

Learning Energy-Efficient Filterbanks for On-device Bioacoustic Classification

Yasmine Benhamadi

Nantes Université, École Centrale
Nantes, CNRS, LS2N, UMR 6004,
F-44000 Nantes, France
Yasmine.Benhamadi@ls2n.fr

Vincent Lostanlen

Nantes Université, École Centrale
Nantes, CNRS, LS2N, UMR 6004,
F-44000 Nantes, France
vincent.lostanlen@ls2n.fr

Mathieu Lagrange

Nantes Université, École Centrale
Nantes, CNRS, LS2N, UMR 6004,
F-44000 Nantes, France
mathieu.lagrange@ls2n.fr

Abstract—Energy efficiency is essential for enabling more autonomous and sustainable passive acoustic monitoring (PAM). Yet, in the context of on-device bioacoustic classification, the short-time Fourier transform (STFT) accounts for a substantial fraction of the inference pipeline’s energy consumption. In this article, we demonstrate that raw-audio processing using convolutional layers can reduce the computational cost of feature extraction by up to $3\times$ compared with STFT-based processing. We design a parametric filterbank frontend based on real-valued Gabor filters (inspired by LEAF) and optimize it for the target hardware through hardware-informed structured pruning and mixed-precision quantization with bit slicing. For a realistic task of Yellowhammer song type classification on microcontroller hardware, specifically the Apollo510 and ESP32-S3, our measurements show pipeline-level energy reductions of around 47.41% and 23.82%, respectively, corresponding to pipeline speedups of $1.66\times$ and $1.35\times$, while maintaining a competitive F_1 -score. This work demonstrates the potential of 1D convolutional frontends for energy-efficient on-device audio processing and highlights the importance of hardware-aware inference pipeline design.

Index Terms—Audio frontends, bioacoustics, embedded AI, microcontroller, time–frequency representation, on-device inference.

I. INTRODUCTION

Passive acoustic monitoring (PAM) relies on autonomous recording units to capture and store animal sounds over extended periods of time [11], [18], [27]. Such systems are part of the Internet of Sounds (IoS) paradigm, and more specifically the Internet of Audio Things, which involves connected devices for acquiring, processing, and communicating audio data [36], [37]. The need for real-time monitoring, along with considerations of privacy, uptime, and e-waste [2], [4], [9], motivates a shift towards an edge paradigm, where computation is performed directly on-device. In this context, “tiny machine learning” (TinyML) aims to deploy statistical models on resource-constrained hardware, with applications in sound event detection and classification. Recently, multiple research teams have demonstrated the feasibility of bioacoustic species classification using TinyML [1], [14], [33], [40].

State-of-the-art TinyML models for audio do not operate directly on the raw audio waveform. Instead, they depend on a

time–frequency representation as feature extractor: e.g., a mel-frequency spectrogram or mel-frequency cepstral coefficients (MFCC). This is unlike TinyML for computer vision, which operates directly on RGB pixels. Accomplishing the same for audio requires to make the time–frequency representation learnable, thereby replacing mel-based feature engineering by feature learning. However, the current generation of feature learning systems for raw audio tend to underperform feature engineering, particularly with limited labeled data. Moreover, since these systems rely on fast Fourier transforms (FFT) with floating-point arithmetic, they are unsuitable for tiny hardware, where such numerical routines are computationally expensive.

In this article, we address the problem of training a TinyML model for bioacoustic classification and deploying it on low-cost and low-resource hardware. The novelty of our approach is that our model is trained in an “end-to-end” fashion, i.e., without feature extraction. The first layer of our model is a parametric filterbank which consists of real Gabor filters with learnable center frequencies and bandwidths. This parametrization is inspired by a well-known model for end-to-end learning on raw audio, known as LEAF [41]. A known limitation of Gabor filters in LEAF is that they have a large finite impulse response (FIR), making them more computationally intensive than the mel-frequency spectrogram itself [32].

Our article circumvents this limitation via two complementary techniques: hardware-aware pruning (Section III-C) and mixed-precision quantization with bit slicing (Section III-D). Applied together, these two techniques accelerate and alleviate raw audio processing with real Gabor filters, surpassing the efficiency of the non-learned mel-frequency spectrogram frontend. To our knowledge, our article presents the first successful implementation of a waveform-based deep learning system for bioacoustic classification on microcontroller hardware.

As an example use case, we consider the problem of song type classification for the Yellowhammer (*Emberiza citrinella*), a passerine bird in the bunting family. This use case is motivated by the fact that Yellowhammer song is a model species for studies on individual vocal variation [29] and geographic variation in birdsong [30]. Hence, the prospect of designing tiny hardware for long-duration PAM surveys has the potential to advance our understanding of vocal learning

and cultural evolution.

Compared to mel spectrograms, our proposed method achieved speedups of $3.23\times$ and $2.59\times$ on the Apollo510 and ESP32-S3, respectively, with a decrease of 0.06 in F1-score. Assuming similar inference backbones, this would translate to overall pipeline speedups of approximately $1.66\times$ and $1.35\times$, respectively. These findings demonstrate that Conv1D-based learnable filterbanks are a promising alternative to conventional feature extraction methods for improving the efficiency of audio inference pipelines.

II. RELATED WORK

A. Audio frontends

Raw audio frontends such as SincNet [31] and LEAF [41] employ time-domain convolutions to implement parameterized band-pass filters that are fully differentiable and optimized through gradient descent. SincNet constrains its filters using sinc functions, whereas LEAF uses Gabor-Gaussian filters. By restricting the space of learnable Conv1D filters, both approaches incorporate prior knowledge about audio signals while guiding optimization toward more meaningful and interpretable representations.

LEAF was evaluated by [8] against mel spectrograms for insect classification and was found to improve performance, particularly when using higher-frequency filters. [10] compares different audio frontends as well as different normalization and compression schemes through an in-depth ablation study, concluding that normalization and compression are the main drivers behind performance. [13], [42] design lightweight architectures based on convolution operations, attention, and feature fusion for bioacoustic tasks that use SincNet and LEAF frontends, respectively. [6] proposes learnable filters for on-device bird species classification, but their method is based on the STFT, relying on a differentiable frequency warping function rather than time-domain filtering.

Some studies explore time-domain convolution on raw audio with the explicit purpose of increased efficiency. [5], [14], [23] use randomly initialized CNNs for this purpose. However, [12], [20] argue that randomly initialized and unconstrained time-domain convolution can be unstable and harder to train. Some prior work has used SincNet [22] for efficient keyword spotting. Others deploy efficient Gabor filters [2] for single-species detection. To the best of our knowledge, no previous work has studied efficient learnable audio frontends for more complex fine-grained bioacoustic tasks. These tasks are particularly challenging because of diverse acoustic environments and the arbitrary distance between vocalizing sources and the recording device.

Another notable work [32] proposed an efficient implementation of the LEAF frontend that utilizes their time-frequency localization property to adjust the kernel size, as well as the stride and the pooling parameters.

B. Quantization

Beyond the conventional *int8* quantization scheme [15], a range of alternative numerical representations have been

TABLE I
MATHEMATICAL NOTATIONS. SEE SECTION III-A ABOUT GABOR FILTERS, III-C ABOUT PRUNING, AND III-D ABOUT BIT SLICING.

Notation	Description
x	audio signal
N	number of filters
g_n	n^{th} real Gabor filter
f_n	n^{th} center frequency
σ_n	n^{th} temporal width
b_n	n^{th} bandwidth
L	finite impulse response (FIR) length, i.e., kernel length
L_{pruned}	kernel length after pruning
$x^{(i16)}$	signed 16-bit integer representation of the audio signal
$x_{\text{MSB}}^{(i8)}$	most significant byte of the audio signal, interpreted as a signed 8-bit integer
$x_{\text{LSB}}^{(ui8)}$	least significant byte of the audio signal, interpreted as an unsigned 8-bit integer
ϵ	shift error

explored, including low-bit quantization, in which values are represented using 4, 3, or even 2 bits [3], as well as higher-precision formats such as 16-bit integer and 16-bit floating-point representations.

Different network layers, features, and data representations may require different levels of numerical precision. Mixed-precision (or multiprecision) quantization addresses this by assigning different bit widths to different layers or parts of the computation. Existing approaches adapt precision based on either hardware or data characteristics. For example, [39] uses reinforcement learning to determine layer-wise bit widths in a hardware-aware manner. In contrast, [7] proposes a data-aware quantization scheme in which most feature dimensions are represented using 8-bit integers, while outlier dimensions with a larger dynamic range are stored in 16-bit precision.

An alternative approach to mixed-precision quantization is based on bit decomposition and fusion. Rather than representing values using a fixed bit width, [35] decomposes each quantized value into its constituent binary bits, which can then be selectively fused to achieve the desired numerical precision. This enables flexible multi-precision inference while reusing the same underlying binary computations.

Similar ideas have also been explored at the hardware level. [17] proposes a DNN accelerator that decomposes 8-bit operands into lower-bit slices to exploit data sparsity and skip unnecessary multiply-accumulate (MAC) operations during matrix multiplication. Likewise, [34] presents a bit-level accelerator that dynamically composes processing elements to support different operand precisions through bit fusion. At the arithmetic level, [24] proposes a multiplier architecture that partitions 16-bit operands into two 8-bit segments corresponding to the most significant bits (MSBs) and least significant bits (LSBs), enabling more efficient computation.

III. METHODS

A. Learnable Real Gabor filterbank

Within a discrete-time scheme ($t \in \mathbb{Z}$), we construct a Gabor filter by multiplying a complex sine wave of frequency f by a Gaussian window of temporal width σ :

$$\varphi[t] = e^{i2\pi ft} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{t^2}{2\sigma^2}}, \quad -\frac{L}{2} < t < \frac{L}{2}, \quad (1)$$

where the odd integer L denotes the chosen finite impulse response (FIR). By juxtaposing $N > 1$ Gabor filters with center frequencies f_n and widths σ_n , we obtain a filterbank which is equivalent to a complex-valued convnet layer of kernel length L and width N . Given an input signal, $\mathbf{x} \in \ell^2(\mathbb{Z})$, the response of the n^{th} filter in the filterbank, here denoted by φ_n , is:

$$(\varphi_n * \mathbf{x})[t] = \sum_{\tau=-\lfloor L/2 \rfloor}^{\lfloor L/2 \rfloor} \varphi[\tau] \mathbf{x}[t - \tau]. \quad (2)$$

Observe that each FIR coefficient $\varphi_n[t]$ is complex-valued and that its real and imaginary parts are differentiable. The learnable audio frontend model (LEAF) takes advantage of this differentiability property in order to backpropagate the gradient of the loss function with respect to the filter parameters f_n and σ_n , by way of the complex-valued activation $(\varphi_n * \mathbf{x})$.

Compared to LEAF, our implementation of a learnable Gabor filterbank contains three important modifications. First, for reasons of numerical efficiency, we only retain the real part of the FIR in Equation 1, thereby replacing the complex sine wave $e^{i2\pi ft}$ by a real sine wave $\cos(2\pi ft)$. Learnable real-valued Gabor filters have first been proposed by [28]. Secondly, we reparametrize the temporal widths σ_n in terms of Fourier bandwidths $b_n = 1/(\sqrt{2\pi}\sigma_n)$, so that b_n and f_n are physically homogeneous. Thirdly, we remove the factor $1/(\sqrt{2\pi}\sigma_n)$ for simplicity. Hence the revised definition:

$$\mathbf{g}_n[t] = \cos(2\pi f_n t) e^{-(\pi b_n t)^2}, \quad -\frac{L}{2} < t < \frac{L}{2}. \quad (3)$$

B. Efficient filterbank analysis

To achieve energy efficiency, it is essential to understand the optimized hardware-specific libraries used in audio processing pipelines. Here, we consider two such libraries: CMSIS-NN for Cortex-M hardware and ESP-NN for ESP32 hardware, respectively. Recent hardware extensions have provided further efficiency for signal processing and NN inference through vector instructions that are often integrated within the execution paths of these tools. We survey two such technologies in the context of filterbank analysis: Helium for the Cortex-M hardware and Xtensa LX7 for the ESP32 hardware, respectively.

1) *Helium and CMSIS-NN*: Arm Helium, also known as M-Profile Vector Extension (MVE), extends Cortex-M via eight 128-bit vector registers and a SIMD (“single instruction, multiple data”) execution model [21]. The CMSIS-NN library implements neural network operations in a way that is optimized for Cortex-M hardware¹. To take advantage of Helium’s

SIMD capabilities, FIR convolution with CMSIS-NN relies on general matrix multiplication (GEMM) routines [19].

As a preliminary benchmark, we implement a 1D convnet layer on a Helium-enabled microcontroller (Apollo510, see Section IV-C) using TensorFlow Lite Micro with CMSIS-NN acceleration. We compare the effect of the kernel size (L), and the number of filters (N), for both *int8* and *int16* data types. Figure 1 summarizes our findings.

As expected, *int16* inference is considerably slower than *int8*, all other things being equal. Moreover, we observe that FIR convolution is particularly efficient, with execution time below 0.5 ms, for short *int8* kernels ($L \leq 61$) regardless of the values of N . Starting from $L = 81$, the convolution time jumps to much higher values starting at 0.7 ms for $N = 4$ up to 9 ms for $N = 128$, all while still being in *int8*. These results point towards two distinct regimes: one in which the convolutional operation is highly optimized, and another in which performance is more modest. When operating in *int8*, the main factor determining which regime we are in appears to be the kernel length L . A plausible explanation is that imposing $L \leq 61$ allows the kernels to fit into Helium vector registers, thereby making the most of SIMD parallelism.

In light of this preliminary benchmark, we propose to optimize real Gabor filters for the Apollo510 by pruning them to a kernel size L_{pruned} such that $L_{\text{pruned}} \leq 61$. The pruning operation will be presented in Section III-C.

2) *ESP-NN and Xtensa LX7 vector instructions*: The ESP32-S3 microcontroller unit (MCU) integrates a dual-core Xtensa LX7, that is, a 32-bit RISC processor architecture. Similarly to CMSIS-NN with Helium, the ESP-NN library takes advantage of the SIMD capabilities of Xtensa LX7 to optimize neural network operations. However, unlike CMSIS-NN, ESP-NN only supports signed *int8* quantization.

As another preliminary benchmark, we implement a 1D convnet layer on an ESP32-S3 using TensorFlow Lite Micro with ESP-NN acceleration. We compare the effect of the kernel size (L) as well as the number of filters (N) and the stride (s), all with the *int8* numeric type. Figure 2 summarizes our findings. Contrary to Apollo510 (see above), there are no distinct “regimes” here. In Apollo510, the change occurs abruptly as soon as L exceeds 61. For the ESP32-S3, however, the effects are more gradual. N is the most influential factor, followed by the stride s , and, to a much lesser extent, the kernel length L .

In light of this preliminary benchmark, we propose to optimize real Gabor filters on the ESP32-S3 by using *int8* and restricting the number of filters N . Kernel pruning can still be used, but it is less necessary here. The problem is that *int8* quantization severely hinders classification accuracy when working with raw audio; we can circumvent this shortcoming via a dedicated low-level optimization known as bit slicing (see Section III-D).

Taken together, our observations show that energy-efficient filterbank analysis must be tailored to the underlying SIMD architecture. Table II summarizes the characteristics of the studied hardware.

¹<https://arm-software.github.io/CMSIS-NN/latest/index.html>

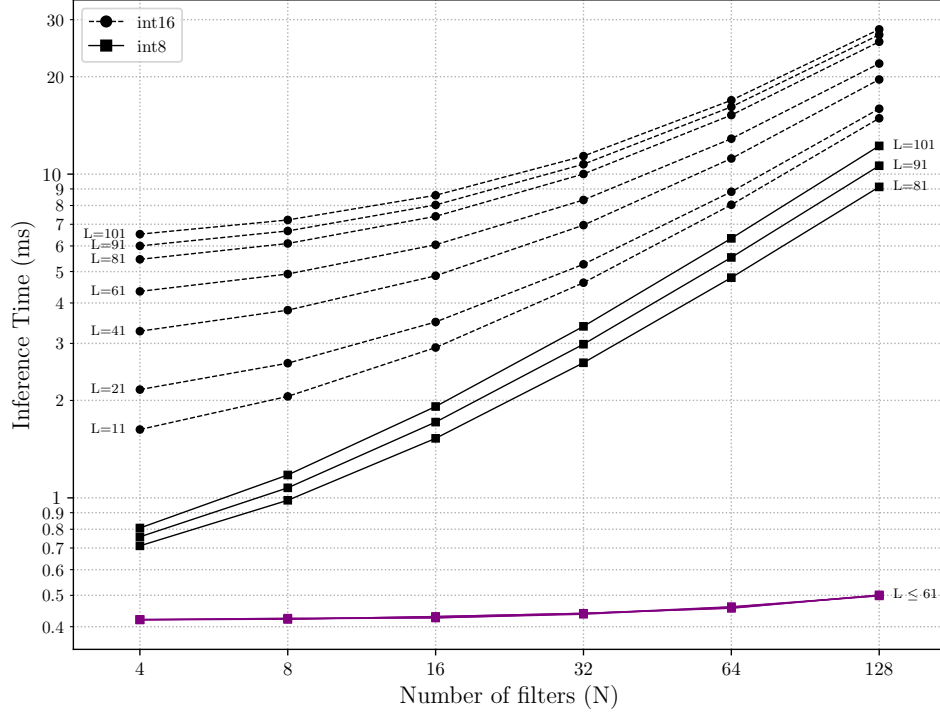


Fig. 1. Execution time (ms) of a 1D convolution layer on Apollo510 as a function of the number of filters (N) for different kernel sizes (L) and quantization schemes. Kernel sizes $L \leq 61$ (purple) exhibit nearly identical performance, with the implementation achieving a peak throughput of approximately 5 GMAC/s.

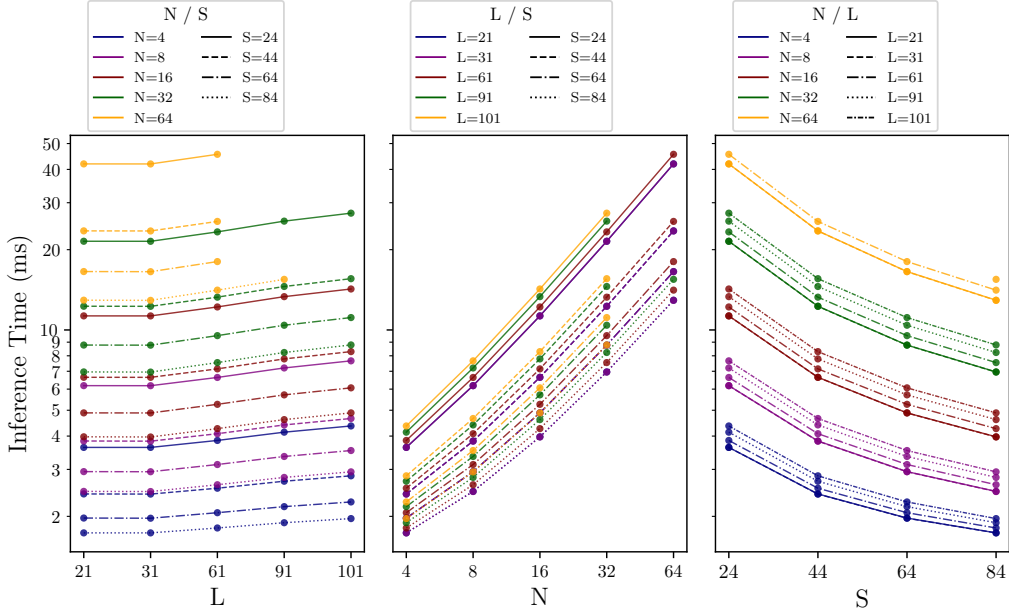


Fig. 2. Execution time (ms) of a 1D convolution layer on the ESP32-S3 as a function of the number of filters (N) for different kernel sizes (L) and strides (s), using *int8* convolution. In contrast to the Apollo510, the kernel size (L) does not appear to be the dominant factor affecting execution time. Instead, the number of filters (N) and the convolution stride (s) have the greatest impact on performance. For large values of L and N combined with a small stride, the convolution scratch buffer exceeds the available memory, and the corresponding measurements are therefore unavailable.

TABLE II

HARDWARE CHARACTERISTICS OF APOLLO510 AND ESP32-S3. THE LAST COLUMN INDICATES WHETHER PRUNING IS REQUIRED OR NOT ON EACH DEVICE. SEE SECTIONS III-B, IV-C FOR DETAILS.

Hardware	Processor	Clock	RAM	NVM	NN library	DSP library	Quantization	Pruning
Apollo510	Arm Cortex-M55	250 MHz	3.75 MB	16 MB	CMSIS-NN	CMSIS-DSP	<i>int8, uint8, int16</i>	✓
ESP32-S3	Dual-core Xtensa LX7	240 MHz	512 KB	16 MB	ESP-NN	ESP-DSP	<i>int8</i>	✗

C. Sparsifying and hardware-aware pruning of Gabor filterbank

The uncertainty principle states that wider localization in frequency correlates with narrow localization in time [28]. This means that the higher the bandwidth, the more localized the filter coefficients become around the center of the kernel and leaving zero values at the edges (see Figure 3). Similarly to [32], we take advantage of this property to perform structured pruning on the kernels, retaining only the non-zero coefficients.

The main difference between us and EfficientLEAF [32] is that on our target hardware, the different kernels benefit from being grouped together. Having inhomogeneous kernel sizes and strides can incur substantial overhead. Because of this, we choose to introduce a constraint on the bandwidths during training. This constraint is informed by empirical results performed on hardware making our training hardware-aware.

As shown in Figure 1, the execution time is highly dependent on the kernel size for Apollo510 and, to a lesser extent, on the ESP32-S3. To make sure we are operating within the accelerated regime, we identify a minimum boundary threshold of 350 Hz that corresponds to the transition kernel size $L_{\max} = 61$. During training, we impose the following constraint on the bandwidths:

$$b_n \geq b_{\min}, \quad \forall n \in \{0, \dots, N-1\},$$

where $b_{\min}$ is set to at least the hardware-defined threshold.

After training, we determined L_{pruned} , the new layer-wise kernel size, according to the kernel with the highest number of non-zero values (see Figure 3).

D. Bit slicing and quantization

Our quantization method presupposes a raw audio $x^{(i16)}$ acquired with a bit depth of 16, which is often the case in many bioacoustic applications. However, in many microcontroller libraries, ESP-NN included, only *int8* optimizations are available. Furthermore, even when kernels in *int16* are available, they tend to be severely (up to 10×) slower compared to the *int8* operations, as is the case for CMSIS-NN (see Figure 1). To solve this issue we use bit slicing, in a similar way to [24]. In our case, we implement this fully in software without any modification to the hardware.

As illustrated in Figure 4, we start by converting the $x^{(i16)}$ into a representation that are compatible with the available operations, which means that they need to be in **signed 8-bit** integer representation. For this purpose, we express $x^{(i16)}$ as follows:

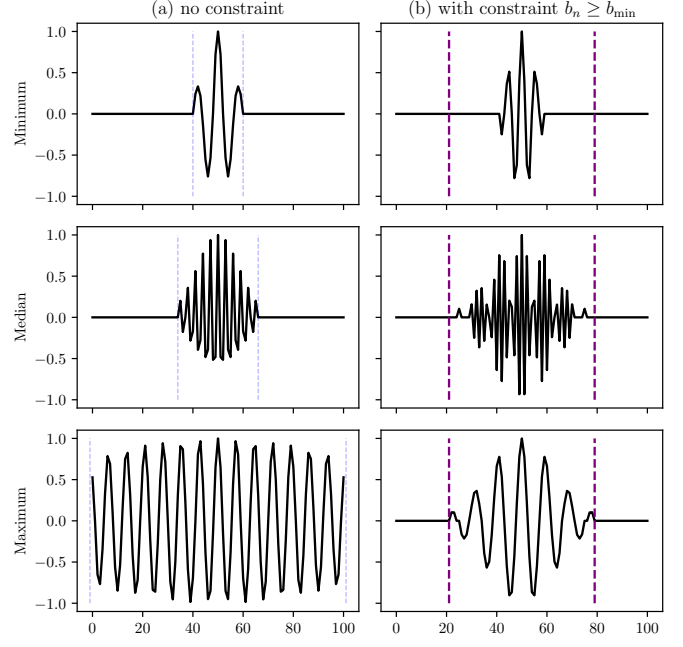


Fig. 3. Minimum (top), median (center), and maximum (bottom) kernel lengths for learned Gabor filterbanks: (a) unconstrained, (b) under the constraint $b_n \geq b_{\min}$. Dashed lines indicate pruning boundaries. In contrast to the per-filter pruning strategy proposed by [32] (a), our approach determines a common pruning boundary from the least sparse filter (b), resulting in a structured filterbank with a uniform kernel length.

$$x^{(i16)} = (x_{\text{MSB}}^{(i8)} \ll 8) + (x_{\text{LSB} \gg 1}^{(i8)} \ll 1) + \varepsilon \quad (4)$$

where

$$x_{\text{LSB} \gg 1}^{(i8)} = x_{\text{LSB}}^{(ui8)} \gg 1,$$

with $\gg$ denoting a right bit shift, $\ll$ denoting a left bit shift, and $\varepsilon \in \{0, 1\}$ representing the error introduced when converting the unsigned LSB to its signed representation.

To quantize the Gabor layer, we create two calibration datasets from our training set, the first containing the MSBs $x_{\text{MSB}}^{(i8)}$ and the other a set of shifted LSBs $(x_{\text{LSB} \gg 1}^{(i8)})$. We use these to get two different output scales and zero points associated with the MSBs, and LSBs respectively. The quantization parameters of the weights and audio input are the same for both.

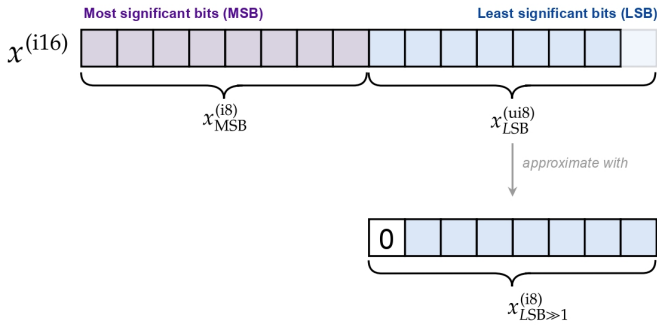


Fig. 4. Bit slicing of a 16-bit signed integer representation into two signed 8-bit integer representations

TABLE III
SUMMARY OF THE PROPOSED CNN ARCHITECTURES.

Mel-2DCNN	Gabor-1DCNN
Mel-spectrogram	Gabor filterbank response
Conv2D (8, 7×7 , stride (1, 4))	Conv1D (64, kernel 7, stride 2)
MaxPool2D (2×2)	MaxPool1D (2)
Conv2D (32, 3×3)	Conv1D (32, kernel 7, stride 2)
MaxPool2D (2×2)	MaxPool1D (8)
Conv2D (64, 3×3)	Flatten
Global Average Pooling	Dense (64, ReLU)
Dropout (0.2)	
Dense (128, ReLU)	

During inference, we calculate the convolution output separately then we reconstruct the output as follows:

$$(x^{(i16)} * g) \approx ((x_{MSB}^{(i8)} * g) \ll 8) + ((x_{LSB \geq 1}^{(i8)} * g) \ll 1) \quad (5)$$

In other words, our decomposition accumulates an error of $\varepsilon * g_n$ at each frequency band n . This error is expected to increase with the kernel length (L), as a larger kernel involves more non-zero terms in the convolution.

E. Backbones: training and quantization

We used a standard CNN architecture for inference. For the models taking Mel spectrograms as input we used a 2D-CNN and for those using Gabor filterbanks, we used a 1D-CNN. Table III describes these CNNs. Both backbones were quantized to *int8*.

IV. EXPERIMENTAL SETUP

Here, we detail our experimental setup for evaluating the predictive performance, latency, and energy consumption of the compared methods. Code is publicly available at: ².

A. Datasets

Our dataset comprises six Yellowhammer song types together with a non-target class. We use the dataset introduced in [25], [26], resampled to 20.48 kHz and segmented into 1.5-second clips containing only the vocalizations of interest.

The recordings were collected as part of an acoustic transmission experiment in which high-quality Yellowhammer

songs were replayed and re-recorded at distances ranging from 6.5 to 200 m in two habitat types: forest and meadow. Overall, the dataset contains approximately 3 h of positive samples and 5.6 h of non-target recordings. The dataset was randomly split between the training, validation and test sets in a 60:20:20 ratio.

Data augmentation: We followed the background noise augmentation used in [25] for ten target SNR values ranging from -15 dB to 30 dB in 5 dB increments.

B. Training, pruning and quantization

A Gabor frontend is first trained entirely in floating-point using Keras. We use 32 filters with a filter length of $L = 101$ and a stride of 64. The center frequencies are initialized on the Mel scale from $f_{\min} = 2$ kHz to $f_{\max} = 10$ kHz, while the bandwidths are initialized randomly. We set $b_{\min} = 500$ Hz. This value is higher than the hardware-defined threshold because it provides better accuracy for the bit-sliced Gabor implementation without affecting the floating-point or other quantized versions. This may be due to the fact that sparser filters reduce the bit-slicing error $\varepsilon * g_n$.

We prune the filters to the maximum length that preserves the non-zero coefficients of the filter with the largest number of non-zero values. As shown in Figure 3, the non-zero values are concentrated near the edges of the filters. We therefore apply a centered truncation to all filters, removing an equal number of coefficients from both ends. This results in filters with a length of $L_{\text{pruned}} = 41$.

After training, the three Gabor frontends are implemented using TensorFlow Lite:

- **Gabor (QINT16):** A Gabor-1DCNN, where Gabor spectrograms are quantized to *int16*. The rest of the backbone is quantized to *int8*.
- **Gabor (QINT8):** A Gabor-1DCNN, where the entire neural network, including the Gabor frontend, is quantized to *int8*.
- **Bit-sliced Gabor:** A Gabor-1DCNN where the frontend is computed using the method proposed in Section III-D. The remainder of the backbone is quantized to *int8*.

The methods are named after their audio features, illustrated in Figure 5, although the evaluation also includes the backbone architectures described in Section III-E.

Baseline: The baseline is a Mel-2DCNN, where mel spectrograms are computed using the floating-point implementations provided by CMSIS-DSP and ESP-DSP and are then fully quantized to an *int8* representation, together with the rest of the backbone. We use a window length of 512 and a hop length of 320.

We report the average F_1 -score as well as its standard deviation across 20 runs.

C. Hardware

Apollo510: The Apollo510 is an ultra-low-power micro-controller developed by Ambiq and based on the Arm Cortex-M55 processor³. The Apollo510 features a Cortex-M55 CPU

²<https://github.com/Yasminebenhamadi/EnergyEfficientFilterbanks>

³Apollo510 specification: <https://ambiq.com/product/apollo510/>

clocked at 250 MHz, 3.75 MB of tightly coupled memory (TCM), and 16 MB of on-chip pseudo-static RAM (PSRAM). The standalone SoC costs approximately \$16, while a complete embedded system based on the closely related Apollo4 platform was reported in [16] to cost less than \$50. We used the Apollo510 Evaluation Board Rev. 2.2⁴. Software development was based on the Apollo SDK together with the NeuralSPOT framework⁵, which provides optimized examples and integration with TensorFlow Lite Micro, CMSIS-NN, and CMSIS-DSP.

ESP32-S3: The ESP32-S3 is a microcontroller based on a dual-core 32-bit Xtensa LX7 processor running at up to 240 MHz⁶. It features 512 KB of on-chip SRAM and, in our configuration, 16 MB of external flash memory. We have used the ESP32-Korvo-2 audio development board. Software development was based on the Espressif IoT Development Framework (ESP-IDF) together with ESP-NN, Espressif’s optimized neural network inference library. We also rely on the code provided by [38] for our evaluation.

Table II summarizes the software libraries, supported quantization formats, and the required optimizations for each hardware platform.

D. Energy measurements

Power consumption of the Apollo510 and ESP32-S3 platforms was measured using Nordic Semiconductor’s Power Profiler Kit II (PPK2). For the Apollo510, the PPK2 was used as an ammeter to measure the current drawn from the 5 V USB power supply. For the ESP32-Korvo-2 development board, the PPK2 was operated as a source meter, supplying the board directly at 5 V while simultaneously measuring its current consumption.

Measurements were repeated ten times, with each measurement lasting 30 s while the code was executed continuously in a while loop. The reported value represents the mean current consumption over each 30 s measurement interval, averaged across all ten measurements.

Execution time was measured separately over 200 runs.

V. RESULTS AND DISCUSSION

Table V shows the classification performance, execution time, and energy consumption measurements for the different frontends. Table IV reports the execution time and energy consumption of the different backbones.

As indicated in Table V, our proposed audio frontend is 3.23× and 2.59× faster than the Mel spectrogram frontend on the Apollo510 and ESP32-S3, respectively. This speedup is accompanied by substantial frontend-level energy savings of 74.62% and 54.91%, respectively. The fully quantized *int8* Gabor frontends achieve larger energy reductions, but they come at a higher cost to the F_1 -score (0.06 instead of 0.18).

To contextualize how the frontend-level savings translate to the full inference pipeline, we distinguish between frontend-level evaluation, which considers only the frontend portion of the inference pipeline, and pipeline-level evaluation which assesses the effect of replacing the frontend on the performance of the complete inference pipeline. Pipeline-level performance depends not only on the frontend-level improvements, but also on the contribution of the baseline frontend (Mel in our case) to the overall pipeline in terms of latency and energy consumption. Table VI summarizes these results assuming that the inference backbone remains unchanged and using the 2D-CNN as the backbone.

Feature extraction accounts for 56.78% of the total inference time on Apollo510 and 42.36% on ESP32-S3, indicating that it constitutes a substantial portion of the overall computational cost (see Table VI). Using Amdahl’s law, the estimated pipeline-level speedup is 1.66× on Apollo510 and 1.35× on ESP32-S3.

Similarly, the pipeline-level energy reductions are derived by combining the frontend energy reduction with the frontend’s contribution to the total pipeline energy. Given that the frontend accounts for 63.21% of the total energy consumption on Apollo510 and 43.32% on ESP32-S3, the estimated pipeline energy reductions are 47.41% and 23.82% for the Mel-2DCNN configuration. In absolute terms, our approach saves 464.85 μJ on Apollo510 and 8161.49 μJ on ESP32-S3.

Pipeline-level improvements are most substantial in cases where the backbone accounts for a smaller proportion of the overall computational and energy cost. Conversely, as in complex tasks requiring computationally intensive backbones, the impact of frontend optimization on the overall pipeline is expected to be more limited. This suggests that as model compression and miniaturization techniques enable increasingly lightweight backbones, frontend optimization could become more relevant, motivating further investigation of joint frontend–backend optimization approaches.

VI. CONCLUSION AND FUTURE WORK

This work addresses the often-overlooked feature extraction stage of audio processing pipelines. We leverage hardware-specific convolution implementations, together with structured pruning and bit-slicing, to implement and deploy efficient real Gabor filterbanks on microcontroller hardware. Compared with a Mel-frequency spectrogram frontend, our bit-sliced Gabor frontend achieves an overall pipeline speedup of 1.66× on the Apollo510 and 1.35× on the ESP32-S3. Assuming the same inference backbone, this translates to an overall pipeline energy reduction of 47.41% and 23.82%, respectively. These findings highlight the importance of end-to-end optimization of embedded audio pipelines.

However, this work primarily focuses on the efficient deployment of audio frontends. Future efforts should extend toward true end-to-end optimization by jointly considering frontend design, quantization strategies, and inference backbones. In particular, the quantization error introduced by bit-slicing could be further mitigated through quantization-aware

⁴Evaluation board: <https://www.ambiq.top/en/apollo510-soc-eval-board>

⁵NeuralSPOT framework: <https://ambiqai.github.io/neuralSPOT/>

⁶Official website: <https://www.espressif.com/en/products/socs/esp32-s3>

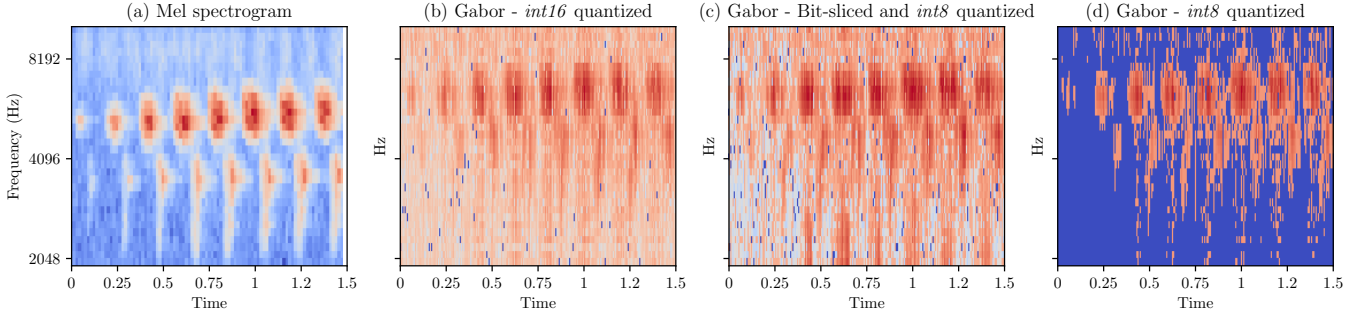


Fig. 5. The four compared time-frequency feature representations shown for a minimum frequency $f_{\min} = 2$ kHz, and a maximum frequency $f_{\max} = 10$ kHz at 20.48 kHz

TABLE IV
INFERENCE TIME, CURRENT CONSUMPTION, ENERGY CONSUMPTION, AND MEMORY ARENA SIZE FOR THE TWO INFERENCE BACKBONES RUNNING IN *int8* ON APOLLO510 AND ESP32-S3.

Model	Apollo510				ESP32-S3			
	Time (ms)	Current (mA)	Energy (μ J)	Arena (kB)	Time (ms)	Current (mA)	Energy (μ J)	Arena (kB)
Mel 2DCNN	4.94	14.60	360.62	13	72.90	53.27	19416.92	41
Gabor 1DCNN	1.94	18.00	174.60	35	64.56	50.63	16343.36	40

TABLE V
CLASSIFICATION PERFORMANCE, INFERENCE TIME, FRONTEND-LEVEL SPEEDUP, CURRENT CONSUMPTION, ENERGY CONSUMPTION, AND FRONTEND-LEVEL ENERGY REDUCTION ON THE APOLLO510 AND ESP32-S3 PLATFORMS. MEL-QINT8 IS USED AS THE BASELINE FOR CALCULATING SPEEDUP AND ENERGY REDUCTION. GABOR-QINT16 RESULTS ARE UNAVAILABLE ON ESP32-S3 BECAUSE ESP-NN SUPPORTS ONLY *int8* QUANTIZATION.

Audio frontend	F_1 -score $\uparrow$	Time (ms) $\downarrow$		Speedup $\uparrow$		Current (mA) $\downarrow$		Energy (μ J) $\downarrow$		Energy reduction $\uparrow$	
		AP510	ESP32	AP510	ESP32	AP510	ESP32	AP510	ESP32	AP510	ESP32
Mel (QINT8)	0.86 ± 0.02	6.49	55.38	—	—	19.10	53.59	619.80	14839.07	—	—
Gabor (QINT16)	0.84 ± 0.01	4.40	—	$1.48\times$	—	14.36	—	315.92	—	49.03%	—
Gabor (QINT8)	0.68 ± 0.07	0.42	9.39	$15.49\times$	$5.90\times$	15.38	63.05	32.22	2960.20	94.80%	80.05%
Bit-sliced Gabor	0.80 ± 0.02	2.01	21.37	$3.23\times$	$2.59\times$	15.65	62.62	157.28	6690.95	74.62%	54.91%

approaches that incorporate the bit-sliced representation during calibration or training, rather than applying it solely as a post-processing step. Furthermore, the design and training of learnable filterbanks—including initialization strategies, dynamic range compression, and optimization procedures—should be revisited to better support resource-constrained deployment. Finally, exploring a broader range of backbone architectures with varying capabilities and design characteristics will be important for understanding the interaction between model architecture, and quantization performance.

ACKNOWLEDGMENT

We thank Ilaria Morandi, Minkyung Kwak, Pavel Linhart and Tereza Petrusková for providing us with the data. We thank Robin Gerzaguët for helpful feedback.

REFERENCES

- [1] Julia Azziz, Josefina Lema, Leonardo Steinfeld, Emiliano Acevedo, and Martín Rocamora. Selective audio recording device for wildlife research using embedded machine learning. In *Proceedings of the IEEE Latin Conference on IoT (LCIoT)*, pages 65–68. IEEE, 2025.
- [2] Yasmine Benhamadi, Antoine Bernabeu, Ilaria Morandi, Minkyung Kwak, Vincent Lostanlen, Pierre-Emmanuel Hladik, Pavel Linhart, Tereza Petrusková, and Mathieu Lagrange. Modeling Battery Lifetime in TinyML-enabled Bioacoustic Surveys: A case study with the AudioMoth sensor and Yellowhammer (*Emberiza citrinella*). HAL preprint hal-05618450, May 2026.
- [3] Yash Bhalgat, Jinwon Lee, Markus Nagel, Tijmen Blankevoort, and Nojun Kwak. LSQ+: Improving Low-Bit Quantization Through Learnable Offsets and Better Initialization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 696–697, 2020.
- [4] Giovanni Carmantini, Yasmine Benhamadi, Matthieu Carreau, Minkyung Kwak, Ilaria Morandi, Friedrich Förstner, Pierre-Emmanuel Hladik, Mathieu Lagrange, Pavel Linhart, Tereza Petrusková, Vincent Lostanlen, and Stefan Kahl. Bioacoustics on tiny hardware at the biodcase 2025 challenge. In *Proceedings of the International Workshop on Detection and Classification of Acoustic Scenes and Events (DCase)*, 2025.
- [5] Debolina Chowdhury, Chawan Vinod, Suman Samui, Mousumi Saha, and Sujoy Saha. Efficient Sound Classification Model using Raw Audio Waveforms in Resource Constrained Devices: Design, Implementation, and Evaluation. In *2026 18th International Conference on Communication Systems and Networks (COMSNETS)*, pages 1238–1241, January 2026.
- [6] Stefano Ciapponi, Leonardo Mannini, Jarek Scanferla, Matteo Anderle, and Elisabetta Farella. Enabling Multi-Species Bird Classification on Low-Power Bioacoustic Loggers. In *Proceedings of the IEEE*

TABLE VI

PIPELINE-LEVEL SPEEDUP AND ENERGY REDUCTION. VALUES ARE DERIVED FROM THE MEASUREMENTS REPORTED IN TABLES IV AND V. THE FINAL ROW INDICATES THE MEL FRONTEND’S CONTRIBUTION TO THE TOTAL PIPELINE INFERENCE TIME AND ENERGY, CALCULATED WITH THE 2D-CNN BACKEND.

Audio frontend	Pipeline speedup ($\times$) $\uparrow$		Pipeline energy reduction (%) $\uparrow$	
	AP510	ESP32	AP510	ESP32
Gabor (QINT16)	1.22 $\times$	–	30.97%	–
Gabor (QINT8)	2.13 $\times$	1.54 $\times$	60.06%	34.65%
Bit-sliced Gabor	1.66 $\times$	1.35 $\times$	47.41%	23.82%
Mel contribution to total pipeline	56.78%	42.36%	63.21%	43.32%

International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 15212–15216, May 2026.

- [7] Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. GPT3.int8(): 8-bit Matrix Multiplication for Transformers at Scale. *Advances in Neural Information Processing Systems (NeurIPS)*, 35:30318–30332, December 2022.
- [8] Marius Fäßl and Dan Stowell. Adaptive representations of sound for automatic insect recognition. *PLOS Computational Biology*, 19(10):e1011541, 2023.
- [9] Leonardo Gabrielli and Luca Turchet. Towards a Sustainable Internet of Sounds. In *Proceedings of the 17th International Audio Mostly Conference*, AM ’22, pages 231–238, New York, NY, USA, October 2022. Association for Computing Machinery.
- [10] Houtan Ghaffari and Paul Devos. On the role of audio frontends in bird species recognition. *Ecological Informatics*, 81:102573, July 2024.
- [11] Rory Gibb, Ella Browning, Paul Glover-Kapfer, and Kate E Jones. Emerging opportunities and challenges for passive acoustics in ecological assessment and monitoring. *Methods in Ecology and Evolution*, 10(2):169–185, 2019.
- [12] Daniel Haider, Vincent Lostanlen, Martin Ehler, and Peter Balazs. Instabilities in convnets for raw audio. *IEEE Signal Processing Letters*, 31:1084–1088, 2024.
- [13] Shipeng Hu, Yihang Chu, Lu Tang, Guoxiong Zhou, Aibin Chen, and Yurong Sun. A lightweight multi-sensory field-based dual-feature fusion residual network for bird song recognition. *Applied Soft Computing*, 146:110678, October 2023.
- [14] Z. Huang, A. Tousnakhoff, P. Kozyr, R. Rehausen, F. Bießmann, R. Lachlan, C. Adjih, and E. Baccelli. TinyChirp: Bird Song Recognition Using TinyML Models on Low-power Wireless Acoustic Sensors. In *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)*, pages 1–10, September 2024.
- [15] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2704–2713, Salt Lake City, UT, June 2018. IEEE.
- [16] Devin Jean, Jesse Turner, Will Hedgecock, György Kalmár, George Wittemyer, and Ákos Lédeczi. Animal-Borne Adaptive Acoustic Monitoring. *Journal of Sensor and Actuator Networks*, 14(4):66, August 2025.
- [17] Dongyun Kam, Myeongji Yun, Sunwoo Yoo, Seungwoo Hong, Zhengya Zhang, and Youngjoo Lee. Panacea: Novel DNN Accelerator using Accuracy-Preserving Asymmetric Quantization and Energy-Saving Bit-Slice Sparsity. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 701–715, March 2025.
- [18] Justin Kitzes and Lauren Schricker. The necessity, promise and challenge of automated biodiversity surveys. *Environmental Conservation*, 46(4):247–250, 2019.
- [19] Liangzhen Lai, Naveen Suda, and Vikas Chandra. CMSIS-NN: Efficient Neural Network Kernels for Arm Cortex-M CPUs. *arXiv preprint arXiv:1801.06601*, 2018.
- [20] Vincent Lostanlen, Daniel Haider, Han Han, Mathieu Lagrange, Peter Balazs, and Martin Ehler. Fitting Auditory Filterbanks with Multiresolution Neural Networks. In *Proceedings of the 2023 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 1–5, October 2023.
- [21] Jon Marsh. *Arm® Helium™ Technology M-Profile Vector Extension (MVE) for Arm® Cortex-M Processors: Reference Book*. Arm Education Media, 2021.
- [22] Simon Mittermaier, Ludwig Kürzinger, Bernd Waschneck, and Gerhard Rigoll. Small-Footprint Keyword Spotting on Raw Audio Data with Sinc-Convolutions. In *Proceedings of the ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7454–7458, May 2020.
- [23] Md Mohaimenuzzaman, Christoph Bergmeier, Ian West, and Bernd Meyer. Environmental Sound Classification on the Edge: A Pipeline for Deep Acoustic Networks on Extremely Resource-Constrained Devices. *Pattern Recognition*, 133:109025, January 2023.
- [24] H Mohammed Adhil, J Dhanasekar, Kp Dhivya Archana, and B Gayathri Devi. Implementation of Hybrid 16x16 Multiplier with Approximate and Accurate Units for Low-Power Applications using Cadence SimVision. In *Proceedings of the International Conference on Electronics and Sustainable Communication Systems (ICESC)*, pages 54–59, September 2025.
- [25] Ilaria Morandi, Manuel Alejandro Jaramillo Rodriguez, Minkyung Kwak, Tereza Petrusková, and Pavel Linhart. Assessing the Effective Range for Individual Acoustic Identification: Comparison of Manual and Automatic Methods. *bioRxiv preprint*, 2025.
- [26] Ilaria Morandi, Pavel Linhart, Minkyung Kwak, and Tereza Petrusková. BioDCASE 2025 Task 3: Bioacoustics for Tiny Hardware Development Set. [dataset], 2025. Zenodo, v1. <https://doi.org/10.5281/zenodo.15228365>.
- [27] Thomas Napier, Euijoon Ahn, Slade Allen-Ankins, Lin Schwarzkopf, and Ickjai Lee. Advancements in preprocessing, detection and classification techniques for ecoacoustic data: a comprehensive review for large-scale passive acoustic monitoring. *Expert Systems with Applications*, 252:124220, 2024.
- [28] Paul-Gauthier Noé, Titouan Parcollet, and Mohamed Morchid. CGCNN: Complex Gabor Convolutional Neural Network on Raw Speech. In *Proceedings of the ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7724–7728, May 2020.
- [29] Anna N Osiecka, Marcos Q Oliva, Jan Kouřil, Tereza Petrusková, and Lara S Burchardt. Yellowhammer (*Emberiza citrinella*) males sing using individual isochronous rhythms and maximise rhythmic dissimilarity with neighbours. *bioRxiv preprint*, 2025.
- [30] Tereza Petrusková, Lucie Diblíková, Pavel Pipek, Eckehard Frauendorf, Petr Procházka, and Adam Petrusek. A review of the distribution of Yellowhammer (*Emberiza citrinella*) dialects in Europe reveals the lack of a clear macrogeographic pattern. *Journal of Ornithology*, 156(1):263–273, 2015.
- [31] Mirco Ravanelli and Yoshua Bengio. Speaker Recognition from Raw Waveform with SincNet. In *Proceedings of the IEEE Spoken Language Technology Workshop (SLT)*, pages 1021–1028, December 2018.
- [32] Jan Schlüter and Gerald Gutenbrunner. EfficientLEAF: A Faster LEarnable Audio Frontend of Questionable Use. In *Proceedings of the European Signal Processing Conference (EUSIPCO)*, pages 205–208, August 2022.
- [33] Lukas Schulthess, Steven Marty, Matilde Dirodi, Mariana D Rocha, Linus Rüttimann, Richard HR Hahnloser, and Michele Magno. TinyBird-ML: An ultra-low power smart sensor node for bird vocalization analysis and syllable classification. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5. IEEE, 2023.

- [34] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Joon Kyung Kim, Vikas Chandra, and Hadi Esmaeilzadeh. Bit Fusion: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network. In *Proceedings of the ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 764–775, June 2018.
- [35] Qigong Sun, Fanhua Shang, Kang Yang, Xiufang Li, Yan Ren, and Licheng Jiao. Multi-Precision Quantized Neural Networks via Encoding Decomposition of $\{-1,+1\}$. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):5024–5032, July 2019.
- [36] Luca Turchet, György Fazekas, Mathieu Lagrange, Hossein S. Ghadikolaei, and Carlo Fischione. The Internet of Audio Things: State of the Art, Vision, and Challenges. *IEEE Internet of Things Journal*, 7(10):10233–10249, October 2020.
- [37] Luca Turchet, Mathieu Lagrange, Cristina Rottondi, György Fazekas, Nils Peters, Jan Østergaard, Frederic Font, Tom Bäckström, and Carlo Fischione. The Internet of Sounds: Convergent Trends, Insights, and Future Directions. *IEEE Internet of Things Journal*, 10(13):11264–11292, July 2023.
- [38] Christian Walter, Yasmine Benhamadi, Tom Seidel, Giovanni Carmanini, and Stefan Kahl. Biodcase-tiny 2026: A framework for bird species recognition on resource-constrained hardware. <https://github.com/birdnet-team/BioDCASE-Tiny-2026>, 2026.
- [39] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-Aware Automated Quantization With Mixed Precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8604–8612, Long Beach, CA, USA, June 2019. IEEE.
- [40] Connor M Wood, Felix Günther, Angela Rex, Daniel F Hofstadter, Hendrik Reers, Stefan Kahl, M Zachariah Peery, and Holger Klinck. Real-time acoustic monitoring facilitates the proactive management of biological invasions. *Biological Invasions*, 26(12):3989–3996, 2024.
- [41] Neil Zeghidour, Olivier Teboul, Félix de Chaumont Quitry, and Marco Tagliasacchi. LEAF: A learnable frontend for audio classification. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.
- [42] Qin Zhang, Shipeng Hu, Lu Tang, Rui Deng, Choujun Yang, Guoxiong Zhou, and Aibin Chen. SDFIE-NET: A self-learning dual-feature fusion information capture expression method for birdsong recognition. *Applied Acoustics*, 221:110004, May 2024.