Tiny Machine Learning for Concept Drift

Simone Disabato and Manuel Roveri, Senior Member, IEEE

Abstract—Tiny machine learning (TML) is a new research area whose goal is to design machine and deep learning (DL) techniques able to operate in embedded systems and the Internet-of-Things (IoT) units, hence satisfying the severe technological constraints on memory, computation, and energy characterizing these pervasive devices. Interestingly, the related literature mainly focused on reducing the computational and memory demand of the inference phase of machine and deep learning models. At the same time, the training is typically assumed to be carried out in cloud or edge computing systems (due to the larger memory and computational requirements). This assumption results in TML solutions that might become obsolete when the process generating the data is affected by concept drift (e.g., due to periodicity or seasonality effect, faults or malfunctioning affecting sensors or actuators, or changes in the users’ behavior), a common situation in real-world application scenarios. For the first time in the literature, this article introduces a TML for concept drift (TML-CD) solution based on deep learning feature extractors and a \( k \)-nearest neighbors (\( k \)-NNs) classifier integrating a hybrid adaptation module able to deal with concept drift affecting the data-generating process. This adaptation module continuously updates (in a passive way) the knowledge base of TML-CD and, at the same time, employs a change detection test (CDT) to inspect for changes (in an active way) to quickly adapt to concept drift by removing obsolete knowledge. Experimental results on both image and audio benchmarks show the effectiveness of the proposed solution, whilst the porting of TML-CD on three off-the-shelf micro-controller units (MCUs) shows the feasibility of what is proposed in real-world pervasive systems.

Index Terms—Adaptation, concept drift, deep learning (DL), \( k \)-nearest neighbor (\( k \)-NN), tiny machine learning (TML).

I. INTRODUCTION

INTERNET-OF-THINGS (IoT) and embedded systems are nowadays part of our everyday life in a wide range of application scenarios (e.g., automotive, medical devices, and smart cities, to name a few). In recent years, the scientific and technological trend about these pervasive devices is to move the processing (and in particular the intelligent processing) as close as possible to where data are generated. The reason is twofold. First, the IoT units and embedded systems already operate pervasively in the environment processing large amounts of data acquired by the sensors. Second, machine and deep learning (DL) solutions processing these data directly on the pervasive devices are crucial to support real-time applications, prolong the system lifetime, and increase the quality-of-service. Nevertheless, machine and deep learning solutions are typically characterized by memory and computational demands that rarely match the constraints on memory, computation, and energy characterizing the IoT units and embedded systems [1], [2], [3].

Tiny machine learning (TML) [4] is a relatively new research area aiming at filling this gap by designing “tiny” machine and deep learning solutions able to run on the IoT units and embedded systems. Section II analyses the related literature, highlighting that most TML solutions focus on approximation, pruning, and quantization mechanisms to reduce memory and computational demand of machine and deep learning models. Although these solutions run on embedded systems and the IoT units, their training is typically carried out on high-performing units (such as cloud or edge computing systems), with very few papers proposing on-device incremental learning mechanisms [5], [6].

The ability to learn TML models directly on the devices is crucial to improve the accuracy over time by exploiting fresh information coming from the field, and to deal with concept drift, i.e., variations in the statistical behavior of the data-generating process, a quite common situation in real-world applications (e.g., due to seasonality or periodicity effects, faults affecting sensors or actuators, changes in the user’s behavior, or aging consequences). Failing to adapt TML models to concept drift results in a (possibly dramatic) decrease of the accuracy over time [7].

This article aims at addressing this challenge by introducing, for the first time in the literature, a TML algorithm for concept drift (TML-CD) that can learn directly on the IoT unit or embedded system and adapt the knowledge base in response to a concept drift (thus tracking the evolution of the data generating process). In order to allow TML-CD being able to effectively adapt to concept drift, we here introduce three adaptation mechanisms, i.e., passive, active, and hybrid—that take into account the constraints on memory (and computation) of the device they are deployed. The proposed TML-CD is meant to operate in a supervised setting [7], [8], hence activating the adaptation when the supervised information is made available, e.g., a vocal assistant or a smart doorknob that interact with the users. Among these three adaptation mechanisms, we focus on the hybrid solution thanks to its ability to tradeoff adaptation with memory demand. The three proposed TML-CD adaptive mechanisms have been tested in two different application scenarios (i.e., image classification and speech command recognition) and ported to three real-world micro-controller units (MCUs), showing their feasibility and effectiveness. Finally, the code is made available to the scientific community.1

1The repository link is https://github.com/simdisis/Adaptive-TML.
The article is organized as follows. Section II revises the related literature. Section III formalizes the addressed problem, whereas Sections IV–VI present the proposed TML-CD solution and its stages. Finally, Section VII details the experimental results and Section VIII draws the conclusion.

II. RELATED LITERATURE

This section discusses the related literature about machine and deep learning solutions in presence of concept drift as well as TML solutions.

A. Machine and Deep Learning Solutions in Presence of Concept Drift

The literature about machine and deep learning in presence of concept drift refers to adaptive solutions able to deal with concept drift affecting the data-generating process. The related literature usually groups them into two main families: 1) passive and 2) active [7], [8], [9].

Passive solutions adapt the model at each incoming data, disregarding the fact that a concept drift has occurred in the data-generating process (or not). The gradual forgetting classifiers, e.g., [10], [11], which reduce the importance of older data-generating process (or not), is an example of passive solutions. The concept drifts very fast decision tree (CDVFDT) [12] introduces a decision tree that learns new subtrees on incoming data. However, most passive solutions employ ensemble methods and their adaptation mechanisms consist in adding, removing, or weighting the ensemble base classifiers, e.g., streaming ensemble algorithm [13], dynamic classifier selection [14], or the adaptive ensemble of decision trees proposed in [15]. Deep learning-based passive solutions can be found in [16], [17], and [18].

On the contrary, active solutions aim at detecting concept drift in the data generation process and, only in that case, they adapt their model to the new conditions. Change detection tests (CDTs) are statistical techniques meant to sequentially process the incoming data inspecting for concept drift. Dittrler and Polikar [19] proposed to use the Hellinger distance between the reference probability distribution and the one estimated on incoming data along with a t-test to detect changes. Dasu et al. [20] rely on bootstrapping several windows of data and the Kullback–Leibler divergence as a measure of the distance among them. A few works detect changes with density estimation techniques [21], [22]. Other examples of CDT used in active solutions can be found in [23], [24], [25], and [26]. In active solutions, the adaptation stage following a concept drift detection is usually carried out in two steps [27]: first, the instant the concept drift occurred is estimated by ad hoc mechanisms (e.g., by change-point methods); second, the obsolete knowledge, i.e., that acquired before the concept drift occurred, is discarded. To achieve this goal, the adaptation mechanisms typically rely on a window over the last acquired data, whose size is usually optimized over time to reduce its memory requirements [28], [29], or on all the samples seen so far (suitably weighted) [30]. Finally, examples of adaptive classifiers are the ADaptive WINdow (ADWIN)-k-NN [31] or the self-adjusting memory (SAM)-k-NN [32], whereas deep learning-based active approaches (integrating deep learning solutions with active adaptive solutions) can be found in [33], [34], and [35]. In particular, Vaquet et al. [34] recently suggested a pipeline that is close to ours, with a deep learning feature extractor prior to a classifier, in an online learning scenario.

B. Tiny Machine Learning

TML techniques aim at designing machine and deep learning models that take into account the severe technological constraints on memory, computation, and energy characterizing IoT units and embedded systems [2], [4], [6].

To achieve this goal, most solutions employ approximation techniques from the deep learning literature. These approximation mechanisms can be grouped into three main families according to the way the approximation is carried out: pruning of processing layers (and part of them) [36], [37], quantization of parameters and activations with limited precision or binary parameters [38], [39], [40], or solutions integrating both pruning and quantization [1].

As regards the target of the approximation mechanisms, most of the TML literature focuses on approximated convolutional neural networks [41], [42], [43], [44], with a few works considering recurrent DL architectures [45], [46]. In particular, Fedorov et al. [47] introduces a methodology to explore sparse (and pruned) convolutional neural network (CNN) architectures able to be executed on microcontroller units, whereas [5] proposes a tiny-CNN whose biases can be learned directly on the device. Finally, Rusci et al. [48] investigates the impact of quantized networks in TinyML embedded systems.

Although there are very few works proposing on-device learning, e.g., [5], [6], and [49], to the best of our knowledge, no work presents a Tiny-ML solution able to adapt over time to concept drift.

III. PROBLEM FORMULATION

Let $\mathcal{P}$ be a data generating process that, at each instant $t$, provides a pair $(x_t, y_t)$ sampled from an unknown probability distribution $p_t(x, y)$, where $x$ is the input of the proposed solution $D$ (e.g., an image or an audio clip) and $y \in \Delta$ its classification label.\footnote{Without any loss of generality, the supervised information might not be provided at every instant. In those cases, the proposed solution only provides its classification output.}

Moreover, following a test-then-train approach [7], the proposed solution $D$ receives the supervised information (the true label $y_t$) only after it provides the classification output $\hat{y}_t = D(x_t)$ on input $x_t$, at each instant $t$.\footnote{Let $\delta$ be the cardinality of $\Delta$, i.e., the number of classes in the considered classification problem.}

In a concept drift scenario, the process $\mathcal{P}$ might evolve over time, hence inducing a shift in the distribution $p_t(x, y)$ at an unknown instant $t^\ast$. It is worth noting that the change in $p_t(x, y)$ might affect the input $x$ (e.g., by the introduction of noise), the set $\Delta$ (e.g., class change), or both [7] and [34].

The goal of the proposed TML-CD solution is to react and adapt $D$ to changes in $p_t(x, y)$ so as to guarantee the highest accuracy over time.
IV. PROPOSED TML FOR CONCEPT DRIFT (TML-CD)

Fig. 1 shows the general architecture of the proposed solution for TML-CD, which comprises the following five different modules as follows.

- **Feature Extractor $\varrho$**: The feature extractor extracts features from the input $x_t$. As in [1] and [6], the feature extractor is a pretrained DL model approximated by means of task-dropping (e.g., pruning of layers), precision scaling (e.g., weights precision reduction), or both, to satisfy the constraints on computation, memory, and energy characterizing the embedded systems and IoT units running $D$.

- **Dimensionality Reduction Operator $\varsigma$**: The dimensionality reduction operator $\varsigma$ (that can be optionally activated) reduces the dimensionality of features extracted by $\varrho$. In this article, among the approaches presented in [6], we focused on the filter-selection without supervised information. This technique selects the $f$ out of $F$ filters of the last $\varrho$ convolutional layer (and its subsequent batch-normalization channels, if any) providing the highest mean activation on publicly available benchmarks or datasets. It is crucial to point out that the adaptation step of $D$ does not affect the feature extractor $\varrho$ nor the dimensionality reduction operator $\varsigma$, which are therefore fixed over time. Moreover, since the choice of the $f$ filters to keep does not rely on the specific data-generation process $\mathcal{P}$, the block $\varsigma \circ \varrho$ can be defined at design time and prior to the porting of $D$ on the IoT units. This is the reason why $\varsigma \circ \varrho$ is an input to our algorithm, and $\varsigma$ takes part in the choice of the approximated DL-feature extractor.

- **k-NN Classifier and Training Set $T$**: The k-NN [50] classifier $K(\cdot)$, whose input is either the output of the dimensionality-reduction operator $\varsigma \circ \varrho$ or that of the feature extractor $\varrho$ (when no dimensionality reduction is considered), provides the classification $\hat{y}_t$ of the input $x_t$, while $T$ is its training set. From the algorithmic point of view, the k-NN is a statistical classifier based on majority voting, i.e., the predicted class corresponds to the majority class of the $k$-nearest neighbors (k-NNs) of the input sample within $K$’s training set $T$. Interestingly, it does not require a training phase, but only the initialization of its training set $T$. Unless otherwise specified, the parameter $k$, i.e., the number of neighbors, is set to the ceiling of the square root of the available samples, as suggested in [27].

- **Adaptation Module**: The adaptation module receives as input the sample $x_t$ and its k-NN $K$ prediction $\hat{y}_t$ and, when the supervised information $y_t$ is available, it updates the TML-CD solution $D$ so as to make it adaptive over time to concept drift. Among the four presented modules, the adaptation involves only the $K$’s training set $T$ [32], [51]. The k-NN classifier adaptation indeed requires to simply add the new supervised information $(x_t, y_t)$ to its training set $T$.

Algorithm 1, instead, details how the proposed TML-CD $D$ works. More in detail, the TML-CD $D$, which receives in input a feature extractor along with a dimensionality reduction operator ($\varsigma \circ \varrho$) and an initial training set $T$, comprises two different stages: 1) configuration and 2) testing.

The **configuration** stage, detailed in lines 1–3 and shown in Fig. 1(a), encompasses an initial preprocessing step where the training set $T$ is preprocessed to reduce the memory occupation (line 1) by means of a condensing mechanism (Algorithm 2). Once the preprocessing step has been carried out, the knowledge base of the k-NN classifier is initialized on the features extracted from the preprocessed training set $T$, i.e., the training set of $K$ is $\varsigma \circ \varrho(T)$. Section V will detail the configuration stage.

After the completion of the configuration stage, the TML-CD solution $D = K \circ \varsigma \circ \varrho$ enters the testing stage where it is able to operate on the novel incoming samples provided by the data-generating process $\mathcal{P}$ (lines 4–6). At each instant $t = 1, 2, \ldots$, the proposed solution $D$ receives in input $x_t$ and provides the output $\hat{y}_t = D(x_t)$ (line 5). Then, when the supervised information $y_t$ about $x_t$ is made available as per the “test-and-train” approach, it activates the adaptation step (line 6). Section VI will detail the testing stage by describing the proposed three adaptive mechanisms for TML-CD.

V. CONFIGURATION STAGE: CONDENSING $T$

The k-NN classifier has the great advantage of not requiring a proper training phase. However, this advantage comes, in principle, at the expense of the following two drawbacks. First, a k-NN-based classifier requires to store all the data of the training set. Second, the larger the amount of the training
data, the higher the time to provide a classification in output. These drawbacks are more severe as the samples within \( T \) increase.

The related literature addresses these two issues from three different perspectives.

First, condensing techniques \([52], [53]\) aim at identifying the smallest subset of training data that can correctly classify all the training samples. Second, editing techniques \([54], [55], [56]\) instead reduce the number of stored samples by removing the noisy ones, i.e., those not agreeing with their neighborhoods. Third, Smith et al. \([57]\) proposed to train a supervised parametric classifier on available data and to remove all the samples having a classification probability below a hard threshold.

In this work, we focus on the first approach and, in particular, on the condensed nearest neighbor algorithm. In more detail, \( D \) applies this algorithm during the preprocessing step (Algorithm 1–Line 1) in order to optimize both the memory and computational requirements of the classifier \( \mathcal{K} \). More specifically, given a \( N \)-dimensional training set \( T = \{(x_t, y_t), t = 1, \ldots, N\} \), the preprocessing step computes the condensed representation of \( T \), i.e., the minimum subset \( \tilde{T} \subseteq T \) for which \( \mathcal{K} \) is able to correctly classify all the samples in \( T \). Algorithm 2 shows the pseudo-code of the condensing algorithm proposed by Hart \([52]\) that is employed in the preprocessing step. Section VII-D experimentally evaluates the impact on accuracy and memory demand of this condensing stage, highlighting that the significant savings in terms of memory come at the expense of a negligible drop in accuracy (in stationary conditions).

\begin{algorithm}
\caption{The Condensed Nearest Neighbor \([52]\)}
\begin{algorithmic}
\State \textbf{Input:} Training Set \( T \).
\State \textbf{Output:} Condensed Representation \( \tilde{T} \subseteq T \).
\State \( H \) contains one sample, \( D \) all the others.
\State Initialize \( H \leftarrow \{t \in T\} \) and \( D \leftarrow T \setminus H \).
\State Initialize the \( k \)-NN \( \mathcal{K} \) with \( H \).
\For\( t \leftarrow (x, y) \in D \)
\State Predict \( \hat{y} = \mathcal{K}(t) \).
\If\( \hat{y} \neq y \) \Comment{Condensing Update:}
\State \( D \leftarrow D \setminus \{t\} \). \Comment{Move \( t \) from \( D \) to \( H \).}
\State \( H \leftarrow H \cup \{t\} \).
\EndIf
\EndFor
\While \( H \) and \( D \) are modified in the foreach loop.
\State return \( \tilde{T} \leftarrow H \)
\end{algorithmic}
\end{algorithm}

VI. Testing Stage: Adapting \( T \)

The adaptation module, which is the core of the proposed \( \mathcal{D} \), has been declined from three different perspectives, differing in the type of adaptation mechanism therein employed as follows.

1) \textbf{Passive Update (Section VI-A):} The adaptation module relies on a fully passive approach where the adaptation is carried out at each new incoming supervised samples without requiring an explicit detection of a change in the data-generating process \( \mathcal{P} \).

2) \textbf{Active Update (Section VI-B):} This adaptation module relies on a CDT to detect changes in \( \mathcal{P} \). Once a change is detected, the algorithm adapts \( \mathcal{K} \) accordingly.

3) \textbf{Hybrid Update (Section VI-C):} The hybrid adaptation module integrates the passive approach with a CDT to speed up the adaptation stage exactly when needed.

It is worth nothing that the adaptation phase is activated only when the supervised information is available. This is quite a general assumption in the field of learning in presence of concept drift \([7], [8]\). However, when the supervised information cannot be provided frequently enough, semi-supervised active or hybrid adaptation mechanisms could be designed. This is an open point in the literature and we will consider it the next step of our work.

A. Passive Update: The Condensing-in-Time (CIT) Approach

The passive approach, called CIT algorithm, updates the training set \( T \) every time a new supervised sample is available. Algorithm 3 presents the CIT algorithm.

It receives in input the feature extractor along with a dimensionality reduction operator \( \varsigma \circ \varrho \) and training set \( T \), whose condensed representation \( \tilde{T} \subseteq T \) (see Algorithm 2) is used to initialize the training set \( T \) of the \( k \)-NN. Once initialized, the CIT–\( \mathcal{D} \), i.e., the \( \mathcal{D} \) solution implementing the CIT adaption module, is ready to classify novel incoming samples. The CIT passively updates the \( \mathcal{K} \)'s knowledge set \( T \) at every instant \( t \) for which the supervised information, i.e., the true label \( y_t \), is available. More in detail, CIT–\( \mathcal{D} \) adds the sample \( x_t \) and its true label \( y_t \) (at time \( t \)) to the \( k \)-NN \( \mathcal{K} \) knowledge set \( T \) if and only if \( x_t \) is misclassified, i.e., \( \hat{y}_t \neq y_t \) (Algorithm 3, Lines 6–11). This idea is inspired by the condensing algorithm update, shown at Lines 6–9 in Algorithm 2, but it is here tailored to the time evolution of the data-generating process.

It is worth noting that the CIT algorithm can only add a new supervised sample to the knowledge set \( T \) of the \( k \)-NN \( \mathcal{K} \), hence potentially introducing critical issues in the memory and computational demand of the \( k \)-NN when the number of

\begin{algorithm}
\caption{The Condensing-in-Time (Passive)}
\begin{algorithmic}
\State \textbf{Input:} Training Set \( T \), Feature Extractor \( \varsigma \circ \varrho \).
\State \textbf{Parameters:} Maximum number of training samples \( p \).
\State 1 Compute \( T \leftarrow \tilde{T} \) with Algorithm 2. \Comment{Condense \( T \).}
\State 2 Initialize the \( k \)-NN \( \mathcal{K} \) with \( \varsigma \circ \varrho (T) \).
\State 3 \Comment{Loop over samples arriving at time \( t \).}
\For\( (x_t, y_t) \sim \mathcal{P}, t = 1, 2, \ldots \)
\State Predict \( \hat{y}_t \leftarrow \mathcal{D}(x_t) \).
\If\( \hat{y}_t \neq y_t \) \Comment{Passive Update.}
\State \( T \leftarrow T \cup (x_t, y_t) \).
\EndIf
\If\( |T| > p \) \Comment{Window Size Check.}
\State \( (x_t, y_t) \leftarrow \arg \min \{ (x_i, y_i) \in T \} \).
\State \( T \leftarrow T \setminus \{(x_t, y_t)\} \).
\EndIf
\EndFor
\State Update \( \mathcal{D} \) with \( T \).
\end{algorithmic}
\end{algorithm}
Algorithm 4 The Active Tiny $k$-NN

**Input:** Feature Extractor $\varsigma \circ \varrho$, CDT $\vartheta$, Training Set $T$

**Parameters:** History Window Size $\sigma$, CDT threshold $h$

1. Compute $\tilde{T} \leftarrow T$ with Algorithm 2. $\triangleright$ Condense $T$.
2. Initialize the $k$-NN classifier $K$ with $\varsigma \circ \varrho (T)$.
3. Define $D = K \circ \varsigma \circ \varrho$.
4. Initialize $W \leftarrow \emptyset$. $\triangleright$ History Window.
5. Loop over samples arriving at time $t$.
6. foreach $(x_t, y_t) \sim P, t = 1, 2, \ldots$
   a. Predict $\hat{y}_t \leftarrow D(x_t)$.
   b. $W \leftarrow W \cup \{(x_t, y_t)\}$. $\triangleright$ Update History Window.
   c. if $|W| \geq \sigma$ then
      d. \quad $W \leftarrow W \setminus \{(x_{t-\sigma}, y_{t-\sigma})\}$.
      e. $s_t \leftarrow$ CDT metric as in Eq. (1). $\triangleright$ Active Step.
      f. $g_t \leftarrow \vartheta(s_1, \ldots, s_t)$. $\triangleright$ Apply CDT.
   g. if $g_t \geq h$ then $\triangleright$ Change Detection Check.
   h. $t_c \leftarrow$ Estimated Real Change Time as in Eq. (5).
   i. $T \leftarrow \{(x_t, y_t) \in W : \bar{t} \geq t_c\}$. $\triangleright$ Novel Samples.
   j. \[Optional\] Condense $T$ with Algorithm 2.
   k. Update $D$ with $T$.

samples in $T$ increases. To keep under control the cardinality of $T$, the CIT algorithm employs two different solutions. The former introduces a maximum number of samples $p$ that can be stored, i.e., $|T| \leq p$, being $|\cdot|$ cardinality operator. Hence, every time the adaptation stage introduces a sample in $T$ overcoming this limit, the oldest sample is removed (Algorithm 3, Lines 8–10). As a consequence, the solution $D$ based on the CIT classifier operates on the last $p$ supervised samples introduced in $T$. Besides, this mechanism allows also to remove old samples in $T$ by introducing only misclassified samples, i.e., those bringing more information to the classifier.

The latter introduces a probability for a misclassified sample to be added to the training set $T$. Ideally, such probability should be close to zero in stationary conditions and close to one immediately after a change. The definition of this probabilistic memory management mechanism is left as future work.

**B. Active Update: Active Tiny $k$-NN**

The active tiny $k$-NN, whose pseudocode is shown in Algorithm 4, relies on a CDT $\vartheta$ to detect changes in the data generation process $P$. The core of this algorithm is the ability to adapt the classifier $K$’s training set $\tilde{T}$ only after the detection of a concept drift. In addition, the active tiny $k$-NN allocates space for a history window $W$ of size $\sigma$, being $\sigma$ a parameter of the algorithm (described in the sequel).

In more detail, for each sample $(x_t, y_t)$ provided by $P$ at instant $t$, the active tiny $k$-NN predicts the label $\hat{y}_t = D(x_t)$ and, when the supervised information $y_t$ is available, the active update is activated (Algorithm 4, Lines 7–16).

At first, it adds the pair $(x_t, y_t)$ to the history window $W$ and discards the oldest pair if the window already contains $\sigma$ pairs (Algorithm 4, Lines 7–8). After that, the Active Tiny $k$-NN computes the figure of merit $s_t$ (at time $t$) and applies the CDT decision function $\vartheta$ to inspect for changes in $P$ (Algorithm 4, Lines 10–11), i.e., $g_t = \vartheta(s_1, \ldots, s_t)$. In the most general situation, the computation of $g_t$ at time $t$ takes into account all the figures of merits computed from $t = 1$. A change is detected in the data-generation process $P$ when $g_t$ overcomes the detection threshold $h$, being $h$ a parameter of the algorithm (Algorithm 4, Line 12).

Once a change is detected, the adaptation stage starts (Algorithm 4, Lines 13–16). In the first place, it estimates the time $t_c$ the change occurred (e.g., with a change point method). After that, it discards from the history window $W$ all the samples older than the estimated change time $t_c$. The updated history window $W$ (optionally condensed through Algorithm 2) becomes the new $K$’s training set $\tilde{T}$.

It is noteworthy to point out that the memory footprint of the active tiny $k$-NN is bounded over time since it requires to store the training set $\tilde{T}$ and history window $W$ of at most $\sigma$ samples (the CDT memory footprint can be neglected). Moreover, since the adaptation stage modifies the knowledge set $\tilde{T}$ only through copies of the (at most whole) history window $W$, the total memory footprint cannot overcome twice the memory of the history window $W$, i.e., that of $2\sigma$ samples.

Although the solution accepts as input any CDT $\vartheta$, in the context of this article, $\vartheta$ is the well-known and theoretically grounded CUSUM algorithm [24] in its generalized version [58], monitoring the accuracy of the active tiny $k$-NN over time. As a consequence, any change in the data-generation process $P$ is assumed to reflect on the $K$ classification accuracy.

The generalized CUSUM CDT is designed as follows. Let $n_0$ be the stationary classification accuracy (estimated on the first $\varsigma$ supervised samples in the testing stage, being $\varsigma$ parameter of the active tiny $k$-NN algorithm). A Bernoulli distribution with parameter $n_0$ and, in turn, a Binomial distribution with parameters $n_0$ and $n$ (with $n$ size of the batches on which the accuracy is computed in the following) model our scenario in stationary conditions. The figure of merit of the CUSUM CDT is the likelihood ratio of the probability distributions modeling the scenario after and before the change, i.e.,

$$ s_t = \ln \frac{p_{n_0}(\zeta_t)}{p_{n_0}(\zeta_t)} \quad (1) $$

where $p_{n_1}$ represents the classification accuracy after the change and $\zeta_t$ the realization of the Binomial distribution at time $t$, i.e., the accuracy on the $n$ supervised samples arrived before time $t$.

4 Although the Active Tiny $k$-NN algorithm is general enough to deal with any CDT, in the described CUSUM case with Binomial distribution of size $n$, the CDT figure of merit is not computed for every supervised sample, but every $n$. As a consequence, all the $n - 1$ $s_t$ values before a window of size $n$ is full are set to zero.

5 The cardinality of $\varsigma_t$, i.e., the number of tested values $\varsigma_t$, is a parameter of the active tiny $k$-NN.
Algorithm 5 The Hybrid Tiny k-NN

Input: Feature Extractor $\varsigma \circ \varrho$, CDT $\vartheta$, Training Set $T$.
Parameters: Maximum $T$ Size $\sigma$, CDT threshold $h$.
1 Compute $T \leftarrow T$ with Algorithm 2. ▷ Condense $T$.
2 Initialize the k-NN classifier $K$ with $\varsigma \circ \varrho(T)$.
3 Define $D = K \circ \varsigma \circ \varrho$.
▷ Loop over samples arriving at time $t$.
4 foreach $(x_t, y_t) \sim \mathcal{P}, t = 1, 2, \ldots$ do
5 Predict $\hat{y}_t \leftarrow D(x_t)$.
6 if $\hat{y}_t \neq y_t$ then ▷ Passive Update.
7 $T \leftarrow T \cup (x_t, y_t)$
8 if $|T| \geq \sigma$ then
9 $t_{\text{min}} \leftarrow \min\{t(x_t, y_t) \in T\}$
10 $T \leftarrow T \setminus \{(x_{t_{\text{min}}}, y_{t_{\text{min}}})\}$
11 Update $D$ with $T$.
12 $s_t \leftarrow \text{CDT metric as in Eq. (1)}$. ▷ Active Step.
13 $g_t \leftarrow \vartheta(s_1, \ldots, s_t)$. ▷ Apply CDT.
14 if $g_t \geq h$ then ▷ Change Detection Check.
15 $t_r \leftarrow \text{Estimated Real Change Time as in Eq. (5)}$.
16 $T \leftarrow \{(x_t, y_t) \in T : t \geq t_r\}$. ▷ Novel Samples.
17 [Optional] Condense $T$ with Algorithm 2.
18 Update $D$ with $T$.

decision function $\vartheta$ is

$$g_t = \vartheta(s_1, \ldots, s_t) = \max_{1 \leq j \leq t} \sup_{v_i \in T_1} S_j'(v_1)$$

where

$$S_j'(v_1) = \sum_{i=j}^t s_i$$

represents the sum of the log-likelihood ratios $s$ from time $j$ to time $t$.

Assuming the parameter $n$ is large enough, the considered Binomial distribution can be approximated as a normal one with mean $n v_0$ and variance $n v_0 (1 - v_0)$. Hence, the log-likelihood ratio $s_j$ in (1) becomes

$$s_j = \frac{v_1 \bar{b}_j - v_0 \bar{b}_0}{2 n v_0 \bar{b}_0 v_1 \bar{b}_1} + \frac{\bar{b}_0 - \bar{b}_1}{\bar{b}_0 \bar{b}_1} \zeta_j + \frac{n(v_0 \bar{b}_1 - v_1 \bar{b}_0)}{2 \bar{b}_0 \bar{b}_1} + \ln \sqrt{\frac{v_0 \bar{b}_0}{v_1 \bar{b}_1}}$$

where $\bar{b}_0 = 1 - v_0$ and $\bar{b}_1 = 1 - v_1$.

As a final remark, the CUSUM CDT is also endowed with the ability to estimate the change time $t_r$ (and, if desired, of the parameter $v_1$ after the change). The estimated change time $t_r$ is indeed the index $j$ maximizing the decision function $\vartheta$ in (2), i.e.,

$$\left(\tilde{j}, \tilde{v}_1\right) = \arg \max_{1 \leq j \leq t} \sup_{v_i \in T_1} S_j'(v_1).$$

C. Hybrid Tiny k-NN: Integrating CIT and Active Tiny k-NN

The core of the proposed hybrid update is to integrate the “CIT” ability of the passive update with the capability to quickly adapt to changes by discarding obsolete knowledge of the active one.

In more detail, the (CIT) passive update continuously adapts $T$ when supervised information is available, regardless of a concept drift occurred (or not). This ability comes at the expense of two weak points. First, there is (in principle) no bound on the memory occupancy, although two solutions have been suggested to mitigate the problem. Second, when a change occurs, the passive update does not discard the obsolete knowledge present in $T$, i.e., samples generated by $P$ before the concept drift occurred.

On the contrary, the active adaptation provides a bound on the memory occupancy (i.e., twice the history window size $W$) and, in turn, on the required computation. However, similar to the other active approaches present in the literature [23], [25], and [33], the effectiveness of the active adaptation phase is strictly related to the ability to promptly detect the concept drift in $P$.

The proposed hybrid update aspires at compensating the weak points of passive and active updates by integrating the “CIT” solution described in Algorithm 2 with the CUSUM-based CDT detailed in Section VI-B. The resulting algorithm, namely the Hybrid Tiny $k$-NN, is shown in Algorithm 5. Here, the inputs and the initialization are the same as active tiny $k$-NN. The only difference resides in the fact that the hybrid tiny $k$-NN does not allocate a history window, but it relies on the training set $T$ (whose size is bounded by $\sigma$) as history window.

Similar to the algorithms it derives from, the hybrid tiny $k$-NN predicts the label $\hat{y}_t = D(x_t)$ and, when the supervised information $y_t$ is made available, it carries out both a passive (Algorithm 5, Lines 6–6) and an active update (Algorithm 5, Lines 12–18), for each sample $(x_t, y_t)$ generated by $P$ at instant $t$. Although the passive update is equal to that of the “CIT” algorithm, the active one requires to take into account the effects of the passive updates, which are supposed to increase the classification capability of the algorithm over time (until the accuracy of hybrid tiny $k$-NN reaches its maximum value). Consequently, the CUSUM CDT is slightly modified in its set $T_1$, which contains only values that are smaller than $v_0$, i.e., the accuracy estimated on an initial window of data. In this way, the hybrid update does not detect as concept drift the increases in the accuracy brought by the passive update (hence focusing on changes inducing a drop in the accuracy). Moreover, the adaptation phase triggered by the CDT involves directly the knowledge set $T$ of the classifier $K$, where samples older than the estimated time of change $t_r$ are discarded (Algorithm 5, Lines 15–18).

Summing up, the hybrid update continuously adapts $T$ over time thanks to the passive adaptation, hence avoiding the risk of non-detecting changes due to false-negative detections of the CDT. At the same time, the active adaptation present in the hybrid update can quickly discard obsolete knowledge when a change is detected and set a bound on the memory footprint of $T$.

VII. EXPERIMENTAL RESULTS

The proposed solutions have been validated in two different application scenarios (described in Section VII-A), two types of concept drift (defined in Section VII-B) and three different MCUs from STMicroelectronics (whose technological details
are given in Section VII-G). In addition, the rest of the Section is organized as follows. Section VII-C discusses the experimental settings, whereas Sections VII-D and VII-E provides the experimental results. Finally, Section VII-G presents the porting of the Hybrid Tiny \( k \)-NN algorithm on the three considered MCUs.

It is crucial to point out that TML in presence of concept drift is a completely new research area and, to the best of our knowledge, this is the first work in the related literature proposing adaptive mechanisms for TML running on MCUs.

### A. Application Scenarios and Datasets

In the experimental section, the following two application scenarios have been considered as follows.

1) The **speech-command identification** scenario whose goal is to correctly recognize a user-speech command present in a one-second-long audio clip. For this purpose, the *synthetic speech commands dataset* \([59], [60]\) has been considered. This dataset comprises 30 classes of commands, corresponding, for example, to “up,” “left,” “yes,” “go,” or a number from “zero” to “nine.” Moreover, the audio files within the dataset comprise different kinds of voices as well as different types of noisy classes.

2) The **image classification** scenario whose goal is to classify an image containing exactly one object. The well-known ImageNet \([61]\) dataset, comprising 1000 classes, has been considered.

### B. Considered Concept Drift Affecting \( \mathcal{P} \)

Two different kinds of concept drift affecting the data-generation process \( \mathcal{P} \) have been considered as follows.

1) The addition of noise on \( x \). This type of concept drift models the scenario where a failure on the microphone acquiring the audio clip occurs. The added noise includes 1) the distortion of the sound speed; 2) the introduction of reverberation; and 3) the introduction of up to three echoes. The noise addition affecting \( x \) has been modeled in two configurations: 1) abrupt and 2) gradual drift.

2) A change in the classification problem, i.e., a variation in the set of classes \( \Delta \) \([34]\). This concept drift models, for example, the change of user interests in a recommender system or the set of available commands in a vocal assistant.

### C. Experimental Settings

In this experimental analysis, the considered feature extractor \( q \) refers to the first layer of the well-known ResNet-18 CNN \([62]\). This layer comprises a convolutional layer with \( 64 \times 7 \times 7 \) 3-D filters with stride 2, a batch-normalization layer, a ReLU non-linearity, and a \( 3 \times 3 \) max-pooling layer with stride 2. The dimensionality reduction operator \( \varsigma \) discards 63 out of 64 filters by keeping only the one with the highest mean activation on the ImageNet benchmark. Consequently, the resulting DL model \( \varsigma \circ q \) is a single \( 7 \times 7 \times 3 \) filter that has 147 parameters and occupies 588B with a 32-bit floating-point representation.

In the **speech-command identification** scenario, the audio waveform (sampled at \( f_a = 22050 \) Hz) is converted into a spectrogram through a short-time Fourier transform with window size \( n_m = 512 \) and a step \( h_1 = 512 \) and then converted into a colored one by means of a colormap. In the **image classification**, instead, the images are resized to \( 224 \times 224 \times 3 \) before being passed as input to \( \mathcal{D} \). The resulting 1-s audio has a memory footprint of 88 200B, the image \( 602112B \), whereas the resulting colored spectrogram of size \( 257 \times 44 \times 3 \) requires 135 696B.

The change always starts after half of the available data, i.e., 500 samples per class in the **image classification scenario** and 800 in the **speech command identification** one. In particular, the gradual drift lasts 500 samples. Finally, 100 samples per class are provided to all the algorithm as initial training set \( \mathcal{T} \), i.e., \( |\mathcal{T}| = 100 \cdot \delta \). Experimental results are averaged over twenty runs and the value of \( \delta \) ranges from 2 to 5, with the classes randomly sub-sampled from the original dataset.

### D. Evaluating Effects of Preprocessing Through Condensing

The first aspect considered in this experimental analysis aims at studying the impact of condensing the \( k \)-NN \( k \)-NN training set \( \mathcal{T} \) with Algorithm 2. To achieve this goal, the proposed TML-CD \( \mathcal{D} \) is configured with the block \( \varsigma \circ q \) presented in Section VII-C and an initial training set \( \mathcal{T} \) with size \( |\mathcal{T}| = 100 \cdot \delta \). Then, the classification capabilities of \( \mathcal{D} \) when condensing is employed are evaluated in stationary conditions, i.e., no adaptation is carried out during the operational life of \( \mathcal{D} \). In addition to \( \mathcal{D} \) without or with the initial condensing (referred to as \( k \)-NN and \( k \)-NN + C in the following, respectively), the comparison comprises two well-known classifiers, i.e., a support vector machine (SVM) and a single fully-connected layer neural network classifier (NN–FC1). Both the classifiers are applied on the same features of the \( k \)-NN \( k \)-NN, i.e., those extracted by \( \varsigma \circ q \) (on the initial training set \( \mathcal{T} \)). Moreover, the SVM is trained until convergence, whereas the NN–FC1 is trained for 3 epochs with stochastic gradient descent, no momentum, and a learning rate \( \eta \in [1e^{-2}, 5e^{-3}, 1e^{-3}, \ldots, 1e^{-5}] \). In our experiments, only the best performing NN–FC1 classifier is shown. We emphasize that both the SVM and the NN are characterized by an unfeasible training procedure in MCUs, so they cannot be considered for the on-device training phase.

Table I shows the result in the proposed application scenarios with a different number of classes. The SVM and the NN–FC1 classifiers present the highest and worst accuracy in all the considered application scenarios, respectively. The proposed TML solution \( \mathcal{D} \) (without condensing) shows accuracies smaller than the SVM classifier by 4%–8% in **speech command identification** and 7%–13% in **image classification** scenarios. As expected, condensing the training set \( \mathcal{T} \) has a limited impact on the accuracy (at most 1% drop in **speech command identification** scenario), but it allows to reduce the memory requirements significantly. Without considering the NN–FC1 classifier, \( \mathcal{D} \) with condensing is the algorithm...
TABLE I

| Algorithm       | Speech Command Identification | Image Classification |
|-----------------|-------------------------------|----------------------|
|                 | \(\alpha_2\) \(m_2\) \(\alpha_3\) \(m_3\) \(\alpha_5\) \(m_5\) | \(\alpha_2\) \(m_2\) \(\alpha_3\) \(m_3\) \(\alpha_5\) \(m_5\) |
| SVM             | 0.93±0.05 138±17 0.88±0.04 244±16 0.81±0.05 447±18 | 0.73±0.07 188±9 0.61±0.05 292±6 0.46±0.05 496±4 |
| NN-FC1          | 0.71±0.14 2 0.75±0.06 3 0.49±0.12 5 | 0.60±0.08 2 0.45±0.05 3 0.29±0.05 5 |
| kNN             | 0.89±0.06 200 0.82±0.06 300 0.74±0.07 500 | 0.66±0.07 200 0.49±0.07 300 0.33±0.06 500 |
| kNN + C         | 0.88±0.06 120.2 0.81±0.04 128.2 0.73±0.06 255.3 33 | 0.66±0.07 115±18 0.51±0.06 214±19 0.35±0.04 414±16 |

Fig. 2. Mean accuracy and the number of samples to be kept in \(K\)’s training set \(\mathcal{T}\) over time of the three proposed adaptive algorithms. The plots represent the mean ± standard deviation accuracy over 20 experiments with \(\delta = \{2, 3, 5\}\) classes and the considered scenarios/concept drift. The algorithms receive as input a training set \(\mathcal{T}\), with |\(\mathcal{T}\)| = 100 · \(\delta\). 2-class speech command identification with (a) gradual noise and (b) abrupt noise. (c) 2-class image classification where a class changes. (d) 3-class speech command identification with gradual noise. (e) 5-class speech command identification with abrupt noise.

providing the lowest memory demand.\(^6\) In the speech command identification scenario, the number of stored samples indeed ranges from 32% to 50% of the provided samples (with \(\delta\) from 2 to 5), representing the 46%–57% of the ones required by the SVM, i.e., its support vectors. In the image classification scenario, the memory saving is significantly lower, with the SVM retaining almost all the samples as support vectors and the condensed \(\mathcal{D}\) storing 60%–82% of them. From now on, the proposed TML-CD \(\mathcal{D}\) is assumed to always rely on condensing algorithm in the configuration and testing stages (when available).

\(^6\)The NN-FC1 memory footprint corresponds to that of its weights, which are equal to the number of classes multiplied by the size of classifier inputs. Since the input size is the same for all the classifiers, the NN-FC1 memory is that of \(\delta\) samples.

E. Experimental Results in Presence of Concept Drift

Fig. 2 compares the three proposed adaptive algorithms, i.e., CIT, active tiny \(k\)-NN, and hybrid tiny \(k\)-NN, in the two considered scenarios with different numbers of classes \(\delta\). We considered two different figures of merit: 1) the mean ± std accuracy (the curve of each experiment is the convolution of the correct predictions of each experiment with a 100-D filter with all values of 0.01) and 2) memory footprint, measured as the number of samples within the training set \(\mathcal{T}\), over all the experiments. It is crucial to point out that the memory footprint does not include any other auxiliary source of memory, e.g., the history window \(W\) of the active tiny-\(k\)-NN algorithm (that has a size of \(\sigma = 100 \cdot \delta\)).

In more detail, Fig. 2 shows the accuracy and memory footprint in five different configurations: the speech command
TABLE II
MEAN ± STD ACCURACY OVER 20 EXPERIMENTS OF THE CIT AND THE HYBRID TINY k-NN ALGORITHMS COMPARED WITH TWO STATE OF THE ART ONES. THE EXPERIMENTS HAVE $\delta = \{2, 3, 5\}$ CLASSES IN THE CONSIDERED SCENARIO/CONCEPT DRIFT. ALL THE ALGORITHMS ARE CONSTRAINED BY $|T| = 50$

| Scenario | CIT | Hybrid Tiny k-NN | ADWIN k-NN | SAM k-NN |
|----------|-----|------------------|------------|----------|
| Image, $\delta = 2$, One-Class Change | 0.624 ± 0.052 | 0.624 ± 0.052 | 0.629 ± 0.055 | 0.620 ± 0.055 |
| Image, $\delta = 3$, One-Class Change | 0.456 ± 0.040 | 0.456 ± 0.040 | 0.459 ± 0.045 | 0.448 ± 0.043 |
| Image, $\delta = 5$, One-Class Change | 0.292 ± 0.030 | 0.292 ± 0.030 | 0.294 ± 0.032 | 0.284 ± 0.029 |
| Speech, $\delta = 2$, Gradual Noise | 0.867 ± 0.057 | 0.868 ± 0.057 | 0.849 ± 0.051 | 0.825 ± 0.055 |
| Speech, $\delta = 3$, Gradual Noise | 0.756 ± 0.070 | 0.756 ± 0.070 | 0.728 ± 0.061 | 0.688 ± 0.061 |
| Speech, $\delta = 5$, Gradual Noise | 0.597 ± 0.050 | 0.597 ± 0.050 | 0.568 ± 0.043 | 0.521 ± 0.039 |
| Speech, $\delta = 2$, Abrupt Noise | 0.884 ± 0.055 | 0.885 ± 0.055 | 0.858 ± 0.051 | 0.834 ± 0.055 |
| Speech, $\delta = 3$, Abrupt Noise | 0.771 ± 0.071 | 0.771 ± 0.071 | 0.737 ± 0.062 | 0.697 ± 0.065 |
| Speech, $\delta = 5$, Abrupt Noise | 0.611 ± 0.052 | 0.611 ± 0.052 | 0.581 ± 0.044 | 0.533 ± 0.040 |
| Speech, $\delta = 2$, One-Class Change | 0.846 ± 0.058 | 0.846 ± 0.058 | 0.823 ± 0.044 | 0.794 ± 0.045 |
| Speech, $\delta = 3$, One-Class Change | 0.704 ± 0.067 | 0.704 ± 0.067 | 0.674 ± 0.060 | 0.633 ± 0.056 |
| Speech, $\delta = 5$, One-Class Change | 0.529 ± 0.040 | 0.529 ± 0.040 | 0.508 ± 0.037 | 0.463 ± 0.032 |

Fig. 3. Mean accuracy over time of the three proposed adaptive algorithms compared with two state of the art ones. The plots represent the mean ± standard deviation over 20 experiments with $\delta = 2$ classes in the speech command identification scenario with the introduction of noise. All the algorithms are constrained by $|T| = 50$. (a) Gradual noise affects data after half samples in 500 steps. (b) Abrupt noise affects data after half samples.

identification scenario where either gradual or abrupt noise affects data with $\delta = \{2, 3, 5\}$ [Fig. 2(a), (b), (d), and (e)]; and the image classification scenario where one-class changes with $\delta = 2$ [Fig. 2(c)]. The proposed algorithms work as expected. On the one hand, the (passive) CIT algorithm continuously improves over time, at the expense of an unbounded memory growth (in these experiments, none of the approaches detailed in Section VI-A to control it has been considered). Moreover, in all the considered scenarios, the slope of the samples’ curve increases at the change time, highlighting the accuracy drop due to the change itself. On the other hand, the active $k$-NN algorithm is able to recover after a change keeping its memory footprint nearly constant and significantly lower than the size of history window $W$ (not shown in the Fig. 2) due to condensing. Finally, the hybrid tiny $k$-NN algorithm combines the advantages of both the CIT and the active tiny $k$-NN. It can recover faster than the two other algorithms in almost all the considered scenarios and keep the memory footprint under control (by taking into account also the active tiny $k$-NN history window memory footprint, the hybrid tiny $k$-NN has the lowest footprint). Moreover, it shows the best accuracy, being overcome by CIT only when it saturates the maximum size of its training set $T$ that is here fixed to $|\bar{T}| = 100 - \delta$. This effect is visible in particular in Fig. 2(d) and (e).

It is worth noting that the proposed solutions is effective even in the case of gradual drift as shown in Fig. 2(a). Here, results show that the hybrid tiny $k$-NN and the passive CIT algorithm are faster than the active tiny $k$-NN due to the passive updates that continuously react to the change, whereas the active update requires 50 samples to define a binomial window and inspect for a concept drift. Moreover, all three algorithms seem to react more than once to the gradual drift.

F. Comparison With State-of-the-Art Adaptive $k$-NN

This section aims at comparing the proposed three adaptation mechanisms with two state-of-the-art solutions present in the related literature for adaptive $k$-NN classifiers in presence of concept drift as follows.

1) The $k$-NN with ADWIN [31] continuously adds the novel supervised samples to its training set and relies on the ADWIN CDT [63] to decide which samples to forget or to maintain.

2) The SAM $k$-NN classifier [32] combines a short-term and long-term memory, the first one containing the last samples, i.e., the newest knowledge, the latter keeping consolidated knowledge over time.

Authorized licensed use limited to the terms of the applicable license agreement with IEEE. Restrictions apply.
We compare these two state-of-the-art solutions with the two of the proposed adaptation mechanisms, i.e., passive and hybrid. For this experimental campaign, the maximum memory size has been fixed to $|\mathcal{T}| = 50$, to both allow a fair comparison among the considered solutions and consider them in the real technological scenario then detailed in Section VII-G.

Table II summarizes the results for the considered application scenarios (i.e., speech-command identification and image classification, see Section VII-A) and the employed concept drift (i.e., gradual and drift addition of noise and class change, see Section VII-B). In more detail, Table II shows the mean accuracy computed over the test set. The results are particularly interesting, showing that both the Hybrid Tiny $k$-NN and the passive CIT algorithms overcome the SAM $k$-NN solution in all 12 cases. The ADWIN $k$-NN is the best solution in the image classification scenario. In addition, the proposed CIT and Hybrid solution overcome ADWIN in the speech command identification for all the three considered types of concept drift.

Fig. 3 shows the accuracy over time in the speech command identification scenario with gradual and abrupt noise addition. Interestingly, in the latter case, which is shown in Fig. 3(b), both the hybrid tiny $k$-NN and the CIT algorithms have higher accuracy than the two state-of-art solutions, ADWIN and SAM $k$-NNs. In particular, after the change, they both provide the fastest adaptation abilities (in terms of the ability to recover the accuracy after the drop induced by the abrupt change). The results are similar with gradual concept drift, both before and after the drift, with hybrid tiny $k$-NN and CIT overcoming the state-of-the-art solutions. However, during the gradual drift, as shown in Fig. 3(a), both the ADWIN $k$-NN and the SAM $k$-NN exhibit a higher accuracy with respect to all the proposed solutions, i.e., they are subject to a smaller accuracy drop due to the drift.

Fig. 3 also shows the results with the active tiny $k$-NN algorithm that is constrained by $|\mathcal{T}| = 50$ only on the history window size and thus cannot be considered in the porting scenario. In both the scenarios, the active tiny $k$-NN is characterized by a relevant drop in accuracy due to the concept drift and by a slow adaptation ability, since it detects the concept drift but has troubles in defining a suitable representation of the novel concept that results in multiple detections and adaptations before defining an effective one (this behavior is highlighted by the multiple changes of direction in the accuracy plot).

### G. Porting the Hybrid Tiny $k$-NN on the STM32 MCUs

The aim of this section is to show the technological feasibility of the proposed hybrid tiny $k$-NN algorithm in the speech command identification scenario. To achieve this goal, we considered the following three different MCUs as follows.

1) The STM32H743 is a high-performance MCU having a 480 MHz Cortex-M7 processor, 1024 KB of RAM (split into five blocks of different speed), and 2048 KB of flash memory.

2) The STM32F767 is a high-performance MCU having a 216 MHz Cortex-M7 processor, 512 KB of RAM, and 2048 KB of flash memory.

3) The STM32F401 is a general-purpose MCU having a 84 MHz Cortex-M4 processor, 96 KB of RAM, and 512 KB of flash memory.

The main technological constraint imposed by such board is the one on the memory, i.e., the maximum memory footprint of the hybrid $k$-NN algorithm cannot overcome the available RAM of each MCU (in the case of STM32H743 that limit is lowered to 512KB, i.e., the size of the fastest RAM block). To satisfy this memory constraint, we set the maximum training set size of the hybrid tiny $k$-NN algorithm to $|\mathcal{T}| = 50$.

In addition, for the STM32F401 board, the sampling frequency $f_a$ is reduced from $f_a = 22050$ Hz to $f_a = 4410$ Hz as suggested in [64]. This guarantees a strong reduction in the memory footprint of the input audio (17640B), the generated spectrogram with windows of size $n_{win} = 128$ and step $h_i = 128$ ($65 \times 35 \times 3$ occupying 27300B), and on the output of the feature extractor ($17 \times 9$ occupying 588B).

Table III details the memory footprint of the hybrid tiny $k$-NN deployed for the STM32H743 and STM32F401 [Table III(a)] and the STM32F767 [Table III(b)].

As shown in Section VII-F along with Fig. 3 and Table II, the hybrid Tiny $k$-NN provides the higher accuracy in almost all the considered scenarios and types of concept drift when $|\mathcal{T}| \leq 50$. Moreover, with respect to the CIT algorithm that has similar accuracies, it keeps a smaller amount of samples within $\mathcal{T}$ (CIT has always $|\mathcal{T}| = 50$). For both these reasons, it has been employed as the target solution for the considered tiny device.

Table IV reports the experimental execution times of the $D$ blocks in the considered MCUs. More in detail, the measured quantities are: the processing time $t_p$ needed to transform the acquired 1 s audio into a spectrogram, the feature extractor and dimensionality reduction blocks’ execution $t_{\text{ext}}$, and the $k$-NN $K$ prediction time $t_{\text{pred}}$. The $k$-NN $K$ prediction time $t_{\text{pred}}$. The $k$-NN $K$ prediction time $t_{\text{pred}}$. The $k$-NN $K$ prediction time $t_{\text{pred}}$.

### Table III

| (a) Size | Memory (B) |
|-----------------|-----------|
| Audio ($t_a = 1 s, f_a = 22050$ Hz) | 1x22050 | 88200 |
| Spectrogram ($n_{win} = h_i = 512$) | 257x46x3 | 135696 |
| $\zeta \circ \phi$ (1 convolutional filter 7x7x3) | 7x7x3 | 588 |
| $\zeta \circ \phi$ output | 65x11 | 2860 |
| $K$’s Training Set $\mathcal{T}$ | 50 | 145400 |
| Total | | 370744 |

| (b) Size | Memory (B) |
|-----------------|-----------|
| Audio ($t_a = 1 s, f_a = 4410$ Hz) | 1x4410 | 17640 |
| Spectrogram ($n_{win} = h_i = 128$) | 64x33x3 | 27300 |
| $\zeta \circ \phi$ (1 convolutional filter 7x7x3) | 7x7x3 | 588 |
| $\zeta \circ \phi$ output | 1x7x9 | 612 |
| $K$’s Training Set $\mathcal{T}$ | 50 | 31000 |
| Total | | 77140 |
of its training set, 10 and 50 (the latter also shows the worst measured prediction time when an adaptation has been made). Results are particularly interesting. In particular, the processing and the feature extraction are the two predominant times, requiring 41.5 and 95.3 ms on a high-performance MCUs (the STM32H7 and the STM32F7) and 77.7 ms on a general-purpose one (the STM32F4, although on a smaller spectromgram). The K’s prediction and the adaptation, when employed, are negligible with respect to the other times, being the 7 and the 15% on the STM32H7, the 4% and the 13% on the STM32F7, and the 6 and the 175% on the STM32F4 (adaptation is the only exception). As a final remark, the total time required from the processing of the acquired audio to the final prediction, including the possible adaptations, is significantly lower than that of the acquisition, showing the effectiveness of the proposed Hybrid Tiny \(k\)-NN algorithm on three real MCUs.

### VIII. CONCLUSION

For the first time in the literature, this article introduced an adaptive TML solution for concept drift. This solution, characterized by a hybrid approach integrating an active and a passive adaptation step, takes into account the technological constraints on memory, computation, and energy typically characterizing embedded systems and the IoT units it runs on. The proposed solution has been deployed to three different MCUs with 96 to 512KB of RAM, showing its feasibility in real-world scenarios and on off-the-shelf technological units.

Future work will encompass the definition of advanced memory control mechanisms on passive updates (e.g., by deepening the suggested probabilistic approach), further optimization of the \(k\)-NN memory requirements, the definition of learning mechanisms at the feature extractor block, and the exploration of sparse or quantized solutions for the TML algorithms.

### REFERENCES

[1] C. Alippi, S. Disabato, and M. Roveri, “Modeling convolutional neural networks to embedded systems: The AlexNet and VGG-16 case,” in Proc. 17th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN), Apr. 2018, pp. 212–223.

[2] R. Sanchez-Iborra and A. F. Skarmeta, “TinyML-enabled frugal smart objects: Challenges and opportunities,” IEEE Circuits Syst. Mag., vol. 20, no. 3, pp. 4–18, 3rd Quart., 2020.

[3] J. Tang, D. Sun, S. Liu, and J.-L. Gaudiot, “Enabling deep learning on IoT devices,” Computer, vol. 50, no. 10, pp. 92–96, 2017.

[4] C. R. Banbury et al., “Benchmarking TinyML systems: Challenges and direction,” 2020, arXiv:2003.04821.

[5] H. Cai, C. Gan, L. Zhu, and S. Han, “TinyTL: Reduce memory, not parameters for efficient on-device learning,” in Proc. Adv. Neural Inf. Process. Syst., vol. 33, 2020, pp. 1–13.

[6] S. Disabato and M. Roveri, “Incremental on-device tiny machine learning,” in Proc. 2nd Int. Workshop Challenges Artif. Intell. Mach. Learn. Internet Things, Nov. 2020, pp. 7–13.

[7] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, “Learning in nonstationary environments: A survey,” IEEE Comput. Intell. Mag., vol. 10, no. 4, pp. 12–25, Apr. 2015.

[8] J. Gama, I. Zhlobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia, “A survey on concept drift adaptation,” ACM Comput. Surv., vol. 46, no. 4, Apr. 2014.

[9] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, “Learning under concept drift: A review,” IEEE Trans. Knowl. Data Eng., vol. 31, no. 12, pp. 2346–2363, Dec. 2019.

[10] R. Elwell and R. Polikar, “Incremental learning of concept drift in nonstationary environments,” IEEE Trans. Neural Netw., vol. 22, no. 10, pp. 1517–1531, Oct. 2011.

[11] B. Krawczyk and M. Woźniak, “One-class classifiers with incremental learning and forgetting for data streams with concept drift,” Soft Comput., vol. 19, no. 12, pp. 3387–3400, 2015.

[12] G. Hulten, L. Spencer, and P. Domingos, “Mining time-changing data streams,” in Proc. 7th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD), 2001, pp. 97–106.

[13] W. N. Street and Y. Kim, “A self-training ensemble algorithm (SEA) for large-scale classification,” in Proc. 7th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD), 2001, pp. 377–382.

[14] P. R. L. Almeida, L. S. Oliveira, A. S. Britto, Jr., and R. Sabourin, “Adapting dynamic classifier selection for concept drift,” Expert Syst. Appl., vol. 104, pp. 67–85, Aug. 2018.

[15] L. Pietruszczuk, L. Rutkowski, M. Jaworski, and P. Duda, “How to adjust an ensemble size in stream data mining?” Inf. Sci., vol. 381, pp. 46–54, Mar. 2017.

[16] H. Li, P. Barnaghi, S. Enshaeefar, and F. Ganz, “Continual learning using Bayesian neural networks,” IEEE Trans. Neural Netw. Learn. Syst., vol. 32, no. 9, pp. 4243–4252, Sep. 2021.

[17] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, “Continual lifelong learning with neural networks: A review,” Neural Netw., vol. 113, pp. 54–71, May 2019.

[18] B. Pérez-Sánchez, O. Fontenla-Romero, and B. Guijarro-Berthelàns, “A review of adaptive online learning for artificial neural networks,” Artif. Intell. Rev., vol. 49, no. 2, pp. 281–299, Feb. 2018.

[19] G. Ditzler and R. Polikar, “Hellingor distance based drift detection for nonstationary environments,” in Proc. IEEE Symp. Comput. Intell. Dyn. Uncertain Environments (CIDUE), Apr. 2011, pp. 41–48.

[20] T. Dasu, S. Krishnan, S. Venkatasubramanian, and K. Yi, “An information-theoretic approach to detecting changes in multi-dimensional data streams,” in Proc. Symp. Interface Statist., Comput. Sci. Appl., 2006, pp. 1–24.

[21] L. Bu, C. Alippi, and D. Zhao, “A pdf-free change detection test based on density difference estimation,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 2, pp. 324–334, Feb. 2018.

[22] P. Duda, L. Rutkowski, M. Jaworski, and D. Rutkowska, “On the Parzen kernel-based probability density function learning procedures over time-varying streaming data with applications to pattern classification,” IEEE Trans. Cybern., vol. 50, no. 4, pp. 1683–1696, Apr. 2020.

[23] M. Baena-García, J. del Campo-Avila, R. Fidalgo, A. Bifet, R. Gavalda, and R. Morales-Bueno, “Early drift detection method,” in Proc. 4th Int. Workshop Knowl. Discovery Data Streams, vol. 6, 2006, pp. 77–86.

[24] A. R. Kamat, “Continuous inspection schemes,” Bionetrika, vol. 41, no. 1–2, pp. 100–115, Apr. 1954.

[25] X. Wang, Q. Kang, M. Zhou, L. Pan, and A. Abusorrah, “Multiscale drift detection test to enable fast learning in nonstationary environments,” IEEE Trans. Cybern., vol. 51, no. 7, pp. 3483–3495, Jul. 2021.

[26] D. Zambon, C. Alippi, and L. Livi, “Concept drift and anomaly detection in graph streams,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 11, pp. 5592–5605, Nov. 2018.

[27] C. Alippi, G. Boracchi, and L. Livi, “Concept drift and anomaly detection,” IEEE Trans. Computer, vol. 51, no. 7, pp. 3483–3495, Jul. 2021.

[28] Z. V. S. Dambom, C. Alippi, and L. Livi, “Concept drift and anomaly detection in graph streams,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 11, pp. 5592–5605, Nov. 2018.

[29] C. Alippi, G. Boracchi, and L. Livi, “Concept drift and anomaly detection,” IEEE Trans. Computer, vol. 51, no. 7, pp. 3483–3495, Jul. 2021.

[30] X. Wang, Q. Kang, M. Zhou, L. Pan, and A. Abusorrah, “Multiscale drift detection test to enable fast learning in nonstationary environments,” IEEE Trans. Cybern., vol. 51, no. 7, pp. 3483–3495, Jul. 2021.

[31] D. Zambon, C. Alippi, and L. Livi, “Concept drift and anomaly detection in graph streams,” IEEE Trans. Neural Netw. Learn. Syst., vol. 29, no. 11, pp. 5592–5605, Nov. 2018.

[32] C. Alippi, G. Boracchi, and L. Livi, “Concept drift and anomaly detection,” IEEE Trans. Computer, vol. 51, no. 7, pp. 3483–3495, Jul. 2021.
A. Bifet, B. Pfahringer, J. Read, and G. Holmes, “Efficient data stream classification via probabilistic adaptive windows,” in Proc. 28th Ann. ACM Symp. Appl. Comput. (SAC), 2013, pp. 801–806.

V. Losing, B. Hammer, and H. Wersing, “KNN classifier with self adjusting memory for heterogeneous concept drift,” in Proc. IEEE 16th Int. Conf. Data Mining (ICDM), Dec. 2016, pp. 291–300.

S. Disabato and M. Roveri, “Learning convolutional neural networks in presence of concept drift,” in Proc. Int. Joint Conf. Neural Netw. (IJCNN), Jul. 2019, pp. 1–8.

V. Vaquet, F. Hinder, J. Vaquet, J. Brinkkolf, and B. Hammer, “Online learning on non-stationary data streams for image recognition using deep embeddings,” in Proc. IEEE Int. Symp. Ser. Comput. Intell. (SSCI), Dec. 2021, pp. 1–7.

Z. Yang, S. Al-Dahidi, P. Baraldi, E. Zio, and L. Montelatici, “A novel concept drift detection method for incremental learning in nonstationary environments,” IEEE Trans. Neural Netw. Learn. Syst., vol. 31, no. 1, pp. 309–320, Jan. 2020.

Y. He, X. Zhang, and J. Sun, “Channel pruning for accelerating very deep neural networks,” in Proc. IEEE Int. Conf. Comput. Vis. (ICCV), Oct. 2017, pp. 1389–1397.

S. Lin, R. Ji, C. Chen, D. Tao, and J. Luo, “Holistic CNN compression via low-rank decomposition with knowledge transfer,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 41, no. 12, pp. 2889–2905, Dec. 2019.

A. Bulat and G. Tzimiropoulos, “Bit-mixer: Mixed-prediction networks with runtime bit-width selection,” 2021, arXiv:2103.17257.

S. Gupta, A. Agrowal, K. Gopalakrishnan, and P. Narayanan, “Deep learning with limited numerical precision,” in Proc. 32nd Int. Conf. Mach. Learn., 2015, pp. 1737–1746.

M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “XNOR-Net: ImageNet classification using binary convolutional neural networks,” in Proc. Eur. Conf. Comput. Vis. Cham, Switzerland: Springer, 2016, pp. 525–542.

C. Banbury et al., “MicroNets: Neural network architectures for deploying tinyml applications on commodity microcontrollers,” in Proc. Mach. Learn. Syst., vol. 3, 2021, pp. 517–532.

S. Gopinath, N. Ghanathe, V. Seshadri, and R. Sharma, “Compiling KB-sized machine learning models to tiny IoT devices,” in Proc. 40th ACM SIGPLAN Conf. Program. Lang. Design Implement., Jun. 2019, pp. 79–95.

A. Kumar, S. Goyal, and M. Varma, “Resource-efficient machine learning in 2 KB RAM for the Internet of Things,” in Proc. Int. Conf. Mach. Learn., 2017, pp. 1935–1944.

J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, “MCUNet: Tiny deep learning on IoT devices,” in Proc. Adv. Neural Inf. Process. Syst., vol. 33, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds. Red Hook, NY, USA: Curran Associates, 2020, pp. 11711–11722.

I. Fedorov et al., “TinyLSTMs: Efficient neural speech enhancement for hearing aids,” 2020, arXiv:2005.11138.

M. Kenzke, D. Klick, P. Kubik, A. Ali, J. Dell Missier, and V. Turau, “Artificial neural networks for sensor data classification on small embedded systems,” 2020, arXiv:2012.08403.

I. Fedorov, R. P. Adams, M. Mattina, and P. Whatmough, “SpArSe: Sparse architecture search for CNNs on resource-constrained microcontrollers,” in Proc. Adv. Neural Inf. Process. Syst., vol. 32, 2019, pp. 1–13.

M. Rusci, M. Fariselli, A. Capotondi, and L. Benini, “Leveraging automated mixed-precision quantization for tiny edge microcontrollers,” in IoT Streams for Data-Driven Predictive Maintenance and IoT Edge, and Mobile for Embedded Machine Learning, Cham, Switzerland: Springer, 2020, pp. 296–308.

H. Ren, D. Anicic, and T. Runkler, “TinyOL: TinyML with online-learning on microcontrollers,” 2021, arXiv:2103.08295.

N. S. Altman, “An introduction to kernel and nearest-neighbor non-parametric regression,” Amer. Statist., vol. 46, no. 3, pp. 175–185, 1992.

M. Roseberry and A. Cano, “Multi-label punitive kNN with self-adjusting memory for drifting data streams,” in 2nd Int. Workshop Learn. Imbalanced Domains: Theory Appl., 2018, pp. 23–37.

P. Hart, “The condensed nearest neighbor rule (Corresp.),” IEEE Trans. Inf. Theory, vol. IT-14, no. 3, pp. 515–516, May 1968.

I. Tomek, “Two modifications of CNN,” IEEE Trans. Syst., Man, Cybern., vol. SMC-6, no. 11, pp. 769–772, Nov. 1976.

J. Laurikkala, “Improving identification of difficult small classes by balancing class distribution,” in Proc. Conf. Artif. Intell. Med. Eur. Berlin, Germany: Springer, 2001, pp. 63–66.

I. Tomek, “An experiment with the edited nearest-neighbor rule,” IEEE Trans. Syst., Man, Cybern., vol. SMC-6, no. 11, pp. 448–452, Jun. 1976.

D. L. Wilson, “Asymptotic properties of nearest neighbor rules using edited data,” IEEE Trans. Syst., Man, Cybern., vol. SMC-2, no. 3, pp. 408–421, Jul. 1972.

M. R. Smith, T. Martinez, and C. Giraud-Carrier, “An instance level analysis of data complexity,” Mach. Learn., vol. 95, no. 2, pp. 225–256, 2014.

G. Lorden, “Procedures for reacting to a change in distribution,” Ann. Math. Stat., vol. 42, pp. 1897–1908, Dec. 1971.

J. Buchner. (2017). Synthetic Speech Commands: A Public Dataset for Single-Word Speech Recognition. [Online]. Available: https://www.kaggle.com/jbuchner/synthetic-speech-commands-dataset.

P. Warden, “Speech commands: A dataset for limited-vocabulary speech recognition,” 2018, arXiv:1804.03209.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2009, pp. 248–255.

K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2016, pp. 770–778.

A. Bifet and R. Gavalda, “Learning from time-changing data with adaptive windowing,” in Proc. SIAM Int. Conf. Data Mining, Apr. 2007, pp. 443–448.

S. Disabato, G. Canonaco, P. G. Flikkema, M. Roveri, and C. Alippi, “Birdsong detection at the edge with deep learning,” in Proc. IEEE Int. Conf. Smart Comput. (SMARTCOMP), Aug. 2021, pp. 9–16.

Simone Disabato received the master’s degree in computer science and engineering and the Ph.D. degree in information technologies from the Politecnico di Milano, Milan, Italy, in 2017 and 2022, respectively, with a thesis on adaptive machine/deep learning (DL) solutions able to operate in non-stationary environments and memory and computational constrained cyber-physical systems.

Manuel Roveri (Senior Member, IEEE) received the M.S. degree in computer science from the University of Illinois at Chicago, Chicago, IL, USA, in 2003, and the Dr.Eng. degree in computer science engineering, in 2003, and the Ph.D. degree in computer engineering from the Politecnico di Milano, Milan, Italy, in 2007. Currently, he is a Full Professor with the Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano. He holds one patent and has published more than 110 articles in international journals and conference proceedings. His research interests include embedded and edge AI, intelligent embedded and cyber-physical systems, and learning in presence of concept-drift and privacy-preserving machine and deep learning (DL). Dr. Roveri was a recipient of the 2018 IEEE Computational Intelligence Magazine “Outstanding Paper Award,” the 2016 IEEE Computational Intelligence Society “Outstanding Transactions on Neural Networks and Learning Systems Paper Award,” the Best Regular Paper Award at the INNS Conference on Big Data in 2016, and the 2021 “Outstanding Associate Editor” of the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE. He served as the chair and a member in several IEEE committees and subcommittees and as an Associated Editor for the IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS, IEEE TRANSACTIONS ON ARTIFICIAL INTELLIGENCE, IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE, IEEE Computational Intelligence Magazine, and Neural Networks (Elsevier).