TinySV: Speaker Verification in TinyML with On-device Learning

Massimo Pavan[§]
Politecnico di Milano
Milan, Italy
massimo.pavan@polimi.it

Gioele Mombelli[§]
Politecnico di Milano
Milan, Italy
gioele.mombelli@mail.polimi.it

Francesco Sinacori
Infineon Technologies Italia s.r.l.
Milan, Italy
francesco.sinacori@infineon.com

Manuel Roveri
Politecnico di Milano
Milan, Italy
manuel.roveri@polimi.it

***Abstract*—TinyML is a novel area of machine learning that gained huge momentum in the last few years thanks to the ability to execute machine learning algorithms on tiny devices (such as Internet-of-Things or embedded systems). Interestingly, research in this area focused on the efficient execution of the inference phase of TinyML models on tiny devices, while very few solutions for on-device learning of TinyML models are available in the literature due to the relevant overhead introduced by the learning algorithms.**

The aim of this paper is to introduce a new type of adaptive TinyML solution that can be used in tasks, such as the presented *Tiny Speaker Verification* (TinySV), that require to be tackled with an on-device learning algorithm. Achieving this goal required (i) reducing the memory and computational demand of TinyML learning algorithms, and (ii) designing a TinyML learning algorithm operating with few and possibly unlabelled training data. The proposed TinySV solution relies on a two-layer hierarchical TinyML solution comprising Keyword Spotting and Adaptive Speaker Verification module. We evaluated the effectiveness and efficiency of the proposed TinySV solution on a dataset collected expressly for the task and tested the proposed solution on a real-world IoT device (Infineon PSoC 62S2 Wi-Fi BT Pioneer Kit).

***Index Terms*—component, formatting, style, styling, insert**

I. Introduction

Tiny Machine Learning (TinyML) recently became one of the most promising areas in the field of Machine Learning. By enabling machine and deep learning models and algorithms to operate on battery-operated devices [1], [2] (e.g., embedded and Internet-of-Things units), TinyML created a whole new class of tasks and applications ranging from Keyword Spotting (KS) [3], i.e., recognizing a pre-determined word or command in an audio stream, to object or anomaly detection [4], [5] in images or accelerometers data.

A growing literature exists in the field of TinyML [6], [7]. Solutions in this field aim at either designing efficient architectures for machine and deep learning models (e.g., neural networks models employing efficient and lightweight layers) [8], [9] or approximate computing strategies to optimize the memory and computational demand (e.g., quantization or pruning mechanisms) [10], [11].

Interestingly, current solutions assume that the training phase of TinyML models is carried out in the Cloud where appropriate computing and memory resources are available, while just the inference phase is performed on the target tiny devices.

Unfortunately, this approach does not allow TinyML solutions to exploit data collected directly from the field by the device, hence preventing the incremental training or adaptation of the TinyML algorithms during the operational life. Many applications that require on-device adaptation capabilities are consequently still not viable in TinyML. An example in this field is "Speaker verification" (SV) [12], a task that consists of recognizing the identity of a user by analyzing audio captions provided by the user as a reference and comparing them to newly collected audio data. In this context, the implementation of a SV system on a tiny device would enforce relevant applications, including smart locks that can recognize their owners or smart objects offering different behaviors according to the specific person it is interacting with.

In this work we propose, for the first time in the literature, the definition of *Tiny Speaker Verification* (TinySV), a task specifically tailored to the *on-device learning* context, and introduce a TinyML algorithm supporting the on-device learning of SV applications. The proposed solution has been specifically designed to:

- Learn a TinyML model directly on-device in a **one-class** manner (with data belonging to only one class of label);
- Operate in a **few-shot** setting (hence enforcing the learning on a small amount of data);
- Run continuously in an "always-on" manner on a tiny, battery-operated device.

In more detail, the proposed solution operates in a *text-dependant* way (i.e., a pre-determined keyword is used to recognize the identity of the speaker [13]), and relies on a two-layer hierarchical solution comprising Keyword Spotting (KS) and Adaptive Speaker Verification (ASV) operating in a cascade manner. The solution has been tested on a text-dependent SV dataset that has been expressly collected for this task, which is released to the community along with the code for the experiments and the implementation in the project repository[§].

The paper is organized as follows. Sec. II introduces the related literature. Sec. III formalizes the task of *Tiny Speaker Verification* proposed in this work. The proposed solution

Identify applicable funding agency here. If none, delete this.

[§]These authors contributed equally to this work

[§]https://github.com/AI-Tech-Research-Lab/TinySV

is described in Sec. IV. Sec. V describes the experimental settings and results. Details on the on-device implementation of TinySV are given in Sec. VI, while conclusions are finally drawn in Sec. VII.

II. Related Literature

In this Section, we discuss the related literature in the following fields: TinyML (Section II-A), Incremental on-device Learning in TinyML (Section II-B), and Speaker Verification (Section II-C).

A. TinyML

TinyML is a field of study that combines embedded systems and machine learning (ML). It studies ML models and architectures designed to be executed on small and low-power devices, hence taking into account their severe technological constraints in terms of memory (less than 1 MB of RAM available on-device), computation (clock frequency is in the order of hundreds of KHz), and power consumption (less than tens of mW) [1]. Most of the solutions present in this field focus on the design of *approximated machine and deep learning solutions* [6], [14]. In particular, techniques such as weight quantization [10], pruning [11] and gate-classification [15] have been developed to reduce the memory and computational demand of machine and deep learning models, while guaranteeing their accuracy [16], [17].

TinyML paved the way for a wide range of intelligent embedded applications like visual wake-word detection [4], anomaly detection with accelerometers [5], and presence detection with radar [18]. Among the wide range of applications keyword spotting (KS) [3] received a lot of attention from both the academic and the industrial perspective thanks to the ability to detect the presence of a pre-determined word or command in a continuous audio stream.

B. Incremental on-device Learning in TinyML

Incremental on-device TinyML is a novel and promising area of TinyML aiming to directly support the incremental learning of TinyML models on the tiny devices, hence overcoming the traditional "train-on-cloud and deploy-on-device" paradigm in TinyML.

Solutions present in the literature can be organized into two main categories [19]: instance-based (called lazy learning) and model-based (called eager learning).

1) Instance-based: The instance-based solutions present in the literature [20], [21] and [22] rely on a Convolutional Neural Network (CNN) to perform feature extraction and dimensionality reduction on the input data. In these models, the learning phase consists of storing the labeled representations, while the inference phase involves the computation of a distance metric between the unlabeled representation of the input data and the previously extracted representations. The main advantages of this approach lie in the fact that (i) the training, which is usually the most computationally demanding task in ML, consists just of storing a dimensionality-reduced version of the data and (ii) these solutions provide acceptable results even with a small amount of data available [22].

2) Model-based: Model-based learning mostly relies on the use of an optimized version of backpropagation for the adaptation of neural networks directly on-device. All the solutions present in the literature freeze some parts of the neural network to reduce the number of weights that need to be trained [23], [24]. The same approach is used in [25] on a task of anomaly detection. All these solutions rely on training in an online manner (i.e., train on one datum at a time and discard it), and for this reason, they are limited in their ability to learn complex patterns and exploit batches of data to avoid overfitting. A solution to enable learning over batches of data is explored in [26], which proposed to store only the latent representations (i.e., lighter representation of data in terms of memory occupation with respect to the complete datum) in order to perform multiple training epochs. Despite that, the amount of latent representations storable on tiny devices is usually orders of magnitude smaller than the one usually used in standard ML pipelines. For this reason [27] proposed a hybrid approach that continuously adapts the last layers of the network on batches of data stored as latent replays. The only model-based solution present in the literature that does not rely on neural networks is [28], an extremely efficient binary classifier that works on low-dimensional data. We emphasize that all the model-based solutions present in the literature assume a large availability of labeled data to perform training, a requirement seldom satisfiable in the TinyML environment [29].

Currently, none of the works present in the On-device TinyML literature encompass on-device learning mechanisms able to work in a few-shot and one-class manner at the same time.

C. Speaker Verification

The Speaker Verification (SV) task can be formalized as a binary classification problem where the goal, given an audio segment containing the voice of a user, is to distinguish whether this voice belongs or not to a previously enrolled speaker. The enrolled speaker is expected to provide a series of audio recordings containing his/her voice so as to configure the SV system.

The SV task can be tackled with either a text-dependent approach (the user is expected to pronounce a pre-determined word to be recognized) or a text-independent one (the algorithm is expected to recognize the enrolled user independently from what they are saying) [12].

Available solutions for SV include Gaussian Mixture-Model-Universal Background Models (GMM-UBM), Gaussian Mixture-Model Support Vector Machines (GMM-SVM), Joint Factor Analysis (JFA) and i-vectors [30] [31]. With the advent of deep learning and its strong representation and classification abilities, the research in SV took two different directions: deep learning models operating in traditional frameworks, e.g., the DNN/i-vector approach [31], and sole deep learning models extracting a representation of speakers' voice characteristics in a low-dimensional space called "embedding", on which classification and comparison algorithms

can run [31]. Some works targeting low memory footprint applications are present in the literature [32]–[34]. Among these articles, the "d-vector-based method" introduced in [32] is one of the most suitable ones for edge applications. This method relies on a neural network able to extract a voice-dependent low-dimensional vector, called "d-vector", from input speech that can be used by an instance-based solution for recognizing the identities of the speakers.

Interestingly, some reference datasets are present in the literature both for text-independent [35] and text-dependent SV [36], [37], but all of them encompass long audio recordings ($>$ 3s), a fact that makes their usage harder while developing solutions for extremely constrained environments.

All in all, none of the solutions for SV present in the literature is tailored for tiny devices nor presents a deployment on embedded devices encompassing both the enrollment and inference phases.

III. Tiny Speaker Verification: the use case

The goal of this section is to introduce TinySV, a new application for on-device learning and speaker verification in TinyML. We emphasize that the task is a particular type of text-dependent SV (i.e., recognizing the identity of the enrolled speaker from utterances of a specific word), in which both the keyword (i.e., the specific word or passphrase) and the identity of the speaker must be recognized at the same time from a continuous audio stream directly on a tiny device.

In addition, this task must be tackled while keeping into consideration the relevant and challenging characteristics of the TinyML context:

- the SV algorithm must be adapted directly on-device, meaning that a new user should be able to enroll in the SV application by providing examples of their voice directly through the target device;
- the algorithm must operate in a one-class manner, meaning that it should be able to learn to distinguish between the enrolled user and any other users only from data coming from the enrolled one;
- the algorithm must follow a few-shot learning approach, meaning that it should be able to operate even with few training data of the enrolled speaker;
- the algorithm must match the strict technical requirements of tiny devices, meaning that it must operate requiring a small amount of memory and computation during both the inference and learning phase.

More formally, the tiny device is continuously recording an audio stream by using a microphone characterized by the sampling frequency f_r. At time t, the most recent window I_t, whose length is W seconds, is extracted from the stream and used as input for the algorithm.

Given a pre-defined keyword k, the task of the TinySV algorithm is to assign a label $x_t \in \{0, 1, 2\}$ to the most recent segment I_t of the stream where:

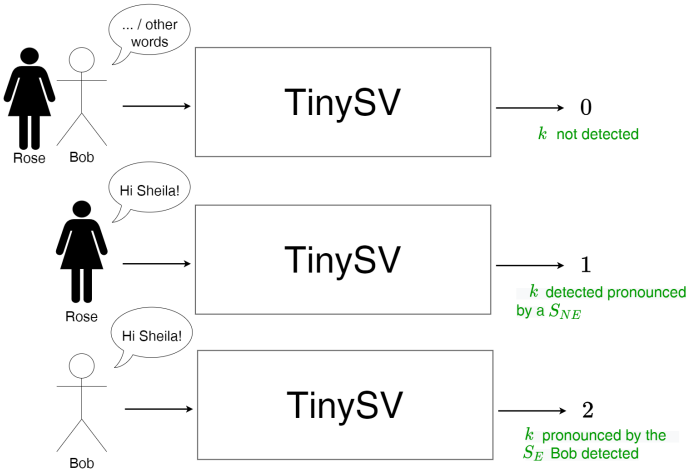

Fig. 1. Examples of the use case, in which k = "Sheila" and the enrolled speaker S_E is Bob.

$$x_t : \begin{cases} 0 : & k \text{ not present in } I_t \\ 1 : & k \text{ present in } I_t \text{ and pronounced by } S_{NE} \\ 2 : & k \text{ present in } I_t \text{ and pronounced by } S_E \end{cases} \quad (1)$$

where S_E is the enrolled speaker (i.e., the speaker whose voice must be recognized by the algorithm), and S_{NE} is any other, not-enrolled, speaker. The general use case of TinySV is depicted in Fig. 1.

IV. Enabling TinySV: the proposed solution

The proposed solution for TinySV on audio streams relies on a two-layer hierarchical solution comprising:

- the Keyword Spotting (KS) module;
- the Adaptive Speaker Verification (ASV) module.

The KS model is used to determine if the audio segment I_t under inspection includes the pre-determined keyword k. If k is detected in I_t, the audio segment is forwarded to the ASV module, which is meant to (i) create a personalized model for the enrolled speaker S_E during the model adaptation phase and (ii) distinguish if k was pronounced by S_E or by a non-enrolled speaker S_{NE} during the inference phase.

We emphasize that the combination of the aforementioned two modules is used to address the problem formalized in Sec. III, while a visual representation of the high-level pipeline of the proposed solution is depicted in Fig. 2.

As detailed in what follows, before being used as input by the two modules, I_t is pre-processed and transformed into a Mel-frequency cepstral coefficients (MFCC) spectrogram P_t through a module called *MFCC extractor*. In order to reduce the number of operations needed to execute the pipeline on-device, the preprocessing is shared among the keyword spotting and the speaker verification module.

The rest of the section is organized as follows. In Sec. IV-A the preprocessing phase performed by the MFCC extractor is described. The KS and ASV modules are described in Sec.

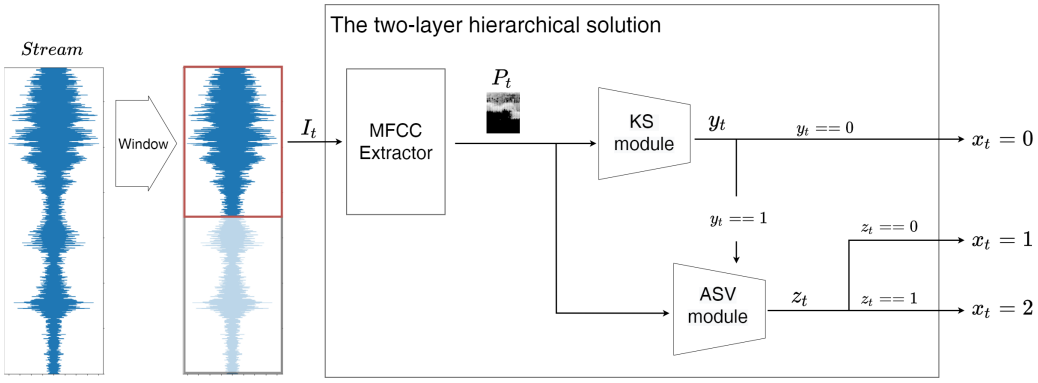

Fig. 2. An high level representation of the proposed solution.

IV-B and IV-C, respectively. Finally, a description of the two-layer hierarchical solution is drawn in Sec. IV-D, followed by the comments on the memory requirements in Sec. IV-E.

A. MFCC Extractor

The goal of this module is to transform the raw input I_t into an MFCC spectrogram $P_t \in \mathbb{R}^{i \times j}$, highlighting the relevant audio features present in the data and, at the same time, reducing the data dimensions.

The MFCC extractor relies on the pre-processing pipeline used in [38] for keyword spotting, receiving in input a W-second long audio record sampled at f_S (hence represented by a vector of dimension $W \cdot f_S$), and producing as output a $i \times j$ Mel Frequency Cepstral Coefficients (MFCC) spectrogram, being i the number of frequency bins extracted from the pre-processing pipeline and j the number of audio segments obtainable from a single window. The MFCC extractor operates by splitting the W-second long input into λ-seconds long audio segments and processing them through the use of FFT and Mel frequency downsampling. Since the λ second-long segments are overlapped with a stride value of ϕ, the value of j can be computed as $j = W/\phi - \lambda/\phi$.

In the proposed implementation and experimental section, the input I_t (characterized by $W = 1$s, $f_r = 16$KHz) is preprocessed into a spectrogram P_t of dimensions $i = 40 \times j = 49$, while λ is equal to 30ms and $\phi = 20$ms.

B. The KS module

The KS module aims at recognizing if I_t contains the predetermined keyword k. The problem can be formalized as a binary classification task, whose goal is the association of I_t to a label $y_t \in \{0, 1\}$ where:

$$y_t : \begin{cases} 0 : & k \text{ not present in } I_t \\ 1 : & k \text{ present in } I_t \end{cases}. \tag{2}$$

The KS module consists of a Convolutional Neural Network (CNN) Φ_k trained in a supervised manner to distinguish among silence, unknown words (i.e., speech that does not contain the keyword k), and the keyword k. It receives in input the spectrogram P_t and, following other architectures used for keyword spotting [39], produces as output one of the 3 classes (i.e., silence, unknown, and keyword). The assigned value is $y_t = 0$ in the case in which the network assigns the silence or unknown class to the datum, $y_t = 1$ if it recognizes a keyword.

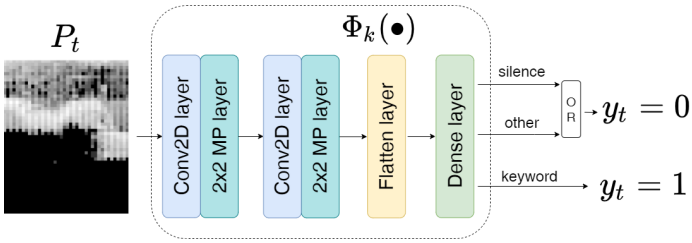

Fig. 3. The architecture of the neural network used for keyword spotting.

Φ_k is organized as the state-of-the-art architecture labeled as *cnn-trad-fpool3* proposed in [39], consisting of two 2D-convolutional/max-pooling blocks, comprising a 2D convolutional layer (characterized by a number m of $r \times q$ filters and stride $= s$) and a 2×2 2D max pooling layer, a flattening layer and a dense layer (characterized by a number a of neurons). A high-level representation of the architecture is depicted in Fig. 3.

Φ_k is also characterized by its total number of weights ω_{Φ_k} and by the number of parameters required to store its activation α_{Φ_k}, which can be estimated as:

$$\omega_{\Phi_k} = \sum_{l \in \Phi_k} \omega_l,$$
$$\alpha_{\Phi_k} = \sum_{l \in \Phi_k} \alpha_l.$$

being ω_l and α_l the number of weights and the output dimension of a layer l, respectively. ω_{Φ_k} and α_{Φ_k} obviously depend on the hyperparameters of the specific implementation of Φ_k. The hyperparameters and the α_l and ω_l of the processing layers in Φ_k used for the on-device implementation in Sec. VI are reported in Tab. I.

l	Hyperparameters	α	ω
Input	-	1960	0
Conv2D	$r = 8$, $q = 20$, $m = 16$, $s = 2$	8000	2576
MP 2D	2×2	1920	0
Conv2D	$r = 4$, $q = 10$, $m = 32$, $s = 1$	3840	20512
MP 2D	2×2	960	0
Flatten	-	960	0
Dense	$a = 3$	3	2883
Tot. Φ_k		17,643	25,971

TABLE I
HYPERPARAMETERS, α AND ω VALUES OF THE $\Phi_k(\bullet)$ USED IN THE ON-DEVICE IMPLEMENTATION.

C. The ASV module

The task of the ASV module is to recognize if the keyword k contained in the audio record I_t was pronounced by the enrolled speaker S_E or by another, non-enrolled, speaker S_{NE}. As before, the problem can be formalized as a binary classification task that consists of associating to I_t a label $z_t \in \{0, 1\}$ where:

$$z_t : \begin{cases} 0 : & k \text{ was pronounced by } S_{NE} \\ 1 : & k \text{ was pronounced by } S_E \end{cases}. \quad (3)$$

The ASV module consists of a fixed d-vector extractor model $\Phi_f(\bullet)$ and an adaptive instance-based model used for the classification, $\Phi_c(\bullet)$. Both models are now detailed.

1) The convolutional d-vector extractor $\Phi_f(\bullet)$: The generated spectrograms P_t are used as inputs for a convolutional neural network $\Phi_f(\bullet)$. Following a transfer learning approach $\Phi_f(\bullet)$ is developed by training a neural network to perform a speaker classification task in a supervised manner, and then removing its final classification layers. In more detail, $\Phi_f(\bullet)$ is composed of a batch normalization layer, a sequence of 2D convolution (characterized by a number m of $r \times q$ filters and stride = s) and Maxpooling layers, and a final flattening layer. A high level representation of the $\Phi_f(\bullet)$ architecture is provided in Fig. 4.

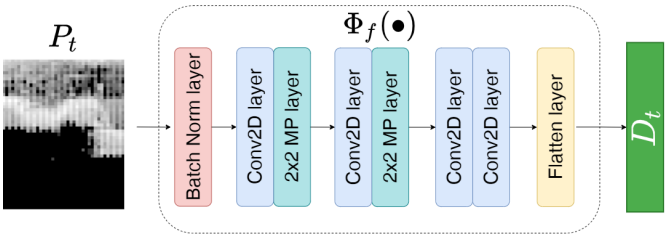

Fig. 4. The architecture of the neural network used for extracting the d-vectors.

$\Phi_f(\bullet)$ is characterized by its total number of weights ω_{Φ_f} and by the number of parameters required to store its activation α_{Φ_f}, which, similarly to ω_{Φ_k} and α_{Φ_k}, can be estimated as:

$$\omega_{\Phi_f} = \sum_{l \in \Phi_f} \omega_l,$$
$$\alpha_{\Phi_f} = \sum_{l \in \Phi_f} \alpha_l.$$

being ω_l and α_l the number of weights and activations of a layer l of Φ_f, respectively. The hyperparameters and values of the α_l and ω_l of the layers in Φ_f used for the experiments in Sect. V and in the on-device implementation in Sec. VI are reported in Tab. II.

l	Hyperparameters	α	ω
Input	-	1960	0
BatchNorm	-	1960	4
Conv2D	$r = 3$, $q = 3$, $m = 8$, $s = 1$	15680	80
MP 2D	3×3	1664	0
Conv2D	$r = 3$, $q = 3$, $m = 16$, $s = 1$	3328	1168
MP 2D	2×2	768	0
Conv2D	$r = 3$, $q = 3$, $m = 32$, $s = 1$	384	4640
Conv2D	$r = 3$, $q = 3$, $m = 64$, $s = 2$	256	18496
Flatten	-	256	0
Tot. Φ_f		26,656	24,388

TABLE II
HYPERPARAMETERS, α AND ω VALUES OF THE $\Phi_f(\bullet)$ USED IN THE ON-DEVICE IMPLEMENTATION.

The latent representation $D_t \in \mathbf{R}^d$ (where d correspond to the value α of the Flatten layer of Φ_f) that $\Phi_f(\bullet)$ produces in output is called the D-vector, and it will be used as input for the training and inference of the classification model $\Phi_c(\bullet)$. In the experiments and in the on-device implementation, $d = 256$.

2) The instance-based model $\Phi_c(\bullet)$: It is the only part of the pipeline that is adapted directly on-device. It operates in two distinct phases: *the learning phase* and *the inference phase*.

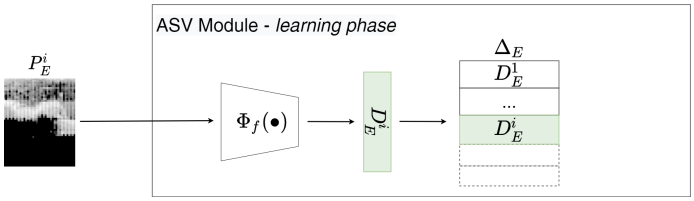

Fig. 5. The adaptation phase of the proposed adaptive speaker verification model.

2a) Learning phase: Being $\Phi_c(\bullet)$ an instance-based model, the training phase of the algorithm consists just in the collection of a pre-determined number n of enrollment D-vectors D_E, collected from the enrolled Speaker S_E. This set of D-vectors is called *the enrollment set* $\Delta_E = \{D_E^1, ..., D_E^i ..., D_E^n\}$, being D_E^i the i-th D-vector generated from the i-th Spectrogram P_E^i that contains the keyword k. The Learning phase is depicted in Fig. 5. In the on-device implementation described in Sect. VI, the value $n = 16$ was used, while different values of n were tested in the experiments.

2b) Inference phase: During the inference phase, the cosine similarity between the newly collected D-vector D_t extracted from $\Phi_f(\bullet)$ and all the other vectors in Δ_E is computed and the best-match cosine similarity $\sigma(\bullet)$, defined as follows, is computed:

$$\sigma(D_t, \Delta_E) = \max_{\{D_i \in \Delta_E\}} \frac{D_t \cdot D_i}{||D_t|| \cdot ||D_i||} \tag{4}$$

This value is compared to a user-defined threshold τ that can be tuned by the user in order to control the false positive vs false negative trade-off. Formally, the class z_t is assigned to D_T by using the formula:

$$z_t = \begin{cases} 1 & \text{if } \sigma > \tau \\ 0 & \text{if } \sigma \leq \tau \end{cases} . \tag{5}$$

We emphasize that during inference phase, this approach requires having enough memory to keep the entire set of enrollment D-vectors Δ_E stored, togheter with the memory to store the input D-vector D_t. This aspect is deepened in Sect. IV-E. The Inference phase is depicted in Fig. 6.

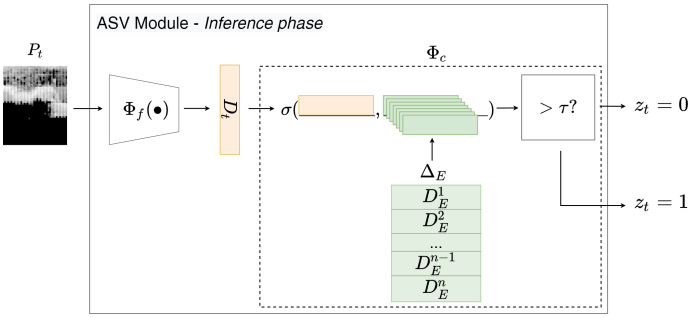

Fig. 6. The inference phase of the proposed adaptive speaker verification model.

D. The two-layer hierarchical solution

By executing the two proposed modules in a hierarchical manner, it is possible to enable the execution of TinySV on a tiny device. The pseudo-code provided in Alg. 1 describes the execution of the proposed two-layer hierarchical solution algorithm.

We enforce that, since the two algorithms are executed in a hierarchical fashion, the ASV module is executed only when the keyword k is detected by the KS module. In this sense, the KS module acts as a filter, almost halving the amount of computation that would be performed at each inference cycle if the two algorithms were being executed in parallel, and it ensures the quality of the data given as input to the ASV module by centering the window of the input data on the keyword.

E. Memory requirements

The memory requirements for each component of TinySV, i.e., the intermediate computations I_t, P_t, and D_t and the models Φ_k, Φ_f, and Φ_c, can be estimated with the formulas provided in Tab. III. We highlight that this estimation is system-agnostic, and thus does not consider any form of on-device optimization of a specific toolchain for the neural networks.

We emphasize that the memory of all the components can be computed as the product of the number of parameters required by the component and the precision b (e.g., 1 Byte, 4 Bytes ...) in which they are stored. In Tab. III the memory requirements of the components implemented in the on-device implementation in Sec. VI are also reported. For this estimation, the value $b = 4$B was considered for all the components except for I_t, which is stored with a $b_1 = 2$B precision.

Input: I_t
Output: $x_t \in \{0, 1, 2\}$
$P_t \leftarrow MFCC(I_t)$;
$y_t \leftarrow \Phi_k(P_t)$;
if $y_t == 1$ **then**
  $D_t \leftarrow \Phi_f(P_t)$;
  if $|\Delta_E| < n$ **then**
    $\Delta_E \leftarrow \Delta_E \cup D_t$;
  else
    $y_t \leftarrow \Phi_c(D_t)$;
    if $z_t == 0$ **then**
      $x_t \leftarrow 2$;
    else
      $x_t \leftarrow 1$;
    end
  end
else
  $x_t \leftarrow 0$;
end

Algorithm 1: Pseudocode of the proposed two-layer hierarchical solution

Component	Estimation formula	On-device mem. estimation
I_t	$(f_r \times W) \times b_1$	32 kB
P_t	$(i \times j) \times b$	7.5 kB
D_t	$d \times b$	1 kB
Φ_k - weights	$\omega_{\Phi_k} \times b$	101.44 kB
Φ_k - act.	$\alpha_{\Phi_k} \times b$	68.92 kB
Φ_f - weights	$\omega_{\Phi_f} \times b$	95.27 kB
Φ_f - act.	$\alpha_{\Phi_f} \times b$	104.12 kB
Φ_c	$(d \times n) \times b$	16 kB

TABLE III
MEMORY ESTIMATION FOR EACH COMPONENT OF TINYSV.

V. EXPERIMENTAL SETTING AND RESULTS

In this section, we describe the experiments performed to analyze the performance of the ASV module. The experimental setting is outlined in Sect. V-A. In Sect. V-B the two proposed comparison are detailed, while in Sect. V-C the experimental results are provided.

A. Experimental Setting

The experimental setting for the ASV module was designed keeping in mind the one-class, few-shot conditions described in Sect. III. The one-class condition has been ensured by enrolling one speaker at a time and using only samples from that speaker to perform the enrollment. The few-shot conditions have been tested by limiting the number of samples n used for the training phase. We provide the results for different values of n, i.e., $n = \{1, 8, 16, 64\}$.

1) The collected dataset: For the test of the proposed ASV model, we used a newly collected dataset comprising 376 recordings of the locution "Hey Cypress" pronounced by 4 different speakers (3 Male subjects and 1 Female, 94 recordings per subject). The mother tongue of all the speakers is Italian, so a possible bias in the English accent is present in the dataset. Training (68%), validation (16%) and test (16%) sets have been extracted from the dataset for each user.

The length of the recordings in the dataset is 1 second, compatible with the length of the proposed time window W. It is worth noting that a manual alignment of such samples has been performed to center the "Hey Cypress" phrase in the middle of the 1-second audio window.

2) The ASV module: In the ASV module used in the experiments, the implementation of Φ_f described in Tab. II was obtained from a model originally trained for a speaker classification task on the LibriSpeech-train-100 dataset [35]. Further details on the training of Φ_f can be found in the project repository. Φ_c has been evaluated by considering each combination of the enrolled speaker S_E and the number n of d-vectors used to build the model. The values that have been tested for the parameter n are $\{1, 8, 16, 64\}$, while all the four speakers in the dataset were used one at a time as S_E.

3) Metrics and evaluation: Four different metrics were selected for the evaluation of the proposed solution: accuracy, F1 score, Equal Error Rate (EER), and Area Under Curve (AUC). The first two figures of merit evaluate the performance of the algorithm on the testing set after the setting of the parameter τ, while the last ones are independent from that parameter and are computed on the validation set.

In order to compute the accuracy and F1 score results for each speaker, the tunable parameter τ was set to the threshold value corresponding to the Equal Error Rate for the speaker S computed on the validation set.

For all the figure of merit and values of n, we provide the average results of the 4 models of the speakers included in the dataset.

B. The proposed comparisons

As a comparison for the ASV module, we considered the following two solutions coming from the SV literature:

1) Mean Cosine Similarity (MCS): This solution maintains the same d-vector extractor $\Phi_f(\bullet)$ used in the proposed ASV module, but replacing the similarity metric $\sigma(\bullet)$ with the *mean cosine similarity*. This metric is common in the Speaker Verification literature, and it consists in computing the cosine similarity between D_t and D_{AVG}, extracted from Δ_E by computing the element-wise average of the d-vectors in the set. The memory requirements of this model are equal to $d \times b$, and, differently from the ones of the proposed Φ_c, it does not vary with n.

2) GE2E LSTM: To provide a comparison with a state-of-the-art system, we tested an implementation of the Speaker Verification algorithm described in [40] and [41]. Similarly to our ASV module, this solution encompasses a d-vector extractor Φ_f^{LSTM} and a similarity metric. Φ_f^{LSTM} is an LSTM neural network, with three layers, each containing 256 nodes. The network was trained with a *generalized end-to-end* loss that aims at training models that better emphasize the differences in the feature space. The similarity metric used in this work is the *Mean Cosine Similarity* described in the other comparison. This solution is not meant to be run on tiny devices, since Φ_f^{LSTM} requires more than 4MB only for storing the weights.

Technical details on the implementation of the two comparisons can be found in the project repository.

C. Experimental Results

The results of the proposed solution and of the comparison on the accuracy, F1 score. EER and AUC metrics are provided in Tab. IV.

solution	metric	$n = 1$	8	16	64	Tiny device
ASV(our)	Acc.	0.773	0.825	0.833	0.846	✓
	F1	0.639	0.708	0.732	0.739	
	EER	0.244	0.099	0.058	0.038	
	AUC	0.855	0.953	0.975	0.987	
MCS	Acc.	0.773	0.816	0.825	0.770	✓
	F1	0.639	0.703	0.725	0.725	
	EER	0.244	0.138	0.157	0.160	
	AUC	0.855	0.883	0.895	0.879	
GE2E [40]	Acc.	0.883	0.937	0.966	0.975	✗
	F1	0.815	0.876	0.932	0.946	
	EER	0.100	0.037	0.020	0	
	AUC	0.892	0.985	0.997	1	

TABLE IV
COMPARISON BETWEEN OUR ASV MODULE AND THE COMPARISONS.

The results show that our solution is extremely competitive with respect to the state-of-the-art solution meant to be run on larger, more flexible devices, while at the same time improving the state-of-the-art approach for tiny devices.

Indeed, in all the metrics, the proposed solution outperforms the *MCS* approach, particularly in the threshold-independent metrics EER and AUC, and with larger values of n. As expected, the *MCS* and ASV approaches are equivalent and have exactly the same performance in the case $n = 1$. Interestingly, the *MCS* approach reported the worst performance with $n = 64$, indicating that this type of model fatigues in incorporating the knowledge from larger, noisy enrollment datasets. The great differences in the EER and AUC metrics (i.e., 8% - 10%) between the proposed ASV and *MCS* indicate also that with the proposed *Best-Match Cosine Similarity* better tradeoffs are possible in the selection of the parameter τ.

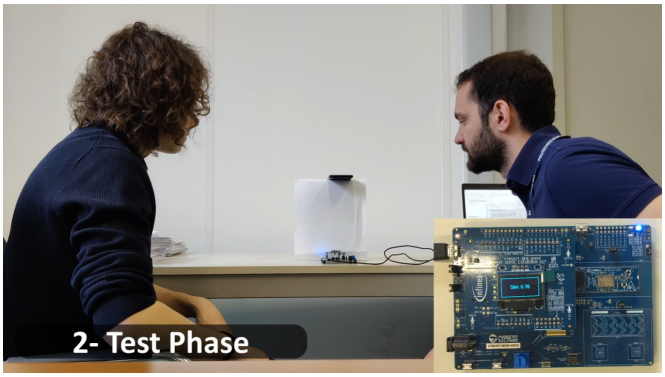

Fig. 7. A frame of the video demonstrating the on-device implementation of the system.

Compared to the *GE2E LSTM* approach, the proposed ASV approach has a reduction in performance in the order of 2% - 4% for threshold-independent metrics, and in the order of 10% - 20 % in the threshold-dependent metrics. The proposed solution is nevertheless at least an order of magnitude less memory-demanding, and thus can be executed on tiny devices.

VI. On-device implementation

The proposed TinySV solution has been implemented on an off-the-shelf hardware platform to test its performance in a real-world scenario. The aim of this section is to describe the on-device implementation of TinySV, in which both the enrollment phase and the inference phase are executed on the target device.

At startup, the TinySV demo application asks the user to provide the enrollment samples by pronouncing $n = 16$ times the keyword k = "Sheila". Afterwards, the model switches to the *inference phase* and recognizes if k was pronounced by the enrollment user S_E or not.

A video of the demo application can be found in the project repository, and a frame of the video is presented in Fig. 7.

The section is organized as follows. In Sec. VI-A the considered hardware platform is presented. In Sec. VI-B the implementation details are reported, while Sec. VI-C reports all the considerations on the measured memory occupations, power consumption and execution times.

A. The board

The considered hardware platform is the Infineon PSoC 62S2 Wi-Fi BT Pioneer Board, which is a programmable embedded system-on-chip, integrating a 150-MHz Arm® Cortex®-M4 as the primary application processor, a 100-MHz Arm Cortex-M0+ that supports low-power operations, up to 2 MB Flash and 1 MB SRAM, and the compatibility with Arduino™ shields. The application has been written to run on the Cortex®-M4 processor. The board is also equipped with RBG LEDs, and the Infineon CY8CKIT-028-SENSE shield, which contains a digital microphone and an OLED screen.

B. Implementation details

The system has been implemented using windows of $W = 1$ s and $f_r = 16$ KHz. Each I_t is consequently a 16000-element long vector. Windows are partly overlapped, and the overlapping of the window in seconds corresponds to 0.75 s, computed as $W - T_{\Phi_k}$ where T_{Φ_k} is the inference time of Φ_k.

For the training and validation of Φ_k the Google Speech Commands dataset [3] has been used, while Φ_f was obtained from a model originally trained for a speaker classification task on the LibriSpeech-train-100 dataset [35]. Details on the training of Φ_k and Φ_C can be found in the project repository.

C. Flash and RAM memory occupation, execution times, and power consumption

The on-device deployment to the board was performed through the use of the Infineon ModusToolbox [42], which was used also to measure the actual memory requirements on the board. The whole application requires about 356.73 kB of flash memory to be stored.

At runtime, the total RAM memory request is 391.92 kB. Details on the measured RAM memory occupation of each component can be found in Tab. V.

Component	Memory required
I_t	32 kB
P_t	7.5 kB
D_t	1 kB
Φ_k - weights	104.21 kB
Φ_k - act.	39.68 kB
Φ_f - weights	98.08 kB
Φ_f - act.	70.56 kB
Φ_c	16 kB
Other (application overhead)	22.89 kB
Total	**391.92 kB**

TABLE V
Measured runtime RAM memory occupation for each component of TinySV on the PSoC 6 MCU Board.

It's important to note that the toolbox implements a common optimization on the memory requirements for the activations of the neural networks [27], resulting in significantly smaller memory requirements with respect to the estimation provided in Tab. III.

The execution times of the two CNNs used in the application are reported in Tab. VI. Compared to their execution times, the execution time of Φ_c is negligible.

Component	Time (s)
T_{MFCC}	0.020
T_{Φ_k}	0.250
T_{Φ_f}	0.036
T_{Φ_c}	~ 0

TABLE VI
Execution Time measured for all the modules in the on-device implementation.

While executing the application, the MCU runs at 150 MHz which is the maximum clock speed. PSoC 6 MCU operates at 3.3V. Taking into account all the active peripherals, the

application consumes 19 mA of current, leading to a total power consumption of 62.7 mW. The expected runtime of the system when powered by a 1000mAh battery is 159 hours.

VII. Conclusions

The aim of this paper was to introduce a new type of adaptive TinyML solutions and a novel TinyML task, named TinySV, that requires the usage of on-device learning. The proposed two-layer hierarchical TinyML solution relies on two modules, i.e., Keyword Spotting and Speaker Verification, used in a cascade manner. The proposed solution adapts the TinyML model directly on-device with the data of the user, making use of a novel one-class, few-shot learning approach that deals with the lack of data and labels common to the TinyML environment. The effectiveness of the proposed solution has been successfully evaluated on a newly collected dataset that has been released to the scientific community. The efficiency of the solution has been demonstrated with the on-device implementation on an IoT device, the Infineon PSoC 62S2 Wi-Fi BT Pioneer Board, where the memory occupation, power consumption, and execution times have been evaluated.

Future works will encompass the exploration of methods to improve the d-vector extraction, the testing of other algorithms that can be trained with a few-shot, one-class approach, and the extension of the proposed methodology to other TinyML learning tasks that have been, until now, faced only with standard supervised learning methodologies, such as object detection in pictures.

Acknowledgment

References

[1] P. Warden and D. Situnayake, *TinyML: machine learning with TensorFlow Lite on Arduino and ultra-low-power microcontrollers*, first edition ed. Bejing Boston Farnham Sebastopol Tokyo: O'Reilly, 2020.

[2] C. Alippi and M. Roveri, "The (Not) Far-Away Path to Smart Cyber-Physical Systems: An Information-Centric Framework," *Computer*, vol. 50, no. 4, pp. 38–47, Apr. 2017, conference Name: Computer.

[3] P. Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition," *arXiv:1804.03209 [cs]*, Apr. 2018, arXiv: 1804.03209. [Online]. Available: http://arxiv.org/abs/1804.03209

[4] A. Chowdhery, P. Warden, J. Shlens, A. Howard, and R. Rhodes, "Visual Wake Words Dataset," *arXiv:1906.05721 [cs, eess]*, Jun. 2019, arXiv: 1906.05721. [Online]. Available: http://arxiv.org/abs/1906.05721

[5] M. Antonini, M. Pincheira, M. Vecchio, and F. Antonelli, "An adaptable and unsupervised tinyml anomaly detection system for extreme industrial environments," *Sensors*, vol. 23, no. 4, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/4/2344

[6] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, S. Regev, R. Rhodes, T. Wang, and P. Warden, "TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems," vol. Proceedings of the 4 th MLSys Conference, San Jose, CA, USA, p. 12, 2021.

[7] M. Roveri, "Is tiny deep learning the new deep learning?" in *Computational Intelligence and Data Analytics: Proceedings of ICCIDA 2022*. Springer, 2022, pp. 23–39.

[8] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," Apr. 2017, number: arXiv:1704.04861 arXiv:1704.04861 [cs]. [Online]. Available: http://arxiv.org/abs/1704.04861

[9] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," *CoRR*, vol. abs/1905.11946, 2019. [Online]. Available: http://arxiv.org/abs/1905.11946

[10] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," *arXiv:1712.05877 [cs, stat]*, Dec. 2017, arXiv: 1712.05877 version: 1. [Online]. Available: http://arxiv.org/abs/1712.05877

[11] J. Liu, S. Tripathi, U. Kurup, and M. Shah, "Pruning Algorithms to Accelerate Convolutional Neural Networks for Edge Applications: A Survey," *arXiv:2005.04275 [cs, stat]*, May 2020, arXiv: 2005.04275. [Online]. Available: http://arxiv.org/abs/2005.04275

[12] A. Irum and A. Salman, "Speaker verification using deep neural networks: A," *International Journal of Machine Learning and Computing*, vol. 9, no. 1, 2019.

[13] Y. Tu, W. Lin, and M.-W. Mak, "A survey on text-dependent and text-independent speaker verification," *IEEE Access*, vol. PP, pp. 1–1, 01 2022.

[14] R. Sanchez-Iborra and A. F. Skarmeta, "TinyML-Enabled Frugal Smart Objects: Challenges and Opportunities," *IEEE Circuits and Systems Magazine*, vol. 20, no. 3, pp. 4–18, 2020, conference Name: IEEE Circuits and Systems Magazine.

[15] S. Disabato and M. Roveri, "Reducing the Computation Load of Convolutional Neural Networks through Gate Classification," in *2018 International Joint Conference on Neural Networks (IJCNN)*. Rio de Janeiro: IEEE, Jul. 2018, pp. 1–8. [Online]. Available: https://ieeexplore.ieee.org/document/8489276/

[16] P. P. Ray, "A review on TinyML: State-of-the-art and prospects," *Journal of King Saud University - Computer and Information Sciences*, Nov. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157821003335

[17] C. Alippi, S. Disabato, and M. Roveri, "Moving Convolutional Neural Networks to Embedded Systems: The AlexNet and VGG-16 Case," in *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, Apr. 2018, pp. 212–223.

[18] M. Pavan, A. Caltabiano, and M. Roveri, "TinyML for UWB-radar based presence detection," in *2022 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2022, pp. 1–8, iSSN: 2161-4407.

[19] V. Rajapakse, I. Karunanayake, and N. Ahmed, "Intelligence at the Extreme Edge: A Survey on Reformable TinyML," arXiv, Tech. Rep. arXiv:2204.00827, Apr. 2022, arXiv:2204.00827 [cs, eess] type: article. [Online]. Available: http://arxiv.org/abs/2204.00827

[20] S. Disabato and M. Roveri, "Incremental On-Device Tiny Machine Learning," p. 7, 2020.

[21] S. Disabato and Roveri, "Tiny Machine Learning for Concept Drift," *arXiv:2107.14759*, jul 2021, arXiv: 2107.14759. [Online]. Available: http://arxiv.org/abs/2107.14759

[22] M. Rusci and T. Tuytelaars, "Few-shot open-set learning for on-device customization of keyword spotting systems," *arXiv preprint arXiv:2306.02161*, 2023.

[23] J. Lin, L. Zhu, W.-M. Chen, W.-C. Wang, C. Gan, and S. Han, "On-Device Training Under 256KB Memory," Jul. 2022, arXiv:2206.15472 [cs]. [Online]. Available: http://arxiv.org/abs/2206.15472

[24] V. Ramanathan, "Online On-device MCU Transfer Learning," p. 7.

[25] H. Ren, D. Anicic, and T. Runkler, "TinyOL: TinyML with Online-Learning on Microcontrollers," *arXiv:2103.08295 [cs, eess]*, Apr. 2021, arXiv: 2103.08295. [Online]. Available: http://arxiv.org/abs/2103.08295

[26] L. Ravaglia, M. Rusci, D. Nadalini, A. Capotondi, F. Conti, L. Benini, and L. Benini, "A TinyML Platform for On-Device Continual Learning with Quantized Latent Replays," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, pp. 1–1, 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9580920/

[27] M. Pavan, E. Ostrovan, A. Caltabiano, and M. Roveri, "TyBox: an automatic design and code-generation toolbox for TinyML incremental on-device learning," *ACM Transactions on Embedded Computing Systems*, p. 3604566, Jun. 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3604566

[28] B. Sudharsan, P. Yadav, J. G. Breslin, and M. Intizar Ali, "Train++: An Incremental ML Model Training Algorithm to Create Self-Learning IoT Devices," in *2021 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/IOP/SCI)*, Oct. 2021, pp. 97–106.

[29] P. Warden, "Why isn't there more training on the edge?" Online, Sep. 2020. [Online]. Available: https://petewarden.com/2022/09/06/why-isnt-there-more-training-on-the-edge/

[30] N. Dehak, P. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-end factor analysis for speaker verification," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 19, pp. 788 – 798, 06 2011.

[31] X. Yuan, G. Li, J. Han, D. Wang, and Z. Tiankai, "Overview of the development of speaker recognition," *Journal of Physics: Conference Series*, vol. 1827, p. 012125, 03 2021.

[32] E. Variani, X. Lei, E. McDermott, I. L. Moreno, and J. Gonzalez-Dominguez, "Deep neural networks for small footprint text-dependent speaker verification," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, may 2014.

[33] G. Heigold, I. Moreno, S. Bengio, and N. Shazeer, "End-to-end text-dependent speaker verification," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 5115–5119.

[34] Y.-h. Chen, I. L. Moreno, T. Sainath, M. Visontai, R. Alvarez, and C. Parada, "Locally-connected and convolutional neural networks for small footprint speaker recognition," 2015.

[35] V. Panayotov, G. Chen, D. Povey, and S. Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2015, pp. 5206–5210.

[36] X. Qin, H. Bu, and M. Li, "Hi-mia: A far-field text-dependent speaker verification database and the baselines," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 7609–7613.

[37] A. Larcher, K. A. Lee, B. Ma, and H. Li, "Text-dependent speaker verification: Classifiers, databases and rsr2015," *Speech Communication*, vol. 60, pp. 56–77, 2014.

[38] Y. Zhang, N. Suda, L. Lai, and V. Chandra, "Hello edge: Keyword spotting on microcontrollers," *arXiv preprint arXiv:1711.07128*, 2017.

[39] T. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," 2015.

[40] L. Wan, Q. Wang, A. Papir, and I. L. Moreno, "Generalized end-to-end loss for speaker verification," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 4879–4883.

[41] YistLin, "Generalized end-to-end loss for speaker verification," https://github.com/yistLin/dvector, 2023.

[42] "Infineon modus toolbox," https://www.infineon.com/cms/en/design-support/tools/sdk/modustoolbox-software/, accessed: 2023-10-17.