TinyChirp: Bird Song Recognition Using TinyML Models on Low-power Wireless Acoustic Sensors

Z. Huang*, A. Tousnakhoff*, P. Kozyr†, R. Rehausen‡, F. Bießmann†¶ R. Lachlan§, C. Adjih‖, E. Baccelli‖*
*Freie Universität Berlin †Berlin University of Applied Sciences
‡Woks Audio GmbH §Royal Holloway University of London
¶Einstein Center for Digital Future ‖Inria, France

***Abstract*—Monitoring biodiversity at scale is challenging. Detecting and identifying species in fine grained taxonomies requires highly accurate machine learning (ML) methods. Training such models requires large high quality data sets. And deploying these models to low power devices requires novel compression techniques and model architectures. While species classification methods have profited from novel data sets and advances in ML methods, in particular neural networks, deploying these state-of-the-art models to low power devices remains difficult. Here we present a comprehensive empirical comparison of various tinyML neural network architectures and compression techniques for species classification. We focus on the example of bird song detection, and more concretely on a data set curated for studying the corn bunting bird species. We publish the data set along with all the code and experiments of this study. In our experiments we comparatively evaluate predictive performance, memory and time complexity of spectrogram-based methods and of more recent approaches operating directly on the raw audio signal. Our results demonstrate that *TinyChirp* – our approach – can robustly detect individual bird species with precisions over 0.98 and reduce energy consumption compared to state-of-the-art, such that an autonomous recording unit lifetime on a single battery charge is extended from 2 weeks to 8 weeks, almost an entire season.**

***Index Terms*—TinyML, Microcontroller, Acoustic Sensors, Machine Learning**

I. Introduction

The typical data pipeline with bio-acoustic research requires the deployment of sensors, in which each node is battery-powered and left in the field to record environmental sound, continuously, for a long period (e.g. for a whole season). Thereafter, the recorded data is manually collected by researchers on-site, from each node, and further analyzed in laboratory. In the case of avian species for instance, only the targeted bird species is relevant within the recorded data, and the rest of the data is to be discarded ultimately. This is both a cumbersome process and a waste of resources, not only at this downstream stage in the lab, but also upstream in the field, on each node, while recording large amounts of irrelevant data. In this pipeline, continuous recording of audio thus creates a bottleneck in terms of memory and energy budgets available on individual sensors – typically limited to a small battery, and an SD card, respectively, driven by rudimentary software running on a low-power microcontroller. Such hardware is very energy-efficient, but very limited in memory resources, with RAM memory budgets in the order of 500 kiloBytes [1], which yields specific constraints on software embedded on such devices [2].

Meanwhile, as Artificial Intelligence (AI) achieves excellent performance in pattern recognition of audio and biosignals, more and more bioacoustic researchers leverage Machine Learning (ML) related methods to improve accuracy and efficiency. However, until recently, such ML models, e.g. BirdNET [3] were confined to lab use only, as their resource requirements (GigaBytes of RAM) are way beyond the capacity of microcontroller-based hardware available on sensors in the field. However, recent advances in TinyML, a lively field of research targeting machine learning for microcontrollers, offer a glimpse of hope that pattern recognition of bioacoustic signals might become possible, over longer periods, as required by the aforementioned use cases.

Paper Contributions. In this paper, we explore the possibility of using TinyML in practice, on common low-power microcontroller hardware, for a concrete use case: monitoring corn bunting birds' songs, in a rural area in the UK, over several months in a row, using a fleet of energy-efficient acoustic sensors. We aim to answer the following questions: (Q1) Can we pre-screen the audio on the low-power sensor node, so as to only store the targeted bird songs on each sensor? (Q2) How does that extend the device's lifetime, in terms of memory and energy budgets? More in detail, our contributions are as follows:

- We propose a pipeline to record, recognize and store specific bird songs on low-power microcontroller-based devices, which can be further scale-up to universal species;
- We developed a neural network with high accuracy on bird song recognition. Furthermore, we optimize the network with partial convolution technology to minimize the memory consumption on deployment;
- We propose a two-stage binary classification approach to reduce computational and storage cost and enhance the overall accuracy;
- We provide experimental results on both the classification performance of models and resource consumption on low-power devices;
- We publish a new data set of curated audio recordings snippets of corn bunting bird songs;

Corresponding author: zhaolan.huang@fu-berlin.de.

- We provide open source code[1] to reproduce the results on common Microcontroller Units (MCUs).

II. Background & Related Work

A. Bird Song Recognition

Bird song detection has seen significant recent advances using various machine learning techniques. The common prevalent approach uses deep learning methods on preprocessed audio, which transforms the signal into a spectrogram using Short-Time Fourier Transform (STFT) [4]. This process effectively redefines the audio recognition task as an image recognition task operating on a (log) Mel spectrogram [5], [6]. Convolutional Neural Networks (CNNs) [4], [5], [7] and Residual Neural Networks (ResNets) [6], [8] are widely utilized for such image recognition tasks in bird song detection. BirdNET [3], a well-known neural network for bird song recognition, is derived from the family of residual networks and can recognize 6,000 of the world's most common species at the time of writing.

Another approach involves transforming the audio signal into Mel-Frequency Cepstral Coefficients (MFCCs), which are then used as input for bird song classification [7], [9], [10]. Alternatively, feature extraction using wavelet decomposition can be employed for this purpose [11]. Additionally, certain studies combine different architectures [5], [9]. While these methods are effective for bird song recognition, their computational requirements are typically too demanding for execution on microcontrollers.

In this work we focus on bird songs, rather than calls. The main reasons for this are the higher complexity of bird songs, their role in sexual selection, and the generally richer behavioral information they provide.

B. Bioacoustic Audio Datasets

High-quality and diverse datasets are crucial for training models to accurately recognize birds and differentiate between species by their unique songs. The Xeno-canto database[2] is one of the largest and most comprehensive collections of bird sounds, containing over 500,000 recordings from more than 10,000 species. Another extensive collection of bird audio recordings is available in the Macaulay Library[3], renowned for its high-quality bird audio recordings and detailed metadata. The combination of the Macaulay Library and Xeno-canto audio recordings was used to train BirdNet [3]. In [5], the public dataset CLO-43DS contains recordings of 43 different North American wood-warblers. Another notable collection is the Birdsdata dataset with 20 bird species from the Beijing Academy of Artificial Intelligence (BAAI) repository [6], [8], [9]. The BEANS benchmark [12], designed to evaluate ML algorithms for bioacoustics tasks across various species, includes 12 public datasets on birds, mammals, anurans, and insects. To complement datasets focused on bird species, the Google AudioSet [3] and Urbansound8K [8] datasets are often used to include non-bird species sounds.

C. Tiny Machine Learning (TinyML)

For models to operate on low-power devices, they must be compact and computationally efficient. Studies have demonstrated the use of lightweight CNNs for various Internet of Sounds applications [13]–[18]. Other model architectures, such as tiny vision transformers have also been employed for audio classification tasks in various studies [19]–[23]. Moreover, various quantization techniques are applied to models, aiming to reduce their size to fit within the constraints of low-power devices [24]–[26].

Creating spectrograms from audio signals can be power-intensive; hence, some studies use instead raw time-series data as input for neural networks [27]–[29]. Notably, raw audio signals have been used for urban sound analysis [25], [30]. The advantages of using time-series data include reduced computational load and suitability for TinyML on low-power devices.

In a nutshell: the integration of TinyML and acoustic sensors has become a hot topic in the field of Internet of Sounds [31]. As demonstrated in prior bioacoustic applications such as [4], [32], [33], TinyML promises to offer new, appealing combinations of high accuracy and resource-efficiency.

D. Embedded Software Platforms for TinyML

The widely used model transpiler TVM (Tensor Virtual Machine [34]) has recently been extended with uTVM, providing automated transpilation and compilation for models output by major ML frameworks (TFLM, Pytorch, etc.). As such uTVM exposes low-level routines and optimizes these for execution on different processing units, including for targets such as a large variety of microcontrollers. Prior works such as [35], [36] or MLPerfTiny [37] focused on the production, performance and analysis of standard benchmark suites of representative TinyML tasks on different microcontrollers. Conversely, prior work such as U-TOE [38] or RIOT-ML [39] provide embedded operating system integration of TinyML, facilitating TinyML benchmarking and continuous deployment over low-power wireless network links such as IEEE 802.15.4 or BLE.

III. Bioacoustic Monitoring Scenario

As depicted in Figure 1, a network of battery-powered, autonomous recording units (ARUs, i.e. microcontroller-based acoustic sensors) is deployed across a monitored area. An example of ARU is given in [40].

These sensors remain in the field for an entire season, typically around six months. Locations are often remote and hard to access and devices may be distributed over a relatively wide area, making visiting them time-consuming. Birds typically do not sing consistently during the whole day, while each bird has several different song types that need to be regularly sampled. Moreover, each individual bird moves around its individual territory, singing from a range of song posts, not all of which will be adequately recorded from any one location.

[1] see https://github.com/TinyPART/TinyChirp

[2] see https://www.xeno-canto.org

[3] see https://www.macaulaylibrary.org/

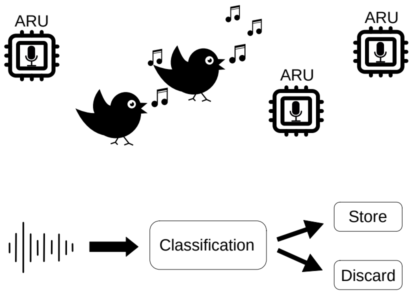

Fig. 1. *Top*: Low-power autonomous recording units (ARU) distributed spatially to monitor birds. *Bottom*: audio sample classification on an ARU.

On the network aspect, the sensors are distributed approximately evenly, ensuring that each sensor can be wireless connected by more than two neighboring sensors. This arrangement facilitates approximate triangulation through distance estimation based on signal strength. The placement process can be streamlined using LEDs and a basic ultra-low-power wireless protocol, such as IEEE 802.15.4 or LoRa. For space reason, we do not detail network aspects in the paper, but rather focus on the on-device machine learning aspects.

Although our work is designed around one very specific real-world research scenario (monitoring common Corn Bunting bird songs) the architectures should be readily applicable to other audio data classification tasks.

A. Corn Bunting Monitoring Use-Case

SongBeam [40] microcontroller-based recorders have been used to monitor corn bunting (*Emberiza calandra*) birds in Oxfordshire, UK. A set of 30-40 devices have been deployed since 2022, currently comprising 3 complete breeding seasons (February - July).

SongBeam devices are based on an ARM Cortex-M microcontroller, run on 4 D-cell batteries and record 4-channel WAV files onto 128 GB microSD cards. To maintain such a fleet of sensors, devices are currently checked approximately every 2 weeks, especially because memory space is soon exhausted on the microSD cards.

The above deployment has been used to produce a significant part of the dataset we present in Section V-A. Our experience using SongBeam and a preliminary analysis of the raw dataset we produced shows that less than 10% of recording time contains useful recordings. Given the extended deployment period (6 months), remaining low-power while improving storage efficiency is thus paramount.

This description of the limitations of using SongBeam recorders to monitor corn buntings is likely to apply to many biomonitoring scenarios: solar power may allow for extending deployment lifetime in remote locations, but storage of recordings is a limiting factor.

IV. Evaluation Metrics

On the one hand, as the basic part of bird song recording, the model should identify the target bird song as much as possible, potentially requiring a complex structure with a large size. On the other hand, the limited resource budget allows only simple models to be deployed on Internet of Things (IoT) devices.

Thus we considered two orthogonal types of metrics to evaluate the prediction performance on bird song classification and resource usage on low-power IoT devices of TinyML models. Preliminary results are presented in Section X.

A. On-device Resource Usage

We performed extensive profiling of all models on low-power devices to capture their effective memory and time complexity in a real-world application scenario. The metrics are helpful to investigate the energy efficiency of the models, complementing the commonly used metrics for predictive performance.

Memory (RAM) Consumption – This metric measures the amount of dynamic memory space (primary RAM) consumed by the model during inference. It reflects the memory footprint of the model activation and is important for low-power devices that have limited memory resources.

Storage (Flash memory) Consumption – This metric quantifies the amount of storage space, typically in terms of Flash memory region, required to store the compute instruction and associated parameters.

Computational Latency – This metric measures the time consumption of performing inference for each input sample at the model level. It reflects the inference speed of the model on the low-power device and plays a crucial role in real-time or latency-sensitive applications.

Energy Consumption – This metric is crucial for battery-operated and resource-constrained devices, where efficient energy usage can significantly impact device longevity and performance. Energy consumption encompasses both active power (when the device is triggered to perform tasks) and idle power (when the device is in sleep mode).

B. On-device Prediction Performance

Accuracy – It is defined as the ratio of correctly predicted instances to the total instances in the dataset. While accuracy is intuitive, it can be misleading, especially in highly imbalanced datasets. To address this limitation, accuracy should be combined with other metrics that offer a more detailed view of model performance.

Precision and Recall – They are two fundamental metrics of classification models used to evaluate the ability to distinguish true positives (TP) from false positives (FP) and false negatives (FN), particularly in the context of imbalanced datasets.

F-Score – There is often a trade-off between precision and recall; increasing one can lead to a decrease in the other. As two common instances, F_1-score weighs them evenly, while F_2-score treats recall as two times more important than precision, applying in scenarios where false positives are more tolerant than false negatives.

Receiver Operating Characteristic (ROC) Curve – While the above metrics are sensitive to class imbalance – which occurs in many species classification and detection tasks – there are other metrics that are more robust towards class imbalance. The receiver-operator characteristic (ROC) curve depicts the True Positive Rate (TPR) and False Positive Rate (FPR) across different decision thresholds.

V. Methodology

We used a combination of project-specific and publicly accessible data to train and validate our classification models. In the data pre-processing phase, the raw audio signals were further segmented, labeled, down-sampled and divided into different groups to generate classification datasets. The corresponding Mel-spectrograms of the pre-processed audio segments were also created for spectral-based methods (models). During the pre-processing stage, we conducted a pilot analysis to establish guidelines and determine specific hyperparameters for downsampling and spectrogram generation.

A. Data Acquisition

The publicly accessible data originated from Macaulay, Xeno-Canto, Google AudioSet, while the project-specific data were previously collected in a long-term research of bird song patterns. In order to ensure generalization across a wide variety of conditions for the target species, the corn bunting, we gathered as many recordings as possible from a heterogeneous suite of publicly available repositories, in addition to the custom data set collected. We aimed at increasing precision of all models by including a wide variety of non-target sounds. Our data set is made publicly available[4].

- Oxfordshire Corn Buntings. This project-specific library contains recordings of corn buntings along a transect of approximately 20 km in Southern England. Corn buntings sing in a mosaic-like pattern of geographical variation called dialects; our sample contains approximately 6 different dialects. The recordings were performed with directional parabolic microphones as described in [40].
- Macaulay Library. This library contains the necessary audio recordings for our target species as well as the other identified species. There are a total of 278 entries of audio recordings for the target species. For other species as non-target, we limit the selection to a maximum of 30 recordings per species and choose those with the highest average community rating, acquiring a total of 1468 recordings.
- Xeno-Canto. This library also contains recordings of target species and other species. We retrieved the recordings only marked with *song* and rated with the best quality level, resulting in 303 recordings of target species and 9622 recordings of other species as non-target.
- Google AudioSet. This dataset provides environmental sounds which contain non-bird sounds. We create a list of excluded categories to avoid overlap with other bird sounds. This data set includes weather-related sounds, such as rain, wind, thunderstorms, insects, other animal and human-related sounds, such as church bells, trucks, etc.

After gathering all audio recordings from the above libraries, we used BirdNET with the confidence threshold at 0.92 to identify and chop corn-bunting segments from all datasets, and labeled them as target bird songs; all segments were truncated or zero-padded to the length of 3 seconds. This high confidence threshold is intended to minimize incorrect labels, although it does not entirely eliminate the label noise, which could potentially skew the performance metrics of our models, particularly in cases where the models are trained on mislabeled data. Meanwhile, we chose other 3-s segments randomly and ensured no overlap with corn-bunting segments, with labeled as non-target bird songs. To avoid data leakage, we implement an additional function that segments occurring sequentially in the source audio file were not distributed across the training, validation and test sets. Table I shows the distribution of target and non-target segments over all datasets.

B. Data Pre-processing

This phase contains the following steps:

1) We divided the segments into training, validation and test sets with the ratio of $80 : 10 : 10$.
2) All segments were downsampled to 16 kHz using zero-order holder. These downsampled segments constituted the audio (waveform) dataset.
3) We transformed the downsampled audio segments into Mel-Spectrograms, and grouped them with the same splitting ratio to form the spectral dataset.

Pilot Analysis – We averaged the STFT spectrograms of target and non-target segments in the training dataset to investigate their frequency characteristics, as depicted in Figure 2. Obviously, a bright band lays on the spectrogram of target segments roughly between 4000 and 8000 Hz, hints that a sample rate with 16 kHz should be sufficient to preserve all frequency components of corn-bunting song according to Nyquist–Shannon theorem. A highpass filter can also be applied to eliminate low-frequency noise without damaging target songs. Thus, we downsampled the segments from 48 kHz to 16 kHz to reduce the resource consumption and improve the efficiency of classification phase. In the baseline model we also designed a highpass filter to enhance Signal-to-noise ratio (SNR).

Mel-Spectrogram – Mel-spectrogram is commonly used in audio classification tasks due to its ability to closely approximate human auditory perception. To create log Mel-spectrograms, we first generated magnitude spectrograms of STFT for all downsampled audio segments. The STFT was performed with a Hann window with a width of 1024 samples and a step size of 256 samples. This process generated 184 window frames and 513 frequency bins. Thereafter, the magnitude spectrograms were mapped onto the Mel scales with 80 Mel bins. In creating the Mel-spectrograms, we employed a sampling rate of 16 kHz and focused on the frequency range of 80-8000 Hz. This also allows ML models to capture

[4] see https://github.com/TinyPART/TinyChirp/tree/main/datasets

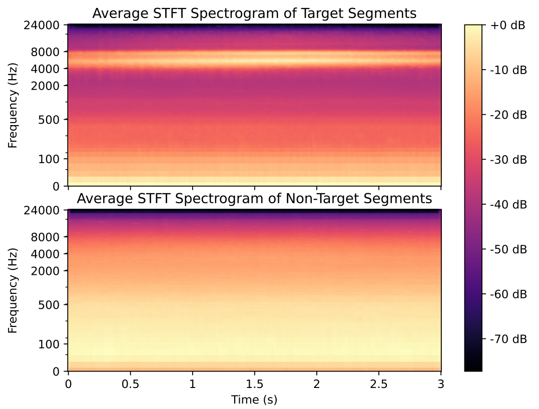

Fig. 2. Average STFT of target and non-target audio segments.

TABLE I
DATA DISTRIBUTION OVER DIFFERENT DATASETS.

Dataset	# Target	# Non-Target
Oxfordshire Corn Buntings	1566	0
Macaulay Library	1059	2057
Xeno-Canto	2111	2621
Google AudioSet	0	4651
Summary	4736	9329

subtle patterns outside the primary band (4000 Hz to 8000 Hz), adapt to variations, and improve overall classification accuracy by leveraging information that a simple filter might overlook. Lastly, we converted the Mel-spectrograms into a logarithmic magnitude scale to obtain log Mel-spectrograms, with the shape of 184×80.

VI. Baseline & Decision Strategy

We propose an approach to optimizing the screening process by combining a high-speed signal processing step (hereafter named ”baseline”) with the precision of an TinyML model, ensuring reliable results while maintaining resource-consumption efficiency.

A. Baseline

We developed a lightweight pre-selector signal processing step, not relying on machine learning. First, the audio segments are normalized by min-max schema. Thereafter, a 9th-order highpass Butterworth filter with the cut-off frequency of 7000 Hz[5] is imposed on the normalized segments, in order to suppress noise and non-relative sound in low-frequency and to preserve only target components. Finally, we calculate the signal power P of the filtered segments,

$$P = \frac{1}{N}\sum_{n=1}^{N} x(n)^2, \tag{1}$$

where $x(n)$ and N denote the data points and the length of a segment, respectively. We define two thresholds: t_{low} and t_{high}. If P is smaller than t_{low}, the MCU remains in the idle/low-power state, and the audio sample is discarded directly. Else, P is compared to t_{high}. Depending on this result, and on the decision strategy (see below), the audio sample is either stored or further analyzed via inference using the machine learning model. Both t_{low} and t_{high} were tuned on training dataset for optimized recall and F_2-score, respectively. The evaluation results are presented in Table IV.

B. TinyChirp Decision Strategy

We employ a double-phase approach for enhanced accuracy and efficiency of bird song recognition, combining the baseline with a TinyML model for further verification. The process is as follows:

1) **Step 1:** *Baseline Filtering* – Pre-screening is conducted using the baseline pre-processing step described in Section VI-A, designed to provide a quick and efficient preliminary analysis. If P is smaller than t_{low}, the microcontroller remains in the idle/low-power state, the audio sample is discarded and the process aborts. Else proceed to Step 2.
2) **Step 2:** *Inference-based Classification* – If the power-saving flag is set and if P is smaller than t_{high} a TinyML-based classification with higher accuracy is conducted via model inference. If the audio sample is not classified as the target, the audio sample is discarded and the process aborts. Else proceed to Step 3.
3) **Step 3:** *Storage* – Without further processing, the audio sample is logged, i.e. stored on the SD card.

Note that the decision strategy can therefore be configured with two ”knobs”. On one hand the power-saving flag determines skipping (or not) some of the computation and can thus decrease energy consumption. On the other hand, different TinyML models can be inserted in Step 2.

Next, we thus study two families of TinyML models for Step 2: the first category of model takes a Mel-Spectrogram as input, while the second category of model takes time-series as input. Table II presents a summary of the explored TinyML model structures.

VII. Classification Based on Spectrograms

In contrast to the computer vision community, where the transition from frequency decomposition based feature extraction to neural feature extraction on raw image data introduced in 2012 led to a significant increase in predictive performance [41], neural feature extraction on raw audio time series did not contribute to a comparable breakthrough yet. Instead most ML techniques for *audio* pattern recognition (somewhat surprisingly) were relying on *image* processing pipelines. Basically, image classification is performed, using a neural network, analyzing the spectrogram rendering of the audio trace.

Unlike the spatial dimensions in image classification, the axes of the input represent time and frequency in audio classification. Moreover, the pattern structure differs significantly:

[5] The parameters of the filter were determined by hyper-parameter search for optimized F_2-score on training dataset.

natural objects in images often have well-defined, closed contours, while spectrograms of sounds tend to be wide-band and sparsely distributed across the frequency-time domain. These distinctions require tailored approaches to effectively model and interpret the data in audio classification tasks.

For this reason, we initially consider the below two models, based on state-of-the-art neural network architectures.

CNN-Mel – This model contains two 2D convolutional layers for Mel-spectral inputs as feature extraction and two fully connected layers as classifier. The output of the classifier are normalized by Softmax function as well.

SqueezeNet-Mel – This model is based on SqueezeNet [42], an advanced backbone focused on optimizing the efficiency of computer vision applications by strategically reducing parameters, with comparable performance to AlexNet [41]. It leverages a so-called *Fire module* to achieve higher efficiency compared to standard convolutions with only a slight decrease in accuracy. We aligned its input shape with the Mel-spectrogram and tailored the output for binary classification.

Limitations of Spectrograms on Microcontrollers Practical experience on low-power microcontroller-based devices has shown however that producing and manipulating mel-spectrograms from audio signal streams on such devices is problematic. It inserts an additional step in the processing pipeline, which increases latency and memory requirements. Prior work (such as [43], section 6.2) concludes that the CPU bottleneck is not just the model inference time, but also the spectrogram calculation, which, compared to inference alone, almost doubles latency. Other previous work such as [44] measures on a quite powerful STM32F7 microcontroller that computing and writing in memory the mel-spectrogram of 30 columns and 40 frequency bands takes approximately 1 second. Note that these spectrogram dimensions are much smaller than our requirements (184 columns × 80 frequency bands) hence even more latency can be expected in our case. For these reasons, we next explore pipelines which skip the spectral pre-processing stage as described below.

VIII. Classification Based on Time-series

Contrary to the models described in the previous section and inspired by the success of neural feature extraction in the computer vision domain [41], the pipelines we aim at next take directly the raw audio signal time-series as input. More precisely, we designed the three models described below. We inserted the dropouts before average pooling to gain a more stable output [45].

CNN-Time – This simple model performs feature extraction with two sequential (1D) temporal convolutional layers followed by average pooling. A max pooling is inserted in-between to reduce the dimension of feature maps and enhance the non-linearity of the network. Two fully connected layers act as classifier at the end of the network, with the probability outputs normalized by Softmax function.

Transformer-Time – Inspired by [46], we designed an efficient, attention-based model with only one temporal convolutional layer and one single-head transformer. The convolutional and pooling layers serve as feature extraction to transfer raw audio signal into embeddings for the transformer. The activations of the fully connected layer are normalized by Softmax function as well.

TABLE II
TinyML Models: Structure and Characteristics.

Model	Layer	Input Shape	Output Shape
CNN-Mel (25.6K parameters)	3×3 Conv2D + ReLU	$184 \times 80 \times 1$	$182 \times 78 \times 4$
	MaxPooling	$182 \times 78 \times 4$	$91 \times 39 \times 4$
	3×3 Conv2D + ReLU	$91 \times 39 \times 4$	$89 \times 37 \times 4$
	MaxPooling	$89 \times 37 \times 4$	$44 \times 18 \times 4$
	Reshape	$44 \times 18 \times 4$	3168×1
	FC + ReLU	3168×1	8×1
	FC + Softmax	8×1	2×1
SqueezeNet-Mel (727K parameters)	Same as SqueezeNet [42]; Input and output layer are tailored to fit the data.	$184 \times 80 \times 1$	2×1
CNN-Time (748 parameters)	3×1 Conv1D + ReLU	1×48000	4×48000
	MaxPooling	4×48000	4×24000
	3×1 Conv1D	4×24000	8×24000
	Dropout 0.25	8×12000	8×12000
	Average Pooling	8×12000	8×1
	FC + ReLU	8×1	64×1
	FC + Softmax	64×1	2×1
Transformer-Time (1.6K parameters)	Conv1D + ReLU	1×48000	16×48000
	MaxPooling	16×48000	16×24000
	Dropout 0.25	16×48000	16×24000
	Average Pooling	16×24000	16×1
	SingleHeadTransformer	16×1	16×1
	FC + Softmax	16×1	2×1
SqueezeNet-Time (31.1K parameters)	SqueezeNet as backbone; Conv2D → Conv1D; reduce filter number by 70%.	1×48000	2×1

TABLE III
Lightweight variant of the Fire module of 1D SqueezeNet. The number of filters in original SqueezeNet is represented by x.

Name	Layer	Number of filters
Squeeze 1×1	Conv1D	$3 \times \lfloor 0.3 \times x \rfloor$
Expand 1×1	Conv1D	$4 \times \lfloor 0.3 \times x \rfloor$
Expand 3×1	Conv1D	$4 \times \lfloor 0.3 \times x \rfloor$

SqueezeNet-Time – This model is based on SqueezeNet [42]. We tailored this basis to align the input shape with the audio segment, and narrowed the output channels for binary classification. To adapt the backbone to time-series input, all 2D convolutional layers are replaced by temporal convolutional layers. Furthermore, to have a more compact structure, we decreased the filter number of all convolutional layers by roughly 70%, as presented in Table III.

IX. Additional Model Optimizations for TinyML

In this study, we used two optimization techniques to compress the models and further reduce their memory consumption: model quantization on the one hand and on the other hand partial convolution, as described below.

A. Model Quantization

Quantization reduces the model size and inference time by converting the weights and activations from higher precision (e.g., 32-bit floating-point) to lower precision (e.g., 8-bit integer) [47]. In this study we adopted Post-Training Quantization (PTQ) with weights and activations quantized in 8-bit integer. To avoid substantial loss of prediction performance, the training dataset was used to find the optimal scale factor and zero-point of the activations [48].

B. Partial Convolution

We observed that the outputs of the Conv1D layer require significant memory, which impedes deployment on resource-constrained tiny devices. As presented in Table II, the peak memory consumption occurs at the first Conv1D layer both in CNN-Time and Transformer-Time with 768 kB and 3 MB, respectively. This indicates that deployment on resource-constrained tiny devices is impractical.

To address this issue and inspired by [49], we exploited the fact that average pooling can be computed iteratively point-by-point on the output channels of the final Conv1D layer over a small sliding window of inputs. Each point in the output channels depends only on a small subset (kernel window) of the outputs from the previous layer. That is, for a block of L Conv1D layers following by an average pooling layer, its output y can be iteratively computed on the input sequence $x(n), n = 1 \dots N$ as following:

$$y_j(k) = y_j(k-1) + \frac{1}{N_L} A_j^L(k), \qquad (2)$$
$$k = 1 \dots N_L, j = 1 \dots C_L.$$
$$A_c^l(n) = \sum_{i=1}^{C_{l-1}} W_c^l(i) \cdot [A_i^{l-1}(n - \frac{K^l}{2}) \cdots \qquad (3)$$
$$A_i^{l-1}(n) \cdots A_i^{l-1}(n + \frac{K^l}{2})]^T$$
$$A^0(n) = x(n), \qquad (4)$$
$$l = 1 \dots L, c = 1 \dots C_l, n = 1 \dots N_l, N_0 = N$$

where $y_j(N_L)$ is the result of the average pooling of the j-th channel; $A_c^l(n)$ denotes the n-th point in channel c of the l-th Conv1D layer calculated with kernel weights W. Each Conv1D layer contains C_l channels with filter size of K^l and output size of N_l. Without loss of generality, the formula of partial convolution contains only the linear components and considers only one-dimension case for the sake of simplicity; it can be generalized in high-dimension and integrated with non-linear building blocks (e.g., stride, non-linear activations, pooing layers, etc.).

Figure 3 provides a comparison between classic convolution and partial convolution. Unlike classic convolution where entire channels ($C \times N$) are computed and stored before being processed by the next layer, partial convolution requires only a small part of the channels ($C \times K, K << N$) for each layer. In our case with $K = 3$ and $N = 48000$, it can be expected roughly $16000\times$ smaller memory consumption, making the implementation more suitable for tiny devices. We applied this strategy to the Conv1D layers combined with average pooling in the CNN-Time and Transformer-Time architectures.

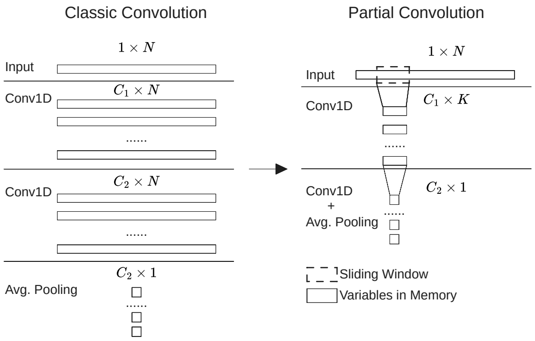

Fig. 3. Partial convolution reduces memory consumption by applying small sliding window on inputs.

X. TinyChirp Performance Evaluation

In the following, we conducted a comprehensive evaluation of the discriminative performance of TinyChirp with different TinyML models, as well as of resource consumption and computation time on typical low-power boards based on microcontrollers.

A. Classification Performance Comparison

Our evaluation process began with the generation of ROC curves and the corresponding Area Under the Curves (AUCs) on the training dataset, providing a clear picture of the models' abilities to distinguish between classes. As shown in Figure 4, the spectrogram-based models – CNN-Mel and SqueezeNet-Mel – achieved the best classification performance with AUCs of 1.0 and 0.99, respectively, followed by time-series models – CNN-Time and Transformer-Time – both with 0.98 AUC. Unexpectedly, with a more complex structure, SqueezeNet-Time performs worst (lower AUC than the baseline).

We next measured accuracy, precision, recall, F_1- and F_2-score, shown in Figure 5. Spectrogram-based models achieve the highest metrics for the whole range of threshold values. For time-series models, Transformer-Time worked overall better than CNN-Time; Again, SqueezeNet-Time performs worst (consistent with the above ROC analysis).

Next, we focused on finding the threshold settings for optimal F_2-score prioritizing recall, i.e. reducing the likelihood of mistakenly discarding records of the targeted bird). The results are shown in Table IV for training data, thereafter verified on test (previously unseen) data as shown in Table V. Note that this threshold influences energy consumption: more target bird classification leads to more energy consumption (as data needs to we stored on to SD card in this case).

These results are promising and allowing to focus only on the relevant audio segments (10% of the total recording time), as the classification performance is compelling. Thus, TinyChirp can save 90% of SD card space compared to the initial monitoring scenario (Section III-A), potentially extending

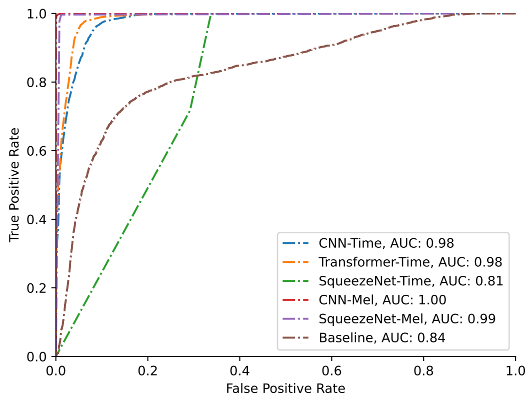

Fig. 4. ROC curves of TinyML models and the corresponding AUCs, evaluating on training dataset.

TABLE IV
MODEL THRESHOLD AND PREDICTION METRICS FOR OPTIMIZED F_2-SCORES, EVALUATING ON TRAINING DATASET.

Model	Threshold (t)	Acc.	Precision	Recall	F_2
Baseline t_{low}	1.00×10^{-7}	0.44	0.38	0.99	0.75
Power-saving t_{high}	1.29×10^{-5}	0.77	0.62	0.80	0.76
SqueezeNet-Mel	0.01	0.99	0.98	1.00	0.99
CNN-Mel	0.27	1.00	0.99	1.00	1.00
CNN-Time	0.23	0.92	0.82	0.98	0.94
Transformer-Time	0.27	0.95	0.89	0.98	0.96
SqueezeNet-Time	0.01	0.78	0.60	1.00	0.88

TABLE V
EVALUATION ON TEST DATASET USING MODEL THRESHOLD t FOR OPTIMIZED F_2-SCORES.

Model	Accuracy	Precision	Recall	F_2
Baseline t_{low}	0.44	0.37	0.99	0.74
Power-saving t_{high}	0.78	0.62	0.82	0.77
SqueezeNet-Mel	0.99	0.96	1.00	0.99
CNN-Mel	0.99	0.98	0.99	0.99
CNN-Time	0.94	0.86	0.98	0.95
Transformer-Time	0.93	0.90	0.91	0.91
SqueezeNet-Time	0.79	0.61	1.00	0.88

the deployment time to 18 weeks. However, complementary experiments must now be carried out on targeted hardware, so as to evaluate other key metrics: computation time, energy consumption and memory footprint on typical microcontroller-based devices. The next sections focus on that part.

B. Performance Evaluation on Microcontrollers

Experimental Setup – For our measurements, we used a common low-power off-the-shelf board: the nRF52840 Development kit (nrf52840dk). This board is based on an ARM Cortex-M4 processor (core frequency at 64 MHz), with a typical memory budget: 1 MB of Flash memory and 256 kB of RAM. As embedded software base we used RIOT-ML [39]. Throughout the experiments, we monitored the (RAM/Flash) resource footprint using built-in diagnostic tools provided by RIOT-ML and RIOT [50], as well as external measurement equipment as described below.

Energy consumption was measured using an ampermeter to gauge the board's energy efficiency. This involved recording the current draw of the MCU and the external storage system (SD card) in different stages. We used a voltage regulator to supply 3.3 V Direct Current (DC) output and a logger to capture detailed current profiles over time.

Scope of the Measurements – Besides the model inference stage, we also considered the resource footprint of pre-processing during bird song recognition. Pre-processing refers to down-sampling and calculation of Mel-spectrogram as described in Section V-B. Also, the energy consumption in MCU's lowest power mode (idle stage) was measured as the reference of absence of sound with sufficient intensity.

Measurement Results on Microcontrollers – The first striking observation is that SqueezeNet-Time and SqueezeNet-Mel were not deployable due to out-of-memory (OOM) (exceded memory budget) and were thus excluded in this experiment. All other models fit in the constraints of memory and storage budget, even when discounting the memory footprint of the OS and the network stack for wireless communication.

The resource footprint of different stages and models measured on an nrf52840dk board are shown in Table VI. We observe that the model inference stage takes the bulk of the overall energy consumption, as it requires the most process time and MCU in full power mode. As expected, the idle stage requires the least power, as all peripherals and the MCU are in the lowest power (standby) mode (until woken up by sound with sufficient amplitude).

Spectrogram *vs* Time-series – Considering only the prediction performance in Section X-A, it would seem natural to choose CNN-Mel as the best candidate, since it also provides the best inference latency. However, taking pre-processing time into account (which for CNN-Mel includes producing a spectrogram) shows a different picture. It takes roughly 2.4 seconds in total for a 3-second audio snippet, which is dangerously near the real-time constraint and costs high power consumption. With the best prediction performance and the lowest total compute latency among time-series models, the Transformer-Time now appears like the best choice to deploy on tiny devices.

Next, we consider jointly the overall resource consumption and the classification performance to discuss variants for the decision strategy described in Section VI-B:

Baseline Only – With the *Baseline* t_{low} (Step 1 and Step 3 only) setting, the required computational resources are minimal, while achieving a recall of 0.99; however, due to the low precision, many false positives are stored, leading to the lowest storage efficiency.

TinyChirp Skipping Baseline – With this variant, the TinyML model is applied immediately, without pre-screening by the baseline (Step 2 and Step 3 Only), leading to high predictive performance and high storage efficiency, but requiring higher energy consumption. Even when using the Transformer-Time with the lowest energy footprint, this requires more than

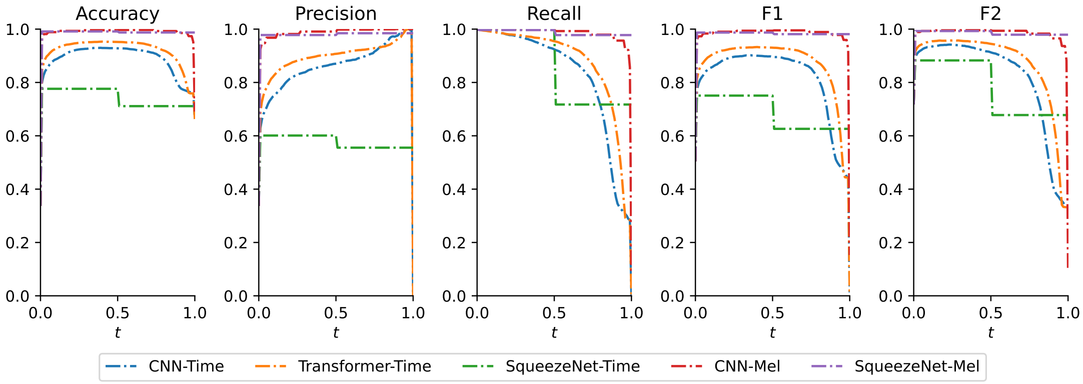

Fig. 5. Accuracy, recall, precision, F-scores of TinyML models on different threshold values t, evaluating on training dataset.

TABLE VI
RESOURCE CONSUMPTION OF TINYML MODELS ON NRF52840DK BOARD.

Model / Stage	Memory (kB)	Storage (kB)	Latency (ms)		Power (mW)	Energy (mJ)	
			Inference	Preprocess		Inference	Total
Baseline	67.216	20.34	213.755	2.0	9.900	2.116	2.136
CNN-Mel	104.328	37.868	406.146	1980.259	17.820	7.238	42.525
SqueezeNet-Mel	OOM	-	-	-	-	-	
CNN-Time	75.564	24.104	1490.687	2.0	17.160	25.580	25.614
Transformer-Time	83.468	24.712	1079.293	2.0	17.820	19.233	19.268
SqueezeNet-Time	OOM						
Idle	-	-	-	-	6.270	-	
OS with Network Stack	7.556	35.808	-	-	-	-	

9 times the energy required for the baseline. This variant is ideal for devices currently running out of storage space while still with substantial battery left.

TinyChirp – This variant refers to the full series of Step 1-3 described in Section VI-B, whereby the samples not discarded by the baseline are double-checked by the TinyML model. This approach offers a good compromise of compute, storage and energy footprint, by discarding early, without impacting too much the overall prediction performance achievable by the TinyML model on its own.

TinyChirp with Power-saving – This variant (Step 1-3 with the power-saving flag enabled) provides a more energy-efficient twist to TinyChirp. However, the overall precision is determined by the threshold setting t_{high}, which causes an extra 40% of false positives to be stored onto the SD card, leading to lower storage efficiency. This variant seems more suitable for currently running out of battery, but with still sufficient storage resources.

XI. Conclusion

In this paper we present an empirical study of a bioacoustic monitoring use case showing that machine learning is usable directly on autonomous recording units based on low-power microcontrollers. We focused on a concrete use-case: monitoring Corn Bunting birds in rural UK. We publish a data set of the recordings, based on which we demonstrated that TinyML can perform on-device high accuracy classification for bird song classification. Compared to the traditional approach used so far, our approach TinyChirp can extend from 2 weeks to 18 weeks – almost a full season – the time intervals until which researchers must harvest overfull data storage embedded on the recording units deployed in the field. Our experiments also observe hands-on how performing inference directly on the audio time-series is the better approach on microcontroller-based hardware, compared to inference on mel-sepctrograms typically used in audio pattern recognition. Looking beyond monitoring Corn Bunting birds' songs, TinyChirp is applicable to a wider range of similar bioacoustic use cases using a large variety of low-power microcontroller-based hardware, based on our general-purpose TinyML approach and our open-source code implementation integrated into the RIOT operating system.

Acknowledgment

The research leading to these results partly received funding from the MESRI-BMBF German/French cybersecurity program under grant agreements No. ANR-20-CYAL-0005 and 16KIS1395K. We acknowledge the receipt of media from the Macaulay Library at the Cornell Lab of Ornithology and the Xeno-Canto database. Felix Bießmann received support from the Einstein Center Digital Future, Berlin, the German Research Foundation (DFG) (528483508 - FIP 12), the Federal Ministry of Economic Affairs and Climate Action (RIWWER 01MD22007H), the German Federal Ministry of the Environment (GCA 67KI2022A) and the German Federal Ministry of Education and Research (KIP-SDM 16SV8856).

References

[1] C. Bormann *et al.*, "Terminology for Constrained-Node Networks," RFC 7228, May 2014.

[2] O. Hahm *et al.*, "Operating Systems for Low-end devices in the Internet of Things: a survey," *IEEE Internet of Things Journal*, 2015.

[3] S. Kahl *et al.*, "BirdNET: A deep learning solution for avian diversity monitoring," *Ecological Informatics*, vol. 61, p. 101236, Mar. 2021.

[4] S. Disabato *et al.*, "Birdsong Detection at the Edge with Deep Learning," in *IEEE SMARTCOMP*. Irvine, CA, USA: IEEE, Aug. 2021, pp. 9–16.

[5] J. Xie *et al.*, "Investigation of Different CNN-Based Models for Improved Bird Sound Classification," *IEEE Access*, 2019.

[6] H. Xiao *et al.*, "AMResNet: An automatic recognition model of bird sounds in real environment," *Applied Acoustics*, Dec. 2022.

[7] M. T. García-Ordás *et al.*, "Multispecies bird sound recognition using a fully convolutional neural network," *Applied Intelligence*, Oct. 2023.

[8] S. Hu *et al.*, "Deep learning bird song recognition based on MFF-ScSEnet," *Ecological Indicators*, vol. 154, p. 110844, Oct. 2023.

[9] S. Zhang *et al.*, "A Novel Bird Sound Recognition Method Based on Multifeature Fusion and a Transformer Encoder," *Sensors*, Sep. 2023.

[10] D. Stowell *et al.*, "Automatic acoustic detection of birds through deep learning: the first Bird Audio Detection challenge," *Methods in Ecology and Evolution*, Mar. 2019, arXiv:1807.05812 [cs, eess].

[11] A. Selin *et al.*, "Wavelets in Recognition of Bird Sounds," *EURASIP Journal on Advances in Signal Processing*, Dec. 2006.

[12] M. Hagiwara *et al.*, "Beans: The benchmark of animal sounds," 2022.

[13] M. Maayah *et al.*, "LimitAccess: on-device TinyML based robust speech recognition and age classification," *Discover Artificial Intelligence*, vol. 3, no. 1, p. 8, Feb. 2023.

[14] O. Atanane *et al.*, "Smart Buildings: Water Leakage Detection Using TinyML," *Sensors*, vol. 23, no. 22, p. 9210, Nov. 2023.

[15] G. Donati *et al.*, "Tiny Deep Learning Architectures Enabling Sensor-Near Acoustic Data Processing and Defect Localization," *Computers*, vol. 12, no. 7, p. 129, Jun. 2023.

[16] K. Fang *et al.*, "A Fall Detection using Sound Technology Based on TinyML," in *Proceedings of ITME*, Wuyishan, China, Nov. 2021.

[17] M. S. Hussain *et al.*, "SwishNet: A Fast Convolutional Neural Network for Speech, Music and Noise Classification and Segmentation," 2018.

[18] F. Zhu-Zhou *et al.*, "Computationally constrained audio-based violence detection through transfer learning and data augmentation techniques," *Applied Acoustics*, vol. 213, p. 109638, Oct. 2023.

[19] Jinyang Yu *et al.*, "Tiny Audio Spectrogram Transformer: Mobilevit for Low-Complexity Acoustic Scene Classification with Decoupled Knowledge Distillation," 2023.

[20] P. Busia *et al.*, "A Tiny Transformer for Low-Power Arrhythmia Classification on Microcontrollers," 2024.

[21] Y. Liang *et al.*, "MCUFormer: Deploying Vision Transformers on Microcontrollers with Limited Memory," 2023.

[22] Z. Yao *et al.*, "A CNN-Transformer Deep Learning Model for Real-time Sleep Stage Classification in an Energy-Constrained Wireless Device *," in *2023 11th International IEEE/EMBS Conference on Neural Engineering (NER)*. Baltimore, MD, USA: IEEE, Apr. 2023, pp. 1–4.

[23] S. Wyatt *et al.*, "Environmental Sound Classification with Tiny Transformers in Noisy Edge Environments," in *Proceedings of WF-IoT*, Jun. 2021, pp. 309–314.

[24] G. Xiao *et al.*, "SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models," 2022.

[25] J. Park *et al.*, "Urban Noise Analysis and Emergency Detection System using Lightweight End-to-End Convolutional Neural Network," *International Journal of Computers Communications & Control*, vol. 18, no. 5, Aug. 2023.

[26] I. Choi *et al.*, "Reducing Energy Consumption and Health Hazards of Electric Liquid Mosquito Repellents through TinyML," *Sensors*, vol. 22, no. 17, p. 6421, Jan. 2022.

[27] S. Abdoli *et al.*, "End-to-end environmental sound classification using a 1D convolutional neural network," *Expert Systems with Applications*, vol. 136, pp. 252–263, Dec. 2019.

[28] A. Sawhney *et al.*, "Latent feature extraction for musical genres from raw audio," in *Proceedings of the 32nd Conference on Neural Information Processing Systems (NIPS 2018)*, 2021, pp. 2–8.

[29] Shalini Mukhopadhyay *et al.*, "Time Series Classification on Edge with Lightweight Attention Networks," in *2024 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, Mar. 2024, pp. 487–492.

[30] J. Sang *et al.*, "Convolutional Recurrent Neural Networks for Urban Sound Classification Using Raw Waveforms," in *2018 26th European Signal Processing Conference (EUSIPCO)*, Sep. 2018, pp. 2444–2448.

[31] L. Turchet *et al.*, "The internet of sounds: Convergent trends, insights, and future directions," *IEEE Internet of Things Journal*, vol. 10, no. 13, pp. 11 264–11 292, 2023.

[32] L. Schulthess *et al.*, "Tinybird-ml: An ultra-low power smart sensor node for bird vocalization analysis and syllable classification," in *2023 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2023, pp. 1–5.

[33] J. Miquel *et al.*, "Energy-efficient audio processing at the edge for biologging applications," *Journal of Low Power Electronics and Applications*, vol. 13, no. 2, p. 30, 2023.

[34] T. Chen *et al.*, "TVM: An automated End-to-End optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 578–594.

[35] C. R. Banbury *et al.*, "Benchmarking tinyml systems: Challenges and direction," *arXiv preprint arXiv:2003.04821*, 2020.

[36] F. M. Polo *et al.*, "tinyBenchmarks: Evaluating LLMs with fewer examples," *Preprint arXiv:2402.14992*, Feb. 2024.

[37] C. Banbury *et al.*, "Mlperf tiny benchmark," *arXiv preprint arXiv:2106.07597*, 2021.

[38] Z. Huang *et al.*, "U-TOE: Universal TinyML On-Board Evaluation Toolkit for Low-Power IoT," in *Proceedings of IFIP/IEEE PEMWN*, 2023.

[39] Z. Huang *et al.*, "RIOT-ML: toolkit for over-the-air secure updates and performance evaluation of TinyML models," *Annals of Telecommunications*, pp. 1–15, 2024.

[40] L. Zandberg *et al.*, "SongBeam: an automated recorder using beamforming to make high-quality recordings," *Open Science Framework Preprint*, 2023.

[41] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.

[42] F. N. Iandola *et al.*, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 mb model size," *Preprint arXiv:1602.07360*, 2016.

[43] J. O. Nordby, "Environmental sound classification on microcontrollers using convolutional neural networks," Master's thesis, Norwegian University of Life Sciences, 2019.

[44] J. Wang *et al.*, "Keyword spotting system and evaluation of pruning and quantization methods on low-power edge microcontrollers," *arXiv preprint arXiv:2208.02765*, 2022.

[45] B. J. Kim *et al.*, "How to use dropout correctly on residual networks with batch normalization," in *PMLR*, 2023.

[46] D. Elliott *et al.*, "Tiny transformers for environmental sound classification at the edge," *arXiv preprint arXiv:2103.12157*, 2021.

[47] M. Nagel *et al.*, "A white paper on neural network quantization," *arXiv preprint arXiv:2106.08295*, 2021.

[48] B. Jacob *et al.*, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[49] H. Pinckaers *et al.*, "Streaming convolutional neural networks for end-to-end learning with multi-megapixel images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[50] E. Baccelli *et al.*, "RIOT: An open source operating system for low-end embedded devices in the IoT," *IEEE Internet of Things Journal*, 2018.