, devices first share their feature vectors, multiply them with the corresponding values from W, and then exchange the resulting products. This process requires transferring the entire feature vectors (e.g., 10,000 float values) and performing a second communication step, leading to excessive overhead.

In contrast, input partitioning allows each device to multiply its features by the corresponding values from W locally before exchanging the partial results and summing them. As a result, we need to exchange only Nn_c values. For example, with 20 devices and 10 classes, this approach reduces communication by a factor of 50 compared to output partitioning. The only trade-off is a redundant calculation of the sum of the partial results on every device, which requires an additional $(N-1)n_c$ float additions per device. However, these few hundred extra operations are far less expensive than the communication overhead of output partitioning. We hence partition the linear classifier using input partitioning. Each device i is assigned a portion W_i of the weights W

$$W = \left[W_1, W_2, \ldots, W_N\right]. \tag{4}$$

This reduces overall computing time and decreases RAM usage since each device has to store fewer features and weights.

All devices process one timeseries simultaneously. First, they predict the class $\hat{c}$ of the timeseries x (forward pass). Then, they calculate the gradient of the cross-entropy loss between their prediction and the true class label c with respect to the weights W (backward pass). The devices accumulate the gradient over multiple timeseries and update the weights once the batch size is reached, using ADAM [36]. Evaluation steps can be performed

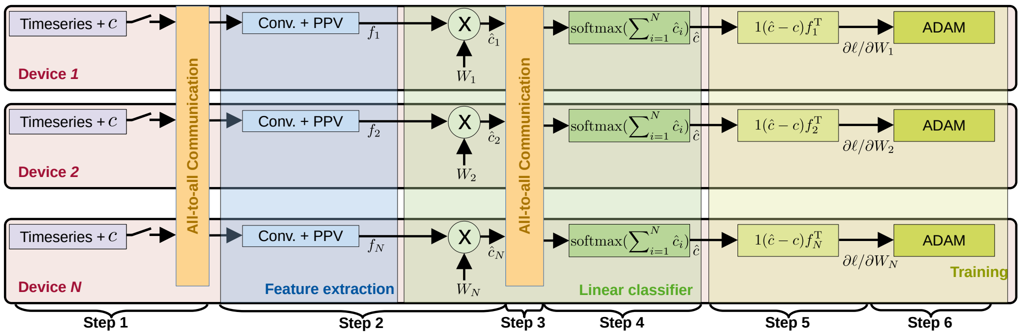


Fig. 2. Overview of RockNet. *During the forward pass (feature extraction + linear classifier), all devices exchange their data with each other. The training step does not require any additional communication, making RockNet more efficient.*

between training steps to stop early and thus avoid overfitting. In detail, all devices perform the following steps simultaneously (see Figure 2).

Step 1: Measuring Data. One device j sends its timeseries x with class c from its local data $\mathcal{D}_j$ to all devices. [1]

Step 2: Feature Extraction and Local Computation. Each device i extracts its assigned features f_i from the timeseries and multiplies these features f_i with its portion of W_i.

Step 3: All-to-all Exchange of Results. The results $\hat{c}_i = W_i f_i$ are exchanged via all-to-all communication, where all devices send their $\hat{c}_i$ values and receive the $\hat{c}_i$ values from all other devices.

Step 4: Summation of Results. Each device sums the received values and applies the softmax operation to obtain class probabilities for the input x. As a result, each device knows the predicted class c. RockNet performs the same computations as ROCKET:

$$\begin{aligned}\mathrm{softmax}(\sum_{i=1}^{N} W_i f_i) &= \mathrm{softmax}\big(\left[W_1, W_2, \ldots, W_M\right] \left[f_1^T, f_2^T, \ldots, f_N^T\right]^T \big) \\ &= \mathrm{softmax}(W \left[1, g(k_1 * x), g(k_2 * x), \ldots, g(k_{M-1} * x)\right]^T).\end{aligned}$$

Step 5: Backward Pass. Up to this step, the devices have completed the forward pass, i.e., inference. At this point, each device i already has all the information needed to perform the backward step without additional communication, making the parallelized training process very communication efficient (**P2**). Each device calculates the gradient of the cross-entropy loss l between its prediction $\hat{c}$ and the true value c with respect to its weights W_i

$$\frac{\partial}{\partial W_i} l(\hat{c}, c) = (\hat{c} - c) f_i^T. \tag{5}$$

Step 6: Update Step. After accumulating gradients over multiple timeseries, each device i updates its weights W_i using ADAM [36]. To achieve this, the first and second moment estimates of the associated weights must be retained in memory. To minimize memory usage, we apply quantization as outlined in Section 3.5.2. Since each device possesses all necessary information for an ADAM update of its respective weights, it can execute this operation locally. Consequently, Steps 5 and 6 collectively conduct an exact gradient descent via ADAM, ensuring that all theoretical insights and empirical strategies of ADAM remain applicable.

To summarize, RockNet can efficiently distribute the training of ROCKET without much communication (**P2**). It communicates the timeseries, usually a few hundred of bytes, and the parts of the linear classifier $\hat{c}_i$, a few tens of bytes, for the forward pass. For the backward pass, RockNet needs no additional communication.

[1]The original ROCKET methods only support univariate timeseries. While we stick with the original univariate setting for reasons of comparability, RockNet in principle also supports multivariate extensions of ROCKET.

3.4 RockNet: Wireless Communication Support

The RockNet parallelization strategy demands fully connected and synchronized communication, a requirement that sharply contrasts with the sparse and dynamic characteristics of the multi-hop network used for device communication. To reconcile these differences, we utilize a tailored multi-hop communication protocol that must meet the following requirements:

- **R1 All-to-All Data Exchange.** Each device must transmit its data and the results to all other devices (Steps 1 and 3), necessitating all-to-all data exchange.
- **R2 Reliable and Efficient Communication.** Data exchange must be highly reliable with negligible message losses, as any lost message hinders learning progress. Although we efficiently parallelize ROCKET, the data exchange must also be efficient to close the communication bottleneck further.
- **R3 Synchronized Execution.** RockNet's parallelization strategy requires a globally synchronized execution of computation and communication to minimize waiting times and accelerate the learning process.

Meeting requirements **R1**, **R2**, and **R3** are challenging as wireless communication is notoriously unreliable because signals are exposed to environmental disturbances like interference and fading effects. The severely limited communication bandwidth and energy availability (**C2**) further add to these challenges. These issues are further amplified by the sparsely connected topology of the multi-hop network, while device mobility introduces a dynamic network structure that reinforces these challenges.

To address these key challenges and satisfy **R1**–**R3**, RockNet leverages a state-of-the-art low-power wireless communication protocol called Mixer [30]. Mixer combines synchronous transmissions [23] and random linear network coding [31], enabling more efficient and reliable data exchange compared to traditional wireless protocols [65, 91]. The effectiveness of similar synchronous transmissions-based protocols has already been demonstrated for various real-world applications, including wireless feedback control with dependability and real-time guarantees [47, 79].

Mixer organizes its operation into discrete rounds of deterministic and constant duration, each of which is subdivided into multiple slots synchronized with a precision of a few microseconds. During any given slot, a device operates in either transmit or receive mode, with the decision governed by heuristics (for further details, see [30]). When transmitting, a device sends a linear combination of its own message and the data received from other devices in previous slots, a process known as network coding [31]. Each device stores the data it receives in memory, up to the number of transmitted messages. This stored data represents linear combinations of the messages originally sent by all devices. By solving the resulting system of linear equations at the end of the round, the device can decode every original message, thus realizing an all-to-all communication pattern (**R1**).

When multiple neighboring devices transmit simultaneously, a receiving device exploits the capture effect [42], which is the underlying principle of synchronous transmission techniques [23]. Owing to this effect, the receiver can extract the signal with the highest strength with high probability, thereby eliminating the need for complex scheduling mechanisms to avoid simultaneous transmissions. This reliance on the capture effect hence significantly reduces communication overhead making communication very efficient.

Achieving this benefit, however, requires that slots be synchronized with an accuracy of a few microseconds, a challenging task due to inevitable clock drifts. Mixer effectively overcomes this challenge by enabling each receiving device to adjust its clock "on the fly" to that of the transmitting device using a phase-locked loop that uses the precise reception time of the signal. This dynamic synchronization satisfies **R3** efficiently.

This combination of synchronous transmissions and network coding leads to a very high performance and reliability of Mixer (**R2**). In experimental evaluations, Mixer exhibited a message loss rate below 0.01 % even under conditions of rapid device movement [30]. Furthermore, Mixer achieves order-optimal scaling of $O(N+T)$, where T scales linearly with the message size, thereby ensuring minimal communication latencies.

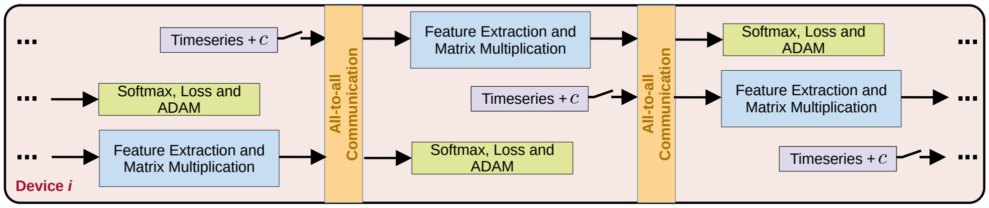


Fig. 3. Pipelining of RockNet. *In our implementation, we process three timeseries simultaneously to reduce the overhead induced by communication.*

This brief overview only scratches the surface of Mixer's internal mechanisms—numerous other aspects contribute to its beneficial properties. For an in-depth explanation, we refer to the original Mixer paper[30].

Additional to all of this, Mixer has another benefit. It spares the split learning side of RockNet from having to manage the intricate dynamics of wireless multi-hop communication. Mixer functions like a conductor, synchronizing all devices into rounds of all-to-all communication (**R1** and **R3**) adding an abstraction layer over the multi-hop network.

Within this abstraction layer, communication via Mixer emulates a wired bus that also provides a global clock signal for all devices. Synchronized by this clock, all devices simultaneously write their messages to the bus, and after a fixed, deterministic latency, every device receives all the messages. Following this, each device is allocated a constant duration to perform its computations. Once this computing phase concludes, the devices write their data to the bus again, and the cycle continues. This schedule aligns exactly with the timing requirements of our parallelization strategy (cf. Fig. 2).

Overall, Mixer's combination of synchronous transmissions and network coding offers communication properties that are well-suited for the dynamic nature of various CPS in general and for the specific requirements of split learning in RockNet in particular.

3.5 RockNet: Optimizing the Execution

Despite RockNet's inherent resource efficiency, we further enhance it through pipelining and quantization.

3.5.1 Pipelining. Mixer's communication latency has a constant offset due to overhead like synchronization and computings, independent of data size. During one RockNet round, this overhead would occur twice (in Steps 1 and 3), which is particularly significant in Step 3, where the transmitted messages are small (only one 32 bit floating point value per class). To speed up RockNet, we use pipelining by combining the execution of Step 1 for the next data, the execution of Steps 2 and 3 for the current data and the execution of Steps 4, 5 and 6 for the previous data into a single round (Figure 3). This merges the communication rounds of Steps 1 and 3 into one. Thus, we need only $n + 1$ instead of $2n$ communication rounds for n timeseries, reducing latency caused by communication overhead.

3.5.2 Quantization. With quantization of data, we aim to reduce both communication latency and memory.
Quantization of Timeseries. To reduce communication latency, we minimize the amount of transmitted data by quantizing 32 bit floating point values to 8 bit integers. We specifically focus on quantizing the timeseries, which constitute a significant portion of the transmitted data, measuring in hundreds of bytes, compared to only four bytes per class consumed by $\hat{c}_i$. The quantization uniformly maps the range of values onto 8 bit [26, 86], efficiently reducing the amount of data transmitted by up to 75 %.
Quantization of ADAM. ADAM, used for training, consumes three times the memory of the weights: once for the gradients and twice for its internal states (first and second-order momentum), significantly contributing to memory consumption. Reducing the overall memory consumption of RockNet by about 30 %, we follow Dettmers et al. [18] and quantize ADAM's internal states to 8 bit.

8-bit ADAM uses a non-uniform quantization that provides a higher accuracy for small values than uniform quantization. Additionally, it uses block-wise quantization. The state is split into blocks (here, of size 256), for which its own scaling is calculated. This reduces quantization errors and also memory during the execution of ADAM, as we can buffer it into chunks of 256 values.

3.6 Analysis of Resource Consumption

This section provides a formal analysis of the resource consumption associated with our approach, focusing on three key metrics: the amount of memory used, the number of floating point operations performed, and the volume of data transmitted per round. In the subsequent Section 4, we experimentally assess resource consumption, particularly in terms of latency and energy usage, through hardware experiments.

3.6.1 Memory. The memory consumption (RAM) of a device during each round is quantified in bytes, under the assumption that the number of features F is divisible by the number of devices N. It can be expressed as follows

$$\text{memory} = \underbrace{4n_c\frac{F}{N}}_{\text{Weight}} + \underbrace{4n_c\frac{F}{N}}_{\text{Gradient}} + \underbrace{2n_c\frac{F}{N}}_{\text{ADAM states}} + \underbrace{4\frac{F}{N}}_{\text{Features}} + \underbrace{T}_{\text{Timeseries}} + \underbrace{C_{\text{memory}}}_{\text{Rest}}, \tag{6}$$

where C_{memory} represents overhead components such as smaller buffers and intermediate variables, which in our implementation amount to approximately 25 kB. Consequently, the complexity of memory consumption regarding number of devices is characterized by $O(\frac{1}{N} + T + C_{\text{memory}})$. As a result, increasing the network size results in a decrease in memory that converges to a constant overhead.

3.6.2 Floating Point Operations. For floating point operations, we do not differentiate between operations such as addition, multiplication, multiply-and-accumulate, and division. Operations performed during the Mixer phase are excluded from consideration as they are difficult to assess. The number of operations per round is given by the following expression (assuming for simplicity that all convolutions have the same length K and padding P)

$$\begin{aligned}\text{ops} = {} & \underbrace{K\frac{F-K+2P+1}{N}}_{\text{Convolution}} + \underbrace{\frac{F-K+2P+1}{N}}_{\text{Feature Extraction}} + \underbrace{n_c\frac{F}{N}}_{\text{Weight Multiplication}} + \underbrace{c_{\exp}n_c + 2n_c - 1}_{\text{Softmax}} \\ & + \underbrace{n_c + n_c\frac{F}{N}}_{\text{Gradient}} + \underbrace{(14 + c_{\text{sqrt}})n_c\frac{F}{N}}_{\textit{ADAM}} + \underbrace{C_{\text{ops}}}_{\text{Rest}}\end{aligned} \tag{7}$$

where $c_{\exp}$ and c_{sqrt} are floating operations per calculation required for executing a single exponential or square-root function, respectively, and C_{ops} accounts minor overheads. Thus, the complexity of floating point operations concerning the number of devices is analogous to memory consumption and can be expressed as $O(\frac{1}{N} + C)$. Increasing the network size hence reduces the computational load.

3.6.3 Communication. We evaluate the data communication as the sum of data each device intends to transmit. Due to the Mixer mechanism, where each device combines messages and transmits multiple times across various slots (see Section 3.4), the volume of data sent via the wireless channel is higher. The precise relationship between these factors is challenging to determine, as it depends on the configuration of the multi-hop network (cf. the analysis of Hermann et al. [30]). To address this, we perform an empirical analysis in Section 4.3.2.

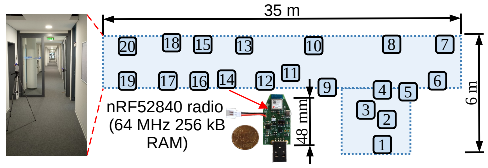


Fig. 4. Overview of our testbed. *It consists of up to 20 ultra-low-power Bluetooth radios forming a two-hop network.*

The amount of data communicated is in byte

$$\text{data} = \underbrace{T}_{\text{Timeseries}} + \underbrace{4 n_c N}_{\text{Prediction}} . \tag{8}$$

Thus, its complexity is $O(N + T)$. Consequently, the amount of data increases steadily with each additional device, in addition to a constant offset.

This section has provided an analytical examination of memory complexity, floating-point operations, and communication. In the following section, we will illustrate how these properties apply to real hardware implementations.

4 Experimental Results

This section studies the performance and efficiency of RockNet using simulations and measurements from a wireless testbed with 20 ultra-low-power devices. Our key findings are as follows:

- RockNet can train ROCKET classifiers on severe memory constraint ultra-low-power hardware. While running ROCKET on just one device would exceed its RAM by 428 %, RockNet running on 20 devices utilizes 37.2 % of the RAM and reduces latency and per-device energy by up to 89 % compared to ROCKET running on one device. This is the first demonstration of training of SOTA timeseries classifiers on ultra-low-power hardware.
- RockNet's accuracy surpasses central on-device NN training [88] over time in 78.51 % of runs on the UCR archive with a mean absolute accuracy improvement of 12.7 %.
- There is a price to pay for the improved accuracy: Longer training times (up to 86×) and higher energy consumption (up to 57×) compared to central on-device NN training [88].
- RockNet effectively pools the resources of multiple devices. Training times and RAM utilization decrease by up to 65.0 % and 55 % when RockNet scales from five to 20 devices.

4.1 Methodology

Testbed. Figure 4 shows our testbed and the positions of the 20 nRF52840 devices distributed across a hallway and an adjacent office. Each device has 256 kB of RAM and a 32-bit ARM Cortex-M4 ultra-low-power microcontroller running at 64 MHz. Devices transmit at 8 dBm using Bluetooth Low Energy radios in a genuine multi-hop network with at least two hops. Interference from co-located Wi-Fi networks, a microwave, and people cater to realistic wireless conditions.

Datasets. We train non-linear classifiers using labeled datasets from the popular UCR archive [10, 12] containing 128 timeseries classification problems. In our simulations, we use all 128 datasets. For our testbed experiments, we select 3 representative CPS datasets from the UCR archive to keep the running time of the experiments manageable: *OSULeaf* (nature observation, 6 classes), *ElectricDevices* (smart grids, 7 classes), and *FaceAll* (face detection, 13 classes).

Datasets are shuffled and partitioned among devices, leading to identically and independently distributed (i.i.d.) local data and, consequently, i.i.d. batches during training. While this is a simplification, it closely mirrors real-world data distributions in RockNet. Devices collect data over an extended time horizon and store it locally in $\mathcal{D}_i$. By randomly selecting entries from $\mathcal{D}_i$ for transmission and subsequent training, the data stream from each device becomes i.i.d. Moreover, as each device transmits periodically (in a round-robin fashion), the batches incorporate data from all devices and are thus also approximately i.i.d. We investigated this in Appendix A demonstrating that non-i.i.d. data does not reduce RockNet's accuracy.

Metrics. We measure performance in terms of *accuracy*, the percentage of correctly classified timeseries when assessing the trained models on the test datasets. To quantify efficiency, we measure *memory consumption* as the amount of RAM used, *latency* as the required training time, and *energy consumption* as the energy consumed during training. We measure energy during training by logging the times a device computes, communicates, and spends in a low-power mode and multiplying these times by the corresponding average power draws.

Acknowledging the limited memory and energy budgets inherent in our setting, we report the per-device memory and energy consumption. Analyzing how these resources scale, particularly with respect to the number of participating devices, provides valuable insights into the conditions under which the investigated algorithms are inside the constraints and become feasible.

Approaches. Our RockNet implementation uses MiniROCKET, the most memory and computing efficient ROCKET variant that achieves the same accuracy as the original ROCKET [16].

We compare RockNet to two state-of-the-art baselines:

- *AIfES* is a very recent TinyML approach for NN training on a single ultra-low-power device without pretraining [88]. To ensure a fair comparison, we tune AIfES's NN to use as much as possible of the 256 kB of RAM available on the nRF52840 without causing divergence issues for any of the 128 datasets. We thus conduct all experiments using ten hidden layers with 17 neurons each, which consumes 74 % of the available RAM. As for RockNet, the ADAM optimizer is used to train the NN. *Remark:* The network architecture is constrained by what is currently supported by AIfES. In its original publications [15, 16, 16], ROCKET has been shown to outperform more complex architectures such as ResNet while requiring far fewer computing resources. Therefore, if full neural network training on MCUs were feasible, ROCKET would be expected to show a performance advantage. *Remark:* The network architecture is constrained by the current limitations of microcontroller neural network training. In its original publications [15, 16, 16], ROCKET has been shown to outperform more complex architectures like ResNet while requiring far fewer computing resources. Therefore, if full neural network training on MCUs were feasible, it is expected that ROCKET would show a performance advantage.
- *Central ROCKET* refers to running MiniROCKET on a single ultra-low-power device. Because of RAM limitations, however, central ROCKET is not feasible on the nRF52840 (cf. Section 4.3.2). To still be able to compare against this state-of-the-art baseline, we estimate all metrics based on RockNet's measurements. Since RockNet is an exact distributed realization of ROCKET, both approaches achieve the same accuracy. To determine central ROCKET's efficiency, we scale the computing resources of RockNet by the number of devices.

For both baselines, we neglect the time and energy needed to communicate the local data $\mathcal{D}_i$ to the central device for training (see Figure 1). This makes our comparisons favorable to the baselines.

Hyperparameters. We intentionally did not perform a dedicated hyperparameter search for RockNet, as it targets ultra-low-power hardware. In this setting, executing multiple training cycles for hyperparameter tuning is compute expensive and may exceed available budget constraints, thus necessitating the reliance on a default learning rate. Consequently, all our experiments are carried out using a learning rate of 1×10^{-3}, which is consistent with the learning rate employed in the original ROCKET papers. Additionally, we fixed the batch size at 128.

The code used to generate the results of our experiments can be found in github.com/Data-Science-in-Mechanical-Engineering/RockNet.

(a) 32 bit ADAM

(b) 8 bit ADAM

Fig. 5. Accuracy of RockNet versus central NN training with AIfES. *RockNet achieves a higher accuracy than AIfES in more than* 78 % *of the runs with* 32 bit *ADAM and in* 67 % *of the runs with* 8 bit *ADAM.*

Table 2. RockNet versus AIfES in three datasets from the UCR archive [12].

Dataset	Method	Maximum test accuracy	Time to maximum accuracy	Energy per device to maximum accuracy	Time to surpass AIfES	Energy per device to surpass AIfES
OSULeaf	AIfES	35.5 %	0.074 h	2.970 J	-	-
	RockNet	92.1 %	6.282 h	144.52 J	0.306 h	8.983 J
ElectricDevices	AIfES	51.5 %	0.100 h	4.007 J	-	-
	RockNet	67.5 %	7.565 h	230.816 J	0.265 h	6.615 J
FaceAll	AIfES	47.5 %	0.132 h	5.286 J	-	-
	RockNet	72.5 %	4.963 h	119.454 J	0.652 h	14.622 J

4.2 Accuracy

We begin by comparing the accuracy of RockNet to AIfES. Note that RockNet and central ROCKET have the same accuracy.

Settings. We train AIfES and RockNet in simulation on the entire UCR archive using ten random seeds. We randomly initialize the weights and shuffle the training and evaluation splits randomly for each seed. We simulate RockNet with 32 bit ADAM and with 8 bit ADAM. The training runs for 1000 epochs, with the model from the epoch with the highest evaluation accuracy being selected for each seed. These models are then evaluated on the test datasets.

Results. Figure 5 plots, for each seed and test dataset, the accuracy of RockNet against the accuracy of AIfES. We find that for 32 bit ADAM (Figure 5a), RockNet achieves a higher accuracy than AIfES in 78.5 % of the runs and a mean accuracy improvement of 12.7 %. For 8 bit ADAM (Figure 5b), RockNet outperforms AIfES in 67 % of the runs with a mean accuracy improvement of 6.5 %. While 8 bit ADAM saves about 30 % of RAM compared to 32 bit ADAM, it leads to an average drop in accuracy of 4.2 % in 88.3 % of the runs.

These results support our design decision to use ROCKET as the base ML technique of RockNet. Thanks to RockNet's split learning architecture, it outperforms state-of-the-art TinyML for NNs on ultra-low-power hardware. The choice between 32 bit ADAM and 8 bit ADAM is a trade-off between memory consumption and accuracy. While 30 % less memory is already significant, we show in the next section that orders of magnitude higher per-device memory savings are enabled by RockNet *without* sacrificing on accuracy.

4.3 Efficiency

This section evaluates the efficiency of the learning process in terms of memory consumption, latency, and energy consumption.

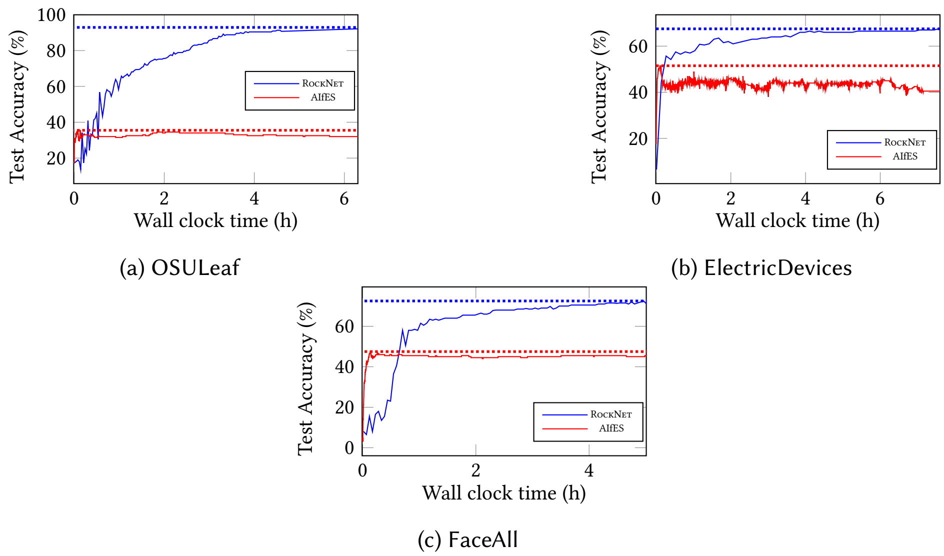


Fig. 6. RockNet training on three different datasets compared to on-device NN training (AIfES [88]). *Dashed lines show maximum accuracy reached.*

Settings. We run experiments on our 20-node wireless testbed (Figure 4). We first compare RockNet to AIfES on the three selected datasets: OSULeaf, ElectricDevices, and FaceAll. Then, we analyze RockNet's scaling behavior with the number of devices. For all three datasets, RockNet uses 8 bit ADAM, which achieves the same accuracy on these specific datasets as 32 bit ADAM while reducing memory by about 30 %. However, this reduction in memory consumption comes with a 26 % increase in latency and energy consumption due to additional computations induced by the 8-bit dynamic tree quantization, which the hardware of the processor does not support. This increase does not influence the general scaling behavior, so we only show results for 8 bit ADAM in Section 4.3.2.

4.3.1 RockNet versus AIfES. Figure 6 and Table 2 compare RockNet running on 20 devices with AIfES running on a single device. We observe that RockNet significantly outperforms AIfES in accuracy. For example, on the OSULeaf dataset, RockNet achieves an accuracy of 92.9 %, whereas AIfES achieves only an accuracy of 39 %. At the same time, RockNet needs only 55 kB of RAM per device, whereas AIfES needs more than three times as much memory. The higher accuracy of RockNet requires a longer convergence time, ranging between 5 h and 7.6 h for the three datasets. This also entails higher energy costs: RockNet consumes up to 57× more energy per device until it converges and 5× more energy until its accuracy exceeds the accuracy of AIfES.

Nevertheless, despite these costs, RockNet is suitable for scenarios where learning is an infrequent activity. Consider, for example, the power tool scenario illustrated above. As long as the factory's processes remain unchanged, relearning the model is unnecessary. Only when the factory begins producing a new product, the model may become inaccurate and require retraining. Since product cycles span months to years, RockNet is a viable solution. Waiting a few more hours for a model with a significantly higher accuracy enables better decision-making and higher resource savings later on.

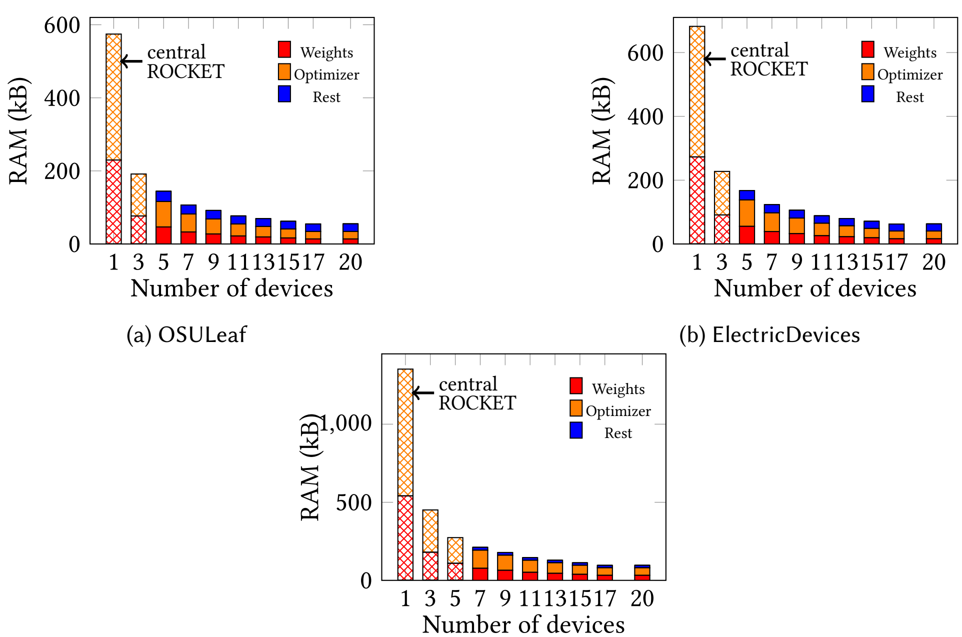
4.3.2 Scaling Behavior. The following experiments take a closer look at RockNet's performance on the number of devices.

Memory (Figure 7). Central ROCKET consumes up to 1352 kB of RAM, which exceeds the available 256 kB per device by 428 %. Therefore, training and inferencing MiniROCKET on a single device is not feasible. RockNet's distributed execution enables ROCKET for ultra-low-power hardware. Depending on the number of classes, at least 5 to 7 devices are required for MiniROCKET.

The memory consumption falls approximately inversely proportional with the number of devices ($\sim 1/N$) as derived in Section 3.6.1. Initially, there is a significant reduction in memory usage with a low number of devices, which then becomes more gradual as the number of devices increases. For example, when running RockNet on five devices for ElectricDevices, the per-device memory consumption is reduced by 75.5 % compared to central ROCKET. As a result, RockNet on five devices utilizes only 65.4 % of the available memory. Further, scaling up to 20 devices reduces memory consumption to just 24.6 % of the available memory, which is a relative reduction of 90 % compared to central ROCKET. The largest memory consumer is the ADAM optimizer (66 % for 32 bit and 48 % for 8 bit ADAM). The weights themselves are the second-largest memory consumers (32 %), followed by the features and other program code (20 %).

Since each device only has 256 kB of RAM, training and inferencing MiniROCKET on a single device, i.e., central ROCKET, is not feasible. Depending on the number of classes, at least 5 to 7 devices are required for MiniROCKET. Thus, RockNet's distributed execution enables ROCKET for ultra-low-power hardware.

As RockNet's memory consumption falls with the number of devices, we can reduce per-device memory by including more devices in RockNet. This allows us to adapt the memory consumption to the devices' available memory budget, e.g., when parts of the memory are occupied by other applications.



(a) OSULeaf

(b) ElectricDevices

(c) FaceAll

Fig. 7. Per device memory consumption of RockNet regarding network size. *Crosshatched bars show predictions for configurations that would require more than 256 KB of available RAM per device, thus exceeding the hardware limits. The first bar (1 device) corresponds to central ROCKET. It significantly exceeds the available memory. Split learning via RockNet enables ROCKET on this hardware, with the memory consumption falling with the number of devices.*

Latency (Figure 8). We measured the maximum latency of RockNet per step as the number of devices increases measured over 500 training and 500 evaluation steps. The latency first decreases drastically for a low number of devices and only slightly for a higher number of devices. For FaceAll, for example, computing latency scales from 5 to 15 devices with a factor of around 2.76 and to 20 devices with a factor of 3.3. The gaps to "perfect" scaling (i.e., factor 3 and 4) are caused by overhead every device has to do, like processing, preparing communication messages and the calculation of softmax scores. In absolute values, scaling from 5 to 15 devices reduces computing latency by 269.59 ms and by 294.39 ms for 20 devices. On the other side, communication latency stays almost constant at 100 ms, with an increase of only 3 % or 3 ms from 5 to 20 devices. Consequently, when scaling from five to 20 devices, the overall latency reduces by 55 %.

Especially compared to central ROCKET, RockNet significantly decreases latency. For instance, in our complete training run for ElectricDevices discussed in Section 4.3.1, central ROCKET would take more than two days to converge. In contrast, RockNet with five devices speeds up training by 2.3×, reducing it to 22 h. Finally, using 20 devices with RockNet, convergence is achieved in just 7.5 h, representing a 6.9× speedup. This improvement stems primarily from a significant reduction in computing latency, which dropped from 1215.99 ms to 72.99 ms (16.6×) compared to central ROCKET. This reduction is counteracted by additional 100 ms of communication overhead, which amounts to roughly 58 % of the total latency for 20 devices. However, as it is significantly lower than the speedup in computing, RockNet still achieves an overall speedup of 6.9×.

The reason for this behavior is that the compute time decreases almost inversely ($\sim 1/N$) with the number of devices (cf. Section 3.6.2). Therefore, it decreases significantly with a low number of devices and only slightly with a high number. On the other hand, communication time increases only slightly due to Mixer's order-optimal scaling behavior [30]. Thus, RockNet efficiently pools compute power across multiple devices, significantly speeding up the training process as more devices are added.

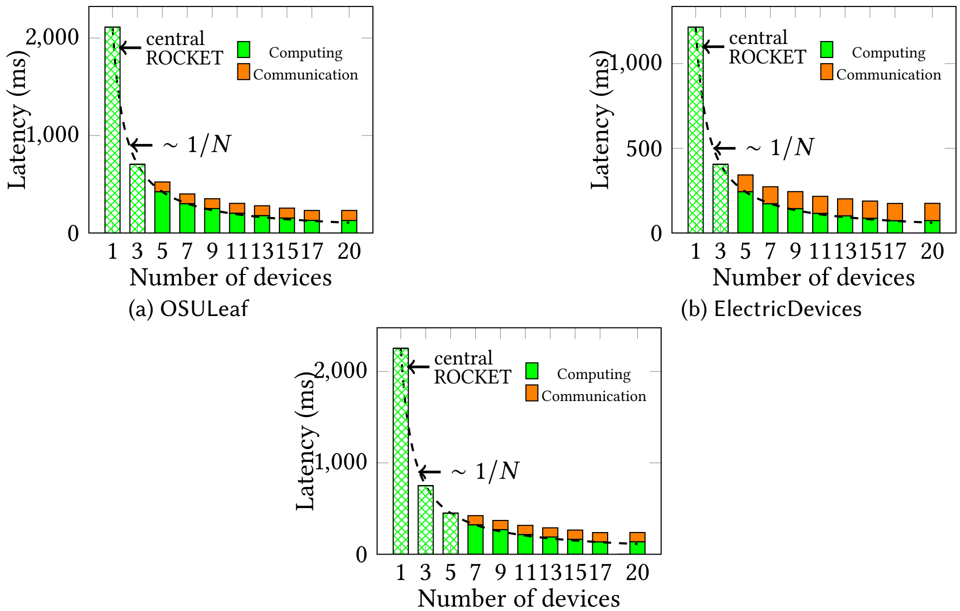
Energy (Figure 9). Figure 9 shows the mean energy consumption per training step as the device number increases measured over 500 training and 500 evaluation steps. We measured mean consumption as it is the most relevant one for e.g., battery powered devices.

We can see that initially the energy falls drastically at the beginning, reaches a minimum, and then grows slightly. For OSULeaf, the minimum is reached at 20 devices, for ElectricDevices at 7 and for FaceAll at 15. The energy required for compute decreases roughly in inverse proportion to the number of devices ($\sim 1/N$), aside from a small overhead (see latency). For instance, on the ElectricDevices dataset, scaling from 5 to 15 devices reduces per-device energy consumption by about 2.77× and from 5 to 20 devices by 3.29×. At the same time, total energy consumption per round follows an affine trend, rising by around 0.05 J per device, from 0.42 J for 5 devices to 1.2 J for 20 devices (a factor of 2.86×). At 20 devices, communication becomes the main contributor, accounting for 81.96 % of the total energy consumption.

Compared to central ROCKET, RockNet significantly reduces per-device energy consumption. For example, in the case of ElectricDevices, mean energy consumption falls by 73.6 % when moving from central ROCKET to RockNet to the minimum at seven devices. However, as the number of devices increases to 20, energy consumption grows by 28 %, leading to a reduction in energy consumption of 66.2 %. This behavior is caused by energy for compute decreasing approximately inversely with the number of devices ($\sim 1/N$), while Mixer's energy consumption scales approximately linearly with the number of devices (cf. Section 3.6).

5 Discussion

Our evaluation demonstrates that RockNet effectively pools the resources of multiple devices to perform learning. Utilizing more devices reduces the compute load and energy consumption on each device.



(a) OSULeaf

(b) ElectricDevices

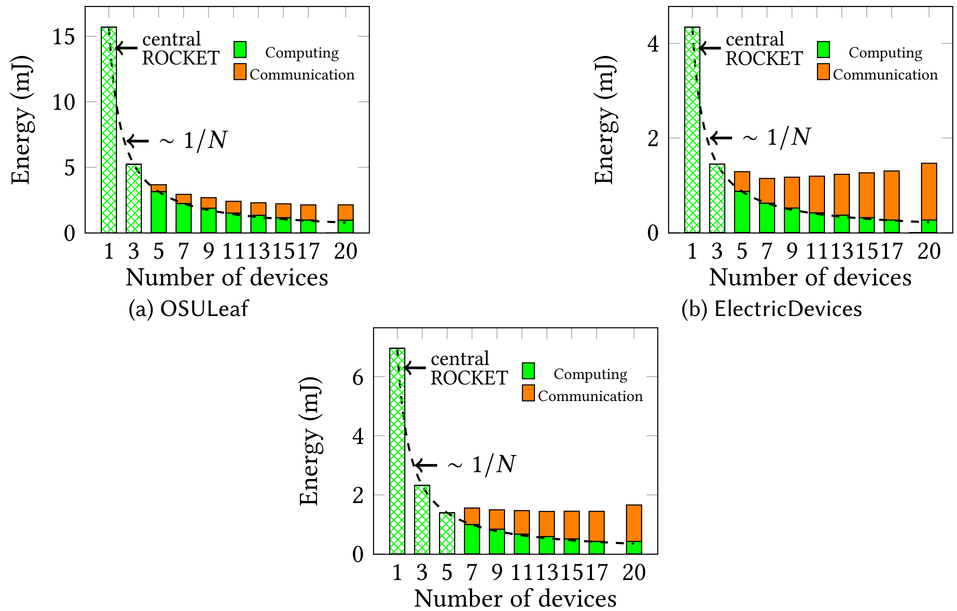
(c) FaceAll

Fig. 8. Latency of RockNet per training step. *The latency for computing falls inversely proportional with number of devices, while the communication's latency only grows slightly. As a result, the latency first falls drastically for a low number of devices and then more gradually for a higher number.*

Our experiments compared RockNet to two central ML approaches: AIfES and central ROCKET. RockNet achieves a significantly higher accuracy than AIfES. Conversely, AIfES consumes fewer overall resources, particularly in terms of energy and latency. However, as discussed in Section 4.3.1, the increased energy consumption and latency of RockNet is justified in scenarios where a model is learned once and then used for a longer time.

RockNet significantly reduces latency, memory usage and per-device energy compared to the execution of ROCKET on a single device (assuming the device is equipped with enough RAM), As more devices are added, these per-device resource consumption is reduced until a limit is reached. In our BLE-based hardware implementation, we reach this point for latency at approximately 20 devices, where we achieve a maximum reduction of 89 % relative to a single-device ROCKET execution for the FaceAll dataset. Beyond 20 devices, latency begins to rise. Regarding energy consumption, this point is reached earlier, between 7 and 15 devices. This behavior is expected when dividing a fixed-size computing task. While the per-device workload decreases proportional to $1/N$, the communication overhead increases linearly (cf. Section 3.6), resulting in diminishing absolute gains, and at some point, in an increase in resource consumption. The minimum point of energy consumption is reached much earlier than for latency, as power drawn from our chip during receiving/transmitting is around 3.67× higher compared to computing.

Another noteworthy observation is that, while communication latency via Mixer remains nearly constant, energy consumption continues to rise. This occurs because Mixer automatically places devices in energy-saving mode whenever possible. Consequently, sending smaller amounts of data increases the share of power-saving slots relative to RX/TX slots. Conversely, transferring larger amounts of data leaves fewer opportunities for energy-saving intervals, thus increasing overall energy consumption.



(a) OSULeaf (b) ElectricDevices (c) FaceAll

Fig. 9. Energy consumption of RockNet per device and per training step. *We measured mean energy consumption, the most important measure for battery powered devices, for 500 training and 500 inference steps. The energy for computing falls inversely proportional with number of devices, while the communication's energy consumption grows linearly. As a result, for a specific number of devices, the energy consumption is minimal.*

We demonstrated in our experiments that split learning across multiple low-power devices is feasible. In this way, we avoid the need of a central device. In other application (see Sec. 1.2), a central device equipped with more powerful hardware might be preferred. Typically, stronger hardware is more energy-efficient (more floating point operations per Watts) and faster. In practice, one must evaluate whether using such a device is economically feasible–essentially determining whether the benefits of faster and more energy-efficient training justify the cost of the stronger hardware.

6 Limitations

This section addresses the limitations of our work and proposes potential directions for future research. Our current focus is on time-series classification, where ROCKET stands as a state-of-the-art method. Given that RockNet is directly built upon ROCKET, its applicability however remains confined to this specific problem domain.

While time-series classification holds significant importance for CPS [22, 43, 53], tasks such as image classification also represent significant research directions, to which ROCKET and consequently RockNet are not applicable directly. To overcome this limitation, our approach could be adapted to fine-tune the final layer of pre-trained NN models. As a last layer functions as a linear classifier on features extracted from earlier layers, the entire training procedure of RockNet (Steps 3–6 in Figure 2) can be utilized. In this scenario, devices would share the computational load of the earlier layers, akin to how RockNet distributes the feature extraction process in ROCKET (Steps 1 and 2 in Figure 2). For convolutional NNs, this can be accomplished by leveraging existing methods for distributed inference [48, 49, 63].

It is important to recognize that fine-tuning only the last layer of NNs results in lower accuracy compared to fine-tuning earlier layers, as demonstrated by Lin et al. [44]. Our current approach does not support this level of training. Consequently, exploring how multiple cooperating devices can fine-tune hidden layers of NNs, a process that would require communication during the backward pass, represents an interesting direction for future research. While our Mixer-based communication approach could serve as a foundation, substantial investigation is necessary to reduce communication overhead during the backward pass without compromising training performance.

At present, our method is backed up by the reported empirical evaluations. Exploring the theoretical properties of ROCKET, particularly in conjunction with the effects of quantization, offers an interesting direction for future research.

7 Conclusions

In this work, we have demonstrated that designing split learning at the intersection of ML and communication enables high accuracy training on ultra-low-power hardware. The result of our design–RockNet–is the first TinyML split learning method capable of training non-linear timeseries classifiers on ultra-low-power hardware. It integrates ROCKET classifiers with distributed compute and Mixer, a wireless communication protocol based on synchronous transmissions and network coding. By efficiently pooling the resources of all devices within a CPS, this enables training of ROCKET classifiers, which are too large for a single ultra-low-power device.

Our experiments demonstrate that RockNet features a significantly higher accuracy than on-device central NN training. RockNet scales well with number of devices, significantly reducing per-device memory consumption and latency.

Additionally, we demonstrated that ROCKET can be trained on ultra-low-power hardware. Although ROCKET delivers superior accuracy and lower resource consumption compared to classifiers such as HIVE-COTE, TS-CHIEF, and ResNet [15–17, 74], its suitability for ultra-low-power contexts was previously unproven. Our parallelization strategy, merging ROCKET with Mixer, enables this and opens new avenues for deploying advanced ML algorithms on resource-constrained hardware.

In this work, we have not investigated the fault tolerance of our approach, e.g., regarding message loss and node failures. While the first seldom occurs due to Mixer's high reliability [30], the latter should be investigated in future work.

A promising direction for future research is to examine the interplay between RockNet's online training capabilities and the efficiency gains achievable through ROCKET pruning methods [28]. Combining these techniques post-training may further reduce resource consumption during inference.

Finally, our split learning approach is not limited to ROCKET classifiers. Mixer is generally well-suited for distributed AI in low-power, multi-hop networks, suggesting that future research should explore how other ML algorithms can be effectively distributed across CPS devices using this protocol.

Acknowledgments

This work was supported by the German Research Foundation (DFG) within the priority program 1914 (grant TR 1433/2) and within the Emmy Noether project NextIoT (ZI 1635/2-1), and by the LOEWE initiative (Hesse, Germany) within the emergenCITY center (LOEWE/1/12/519/03/05.001(0016)/72).

The authors gratefully acknowledge the computing time provided to them at the NHR Center NHR4CES at RWTH Aachen University (p0021919). This is funded by the Federal Ministry of Education and Research, and the state governments participating on the basis of the resolutions of the GWK for national high performance computing at universities (www.nhr-verein.de/unsere-partner).

We thank Andrés Posada Moreno and Friedrich Solowjow for insightful discussions, and Alperen Cebecik for his support with implementing the approach.

References

[1] Anuradha M Annaswamy, Karl H Johansson, and G Pappas. 2024. Control for societal-scale challenges: Road map 2030. *IEEE Control Systems Magazine* (2024).

[2] Rachad Atat, Lingjia Liu, Jinsong Wu, Guangyu Li, Chunxuan Ye, and Yi Yang. 2018. Big data meet cyber-physical systems: A panoramic survey. *IEEE Access* (2018).

[3] Dominik Baumann, Fabian Mager, Ulf Wetzker, Lothar Thiele, Marco Zimmerling, and Sebastian Trimpe. 2020. Wireless control for smart manufacturing: Recent approaches and open challenges. *Proc. IEEE* (2020).

[4] Jonas Beuchert, Friedrich Solowjow, Sebastian Trimpe, and Thomas Seel. 2020. Overcoming bandwidth limitations in wireless sensor networks by exploitation of cyclic signal patterns: An event-triggered learning approach. *Sensors* (2020).

[5] Christopher M. Bishop and Hugh Bishop. 2024. *Deep Learning: Foundations and Concepts*. Springer.

[6] Rohit Kumar Bondugula, Siba K Udgata, and Kaushik Bhargav Sivangi. 2023. A novel deep learning architecture and MINIROCKET feature extraction method for human activity recognition using ECG, PPG and inertial sensor dataset. *Applied Intelligence* (2023).

[7] Han Cai, Chuang Gan, Ligeng Zhu, and Song Han. 2020. Tinytl: Reduce memory, not parameters for efficient on-device learning. *Advances in Neural Information Processing Systems* (2020).

[8] Yicheng Cai, Haizhou Wang, Huali Ye, Yanwen Jin, and Wei Gao. 2023. Depression detection on online social network with multivariate time series feature of user depressive symptoms. *Expert Systems with Applications* (2023).

[9] Shanshan Chen, John Lach, Benny Lo, and Guang-Zhong Yang. 2016. Toward pervasive gait analysis with wearable sensors: A systematic review. *IEEE journal of biomedical and health informatics* (2016).

[10] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. 2015. The UCR Time Series Classification Archive. www.cs.ucr.edu/~eamonn/time_series_data/.

[11] Cisco. 2015. Fog Computing and the Internet of Things: Extend the Cloud to Where the Things Are. Accessed: 26-3-2024.

[12] Hoang Anh Dau, Eamonn Keogh, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, Yanping, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, Gustavo Batista, and Hexagon-ML . 2018. The UCR Time Series Classification Archive. https://www.cs.ucr.edu/~eamonn/time_series_data_2018/.

[13] Fabrizio De Vita, Rawan MA Nawaiseh, Dario Bruneo, Valeria Tomaselli, Marco Lattuada, and Mirko Falchetto. 2023. μ-ff: On-device forward-forward training algorithm for microcontrollers. In *IEEE International Conference on Smart Computing (SMARTCOMP)*.

[14] Fabrizio De Vita, Giorgio Nocera, Dario Bruneo, Valeria Tomaselli, and Mirko Falchetto. 2022. On-device training of deep learning models on edge microcontrollers. In *2022 IEEE International Conferences on Internet of Things (iThings) and IEEE Green Computing & Communications (GreenCom) and IEEE Cyber, Physical & Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*. IEEE.

[15] Angus Dempster, François Petitjean, and Geoffrey I Webb. 2020. ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery* (2020).

[16] Angus Dempster, Daniel F Schmidt, and Geoffrey I Webb. 2021. Minirocket: A very fast (almost) deterministic transform for time series classification. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*.

[17] Angus Dempster, Daniel F Schmidt, and Geoffrey I Webb. 2023. Hydra: Competing convolutional kernels for fast and accurate time series classification. *Data Mining and Knowledge Discovery* (2023).

[18] Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 2022. 8-bit Optimizers via Block-wise Quantization. In *International Conference on Learning Representations*.

[19] Simone Disabato and Manuel Roveri. 2020. Incremental on-device tiny machine learning. In *Proceedings of the 2nd International workshop on challenges in artificial intelligence and machine learning for internet of things*.

[20] Jiangsu Du, Yuanxin Wei, Shengyuan Ye, Jiazhi Jiang, Xu Chen, Dan Huang, and Yutong Lu. 2024. Co-designing Transformer Architectures for Distributed Inference with Low Communication. *IEEE Transactions on Parallel and Distributed Systems* (2024).

[21] Lachit Dutta and Swapna Bharali. 2021. Tinyml meets IoT: A comprehensive survey. *Internet of Things* (2021).

[22] Xiang Fei, Nazaraf Shah, Nandor Verba, Kuo-Ming Chao, Victor Sanchez-Anguix, Jacek Lewandowski, Anne James, and Zahid Usman. 2019. CPS data streams analytics based on machine learning for Cloud and Fog Computing: A survey. *Future generation computer systems* (2019).

[23] Federico Ferrari, Marco Zimmerling, Lothar Thiele, and Olga Saukh. 2011. Efficient network flooding and time synchronization with Glossy. In *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*.

[24] Yansong Gao, Minki Kim, Sharif Abuadbba, Yeonjae Kim, Chandra Thapa, Kyuyeon Kim, Seyit A Camtepe, Hyoungshick Kim, and Surya Nepal. 2020. End-to-end evaluation of federated learning and split learning for internet of things. *arXiv preprint arXiv:2003.13376* (2020).

[25] Yansong Gao, Minki Kim, Chandra Thapa, Alsharif Abuadbba, Zhi Zhang, Seyit Camtepe, Hyoungshick Kim, and Surya Nepal. 2021. Evaluation and optimization of distributed machine learning techniques for internet of things. *IEEE Trans. Comput.* 10 (2021).
[26] Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. 2022. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*.
[27] Nil Llisterri Giménez, Joan Miquel Solé, and Felix Freitag. 2023. Embedded federated learning over a LoRa mesh network. *Pervasive and Mobile Computing* (2023).
[28] Marco Giordano, Silvano Cortesi, Michele Crabolu, Lavinia Pedrollo, Giovanni Bellusci, Tommaso Bendinelli, Engin Türetken, Andrea Dunbar, and Michele Magno. 2023. Optimizing Iot-Based Asset and Utilization Tracking: Efficient Activity Classification With Minirocket on Resource-Constrained Devices. *arXiv preprint arXiv:2310.14758* (2023).
[29] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604* (2018).
[30] Carsten Herrmann, Fabian Mager, and Marco Zimmerling. 2018. Mixer: Efficient Many-to-All Broadcast in Dynamic Wireless Mesh Networks. In *16th ACM Conference on Embedded Networked Sensor Systems*.
[31] Tracey Ho, Muriel Médard, Ralf Koetter, David R. Karger, Michelle Effros, Jun Shi, and Ben Leong. 2006. A Random Linear Network Coding Approach to Multicast. *IEEE Transactions on Information Theory* (2006). https://doi.org/10.1109/TIT.2006.881746
[32] Mohd Javaid, Abid Haleem, Ravi Pratap Singh, and Rajiv Suman. 2022. Artificial intelligence applications for industry 4.0: A literature-based study. *Journal of Industrial Integration and Management* (2022).
[33] Yuang Jiang, Shiqiang Wang, Victor Valls, Bong Jun Ko, Wei-Han Lee, Kin K Leung, and Leandros Tassiulas. 2022. Model pruning enables efficient federated learning on edge devices. *IEEE Transactions on Neural Networks and Learning Systems* (2022).
[34] Tian Jin and Seokin Hong. 2019. Split-cnn: Splitting window-based operations in convolutional neural networks for memory system optimization. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*.
[35] Hamidreza Keshavarz, Mohammad Saniee Abadeh, and Reza Rawassizadeh. 2020. Sefr: A fast linear-time classifier for ultra-low power devices. *arXiv preprint arXiv:2006.04620* (2020).
[36] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[37] Yusuke Koda, Jihong Park, Mehdi Bennis, Koji Yamamoto, Takayuki Nishio, and Masahiro Morikura. 2019. One pixel image and RF signal based split learning for mmWave received power prediction. In *Proceedings of the 15th International Conference on emerging Networking EXperiments and Technologies*.
[38] Kavya Kopparapu and Eric Lin. 2021. TinyFedTL: Federated transfer learning on tiny devices. *arXiv preprint arXiv:2110.01107* (2021).
[39] Kavya Kopparapu, Eric Lin, John G Breslin, and Bharath Sudharsan. 2022. Tinyfedtl: Federated transfer learning on ubiquitous tiny IoT devices. In *IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*.
[40] Mai Le, Thien Huynh-The, Tan Do-Duy, Thai-Hoc Vu, Won-Joo Hwang, and Quoc-Viet Pham. 2024. Applications of Distributed Machine Learning for the Internet-of-Things: A Comprehensive Survey. *IEEE Communications Surveys & Tutorials* (2024).
[41] Seulki Lee and Shahriar Nirjon. 2020. Learning in the wild: When, how, and what to learn for on-device dataset adaptation. In *Proceedings of the 2nd International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*.
[42] Krijn Leentvaar and Jan H. Flint. 1976. The Capture Effect in FM Receivers. *IEEE Transactions on Communications* (1976). https://doi.org/10.1109/TCOM.1976.1093327
[43] Fan Liang, William Grant Hatcher, Weixian Liao, Weichao Gao, and Wei Yu. 2019. Machine learning for security and the internet of things: the good, the bad, and the ugly. *EEE Access* (2019).
[44] Ji Lin, Ligeng Zhu, Wei-Ming Chen, Wei-Chen Wang, Chuang Gan, and Song Han. 2022. On-device training under 256kb memory. *Advances in Neural Information Processing Systems* (2022).
[45] Jason Lines, Sarah Taylor, and Anthony Bagnall. 2018. Time series classification with HIVE-COTE: The hierarchical vote collective of transformation-based ensembles. *ACM Transactions on Knowledge Discovery from Data (TKDD)* (2018).
[46] Nil Llisterri Giménez, Marc Monfort Grau, Roger Pueyo Centelles, and Felix Freitag. 2022. On-device training of machine learning models on microcontrollers with federated learning. *Electronics* (2022).
[47] Fabian Mager, Dominik Baumann, Romain Jacob, Lothar Thiele, Sebastian Trimpe, and Marco Zimmerling. 2019. Feedback control goes wireless: Guaranteed stability over low-power multi-hop networks. In *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*.
[48] Jiachen Mao, Xiang Chen, Kent W Nixon, Christopher Krieger, and Yiran Chen. 2017. Modnn: Local distributed mobile computing system for deep neural network. In *Design, Automation & Test in Europe Conference & Exhibition*.
[49] Jiachen Mao, Zhongda Yang, Wei Wen, Chunpeng Wu, Linghao Song, Kent W Nixon, Xiang Chen, Hai Li, and Yiran Chen. 2017. Mednn: A distributed mobile system with enhanced partition and deployment for large-scale dnns. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*.

[50] Xavier Marimon, Itziar Mengual, Carlos López-de Celis, Alejandro Portela, Jacobo Rodríguez-Sanz, Iria Andrea Herráez, and Albert Pérez-Bellmunt. 2024. Kinematic analysis of human gait in healthy young adults using IMU sensors: exploring relevant machine learning features for clinical applications. *Bioengineering* (2024).

[51] Akhil Mathur, Daniel J Beutel, Pedro Porto Buarque de Gusmao, Javier Fernandez-Marques, Taner Topal, Xinchi Qiu, Titouan Parcollet, Yan Gao, and Nicholas D Lane. 2021. On-device federated learning with flower. *arXiv preprint arXiv:2104.03042* (2021).

[52] Hiroki Matsutani, Mineto Tsukada, and Masaaki Kondo. 2022. On-Device Learning: A Neural Network Based Field-Trainable Edge AI. *arXiv preprint arXiv:2203.01077* (2022).

[53] Mehdi Mohammadi, Ala Al-Fuqaha, Sameh Sorour, and Mohsen Guizani. 2018. Deep learning for IoT big data and streaming analytics: A survey. *IEEE Communications Surveys & Tutorials* (2018).

[54] Patrick W Moore, Hrishikesh M Rao, Christine Beauchene, Emilie Cowen, Sophia Yuditskaya, Thomas Heldt, and Laura J Brattain. 2023. Efficient deep learning on wearable physiological sensor data for pilot flight performance analysis. In *IEEE 19th International Conference on Body Sensor Networks (BSN)*.

[55] Weijian Ni, Chengqin Zhang, Tong Liu, Qingtian Zeng, Lingzhe Xu, and Huaiqing Wang. 2024. An efficient astronomical seeing forecasting method by random convolutional Kernel transformation. *Engineering Applications of Artificial Intelligence* (2024).

[56] Felix O Olowononi, Danda B Rawat, and Chunmei Liu. 2020. Resilient machine learning for networked cyber physical systems: A survey for machine learning security to securing machine learning for CPS. *IEEE Communications Surveys & Tutorials* (2020).

[57] Maarten G Poirot, Praneeth Vepakomma, Ken Chang, Jayashree Kalpathy-Cramer, Rajiv Gupta, and Ramesh Raskar. 2019. Split learning for collaborative deep learning in healthcare. *arXiv preprint arXiv:1912.12115* (2019).

[58] Francesco Prinzi, Alessia Orlando, Salvatore Gaglio, and Salvatore Vitabile. 2024. Breast cancer classification through multivariate radiomic time series analysis in DCE-MRI sequences. *Expert Systems with Applications* (2024).

[59] Christos Profentzas, Magnus Almgren, and Olaf Landsiedel. 2022. MiniLearn: On-Device Learning for Low-Power IoT Devices.. In *EWSN*.

[60] Leonardo Ravaglia, Manuele Rusci, Davide Nadalini, Alessandro Capotondi, Francesco Conti, and Luca Benini. 2021. A tinyml platform for on-device continual learning with quantized latent replays. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* (2021).

[61] Haoyu Ren, Darko Anicic, and Thomas A Runkler. 2021. Tinyol: Tinyml with online-learning on microcontrollers. In *International Joint Conference on Neural Networks (IJCNN)*.

[62] Wei-Qing Ren, Yu-Ben Qu, Chao Dong, Yu-Qian Jing, Hao Sun, Qi-Hui Wu, and Song Guo. 2023. A survey on collaborative DNN inference for edge intelligence. *Machine Intelligence Research* (2023).

[63] Rohit Sahu, Ryan Toepfer, Mathew D Sinclair, and Henry Duwe. 2021. DENNI: Distributed Neural Network Inference on Severely Resource Constrained Edge Devices. In *IEEE International Performance, Computing, and Communications Conference*.

[64] Ashish Kumar Saxena, Varun Kumar Reja, and Koshy Varghese. 2020. Iot enabled framework for real-time management of power-tools at construction projects. In *ISARC. Proceedings of the International Symposium on Automation and Robotics in Construction*. IAARC Publications.

[65] Markus Schuß, Carlo Alberto Boano, Manuel Weber, and Kay Römer. 2017. A Competition to Push the Dependability of Low-Power Wireless Protocols to the Edge. In *International Conference on Embedded Wireless Systems and Networks (EWSN)*. Uppsala, Sweden.

[66] Muhammad Shafique, Theocharis Theocharides, Vijay Janapa Reddy, and Boris Murmann. 2021. TinyML: Current progress, research challenges, and future roadmap. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE.

[67] Ahmed Shifaz, Charlotte Pelletier, François Petitjean, and Geoffrey I Webb. 2020. TS-CHIEF: a scalable and accurate forest algorithm for time series classification. *Data Mining and Knowledge Discovery* (2020).

[68] Rafael Stahl, Alexander Hoffman, Daniel Mueller-Gritschneder, Andreas Gerstlauer, and Ulf Schlichtmann. 2021. DeeperThings: Fully distributed CNN inference on resource-constrained edge devices. *International Journal of Parallel Programming* (2021).

[69] Bharath Sudharsan, John G Breslin, and Muhammad Intizar Ali. 2021. Globe2train: A framework for distributed ml model training using iot devices across the globe. In *2021 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/IOP/SCI)*. IEEE.

[70] Bharath Sudharsan, John G Breslin, and Muhammad Intizar Ali. 2021. Imbal-ol: Online machine learning from imbalanced data streams in real-world iot. In *2021 IEEE International Conference on Big Data (Big Data)*. IEEE.

[71] Bharath Sudharsan, John G Breslin, and Muhammad Intizar Ali. 2021. Ml-mcu: A framework to train ml classifiers on mcu-based IoT edge devices. *IEEE Internet of Things Journal* (2021).

[72] Bharath Sudharsan, John G Breslin, Muhammad Intizar Ali, Peter Corcoran, and Rajiv Ranjan. 2022. ElastiQuant: elastic quantization strategy for communication efficient distributed machine learning in IoT. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*.

[73] Bharath Sudharsan, Piyush Yadav, John G Breslin, and Muhammad Intizar Ali. 2021. Train++: An incremental ml model training algorithm to create self-learning iot devices. In *2021 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/IOP/SCI)*. IEEE.

[74] Chang Wei Tan, Angus Dempster, Christoph Bergmeir, and Geoffrey I Webb. 2022. MultiRocket: multiple pooling operators and transformations for fast and effective time series classification. *Data Mining and Knowledge Discovery* (2022).
[75] Damien Teney, Yong Lin, Seong Joon Oh, and Ehsan Abbasnejad. 2024. Id and ood performance are sometimes inversely correlated on real-world datasets. *Advances in Neural Information Processing Systems* (2024).
[76] Wolfgang Teufl, Michael Lorenz, Markus Miezal, Bertram Taetz, Michael Fröhlich, and Gabriele Bleser. 2018. Towards inertial sensor based mobile gait analysis: Event-detection and spatio-temporal parameters. *Sensors* (2018).
[77] Chandra Thapa, Pathum Chamikara Mahawaga Arachchige, Seyit Camtepe, and Lichao Sun. 2022. Splitfed: When federated learning meets split learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*.
[78] Dante Trabassi, Mariano Serrao, Tiwana Varrecchia, Alberto Ranavolo, Gianluca Coppola, Roberto De Icco, Cristina Tassorelli, and Stefano Filippo Castiglia. 2022. Machine learning approach to support the detection of Parkinson's disease in IMU-based Gait analysis. *Sensors* (2022).
[79] Matteo Trobinger, Gabriel de Albuquerque Gleizer, Timofei Istomin, Manuel Mazo, Amy L. Murphy, and Gian Pietro Picco. 2021. The Wireless Control Bus: Enabling Efficient Multi-Hop Event-Triggered Control with Concurrent Transmissions. (2021).
[80] Yigit Tuncel, Toygun Basaklar, Dina Carpenter-Graffy, and Umit Ogras. 2023. A Self-Sustained CPS Design for Reliable Wildfire Monitoring. *ACM Transactions on Embedded Computing Systems* (2023).
[81] Víctor Manuel Vargas, Riccardo Rosati, César Hervás-Martínez, Adriano Mancini, Luca Romeo, and Pedro Antonio Gutiérrez. 2023. A hybrid feature learning approach based on convolutional kernels for ATM fault prediction using event-log data. *Engineering Applications of Artificial Intelligence* (2023).
[82] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. 2018. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564* (2018).
[83] Xiaolu Wang, Zijian Li, Shi Jin, and Jun Zhang. 2024. Achieving linear speedup in asynchronous federated learning with heterogeneous clients. *IEEE Transactions on Mobile Computing* (2024).
[84] Florian Wenzel, Andrea Dittadi, Peter Gehler, Carl-Johann Simon-Gabriel, Max Horn, Dominik Zietlow, David Kernert, Chris Russell, Thomas Brox, Bernt Schiele, et al. 2022. Assaying out-of-distribution generalization in transfer learning. *Advances in Neural Information Processing Systems* (2022).
[85] Christiane Werner, Ulrike Wallrabe, Andreas Christen, Laura Comella, Carsten Dormann, Anna Göritz, Rüdiger Grote, Simon Haberstroh, Mazin Jouda, Ralf Kiese, et al. 2024. ECOSENSE-Multi-scale quantification and modelling of spatio-temporal dynamics of ecosystem processes by smart autonomous sensor networks. *Research Ideas and Outcomes* (2024).
[86] Hao Wu, Patrick Judd, Xiaojie Zhang, Mikhail Isaev, and Paulius Micikevicius. 2020. Integer quantization for deep learning inference: Principles and empirical evaluation. *arXiv preprint arXiv:2004.09602* (2020).
[87] Lars Wulfert, Navidreza Asadi, Wen-Yu Chung, Christian Wiede, and Anton Grabmaier. 2023. Adaptive Decentralized Federated Gossip Learning for Resource-Constrained IoT Devices. In *Proceedings of the 4th International Workshop on Distributed Machine Learning*.
[88] Lars Wulfert, Johannes Kühnel, Lukas Krupp, Justus Viga, Christian Wiede, Pierre Gembaczka, and Anton Grabmaier. 2024. AIfES: A Next-Generation Edge AI Framework. *IEEE Trans. on Pattern Analysis and Machine Intelligence* (2024).
[89] Lars Wulfert, Christian Wiede, and Anton Grabmaier. 2023. Tinyfl: On-device training, communication and aggregation on a microcontroller for federated learning. In *21st IEEE Interregional NEWCAS Conference (NEWCAS)*.
[90] Qinghua Zhou, Quentin Anthony, Lang Xu, Aamir Shafi, Mustafa Abduljabbar, Hari Subramoni, and Dhabaleswar K DK Panda. 2023. Accelerating distributed deep learning training with compression assisted allgather and reduce-scatter communication. In *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE.
[91] Marco Zimmerling, Luca Mottola, and Silvia Santini. 2020. Synchronous Transmissions in Low-Power Wireless: A Survey of Communication Protocols and Network Services. *ACM Comput. Surv.* (2020).

A I.I.D vs Non-I.I.D Distributed Datasets

We conducted experiment on the UCR dataset with i.i.d. sampled data and non-i.i.d. sampled data. For the non-i.i.d. data, we sorted the dataset according to its indexes. Then, we split it evenly among 10 devices. Through this, a device only holds datapoints form a few classes and different devices hold data from different classes. Our results shown in Figure 10 demonstrate that RockNet retains the same accuracy, when trained on non-i.i.d. sampled data, compared to i.i.d. sampled data.

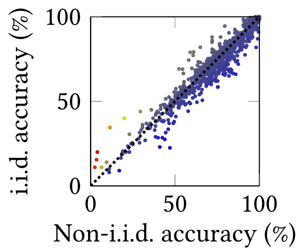


Fig. 10. Ablation i.i.d. vs. non-i.i.d distributed datasets. The experiment demonstrates that RockNet is robust against non-i.i.d. data.