POLITECNICO DI TORINO
Repository ISTITUZIONALE

Advancing Beekeeping: IoT and TinyML for Queen Bee Monitoring Using Audio Signals

Original
Advancing Beekeeping: IoT and TinyML for Queen Bee Monitoring Using Audio Signals / De Simone, A., Barbisan, L., Turvani, G., Riente, F.. - In: IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT. - ISSN 0018-9456. - 73:(2024), pp. 1-9. [10.1109/tim.2024.3449981]

Availability:
This version is available at: 11583/2995167 since: 2024-12-10T22:27:50Z

Publisher:
IEEE

Published
DOI:10.1109/tim.2024.3449981

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

17 November 2026

Advancing Beekeeping: IoT and TinyML for Queen Bee Monitoring Using Audio Signals

Andrea De Simone, Luca Barbisan, *Student Member, IEEE*, Giovanna Turvani, and Fabrizio Riente, *Member, IEEE*

***Abstract*— Beekeeping plays an essential role in maintaining ecosystems through pollination and enhancing biodiversity. The presence of the queen bee inside the hive is an important indicator for the health of the bee colony. Monitoring the health of honeybees and their hives is crucial not only for bees but also for the entire ecosystem. This article introduces a tiny machine learning (ML) application for edge computing in the Internet-of-Things (IoT) systems, designed to predict the queen bee's presence. The solution, implemented on a low-power microcontroller (MCU), listens to the sound produced by honeybees and aids beekeepers by automating health assessments of the colony. The system utilizes audio recordings of honeybees combined with artificial intelligence (AI) techniques, while the second focuses on optimizing a feature extraction algorithm from these recordings to minimize latency and energy use in the IoT setup. The findings show that despite the implementation of a simpler ML model and audio preprocessing with lower computational precision, the final metrics remain comparable to those analyzed, with only a limited reduction.**

***Index Terms*— Artificial intelligence (AI), Internet of Things (IoT), neural network (NN), sound analysis, tiny machine learning (TinyML).**

I. Introduction

HONEYBEES play a crucial role in the pollination and reproductive cycles of the ecosystem, making them an indispensable part of the environment [1]. However, in recent years, the bee population has experienced a decline, highlighting the fragility and significance of this precious species [2]. Factors, such as habitat loss, pesticide exposure, climate change, and diseases, have contributed to this worrying trend [3]. The prevailing view has been that the wild European honey bee, Apis mellifera, has vanished from natural environments. Even if, recent research indicates that tree cavities in beech forests present a suitable habitat for wild or feral bee populations. Beekeepers play a crucial role in mitigating the impacts of diseases and in supporting the dietary needs of bees by planting flora that provides pollen, propolis, and nectar, and ensuring that water sources are available for colony development [4]. Bee farms are often located in secluded, hard-to-reach locations, necessitating considerable travel for beekeepers to visit and monitor their hives, a challenge that is particularly pronounced in the practice of nomadic beekeeping. These considerations serve as a strong motivation for the creation of innovative approaches designed to assist beekeepers and researchers. The goal is to deepen our comprehension of the factors contributing to the mortality of honey bees and to devise strategies for their preservation [5]. This undertaking is vital for safeguarding the health and sustainability of honey bee populations, ensuring their vital role in pollination and biodiversity is maintained. By addressing these challenges, we can work toward securing a future where the ecological and economic contributions of honey bees are protected and sustained.

Several techniques have been proposed in the literature with the aim of supporting both researchers and beekeepers preserving bees' colonies. Zacepins et al. [6] present one example of wireless system to monitoring environmental parameters. The developed system is based on temperature, sound, and video monitoring that requires the modification of the structure of the beehive to introduce the electronics. On the one side, analyzing audio sound has emerged as a common method for assessing hive health, as the sound produced by buzzing can serve as an indicator of bee activity, as studied in [7]. Machine learning (ML) techniques are often employed to make prediction based on audio sound collected inside the beehive. For instance, in [8], the audio signal is used to detect the presence of the Varroa destructor within the beehive. Swarming activity is also detected by using various approaches as outlined in [9] and [10]. ML has further been used to discriminate the bee sounds from external noise in [11] and [12] or for bee audio classification in [13] and [14]. An important indicator of the health status of the bee colony is the presence of the queen bee inside the hive. As claimed in [15], [16], and [17], ML classifiers can achieve high accuracies, in detecting the queen, close to 99% by extracting coefficients dependent on the frequency spectrum from raw audio collected in the hives. Among the most commonly used coefficients, there are mel-frequency cepstral coefficients (MFCCs), mel-spectrogram, and short-time Fourier transform (STFT).

The aim of this study consists in developing a compact Internet-of-Things (IoT) system based on a low-power MCU capable of detecting the presence of the queen bee from the audio recorded in the beehive through a tiny ML (TinyML) classifier. We think that innovative solutions must be compact, power efficient, and easy to be introduced inside the beehive, without requiring hive modifications. Indeed, the prediction must be transmitted to a cloud dashboard in order to be always

Manuscript received 4 June 2024; revised 26 July 2024; accepted 4 August 2024. Date of publication 26 August 2024; date of current version 6 September 2024. The Associate Editor coordinating the review process was Dr. Benkuan Wang. *(Corresponding author: Fabrizio Riente.)*

The authors are with the Department of Electronics and Telecommunications, Politecnico di Torino, 10129 Torino, Italy (e-mail: fabrizio.riente@polito.it).

Digital Object Identifier 10.1109/TIM.2024.3449981

© 2024 The Authors. This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

consultable. The classifier is executed at the edge on the MCU, necessitating the ML network to be extremely compact to fit within the MCU's memory constraints. Furthermore, the feature extraction process must be streamlined to minimize latency and energy consumption. This approach is preferred, because the bandwidth of the long range (LoRa) protocol is extremely limited, making it impossible to transmit either the entire audio file or a preprocessed version. In addition, this choice aims to minimize the latency of the prediction with a limited power budget. The key contributions of this article include the following: 1) the development of a TinyML model with a minimal number of parameters; 2) the creation of a method to evaluate quantized models across various data representations; 3) development of a fixed-point algorithm for feature extraction in both *int16* and *int32* formats, and comparing their efficiency against the floating-point representation; and 4) measuring the energy usage on a custom designed IoT node for both feature extraction and inference processes at the edge. Our goal is to achieve accuracies comparable with more complex ML models that use larger floating-point precision-based parameters for feature extraction.

The rest of this article is organized as follows. Section II gives an overview of the approaches adopted to detect the queen bee presence. Section III discusses the methodology employed to extract the ML features. Section IV provides a description of the ML model configuration. Section V illustrates the conversion process to execute the inference on the MCU, while Section VI reports the energy requirement of the application. Section VII analyzes the metrics obtained by the tested models. Finally, Section VIII draws conclusions.

II. Background

Raw audio files typically require a preprocessing stage before they can be used to train ML models, and this is due to their high-dimensional nature. In [14], there is an example where raw audio features are directly employed to train a deep neural network (NN), but the number of parameters of the model is on the order of millions. The most commonly employed audio features to forecast the presence of the queen bee are MFCCs combined with convolutional NN (CNN) or other classifiers, such as support vector machine (SVM) or random forest (RF) [18], [19], [20], [21]. Cejrowski et al. [22] adopted linear predictive coding as preprocessing technique in combination with an SVM. Comparison among different extraction methods, such as STFT, MFCCs, and the listed classifiers, is investigated in [15], [16], [17], [23], [24], and [25].

From the related works, we selected the MFCCs feature, which is the most used in the literature, with the addition of STFT, which represents the first step for MFCCs extraction. Regarding the classifier, CNN and SVM are taken into consideration. In particular, the CNN architecture is simplified by removing initial convolutional layers to reduce the size of the network. The chosen MCU supports the LoRaWAN protocol for data transmission. The audio is recorded using a MEMS digital microphone and written on a flash memory. Subsequently, the features are extracted from audio samples.

TABLE I
Dataset Used for Training

Dataset	Number of audio samples	Audio duration	Sampling rate
Dataset I [26]	7100	1 minute	22050 Hz
Dataset II [27]	573	10 minute	22050 Hz

A very short audio recorded at distant intervals allows recognizing a disturbed state inside the beehive, identifying the queen bee's absence. The entire IoT system is designed to be powered by a battery, with a target lifetime of several years, making the system practical for end users.

III. Feature Extraction

The ML model is trained using audio files coming from two distinct datasets openly available: Dataset I [26] and Dataset II [27], as detailed in Table I. These datasets provide audio files annotated with the corresponding labels, indicating the queen bee presence. Given the energy consumption concerns related to the MCU recording operation, we explore audio recordings that are shorter than 7 s for the training process. Moreover, the ML model only needs to make a binary decision. Using such short audio is sufficient to achieve an accuracy higher than 98%. Therefore, longer audio samples are not explored. Consequently, the audio files in the datasets are split to satisfy this constraint (Fig. 1, *"chunk extraction"*).

From the two datasets, STFT and MFCCs features are extracted as outlined in diagram depicted in Fig. 1, *"feature extraction."*

A. STFT

STFT consists in the application of the fast Fourier transform (FFT) algorithm to the audio signal split in short frames. FFT computes the power spectrum from the audio samples, as shown in the following equation:

$$x = \sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi kn}{N}} \tag{1}$$

where N represents the total number of samples to which the algorithm is applied (will be referred to as FFT size or simply FFT). The result of the FFT consists of $N/2 + 1$ complex coefficients related to the positive frequencies and $N/2$ related to the negative frequencies. The complete procedure of STFT extraction, displayed in the upper part of Fig. 2, is summarized by the following steps.

1) *Windowing:* The audio signal is divided into frames, not overlapped, with a number of samples equal to FFT. Each frame is multiplied by the Hann window function.
2) *FFT:* The FFT is used to compute the power spectrum of the samples, and only the positive frequencies are maintained.
3) *Magnitude:* The output of the FFT calculation is complex number, and the magnitude is calculated for each coefficient.

The main tested parameter of STFT extraction is FFT, which determines the frame length and, thus, the number of FFT

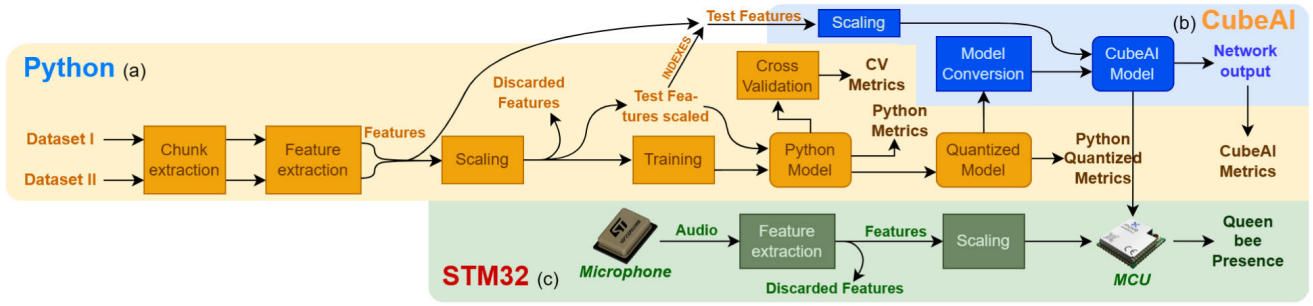


Fig. 1. Diagram of the methodology adopted. (a) Operations performed by Python environment. (b) Operations executed utilizing *stm32ai*. (c) Process on the MCU.

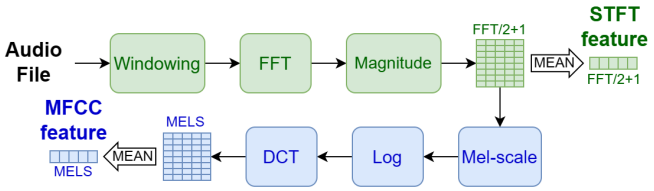


Fig. 2. Extraction diagram for STFT and MFCC features.

operations. Increasing the FFT size reduces the number of windows needed to cover the entire audio file but increases the computation time for a single frame. The STFT outputs are $N/2+1$ coefficients for each frame. So, the final output is a matrix where the first dimension represents the frame number and the second dimension has a size equal to $N/2+1$. To additionally compress features, the average along the frame number axis is computed, resulting in $N/2+1$ features.

B. MFCCs

MFCCs can be computed by extending the STFT algorithm and mapping the results onto a mel-scale followed by compression using discrete cosine transform (DCT). The exact steps required to compute the coefficients, illustrated in Fig. 2, are as follows.

1) *STFT:* The STFT coefficients are computed.
2) *Mel-Scale:* The coefficients are mapped onto the mel-scale, which is based on the equation

$$\text{mel}(f) = 2595 \cdot \log\left(1 + \frac{f}{700}\right). \tag{2}$$

It is possible to choose the frequency range and the number of frequency steps (referred as MELS).
3) *Log:* It is computed in the logarithm in base 10 of the resulting coefficients.
4) *DCT:* The coefficients are compressed by applying the DCT; in this study, the DCT type II is adopted, which is based on the formula

$$X_k = \sqrt{\frac{2}{N}} \cdot \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \quad \text{for } k = 1, \dots N-1 \tag{3}$$

$$X_0 = \frac{1}{\sqrt{N}} \cdot \sum_{n=0}^{N-1} x_n \quad (\text{where } N = \text{MELS}) \quad \text{for } k = 0. \tag{4}$$

TABLE II
CONFIGURATION USED FOR FEATURE EXTRACTION IN LIBROSA

STFT param	*hop_length*	*center*	*pad_mode*	*window*
	<FFT size>	False	constant	hann

MFCC param	*norm*	*lifter*	*f_min*	*f_max*	*power*	*htk*	*dct*
	ortho	0	0	2756	1	True	2

The main tested parameters for MFCC extraction include FFT, which regard the STFT extraction, and MELS. The output shape of the extraction is similar to that of the STFT operation, forming a matrix with the second dimension being MELS. The same compression method is applied to obtain only MELS features at the end.

C. Feature Extraction Deployment on Python and MCU

As a first step, the features are extracted in floating point using the Python library Librosa [28]. Subsequently, the extraction process is transposed into C code in fixed point employing the CMSIS-DSP library [29]. Two data types, *int16* and *int32*, are evaluated. The correspondence of results between the two libraries is achieved with the Librosa configuration detailed in Table II. At the beginning, various tests are conducted to determine the best configuration for FFT and MELS that require less energy to be computed. To estimate the energy consumption, we measured the extraction time for different configurations. The values in Table III include the time required for reading the audio file from the flash memory and extract the correspondent feature. To maintain consistency between the two data types, for the *int32* format, samples are read as *int16* and shifted. For STFT extraction, the optimal FFT size, based on extraction time, is found to be 256, while for MFCC, it is 1024. Increasing MELS from 20 to 30 does not impact time requirements. Higher values of MELS are not tested to limit the complexity of the extraction. The quality of the prediction does not appear to be significantly influenced by the FFT size, as observed in various tests conducted using cross validation (CV) in Python. Consequently, the selection of the FFT is primarily based on the energy consumption requirements.

The coefficients obtained with floating-point computation using Librosa are taken as a reference to evaluate the *int16* and *int32* features. Fig. 3(a) and (b) illustrates the relative

TABLE III
TIME REQUIREMENTS FOR STFT AND MFCC EXTRACTION

MFCC

FFT - MELS	TIME [ms]	
	int16	*int32*
256 - 20	2655.52	3342.44
512 - 20	2655.77	3443.32
512 - 30	2708.36	3496.48
1024 - 20	**2492.67**	**3328.37**
1024 - 30	2545.51	3329.79
2048 - 20	2589.24	3469.33
2048 - 30	2637.69	3511.88

STFT

FFT	TIME [ms]	
	int16	*int32*
128	2500.39	3021.95
256	**2393.93**	**2972.83**
512	2550.24	3183.31
1024	2490.83	3175.07
2048	2588.17	3371.05

error for the STFT features computed on 1000 samples from Dataset II. The points represent the average relative error across the samples, while the vertical bands represent the standard deviation. The difference between the two data types is pronounced for STFT feature. The relative error tends to increase for higher coefficients (associated with higher frequencies), because although the absolute error remains stable, the coefficients' values decrease and eventually become 0 for fixed-point computation (resulting in a relative error of 100%). Therefore, it is preferable to discard high-frequency coefficients to retain only the features computed with greater accuracy. The figures display only low-frequency coefficients to better illustrate the computed errors.

Fig. 3(c) and (d) presents the relative error of MFCC extraction referred to floating-point computation. This analysis is conducted using the same audio files as the previous test. In this scenario, the difference between the two data types is less marked. An aspect that emerges from the analysis is that some coefficients exhibit high standard deviation. This behavior originates from the same underlying cause as that of the STFT coefficients, where integer values become saturated to zero, leading to significant relative errors.

D. Feature Selection and Scaling

The number of features directly impacts the size of the ML model, which influences the flash memory requirements of the MCU and the latency of the inference. Various approaches are available for selecting the features, allowing for the exclusion of features that contribute less to the prediction. Based on the considerations made during fixed-point extraction, the chosen method for feature selection is based on variance. Specifically, features with the lowest variance are discarded. This approach eliminates coefficients related to high frequency, because they are close to 0 and exhibits the lowest variance, maintaining only the coefficients that demonstrate higher accuracy. For MFCC features, 10–20 coefficients are usually sufficient, while for STFT, 40–80 coefficients are required to achieve satisfactory accuracy.

The last operation carried on features is scaling. Features originate from diverse sources and can exhibit varying ranges depending on the frequency. This disparity can negatively impact the performance of the ML model, as high-value features might overshadow those with smaller ranges. One common method, used for this study, involves normalizing features, such that their mean becomes 0, and their standard deviation becomes 1. The *StandardScaler* class provided by

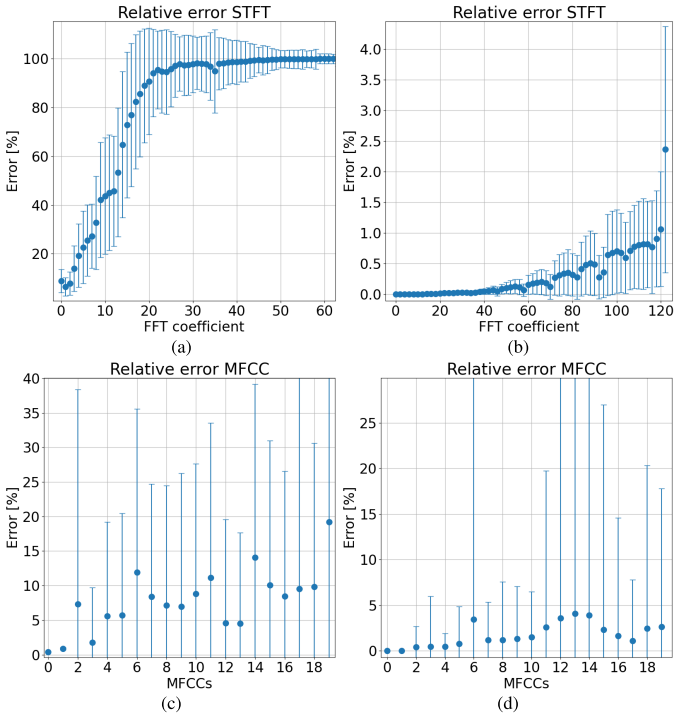


Fig. 3. Relative error for (a) and (b) STFT and (c) and (d) MFCC extracted with CMSIS in *int16* and *int32* compared with *float*, extracted with Librosa. Points and bands represent mean and standard deviation on errors calculated on 1000 audios 5-s long from Dataset II.

the *scikit-learn* library performs this operation. The scaling formula used is

$$z = \frac{x - u}{s} \tag{5}$$

where x is the sample to be scaled, u is the mean, and s is the standard deviation calculated from all the samples of the feature.

IV. ML MODELS

Based on the analysis of previous works [15], [16], [17], [23], [24], [25], the two classifiers considered are SVM and NN. SVM is discarded at the start due to the lack of methods for quantization in conventional libraries. In addition, the memory required to store the parameters necessary for SVM classification typically exceeds the flash memory capacity of the selected MCU. Before starting the training operation, a percentage of the features (20%) are retained and reserved for a final test to compute the metrics of the network (the metrics are listed in Section VII). Initially, a series of tests are conducted in Python to identify which configurations obtain appreciable results. Regarding feature data type, both floating point and fixed point are tested for the training. Feature extraction in Python in fixed point is accomplished using the package *cmsisdsp*, which emulates the computation of the C library.

In this study, NN models consist of three fully connected (FC) layers in sequence. The last layer has only one output between 0 and 1; when the result is above the threshold of 0.5, the presence of the queen bee is inferred. In FC layers, each output is a linear combination of the inputs, and the number

of coefficients (commonly referred to as weights) depends on the number of inputs and outputs of the layer, and the latter can be controlled by the parameter dense size (DS). After each FC layer, a *ReLU* activation function is applied, while for the last layer, the *sigmoid* function is adopted, which is typically employed for binary classification. The ML models are built, trained, and tested using a Python script and the TensorFlow library [30], and the training uses the early stopping technique. The configuration of the model is the following: ***max_epoch***: *500*, ***patience***: *50*, ***batch_size***: *64*, ***learning_rate***: *0.001*, ***optimizer***: *Adam*, and ***loss***: *BinaryCrossentropy*.

The technique employed for assessing the Python models is CV. Even when the same input features and network configuration are chosen, different training operations can lead to varying final models due to the training algorithm or the random initialization of the weights. To achieve an evaluation of a model's performance that is robust and less affected by small random fluctuations, the CV approach is employed. In CV, the dataset is divided into a certain number of folds, often ranging from 5 to 10; for this study, tenfold are chosen. For each fold, a model is trained while using onefold as the validation set and the remaining folds as the training set. This process is repeated for each fold, resulting in multiple models. Afterward, the mean and standard deviation of the metrics of these models are computed.

A. Quantization

TensorFlow models use *float32* data type for input features and internal weights. Using *int8* reduces the model size of 25% and leads to reduced latency and energy requirements during the inference. The main challenge in converting from *float32* to *int8* is that floating point representations have a much larger dynamic range when compared with fixed point. Directly mapping the *float32* values to *int8* would result in a enormous loss of precision. To overcome this issue, a calibration is necessary, and the *int8* samples does not represent all possible *float32* values but only a subset that includes the majority of the samples. Quantization is carried out using the library TensorFlow Lite [31], which apply the calibration expressed by the following equation:

$$Q(r) = \text{round}(r/S + Z) \tag{6}$$

where r is the input in floating-point precision and S is the scale factor, which depends on the range of input samples, while Z is the zero point, which serves as a bias to map the values to the center of the integer dynamic.

To reduce the model complexity, we adopted the post-training quantization (PTQ) available in TensorFlow Lite. In PTQ, the quantization process is performed after the training of the model is completed. This means that the model, which is originally trained and represented in *float32* precision, is converted to a fixed-point format. The weights are converted in *int8* data type using the calibration formula. Usually, the S and Z parameters are shared by the same layers to reduce the number of additional parameters to store. PTQ generally results in a loss of some percentile points on the model metrics. Nevertheless, the benefits regarding memory and energy consumption for the MCU are substantially greater. Fig. 1(a) displays all the steps performed by the Python framework.

V. Deployment on the MCU

This step involves converting and loading the Python model into the MCU to perform inference using the audio recorded by the microphone on the board. The microphone records with an effective sample rate of 21 978 Hz and collects samples in *int16* data type. The PTQ model is converted using the executable *stm32ai* provided by the *X-CUBE-AI* package from *STM32*. The executable generates a report detailing the flash memory required to store the weights and the code, the RAM memory needed for executing the code, and the total number of operations required for the inference. In addition, the command *validate* can be utilized to assess the performance of the converted model. By providing the input features, the corresponding outputs of the expected inference are generated. These results are analyzed to compare the performance of the initial model and determine the loss of metrics. Fig. 1(b) illustrates how these steps are integrated into the entire flow.

MFCC and STFT features are extracted in the MCU using the CMSIS-DSP library from audio recorded by the onboard microphone and stored on a dedicated flash memory. Subsequently, the feature selection process is replicated by discarding the unused coefficients. The next step involves the scaling operation, which includes the transformation with StandardScaler and the conversion to *int8* data type. To avoid floating-point computation, constants required for performing the two operations are precomputed. The complete procedure is expressed by the following equation:

$$X_s[i] = \underbrace{\frac{x[i] - \text{mean}[i]}{\text{std}[i]}}_{\text{StandardScaler}} \cdot \underbrace{\frac{1}{s}}_{int8} \gg \text{shift} + \underbrace{zp}_{int8}. \tag{7}$$

The value *x[i]* represents the coefficient in *int16* or *int32* to scale. For the first operation, the mean and standard deviation (std) of the corresponding coefficient are computed, while s and zp, which are the same for all the coefficients, are extracted by the parameters of the quantized model. The shift operation serves to move the coefficient to *int8* format. By computing, in advance, the value $(\text{std}[i] \cdot \text{d})^{-1}$, which is typically higher than 1, and rounding to integer, it is possible to perform the scaling operation entirely using integer operations.

VI. Energy Consumption

The length of the recorded audio directly impacts the energy consumption of the board. This occurs because the microphone must be powered up for a longer duration, and the amount of data produced is higher. Estimations on the consumption are made by measuring the current absorbed by the board, while it is power by 3.6 V, which is the nominal voltage of the battery. During the measurement, the feature extracted is MFCC (FFT = 1024 and MELS = 20 in *int16*). Each process of *recording–extraction* (R&E) is spaced by one and a half hours. After the measurement, the queen presence state is inferred and transmitted by the LoRaWAN application through

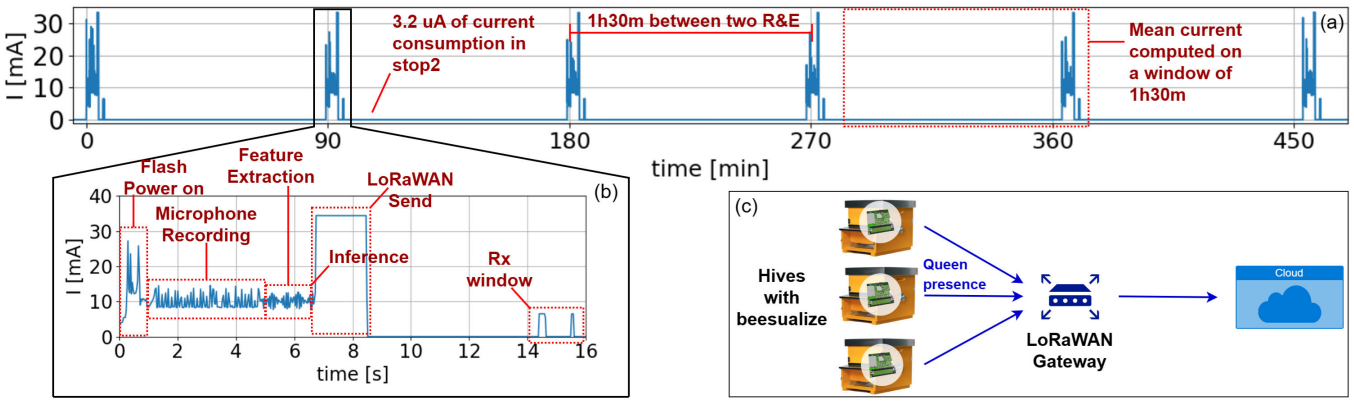


Fig. 4. (a) Current consumption of the board. (b) With zoom on R&E phase. (c) Illustration of LoRaWAN application.

TABLE IV
RESULTS OF CURRENT AND TIME MEASUREMENTS

Audio Length [s]	Mean Current [μA]		R&E Time [ms]		Energy *1h30m* [mJ]
	mean	*std*	*mean*	*std*	
1	18.37	0.2	1788.9	4.60	357.15
2	21.23	0.29	3298.0	7.82	412.77
3	24.27	0.37	4811.7	6.67	471.72
4	26.86	0.3	6280.0	7.92	522.23
5	29.76	0.42	7794.4	4.84	578.45
6	32.52	0.4	9308.1	8.89	631.75
7	35.43	0.26	10839.3	30.6	688.70

the cloud service. Between each R&E, the MCU enters in *stop 2* mode, where most clocks and peripherals are powered off. A current consumption measurement is conducted in *stop 2*, yielding a mean result of 3.202 μA with a standard deviation of 2.084 μA, using the Tektronic dm7510 multimeter as the measuring device. An overview of the activity of the IoT system is detailed in Fig. 4, and it includes three steps as follows:

1) flash memory power-on;
2) R&E and inference of queen bee presence;
3) sending and receiving window for LoRaWAN tasks.

The current is measured for about 50 s, centered on the R&E phase. To estimate the energy consumption of a complete cycle, the mean current is computed over one and a half hours, considering 3.202 μA for the unmeasured portion where the MCU is in *stop 2*. Another parameter measured is the time required only for the R&E phase. In this case, a timer of the MCU is exploited to measure the time. For both current and time, ten measurements are taken, and the mean and standard deviation are computed and reported in Table IV. The last column illustrates the energy needed to execute a cycle of 1 h and half, computed by multiplying the current by the voltage and the total time. The energy has a strong linear proportionality to the audio length. By interpolating the points with a polynomial fit algorithm (using Numpy's implementation), the increase is about 55 mJ/s. Measurements on longer audio samples are not taken, because they do not enhance the accuracy of the ML model. Considering that the battery utilized for this project has a capacity of 2600 mAh,

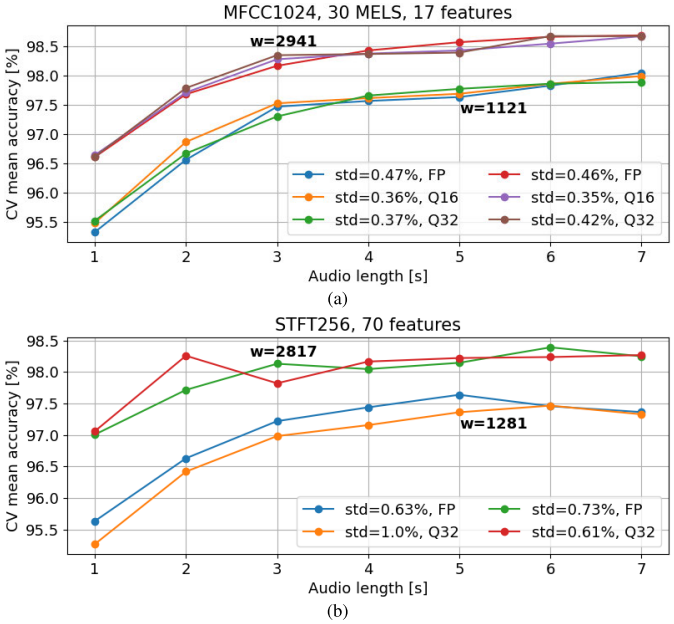


Fig. 5. CV accuracy of Python NN trained with (a) MFCC and (b) STFT for different audio lengths. Two DS are assessed. *Std* indicates the maximum CV standard deviation accuracy.

the duration should range from 24 years for 1-s audio length to 12 years for 7-s audio length. The type of battery selected is *Li-Socl2*, suitable for long-term applications up to 20 years. The computation does not include the self-discharge, which is claimed to be under 1%.

VII. NETWORK RESULTS

The metrics analyzed during these tests are *accuracy*, *precision*, *recall*, and *F*1, but only *accuracy* is displayed in most tests for brevity. The best configuration for MFCC and STFT extraction is already investigated in Section III and will be maintained for the following tests.

A second important consideration regards the duration of the recordings. Fig. 5 illustrates how the mean CV accuracy changes for different audio lengths. For the two features, two networks with different DSs are tested, and the figures report the number of weights (w). For MFCC features, Fig. 5(a),

TABLE V
ACCURACY OF NETWORKS TRAINED WITH MFCC AND STFT

MFCC

DS	Float training			Int16 training		Int32 training		
	Py	CuAI		Py	CuAI	Py	CuAI	
	float	*int16*	*int32*	*int16*	*int16*	*int32*	*int16*	*int32*
32-16	97.27%	97.04%	97.03%	97.41%	97.02%	97.20%	97.05%	97.16%
32-24	96.95%	96.84%	96.92%	96.85%	96.81%	97.23%	97.08%	97.18%
32-30	97.23%	97.01%	97.02%	97.01%	96.81%	97.17%	96.69%	97.07%
60-16	97.33%	97.33%	97.33%	97.18%	97.17%	97.41%	97.60%	97.34%
60-24	97.23%	97.19%	97.10%	97.58%	97.63%	97.32%	96.82%	97.25%
60-30	97.24%	97.26%	97.29%	97.66%	97.50%	97.40%	97.51%	97.41%

STFT

DS	Float training		Int32 training	
	Py	CuAI	Py	CuAI
	float	*int32*	*int32*	*int32*
16-8	97.39%	91.87%	97.91%	93.54%
16-12	97.36%	95.47%	97.96%	95.24%
16-16	97.32%	94.29%	97.94%	95.77%
32-8	97.80%	95.06%	98.44%	93.90%
32-12	97.88%	96.26%	98.43%	92.79%
32-16	98.02%	92.58%	98.62%	95.48%

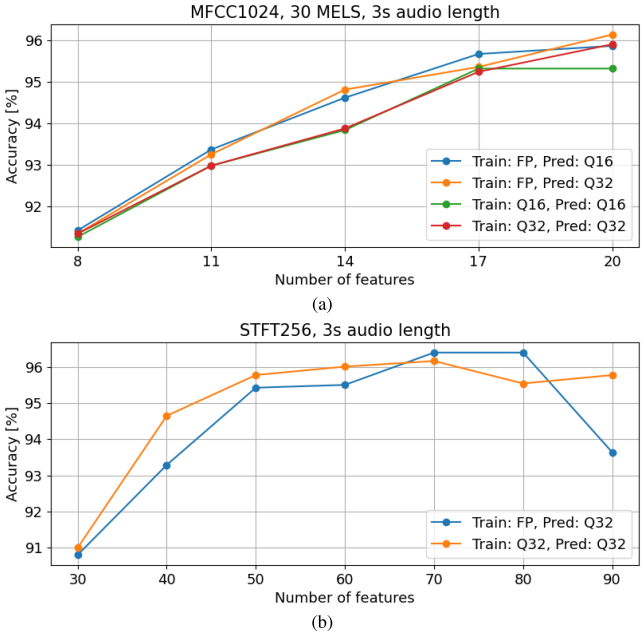


Fig. 6. Accuracy of CubeAI NN trained with (a) MFCC and (b) STFT maintaining a different number of feature. *Train* indicates the data type utilized for training in Python, while *Pred* shows the data type employed for the inference in the converted CubeAI model.

the accuracy of the models trained with the three data types is illustrated. While for STFT feature, Fig. 5(b), the *int16* format does not achieve satisfying results, and it is discarded, indeed, as explained in Section III, and the number of coefficients with a small error is extremely low. These tests are performed only in Python, and Fig. 5 shows the results named *CV metrics* in Fig. 1(a). Increasing the audio length collected more information, and the prediction is usually more precise. The accuracy starts to saturate for a duration of 3–4 s. We identified the 3-s audio duration as a good tradeoff between energy saving and loss in accuracy.

The third consideration is about the feature selection process. In tests conducted in Python with floating point, increasing the number of features improves the metrics. However, when features are extracted in fixed point, this behavior is not ensured. Due to fixed-point computation, some coefficients can saturate to 0 with very limited variance. This behavior can result in a saturation or decline of the accuracy. Fig. 6 shows the accuracy of some tests with a network configured in the same way as the previous test (utilizing the larger DS). In this case, the accuracy from *CubeAI metrics* is displayed.

TABLE VI
CUBEAI REPORT FOR MFCC AND STFT

MFCC

W	FLASH [KB]	RAM [B]	MACC
1121	14.637	1980	1122
1393	14.930	1988	1394
1597	15.148	1996	1598
2073	15.648	2064	2074
2569	16.156	2072	2570
2941	16.535	2080	2942

STFT

W	FLASH [KB]	RAM [B]	MACC
1281	14.723	2096	1282
1353	14.805	2096	1354
1425	14.891	2096	1426
2545	16.008	2112	2546
2681	16.148	2112	2682
2817	16.293	2112	2818

For MFCC, when the inference exploits *int16* data type, the accuracy saturates for 17 coefficients, while continues to increase for the *int32* data type. The behavior is similar for both training in floating point and fixed point. For STFT, performing the training using floating-point coefficients results in a drop of accuracy for a high number of coefficients. After this step, 17 features are selected for MFCC and 70 for STFT.

The final tests are performed, maintaining the audio duration (3 s) and number of features (17 for MFCC and 70 for STFT) determined. Six NNs with two layers with different DSs are assessed, with DS selected to generate networks with a number of weights ranging from 1000 to 3000. For each NN, two different approaches are evaluated as follows.

1) The training is performed with floating-point features; then, the CubeAI model is tested with fixed-point features.
2) Both training and testing of CubeAI models are performed in fixed point with the same features.

Table V reports the model accuracies obtained. The **"Py"** column indicates the accuracy of the model before the quantization (*Python metrics* as named in Fig. 1), while the **"CuAI"** column indicates the accuracy of the model executed into the MCU (*CubeaAI metrics*).

It is evident that there is an accuracy loss in most of cases due to the quantization process and the scaling operation. When the model is trained with *int32*, only for MFCC, the behavior of the inference utilizing *int16* is evaluated. The aim is to assess if the inference in *int16* achieves a better accuracy when the training is performed with features extracted with higher precision. For MFCC, the most constant behavior is achieved by the columns "**CuAI** and *int32*." The loss in accuracy, compared with the correspondent **"Py"** column, is lower than 0.10%. The behavior of the other columns is more varied, with a maximum accuracy loss of 0.50%, while in some cases, the fixed-point model performs better

compared with the corresponding Python model. In general, for a higher number of parameters, the accuracy loss is more limited; while comparing the difference between *int16* and *int32*, there is no significant improvement in the accuracy. Utilizing STFT, the accuracy of CubeAI model decreases in a more consistent way. The accuracy drop ranges from 1.6% to 6%, and there is no evident advantage in training the network with fixed-point features. Regarding the accuracy loss due to quantization and fixed-point extraction, a comparison can be made with [32], where an integer algorithm for computing MFCCs is developed, and the degradation in accuracy of a deep NN trained with floating-point features is assessed. Similar to this study, for *int32*, the accuracy drop is minimal, while for *int16*, it is smaller than 0.6%.

Table VI illustrates statistics extracted by *stm32ai* from tested models. Specifically, they are related to the models in the "*Int32* training, **CuAI**, *int32*" columns of Table V. The other columns exhibit the same values with some small variations due to difference in code size. The fields include the following.

1) *W:* The number of weights.
2) *FLASH:* The memory needed to store weights and code.
3) *RAM:* The memory needed to run the inference.
4) *MACC:* The number of multiplication and accumulation to be made during the inference.

As expected, the number of **MACC** operations is proportional to the number of weights of the network. Moreover, the achieved low memory footprint enables the deployment on compact and low end MCU.

VIII. Conclusion

In conclusion, this study demonstrates the effective application of TinyML for edge computing in IoT systems to predict the presence of the queen bee in a hive, which is crucial for assessing the health state of the queen bee. Using coefficients extracted from audio recordings obtained from open source datasets, the developed ML models are exceptionally compact, occupying less than 17 kB and require only 2-kB RAM for inference. The models achieve accuracy rates above 97%, using MFCC features computed in *int16* and above 93% with STFT features computed in *int32*. These results are comparable to those in the literature that employ more complex models and floating-point computations, which are impractical for integration in the selected MCU due to stringent energy and memory constraints. The feature extraction algorithm is optimized to minimize latency and energy consumption, significantly extending battery life, with approximately 400 mJ required for measuring and inferring the queen bee's presence. These findings underscore the importance of balancing accuracy, power budget, and available computational resources. The study highlights that with careful optimization, TinyML can provide highly efficient and accurate solutions for IoT applications, even within the tight constraints of edge devices.

References

[1] A.-M. Klein et al., "Importance of pollinators in changing landscapes for world crops," *Proc. Roy. Soc. B, Biol. Sci.*, vol. 274, no. 1608, pp. 303–313, Feb. 2007.

[2] D. vanEngelsdorp, J. Hayes, R. M. Underwood, and J. S. Pettis, "A survey of honey bee colony losses in the United States, fall 2008 to spring 2009," *J. Apicultural Res.*, vol. 49, no. 1, pp. 7–14, Jan. 2010.

[3] S. Cecchi, S. Spinsante, A. Terenzi, and S. Orcioni, "A smart sensor-based measurement system for advanced bee hive monitoring," *Sensors*, vol. 20, no. 9, p. 2726, May 2020.

[4] B. Maciejovsky, B. Baer-Imhoof, and B. Baer, "Hobbyist beekeepers and their importance for future beekeeping and food production," *Bee World*, vol. 100, nos. 3–4, pp. 80–83, Oct. 2023.

[5] N. E. Duffus, A. Echeverri, L. Dempewolf, J. A. Noriega, P. R. Furumo, and J. Morimoto, "The present and future of insect biodiversity conservation in the neotropics: Policy gaps and recommendations," *Neotropical Entomol.*, vol. 52, no. 3, pp. 407–421, Mar. 2023.

[6] A. Zacepins, A. Kviesis, P. Ahrendt, U. Richter, S. Tekin, and M. Durgun, "Beekeeping in the future—Smart apiary management," in *Proc. 17th Int. Carpathian Control Conf. (ICCC)*, May 2016, pp. 808–812.

[7] S. Ferrari, M. Silva, M. Guarino, and D. Berckmans, "Monitoring of swarming sounds in bee hives for early detection of the swarming period," *Comput. Electron. Agricult.*, vol. 64, no. 1, pp. 72–77, Nov. 2008.

[8] A. Qandour, I. Ahmad, D. Habibi, and M. Leppard, "Remote beehive monitoring using acoustic signals," *Acoust. Australia/Australian Acoust. Soc.*, vol. 42, pp. 204–209, Dec. 2014.

[9] C. Uthoff, M. N. Homsi, and M. von Bergen, "Acoustic and vibration monitoring of honeybee colonies for beekeeping-relevant aspects of presence of queen bee and swarming," *Comput. Electron. Agricult.*, vol. 205, Feb. 2023, Art. no. 107589.

[10] D. S. Kridi, C. G. N. D. Carvalho, and D. G. Gomes, "A predictive algorithm for mitigate swarming bees through proactive monitoring via wireless sensor networks," in *Proc. 11th ACM Symp. Perform. Eval. Wireless Ad Hoc, Sensor, Ubiquitous Netw.*, Sep. 2014, pp. 41–47.

[11] I. Nolasco and E. Benetos, "To bee or not to bee: Investigating machine learning approaches for beehive sound recognition," 2018, *arXiv:1811.06016*.

[12] J. Kim, J. Oh, and T.-Y. Heo, "Acoustic scene classification and visualization of beehive sounds using machine learning algorithms and grad-CAM," *Math. Problems Eng.*, vol. 2021, pp. 1–13, May 2021.

[13] T.-T.-H. Phan, H.-D. Nguyen, and D.-D. Nguyen, "Evaluation of feature extraction methods for bee audio classification," in *Proc. Int. Conf. Intell. Things*. Cham, Switzerland: Springer, 2022, pp. 194–203.

[14] V. A. Kulyukin, S. Mukherjee, and Y. B. Burkatovskaya, "Classification of audio samples by convolutional networks in audio beehive monitoring," *Vestnik Tomskogo Gosudarstvennogo Universiteta. Upravlenie, Vychislitel'naya Tekhnika I Informatika*, vol. 45, no. 45, pp. 68–75, Dec. 2018. [Online]. Available: https://vital.lib.tsu.ru/vital/access/manager/Repository/vtls:000645833

[15] I. Nolasco, A. Terenzi, S. Cecchi, S. Orcioni, H. L. Bear, and E. Benetos, "Audio-based identification of beehive states," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2019, pp. 8256–8260.

[16] A. Terenzi, N. Ortolani, I. Nolasco, E. Benetos, and S. Cecchi, "Comparison of feature extraction methods for sound-based classification of honey bee activity," *IEEE/ACM Trans. Audio, Speech, Language Process.*, vol. 30, pp. 112–122, 2022.

[17] A. Orlowska, D. Fourer, J.-P. Gavini, and D. Cassou-Ribehart, "Honey bee queen presence detection from audio field recordings using summarized spectrogram and convolutional neural networks," in *Proc. Int. Conf. Intell. Syst. Design Appl.* Cham, Switzerland: Springer, 2021, pp. 83–92.

[18] B. S. Soares, J. S. Luz, V. F. de Macêdo, R. R. V. E. Silva, F. H. D. de Araújo, and D. M. V. Magalhães, "MFCC-based descriptor for bee queen presence detection," *Expert Syst. Appl.*, vol. 201, Sep. 2022, Art. no. 117104.

[19] A. Robles-Guerrero, T. Saucedo-Anaya, E. González-Ramérez, and C. E. Galván-Tejada, "Frequency analysis of honey bee buzz for automatic recognition of health status: A preliminary study," *Res. Comput. Sci.*, vol. 142, no. 1, pp. 89–98, Dec. 2017.

[20] S. Ruvinga, G. J. A. Hunter, O. Duran, and J.-C. Nebel, "Use of LSTM networks to identify 'queenlessness' in honeybee hives from audio signals," in *Proc. 17th Int. Conf. Intell. Environ. (IE)*, Jun. 2021, pp. 1–4.

[21] A. Robles-Guerrero, T. Saucedo-Anaya, E. González-Ramírez, and J. I. De la Rosa-Vargas, "Analysis of a multiclass classification problem by lasso logistic regression and singular value decomposition to identify sound patterns in queenless bee colonies," *Comput. Electron. Agricult.*, vol. 159, pp. 69–74, Apr. 2019.

[22] T. Cejrowski, J. Szymański, H. Mora, and D. Gil, "Detection of the bee queen presence using sound analysis," in *Intelligent Information and Database Systems*, N. T. Nguyen, D. H. Hoang, T.-P. Hong, H. Pham, and B. Trawiński, Eds., Cham, Switzerland: Springer, 2018, pp. 297–306.
[23] H.-T. Ho, M.-T. Pham, Q.-D. Tran, Q.-H. Pham, and T.-T.-H. Phan, "Evaluating audio feature extraction methods for identifying bee queen presence," in *Proc. 12th Int. Symp. Inf. Commun. Technol.*, Dec. 2023, pp. 93–100.
[24] L. Barbisan and F. Riente, "Machine learning framework for the acoustic detection of the queen bee presence," in *Proc. 10th Conv. Eur. Acoust. Assoc. Forum Acusticum*, Jan. 2024, pp. 4347–4350.
[25] L. Barbisan, G. Turvani, and R. Fabrizio, "Audio-based identification of queen bee presence inside beehives," in *Proc. IEEE Conf. AgriFood Electron. (CAFE)*, Sep. 2023, pp. 70–74.
[26] A. Yang. (2022). *Smart Bee Colony Monitor: Clips of Beehive Sounds*. [Online]. Available: https://www.kaggle.com/dsv/4451415
[27] I. Nolasco, A. Terenzi, S. Cecchi, S. Orcioni, H. L. Bear, and E. Benetos, "Audio-based identification of beehive states: The dataset," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2019, pp. 8256–8260, doi: 10.5281/zenodo.2667806.
[28] B. McFee et al., "Librosa: Audio and music signal analysis in Python," in *Proc. Python Sci. Conf.*, 2015, pp. 18–24.
[29] ARM. (2023). *CMSIS DSP Software Library*. Accessed: Oct. 10, 2023. [Online]. Available: https://www.keil.com/pack/doc/CMSIS/DSP/html/index.html
[30] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," Tensorflow, Mountain View, CA, USA, 2015.
[31] TensorFlow. (2023). *TensorFlow Lite for Microcontrollers*. Accessed: Jun. 25, 2023. [Online]. Available: https://www.tensorflow.org/lite/microcontrollers
[32] M. Fariselli, M. Rusci, J. Cambonie, and E. Flamand, "Integer-only approximated MFCC for ultra-low power audio NN processing on multi-core MCUs," in *Proc. IEEE 3rd Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, Jun. 2021, pp. 1–4.



Andrea De Simone received the bachelor's degree in electronics and telecommunication engineering and the master's degree in electronics engineering from the Politecnico di Torino, Turin, Italy, in 2021 and 2023, respectively.

Currently, he is a Research Fellow with the Department of Electronics and Telecommunications, Politecnico di Torino. His research activity includes the development of the IoT systems for agritech application and tiny machine learning.



Luca Barbisan (Student Member, IEEE) received the bachelor's degree in electronics and telecommunication engineering and the master's degree in electronics engineering from the University of Trento, Turin, Italy, in 2017 and 2021, respectively, where he is currently pursuing the Ph.D. degree in electronics and telecommunications engineering.

His current research interests include digital signal processing and modern machine learning techniques, mainly applied to audio signals.



Giovanna Turvani received the master's degree (magna cum laude) in electronic and telecommunication engineering and the Ph.D. degree from the Politecnico di Torino, Turin, Italy, in 2012 and 2016, respectively.

In 2015, she went to the Technical University of Munich (TUM), Munich, Germany, for research on nanocomputing. After that, she worked as a Post-Doctoral Researcher at the Politecnico di Torino, focusing on microwave imaging systems for biomedical use and food safety, where she is currently an Associate Professor, researching low-power embedded systems for environmental monitoring, quantum computing, and digital architectures based on the logic-in-memory paradigm.



Fabrizio Riente (Member, IEEE) received the M.Sc. degree (magna cum laude) in electronic engineering and the Ph.D. degree from the Politecnico di Torino, Turin, Italy, in 2012 and 2016, respectively.

He was a Post-Doctoral Research Associate with the Technical University of Munich, Munich, Germany, in 2016. He is currently an Assistant Professor with the Department of Electronics and Telecommunications, Politecnico di Torino. His primary research interests include device modeling and circuit design for nanocomputing, with a particular interest on field-coupled nanotechnologies. His interests also cover the development of audio processing systems for event detection and low-power IoT devices.

Open Access funding provided by 'Politecnico di Torino' within the CRUI CARE Agreement